Academic Year: 2023-24 **Semester: VIII**

Class / Branch: BE Computer **Subject: Social Media Analytics Lab**

Name: Shivam Pandey

Date of performance: 28/02/2024

Date of Submission: 20/03/2024

Experiment No. 06

Aim: Develop a structure-based social media analytics model for any business.

Objective: Apply network analysis methodologies to explore the structure and dynamics of COVID-19 literature, uncovering influential papers, key researchers, and thematic trends. Utilize centrality measures, community detection, and visualization techniques to enhance interdisciplinary research efforts, inform evidence-based decision-making, and drive innovation in pandemic response strategies.

Software used: Kaggle Notebook.

Theory:

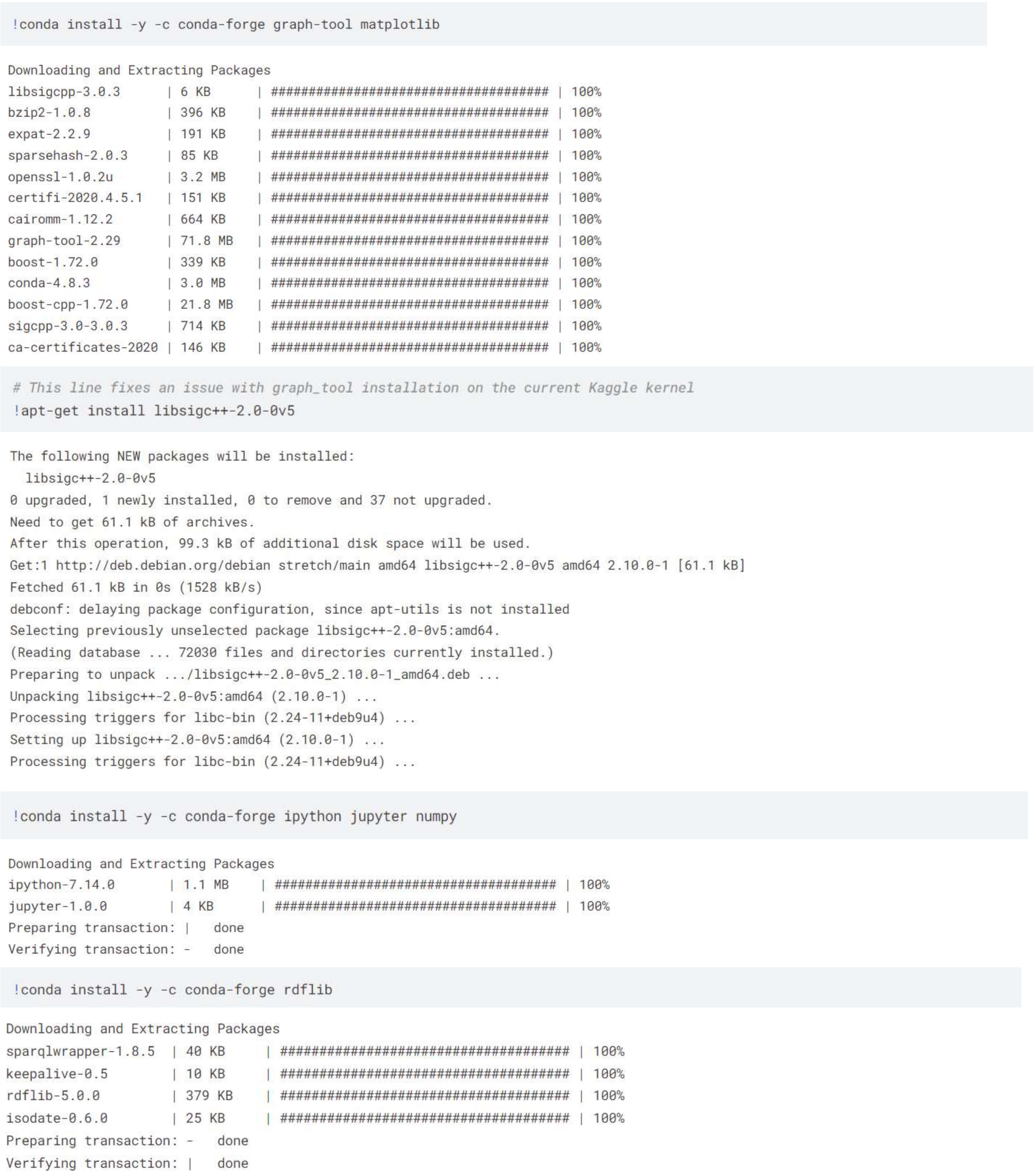
Network analysis is a robust methodology employed across various disciplines to study complex systems consisting of interconnected entities, such as social networks, biological networks, and information networks. At its core, network analysis examines the structure, behavior, and dynamics of these systems by representing them as networks composed of nodes (representing entities) and edges (representing relationships or interactions between entities).

In network analysis, the choice of methodology and tools depends on the specific characteristics of the dataset and the research objectives. Various techniques such as centrality analysis, community detection, and network visualization are commonly employed to explore different aspects of network structure and dynamics. Centrality measures like degree centrality, betweenness centrality, and eigenvector centrality help identify nodes that play crucial roles in information flow, brokerage, and influence propagation within the network. Community detection algorithms such as modularity optimization and hierarchical clustering reveal clusters or modules of nodes that exhibit strong internal connections, often corresponding to cohesive subgroups with shared characteristics or functions. Additionally, network visualization techniques provide intuitive representations of complex networks, enabling researchers to visually inspect and interpret network structures, patterns, and relationships.

The motivation for utilizing network analysis stems from its ability to unveil hidden patterns, identify central elements, and discern emergent properties within intricate datasets. By analyzing the topology of networks, researchers can identify key nodes with high centrality measures, indicating their significance within the network. Additionally, network analysis enables the exploration of clustering and community structures, shedding light on cohesive groups or modules within the system.

The application of network analysis to COVID-19 literature facilitates comprehensive exploration and understanding of the research landscape surrounding the pandemic. By constructing citation networks, co-authorship networks, or semantic networks from COVID-19 literature databases, researchers can uncover influential papers, key researchers, and thematic trends. Network analysis also facilitates the identification of interdisciplinary collaborations and knowledge diffusion pathways, thereby enhancing interdisciplinary research efforts and knowledge exchange. Through the integration of advanced network analysis methodologies and domain-specific insights, researchers can derive actionable insights, inform evidence-based decision-making, and drive innovation in combating the COVID-19 pandemic.

Implementation and Output:



```
!conda install -y -c conda-forge graph-tool matplotlib
```

```
Downloading and Extracting Packages
libsigcpp-3.0.3      | 6 KB      | ##################################### | 100%
bzip2-1.0.8          | 396 KB    | ##################################### | 100%
expat-2.2.9          | 191 KB    | ##################################### | 100%
sparsehash-2.0.3     | 85 KB     | ##################################### | 100%
openssl-1.0.2u       | 3.2 MB    | ##################################### | 100%
certifi-2020.4.5.1   | 151 KB    | ##################################### | 100%
cairomm-1.12.2       | 664 KB    | ##################################### | 100%
graph-tool-2.29      | 71.8 MB   | ##################################### | 100%
boost-1.72.0         | 339 KB    | ##################################### | 100%
conda-4.8.3          | 3.0 MB    | ##################################### | 100%
boost-cpp-1.72.0     | 21.8 MB   | ##################################### | 100%
sigcpp-3.0-3.0.3     | 714 KB    | ##################################### | 100%
ca-certificates-2020 | 146 KB    | ##################################### | 100%
```

```
# This line fixes an issue with graph_tool installation on the current Kaggle kernel
!apt-get install libsigc++-2.0-0v5
```

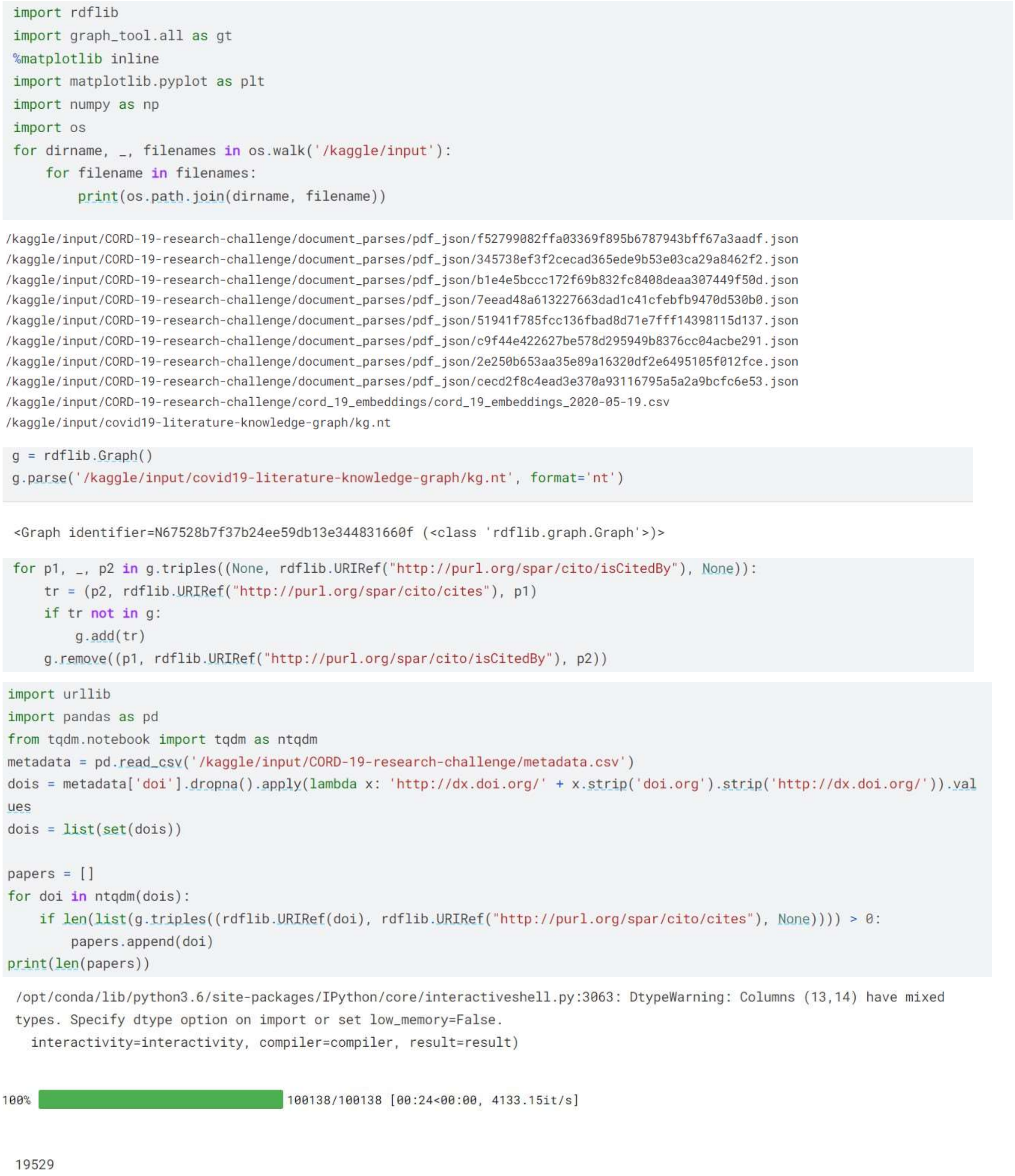
```
The following NEW packages will be installed:
  libsigc++-2.0-0v5
0 upgraded, 1 newly installed, 0 to remove and 37 not upgraded.
Need to get 61.1 kB of archives.
After this operation, 99.3 kB of additional disk space will be used.
Get:1 http://deb.debian.org/debian stretch/main amd64 libsigc++-2.0-0v5 amd64 2.10.0-1 [61.1 kB]
Fetched 61.1 kB in 0s (1528 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package libsigc++-2.0-0v5:amd64.
(Reading database ... 72030 files and directories currently installed.)
Preparing to unpack .../libsigc++-2.0-0v5_2.10.0-1_amd64.deb ...
Unpacking libsigc++-2.0-0v5:amd64 (2.10.0-1) ...
Processing triggers for libc-bin (2.24-11+deb9u4) ...
Setting up libsigc++-2.0-0v5:amd64 (2.10.0-1) ...
Processing triggers for libc-bin (2.24-11+deb9u4) ...
```

```
!conda install -y -c conda-forge ipython jupyter numpy
```

```
Downloading and Extracting Packages
ipython-7.14.0       | 1.1 MB    | ##################################### | 100%
jupyter-1.0.0        | 4 KB      | ##################################### | 100%
Preparing transaction: |   done
Verifying transaction: -   done
```

```
!conda install -y -c conda-forge rdflib
```

```
Downloading and Extracting Packages
sparqlwrapper-1.8.5  | 40 KB     | ##################################### | 100%
keepalive-0.5        | 10 KB     | ##################################### | 100%
rdflib-5.0.0         | 379 KB    | ##################################### | 100%
isodate-0.6.0        | 25 KB     | ##################################### | 100%
Preparing transaction: -   done
Verifying transaction: |   done
```



```
import rdflib
import graph_tool.all as gt
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/CORD-19-research-challenge/document_parses/pdf_json/f52799082ffa03369f895b6787943bff67a3aadf.json
/kaggle/input/CORD-19-research-challenge/document_parses/pdf_json/345738ef3f2cecad365ede9b53e03ca29a8462f2.json
/kaggle/input/CORD-19-research-challenge/document_parses/pdf_json/b1e4e5bccc172f69b832fc8408deaa307449f50d.json
/kaggle/input/CORD-19-research-challenge/document_parses/pdf_json/7eead48a613227663dad1c41cfebfb9470d530b0.json
/kaggle/input/CORD-19-research-challenge/document_parses/pdf_json/51941f785fcc136fbad8d71e7fff14398115d137.json
/kaggle/input/CORD-19-research-challenge/document_parses/pdf_json/c9f44e422627be578d295949b8376cc04acbe291.json
/kaggle/input/CORD-19-research-challenge/document_parses/pdf_json/2e250b653aa35e89a16320df2e6495105f012fce.json
/kaggle/input/CORD-19-research-challenge/document_parses/pdf_json/cecd2f8c4ead3e370a93116795a5a2a9bcfc6e53.json
/kaggle/input/CORD-19-research-challenge/cord_19_embeddings/cord_19_embeddings_2020-05-19.csv
/kaggle/input/covid19-literature-knowledge-graph/kg.nt
```

```
g = rdflib.Graph()
g.parse('/kaggle/input/covid19-literature-knowledge-graph/kg.nt', format='nt')
```

```
<Graph identifier=N67528b7f37b24ee59db13e344831660f (<class 'rdflib.graph.Graph'>)>
```

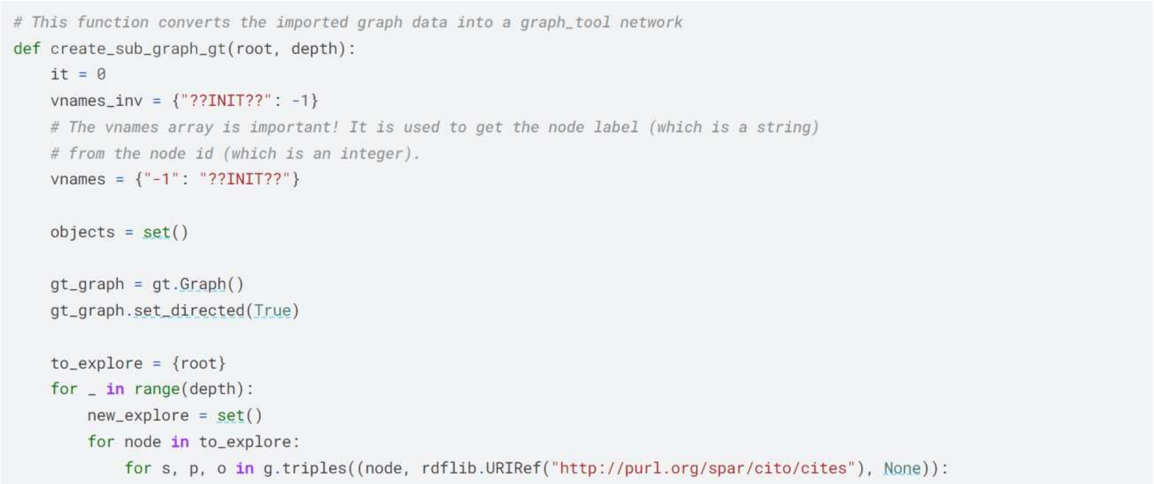
```
for p1, _, p2 in g.triples((None, rdflib.URIRef("http://purl.org/spar/cito/isCitedBy"), None)):
    tr = (p2, rdflib.URIRef("http://purl.org/spar/cito/cites"), p1)
    if tr not in g:
        g.add(tr)
    g.remove((p1, rdflib.URIRef("http://purl.org/spar/cito/isCitedBy"), p2))
```

```
import urllib
import pandas as pd
from tqdm.notebook import tqdm as ntqdm
metadata = pd.read_csv('/kaggle/input/CORD-19-research-challenge/metadata.csv')
dois = metadata['doi'].dropna().apply(lambda x: 'http://dx.doi.org/' + x.strip('doi.org').strip('http://dx.doi.org/')).val
ues
dois = list(set(dois))

papers = []
for doi in ntqdm(dois):
    if len(list(g.triples((rdflib.URIRef(doi), rdflib.URIRef("http://purl.org/spar/cito/cites"), None)))) > 0:
        papers.append(doi)
print(len(papers))
```

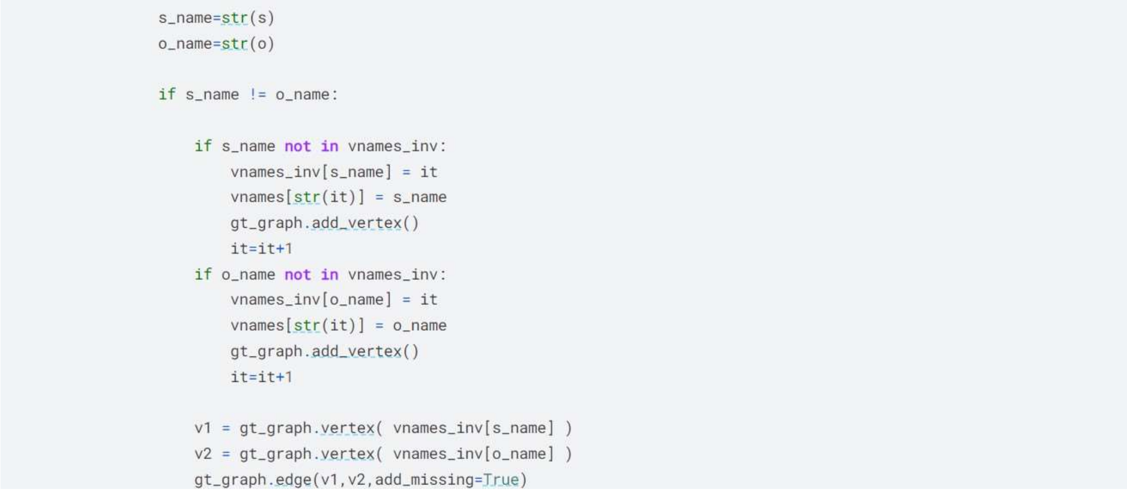
```
/opt/conda/lib/python3.6/site-packages/IPython/core/interactiveshell.py:3063: DtypeWarning: Columns (13,14) have mixed
types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

```
100% ████████████████ 100138/100138 [00:24<00:00, 4133.15it/s]
```

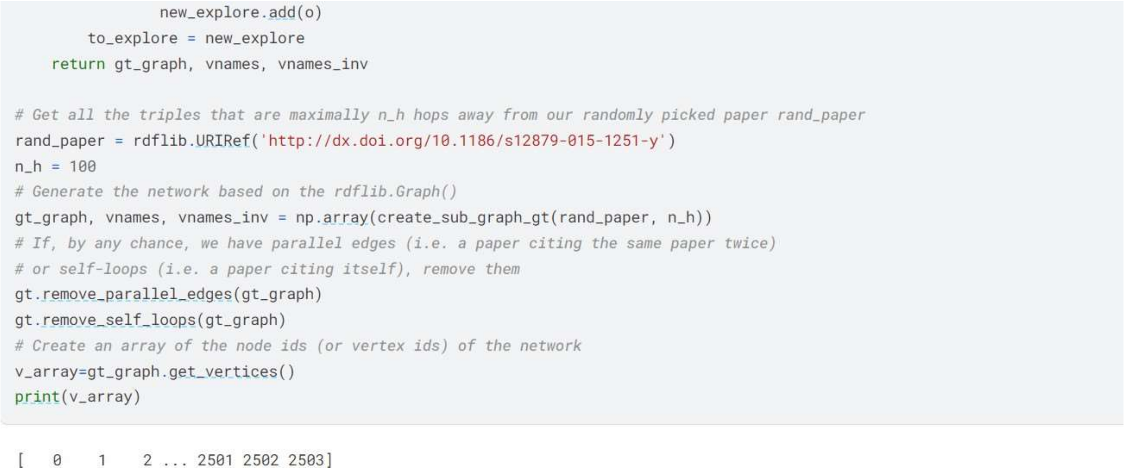
```
19529
```



```
# This function converts the imported graph data into a graph_tool network
def create_sub_graph_gt(root, depth):
    it = 0
    vnames_inv = {"??INIT??": -1}
    # The vnames array is important! It is used to get the node label (which is a string)
    # from the node id (which is an integer).
    vnames = {"-1": "??INIT??"}

    objects = set()

    gt_graph = gt.Graph()
    gt_graph.set_directed(True)

    to_explore = {root}
    for _ in range(depth):
        new_explore = set()
        for node in to_explore:
            for s, p, o in g.triples((node, rdflib.URIRef("http://purl.org/spar/cito/cites"), None)):
```

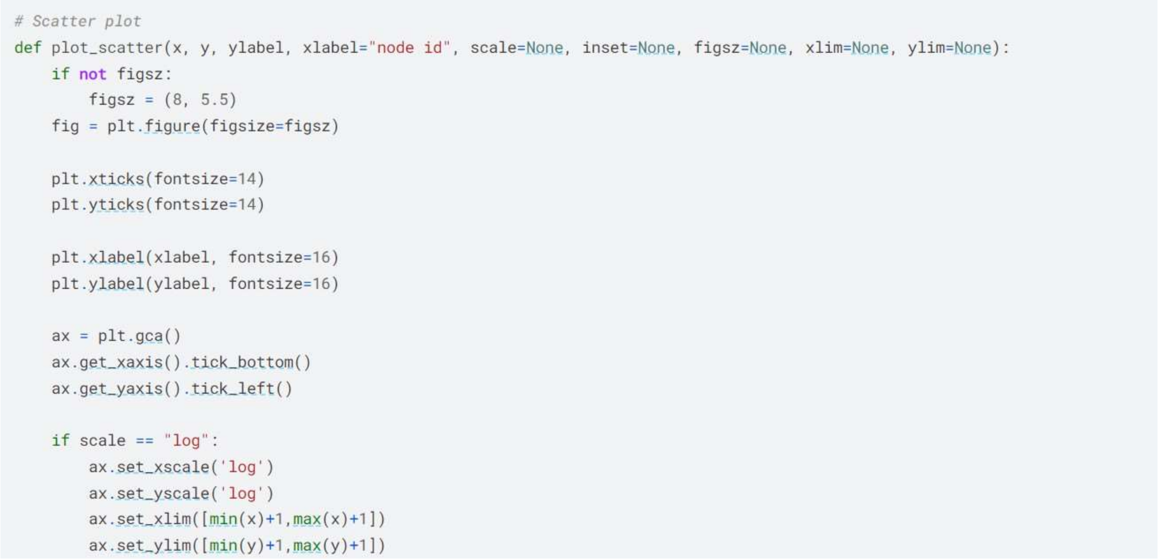


```
                s_name=str(s)
                o_name=str(o)

                if s_name != o_name:

                    if s_name not in vnames_inv:
                        vnames_inv[s_name] = it
                        vnames[str(it)] = s_name
                        gt_graph.add_vertex()
                        it=it+1
                    if o_name not in vnames_inv:
                        vnames_inv[o_name] = it
                        vnames[str(it)] = o_name
                        gt_graph.add_vertex()
                        it=it+1

                    v1 = gt_graph.vertex( vnames_inv[s_name] )
                    v2 = gt_graph.vertex( vnames_inv[o_name] )
                    gt_graph.edge(v1,v2,add_missing=True)
```

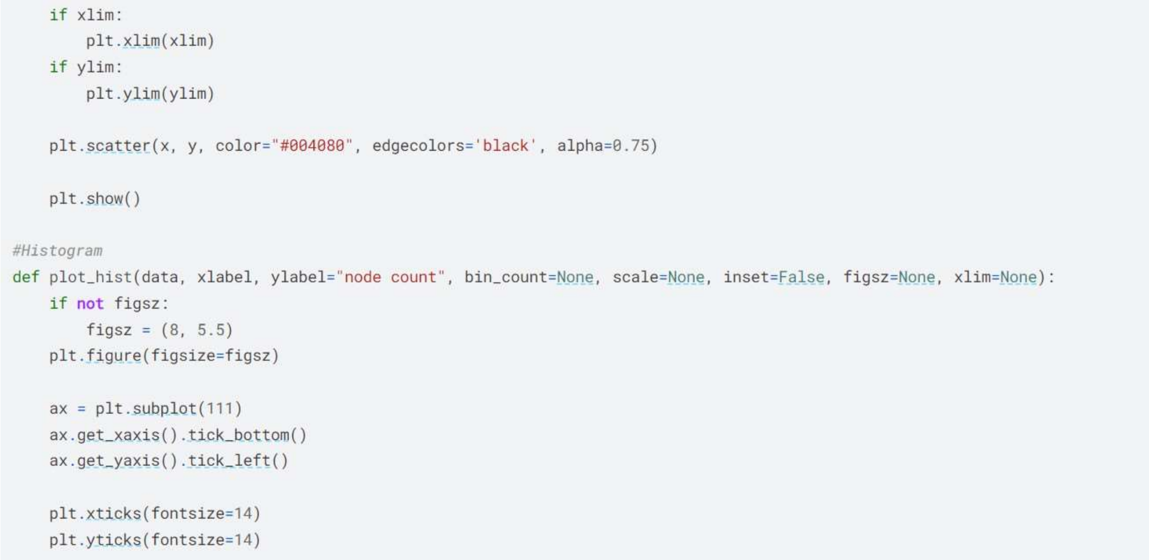


```
                new_explore.add(o)
        to_explore = new_explore
    return gt_graph, vnames, vnames_inv

# Get all the triples that are maximally n_h hops away from our randomly picked paper rand_paper
rand_paper = rdflib.URIRef('http://dx.doi.org/10.1186/s12879-015-1251-y')
n_h = 100
# Generate the network based on the rdflib.Graph()
gt_graph, vnames, vnames_inv = np.array(create_sub_graph_gt(rand_paper, n_h))
# If, by any chance, we have parallel edges (i.e. a paper citing the same paper twice)
# or self-loops (i.e. a paper citing itself), remove them
gt.remove_parallel_edges(gt_graph)
gt.remove_self_loops(gt_graph)
# Create an array of the node ids (or vertex ids) of the network
v_array=gt_graph.get_vertices()
print(v_array)
```

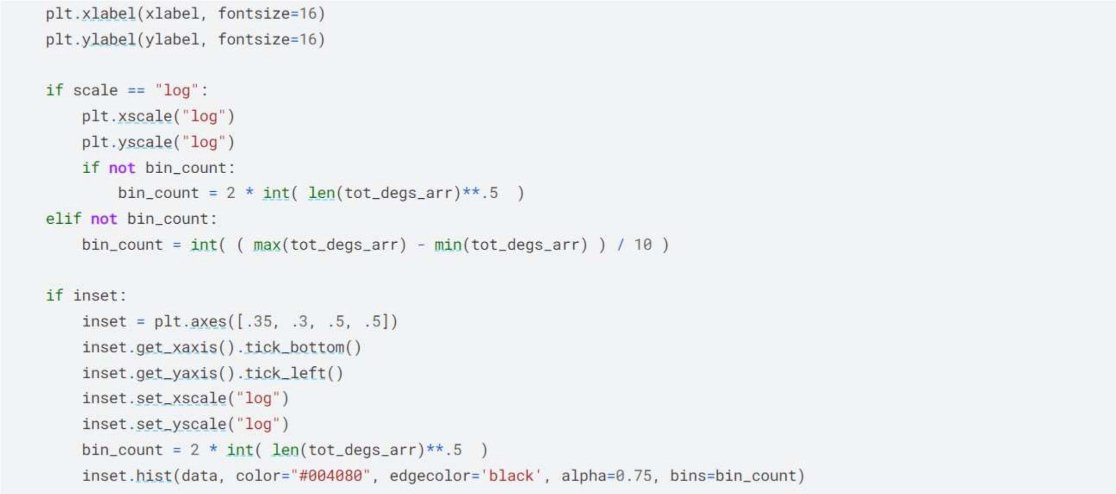
```
[   0    1    2 ... 2501 2502 2503]
```



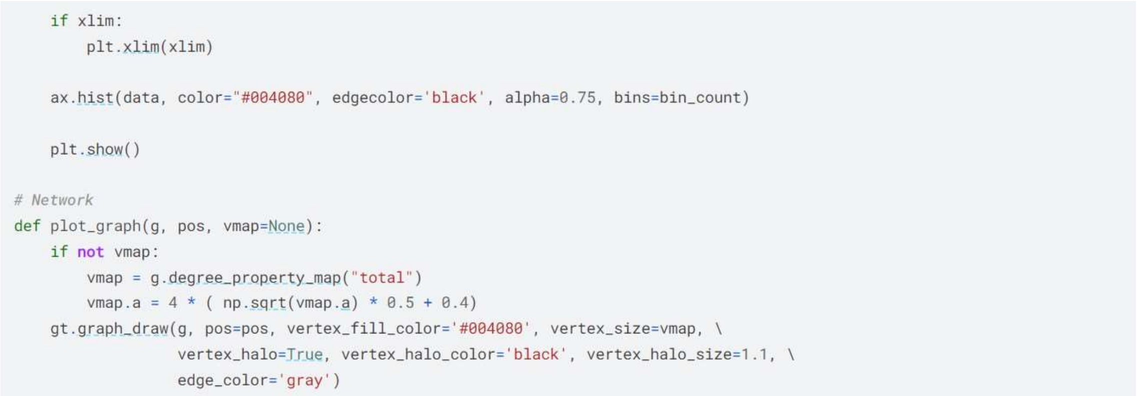
```
# Scatter plot
def plot_scatter(x, y, ylabel, xlabel="node id", scale=None, inset=None, figsz=None, xlim=None, ylim=None):
    if not figsz:
        figsz = (8, 5.5)
    fig = plt.figure(figsize=figsz)

    plt.xticks(fontsize=14)
    plt.yticks(fontsize=14)

    plt.xlabel(xlabel, fontsize=16)
    plt.ylabel(ylabel, fontsize=16)

    ax = plt.gca()
    ax.get_xaxis().tick_bottom()
    ax.get_yaxis().tick_left()

    if scale == "log":
        ax.set_xscale('log')
        ax.set_yscale('log')
        ax.set_xlim([min(x)+1,max(x)+1])
        ax.set_ylim([min(y)+1,max(y)+1])
```



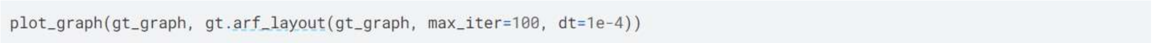
```
    if xlim:
        plt.xlim(xlim)
    if ylim:
        plt.ylim(ylim)

    plt.scatter(x, y, color="#004080", edgecolors='black', alpha=0.75)

    plt.show()

#Histogram
def plot_hist(data, xlabel, ylabel="node count", bin_count=None, scale=None, inset=False, figsz=None, xlim=None):
    if not figsz:
        figsz = (8, 5.5)
    plt.figure(figsize=figsz)

    ax = plt.subplot(111)
    ax.get_xaxis().tick_bottom()
    ax.get_yaxis().tick_left()

    plt.xticks(fontsize=14)
    plt.yticks(fontsize=14)
```



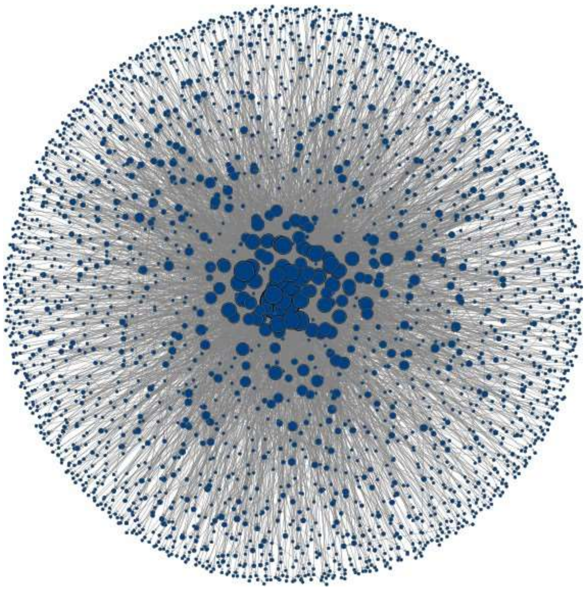
```
    plt.xlabel(xlabel, fontsize=16)
    plt.ylabel(ylabel, fontsize=16)

    if scale == "log":
        plt.xscale("log")
        plt.yscale("log")
        if not bin_count:
            bin_count = 2 * int( len(tot_degs_arr)**.5  )
    elif not bin_count:
        bin_count = int( ( max(tot_degs_arr) - min(tot_degs_arr) ) / 10 )

    if inset:
        inset = plt.axes([.35, .3, .5, .5])
        inset.get_xaxis().tick_bottom()
        inset.get_yaxis().tick_left()
        inset.set_xscale("log")
        inset.set_yscale("log")
        bin_count = 2 * int( len(tot_degs_arr)**.5  )
        inset.hist(data, color="#004080", edgecolor='black', alpha=0.75, bins=bin_count)
```

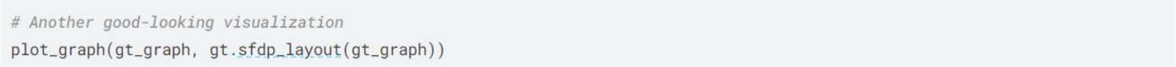


```
    if xlim:
        plt.xlim(xlim)

    ax.hist(data, color="#004080", edgecolor='black', alpha=0.75, bins=bin_count)

    plt.show()

# Network
def plot_graph(g, pos, vmap=None):
    if not vmap:
        vmap = g.degree_property_map("total")
        vmap.a = 4 * ( np.sqrt(vmap.a) * 0.5 + 0.4)
    gt.graph_draw(g, pos=pos, vertex_fill_color='#004080', vertex_size=vmap, \
                  vertex_halo=True, vertex_halo_color='black', vertex_halo_size=1.1, \
                  edge_color='gray')
```

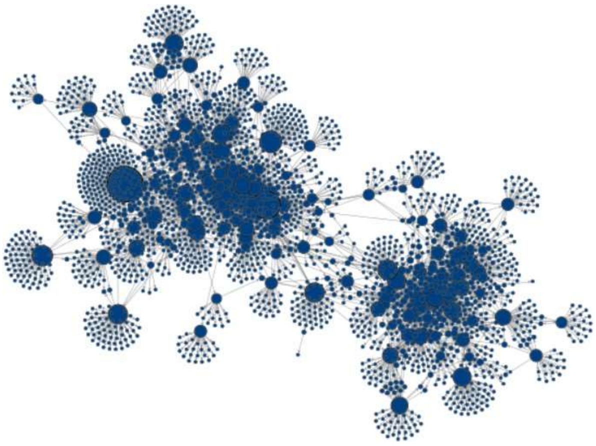


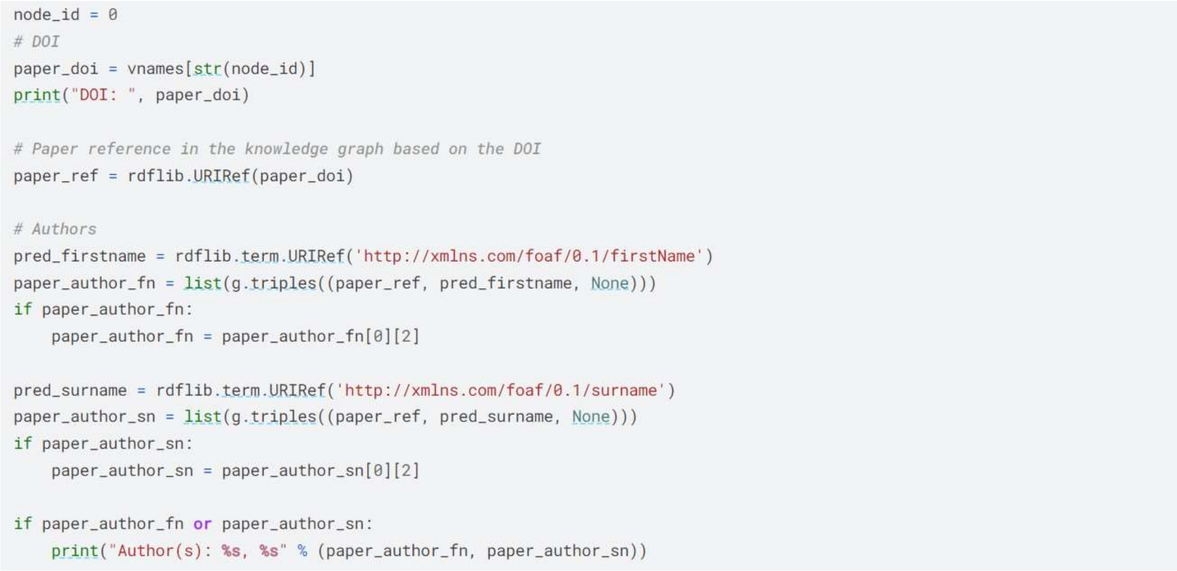
```
plot_graph(gt_graph, gt.arf_layout(gt_graph, max_iter=100, dt=1e-4))
```



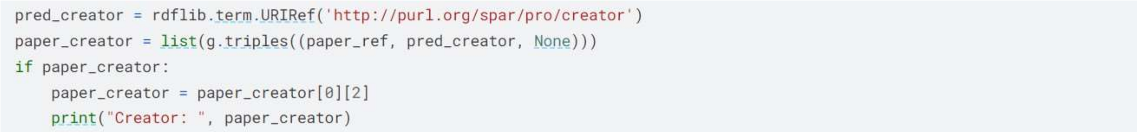


```
# Another good-looking visualization
plot_graph(gt_graph, gt.sfdp_layout(gt_graph))
```

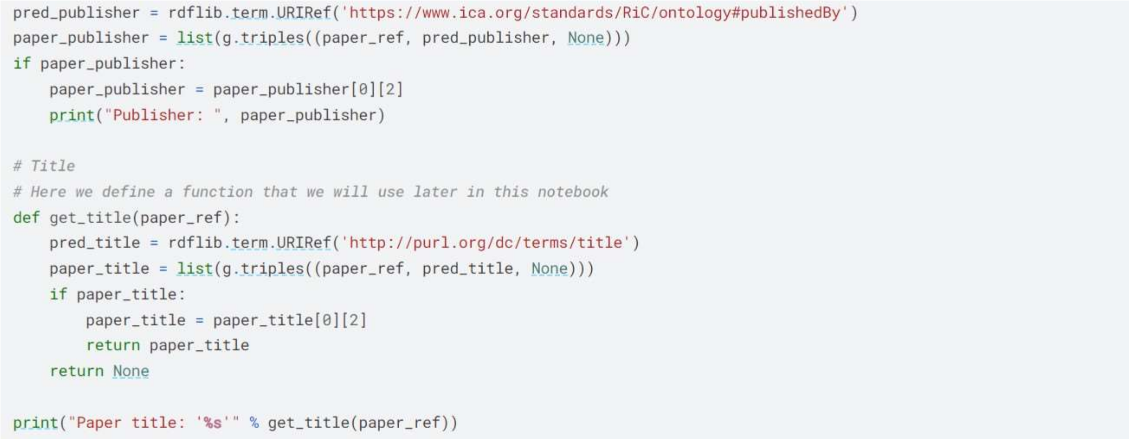




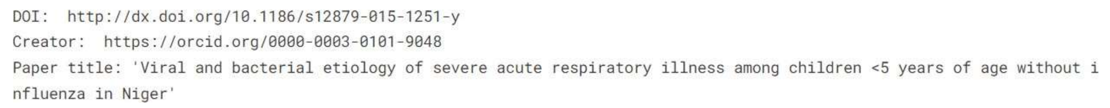
```
node_id = 0
# DOI
paper_doi = vnames[str(node_id)]
print("DOI: ", paper_doi)

# Paper reference in the knowledge graph based on the DOI
paper_ref = rdflib.URIRef(paper_doi)

# Authors
pred_firstname = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/firstName')
paper_author_fn = list(g.triples((paper_ref, pred_firstname, None)))
if paper_author_fn:
    paper_author_fn = paper_author_fn[0][2]

pred_surname = rdflib.term.URIRef('http://xmlns.com/foaf/0.1/surname')
paper_author_sn = list(g.triples((paper_ref, pred_surname, None)))
if paper_author_sn:
    paper_author_sn = paper_author_sn[0][2]

if paper_author_fn or paper_author_sn:
    print("Author(s): %s, %s" % (paper_author_fn, paper_author_sn))
```



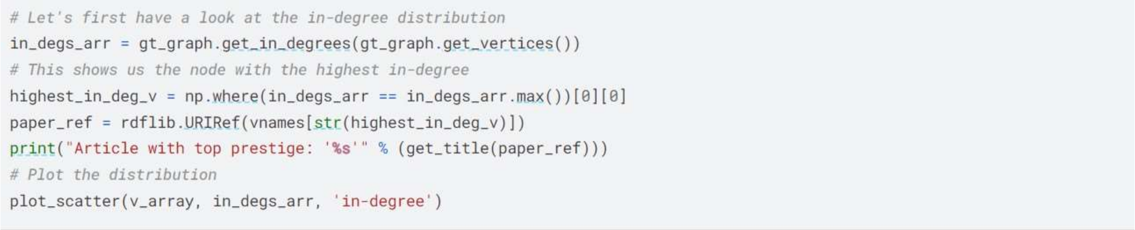
```
pred_creator = rdflib.term.URIRef('http://purl.org/spar/pro/creator')
paper_creator = list(g.triples((paper_ref, pred_creator, None)))
if paper_creator:
    paper_creator = paper_creator[0][2]
    print("Creator: ", paper_creator)
```



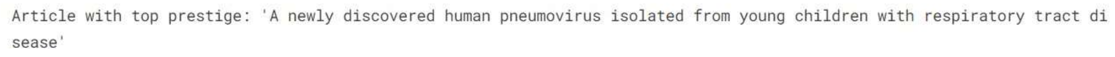
```
pred_publisher = rdflib.term.URIRef('https://www.ica.org/standards/RiC/ontology#publishedBy')
paper_publisher = list(g.triples((paper_ref, pred_publisher, None)))
if paper_publisher:
    paper_publisher = paper_publisher[0][2]
    print("Publisher: ", paper_publisher)

# Title
# Here we define a function that we will use later in this notebook
def get_title(paper_ref):
    pred_title = rdflib.term.URIRef('http://purl.org/dc/terms/title')
    paper_title = list(g.triples((paper_ref, pred_title, None)))
    if paper_title:
        paper_title = paper_title[0][2]
        return paper_title
    return None

print("Paper title: '%s'" % get_title(paper_ref))
```



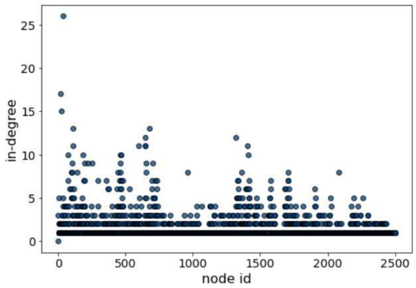
```
DOI:  http://dx.doi.org/10.1186/s12879-015-1251-y
Creator:  https://orcid.org/0000-0003-0101-9048
Paper title: 'Viral and bacterial etiology of severe acute respiratory illness among children <5 years of age without i
nfluenza in Niger'
```

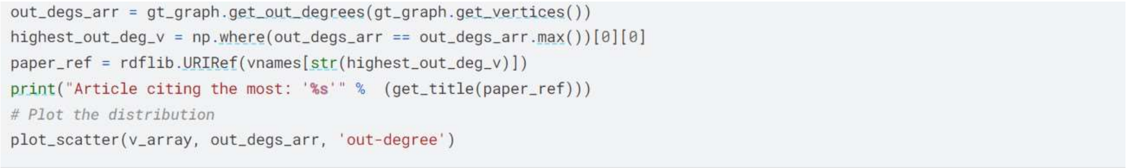


```
# Let's first have a look at the in-degree distribution
in_degs_arr = gt_graph.get_in_degrees(gt_graph.get_vertices())
# This shows us the node with the highest in-degree
highest_in_deg_v = np.where(in_degs_arr == in_degs_arr.max())[0][0]
paper_ref = rdflib.URIRef(vnames[str(highest_in_deg_v)])
print("Article with top prestige: '%s'" % (get_title(paper_ref)))
# Plot the distribution
plot_scatter(v_array, in_degs_arr, 'in-degree')
```



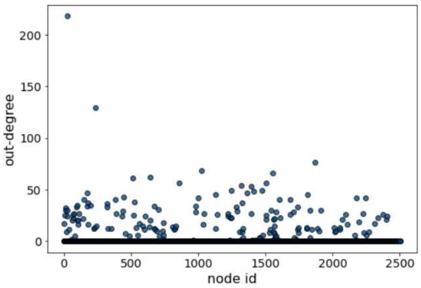
```
Article with top prestige: 'A newly discovered human pneumovirus isolated from young children with respiratory tract di
sease'
```

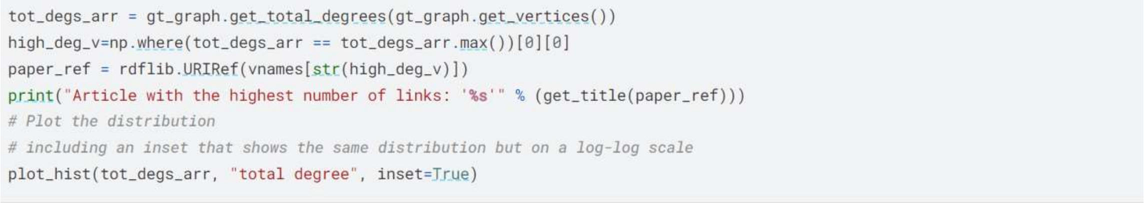




```
out_degs_arr = gt_graph.get_out_degrees(gt_graph.get_vertices())
highest_out_deg_v = np.where(out_degs_arr == out_degs_arr.max())[0][0]
paper_ref = rdflib.URIRef(vnames[str(highest_out_deg_v)])
print("Article citing the most: '%s'" %  (get_title(paper_ref)))
# Plot the distribution
plot_scatter(v_array, out_degs_arr, 'out-degree')
```

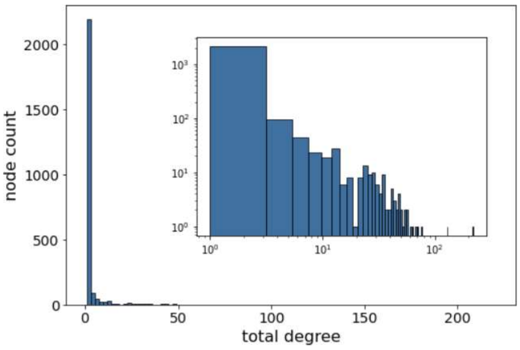
```
Article citing the most: 'Human rhinoviruses: the cold wars resume'
```

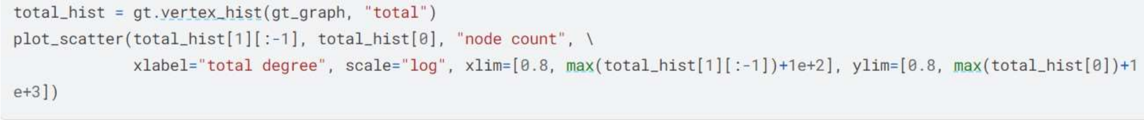




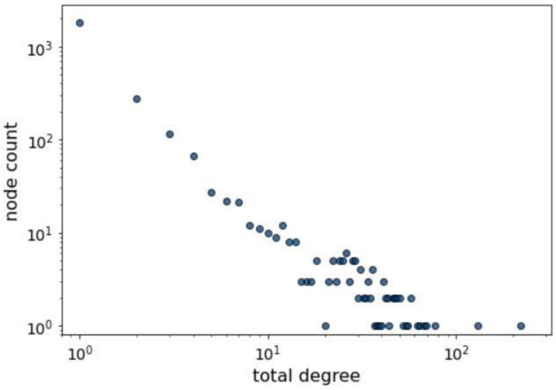
```
tot_degs_arr = gt_graph.get_total_degrees(gt_graph.get_vertices())
high_deg_v=np.where(tot_degs_arr == tot_degs_arr.max())[0][0]
paper_ref = rdflib.URIRef(vnames[str(high_deg_v)])
print("Article with the highest number of links: '%s'" % (get_title(paper_ref)))
# Plot the distribution
# including an inset that shows the same distribution but on a log-log scale
plot_hist(tot_degs_arr, "total degree", inset=True)
```

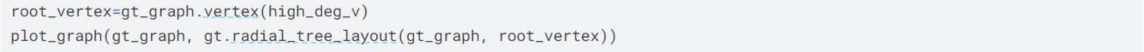
```
Article with the highest number of links: 'Human rhinoviruses: the cold wars resume'
```



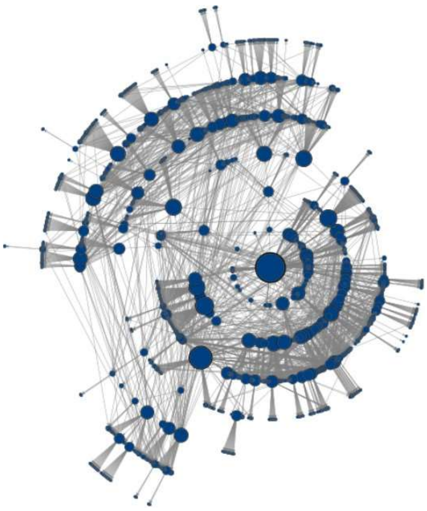


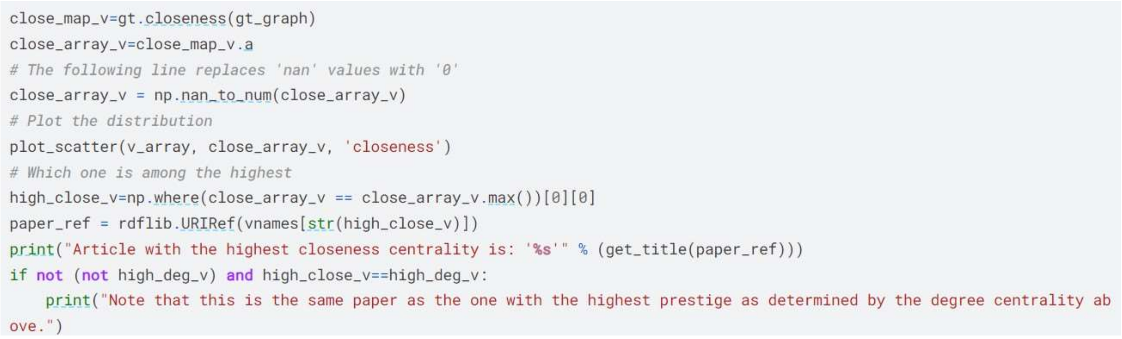
```
total_hist = gt.vertex_hist(gt_graph, "total")
plot_scatter(total_hist[1][:-1], total_hist[0], "node count", \
          xlabel="total degree", scale="log", xlim=[0.8, max(total_hist[1][:-1])+1e+2], ylim=[0.8, max(total_hist[0])+1
e+3])
```



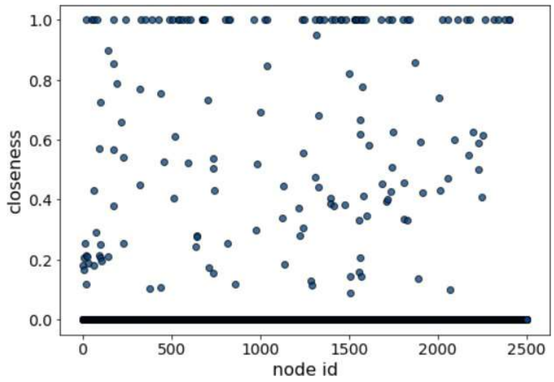


```
root_vertex=gt_graph.vertex(high_deg_v)
plot_graph(gt_graph, gt.radial_tree_layout(gt_graph, root_vertex))
```



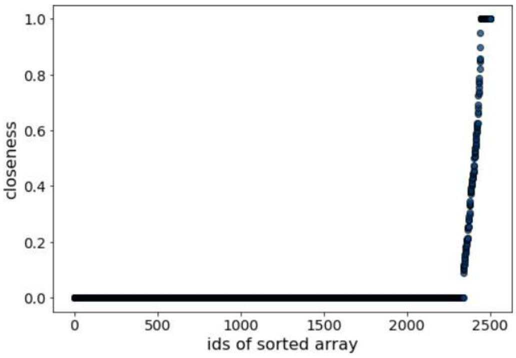


```
close_map_v=gt.closeness(gt_graph)
close_array_v=close_map_v.a
# The following line replaces 'nan' values with '0'
close_array_v = np.nan_to_num(close_array_v)
# Plot the distribution
plot_scatter(v_array, close_array_v, 'closeness')
# Which one is among the highest
high_close_v=np.where(close_array_v == close_array_v.max())[0][0]
paper_ref = rdflib.URIRef(vnames[str(high_close_v)])
print("Article with the highest closeness centrality is: '%s'" % (get_title(paper_ref)))
if not (not high_deg_v) and high_close_v==high_deg_v:
    print("Note that this is the same paper as the one with the highest prestige as determined by the degree centrality ab
ove.")
```

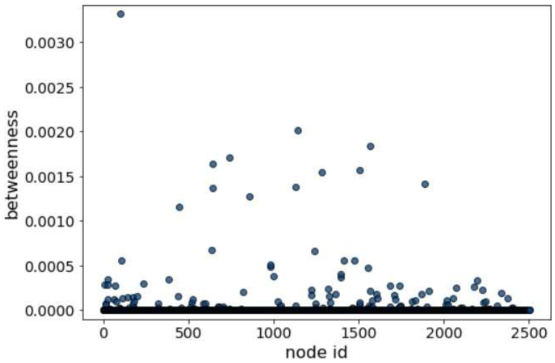


```
Article with the highest closeness centrality is: 'Viruses and bacteria in sputum samples of children with community-ac
quired pneumoniae'
```

```
close_array_v_sorted=np.sort(close_array_v)
# Plotting in ascending order, we can see that the fast majority of nodes has closeness centrality values close to zero
plot_scatter(v_array, close_array_v_sorted, 'closeness', xlabel='ids of sorted array')
```

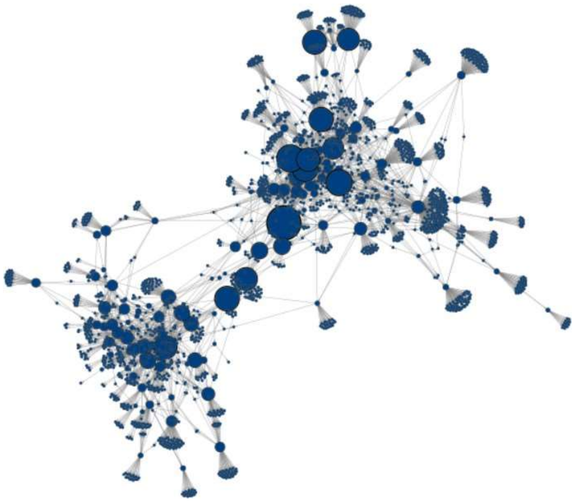


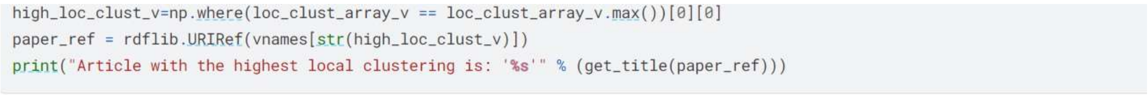
```
betw_map_v, betw_map_e=gt.betweenness(gt_graph)
betw_array_v=betw_map_v.a

plot_scatter(v_array, betw_array_v, 'betweenness', ylim=[min(betw_array_v)-1e-4,max(betw_array_v)+1e-4])

# Get one among the highest
high_betw_v=np.where(betw_array_v == betw_array_v.max())[0][0]
paper_ref = rdflib.URIRef(vnames[str(high_betw_v)])
print("Article with the highest betweenness centrality is: '%s'" % (get_title(paper_ref)))
```



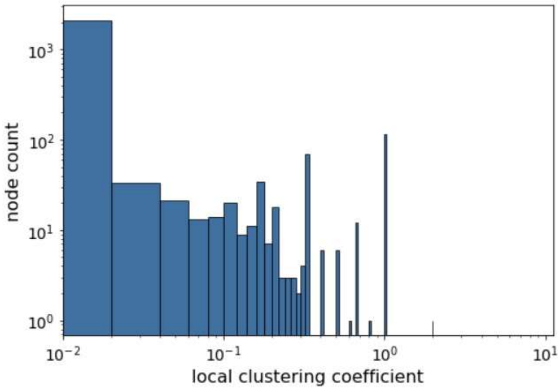
```
Article with the highest betweenness centrality is: 'A novel pancoronavirus RT-PCR assay: frequent detection of human c
oronavirus NL63 in children hospitalized with respiratory tract infections in Belgium'
```

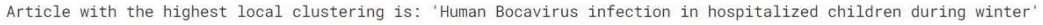
```
# Get the betweenness values
vmap=betw_map_v.copy()
# Scale them for a more insightful visualization
vmap.a = 500 * (np.sqrt(vmap.a) + 0.005)
# Draw the graph
plot_graph(gt_graph, gt.fruchterman_reingold_layout(gt_graph), vmap=vmap)
```



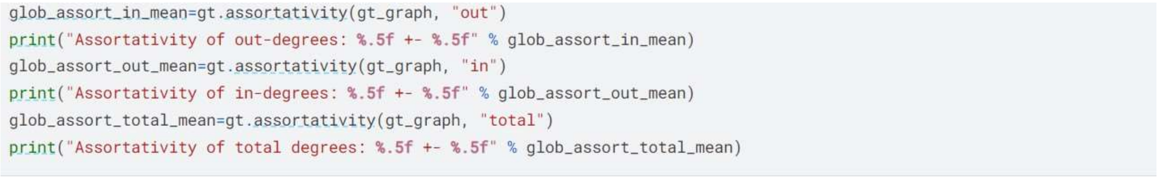


```
high_loc_clust_v=np.where(loc_clust_array_v == loc_clust_array_v.max())[0][0]
paper_ref = rdflib.URIRef(vnames[str(high_loc_clust_v)])
print("Article with the highest local clustering is: '%s'" % (get_title(paper_ref)))
```

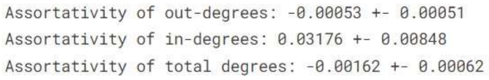




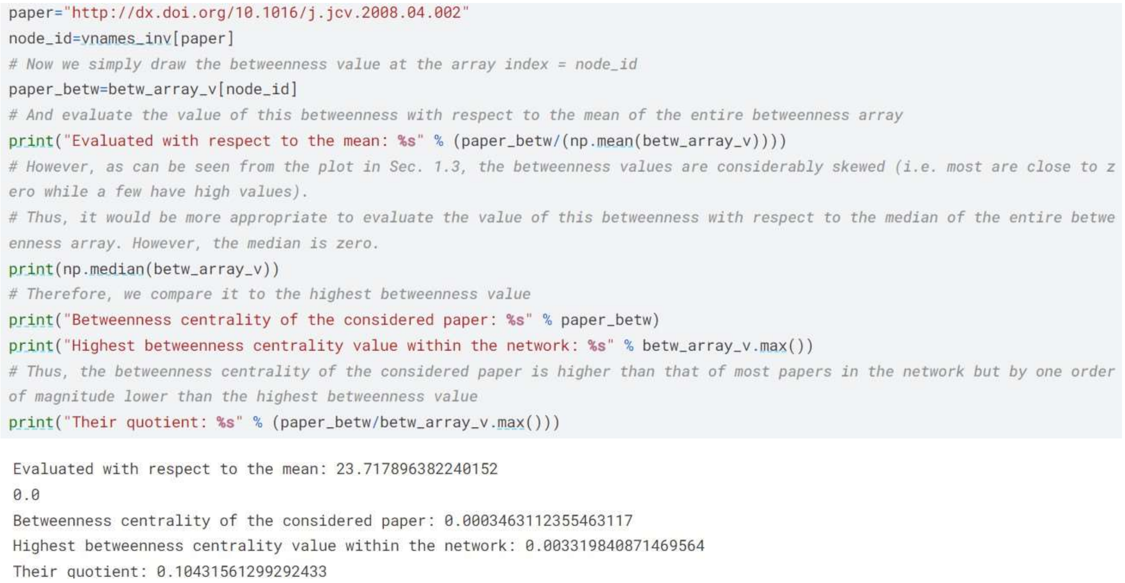
```
Article with the highest local clustering is: 'Human Bocavirus infection in hospitalized children during winter'
```



```
glob_assort_in_mean=gt.assortativity(gt_graph, "out")
print("Assortativity of out-degrees: %.5f +- %.5f" % glob_assort_in_mean)
glob_assort_out_mean=gt.assortativity(gt_graph, "in")
print("Assortativity of in-degrees: %.5f +- %.5f" % glob_assort_out_mean)
glob_assort_total_mean=gt.assortativity(gt_graph, "total")
print("Assortativity of total degrees: %.5f +- %.5f" % glob_assort_total_mean)
```



```
Assortativity of out-degrees: -0.00053 +- 0.00051
Assortativity of in-degrees: 0.03176 +- 0.00848
Assortativity of total degrees: -0.00162 +- 0.00062
```



```
paper="http://dx.doi.org/10.1016/j.jcv.2008.04.002"
node_id=vnames_inv[paper]
# Now we simply draw the betweenness value at the array index = node_id
paper_betw=betw_array_v[node_id]
# And evaluate the value of this betweenness with respect to the mean of the entire betweenness array
print("Evaluated with respect to the mean: %s" % (paper_betw/(np.mean(betw_array_v))))
# However, as can be seen from the plot in Sec. 1.3, the betweenness values are considerably skewed (i.e. most are close to z
ero while a few have high values).
# Thus, it would be more appropriate to evaluate the value of this betweenness with respect to the median of the entire betwe
enness array. However, the median is zero.
print(np.median(betw_array_v))
# Therefore, we compare it to the highest betweenness value
print("Betweenness centrality of the considered paper: %s" % paper_betw)
print("Highest betweenness centrality value within the network: %s" % betw_array_v.max())
# Thus, the betweenness centrality of the considered paper is higher than that of most papers in the network but by one order
of magnitude lower than the highest betweenness value
print("Their quotient: %s" % (paper_betw/betw_array_v.max()))
```

```
Evaluated with respect to the mean: 23.717896382240152
0.0
Betweenness centrality of the considered paper: 0.0003463112355463117
Highest betweenness centrality value within the network: 0.003319840871469564
Their quotient: 0.10431561299292433
```

Conclusion:

In conclusion, network analysis emerges as a powerful methodology for unraveling intricate systems across diverse domains, including the analysis of COVID-19 literature. Through its adeptness in unveiling hidden patterns, identifying central elements, and elucidating emergent properties, network analysis offers invaluable insights into the structure and dynamics of complex datasets. By leveraging this approach in literature analysis, researchers can uncover influential research papers, key authors, and thematic clusters, thereby enriching our understanding of the pandemic's scholarly discourse. Network analysis serves as a versatile framework for knowledge discovery and facilitates informed decision-making processes.