Analysing Campylobacter time series in R

# Copyright and License

**Source:**

This case study was designed under a ECDC contract for “Expertise in biostatistics for analysis of surveillance data”. This tutorial was first designed by Esther Kissling, EpiConcept, April 2012 and was modified for R by Thomas Seyler, EpiConcept, July 2015. In 2018 the R version was re-worked by Alexander Spina (correction and updated); some parts of code were adapted from the EPIET time series analysis module case studies.

**Revisions:**

*If you modify this case study, please indicate below your name and changes you made*

**You are free:**

* **to Share** — to copy, distribute and transmit the work
* **to Remix** — to adapt the work

**Under the following conditions:**

* **Attribution** — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work). The best way to do this is to keep as it is the list of contributors: sources, authors and reviewers.
* **Share Alike** — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one. Your changes must be documented. Under that condition, you are allowed to add your name to the list of contributors.
* You cannot sell this work alone but you can use it as part of a teaching.

**With the understanding that:**

* **Waiver** — Any of the above conditions can be waived if you get permission from the copyright holder.
* **Public Domain** — Where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license.
* **Other Rights** — In no way are any of the following rights affected by the license:
  + Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations;
  + The author’s moral rights;
  + Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.
* **Notice** — For any reuse or distribution, you must make clear to others the license terms of this work by keeping together this work and the current license.

This licence is based on <http://creativecommons.org/licenses/by-sa/3.0/>

## Objectives of tutorial

At the end of the tutorial participants should be able to

* create a plan of analysis for simple time series analysis;
* use R for exploratory time series analysis including installing the software, importing a dataset, setting time units for aggregation of time series, data checking, trend and seasonality analysis as well as analysis of residuals;
* understand how to create simple aberration detection algorithms, taking outbreaks into account.

## Note on the dataset

The dataset is an original dataset on Campylobacter disease in a European country that has been modified for training purposes.

## Background to the tutorial

The objective of the tutorial is to carry out an exploratory time series analysis on Campylobacter data using the TSA tool. The dataset is called *campy.csv*.

The dataset gives the number of daily cases and contains a time, count and period variable. The table below shows the database dictionary:

|  |  |  |
| --- | --- | --- |
| Variable name | Variable type | Definition |
| time | Daily date | Date defined as ‘Date used for statistics’ |
| nb | Numeric | Number of reports |
| period | Numeric | The reporting period (e.g. 1=daily; 7=week; 30=month; 90=quarter; etc.) |

# Question 1

Open the dataset (*campy.csv*) in R and briefly explore it (make sure your working directory is set in R). Import the necessary packages and install the corresponding packages. With the assumption that the reporting in the dataset here is stable over time, create an outline of a mini-protocol for TSA, including hypotheses to be tested, a plan of analysis, with specifications for the dataset, inclusion/exclusion criteria, etc.

# Help Question 1

**For information:** Background on Campylobacteriosis

Campylobacteriosis is a diarrhoeal disease caused by Campylobacter bacteria, found in animals such as poultry, cattle, pigs, wild birds and wild mammals.

The most frequent way of getting infected is through the consumption of contaminated food (mainly poultry) or water. Other risk factors include swimming in natural surface-waters and direct contact with infected animals.

After an incubation period 2-5 days (range 1-10 days) common symptoms are severe abdominal pain, watery and /or bloody diarrhoea and fever. Usually, symptoms last for a few days and the disease is self-limiting but occasionally they will persist and result in hospitalisation. Antimicrobial therapy is seldom needed.

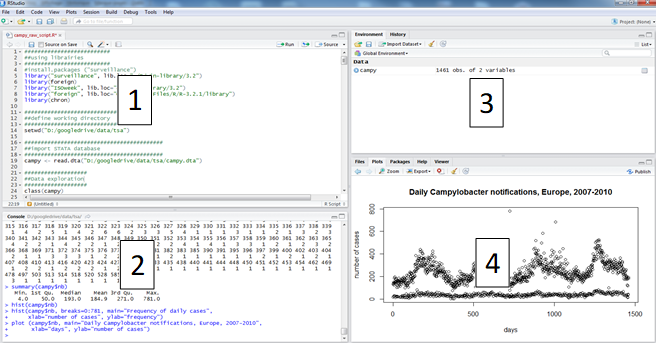
Campylobacter infection has been associated with complications such as later joint inflammation (5-10% of cases) and, on rare occasions, Guillian-Barre syndrome (a temporary but severe paralysis that may be total).

As prophylactic measure, control of Campylobacter colonisation in poultry is important, as well as hygienic processing of meat, and the protection and control of private drinking water supplies.

## R-Studio environment

*R-Studio* is an integrated development environment (IDE) for *R*. It includes a console, syntax highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management. You can download it freely from the [*R-Studio* website](https://www.rstudio.com/).

The *RStudio* window comprises the following sections:

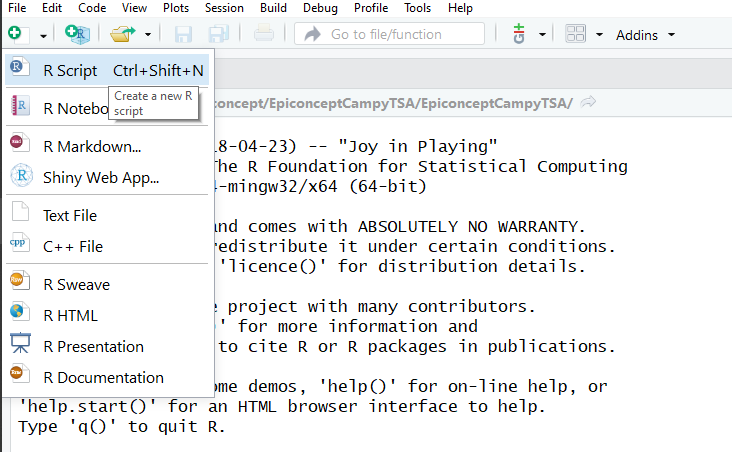


1. A script which is equivalent to a do file in STATA or pgn in EpiInfo.
2. A console where you can write commands and execute them, and where results other than graphs are shown.
3. A “Environment” and “History” windows where you can find the list of R objects being stored and the log of your session, respectively.
4. The “Plots”, “Packages” and “Help” windows, for graphic visualisation, selection of installed packages and help documentation, respectively.

## R-scripts

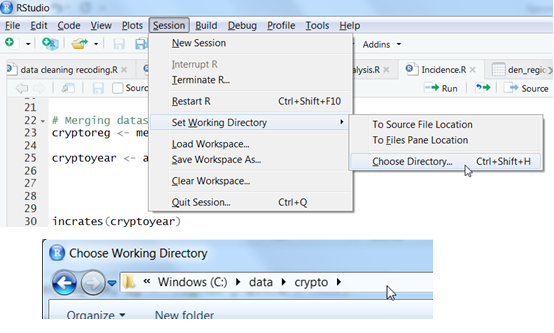
Throughout this case study we strongly encourage you to use *R-scripts* to keep track of the commands you are using to follow your plan of analysis. It will contain data management steps as well as descriptive and statistical analysis.

Open R studio and open a new *R-script*. To open a new *R-script*, click on the plus button on the top left (below File), then select *R-script* from the drop-down menu.



## Working directory

Make sure you have created a working directory on your computer. This is where you store your data and your R scripts and outputs. An example of working directory could be “C:/data/crypto” if you are using a PC. To set your working directory you can use the menu function by clicking “Session” > “Set Working Directory” > “Choose Directory…” and then you can select the appropriate folder.



You could alternatively do this by adding code to your *R-script* (or the command line).

You can check the path for your current working directory using the *getwd* function.

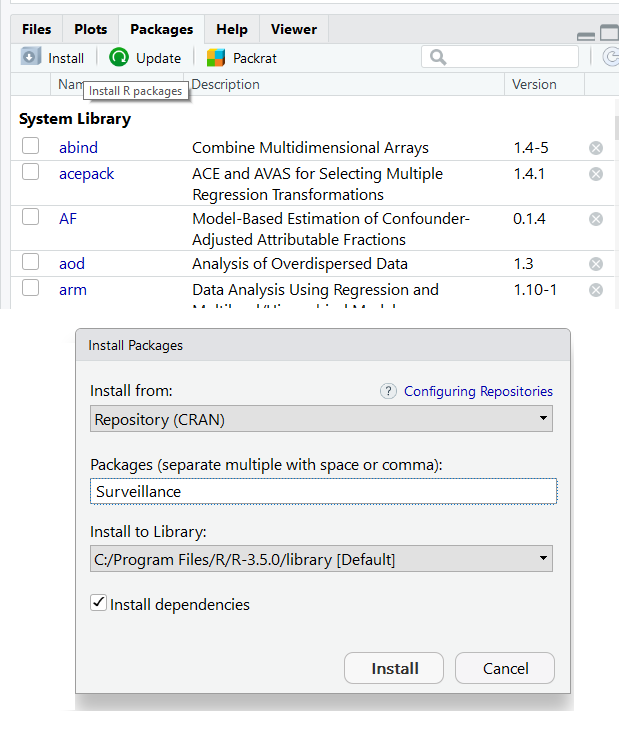
#Check your current working directory  
getwd()

To set your working directory you can use the *setwd* function.

#set working directory   
 #note the forward rather than backwards slashes  
setwd("C:/data/crypto")

## Packages

R comes with a set of default functions. However one of the strengths of R is that many user-written functions exist to help you do your work. To use them, you first need to install the package they belong to. To do this you can click on “Install” on the bottom right in the “Packages” window.



We will mainly use packages which come ready installed with R (base code), but where it makes things easier we will use add-on packages. All the R packages you need for the exercises can be installed over the Internet.

You will need to install these before starting. We will be using the following packages.

* *ISOweek*: for adjusting dates and switching between units
* *surveillance*: for fitting time series and detection algorithms
* *zoo*: for producing rolling statistics
* *moments*: for looking at skewness and kurtosis of time series

You can also install packages using the *install.packages* function, as below. Do not try to understand this code.

#choose packages required   
required\_packages <- c("ISOweek", "surveillance", "zoo", "moments")

# Installing required packages for this case study  
for (pkg in required\_packages) {  
 if (!pkg %in% rownames(installed.packages())) {  
 install.packages(pkg, dependencies = TRUE)  
 }  
}

You only need to install packages once. However you will need to load them each time you start a new session using the *library* function.

library(ISOweek)

Here too, you can load multiple packages at the same time within a loop; again do not try to understand this just yet, just appreciate it is possible.

# Loading required packages for this case study  
for (i in seq(along = required\_packages)) {  
 library(required\_packages[i], character.only = TRUE)  
}

## Reading in data

Import the dataset from a comma separated value (.csv) file using the *read.csv* function, storing it as a data frame within *R* called campy. The arrow is used to assign the data frame to campy. Here we also specify that we do not want to read in string (character variables) as factors.

campy <- read.csv("campy.csv", stringsAsFactors = FALSE)

## Variable types

Campy is now a *R* object, of the data frame type. A data frame is simply the how datasets are classified in *R*, thus given class “data.frame”. You can check the type of object by typing: class(campy)

Individual variables from the data frame are called using the name of the data frame followed by the **$** sign and the name of the variable. For instance, if you wish to enquire about the type of variables:

#check class of variable time  
class(campy$time);  
  
#check class of variable count  
class(campy$count)

You will notice that the time variable is of class “character”. However we want this as a date. You can change this by overwriting the variable using the *as.Date* function.

#overwrite time variable as a date   
campy$time <- as.Date(campy$time)

# Browse data

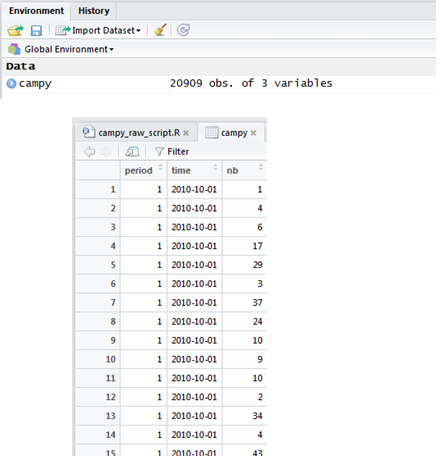
To browse the first six records you can use the *head* function.

#see first six rows of data   
head(campy)

To get summary statistics of all variables included in the data frame use the *summary* function.

#see summary statistics of variables  
summary(campy)

Alternatively you can view the data by clicking on the data by clicking on the data icon to the right of the view.



You could achieve the same thing by using the *View* function (note the capital V).

#view your dataset   
View(campy)

**Examples of hypotheses to be tested could be:**

* Is there a trend in the Campylobacter reporting?
* If yes, what type of trend is it?
* Is there any seasonality in Campylobacter reporting?
* If yes, around what time of the year does it occur?
* Are the residuals white noise? Or are there other events than random variation?

**Inclusion/exclusion criteria:** Only records with daily reporting will be considered (drop records with yearly reporting)

**Time unit for analysis:** Weeks

**Draft plan of analysis:**

**Data cleaning**

* Plot original data
* Data description (frequency distributions, summary tables) of all variables
* Checks for missing data
* Checks for numeric attraction in reporting due to reporting bias, including statistical tests

**Data description**

* To better understand the nature of the data
  + Plot the data
  + Calculate statistical descriptors
  + Checks for stationarity
    - Check for constant mean (linear regression)
    - Check for constant variance (mean vs. Variance plot)
  + Checks for autocorrelation
    - Autocorrelation
    - Partial autocorrelation
    - Correlogram
  + Checks for distribution
    - Histogramme
    - Symmetry plot
    - Normal quantile plot
    - Shapiro-Wilk normality test
  + Checks for transformations
  + Moving average (exploring trend and seasonality)
* Decomposition of the dataset
  + Checks for trend (check R2)
  + Checks for seasonality (taking trend into account)
    - Spectral analysis, Periodogram
    - Check R2 of models
* Residual analysis
  + Plot residuals
  + Statistical descriptors of residuals
  + Checks for stationarity
    - Check for constant mean (linear regression)
    - Check for constant variance (mean vs. Variance plot)
    - Plot of residuals against model
  + Checks for autocorrelation
    - Autocorrelation
    - Partial autocorrelation
    - Correlogram
  + Checks for distribution
    - Histogramme, Symmetry plot, Normal quantile plot
    - Shapiro-Wilk normality test
  + Test for white noise
  + Depending on results of above analyses, carry out transformation, adjust trend/seasonality
  + Possible: Simple modelling of data series with 95% CI, taking outbreaks into account (downweighting/removing)

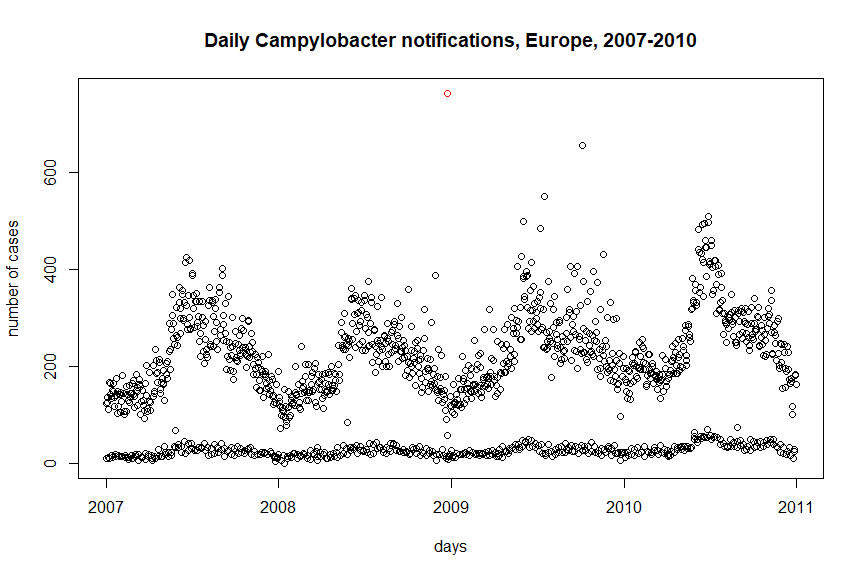
# Question 2

Make any changes necessary to the dataset. Specify the appropriate time unit for aggregation (start with days). Carry out data checking, including tables and plots of all variables, checks for missing data and checks for numeric attraction by day of week and day of month for possible reporting bias. Interpret your findings.

# Help Question 2

In the current form, the data frame is daily counts (aggregate counts by date of notification). Dates are unique, i.e. there is only one count reported for each day. You could double check this using the *duplicated* function.

#plot counts by day  
plot(campy$time, campy$count,   
 main = "Daily Campylobacter notifications, Europe, 2007-2010",   
 xlab = "days", ylab = "number of cases")



This plot shows the original data. There do appear to be some outliers (e.g. increased reporting at the end of December 2008, highlighted in red). It would be interesting to investigate this further and also go back to the country to ask for more information.

Now check if there are missing values in the dataset. You could either do this with the *summary* function, as in question 1, or you could do the following to get counts of missings for the dataset.

# get counts of missing values in the dataset   
sum(is.na(campy))

## [1] 0

To check for notification bias, let us plot the number of cases notified by day of the week. The first step is to extract the date of the week from the date (‘date’ variable) by creating a new variable “dayweek”. If the language on your computer is not set to English, see the appendix for how to change this and get English days of the week.

#extract the weekdays from date  
campy$dayweek <- weekdays(campy$time)

Make sure the days of the week are in the correct order by making it a factor variable with appropriate levels.

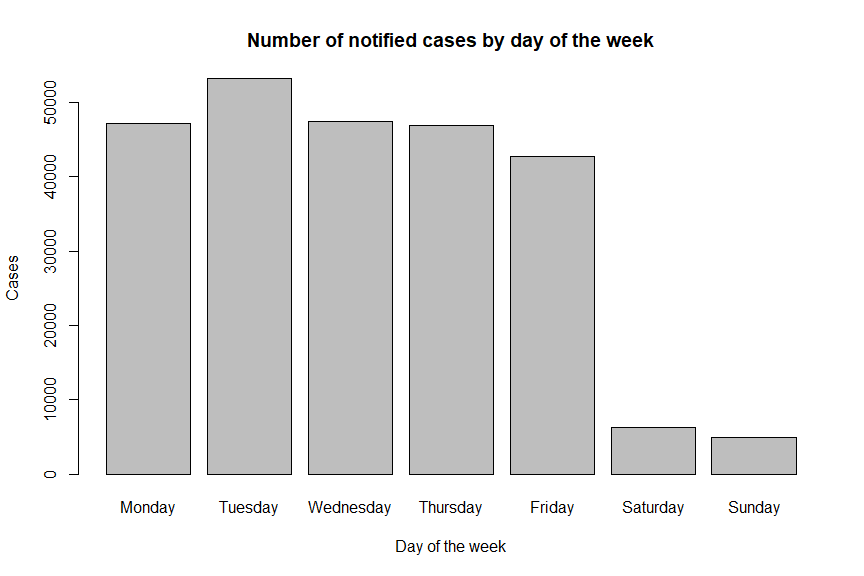
#change to factor with levels   
campy$dayweek <- factor(campy$dayweek,   
 levels = c("Monday", "Tuesday", "Wednesday",   
 "Thursday", "Friday", "Saturday", "Sunday"))

You then need to use the *aggregate* function to collapse your dataset and get counts add the counts for each day together. In the below step, you could also not use the *with* function and instead specify the dataset within *aggregate*. However this produces less useful column names.

# using campy dataframe   
 # sum counts by day of the week   
campy\_dayweek <- with(campy, aggregate(count ~ dayweek, FUN = sum))

Next make a bar chart of the number of cases per day of the week over the whole period. You can use the aggregated *campy\_dayweek* data in the *barplot* function. You may not see all of the x-labels in the plot window, however if you click the “zoom” button these should appear.

# plot counts by day   
 # names.arg is required to label the x axis appropriately  
barplot(campy\_dayweek$count, names.arg = campy\_dayweek$dayweek,   
 main = "Number of notified cases by day of the week",   
 xlab = "Day of the week",   
 ylab = "Cases")



Let us check for numeric attraction by day of the month (think of potential higher notification on the 1st of the month for administrative purposes). As we have seen that there is much fewer notifications during weekends, let us exclude Saturday and Sunday from this analysis. We create a new object that is the copy of ‘campy’ but without all weekend records.

# keep rows from campy that are not saturday or sunday  
campyw <- subset(campy,   
 campy$dayweek != "Saturday" & campy$dayweek != "Sunday")

Create a new variable “daynum”“, extracting the day of the month from the”time" variable. To do this use the *format* function and specify day using “%d”.

We also need to change this to a numeric variable using the *as.numeric* function.

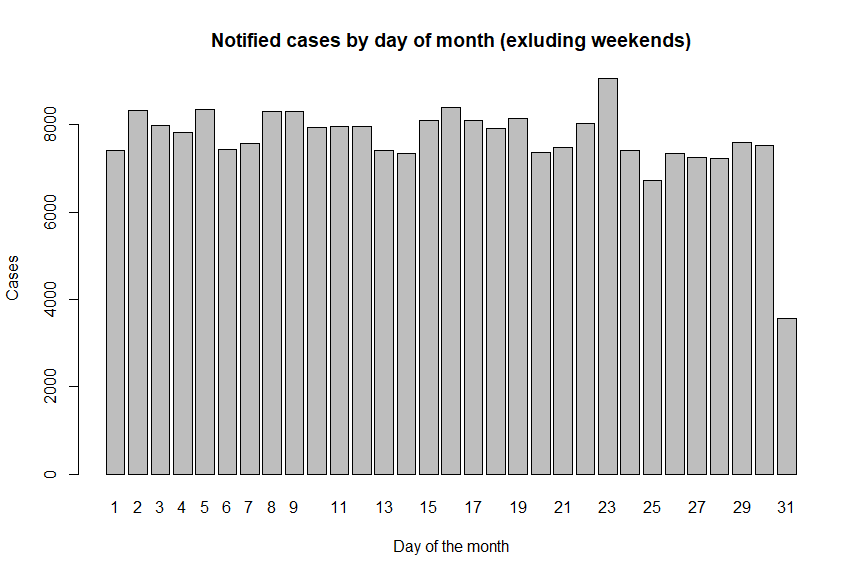
# create variable with day of month  
 # "%d" specifies that day of month is what you want  
campyw$daynum <- format(campyw$time, "%d")  
  
# change to numeric   
campyw$daynum <- as.numeric(campyw$daynum)

We then need to aggregate the number of cases by day of the month, by creating a new data frame “campyw\_daynum”,

# using campyw dataframe  
 # sum counts by day  
campy\_daynum <- with(campyw, aggregate(count ~ daynum, FUN = sum))

Plot another bar chart.

barplot(campy\_daynum$count, names.arg = campy\_daynum$daynum,   
 main = "Notified cases by day of month (exluding weekends)",   
 xlab = "Day of the month",   
 ylab = "Cases")



There appears to be numeric attraction by day of week and day of month. The low number of notifications on Saturday and Sunday is likely due to the fact that some health care settings are closed on weekends. This is a strong argument to work with weekly cases. There also appears to be some higher magnitude of reporting on a Tuesday and on the 23rd. This may be influenced by the high reporting on the 23rd of December 2008 (it is a Tuesday). You could do the plots on the data excluding 2008 to see if the graphs change.

# Question 3

Describe the time series:

* Plot the time series aggregated by time unit of interest
* Check the data for stationarity, autocorrelation , distribution and for possible transformations of the data. Interpret your findings and plan next steps.
  + Note: we do not expect data to be stationary, autocorrelated or normally distributed before it has been decomposed. This is nevertheless an important part in descriptive analysis.
* Use the “moving average” to explore possible trends and seasonality within the time series

# Help Question 3

The first thing to do is to create a data frame where the counts are aggregated by week. Be careful, there are different methods to build the week of the year. Here we will use the *ISOweek* function (part of *ISOweek* package).

# create a variable with week   
campy$week <- ISOweek(campy$time)

Have a look at the new variable

# look at first few rows  
head(campy$date)   
  
# look at your dataset   
View(campy$date)

As before, use the aggregate function to sum the cases by ISO-week.

# using campy dataframe  
 # sum counts by week  
campy\_week <- with(campy, aggregate(count ~ week, FUN = sum))

We will then create a Time-Series object. These are vector or matrices with class of “ts” (and additional attributes) which represent data that have been sampled at equispaced points in time. In the matrix case, each column of the matrix data is assumed to contain a single (univariate) time series. Working with a Time-Series object will make it easier to analyse the time series. We only keep the counts variable and then define the time unit of the new object by specifying the starting point (beginning of first week of 2007) and the frequency (52 weeks).

# Create time series only using campy$week  
campy\_ts <- ts(campy\_week$count, start = c(2007, 0), end = c(2010, 52), frequency = 52)

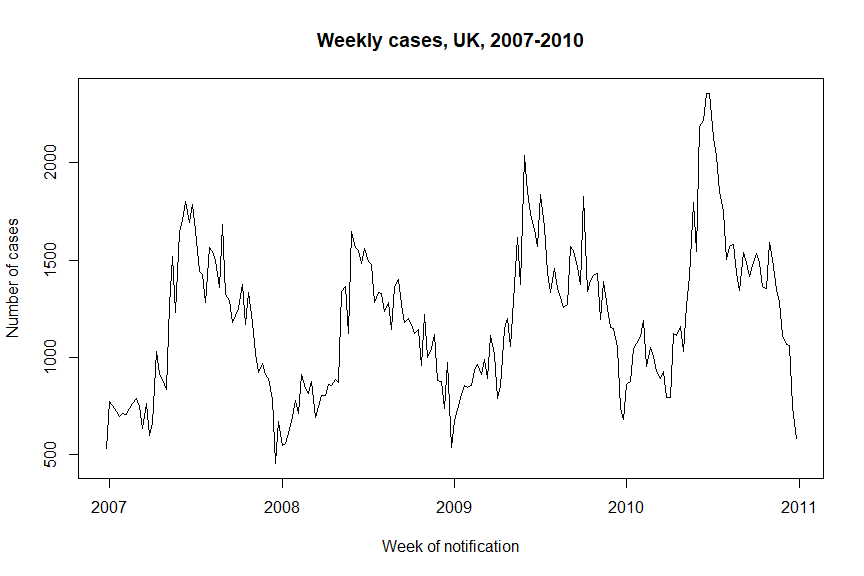
# check the class of camp\_ts  
class(campy\_ts)

You can visualise the data from the time series object, look at basic descriptive statistics and check for missing values.

# view your timeseries object   
campy\_ts  
  
# get descriptive statistics   
summary(campy\_ts)  
  
# check missings   
sum(is.na(campy\_ts))

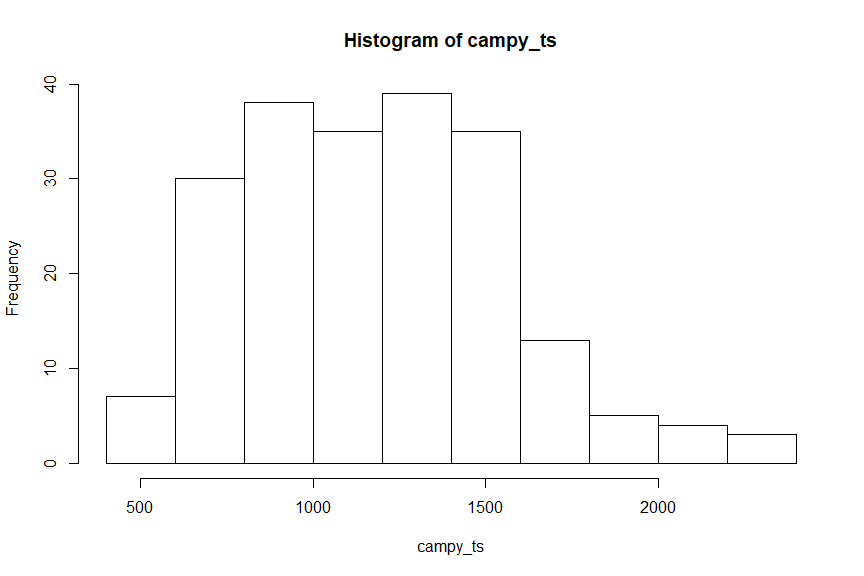
Plot your time series.

# Plot time series by week   
plot(campy\_ts, main = "Weekly cases, UK, 2007-2010",   
 ylab = "Number of cases", xlab = "Week of notification")



Examine the distribution of counts per week.

# plot histogram of counts occurences  
hist(campy\_ts)



Look at the summary statistics, as well as the skewness and kurtosis using the respective functions from the *moments* package.

# summary statistics  
summary(campy\_ts)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 457 876 1178 1190 1437 2356

# look at the skewness  
skewness(campy\_ts)

## [1] 0.4935141

# look at kurtosis  
kurtosis(campy\_ts)

## [1] 3.008542

The mean and median are not too different. There appears to be some skewness in the data. Kurtosis is around 3, which means there is a similar peak as in a normal distribution.

## Checks for stationarity

### Linear regression

Simply speaking, a time series is stationary if its mean level and variance stay steady over time, or as explained by Hyndman and Athanasopoulos (2013): “[a] stationary time series is one whose properties do not depend on the time at which the series is observed”.

We can use linear regression to check whether the mean is constant over time: We first define the time variable as “t”, ranging from week 1 to week 209, covering the whole period:

# define number of timepoints  
t <- 1:209

We can then use our time-series data frame and the time variable in a linear regression using the *lm* function.

# run linear regression for counts over time  
linreg <- lm(campy\_ts ~ t)

It is possible to apply the *summary* function to this regression in order to get a nicely formatted output.

# view output of regression  
summary(linreg)

##   
## Call:  
## lm(formula = campy\_ts ~ t)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -825.55 -269.46 -33.53 208.21 1003.52   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 967.7821 49.6292 19.500 < 2e-16 \*\*\*  
## t 2.1137 0.4098 5.158 5.84e-07 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 357.5 on 207 degrees of freedom  
## Multiple R-squared: 0.1139, Adjusted R-squared: 0.1096   
## F-statistic: 26.6 on 1 and 207 DF, p-value: 5.836e-07

There appears to be an upwards trend in the data. On average, there are 2.11 more cases notified every week. We can also use linear regression to check whether the variance is constant over time.

We first need to calculating a rolling standard deviation. In this case we choose a moving window of 10 weeks. This means that for each time point a standard deviation is calculated based on the five points before and after.

# produce a centred rolling standard deviation with 10 week window  
variance <- rollapply(data = campy\_ts, width = 10, FUN = sd, fill = NA)

# run linear regression of standard deviation over time  
variancereg <- lm(variance ~ t)  
  
# view output of linear regression   
summary(variancereg)

##   
## Call:  
## lm(formula = variance ~ t)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -135.72 -67.23 -19.76 38.31 313.23   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 150.8724 13.5343 11.147 <2e-16 \*\*\*  
## t 0.2922 0.1134 2.578 0.0107 \*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 92.56 on 198 degrees of freedom  
## (9 observations deleted due to missingness)  
## Multiple R-squared: 0.03247, Adjusted R-squared: 0.02758   
## F-statistic: 6.645 on 1 and 198 DF, p-value: 0.01067

There does not appear to be a significant difference in variance over time.

## Checks for autocorrelation

Does the series correlate with itself, and for what lags?

In order to do estimate autocorrelation we will use the *acf* function. This function also creates a plot, however we will not use the plotting function immediately.

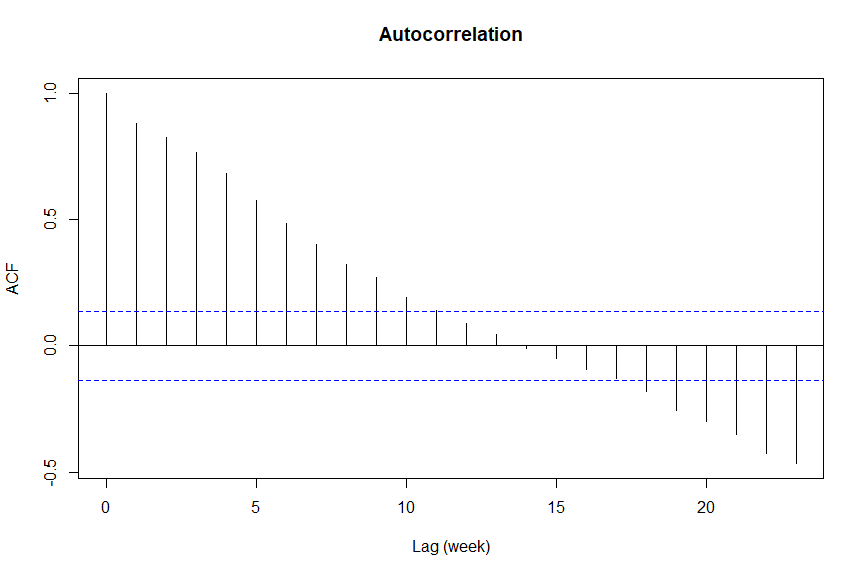
# estimate autocorrelation without plotting  
acplot <- acf(campy\_ts, plot = FALSE)

We then need to transform lags from years to weeks by multiplying the lag variable by 52 (number of weeks in a year).

# change lag from years to weeks  
acplot$lag <- acplot$lag \* 52

Then you can use the regular plot function to visualise these estimates.

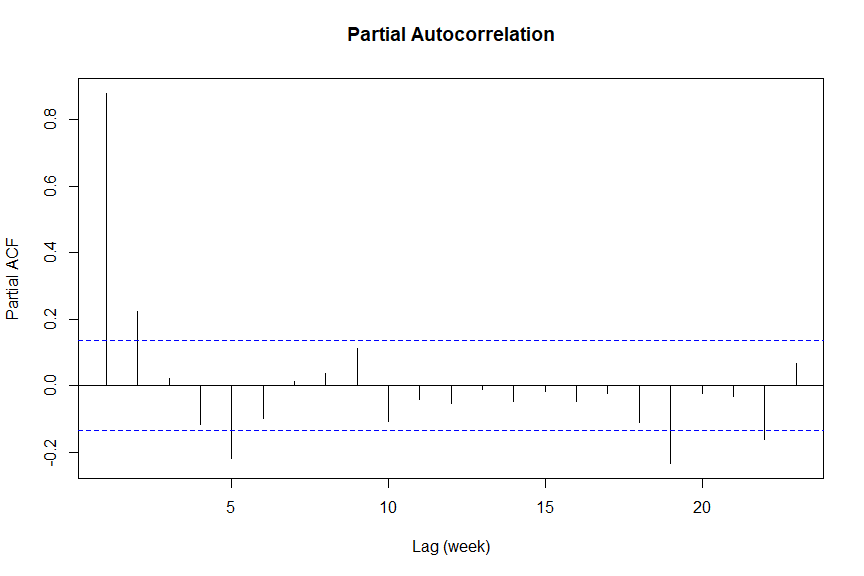
# plot your autocorrelation graph  
plot(acplot, main = "Autocorrelation", xlab = "Lag (week)")



The first line on this autocorrelation graph shows the correlation between each obsersation and itself (at lag zero, therefore always 1), the second shows correlation between each observation and the preceding one (lag one), and so on. Blue lines show the 95% confidence intervals. Where lines go above or below this there is significant autocorrelation (in order to produce a similar confidence intervals to Stata, add *ci.type = “ma”* to the *acf* function). This graph therefore shows substantial autocorrelation in the data.

We now need to investigate possible seasonality (or periodicity). To do this we use the *pacf* function for partial autocorrelation. Essentially what this does, is adjust the autocorrelation in each observation for any autocorrelation from previous observations.

# estimate partial autocorrelation without plotting  
pacplot <- pacf(campy\_ts, plot = FALSE)   
  
# change lag from years to weeks  
pacplot$lag <- pacplot$lag \* 52  
  
# plot your partial autocorrelation graph  
plot(pacplot, main = "Partial Autocorrelation", xlab = "Lag (week)")

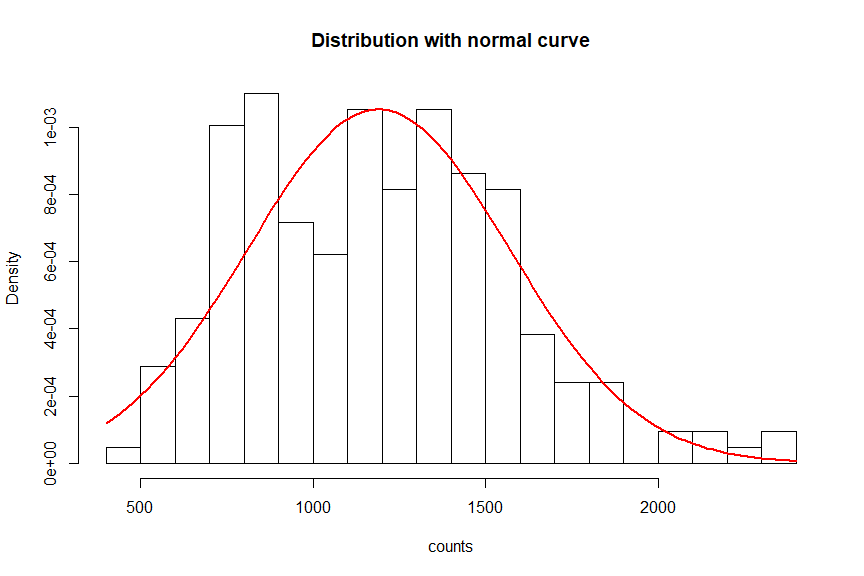


The partial autocorrelation graph shows autocorrelation at the first lag and further autocorrelation at around the 5th and 18/19th lag. The seasonal component of this time series needs to be examined.

## Checks for normality

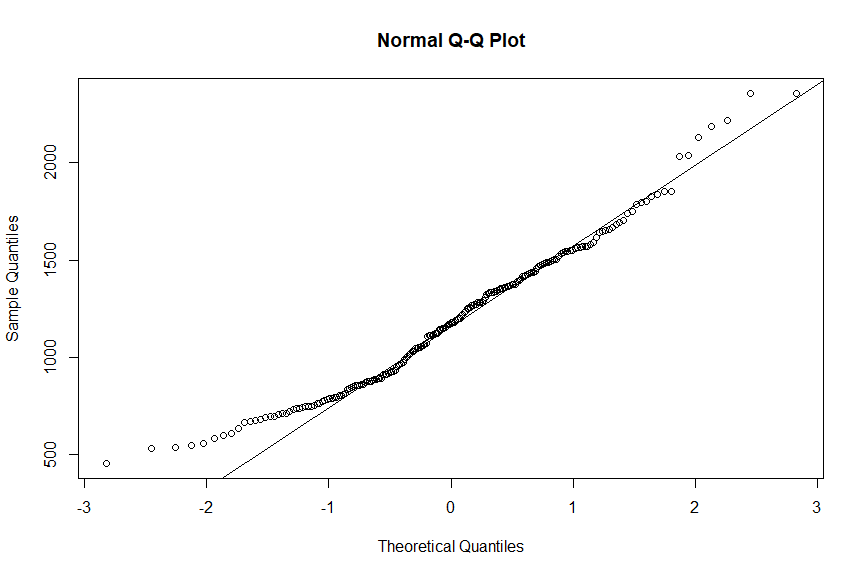
Begin by visually inspecting the data. We can plot a histogram of the data using *hist* and add a normal curve by combining the *curve* and *dnorm* functions. In this case we want to plot probability density rather than frequency on our histogram.

#plot your histogram defining labels and breaks  
hist(campy\_ts, probability = TRUE,   
 main = "Distribution with normal curve",   
 xlab = "counts",   
 breaks = 15  
 )  
  
#plot your normal curve on top  
curve(dnorm(x, mean = mean(campy\_ts), sd = sd(campy\_ts)),   
 add = TRUE, col = "red", lwd = 2)



You can also inspect this using a normal quantile plot (or QQ-plot). To do this combine the *qqnorm* and *qqline* functions. If the dots do not follow the straight line then it is indicative of a non-normal distribution.

# quantile plot for the data  
qqnorm(campy\_ts)  
qqline(campy\_ts)



Both of these plots show that the data is skewed and not normally distributed.

This can also be show statistically using the shapiro-Wilk test for normality. This shows significant deviation from normal, so we may need to consider transforming the data.

# run shapiro-wilk test on data  
shapiro.test(campy\_ts)

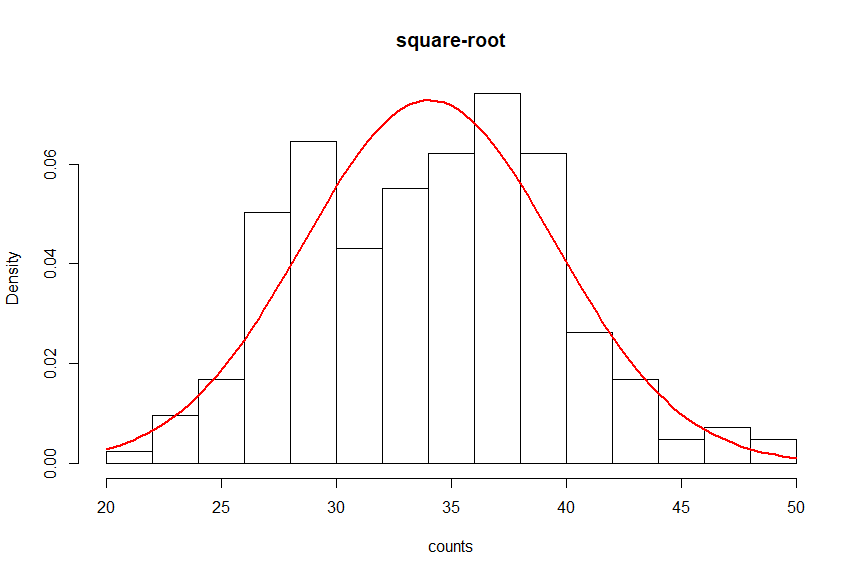
##   
## Shapiro-Wilk normality test  
##   
## data: campy\_ts  
## W = 0.97324, p-value = 0.0005171

## Check transformations

We will check transforming using square-root, log and square.

First try the square-root using the *sqrt* function.

# check transformation by square-root   
  
# apply square-root to your dataset  
sqrtcampy\_ts <- sqrt(campy\_ts)  
  
# plot your histogram defining labels and breaks  
hist(sqrtcampy\_ts, probability = TRUE,   
 main = "square-root",   
 xlab = "counts",   
 breaks = 15  
 )  
  
# plot your normal curve on top  
curve(dnorm(x, mean = mean(sqrtcampy\_ts), sd = sd(sqrtcampy\_ts)),   
 add = TRUE, col = "red", lwd = 2)

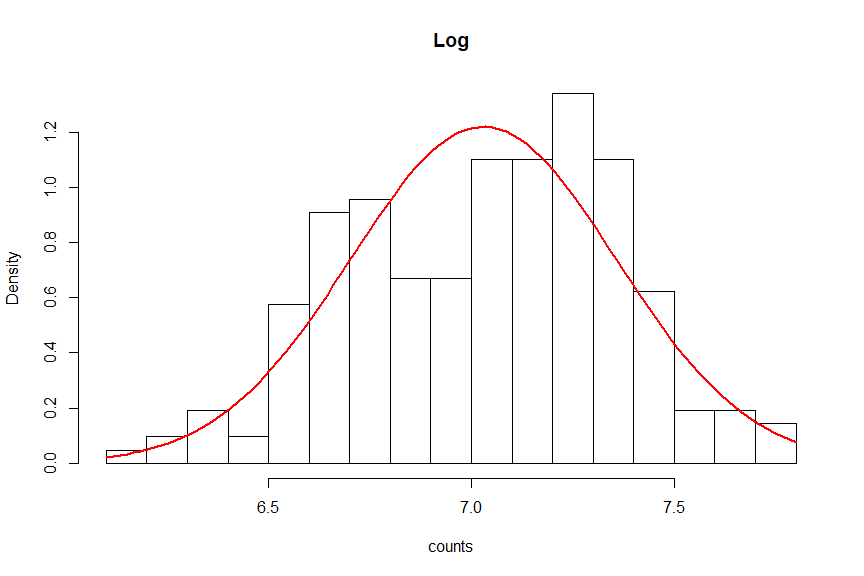


# run shapiro-wilk test on data  
shapiro.test(sqrtcampy\_ts)

##   
## Shapiro-Wilk normality test  
##   
## data: sqrtcampy\_ts  
## W = 0.98762, p-value = 0.06633

Next try log transformation using the *log* function.

# check transformation by log   
  
# apply square-root to your dataset  
logcampy\_ts <- log(campy\_ts)  
  
# plot your histogram defining labels and breaks  
hist(logcampy\_ts, probability = TRUE,   
 main = "Log",   
 xlab = "counts",   
 breaks = 15  
 )  
  
# plot your normal curve on top  
curve(dnorm(x, mean = mean(logcampy\_ts), sd = sd(logcampy\_ts)),   
 add = TRUE, col = "red", lwd = 2)

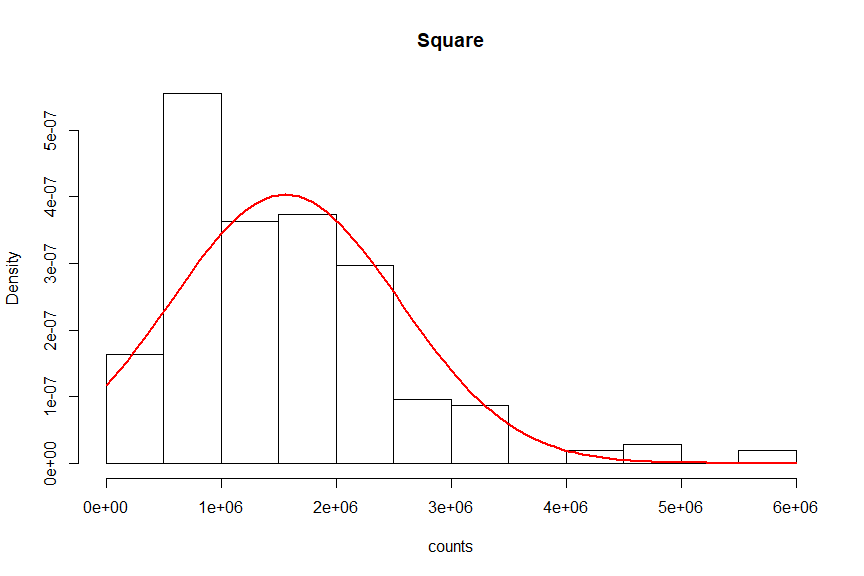


# run shapiro-wilk test on data  
shapiro.test(logcampy\_ts)

##   
## Shapiro-Wilk normality test  
##   
## data: logcampy\_ts  
## W = 0.98509, p-value = 0.02666

Finally, try transforming by the square by raising the data to the power of 2.

# check transformation by square   
  
# apply square-root to your dataset  
sqcampy\_ts <- campy\_ts^2  
  
# plot your histogram defining labels and breaks  
hist(sqcampy\_ts, probability = TRUE,   
 main = "Square",   
 xlab = "counts",   
 breaks = 15  
 )  
  
# plot your normal curve on top  
curve(dnorm(x, mean = mean(sqcampy\_ts), sd = sd(sqcampy\_ts)),   
 add = TRUE, col = "red", lwd = 2)



# run shapiro-wilk test on data  
shapiro.test(sqcampy\_ts)

##   
## Shapiro-Wilk normality test  
##   
## data: sqcampy\_ts  
## W = 0.89747, p-value = 8.962e-11

The transformation that renders the original time series most normal is the square root (W = 0.98762, p-value = 0.06633). However, does this transformation make sense? A log transformation and also no transformation should also be considered.

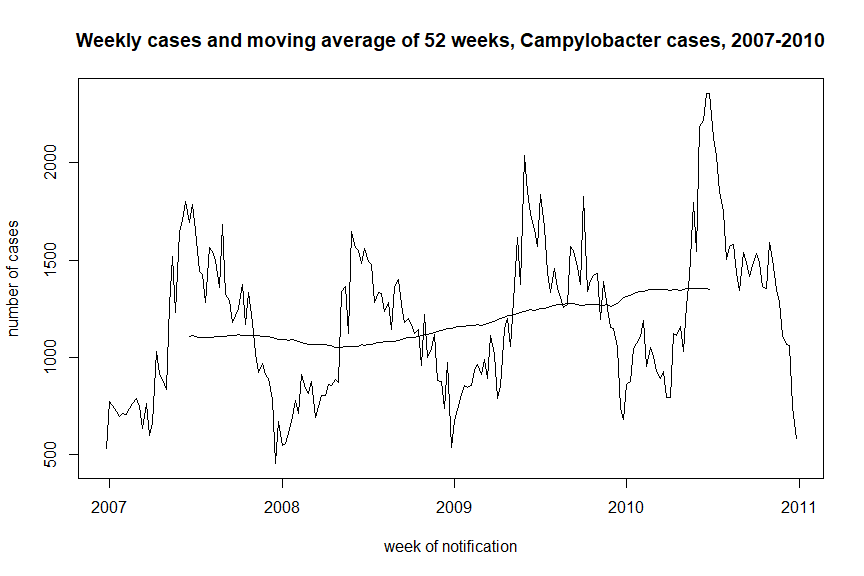
# Moving average

We now want to use moving averages to investigate possible trends and seasonality. We can calculate these using the *rollapply* function. Similar to how we calculated the standard deviation when checking for variation over time. We will calculate the centred moving average for several different time windows. Try for example to do one each for 52 weeks, 26 weeks and 13 weeks.

# calculate annual moving average  
ma52 <- rollapply(campy\_ts, width = 52, FUN = mean)

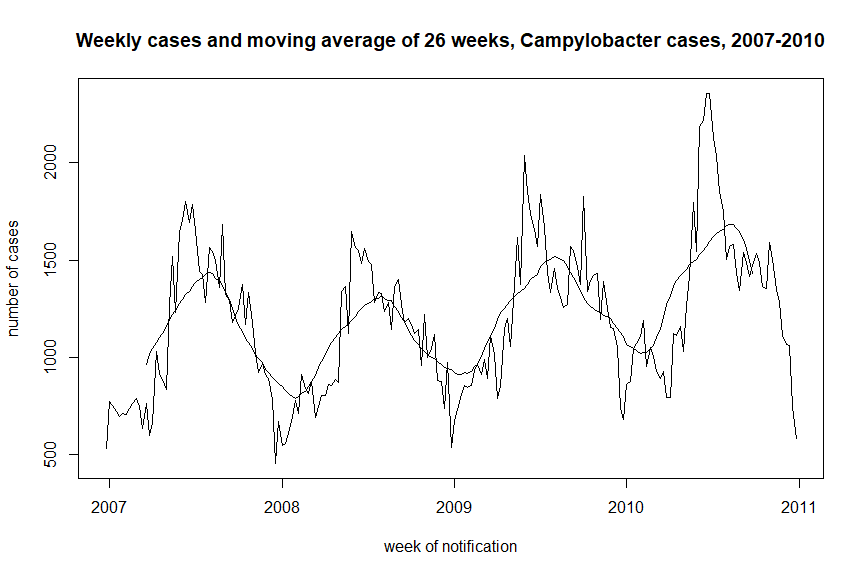
We can then plot this using the *ts.plot* function.

# plot your timeseries with annual moving average  
ts.plot(campy\_ts, ma52,   
 main = "Weekly cases and moving average of 52 weeks, Campylobacter cases, 2007-2010",   
 ylab = "number of cases",   
 xlab = "week of notification")



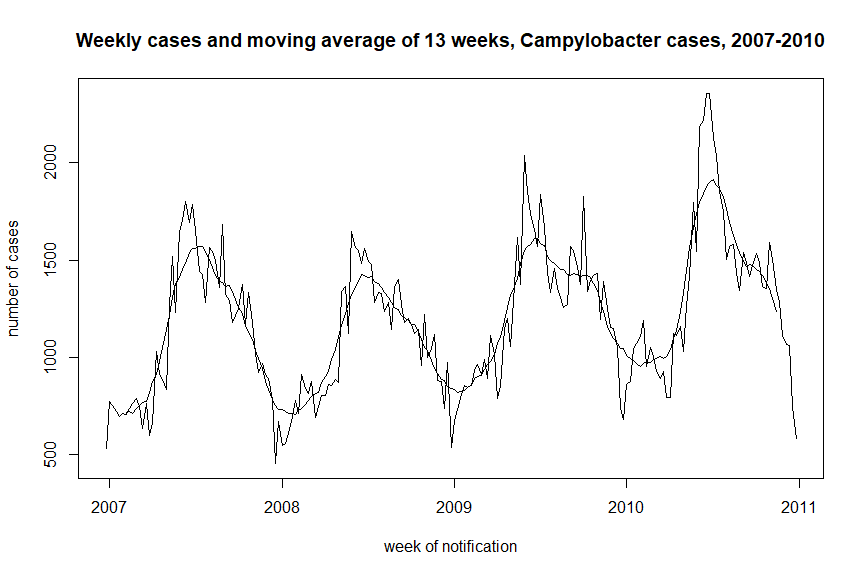
Now try the same for a 26 week window.

# calculate 26 weekly moving average  
ma26 <- rollapply(campy\_ts, width = 26, FUN = mean)  
  
# plot your timeseries with annual moving average  
ts.plot(campy\_ts, ma26,   
 main = "Weekly cases and moving average of 26 weeks, Campylobacter cases, 2007-2010",   
 ylab = "number of cases",   
 xlab = "week of notification")



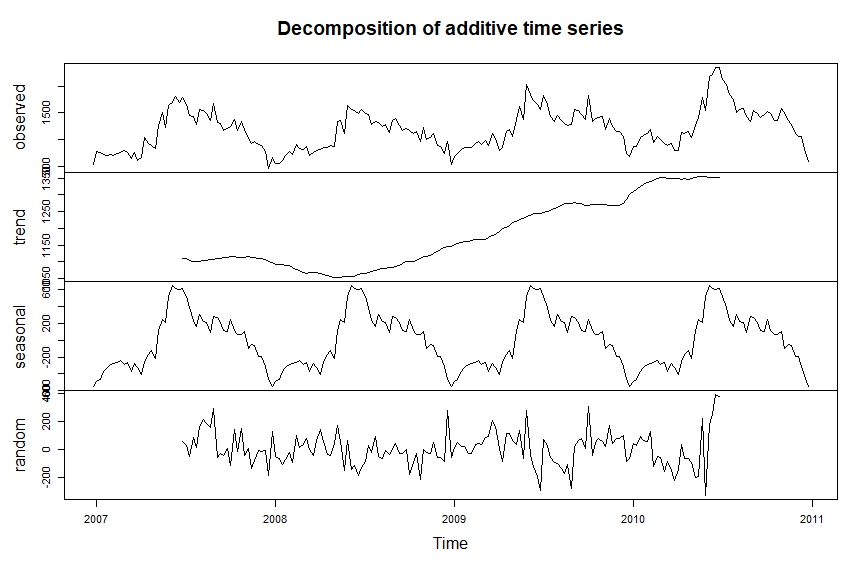
And now do the same for a 13 week window.

# calculate 13 weekly moving average  
ma13 <- rollapply(campy\_ts, width = 13, FUN = mean)  
  
# plot your timeseries with annual moving average  
ts.plot(campy\_ts, ma13,   
 main = "Weekly cases and moving average of 13 weeks, Campylobacter cases, 2007-2010",   
 ylab = "number of cases",   
 xlab = "week of notification")



It is also possible to decompose a time series automatically into seasonal, trend and irregular components using moving averages with the *decompose* function.

plot(decompose(campy\_ts))



In the UK, there appears to be an increasing annual trend in Campylobacter reports from 2007 to 2010 (this can be seen with the moving average of 51 weeks). There appears also to be an annual peak of reports in the summer (can be seen with the moving average of 25 weeks). The moving average window of 13 weeks, gives more information and shows that there appears to be a peak in early summer and a second smaller peak after the summer. Without more background it is difficult to judge whether the trend is a real trend, the peaks are real peaks or they are perhaps due to batch reporting. However a literature search indicates that Great Britain has marked seasonal peaks in Campylobacter enteritis occurring in May and September (“Investigation of Faecal Specimens for Bacterial Pathogens”; National Standards Methods, 2010. <http://www.hpa-standardmethods.org.uk/documents/bsop/pdf/bsop30.pdf> [Accessed October 2011]).

# Question 4

Decompose the time series: - Decide on an appropriate trend to use - With the trend removed, analyse seasonality - What seasonality will you include in a final model? Why?

# Help Question 4

## Regression type

We are using counts data, where sometimes Poisson regression or negative binomial regression (if overdispersion is present) may be indicated. However we have very high counts, with all counts per week exceeding 500. In addition, the histogram does not look particularly Poisson distributed. Therefore it may make sense to use linear regression in the first instance. At residual analysis stage we can then verify if this choice is appropriate. A next step may be to decide if the data follow a linear or a quadratic trend. Look at the graphs and regression outputs for both trends.

First, define your time variable. The last time variable we defined by typing in the upper limit as a fixed value, however we can similarly make reference to the length of our time series object.

# define time variable (number of weeks)   
t <- 1:length(campy\_ts)

We now run a linear regression as previously, showing the summary as well as the confidence intervals.

# run linear regression for counts over time  
linreg <- lm(campy\_ts ~ t)   
  
# view output of regression  
summary(linreg)

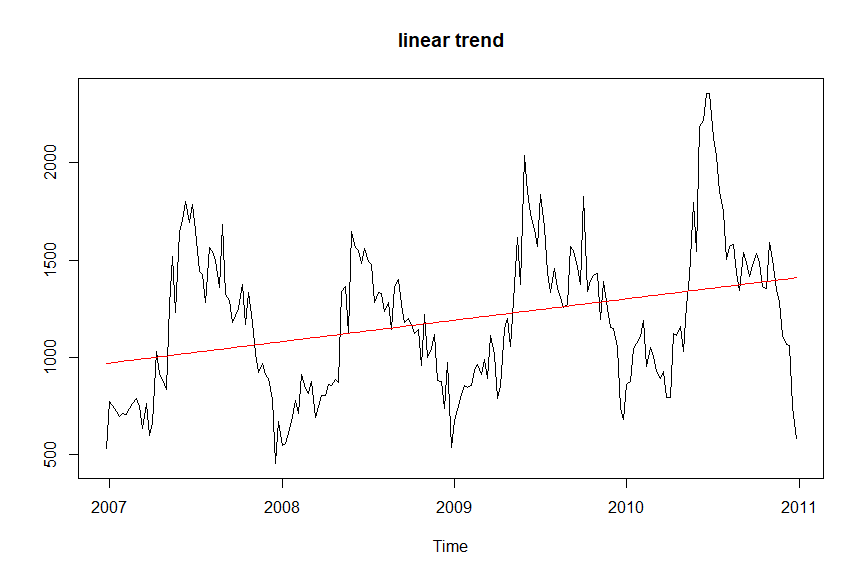
##   
## Call:  
## lm(formula = campy\_ts ~ t)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -825.55 -269.46 -33.53 208.21 1003.52   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 967.7821 49.6292 19.500 < 2e-16 \*\*\*  
## t 2.1137 0.4098 5.158 5.84e-07 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 357.5 on 207 degrees of freedom  
## Multiple R-squared: 0.1139, Adjusted R-squared: 0.1096   
## F-statistic: 26.6 on 1 and 207 DF, p-value: 5.836e-07

# view confidence intervals   
confint(linreg)

## 2.5 % 97.5 %  
## (Intercept) 869.938599 1065.62563  
## t 1.305756 2.92168

We can then plot the fitted values together with our observed data, similar to moving averages above, using the *ts.plot* function.

ts.plot(campy\_ts, linreg$fitted.values,   
 col = c("black", "red"),   
 main = "linear trend")

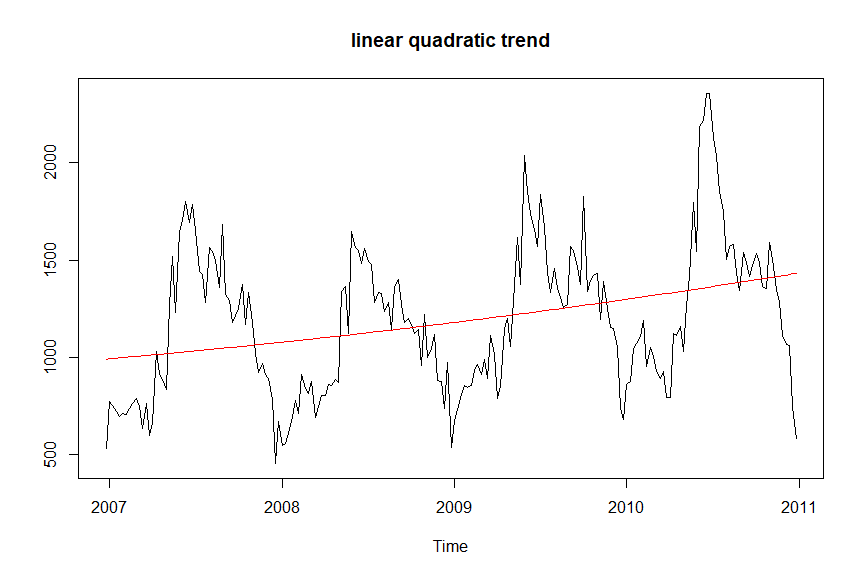


For a quadratic model we need to first square our time variable and then run it together with the previous linear regression.

# create a new squared timed variable  
t2 <- t^2  
  
# run linear regression   
quadreg <- lm(campy\_ts ~ t + t2)  
  
# view output   
summary(quadreg)

##   
## Call:  
## lm(formula = campy\_ts ~ t + t2)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -848.20 -274.95 -38.82 206.46 996.29   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 9.911e+02 7.504e+01 13.207 <2e-16 \*\*\*  
## t 1.451e+00 1.650e+00 0.879 0.380   
## t2 3.157e-03 7.610e-03 0.415 0.679   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 358.2 on 206 degrees of freedom  
## Multiple R-squared: 0.1146, Adjusted R-squared: 0.106   
## F-statistic: 13.33 on 2 and 206 DF, p-value: 3.586e-06

# plot with observed data   
ts.plot(campy\_ts, quadreg$fitted.values,   
 col = c("black", "red"),   
 main = "linear quadratic trend")



The adjusted R2 of the model with a quadratic trend (R2=0.106) is slightly low than the R2 of the model with a linear trend (R2= 0.109). However the difference between these values is minimal. In addition, R2 values do not take the complexity of the model into account (a quadratic model is more complex than a linear model). We can examine the Akaike Information Criterion (AIC), which takes model complexity into account.

# view linear regression AIC   
AIC(linreg)

## [1] 3054.531

# view quadratic AIC  
AIC(quadreg)

## [1] 3056.357

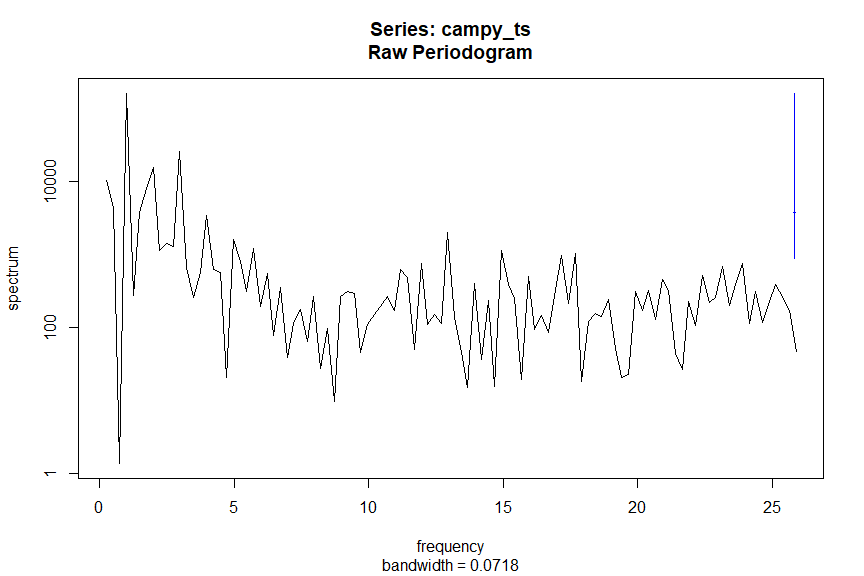
The AIC for the model with a linear trend is 3054.5 and the AIC for the model with quadratic trend is 3056.4 (the smaller the AIC the better the fit). The difference is negligible. We may want to use the principle of parsimony and continue our modelling using the simpler model, with the linear trend.

Conclusions: In the overall dataset, we see an upwards trend over time, which we expected from the results of the moving average analysis. The regression output shows a coefficient of 2.11 which indicates an increase of 2.11 cases for each week in the dataset. The p-value is highly significant, which can also be a reflection of the high sample size. It appears that the trend is increasing, but we cannot conclude that without more information on the surveillance system and reporting practices. Please also note that for educational purposes we are using linear regression to model the trend here – linear regression may only be used if the data show no signs of overdispersion, which is rare in count data. However if counts are very high, then the distribution may approximate the normal distribution in which case it is OK to use linear regression. Furthermore, it is important that we have no missing data. If this were the case, it would be important to set actions for NA values when running the *lm* function (see the help section for more).

# Seasonality

We will check for seasonality on the data after the trend is removed, however we get basically the same results if we check for seasonality on the original data. We can find the most important periods in the data by doing a spectral analysis. This can be done using the *spec.pgram* function. By specifying *fast* as false, stops the function from “padding” the series with extra zeros to enable fast fourier transform.

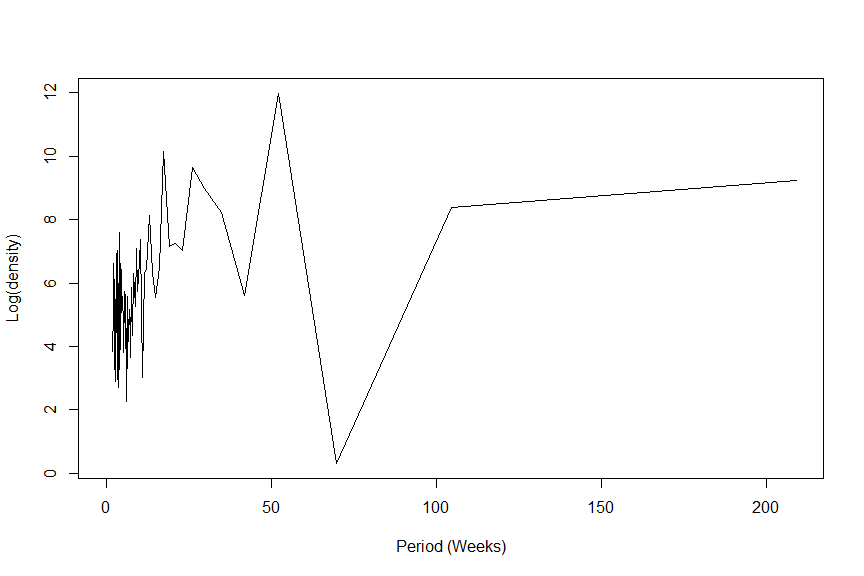
# get spectrograme with detrended   
periodo <- spec.pgram(campy\_ts, fast = FALSE)



To plot the periodogram appropriately we need to divide the frequencies by 52, as it is originally given in years.

From the periodogram we can see that there may be several important periods. The first period of importance is obviously 52 weeks, after that we see 1 or 2 further important peaks.

# plot spectogram as weeks (rather than years)  
plot(1/(periodo$freq/52), log(periodo$spec),   
 type = 'l',  
 xlim = c(0, 209),   
 xlab = "Period (Weeks)", ylab = "Log(density)")



We can extract the frequencies with the highest peaks and extract the weeks – note that you need to use the *order* function in combination with the negative sign to sort the output based on highest first.

# extract important weeks   
 # multiply by 52 because currently in years  
1 / periodo$freq[order(-periodo$spec)] \* 52

## [1] 52.250000 17.416667 26.125000 209.000000 29.857143 104.500000  
## [7] 34.833333 13.062500 4.019231 10.450000 20.900000 19.000000  
## [13] 9.086957 3.483333 23.222222 2.943662 3.028986 9.952381  
## [19] 4.354167 2.177083 2.247312 16.076923 12.294118 4.644444  
## [25] 13.933333 11.611111 8.360000 2.322222 3.265625 4.543478  
## [31] 2.488095 3.732143 2.200000 3.426230 2.069307 7.740741  
## [37] 3.073529 2.548780 2.458824 5.648649 2.612500 2.132653  
## [43] 9.500000 5.500000 41.800000 6.531250 2.049020 5.805556  
## [49] 4.860465 14.928571 2.271739 3.370968 2.750000 3.603448  
## [55] 2.375000 2.090000 2.296703 2.985714 2.223404 4.976190  
## [61] 8.708333 6.966667 2.580247 4.750000 2.029126 2.824324  
## [67] 5.097561 4.180000 3.166667 2.786667 3.943396 2.518072  
## [73] 2.863014 2.111111 4.098039 7.206897 2.154639 4.265306  
## [79] 5.225000 2.348315 6.147059 3.215385 3.119403 8.038462  
## [85] 6.741935 2.714286 4.446809 2.009615 5.358974 3.870370  
## [91] 2.430233 7.464286 3.666667 6.333333 2.402299 2.645570  
## [97] 11.000000 2.679487 3.317460 2.902778 3.542373 3.800000  
## [103] 5.971429 69.666667

We can see that after a period of 52.25, the period of 17.4 is most important. This is approximately a third of a year, so this may be a plausible further period to explore. The next most important period is 26.1, which is around half a year, again a possible plausible period to explore. Let’s just start with the most important period of 52.25 weeks. We can create a sine and cosine for this period and add it to our model with the linear trend.

# defining sine and cosine terms  
sin52 <- sin(2 \* pi \* t / 52.25)   
cos52 <- cos(2 \* pi \* t / 52.25)

Once you have defined your sine and cosine, you can add these to a linear regression as before.

# run linear regression for counts over time  
linreg2 <- lm(campy\_ts ~ t + sin52 + cos52)   
  
# view output of regression  
summary(linreg2)

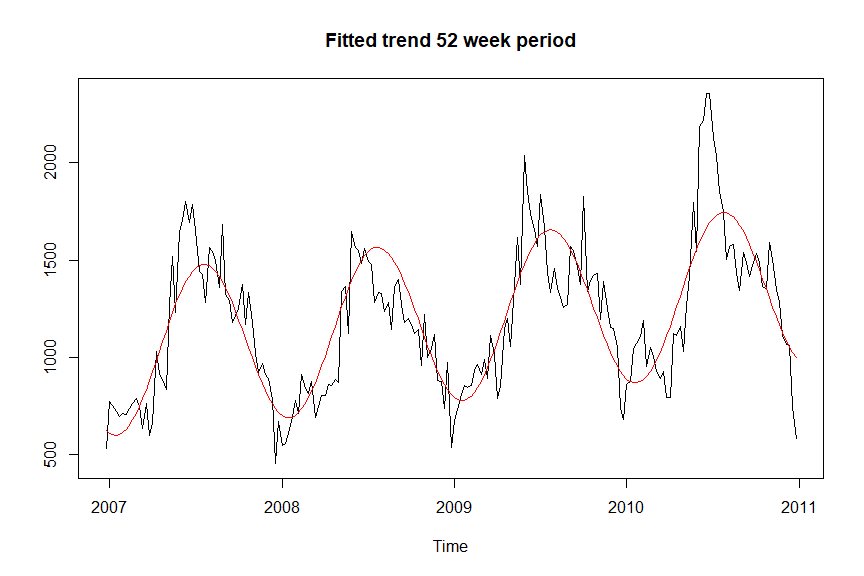
##   
## Call:  
## lm(formula = campy\_ts ~ t + sin52 + cos52)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -418.07 -143.72 -3.37 101.39 690.68   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 1009.5423 28.7462 35.119 < 2e-16 \*\*\*  
## t 1.7160 0.2385 7.195 1.16e-11 \*\*\*  
## sin52 -196.3375 20.3486 -9.649 < 2e-16 \*\*\*  
## cos52 -366.1131 19.9606 -18.342 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 204 on 205 degrees of freedom  
## Multiple R-squared: 0.7141, Adjusted R-squared: 0.7099   
## F-statistic: 170.7 on 3 and 205 DF, p-value: < 2.2e-16

# view confidence intervals   
confint(linreg2)

## 2.5 % 97.5 %  
## (Intercept) 952.866290 1066.21838  
## t 1.245762 2.18624  
## sin52 -236.456898 -156.21808  
## cos52 -405.467499 -326.75869

The adjusted R-squared is now around 0.71. This is a great improvement over the R-squared from the model just with a linear trend. The sine and cosine parameters are also statistically significant. We can look at the graph associated with the model.

ts.plot(campy\_ts, linreg2$fitted.values,   
 col = c("black", "red"),   
 main = "Fitted trend 52 week period")



The model does not look too bad, but it does not capture the dual peaks that we see in the data. We can now see if a further period improves our model. We create further sine and cosine terms and can add these to the model.

# defining sine and cosine terms  
sin17 <- sin(2 \* pi \* t / 17.4)   
cos17 <- cos(2 \* pi \* t / 17.4)   
  
# run linear regression for counts over time  
linreg3 <- lm(campy\_ts ~ t + sin52 + cos52 + sin17 + cos17)   
  
# view output of regression  
summary(linreg3)

##   
## Call:  
## lm(formula = campy\_ts ~ t + sin52 + cos52 + sin17 + cos17)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -337.25 -117.84 -29.87 105.71 520.32   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 1002.1301 23.5740 42.510 < 2e-16 \*\*\*  
## t 1.7879 0.1957 9.136 < 2e-16 \*\*\*  
## sin52 -195.1902 16.6607 -11.716 < 2e-16 \*\*\*  
## cos52 -365.8815 16.3418 -22.389 < 2e-16 \*\*\*  
## sin17 58.7585 16.3823 3.587 0.00042 \*\*\*  
## cos17 -155.0303 16.3351 -9.491 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 167 on 203 degrees of freedom  
## Multiple R-squared: 0.8102, Adjusted R-squared: 0.8056   
## F-statistic: 173.3 on 5 and 203 DF, p-value: < 2.2e-16

# view confidence intervals   
confint(linreg3)

## 2.5 % 97.5 %  
## (Intercept) 955.648798 1048.611304  
## t 1.402008 2.173724  
## sin52 -228.040457 -162.340033  
## cos52 -398.102992 -333.660011  
## sin17 26.457258 91.059823  
## cos17 -187.238523 -122.822071

We can see that all sine and cosine terms are significant. The adjusted R-squared is around 0.81. The adjusted R-squared for the previous model was around 0.71 – therefore the new model is quite a big improvement.

We can also compare the AIC for these models

# show AIC for both models  
AIC(linreg2)

## [1] 2822.12

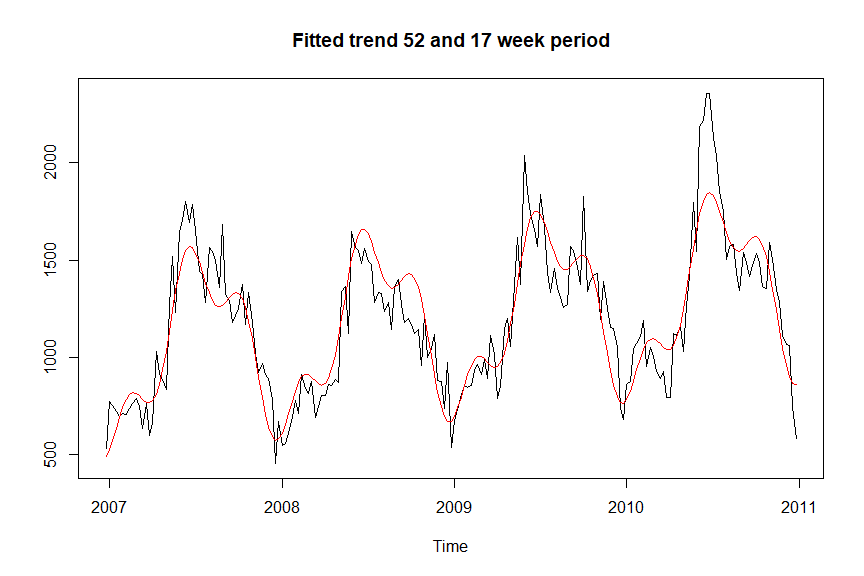
AIC(linreg3)

## [1] 2740.456

The AIC for the model with the extra seasonal term is 2740.5, which is substantially lower than the AIC with only one seasonal term.

We can also plot the new model.

ts.plot(campy\_ts, linreg3$fitted.values,   
 col = c("black", "red"),   
 main = "Fitted trend 52 and 17 week period")



This graphs explains the data better. We can see the double peak in the data around the late spring/summer and early autumn period. We also see a third “peak” at the beginning of the year, which we do not necessarily expect in terms of disease. This may however reflect changes in reporting around Christmas and Easter time. From the periodogram, we also saw that a period of 26 may also be an important period. We could try and see if adding this period improves the model substantially.

# defining sine and cosine terms  
sin26 <- sin(2 \* pi \* t / 26.1)   
cos26 <- cos(2 \* pi \* t / 26.1)   
  
# run linear regression for counts over time  
linreg4 <- lm(campy\_ts ~ t + sin52 + cos52 + sin17 + cos17 + sin26 + cos26)   
  
# view output of regression  
summary(linreg4)

##   
## Call:  
## lm(formula = campy\_ts ~ t + sin52 + cos52 + sin17 + cos17 + sin26 +   
## cos26)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -374.82 -95.13 -9.30 90.04 502.95   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 1008.7035 21.6053 46.688 < 2e-16 \*\*\*  
## t 1.7245 0.1796 9.604 < 2e-16 \*\*\*  
## sin52 -196.1872 15.2143 -12.895 < 2e-16 \*\*\*  
## cos52 -366.0299 14.9207 -24.532 < 2e-16 \*\*\*  
## sin17 58.3806 14.9579 3.903 0.00013 \*\*\*  
## cos17 -155.1074 14.9145 -10.400 < 2e-16 \*\*\*  
## sin26 -40.9407 14.9992 -2.730 0.00691 \*\*   
## cos26 88.3819 14.9143 5.926 1.33e-08 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 152.5 on 201 degrees of freedom  
## Multiple R-squared: 0.8434, Adjusted R-squared: 0.8379   
## F-statistic: 154.6 on 7 and 201 DF, p-value: < 2.2e-16

# view confidence intervals   
confint(linreg4)

## 2.5 % 97.5 %  
## (Intercept) 966.101380 1051.305573  
## t 1.370442 2.078594  
## sin52 -226.187365 -166.187117  
## cos52 -395.451067 -336.608724  
## sin17 28.886028 87.875104  
## cos17 -184.516421 -125.698365  
## sin26 -70.516745 -11.364691  
## cos26 58.973389 117.790373

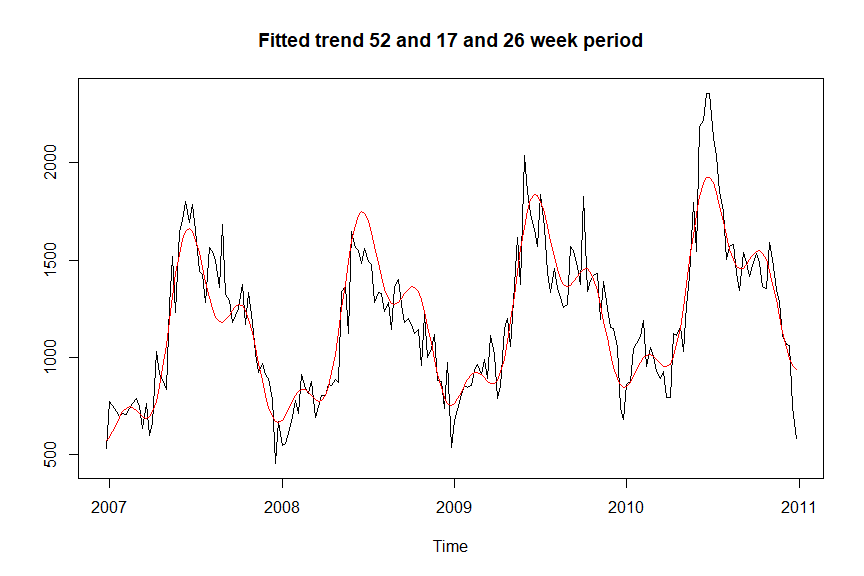
# view AIC for regression   
AIC(linreg4)

## [1] 2704.356

The model including 3 periods still shows that all sine and cosine parameters are significant. The adjusted R2 is now around 0.84. While this is higher than the last model, which had an adjusted R-squared of around 0.81, it is not substantially higher. The AIC is around 2704, which is lower than the previous one at 2740

If we graph the model, we can see that the model fits the data well, but it is difficult to visually see the difference between the previous and current model.

ts.plot(campy\_ts, linreg4$fitted.values,   
 col = c("black", "red"),   
 main = "Fitted trend 52 and 17 and 26 week period")



The model with just 2 seasonal terms may be an adequate model, given the principle of parsimony, however the model with 3 seasonal terms is interesting too. The detail of the model we are interested in depends on the objectives of our time series analysis. Before choosing a final model, we also need to analyse the residuals.

# Question 5

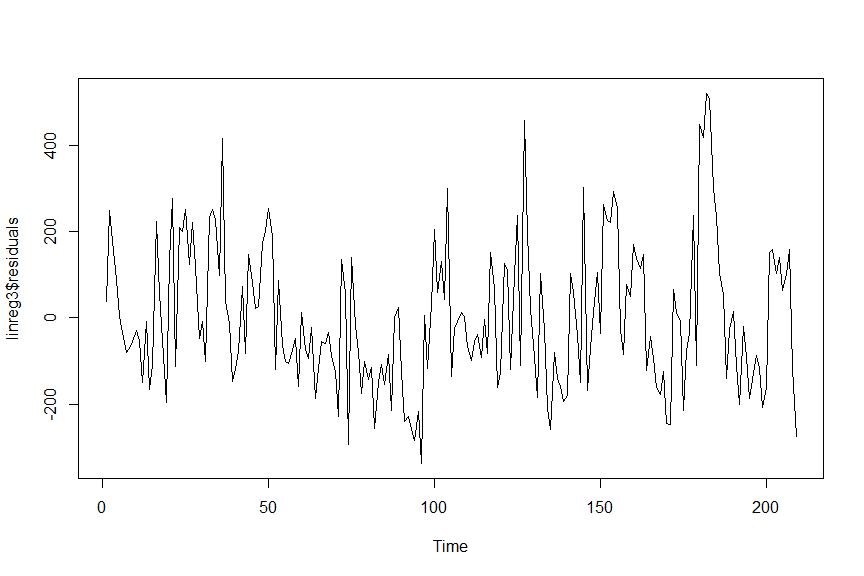
Analyse the residuals of the model containing a linear trend and seasonality with two periods: 52.25 and 17.4 weeks.  
You can check for normality, check stationarity and for autocorrelation. What do you conclude? What would be your next steps?

# Help Question 5

## Initial inspection

We can extract the residuals quite simply from a linear regression and plot them. In this case we go back to linreg3 which contained periods of 52 and 17.33 weeks.

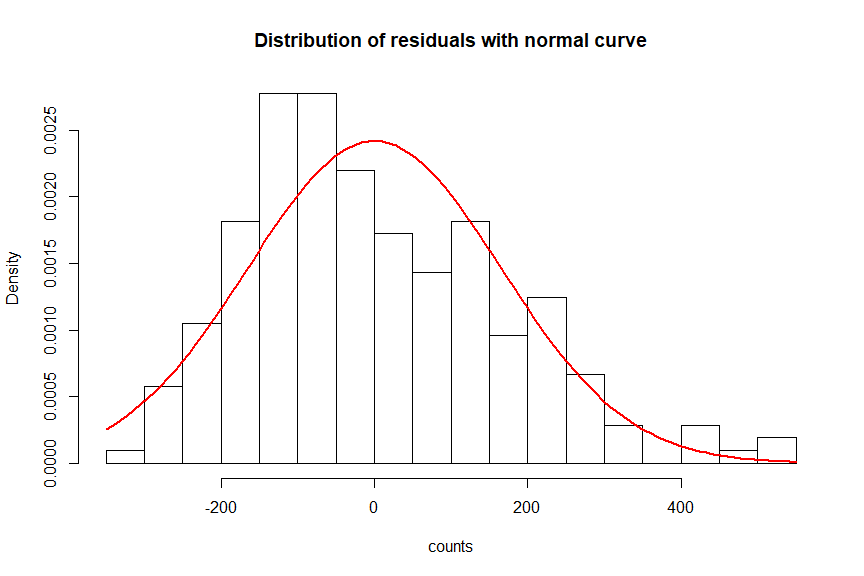
# plot residuals of linear regression   
ts.plot(linreg3$residuals)



When looking at the residual plot it looks like there may be some leftover seasonality. Therefore we may want to bear in mind that the model with 3 seasonal terms may be better.

## Normality

#plot your histogram defining labels and breaks  
hist(linreg3$residuals, probability = TRUE,   
 main = "Distribution of residuals with normal curve",   
 xlab = "counts",   
 breaks = 15  
 )  
  
#plot your normal curve on top  
curve(dnorm(x, mean = mean(linreg3$residuals),   
 sd = sd(linreg3$residuals)),   
 add = TRUE, col = "red", lwd = 2)



# run shapiro-wilk test on data  
shapiro.test(linreg3$residuals)

##   
## Shapiro-Wilk normality test  
##   
## data: linreg3$residuals  
## W = 0.96603, p-value = 6.382e-05

The histogram is skewed to the right and the Shapiro-Wilk test for normality is significant, which tells us the data are not compatible with the normal distribution. We can also look at summary statistics.

# summary statistics  
summary(linreg3$residuals)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## -337.25 -117.84 -29.87 0.00 105.71 520.32

# look at the skewness  
skewness(linreg3$residuals)

## [1] 0.6597569

# look at kurtosis  
kurtosis(linreg3$residuals)

## [1] 3.195109

We see that there is some skewness in the data (it is positive, so it is skewed to the right) and also a kurtosis that is higher than 3. The mean and the median are different. This could indicate a presence of outbreaks. It may be worth revising the seasonality – to see if we adequately described seasonality.

## Stationarity

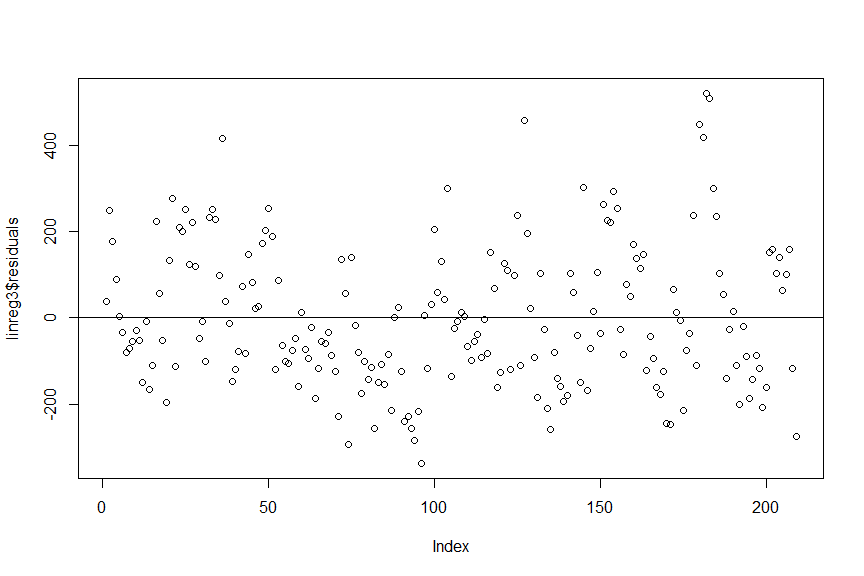
If there is a trend in the dataset then it is not stationary. However we removed the trend with the regression earlier, so we are pretty sure that at least there is no linear trend left.

# regress residuals against time  
residreg <- lm(linreg3$residuals ~ t)   
  
  
# view output of regression   
summary(residreg)

##   
## Call:  
## lm(formula = linreg3$residuals ~ t)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -337.25 -117.84 -29.87 105.71 520.32   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)  
## (Intercept) -7.928e-14 2.297e+01 0 1  
## t 5.865e-16 1.897e-01 0 1  
##   
## Residual standard error: 165.4 on 207 degrees of freedom  
## Multiple R-squared: 3.857e-32, Adjusted R-squared: -0.004831   
## F-statistic: 7.984e-30 on 1 and 207 DF, p-value: 1

We could plot this regression against our residuals.

# plot residuals   
plot(linreg3$residuals)   
  
# add line from regression   
abline(residreg)



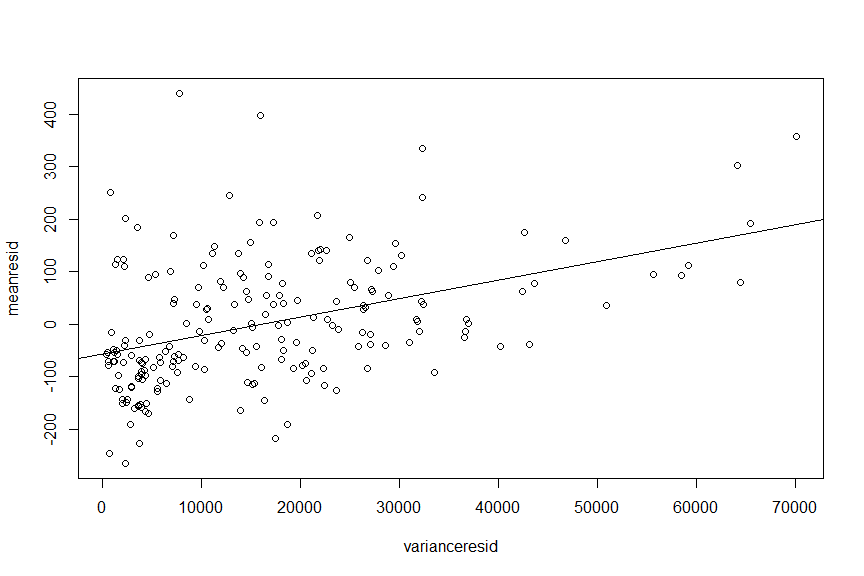
We can see that the coefficient is close to 0 and that slope is not statistically significant from a straight line.

It would also be possible to investigate that the variance is constant by comparing variance against the mean. We could use a 5-week interval for this.

# calculate variance and mean means  
varianceresid <- rollapply(linreg3$residuals, width = 5, FUN = var, fill = NA)  
meanresid <- rollapply(linreg3$residuals, width = 5, FUN = mean, fill = NA)  
  
  
# linear regression  
  
residreg\_mean <- lm(meanresid ~ varianceresid)  
  
# view linear outcome   
summary(residreg\_mean)

##   
## Call:  
## lm(formula = meanresid ~ varianceresid)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -222.56 -72.57 -20.11 52.84 468.52   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) -5.678e+01 1.143e+01 -4.966 1.44e-06 \*\*\*  
## varianceresid 3.507e-03 5.323e-04 6.588 3.76e-10 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 108.3 on 203 degrees of freedom  
## (4 observations deleted due to missingness)  
## Multiple R-squared: 0.1761, Adjusted R-squared: 0.1721   
## F-statistic: 43.4 on 1 and 203 DF, p-value: 3.761e-10

# plot regression   
plot(varianceresid, meanresid)  
abline(residreg\_mean)

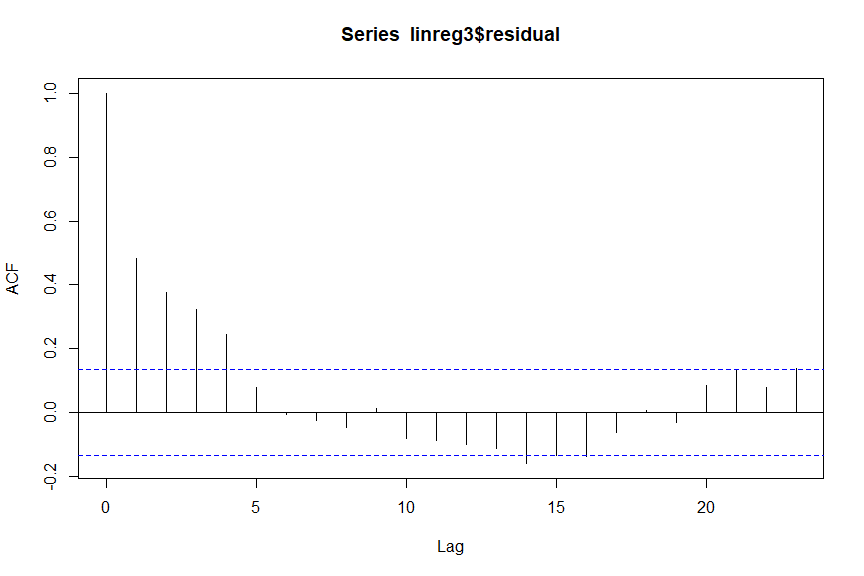


While there are some outliers, there are very few. Although signficant, the minimal slope of this regression shows that we can be happy that the residuals are stationary.

## Autocorrelation

We can check for autocorrelation using the *acf* function. This function also creates a plot.

# estimate autocorrelation without plotting  
acplotresid <- acf(linreg3$residual)



The ACF plot shows us that our data are highly correlated at early lags, which is expected from surveillance data. We also see that a seasonal pattern still remains in the residuals, even if it is not significant. This indicates that it may be worth checking the residuals of the model with the further seasonal term (of 26 weeks). And in fact, what we see that the normality of residuals is not improved in the model with 3 seasonal terms. The data remain stationary (no increasing variance with increasing mean). However the ACF plot is improved. There is still autocorrelation among the first few lags, but it is lower and we see less of a seasonal pattern remaining in the residuals.

# Question 6

Using the model containing a linear trend and two seasonal periods at 52 and 17.33 weeks, model the data including 95% prediction interval and graph them.

In a second step, downweight outliers that are above 2 standard deviations of the model. In public health models that downweight outliers a variety of weights are used. Here we will use a very simple weight of 1/(square of the standardised residuals). This means that the higher the outlier, the lower the weight will be assigned to them. Create the new model and 95% detection intervals and graph them.

How do you interpret these models and what do you conclude?

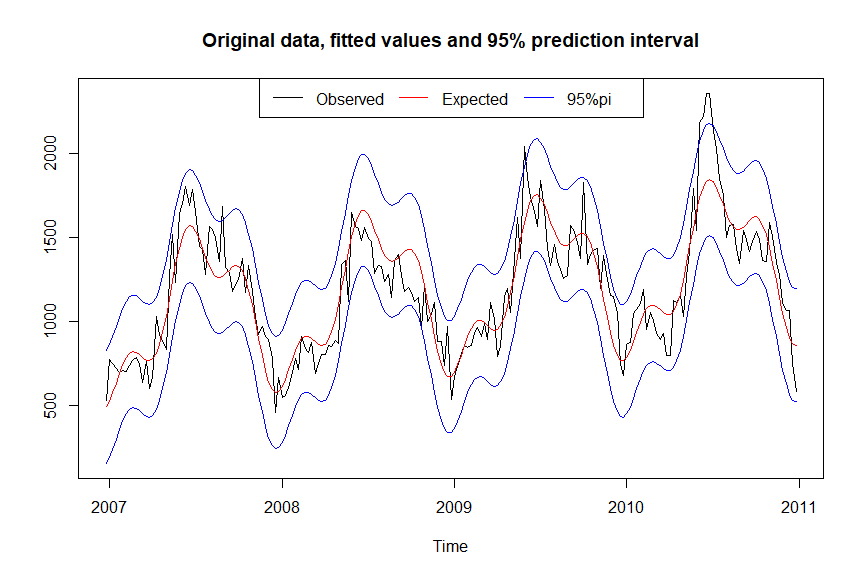
# Help Question 6

Retrieving prediction intervals is possible using the *predict* function on your regression of choice. This can then be plotted along with your time series data as previously.

# retrieve your 95% prediction intervals from regression  
pred\_int <- predict(linreg3, interval = "prediction", level = 0.95)

## Warning in predict.lm(linreg3, interval = "prediction", level = 0.95): predictions on current data refer to \_future\_ responses

# plot   
ts.plot(campy\_ts, pred\_int,  
 col = c("black", "red", "blue", "blue"),  
 main = "Original data, fitted values and 95% prediction interval")  
  
legend("top",   
 lwd = 1,   
 horiz = TRUE,   
 col = c("black", "red", "blue"),  
 legend = c("Observed", "Expected", "95%pi"))



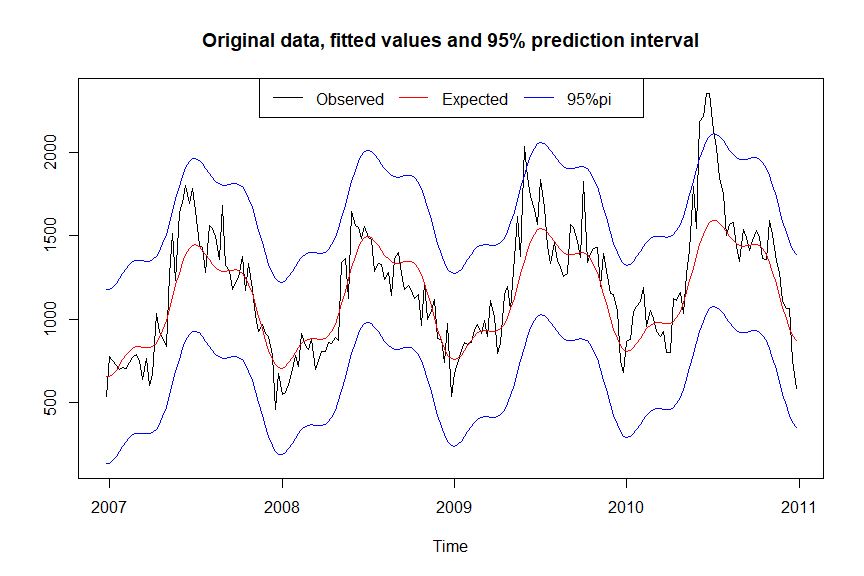
So we can see that at certain points the data exceed the intervals. These outliers may represent outbreaks. However we have also been including these data points to estimate the model and the prediction intervals to determine what is expected. This will artificially inflate our prediction intervals.

We could take these outliers into account when estimating our model and create 95% detection intervals around it. There are very many ways of taking outliers into account. Here we will use a simple method of downweighting outliers. This is an empirical approach and the weight given to the outliers can be adjusted according to how the model performs.

This approach will be done in several steps:

* We will identify outliers as those exceeding the model by 2 standard deviations. For this we will standardise our residuals. This means that we subtract the mean from the residual and divide this by the standard deviation of the residuals.
* We will create a weight that will be defined as one for data points that are not outliers, and as 1/((standardised residual)2) for outliers.
* We will create a new variable for counts that will contain the original counts, but with the outliers downweighted. These outliers will be defined as 2 standard deviations above the model plus the exceedance above this weighted by the weighting factor.

# extract the standard deviations for each point  
stdres <- residuals(linreg3) / sigma(linreg3)  
  
# create weights   
 # if standard deviation greater than 2 then  
 # 1 / stdres^2 otherwise 1  
wts <- ifelse(stdres > 2,   
 1 / (stdres^2), 1)  
  
  
# modify counts   
campy\_tswtd <- campy\_ts \* wts  
  
  
# run linear regression for counts over time  
linreg3wtd <- lm(campy\_tswtd ~ t + sin52 + cos52 + sin17 + cos17)  
  
  
# retrieve your 95% prediction intervals from regression  
pred\_intwtd <- predict(linreg3wtd, interval = "prediction", level = 0.95)  
  
# plot   
ts.plot(campy\_ts, pred\_intwtd,  
 col = c("black", "red", "blue", "blue"),  
 main = "Original data, fitted values and 95% prediction interval")  
  
legend("top",   
 lwd = 1,   
 horiz = TRUE,   
 col = c("black", "red", "blue"),  
 legend = c("Observed", "Expected", "95%pi"))



The difference between the model not taking and taking outliers into account is not large in this example. Depending on your objectives for time series analysis, you can adjust the weighting and the definition of what is an outlier (e.g. from 2 standard deviations to 1.5 standard deviations). This can lead to a more or less sensitive model.

# Question 7

Using the *surveillance package*, try other detection algorithms and compare how they perform (sensitivity, specificity, positive predictive value) by varying their parameters:

* The Early Aberration Detection System (EARS) used by CDC
* The CDC algorithm used by CDC
* The Farrington method used by PHE
* The CUSUM method
* The GLRNB method

# Help Question 7

## Creating a *sts* object for using surveillance package

This section is based on *Salmon, Schumacher and Höhle, 2014*. They provide a good overview of the surveillance package and the detection algorithms they use. The first step is to create a surveillance time series object (*sts*) based on our weekly aggregates, similarly to how we made a time series object previously.

# define surveillance time series object  
campy\_sts <- sts(observed = campy\_week$count,  
 start = c(2007, 1),   
 freq = 52)

## EARS C1 algorithm

This method is suitable when few historical data are available, but does not adjust for trend, seasonality, nor does it take into account past outbreaks. The baseline is the 7 time points before the assessed time point . The expected value is the mean of the baseline. The method is based on a statistic called defined as:

where:

Under the null hypothesis of no outbreak, we assume that is normally distributed with a mean of 0 and standard deviation of 1. The upper bound is defined by a confidence interval of for the mean:

An alarm is raised if . To apply the detection algorithm in R, you first need to construct a control slot that defines some of the parameters (range, method, the alpha error, etc…) of the algorithm.

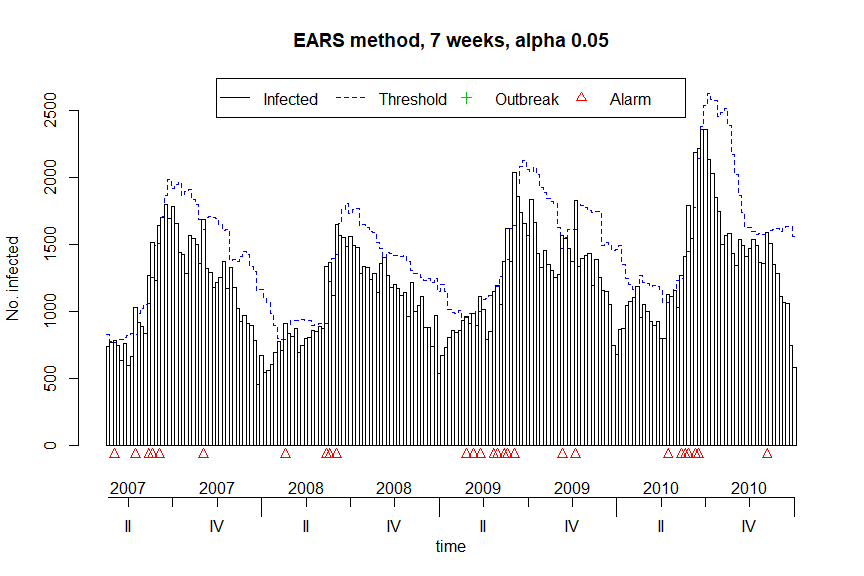
# define control  
ctrl <- list(range = 8:209,   
 method = "C1",  
 alpha = 0.05)

We can then use the *earsC* function from the surveillance package.

# apply EARS C1 algorithm   
earmethod <- earsC(campy\_sts, control = ctrl)

You can also plot the outcome of this algorithm.

# plot your counts with algorithm applied   
plot(earmethod,   
 main = "EARS method, 7 weeks, alpha 0.05",  
 legend.opts = list("top", horiz = TRUE))

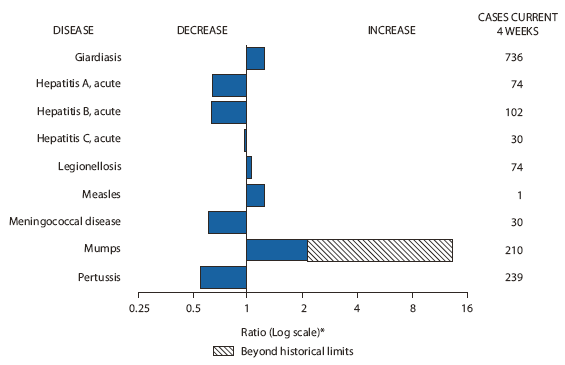


# CDC algorithm - Historical Mean Method (Stroup et al, 1989)

This signal detection method is used on a weekly basis by CDC (see example of output below). It is based on the totals from previous, comparable, and subsequent 4-week periods for the past 5 years. In our case, we cannot go as far as 5 years in the past; we will build our baseline using the past 2 years. This method does not adjust for trend and does not adjust for past outbreaks.

When alpha is set to 0.05, the historical limit is computed as:

Selected notifiable disease reports, US, comparison of provisional 4-week totals, March 6 2010 with historical data



* Ratio of current 4-week total to mean of 15 4-week totals (from previous, comparable, and subsequent 4-week periods for the past 5 years). The point where the hatched area begins is based on the mean and two standard deviations of these 4-week totals.

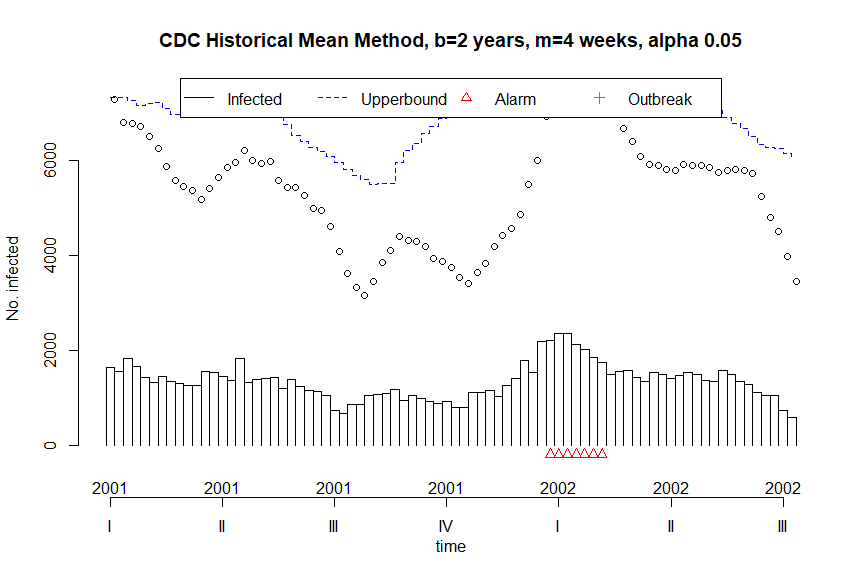
To apply the CDC detection algorithm we need to create *disProg* object from our *sts* object.

# create a disprog object   
campy\_stsd <- sts2disProg(campy\_sts)

# apply the cdc algorithm   
cdcmethod <- algo.cdc(campy\_stsd,   
 control = list(range = 130:209, b = 2, m = 4, alpha = 0.05))

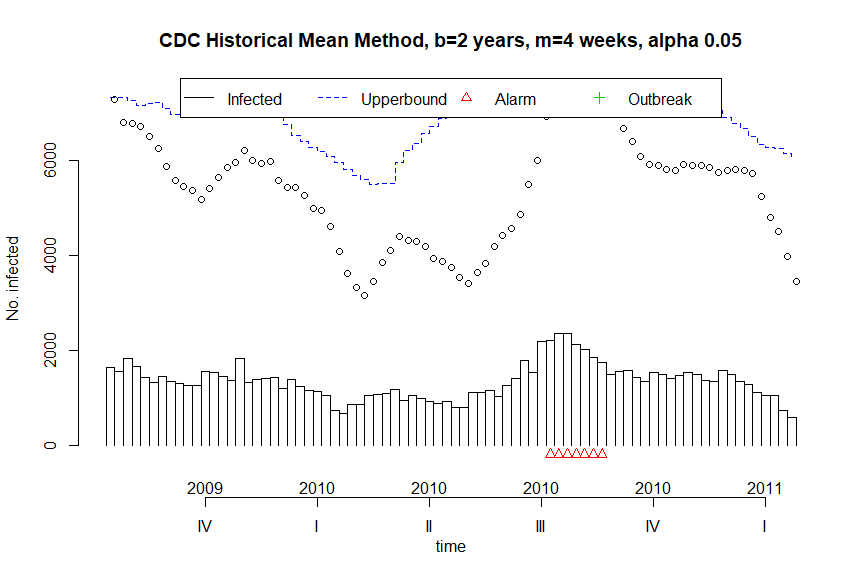
You can also plot the outcome of this algorithm.

plot(cdcmethod,  
 main = "CDC Historical Mean Method, b=2 years, m=4 weeks, alpha 0.05",  
 legend.opts = list(x = "top", horiz = TRUE))



It is clear from this plot that the x-axis labels of time are not correct. This appears to be an in-built error of the package and can be corrected as below.

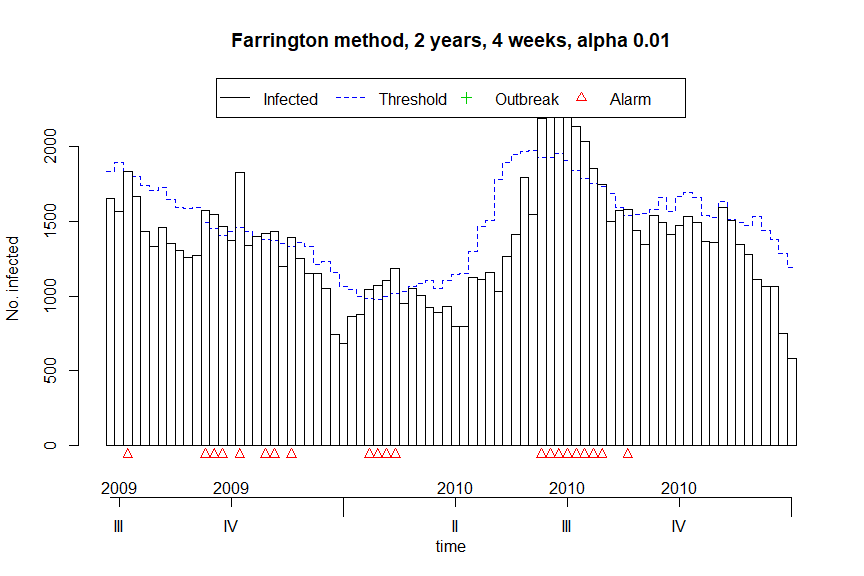
# define the start week and year for your plot labels  
 # nb. %% is the modulus  
syear <- floor(cdcmethod$disProgObj$start[1] + 130 / cdcmethod$disProgObj$freq)  
sweek <- -((cdcmethod$disProgObj$start[2] + 130) %% cdcmethod$disProgObj$freq)  
  
# plot your counts with algorithm applied   
plot(cdcmethod, startyear = syear, firstweek = sweek,  
 main = "CDC Historical Mean Method, b=2 years, m=4 weeks, alpha 0.05",  
 legend.opts = list(x = "top", horiz = TRUE))



# Farrington algorithm - used by PHE

This method fits an over-dispersed Poisson generalized linear model with log link to the reference data. It takes seasonality into account by using a subset of the available data as reference data for fitting the model: time points centered around the time point located years before , amounting to a total reference values. It is better to exclude the most recent weeks before from the baseline in order to avoid reducing sensitivity when an outbreak has started recently before (pastWeeksNotIncluded). The time trend is included only when significant at a chosen level pThresholdTrend, when there are more than three years reference data and if not overextrapolation occurs because of the time trend. Past outbreaks are reweighted based on their Anscombe residuals. If the Anscombe residuals of a count is higher than weightsThreshold it is reweighted accordingly in a second fitting of the linear model.

# define control  
ctrl <- list(range = 130:209,  
 b = 2, w = 4,   
 weightsThreshold = 2.56,  
 pastWeeksNotIncluded = 3,  
 pThresholdTrend = 0.05,  
 thresholdMethod = "delta")  
  
# apply farrington flexible method   
farringtonmethod <- farringtonFlexible(campy\_sts, ctrl)  
  
# plot your counts with algorithm applied   
plot(farringtonmethod,   
 main = "Farrington method, 2 years, 4 weeks, alpha 0.01",  
 legend.opts = list("top", horiz = TRUE))



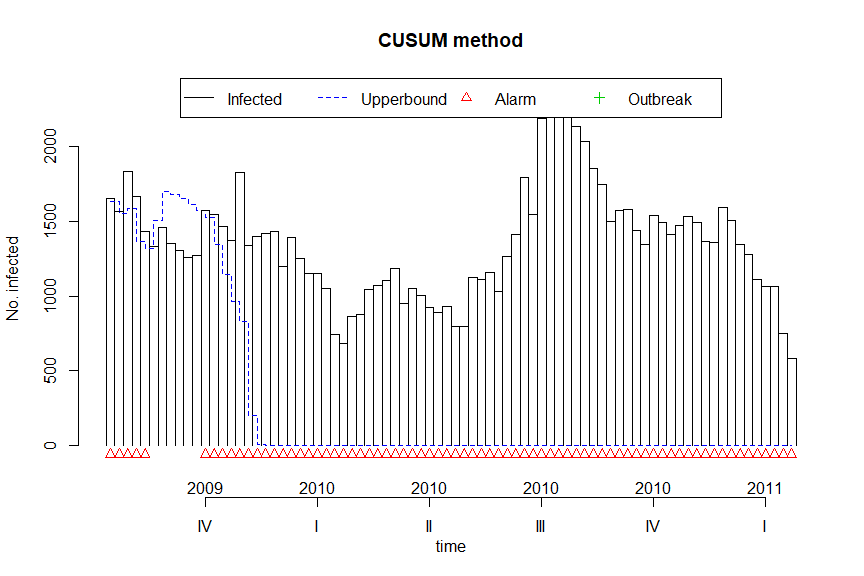
# CUSUM – used to detect sustained shift in the underlying process

The CUSUM algorithm is a technique of statistical process control (SPC). It assumes a two-state situation, so-called in-control and out-of-control (i.e. an outbreak) and we assume different distribution of case counts in both states. The aim of the CUSUM is to find the point where the distribution changes. As we are dealing with a time series with underlying seasonality we need to define m as glm to control for it.

# define control range and method  
ctrl <- list(range = 130:209,  
 m = "glm")  
  
# apply cusum algorithm   
cusummethod <- algo.cusum(campy\_stsd, control = ctrl)

Here too there is an error with axis labels and this can be corrected as below before plotting.

# correct start week and year for axis labels  
syear <- floor(cdcmethod$disProgObj$start[1] + 130 / cdcmethod$disProgObj$freq)  
sweek <- -((cdcmethod$disProgObj$start[2] + 130) %% cdcmethod$disProgObj$freq)  
  
# plot with algorithm applied  
plot(cusummethod, startyear = syear, firstweek = sweek,  
 main = "CUSUM method",  
 legend.opts = list(x = "top", horiz = TRUE))

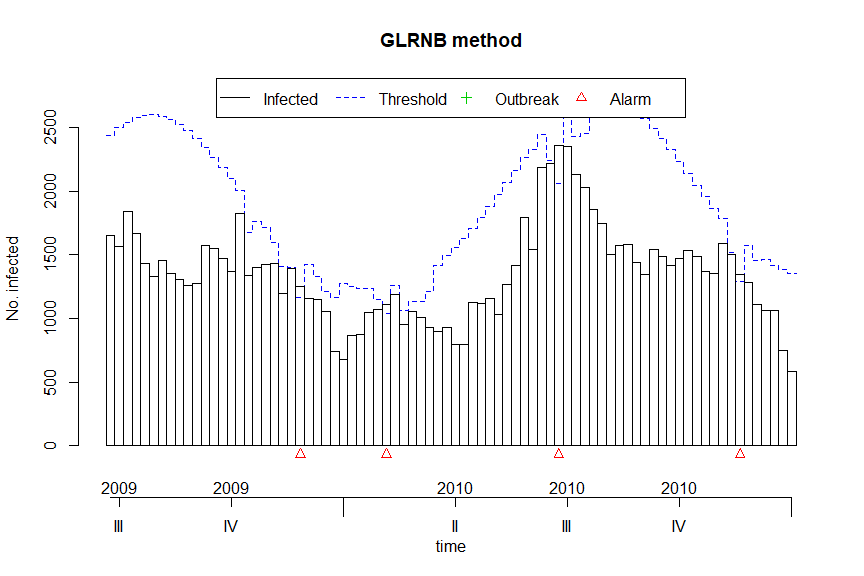


# GLRNB method

This method is a synthesis of traditional methods like the Farrington algorithm and statistical process control algorithms like CUSUM. The idea is to combine the possibility to detect sustained shifts with the ability of taking overdispersion or seasonality into account. You need to split your data into two phases, where phase one is used for fitting a GLM model. Technically, in the glrnb functions, the time period defined in the range option will be phase 2 and everything before it will be considered phase 1.

# define your period of interest  
phase2 <- 130:209  
  
# define control options  
ctrl <- list(range = phase2, c.ARL = 4, theta = log(1.4), ret = "cases",  
 mu0 = list(S = 1, trend = TRUE, refit = FALSE))  
  
# apply the algorithm  
glrnbmethod <- glrnb(campy\_sts, control = ctrl)

# plot with algorithm applied  
plot(glrnbmethod, main = "GLRNB method",  
 legend.opts = list("top", horiz = TRUE))



# Comparing algorithms

It is then possible to compare the various algorithms using the *algo.compare* function.

# list algorithms used   
campy\_algos <- list(earmethod, cdcmethod, cusummethod, farringtonmethod, glrnbmethod)  
  
# compare algorithms  
algo.compare(campy\_algos)

## TP FP TN FN sens spec dist mlag  
## [1,] 0 27 175 0 NaN 0.8663366 NaN 0   
## [2,] 0 7 73 0 NaN 0.9125 NaN 0   
## [3,] 0 74 6 0 NaN 0.075 NaN 0   
## [4,] 0 21 59 0 NaN 0.7375 NaN 0   
## [5,] 0 4 76 0 NaN 0.95 NaN 0

# Appendix

## Setting system language to english

The following code changes your R system locales to English. This is only done for the current session, so each time you open a new R session you would need to set it to English. For a list of language options and the strings to enter see the [windows website](https://docs.microsoft.com/en-us/cpp/c-runtime-library/language-strings?view=vs-2017).

Sys.setlocale("LC\_ALL", "english\_UK")

In order to permanently change your language settings for R (so that you do not need to reset this each time) you have to change your startup settings. To do this (on Windows OS), copy and paste the above line of code in to your *Rprofile.site*, which you would find in the R > etc folder. Normally this would be in programme files but it depends where you installed R.