Importing libraries

In [1]:
```
from collections import Counter, defaultdict, deque

import copy

import math

import networkx as nx

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

import random

from tqdm import tqdm

import re

import matplotlib.pyplot as plt

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import classification_report, roc_auc_score

from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix
```

Exploratory data Analysis

In [2]:
```
with open("fb-pages/fb-pages.nodes") as f:
    fb_nodes = f.read().splitlines()

# load edges (or links)
with open("fb-pages/fb-pages.edges") as f:
    fb_links = f.read().splitlines()

len(fb_nodes), len(fb_links)
```

Out[2]: (621, 2102)

In [3]:
```
# captture nodes in 2 separate lists
node_list_1 = []
node_list_2 = []

for i in tqdm(fb_links):
  node_list_1.append(i.split(',')[0])
  node_list_2.append(i.split(',')[1])

fb_df = pd.DataFrame({'node_1': node_list_1, 'node_2': node_list_2})
```

```
100%|██████████| 2102/2102 [00:00<00:00, 632183.21it/s]
```

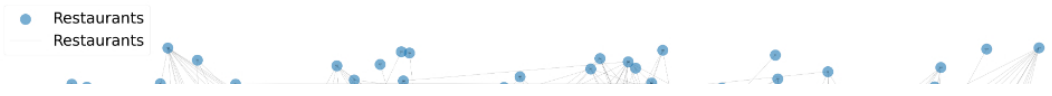
In [4]:
```
fb_df.head()
```

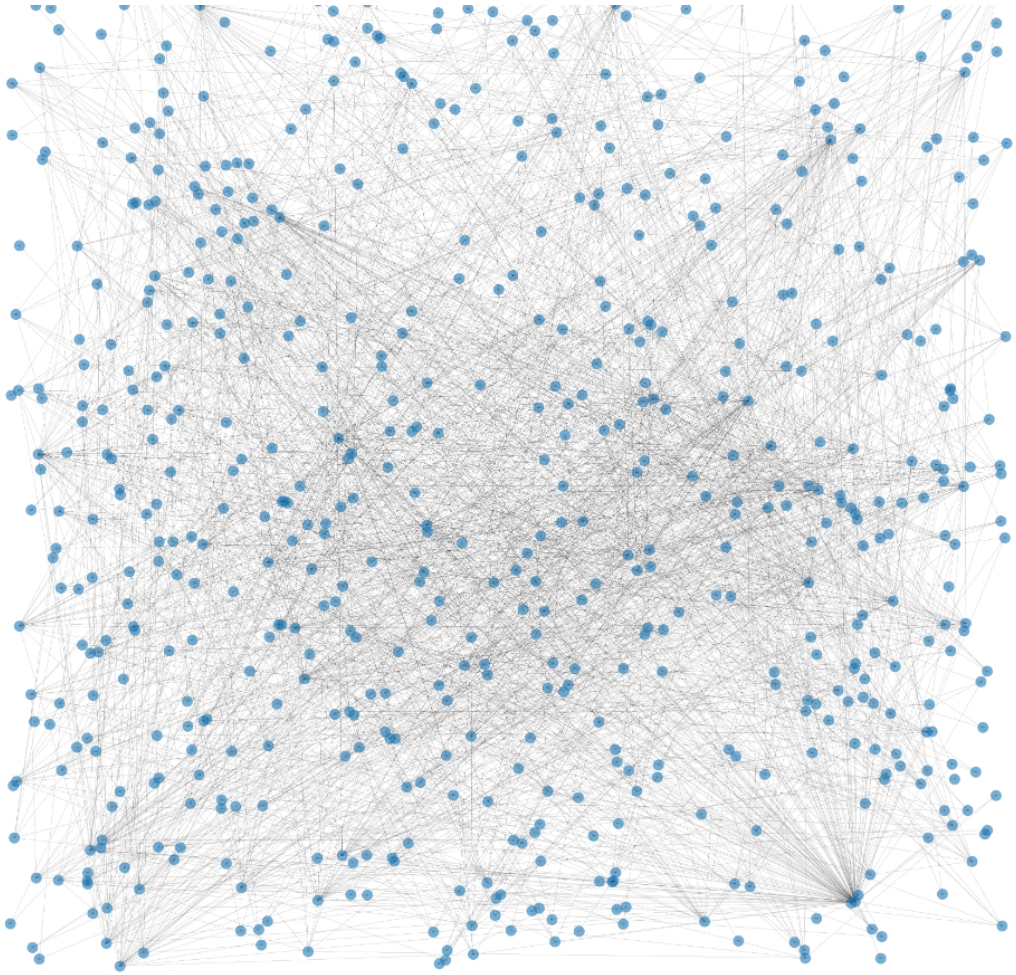
Out[4]:

	node_1	node_2
0	0	276
1	0	58
2	0	132
3	0	603
4	0	398

Create a Graph

In [5]:
```
# create graph
G = nx.from_pandas_edgelist(fb_df, "node_1", "node_2", create_using=nx.Graph())

# plot graph
plt.figure(figsize=(100,100))

pos = nx.random_layout(G, seed=23)
nx.draw(G, with_labels=True,  pos = pos,label = 'Restaurants', node_size = 5000, alpha = 0.6, width = 0.7)

plt.legend(loc=0, prop={'size': 100},scatterpoints = 1,numpoints = 2)

plt.show()
```





Community Detection

Girvan-Newmann Approach

In [6]:
```
def edge_to_remove(graph):

  G_dict = nx.edge_betweenness_centrality(graph)

  edge = ()

  # extract the edge with highest edge betweenness centrality score

  for key, value in sorted(G_dict.items(), key=lambda item: item[1], reverse = True):

      edge = key

      break

  return edge
```

In [7]:
```
def girvan_newman(graph):

    # find number of connected components

    sg = nx.connected_components(graph)

    sg_count = nx.number_connected_components(graph)

    while(sg_count == 1):

        graph.remove_edge(edge_to_remove(graph)[0], edge_to_remove(graph)[1])

        sg = nx.connected_components(graph)

        sg_count = nx.number_connected_components(graph)

    return sg
```

In [8]:
```
# find communities in the graph
c = girvan_newman(G.copy())

# find the nodes forming the communities
node_groups = []

for i in c:
```

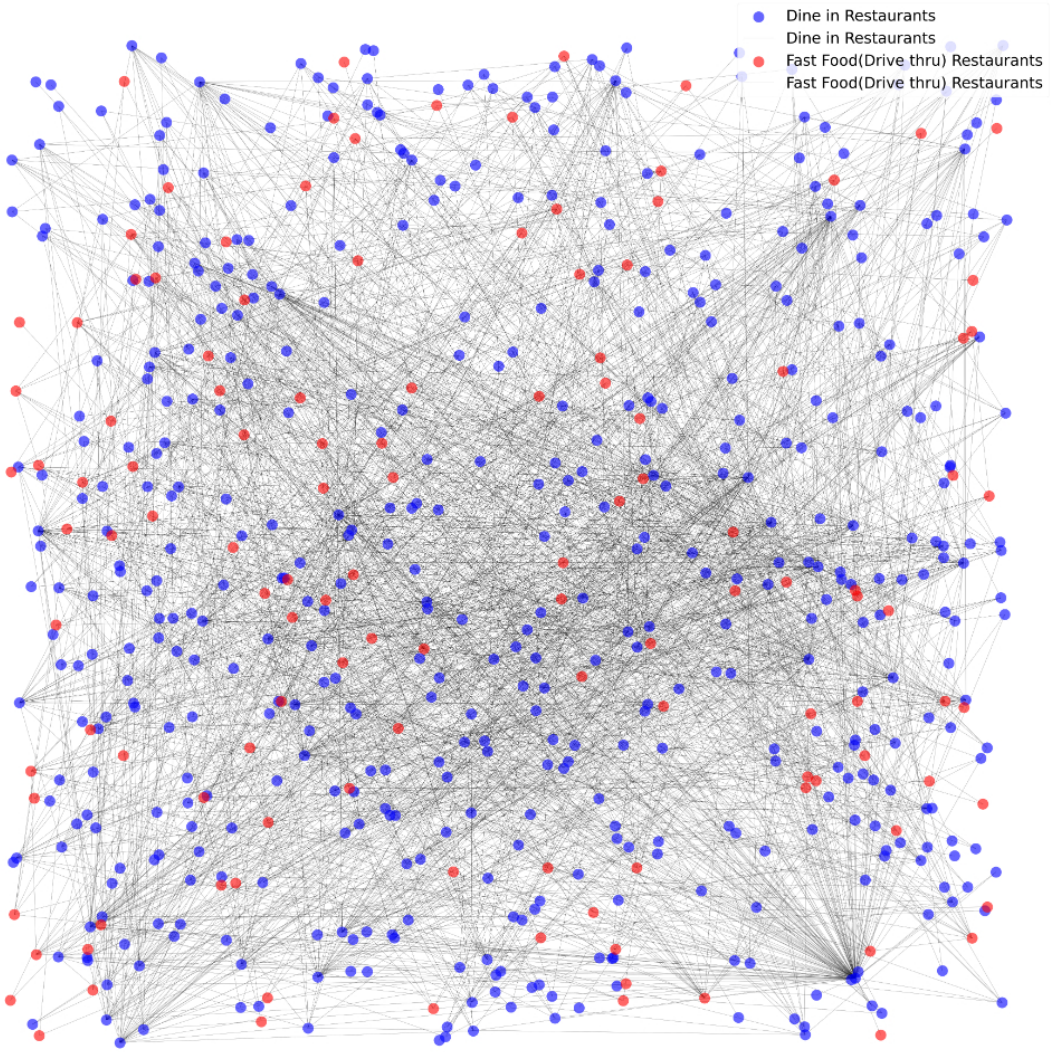
```
    node_groups.append(list(i))
```

In [9]:
```
node_groups
```

Out[9]:
```
[['111',
  '525',
  '506',
  '409',
  '538',
  '619',
  '411',
  '505',
  '288',
  '336',
  '103',
  '188',
  '171',
  '262',
  '416',
  '268',
  '470',
  '70',
  '586',
```

Plotting the communities

In [12]:
```
plt.figure(figsize=(100,100))

nx.draw(G,nodelist=node_groups[0],node_color='blue', label='Dine in Restaurants',   pos = pos, node_size = 5000, alpha = 0.6, wi
nx.draw(G,nodelist=node_groups[1],node_color='red', label='Fast Food(Drive thru) Restaurants',  pos = pos, node_size = 5000, alp

plt.legend(loc=1, prop={'size': 100},scatterpoints = 1)
plt.show()
```



In [13]:
```
# color_map = []
# plt.figure(figsize=(100,100))



# for node in G:
#     if node in node_groups[0]:
#         color_map.append('blue')
```

```
#     else:
#         color_map.append('red')

# nx.draw(G, node_color=color_map, with_labels=True,  pos = pos, node_size = 5000, alpha = 0.6, width = 0.7)
# plt.legend(scatterpoints = 1)
# plt.show()
```

Most Inflential nodes for marketing

Influential nodes based on degree centrality

In [14]:
```
print("Top 100 Degree centrality nodes: ")
centrality = nx.degree_centrality(G)
centrality_dict = dict(sorted(centrality.items(), reverse= True, key=lambda x: x[1])[:100])
print(centrality_dict)
```

```
Top 100 Degree centrality nodes:
{'265': 0.21647819063004847, '518': 0.09531502423263329, '67': 0.09046849757673668, '340': 0.0888529886914378, '90': 0.08239095
315024234, '611': 0.07592891760904685, '56': 0.07592891760904685, '70': 0.07592891760904685, '89': 0.06946688206785137, '288':
0.05977382875605816, '505': 0.05654281098546042, '498': 0.053311793214862686, '317': 0.051696284329563816, '182': 0.05008077544
4264945, '229': 0.050080775444264945, '217': 0.050080775444264945, '87': 0.048465266558966075, '389': 0.048465266558966075, '52
4': 0.04684975767366721, '198': 0.04684975767366721, '9': 0.04523424878836834, '603': 0.04361873990306947, '289': 0.04200323101
77706, '131': 0.0420032310177706, '357': 0.0420032310177706, '248': 0.04038772213247173, '31': 0.04038772213247173, '350': 0.03
8772213247172865, '343': 0.037156704361873995, '597': 0.037156704361873995, '616': 0.037156704361873995, '35': 0.03554119547657
5124, '434': 0.035541195476575124, '545': 0.035541195476575124, '374': 0.035541195476575124, '164': 0.035541195476575124, '60
1': 0.035541195476575124, '116': 0.035541195476575124, '151': 0.033925686591276254, '550': 0.033925686591276254, '446': 0.03231
017770597738, '352': 0.030694668820678516, '555': 0.02746365105008078, '254': 0.02746365105008078, '526': 0.02746365105008078,
'576': 0.02746365105008078, '465': 0.025848142164781908, '613': 0.025848142164781908, '128': 0.025848142164781908, '253': 0.025
848142164781908, '372': 0.025848142164781908, '58': 0.024232633279483037, '227': 0.024232633279483037, '432': 0.024232633279483
037, '491': 0.024232633279483037, '181': 0.024232633279483037, '224': 0.024232633279483037, '235': 0.024232633279483037, '584':
0.024232633279483037, '245': 0.024232633279483037, '593': 0.02261712439418417, '570': 0.02261712439418417, '23': 0.022617124394
18417, '195': 0.02261712439418417, '50': 0.02261712439418417, '136': 0.02261712439418417, '404': 0.02261712439418417, '299': 0.
0210016155088853, '516': 0.0210016155088853, '334': 0.0210016155088853, '117': 0.0210016155088853, '143': 0.0210016155088853,
'238': 0.0210016155088853, '230': 0.0210016155088853, '41': 0.0210016155088853, '558': 0.0210016155088853, '563': 0.02100161550
88853, '311': 0.0210016155088853, '107': 0.019386106623586433, '264': 0.019386106623586433, '43': 0.019386106623586433, '16':
0.019386106623586433, '119': 0.019386106623586433, '400': 0.019386106623586433, '274': 0.019386106623586433, '329': 0.019386106
623586433, '179': 0.019386106623586433, '397': 0.019386106623586433, '494': 0.019386106623586433, '154': 0.019386106623586433,
'323': 0.019386106623586433, '135': 0.019386106623586433, '276': 0.017770597738287562, '345': 0.017770597738287562, '351': 0.01
7770597738287562, '237': 0.017770597738287562, '75': 0.017770597738287562, '147': 0.017770597738287562, '449': 0.01777059773828
7562, '419': 0.017770597738287562}
```

In [15]:
```
blue_community = []
red_community = []
x = centrality.keys()
for node in x:
    if node in node_groups[0]:
        blue_community.append(node)
    else:
        red_community.append(node)
```

In [16]:
```
print ("Most influencing node in Blue Community according to degree centrality:")
node = centrality_dict.keys()
for n in node:
    if n in blue_community:
        print(n)
        print(centrality_dict[n])
        break;
```

```
Most influencing node in Blue Community according to degree centrality:
265
0.21647819063004847
```

In [17]:
```
print ("Most influencing node in Red Community according to degree centrality:")
node = centrality_dict.keys()
for n in node:
    if n in red_community:
        print(n)
        print(centrality_dict[n])
        break;
```

```
Most influencing node in Red Community according to degree centrality:
545
0.035541195476575124
```

Influential nodes based on closeness centrality

In [18]:
```
print("Top 170 CLoseness centrality nodes: ")
centrality = nx.closeness_centrality(G)
centrality_dict = dict(sorted(centrality.items(), reverse= True, key=lambda x: x[1])[:170])
print(centrality_dict)
```

```
Top 170 CLoseness centrality nodes:
{'265': 0.33137044967880086, '611': 0.3028375733855186, '70': 0.29816955684007707, '90': 0.2980259990370727, '56': 0.2947619047
619048, '217': 0.2923948984411904, '505': 0.29115710253998117, '67': 0.2908834586466165, '340': 0.28925233644859816, '248': 0.2
8844361602982294, '41': 0.2880409492787343, '317': 0.2875058058522991, '87': 0.287372330547818, '288': 0.2865740740740741, '19
8': 0.2853849700322729, '449': 0.28499079189686927, '131': 0.28485964104924066, '35': 0.2845977011494253, '434': 0.283684692942
2548, '169': 0.2835547411818598, '597': 0.28136363636363637, '601': 0.2807256235827664, '89': 0.2803442028985507, '465': 0.2795
8446251129176, '229': 0.2793321299638989, '350': 0.2783273381294964, '352': 0.2779524023349798, '9': 0.277205553067622, '238':
0.2759696834596522, '498': 0.27425786442179884, '58': 0.27353071144498453, '289': 0.27353071144498453, '599': 0.273530711444984
53, '343': 0.2730480811645346, '439': 0.2730480811645346, '245': 0.2714912280701754 5, '576': 0.270541958041958, '116': 0.270423
7658366099, '107': 0.27018769096464423, '616': 0.2700698080279232, '195': 0.26971677559912854, '603': 0.2695993031358885, '58
4': 0.26889661164205036, '383': 0.2684301821335646, '446': 0.26808142052836725, '182': 0.26773356401384085, '397': 0.2677335640
1384085, '62': 0.26773356401384085, '242': 0.26761781236489407, '372': 0.2671558049201554, '269': 0.2670405522001726, '121': 0.
2665805340223945, '321': 0.26623655913978495, '227': 0.2660077352814783, '68': 0.2660077352814783, '254': 0.265893470790378, '5
70': 0.265893470790378, '43': 0.265893470790378, '183': 0.265893470790378, '23': 0.26543739279588335, '338': 0.2654373927958833
5, '389': 0.26532361765966567, '15': 0.26520994001713794, '31': 0.2650963597430407, '151': 0.26486949080017114, '351': 0.264643
00983326206, '275': 0.26396588486140726, '404': 0.26362862010221466, '558': 0.26351638995317156, '88': 0.26351638995317156, '59
```

```
4': 0.2629566694987256, '419': 0.26262197708952056, '243': 0.26262197708952056, '148': 0.26262197708952056, '433': 0.26262197708952056, '578': 0.26239932174650277, '181': 0.26206604572396275, '157': 0.26162299239222314, '510': 0.26041228439209085, '257': 0.2600840336134454, '531': 0.2600840336134454, '577': 0.2589958158995816, '83': 0.2579166666666667, '323': 0.2572734829592685, '128': 0.25716659742417947, '7': 0.2569530925695309, '112': 0.25673994193280797, '479': 0.25663349917081263, '585': 0.25663349917081263, '146': 0.2565271446332366, '432': 0.25642087821043913, '102': 0.2563146997929607, '130': 0.2563146997929607, '60': 0.2561026065370294, '436': 0.2558908639933857, '426': 0.2555739058629232, '143': 0.2553630363036304, '615': 0.2552577319587629, '375': 0.2552577319587629, '302': 0.2552577319587629, '136': 0.2552577319587629, '361': 0.2551525144270404, '166': 0.25483738163853437, '550': 0.2546277252159605, '473': 0.2546277252159605, '345': 0.2545230263157895, '360': 0.2545230263157895, '50': 0.2545230263157895, '324': 0.2544184134812988, '526': 0.25400082068116536, '456': 0.2538966365873667, '32': 0.2535845964768538, '364': 0.2530662305805397, '458': 0.2530662305805397, '476': 0.2530662305805397, '271': 0.252859477124183, '454': 0.25275622703144414, '569': 0.2524469820554649 5, '54': 0.25234406848756624, '536': 0.25234406848756624, '159': 0.25224123879380606, '300': 0.25213849287169043, '111': 0.25213849287169043, '261': 0.2520358306188925, '155': 0.2516260162601626, '386': 0.2516260162601626, '2': 0.25152377082486793, '408': 0.25131952902963867, '175': 0.25131952902963867, '312': 0.25131952902963867, '320': 0.2512175324675325, '588': 0.2511156186612576, '59': 0.25101378751013786, '246': 0.25101378751013786, '593': 0.2509120389136603, '277': 0.2509120389136603, '71': 0.2509120389136603, '172': 0.2506072874493927, '193': 0.2501010101010101, '549': 0.25, '34': 0.24989907145740817, '341': 0.24989907145740817, '322': 0.24979822437449556, '4': 0.24969745865268253, '221': 0.24919484702093397, '140': 0.24919484702093397, '346': 0.24919484702093397, '1': 0.24909456740442656, '120': 0.24909456740442656, '18': 0.2489943684633395, '44': 0.248994368463395, '97': 0.248994368463395, '161': 0.248994368463395, '233': 0.248994368463395, '349': 0.248994368463395, '384': 0.248994368463395, '309': 0.248994368463395, '464': 0.248994368463395, '235': 0.2488942501005227, '202': 0.24799679487179488, '459': 0.24799679487179488, '342': 0.24720447284345048, '580': 0.24681020733652312, '72': 0.2465153325368379, '410': 0.2464171974522293, '253': 0.2462211614956245, '355': 0.2449544914918876, '244': 0.2449544914918876, '618': 0.2443742597710225, '518': 0.24427782162588793}
```

In [19]:
```
blue_community = []
red_community = []
x = centrality.keys()
for node in x:
    if node in node_groups[0]:
        blue_community.append(node)
    else:
        red_community.append(node)
```

In [20]:
```
print ("Most influencing node in Blue Community according to closeness centrality:")
node = centrality_dict.keys()
for n in node:
    if n in blue_community:
        print(n)
        print(centrality_dict[n])
        break;
```

```
Most influencing node in Blue Community according to closeness centrality:
265
0.33137044967880086
```

In [21]:
```
print ("Most influencing node in Red Community according to closeness centrality:")
node = centrality_dict.keys()
for n in node:
    if n in red_community:
        print(n)
        print(centrality_dict[n])
        break;
```

```
Most influencing node in Red Community according to closeness centrality:
618
0.2443742597710225
```

Influential nodes based on Betweeness Centrality

In [22]:
```
print("Top 100 Betweenness centrality nodes: ")
centrality = nx.betweenness_centrality(G)
centrality_dict = dict(sorted(centrality.items(), reverse= True, key=lambda x: x[1])[:100])
print(centrality_dict)
```

```
Top 100 Betweenness centrality nodes:
{'265': 0.34990766617377767, '31': 0.1619605706800918, '518': 0.14456288292404343, '618': 0.0932726061636337, '35': 0.09141807568331686, '216': 0.0791116699368157, '498': 0.07229884637101391, '101': 0.05838784338316884, '217': 0.057229302697032476, '148': 0.057020665866287595, '555': 0.055708636860766714, '449': 0.053843029611416536, '434': 0.053573664760333005, '8': 0.052854053335660914, '605': 0.05181350328187588, '585': 0.051722645416473374, '57': 0.05044152014680741, '243': 0.04888551038307147, '264': 0.04763503092470892, '41': 0.047419660544545904, '521': 0.04702144888119783, '340': 0.044111326874311484, '67': 0.04336538407807647, '220': 0.04242625106408027, '169': 0.04177575210707173, '98': 0.040163699179508065, '70': 0.03923916119497953, '179': 0.037761959992914026, '352': 0.03689361658692651, '164': 0.03468671329888868, '611': 0.034435744992609416, '58': 0.033441447 3927701, '154': 0.033692300906915745, '437': 0.033178453661261965, '442': 0.03240275355872105, '441': 0.03135997478398651, '129': 0.029424151076520053, '202': 0.0289570948141725, '459': 0.0289570948141725, '0': 0.028407883773764766, '337': 0.026752148696434 24, '71': 0.02665779343027017, '501': 0.02577116267920569, '90': 0.02547461922006663, '248': 0.024871321345500633, '524': 0.024828786070646845, '269': 0.024081133051677882, '227': 0.023969671900341064, '331': 0.022397540662201797, '545': 0.021944004574974613, '374': 0.021944004574974613, '288': 0.021806113900797612, '176': 0.021771026237974958, '99': 0.021577024296370494, '56': 0.021225935317452115, '389': 0.02117125670223031, '383': 0.02112259194376992, '505': 0.01991453623500195, '146': 0.019399553551188 5, '180': 0.019276314757595246, '45': 0.01923904718891936, '50': 0.019116368115017723, '186': 0.018482254439651128, '7': 0.01750585109659089, '603': 0.01678965737151206, '461': 0.016050525171092324, '397': 0.015481581276640243, '253': 0.015317207150310657, '414': 0.015284960928336951, '142': 0.015049911481012358, '157': 0.014914153029144266, '12': 0.01397192282409423, '224': 0.013820540383199116, '508': 0.013811405130196934, '526': 0.013436713965452127, '181': 0.013199169520630586, '49': 0.012918842898296135, '278': 0.012861332873253133, '537': 0.012861332873253133, '530': 0.012861332873253133, '357': 0.01267425169802053, '311': 0.012672094975747405, '187': 0.012499504412968005, '89': 0.011864705397385007, '114': 0.01169543383990642, '398': 0.011678932742513463, '193': 0.011544564771232562, '229': 0.011331392339664358, '597': 0.011247027614268055, '6': 0.01064135998978152, '577': 0.010506197296614929, '548': 0.010359207141129128, '108': 0.010277128632343719, '350': 0.010236250472230294, '563': 0.010189413636873937, '345': 0.010105983091346556, '432': 0.009881405951439827, '131': 0.009803728953177188, '141': 0.009677368759508762, '47': 0.009672140575413941}
```

In [23]:
```
blue_community = []
red_community = []
x = centrality.keys()
for node in x:
    if node in node_groups[0]:
        blue_community.append(node)
    else:
        red_community.append(node)
```

In [24]:
```
print ("Most influencing node in Blue Community according to betweeness centrality:")
node = centrality_dict.keys()
for n in node:
    if n in blue_community:
        print(n)
```

```
        print(centrality_dict[n])
        break;
```

```
Most influencing node in Blue Community according to betweeness centrality:
265
0.34990766617377767
```

In [25]:
```
print ("Most influencing node in Red Community according to betweeness centrality:")
node = centrality_dict.keys()
for n in node:
    if n in red_community:
        print(n)
        print(centrality_dict[n])
        break;
```

```
Most influencing node in Red Community according to betweeness centrality:
618
0.0932726061636337
```

Link Prediction

Create an adjacency matrix to find unconnected pairs

In [26]:
```
# combine all nodes in a list
node_list = node_list_1 + node_list_2

# remove duplicate items from the list
node_list = list(dict.fromkeys(node_list))

# build adjacency matrix
adj_G = nx.to_numpy_matrix(G, nodelist = node_list)
adj_G.shape
```

Out[26]: (620, 620)

Get all unconnected pairs (Negative Samples)

In [27]:
```
# get unconnected node-pairs
all_unconnected_pairs = []

# traverse adjacency matrix
offset = 0
for i in tqdm(range(adj_G.shape[0])):
  for j in range(offset,adj_G.shape[1]):
    if i != j:
      if nx.shortest_path_length(G, str(i), str(j)) <=2:
        if adj_G[i,j] == 0:
          all_unconnected_pairs.append([node_list[i],node_list[j]])

  offset = offset + 1
```

```
100%|██████████| 620/620 [00:14<00:00, 41.98it/s]
```

In [28]:
```
len(all_unconnected_pairs)
```

Out[28]: 19018

In [29]:
```
node_1_unlinked = [i[0] for i in all_unconnected_pairs]
node_2_unlinked = [i[1] for i in all_unconnected_pairs]

data = pd.DataFrame({'node_1':node_1_unlinked,
                     'node_2':node_2_unlinked})

# add target variable 'link'
data['link'] = 0
```

Remove Links from Connected Node Pairs – Positive Samples

In [30]:
```
initial_node_count = len(G.nodes)

fb_df_temp = fb_df.copy()

# empty list to store removable links
omissible_links_index = []

for i in tqdm(fb_df.index.values):

  # remove a node pair and build a new graph
  G_temp = nx.from_pandas_edgelist(fb_df_temp.drop(index = i), "node_1", "node_2", create_using=nx.Graph())

  # check there is no spliting of graph and number of nodes is same
  if (nx.number_connected_components(G_temp) == 1) and (len(G_temp.nodes) == initial_node_count):
    omissible_links_index.append(i)
    fb_df_temp = fb_df_temp.drop(index = i)
```

```
100%|██████████| 2102/2102 [00:07<00:00, 277.42it/s]
```

In [31]:
```
len(omissible_links_index)
```

Out[31]: 1483

Data for Model Training

In [32]:
```
# create dataframe of removable edges
fb_df_ghost = fb_df.loc[omissible_links_index]
```

```
# add the target variable 'link'
fb_df_ghost['link'] = 1

data = data.append(fb_df_ghost[['node_1', 'node_2', 'link']], ignore_index=True)
```

In [33]:
```
data['link'].value_counts()
```

Out[33]:
```
0    19018
1     1483
Name: link, dtype: int64
```

Feature Extraction

In [34]:
```
# drop removable edges
fb_df_partial = fb_df.drop(index=fb_df_ghost.index.values)

# build graph
G_data = nx.from_pandas_edgelist(fb_df_partial, "node_1", "node_2", create_using=nx.Graph())
```

Train the node2vec model

In [35]:
```
from node2vec import Node2Vec

# Generate walks
node2vec = Node2Vec(G_data, dimensions=100, walk_length=16, num_walks=50)

# train node2vec model
n2w_model = node2vec.fit(window=7, min_count=1)
```

```
Computing transition probabilities: 100%|██████████| 620/620 [00:00<00:00, 16941.60it/s]
Generating walks (CPU: 1): 100%|██████████| 50/50 [00:16<00:00,  3.12it/s]
```

Building our Link Prediction Model

In [36]:
```
x = [(n2w_model[str(i)]+n2w_model[str(j)]) for i,j in zip(data['node_1'], data['node_2'])]

xtrain, xtest, ytrain, ytest = train_test_split(np.array(x), data['link'],
                                                test_size = 0.3,
                                                random_state = 35)
```

```
<ipython-input-36-333635760f4b>:1: DeprecationWarning: Call to deprecated `__getitem__` (Method will be removed in 4.0.0, use s
elf.wv.__getitem__() instead).
  x = [(n2w_model[str(i)]+n2w_model[str(j)]) for i,j in zip(data['node_1'], data['node_2'])]
```

Fitting it to a logistic regression model

In [37]:
```
lr = LogisticRegression(class_weight="balanced",max_iter=1000)

lr.fit(xtrain, ytrain)
```

Out[37]:
```
LogisticRegression(class_weight='balanced', max_iter=1000)
```

In [41]:
```
predictions = lr.predict(xtest)
preds = lr.predict_proba(xtest)
```

In [42]:
```
roc_auc_score(ytest, preds[:,1])
```

Out[42]:
```
0.8133410437212157
```

Visualisation

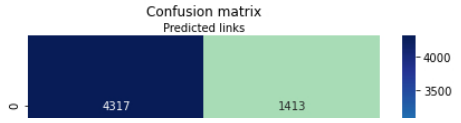
In [43]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

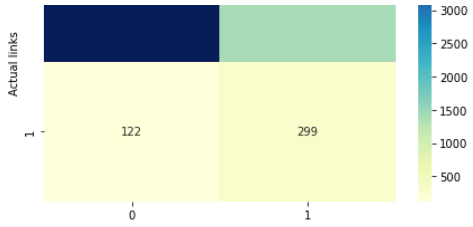
In [45]:
```
import sklearn.metrics as metrics
cm =  metrics.confusion_matrix(ytest, predictions)
print(cm)
```

```
[[4317 1413]
 [ 122  299]]
```

In [46]:
```
class_names=[0,1] # name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cm), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual links')
plt.xlabel('Predicted links')
```

Out[46]:
```
Text(0.5, 257.44, 'Predicted links')
```

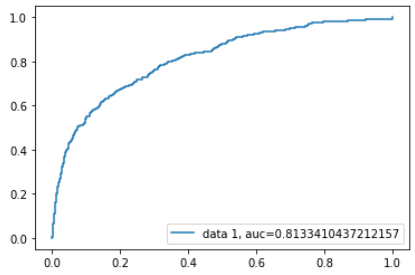




In [47]:
```
print("Accuracy:",metrics.accuracy_score(ytest, predictions))
print("Precision:",metrics.precision_score(ytest,predictions))
print("Recall:",metrics.recall_score(ytest, predictions))
```

```
Accuracy: 0.750447081775321
Precision: 0.1746495327102804
Recall: 0.7102137767220903
```

In [48]:
```
preds = lr.predict_proba(xtest)[::,1]
fpr, tpr, _ = metrics.roc_curve(ytest, preds)
auc = metrics.roc_auc_score(ytest, preds)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



Fitting it to a LightGBM model to increase efficiency

In [49]:
```
import lightgbm as lgbm

train_data = lgbm.Dataset(xtrain, ytrain)
test_data = lgbm.Dataset(xtest, ytest)

# define parameters
parameters = {
    'objective': 'binary',
    'metric': 'auc',
    'boosting_type':'gbdt',
    'is_unbalance': 'true',
    'feature_fraction': 0.5,
    'bagging_fraction': 0.5,
    'bagging_freq': 20,
    'num_threads' : 2,
    'seed' : 76
}

# train lightGBM model
model = lgbm.train(parameters,
                   train_data,
                   valid_sets=test_data,
                   num_boost_round=1000
                   )
```

```
[LightGBM] [Info] Number of positive: 1062, number of negative: 13288
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of testing was 0.005885 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 25500
[LightGBM] [Info] Number of data points in the train set: 14350, number of used features: 100
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.074007 -> initscore=-2.526707
[LightGBM] [Info] Start training from score -2.526707
[1]     valid_0's auc: 0.738778
[2]     valid_0's auc: 0.787866
[3]     valid_0's auc: 0.809569
[4]     valid_0's auc: 0.822418
[5]     valid_0's auc: 0.830611
[6]     valid_0's auc: 0.838479
[7]     valid_0's auc: 0.842538
[8]     valid_0's auc: 0.849197
[9]     valid_0's auc: 0.854569
[10]    valid_0's auc: 0.856568
[11]    valid_0's auc: 0.857311
[12]    valid_0's auc: 0.858705
```

In [50]:
```
y_pred = model.predict(xtest)
```

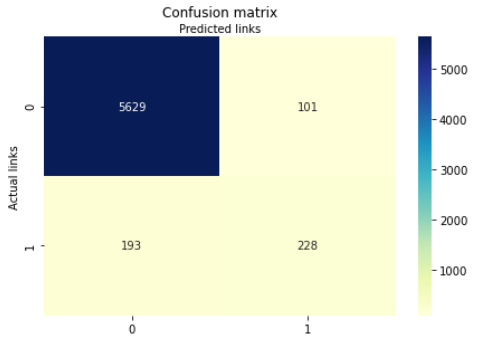
In [51]:
```
for i in range(len(y_pred)):
    if y_pred[i]>=.5:       # setting threshold to .5
       y_pred[i]=1
    else:
       y_pred[i]=0
```

In [52]:
```
cm_lgbm =  metrics.confusion_matrix(ytest, y_pred)

print(cm_lgbm)
```

```
[[5629  101]
 [ 193  228]]
```

In [53]:
```
class_names=[0,1] # name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cm_lgbm), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual links')
plt.xlabel('Predicted links')
```

Out[53]: Text(0.5, 257.44, 'Predicted links')



In [54]:
```
print("Accuracy:",metrics.accuracy_score(ytest, y_pred))
print("Precision:",metrics.precision_score(ytest,y_pred))
print("Recall:",metrics.recall_score(ytest, y_pred))
```

```
Accuracy: 0.9522028938384003
Precision: 0.6930091185410334
Recall: 0.5415676959619953
```

In [ ]: