Sample Collection

We merged private property transaction data from URA between 2020–2025 and supplemented it with:

- Geolocation Data: Used OneMap API to map coordinates for each property and obtain the coorodinates of the nearest MRT stations.
- Distance to MRT: Calculated the distance using the coordinates of the property and the nearest MRT stations.

```python
import pandas as pd
import requests
import time
import glob
import os
from bs4 import BeautifulSoup
from geopy.distance import geodesic

# For private property dataset
folder_path = r"C:\Users\felic\OneDrive\Code\DSAI\Project dataset combi"

# Get a list of all CSV files in the folder
csv_files = glob.glob(os.path.join(folder_path, "*.csv"))

# Read and combine all CSV files with error handling for encoding issues
df_list = []
for file in csv_files:
    try:
        df = pd.read_csv(file, encoding="utf-8")  # Try reading with UTF-8
    except UnicodeDecodeError:
        df = pd.read_csv(file, encoding="ISO-8859-1")  # Fallback to ISO-8859-1
    df_list.append(df)

# Combine all DataFrames
df_combined = pd.concat(df_list, ignore_index=True)

# Save the combined DataFrame to a new CSV file
output_path = os.path.join(folder_path, "combined_dataset.csv")
df_combined.to_csv(output_path, index=False, encoding="utf-8")

print(f"CSV files successfully combined and saved as '{output_path}'.")
```

```
# Getting coordinates
df = pd.read_csv("private_2020-01.csv")

# Mapping coordinates
token_key =
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJmNTkyNjg4NzAxNTI0MDFi
MmE2NjM3NjQ3N2M1MzU5MSIsImlzcyI6Imh0dHA6Ly9pbnRlcm5hbC1hbGItb20tcHJkZX
ppdC1pdC1uZXctMTYzMzc5OTU0Mi5hcC1zb3V0aGVhc3QtMS5lbGIuYW1hem9uYXdzLmNv
bS9hcGkvdjIvdXNlci9wYXNzd29yZCIsImlhdCI6MTc0MzA2ODQyNCwiZXhwIjoxNzQzMz
I3NjI0LCJuYmYiOjE3NDMwNjg0MjQsImp0aSI6IkwwWDVnWjRxMURpdmFEMEUiLCJ1c2Vy
X2lkIjo2NTc1LCJmb3JldmVyIjpmYWxzZX0.l56pqZIjM4ASobS1dAbTEpavNyZoBi7ov7
IEBNd7Lec"

def get_location_data(name, street, cache):
    search_term = f"{name} {street}"
    if search_term in cache:
        return cache[search_term]  # Return cached result to avoid
duplicate queries

    url = "https://www.onemap.gov.sg/api/common/elastic/search"
    params = {"searchVal": search_term, "returnGeom": "Y",
"getAddrDetails": "Y"}
    headers = {"Authorization": token_key}

    response = requests.get(url, params=params, headers=headers)
    if response.status_code == 200:
        data = response.json()
        if data["found"] > 0:
            result = data["results"][0]
            cache[search_term] = (result["X"], result["Y"],
result["LONGITUDE"], result["LATITUDE"])
            return cache[search_term]

    cache[search_term] = (None, None, None, None, None)  # Cache
failed lookups too
    return cache[search_term]

# Dictionary to store already fetched results
cache = {}

# Apply function with caching
df[["postal_code", "x", "y", "longitude", "latitude"]] = df.apply(
    lambda row: pd.Series(get_location_data(row['Project Name'],
row["Street Name"], cache)), axis=1
)

# Save the updated CSV
df.to_csv("private_with_api.csv", index=False)
```

```python
# URL of the MRT stations list
url = "https://mrtmapsingapore.com/mrt-stations-singapore/"

# Fetch the page content
response = requests.get(url)
soup = BeautifulSoup(response.content, "html.parser")

# Find the table containing the station data
station_table = soup.find("table")

# Extract station names
station_names = []
for row in station_table.find_all("tr")[1:]:  # Skip header row
    cols = row.find_all("td")
    if cols:
        station_name = cols[1].text.strip()  # Station name is in the
second column
        station_names.append(station_name)

print(f"Retrieved {len(station_names)} MRT station names.")


# Function to get coordinates from OneMap API
def get_coordinates(station_name):
    base_url = "https://www.onemap.gov.sg/api/common/elastic/search"
    params = {
        "searchVal": station_name + " MRT Station",
        "returnGeom": "Y",
        "getAddrDetails": "N",
        "pageNum": 1
    }
    response = requests.get(base_url, params=params)
    results = response.json().get("results", [])
    if results:
        lat = results[0]["LATITUDE"]
        lon = results[0]["LONGITUDE"]
        return float(lat), float(lon)
    else:
        return None, None

# Retrieve coordinates for each station
mrt_data = []
for station in station_names:
    lat, lon = get_coordinates(station)
    print(f"{station}: {lat}, {lon}")
    mrt_data.append({"Station": station, "Latitude": lat, "Longitude":
lon})
    time.sleep(0.2)  # To avoid overwhelming the API

# Convert to DataFrame
```

```
mrt_df = pd.DataFrame(mrt_data)

# Save to CSV
mrt_df.to_csv("mrt_stations_coordinates.csv", index=False)
print("Saved MRT station coordinates to
'mrt_stations_coordinates.csv'.")

# Load MRT station coordinates
private_data = "../datasets/cleaned/cleaned_private.csv"

df = pd.read_csv(private_data, quotechar='"', escapechar='\\',
thousands=',')
mrt_df = pd.read_csv("mrt_stations_coordinates.csv")

# Function to get nearest station distance (vectorized)
def find_nearest_mrt(lat, lon):
    min_dist = float('inf')
    for _, mrt in mrt_df.iterrows():
        station_coord = (mrt['Latitude'], mrt['Longitude'])
        property_coord = (lat, lon)
        dist = geodesic(property_coord, station_coord).km
        if dist < min_dist:
            min_dist = dist
    return min_dist

# Compute distances
df['Distance to MRT (km)'] = df.apply(
    lambda row: find_nearest_mrt(row['latitude'], row['longitude']) if
pd.notna(row['latitude']) and pd.notna(row['longitude']) else None,
    axis=1
)

# Save result
df.to_csv("properties_with_mrt_distance.csv", index=False)
```