DREXEL UNIVERSITY

DSCI591
DATA ACQUISITION &
PRE-PROCESSING REPORT

REAL ESTATE TRENDS & INVESTIGATING RELATIONSHIPS WITH COVID-19

Lawrence Love
Gustavo Ferreira
Yan Li
Frank Zhao

2020

CONTENTS

IDENTIFYING DATA

DATA SOURCES |

HOUSING DATA

The data source for the housing data is from the website Realtor.com. On this website, you can find houses for sale, houses for rent, recently sold houses, etc. The reason we choose this data source is we want to gain high-level insight into various real estate attribute statistics. Also, we want to know how the corona virus affects the housing prices. By using this website, we can get all the information that we need to do the analysis like price, address, bedrooms, bathrooms, nearby schools, property type, neighborhood and so on.

https://realtor.p.rapidapi.com/properties/v2/list-for-sale
https://realtor.p.rapidapi.com/properties/v2/detail
https://realtor.p.rapidapi.com/properties/v2/list-for-rent

COVID-19 DATA

The data source for the COVID-19 data comes from opendataphilly.org. As the project progresses, and after we consider the full extent of our limitations, we may consider gathering additional COVID-19 data from sources such as: data.cityofchicago.org, nyc.gov, and other sites for major cities offering COVID-19 data.

Opendataphilly.org was chosen because it (and the other sources like it) are local government endorsed and hosted data. Also, the data is easily accessible. The site contains data on COVID-19 hospitalizations, cases, and deaths by age, race, sex, date and zip code. The data can help us get a broad understanding of the demographic nature of the virus and allow us the ability to combine the zip code data from the housing data source so see what inferences we can make about the virus' effect on the real estate market.

*https://phl.carto.com/api/v2/sql?q=SELECT*FROMcovid_hospitalizations_by_zip*
*https://phl.carto.com/api/v2/sql?q=SELECT * FROM covid_deaths_by_zip*
*https://phl.carto.com/api/v2/sql?q=SELECT * FROM covid_cases_by_zip*

ACQUISITION PROCESS

COVID-19 |

The initial data acquisition process for the COVID data was a simple browsing of what the stie had to offer and downloading the CSV files we found to be valuable. After acquiring this data, it was loaded into a pandas dataframe.

Later, for zip code data, the opendataphilly.org API was used (see URLs above) to acquire the data through utilization of the request function and output to JSON format. Then the data was simply converted to a dataframe using the pandas **json_normalize()** functon. In the revisions to come later, we intend to revisit our initial data acquisition for the purpose of writing code to acquire all of the data through the API, rather than browse the site and download CSV files.

PROPERTIES |

The dataset for both house-for-sale and house-for-rent is acquired through rapid.api, an open-source API. There is no dataset ready for download. We utilize the **requests** function for data acquisition and output to JSON format, **response.json()**. Then, we create an empty list **convert_list**, call **property** in each **response.json()**, which contains the info of a property, transform the result and append it to **convert_list**. In the end, we concatenate all rows into a data frame. Those data come from a single API source, which makes the work easy to process.

INTEGRATION

Given that both data sources have zip code data, we can easily integrate the two sources with each other.

ISSUES

REQUEST LIMITATION

This is the biggest challenge in our project, that rapid.api limits the number of requests to a 500/month quota for basic plan. To increase the number of requests, we must upgrade our subscription plan at higher prices, which is not practical for our project. There are 9562 instances in **house_for_sale** dataset and 5277 instances in **house_for_rent**, which means we have used 275 quotas for a single subscription. To get details for each property, for example, historical prices for a property, descriptions for sentiment analysis, etc., we have to request almost 15k, which is enormous. Our current solution is to apply more accounts for the free subscription plan, which will takes days to finish all the runs. Due to data acquisition limitations, we may change our project data from national to regional in Philadelphia, PA.

* Aside from what appears to be collection issues from the data source, there were no acquisition issues for the COVID-19 data.

DATA-PROCESSING

PROPERTIES |

DUPLICATED DATA

There might be some glitch that some **requests** call duplicate data. Our solution is removing the row, which has duplicate **property_id**.

RENAME FEATURE COLUMNS

After converting the list to data frame, we see that some features have list of lists or dictionaries. Such as **address**, **branding**, **building_size**, **agents**, **lot_size**. It's necessary to change the column names once we extract the data. We also create new features for those details we extract. The original data set has 18 features, while after renaming, we have 25 features.

MISSING DATA

For **house_for_sale**, seven features have missing values:

prop_sub_type: 2512 missing data. This feature describes if a property is a duplex, triplex, townhouse, or condos. For our logistics regression in Phase two, we may drop the rows with missing data in this feature.

baths_full: 1760 missing data. This feature describes the number of full bathrooms a property has. Some properties are lands that are not under construction, which may result in missing data. Or a property has no full bathroom, which is for another explanation. We can apply `.fillna()` to fill those missing data with 0s.

baths_half: 6168 missing data. This feature describes the number of half bathrooms a property has. It's similar to baths_full`, and we fill those missing data with 0s.

beds: 1182 missing data. This feature describes the number of bedrooms a property has. Some properties are studios that have no bedrooms. In such a case, we can fill missing data with 0s.

building_size and **lot_size**: the former has 1381 missing data, while the latter has 1687 missing data. We can fill missing data with 0s.

agent_id and **brand_name**: the former has 94 missing data, while the latter has 80 missing data. We would delete these two features if necessary or delete rows with missing values in these two features.

ORDERING

- **Age** – the age data set intially shows the ages scrambled and not in any order. The reorder this data into proper chronology makes it easier to understand

- **Race** - **count** - the count column needed to be set in descending order for best observation

- **Date** – the dates datasets did not appear to be in any particular order at all

DATE-TIME OBJECT

In the date datasets, various dates are listed; sometimes day to day, and sometimes a week apart. To consolidate, the date instances were converted to date-time objects, and a new 'month' column was created to easily make chronological observations.

CASING INCONSISTENCIES

For **house_for_sale**, seven features have missing values:

Many responses in datasets would sometimes be in all caps while the other responses were in title casing. The title() function in pandas was often used to keep casing consistent.

APPENDIX

Code example:
1. Requesting the properties data, updating the offset value by 200 at each request.

```
#The requests. Since the results come in batches of 200 properties, the offset must be updated at every request.
#The responses are converted to JSON; then the dictionaries are appended to a list containing all the listings.

url = "https://realtor.p.rapidapi.com/properties/v2/list-sold"

headers = {
    'x-rapidapi-host': "realtor.p.rapidapi.com",
    'x-rapidapi-key': "c524e09f88msh80e8e474a4c48d7p1f0febjsn6a9ae0584239"
    }


all_properties = []
total_requests = 0

offset = 0

querystring = {"city":"Philadelphia",
               "offset":"0",
               "state_code":"PA",
               "limit":"200",
               "prop_type":"condo",
               "sort":"sold_date"}

while True:
    querystring['offset'] = str(offset)

    try:
        response = requests.request("GET", url, headers=headers, params=querystring)
        response_dict = response.json()
        all_properties += response_dict['properties']

        offset += 200
        total_requests += 1
        time.sleep(0.3)
        print('Total requests:', str(total_requests))

    except Exception as e:
        if offset >= 10000:
            print('Offset limit reached')
            break
        else:
            print('Unsuccessful Request')
            print(e)
```

2. Transforming housing data (JSON) to pandas dataframe

```
## convert response to pandas df

def process_response(response_json):
    '''
    This function is to convert each request result to a dataframe.

    1. create an empty list
    2. loop for each response and get details from key 'properties'
    3. convert details to df
    4. append single df to list
    5. concat the list to one df

    '''

    # empty list
    convert_list=[]
    for col in response_json['properties']:

        # convert details to dataframe
        single_df = pd.DataFrame.from_dict(col, orient='index').T

        # append to list
        convert_list.append(single_df)

    # concat to a whole df, null for missing vals
    return pd.concat(convert_list, axis = 0, ignore_index=True, sort=False)
```

Sample Data:

1. Raw data for **house_for_sale**

	property_id	prop_type	prop_sub_type	address	branding	prop_status	price	baths_full	baths	beds	building_size	agents
0	M4046594895	condo	duplex_triplex	{'city': 'Philadelphia', 'line': '1516 N 62nd ...	{'listing_office': {'list_item': {'name': 'Arc...	for_sale	249900	3.0	3	6.0	{'size': 1632, 'units': 'sqft'}	[{'primary': True, 'advertiser_id': '1291281',...
1	M3939384476	condo	townhomes	{'city': 'Philadelphia', 'line': '6102 Reedlan...	{'listing_office': {'list_item': {'name': 'Vih...	for_sale	116800	1.0	1	3.0	{'size': 1092, 'units': 'sqft'}	[{'primary': True, 'advertiser_id': '347285', ...
2	M4036371277	condo	townhomes	{'city': 'Philadelphia', 'line': '5703 N 13th ...	{'listing_office': {'list_item': {'name': 'Pre...	for_sale	215000	1.0	2	3.0	{'size': 1360, 'units': 'sqft'}	[{'primary': True, 'photo': None, 'name': 'Kev...
3	M3553029343	single_family	NaN	{'city': 'Philadelphia', 'line': '1009 Rhawn S...	{'listing_office': {'list_item': {'name': 'Re/...	for_sale	394800	1.0	2	3.0	{'size': 1856, 'units': 'sqft'}	[{'primary': True, 'advertiser_id': '4759', 'i...
4	M3649199107	condo	townhomes	{'city': 'Philadelphia', 'line': '3850 N Bouvi...	{'listing_office': {'list_item': {'name': 'Re/...	for_sale	130000	1.0	2	3.0	{'size': 1180, 'units': 'sqft'}	[{'primary': True, 'advertiser_id': '391546', ...
...	...	...	...	...	...	...	...	...	...	...	...	...
9757	M3400474681	condo	townhomes	{'city': 'Philadelphia', 'line': '2077 Bridge ...	{'listing_office': {'list_item': {'name': 'Re/...	for_sale	94900	1.0	1	4.0	{'size': 1296, 'units': 'sqft'}	[{'primary': True, 'photo': None, 'name': ''}]

2. Raw data for **house_for_rent**

	property_id	listing_id	prop_type	list_date	last_update	year_built	listing_status	beds	branding	baths_full	price_reduced
0	R9220820530	2920289582	condo	2020-08-20T21:07:02.000Z	2020-09-16T13:46:59.000Z	1900.0	active	2	{'listing_office': {'list_item': {'name': 'BHH...	1.0	202 16T17:50:36
1	R9722862130	2921389874	condo	2020-09-16T17:15:26.000Z	2020-09-16T13:12:24.000Z	1960.0	active	0	{'listing_office': {'list_item': {'name': 'By ...	1.0	
2	R9175211338	2921270830	townhome	2020-09-13T02:05:48.000Z	2020-09-14T21:30:22.000Z	1925.0	active	1	{'listing_office': {'list_item': {'name': 'Uni...	1.0	
3	R4647948039	2921177051	condo	2020-09-10T20:09:38.000Z	2020-09-12T10:18:11.000Z	1900.0	active	2	{'listing_office': {'list_item': {'name': 'Kel...	2.0	
6288	R3304893346	2919464556	townhome	2020-08-05T17:46:50.000Z	2020-08-05T13:45:31.000Z	NaN	active	2	{'listing_office': {'list_item': {'name': "Kur...	2.0	

3. Processed data for **house_for_sale**

	property_id	prop_type	prop_sub_type	prop_status	price	baths_full	baths	beds	last_update	photo_count	...	state_code	county	
0	M4046594895	condo	duplex_triplex	for_sale	249900	3.0	3	6.0	2020-10-13T17:54:05Z	9	...	PA	Philadelphia	39.97
1	M3939384476	condo	townhomes	for_sale	116800	1.0	1	3.0	2020-10-13T18:18:18Z	7	...	PA	Philadelphia	39.92
2	M4036371277	condo	townhomes	for_sale	215000	1.0	2	3.0	2020-10-13T17:24:20Z	35	...	PA	Philadelphia	40.03
3	M3553029343	single_family	NaN	for_sale	394800	1.0	2	3.0	2020-10-13T17:11:54Z	123	...	PA	Philadelphia	40.07
4	M3649199107	condo	townhomes	for_sale	130000	1.0	2	3.0	2020-10-13T17:02:13Z	33	...	PA	Philadelphia	40.01

5 rows × 26 columns

4. Build dataframe for deaths by zip divided by median listing price for corresponding zip.

```
death_zip['zip_code'] = death_zip['zip_code'].astype(int)
death_merge = death_zip.merge(price_median_by_zip, how = 'inner', on = 'zip_code')
```

```
## Death rate by median listing price
death_rate_by_zip_median = []

for i in range(0, len(death_merge['count'])):
    rate = round(death_merge['count'][i] / death_merge['median_price'][i], 5) * 100
    death_rate_by_zip_median.append(rate)

death_merge['death_rate_by_zip_median'] = death_rate_by_zip_median
death_merge = death_merge.sort_values('death_rate_by_zip_median', ascending=False).reset_index(drop=True)
```

```
death_merge.head()
```

	cartodb_id	the_geom	the_geom_webmercator	zip_code	covid_outcome	count	etl_timestamp	median_price	death_rate_by_zip_median
0	2	None	None	19132	DIED	39	2020-10-24T17:20:02Z	50000.0	0.078
1	38	None	None	19140	DIED	61	2020-10-24T17:20:02Z	80000.0	0.076
2	25	None	None	19131	DIED	97	2020-10-24T17:20:02Z	135000.0	0.072
3	34	None	None	19144	DIED	97	2020-10-24T17:20:02Z	150000.0	0.065
4	16	None	None	19104	DIED	77	2020-10-24T17:20:02Z	134000.0	0.057

5. The month by month test results analysis.

```
# Add months to the data set and group by month to get a month by month analysis.
months = []
for i in cases_date['collection_date']:
    months.append(datetime.datetime.strptime(i, "%Y-%m-%d").month)

month_names = []
for i in months:
    month_names.append(calendar.month_abbr[i])

cases_date['month'] = month_names

new_cases_date = cases_date.groupby(['month', 'test_result'], as_index=False).sum()
new_cases_date.drop(['the_geom', 'the_geom_webmercator'], axis=1, inplace=True)
new_cases_date
```

	month	test_result	count
0	Apr	negative	25087
1	Apr	positive	12906
2	Aug	negative	83500
3	Aug	positive	3631
4	Jul	negative	74585
5	Jul	positive	4314
6	Jun	negative	52372
7	Jun	positive	3474
8	Mar	negative	6705
9	Mar	positive	3202
10	May	negative	43253
11	May	positive	7059
12	Oct	negative	24756
13	Oct	positive	1035
14	Sep	negative	89386
15	Sep	positive	2974

Data Definition

Variable	Description	DataType
property_id	Unique Id for each house	Object
prop_type	Type of each house	Object
prop_sub_type	Subtype of each house	Object
prop_status	Status of each house	Object
price	Price of each house	Integer
baths_full	Number of full baths	Float
baths	Sum of full baths and half baths	Integer
beds	Number of bedrooms	Float
last_update	Latest update date	Object
photo_count	Number of photos of each house	Integer
page_no	The page number which the house is listed	Integer
baths_half	Number of half baths	Float
city	City name of each house	Object
line	Address of each house	Object
postal_code	Postal code of each house	Integer
state_code	Abbreviation of the state name	Object
county	County name of each house	Object
lat	Latitude of each house	Float
lon	Longitude of each house	Float
neighborhood_name	Name of the neighborhood	Object
buiding_size(sqft)	Size of each house in square feet	Float
lot_size	Size of parking lot of each house in square feet	Float
agent_id	Unique Id for each agent	Float
agent_name	Name of each agent	Object
brand_name	Company name of each real estate agent	Object

Table of Contributions

Section	Writing	Editing
Data Sources	L. Love, Y. Li	G. Ferreira
Data Pre-Processing	L. Love, F. Zhao	G. Ferreira
Appendix	L. Love, Y. Li, F. Zhao	G. Ferreira