Lab 2 - Social Network Visualization

Load Needed Packages

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

If the package had not been installed you will need to install it with `install.packages()` in the console and then load it with `require()` or `library()`. For this lab we will be using `readxl`, `igraph`, and `RColorBrewer`. You can get more information about them by in the console running the name of the package with a "?" in front of it. For example try `?readxl` in the console. This also works for functions like `library()`, just type `?library`.

You can see all the packages that are installed in the "Packages" tab in the bottom right panel. You can also update packages that way too.

Reading Excel Data in R

Now we will input our nodes and edges using the `read_excel()` function. First lets look at its help file. Run `?read_excel`.

This will show you a description, the default *usage*, the *argument* the function takes and what they mean, what *value* it outputs, and some examples.

The most important of these is the *arguments*, which are what goes in the "()" after the function name. For `read_excel()` they are path, sheet, range, col_names, etc. You will see under the *usage* header you can see if there are default values already assigned to the function. The two that we are concerned with are "path" and "sheet".

Arguments

- Path = the path of the file. This can be a full path to a files location on your computer, or if that file is in your *working directory* (`getwd()`) it can be just the file name . Make sure to put the filename in "quotation marks".
- Sheet = the name of the sheet to read, either by it's name, or its position represented by an integer. If it is left empty or `NULL` then it will only read the first sheet.

NOTE: R is ALWAYS case-sensitive, nodes and Nodes are not interchangeable.

Lets look at our "Nodes" sheet in the "fhact50data.xlsx" file.

```
read_excel(path = "fhact50data.xlsx", sheet = "Nodes")
```

```
## # A tibble: 38 x 3
##    name                          type       title
##    <chr>                         <chr>      <chr>
##  1 Franklinton Senior Housing 1 Building   Franklinton Senior Housing 1
##  2 National Church Residents    Non-profit National Church Residents
##  3 Finance Fund                 Financial  Finance Fund
##  4 OHFA                         Financial  Ohio Housing Finace Agency
##  5 Community Development Collab Government Community Development Collab
##  6 OCCH                         Financial  Ohio Capital Corporation for Housing
##  7 FDA                          Non-profit Franklinton Development Association
##  8 FAC                          Government Franklinton Area Commission
##  9 OCFC                         Financial  Ohio Capital Finacial Corporation
## 10 OCIC                         Financial  Ohio Capital Impact Corportation
## # ... with 28 more rows
```

A tibble!

This is a *data* type that R uses to visualize tables and dataframes. It has the table dimensions, column names, and column data types.

Create a Dataframe Object

R uses "Object Oriented Programming" which mean it stores all data in objects. This is simply assigns a name to a generic container that stores whatever you put in it. You do this by using the `<-` arrow. When writing code you can just hit "alt + minus".

The pattern goes `objectName <- value`

It is important to think about how you name things in R. This is the issue that causes the most problems when getting started. Make sure to use simple names and also a consistent style. You can use letters and underscores. I normally use no underscore and capitalize the first letter in every work past the first. ex. nodeData or thisIsAObjectName, etc. but you could use Node_Data, or This_is_a_object_name. I find that underscores are cumbersome and mostly unnecessary. This help too with using the auto-fill feature in RStudio.

Let's save our node data and edge data using `read_excel()`.

```
nodeData <- read_excel(path = "fhact50data.xlsx", sheet = "Nodes")
edgeData <- read_excel(path = "fhact50data.xlsx", sheet = "Edges")
```

HINT: When typing in the arguments for a function you can hit TAB and it will bring up a list of all arguments for that function.

Looking at the Data

Now within our "environment" when have two objects, `nodeData` and `edgeData`. You will see them in the "Environment" tab in the top right panel of RStudio. We can look at them a number of ways. If we just type in the name, it "calls" the object.

```
edgeData
```

```
## # A tibble: 65 x 7
##    from              to                type   title           label value arrows
##    <chr>             <chr>             <chr>  <chr>           <lgl> <lgl> <lgl>
##  1 National Church ~ Franklinton Seni~ Physi~ Builder/Develo~ NA    NA    NA
##  2 Finance Fund      National Church ~ Finan~ Investment Cap~ NA    NA    NA
##  3 OHFA              Finance Fund      Finan~ Leverage Priva~ NA    NA    NA
##  4 OHFA              FDA               Finan~ Funding/Projec~ NA    NA    NA
##  5 Neighborhood Des~ Finance Fund      Polit~ Design and Pla~ NA    NA    NA
##  6 OCCH              Community Develo~ Finan~ Investment Cap~ NA    NA    NA
##  7 Community Develo~ FDA               Finan~ Investment Cap~ NA    NA    NA
##  8 FAC               FDA               Social Member Partner~ NA    NA    NA
##  9 FDA               FAC               Social Member Partner~ NA    NA    NA
## 10 FAC               FHAct50 Committee Social Member Partner~ NA    NA    NA
## # ... with 55 more rows
```

We can also look at what type of object it is with `class()`

```
class(edgeData)
```

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

A tibble is a "table dataframe", a "dataframe", and a "dataframe". Meaning any function that takes any of these types of objects will accept the object.

Looking at the Specifics of the Data

We can also look at specific column of the dataframe with the `$` operator. This calls a named element of the data. For example lets look at the `type` column of the edge data.

```
edgeData$type
```

```
##  [1] "Physical"  "Financial" "Financial" "Financial" "Political" "Financial"
##  [7] "Financial" "Social"    "Social"    "Social"    "Social"    "Political"
## [13] "Social"    "Political" "Social"    "Social"    "Social"    "Social"
## [19] "Political" "Financial" "Political" "Financial" "Financial" "Financial"
## [25] "Financial" "Financial" "Financial" "Physical"  "Financial" "Financial"
## [31] "Social"    "Financial" "Physical"  "Financial" "Social"    "Financial"
## [37] "Physical"  "Political" "Political" "Social"    "Social"    "Political"
## [43] "Political" "Physical"  "Physical"  "Political" "Physical"  "Physical"
## [49] "Political" "Financial" "Financial" "Financial" "Political" "Financial"
## [55] "Financial" "Physical"  "Social"    "Physical"  "Physical"  "Physical"
## [61] "Political" "Physical"  "Physical"  "Financial" "Political"
```

You can also select a specific entry based on it's position in a list. For example if I want the type of the 4th edge. This will be more useful later.

```
edgeData$type[4]
```

```
## [1] "Financial"
```

There are some other standard function that are useful.

```r
unique(edgeData$type)
```

```
## [1] "Physical"  "Financial" "Political" "Social"
```

or...

```r
length(unique(edgeData$type))
```

```
## [1] 4
```

Making a Network Object from Data Frames

`nodeData` is a node list and `edgeData` is a edge list. We can combine these to make a graph using `graph_from_data_frame()`, a function the `igraph` package. It takes 3 arguments: d which is your edge list, directed which is a logical scalar (`TRUE` or `FALSE`) to say if it is a directed graph, and vertices which is your node list. These are not in quotes because they are a objects.

```
netData <- graph_from_data_frame(d = edgeData,
                                 vertices = nodeData,
                                 directed = TRUE)
class(netData)
```

```
## [1] "igraph"
```

NOTE: This may be a point that you get a Error. The most common mistake is that there are a issues with the spelling or names of your nodes and edges. Make sure that every time a node is named it is spelled the same, capitalization too.

Looking at an Igraph Object

An igraph object is a specific type of object. Let take a look at it.

```
netData
```

```
## IGRAPH fe6df93 DN-B 38 65 --
## + attr: name (v/c), type (v/c), title (v/c), type (e/c), title (e/c),
## | label (e/l), value (e/l), arrows (e/l)
## + edges from fe6df93 (vertex names):
## [1] National Church Residents   ->Franklinton Senior Housing 1
## [2] Finance Fund                ->National Church Residents
## [3] OHFA                        ->Finance Fund
## [4] OHFA                        ->FDA
## [5] Neighborhood Design Center  ->Finance Fund
## [6] OCCH                        ->Community Development Collab
## [7] Community Development Collab->FDA
## + ... omitted several edges
```

Igraph has combined the two dataframes into one network object including the node and edge attributes. Igraph has two functions that let us look at either the "vertices" or the "edges", `V()` and `E()`.

```
V(netData)
```

```
## + 38/38 vertices, named, from fe6df93:
##  [1] Franklinton Senior Housing 1 National Church Residents
##  [3] Finance Fund                 OHFA
##  [5] Community Development Collab OCCH
##  [7] FDA                          FAC
##  [9] OCFC                         OCIC
## [11] CPO                          Cols Dept of Dev
## [13] US HUD HOME Fund             FHAct50 Committee
## [15] Neighborhood Design Center   Kaufman Developer
## [17] Homeport                     McDowell Place
## [19] Warner Junction              Model Group
## + ... omitted several vertices
```

```
E(netData)
```

```
## + 65/65 edges from fe6df93 (vertex names):
##  [1] National Church Residents   ->Franklinton Senior Housing 1
##  [2] Finance Fund                ->National Church Residents
##  [3] OHFA                        ->Finance Fund
##  [4] OHFA                        ->FDA
##  [5] Neighborhood Design Center  ->Finance Fund
##  [6] OCCH                        ->Community Development Collab
##  [7] Community Development Collab->FDA
##  [8] FAC                         ->FDA
##  [9] FDA                         ->FAC
## [10] FAC                         ->FHAct50 Committee
## + ... omitted several edges
```

We can see the different node attributes with `vertex_attr()` and the edge attributes with `edge_attr()`. More of this in a sec.

Changing Node Data

Let's look at the node attributes.

```
vertex_attr(netData)
```

```
## $name
##  [1] "Franklinton Senior Housing 1" "National Church Residents"
##  [3] "Finance Fund"                 "OHFA"
##  [5] "Community Development Collab" "OCCH"
##  [7] "FDA"                          "FAC"
##  [9] "OCFC"                         "OCIC"
## [11] "CPO"                          "Cols Dept of Dev"
## [13] "US HUD HOME Fund"             "FHAct50 Committee"
## [15] "Neighborhood Design Center"   "Kaufman Developer"
## [17] "Homeport"                     "McDowell Place"
## [19] "Warner Junction"              "Model Group"
## [21] "THRIVE"                       "Mount Carmel"
## [23] "Homes on Hartford"            "COCIC"
## [25] "COCLT"                        "Town Square Station"
## [27] "City View Homes"              "Boulevard Homes"
## [29] "US HUD CDBG Fund"             "Affordable Housing Trust"
## [31] "Gravity 1"                    "Gravity 2"
## [33] "River and Rich"               "Casto"
## [35] "CMHA"                         "400 W Rich"
## [37] "Urban Smart Growth"           "Land Bank Properties"
##
## $type
##  [1] "Building"          "Non-profit"        "Financial"
##  [4] "Financial"         "Government"        "Financial"
##  [7] "Non-profit"        "Government"        "Financial"
## [10] "Financial"         "Non-profit"        "Government"
## [13] "Government"        "Government"        "Non-profit"
## [16] "Private Developer" "Private Developer" "Building"
## [19] "Building"          "Private Developer" "Private Developer"
## [22] "Non-profit"        "Building"          "Non-profit"
## [25] "Non-profit"        "Building"          "Building"
## [28] "Building"          "Financial"         "Financial"
## [31] "Building"          "Building"          "Building"
## [34] "Developer"         "Government"        "Building"
## [37] "Developer"         "Building"
##
## $title
##  [1] "Franklinton Senior Housing 1"
##  [2] "National Church Residents"
##  [3] "Finance Fund"
##  [4] "Ohio Housing Finace Agency"
##  [5] "Community Development Collab"
##  [6] "Ohio Capital Corporation for Housing"
##  [7] "Franklinton Development Association"
##  [8] "Franklinton Area Commission"
##  [9] "Ohio Capital Finacial Corporation"
## [10] "Ohio Capital Impact Corportation"
## [11] "Community Properties of Ohio"
## [12] "Columbus City Department of Development"
```

```
## [13] "US HUD Home Fund"
## [14] "FHAct50 Committee"
## [15] "Neighborhood Design Center"
## [16] "Kaufman Developer"
## [17] "Homeport"
## [18] "McDowell Place"
## [19] "Warner Junction"
## [20] "Model Group"
## [21] "THRIVE"
## [22] "Mount Carmel"
## [23] "Homes on Hartford"
## [24] "Central Ohio Community Improvement Corporation (Land Bank)"
## [25] "Central Ohio Community Land Trust (Land Trust)"
## [26] "Town Square Station"
## [27] "City View Homes"
## [28] "Boulevard Homes"
## [29] "US HUD Community Development Block Grant"
## [30] "Affordable Housing Trust"
## [31] "Gravity 1"
## [32] "Gravity 2"
## [33] "River and Rich"
## [34] "Casto"
## [35] "Columbus Metropolitian Housing Authority"
## [36] "400 W Rich"
## [37] "Urban Smart Growth"
## [38] "Land Bank Properties"
```

What if we wanted to add a node attribute? Say we want to add an "id" column with unique numbers for each node. This could be useful if we want to label them later. We can use `seq.int()` to get sequential integers from `1` to the length of the node list. You can almost hear the formula already, right? Sequential.integers(from = 1, to = length of node list)

```
seq.int(from = 1,to = length(V(netData)))
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30 31 32 33 34 35 36 37 38
```

There are the numbers, let's assign them to a new column in the network data. Lets call it `$id`.

```
V(netData)$id <- seq.int(from = 1,to = length(V(netData)))
```

And check to make sure it worked.

```
vertex_attr(netData)$id
```

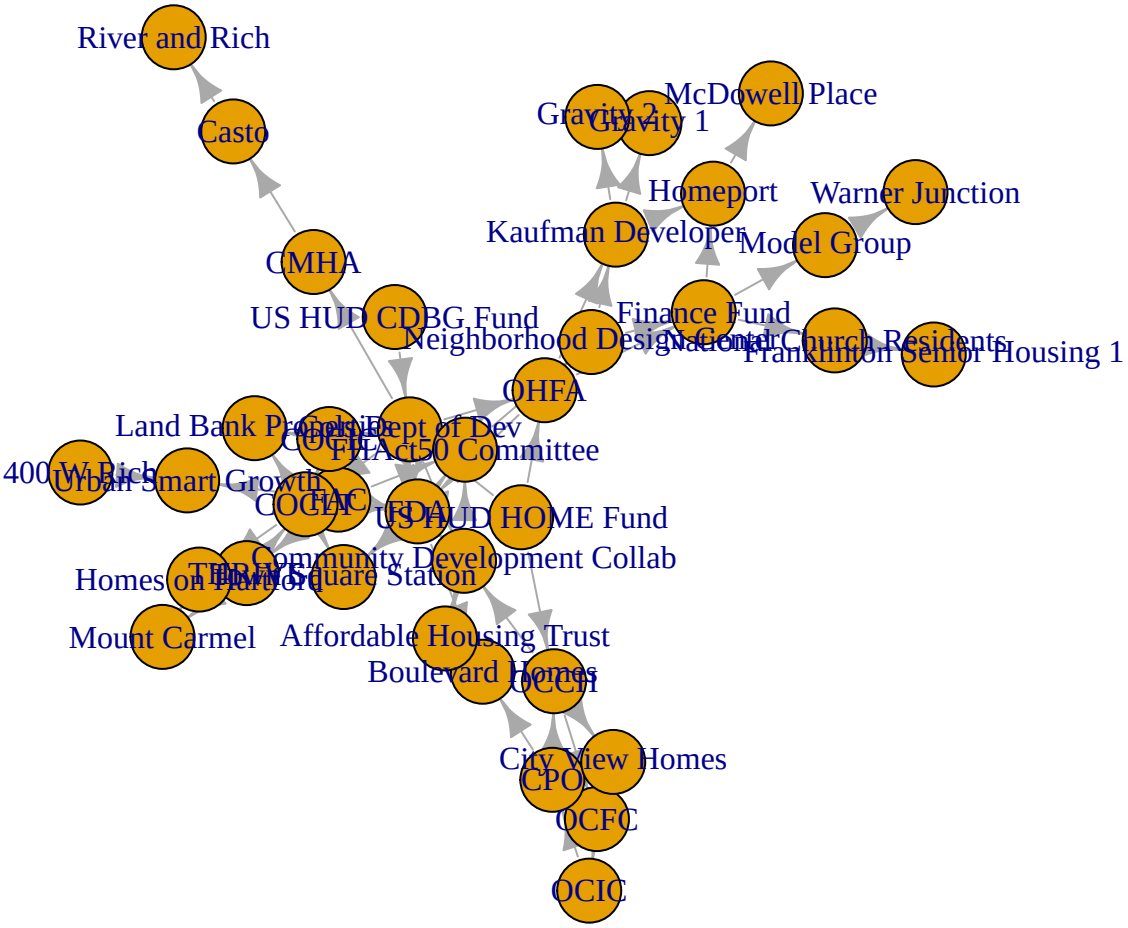
```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30 31 32 33 34 35 36 37 38
```

Violà!

Plotting the Network

A lot of the changes we will make to the network will be done when we plot it using the function, you guessed it, `plot()`.

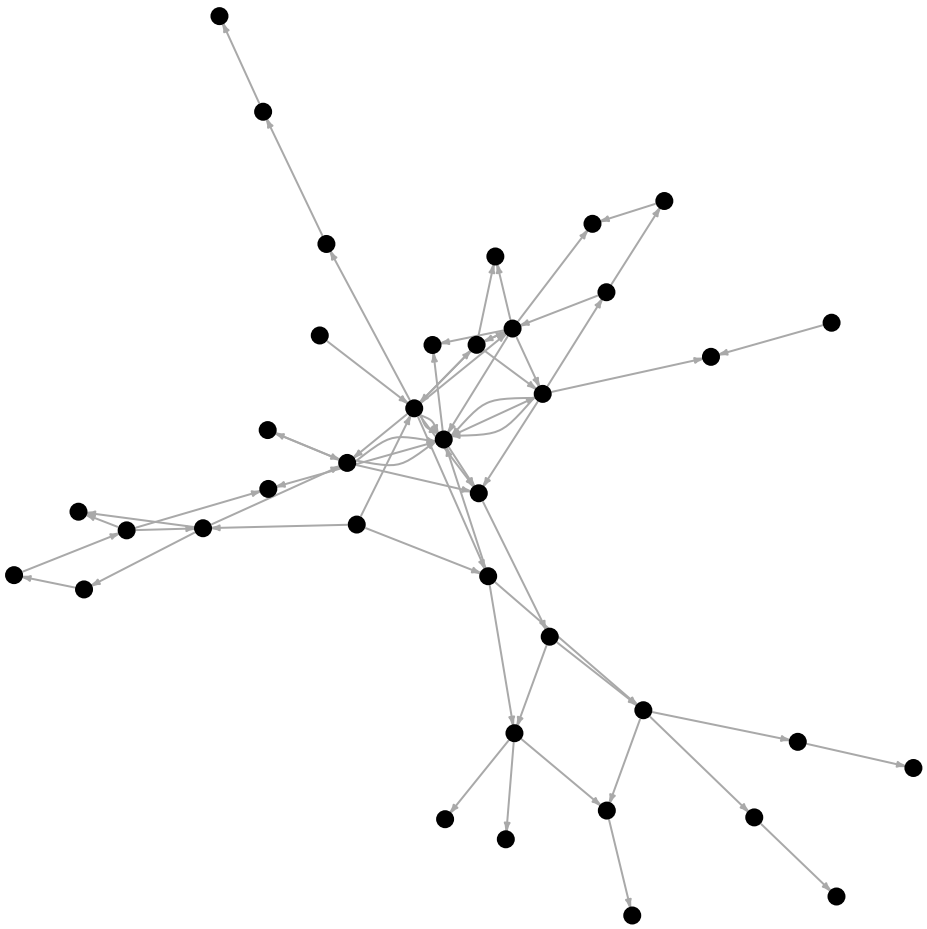
```
plot(netData)
```



eww.

Let's us the arguments in `plot()` to simplify our network. Let's look at `?igraph.plotting` to see the arguments it takes.

```
plot(netData,
     vertex.frame.color=NA,
     vertex.size=4,
     vertex.color="Black",
     vertex.label=NA,
     edge.arrow.size=.2)
```



That's better.

Visualizing Nodes

There are three main styles of the nodes that we will be changing to get our desired effect, color, size and label.

Plot() Arguments and their Node Parameters

Argument	Parameter
vertex.color	Node color
vertex.frame.color	Node border color
vertex.shape	One of “none”, “circle”, “square”, “csquare”, “rectangle”, “crectangle”, “vrectangle”, “pie”, “raster”, or “sphere”
vertex.size	Size of the node (default is 15)
vertex.size2	The second size of the node (e.g. for a rectangle)
vertex.label	Character vector used to label the nodes
vertex.label.family	Font family of the label (e.g.“Times”, “Helvetica”)
vertex.label.font	Font: 1 plain, 2 bold, 3, italic, 4 bold italic, 5 symbol
vertex.label.cex	Font size (multiplication factor, device-dependent)
vertex.label.dist	Distance between the label and the vertex
vertex.label.degree	The position of the label in relation to the vertex,where 0 right, “pi” is left, “pi/2” is below, and “-pi/2” is above

Adjusting the Colors of Nodes Let’s color the nodes according to their type. First we need to create a palette using `hues`, an R extension of medialab.github.io/iwanthue/. `iwanthue()` allows us to make a nice custom palette of any number of colors. It uses HCL values to get a pool of colors then divides them according to the n value. If you go to the website link you can play with the values then use them to write a function in R.

```
nodePal <- iwanthue(n = length(unique(V(netData)$type)),0,360,23,100,60,100)

nodePal
```

```
## [1] "#D98CD1" "#8BD1AA" "#86B5DD" "#D3C14F" "#82D25E" "#DE9E7F"
```

The next step is a little tricky. We have a list of colors and a list of node types. We need to select the color from `nodePal` based on `V(netData)$type`. Remember that we can select something from a list using `[]`. So...

```
nodePal[2]
```

```
## [1] "#8BD1AA"
```

We can use a couple of functions to return a number for each unique node type. This could be assigned to a column but since it is only being used to assign a color that is not necessary. Factors are a specific type of data which is list of data but also has *levels* which are basically each unique variable. Factors are often used for categorical data. What these functions are doing is returning what level each type is as a numeric. This can then be used to call out which color will be assigned using `[]`.

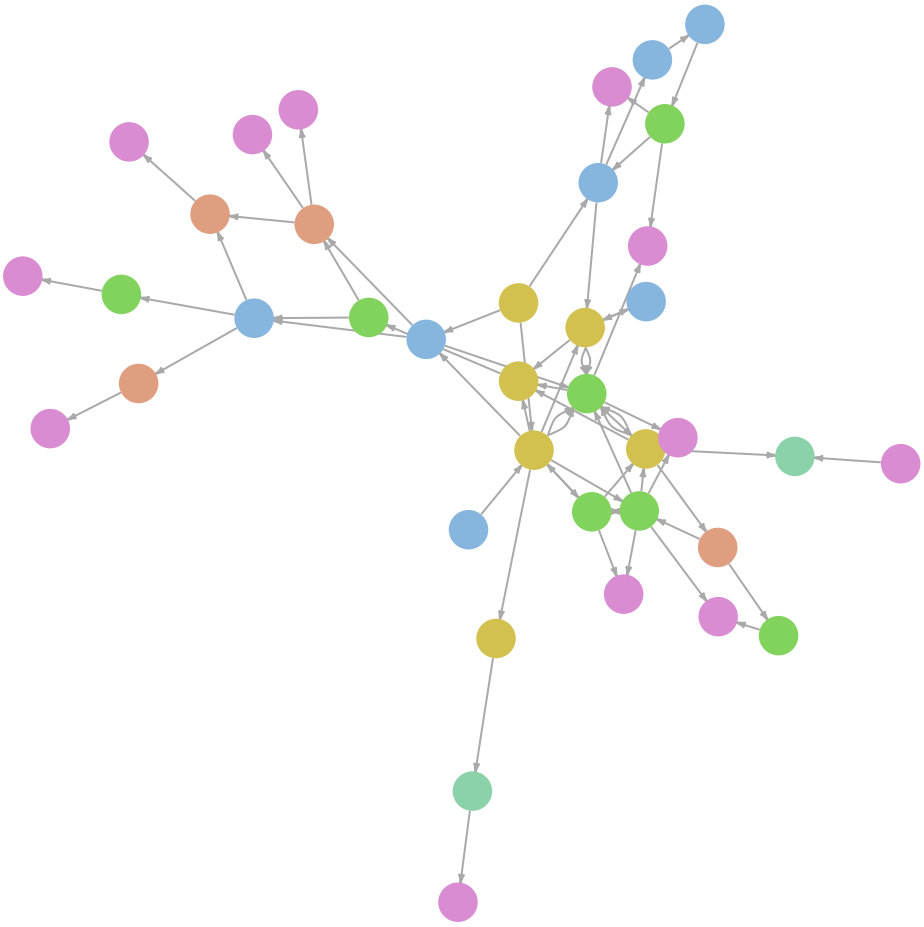
```
nodePal[as.numeric(as.factor(vertex_attr(netData,"type")))]
```

```
##  [1] "#D98CD1" "#82D25E" "#86B5DD" "#86B5DD" "#D3C14F" "#86B5DD" "#82D25E"
##  [8] "#D3C14F" "#86B5DD" "#86B5DD" "#82D25E" "#D3C14F" "#D3C14F" "#D3C14F"
## [15] "#82D25E" "#DE9E7F" "#DE9E7F" "#D98CD1" "#D98CD1" "#DE9E7F" "#DE9E7F"
## [22] "#82D25E" "#D98CD1" "#82D25E" "#82D25E" "#D98CD1" "#D98CD1" "#D98CD1"
```

```
## [29] "#86B5DD" "#86B5DD" "#D98CD1" "#D98CD1" "#D98CD1" "#8BD1AA" "#D3C14F"
## [36] "#D98CD1" "#8BD1AA" "#D98CD1"
```

And then this can be used to for the `vertex.color` parameter.

```
plot(netData,
     vertex.frame.color=NA,
     vertex.size=9,
     vertex.color=nodePal[as.numeric(as.factor(vertex_attr(netData,"type")))],
     vertex.label=NA,
     edge.arrow.size=.2)
```



Visualizing Edges

Plot() Arguments and their Edge Parameters

Argument	Parameter
edge.color	Edge color
edge.width	Edge width, defaults to 1
edge.arrow.size	Arrow size, defaults to 1
edge.arrow.width	Arrow width, defaults to 1
edge.lty	Line type, could be 0 or "blank", 1 or "solid", 2 or "dashed",3 or "dotted", 4 or "dotdash", 5 or "longdash", 6 or "twodash"
edge.label	Character vector used to label edges
edge.label.family	Font family of the label (e.g."Times", "Helvetica")
edge.label.font	Font: 1 plain, 2 bold, 3, italic, 4 bold italic, 5 symbol
edge.label.cex	Font size for edge labels
edge.curved	Edge curvature, range 0-1 (FALSE sets it to 0, TRUE to 0.5)
arrow.mode	Vector specifying whether edges should have arrows,possible values: 0 no arrow, 1 back, 2 forward, 3 both

Mapping Specific Colors to Edge Type What if we wanted to map a specific color to an edge type? We can do this by adding a edge attribute and selecting a color based on `$type` with a number of `ifelse()` statements. `ifelse` is a simple function that takes three arguments, a test, what to return if `TRUE`, and what to return if `FALSE`. A double `==` in R compares words or strings. The first line will assign "brown4" to any row where `$type` == "Natural" is `TRUE` and `NULL` to any where `$type` == "Natural" is `FALSE`. The next lines will ask the same for each CCF type and if `TRUE`, assign a color, or if `FALSE`, keep the color that is already assigned.

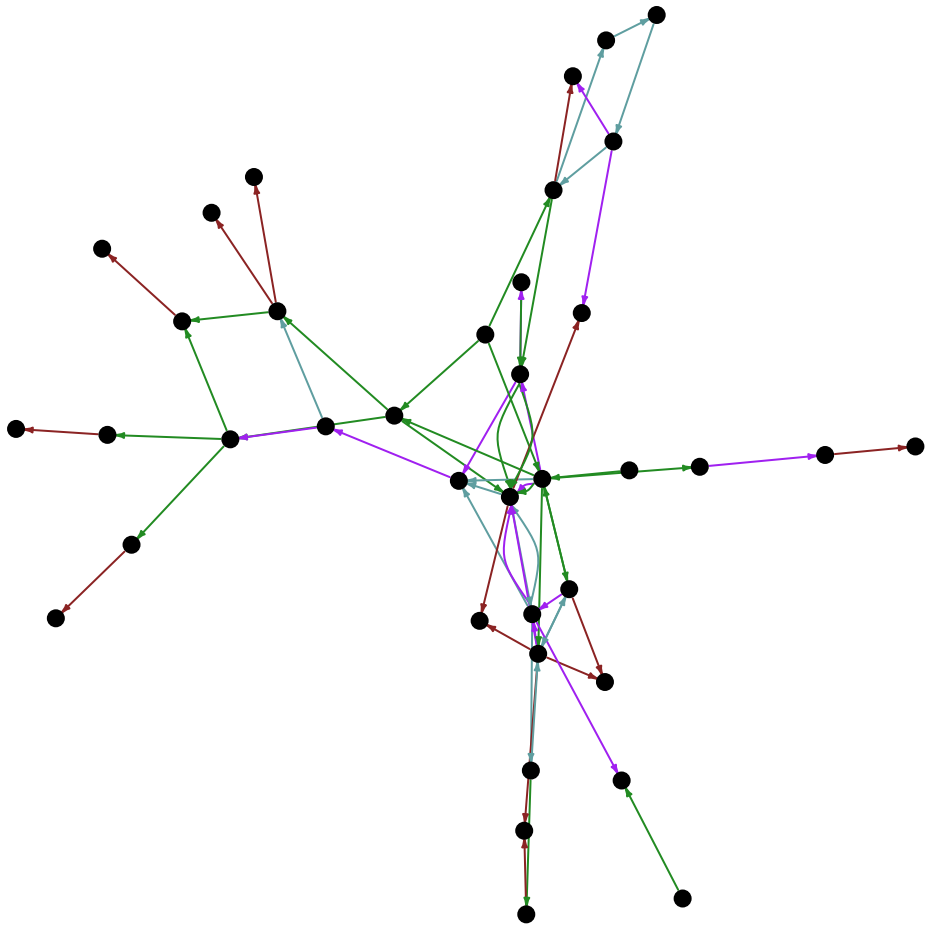
Sense we added a new attribute `$color`, we can just call the attribute with `egde.color`.

NOTE: All named colors that R accepts can be found at http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf.

NOTE 2: Notice how that functions don't have the `argument =` part of each argument? That is because R will just use them in their default order if they are not named. I suggest at first including them, but once you become familiar with the function you can use them less.

```
E(netData)$color <- ifelse(
  test = (E(netData)$type == "Natural"),
  yes = "darkgreen",
  no = "NULL")
E(netData)$color <- ifelse((E(netData)$type == "Physical"),"brown4",E(netData)$color)
E(netData)$color <- ifelse((E(netData)$type == "Financial"),"forestgreen",E(netData)$color)
E(netData)$color <- ifelse((E(netData)$type == "Social"),"cadetblue",E(netData)$color)
E(netData)$color <- ifelse((E(netData)$type == "Cultural"),"darkgoldenrod",E(netData)$color)
E(netData)$color <- ifelse((E(netData)$type == "Human"),"coral",E(netData)$color)
E(netData)$color <- ifelse((E(netData)$type == "Political"),"purple",E(netData)$color)

plot(netData,
     vertex.frame.color=NA,
     vertex.size=4,
     vertex.color="black",
     vertex.label=NA,
```

```
     edge.arrow.size=.2,
     edge.color=E(netData)$color)
```



Visualizing Plot Area (ie. Layouts and Legends and Titles)

Plot() Arguments and their Plot Area Parameters

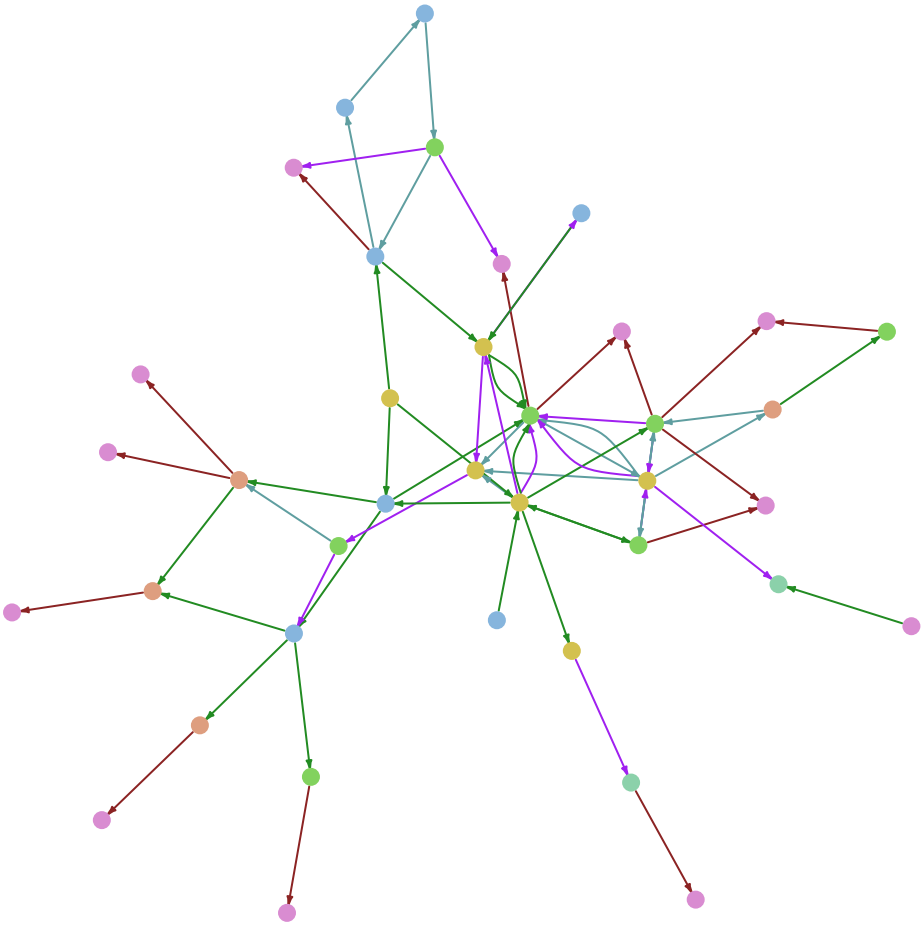
Argument	Parameter
margin	Empty space margins around the plot, vector with length 4
frame	if TRUE, the plot will be framed
main	If set, adds a title to the plot
sub	If set, adds a subtitle to the plot

Changing Layouts A Layout is the positioning of the nodes relative to other nodes and edges. This is done through algorithms in R called with different functions. One of the most frequently used is Fruchterman-Reingold. But check out the other layouts at https://igraph.org/r/doc/layout_.html.

Here is an example of using Kamada Kawai, with the colored nodes and edges.

```
kk <- layout_with_kk(netData)

plot(netData,
     layout=kk,
```

```
     vertex.frame.color=NA,
     vertex.size=4,
     vertex.color=nodePal[as.numeric(as.factor(vertex_attr(netData,"type")))],
     vertex.label=NA,
     edge.arrow.size=.2,
     edge.color=E(netData)$color)
```



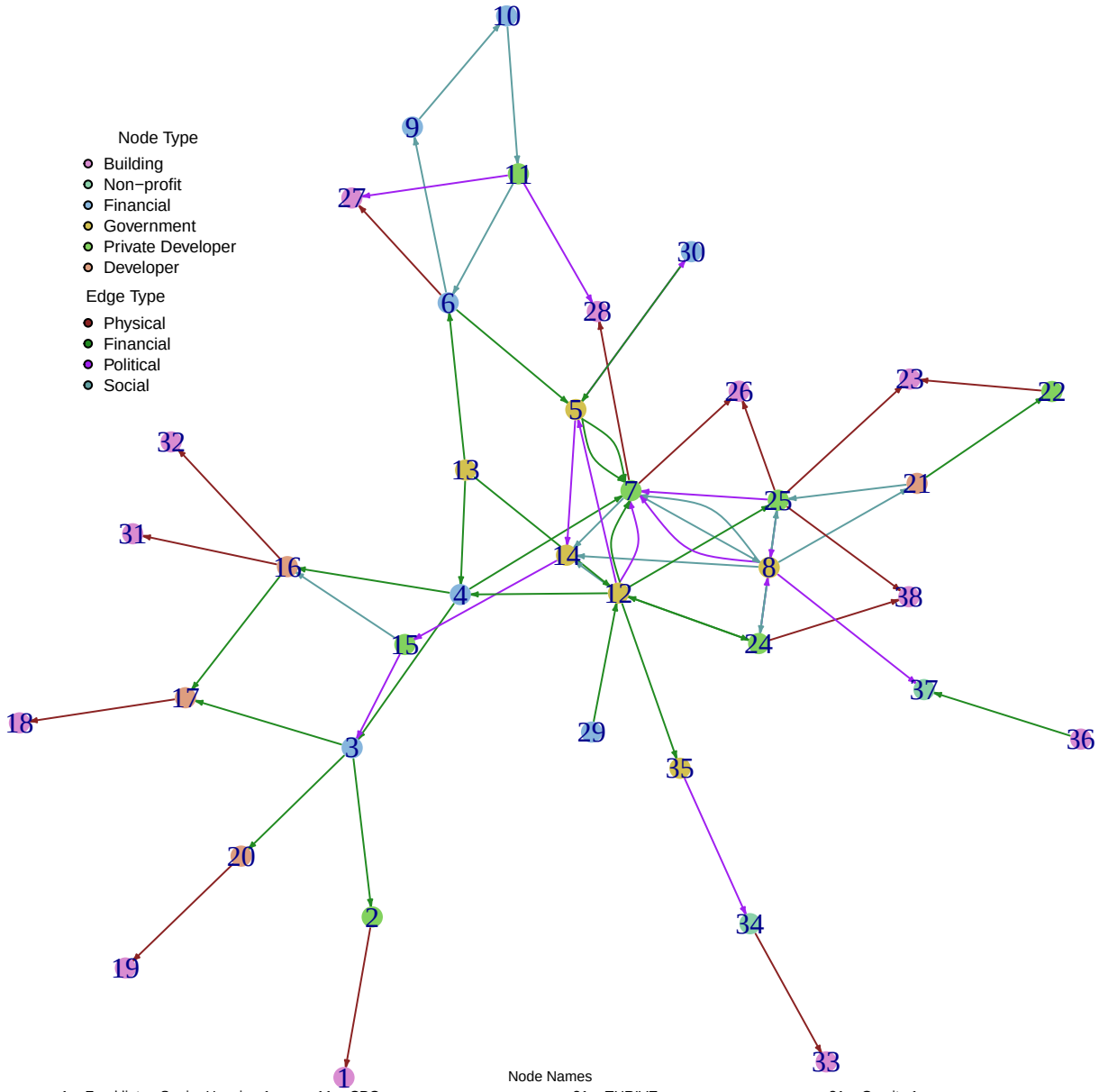
Creating Legends Now lets add some legends to clear things up.

`legend()` is a function like plot, and is very versatile. It will create a legend for the most recent plot. I has a number of arguments that are useful but can require a lot of trial and error to get it right.

```
plot(netData,
     layout=kk,
     main = "Franklinton, Ohio Affordable Housing Network",
     vertex.frame.color=NA,
     vertex.size=4,
     vertex.color=nodePal[as.numeric(as.factor(vertex_attr(netData,"type")))],
     vertex.label=V(netData)$id,
     edge.arrow.size=.2,
     edge.color=E(netData)$color)
legend(x = -.9,y = .8,
       legend = unique(V(netData)$type),
       pt.bg = nodePal,
       pch = 21,
```

```
       bty = "n",
       cex = .6,
       title = "Node Type")
legend(x = -.9,y = 0.5,
       legend = unique(E(netData)$type),
       pt.bg = unique(E(netData)$color),
       pch = 21,
       bty = "n",
       cex = .6,
       title = "Edge Type")
legend("bottom",
       legend = paste(V(netData)$id,V(netData)$name,sep=" - "),
       bty = "n",
       title = "Node Names",
       ncol = 4,
       cex = .5,
       y.intersp = .9)
```

Franklinton, Ohio Affordable Housing Network



Node Names

1 − Franklinton Senior Housing 1
2 − National Church Residents
3 − Finance Fund
4 − OHFA
5 − Community Development Collab
6 − OCCH
7 − FDA
8 − FAC
9 − OCFC
10 − OCIC
11 − CPO
12 − Cols Dept of Dev
13 − US HUD HOME Fund
14 − FHAct50 Committee
15 − Neighborhood Design Center
16 − Kaufman Developer
17 − Homeport
18 − McDowell Place
19 − Warner Junction
20 − Model Group
21 − THRIVE
22 − Mount Carmel
23 − Homes on Hartford
24 − COCIC
25 − COCLT
26 − Town Square Station
27 − City View Homes
28 − Boulevard Homes
29 − US HUD CDBG Fund
30 − Affordable Housing Trust
31 − Gravity 1
32 − Gravity 2
33 − River and Rich
34 − Casto
35 − CMHA
36 − 400 W Rich
37 − Urban Smart Growth
38 − Land Bank Properties

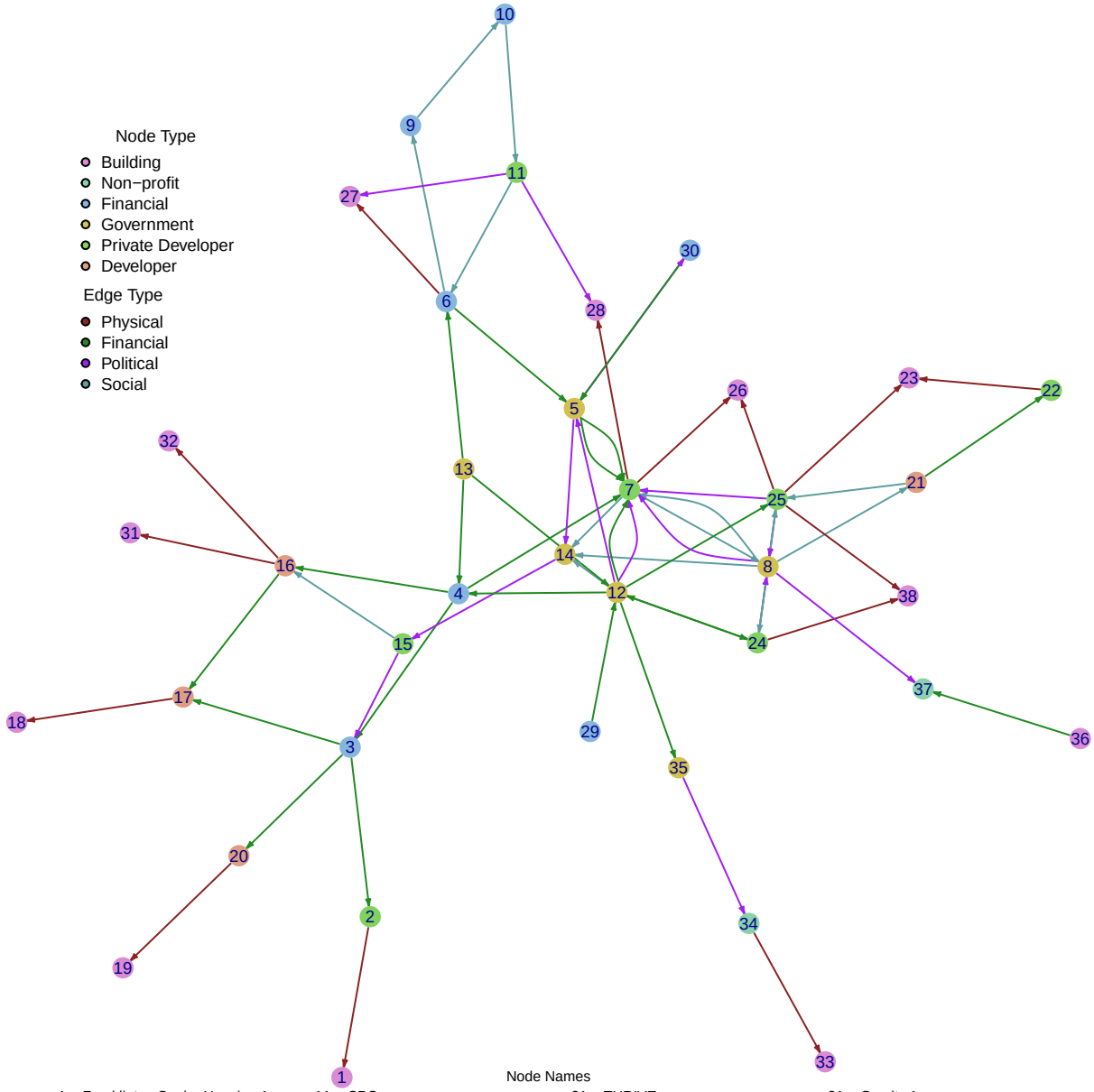
Final Script

Now let put everything together. The final script would look something like this.

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

nodeData <- read_excel(path = "fhact50data.xlsx", sheet = "Nodes")
edgeData <- read_excel(path = "fhact50data.xlsx", sheet = "Edges")

netData <- graph_from_data_frame(d = edgeData,
                                 vertices = nodeData,
                                 directed = TRUE)

V(netData)$id <- seq.int(from = 1,to = length(V(netData)))

nodePal <- iwanthue(n = length(unique(V(netData)$type)),0,360,23,100,60,100)

E(netData)$color <- ifelse((E(netData)$type == "Natural"),"darkgreen","NULL")
E(netData)$color <- ifelse((E(netData)$type == "Physical"),"brown4",E(netData)$color)
E(netData)$color <- ifelse((E(netData)$type == "Financial"),"forestgreen",E(netData)$color)
E(netData)$color <- ifelse((E(netData)$type == "Social"),"cadetblue",E(netData)$color)
E(netData)$color <- ifelse((E(netData)$type == "Cultural"),"darkgoldenrod",E(netData)$color)
E(netData)$color <- ifelse((E(netData)$type == "Human"),"coral",E(netData)$color)
E(netData)$color <- ifelse((E(netData)$type == "Political"),"purple",E(netData)$color)

kk <- layout_with_kk(netData)

plot(netData,
     layout=kk,
     main = "Franklinton, Ohio Affordable Housing Network",
     vertex.frame.color=NA,
     vertex.size=4,
     vertex.color=nodePal[as.numeric(as.factor(vertex_attr(netData,"type")))],
     vertex.label=V(netData)$id,
     vertex.label.cex = .6,
     vertex.label.family = "Helvetica",
     edge.arrow.size=.2,
     edge.color=E(netData)$color)
legend(x = -.9,y = .8,
       legend = unique(V(netData)$type),
       pt.bg = nodePal,
       pch = 21,
       bty = "n",
       cex = .6,
       title = "Node Type")
legend(x = -.9,y = 0.5,
       legend = unique(E(netData)$type),
       pt.bg = unique(E(netData)$color),
       pch = 21,
       bty = "n",
       cex = .6,
       title = "Edge Type")
legend("bottom",
```

```
       legend = paste(V(netData)$id,V(netData)$name,sep=" - "),
       bty = "n",
       title = "Node Names",
       ncol = 4,
       cex = .5,
       y.intersp = .9)
```

Franklinton, Ohio Affordable Housing Network



Node Names

1 − Franklinton Senior Housing 1
2 − National Church Residents
3 − Finance Fund
4 − OHFA
5 − Community Development Collab
6 − OCCH
7 − FDA
8 − FAC
9 − OCFC
10 − OCIC
11 − CPO
12 − Cols Dept of Dev
13 − US HUD HOME Fund
14 − FHAct50 Committee
15 − Neighborhood Design Center
16 − Kaufman Developer
17 − Homeport
18 − McDowell Place
19 − Warner Junction
20 − Model Group
21 − THRIVE
22 − Mount Carmel
23 − Homes on Hartford
24 − COCIC
25 − COCLT
26 − Town Square Station
27 − City View Homes
28 − Boulevard Homes
29 − US HUD CDBG Fund
30 − Affordable Housing Trust
31 − Gravity 1
32 − Gravity 2
33 − River and Rich
34 − Casto
35 − CMHA
36 − 400 W Rich
37 − Urban Smart Growth
38 − Land Bank Properties