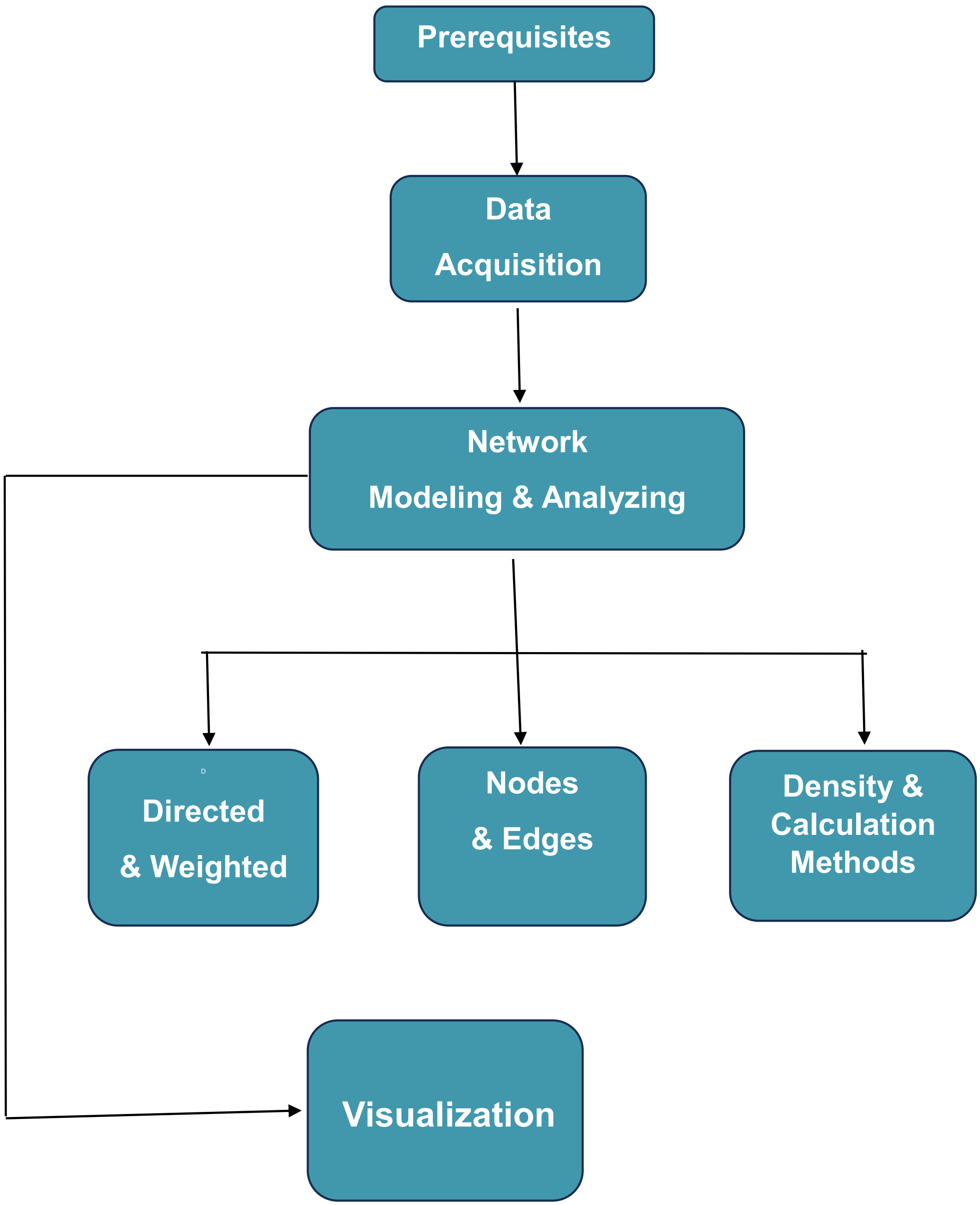
Lab 1

Hsu-Chieh (Jasmine) Ma

Key Questions

1. What are the main steps you took?
2. Is your network directed or undirected?
3. Is your network weighted or unweighted? If so, what is the weight of the nodes and edges in your network?
4. How many nodes and edges do you have in the network model?
5. What is the network density? How do you calculate network density
6. Visualize the network in ArcMap, ArcGIS Pro, or Python environments.

Workflow


Prerequisites
Data Acquisition
Network Modeling & Analyzing
Directed & Weighted
Nodes & Edges
Density & Calculation Methods
Visualization

Prerequisites

Using VS Code rather than Jupyter Notebook this time because the former better suited for handling large datasets and complex operations, like network analysis and visualization.

1. **Environment settings**
   Install necessary packages via pip, ensuring all packages are successfully installed at Command Prompt.

```
Microsoft Windows [Version 10.0.26100.1742]
(c) Microsoft Corporation. All rights reserved.

C:\Users\jasmi>pip show osmnx networkx matplotlib
Name: osmnx
Version: 1.9.4
Summary: Download, model, analyze, and visualize street networks and other geospatial features from OpenStreetMap
Home-page:
Author:
Author-email: Geoff Boeing <boeing@usc.edu>
License: MIT License
Location: C:\Users\jasmi\AppData\Local\Programs\Python\Python312\Lib\site-packages
Requires: geopandas, networkx, numpy, pandas, requests, shapely
Required-by:
---
Name: networkx
Version: 3.3
Summary: Python package for creating and manipulating graphs and networks
Home-page: https://networkx.org/
Author:
Author-email: Aric Hagberg <hagberg@lanl.gov>
License:
Location: C:\Users\jasmi\AppData\Local\Programs\Python\Python312\Lib\site-packages
Requires:
Required-by: osmnx
---
Name: matplotlib
Version: 3.9.2
Summary: Python plotting package
Home-page: https://matplotlib.org
Author: John D. Hunter, Michael Droettboom
Author-email: Unknown <matplotlib-users@python.org>
License: License agreement for matplotlib versions 1.3.0 and later
=========================================================
```

2. **Import required packages**
   Open VS Code and import necessary packages, naming them with simple alias.

```python
import osmnx as ox # data fetching
import networkx as nx  # graph generation & analysis
import matplotlib.pyplot as plt #visualization
```

Data Acquisition

Download network data: after defining the location for the network model ("DeKalb County, Georgia, USA") as “location_name”, using OSMnx to fetch the road network.

```python
# Define the location
location_name = "DeKalb County, Georgia, USA"
# Download the road network
G = ox.graph_from_place(location_name, network_type='drive')
```

Network Modeling & Analyzing

This part could be break down as follows:

1. **Checking if the network is directed, and weighted degree.**
2. **Observing the nodes and edges of the network.**
3. **Calculating the density.**

1. For direction checking, using **.is_directed() function**, the terminal would return Boolean value of **“True”** if this network is directed.

```python

# Check the network is directed or not?
print("Directed or not?", nx.is_directed(G))

# Check if the network is weighted by printing edge lengths
count = 0
for u, v, key, data in G.edges(keys=True, data=True):
    print("Edge length (meters):", data['length'])
    count += 1
    if count >= 10:  # Stop after printing 10 edges
        break
```

```
>>>
>>> location_name = "DeKalb County, Georgia, USA"
>>> # Download the road network
>>>
>>> G = ox.graph_from_place(location_name, network_type='drive')
>>> print("Directed or not?", nx.is_directed(G))
Directed or not? True
```

Turning to check if the network is weighted or not, I ran the following for loop (line 13-19) to generate the first 10 rows of edges data as validation, without cluttering the console.

```
>>> count = 0
>>> for u, v, key, data in G.edges(keys=True, data=True):
...     print("Edge length (meters):", data['length'])
...     count += 1
...     if count >= 10:  # Stop after printing 10 edges
...         break
...
Edge length (meters): 156.531
Edge length (meters): 283.764
Edge length (meters): 211.739
Edge length (meters): 310.388
Edge length (meters): 156.531
Edge length (meters): 160.741
Edge length (meters): 53.489
Edge length (meters): 432.401
Edge length (meters): 426.994
Edge length (meters): 432.40099999999995
>>>
```

It could be stated this network is **weighted** for its distinct edge lengths.

After the testing step, removing the initial segment, storing all edge lengths into a **list** and printing out. The **weighted edge length is 39.426 meters**.

```
13  # Store edge lengths in a list
14  edge_lengths = []
15  for u, v, key, data in G.edges(keys=True, data=True):
16      edge_lengths.append(data['length'])
...
>>> print("Edge length (meters):", data.get('length', 0))
Edge length (meters): 39.426
```

Steps above could be applied to nodes calculation as well.

```python
# Calculate the weighted degree of nodes
node_weights = []
for node in G.nodes():
    degree = sum(data['length'] for u, v, key, data in G.edges(node, keys=True, data=True))
    node_weights.append(degree)  # Add the weighted degree to the list

# Calculate the total weighted degree for all nodes
total_weighted_nodes = sum(node_weights)
print(f"Weighted nodes: {total_weighted_nodes} meters")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
...
>>> print("Edge length (meters):", data.get('length', 0))
Edge length (meters): 39.426
>>> node_weights = []
>>> for node in G.nodes():
...     degree = sum(data['length'] for u, v, key, data in G.edges(node, keys=True, data=True))
...     node_weights.append(degree)  # Add the weighted degree to the list
... # Calculate the total weighted degree for all nodes
...
>>> total_weighted_nodes = sum(node_weights)
>>> print(f"Weighted nodes: {total_weighted_nodes} meters")
Weighted nodes: 9264779.038999973 meters
>>>
```

The **total weighted nodes** are **9,264,779 meters.**

2. **.number_of_nodes()** and **.number_of_edges()** here are served to presenting the number of the two main constituents of a network separately. Considering the large amount of data, I then printed out the number of nodes and edges by the command below.

```
# Checking the qty of nodes and edges
num_nodes = G.number_of_nodes()
num_edges = G.number_of_edges()
print(f"Num of nodes: {num_nodes}")
print(f"Num of edges: {num_edges}")
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
Edge length (meters): 426.994
Edge length (meters): 432.40099999999995
>>> edge_lengths = []
>>> for u, v, key, data in G.edges(keys=True, data=True):
...     edge_lengths.append(data['length'])
...
>>> num_nodes = G.number_of_nodes()
>>> num_edges = G.number_of_edges()
>>> print(f"Num of nodes: {num_nodes}")
Num of nodes: 24003
>>> print(f"Num of edges: {num_edges}")
Num of edges: 56460
>>> 
```

As the screenshot shows, this network had **24003 of nodes**, and **56460 of edges**.

3. Here to calculate the density of network, function **nx.density(G)** in NetworkX is used to calculate the density of a graph G.

   To simplify the lengthy value, implement the built-in **round** function to round the decimal places to 6, for density is too small and would be zero if typed the value under 6.

```python
# Calculate and round network density
density = nx.density(G)
print(f"Network density: {density}")
rounded_density = round(density, 6)
print(f"Network density (rounded): {rounded_density}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

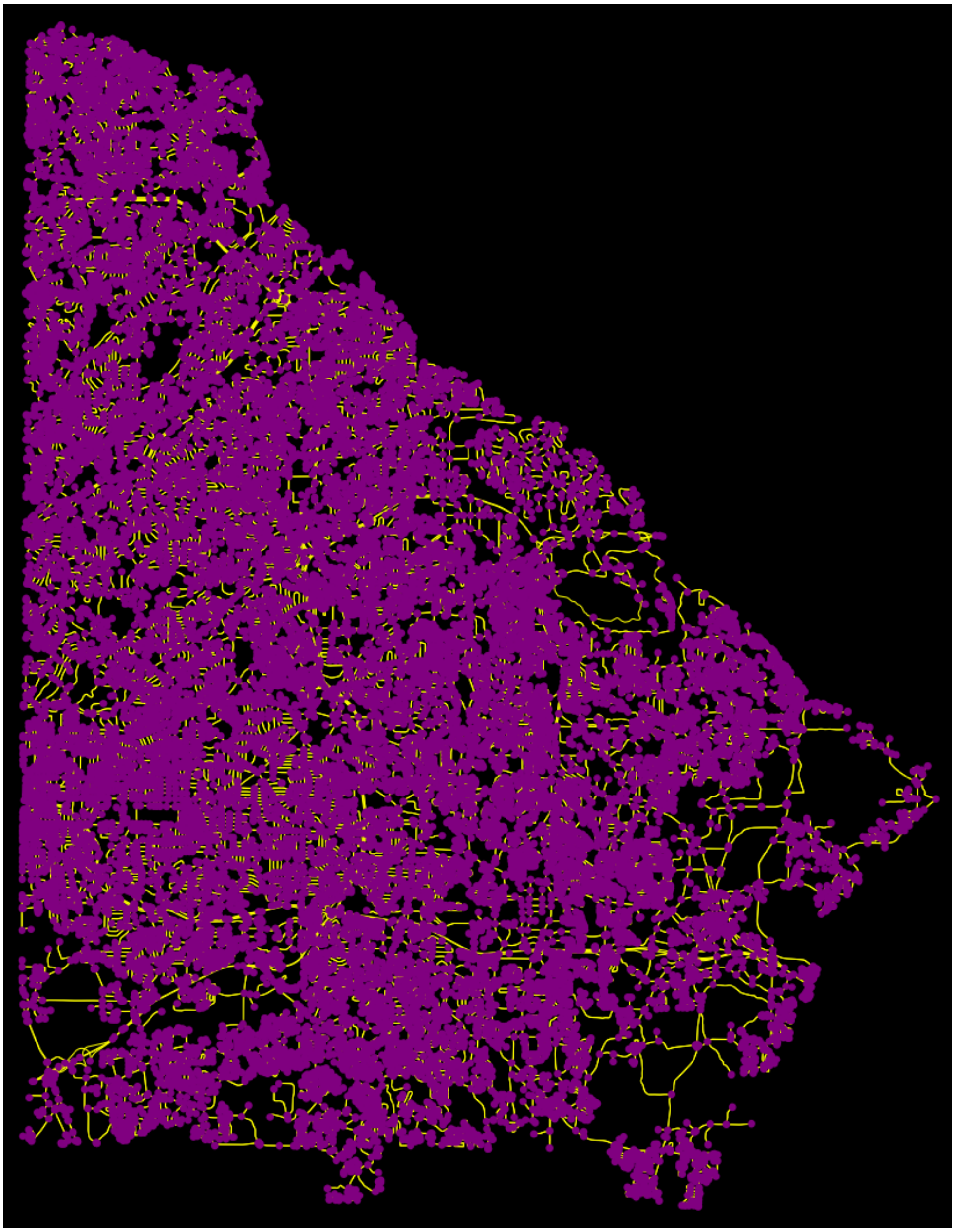
```
>>> num_nodes = G.number_of_nodes()
>>> num_edges = G.number_of_edges()
>>> print(f"Num of nodes: {num_nodes}")
Num of nodes: 24003
>>> print(f"Num of edges: {num_edges}")
Num of edges: 56460
>>> # Calculate and round network density
>>>
>>> density = nx.density(G)
>>> print(f"Network density: {density}")
Network density: 9.800041555925416e-05
>>> rounded_density = round(density, 6)
>>> print(f"Network density (rounded): {rounded_density}")
Network density (rounded): 9.8e-05
>>>
```

As result, the density of this network is $\mathbf{9.8 \times 10^{-5}}$

Visualization

Visualizing the graph requires following steps. To increase the contrast of the network, I adjusted the default black and white color palette to purple and yellow, for nodes and edges respectively.

```python
# Visualize the entire network
fig, ax = ox.plot_graph(
    G,
    node_size=10,
    node_color='purple',
    edge_color='yellow',
    bgcolor='black',
    edge_linewidth=0.5,    # Set edge width
    figsize=(10, 10)       # Adjust figure size
)
# Save the plot as an image
fig.savefig("dekalb_road_network.png", dpi=300)
```



Despite the graph provides a view of higher density, the network density calculation is based on **the ratio of actual edges to the maximum possible number of edges in a fully connected graph**. Compared to the fully connected one, a great number of nodes but relatively few edges would result in a lower value.

Recap: Key Questions

1. **What are the main steps you took?**

   Prerequisites, data acquisition, network modeling & analyzing, and graph visualization.

2. **Is your network directed or undirected?**

   Directed.

3. **Is your network weighted or unweighted? If so, what is the weight of the nodes and edges in your network?**

   Weighted; 9,264,779 and 39.426 meters separately.

4. **How many nodes and edges do you have in the network model?**

   24003 of nodes and 56460 of edges.

5. **What is the network density? How do you calculate network density?**

   Function nx.density(G) in NetworkX is used to calculate the density of a graph G), the output is 9.8×10^{-5}