Web Crawler for Boston Realty Sites - Empowering Smart Living Choices

Karthik Gowda Ramakrishna, Karan Vasepalli, Sashank Mudaliar

1. Introduction

This project aims to reform the housing search experience in Boston through a comprehensive data-driven approach that includes web crawling, real-time dashboarding, and automated deployment. The rising costs of housing and limited options have posed significant challenges for students and young professionals seeking affordable living spaces. This project addresses these issues by developing an efficient crawler capable of collecting housing and apartment data from various realty sites in Boston and its neighboring areas such as Cambridge, Brookline, Allston and Medford. The crawler will ensure comprehensive coverage and accuracy of available listings.

In addition to the web crawler, this project emphasizes the creation of an intuitive and real-time housing dashboard. The dashboard will dynamically update with the latest data from the crawler, allowing users to explore various housing options, compare locations, amenities, and prices, and make well-informed decisions about their ideal homes.

To further enhance user experience and automate the data collection process, this project will include an automated deployment system. The deployer will run the web crawler at specified intervals, ensuring that the dashboard always reflects the most up-to-date housing information available.

By combining algorithm design, data collection, cleaning, exploratory data analysis, real-time dashboarding, and automated deployment, this project aims to streamline the housing search process for Boston residents. The innovative solution provided by this project will empower individuals to access affordable housing options while minimizing the pressures and economic burdens associated with the housing hunt.

1.1. Our motivation

Karthik:

In my personal experience, the struggle to find affordable housing in Boston has been a constant source of stress and financial strain. As a graduate student and young professional, I understand the challenges faced by students and individuals like myself, trying to balance the desire for a convenient and comfortable living space

with the realities of the city's skyrocketing rents and the frustrations caused by deceptive listings that lead to wasted time and false hope.

Having witnessed friends and peers forced to compromise on the quality and location of their housing due to limited options, I am deeply motivated to find a solution. The importance of this project lies in its potential to ease the burden on students and young professionals, enabling them to access affordable housing options without compromising on their proximity to their workplaces, schools, or essential amenities.

Karan:

As someone who has worked in the data space before, I wanted to combine my already existing skills with the new techniques learnt in this course to solve a real world problem. Boston's housing problem is an ideal project for this. Keeping track of inventory and rents by areas will prove to be a steep learning curve, making use of skills like algorithm design, data collection, data cleaning, data wrangling, EDA, data visualization and dashboarding.

Sashank :

In the college city of Boston, finding the perfect apartment is a challenge that many face. Over the last several years, the rents in Boston have been skyrocketing, creating an affordability crisis and placing significant demand on affordable housing options. As a result, students, in particular, find themselves struggling with the dilemma of either paying exorbitant rents or having to live further away from their campuses to secure more affordable housing.

We aim to tackle this issue by developing a web crawler and dashboard that revolutionizes apartment hunting in Boston. By aggregating housing and apartment information from various realty sites, we want to create a user-friendly platform that empowers individuals to make well-informed decisions about their living spaces.

1.2. Clearly Defined Question

How can we develop an efficient web scraper to collect housing and apartment data from various Boston realty sites and create an intuitive and user-friendly dashboard for users to easily find their ideal homes?

2. Analysis

This section provides a comprehensive overview of the analysis conducted in this project, with a focus on meeting the specific requirements outlined in the original proposal document. Throughout the analysis, we describe the methods employed to gather the data and address the problem of finding affordable housing in Boston. Clear and detailed steps are presented at each stage of the analysis, referencing specific topics and modules covered in the CS 5800 course.

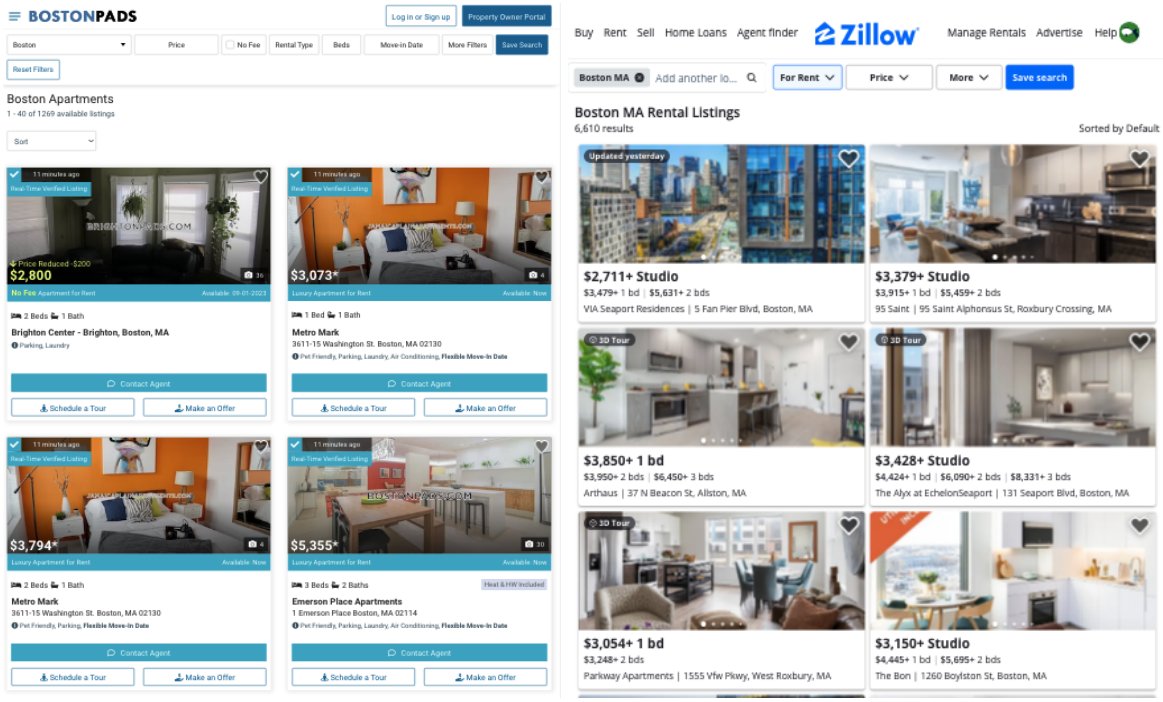
High level flow diagram of the complete crawling process (Source: https://data-ox.com/what-is-data-scraping-basic)

2.1. Web crawler

In the data gathering phase, we developed a web crawler using Python. The web crawler was designed to visit popular realty websites such as Bostonpads.com, Zillow.com, Hotpads.com, and Ygl.is. It starts by visiting the provided main pages of these websites as roots links for the crawler, and searching for rental listings. Upon finding a listing, the web crawler extracts essential details such as price, number of beds, baths, amenities, location, and agent information.

To systematically traverse through these websites, we implemented the Breadth-First Search (BFS) algorithm. BFS ensured that the crawler explored each webpage in a layer-by-layer manner, visiting all relevant pages before moving to the next level. This approach allowed us to gather comprehensive data across all pages of the websites, enhancing the coverage and accuracy of our dataset.

We used libraries like BeautifulSoup for HTML parsing, enabling us to extract specific information from the websites' HTML content efficiently. The Requests library facilitated HTTP requests to access and navigate through the websites, ensuring seamless crawling.



Some examples of the websites our program crawled

2.2. Saving data to SQLite database

After gathering the rental listing data, we organized it by saving it into an SQLite database file. SQLite is a lightweight, serverless, and self-contained database engine, ideal for managing relatively small datasets like ours. We created appropriate tables within the SQLite database to store each attribute of the rental listings, making data retrieval and analysis efficient.

	last_updated	price	n_beds	n_baths	avbl_date	location	utilities
0	2023-08-05	3073	1.0	1.0	2023-08-02 00:00:00	3611-15 Washington St.	Utilities paid for: Heat - No, Hot Water - No,
1	2023-08-05	3794	2.0	1.0	2023-08-02 00:00:00	3611-15 Washington St.	Utilities paid for: Heat - No, Hot Water - No,
2	2023-08-05	2667	1.0	1.0	2023-08-02 00:00:00	3611-15 Washington St.	Utilities paid for: Heat - No, Hot Water - No,
3	2023-08-05	5355	3.0	2.0	2023-08-03 00:00:00	1 Emerson Place	Utilities paid for: Heat - Yes, Hot Water - Yes
4	2023-08-05	8214	3.0	2.0	2023-07-31 00:00:00	131 Seaport Boulevard	Utilities paid for: Heat - No, Hot Water - No,
5	2023-08-05	6291	2.0	2.0	2023-08-01 00:00:00	100 Sudbury St.	Utilities paid for: Heat - No, Hot Water - No,
6	2023-08-05	6750	5.0	2.0	2023-08-15 00:00:00	BOSTON - ALLSTON	Utilities paid for: Heat - Yes, Hot Water - Yes
7	2023-08-05	4500	2.0	2.0	2023-07-15 00:00:00	BOSTON - NORTH END	Utilities paid for: Heat - No, Hot Water - No,
8	2023-08-05	2800	2.0	1.0	2023-09-01 00:00:00	BOSTON - BRIGHTON - OAK SQUARE	Utilities paid for: Heat - Yes, Hot Water - Yes
9	2023-08-05	3300	3.0	1.0	2023-09-01 00:00:00	BOSTON - EAST BOSTON - MAVERICK	Utilities paid for: Heat - No, Hot Water - No,
10	2023-08-05	3000	4.0	1.0	2023-09-01 00:00:00	BOSTON - DORCHESTER/SOUTH BOSTON BORDER	Utilities paid for: Heat - No, Hot Water - No,
11	2023-08-05	2700	3.0	1.0	2023-09-01 00:00:00	BOSTON - DORCHESTER/SOUTH BOSTON BORDER	Utilities paid for: Heat - No, Hot Water - No,
12	2023-08-05	3000	4.0	1.0	2023-09-01 00:00:00	BOSTON - DORCHESTER/SOUTH BOSTON BORDER	Utilities paid for: Heat - No, Hot Water - No,
13	2023-08-05	2300	1.0	1.0	2023-09-01 00:00:00	BOSTON - MISSION HILL	Utilities paid for: Heat - Yes, Hot Water - Yes
14	2023-08-05	5900	4.0	3.0	2023-09-01 00:00:00	BOSTON - DORCHESTER - SAVIN HILL	Utilities paid for: Heat - No, Hot Water - No,
15	2023-08-05	3200	1.0	1.0	2023-09-01 00:00:00	BOSTON - SOUTH END	Utilities paid for: Heat - No, Hot Water - No,

Snapshot of the extracted table

Each rental listing entry was represented as a row in the database, with columns representing different attributes, such as price, beds, baths, location, and amenities. The use of an SQLite database facilitated easy querying and manipulation of data, enabling us to perform various analyses on the dataset.

2.3. Loading Database to Tableau

To visualize and explore the rental housing data, we utilized Tableau, a powerful data visualization tool. We connected Tableau to the SQLite database, enabling direct access to the rental listing information.

In Tableau, we designed various charts, graphs, and dashboards to present insights on the Boston housing market. We created visualizations to display trends in rental prices, distributions of beds and baths, and the most common amenities. The dashboard allowed users to filter the data interactively, enabling them to explore the dataset based on their specific preferences.

To show relationships between different attributes, we used Tableau's interactive features like linking visualizations and adding filters. For example, users could select a specific location on a map to view rental listings available in that area and filter by price range, beds, or baths.

2.4. Automating the Process with PySpark

To automate the entire data pipeline, we incorporated PySpark, a Python library for distributed data processing with Apache Spark. PySpark allowed us to efficiently process large datasets and automate repetitive tasks.

We developed PySpark scripts to run the web crawler automatically at predefined intervals. The crawler would extract rental listing data, clean and preprocess it, and then save it into the database. This automation ensured that the data remained up-to-date, reflecting the latest listings available on the websites.

By integrating PySpark into the analysis workflow, we streamlined the data collection and maintenance process, providing users with real-time and accurate housing information when accessing the Tableau dashboard.

The provided Python program, included as an appendix to this report, encompasses the web crawler, data processing scripts, and automation using PySpark. This program demonstrates the systematic approach we adopted throughout the analysis, in alignment with the topics and modules covered in the CS 5800 course. By following clear and well-defined steps, the program showcases our commitment to

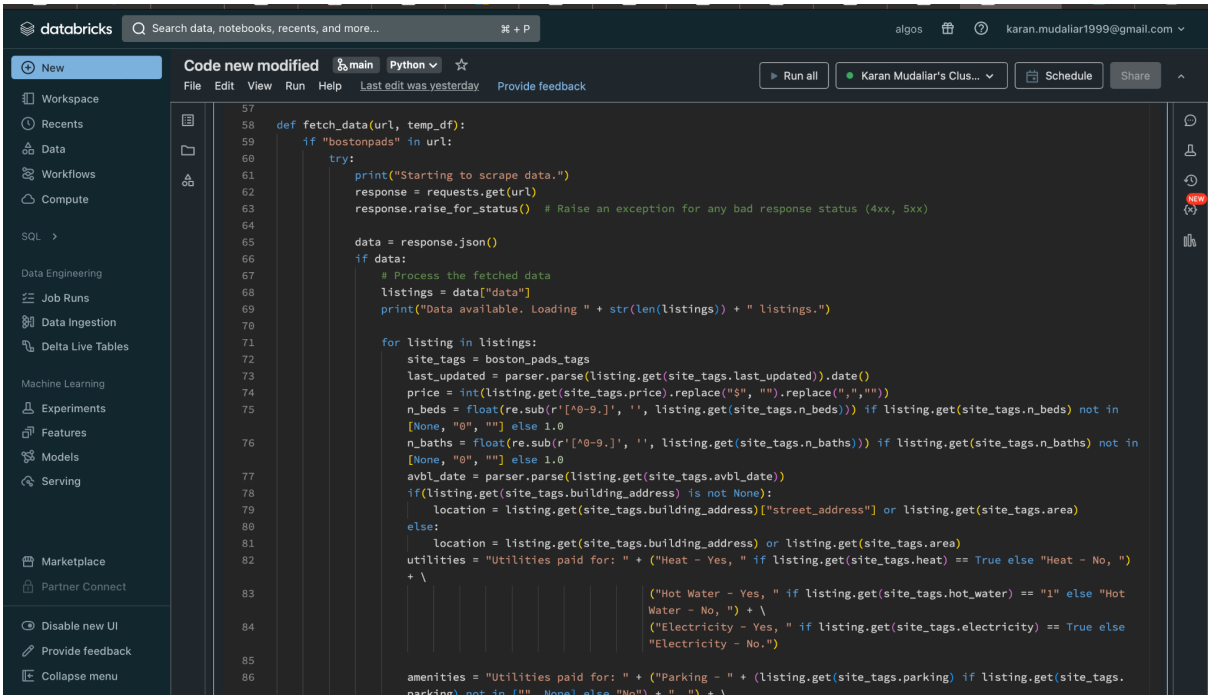
delivering a robust solution for data gathering, processing, and visualization to address the challenges of affordable housing in Boston.

2.5. Deployment of the project

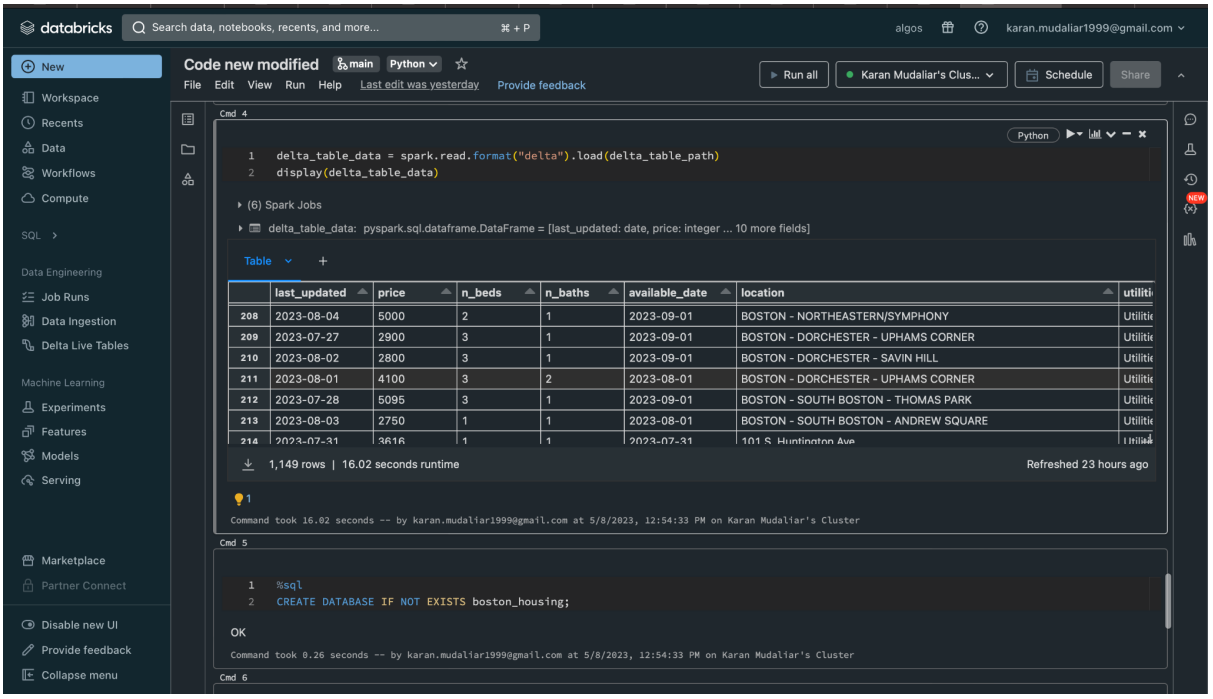
For the deployment of this project, we decided to go with a simple architecture to ensure the smooth orchestration of data operations. Databricks runs on a spark environment natively. Using Databricks, the process of loading, scraping, and processing data was streamlined, enabling efficient data ingestion and transformation. The utilization of Delta tables provided a robust solution for storing the processed data, ensuring data integrity and enabling easy data manipulation. To ensure continuous updates and insights, the entire workflow is automated and scheduled to trigger daily in the morning, guaranteeing that the latest information is always available. This entire end-to-end solution is hosted on the Azure cloud, harnessing the scalability and reliability of cloud infrastructure to deliver consistent and dependable performance. This deployment strategy not only enhances the efficiency of the data pipeline but also lays a strong foundation for further expansion and optimization of the system.

In the deployment version, a strategic decision was made to enhance the solution's architecture by transitioning away from SQLite and adopting a more robust and scalable approach. By shifting to this deployment methodology, the project gained significant improvements in terms of performance, scalability, and maintainability, positioning it for seamless expansion and sustained impact.

Another advantage of the chosen deployment strategy is the direct integration of Delta tables with Tableau. This integration enables the creation of dynamic and real-time dashboards that provide live updates based on the daily refreshed data. By establishing this connection between Delta tables and Tableau’s cloud infrastructure, the project maximizes the accessibility and usability of the collected housing data, empowering stakeholders with timely insights into Boston housing trends. This direct linkage not only streamlines the process of generating visualizations but also ensures that decision-makers have access to the most current and accurate information, further enhancing the solution's impact and utility.



Crawler Code



Delta Table

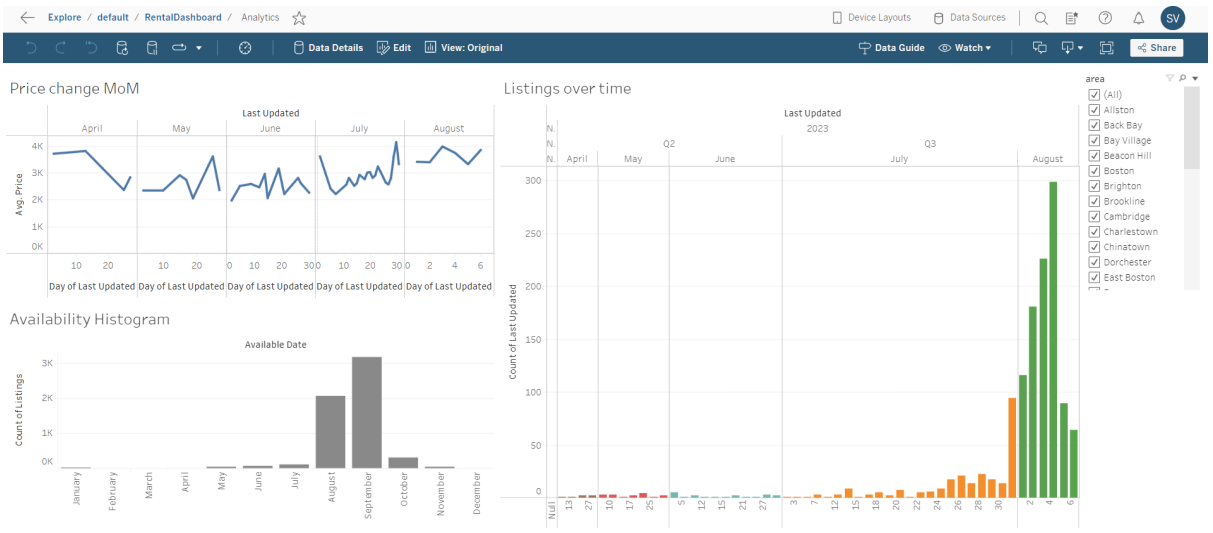


Tableau Online Analytics Dashboard

Rental Data

Id	area	Location	Month of A..	Day of A..	N Beds	N Baths	Price
1	West End	1 Emerson Place	August	3	3	2	5,355
2	Boston	BOSTON - NORTHEASTERN/SYMPHONY	September	1	1	1	1,900
3	Boston	BOSTON - DOWNTOWN	September	1	1	1	3,000
4	Charlestown	BOSTON - CHARLESTOWN	September	1	1	1	1,500
5	Back Bay	BOSTON - BACK BAY	September	1	2	1	3,900
6	Boston	BOSTON - FENWAY/KENMORE	August	15	1	1	2,395
7	Boston	BOSTON - DOWNTOWN	September	1	1	1	2,500
8	Allston	BOSTON - ALLSTON	August	31	1	1	2,300
9	Jamaica Plain	BOSTON - JAMAICA PLAIN - FOREST HILLS	September	2	3	1	3,200
10	Jamaica Plain	BOSTON - JAMAICA PLAIN - FOREST HILLS	September	1	3	1	3,200
11	Mission Hill	BOSTON - MISSION HILL	September	1	2	1	3,000
12	Allston	BOSTON - ALLSTON	August	15	5	2	6,500
13	North End	BOSTON - NORTH END	August	15	2	2	4,950
14	Brighton	BOSTON - BRIGHTON - CLEVELAND CIRC..	September	1	1	1	2,775
15	Dorchester	BOSTON - DORCHESTER - UPHAMS COR..	September	1	6	3	7,000
16	Fort Hill	BOSTON - FORT HILL	September	1	6	3	6,000
17	Boston	BOSTON - FENWAY/KENMORE	September	1	1	1	2,995
18	Boston	BOSTON - FENWAY/KENMORE	September	1	1	1	3,080
19	Brighton	BOSTON - BRIGHTON - BOSTON COLLEGE	September	1	6	2	11,130
20	Boston	BOSTON - FENWAY/KENMORE	September	1	1	1	2,845
21	Boston	BOSTON - FENWAY/KENMORE	September	1	1	1	2,895
22	Boston	BOSTON - FENWAY/KENMORE	September	1	1	1	2,795

Areas
- (All)
- Allston
- Back Bay
- Bay Village
- Beacon Hill
- Boston
- Brighton
- Brookline
- Cambridge
- Charlestown
- Chinatown
- Dorchester
- East Boston
- Fenway

Beds
1.00 — 11.00

Baths
0.00 — 11.50

Price
0 — 50000

Rental Details

Price: 2,895 $
Beds: 1.000
Baths: 1.000
Location: BOSTON - FENWAY/KENMORE
Last Updated: August 6, 2023

Utilities paid for: Heat - No, Hot Water - No, Electricity - No.
Amenities provided: Parking - No, Dish Washer - No, Laundry - No, Air Conditioning - No.

Agent Details

Nicholas Rizzolo
(857) 693-0029
nrizzolo@rentboardwalk.com
Agent Fees: Yes

Live Tableau Online Database

3. Conclusion

Sashank:

The successful implementation of our web crawler and user-friendly dashboard is significant milestone in changing the apartment hunting experience for students in Boston. By aggregating housing data from various realty sites, this project has provided convenient platform for students to make well-informed decisions about their living spaces.

Karan:

The project addresses the critical issue of affordability and accessibility to rental housing in the college city of Boston. With ever rising rents and limited affordable options, students and residents alike face immense challenges in finding their suitable rental homes near campuses, or offices. Our dashboard empowers users to navigate this complex housing market with less effort than most websites available online.

Achyut:

We've created this tool to make apartment hunting in Boston much easier. We all know how hard it can be to find the perfect place with all the limited options out there and their crazy high rents. Our Project searches through different rental websites and gathers all the rental home information in one place. We also built a user-friendly online dashboard that shows you a list of all the homes that match your criteria, empowering users to explore available options, assess rental trends, and ultimately find homes that best suit their needs and preferences.

References

1. Websites used for web crawling:

    - https://bostonpads.com/
    - https://www.yougotlistings.com/
    - https://www.zillow.com/
    - https://hotpads.com/

2. Material used for creating a dashboard in tableau.
   https://help.tableau.com/current/guides/get-started-tutorial/en-us/get-started-tutorial-connect.htm

3. Material used for setting up the online databricks pipeline.
   https://docs.databricks.com/en/getting-started/data-pipeline-get-started.html

4. Educational videos used as reference for web crawling and scraping.
   https://www.youtube.com/watch?v=8wjQTweW_uQ&ab_channel=TylerCaraza-Harter, https://www.youtube.com/watch?v=bIOzv83Yo58

5. Python data transformation and cleaning references.
   https://www.w3schools.com/python/python_ref_string.asp

Appendix

Github link- Link
Youtube link- https://youtu.be/CxU95pJUzOc

Web crawler code snippet:

```
import requests
import sqlite3
from dateutil import parser
from datetime import date
from bs4 import BeautifulSoup
import re

from WebpageClass import WebpageData
from SubareaFinder import find_area_bostonpads, find_area_ygl

# Set up SQLite database and server

# Create a connection to the SQLite database
conn = sqlite3.connect('Housing_Data.db')

# Create a cursor object to interact with the database
cursor = conn.cursor()

boston_pads_tags = WebpageData('date_modified', 'price',
'bed_room', 'baths',
                               'date_available',
'building_address', 'sub_area_name',
                               'agent_full_name', 'agent_email',
'agent_phone',
                               'fee', 'heat', 'hot_water',
'electricity', 'parking',
                               'dish_washer', 'laundry_location',
'air_conditioning')

# Create a table to store the scraped data
cursor.execute('''
               CREATE TABLE IF NOT EXISTS boston_rental_data (
                   id INTEGER PRIMARY KEY,
                   last_updated DATETIME,
```

```
                    price INT,
                    n_beds FLOAT,
                    n_baths FLOAT,
                    available_date DATETIME,
                    location TEXT,
                    area TEXT,
                    utilities TEXT,
                    amenities TEXT,
                    agent_name TEXT,
                    agent_email TEXT,
                    agent_phone TEXT,
                    agent_fees TEXT
                    )
                ''')

# Commit changes to the database
conn.commit()

def fetch_data(url):
    if "bostonpads" in url:
        try:
            print("Starting to scrape data.")
            response = requests.get(url)
            response.raise_for_status()  # Raise an exception for
any bad response status (4xx, 5xx)

            data = response.json()
            if data:
                # Process the fetched data
                listings = data["data"]
                print("Data available. Loading " +
str(len(listings)) + " listings.")

                for listing in listings:
                    site_tags = boston_pads_tags
                    last_updated =
parser.parse(listing.get(site_tags.last_updated)).date()
                    price =
int(listing.get(site_tags.price).replace("$", "").replace(",",""))
                    n_beds = float(re.sub(r'[^0-9.]', '',
listing.get(site_tags.n_beds))) if listing.get(site_tags.n_beds)
not in [None, "0", ""] else 1
```

```
                    n_baths = float(re.sub(r'[^0-9.]', '',
listing.get(site_tags.n_baths))) if listing.get(site_tags.n_baths)
not in [None, "0", ""] else 1
                    avbl_date =
parser.parse(listing.get(site_tags.avbl_date))
                    if(listing.get(site_tags.building_address) is
not None):
                        location =
listing.get(site_tags.building_address)["street_address"] or
listing.get(site_tags.area)
                    else:
                        location =
listing.get(site_tags.building_address) or
listing.get(site_tags.area)

                    area =
find_area_bostonpads(listing.get(site_tags.area))
                    utilities = "Utilities paid for: " + ("Heat -
Yes, " if listing.get(site_tags.heat) == True else "Heat - No, ")
+ \
                                                        ("Hot
Water - Yes, " if listing.get(site_tags.hot_water) == "1" else
"Hot Water - No, ") + \

("Electricity - Yes, " if listing.get(site_tags.electricity) ==
True else "Electricity - No.")

                    amenities = "Utilities paid for: " + ("Parking
- " + (listing.get(site_tags.parking) if
listing.get(site_tags.parking) not in ["", None] else "No") + ",
") + \
                                                        ("Dish
Washer - Yes, " if listing.get(site_tags.dish_washer) == True else
"Dish Washer - No, ") + \
                                                        ("Laundry
- " + (listing.get(site_tags.laundry_location) if
listing.get(site_tags.laundry_location) not in ["", None] else
"No") + ", ") + \
                                                        ("Air
Conditioning - " + (listing.get(site_tags.air_conditioning) if
listing.get(site_tags.air_conditioning) not in ["", None] else
"No") + ".")
```

```
                    agent_name = listing.get(site_tags.agent_name)
                    agent_email =
listing.get(site_tags.agent_email)
                    agent_phone =
listing.get(site_tags.agent_phone)
                    agent_fees = "No" if
listing.get(site_tags.agent_fees) == 0 else "Yes"
                    save_to_db(last_updated, price, n_beds,
n_baths, avbl_date, location, area,
                               utilities, amenities, agent_name,
agent_email, agent_phone, agent_fees)

            else:
                print("No data available.")
        except requests.exceptions.RequestException as e:
            print(f"An error occurred while fetching data:
{str(e)}")
            return None

    else:
        page_num = 1
        last_page_num = None

        while True:
            # Append the current page number to the URL
            current_url = f"{url}&page={page_num}"

            if page_num % 10 == 0:
                print(f'Page {page_num} fetched')

            # Make an HTTP request to the URL
            response = requests.get(current_url)
            response.raise_for_status()
            soup = BeautifulSoup(response.content, 'html.parser')

            # Find the listings on the current page
            listings = soup.find('div',
class_='search_results_area').find_all('div',
class_='property_item')
            if not listings:
                print("All listings from url fetched.")
```

```
                break

            # Fetch the broker details for the listings
            agent_details = soup.find('div',
class_='contacts').find_all('li')
            if agent_details:
                agent_name = agent_details[0].text.strip()
                agent_email = agent_details[2].find('a',
href=True)['href'].split(':')[1]
                agent_phone = agent_details[1].find('a',
href=True)['href'].split(':')[1]

            # Process the listings on the current page
            for listing in listings:
                listing_details = listing.find('div',
class_='item_props').find_all('div', class_='column')
                try:
                    price = int(re.sub(r'[^0-9]', '',
listing_details[0].text.strip()))
                except IndexError:
                    pass

                try:
                    n_beds = re.sub(r'[^0-9.]', '',
listing_details[1].text.strip())
                    n_beds = float(n_beds) if n_beds not in [None,
"0", ""] else round(float(1), 1)
                except IndexError:
                    pass

                try:
                    n_baths = listing_details[2].text.strip()
                    n_baths = float(re.sub(r'[^0-9.]', '',
n_baths)) if n_baths not in [None, "0", ""] else round(float(1),
1)
                except IndexError:
                    pass

                try:
                    avbl_date = re.sub(r'[^0-9/]', '',
listing_details[3].text.strip())
                    avbl_date = parser.parse(avbl_date) if
```

```
avbl_date not in ["",None] else date.today()
                except IndexError:
                    pass

                location = listing.find('a', class_='item_title')
                if location:
                    location = location.text.strip()
                    area = find_area_ygl(location)

                utilities = listing.find('a')
                if utilities:
                    utilities = utilities['href']
                amenities = listing.find('a')
                if amenities:
                    amenities = amenities['href']
                save_to_db(None, price, n_beds, n_baths,
avbl_date, location, area, utilities, amenities, agent_name,
agent_email, agent_phone, "Yes")

            # Check for the last page
            counter_div = soup.find('div', class_='counter')
            if counter_div:
                page_text = counter_div.text.strip()
                match = re.search(r'\d+', page_text)
                if match:
                    current_page = int(match.group())
                    if last_page_num is None:
                        last_page_num = current_page
                    elif current_page == last_page_num:
                        print("Reached the last page.")
                        return

            # Increment the page number for the next iteration
            page_num += 1

            # END OF fetch_data()

def save_to_db(last_updated, price, n_beds, n_baths, avbl_date,
location, area, utilities, amenities, agent_name, agent_email,
agent_phone, agent_fees):
    insert_query = 'INSERT INTO boston_rental_data (last_updated,
price, n_beds, n_baths, available_date, location, area, utilities,
```

```
amenities, agent_name, agent_email, agent_phone, agent_fees)
VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)'
    values = (last_updated, price, n_beds, n_baths, avbl_date,
location, area, utilities, amenities, agent_name, agent_email,
agent_phone, agent_fees)
    conn.execute(insert_query, values)
    conn.commit()

def main():
    website_list =
["https://m.bostonpads.com/api/listings-short?location=boston&uniq
ue=1&results_per_page=5000",

"https://ygl.is/bryn-allen-1?areas%5B%5D=Boston&areas%5B%5D=Boston
%3AAllston&areas%5B%5D=Boston%3ABack+Bay&areas%5B%5D=Boston%3ABay+
Village&areas%5B%5D=Boston%3ABeacon+Hill&areas%5B%5D=Boston%3ABrig
hton&areas%5B%5D=Boston%3ACharlestown&areas%5B%5D=Boston%3AChinato
wn&areas%5B%5D=Boston%3ADorchester&areas%5B%5D=Boston%3AEast+Bosto
n&areas%5B%5D=Boston%3AFenway&areas%5B%5D=Boston%3AFinancial+Distr
ict&areas%5B%5D=Boston%3AFort+Hill&areas%5B%5D=Boston%3AHyde+Park&
areas%5B%5D=Boston%3AJamaica+Plain&areas%5B%5D=Boston%3AKenmore&ar
eas%5B%5D=Boston%3ALeather+District&areas%5B%5D=Boston%3AMattapan&
areas%5B%5D=Boston%3AMidtown&areas%5B%5D=Boston%3AMission+Hill&are
as%5B%5D=Boston%3ANorth+End&areas%5B%5D=Boston%3ARoslindale&areas%
5B%5D=Boston%3ARoxbury&areas%5B%5D=Boston%3ASeaport+District&areas
%5B%5D=Boston%3ASouth+Boston&areas%5B%5D=Boston%3ASouth+End&areas%
5B%5D=Boston%3ATheatre+District&areas%5B%5D=Boston%3AWaterfront&ar
eas%5B%5D=Boston%3AWest+End&areas%5B%5D=Boston%3AWest+Roxbury&page
=1",

"https://ygl.is/andi-bauer-1?areas%5B%5D=Brookline&areas%5B%5D=Bro
okline%3ABeaconsfield&areas%5B%5D=Brookline%3ABrookline+Hills&area
s%5B%5D=Brookline%3ABrookline+Village&areas%5B%5D=Brookline%3AChes
tnut+Hill&areas%5B%5D=Brookline%3ACoolidge+Corner&areas%5B%5D=Broo
kline%3ALongwood&areas%5B%5D=Brookline%3AReservoir&areas%5B%5D=Bro
okline%3AWashington+Square&areas%5B%5D=Cambridge&areas%5B%5D=Cambr
idge%3AAgassiz&areas%5B%5D=Cambridge%3ACambridge+Highlands&areas%5
B%5D=Cambridge%3ACambridgeport&areas%5B%5D=Cambridge%3ACentral+Squ
are&areas%5B%5D=Cambridge%3AEast+Cambridge&areas%5B%5D=Cambridge%3
AHarvard+Square&areas%5B%5D=Cambridge%3AHuron+Village&areas%5B%5D=
Cambridge%3AInman+Square&areas%5B%5D=Cambridge%3AKendall+Square&ar
eas%5B%5D=Cambridge%3AMid+Cambridge&areas%5B%5D=Cambridge%3ANeighb
```

```
orhood+Nine&areas%5B%5D=Cambridge%3ANorth+Cambridge&areas%5B%5D=Ca
mbridge%3APorter+Square&areas%5B%5D=Cambridge%3ARiverside&areas%5B
%5D=Cambridge%3AWellington-Harrington&areas%5B%5D=Cambridge%3AWest
+Cambridge&areas%5B%5D=Somerville&areas%5B%5D=Somerville%3AAssembl
y+Square&areas%5B%5D=Somerville%3ABall+Square&areas%5B%5D=Somervil
le%3ADavis+Square&areas%5B%5D=Somerville%3AEast+Somerville&areas%5
B%5D=Somerville%3AInman+Square&areas%5B%5D=Somerville%3APowderhous
e+Square&areas%5B%5D=Somerville%3AProspect+Hill&areas%5B%5D=Somerv
ille%3ASpring+Hill&areas%5B%5D=Somerville%3ATeele+Square&areas%5B%
5D=Somerville%3ATen+Hills&areas%5B%5D=Somerville%3AUnion+Square&ar
eas%5B%5D=Somerville%3AWest+Somerville&areas%5B%5D=Somerville%3AWi
nter+Hill&page=1"
                    ]
    for url in website_list:
        fetch_data(url)

if __name__ == "__main__":
    main()
```