London Gang Network Analysis

Stan Ionel Eduard

February 2019

Introduction

We work on the London-based inner-city street gang data from 2005 to 2009, concerning black people, operating from a social housing estate that have co-offended together. Data is available on UCINET Software website and comes from anonymised police arrest and conviction data for **all confirmed** members of the gang.

The network is **undirected** and it has 54 nodes representing the monitored people having the following attributes:

- `Person`, representing the person's **identifier**,
- `Age`, representing the **age** of such person,
- `Birthplace`, representing the **birthplace** of such person (1 for West Africa, 2 for Caribbean, 3 for United Kingdom, and 4 for East Africa),
- `Residence`, representing the fact that the person was **resident**,
- `Arrests`, representing the **number of arrests** of such person,
- `Convictions`, representing the **number of convictions** of such person,
- `Prison`, representing the fact that such person had already been **arrested**,
- `Music`, representing the fact that such person **listened to music**,
- `Ranking`, representing a **ranking** of such person.

The ties of such network are given as a **weighted adjacency matrix**, i.e. a 54×54 (weighted) matrix, where the weights are:

- 1, representing the fact that the connected people have **hanged-out**,
- 2, representing the fact that the connected people have **co-offended together**,
- 3, representing the facts that the connected people have **co-offended together** and that they have **committed serious crimes**,
- 4, representing the facts that the connected people have **co-offended together**, **committed serious crimes** and that there is a **kin relationship** (i.e. a family relationship) in-between nodes.

In the subsequent analysis, we are going to use **nodes** and **people** interchangeably. The same holds for **edges**, **ties** and **relations**. We use the well-known **tidy approach** to make our analysis.

Our objective is the analysis of such network in terms of **centrality and power of people**. Moreover, we aim to find patterns between subjects based on their birthplace, therefore we focus on the **similarity between nodes** and present some **social validation facts**, thanks to the knowledge that we acquire during the analysis.

Installation

Before we begin our analysis, first we need to install the following packages:

```
install.packages("tidyverse") # core of the tidy approach
install.packages("igraph")    # igraph package for graphs
install.packages("ggraph")    # ggraph package for graphs
```

and then we load such packages:

```
library(tidyverse)
```

```
## -- Attaching packages ----------------------------------------------------------------------- tidyverse 1
```

```
## v ggplot2 3.0.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.6
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
```

```
## -- Conflicts -------------------------------------------------------------------------- tidyverse_conflic
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(igraph)
```

```
##
## Attaching package: 'igraph'
```

```
## The following objects are masked from 'package:dplyr':
##
##     as_data_frame, groups, union
```

```
## The following objects are masked from 'package:purrr':
##
##     compose, simplify
```

```
## The following object is masked from 'package:tidyr':
##
##     crossing
```

```
## The following object is masked from 'package:tibble':
##
##     as_data_frame
```

```
## The following objects are masked from 'package:stats':
##
##     decompose, spectrum
```

```
## The following object is masked from 'package:base':
##
##     union
```

```
library(ggraph)
```

Data import

We read the CSV files and store such information as `tibbles`:

```
# gather the relations from the CSV file
relations <- as_tibble(read.table("data/gang.csv", header=TRUE, sep=",")) %>%
  # select all columns, but not the first one
  select(X1:X54)

# gather the people from the CSV file
people <- read_csv("data/attr.csv", col_names=TRUE) %>%
  # rename the column X1 as Person
  rename(Person=X1)
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
## Parsed with column specification:
## cols(
##   X1 = col_integer(),
##   Age = col_integer(),
##   Birthplace = col_integer(),
##   Residence = col_integer(),
##   Arrests = col_integer(),
##   Convictions = col_integer(),
##   Prison = col_integer(),
##   Music = col_integer(),
##   Ranking = col_integer()
## )
```

Then we perform a preliminary unit test in order to check that the information is stored as we want:

```
dim(relations)   # 54 x 54 (i.e. the weighted adjacency matrix)
```

```
## [1] 54 54
```

```
class(relations) # print the class of the relations
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
head(relations)  # print the first observations of the relations
```

```
## # A tibble: 6 x 54
##      X1    X2    X3    X4    X5    X6    X7    X8    X9   X10   X11   X12
##   <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int>
## 1     0     1     1     2     1     1     2     3     2     2     3     1
## 2     1     0     3     0     0     2     1     2     1     2     2     1
## 3     1     3     0     4     4     3     1     3     2     2     0     1
## 4     2     0     4     0     4     3     1     0     0     0     0     1
## 5     1     0     4     4     0     3     1     0     1     1     0     1
## 6     1     2     3     3     3     0     1     0     0     0     0     1
## # ... with 42 more variables: X13 <int>, X14 <int>, X15 <int>, X16 <int>,
## #   X17 <int>, X18 <int>, X19 <int>, X20 <int>, X21 <int>, X22 <int>,
## #   X23 <int>, X24 <int>, X25 <int>, X26 <int>, X27 <int>, X28 <int>,
## #   X29 <int>, X30 <int>, X31 <int>, X32 <int>, X33 <int>, X34 <int>,
## #   X35 <int>, X36 <int>, X37 <int>, X38 <int>, X39 <int>, X40 <int>,
## #   X41 <int>, X42 <int>, X43 <int>, X44 <int>, X45 <int>, X46 <int>,
## #   X47 <int>, X48 <int>, X49 <int>, X50 <int>, X51 <int>, X52 <int>,
## #   X53 <int>, X54 <int>
```

```
dim(people)      # 54 x 9 (i.e. 54 people with their own attributes)
```

```
## [1] 54  9
```

```
class(people)    # print the class of the people
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
head(people)     # print the first observations of the people
```

```
## # A tibble: 6 x 9
##   Person   Age Birthplace Residence Arrests Convictions Prison Music
##    <int> <int>      <int>     <int>   <int>       <int>  <int> <int>
## 1      1    20          1         0      16           4      1     1
```

```
## 2      2    20          2          0      16           7      1      0
## 3      3    19          2          0      12           4      1      0
## 4      4    21          2          0       8           1      0      0
## 5      5    24          2          0      11           3      0      0
## 6      6    25          3          1      17          10      0      0
## # ... with 1 more variable: Ranking <int>
```

Graph generation

We need to manipulate the `relations` data frame in order to build a graph. We generate some containers (i.e. `from`, `to`, and `weight`) that store the information about such weighted `relations` as:

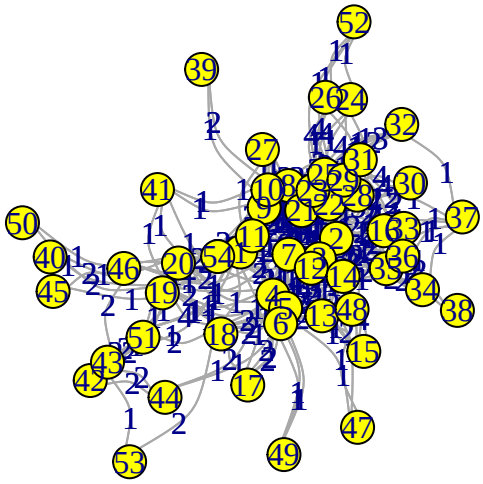
```
from <- to <- weight <- NULL # containers
# for each row
for (i in 1:nrow(relations)) {
  # for each column
  for (j in 1:ncol(relations)) {
    if (relations[i,j] != 0) {
      # attach an observation for each edge: (from, to, weight)
      from <- c(from, i)
      to <- c(to, j)
      weight <- c(weight, relations[i,j])
    }
  }
}
```

and create 2 `tibbles`, one for the `ties` and one for the `nodes`, as:

```
# ties
ties <- tibble(
  from = from,            # from
  to = to,                # to
  weight = unlist(weight) # weight
)
# nodes
nodes <- tibble(
  person = 1:nrow(people),                # person
  age = people$Age,                       # age
  birthplace = factor(people$Birthplace), # birthplace
  residence = factor(people$Residence),   # residence
  arrests = people$Arrests,               # arrests
  convinctions = people$Convictions,      # convictions
  prison = factor(people$Prison),         # prison
  music = factor(people$Music),           # music
  ranking = factor(people$Ranking)        # ranking
)
```

Now, we are ready to generate our graph using the `igraph` package and to (preliminarly) plot such object as:

```
# create an undirected graph
g <- graph_from_data_frame(ties, directed=FALSE, vertices = nodes)
# plot the graph
plot(g, layout=layout_nicely(g),
     vertex.size=15, vertex.color='yellow', vertex.shape="circle",
     edge.label=ties$weight, edge.curved=TRUE)
```



Moreover, we can check the attributes of such graph as:

```
attributes(summary(g)) # explore the attributes
```

```
## IGRAPH 04fb8a3 UNW- 54 630 --
## + attr: name (v/c), age (v/n), birthplace (v/c), residence (v/c),
## | arrests (v/n), convinctions (v/n), prison (v/c), music (v/c),
## | ranking (v/c), weight (e/n)
```

```
## $class
## [1] "igraph"
```

Explanatory data analysis (EDA)

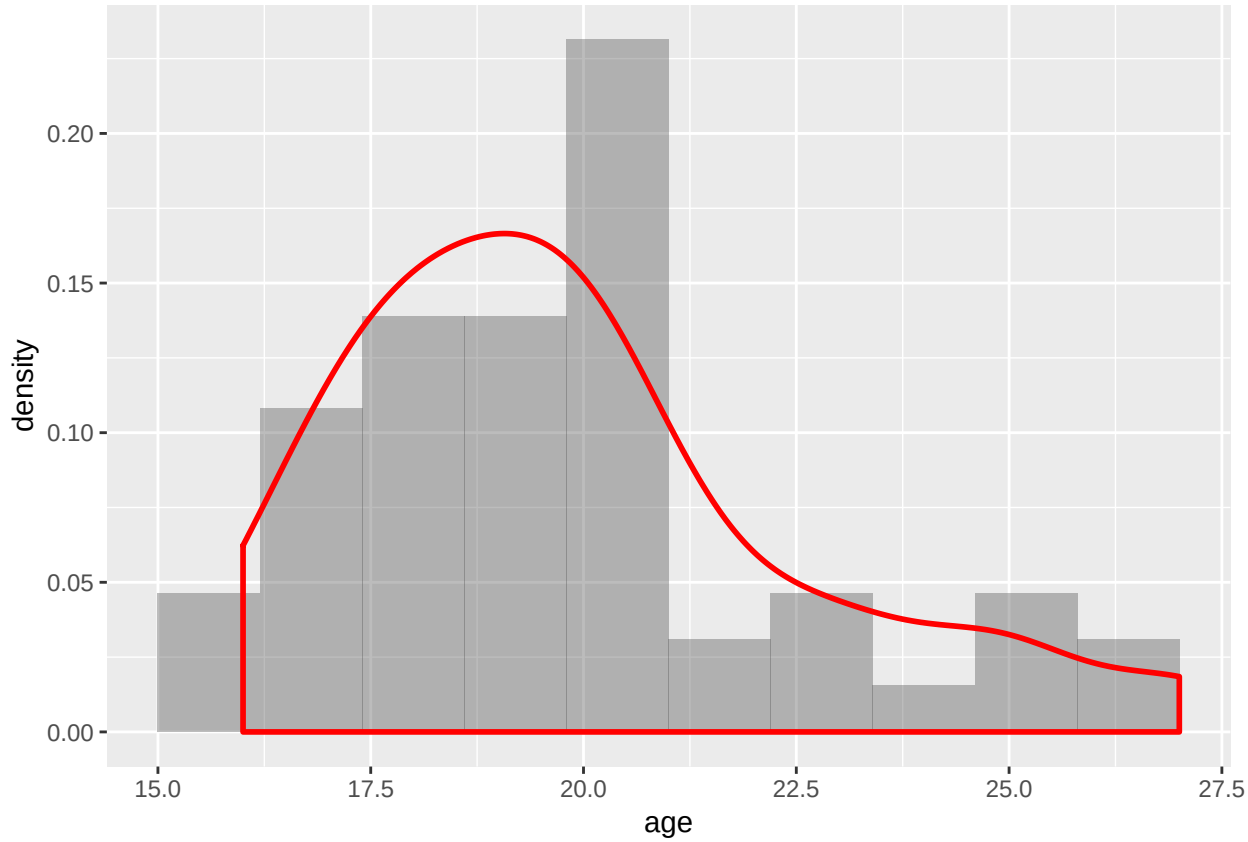
In this section we are going to explore our dataset in order to find some preliminary patterns.

In Statistics, a **rule of thumb**, when performing EDA, is to present graphical plots with (numerical) summaries. Therefore, we present such information as:

```
summary(nodes$age)                          # summary for age
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   16.00   18.00   19.00   19.83   21.00   27.00
```

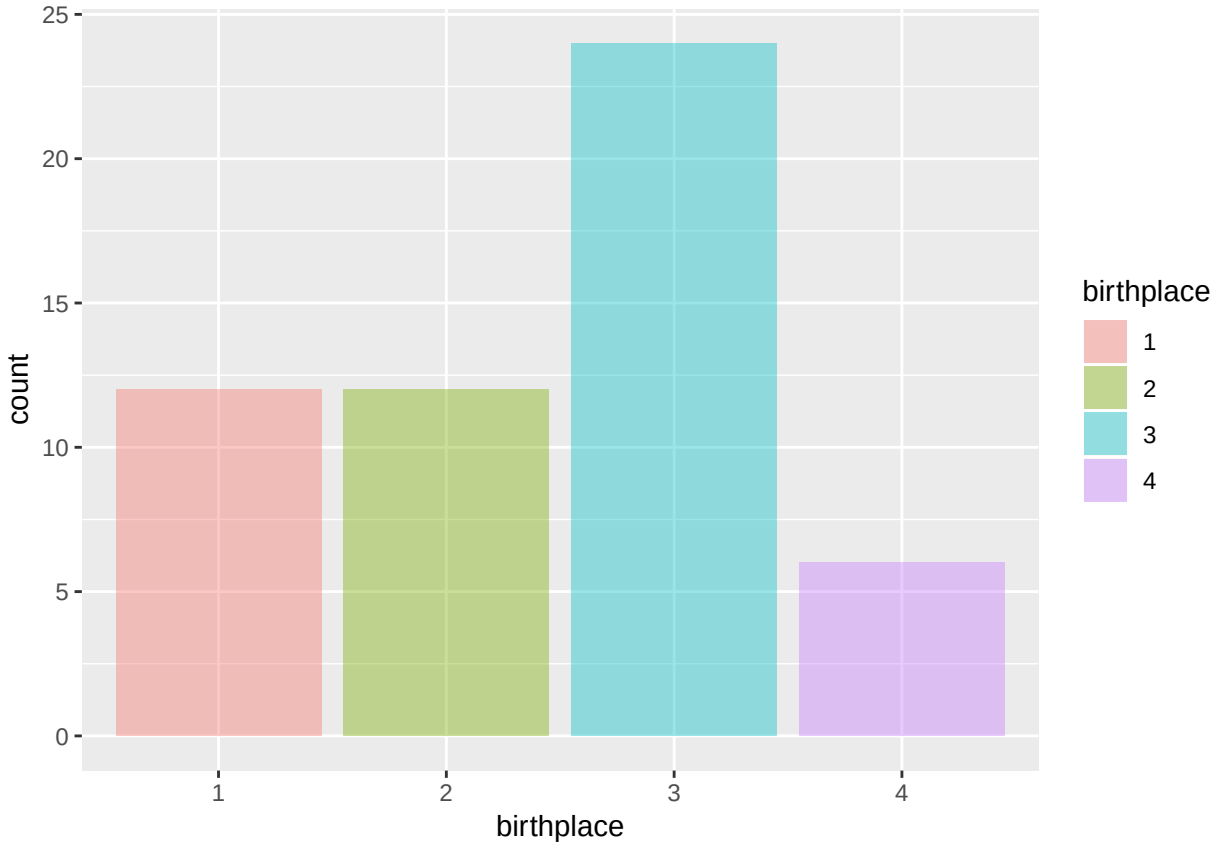
```
ggplot(nodes, aes(x=age, stat(density))) +  # distribution for age
  geom_histogram(binwidth=1.2, alpha=2/5) + # histogram
  geom_density(color="red", size=1)         # density estimate
```




```
table(nodes$birthplace)                    # contingency table for birthplace
```

```
##
##  1  2  3  4
## 12 12 24  6
```

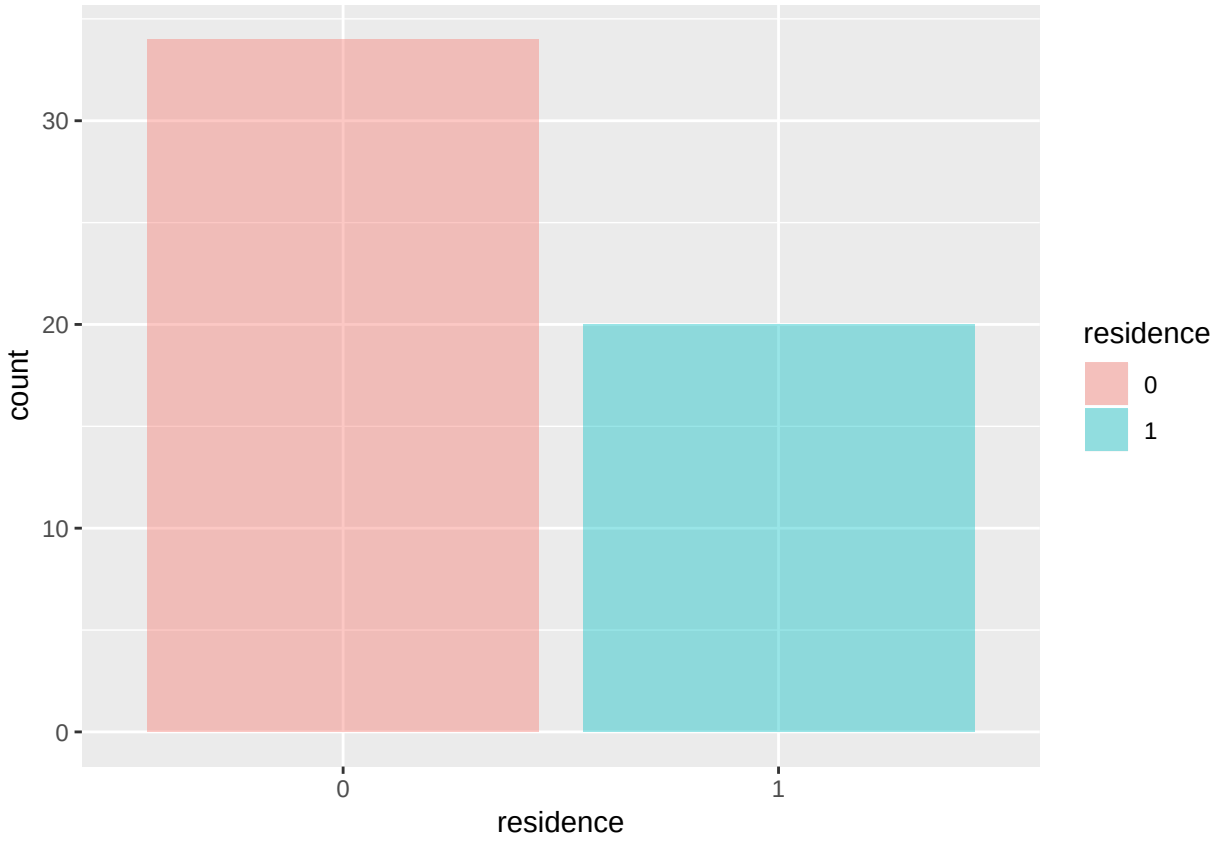
```
ggplot(nodes, aes(x=birthplace)) +         # plot such contingency table
  geom_bar(aes(fill=birthplace), alpha=2/5) # barplot
```




```
table(nodes$residence)                    # contingency table for residence
```

```
##
##  0  1
## 34 20
```

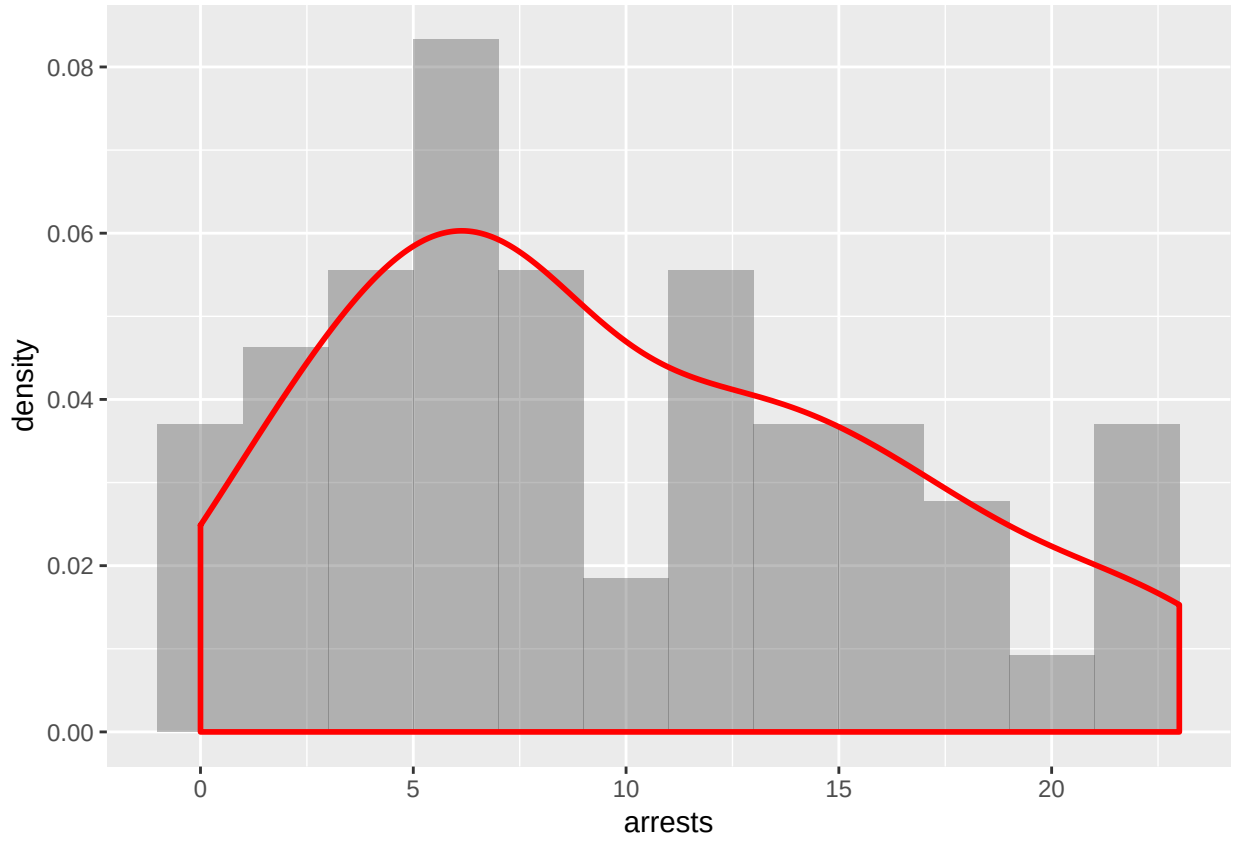
```
ggplot(nodes, aes(x=residence)) +         # plot such contingency table
  geom_bar(aes(fill=residence), alpha=2/5) # barplot
```




```
summary(nodes$arrests)                          # summary for arrests
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   5.000   8.000   9.907  14.750  23.000
```

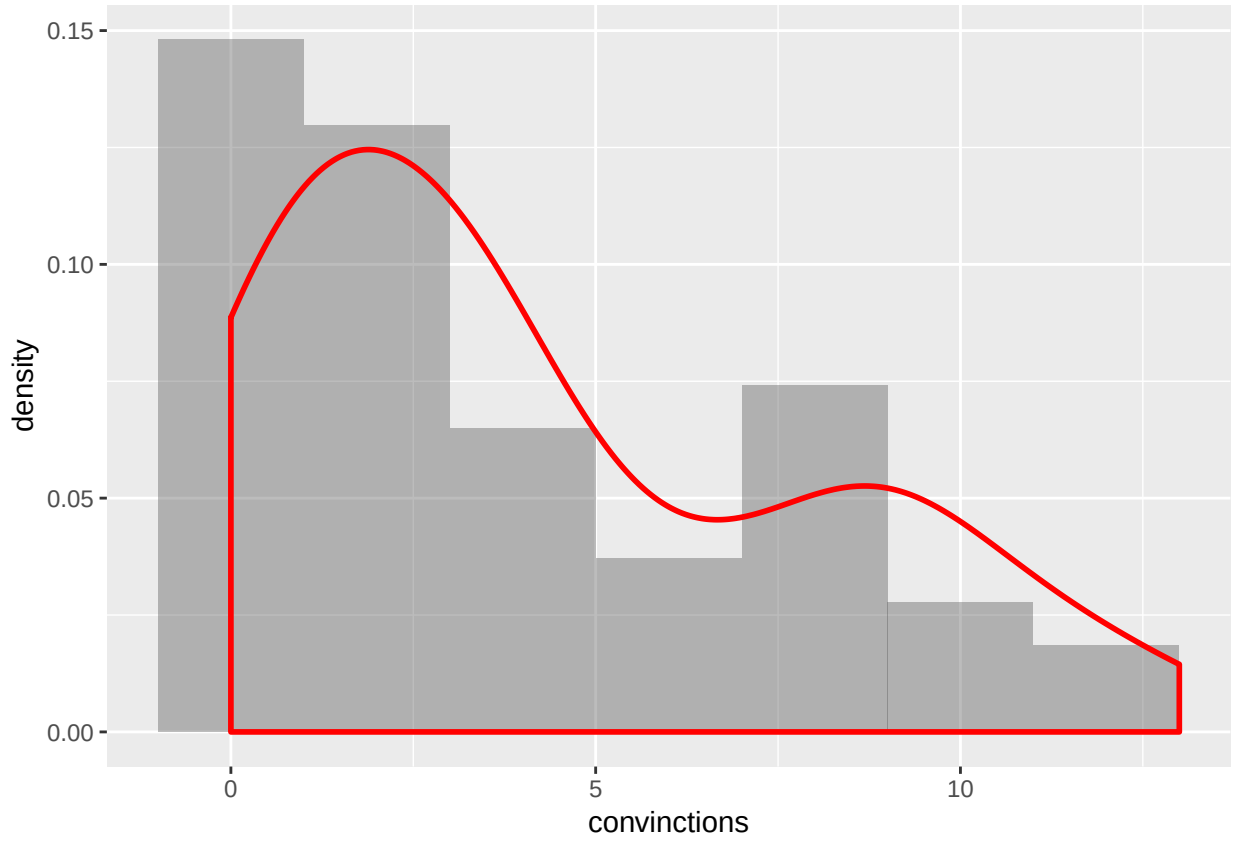
```
ggplot(nodes, aes(x=arrests, stat(density))) +  # distribution for arrests
  geom_histogram(binwidth=2, alpha=2/5) +       # histogram
  geom_density(color="red", size=1)             # density estimate
```




```
summary(nodes$convinctions)                          # summary for convictions
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   1.000   3.000   4.204   7.000  13.000
```

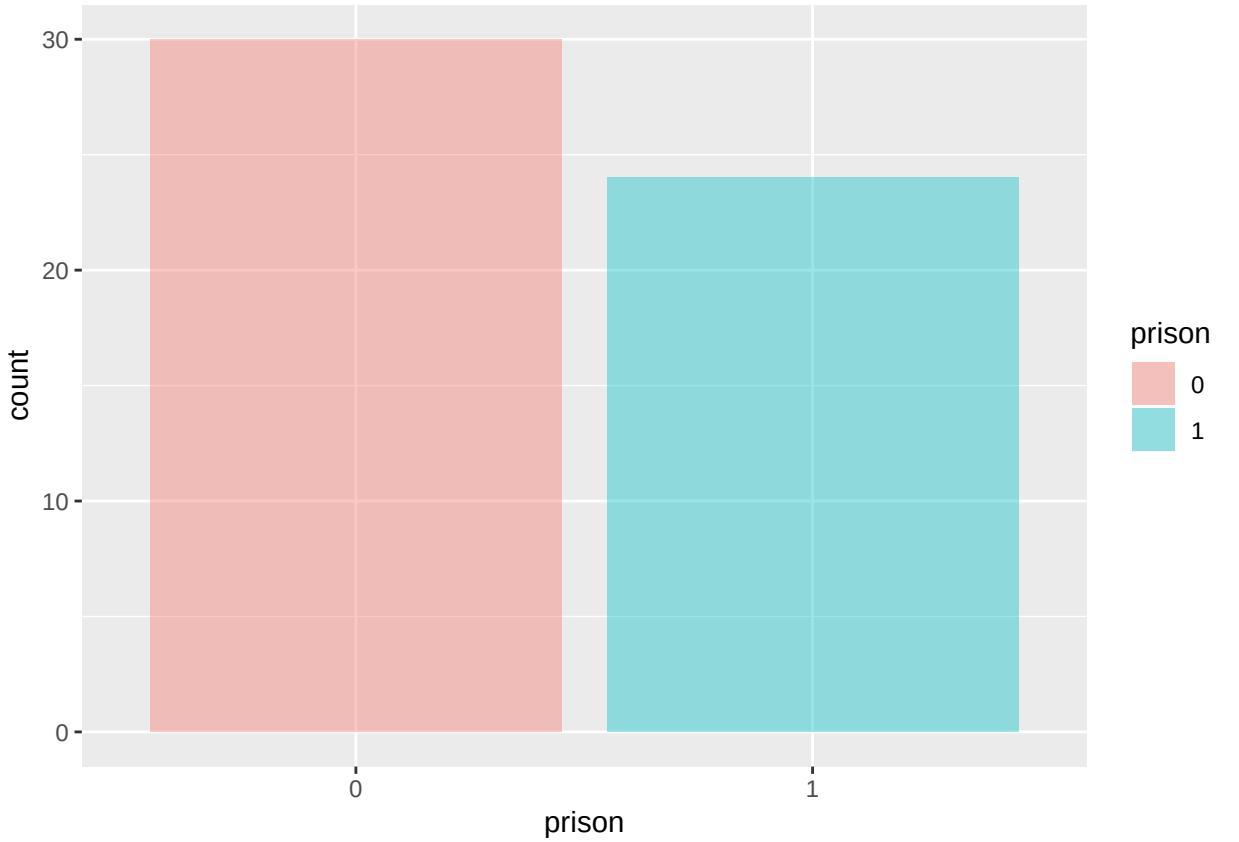
```
ggplot(nodes, aes(x=convinctions, stat(density))) +  # distribution for convictions
  geom_histogram(binwidth=2, alpha=2/5) +            # histogram
  geom_density(color="red", size=1)                  # density estimate
```




```
table(nodes$prison)                    # contingency table for prison
```

```
##
##  0  1
## 30 24
```

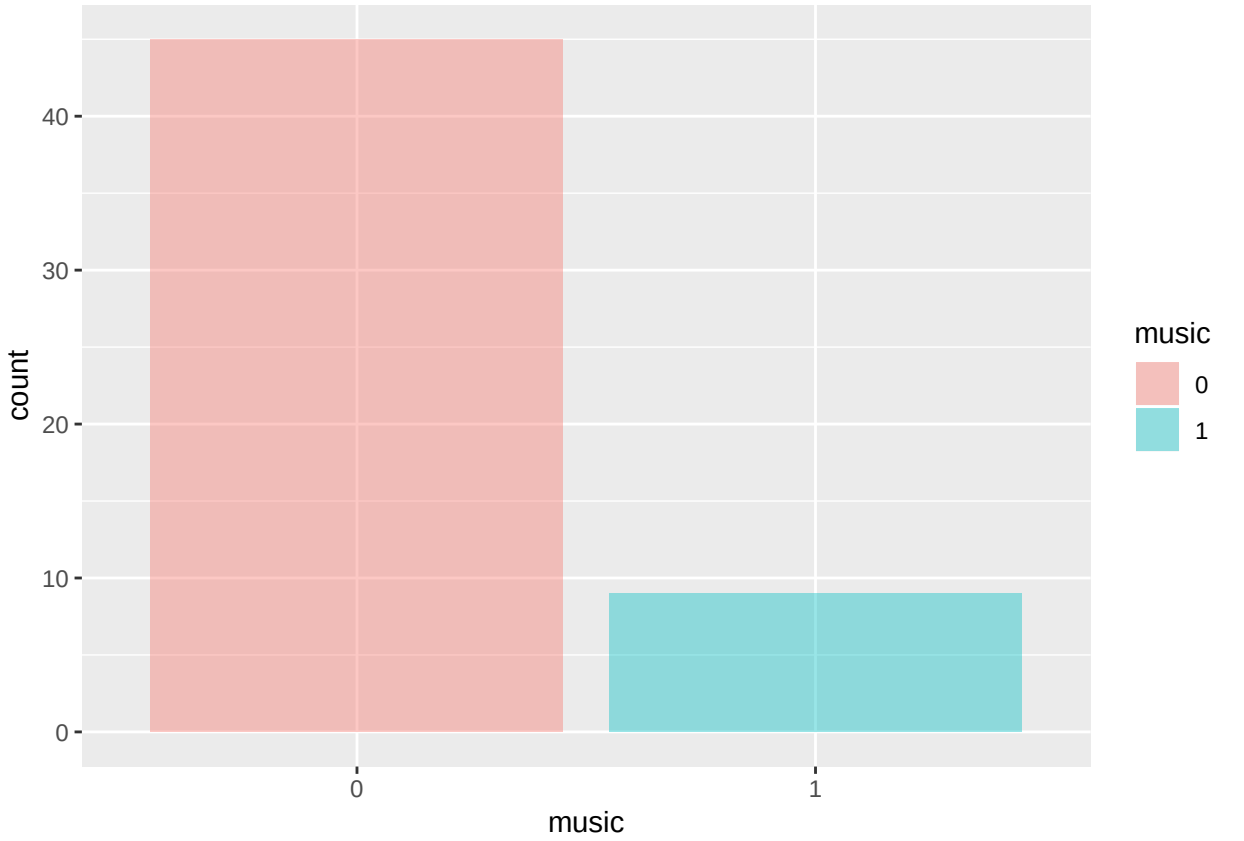
```
ggplot(nodes, aes(x=prison)) +         # plot such contingency table
  geom_bar(aes(fill=prison), alpha=2/5) # barplot
```




```
table(nodes$music)                  # contingency table for music
```

```
##
##  0  1
## 45  9
```

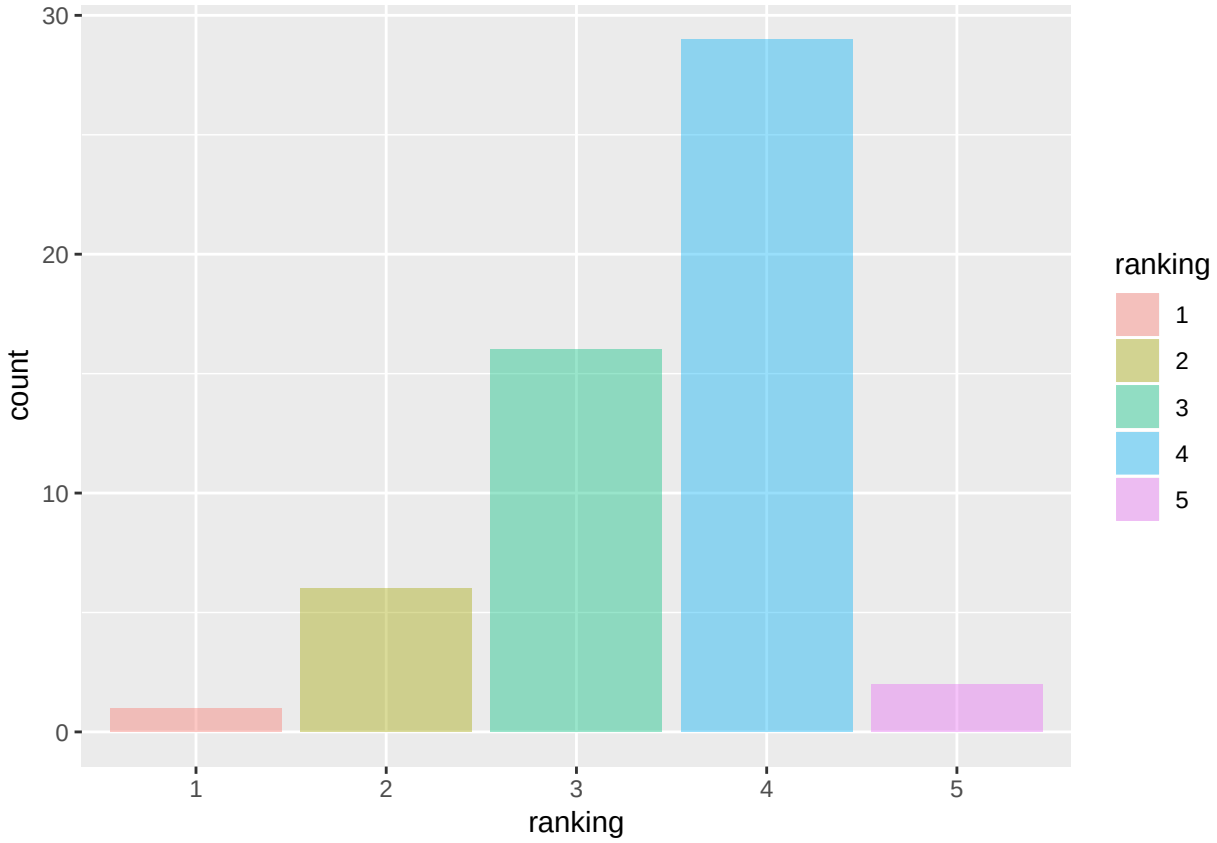
```
ggplot(nodes, aes(x=music)) +       # plot such contingency table
  geom_bar(aes(fill=music), alpha=2/5) # barplot
```




```
table(nodes$ranking)                    # contingency table for ranking
```

```
##
##  1  2  3  4  5
##  1  6 16 29  2
```

```
ggplot(nodes, aes(x=ranking)) +         # plot such contingency table
  geom_bar(aes(fill=ranking), alpha=2/5) # barplot
```

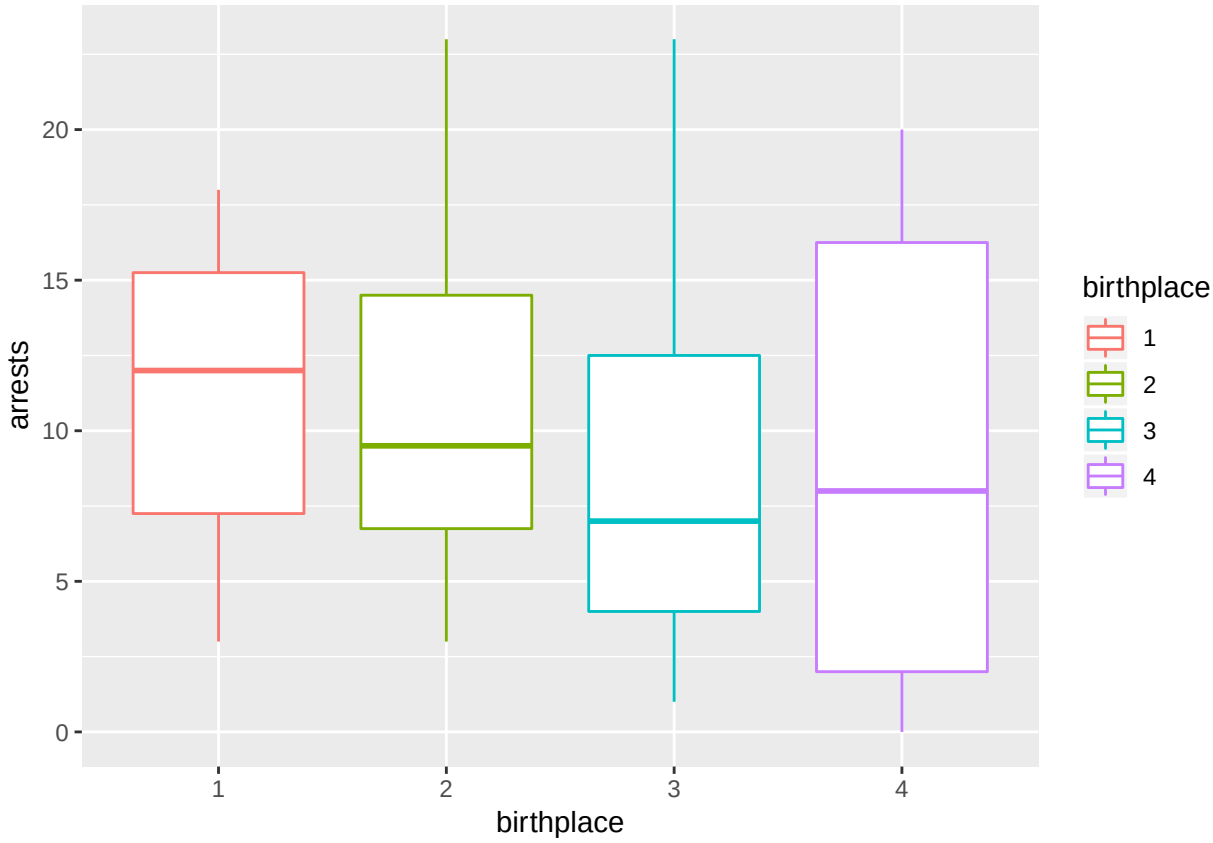


Suppose that we would like to know:

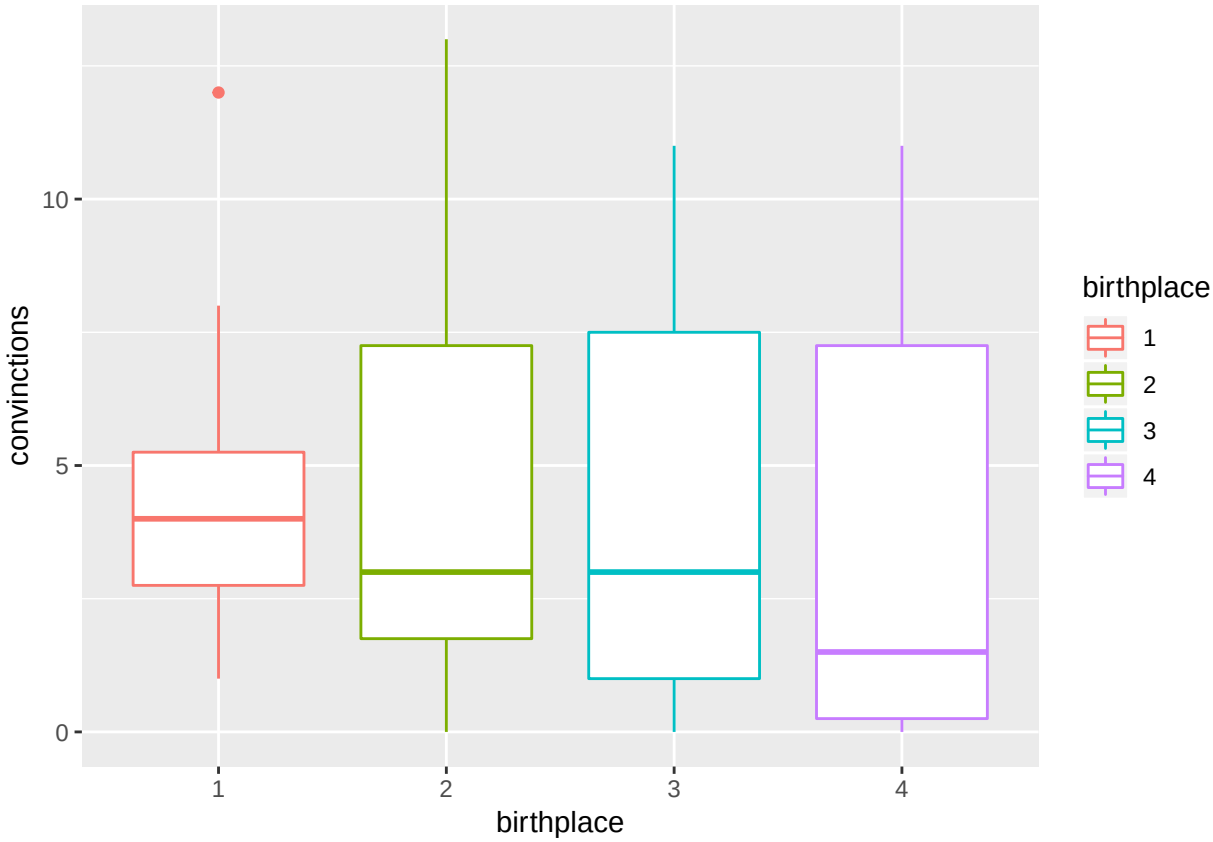
How the arrests and convinctions change as a function of birthplace and residence?

We could extract such information as:

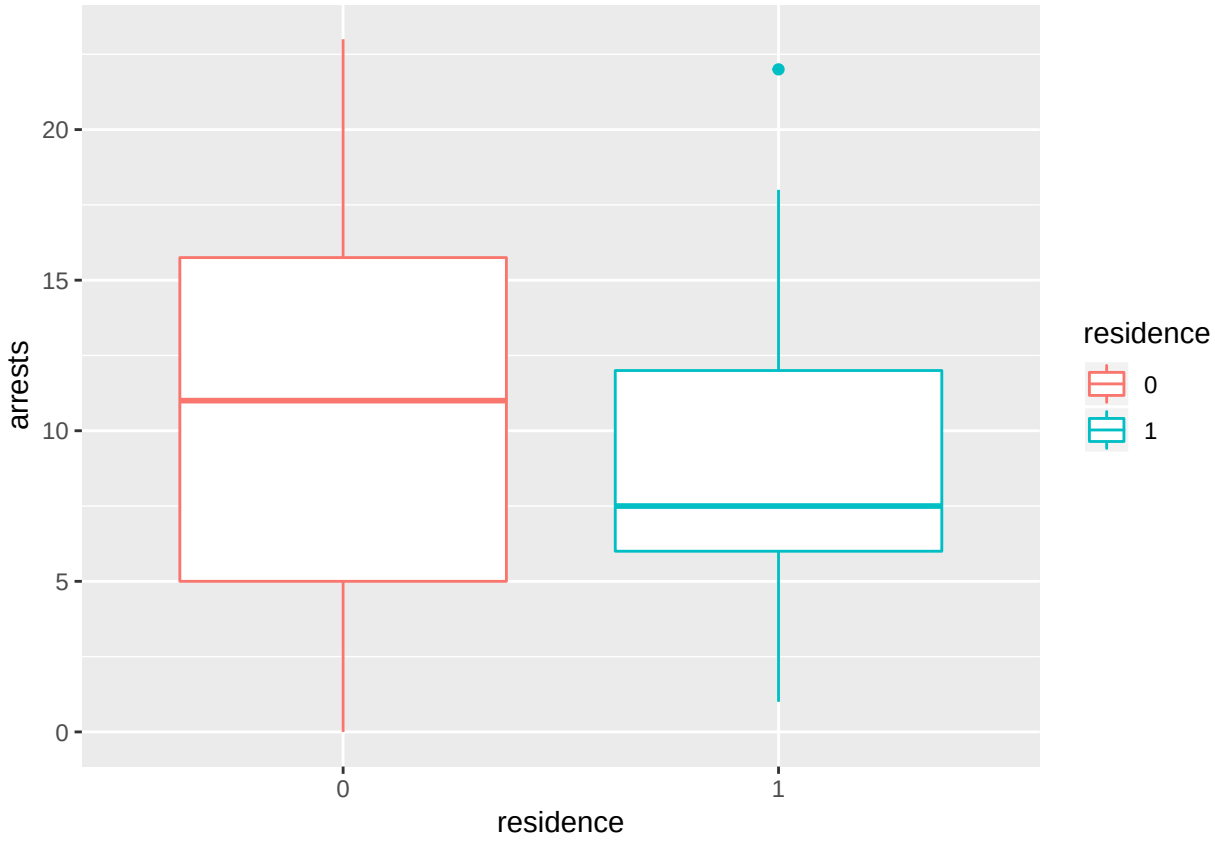
```
ggplot(nodes, aes(x=birthplace, y=arrests)) + # arrests as a function of birthplace
  geom_boxplot(aes(color=birthplace))         # boxplot arrests vs birthplace
```



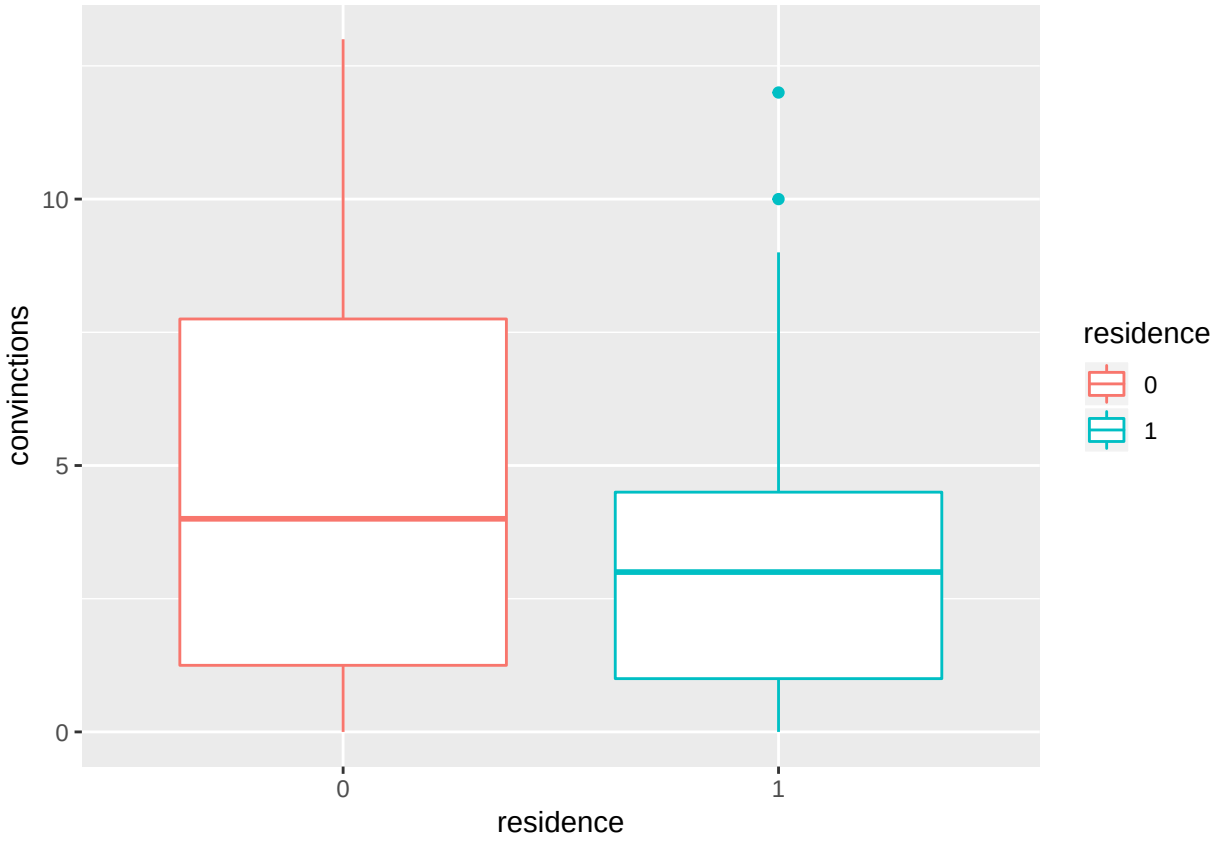

```
ggplot(nodes, aes(x=birthplace, y=convinctions)) + # convinctions as a function of birthplace
  geom_boxplot(aes(color=birthplace))                # boxplot convinctions vs birthplace
```




```
ggplot(nodes, aes(x=residence, y=arrests)) + # arrests as a function of residence
  geom_boxplot(aes(color=residence))         # boxplot arrests vs residence
```




```
ggplot(nodes, aes(x=residence, y=convinctions)) + # convinctions as a function of residence
  geom_boxplot(aes(color=residence))              # boxplot convinctions vs residence
```



Note that, although we have a small dataset, we can say that **people from West Africa** (i.e. `birthplace` is equal to 1) **tend to be more criminals** than others; witnessed by the fact that the arrests and convinctions are (in general) higher. Moreover, **people without a residence tend to be more criminal**, and the reasoning is similar.

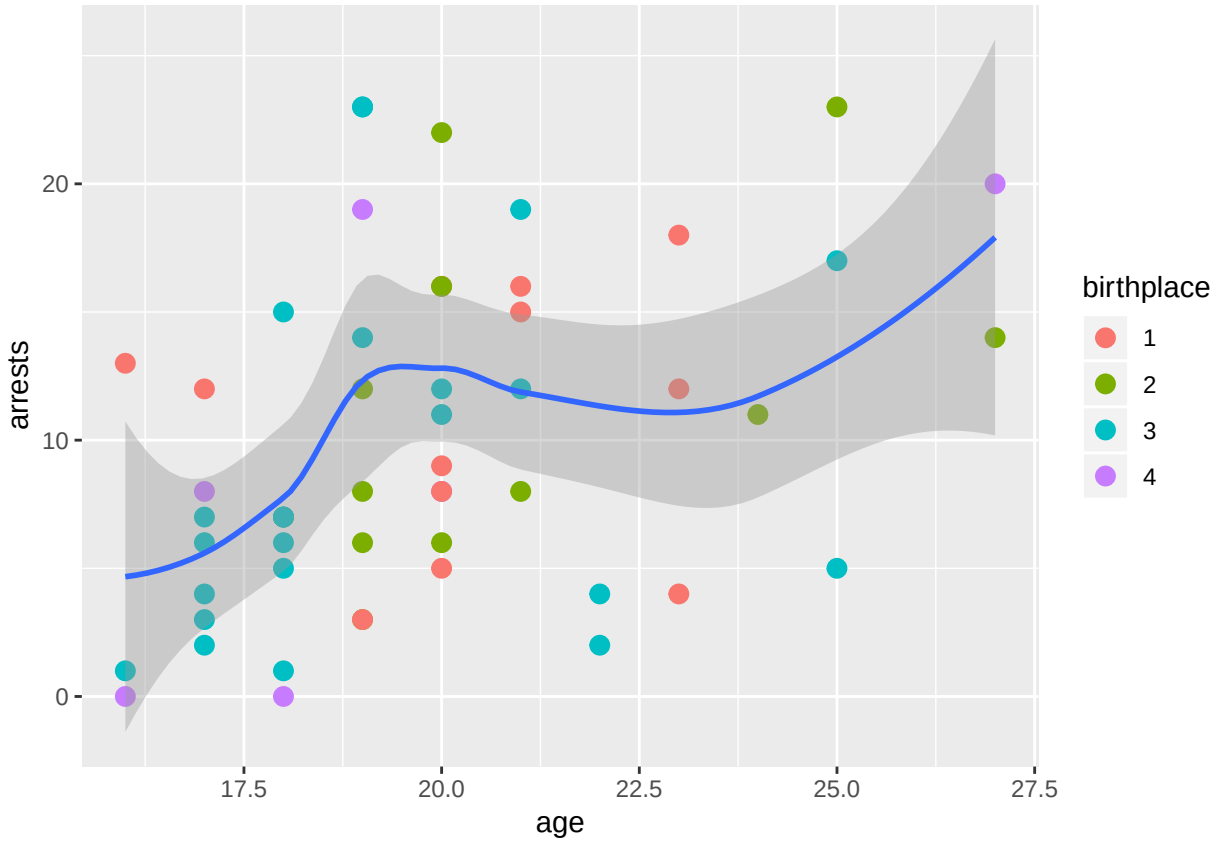
Now, suppose that we would like to know:

> How arrests and convinctions change as a function of age?

We can extract such information through a scatterplot in the following way:

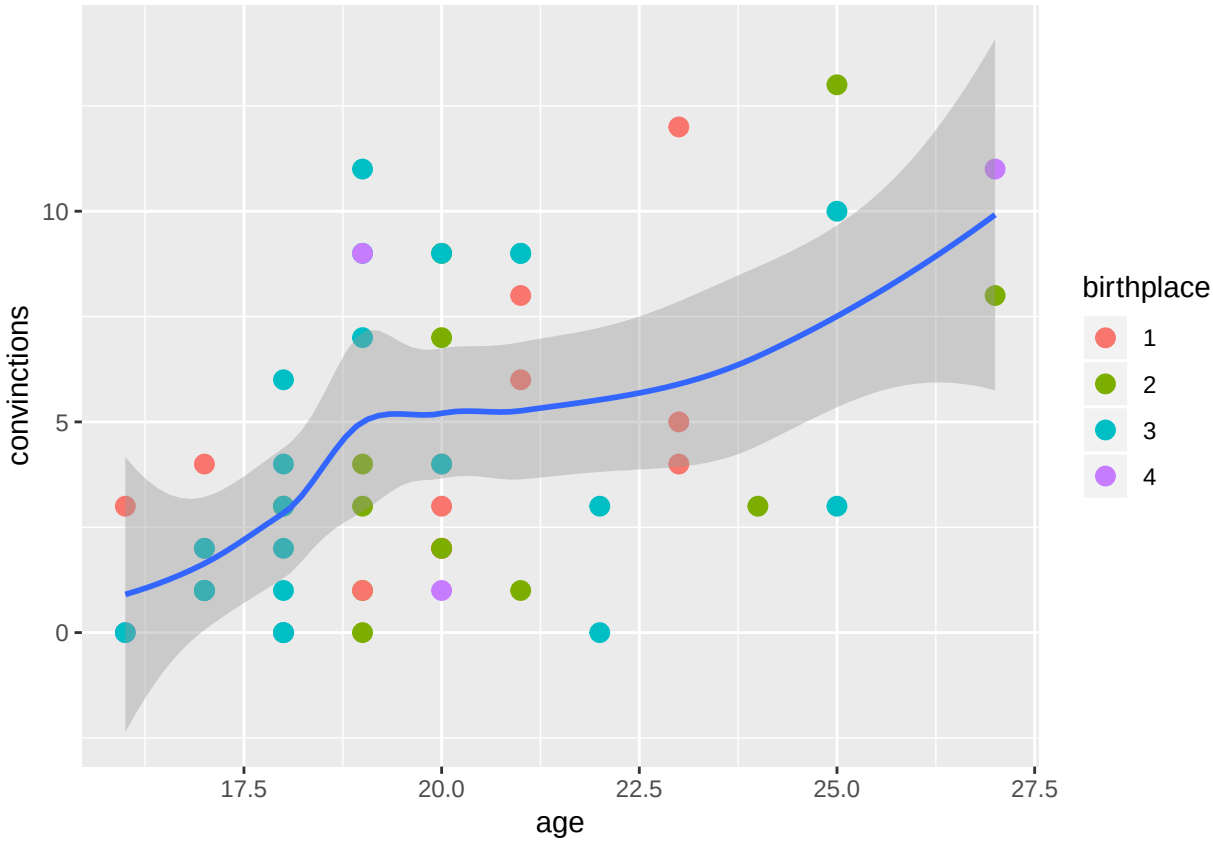
```
ggplot(nodes, aes(x=age, y=arrests)) +         # arrests as a function of age
  geom_point(aes(color=birthplace), size=3) + # scatterplot
  geom_smooth()                               # smooth line through points (with standard error)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




```
ggplot(nodes, aes(x=age, y=convinctions)) +    # convinctions as a function of age
  geom_point(aes(color=birthplace), size=3) + # scatterplot
  geom_smooth()                               # smooth line through points (with standard error)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Note that, we have an expected behaviour, that is, **people tend to be less criminal until a certain age**, in this case $\approx$ 18 years old, then they tend to be more criminal. Perhaps this is due the fact that until a "fragile" age, the parents of such individuals assume the full-responsability of their own childs, but then such individuals are grown-up and they are more likely to act as criminals.

Centrality measures

We can simply add centralities to nodes with the following commands:

```
V(g)$degree <- degree(g)            # degree centrality
V(g)$betweenness <- betweenness(g) # betweenness centrality
V(g)$closeness <- closeness(g)      # closeness centrality
V(g)$pr <- page_rank(g)$vector      # pagerank centrality
attributes(summary(g))              # investigate the (new added) attributes
```

```
## IGRAPH 04fb8a3 UNW- 54 630 --
## + attr: name (v/c), age (v/n), birthplace (v/c), residence (v/c),
## | arrests (v/n), convinctions (v/n), prison (v/c), music (v/c),
## | ranking (v/c), degree (v/n), betweenness (v/n), closeness (v/n),
## | pr (v/n), weight (e/n)
```

```
## $class
## [1] "igraph"
```

In order to use the `dplyr` package, which is already included by the `tidyverse`, we add the same information to the `nodes` as:

```
nodes<- nodes %>%
  mutate(
    degree = degree(g),            # degree centrality
    betweenness = betweenness(g), # betweeness centrality
    closeness = closeness(g),      # closeness centrality
    pr = page_rank(g)$vector)      # pagerank centrality
```

Let's start with a **simple question**:

> Which is the most central person w.r.t. the centrality measures?

To answer this question, we sort in a decreasing fashion the nodes by centralities as:

```
nodes %>%
  select(person,degree,betweenness,closeness,pr) %>%
  arrange(desc(degree))      # arrange by degree centrality (decreasing order)
```

```
## # A tibble: 54 x 5
##    person degree betweenness closeness     pr
##     <int>  <dbl>       <dbl>     <dbl>  <dbl>
##  1      1     50       226.    0.00962 0.0263
##  2      7     50       137.    0.00926 0.0247
##  3     12     50       207.    0.0103  0.0213
##  4     14     48        51.9   0.00813 0.0308
##  5     22     48        65.7   0.00840 0.0236
##  6     23     46        37.3   0.00813 0.0280
##  7     25     46       100.    0.00962 0.0258
##  8      2     44        73.8   0.00893 0.0250
##  9      3     44        31.9   0.00820 0.0260
## 10     10     44        60.8   0.00840 0.0266
## # ... with 44 more rows
```

```
nodes %>%
  select(person,degree,betweenness,closeness,pr) %>%
  arrange(desc(betweenness)) # arrange by betweenness centrality (decreasing order)
```

```
## # A tibble: 54 x 5
##    person degree betweenness closeness     pr
##     <int>  <dbl>       <dbl>     <dbl>  <dbl>
##  1      1     50       226.    0.00962 0.0263
##  2     12     50       207.    0.0103  0.0213
##  3      7     50       137.    0.00926 0.0247
##  4      4     42       123.    0.00877 0.0262
##  5     20     30       117.    0.00730 0.0242
##  6     28     36       110.    0.00926 0.0197
##  7      5     38       105.    0.00917 0.0253
##  8     25     46       100.    0.00962 0.0258
##  9      9     42        83.9   0.00862 0.0249
## 10      2     44        73.8   0.00893 0.0250
## # ... with 44 more rows
```

```
nodes %>%
  select(person,degree,betweenness,closeness,pr) %>%
  arrange(desc(closeness))   # arrange by closeness centrality (decreasing order)
```

```
## # A tibble: 54 x 5
##    person degree betweenness closeness     pr
```

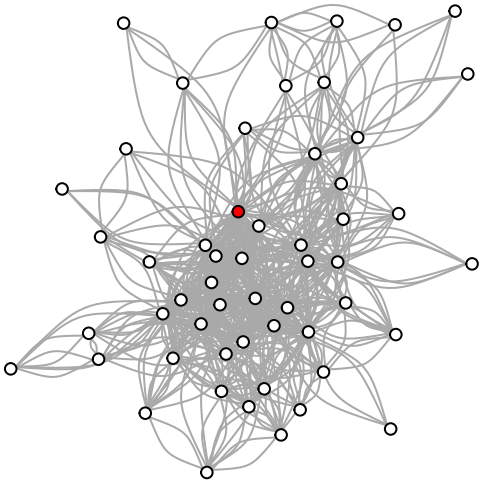
```
##     <int>  <dbl>       <dbl>     <dbl>  <dbl>
##  1     12     50      207.     0.0103  0.0213
##  2      1     50      226.     0.00962 0.0263
##  3     25     46      100.     0.00962 0.0258
##  4      7     50      137.     0.00926 0.0247
##  5     28     36      110.     0.00926 0.0197
##  6      5     38      105.     0.00917 0.0253
##  7      2     44       73.8    0.00893 0.0250
##  8      4     42      123.     0.00877 0.0262
##  9      9     42       83.9    0.00862 0.0249
## 10     29     34        8.04   0.00855 0.0215
## # ... with 44 more rows
```

```
nodes %>%
  select(person,degree,betweenness,closeness,pr) %>%
  arrange(desc(pr))          # arrange by pagerank centrality (decreasing order)
```

```
## # A tibble: 54 x 5
##    person degree betweenness closeness     pr
##     <int>  <dbl>       <dbl>     <dbl>  <dbl>
##  1     14     48        51.9   0.00813 0.0308
##  2     23     46        37.3   0.00813 0.0280
##  3     10     44        60.8   0.00840 0.0266
##  4      1     50       226.    0.00962 0.0263
##  5      4     42       123.    0.00877 0.0262
##  6      3     44        31.9   0.00820 0.0260
##  7     19     26        41.3   0.00781 0.0259
##  8     25     46       100.    0.00962 0.0258
##  9      5     38       105.    0.00917 0.0253
## 10      2     44        73.8   0.00893 0.0250
## # ... with 44 more rows
```

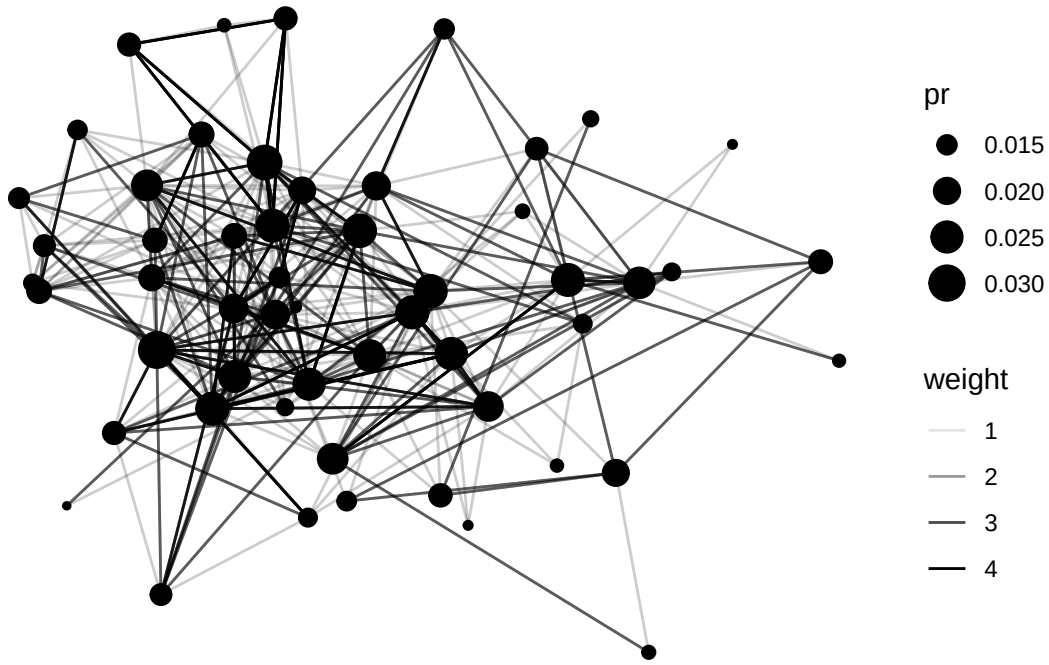
Note that, for this network, more or less, all centralities agree that node 1 is the most central. Therefore, we can point out such node in the network as:

```
V(g)$color <- "white"  # set the color for all nodes to white
V(g)[1]$color <- "red" # set the color for node 1 to red
plot(g, layout=layout_with_fr(g), vertex.label=NA, vertex.size = 5)
```



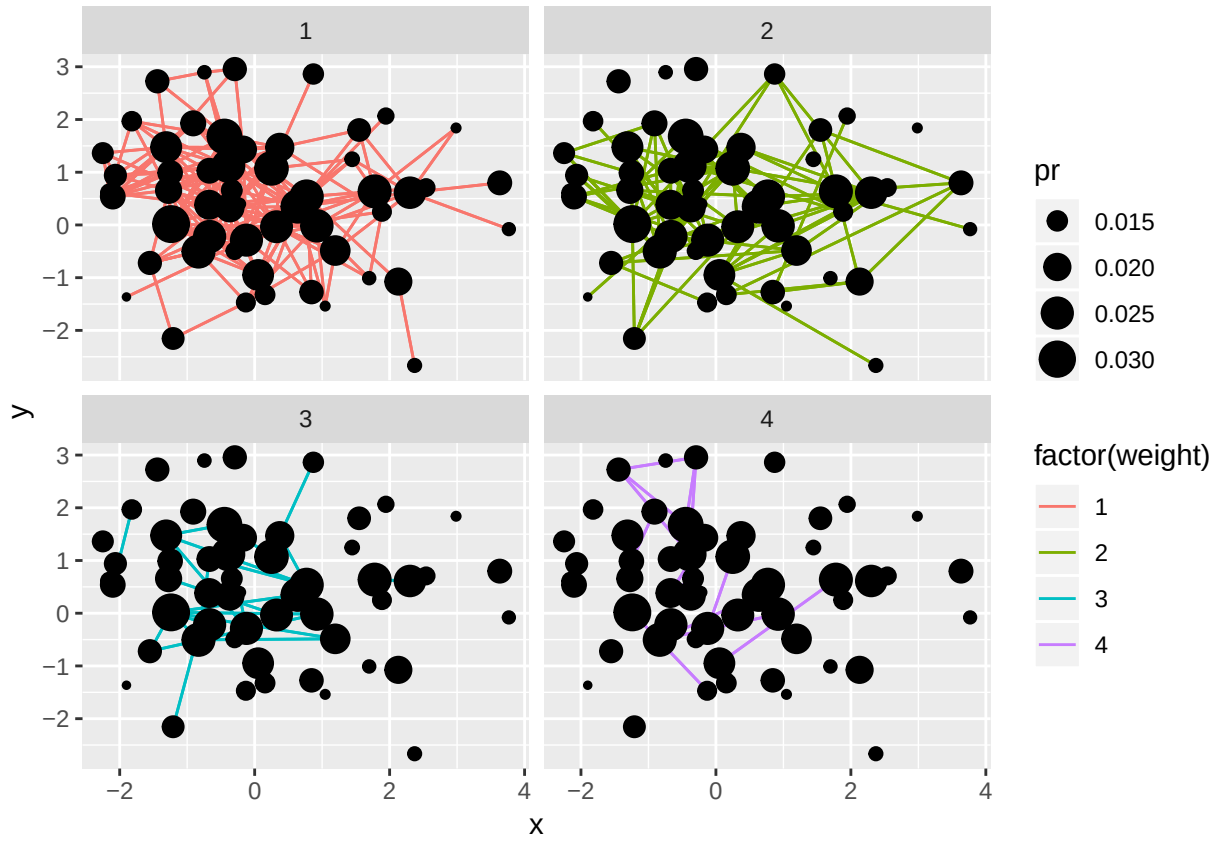
We can use the `ggraph` package to plot a beautiful graph, where the edges are more or less transparent based on their weights (`weight`) and the nodes are resized based on their PageRank (`pr`) centrality:

```
ggraph(g, layout = "with_kk") +
  geom_edge_link(aes(alpha = weight)) + # edge link geom, mapping alpha to weight
  geom_node_point(aes(size = pr)) +     # node point geom, mapping size to pagerank centrality
  theme_graph(base_family="sans")
```



or perhaps a more intuitive plot is the following:

```
ggraph(g, layout = "with_kk") +
  geom_edge_link(aes(color = factor(weight))) + # edge link geom, mapping color to weight
  geom_node_point(aes(size = pr)) +             # node point geom, mapping size to pr centrality
  facet_edges(~factor(weight))                  # facet geom as function of weight
```



where we point out the relations between nodes. For instance, if `weight` is 4, then the link means that the connected nodes have co-offended together, committed serious crimes and there is a kin relationship (bottom, right); this brings us to the next question:

What is the proportion of weak ties in the network?

Weak ties are very important for several reasons. For instance, strengthening the weak ties may lead to better opportunities. In our case, one will be more likely to commit crimes with others (**which is not so good, isn't it?**). We can compute such proportion in a straightforward way:

```
ties %>%
  group_by(weight) %>%                                  # group by weight
  summarise(n = n(), proportion = n / nrow(ties)) %>% # compute summary
  arrange(desc(n))                                      # sort in decreasing order by n
```

```
## # A tibble: 4 x 3
##   weight     n proportion
##    <int> <int>      <dbl>
## 1      1   364     0.578
## 2      2   184     0.292
## 3      3    50     0.0794
## 4      4    32     0.0508
```

It seems that $\approx 58\%$ of the people in the network have hanged-out with each other: perhaps **hanging out with the "wrong" people may lead people to commit (serious) crimes**. Therefore, reasoning by complementation, there are $\approx 42\%$ of the people that have done some crimes together.

Connection between nodes

Finding connection patterns between nodes is perhaps one of the **most interesting things** that an analyst would like to explore. We would like to given an answer the following question:

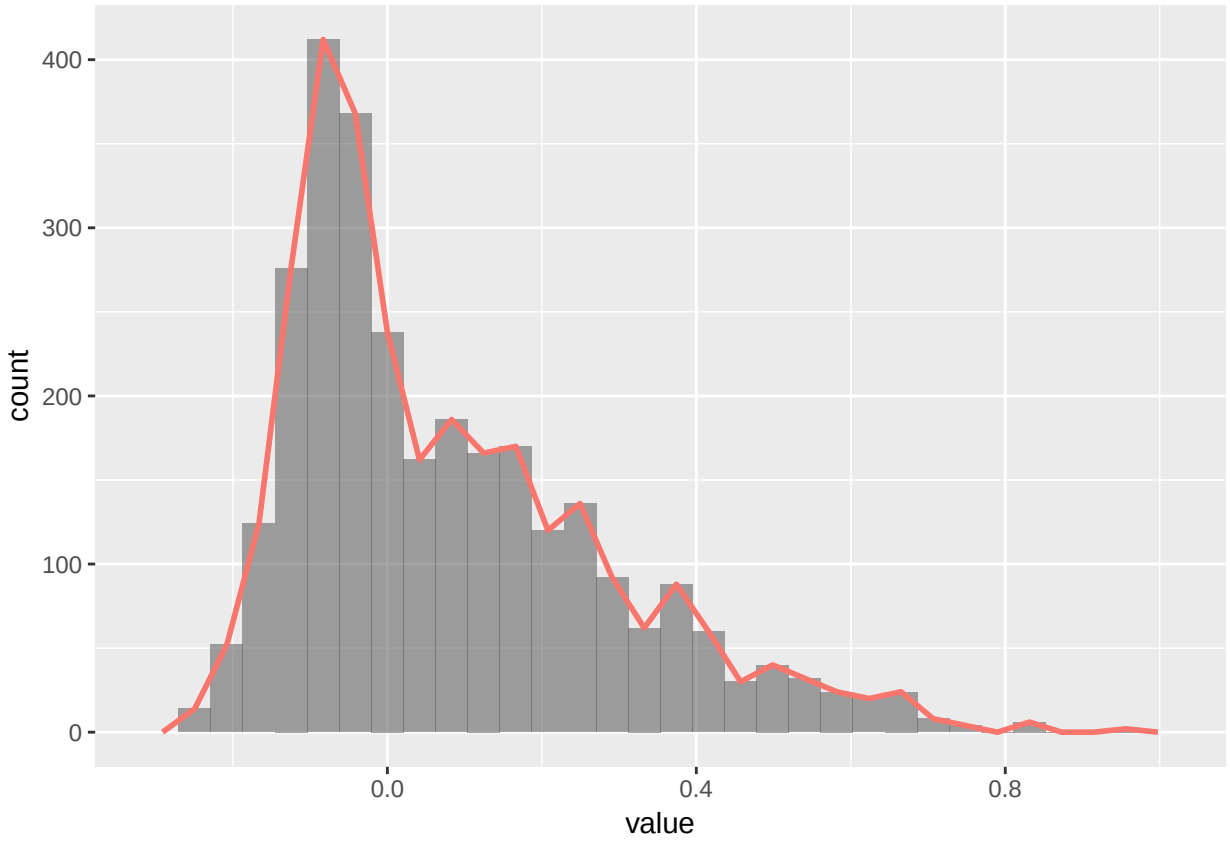
> What is the proportion of similar and dissimilar people?

To answer this question, we use the **Pearson correlation coefficient** to detect similarities. In order to compute such metric, we extract the adjacency matrix from our graph `g` and compute the correlation matrix as:

```
A <- as_adjacency_matrix(g, attr = "weight", names = FALSE) # get adjacency matrix
S <- cor(as.matrix(A))                                      # compute the correlation matrix
diag(S) <- 0                                                # set the diagonal to 0
```

Given `S`, we can plot the distribution of the similarities in the following way:

```
V <- as.vector(S) # flatten S to a vector
ggplot(as.tibble(V), aes(x=value)) +
  geom_histogram(aes(alpha=2/5), bins=30, show.legend=FALSE) +        # histogram
  geom_freqpoly(aes(color="red"), size=1, bins=30, show.legend=FALSE) # frequency polygon
```



where 1 means positive similarity, -1 means negative similarity (or dissimilarity), and 0 means no correlation at all

Note that, the above plot points out a **positive skewness** (or, alternatively called, **right skewness**). To answer the question, we compute the proportion of similar (dissimilar) nodes whose similarity is greater (lower or equal) than 0 as:

```r
cnt <- 0 # initialize a counter

# for each observation
for (i in 1:length(V)) {
  # if the similarity is greater than 0, then we increment the counter
  if (V[i] > 0) cnt <- cnt + 1
}

prop_sim <- cnt/length(V) # compute the proportion of similar nodes
print(prop_sim)           # print such proportion
```

```
## [1] 0.5171468
```

```r
print(1-prop_sim)         # print the 1-complementary proportion (i.e. dissimilar proportion)
```

```
## [1] 0.4828532
```

Not surprisingly, there is a **lot of similarity** between nodes, and this is an expected behavior within this context. Moreover, let's try to give an answer to the following question:

> Which are the most similar people?

To answer this question, we can directly use the computed similarity matrix `S` by generating a weighted (undirected) graph `simil_graph` from it, where the weights are the similarity scores, then compute a `tibble`, named `simil_edges`, where each row corresponds to an edge with its similarity score, with the following commands:

```r
# create the similarity graph from the similarity matrix S
simil_graph <- graph_from_adjacency_matrix(S, mode = "undirected", weighted = TRUE)

# extract the (weighted) edges from such similarity graph
simil_edges <- as.tibble(as_data_frame(simil_graph, what = "edges")) %>%
  mutate(from = as.integer(from), to = as.integer(to)) %>%
  arrange(desc(weight))
```

Finally, we are ready to answer the question. In order to give a more **realistic answer**, we filter only those nodes whose degree is greater or equal than 5, then we print the highest similar nodes:

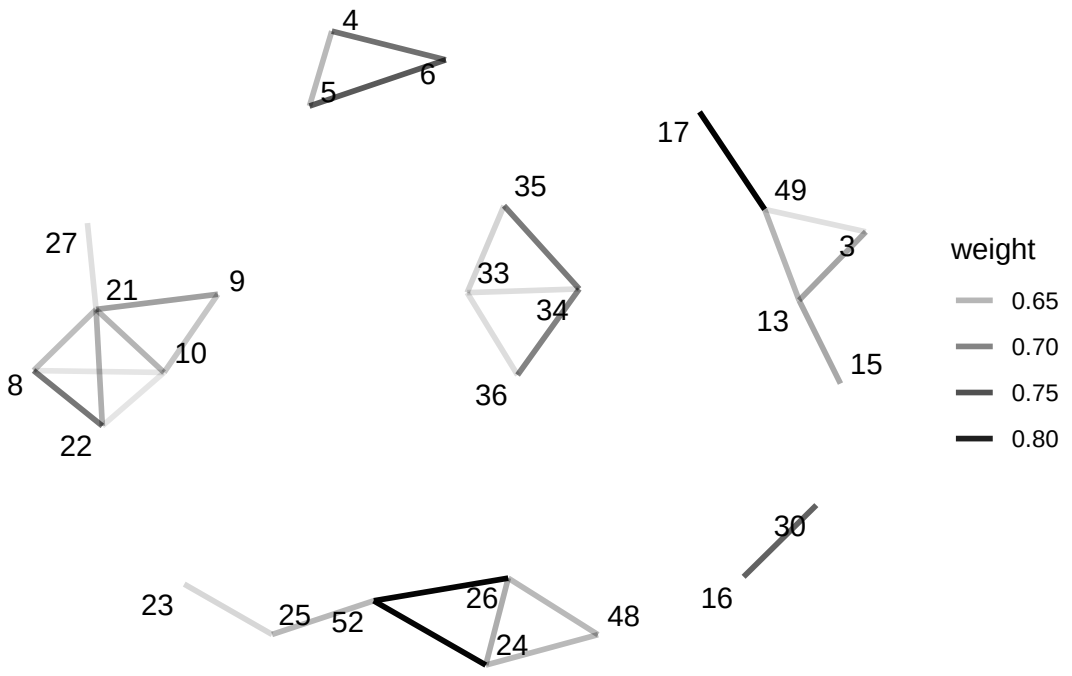
```r
# compute the joint similarities for the people whose degree is greater or equal to 5
joint_simil <- simil_edges %>%
  left_join(nodes, c("from" = "person")) %>%
  left_join(nodes, c("to" = "person")) %>%
  filter(degree.x >= 5, degree.y >= 5) %>%
  select(from, to, weight)
joint_simil # print the most similar nodes
```

```
## # A tibble: 1,127 x 3
##     from    to weight
##    <int> <int>  <dbl>
##  1    17    49  0.830
##  2    24    52  0.827
##  3    26    52  0.827
##  4     5     6  0.741
##  5    16    30  0.733
##  6     4     6  0.718
##  7     8    22  0.713
##  8    34    35  0.709
##  9    34    36  0.702
```

```
## 10     9    21  0.671
## # ... with 1,117 more rows
```

To have an intuitive idea on the similarity in the original network, we can filter out the most similar pairs and then plot the graph generated for such pairs as:

```
# filter those pairs with similarity higher than 0.6
filtered_pairs <- joint_simil %>%
  filter(weight > 0.6)

# create a similarity network with the filtered pairs
similarity_network <- graph_from_data_frame(filtered_pairs, directed=FALSE)

# get the nodes (in order to plot their id)
simil_nodes <- as.tibble(as_data_frame(similarity_network, what = "vertices"))

# plot the result
ggraph(similarity_network, layout="with_kk") +
  geom_edge_link(aes(alpha=weight), edge_width=1) +          # edge link, mapping alpha to weight
  geom_node_text(aes(label=simil_nodes$name), repel=TRUE) + # node text, mapping label to nodes' id (i.
  theme_graph(base_family="sans")
```



Does the similarity between the most similar people changes as a function of their birthplace?

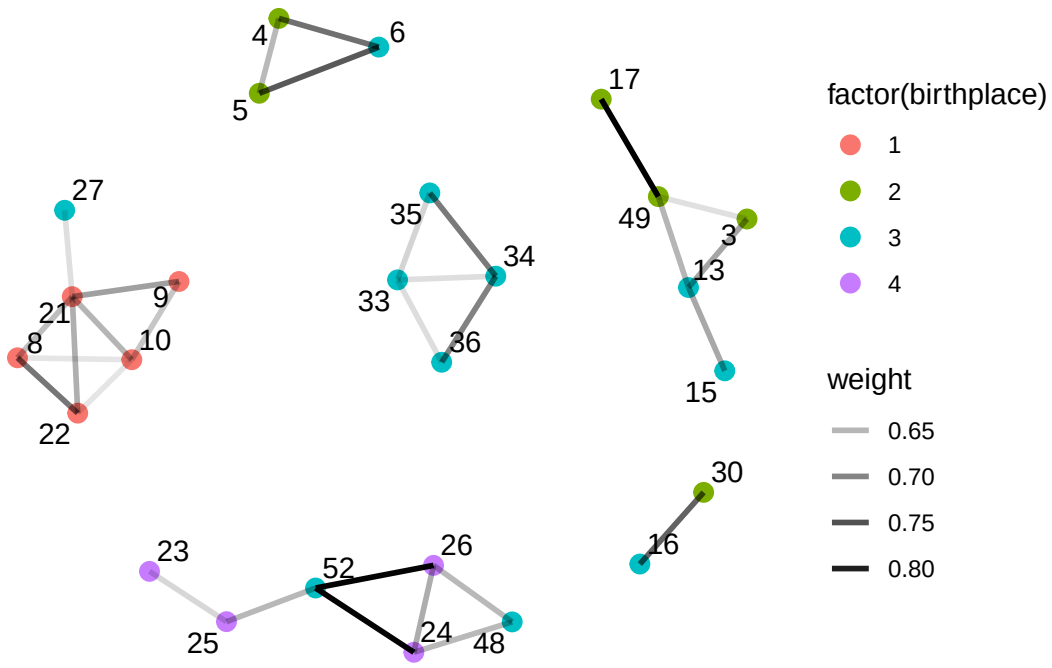
In order to calculate such score, we need to create a `tibble`, named `pairs_with_bp`, that is the join between the `filtered_pairs`, which we have calculated before, and `nodes_bp`, which are the most similar nodes projected only on their `name` and `birthplace`. Given `pairs_with_bp`, we compute the metric of interest (i.e. the proportion of similar people as a function of their birthplace) in the following way:

```
# convert the name of the similar nodes from the similarity network to an integer (i.e. its id)
simil_nodes$name <- as.integer(simil_nodes$name)

# create a tibble of (similar) nodes with their birthplace
simil_nodes_with_bp <- simil_nodes %>%
  left_join(nodes, by=c("name" = "person")) %>%
  select(name, birthplace)

# create a joint tibble of the most similar pairs with their birthplace
# recall that filtered_pairs are those pairs whose similarity is greater than 0.6
pairs_with_bp <- filtered_pairs %>%
  left_join(simil_nodes_with_bp, c("from" = "name")) %>%
  left_join(simil_nodes_with_bp, c("to" = "name"))

# compute the proportion of similar people as a function of their birthplace
cnt <- 0 # initialize a counter

# for each pair
for (i in 1:nrow(pairs_with_bp)) {
  # if their birthplace is the same, then increment the counter
  if(pairs_with_bp$birthplace.x[i] == pairs_with_bp$birthplace.y[i])
    cnt <- cnt + 1
}

# print the proportion of similar nodes as a function of their birthplace
cnt/nrow(pairs_with_bp)
```

```
## [1] 0.6333333
```

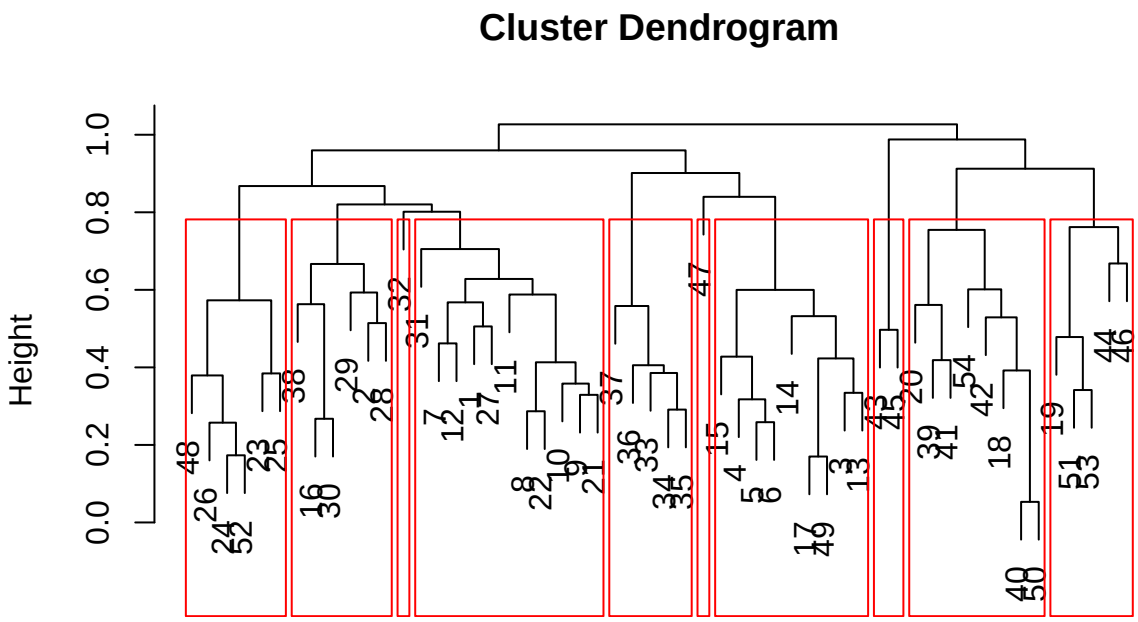
This behaviour can be checked by the following plot:

```
# generate the same similarity network augmented with the nodes' birthplace
similarity_network_with_bp <- graph_from_data_frame(filtered_pairs,
                                                    directed=FALSE,
                                                    vertices=simil_nodes_with_bp)

ggraph(similarity_network_with_bp, layout="with_kk") +
  geom_node_point(aes(color=factor(birthplace)), size=3) + # geom node point, mapping color to birthpla
  geom_edge_link(aes(alpha=weight), edge_width=1) +         # geom edge link, mapping alpha to weight
  geom_node_text(aes(label=simil_nodes$name), repel=TRUE) + # geom node text, mapping label to id
  theme_graph(base_family="sans")
```

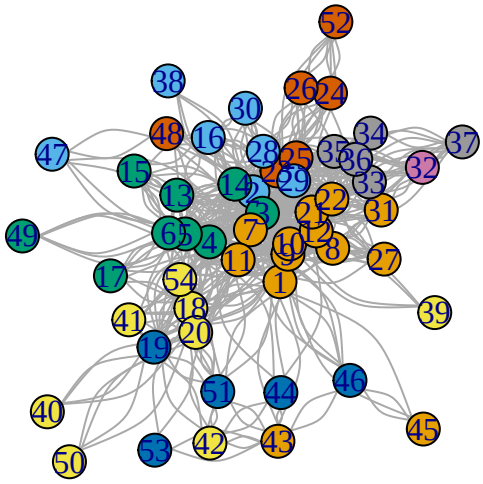


The same behaviour is also empirically proven by the following (bottom-up) **clustering analysis** (an unsupervised learning algorithm that groups people together based on their similarities). The analysis is the following:

```
# distance matrix from similarity matrix
D <- 1-S

# distance object from distance matrix
d <- as.dist(D)

# perform an average-linkage clustering method
cc <- hclust(d, method="average")

# plot the dendrogram of such clustering
plot(cc)

# cut the dendrogram at 10 clusters and color such clusters
clusters.list = rect.hclust(cc, k = 10, border="red")
```

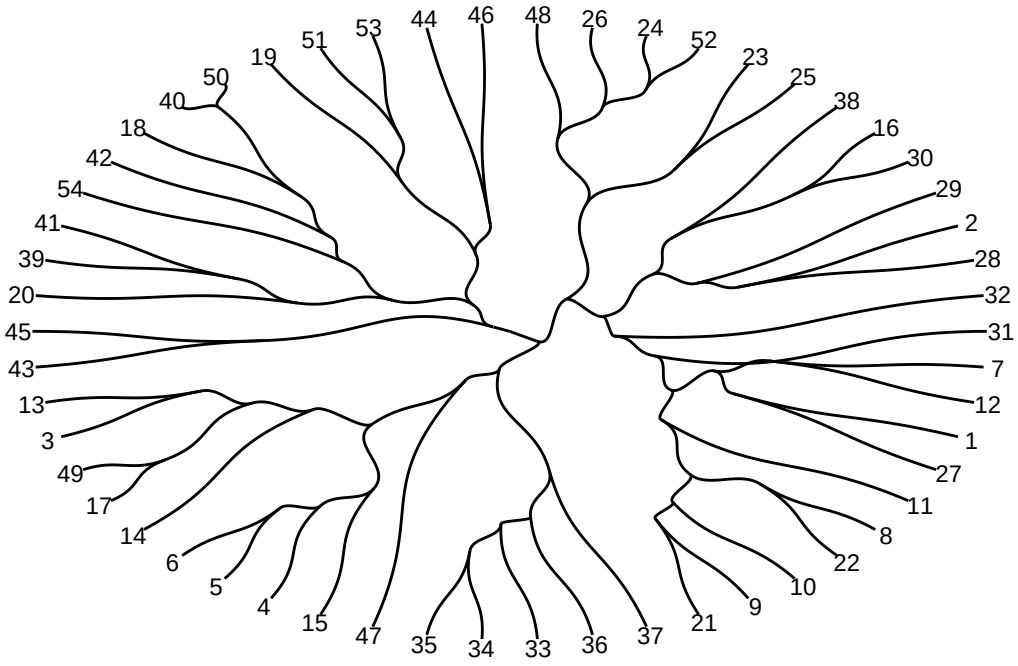



```
# cut dendrogram at 10 clusters
clusters = cutree(cc, k = 10)

# plot the graph with nodes colored based on their cluster membership
plot(g, vertex.color=clusters, layout=layout_with_fr(g))
```



Note that, similar nodes, from the previous analysis, are clustered together. Perhaps a more stilish plot, containing the same information, is the following:

```
# extract dendogram
dendrogram = as.dendrogram(cc)

# plot a circular dendrogram
ggraph(dendrogram, layout = 'dendrogram', circular = TRUE) +
  geom_edge_diagonal() +
  geom_node_text(aes(filter = leaf, label = label, x = x*1.03, y=y*1.03), size = 3) +
  theme_graph(base_family="sans")
```



Now, we would like to answer the **more challenging question**:

> Is it true that even if people know more people tend to commit crimes with those that belong to their own birthplace?

Note that, in the above analysis, we have filtered only those nodes with degree greater or equal to 5. To answer this interesting question, basically we redo the same analysis, but we filter on nodes with degree greater or equal to 15:

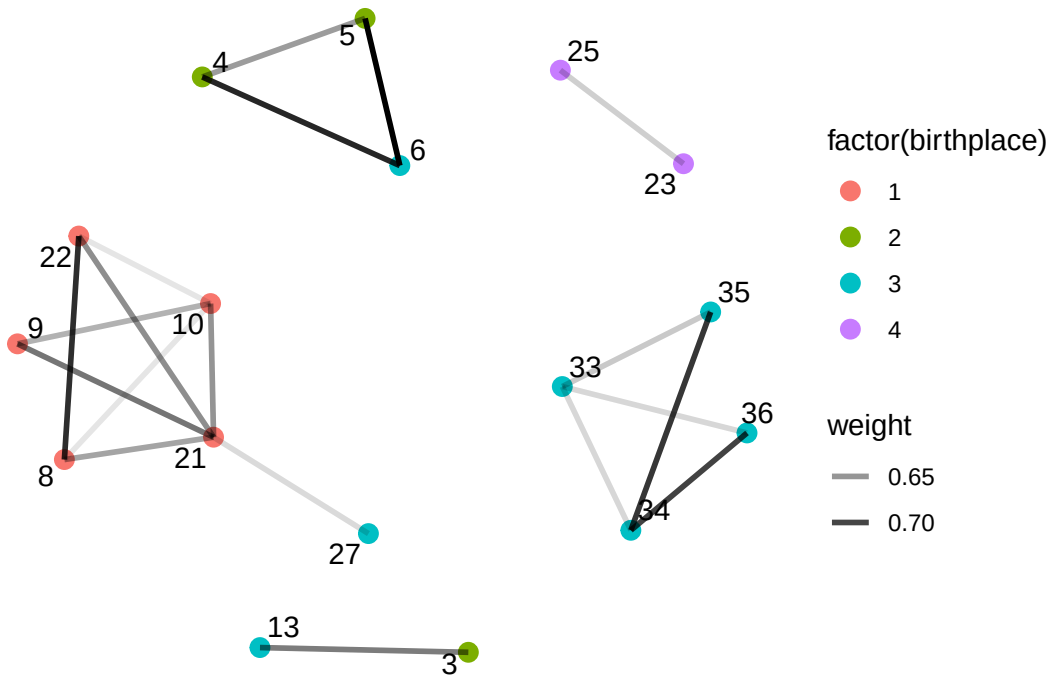
```
# compute the joint similarities
joint_simil2 <- simil_edges %>%
  left_join(nodes, c("from" = "person")) %>%
  left_join(nodes, c("to" = "person")) %>%
  filter(degree.x >= 15, degree.y >= 15) %>%
  select(from, to, weight)

# filter those pairs with similarity higher than 0.6
filtered_pairs2 <- joint_simil2 %>%
  filter(weight > 0.6)

# create a similarity network with the filtered pairs
similarity_network2 <- graph_from_data_frame(filtered_pairs2, directed=FALSE)

# get the nodes
simil_nodes2 <- as.tibble(as_data_frame(similarity_network2, what = "vertices"))

# convert the name to an integer (i.e. its id)
simil_nodes2$name <- as.integer(simil_nodes2$name)
```

```r
simil_nodes_with_bp2 <- simil_nodes2 %>%
  left_join(nodes, by=c("name" = "person")) %>%
  select(name, birthplace)

# create a joint tibble of the most similar pairs with their birthplace
pairs_with_bp2 <- filtered_pairs2 %>%
  left_join(simil_nodes_with_bp2, c("from" = "name")) %>%
  left_join(simil_nodes_with_bp2, c("to" = "name"))

# compute the proportion of similar people as a function of their birthplace
cnt2 <- 0
for (i in 1:nrow(pairs_with_bp2)) {
  if(pairs_with_bp2$birthplace.x[i] == pairs_with_bp2$birthplace.y[i])
    cnt2 <- cnt2 + 1
}
# print such proportion
cnt2/nrow(pairs_with_bp2)
```

```
## [1] 0.7894737
```

Again, the behaviour can be checked by the following plot:

```r
similarity_network_with_bp2 <- graph_from_data_frame(filtered_pairs2,
                                                     directed=FALSE,
                                                     vertices=simil_nodes_with_bp2)
ggraph(similarity_network_with_bp2, layout="with_kk") +
  geom_node_point(aes(color=factor(birthplace)), size=3) +
  geom_edge_link(aes(alpha=weight), edge_width=1) +
  geom_node_text(aes(label=simil_nodes2$name), repel=TRUE) +
  theme_graph(base_family="sans")
```



Hence, we claim that it is true that **people that have higher acquintances tend to perfom crimes together with those that are ethnically similar to them**.

Power analysis

We are interested in comparing the centrality of people against their (PageRank) power. In particular, we would like to identify people that are central and powerful. In order to perform such analysis, we define the following function (powered by Bozzo, Enrico and Franceschet, Massimo) that computes the power of people:

```
# Compute power x = (1/x) A
#INPUT
# A = graph adjacency matrix
# t = precision
# OUTPUT
# A list with:
# vector = power vector
# iter = number of iterations

power = function(A, t) {
  n = dim(A)[1];
  # x_2k
  x0 = rep(0, n);
  # x_2k+1
  x1 = rep(1, n);
  # x_2k+2
```

```
  x2 = rep(1, n);
  diff = 1
  eps = 1/10^t;
  iter = 0;
  while (diff > eps) {
    x0 = x1;
    x1 = x2;
    x2 = (1/x2) %*% A;
    diff = sum(abs(x2 - x0));
    iter = iter + 1;
  }
  # it holds now: alpha x2 = (1/x2) A
  alpha = ((1/x2) %*% A[,1]) / x2[1];
  # hence sqrt(alpha) * x2 = (1/(sqrt(alpha) * x2)) A
  x2 = sqrt(alpha) %*% x2;
  return(list(vector = as.vector(x2), iter = iter))
}
```

Such function computes the power of the people with an interative fashion. Therefore, we compute such powers as:

```
damping = 0.15                  # damping factor
I <- diag(damping, vcount(g)) # identity matrix (with the damping factor on its diagonal)
Ad <- A + I                     # A + I
p <- power(Ad, 6)$vector        # compute power
V(g)$power <- p                 # assign power to the nodes

nodes <- nodes %>%              # mutate the tibble nodes adding the new attribute power
  mutate(power = p)

nodes %>%                       # sort the nodes in decreasing order by their pagerank centrality
  select(person, pr) %>%
  arrange(desc(pr))
```

```
## # A tibble: 54 x 2
##    person     pr
##     <int>  <dbl>
##  1     14 0.0308
##  2     23 0.0280
##  3     10 0.0266
##  4      1 0.0263
##  5      4 0.0262
##  6      3 0.0260
##  7     19 0.0259
##  8     25 0.0258
##  9      5 0.0253
## 10      2 0.0250
## # ... with 44 more rows
```

```
nodes %>%                       # sort the nodes in decreasing order by their power score
  select(person, power) %>%
  arrange(desc(power))
```

```
## # A tibble: 54 x 2
##    person power
```

```
##     <int> <dbl>
##  1     14  23.8
##  2     19  19.1
##  3     10  17.9
##  4      4  15.8
##  5     20  15.6
##  6      2  14.8
##  7      5  13.3
##  8      9  12.9
##  9     23  12.9
## 10      3  12.6
## # ... with 44 more rows
```

We can also compute the correlations between such scores as:

```
cor(nodes$pr, nodes$power)                    # Pearson correlation coef.
```

```
## [1] 0.9630992
```

```
cor(nodes$pr, nodes$power, method="spearman") # Spearman rank correlation coef.
```

```
## [1] 0.9777202
```

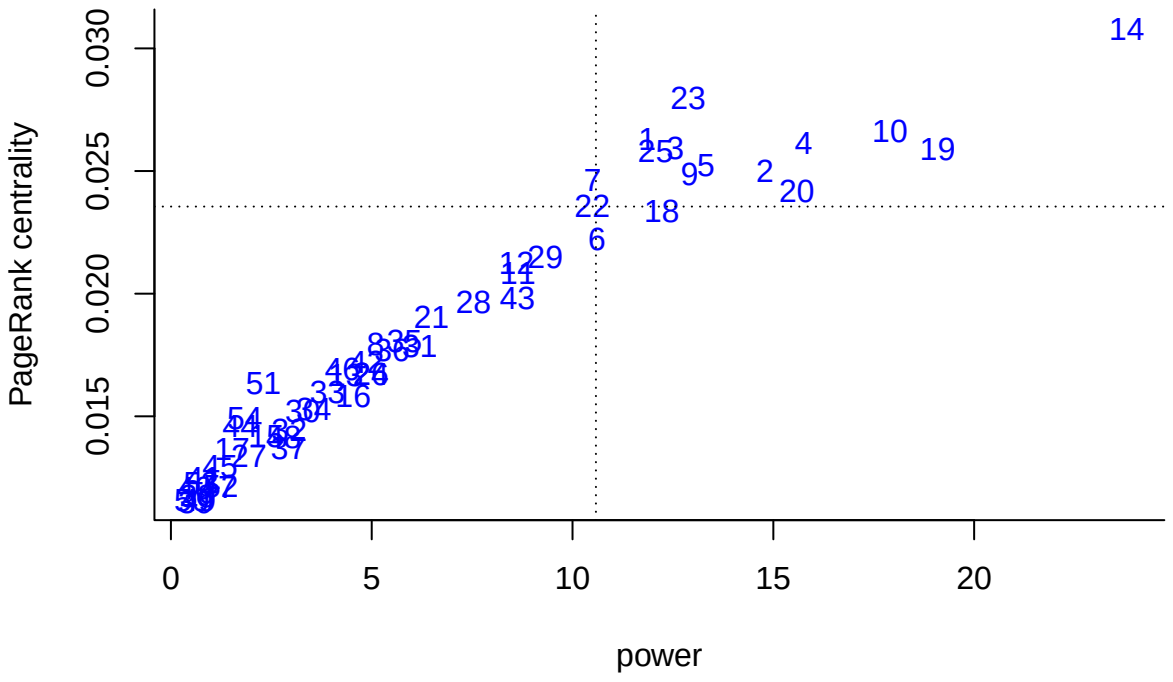
```
cor(nodes$pr, nodes$power, method="kendall")  # Kendall tau correlation coef.
```

```
## [1] 0.8850053
```

Who are the most central and powerful people?

To answer this question, we plot such people and create a `tibble`, named `power_central_nodes`, as:

```
mc <- quantile(nodes$pr, 0.75)    # compute the 0.75 pagerank centrality quantile
mp <- quantile(nodes$power, 0.75) # compute 0.75 power measure quantile

# scatterplot : pagerank centrality vs power
plot(nodes$power, nodes$pr, xlab="power", ylab="PageRank centrality", pch=NA, bty="l")
text(nodes$power, nodes$pr, labels=nodes$person, adj=c(0.5,0.5), cex=1, col = "blue")

abline(h=mc, lty=3)               # add horizontal (dotted) line
abline(v=mp, lty=3)               # add vertical (dotted) line
```




```
power_central_nodes <- nodes %>% # compute the most powerful and central people
  filter(power >= mp & pr >= mc) %>%
  select(person, age, birthplace, pr, power)

power_central_nodes              # print such people
```

```
## # A tibble: 12 x 5
##    person   age birthplace     pr power
##     <int> <int> <fct>       <dbl> <dbl>
##  1      1    20 1          0.0263  11.9
##  2      2    20 2          0.0250  14.8
##  3      3    19 2          0.0260  12.6
##  4      4    21 2          0.0262  15.8
##  5      5    24 2          0.0253  13.3
##  6      9    20 1          0.0249  12.9
##  7     10    23 1          0.0266  17.9
##  8     14    19 3          0.0308  23.8
##  9     19    20 2          0.0259  19.1
## 10     20    20 1          0.0242  15.6
## 11     23    19 4          0.0280  12.9
## 12     25    17 4          0.0258  12.1
```

Based on their similarities, power and (PageRank) centrality, what is the proportion of people that have a parental relationship?

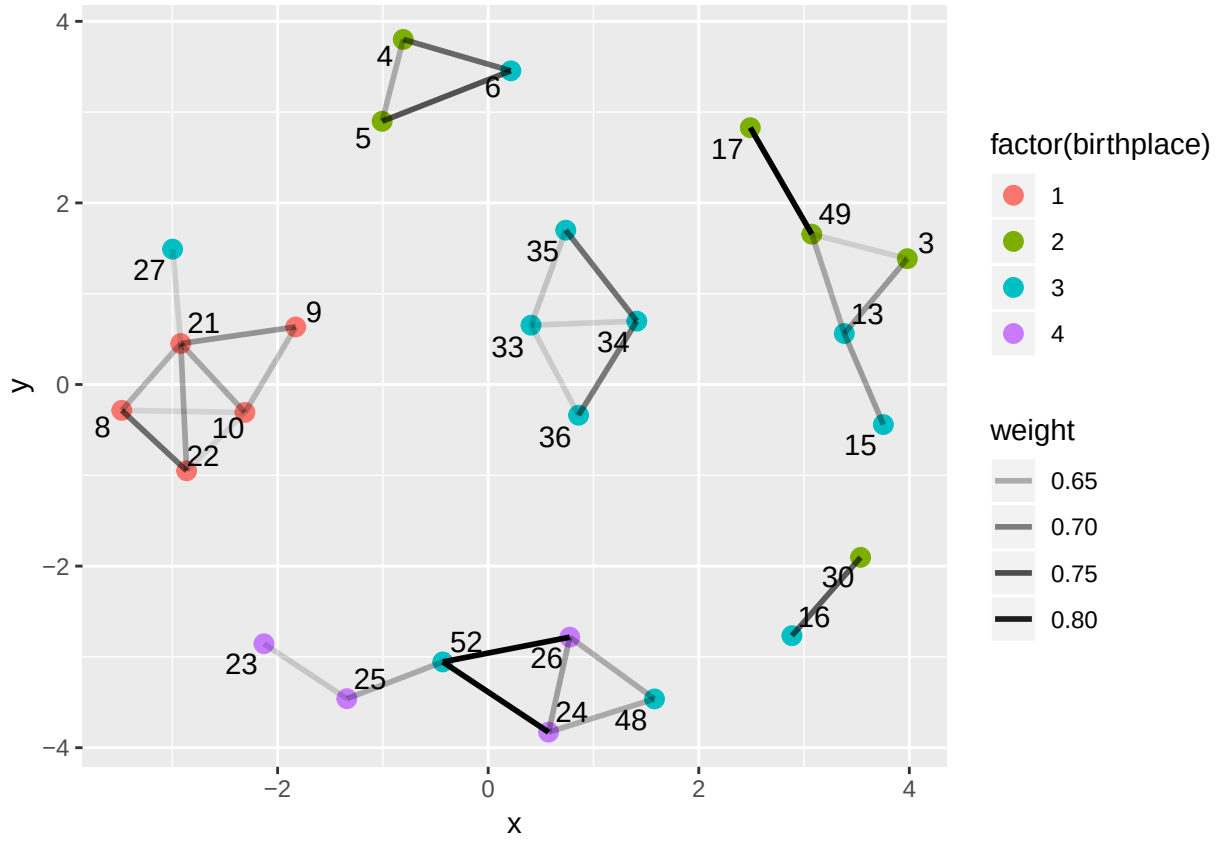
To get the most powerful and central nodes, based on the similarity between them, we can compute them easily as:

```
simil_power_central_nodes <- power_central_nodes %>% # compute similar, powerful and central nodes
  inner_join(simil_nodes, c("person" = "name"))

simil_power_central_nodes                            # print such people
```

```
## # A tibble: 7 x 5
##   person   age birthplace     pr power
##    <int> <int> <fct>       <dbl> <dbl>
## 1      3    19 2          0.0260  12.6
## 2      4    21 2          0.0262  15.8
## 3      5    24 2          0.0253  13.3
## 4      9    20 1          0.0249  12.9
## 5     10    23 1          0.0266  17.9
## 6     23    19 4          0.0280  12.9
## 7     25    17 4          0.0258  12.1
```

Recall our similarity network graph, where we have restricted ourselves on the people who have at least 5 acquintances:

```
ggraph(similarity_network_with_bp, layout="with_kk") +
  geom_node_point(aes(color=factor(birthplace)), size=3) +
  geom_edge_link(aes(alpha=weight), edge_width=1) +
  geom_node_text(aes(label=simil_nodes$name), repel=TRUE)
```



Note that, given the graph and the `simil_power_central_nodes` structure, the person 3 is isolated from the others, therefore we can exclude such person from the analysis. To answer our question, we can do the following:

```
# remove the node 3
simil_power_central_nodes <- simil_power_central_nodes %>%
  filter(person != 3)

simil_power_central_nodes # print the most similar, powerful and central nodes (without 3)
```

```
## # A tibble: 6 x 5
##   person   age birthplace     pr power
##    <int> <int> <fct>       <dbl> <dbl>
## 1      4    21 2          0.0262  15.8
## 2      5    24 2          0.0253  13.3
## 3      9    20 1          0.0249  12.9
## 4     10    23 1          0.0266  17.9
## 5     23    19 4          0.0280  12.9
## 6     25    17 4          0.0258  12.1
```

```
# compute the edges between such nodes
# NOTE: since the graph is undirected we have only a pair for each node
ee <- filtered_pairs %>%
  inner_join(simil_power_central_nodes, c("from" = "person")) %>%
  inner_join(simil_power_central_nodes, c("to" = "person"))

ee_f <- as.vector(ee$from) # flatten "from" to a vector
ee_t <- as.vector(ee$to)   # flatten "to" to a vector

# get the original weight for such pairs
# NOTE: in the similarity graph that we have computed before
#       the weights of the edges are the similarity scores between nodes
filtered_pairs %>%
  inner_join(ties, c("from" = "from", "to" = "to")) %>%
  filter(from %in% ee_f, to %in% ee_t) %>%
  select(from, to, weight.y) %>%
  rename(weight = weight.y)
```

```
## # A tibble: 3 x 3
##    from    to weight
##   <int> <int>  <int>
## 1     4     5      4
## 2     9    10      4
## 3    23    25      4
```

As you can see, not surprisingly, **people that are similar, powerful and central** (w.r.t the measures computed before) **are more likely to stick with their own family**, given in terms as the weight on the relation between such nodes (i.e. 4 in this case); this answers our question, that is, we have a 100% proportion of such subjects.

Model

Suppose now that we would like to make predictions on individuals. For instance, we could try to classify people based on the fact that they have been arrested or not (i.e. `prison=1` or `prison=0`). Therefore, we can formalize a **classification problem**. Briefly, the classification setting lies within the statistical learning framework, where for each observation (the people, in our case) has a label (i.e. the fact that they have been arrested or not, in our aforementioned example). Moreover, since we have labels for each observation, such

problem is said to be a **supervised learning problem**.

Hereby, we briefly analyze a fitted **generalized linear model**:

```
# create a data frame
d <- as.data.frame(people)

# convert the Prison attribute to a factor
d$Prison <- as.factor(d$Prison)

# fit a generalized linear model
mod1 <- glm(Prison~Age+Birthplace+Arrests+Convictions+Ranking+Music+Residence,
           d, family=binomial)

# analyze the fitted model
summary(mod1)
```

```
##
## Call:
## glm(formula = Prison ~ Age + Birthplace + Arrests + Convictions +
##     Ranking + Music + Residence, family = binomial, data = d)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5366  -0.7919  -0.4358   0.6970   1.8948
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   7.7251     4.7851   1.614   0.1064
## Age          -0.2813     0.1701  -1.654   0.0982 .
## Birthplace    0.1102     0.3691   0.299   0.7653
## Arrests      -0.1401     0.1532  -0.915   0.3603
## Convictions   0.6551     0.3021   2.169   0.0301 *
## Ranking      -1.1532     0.5967  -1.933   0.0533 .
## Music        -0.7485     0.9789  -0.765   0.4445
## Residence     0.2890     0.7230   0.400   0.6894
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 74.192  on 53  degrees of freedom
## Residual deviance: 53.591  on 46  degrees of freedom
## AIC: 69.591
##
## Number of Fisher Scoring iterations: 4
```

Note that, we have a lot of p-values for the z-test statistics that are very high (i.e. greater than 0.05), this means that not all regressors, namely the attributes, are statistically significant. In Statistics, an approach to fit a better model is to remove, one at the time, those regressors with highest p-values for the test statistic, and then to refit a (new) model with the remaining ones. We skip such (headache) analysis, and we have decided to present the last model:

```
# fit a better model
mod2 <- glm(Prison~Arrests, d, family=binomial)
```

```r
# analyze the fitted model
summary(mod2)
```

```
##
## Call:
## glm(formula = Prison ~ Arrests, family = binomial, data = d)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0637  -0.8818  -0.5598   0.9085   1.8899
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.94580    0.63235  -3.077  0.00209 **
## Arrests      0.17168    0.05533   3.103  0.00192 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 74.192  on 53  degrees of freedom
## Residual deviance: 61.741  on 52  degrees of freedom
## AIC: 65.741
##
## Number of Fisher Scoring iterations: 4
```

This model is better since the AIC score is lower. Moreover, we can visually check this behaviour by fitting a **decision tree**, using the `rminer` package, with the following commands:

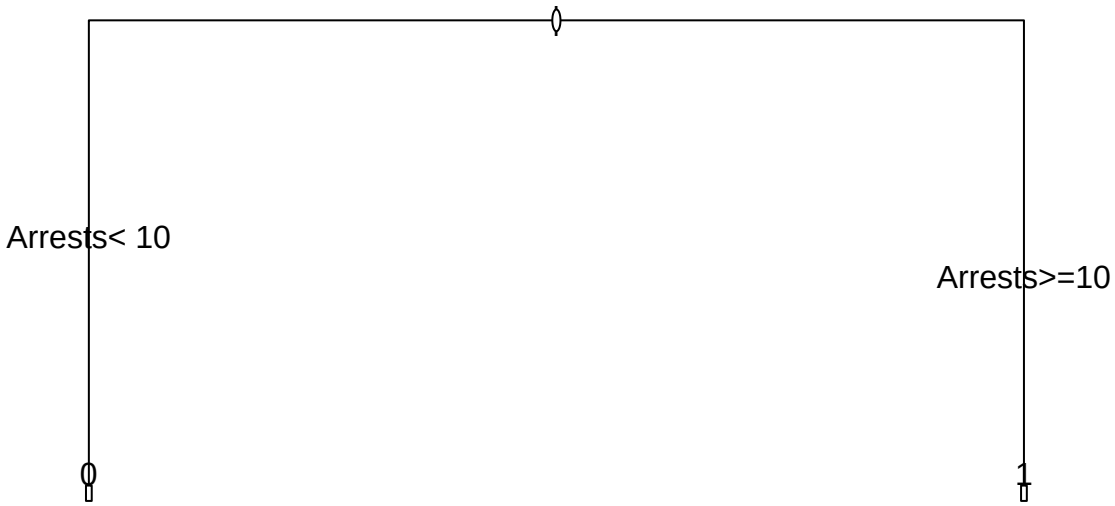
```r
# load rminer package
library(rminer)
```

```
## Warning: package 'rminer' was built under R version 3.5.2
```

```r
# fit a decision tree with all regressors (i.e. attributes)
M <- fit(Prison~Age+Birthplace+Arrests+Convictions+Ranking+Music+Residence,
         data=d,
         model="rpart")

# print the model
print(M@object)
```

```
## n= 54
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 54 24 0 (0.5555556 0.4444444)
##   2) Arrests< 10 30  7 0 (0.7666667 0.2333333) *
##   3) Arrests>=10 24  7 1 (0.2916667 0.7083333) *
```

```r
# plot the (compressed) model
plot(M@object,uniform=T,compress=T)

# put "fancy" text on the model (the nodes) with additional options for a better view
text(M@object,fancy=T,xpd=T,fwidth=0.1,fheight=0.2)
```



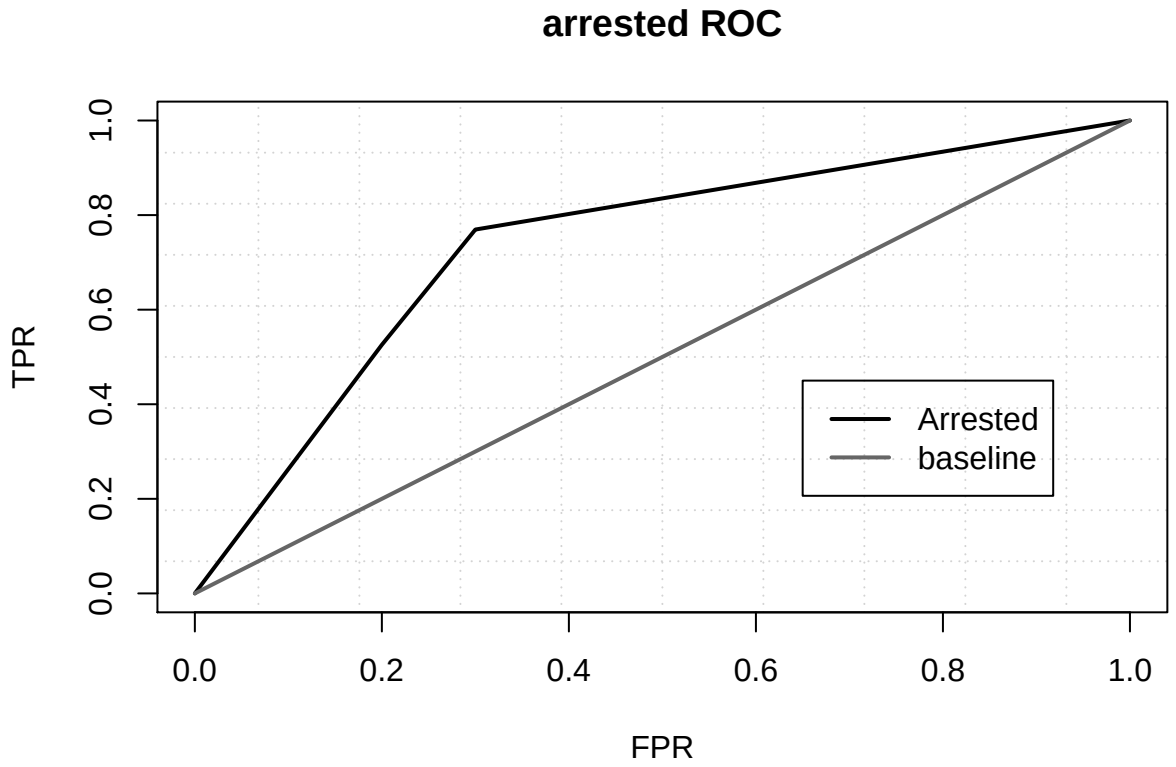

```
# predict on the training set (i.e. the same data)
P <- predict(M,d)

# confusion matrix
print(mmetric(d$Prison,P,"CONF")$conf)
```

```
##       pred
## target  0  1
##      0 23  7
##      1  7 17
```

```
# all performances (accuracy ACC, classification error CE, true positive rate TPR, etc.)
print(mmetric(d$Prison,P,"ALL"))
```

```
##         ACC          CE         BER       KAPPA     CRAMERV    ACCLASS1
##  74.0740741  25.9259259  26.2500000  47.5000000   0.4375000  74.0740741
##    ACCLASS2        TPR1        TPR2        TNR1        TNR2  PRECISION1
##  74.0740741  76.6666667  70.8333333  70.8333333  76.6666667  76.6666667
##  PRECISION2         F11         F12        MCC1        MCC2       BRIER
##  70.8333333  76.6666667  70.8333333   0.6111111   0.6111111   0.1912037
## BRIERCLASS1 BRIERCLASS2         AUC   AUCCLASS1   AUCCLASS2        NAUC
##   0.1912037   0.1912037   0.7375000   0.7375000   0.7375000   0.7375000
##    TPRATFPR       ALIFT      NALIFT ALIFTATPERC
##   1.0000000   0.6334877   0.6334877   1.0000000
```

```
# ROC curve analysis for target class (TC) 1, i.e. arrested
mgraph(d$Prison,P,graph="ROC",TC=1,main="arrested ROC",
       baseline=TRUE,leg="Arrested",Grid=10)
```



Conclusions and discussion

In this work, we have analyzed a gang network, concerning black people, who performed crimes in London, during a police observation-time from 2005 to 2009. We have downloaded the (anonymised) data, then we have built a graph from it. Each self-respecting statistical analysis has to perform a preliminary Explanatory Data Analysis (EDA), which we have done briefly.

We then moved on with the centrality analysis of the network and have pointed out some (preliminary) interesting facts. For instance, the network is very well tied, that is, people have hanged-out together and more than 40% have commited crimes together, either if they belong to the same family or not.

In our opinion, the most interesting part of the presented analysis is relative to the part of the connection patterns between nodes. The criminality analysis was driven by the birthplace, and very important empirical proofs have outcomed. For instance, very connected people tend to act as criminals with their own people, that is, with those that belong to their own birthplace. Moreover, strongly related people that are powerful and central also tend to commit crimes along with their families, and this is very natural, if we think, for example, at the Italian Mafia.

For the sake of completeness, we have presented a short introduction on how one could fit a generalized linear model in order to classify instances. The scope of this work was not centered on such type of analysis, although one could investigate more on such aspects. For instance, one could fit a linear model in order to model a regression problem, or could try to use `rminer`'s arsenal of models in order to dig deeper into the model playground.

We have achieved our goals proving that network science analysis is a powerful tool to investigate on the peculiarities of networks. For example, based on our case-study, the presented analysis could be explored by the police in order to catch the bad guys.