Collaborators: None

Project Overview

- **Goal**: How can we find natural groups among subsidized housing based on their key features?
- **Dataset**: Subsidized Housing - Six Metro Areas - 2017, 347 KB (1,945 lines)

Data Processing

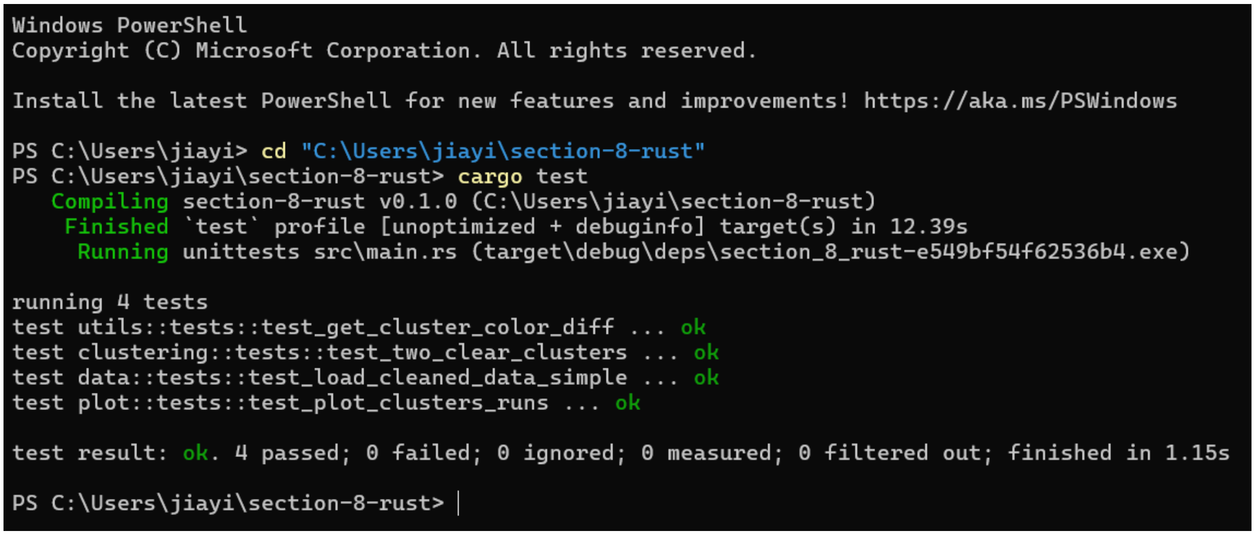
- **Loading**: The function *load_cleaned_data()* opens the CSV via *csv::Reader* and *serde::Deserialize*
    - Skips invalid rows (logging errors) and returns a *Vec<HousingProperty>* with fields *total_units: u32*, *subsidy_count: u32*, and *owner_type: String*
- **Cleaning/transformations**: Removed rows with missing entries (and referenced the CSV in SPSS to double-check), selected only the variables Latitude, Longitude, TotalUnits, ActiveSubs, and OwnerType (but finalized with the last three), and wrote out to "cleaned_subsidized_housing.csv" from "data_cleaning.py"

Code Structure

- **Modules**
    - "data.rs" loads and deserializes the cleaned CSV into structs
    - "clustering.rs" converts structs into a scaled 2D dataset, runs k-means (k = 4), denormalizes centroids, and prints summary statistics
    - "plot.rs" creates a 1600x1200 PNG scatter plot of TotalUnits and ActiveSubs, colored by cluster, with a legend and 'x' markers on each centroid
    - "utils.rs" provides a function used to match cluster IDs to colors consistently
    - "main.rs" runs all of the above where the project loads, clusters, and plots
- **Key functions & types**
    - *struct HousingProperty* ("data.rs") → represents one housing record with fields *total_units, subsidy_count*, *owner_type*
    - *load_cleaned_data(path)* ("") → inputs file path to cleaned CSV and outputs *Vec<HousingProperty>* or error; it opens a file, iterates, and pushes valid entries
    - *to_ndarray_with_scales(properties)* ("clustering.rs") → inputs slice of *HousingProperty* and outputs *(Array2<f64>, min[0, 1], range[0, 1])*; it fills raw values, computes the minimum and maximum per column, and applies a normalization formula
    - *cluster_properties(properties, k)* ("") → inputs data slice and cluster count and outputs *Vec<usize>* of labels; it calls the scaler, fits k-means, predicts labels,

    and computes denormalized centroids, sizes, and owner-type distributions before printing
    - *plot_clusters(properties, labels)* (“plot.rs”) → inputs slice of properties and labels and outputs *()* or error; it automatically increments a new filename, builds *BitMapBackend*, draws points by cluster with legend entries, computes centroids and overlays markers, renders a legend, and saves
    - *get_cluster_color(cluster_id)* (“utils.rs”) → inputs cluster index and outputs *RBGColor* for plotting
- **Main workflow**
    - *main.rs* calls *load_cleaned_data* → *cluster_properties* → *plot_clusters*
    - Each stage uses only the public functions above, keeps data in memory, and communicates via return values and printed logs

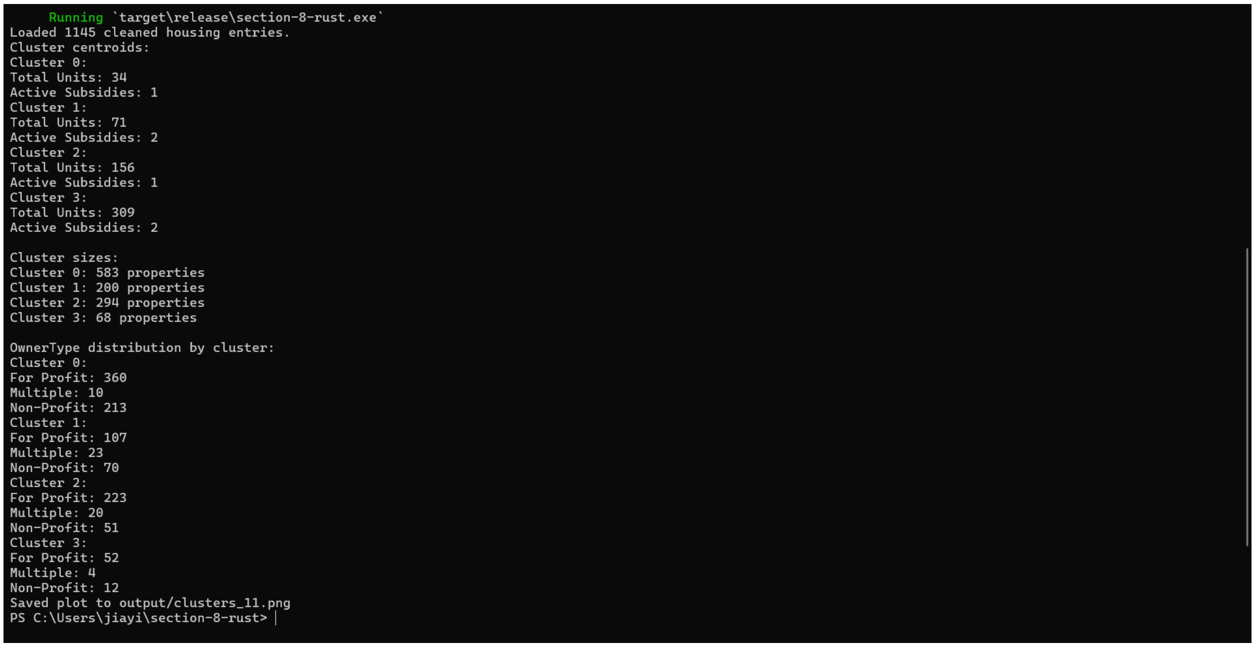
Tests



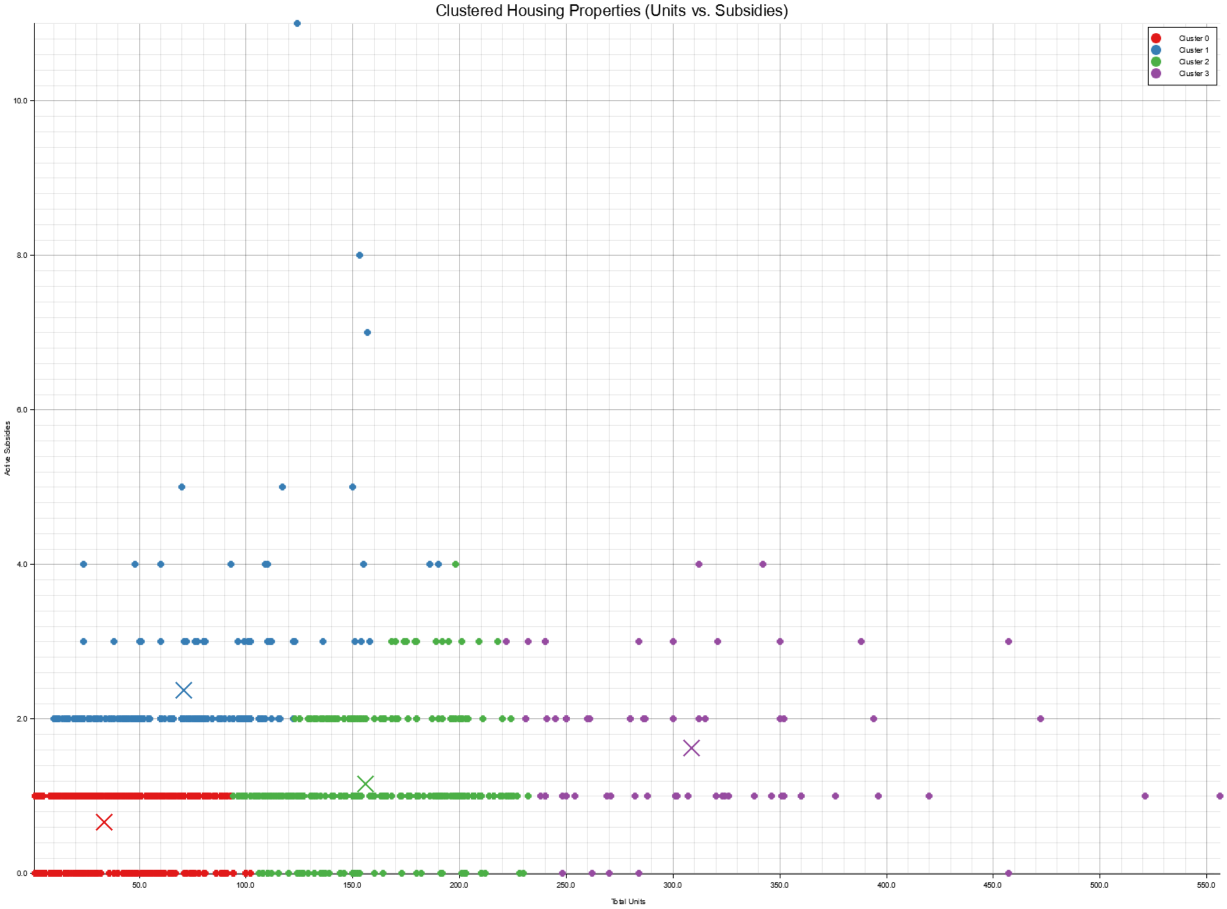
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\jiayi> cd "C:\Users\jiayi\section-8-rust"
PS C:\Users\jiayi\section-8-rust> cargo test
   Compiling section-8-rust v0.1.0 (C:\Users\jiayi\section-8-rust)
    Finished `test` profile [unoptimized + debuginfo] target(s) in 12.39s
     Running unittests src\main.rs (target\debug\deps\section_8_rust-e549bf54f62536b4.exe)

running 4 tests
test utils::tests::test_get_cluster_color_diff ... ok
test clustering::tests::test_two_clear_clusters ... ok
test data::tests::test_load_cleaned_data_simple ... ok
test plot::tests::test_plot_clusters_runs ... ok

test result: ok. 4 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 1.15s

PS C:\Users\jiayi\section-8-rust> |
```

- For “**utils.rs**”: *test_get_cluster_color_diff()* checks whether different cluster IDs map to different colors on the RGB scale
- For “**clustering.rs**”: *test_two_clear_clusters()* checks if *cluster_properties* groups clearly distinct sample points into exactly two clusters
- For “**data.rs**”: *test_load_cleaned_data_simple()* checks that CSV parsing works on a basic example as a behavior scan
- For “**plot.rs**”: *test_plot_clusters_runs()* checks for *plot_clusters* execution doesn’t have errors and writes a PNG output file

Results



```
     Running `target\release\section-8-rust.exe`
Loaded 1145 cleaned housing entries.
Cluster centroids:
Cluster 0:
Total Units: 34
Active Subsidies: 1
Cluster 1:
Total Units: 71
Active Subsidies: 2
Cluster 2:
Total Units: 156
Active Subsidies: 1
Cluster 3:
Total Units: 309
Active Subsidies: 2

Cluster sizes:
Cluster 0: 583 properties
Cluster 1: 200 properties
Cluster 2: 294 properties
Cluster 3: 68 properties

OwnerType distribution by cluster:
Cluster 0:
For Profit: 360
Multiple: 10
Non-Profit: 213
Cluster 1:
For Profit: 107
Multiple: 23
Non-Profit: 70
Cluster 2:
For Profit: 223
Multiple: 20
Non-Profit: 51
Cluster 3:
For Profit: 52
Multiple: 4
Non-Profit: 12
Saved plot to output/clusters_11.png
PS C:\Users\jiayi\section-8-rust> |
```



- **Interpretation**

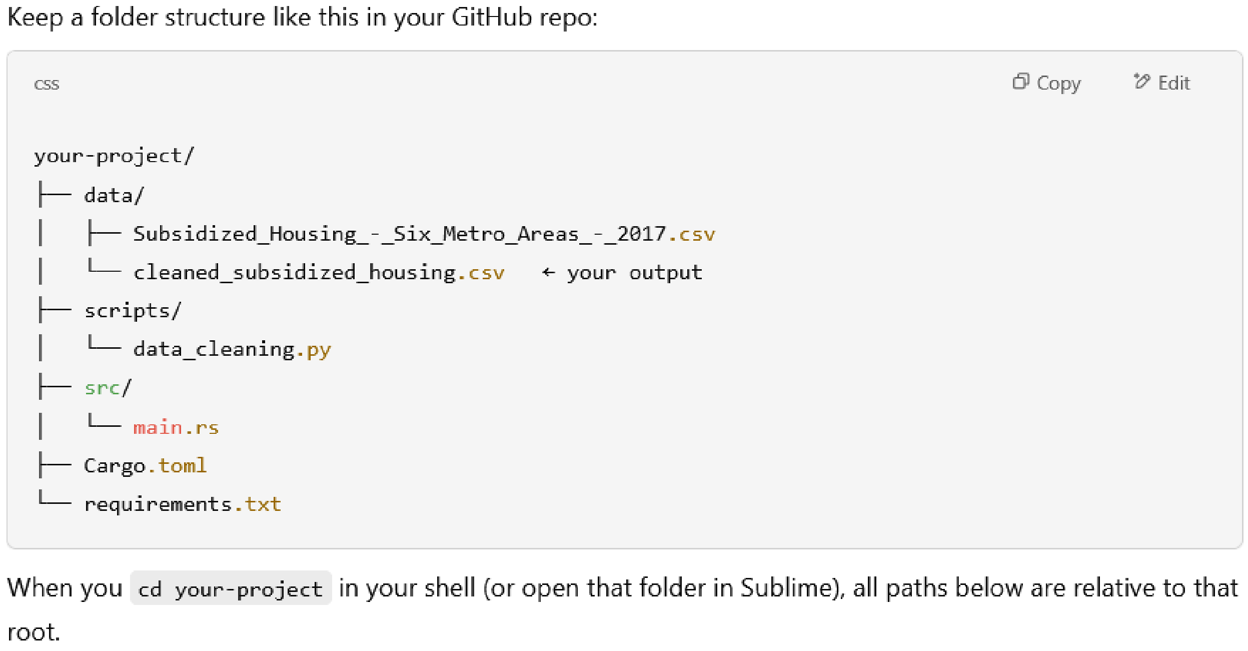
- Cluster 0 grouped a large number of small, least subsidized buildings where for-profits operate a majority and nonprofits fill in the rest; this suggests a widespread baseline assistance across both sectors
- Cluster 1 grouped mid-sized developments with slightly more aid that are controlled mostly by for-profit companies; may suggest that gentrification is increasing
- Cluster 2 grouped larger buildings receiving minimal aid where for-profits hold a three-quarters supermajority ownership; this suggests more housing scarcity and inequity over the distribution of resources
- Cluster 3 grouped a small amount of the largest buildings with higher total subsidies, also controlled primarily by for-profits; may suggest housing displacement if the owners choose profit over people

Usage Instructions

1. **Clone**
    a. git clone https://github.com/<username>/section-8-rust.git
    b. cd section-8-rust
2. **Build/Test**
    a. cargo test
    b. cargo build
3. **Run**
    a. cargo run --release
4. **Runtime**
    a. Data loading + clustering + plotting should finish in about 3-5 seconds

AI-Assistance Disclosure and Other Citations

- I used ChatGPT to help me navigate file I/O when writing in Python (using Pandas) to clean my dataset. I am used to using absolute paths, but since the context of this assignment was to build a publicly accessible project, that was not viable. For this, I asked the following: “How do I use a relative path? In the grand scheme of things, I need my entire project to be accessible through GitHub (I want to do a push-pull request to update the work I've done today) so I don't think using absolute paths is convenient.” It then produced this:



Keep a folder structure like this in your GitHub repo:

```
your-project/
├─ data/
│   ├─ Subsidized_Housing_-_Six_Metro_Areas_-_2017.csv
│   └─ cleaned_subsidized_housing.csv   ← your output
├─ scripts/
│   └─ data_cleaning.py
├─ src/
│   └─ main.rs
├─ Cargo.toml
└─ requirements.txt
```

When you `cd your-project` in your shell (or open that folder in Sublime), all paths below are relative to that root.

- Among other things, ChatGPT helped me debug cases where my “NaN” filters were having errors and helped me track input and output locations (very useful since I started from C:\Users\jiayi and was not doing it in my dedicated DS 210 folder)
- Found this blog post to be particularly useful when thinking of how to visualize k-means clustering with my specific dataset, since my proposal was rejected for lack of graphability (and this was done in R, so no, I did not use any of their code)