Midterm: Predicting Employee Churn with Network Analytics

Emma Kruis

2020-02-01

Instructions

This script reviews *Network Analytics* as part of the *Midterm Review*. You will use content from the lecture and assignment materials on *Network Analytics* to complete this script. You will *copy and paste* relevant code from those files into this script and answer the associated questions for each task. You will respond to questions in each section after executing relevant code to answer a question. You will submit this script to its *Submissions* folder on *D2L* as part of the *Midterm Review*. For this script, you will submit *two* files:

1. this completed *R Markdown* script, and
2. as a first preference, a *PDF* (if you already installed `TinyTeX` properly), as a second preference, a *Microsfot Word* (if your computer has *Microsoft Word*) document, or, as a third preference, an *HTML* (if you did *not* install `TinyTeX` properly and your computer does *not* have *Microsoft Word*) file to *D2L*.

For the *Midterm Review*, create the project directory: *~/mgt_592/assignments/midterm_review*. Convert your project directory into a formal *R Project* directory by going to the *File* menu in *RStudio*, selecting *New Project...*, choosing *Existing Directory*, and going to your *~/mgt_592/assignments/midterm_review* folder to select it as the top-level directory for this **R Project**.
The project directory should contain the following folders: *scripts*, *data*, and *plots*. Store this script in the *scripts* folder and the relevant data in the *data* folder.

Global Settings

The first code chunk sets the global settings for the remaining code chunks in the document. Do *not* change anything in this code chunk.

Task 1: Load Libraries

For this task, you will load the libraries you need for this script.

Task 1.1

In this code chunk, load the following packages:

1. **here**,
2. **tidyverse**,
3. **tidymodels**,
4. **corrr**,
5. **igraph**,

6. **tidygraph**, and
7. **ggraph**.

Make sure you installed these packages before loading the libraries.

You will use functions from these packages to complete this script.

```
### load libraries for use in current working session
## here for project work flow
library(here)
```

```
## here() starts at /Users/emmakruis/Library/Mobile Documents/com~apple~CloudDocs/year_2/WQ21/mgt_592/a
```

```
## tidyverse for data manipulation and plotting
## loads eight different libraries simultaneously
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.0     v dplyr   1.0.5
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1

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

```
## tidymodels for modeling
library(tidymodels)
```

```
## -- Attaching packages -------------------------------------- tidymodels 0.1.2 --

## v broom     0.7.5      v recipes   0.1.15
## v dials     0.0.9      v rsample   0.0.9
## v infer     0.5.4      v tune      0.1.3
## v modeldata 0.1.0      v workflows 0.2.2
## v parsnip   0.1.5      v yardstick 0.0.7

## -- Conflicts ----------------------------------------- tidymodels_conflicts() --
## x scales::discard() masks purrr::discard()
## x dplyr::filter()   masks stats::filter()
## x recipes::fixed()  masks stringr::fixed()
## x dplyr::lag()      masks stats::lag()
## x yardstick::spec() masks readr::spec()
## x recipes::step()   masks stats::step()
```

```
## corrr for correlation matrices
library(corrr)

## igraph for analyzing networks
library(igraph)
```

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

## The following objects are masked from 'package:dials':
##
##     degree, neighbors

## 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
```

```
## tidygraph for graph data tables
library(tidygraph)
```

```
##
## Attaching package: 'tidygraph'

## The following object is masked from 'package:igraph':
##
##     groups

## The following object is masked from 'package:stats':
##
##     filter
```

```
## ggraph for plotting networks
library(ggraph)
```

Task 2: Import Data

For this task, you will import the data file: **employee_net.rdata**.

Task 2.1

Use **load()** and **here()** to import **employee_net.rdata** from the **data** folder in the project directory. Preview the **nodes** and **edges** data tables via **glimpse()**. You will work with the complete network for this assignment.

```
### import data object
## use load() and here() to import the data file
load(
  ## use here() to locate file in our project directory
  here("data", "employee_net.rdata")
)

## preview nodes
glimpse(nodes)
```

```
## Rows: 1,000
## Columns: 2
## $ id    <chr> "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "~
## $ churn <fct> No, No, No, No, No, No, No, No, No, No, No, No, No, No, No, No, ~
```

```
## preview edges
glimpse(edges)
```

```
## Rows: 196,966
## Columns: 2
## $ from <chr> "1", "1", "3", "3", "3", "4", "1", "3", "4", "7", "1", "3", "5", ~
## $ to   <chr> "3", "4", "4", "5", "6", "6", "7", "7", "7", "8", "9", "9", "9", ~
```

Task 3: Table Graph

For this task, you will create a *table graph* and visualize the employee network.

Task 3.1

Create a *table graph* named **employee_tg** using the **nodes** and **edges** data tables. Set the *node key* to **id** and create an *undirected* graph.

Next, update the *node* data table in **employee_tg** by computing **edge_id**, which is equivalent to the *row numbers*. Continue with a chained command to update the *edge* data table in **employee_tg** by computing **churn_type**. The **churn_type** variable should consist of three categories named **cc**, **ss**, and **cs** to indicate whether the nodes for an edge both *left*, both *remain*, or a mixture of the two, respectively.

Print the updated **employee_tg**.

```
### create graph table
## save as object
employee_tg <- tbl_graph(
  # nodes
  nodes = nodes,
  # edges
  edges = edges,
```

```r
  # undirected
  directed = FALSE,
  # node id
  node_key = "id"
)



## overwrite table graph
employee_tg <- employee_tg %>%
  ## activate nodes
  activate(nodes) %>%
  ## add variable
  mutate(
    # add edge id
    edge_id = row_number()
  ) %>%
  ## activate edges
  activate(edges) %>%
  ## add variable
  mutate(
    # edge color by churn mix
    churn_type = case_when(
      # both churn
      .N()$churn[from] == "Yes" & .N()$churn[to] == "Yes" ~ "cc",
      # both stay
      .N()$churn[from] == "No" & .N()$churn[to] == "No" ~ "ss",
      # churn and stay
      .N()$churn[from] == "Yes" & .N()$churn[to] == "No" ~ "cs",
      # stay and churn
      .N()$churn[from] == "No" & .N()$churn[to] == "Yes" ~ "cs"
    )
  )

## print
employee_tg
```

```
## # A tbl_graph: 1000 nodes and 196966 edges
## #
## # An undirected simple graph with 1 component
## #
## # Edge Data: 196,966 x 3 (active)
##    from    to churn_type
##   <int> <int> <chr>
## 1     1     3 ss
## 2     1     4 ss
## 3     3     4 ss
## 4     3     5 ss
## 5     3     6 ss
## 6     4     6 ss
## # ... with 196,960 more rows
## #
## # Node Data: 1,000 x 3
```

```
##   id    churn edge_id
##   <chr> <fct>   <int>
## 1 1     No          1
## 2 2     No          2
## 3 3     No          3
## # ... with 997 more rows
```

Task 3.2

Set the random seed of your computer to **1736**. Create a new data table named **employee_nodes_samp** from **employee_tg**. The **id** variable should be removed and the **edge_id** variable should be converted to a *character* variable. The **employee_nodes_samp** data table should consist of **100** randomly sampled nodes from **employee_tg** using **slice_sample()**.

Next, create a new data table named **employee_edges_samp** from **employee_tg**. Convert **from** and **to** to *character* variables. Filter the rows by **from** and **to** being in **employee_nodes_samp$edge_id**.

Then, create a new *table graph* named **employee_tg_samp** from **employee_nodes_samp** and **employee_edges_samp**. Set the *node key* to **edge_id** and create an *undirected* graph.

Print **employee_tg_samp**.

```r
### sample from employee_tg
## set seed
set.seed(1736)

### sample nodes
## save as object
employee_nodes_samp <- employee_tg %>%
  ## activate edges
  activate(nodes) %>%
  ## tibble
  as_tibble() %>%
  ## alter variables
  mutate(
    # remove old id variable
    id = NULL,
    # change to character
    edge_id = as.character(edge_id)
  ) %>%
  ## sample
  slice_sample(n = 100)



### sample edges
## save as object
employee_edges_samp <- employee_tg %>%
  ## activate edges
  activate(edges) %>%
  ## tibble
  as_tibble() %>%
  ## alter variables
  mutate(
    # from
```

```r
    from = as.character(from),
    # to
    to = as.character(to)
  ) %>%
  ## filter for rows
  filter(
    # from
    from %in% employee_nodes_samp$edge_id,
    # to
    to %in% employee_nodes_samp$edge_id
  )



### create graph table
## save as object
employee_tg_samp <- tbl_graph(
  # nodes
  nodes = employee_nodes_samp,
  # edges
  edges = employee_edges_samp,
  # undirected
  directed = FALSE,
  # node id
  node_key = "edge_id"
)



## print
employee_tg_samp
```

```
## # A tbl_graph: 100 nodes and 1949 edges
## #
## # An undirected simple graph with 1 component
## #
## # Node Data: 100 x 2 (active)
##   churn edge_id
##   <fct> <chr>
## 1 No    61
## 2 No    10
## 3 Yes   285
## 4 No    604
## 5 No    459
## 6 Yes   893
## # ... with 94 more rows
## #
## # Edge Data: 1,949 x 3
##    from    to churn_type
##   <int> <int> <chr>
## 1    32    37 ss
## 2    25    37 ss
## 3     2    32 ss
```

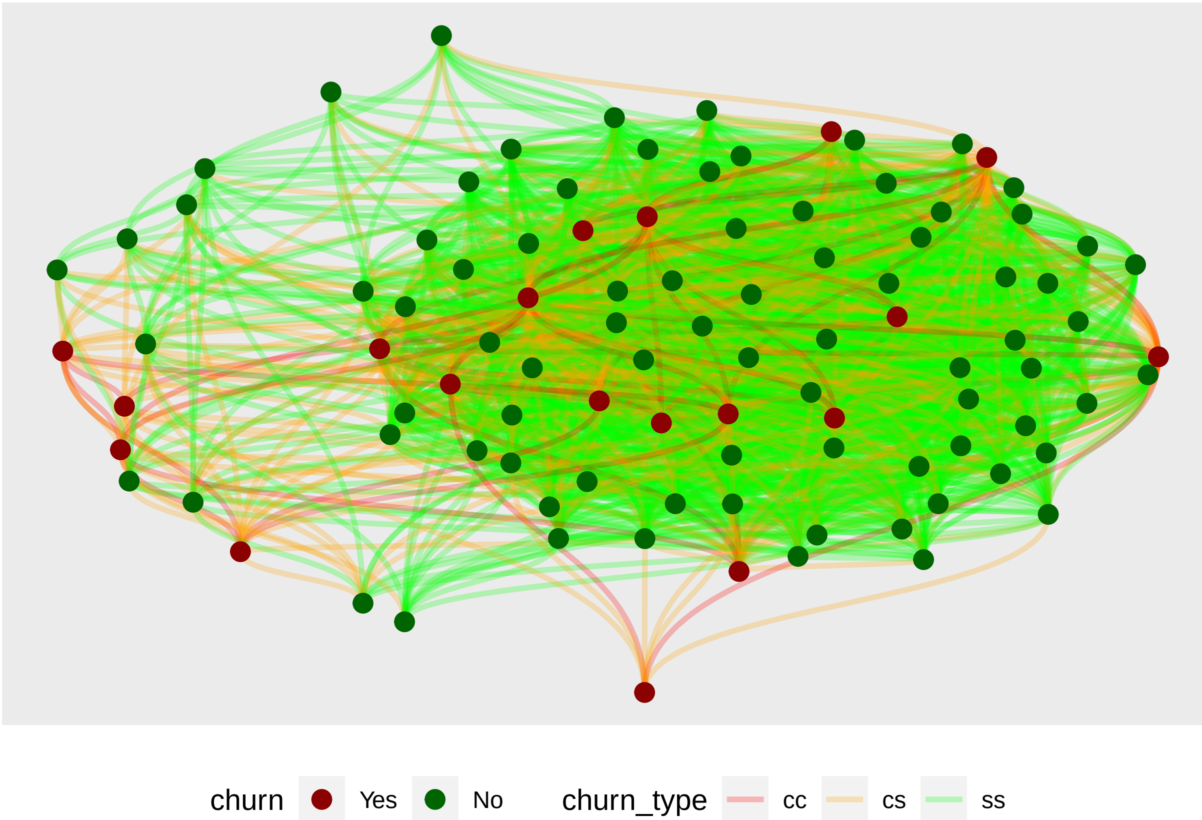
```
## # ... with 1,946 more rows
```

Task 3.3

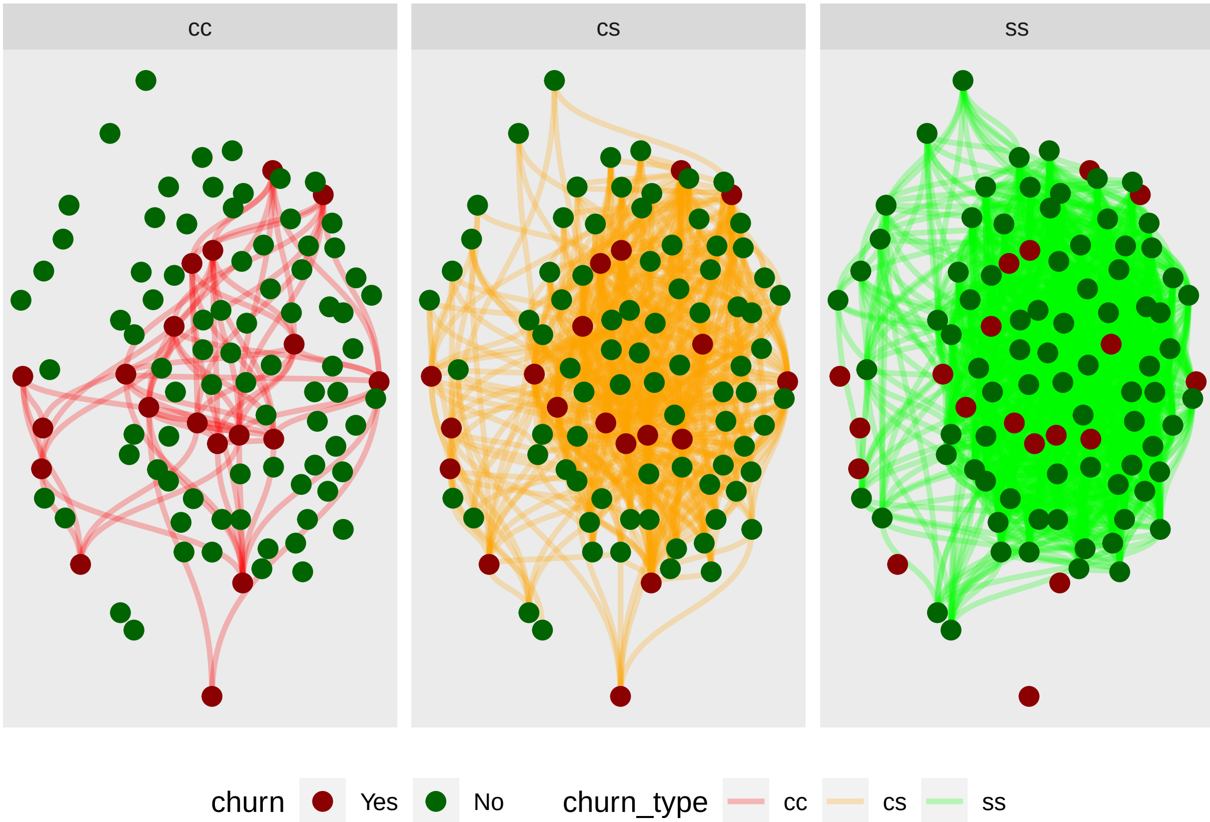
Create a plot named **employee_net_samp_plot** using **ggraph()** and **employee_tg_samp** to view a representative sample of the network. Set the *layout* to **kk**. Choose **geom_node_point()** as the node geometry and set **color** to **churn** and **size** to **3**. Choose **geom_edge_diagonal()** as the edge geometry and set **color** to **churn_type**, **alpha** to **0.25**, and **width** to **1**. Add **scale_color_manual()** to set the node colors to **darkred** and **darkgreen**. Add **scale_edge_color_manual()** to set the *churn type* colors to **red**, **orange**, and **green**. Place the legend at the bottom. Print the plot.

Create another plot named **employee_net_samp_facet_plot** using **ggraph()** and **employee_tg_samp** to view a representative sample of the network. Set the *layout* to **kk**. Choose **geom_node_point()** as the node geometry and set **color** to **churn** and **size** to **3**. Choose **geom_edge_diagonal()** as the edge geometry and set **color** to **churn_type**, **alpha** to **0.25**, and **width** to **1**. Add **scale_color_manual()** to set the node colors to **darkred** and **darkgreen**. Add **scale_edge_color_manual()** to set the *churn type* colors to **red**, **orange**, and **green**. Add **facet_edges()** to facet by **churn_type**. Place the legend at the bottom. Print the plot.

```r
### create plot
## call network plot
employee_net_samp_plot <- ggraph(
  # data
  employee_tg_samp,
  # layout
  layout = "kk"
) +
  ## add edges
  geom_edge_diagonal(aes(color = churn_type), alpha = 0.25, width = 1) +
  ## add nodes
  geom_node_point(aes(color = churn), size = 3) +
  ## scale nodes
  scale_color_manual(values = c("darkred", "darkgreen")) +
  ## scale edge colors
  scale_edge_color_manual(values = c("red", "orange", "green")) +
  ## legend on bottom
  theme(legend.position = "bottom")


## display plot
employee_net_samp_plot
```




```r
### create plot
## call network plot
employee_net_samp_facet_plot <- ggraph(
  # data
  employee_tg_samp,
  # layout
  layout = "kk"
) +
  ## add edges
  geom_edge_diagonal(aes(color = churn_type), alpha = 0.25, width = 1) +
  ## add nodes
  geom_node_point(aes(color = churn), size = 3) +
  ## scale nodes
  scale_color_manual(values = c("darkred", "darkgreen")) +
  ## scale edge colors
  scale_edge_color_manual(values = c("red", "orange", "green")) +
  ## facet edges
  facet_edges(~ churn_type) +
  ## legend on bottom
  theme(legend.position = "bottom")

## display plot
employee_net_samp_facet_plot
```



Task 4: Calculate Measures

For this task, you will calculate network measures on the employee network.

Task 4.1

First, update **employee_tg** by computing the following network measures in the appropriate data tables using the relevant functions:

1. first-order neighborhood size,
2. second-order neighborhood size,
3. average degree of neighborhood,
4. number of triangles,
5. local transitivity,
6. betweenness centrality,
7. closeness centrality,
8. eigenvector centrality,
9. PageRank centrality,
10. edge betweenness centrality,
11. average edge PageRank centrality,

Print the updated version of **employee_tg** to confirm calculations.

Second, updated **employee_tg** by computing the following node neighborhood measures using **map_local_dbl()** with a *user-defined function* appropriately:

1. number of churners in first-order neighborhood,
2. number of non-churners in first-order neighborhood,
3. churn probability of first-order neighborhood,
4. number of churners in second-order neighborhood,
5. number of non-churners in second-order neighborhood, and
6. churn probability of second-order neighborhood.

Remain patient with the calculations. Print *wide* the updated version of **employee_tg** to examine calculations.

Questions 4.1: Answer these questions: (1) What is the *closeness centrality* for the *first* listed node? (2) What is the *average neighborhood degree* for the *second* listed node? (3) What is the *number of non-churners* in the *first-order neighborhood* for the *first* listed node? (4) What is the *churn probability* in the *second-order neighborhood* for the *second* listed node?

Responses 4.1: *(1) 0.650 (2) 334 (3) 401 (4) 0.140.*

```
### calculate attributes
## overwrite table graph
employee_tg <- employee_tg %>%
  ## activate nodes
  activate(nodes) %>%
  ## add variables
  mutate(
    #1 first-order neighborhood of node
    degree_1 = local_size(order = 1, mindist = 1),
    #2 second-order neighborhood of node
    degree_2 = local_size(order = 2, mindist = 2),
    #3 neighborhood average degree
    neigh_avg_deg = local_ave_degree(),
    #4 number of triangles
    n_triangle = local_triangles(),
    #5 local transitivity
    loc_trans = local_transitivity(),
    #6 betweenness centrality
    between = centrality_betweenness(directed = FALSE, normalized = TRUE),
    #7 closeness centrality
    closeness = centrality_closeness(normalized = TRUE),
    #8 eigenvector centrality
    eigen = centrality_eigen(),
    #9 PageRank
    page_rank = centrality_pagerank(directed = FALSE)
  ) %>%
  ## activate edges
  activate(edges) %>%
  ## add variables
  mutate(
    #10 edge betweenness
    edge_between = centrality_edge_betweenness(directed = FALSE),
    #11 mean PageRank
    mean_page_rank = (.N()$page_rank[from] + .N()$page_rank[to]) / 2
  )

## print
employee_tg
```

```
## # A tbl_graph: 1000 nodes and 196966 edges
## #
## # An undirected simple graph with 1 component
## #
## # Edge Data: 196,966 x 5 (active)
##    from    to churn_type edge_between mean_page_rank
##   <int> <int> <chr>             <dbl>          <dbl>
## 1     1     3 ss                 3.77        0.00112
## 2     1     4 ss                 3.51        0.00111
## 3     3     4 ss                 3.66        0.00110
## 4     3     5 ss                 3.69        0.00111
## 5     3     6 ss                 3.81        0.00108
## 6     4     6 ss                 3.59        0.00106
## # ... with 196,960 more rows
## #
## # Node Data: 1,000 x 12
##   id    churn edge_id degree_1 degree_2 neigh_avg_deg n_triangle loc_trans
##   <chr> <fct>   <int>    <dbl>    <dbl>         <dbl>      <dbl>     <dbl>
## 1 1     No          1      460      539          429.      49561     0.469
## 2 2     No          2      136      863          334.       2702     0.294
## 3 3     No          3      446      553          425.      45944     0.463
## # ... with 997 more rows, and 4 more variables: between <dbl>, closeness <dbl>,
## #   eigen <dbl>, page_rank <dbl>
```

```r
### compute adjacency-based attributes
## call graph table
employee_tg <- employee_tg %>%
  ## activate nodes
  activate(nodes) %>%
  ## add variables
  mutate(
    ##1 first-order churn neighbors
    churn_neigh_1 = map_local_dbl(
      ## first order neighborhood
      order = 1,
      ## minimum distance
      mindist = 1,
      ## function
      .f = function(neighborhood, ...) {
        # sum of churn
        sum(
          # tibble
          as_tibble(neighborhood, active = "nodes")$churn == "Yes"
        )
      }
    ),
    ##2 first-order non-churn neighbors
    non_churn_neigh_1 = map_local_dbl(
      ## first order neighborhood
      order = 1,
      ## minimum distance
      mindist = 1,
      ## function
      .f = function(neighborhood, ...) {
```

```r
        # sum of churn
        sum(
          # tibble
          as_tibble(neighborhood, active = "nodes")$churn == "No"
        )
      }
    ),
    ##3 first-order neighbor churn probability
    neigh_churn_prob_1 = churn_neigh_1 / (churn_neigh_1 + non_churn_neigh_1),
    ##4 second-order churn neighbors
    churn_neigh_2 = map_local_dbl(
      ## second order neighborhood
      order = 2,
      ## minimum distance
      mindist = 2,
      ## function
      .f = function(neighborhood, ...) {
        # sum of churn
        sum(
          # tibble
          as_tibble(neighborhood, active = "nodes")$churn == "Yes"
        )
      }
    ),
    ##5 second-order non-churn neighbors
    non_churn_neigh_2 = map_local_dbl(
      ## second order neighborhood
      order = 2,
      ## minimum distance
      mindist = 2,
      ## function
      .f = function(neighborhood, ...) {
        # sum of churn
        sum(
          # tibble
          as_tibble(neighborhood, active = "nodes")$churn == "No"
        )
      }
    ),
    ##6 second-order neighbor churn probability
    neigh_churn_prob_2 = churn_neigh_2 / (churn_neigh_2 + non_churn_neigh_2)
  )



## call data
employee_tg %>%
  ## nodes
  activate(nodes) %>%
  ## print wide
  print(width = Inf)
```

```
## # A tbl_graph: 1000 nodes and 196966 edges
```

```
## #
## # An undirected simple graph with 1 component
## #
## # Node Data: 1,000 x 18 (active)
##   id    churn edge_id degree_1 degree_2 neigh_avg_deg n_triangle loc_trans
##   <chr> <fct>   <int>    <dbl>    <dbl>         <dbl>      <dbl>     <dbl>
## 1 1     No          1      460      539          429.      49561     0.469
## 2 2     No          2      136      863          334.       2702     0.294
## 3 3     No          3      446      553          425.      45944     0.463
## 4 4     No          4      433      566          430       44224     0.473
## 5 5     No          5      448      551          433.      48063     0.480
## 6 6     No          6      420      579          430.      41544     0.472
##    between closeness eigen page_rank churn_neigh_1 non_churn_neigh_1
##      <dbl>     <dbl> <dbl>     <dbl>         <int>             <int>
## 1 0.000771     0.650 0.940  0.00114             59               401
## 2 0.000250     0.537 0.210  0.000482            28               108
## 3 0.000785     0.644 0.904  0.00111             51               395
## 4 0.000659     0.638 0.888  0.00108             45               388
## 5 0.000647     0.645 0.925  0.00111             58               390
## 6 0.000608     0.633 0.862  0.00105             48               372
##   neigh_churn_prob_1 churn_neigh_2 non_churn_neigh_2 neigh_churn_prob_2
##                <dbl>         <int>             <int>              <dbl>
## 1              0.128            90               449              0.167
## 2              0.206           121               742              0.140
## 3              0.114            98               455              0.177
## 4              0.104           104               462              0.184
## 5              0.129            91               460              0.165
## 6              0.114           101               478              0.174
## # ... with 994 more rows
## #
## # Edge Data: 196,966 x 5
##    from    to churn_type edge_between mean_page_rank
##   <int> <int> <chr>             <dbl>          <dbl>
## 1     1     3 ss                 3.77        0.00112
## 2     1     4 ss                 3.51        0.00111
## 3     3     4 ss                 3.66        0.00110
## # ... with 196,963 more rows
```

Task 4.2

Extract the *nodes* data table from **employee_tg** and save it as **employee_nodes**. Extract the *edges* data table from **employee_tg** and save it as **employee_edges**.

Create **employee_edges_agg** from **employee_edges** by:

1. pivoting longer by **from** and **to** and setting the names to **edge_ndx** and values to **edge_id**,
2. grouping by **edge_id**, and
3. summarizing **edge_between** and **mean_page_rank** with the **mean** function.

Update **employee_nodes** by applying **left_join()** with **employee_edges_agg** and joining by **edge_id**. Replace any missing values on the network measures with *zero* values. Print *wide* the updated version of **employee_nodes** to examine calculations.

```r
### extract nodes as a tibble
## save as object
employee_nodes <- employee_tg %>%
  ## activate nodes
  activate(nodes) %>%
  ## tibble
  as_tibble()



### extract edges as a tibble
## save as object
employee_edges <- employee_tg %>%
  ## activate nodes
  activate(edges) %>%
  ## tibble
  as_tibble()



### aggregate edges tibble
## save as object
employee_edges_agg <- employee_edges %>%
  ## make long
  pivot_longer(
    # columns to make long
    cols = c(from, to),
    # names
    names_to = "edge_ndx",
    # edge ids
    values_to = "edge_id"
  ) %>%
  ## group by id
  group_by(edge_id) %>%
  ## summarize variables
  summarize(
    # apply functions to variables
    across(
      # columns
      .cols = c(edge_between, mean_page_rank),
      # functions
      .fns = mean
    )
  )




### join nodes with edges
## overwrite nodes tibble
employee_nodes <- employee_nodes %>%
  ## left_join
  left_join(employee_edges_agg, by = "edge_id") %>%
```

```r
  ## replace missing values
  mutate(
    # apply functions to variables
    across(
      # columns
      .cols = degree_1:mean_page_rank,
      # functions
      .fns = ~ replace_na(.x, 0)
    )
  )
```

```r
## call data
employee_nodes %>%
  ## print wide
  print(width = Inf)
```

```
## # A tibble: 1,000 x 20
##    id    churn edge_id degree_1 degree_2 neigh_avg_deg n_triangle loc_trans
##    <chr> <fct>   <int>    <dbl>    <dbl>         <dbl>      <dbl>     <dbl>
##  1 1     No          1      460      539          429.      49561     0.469
##  2 2     No          2      136      863          334.       2702     0.294
##  3 3     No          3      446      553          425.      45944     0.463
##  4 4     No          4      433      566          430       44224     0.473
##  5 5     No          5      448      551          433.      48063     0.480
##  6 6     No          6      420      579          430.      41544     0.472
##  7 7     No          7      422      577          429.      41787     0.470
##  8 8     No          8      428      571          431.      43373     0.475
##  9 9     No          9      418      581          431.      41583     0.477
## 10 10    No         10      435      564          428.      44157     0.468
##     between closeness eigen page_rank churn_neigh_1 non_churn_neigh_1
##       <dbl>     <dbl> <dbl>     <dbl>         <dbl>             <dbl>
##  1 0.000771     0.650 0.940  0.00114             59               401
##  2 0.000250     0.537 0.210  0.000482            28               108
##  3 0.000785     0.644 0.904  0.00111             51               395
##  4 0.000659     0.638 0.888  0.00108             45               388
##  5 0.000647     0.645 0.925  0.00111             58               390
##  6 0.000608     0.633 0.862  0.00105             48               372
##  7 0.000635     0.634 0.863  0.00106             53               369
##  8 0.000628     0.636 0.880  0.00107             57               371
##  9 0.000584     0.632 0.860  0.00104             48               370
## 10 0.000710     0.639 0.887  0.00109             51               384
##    neigh_churn_prob_1 churn_neigh_2 non_churn_neigh_2 neigh_churn_prob_2
##                 <dbl>         <dbl>             <dbl>              <dbl>
##  1              0.128            90               449              0.167
##  2              0.206           121               742              0.140
##  3              0.114            98               455              0.177
##  4              0.104           104               462              0.184
##  5              0.129            91               460              0.165
##  6              0.114           101               478              0.174
##  7              0.126            96               481              0.166
##  8              0.133            92               479              0.161
```

```
## 9              0.115         101              480              0.174
## 10             0.117          98              466              0.174
##    edge_between mean_page_rank
##           <dbl>          <dbl>
## 1          3.84       0.00110
## 2          9.18       0.000681
## 3          3.99       0.00109
## 4          3.83       0.00108
## 5          3.67       0.00109
## 6          3.82       0.00106
## 7          3.87       0.00106
## 8          3.80       0.00107
## 9          3.78       0.00106
## 10         3.92       0.00108
## # ... with 990 more rows
```

Task 5: Predict Churn

For this task, you will fit logistic regression models to predict employee churn.

Task 5.1

Set the random seed of your computer to **1736**. Create a new object named **employee_nodes_split** using **initial_split()** applied to **employee_nodes**. Split the data into *80%* training and *20%* testing. Use **churn** as the stratification variable (i.e., set **strata = churn**).

Plot the correlations among the node network measures on the training data. Appropriately apply **training()**, **select()**, **correlate()**, **rearrange()**, **shave()**, and **rplot()** to **employee_nodes_split**. Alter the *theme* of the plot to make the x-axis labels angled at *45* degrees.

Examine the plot.

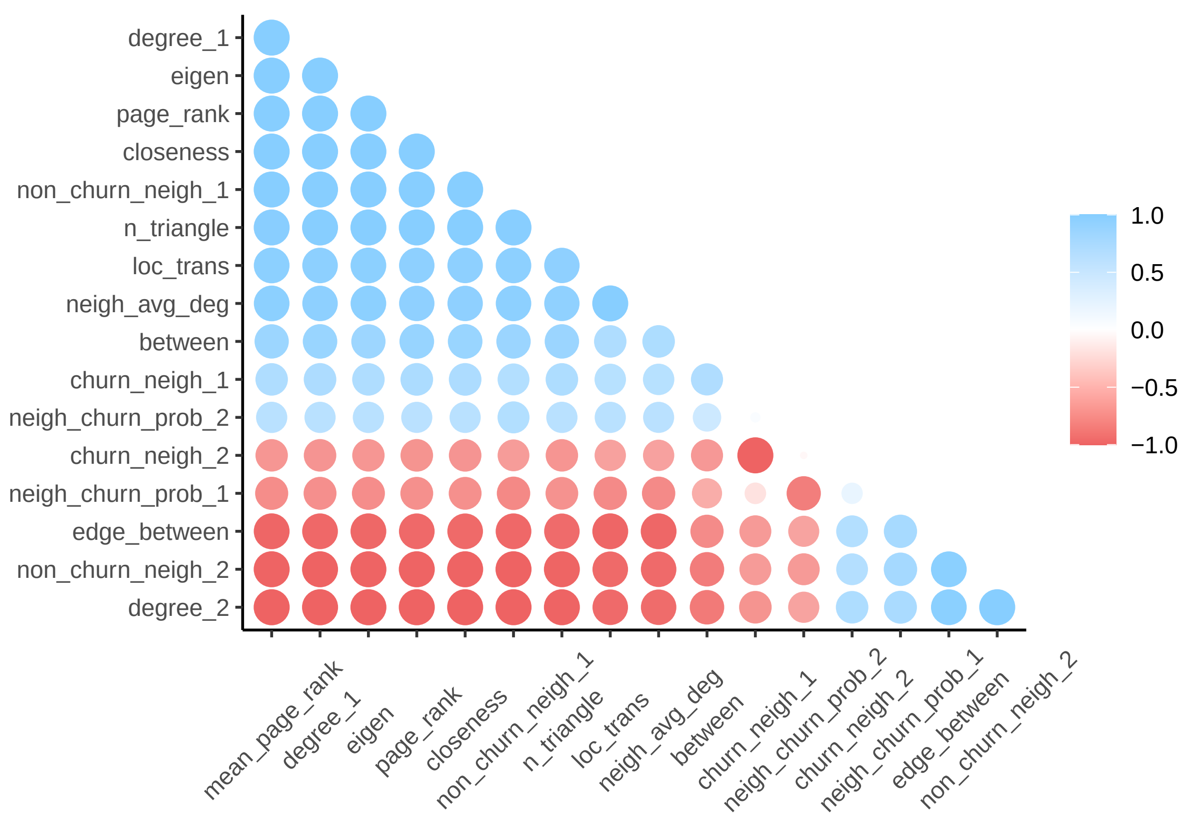
Questions 8.1: Answer these questions: (1) Is the correlation between **degree_1** and **loc_trans** *positive* or *negative*? (2) Is the correlation between **degree_1** and **degree_2** *high* or *low*?

Responses 8.1: *(1) positive (2) Its negative by high/strong.*

```
### create training and testing split
## set seed
set.seed(1736)




## create split
employee_nodes_split <- initial_split(
  # data
  employee_nodes,
  # proportion
  prop = 0.8,
  #stratification variable?
  strata = churn
)
```

```r
### examine correlations
## call data
employee_nodes_split %>%
  ## extract training data
  training() %>%
  ## select variables
  select(degree_1:mean_page_rank) %>%
  ## correlation matrix
  correlate() %>%
  ## rearrange by correlations
  rearrange() %>%
  ## shave upper triangle
  shave() %>%
  ## plot
  rplot() +
    ## theme
    theme(
      # adjust x-axis labels
      axis.text.x = element_text(angle = 45, vjust = 0.5)
    )
```

```
##
## Correlation method: 'pearson'
## Missing treated using: 'pairwise.complete.obs'

## Don't know how to automatically pick scale for object of type noquote. Defaulting to continuous.
```



Task 5.2

First, create a modeling *workflow* named **glm_wrkflw** for *logistic regression* using the **glm** engine.

Second, fit a *logistic regression* model on the *training* data of **employee_nodes_split** where **churn** is predicted by these network node measures: (1) **neigh_avg_deg**, (2) **loc_trans**, (3) **between**, (4) **closeness**, (5) **eigen**, (6) **page_rank**, (7) **neigh_churn_prob_1**, (8) **neigh_churn_prob_2**, and (9) **edge_between**. Save the model as **glm**.

Third, apply **predict()** to **glm** to calculate *probabilities* of **churn** in the *testing* data of **employee_nodes_split**. Bind the predictions with the *testing* data. Save the result as **glm_probs**.

Fourth, apply **roc_curve()** to **glm_probs**. Save the result as **glm_roc**.

Fifth, calculate the area under the receiver-operator characteristic (ROC) curve for **glm_probs** using **roc_auc()**.

Examine the results.

Questions 5.2: What is the area under the ROC curve for this model?

Responses 5.2: *0.696*.

```
### logistic regression workflow
## save as object
glm_wrkflw <- workflow() %>%
  ## add model
  add_model(
    # specify logistic regression
    logistic_reg() %>%
      # engine
      set_engine("glm")
  )



### first logistic regression model
## save as object
glm <- glm_wrkflw %>%
  ## add formula
  add_formula(
    # specify equation
    churn ~ neigh_avg_deg + loc_trans + between +
      closeness + eigen + page_rank + neigh_churn_prob_1 + neigh_churn_prob_2 + edge_between
  ) %>%
  ## fit to training data
  fit(training(employee_nodes_split))



### evaluate predictions
## save as object
glm_probs <- predict(
  # model
  glm,
  # test data
  testing(employee_nodes_split),
```

```
  # type of values
  type = "prob"
) %>%
  ## bind with testing data
  bind_cols(testing(employee_nodes_split))




### ROC curve
## call data
glm_roc <- glm_probs %>%
  ## ROC curve
  roc_curve(churn, .pred_Yes)




### area under ROC curve
## call data
glm_probs %>%
  ## calculate
  roc_auc(churn, .pred_Yes)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.638
```

Task 5.3

First, create a modeling *workflow* named **glmnet_wrkflw** for an *elastic net* using **logistic_reg()** and the **glmnet** engine. Inside of **logistic_reg()**, set **penalty = 1e-10** and **mixture = 0.5**.

Second, fit an *elastic net* model using **glmnet_wrkflw** on the *training* data of **employee_nodes_split** where **churn** is predicted by seven network node measures: (1) **neigh_churn_prob_1**, (2) **neigh_churn_prob_2**, (3) **eigen**, (4) **mean_page_rank**, (5) **closeness**, (6) **loc_trans**, and (7) **n_triangle**. Save the model as **glmnet**.

Third, apply **predict()** to **glmnet** to calculate *probabilities* of **churn** in the *testing* data of **employee_nodes_split**. Bind the predictions with the *testing* data. Save the result as **glmnet_probs**.

Fourth, apply **roc_curve()** to **glmnet_probs**. Save the result as **glmnet_roc**.

Fifth, calculate the area under the receiver-operator characteristic (ROC) curve for **glmnet_probs** using **roc_auc()**.

Sixth, apply **ggplot()** to start a plot, add a diagonal reference line with **geom_abline()**, and add **glm_roc** and **glmnet_roc** via **geom_path()**. Make sure to name the colors of each path inside of **geom_path()** for each model. Add appropriate labels for the axes and legend. Use **scale_color_manual()** to label the paths in the legend. Use **theme_bw()** as the theme. Position the legend at the bottom. Save the plot as **roc_plot**. Display the plot.

Examine the results.

Questions 5.3: Answer these questions: (1) What is the area under the ROC curve for the *elastic net*? (2) Does the *first* or *second* model predict better?

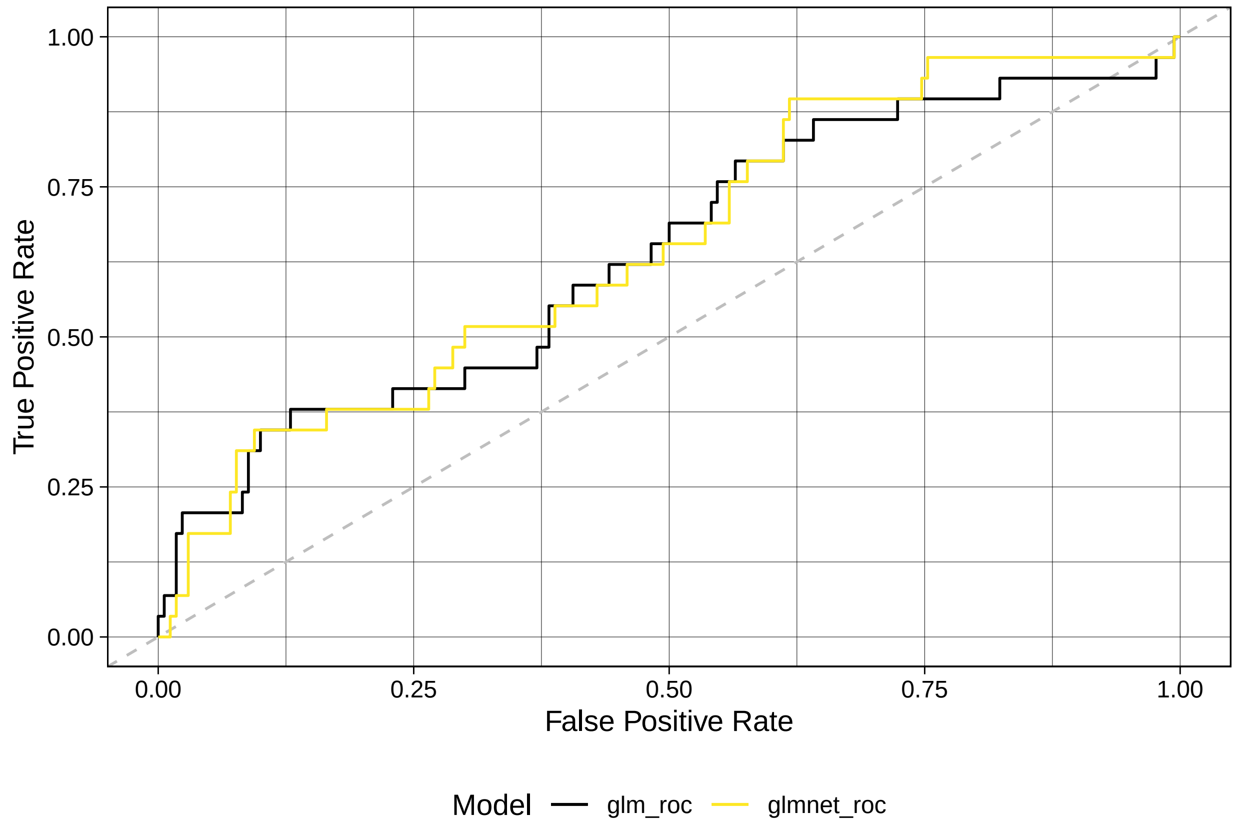
Responses 5.3: *(1) 0.709 (2)Second model.*

```
### logistic regression workflow
## save as object
glmnet_wrkflw <- workflow() %>%
  ## add model
  add_model(
    # specify logistic regression
    logistic_reg(
      penalty = 1e-10,
      mixture = 0.5
    ) %>%
      # engine
      set_engine("glm")
  )


##second logistic regression model
glmnet <- glmnet_wrkflw %>%
  ## add formula
  add_formula(
    # specify equation
    churn ~ neigh_churn_prob_1 + neigh_churn_prob_2 + eigen +
      mean_page_rank  + closeness + loc_trans + n_triangle
  ) %>%
  ## fit to training data
  fit(training(employee_nodes_split))



### evaluate predictions
## save as object
glmnet_probs <- predict(
  # model
  glmnet,
  # test data
  testing(employee_nodes_split),
  # type of values
  type = "prob"
) %>%
  ## bind with testing data
  bind_cols(testing(employee_nodes_split))



### ROC curve
## call data
glmnet_roc <- glmnet_probs %>%
  ## ROC curve
  roc_curve(churn, .pred_Yes)



### area under ROC curve
```

```r
## call data
glmnet_probs %>%
  ## calculate
  roc_auc(churn, .pred_Yes)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.651
```

```r
### save plot
## call plot
roc_plot <- ggplot() +
  ## add dotted diagonal line
  geom_abline(intercept = 0, slope = 1, linetype = 2, color = "gray") +
  ## add first model
  geom_path(
    # data
    glm_roc,
    # mapping
    mapping = aes(x = 1 - specificity, y = sensitivity, color = "glm_roc")
  ) +
  ## add second model
  geom_path(
    # model
    data = glmnet_roc,
    # mapping
    mapping = aes(x = 1 - specificity, y = sensitivity, color = "glmnet_roc")
  ) +
  ## add colors
  scale_color_manual(
    # matching vector
    values = c("glm_roc" = "#000000", "glmnet_roc" = "#FDE725FF")
  ) +
  ## axes and legend labels
  labs(x = "False Positive Rate", y = "True Positive Rate", color = "Model") +
  ## alter theme
  theme_linedraw() +
  ## move legend to bottom
  theme(legend.position = "bottom")

## display plot
roc_plot
```



Task 6: Save Objects

For this task, you will save a plot.

Task 6.1

Save **roc_plot** as **network_roc_auc.png** in the **plots** folder of the project directory using **ggsave()**. Use a width of *9 inches* and height of *9 inches* for all plots.

```
### save plots to folder in project directory
## save a single plot to a file
ggsave(
  ## file path
  here("plots", "network_roc_auc.png"),
  ## plot object
  plot = roc_plot,
  ## dimensions
  units = "in", width = 9, height = 9
)
```

Task 7: Conceptual Question

For your last task, you will respond to a conceptual question.

Question 7.1: What is the difference between *counting the number of triangles* a focal node forms with two other nodes and a node's *local transitivity*?

Response 7.1: *Local transitivity measures what proportion of POSSIBLE first degree neighborhood triangles exist around a focal node while a triangle is the connections already made between three nodes..*