R Project - Unsupervised Learning in finding sales pattern

Machine Learning in Management - year 2022/23

Matteo Scarciglia

Contents

Starting Point: The Problem

Since entrepreneurship was born, one of the biggest problem is to "predict" what, where, how much and when a product should be sold to somebody. In the age when statistical learning did not exists, this process was the result of a pure sensational decision, made without any kind of scientific criterion and not based on what is really happened before.

Although the business rules and decisions was imprecise this way of thinking have yielded his results through that time, at least until the technology has carried the disruption. Moreover, globalization's improvement caused deeper and more complex linkage among different actors in the global economy, causing easier way to produce but at the same time increasing the difficult of selling products. The beginning hyper-competitive scenario didn't give any possibilities to whoever have continued by his way without looking for a route change nor give importance to analyzing the past.

With the arrival of the information and technology era, this mechanism changed once again, and is all became data-driven. The higher affordable technologies caused by industrial cost reduction have permitted a broader usage of these so the data production has increased a lot, as the industries have started to use this in the production's activities and over.

This new data-driven approach had with himself opportunities for all the players, starting for the business' actors. Some of these in fact have started to gather data about sales, like customer's personal feature, sale's seasonality against months or year and so on. Some other (the most advanced), have started to tidy and transform this data in information, and to make it useful for support the decisions.

Thus, the aim of this project is to try to transform this data in knowledge via some scientific proven method like machine learning is, and simulate how this knowledge generating process could be useful for finding what is invisible to the eyes. In this case, there are time series of 800 product sales across 53 weeks. What it is going to do, is to find similar time series among the given product sale's time series, for finding relationship among some products to identify seasonality or eventually complementary/ substitute products.

For demonstrate the usefulness of understanding what data want to tell, a machine learning technique is here used and it is the hierarchical clustering. This approach is a part of the Unsupervised Learning branch, which

is a subset of the machine learning science. By using this technique, is possible to cluster some observations without defining the K clusters otherwise that other machine learning techniques require to work with.

The expected results is a tree-based representation (Dendrogram) of the observations, where each leaf represent one of the observation. Starting from the bottom, moving up along the tree, these leaves begin to fuse into branches. These branches correspond to observation that are similar to each other. Further, the branches themselves fuse with leaves or other branches, and so on. The groups of observation located at a certain high, are supposed to be similar themselves to; for example, groups located in the lower part of the tree are similar each other but different compared to which form the branches in the higher position.

Data Description - Importing Dataset and Libraries

As first, all the needed libraries have to be imported. We need the following libraries to manage the problem.

```
library(tseries)
library(data.table)
library(ggplot2)
library(reshape2)
library(RColorBrewer)
library(stats)
library(dendextend)
library(colorspace)
```

Then, import the dataset. The time series is stored in Excel file (xlsx) and contains 800 products (P1 to P800) and the sales which is referred to, for 53 weeks (W0 to W51).

```
dataset <- readxl::read_excel('Sales_Transactions_Dataset_Weekly.xlsx')
dim(dataset)
```

```
## [1] 792  53
```

```
typeof(dataset)
```

```
## [1] "list"
```

```
head(dataset)
```

```
## # A tibble: 6 x 53
##   Product_Code    W0    W1    W2    W3    W4    W5    W6    W7    W8    W9   W10
##   <chr>        <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 P1              11    12    10     8    13    12    14    21     6    14    11
## 2 P2               7     6     3     2     7     1     6     3     3     3     2
## 3 P3               7    11     8     9    10     8     7    13    12     6    14
## 4 P4              12     8    13     5     9     6     9    13    13    11     8
## 5 P5               8     5    13    11     6     7     9    14     9     9    11
## 6 P6               3     3     2     7     6     3     8     6     6     3     1
## # ... with 41 more variables: W11 <dbl>, W12 <dbl>, W13 <dbl>, W14 <dbl>,
## #   W15 <dbl>, W16 <dbl>, W17 <dbl>, W18 <dbl>, W19 <dbl>, W20 <dbl>,
## #   W21 <dbl>, W22 <dbl>, W23 <dbl>, W24 <dbl>, W25 <dbl>, W26 <dbl>,
## #   W27 <dbl>, W28 <dbl>, W29 <dbl>, W30 <dbl>, W31 <dbl>, W32 <dbl>,
## #   W33 <dbl>, W34 <dbl>, W35 <dbl>, W36 <dbl>, W37 <dbl>, W38 <dbl>,
## #   W39 <dbl>, W40 <dbl>, W41 <dbl>, W42 <dbl>, W43 <dbl>, W44 <dbl>,
## #   W45 <dbl>, W46 <dbl>, W47 <dbl>, W48 <dbl>, W49 <dbl>, W50 <dbl>, ...
```

Transform, Tidy and Prepare the Dataset

Once the dataset has been imported, what is noticed first is that a transposition is needed; moreover, the dataset's type is character so it requires to be converted also. Addictionally, there are some missing products since the number of rows according to dim() function is just 792 compared to the 800 previewed.

So in order let's transpose, convert and check whether there are missing values as well as missing products:

```
t_dataset <- t(dataset) # Transpose the dataset

t_dataset[1:5,1:10] # Use this instead of head because we expect too much columns
```

```
##              [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## Product_Code "P1" "P2" "P3" "P4" "P5" "P6" "P7" "P8" "P9" "P10"
## W0           "11" " 7" " 7" "12" " 8" " 3" " 4" " 8" "14" "22"
## W1           "12" " 6" "11" " 8" " 5" " 3" " 8" " 6" " 9" "19"
## W2           "10" " 3" " 8" "13" "13" " 2" " 3" "10" "10" "19"
## W3           " 8" " 2" " 9" " 5" "11" " 7" " 7" " 9" " 7" "29"
```

```
sum(is.na(t_dataset)) # Sum missing values (Not Available)
```

```
## [1] 0
```

Fortunately there are no missing values but still the heading is not appropriate and the type is character again. The type of the dataset is character but since this is not what is needed because this kind of object cannot be transformed in a time series object, is better to provide for a "sustainable" coercion:

```
t_data <- as.data.frame(t_dataset)

t_data[1:5,1:10]
```

```
##              V1 V2 V3 V4 V5 V6 V7 V8 V9 V10
## Product_Code P1 P2 P3 P4 P5 P6 P7 P8 P9 P10
## W0           11  7  7 12  8  3  4  8 14  22
## W1           12  6 11  8  5  3  8  6  9  19
## W2           10  3  8 13 13  2  3 10 10  19
## W3            8  2  9  5 11  7  7  9  7  29
```

```
typeof(t_data)
```

```
## [1] "list"
```

Now it's a list; then let's set the first row as Header and remove it:

```
colnames(t_data)=t_data[1,] # Set the first row as Header
t_data <- t_data[-1,] # Remove the first row because now it is useless

t_data[1:5,1:10]
```

```
##    P1 P2 P3 P4 P5 P6 P7 P8 P9 P10
## W0 11  7  7 12  8  3  4  8 14  22
## W1 12  6 11  8  5  3  8  6  9  19
## W2 10  3  8 13 13  2  3 10 10  19
## W3  8  2  9  5 11  7  7  9  7  29
## W4 13  7 10  9  6  6  8  6 11  20
```

And finally the time series object could be created. The ts() function is used to create these kind of objects. In these object the data has been sampled at equispaced points in time, so the data points themselves seems different compared to the original. Moreover, the time series values need to be scaled because some time series might show a too different "magnitudo", and make them not easily comparable. The scaling process is made via the scale() function as follows:

```
ts_data <- scale(ts(t_data))

typeof(ts_data)
```

```
## [1] "double"
```

```
ts_data[1:5,1:10]
```

```
##              P1         P2         P3          P4         P5         P6
## [1,]  0.4335318  1.2772433 -0.5794039  1.11132317 -0.1350265 -0.4982192
## [2,]  0.7263066  0.8542073  0.7900962 -0.06205358 -1.0924870 -0.4982192
## [3,]  0.1407571 -0.4149007 -0.2370289  1.40466736  1.4607411 -0.9030223
## [4,] -0.4447924 -0.8379367  0.1053462 -0.94208614  0.8224341  1.1209932
## [5,]  1.0190814  1.2772433  0.4477212  0.23129061 -0.7733335  0.7161901
##               P7         P8         P9       P10
## [1,] -0.02531756 -0.2371232  0.9989861 0.4505599
## [2,]  1.73003360 -0.9624411 -0.4049943 0.0166874
## [3,] -0.46415536  0.4881948 -0.1241983 0.0166874
## [4,]  1.29119581  0.1255358 -0.9665865 1.7521773
## [5,]  1.73003360 -0.9624411  0.1565978 0.2336236
```

As noticed, the type of the object is now integer and the time series is scaled. This coercion is important because without this kind of integer object it is not possible to compute the **correlation distance**, that is used for operate the hierarchical clustering. Since the correlation distance is useful starting from now, let's calculate immediately:

```
Distance <- sqrt(1-cor(ts_data))

typeof(Distance)
```

```
## [1] "double"
```

Still, for having a "sustainable" object to work with, Distance object's type is here changed:

```
Distanze <- as.data.frame(Distance)

typeof(Distanze)
```

```
## [1] "list"
```

Lastly, this correlation explain how distant the time series are, based on their correlation. It's interpretation is the reverse of the correlation, such that while the highest is the correlation between two observation the most similar the observation's beavhiour is, in case of correlation distance is exactly the opposite. So correlation distance whose tend to zero is what should be looked for.

Some Representation

Just for a graphical little wrangling, it is always a good rule to plot some observation for let the eye notice how the data is made. In this case, the plot which will be shown is one product sales time series against another; once again, here distance is useful for finding all products that are more or less correlated among them. Down here, is looking for the maximum distance (or less correlated) between P2 (product n. 2) and the most distant.

```
names(Distanze[which.max(Distanze[,2])])
```

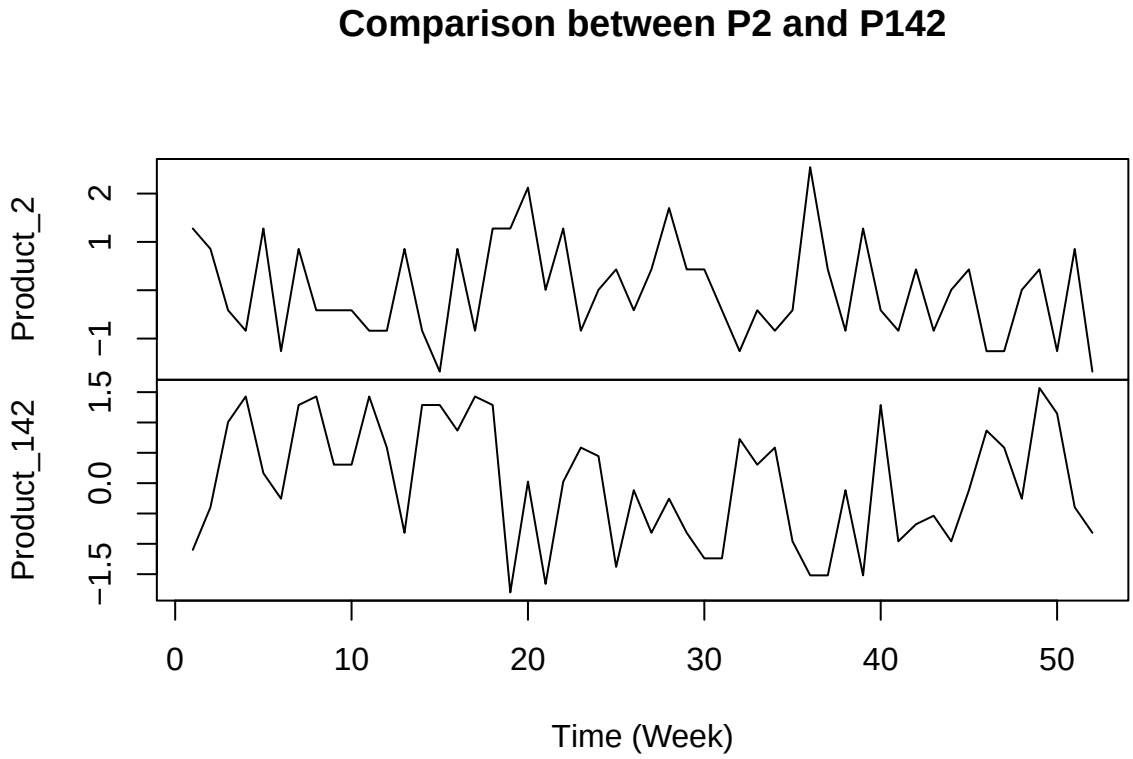
```
## [1] "P142"
```

```
sqrt(1-cor(ts_data[,2], ts_data[,142])) # Show the correlation-distance
```

```
## [1] 1.181094
```

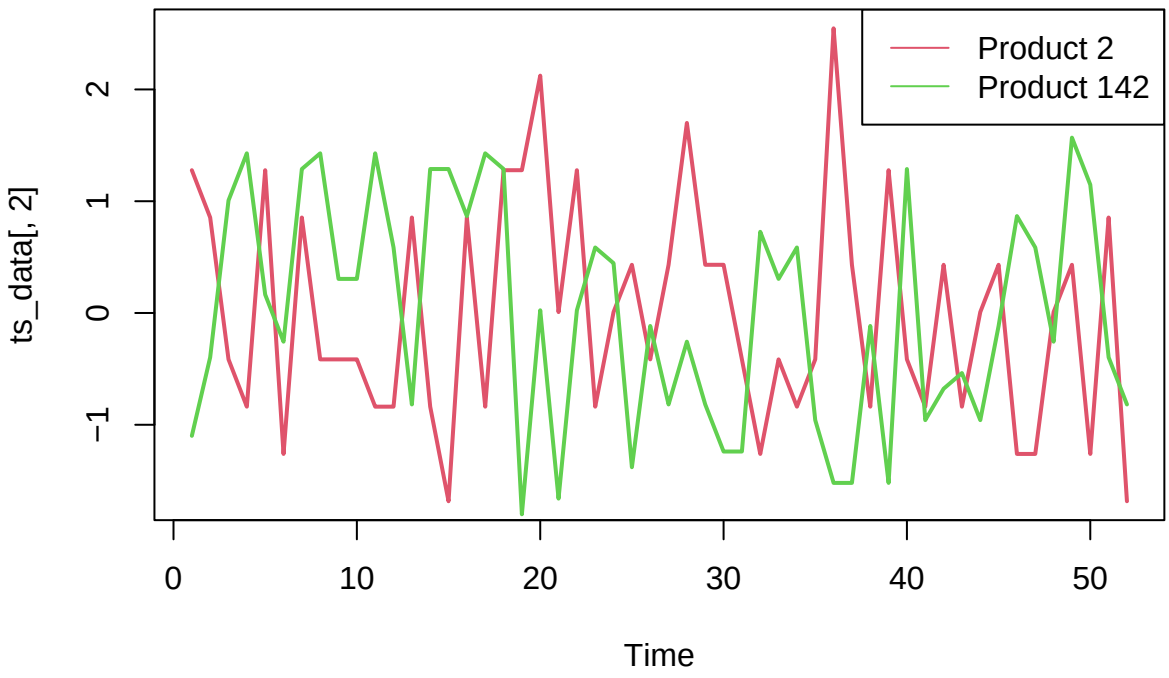
As result, P142 which stands for product n. 142 is shown. The most distant product from P2 is **P142**, and this represent a good "plot opportunity".

```
Product_2 <- ts_data[,2]
Product_142 <- ts_data[,142]

plot.ts(cbind(Product_2, Product_142),
        main="Comparison between P2 and P142", xlab="Time (Week)")
```



Another way to better see what's going on:

```
plot(ts_data[,2], type ="line", col=2, lwd=2)
lines(ts_data[,142], type="line", col=3, lwd=2)
legend("topright",c("Product 2","Product 142"), lty=1, col=2:4)
```



What is noticed is that these two products have been sold in a very different way, especially in some moments where they show an opposite peak in sales. Considering that the product is here just known as P2 and P142 it is impossible to make some rich findings; surely instead seem that these two products have been bought as substitute product to each other just like buying gold in a very high inflation period. Actually, they are not asset class but products so a major plausible hypothesis is that they could be two different labels of biscuits having a different price for meets the needs of different customers, based on their economic availability. In this case, it is possible that the sales have been the opposite for the 53 weeks as a result of a pricing policy for one of these (maybe one brand has increased the price because the receipt is changed) and the customers which are not loyal to a brand have changed their habitual consumption starting buy the biscuits that were cheaper.

Hierarchical Clustering

As said so far, the machine learning problem here is to cluster products which are similar (or very different). The seeking process is made, therefore, grouping products in a hierarchy of clusters.

Returning back on the operability, the hierarchical clustering is computed by using distance instead of euclidean distance. As first we recall the previous computed Distance but the distance which is not converted in a dataframe:

```
Distance <- sqrt(1-cor(ts_data))

Distance[1:5,1:5]
```

```
##           P1        P2        P3        P4        P5
```

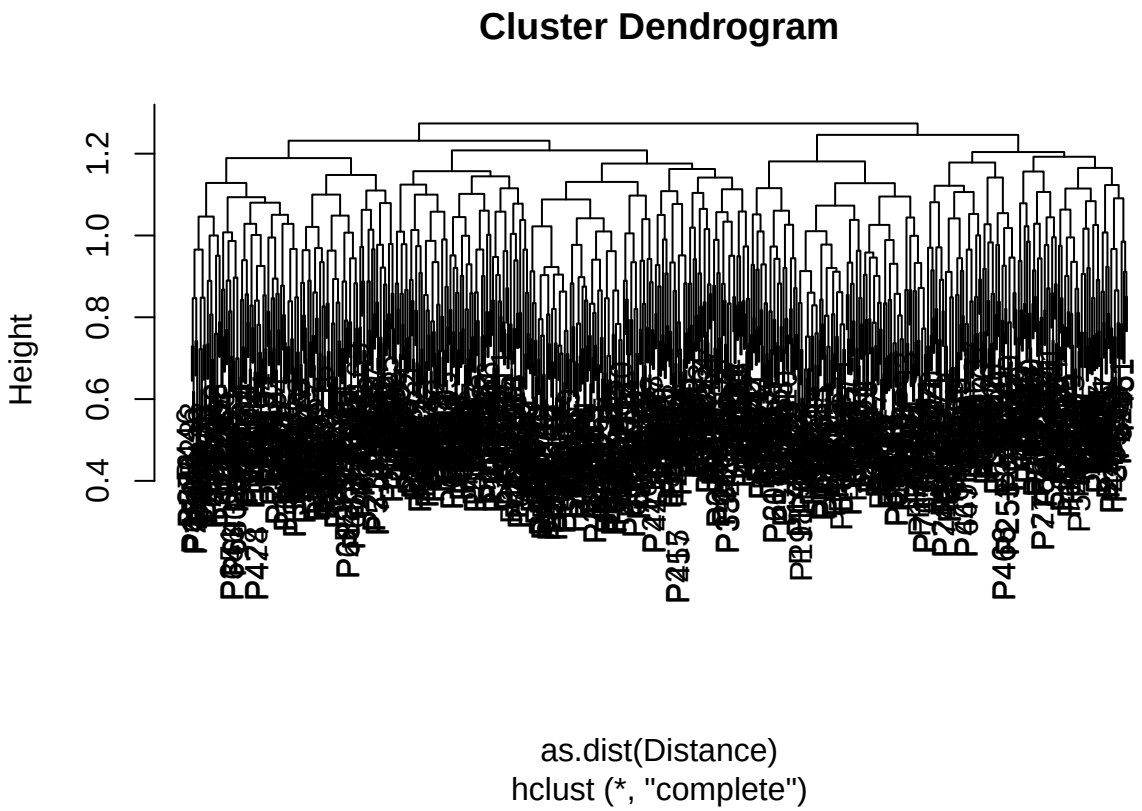
```
## P1 0.0000000 0.9443963 0.9996976 1.0164616 0.9592498
## P2 0.9443963 0.0000000 0.9818119 0.9251617 1.0384121
## P3 0.9996976 0.9818119 0.0000000 0.8090841 0.9927218
## P4 1.0164616 0.9251617 0.8090841 0.0000000 0.9904581
## P5 0.9592498 1.0384121 0.9927218 0.9904581 0.0000000
```

This distance (Distance and not Distanze) will be used to create the real clustering algorithm, who can take it's form as follows:

```
ts.clust <- hclust(as.dist(Distance), method="complete")
ts.clust
```

```
##
## Call:
## hclust(d = as.dist(Distance), method = "complete")
##
## Cluster method   : complete
## Number of objects: 792
```

```
plot(ts.clust)
```



The reading process should start from the bottom. Every leaf (every line that is an observation) converge together forming a branch, which is a group of observation grouped for their similarity (in this case the sale's behavior along the 53 weeks). Is important to notice that leafs are not at the same height, this because the height shows how similar the products are. Anyway, two products that are located at the same height

are less similar than two products located at different height but linked together forming a cluster. Hence, looking for similarity, an horizontal reading is advised at first.

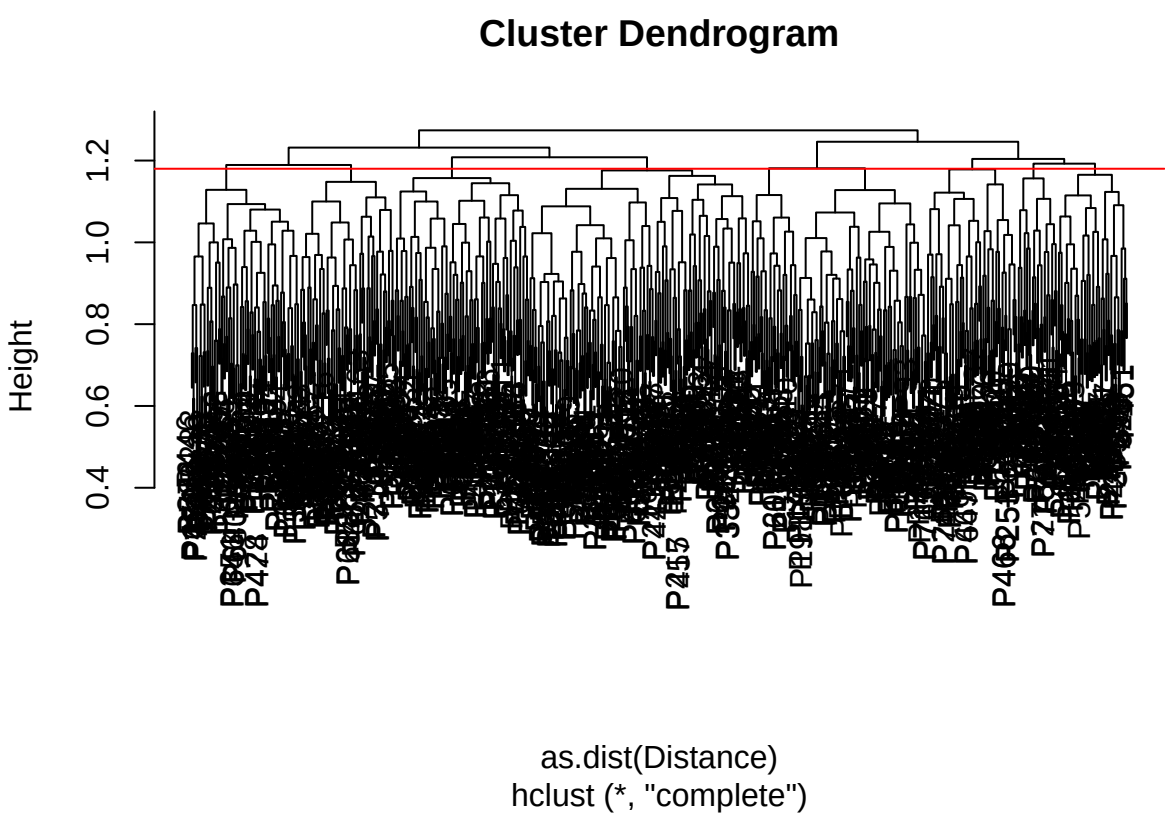
Considering the large amount of the products, every kind of reading is resulting counter-productive because there are too many products in a very little portion of space. The original plot of the dendrogram is useful for have an idea about the entire clustering only.

One way to get in touch with the real usefulness of the dendrogram is to cut it. Cutting the dendrogram is functional for a deeper analysis of how the clusters are composed and latter what observations are similar. The cutting process consists in dividing the tree at a certain height and separating the clusters according to the cut. This results definitely helpful in build other graphical object for interpreting the problem. Here, since the number of clusters is really big, it is better to have a less number but more precise usable number of clusters to work with than a big number of clusters tending to be composed by the single observation itself. By eye, a good compromise could be a cut at the height 1.18/ 1.20, which should create 10 clusters more or less:

```
ts.cutted <- cutree(ts.clust, h=1.18)

ts.cutted[1:5]
```

```
## P1 P2 P3 P4 P5
##  1  2  3  3  4
```

```
# The result is a vector which contains all product within the
#  respective cluster

plot(ts.clust) +
  abline(ts.clust, h = 1.18, col="red")
```

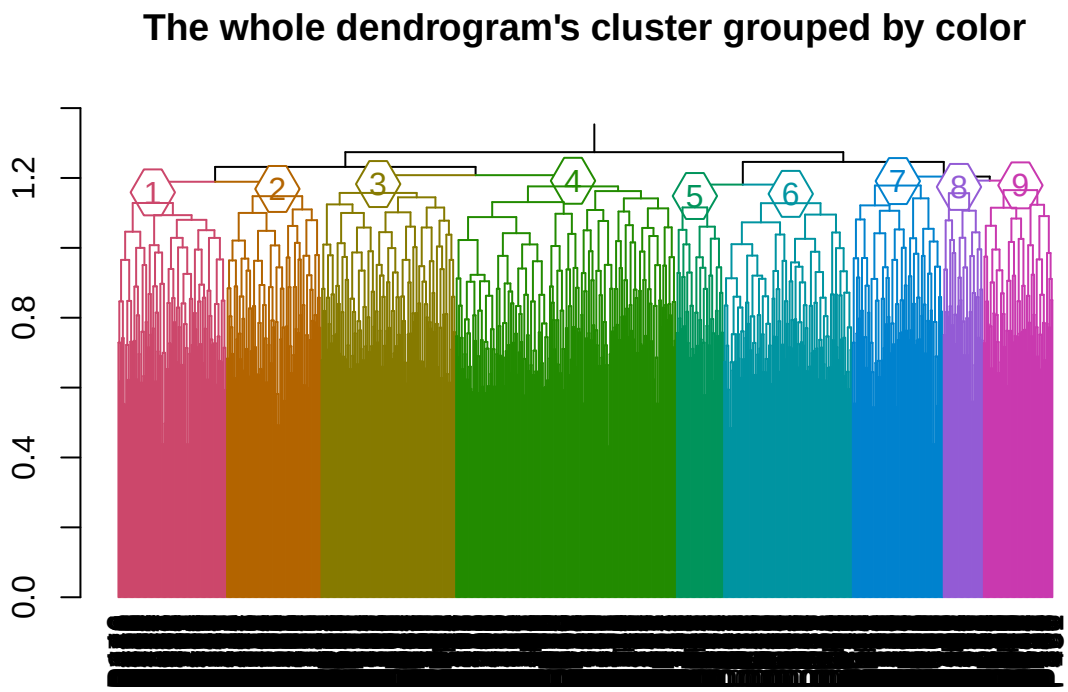



```
## integer(0)
```

Now the tree is cut, and every consideration is adaptable for a smaller group of observations.

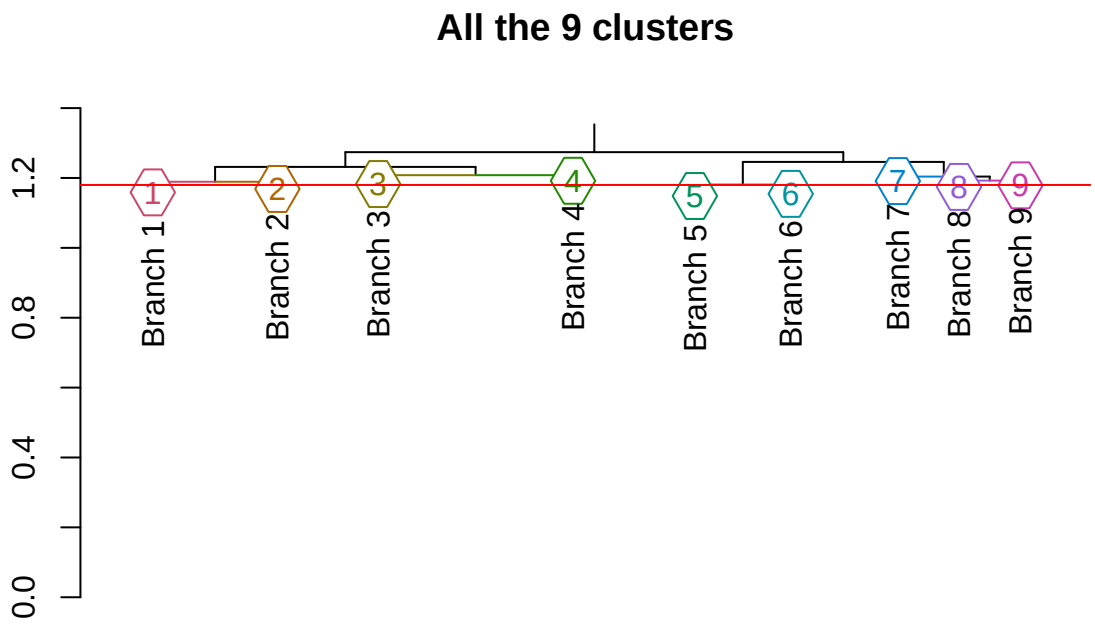
Since a more usable map is needed, it is possible to fulfill the main graph with some decorations, or to zoom in driving along the branches. Probably the first way to make it more readable is to underlying the cluster's subdivision with colors. For instance, a try is exposed here below:

```
# library(dendextend) - previously imported

# Since the plot function inside dendextend needs dendrogram objects for work, as
#  first the hclust object have to be coerced:

dend <- as.dendrogram(ts.clust)

plot(color_branches(dend, h=1.18, groupLabels = TRUE),
     main = "The whole dendrogram's cluster grouped by color",
     edge.root=TRUE)
```



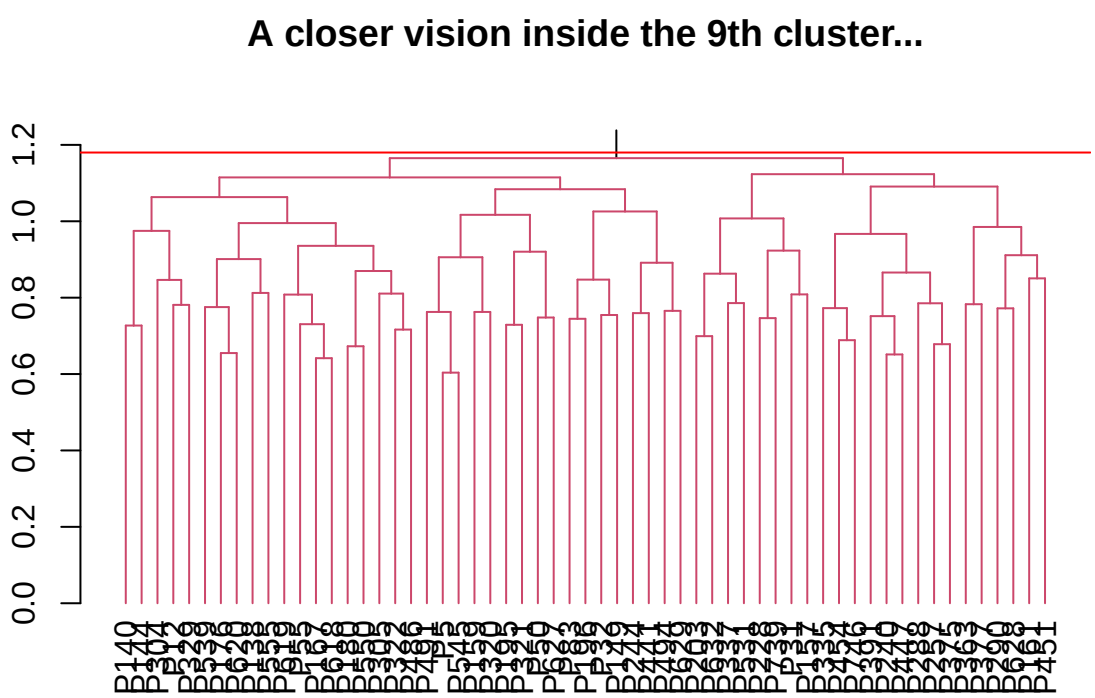
The color subdivision is resulting helpful in identify every sub-cluster standing beneath the previous simpler red line. As expected, the number of clusters is 10 but now it's possible to extend more detailed analysis to every single cluster out of 10 also. For instance, let's dive inside the cluster number 9:

```
# First of all separate the cut tree at the line's height:
Above_the_line <- cut(dend, h = 1.18)$upper
Below_the_line_9 <- cut(dend, h = 1.18)$lower[[9]]

plot(color_branches(Above_the_line, k=9, groupLabels=TRUE),
     main="All the 9 clusters",edge.root=TRUE) +
  abline(ts.clust, h = 1.18, col="red")
```




```
## integer(0)
```

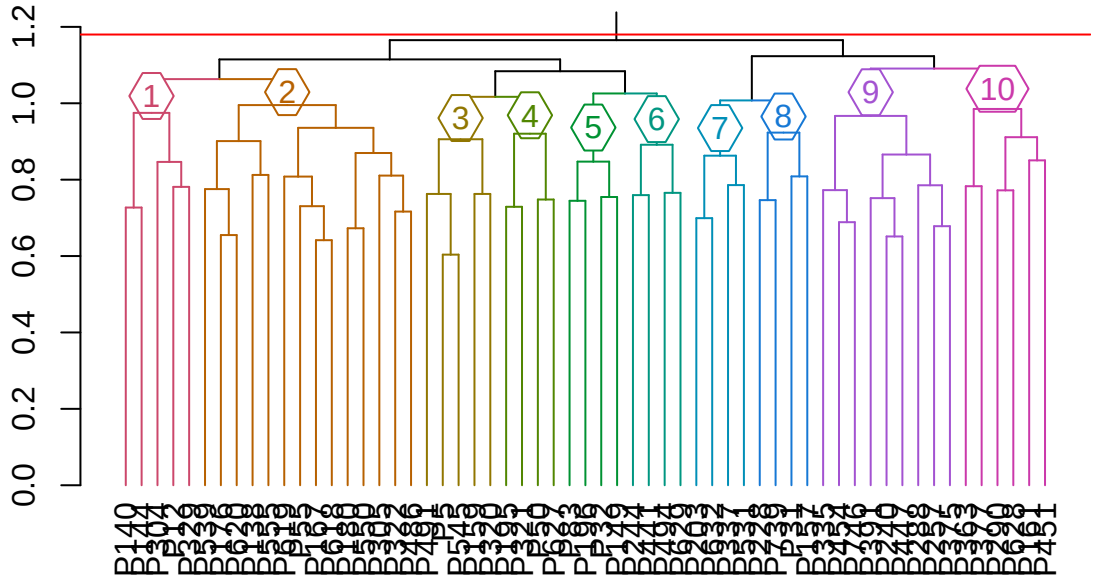
```
plot(color_branches(Below_the_line_9, k=1, groupLabels=FALSE),
     main="A closer vision inside the 9th cluster...",
     edge.root=TRUE)+
  abline(ts.clust, h = 1.18, col="red")
```




```
## integer(0)
```

Now, it is possible to cluster all the products inside the 9th cluster cutting the tree as before:

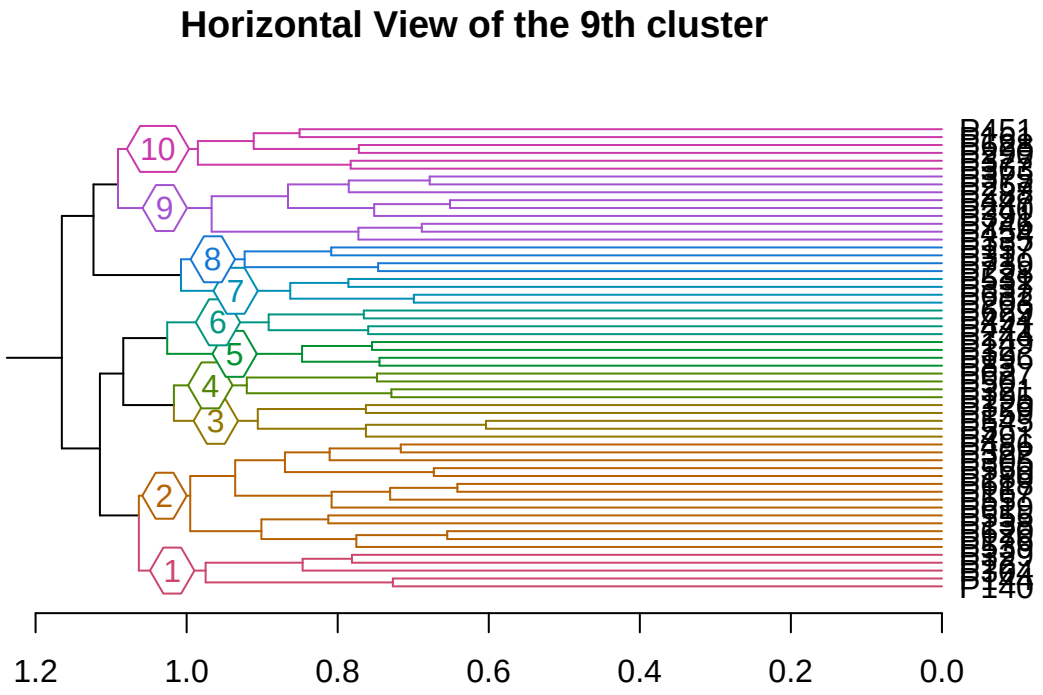
```
plot(color_branches(Below_the_line_9, h = 1, groupLabels = TRUE),
     main= "A closer vision inside the 9th (grouped) cluster...",
     edge.root=TRUE) +
  abline(ts.clust, h = 1.18, col="red")
```




```
## integer(0)
```

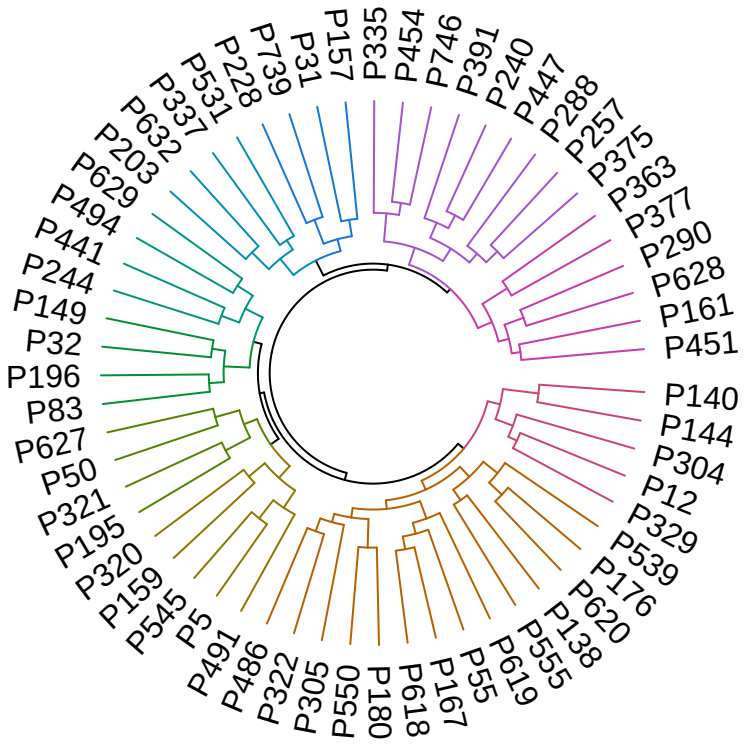
For another representation is also possible to transpose the dendrogram:

```
# library(dendextend) Previously imported

plot(color_branches(Below_the_line_9, groupLabels = TRUE, k=10), horiz=TRUE,
     center=TRUE, cex=1, edgePar = list(lty=1:2), edge.root=TRUE,
     main="Horizontal View of the 9th cluster")
```



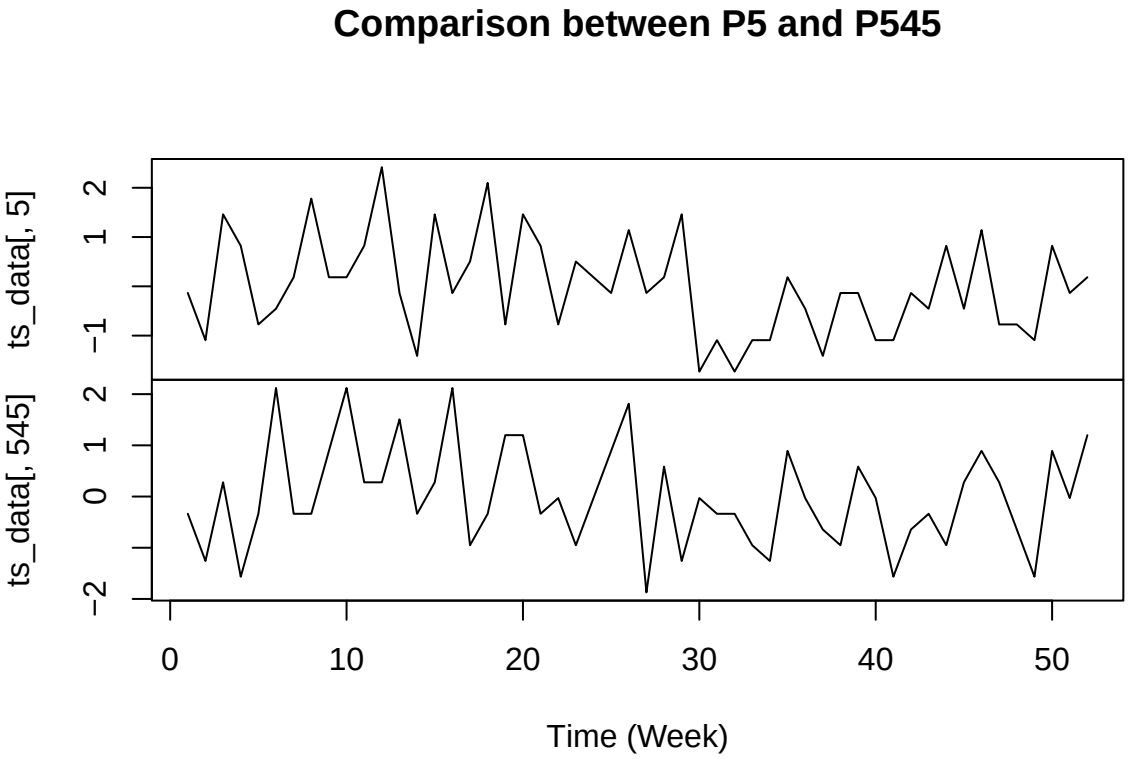
Since this interpretation of the labels is not easier compared to the previous one, another try is not too much. So go trying for a circular version, a little bit more readable:

```
library(dendextend)
library(circlize)

Below_the_line_9 <- color_branches(Below_the_line_9, k = 10)
circlize_dendrogram(Below_the_line_9, labels_track_height = NA,
                                  dend_track_height = 0.5)
```



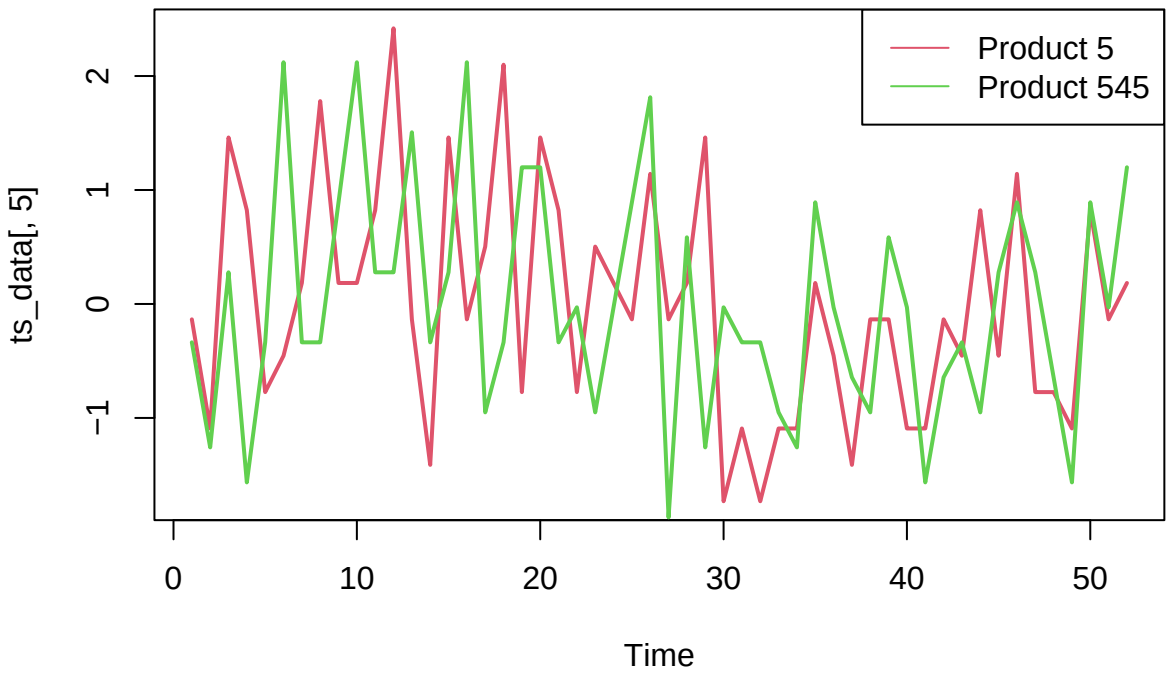
Now all the labels are easily understandable, making finally this clustering analysis adaptable to a real analysis for those that need a graphical interface useful for make decisions. For demonstrating the power of this representation, let's compare two products. Take in examination product n. 5 (P5) and product n. 545 (P545), which are in the same cluster and at the same height as the previous representation suggests:

```
plot.ts(cbind(ts_data[,5], ts_data[,545]), main="Comparison between P5 and P545", xlab="Time (Week)")
```



Or even:

```
plot(ts_data[,5], type ="line", col=2, lwd=2)
lines(ts_data[,545], type="line", col=3, lwd=2)
legend("topright",c("Product 5","Product 545"), lty=1, col=2:4)
```



As expected, these two products are quite similar. The sales had the same trend along the 53 weeks, and on the average have been the same behavior except that the product 545 have in some way anticipated the sales volume of the product 5. Since the pattern during the 53 weeks have been almost exactly the same, the most quoted consideration could be made above the seasonality of the product. Probably they are products which are bought in a certain period of time like umbrella for sea on summer which have a rise on summer (let's say June) and a decline on winter (around December). Another hypothesis is that product 5 is pencil and product 545 is notebook and people buy them in major quantity on August/ September and in a minor way during the rest of the academic year. Despite this, the consumption lasts more or less until the year ends and is plausible that they have the same service life.

Conclusion

In conclusion, hierarchical clustering is a very useful clustering method in machine learning which can be particularly useful for this kind of time series also. The knowledge generated from the algorithm is high and is usable thanks to the graphical representation of this one. Thus, the economic theory and machine learning combined can lead to a high value decision as never before.