Meeting Notes

Martin Schobben & Ilja Kocken

May 22, 2019

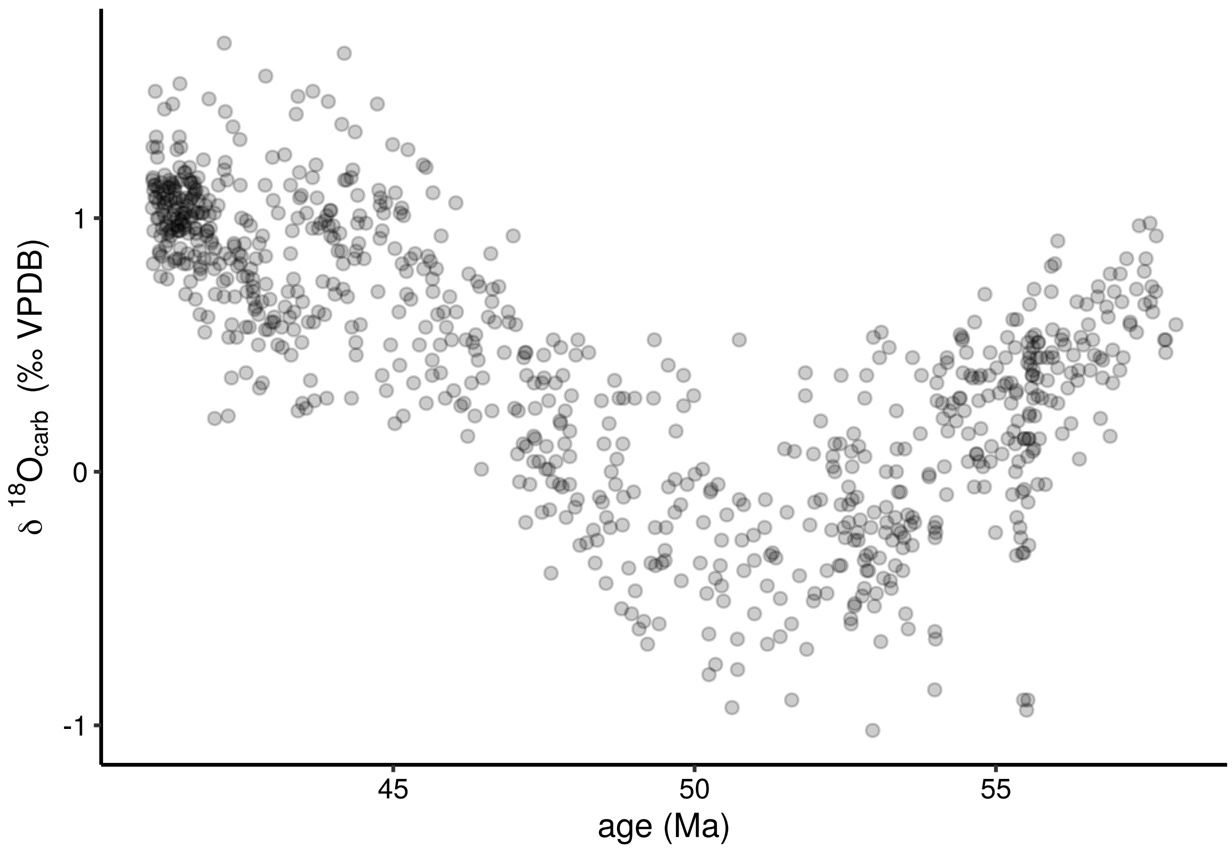
Timeseries analysis? Smoothing noisy datasets in the time domain

During this meetings we discussed the multiple options of smoothing your data (e.g. chemical proxies in palaeoclimate research) which is plotted along a time axis. Note that there are possibility many more options then the ones we show here. For the following exercises we use the packages pangaear, ggplot2, dplyr, tidyquant, caTools (base) and SiZer, where the first one is for data extraction from the PANGAEA database https://pangaea.de/, the following two for plotting and data transforming, and the last three plus ggplot2 for the actual statistics of smoothing.

Our example dataset (Zachos et al. 2008)

Firstly, we will download the Zachos et al. 2008 dataset from the PANGAEA database, and we'll plot the data for a first inspection with a ggplot.

```
p <-  ggplot(zachos, aes(x = age, y = d18O))
p <- p + geom_point(alpha = .2)
p <- p  + ylab(expression(delta^{18}*O[carb]~"(\u2030 VPDB)"))
p <- p  + xlab("age (Ma)")
p
```



Although we can see a clear trend in this oxygen isotope dataset, there is considerable scatter, so we might want to clarify the long term. We want to smooth our timeseries. The most familiar option is perhaps the usage of a running (rolling) mean or median. We'll show two options.

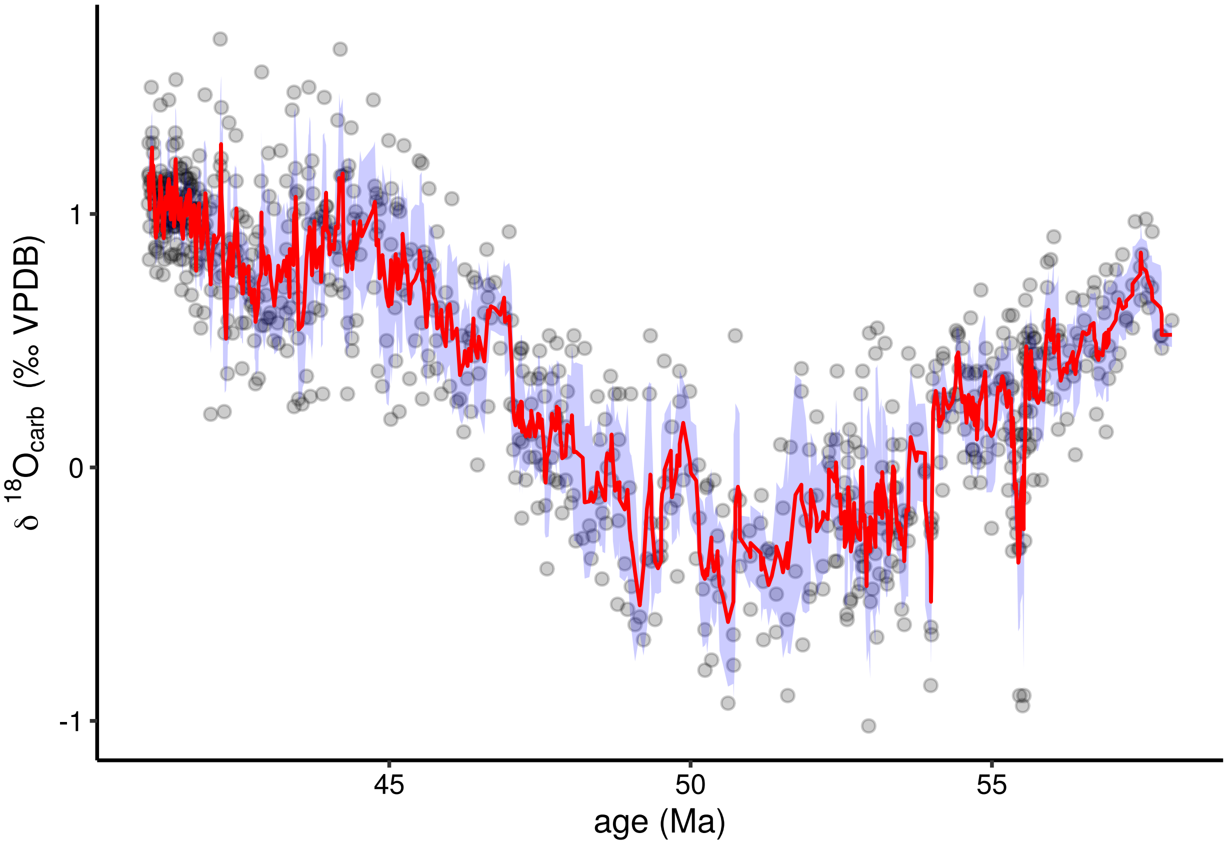
Option 1 running mean with caTools

We create new columns with `mutate()` and the usage of pipes. We transform the raw d18O data to running mean with `runmean()` and the running CI (95%) with `runsd()` and the general statistical treatment to obtain the CI (95%).

```r
# option 1 caTools

# number of observations
k.sm <- 5

zachos <- zachos %>%

# running mean
  mutate(av.d18O = runmean(d18O, k = k.sm),
# running CI lower bound
         ci.min.d18O = av.d18O - (runsd(d18O, k = k.sm) /
                                    sqrt(k.sm) * 1.96 ),
# running CI upper bound
         ci.max.d18O = av.d18O + (runsd(d18O, k = k.sm) /
                                    sqrt(k.sm) * 1.96))
```

Here, we plot the confidence interval and running mean. We recreate the previous plot, but now with the new dataframe that includes the running mean (and the CI) as a new column.

```r
p <-  ggplot(zachos, aes(x = age, y = d18O))
p <- p + geom_point(alpha = .2)
p <- p  + ylab(expression(delta^{18}*O[carb]~"(\u2030 VPDB)"))
p <- p  + xlab("age (Ma)")

# confidence interval
p <- p + geom_ribbon(aes(ymax = ci.max.d18O, ymin = ci.min.d18O),
                     fill = "blue", alpha = 0.2)

p <- p + geom_line(aes(y = av.d18O), col="red")

p
```

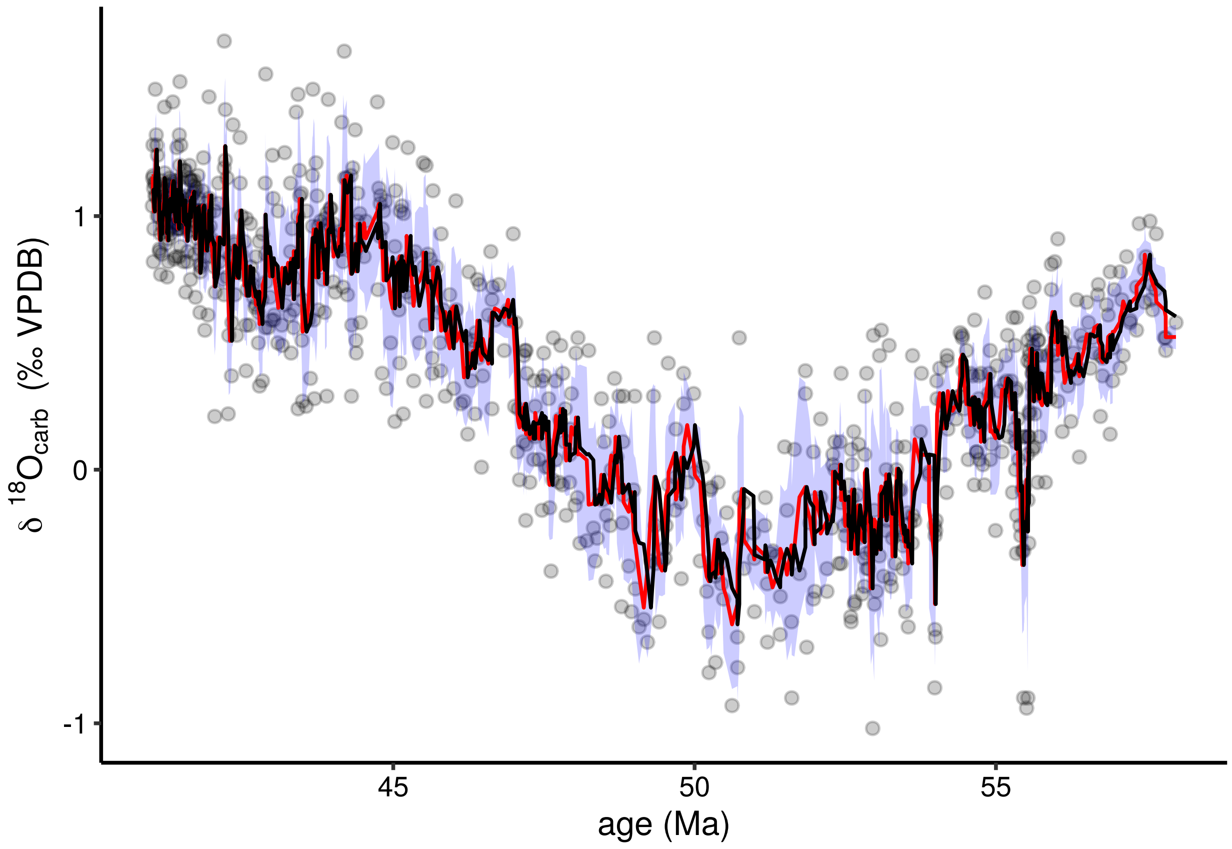


Option 2 running mean with tidyquant

Option is the quickest means of plotting a running mean, and can be called as a `geom` that can be put on top of the plot. Here we plot both the previous option and the tidyquant option. Note that we see some differences, these are likely include in the other parameters that define the statistics behind the calls `runmean()` and `geom_ma()`.

```r
# option 2 running mean with tidyquant

p <-  p + geom_ma(n = k.sm, col = "black", linetype = 1)
```

```
p
```



Moving windows

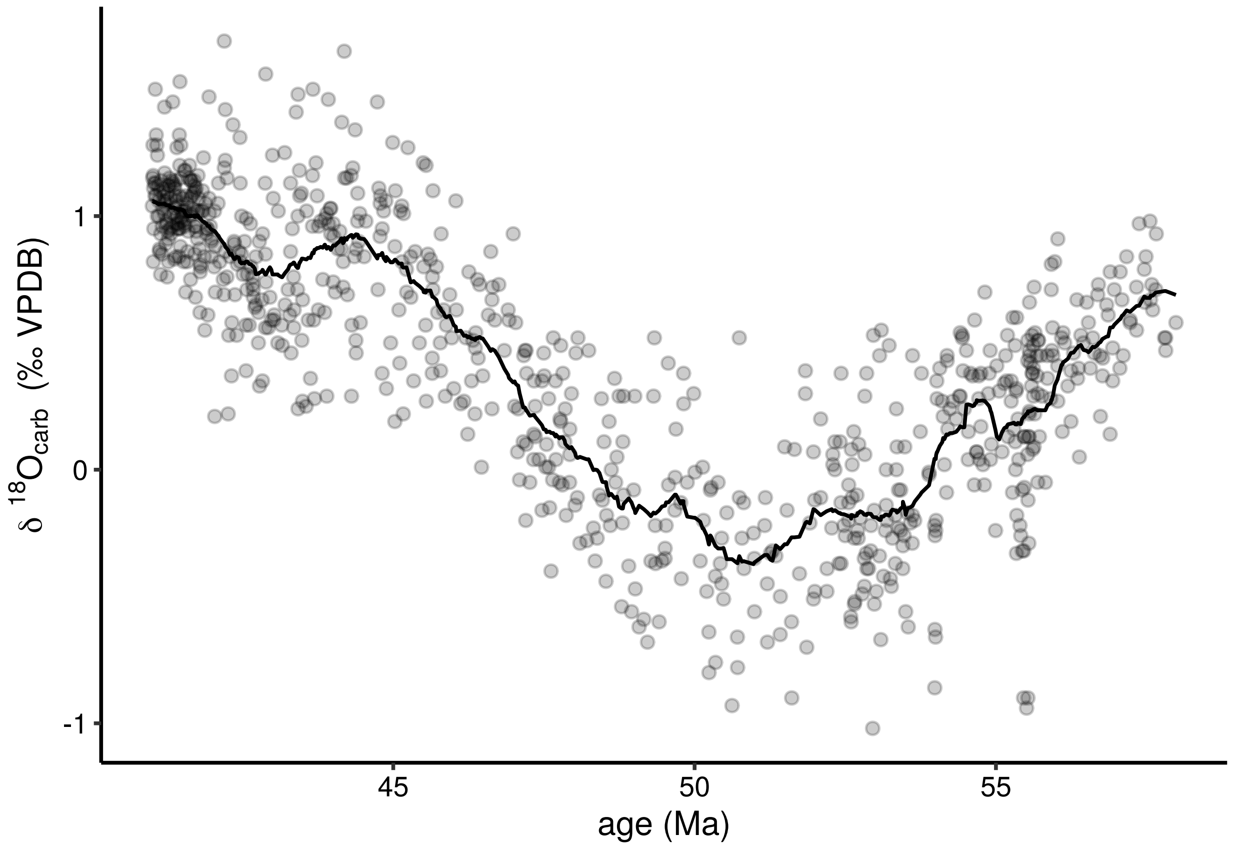
Instead of using the nearest neighbours for the statistics, we can also make use of sliding window which we can fix to the independent variable (the time domain), a.k.a kernel regression. For this, we use the base function `ksmooth()`. We create a new vector in which we store the model fit of the moving window. Note that we can change the kernel (or weighing function) to, for instance, from a box (equal weight for all values) to a normal distribution (weight tapering of to the edges of the window). For reference look at the following website: https://rafalab.github.io/dsbook/smoothing.html, and the awesome animations, that were also the inspiration for the here presented exercises.

```r
# span of the window
s.sm <-  1 # million year

fit.mov <- with(zachos, ksmooth(age, d18O, x.points = age, kernel = "box", bandwidth = s.sm))

zachos <- zachos %>% mutate(smooth.mov = fit.mov$y)
```

We plot the moving window mean curve again to first basic graph of the Zachos et al 2008 dataset.

```r
p <-  ggplot(zachos, aes(x = age, y = d18O))
p <-  p + geom_point(alpha = 0.2)
p <- p  + ylab(expression(delta^{18}*O[carb]~"(\u2030 VPDB)"))
p <- p  + xlab("age (Ma)")
p <-  p + geom_line(aes(y = smooth.mov))
```

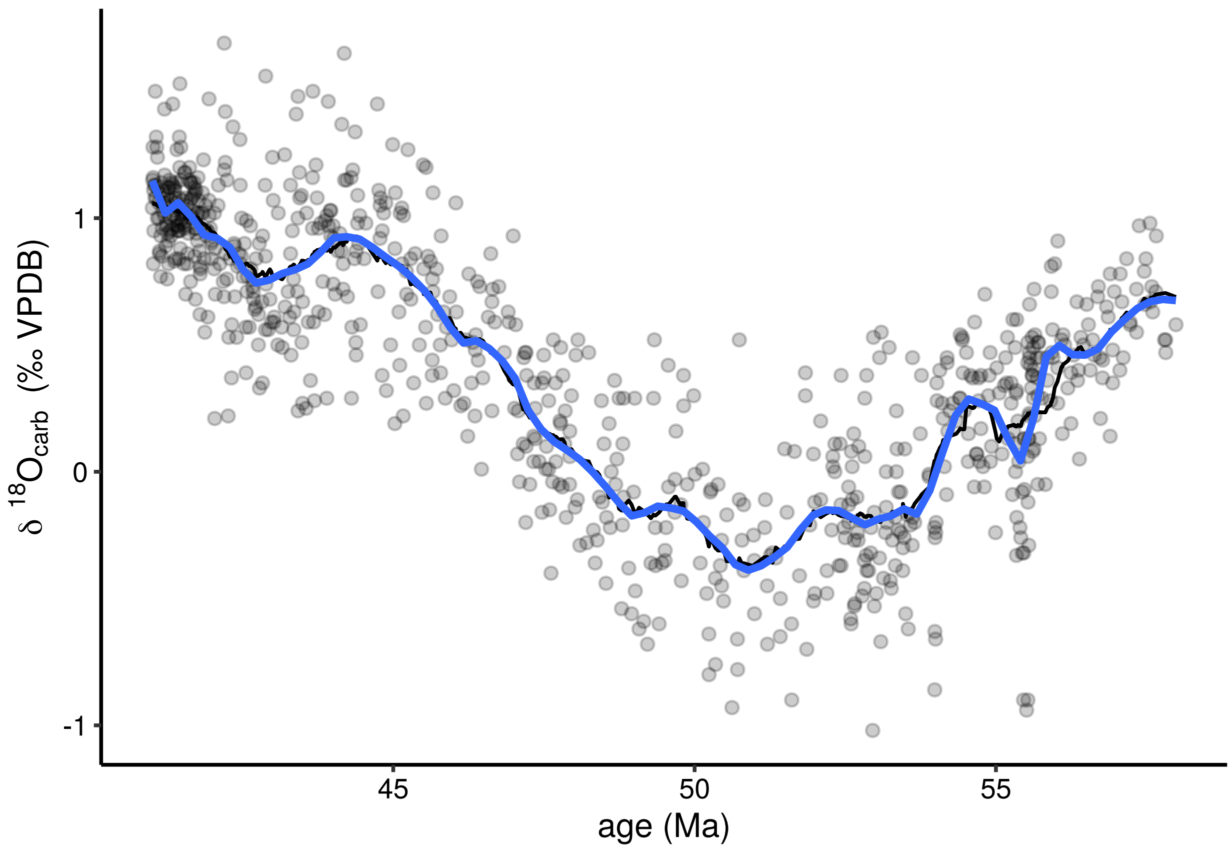
```
p
```



Local regression analysis (nearest neighbour)

We'll present now two options for performing local regression analysis. Local regressions, instead of a global regression, perform a localised regression analysed whilst sliding through the data. In this case whilst sliding along the time axis. These localised regressions are then spliced together with a interpolation scheme. For more info behind the statistics look in the TeachingDemos package (`TeachingDemos::loess.demo(zachos$age, zachos$d18O)`). Our first option involves the usage of another geom which is included in ggplot2, and is called `geom_smooth`. This geom has several basic stats included, among which a LOESS fit. This fit uses nearest neighbouring datapoints for it's windows size (e.g. the windowsize is dynamic and can jump across gaps). This windowsize is set with the parameter span, which defines the relative proportion of the total dataset to be included (10% in our example).

```
# span of the nearest neighbour window
s.sm <-  0.1

p <-  p + geom_smooth(method = "loess", span = s.sm, se = FALSE)

p
```

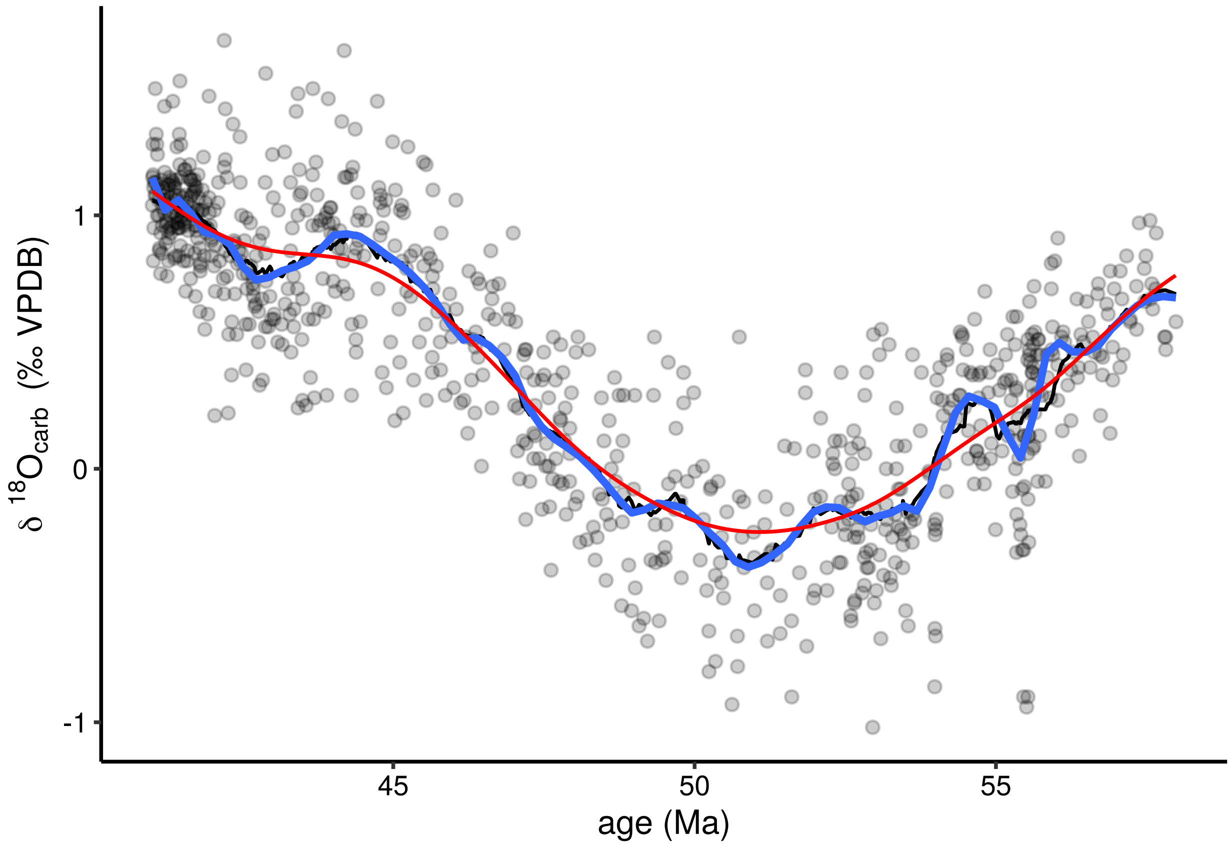


Local regression analysis (fixed binwidth)

The second option includes a LOESS fit but then with a fixed binwidth. So, here we can fix the span to the independent variable; the time domain. For this we use the package SiZer, and we create a new dataframe that will contain the LOESS fit. This function is different in that we have to create a grid along which the window slides.

```r
# span of the window
s.sm <- 1 # miilion year

# grid
grid <-  seq(min(zachos$age), max(zachos$age), 0.01)

# the fit
fit.loess <-  with(zachos, locally.weighted.polynomial(age, d18O, h = s.sm,
                                                       x.grid = grid,
                                                       kernel.type = "Normal"))

fit.loess <- data.frame(age = grid,
                        smooth.loess = fit.loess$Beta[1,])


# adding the fit to the plot
p <- p + geom_line(data = fit.loess, aes( x =age, y = smooth.loess), col ="red")
```

```
p
```



Interactive plots with plotly

The next chunk shows how you can make the previous plot interactive by creating html that allows manipulation of the plot by just clicking on features. Do not use this package within your Rmarkdown

```
library(plotly)

ggplotly(p, dynamicTicks = TRUE)

toWebGL(ggplotly(p, dynamicTicks = TRUE))
```