

# Instruction Document: Handling and Visualizing Geospatial Data using GeoPandas

## Introduction

This guide covers essential operations using GeoPandas for handling and visualizing geospatial data. You'll learn how to load geographic datasets, perform spatial operations, and create visualizations of geographic features like garbage collection zones.

## 1. Installing Required Libraries

Ensure you have Python installed, along with GeoPandas and other required libraries:

pip install geopandas shapely matplotlib

## 2. Loading the Dataset

GeoPandas supports reading multiple formats such as shapefiles, GeoJSON, and CSV files containing geographic coordinates.

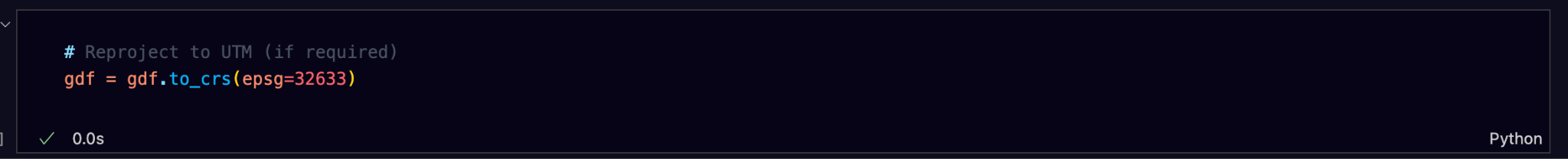
Here, we load a CSV file that contains information about garbage collection zones, including latitude, longitude, and geo\_shape (polygon information):



## 3. Reprojecting to UTM (if needed)

Reprojecting a GeoDataFrame is essential when working with different datasets that may have different coordinate reference systems (CRS). GeoPandas makes it easy to convert between CRSs.

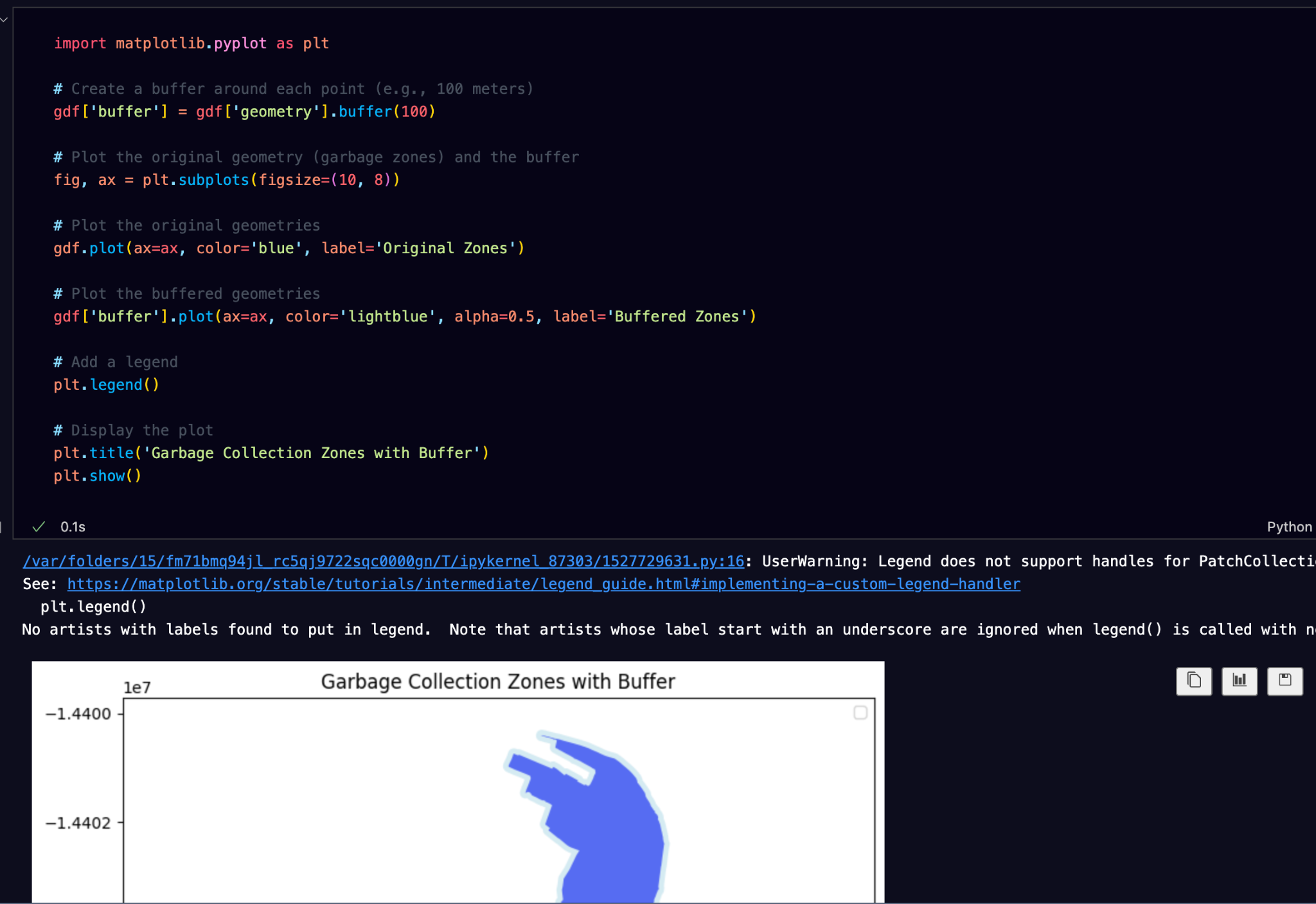
* **to\_crs()**: Reprojects the geometries to a different CRS. For example, EPSG:32633 represents UTM Zone 33N, a projected CRS often used for area-based calculations.
* **Why reproject?**: Reprojecting is necessary for accurate spatial measurements (like area and distance), as geographic CRS like EPSG:4326 distort distances and areas at different latitudes.



## 4. Spatial Operations

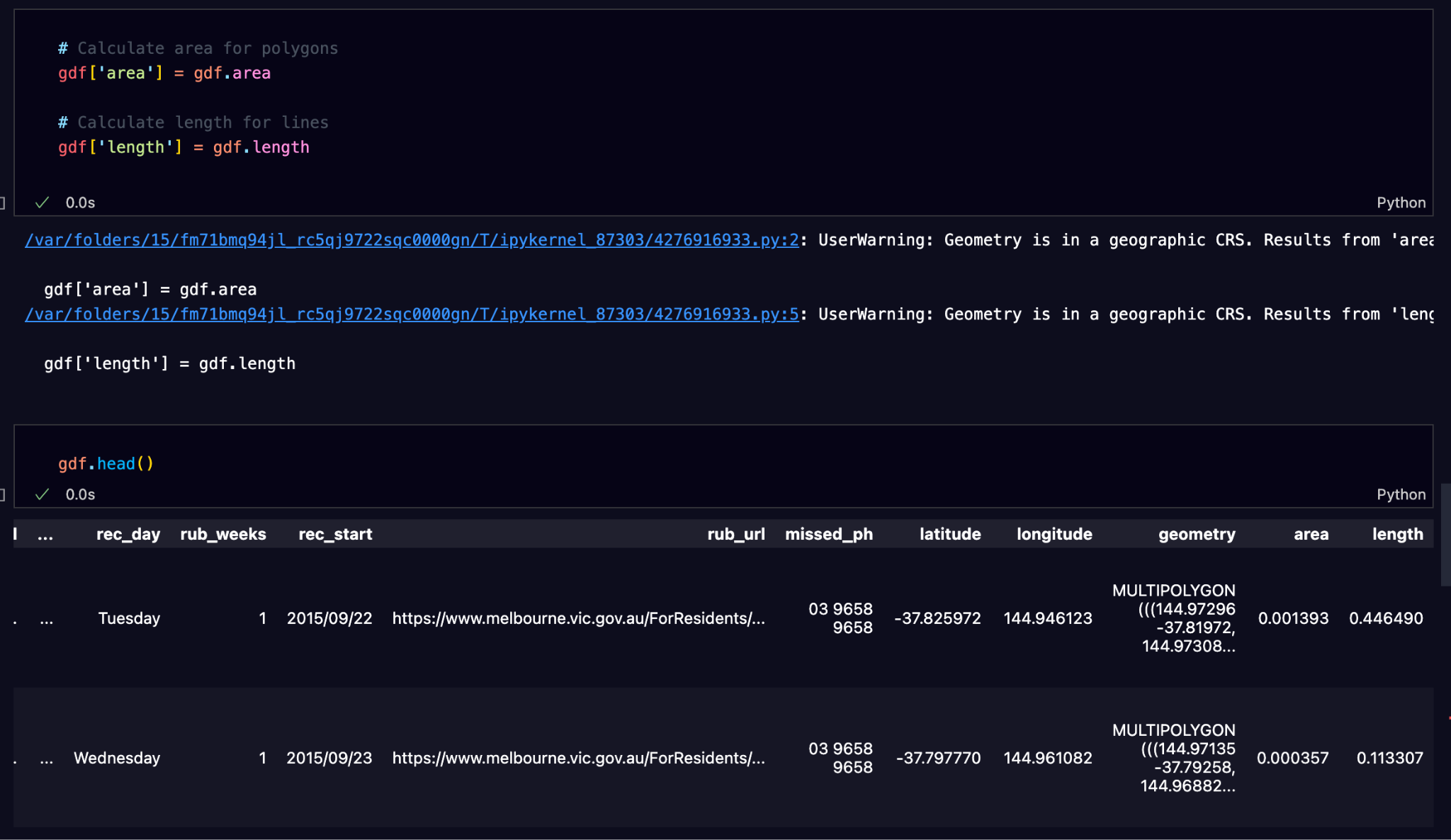
### **a) Buffering**

You can create a buffer around each geometry to represent proximity zones, like buffering garbage collection zones by 100 meters:



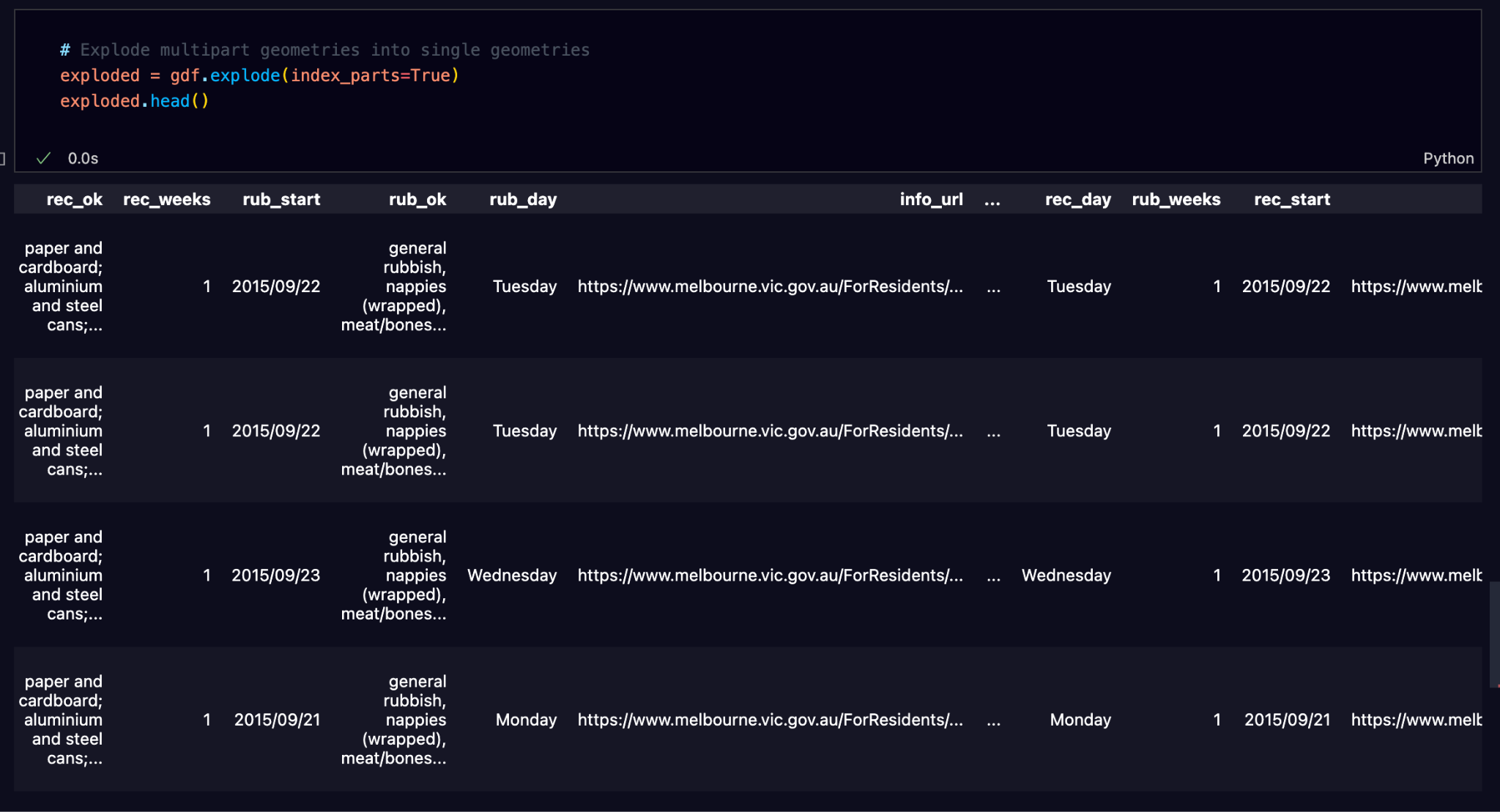
### **b) Calculating Area and Length**

To calculate the area of polygons or the length of lines in your dataset:



### **c) Exploding Multi-Part Geometries**

GeoPandas can handle geometries composed of multiple parts (e.g., a multi-polygon). If you need each part as a separate geometry, you can use the explode() function. **explode()**: Splits multi-part geometries into individual geometries. Each part becomes a separate row in the GeoDataFrame. This is useful when dealing with complex geometries like administrative boundaries or multi-part features.



### **d) Convex Hull**

The convex hull is the smallest convex polygon that can enclose a geometry. It is useful for summarizing the extent of points or polygons. **convex\_hull**: Calculates the convex hull for each geometry in the GeoDataFrame. This operation is useful for understanding the overall spatial extent of a set of geometries.

A black rectangular object with a black background

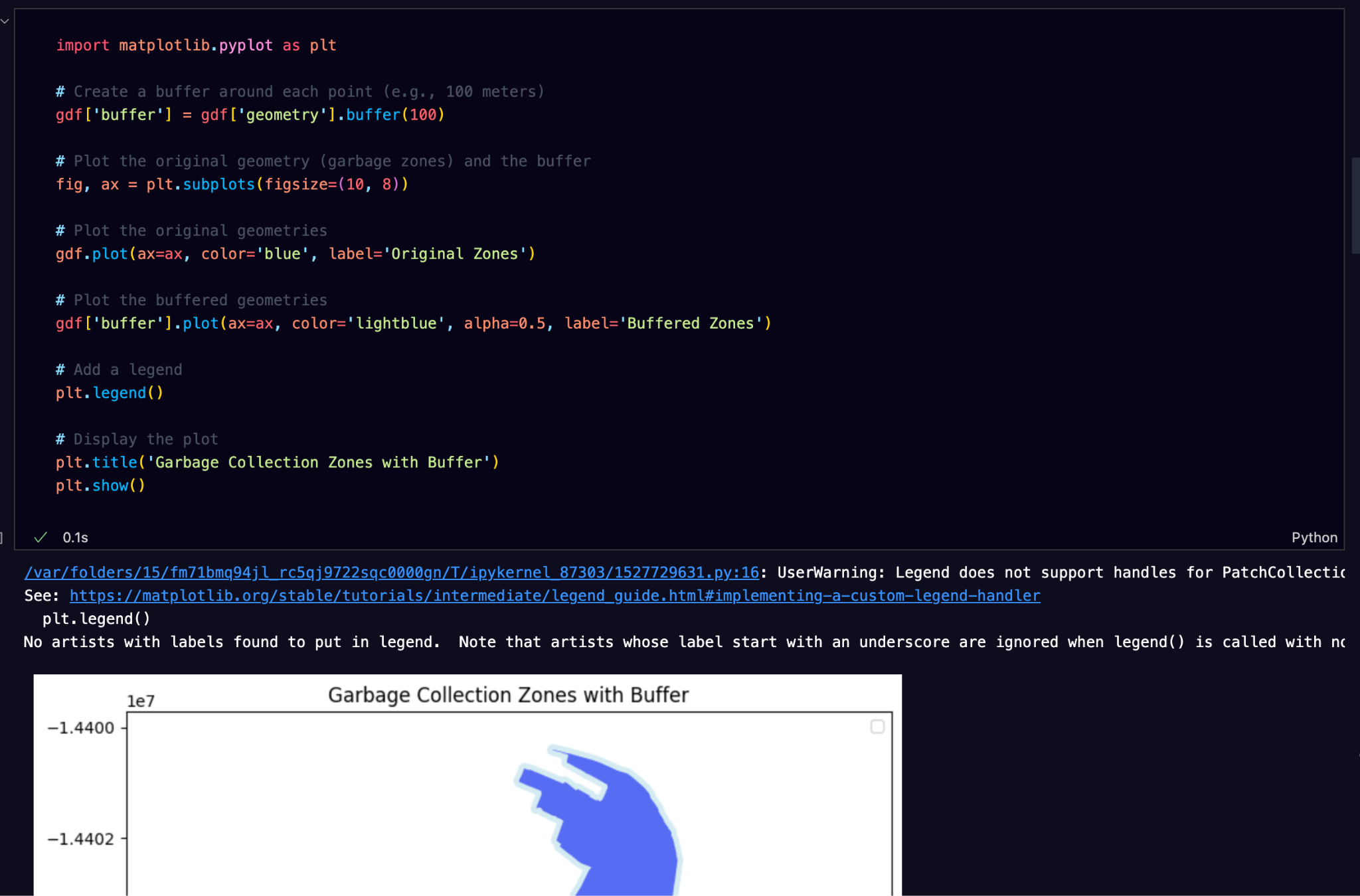
Description automatically generated

## 

## 5. Visualizing Geospatial Data

### **a) Basic Plot**

You can easily plot geographic data using the built-in plotting functionality of GeoPandas:



### 

### **b) Choropleth Map**

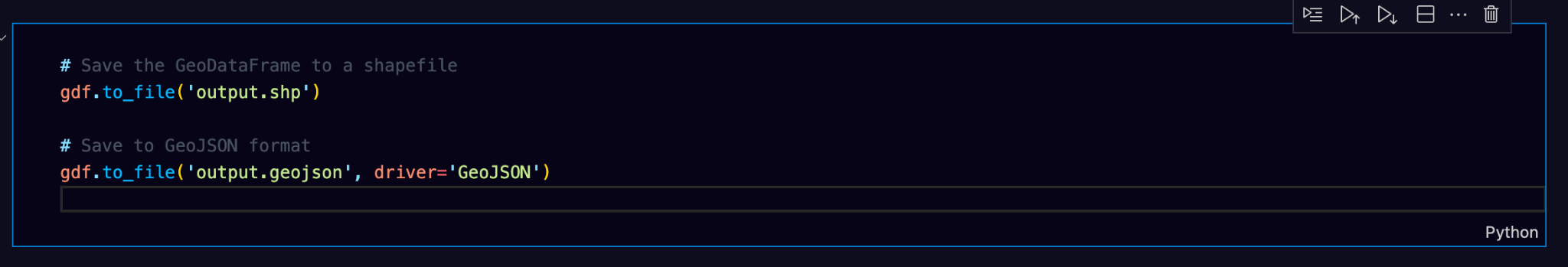
To create a choropleth map based on a specific column (e.g., day of waste collection):



## 6. Saving Processed Data

You can export the results of your geospatial analysis back into standard geographic formats like shapefiles or GeoJSON.

* **to\_file()**: Saves the GeoDataFrame into a file format. You can choose to save it as a shapefile (.shp) or GeoJSON (.geojson) depending on your needs.
* **driver='GeoJSON'**: Specifies the output format. GeoPandas uses shapefiles by efault unless the driver is specified.



## 

## Conclusion

This guide has covered essential operations using GeoPandas, including loading geographic data, performing spatial operations such as buffering, calculating areas and lengths, and visualizing geospatial data. These tools are invaluable for handling geographic datasets and performing spatial analyses such as optimizing waste management routes or understanding spatial distribution patterns.