Geographic Data and Spatial ETL

Attribute Operations, Pre-processing, Workflows

Alexander Abuabara

DAEN 489 ISEN 489 Spring 2026

ENGINEERING
TEXAS A&M UNIVERSITY

Plan for Today

1. Previously
2. Attribute Data in Spatial ETL
   - Intro to Thematic Maps
3. Vector Attribute Operations
4. Raster Attribute Operations
5. Pre-Processing Workflows in Spatial ETL
6. Questions to Practice
7. Assignment #2

Previously

Spatial Data Models

- Vector vs Raster
- Coordinate reference systems (CRS)

Spatial Objects

- `sf` objects: geometry + attributes
- `terra` rasters: value grids with implicit spatial metadata

Spatial Data Models

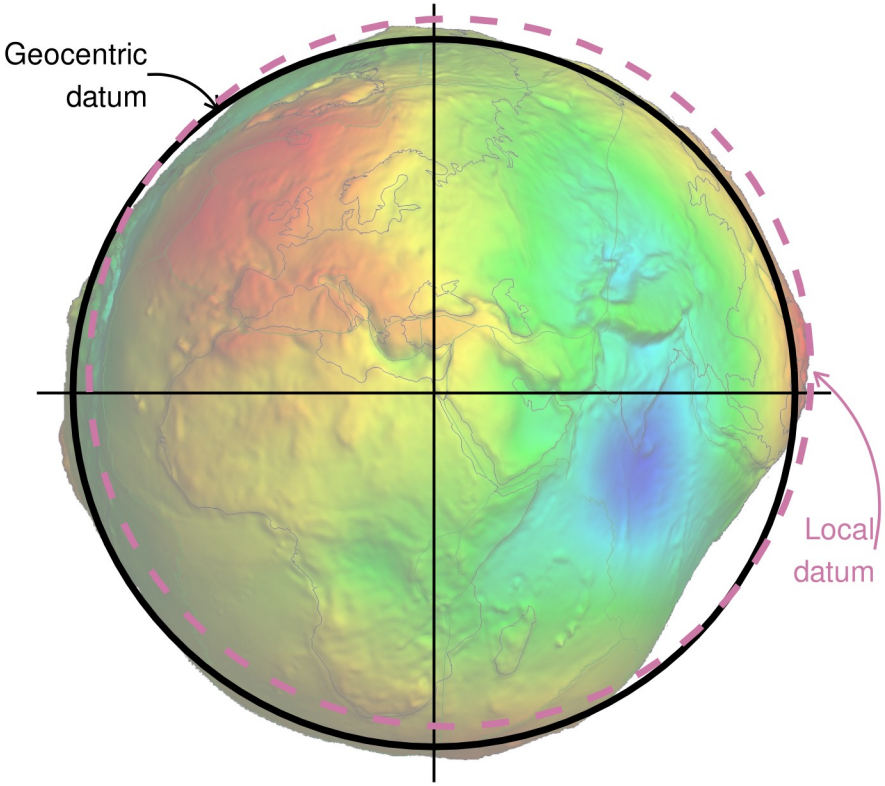
Vector

- Points, Lines, Polygons
- Discrete features with attributes
- `sf` package

Raster

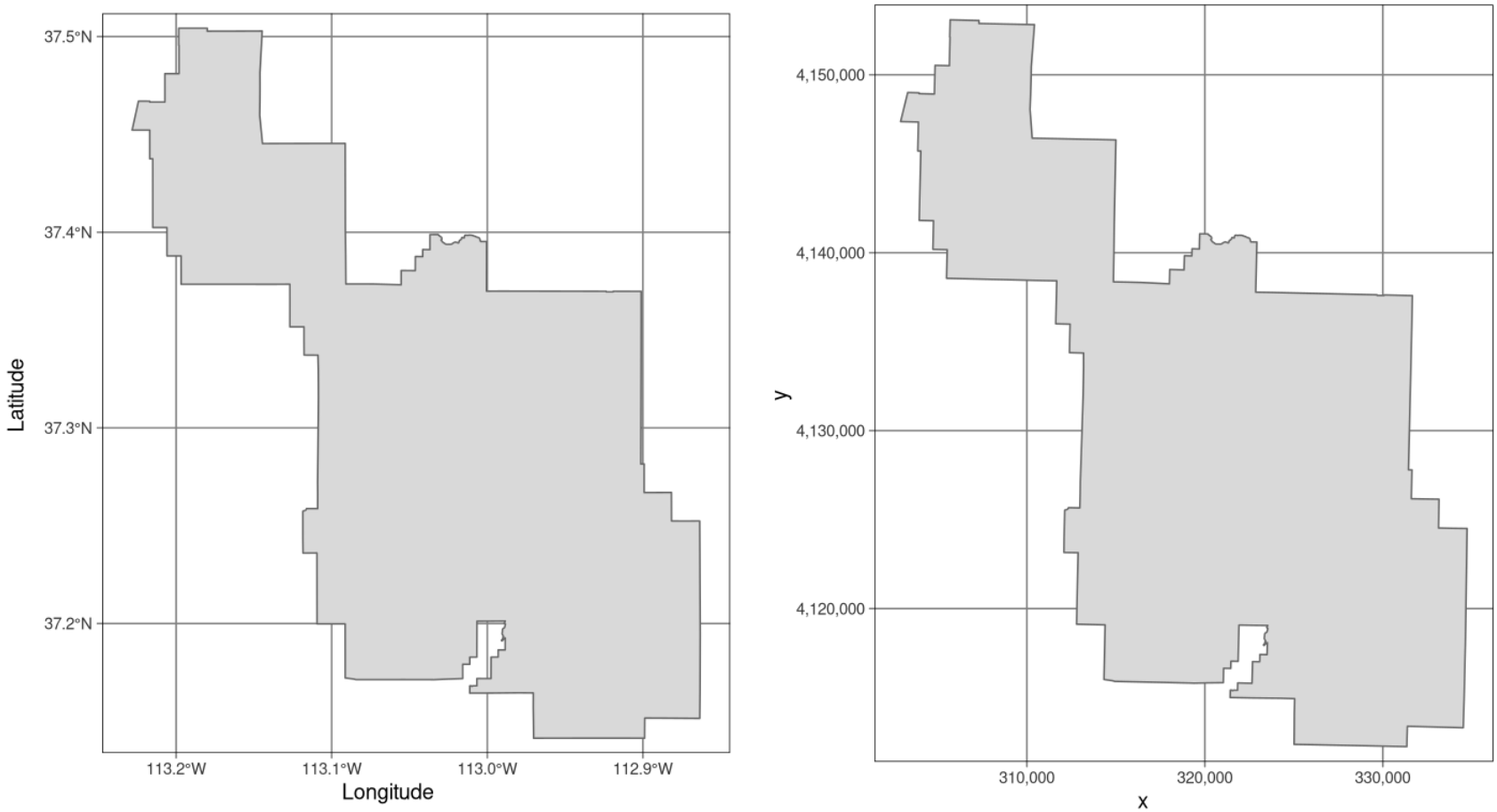
- Grid cells with values
- Continuous surfaces
- `terra` package

Coordinate Reference Systems (CRS)



Geocentric and local geodetic datums shown on top of a geoid (in false color and the vertical exaggeration by 10,000 scale factor). Image of the geoid is adapted from the work of Ince et al. (2019).

CRS Example



Examples of geographic (WGS 84; left) and projected (NAD83 / UTM zone 12N; right) coordinate systems for a vector data type.

Some Essential R Packages

`dplyr`

- Pipe operators (`%>%`)
- `filter()`, `select()`, `mutate()`
- `group_by()`, `summarize()`
- Works with sf objects

`sf`

- Vector data handling
- Integrates with `tidyverse`
- Sticky geometry column
- Fast spatial operations

`spData`

- Datasets for Spatial Analysis

`units`

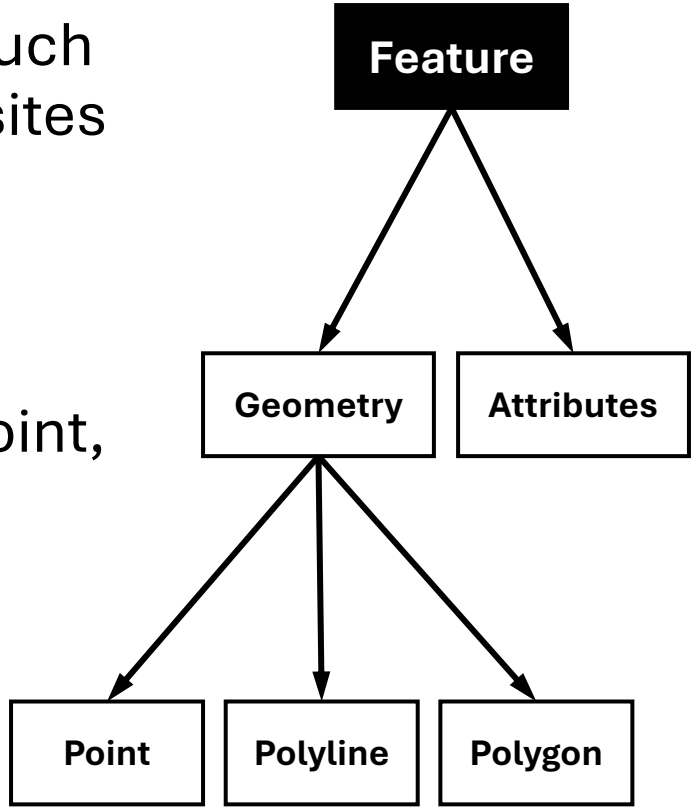
- `sf` objects have native support for `units`

`terra`

- Raster data handling
- Fast computation
- Multi-layer support
- Memory efficient

Features and Attribute Data

- **Features** are real world things such as roads, property boundaries, sites and so on
- A feature has a:
  - **Geometry**
    (which determines if it is a point, polyline or polygon)
  - **Attributes**
    (which describe the feature)



Attribute Data

What are attributes?

- Non-spatial data combined with spatial features
  (Examples: bus stop **name**, population **count**, elevation **value**, land use **category**)

Difference between geometry and attribute data

- Geometry data defines the spatial location and shape of features using coordinates, answering “where” an object is
- Attribute data describes the non-spatial, qualitative, or quantitative characteristics of those features, such as names, types, or values, answering “what” the object is
- They are linked by unique identifiers in GIS

Attribute Table

Attribute

Record

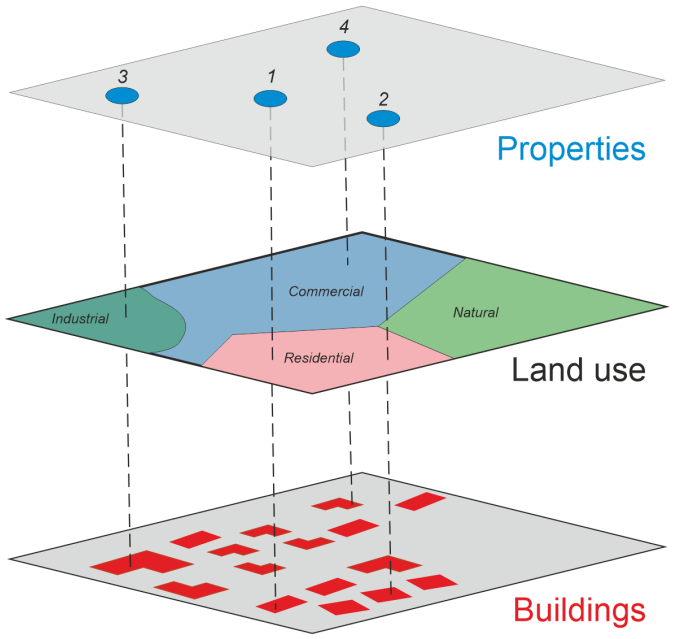
GEOID	NAME	total_pop	area	density
48319	Mason County, Texas	3953	2413.9534 [km^2]	1.63756266
48425	Somervell County, Texas	9205	496.9554 [km^2]	18.52278979
48041	Brazos County, Texas	233849	1531.1716 [km^2]	152.72553109
48489	Willacy County, Texas	20164	1631.2412 [km^2]	12.36113917
48303	Lubbock County, Texas	310639	2332.9853 [km^2]	133.15086132
48235	Irion County, Texas	1513	2722.3549 [km^2]	0.55576883
48207	Haskell County, Texas	5416	2357.5641 [km^2]	2.29728638
48053	Burnet County, Texas	49130	2646.5072 [km^2]	18.56409056
48283	La Salle County, Texas	6664	3869.9889 [km^2]	1.72196875
48239	Jackson County, Texas	14988	2219.1322 [km^2]	6.75399156
48011	Armstrong County, Texas	1848	2367.0046 [km^2]	0.78073359
48263	Kent County, Texas	753	2338.3738 [km^2]	0.32201867
48459	Upshur County, Texas	40892	1534.8495 [km^2]	26.64235196
48369	Parmer County, Texas	9869	2292.3252 [km^2]	4.30523553
48355	Nueces County, Texas	353178	2256.0068 [km^2]	156.55005781

Attribute Data; Extract, Transform, Load

Why attribute operations are central to ETL

- Essential for filtering, merging, aggregating, and reshaping datasets
- Direct impact on analytics outcomes

Attribute Data Workflow



Layer attributes

Id	Address	N. of rooms	Floor area (m2)	Price
1	Street 1	3	70	€600k
2	Street 2	2	42	€450k
3	Street 3	4	300	€350k
4	Street 4	1	600	€200k

+

Landuse	Land cover	Area (km2)
Residential	Urban fabric	3.2
Residential	Urban fabric	3.2
Industrial	Urban fabric	2.1
Commercial	Urban fabric	5.3

+

Construction year	N. of floors	Elevator
1932	5	Yes
1960	4	No
1999	1	No
2007	2	Yes

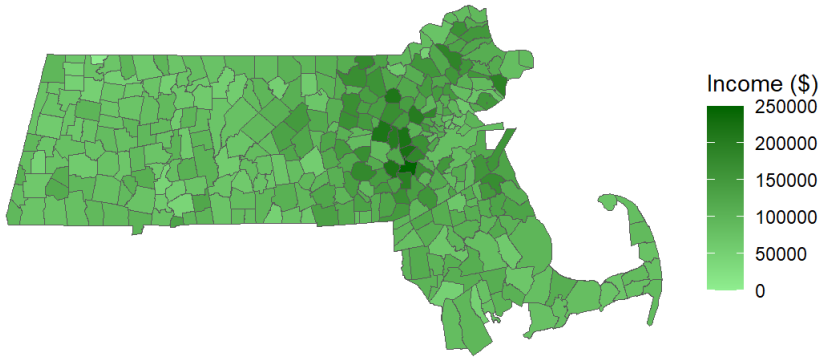
Combined data after spatial join

Id	Address	N. of rooms	Floor area (m2)	Price	Landuse	Land cover	Area (km2)	Construction year	N. of floors	Elevator
1	Street 1	3	70	€600k	Residential	Urban fabric	3.2	1932	5	Yes
2	Street 2	2	42	€450k	Residential	Urban fabric	3.2	1960	4	No
3	Street 3	4	300	€350k	Industrial	Urban fabric	2.1	1999	1	No
4	Street 4	1	600	€200k	Commercial	Urban fabric	5.3	2007	2	Yes

Statistical Maps & Thematic Mapping

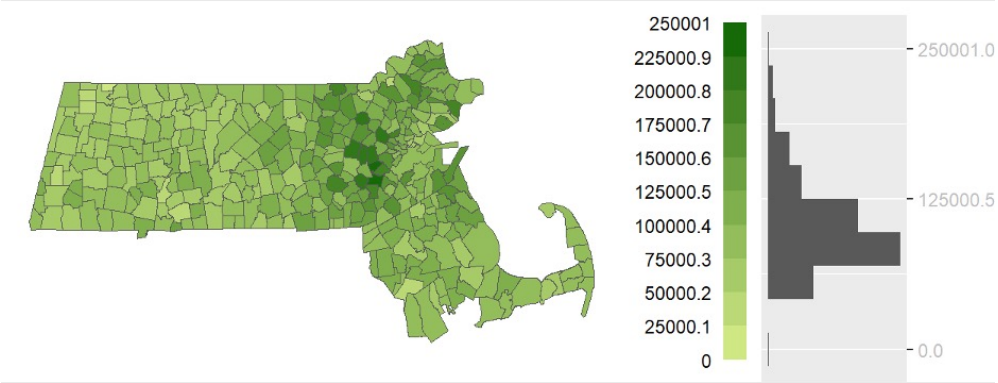
- A **thematic map** is used to show **spatial patterns**
- Data is **aggregated** by predefined spatial units (e.g., countries, states, districts)
- Thematic maps include types such as **choropleth**, **dot density**, **proportional symbol**, and **isopleth maps**
- They display the **spatial distribution** of one or more data themes across selected geographic areas
- Data can be:
  - Quantitative (e.g., population change, income levels)
  - Qualitative (e.g., soil types, land use)
- To create a **vector thematic map**, select the **attribute field** in the map layer that represents the data to be mapped

Distribution Maps



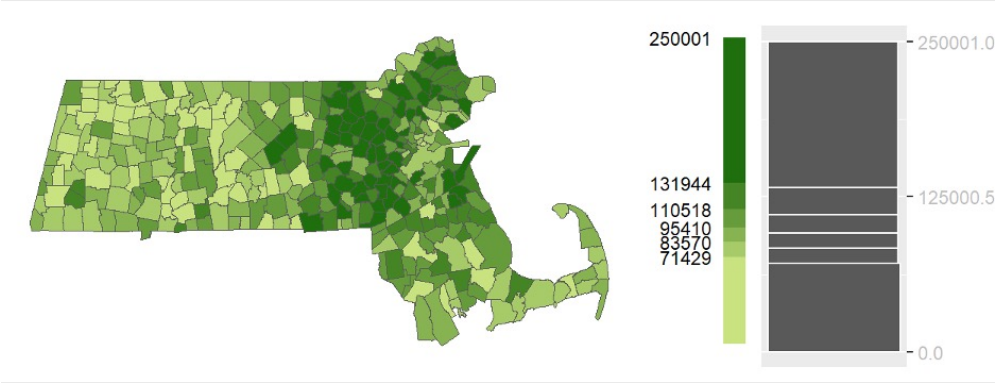
- Example of a continuous color scheme applied to a choropleth map
- Such a map(continuous) may not be as informative as one would like it to be
- In statistics, we seek to reduce large sets of continuous values to discrete entities to help us better “handle” the data

Equal Interval



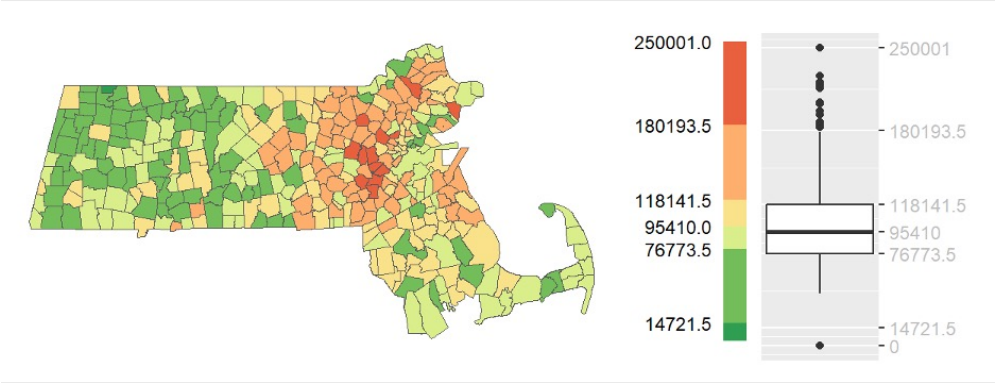
- An equal interval choropleth map using 10 bins
- In the field of statistics, discretization of values can take on the form of a histogram where values are assigned to one of several equal width bins
- A choropleth map classification equivalent is the equal interval classification scheme

Quantile Map



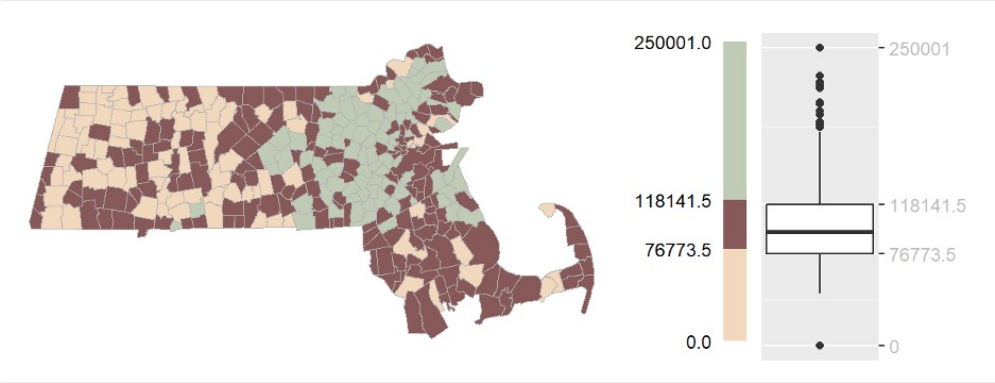
- While an equal interval map benefits from its intuitiveness, it may not be well suited for data that are not uniformly distributed across their range
- Quantiles define ranges of values that have equal number of observations
- Six quantiles with each quantile representing the same number of observations

Boxplot Map



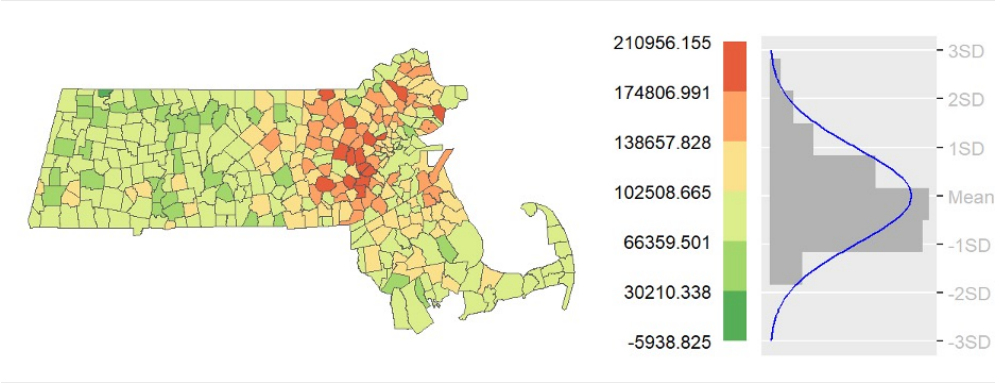
- The discretization of continuous values can also include measures of centrality (e.g., the mean and the median) and measures of spread (e.g., standard deviation units) with the goal of understanding the nature of the distribution such as its shape (e.g., symmetrical, skewed, normal) and range

IQR Map



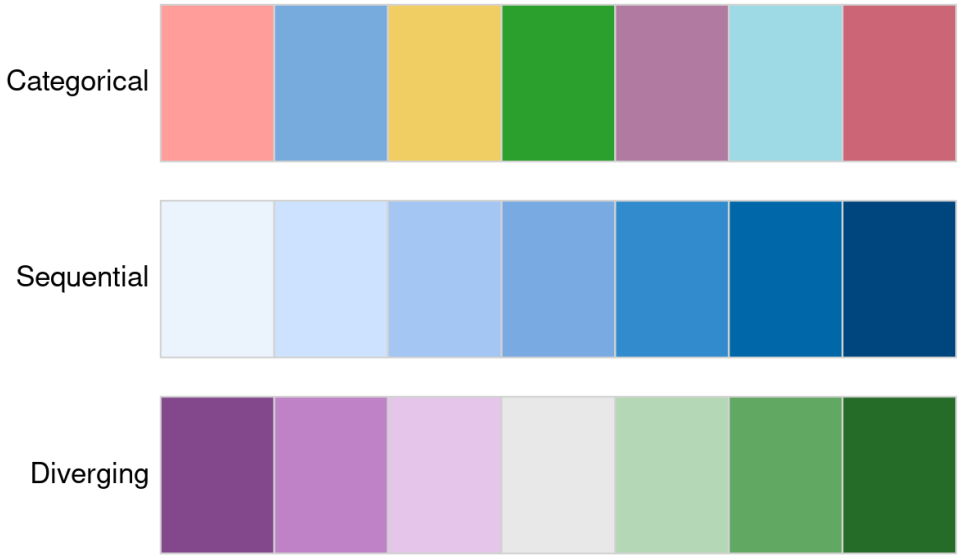
- The IQR map is a reduction of the boxplot map whereby we reduce the classes to just three: the interquartile range (IQR) and the upper and lower extremes

Standard Deviation Map



- Note from the figure that the income data do not follow a normal distribution exactly; they have a slight skew toward higher values
- This results in more polygons being assigned higher class breaks than lower ones

Color Ramps



COLORBREWER - color advice for cartography at https://colorbrewer2.org/

Color Ramps Best Practices

Categorical palettes

- Consist of easily distinguishable colors and are most appropriate for categorical data without any particular order such as state names or land cover classes.
- Colors should be intuitive: rivers should be blue, for example, and pastures green.
- Avoid too many categories: maps with large legends and many colors can be uninterpretable.

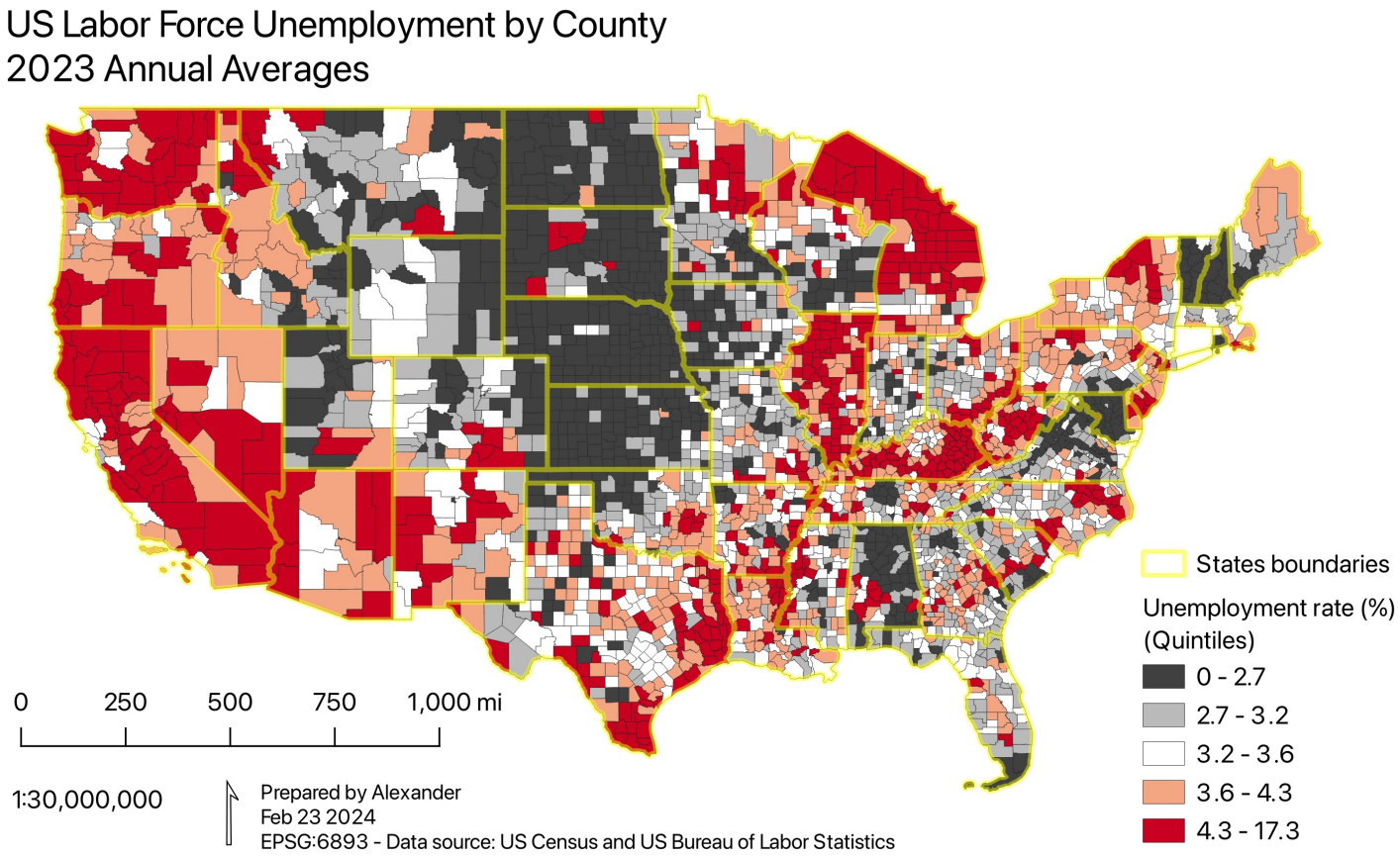
Sequential palettes

- Follow a gradient, for example from light to dark colors (light colors often tend to represent lower values), and are appropriate for continuous (numeric) variables.
- Sequential palettes can be single (greens goes from light to dark green, for example) or multi-color/hue.

Diverging palettes

- Typically range between three distinct colors (purple-white-green) and are usually created by joining two single-color sequential palettes with the darker colors at each end.
- Their main purpose is to visualize the difference from an important reference point, e.g., a certain temperature, the median household income or the mean probability for a drought event.

Color Ramp Example



Vector Attribute Operations 1

Subsetting

- Row filtering with `filter()`
- Column selection with `select()`
- Base `R` vs `tidyverse` approaches

Subsetting

Base R Approach

Using `[ ]` operator:

```
# Subset rows & columns
world[1:5, ]

# Logical subsetting
small = world[
  world$area < 10000, ]
```

tidyverse Approach

Using `dplyr` verbs:

```
# Select columns
select(world, name, pop)

# Filter rows
filter(world,
  area_km2 < 10000)
```

Key Functions

`select()`	choose columns
`filter()`	subset rows by condition
`slice()`	select rows by position
`pull()`	extract single column as vector

Chaining Operations with Pipes (Ref.)

The pipe operator enables expressive, readable code

Without Pipes (Nested)

```
result = slice(
  select(
    filter(world,
      continent=='Asia'),
    name, continent),
  1:5)
```

❌ Hard to read

With Pipes (|> or %>%)

```
result = world |>
  filter(
    continent == 'Asia') |>
  select(name, continent) |>
  slice(1:5)
```

✓ **Clear & readable**

Benefits of Piping

- No intermediate variable names needed
- Easy to add/remove steps
- Works with RStudio keyboard shortcuts
  - `Ctl + Shift + M` on Windows or Linux
  - `Cmd + Shift + M` on MacOS

Vector Attribute Operations 2

Aggregation techniques

- Summarizing data by groups (e.g., all or categories)
- Use of `aggregate()`, `group_by()`, `summarise()`
- Handling spatial attributes in aggregation

Aggregation

Summarize data with grouping variables

Example: **Population by Continent**

```
world_agg = world |>
  group_by(continent) |>
  summarize(
    Pop = sum(pop),
    Area = sum(area_km2),
    N = n()
  ) |>
  mutate(
    Density = Pop / Area
  )
```

Result: New sf object with one row per group + unified geometry

Key Concepts

`group_by()`	Define grouping variable
`summarize()`	Aggregate with functions
`mutate()`	Create new columns
`n()`	Count observations

Vector Attribute Operations 3

Attribute joins

- Joining non-spatial tables to spatial data via **keys**
- Differences between relational joins and spatial joins (spatial joins covered later in spatial operations)

Attribute Joins

Combine tables based on shared key variable

`left_join()`

Keep all rows from first table, add matching data from second

```
world_coffee = left_join(world, coffee_data)
```

`inner_join()`

Keep only rows with matches in both tables

```
world_coffee = inner_join(world, coffee_data)
```

`full_join()`

Keep all rows from both tables, fill NA where no match

```
world_coffee = full_join(world, coffee_data)
```

Vector Attribute Operations 4

Creating / transforming attributes

- Generating new metrics (e.g., density, ratios)
- Use of `mutate()`, `transmute()`
- Renaming attributes with `rename()`, `setNames()`

Vector Attribute Operations 4

Geometry retention and removal

- How operations preserve or drop geometry
- When to use `st_drop_geometry()`
  Or `st_set_geometry(NULL)`

Raster Attribute Operations

Raster attributes basics

- Grid values as attributes
- Spatial meaning of raster cell indices and resolution

Subsetting and extraction

- Extracting cell values from layers
- Working with continuous vs categorical rasters

Global raster operations

- Summary statistics across full raster extents

Raster Data Operations

Raster Data Operations

```
# Numeric raster
elev = rast(
  nrows = 6, ncols = 6,
  xmin = -1.5, xmax = 1.5,
  ymin = -1.5, ymax = 1.5,
  vals = 1:36
)

# Categorical raster
grain = rast(...,
  vals = factor(grain_char))
```

Summary Statistics

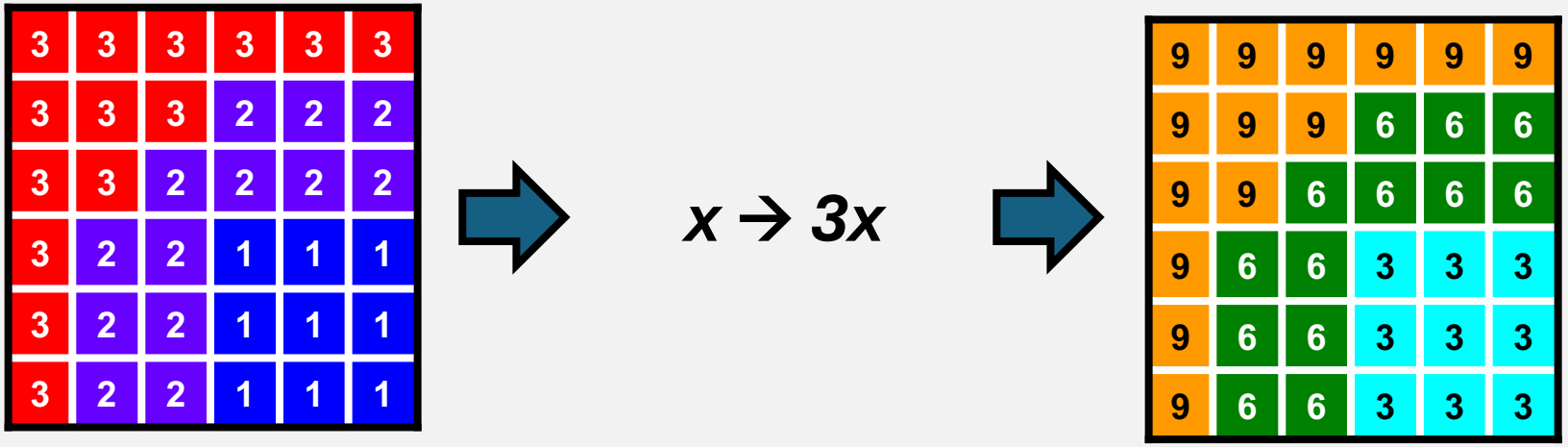
`global()`	Calculate statistics
`freq()`	Frequency table
`summary()`	Quick overview
`hist()`	Visualization

Subsetting Methods

```
# Row-column indexing
elev[1, 1]

# Cell ID
elev[1]

# Extract all values
values(elev)

# Modify values
elev[1, 1] = 0
```

Map Algebra
(More Later in Raster Analysis)

- Cell-by-cell combination of raster data layers
- Each cell has a value
- Simple operations can be applied to values
  - Unary operations apply to one layer
  - Binary operations apply to two data layers
  - More complex operations apply to many layers



Pre-Processing Workflows

Establishing a reproducible workflow

- Scripted workflows vs ad-hoc steps
- Importance of documentation and version control

Typical pre-processing steps

- Data import and initial inspection
- Attribute cleaning (e.g., selecting relevant fields, renaming)
- Coordinate system standardization
- Handling missing or inconsistent values

Spatial ETL Workflow

Extract → Transform → Load pipeline for spatial data

1. **Extract**
   - Read spatial data (`sf`, `terra`)
   - Import attributes (from CSV/DB)
   - Web APIs & services
2. **Transform**
   - Filter & subset
   - Aggregate & summarize
   - Join & merge
   - Create new attributes
3. **Load**
   - Write to file formats
   - Export to database
   - Publish to web services

Best Practices

• Use pipes for readable workflows • Preserve geometries with sf operations
• Drop geometry when speed matters • Document transformations

Practical Example: Coffee Production

Complete workflow from data join to visualization

```
# 1. Load data
library(sf)
library(dplyr)
library(spData)

# 2. Join spatial + attribute data
world_coffee = left_join(world, coffee_data,
                         by = 'name_long')

# 3. Filter & aggregate
top_producers = world_coffee |>
  filter(!is.na(coffee_production_2017)) |>
  select(name_long, coffee_production_2017) |>
  arrange(desc(coffee_production_2017)) |>
  slice(1:10)

# 4. Visualize
plot(world_coffee['coffee_production_2017'])
```

Geometry column preserved throughout!
`sf` objects behave like **`data.frames`**

Best Practices

Efficiency considerations

- When to drop geometry for speed
- `tidyverse` vs base R performance tradeoffs

Data integrity and accuracy

- Validate joins and aggregations
- Check spatial and attribute consistency

Reproducibility standards

- Use of R Markdown, notebooks, and code containers(Docker)

Common Pitfalls

Losing geometry column

Solution: Use `dplyr` verbs or `st_drop_geometry()` explicitly

Avoid: `world$name` **Use**: `pull(world, name)`

Mismatched join keys

Solution: Transform to same coordinate system before operations

```
left_join(world, data, by = join_by(name == country))
```

Mismatched join keys

Solution: Transform to same coordinate system before operations

```
left_join(world, data, by = join_by(name == country))
```

Mismatched join keys

Solution: Transform to same coordinate system before operations

```
st_transform(data, crs = st_crs(world))
```

Questions to Practice

1. Can you explain **what attribute transformation is** and give **an example** of when you would **create a new feature** from existing data?
2. How do you typically handle **missing or inconsistent values** during the data pre-processing stage?
3. Why is **data normalization or standardization** important, and in what situations would you apply it?
4. Can you describe a **typical data analysis workflow**, from receiving raw data to producing final insights?

Assignment #2 – Objective

Workflow in **R** (or **Python**)

- Load spatial data (**`sf`**, **`terra`**)
- Subset and clean attributes
- Join external attribute table to spatial data
- Compute summary statistics
- Export results for visualization or modeling

Assignment #2 – Background

- Land parcels are boundaries that have associated information such as property owner, land use, value, and location attributes
- Improvement value refers to the assessed monetary value attributed not to the land itself but to the physical improvements on the land (typically buildings and other structures). In the context of property assessment, this is often used as a proxy for the built environment's contribution to total property worth (e.g., houses, garages, renovations).

Follows

1. Data Acquisition
2. Data Preparation
3. Aggregation and Analysis
4. Integrate Census Data
5. Visualizations
6. Documentation and Reporting

Assignment #2 – Part 1

1. Data Acquisition

Parcel Data

- Visit the **Texas Natural Resources Information System** (TNRIS) portal https://tnris.org/stratmap/land-parcels.html
- Select parcel data for 5 neighboring counties of your choice
- Download the 2025 parcel datasets for each county and unzip in a folder (a local folder with raw parcel shapefiles files for the counties)

Assignment #2 – Part 2

2. Data Preparation

Using **R** (with **`sf`**, **`dplyr`**, and **`tidyr`**)

a. Read the parcel files, standardize and inspect
   - Combine all counties into a single sf object
   - If needed, ensure projections(and fields) are consistent across counties
   - Inspect **`STAT_LAND_` and `IMP_VALUE`**

b. Filter for single-family houses
   - Define single family residential by the parcel state code: **`STAT_LAND_ == "A"`**

Assignment #2 – Hints 1

```
library(sf)
target_crs <- 4326  # WGS84, change as needed/appropriated

parcels <- list.files(
  path = "/Users/abuabara/Downloads/DAEN489_data/parcels",
  pattern = "\\.shp$",
  full.names = TRUE,
  recursive = TRUE
)

parcels_all <- do.call(
  rbind,
  lapply(
    parcels,
    function(f) {
      x <- st_read(f, quiet = TRUE)

      if (is.na(st_crs(x))) {
        st_crs(x) <- target_crs
      }

      x <- st_transform(x, target_crs)
      x$source_file <- basename(f)
      x
    }
  )
)

parcels_all$STAT_LAND_2 <- substr(parcels_all$STAT_LAND_, 1, 1)
```

Assignment #2 – Part 3

3. Aggregation and Analysis

a. Aggregate Improvement Value

- For each county, calculate:
  - The total improvement value of single-family parcels
  - Parcel count

Assignment #2 – Hints 2

```
library(dplyr)

agg_parcels <- parcels_all %>%
  filter(STAT_LAND_ == "A") %>%
  st_set_geometry(NULL) %>%
  group_by(COUNTY) %>%
  summarise(GEOID = unique(FIPS),
            total_imp_value = sum(IMP_VALUE, na.rm = TRUE),
            parcel_count = n())

library(tidyr)

parcels_all %>%
  st_drop_geometry() %>%
  filter(!is.na(IMP_VALUE) & IMP_VALUE > 0) %>%
  group_by(COUNTY, STAT_LAND_2) %>%
  summarise(n = n(),
            IMP_VALUE = sum(IMP_VALUE, na.rm = TRUE),
            .groups = 'drop') %>%
  arrange(desc(n)) %>%
  pivot_wider(names_from = STAT_LAND_2,
              values_from = n,
              values_fill = 0)
```

Assignment #2 – Part 4

4. Integrate Census Data

a. Acquire Census TIGER/Line Data

- Use the **`tidycensus`** packages to download:
- County boundaries
- Population estimates (e.g., ACS 5-year estimate for total population)

b. Merge with Parcel Aggregates

- Join population data by county GEOID or name

c. Compute Per Capita Metrics

- Average improvement value per capita
- Average population per single-family parcel

Assignment #2 – Hints 3

```
library(tidycensus)

census_api_key("YOUR API GOES HERE")

pop_data <- get_acs(geography = "county",
                    # county = "Brazos",
                    county = unique(agg_parcels$COUNTY),
                    output = "wide",
                    geometry = FALSE,
                    variables = c(pop = "B01001_001"),
                    state = "TX",
                    year = 2023)

agg_parcels <- agg_parcels %>%
            left_join(pop_data, by = "GEOID") %>%
            mutate(
              imp_value_per_capita = total_imp_value / popE
            )
```

Assignment #2 – Part 5

5. Visualization

a. Basic thematic maps

- Use `tmap` (or `ggplot2`) to produce:
  - Basic choropleth map of total improvement value by county
  - Basic choropleth map of average improvement value per inhabitant

b. You may also consider including supporting plots, such as:

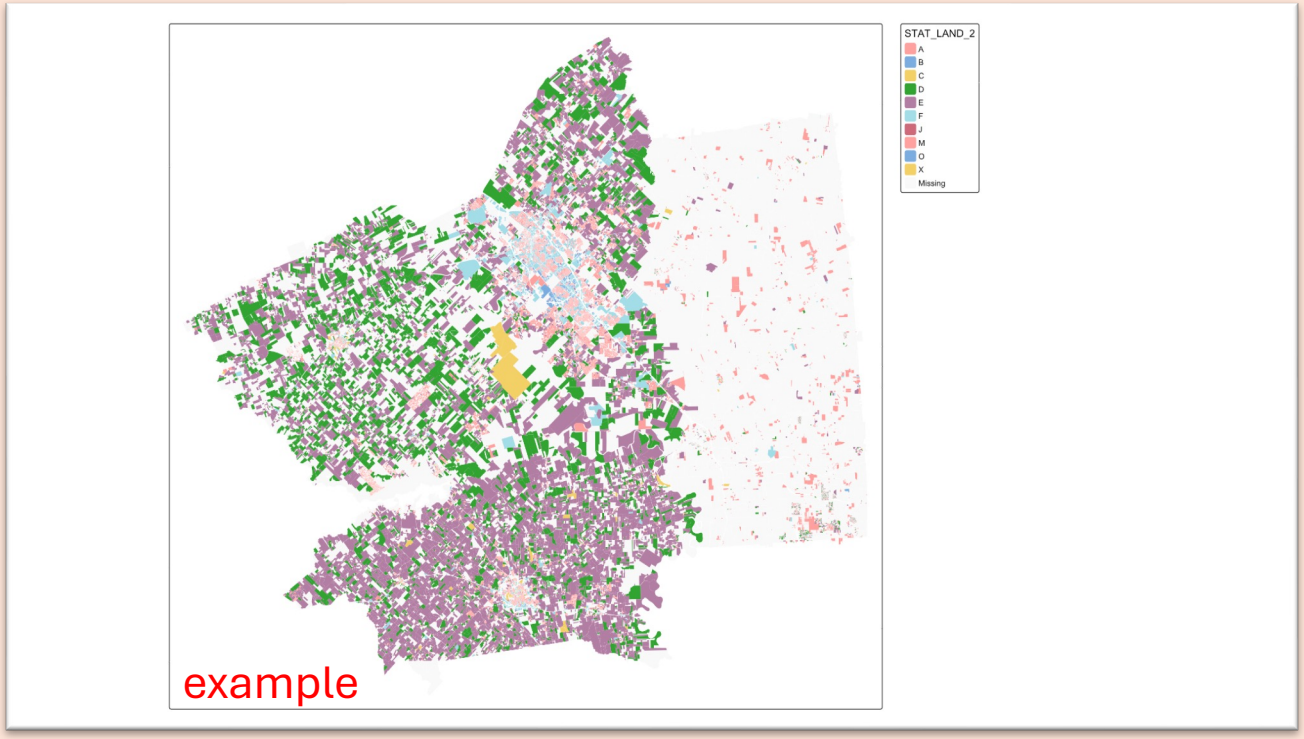
- Bar charts or scatterplots showing:
  - Population vs. improvement value
  - Parcel count vs. per capita value

c. Interpretation

- Comment on spatial patterns
  (e.g., regional real estate markets, population density relationships)

Assignment #2 – Hints 4

```
library(tmap)

tm_shape(parcels_all) +
    tm_polygons(fill = "STAT_LAND_2", lwd = 0)
```



Assignment #2 – Part 6

6. Documentation and Reporting

a. Reproducible script or Jupyter notebook

- Clean, commented (as possible)
- Reproducible from data download to map output

b. Include

- Data sources and preprocessing steps
- Results and interpretations
- Limitations and potential extensions (e.g., overlay zoning, flood risk)

c. Visuals

- Please include / submit maps and charts with captions

Appendix

Geographic Data & Spatial ETL with Python

Attribute Operations 1

Libraries and Data

```
# Libraries
import numpy as np
import pandas as pd
import geopandas as gpd
import rasterio
import rasterio.plot
import matplotlib.pyplot as plt

# Data
world = gpd.read_file('/Users/abuabara/…/DAEN489/pydata/world.gpkg')
src_elev = rasterio.open('/Users/abuabara/…/DAEN489/pydata/elev.tif')
src_grain = rasterio.open('/Users/abuabara/…/DAEN489/pydata/grain.tif')
src_multi_rast = rasterio.open('/Users/abuabara/…/DAEN489/pydata/landsat.tif')
```

Attribute Operations 2

Vector Attribute Subsetting

Subset Countries with Small Area

```
world.head()

idx_small = world['area_km2'] < 10000
small_countries = world[idx_small]
small_countries
```

Logic to Combine Conditions

```
asia_small = world[
    (world['continent'] == 'Asia') &
    (world['area_km2'] < 10000)
]
asia_small[['name_long', 'continent', 'area_km2']]
```

Select Specific Columns & Rows

```
asia_subset = world[world['continent'] == 'Asia'] \
    .loc[:, ['name_long', 'continent']] \
    .iloc[0:5, :]
asia_subset
```

Attribute Operations 3

Aggregation With Attributes

Sum Population by Continent (Attribute Only)

```
world_pop = world.groupby('continent')[['pop']].sum().reset_index()
world_pop
```

Spatial Aggregation with Geometry

```
world_agg = world[['continent','pop','geometry']] \
    .dissolve(by='continent', aggfunc='sum') \
    .reset_index()
world_agg.plot(column='pop', legend=True)
```

Joining Attributes Between Datasets

Attribute Join

```
coffee_data.head()

world_coffee = pd.merge(
    world, coffee_data,
    on='name_long', how='left'
)
world_coffee[['name_long', 'coffee_production_2016']]
```

Join Variation — Inner Join

```
coffee_inner = pd.merge(
    world, coffee_data,
    on='name_long', how='inner'
)
coffee_inner
```

Attribute Operations 4

Creating & Transforming Attributes

Population Density

```
world2 = world.copy()
world2['pop_density'] = world2['pop'] / world2['area_km2']
world2[['name_long','pop_density']]
```

Drop Geometry for Pure Attribute Table

```
world2 = world2.drop('geometry', axis=1)
world2 = pd.DataFrame(world2)
world2.head()
```

Attribute Operations 5

Read Raster as NumPy Array

Read Raster as NumPy Array

```
plt.close('all')
rasterio.plot.show(elev)

elev = src_elev.read(1)
elev
```

Raster Summaries

```
np.mean(elev)

elev_float = elev.astype('float64')
elev_float[0,2] = np.nan
np.nanmean(elev_float)
```

Raster Categorical Frequency

```
plt.close('all')
rasterio.plot.show(grain)

grain = src_grain.read(1)
freq = np.unique(grain, return_counts=True)
plt.bar(*freq)
plt.show()
```