In [1]:
```
from __future__ import annotations
!pip install --user --upgrade pip
```

```
Requirement already satisfied: pip in c:\users\infin\appdata\roaming\python\python38
\site-packages (24.2)
```

Upgrades pip, optional. Requires --user permissions and can be bypassed and done manually.

In [3]:
```
#!pip install python==3.11.9
#!pip install tensorflow==2.3
#!pip install keras==2.4.3
#!pip install numpy==1.23.1
#!pip install scikit-learn
```

Installs and runs Tensorflow and Keras, main dependencies of ANN models.

In [5]:
```
import pandas as pd

flatdb = pd.read_csv("datasets\completed_flat_price_2017_2023.csv")
cpidb = pd.read_csv("datasets\CPI_2000to2022Feb.csv")
```

Imports pandas and reads CSV files in local /datasets folder.

In [7]:
```
flatdb2 = flatdb.drop(columns=['block','street_name','storey_range','full_address',
print(flatdb2)
```

```
          month        town  flat_type  floor_area_sqm      flat_model  \
0       2017-01  ANG MO KIO     2 ROOM            44.0        Improved
1       2017-01  ANG MO KIO     3 ROOM            67.0  New Generation
2       2017-01  ANG MO KIO     3 ROOM            67.0  New Generation
3       2017-01  ANG MO KIO     3 ROOM            68.0  New Generation
4       2017-01  ANG MO KIO     3 ROOM            67.0  New Generation
...         ...         ...        ...             ...             ...
146867  2023-02     YISHUN     5 ROOM           127.0        Improved
146868  2023-02     YISHUN     5 ROOM           122.0        Improved
146869  2023-02     YISHUN  EXECUTIVE           181.0       Apartment
146870  2023-02     YISHUN  EXECUTIVE           146.0      Maisonette
146871  2023-02     YISHUN  EXECUTIVE           142.0       Apartment

        lease_commence_date  resale_price
0                      1979      232000.0
1                      1978      250000.0
2                      1980      262000.0
3                      1980      265000.0
4                      1980      265000.0
...                     ...           ...
146867                 1988      700000.0
146868                 1988      700000.0
146869                 1992     1068000.0
146870                 1988      838000.0
146871                 1987      765000.0

[146872 rows x 7 columns]
```

Isolation of dataset contents into classifiable features, stored in flatdb2 to preserve original imported dataset as contingency.

In [9]:
```
g = cpidb['Tranc_Mth'].unique()

cpidb2 = cpidb.replace(g, ['01', '02', '03', '04', '05', '06', '07', '08', '09', '1
print(cpidb2)
```

```
    Tranc_Yr Tranc_Mth  CPI Housing
0        2000        01       75.962
1        2000        02       75.961
2        2000        03       75.961
3        2000        04       75.962
4        2000        05       75.962
..        ...       ...          ...
273      2022        10      106.702
274      2022        11      108.770
275      2022        12      109.040
276      2023        01      109.290
277      2023        02      109.350

[278 rows x 3 columns]
```

In [10]:
```
data = []
for i in range(len(cpidb2)):
    data.append((str(cpidb2['Tranc_Yr'][i]) + "-" + str(cpidb2['Tranc_Mth'][i])))

data2 = {
```

```
    "month" : data,
    "CPI Housing" : cpidb2['CPI Housing']
}

cpidb3 = pd.DataFrame(data2)
print(cpidb3)
```

```
      month  CPI Housing
0    2000-01       75.962
1    2000-02       75.961
2    2000-03       75.961
3    2000-04       75.962
4    2000-05       75.962
..       ...          ...
273  2022-10      106.702
274  2022-11      108.770
275  2022-12      109.040
276  2023-01      109.290
277  2023-02      109.350

[278 rows x 2 columns]
```

In [11]:
```
data = []
for i in range(len(flatdb2)):
    a = int(flatdb2['month'][i][:4])
    if a >= 2000 and a < 2021:
        data.append(flatdb2.iloc[i])

trainset = pd.DataFrame(data)

data = []
for i in range(len(flatdb2)):
    a = int(flatdb2['month'][i][:4])
    if a >= 2021 and a <= 2023:
        data.append(flatdb2.iloc[i])

testset = pd.DataFrame(data)
```

In [12]:
```
train2set = trainset

num = []
for i in range(len(trainset['town'].unique())):
    num.append(i)

arg = dict(zip(trainset['town'].unique(),num))

train2set['town'] = train2set['town'].map(arg)

num = []
for i in range(len(trainset['flat_type'].unique())):
    num.append(i)

arg = dict(zip(trainset['flat_type'].unique(),num))

train2set['flat_type'] = train2set['flat_type'].map(arg)
```

```
num = []
for i in range(len(trainset['flat_model'].unique())):
    num.append(i)

arg = dict(zip(trainset['flat_model'].unique(),num))

train2set['flat_model'] = train2set['flat_model'].map(arg)

print(train2set)
```

```
         month  town  flat_type  floor_area_sqm  flat_model  \
0      2017-01     0          0            44.0           0
1      2017-01     0          1            67.0           1
2      2017-01     0          1            67.0           1
3      2017-01     0          1            68.0           1
4      2017-01     0          1            67.0           1
...        ...   ...        ...             ...         ...
87584  2020-12    25          4           146.0          10
87585  2020-12    25          4           145.0           4
87586  2020-12    25          4           142.0           4
87587  2020-12    25          4           146.0          10
87588  2020-12    25          4           142.0           4

       lease_commence_date  resale_price
0                     1979      232000.0
1                     1978      250000.0
2                     1980      262000.0
3                     1980      265000.0
4                     1980      265000.0
...                    ...           ...
87584                 1988      560000.0
87585                 1988      540000.0
87586                 1987      638000.0
87587                 1988      683500.0
87588                 1987      670000.0

[87589 rows x 7 columns]
```

In [13]:
```
test2set = testset

num = []
for i in range(len(testset['town'].unique())):
    num.append(i)

arg = dict(zip(testset['town'].unique(),num))

test2set['town'] = test2set['town'].map(arg)

num = []
for i in range(len(testset['flat_type'].unique())):
    num.append(i)

arg = dict(zip(testset['flat_type'].unique(),num))

test2set['flat_type'] = test2set['flat_type'].map(arg)
```

```
num = []
for i in range(len(testset['flat_model'].unique())):
    num.append(i)

arg = dict(zip(testset['flat_model'].unique(),num))

test2set['flat_model'] = test2set['flat_model'].map(arg)

print(test2set)
```

```
         month  town  flat_type  floor_area_sqm  flat_model  \
87589   2021-01     0          0            45.0           0
87590   2021-01     0          0            45.0           0
87591   2021-01     0          1            68.0           1
87592   2021-01     0          1            68.0           1
87593   2021-01     0          1            68.0           1
...         ...   ...        ...             ...         ...
146867  2023-02    25          3           127.0           0
146868  2023-02    25          3           122.0           0
146869  2023-02    25          4           181.0           6
146870  2023-02    25          4           146.0           7
146871  2023-02    25          4           142.0           6

        lease_commence_date  resale_price
87589                  1986      211000.0
87590                  1986      225000.0
87591                  1981      260000.0
87592                  1980      265000.0
87593                  1980      265000.0
...                     ...           ...
146867                 1988      700000.0
146868                 1988      700000.0
146869                 1992     1068000.0
146870                 1988      838000.0
146871                 1987      765000.0

[59283 rows x 7 columns]
```

In [14]:
```
train3set = train2set.merge(cpidb3, on='month')
print(train3set)
test3set = test2set.merge(cpidb3, on='month')
```

```
         month  town  flat_type  floor_area_sqm  flat_model  \
0      2017-01     0          0            44.0           0
1      2017-01     0          1            67.0           1
2      2017-01     0          1            67.0           1
3      2017-01     0          1            68.0           1
4      2017-01     0          1            67.0           1
...        ...   ...        ...             ...         ...
87584  2020-12    25          4           146.0          10
87585  2020-12    25          4           145.0           4
87586  2020-12    25          4           142.0           4
87587  2020-12    25          4           146.0          10
87588  2020-12    25          4           142.0           4

       lease_commence_date  resale_price  CPI Housing
0                     1979      232000.0      106.614
1                     1978      250000.0      106.614
2                     1980      262000.0      106.614
3                     1980      265000.0      106.614
4                     1980      265000.0      106.614
...                    ...           ...          ...
87584                 1988      560000.0      101.119
87585                 1988      540000.0      101.119
87586                 1987      638000.0      101.119
87587                 1988      683500.0      101.119
87588                 1987      670000.0      101.119

[87589 rows x 8 columns]
```

In [15]:
```
print(train3set.columns)
```

```
Index(['month', 'town', 'flat_type', 'floor_area_sqm', 'flat_model',
       'lease_commence_date', 'resale_price', 'CPI Housing'],
      dtype='object')
```

In [16]:
```
train_targets = train3set.pop('resale_price') / 100000
test_targets = test3set.pop('resale_price') / 100000

train_data = train3set.copy().drop(columns=['month']).fillna(0).astype('float64')
test_data = test3set.copy().drop(columns=['month']).fillna(0).astype('float64')

print(train_targets, test_targets)
print(train_data.dtypes)
```

```
0        2.320
1        2.500
2        2.620
3        2.650
4        2.650
         ...
87584    5.600
87585    5.400
87586    6.380
87587    6.835
87588    6.700
Name: resale_price, Length: 87589, dtype: float64 0         2.11
1         2.25
2         2.60
3         2.65
4         2.65
         ...
59278     7.00
59279     7.00
59280    10.68
59281     8.38
59282     7.65
Name: resale_price, Length: 59283, dtype: float64
town                   float64
flat_type              float64
floor_area_sqm         float64
flat_model             float64
lease_commence_date    float64
CPI Housing            float64
dtype: object
```

In [17]:
```
import numpy as np
np.object = object
np.bool = np.bool_
```

In [18]:
```
from sklearn import preprocessing
import tensorflow.keras.utils as utils
import tensorflow.data as data

scaler = preprocessing.MinMaxScaler()

train_data = scaler.fit_transform(train_data)
train_data = np.where(np.isfinite(train_data), train_data, 0)

test_data =  scaler.fit_transform(test_data)
test_data = np.where(np.isfinite(test_data), test_data, 0)

print(train_data.shape)
print(train_data)

#print(test_data.dtypes)
```

```
(87589, 6)
[[0.         0.         0.05963303 0.         0.24528302 1.        ]
 [0.         0.16666667 0.16513761 0.05263158 0.22641509 1.        ]
 [0.         0.16666667 0.16513761 0.05263158 0.26415094 1.        ]
 ...
 [1.         0.66666667 0.50917431 0.21052632 0.39622642 0.37641852]
 [1.         0.66666667 0.52752294 0.52631579 0.41509434 0.37641852]
 [1.         0.66666667 0.50917431 0.21052632 0.39622642 0.37641852]]
```

In [19]:
```
import tensorflow
import tensorflow.keras as keras
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers

def rss(y_true, y_pred):
    RSS = []
    for y_true, y_pred in zip(y_true, y_pred):
        RSS.append((y_true-y_pred)**2)

    return sum(RSS)

import numpy as np

k = 4
num_val_samples = len(train_data) // k
for i in range(k):
    partial_train_data = np.concatenate(
        [train_data[:i * num_val_samples],
         train_data[(i + 1) * num_val_samples:]],
        axis=0)
    partial_train_targets = np.concatenate(
        [train_targets[:i * num_val_samples],
         train_targets[(i + 1) * num_val_samples:]],
        axis=0)

def build_model():
    model = tensorflow.keras.Sequential()
    model.add(layers.Dense(train_data.shape[1], activation='relu', input_shape=(tra
    model.add(layers.Dense(1, activation='sigmoid'))
    opt = keras.optimizers.RMSprop(lr=0.05, momentum=0.1, clipnorm=1, clipvalue=1)
    model.compile(optimizer=opt, loss=rss, metrics=['mean_absolute_error','mean_abs
    return model
```

In [22]:
```
model = build_model()
model.summary()
model.get_weights()
model.fit(partial_train_data, partial_train_targets, epochs=15, batch_size=128)
results = model.evaluate(test_data, test_targets, batch_size=128)
print(results)
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 6)                 42
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 7
=================================================================
Total params: 49
Trainable params: 49
Non-trainable params: 0
_________________________________________________________________
Epoch 1/15
514/514 [==============================] - 31s 61ms/step - loss: 1790.3440 - mean_ab
solute_error: 3.4097 - mean_absolute_percentage_error: 75.1373
Epoch 2/15
514/514 [==============================] - 32s 63ms/step - loss: 1772.9562 - mean_ab
solute_error: 3.3903 - mean_absolute_percentage_error: 74.6465
Epoch 3/15
514/514 [==============================] - 32s 63ms/step - loss: 1772.8177 - mean_ab
solute_error: 3.3903 - mean_absolute_percentage_error: 74.6454
Epoch 4/15
514/514 [==============================] - 31s 61ms/step - loss: 1772.7008 - mean_ab
solute_error: 3.3900 - mean_absolute_percentage_error: 74.6381
Epoch 5/15
514/514 [==============================] - 32s 63ms/step - loss: 1772.5710 - mean_ab
solute_error: 3.3900 - mean_absolute_percentage_error: 74.6382
Epoch 6/15
514/514 [==============================] - 33s 63ms/step - loss: 1772.8347 - mean_ab
solute_error: 3.3900 - mean_absolute_percentage_error: 74.6381
Epoch 7/15
514/514 [==============================] - 33s 63ms/step - loss: 1772.8311 - mean_ab
solute_error: 3.3900 - mean_absolute_percentage_error: 74.6382
Epoch 8/15
514/514 [==============================] - 32s 62ms/step - loss: 1772.7292 - mean_ab
solute_error: 3.3900 - mean_absolute_percentage_error: 74.6382
Epoch 9/15
514/514 [==============================] - 32s 62ms/step - loss: 1772.7793 - mean_ab
solute_error: 3.3900 - mean_absolute_percentage_error: 74.6381
Epoch 10/15
514/514 [==============================] - 32s 62ms/step - loss: 1772.6891 - mean_ab
solute_error: 3.3900 - mean_absolute_percentage_error: 74.6382
Epoch 11/15
514/514 [==============================] - 32s 63ms/step - loss: 1772.7670 - mean_ab
solute_error: 3.3900 - mean_absolute_percentage_error: 74.6381
Epoch 12/15
514/514 [==============================] - 32s 63ms/step - loss: 1772.6914 - mean_ab
solute_error: 3.3900 - mean_absolute_percentage_error: 74.6382
Epoch 13/15
514/514 [==============================] - 32s 62ms/step - loss: 1772.7126 - mean_ab
solute_error: 3.3900 - mean_absolute_percentage_error: 74.6381
Epoch 14/15
514/514 [==============================] - 32s 62ms/step - loss: 1772.6642 - mean_ab
solute_error: 3.3900 - mean_absolute_percentage_error: 74.6382
Epoch 15/15
514/514 [==============================] - 32s 62ms/step - loss: 1772.6749 - mean_ab
```

```
solute_error: 3.3900 - mean_absolute_percentage_error: 74.6382
464/464 [==============================] - 14s 31ms/step - loss: 2739.0549 - mean_ab
solute_error: 4.3125 - mean_absolute_percentage_error: 79.3185
[2739.054931640625, 4.31248140335083, 79.31849670410156]
```

In [ ]: