Networks are everywhere. We have social networks like Facebook, competitive product networks or various networks in an organisation.

In the past, we used the tool [Gephi](https://gephi.org/) to visualize our results in network analysis. Impressed by this outstanding pretty and interactive visualization, our idea was to find a way to do visualizations in the same quality directly in R and present it to our customers in an R Shiny app.

Our first intention was to visualize networks with [igraph](https://igraph.org/r/), a package that contains a collection of network analysis tools with the emphasis on efficiency, portability and ease of use. We use it in the past in our [helfRlein](https://github.com/STATWORX/helfRlein) package for the function getnetwork, described in [this](https://www.statworx.com/de/blog/flowcharts-of-functions/) blog post. Unfortunately, igraph can create beautiful network visualizations, but they’re solely static. To build interactive network visualizations, you can use particular packages in R that are all using javascript libraries.

Our favorite package for this visualization task is **visNetwork**, which uses [vis.js](http://visjs.org/) javascript library and is based on [htmlwidgets](http://www.htmlwidgets.org/). It’s compatible with Shiny, R Markdown documents and RStudio viewer. visNetwork has many adjustments to personalize your network, a pretty output and good performance, which is very important when using the output in Shiny. Furthermore, you can find excellent documentation [here](http://datastorm-open.github.io/visNetwork/).

So let us go through the steps that have to be done from your data basis up till the perfect visualization in R Shiny. To do so, we use the Les Misérables Characters network in the following as an example. This undirected network contains co-occurrences of characters in Victor Hugo’s novel ‚Les Misérables‘. A node represents a character, and an edge between two nodes shows that these two characters appeared in the same chapter of the book. The weight of each link indicates how often such a co-appearance occurred.

**Data Preparation**

First of all, we have to install the package with install.packages("visNetwork") and load the dataset lesmis. You can find the dataset in the package geomnet.

To visualize the network between the Les Miserables characters, the package visNetwork needs two data frames. One for the nodes and one for the edges of the network. Fortunately, our loaded data provides both, and we only have to bring them in the right format.

rm(list = ls())

# Libraries ---------------------------------------------------------------

library(visNetwork)

library(geomnet)

library(igraph)

# Data Preparation --------------------------------------------------------

#Load dataset

data(lesmis)

#Nodes

nodes <- as.data.frame(lesmis[2])

colnames(nodes) <- c("id", "label")

#id has to be the same like from and to columns in edges

nodes$id <- nodes$label

#Edges

edges <- as.data.frame(lesmis[1])

colnames(edges) <- c("from", "to", "width")

The following function needs specific names for the columns to detect the right column. For this purpose, edges must be a dataframe with at least one column that indicates in which node an edge starts (from) and where it ends (to). For the nodes, we require at a minimum a unique ID (id) which has to coincide to the from and to entries.

**Nodes:**

* label: A column that defines how a node is labelled
* value: Defines the size of a node inside the network
* group: Assigns a node to a group; this can be a result of a cluster analysis or a community detection
* shape: Defines how a node is presented. For example as a circle, square or triangle
* color: Defines the color of a node
* title: Sets the tooltip, which occurs when you hover over a node (this can be HTML or character)
* shadow: Defines if a node has a shadow or not (TRUE/FALSE)

**Edges:**

* label, title, shadow
* length, width: Defines the length/width of an edge inside the network
* arrows: Defines where to set a possible arrow on the edge
* dashes: Defines if the edges should be dashed or not (TRUE/FALSE)
* smooth: Smooth lines (TRUE/FALSE)

These are the most important settings. They were made for every single node or edge particularly. To set some configurations for all nodes or edges like the same shape or arrows you can do this later when you specify the output with visNodes and visEdges. We’ll show you this possibility later on.

Additionally, we want to have a more interesting network with groups inside. We’ll highlight the groups later by adding colors to the edges in the network. Therefore we cluster the data with the community detection method Louvain and get a group column:

#Create graph for Louvain

graph <- graph\_from\_data\_frame(edges, directed = FALSE)

#Louvain Comunity Detection

cluster <- cluster\_louvain(graph)

cluster\_df <- data.frame(as.list(membership(cluster)))

cluster\_df <- as.data.frame(t(cluster\_df))

cluster\_df$label <- rownames(cluster\_df)

#Create group column

nodes <- left\_join(nodes, cluster\_df, by = "label")

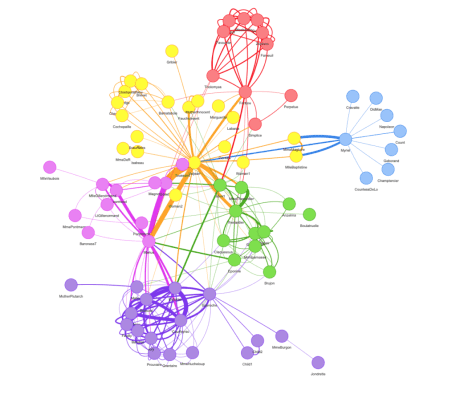
colnames(nodes)[3] <- "group"

**Output Options**

To give you an impression which possibilities we have when it comes to the design and functional options when creating our output, we will have a more in-depth look at two presentations of the Les Misérables network.

We’re starting with the easiest possibility and only give the nodes and edges dataframes to the function:

visNetwork(nodes, edges)



Using the pipe operator we can customize our network with some other functions like visNodes, visEdges, visOptions, visLayout or visIgraphLayout:

visNetwork(nodes, edges, width = "100%") %>%

visIgraphLayout() %>%

visNodes(

shape = "dot",

color = list(

background = "#0085AF",

border = "#013848",

highlight = "#FF8000"

),

shadow = list(enabled = TRUE, size = 10)

) %>%

visEdges(

shadow = FALSE,

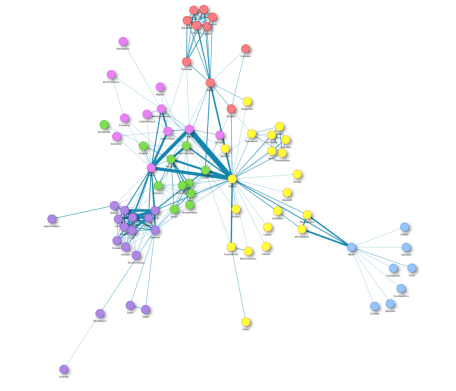
color = list(color = "#0085AF", highlight = "#C62F4B")

) %>%

visOptions(highlightNearest = list(enabled = T, degree = 1, hover = T),

selectedBy = "group") %>%

visLayout(randomSeed = 11)



visNodes and visEdges describe the overall appearance of the nodes and edges in the network. For example, we can set the shape of all nodes or define the colors of the edges.

When it comes to publication in R, rendering the network can take a long time. To deal with this issue, we use the visIgraphfunction. It decreases plotting time while computing coordinates in advance and provides all available igraph layouts.

With visOptions we can adjust how the network reacts when we interact with it. For example, what happens if we click on a node.

visLayout allows us to define the look of the network. Should it be a hierarchical one or do we want to improve the layout with a special algorithm? Furthermore, we can provide a seed (randomSeed), so that the network always looks the same when you load it.

These are only some example functions of how we can customize our network. The package provides much more options for customization. For more details have a look at the [documentation](http://datastorm-open.github.io/visNetwork/).

**Shiny Integration**

To present the interactive results to our customers, we want to integrate them into a Shiny app. Therefore we prepare the data „offline“, save the nodes and edges files and create the output inside the Shiny app „online“. Here is a minimal example code of the scheme you can use for Shiny:

**global.R:**

library(shiny)

library(visNetwork)

**server.R:**

shinyServer(function(input, output) {

output$network <- renderVisNetwork({

load("nodes.RData")

load("edges.RData")

visNetwork(nodes, edges) %>%

visIgraphLayout()

})

})

**ui.R:**

shinyUI(

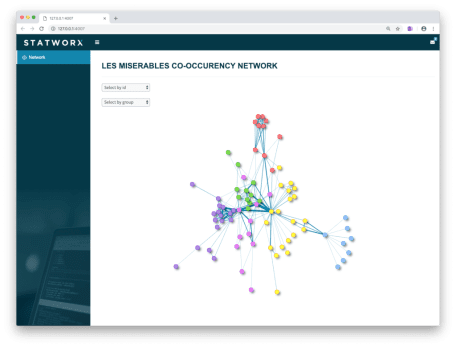
fluidPage(

visNetworkOutput("network")

)

)

A screenshot of our Shiny app illustrates a possible result:



**Conclusion**

Besides other available packages to visualize networks interactively in R, visNetwork is our absolute favorite. It is a powerful package to create interactive networks directly in R and publish it in Shiny. We can integrate our networks directly into our Shiny application and run it with a stable performance when using the visIgraphLayout function. We would not need external software like Gephi anymore.

Did I spark your interest to visualize your own networks? Feel free to use my [code](https://github.com/STATWORX/blog/tree/master/interactive_network_visualization) or contact me and visit the github page of the used package [here](https://github.com/datastorm-open/visNetwork).

**References**

Knuth, D. E. (1993) „Les miserables: coappearance network of characters in the novel les miserables“, *The Stanford GraphBase: A Platform for Combinatorial Computing*, Addison-Wesley, Reading, MA