# Time series clustering

Julien JACQUES

Introduction

Basics of Time Series Clustering

To go further

# Introduction

# Clustering

The **goal of clustering** is to create homogeneous group of obsevations, s.t.:

- observations within a group are as similar as possible
- groups are as different as possible from each other

The groups are called **clusters**.

# Use of clustering

- Clustering is an unsupervised technique.
- It aims to explore the data and to discover some typical pattern.
- It is often used as a preliminary step between supervised approach.

# The data

Our goal is to cluster time series.

For instance, the number of new cases of Covid19 in the different countries of the world.

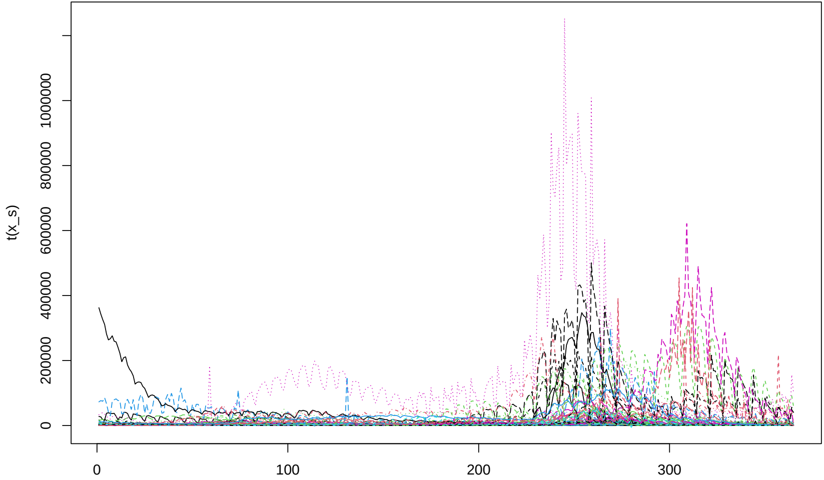
```
covid19 = read.csv("data/2022-05-13-WHO-COVID-19-global-dat
x=matrix(covid19$New_cases,ncol = 861,byrow = TRUE)
rownames(x)=unique(covid19$Country_code)
x_s=x[,(861-364):861]
x_s=x_s[rowMeans(x_s)>1000,]
```

For the example, a subset of countries having large number of cases are selected

# The data

We have 80 time series, observed on 365 points

```
matplot(t(x_s),pch=20,type='l')
```




We want to cluster them into homogeneous group

# Basics of Time Series Clustering

# Time serie clustering

An easy way is to use usual algorithms:

- kmeans,
- hierarchical clustering,

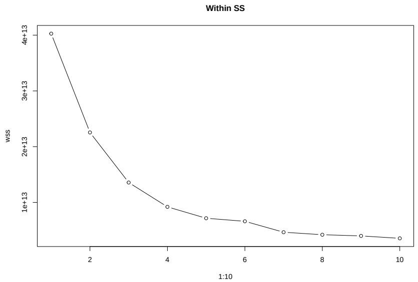
applied on a given distance for time series:

- either the usual Euclidean distance
- or specific distances as DTW

# Kmeans with Euclidean distance

We can apply the usual kmeans algorithm using the Euclidean distance between time series:

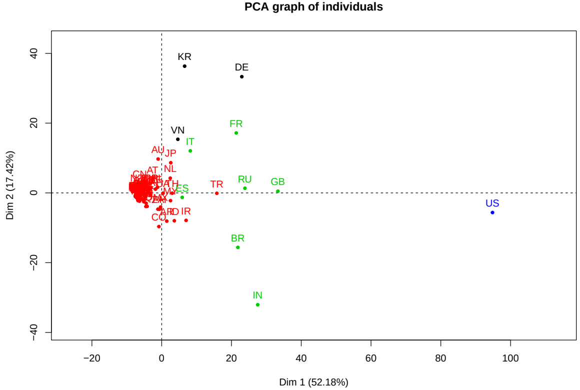
```
wss=NULL
for (k in 1:10){
  tmp=kmeans(x_s,centers=k)
  wss=c(wss,tmp$tot.withinss)
}
plot(1:10,wss,main="Within SS",type="b")
```




May be 4 clusters ?

# Kmeans with Euclidean distance

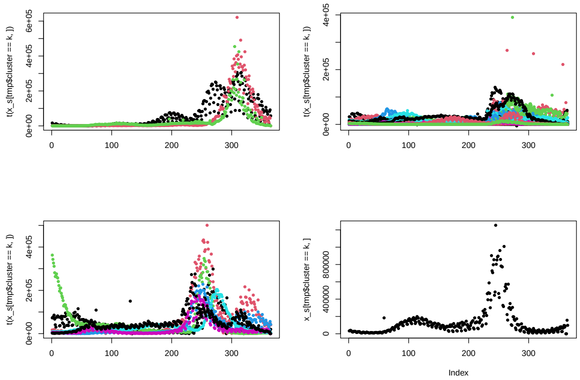
```
tmp=kmeans(x_s,centers=4)
library(FactoMineR)
pca<- PCA(x_s, graph = FALSE)
plot(pca,choix = "ind",col.ind = tmp$cluster,
     graph.type = "classic")
```



# Kmeans with Euclidean distance

Representation of the curves per cluster

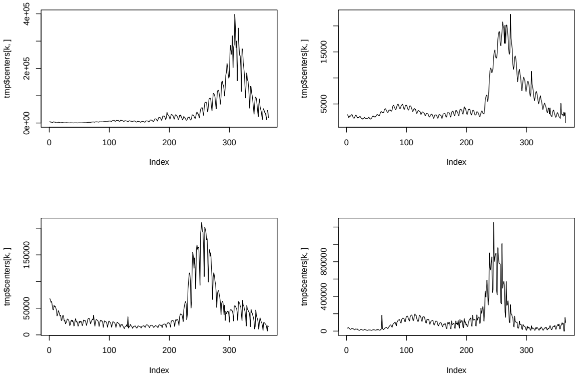
```
par(mfrow=c(2,2))
for (k in 1:4){
  if (tmp$size[k]>1){matplot(t(x_s[tmp$cluster==k,]),pch=20)}
  else{plot(x_s[tmp$cluster==k,],pch=20)}
}
```



# Kmeans with Euclidean distance

Representation of the cluster means

```
par(mfrow=c(2,2))
for (k in 1:4){
    plot(tmp$centers[k,],type='l')
}
```



# Dynamic Time Warping

The Euclidean distance is influenced by non-alignment of time series:

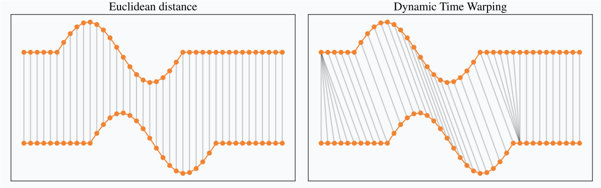



Figure from https://rtavenar.github.io/blog/dtw.html

**Dynamic Time Warping** look for the best alignment of the 2 time series.

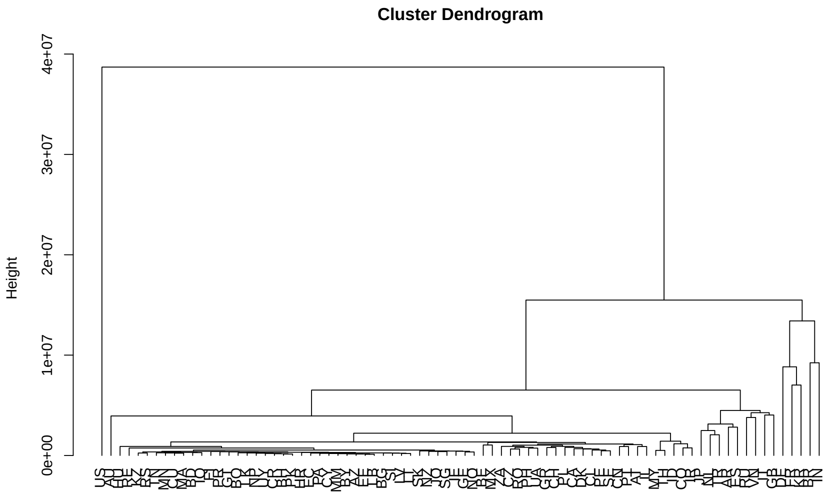
A distance can be build by measuring the distance between the best time series alignement.

```
library(dtw)
distMatrix <- dist(x_s, method="DTW")
```

# Hierarchical clustering

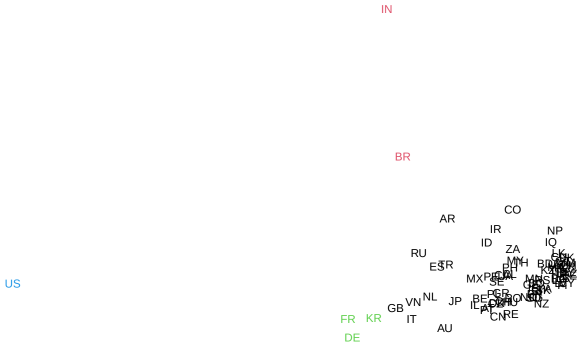
We can then apply any clustering algorithm using this DTW distance

```r
hc <- hclust(distMatrix, method="average")
plot(hc, hang = -1)
```



# Clustering representation

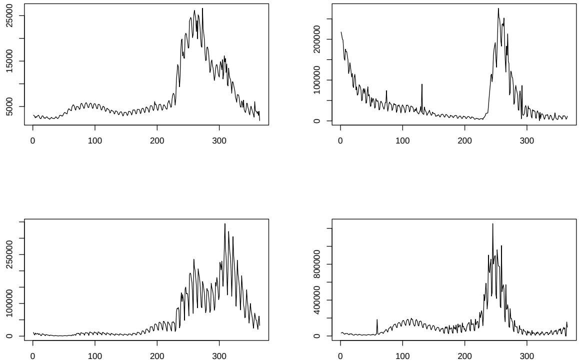
```r
cluster=cutree(hc,4)
mds2 <- -cmdscale(distMatrix)
plot(mds2, type="n", axes=FALSE, ann=FALSE)
text(mds2, labels=rownames(x_s), xpd = NA,col = cluster)
```



# Clustering representation

Representation of the cluster means

```
par(mfrow=c(2,2))
for (k in 1:4){
  if (sum(cluster==k)>1){
    plot(colMeans(x_s[cluster==k,]),type='l',xlab='',ylab='')}
  else{plot(x_s[cluster==k,],type='l',xlab='',ylab='')}
}
```

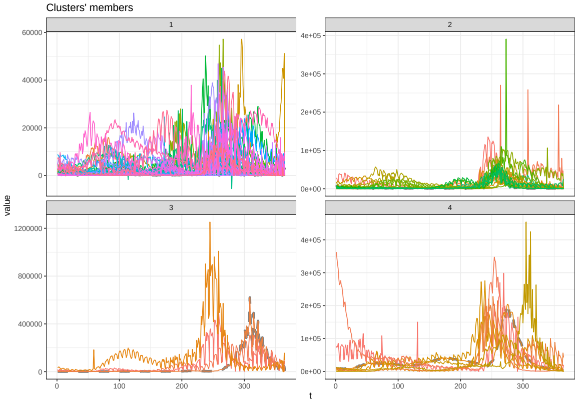


# To go further

# The dtwclust package

The following package allows different type of clustering, based on DTW distances:

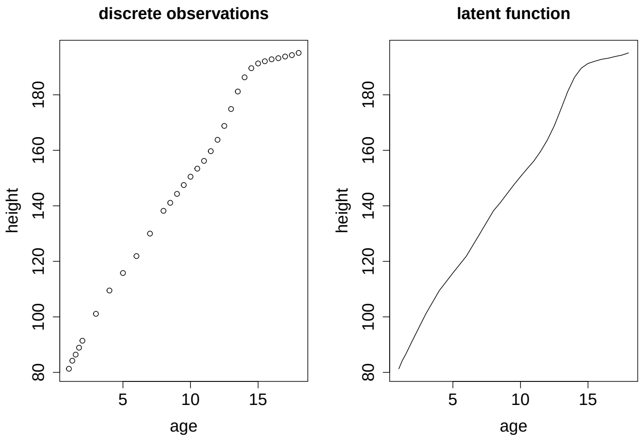
```
library(dtwclust)
tmp=tsclust(x_s,k=4)
plot(tmp)
```



# Functional data approach

An alternative way to work with time series is to assumes that $x_i(t_1), \ldots, x_i(t_m)$ are **discrete observations of a function**:

$$x_i(t) \qquad \text{with } t \in [0, T]$$




This is the **functional data approach**

# Functional data approach

The advantages of the functional data approach vs the usual multidimensional approach:

- parsimonious modelling of the curves
- allows to deal with irregularly sampled time series

# The funHDDC package

The funHDDC package provides clustering algorithm for functional data

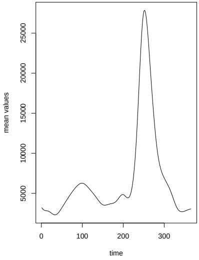
```
library(funHDDC)
basis<- create.bspline.basis(c(1,365), nbasis=25)
var<-smooth.basis(argvals=seq(1,365,length.out =365),
                  y=t(x_s),fdParobj=basis)$fd
res<-funHDDC(var,K=1:4,init='random')
```

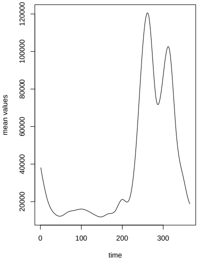
```
## funHDDC:
##       model K threshold complexity         BIC              co
## 1 AKJBKQKDK 2       0.2        103  -62,356.33
## 2 AKJBKQKDK 1       0.2         75 -236,542.16
## 3 AKJBKQKDK 3       0.2       <NA>        -Inf pop<min.indivi
## 4 AKJBKQKDK 4       0.2       <NA>        -Inf pop<min.indivi
##
## SELECTED: model  AKJBKQKDK  with  2  clusters.
## Selection Criterion: BIC.
```

# The funHDDC package

Representation of the cluster means

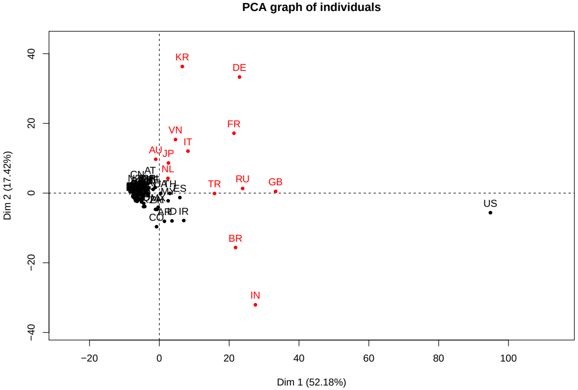
```
par(mfrow=c(1,res$K))
for (k in 1:res$K){
  plot(mean.fd(fd(var$coefs[,which(res$class==k)],
                  var$basis)))
}
```



# Clustering representation

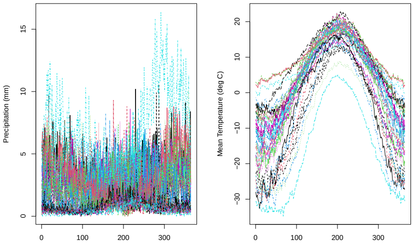
```r
plot(pca,choix = "ind",col.ind = res$class,
     graph.type = "classic")
```



# Exercice

Carry out a clustering of Canadian weather stations based on precipitation, then temperatures.

```
library(fda)
data("CanadianWeather")
par(mfrow=c(1,2))
matplot(day.5, CanadianWeather$dailyAv[, , "Precipitation.mm"],
        type="l", xlab="", ylab="Precipitation (mm)")
matplot(day.5, CanadianWeather$dailyAv[, , "Temperature.C"],
        type="l", xlab="", ylab="Mean Temperature (deg C)")
```




Is there a link with the geographical location of the cities (Atlantic, Pacific, Continental, Arctic), available in the variable *region*?