Processing Script

Jessica Karch

December 3, 2018

Introduction

This document presents an exemplar for how to process pupillometric data. Processed data have been published in *Eye Tracking for the Chemistry Education Researcher* (DOI:10.1021/bk-2018-1292.ch008), and submitted for journal publication as of December 2018. The following workflow has been developed in RStudio, using packages from the Tidyverse, among others.

A full list of packages in the order they should be loaded are listed in the below code chunk:

```
library(readxl)
library(plyr) #we'll use a `plyr` function later down the pipeline; however, because it
# and dplyr have overlapping functions we must call it first
library(dplyr)
library(zoo)
library(tidyr)
library(ggplot2)
```

Calculate and Subtract Baseline

To find pupil dilations, a baseline value should be subtracted from each value in your data set. First, we load the data. The data sets were collected in Tobii Studio using an X2-60 eye tracker. The full datasets were exported from the eye tracker with no pre-processing in Excel.

```
# load data
p1_raw <- readxl:::read_xlsx("data/n001_q6.xlsx") # This is the trial data set
p1_bl <- readxl:::read_xlsx("data/Josibel_8 Chem questions 2016 new_N001_Question 6 XX.xlsx") #Baseline

#Inspect the shape of the data sets
head(p1_raw)
```

```
## # A tibble: 6 x 81
##   ExportDate ParticipantName RecordingDuration FixationFilter  SegmentName
##   <chr>      <chr>                       <dbl> <chr>           <chr>
## 1 7/31/2017  N001                       136885 Tobii Fixation~ Question 6
## 2 7/31/2017  N001                       136885 Tobii Fixation~ Question 6
## 3 7/31/2017  N001                       136885 Tobii Fixation~ Question 6
## 4 7/31/2017  N001                       136885 Tobii Fixation~ Question 6
## 5 7/31/2017  N001                       136885 Tobii Fixation~ Question 6
## 6 7/31/2017  N001                       136885 Tobii Fixation~ Question 6
## # ... with 76 more variables: SegmentStart <dbl>, SegmentEnd <dbl>,
## #   SegmentDuration <dbl>, SceneName <lgl>, SceneSegmentStart <lgl>,
## #   SceneSegmentEnd <lgl>, SceneSegmentDuration <lgl>,
## #   RecordingTimestamp <dbl>, LocalTimeStamp <chr>,
## #   EyeTrackerTimestamp <dbl>, MouseEventIndex <lgl>, MouseEvent <lgl>,
## #   `MouseEventX (ADCSpx)` <lgl>, `MouseEventY (ADCSpx)` <lgl>,
## #   `MouseEventX (MCSpx)` <lgl>, `MouseEventY (MCSpx)` <lgl>,
## #   KeyPressEventIndex <dbl>, KeyPressEvent <chr>, FixationIndex <dbl>,
## #   SaccadeIndex <lgl>, GazeEventType <chr>, GazeEventDuration <dbl>,
```

```
## #   `FixationPointX (MCSpx)` <chr>, `FixationPointY (MCSpx)` <chr>,
## #   SaccadicAmplitude <dbl>, AbsoluteSaccadicDirection <dbl>,
## #   RelativeSaccadicDirection <dbl>, `AOI[questionA]Hit` <lgl>,
## #   `AOI[diagram]Hit` <lgl>, `AOI[questionB]Hit` <lgl>,
## #   `AOI[equation]Hit` <lgl>, `AOI[answers]Hit` <lgl>,
## #   `AOI[questionA]Hit__1` <dbl>, `AOI[diagram]Hit__1` <dbl>,
## #   `AOI[questionB]Hit__1` <dbl>, `AOI[answers]Hit__1` <dbl>,
## #   `AOI[questionTF]Hit` <lgl>, `AOI[answersTF]Hit` <lgl>,
## #   `AOI[questionA]Hit__2` <lgl>, `AOI[equation]Hit__1` <lgl>,
## #   `AOI[questionB]Hit__2` <lgl>, `AOI[answers]Hit__2` <lgl>,
## #   GazePointIndex <dbl>, `GazePointLeftX (ADCSpx)` <dbl>, `GazePointLeftY
## #   (ADCSpx)` <dbl>, `GazePointRightX (ADCSpx)` <dbl>, `GazePointRightY
## #   (ADCSpx)` <dbl>, `GazePointX (ADCSpx)` <dbl>, `GazePointY
## #   (ADCSpx)` <dbl>, `GazePointX (MCSpx)` <dbl>, `GazePointY
## #   (MCSpx)` <dbl>, `GazePointLeftX (ADCSmm)` <dbl>, `GazePointLeftY
## #   (ADCSmm)` <dbl>, `GazePointRightX (ADCSmm)` <dbl>, `GazePointRightY
## #   (ADCSmm)` <dbl>, `StrictAverageGazePointX (ADCSmm)` <dbl>,
## #   `StrictAverageGazePointY (ADCSmm)` <dbl>, `EyePosLeftX
## #   (ADCSmm)` <dbl>, `EyePosLeftY (ADCSmm)` <dbl>, `EyePosLeftZ
## #   (ADCSmm)` <dbl>, `EyePosRightX (ADCSmm)` <dbl>, `EyePosRightY
## #   (ADCSmm)` <dbl>, `EyePosRightZ (ADCSmm)` <dbl>, CamLeftX <lgl>,
## #   CamLeftY <lgl>, CamRightX <lgl>, CamRightY <lgl>, DistanceLeft <dbl>,
## #   DistanceRight <dbl>, PupilLeft <dbl>, PupilRight <dbl>,
## #   ValidityLeft <dbl>, ValidityRight <dbl>, IRMarkerCount <lgl>,
## #   IRMarkerID <lgl>, PupilGlassesRight <lgl>
```

We see that these data sets are pretty massive, so we'll need to reduce the information included to just what we need. To calculate a single baseline value, we will average the last 400ms worth of values in the p1_bl set. We'll use the Tidyverse packages `plyr` and `dplyr` for this, as well as for subsequent processing.

First, Tobii outputs pupil sizes for both the right and left pupils. Because these should be dilating to the same extents (induced by the same underlying cognitive process), we can average these to get a single pupil value. We want to average the last 400ms of the baseline acclimation period to get a single baseline value. Because the X2-60 is a 60Hz machine, to average the last 400ms, we average the last 24 values in the data set.

Please note that in the following workflow, when a function such as `mean()` is called, `na.rm` must be set to `TRUE`. This is because eye tracking data is very messy with many missing data points; if missing (NA) values are not removed, the functions will not work as the default for these calls is `na.rm = FALSE`.

```
# Use dplyr functions in this code chunk
# Calculate the subtraction value for p1
p1_baseline <- p1_bl %>%
  rowwise() %>% # group the values in the dataframe by row index
  mutate(pupilavg = mean(c(PupilRight, PupilLeft), na.rm = TRUE)) %>% # average values from
  # left and right pupils
  ungroup() %>% # ungroup the values
  summarise(value = mean(tail(pupilavg, 24), na.rm = TRUE)) # average the last 24 values in
# the new averaged pupil column
```

We'll now subtract the calculated baseline value from the pupil diameters in our experimental data set. This yields a value for dilation. We want to retain values for each row index, because each row corresponds to a single time point. We are still using `dplyr` here.

We will also condense the data set in this step. The Excel file output by Tobii has more rows than we need. For this analysis, we only need to retain information about dilation (mm), time (ms), and whether there was a hit on an AOI or not.

```
p1 <- p1_raw %>% rowwise() %>%
  mutate(dilation = mean(c(PupilRight, PupilLeft), na.rm = TRUE) - # again, average the left
           # and right pupil values
           p1_baseline$value) %>% # subtracted the calculated baseline value from each row
  mutate(time = RecordingTimestamp - SegmentStart) %>% # this is to give time from zero in the
  # trial, as Tobii did not output time in ms
# Now subset the larger data frame so only relevant information is included
  select(time, dilation, 38:41) %>% # rows 38-41 contain information about AOIs
  ungroup()

## Check the shape of the data at this step
head(p1)
```

```
## # A tibble: 6 x 6
##    time dilation `AOI[questionA]Hit~ `AOI[diagram]Hit_~ `AOI[questionB]Hi~
##   <dbl>    <dbl>               <dbl>              <dbl>               <dbl>
## 1    16  -0.0831                   0                  1                   0
## 2    32   0.0419                   0                  1                   0
## 3    49   0.0319                   0                  1                   0
## 4    66   0.0369                   0                  1                   0
## 5    82   0.0519                   0                  1                   0
## 6    99   0.0119                   0                  1                   0
## # ... with 1 more variable: `AOI[answers]Hit__1` <dbl>
```

Process the AOIs

After this step, all we have left in the data frame is time, dilation, and binaries that indicate whether or not there was a hit at a given AOI.

Currently, the data set has 0s/1s to indicate whether there was a hit at a given AOI. However, we're ultimately interested in condensing this into one column. For each column, we will revalue and 0s and 1s as categorical variables. These categorical variables will indicate the name of the AOI. We'll use the `plyr` function to rename all of the 1s to the name of the AOI that is indicated in each column. 0s will be revalued as `NA`.

```
# rename AOIs; use [[]] because p1 is of class tibble, so double brackets are required to
# subset
# `revalue` is from the plyr library
p1[[3]] <-
  plyr:::revalue(as.character(p1[[3]]), c("1" = "questionA", "0" = NA))
p1[[4]] <-
  plyr:::revalue(as.character(p1[[4]]), c("1" = "diagram", "0" = NA))
p1[[5]] <-
  plyr:::revalue(as.character(p1[[5]]), c("1" = "questionB", "0" = NA))
p1[[6]] <-
  plyr:::revalue(as.character(p1[[6]]), c("1" = "answers", "0" = NA))
head(p1)
```

```
## # A tibble: 6 x 6
##    time dilation `AOI[questionA]Hit~ `AOI[diagram]Hit_~ `AOI[questionB]Hi~
##   <dbl>    <dbl> <chr>               <chr>              <chr>
## 1    16  -0.0831 <NA>                diagram            <NA>
## 2    32   0.0419 <NA>                diagram            <NA>
## 3    49   0.0319 <NA>                diagram            <NA>
## 4    66   0.0369 <NA>                diagram            <NA>
```

```
## 5    82   0.0519 <NA>                diagram            <NA>
## 6    99   0.0119 <NA>                diagram            <NA>
## # ... with 1 more variable: `AOI[answers]Hit__1` <chr>
```

Now we can combine all of the information about the AOIs into one column. Again, we're using functions from the library `dplyr`.

```
# combine into one column
p1 <- p1 %>%
  mutate(focus = coalesce(p1[[3]], p1[[4]], p1[[5]],  p1[[6]])) %>% # `coalesce()` combines the
  # columns such that it finds the first non-missing element. Each row index should only have 1
  # non-missing element, because if there was a hit at 1 AOI, there should be an NA value for
  # all other AOIs
  select(time, dilation, focus) # reduce the number of columns

# replace "NAs" with "other" tag to indicate that focus was on e.g. whitespace
p1$focus[is.na(p1$focus)] <- "other"

head(p1)
```

```
## # A tibble: 6 x 3
##    time dilation focus
##   <dbl>    <dbl> <chr>
## 1    16  -0.0831 diagram
## 2    32   0.0419 diagram
## 3    49   0.0319 diagram
## 4    66   0.0369 diagram
## 5    82   0.0519 diagram
## 6    99   0.0119 diagram
```

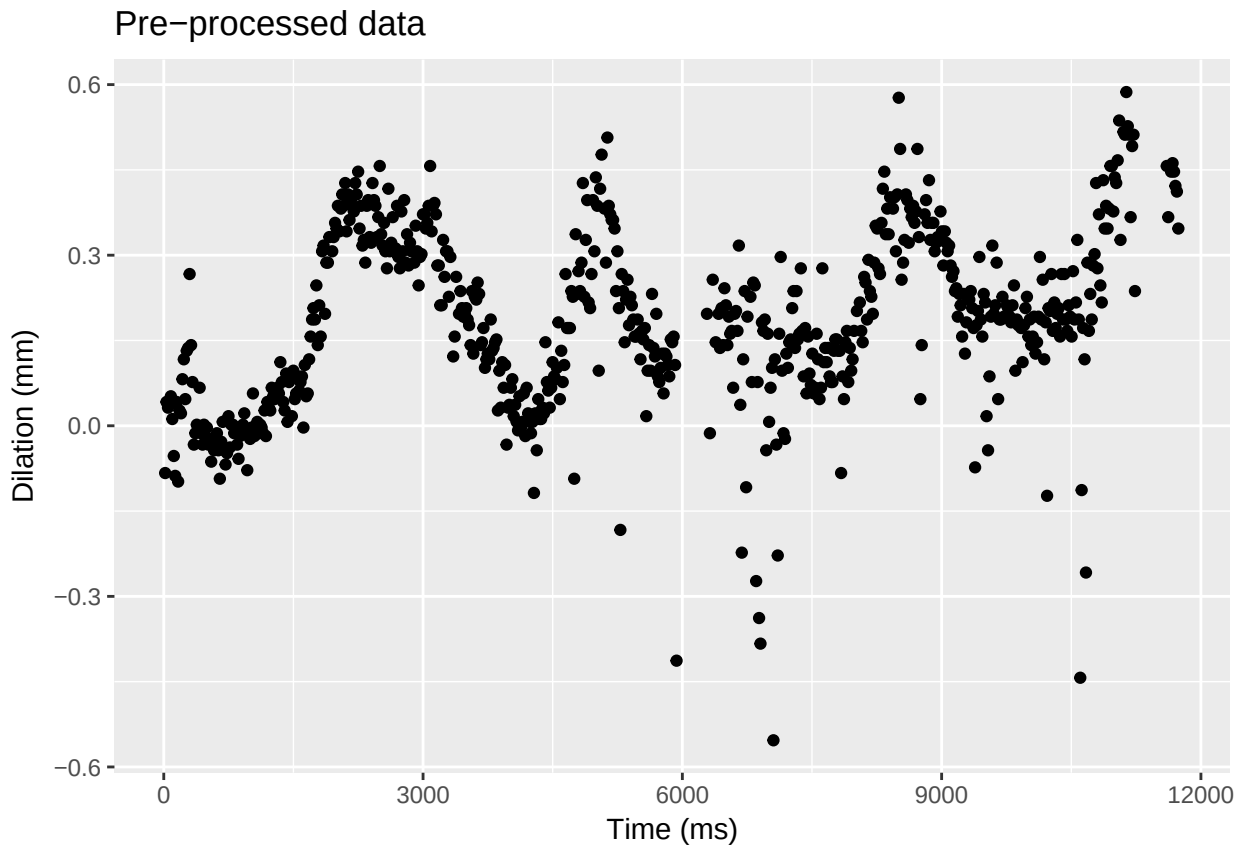
Smooth the Data

Now we will smooth the data, to try to clean up some instrument noise. Because our signal of interest is much lower frequency, smoothing the data should not lose any signal. However, we will also plot the data to visually inspect it and make sure we did not materially change the shape of the data.

```
# library(zoo)
## Now smooth the data using `rollapply` from the `zoo` package
p1 <- p1 %>%
  mutate(smooth = zoo::rollapply(p1$dilation, 3, mean, na.rm=TRUE, y = 1, partial = TRUE, fill = NA))
```

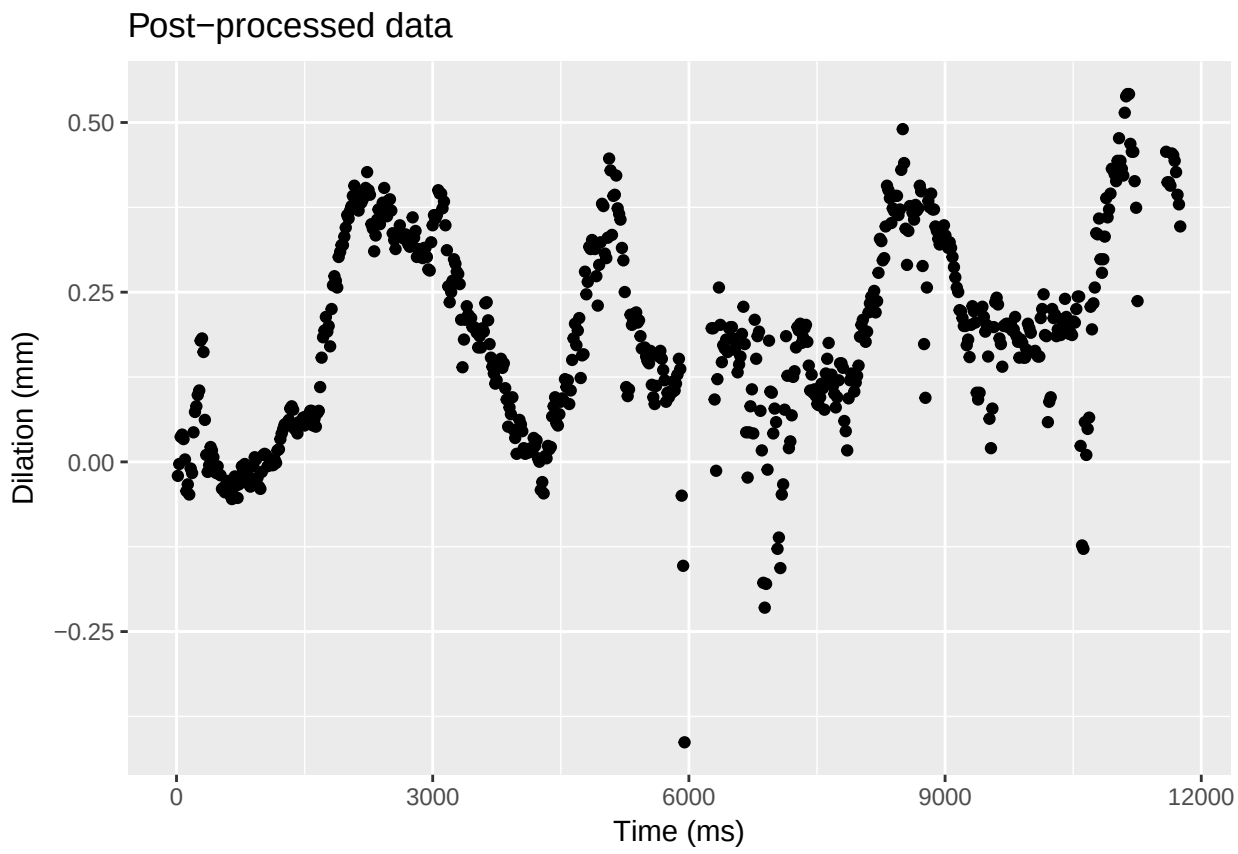
```
# library(ggplot2)
# You can use base R for your plots, but I personally like ggplot2
print(
  ggplot(p1, aes(x=time, y=dilation))
      + geom_point() +
    labs(x= "Time (ms)", y = "Dilation (mm)", title = "Pre-processed data")
       )
```

```
## Warning: Removed 58 rows containing missing values (geom_point).
```




```
print(
  ggplot(p1, aes(x=time, y=smooth)) +
    geom_point() +
  labs(x = "Time (ms)", y = "Dilation (mm)", titled = "Post-processed data")
  )
```

```
## Warning: Removed 37 rows containing missing values (geom_point).
```



We see that we've lost some noisiness in the data, but the shapes of the peaks and valleys stay broadly intact. The smoothing seems to have kept the integrity of the data.

Housekeeping

We'll also add some housekeeping tags, in case we want to aggregrate the data into a massive data frame later in the analysis, or if we want to calculate statistics.

The first housekeeping tag will be segments. Each segment corresponds to the time spent in a particular AOI. If there is a transition (to whitespace or to a different AOI), there is a new segment as well.

```
z <- rle(p1$focus)$lengths
p1$segment <- rep(1:length(z), z)
```

The next housekeeping tags will just be some categorical variables about the subject's codename and whether they correctly answered the question.

```
library(tidyr)
p1 <- p1 %>% dplyr:::mutate(times = time/1000) # add a column for time in seconds
p1 <- p1 %>% tidyr:::crossing(subject = "p1", correct = "0") #add in a tag for the codenumber, as well
head(p1)
```

```
## # A tibble: 6 x 8
##    time dilation focus     smooth segment times subject correct
##   <dbl>    <dbl> <chr>      <dbl>   <int> <dbl> <chr>   <chr>
## 1    16  -0.0831 diagram -0.0206        1 0.016 p1      0
## 2    32   0.0419 diagram -0.00312       1 0.032 p1      0
## 3    49   0.0319 diagram  0.0369        1 0.049 p1      0
```

```
## 4    66   0.0369 diagram  0.0402        1 0.066 p1      0
## 5    82   0.0519 diagram  0.0335        1 0.082 p1      0
## 6    99   0.0119 diagram  0.00354       1 0.099 p1      0
```

Epochs Coding

Now we'll add in the information about the epochs. The epochs were coded for in NVivo, using the video coding funtionality. We used the transcript feature to add a .txt file as a transcript. This transcript had row number, timestamp (ms), and average dilation. After coding the videos, we were able to use coding stripes to correlate the end of each coded video segment to the row index of the data frame using the transcript.

For this, we use the row number to figure out where the epoch in the full data set changes (`breaks_p1` below). `breaks_p1` returns a vector of values. We then use the function `diff` from base R to calculate how many points each category should be applied to (how many of the rows should have a certain epoch assigned to them). This gives us `lengths_p1`, which is also a vector of values. We create a df with the epochs we coded in the video coding, in sequential order. We then use the function `rep()`. `rep()` repeats an x value for a certain number of times, determined by rep(x, ...), where "..." is how many times x should be repeated. We use this to create a column that has the epoch tags as categorical variables to each time point.

```
breaks_p1 <- c(0, 44, 282, 386, 528, 643, 705) # each value here is the row index number
# use diff to calculate how many points each category should be applied to
lengths_p1 <- diff(breaks_p1)
# The categories in the order matching up to the intervals above
categories_p1 <- c("initiation", "reads description", "reads question with glance at answers",
                   "looks at answers","looks at diagram", "looks at answers")
# repeat each category to build df$out
p1$epochs <- rep(categories_p1, lengths_p1)

# inspect the output
str(p1$epochs)
```

```
##  chr [1:705] "initiation" "initiation" "initiation" "initiation" ...
```

We'll add another housekeeping variable, to make sure that each epoch is correlated to the order they appear in. We already have a variable that accounts for the length of each epoch (`lengths_p1`), so we can use that again here.

```
p1$epoch_order <- rep(1:length(lengths_p1), lengths_p1)
```

Remove Blinks

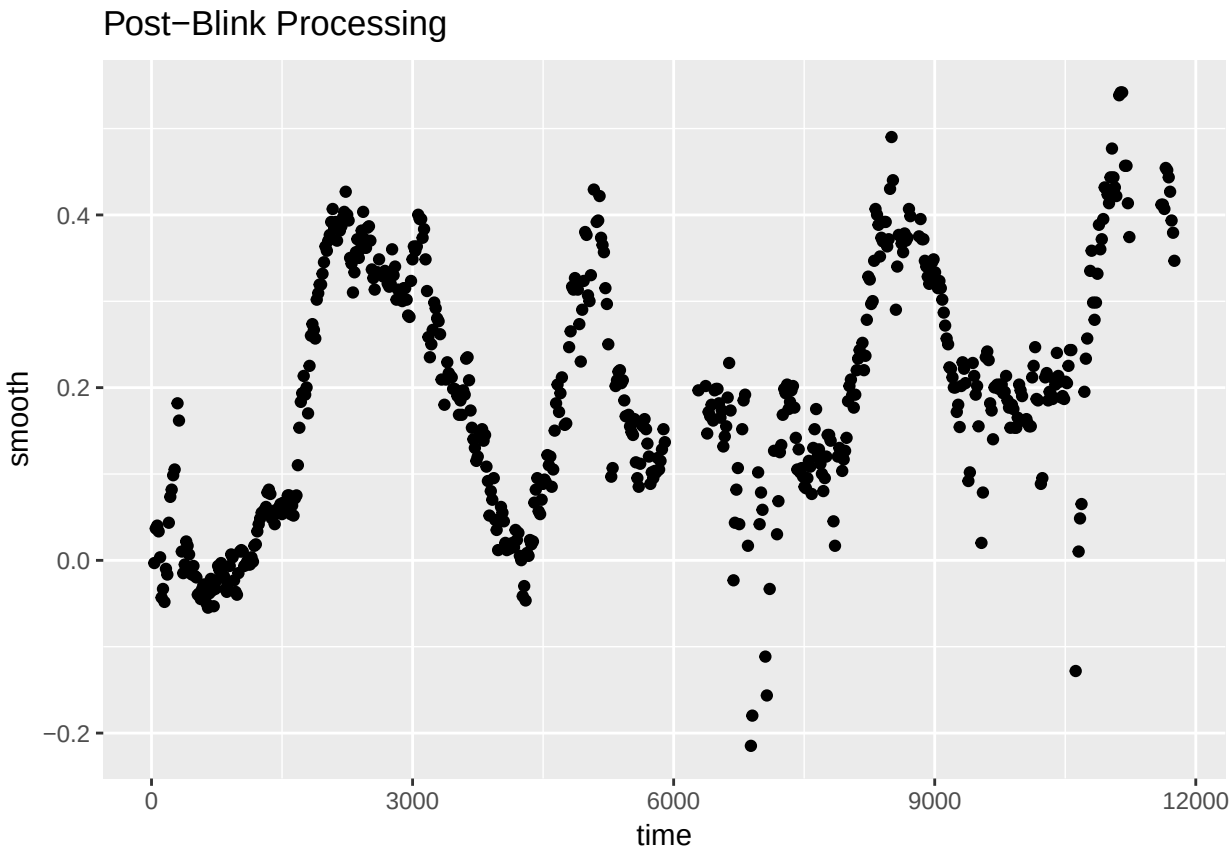
(Note: This should be done earlier in the processing, but as this was the last step developed it was done last. However, this should not affect the quality of the analysis.)

We also want to remove noise and outliers from blinks. To do this, we're going to follow the method from Kret and Sjak-Shie (2018). We first calculate a velocity profile, to see how rapidly dilations change. In pupillometric data, a signifying characteristic of blinks is that the measured pupil dilation changes more rapidly than physiologically possible, because the eye closing prevents the pupil from being measured. To calculate the velocity profile, we'll find the change in dilation over the change in time.

Then we'll use a method from Leys et al (2013). We'll set an acceptable amount of deviation for the velocity. To do this, we calculate the median absolute standard deviation, and set upper and lower bounds (thresholds) of acceptable values. We'll make these upper and lower bounds within 3 median absolute standard deviations of the median. We then remove these values from the data set!

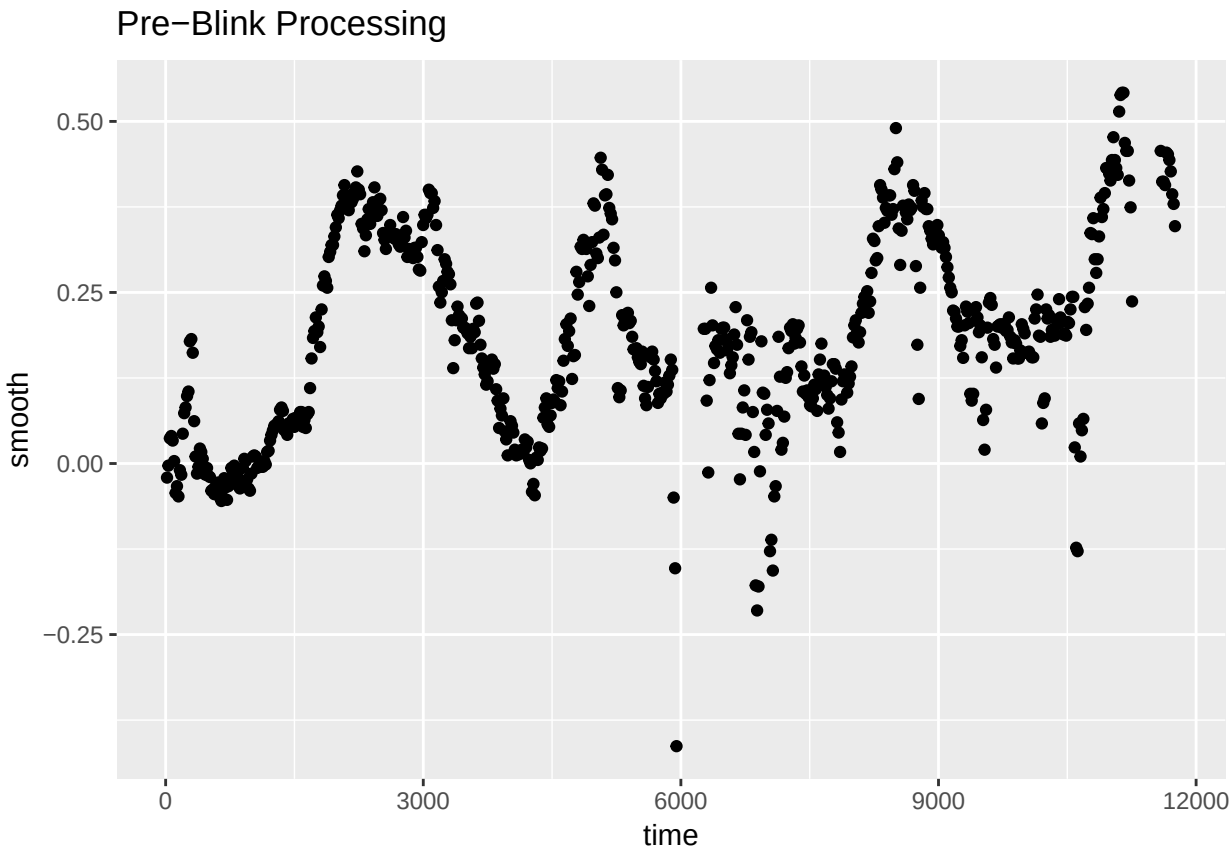
```r
# calculate a velocity profile
p1 <- p1 %>% mutate(vel = c(NA, diff(smooth)/diff(time)))
# calculate the absolute standard deviation from the median
mad_1 <- mad(p1$vel, na.rm = TRUE)
# set lower and upper bounds from the median
threshold_1 <- median(p1$vel, na.rm =TRUE) + 3*mad_1
threshold_1l <- median(p1$vel, na.rm =TRUE) - 3*mad_1

# Let's call a new data frame to remove the values from, so we can compare pre and post
# processing graphically
vel_p1 <- subset(p1, vel < threshold_1 & vel > threshold_1l)
print(ggplot(vel_p1, aes(x=time, y=smooth)) +geom_point() +ggtitle("Post-Blink Processing"))
```




```r
print(ggplot(p1, aes(x=time, y=smooth)) +geom_point() + ggtitle("Pre-Blink Processing"))
```

```
## Warning: Removed 37 rows containing missing values (geom_point).
```



The plots look good! So we set rename `vel_p1` as `p1`. At this point, if there were any visible single outliers that seemed to be physiologically impossible, we could manually remove those from the set.

```
p1 <- vel_p1
```

And that's it! Let's check out our final data frame:

```
print(str(p1))
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    615 obs. of  11 variables:
##  $ time       : num  32 49 66 82 99 116 132 149 166 182 ...
##  $ dilation   : num  0.0419 0.0319 0.0369 0.0519 0.0119 ...
##  $ focus      : chr  "diagram" "diagram" "diagram" "diagram" ...
##  $ smooth     : num  -0.00312 0.03688 0.04021 0.03354 0.00354 ...
##  $ segment    : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ times      : num  0.032 0.049 0.066 0.082 0.099 0.116 0.132 0.149 0.166 0.182 ...
##  $ subject    : chr  "p1" "p1" "p1" "p1" ...
##  $ correct    : chr  "0" "0" "0" "0" ...
##  $ epochs     : chr  "initiation" "initiation" "initiation" "initiation" ...
##  $ epoch_order: int  1 1 1 1 1 1 1 1 1 1 ...
##  $ vel        : num  0.001094 0.002353 0.000196 -0.000417 -0.001765 ...
## NULL
```

We now have a data frame with dilation values (raw and processed), time in two units, information about the AOIs from the eye tracker, and the epochs codes.