Utilizing Python in GIS for Spatial Analysis of Travel Habits

Chenyi Weng

SSCI586 - GIS Programming and Customization

Professor Jennifer N. Swift

September 29th, 2024

1. Introduction

The objective of this project was to explore the fundamentals of programming with Python in GIS, specifically using ArcPy in ArcGIS Pro to manipulate spatial and tabular data. The workflow includes importing CSV files, converting them into spatial data, and analyzing travel habits based on latitude zones. By combining Python scripting with geoprocessing tools, this project aimed to understand author's travel preferences by plotting visited cities and performing spatial analysis.

2. Study Area

The study area for this project encompasses 20 cities worldwide that were visited. These cities span across different continents and various climate zones, allowing for an analysis of travel habits in relation to geographic locations. The map generated in ArcGIS Pro (Figure 1) illustrates the distribution of these cities across the globe.

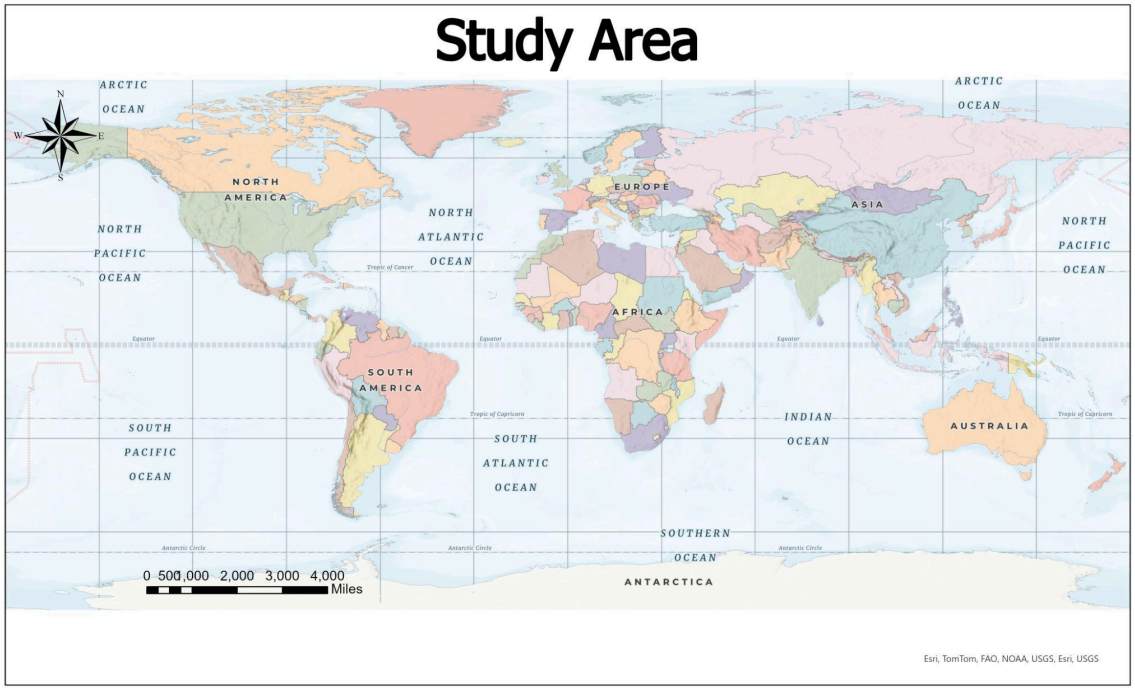


Figure1. Global Study Area Map of Visited Cities

3. Data and Data Processing

The dataset used for this project consists of a CSV file containing 20 cities' names along with their respective latitude (y) and longitude (x) coordinates. These coordinates were manually gathered using **Google Maps** and structured into three fields: city name, latitude, and longitude. The original projection of this data was in **WGS 84 (EPSG:4326)**, which is the standard geographic coordinate system used for most latitude and longitude data.

After verifying the CSV's readability and accuracy using **pandas** in Python, the data was imported into **ArcGIS Pro** using **ArcPy**. The CSV was then converted into a point feature class using the **XYTableToPoint** tool. The data was spatially referenced in **WGS 84** to maintain consistency with the original projection.

```
[1] import csv

    data = [
        ["Seol, South Korea", "37.547342589652274", "126.98511144648732"],
        ["Nagoya, Japan", "35.174510528584854", "136.9063837249868"],
        ["Osaka, Japan", "34.67164118765276", "135.4977066541952"],
        ["Tokyo, Japan", "35.68503670612925", "139.7671247135716"],
        ["Neuschwanstein Castle, Germany", "47.5577694601196", "10.749746752837519"],
        ["Luxembourg", "49.62169144041758", "6.124901815608386"],
        ["Jungfraujoch, Switzerland", "46.537461966197725", "7.963754"],
        ["Times Square, New York, USA", "40.75825098254559", "-73.98557478981621"],
        ["Harvard University, Boston, USA", "42.37597910680261","-71.11809237548384"],
        ["Niagara Falls, Canada", "43.082957218787534", "-79.07415217436512"],
        ["Taipei 101, Taipei, Taiwan", "25.033898210302947", "121.564506709616"],
        ["Spatial Sciences Institute, Los Angeles, USA", "34.01939171206854", "-118.2848439579041"],
        ["Petronas Towers, Kuala Lumpur, Malaysia", "3.157584957907343", "101.7115455019045"],
        ["Sabah, Malaysia", "5.437770891464025", "116.78697062549274"],
        ["Universal Studios, Singapore", "1.2542317063356991", "103.82382785086752"],
        ["Jinli, Chengdu, China", "30.660787244747738", "104.0563049062817"],
        ["Beijing, China", "39.906420928722554", "116.40216037899852"],
        ["Hong Kong", "22.326601365972223", "114.16959602277512"],
        ["Shanghai, China", "31.242716667012893", "121.44647926869122"],
        ["Hangzhou, Zhejiang, China", "30.2910838667552", "120.20264806438426"],
    ]

    with open('locations.csv', 'w', newline='', encoding='utf-8') as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(['location', 'y', 'x'])
        writer.writerows(data)

    print("travel_locations.csv")

    travel_locations.csv
```

Figure 2. CSV Data Verification in Python

4. Method

4.1 CSV Creation

Latitude and longitude data were manually gathered from Google Maps for the visited cities. These coordinates were saved into a CSV file, structured with fields for the city name, latitude (y), and longitude (x).

4.2 Python Script to Import Data

A Python script using ArcPy was written to import the CSV file into ArcGIS Pro and convert it into a point layer (Figure 3). This process involved using **XYTableToPoint** to create point features from the coordinates.

```
import arcpy

# Set the workspace
workspace = r'C:\Users\wengchen\Desktop\Project2'
arcpy.env.workspace = workspace
arcpy.env.overwriteOutput = True

# Path to the CSV file
csv_file = r'C:\Users\wengchen\Desktop\Project2\travel_locations.csv'

# Define the output point feature class
output_feature_class = r'C:\Users\wengchen\Desktop\Project2\travel_locations_points.shp'

# Set the spatial reference (WGS 84, EPSG 4326)
spatial_reference = arcpy.SpatialReference(4326)

# Use the XY Table To Point tool to convert the CSV to a point layer
# I use 'x' for longitude and 'y' for latitude based on my CSV structure
arcpy.management.XYTableToPoint(csv_file, output_feature_class, "x", "y", coordinate_system=spatial_reference)

print(f"CSV has been successfully converted to a point layer: {output_feature_class}")
```

```
CSV has been successfully converted to a point layer: C:\Users\wengchen\Desktop\Project2\travel_locations_points.shp
```

Figure 3. CSV Import and Point Feature Creation in ArcGIS Pro

4.3 Latitude Zone Classification

The cities were categorized into latitude zones—**Tropical, Temperate, and Polar**—based on their latitudes. A Python script was used to classify the cities, and the results were stored as an additional field in the attribute table (Figure 4 and Figure 5).

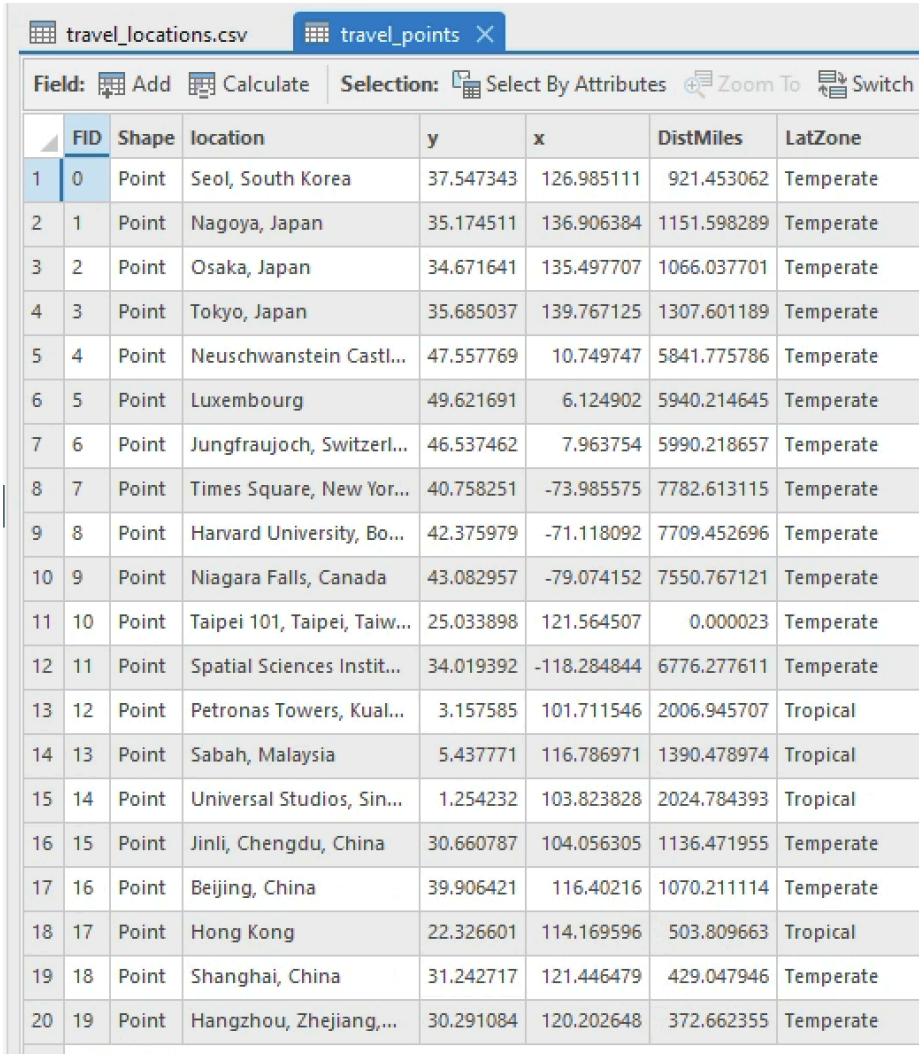
```python
# Add a new field to store latitude zone classification
arcpy.management.AddField(point_layer, "LatZone", "TEXT")

# Classify the latitude zones (Tropical, Temperate, Polar) and update the attribute table
with arcpy.da.UpdateCursor(point_layer, ['location', 'SHAPE@Y', 'LatZone']) as cursor:
    for row in cursor:
        city, lat, zone_field = row
        if lat >= -23.5 and lat <= 23.5:
            row[2] = 'Tropical'
        elif lat > 23.5 and lat <= 66.5:
            row[2] = 'Temperate'
        else:
            row[2] = 'Polar'
        cursor.updateRow(row)

print("Latitude zones have been classified and saved to the attribute table.")
```

```
Latitude zones have been classified and saved to the attribute table.
```

Figure 4. Latitude Zone Classification Code



	FID	Shape	location	y	x	DistMiles	LatZone
1	0	Point	Seol, South Korea	37.547343	126.985111	921.453062	Temperate
2	1	Point	Nagoya, Japan	35.174511	136.906384	1151.598289	Temperate
3	2	Point	Osaka, Japan	34.671641	135.497707	1066.037701	Temperate
4	3	Point	Tokyo, Japan	35.685037	139.767125	1307.601189	Temperate
5	4	Point	Neuschwanstein Castl...	47.557769	10.749747	5841.775786	Temperate
6	5	Point	Luxembourg	49.621691	6.124902	5940.214645	Temperate
7	6	Point	Jungfraujoch, Switzerl...	46.537462	7.963754	5990.218657	Temperate
8	7	Point	Times Square, New Yor...	40.758251	-73.985575	7782.613115	Temperate
9	8	Point	Harvard University, Bo...	42.375979	-71.118092	7709.452696	Temperate
10	9	Point	Niagara Falls, Canada	43.082957	-79.074152	7550.767121	Temperate
11	10	Point	Taipei 101, Taipei, Taiw...	25.033898	121.564507	0.000023	Temperate
12	11	Point	Spatial Sciences Instit...	34.019392	-118.284844	6776.277611	Temperate
13	12	Point	Petronas Towers, Kual...	3.157585	101.711546	2006.945707	Tropical
14	13	Point	Sabah, Malaysia	5.437771	116.786971	1390.478974	Tropical
15	14	Point	Universal Studios, Sin...	1.254232	103.823828	2024.784393	Tropical
16	15	Point	Jinli, Chengdu, China	30.660787	104.056305	1136.471955	Temperate
17	16	Point	Beijing, China	39.906421	116.40216	1070.211114	Temperate
18	17	Point	Hong Kong	22.326601	114.169596	503.809663	Tropical
19	18	Point	Shanghai, China	31.242717	121.446479	429.047946	Temperate
20	19	Point	Hangzhou, Zhejiang,...	30.291084	120.202648	372.662355	Temperate

Figure 5. The attribute table for "travel_points"

4.4 Distance Analysis

The distances from each city to Taipei (the home city) were calculated using the Haversine formula; also, this Python function calculated the distances in both kilometers and miles. However, the results were added as new fields in the attribute table (Figure 6).

```python
import math

# Taipei's coordinates
taipei_lat = 25.033898
taipei_lon = 121.564507

# Point Layer generated from your CSV data
point_layer = r'C:\Users\wengchen\Desktop\Project2\travel_locations_points.shp'

# Define the Haversine function to calculate distance in kilometers
def haversine(lat1, lon1, lat2, lon2):
    R = 6371.0  # Radius of Earth in kilometers
    dlat = math.radians(lat2 - lat1)
    dlon = math.radians(lon2 - lon1)
    a = math.sin(dlat / 2)**2 + math.cos(math.radians(lat1)) * math.cos(math.radians(lat2)) * math.sin(dlon / 2)**2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
    distance = R * c
    return distance

# Calculate distance from Taipei to each city in the point Layer
with arcpy.da.SearchCursor(point_layer, ['location', 'SHAPE@XY']) as cursor:
    for row in cursor:
        city, (lon, lat) = row
        distance_km = haversine(taipei_lat, taipei_lon, lat, lon)

        # Convert km to miles
        distance_miles = distance_km * 0.621371  # 1 km = 0.621371 miles

        # Print both kms and miles
        print(f"Distance from Taipei to {city}: {distance_km:.2f} km ({distance_miles:.2f} miles)")
```

```
Distance from Taipei to Seol, South Korea: 1482.94 km (921.45 miles)
Distance from Taipei to Nagoya, Japan: 1853.32 km (1151.60 miles)
Distance from Taipei to Osaka, Japan: 1715.62 km (1066.04 miles)
Distance from Taipei to Tokyo, Japan: 2104.38 km (1307.60 miles)
Distance from Taipei to Neuschwanstein Castle, Germany: 9401.43 km (5841.78 miles)
Distance from Taipei to Luxembourg: 9559.85 km (5940.21 miles)
Distance from Taipei to Jungfraujoch, Switzerland: 9640.33 km (5990.22 miles)
Distance from Taipei to Times Square, New York, USA: 12524.91 km (7782.61 miles)
Distance from Taipei to Harvard University, Boston, USA: 12407.17 km (7709.45 miles)
Distance from Taipei to Niagara Falls, Canada: 12151.79 km (7550.77 miles)
Distance from Taipei to Taipei 101, Taipei, Taiwan: 0.00 km (0.00 miles)
Distance from Taipei to Spatial Sciences Institute, Los Angeles, USA: 10905.37 km (6776.28 miles)
Distance from Taipei to Petronas Towers, Kuala Lumpur, Malaysia: 3229.87 km (2006.95 miles)
Distance from Taipei to Sabah, Malaysia: 2237.76 km (1390.48 miles)
Distance from Taipei to Universal Studios, Singapore: 3258.58 km (2024.78 miles)
Distance from Taipei to Jinli, Chengdu, China: 1828.97 km (1136.47 miles)
Distance from Taipei to Beijing, China: 1722.34 km (1070.21 miles)
Distance from Taipei to Hong Kong: 810.80 km (503.81 miles)
Distance from Taipei to Shanghai, China: 690.49 km (429.05 miles)
Distance from Taipei to Hangzhou, Zhejiang, China: 599.74 km (372.66 miles)
```

```python
# Add the DisMiles field to attribute table
arcpy.management.AddField(point_layer, "DistMiles", "DOUBLE")

# Update the attribute table with the distance values in miles
with arcpy.da.UpdateCursor(point_layer, ['location', 'SHAPE@XY', 'DistMiles']) as cursor:
    for row in cursor:
        city, (lon, lat), distance_field = row
        distance_km = haversine(taipei_lat, taipei_lon, lat, lon)
        distance_miles = distance_km * 0.621371
        row[2] = distance_miles  # Update DistMiles field
        cursor.updateRow(row)

print("Distance in miles has been saved to the attribute table.")
```

```
Distance in miles has been saved to the attribute table.
```

Figure 6. Distance Calculation Using Haversine Formula

5. Results Visualization

After completing the workflow, the spatial data was analyzed to understand the travel habits. The final map (Figure 7 and Figure 8) indicates that most of the travel occurred within the Temperate Zone, aligning with a preference for visiting cities with moderate climates. Furthermore, the Tropical Zone was the second most visited, while no cities in the Polar Zone were visited.

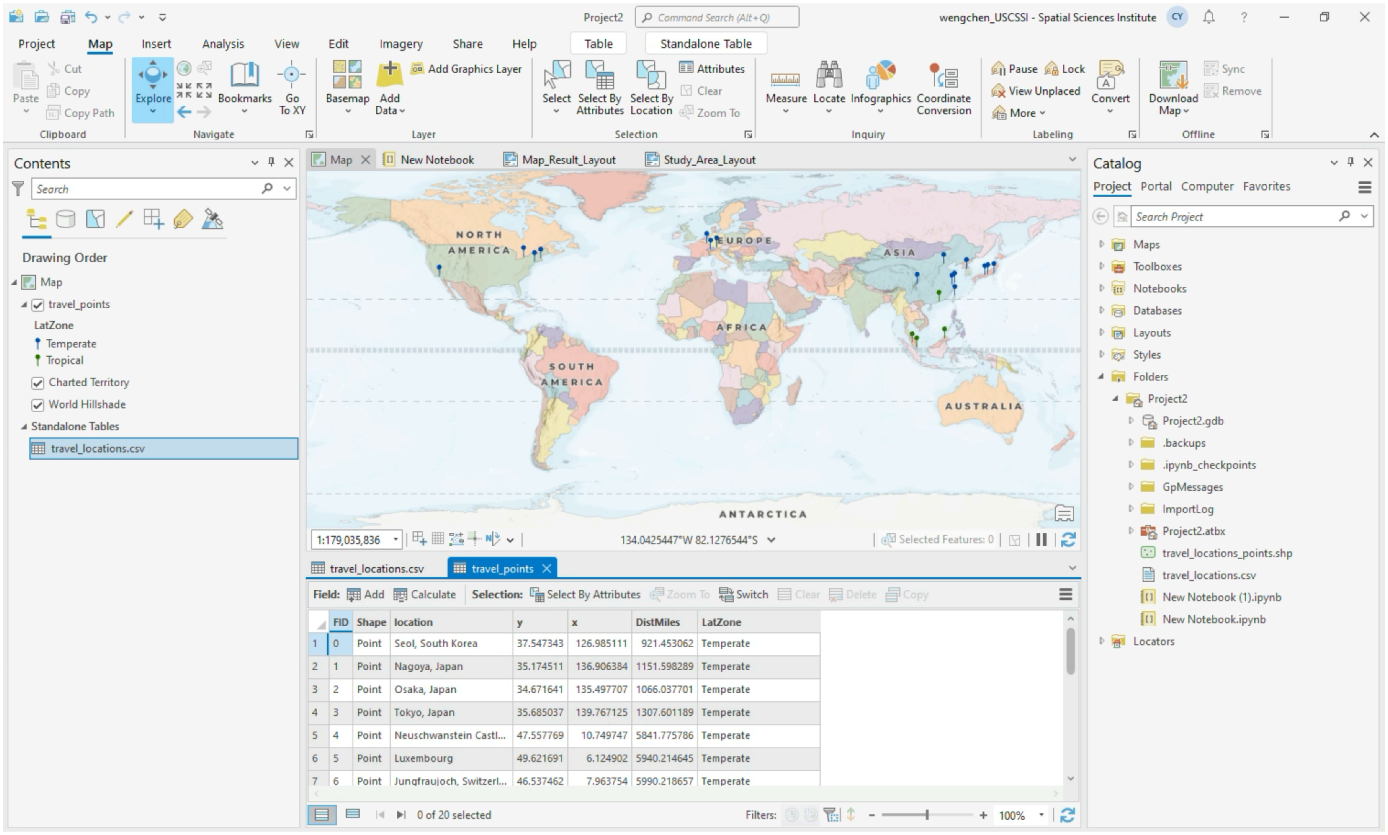


Figure 7. The Final Map Analysis Interface

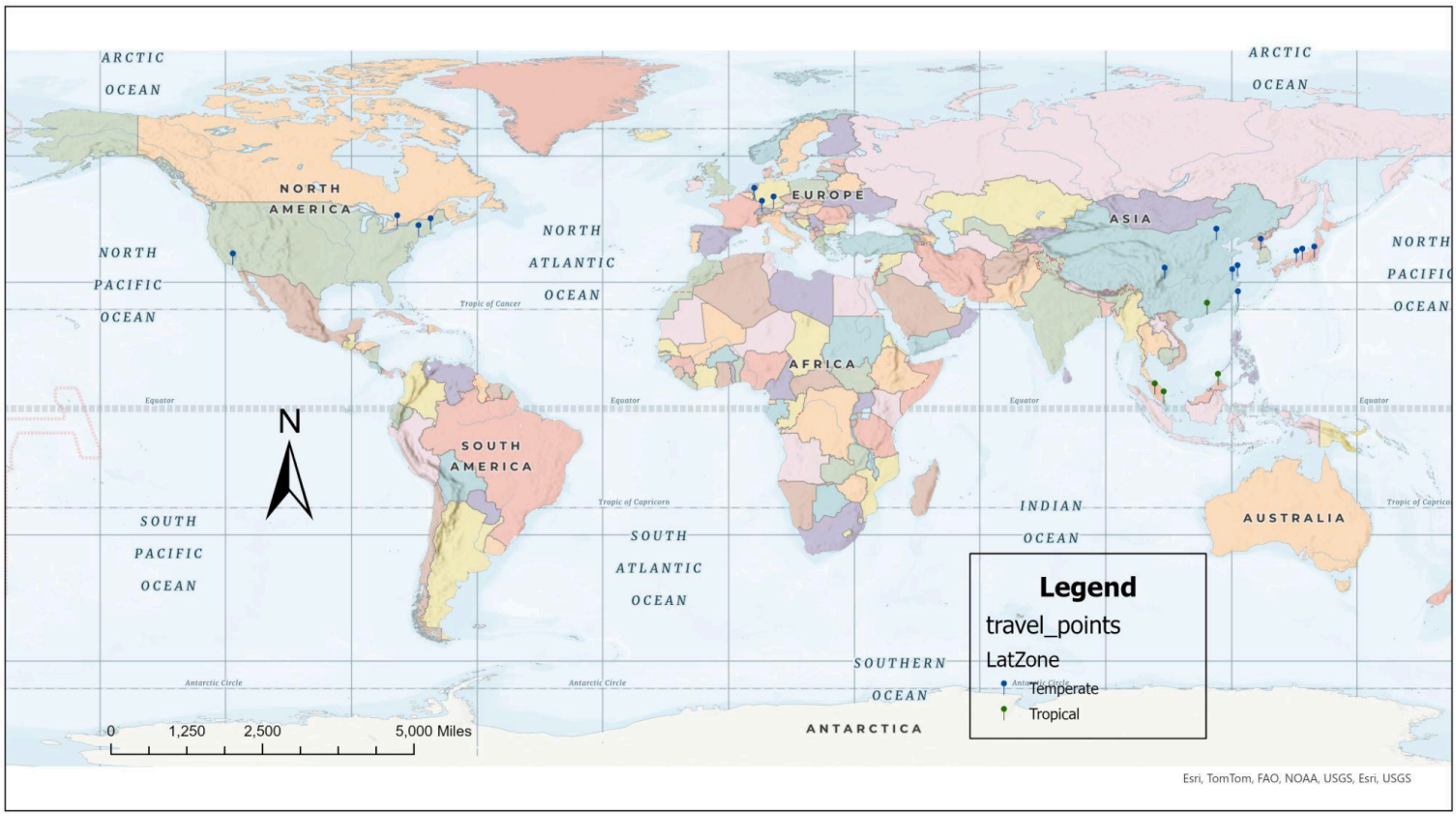


Figure 8. Final Map Categorizing Cities by Latitude Zones

6. Discussion

The results of this project revealed clear trends in travel preferences, which are biased toward temperate cities. This analysis offers valuable insights into how geographic and climate factors influence travel choices. Moreover, the limitations of this analysis include the small sample size of cities and the manual collection of geographic coordinates, which may introduce minor inaccuracies.

Additionally, this workflow could be reused to analyze future travel data or adapted for different geographic analysis tasks. For example, the code could be expanded to include population or economic data for each city, offering deeper insights into other factors influencing travel habits.

References

What is the data access module (available at https://pro.arcgis.com/en/pro-app/arcpy/data-access/what-is-the-data-access-module-.htm).

XY Table to Point (Data Management) (available at https://pro.arcgis.com/en/pro-app/tool-reference/data-management/xy-table-to-point.htm).

Zandbergen, P. (2024). *Python Scripting for ArcGIS Pro*. Chapters 6–11.