Import all required liabraries

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

load the dataset california housing

In [2]:
```
from sklearn.datasets import fetch_california_housing
housing = fetch_california_housing()
```

In [3]:
```
type(housing)
```

Out[3]:
```
sklearn.utils._bunch.Bunch
```

In [4]:
```
housing
```

Out[4]:
```
{'data': array([[   8.3252    ,   41.        ,    6.98412698, ...,    2.55555556,
          37.88      , -122.23      ],
       [   8.3014    ,   21.        ,    6.23813708, ...,    2.10984183,
          37.86      , -122.22      ],
       [   7.2574    ,   52.        ,    8.28813559, ...,    2.80225989,
          37.85      , -122.24      ],
       ...,
       [   1.7       ,   17.        ,    5.20554273, ...,    2.3256351 ,
          39.43      , -121.22      ],
       [   1.8672    ,   18.        ,    5.32951289, ...,    2.12320917,
          39.43      , -121.32      ],
       [   2.3886    ,   16.        ,    5.25471698, ...,    2.61698113,
          39.37      , -121.24      ]]),
 'target': array([4.526, 3.585, 3.521, ..., 0.923, 0.847, 0.894]),
 'frame': None,
 'target_names': ['MedHouseVal'],
 'feature_names': ['MedInc',
  'HouseAge',
  'AveRooms',
  'AveBedrms',
  'Population',
  'AveOccup',
  'Latitude',
  'Longitude'],
 'DESCR': '.. _california_housing_dataset:\n\nCalifornia Housing dataset\n--------
------------------\n\n**Data Set Characteristics:**\n\n:Number of Instances: 20640
\n\n:Number of Attributes: 8 numeric, predictive attributes and the target\n\n:Att
ribute Information:\n    - MedInc        median income in block group\n    - House
Age      median house age in block group\n    - AveRooms      average number of ro
oms per household\n    - AveBedrms     average number of bedrooms per household\n
- Population    block group population\n    - AveOccup      average number of hous
ehold members\n    - Latitude      block group latitude\n    - Longitude     block
group longitude\n\n:Missing Attribute Values: None\n\nThis dataset was obtained fr
om the StatLib repository.\nhttps://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housin
g.html\n\nThe target variable is the median house value for California district
s,\nexpressed in hundreds of thousands of dollars ($100,000).\n\nThis dataset was
derived from the 1990 U.S. census, using one row per census\nblock group. A block
group is the smallest geographical unit for which the U.S.\nCensus Bureau publishe
s sample data (a block group typically has a population\nof 600 to 3,000 peopl
e).\n\nA household is a group of people residing within a home. Since the average
\nnumber of rooms and bedrooms in this dataset are provided per household, these\n
columns may take surprisingly large values for block groups with few households\na
nd many empty houses, such as vacation resorts.\n\nIt can be downloaded/loaded usi
ng the\n:func:`sklearn.datasets.fetch_california_housing` function.\n\n.. rubric::
References\n\n- Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregression
s,\n  Statistics and Probability Letters, 33 (1997) 291-297\n'}
```

In [5]:
```python
print(housing.DESCR)
```

```
.. _california_housing_dataset:

California Housing dataset
--------------------------

**Data Set Characteristics:**

:Number of Instances: 20640

:Number of Attributes: 8 numeric, predictive attributes and the target

:Attribute Information:
    - MedInc        median income in block group
    - HouseAge      median house age in block group
    - AveRooms      average number of rooms per household
    - AveBedrms     average number of bedrooms per household
    - Population    block group population
    - AveOccup      average number of household members
    - Latitude      block group latitude
    - Longitude     block group longitude

:Missing Attribute Values: None

This dataset was obtained from the StatLib repository.
https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html

The target variable is the median house value for California districts,
expressed in hundreds of thousands of dollars ($100,000).

This dataset was derived from the 1990 U.S. census, using one row per census
block group. A block group is the smallest geographical unit for which the U.S.
Census Bureau publishes sample data (a block group typically has a population
of 600 to 3,000 people).

A household is a group of people residing within a home. Since the average
number of rooms and bedrooms in this dataset are provided per household, these
columns may take surprisingly large values for block groups with few households
and many empty houses, such as vacation resorts.

It can be downloaded/loaded using the
:func:`sklearn.datasets.fetch_california_housing` function.

.. rubric:: References

- Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,
  Statistics and Probability Letters, 33 (1997) 291-297
```

In [6]:
```
print(housing.feature_names)
```

```
['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup', 'Latitud
e', 'Longitude']
```

In [7]:
```
print(housing.target)
```

```
[4.526 3.585 3.521 ... 0.923 0.847 0.894]
```

In [8]:
```
print(housing.data)
```

```
[[   8.3252       41.           6.98412698 ...    2.55555556
    37.88       -122.23      ]
 [   8.3014       21.           6.23813708 ...    2.10984183
    37.86       -122.22      ]
 [   7.2574       52.           8.28813559 ...    2.80225989
    37.85       -122.24      ]
 ...
 [   1.7          17.           5.20554273 ...    2.3256351
    39.43       -121.22      ]
 [   1.8672       18.           5.32951289 ...    2.12320917
    39.43       -121.32      ]
 [   2.3886       16.           5.25471698 ...    2.61698113
    39.37       -121.24      ]]
```

prepare the data

In [10]:
```
dataset = pd.DataFrame(housing.data, columns=housing.feature_names)
```

In [11]:
```
type(dataset)
```

Out[11]:
```
pandas.core.frame.DataFrame
```

In [12]:
```
dataset.head()
```

Out[12]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25

In [13]:
```
dataset.tail()
```

Out[13]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Long
20635	1.5603	25.0	5.045455	1.133333	845.0	2.560606	39.48	-1
20636	2.5568	18.0	6.114035	1.315789	356.0	3.122807	39.49	-1
20637	1.7000	17.0	5.205543	1.120092	1007.0	2.325635	39.43	-1
20638	1.8672	18.0	5.329513	1.171920	741.0	2.123209	39.43	-1
20639	2.3886	16.0	5.254717	1.162264	1387.0	2.616981	39.37	-1

In [14]:
```python
dataset['Price'] = housing.target
dataset.head()
```

Out[14]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25

In [15]:
```python
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   MedInc      20640 non-null  float64
 1   HouseAge    20640 non-null  float64
 2   AveRooms    20640 non-null  float64
 3   AveBedrms   20640 non-null  float64
 4   Population  20640 non-null  float64
 5   AveOccup    20640 non-null  float64
 6   Latitude    20640 non-null  float64
 7   Longitude   20640 non-null  float64
 8   Price       20640 non-null  float64
dtypes: float64(9)
memory usage: 1.4 MB
```

In [17]:
```python
dataset.describe()
```

Out[17]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup
count	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	3.870671	28.639486	5.429000	1.096675	1425.476744	3.070655
std	1.899822	12.585558	2.474173	0.473911	1132.462122	10.386050
min	0.499900	1.000000	0.846154	0.333333	3.000000	0.692308
25%	2.563400	18.000000	4.440716	1.006079	787.000000	2.429741
50%	3.534800	29.000000	5.229129	1.048780	1166.000000	2.818116
75%	4.743250	37.000000	6.052381	1.099526	1725.000000	3.282261
max	15.000100	52.000000	141.909091	34.066667	35682.000000	1243.333333

In [19]:
```
## check the null value
dataset.isnull().sum()
```

Out[19]:
```
MedInc        0
HouseAge      0
AveRooms      0
AveBedrms     0
Population    0
AveOccup      0
Latitude      0
Longitude     0
Price         0
dtype: int64
```

EDA(Exploratory Data Analysis)

AvgBedrooms and Avgrooms : 0.847621

Longitude and Latitude : -0.924664

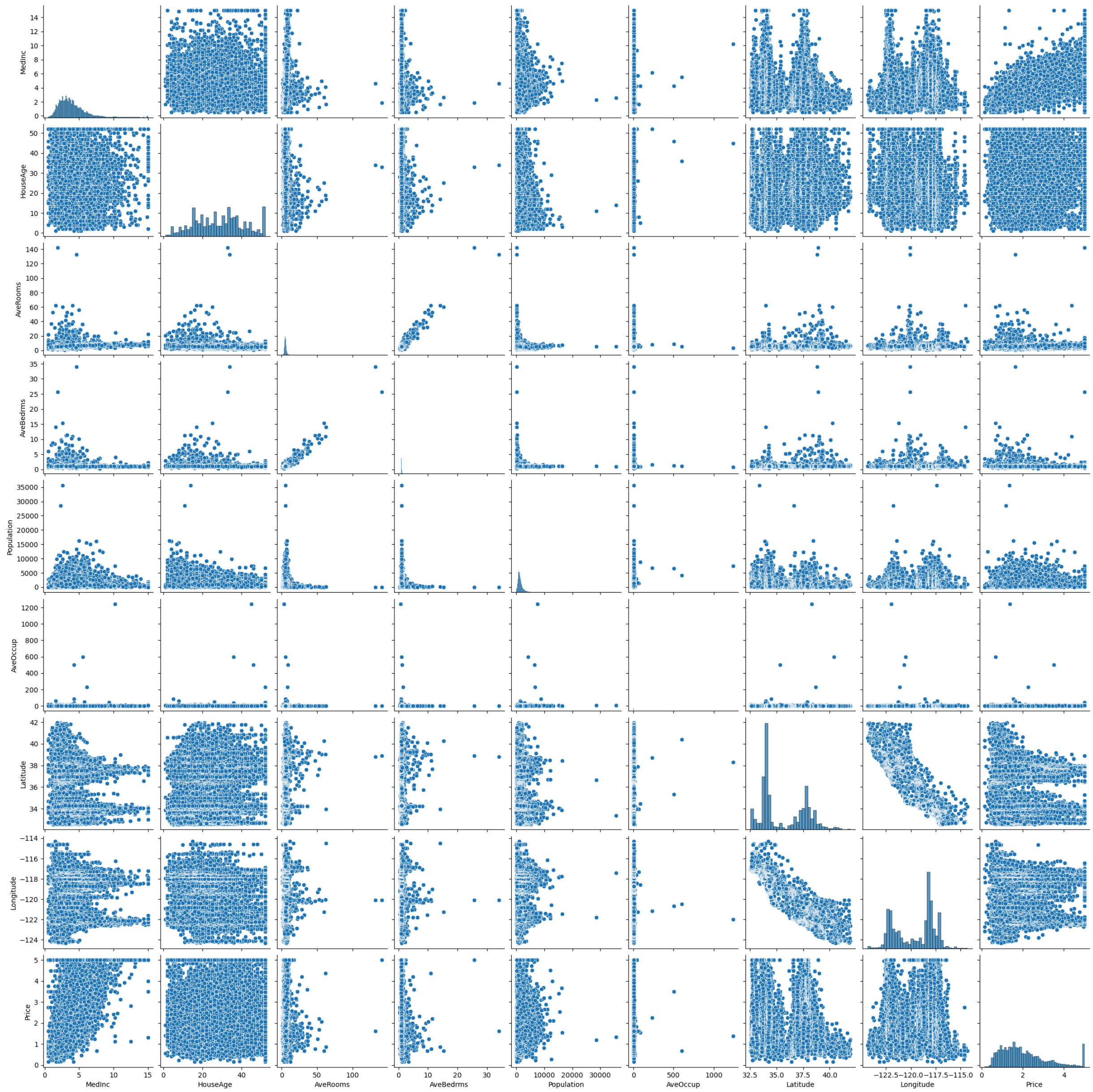
In [20]:
```
dataset.corr()
```

Out[20]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitud
MedInc	1.000000	-0.119034	0.326895	-0.062040	0.004834	0.018766	-0.07980
HouseAge	-0.119034	1.000000	-0.153277	-0.077747	-0.296244	0.013191	0.01117
AveRooms	0.326895	-0.153277	1.000000	0.847621	-0.072213	-0.004852	0.10638
AveBedrms	-0.062040	-0.077747	0.847621	1.000000	-0.066197	-0.006181	0.06972
Population	0.004834	-0.296244	-0.072213	-0.066197	1.000000	0.069863	-0.10878
AveOccup	0.018766	0.013191	-0.004852	-0.006181	0.069863	1.000000	0.00236
Latitude	-0.079809	0.011173	0.106389	0.069721	-0.108785	0.002366	1.00000
Longitude	-0.015176	-0.108197	-0.027540	0.013344	0.099773	0.002476	-0.92466
Price	0.688075	0.105623	0.151948	-0.046701	-0.024650	-0.023737	-0.14416

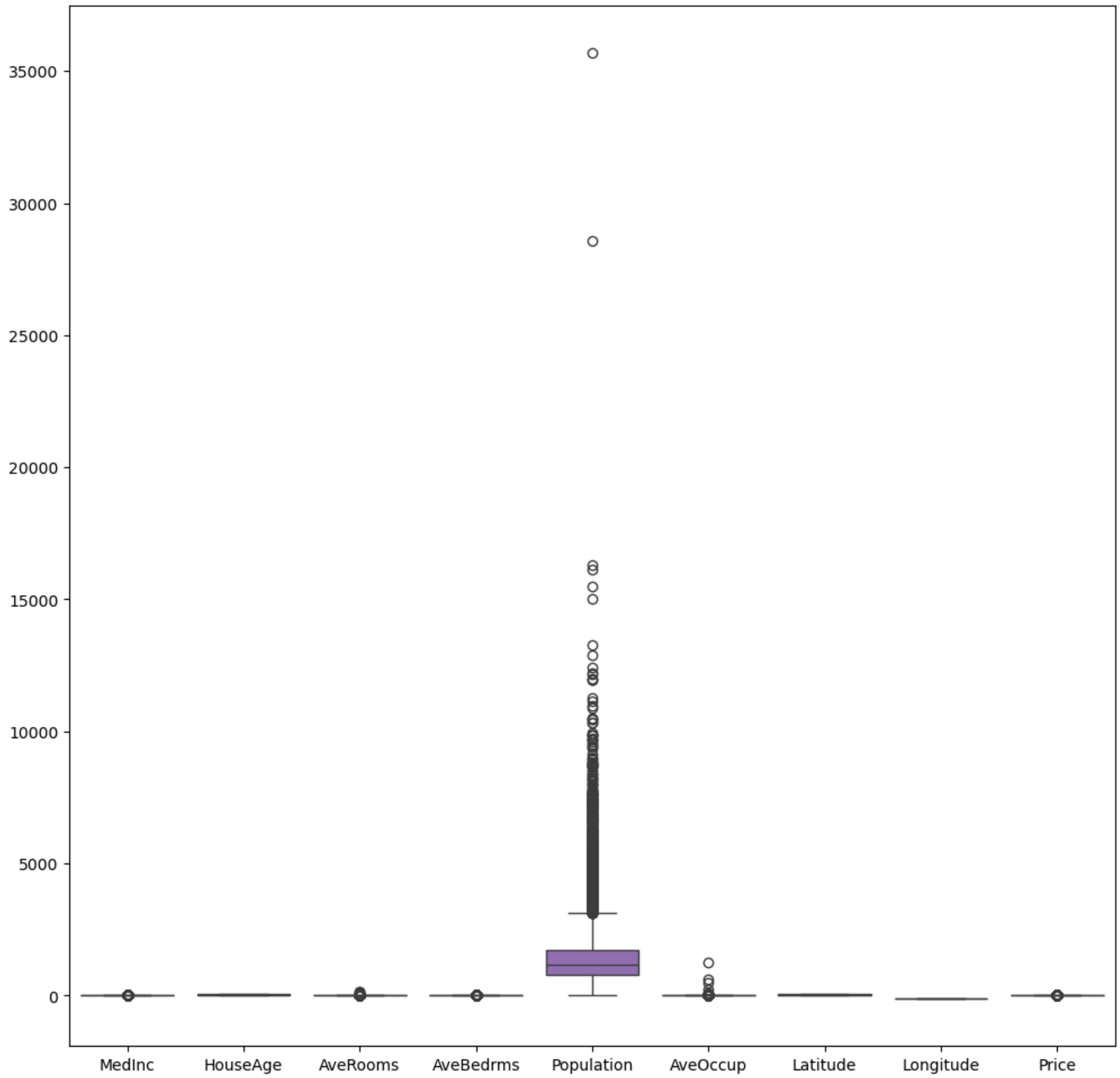
In [21]:
```
sns.pairplot(dataset)
```

Out[21]: <seaborn.axisgrid.PairGrid at 0x1e1d92b2f60>



Boxplot : To detect the outliers in a given dataset

In [24]:
```
fig, ax = plt.subplots(figsize=(12,12))
sns.boxplot(data = dataset, ax=ax)
plt.savefig("boxplot.jpg")
```



In [25]:
```
## split the data into dependent and independent feature
x = dataset.iloc[:,:-1]
y = dataset.iloc[:,-1]
```

In [26]:
```
x
```

Out[26]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Long
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-1
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-1
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-1
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-1
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-1
...	...	...	...	...	...	...	...	
20635	1.5603	25.0	5.045455	1.133333	845.0	2.560606	39.48	-1
20636	2.5568	18.0	6.114035	1.315789	356.0	3.122807	39.49	-1
20637	1.7000	17.0	5.205543	1.120092	1007.0	2.325635	39.43	-1
20638	1.8672	18.0	5.329513	1.171920	741.0	2.123209	39.43	-1
20639	2.3886	16.0	5.254717	1.162264	1387.0	2.616981	39.37	-1

20640 rows × 8 columns

In [27]:

```
y
```

Out[27]:

```
0        4.526
1        3.585
2        3.521
3        3.413
4        3.422
         ...
20635    0.781
20636    0.771
20637    0.923
20638    0.847
20639    0.894
Name: Price, Length: 20640, dtype: float64
```

In [28]:

```
## split the data into training and testing set
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_sta
```

In [29]:

```
x_train
```

Out[29]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Long
7061	4.1312	35.0	5.882353	0.975490	1218.0	2.985294	33.93	-1
14689	2.8631	20.0	4.401210	1.076613	999.0	2.014113	32.79	-1
17323	4.2026	24.0	5.617544	0.989474	731.0	2.564912	34.59	-1
10056	3.1094	14.0	5.869565	1.094203	302.0	2.188406	39.26	-1
15750	3.3068	52.0	4.801205	1.066265	1526.0	2.298193	37.77	-1
...	...	...	...	...	...	...	...	
11284	6.3700	35.0	6.129032	0.926267	658.0	3.032258	33.78	-1
11964	3.0500	33.0	6.868597	1.269488	1753.0	3.904232	34.02	-1
5390	2.9344	36.0	3.986717	1.079696	1756.0	3.332068	34.03	-1
860	5.7192	15.0	6.395349	1.067979	1777.0	3.178891	37.58	-1
15795	2.5755	52.0	3.402576	1.058776	2619.0	2.108696	37.77	-1

14448 rows × 8 columns

In [30]:
```
x_test
```

Out[30]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Long
20046	1.6812	25.0	4.192201	1.022284	1392.0	3.877437	36.06	-1
3024	2.5313	30.0	5.039384	1.193493	1565.0	2.679795	35.14	-1
15663	3.4801	52.0	3.977155	1.185877	1310.0	1.360332	37.80	-1
20484	5.7376	17.0	6.163636	1.020202	1705.0	3.444444	34.28	-1
9814	3.7250	34.0	5.492991	1.028037	1063.0	2.483645	36.62	-1
...	...	...	...	...	...	...	...	
17505	2.9545	47.0	4.195833	1.020833	581.0	2.420833	37.36	-1
13512	1.4891	41.0	4.551852	1.118519	994.0	3.681481	34.11	-1
10842	3.5120	16.0	3.762287	1.075614	5014.0	2.369565	33.67	-1
16559	3.6500	10.0	5.502092	1.060371	5935.0	3.547519	37.82	-1
5786	3.0520	17.0	3.355781	1.019695	4116.0	2.614994	34.15	-1

6192 rows × 8 columns

In [31]:
```
y_train
```

Out[31]:
```
7061     1.93800
14689    1.69700
17323    2.59800
10056    1.36100
15750    5.00001
          ...
11284    2.29200
11964    0.97800
5390     2.22100
860      2.83500
15795    3.25000
Name: Price, Length: 14448, dtype: float64
```

In [32]:
```
y_test
```

Out[32]:
```
20046    0.47700
3024     0.45800
15663    5.00001
20484    2.18600
9814     2.78000
          ...
17505    2.37500
13512    0.67300
10842    2.18400
16559    1.19400
5786     2.09800
Name: Price, Length: 6192, dtype: float64
```

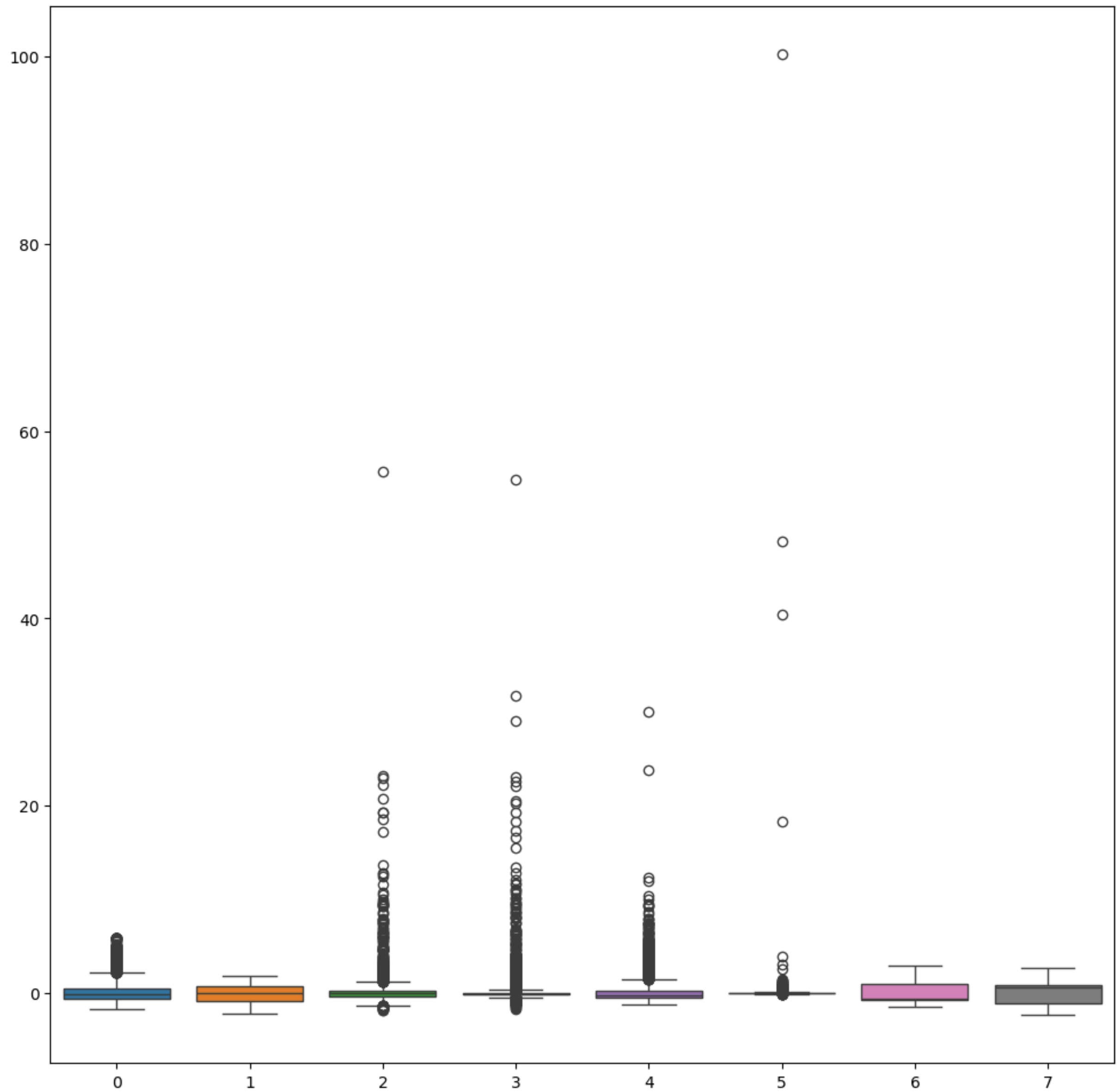
In [35]:
```
## Normalization of the given data points
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
x_train_norm = scaler.fit_transform(x_train)
```

In [36]:
```
x_train_norm
```

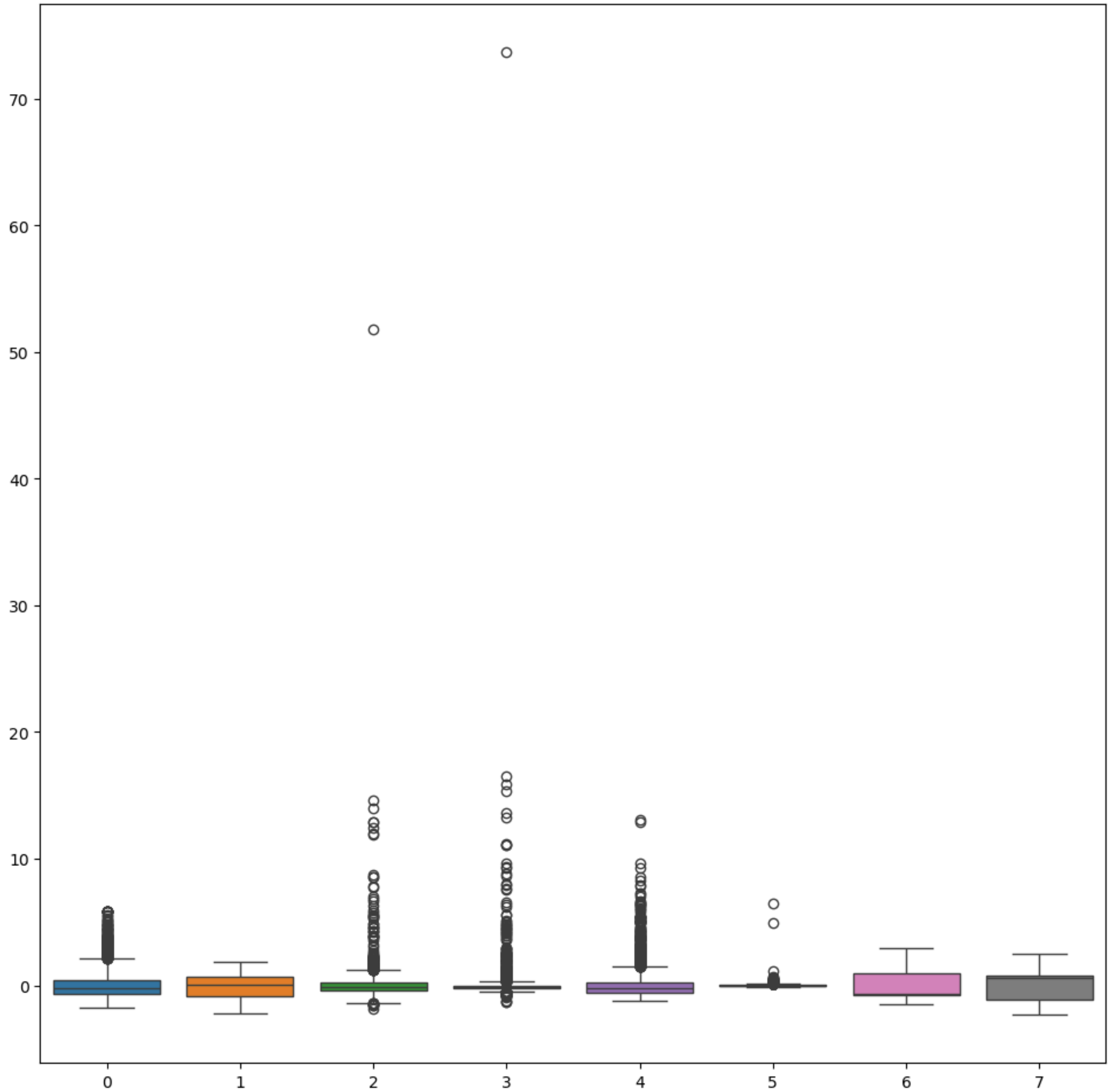
Out[36]:
```
array([[ 0.13350629,  0.50935748,  0.18106017, ..., -0.01082519,
        -0.80568191,  0.78093406],
       [-0.53221805, -0.67987313, -0.42262953, ..., -0.08931585,
        -1.33947268,  1.24526986],
       [ 0.1709897 , -0.36274497,  0.07312833, ..., -0.04480037,
        -0.49664515, -0.27755183],
       ...,
       [-0.49478713,  0.58863952, -0.59156984, ...,  0.01720102,
        -0.75885816,  0.60119118],
       [ 0.96717102, -1.07628333,  0.39014889, ...,  0.00482125,
         0.90338501, -1.18625198],
       [-0.68320166,  1.85715216, -0.82965604, ..., -0.0816717 ,
         0.99235014, -1.41592345]])
```

In [37]:
```
fig, ax = plt.subplots(figsize=(12,12))
sns.boxplot(data = x_train_norm, ax=ax)
plt.savefig("boxplotTrainData.jpg")
```



In [38]:
```
x_test_norm = scaler.transform(x_test)
```

In [39]:
```
fig, ax = plt.subplots(figsize=(12,12))
sns.boxplot(data = x_test_norm, ax=ax)
plt.savefig("boxplotTestData.jpg")
```



In [40]:
```
## Train set-> fit_transform(x_train)
## Test set -> transform(x_test)
## why usually this happen?
```

Model Training

In [42]:
```
from sklearn.linear_model import LinearRegression
regression = LinearRegression()
regression.fit(x_train_norm, y_train)
```

Out[42]:
```
▾ LinearRegression
LinearRegression()
```

In [44]:
```
print(regression.coef_)
```

```
[ 8.49221760e-01  1.22119309e-01 -2.99558449e-01  3.48409673e-01
 -8.84488134e-04 -4.16980388e-02 -8.93855649e-01 -8.68616688e-01]
```

In [45]:
```
print(regression.intercept_)
```

```
2.0692396089424165
```

Model Prediction

In [47]:
```
reg_pred = regression.predict(x_test_norm)
reg_pred
```

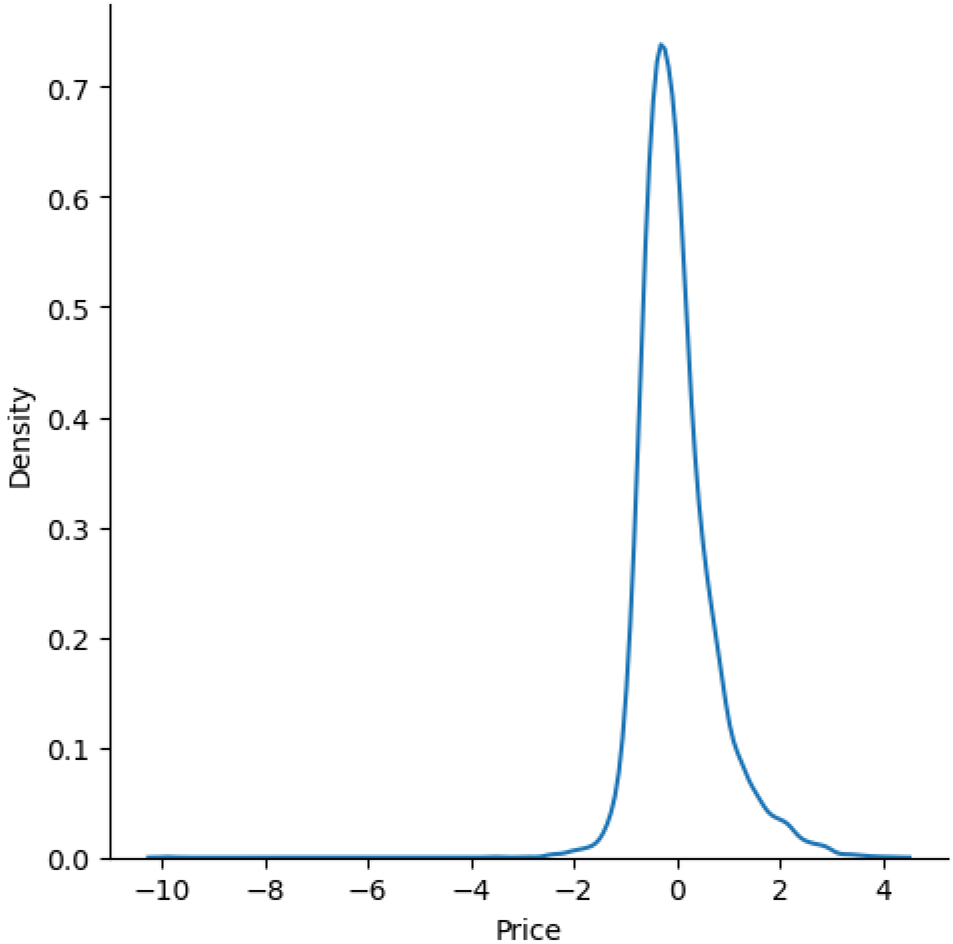
Out[47]:
```
array([0.72604907, 1.76743383, 2.71092161, ..., 2.07465531, 1.57371395,
       1.82744133])
```

In [48]:
```
## calculate the error or the residual
residuals = y_test - reg_pred
residuals
```

Out[48]:
```
20046   -0.249049
3024    -1.309434
15663    2.289088
20484   -0.649147
9814     0.173042
           ...
17505    0.155059
13512   -0.237516
10842    0.109345
16559   -0.379714
5786     0.270559
Name: Price, Length: 6192, dtype: float64
```

In [49]:
```
## Distribution plot of the residuals
sns.displot(residuals, kind='kde')
```

Out[49]:
```
<seaborn.axisgrid.FacetGrid at 0x1e1fb3d42f0>
```



Model Performance

In [68]:
```
## lower error value - MSE AND MAE
## higher value of r2 score and adjested r2 score
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
print(mean_squared_error(y_test, reg_pred))
print(mean_absolute_error,(y_test, reg_pred))
print(r2_score(y_test, reg_pred))
print(np.sqrt(mean_squared_error(y_test, reg_pred)))
```

```
0.5305677824766752
<function mean_absolute_error at 0x000001E1F82FC040> (20046    0.47700
3024     0.45800
15663    5.00001
20484    2.18600
9814     2.78000
          ...
17505    2.37500
13512    0.67300
10842    2.18400
16559    1.19400
5786     2.09800
Name: Price, Length: 6192, dtype: float64, array([0.72604907, 1.76743383, 2.7109216
1, ..., 2.07465531, 1.57371395,
       1.82744133]))
0.5957702326061665
0.7284008391515452
```

In [59]:
```python
score =print(r2_score, y_test, reg_pred)
```

```
<function r2_score at 0x000001E1F82FCC20> 20046    0.47700
3024     0.45800
15663    5.00001
20484    2.18600
9814     2.78000
          ...
17505    2.37500
13512    0.67300
10842    2.18400
16559    1.19400
5786     2.09800
Name: Price, Length: 6192, dtype: float64 [0.72604907 1.76743383 2.71092161 ... 2.07
465531 1.57371395 1.82744133]
```

In [61]:
```python
score = r2_score(y_test, reg_pred)
```

In [62]:
```python
score
```

Out[62]: 0.5957702326061665

In [53]:
```python
x_test_norm.shape
```

Out[53]: (6192, 8)

In [64]:
```python
1 -(1-score)*(len(y_test)-1/(len(y_test)-x_test_norm.shape[1]-1))
```

Out[64]: -2501.9906543250063

In [67]:
```python
x_test_norm.shape[1]
```

Out[67]: 8

Save the Model -> pickle file

In [69]:
```
import pickle
pickle.dump(regression, open('model.pkl', 'wb'))
```

load the file and use it for future test data prediction

In [74]:
```
model = pickle.load(open('model.pkl', 'rb'))
```

In [75]:
```
model
```

Out[75]:
```
▾ LinearRegression
LinearRegression()
```

In [77]:
```
housing.data[0]
```

Out[77]:
```
array([   8.3252    ,   41.        ,    6.98412698,    1.02380952,
        322.        ,    2.55555556,   37.88      , -122.23      ])
```

In [89]:
```
model.predict(scaler.transform(housing.data[0]).reshape(1, -1)))
```

```
  Cell In[89], line 1
    model.predict(scaler.transform(housing.data[0]).reshape(1, -1)))
                                                                   ^
SyntaxError: unmatched ')'
```

In [90]:
```
model.predict(x_test_norm)
```

Out[90]:
```
array([0.72604907, 1.76743383, 2.71092161, ..., 2.07465531, 1.57371395,
       1.82744133])
```

In [ ]: