Part 1

7. What does CRS mean, and why does it matter for mapping and area-based distortion?

Coordinate Reference System (CRS) defines the spatial data's coordinates, in fact the location of data. It is important because different layers must share the same CRS to align correctly, and projected CRSs either in meters or feet are necessary for accurate area measurements and meaningful polygon distortion in cartograms. In my own experience, when I first ran the code, I didn't get the correct results because the raster and polygon layers were in different CRSs, so they didn't align properly until I reprojected them to a common CRS.

8. What does gdf.total_bounds return, and how would you use it to sanity-check your data?

gdf.total_bounds returns the minimum and maximum x and y coordinates of all the geometries in a GeoDataFrame as [minx, miny, maxx, maxy]. I used it in my project to quickly check whether my tracts and study area were in the correct location and roughly matched the expected extent of Mecklenburg County. At first, I noticed the bounds didn't make sense, which tipped me off that my layers were in different CRSs before I reprojected them.

9. Which line reprojects the data, and what would happen if WORK_EPSG is invalid?

The line is "tracts_m = tracts.to_crs(WORK_EPSG)", it changes the CRS of the GeoDataFrame to the working projection (WORK_EPSG). If WORK_EPSG is invalid, Python will produce an error, the data will not be reprojected, and any subsequent analysis (like overlaying layers, calculating areas, or creating cartograms) will fail or give incorrect results. In my case, using an incorrect CRS initially caused my layers not to align, and I had to fix the EPSG code before proceeding.

Part 2

10. Which parameter controls the number of classes (bins)?

The **k** parameter controls the number of classes (bins) in the choropleth. For example, **k=5** divides the data into five bins based on the chosen classification scheme. I noticed that if the data has very few unique values, k may be automatically reduced, which happened when my LST data had many repeated values.

11. Which parameter changes the color palette? Give two valid cmap examples.

The **cmap** parameter sets the color palette for the map. Two examples of valid color maps are "Blues" and "Reds". I experimented with both for my LST and Tree Cover maps to see which made patterns clearer visually.

12. Which parameters control polygon boundary appearance (color and line width)?

edgecolor controls the boundary color, and linewidth controls the thickness of polygon edges. In my maps, I used **edgecolor="white"** and **linewidth=0.3** to make tract boundaries clear but not too visually dominant.

13. What does scheme='Quantiles' mean, and how is it different from 'EqualInterval'?

scheme='Quantiles' divides the data into classes that each contain roughly the same number of features, so each bin has an equal count of tracts. In contrast, **'EqualInterval'** divides the data range into equal numeric intervals, which can result in bins with very different numbers of tracts. In my experience, Quantiles worked better for LST because some tracts had similar values and I wanted evenly distributed classes.

Part 3

14. What does pd.qcut do, and why might duplicates='drop' be necessary?

pd.qcut divides a numeric variable into quantile-based bins, so that each bin contains approximately the same number of observations. The argument **duplicates='drop'** is needed when the data has repeated or tied values that would prevent creating the exact number of bins requested. In my dataset, some tracts had identical Tree Cover values, so this option avoided errors.

15. How is bi_class computed? Explain the formula in one sentence.

bi_class is computed as y_bin * N + x_bin, which combines the row (Y variable) and column (X variable) bin indices into a single integer representing each bivariate class.

16. Which lines would you change to switch from a 3x3 to a 4x4 bivariate map?

In your workflow, the bivariate map bins are created with these lines:

N = 3

tracts_clip["x_bin"] = pd.qcut(tracts_clip[X], N, labels=False, duplicates="drop")

tracts_clip["y_bin"] = pd.qcut(tracts_clip[Y], N, labels=False, duplicates="drop")

To switch from 3×3 to 4×4, I would change N = 3 to N = 4 and recalculate x_bin and y_bin using pd.qcut. I would also create a new 16-color palette to match the 4×4 classification.

17. Which variable controls the colors used for the map?

In the workflow, the variable that actually determines the colors is: tracts_clip["bi_color"]

These variable maps the bi_class values to the colors in your palette, so the color=tracts_clip["bi_color"] argument in tracts_clip.plot() sets the polygon colors on the map.

Part 4

18. Why must the cartogram field be positive?

I made sure the CART_FIELD (LST_mean) values were positive because cartogram distortion relies on scaling polygons proportionally to the numeric values. Negative or zero values would break the geometry scaling and produce invalid or inverted polygons.

19. What do max_iterations and max_average_error control?

I set max_iterations=60 and max_average_error=0.05 to control how long the cartogram algorithm runs and how much distortion error is allowed. More iterations let the polygons adjust better, while a lower error threshold ensures the final map closely reflects the data.

20. What is c.average_error measuring (in plain language)?

The c.average_error measures how closely the distorted polygon areas match the intended values from CART_FIELD. It means, it tells me how accurate the cartogram is, the smaller the error, the better the map represents the data.

21. In the fallback method, why do we use sqrt(value/mean) rather than value/mean directly?

I learned that using sqrt (value/mean) smooths the scaling, preventing extreme distortion of very large or very small values. It gives a more visually balanced cartogram, avoiding overly stretched or shrunken polygons.