London Train Network Analysis

Maya Maciel-Seidman

2024-04-14

Motivation

London has an extensive and complex train system, which many Londoners and tourists rely onto traverse the city. The research questions I am trying to answer are how can the London train system be mapped as a network? Additionally, which train stations in London are the most connected and integral within this network?

Data

```
# Read in data for nodes:
stations <- read.csv("./stops.csv")
# Read in data for edges:
routes <- read.csv("./routes.csv")

# Examine node data:
dim(stations)
```

```
## [1] 369   6
```

```
head(stations)
```

```
##   X..index              name         nodeLabel  nodeLat     nodeLong
## 1        0         abbeyroad         abbeyroad 51.53195  0.003737786
## 2        1           westham           westham 51.52853  0.005331807
## 3        2      actoncentral      actoncentral 51.50876 -0.263415792
## 4        3 willesdenjunction willesdenjunction 51.53223 -0.243894729
## 5        4         actontown         actontown 51.50307 -0.280288296
## 6        5      chiswickpark      chiswickpark 51.49437 -0.267722520
##                             X_pos
## 1 array([1.67398194, 6.84895367])
## 2 array([1.69016166, 6.8574384 ])
## 3 array([1.92309925, 7.34721208])
## 4 array([1.89444118, 7.329156  ])
## 5 array([2.03649598, 7.40223207])
## 6 array([2.02276546, 7.38930105])
```

```r
summary(stations)
```

```
##     X..index       name            nodeLabel           nodeLat
##  Min.   :  0   Length:369         Length:369         Min.   :51.38
##  1st Qu.: 92   Class :character   Class :character   1st Qu.:51.50
##  Median :184   Mode  :character   Mode  :character   Median :51.52
##  Mean   :184                                         Mean   :51.53
##  3rd Qu.:276                                         3rd Qu.:51.56
##  Max.   :368                                         Max.   :51.71
##     nodeLong           X_pos
##  Min.   :-0.61142   Length:369
##  1st Qu.:-0.22446   Class :character
##  Median :-0.13826   Mode  :character
##  Mean   :-0.14722
##  3rd Qu.:-0.04752
##  Max.   : 0.25142
```

```r
# Examine edge data:
dim(routes)
```

```
## [1] 441   4
```

```r
head(routes)
```

```
##   X..source target weight layer
## 1         0      1      1     3
## 2         0    352      1     3
## 3         1     77      2     1
## 4         1    106      1     1
## 5         1    219      1     1
## 6         1    321      2     1
```

```r
summary(routes)
```

```
##    X..source         target          weight          layer
##  Min.   :  0.0   Min.   :  1.0   Min.   :1.000   Min.   :1.000
##  1st Qu.: 48.0   1st Qu.:131.0   1st Qu.:1.000   1st Qu.:1.000
##  Median :114.0   Median :223.0   Median :1.000   Median :1.000
##  Mean   :130.7   Mean   :212.2   Mean   :1.141   Mean   :1.397
##  3rd Qu.:206.0   3rd Qu.:300.0   3rd Qu.:1.000   3rd Qu.:2.000
##  Max.   :367.0   Max.   :368.0   Max.   :3.000   Max.   :3.000
```

The data I am using to examine London's train network is data collected in 2013 from the Transport for London official website. It can be downloaded from the Netzchleuder network catalogue and repository at this link: https://networks.skewed.de/net/london_transport.

The data from Netzchleuder is in two csv files. The first file (`stations`) is a dataset of all of the Transport for London train stations. There are 369 observations of stations and each station is described by 6 variables including a numerical index, station name, latitude, and longitude. There is no missingness in this data. The second file (`routes`) is a dataset of all of the Transport for London routes. There are 441 observations of routes and each route is described by 4 variables including the route's source (starting station index), target (ending station index), and weight. There is also no missingness in this data.

This data can be used for network analysis since the `routes` df describes the edges of the network and is formated as an edgelist, which can easily be converted to a network object. The `stations` df describes the nodes of the network. This network is weighted and directed. The weights of the edges are the distances between each station, which is already described as the weight variable in the `routes` file. The network is also directed because train routes are directed from one station to another. This is reflected by the source and target variables of the `routes` file.

```
# Get rid of the first index being 0 in both data sets:
routes <- routes %>% mutate(X..source = X..source+1)
stations <- stations %>% mutate(X..index = X..index+1)

# Selecting only sources and targets for edges:
routes <- routes %>% select(X..source, target)

# Selecting only indices and names for nodes:
stations <- stations %>% select(X..index, name)
stations <- stations %>% filter(X..index!=369)

# Creating network object from edgelist:
map <- igraph::graph_from_edgelist(as.matrix(routes), directed=TRUE)

# Obtain weights:
weights <- read.csv("./routes.csv") %>% select(weight)
# Set weights of network:
E(map)$weight <- weights
```

To construct the network, I first increased the index number of each station by 1 in order to get rid of the lowest index number being 0. Then, I selected only the sources and targets from `routes` to have only the required elements for the edgelist. I selected only the station indices and names from `stations` to have only the necessary information that I needed for the nodes. Then, I made the network, which I named `map` by using `graph_from_edgelist()`, which creates an igraph network object from an edgelist matrix. I passed in `routes` as a matrix and set `directed` to TRUE since this is a directed network. Then, I obtained the weights of each edge of the network and set the weights of each edge to its corresponding weight.

Description

```
# Summary of the structure of the network:
summary(map)
```

```
## IGRAPH e595339 D-W- 368 441 --
## + attr: weight (e/x)
```

```
map
```

```
## IGRAPH e595339 D-W- 368 441 --
## + attr: weight (e/x)
## + edges from e595339:
##  [1]  1->  1  1->352  2-> 77  2->106  2->219  2->321  2->347  3->  3  3->216
## [10]  4->224  4->258  4->224  4->258  4->260  4->338  5->  5  5->  6  5->160
## [19]  5->301  6->  6  7->215  7->221  7->345  8->  8  8->295  9->  9  9-> 34
```

```
## [28]  9-> 39  9-> 58 10-> 10 10->295 11->348 11-> 36 11->251 12-> 12 12->156
## [37] 13-> 65 13->290 13->324 14-> 14 14->354 15->300 16-> 16 17->117 17->122
## [46] 18-> 18 18->318 19->309 20-> 20 20->181 21-> 39 22-> 22 22->169 23->262
## [55] 24-> 24 24-> 73 25->311 26-> 26 26-> 95 27-> 97 27->191 27->192 28-> 28
## [64] 28-> 29 28-> 30 28-> 31 28->175 28->187 28->350 29-> 67 29-> 68 29->279
## + ... omitted several edges
```

```
# Obtain number of nodes in network:
igraph::vcount(map)
```

```
## [1] 368
```

```
# Obtain number of edges in network:
igraph::ecount(map)
```

```
## [1] 441
```

```
# Obtain list of vertices of network:
igraph::V(map)
```

```
## + 368/368 vertices, from e595339:
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##  [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##  [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
##  [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
##  [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
##  [91]  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
## [109] 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
## [127] 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
## [145] 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162
## [163] 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
## + ... omitted several vertices
```

```
# Obtain list of edges of network:
igraph::E(map)
```

```
## + 441/441 edges from e595339:
##  [1]  1->  1  1->352  2-> 77  2->106  2->219  2->321  2->347  3->  3  3->216
## [10]  4->224  4->258  4->224  4->258  4->260  4->338  5->  5  5->  6  5->160
## [19]  5->301  6->  6  7->215  7->221  7->345  8->  8  8->295  9->  9  9-> 34
## [28]  9-> 39  9-> 58 10-> 10 10->295 11->348 11-> 36 11->251 12-> 12 12->156
## [37] 13-> 65 13->290 13->324 14-> 14 14->354 15->300 16-> 16 17->117 17->122
## [46] 18-> 18 18->318 19->309 20-> 20 20->181 21-> 39 22-> 22 22->169 23->262
## [55] 24-> 24 24-> 73 25->311 26-> 26 26-> 95 27-> 97 27->191 27->192 28-> 28
## [64] 28-> 29 28-> 30 28-> 31 28->175 28->187 28->350 29-> 67 29-> 68 29->279
## [73] 30-> 49 30->315 31->184 32-> 68 33-> 33 33->127 34->358 35-> 35 35-> 39
## [82] 35->119 35->180 35-> 36 36-> 56 36-> 69 36->343 37->337 37->289 37->360
## + ... omitted several edges
```

```r
# Check that network is weighted:
is_weighted(map)
```

```
## [1] TRUE
```

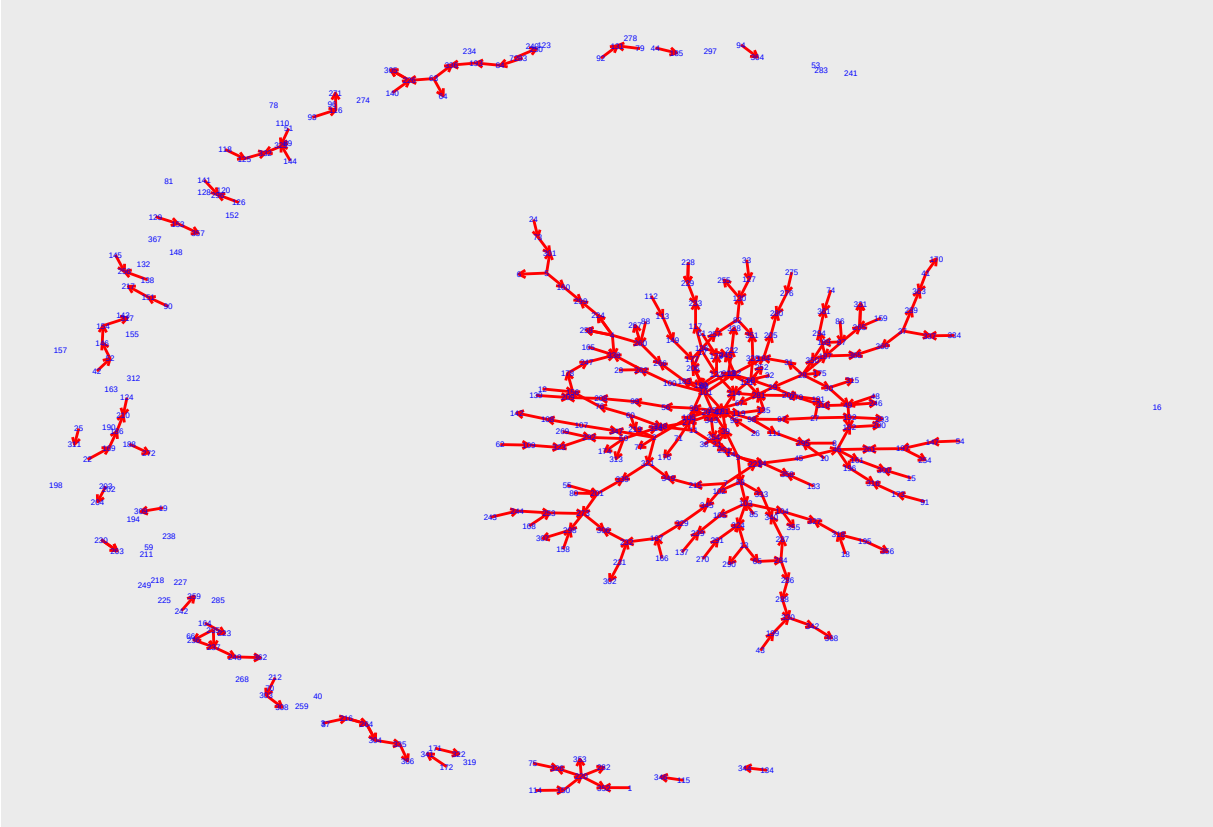
```r
# Obtain the edge weights:
head(E(map)$weight)
```

```
## [[1]]
##   [1] 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 3 1 2 1 2 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 3 1 1 1 1 1 1 1 1 1
##  [75] 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 1 2 1 1 1 1 1 1 1 2 1 2 1 1 1 2 2 2
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1
## [223] 1 1 1 1 1 1 2 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [260] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 2 2 1 1 2 1 1 2 1 1 1 1 1
## [297] 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 1 1 1
## [334] 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [371] 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 1 1 1 1
## [408] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1
##
## [[2]]
##   [1] 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 3 1 2 1 2 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 3 1 1 1 1 1 1 1 1 1
##  [75] 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 1 2 1 1 1 1 1 1 1 2 1 2 1 1 1 2 2 2
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1
## [223] 1 1 1 1 1 1 2 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [260] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 2 2 1 1 2 1 1 2 1 1 1 1 1
## [297] 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 1 1 1
## [334] 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [371] 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 1 1 1 1
## [408] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1
##
## [[3]]
##   [1] 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 3 1 2 1 2 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 3 1 1 1 1 1 1 1 1 1
##  [75] 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 1 2 1 1 1 1 1 1 1 2 1 2 1 1 1 2 2 2
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1
## [223] 1 1 1 1 1 1 2 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [260] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 2 2 1 1 2 1 1 2 1 1 1 1 1
## [297] 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 1 1 1
## [334] 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [371] 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 1 1 1 1
## [408] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1
##
## [[4]]
##   [1] 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 3 1 2 1 2 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 3 1 1 1 1 1 1 1 1 1
```

```
##  [75] 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 1 2 1 1 1 1 1 1 1 2 1 2 1 1 1 2 2 2
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1
## [223] 1 1 1 1 1 2 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [260] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 2 2 1 1 2 1 1 2 1 1 1 1
## [297] 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 1 1 1
## [334] 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [371] 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 1 1 1 1
## [408] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1
##
## [[5]]
##   [1] 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 3 1 2 1 2 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 3 1 1 1 1 1 1 1 1 1
##  [75] 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 1 2 1 1 1 1 1 1 1 2 1 2 1 1 1 2 2 2
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1
## [223] 1 1 1 1 1 2 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [260] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 2 2 1 1 2 1 1 2 1 1 1 1
## [297] 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 1 1 1
## [334] 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [371] 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 1 1 1 1
## [408] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1
##
## [[6]]
##   [1] 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 3 1 2 1 2 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 3 1 1 1 1 1 1 1 1 1
##  [75] 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 1 2 1 1 1 1 1 1 1 2 1 2 1 1 1 2 2 2
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1
## [223] 1 1 1 1 1 2 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [260] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 2 2 1 1 2 1 1 2 1 1 1 1
## [297] 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 1 1 1
## [334] 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [371] 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 1 1 1 1
## [408] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1
```

In order to describe the network, I first found the summary of the structure of the network. With this, the network can be described as D-W- 368 441. This means that it is a directed and weighted network with 368 nodes and 441 edges. To confirm this summary, I obtained the number of nodes and number of edges, which in fact were 368 and 441, respectively. I then obtained a list of the nodes and a list of the edges to ensure that the network looked correct. Then I double checked that the network was weighted, which returned TRUE. I also took a look at the weights of the edges, which aren't a measure of distance in km, but rather a number that is either 1, 2, or 3.

```
# Create visualization of the network:
ggraph(map, weights=NA, layout="kk") + geom_edge_link(color="red", arrow = arrow(length = unit(1, 'mm'))
```

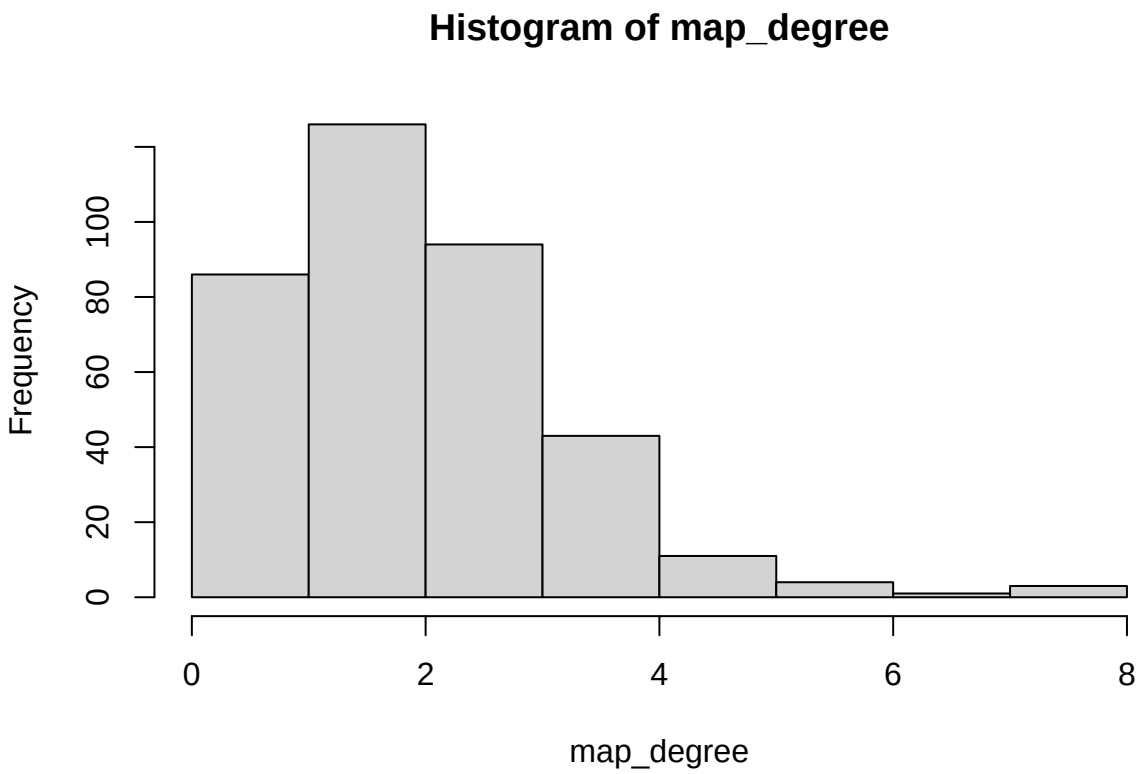


To visualize my network plot, I used ggraph. I made the edge colors red and added arrows for the directions of the edges. Instead of circular nodes, I used the station index number of each node. I colored them blue to stand out from the red. I chose these colors to represnet the Union Jack flag. Since this network is extremely large and complex, it is difficult to see on a laptop screen, but would be more easily read and interpreted on a larger screen. The `kk` layout was best for this network since after testing many different layouts, the `kk` layout arranged the network in a way in which the different nodes and edges were most visible and legible. While this visualization is still a little bit difficult to interpret on a small screen, this was the best layout that I could determine and would be much easier to view on a larger screen. From the graphical representation of the network, we can see that there are some routes between stations that only connect between 2 or 3 stations. In the larger network, we can see some of London's different train lines, which connect many stations. Additionally, there seems to be some stations in the middle of the network from which many routes branch out. These nodes are most likely the most centrally located in London or are some of the most high volume stations.

```
# Density analysis-obtain edge density:
map_density <- edge_density(map)
map_density
```

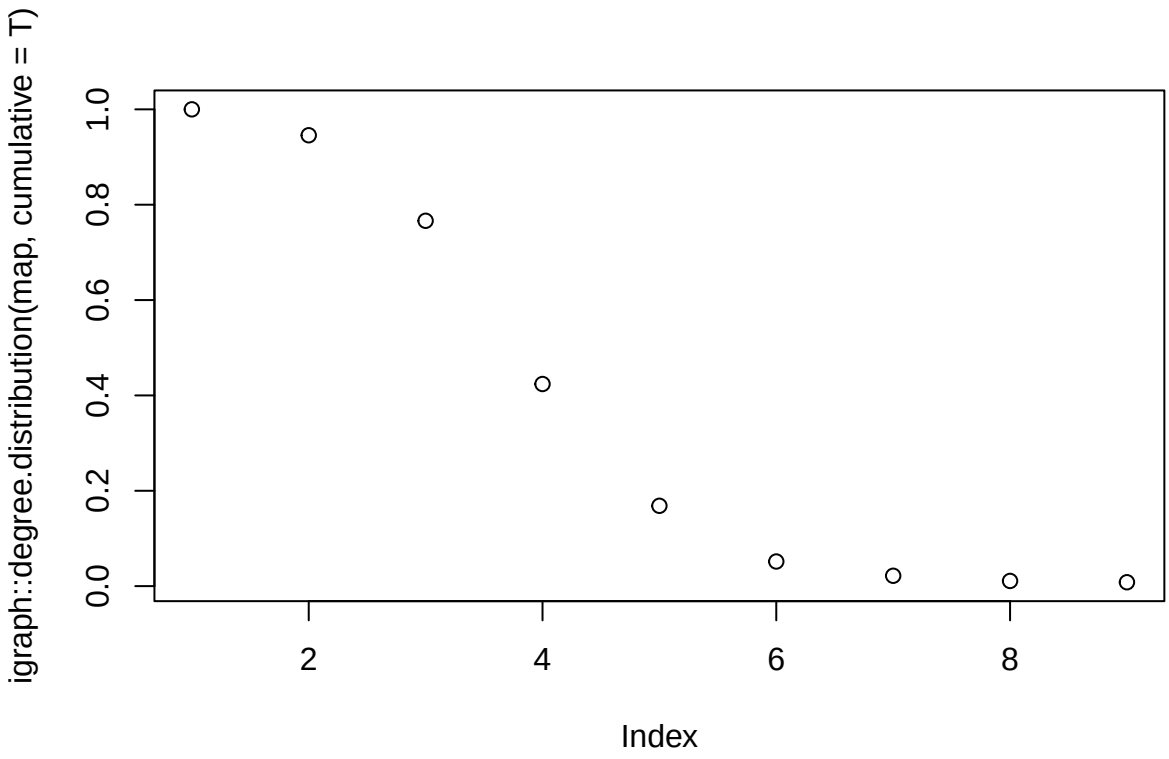
```
## [1] 0.003265312
```

```
# Centrality analysis-obtain degree centrality:
map_degree <- igraph::degree(map)
# Visualize degree centrality with histogram:
hist(map_degree)
```




```
# Visualize degree centrality with scatter plot:
plot(igraph::degree.distribution(map, cumulative = T))
```

```
## Warning: ‘degree.distribution()‘ was deprecated in igraph 2.0.0.
## i Please use ‘degree_distribution()‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```




```
# Create table for degree centrality to find the most central stations:
degree_table <- igraph::degree(map) %>% as.data.frame() %>% tibble::rownames_to_column("node") %>% dply
head(degree_table)
```

```
##             station node degree centrality
## 1       bakerstreet   28                 8
## 2         greenpark   68                 8
## 3          waterloo  181                 8
## 4        earlscourt   46                 7
## 5 willesdenjunction    4                 6
## 6              bank   35                 6
```

```
# Distances analysis:
# Obtain path:
map_path <- igraph::mean_distance(map, weights=NA)
map_path
```

```
## [1] 2.019841
```

```
# Obtain geodesic distance:
map_geodesic <- igraph::distance_table(map)
map_geodesic
```

```
## $res
```

```
## [1] 313 217 145  64  14   2   1
##
## $unconnected
## [1] 134300
```

```
# Obtain diameter:
map_diameter <- igraph::diameter(map, weights=NA)
map_diameter
```

```
## [1] 7
```

```
igraph::get_diameter(map, weights=NA)
```

```
## + 8/368 vertices, from e595339:
## [1]  13  65 284 286 288 330 342 368
```

```
# Partitions analysis:
# Obtain components:
map_strong_components <- igraph::components(map, mode="strong")
# Obtain number of clusters:
map_strong_components$no
```

```
## [1] 368
```

```
# Obtain sizes of the clusters:
map_strong_components$csize
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [260] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [297] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [334] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
# Obtain cluster id to which each vertex belongs:
map_strong_components$membership
```

```
##   [1] 367 359 353 345 341 344 336 334 312 311 298 294 284 283 281 280 274 272
##  [19] 270 269 268 264 262 260 258 256 248 223 240 234 230 222 218 332 215 300
##  [37] 210 209 324 208 206 201 199 197 191 192 190 189 236 184 182 202 181 177
##  [55] 171 307 170 313 169 167 166 164 159 163 286 158 247 243 301 157 155 153
##  [73] 261 152 150 168 366 149 147 146 145 143 140 141 139 134 135 133 131 128
##  [91] 126 125 122 120 257 119 253 254 308 115 116 323 316 320 318 365 112 113
## [109] 165 111 255 107 108 102 100 123 278  98 216  97  96 275  95  94  99  92
## [127] 219  91  88 220 148  87  86  84 217  83  82  80  79  78  77 114  76  75
## [145]  74 203 178  73 109 103 129  72  89 204  71 295  70  69  68 343 196 238
```

```
## [163]  67  65  64  60  61  58 265 207  56  54 127 188 229 156 110 296 277 276
## [181] 325 117 306 231 198 266 225  52 138  51 251 249 142  50  48 194 179  47
## [199] 200 239 172  46  44  45 232 305 252 309 310 267  43  40 302 330 340 354
## [217] 130  39 364 104 338 303  66 351  38 161  37  35  36  33  31  63 279  30
## [235]  25  26  27  24 319  23  22  20  18  19 118 237 297  28  17 154 339 246
## [253]  59 180 221  81 144 350  16 347 193 263  34 355 136 349 348  15  14  12
## [271] 124  53 173  11   9  10 328   8 241 233 331 106   7 287   6 289 288 290
## [289] 213 285  13 186 250 227 335  93   5 175 352 282 342  32  41 356 162 174
## [307] 176  42 271 242 259   4 185 329 235 195 205 273   3 244 361  57 187 317
## [325] 183 151 322 304  62 291 137 132 314   2 357 160 212 346 363 315  55 292
## [343] 326  85 337 101 360 299 362 224 245 368 105 327 321  49  90 333  21 211
## [361] 228  29 214 121 226 358   1 293
```

In order to describe the network quantitatively, I analyzed the network's density, centrality, distances, and partitions.

Density: The density of this network is 0.003265312. This means that this network has a low density and is connected by a low number of edges. This makes sense for a train network since the most efficient train network would connect all of the stations with the least amount of routes possible.

Centrality: I used degree centrality to analyze the network's centrality. Degree centrality is the number of adjacent edges for each node. I created a histogram to view the distribution of degree centrality. The degree centrality is low, with most nodes having 1-3 adjacent edges. This makese sense since London's train system is constructed with lines which connected one station to the next in a linear manner. We see this same trend in degree centrality in the scatter plot, as well. The number of nodes decreases as degree centrality increases, meaning most nodes have low degree centralities. I also wanted to find the most connected stations, so I created a table of nodes with their station names and degree centralities. The stations with the highest degree centralities and are the most connected to other stations are Baker Street, Green Park, and Waterloo. They each have a degree centrality of 8. These stations all make sense to be the most interconnected with other stations. Baker Street station is one of the original Underground stations and has accumulated many routes since 1863. Green Park is located in an area with many London landmarks, such as Buckingham Palace, Green Park, and The Mall so many tourists would need connections to this station from many others around the city. Waterloo is Britain's largest train station, connected to many other stations. Knowing these facts, we can confirm that it makes sense that these three stations would all have the highest degree centralities.

Distances: First, I determined the path, which is the set of edges from one node to another that never repeat a node or edge, using `mean_distance()`. I found this to be 2.019841. Then, I found the geodesic distance using `distance_table()`, which is the number of edges on the shortest path between two nodes. The `unconnected` value from this result is 134300, which is the number of pairs for which the starting station is not reachable from the ending station. The `res`result is 313 217 145 64 14 2 1, which is a numeric vector which is a histogram of the distances. I also determined the map diameter to be 7, which is the longest geodesic distance in the network.

Partitions: I then evaluated the partitions of the network by finding the components, which are the largest clusters of vertices where every vertex can reach every other vertex. I used the strong method, which takes into account the direction of the edges.The number of clusters for this network is 368. Then I found the sizes of the clusters and the cluster to which each vertex belongs.

Interpretation and Insight

Drawing conclusions from these results, this network is complex and extensive yet logical. It is weighted and directed, with each node as a train station and each edge as a train route. It is a low density network, meaning that the train routes are very efficient at connecting the many different stations around London.

The degree centralities are mainly 1-3, meaning that stations mainly have 1-3 adjacent routes. Additionally, with some background research, I found that some of the busiest and most prominently located stations in London have the highest degree centralities, which makes sense given the nature of those stations. I also found the longest number of edges on the shortest path between two nodes, the geodesic distance, to be 7 edges, which is the shortest way to make the longest trip between 8 stations on one train line. Additionally, I found that the number of components is equal to the number of nodes, meaning that every station which is connected to other stations can be reached from any other station that is also connected to any other station, which is important for a city to have since traingoers must be able to reach any other part of the city from any given starting point within the city. Overall, all of these findings can answer my two research questions. London's train network is efficient and well-connected, enabling riders to get to destinations quickly while also being able to reach any destination from any starting point. I think these are good measures of a train system, and ones which show thoughtful design and thorough development of the network. As for the most connected stations within this network, they are Baker Street, Green Park, and Waterloo, all of which are busy stations that are positioned close to sites of interest in London.

A new research question that the visualization of my network now motivates is: How does the KK layout of London's train network differ from its geographic layout? I think it would be interesting to map the network of stations and routes overlaid on a map of London in their geographic locations and compare that layout to the layout of the network which I produced. I think that this could provide some further insight into why the network looks the way it does in my visualization and if the stations that have the highest degree of centrality in my network also have the highest degree of centrality in the geographic layout.

The network does look similar to what I expected because there are a few key busy stations which are at the center of the network from which many of the train routes radiate out from. However, I was surprised to find the smaller clusters of stations with which only 2 or 3 stations are connected to each other and not connected to the wider network of the other train routes. These smaller networks of stations within in the wider map could be local stations on the outskirts of London which are only connected to each other and serve communities outside of the center of the city. I was also surprised to find a few stations that are not connected to any other station. I theorize that these stations might be out of service but are still included in the data since they were connected to other stations at some point in time.