A Multiple Linear Regression approach towards HDB Housing price prediction.

By: Loo Guan Yee

HDB Primer

Founded in 1960, HDB is a statutory board under the Ministry of National Development. In its early years, HDB focused on modernising the Singapore housing landscape by facilitating the movement of Singaporeans from traditional housing models such as Kampongs and Shophouses. In modern times, HDB has shifted its focus to upgrading and redevelopment of maturing estates. Notable examples are the Lift Upgrading Schemes of maturing estates and the construction of Build Order flats in a mature estate. HDB has kept a live collection of HDB transactions from 1990 to the current on the data.gov.sg website.

Aim

We will be focusing on the data collected from 2015 onwards for our multiple regression analysis. We will explore the relationship between factors and resales HDB price in both *blinded* and *unblinded* (where the town names are considered as additional variables in the Multiple Linear Regression) settings.

Explanation of each HDB column.

The following datasets (HDB 2021) will be downloaded for analysis.

- resale-flat-prices-based-on-registration-date-from-Jan-2015-to-dec-2016.csv
- resale-flat-prices-based-on-registration-date-from-Jan-2017-onwards.csv

There are 8 factors in the CSV that might affect the HDB resale prices. The factors are

- month of resale
- town
- flat_type
- street name
- storey
- floor_area
- remaining_lease

Before we pre-process the data, we dissect the terminologies found on both CSV files.

Month of Transaction

- It records the year and the month when the HDB flat transaction was made.

Street Name

- It refers to a particular road or lane in Singapore. Ang Mo Kio Avenue 4 is an example of a street name in Singapore
- Street Name allows us to compute the distance to the nearest MRT Station and CBD (Central Business District).
- In this regression analysis, we will consider the CBD area to be located at Raffles Place MRT Station.

Town

- A town consists of multiple streets with the town name. For example, Ang Mo Kio Avenue 4, Ang Mo Kio Avenue 9 are in the town of Ang Mo Kio.
- There are 16 towns in Singapore.
- The town factor will be omitted in the first multiple regression analysis to enforce blinding. The town factor will be added to the second dataset.

Flat Types

- The Flat_type refers to the number of rooms inside a flat.
- Examples of flat types are the 2 rooms, 3 rooms, 4 rooms, 5 rooms, Executive and Multi-Generations.
- The Multi-Generations flats are discontinued in the 2000s. Executive flats come in the form of 3 or 4 or even 5 rooms. We will narrow down to the entries with 2,3,4,5 rooms for consistency purposes. Hence, the Executive and Multi-Generations flat transactions are deliberately excluded from the analysis.

Storey

- The storey refers to a specific level in the HDB flat.
- Higher levels command a higher price premium over the middle and lower levels HDB blocks.
- In this analysis, we will investigate the extent of the increase in the resale price when the storey increases.

Floor Area

- The floor area refers to the size of the HDB flat.
- A higher floor area means that there is more space in the HDB flat.
- We will investigate the extent of the increase of floor area on the resale prices.

Remaining Year

- Each HDB flat has a lease year of 99 years.
- When the lease expires after 99 years, the flat will be returned to HDB at 0 costs (HDB, 2020).
- Newer flats will command higher premiums over the older ones since they have higher remaining lease years before lease expiry.
- We will investigate the extent of the increase of remaining lease years on the resale prices.

Data pre-processing

Before we apply multiple linear regression analysis, we will be performing the data pre-processing. The data pre-processing steps will be broken down into the following steps

1) Read the CSV and merge the 2 CSV files.
2) Create a nested dictionary of MRT stations names as primary keys, latitude and longitude as second keys which map to their coordinates.
3) Create a nested dictionary of the street name as a primary key, latitude and longitude as secondary keys which map to their coordinates.
4) Create a dictionary of town name as key and the NumPy array index as the value
5) Create getter functions to retrieve the specific values
6) Initialise the empty Numpy array of blinded and unblinded and use the getter function to populate both arrays.
7) Apply feature scaling to both NumPy Arrays.

1) CSV preparations

We will implement the function to read CSV.

```
# We will import the relevant files
import csv
import numpy as np

# read_csv function
def read_csv(csvfilename):
    rows = []
    with open(csvfilename) as csvfile:
        file_reader = csv.reader(csvfile)
        for row in file_reader:
            rows.append(row)
    # convert the list into Numpy array
    rows = np.array(rows)
    return rows
```

We will read CSV for two CSV files.

```
firstData = read_csv("resale-flat-prices-based-on-registration-date-from-jan-2015-to-dec-2016.csv")
secondData = read_csv("resale-flat-prices-based-on-registration-date-from-jan-2017-onwards.csv")

firstData
```

```
array([['month', 'town', 'flat_type', ..., 'lease_commence_date',
        'remaining_lease', 'resale_price'],
       ['2015-01', 'ANG MO KIO', '3 ROOM', ..., '1986', '70', '255000'],
       ['2015-01', 'ANG MO KIO', '3 ROOM', ..., '1981', '65', '275000'],
       ...,
       ['2016-12', 'YISHUN', 'EXECUTIVE', ..., '1992', '74', '778000'],
       ['2016-12', 'YISHUN', 'EXECUTIVE', ..., '1988', '70', '575000'],
       ['2016-12', 'YISHUN', 'MULTI-GENERATION', ..., '1987', '70',
        '735000']], dtype='<U22')
```

```
secondData
```

```
array([['month', 'town', 'flat_type', ..., 'lease_commence_date',
        'remaining_lease', 'resale_price'],
       ['2017-03', 'BUKIT MERAH', '1 ROOM', ..., '1975', '57 years',
        '200000'],
       ['2017-03', 'BUKIT MERAH', '1 ROOM', ..., '1975', '57 years',
        '218000'],
       ...,
       ['2021-11', 'YISHUN', 'MULTI-GENERATION', ..., '1987',
        '65 years 01 month', '838000'],
       ['2021-03', 'YISHUN', 'MULTI-GENERATION', ..., '1987',
        '65 years 10 months', '860000'],
       ['2021-10', 'YISHUN', 'MULTI-GENERATION', ..., '1987',
        '65 years 01 month', '760000']], dtype='<U22')
```

We will then combine the 2 CSV using the NumPy concatenation function.

```
# We will now join the csv together from 2015 to 2021
data = np.concatenate((firstData,secondData[1:]))
data
```

```
array([['month', 'town', 'flat_type', ..., 'lease_commence_date',
        'remaining_lease', 'resale_price'],
       ['2015-01', 'ANG MO KIO', '3 ROOM', ..., '1986', '70', '255000'],
       ['2015-01', 'ANG MO KIO', '3 ROOM', ..., '1981', '65', '275000'],
       ...,
       ['2021-11', 'YISHUN', 'MULTI-GENERATION', ..., '1987',
        '65 years 01 month', '838000'],
       ['2021-03', 'YISHUN', 'MULTI-GENERATION', ..., '1987',
        '65 years 10 months', '860000'],
       ['2021-10', 'YISHUN', 'MULTI-GENERATION', ..., '1987',
        '65 years 01 month', '760000']], dtype='<U22')
```

We will remove the ‘multi-generation’ and ‘executive’ flats from the data NumPy array.

```
# 'MULTI-GENERATION' and 'EXECUTIVE' do not have a fixed number of rooms.
# We will exclude these entries from the data.

def removeUnlabelledData(data):
    data1 = []
    for i in range(1,len(data)):
        row = data[i]
        # row with executive and multi-generation
        if row[2] == 'EXECUTIVE' or row[2] == 'MULTI-GENERATION':
            continue;
        else:
            data1.append(row)

    # convert the list to numpy array
    data1 = np.array(data1)
    return data1

data1 = removeUnlabelledData(data)
data1
```

```
array([['2015-01', 'ANG MO KIO', '3 ROOM', ..., '1986', '70', '255000'],
       ['2015-01', 'ANG MO KIO', '3 ROOM', ..., '1981', '65', '275000'],
       ['2015-01', 'ANG MO KIO', '3 ROOM', ..., '1980', '64', '285000'],
       ...,
       ['2021-11', 'YISHUN', '5 ROOM', ..., '1988', '65 years 04 months',
        '580000'],
       ['2021-11', 'YISHUN', '5 ROOM', ..., '1988', '66 years 01 month',
        '645000'],
       ['2021-12', 'YISHUN', '5 ROOM', ..., '1987', '64 years 11 months',
        '658000']], dtype='<U22')
```

2) Create a nested dictionary of MRT Stations coordinates

We will add the MRT and LRT Station into a consolidated CSV file and read the CSV file.

```
# We will retrieve the MRT Station from the MRT CSV file
mrtData = read_csv("MRTstations.csv")
mrtData
```

```
array([['ï»¿Station Code', 'MRT Station Name', 'Line Color', 'Line Name'],
       ['NS1', 'Jurong East', 'Red', 'North South'],
       ['NS2', 'Bukit Batok', 'Red', 'North South'],
       ['NS3', 'Bukit Gombak', 'Red', 'North South'],
       ['NS4 BP1', 'Choa Chu Kang', 'Red', 'North South'],
       ['NS5', 'Yew Tee', 'Red', 'North South'],
       ['NS7', 'Kranji', 'Red', 'North South'],
       ['NS', 'Marsiling', 'Red', 'North South'],
       ['NS9 TE2', 'Woodlands', 'Red', 'North South'],
       ['NS10', 'Admiralty', 'Red', 'North South'],
       ['NS11', 'Sembawang', 'Red', 'North South'],
       ['NS12', 'Canberra', 'Red', 'North South'],
       ['NS13', 'Yishun', 'Red', 'North South'],
       ['NS14', 'Khatib', 'Red', 'North South'],
       ['NS15', 'Yio Chu Kang', 'Red', 'North South'],
       ['NS16', 'Ang Mo Kio', 'Red', 'North South'],
       ['NS17 CC15', 'Bishan', 'Red', 'North South'],
       ['NS18', 'Braddell', 'Red', 'North South'],
       ['NS19', 'Toa Payoh', 'Red', 'North South'],
```

2.1) MRT station dictionary creation function

- We will iterate through the MRTData array.
- MRT and LRT stations will be filtered by the line code. ‘BP’, ‘Pu’ and ‘SK’ correspond to the LRT Station lines.
- Then, we will use the OneMap API (SLA, 2016) to retrieve the results for the MRT Locations.

```
Usage:

 /commonapi/search?searchVal={SearchText}&returnGeom={Y/N}&getAddrDetails={Y/N}&pageNum={PageNumber}
```

We will use JSON to convert the result into the dictionary and retrieve the latitude and longitude.

```python
# library needed for the operation to happen
import json
import requests

# We will create a function to return dictionary of MRT stations with their longitudes and lattitudes
def createStationLocation(data):
    # create dictionary
    mrtStationCoordinates = {}
    # iterate through the stations in mrtData
    for i in range(1,len(data)):
        row = data[i]
        # LRT Station
        if (row[2] == 'SK' or row[2] == 'Pu' or row[2] =='BP'):
            stationName = row[1] + ' LRT STATION'
        # MRT Station
        else:
            stationName = row[1] + ' Mrt station'
        # print(stationName)
        # we will use the Station code to query the longitude and lattitude from the onemap api provided by SLA
        link = "https://developers.onemap.sg/commonapi/search?searchVal=%s&returnGeom=Y&getAddrDetails=Y" % stationName
        # print(link)
        resp = requests.get(link)

        # print(json.loads(resp.content))
        # we will narrow down to the the result section from the dictionary
        query = json.loads(resp.content)['results'][0]
        # print(query)

        # initialise the nested dictionary
        mrtStationCoordinates[row[1]] = {}

        # add the lattitude and convert to float value
        mrtStationCoordinates[row[1]]['Latitude'] = float(query['LATITUDE'])

        # add the longitude and convert to float value
        mrtStationCoordinates[row[1]]['Longtitude'] = float(query['LONGTITUDE'])

    return mrtStationCoordinates

mrtStationCoordinates = createStationLocation(mrtData)
mrtStationCoordinates
```

Output:

```
{'Jurong East': {'Latitude': 1.33315281585758, 'Longtitude': 103.742286332403},
 'Bukit Batok': {'Latitude': 1.34903331201636, 'Longtitude': 103.749566478309},
 'Bukit Gombak': {'Latitude': 1.35861159094192,
  'Longtitude': 103.751790910733},
 'Choa Chu Kang': {'Latitude': 1.38536316540225,
  'Longtitude': 103.744370779756},
 'Yew Tee': {'Latitude': 1.39753506936297, 'Longtitude': 103.747405150236},
 'Kranji': {'Latitude': 1.42508698073648, 'Longtitude': 103.762137459497},
 'Marsiling': {'Latitude': 1.43252114855026, 'Longtitude': 103.774074641403},
 'Woodlands': {'Latitude': 1.43605761708128, 'Longtitude': 103.787938777173},
 'Admiralty': {'Latitude': 1.44058856161847, 'Longtitude': 103.800990519771},
 'Sembawang': {'Latitude': 1.44905082158502, 'Longtitude': 103.820046140211},
 'Canberra': {'Latitude': 1.44307664075699, 'Longtitude': 103.829702590959},
 'Yishun': {'Latitude': 1.42944308477331, 'Longtitude': 103.835005047246},
 'Khatib': {'Latitude': 1.41738337009565, 'Longtitude': 103.832979908243},
 'Yio Chu Kang': {'Latitude': 1.38175587099132, 'Longtitude': 103.84494727118},
 'Ang Mo Kio': {'Latitude': 1.36993284962264, 'Longtitude': 103.849558091776},
 'Bishan': {'Latitude': 1.35131580146658, 'Longtitude': 103.84914026532},
 'Braddell': {'Latitude': 1.3404690010277, 'Longtitude': 103.846799083148},
```

3) Street names dictionary creation

In this component, we will create 3 functions.

1) Create a function that takes in 2 distinct latitude and longitude coordinates and return the distance using Haversine Formula
2) Create a function that takes in latitude and longitude of the street name and the mrtStationCoordinates Dictionary that we have created earlier and return a list.
3) Create a function that takes in the dataset and return the mrtStationCoordinates dictionary.

3.1) Haversine calculation function

We will use the haversine formula (Baker, 1995) to compute the distance. *Note that the longitude and latitude must be converted into radians first before the calculation.

Haversine Formula (from R.W. Sinnott, "Virtues of the Haversine", Sky and Telescope, vol. 68, no. 2, 1984, p. 159):

dlon = lon2 - lon1
dlat = lat2 - lat1
a = (sin(dlat/2))^2 + cos(lat1) * cos(lat2) * (sin(dlon/2))^2
c = 2 * arcsin(min(1,sqrt(a)))
d = R * c

```
import math
from math import radians,cos,sin,asin
# function that return the distance between 2 coordinates
def distanceCalculator(latitude1, longitude1,latitude2, longitude2):
    deltaLon = (longitude2- longitude1) * (math.pi/180)
    deltaLat = (latitude2 - latitude1) * (math.pi/180)
    a = (math.sin(deltaLat/2))**2 + cos(latitude1) * cos(latitude2)* ((sin(deltaLon/2))**2)

    c = 2 * math.asin(min(1,math.sqrt(a)))

    earthRadius = 6371 * 1000

    # find the distance
    distance = earthRadius * c

    return distance
```

3.2) Information creation function

We will first initialise the 3 variables nearest MRT, nearestDistanceToMRT, nearestDistanceToCBD and an empty list called information.

```
    nearestMRT = ''
    nearestDistanceToMRT = 10000000
    nearestDistanceToCBD = 0
    information = []
```

We will iterate through the mrtStationCoordinates dictionary. If the temp distance is shorter than the previously recorded distance, we will update it with the new lower distance value and the new name for the nearestMRTStation variable.

```
temp = 0
# we will iterate through the mrtStationCoordinates
for key in mrtStationCoordinates:
    stationLatitude = mrtStationCoordinates[key]['Latitude']
    stationLongitude = mrtStationCoordinates[key]['Longtitude']
    temp = distanceCalculator(latitude,longitude,stationLatitude,stationLongitude)
    if temp < nearestDistanceToMRT:
        # reupdate the nearestDistanceToMRT
        nearestDistanceToMRT = temp
        # reupdate the stationame
        nearestMRT = key
```

Next, we will then calculate the nearest distance to CBD by measuring the distance to the Raffles Place MRT Station.

```
# find the nearestDistanceToCBD
nearestDistanceToCBD = distanceCalculator(latitude,longitude,
                                          mrtStationCoordinates['Raffles Place']['Latitude'],
                                          mrtStationCoordinates['Raffles Place']['Longtitude'])
```

Then, we will add the 3 variables into a list and return the list.

```
information.append(nearestMRT)
information.append(nearestDistanceToMRT)
information.append(nearestDistanceToCBD)

# return the list
return information
```

3.3) streetName Dictionary function creation

This function will take in the data and mrtStationCoordinates Dictionary.

We will first initialise the street name location dictionary

```
# intialise the dictionary
streetNameLocation = {}
```

We will iterate through the data. The street name corresponds to the index 4 in the row

```
# We will iterate the data
for i in range(1,len(data)):
    row = data[i]
    streetName = row[4]
```

Create the link using OneMap API and Street Name entry in the data.

**Note that OneMap API will return 0 results for the "ST. GEORGE'S RD". We will use its postal code 60241 to prevent errors due to an empty dictionary.*

```
# ST George road
if row[4] == "ST. GEORGE'S RD":
    # We will use this bus code instead
    postalCode = 60241
    link = "https://developers.onemap.sg/commonapi/search?searchVal=%s&returnGeom=Y&getAddrDetails=Y" % postalCode
else:
    # We use onemap api to find the streetname coordinates
    link = "https://developers.onemap.sg/commonapi/search?searchVal=%s&returnGeom=Y&getAddrDetails=Y" % streetName

resp = requests.get(link)
```

We will use JSON to generate a dictionary from the link query. The JSON will produce a dictionary containing the longitude and latitude.

{'found': 2, 'totalNumPages': 1, 'pageNum': 1, 'results': [{'SEARCHVAL': 'MY FIRST SKOOL', 'BLK_NO': '541', 'ROAD_NAME': 'ANG MO KIO AVENUE 10', 'BUILDING': 'MY FIRST SKOOL', 'ADDRESS': '541 ANG MO KIO AVENUE 10 MY FIRST SKOOL SINGAPORE 560541', 'POSTAL': '560541', 'X': '30482.026488265', 'Y': '39546.8841999061', 'LATITUDE': '1.37392238703482', 'LONGITUDE': '103.855621370524', 'LONGTITUDE': '103.855621370524'}, {'SEARCHVAL': 'CHENG SAN GREEN', 'BLK_NO': '541', 'ROAD_NAME': 'ANG MO KIO AVENUE 10', 'BUILDING': 'CHENG SAN GREEN', 'ADDRESS': '541 ANG MO KIO AVENUE 10 CHENG SAN GREEN SINGAPORE 560541', 'POSTAL': '560541', 'X': '30482.026548801', 'Y': '39546.8847144619', 'LATITUDE': '1.37392239168826', 'LONGITUDE': '103.855621371068', 'LONGTITUDE': '103.855621371068'}]}

We will narrow down to the resulting key from the dictionary

```
# we will narrow down to the the result section from the dictionary
query = json.loads(resp.content)['results'][0]
```

We will call the function findNearestMRTStation using the latitude and longitude sourced from the API.

```
# retrieve the list from this nearest MRT Station
information = findNearestMRTStation(float(query['LATITUDE']),float(query['LONGTITUDE']),mrtStationCoordinates)
```

We will create the street name key which will map to the 3 variables returned from the information list.

```
        # create a nested dictionary
        streetNameLocation[row[4]] = {}

        # add the nearest MRT station into the dictionary -- this is for debugging if any
        streetNameLocation[row[4]]["Nearest MRT"] = information[0]

        # add the nearest distance into the dictionary -- this will be used later
        streetNameLocation[row[4]]["Nearest Distance To MRT/LRT"] = information[1]

        # add the nearest distance to CDB into the dictionary -- this will be used later
        streetNameLocation[row[4]]["Nearest Distance to CBD"] = information[2]

    return streetNameLocation
```

4) Dictionary Location creation (for unblinded dataset)

The town is a categorical variable. Hence, we will apply one hot encoding (dummy variable) technique in this unblinded dataset. Each town name will map to a unique index in each row in the NumPy array.

```
# Return dictionary of town names that map to the location in the array
def retrieveTownNames(data1):
    townName = {}
    count = 7
    for i in range(1,len(data1)):
        row = data1[i]
        if row[1] not in townName:
            townName[row[1]] = count
            # inclement count + 1
            count += 1
        else:
            continue
    return townName
townName = retrieveTownNames(data1)
townName
```

```
{'ANG MO KIO': 7,
 'BEDOK': 8,
 'BISHAN': 9,
 'BUKIT BATOK': 10,
 'BUKIT MERAH': 11,
 'BUKIT PANJANG': 12,
 'BUKIT TIMAH': 13,
 'CENTRAL AREA': 14,
 'CHOA CHU KANG': 15,
 'CLEMENTI': 16,
 'GEYLANG': 17,
 'HOUGANG': 18,
 'JURONG EAST': 19,
 'JURONG WEST': 20,
 'KALLANG/WHAMPOA': 21,
 'MARINE PARADE': 22,
 'PASIR RIS': 23,
 'PUNGGOL': 24,
 'QUEENSTOWN': 25,
 'SEMBAWANG': 26,
 'SENGKANG': 27,
 'SERANGOON': 28,
 'TAMPINES': 29,
 'TOA PAYOH': 30,
 'WOODLANDS': 31,
 'YISHUN': 32}
```

5) Retrieval Functions

The blinded and unblinded datasets will use the following retrieve functions for the dataset.

1) retrieveYear
2) retrieveRoom
3) retrieveMeanStorey
4) retrieveFloorArea
5) retrieveRemainingLease
6) retrieveDistanceToMRT
7) retrieveDistanceToCbd
8) retrievePrice (for dependent variable)

5.1) retrieveYear

In the CSV records, the time is recorded in the year/month order and string format. We will slice the string to get the year in float format, then we will slice the string to get the month in float format. Lastly, we will add the time together and return the time.

```
def retrieveYear(entry):
    year = float(entry[:4])
    months = float(entry[5:7])/13
    time = year + months
    return time
```

5.2) retrieveRoom

In the Room column of the dataset, we will slice the string to obtain the number of rooms.

```
def retrieveRoom(entry):
    return float(entry[0])
```

5.3) retrieveMeanStorey

A range for the storey of HDB flats is given instead of a specific storey. Hence, we will obtain the average storey of the range given.

```
def retrieveMeanStorey(entry):
    # We will focus on the first 2 digits of the entry
    firstPart = entry[0]
    secondPart = entry[1]

    # single digit storey
    if firstPart == '0':
        # convert second digit to float
        startStorey = float(secondPart)
        mean = (startStorey + startStorey + 2)/2
        return mean
    else:
        # convert first and second digit to float
        startStorey = float(entry[:2])
        mean = (startStorey + startStorey + 2)/2
        return mean
```

5.4) retrieveFloorArea

We will convert the string into float format and return the floor area value.

```
def retrieveFloorArea(entry):
    return float(entry)
```

5.5) retrieveRemainingLease(entry)

The remaining Lease column is presented in the XX years and XX months. Some of the rows do not have months on them. Hence, the retrieval function must consider instances where there are only

- no words are present. (from 2015 to 2016 dec.csv file)
- only 'years' is present.
- both 'years' and 'months' are present.
- Redundant months.

```
def retrieveRemainingLease(entry):
    # only digits
    if 'years' in entry:
        if 'months' in entry:
            years = float(entry[:2])
            months = entry[9:12]
            # bad entry '0 m' representation in 2017 to 2021 csv
            if months == '0 m':
                return years
            else:
                months = float(entry[9:12])/12
                time = years + months
                return time
        else:
            years = float(entry[:2])
            return years
    else:
        return float(entry)
```

5.6) retrieveDistanceToMrt

We will retrieve the distance to the nearest MRT station using streetNameLocation dictionary

```
def retrieveDistanceToMrt(entry,streetNameLocation):
    return streetNameLocation[entry]["Nearest Distance To MRT/LRT"]
```

5.7) retrieveDistanceToMrt

We will retrieve the distance to the CBD using the streetNameLocation dictionary

```
def retrieveDistanceToCbd(entry,streetNameLocation):
    return streetNameLocation[entry]['Nearest Distance to CBD']
```

5.8) retrievePrice (dependent variable)

We will convert the string value into float format and return it.

```
def retrievePrice(entry):
    return float(entry)
```

6) Populate the NumPy array

6.1) Initialisation

We will now initialise both the blinded and unblinded NumPy array. The blinded array should have 7 columns in each row while the unblinded array will have 33 columns.

blinded

```
# We will now initialise the numpy with zeros
# https://www.dataquest.io/blog/numpy-tutorial-python/
xBlinded = np.zeros((len(data1),7))
xBlinded
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

Unblinded

```
# Create the unblinded dataset
# We will now initialise the numpy with zeros
xUnblinded = np.zeros((len(data1),33))
xUnblinded
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

Lastly, we will create our dependent variable Y.

```
y = np.zeros((len(data1),1))
y
```

```
array([[0.],
       [0.],
       [0.],
       ...,
       [0.],
       [0.],
       [0.]])
```

6.2) Populate the Data

We will now use the retrieval functions that we have created earlier to populate the blinded dataset.

```
# We will repopulate the data with independent and dependent variables
def populateBlindedX(data,xBlinded,streetNameLocation):
    for i in range(1,len(data)):
        row = data[i]
        # add the year
        xBlinded[i-1][0] = retrieveYear(row[0])

        # add the room number
        xBlinded[i-1][1] = retrieveRoom(row[2])

        # add the storey
        xBlinded[i-1][2] = retrieveMeanStorey(row[5])

        # add the floor area
        xBlinded[i-1][3] = retrieveFloorArea(row[6])

        # add the remaining lease
        xBlinded[i-1][4] = retrieveRemainingLease(row[9])

        # add the Distance to MRT/LRT
        xBlinded[i-1][5] = retrieveDistanceToMrt(row[4],streetNameLocation)

        # add the distance to CBD
        xBlinded[i-1][6] = retrieveDistanceToCbd(row[4],streetNameLocation)

    return xBlinded

xBlinded = populateBlindedX(data1,xBlinded,streetNameLocation)
np.set_printoptions(suppress=True,precision = 2)
xBlinded
```

```
array([[ 2015.08,     3.  ,     2.  , ...,    65.  ,   741.18,  8753.96],
       [ 2015.08,     3.  ,     2.  , ...,    64.  ,   208.97, 10685.21],
       [ 2015.08,     3.  ,     2.  , ...,    63.  ,   741.18,  8753.96],
       ...,
       [ 2021.85,     5.  ,     5.  , ...,    66.  ,   225.9 , 14625.29],
       [ 2021.92,     5.  ,    11.  , ...,    64.92,   225.9 , 14625.29],
       [    0.  ,     0.  ,     0.  , ...,     0.  ,     0.  ,     0.  ]])
```

We will now populate our unblinded dataset. We will need to use row[1] which is the town to guide python to add 1 at the specific index of each row. This was done after adding the distance to the CBD.

```
# We will create our unblinded dataset

def populateUnblindedX(data,xUnblinded,streetNameLocation,townName):
    for i in range(1,len(data)):
        row = data[i]
        # add the year
        xUnblinded[i-1][0] = retrieveYear(row[0])

        # add the room number
        xUnblinded[i-1][1] = retrieveRoom(row[2])

        # add the storey
        xUnblinded[i-1][2] = retrieveMeanStorey(row[5])

        # add the floor area
        xUnblinded[i-1][3] = retrieveFloorArea(row[6])

        # add the remaining lease
        xUnblinded[i-1][4] = retrieveRemainingLease(row[9])

        # add the Distance to MRT/LRT
        xUnblinded[i-1][5] = retrieveDistanceToMrt(row[4],streetNameLocation)

        # add the distance to CBD
        xUnblinded[i-1][6] = retrieveDistanceToCbd(row[4],streetNameLocation)

        # add the 1 to the correct location
        xUnblinded[i-1][townName[row[1]]] = 1

    return xUnblinded

xUnblinded = populateUnblindedX(data1,xUnblinded,streetNameLocation,townName)
np.set_printoptions(suppress=True)
xUnblinded
```

```
array([[2015.08,    3.  ,    2.  , ...,    0.  ,    0.  ,    0.  ],
       [2015.08,    3.  ,    2.  , ...,    0.  ,    0.  ,    0.  ],
       [2015.08,    3.  ,    2.  , ...,    0.  ,    0.  ,    0.  ],
       ...,
       [2021.85,    5.  ,    5.  , ...,    0.  ,    0.  ,    1.  ],
       [2021.92,    5.  ,   11.  , ...,    0.  ,    0.  ,    1.  ],
       [   0.  ,    0.  ,    0.  , ...,    0.  ,    0.  ,    0.  ]])
```

Lastly, we will populate the dependent variable y with the corresponding resale prices.

```
def populateY(data,y):
    for i in range(1,len(data)):
        row = data[i]
        y[i-1][0] = float(row[10])
    return y

y = populateY(data1,y)
y
```

```
array([[275000.],
       [285000.],
       [290000.],
       ...,
       [645000.],
       [658000.],
       [     0.]])
```

7) Feature Scaling

By inspecting the unblinded and blinded dataset, the column with the largest magnitude is the distance to CBD. It is in the range of 4 to 5 digits. The column with the smallest magnitude is the number of rooms. We need to apply feature scaling to ensure that all the variables are represented in the regression with the same footing (Roy, 2020).

Hence, we will use the standardization function imported from the sklearn.preprocessing library.

```
# Feature scaling to ensure all the variables are equally represented
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

With the standardization function, the data is transformed to have a zero mean and a variance of 1, removing the units as well (Roy, 2020).

We will apply the feature scaling to the blinded dataset.

```
# Feature Scaling for the xBlinded dataset
xBlindedFS = sc.fit_transform(xBlinded)
xBlindedFS
```

```
array([[  -0.64,   -1.27,   -1.14, ...,   -0.73,    1.94,   -0.14],
       [  -0.64,   -1.27,   -1.14, ...,   -0.81,   -0.46,    0.29],
       [  -0.64,   -1.27,   -1.14, ...,   -0.89,    1.94,   -0.14],
       ...,
       [   0.54,    1.32,   -0.63, ...,   -0.66,   -0.38,    1.17],
       [   0.55,    1.32,    0.39, ...,   -0.74,   -0.38,    1.17],
       [-350.47,   -5.15,   -1.48, ...,   -5.66,   -1.4 ,   -2.1 ]])
```

We will apply feature scaling to the unblinded dataset.

```
# Feature scaling for the xUnblinded
# Feature Scaling for the overall X dataset
xUnblindedFS = sc.fit_transform(xUnblinded)
xUnblindedFS
```

```
array([[  -0.64,   -1.27,   -1.14, ...,   -0.19,   -0.27,   -0.27],
       [  -0.64,   -1.27,   -1.14, ...,   -0.19,   -0.27,   -0.27],
       [  -0.64,   -1.27,   -1.14, ...,   -0.19,   -0.27,   -0.27],
       ...,
       [   0.54,    1.32,   -0.63, ...,   -0.19,   -0.27,    3.76],
       [   0.55,    1.32,    0.39, ...,   -0.19,   -0.27,    3.76],
       [-350.47,   -5.15,   -1.48, ...,   -0.19,   -0.27,   -0.27]])
```

8) Application of Multiple Linear Regression

Formula and Calculation of Multiple Linear Regression

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} + \epsilon$$

where, for $i = n$ observations:

$y_i =$ dependent variable

$x_i =$ explanatory variables

$\beta_0 =$ y-intercept (constant term)

$\beta_p =$ slope coefficients for each explanatory variable

$\epsilon =$ the model's error term (also known as the residuals)

Multiple Linear Regression Formula(Hayes, 2021)

By definition of multiple linear regression, we need to add the y-intercept(constant term) into the dataset as well.

We will add a constant to the xBlinded dataset.

```
import statsmodels.api as sm

# Add the constant variable once
xBlindedFS1 = sm.add_constant(xBlindedFS)
xBlindedFS1
```

```
array([[   1.  ,   -0.64,   -1.27, ...,   -0.73,    1.94,   -0.14],
       [   1.  ,   -0.64,   -1.27, ...,   -0.81,   -0.46,    0.29],
       [   1.  ,   -0.64,   -1.27, ...,   -0.89,    1.94,   -0.14],
       ...,
       [   1.  ,    0.54,    1.32, ...,   -0.66,   -0.38,    1.17],
       [   1.  ,    0.55,    1.32, ...,   -0.74,   -0.38,    1.17],
       [   1.  , -350.47,   -5.15, ...,   -5.66,   -1.4 ,   -2.1 ]])
```

We will add a constant to the xUnblinded dataset.

```
# We will create our constant to the X column
xUnblindedFS1 = sm.add_constant(xUnblindedFS)
xUnblindedFS1
```

```
array([[   1.  ,   -0.64,   -1.27, ...,   -0.19,   -0.27,   -0.27],
       [   1.  ,   -0.64,   -1.27, ...,   -0.19,   -0.27,   -0.27],
       [   1.  ,   -0.64,   -1.27, ...,   -0.19,   -0.27,   -0.27],
       ...,
       [   1.  ,    0.54,    1.32, ...,   -0.19,   -0.27,    3.76],
       [   1.  ,    0.55,    1.32, ...,   -0.19,   -0.27,    3.76],
       [   1.  , -350.47,   -5.15, ...,   -0.19,   -0.27,   -0.27]])
```

We will create the ordinary least squares model for the blinded X dataset and apply the predict function in the model class. Lastly, we will print out the model.summary2.

```python
# We assumed the name of town does not influence the price of the HDB.
model = sm.OLS(y,xBlindedFS1).fit()

# Run the predictions
predictions1 = model.predict(xBlindedFS1)

# Add the variables name
#(https://stackoverflow.com/questions/36561897/naming-explanatory-variables-in-regression-output)
print(model.summary2(xname=['Constant','Year','Room Number','Storey','floor area',
                            'Remaining Lease','Distance To MRT/LRT',
                            'Distance To CBD'],yname = 'resale_price'))
```

```
                         Results: Ordinary least squares
==================================================================================
Model:                  OLS                   Adj. R-squared:          0.701
Dependent Variable:     resale_price          AIC:                     3558084.8083
Date:                   2022-01-03 09:01      BIC:                     3558163.6057
No. Observations:       140039                Log-Likelihood:          -1.7790e+06
Df Model:               7                     F-statistic:             4.681e+04
Df Residuals:           140031                Prob (F-statistic):      0.00
R-squared:              0.701                 Scale:                   6.3383e+09
----------------------------------------------------------------------------------
                       Coef.     Std.Err.      t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
Constant            437124.7705 212.7466 2054.6738 0.0000 436707.7913 437541.7497
Year                  4891.2405 212.9230   22.9719 0.0000   4473.9156   5308.5654
Room Number          25155.0368 626.1838   40.1720 0.0000  23927.7285  26382.3452
Storey               34605.9726 230.3799  150.2126 0.0000  34154.4323  35057.5129
floor area           56849.3890 613.0259   92.7357 0.0000  55647.8700  58050.9080
Remaining Lease      48863.6269 260.1481  187.8300 0.0000  48353.7415  49373.5122
Distance To MRT/LRT  -5981.6873 220.8094  -27.0898 0.0000  -6414.4696  -5548.9050
Distance To CBD     -71535.7399 238.6506 -299.7509 0.0000 -72003.4906 -71067.9892
----------------------------------------------------------------------------------
Omnibus:                  24535.856          Durbin-Watson:               0.509
Prob(Omnibus):            0.000              Jarque-Bera (JB):            92521.223
Skew:                     0.849              Prob(JB):                    0.000
Kurtosis:                 6.602              Condition No.:               6
==================================================================================
```

Observation

- The Adjusted R squared corresponds to the percentage of explained variance of the predictions. The model can explain (McAleer, 2020) 70.1 per cent of the data.
- The p-value in the model is less than 0.05. We can conclude that our variables have a causal relationship with the HDB resale price. There is a decrease of $71579.07 in the price for every meter away from the CBD area.

Regression Error Exploration

We will now explore the 3 key errors for regression

- Mean Absolute Error (MAE)
- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)

We will use the metrics function from the sklearn library to generate the MAE, MSE and RMSE

```
# Regression error for xBlinded dataset
from sklearn import metrics

MAE1 = metrics.mean_absolute_error(y,predictions1)
MSE1 = metrics.mean_squared_error(y,predictions1)
RMSE1 = np.sqrt(metrics.mean_squared_error(y,predictions1))

print('Mean Absolute Error:', MAE1)
print('Mean Squared Error:', MSE1)
print('Root Mean Squared Error:', RMSE1)
```

```
Mean Absolute Error: 60968.28232618647
Mean Squared Error: 6337956763.354466
Root Mean Squared Error: 79611.28540197342
```

We will find the HDB Resale price of the Y NumPy array.

```
# find the mean of Resale Price
mean = np.mean(y)
mean
```

```
437124.7704598719
```

We will find the error rate by finding the ratio of RMSE to the mean of HDB resale price

```
# Error rate

ErrorRate1 = (RMSE1/mean)*100
print(ErrorRate1)
```

```
18.21248549200708
```

By inspecting our RMSE value, our model prediction will miss the actual value of the property by $79565.55 on average. The error rate in the percentage calculated above is 18.2 % which is high.

Unblinded

Now, we will create the model for the unblinded X modified dataset.

```
# We assumed the name of town does not influence the price of the HDB.
model = sm.OLS(y,xUnblindedFS1).fit()

# Run the predictions
predictions2 = model.predict(xUnblindedFS1)

# Add the variables name
print(model.summary2(xname = ['Constant',
                             'Year',
                             'Room Number',
                             'Storey',
                             'floor area',
                             'Remaining Lease',
                             'Distance To MRT/LRT',
                             'Distance To CBD',
                             'Ang Mo Kio',
                             'Bedok',
                             'Bishan',
                             'Bukit Batok',
                             'Bukit Merah',
                             'Bukit Panjang',
                             'Bukit Timah',
                             'Central Area',
                             'Choa Chu Kang',
                             'Clementi',
                             'Geylang',
                             'Hougang',
                             'Jurong East',
                             'Jurong West',
                             'Kallang/Whampoa',
                             'Marine Parade',
                             'Pasir Ris',
                             'Punggol',
                             'QueenTown',
                             'Sembawang',
                             'Sengkang',
                             'Serangoon',
                             'Tampines',
                             'Toa Payoh',
                             'Woodlands',
                             'Yishun'],
                             yname = 'resale_price'))
```

We will print out the model summary for the unblinded dataset.

```
                          Results: Ordinary least squares
=========================================================================================
Model:                    OLS                    Adj. R-squared:           0.828
Dependent Variable:       resale_price           AIC:                      3480746.8431
Date:                     2022-01-03 09:01       BIC:                      3481081.7321
No. Observations:         140039                 Log-Likelihood:           -1.7403e+06
Df Model:                 33                     F-statistic:              2.038e+04
Df Residuals:             140005                 Prob (F-statistic):       0.00
R-squared:                0.828                  Scale:                    3.6480e+09
-----------------------------------------------------------------------------------------
                          Coef.        Std.Err.      t      P>|t|      [0.025        0.975]
-----------------------------------------------------------------------------------------
Constant               437124.7705    161.3989 2708.3511 0.0000   436808.4318   437441.1091
Year                    42341.3209    463.6602   91.3197 0.0000    41432.5558    43250.0861
Room Number             18213.0977    492.2534   36.9994 0.0000    17248.2904    19177.9050
Storey                  27332.7035    179.9933  151.8540 0.0000    26979.9199    27685.4870
floor area              68957.3253    491.3246  140.3498 0.0000    67994.3386    69920.3121
Remaining Lease         66142.9686    220.6629  299.7467 0.0000    65710.4736    66575.4637
Distance To MRT/LRT    -10402.4802    213.5172  -48.7196 0.0000   -10820.9699    -9983.9905
Distance To CBD        -59251.7052   1059.9459  -55.9007 0.0000   -61329.1788   -57174.2315
Ang Mo Kio           -3195686.2721 36907.7682  -86.5857 0.0000 -3268024.7938 -3123347.7504
Bedok                -3560407.4618 40914.8403  -87.0200 0.0000 -3640599.7685 -3480215.1550
Bishan               -2035192.0988 23609.1319  -86.2036 0.0000 -2081465.5471 -1988918.6505
Bukit Batok          -2757983.6294 31607.3282  -87.2577 0.0000 -2819933.3900 -2696033.8689
Bukit Merah          -3007163.7338 34675.1546  -86.7239 0.0000 -3075126.3755 -2939201.0922
Bukit Panjang        -2885903.2649 33008.1145  -87.4301 0.0000 -2950598.5399 -2821207.9899
Bukit Timah           -706017.9114  8213.3483  -85.9598 0.0000  -722115.9174  -689919.9053
Central Area         -1474167.5973 17064.9632  -86.3856 0.0000 -1507614.5997 -1440720.5950
Choa Chu Kang        -3101382.7976 35433.0891  -87.5279 0.0000 -3170830.9765 -3031934.6186
Clementi             -2328791.7859 26830.9232  -86.7951 0.0000 -2381379.8838 -2276203.6881
Geylang              -2414897.3086 27802.5458  -86.8589 0.0000 -2469389.7681 -2360404.8491
Hougang              -3211031.9076 36850.6506  -87.1364 0.0000 -3283258.4801 -3138805.3351
Jurong East          -2183519.9064 25026.0889  -87.2497 0.0000 -2232570.5634 -2134469.2493
Jurong West          -3816877.2544 43611.9225  -87.5191 0.0000 -3902355.7908 -3731398.7180
Kallang/Whampoa      -2609556.5968 30080.0876  -86.7536 0.0000 -2668512.9949 -2550600.1988
Marine Parade        -1219680.4302 14154.6345  -86.1683 0.0000 -1247423.2439 -1191937.6165
Pasir Ris            -2367602.0194 27149.5198  -87.2060 0.0000 -2420814.5604 -2314389.4785
Punggol              -3832200.8029 43933.1569  -87.2280 0.0000 -3918308.9527 -3746092.6532
QueenTown            -2568537.7440 29668.4873  -86.5746 0.0000 -2626687.4132 -2510388.0747
Sembawang            -2216439.4500 25464.1162  -87.0417 0.0000 -2266348.6321 -2166530.2679
Sengkang             -4145409.6969 47414.3708  -87.4294 0.0000 -4238340.9593 -4052478.4344
Serangoon            -2025573.8822 23355.9742  -86.7262 0.0000 -2071351.1462 -1979796.6183
Tampines             -3763053.8770 43237.7060  -87.0318 0.0000 -3847798.9562 -3678308.7978
Toa Payoh            -2706999.2962 31226.0082  -86.6905 0.0000 -2768201.6769 -2645796.9156
Woodlands            -3760592.2000 43188.5238  -87.0739 0.0000 -3845240.8831 -3675943.5170
Yishun               -3737874.2618 43039.1418  -86.8483 0.0000 -3822230.1589 -3653518.3648
-----------------------------------------------------------------------------------------
Omnibus:                  17183.069            Durbin-Watson:               0.830
Prob(Omnibus):            0.000                Jarque-Bera (JB):            40099.822
Skew:                     0.730                Prob(JB):                    0.000
Kurtosis:                 5.177                Condition No.:               1761
=========================================================================================
```

Observation

- Our R-Squared value has improved to explain 82.7% of the variance in the dataset.
- Each town has a unique decrease in the price of the HDB resale price.
- Bukit Timah, Marine Parade, Bishan are highly desirable estates with the lower reduction in resale price. The cities that are located further away from CBD such as Woodlands, Jurong and Yishun have the greatest reduction in resale price.

Regression Error Revisited

We will generate our MAE, MSE and MSE for the unblinded model as well.

```
# Regression error for X modified dataset
from sklearn import metrics

MAE2 = metrics.mean_absolute_error(y,predictions2)
MSE2 = metrics.mean_squared_error(y,predictions2)
RMSE2 = np.sqrt(metrics.mean_squared_error(y,predictions2))

print('Mean Absolute Error:', MAE2)
print('Mean Squared Error:', MSE2)
print('Root Mean Squared Error:', RMSE2)
```

```
Mean Absolute Error: 46334.98214794251
Mean Squared Error: 3647072974.417175
Root Mean Squared Error: 60391.00077343622
```

We will now calculate the error rate for the unblinded model.

```
# Error rate

mean = np.mean(y)

ErrorRate2 = (RMSE2/mean)*100
print(ErrorRate2)
```

```
13.815506430784644
```

The MAE, MSE and RMSE values have decreased compared to the blinded model. The error rate has decreased to 13.8% from 18.2%.

Conclusion

There is a significant relationship between the distance to the CBD and the price per sqm. However, our model is not able to fully explain the HDB housing prices in Singapore. The needs and priorities of buyers are unique when purchasing HDB flats. Hence, the addition of factors such as amenities like hospitals, education institutions and the maturity of the estate could potentially increase the predictive powers in the HDB price. More research is needed to establish the common driving forces – intangible or tangible that push the buyers to purchase the HDB resale flats.

References

Baker, H. (1995). *Computing distances*. Computing Distances. Retrieved December 17, 2021, from https://cs.nyu.edu/visual/home/proj/tiger/gisfaq.html

Hayes, A. (2021, September 7). *Multiple linear regression (MLR) definition*. Investopedia. Retrieved December 28, 2021, from https://www.investopedia.com/terms/m/mlr.asp

An HDB flat for your different life cycle needs. HDB. (2020). Retrieved December 22, 2021, from https://www.hdb.gov.sg/about-us/news-and-publications/publications/hdbspeaks/an-hdb-flat-for-your-different-life-cycle-needs

Housing and Development Board. (2021, July 28). *Resale flat prices*. Data.gov.sg. Retrieved December 17, 2021, from https://data.gov.sg/dataset/resale-flat-prices

McAleer, T. (2020, December 14). *Interpreting linear regression through statsmodels .summary()*. Medium. Retrieved December 22, 2021, from https://medium.com/swlh/interpreting-linear-regression-through-statsmodels-summary-4796d359035a

Müller Andreas Christian, & Guido, S. (2017). *Introduction to machine learning with python a guide for Data scientists*. O'Reilly.

Perktold, J., Seabold, S., & Taylor, J. (2009). *Ordinary least squares*¶. statsmodels. Retrieved December 20, 2021, from https://www.statsmodels.org/dev/examples/notebooks/generated/ols.html#

Roy, B. (2020, April 7). *All about feature scaling*. Medium. Retrieved December 28, 2021, from https://towardsdatascience.com/all-about-feature-scaling-bcc0ad75cb35#:~:text=Feature%20scaling%20is%20essential%20for,that%20calculate%20distances%20between%20data.&text=Therefore%2C%20the%20range%20of%20all,proportionately%20to%20the%20final%20distance.

Yuan, Y. (2020, April 3). *Working with apis in data science-explore bit-rent theory in Singapore's HDB resale market*. Medium. Retrieved December 17, 2021, from https://towardsdatascience.com/working-with-apis-in-data-science-explore-bit-rent-theory-in-singapores-hdb-resale-market-d7760fdfc601