Lab-5: Handling Vector Data in Python

Ranjeet Gupta/ SC24M138

Points

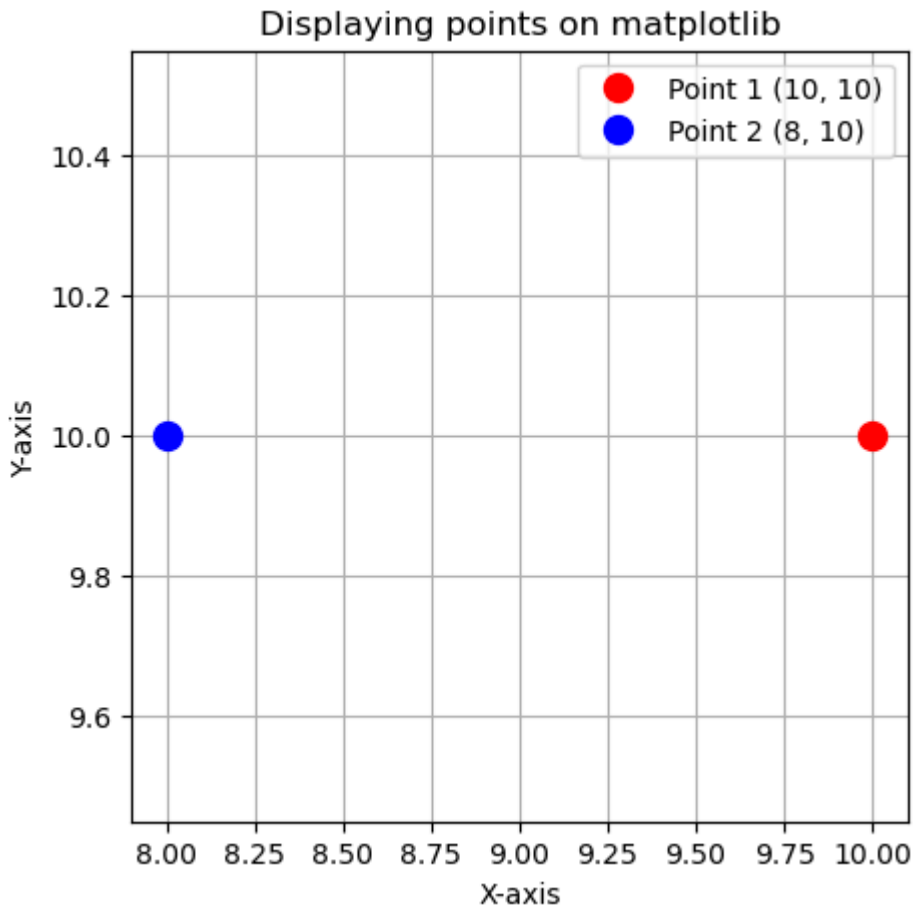
!pip install shapely

1. Create Point geometric object(s) with coordinates

In [2]:
```
from shapely.geometry import Point
point1 = Point(10.0, 10.0)
point2 = Point(8.0, 10.0)
```

2. Display the point on screen

In [3]:
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(5,5))

# Plot the points
ax.plot(point1.x, point1.y, 'ro', markersize=10, label='Point 1 (10, 10)')  # Re
ax.plot(point2.x, point2.y, 'bo', markersize=10, label='Point 2 (8, 10)')  # Blu

ax.set_xlabel("X-axis")
ax.set_ylabel("Y-axis")
ax.set_title("Displaying points on matplotlib")

ax.legend()

plt.grid(True)
plt.show()
```



3. Print the Points

In [4]:
```
print("Point 1:", point1)
print("Point 2:", point2)
```

```
Point 1: POINT (10 10)
Point 2: POINT (8 10)
```

4. Display the Type of the Point Data

In [28]:
```
print("Type of point1:", type(point1))
print("Type of point2:", type(point2))
```

```
Type of point1: <class 'shapely.geometry.point.Point'>
Type of point2: <class 'shapely.geometry.point.Point'>
```

5. Getting the xy coordinate of points

In [7]:
```
# Get the x and y coordinates of each point
print(f"Point 1 - x: {point1.x}, y: {point1.y}")
print(f"Point 2 - x: {point2.x}, y: {point2.y}")

# Using coords to get the coordinates as tuples
print(f"Point 1 Coordinates: {(point1.coords)}")
print(f"Point 2 Coordinates: {list(point2.coords)}")
```

```
Point 1 - x: 10.0, y: 10.0
Point 2 - x: 8.0, y: 10.0
Point 1 Coordinates: <shapely.coords.CoordinateSequence object at 0x0000023177741
250>
Point 2 Coordinates: [(8.0, 10.0)]
```

6. Read x and y coordinates separately and Display the coordinates

In [34]:
```
print("Point 1:")
print(f" X - Coordinate: {point1.x}")
print(f" Y- Coordinate: {point1.y}")

print("Point 2:")
print(f" X - Coordinate: {point2.x}")
print(f" Y- Coordinate: {point2.y}")
```

```
Point 1:
 X - Coordinate: 10.0
 Y- Coordinate: 10.0
Point 2:
 X - Coordinate: 8.0
 Y- Coordinate: 10.0
```

7. Calculating the distance between two points

In [36]:
```
distance = point1.distance(point2)
print(f"The distance between Point 1 and Point 2 is: {distance} units")
```

```
The distance between Point 1 and Point 2 is: 2.0 units
```

In [ ]:
```

```

Linestring

1. Create a LineString from the Point objects

In [1]:
```
from shapely.geometry import Point, LineString
#Create Point geometry objects
point1 = Point(5.0, 10.0)
point2 = Point(15.0, 30.0)
point3 = Point(25.0, 20.0)

line1 = LineString([point1, point2, point3])  # create a linestring
print("LineString :", line1)
```

```
LineString : LINESTRING (5 10, 15 30, 25 20)
```

2. Create a LineString using coordinate tuples

In [2]:
```
#Define Coordinates as tuples
coordinates = [(5.0, 10.0), (15.0, 30.0), (25.0, 20.0)]

line2 = LineString(coordinates)
print(f"LineString: {line2}")
```

```
LineString: LINESTRING (5 10, 15 30, 25 20)
```

3. Check if lines are identical

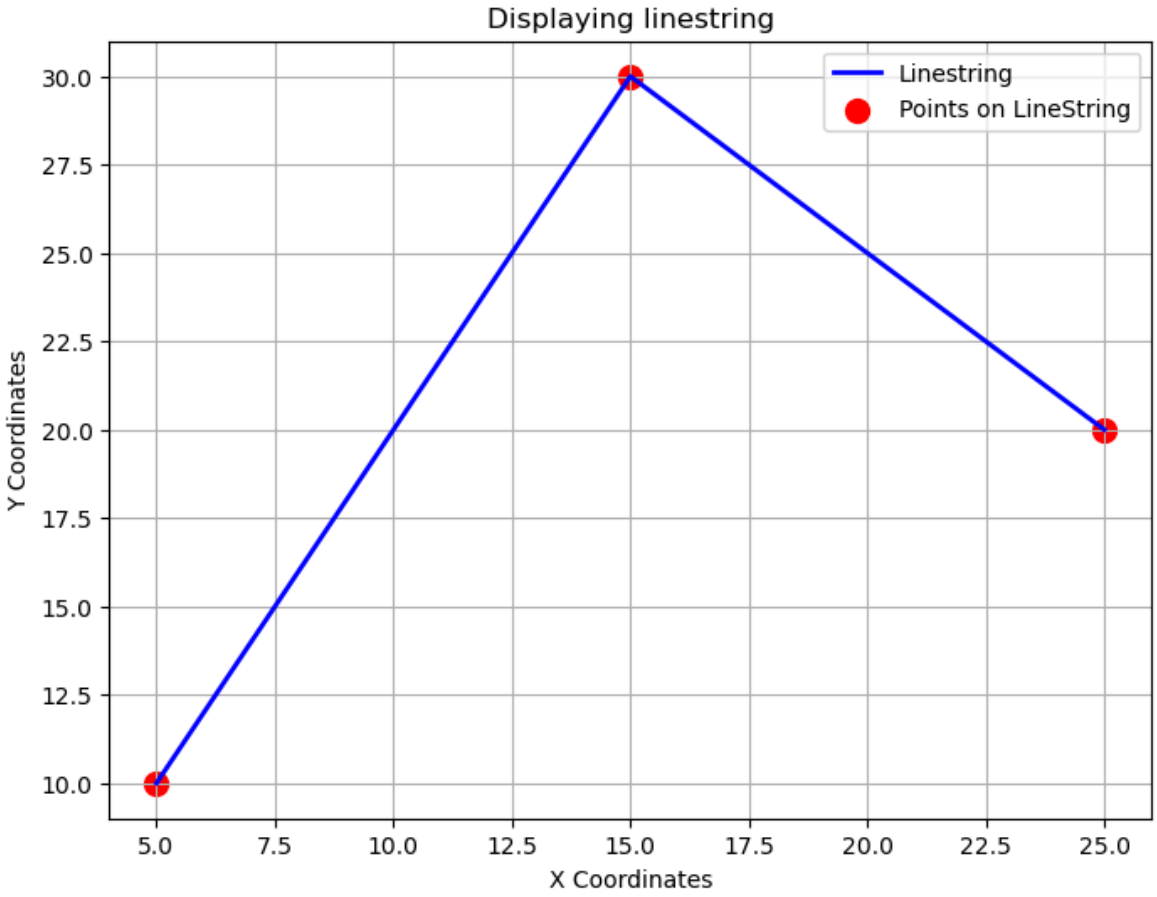
In [3]:
```
b = line1.equals(line2)
print("Are line1 and line2 identical? ->", b)
```

```
Are line1 and line2 identical? -> True
```

4. Display the linestring

In [4]:
```
import matplotlib.pyplot as plt

# Get the x and y coordinates
x, y = line1.xy

# Create a plot
plt.figure(figsize=(8,6))
plt.plot(x, y, color='blue', linewidth=2, label='Linestring')
plt.scatter(x, y, color='red', marker='o', label='Points on LineString', s=100)

# Set labels and title
plt.title('Displaying linestring')
plt.xlabel('X Coordinates')
plt.ylabel('Y Coordinates')

plt.legend()
plt.grid(True)
plt.show
```

Out[4]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



5. Print the Linestring

In [18]:
```
print("Linestring :", line1)  # line1 = LineString([point1, point2, point3])
print("Lnestring :", line2)   #coordinates = [(5.0, 10.0), (15.0, 30.0), (25.0,
```

```
Linestring : LINESTRING (5 10, 15 30, 25 20)
Lnestring : LINESTRING (5 10, 15 30, 25 20)
```

6. Display the Type of the Line Object

In [21]:
```
print("Type of the Linestring 1 Object", type(line1))
print("Type of the Linestring 2 Object", type(line2))
```

```
Type of the Linestring 1 Object <class 'shapely.geometry.linestring.LineString'>
Type of the Linestring 2 Object <class 'shapely.geometry.linestring.LineString'>
```

7. Display the Geometry of the Line Object

In [6]:
```
print("Geometry of the Line Object :", line1.geom_type)  #Returns the type of th
print("Geometry of the Line Object (WKT):", line1.wkt) #Provides the full WKT (i
```

```
Geometry of the Line Object : LineString
Geometry of the Line Object (WKT): LINESTRING (5 10, 15 30, 25 20)
```

8. Get the xy coordinate tuples

In [7]:
```
print("XY Coordinate Tuples:", list(line1.coords))
```

```
XY Coordinate Tuples: [(5.0, 10.0), (15.0, 30.0), (25.0, 20.0)]
```

9. Read x and y coordinates separately and Display the coordinates

In [8]:
```
# Extract x and y coordinates separately
x, y = line1.xy
print("X Coordinates:", x)
print("Y Coordinates", y)
```

```
X Coordinates: array('d', [5.0, 15.0, 25.0])
Y Coordinates array('d', [10.0, 30.0, 20.0])
```

10. Calculate the length of the line

In [9]:
```
# Calculate the length of the LineString
print("Length of the Line:", line1.length)
```

```
Length of the Line: 36.50281539872885
```

11. Calculate the centroid of the line

In [10]:
```
# Calculate the centroid of the LineString
centroid = line1.centroid
print("Centroid of the Line:", (centroid.x, centroid.y))
```

```
Centroid of the Line: (13.874258867227931, 21.937129433613965)
```

In [ ]:
```

```

POLYGON

1. Create a Polygon from the coordinates

In [45]:
```
from shapely.geometry import Polygon

# Define a list of coordinate tuples
coords = [(0,0), (2,0), (3,3), (2,5), (-1,2), (0,0)]
polygon1 = Polygon(coords)
```

2. Create a Polygon based on information from the Shapely points

In [22]:
```
from shapely.geometry import Point

# Create Shapely points
points = [Point(0,0), Point(2,0), Point(3,3), Point(2,5), Point(-1,2), Point(0,0

# Extract coordinates from points to form a Polygon
polygon2 = Polygon([point.coords[0] for point in points])
```

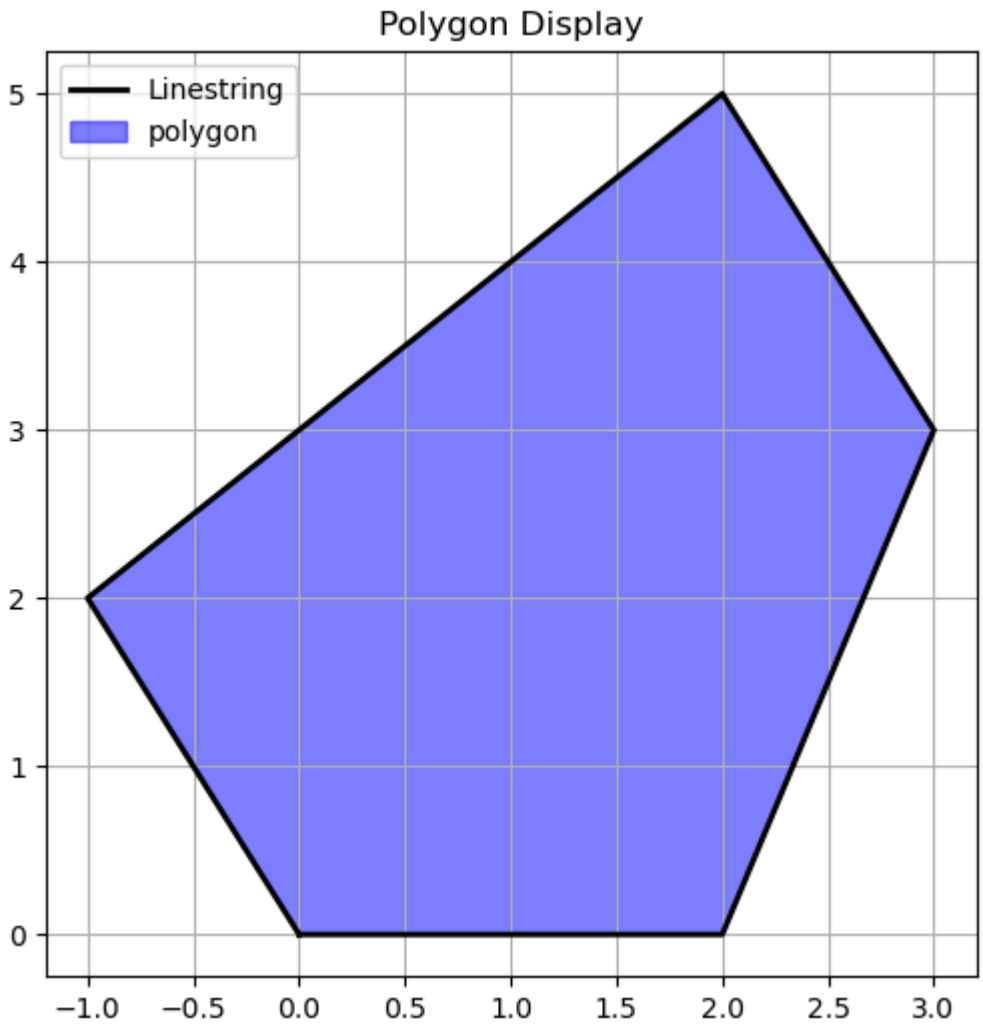
3. Check if Polygons are identical

In [23]:
```
print("Poygons are identical:", polygon1.equals(polygon2))
```

```
Poygons are identical: True
```

4. Display the Polygon on screen

In [24]:
```
import matplotlib.pyplot as plt

# Get x and y coordinates
x, y = polygon1.exterior.xy

plt.figure(figsize=(6,6))
plt.plot(x, y, color='black', linewidth=2, label='Linestring')
plt.fill(x, y, color='blue', alpha=0.5, label='polygon') # Fill the polygon with
plt.title('Polygon Display')
plt.legend()
plt.grid(True)
plt.show()
```



5. Print the Polygon

In [20]:
```
print("Polygon 1:", polygon1)
print("Polygon 1:", polygon2)
```

```
Polygon 1: POLYGON ((0 0, 2 0, 3 3, 2 5, 1 2, 0 0))
Polygon 1: POLYGON ((0 0, 2 0, 3 3, 2 5, 1 2, 0 0))
```

6. Display the type of the polygon object

In [26]:
```
print("Type of the polygon1 object:", polygon1.geom_type)
print("Type of the polygon2 object:", polygon2.geom_type)
```

```
Type of the polygon1 object: Polygon
Type of the polygon2 object: Polygon
```

7. Display the geometry of the polygon object

In [27]:
```
print("Geometry of the Polygon (WKT):", polygon1.wkt)
```

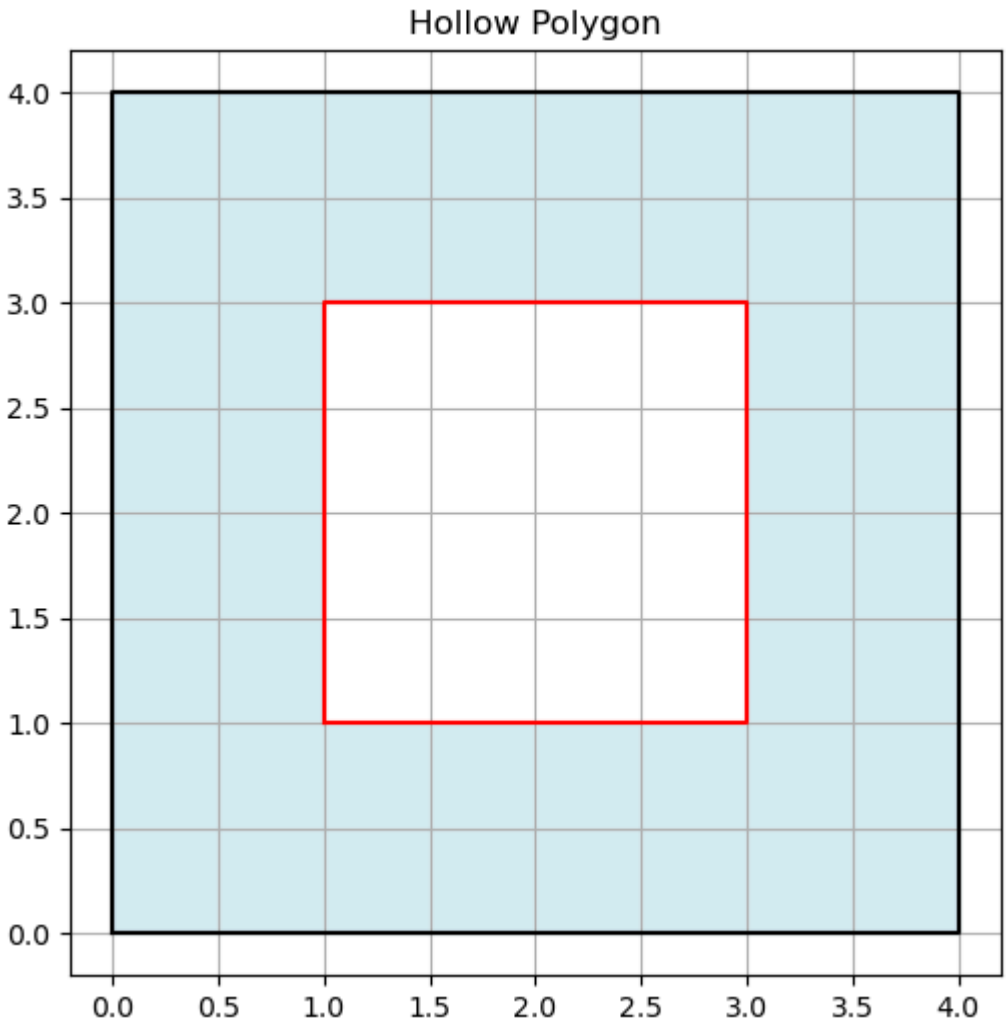
```
Geometry of the Polygon (WKT): POLYGON ((0 0, 2 0, 3 3, 2 5, -1 2, 0 0))
```

8. Create a hollow polygon

In [31]:
```
# Outer boundary
outer = [(0,0), (4,0), (4,4), (0,4), (0,0)]
# Inner boundary (hole)
hole = [(1,1), (3,1), (3,3), (1,3), (1,1)]

hollow_polygon = Polygon(shell=outer, holes=[hole])
```

9. Display the Hollow Polygon

In [32]:
```
x, y = hollow_polygon.exterior.xy  # Exterior coordinates
hx, hy = zip(*hollow_polygon.interiors[0].coords) # Interior coordinates

plt.figure(figsize=(6,6))
plt.plot(x, y, color= 'black')
plt.fill(x, y, color= 'lightblue', alpha=0.5)
plt.plot(hx, hy, color='red') # Interior boundary
plt.fill(hx, hy, color='white')
plt.title('Hollow Polygon')
plt.grid(True)
plt.show()
```



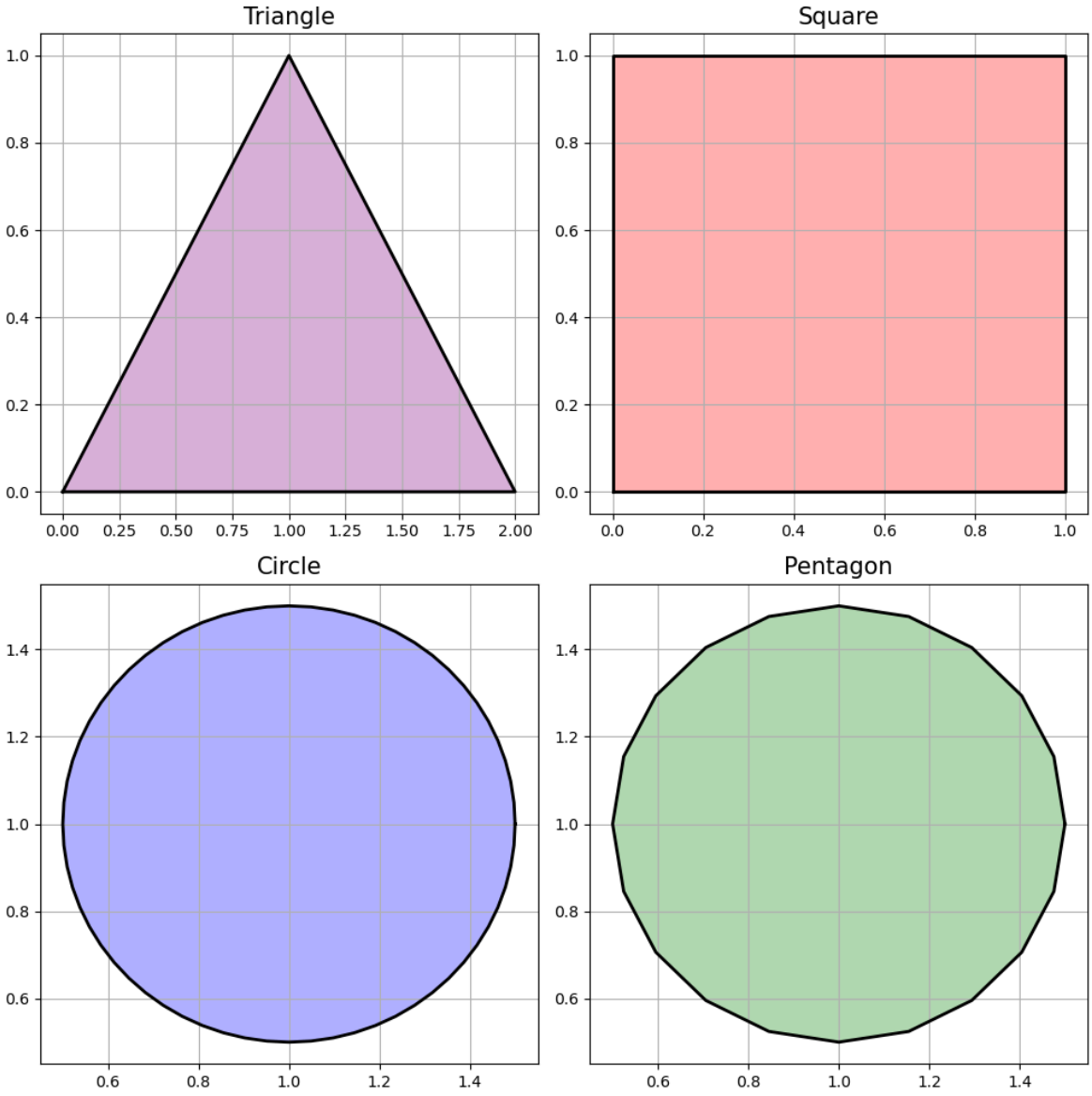
10.Display the parameters of the Polygon such as area, centroid, bounding box, exterior length

In [35]:
```
print("Area:", polygon1.area)
print("Centroid:", polygon1.centroid) # in x & y
print("Bounding box:", polygon1.bounds)   # (minx, miny, maxx, maxy)
print("Exterior length:", polygon1.length)
```

```
Area: 12.0
Centroid: POINT (1.1666666666666667 2.125)
Bounding box: (-1.0, 0.0, 3.0, 5.0)
Exterior length: 13.877054302287245
```

11.Display Geometric Shapes Triangle, Square, Circle, Pentagon

In [44]:
```
from shapely.geometry import Polygon, Point

# Create shapes
triangle = Polygon([(0, 0), (1, 1), (2, 0), (0, 0)])
square = Polygon([(0, 0), (1, 0), (1, 1), (0, 1), (0, 0)])
circle = Point(1, 1).buffer(0.5)  # Approximate circle with buffer
pentagon = Point(1, 1).buffer(0.5, resolution=5)  # Pentagon (buffer with resolu

# Display the shapes
fig, ax = plt.subplots(2, 2, figsize=(10, 10))

# List of shapes and their titles
shapes = [triangle, square, circle, pentagon]
titles = ["Triangle", "Square", "Circle", "Pentagon"]
colors = ['purple', 'red', 'blue', 'green']

for shape, title, color, subplot in zip(shapes, titles, colors, ax.flatten()):
    # Get the exterior coordinates of the shape
    x, y = shape.exterior.xy

    # Plot the shape
    subplot.plot(x, y, color='black', linewidth=2)
    subplot.fill(x, y, color=color, alpha=0.3) # Fill the shape with some color
    # Set title and grid for each subplot
    subplot.set_title(title, fontsize=15)
    subplot.grid(True)

# Display the plot
plt.tight_layout()
plt.show()
```



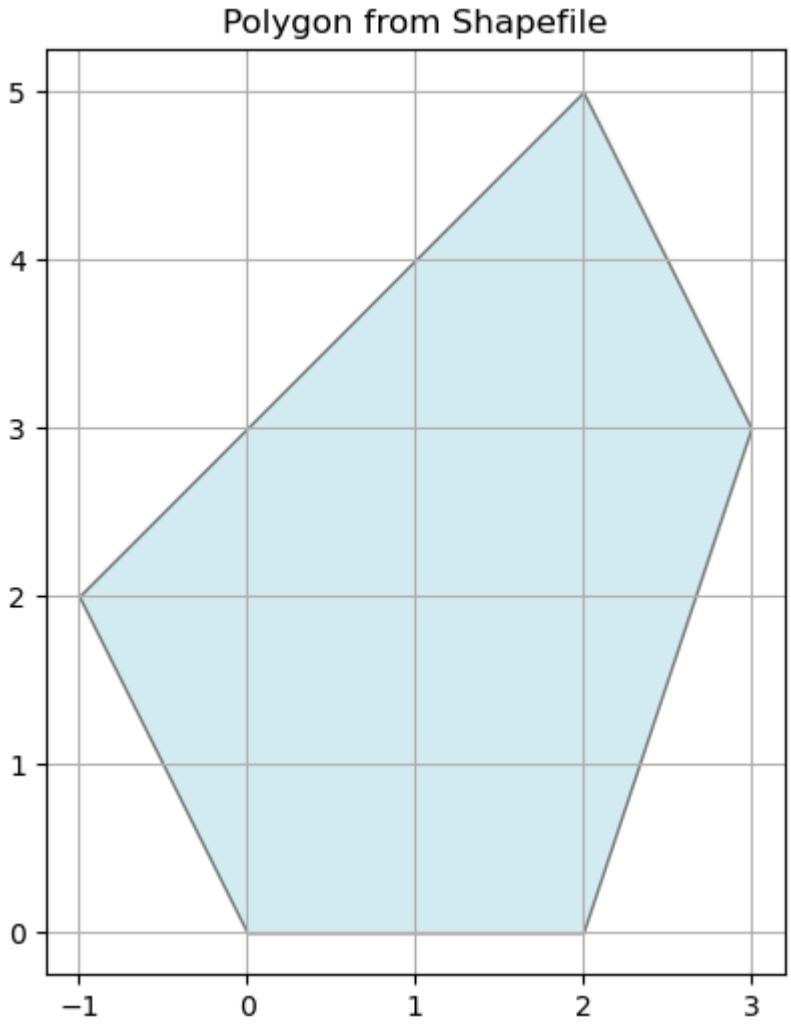
12.Export any shape into shapefile

In [56]:

```
import geopandas as gpd

# Create a GeoDataFrame with the polygon
gdf = gpd.GeoDataFrame([1], geometry=[polygon1], columns=['ID'])   # [1]: This c
                                                                    # (this can

# Assign a CRS (WGS84 - EPSG:4326) for dealing the some warning occured in outpu
gdf.set_crs(epsg=4326, inplace=True)

# Export the Polygon to a Shapefile
shapefilepath = r"C:\Users\Ranjeet Gupta\Downloads\Scientific Computing Lab\Lab-
gdf.to_file(shapefilepath, driver='ESRI Shapefile')
print(f"Shapefile saved at: {shapefilepath}")

# Read the Shapefile (to verify the export)
gdf_loaded = gpd.read_file(shapefilepath)
print("Loaded Shapefile Data")
print(gdf_loaded)

# Display the Polygon from the Shapefile
```

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(6,6))
gdf_loaded.plot(ax=ax, color='lightblue', edgecolor='black', alpha=0.5)
plt.title("Polygon from Shapefile")
plt.grid(True)
plt.show()
```

```
Shapefile saved at: C:\Users\Ranjeet Gupta\Downloads\Scientific Computing Lab\Lab
-5\polygon.shp
Loaded Shapefile Data
   ID                                   geometry
0   1  POLYGON ((0 0, -1 2, 2 5, 3 3, 2 0, 0 0))
```



In [ ]:

Handling Shapefile

1. From the given shapefile, display the number of records.

In [61]:
```
import geopandas as gpd

# Load the shapefile
file_path = r"C:\Users\Ranjeet Gupta\Downloads\Scientific Computing Lab\Lab-5\In
```

```
gdf = gpd.read_file(file_path)

print("Number of records", len(gdf))
```

```
Number of records 36
```

2. Display the projection system.

In [64]:
```
print("Projection System (CRS):", gdf.crs)
```

```
Projection System (CRS): EPSG:3857
```

3. Make a copy of the file in the working directory

In [66]:
```
copy_path = "Copy_of_India_State_Boundary_Copy.shp"
gdf.to_file(copy_path)

print(f"Shapefile copied successfully to: {copy_path}")
```

```
Shapefile copied successfully to: Copy_of_India_State_Boundary_Copy.shp
```

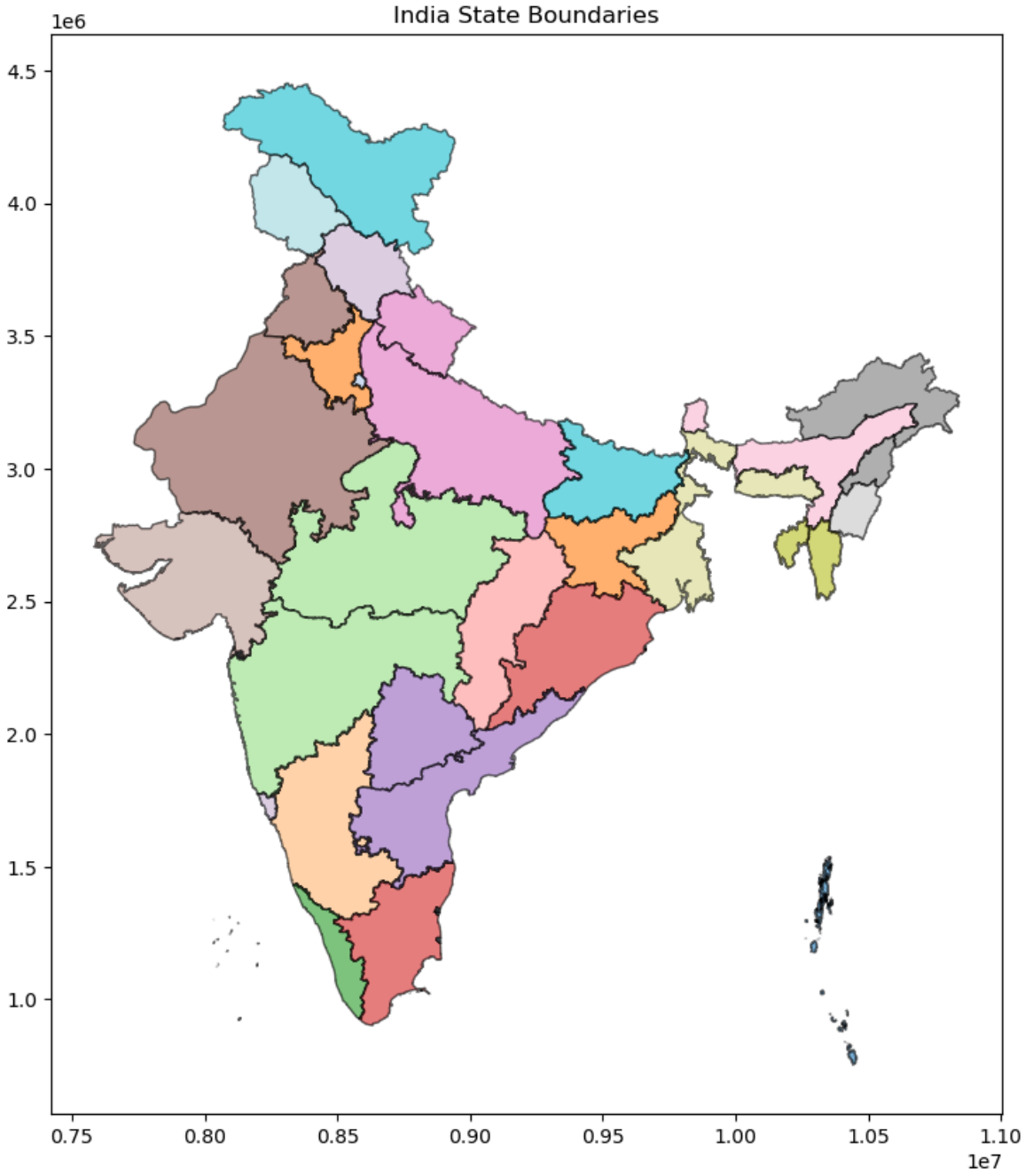
4. Compute the area of the Polygons.

In [70]:
```
# Note: Assumes the shapefile has a geographic CRS, so area might need projectio
gdf['Area'] = gdf.to_crs(epsg=6933).area  # EPSG:6933 = World Equal Area
print(gdf[['State_Name', 'Area']].head())  #This attributes is get to know from
```

```
                                State_Name          Area
0                         Andaman & Nicobar  7.313767e+09
1                                Chandigarh  1.146091e+08
2  Daman and Diu and Dadra and Nagar Haveli  5.812267e+08
3                                     Delhi  1.484868e+09
4                                   Haryana  4.419171e+10
```

5. Plot the data

In [72]:
```
fig, ax = plt.subplots(figsize=(10, 10))
gdf.plot(ax=ax, edgecolor='black', alpha=0.6, cmap='tab20')  #Qualitative Colorm
plt.title("India State Boundaries")
plt.show()
```



6. From the given shapefile, find out the entry with largest and smallest area.

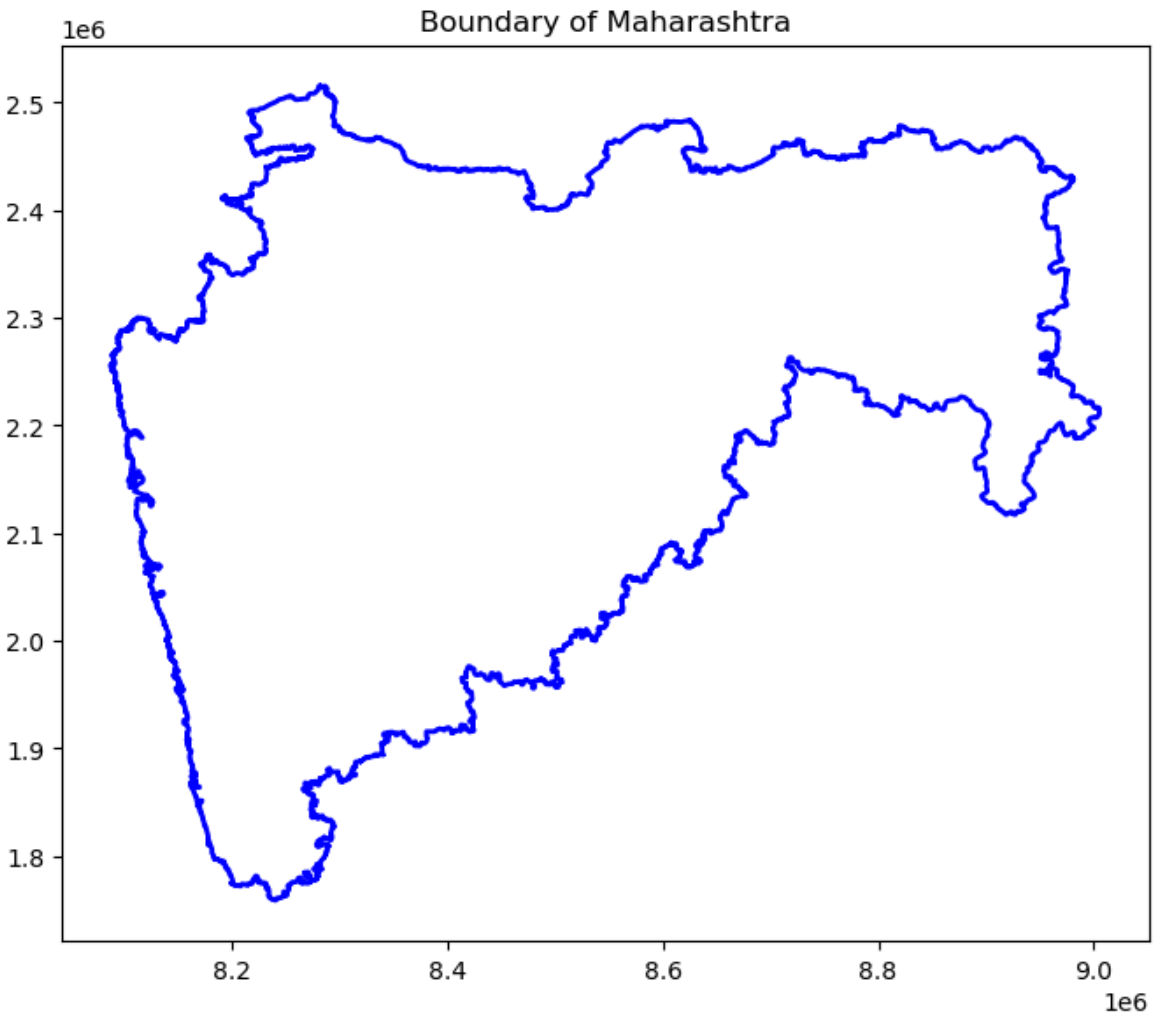
In [73]:
```
largest_area_state = gdf.loc[gdf['Area'].idxmax()]
smallest_area_state = gdf.loc[gdf['Area'].idxmin()]
print("Largest Area State:", largest_area_state['State_Name'], "Area:", largest_
print("Smallest Area State:", smallest_area_state['State_Name'], "Area:", smalle
```

```
Largest Area State: Rajasthan Area: 342175249621.412
Smallest Area State: Lakshadweep Area: 33252190.169754043
```

7. Extract the boundary of your homestate and project it into the appropriate coordinate system.

In [79]:
```
homestate = gdf[gdf["State_Name"] == 'Maharashtra']
homestate_boundary = homestate.boundary
print(f"Boundary extracted for {homestate.iloc[0]['State_Name']}")  #homestate.i
                                                                    # contains only one stat

fig, ax = plt.subplots(figsize=(8,8))
homestate_boundary.plot(ax=ax, color='blue', linewidth=2)
plt.title("Boundary of Maharashtra")
plt.show()
```

```
Boundary extracted for Maharashtra
```



8. Attempt to change the projection and save it as new shapefile

In [86]:
```
new_crs = "EPSG:32643"  # UTM Zone 43N (India region)
gdf_projected = gdf.to_crs(new_crs)

# Reduce precision of area values (for example, limit to 2 decimal places)
gdf['Area'] = gdf['Area'].round(2)

projected_path = "change_projection_India_State_Boundary_UTM43N.gpkg"
gdf_projected.to_file(projected_path, driver='GPKG') # Save as GeoPackage
print(f"Projection changed and saved as {projected_path}.")
```

```
gdf_projected.crs
```

```
Projection changed and saved as change_projection_India_State_Boundary_UTM43N.gpk
g.
```

Out[86]:
```
<Projected CRS: EPSG:32643>
Name: WGS 84 / UTM zone 43N
Axis Info [cartesian]:
- E[east]: Easting (metre)
- N[north]: Northing (metre)
Area of Use:
- name: Between 72°E and 78°E, northern hemisphere between equator and 84°N, on
shore and offshore. China. India. Kazakhstan. Kyrgyzstan. Maldives. Pakistan. R
ussian Federation. Tajikistan.
- bounds: (72.0, 0.0, 78.0, 84.0)
Coordinate Operation:
- name: UTM zone 43N
- method: Transverse Mercator
Datum: World Geodetic System 1984 ensemble
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
```