Exploring mobility dynamics in FVG - part 2: network

Table of contents

0.1 Objective

Mobility patterns are **encoded in tabular form**, where each row represents a flow from location A to location B, characterized by categorical variables (inbound or outbound, resident or traveler) and a quantitative indicator of the number of journeys.

The **objective** of this notebook is to **construct a network** where nodes represent locations A, B, C, D, etc., and edges encode information on mobility flows, within a user-specified time frame. The core task is to find a meaningful definition for **edge weights**, considering the magnitude of flows between locations.

Finally, we aim to develop a **statistical model** for edge weight estimation, allowing for both *point estimates* and *interval estimates*, providing insights into the dispersion of data and its evolution over time.

The main dataset has been prepared in script 01. In the first part of this script however we use a smaller test dataset (available in subfolder `dummy_data`) to clarify the methodology.

coding specifications

The R code utilizes the `readxl` package to import dummy data from Excel and the `tidyverse` package for efficient data manipulation. Specifically, `tidyverse`an efficient syntax and verbs like `mutate`,`rename`, and `select` to clean and preprocess the data, enhanching readability and facilitating code debug and reuse. Netrowks are created and analysed with`igraph` package.

1 Methodology

To explain the methodology, we can use a simple example of flows between two locations A-B, in single day.

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.2     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.3     v tibble    3.2.1
v lubridate 1.9.2     v tidyr     1.3.0
v purrr     1.0.1
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to beco
```

```
library(readxl)
library(igraph)
```

```
Caricamento pacchetto: 'igraph'

I seguenti oggetti sono mascherati da 'package:lubridate':

    %--%, union

I seguenti oggetti sono mascherati da 'package:dplyr':

    as_data_frame, groups, union
```

```
I seguenti oggetti sono mascherati da 'package:purrr':

    compose, simplify

Il seguente oggetto è mascherato da 'package:tidyr':

    crossing

Il seguente oggetto è mascherato da 'package:tibble':

    as_data_frame

I seguenti oggetti sono mascherati da 'package:stats':

    decompose, spectrum

Il seguente oggetto è mascherato da 'package:base':

    union
```

Loading a toy dataset

```
#read data a
edges_simple <- read_excel('./dummy_data/simple_AB_1day.xlsx')
head(edges_simple)
```

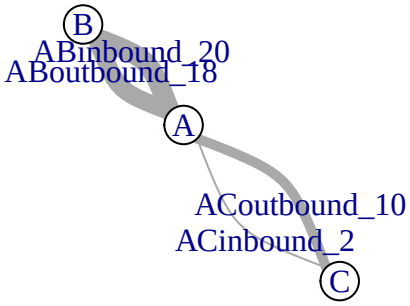
```
# A tibble: 4 x 5
  origin destination day                 direction     n
  <chr>  <chr>       <dttm>              <chr>     <dbl>
1 A      B           2020-06-01 00:00:00 inbound      20
2 A      B           2020-06-01 00:00:00 outbound     18
3 A      C           2020-06-01 00:00:00 inbound       2
4 A      C           2020-06-01 00:00:00 outbound     10
```

This data can be used straightforward to build a network, with an edge for each row, and weight that matches the direction. Here we show how, and explain why it is not the an optimal way of encoding the information.

We assume that the edge has a weight that is $w = +n$ if direction is *out*, and $w = -n$ if direction is *in*.

```
add_edge_labels <- function(g){
    for (i in 1:ecount(g)) {
        edge <- E(g)[i]
        first_node_name <- V(g)[.from(edge)]$name
        second_node_name <- V(g)[.to(edge)]$name
        direction = E(g)[i]$direction
        weight = E(g)[i]$weight
        edge_label <- paste0(first_node_name,
                             second_node_name,
                             direction,
                             "_",
                             weight)
        E(g)$label[i] <- edge_label
    }
    return(g)
}
```

```
g <- graph_from_data_frame(edges_simple, directed = FALSE)
E(g)$weight <- E(g)$n
g <-add_edge_labels(g)
plot(g,
     edge.label = E(g)$label,
     edge.width = E(g)$n/2,
     vertex.color = 'white',
     vertex.size = 30)
```



A more meaningful choice for weight is mean_flow = mean(in, out), which produces a single edge per day. Moreover, we can measure the symmetry of the flows with `net_flow = out-in` and `sk= net / mean_flow`

```
edges_flow = read_excel('./dummy_data/simple_AB_1day.xlsx') %>%
  group_by(origin, destination, direction) %>%
  summarise(n_mean = mean(n)) %>%
  ungroup() %>%
  pivot_wider(names_from = direction, values_from = n_mean, values_fill = 0) %>%
  mutate(flow_mean = (inbound + outbound)/2)  %>%
  mutate(flow_net = inbound - outbound) %>%
  mutate(flow_ratio = flow_net / flow_mean)
```

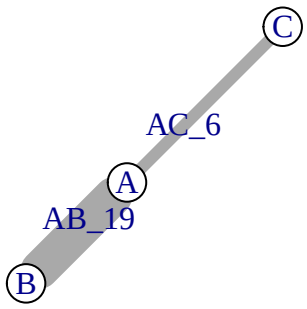
```
`summarise()` has grouped output by 'origin', 'destination'. You can override
using the `.groups` argument.
```

```
edges_flow
```

```
# A tibble: 2 x 7
  origin destination inbound outbound flow_mean flow_net flow_ratio
```

```
  <chr>  <chr>          <dbl>    <dbl>     <dbl>    <dbl>       <dbl>
1 A      B                 20       18        19        2       0.105
2 A      C                  2       10         6       -8      -1.33
```

```
g <- graph_from_data_frame(edges_flow, directed = FALSE)
E(g)$weight = E(g)$flow_mean
E(g)$direction = ""
g <-add_edge_labels(g)
plot(g,
     edge.label = E(g)$label,
     edge.width = E(g)$weight,
     vertex.color = 'white',
     vertex.size = 30)
```



2 Using the real data

Load data

```
flows <- read_csv('./data/flows.csv')
```

```
Rows: 117510 Columns: 7
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (4): direction, res_trav, origin, destination
dbl  (2): n, weekday
date (1): day

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
ndays = length( unique(flows$day))
print(ndays)
```

```
[1] 5
```

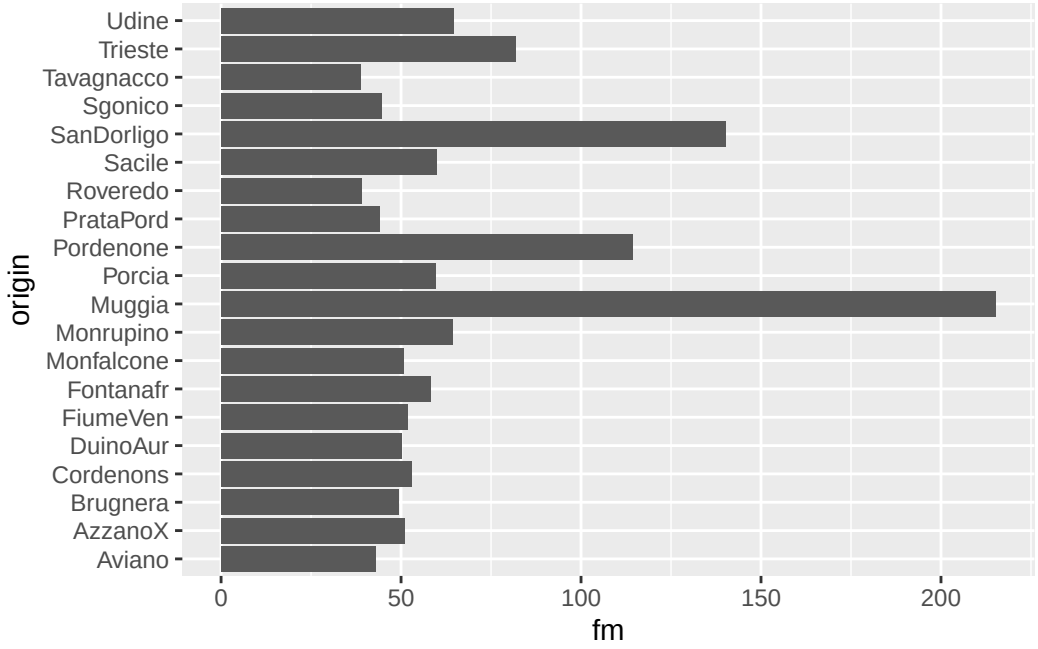
Caluclate network edges from flows

```
ndays = length( unique(flows$day))
# Group by departure and destination, then calculate the sum of n_viaggi
 edges <- flows %>%
  select(day, origin, destination, direction, n) %>%
  group_by(origin, destination, direction, day) %>%
  summarise(n_mean = sum(n) / ndays) %>%
  ungroup() %>%
  pivot_wider(names_from = direction, values_from = n_mean, values_fill = 0) %>%
  mutate(
    flow_mean = (inbound + outbound) / 2,
    flow_net = inbound - outbound,
    flow_ratio = flow_net / max(inbound, outbound)
  ) %>%
  group_by(origin, destination) %>%
  filter(flow_mean >= 5) %>%
  ungroup()
```

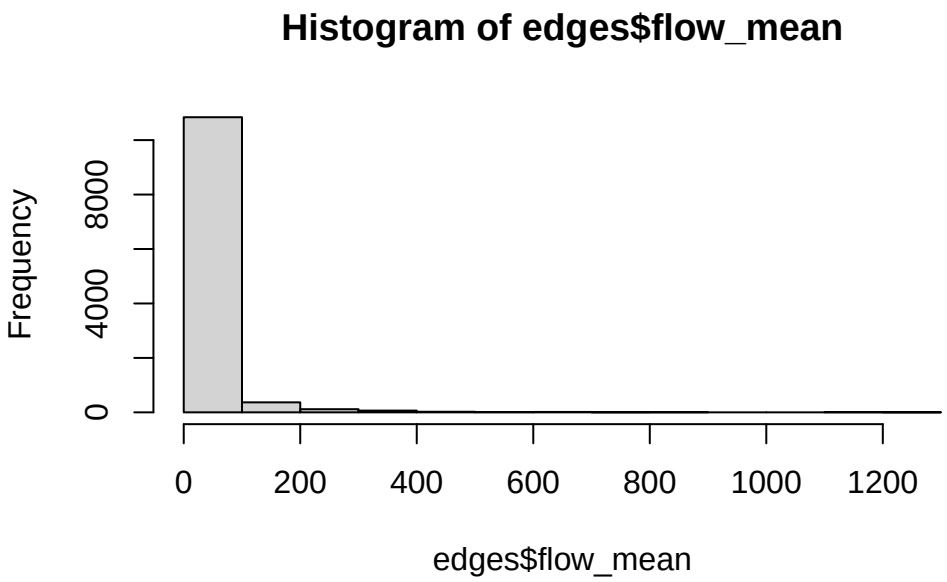
```
`summarise()` has grouped output by 'origin', 'destination', 'direction'. You
can override using the `.groups` argument.
```

```
edges%>%
    group_by(origin)%>%
```

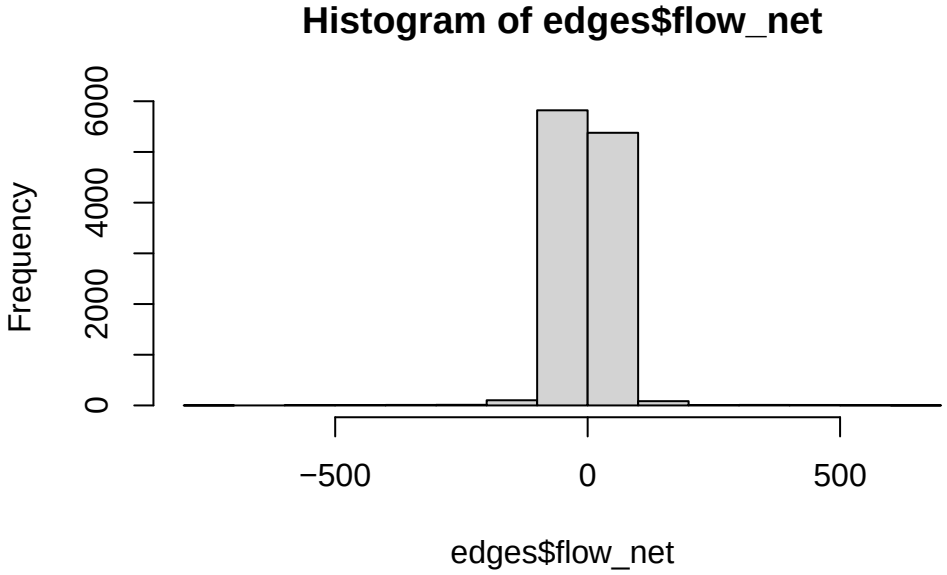
```
  summarize(fm = mean(flow_mean,  na.rm = TRUE)   ) %>%
  arrange(-fm) %>% head(20)%>%
  ggplot(aes(y = origin, x = fm )) + geom_col()
```



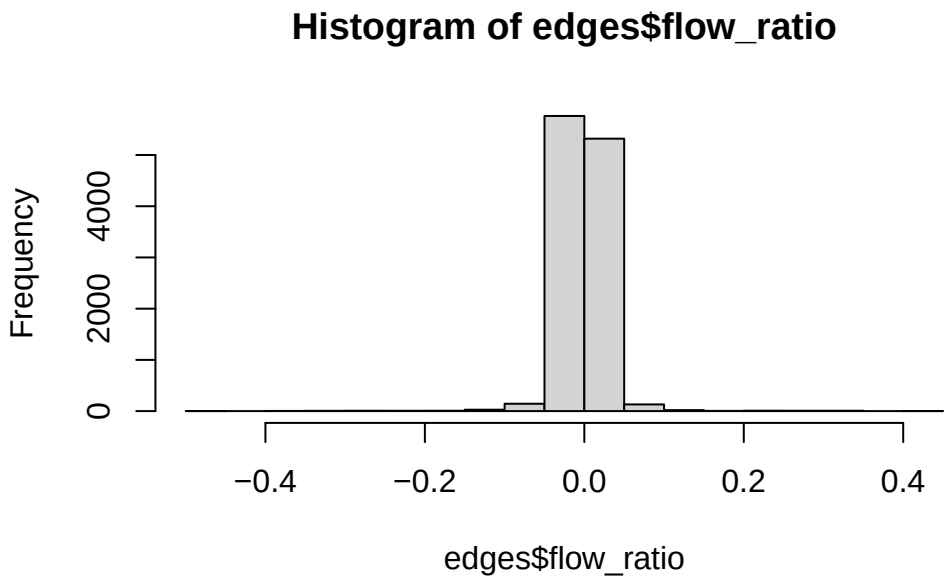

```
hist(edges$flow_mean )
```




```r
hist(edges$flow_net )
```



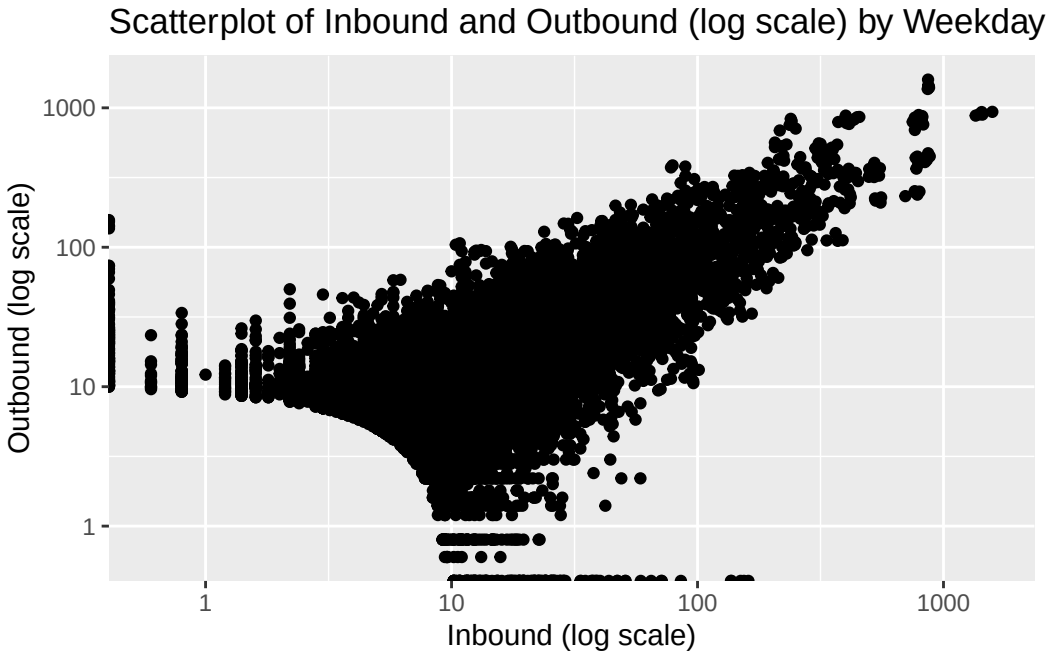
```r
hist(edges$flow_ratio )
```



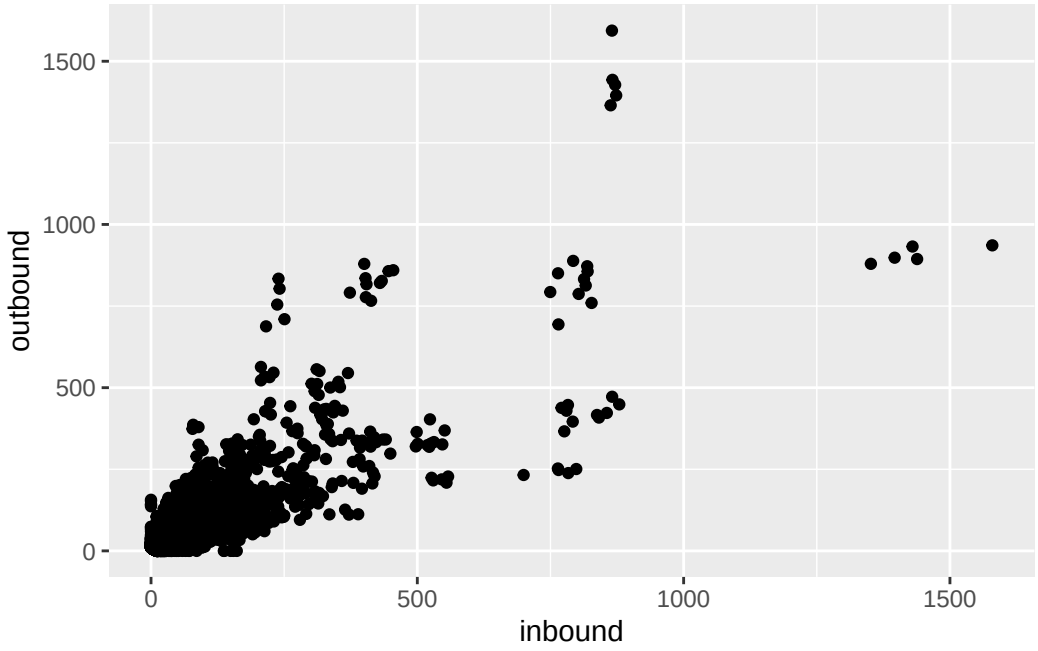

```r
# Create scatterplot
edges %>% ggplot(aes(x = inbound, y = outbound)) +
  geom_point() +
  scale_x_log10() +
  scale_y_log10() +
  labs(x = "Inbound (log scale)", y = "Outbound (log scale)", color = "Weekday") +
  ggtitle("Scatterplot of Inbound and Outbound (log scale) by Weekday")
```

```
Warning: Transformation introduced infinite values in continuous x-axis
```

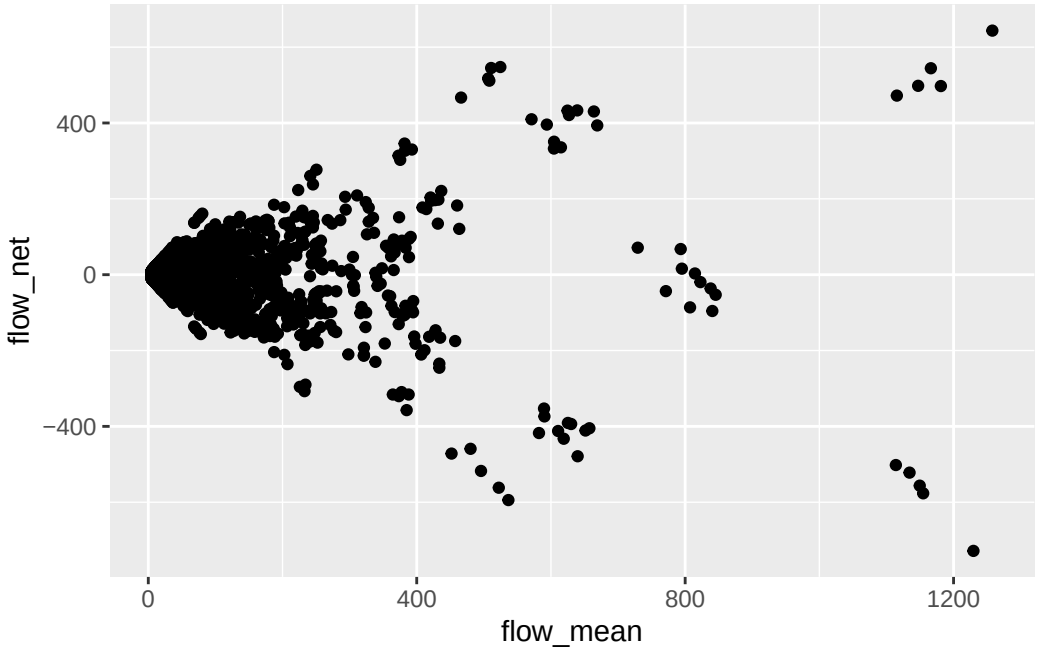
```
Warning: Transformation introduced infinite values in continuous y-axis
```




```
#Create scatterplot
edges %>% ggplot(aes(x = inbound, y = outbound)) +
  geom_point()
```



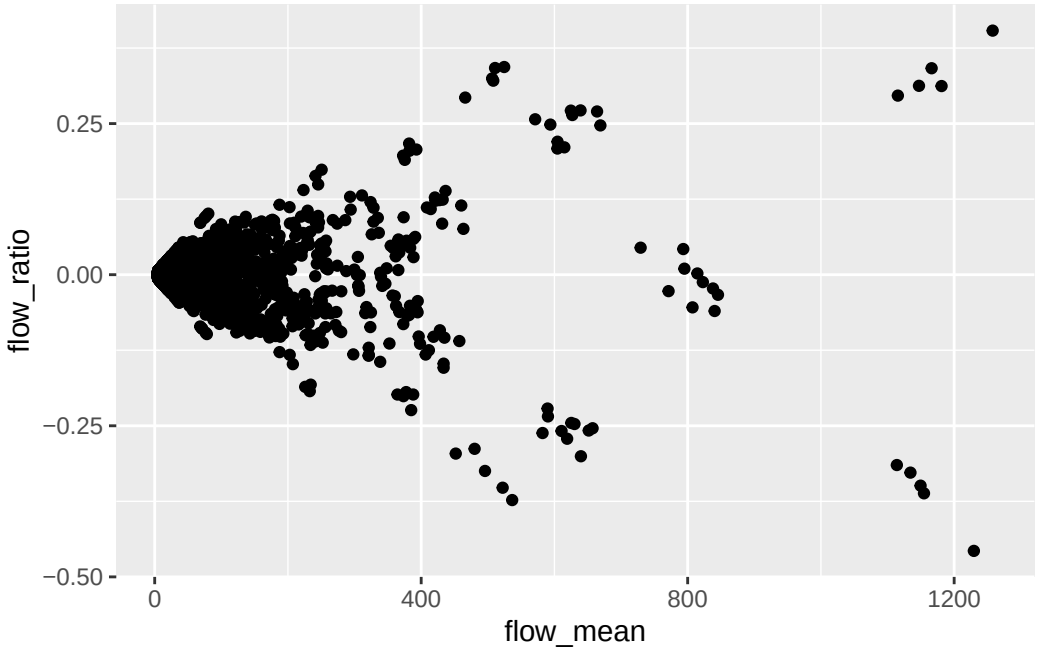

```
# Create scatterplot
edges %>% ggplot(aes(x = flow_mean, y = flow_net)) +
  geom_point()
```




```
#+  scale_x_log10()



# Create scatterplot
edges %>% ggplot(aes(x = flow_mean, y = flow_ratio)) +
  geom_point()
```




```
#+  scale_x_log10()
```

```
edges %>%
    group_by(origin)%>%
    summarize(ff = sum(flow_mean))%>%
    arrange(-ff)
```

```
# A tibble: 191 x 2
   origin         ff
   <chr>       <dbl>
 1 Udine      37587.
 2 Pordenone  24480.
 3 Trieste    17698.
 4 Monfalcone  9428.
 5 Gorizia     8658.
 6 Tavagnacco  7939.
 7 Porcia      6868.
 8 FiumeVen    6645.
 9 Cordenons   6286.
10 Muggia      6240.
# i 181 more rows
```

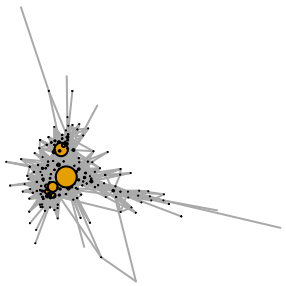
```
g <- igraph::graph_from_data_frame(edges,  directed = FALSE)
E(g)$weight <- edges$flow_mean
print(paste("g is simple: ",is.simple(g)))
```

```
[1] "g is simple:  FALSE"
```

```
g <- igraph::simplify(g,edge.attr.comb = "sum" )
print(paste("g is simple: ",is.simple(g)))
```

```
[1] "g is simple:  TRUE"
```

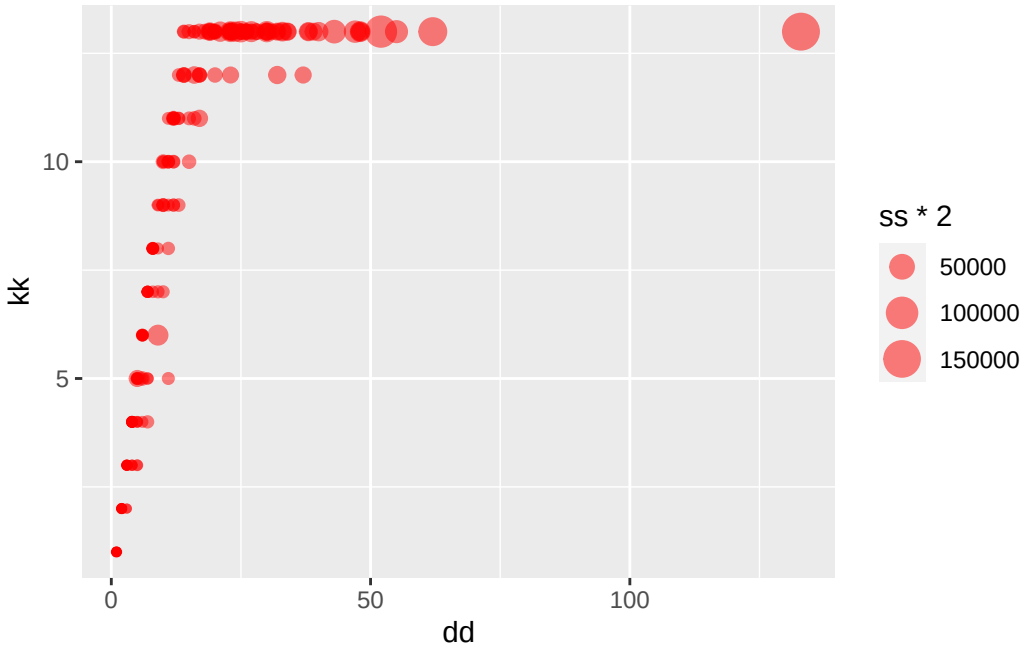
```
plot(g,
     vertex.label=NA,
     vertex.size = strength(g)/5000 )
```



check strength and coreness

```
df <- data.frame(loc = V(g)$name,
                 dd = degree(g),
                 ss = round(strength(g),0),
                 kk = coreness(g)) %>%
    arrange(-ss)

df %>% ggplot(aes(x = dd, y = kk)) + geom_point(aes(size = ss*2), color = 'red',alpha = 0.
```



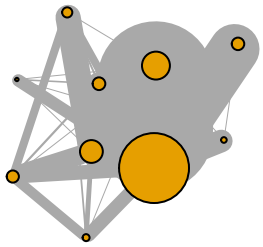

```
df %>% arrange(-ss) %>% head(10)
```

	loc	dd	ss	kk
Udine	Udine	133	76361	13
Pordenone	Pordenone	52	49112	13
Trieste	Trieste	62	35741	13
Monfalcone	Monfalcone	43	18983	13
Gorizia	Gorizia	55	17504	13
Tavagnacco	Tavagnacco	47	15749	13
Porcia	Porcia	25	13702	13
FiumeVen	FiumeVen	30	13198	13
Cordenons	Cordenons	27	12540	13
Muggia	Muggia	9	12402	6

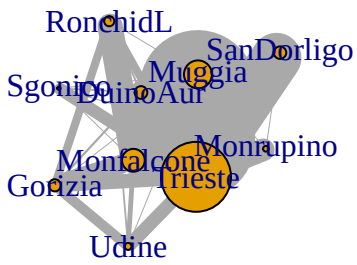
show neighborhood

```
g1 <- g
selected <- which(V(g1)$name == "Muggia")
nbh<-make_ego_graph(
  g1,
  order = 1,
  nodes = selected,
  mode = "all",
  mindist = 0
)[[1]]

lo = layout.graphopt(nbh)
plot(nbh, vertex.size = strength(nbh)/500, edge.width = E(nbh)$weight/200, vertex.label =
```



```
plot(nbh, vertex.size = strength(nbh)/500, edge.width = E(nbh)$weight/200,layout = lo)
```




```
file_name = "mobility_fvg_sample_01.graphml"
g %>% igraph::write_graph(file_name, format="graphml")
```