Fetaure Transformation

There are so many types of feature transformation methods, we will talk about the most useful and popular ones.But, sometime Feature Scaling - standardization & Normalization also included in feature transformation so, below are some feature transformation technique:

A. Feature Scaling

Standardization and Normalization

```
1. Standardization

2. Normalization:

    2.1 Min – Max Scaling

    2.2 Robust Scaler

    (As well as there are others like Mean Normalization, Max Absolute)
```

B. Gaussian Transformation/Mathematical Transformation

```
3. Logarithmic Transformation

4. Reciprocal Transformation

5. Square Root Translation

6. Box-Cox Transformation
```

1. Standardization

In [3]:
```
import pandas as pd
import numpy as np
```

In [4]:
```
df=pd.read_csv(r"C:\Users\USER\Downloads\house-prices.csv")
df.head()
```

Out[4]:

	Home	Price	SqFt	Bedrooms	Bathrooms	Offers	Brick	Neighborhood
0	1	114300	1790	2	2	2	No	East
1	2	114200	2030	4	2	3	No	East
2	3	114800	1740	3	2	1	No	East
3	4	94700	1980	3	2	3	No	East
4	5	119800	2130	3	3	3	No	East

In [5]:
```
df= df.drop(['SqFt', 'Offers', 'Brick','Neighborhood','Home'], axis=1)
```

In [6]:
```
df.head()
```

Out[6]:

	Price	Bedrooms	Bathrooms
0	114300	2	2
1	114200	4	2
2	114800	3	2
3	94700	3	2
4	119800	3	3

Now, here we can see huge difference between Price and Bedroom, Bathroom , lets see numbers and Price is just a targeted variable

In [7]:
```
df.describe()
```

Out[7]:

	Price	Bedrooms	Bathrooms
count	128.000000	128.000000	128.000000
mean	130427.343750	3.023438	2.445312
std	26868.770371	0.725951	0.514492
min	69100.000000	2.000000	2.000000
25%	111325.000000	3.000000	2.000000
50%	125950.000000	3.000000	2.000000
75%	148250.000000	3.000000	3.000000
max	211200.000000	5.000000	4.000000

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
Price        0
Bedrooms     0
Bathrooms    0
dtype: int64
```

Lets separate data into X_train,X_test,y_train and y_test

In [9]:
```
X=df.iloc[:,1:3]
```

In [10]:
```
X
```

Out[10]:

	Bedrooms	Bathrooms
0	2	2
1	4	2
2	3	2
3	3	2
4	3	3
...	...	...
123	3	3
124	4	3
125	2	2
126	3	3
127	3	3

128 rows × 2 columns

In [11]:
```
y=df.iloc[:,0]
```

In [12]:
```
y
```

Out[12]:
```
0      114300
1      114200
2      114800
3       94700
4      119800
        ...
123    119700
124    147900
125    113500
126    149900
127    124600
Name: Price, Length: 128, dtype: int64
```

In [13]:
```
from sklearn.model_selection import train_test_split
```

In [14]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

In [15]:
```
X_train.describe()
```

Out[15]:

	Bedrooms	Bathrooms
count	102.000000	102.000000
mean	3.058824	2.480392
std	0.728807	0.521430
min	2.000000	2.000000
25%	3.000000	2.000000
50%	3.000000	2.000000
75%	3.750000	3.000000
max	5.000000	4.000000

In [16]:
```
X_test.describe()
```

Out[16]:

	Bedrooms	Bathrooms
count	26.000000	26.000000
mean	2.884615	2.307692
std	0.711445	0.470679
min	2.000000	2.000000
25%	2.000000	2.000000
50%	3.000000	2.000000
75%	3.000000	3.000000
max	4.000000	3.000000

Now standarizing data to bring down where mean will be 0(Zero) and standard deviation is 1(One).

In [17]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
```

In [18]:
```
X_trained_scaled=sc.fit_transform(X_train)
X_test_scaled=sc.transform(X_test)
```

Now lets checked mean and standard deviation of scaled datasets.

In [19]:
```
X_trained_scaled.describe()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[19], line 1
----> 1 X_trained_scaled.describe()

AttributeError: 'numpy.ndarray' object has no attribute 'describe'
```

its numpy so we have to changed it in dataframe.

In [20]:
```
X_trained_scaled=pd.DataFrame(X_trained_scaled,columns=X_train.columns)
```

In [21]:
```
X_trained_scaled.describe()
```

Out[21]:

	Bedrooms	Bathrooms
count	1.020000e+02	1.020000e+02
mean	3.593259e-16	-9.143013e-17
std	1.004938e+00	1.004938e+00
min	-1.459993e+00	-9.258476e-01
25%	-8.111071e-02	-9.258476e-01
50%	-8.111071e-02	-9.258476e-01
75%	9.530508e-01	1.001427e+00
max	2.676653e+00	2.928702e+00

In [22]:
```
#Lets do round using numpy
np.round(X_trained_scaled.describe(),1)
```

Out[22]:

	Bedrooms	Bathrooms
count	102.0	102.0
mean	0.0	-0.0
std	1.0	1.0
min	-1.5	-0.9
25%	-0.1	-0.9
50%	-0.1	-0.9
75%	1.0	1.0
max	2.7	2.9

Statistically Standardization Formula is:

$$X_{std} = \frac{X - X_{mean}}{X_{stddev}}$$

2. Min—Max Scaling/Normalization

In [23]:
```
# Here also we can use same data
df1=pd.read_csv(r"C:\Users\USER\Downloads\house-prices.csv")
```

In [24]:
```
df1=df1.iloc[:,[1,3,4]]
```

In [25]:
```
df1.head()
```

Out[25]:

	Price	Bedrooms	Bathrooms
0	114300	2	2
1	114200	4	2
2	114800	3	2
3	94700	3	2
4	119800	3	3

In [26]:
```
X=df.iloc[:,1:3]
y=df.iloc[:,0]
```

In [27]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
from sklearn.preprocessing import MinMaxScaler
sc=MinMaxScaler()
```

In [28]:
```
#we fit and train to X_train only and transform to both X_train & X_test
X_train_scaled=sc.fit_transform(X_train)
X_test_scaled=sc.transform(X_test)
```

In [29]:
```
X_train_scaled=pd.DataFrame(X_train_scaled,columns=X_train.columns)
X_test_scaled=pd.DataFrame(X_test_scaled,columns=X_test.columns)
```

In [30]:
```
np.round(X_train_scaled.describe(),1)
```

Out[30]:

	Bedrooms	Bathrooms
count	102.0	102.0
mean	0.4	0.2
std	0.2	0.3
min	0.0	0.0
25%	0.3	0.0
50%	0.3	0.0
75%	0.6	0.5
max	1.0	1.0

Statistical formula for MinMax Scaler is:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

In simple terms, min-max scaling brings down feature values to a range of 0 to 1. If the dataset has too many outliers, both Standardization and Normalization can be hard to depend on, in such case you can use Robust Scaler for feature scaling.

3. Robust Scaler

Statistical formula for Robust Scaler is:

The robust scaler subtracts feature values by their median and then divides by its IQR.

- 25th percentile = 1st quartile
- 50th percentile = 2nd quartile (also called the median)
- 75th percentile = 3rd quartile
- 100th percentile = 4th quartile (also called the maximum)
- IQR= Inter Quartile Range
- IQR= 3rd quartile—1st quartile

$$X_{robust} = \frac{X - X_{median}}{X_{75th} - X_{25th}}$$

$$where\ X_{75th} - X_{25th} = Interquartile\ range$$

Before moving to Robust scaler, lets look into original datasets graphs

In [31]:

```
import numpy as np
```

```python
import pandas as pd
import seaborn as sns
```

In [32]:
```python
df_orginal=pd.read_csv(r"C:\Users\USER\Downloads\house-prices.csv")
df_orginal= df_orginal.drop(['SqFt', 'Offers', 'Brick','Neighborhood','Home'], axis=1)
```

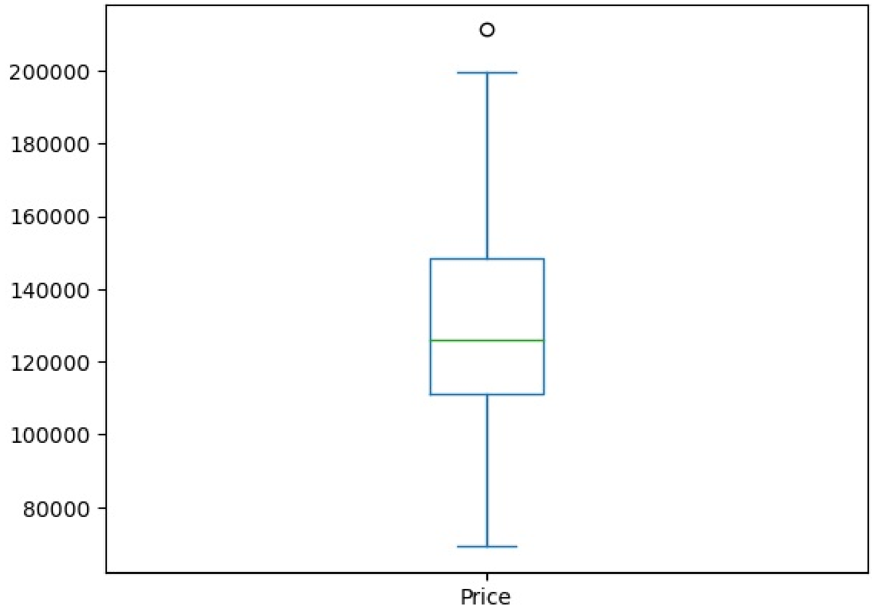
In [33]:
```python
df_orginal.head()
```

Out[33]:

	Price	Bedrooms	Bathrooms
0	114300	2	2
1	114200	4	2
2	114800	3	2
3	94700	3	2
4	119800	3	3

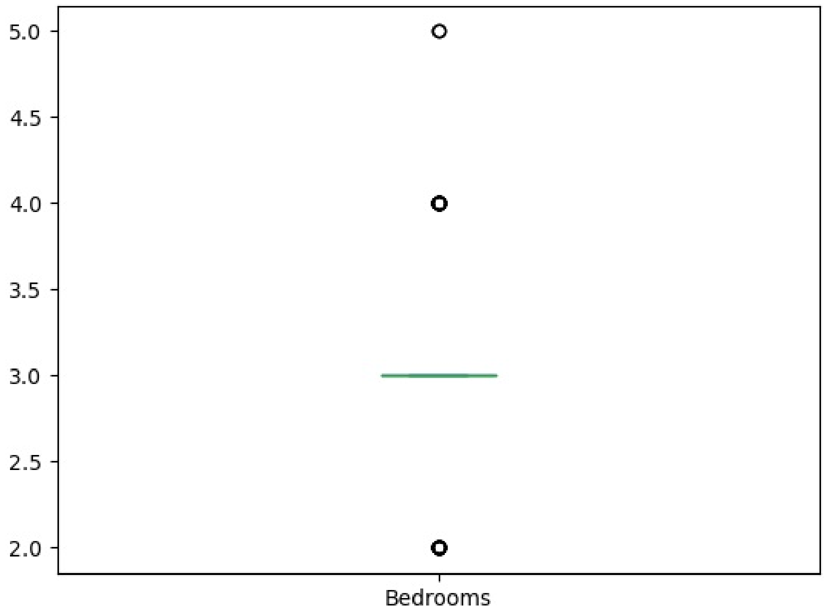
In [34]:
```python
df_orginal['Price'].plot(kind='box')
```

Out[34]: <Axes: >



In [35]:
```python
df_orginal['Bedrooms'].plot(kind='box')
```

Out[35]: <Axes: >



In [36]:
```python
df_orginal['Bathrooms'].plot(kind='box')
```

Out[36]: <Axes: >

We can see outlier in Price and Bedrooms, but Standardization & Normalization(MinMax Scaler) are sensitivie to outliers. Both standardization and Normalization somewhat shrinks the data,Standardization make mean & standard deviation to 0 and 1. Similarly, MinMax Scaler brings all datasets within 0 and 1

In [37]:
```
X=df_orginal.iloc[:,1:3]
y=df_orginal.iloc[:,0]
```

In [38]:
```
X     #Just checking for conformation
```

Out[38]:

	Bedrooms	Bathrooms
0	2	2
1	4	2
2	3	2
3	3	2
4	3	3
...	...	...
123	3	3
124	4	3
125	2	2
126	3	3
127	3	3

128 rows × 2 columns

In [39]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
from sklearn.preprocessing import RobustScaler
rb=RobustScaler()
```

In [40]:
```
#we fit and train to X_train only and transform to both X_train & X_test
X_train_scaled=rb.fit_transform(X_train)
X_test_scaled=rb.transform(X_test)
```

In [41]:
```
X_train_scaled=pd.DataFrame(X_train_scaled,columns=X_train.columns)
X_test_scaled=pd.DataFrame(X_test_scaled,columns=X_test.columns)
```

In [42]:
```
np.round(X_train_scaled.describe(),1)
```

Out[42]:

	Bedrooms	Bathrooms
count	102.0	102.0
mean	0.1	0.5
std	1.0	0.5
min	-1.3	0.0
25%	0.0	0.0
50%	0.0	0.0
75%	1.0	1.0
max	2.7	2.0

We can see that the distributions have been adjusted. The median values are now zero and the standard deviation values are now 1 or close to 1.0.

Gaussian Transformation / Mathematical Transformation

- Logarithmic Transformation
- Reciprocal Transformation
- Square Root Translation
- Box-Cox Transformation

In [43]:

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as py
import seaborn as sns
```

In [44]:

```
d3=pd.read_csv(r"C:\Users\USER\Downloads\Real estate.csv")
d3.head()
```

Out[44]:

	No	X1 transaction date	X2 house age	X3 distance to the nearest MRT station	X4 number of convenience stores	X5 latitude	X6 longitude	Y house price of unit area
0	1	2012.917	32.0	84.87882	10	24.98298	121.54024	37.9
1	2	2012.917	19.5	306.59470	9	24.98034	121.53951	42.2
2	3	2013.583	13.3	561.98450	5	24.98746	121.54391	47.3
3	4	2013.500	13.3	561.98450	5	24.98746	121.54391	54.8
4	5	2012.833	5.0	390.56840	5	24.97937	121.54245	43.1

In [45]:

```
d3=d3.drop(['No','X1 transaction date','X5 latitude','X6 longitude'],axis=1)
```

In [46]:

```
d3.head()
```

Out[46]:

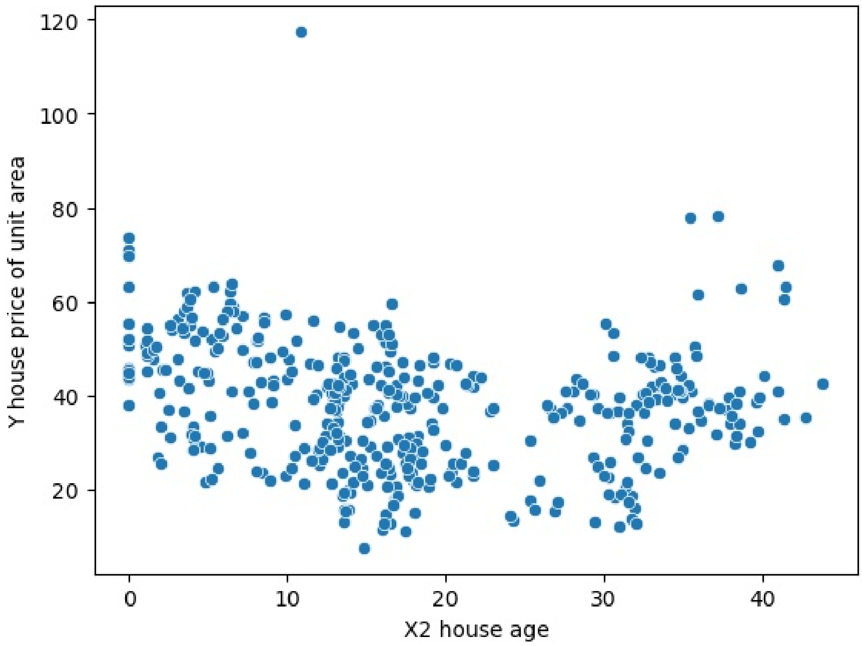
	X2 house age	X3 distance to the nearest MRT station	X4 number of convenience stores	Y house price of unit area
0	32.0	84.87882	10	37.9
1	19.5	306.59470	9	42.2
2	13.3	561.98450	5	47.3
3	13.3	561.98450	5	54.8
4	5.0	390.56840	5	43.1

In [47]:

```
sns.scatterplot(x='X2 house age', y='Y house price of unit area',data=d3)
```

Out[47]:

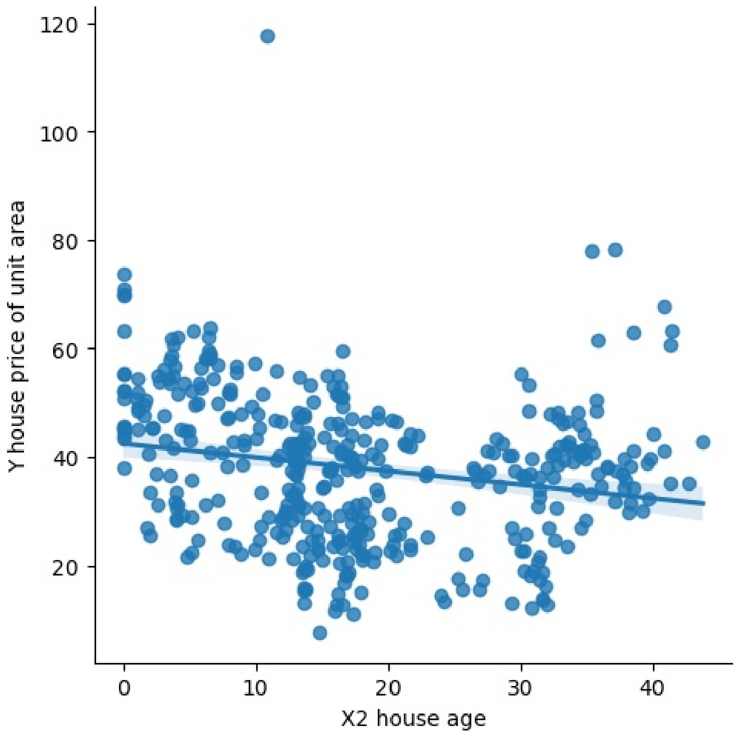
```
<Axes: xlabel='X2 house age', ylabel='Y house price of unit area'>
```



In [48]:
```
#lmplot is better than scatterplot, it will show how our data is distance/scatter from normal curve
sns.lmplot(x='X2 house age', y='Y house price of unit area',data=d3)
```

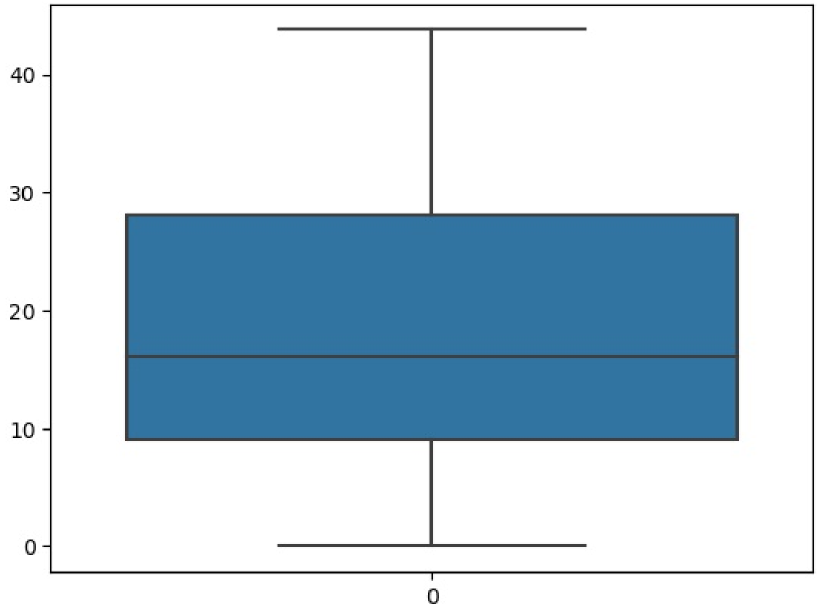
```
C:\Users\USER\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed t
o tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[48]:
```
<seaborn.axisgrid.FacetGrid at 0x252c442bc50>
```



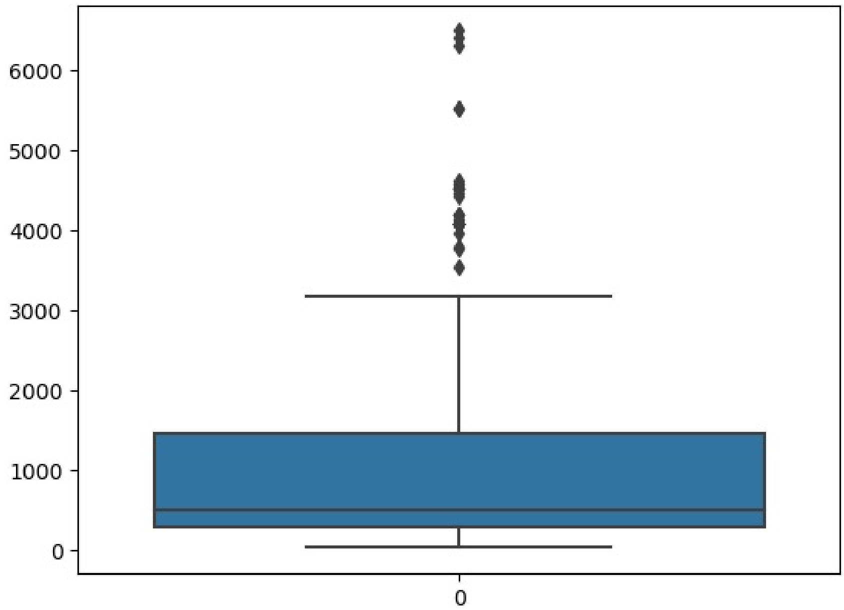
In [49]:
```
sns.boxplot(d3["X2 house age"])
```

Out[49]: <Axes: >



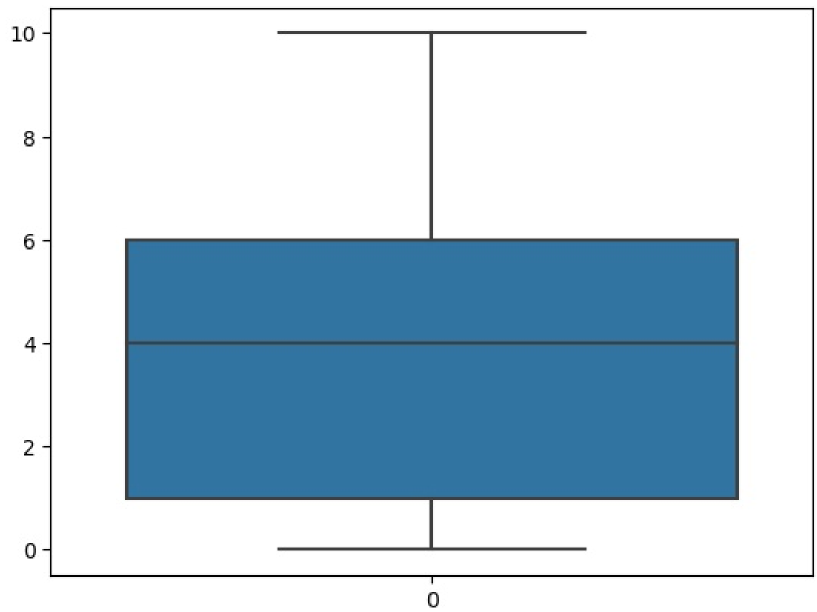
In [50]:
```
sns.boxplot(d3["X3 distance to the nearest MRT station"])
```

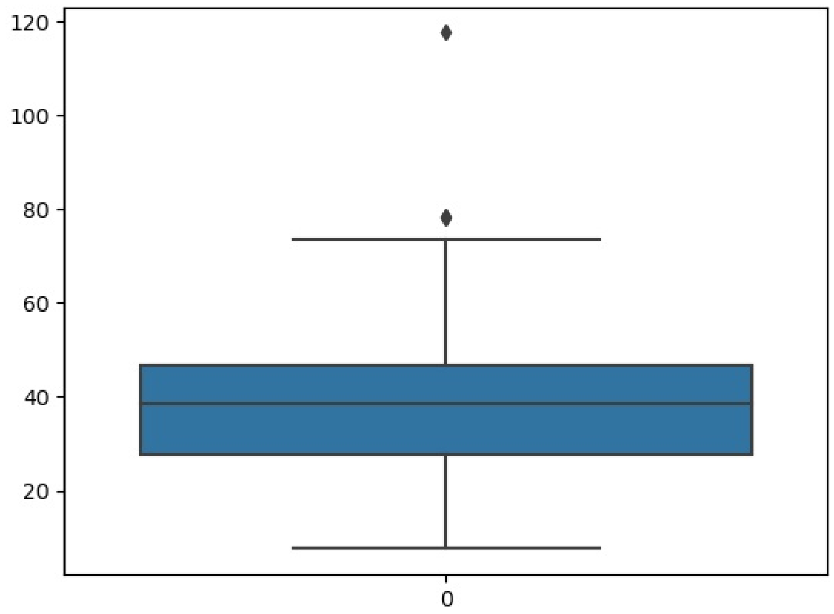
Out[50]: <Axes: >



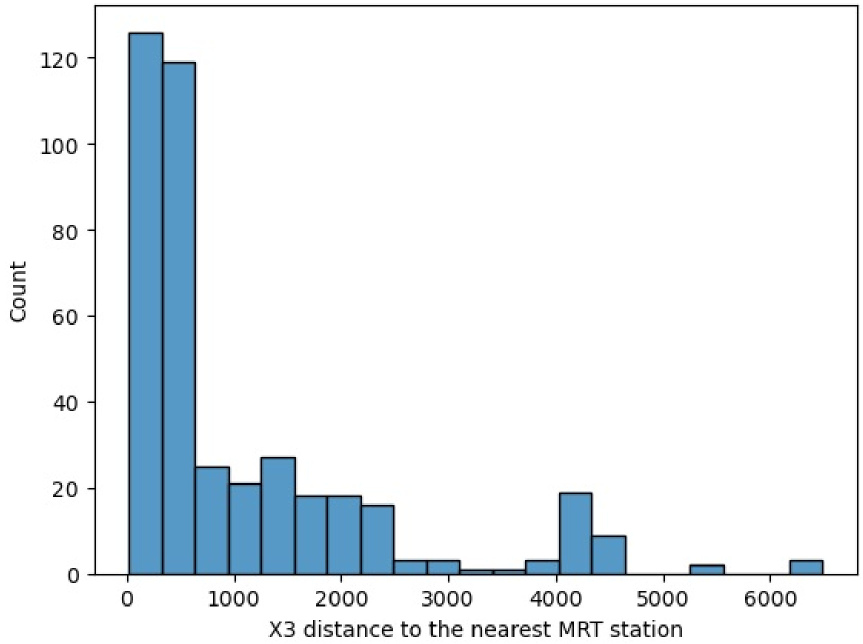
In [51]:
```
sns.boxplot(d3["X4 number of convenience stores"])
```

Out[51]: <Axes: >



In [52]:
```
sns.boxplot(d3["Y house price of unit area"])
```
Out[52]:
```
<Axes: >
```


Lets check histogram

In [53]:
```
sns.histplot(data=d3, x="X3 distance to the nearest MRT station")
```
Out[53]:
```
<Axes: xlabel='X3 distance to the nearest MRT station', ylabel='Count'>
```


This histogram is rightly sekewed, means not gussian so for this type of data set log transformation is used.

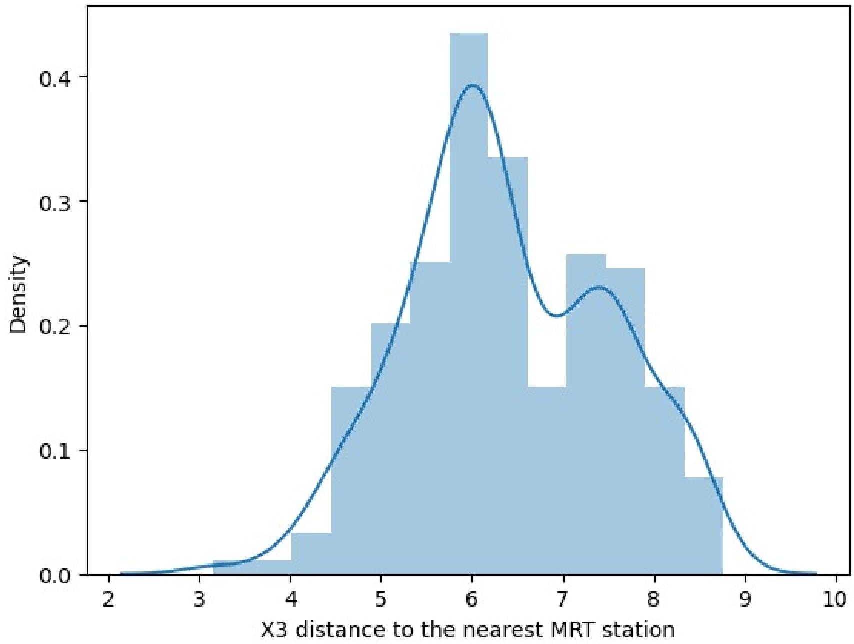
In [54]:
```
#Apply log transformation
from sklearn.preprocessing import FunctionTransformer
```
In [55]:
```
log_transformer = FunctionTransformer(np.log)
```
In [60]:
```
X=d3['X3 distance to the nearest MRT station']
```
In [61]:
```
log_X = log_transformer.transform(X)
```
In [64]:
```
log_X
```

Out[64]:
```
0      4.441225
1      5.725527
2      6.331474
3      6.331474
4      5.967603
         ...
409    8.314346
410    4.504864
411    5.968630
412    4.652150
413    4.504864
Name: X3 distance to the nearest MRT station, Length: 414, dtype: float64
```

In [67]:
```
sns.distplot(log_X)
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_12204\3898673339.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(log_X)
```

Out[67]:
```
<Axes: xlabel='X3 distance to the nearest MRT station', ylabel='Density'>
```



In [68]:
```
# Now see the difference between above histogram and below.
```

Similarly we can use other transformation - Mathematical, we will try while doing project practice.