```
from google.colab import drive
drive.mount('/content/gdrive/')
```

```
Drive already mounted at /content/gdrive/; to attempt to forcibly remount, call drive.mount("/content/gdrive/", force_remount=True).
```

```
import os
import sys
dir_path = '/content/gdrive/MyDrive/Colab Notebooks/dataset'
sys.path.append(dir_path)
```

```
!pip install geopandas matplotlib networkx  numpy  pandas seaborn
```

```
Requirement already satisfied: geopandas in /usr/local/lib/python3.10/dist-packages (0.13.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.7.1)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (3.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.23.5)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (1.5.3)
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-packages (0.12.2)
Requirement already satisfied: fiona>=1.8.19 in /usr/local/lib/python3.10/dist-packages (from geopandas) (1.9.4.post1)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from geopandas) (23.1)
Requirement already satisfied: pyproj>=3.0.1 in /usr/local/lib/python3.10/dist-packages (from geopandas) (3.6.0)
Requirement already satisfied: shapely>=1.7.1 in /usr/local/lib/python3.10/dist-packages (from geopandas) (2.0.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.1.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (4.42.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.5)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2023.3.post1)
Requirement already satisfied: attrs>=19.2.0 in /usr/local/lib/python3.10/dist-packages (from fiona>=1.8.19->geopandas) (23.1.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from fiona>=1.8.19->geopandas) (2023.7.22)
Requirement already satisfied: click~=8.0 in /usr/local/lib/python3.10/dist-packages (from fiona>=1.8.19->geopandas) (8.1.7)
Requirement already satisfied: click-plugins>=1.0 in /usr/local/lib/python3.10/dist-packages (from fiona>=1.8.19->geopandas) (1.1.1)
Requirement already satisfied: cligj>=0.5 in /usr/local/lib/python3.10/dist-packages (from fiona>=1.8.19->geopandas) (0.7.2)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from fiona>=1.8.19->geopandas) (1.16.0)
```

```
# including modules
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import datetime
import seaborn as sns
import operator
import numpy as np
import geopandas as gp
%load_ext autoreload
%autoreload 2
sys.path.insert(0,'/content/gdrive/MyDrive/Colab Notebooks/utils')
import airtraffic_helpers
import networkx as nx
import community
import random
from shapely.geometry import Point
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

```
# Dometics airport details
airports_df = pd.read_csv("/content/gdrive/MyDrive/Colab Notebooks/dataset/288804893_T_MASTER_CORD.csv")
print('Shape of the dataframe:',airports_df.shape,'/n')
print('Printing one record:',airports_df[:1].T)
```

```
Shape of the dataframe: (18101, 28) /n
Printing one record:                                                    0
AIRPORT_ID                                      10001
AIRPORT                                           01A
DISPLAY_AIRPORT_NAME            Afognak Lake Airport
DISPLAY_AIRPORT_CITY_NAME_FULL      Afognak Lake, AK
AIRPORT_WAC                                        1
AIRPORT_COUNTRY_NAME                   United States
AIRPORT_COUNTRY_CODE_ISO                          US
AIRPORT_STATE_NAME                            Alaska
AIRPORT_STATE_CODE                                AK
AIRPORT_STATE_FIPS                               2.0
CITY_MARKET_ID                                 30001
```

```
DISPLAY_CITY_MARKET_NAME_FULL       Afognak Lake, AK
CITY_MARKET_WAC                                1
LAT_DEGREES                                 58.0
LAT_HEMISPHERE                                 N
LAT_MINUTES                                  6.0
LAT_SECONDS                                 34.0
LATITUDE                                58.109444
LON_DEGREES                                152.0
LON_HEMISPHERE                                 W
LON_MINUTES                                 54.0
LON_SECONDS                                 24.0
LONGITUDE                             -152.906667
AIRPORT_START_DATE                     2007-07-01
AIRPORT_THRU_DATE                             NaN
AIRPORT_IS_CLOSED                               0
AIRPORT_IS_LATEST                               1
Unnamed: 27                                   NaN
```

```
# Dometics airlines connections
trips_df = pd.read_csv("/content/gdrive/MyDrive/Colab Notebooks/dataset/288798530_T_T100D_MARKET_ALL_CARRIER.csv")
print('Shape of the dataframe:',trips_df.shape,'/n')
print('Printing one record:',trips_df[:1].T)
```

```
Shape of the dataframe: (41930, 8) /n
Printing one record:                                        0
UNIQUE_CARRIER                       04Q
UNIQUE_CARRIER_NAME  Tradewind Aviation
ORIGIN_AIRPORT_ID                  13535
ORIGIN                               MVL
DEST_AIRPORT_ID                    12197
DEST                                 HPN
MONTH                                  1
Unnamed: 7                           NaN
```

```
# Extracting edges - we consider a connection from one airport to another as an edge.
# Note: these edges will be directed.
edges = list(zip(trips_df['ORIGIN_AIRPORT_ID'],trips_df['DEST_AIRPORT_ID']))
```

```
# creating directed and undirected graphs based on airports and their connections
G = nx.DiGraph()
G.add_edges_from(edges)
G_undirected = nx.Graph()
G_undirected.add_edges_from(edges)
```

```
# printing the total number of nodes and edges of directed graph
print('Total number of airports:',len(list(G.nodes)))
print('Total number of connections:',len(list(G.edges)))
```

```
Total number of airports: 894
Total number of connections: 13760
```

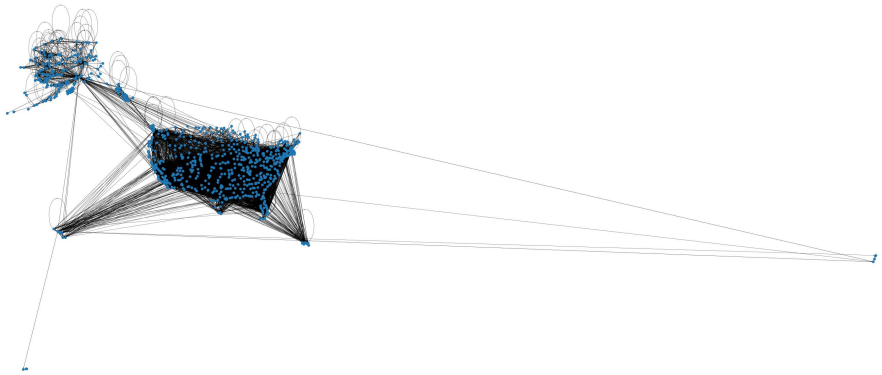
```
# creating a GeoDataframe for plotting airports on a map
airport_ids = list(G.nodes)
edgelist = list(G.edges)
geo_stations = airtraffic_helpers.get_geodataframe_airports(airports_df,airport_ids)
```

```
# reading the shape file of US
shp_us = gp.GeoDataFrame.from_file("/content/gdrive/MyDrive/Colab Notebooks/dataset/Igismap/Alabama_AL4_US_Poly.shp")
shp_us=shp_us.to_crs({'init':'epsg:4326'})
shp_us.plot(figsize=(100,100),color='g',alpha=0.75)
```

```
/usr/local/lib/python3.10/dist-packages/pyproj/crs/crs.py:141: FutureWarning: '+init=<authority>:<code>
  in_crs_string = _prepare_from_proj_string(in_crs_string)
<Axes: >
```

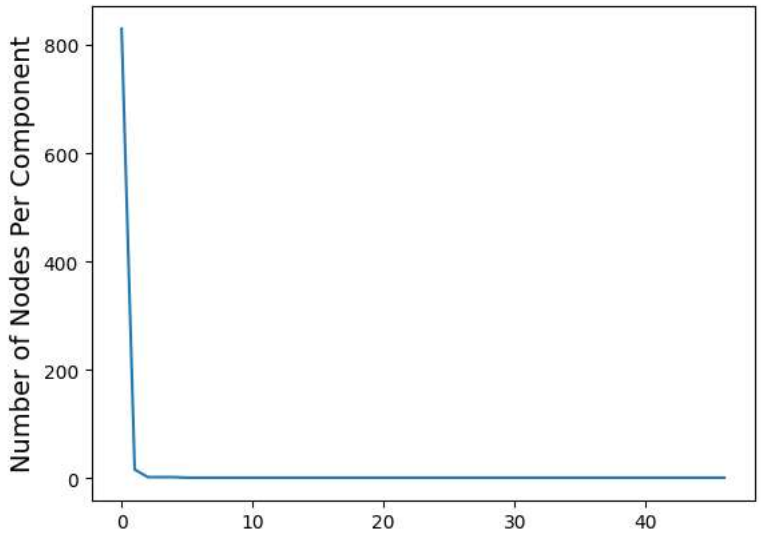
```
# plotting the nodes in the shape file of US
plt.style.use("default")
%matplotlib inline
fig, ax = plt.subplots(1,1,figsize=(50,50))
base = shp_us.plot(ax=ax, color='gray', alpha=0.2)
geo_stations.plot(ax=base, marker="o", color="r", markersize=10,alpha=0.8, zorder=0)
_ = ax.axis('off')
ax.set_title("Plot of airports in United States",fontsize=20)
fig.tight_layout()
fig.savefig("/content/gdrive/MyDrive/Colab Notebooks/figures/us_airports_nodes.pdf")
```


```
# displaying the network based on actual geographical placement of nodes
position_dict = {} # will contain the node coordinates
for item in list(geo_stations.station_ids):
  position_dict[item] = geo_stations[geo_stations.station_ids==item].geometry.values[0].x,geo_stations[geo_stations.station_ids==item].geomet
fig, ax = plt.subplots(1,1,figsize=(100,40))
nx.draw(G,pos=position_dict)
fig.tight_layout()
fig.savefig("/content/gdrive/MyDrive/Colab Notebooks/figures/us_airports_network_withposition.pdf")
```



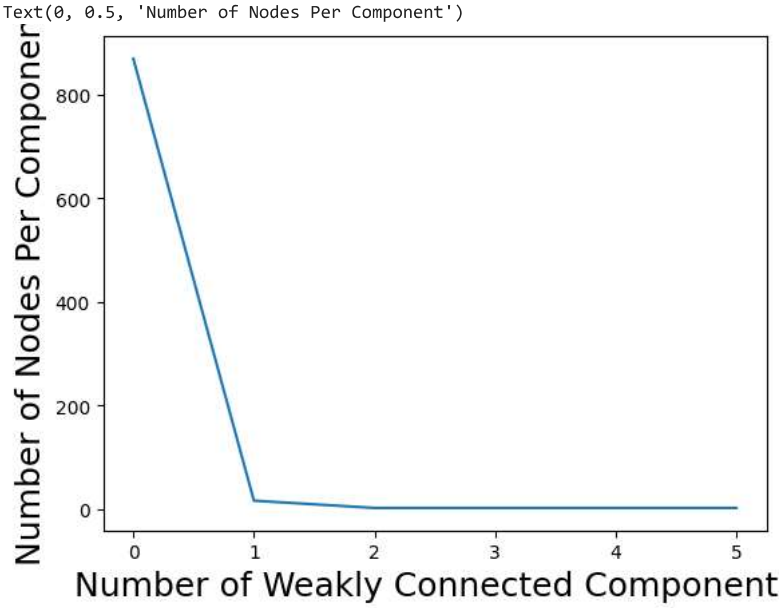
```
#Strongly Connected Components
scc=[list(l) for l in nx.strongly_connected_components(G)]  #Strongly Connected Components
print("Number of Strongly Connected Components",len(scc),"/nSample Strongly Connected Components",scc[:3])
plt.plot(list(sorted(map(lambda x: len(x),scc),reverse=True)))
plt.xlabel("Number of Strongly Connected Components",fontsize=14)
plt.ylabel("Number of Nodes Per Component",fontsize=14)
```

```
Number of Strongly Connected Components 47 /nSample Strongly Connected Components [[12222], [12583], [1
Text(0, 0.5, 'Number of Nodes Per Component')
```




```
# Weakly Connected Components
wcc=[list(l) for l in nx.weakly_connected_components(G)]  #Strongly Connected Components
print("Number of Weakly Connected Components",len(wcc),"/nSample Weakly Connected Components",wcc[:3])
plt.plot(list(sorted(map(lambda x: len(x),wcc),reverse=True)))
plt.xlabel("Number of Weakly Connected Components",fontsize=18)
plt.ylabel("Number of Nodes Per Component",fontsize=18)
```

```
Number of Weakly Connected Components 6 /nSample Weakly Connected Components [[10241, 10243, 10245, 163
Text(0, 0.5, 'Number of Nodes Per Component')
```



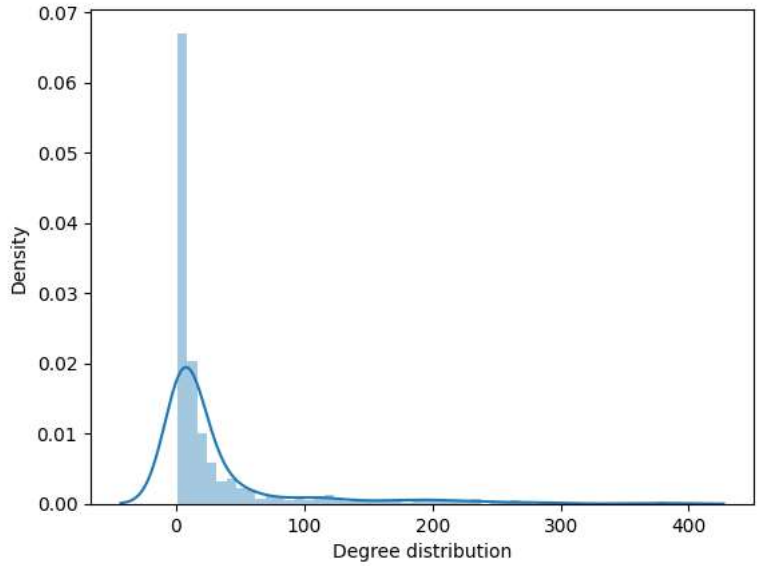

```
# node degree calculation
node_indegrees=[item for item in dict(G.in_degree()).items()]
node_outdegrees=[item for item in dict(G.out_degree()).items()]
sorted_indegrees=sorted(node_indegrees,key=operator.itemgetter(1),reverse=True)
sorted_outdegrees=sorted(node_outdegrees,key=operator.itemgetter(1),reverse=True)

print("Sample Indegree List",node_indegrees[:5],"/n")
print("Sorted In Decreasing Order of Indegrees",sorted_indegrees[:5],"/n")
print("Sample Outdegree List",node_outdegrees[:5],"/n")
print("Sorted In Decreasing Order of Outdegree",sorted_outdegrees[:5],"/n")

node_degrees=airtraffic_helpers.getdegree(G)
print("Sample degree list",node_degrees[:5],"/n")
sorted_degrees=sorted(node_degrees,key=operator.itemgetter(1),reverse=True)
print("Sorted in decreasing order of degrees",sorted_degrees[:5],"/n")
```

```
Sample Indegree List [(13535, 5), (12197, 58), (10792, 65), (10540, 39), (13303, 101)] /n
Sorted In Decreasing Order of Indegrees [(11292, 192), (11298, 192), (13930, 183), (10397, 174), (13244, 154)] /n
```

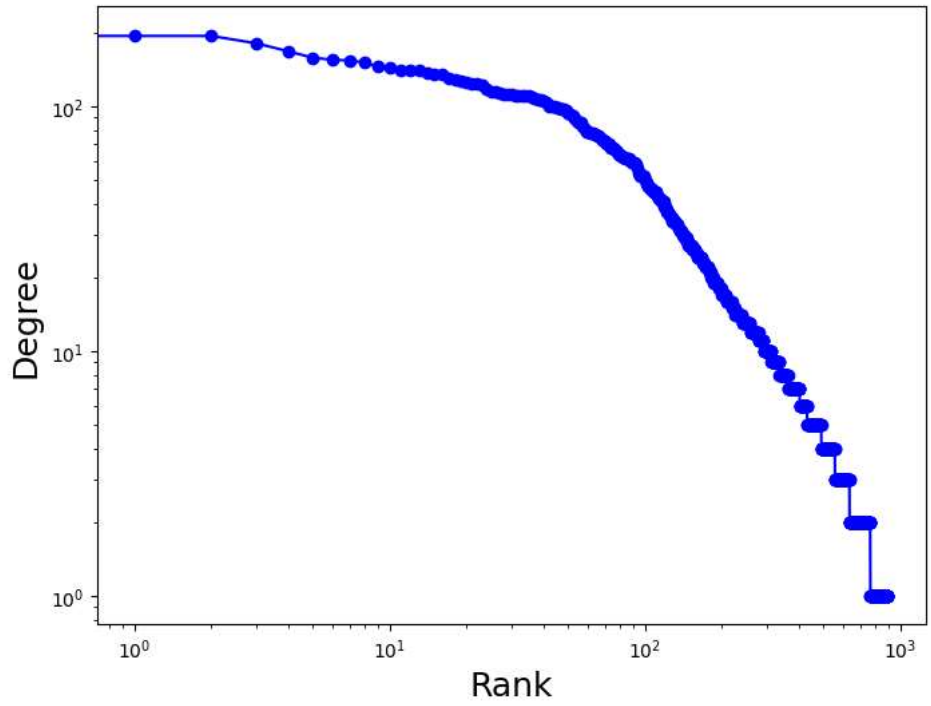
```
Sample Outdegree List [(13535, 2), (12197, 60), (10792, 61), (10540, 39), (13303, 103)] /n
Sorted In Decreasing Order of Outdegree [(11292, 191), (11298, 191), (13930, 188), (10397, 177), (13244, 148)] /n
Sample degree list [(13535, 7), (12197, 118), (10792, 126), (10540, 78), (13303, 204)] /n
Sorted in decreasing order of degrees [(11292, 383), (11298, 383), (13930, 371), (10397, 351), (13244, 302)] /n
```

```
# plotting the degree distribution
sns.distplot(pd.Series(np.array(node_degrees).T[1], name="Degree distribution"))
```

```
<ipython-input-27-44cc6e898ae4>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(pd.Series(np.array(node_degrees).T[1], name="Degree distribution"))
<Axes: xlabel='Degree distribution', ylabel='Density'>
```




```
# plotting the degree-rank plot
airtraffic_helpers.generate_degree_rank_plot(edgelist)
```



```
# find network density
print('Network density:',nx.density(G))
```

```
Network density: 0.017235721031838486
```

```
sub_graphs = [G_undirected.subgraph(c).copy() for c in nx.connected_components(G_undirected)]
print('Number of connected components: ',len([k for k in sub_graphs]))
```

```
Number of connected components:  6
```

```
# finding and plotting the giant connected component
Gc = max(sub_graphs, key=len)
Gc=nx.convert_node_labels_to_integers(Gc)
Gc.name='GCC'
print(Gc)
print('Diameter of the Giant connected component:',nx.diameter(Gc))
print('Average shortest path:',nx.average_shortest_path_length(Gc))
```

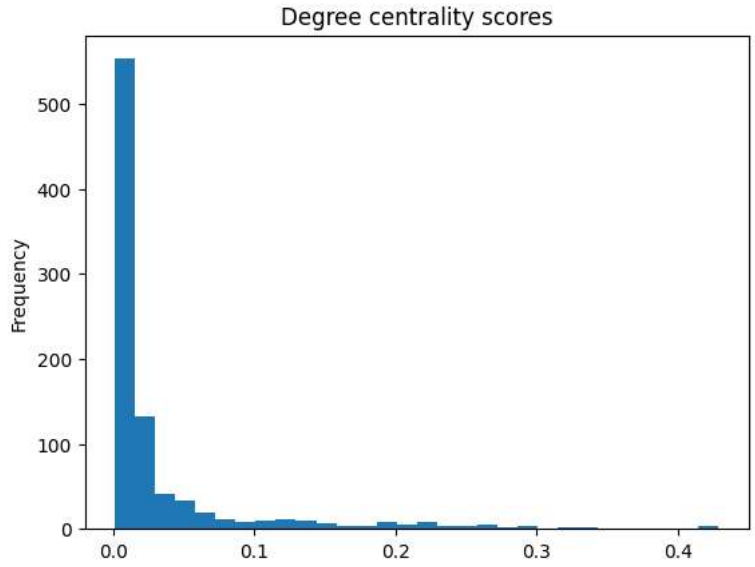
```
Graph named 'GCC' with 870 nodes and 8097 edges
Diameter of the Giant connected component: 10
Average shortest path: 3.1862068965517243
```

```
# plotting the GCC
airtraffic_helpers.plot_network(Gc,title="",edgealpha=0.08,node_dist=1,nodesize=20,savefig=False,figsize=(15,15))
```

```
1
```

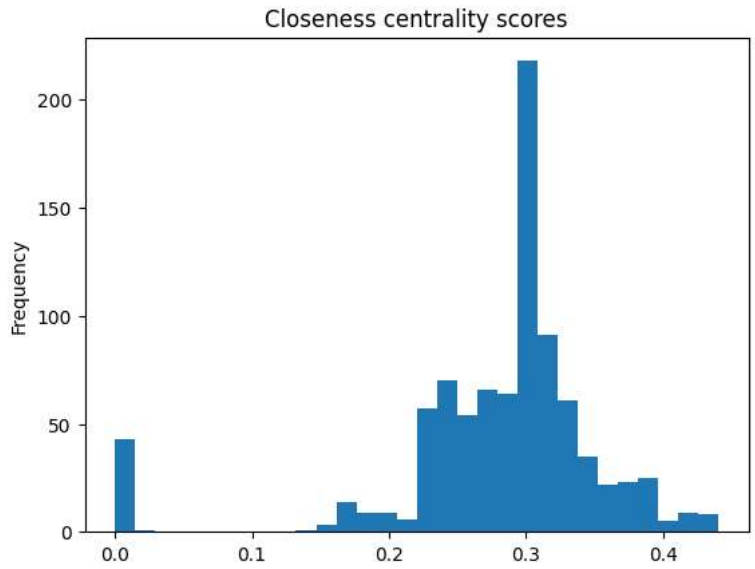
```
pd.Series(nx.degree_centrality(G)).sort_values().plot(kind='hist',bins=30,title="Degree centrality scores")
```

```
<Axes: title={'center': 'Degree centrality scores'}, ylabel='Frequency'>
```



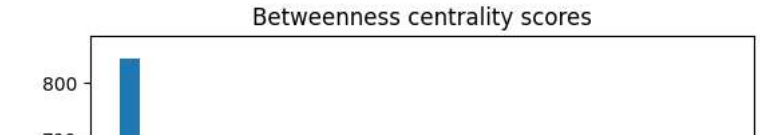

```
pd.Series(nx.closeness_centrality(G)).sort_values().plot(kind='hist',bins=30,title="Closeness centrality scores")
```

```
<Axes: title={'center': 'Closeness centrality scores'}, ylabel='Frequency'>
```



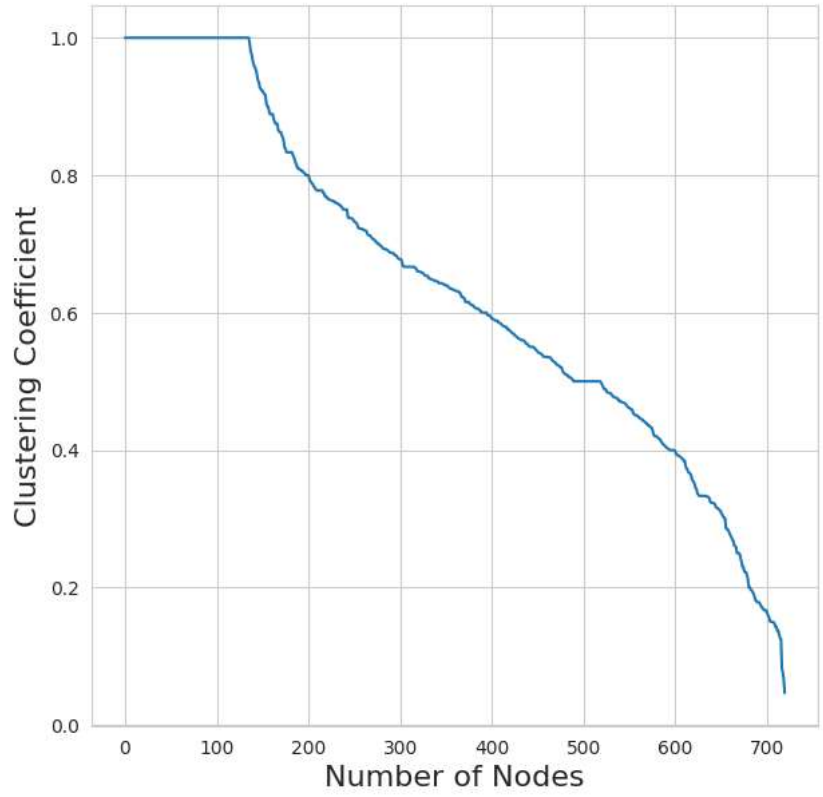

```
pd.Series(nx.betweenness_centrality(G)).sort_values().plot(kind='hist',bins=30,title="Betweenness centrality scores")
```

```
<Axes: title={'center': 'Betweenness centrality scores'}, ylabel='Frequency'>
```




```
# plotting average clustering coefficient
airtraffic_helpers.generate_clustering_coefficient_plot(G)
clust_coeff=nx.average_clustering(G)
print("Average Clustering Coefficient",round(clust_coeff,4))
```

```
Average Clustering Coefficient 0.5151
```




```
# As the network is large, we will randomly select 400 nodes for node centric community detection analysis
airportids_subset = random.sample(airport_ids,k=300)


# filter edges to contain nodes only from the above generated airports subset
edgelist_subset = []
for item in edgelist:
  if item[0] in airportids_subset and item[1] in airportids_subset:
    edgelist_subset.append(item)


G_subset_undirected = nx.Graph()
G_subset_undirected.add_edges_from(edgelist_subset)


# identifying the cliques in the network
cl=nx.enumerate_all_cliques(G_subset_undirected)
#print last 10 cliques
print([l for l in cl][-10:])
```

```
[[10792, 12953, 13930, 11697, 13204, 11618, 11259, 11292, 14635, 13244, 12451, 13296, 11193, 10529], [10792, 12264, 13930, 14524, 11697,
```

```
# printing the 5 largest cliques
print("5 Largest Cliques",sorted([l for l in nx.find_cliques(G_subset_undirected)],key=lambda x: len(x),reverse=True)[:5])
```

```
5 Largest Cliques [[11292, 13930, 13244, 13204, 12953, 10529, 11193, 11618, 10792, 11259, 12451, 12264, 11697, 14635, 13296], [11292, 13
```

```
# printing cliques containing Node ID 14107
print(nx.cliques_containing_node(G_subset_undirected,13495))
```

```
[]
<ipython-input-44-735625c6a525>:2: DeprecationWarning:

cliques_containing_node is deprecated and will be removed.
Use the result of find_cliques directly to compute maximal cliques
containing each node:

    {n: [c for c in nx.find_cliques(G) if n in c] for n in G}


  print(nx.cliques_containing_node(G_subset_undirected,13495))
```

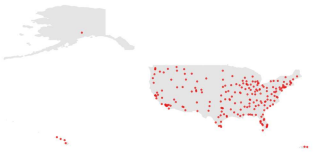
```
# Find K cliques using percolation method. K=10
print([l for l in nx.algorithms.community.k_clique_communities(G_subset_undirected,10)])
```

```
[frozenset({13891, 13204, 12953, 11292, 11996, 10529, 11618, 12451, 12264, 10792, 13930, 14635, 13485, 10800, 11697, 13296, 14771, 1452
```

```
# finding k-core, in this case k=12
G_noselfloop = G_undirected
G_noselfloop.remove_edges_from(nx.selfloop_edges(G_noselfloop))
g_k=nx.k_core(G_noselfloop,k=12)
airtraffic_helpers.plot_network(g_k,node_dist=4,figsize=(12,12),nodesize=10)
```

```
1
```

```
#Code to generate the shape file and station k_core visualization.
labels=g_k.nodes()
fig, ax = plt.subplots(1,1,figsize=(100,40))
base = shp_us.plot(ax=ax, color='gray', alpha=0.2)
geo_stations[geo_stations.station_ids.isin(labels)].plot(ax=base, marker="o", color="r", markersize=100,alpha=0.8, zorder=0)
_ = ax.axis('off')
```



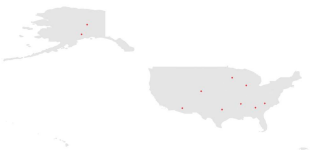
```
pg_rank=sorted([l for l in nx.pagerank(G).items()],key=lambda x: x[1],reverse=True)
print("Top 10 Stations By Pagerank",pg_rank[:10])
```

```
Top 10 Stations By Pagerank [(11292, 0.01221494725476268), (11298, 0.011381590445025938), (13930, 0.00997962312402018), (10299, 0.009395
```

```
hubs,authorities=nx.hits(G)
hubs=sorted([l for l in hubs.items()],key=lambda x: x[1],reverse=True)
authorities=sorted([l for l in authorities.items()],key=lambda x: x[1],reverse=True)
print("Top 10 Biggest Hubs",hubs[:10])
print("/nTop 10 Biggest Authorities",authorities[:10])
```

```
Top 10 Biggest Hubs [(10397, 0.011091413715196816), (11292, 0.010867112430822475), (13930, 0.010807784511584905), (11298, 0.01072620103
/nTop 10 Biggest Authorities [(11292, 0.010985791978555702), (10397, 0.01086412335941236), (11298, 0.010761317526652964), (13930, 0.0106
```

```
# Top 10 pagerank nodes visualization
labels=list(map(lambda x: x[0],pg_rank[:10]))
fig, ax = plt.subplots(1,1,figsize=(100,40))
base = shp_us.plot(ax=ax, color='gray', alpha=0.2)
geo_stations[geo_stations.station_ids.isin(labels)].plot(ax=base, marker="o", color="r", markersize=100,alpha=0.8, zorder=0)
_ = ax.axis('off')
fig.tight_layout()
fig.savefig("/content/gdrive/MyDrive/Colab Notebooks/figures/pagerank.pdf")
```



```
# Top 10 hubs nodes visualization
labels=list(map(lambda x: x[0],hubs[:10]))
fig, ax = plt.subplots(1,1,figsize=(100,40))
base = shp_us.plot(ax=ax, color='gray', alpha=0.2)
geo_stations[geo_stations.station_ids.isin(labels)].plot(ax=base, marker="o", color="r", markersize=100,alpha=0.8, zorder=0)
_ = ax.axis('off')
fig.tight_layout()
fig.savefig("/content/gdrive/MyDrive/Colab Notebooks/figures/hubs.pdf")
```

```
# Top 10 authorities nodes visualization
labels=list(map(lambda x: x[0],authorities[:10]))
fig, ax = plt.subplots(1,1,figsize=(100,40))
base = shp_us.plot(ax=ax, color='gray', alpha=0.2)
geo_stations[geo_stations.station_ids.isin(labels)].plot(ax=base, marker="o", color="r", markersize=100,alpha=0.8, zorder=0)
_ = ax.axis('off')
fig.tight_layout()
fig.savefig("/content/gdrive/MyDrive/Colab Notebooks/figures/authorities.pdf")
```