. Load the dataset and import it into a Pandas DataFrame.

In [8]:
```
from google.colab import files
import pandas as pd
import tensorflow as tf

# Upload the CSV file
uploaded = files.upload()   # Choose "Real estate.csv"

# Load into Pandas DataFrame
df = pd.read_csv("Real estate - Real estate.csv")  # make sure filename matche
print("✅ Dataset loaded successfully!")
print(df.head())

# Convert to TensorFlow tensor
data_tensor = tf.convert_to_tensor(df.values, dtype=tf.float32)
print("\nTensorFlow tensor shape:", data_tensor.shape)
```

Browse... Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Real estate - Real estate.csv to Real estate - Real estate (1).csv
✅ Dataset loaded successfully!
   X1 transaction date  X2 house age  X3 distance to the nearest MRT station  \
0             2012.917          32.0                                84.87882
1             2012.917          19.5                               306.59470
2             2013.583          13.3                               561.98450
3             2013.500          13.3                               561.98450
4             2012.833           5.0                               390.56840

   X4 number of convenience stores  X5 latitude  X6 longitude  \
0                             10.0     24.98298     121.54024
1                              9.0     24.98034     121.53951
2                              5.0     24.98746     121.54391
3                              5.0     24.98746     121.54391
4                              5.0     24.97937     121.54245

   Y house price of unit area
0                        37.9
1                        42.2
2                        47.3
3                        54.8
4                        43.1

TensorFlow tensor shape: (415, 7)
```

Display the first five rows and the last three rows of the dataset

In [9]:
```
print("First 5 rows:")
print(df.head())
print("\nLast 3 rows:")
print(df.tail(3))
```

```
First 5 rows:
   X1 transaction date  X2 house age  X3 distance to the nearest MRT station  \
0             2012.917          32.0                                84.87882
1             2012.917          19.5                               306.59470
2             2013.583          13.3                               561.98450
3             2013.500          13.3                               561.98450
4             2012.833           5.0                               390.56840

   X4 number of convenience stores  X5 latitude  X6 longitude  \
0                             10.0     24.98298     121.54024
1                              9.0     24.98034     121.53951
2                              5.0     24.98746     121.54391
3                              5.0     24.98746     121.54391
4                              5.0     24.97937     121.54245

   Y house price of unit area
0                        37.9
1                        42.2
2                        47.3
3                        54.8
4                        43.1

Last 3 rows:
     X1 transaction date  X2 house age  \
412             2013.000           8.1
413             2013.500           6.5
414             2013.167           1.9

     X3 distance to the nearest MRT station  X4 number of convenience stores  \
412                               104.81010                              5.0
413                                90.45606                              9.0
414                               355.00000                              NaN

     X5 latitude  X6 longitude  Y house price of unit area
412     24.96674     121.54067                        52.5
413     24.97433     121.54310                        63.9
414     24.97293     121.54026                        40.5
```

Get the dimensions (number of rows and columns) of the dataset.

In [10]:
```
print("\nShape of dataset (rows, columns):", df.shape)
```

```
Shape of dataset (rows, columns): (415, 7)
```

Generate descriptive statistics (mean, median, standard deviation, five-point summary, IQR, etc.) for the data

In [11]:
```
print("\nDescriptive Statistics:")
print(df.describe(include="all"))
print("\nMedian values:")
print(df.median())

print("\nFive-point summary:")
print(df.describe().loc[["min", "25%", "50%", "75%", "max"]])

# IQR
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
print("\nInterquartile Range (IQR):")
print(IQR)
```

```
Descriptive Statistics:
       X1 transaction date  X2 house age  \
count           415.000000    415.000000   
mean           2013.149014     17.674458   
std               0.281628     11.405161   
min            2012.667000      0.000000   
25%            2012.917000      8.950000   
50%            2013.167000     16.100000   
75%            2013.417000     28.100000   
max            2013.583000     43.800000   

       X3 distance to the nearest MRT station  \
count                              415.000000   
mean                              1082.129338   
std                               1261.092057   
min                                 23.382840   
25%                                289.324800   
50%                                492.231300   
75%                               1452.760000   
max                               6488.021000   

       X4 number of convenience stores  X5 latitude  X6 longitude  \
count                       414.000000   415.000000    415.000000   
mean                          4.094203    24.969039    121.533378   
std                           2.945562     0.012397      0.015332   
min                           0.000000    24.932070    121.473530   
25%                           1.000000    24.963010    121.528570   
50%                           4.000000    24.971100    121.538630   
75%                           6.000000    24.977450    121.543300   
max                          10.000000    25.014590    121.566270   

       Y house price of unit area  
count                  415.000000  
mean                    37.986265  
std                     13.590608  
min                      7.600000  
25%                     27.700000  
50%                     38.500000  
75%                     46.600000  
max                    117.500000  

Median values:
X1 transaction date                       2013.16700
X2 house age                                16.10000
X3 distance to the nearest MRT station     492.23130
X4 number of convenience stores              4.00000
X5 latitude                                 24.97110
X6 longitude                               121.53863
Y house price of unit area                  38.50000
dtype: float64

Five-point summary:
     X1 transaction date  X2 house age  \
min             2012.667          0.00   
```

```
25%               2012.917          8.95
50%               2013.167         16.10
75%               2013.417         28.10
max               2013.583         43.80

     X3 distance to the nearest MRT station  X4 number of convenience stores  \
min                                23.38284                              0.0
25%                               289.32480                              1.0
50%                               492.23130                              4.0
75%                              1452.76000                              6.0
max                              6488.02100                             10.0

     X5 latitude  X6 longitude  Y house price of unit area
min     24.93207     121.47353                         7.6
25%     24.96301     121.52857                        27.7
50%     24.97110     121.53863                        38.5
75%     24.97745     121.54330                        46.6
max     25.01459     121.56627                       117.5

Interquartile Range (IQR):
X1 transaction date                          0.50000
X2 house age                                19.15000
X3 distance to the nearest MRT station    1163.43520
X4 number of convenience stores              5.00000
X5 latitude                                  0.01444
X6 longitude                                 0.01473
Y house price of unit area                  18.90000
dtype: float64
```

Print a concise summary of the dataset as information on data types (schema) and missing values

In [12]:
```
print("\nInfo about dataset:")
print(df.info())

print("\nMissing values count:")
print(df.isnull().sum())
```

```
Info about dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 415 entries, 0 to 414
Data columns (total 7 columns):
 #   Column                                  Non-Null Count  Dtype  
---  ------                                  --------------  -----  
 0   X1 transaction date                     415 non-null    float64
 1   X2 house age                            415 non-null    float64
 2   X3 distance to the nearest MRT station  415 non-null    float64
 3   X4 number of convenience stores         414 non-null    float64
 4   X5 latitude                             415 non-null    float64
 5   X6 longitude                            415 non-null    float64
 6   Y house price of unit area              415 non-null    float64
dtypes: float64(7)
memory usage: 22.8 KB
None

Missing values count:
X1 transaction date                       0
X2 house age                              0
X3 distance to the nearest MRT station    0
X4 number of convenience stores           1
X5 latitude                               0
X6 longitude                              0
Y house price of unit area                0
dtype: int64
```

Add a new column named “X22” by converting the “house age” from years to days

In [13]:
```
df["X22"] = df["X2 house age"] * 365
print(df[["X2 house age", "X22"]].head())
```

```
   X2 house age      X22
0          32.0  11680.0
1          19.5   7117.5
2          13.3   4854.5
3          13.3   4854.5
4           5.0   1825.0
```

Delete the column “X22” from the dataset.

In [14]:
```
df.drop("X22", axis=1, inplace=True)
print(df.head())
```

```
   X1 transaction date  X2 house age  X3 distance to the nearest MRT station  \
0             2012.917          32.0                                84.87882
1             2012.917          19.5                               306.59470
2             2013.583          13.3                               561.98450
3             2013.500          13.3                               561.98450
4             2012.833           5.0                               390.56840

   X4 number of convenience stores  X5 latitude  X6 longitude  \
0                             10.0     24.98298     121.54024
1                              9.0     24.98034     121.53951
2                              5.0     24.98746     121.54391
3                              5.0     24.98746     121.54391
4                              5.0     24.97937     121.54245

   Y house price of unit area
0                        37.9
1                        42.2
2                        47.3
3                        54.8
4                        43.1
```

Create three new instances synthetically and add them to the dataset

In [16]:
```
import pandas as pd

# Suppose your DataFrame is df
print("Shape before adding:", df.shape)

# Create 3 synthetic instances with the same columns
new_rows = pd.DataFrame([
    [0, 15, 560.0, 2, 24.98, 121.54, 45.0],   # Example instance
    [0, 30, 1800.0, 3, 24.96, 121.52, 35.0],  # Example instance
    [0, 5, 350.0, 1, 24.97, 121.50, 55.0]     # Example instance
], columns=df.columns)   # ⍰ use same column names

# Append to dataset
df = pd.concat([df, new_rows], ignore_index=True)

print("Shape after adding:", df.shape)
print(df.tail(5))   # show last rows including new ones
```

```
Shape before adding: (415, 7)
Shape after adding: (418, 7)
     X1 transaction date  X2 house age  \
413              2013.500           6.5
414              2013.167           1.9
415                 0.000          15.0
416                 0.000          30.0
417                 0.000           5.0

     X3 distance to the nearest MRT station  X4 number of convenience stores  \
413                                90.45606                              9.0
414                               355.00000                              NaN
415                               560.00000                              2.0
416                              1800.00000                              3.0
417                               350.00000                              1.0

     X5 latitude  X6 longitude  Y house price of unit area
413     24.97433     121.54310                        63.9
414     24.97293     121.54026                        40.5
415     24.98000     121.54000                        45.0
416     24.96000     121.52000                        35.0
417     24.97000     121.50000                        55.0
```

Delete the newly inserted three instances from the dataset.

```
In [17]: df = df[:-3]
         print("After deleting new rows:", df.tail(5))
```

```
After deleting new rows:      X1 transaction date  X2 house age  \
410              2012.667           5.6
411              2013.250          18.8
412              2013.000           8.1
413              2013.500           6.5
414              2013.167           1.9

     X3 distance to the nearest MRT station  X4 number of convenience stores  \
410                                90.45606                              9.0
411                               390.96960                              7.0
412                               104.81010                              5.0
413                                90.45606                              9.0
414                               355.00000                              NaN

     X5 latitude  X6 longitude  Y house price of unit area
410     24.97433     121.54310                        50.0
411     24.97923     121.53986                        40.6
412     24.96674     121.54067                        52.5
413     24.97433     121.54310                        63.9
414     24.97293     121.54026                        40.5
```

. Update the “house price of unit area” to 110, provided it is currently greater than the amount.

```
In [20]: df.loc[df["Y house price of unit area"] > 110, "Y house price of unit area"] =
```

Find the latitude and longitude of the houses whose prices are less than or equal to 20.

In [21]:
```
cheap_houses = df[df["Y house price of unit area"] <= 20][["X5 latitude", "X6
print(cheap_houses)
```

```
     X5 latitude  X6 longitude
8       24.95095     121.48458
40      24.94155     121.50381
41      24.94297     121.50342
48      24.94684     121.49578
49      24.94925     121.49542
55      24.94968     121.53009
73      24.94155     121.50381
83      24.96056     121.50831
87      24.94297     121.50342
93      24.94920     121.53076
113     24.96172     121.53812
116     24.94375     121.47883
117     24.93885     121.50383
155     24.94155     121.50381
156     24.94883     121.52954
162     24.94297     121.50342
170     24.94741     121.49628
176     24.94867     121.49507
180     24.94898     121.49621
183     24.94155     121.50381
226     24.94155     121.50381
229     24.94890     121.53095
231     24.94235     121.50357
232     24.95032     121.49587
249     24.95743     121.47516
251     24.94960     121.53018
255     24.95095     121.48458
298     24.94155     121.50381
309     24.94883     121.52954
320     24.93885     121.50383
329     24.93885     121.50383
330     24.94935     121.53046
331     24.94826     121.49587
347     24.95719     121.47353
384     24.94297     121.50342
409     24.94155     121.50381
```

Add the missing convenience store values of instances by calculating the average number of convenience stores

In [25]:
```
# Step 1: Calculate average number of convenience stores (ignoring NaN)
avg = df["X4 number of convenience stores"].mean()

# Step 2: Fill missing values with that average
df["X4 number of convenience stores"] = df["X4 number of convenience stores"].
```

```
# Step 3: Verify
print("Missing values after filling:")
print(df["X4 number of convenience stores"].isnull().sum())
```

```
Missing values after filling:
0
```

Find the normalized distance to the nearest train station by performing: (a) Z-score normalization. (b) Min-max normalization. (c) Decimal scaling.

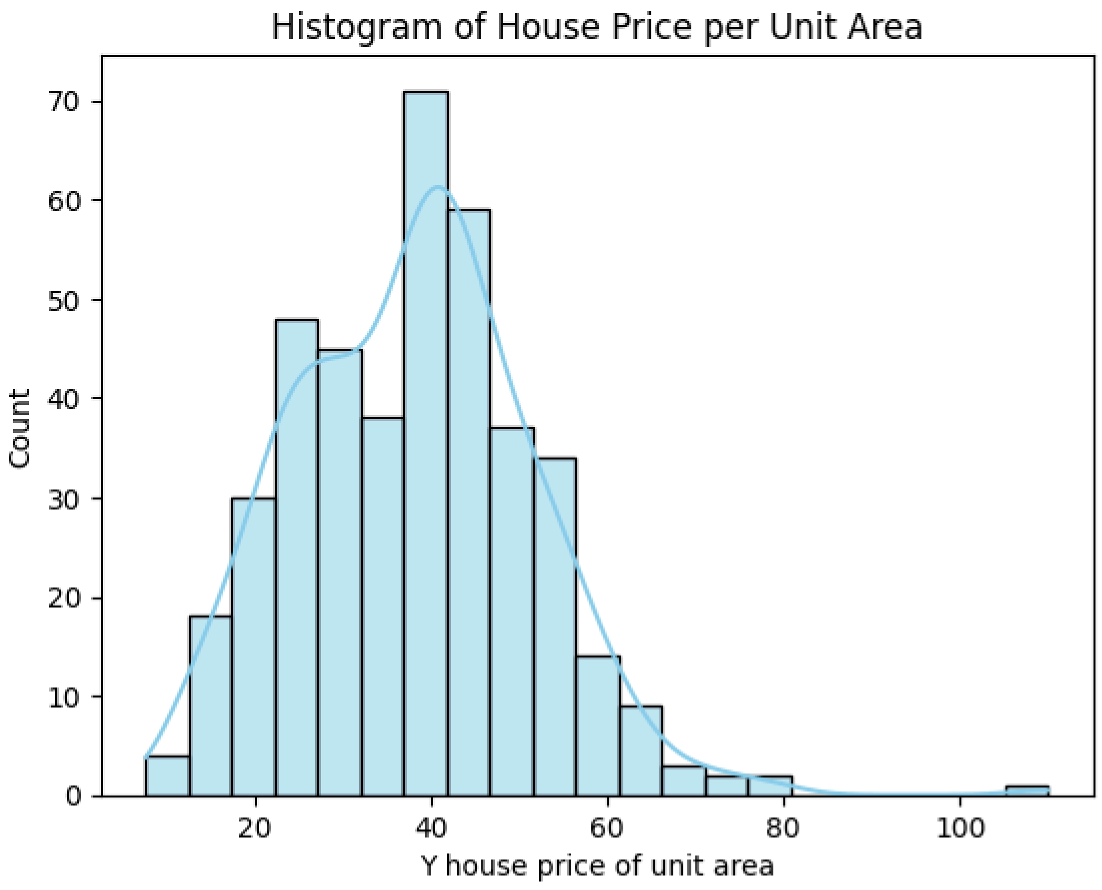
In [26]:
```
x3 = df["X3 distance to the nearest MRT station"]

# (a) Z-score
z_score = (x3 - x3.mean()) / x3.std()

# (b) Min-Max
min_max = (x3 - x3.min()) / (x3.max() - x3.min())

# (c) Decimal scaling
scaling_factor = 10**len(str(int(x3.abs().max())))
decimal_scaled = x3 / scaling_factor

print("\nZ-score normalization:\n", z_score.head())
print("\nMin-Max normalization:\n", min_max.head())
print("\nDecimal scaling:\n", decimal_scaled.head())
```

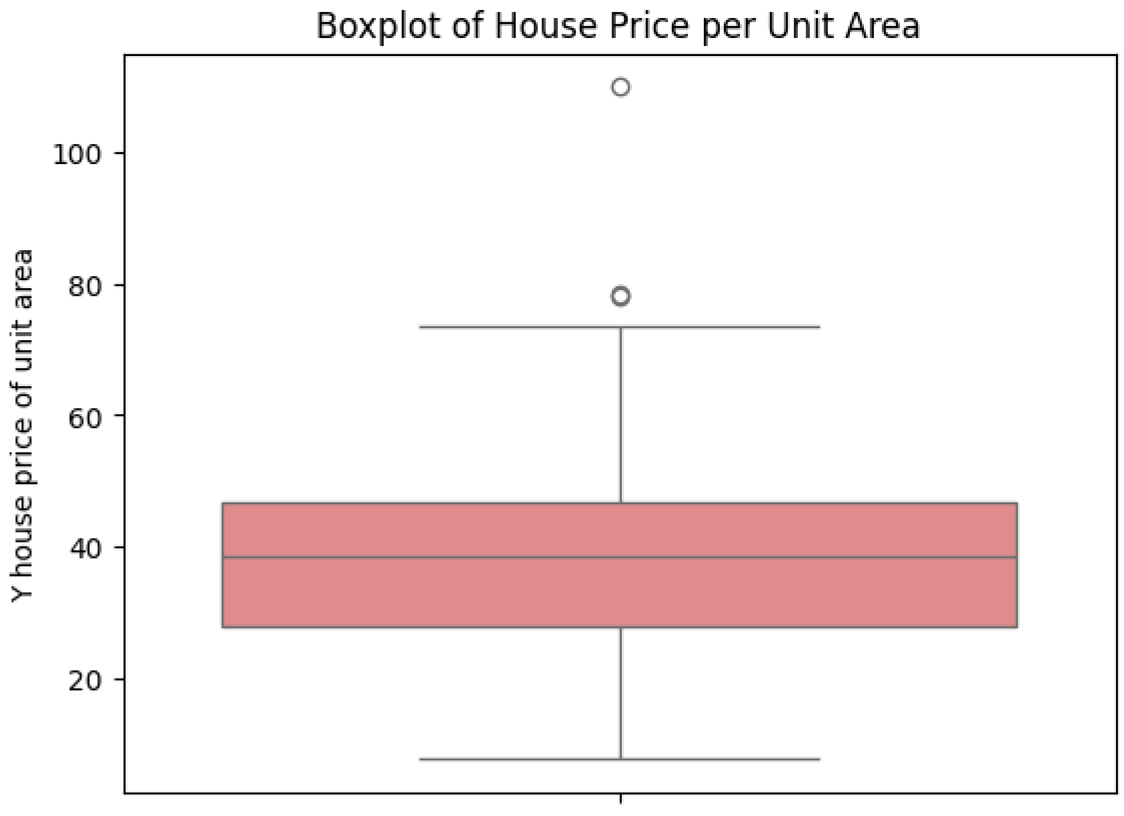
```
Z-score normalization:
 0   -0.790783
1   -0.614971
2   -0.412456
3   -0.412456
4   -0.548383
Name: X3 distance to the nearest MRT station, dtype: float64

Min-Max normalization:
 0    0.009513
1    0.043809
2    0.083315
3    0.083315
4    0.056799
Name: X3 distance to the nearest MRT station, dtype: float64

Decimal scaling:
 0    0.008488
1    0.030659
2    0.056198
3    0.056198
4    0.039057
Name: X3 distance to the nearest MRT station, dtype: float64
```

Generate the following basic visualizations using Seaborn. Customize your visualizations by adding titles, labels, legends, and appropriate color schemes. (a) Create a histogram for the ‘’Y house price of unit area” attribute.

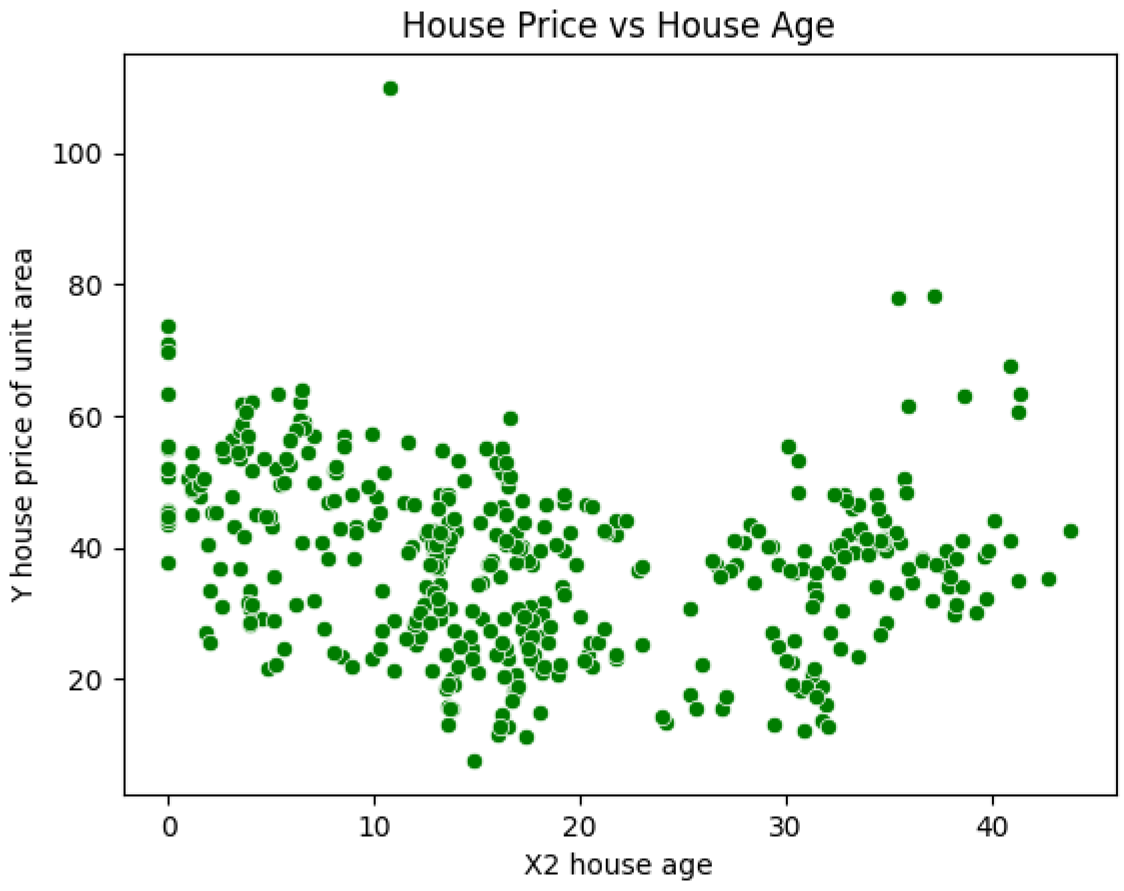
(b) Create a box-and-whisker plot for the ‘’Y house price of unit area” attribute. (c) Create a scatter plot showing house prices against house age. (d) Add a second scatter plot showing house prices against distance to the nearest MRT station.

In [27]:
```python
sns.histplot(df["Y house price of unit area"], kde=True, color="skyblue")
plt.title("Histogram of House Price per Unit Area")
plt.show()

sns.boxplot(y=df["Y house price of unit area"], color="lightcoral")
plt.title("Boxplot of House Price per Unit Area")
plt.show()

sns.scatterplot(x=df["X2 house age"], y=df["Y house price of unit area"], colo
plt.title("House Price vs House Age")
plt.show()

sns.scatterplot(x=df["X3 distance to the nearest MRT station"], y=df["Y house
plt.title("House Price vs Distance to MRT station")
plt.show()
```

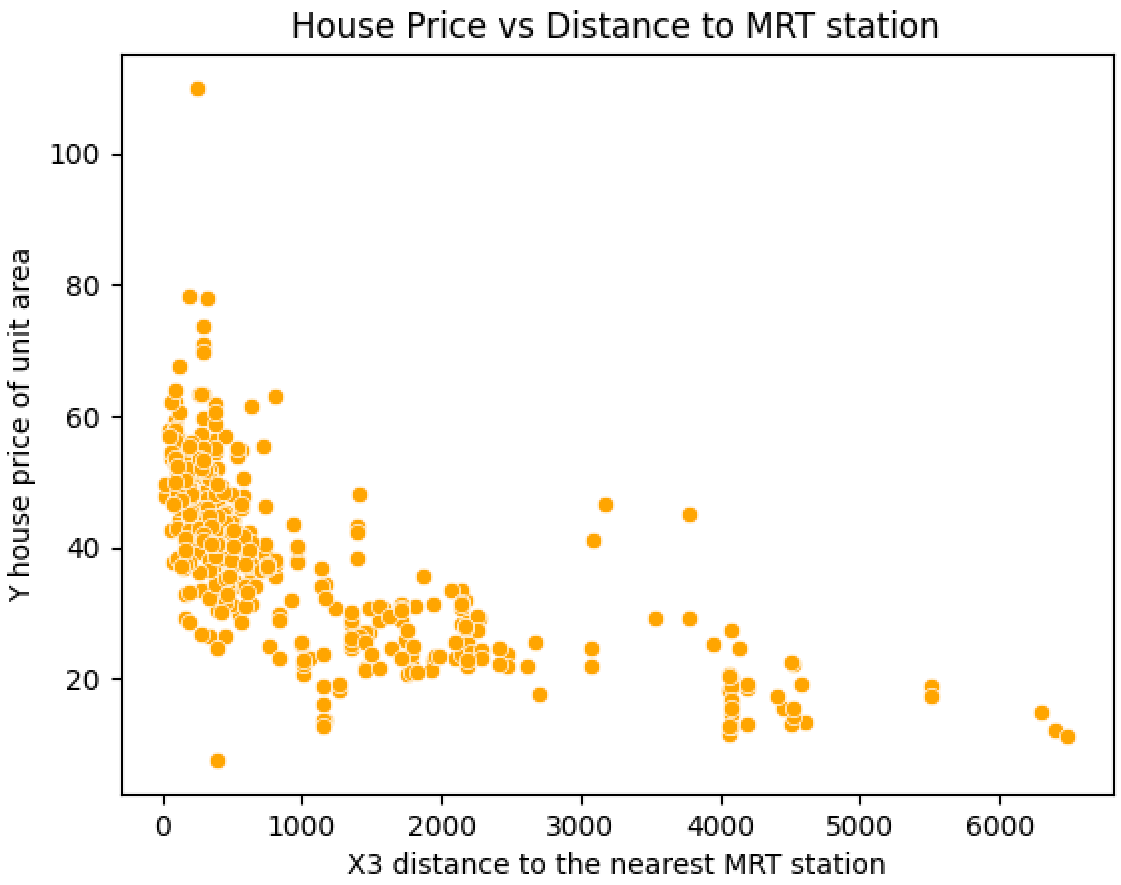




Boxplot of House Price per Unit Area
Y house price of unit area
20
40
60
80
100



House Price vs House Age
Y house price of unit area
X2 house age
0
10
20
30
40
20
40
60
80
100



Form the Design Matrix X of shape m × n + 1 in order to apply normal equation method where m is the number of training examples and n is the number of input features. Only use the two normalized input features ’X2 house age’ and ’X3 distance to the nearest MRT station’ from the dataset as second and third columns respectively and all 1 s as the first column. Also, form output vector Y of shape m × 1

In [28]:
```
# Using normalized features (Min-Max)
x2 = (df["X2 house age"] - df["X2 house age"].min()) / (df["X2 house age"].max
x3 = (df["X3 distance to the nearest MRT station"] - df["X3 distance to the ne

m = len(df)
X = np.c_[np.ones(m), x2, x3]  # m × (n+1)
Y = df["Y house price of unit area"].values.reshape(-1, 1)  # m × 1

print("X shape:", X.shape)
print("Y shape:", Y.shape)
```

```
X shape: (415, 3)
Y shape: (415, 1)
```

Find the parameter vector W using the normal equation method as W = (XT X) −1XT Y .

In [29]:
```
W = np.linalg.inv(X.T @ X) @ X.T @ Y
print("Parameters (W) from Normal Equation:\n", W)
```

```
Parameters (W) from Normal Equation:
 [[ 49.61767979]
 [ -9.99718231]
 [-46.49889746]]
```

Implement the gradient descent algorithm with the following steps. • Form the Design Matrix X of shape n × m. Only use the two normalized input features ’X2 house age’ and ’X3 distance to the nearest MRT station’ and the output vector Y of shape 1 × m • Initialize the parameter vector W of shape 1 × n and bias b (scalar). • Repeat the following steps to a certain number of iterations with learning rate α = 0.01, and print the final parameter values. (a) Calculate the prediction Yˆ = W X + b. (b) Compute loss L = 1 2 × (Yˆ − Y) 2 (c) Compute error E = Yˆ − Y (d) Compute the gradient with respect to W as dW = 1 mE ·XT and with respect to b as db = 1 m × E (sum over the columns) (e) Update W = W − αdW and b = b − αdb • Use tensorflow GradientTape() to automatically calculate the gradients in the above step (d) and redo the training steps and print the final parameter values.

In [30]:
```
alpha = 0.01
iterations = 1000

# Initialize
W = np.zeros((1, 2))  # weights for x2, x3
b = 0.0

X_gd = np.vstack([x2, x3])   # shape (2, m)
Y_gd = Y.reshape(1, -1)      # shape (1, m)
m = Y_gd.shape[1]

for i in range(iterations):
    Y_hat = np.dot(W, X_gd) + b
    E = Y_hat - Y_gd
    dW = (1/m) * np.dot(E, X_gd.T)
    db = (1/m) * np.sum(E)
    W -= alpha * dW
    b -= alpha * db

print("Final parameters (manual GD):")
print("W:", W, " b:", b)

# TensorFlow GradientTape
W_tf = tf.Variable([[0.0, 0.0]])
b_tf = tf.Variable(0.0)

X_tf = tf.constant(X_gd.T, dtype=tf.float32)  # (m, 2)
Y_tf = tf.constant(Y, dtype=tf.float32)       # (m, 1)

optimizer = tf.keras.optimizers.SGD(learning_rate=0.01)
```

```python
for epoch in range(1000):
    with tf.GradientTape() as tape:
        Y_pred = tf.matmul(X_tf, tf.transpose(W_tf)) + b_tf
        loss = tf.reduce_mean(tf.square(Y_pred - Y_tf))
    grads = tape.gradient(loss, [W_tf, b_tf])
    optimizer.apply_gradients(zip(grads, [W_tf, b_tf]))

print("Final parameters (TF GD):")
print("W:", W_tf.numpy(), " b:", b_tf.numpy())
```

```
Final parameters (manual GD):
W: [[  3.33859658 -10.38361448]]  b: 37.78556232358561
Final parameters (TF GD):
W: [[ -2.3015294 -21.340294 ]]  b: 42.05744
```

Define a class to create a Linear Regression model with methods fit and predict. Use the above iterative process to implement the model's training within the fit method.

In [31]:
```python
class MyLinearRegression:
    def __init__(self, lr=0.01, epochs=1000):
        self.lr = lr
        self.epochs = epochs

    def fit(self, X, Y):
        m, n = X.shape
        self.W = np.zeros((1, n))
        self.b = 0.0
        for i in range(self.epochs):
            Y_hat = np.dot(self.W, X.T) + self.b
            E = Y_hat - Y.T
            dW = (1/m) * np.dot(E, X)
            db = (1/m) * np.sum(E)
            self.W -= self.lr * dW
            self.b -= self.lr * db

    def predict(self, X):
        return np.dot(self.W, X.T) + self.b

# Example usage
model = MyLinearRegression(lr=0.01, epochs=1000)
model.fit(X[:,1:], Y)  # use only features (skip bias column)
preds = model.predict(X[:,1:])
print("Predictions shape:", preds.shape)
```

```
Predictions shape: (1, 415)
```