post2-serena-wu

Serena Wu

11/19/2017

(In case the reader needs to access my post, I have attached my github website here.)

Data Visualization

3D Scatter Plot and Network Graph

Introduction

Data visualization becomes increasingly important as it plays an enormous part in the reporting and presenting process. A nicely-done project must come with clear data visualization, which ensures effective communication of insights and engagement of attention on key concepts. Therefore, I would like to write a post about data visualization. As we have already learnt Bar chart, Histogram, and Scatter plot in Stat 133, I will extend from these diagrams and mainly focus on the other two: 3D Scatter Plot and Network Graph.

I will use the `cleanscores.csv` that we got from hw04 as the primary data to illustrate my examples for the following sections. I have put `cleanscores.csv` under the `post2` folder inside the `data` folder which is publicly available since I have pushed my `post2` folder to my github (you can access and download it from my github website attached at the very top of this post). Since everyone's `cleanscores.csv` might be a little different, I have attached my `cleanscores_dictionary.md` in the same folder right next to `cleanscores.csv`. And below I will read the first 100 lines of this csv and create `dat` that I will be using as my example.

```
dat <- read.csv("../data/cleanscores.csv")
dat <- head(dat, 100)
```

3D Scatter Plot

As we have learnt how to draw scatter plot in Stat 133, I want to extend data visualization a little bit further to see how we can visualize relationship among at most 3 factors in one plot. I will use the "scatterplot3d" package and therefore the first step we will take is to install and load this package.

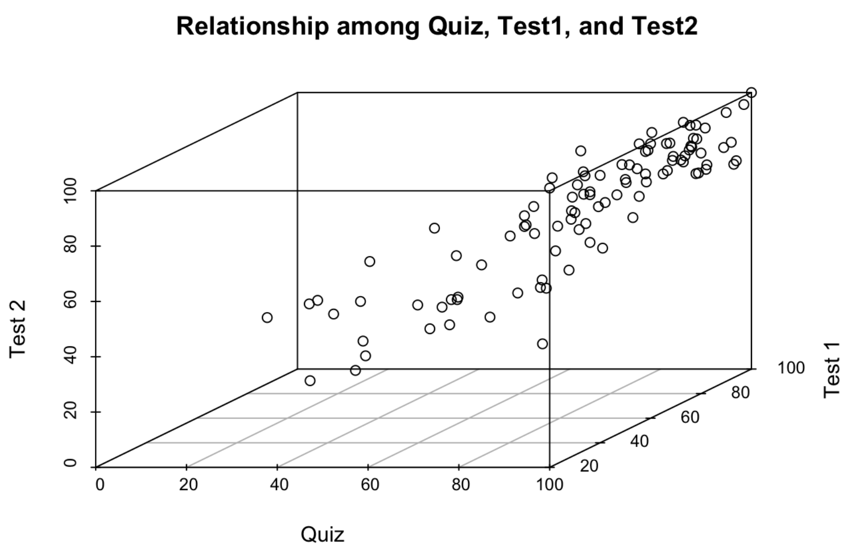
```
# install.packages("scatterplot3d")
library("scatterplot3d")
```

The main arguments that method `scatterplot3d()` takes in:

1. `x` : the coordinates of points in the plot.
2. `y` : the y coordinates of points in the plot, optional if x is an appropriate structure.
3. `z` : the z coordinates of points in the plot, optional if x is an appropriate structure.
4. `color` : colors of points in the plot, optional if x is an appropriate structure. Will be ignored if highlight.3d = TRUE.
5. `pch` : plotting "character", i.e. symbol to use.
6. `main` : an overall title for the plot.
7. `sub` : sub-title.
8. `xlim` , `ylim` , `zlim` : the x, y and z limits (min, max) of the plot. Note that setting enlarged limits may not work as exactly as expected (a known but unfixed bug).
9. `xlab` , `ylab` , `zlab` : titles for the x, y and z axis.
10. `scale.y` : scale of y axis related to x- and z axis.
11. `angle` : angle between x and y axis (Attention: result depends on scaling).
12. `axis` : a logical value indicating whether axes should be drawn on the plot.
13. `grid` : a logical value indicating whether a grid should be drawn on the plot.
14. `box` : a logical value indicating whether a box should be drawn around the plot.
15. `lab` : a numerical vector of the form c(x, y, len). The values of x and y give the (approximate) number of tickmarks on the x and y axes.
16. `highlight.3d` : points will be drawn in different colors related to y coordinates (only if type = "p" or type = "h", else color will be used). On some devices not all colors can be displayed. In this case try the postscript device or use highlight.3d = FALSE.

Now let's start to make some simple graphs using `scatterplot3d` ! If we want to explore the relationship among `Quiz` , `Test1` , and `Test2` . We can get a basic 3d scatter plot by:

```
scatterplot3d(x = dat$Quiz, y = dat$Test1, z = dat$Test2,
              xlab = "Quiz", ylab = "Test 1", zlab = "Test 2",
              main = "Relationship among Quiz, Test1, and Test2",
              sub = "Basic Graph")
```

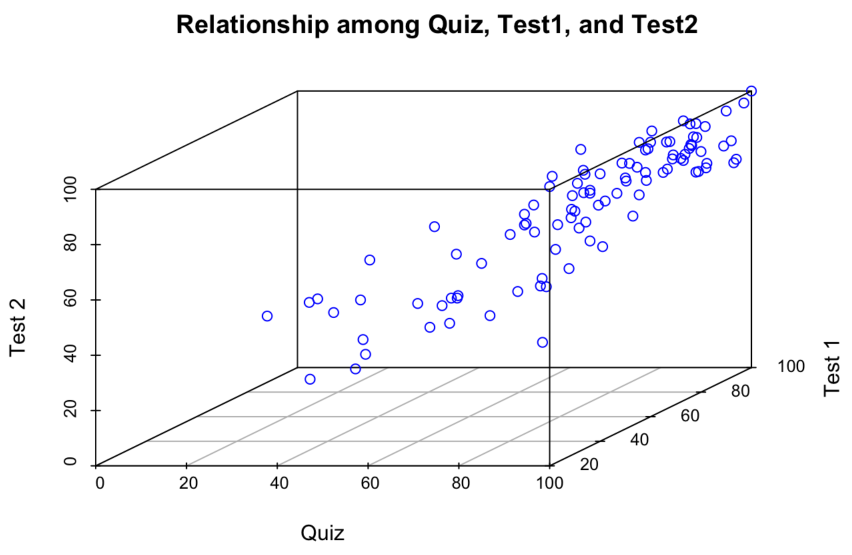


Basic Graph

We can see from the graph above that Quiz and Test1 have a positive relationship, Quiz and Test2 have a positive relationship, and although it's pretty hard to observe, we can still tell that Test1 and Test2 also have a positive relationship. Also, we can observe that `grid` is by default set to be TRUE. `box` is by default TRUE. `box` is by default set to be TRUE. And the `color` is black by default.

We can also change the color of the plot, either by setting `color`, or by setting `highlight.3d` to TRUE: Mehod 1: setting `color`

```
scatterplot3d(x = dat$Quiz, y = dat$Test1, z = dat$Test2,
              xlab = "Quiz", ylab = "Test 1", zlab = "Test 2",
              color = "blue", main = "Relationship among Quiz, Test1, and Test2",
              sub = "Getting color by setting color options")
```

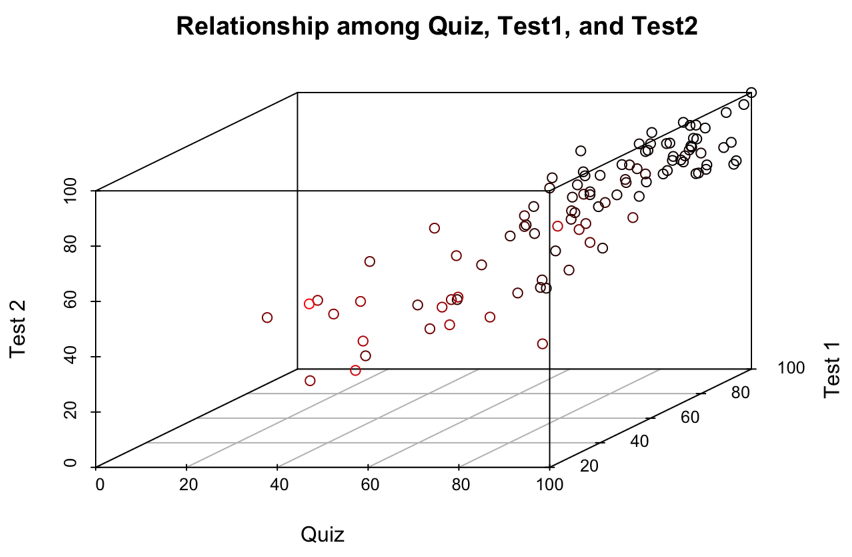


Getting color by setting color options

Here in the graph above we changed the color by setting `color` to "blue". Also, because `highlight.3d` is by default set to be FALSE, it doesn't affect our color here.

Method 2: setting `highlight.3d` to TRUE

```
scatterplot3d(x = dat$Quiz, y = dat$Test1, z = dat$Test2,
              xlab = "Quiz", ylab = "Test 1", zlab = "Test 2",
              main = "Relationship among Quiz, Test1, and Test2",
              sub = "Getting color by highlight.3d",
              highlight.3d = TRUE)
```



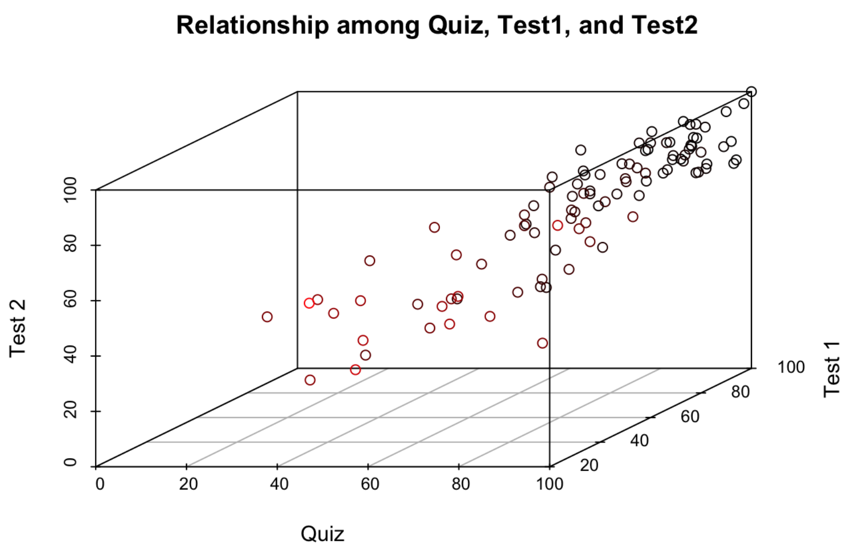
Getting color by highlight.3d

Here in the graph above we are setting `highlight.3d` to TRUE, and therefore points are drawn to different shades of red in response to y-coordinate.

In addition, let's set what will happen if we set both `color = "blue"` and `highlight.3d = TRUE`!

```
scatterplot3d(x = dat$Quiz, y = dat$Test1, z = dat$Test2,
              xlab = "Quiz", ylab = "Test 1", zlab = "Test 2",
              color = "blue", main = "Relationship among Quiz, Test1, and Test2",
              sub = "Getting color by setting both color options and highlight.3d",
              highlight.3d = TRUE)
```

```
## Warning in scatterplot3d(x = dat$Quiz, y = dat$Test1, z = dat$Test2, xlab =
## "Quiz", : color is ignored when highlight.3d = TRUE
```

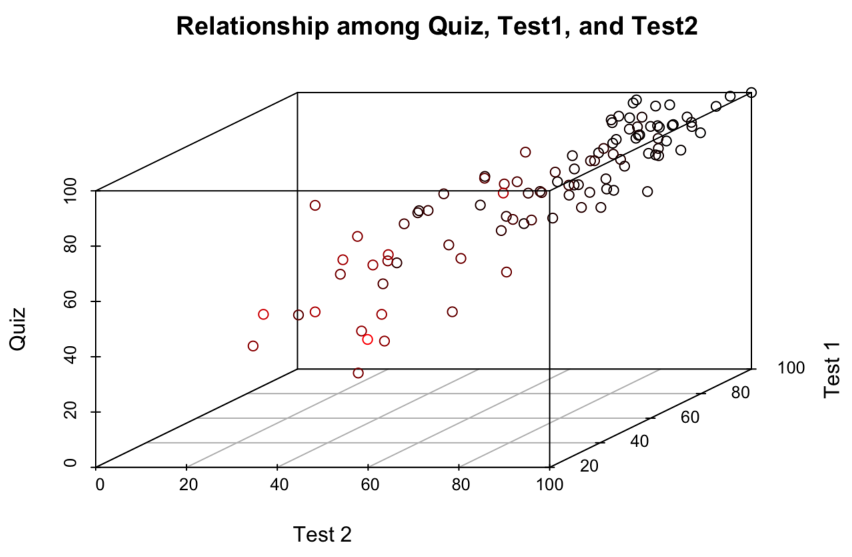


Getting color by setting both color options and highlight.3d

Here we can see `highlight.3d` will override `color = "blue"`, and we get a warning suggesting that color is ignored.

We can also adjust the axes of these three factors. For example, we can place `Test2` at x-axis, `Test1` at y-axis, and `Quiz` at z-axis to get a different angle and a clearer view of relationship between `Test1` and `Test2`.

```
scatterplot3d(x = dat$Test2, y =  dat$Test1, z = dat$Quiz,
              xlab = "Test 2", ylab = "Test 1", zlab = "Quiz",
              main = "Relationship among Quiz, Test1, and Test2",
              sub = "Changing angle", grid = TRUE, highlight.3d = TRUE)
```

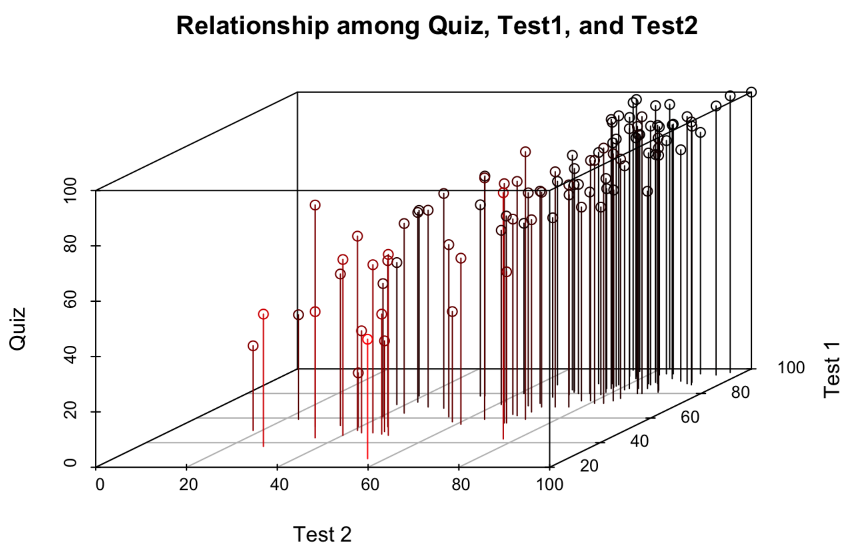


Changing angle

We can see clearly from this plot that `Test1` and `Test2` has a positive relationship.

In addition, a cool way that we can use to display x-y location of points is to set `type = "h"`.

```r
scatterplot3d(x = dat$Test2, y =  dat$Test1, z = dat$Quiz,
              xlab = "Test 2", ylab = "Test 1", zlab = "Quiz",
              main = "Relationship among Quiz, Test1, and Test2",
              sub = "Desplaying x-y location",
              grid = TRUE, highlight.3d = TRUE, type = "h")
```



Desplaying x-y location

Here we can see very clearly the x-y location of points.

Summary of 3D Scatter Plot

1. In the `scatterplot3d()` method, `hightlight.3d` is by default FALSE, and when set to TRUE, the graph displays different shades of red according to y-coordinate
2. `color` can be set to other options if we haven't set `hightlight.3d = TRUE`, and otherwise `color` will be ignored
3. If we cannot tell what relationship two factors are in, we can change the angle of the graph by moving them respectively to x and y axes. In this way we can observe them clearly
4. We can set `type = "h"` to display x-y location of points

Simple Network Graph

When analyzing networks of personal connections or related activities, we need to draw graphs (vertices joined by directed or undirected edges). Network analysis can be really complicated, but here I will focus on the simplest part: creating network graphs. Here I will use the `igraph` package.

```r
# install.packages("igraph")
library(igraph)
```

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

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

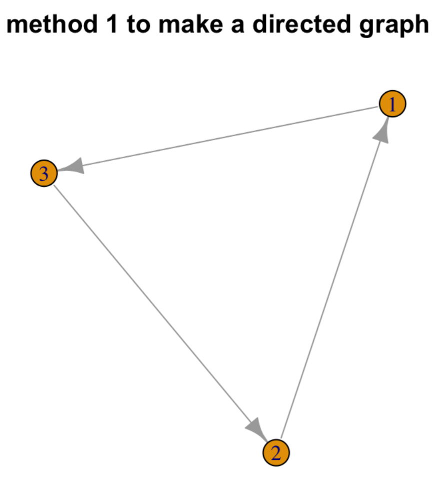
```
## The following object is masked from 'package:base':
##
##     union
```

We can create a graph by calling either `make_graph()`, `make_directed_graph()`, `make_undirected_graph()`, or `graph()`. Arguments the methods above take in:

1. `edges`: A vector defining the edges, the first edge points from the first element to the second, the second edge from the third to the fourth, etc.
2. `n = max(edges)`: The number of vertices in the graph. This argument is ignored (with a warning) if edges are symbolic vertex names.
3. `isolates`: Character vector, names of isolate vertices, for symbolic edge lists. It is ignored for numeric edge lists.
4. `make_graph()` and `graph()` can take an additional argument `directed` that takes `T` for creating directed graph and `F` for creating undirected ones (and the default is `T`).

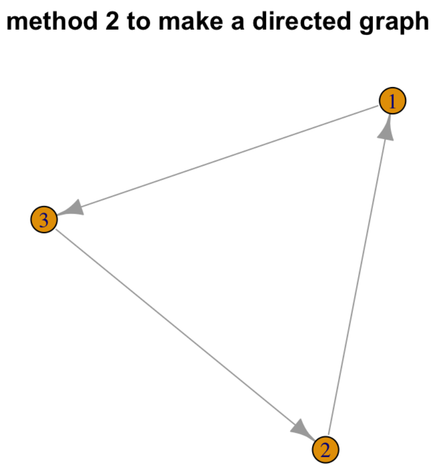
Directed Graph

```
directed_graph1 <- make_directed_graph(edges = c(1, 3, 3, 2, 2, 1))
plot(directed_graph1) + title(main = "method 1 to make a directed graph")
```



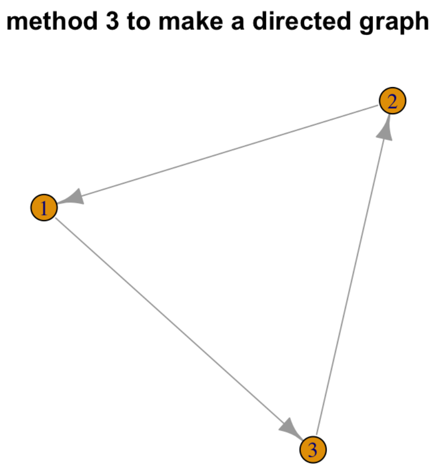
```
## integer(0)
```

```
directed_graph2 <- graph(edges = c(1, 3, 3, 2, 2, 1), directed = T)
plot(directed_graph2) + title(main = "method 2 to make a directed graph")
```



```
## integer(0)
```

```
directed_graph3 <- make_graph(edges = c(1, 3, 3, 2, 2, 1), directed = T)
plot(directed_graph3) + title(main = "method 3 to make a directed graph")
```

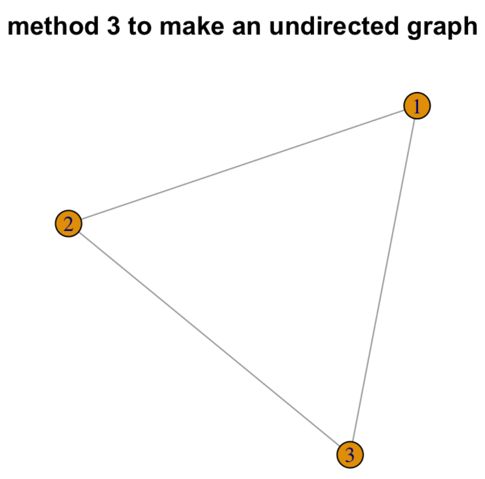


```
## integer(0)
```

Note: In these 3 graphs above, althought the place that these numbers 1, 2, 3 are placed are different places, they are in fact the same as the edges connecting these nodes are in the same direction. These three `logical(0)` is generated because I called method `title()`.

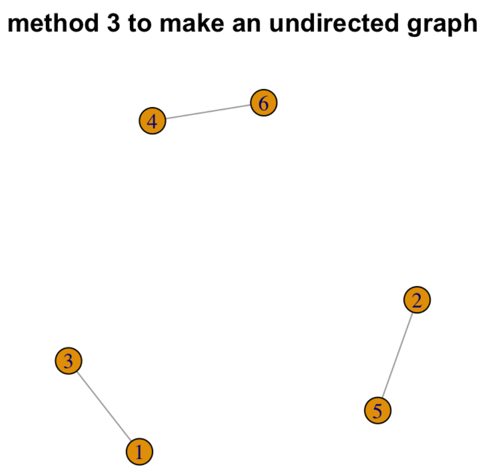
Undirected Graph

```
undirected_graph1 <- make_undirected_graph(edges = c(1, 3, 3, 2, 2, 1))
plot(undirected_graph1) + title(main = "method 3 to make an undirected graph")
```



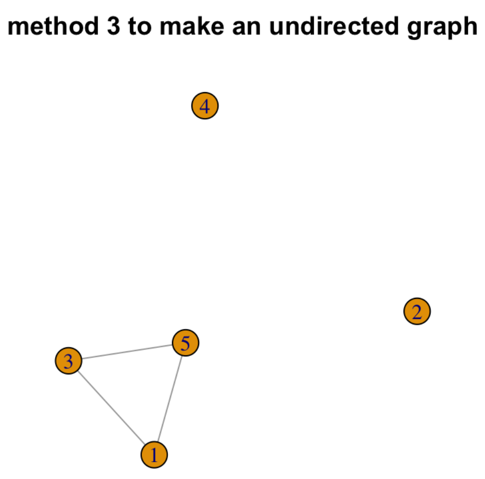
```
## integer(0)
```

```r
undirected_graph2 <- graph(edges = c(1, 3, 5, 2, 6, 4), directed = F)
plot(undirected_graph2) + title(main = "method 3 to make an undirected graph")
```



```
## integer(0)
```

```r
undirected_graph3 <- make_graph(edges = c(1, 3, 3, 5, 5, 1), directed = F)
plot(undirected_graph3) + title(main = "method 3 to make an undirected graph")
```

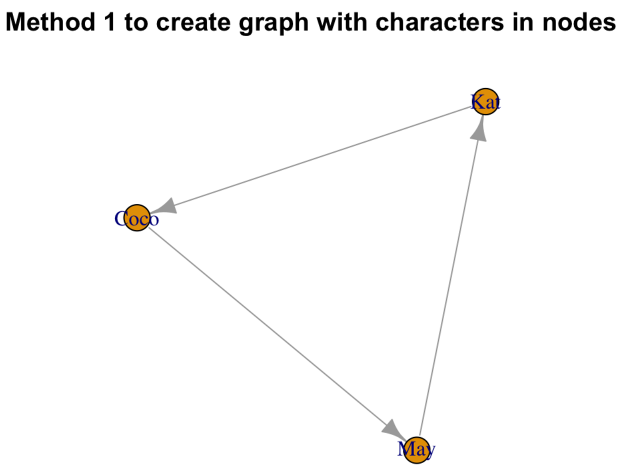


```
## integer(0)
```

Note: in these 3 graphs above, I am deliberately making these three to display different connections. As we can see, the nodes are by default consecutive from 1 to max(edges). Even if we don't mention some of the nodes in `edges`, they are nevertheless displayed, although not connected to any other node. These three `logical(0)` is generated because I called method `title()`.

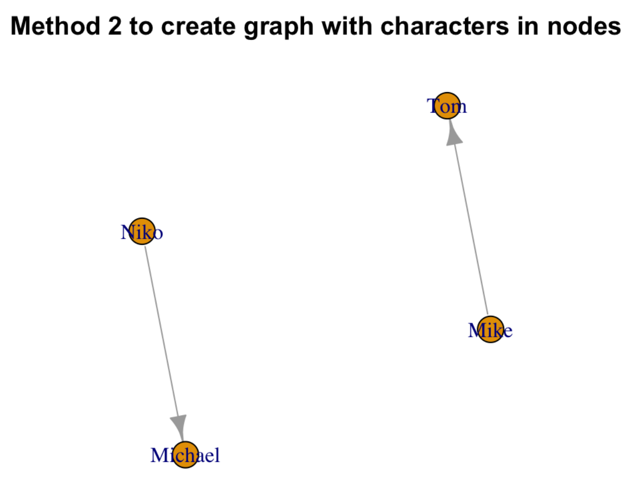
We can also name the nodes by characters, instead of numbers.

```
char_graph1 <- graph(edges = c("Coco", "May", "May", "Kat", "Kat", "Coco"))
plot(char_graph1) + title("Method 1 to create graph with characters in nodes")
```



```
## integer(0)
```

```
char_graph2 <- make_graph(edges = c("Niko", "Michael", "Mike", "Tom"))
plot(char_graph2) + title("Method 2 to create graph with characters in nodes")
```

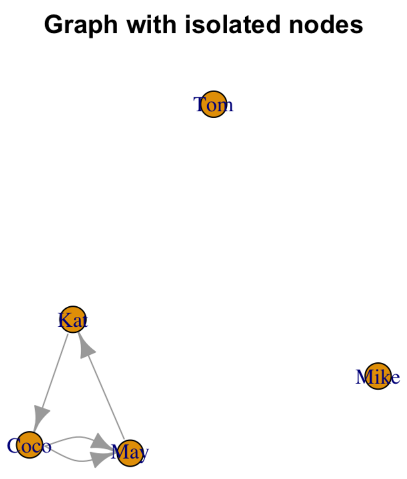


```
## integer(0)
```

Note: in the 2 graphs above, we can observe that the edges can not only be numbers, but can also be characters. Also, `graph` and `make_graph` create directed graphs by default. These two `logical(0)` is generated because I called method `title()`.

We can also create graphs with isolated nodes.

```
isolated_graph <- graph(edges = c("Coco", "May", "May", "Kat", "Kat", "Coco", "Coco", "May"), isolates = c("Tom",
"Mike"))
plot(isolated_graph) + title(main = "Graph with isolated nodes")
```



```
## integer(0)
```

We can see from above that using `isolated` we can make some nodes isolated (no connecting edges with others) from the others.

Now we can access the vertices, edges, and other information about these graphs. Let’s use `isolated_graph` as an example.

```
E(isolated_graph) # access the edges
```

```
## + 4/4 edges from b1a0b2d (vertex names):
## [1] Coco->May  May ->Kat  Kat ->Coco Coco->May
```

```
V(isolated_graph) # access the vertices
```

```
## + 5/5 vertices, named, from b1a0b2d:
## [1] Coco May  Kat  Tom  Mike
```

```r
isolated_graph[] # access the network matrix
```

```
## 5 x 5 sparse Matrix of class "dgCMatrix"
##      Coco May Kat Tom Mike
## Coco    .   2   .   .    .
## May     .   .   1   .    .
## Kat     1   .   .   .    .
## Tom     .   .   .   .    .
## Mike    .   .   .   .    .
```

We can now try to add some attributes to the network:

```r
V(isolated_graph)$name # check the names of vertices
```

```
## [1] "Coco" "May"  "Kat"  "Tom"  "Mike"
```

```r
E(isolated_graph)$type <- rep(c("classmates", "roomates"), 2) # assign names to the edges
edge_attr(isolated_graph) # check the attributes of the edges
```

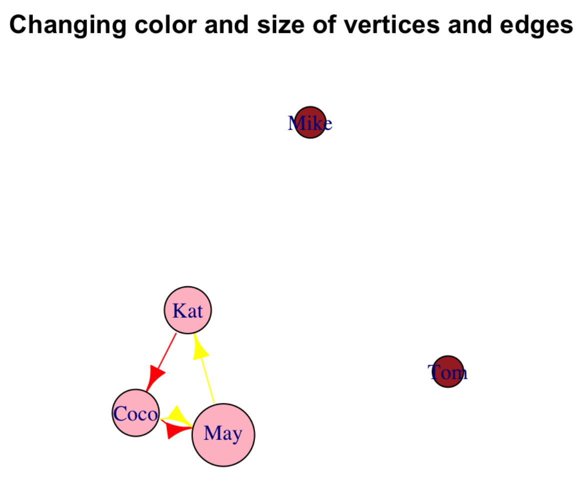
```
## $type
## [1] "classmates" "roomates"   "classmates" "roomates"
```

```r
vertex_attr(isolated_graph) # check the attributes of the vertices
```

```
## $name
## [1] "Coco" "May"  "Kat"  "Tom"  "Mike"
```

Changing color and size of vertices and edges can make it more engaging to analyze. We can use the attributes of the vertices and edges to make them more colorful.

```r
V(isolated_graph)$color <- c("pink", "pink", "pink", "brown", "brown") # change the color of vertices
E(isolated_graph)$color <- c("red", "yellow", "red", "yellow") # change the color of edges
V(isolated_graph)$size <- c(30, 40, 30, 20, 20) # change the size of the vertices
E(isolated_graph)$size <- c(2, 1, 2, 1) # change the size of the edges
plot(isolated_graph) + title(main = "Changing color and size of vertices and edges")
```



```
## integer(0)
```

From the above, we can see that we can use the `color` and `size` attributes from the vertices and edges of the graph to assign different colors and sizes to each node and edge.

Summary of Network Graph

1. We have several ways to create directed graphs and undirected graphs: `make_graph()`, `make_directed_graph()`, `make_undirected_graph()`, and `graph()`. Note that `make_graph()` and `graph()` create directed graphs by default
2. When we create nodes with numeric names, by default consecutive vertices from 1 to max(edges) are created. Even if we don't mention some of the node names in `edges`, they are nevertheless displayed, although not connected to any other node. For example, by saying `graph(edges = c(1, 3, 3, 1))`, we are not only creating node 1 and 3 (which points to each other), but also creating node 2
3. We can use `isolates` in the graph functions to make isolated vertices (vertices that are not connected with other vertices)
4. We can access the attributes of a graph by calling `V(graph_name)` (access vertices), `E(graph_name)` (access edges), and

`graph_name[]` (access network matrix).
5. Changing color and size helps readers to better visualize and analyze the graph. We can change the vertice attributes by setting `V(graph_name)$color` and `V(graph_name)size`. For edges, we can set `E(graph_name)$color` and `E(graph_name)$size` to desirable values

Summary

Data analysis relies heavily on data visualization. Proper visualization of data gives the audience a clearer look of what is going on with the analysis and helps people engage on the most essential idea of the research. Here I focus on two important topics in data visualization: 3D Scatter Plot, and Network Graph.

I start every section by installing and loading necessary packages, and then list the main arguments of the methods I will focus on. Afterwards, I gave extensive examples to illustrate how to use the methods. I have also included a Summary section in the end of each section.

The main take-home massage for this two parts are:

For 3d Scatter Plot:

1. In the `scatterplot3d()` method, `hightlight.3d` is by default FALSE, and when set to TRUE, the graph displays different shades of red according to y-coordinate
2. `color` can be set to other options if we haven't set `hightlight.3d = TRUE`, and otherwise `color` will be ignored
3. If we cannot tell what relationship two factors are in, we can change the angle of the graph by moving them respectively to x and y axes. In this way we can observe them clearly.
4. We can set `type = "h"` to display x-y location of points

(If you are interested, you can also go back to the first section and have a look at the main arguments that `scatterplot3d` takes in; I have listed them right below importing the package.)

For Network Graphs:

1. We have several ways to create directed graphs and undirected graphs: `make_graph()`, `make_directed_graph()`, `make_undirected_graph()`, and `graph()`
2. When we create nodes with numeric names, by default consecutive vertices from 1 to max(edges) are created
3. We can use `isolates` in the graph functions to make isolated vertices (vertices that are not connected with other vertices)
4. We can access the attributes of a graph by calling `V(graph_name)` (access vertices), `E(graph_name)` (access edges), and `graph_name[]` (access network matrix)
5. We can change the vertice attributes by setting `V(graph_name)$color` and `V(graph_name)size`. For edges, we can set `E(graph_name)$color` and `E(graph_name)$size` to desirable values

If you are interested, you can follow my comments and descriptions above every code chunk and try to run my code by yourself. They are really simple and easy to run. You are also welcome to click on the following link under the Reference section and read about relative topics on a higher level.

References:

1. https://en.wikipedia.org/wiki/Data_visualization
2. https://plot.ly/r/3d-scatter-plots/
3. http://www.sthda.com/english/wiki/scatterplot3d-3d-graphics-r-software-and-data-visualization
4. http://kateto.net/networks-r-igraph
5. https://en.wikipedia.org/wiki/Project_network
6. http://www.kateto.net/wp-content/uploads/2016/01/NetSciX_2016_Workshop.pdf
7. https://www.r-bloggers.com/an-example-of-social-network-analysis-with-r-using-package-igraph/