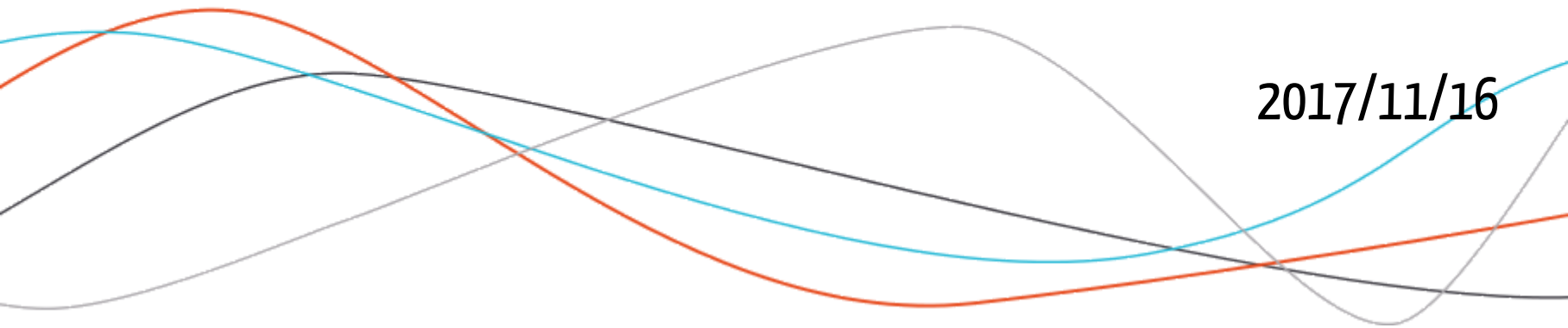
A Gentle Introduction to Network Visualisation

Colin FAY - ThinkR

2017/11/16



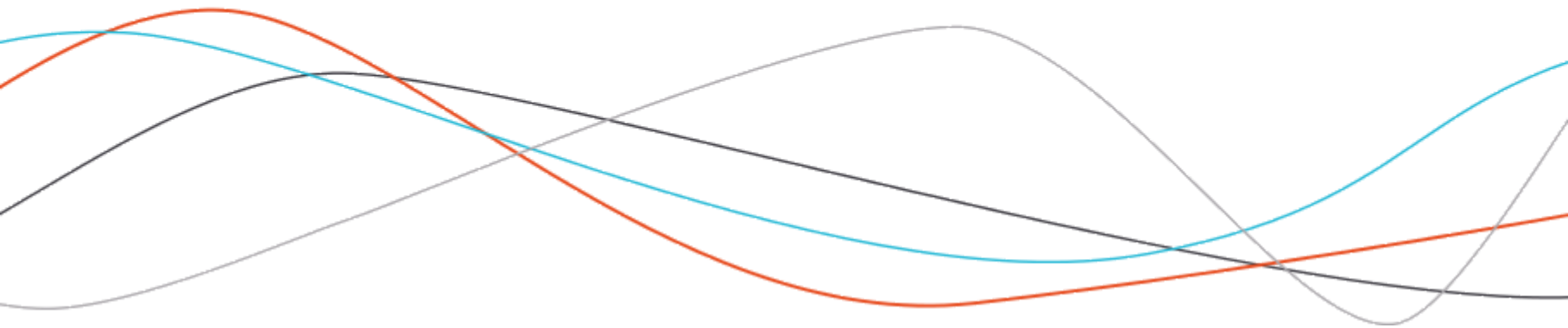
$ whoami

Colin FAY

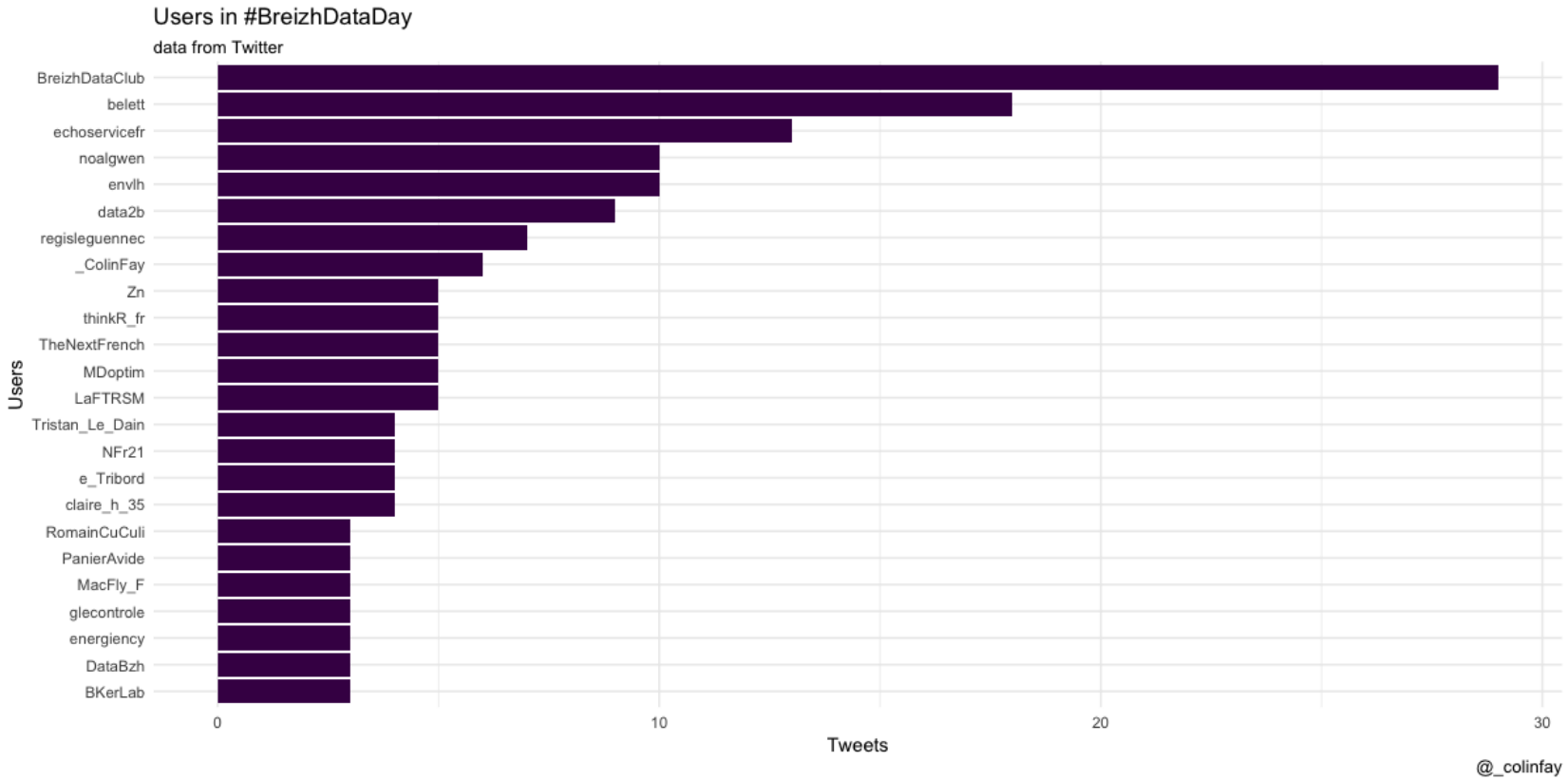
I'm a Data Analyst, R trainer and Social Media Expert at ThinkR, a French agency focused on everything R-related.

- http://thinkr.fr
- http://twitter.com/thinkr_fr
- http://twitter.com/_colinfay
- http://github.com/colinfay

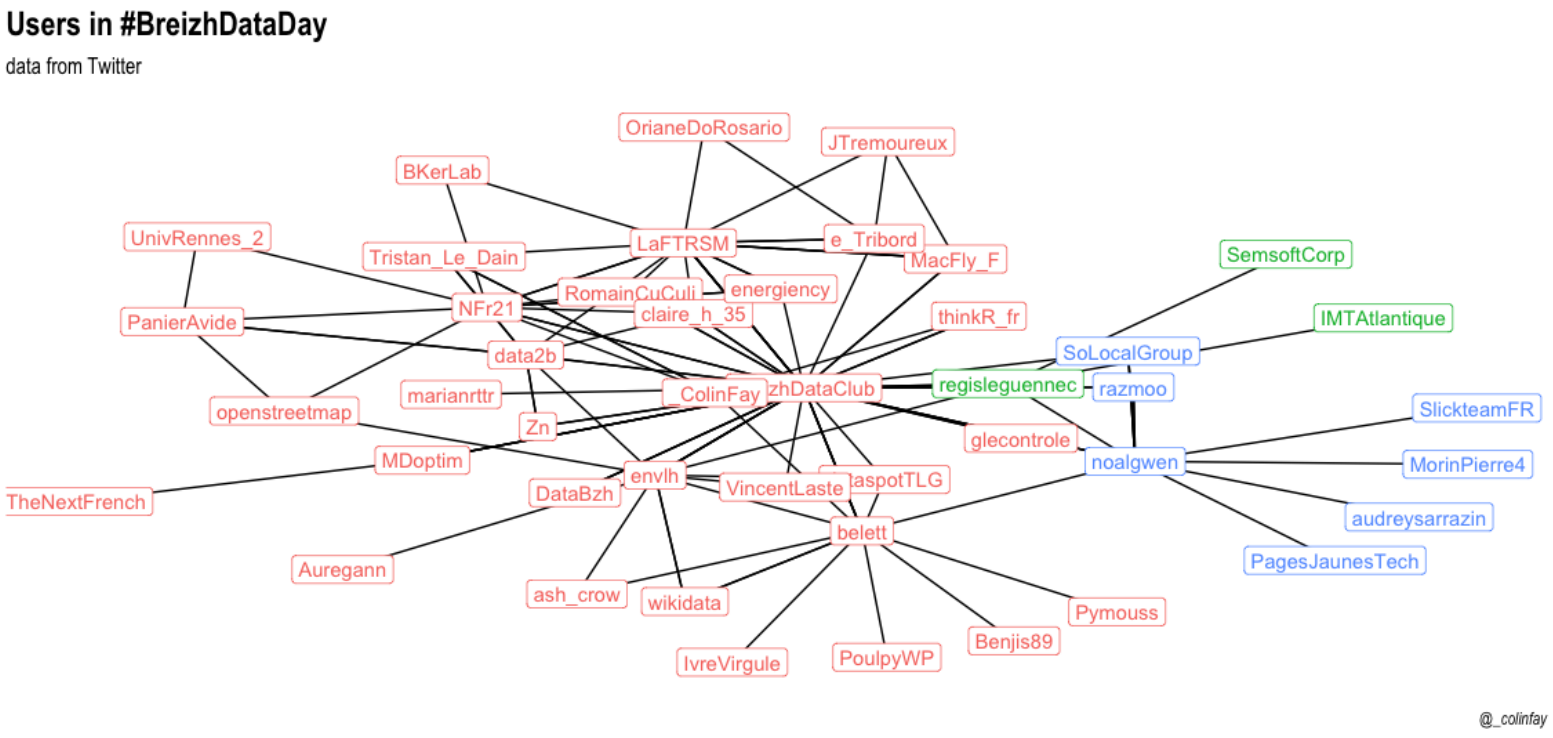
What are we going to talk about today?



Moving from tabular viz...



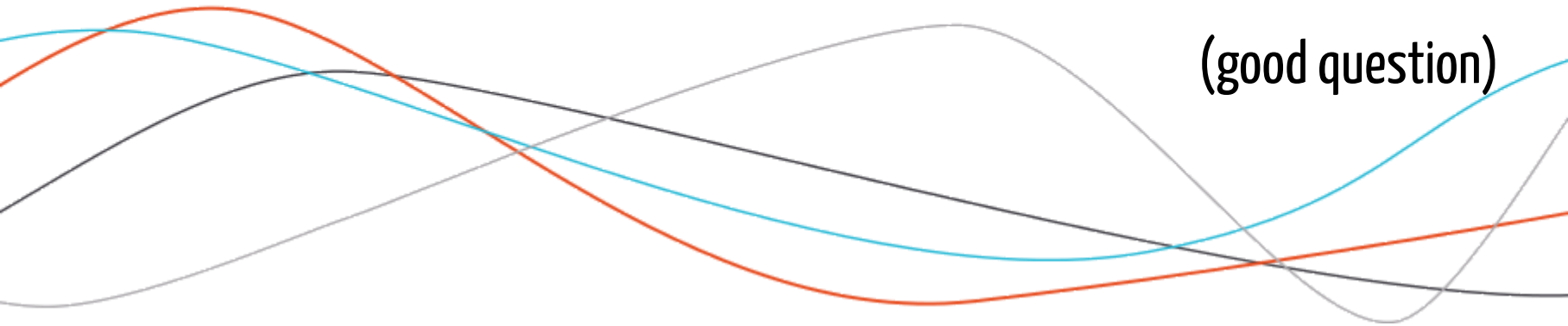
... to Network Visualisation



digression::on()

Why bother?

(good question)



Not your usual tabular plot

Classical plots

As we've just seen, what we can call a "tabular plot" draws your dataset one observation at a time, using the variables to describe each unit.

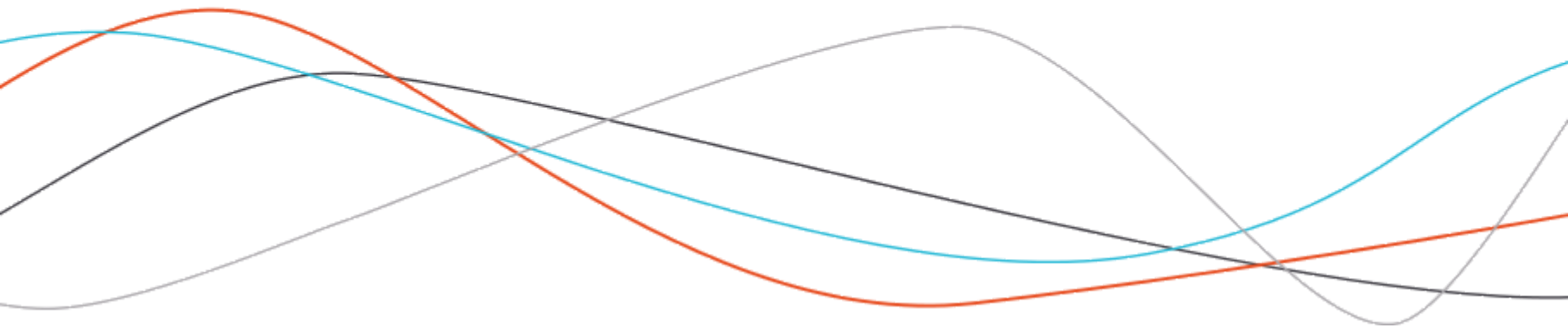
Networks

But we sometimes need more complex structures, which can describe the relationships between observations.

You need to think differently when you're making a network visualisation.

But good news: this is what we're going to talk about today.

digression::off()



What can networks be used for?

In theory

- Key actors
- Link between actors
- Strength of relationship between actors
- Movements inside a dataset
- Cluster / Communities

IRL

- Mapping a conversation on Twitter
- Relationships between words in a corpus
- Customers reviews
- Behaviours on a website
- Cluster / Communities

What's in a graph?

Basically, a network (also called a graph) is composed of two main elements:

- **Points**, which are also called Nodes or Vertices
- **Edges**, which are also called Arcs or Lines

We also can "play" (i.e customise) with:

- Layouts, which is the way the elements are organised in the resulting plot.
- Connections, which are how the different points are connected together, that is to say what are the parameters defining the edges.

What's in a graph?

Point / Nodes / Vertices

A point is the basic unit of a graph. Points are the entities which are connected with each others.

It can be thought of as an anatomic element with no internal structure.

Edge / Arc / Lines

Together with points, edges are one of the two key components of graphs.

They are used to describe the relationship between two points.

They can have a specific direction, or be undirected. Directed edges are also called arrows.

What's in a graph?

Layout

Layouts are the way elements are organised in your plotting space.

In other words, a layout algorithm defines where to draw the points and edges, in order to avoid crossing, overlapping, and so on...

In short, a layout algorithm is an algorithm that transforms network data into x and y positions, in order to find the best overall organisation.

What can we control?

Same as usual plots

- Colors
- Positions
- Sizes (in graph theory, size is called Degree)
- Shapes

Network specific

- Links between elements
- Directions of the links
- Layouts

Network visualisation, what to look for

When we draw a graph, we have to be careful about:

Readability

- Prevent edge crossing
- Prevent node overlap

Harmony

- Find an uniform and meaningfull length for the edges
- Find the best symmetry for the whole graph

Good news

You might be thinking "oh god, that seems like a lot of code..."

The good news is: this has already been coded for you 🎉

Some packages to do network visualisation:

Today, we'll see :

```
library(igraph)
library(ggraph)
library(visNetwork)
```

Creating a network visualisation with igraph

{igraph} is a package designed to deal with everything graph related (i.e, for network analysis).

You can (obviously) create graph structures, but there are also a lot of functions to compute information about your graph:

- clusters
- get nodes and edges attributes

...

For more info:

```
help("igraph")
```

Creating a network visualisation with igraph

Step 0: data collection

Graph for text mining:

```
library(tidyverse)
library(proustr)
library(tidytext)
pr_bigrams <- proust_books() %>%
  unnest_tokens(bigram, text, token = "ngrams", n = 2) %>%
  separate(bigram, c("from", "to"), sep = " ") %>%
  filter(!from %in% proust_stopwords()$word) %>%
  filter(!to %in% proust_stopwords()$word) %>%
  count(from, to, sort = TRUE) %>%
  filter(n > 30)
```

Creating a network visualisation with igraph

Step 1: turning your data.frame into a graph

The function `graph_from_data_frame` from the {igraph} package transforms your tabular data into an igraph object.

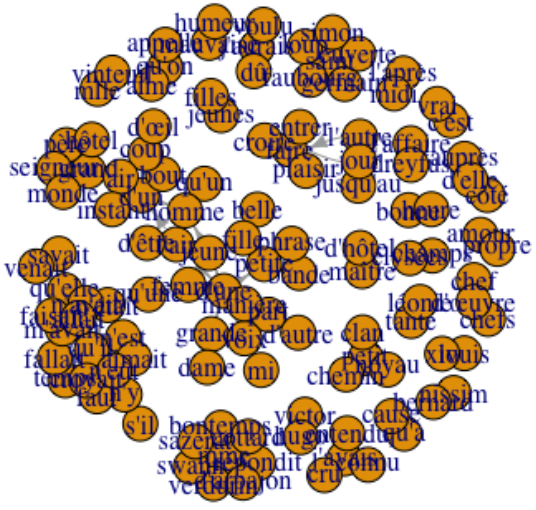
=> By default (i.e. if the `vertices` arg is NULL), the first two columns are considered as the edge lists, and the following columns as edge attributes.

=> By default (i.e. if the `directed` arg is TRUE), the graph is considered as a directed graph, i.e. the first column is the FROM, the second column the TO.

```
pr_graph <- graph_from_data_frame(pr_bigrams)
```

Step 2: plot

```
plot(pr_graph)
```



Step 3: prettify a little bit

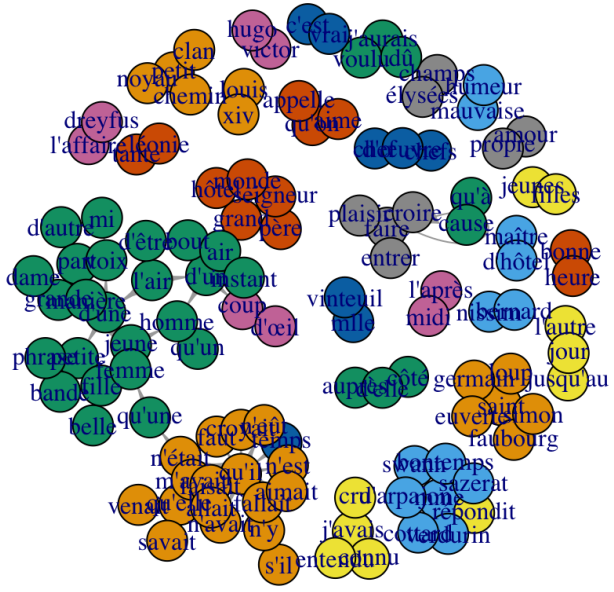
We can use {igraph} `cluster_edge_betweenness` function to return the clusters of our graph data.

```
pr_clust <- cluster_edge_betweenness(pr_graph)
pr_clust
```

```
#> IGRAPH clustering edge betweenness, groups: 31, mod: 0.82
#> + groups:
#>   $`1`
#>   [1] "saint"    "faubourg" "loup"     "germain"  "euverte"  "simon"
#>
#>   $`2`
#>   [1] "mme"       "verdurin"  "swann"     "bontemps"  "cottard"
#>   [6] "sazerat"   "d'arpajon"
#>
#>   $`3`
#>    [1] "d'un"    "jeune"   "d'autre" "d'une"   "petite"  "bout"
#>    [7] "grande"  "l'air"   "qu'un"   "qu'une"  "belle"   "mi"
#>   + ... omitted several groups/vertices
```

Step 3: prettify a little bit

```
plot(pr_graph, vertex.color = pr_clust$membership)
```



Moving to ggraph

About {ggraph}

This package has been designed to work flawflessly with {ggplot2}.

If you're already familiar with the {ggplot2} grammar, you'll be able to render {ggraph} plots in a matter of minutes.

Step 1: from base plot to ggraph

You can construct your graph with {ggraph} just like your regular {ggplot2} plots, that is to say with a ggraph() call, followed by geom layers.

There are a lot of "geoms" you can use, but here are the most common:

For the edge

- `geom_edge_link`: edge as straight lines between nodes
- `geom_edge_arc`: edge as arc between nodes
- `geom_edge_density`: add density to your plot

For the nodes

- `geom_node_point`: nodes as points
- `geom_node_text`: nodes as text
- `geom_node_label`: nodes as label

As said before, {ggraph} has been developped to work with the {ggplot2} grammar, so the layers are to be stacked with +.

Step 1: creating the graph object

We need to pass to `ggraph` an igraph object, which we've created before with `graph_from_data_frame()`

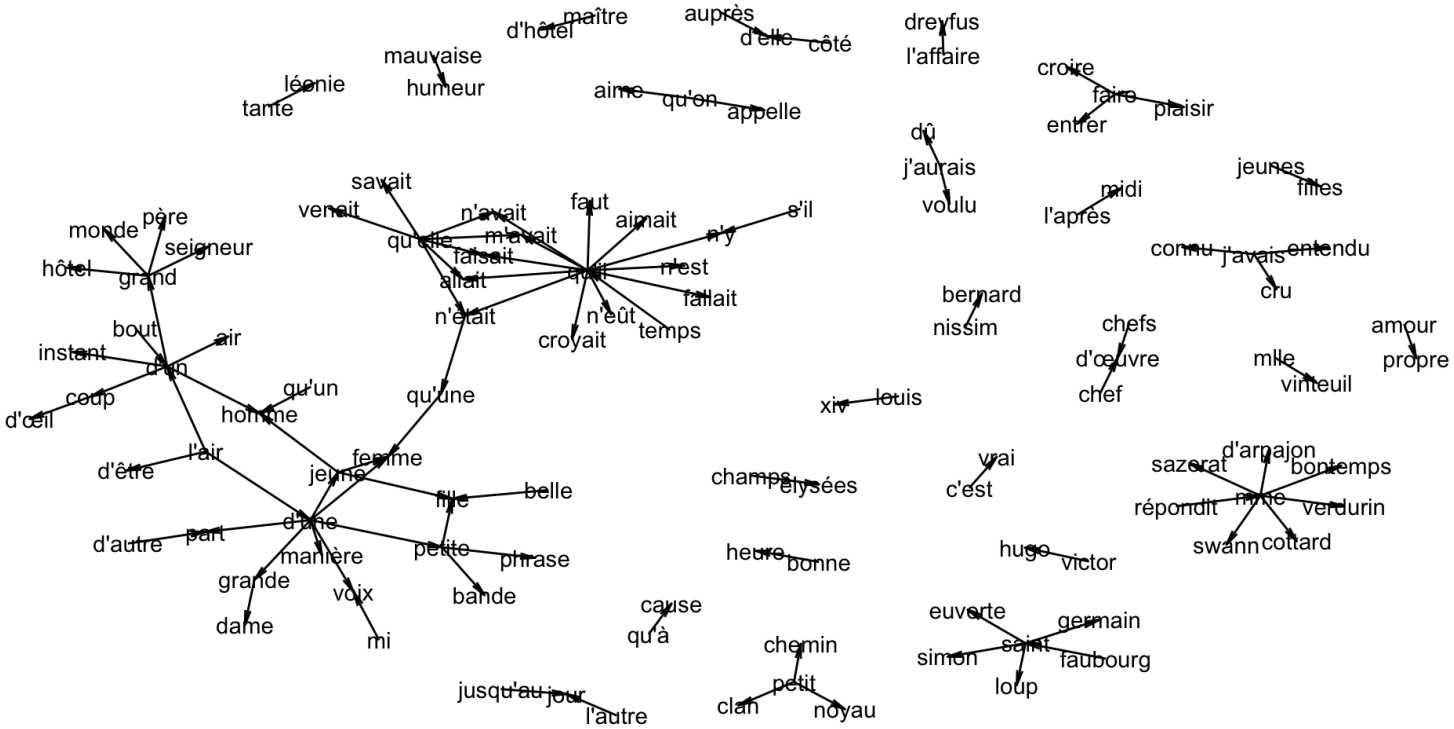
```
str(pr_graph)
```

```
#> List of 10
#>  $ :List of 1
#>   ..$ saint:Class 'igraph.vs'  atomic [1:4] 50 68 72 109
#>   .. .. ..- attr(*, "env")=<weakref>
#>   .. .. ..- attr(*, "graph")= chr "0820bdb7-4590-4465-87a8-37896edb4577"
#>  $ :List of 1
#>   ..$ mme:Class 'igraph.vs'  atomic [1:6] 51 52 64 69 90 95
#>   .. .. ..- attr(*, "env")=<weakref>
#>   .. .. ..- attr(*, "graph")= chr "0820bdb7-4590-4465-87a8-37896edb4577"
#>  $ :List of 1
#>   ..$ d'un:Class 'igraph.vs'  atomic [1:5] 8 49 53 56 105
#>   .. .. ..- attr(*, "env")=<weakref>
#>   .. .. ..- attr(*, "graph")= chr "0820bdb7-4590-4465-87a8-37896edb4577"
#>  $ :List of 1
#>   ..$ jeune:Class 'igraph.vs'  atomic [1:3] 54 56 63
#>   .. .. ..- attr(*, "env")=<weakref>
#>   .. .. ..- attr(*, "graph")= chr "0820bdb7-4590-4465-87a8-37896edb4577"
```

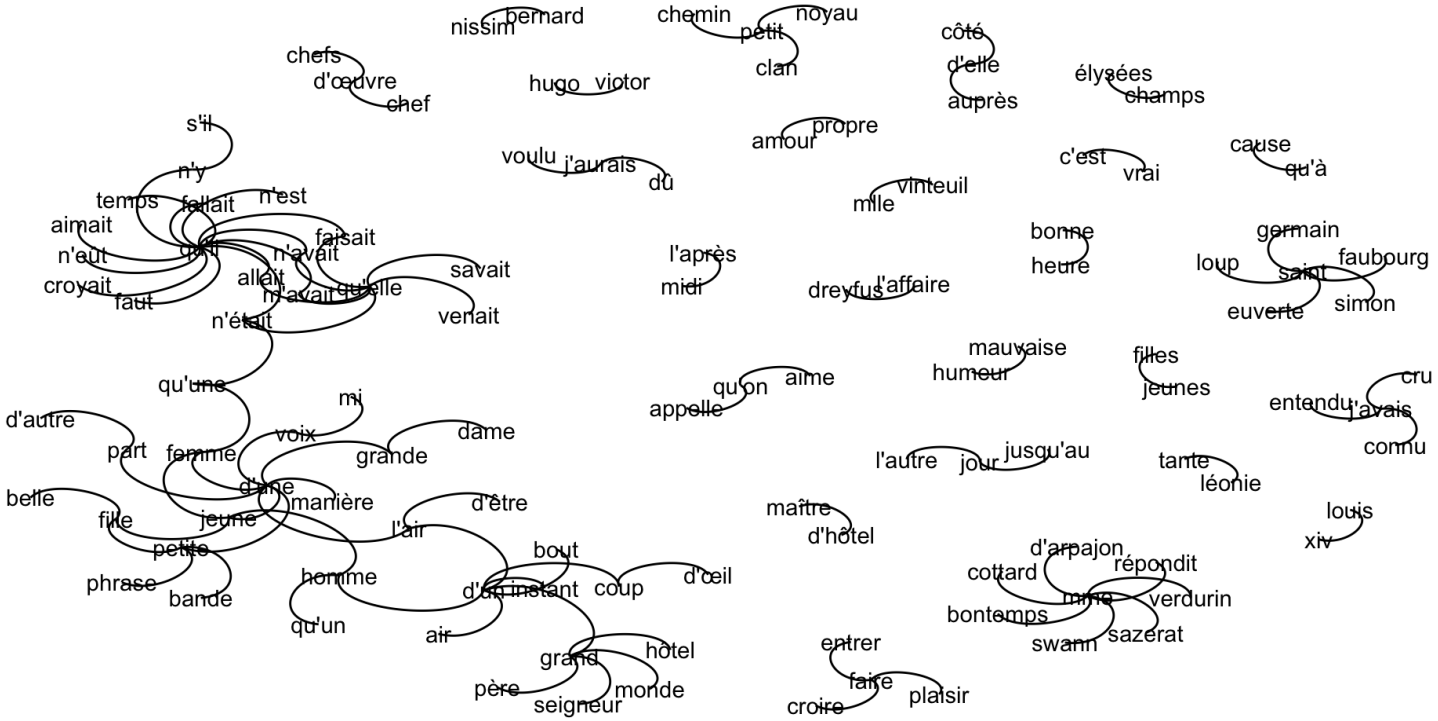
Step 2: plot

```
# Set seed for reproducibility
set.seed(2811)
ggraph(pr_graph) +
  geom_edge_link(arrow = grid::arrow(angle = 10, unit(0.1, "inches"))) +
  geom_node_text(aes(label = name)) +
  #theme_graph is designed for plotting graphs
  theme_graph()
```

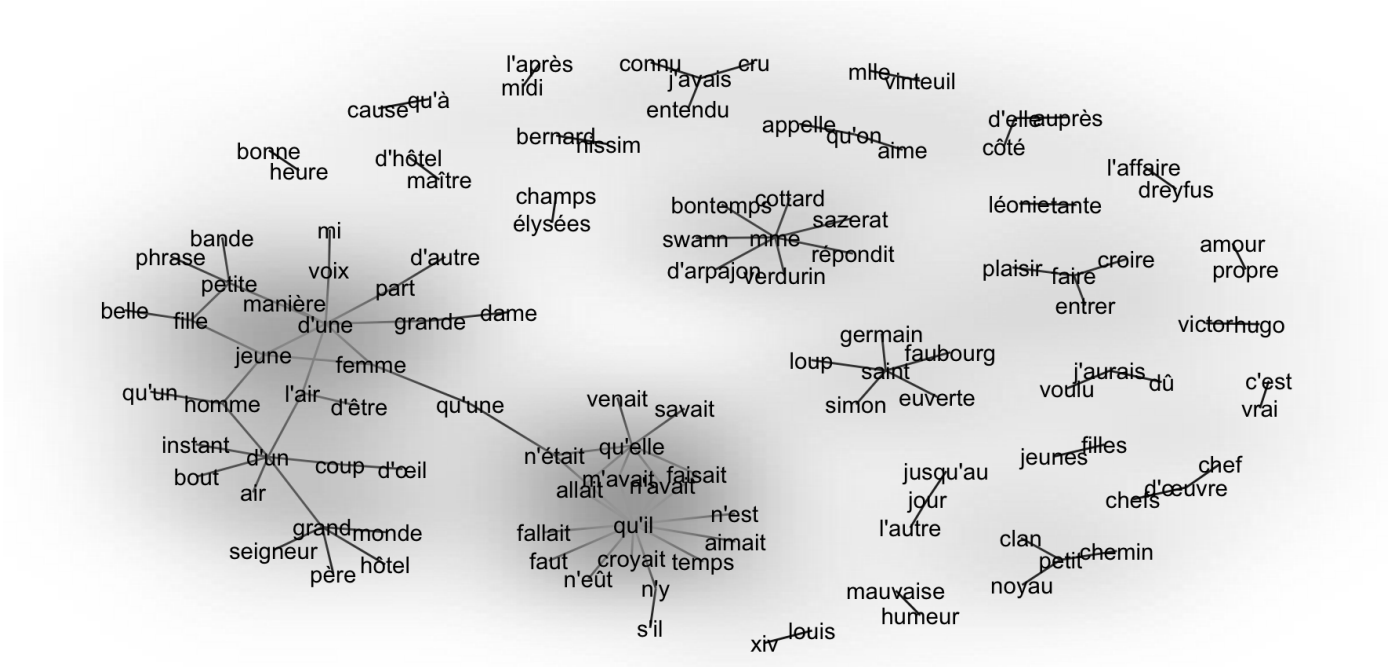
Step 2: plot


d'hôtel
maître
auprès
d'elle
côté
dreyfus
l'affaire
mauvaise
humeur
léonie
tante
aime
qu'on
appelle
croire
faire
plaisir
entrer
dû
j'aurais
voulu
jeunes
filles
midi
l'après
savait
venait
n'avait
faut
aimait
s'il
qu'elle
m'avait
n'y
faisait
n'est
allait
fallait
n'était
n'eût
croyait
temps
monde
père
seigneur
hôtel
grand
bernard
nissim
connu
j'avais
entendu
cru
chefs
d'œuvre
chef
amour
propre
mlle
vinteuil
bout
air
instant
d'un
coup
d'œil
qu'un
homme
qu'une
xiv
louis
d'être
l'air
femme
jeune
champs
élysées
vrai
c'est
d'arnajon
sazerat
bontemps
répondit
mme
verdurin
swann
cottard
fille
belle
d'autre
part
d'une
manière
petite
phrase
grande
voix
bande
dame
mi
heure
bonne
hugo
victor
cause
qu'à
euverte
germain
simon
saint
faubourg
loup
chemin
petit
clan
noyau
jusqu'au
jour
l'autre

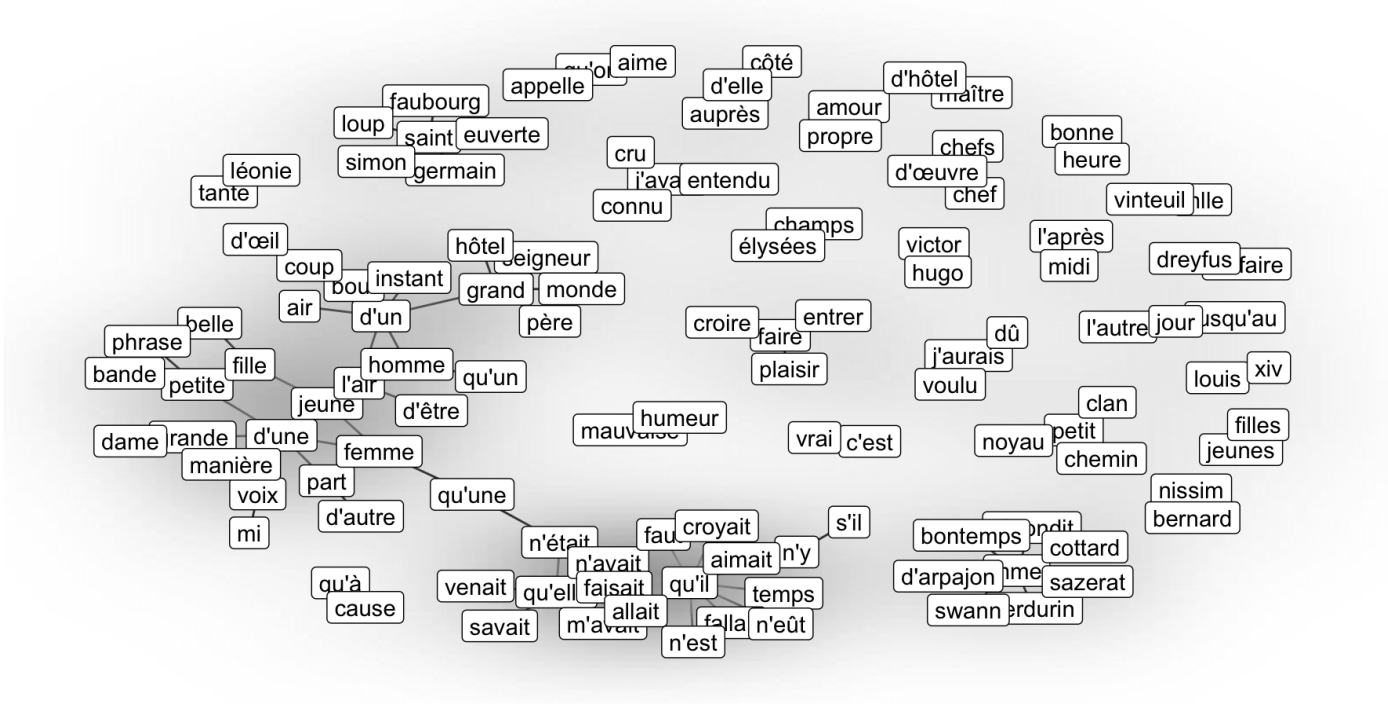
Step 3: change (to geom_edge_arc)



Step 3: change (add geom_edge_density)



Step 3: change (to geom_node_label)



Step 4: Customize (Your own personnal layout)

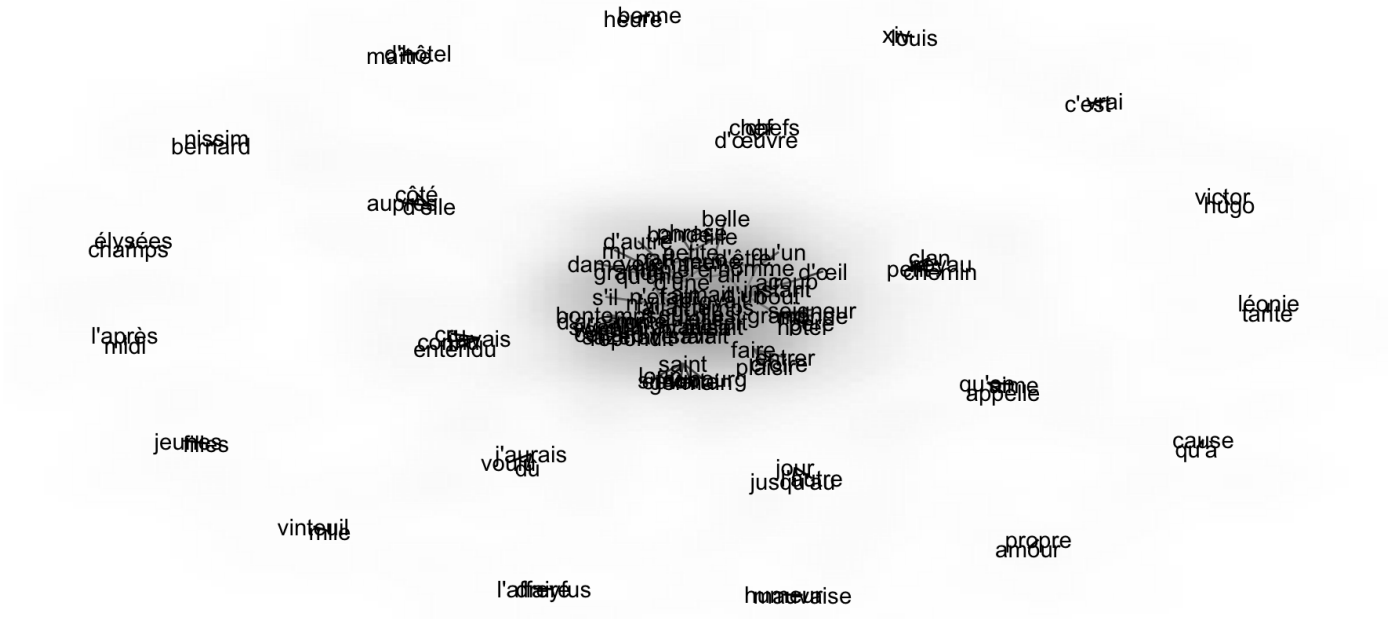
A layout is a global specification of a graph spatial position. As a global parameter, it can to be set out of your plot call.

If you don't specify any layout, {ggraph} chooses one for you. You can change it with `layout = ""` in your `ggraph()` function, or by creating a layout object with `create_layout()`.

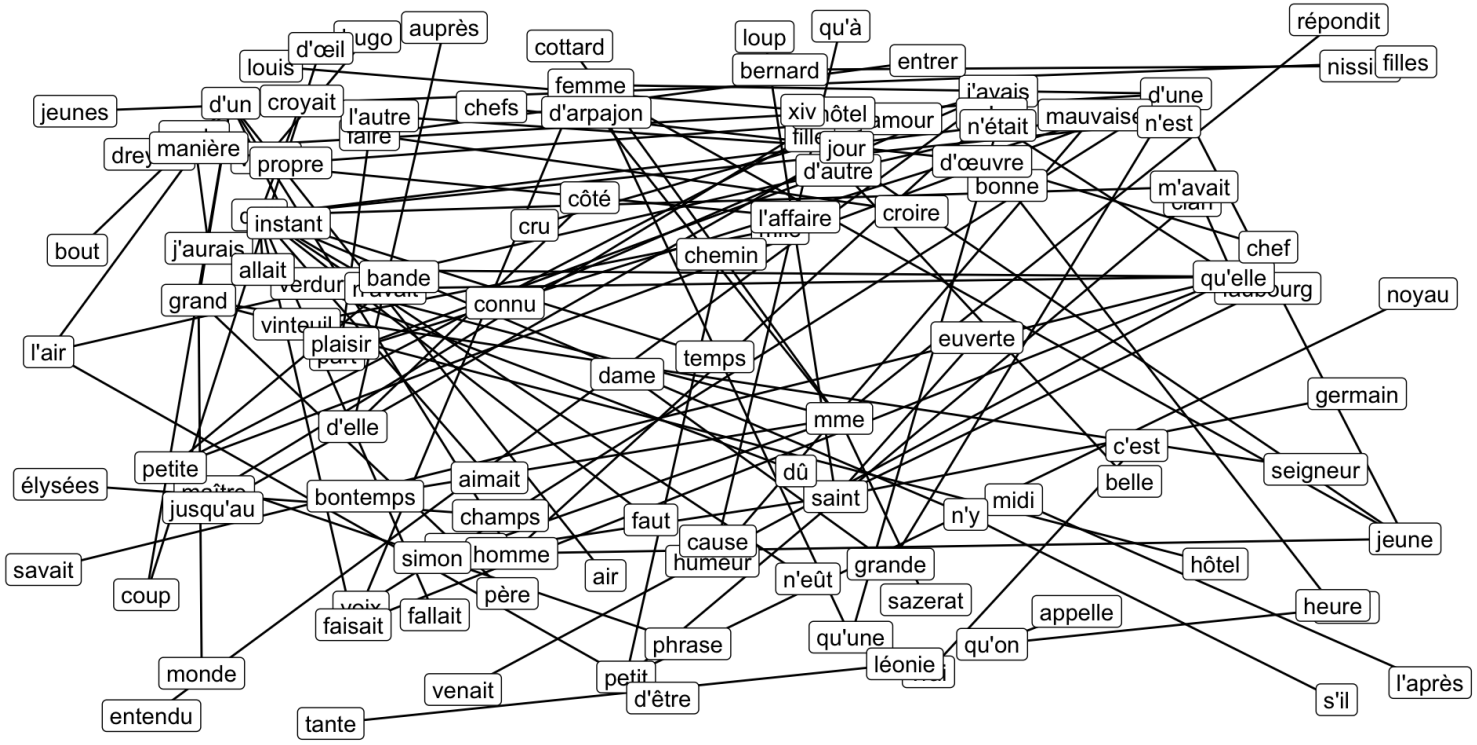
There are 13 different layouts implemented in {ggraph}, but no answer to the question "which is the perfect one?", the answer depends mainly on what network you want to plot.

The default algorithms in the basic layout are 'star', 'circle', 'gem', 'dh', 'graphopt', 'grid', 'mds', 'randomly', 'fr', 'kk', 'drl', 'lgl'.

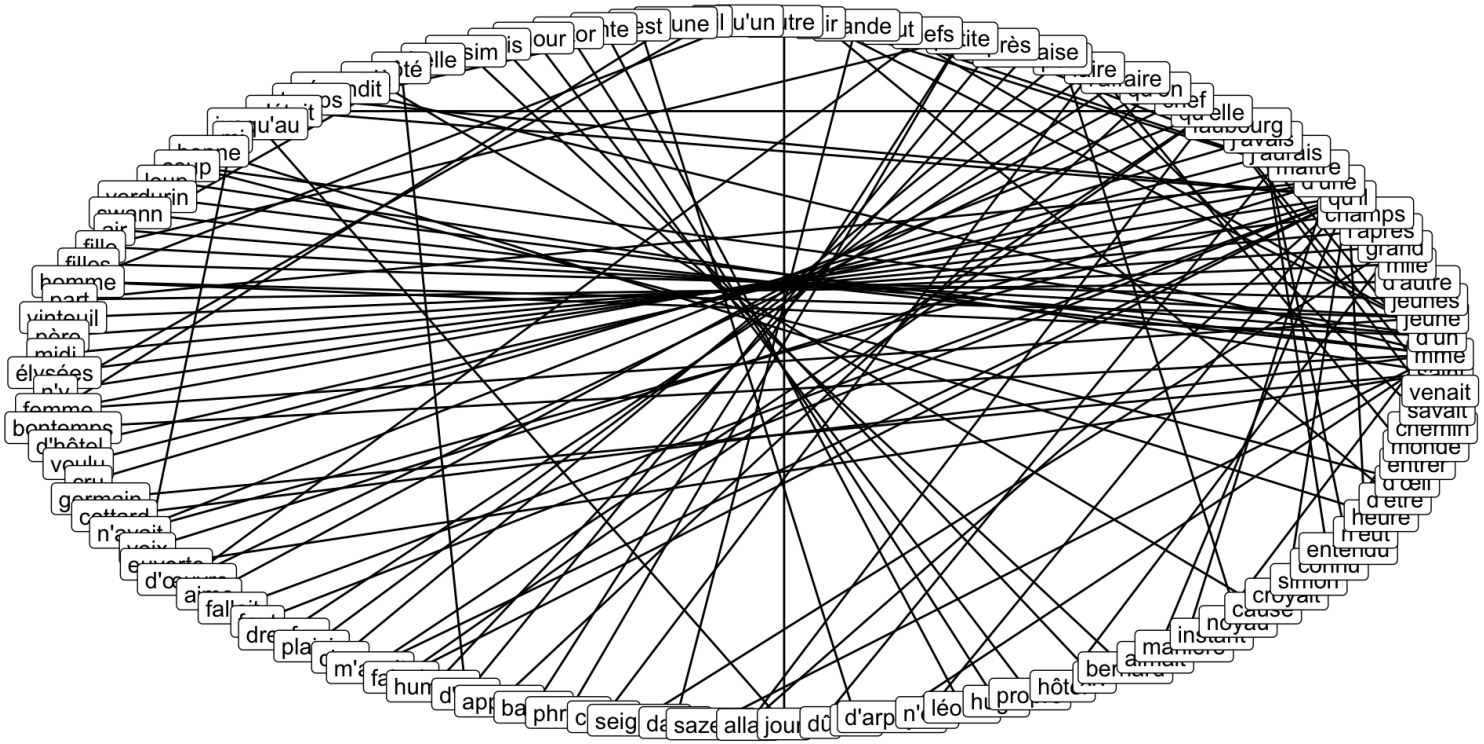
Step 4: Customize (layout = "gem")



Step 4: Customize (layout = "randomly")



Step 4: Customize (layout = "circle")



Step 5: Prettify

{ggraph} is built on top of {ggplot2}, so you can use classic {ggplot2} functions to customize your plot.

There are also functions which are specific to {ggraph}, starting with `scale_`.

Such as

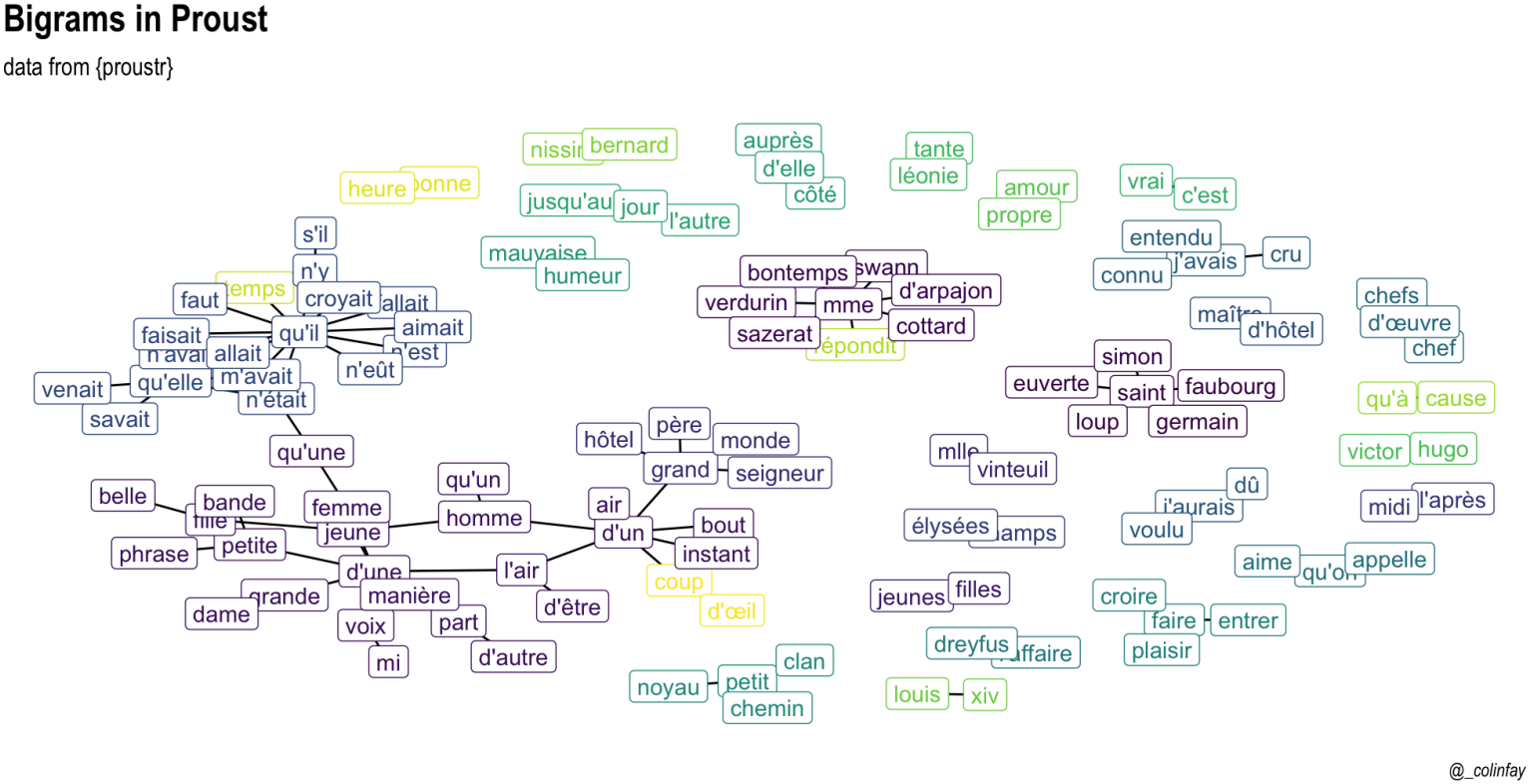
- `scale_edge_alpha_manual`
- `scale_edge_color_viridis`
- `scale_edge_fill_grey`
- `scale_edge_shape_discrete`

If you're familiar with {ggplot2}, these won't surprise you.

Step 5: Prettify

```
ggraph(pr_graph, layout = "fr") +
  geom_edge_link()+
  geom_node_label(aes(label = name,
                      color = as.factor(pr_clust$membership)),
                  show.legend = FALSE) +
  scale_edge_color_continuous(low = "#440154FF", high = "#FDE725FF") +
  scale_color_viridis_d() +
  labs(title = "Bigrams in Proust",
       subtitle = "data from {proustr}",
       caption = "@_colinfay") +
  theme_graph()
```

Step 5: Prettify



Using other networks formats

You can use other type of graph objects with {ggraph} (but we won't see them today).

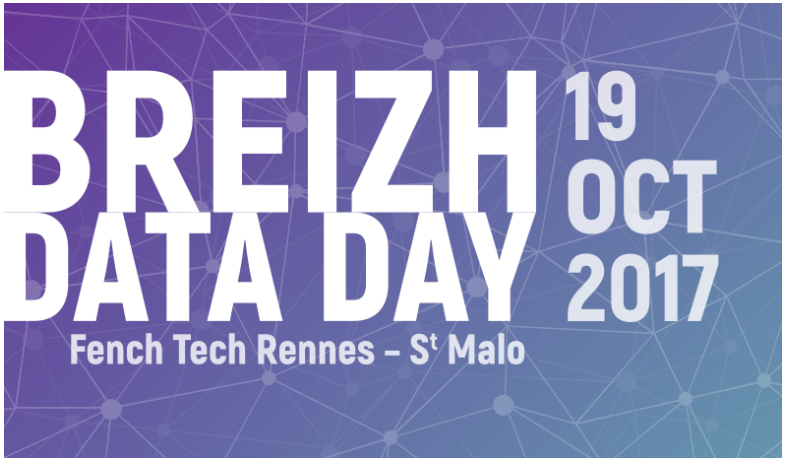
- `layout = "igraph"` or `auto`: the default, algorithm being 'nicely'
- `layout = "dendrogram"`: the nodes are set in a dendrogram maner, with children at position 0 and parent at position 1 above.
- `layout = "linear"`: nodes are arranged linearily.
- `layout = "treemap"`: nodes as tree.
- `layout = "hclust"`: nodes are as hclust.

A Twitter case study

Let's dive into a case study with a (really) small Twitter dataset.

The dataset contains 230 tweets, which have been collected with TAGS v0.6, a google script to automatically extract a search from Twitter on a hourly basis.

The dataset contains all tweets with #BreizhDataDay, a french event which took place in Rennes on the 19th of October.



The dataset

```
glimpse(bdd)
```

```
#> Observations: 230
#> Variables: 18
#> $ id_str                   <dbl> 9.235430e+17, 9.235108e+17...
#> $ from_user                <chr> "thinkR_fr", "Tristan_Le_D...
#> $ text                     <chr> "RT @_ColinFay: Et les twe...
#> $ created_at               <chr> "Thu Oct 26 13:33:01 +0000...
#> $ time                     <chr> "26/10/2017 14:33:01", "26...
#> $ geo_coordinates          <lgl> NA, NA, NA, NA, NA, NA, NA...
#> $ user_lang                <chr> "fr", "fr", "fr", "en", "e...
#> $ in_reply_to_user_id_str  <int> NA, NA, NA, NA, NA, NA, NA...
#> $ in_reply_to_screen_name  <chr> "", "", "", "", "", "", ""...
#> $ from_user_id_str         <dbl> 3.044687e+09, 2.246314e+09...
#> $ in_reply_to_status_id_str <dbl> NA, NA, NA, NA, NA, NA, NA...
#> $ source                   <chr> "<a href=\"http://twitter....
#> $ profile_image_url        <chr> "http://pbs.twimg.com/prof...
#> $ user_followers_count     <int> 1241, 3221, 1210, 3555, 91...
#> $ user_friends_count       <int> 950, 2300, 604, 815, 82, 8...
#> $ user_location            <chr> "", "Rennes, Dinan,  Breta...
#> $ status_url               <chr> "http://twitter.com/thinkR...
#> $ entities_str             <chr> "{\"hashtags\":[{\"text\":...
```

Text mining

Let's repeat our previous bigram analysis :

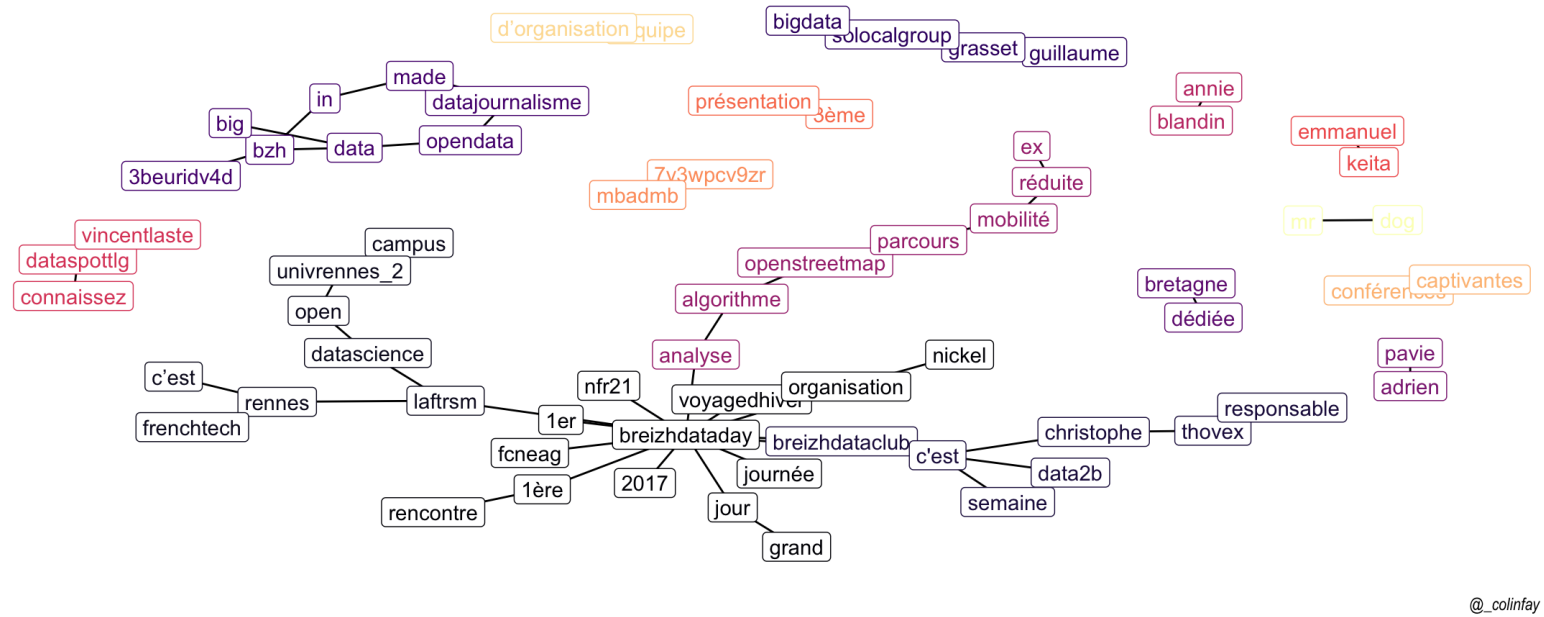
```
# Create custom stopwords
sw <- data.frame(word  = c(as.character(proust_stopwords()$word),
                           "https","rt", "d'une", "t.co"))
# Bigrams
tm_bdd <- bdd %>%
  unnest_tokens(bigram, text, token = "ngrams", n = 2) %>%
  separate(bigram, c("from", "to"), sep = " ") %>%
  filter(!from %in% sw$word) %>%
  filter(!to %in% sw$word) %>%
  count(from, to, sort = TRUE) %>%
  rename(size = n) %>%
  filter(size >= 5)
```

Bigrams in #BreizhDataDay

```
# Create the graph and the clusters
bdd_graph <- graph_from_data_frame(tm_bdd, directed = F)
bdd_clust <- cluster_edge_betweenness(bdd_graph)

#Draw
ggraph(bdd_graph) +
  geom_edge_link()+
  geom_node_label(aes(label = name,
                      color = as.factor(bdd_clust$membership)),
                  show.legend = FALSE) +
  scale_color_viridis_d(option = "A") +
  labs(title = "Bigrams in #BreizhDataDay",
       subtitle = "data from Twitter",
       caption = "@_colinfay") +
  theme_graph()
```

Bigrams in #BreizhDataDay



Trying with {visNetwork}

{visNetwork} is a network visualisation package built on top of the vis.js JavaScript library. The main function we'll see today is `visNetwork`.

This function takes two main arguments:

- a nodes dataframe with a list of node information (the minimum info being `id`), and you can also pass `label`, `group`, or `value`.
- an edged data.frame, with `from`, `to`, `label`, `value`.

You can also pass a `main` arg for title, width and height for plot size, submain for subtitle, or footer.

Trying with {visNetwork}

We've already got an edge object, we'll then need a node object.

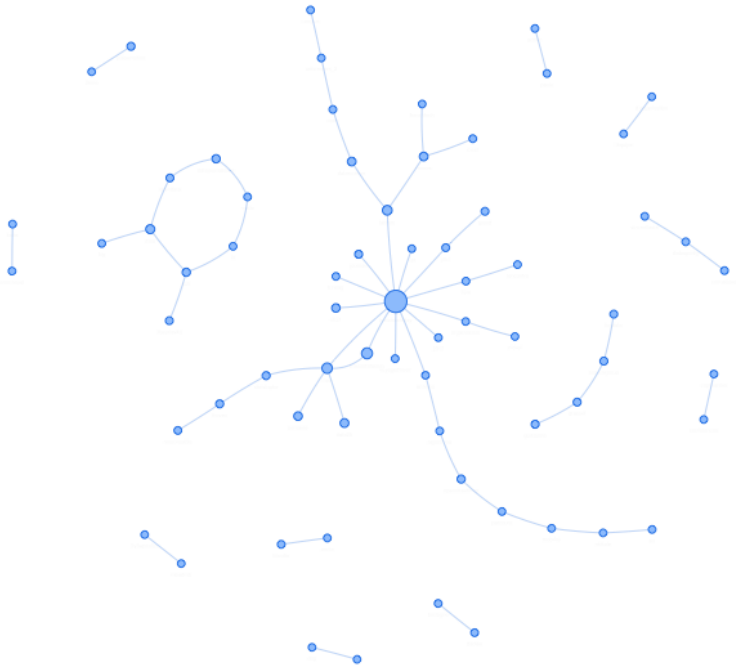
The `value` arg will be the number of time the first unigram appears in the dataset.

```
uni_bdd <- bdd %>%
  unnest_tokens(output = word, input = text) %>%
  anti_join(sw) %>%
  count(word) %>%
  rename(value = n)
nodes_tm_bdd <- tibble(id = unique(c(tm_bdd$from, tm_bdd$to)),
                       label = unique(c(tm_bdd$from, tm_bdd$to))) %>%
  left_join(uni_bdd, by = c(id = "word"))
```

Then we'll call:

```
visNetwork(nodes_tm_bdd, tm_bdd)
```

Trying with {visNetwork}



A Twitter case Study

Let's try something else, and map the conversation in this dataset.

We'll look at who are the most retweeted and mentionned users.

The first step is to create a node object, containing all the users. We'll map the value (the node size) to the number of followers of a given user.

Then, we'll need to parse the `entities_str` column from our dataset, which is a JSON format containing hastags and mentions.

Creating a node object of the users

```
nodes_users <- bdd %>%
  mutate(label = from_user,
         id = from_user) %>%
  group_by(label, id) %>%
  summarise(value = round(mean(user_followers_count,na.rm = TRUE)),
            nbr_tweet = n()) %>%
  na.omit()
glimpse(nodes_users)
```

```
#> Observations: 76
#> Variables: 4
#> $ label     <chr> "_ColinFay", "2m1journaliste", "6Malix", "...
#> $ id        <chr> "_ColinFay", "2m1journaliste", "6Malix", "...
#> $ value     <dbl> 3555, 3606, 3, 2220, 69, 2292, 575, 2520, ...
#> $ nbr_tweet <int> 6, 1, 2, 1, 1, 1, 2, 1, 1, 18, 3, 29, 1, 1...
```

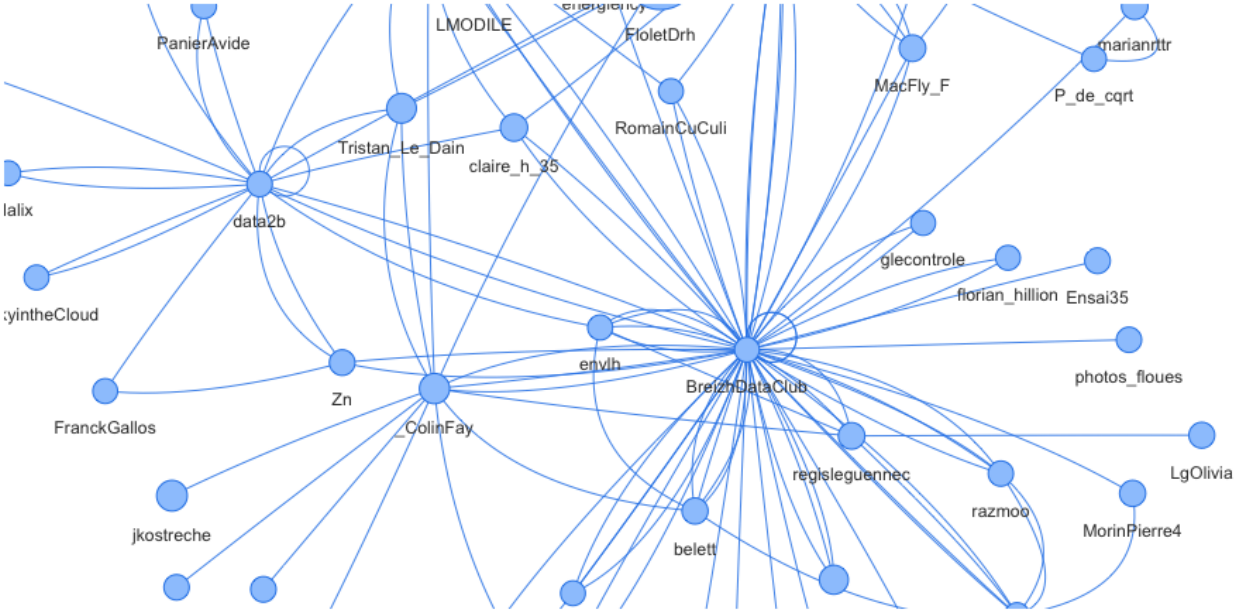
Graph of mentions and RT

Who mention / RT who?

```
parse_mention <- function(user, vec){
  a <- jsonlite::fromJSON(vec, simplifyDataFrame = TRUE)
  res <- a$user_mentions
  res$user <- user
  select(res, from = user, to = screen_name)
}
safe_parse <- safely(parse_mention)
edges_users <- map2(bdd$from_user, bdd$entities_str,
                    ~ safe_parse(user = .x, vec = .y) ) %>%
  map("result") %>%
  compact() %>%
  reduce(bind_rows)
```

Visualize that

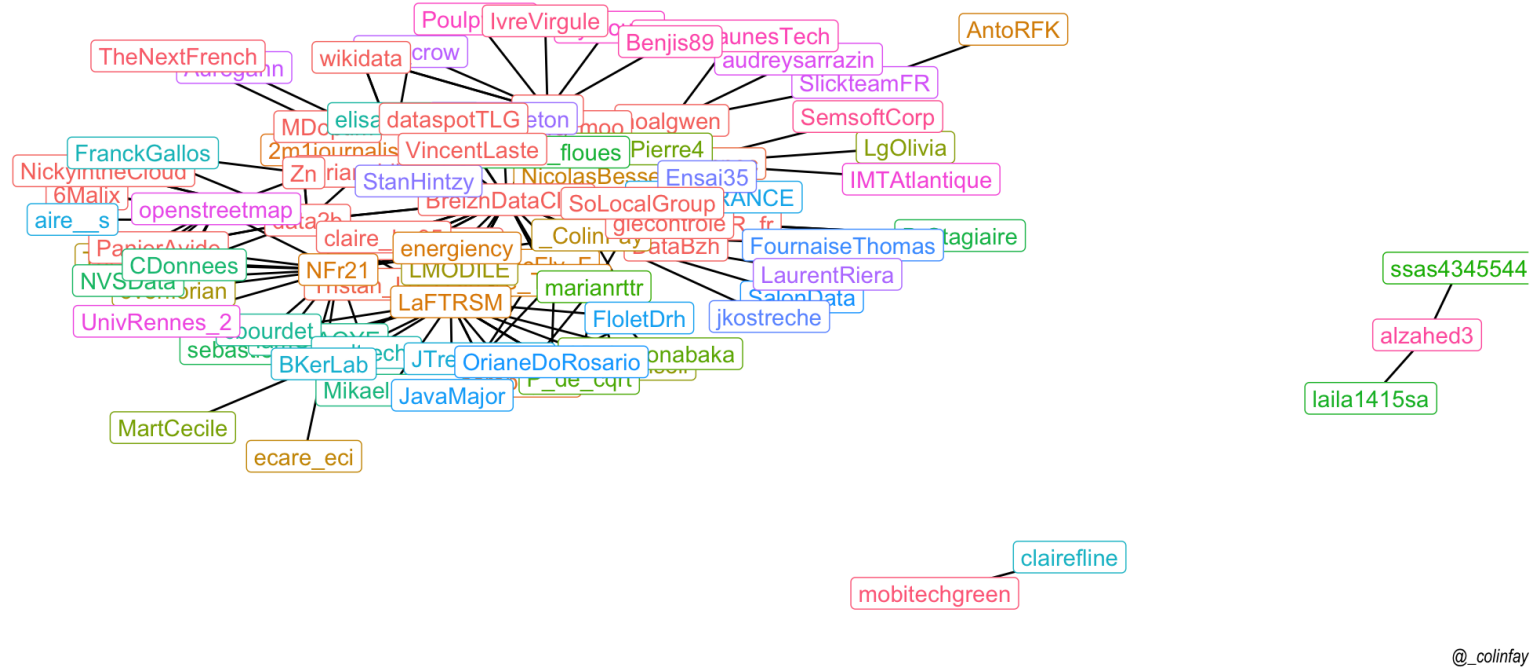
```
visNetwork(nodes_users, edges_users)
```



Do the same with {ggraph}

```
bdd_graph_user <- graph_from_data_frame(edges_users)
bdd_clust_user <- cluster_edge_betweenness(bdd_graph_user)
ggraph(bdd_graph_user, layout = "kk") +
  geom_edge_link()+
  geom_node_label(aes(label = name,
                      color = as.factor(bdd_clust_user$membership)),
                  show.legend = FALSE) +
  labs(title = "Users in #BreizhDataDay",
       subtitle = "data from Twitter",
       caption = "@_colinfay") +
  theme_graph()
```

Do the same with {ggraph}



Time for another function!

Build a function to filter on specific term

Such a big network might not be what you're looking for: it's hard to read information, and even with the JS solution, you'll need to zoom in.

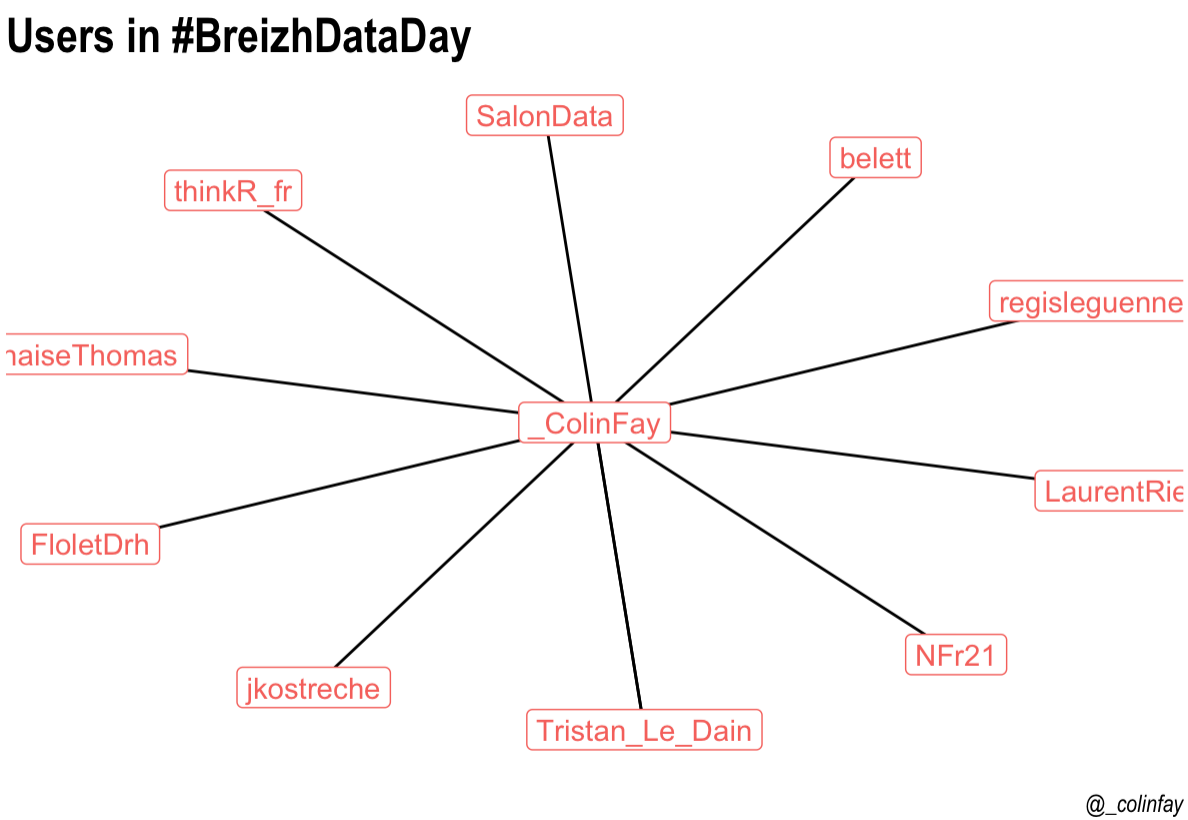
What you might be looking for is a function that can take a list of user names, and draw a graph.

draw_graph()

```
draw_graph <- function(edge_tbl, ...){
  users_filter <- list(...)
  if (length(users_filter) != 0){
   edge_tbl <- filter(edge_tbl, to %in% list(...))
  }
  edges <- graph_from_data_frame(edge_tbl)
  clusters <- cluster_edge_betweenness(edges)
  ggraph(edges, layout = "kk") +
    geom_edge_link(arrow = grid::arrow(angle = 10, unit(0.1, "inches"))) +
    geom_node_label(aes(label = name,
                        color = as.factor(clusters$membership)),
                    show.legend = FALSE) +
    labs(title = "Users in #BreizhDataDay",
         caption = "@_colinfay") +
    theme_graph()
}
```

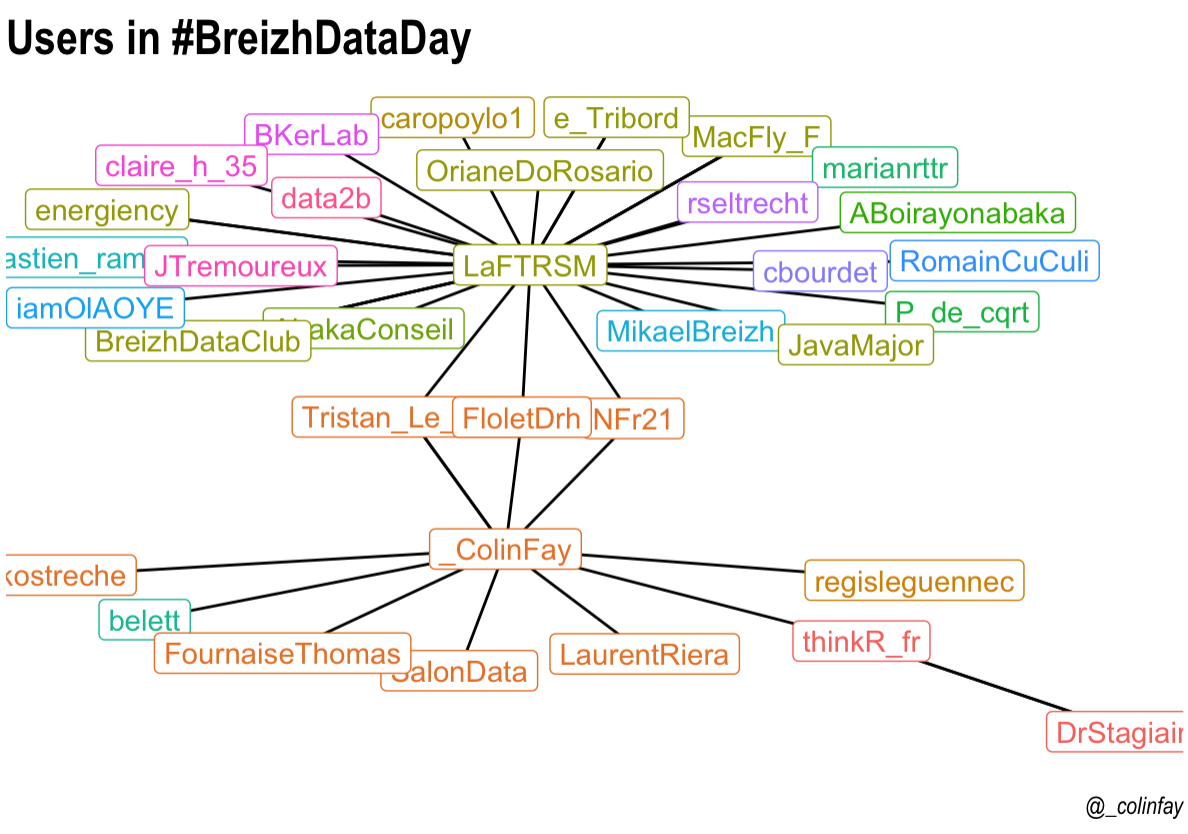
What about me?

```
draw_graph(edges_users, "_ColinFay")
```



Trying 3 users

```
draw_graph(edges_users, "_ColinFay", "thinkR_fr", "LaFTRSM")
```



Trying 6 users

```
draw_graph(edges_users, "_ColinFay", "thinkR_fr", "LaFTRSM",
"BreizhDataClub ", "claire_h_35", "dataspotTLG")
```

Shiny App

The "ultimate" solution to look for specific relationships would be an app to select names, and look for mentions in the dataset.

Good news: there's a package for that, and it's called {shiny}.

Sidenote: there's a shiny output for visNetwork, but I've chosen to focus on {ggraph} here.

Shiny App

```
library(shiny)
ui <- fluidPage(
  titlePanel("users in #BreizhDataDay"),
  sidebarLayout(
    sidebarPanel(
      selectInput(inputId = "user", multiple = TRUE,
                  label = "Users to filter",choices = edges_users$to,
                  selected = "thinkR_fr")
    ),
    mainPanel(
      h3("Network"), plotOutput("network"),
      h3("Exchanges on the graph"), DT::dataTableOutput("table")
    )
  )
)
```

Shiny App

```
server <- function(input, output) {
   output$network <- renderPlot({
     draw_graph(filter(edges_users, to == input$user))
   })
   output$table <- DT::renderDataTable({
     DT::datatable(filter(edges_users, to == input$user))
   })
}
shinyApp(ui = ui, server = server)
```

Shiny App

users in #BreizhDataDay

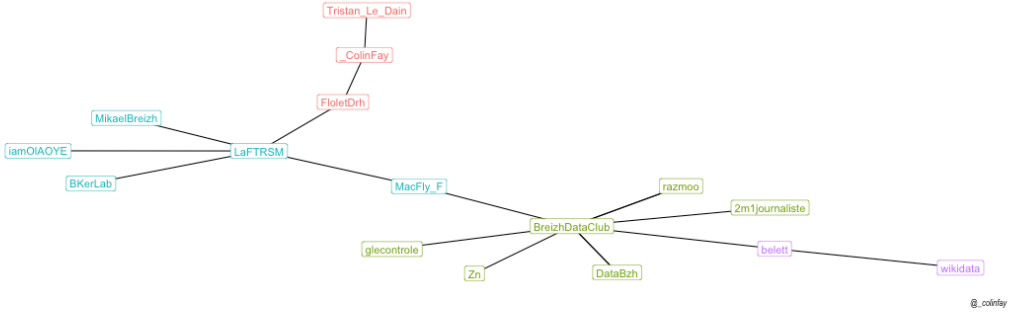
Users to filter

thinkR_fr _ColinFay wikidata BreizhDataClub LaFTRSM data

data2b
dataspotTLG

Network

Users in #BreizhDataDay



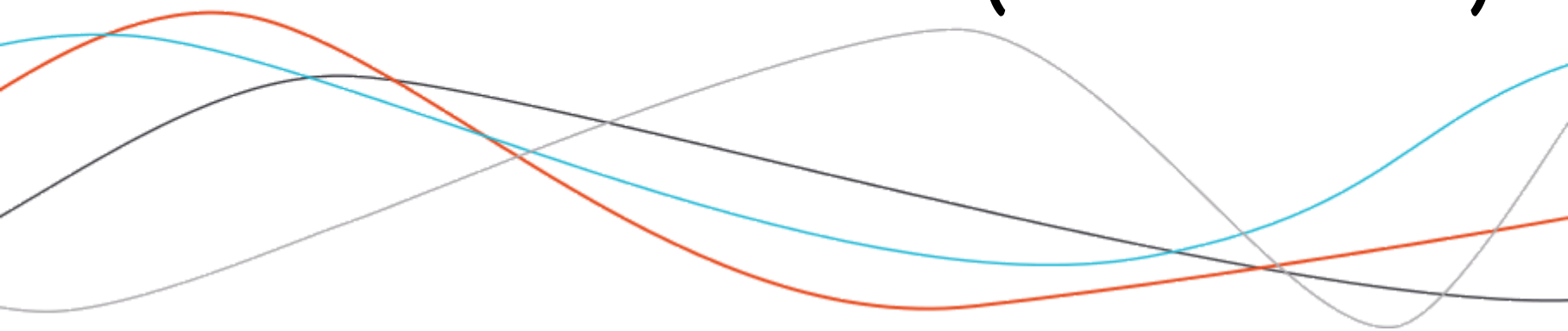
Exchanges on the graph

Show 10 entries Search:

	from	to
1	Tristan_Le_Dain	_ColinFay
2	2m1journaliste	BreizhDataClub
3	MacFly_F	BreizhDataClub
4	MacFly_F	LaFTRSM
5	BreizhDataClub	BreizhDataClub
6	razmoo	BreizhDataClub
7	belett	BreizhDataClub
8	belett	wikidata

Quick demo

(If I have time...)



Thanks !

Any questions ?

Find me in the web:

- colin@thinkr.fr
- http://twitter.com/_colinfay
- http://twitter.com/thinkr_fr
- https://github.com/ColinFay

And also:

- https://thinkr.fr/
- http://colinfay.me/