REPORT

LONDON PROPERTIES | PROJECT

BY

JUDE ONYEAJUNWANNE

(jude.onyeajunwanne@yahoo.com)

May, 2023

Introduction:

This project is carried to create a master list of all the property listing in London. We considered properties list under 'for sale' category.

It required the collation of property listing from major listing websites by scraping the websites to obtain the required data.

Procedure:

We first considered the top two listing websites which according to google search are rightmove.co.uk and zoopla.co.uk, scraped the data from them, cleaned the data and combined them.

Scraping Rightmove

Scraping the rightmove.co.uk website required scraping hidden web data as most of the valuable listing features with regards to our objective are hidden. This is because, reghtmove uses a JSON-powered front end to render its data. All the listing features on rightmove website are hidden in HTML variable script called PAGE_MODEL. To scrape such hidden data, we are required to scrape each listing page to retrieve the HTML of the property page (we used httpx), then use parsel to parse the HTML in order to find the script variable PAGE_MODEL and finally load the PAGE_MODEL JSON content as python dictionary and clean it up.

We first scraped out the URL of each listing page for a particular London borough. Rightmove uses special code for each London Borough. So we created a dictionary of all the London boroughs and their rightmove codes. We then scraped out the listing URLs and saved as a list of links called 'Links_data'.

```
#Converting the data to a dataframe and links_list
df = pd.DataFrame.from_dict(data)

#removing duplicates
df_rightmove_links= df.drop_duplicates()

#saving the links as a csv file
df_rightmove_links.to_csv(r"results/RL.csv", encoding="utf-8", header="true", index = False)

#cleaning up the links
df_rightmove_links['Links'] = df_rightmove_links['Links'].str.replace(r'\?channel=RES_BUY', '')


# Convert the link dataframe back to links_list
Links_list = df_rightmove_links['Links'].tolist()
```

We then fed the Links_data to the main scraping python program in order to scrape out the property data.

```python
async def run():
    """example run calls for our scraper functions"""
    out_path = Path.cwd() / "results"
    out_path.mkdir(exist_ok=True)


    data = await scrape_properties(Links_list)
    out_path.joinpath("rm_properties_all.json").write_text(json.dumps(data, indent=2))


if __name__ == "__main__":
    asyncio.run(run())
```

In the main scraper, we used httpx.AsyncClient object to avoid blocking because it is able to retrieve HTML pages asynchronously. This was done under the scrape_properties function which uses asyncio.as_completed to gather up HTTP requests and send concurrently. The responses were then parsed using simple XPath which finds <script> element containing PAGE_MODEL variable. This process created so much data from which we used jmespath to parse in order to trim it down and select the ones needed.

The final scraped data was converted to a csv file and saved as rm_property_all.csv.

```python
import pandas as pd

# Read the JSON file
with open('results/rm_properties_all.json', 'r') as file:
    json_data = json.load(file)

# Convert JSON data to DataFrame
df_rm = pd.DataFrame(json_data)

# save a copy of dataframe as csv (commented out in order not to overwrite what has already been saved)
df_rm.to_csv('results/rm_properties_all.csv', index=False)
```

Scraping Zoopla

Zoopla website is highly protected and it was practically impossible to scrape out data from it unless with the use of an API. I used scrapfly API in scraping the website.

The website uses Next.js to render its pages and it generates hidden web data for each property page. To get the useful property data requires the extraction of the hidden JSON data hidden in <script id="__NEXT_DATA__"> and parsing it using jmespath.

First, I scraped out the URL for the City of London Borough and Barking and Dagenham and formed a list of URLs called ‘urls’. This was then fed to the main scraper.

```python
async def scrape_properties(urls: List[str]) -> List[PropertyResult]:
    """scrape Zoopla property pages and parse results"""
    to_scrape = [ScrapeConfig(url=url, asp=True, country="GB") for url in urls]
    properties = []
    async for result in scrapfly.concurrent_scrape(to_scrape):
        parsed_result = parse_property(result)
        properties.append(parsed_result)
    return properties


if __name__ == "__main__":

    zp_properties_all = asyncio.run(scrape_properties(urls))

    df = pd.DataFrame(zp_properties_all)

     # Save as CSV(commented out in order not to overwrite what has already been saved)
    df.to_csv("results/zp_properties_all.csv", index=False)

    # Save as JSON
    df.to_json("results/zp_properties_all.json", orient="records")
```

The main scraper program used scrapfly API to establish connection with the website to avoid blocking.

```python
from scrapfly import ScrapeConfig, ScrapeApiResponse, ScrapflyClient

scrapfly = ScrapflyClient(key="scp-live-595ffb14156649c2b839689a8987c9e1", max_concurrency=2)
```

The hidden data was then extracted by loading the HTML as a **parsel.selector** object and using a css selector to select the <script> with the element id=__NEXT_DATA__ where all the property data are hidden in a JSON file format. The JSON file was then loaded up as a python dictionary and returned as **'result'.** The final JSON data which contained so much data was reduced using JMESPath query.

The final scraped data was saved as a csv file named zp_properties_all.csv .

N.B: I only scraped data for city of London and Barking and Dagenham for zoopla due to the limit in the purchased scrapfly API. I can scrape as much data as needed with unlimited API unit which costs $500 per month.

Working with Scraped Data

Working on RightMove Data:

The csv data rm_properties_all.csv was converted to a dataframe called df_rm having 1785 rows and 28 columns. The data contained unrefined data from the scraping and therefore needed to be cleaned.

```
df_rm
```

	id	Status	title	description	property_type	bedrooms	bathrooms	price	price_sqft	address	...	Agent_Address	neare
0	74504326	True	1 bedroom apartment for sale in One Bishopsgat...	This 21st floor, one bedroom apartment is well...	Apartment	1.0	1.0	£1,325,000	£2086.61 per sq. ft.	{'displayAddress': 'One Bishopsgate Plaza, 80 ...	...	33 Margaret Street\r \nLondon \r\nW1G 0JD\r\n	
1	133932536	True	Studio apartment for sale in Crane Court, Lond...	Recently refurbished studio in Central London....	Apartment	NaN	1.0	£450,000	NaN	{'displayAddress': 'Crane Court, London, EC4A'...	...	122-124 St Johns Street,\r \nLondon, \r\nEC1V 4JS	
2	128173136	True	Studio flat for sale in The Heron, Moorgate,	Set on the 14th floor, this immaculate	Studio	NaN	NaN	£575,000	£1399.03 per sq. ft.	{'displayAddress': 'The Heron, Moorgate, Londo...	...	50 - 54 Clerkenwell Road,\r \nLondon,	

The cleaning process required changing all the Boolean to strings, removing html tags in different columns and removing unnecessary symbols

```
from bs4 import BeautifulSoup
#replacing boolean to string in status and CouncilTax_Status column
df_rm['Status'] = df_rm['Status'].replace({True: 'Available', False: 'Unavailable'})
df_rm['CouncilTax_Status'] = df_rm['CouncilTax_Status'].replace({True: 'Not Exempted', False: 'Exempted'})

#cleaning the agen address column
df_rm['Agent_Address'] = df_rm['Agent_Address'].str.replace('\r\n', ' ')

#cleaning up the nearest station
df_rm['nearest_stations'] = df_rm['nearest_stations'].apply(lambda x: ast.literal_eval(x))
df_rm['nearest_stations'] = df_rm['nearest_stations'].apply(lambda x: [f"{item['name']}:{item['distance']}" for item in x
df_rm['nearest_stations'] = df_rm['nearest_stations'].apply(lambda x: ', '.join(x))
#cleaning and refining the description column
df_rm['description'] = df_rm['description'].str.replace('<strong>', '').str.replace('</strong>', '').str.replace('<br>',
#cleaning the address column
df_rm['address'] = df_rm['address'].apply(lambda x: ast.literal_eval(x))
df_rm['address']=df_rm['address'].apply(lambda x: x['displayAddress'])
df_rm['address'] = df_rm['address'].apply(lambda x: BeautifulSoup(x, 'html.parser').get_text())

#cleaning and refining photos and floorplans column
df_rm['photos'] = df_rm['photos'].apply(lambda x: ast.literal_eval(x))
df_rm['photos'] = df_rm['photos'].apply(lambda x: [item['url'] for item in x])

df_rm['floorplans'] = df_rm['floorplans'].apply(lambda x: ast.literal_eval(x))
df_rm['floorplans'] = df_rm['floorplans'].apply(lambda x: [item['url'] for item in x])

df_rm['nearest_airports'] = df_rm['nearest_airports'].str.replace('[', '').str.replace(']', '')

df_rm['features'] =df_rm['features'].str.replace('[', '').str.replace(']', '')

df_rm['description'] = df_rm['description'].apply(lambda x: BeautifulSoup(x, 'html.parser').get_text())
```

For proper identification of rightmove data, we prefixed all the columns with RM_. Also we created a new column ‘Coordinate’ by concatenating the longitude and latitude columns. This is column is suppose to be unique to each property listing on the assumption that no two points should have exactly the same latitude and longitude.

```
#Prefixing all the columns of df_rm with RM
df_rm = df_rm.add_prefix('RM_')

#Creating a unique column called Coordinate by combining the latititude and longitude with semicolon
df_rm['Coordinate'] = df_rm['RM_latitude'].astype(str) + ': ' + df_rm['RM_longitude'].astype(str)

#making the coordinate column the first column
df_rm = df_rm[['Coordinate'] + df_rm.columns[:-1].tolist()]
```

The final cleaned and prefixed dataframe df_rm is shown below.

	Coordinate	RM_id	RM_Status	RM_title	RM_description	RM_property_type	RM_bedrooms	RM_bathrooms	RM_price	RM_price_sqft	...	R
0	51.5166: -0.080557	74504326	Available	1 bedroom apartment for sale in One Bishopsgat...	This 21st floor, one bedroom apartment is well...	Apartment	1.0	1.0	£1,325,000	£2086.61 per sq. ft.	...	
1	51.514636: -0.109152	133932536	Available	Studio apartment for sale in Crane Court,	Recently refurbished studio in Central London....	Apartment	NaN	1.0	£450,000	NaN	...	

Working on Zoopla Data:

The csv data zp_properties_csv was used to create a dataframe called df_zp which has 2000 rows and 16 columns.

```
#READING THE SCRAPED CSV FILE INTO A DATAFRAME CALLED df_zp
df_zp=pd.read_csv('results/zp_properties_all.csv')
```

```
1 df_zp
```

	id	title	description	url	price	type	date	category	section	features	floor_plan
0	46020849	2 bed flat for sale	Sandra Davidson are please to offer an opportu...	/for-sale/details/46020849/	£1,439,000	flat	2017-12-18T20:03:10	residential	for-sale	['Two Bedrooms', 'Open Plan Living/Kitchen are...	{'filename': None, 'caption': None}
1	46020848	1 bed flat for sale	Sandra Davidson are please to offer an opportu...	/for-sale/details/46020848/	£1,102,000	flat	2017-12-18T20:03:10	residential	for-sale	['Double Bedroom', 'Open Plan Living/Kitchen',...	{'filename': None, 'caption': None}

The dataframe contains some columns such as ‘details and ‘agent’ columns which contain several property information. We then needed to extract out these useful data and used them to form new columns. We also performed cleaning exercise on the columns in order to remove all accompanying html tags and symbols that are not useful. We also created some useful columns such as postcode by joining two columns. We dropped the columns from where data have been extracted and finally removed all the duplicate rows.

```
#CREATING DESIRED COLUMNS AND CLEANING UP THE SCRAPED DATA

from bs4 import BeautifulSoup
#deleting irrelevant columns
df_zp = df_zp.drop(['photos', 'floor_plan','url', 'section'], axis=1)

#converting the coordinates to separate columns
import ast
df_zp['coordinates'] = df_zp['coordinates'].apply(ast.literal_eval)
df_zp['latitude'] = df_zp['coordinates'].apply(lambda x: x['lat'])
df_zp['longitude'] = df_zp['coordinates'].apply(lambda x: x['lng'])

#Extracting out and creating different columns from the agency column
df_zp['agency'] = df_zp['agency'].apply(ast.literal_eval)
df_zp['Agent_Name'] = df_zp['agency'].apply(lambda x: x['name'])
df_zp['Agent_Phone'] = df_zp['agency'].apply(lambda x: x['phone'])
df_zp['Agent_address'] = df_zp['agency'].apply(lambda x: x['address'])
df_zp['Agent_Post_code'] = df_zp['agency'].apply(lambda x: x['postcode'])

#Extracting out and creating different columns from the details column
df_zp['details'] = df_zp['details'].apply(ast.literal_eval)
df_zp['Bedrooms'] = df_zp['details'].apply(lambda x: x['numBeds'])
df_zp['Bathrooms'] = df_zp['details'].apply(lambda x: x['numBaths'])
df_zp['Receptions'] = df_zp['details'].apply(lambda x: x['numRecepts'])
df_zp['address'] = df_zp['details'].apply(lambda x: x['displayAddress'])
df_zp['address'] = df_zp['address'].apply(lambda x: BeautifulSoup(x, 'html.parser').get_text())
df_zp['Area'] = df_zp['details'].apply(lambda x: x['areaName'])
df_zp['Outcode']=df_zp['details'].apply(lambda x: x['outcode'])
df_zp['Incode']=df_zp['details'].apply(lambda x: x['incode'])
df_zp['Region'] = df_zp['details'].apply(lambda x: x['regionName'])
df_zp['Tenure_type'] = df_zp['details'].apply(lambda x: x['tenure'])
df_zp['zindex'] = df_zp['details'].apply(lambda x: x['zindex'])
df_zp['Furnished_Status'] = df_zp['details'].apply(lambda x: x['furnishedState'])
```

To easily identify zoopla data, we prefixed all zoopla columns with ‘ZP_’. Also, we created a new column called Coordinate which contains the latitude and longitude concatenated with a colon to form a unique column as it is assumed that every listing should have a unique combination of latitude and longitude.

```
#PREPROCESSING OF SCRAPED ZOOPLA DATA

#Prefixing all the columns of df_rm with ZP
df_zp = df_zp.add_prefix('ZP_')

#Creating a unique column called Coordinate
df_zp['Coordinate'] = df_zp['ZP_latitude'].astype(str) + ': ' + df_zp['ZP_longitude'].astype(str)

#making the coordinate column the first column
df_zp = df_zp[['Coordinate'] + df_zp.columns[:-1].tolist()]
```

The final zoopla dataframe df_zp is shown

	Coordinate	ZP_id	ZP_title	ZP_description	ZP_price	ZP_type	ZP_date	ZP_category	ZP_features	ZP_nearby	...	ZP_Bedroon
0	51.522098: -0.078848	46020849	2 bed flat for sale	Sandra Davidson are please to offer an opportu...	£1,439,000	flat	2017-12-18T20:03:10	residential	'Two Bedrooms', 'Open Plan Living/Kitchen area...	Shoreditch High Street:0.1, The Lyceum:0.2, Lo...	...	2
1	51.522098: -0.078848	46020848	1 bed flat for sale	Sandra Davidson are please to offer an opportu...	£1,102,000	flat	2017-12-18T20:03:10	residential	'Double Bedroom', 'Open Plan Living/Kitchen', ...	Shoreditch High Street:0.1, The Lyceum:0.2, Lo...	...	1

Data Blending

The next step was data blending by merging the two dataframes together on a common column which in this case is the 'Coordinate' column to form a new dataframe named df_combined.

```
#Merging both dataframes on common column which is 'Coordinate'
df_combined = pd.merge(df_rm, df_zp, on='Coordinate', how='outer')
```

We set the '**how'** clause to outer to ensure that rows in which the Coodinate columns do not match will be included in the dataset.

Next we arranged all the columns in the df_combined in such a way that similar columns from each dataset are adjacent to each other.

```
# Reordering the columns
column_order = ['Coordinate','RM_id', 'ZP_id', 'RM_Status', 'RM_Listinghistory', 'ZP_date', 'RM_title', 'ZP_title', 'RM_

# Reorder the columns
df_combined = df_combined[column_order]
```

Next, for each pair of similar columns from df_zp and df_rm, we selected one of the columns and dropped the other if both columns are available, otherwise we select only the one that is available. For instance, Zp_date and Rm_listing history mean the listing date for zoopla and rightmove respectively. if both are available, we select the one from zoopla and set it to a new column called 'Listed_date' with the code below

```
df_combined['Listed_date'] = df_combined['ZP_date'].fillna(df_combined['RM_Listinghistory'])
```

Also, for title, if both are available, we select the one from rightmove and set it to a new colum called Title with the code below.

```
df_combined['Title'] = df_combined['RM_title'].fillna(df_combined['ZP_title'])
```

For columns without corresponding field from both rightmove and zoopla, we simply created a new column and filled it with the data from the corresponding dataframe using the id column as a reference.

```
df_combined.loc[~df_combined['RM_id'].isna(), 'Photos'] = df_combined.loc[~df_combined['RM_id'].isna(), 'RM_photos']
```

```
#creating new columns with value from either rightmove or zoopla data
df_combined['Listed_date'] = df_combined['ZP_date'].fillna(df_combined['RM_Listinghistory'])
df_combined.loc[~df_combined['RM_id'].isna(), 'Status'] = df_combined.loc[~df_combined['RM_id'].isna(), 'RM_Status']
df_combined['Title'] = df_combined['RM_title'].fillna(df_combined['ZP_title'])
df_combined['Price'] = df_combined['RM_price'].fillna(df_combined['ZP_price'])
df_combined.loc[~df_combined['RM_id'].isna(), 'Price_sqft'] = df_combined.loc[~df_combined['RM_id'].isna(), 'RM_price_sq
df_combined.loc[~df_combined['ZP_id'].isna(), 'Category'] = df_combined.loc[~df_combined['ZP_id'].isna(), 'ZP_category']
df_combined['Property_type'] = df_combined['RM_property_type'].fillna(df_combined['ZP_type'])
df_combined['Description'] = df_combined['RM_description'].fillna(df_combined['ZP_description'])
df_combined['Features'] = df_combined['RM_features'].fillna(df_combined['ZP_features'])
df_combined['Bedrooms'] = df_combined['RM_bedrooms'].fillna(df_combined['ZP_Bedrooms'])
df_combined['Bathrooms'] = df_combined['RM_bathrooms'].fillna(df_combined['ZP_Bathrooms'])
df_combined.loc[~df_combined['ZP_id'].isna(), 'Receptions'] = df_combined.loc[~df_combined['ZP_id'].isna(), 'ZP_Receptio
df_combined.loc[~df_combined['RM_id'].isna(), 'Photos'] = df_combined.loc[~df_combined['RM_id'].isna(), 'RM_photos']
df_combined.loc[~df_combined['RM_id'].isna(), 'Floorplans'] = df_combined.loc[~df_combined['RM_id'].isna(), 'RM_floorpla
df_combined.loc[~df_combined['RM_id'].isna(), 'Nearest_stations'] = df_combined.loc[~df_combined['RM_id'].isna(), 'RM_ne
df_combined.loc[~df_combined['ZP_id'].isna(), 'Nearby_landmarks'] = df_combined.loc[~df_combined['ZP_id'].isna(), 'ZP_ne
df_combined.loc[~df_combined['RM_id'].isna(), 'Nearest_airports'] = df_combined.loc[~df_combined['RM_id'].isna(), 'RM_ne
df_combined['Latitude'] = df_combined['RM_latitude'].fillna(df_combined['ZP_longitude'])
df_combined['Longitude'] = df_combined['RM_longitude'].fillna(df_combined['ZP_Tenure_type'])
df_combined['Tenure_type'] = df_combined['RM_Tenure_type'].fillna(df_combined['ZP_title'])
df_combined.loc[~df_combined['RM_id'].isna(), 'Lease_year_left'] = df_combined.loc[~df_combined['RM_id'].isna(), 'RM_Lea
df_combined.loc[~df_combined['RM_id'].isna(), 'CouncilTax_Status'] = df_combined.loc[~df_combined['RM_id'].isna(), 'RM_C
df_combined.loc[~df_combined['RM_id'].isna(), 'Annual_Ground_Rent'] = df_combined.loc[~df_combined['RM_id'].isna(), 'RM_
df_combined.loc[~df_combined['RM_id'].isna(), 'Annual_Service_Charge'] = df_combined.loc[~df_combined['RM_id'].isna(), '
df_combined.loc[~df_combined['RM_id'].isna(), 'Council_Tax_Band'] = df_combined.loc[~df_combined['RM_id'].isna(), 'RM_co
df_combined.loc[~df_combined['RM_id'].isna(), 'Domestic_Rates'] = df_combined.loc[~df_combined['RM_id'].isna(), 'RM_dome
df_combined['Address'] = df_combined['ZP_address'].fillna(df_combined['RM_address'])
df_combined.loc[~df_combined['ZP_id'].isna(), 'Area'] = df_combined.loc[~df_combined['ZP_id'].isna(), 'ZP_Area']
df_combined.loc[~df_combined['ZP_id'].isna(), 'Region'] = df_combined.loc[~df_combined['ZP_id'].isna(), 'ZP_Region']
df_combined.loc[~df_combined['ZP_id'].isna(), 'Postcode'] = df_combined.loc[~df_combined['ZP_id'].isna(), 'ZP_Postcode']
df_combined.loc[~df_combined['ZP_id'].isna(), 'Zindex'] = df_combined.loc[~df_combined['ZP_id'].isna(), 'ZP_zindex']
df_combined['Agent_name'] = df_combined['RM_Agent_Name'].fillna(df_combined['ZP_Agent_Name'])
df_combined['Agent_Phone_no'] = df_combined['RM_Agent_No'].fillna(df_combined['ZP_Agent_Phone'])
df_combined['Agent_address'] = df_combined['RM_Agent_Address'].fillna(df_combined['ZP_Agent_address'])
df_combined.loc[~df_combined['ZP_id'].isna(), 'Agent_post_code'] = df_combined.loc[~df_combined['ZP_id'].isna(), 'ZP_Age
df_combined.loc[~df_combined['ZP_id'].isna(), 'Furnished_status'] = df_combined.loc[~df_combined['ZP_id'].isna(), 'ZP_Fu
```

We created the final dataframe '**df_combined_final'** by dropping the rightmove and zoopla prefixed columns leaving only the new columns formed by merging the data from both df_rm and df_zp.

```
#We can now drop used individual columns and some unuseful columns

df_combined_final = df_combined.drop(['Coordinate','RM_id','ZP_id','RM_Status','RM_Listinghistory', 'ZP_date', 'RM_title

#droping the pound sign from price and price per sq
df_combined_final['Price'] = df_combined_final['Price'].str.replace('£', '')

df_combined_final['Price_sqft'] = df_combined_final['Price_sqft'].str.replace('£', '').str.replace('per sq. ft.', '')
```

```
#THE FINAL DATA FRAME
df_combined_final
```

	Listed_date	Status	Title	Price	Price_sqft	Category	Property_type	Description	Features	Bedrooms	...	Council_Tax_
0	2019-10-03T16:03:59	Available	1 bedroom apartment for sale in One Bishopsgat...	1,325,000	2086.61	residential	Apartment	This 21st floor, one bedroom apartment is well...	"Unique views of some of London's most iconic ...	1.0	...	
1	2023-04-21T12:44:44	Available	Studio apartment for sale in Crane Court, Lond...	450,000	NaN	residential	Apartment	Recently refurbished studio in Central London....	'Recently Refurbished', 'Centrally Located', '...	NaN	...	

Finally, the **df_combined_final** dataframe was converted to a csv file named **final_dataset.csv**

```
#Coverting the final dataframe to a csv file called final_dataset
df_combined_final.to_csv('results/final_dataset.csv', index=False)
```

Conclusion

The final dataset contains 2779 property listing in the City of London and Barking and Dagenham scraped from rightmove and zoopla websites. The scraped data from both websites were merged on a common column Coordinate which contains the combination of latitude and longitude of each property. We have assumed that 'Coordinate' is unique to each property, however, from inspection it is not so which means that there will be duplicity in the data. However, there is the possibility of getting more property features and/or feature combination that can create uniqueness for every property listing. This will will be followed up on.

Althouh we have only scraped properties for city of London and barking and Dagenham in this project, the scraping codes used were designed to be both scalable and reusable. With the codes, we can scrape properties from every part of UK and we can also produce any required property feature.