DSAI Mini-project (Data Collection)

HDB Resale Price Analysis

The Housing Development Board (HDB) flat is a cornerstone of the Authentic Singaporean Experience (for 78% of us). While many of us purchase flats directly from HDB through the Build-to-Order (BTO) exercise, there are some factors which will lead to many others purchasing an HDB flat on the resale market instead.

Because a home purchase is typically the most expensive item that many of us purchase in our lives, it is really important that we pay the right amount for the house, as overpaying for a home can cost us a lot of money, typically in the tens of thousands, or hundreds of thousands of dollars.

However, housing prices are determined by a myriad of factors, and it can get very confusing to understand what is the right value to offer for a home. So, this project will aim to address the issue of predicting the resale value of a house.

The problem statement is:

Can we accurately predict the resale value of a Housing Development Board (HDB) flat based on various housing-related data?

The audience is mainly sellers and buyers of hdb flat, providing them info on the what, where and when that affects the hdb price

This file contains the data collection step for our dataset. In which we employed various methods to collect data such as

- Dataset download from data.gov https://data.gov.sg/collections/189/view.
- Beautiful soup webscraping from wikipedia
- Use of Onemap api to collect locational data

This file only contains the data collection. All EDA, Data visualisation and ML tasks are done in the other file

In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import os
import json
```

The main dataset that out analysis would be about is the HDB resale price dataset collected from kaggle. There are in total 4 separate resale datasets, containing resale data from the years 2000 to 2025 March

In [ ]:
```
dfs = []
for file in os.listdir('../datasets'):
    if file.endswith('.csv'):  # Check if the file is a CSV
        dfs.append(pd.read_csv(f"../datasets/{file}"))

df = pd.concat(dfs, ignore_index=True)  # Ignore index to avoid duplication in the concatenated dataframe
df.head(2)
```

Out[ ]:

	month	town	flat_type	block	street_name	storey_range	floor_area_sqm	flat_model	lease_commence_date	resale_price	remaining_lease
0	2000-01	ANG MO KIO	3 ROOM	170	ANG MO KIO AVE 4	07 TO 09	69.0	Improved	1986	147000.0	NaN
1	2000-01	ANG MO KIO	3 ROOM	174	ANG MO KIO AVE 4	04 TO 06	61.0	Improved	1986	144000.0	NaN

WebScraping and Geo data collection

To make our project more fruitful, we decided to add things such as:

- MRT station, with lat and long data
- Malls with lat and long data
- Lat and long extraction for each house

This will allow us to calculate things outside of our dataset, such as distances between each home to ammenities, nearest public space available and etc that can greatly help with the accuracy of our model and understanding of our data

Tools

- Beautiful soup for web scraping
- One map for lat and long finding of locations

In [7]:
```
import requests
from bs4 import BeautifulSoup
import json
import re
from tqdm import tqdm
```

MRT Station Extraction

First we will try to extract lat and long data of MRT stations in Singapore. This requires a 2 step process:

- Web scrapping for all available MRT stations in SG currently
- Lat and long location using One Map API

We will be scrapping from wikipedia, even though it may not be the most reliable source of information it is the easiest to extract just via html elements, and since it is a Singapore page, it is unlikely to have major reliability issues.

In [8]:
```
url = "https://en.wikipedia.org/wiki/List_of_Singapore_MRT_stations"  # Example URL
response = requests.get(url)  # Send HTTP request

# Check if request was successful
if response.status_code == 200:
    print("Request successful!")
else:
    print("Failed to retrieve page.")

soup = BeautifulSoup(response.text, "html.parser")  # Parse HTML
```

```
Request successful!
```

In [9]:
```
tables = soup.find_all("table")  # Find all tables
print(f"Number of tables found: {len(tables)}")
```

```
Number of tables found: 23
```

There have been 23 tables found in the wiki, Singapore only has 6 that are official constructed. After some identification, table 2 to 8 are the correct tables we are looking for.

Below is a function that has been created by us to extract the mrt station name. If it meets any issues, it will just ignore and proceed

In [10]:
```
def extract_mrt(table):
    rows = table.find_all("tr")  # Get all table rows
    mrt_stations = pd.DataFrame({
        'station_name': [],
        'code': []
    })

    for row in rows:  # Print first 5 rows
        cols = row.find_all("td")  # Get table columns
        row = [col.text.strip() for col in cols]


        try:
            match = re.match(r'([A-Za-z]+[0-9]+)', row[0])
            if match:
                result = match.group(0)  # This gives you 'NS1'
            mrt_stations.loc[len(mrt_stations)] = [row[1], result]
        except:
            continue

    return mrt_stations
```

Below we extracted the north south line as a test and see that it works

In [11]:
```
mrt_stations = extract_mrt(tables[2])
mrt_stations.head()
```

Out[11]:

	station_name	code
0	Jurong East	NS1
1	Bukit Batok	NS2
2	Bukit Gombak	NS3
3	Brickland	NS3
4	Choa Chu Kang	NS4

Now we can apply it for all MRT station tables, combine them into a single dataframe and then delete any duplicate stations

In [12]:
```
mrt_dfs = []
for table in tables[2:8]:
    mrt_dfs.append(extract_mrt(table))

print(len(mrt_dfs))
mrt_df = pd.concat(mrt_dfs)
mrt_stations.drop_duplicates(inplace=True)

print(mrt_df.shape)
mrt_df.head()
```

```
6
(190, 2)
```

Out[12]:

	station_name	code
0	Jurong East	NS1
1	Bukit Batok	NS2
2	Bukit Gombak	NS3
3	Brickland	NS3
4	Choa Chu Kang	NS4

We see that we are left with 190 different MRT stations. We can now get the longitude and latitude of the MRT stations using one map

In [13]:
```
mrt_building = []
mrt_lat = []
mrt_long = []
headers = {"Authorization": "Bearer **********"}

# Query and look for location using code as it is more accurate
for index, row in tqdm(mrt_df.iterrows(), total=len(mrt_df), desc="Fetching MRT coordinates"):
    query_address = row['code']  # Access value in the 'code' column
    url = f"https://www.onemap.gov.sg/api/common/elastic/search?searchVal={query_address}%MRT&returnGeom=Y&getAddrDetails=Y&pageNum=1"
    response = requests.get(url, headers=headers)

    data_mrt = json.loads(response.content)

    if data_mrt['found'] != 0:
        mrt_building.append(data_mrt["results"][0]["BUILDING"])
        mrt_lat.append(data_mrt["results"][0]["LATITUDE"])
        mrt_long.append(data_mrt["results"][0]["LONGITUDE"])

        # print(f"{query_address}, Lat: {data_mrt['results'][0]['LATITUDE']} Long: {data_mrt['results'][0]['LONGITUDE']}")
    else:
        mrt_building.append('NotFound')
        mrt_lat.append('NotFound')
        mrt_long.append('NotFound')

# Store this information in a dataframe
mrt_location = pd.DataFrame({
    'mrt': mrt_df['station_name'],
    'building': mrt_building,
    'latitude': mrt_lat,
    'longitude': mrt_long,
    'code': mrt_df['code']
})

mrt_location.head()
```

```
Fetching MRT coordinates: 100%|██████████| 190/190 [00:23<00:00,  7.93it/s]
```

Out[13]:

	mrt	building	latitude	longitude	code
0	Jurong East	JURONG EAST MRT STATION (EW24 / NS1)	1.33315281585758	103.742286332403	NS1
1	Bukit Batok	BUKIT BATOK MRT STATION (NS2)	1.34903331201636	103.749566478309	NS2
2	Bukit Gombak	BUKIT GOMBAK MRT STATION (NS3)	1.35861159094192	103.751790910733	NS3
3	Brickland	BUKIT GOMBAK MRT STATION (NS3)	1.35861159094192	103.751790910733	NS3
4	Choa Chu Kang	CHOA CHU KANG MRT STATION (NS4)	1.38536316540225	103.744370779756	NS4

We have managed to capture the lat and long of our mrt stations, with only 4 missing. And when we inspect further these have issues as they don't exist yet so we can drop them

In [14]:
```
mrt_location[mrt_location['building'] == 'NotFound']
```

Out[14]:

	mrt	building	latitude	longitude	code
6	Sungei Kadut	NotFound	NotFound	NotFound	NS6
37	DE1	NotFound	NotFound	NotFound	DE1
38	Sungei Kadut	NotFound	NotFound	NotFound	DE2
32	Changi Airport Terminal 5[d]	NotFound	NotFound	NotFound	CR1

In [15]:
```
mrt_location =  mrt_location[mrt_location['building'] != 'NotFound']
mrt_location.head()
```

Out[15]:

	mrt	building	latitude	longitude	code
0	Jurong East	JURONG EAST MRT STATION (EW24 / NS1)	1.33315281585758	103.742286332403	NS1
1	Bukit Batok	BUKIT BATOK MRT STATION (NS2)	1.34903331201636	103.749566478309	NS2
2	Bukit Gombak	BUKIT GOMBAK MRT STATION (NS3)	1.35861159094192	103.751790910733	NS3
3	Brickland	BUKIT GOMBAK MRT STATION (NS3)	1.35861159094192	103.751790910733	NS3
4	Choa Chu Kang	CHOA CHU KANG MRT STATION (NS4)	1.38536316540225	103.744370779756	NS4

Mall Extraction

Next we can do the same for the malls in Singapore, however some tweaks are needed as the wiki page is different

In [16]:
```
url = "https://en.wikipedia.org/wiki/List_of_shopping_malls_in_Singapore"  # Example URL
response = requests.get(url)  # Send HTTP request

# Check if request was successful
if response.status_code == 200:
    print("Request successful!")
```

```
else:
    print("Failed to retrieve page.")

soup = BeautifulSoup(response.text, "html.parser")  # Parse HTML
```

```
Request successful!
```

In [17]:

```
malls = soup.find_all(class_='div-col') # Find all tables
print(f"Number of lists found: {len(malls)}")
```

```
Number of lists found: 7
```

To find them we used located the div col class element as they were the only elements with this class. We managed to find 7 which is correct based off manually counting the wikipedia page

In [18]:

```
def extract_malls(malls_div):
    mall_names = []
    for div in malls_div:
        li_tags = div.find_all('li')
        for li in li_tags:
            # Check if the 'li' contains a link (for malls with 'a' tags)
            if li.a:
                mall_names.append(li.a.text.strip())
            else:
                mall_names.append(li.text.strip())

    return mall_names

list_of_malls = extract_malls(malls)
list_of_malls[0:10]
```

Out[18]:

```
['100 AM',
 '313@Somerset',
 'Aperia',
 'Balestier Hill Shopping Centre',
 'Bugis Cube',
 'Bugis Junction',
 'Bugis+',
 'Capitol Piazza',
 'Cathay Cineleisure Orchard',
 'Clarke Quay Central']
```

In [19]:

```
mall_building = []
mall_lat = []
mall_long = []
headers = {"Authorization": "Bearer **********"}  # API token

# Query and look for location using code as it is more accurate
for mall in tqdm(list_of_malls, desc="Fetching Mall coordinates"):
    url = f"https://www.onemap.gov.sg/api/common/elastic/search?searchVal={mall}%MALL&returnGeom=Y&getAddrDetails=Y&pageNum=1"  # Modify for m
    response = requests.get(url, headers=headers)
    try:
        data_mall = json.loads(response.content)
    except:
        mall_building.append('NotFound')
        mall_lat.append('NotFound')
        mall_long.append('NotFound')
        continue

    if data_mall['found'] != 0:
        mall_building.append(data_mall["results"][0]["BUILDING"])
        mall_lat.append(data_mall["results"][0]["LATITUDE"])
        mall_long.append(data_mall["results"][0]["LONGITUDE"])
        # print(f"{query_address}, Lat: {data_mall['results'][0]['LATITUDE']} Long: {data_mall['results'][0]['LONGITUDE']}")
    else:
        mall_building.append('NotFound')
        mall_lat.append('NotFound')
        mall_long.append('NotFound')

# Store this information in a dataframe
mall_location = pd.DataFrame({
    'mall': list_of_malls,
    'building': mall_building,
    'latitude': mall_lat,
    'longitude': mall_long,
})

mall_location.head()  # Display the first few rows
```

```
Fetching Mall coordinates: 100%|██████████| 174/174 [00:26<00:00,  6.68it/s]
```

Out[19]:

	mall	building	latitude	longitude
0	100 AM	100 AM	1.27458821795426	103.84347073661
1	313@Somerset	NotFound	NotFound	NotFound
2	Aperia	NotFound	NotFound	NotFound
3	Balestier Hill Shopping Centre	BALESTIER 288	1.32297698121569	103.852811354547
4	Bugis Cube	BUGIS MRT STATION	1.30026468984101	103.855614760658

After the extraction, there are a few malls with issues from the API but we can actually just populate them manually

In [20]:
```
mall_location[mall_location['building'] == 'NotFound']
```

Out[20]:

	mall	building	latitude	longitude
1	313@Somerset	NotFound	NotFound	NotFound
2	Aperia	NotFound	NotFound	NotFound
14	Duo	NotFound	NotFound	NotFound
16	Funan	NotFound	NotFound	NotFound
84	Paya Lebar Quarter (PLQ)	NotFound	NotFound	NotFound
138	HillV2	NotFound	NotFound	NotFound
149	VivoCity	NotFound	NotFound	NotFound
154	IMM	NotFound	NotFound	NotFound
155	Jem	NotFound	NotFound	NotFound
156	Westgate	NotFound	NotFound	NotFound
161	Anchorpoint	NotFound	NotFound	NotFound
162	OD Mall	NotFound	NotFound	NotFound

In [21]:
```
pd.set_option('display.float_format', '{:,.7f}'.format)  # This will show up to 10 decimal places7
mall_info = {
    "Duo": ["DUO GALLERIA", 1.2995344, 103.8584017],
    "Aperia": ["APERIA", 1.3097113, 103.8643265],
    "313@Somerset": ["313 @ SOMERSET", 1.3010144, 103.8383607],
    "Funan": ["FUNAN", 1.2913476, 103.8499898],
    "Paya Lebar Quarter (PLQ)": ["PAYA LEBAR QUARTER", 1.3161739, 103.893139],
    "HillV2": ["HillV2", 1.3626746, 103.764173],
    "VivoCity": ["VIVOCITY", 1.2642932, 103.8223047],
    "IMM": ["IMM BUILDING", 1.3348753, 103.7468948],
    "Jem": ["JEM", 1.3332934, 103.7432788],
    "Westgate": ["WESTGATE", 1.3341577, 103.7427665],
    "Anchorpoint": ["ANCHORPOINT SHOPPING CENTRE", 1.2889348, 103.8056078],
    "OD Mall": ["GRANTRAL MALL @ CLEMENTI", 1.3142705, 103.7651475]
}

for index, row in mall_location.iterrows():
    mall_name = row['mall']
    if mall_name in mall_info:
        mall_location.at[index, 'building'] = mall_info[mall_name][0]
        mall_location.at[index, 'latitude'] = mall_info[mall_name][1]
        mall_location.at[index, 'longitude'] = mall_info[mall_name][2]

mall_location.head()
```

Out[21]:

	mall	building	latitude	longitude
0	100 AM	100 AM	1.27458821795426	103.84347073661
1	313@Somerset	313 @ SOMERSET	1.3010144	103.8383607
2	Aperia	APERIA	1.3097113	103.8643265
3	Balestier Hill Shopping Centre	BALESTIER 288	1.32297698121569	103.852811354547
4	Bugis Cube	BUGIS MRT STATION	1.30026468984101	103.855614760658

Here we have successfully extracted details regarding the malls in Singapore

Obtaining Housing locations

The final data that we need to gather is the housing lat and long locations of each of our HDB flats. And to do this we can use one map as well

We will need to get all of the blocks and street names combined, so that we can query the API properly. However we can also remove the duplicate addresses to prevent repeat searches

In [22]:
```
df['address'] = df['block'] + ' ' + df['street_name'] + ' '+ 'SINGAPORE'
df_dedup = df.drop_duplicates(subset='address', keep='first')
len(df_dedup)

# Next let's grab the unique addresses and create a list
address_list = df_dedup['address'].tolist()
len(address_list)
```

Out[22]: 9838

In [ ]:
```
hdb_building = []
hdb_lat = []
hdb_long = []
headers = {"Authorization": "Bearer **********"}  # API token
# Query and look for location using code as it is more accurate
for hdb in tqdm(address_list, desc="Fetching HDB coordinates"):
    url = f"https://www.onemap.gov.sg/api/common/elastic/search?searchVal={hdb}%MALL&returnGeom=Y&getAddrDetails=Y&pageNum=1"  # Modify for ma
    response = requests.get(url, headers=headers)
    try:
```

```
        data_hdb = json.loads(response.content)
    except:
        hdb_building.append('NotFound')
        hdb_lat.append('NotFound')
        hdb_long.append('NotFound')
        continue

    if data_mall['found'] != 0:
        hdb_lat.append(data_hdb["results"][0]["LATITUDE"])
        hdb_long.append(data_hdb["results"][0]["LONGITUDE"])

        # print(f"{query_address}, Lat: {data_mall['results'][0]['LATITUDE']} Long: {data_mall['results'][0]['LONGITUDE']}")
    else:
        hdb_building.append('NotFound')
        hdb_lat.append('NotFound')
        hdb_long.append('NotFound')

# Store this information in a dataframe
hdb_location = pd.DataFrame({
    'hdb': address_list,
    'latitude': hdb_lat,
    'longitude': hdb_long,
})

hdb_location.head()  # Display the first few rows
```

```
Fetching HDB coordinates: 100%|██████████| 9838/9838 [35:48<00:00,  4.58it/s]
```

Out[ ]:

	hdb	latitude	longitude
0	170 ANG MO KIO AVE 4 SINGAPORE	1.37914986228487	103.84093819913
1	174 ANG MO KIO AVE 4 SINGAPORE	1.37914986228487	103.84093819913
2	216 ANG MO KIO AVE 1 SINGAPORE	1.36619678831054	103.841505011903
3	215 ANG MO KIO AVE 1 SINGAPORE	1.36655830166122	103.841624082978
4	218 ANG MO KIO AVE 1 SINGAPORE	1.36619678831054	103.841505011903

In [24]:
```
hdb_location[hdb_location['latitude'] == 'NotFound']
```

Out[24]:

	hdb	latitude	longitude

Save all location data to csv

In [ ]:
```
hdb_location.to_csv('../datasets/hdb_location.csv',index=False)
mrt_location.to_csv('../datasets/mrt_location.csv',index=False)
mall_location.to_csv('../datasets/mall_location.csv',index=False)
```

In [ ]:
```

```