Workshop PhytoNokoué - TP1 - Data Loading

Arthur Capet

2024-03-17

Part 1 : Basic R.

Variables

In R we can assign value to variables, and use them afterwards for computation.

```
a <- 5
print(4+a)
```

Variables can have different types : - “numeric” is a number (real). - “chr” is a character string.

```
# We can also include comment lines in code blocs.
print("The type and value of 'a' are :")
typeof(a)
print(a)
```

```
# Let's define a "character string" variables
b <- 'Hello'
print("The type and value of 'b' is :")
typeof(b)
print(b)
```

```
print("The type and value of 'TRUE' is :")
typeof(TRUE)
print(TRUE)
```

```
myflag <- 1+1==2
print("The type and value of 'myflag' is :")
typeof(myflag)
print(myflag)
```

Those are variables that contain only a single element. Sometimes, it is usefull to build vector variables that contains several elements.

```
A <- c(5,8,13)
print(A)
print(A[2])
```

Vectors can be built for variables of any type.

```
B <- c('Hello','World')
B
```

```
C <- (A < 10)
print(C)
```

Of course, methods exists to build vectors implicitly :

```
print(seq (from = 1, to = 8, by = 2))
print(1:4)
print(5:-1)
```

Functions

In general, we use functions to apply the same operation to different arguments.

```
squaremynumber <- function(number){
  return(number*number)
}

print (squaremynumber(2))

print (squaremynumber(5))

print (squaremynumber(A))
```

Part 2 : Data load and first manipulation

We received monitoring data from the Lake Nokoué. These have been stored in an '.xls' file. Our first objective is to read these data and import them in an interactive R session.

```
filename <-"nokoue.xlsx"
print(filename)
```

```
dfmai <- read_excel(filename,"Mai")
dfmai
```

The `dfmai` variable now contains a **Tibble Dataframe** (we'll just use the word "**dataFrame**"). This is a very convenient variable type for data analysis. It contains different columns ("Variables") and rows ("Observations"). The nice thing with that type is that different columns can have different types (off course all observations in a given column has to have the same type).

The content of a given column can be accessed simply :

```
print(dfmai$T)

print(dfmai[['T']])
```

When using a sinlge pair of bracket, the returned element is a one-column dataframe

```
print(dfmai['T'])
```

.. and there are plenty of command to manipulate them, for instance for subseting,

```
subset(dfmai, dfmai$T>29.5)
```

What we need to do now is : - Build a descriptor dataframe with units and definitions for all variables - Ensure that observations have the correct type. - Gather all months in a single large data frame.

Variable Description Dataframe.

The columns in our data frame are of different nature. We need to distinguish :

- Coordinates variables: those are not analysed **per se**, but are used as a reference basis to analyze other variables.

- Environmental variables: those characterize the bio-chemo-physical environment, in our context we think of them as related to the "habitat".
- Species variables : those are counts of species (i.e. phytoplanctons).

There is a dedicated sheet in the data file to describe the variables.

```
dfvar <- read_excel(filename,"Units")
dfvar
```

To give an idea of why this is very useful, let's consider the following for loop :

```
for (v in colnames(dfmai)){
  varname <- v
  local_dfvar <- subset(dfvar, dfvar$Parameters==v)
  vartype <- local_dfvar$Type
  varunit <- local_dfvar$Units
  varname <- local_dfvar$Longname
  print(paste(v," is a ", vartype, "variable, expressed in ", varunit, " and representing", varname  ))

}
```

Observation type

But Species counts should un fact be given as integers !!

We can fo this manually, column by column :

```
dfmai$EUGL <-as.integer(dfmai$EUGL)
dfmai
```

But this is boring, so let us use our variable description dataframe instead !

```
for (v in subset(dfvar, dfvar$Type=="Species")$Parameters){
  print(v)
  dfmai[[v]] <-as.integer(dfmai[[v]])
}

dfmai
```

Combining dataframe for the different months.

We have one sheet per month. It will be much more convenient to have all data in a single data frame. Since we have to apply the conversion to integer, we can make a function to load a monthly data frame.

```
load_monthly_df <- function(monthstr){
  df <- read_excel(filename,monthstr)
  for (v in subset(dfvar, dfvar$Type=="Species")$Parameters){
    df[[v]] <-as.integer(df[[v]])
  }
  return(df)
}
```

This is just to show that the function works.

```
df <- load_monthly_df('Fevrier')

df
```

Now we will use this function in a loop, to build a list of dataframes, i.e. a single variable object that includes all the dataframe.

```
listofdf <- list()

monthnames <- c('Janvier','Fevrier','Mars','Avril','Mai','Juin','Juillet','Aout','Septembre','Octobre',

for (i in 1:12){
  print(monthnames[i])
  df <- load_monthly_df(monthnames[i])
  listofdf[[i]] <- df
}
```

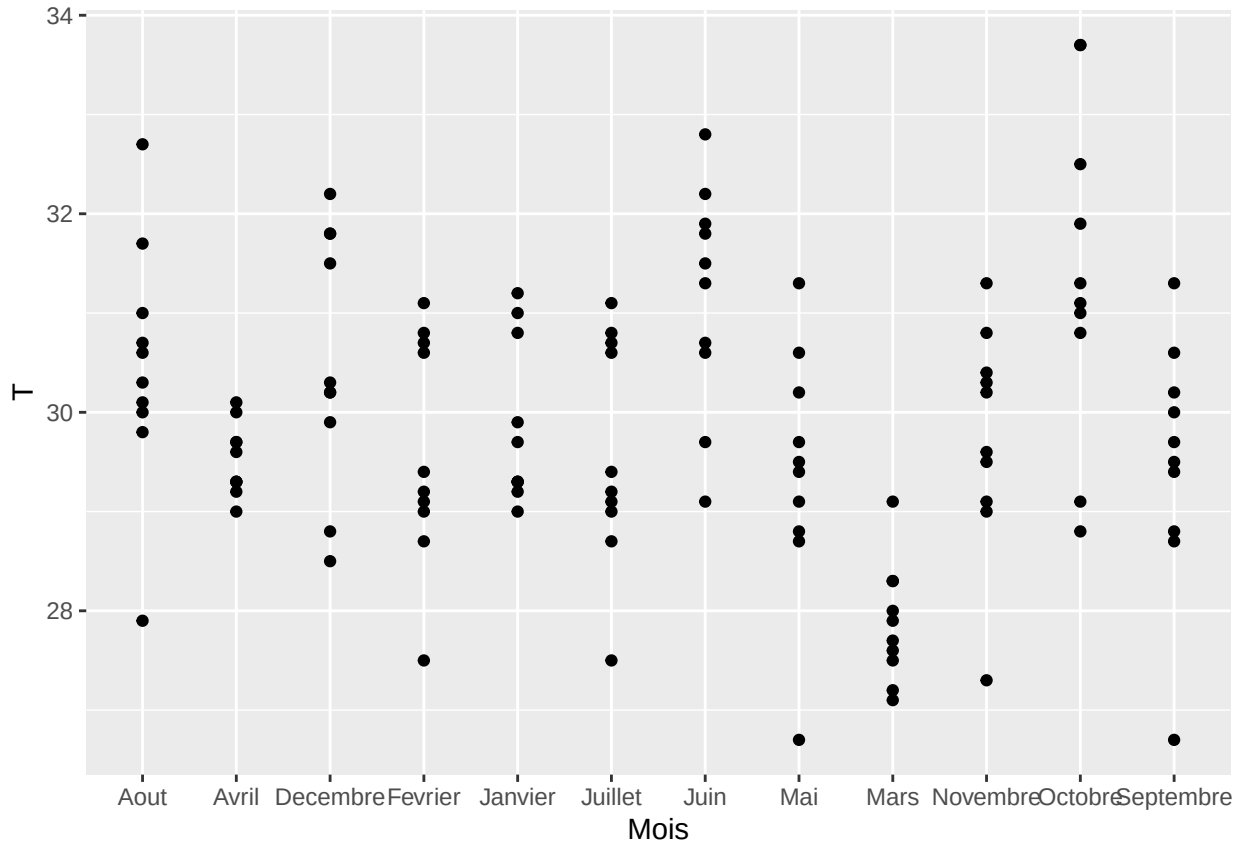
Now we have a list of dataframes, but we still need to gather them in a single dataframe. This operation is called concatenate, and can be achieved with the function rbind, where 'r' stands for 'rows' (the different dataframes are to be considered as group of rows of a larger dataframe).

```
dfdata <- do.call(rbind, listofdf)
```

Our dataset is ready .. or almost.

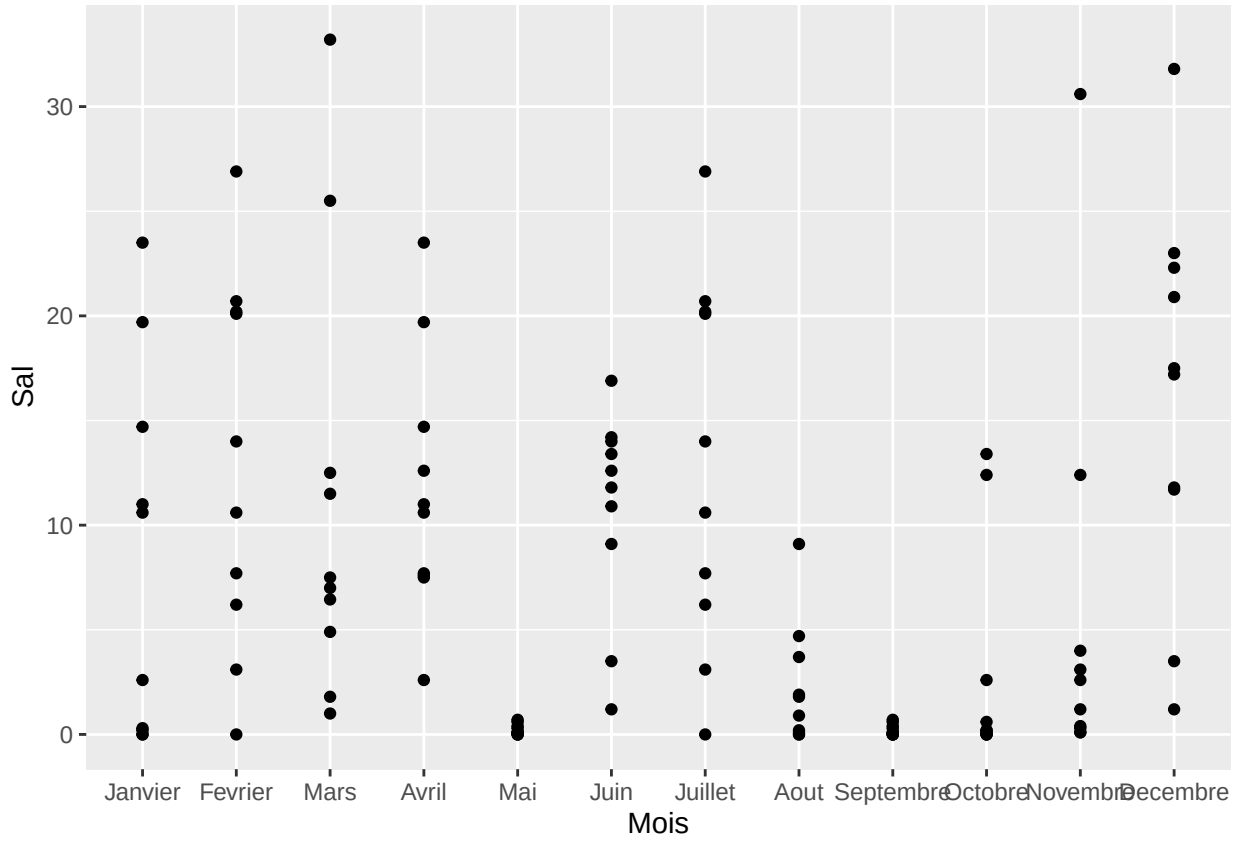
First plots (using ggplot)

```
ggplot(dfdata, aes(x=Mois, y = T))+geom_point()
```



There is an issue, because the variable `Mois` is just a series of character strings, ggplot doesn't know in which order they should appear. We can clarify this, byt defining this column as a set of 'factors'. Factors are discrete categorical variables, but they can be defined to be ordered.

```
dfdata$Mois <- factor(dfdata$Mois, ordered=TRUE, levels=monthnames)

ggplot(dfdata, aes(x=Mois, y = Sal))+geom_point()
```



In general, it is easier to work in ggplot with 'long' table format, rather than 'wide' table format. We can pass from 'wide' to 'long' using the command `melt`.

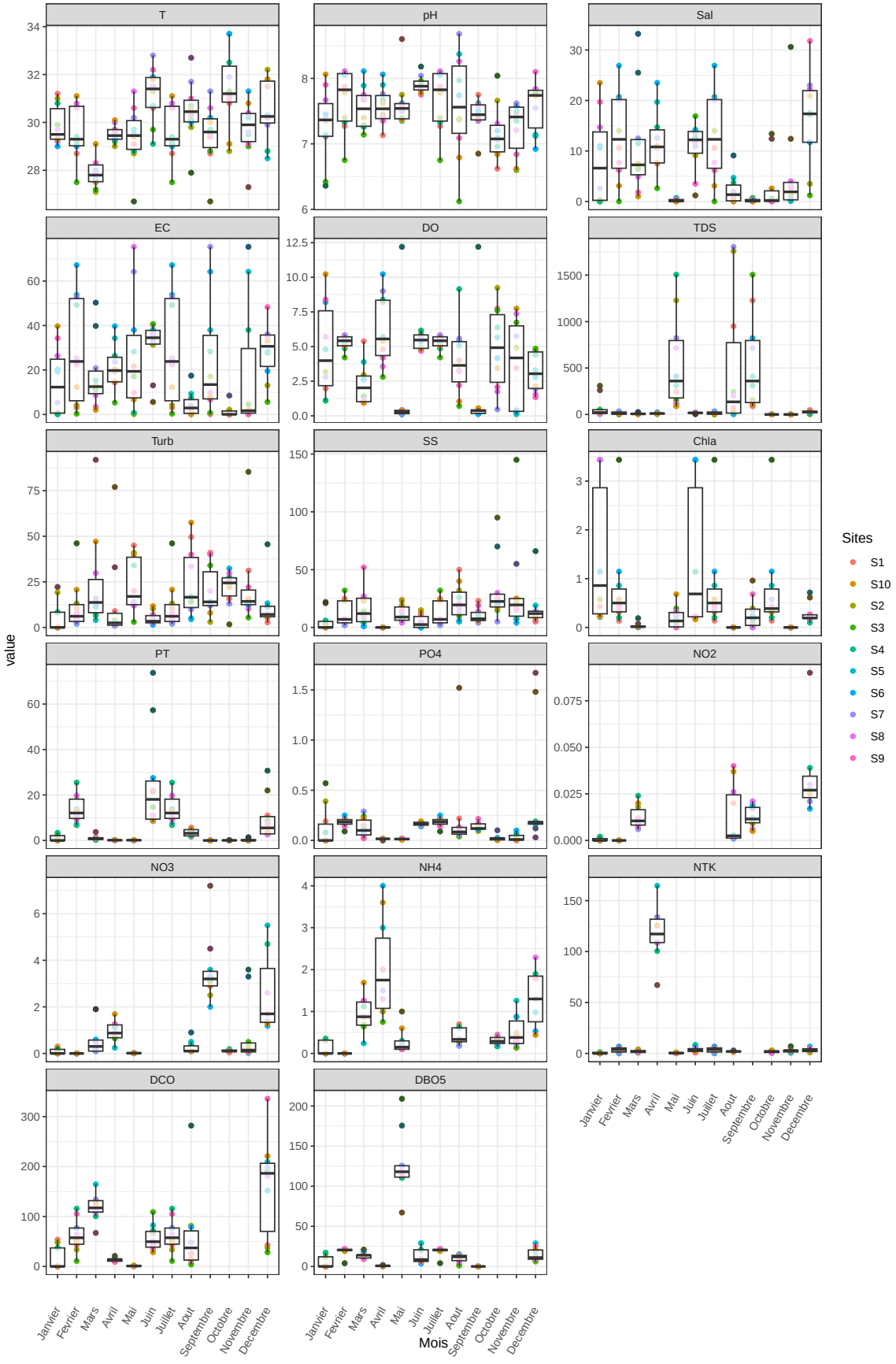
```
ldf <- melt(dfdata, id.vars = subset(dfvar, dfvar$Type=='Coordinate')$Parameters )
ldf
```

Now, we can use a bit more advanced ggplot nomenclature to build more complex plots.

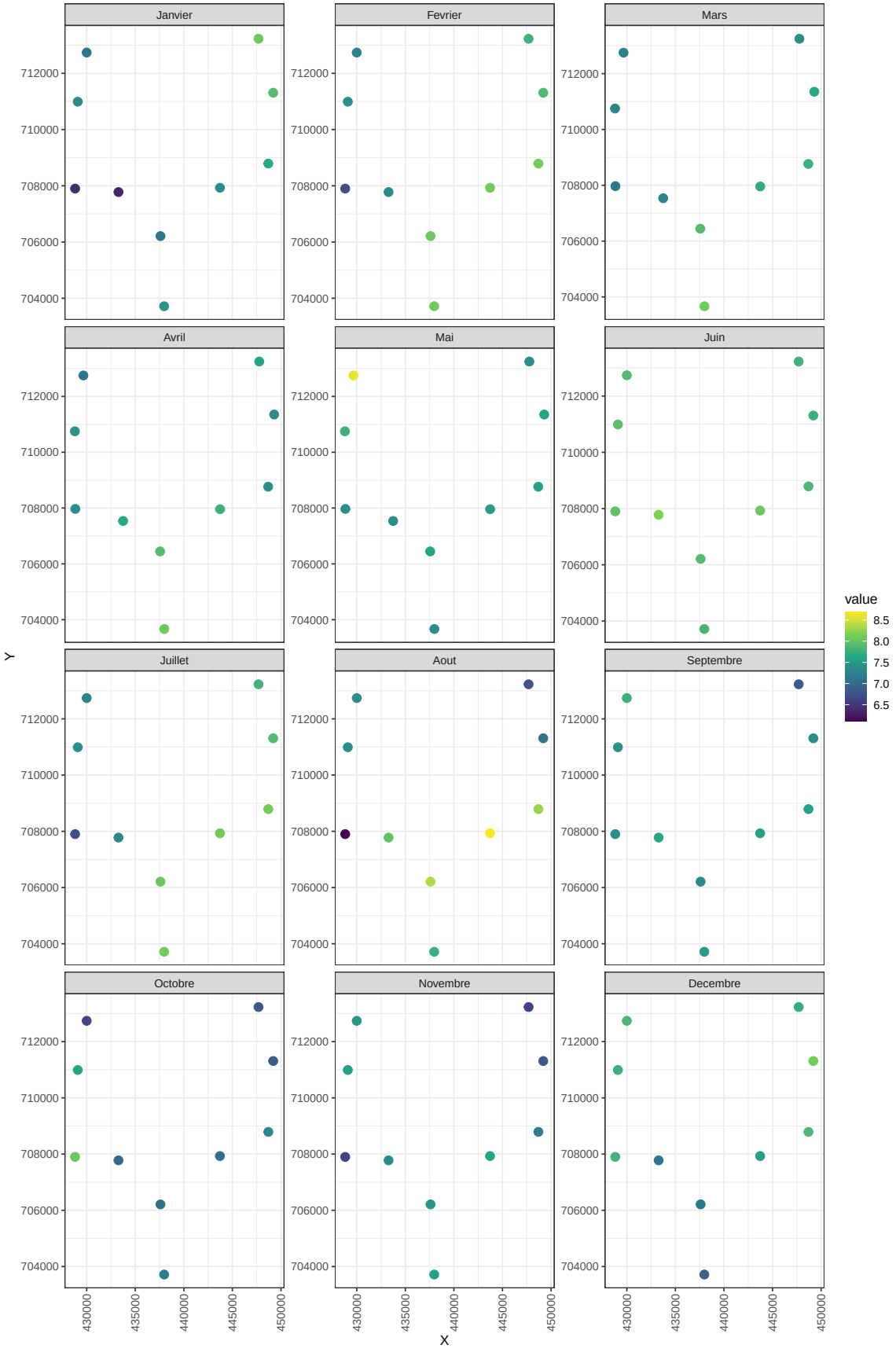
```
envdf <- subset(ldf, ldf$variable %in% subset(dfvar, dfvar$Type == 'Environment')$Parameters)

ggplot(envdf, aes(x=Mois, y = value))+
  geom_point(aes( color=Sites))+
  geom_boxplot(alpha=.7)+
  facet_wrap(vars(variable), scales = "free_y", ncol=3)+
  theme_bw()+ theme(axis.text.x = element_text(angle = 60, vjust = 0.5, hjust=1))
```

```
## Warning: Removed 191 rows containing non-finite values (`stat_boxplot()`).
```

```
## Warning: Removed 191 rows containing missing values (`geom_point()`).
```


T
pH
Sal
EC
DO
TDS
Turb
SS
Chla
PT
PO4
NO2
NO3
NH4
NTK
DCO
DBO5
value
Mois
Janvier
Fevrier
Mars
Avril
Mai
Juin
Juillet
Aout
Septembre
Octobre
Novembre
Decembre
Sites
S1
S10
S2
S3
S4
S5
S6
S7
S8
S9

```r
ggplot(subset(ldf, ldf$variable=='pH'), aes(x=X, y = Y, color=value))+
  geom_point(size=3)+
  # geom_boxplot()+
  facet_wrap(vars(Mois), scales = "free_y", ncol=3)+
  theme_bw()+ theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))+
  scale_color_viridis_c()
```


Janvier
Fevrier
Mars
Avril
Mai
Juin
Juillet
Aout
Septembre
Octobre
Novembre
Decembre
712000
710000
708000
706000
704000
430000
435000
440000
445000
450000
X
Y
value
8.5
8.0
7.5
7.0
6.5