Network Analysis of Airport Routes

Objective

In this exercise, you will analyze a network of airport routes to understand its structure and connectivity. You'll work with `networkx` to build, visualize, and analyze the network properties of real-world flight routes between airports.

Data Description

The data consists of two files:

1. **`airports.csv`**: Contains information about each airport, including a unique identifier, name, location (latitude, longitude), and other metadata.
2. **`routes.csv`**: Lists the flights between airports. Each row represents a directed flight route from one airport (source) to another (destination).

Each airport will be represented as a **node** in the network, and each flight route will be represented as a **directed edge**.

Step 0: Data Description and Importing Data

1. **Define Column Names**: Start by defining the column names for each dataset.

```
# For airports.csv
airport_columns = ["Airport ID", "Name", "City", "Country", "IATA",
"ICAO", "Latitude", "Longitude", "Altitude", "Timezone", "DST", "TZ
Database Timezone", "Type", "Source"]

# For routes.csv
route_columns = ["Airline", "Airline ID", "Source Airport", "Source
Airport ID", "Destination Airport", "Destination Airport ID",
"Codeshare", "Stops", "Equipment"]
```

2. **Import the Data**: Use `pandas` to load each dataset with the defined column names.
   - Be sure to specify the correct encoding (`utf-8`).
   - To ensure consistency, convert the IDs to strings immediately after loading the data.

```
import pandas as pd
```

```
# Load datasets without headers
airports_df = pd.read_csv("airports.csv", names=airport_columns,
encoding="utf-8")
routes_df = pd.read_csv("routes.csv", names=route_columns,
encoding="utf-8")

# Convert ID columns to strings
airports_df['Airport ID'] = airports_df['Airport ID'].astype(str)
routes_df['Source airport ID'] = routes_df['Source airport
ID'].astype(str)
routes_df['Destination airport ID'] = routes_df['Destination airport
ID'].astype(str)
```

Step 1: Create the Network

1. **Filter the Data**: Keep only the routes where both the source and destination airports exist in `airports.csv`.
   - **Hint**: Use `pandas.isin()` to filter the data based on IDs.
2. **Initialize the Graph**: Create a directed graph in `networkx` using `nx.DiGraph()`.
3. **Add Nodes and Edges**:
   - For each airport, add a node with attributes such as `name`, `latitude`, and `longitude`.
   - For each route, add a directed edge from the source airport to the destination airport.
4. **Functions**:
   - `G.add_node()` to add nodes with attributes.
   - `G.add_edge()` to add directed edges.

Step 2: Filter the Network for Visualization

1. **Remove Isolated Nodes**: Identify nodes with no incoming or outgoing connections (degree zero) and remove them from the graph.
   - **Hint**: Use `G.degree()` to check the degree of each node and remove isolated nodes.
2. **Create a Subgraph**:
   - Filter down to the top 200 airports by degree to make the visualization easier to interpret.
   - Create a subgraph that includes only these top nodes.
3. **Functions**:
   - `sorted()` to rank nodes by degree.
   - `G.subgraph()` to create a new graph with only the specified nodes.

4. **Visualize the Network**:
    - **Geographical Plot**: Use the latitude and longitude of each airport to position the nodes. Color the top 10 most connected airports differently and add a legend for clarity.
    - **Force-Directed Layout**: Use `nx.spring_layout()` to generate a layout that spaces nodes based on their connections. Adjust the node size by degree and highlight the top 10 most connected nodes.
5. **Functions**:
    - `nx.draw_networkx_nodes()` and `nx.draw_networkx_edges()` to draw the network.
    - `nx.spring_layout()` to create a force-directed layout.

Step 3: Analyze Network Properties

A) Connectivity Analysis

1. **Connected Components**:
    - Calculate the number and relative size of both weakly and strongly connected components to understand network connectivity.
2. **Functions**:
    - `nx.weakly_connected_components()` to get weakly connected components.
    - `nx.strongly_connected_components()` to get strongly connected components.

B) Path Lengths and Reachability

1. **Path Length Distribution**:
    - Calculate the shortest path lengths in the largest strongly connected component and plot their distribution.
2. **Average Path Length and Diameter**:
    - Calculate the average shortest path length and the diameter of the largest strongly connected component.
3. **Functions**:
    - `nx.shortest_path_length()` to calculate shortest paths.
    - `nx.average_shortest_path_length()` to compute the average path length.
    - `nx.diameter()` to determine the diameter.

C) Degree Analysis

1. **Degree Distributions**:
    - Compute the in-degree and out-degree distributions and plot them on a log-log scale.
2. **Functions**:

- `G.in_degree()` and `G.out_degree()` to get node in-degrees and out-degrees.

3. **Top Hubs**:
   - Identify the top 5 airports by in-degree and out-degree, displaying the airport names and degree counts.
4. **Functions**:
   - `sorted()` with `G.in_degree()` or `G.out_degree()` to find top hubs by degree.

D) Clustering and Assortativity

1. **Global Clustering Coefficient**:
   - Compute the global clustering coefficient to assess the likelihood that an airport's neighbors are also connected.
2. **Function**:
   - `nx.transitivity()` to calculate the global clustering coefficient.
3. **Assortativity Analysis**:
   - **In-Degree Assortativity**: Calculate the average in-degree of neighbors for each node to see if highly connected nodes tend to link to other highly connected nodes.
   - **Geographical Assortativity**: Compute and plot the correlation between an airport's latitude (or longitude) and the average latitude (or longitude) of its neighbors.
4. **Functions**:
   - `nx.average_neighbor_degree()` to calculate the average degree of neighbors.
   - Use latitude and longitude values from the node attributes for geographical assortativity.