etl

July 10, 2023

1 HDB Resale Price Predictor & Visualisation

This project aims to create a data pipeline with the help of availale APIs (Data.gov.sg and OneMap) to build a web-based application for 1. HDB Price visualisation 2. HDB Price prediction

The prototype aims to read latest data directly from data.gov.sg and perform ETL (Extract, Transform, and Load) to a local/web database of choice.

[74]:
```
import requests
import requests_cache
import numpy as np
import pandas as pd
import json
import logging
import time
from requests.exceptions import HTTPError
from pprint import pprint
from functools import wraps
from geopy.distance import geodesic as GD
```

1.1 Data Wrangling Contents

1. API call data
2. Data Wrangling
3. Feature Engineering

1.2 1. Getting the data through API call

1.2.1 Wrapper functions

- To time function calls
- To error handle HTTPerrors and other Exceptions
- To cache API calls

[75]:
```
logging.basicConfig(filename='wrangling.log', filemode='a', format='%(asctime)s␣
↪- %(name)s - %(levelname)s - %(message)s')
logging.warning(f"{'-'*20}New run started {'-'*100}")

# Enable caching
```

```
session = requests_cache.CachedSession('F:\python_stuff\hdb_project_cache')
```

[76]:
```
# Wrapper for timing function calls:
def timeit(func):
    '''
    Wrapper to time function call
    '''
    @wraps(func)
    def timeit_wrapper(*args, **kwargs):
        '''
        *args and **kwargs here allow parameters for the original function to␣
 ↪be taken in
        and passed to the function contained in the wrapper.
        '''
        current_time = time.strftime("%H:%M:%S", time.localtime())
        start = time.perf_counter()
        result = func(*args, **kwargs)
        end = time.perf_counter()
        time_taken = end-start
        print(f'{func.__name__}() called at \t{current_time} \texecution time:␣
 ↪{time_taken:.4f} seconds')
        logging.info(f'{func.__name__}() called at \execution time:␣
 ↪{time_taken:.4f} seconds')
        return result
    return timeit_wrapper

def error_handler(func, max_attempts=3, delay=120):
    '''
    Wrapper to catch and handle errors
    '''
    @wraps(func)
    def error_handler_wrapper(*args, **kwargs):
        '''
        *args and **kwargs here allow parameters for the original function to␣
 ↪be taken in
        and passed to the function contained in the wrapper, without needed to␣
 ↪declare them in the wrapper function.
        '''
        for i in range(max_attempts):
            try:
                result = func(*args, **kwargs)
            except HTTPError as err:
                logging.error(f'{func.__name__}() encountered {err}')
                # Raise exception if we reach max tries
                if i == max_attempts:
                    raise HTTPError(f'Exceeded max tries of {max_attempts}')
                print(f'{func.__name__}() encountered {err}')
```

```
                # err.response gives us the Response object from requests␣
↪module, we can call .status_code to get the code as int
                if err.response.status_code == 429:
                    print(f'Sleeping for {delay} seconds', end = '\t')
                    time.sleep(delay)
                    print('Retrying...', end='\t')
            except Exception as err:
                logging.error(f'{func.__name__}() encountered {err}')
                print(f'{func.__name__}() encountered {err}')
                break
            else:
                return result
    return error_handler_wrapper
```

1.2.2 Details for Data.gov.sg API call can be found at

https://data.gov.sg/dataset/ckan-datastore-search

[77]:
```
@timeit
@error_handler
def get_token(location: str):
    '''
    Function to check if API token is still valid and updates API token if␣
↪outdated
    ##Parameters
        location: filepath (str)
    Returns API token : str
    '''
    with open(location, 'r+') as fp:
        file = fp.read()
        data = json.loads(file)
        response = requests.post("https://developers.onemap.sg/privateapi/auth/
↪post/getToken", data=data)
        token = response.json()
        if token['access_token'] != data['access_token']:
            print(f"New token found")
            data['access_token'] = token['access_token']
            data['expiry_timestamp'] = token['expiry_timestamp']
            fp.seek(0)
            json.dump(data, fp = fp, indent=4)
            print('Updated token json')
            data = json.loads(file)
        return data['access_token']

@timeit
@error_handler
```

```
def datagovsg_api_call(url: str, sort: str = 'month desc', limit: int = 100,
                       months:list =[1,2,3,4,5,6,7,8,9,10,11,12],
                       years:list =["2022"]) -> pd.DataFrame:
    '''
    Function to build the API call and construct the pandas dataframe
    ## Parameters
    url: str
        url for API, with resource_id parameters
    sort: str
        field, by ascending/desc, default by Latest month
    limit: int
        maximum entries (API default by OneMap is 100, if not specified)
    months: list
        months desired, int between 1-12
    years: list
        months desired , int
    Returns Dataframe of data : pd.DataFrame
    '''
    month_dict = '{"month":['
    for year in years:
        for month in months: # months 1-12
            month_dict = month_dict + f'"{year}-{str(month).zfill(2)}", '
    month_dict = month_dict[:-2] # Cancel out extra strings <, >
    month_dict = month_dict + ']}'
    url = url+f'&sort={sort}&filters={month_dict}'
    if limit: # API call's default is 100 even without specifying
        print(f'Call limit : {limit}')
        url = url+f'&limit={limit}'
    pprint(f'API call = {url}')
    response = requests.get(url)
    response.raise_for_status()
    data = response.json()
    df = pd.DataFrame(data['result']['records'])
    return df
```

[78]:
```
df = datagovsg_api_call('https://data.gov.sg/api/action/datastore_search?
 ↪resource_id=f1765b54-a209-4718-8d38-a39237f502b3',
                        sort='month desc',
                        limit = 1000000,
                        months = [7],
                        years=[2023])
df
```

```
Call limit : 1000000
('API call = '
 'https://data.gov.sg/api/action/datastore_search?resource_id=f1765b54-a209-4718
-8d38-a39237f502b3&sort=month '
 'desc&filters={"month":["2023-07"]}&limit=1000000')
```

```
datagovsg_api_call() called at  23:41:08        execution time: 0.9239 seconds
```

[78]:

	town	flat_type	flat_model	floor_area_sqm	street_name
0	ANG MO KIO	4 ROOM	New Generation	93	ANG MO KIO AVE 5
1	BEDOK	4 ROOM	New Generation	92	BEDOK STH RD
2	BEDOK	5 ROOM	Improved	112	BEDOK CTRL
3	BEDOK	5 ROOM	Improved	118	BEDOK NTH AVE 3
4	BEDOK	5 ROOM	Improved	122	BEDOK STH AVE 3
..	…	…	…	…	…
390	BEDOK	4 ROOM	Model A	93	BEDOK NTH ST 4
391	BEDOK	4 ROOM	Model A	93	BEDOK NTH ST 4
392	BEDOK	4 ROOM	New Generation	92	BEDOK RESERVOIR RD
393	BEDOK	4 ROOM	New Generation	91	BEDOK RESERVOIR RD
394	BEDOK	4 ROOM	New Generation	98	BEDOK STH AVE 2

	resale_price	month	remaining_lease	lease_commence_date
0	538000	2023-07	56 years 02 months	1980
1	470000	2023-07	53 years 07 months	1978
2	985000	2023-07	85 years 11 months	2010
3	635000	2023-07	56 years 01 month	1980
4	740000	2023-07	61 years 07 months	1985
..	…	…	…	…
390	788000	2023-07	94 years 06 months	2018
391	728888	2023-07	94 years 06 months	2018
392	505000	2023-07	61 years 02 months	1985
393	500000	2023-07	56 years 09 months	1981
394	521000	2023-07	53 years 09 months	1978

	storey_range	_id	block
0	10 TO 12	156373	523
1	01 TO 03	156394	71
2	10 TO 12	156395	219B
3	01 TO 03	156396	406
4	04 TO 06	156397	156
..	…	…	…
390	10 TO 12	156389	188A
391	04 TO 06	156390	186A
392	10 TO 12	156391	110
393	07 TO 09	156392	623
394	13 TO 15	156393	33

[395 rows x 12 columns]

1.3 2. Data wrangling steps

1. Reindexed dataframe using _id (unique to every resale transaction)
2. Changed room types into float values, with Executive as 5.5 rooms (extra study/balcony/bathroom)

3. Storey range was converted to avg_storey, the avg floor would be used (every value is a difference of 3 storeys)
4. Resale_price, Floor area converted to float values
5. Month was converted into datetime format, to be used to detrend the time series moving average
6. Year/Month was separated into Year and Month for visualisation purposes
7. Remaining lease was converted into remaining months (float)
8. Update capitalisation and street naming conventions (for purpose of API call later)
9. Categorised towns into regions (North, West, East, North-East, Central) https://www.hdb.gov.sg/about-us/history/hdb-towns-your-home

[79]:
```
@timeit
def clean_df(df: pd.DataFrame):
    '''
    Function to clean the raw dataframe
    ##Parameters
    pd.DataFrame
    ##Cleaning done
        1. Reindexed dataframe using _id (unique to every resale transaction)
        2. Changed room types into float values, with Executive as 4.5 rooms␣
 ↪(extra study/balcony), and Multigeneration 6 rooms
        3. Storey range was converted to avg_storey, the avg floor would be␣
 ↪used (every value is a difference of 3 storeys)
        4. Resale_price, Floor area converted to float values
        5. Month was converted into datetime format, to be used to detrend the␣
 ↪time series moving average
        6. Year/Month was separated into Year and Month for visualisation␣
 ↪purposes
        7. Remaining lease was converted into remaining months (float)
        8. Update capitalisation and street naming conventions (for purpose of␣
 ↪API call later)
        9. Categorised towns into regions (North, West, East, North-East,␣
 ↪Central)
    Returns the cleaned dataframe
    '''
    try:
        # Start
        # Step 1: set index to overall id
        step = 1
        df.set_index('_id', inplace=True)

        # Step 2: Create feature "rooms", "avg_storey"
        def categorise_rooms(flat_type):
            '''
            Helper function for categorising number of rooms
            '''
            if flat_type[0] == 'E' or flat_type[0] == 'M':
```

```
                return 5.5
            else:
                return float(flat_type[0])

        step = 2
        df['rooms'] = df['flat_type'].apply(categorise_rooms)
        step = 3
        df['avg_storey'] = df['storey_range'].apply(lambda x: (int(x[:
↪2])+int(x[-2:]))/2)

        # Step 4-6: Change dtypes
        df['resale_price'] = df['resale_price'].astype('float')
        df['floor_area_sqm'] = df['floor_area_sqm'].astype('float')
        step = 5
        df['timeseries_month'] = pd.to_datetime(df['month'], format="%Y-%m")
        step = 6
        df['year'] = df['timeseries_month'].dt.year
        df['month'] = df['timeseries_month'].dt.month
        step = 7
        df['lease_commence_date'] = df['lease_commence_date'].astype('int')

        # Calculate remaining_lease
        def year_month_to_year(remaining_lease):
            '''
            Helper function to change year & months, into years (float)
            '''
            remaining_lease = remaining_lease.split(' ')
            if len(remaining_lease) > 2:
                year = float(remaining_lease[0]) + float(remaining_lease[2])/12
            else:
                year = float(remaining_lease[0])
            return year

        df['remaining_lease'] = df['remaining_lease'].apply(year_month_to_year)

        step = 8
        # Step 8: Change capitalization of strings
        for column in df.columns:
            if df[column].dtype == 'O':
                df[column] = df[column].str.title()

        # Update address abbreviations for onemap API call
        abbreviations = {'Sth':'South',
                         '[S][t][^.ri]':'Street ',
                         '[S][t]$':'Street',
                         '[S][t][.]':'Saint',
                         'Nth':'North',
```

```
                        'Ave':'Avenue',
                        'Dr':'Drive',
                        'Rd':'Road'}
        for abbreviation, full in abbreviations.items():
            df['street_name'] = df['street_name'].str.replace(abbreviation,␣
↪full, regex=True)
        # Concatenate block and street into a full address
        df['address'] = df['block'] + ', ' + df['street_name']

        # Step 9: Categorise town regions
        step = 9
        town_regions = {'Sembawang' : 'North',
                    'Woodlands' : 'North',
                    'Yishun' : 'North',
                    'Ang Mo Kio' : 'North-East',
                    'Hougang' : 'North-East',
                    'Punggol' : 'North-East',
                    'Sengkang' : 'North-East',
                    'Serangoon' : 'North-East',
                    'Bedok' : 'East',
                    'Pasir Ris' : 'East',
                    'Tampines' : 'East',
                    'Bukit Batok' : 'West',
                    'Bukit Panjang' : 'West',
                    'Choa Chu Kang' : 'West',
                    'Clementi' : 'West',
                    'Jurong East' : 'West',
                    'Jurong West' : 'West',
                    'Tengah' : 'West',
                    'Bishan' : 'Central',
                    'Bukit Merah' : 'Central',
                    'Bukit Timah' : 'Central',
                    'Central Area' : 'Central',
                    'Geylang' : 'Central',
                    'Kallang/Whampoa' : 'Central',
                    'Marine Parade' : 'Central',
                    'Queenstown' : 'Central',
                    'Toa Payoh' : 'Central'}
        df['region'] = df['town'].map(town_regions)
    except Exception as err:
        print(f"Error at step {step}, error message: {err}")
    else:
        # Reorder columns

        df = df[['resale_price', 'year', 'month', 'timeseries_month', 'region',␣
↪'town', 'rooms', 'avg_storey', 'floor_area_sqm', 'remaining_lease',␣
↪'address']]
```

```
            # Unused columns - 'lease_commence_date', 'flat_model',␣
↪'storey_range', 'flat_type', 'block', 'street_name'
    return df
```

[80]:
```
df = clean_df(df)
display(df.dtypes)
df
```

```
clean_df() called at    23:41:09        execution time: 0.0238 seconds
```

```
resale_price              float64
year                        int32
month                       int32
timeseries_month    datetime64[ns]
region                      object
town                        object
rooms                      float64
avg_storey                 float64
floor_area_sqm             float64
remaining_lease            float64
address                     object
dtype: object
```

[80]:

_id	resale_price	year	month	timeseries_month	region	town	…
156373	538000.0	2023	7	2023-07-01	North-East	Ang Mo Kio	
156394	470000.0	2023	7	2023-07-01	East	Bedok	
156395	985000.0	2023	7	2023-07-01	East	Bedok	
156396	635000.0	2023	7	2023-07-01	East	Bedok	
156397	740000.0	2023	7	2023-07-01	East	Bedok	
…	…	…	…	…	…	…	
156389	788000.0	2023	7	2023-07-01	East	Bedok	
156390	728888.0	2023	7	2023-07-01	East	Bedok	
156391	505000.0	2023	7	2023-07-01	East	Bedok	
156392	500000.0	2023	7	2023-07-01	East	Bedok	
156393	521000.0	2023	7	2023-07-01	East	Bedok	

_id	rooms	avg_storey	floor_area_sqm	remaining_lease	\
156373	4.0	11.0	93.0	56.166667	
156394	4.0	2.0	92.0	53.583333	
156395	5.0	11.0	112.0	85.916667	
156396	5.0	2.0	118.0	56.083333	
156397	5.0	5.0	122.0	61.583333	
…	…	…	…	…	
156389	4.0	11.0	93.0	94.500000	
156390	4.0	5.0	93.0	94.500000	
156391	4.0	11.0	92.0	61.166667	

```
156392    4.0         8.0            91.0        56.750000
156393    4.0        14.0            98.0        53.750000

                           address
_id
156373    523, Ang Mo Kio Avenue 5
156394        71, Bedok South Road
156395            219B, Bedok Ctrl
156396   406, Bedok North Avenue 3
156397   156, Bedok South Avenue 3
…                              …
156389  188A, Bedok North Street 4
156390  186A, Bedok North Street 4
156391   110, Bedok Reservoir Road
156392   623, Bedok Reservoir Road
156393    33, Bedok South Avenue 2

[395 rows x 11 columns]
```

1.4 3. Feature Engineering (Geodata)

Lastly, location plays a huge role in house pricing, hence

3.1 Obtaining latitude, longitude, postal codes

3.2 Distance to city center

3.3 Obtaining MRT locations

3.4 Determine nearest MRT and traveling time

1.4.1 3.1 Latitude & longitude from address

Using street name and block, I utilized OneMap API to obtain the latitude, longitude, and postal codes of each flat https://www.onemap.gov.sg/docs

[81]:
```
@timeit
@error_handler
def get_location_data(address_df: pd.DataFrame, verbose=0):
    '''
    Function to carry out API call for Geodata
    ## Parameters
    address_df : pd.DataFrame
        DataFrame that contains a combination of ['block'] and ['street_name']␣
 ↪as ['address'], and ['town']
    verbose : int
        1 to verbose calls
    '''
    # Getting latitude, longitude, postal code
    def get_lat_long(address_df : pd.DataFrame, sleeptime : float =0.15):
```

```
        '''
        The actual API call to be called row-wise to get latitude, longitude,␣
↪and postal code
        ## Parameters
        address_df : pd.DataFrame
            DataFrame that contains a combination of ['block'] and␣
↪['street_name'] as ['address'], and ['town']
        sleeptime : float
            Incorporates sleep time to not exceed a max of 250 calls per min
            Default 0.15s, not required if we are caching call
        '''

        # Lag time between calls - No longer needed with Cache, since we will␣
↪not likely exceed the call limit
        # time.sleep(sleeptime)

        # API call
        try:
            address = address_df['address']
            if 'Jln Batu' in address:
                address = address.replace('Jln Batu', 'JALAN BATU DI TANJONG␣
↪RHU')
            elif '27, Marine Cres' in address:
                address = address.replace('Marine Cres', 'MARINE CRESCENT␣
↪MARINE CRESCENT VILLE')
            elif '215, Choa Chu Kang Ctrl' in address:
                address = '680215'
            elif '216, Choa Chu Kang Ctrl' in address:
                address = '680216'
            call = f'https://developers.onemap.sg/commonapi/search?
↪searchVal={address}&returnGeom=Y&getAddrDetails=Y'
            # Caching is enabled in the session
            response = session.get(call)
            response.raise_for_status()
            data = response.json()
            if verbose >0:
                print(call)
                pprint(data)

            # Returns a the results in string
            return data['results'][0]['LATITUDE'] + ',' +␣
↪data['results'][0]['LONGITUDE'] + ' ' + data['results'][0]['POSTAL']

        except Exception as err:
            print(f'Error occurred - get_lat_long() API call: {err} on the␣
↪following call:')
```

```
            pprint(call)
            return '0,0 0' # Still return 0 values

    def to_numpy_array(lat_long_df):
        # Build a numpy array from latitude and longitude
        combi = np.array([lat_long_df[0], lat_long_df[1]])
        return combi

    # This calls the API call function row wise
    position = address_df.apply(get_lat_long, axis=1)

    try:
        # Split the string into two columns (column 0 is the latitude and␣
↪longitude, column 1 is the postal code)
        temp_df = position.str.split(expand=True)
        # Postal code
        temp_df.iloc[:,1] = temp_df.iloc[:,1].apply(lambda x: 0 if x=='NIL'␣
↪else x)
        temp_df.iloc[:,1] = temp_df.iloc[:,1].astype('int')
        # Latitude and longitude split (by ,)
        lat_long_df = temp_df.iloc[:,0].str.split(pat=',', expand=True)
        lat_long_df = lat_long_df.astype('float')
        # Convert into numpy array, for faster matrix operations later
        numpy_array = lat_long_df.apply(to_numpy_array, axis=1)

    except Exception as err:
        print(f"Error occurred - Splitting data : {err}")
    else:
        geo_data_df = pd.concat([temp_df, lat_long_df, numpy_array], axis=1)
        geo_data_df.columns = ['lat_long', 'postal_code', 'latitude',␣
↪'longitude', 'numpy_array']
        return geo_data_df
```

[82]:
```
geo_data_df= get_location_data(df[['address']])
display(geo_data_df.dtypes)
geo_data_df
```

```
get_location_data() called at   23:41:09        execution time: 0.9606 seconds

lat_long        object
postal_code     object
latitude       float64
longitude      float64
numpy_array     object
dtype: object
```

[82]:
```
                                lat_long postal_code  latitude   longitude  \
 _id
```

```
156373   1.3729419359545,103.852866402032      560523  1.372942  103.852866
156394  1.32041601736618,103.942732310081      460071  1.320416  103.942732
156395  1.32542365484015,103.933559134228      462219  1.325424  103.933559
156396  1.32822550655778,103.934469289425      460406  1.328226  103.934469
156397  1.31828663769919,103.945481750941      460156  1.318287  103.945482
…                                     …       …       …         …
156389  1.33163353101368,103.941371320775      461188  1.331634  103.941371
156390  1.32991427421905,103.940310513079      461186  1.329914  103.940311
156391  1.33014480390561,103.910318002341      470110  1.330145  103.910318
156392  1.33365754294799,103.916804692739      470623  1.333658  103.916805
156393   1.3228412425757,103.939128684332      460033  1.322841  103.939129

                                  numpy_array
_id
156373   [1.3729419359545, 103.852866402032]
156394  [1.32041601736618, 103.942732310081]
156395  [1.32542365484015, 103.933559134228]
156396  [1.32822550655778, 103.934469289425]
156397  [1.31828663769919, 103.945481750941]
…                                         …
156389  [1.33163353101368, 103.941371320775]
156390  [1.32991427421905, 103.940310513079]
156391  [1.33014480390561, 103.910318002341]
156392  [1.33365754294799, 103.916804692739]
156393   [1.3228412425757, 103.939128684332]

[395 rows x 5 columns]
```

1.4.2 3.2 Distance to city center

The central district of Singapore has the highest housing prices. Property nearer to the city centre tend to have a higher price.

We will make use of this to create a new feature to test if it is significant in model building.

[83]:
```
@error_handler
def distance_to(df_series : pd.Series, to_address : str , dist_type :␣
↪str='latlong', verbose : int=0):
    '''
    Function to determine distance to a location (from a series of locations in␣
↪a dataframe
    ## Parameters
    df_series : pd.Series contains numpy array containing [latitude, longitude]
    to_address : str
        place and streetname
    dist_type : str
        type of distance (latlong, or geodesic)
    verbose : int
```

```
        whether to show the workings of the function

    Returns np.Series of distance between input and location
    '''
    # if an address is given
    if isinstance(to_address, str):
        call = f'https://developers.onemap.sg/commonapi/search?
↪searchVal={to_address}&returnGeom=Y&getAddrDetails=Y'
        response = requests.get(call)
        response.raise_for_status()
        data = response.json()
        to_coordinates = np.array([float(data['results'][0]['LATITUDE']),␣
↪float(data['results'][0]['LONGITUDE'])])

    if verbose==1:
        print(f'Coordinates of {to_address} : {to_coordinates}')

    def matrix_operations(from_coordinates, to_coordinates):
        # Matrix substraction to get difference
        distance_diff = from_coordinates - to_coordinates
        absolute_dist = np.absolute(distance_diff)

        #Matrix sum over latitude and longitude of each entry
        sum_of_distances = np.sum(absolute_dist)

        if verbose==2:
            print(f'Difference in distances: \n{distance_diff}')
            print()
            print(f'Absolute difference: \n{absolute_dist}')
            print()
            print(f'Sum of distances \n {sum_of_distances}')

        return sum_of_distances

    def geodesic_operations(from_coordinates, coordinates):
        from_coordinates = tuple(from_coordinates)
        coordinates = tuple(coordinates)
        geodesic_dist = GD(from_coordinates, coordinates).kilometers
        return np.round(geodesic_dist,2)

    if dist_type == 'geodesic':
        diff_dist = df_series.apply(geodesic_operations,␣
↪coordinates=to_coordinates)
    else:
        diff_dist = df_series.apply(matrix_operations,␣
↪coordinates=to_coordinates)
```

```
    return diff_dist
```

[84]:
```
dist_to_marina_bay = distance_to(geo_data_df['numpy_array'], 'Marina Bay',␣
 ↪dist_type='geodesic', verbose=1)
dist_to_marina_bay = pd.Series(dist_to_marina_bay, name='dist_to_marina_bay')
df = pd.concat([df, dist_to_marina_bay, geo_data_df['latitude'],␣
 ↪geo_data_df['longitude'], geo_data_df['postal_code']], axis=1)
df
```

```
Coordinates of Marina Bay : [  1.2834542  103.86080905]
```

[84]:

_id	resale_price	year	month	timeseries_month	region	town
156373	538000.0	2023	7	2023-07-01	North-East	Ang Mo Kio
156394	470000.0	2023	7	2023-07-01	East	Bedok
156395	985000.0	2023	7	2023-07-01	East	Bedok
156396	635000.0	2023	7	2023-07-01	East	Bedok
156397	740000.0	2023	7	2023-07-01	East	Bedok
…	…	…	…	…	…	…
156389	788000.0	2023	7	2023-07-01	East	Bedok
156390	728888.0	2023	7	2023-07-01	East	Bedok
156391	505000.0	2023	7	2023-07-01	East	Bedok
156392	500000.0	2023	7	2023-07-01	East	Bedok
156393	521000.0	2023	7	2023-07-01	East	Bedok

_id	rooms	avg_storey	floor_area_sqm	remaining_lease
156373	4.0	11.0	93.0	56.166667
156394	4.0	2.0	92.0	53.583333
156395	5.0	11.0	112.0	85.916667
156396	5.0	2.0	118.0	56.083333
156397	5.0	5.0	122.0	61.583333
…	…	…	…	…
156389	4.0	11.0	93.0	94.500000
156390	4.0	5.0	93.0	94.500000
156391	4.0	11.0	92.0	61.166667
156392	4.0	8.0	91.0	56.750000
156393	4.0	14.0	98.0	53.750000

_id	address	dist_to_marina_bay	latitude	longitude
156373	523, Ang Mo Kio Avenue 5	9.93	1.372942	103.852866
156394	71, Bedok South Road	9.99	1.320416	103.942732
156395	219B, Bedok Ctrl	9.33	1.325424	103.933559
156396	406, Bedok North Avenue 3	9.58	1.328226	103.934469
156397	156, Bedok South Avenue 3	10.18	1.318287	103.945482
…	…	…	…	…

```
156389  188A, Bedok North Street 4                10.43  1.331634  103.941371
156390  186A, Bedok North Street 4                10.23  1.329914  103.940311
156391   110, Bedok Reservoir Road                 7.55  1.330145  103.910318
156392   623, Bedok Reservoir Road                 8.35  1.333658  103.916805
156393    33, Bedok South Avenue 2                 9.74  1.322841  103.939129

       postal_code
_id
156373      560523
156394      460071
156395      462219
156396      460406
156397      460156
…            …
156389      461188
156390      461186
156391      470110
156392      470623
156393      460033

[395 rows x 15 columns]
```

1.4.3 3.3 MRT Locations

The location of all MRT stations was also obtained using OneMap API and saved as a json file locally

[85]:
```
@timeit
@error_handler
def update_mrt_coordinates(mrt_stations=None, filepath='static/mrt_dict.json'):
    '''
    Function to API call for MRT station coordinates and write to json file
    ## Parameters
    mrt_stations : list
        list of mrt station names, default to All stations if nothing is given
    filepath : str
        filepath and name of json file to write to, should end with .json
    Returns None
    '''
    if not mrt_stations:
        mrt_stations = ['Admiralty MRT', 'Aljunied MRT', 'Ang Mo Kio MRT',␣
 ↪'Bakau LRT', 'Bangkit LRT', 'Bartley MRT', 'Bayfront MRT',
                        'Bayshore MRT', 'Beauty World MRT', 'Bedok MRT', 'Bedok␣
 ↪North MRT', 'Bedok Reservoir MRT', 'Bencoolen MRT',
                        'Bendemeer MRT', 'Bishan MRT', 'Boon Keng MRT', 'Boon␣
 ↪Lay MRT', 'Botanic Gardens MRT', 'Braddell MRT',
```

```
                        'Bras Basah MRT', 'Buangkok MRT', 'Bugis MRT', 'Bukit␣
↪Batok MRT', 'Bukit Brown MRT', 'Bukit Gombak MRT',
                        'Bukit Panjang MRT', 'Buona Vista MRT', 'Caldecott␣
↪MRT', 'Cashew MRT', 'Changi Airport MRT',
                        'Chinatown MRT', 'Chinese Garden MRT', 'Choa Chu Kang␣
↪MRT', 'City Hall MRT', 'Clarke Quay MRT',
                        'Clementi MRT', 'Commonwealth MRT', 'Compassvale LRT',␣
↪'Cove LRT', 'Dakota MRT', 'Dhoby Ghaut MRT',
                        'Downtown MRT', 'Xilin MRT', 'Tampines East MRT',␣
↪'Mayflower MRT', 'Upper Thomson MRT',
                        'Lentor MRT', 'Woodlands North MRT', 'Woodlands South␣
↪MRT', 'Esplanade MRT', 'Eunos MRT',
                        'Expo MRT', 'Fajar LRT', 'Farmway LRT', 'Farrer Park␣
↪MRT', 'Fort Canning MRT',
                        'Gardens by the Bay MRT', 'Geylang Bahru MRT',␣
↪'HarbourFront MRT', 'Haw Par Villa MRT', 'Hillview MRT',
                        'Holland Village MRT', 'Hougang MRT', 'Jalan Besar␣
↪MRT', 'Joo Koon MRT', 'Jurong East MRT',
                        'Jurong West MRT', 'Kadaloor LRT', 'Kaki Bukit MRT',␣
↪'Kallang MRT', 'Kembangan MRT', 'Keppel MRT',
                        'King Albert Park MRT', 'Kovan MRT', 'Kranji MRT',␣
↪'Labrador Park MRT', 'Lakeside MRT', 'Lavender MRT',
                        'Layar LRT', 'Little India MRT', 'Lorong Chuan MRT',␣
↪'MacPherson MRT', 'Marina Bay MRT', 'Marina South Pier MRT',
                        'Marsiling MRT', 'Marymount MRT', 'Mattar MRT',␣
↪'Meridian LRT', 'Mountbatten MRT',
                        'Newton MRT', 'Nibong LRT', 'Nicoll Highway MRT',␣
↪'Novena MRT', 'Oasis LRT', 'One-North MRT', 'Orchard MRT',
                        'Outram Park MRT', 'Paya Lebar MRT', 'Pasir Ris MRT',␣
↪'Paya Lebar MRT', 'Pasir Ris MRT', 'Paya Lebar MRT', 'Pasir Ris MRT',
                        'Pioneer MRT', 'Potong Pasir MRT', 'Promenade MRT',␣
↪'Punggol MRT', 'Queenstown MRT', 'Raffles Place MRT', 'Redhill MRT',
                        'Riviera LRT', 'Rochor MRT', 'Sembawang MRT', 'Sengkang␣
↪MRT', 'Serangoon MRT', 'Simei MRT', 'Sixth Avenue MRT',
                        'Somerset MRT', 'Springleaf MRT', 'Stadium MRT',␣
↪'Stevens MRT', 'Sumang LRT', 'Tai Seng MRT', 'Tampines MRT',
                        'Tampines East MRT', 'Tampines West MRT', 'Tanah Merah␣
↪MRT', 'Tanjong Pagar MRT', 'Tanjong Rhu MRT', 'Teck Lee LRT',
                        'Telok Ayer MRT', 'Telok Blangah MRT', 'Thanggam LRT',␣
↪'Tiong Bahru MRT', 'Toa Payoh MRT',
                        'Tuas Crescent MRT', 'Tuas Link MRT', 'Tuas West Road␣
↪MRT', 'Ubi MRT', 'Upper Changi MRT',
                        'Woodlands MRT', 'Woodlands South MRT', 'Woodlands␣
↪North MRT', 'Yew Tee MRT', 'Yio Chu Kang MRT', 'Yishun MRT']
    # Future stations - 'Tampines North MRT', 'Tengah MRT'
```

```
    mrt_coordinates = {}
    for mrt in mrt_stations:
        response = requests.get(f"https://developers.onemap.sg/commonapi/search?
↪searchVal={mrt}&returnGeom=Y&getAddrDetails=Y")
        response.raise_for_status()
        data = response.json()
        # string (lat,long) as key
        #␣
↪mrt_coordinates[f"{data['results'][0]['LATITUDE']},{data['results'][0]['LONGITUDE']}"]␣
↪= mrt
        mrt_coordinates[mrt] =␣
↪(float(data['results'][0]['LATITUDE']),float(data['results'][0]['LONGITUDE']))

    with open(filepath, 'w')as f:
        json.dump(mrt_coordinates, f, indent=4)

@timeit
@error_handler
def get_mrt_coordinates(filepath = 'static/mrt_dict.json'):
    '''
    Function to read saved mrt_coordinates from json file
    ## Parameters
    filepath : str
        filepath to json file
    Returns data : dictionary
    '''
    with open(filepath, 'r') as f:
        file = f.read()
        data = json.loads(file)
        return data
```

Load Json file and convert to numpy array to utilize matrix operations.

[86]:
```
mrt_coordinates_dict = get_mrt_coordinates()

# Convert coordinates into numpy arrays
mrt_stations = np.array(list(mrt_coordinates_dict.keys()))
mrt_coordinates = np.array(list(mrt_coordinates_dict.values()))
```

```
get_mrt_coordinates() called at         23:41:11        execution time: 0.0006
seconds
```

1.4.4 3.4 Nearest MRT stations and Minimum distance/time

- Using the matrix operations, we are able to find the nearest MRT station by absolute distance
- Then use OneMap's route_api_call() to get distance/time to MRT stations

[87]:
```
@error_handler
def find_nearest_stations(geo_data_df : pd.DataFrame, mrt_stations : np.
↪array=mrt_stations, mrt_coordinates : np.array=mrt_coordinates,
                          n_nearest_stations: int=2, verbose : int=0):
    '''
    Function to determine nearest MRT station of the resale_flat based on␣
↪latitude and longitude
    ## Parameters
        geo_data_df : pd.DataFrame
        mrt_stations : np.array
        mrt_coordinates : np.array
        n_nearest_stations: int=2
        verbose : int=0

    Returns a list of n_nearest stations
    '''
    # Matrix substraction to get difference with each MRT, convert to absolute␣
↪values
    distance_diff = geo_data_df['numpy_array'] - mrt_coordinates
    absolute_dist = np.absolute(distance_diff)

    # Matrix sum over latitude and longitude of each entry
    sum_of_distances = np.sum(absolute_dist, axis=1)

    # Sort and search based on desired n_nearest_stations
    sorted_distances = np.sort(sum_of_distances)
    nearest_stations = []
    for n in range(n_nearest_stations):
        idx = np.where(sum_of_distances==sorted_distances[n])
        from_coordinates = tuple(geo_data_df['numpy_array'])
        to_coordinates = tuple(mrt_coordinates[idx][0])
        geodesic_dist = GD(from_coordinates, to_coordinates).kilometers
        nearest_stations.append(mrt_stations[idx][0])
        nearest_stations.append(np.round(geodesic_dist,2))

    if verbose==1:
        print(f'Difference in distances: \n{distance_diff[:5]}')
        print()
        print(f'Absolute difference: \n{absolute_dist[:5]}')
        print()
        print(f'Sum of distances \n {sum_of_distances[:5]}')
        print()
        print(f'Sorted distances\n{sorted_distances[:5]}')
        print()
        print(f'Top {n_nearest_stations}')
        print(nearest_stations)
```

```
    return nearest_stations
```

[88]:
```
n_nearest_stations = 1
# Matrix operations to find nearest MRT stations for each row
nearest_stations = geo_data_df.apply(find_nearest_stations,␣
 ↪n_nearest_stations=n_nearest_stations, axis=1, verbose=0)
nearest_stations_df = pd.DataFrame(nearest_stations.tolist(), index=geo_data_df.
 ↪index, columns=['nearest_station_'+ str(x) for x in␣
 ↪range(n_nearest_stations)] + ['dist_to_station_'+ str(x) for x in␣
 ↪range(n_nearest_stations)])
nearest_stations_df
```

[88]:

_id	nearest_station_0	dist_to_station_0
156373	Ang Mo Kio MRT	0.50
156394	Bayshore MRT	0.81
156395	Bedok Reservoir MRT	1.25
156396	Bedok Reservoir MRT	0.96
156397	Bayshore MRT	0.67
…	…	…
156389	Tanah Merah MRT	0.74
156390	Tanah Merah MRT	0.74
156391	Kaki Bukit MRT	0.57
156392	Bedok North MRT	0.18
156393	Tanah Merah MRT	0.94

[395 rows x 2 columns]

[89]:
```
df = pd.concat([df, nearest_stations_df], axis=1)
'''df = df[['resale_price', 'year', 'month', 'timeseries_month', 'region',␣
 ↪'town',
        'rooms', 'avg_storey', 'floor_area_sqm', 'remaining_lease',␣
 ↪'dist_to_marina_bay',
        'latitude', 'longitude',
        'nearest_station_0', 'dist_to_station_0',
        'postal_code', 'address',]]'''
display(df.dtypes)
df
```

```
resale_price                 float64
year                           int32
month                          int32
timeseries_month      datetime64[ns]
region                        object
town                          object
rooms                        float64
avg_storey                   float64
floor_area_sqm               float64
```

```
remaining_lease              float64
address                       object
dist_to_marina_bay           float64
latitude                     float64
longitude                    float64
postal_code                   object
nearest_station_0             object
dist_to_station_0            float64
dtype: object
```

[89]:

_id	resale_price	year	month	timeseries_month	region	town	rooms	avg_storey	floor_area_sqm	remaining_lease	address	dist_to_marina_bay	latitude	longitude
156373	538000.0	2023	7	2023-07-01	North-East	Ang Mo Kio	4.0	11.0	93.0	56.166667	523, Ang Mo Kio Avenue 5	9.93	1.372942	103.852866
156394	470000.0	2023	7	2023-07-01	East	Bedok	4.0	2.0	92.0	53.583333	71, Bedok South Road	9.99	1.320416	103.942732
156395	985000.0	2023	7	2023-07-01	East	Bedok	5.0	11.0	112.0	85.916667	219B, Bedok Ctrl	9.33	1.325424	103.933559
156396	635000.0	2023	7	2023-07-01	East	Bedok	5.0	2.0	118.0	56.083333	406, Bedok North Avenue 3	9.58	1.328226	103.934469
156397	740000.0	2023	7	2023-07-01	East	Bedok	5.0	5.0	122.0	61.583333	156, Bedok South Avenue 3	10.18	1.318287	103.945482
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
156389	788000.0	2023	7	2023-07-01	East	Bedok	4.0	11.0	93.0	94.500000	188A, Bedok North Street 4	10.43	1.331634	103.941371
156390	728888.0	2023	7	2023-07-01	East	Bedok	4.0	5.0	93.0	94.500000				
156391	505000.0	2023	7	2023-07-01	East	Bedok	4.0	11.0	92.0	61.166667				
156392	500000.0	2023	7	2023-07-01	East	Bedok	4.0	8.0	91.0	56.750000				
156393	521000.0	2023	7	2023-07-01	East	Bedok	4.0	14.0	98.0	53.750000				

```
156390  186A, Bedok North Street 4               10.23  1.329914  103.940311
156391   110, Bedok Reservoir Road                7.55  1.330145  103.910318
156392   623, Bedok Reservoir Road                8.35  1.333658  103.916805
156393    33, Bedok South Avenue 2                9.74  1.322841  103.939129

       postal_code    nearest_station_0  dist_to_station_0
_id
156373      560523       Ang Mo Kio MRT               0.50
156394      460071         Bayshore MRT               0.81
156395      462219  Bedok Reservoir MRT               1.25
156396      460406  Bedok Reservoir MRT               0.96
156397      460156         Bayshore MRT               0.67
…            …                 …                  …
156389      461188      Tanah Merah MRT               0.74
156390      461186      Tanah Merah MRT               0.74
156391      470110       Kaki Bukit MRT               0.57
156392      470623      Bedok North MRT               0.18
156393      460033      Tanah Merah MRT               0.94

[395 rows x 17 columns]
```

[90]:
```python
name = input('Name save file: e.g. <2023_apr>\n')
if name != '':
    filename= f'static/{name}.csv'
    df.to_csv(filename)
    print(f'File saved as {filename}')
```

```
File saved as static/2023_test.csv
```