Michaela Mellen

CSCI E-7

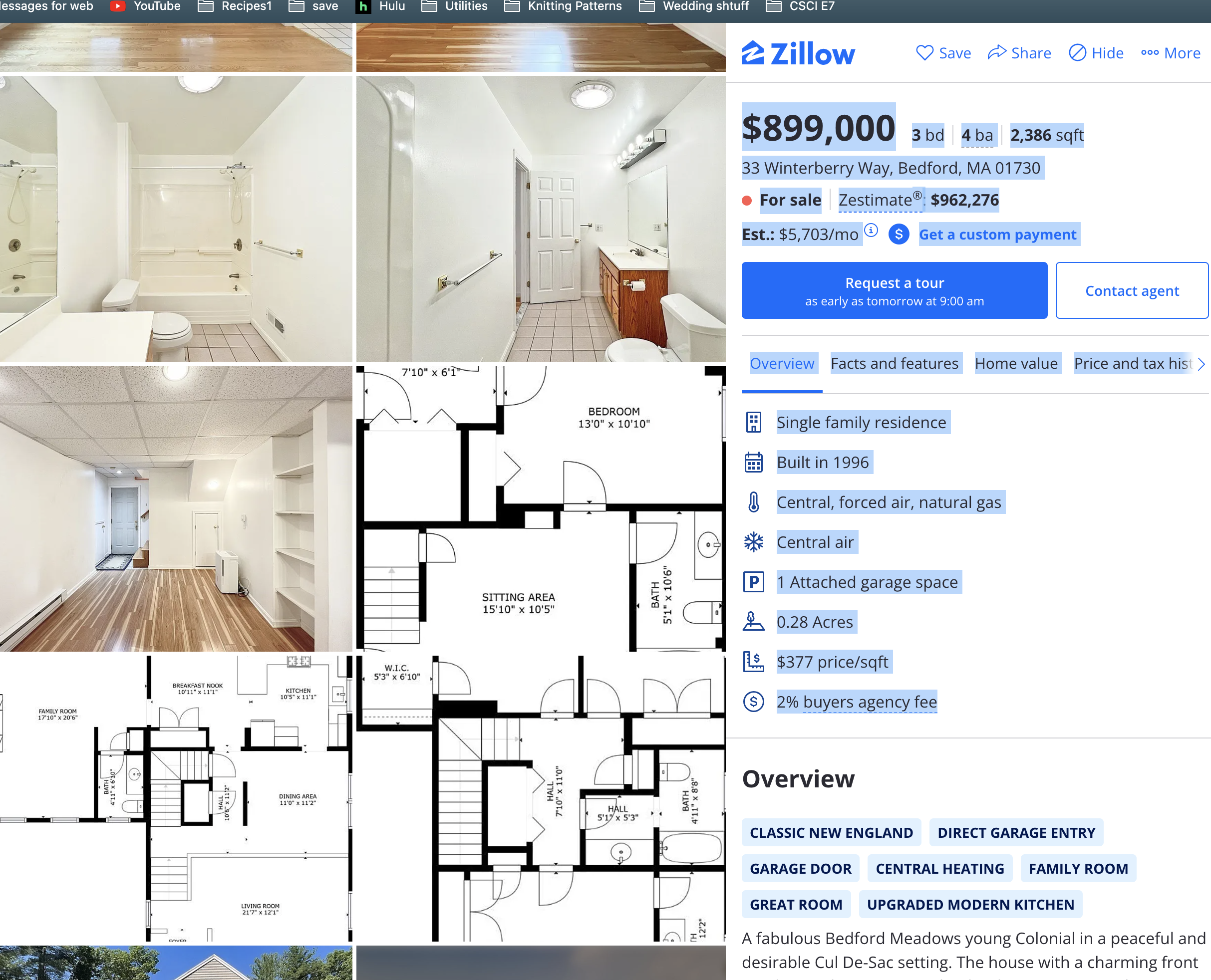
Term Project

Zillow Parser

Zillowparse.py is a program intended to extract key datapoints from text copied and pasted from a Zillow real estate listing. Once extracted, this data is appended to a .csv. In addition to collecting data from Zillow, the address is then geocoded in order to calculate estimated distance/commute times to a given point. The address zip code is also matched to a .csv export of the Federal Housing Finance Agency House Price Index, a measure of tracking real estate values and pricing changes in single family houses. The following is an outline of all functions defined within the program:

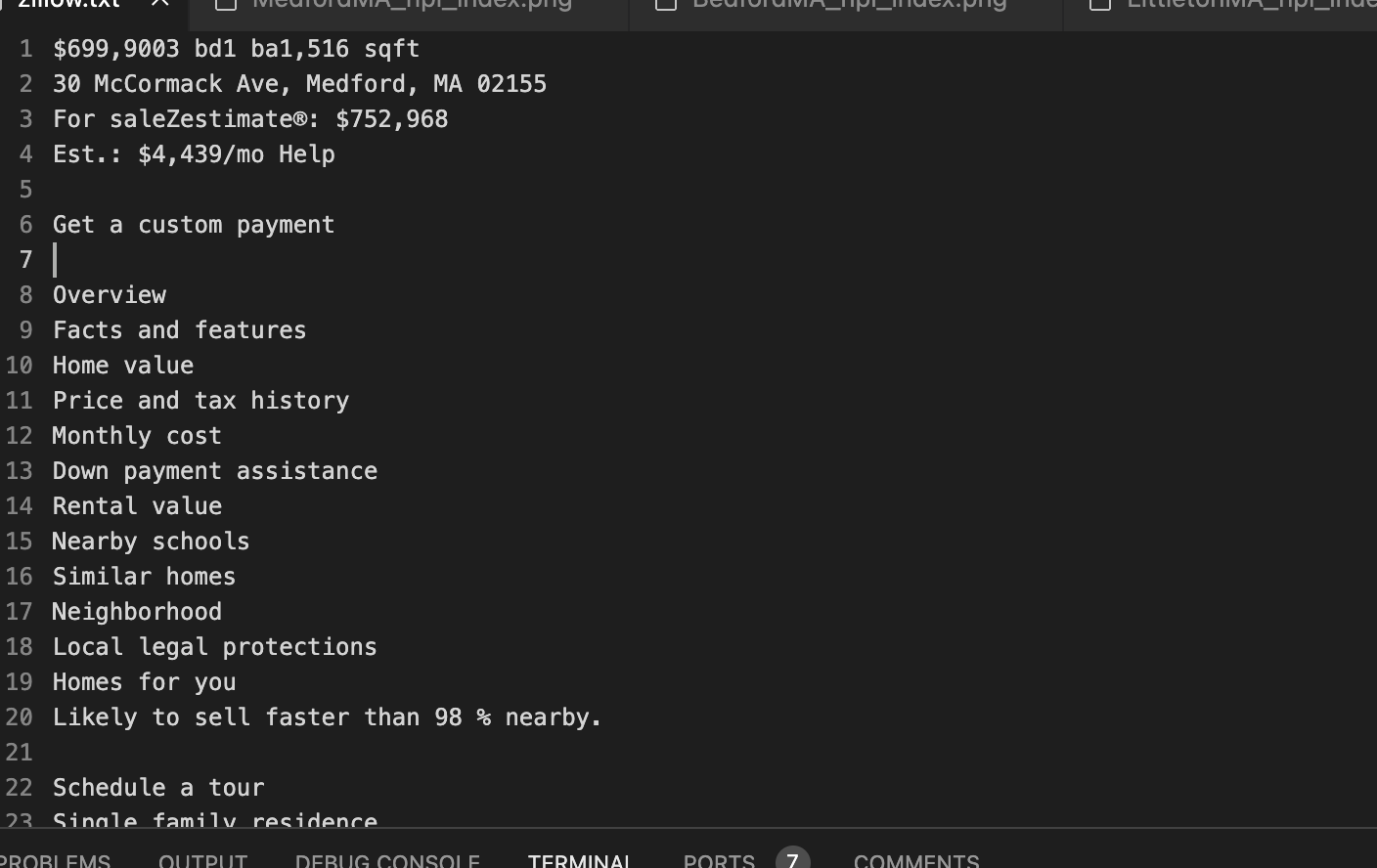
**update\_dict():**

This is the main function used to parse data from zillow.txt. The user first copies the key data from a Zillow listing:

****

This data is then pasted in the Zillow.txt file. The function will then run through each line of the text file in order to identify markers that indicate the location of key data points:

1. *Price:* When copying/pasting from Zillow, the price is always marked by the first character in the line being a $. This should also always be the largest number in the text (with the exception of the Zestimate, which can be higher). However, when copying and pasting, the space is lost between price and bedroom count. As a result, numbers from the $ to the second to last character before the next space are extracted.





*\*\*The assumption is no listing run through the program will have 10 or more bedrooms/bathrooms. While this may be a limitation, I expect this to be a very small edge case.*

1. *Bedroom and bathroom count:* Again, the function will identify what line bedroom/bathroom count is on based on finding the string ‘bd’. From there, it will extract each number based on its location in the line.
2. *Address:* The line for the address is identified by any matching string from the list of state abbreviations. From there, each part of the address (address, city, state, zip) is isolated by its location in reference to other elements and commas in the line.
3. *Home size:* The square footage of each home is also included on the line with price and beds/baths. Assuming ‘ba’ is in the line, along with ‘sqft’, the number will be isolated.
4. *Plot size:* Plot sizes can be listed in either sqft or acres. Lines containing this information are identified either by containing the string ‘acre’ or ‘sqft’ without the presence of a $.
5. *Heating type:* The program will identify whether a line contains any elements from a list of common heating types. It will also index the location of the line, which will then be used to find the line containing A/C type. There is some overlap in terminology for heating and AC, so this is intended to handle instances where heating (such as ‘Central’) and AC are the same type.
6. *AC type:* The line containing AC type is identified by its location in relation to the heating type line, along with its inclusion of an AC type from the AC type list.

Once the file has been parsed, a dictionary is returned.

**geocode(address):**

Addresses are geocoded using the opensource Nominatim module from the geopy.geocoders library. Through error handling, this function geocodes addresses either passed via dictionary (returned from the update\_dict() function) or via string input. Coordinates (formatted as a tuple of longitude, latitude) are returned from the function.

**commutetime():**

This function will take coordinates passed from the geocode(address) function. It will also accept a user input for an address, or will default to a location in Bedford, MA. Using the openrouteservice API, a dictionary with details about commute time (in seconds) and distance (in meters) is outputted. From there, summary information is extracted from the output dictionary and returned by the function.

**csv\_add(dictionary):**

This function will accept a dictionary as a parameter (in this case, the dictionary output by the update\_dict() function). First, it will append the values from commutetime() to the dictionary. From there, the DictWriter command will append it to zillow.csv.

**hpi\_index(dictionary):**

This function will also accept a dictionary as a parameter (again, the one output by the update\_dict() function), and will use the zip code from the dictionary to filter the House Price Index file for the given zip code and a cohort group of 3-digit zip code matches (for instance, zip code 02155 will have a cohort of zips 021\_\_). The pandas library is used to filter the dataframe (HPI\_AT\_BDL\_ZIP5.csv), and creates two separate dataframes (filtered\_df and cohort\_df). In order to account for zip codes beginning with 0, zip codes are also converted to strings and, in instances where a leading 0 may have been stripped (resulting in only a 4-digit zip), the zfill command is used to re-add the dropped 0. Rows missing data (or containing non-numerical data) for the ‘Annual Change’ column are also removed. In the cohort dataframe, rows are grouped by year and the mean Annual Change is calculated.

Once both dataframes have been created, the year-over-year percent changes in HPI index are plotted using matplotlib, and an image file is saved. This is intended to act as a resource for seeing how home values in a particular community have shifted over time, and how it may compare to similar communities.

**main():**

This is the main function which will call the other functions. First, a user is asked whether they would like to update the Zillow.csv file (Y/N). If Y, it will call the update\_dict() function to return a dictionary, then used in the csv\_add(dictionary) and hpi\_index(dictionary) functions. Once the program has finished executing, a confirmation statement is printed.

**Conclusion:**

While I initially set out with a very ambitious goal of scraping Zillow directly and populating a user-facing dashboard, I encountered many obstacles. First, I was unable to collect data directly from Zillow due to a CAPTCHA system. Hence, I went down the route of copying/pasting into a .txt file as a work-around. Furthermore, I also encountered technical issues using a Google Maps API as I initially intended; instead, the openrouteservice API served as a replacement.

I was also limited in creating a dashboard to highlight data collected in the final .csv. Instead, I scaled back my initial goals with this program, so the final outputs are the zillow.csv file and the HPI plots.

While I was unable to accomplish many of the items outlined in my initial project proposal, I do believe my end program will be an extremely useful tool for myself and anyone else searching for homes.

**Sources:**

Nominatim: <https://nominatim.org/>

Open Route Service: <https://openrouteservice.org/>

Zillow: <https://www.zillow.com/>

Federal Housing Finance Agency House Price Index Dataset: <https://www.fhfa.gov/DataTools/Downloads/Pages/House-Price-Index-Datasets.aspx>