Webinar 3

Francesco Pirotti <francesco.pirotti@unipd.it>

#Summer Webinar Series — August xx, 2020

Contents

Code & Copyleft

Code and data available in GITHUB here

```r
library(icon)
```

Agenda

- Part 2 from webinar related to how Google Earth Engine (GEE) can be used to extract time series per each polygon of areas damaged by VAIA
- Import extracted data in R CRAN
- Plot time series data and applying a Z-score filter over past N values to detect changes

Before we begin

- For this Webinar we use R and RStudio IDE - if you want to follow along, make sure you have have installed R + RStudio in your PC

- Make sure you have downloaded the data download here.

Suggested reading

- Some helpful pre-webinar reading

- Data from Google Earth Engine is in GeoJSON format
- Date information is in ou data is in Date-time UNIX standard
- `For` loops here
- Plotting with ggplot2:
  * basics
  * cheatsheet
-

Data

Data from previous webinar and you can download here.
The data is in GeoJSON format, so processing it takes a bit of extra steps as it is different from typical tabular data. JSON data is unstructured data and very popular in the "BigDATA" era. For further reading see wikipedia

R libraries

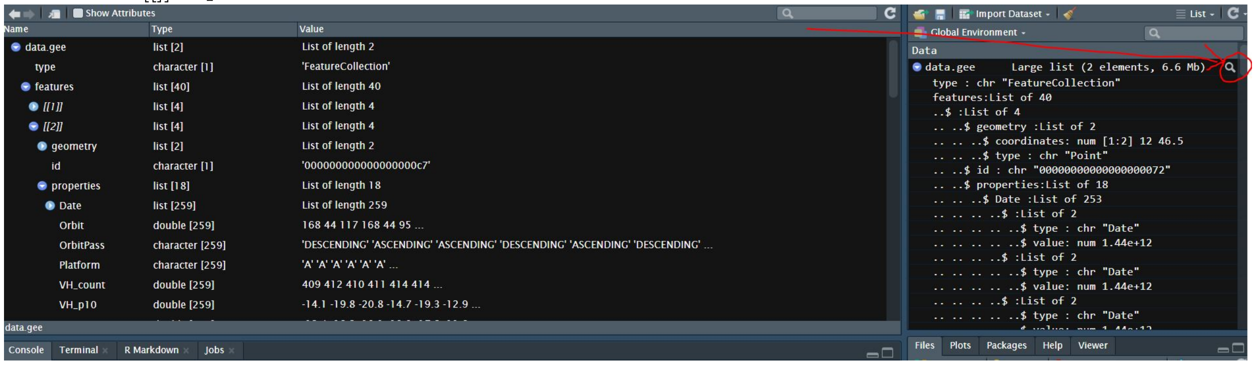
```r
library(sf)
library(mapview)
library(raster)
library(RJSONIO)
library(ggplot2)
library(leaflet)
library(mapview)
library(plotly)
```

Read data

Data is zipped, but we can extract with R and then read the file. The first line extracts the file **webinar2vaia.geojson** from the archived dataset in **GEEdataFromWebinar2.zip**... The second line reads the GeoJSON data.

```r
mydata<-unzip("data/webinar3/GEEdataFromWebinar2.zip")
data.gee <-    fromJSON( mydata )
```

To see how data is structured, go to top left panel in RStudio, "Environment" and click the "data.gee" icon on the right like shown in picture below. You can see many layers of information, that you can access with the "$" or "[[]]" operators.



```
Name              Type              Value
data.gee          list [2]          List of length 2
  type            character [1]     'FeatureCollection'
  features        list [40]         List of length 40
    [[1]]         list [4]          List of length 4
    [[2]]         list [4]          List of length 4
      geometry    list [2]          List of length 2
      id          character [1]     '000000000000000000c7'
      properties  list [18]         List of length 18
        Date      list [259]        List of length 259
        Orbit     double [259]      168 44 117 168 44 95 ...
        OrbitPass character [259]   'DESCENDING' 'ASCENDING' 'ASCENDING' 'DESCENDING' 'ASCENDING' 'DESCENDING' ...
        Platform  character [259]   'A' 'A' 'A' 'A' 'A' 'A' ...
        VH_count  double [259]      409 412 410 411 414 414 ...
        VH_p10    double [259]      -14.1 -19.8 -20.8 -14.7 -19.3 -12.9 ...
data.gee
```

```
Data
data.gee      Large list (2 elements, 6.6 Mb)
  type : chr "FeatureCollection"
  features:List of 40
  ..$ :List of 4
  .. ..$ geometry :List of 2
  .. .. ..$ coordinates: num [1:2] 12 46.5
  .. .. ..$ type : chr "Point"
  .. ..$ id : chr "00000000000000000072"
  .. ..$ properties:List of 18
  .. .. ..$ Date :List of 253
  .. .. .. ..$ :List of 2
  .. .. .. .. ..$ type : chr "Date"
  .. .. .. .. ..$ value: num 1.44e+12
  .. .. .. ..$ :List of 2
  .. .. .. .. ..$ type : chr "Date"
  .. .. .. .. ..$ value: num 1.44e+12
  .. .. .. ..$ :List of 2
  .. .. .. .. ..$ type : chr "Date"
```

For example below we analyse the object "data.gee" and show "type", "id" of first feature(polygon) "features"[[1]]$"id".
Please note how we can use index ([[N]]) where N is the index. If we use **single square brakets** ([]) then we can show multiple indexes such as 1:3 (last line). The **":"** is a "range" of number, thus 1:3 means 1,2,3.

```
print(data.gee$type)
```

```
## [1] "FeatureCollection"
```

```
print(data.gee$features[[1]]$id )
```

```
## [1] "00000000000000000072"
```

```
print(data.gee$features[[1]]$properties$Date[1:3] )
```

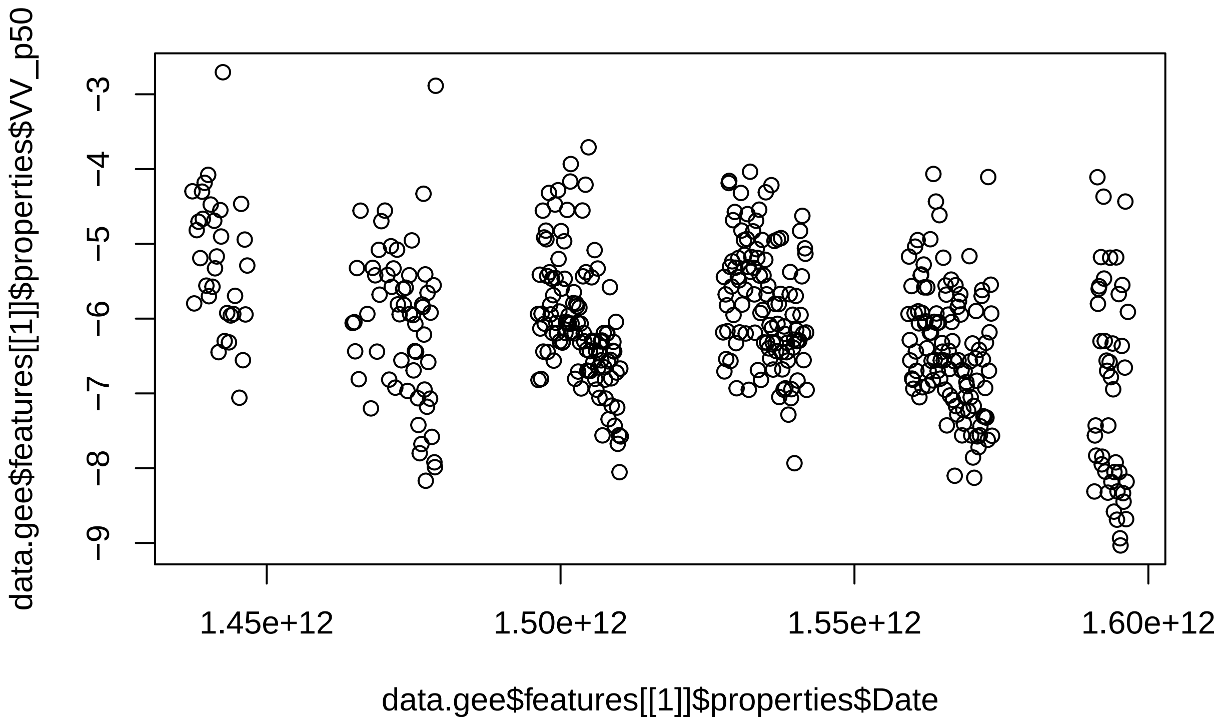
```
## [1] 1.437370e+12 1.437671e+12 1.438103e+12
```

The data has 40 polygons of areas heavily damaged by VAIA storm. The "address" (index) of the information (properties) of the first area is

```
data.gee$features[[1]]$properties
```

and the following plots the data.

```
plot( data.gee$features[[1]]$properties$Date,
      data.gee$features[[1]]$properties$VV_p50 )
```



But the plot is strange… why is "Date" in numbers? We can ask R to provide the class that the data is in by typing `class` function.

```r
class( data.gee$features[[1]]$properties$Date)
```

```
## [1] "numeric"
```

A Date timestamp is actually stored as a big integer, and R reads it as "numeric" and we must convert the class **numeric** to **date**.
R has many types of "Date-Time" classes, one is called "POSIXct". Get more info by typing `?POSIXct` in the console and read the documentation.

We can proceed by using the function "as.POSIXct". But before we are dividing by 1000, because our Date from GEE is in milliseconds that elapsed from a starting date. and then provide an "origin" which is from when the "clock" started to record time. It is a common standard, also used by GEE, to count time since the first of January in 1970 see here why.

```r
 as.POSIXct.numeric( as.numeric( data.gee$features[[1]]$properties$Date[1:10] )/1000,
                    origin = "1970-01-01", tz = "GMT")
```

```
##  [1] "2015-07-20 05:26:37 GMT" "2015-07-23 16:58:34 GMT"
##  [3] "2015-07-28 17:06:46 GMT" "2015-08-01 05:26:43 GMT"
##  [5] "2015-08-04 16:58:24 GMT" "2015-08-08 05:18:23 GMT"
##  [7] "2015-08-09 17:06:27 GMT" "2015-08-13 05:26:44 GMT"
##  [9] "2015-08-16 16:58:25 GMT" "2015-08-20 05:18:23 GMT"
```

Ok, so how do we convert all dates from all features? We must use a "loop" over all the features. There are several ways to do this:

- "starters" reading about simple "for" loops here
- further reading on for/while/repeat/apply loops here
- reading on parallel loops (very important if you have very big datasets) here

First we convert the properties (that we extracted from GEE remember?) from the first feature to a **data frame** which is R's way to define a table. The function "head" provides a view of the first rows of the table, to see how our data looks.

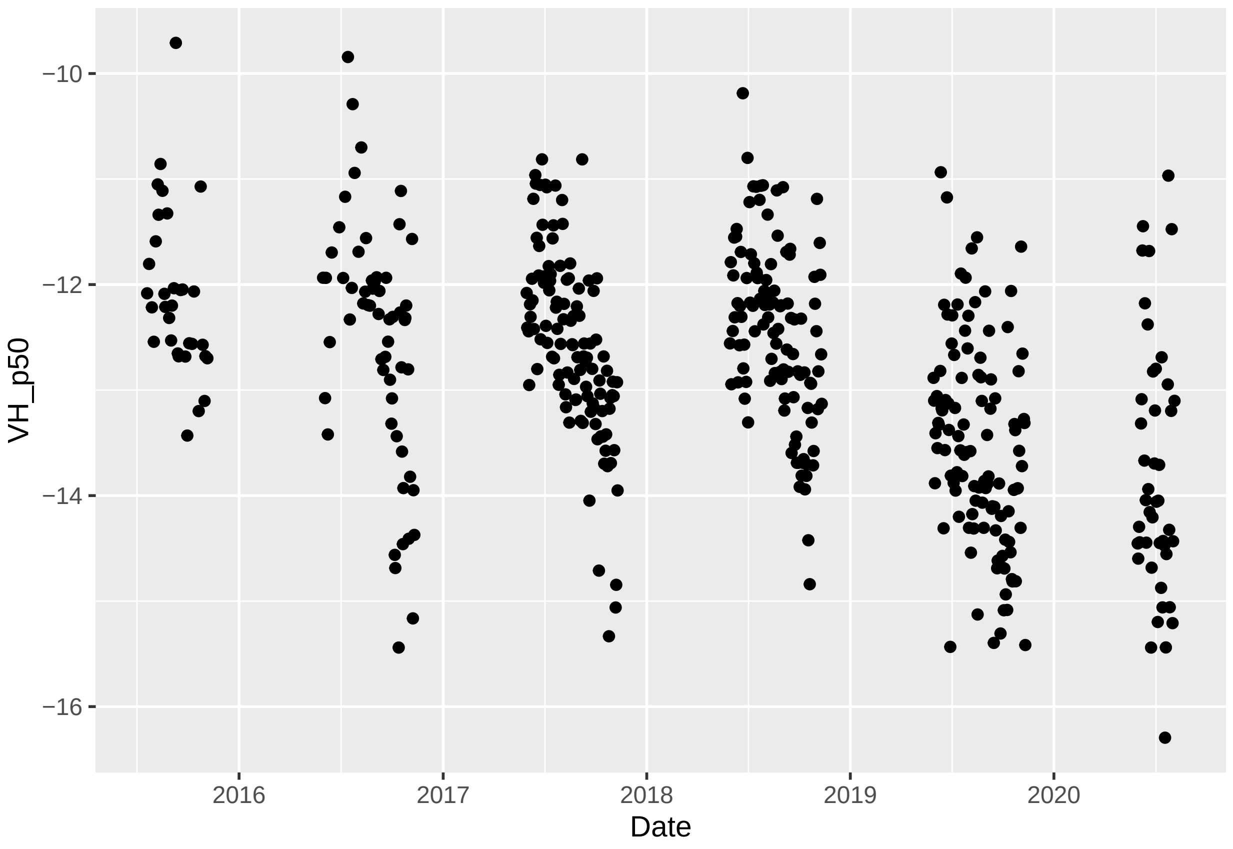
```r
  feature1<-as.data.frame( data.gee$features[[1]]$properties )
  head(feature1)
```

```
##           Date Orbit  OrbitPass Platform VH_count    VH_p10    VH_p25    VH_p50
## 1 1.437370e+12   168 DESCENDING        A      346 -19.88678 -14.91552 -12.08316
## 2 1.437671e+12    44  ASCENDING        A      342 -14.94571 -13.67300 -11.80462
## 3 1.438103e+12   117  ASCENDING        A      337 -15.42866 -13.94143 -12.21600
## 4 1.438407e+12   168 DESCENDING        A      345 -18.16550 -14.94717 -12.54223
## 5 1.438708e+12    44  ASCENDING        A      348 -14.90782 -13.15371 -11.59059
## 6 1.439011e+12    95 DESCENDING        A      341 -14.10823 -12.71105 -11.05058
##       VH_p75    VH_p90 VV_count     VV_p10    VV_p25    VV_p50    VV_p75
## 1 -10.058983 -8.268616      346 -11.783320 -6.950433 -4.297575 -1.693068
## 2 -10.306942 -8.794244      342  -9.369257 -7.447982 -5.796786 -4.196441
## 3 -10.457614 -8.455414      337  -8.427476 -6.952696 -4.817103 -2.462672
## 4 -10.161611 -8.692222      345 -11.180967 -7.663552 -4.704455 -2.805866
## 5 -10.398530 -8.841611      348  -8.458085 -6.813063 -5.191033 -3.686764
## 6  -9.410976 -7.977846      341  -8.068577 -6.444503 -4.302057 -2.208312
##       VV_p90   id layover_count layover_p10 layover_p25 layover_p50 layover_p75
## 1 -0.1952980 1237           346           0           1           1           1
## 2 -2.8290188 1237           342           1           1           1           1
## 3 -0.9399215 1237           337           1           1           1           1
## 4 -1.5724907 1237           345           0           1           1           1
## 5 -2.1751318 1237           348           1           1           1           1
## 6 -0.7156182 1237           341           1           1           1           1
##   layover_p90 perc_disb shadow_count shadow_p10 shadow_p25 shadow_p50
## 1           1        80          346          1          1          1
## 2           1        80          342          1          1          1
## 3           1        80          337          1          1          1
## 4           1        80          345          1          1          1
## 5           1        80          348          1          1          1
## 6           1        80          341          1          1          1
##   shadow_p75 shadow_p90
## 1          1          1
## 2          1          1
## 3          1          1
## 4          1          1
## 5          1          1
## 6          1          1
```

Plot data

Ok so let's convert the Date column and start some more advanced plotting using **ggplot2** library. This library allows advanced plotting.

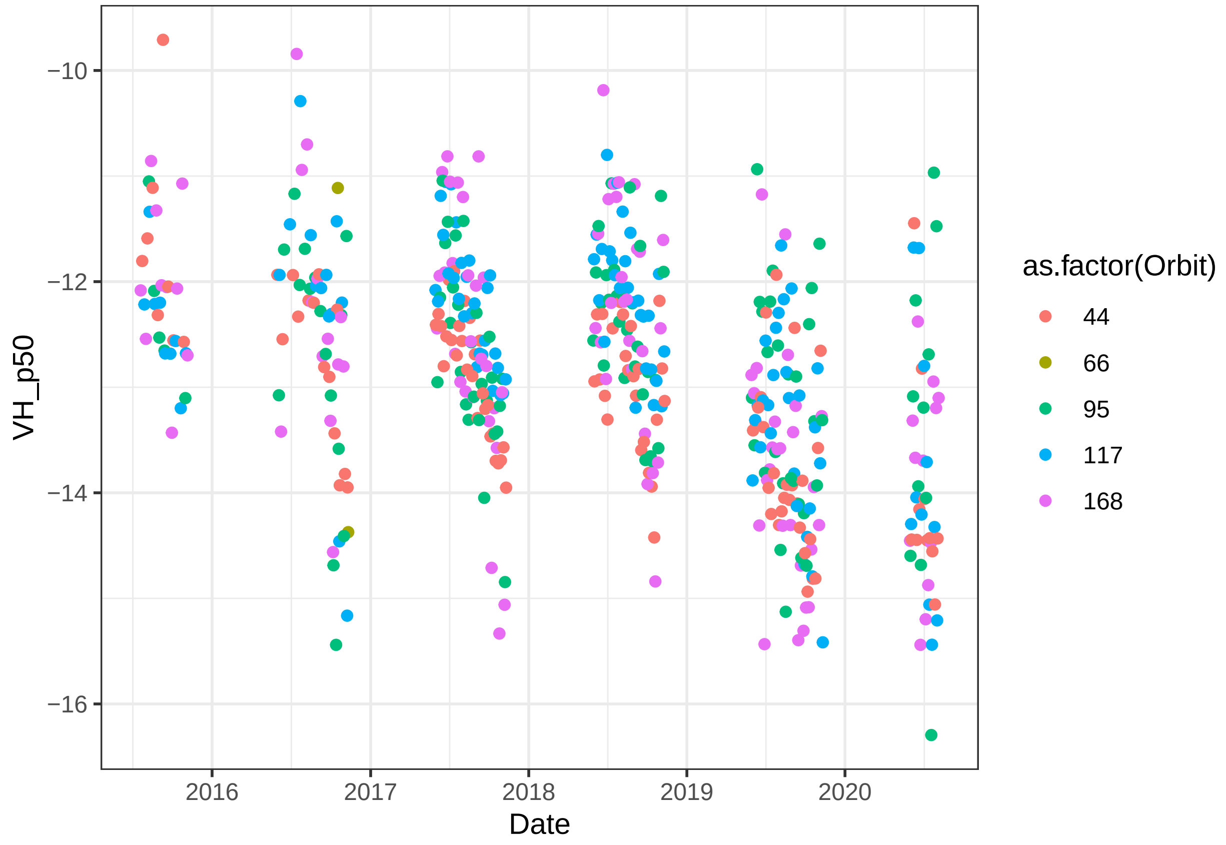
```
feature1$Date<- as.POSIXct.numeric( as.numeric( feature1$Date  )/1000,
                                    origin = "1970-01-01", tz = "GMT")

ggplot(feature1, aes(x=Date, y=VH_p50) ) + geom_point()
```



Even better, we want white background (better for papers), and we want a different color depending on the value of the *Orbit* column. This column provides information on the Sentinel 1 image - same Orbit value means that it is the same

Why do we put "as.factor" to the Orbit column? Because otherwise it gets interpreted as a number and the ggplot will give a continous color scale... you can try to remove the function and see what happens.

```r
ggplot(feature1, aes(x=Date, y=VH_p50) ) +
  geom_point( aes(colour=as.factor(Orbit) ) ) +
  theme_bw()
```



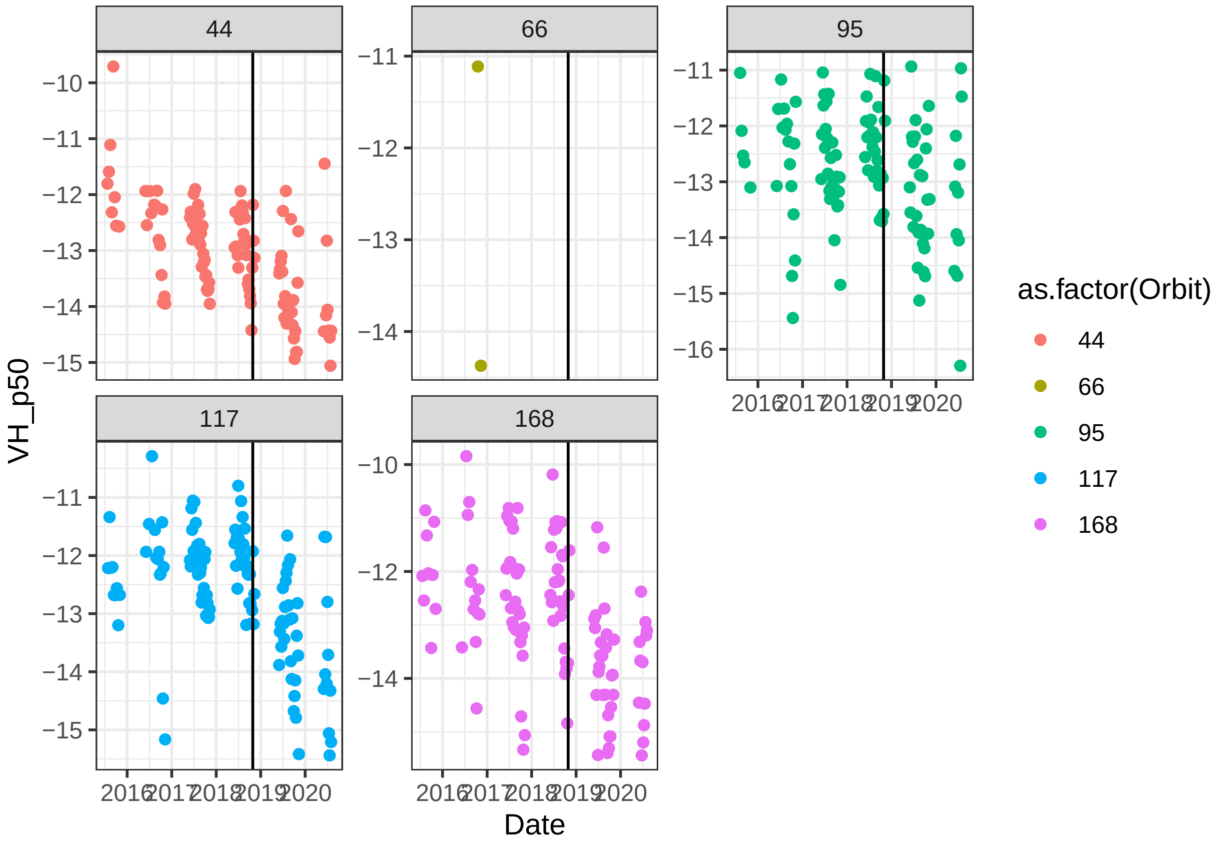
We see from the image above that there is a clear difference between different "Orbit" values. Let's divide plots per Orbit so that we scale them accordingly, and add a line with the date of the VAIA storm.

From results we can see a very weak signal that

+
−
300 m
1000 ft
Leaflet | © OpenStreetMap © CartoDB, CIRGEO-AVEPA

Below we create a more complex plot with “facets”.

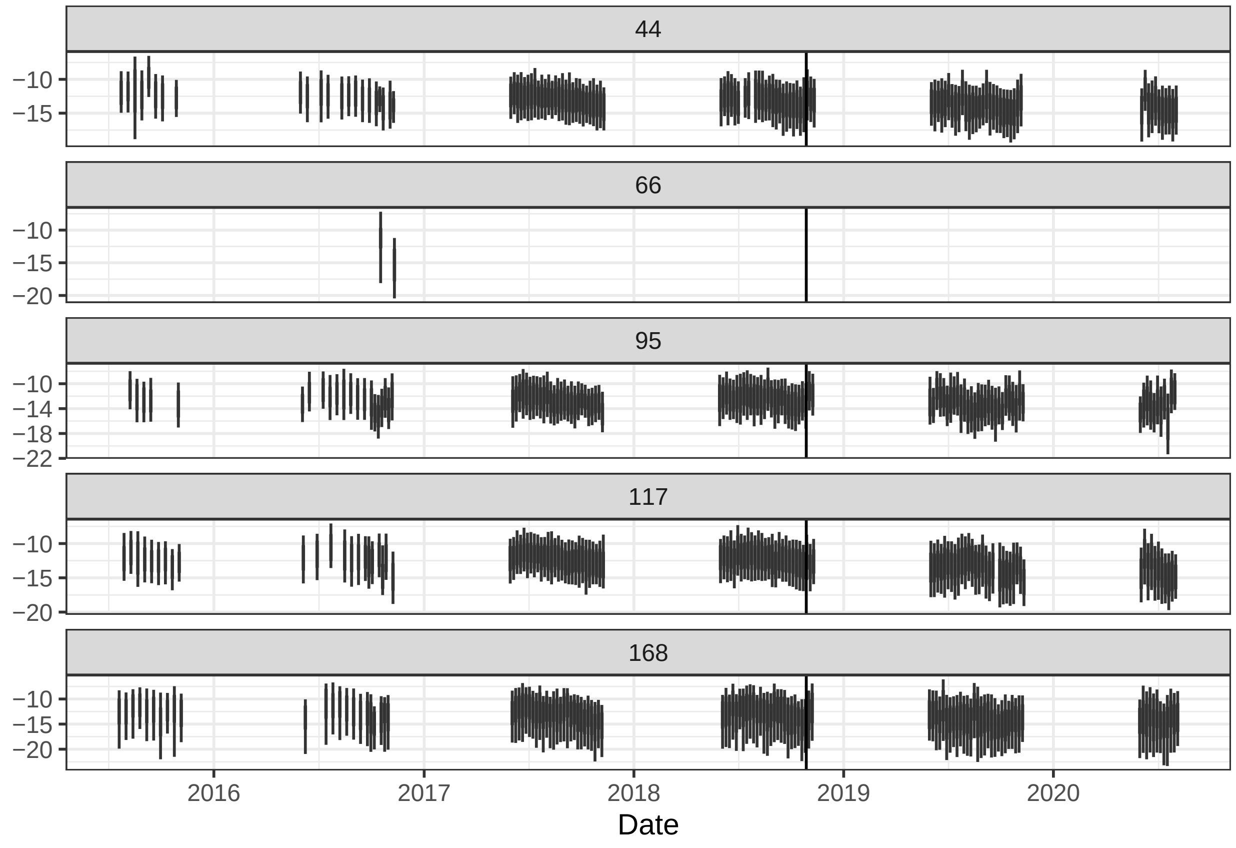
```
p <- ggplot(feature1, aes(x=Date, y=VH_p50) ) +
   geom_point( aes(colour=as.factor(Orbit) ) ) +
   facet_wrap(vars(Orbit), scales = "free_y" )  +
   geom_vline(aes(xintercept= as.POSIXct("2018-10-28") ) ) +
   theme_bw()

p
```



```
# fig <- ggplotly(p)
#
# fig
```

But wait… we plotted the median VH values inside each area, we want to also plot the distribution using the other percentiles to create a box-whisker plot. In this case we add "stat = identity" and provide info on the various parts of the boxplot - see also that we put all facets in a single column to improve time visualization.

```
ggplot(feature1, aes(Date) ) +
  geom_boxplot(
   aes(group=Date, ymin = VH_p10, lower = VH_p25, middle = VH_p50, upper = VH_p75, ymax = VH_p90),
   stat = "identity"
 ) +
    facet_wrap(vars(Orbit), scales = "free_y", ncol = 1 )  +
    geom_vline(aes(xintercept= as.POSIXct("2018-10-28") ) ) +
    theme_bw()
```



Analyze data

We can use “plotly” library to .

```
# plot_ly(feature1, x = ~Date, xstart= ~Date,  xend = ~Date, color = ~Orbit,
#         colors = c("red", "forestgreen"), hoverinfo = "none") %>%
#   add_segments(y = ~VH_p50,  ymin = ~VH_p25, yend = ~VH_p25, size = I(1)) %>%
#  # add_segments(y = ~Open, yend = ~Close, size = I(3)) %>%
#   layout(showlegend = FALSE, yaxis = list(title = "VH")) %>%
#   rangeslider()
```