Chiffordable

1. Data Documentation

This project aims to show the affordability of housing in the Chicago area. For that, it takes three data sources (Zillow, CMAP and Livability Index) to visualize where families can afford rental housing within a specified income and a percentage of their income they are willing to spend on rent.

1.1. Data Sources

Zillow - Marketplace for Housing	**Community Data Snapshots 2024 - Chicago Metropolitan Agency for Planning (CMAP**	**AARP Livability Index - American Association of Retired Persons (AARP)**
An online real estate marketplace that provides information on properties for sale, rent, and mortgage financing.	Project that compiles demographic, housing, employment, land use, and other data for northeastern Illinois. The primary source is the U.S. Census Bureau's 2022 American Community Survey.	Index that scores neighborhoods based on seven key categories using over 50 unique data sources. This data was obtained through web scraping and is used to provide demographic characteristics for neighborhoods in rental areas.
Data Used: - Property price - Size (square footage), - Location (ZIP code), - Number of bedrooms and bathrooms	**Data Used:** - Population statistics - Employment and housing data - Land use information	**Data Used:** - Housing (Affordability housing costs) - Neighborhood (Characteristics of the community) - Transportation (Accessibility and mobility) - Environment (Air quality and environmental factors) - Health (Access to healthcare and wellness services) - Engagement (Community involvement opportunities) - Opportunities (Employment and educational options)
Challenges: - Some listings may be outdated - Data gaps in less active market areas - Multiple scrapings needed for each zip code and listing	**Challenges:** - Only polygon to be merged with listings - Data of age and race distribution is in thousands of people and not in percentage.	**Challenges:** - The website did not provide scrapeable data for all ZIP codes included in our analysis. As a result, we manually collected missing information for specific ZIP codes and integrated it directly into the code before generating the final CSV file.

1.2. Data Flow in the Project

1. Data Extraction (Extracting Module):
    - CMAP: Data is retrieved via an API in JSON format, providing demographic and economic insights.
    - AARP: Web scraping extracts livability scores, and missing ZIP codes are manually supplemented.
    - Zillow: Web scraping is used to collect rental listings, including price, size, and location.

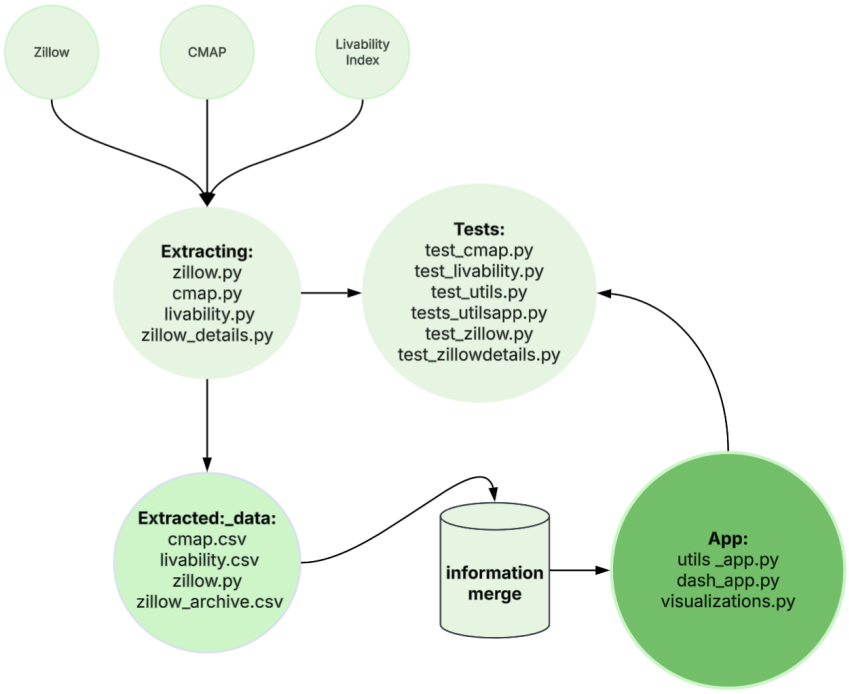
2. Data Processing (Extracted Data Module):
    - CMAP data is processed to compute age and racial distributions as percentages.
    - Zillow listings are cleaned, ensuring accurate price formatting and geolocation data.
    - Livability scores from AARP are extracted and structured.
    - All datasets are standardized for integration.

3. Data Integration (Merging):
    - Zillow, CMAP, and AARP datasets are merged using ZIP codes as the primary key.
    - Missing values in livability scores are manually supplemented.

4. Testing (Tests Module):
    - Unit tests ensure data integrity, verifying total population percentages and livability score accuracy.
    - Mock responses simulate API calls and web scraping to validate extraction functions.
    - CSV outputs are checked to confirm proper formatting and completeness.

5. Visualization & Interaction (App Module)
    - An interactive Dash web application is created to explore affordable housing options.
    - Users input their annual income and rent percentage, filtering affordable listings dynamically.
    - Maps display rental listings, demographic distributions, and livability scores.
    - Clicking on a community reveals detailed insights, including rent prices, racial composition, and livability index scores.

6. Storage & Export (Extracted Data Module):
    - The cleaned and merged data is stored as a CSV file for further analysis.
    - The structured dataset is used in the web dashboard to generate visual insights.



2. Project Structure

2.1. extracting:

The extracting folder contains all the necessary code for retrieving data from the three sources of the project. Each submodule is named after its respective data source and is responsible for handling data extraction, cleaning, and processing.

Zillow.py	Zillow_details.py	livability.py	cmap.py
Scrapes rental property data from Zillow such as price, size, location, and the number of bedrooms and bathrooms for listings in various ZIP codes across Chicago	supplements the main zillow.py scraper by extracting specific details for individual apartment units within multi-unit buildings	Responsible for retrieving, and structuring liability scores for each zip-code analyzed for the Chicago Area	Responsible for retrieving, cleaning, and structuring demographic and housing data from the CMAP 2024 dataset

2.2. extracted_data:

Contains 4 csv files with cleaned data to merge and to create the visualization. These files are: Livability.csv, Zillow.csv, cmap.csv.

2.3. app:

Chiffordable

This app module is responsible for a web application, designed to help users explore affordable housing options in Chicago based on their income and rent preferences. The module processes and formats datasets, converting community boundaries into geometric objects for spatial analysis. The main feature is an interactive map that visualizes affordable housing options and community-level characteristics. Users can interact with the map by selecting locations to view community demographics and rental details. The application runs on a local server and launches in a web browser upon execution. Below is a breakdown of each module and its function.

Utils_app.py	dash_app.py	visualizations.py
provides data visualization features and helper functions for processing spatial data, data reconciliation, and calculating affordability for rental listings	sets up an interactive web application using Dash, allowing users to explore housing affordability in Chicago. It includes data visualizations, interactive maps, and a tab-based navigation system.	Contains functions to generate interactive maps and charts that display affordable housing listings, community demographics, and livability scores based on user-defined rent preferences.

2.4. Tests

A comprehensive suite of unit tests to validate data extraction, processing, and visualization functions. The tests ensure data integrity, correct API responses, and proper merging of datasets.

Test_cmap.py	test_livability.py	test_utils.py	test_zillow.py	test_zillow_details.py	test_utilsapp.py
Verify that the processed demographic data is correctly normalized, ensuring that the sum of age group percentages and racial composition percentages equals 100%.	Ensure that the livability index data is correctly retrieved, processed, and stored by validating API responses, checking data extraction accuracy, and handling missing or invalid ZIP codes.	Verify the correctness of helper functions used across the project, including URL completion, JSON parsing, and CSV file saving, ensuring data integrity and proper functionality.	validate the web scraping process by ensuring correct extraction of property listings, cleaning and formatting of listing details, pagination handling, and price parsing to guarantee accurate rental data collection.	verify the extraction of detailed property information, ensuring accurate retrieval of unit-level data such as prices, bedrooms, bathrooms, and square footage from Zillow listing pages, even when stored under different JSON structures.	Verify the data reconciliation component of the data visualization app, ensuring the correct data is fetched and visualized, based on income, share on rent, community and listings selected.

3. Team Responsibilities

Member	Lead	Contributions
Daniela Ayala	Zillow: complete process of scraping listings including extracting utils and correspondent test functions	Data visualizations, documentation
José Manuel Cardona	Data reconciliation, data visualizations, interactive app and App utils tests	AARP scraping, documentation
Agustín Eyzaguirre	CMAP: API call, cleaning, processing and tests associated with it.	AARP manual extraction, Documentation
María José Reyes	AARP: scraping, manual extraction and tests associated with it. Video.	Documentation

4. Final thoughts

Our main purpose was to raise awareness about the enormous shortage of affordable housing in the city of Chicago, especially for extremely low-income renter households. To achieve this, we developed a tool that displays communities and a selection of available housing options in the market based on an annual income and the proportion of that income allocated to rent.

Given the background of most people in our program, it's *fun* to experiment with expected salaries after completing the MS. However, when one inputs an income of $33,100 (extremely low income for households) or $23,200 for a single individual, allocating 30% of it to rent, the number of communities and available listings reduces dramatically to just a few places. This allows users to understand the severity of the affordable housing shortage in Chicago.

With this information, we firmly believe that authorities and policymakers in Illinois and the City of Chicago must address this issue to prevent a housing crisis in the coming years and the numerous problems that may arise from it.

Appendix I

Functions breakdown for each module

A. Extracting Module

File name	Functions
Zillow.py	• extract_listings(json_data): Extracts property listings from Zillow's JSON response. • get_listing_info(listing): Parses each property listing and structures it into a dictionary with details. • nextpage_from_xpath(response_text): Identifies the URL for the next page of search results using XPath. • one_zipcode_scrape(url, max_pages=20): Scrapes all available rental listings for a given ZIP code, this function calls Zillow details.py when the listing has multiple apartments and makes sure each one is scraped and listed. • main(zip_codes): The main function that iterates through a list of Chicago ZIP codes, scraping rental listings from Zillow for each.
Zillow_details.py	• extract_details(json_data): Extracts individual apartment unit details from Zillow's JSON response. Listings can be found under either the "floorPlans" or "ungroupedUnits" fields. • get_details_info(listings): Parses each unit's information (e.g., price, number of beds and baths, square footage) and formats it into a clean dictionary. • combine_details(listing, details): Merges general property listing details (e.g., address, coordinates) with unit-specific details (e.g., unit price, bedrooms, bathrooms). • get_prices(url): Fetches and parses detailed apartment information from an individual Zillow listing page. • get_details(listing): Calls get_prices(url) and processes unit-level data before returning a structured list.
livability.py	• complete_link(zip_code: str) -> str: Generates the full URL to fetch livability index data for a given ZIP code. • make_request(zip_code: str): Sends an HTTP request to retrieve the webpage for a given ZIP code. • complete_table_scores_link(zip_code: str) -> str: Generates the full API URL to retrieve livability index scores in JSON format. • make_table_request(zip_code: str): Sends a request to the AARP API to retrieve livability index scores for a given ZIP code and returns the scores as a dictionary. • extract_next_chars(text: str, categories: list) -> dict: Uses regex to extract score values for different categories from the raw API response. Data Processing Functions • livindex_by_zc(chicago_zip_codes: list) -> list: Calls make_table_request for each ZIP code in the list and stores the retrieved scores in a structured format. • csv_livability(list_by_zip, filename, save_path="../extracted_data"): Saves the collected ZIP code scores into a CSV file.

cmap.py	• request_url(url: str) -> dict: Sends an HTTP GET request to retrieve JSON data from the CMAP API.
	Data Processing and Cleaning
	• Extracting Headers and Features, defines headers that categorize the CMAP data
	• Parsing the JSON Dats
	Mapping ZIP Codes to Communities
	• zip_comms_features Dictionary
	  ○ Creates a mapping between ZIP codes and Chicago community areas using spatial intersection.
	  ○ Uses Shapely's geometry operations (intersects, contains, Point) to find ZIP codes that overlap with community areas.
	  ○ Stores the intersection relationships in a structured dictionary

Utils.py	• complete_link(zip_code: str) -> str: Generates the full URL if url is incomplete (starts with '/') • fetch_page(url: str, headers_input={}) -> str: Fetches the content of a webpage given a URL using httpx. Raises: httpx.HTTPStatusError: If the response status code is not 2xx • parse_script_content(html: str): Parses the HTML content and extracts the script tag with the '__NEXT_DATA__'. • save_to_csv(listings: list, filename, file_cols, save_path="../extracted_data"): Saves listings to a CSV from extracting to extracted directory and eliminates duplicates.

B. APP Module:

File name	Functions
Utils_app.py	• gdf_to_geojson(gdf): Converts a GeoDataFrame into a GeoJSON format for use in a choropleth map. • get_community_from_point(gdf_dataframe, lat, lon): Finds the community details based on a given latitude and longitude. • get_community_from_name(gdf_dataframe, name): Obtains the community details based on name • get_livability_scores(dataframe, zip_code): Based on a zip code, returns the livability scores • calculate_rent(annual_income, share_on_rent): Computes the maximum amount the user is willing to spend on rent
dash_app.py	• update_tab(tab_name): Updates the displayed page based on the selected tab. • update_map(annual_income, share_on_rent): Generates an affordability map based on user input. • display_info(clickData): Shows community details, demographics, and livability scores when a user clicks on a location. • main(): Initializes and runs the Dash web application.

visualizations.py	• create_combined_figure(annual_income, share_on_rent): Generates an interactive affordability map based on income and rent preference. • age_figure(dataframe): Creates a bar chart displaying the age distribution of a selected community. • race_figure(dataframe): Generates a bar chart illustrating the racial composition of a selected community. • livability_figure(dataframe): Produces a horizontal bar chart showcasing livability scores for a zip code

C. Tests

File name	Functions
Test_cmap.py	• test_age_sums_100(): Confirms that the sum of all age group percentages in each community adds up to 100% (with a margin of error of 0.25% due to decimals). • test_race_sums_100(): Same as age but with race..
test_livability.py	• test_make_table_request_success(): Mocks an API request to verify make_table_request correctly retrieves and processes JSON data. • test_livindex_by_zc_valid( ): Tests livindex_by_zc with valid zip codes to ensure correct score retrieval and formatting. • test_livindex_by_zc_empty_list(): Verifies that livindex_by_zc returns an empty list when given no zip codes. • test_livindex_by_zc_invalid_zip(): Ensures livindex_by_zc correctly handles invalid zip codes by returning only the zip code.

Chiffordable

test_utils.py	• test_complete_link_full_url(): Ensures complete_link returns the full URL unchanged when already complete. • test_complete_link_partial_url(): Verifies complete_link correctly appends a base URL to a relative URL. • test_complete_link_empty(): Confirms complete_link returns None when given a None value. • test_parse_script_content_valid(): Checks that parse_script_content correctly extracts and parses JSON from HTML. • test_parse_script_content_missing(): Ensures parse_script_content returns an empty dictionary when no JSON script is found. • test_parse_script_content_invalid_json(): Verifies parse_script_content raises an error when encountering invalid JSON. • test_save_to_csv(): Tests save_to_csv by writing sample data to a CSV (without duplicates) file and verifying its existence and content. • test_fetch_page_success(): Tests that fetch_page returns correct response when request is successful • test_fetch_page_hhtp_error(): Tests that fetch_page raises error when response is not 200 • test_fetch_page_with_custom_headers(): Tests that fectch_page sends custom headers.
test_zillow.py	• test_extract_listings(): Verifies extract_listings correctly extracts a list of property listings from JSON data. • test_extract_listings_empty(): Ensures extract_listings returns an empty list when no listings are present in the JSON data. • test_extract_listings_invalid_json(): Checks that extract_listings raises a KeyError when given an invalid JSON structure. • test_get_listing_info(): Confirms get_listing_info extracts property details correctly, including address, price, and bedrooms. • test_get_listing_info_missing_fields(): Ensures get_listing_info handles missing fields gracefully, returning empty or None values. • test_get_listing_info_price_cleanup(): Verifies that get_listing_info correctly cleans and extracts numerical values from price strings. • test_nextpage_from_xpath(): Checks that nextpage_from_xpath correctly extracts the next page URL from an HTML response. • test_nextpage_from_xpath_last_page(): Ensures nextpage_from_xpath returns None when there is no next page available.

test_zillow_details.py	• test_extract_details_with_floorplans(): Ensures extract_details correctly extracts listings from the "floorPlans" section of the JSON. • test_extract_details_with_ungrouped_units(): Verifies extract_details extracts listings from the "ungroupedUnits" section of the JSON. • test_extract_details_without_listings(): Checks that extract_details returns None when no listings are available. • test_extract_details_invalid_json(): Ensures extract_details returns an empty list when given an invalid JSON structure. • test_extract_details_missing_gdp(): Confirms extract_details returns an empty list when a key is missing in the JSON response. • test_get_details_info_multiple_prices(): Validates get_details_info correctly handles multiple price ranges for different apartments. • test_get_details_info_single_price(): Tests get_details_info for a listing with a single price. • test_get_details_info_missing_prices(): Ensures get_details_info returns an empty list when no price information is provided. • test_get_details_info_same_min_max(): Checks that get_details_info does not duplicate listings when minPrice == maxPrice. • test_combine_details_multiple(): Verifies combine_details correctly merges general listing data with multiple apartment details. • test_combine_details_single(): Ensures combine_details correctly merges general listing data with a single apartment detail.
test_utilsapp.py	• test_community_point_normalset(): Tests if known coordinate points return the correct community names. • test_community_point_outside(): Tests if coordinates outside the defined communities return None. • test_community_name_normalset(): Tests if known community names return correct demographic details. • test_community_name_unknown(): Tests if an unknown community name returns None. • test_calculate_rent_normalset(): Tests if rent calculations return expected values for various incomes and rent share percentages. • test_calculate_rent_noincome():Tests if rent calculations return zero when income is zero. • test_calculate_rent_noshare(): Tests if rent calculations return full income when no share percentage is provided.