Aesthetics in ggraph

Your name here

Two types of aesthetics

There are two main types of "aesthetics" in `ggraph`: changes that apply to all of the nodes/edges, and changes that differ based on some attribute of the nodes/edges.

For this tutorial, we'll be visualizing data from a 1990 paper on the French financial elite (`ffe`) in the `networkdata` package.

Let's load the data as a `tbl_graph`

```r
library(networkdata)
```

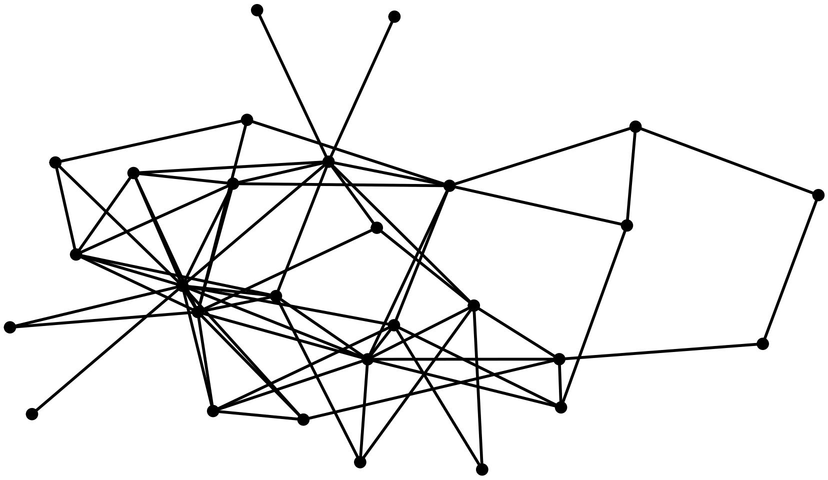
```

Attaching package: 'networkdata'


The following object is masked from 'package:dplyr':

    starwars
```

```r
G <- ffe_friends |> as_tbl_graph()
```

Now, let's see what it looks like it our basic `ggraph` plot.

```r
G |>
  ggraph() +
  geom_edge_fan() +
  geom_node_point()
```

```
Using "stress" as default layout
```



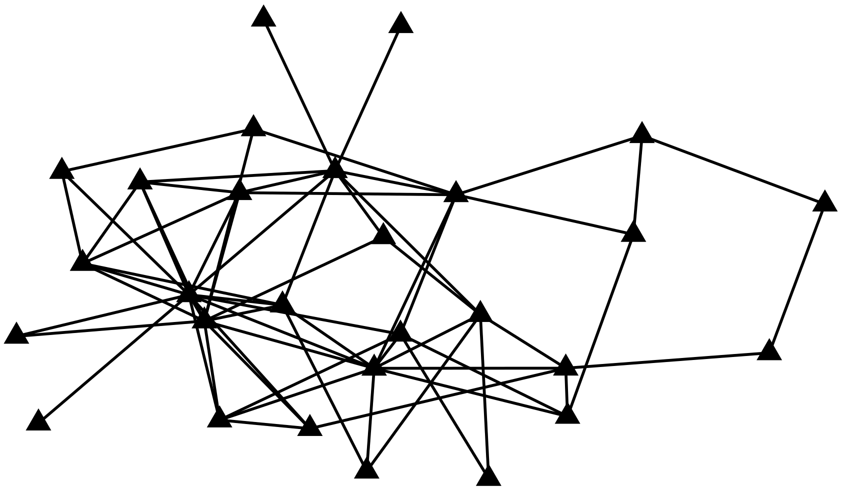
General aesthetics

We've already learned about the first kind of aesthetics—general aesthetics that apply to the whole graph. Things like changing the size of the nodes or the color of the edges.

For example, this changes the shape of the nodes to triangles and makes them a bit bigger.

```r
G |>
  ggraph() +
  geom_edge_fan() +
  geom_node_point(shape='triangle', size = 3)
```

```
Using "stress" as default layout
```



Exercise

Change the width of the edges to .5 and their color to "purple"

```
# Your code here
```

"Mapping" aesthetics

The second type of aesthetics maps some aspect of the data to some aspect of the visualization. For example, we might color nodes differently based on their major or size them differently based on their centrality.

Let's start by figuring out what the attributes of the nodes are in this data.

```
G
```

```
# A tbl_graph: 28 nodes and 66 edges
#
# An undirected simple graph with 1 component
#
# Node Data: 28 x 26 (active)
```

```
   birthdate birthplace cabinet clubs eliteprom elitevote   ena enayear
       <dbl>      <dbl>   <dbl> <dbl>     <dbl>     <dbl> <dbl>   <dbl>
 1      1943          2       1     2         1        24     1    1970
 2      1935          2       2     3         1        52     2      NA
 3      1924          1       2     1         1        45     2      NA
 4      1933          2       2     0         1        28     1    1960
 5      1931          1       1     3         1        45     1    1957
 6      1934          2       1     3         0         6     1    1962
 7      1932          1       2     1         1        48     2      NA
 8      1931          1       2     3         1        34     2      NA
 9      1911          2       2     0         0        13     2      NA
10      1935          1       2     3         1        23     2      NA
# i 18 more rows
# i 18 more variables: fathers.lev <dbl>, finance.min <dbl>, igyear <dbl>,
#   inspec.gen <dbl>, masons <dbl>, na <lgl>, normal.sch <dbl>, party <dbl>,
#   polytech <dbl>, polyyear <dbl>, prestige <dbl>, religion <dbl>,
#   sciencepoly <dbl>, socialreg <dbl>, topboards <dbl>, university <dbl>,
#   vertex.names <chr>, zipcode <dbl>
#
# Edge Data: 66 x 2
   from    to
  <int> <int>
1     2     6
2     3     6
3     1     7
# i 63 more rows
```

This is rich data! There are 26 different variables, some of which are clear (e.g, `birthdate`) and some of which are confusing (e.g., `topboards` or `igyear`). Let's choose one and visualize it.

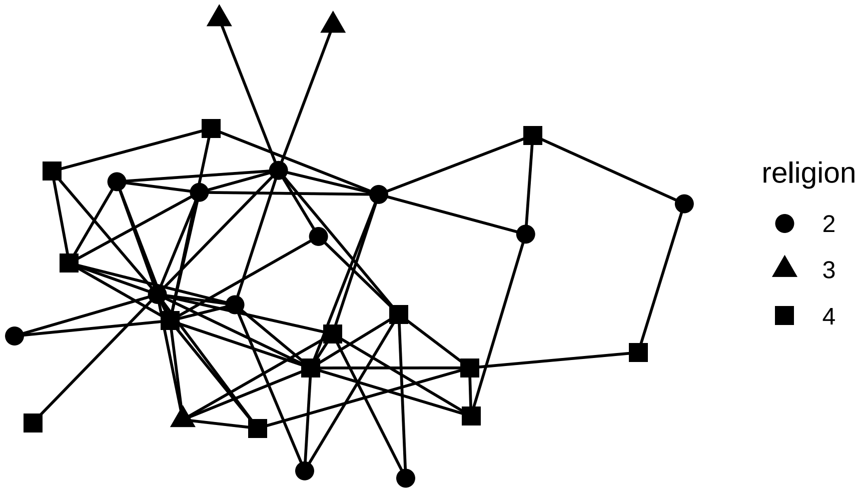
Usually, you would want to be guided by a question about the data which would inform your visualizations. Let's pretend like we really want to know whether people form friendships across religions. Let's visualize different religion as different shapes.

To do that, we "map" the shape of the nodes to the `religion` variable. The syntax to do this is to put the thing we want to change, then `=`, then the variable (column name) we want to use to change it. So, for changing shape to religion, it's `shape = religion`. We put that whole thing in `aes` (which stands for mapping aesthetic), and put it into the `geom_node_point` function.

```r
G |>
  mutate(religion = as_factor(religion)) |>
```

```
ggraph() +
geom_edge_fan() +
geom_node_point(aes(shape=religion), size = 3)
```

```
Using "stress" as default layout
```



Note the line `mutate(religion = as_factor(religion))` Try commenting out that line and running the code again.

You should get an error that is something like `A continuous variable can not be mapped to shape`. This is saying that in this data `religion` is a number and it doesn't make much sense to map a number to a shape, because usually numbers are ordered while shapes are unordered. So, we have to add the `mutate` step to change the `religion` variable type using `as_factor` which changes how R sees it from a number to a `factor`—in other words, a category (which is what religion really is, anyway!).

Exercise

Instead of mapping their shape to `religion`, map the color of the nodes to `religion`.

```
# Your code here
```

There are a number of node aesthetics that we can map variables to. The most common are: `shape`, `size`, and `color`. Let's do one more exercise to practice these.

Exercise

Go back to your code from when we learned about `mutate` and figure out how to calculate `degree` centrality for our network; save the centrality in a column called `degree` and change the `size` of nodes based on the `degree`.

```
# Your code here
```

Edge aesthetics

We can also map the edge aesthetics to variables from our edgelist spreadsheet. The most common approaches are to map the type of the edge to color and the weight of the edge to the `width`, `color`, or `alpha` of the edge.

Looking at our data, the edges don't have weights. For your exercise, we'll put together a few skills from the last few labs.

Exercise

Load the `bkfrab` dataset (also from `networkdata`, so load it in the same way as the `ff_elite` data above). This records how often pairs of subjects were seen talking to each other in a fraternity. Map the `alpha` of the edge to the `weight` in the `edges` dataframe.

Hints: Remember to use `as_tbl_graph` to change the network type, and to `activate` the edges dataframe.

```
# Your code here
```

Challenge Exercise

When we look at the `bkfrab` data, there are a lot of nodes with edge weight 10. Filter to just the edges whose weight is greater than 10, and map the `color` of the edges to their weight.

Note that this will leave a lot of nodes without any edges. The default `'stress'` layout puts them along the bottom, which looks kind of odd. You may want to use a layout like `'fr'` instead.

```
# Your code here
```

You now have all of the key skills to load in, `mutate` and `filter` networks, and to do some pretty cool visualizations with them. From here on out, we'll mostly be reusing these skills in new ways.