Results_for_absolute_change_2015_2020

```
pip install powerlaw
```

```
pip install seaborn
```

```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt
import networkx as nx

# Load the world map
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))

# Load your datasets
nodes_df = pd.read_csv('Countries - Sheet2.csv')
edges_df = pd.read_csv('MIGRATION_2015_2020.csv')

# Create the graph
G = nx.DiGraph()

# Add nodes with positions
for _, row in nodes_df.iterrows():
    G.add_node(row['Country'], latitude=row['Latitude'], longitude=row['Longitude'])

# Add edges
for _, row in edges_df.iterrows():
    country1 = row['Country 1']
    country2 = row['Country 2']
    weight = int(row['Absolute Change'].replace(',', ''))  # Assuming '2020' column is of string type and has commas
    if weight > 0 and country1 in G.nodes and country2 in G.nodes:
        G.add_edge(country1, country2, weight=weight)

# Calculate edge widths proportional to weights
edge_weights = [d['weight'] for u, v, d in G.edges(data=True)]
max_weight = max(edge_weights)
edge_widths = [w / max_weight * 25 for w in edge_weights]  # Adjust multiplier (5) to control edge width scaling
print(edge_widths)
```

```
# Calculate node degrees
degrees = dict(G.degree())
in_degrees = dict(G.in_degree())
out_degrees = dict(G.out_degree())
```

```python
# Calculate incoming population for each country
incoming_population = {}

for country in G.nodes:
    incoming_population[country] = sum(G[edge[0]][edge[1]]['weight'] for edge in G.in_edges(country))

print(incoming_population)
```

```
{'Afghanistan': 0, 'Albania': 0, 'Algeria': 10567, 'American Samoa': 284, 'Andorra': 3098, 'Angola': 27084,
```

```python
# Get top 20 countries with highest incoming population
top_20_countries = sorted(incoming_population.items(), key=lambda x: x[1], reverse=True)[:20]

print("Top 20 countries with highest incoming population:")
for country, population in top_20_countries:
    print(country, population)
```

```
Top 20 countries with highest incoming population:
Germany 5515421
United States 3677138
Saudi Arabia 2593018
Colombia 1745226
Turkey 1717982
Chile 1311961
Peru 1069825
Spain 1055371
United Kingdom 1050055
Australia 954630
Mexico 926099
Uganda 891085
Sudan 750265
United Arab Emirates 699267
France 675481
Bangladesh 670057
Canada 650887
Italy 621128
Qatar 538057
Oman 495628
```

```python
print(out_degrees)
```

```
{'Afghanistan': 51, 'Albania': 44, 'Algeria': 54, 'American Samoa': 2, 'Andorra': 14, 'Angola': 41, 'Anguilla': 7,
```

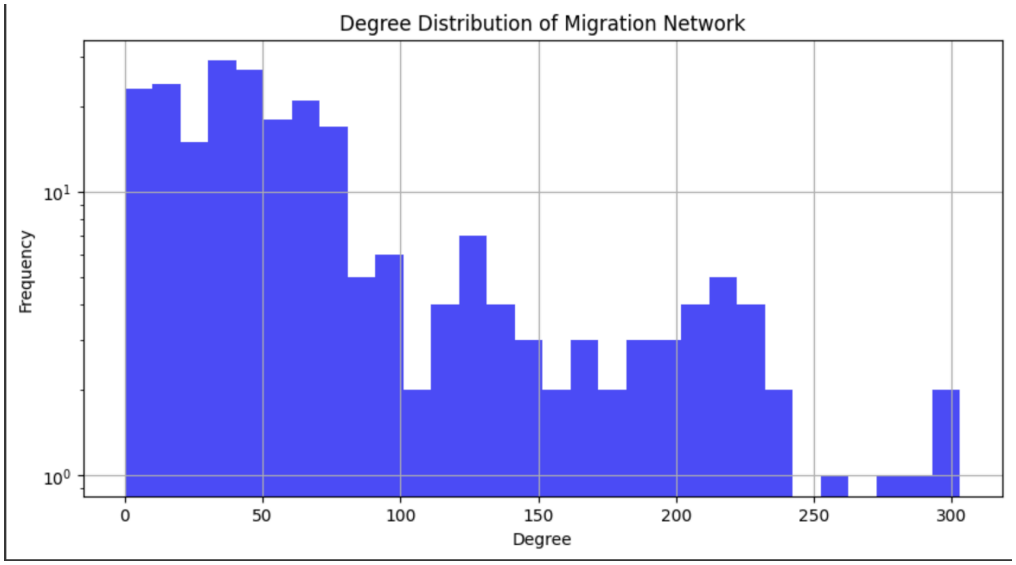
```python
# Calculate outgoing population for each country
outgoing_population = {}

for country in G.nodes:
    outgoing_population[country] = sum(G[edge[0]][edge[1]]['weight'] for edge in G.out_edges(country))

print(outgoing_population)
```

```
{'Afghanistan': 577289, 'Albania': 107631, 'Algeria': 188156, 'American Samoa': 196, 'Andorra': 2488, 'Angola': 55262,
```

```python
# Get top 20 countries with highest incoming population
top_20_countries = sorted(outgoing_population.items(), key=lambda x: x[1], reverse=True)[:20]

print("Top 20 countries with highest outgoing population:")
for country, population in top_20_countries:
    print(country, population)
```

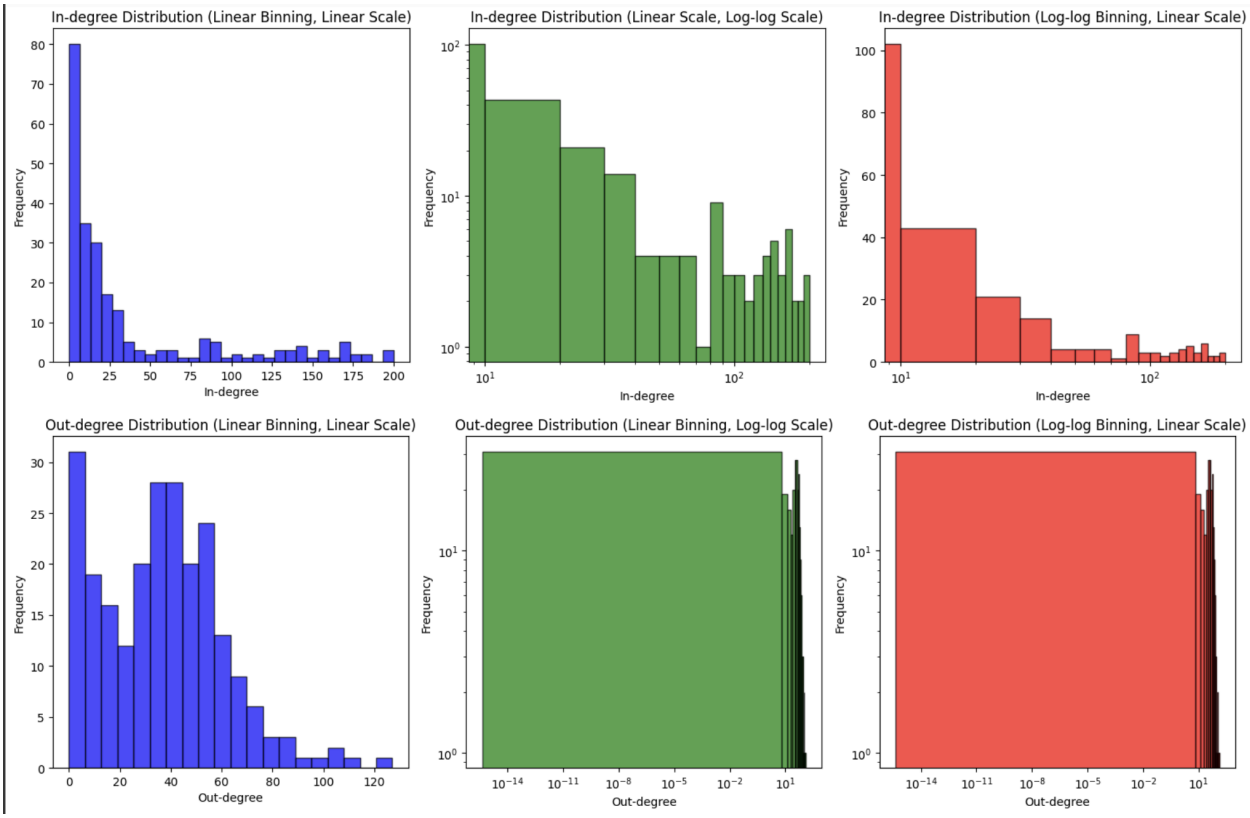
```
Top 20 countries with highest outgoing population:
Venezuela 4707546
Syria 2179074
India 2122877
South Sudan 1515390
Caribbean 959232
Myanmar 873424
Mexico 791371
Pakistan 789005
Poland 770103
Bangladesh 757395
China 721565
Italy 651110
Russia 639145
Romania 610200
Turkey 605833
Philippines 598166
Afghanistan 577289
Indonesia 546791
Bulgaria 520277
Vietnam 483248
```

```python
import numpy as np

# Get degrees of all nodes
node_degrees = [deg for node, deg in G.degree()]

# Plotting the degree distribution
plt.figure(figsize=(10, 5))
plt.hist(node_degrees, bins=30, color='blue', alpha=0.7)  # You can adjust the number of bins
plt.title("Degree Distribution of Migration Network")
plt.xlabel("Degree")
plt.ylabel("Frequency")
plt.yscale('log')  # Log scale may be appropriate if the distribution is heavy-tailed
plt.grid(True)
plt.show()
```



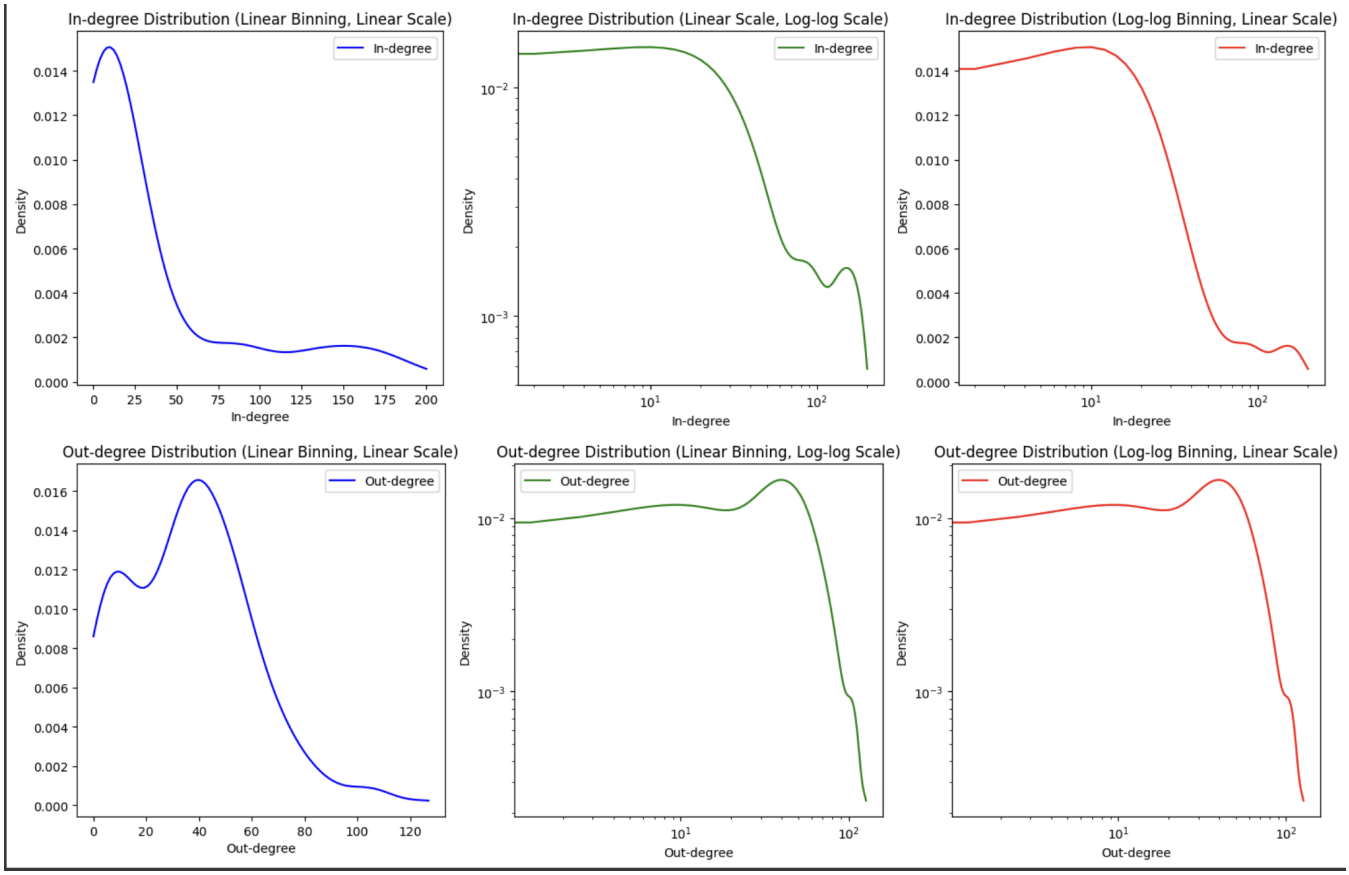
```python
# Calculate in-degree distribution
in_degree_val = [degree for node, degree in G.in_degree()]
# Calculate out-degree distribution
out_degree_val = [degree for node, degree in G.out_degree()]

# Linear binning with linear scale for in-degree
plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
plt.hist(in_degree_val, bins=30, alpha=0.7, color='blue', edgecolor='black')
plt.title('In-degree Distribution (Linear Binning, Linear Scale)')
plt.xlabel('In-degree')
plt.ylabel('Frequency')
# Log-log scale for in-degree
plt.subplot(1, 3, 2)
plt.hist(in_degree_val, bins=20, alpha=0.7, color='green', edgecolor='black')
plt.xscale('log')
plt.yscale('log')
plt.title('In-degree Distribution (Linear Scale, Log-log Scale)')
plt.xlabel('In-degree')
plt.ylabel('Frequency')
# Log-log binning with linear scale for in-degree
log_bins_in = 20
plt.subplot(1, 3, 3)
plt.hist(in_degree_val, bins=log_bins_in, alpha=0.7, color='red',edgecolor='black')
plt.xscale('log')
plt.title('In-degree Distribution (Log-log Binning, Linear Scale)')
plt.xlabel('In-degree')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```

```python
# Linear binning with linear scale for out-degree
plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
plt.hist(out_degree_val, bins=20, alpha=0.7, color='blue', edgecolor='black')
plt.title('Out-degree Distribution (Linear Binning, Linear Scale)')
plt.xlabel('Out-degree')
plt.ylabel('Frequency')
# Log-log scale for out-degree
plt.subplot(1, 3, 2)
plt.hist(out_degree_val, bins=20, alpha=0.7, color='green', edgecolor='black')
plt.xscale('log')
plt.yscale('log')
plt.title('Out-degree Distribution (Linear Binning, Log-log Scale)')
plt.xlabel('Out-degree')
plt.ylabel('Frequency')

# Log-log binning with linear scale for out-degree
plt.subplot(1,3,3)
log_bins_out = 20
plt.hist(out_degree_val, bins=log_bins_out, alpha=0.7, color='red',edgecolor='black')
plt.xscale('log')
plt.yscale('log')
plt.title('Out-degree Distribution (Log-log Binning, Linear Scale)')
plt.xlabel('Out-degree')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```




```python
from scipy.stats import gaussian_kde

# Function to plot smooth curve
def plot_smooth_curve(data, color, label):
    kde = gaussian_kde(data)
    x_vals = np.linspace(min(data), max(data), 100)
    plt.plot(x_vals, kde(x_vals), color=color, label=label)

# Linear binning with linear scale for in-degree
plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
plot_smooth_curve(in_degree_val, 'blue', 'In-degree')
plt.title('In-degree Distribution (Linear Binning, Linear Scale)')
plt.xlabel('In-degree')
plt.ylabel('Density')
plt.legend()
# Log-log scale for in-degree
plt.subplot(1, 3, 2)
plot_smooth_curve(in_degree_val, 'green', 'In-degree')
plt.xscale('log')
plt.yscale('log')
plt.title('In-degree Distribution (Linear Scale, Log-log Scale)')
plt.xlabel('In-degree')
plt.ylabel('Density')
plt.legend()
# Log-log binning with linear scale for in-degree
plt.subplot(1, 3, 3)
plot_smooth_curve(in_degree_val, 'red', 'In-degree')
plt.xscale('log')
plt.title('In-degree Distribution (Log-log Binning, Linear Scale)')
plt.xlabel('In-degree')
plt.ylabel('Density')
plt.legend()
plt.tight_layout()
plt.show()
```

```python
from scipy.stats import gaussian_kde

# Function to plot smooth curve
def plot_smooth_curve(data, color, label):
    kde = gaussian_kde(data)
    x_vals = np.linspace(min(data), max(data), 100)
    plt.plot(x_vals, kde(x_vals), color=color, label=label)

# Linear binning with linear scale for in-degree
plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
plot_smooth_curve(in_degree_val, 'blue', 'In-degree')
plt.title('In-degree Distribution (Linear Binning, Linear Scale)')
plt.xlabel('In-degree')
plt.ylabel('Density')
plt.legend()
# Log-log scale for in-degree
plt.subplot(1, 3, 2)
plot_smooth_curve(in_degree_val, 'green', 'In-degree')
plt.xscale('log')
plt.yscale('log')
plt.title('In-degree Distribution (Linear Scale, Log-log Scale)')
plt.xlabel('In-degree')
plt.ylabel('Density')
plt.legend()
# Log-log binning with linear scale for in-degree
plt.subplot(1, 3, 3)
plot_smooth_curve(in_degree_val, 'red', 'In-degree')
plt.xscale('log')
plt.title('In-degree Distribution (Log-log Binning, Linear Scale)')
plt.xlabel('In-degree')
plt.ylabel('Density')
plt.legend()
plt.tight_layout()
plt.show()
```



```python
from scipy.stats import kstest, poisson
import numpy as np
# Assuming lambda is the average degree
lambda_in = np.mean(in_degree_val)
lambda_out = np.mean(out_degree_val)
# Perform K-S test for in-degree
ks_stat_in, p_value_in = kstest(in_degree_val, 'poisson', args=(lambda_in,))
print(f'In-Degree Poisson K-S Test: KS Statistic={ks_stat_in}, p-value={p_value_in}')
if p_value_in > 0.05:
  print("Interpretation: The in-degree distribution fits well with a Poisson distribution.")


else:
  print("Interpretation: The in-degree distribution does not fit well with a Poisson distribution.")

# Perform K-S test for out-degree
ks_stat_out, p_value_out = kstest(out_degree_val, 'poisson', args=(lambda_out,))
print(f'Out-Degree Poisson K-S Test: KS Statistic={ks_stat_out},p-value={p_value_out}')
if p_value_out > 0.05:
  print("Interpretation: The out-degree distribution fits well with a Poisson distribution.")

else:
  print("Interpretation: The out-degree distribution does not fit well with a Poisson distribution.'
```

```
In-Degree Poisson K-S Test: KS Statistic=0.65261684610368, p-value=1.1169232223618135e-99
Interpretation: The in-degree distribution does not fit well with a Poisson distribution.
Out-Degree Poisson K-S Test: KS Statistic=0.2996756696330917,p-value=1.8867485243783547e-19
Interpretation: The out-degree distribution does not fit well with a Poisson distribution.
```

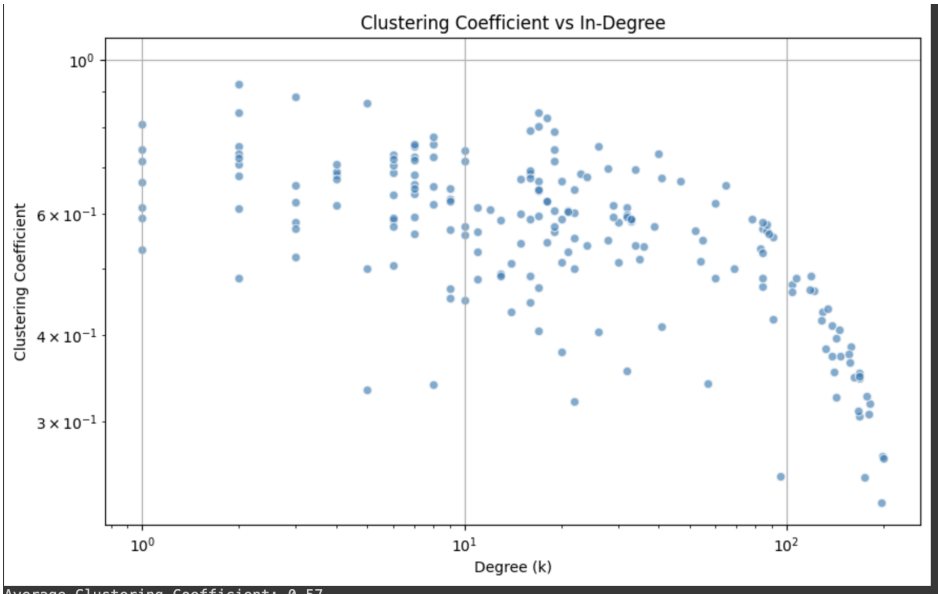
```python
import powerlaw
# Fit and analyze the in-degree distribution
fit_in = powerlaw.Fit(in_degree_val)
R, p = fit_in.distribution_compare('power_law', 'exponential')
print(f"In-Degree Power-Law Fit: Alpha={fit_in.alpha}, Sigma={fit_in.sigma}")
print(f"Likelihood Ratio: {R}, p-value={p}")
if p > 0.05:
  print("Interpretation: The in-degree distribution fits a Power-Law distribution better or as well as an exponential d


else:
  print("Interpretation: There is not enough evidence to suggest the in-degree distribution fits a Power-Law distributi

# Fit and analyze the out-degree distribution
fit_out = powerlaw.Fit(out_degree_val)
R, p = fit_out.distribution_compare('power_law', 'exponential')
print(f"Out-Degree Power-Law Fit: Alpha={fit_out.alpha}, Sigma={fit_out.sigma}")
print(f"Likelihood Ratio: {R}, p-value={p}")
if p > 0.05:
  print("Interpretation: The out-degree distribution fits a Power-Law distribution better or as well as an exponential d

else:
  print("Interpretation: There is not enough evidence to suggest the out-degree distribution fits a Power-Law distribut
```

```
Calculating best minimal value for power law fit
In-Degree Power-Law Fit: Alpha=1.770715213430229, Sigma=0.06838991615433433
Likelihood Ratio: -9.25114353964462, p-value=0.19522743252755703
Interpretation: The in-degree distribution fits a Power-Law distribution better or as well as an exponential distributio
Calculating best minimal value for power law fit
Out-Degree Power-Law Fit: Alpha=5.130155803498095, Sigma=0.5045788396183316
Likelihood Ratio: -0.2800172819448874, p-value=0.8279461822231922
Interpretation: The out-degree distribution fits a Power-Law distribution better or as well as an exponential distributi
Values less than or equal to 0 in data. Throwing out 0 or negative values
Values less than or equal to 0 in data. Throwing out 0 or negative values
```

```python
from scipy.stats import expon, kstest
import numpy as np
# Fit parameters for the exponential distribution
# The scale parameter (beta) is the inverse of the mean of the data
# Scale parameter estimation (mean of the data)
scale_in = np.mean(in_degree_val)
scale_out = np.mean(out_degree_val)
# Since exponential distribution is scale-dependent, we directly use the scale parameter.



# No additional data normalization (like Z-score) is necessary here, as we're fitting the scale parameter direc

# Perform K-S test for in-degree against an exponential distribution
ks_stat_in, p_value_in = kstest(in_degree_val, 'expon', args=(0, scale_in))
print(f'In-Degree Exponential K-S Test: KS Statistic={ks_stat_in}, p-value={p_value_in}')
if p_value_in > 0.05:
  print("Interpretation: The in-degree distribution fits well with an exponential distribution.")

else:
  print("Interpretation: The in-degree distribution does not fit well with an exponential distribution.")

# Perform K-S test for out-degree against an exponential distribution
ks_stat_out, p_value_out = kstest(out_degree_val, 'expon', args=(0, scale_out))
print(f'Out-Degree Exponential K-S Test: KS Statistic={ks_stat_out}, p-value={p_value_out}')
if p_value_out > 0.05:
  print("Interpretation: The out-degree distribution fits well with an exponential distribution.")

else:
  print("Interpretation: The out-degree distribution does not fit well with an exponential distribution.")
```

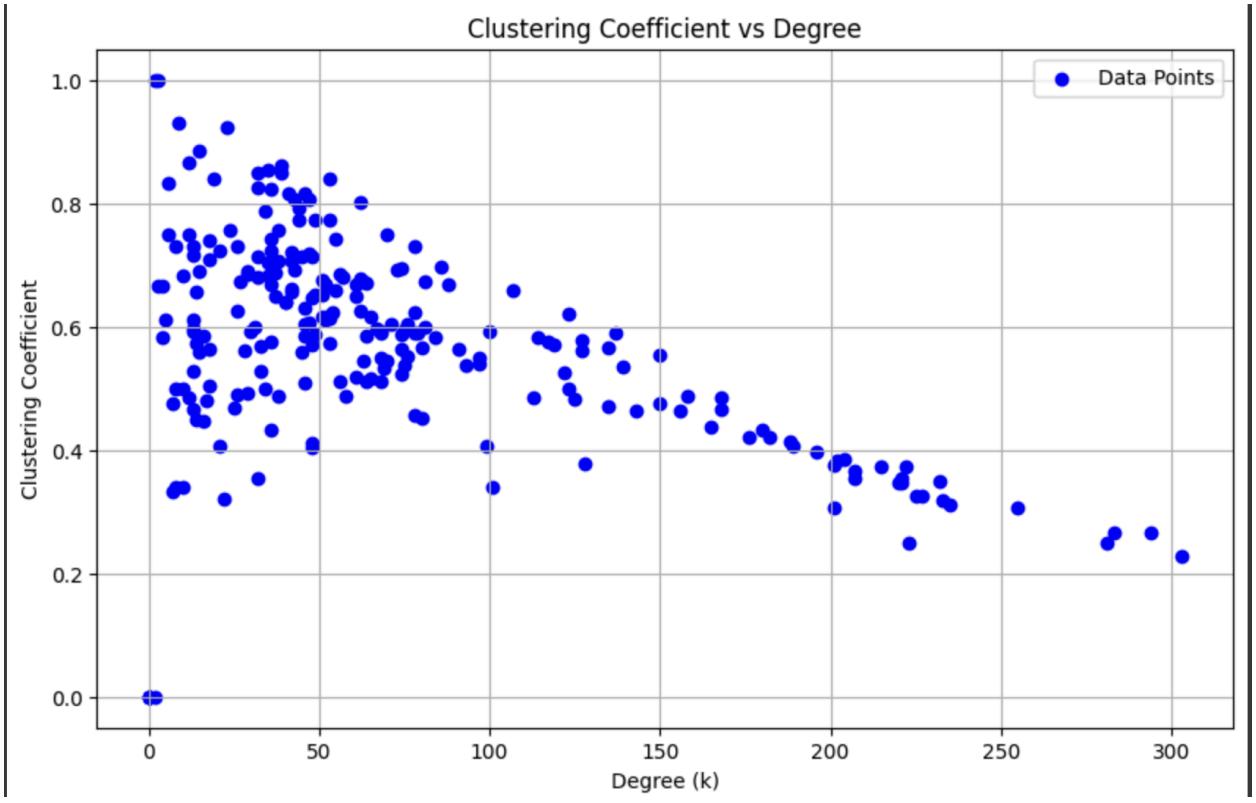
```
In-Degree Exponential K-S Test: KS Statistic=0.20926480070787887, p-value=1.2742931998508181e-09
Interpretation: The in-degree distribution does not fit well with an exponential distribution.
Out-Degree Exponential K-S Test: KS Statistic=0.19255399437475962, p-value=3.332816498779658e-08
Interpretation: The out-degree distribution does not fit well with an exponential distribution.
```

```python
import matplotlib.pyplot as plt
import networkx as nx

# Calculate the clustering coefficient for each node
clustering_coeffs = nx.clustering(G)

# Get the degree of each node
in_degrees = dict(G.in_degree())

# Prepare data for plotting
degree_values = list(in_degrees.values())
clustering_values = list(clustering_coeffs.values())

# Create a scatter plot of clustering coefficient vs degree
plt.figure(figsize=(10, 6))
plt.scatter(degree_values, clustering_values, alpha=0.6, edgecolors='w')
plt.title('Clustering Coefficient vs In-Degree')
plt.xlabel('Degree (k)')
plt.ylabel('Clustering Coefficient')
plt.xscale('log')  # Use logarithmic scale if the range of degrees is large
plt.yscale('log')  # Use logarithmic scale to see patterns more clearly if necessary
plt.grid(True)
plt.show()

# Interpretation
if not clustering_values:  # Check if list is empty
    print("No data to plot.")
else:
    avg_clustering = sum(clustering_values) / len(clustering_values)
    print(f"Average Clustering Coefficient: {avg_clustering:.2f}")
    print("Interpretation: Typically, a high clustering coefficient at lower degrees suggest
```



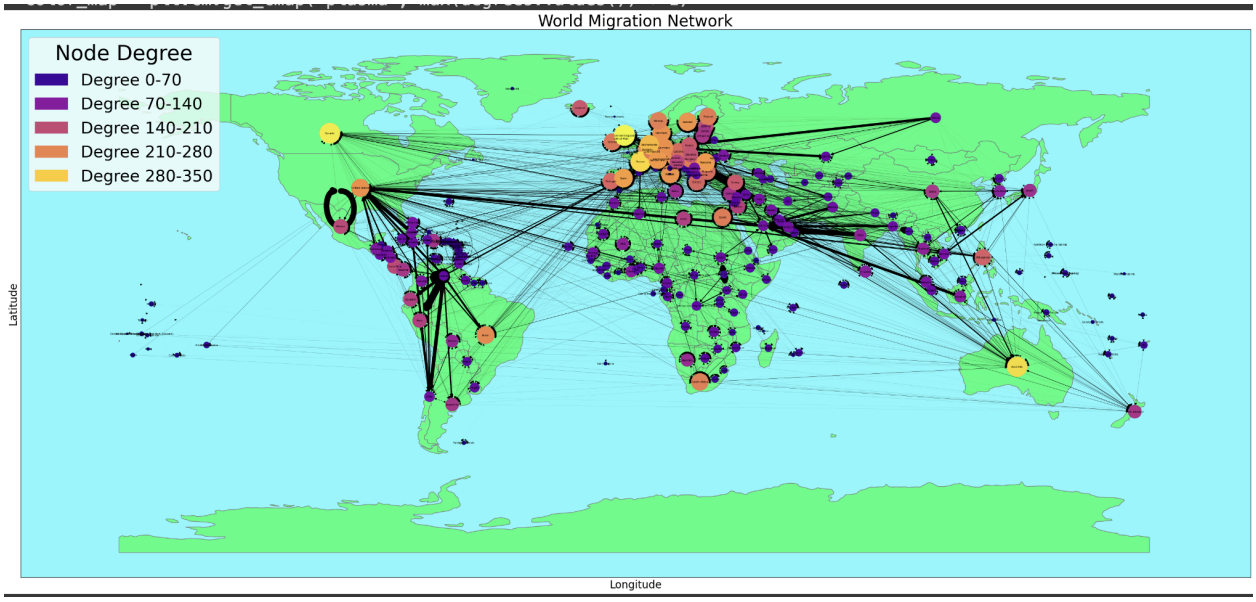

```
Average Clustering Coefficient: 0.57
Interpretation: Typically, a high clustering coefficient at lower degrees suggests that low-degree nodes form tight-knit
```

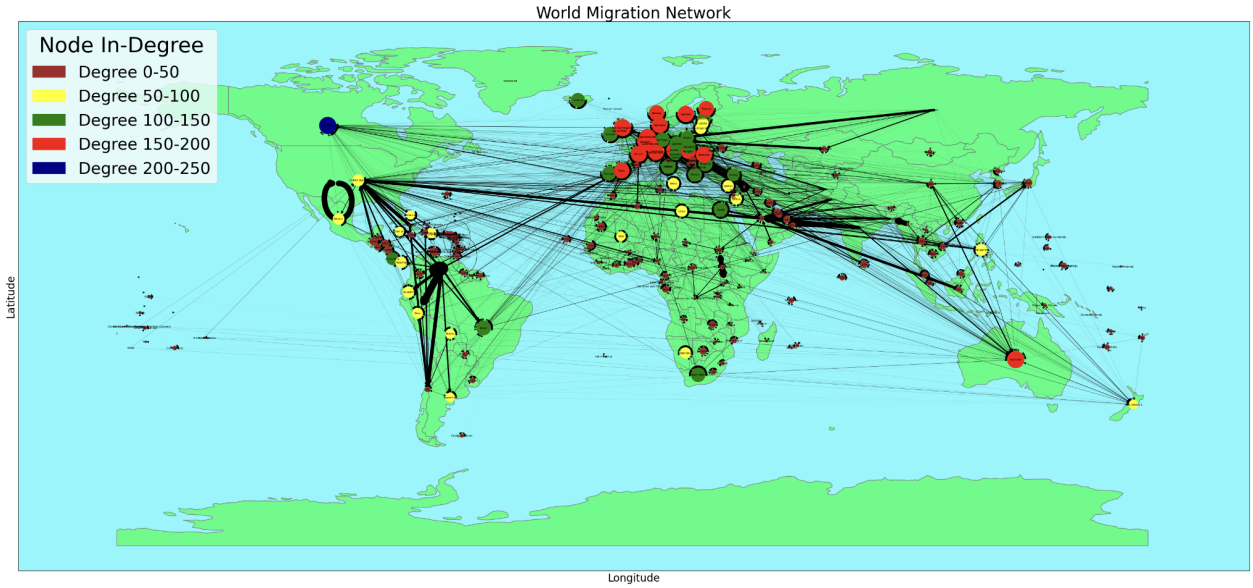
```python
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt

# Calculate clustering coefficient for each node
clustering_coefficient = nx.clustering(G)
# Calculate degree of each node
degree = dict(G.degree())
# Preparing data for plotting
k_vals = list(degree.values())
cc_vals = list(clustering_coefficient.values())
# Sorting data points by degree
sorted_indices = np.argsort(k_vals)
sorted_k_vals = np.array(k_vals)[sorted_indices]
sorted_cc_vals = np.array(cc_vals)[sorted_indices]
# Plotting
plt.figure(figsize=(10, 6))
plt.scatter(k_vals, cc_vals, color='blue', label='Data Points')

plt.title('Clustering Coefficient vs Degree')
plt.xlabel('Degree (k)')
plt.ylabel('Clustering Coefficient')
# plt.xscale('log')
# plt.yscale('log')
plt.grid(True)
plt.legend()
plt.show()
```



```python
import networkx as nx

# Calculate largest connected component
largest_cc = max(nx.strongly_connected_components(G), key=len)
G_largest_cc = G.subgraph(largest_cc)
# Calculate average clustering coefficient and average shortest path length
avg_clustering_coefficient = nx.average_clustering(G_largest_cc)
avg_shortest_path_length = nx.average_shortest_path_length(G_largest_cc)
# Generate random graph with the same number of nodes and edge probability as the largest connected component
# random_graph = nx.gnm_random_graph(len(G_largest_cc.nodes()),len(G_largest_cc.edges()))
random_graph = nx.gnm_random_graph(G.number_of_nodes(), G.number_of_edges(), directed=True)
# Calculate average clustering coefficient and average shortest path length for random graph
avg_clustering_coefficient_random = nx.average_clustering(random_graph)
avg_shortest_path_length_random = nx.average_shortest_path_length(random_graph)
# Print the results
print("Average Clustering Coefficient:", avg_clustering_coefficient)
print("Average Shortest Path Length:", avg_shortest_path_length)
print("Average Clustering Coefficient (Random Graph):", avg_clustering_coefficient_random)
print("Average Shortest Path Length (Random Graph):", avg_shortest_path_length_random)
# Determine if the network exhibits small-world or ultra-small-world characteristics
if avg_clustering_coefficient > avg_clustering_coefficient_random and avg_shortest_path_length < avg_shortest_path_lengt
  print("The network exhibits small-world characteristics.")

elif avg_clustering_coefficient > avg_clustering_coefficient_random and avg_shortest_path_length < avg_shortest_path_ler
  print("The network exhibits ultra-small-world characteristics.")

else:
  print("The network does not exhibit small-world or ultra-small-world characteristics.")
```

```
Average Clustering Coefficient: 0.6080779975470422
Average Shortest Path Length: 2.028632340241796
Average Clustering Coefficient (Random Graph): 0.15354564678405672
Average Shortest Path Length (Random Graph): 1.8496436549303266
The network does not exhibit small-world or ultra-small-world characteristics.
```





Top 20 Countries by Betweenness Centrality:

	Country	Betweenness Centrality	Degree Centrality	\
215	United Kingdom	0.077588	1.278481	
70	France	0.077163	1.185654	
11	Australia	0.061517	1.194093	
36	Canada	0.050700	1.240506	
216	United States	0.048840	0.940928	
60	Egypt	0.033534	0.852321	
190	South Africa	0.025422	0.873418	
143	Netherlands	0.021856	1.075949	
100	Italy	0.020574	0.949367	
162	Philippines	0.018692	0.742616	
76	Germany	0.016683	0.936709	
28	Brazil	0.016198	0.907173	
193	Spain	0.016161	0.991561	
123	Mali	0.014602	0.426160	
19	Belgium	0.014530	0.928270	
42	China	0.014319	0.540084	
33	Burundi	0.013373	0.168776	
208	Turkey	0.012000	0.708861	
114	Libya	0.011646	0.569620	
168	Romania	0.010783	0.983122	

One unexpected inference that can be drawn from the results is the relatively high centrality scores of countries like Egypt, South Africa, and the Philippines in certain centrality measures, despite not being traditionally considered as major destinations or sources of international migration compared to countries like the United Kingdom, France, or Australia.

Here are a few unexpected inferences that could be drawn:

1. **Egypt's Betweenness Centrality**: Egypt ranks relatively high in betweenness centrality, indicating that it plays a significant role in connecting various migration routes. This may be unexpected considering Egypt's location at the crossroads of Africa, Asia, and Europe, which could result in it being a transit point for migrants from different regions.
2. **South Africa's Closeness Centrality**: South Africa's relatively high closeness centrality suggests that it is well-connected to other countries in the migration network. This may be unexpected given that South Africa is often viewed as a destination rather than a transit point for migrants, highlighting its importance as a regional hub for migration in Africa.
3. **The Philippines' Degree Centrality**: The Philippines ranks relatively high in degree centrality, indicating a large number of migration connections with other countries. This may be unexpected considering the Philippines is often seen as a major source of migrant workers rather than a destination for international migrants.

These unexpected inferences highlight the complexity and diversity of migration patterns and the importance of considering a wide range of factors when analyzing migration networks. They also

```
Top 20 Countries by Degree Centrality:
           Country  Betweenness Centrality  Degree Centrality  \
215  United Kingdom                0.077588           1.278481
36           Canada                0.050700           1.240506
11        Australia                0.061517           1.194093
70           France                0.077163           1.185654
143     Netherlands                0.021856           1.075949
193           Spain                0.016161           0.991561
168         Romania                0.010783           0.983122
198     Switzerland                0.008018           0.978903
197          Sweden                0.005132           0.957806
100           Italy                0.020574           0.949367
216   United States                0.048840           0.940928
76          Germany                0.016683           0.936709
55          Denmark                0.005012           0.932489
12          Austria                0.006240           0.932489
19          Belgium                0.014530           0.928270
28           Brazil                0.016198           0.907173
153          Norway                0.003408           0.873418
190    South Africa                0.025422           0.873418
69          Finland                0.002309           0.860759
60            Egypt                0.033534           0.852321
```

Top 20 Countries by Closeness Centrality:

	Country	Betweenness Centrality	Degree Centrality
36	Canada	0.050700	1.240506
11	Australia	0.061517	1.194093
215	United Kingdom	0.077588	1.278481
168	Romania	0.010783	0.983122
143	Netherlands	0.021856	1.075949
197	Sweden	0.005132	0.957806
70	France	0.077163	1.185654
55	Denmark	0.005012	0.932489
198	Switzerland	0.008018	0.978903
117	Luxembourg	0.002063	0.848101
12	Austria	0.006240	0.932489
193	Spain	0.016161	0.991561
19	Belgium	0.014530	0.928270
69	Finland	0.002309	0.860759
153	Norway	0.003408	0.873418
91	Hungary	0.003544	0.848101
28	Brazil	0.016198	0.907173
53	Czechia	0.002034	0.797468
100	Italy	0.020574	0.949367
190	South Africa	0.025422	0.873418

	Closeness Centrality	Eigenvector Centrality
36	0.856804	0.166695
11	0.850162	0.167496
215	0.846879	0.154145
168	0.797606	0.162251
143	0.791848	0.165290
197	0.786171	0.162289
70	0.777808	0.134815
55	0.761603	0.160843
198	0.758968	0.156921
117	0.758968	0.136084
12	0.758968	0.159388
193	0.753752	0.144410
19	0.743531	0.149203
69	0.736046	0.157037
153	0.731139	0.143087
91	0.728710	0.153123
28	0.705279	0.154612
53	0.705279	0.153909
100	0.698541	0.126525
190	0.696323	0.140302

Top 20 Countries by Eigenvector Centrality:

	Country	Betweenness Centrality	Degree Centrality
11	Australia	0.061517	1.194093
36	Canada	0.050700	1.240506
143	Netherlands	0.021856	1.075949
197	Sweden	0.005132	0.957806
168	Romania	0.010783	0.983122
55	Denmark	0.005012	0.932489
12	Austria	0.006240	0.932489
69	Finland	0.002309	0.860759
198	Switzerland	0.008018	0.978903
28	Brazil	0.016198	0.907173
215	United Kingdom	0.077588	1.278481
53	Czechia	0.002034	0.797468
91	Hungary	0.003544	0.848101
76	Germany	0.016683	0.936709
19	Belgium	0.014530	0.928270
92	Iceland	0.001108	0.696203
97	Ireland	0.003443	0.827004
31	Bulgaria	0.004557	0.793249
164	Portugal	0.007751	0.767932
193	Spain	0.016161	0.991561

	Closeness Centrality	Eigenvector Centrality
11	0.850162	0.167496
36	0.856804	0.166695
143	0.791848	0.165290
197	0.786171	0.162289
168	0.797606	0.162251
55	0.761603	0.160843
12	0.758968	0.159388
69	0.736046	0.157037
198	0.758968	0.156921
28	0.705279	0.154612
215	0.846879	0.154145
53	0.705279	0.153909
91	0.728710	0.153123
76	0.687592	0.149815
19	0.743531	0.149203
92	0.679077	0.148762
97	0.696323	0.146396
31	0.685443	0.146322
164	0.666692	0.145443
193	0.753752	0.144410