House Price Prediction Using Machine Learning

September 27, 2024

1 House Price Prediction Using Machine Learning:

2 Name Lakshman Chaudhary

[13]:
```python
#Importing the required libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import mean_absolute_error, r2_score
from sklearn.decomposition import PCA
import matplotlib.dates as mdates
from sklearn.model_selection import train_test_split
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
import os
from statsmodels.tsa.arima.model import ARIMA
import plotly.graph_objs as go
import seaborn as sns
import sys
```

[14]:
```python
print("System Config ::\nPython ::",sys.version)
```

```
System Config ::
Python :: 3.11.7 | packaged by Anaconda, Inc. | (main, Dec 15 2023, 18:05:47)
[MSC v.1916 64 bit (AMD64)]
```

[15]:
```
# Reading the dataset
# Get the current working directory (cwd)
cwd = os.getcwd()
```

[16]:
```
# Get all the files in that directory
files = os.listdir(cwd)
print("Files in %r: %s" % (cwd, files))
```

```
Files in 'C:\\Users\\laksh\\Downloads\\MY Jupyter Notebook Notes\\House Price
Prediction using Machine Learning': ['.ipynb_checkpoints', 'House Price
Prediction using Machine Learning.ipynb', 'House_Price.csv', 'New',
'xgboost_model.pkl', 'xgb_model.pkl']
```

[17]:
```
# Reading the dataset
# Get the current working directory (cwd)
cwd = os.getcwd()
# Get all the files in that directory
files = os.listdir(cwd)
print("Files in %r: %s" % (cwd, files))
```

```
Files in 'C:\\Users\\laksh\\Downloads\\MY Jupyter Notebook Notes\\House Price
Prediction using Machine Learning': ['.ipynb_checkpoints', 'House Price
Prediction using Machine Learning.ipynb', 'House_Price.csv', 'New',
'xgboost_model.pkl', 'xgb_model.pkl']
```

[33]:
```
# import data
# Load the House Price dataset
data = pd.read_csv('House_Price.csv')
data
```

[33]:
```
     price  crime_rate  resid_area  air_qual  room_num   age  dist1  dist2  \
0     24.0     0.00632       32.31     0.538     6.575  65.2   4.35   3.81
1     21.6     0.02731       37.07     0.469     6.421  78.9   4.99   4.70
2     34.7     0.02729       37.07     0.469     7.185  61.1   5.03   4.86
3     33.4     0.03237       32.18     0.458     6.998  45.8   6.21   5.93
4     36.2     0.06905       32.18     0.458     7.147  54.2   6.16   5.86
..     …           …           …         …         …     …      …      …
501   22.4     0.06263       41.93     0.573     6.593  69.1   2.64   2.45
502   20.6     0.04527       41.93     0.573     6.120  76.7   2.44   2.11
503   23.9     0.06076       41.93     0.573     6.976  91.0   2.34   2.06
504   22.0     0.10959       41.93     0.573     6.794  89.3   2.54   2.31
505   19.0     0.04741       41.93     0.573     6.030  80.8   2.72   2.24

     dist3  dist4  teachers  poor_prop airport  n_hos_beds  n_hot_rooms  \
0     4.18   4.01      24.7       4.98     YES       5.480      11.1920
1     5.12   5.06      22.2       9.14      NO       7.332      12.1728
2     5.01   4.97      22.2       4.03      NO       7.394     101.1200
3     6.16   5.96      21.3       2.94     YES       9.268      11.2672
```

```
4     6.37   5.86      21.3       5.33      NO       8.824      11.2896
..     …     …        …        …      …       …         …
501   2.76   2.06      19.0       9.67      NO       9.348      12.1792
502   2.46   2.14      19.0       9.08     YES       6.612      13.1648
503   2.29   1.98      19.0       5.64      NO       5.478      12.1912
504   2.40   2.31      19.0       6.48     YES       7.940      15.1760
505   2.64   2.42      19.0       7.88     YES      10.280      10.1520

          waterbody  rainfall bus_ter     parks
0             River        23     YES  0.049347
1              Lake        42     YES  0.046146
2               NaN        38     YES  0.045764
3              Lake        45     YES  0.047151
4              Lake        55     YES  0.039474
..              …        …      …        …
501  Lake and River        27     YES  0.056006
502  Lake and River        20     YES  0.059903
503             NaN        31     YES  0.057572
504             NaN        47     YES  0.060694
505             NaN        45     YES  0.060336

[506 rows x 19 columns]
```

[94]:
```
# Data Info
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 19 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   price        506 non-null    float64
 1   crime_rate   506 non-null    float64
 2   resid_area   506 non-null    float64
 3   air_qual     506 non-null    float64
 4   room_num     506 non-null    float64
 5   age          506 non-null    float64
 6   dist1        506 non-null    float64
 7   dist2        506 non-null    float64
 8   dist3        506 non-null    float64
 9   dist4        506 non-null    float64
 10  teachers     506 non-null    float64
 11  poor_prop    506 non-null    float64
 12  airport      506 non-null    object
 13  n_hos_beds   498 non-null    float64
 14  n_hot_rooms  506 non-null    float64
 15  waterbody    351 non-null    object
 16  rainfall     506 non-null    int64
```

```
 17   bus_ter       506 non-null    object
 18   parks         506 non-null    float64
dtypes: float64(15), int64(1), object(3)
memory usage: 75.2+ KB
```

[95]:
```
# Coloumn names of the dataset
data.columns
```

[95]:
```
Index(['price', 'crime_rate', 'resid_area', 'air_qual', 'room_num', 'age',
       'dist1', 'dist2', 'dist3', 'dist4', 'teachers', 'poor_prop', 'airport',
       'n_hos_beds', 'n_hot_rooms', 'waterbody', 'rainfall', 'bus_ter',
       'parks'],
      dtype='object')
```

2.1 Data Coloumns:

[64]:
```
# Load the House Price dataset
data = pd.read_csv('House_Price.csv')
data
```

[64]:
```
     price  crime_rate  resid_area  air_qual  room_num   age  dist1  dist2  \
0     24.0     0.00632       32.31     0.538     6.575  65.2   4.35   3.81
1     21.6     0.02731       37.07     0.469     6.421  78.9   4.99   4.70
2     34.7     0.02729       37.07     0.469     7.185  61.1   5.03   4.86
3     33.4     0.03237       32.18     0.458     6.998  45.8   6.21   5.93
4     36.2     0.06905       32.18     0.458     7.147  54.2   6.16   5.86
..     …         …           …         …        …      …     …      …
501   22.4     0.06263       41.93     0.573     6.593  69.1   2.64   2.45
502   20.6     0.04527       41.93     0.573     6.120  76.7   2.44   2.11
503   23.9     0.06076       41.93     0.573     6.976  91.0   2.34   2.06
504   22.0     0.10959       41.93     0.573     6.794  89.3   2.54   2.31
505   19.0     0.04741       41.93     0.573     6.030  80.8   2.72   2.24

     dist3  dist4  teachers  poor_prop airport  n_hos_beds  n_hot_rooms  \
0     4.18   4.01      24.7       4.98     YES       5.480      11.1920
1     5.12   5.06      22.2       9.14      NO       7.332      12.1728
2     5.01   4.97      22.2       4.03      NO       7.394     101.1200
3     6.16   5.96      21.3       2.94     YES       9.268      11.2672
4     6.37   5.86      21.3       5.33      NO       8.824      11.2896
..     …      …         …          …        …          …            …
501   2.76   2.06      19.0       9.67      NO       9.348      12.1792
502   2.46   2.14      19.0       9.08     YES       6.612      13.1648
503   2.29   1.98      19.0       5.64      NO       5.478      12.1912
504   2.40   2.31      19.0       6.48     YES       7.940      15.1760
505   2.64   2.42      19.0       7.88     YES      10.280      10.1520

          waterbody  rainfall bus_ter     parks
0             River        23     YES  0.049347
```

```
1                 Lake        42      YES  0.046144
2                  NaN        38      YES  0.045764
3                 Lake        45      YES  0.047151
4                 Lake        55      YES  0.039474
..                 ...       ...      ...       ...
501     Lake and River        27      YES  0.056006
502     Lake and River        20      YES  0.059903
503                NaN        31      YES  0.057572
504                NaN        47      YES  0.060694
505                NaN        45      YES  0.060336

[506 rows x 19 columns]
```

[65]:

```python
# Coloumn names of the dataset
data.columns
```

[65]:

```
Index(['price', 'crime_rate', 'resid_area', 'air_qual', 'room_num', 'age',
       'dist1', 'dist2', 'dist3', 'dist4', 'teachers', 'poor_prop', 'airport',
       'n_hos_beds', 'n_hot_rooms', 'waterbody', 'rainfall', 'bus_ter',
       'parks'],
      dtype='object')
```

[70]:

```python
# drop the unwanted single coloum
data = data.drop('dist2',axis = 1)
data
```

[70]:

```
     price  crime_rate  air_qual  room_num   age  dist1  dist3  dist4  \
0     24.0     0.00632     0.538     6.575  65.2   4.35   4.18   4.01
1     21.6     0.02731     0.469     6.421  78.9   4.99   5.12   5.06
2     34.7     0.02729     0.469     7.185  61.1   5.03   5.01   4.97
3     33.4     0.03237     0.458     6.998  45.8   6.21   6.16   5.96
4     36.2     0.06905     0.458     7.147  54.2   6.16   6.37   5.86
..     ...         ...       ...       ...   ...    ...    ...    ...
501   22.4     0.06263     0.573     6.593  69.1   2.64   2.76   2.06
502   20.6     0.04527     0.573     6.120  76.7   2.44   2.46   2.14
503   23.9     0.06076     0.573     6.976  91.0   2.34   2.29   1.98
504   22.0     0.10959     0.573     6.794  89.3   2.54   2.40   2.31
505   19.0     0.04741     0.573     6.030  80.8   2.72   2.64   2.42

     teachers  poor_prop airport  n_hos_beds  n_hot_rooms       waterbody  \
0        24.7       4.98     YES       5.480      11.1920           River
1        22.2       9.14      NO       7.332      12.1728            Lake
2        22.2       4.03      NO       7.394     101.1200             NaN
3        21.3       2.94     YES       9.268      11.2672            Lake
4        21.3       5.33      NO       8.824      11.2896            Lake
..        ...        ...     ...         ...          ...             ...
501      19.0       9.67      NO       9.348      12.1792  Lake and River
```

```
502        19.0        9.08       YES        6.612        13.1648  Lake and River
503        19.0        5.64        NO        5.478        12.1912             NaN
504        19.0        6.48       YES        7.940        15.1760             NaN
505        19.0        7.88       YES       10.280        10.1520             NaN

     rainfall bus_ter     parks
0          23     YES  0.049347
1          42     YES  0.046146
2          38     YES  0.045764
3          45     YES  0.047151
4          55     YES  0.039474
..        ...     ...       ...
501        27     YES  0.056006
502        20     YES  0.059903
503        31     YES  0.057572
504        47     YES  0.060694
505        45     YES  0.060336

[506 rows x 17 columns]
```

[71]:
```
# drop the unwanted single coloum
data = pd.read_csv('House_Price.csv')
data
data = data.drop('resid_area',axis = 1)
data
```

[71]:
```
     price  crime_rate  air_qual  room_num   age  dist1  dist2  dist3  dist4  \
0     24.0     0.00632     0.538     6.575  65.2   4.35   3.81   4.18   4.01
1     21.6     0.02731     0.469     6.421  78.9   4.99   4.70   5.12   5.06
2     34.7     0.02729     0.469     7.185  61.1   5.03   4.86   5.01   4.97
3     33.4     0.03237     0.458     6.998  45.8   6.21   5.93   6.16   5.96
4     36.2     0.06905     0.458     7.147  54.2   6.16   5.86   6.37   5.86
..     ...         ...       ...       ...   ...    ...    ...    ...    ...
501   22.4     0.06263     0.573     6.593  69.1   2.64   2.45   2.76   2.06
502   20.6     0.04527     0.573     6.120  76.7   2.44   2.11   2.46   2.14
503   23.9     0.06076     0.573     6.976  91.0   2.34   2.06   2.29   1.98
504   22.0     0.10959     0.573     6.794  89.3   2.54   2.31   2.40   2.31
505   19.0     0.04741     0.573     6.030  80.8   2.72   2.24   2.64   2.42

     teachers  poor_prop airport  n_hos_beds  n_hot_rooms       waterbody  \
0        24.7       4.98     YES       5.480      11.1920           River
1        22.2       9.14      NO       7.332      12.1728            Lake
2        22.2       4.03      NO       7.394     101.1200             NaN
3        21.3       2.94     YES       9.268      11.2672            Lake
4        21.3       5.33      NO       8.824      11.2896            Lake
..        ...        ...     ...         ...          ...             ...
501      19.0       9.67      NO       9.348      12.1792  Lake and River
```

```
502      19.0       9.08     YES       6.612      13.1648  Lake and River
503      19.0       5.64      NO       5.478      12.1912             NaN
504      19.0       6.48     YES       7.940      15.1760             NaN
505      19.0       7.88     YES      10.280      10.1520             NaN

     rainfall bus_ter     parks
0          23     YES  0.049347
1          42     YES  0.046146
2          38     YES  0.045764
3          45     YES  0.047151
4          55     YES  0.039474
..        …      …        …
501        27     YES  0.056006
502        20     YES  0.059903
503        31     YES  0.057572
504        47     YES  0.060694
505        45     YES  0.060336

[506 rows x 18 columns]
```

[72]:

```python
# drop the unwanted single coloum
data = pd.read_csv('House_Price.csv')
data
data = data.drop('air_qual',axis = 1)
data
```

[72]:

```
     price  crime_rate  resid_area  room_num   age  dist1  dist2  dist3  \
0     24.0     0.00632       32.31     6.575  65.2   4.35   3.81   4.18
1     21.6     0.02731       37.07     6.421  78.9   4.99   4.70   5.12
2     34.7     0.02729       37.07     7.185  61.1   5.03   4.86   5.01
3     33.4     0.03237       32.18     6.998  45.8   6.21   5.93   6.16
4     36.2     0.06905       32.18     7.147  54.2   6.16   5.86   6.37
..     …          …           …        …       …    …      …      …
501   22.4     0.06263       41.93     6.593  69.1   2.64   2.45   2.76
502   20.6     0.04527       41.93     6.120  76.7   2.44   2.11   2.46
503   23.9     0.06076       41.93     6.976  91.0   2.34   2.06   2.29
504   22.0     0.10959       41.93     6.794  89.3   2.54   2.31   2.40
505   19.0     0.04741       41.93     6.030  80.8   2.72   2.24   2.64

     dist4  teachers  poor_prop airport  n_hos_beds  n_hot_rooms  \
0     4.01      24.7       4.98     YES       5.480      11.1920
1     5.06      22.2       9.14      NO       7.332      12.1728
2     4.97      22.2       4.03      NO       7.394     101.1200
3     5.96      21.3       2.94     YES       9.268      11.2672
4     5.86      21.3       5.33      NO       8.824      11.2896
..     …         …          …       …          …            …
501   2.06      19.0       9.67      NO       9.348      12.1792
```

```
502   2.14      19.0      9.08    YES      6.612      13.1648
503   1.98      19.0      5.64     NO      5.478      12.1912
504   2.31      19.0      6.48    YES      7.940      15.1760
505   2.42      19.0      7.88    YES     10.280      10.1520

          waterbody  rainfall bus_ter     parks
0             River        23     YES  0.049347
1              Lake        42     YES  0.046146
2               NaN        38     YES  0.045764
3              Lake        45     YES  0.047151
4              Lake        55     YES  0.039474
..              …         …      …        …
501  Lake and River        27     YES  0.056006
502  Lake and River        20     YES  0.059903
503             NaN        31     YES  0.057572
504             NaN        47     YES  0.060694
505             NaN        45     YES  0.060336

[506 rows x 18 columns]
```

[73]:

```python
# drop the unwanted Multiple coloum

'''
# drop the unwanted Multiple coloum

# To drop multiple columns in pandas DataFrame Here are different Method Below:
# Method 1: Using a list of column names

data = pd.read_csv('House_Price.csv')
data
columns_to_drop = ['age', 'dist2', 'dist3', 'dist4']
data = data.drop(columns_to_drop, axis=1)
data


# Method 2: Using the drop method with multiple arguments

data = pd.read_csv('House_Price.csv')
data
data = data.drop(['age', 'dist2', 'dist3', 'dist4'], axis=1)
data
'''
# Method 3: Using conditional dropping

data = pd.read_csv('House_Price.csv')
data
data = data.drop([col for col in data.columns if col.startswith('a')], axis=1)
```

```python
data
```

[73]:

```
     price  crime_rate  resid_area  room_num  dist1  dist2  dist3  dist4  \
0     24.0     0.00632       32.31     6.575   4.35   3.81   4.18   4.01
1     21.6     0.02731       37.07     6.421   4.99   4.70   5.12   5.06
2     34.7     0.02729       37.07     7.185   5.03   4.86   5.01   4.97
3     33.4     0.03237       32.18     6.998   6.21   5.93   6.16   5.96
4     36.2     0.06905       32.18     7.147   6.16   5.86   6.37   5.86
..     …          …           …          …     …      …      …      …
501   22.4     0.06263       41.93     6.593   2.64   2.45   2.76   2.06
502   20.6     0.04527       41.93     6.120   2.44   2.11   2.46   2.14
503   23.9     0.06076       41.93     6.976   2.34   2.06   2.29   1.98
504   22.0     0.10959       41.93     6.794   2.54   2.31   2.40   2.31
505   19.0     0.04741       41.93     6.030   2.72   2.24   2.64   2.42

     teachers  poor_prop  n_hos_beds  n_hot_rooms       waterbody  rainfall  \
0        24.7       4.98       5.480      11.1920           River        23
1        22.2       9.14       7.332      12.1728            Lake        42
2        22.2       4.03       7.394     101.1200             NaN        38
3        21.3       2.94       9.268      11.2672            Lake        45
4        21.3       5.33       8.824      11.2896            Lake        55
..        …          …          …            …               …         …
501      19.0       9.67       9.348      12.1792  Lake and River        27
502      19.0       9.08       6.612      13.1648  Lake and River        20
503      19.0       5.64       5.478      12.1912             NaN        31
504      19.0       6.48       7.940      15.1760             NaN        47
505      19.0       7.88      10.280      10.1520             NaN        45

    bus_ter     parks
0       YES  0.049347
1       YES  0.046146
2       YES  0.045764
3       YES  0.047151
4       YES  0.039474
..      …         …
501     YES  0.056006
502     YES  0.059903
503     YES  0.057572
504     YES  0.060694
505     YES  0.060336

[506 rows x 16 columns]
```

[74]:

```python
# drop the unwanted Multiple coloum

'''
# To drop multiple columns in pandas DataFrame Here are different Method Below:
```

```
# Method 1: Using a list of column names

data = pd.read_csv('House_Price.csv')
data
columns_to_drop = ['age', 'dist2', 'dist3', 'dist4']
data = data.drop(columns_to_drop, axis=1)
data


# Method 2: Using the drop method with multiple arguments

data = pd.read_csv('House_Price.csv')
data
data = data.drop(['age', 'dist2', 'dist3', 'dist4'], axis=1)
data
'''
# Method 3: Using conditional dropping

data = pd.read_csv('House_Price.csv')
data
data = data.drop([col for col in data.columns if col.startswith('dis')], axis=1)
data
```

[74]:

```
     price  crime_rate  resid_area  air_qual  room_num   age  teachers  \
0     24.0     0.00632       32.31     0.538     6.575  65.2      24.7
1     21.6     0.02731       37.07     0.469     6.421  78.9      22.2
2     34.7     0.02729       37.07     0.469     7.185  61.1      22.2
3     33.4     0.03237       32.18     0.458     6.998  45.8      21.3
4     36.2     0.06905       32.18     0.458     7.147  54.2      21.3
..     …          …          …         …        …       …     …
501   22.4     0.06263       41.93     0.573     6.593  69.1      19.0
502   20.6     0.04527       41.93     0.573     6.120  76.7      19.0
503   23.9     0.06076       41.93     0.573     6.976  91.0      19.0
504   22.0     0.10959       41.93     0.573     6.794  89.3      19.0
505   19.0     0.04741       41.93     0.573     6.030  80.8      19.0

     poor_prop airport  n_hos_beds  n_hot_rooms       waterbody  rainfall  \
0         4.98     YES       5.480      11.1920           River        23
1         9.14      NO       7.332      12.1728            Lake        42
2         4.03      NO       7.394     101.1200             NaN        38
3         2.94     YES       9.268      11.2672            Lake        45
4         5.33      NO       8.824      11.2896            Lake        55
..         …       …          …          …           …        …
501       9.67      NO       9.348      12.1792  Lake and River        27
502       9.08     YES       6.612      13.1648  Lake and River        20
503       5.64      NO       5.478      12.1912             NaN        31
504       6.48     YES       7.940      15.1760             NaN        47
```

```
505        7.88      YES       10.280       10.1520              NaN         45

    bus_ter     parks
0       YES  0.049347
1       YES  0.046146
2       YES  0.045764
3       YES  0.047151
4       YES  0.039474
..      …         …
501     YES  0.056006
502     YES  0.059903
503     YES  0.057572
504     YES  0.060694
505     YES  0.060336

[506 rows x 15 columns]
```

[75]:

```
# drop the unwanted Multiple coloum

'''
# To drop multiple columns in pandas DataFrame Here are different Method Below:
# Method 1: Using a list of column names

data = pd.read_csv('House_Price.csv')
data
columns_to_drop = ['age', 'dist2', 'dist3', 'dist4']
data = data.drop(columns_to_drop, axis=1)
data



# Method 2: Using the drop method with multiple arguments

data = pd.read_csv('House_Price.csv')
data
data = data.drop(['age', 'dist2', 'dist3', 'dist4'], axis=1)
data
'''
# Method 3: Using conditional dropping

data = pd.read_csv('House_Price.csv')
data
data = data.drop([col for col in data.columns if col.startswith('n')], axis=1)
data
```

[75]:

```
   price  crime_rate  resid_area  air_qual  room_num   age  dist1  dist2  \
0   24.0     0.00632       32.31     0.538     6.575  65.2   4.35   3.81
1   21.6     0.02731       37.07     0.469     6.421  78.9   4.99   4.70
```

```
2    34.7  0.02729    37.07  0.469   7.185  61.1  5.03  4.86
3    33.4  0.03237    32.18  0.458   6.998  45.8  6.21  5.93
4    36.2  0.06905    32.18  0.458   7.147  54.2  6.16  5.86
..    …      …      …    …      …    …    …    …
501  22.4  0.06263    41.93  0.573   6.593  69.1  2.64  2.45
502  20.6  0.04527    41.93  0.573   6.120  76.7  2.44  2.11
503  23.9  0.06076    41.93  0.573   6.976  91.0  2.34  2.06
504  22.0  0.10959    41.93  0.573   6.794  89.3  2.54  2.31
505  19.0  0.04741    41.93  0.573   6.030  80.8  2.72  2.24

     dist3  dist4  teachers  poor_prop airport       waterbody  rainfall  \
0     4.18   4.01      24.7       4.98     YES           River        23
1     5.12   5.06      22.2       9.14      NO            Lake        42
2     5.01   4.97      22.2       4.03      NO             NaN        38
3     6.16   5.96      21.3       2.94     YES            Lake        45
4     6.37   5.86      21.3       5.33      NO            Lake        55
..     …      …       …        …      …              …       …
501   2.76   2.06      19.0       9.67      NO  Lake and River        27
502   2.46   2.14      19.0       9.08     YES  Lake and River        20
503   2.29   1.98      19.0       5.64      NO             NaN        31
504   2.40   2.31      19.0       6.48     YES             NaN        47
505   2.64   2.42      19.0       7.88     YES             NaN        45

    bus_ter     parks
0       YES  0.049347
1       YES  0.046146
2       YES  0.045764
3       YES  0.047151
4       YES  0.039474
..      …       …
501     YES  0.056006
502     YES  0.059903
503     YES  0.057572
504     YES  0.060694
505     YES  0.060336

[506 rows x 17 columns]
```

[76]:
```python
# drop the unwanted Multiple coloum

# Method 2: Using the drop method with multiple arguments

data = pd.read_csv('House_Price.csv')
data
data = data.drop(['age', 'dist2', 'dist3', 'dist4'], axis=1)
data
```

[76]:
```
     price  crime_rate  resid_area  air_qual  room_num  dist1  teachers  \
0     24.0     0.00632       32.31     0.538     6.575   4.35      24.7
1     21.6     0.02731       37.07     0.469     6.421   4.99      22.2
2     34.7     0.02729       37.07     0.469     7.185   5.03      22.2
3     33.4     0.03237       32.18     0.458     6.998   6.21      21.3
4     36.2     0.06905       32.18     0.458     7.147   6.16      21.3
..     …         …           …         …        …       …       …
501   22.4     0.06263       41.93     0.573     6.593   2.64      19.0
502   20.6     0.04527       41.93     0.573     6.120   2.44      19.0
503   23.9     0.06076       41.93     0.573     6.976   2.34      19.0
504   22.0     0.10959       41.93     0.573     6.794   2.54      19.0
505   19.0     0.04741       41.93     0.573     6.030   2.72      19.0

     poor_prop airport  n_hos_beds  n_hot_rooms       waterbody  rainfall  \
0         4.98     YES       5.480      11.1920           River        23
1         9.14      NO       7.332      12.1728            Lake        42
2         4.03      NO       7.394     101.1200             NaN        38
3         2.94     YES       9.268      11.2672            Lake        45
4         5.33      NO       8.824      11.2896            Lake        55
..         …       …          …           …               …          …
501       9.67      NO       9.348      12.1792  Lake and River        27
502       9.08     YES       6.612      13.1648  Lake and River        20
503       5.64      NO       5.478      12.1912             NaN        31
504       6.48     YES       7.940      15.1760             NaN        47
505       7.88     YES      10.280      10.1520             NaN        45

    bus_ter     parks
0       YES  0.049347
1       YES  0.046146
2       YES  0.045764
3       YES  0.047151
4       YES  0.039474
..      …        …
501     YES  0.056006
502     YES  0.059903
503     YES  0.057572
504     YES  0.060694
505     YES  0.060336

[506 rows x 15 columns]
```

[77]:
```python
# Verify column existence
print(data.columns)

# Drop multiple columns
columns_to_drop = ['dist4', 'dist2', 'dist1']
data = data.drop(columns_to_drop, axis=1, errors='ignore')
```

```python
# Reprint column names
print(data.columns)
```

```
Index(['price', 'crime_rate', 'resid_area', 'air_qual', 'room_num', 'dist1',
       'teachers', 'poor_prop', 'airport', 'n_hos_beds', 'n_hot_rooms',
       'waterbody', 'rainfall', 'bus_ter', 'parks'],
      dtype='object')
Index(['price', 'crime_rate', 'resid_area', 'air_qual', 'room_num', 'teachers',
       'poor_prop', 'airport', 'n_hos_beds', 'n_hot_rooms', 'waterbody',
       'rainfall', 'bus_ter', 'parks'],
      dtype='object')
```

[78]:
```python
# Load the House Price dataset
data = pd.read_csv('House_Price.csv')
data
```

[78]:
```
     price  crime_rate  resid_area  air_qual  room_num   age  dist1  dist2  \
0     24.0     0.00632       32.31     0.538     6.575  65.2   4.35   3.81
1     21.6     0.02731       37.07     0.469     6.421  78.9   4.99   4.70
2     34.7     0.02729       37.07     0.469     7.185  61.1   5.03   4.86
3     33.4     0.03237       32.18     0.458     6.998  45.8   6.21   5.93
4     36.2     0.06905       32.18     0.458     7.147  54.2   6.16   5.86
..     …          …          …         …         …     …      …      …
501   22.4     0.06263       41.93     0.573     6.593  69.1   2.64   2.45
502   20.6     0.04527       41.93     0.573     6.120  76.7   2.44   2.11
503   23.9     0.06076       41.93     0.573     6.976  91.0   2.34   2.06
504   22.0     0.10959       41.93     0.573     6.794  89.3   2.54   2.31
505   19.0     0.04741       41.93     0.573     6.030  80.8   2.72   2.24

     dist3  dist4  teachers  poor_prop airport  n_hos_beds  n_hot_rooms  \
0     4.18   4.01      24.7       4.98     YES       5.480      11.1920
1     5.12   5.06      22.2       9.14      NO       7.332      12.1728
2     5.01   4.97      22.2       4.03      NO       7.394     101.1200
3     6.16   5.96      21.3       2.94     YES       9.268      11.2672
4     6.37   5.86      21.3       5.33      NO       8.824      11.2896
..     …      …         …          …       …          …            …
501   2.76   2.06      19.0       9.67      NO       9.348      12.1792
502   2.46   2.14      19.0       9.08     YES       6.612      13.1648
503   2.29   1.98      19.0       5.64      NO       5.478      12.1912
504   2.40   2.31      19.0       6.48     YES       7.940      15.1760
505   2.64   2.42      19.0       7.88     YES      10.280      10.1520

          waterbody  rainfall bus_ter     parks
0             River        23     YES  0.049347
1              Lake        42     YES  0.046146
2               NaN        38     YES  0.045764
3              Lake        45     YES  0.047151
```

```
4             Lake        55     YES  0.039474
..               …       …     …        …
501  Lake and River        27     YES  0.056006
502  Lake and River        20     YES  0.059903
503             NaN        31     YES  0.057572
504             NaN        47     YES  0.060694
505             NaN        45     YES  0.060336

[506 rows x 19 columns]
```

[85]:

```python
# Verify column existence
print(data.columns)

# Drop multiple columns
columns_to_drop = ['dist4', 'dist2', 'dist1']
data = data.drop(columns_to_drop, axis=1, errors='ignore')

# Reprint column names
print(data.columns)
```

```
Index(['price', 'crime_rate', 'resid_area', 'air_qual', 'room_num', 'age',
       'dist1', 'dist2', 'dist3', 'dist4', 'teachers', 'poor_prop', 'airport',
       'n_hos_beds', 'n_hot_rooms', 'waterbody', 'rainfall', 'bus_ter',
       'parks'],
      dtype='object')
Index(['price', 'crime_rate', 'resid_area', 'air_qual', 'room_num', 'age',
       'dist3', 'teachers', 'poor_prop', 'airport', 'n_hos_beds',
       'n_hot_rooms', 'waterbody', 'rainfall', 'bus_ter', 'parks'],
      dtype='object')
```

[86]:

```python
print(data.columns)
```

```
Index(['price', 'crime_rate', 'resid_area', 'air_qual', 'room_num', 'age',
       'dist3', 'teachers', 'poor_prop', 'airport', 'n_hos_beds',
       'n_hot_rooms', 'waterbody', 'rainfall', 'bus_ter', 'parks'],
      dtype='object')
```

[87]:

```python
data.describe()
```

[87]:

	price	crime_rate	resid_area	air_qual	room_num	age	\
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	
mean	22.528854	3.613524	41.136779	0.554695	6.284634	68.574901	
std	9.182176	8.601545	6.860353	0.115878	0.702617	28.148861	
min	5.000000	0.006320	30.460000	0.385000	3.561000	2.900000	
25%	17.025000	0.082045	35.190000	0.449000	5.885500	45.025000	
50%	21.200000	0.256510	39.690000	0.538000	6.208500	77.500000	
75%	25.000000	3.677083	48.100000	0.624000	6.623500	94.075000	
max	50.000000	88.976200	57.740000	0.871000	8.780000	100.000000	

```
          dist3    teachers   poor_prop  n_hos_beds  n_hot_rooms  \
count  506.000000  506.000000  506.000000  498.000000   506.000000
mean     3.960672   21.544466   12.653063    7.899767    13.041605
std      2.119797    2.164946    7.141062    1.476683     5.238957
min      1.150000   18.000000    1.730000    5.268000    10.057600
25%      2.232500   19.800000    6.950000    6.634500    11.189800
50%      3.375000   20.950000   11.360000    7.999000    12.720000
75%      5.407500   22.600000   16.955000    9.088000    14.170800
max     12.320000   27.400000   37.970000   10.876000   101.120000

         rainfall       parks
count  506.000000  506.000000
mean    39.181818    0.054454
std     12.513697    0.010632
min      3.000000    0.033292
25%     28.000000    0.046464
50%     39.000000    0.053507
75%     50.000000    0.061397
max     60.000000    0.086711
```

[88]:
```python
# Load the House Price dataset
data = pd.read_csv('House_Price.csv')
data
```

[88]:
```
     price  crime_rate  resid_area  air_qual  room_num   age  dist1  dist2  \
0     24.0     0.00632       32.31     0.538     6.575  65.2   4.35   3.81
1     21.6     0.02731       37.07     0.469     6.421  78.9   4.99   4.70
2     34.7     0.02729       37.07     0.469     7.185  61.1   5.03   4.86
3     33.4     0.03237       32.18     0.458     6.998  45.8   6.21   5.93
4     36.2     0.06905       32.18     0.458     7.147  54.2   6.16   5.86
..     …          …          …        …        …    …     …     …
501   22.4     0.06263       41.93     0.573     6.593  69.1   2.64   2.45
502   20.6     0.04527       41.93     0.573     6.120  76.7   2.44   2.11
503   23.9     0.06076       41.93     0.573     6.976  91.0   2.34   2.06
504   22.0     0.10959       41.93     0.573     6.794  89.3   2.54   2.31
505   19.0     0.04741       41.93     0.573     6.030  80.8   2.72   2.24

     dist3  dist4  teachers  poor_prop airport  n_hos_beds  n_hot_rooms  \
0     4.18   4.01      24.7       4.98     YES       5.480      11.1920
1     5.12   5.06      22.2       9.14      NO       7.332      12.1728
2     5.01   4.97      22.2       4.03      NO       7.394     101.1200
3     6.16   5.96      21.3       2.94     YES       9.268      11.2672
4     6.37   5.86      21.3       5.33      NO       8.824      11.2896
..     …      …        …         …       …         …            …
501   2.76   2.06      19.0       9.67      NO       9.348      12.1792
502   2.46   2.14      19.0       9.08     YES       6.612      13.1648
```

```
503   2.29   1.98      19.0       5.64      NO       5.478       12.1912
504   2.40   2.31      19.0       6.48     YES       7.940       15.1760
505   2.64   2.42      19.0       7.88     YES      10.280       10.1520

          waterbody  rainfall bus_ter     parks
0             River        23     YES  0.049347
1              Lake        42     YES  0.046146
2               NaN        38     YES  0.045764
3              Lake        45     YES  0.047151
4              Lake        55     YES  0.039474
..              ...       ...     ...       ...
501  Lake and River        27     YES  0.056006
502  Lake and River        20     YES  0.059903
503             NaN        31     YES  0.057572
504             NaN        47     YES  0.060694
505             NaN        45     YES  0.060336

[506 rows x 19 columns]
```

[89]:
```
print(data.columns)
```

```
Index(['price', 'crime_rate', 'resid_area', 'air_qual', 'room_num', 'age',
       'dist1', 'dist2', 'dist3', 'dist4', 'teachers', 'poor_prop', 'airport',
       'n_hos_beds', 'n_hot_rooms', 'waterbody', 'rainfall', 'bus_ter',
       'parks'],
      dtype='object')
```

[90]:
```
data.describe()
```

[90]:
```
            price  crime_rate  resid_area    air_qual    room_num         age  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    22.528854    3.613524   41.136779    0.554695    6.284634   68.574901
std      9.182176    8.601545    6.860353    0.115878    0.702617   28.148861
min      5.000000    0.006320   30.460000    0.385000    3.561000    2.900000
25%     17.025000    0.082045   35.190000    0.449000    5.885500   45.025000
50%     21.200000    0.256510   39.690000    0.538000    6.208500   77.500000
75%     25.000000    3.677083   48.100000    0.624000    6.623500   94.075000
max     50.000000   88.976200   57.740000    0.871000    8.780000  100.000000

            dist1       dist2       dist3       dist4    teachers   poor_prop  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean     3.971996    3.628775    3.960672    3.618972   21.544466   12.653063
std      2.108532    2.108580    2.119797    2.099203    2.164946    7.141062
min      1.130000    0.920000    1.150000    0.730000   18.000000    1.730000
25%      2.270000    1.940000    2.232500    1.940000   19.800000    6.950000
50%      3.385000    3.010000    3.375000    3.070000   20.950000   11.360000
75%      5.367500    4.992500    5.407500    4.985000   22.600000   16.955000
max     12.320000   11.930000   12.320000   11.940000   27.400000   37.970000
```

	n_hos_beds	n_hot_rooms	rainfall	parks
count	498.000000	506.000000	506.000000	506.000000
mean	7.899767	13.041605	39.181818	0.054454
std	1.476683	5.238957	12.513697	0.010632
min	5.268000	10.057600	3.000000	0.033292
25%	6.634500	11.189800	28.000000	0.046464
50%	7.999000	12.720000	39.000000	0.053507
75%	9.088000	14.170800	50.000000	0.061397
max	10.876000	101.120000	60.000000	0.086711

[91]:
```python
# First 5 Records in the dataset
data.head()
```

[91]:

	price	crime_rate	resid_area	air_qual	room_num	age	dist1	dist2	\
0	24.0	0.00632	32.31	0.538	6.575	65.2	4.35	3.81	
1	21.6	0.02731	37.07	0.469	6.421	78.9	4.99	4.70	
2	34.7	0.02729	37.07	0.469	7.185	61.1	5.03	4.86	
3	33.4	0.03237	32.18	0.458	6.998	45.8	6.21	5.93	
4	36.2	0.06905	32.18	0.458	7.147	54.2	6.16	5.86	

	dist3	dist4	teachers	poor_prop	airport	n_hos_beds	n_hot_rooms	\
0	4.18	4.01	24.7	4.98	YES	5.480	11.1920	
1	5.12	5.06	22.2	9.14	NO	7.332	12.1728	
2	5.01	4.97	22.2	4.03	NO	7.394	101.1200	
3	6.16	5.96	21.3	2.94	YES	9.268	11.2672	
4	6.37	5.86	21.3	5.33	NO	8.824	11.2896	

	waterbody	rainfall	bus_ter	parks
0	River	23	YES	0.049347
1	Lake	42	YES	0.046146
2	NaN	38	YES	0.045764
3	Lake	45	YES	0.047151
4	Lake	55	YES	0.039474

[92]:
```python
# Last 5 Records in the dataset
data.tail()
```

[92]:

	price	crime_rate	resid_area	air_qual	room_num	age	dist1	dist2	\
501	22.4	0.06263	41.93	0.573	6.593	69.1	2.64	2.45	
502	20.6	0.04527	41.93	0.573	6.120	76.7	2.44	2.11	
503	23.9	0.06076	41.93	0.573	6.976	91.0	2.34	2.06	
504	22.0	0.10959	41.93	0.573	6.794	89.3	2.54	2.31	
505	19.0	0.04741	41.93	0.573	6.030	80.8	2.72	2.24	

	dist3	dist4	teachers	poor_prop	airport	n_hos_beds	n_hot_rooms	\
501	2.76	2.06	19.0	9.67	NO	9.348	12.1792	

```
502   2.46   2.14      19.0       9.08     YES       6.612      13.1648
503   2.29   1.98      19.0       5.64      NO       5.478      12.1912
504   2.40   2.31      19.0       6.48     YES       7.940      15.1760
505   2.64   2.42      19.0       7.88     YES      10.280      10.1520

          waterbody  rainfall bus_ter     parks
501  Lake and River        27     YES  0.056006
502  Lake and River        20     YES  0.059903
503             NaN        31     YES  0.057572
504             NaN        47     YES  0.060694
505             NaN        45     YES  0.060336
```

2.2 Missing Value Detection:

2.2.1 Identify and count null values in each column:

[115]:
```python
# Load the House Price dataset
data = pd.read_csv('House_Price.csv')
data
```

[115]:
```
     price  crime_rate  resid_area  air_qual  room_num   age  dist1  dist2  \
0     24.0     0.00632       32.31     0.538     6.575  65.2   4.35   3.81
1     21.6     0.02731       37.07     0.469     6.421  78.9   4.99   4.70
2     34.7     0.02729       37.07     0.469     7.185  61.1   5.03   4.86
3     33.4     0.03237       32.18     0.458     6.998  45.8   6.21   5.93
4     36.2     0.06905       32.18     0.458     7.147  54.2   6.16   5.86
..     …           …           …         …         …     …     …      …
501   22.4     0.06263       41.93     0.573     6.593  69.1   2.64   2.45
502   20.6     0.04527       41.93     0.573     6.120  76.7   2.44   2.11
503   23.9     0.06076       41.93     0.573     6.976  91.0   2.34   2.06
504   22.0     0.10959       41.93     0.573     6.794  89.3   2.54   2.31
505   19.0     0.04741       41.93     0.573     6.030  80.8   2.72   2.24

     dist3  dist4  teachers  poor_prop airport  n_hos_beds  n_hot_rooms  \
0     4.18   4.01      24.7       4.98     YES       5.480      11.1920
1     5.12   5.06      22.2       9.14      NO       7.332      12.1728
2     5.01   4.97      22.2       4.03      NO       7.394     101.1200
3     6.16   5.96      21.3       2.94     YES       9.268      11.2672
4     6.37   5.86      21.3       5.33      NO       8.824      11.2896
..     …      …         …          …       …           …            …
501   2.76   2.06      19.0       9.67      NO       9.348      12.1792
502   2.46   2.14      19.0       9.08     YES       6.612      13.1648
503   2.29   1.98      19.0       5.64      NO       5.478      12.1912
504   2.40   2.31      19.0       6.48     YES       7.940      15.1760
505   2.64   2.42      19.0       7.88     YES      10.280      10.1520

          waterbody  rainfall bus_ter     parks
```

```
0              River        23      YES   0.049347
1               Lake        42      YES   0.046146
2                NaN        38      YES   0.045764
3               Lake        45      YES   0.047151
4               Lake        55      YES   0.039474
..               …       …      …        …
501  Lake and River        27      YES   0.056006
502  Lake and River        20      YES   0.059903
503              NaN        31      YES   0.057572
504              NaN        47      YES   0.060694
505              NaN        45      YES   0.060336

[506 rows x 19 columns]
```

[109]:
```python
# Lets see the number of Null values in each column
data.isnull().sum()
```

[109]:
```
price            0
crime_rate       0
resid_area       0
air_qual         0
room_num         0
age              0
dist1            0
dist2            0
dist3            0
dist4            0
teachers         0
poor_prop        0
airport          0
n_hos_beds       8
n_hot_rooms      0
waterbody      155
rainfall         0
bus_ter          0
parks            0
dtype: int64
```

[110]:
```python
data.isna().sum()
```

[110]:
```
price            0
crime_rate       0
resid_area       0
air_qual         0
room_num         0
age              0
dist1            0
```

```
dist2             0
dist3             0
dist4             0
teachers          0
poor_prop         0
airport           0
n_hos_beds        8
n_hot_rooms       0
waterbody       155
rainfall          0
bus_ter           0
parks             0
dtype: int64
```

[111]:
```
# Droping all null values
data = data.dropna(how='any' ,axis=0)
data.shape
```

[111]:
```
(343, 19)
```

[112]:
```
 # Here we can see the unique values exist in each column
 data.nunique()
```

[112]:
```
price           192
crime_rate      342
resid_area       72
air_qual         78
room_num        313
age             263
dist1           252
dist2           269
dist3           257
dist4           261
teachers         44
poor_prop       319
airport           2
n_hos_beds      320
n_hot_rooms     305
waterbody         3
rainfall         40
bus_ter           1
parks           343
dtype: int64
```

[114]:
```
# All Null Values record dropped
data.isnull().sum()
```

```
[114]: price          0
       crime_rate     0
       resid_area     0
       air_qual       0
       room_num       0
       age            0
       dist1          0
       dist2          0
       dist3          0
       dist4          0
       teachers       0
       poor_prop      0
       airport        0
       n_hos_beds     0
       n_hot_rooms    0
       waterbody      0
       rainfall       0
       bus_ter        0
       parks          0
       dtype: int64
```

2.3 Data Transformation: One-Hot Encoding and Data Type Conversion

2.3.1 Convert categorical variables to numerical variables and change data type to integer

```
[146]: # Load the House Price dataset
       data = pd.read_csv('House_Price.csv')
       data
```

```
[146]:      price  crime_rate  resid_area  air_qual  room_num   age  dist1  dist2  \
       0     24.0     0.00632       32.31     0.538     6.575  65.2   4.35   3.81
       1     21.6     0.02731       37.07     0.469     6.421  78.9   4.99   4.70
       2     34.7     0.02729       37.07     0.469     7.185  61.1   5.03   4.86
       3     33.4     0.03237       32.18     0.458     6.998  45.8   6.21   5.93
       4     36.2     0.06905       32.18     0.458     7.147  54.2   6.16   5.86
       ..     …          …          …        …         …     …      …      …
       501   22.4     0.06263       41.93     0.573     6.593  69.1   2.64   2.45
       502   20.6     0.04527       41.93     0.573     6.120  76.7   2.44   2.11
       503   23.9     0.06076       41.93     0.573     6.976  91.0   2.34   2.06
       504   22.0     0.10959       41.93     0.573     6.794  89.3   2.54   2.31
       505   19.0     0.04741       41.93     0.573     6.030  80.8   2.72   2.24

            dist3  dist4  teachers  poor_prop airport  n_hos_beds  n_hot_rooms  \
       0     4.18   4.01      24.7       4.98     YES       5.480      11.1920
       1     5.12   5.06      22.2       9.14      NO       7.332      12.1728
       2     5.01   4.97      22.2       4.03      NO       7.394     101.1200
       3     6.16   5.96      21.3       2.94     YES       9.268      11.2672
```

```
4     6.37   5.86      21.3       5.33      NO       8.824       11.2896
..     …     …        …        …      …       …         …
501   2.76   2.06      19.0       9.67      NO       9.348       12.1792
502   2.46   2.14      19.0       9.08     YES       6.612       13.1648
503   2.29   1.98      19.0       5.64      NO       5.478       12.1912
504   2.40   2.31      19.0       6.48     YES       7.940       15.1760
505   2.64   2.42      19.0       7.88     YES      10.280       10.1520

          waterbody  rainfall bus_ter     parks
0             River        23     YES  0.049347
1              Lake        42     YES  0.046146
2               NaN        38     YES  0.045764
3              Lake        45     YES  0.047151
4              Lake        55     YES  0.039474
..              …       …     …       …
501  Lake and River        27     YES  0.056006
502  Lake and River        20     YES  0.059903
503             NaN        31     YES  0.057572
504             NaN        47     YES  0.060694
505             NaN        45     YES  0.060336

[506 rows x 19 columns]
```

[147]:

```python
import numpy as np
import pandas as pd

# Load the dataset
data = pd.read_csv('House_Price.csv')

# Replace infinite values with NaN
data.replace([np.inf, -np.inf], np.nan, inplace=True)

# Convert all non-numeric columns to NaN and coerce data to numeric where
 ↪possible
for col in data.columns:
    # Attempt to convert to numeric, if fails, leave as is
    data[col] = pd.to_numeric(data[col], errors='coerce')

# Fill NaN values with 0 (including non-convertible values)
data.fillna(0, inplace=True)

# Convert to integers
data = data.astype(int)

# Display the processed data
data.head()
```

[147]:
```
   price  crime_rate  resid_area  air_qual  room_num  age  dist1  dist2  \
0     24           0          32         0         6   65      4      3
1     21           0          37         0         6   78      4      4
2     34           0          37         0         7   61      5      4
3     33           0          32         0         6   45      6      5
4     36           0          32         0         7   54      6      5

   dist3  dist4  teachers  poor_prop  airport  n_hos_beds  n_hot_rooms  \
0      4      4        24          4        0           5           11
1      5      5        22          9        0           7           12
2      5      4        22          4        0           7          101
3      6      5        21          2        0           9           11
4      6      5        21          5        0           8           11

   waterbody  rainfall  bus_ter  parks
0          0        23        0      0
1          0        42        0      0
2          0        38        0      0
3          0        45        0      0
4          0        55        0      0
```

[148]:
```
import numpy as np
data.dropna(inplace=True)
data = data.astype(int)

data.fillna(0, inplace=True)  # Replace NaN with 0
data = data.astype(int)

data = data.astype(float)

data.replace([np.inf, -np.inf], np.nan, inplace=True)
data.fillna(0, inplace=True)  # Replace NaN with 0
data = data.astype(int)
data
```

[148]:
```
     price  crime_rate  resid_area  air_qual  room_num  age  dist1  dist2  \
0       24           0          32         0         6   65      4      3
1       21           0          37         0         6   78      4      4
2       34           0          37         0         7   61      5      4
3       33           0          32         0         6   45      6      5
4       36           0          32         0         7   54      6      5
..     …         …         …         …       …  …    …    …
501     22           0          41         0         6   69      2      2
502     20           0          41         0         6   76      2      2
503     23           0          41         0         6   91      2      2
504     22           0          41         0         6   89      2      2
505     19           0          41         0         6   80      2      2
```

```
     dist3  dist4  teachers  poor_prop  airport  n_hos_beds  n_hot_rooms  \
0        4      4        24          4        0           5           11
1        5      5        22          9        0           7           12
2        5      4        22          4        0           7          101
3        6      5        21          2        0           9           11
4        6      5        21          5        0           8           11
..     …      …       …         …       …        …          …
501      2      2        19          9        0           9           12
502      2      2        19          9        0           6           13
503      2      1        19          5        0           5           12
504      2      2        19          6        0           7           15
505      2      2        19          7        0          10           10

     waterbody  rainfall  bus_ter  parks
0            0        23        0      0
1            0        42        0      0
2            0        38        0      0
3            0        45        0      0
4            0        55        0      0
..         …        …       …     …
501          0        27        0      0
502          0        20        0      0
503          0        31        0      0
504          0        47        0      0
505          0        45        0      0

[506 rows x 19 columns]
```

[149]:
```python
data= pd.get_dummies(data,drop_first = True)
data =data.astype(int)
data
```

[149]:
```
     price  crime_rate  resid_area  air_qual  room_num  age  dist1  dist2  \
0       24           0          32         0         6   65      4      3
1       21           0          37         0         6   78      4      4
2       34           0          37         0         7   61      5      4
3       33           0          32         0         6   45      6      5
4       36           0          32         0         7   54      6      5
..     …          …          …        …        …  …    …    …
501     22           0          41         0         6   69      2      2
502     20           0          41         0         6   76      2      2
503     23           0          41         0         6   91      2      2
504     22           0          41         0         6   89      2      2
505     19           0          41         0         6   80      2      2

     dist3  dist4  teachers  poor_prop  airport  n_hos_beds  n_hot_rooms  \
```

```
0      4    4     24     4    0     5     11
1      5    5     22     9    0     7     12
2      5    4     22     4    0     7    101
3      6    5     21     2    0     9     11
4      6    5     21     5    0     8     11
..   ...  ...    ...   ...  ...   ...    ...
501    2    2     19     9    0     9     12
502    2    2     19     9    0     6     13
503    2    1     19     5    0     5     12
504    2    2     19     6    0     7     15
505    2    2     19     7    0    10     10

     waterbody  rainfall  bus_ter  parks
0            0        23        0      0
1            0        42        0      0
2            0        38        0      0
3            0        45        0      0
4            0        55        0      0
..         ...       ...      ...    ...
501          0        27        0      0
502          0        20        0      0
503          0        31        0      0
504          0        47        0      0
505          0        45        0      0

[506 rows x 19 columns]
```

[ ]:

2.4 Renaming Columns for Clarity

2.4.1 Update column names for better understanding and analysis

[196]:
```python
# Load the House Price dataset
data = pd.read_csv('House_Price.csv')
data
```

[196]:
```
     price  crime_rate  resid_area  air_qual  room_num   age  dist1  dist2  \
0     24.0     0.00632       32.31     0.538     6.575  65.2   4.35   3.81
1     21.6     0.02731       37.07     0.469     6.421  78.9   4.99   4.70
2     34.7     0.02729       37.07     0.469     7.185  61.1   5.03   4.86
3     33.4     0.03237       32.18     0.458     6.998  45.8   6.21   5.93
4     36.2     0.06905       32.18     0.458     7.147  54.2   6.16   5.86
..     ...         ...         ...       ...       ...   ...    ...    ...
501   22.4     0.06263       41.93     0.573     6.593  69.1   2.64   2.45
502   20.6     0.04527       41.93     0.573     6.120  76.7   2.44   2.11
503   23.9     0.06076       41.93     0.573     6.976  91.0   2.34   2.06
504   22.0     0.10959       41.93     0.573     6.794  89.3   2.54   2.31
```

```
505   19.0     0.04741       41.93     0.573     6.030  80.8   2.72   2.24

     dist3  dist4  teachers  poor_prop airport  n_hos_beds  n_hot_rooms  \
0     4.18   4.01      24.7       4.98     YES       5.480      11.1920
1     5.12   5.06      22.2       9.14      NO       7.332      12.1728
2     5.01   4.97      22.2       4.03      NO       7.394     101.1200
3     6.16   5.96      21.3       2.94     YES       9.268      11.2672
4     6.37   5.86      21.3       5.33      NO       8.824      11.2896
..     …      …         …         …       …          …           …
501   2.76   2.06      19.0       9.67      NO       9.348      12.1792
502   2.46   2.14      19.0       9.08     YES       6.612      13.1648
503   2.29   1.98      19.0       5.64      NO       5.478      12.1912
504   2.40   2.31      19.0       6.48     YES       7.940      15.1760
505   2.64   2.42      19.0       7.88     YES      10.280      10.1520

          waterbody  rainfall bus_ter     parks
0             River        23     YES  0.049347
1              Lake        42     YES  0.046146
2               NaN        38     YES  0.045764
3              Lake        45     YES  0.047151
4              Lake        55     YES  0.039474
..              …         …     …        …
501  Lake and River        27     YES  0.056006
502  Lake and River        20     YES  0.059903
503             NaN        31     YES  0.057572
504             NaN        47     YES  0.060694
505             NaN        45     YES  0.060336

[506 rows x 19 columns]
```

[197]:
```
data.columns
```

[197]:
```
Index(['price', 'crime_rate', 'resid_area', 'air_qual', 'room_num', 'age',
       'dist1', 'dist2', 'dist3', 'dist4', 'teachers', 'poor_prop', 'airport',
       'n_hos_beds', 'n_hot_rooms', 'waterbody', 'rainfall', 'bus_ter',
       'parks'],
      dtype='object')
```

[198]:
```
# Renaming the columns as per the existing dataset
data.rename(columns={'price': 'house_price', 'crime_rate': 'safety_index'},␣
 ↪inplace=True)

# Verify the renaming
print(data.columns)

# Display the first few rows
data.head()
```

```
data
```

```
Index(['house_price', 'safety_index', 'resid_area', 'air_qual', 'room_num',
       'age', 'dist1', 'dist2', 'dist3', 'dist4', 'teachers', 'poor_prop',
       'airport', 'n_hos_beds', 'n_hot_rooms', 'waterbody', 'rainfall',
       'bus_ter', 'parks'],
      dtype='object')
```

[198]:

```
     house_price  safety_index  resid_area  air_qual  room_num   age  dist1  \
0           24.0       0.00632       32.31     0.538     6.575  65.2   4.35
1           21.6       0.02731       37.07     0.469     6.421  78.9   4.99
2           34.7       0.02729       37.07     0.469     7.185  61.1   5.03
3           33.4       0.03237       32.18     0.458     6.998  45.8   6.21
4           36.2       0.06905       32.18     0.458     7.147  54.2   6.16
..           …             …           …        …         …    …     …
501         22.4       0.06263       41.93     0.573     6.593  69.1   2.64
502         20.6       0.04527       41.93     0.573     6.120  76.7   2.44
503         23.9       0.06076       41.93     0.573     6.976  91.0   2.34
504         22.0       0.10959       41.93     0.573     6.794  89.3   2.54
505         19.0       0.04741       41.93     0.573     6.030  80.8   2.72

     dist2  dist3  dist4  teachers  poor_prop airport  n_hos_beds  \
0     3.81   4.18   4.01      24.7       4.98     YES       5.480
1     4.70   5.12   5.06      22.2       9.14      NO       7.332
2     4.86   5.01   4.97      22.2       4.03      NO       7.394
3     5.93   6.16   5.96      21.3       2.94     YES       9.268
4     5.86   6.37   5.86      21.3       5.33      NO       8.824
..     …      …      …         …          …       …           …
501   2.45   2.76   2.06      19.0       9.67      NO       9.348
502   2.11   2.46   2.14      19.0       9.08     YES       6.612
503   2.06   2.29   1.98      19.0       5.64      NO       5.478
504   2.31   2.40   2.31      19.0       6.48     YES       7.940
505   2.24   2.64   2.42      19.0       7.88     YES      10.280

     n_hot_rooms       waterbody  rainfall bus_ter     parks
0        11.1920           River        23     YES  0.049347
1        12.1728            Lake        42     YES  0.046146
2       101.1200             NaN        38     YES  0.045764
3        11.2672            Lake        45     YES  0.047151
4        11.2896            Lake        55     YES  0.039474
..           …               …      …         …       …
501      12.1792  Lake and River        27     YES  0.056006
502      13.1648  Lake and River        20     YES  0.059903
503      12.1912             NaN        31     YES  0.057572
504      15.1760             NaN        47     YES  0.060694
505      10.1520             NaN        45     YES  0.060336
```

```
[506 rows x 19 columns]
```

[199]:
```
data.columns
```

[199]:
```
Index(['house_price', 'safety_index', 'resid_area', 'air_qual', 'room_num',
       'age', 'dist1', 'dist2', 'dist3', 'dist4', 'teachers', 'poor_prop',
       'airport', 'n_hos_beds', 'n_hot_rooms', 'waterbody', 'rainfall',
       'bus_ter', 'parks'],
      dtype='object')
```

[200]:
```
# After Loading the House Price dataset
data = pd.read_csv('House_Price.csv')
data.columns
```

[200]:
```
Index(['price', 'crime_rate', 'resid_area', 'air_qual', 'room_num', 'age',
       'dist1', 'dist2', 'dist3', 'dist4', 'teachers', 'poor_prop', 'airport',
       'n_hos_beds', 'n_hot_rooms', 'waterbody', 'rainfall', 'bus_ter',
       'parks'],
      dtype='object')
```

[201]:
```
# Store the existing (old) column names
existing_columns = data.columns.copy()

# Rename the columns as per the new mapping
data.rename(columns={'price': 'house_price', 'crime_rate': 'safety_index'},␣
 ↪inplace=True)

# Store the new column names
new_columns = data.columns

# Display the changes in column names (comparison)
print("Existing Column Names vs New Column Names:")
for old_col, new_col in zip(existing_columns, new_columns):
    if old_col != new_col:
        print(f"'{old_col}' renamed to '{new_col}'")
    else:
        print(f"'{old_col}' remains unchanged.")

# Display the new column names
print("\nNew Column Names:")
print(new_columns)
```

```
Existing Column Names vs New Column Names:
'price' renamed to 'house_price'
'crime_rate' renamed to 'safety_index'
'resid_area' remains unchanged.
'air_qual' remains unchanged.
'room_num' remains unchanged.
```

```
'age' remains unchanged.
'dist1' remains unchanged.
'dist2' remains unchanged.
'dist3' remains unchanged.
'dist4' remains unchanged.
'teachers' remains unchanged.
'poor_prop' remains unchanged.
'airport' remains unchanged.
'n_hos_beds' remains unchanged.
'n_hot_rooms' remains unchanged.
'waterbody' remains unchanged.
'rainfall' remains unchanged.
'bus_ter' remains unchanged.
'parks' remains unchanged.

New Column Names:
Index(['house_price', 'safety_index', 'resid_area', 'air_qual', 'room_num',
       'age', 'dist1', 'dist2', 'dist3', 'dist4', 'teachers', 'poor_prop',
       'airport', 'n_hos_beds', 'n_hot_rooms', 'waterbody', 'rainfall',
       'bus_ter', 'parks'],
      dtype='object')
```

[202]:

```
# Store the existing (old) column names
existing_columns = data.columns.tolist()

# Rename the columns as per the new mapping
data.rename(columns={'price': 'house_price', 'crime_rate': 'safety_index'},␣
↪inplace=True)

# Store the new column names
new_columns = data.columns.tolist()

# Display the changes in column names (comparison)
print("Column Name Changes:")
changes_found = False
for old_col in existing_columns:
    if old_col in new_columns:
        new_col = new_columns[new_columns.index(old_col)]  # Get the new column␣
↪name if it exists
        if old_col != new_col:
            print(f"'{old_col}' renamed to '{new_col}'")
            changes_found = True
    else:
        print(f"'{old_col}' was removed from the dataset.")

for new_col in new_columns:
    if new_col not in existing_columns:
```

```
        print(f"'{new_col}' is a new column added to the dataset.")
        changes_found = True

if not changes_found:
    print("No changes detected in column names.")

# Display the new column names
print("\nNew Column Names:")
print(new_columns)
```

```
Column Name Changes:
No changes detected in column names.

New Column Names:
['house_price', 'safety_index', 'resid_area', 'air_qual', 'room_num', 'age',
'dist1', 'dist2', 'dist3', 'dist4', 'teachers', 'poor_prop', 'airport',
'n_hos_beds', 'n_hot_rooms', 'waterbody', 'rainfall', 'bus_ter', 'parks']
```

[203]:

```
import pandas as pd

# Load your dataset
data = pd.read_csv('House_Price.csv')

# Store the existing (old) column names
existing_columns = data.columns.tolist()

# Renaming the columns as per the existing dataset
data.rename(columns={'price': 'house_price', 'crime_rate': 'safety_index'},␣
 ↪inplace=True)

# Store the new column names
new_columns = data.columns.tolist()

# Verify the renaming
print("Updated Column Names:")
print(new_columns)

# Check and display changes in column names
print("\nColumn Name Changes:")
changes_found = False
for old_col in existing_columns:
    if old_col in new_columns:
        new_col = new_columns[new_columns.index(old_col)]  # Get the new column␣
 ↪name if it exists
        if old_col != new_col:
            print(f"'{old_col}' renamed to '{new_col}'")
            changes_found = True
```

```python
    else:
        print(f"'{old_col}' was removed from the dataset.")

for new_col in new_columns:
    if new_col not in existing_columns:
        print(f"'{new_col}' is a new column added to the dataset.")
        changes_found = True

if not changes_found:
    print("No changes detected in column names.")

# Display the first few rows of the updated DataFrame
print("\nFirst few rows of the updated DataFrame:")
print(data.head())
```

```
Updated Column Names:
['house_price', 'safety_index', 'resid_area', 'air_qual', 'room_num', 'age',
'dist1', 'dist2', 'dist3', 'dist4', 'teachers', 'poor_prop', 'airport',
'n_hos_beds', 'n_hot_rooms', 'waterbody', 'rainfall', 'bus_ter', 'parks']

Column Name Changes:
'price' was removed from the dataset.
'crime_rate' was removed from the dataset.
'house_price' is a new column added to the dataset.
'safety_index' is a new column added to the dataset.

First few rows of the updated DataFrame:
   house_price  safety_index  resid_area  air_qual  room_num   age  dist1  \
0         24.0       0.00632       32.31     0.538     6.575  65.2   4.35
1         21.6       0.02731       37.07     0.469     6.421  78.9   4.99
2         34.7       0.02729       37.07     0.469     7.185  61.1   5.03
3         33.4       0.03237       32.18     0.458     6.998  45.8   6.21
4         36.2       0.06905       32.18     0.458     7.147  54.2   6.16

   dist2  dist3  dist4  teachers  poor_prop airport  n_hos_beds  n_hot_rooms  \
0   3.81   4.18   4.01      24.7       4.98     YES       5.480      11.1920
1   4.70   5.12   5.06      22.2       9.14      NO       7.332      12.1728
2   4.86   5.01   4.97      22.2       4.03      NO       7.394     101.1200
3   5.93   6.16   5.96      21.3       2.94     YES       9.268      11.2672
4   5.86   6.37   5.86      21.3       5.33      NO       8.824      11.2896

  waterbody  rainfall bus_ter     parks
0     River        23     YES  0.049347
1      Lake        42     YES  0.046146
2       NaN        38     YES  0.045764
3      Lake        45     YES  0.047151
4      Lake        55     YES  0.039474
```

[204]:
```
data
```

[204]:
```
     house_price  safety_index  resid_area  air_qual  room_num   age  dist1  \
0           24.0       0.00632       32.31     0.538     6.575  65.2   4.35
1           21.6       0.02731       37.07     0.469     6.421  78.9   4.99
2           34.7       0.02729       37.07     0.469     7.185  61.1   5.03
3           33.4       0.03237       32.18     0.458     6.998  45.8   6.21
4           36.2       0.06905       32.18     0.458     7.147  54.2   6.16
..           …             …           …        …         …     …      …
501         22.4       0.06263       41.93     0.573     6.593  69.1   2.64
502         20.6       0.04527       41.93     0.573     6.120  76.7   2.44
503         23.9       0.06076       41.93     0.573     6.976  91.0   2.34
504         22.0       0.10959       41.93     0.573     6.794  89.3   2.54
505         19.0       0.04741       41.93     0.573     6.030  80.8   2.72

     dist2  dist3  dist4  teachers  poor_prop airport  n_hos_beds  \
0     3.81   4.18   4.01      24.7       4.98     YES       5.480
1     4.70   5.12   5.06      22.2       9.14      NO       7.332
2     4.86   5.01   4.97      22.2       4.03      NO       7.394
3     5.93   6.16   5.96      21.3       2.94     YES       9.268
4     5.86   6.37   5.86      21.3       5.33      NO       8.824
..     …      …      …         …          …       …           …
501   2.45   2.76   2.06      19.0       9.67      NO       9.348
502   2.11   2.46   2.14      19.0       9.08     YES       6.612
503   2.06   2.29   1.98      19.0       5.64      NO       5.478
504   2.31   2.40   2.31      19.0       6.48     YES       7.940
505   2.24   2.64   2.42      19.0       7.88     YES      10.280

     n_hot_rooms       waterbody  rainfall bus_ter     parks
0        11.1920           River        23     YES  0.049347
1        12.1728            Lake        42     YES  0.046146
2       101.1200             NaN        38     YES  0.045764
3        11.2672            Lake        45     YES  0.047151
4        11.2896            Lake        55     YES  0.039474
..           …               …         …       …        …
501      12.1792  Lake and River        27     YES  0.056006
502      13.1648  Lake and River        20     YES  0.059903
503      12.1912             NaN        31     YES  0.057572
504      15.1760             NaN        47     YES  0.060694
505      10.1520             NaN        45     YES  0.060336

[506 rows x 19 columns]
```

[205]:
```
data.columns
```

[205]:
```
Index(['house_price', 'safety_index', 'resid_area', 'air_qual', 'room_num',
       'age', 'dist1', 'dist2', 'dist3', 'dist4', 'teachers', 'poor_prop',
```

```
       'airport', 'n_hos_beds', 'n_hot_rooms', 'waterbody', 'rainfall',
       'bus_ter', 'parks'],
      dtype='object')
```

[206]:
```python
# After Loading the House Price dataset
data = pd.read_csv('House_Price.csv')
data
data.columns
```

[206]:
```
Index(['price', 'crime_rate', 'resid_area', 'air_qual', 'room_num', 'age',
       'dist1', 'dist2', 'dist3', 'dist4', 'teachers', 'poor_prop', 'airport',
       'n_hos_beds', 'n_hot_rooms', 'waterbody', 'rainfall', 'bus_ter',
       'parks'],
      dtype='object')
```

[ ]:
```python

```

2.5 Data Preparation: Splitting Variables

2.5.1 Split data into independent (input) and dependent (output) variables

[209]:
```python
# House Price dataset columns name
data = pd.read_csv('House_Price.csv')
data
data.columns
```

[209]:
```
Index(['price', 'crime_rate', 'resid_area', 'air_qual', 'room_num', 'age',
       'dist1', 'dist2', 'dist3', 'dist4', 'teachers', 'poor_prop', 'airport',
       'n_hos_beds', 'n_hot_rooms', 'waterbody', 'rainfall', 'bus_ter',
       'parks'],
      dtype='object')
```

[211]:
```python
# Splitting independent (input) and dependent (output) variables
independent = data[['crime_rate', 'resid_area', 'air_qual', 'room_num', 'age',
                    'dist1', 'dist2', 'dist3', 'dist4', 'teachers',
                    'poor_prop', 'airport', 'n_hos_beds', 'n_hot_rooms',
                    'waterbody', 'rainfall', 'bus_ter', 'parks']]

dependent = data['price']

# Display the independent and dependent variables
independent.head(), dependent.head()
```

[211]:
```
(   crime_rate  resid_area  air_qual  room_num   age  dist1  dist2  dist3  \
 0     0.00632       32.31     0.538     6.575  65.2   4.35   3.81   4.18
 1     0.02731       37.07     0.469     6.421  78.9   4.99   4.70   5.12
 2     0.02729       37.07     0.469     7.185  61.1   5.03   4.86   5.01
 3     0.03237       32.18     0.458     6.998  45.8   6.21   5.93   6.16
```

```
4     0.06905      32.18     0.458     7.147  54.2   6.16   5.86   6.37

   dist4  teachers  poor_prop airport  n_hos_beds  n_hot_rooms waterbody  \
0   4.01      24.7       4.98     YES       5.480      11.1920     River
1   5.06      22.2       9.14      NO       7.332      12.1728      Lake
2   4.97      22.2       4.03      NO       7.394     101.1200       NaN
3   5.96      21.3       2.94     YES       9.268      11.2672      Lake
4   5.86      21.3       5.33      NO       8.824      11.2896      Lake

   rainfall bus_ter     parks
0        23     YES  0.049347
1        42     YES  0.046146
2        38     YES  0.045764
3        45     YES  0.047151
4        55     YES  0.039474  ,
0    24.0
1    21.6
2    34.7
3    33.4
4    36.2
Name: price, dtype: float64)
```

[214]:

```

```

2.6 Split Data into Training and Testing Sets:

[225]:

```
data = pd.read_csv('House_Price.csv')
```

[226]:

```
# split data into training and testing

import pandas as pd
from sklearn.model_selection import train_test_split

# Load your dataset
data = pd.read_csv('House_Price.csv')

# Define your feature columns (independent variables) and target column␣
↪(dependent variable)
features = data.drop(columns='price')  # Replace 'price' with the name of your␣
↪target variable if needed
target = data['price']  # Replace 'price' with the name of your target variable␣
↪if needed

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target,␣
↪test_size=0.2, random_state=42)
```

```python
# Display the shapes of the resulting datasets
print("Training Features Shape:", X_train.shape)
print("Testing Features Shape:", X_test.shape)
print("Training Target Shape:", y_train.shape)
print("Testing Target Shape:", y_test.shape)
```

```
Training Features Shape: (404, 18)
Testing Features Shape: (102, 18)
Training Target Shape: (404,)
Testing Target Shape: (102,)
```

[ ]:

```python

```

2.7 Data Preparation and Preprocessing for House Price Prediction

2.7.1 Load dataset, split data, and create preprocessing pipeline

[232]:

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Load your dataset
data = pd.read_csv('House_Price.csv')

# Define the features (independent variables) and the target (dependent␣
 ↪variable)
X = data.drop(columns='price')  # Features
y = data['price']                # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Identify categorical and numerical columns
categorical_cols = X_train.select_dtypes(include=['object']).columns
numerical_cols = X_train.select_dtypes(exclude=['object']).columns

# Create a preprocessor that applies one-hot encoding to categorical features␣
 ↪and scales numerical features
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),  # Scale numerical features
        ('cat', OneHotEncoder(), categorical_cols)   # One-hot encode␣
 ↪categorical features
    ])
```

```python
# Create a pipeline that first preprocesses the data, then fits the model
pipeline = Pipeline(steps=[('preprocessor', preprocessor)])

# Fit and transform the training data
X_train_transformed = pipeline.fit_transform(X_train)

# Transform the test data
X_test_transformed = pipeline.transform(X_test)

# Display the shapes of the resulting datasets
print("Transformed Training Features Shape:", X_train_transformed.shape)
print("Transformed Testing Features Shape:", X_test_transformed.shape)
print("Training Target Shape:", y_train.shape)
print("Testing Target Shape:", y_test.shape)

# Optionally, print the first few rows of the transformed training data
print("Transformed Training Features:\n", X_train_transformed[:5])
```

```
Transformed Training Features Shape: (404, 22)
Transformed Testing Features Shape: (102, 22)
Training Target Shape: (404,)
Testing Target Shape: (102,)
Transformed Training Features:
 [[ 1.28770177e+00  1.03323679e+00  4.89252063e-01 -1.42806858e+00
   1.02801516e+00 -8.01761699e-01 -7.74810776e-01 -7.28016405e-01
  -8.97073577e-01 -8.45342815e-01  1.75350503e+00  9.50291168e-01
   5.92920344e-01  1.48266001e-02  6.66951803e-01  1.00000000e+00
   0.00000000e+00  1.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  1.00000000e+00]
 [-3.36384470e-01 -4.13159558e-01 -1.57233423e-01 -6.80086552e-01
  -4.31199082e-01  2.87001439e-01  3.31660558e-01  3.49238712e-01
   3.25190194e-01 -1.20474139e+00 -5.61474201e-01  3.80047548e-01
   3.45531038e-01 -8.63698876e-01  4.64002903e-01  0.00000000e+00
   1.00000000e+00  0.00000000e+00  0.00000000e+00  1.00000000e+00
   0.00000000e+00  1.00000000e+00]
 [-4.03253321e-01 -7.15218233e-01 -1.00872286e+00 -4.02063044e-01
  -1.61859890e+00  1.38984341e+00  1.37718220e+00  1.31923466e+00
   1.22655073e+00  6.37176313e-01 -6.51595047e-01 -3.03973896e-01
   3.44474721e-01 -1.34289459e+00 -2.83410911e-01  0.00000000e+00
   1.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   1.00000000e+00  1.00000000e+00]
 [ 3.88229827e-01  1.03323679e+00  4.89252063e-01 -3.00450392e-01
   5.91681487e-01 -8.95620590e-01 -8.63891095e-01 -7.60660500e-01
  -8.31005265e-01 -8.45342815e-01  1.52538664e+00  2.24280289e-01
   1.80284792e+01 -7.83832924e-01  7.99934897e-01  1.00000000e+00
   0.00000000e+00  1.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  1.00000000e+00]
```

```
[-3.25282344e-01 -4.13159558e-01 -1.57233423e-01 -8.31094243e-01
  3.37466310e-02 -3.96112399e-03  3.46990821e-03  1.34708834e-02
 -2.87471916e-02 -1.20474139e+00 -1.65787362e-01  4.24745979e-01
 -1.86218650e-01 -9.43564829e-01  2.78838852e-01  1.00000000e+00
  0.00000000e+00  0.00000000e+00  1.00000000e+00  0.00000000e+00
  0.00000000e+00  1.00000000e+00]]
```

[ ]:

2.8 House Price Prediction using Linear Regression:

2.8.1 Load dataset, split data, preprocess, train model, and evaluate

[243]:
```python
# Load the House Price dataset
data = pd.read_csv('House_Price.csv')
data
```

[243]:
```
     price  crime_rate  resid_area  air_qual  room_num   age  dist1  dist2  \
0     24.0     0.00632       32.31     0.538     6.575  65.2   4.35   3.81
1     21.6     0.02731       37.07     0.469     6.421  78.9   4.99   4.70
2     34.7     0.02729       37.07     0.469     7.185  61.1   5.03   4.86
3     33.4     0.03237       32.18     0.458     6.998  45.8   6.21   5.93
4     36.2     0.06905       32.18     0.458     7.147  54.2   6.16   5.86
..     …         …           …         …         …     …      …      …
501   22.4     0.06263       41.93     0.573     6.593  69.1   2.64   2.45
502   20.6     0.04527       41.93     0.573     6.120  76.7   2.44   2.11
503   23.9     0.06076       41.93     0.573     6.976  91.0   2.34   2.06
504   22.0     0.10959       41.93     0.573     6.794  89.3   2.54   2.31
505   19.0     0.04741       41.93     0.573     6.030  80.8   2.72   2.24

     dist3  dist4  teachers  poor_prop airport  n_hos_beds  n_hot_rooms  \
0     4.18   4.01      24.7       4.98     YES       5.480      11.1920
1     5.12   5.06      22.2       9.14      NO       7.332      12.1728
2     5.01   4.97      22.2       4.03      NO       7.394     101.1200
3     6.16   5.96      21.3       2.94     YES       9.268      11.2672
4     6.37   5.86      21.3       5.33      NO       8.824      11.2896
..     …      …         …          …       …          …            …
501   2.76   2.06      19.0       9.67      NO       9.348      12.1792
502   2.46   2.14      19.0       9.08     YES       6.612      13.1648
503   2.29   1.98      19.0       5.64      NO       5.478      12.1912
504   2.40   2.31      19.0       6.48     YES       7.940      15.1760
505   2.64   2.42      19.0       7.88     YES      10.280      10.1520

          waterbody  rainfall bus_ter     parks
0             River        23     YES  0.049347
1              Lake        42     YES  0.046146
2               NaN        38     YES  0.045764
3              Lake        45     YES  0.047151
```

```
4             Lake        55     YES  0.039474
..              …       …    …       …
501  Lake and River        27     YES  0.056006
502  Lake and River        20     YES  0.059903
503             NaN        31     YES  0.057572
504             NaN        47     YES  0.060694
505             NaN        45     YES  0.060336

[506 rows x 19 columns]
```

[244]:
```
data.columns
```

[244]:
```
Index(['price', 'crime_rate', 'resid_area', 'air_qual', 'room_num', 'age',
       'dist1', 'dist2', 'dist3', 'dist4', 'teachers', 'poor_prop', 'airport',
       'n_hos_beds', 'n_hot_rooms', 'waterbody', 'rainfall', 'bus_ter',
       'parks'],
      dtype='object')
```

[49]:
```
import pandas as pd

data = pd.read_csv("House_Price.csv")

# Print the column names
print(data.columns)
```

```
Index(['price', 'crime_rate', 'resid_area', 'air_qual', 'room_num', 'age',
       'dist1', 'dist2', 'dist3', 'dist4', 'teachers', 'poor_prop', 'airport',
       'n_hos_beds', 'n_hot_rooms', 'waterbody', 'rainfall', 'bus_ter',
       'parks'],
      dtype='object')
```

[8]:
```
from sklearn.linear_model import LinearRegression
regressor_f = LinearRegression()
```

[ ]:
```

```

[ ]:
```

```

2.9 Mean Squared Error (MSE), Root Mean Squared Error (RMSE),R-squared and Mean Absolute Error (MAE) Results:

[32]:
```
import pandas as pd

data = pd.read_csv("House_Price.csv")

# Print the column names
print(data.columns)
```

```
Index(['price', 'crime_rate', 'resid_area', 'air_qual', 'room_num', 'age',
       'dist1', 'dist2', 'dist3', 'dist4', 'teachers', 'poor_prop', 'airport',
       'n_hos_beds', 'n_hot_rooms', 'waterbody', 'rainfall', 'bus_ter',
       'parks'],
      dtype='object')
```

[33]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

# Load your dataset
data = pd.read_csv('House_Price.csv')

# Define the features (independent variables) and the target (dependent␣
↪variable)
X = data.drop(columns='price')  # Features
y = data['price']               # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Identify categorical and numerical columns
categorical_cols = X_train.select_dtypes(include=['object']).columns
numerical_cols = X_train.select_dtypes(exclude=['object']).columns

# Create a preprocessor that handles missing values, applies one-hot encoding␣
↪to categorical features,
# and scales numerical features
preprocessor = ColumnTransformer(
    transformers=[
        ('num', Pipeline(steps=[
            ('imputer', SimpleImputer(strategy='mean')),  # Impute missing␣
↪values with mean
            ('scaler', StandardScaler())                 # Scale numerical␣
↪features
        ]), numerical_cols),
        ('cat', Pipeline(steps=[
            ('imputer', SimpleImputer(strategy='most_frequent')),  # Impute␣
↪missing values with mode
            ('onehot', OneHotEncoder(handle_unknown='ignore'))   # One-hot␣
↪encode categorical features
```

```python
        ]), categorical_cols)
    ]
)

# Create a pipeline that first preprocesses the data, then fits the model␣
 ↪(LinearRegression)
pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('regressor',␣
 ↪LinearRegression())])

# Fit the model on the training data
pipeline.fit(X_train, y_train)

# Predict on the test data
y_pred = pipeline.predict(X_test)

# Evaluate the model performance:
# - Mean Squared Error (MSE): Measures the average squared difference between␣
 ↪predictions and actual values
mse = mean_squared_error(y_test, y_pred)

# - Root Mean Squared Error (RMSE): Square root of MSE, provides units of the␣
 ↪target variable
rmse = mean_squared_error(y_test, y_pred, squared=False)

# - R-squared: Coefficient of determination, measures the proportion of␣
 ↪variance explained by the model
r2 = r2_score(y_test, y_pred)

# - Mean Absolute Error (MAE): Measures the average absolute difference between␣
 ↪predictions and actual values
mae = mean_absolute_error(y_test, y_pred)

# Display the evaluation metrics
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R-squared:", r2)
print("Mean Absolute Error (MAE):", mae)

# Optionally, display the first few predictions vs actual values
results = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(results.head())
```

```
Mean Squared Error (MSE): 25.954592013593974
Root Mean Squared Error (RMSE): 5.09456494841257
R-squared: 0.6480459399756863
Mean Absolute Error (MAE): 3.35022741873567
     Actual  Predicted
```

```
173    23.6  30.962751
274    32.4  32.264156
491    13.6  17.151003
72     22.8  23.397942
452    16.1  15.954123
```

[34]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load your dataset
data = pd.read_csv('House_Price.csv')

# Define the features (independent variables) and the target (dependent␣
↪variable)
X = data.drop(columns='price')  # Features
y = data['price']                # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Identify categorical and numerical columns
categorical_cols = X_train.select_dtypes(include=['object']).columns
numerical_cols = X_train.select_dtypes(exclude=['object']).columns

# Create a preprocessor that handles missing values, applies one-hot encoding␣
↪to categorical features,
# and scales numerical features
preprocessor = ColumnTransformer(
    transformers=[
        ('num', Pipeline(steps=[
            ('imputer', SimpleImputer(strategy='mean')),  # Impute missing␣
↪values with mean
            ('scaler', StandardScaler())                  # Scale numerical␣
↪features
        ]), numerical_cols),
        ('cat', Pipeline(steps=[
            ('imputer', SimpleImputer(strategy='most_frequent')),  # Impute␣
↪missing values with mode
            ('onehot', OneHotEncoder())                           # One-hot␣
↪encode categorical features
```

```
        ]), categorical_cols)
    ])

# Create a pipeline that first preprocesses the data, then fits the model
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                          ('regressor', LinearRegression())])  # Add␣
 ↪regressor here

# Fit the model on the training data
pipeline.fit(X_train, y_train)

# Predict on the test data
y_pred = pipeline.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Display the evaluation metrics
print("Mean Squared Error:", mse)
print("R-squared:", r2)

# Optionally, display the first few predictions vs actual values
results = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(results.head())
```

```
Mean Squared Error: 25.954592013593974
R-squared: 0.6480459399756863
     Actual  Predicted
173    23.6  30.962751
274    32.4  32.264156
491    13.6  17.151003
72     22.8  23.397942
452    16.1  15.954123
```

[35]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder

# Load the dataset
def load_data(file_path):
    """Loads the House Price dataset from a CSV file."""
```

```python
    return pd.read_csv(file_path)

# Preprocess the data
def preprocess_data(data):
    """Replaces non-numeric values with NaN, imputes NaN, encodes categorical␣
↪variables, and splits data into features and target."""
    # Replace non-numeric values with NaN
    data = data.replace(['NO', 'yes', 'no', 'Yes', 'No'], np.nan)

    # Impute NaN values
    numeric_cols = data.select_dtypes(include=['int64', 'float64']).columns
    imputer = SimpleImputer(strategy='mean')
    data[numeric_cols] = imputer.fit_transform(data[numeric_cols])

    # Encode categorical variables
    categorical_cols = data.select_dtypes(include=['object']).columns
    encoder = LabelEncoder()
    for col in categorical_cols:
        data[col] = encoder.fit_transform(data[col])

    # Split data into features and target
    X = data.drop("price", axis=1)
    y = data["price"]
    return X, y

# Split data into training and testing sets
def split_data(X, y, test_size=0.2, random_state=42):
    """Splits data into training and testing sets."""
    return train_test_split(X, y, test_size=test_size,␣
↪random_state=random_state)

# Train a RandomForestRegressor model
def train_model(X_train, y_train, n_estimators=100, random_state=42):
    """Trains a RandomForestRegressor model."""
    model = RandomForestRegressor(n_estimators=n_estimators,␣
↪random_state=random_state)
    model.fit(X_train, y_train)
    return model

# Evaluate the model
def evaluate_model(model, X_test, y_test):
    """Makes predictions and calculates MSE, RMSE, R-squared, and MAE."""
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    rmse = mean_squared_error(y_test, y_pred, squared=False)
    r2 = r2_score(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
```

```
    return mse, rmse, r2, mae

# Main function
def main():
    # Load data
    data = load_data("House_Price.csv")

    # Preprocess data
    X, y = preprocess_data(data)

    # Split data
    X_train, X_test, y_train, y_test = split_data(X, y)

    # Train model
    model = train_model(X_train, y_train)

    # Evaluate model
    mse, rmse, r2, mae = evaluate_model(model, X_test, y_test)

    # Print results
    print("MSE:", mse)
    print("RMSE:", rmse)
    print("R-squared:", r2)
    print("MAE:", mae)

if __name__ == "__main__":
    main()
```

```
MSE: 8.799172245098037
RMSE: 2.9663398734969726
R-squared: 0.8806799045466223
MAE: 2.038245098039215
```

2.10 Retrieving Linear Regression Model Coefficients and Intercept

2.10.1 Fit pipeline model and extract model parameters

[303]:
```
data = pd.read_csv('House_Price.csv')
data
data.columns
```

[303]:
```
Index(['price', 'crime_rate', 'resid_area', 'air_qual', 'room_num', 'age',
       'dist1', 'dist2', 'dist3', 'dist4', 'teachers', 'poor_prop', 'airport',
       'n_hos_beds', 'n_hot_rooms', 'waterbody', 'rainfall', 'bus_ter',
       'parks'],
      dtype='object')
```

[9]:
```
from sklearn.linear_model import LinearRegression
regressor_f = LinearRegression()
```

[304]:
```
# Fit the pipeline model
pipeline.fit(X_train, y_train)

## Accessing Linear Regression Model from Pipeline

### Get the Linear Regression model from the pipeline
regressor_f = pipeline.named_steps['regressor']

## Model Parameters Extraction

### Get the coefficients (weights) and intercept
weights = regressor_f.coef_
bias = regressor_f.intercept_

### Print the coefficients and intercept
print("Weights (Coefficients):", weights)
print("Intercept (Bias):", bias)
```

```
Weights (Coefficients): [[-9.89405309e-01 -6.60020990e-01 -5.59406499e-01
3.78739230e+00
  -6.64501064e-01  1.39316342e-01 -1.22322905e-01 -8.80028632e-01
  -1.39427597e+00  1.60763351e+00 -2.91545503e+00  4.81946366e-01
  -7.21325746e-02  2.75498360e-01 -1.34063309e-01  2.89432376e-02
  -2.89432376e-02 -1.09540726e-01 -2.62267881e-01  3.71808608e-01
  -1.88737914e-15]]
Intercept (Bias): [22.05823776]
```

[277]:
```
for col in categorical_features:
    print(f"Unique values in {col}: {data[col].unique()}")
```

```
Unique values in crime_rate: [6.32000e-03 2.73100e-02 2.72900e-02 3.23700e-02
6.90500e-02 2.98500e-02
 8.82900e-02 1.44550e-01 2.11240e-01 1.70040e-01 2.24890e-01 1.17470e-01
 9.37800e-02 6.29760e-01 6.37960e-01 6.27390e-01 1.05393e+00 7.84200e-01
 8.02710e-01 7.25800e-01 1.25179e+00 8.52040e-01 1.23247e+00 9.88430e-01
 7.50260e-01 8.40540e-01 6.71910e-01 9.55770e-01 7.72990e-01 1.00245e+00
 1.13081e+00 1.35472e+00 1.38799e+00 1.15172e+00 1.61282e+00 6.41700e-02
 9.74400e-02 8.01400e-02 1.75050e-01 2.76300e-02 3.35900e-02 1.27440e-01
 1.41500e-01 1.59360e-01 1.22690e-01 1.71420e-01 1.88360e-01 2.29270e-01
 2.53870e-01 2.19770e-01 8.87300e-02 4.33700e-02 5.36000e-02 4.98100e-02
 1.36000e-02 1.31100e-02 2.05500e-02 1.43200e-02 1.54450e-01 1.03280e-01
 1.49320e-01 1.71710e-01 1.10270e-01 1.26500e-01 1.95100e-02 3.58400e-02
 4.37900e-02 5.78900e-02 1.35540e-01 1.28160e-01 8.82600e-02 1.58760e-01
 9.16400e-02 1.95390e-01 7.89600e-02 9.51200e-02 1.01530e-01 8.70700e-02
 5.64600e-02 8.38700e-02 4.11300e-02 4.46200e-02 3.65900e-02 3.55100e-02
```

5.05900e-02	5.73500e-02	5.18800e-02	7.15100e-02	5.66000e-02	5.30200e-02
4.68400e-02	3.93200e-02	4.20300e-02	2.87500e-02	4.29400e-02	1.22040e-01
1.15040e-01	1.20830e-01	8.18700e-02	6.86000e-02	1.48660e-01	1.14320e-01
2.28760e-01	2.11610e-01	1.39600e-01	1.32620e-01	1.71200e-01	1.31170e-01
1.28020e-01	2.63630e-01	1.07930e-01	1.00840e-01	1.23290e-01	2.22120e-01
1.42310e-01	1.71340e-01	1.31580e-01	1.50980e-01	1.30580e-01	1.44760e-01
6.89900e-02	7.16500e-02	9.29900e-02	1.50380e-01	9.84900e-02	1.69020e-01
3.87350e-01	2.59150e-01	3.25430e-01	8.81250e-01	3.40060e-01	1.19294e+00
5.90050e-01	3.29820e-01	9.76170e-01	5.57780e-01	3.22640e-01	3.52330e-01
2.49800e-01	5.44520e-01	2.90900e-01	1.62864e+00	3.32105e+00	4.09740e+00
2.77974e+00	2.37934e+00	2.15505e+00	2.36862e+00	2.33099e+00	2.73397e+00
1.65660e+00	1.49632e+00	1.12658e+00	2.14918e+00	1.41385e+00	3.53501e+00
2.44668e+00	1.22358e+00	1.34284e+00	1.42502e+00	1.27346e+00	1.46336e+00
1.83377e+00	1.51902e+00	2.24236e+00	2.92400e+00	2.01019e+00	1.80028e+00
2.30040e+00	2.44953e+00	1.20742e+00	2.31390e+00	1.39140e-01	9.17800e-02
8.44700e-02	6.66400e-02	7.02200e-02	5.42500e-02	6.64200e-02	5.78000e-02
6.58800e-02	6.88800e-02	9.10300e-02	1.00080e-01	8.30800e-02	6.04700e-02
5.60200e-02	7.87500e-02	1.25790e-01	8.37000e-02	9.06800e-02	6.91100e-02
8.66400e-02	2.18700e-02	1.43900e-02	1.38100e-02	4.01100e-02	4.66600e-02
3.76800e-02	3.15000e-02	1.77800e-02	3.44500e-02	2.17700e-02	3.51000e-02
2.00900e-02	1.36420e-01	2.29690e-01	2.51990e-01	1.35870e-01	4.35710e-01
1.74460e-01	3.75780e-01	2.17190e-01	1.40520e-01	2.89550e-01	1.98020e-01
4.56000e-02	7.01300e-02	1.10690e-01	1.14250e-01	3.58090e-01	4.07710e-01
6.23560e-01	6.14700e-01	3.15330e-01	5.26930e-01	3.82140e-01	4.12380e-01
2.98190e-01	4.41780e-01	5.37000e-01	4.62960e-01	5.75290e-01	3.31470e-01
4.47910e-01	3.30450e-01	5.20580e-01	5.11830e-01	8.24400e-02	9.25200e-02
1.13290e-01	1.06120e-01	1.02900e-01	1.27570e-01	2.06080e-01	1.91330e-01
3.39830e-01	1.96570e-01	1.64390e-01	1.90730e-01	1.40300e-01	2.14090e-01
8.22100e-02	3.68940e-01	4.81900e-02	3.54800e-02	1.53800e-02	6.11540e-01
6.63510e-01	6.56650e-01	5.40110e-01	5.34120e-01	5.20140e-01	8.25260e-01
5.50070e-01	7.61620e-01	7.85700e-01	5.78340e-01	5.40500e-01	9.06500e-02
2.99160e-01	1.62110e-01	1.14600e-01	2.21880e-01	5.64400e-02	9.60400e-02
1.04690e-01	6.12700e-02	7.97800e-02	2.10380e-01	3.57800e-02	3.70500e-02
6.12900e-02	1.50100e-02	9.06000e-03	1.09600e-02	1.96500e-02	3.87100e-02
4.59000e-02	4.29700e-02	3.50200e-02	7.88600e-02	3.61500e-02	8.26500e-02
8.19900e-02	1.29320e-01	5.37200e-02	1.41030e-01	6.46600e-02	5.56100e-02
4.41700e-02	3.53700e-02	9.26600e-02	1.00000e-01	5.51500e-02	5.47900e-02
7.50300e-02	4.93200e-02	4.92980e-01	3.49400e-01	2.63548e+00	7.90410e-01
2.61690e-01	2.69380e-01	3.69200e-01	2.53560e-01	3.18270e-01	2.45220e-01
4.02020e-01	4.75470e-01	1.67600e-01	1.81590e-01	3.51140e-01	2.83920e-01
3.41090e-01	1.91860e-01	3.03470e-01	2.41030e-01	6.61700e-02	6.72400e-02
4.54400e-02	5.02300e-02	3.46600e-02	5.08300e-02	3.73800e-02	3.96100e-02
3.42700e-02	3.04100e-02	3.30600e-02	5.49700e-02	6.15100e-02	1.30100e-02
2.49800e-02	2.54300e-02	3.04900e-02	3.11300e-02	6.16200e-02	1.87000e-02
2.89900e-02	6.21100e-02	7.95000e-02	7.24400e-02	1.70900e-02	4.30100e-02
1.06590e-01	8.98296e+00	3.84970e+00	5.20177e+00	4.26131e+00	4.54192e+00
3.83684e+00	3.67822e+00	4.22239e+00	3.47428e+00	4.55587e+00	3.69695e+00
1.35222e+01	4.89822e+00	5.66998e+00	6.53876e+00	9.23230e+00	8.26725e+00

```
 1.11081e+01 1.84982e+01 1.96091e+01 1.52880e+01 9.82349e+00 2.36482e+01
 1.78667e+01 8.89762e+01 1.58744e+01 9.18702e+00 7.99248e+00 2.00849e+01
 1.68118e+01 2.43938e+01 2.25971e+01 1.43337e+01 8.15174e+00 6.96215e+00
 5.29305e+00 1.15779e+01 8.64476e+00 1.33598e+01 8.71675e+00 5.87205e+00
 7.67202e+00 3.83518e+01 9.91655e+00 2.50461e+01 1.42362e+01 9.59571e+00
 2.48017e+01 4.15292e+01 6.79208e+01 2.07162e+01 1.19511e+01 7.40389e+00
 1.44383e+01 5.11358e+01 1.40507e+01 1.88110e+01 2.86558e+01 4.57461e+01
 1.80846e+01 1.08342e+01 2.59406e+01 7.35341e+01 1.18123e+01 1.10874e+01
 7.02259e+00 1.20482e+01 7.05042e+00 8.79212e+00 1.58603e+01 1.22472e+01
 3.76619e+01 7.36711e+00 9.33889e+00 8.49213e+00 1.00623e+01 6.44405e+00
 5.58107e+00 1.39134e+01 1.11604e+01 1.44208e+01 1.51772e+01 1.36781e+01
 9.39063e+00 2.20511e+01 9.72418e+00 5.66637e+00 9.96654e+00 1.28023e+01
 1.06718e+01 6.28807e+00 9.92485e+00 9.32909e+00 7.52601e+00 6.71772e+00
 5.44114e+00 5.09017e+00 8.24809e+00 9.51363e+00 4.75237e+00 4.66883e+00
 8.20058e+00 7.75223e+00 6.80117e+00 4.81213e+00 3.69311e+00 6.65492e+00
 5.82115e+00 7.83932e+00 3.16360e+00 3.77498e+00 4.42228e+00 1.55757e+01
 1.30751e+01 4.34879e+00 4.03841e+00 3.56868e+00 4.64689e+00 8.05579e+00
 6.39312e+00 4.87141e+00 1.50234e+01 1.02330e+01 5.82401e+00 5.70818e+00
 5.73116e+00 2.81838e+00 2.37857e+00 3.67367e+00 5.69175e+00 4.83567e+00
 1.50860e-01 1.83370e-01 2.07460e-01 1.05740e-01 1.11320e-01 1.73310e-01
 2.79570e-01 1.78990e-01 2.89600e-01 2.68380e-01 2.39120e-01 1.77830e-01
 2.24380e-01 6.26300e-02 4.52700e-02 6.07600e-02 1.09590e-01 4.74100e-02]
Unique values in resid_area: [32.31 37.07 32.18 37.87 38.14 35.96 32.95 36.91
35.64 34.   31.22 30.74
 31.32 35.13 31.38 33.37 36.07 40.81 42.83 34.86 34.49 33.41 45.04 32.89
 38.56 40.01 55.65 51.89 49.58 34.05 32.46 33.44 32.93 30.46 31.52 31.47
 32.03 32.68 40.59 43.89 36.2  34.93 35.86 33.64 33.75 33.97 36.96 36.41
 33.33 31.21 32.97 32.25 31.76 35.32 34.95 43.92 32.24 36.09 39.9  37.38
 33.24 36.06 35.19 31.89 33.78 34.39 34.15 32.01 31.25 31.69 32.02 31.91
 48.1  57.74 39.69 41.93]
Unique values in air_qual: [0.538  0.469  0.458  0.524  0.499  0.428  0.448
0.439  0.41   0.403
 0.411  0.453  0.4161 0.398  0.409  0.413  0.437  0.426  0.449  0.489
 0.464  0.445  0.52   0.547  0.581  0.624  0.871  0.605  0.51   0.488
 0.401  0.422  0.404  0.415  0.55   0.507  0.504  0.431  0.392  0.394
 0.647  0.575  0.447  0.4429 0.4    0.389  0.385  0.405  0.433  0.472
 0.544  0.493  0.46   0.4379 0.515  0.442  0.518  0.484  0.429  0.435
 0.77   0.718  0.631  0.668  0.671  0.7    0.693  0.659  0.597  0.679
 0.614  0.584  0.713  0.74   0.655  0.58   0.532  0.583  0.609  0.585
 0.573 ]
Unique values in waterbody: ['River' 'Lake' nan 'Lake and River']
```

[287]:
```
print(data.isnull().sum())
```

```
price          0
crime_rate     0
resid_area     0
air_qual       0
```

```
room_num         0
age              0
dist1            0
dist2            0
dist3            0
dist4            0
teachers         0
poor_prop        0
airport          0
n_hos_beds       8
n_hot_rooms      0
waterbody      155
rainfall         0
bus_ter          0
parks            0
dtype: int64
```

[288]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LinearRegression

# Load your dataset
data = pd.read_csv('House_Price.csv')

# Define your feature set and target variable
X = data.drop('price', axis=1)
y = data['price']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Identify numerical and categorical columns
numerical_cols = X.select_dtypes(include=['float64', 'int64']).columns.tolist()
categorical_cols = X.select_dtypes(include=['object']).columns.tolist()

# Create preprocessing steps
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),  # Impute missing values with␣
↪mean
    ('scaler', StandardScaler())                  # Scale numerical features
])
```

```python
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),  #␣
↪Impute missing values
    ('onehot', OneHotEncoder(handle_unknown='ignore'))                       #␣
↪One-hot encode categorical features
])

# Combine preprocessing steps using ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_cols),
        ('cat', categorical_transformer, categorical_cols)
    ]
)

# Create the final pipeline
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', LinearRegression())
])

# Fit the pipeline model
pipeline.fit(X_train, y_train)

# Now you can make predictions
predictions = pipeline.predict(X_test)

# Optional: Display some predictions
print(predictions)
```

```
[31.21821981 32.10671443 17.3799487  23.24028457 16.17524363 22.91338255
 16.81451415 13.89308435 21.39316776 21.20456609 22.17812783 19.44597393
 -4.93380572 24.76268886 19.23912595 25.60784098 18.17444849  3.00387244
 39.15119463 17.56933295 25.26884375 28.8163827  12.80071738 24.09806607
 16.7057208  13.68652169 21.88542471 15.86662841 19.38523976 20.4462417
 22.04046372 25.77488586 24.93557389 16.89959037 13.70229837 18.77051113
 33.67224629 19.87427217 20.66215762 26.34881445 13.55655986 30.99216366
 41.18838686 17.50178434 29.46039753 15.539965   15.06726728 27.4363002
 17.3972138  29.97641727 19.78739857 34.54257947 16.32729531 27.67538975
 40.00916382 20.796968   17.16516094 31.14039184 24.86290122 13.91938864
 20.77553137 28.95806971 32.50833764 17.7657163  20.96923827 13.66048259
 17.1037838  24.0378151  29.84206034 13.10823727 22.07397385 25.72260214
  8.92477147 16.13162728 24.97064411  4.09986949 21.84597513 38.52433158
 16.39798893 12.52251414 22.36744045 10.01418569 20.00939465  6.85618479
 22.96390873 29.22675337 19.61584926 25.72292082 28.20101695 21.2609681
 27.76942391  7.36844018 21.02757584 15.69746039 12.15345325 22.29410293
 24.2431937   1.77217659 17.0142382  18.84026696 22.36227534 24.3505833 ]
```

[283]:
```python
# Predicting on the test set
test_pred = pipeline.predict(X_test)  # Ensure you use X_test

# Display the predictions
print("Predictions on the test set:", test_pred)
```

```
Predictions on the test set: [31.21821981 32.10671443 17.3799487  23.24028457
16.17524363 22.91338255
 16.81451415 13.89308435 21.39316776 21.20456609 22.17812783 19.44597393
 -4.93380572 24.76268886 19.23912595 25.60784098 18.17444849  3.00387244
 39.15119463 17.56933295 25.26884375 28.8163827  12.80071738 24.09806607
 16.7057208  13.68652169 21.88542471 15.86662841 19.38523976 20.4462417
 22.04046372 25.77488586 24.93557389 16.89959037 13.70229837 18.77051113
 33.67224629 19.87427217 20.66215762 26.34881445 13.55655986 30.99216366
 41.18838686 17.50178434 29.46039753 15.539965   15.06726728 27.4363002
 17.3972138  29.97641727 19.78739857 34.54257947 16.32729531 27.67538975
 40.00916382 20.796968   17.16516094 31.14039184 24.86290122 13.91938864
 20.77553137 28.95806971 32.50833764 17.7657163  20.96923827 13.66048259
 17.1037838  24.0378151  29.84206034 13.10823727 22.07397385 25.72260214
  8.92477147 16.13162728 24.97064411  4.09986949 21.84597513 38.52433158
 16.39798893 12.52251414 22.36744045 10.01418569 20.00939465  6.85618479
 22.96390873 29.22675337 19.61584926 25.72292082 28.20101695 21.2609681
 27.76942391  7.36844018 21.02757584 15.69746039 12.15345325 22.29410293
 24.2431937   1.77217659 17.0142382  18.84026696 22.36227534 24.3505833 ]
```

[284]:
```python
from sklearn.model_selection import train_test_split

# Assuming your feature set and target variable are already defined
X = data.drop(columns=['price'])  # Replace with your feature columns
y = data['price']  # Your target variable

# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[285]:
```python
# Predicting on the test set using the pipeline
test_pred = pipeline.predict(X_test)

# Display the predictions
print("Predictions on the test set:", test_pred)
```

```
Predictions on the test set: [31.21821981 32.10671443 17.3799487  23.24028457
16.17524363 22.91338255
 16.81451415 13.89308435 21.39316776 21.20456609 22.17812783 19.44597393
 -4.93380572 24.76268886 19.23912595 25.60784098 18.17444849  3.00387244
 39.15119463 17.56933295 25.26884375 28.8163827  12.80071738 24.09806607
 16.7057208  13.68652169 21.88542471 15.86662841 19.38523976 20.4462417
 22.04046372 25.77488586 24.93557389 16.89959037 13.70229837 18.77051113
```

```
 33.67224629 19.87427217 20.66215762 26.34881445 13.55655986 30.99216366
 41.18838686 17.50178434 29.46039753 15.539965   15.06726728 27.4363002
 17.3972138  29.97641727 19.78739857 34.54257947 16.32729531 27.67538975
 40.00916382 20.796968   17.16516094 31.14039184 24.86290122 13.91938864
 20.77553137 28.95806971 32.50833764 17.7657163  20.96923827 13.66048259
 17.1037838  24.0378151  29.84206034 13.10823727 22.07397385 25.72260214
  8.92477147 16.13162728 24.97064411  4.09986949 21.84597513 38.52433158
 16.39798893 12.52251414 22.36744045 10.01418569 20.00939465  6.85618479
 22.96390873 29.22675337 19.61584926 25.72292082 28.20101695 21.2609681
 27.76942391  7.36844018 21.02757584 15.69746039 12.15345325 22.29410293
 24.2431937   1.77217659 17.0142382  18.84026696 22.36227534 24.3505833 ]
```

[316]:
```python
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Calculate metrics
mae = mean_absolute_error(y_test, test_pred)
mse = mean_squared_error(y_test, test_pred)
r2 = r2_score(y_test, test_pred)

# Print metrics
print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Mean Absolute Error: 3.361266381024965
Mean Squared Error: 26.040081169598608
R-squared: 0.6468866747663498
```

[ ]:
```

```

2.11 Evaluating Model Performance with R^2 Score

2.11.1 Calculate and Display R^2 Score

[315]:
```python
from sklearn.metrics import r2_score

# Calculate R² score
r2_score_value = r2_score(y_test, predictions)

# Print the R² score
print(f'R² Score: {r2_score_value}')
```

```
R² Score: 0.6049476162487047
```

[ ]:
```

```

2.12 House Price Prediction using Support Vector Regression (SVR)

2.12.1 Load dataset, split data, preprocess, train model, and evaluate

2.12.2 Calculate and Display R² Score

[339]:
```
data = pd.read_csv('House_Price.csv')
data
data.columns
```

[339]:
```
Index(['price', 'crime_rate', 'resid_area', 'air_qual', 'room_num', 'age',
       'dist1', 'dist2', 'dist3', 'dist4', 'teachers', 'poor_prop', 'airport',
       'n_hos_beds', 'n_hot_rooms', 'waterbody', 'rainfall', 'bus_ter',
       'parks'],
      dtype='object')
```

[340]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.svm import SVR
from sklearn.metrics import r2_score

# Load your dataset
data = pd.read_csv('House_Price.csv')

# Define features and target variable
X = data.drop('price', axis=1)
y = data['price']

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Define numerical and categorical features
numerical_features = X.select_dtypes(include=['float64', 'int64']).columns.
 ↪tolist()
categorical_features = X.select_dtypes(include=['object']).columns.tolist()

# Create transformers for preprocessing
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),  # Impute missing values with␣
 ↪mean
    ('scaler', StandardScaler())
])
categorical_transformer = OneHotEncoder(handle_unknown='ignore')
```

```
# Create the column transformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Create the pipeline with SVR
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', SVR(kernel='linear'))
])

# Fit the pipeline model
pipeline.fit(X_train, y_train)

# Make predictions
predictions = pipeline.predict(X_test)

# Calculate R² score
r2_score_value = r2_score(y_test, predictions)

# Print the R² score
print(f'R² Score: {r2_score_value}')
```

```
R² Score: 0.6049476162487047
```

[341]:
```
# Drop rows with missing values
data = data.dropna()

# Define features and target variable again
X = data.drop('price', axis=1)
y = data['price']

# Proceed with splitting, preprocessing, and fitting as before
```

[342]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.svm import SVR
from sklearn.metrics import r2_score

# Load your dataset
data = pd.read_csv('House_Price.csv')
```

```python
# Drop rows with missing values
data = data.dropna()

# Define features and target variable again
X = data.drop('price', axis=1)
y = data['price']

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 ↪random_state=42)

# Define numerical and categorical features
numerical_features = X.select_dtypes(include=['float64', 'int64']).columns.
 ↪tolist()
categorical_features = X.select_dtypes(include=['object']).columns.tolist()

# Create transformers for preprocessing
numerical_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())  # No imputer needed here since we dropped NaNs
])
categorical_transformer = OneHotEncoder(handle_unknown='ignore')

# Create the column transformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Create the pipeline with SVR
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', SVR(kernel='linear'))
])

# Fit the pipeline model
pipeline.fit(X_train, y_train)

# Make predictions
predictions = pipeline.predict(X_test)

# Calculate R² score
r2_score_value = r2_score(y_test, predictions)

# Print the R² score
print(f'R² Score: {r2_score_value}')
```

R² Score: 0.714447668845476

[343]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.svm import SVR
from sklearn.metrics import r2_score

# Load your dataset
data = pd.read_csv('House_Price.csv')

# Define features and target variable
X = data.drop('price', axis=1)
y = data['price']

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Define numerical and categorical features
numerical_features = X.select_dtypes(include=['float64', 'int64']).columns.
↪tolist()
categorical_features = X.select_dtypes(include=['object']).columns.tolist()

# Create transformers for preprocessing
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),  # Impute missing values with␣
↪mean
    ('scaler', StandardScaler())
])
categorical_transformer = OneHotEncoder(handle_unknown='ignore')

# Create the column transformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Create the pipeline with SVR
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', SVR(kernel='linear'))
])
```

```python
# Fit the pipeline model
pipeline.fit(X_train, y_train)

# Make predictions
predictions = pipeline.predict(X_test)

# Calculate R² score
r2_score_value = r2_score(y_test, predictions)

# Print the R² score
print(f'R² Score: {r2_score_value}')
```

```
R² Score: 0.6049476162487047
```

[ ]:

```python

```

2.13 Support Vector Regression (SVR) Model Implementation:

[13]:

```python
data = pd.read_csv('House_Price.csv')
data
data.columns
```

[13]:

```
Index(['price', 'crime_rate', 'resid_area', 'air_qual', 'room_num', 'age',
       'dist1', 'dist2', 'dist3', 'dist4', 'teachers', 'poor_prop', 'airport',
       'n_hos_beds', 'n_hot_rooms', 'waterbody', 'rainfall', 'bus_ter',
       'parks'],
      dtype='object')
```

[15]:

```python
import sys
sys.path.append(r'C:
 ↪\Users\laksh\AppData\Roaming\Python\Python312\site-packages')

import xgboost as xgb
from xgboost import XGBRegressor
```

[16]:

```python
import sys
print(sys.version)
```

```
3.11.7 | packaged by Anaconda, Inc. | (main, Dec 15 2023, 18:05:47) [MSC v.1916
64 bit (AMD64)]
```

[31]:

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor

# Load the dataset
df = pd.read_csv('House_Price.csv')
```

```
# Convert categorical columns to numeric using one-hot encoding
df = pd.get_dummies(df, columns=['airport', 'waterbody', 'bus_ter'],␣
↪drop_first=True)

# Define X (features) and y (target)
X = df[['crime_rate', 'resid_area', 'air_qual', 'room_num', 'age', 'dist1',␣
↪'dist2', 'dist3', 'dist4', 'teachers', 'poor_prop', 'n_hos_beds',␣
↪'n_hot_rooms', 'rainfall', 'parks']]
y = df['price']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Initialize and train the model
regressor = XGBRegressor(objective='reg:squarederror', colsample_bytree=0.3,␣
↪learning_rate=0.1, max_depth=5, n_estimators=10)
regressor.fit(X_train, y_train)

# Predict using the trained model
test_pred_svm = regressor.predict(X_test)
print(test_pred_svm)
```

```
[22.9376   31.123379 19.11746  23.379925 19.620003 22.590462 20.896221
 16.843922 22.475811 21.316137 22.210857 22.018387 15.649578 22.590462
 21.924305 24.082428 20.010199 16.159008 33.394577 17.979624 24.715958
 25.094517 18.5842   21.88059  17.224627 19.751705 22.551523 18.010162
 21.639828 21.59921  21.55151  24.258318 21.918741 20.363375 18.762835
 18.370613 28.683441 21.871008 21.606745 23.749174 20.03416  26.833946
 35.12516  21.86374  24.066069 20.12842  20.036257 23.749174 20.56301
 25.936686 22.766008 28.009703 20.563692 24.757507 35.539673 22.015957
 18.957102 28.457258 22.931746 21.639828 23.98998  27.535217 29.103569
 21.406525 25.533537 20.648249 16.672003 23.749174 27.824835 19.486322
 21.86374  24.765919 16.154327 22.978622 22.590462 15.437374 21.423622
 33.394577 16.084057 16.789711 22.551523 17.22335  22.926691 17.133368
 22.334497 24.302603 21.017185 23.455826 24.715958 18.73373  22.359589
 15.950306 21.049204 22.063026 19.350859 21.94733  27.08822  17.200665
 17.778051 17.14278  22.063026 23.078959]
```

[25]:

```
# Use the model pipeline to make predictions on X_test
test_pred_svm = model_pipeline.predict(X_test)

# Print the predictions
print(test_pred_svm)
```

```
[27.68774933 30.854312   15.90719269 23.85700325 16.11254699 22.15881996
 17.03078053 15.66756018 19.69453787 20.39172181 20.08774416 19.08748731
```

```
-3.79935201 23.41517722 17.63413238 24.90625101 16.80005645  4.13230138
39.10146553 15.99716999 25.54669036 28.09117407 13.3405864  23.86240486
15.40635284 12.59697841 21.48029823 17.3335918  18.10652826 19.11603281
19.8908167  24.55138026 23.95542081 13.79945847 14.85571545 17.06208832
27.17328946 19.24641213 20.32217997 25.88951636 13.64916716 29.34062791
40.59585655 17.15731683 27.02407857 14.48647737 13.97408206 26.92164516
17.19905306 28.7634184  19.9890775  33.09709388 16.6904357  26.38995098
38.32248391 20.40226511 16.12477841 29.81795038 24.99449484 14.12848065
22.80676088 29.73413448 30.11622747 16.87440088 22.42800267 12.57065449
17.69009408 24.21977343 28.88720988 13.73789871 21.72802439 23.65809754
10.67482429 16.61015981 23.70365002  5.1983618  20.59033397 38.71803079
17.11210272 11.42929477 21.49672925 11.02398276 20.41002275  6.39775143
21.06598205 27.93553727 19.79912199 24.85706514 26.43759788 19.96584781
25.76864429  7.17691164 19.19510645 16.87416018  5.44327011 20.10395502
20.70044181  1.96674689 16.96720121 18.34119973 21.95631729 24.11728425]
```

[23]:

```
### Support Vector Regression (SVR) Model Implementation

#### Step 1: Data Splitting

from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

## Step 2: Data Preprocessing

### Define numerical and categorical features
numeric_features = X_train.select_dtypes(include=['int64', 'float64']).columns
categorical_features = X_train.select_dtypes(include=['object']).columns

### Define transformers for numeric and categorical features
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),  # Impute missing values with␣
↪mean
    ('scaler', StandardScaler())  # Standardize numeric features
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),  # Impute missing␣
↪values with most frequent
```

```python
    ('onehot', OneHotEncoder(handle_unknown='ignore'))  # One-hot encode␣
 ↪categorical features
])

### Combine numeric and categorical transformers
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ]
)

## Step 3: Preprocess Data

### Fit and transform training data
X_train = preprocessor.fit_transform(X_train)

### Transform test data
X_test = preprocessor.transform(X_test)

## Step 4: Train SVR Model

### Initialize and fit SVR regressor
regressor = SVR(kernel='linear')
regressor.fit(X_train, y_train)

## Step 5: Make Predictions

### Predict on test data
test_pred_svm = regressor.predict(X_test)

### Print predictions
print(test_pred_svm)
```

```
[27.68774933 30.854312   15.90719269 23.85700325 16.11254699 22.15881996
 17.03078053 15.66756018 19.69453787 20.39172181 20.08774416 19.08748731
 -3.79935201 23.41517722 17.63413238 24.90625101 16.80005645  4.13230138
 39.10146553 15.99716999 25.54669036 28.09117407 13.3405864  23.86240486
 15.40635284 12.59697841 21.48029823 17.3335918  18.10652826 19.11603281
 19.8908167  24.55138026 23.95542081 13.79945847 14.85571545 17.06208832
 27.17328946 19.24641213 20.32217997 25.88951636 13.64916716 29.34062791
 40.59585655 17.15731683 27.02407857 14.48647737 13.97408206 26.92164516
 17.19905306 28.7634184  19.9890775  33.09709388 16.6904357  26.38995098
 38.32248391 20.40226511 16.12477841 29.81795038 24.99449484 14.12848065
 22.80676088 29.73413448 30.11622747 16.87440088 22.42800267 12.57065449
 17.69009408 24.21977343 28.88720988 13.73789871 21.72802439 23.65809754
 10.67482429 16.61015981 23.70365002  5.1983618  20.59033397 38.71803079
 17.11210272 11.42929477 21.49672925 11.02398276 20.41002275  6.39775143
```

```
21.06598205 27.93553727 19.79912199 24.85706514 26.43759788 19.96584781
25.76864429  7.17691164 19.19510645 16.87416018  5.44327011 20.10395502
20.70044181  1.96674689 16.96720121 18.34119973 21.95631729 24.11728425]
```

[24]:
```python
# Code with Pipeline:

from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split

# Step 1: Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Step 2: Separate numeric and categorical columns
numeric_features = X_train.select_dtypes(include=['int64', 'float64']).columns
categorical_features = X_train.select_dtypes(include=['object']).columns

# Step 3: Define preprocessing for numeric and categorical columns
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),  # Impute missing values with␣
 ↪mean
    ('scaler', StandardScaler())  # Standardize numeric features
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),  # Impute missing␣
 ↪values with the most frequent value
    ('onehot', OneHotEncoder(handle_unknown='ignore'))  # One-hot encode␣
 ↪categorical features
])

# Step 4: Combine the numeric and categorical transformers
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Step 5: Create a pipeline that first preprocesses the data, then applies the␣
 ↪SVR model
model_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', SVR(kernel='linear'))  # Use SVR as the regressor
```

```
])

# Step 6: Fit the pipeline to the training data
model_pipeline.fit(X_train, y_train)

# Step 7: Make predictions on the test data
test_pred_svm = model_pipeline.predict(X_test)

# Step 8: Print predictions
print(test_pred_svm)
```

```
[27.68774933 30.854312   15.90719269 23.85700325 16.11254699 22.15881996
 17.03078053 15.66756018 19.69453787 20.39172181 20.08774416 19.08748731
 -3.79935201 23.41517722 17.63413238 24.90625101 16.80005645  4.13230138
 39.10146553 15.99716999 25.54669036 28.09117407 13.3405864  23.86240486
 15.40635284 12.59697841 21.48029823 17.3335918  18.10652826 19.11603281
 19.8908167  24.55138026 23.95542081 13.79945847 14.85571545 17.06208832
 27.17328946 19.24641213 20.32217997 25.88951636 13.64916716 29.34062791
 40.59585655 17.15731683 27.02407857 14.48647737 13.97408206 26.92164516
 17.19905306 28.7634184  19.9890775  33.09709388 16.6904357  26.38995098
 38.32248391 20.40226511 16.12477841 29.81795038 24.99449484 14.12848065
 22.80676088 29.73413448 30.11622747 16.87440088 22.42800267 12.57065449
 17.69009408 24.21977343 28.88720988 13.73789871 21.72802439 23.65809754
 10.67482429 16.61015981 23.70365002  5.1983618  20.59033397 38.71803079
 17.11210272 11.42929477 21.49672925 11.02398276 20.41002275  6.39775143
 21.06598205 27.93553727 19.79912199 24.85706514 26.43759788 19.96584781
 25.76864429  7.17691164 19.19510645 16.87416018  5.44327011 20.10395502
 20.70044181  1.96674689 16.96720121 18.34119973 21.95631729 24.11728425]
```

[374]:
```
from sklearn.metrics import r2_score
r2_score_value_svm = r2_score(y_test,test_pred_svm)
print(r2_score_value_svm)
```

```
0.607232086501795
```

[ ]:

[34]:
```
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state= 0)
print(regressor)
```

```
DecisionTreeRegressor(random_state=0)
```

[4]:
```
pip install xgboost
```

```
Collecting xgboost
  Using cached xgboost-2.1.1-py3-none-win_amd64.whl.metadata (2.1 kB)
Requirement already satisfied: numpy in c:\users\laksh\anaconda3\lib\site-
packages (from xgboost) (1.26.4)
```

```
Requirement already satisfied: scipy in c:\users\laksh\anaconda3\lib\site-
packages (from xgboost) (1.11.4)
Using cached xgboost-2.1.1-py3-none-win_amd64.whl (124.9 MB)
Installing collected packages: xgboost
Successfully installed xgboost-2.1.1
Note: you may need to restart the kernel to use updated packages.
```

[11]:
```python
import xgboost
print(xgboost.__version__)
```

```
2.1.1
```

[23]:
```
pip install pipeline
```

```
Collecting pipeline
  Using cached pipeline-0.1.0-py3-none-any.whl.metadata (483 bytes)
Using cached pipeline-0.1.0-py3-none-any.whl (2.6 kB)
Installing collected packages: pipeline
Successfully installed pipeline-0.1.0
Note: you may need to restart the kernel to use updated packages.
```

[18]:
```python
import sklearn
print(sklearn.__version__)
```

```
1.2.2
```

[20]:
```
!pip show scikit-learn
```

```
Name: scikit-learn
Version: 1.2.2
Summary: A set of python modules for machine learning and data mining
Home-page: http://scikit-learn.org
Author:
Author-email:
License: new BSD
Location: C:\Users\laksh\anaconda3\Lib\site-packages
Requires: joblib, numpy, scipy, threadpoolctl
Required-by: imbalanced-learn
```

[52]:
```python
# Make predictions on the test data

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split

test_pred_dt = pipeline.predict(X_test)

# Print the predictions
print(test_pred_dt)
```

```
[22.9376   31.123379 19.11746  23.379925 19.620003 22.590462 20.896221
 16.843922 22.475811 21.316137 22.210857 22.018387 15.649578 22.590462
 21.924305 24.082428 20.010199 16.159008 33.394577 17.979624 24.715958
 25.094517 18.5842   21.88059  17.224627 19.751705 22.551523 18.010162
 21.639828 21.59921  21.55151  24.258318 21.918741 20.363375 18.762835
 18.370613 28.683441 21.871008 21.606745 23.749174 20.03416  26.833946
 35.12516  21.86374  24.066069 20.12842  20.036257 23.749174 20.56301
 25.936686 22.766008 28.009703 20.563692 24.757507 35.539673 22.015957
 18.957102 28.457258 22.931746 21.639828 23.98998  27.535217 29.103569
 21.406525 25.533537 20.648249 16.672003 23.749174 27.824835 19.486322
 21.86374  24.765919 16.154327 22.978622 22.590462 15.437374 21.423622
 33.394577 16.084057 16.789711 22.551523 17.22335  22.926691 17.133368
 22.334497 24.302603 21.017185 23.455826 24.715958 18.73373  22.359589
 15.950306 21.049204 22.063026 19.350859 21.94733  27.08822  17.200665
 17.778051 17.14278  22.063026 23.078959]
```

[53]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split

# Assuming your dataset is already loaded into a DataFrame 'df'
# Define your features (X) and target variable (y)
X = df[['crime_rate', 'resid_area', 'air_qual', 'room_num', 'age', 'dist1',␣
 ↪'dist2', 'dist3', 'dist4', 'teachers', 'poor_prop', 'n_hos_beds',␣
 ↪'n_hot_rooms', 'rainfall', 'parks']]
y = df['price']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Define the pipeline with StandardScaler and XGBRegressor
pipeline = Pipeline([
    ('scaler', StandardScaler()),  # Feature scaling
    ('regressor', XGBRegressor(objective='reg:squarederror', colsample_bytree=0.
 ↪3, learning_rate=0.1, max_depth=5, n_estimators=10))
])

# Train the pipeline model
pipeline.fit(X_train, y_train)

# Make predictions on the test data
test_pred_dt = pipeline.predict(X_test)

# Print the predictions
print(test_pred_dt)
```

```
[22.9376   31.123379 19.11746  23.379925 19.620003 22.590462 20.896221
 16.843922 22.475811 21.316137 22.210857 22.018387 15.649578 22.590462
 21.924305 24.082428 20.010199 16.159008 33.394577 17.979624 24.715958
 25.094517 18.5842   21.88059  17.224627 19.751705 22.551523 18.010162
 21.639828 21.59921  21.55151  24.258318 21.918741 20.363375 18.762835
 18.370613 28.683441 21.871008 21.606745 23.749174 20.03416  26.833946
 35.12516  21.86374  24.066069 20.12842  20.036257 23.749174 20.56301
 25.936686 22.766008 28.009703 20.563692 24.757507 35.539673 22.015957
 18.957102 28.457258 22.931746 21.639828 23.98998  27.535217 29.103569
 21.406525 25.533537 20.648249 16.672003 23.749174 27.824835 19.486322
 21.86374  24.765919 16.154327 22.978622 22.590462 15.437374 21.423622
 33.394577 16.084057 16.789711 22.551523 17.22335  22.926691 17.133368
 22.334497 24.302603 21.017185 23.455826 24.715958 18.73373  22.359589
 15.950306 21.049204 22.063026 19.350859 21.94733  27.08822  17.200665
 17.778051 17.14278  22.063026 23.078959]
```

[54]:
```
from sklearn.metrics import r2_score
r2_score_dt = r2_score(y_test,test_pred_dt)
print(r2_score_dt)
```

```
0.6297857808943232
```

[71]:
```
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators = 10, random_state = 0)
regressor.fit(X_train,y_train)
```

[71]:
```
RandomForestRegressor(n_estimators=10, random_state=0)
```

[66]:
```
# Check the available columns in the DataFrame
print(df.columns)
```

```
Index(['price', 'crime_rate', 'resid_area', 'air_qual', 'room_num', 'age',
       'dist1', 'dist2', 'dist3', 'dist4', 'teachers', 'poor_prop',
       'n_hos_beds', 'n_hot_rooms', 'rainfall', 'parks', 'airport_YES',
       'waterbody_Lake and River', 'waterbody_River'],
      dtype='object')
```

[67]:
```
# Check for NaN values in the DataFrame
print(df.isnull().sum())
```

```
price                       0
crime_rate                  0
resid_area                  0
air_qual                    0
room_num                    0
age                         0
dist1                       0
dist2                       0
dist3                       0
```

```
dist4                      0
teachers                   0
poor_prop                  0
n_hos_beds                 8
n_hot_rooms                0
rainfall                   0
parks                      0
airport_YES                0
waterbody_Lake and River   0
waterbody_River            0
dtype: int64
```

[69]:
```python
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

# Create an imputer object with the desired strategy (mean, median, etc.)
imputer = SimpleImputer(strategy='mean')

# Prepare your features and target
X = df.drop(columns=['price'])  # Drop target variable
y = df['price']

# Create a pipeline to handle imputation and regression
pipeline = Pipeline(steps=[
    ('imputer', imputer),
    ('regressor', RandomForestRegressor(n_estimators=10, random_state=0))
])

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 ↪random_state=42)

# Fit the pipeline to the training data
pipeline.fit(X_train, y_train)

# Make predictions on the test data
test_pred = pipeline.predict(X_test)

# Print the predictions
print(test_pred)
```

```
[22.28 29.84 18.08 24.31 16.69 21.57 17.51 15.67 21.75 21.2  20.02 19.34
  7.98 21.86 20.16 29.04 18.68  9.04 44.77 12.56 24.39 23.61 14.4  22.72
 12.32 14.78 21.59 14.48 18.16 21.88 19.14 22.22 28.68 21.14 14.62 15.61
 33.57 19.89 20.81 24.13 16.43 29.64 43.57 19.54 23.97 12.65 14.93 24.43
 16.9  27.23 21.58 33.03 16.57 25.74 47.58 21.96 15.41 31.25 22.82 20.68
```

```
 25.17 33.49 30.1  19.96 27.17 15.41 14.95 22.8  26.81 14.7  20.16 24.81
 10.12 22.01 21.13  6.86 21.2  44.42 10.3  13.86 20.38 13.95 21.04 11.28
 20.91 27.63 16.71 23.86 24.86 17.67 22.61  7.51 20.58 20.24 25.15 20.26
 34.58 10.32 11.39 15.83 20.37 23.12]
```

[70]:
```
# Drop rows with NaN values in the feature set
X = df.drop(columns=['price']).dropna()  # Drop missing values from features
y = df['price'].loc[X.index]  # Align y with remaining X

# Split the dataset again
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Fit the model
regressor = RandomForestRegressor(n_estimators=10, random_state=0)
regressor.fit(X_train, y_train)

# Make predictions
test_pred = regressor.predict(X_test)

# Print the predictions
print(test_pred)
```

```
[21.17 24.19 26.77 23.2  28.09 18.12 33.72 22.73 23.56 17.87 18.68 19.61
 20.98 26.97 20.99 15.28 15.44 14.22 16.14 18.81 21.99 22.04 20.71 23.12
 29.99 22.58 11.57 15.9  22.81 20.68 14.34 22.19 13.13 32.71 19.88 25.69
 17.   21.89 10.15 32.2  31.29 12.12 14.3  23.08 28.7  21.   19.39 25.16
 24.65 20.78 21.37 16.44 15.15 20.99 19.8  25.21 15.99 24.43 11.57 16.74
 15.99 26.68 14.52 22.03 47.34 18.69 18.15 20.73 13.63 24.6  21.14 15.77
 23.56 24.43 19.47 33.33 21.51 21.86 23.51 16.97 23.66 11.31 17.6  19.66
 34.63 15.82 19.13 12.99 16.39 24.04  9.26 21.67 22.18 14.67 19.65 23.85
 10.46 19.8  25.14 19.86]
```

[62]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer

# Define features and target variable
X = data.drop('price', axis=1)
y = data['price']

# Identify categorical and numerical columns
categorical_cols = X.select_dtypes(include=['object']).columns
numerical_cols = X.select_dtypes(exclude=['object']).columns
```

```python
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=0)

# Create a column transformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', Pipeline(steps=[
            ('imputer', SimpleImputer(strategy='mean')),
            ('scaler', StandardScaler())
        ]), numerical_cols),
        ('cat', Pipeline(steps=[
            ('imputer', SimpleImputer(strategy='most_frequent')),
            ('onehot', OneHotEncoder(handle_unknown='ignore'))
        ]), categorical_cols)
    ]
)

# Create a pipeline with the preprocessor and regressor
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', RandomForestRegressor(n_estimators=10, random_state=0))
])

# Fit the pipeline to the training data
pipeline.fit(X_train, y_train)

# Predict on the test set
test_pred_rf = pipeline.predict(X_test)
print(test_pred_rf)
```

```
[25.22 28.67 21.23 10.28 21.45 21.4  21.11 19.86 20.54 20.8   7.24 13.94
 13.63  8.66 47.64 34.43 21.67 33.59 25.55 21.08 24.26 21.31 18.75 24.47
 20.67 18.6  19.22 15.03 42.17 18.58 15.35 19.01 19.9  19.92 23.21 18.55
  8.63 26.78 13.91 15.38 22.24 21.76 23.1  16.2  23.33 23.   21.23 15.75
 13.58 26.65 15.85 19.34 21.91 35.92 15.83 20.82 20.1  18.71 17.84 20.98
 21.58 21.03 32.5  29.68 19.32 28.28 15.67 19.97 18.15 21.15 21.38 23.16
 26.78 30.96 27.75  9.25 40.96 21.99 24.87 20.12 25.25 17.96 23.13 41.71
 42.92 24.28 24.82 14.   27.66 13.54 19.46 12.08 22.89 31.37 21.47 20.66
  8.68 24.59 14.23 17.88 23.93 19.72]
```

[74]:
```python
test_pred_rf = pipeline.predict(X_test)
print(test_pred_rf)
```

```
[22.51 24.53 27.63 24.18 27.71 16.31 40.22 24.8  23.59 19.34 18.98 22.62
 22.01 27.98 19.62 10.32 16.71 14.4  17.34 22.25 21.05 23.79 21.2  24.13
 32.15 21.49 13.21 17.53 21.86 20.38 12.56 21.13 13.37 33.57 19.89 27.22
 18.61 24.73 10.12 35.8  31.22 12.13 14.93 22.22 27.23 21.58 18.68 29.23
```

```
 26.4  22.02 21.7  14.34 13.14 19.13 19.96 25.15 16.01 22.72 12.47 20.85
 20.14 26.81 14.7  22.35 45.4  15.58 15.83 21.19 13.66 25.04 21.88 15.6
 22.58 24.43 18.29 31.55 24.02 21.59 22.95 17.83 23.86 10.76 18.12 20.66
 34.51 15.45 20.24 11.39 17.75 25.53 10.28 21.95 21.57 14.36 19.8  24.11
  9.04 18.45 24.99 20.4 ]
```

[75]:
```
from sklearn.metrics import r2_score
r2_score_value_rf = r2_score(y_test,test_pred_rf)
print(r2_score_value_rf)
```

```
0.9463706019053043
```

[78]:
```
import xgboost as xgb
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split

# Assuming df is your DataFrame with features and target variable
X = df.drop(columns=['price'])
y = df['price']

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Initialize the regressor
regressor = XGBRegressor(objective='reg:squarederror', colsample_bytree=0.3,␣
↪learning_rate=0.1, n_estimators=100)

# Fit the model
regressor.fit(X_train, y_train)

# Make predictions on the test data
test_pred = regressor.predict(X_test)

# Print the predictions
print(test_pred)
```

```
[25.508232  35.811794  17.575161  23.396683  16.890337  22.827967
 17.524542  13.042206  20.549326  20.65457   22.013474  19.709122
 10.707963  23.89058   19.996906  23.857908  19.478094   9.949681
 41.50526   13.55879   26.375652  29.614689  13.835209  21.211405
 12.2106905 15.539087  23.85661   15.166334  20.365824  20.882078
 21.286882  22.470463  18.539637  18.549345  15.98306   15.987008
 33.660927  19.161547  22.000044  23.479023  17.021734  29.915543
 41.18992   20.424505  24.596445  13.672414  15.307308  24.659567
 17.620342  31.004166  20.120522  34.08752   16.740274  25.647547
 44.51242   21.243025  16.955538  33.085457  22.866133  20.277561
 23.540226  30.39383   35.481228  19.685946  24.823677  17.540623
```

```
 12.136053  25.055138  31.837671  17.460163  20.97758   23.400448
 10.959449  20.341093  24.434658   7.1542735 20.217506  41.69911
  8.822905  13.537165  22.537859  14.527461  21.54237    9.606614
 20.426226  27.915216  17.477404  24.474678  25.796412  16.751741
 24.54026   10.327926  20.321913  19.301117  21.227062  21.094172
 31.896328  11.408183  15.179044  17.036226  21.616165  23.873518 ]
```

[80]:

```
import xgboost as xgb
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split

# Assuming df is your DataFrame with features and target variable
X = df.drop(columns=['price'])
y = df['price']

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Initialize the XGBRegressor
regressor = XGBRegressor(objective='reg:squarederror', colsample_bytree=0.3,␣
 ↪learning_rate=0.1, max_depth=5, n_estimators=10)

# Fit the model
regressor.fit(X_train, y_train)

# Make predictions on the test data
test_pred = regressor.predict(X_test)

# Print the predictions
print(test_pred)
```

```
[24.944195 29.983273 19.46991  24.083712 20.046206 22.633327 21.067938
 18.173504 22.00724  21.719873 21.976273 21.566505 15.024009 23.297436
 21.980959 22.471266 20.50211  15.03186  31.361311 16.788229 24.258322
 26.102608 18.942165 21.525118 17.468506 19.832333 22.764723 17.873203
 22.233349 21.708641 23.979969 23.40197  18.456429 20.20459  19.924538
 18.813725 27.274855 22.057856 22.83927  23.511312 20.66002  27.882864
 33.092323 22.017876 24.756979 18.55349  19.780664 24.038622 20.093018
 26.396732 21.896437 28.38257  20.042723 24.416862 32.649708 21.575539
 18.588284 28.469849 23.372292 21.865631 23.69361  26.676098 28.299517
 21.123041 25.223333 21.020971 15.709087 23.9759   28.657387 19.86855
 22.242207 22.791534 15.434276 22.794977 22.873314 14.645867 21.976336
 31.96423  15.166639 17.616528 22.866413 22.03846  22.367762 14.871699
 22.05809  26.393255 19.980621 24.047638 24.451344 19.653084 23.041424
 16.50123  22.685364 21.955593 19.807495 22.630709 24.143896 15.933857
 18.769968 18.233263 22.752169 23.579872]
```

```
[84]: import xgboost as xgb
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split

# Assuming 'df' is your DataFrame with features and the target variable 'price'
X = df.drop(columns=['price'])
y = df['price']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Initialize the XGBRegressor
regressor = XGBRegressor(objective='reg:squarederror',
                          colsample_bytree=0.3,
                          learning_rate=0.1,
                          max_depth=5,
                          n_estimators=10)

# Fit the model on the training data
regressor.fit(X_train, y_train)

# Make predictions on the test data
test_pred = regressor.predict(X_test)

# Print the predictions
print(test_pred)
```

```
[24.944195 29.983273 19.46991  24.083712 20.046206 22.633327 21.067938
 18.173504 22.00724  21.719873 21.976273 21.566505 15.024009 23.297436
 21.980959 22.471266 20.50211  15.03186  31.361311 16.788229 24.258322
 26.102608 18.942165 21.525118 17.468506 19.832333 22.764723 17.873203
 22.233349 21.708641 23.979969 23.40197  18.456429 20.20459  19.924538
 18.813725 27.274855 22.057856 22.83927  23.511312 20.66002  27.882864
 33.092323 22.017876 24.756979 18.55349  19.780664 24.038622 20.093018
 26.396732 21.896437 28.38257  20.042723 24.416862 32.649708 21.575539
 18.588284 28.469849 23.372292 21.865631 23.69361  26.676098 28.299517
 21.123041 25.223333 21.020971 15.709087 23.9759   28.657387 19.86855
 22.242207 22.791534 15.434276 22.794977 22.873314 14.645867 21.976336
 31.96423  15.166639 17.616528 22.866413 22.03846  22.367762 14.871699
 22.05809  26.393255 19.980621 24.047638 24.451344 19.653084 23.041424
 16.50123  22.685364 21.955593 19.807495 22.630709 24.143896 15.933857
 18.769968 18.233263 22.752169 23.579872]
```

```
[141]: test_pred = regressor.predict(X_test)
print(test_pred)
```

```
[24.944195 29.983273 19.46991  24.083712 20.046206 22.633327 21.067938
```

```
 18.173504 22.00724  21.719873 21.976273 21.566505 15.024009 23.297436
 21.980959 22.471266 20.50211  15.03186  31.361311 16.788229 24.258322
 26.102608 18.942165 21.525118 17.468506 19.832333 22.764723 17.873203
 22.233349 21.708641 23.979969 23.40197  18.456429 20.20459  19.924538
 18.813725 27.274855 22.057856 22.83927  23.511312 20.66002  27.882864
 33.092323 22.017876 24.756979 18.55349  19.780664 24.038622 20.093018
 26.396732 21.896437 28.38257  20.042723 24.416862 32.649708 21.575539
 18.588284 28.469849 23.372292 21.865631 23.69361  26.676098 28.299517
 21.123041 25.223333 21.020971 15.709087 23.9759   28.657387 19.86855
 22.242207 22.791534 15.434276 22.794977 22.873314 14.645867 21.976336
 31.96423  15.166639 17.616528 22.866413 22.03846  22.367762 14.871699
 22.05809  26.393255 19.980621 24.047638 24.451344 19.653084 23.041424
 16.50123  22.685364 21.955593 19.807495 22.630709 24.143896 15.933857
 18.769968 18.233263 22.752169 23.579872]
```

[86]:

```
# Make predictions on the test data
test_pred = regressor.predict(X_test)  # Use X_test instead of x_test
print(test_pred)
```

```
[24.944195 29.983273 19.46991  24.083712 20.046206 22.633327 21.067938
 18.173504 22.00724  21.719873 21.976273 21.566505 15.024009 23.297436
 21.980959 22.471266 20.50211  15.03186  31.361311 16.788229 24.258322
 26.102608 18.942165 21.525118 17.468506 19.832333 22.764723 17.873203
 22.233349 21.708641 23.979969 23.40197  18.456429 20.20459  19.924538
 18.813725 27.274855 22.057856 22.83927  23.511312 20.66002  27.882864
 33.092323 22.017876 24.756979 18.55349  19.780664 24.038622 20.093018
 26.396732 21.896437 28.38257  20.042723 24.416862 32.649708 21.575539
 18.588284 28.469849 23.372292 21.865631 23.69361  26.676098 28.299517
 21.123041 25.223333 21.020971 15.709087 23.9759   28.657387 19.86855
 22.242207 22.791534 15.434276 22.794977 22.873314 14.645867 21.976336
 31.96423  15.166639 17.616528 22.866413 22.03846  22.367762 14.871699
 22.05809  26.393255 19.980621 24.047638 24.451344 19.653084 23.041424
 16.50123  22.685364 21.955593 19.807495 22.630709 24.143896 15.933857
 18.769968 18.233263 22.752169 23.579872]
```

[88]:

```
test_pred = pipeline.predict(X_test)
print(test_pred)
```

```
[22.28 29.84 18.08 24.31 16.69 21.57 17.51 15.67 21.75 21.2  20.02 19.34
  7.98 21.86 20.16 29.04 18.68  9.04 44.77 12.56 24.39 23.61 14.4  22.72
 12.32 14.78 21.59 14.48 18.16 21.88 19.14 22.22 28.68 21.14 14.62 15.61
 33.57 19.89 20.81 24.13 16.43 29.64 43.57 19.54 23.97 12.65 14.93 24.43
 16.9  27.23 21.58 33.03 16.57 25.74 47.58 21.96 15.41 31.25 22.82 20.68
 25.17 33.49 30.1  19.96 27.17 15.41 14.95 22.8  26.81 14.7  20.16 24.81
 10.12 22.01 21.13  6.86 21.2  44.42 10.3  13.86 20.38 13.95 21.04 11.28
 20.91 27.63 16.71 23.86 24.86 17.67 22.61  7.51 20.58 20.24 25.15 20.26
 34.58 10.32 11.39 15.83 20.37 23.12]
```

[418]:

```
test_pred = pipeline.predict(X_test)
print(test_pred)
```

```
[25.22 28.67 21.23 10.28 21.45 21.4  21.11 19.86 20.54 20.8   7.24 13.94
 13.63  8.66 47.64 34.43 21.67 33.59 25.55 21.08 24.26 21.31 18.75 24.47
 20.67 18.6  19.22 15.03 42.17 18.58 15.35 19.01 19.9  19.92 23.21 18.55
  8.63 26.78 13.91 15.38 22.24 21.76 23.1  16.2  23.33 23.   21.23 15.75
 13.58 26.65 15.85 19.34 21.91 35.92 15.83 20.82 20.1  18.71 17.84 20.98
 21.58 21.03 32.5  29.68 19.32 28.28 15.67 19.97 18.15 21.15 21.38 23.16
 26.78 30.96 27.75  9.25 40.96 21.99 24.87 20.12 25.25 17.96 23.13 41.71
 42.92 24.28 24.82 14.   27.66 13.54 19.46 12.08 22.89 31.37 21.47 20.66
  8.68 24.59 14.23 17.88 23.93 19.72]
```

[421]:

```
from sklearn.metrics import r2_score
r2_score_xg = r2_score(y_test, test_pred)
print(r2_score_xg)
```

```
0.7631867710970436
```

[118]:

```
import xgboost as xgb
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split
import pickle

# Assuming 'df' is your DataFrame with features and the target variable 'price'
X = df.drop(columns=['price'])
y = df['price']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Initialize the XGBRegressor
regressor = XGBRegressor(objective='reg:squarederror',
                          colsample_bytree=0.3,
                          learning_rate=0.1,
                          max_depth=5,
                          n_estimators=10)

# Fit the model on the training data
regressor.fit(X_train, y_train)

# Save the trained regressor model
filename = 'xgboost_model.pkl'
pickle.dump(regressor, open(filename, 'wb'))

print(f'Model saved to {filename}')
```

```
Model saved to xgboost_model.pkl
```

[119]:
```
data.columns
```

[119]:
```
Index(['price', 'crime_rate', 'resid_area', 'air_qual', 'room_num', 'age',
       'dist1', 'dist2', 'dist3', 'dist4', 'teachers', 'poor_prop', 'airport',
       'n_hos_beds', 'n_hot_rooms', 'waterbody', 'rainfall', 'bus_ter',
       'parks'],
      dtype='object')
```

[24]:
```
preinput = ([[7420,4,2,2,1,0,0,1,0,0]])
```

[123]:
```
import pandas as pd

# Load your data
data = pd.read_csv('House_Price.csv')  # or however you load your data

# Check the data types of all columns
print(data.dtypes)

# Check the data type of the 'price' column specifically
print(data['price'].dtype)

# Predefined input
preinput = [[7420, 4, 2, 2, 1, 0, 0, 1, 0, 0]]

# If you want to make predictions using your model
# Ensure that the model is already trained and named `regressor`
# test_pred = regressor.predict(preinput)
# print(test_pred)
```

```
price          float64
crime_rate     float64
resid_area     float64
air_qual       float64
room_num       float64
age            float64
dist1          float64
dist2          float64
dist3          float64
dist4          float64
teachers       float64
poor_prop      float64
airport         object
n_hos_beds     float64
n_hot_rooms    float64
waterbody       object
rainfall         int64
bus_ter         object
parks          float64
```

```
dtype: object
float64
```

[124]:
```python
# Check the data type of the 'price' column specifically
print(data['price'].dtype)
```

```
float64
```

[125]:
```python
data['price'].dtype
```

[125]:
```
dtype('float64')
```

[126]:
```python
import pandas as pd

# Load your dataset
data = pd.read_csv('House_Price.csv')  # Adjust the file path if necessary

# Check the data types of all columns
print("Data types of all columns:")
print(data.dtypes)

# Check the data type of the 'price' column specifically
print("\nData type of 'price' column:")
print(data['price'].dtype)
```

```
Data types of all columns:
price          float64
crime_rate     float64
resid_area     float64
air_qual       float64
room_num       float64
age            float64
dist1          float64
dist2          float64
dist3          float64
dist4          float64
teachers       float64
poor_prop      float64
airport         object
n_hos_beds     float64
n_hot_rooms    float64
waterbody       object
rainfall         int64
bus_ter         object
parks          float64
dtype: object

Data type of 'price' column:
float64
```

[ ]:

[105]:
```python
import pandas as pd

# Assuming you have a DataFrame called 'data'
data = pd.read_csv('House_Price.csv')  # or however you load your data

# Check the data types of all columns
print(data.dtypes)

# Check the data type of the 'price' column specifically
print(data['price'].dtype)
```

```
price          float64
crime_rate     float64
resid_area     float64
air_qual       float64
room_num       float64
age            float64
dist1          float64
dist2          float64
dist3          float64
dist4          float64
teachers       float64
poor_prop      float64
airport         object
n_hos_beds     float64
n_hot_rooms    float64
waterbody       object
rainfall         int64
bus_ter         object
parks          float64
dtype: object
float64
```

[ ]:

[52]:
```python
import pandas as pd
import pickle

# Load your data (if necessary)
data = pd.read_csv('House_Price.csv')

# Load the trained model from the saved file
with open('xgb_model.pkl', 'rb') as file:
    load_model = pickle.load(file)

# Predefined input (make sure it has 15 features)
```

```
preinput = [[7420, 4, 2, 2, 1, 0, 0, 1, 0, 0, 3.5, 20.0, 5, 5000, 1]]

# Make predictions
result = load_model.predict(preinput)
print(result)
```

```
[28.901625]
```

[6]:
```
preinput = [[7420, 4, 2, 2, 1, 0, 0, 1, 0, 0, 3.5, 20.0, 5, 5000, 1]]  # Make␣
↪sure all 15 features are provided
```

[7]:
```
import pandas as pd
import pickle

# Load your data (if necessary)
data = pd.read_csv('House_Price.csv')

# Load the trained model from the saved file
with open('xgb_model.pkl', 'rb') as file:
    load_model = pickle.load(file)

# Predefined input (make sure it has 15 features)
preinput = [[7420, 4, 2, 2, 1, 0, 0, 1, 0, 0, 3.5, 20.0, 5, 5000, 1]]

# Make predictions
result = load_model.predict(preinput)
print(result)
```

```
[28.901625]
```

[8]:
```
result = load_model.predict(preinput)
print(result)
```

```
[28.901625]
```

[9]:
```
import os

# Check if the file exists in the current directory
filename = 'your_model_filename.pkl'  # Replace with the correct file name
if os.path.exists(filename):
    with open(filename, 'rb') as file:
        load_model = pickle.load(file)

    # Predefined input
    preinput = [[7420, 4, 2, 2, 1, 0, 0, 1, 0, 0, 3.5, 20.0, 5, 5000, 1]]
    result = load_model.predict(preinput)
    print(result)
else:
    print(f"File '{filename}' not found. Please check the file name and path.")
```

```
File 'your_model_filename.pkl' not found. Please check the file name and path.
```

[10]:
```python
filename = 'path_to_your_file/xgb_model.pkl'  # Provide the full path to the␣
 ↪model file
```

[17]:
```python
import pickle

# Assuming 'regressor' is your trained model
filename = 'xgb_model.pkl'
with open(filename, 'wb') as file:
    pickle.dump(regressor, file)

print("Model saved successfully.")
```

```
Model saved successfully.
```

[27]:
```python
import pickle

# Assuming 'load_model' is your trained model (replace this with the actual␣
 ↪model name)
filename = 'xgb_model.pkl'
with open(filename, 'wb') as file:
    pickle.dump(load_model, file)

print("Model saved successfully.")
```

```
Model saved successfully.
```

[20]:
```python
import os
print("Current Directory:", os.getcwd())
```

```
Current Directory: C:\Users\laksh\Downloads\MY Jupyter Notebook Notes\House
Price Prediction using Machine Learning
```

[21]:
```python
import os
print(os.listdir())
```

```
['.ipynb_checkpoints', 'House Price Prediction.ipynb', 'House_Price.csv',
'xgboost_model.pkl', 'xgb_model.pkl']
```

[22]:
```python
# Get the features from your dataset
print(data.columns)  # This will list all the column names in your DataFrame
```

```
Index(['price', 'crime_rate', 'resid_area', 'air_qual', 'room_num', 'age',
       'dist1', 'dist2', 'dist3', 'dist4', 'teachers', 'poor_prop', 'airport',
       'n_hos_beds', 'n_hot_rooms', 'waterbody', 'rainfall', 'bus_ter',
       'parks'],
      dtype='object')
```

[ ]:
```python

```

[23]:

```
import pandas as pd
import pickle

# Step 1: Load your dataset
data = pd.read_csv('House_Price.csv')

# Step 2: Print the first few rows to confirm loading
print("Data Loaded Successfully:")
print(data.head())

# Step 3: Prepare your predefined input
# Make sure this input matches the feature structure used during training.
# Replace the following list with your actual feature values accordingly.
preinput = [[7420, 4, 2, 2, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1]]  # Adjust this to␣
 ↪fit your features

# Step 4: Load the trained model from the saved file
filename = 'xgb_model.pkl'  # Ensure this is the correct filename
try:
    with open(filename, 'rb') as file:
        load_model = pickle.load(file)
    print("Model Loaded Successfully.")
except FileNotFoundError:
    print(f"File {filename} not found. Please check the file path.")

# Step 5: Make predictions using the loaded model
try:
    result = load_model.predict(preinput)
    print("Prediction Result:")
    print(result)
except ValueError as e:
    print(f"ValueError: {e}")
```

```
Data Loaded Successfully:
   price  crime_rate  resid_area  air_qual  room_num   age  dist1  dist2  \
0   24.0     0.00632       32.31     0.538     6.575  65.2   4.35   3.81
1   21.6     0.02731       37.07     0.469     6.421  78.9   4.99   4.70
2   34.7     0.02729       37.07     0.469     7.185  61.1   5.03   4.86
3   33.4     0.03237       32.18     0.458     6.998  45.8   6.21   5.93
4   36.2     0.06905       32.18     0.458     7.147  54.2   6.16   5.86

   dist3  dist4  teachers  poor_prop airport  n_hos_beds  n_hot_rooms  \
0   4.18   4.01      24.7       4.98     YES       5.480      11.1920
1   5.12   5.06      22.2       9.14      NO       7.332      12.1728
2   5.01   4.97      22.2       4.03      NO       7.394     101.1200
3   6.16   5.96      21.3       2.94     YES       9.268      11.2672
4   6.37   5.86      21.3       5.33      NO       8.824      11.2896
```

```
  waterbody  rainfall bus_ter     parks
0     River        23     YES  0.049347
1      Lake        42     YES  0.046146
2       NaN        38     YES  0.045764
3      Lake        45     YES  0.047151
4      Lake        55     YES  0.039474
Model Loaded Successfully.
Prediction Result:
[25.21143]
```

[ ]:

[ ]:

2.14 Mean Squared Error (MSE), Root Mean Squared Error (RMSE),R-squared and Mean Absolute Error (MAE) Results:

[39]:
```
import pandas as pd

data = pd.read_csv("House_Price.csv")

# Print the column names
print(data.columns)
```

```
Index(['price', 'crime_rate', 'resid_area', 'air_qual', 'room_num', 'age',
       'dist1', 'dist2', 'dist3', 'dist4', 'teachers', 'poor_prop', 'airport',
       'n_hos_beds', 'n_hot_rooms', 'waterbody', 'rainfall', 'bus_ter',
       'parks'],
      dtype='object')
```

[40]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

# Load your dataset
data = pd.read_csv('House_Price.csv')

# Define the features (independent variables) and the target (dependent␣
 ↪variable)
X = data.drop(columns='price')  # Features
y = data['price']              # Target variable

# Split the data into training and testing sets
```

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Identify categorical and numerical columns
categorical_cols = X_train.select_dtypes(include=['object']).columns
numerical_cols = X_train.select_dtypes(exclude=['object']).columns

# Create a preprocessor that handles missing values, applies one-hot encoding␣
 ↪to categorical features,
# and scales numerical features
preprocessor = ColumnTransformer(
    transformers=[
        ('num', Pipeline(steps=[
            ('imputer', SimpleImputer(strategy='mean')),  # Impute missing␣
 ↪values with mean
            ('scaler', StandardScaler())                # Scale numerical␣
 ↪features
        ]), numerical_cols),
        ('cat', Pipeline(steps=[
            ('imputer', SimpleImputer(strategy='most_frequent')),  # Impute␣
 ↪missing values with mode
            ('onehot', OneHotEncoder(handle_unknown='ignore'))   # One-hot␣
 ↪encode categorical features
        ]), categorical_cols)
    ]
)

# Create a pipeline that first preprocesses the data, then fits the model␣
 ↪(LinearRegression)
pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('regressor',␣
 ↪LinearRegression())])

# Fit the model on the training data
pipeline.fit(X_train, y_train)

# Predict on the test data
y_pred = pipeline.predict(X_test)

# Evaluate the model performance:
# - Mean Squared Error (MSE): Measures the average squared difference between␣
 ↪predictions and actual values
mse = mean_squared_error(y_test, y_pred)

# - Root Mean Squared Error (RMSE): Square root of MSE, provides units of the␣
 ↪target variable
rmse = mean_squared_error(y_test, y_pred, squared=False)
```

```
# - R-squared: Coefficient of determination, measures the proportion of␣
 ↪variance explained by the model
r2 = r2_score(y_test, y_pred)

# - Mean Absolute Error (MAE): Measures the average absolute difference between␣
 ↪predictions and actual values
mae = mean_absolute_error(y_test, y_pred)

# Display the evaluation metrics
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R-squared:", r2)
print("Mean Absolute Error (MAE):", mae)

# Optionally, display the first few predictions vs actual values
results = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(results.head())
```

```
Mean Squared Error (MSE): 25.954592013593974
Root Mean Squared Error (RMSE): 5.09456494841257
R-squared: 0.6480459399756863
Mean Absolute Error (MAE): 3.35022741873567
     Actual  Predicted
173    23.6  30.962751
274    32.4  32.264156
491    13.6  17.151003
72     22.8  23.397942
452    16.1  15.954123
```

[ ]:

```

```

2.15 Final Result: The Price for the house details

[37]:

```
import pandas as pd

data = pd.read_csv("House_Price.csv")

# Print the column names
print(data.columns)
```

```
Index(['price', 'crime_rate', 'resid_area', 'air_qual', 'room_num', 'age',
       'dist1', 'dist2', 'dist3', 'dist4', 'teachers', 'poor_prop', 'airport',
       'n_hos_beds', 'n_hot_rooms', 'waterbody', 'rainfall', 'bus_ter',
       'parks'],
      dtype='object')
```

[3]:
```python
# Input prompts

price = float(input("Price: "))
crime_rate = float(input("Crime Rate (scale of 1-10): "))
resid_area = float(input("Residential Area (sqft): "))
air_qual = float(input("Air Quality Index: "))
room_num = int(input("Number of Rooms: "))
age = int(input("Age of Property (years): "))
dist1 = float(input("Distance to City Center (km): "))
dist2 = float(input("Distance to Public Transport (km): "))
dist3 = float(input("Distance to School (km): "))
dist4 = float(input("Distance to Hospital (km): "))
teachers = int(input("Number of Teachers in nearby School: "))
poor_prop = float(input("Proportion of Poor Families in Area (%): "))
airport = int(input("Distance to Airport (km): "))
n_hos_beds = int(input("Number of Hospital Beds in nearby Hospital: "))
n_hot_rooms = int(input("Number of Hotel Rooms in Area: "))
waterbody = int(input("Proximity to Waterbody (1-5 scale): "))
rainfall = float(input("Annual Rainfall in Area (mm): "))
bus_ter = int(input("Distance to Bus Terminal (km): "))
parks = int(input("Number of Parks in Area: "))
```

```
Price:  4000
Crime Rate (scale of 1-10):  2
Residential Area (sqft):  3
Air Quality Index:  3
Number of Rooms:  5
Age of Property (years):  25
Distance to City Center (km):  2
Distance to Public Transport (km):  3
Distance to School (km):  5
Distance to Hospital (km):  2
Number of Teachers in nearby School:  2
Proportion of Poor Families in Area (%):  2.5
Distance to Airport (km):  10
Number of Hospital Beds in nearby Hospital:  5
Number of Hotel Rooms in Area:  5
Proximity to Waterbody (1-5 scale):  4
Annual Rainfall in Area (mm):  100
Distance to Bus Terminal (km):  4
Number of Parks in Area:  2
```

[4]:
```python
import pandas as pd
import xgboost as xgb

# Load trained XGBoost model
try:
    load_model = xgb.XGBRegressor()
```

```
    load_model.load_model('xgb_model.json')
    print("Model loaded successfully")
    print(load_model)
except Exception as e:
    print("Error loading model:", str(e))

# Input prompts
inputs = {
    "Price": float,
    "Crime Rate (scale of 1-10)": float,
    "Residential Area (sqft)": float,
    "Air Quality Index": float,
    "Number of Rooms": int,
    "Age of Property (years)": int,
    "Distance to City Center (km)": float,
    "Distance to Public Transport (km)": float,
    "Distance to School (km)": float,
    "Distance to Hospital (km)": float,
    "Number of Teachers in nearby School": int,
    "Proportion of Poor Families in Area (%)": float,
    "Distance to Airport (km)": int,
    "Number of Hospital Beds in nearby Hospital": int,
    "Number of Hotel Rooms in Area": int,
    "Proximity to Waterbody (1-5 scale)": int,
    "Annual Rainfall in Area (mm)": float,
    "Distance to Bus Terminal (km)": int,
    "Number of Parks in Area": int,
}

# Get user input
data = {}
for prompt, dtype in inputs.items():
    while True:
        try:
            value = dtype(input(prompt + ": "))
            data[prompt.strip()] = [value]
            print(f"Input received for {prompt}")
            break
        except Exception as e:
            print(f"Invalid input for {prompt}. Please enter a valid {dtype.
↪__name__}.")

# Create DataFrame
try:
    df = pd.DataFrame(data)
    print("DataFrame created successfully")
    print(df.head())
```

```python
except Exception as e:
    print("Error creating DataFrame:", str(e))

# Make prediction
try:
    prediction = load_model.predict(df)
    print("Predicted Price:", prediction[0])
except Exception as e:
    print("Error making prediction:", str(e))
```

```
Error loading model: [20:10:49] C:\buildkite-agent\builds\buildkite-windows-cpu-
autoscaling-group-i-0015a694724fa8361-1\xgboost\xgboost-ci-
windows\src\common\io.cc:147: Opening xgb_model.json failed: The system cannot
find the file specified.

Price:  1000

Input received for Price

Crime Rate (scale of 1-10):  2

Input received for Crime Rate (scale of 1-10)

Residential Area (sqft):  100

Input received for Residential Area (sqft)

Air Quality Index:  0.5

Input received for Air Quality Index

Number of Rooms:  5

Input received for Number of Rooms

Age of Property (years):  25

Input received for Age of Property (years)

Distance to City Center (km):  2

Input received for Distance to City Center (km)

Distance to Public Transport (km):  3

Input received for Distance to Public Transport (km)

Distance to School (km):  4

Input received for Distance to School (km)

Distance to Hospital (km):  1

Input received for Distance to Hospital (km)

Number of Teachers in nearby School:  2

Input received for Number of Teachers in nearby School
```

```
Proportion of Poor Families in Area (%):  2.5

Input received for Proportion of Poor Families in Area (%)

Distance to Airport (km):  7

Input received for Distance to Airport (km)

Number of Hospital Beds in nearby Hospital:  5

Input received for Number of Hospital Beds in nearby Hospital

Number of Hotel Rooms in Area:  5

Input received for Number of Hotel Rooms in Area

Proximity to Waterbody (1-5 scale):  4

Input received for Proximity to Waterbody (1-5 scale)

Annual Rainfall in Area (mm):  90

Input received for Annual Rainfall in Area (mm)

Distance to Bus Terminal (km):  2

Input received for Distance to Bus Terminal (km)

Number of Parks in Area:  2

Input received for Number of Parks in Area
DataFrame created successfully
    Price  Crime Rate (scale of 1-10)  Residential Area (sqft)  \
0  1000.0                         2.0                    100.0

   Air Quality Index  Number of Rooms  Age of Property (years)  \
0                0.5                5                       25

   Distance to City Center (km)  Distance to Public Transport (km)  \
0                           2.0                                3.0

   Distance to School (km)  Distance to Hospital (km)  \
0                      4.0                        1.0

   Number of Teachers in nearby School  \
0                                    2

   Proportion of Poor Families in Area (%)  Distance to Airport (km)  \
0                                      2.5                         7

   Number of Hospital Beds in nearby Hospital  Number of Hotel Rooms in Area  \
0                                           5                              5

   Proximity to Waterbody (1-5 scale)  Annual Rainfall in Area (mm)  \
0                                   4                          90.0
```

```
   Distance to Bus Terminal (km)  Number of Parks in Area
0                              2                        2
Error making prediction: [20:13:12] C:\buildkite-agent\builds\buildkite-windows-
cpu-autoscaling-group-i-0015a694724fa8361-1\xgboost\xgboost-ci-
windows\src\learner.cc:764: Check failed: mparam_.num_feature != 0 (0 vs. 0) : 0
feature is supplied.  Are you using raw Booster interface?
```

[ ]:

```

```

[2]:

```python
import pandas as pd
import pickle

# Load your model
filename = 'xgb_model.pkl'  # Ensure this is the correct filename
with open(filename, 'rb') as file:
    load_model = pickle.load(file)

# Function to collect inputs from the user
def collect_inputs():
    inputs = {
        "Crime Rate": float(input("Enter Crime Rate (scale of 1-10): ")),
        "Residential Area (sqft)": float(input("Enter Residential Area (sqft):␣
↪")),
        "Air Quality Index": float(input("Enter Air Quality Index: ")),
        "Number of Rooms": int(input("Enter Number of Rooms: ")),
        "Age of Property (years)": int(input("Enter Age of Property (years):␣
↪")),
        "Distance to City Center (km)": float(input("Enter Distance to City␣
↪Center (km): ")),
        "Distance to Public Transport (km)": float(input("Enter Distance to␣
↪Public Transport (km): ")),
        "Distance to School (km)": float(input("Enter Distance to School (km):␣
↪")),
        "Distance to Hospital (km)": float(input("Enter Distance to Hospital␣
↪(km): ")),
        "Number of Teachers in nearby School": int(input("Enter Number of␣
↪Teachers in nearby School: ")),
        "Proportion of Poor Families in Area (%)": float(input("Enter␣
↪Proportion of Poor Families in Area (%): ")),
        "Distance to Airport (km)": int(input("Enter Distance to Airport (km):␣
↪")),
        "Number of Hospital Beds in nearby Hospital": int(input("Enter Number␣
↪of Hospital Beds in nearby Hospital: ")),
        "Number of Hotel Rooms in Area": int(input("Enter Number of Hotel Rooms␣
↪in Area: ")),
```

```
        "Proximity to Waterbody (1-5 scale)": int(input("Enter Proximity to␣
↪Waterbody (1-5 scale): ")),
    }
    return inputs

# Collect user inputs
user_inputs = collect_inputs()

# Format the inputs for the model (ensure 15 features)
preinput = [[
    user_inputs["Crime Rate"],
    user_inputs["Residential Area (sqft)"],
    user_inputs["Air Quality Index"],
    user_inputs["Number of Rooms"],
    user_inputs["Age of Property (years)"],
    user_inputs["Distance to City Center (km)"],
    user_inputs["Distance to Public Transport (km)"],
    user_inputs["Distance to School (km)"],
    user_inputs["Distance to Hospital (km)"],
    user_inputs["Number of Teachers in nearby School"],
    user_inputs["Proportion of Poor Families in Area (%)"],
    user_inputs["Distance to Airport (km)"],
    user_inputs["Number of Hospital Beds in nearby Hospital"],
    user_inputs["Number of Hotel Rooms in Area"],
    user_inputs["Proximity to Waterbody (1-5 scale)"]
]]

# Make predictions using the loaded model
try:
    result = load_model.predict(preinput)
    final_result = round(result[0], 0)  # Rounding to nearest whole number for␣
↪price
    print(f"The Price for the house details mentioned above is:␣
↪{int(final_result)}")
except Exception as e:
    print(f"Error in prediction: {e}")
```

```
Enter Crime Rate (scale of 1-10):  1
Enter Residential Area (sqft):  45
Enter Air Quality Index:  0.5
Enter Number of Rooms:  5
Enter Age of Property (years):  24
Enter Distance to City Center (km):  2
Enter Distance to Public Transport (km):  2
Enter Distance to School (km):  2
Enter Distance to Hospital (km):  2
Enter Number of Teachers in nearby School:  20
```

```
Enter Proportion of Poor Families in Area (%):  29
Enter Distance to Airport (km):  5
Enter Number of Hospital Beds in nearby Hospital:  5
Enter Number of Hotel Rooms in Area:  5
Enter Proximity to Waterbody (1-5 scale):  4

The Price for the house details mentioned above is: 18
```

[ ]:

```

```

[1]:

```python
import pandas as pd
import pickle

# Load your model
filename = 'xgb_model.pkl'  # Ensure this is the correct filename
with open(filename, 'rb') as file:
    load_model = pickle.load(file)

# Function to collect inputs from the user
def collect_inputs():
    inputs = {
        "Crime Rate": float(input("Enter Crime Rate (scale of 1-10): ")),
        "Residential Area (sqft)": float(input("Enter Residential Area (sqft):␣
 ↪")),
        "Air Quality Index": float(input("Enter Air Quality Index: ")),
        "Number of Rooms": int(input("Enter Number of Rooms: ")),
        "Age of Property (years)": int(input("Enter Age of Property (years):␣
 ↪")),
        "Distance to City Center (km)": float(input("Enter Distance to City␣
 ↪Center (km): ")),
        "Distance to Public Transport (km)": float(input("Enter Distance to␣
 ↪Public Transport (km): ")),
        "Distance to School (km)": float(input("Enter Distance to School (km):␣
 ↪")),
        "Distance to Hospital (km)": float(input("Enter Distance to Hospital␣
 ↪(km): ")),
        "Number of Teachers in nearby School": int(input("Enter Number of␣
 ↪Teachers in nearby School: ")),
        "Proportion of Poor Families in Area (%)": float(input("Enter␣
 ↪Proportion of Poor Families in Area (%): ")),
        "Distance to Airport (km)": int(input("Enter Distance to Airport (km):␣
 ↪")),
        "Number of Hospital Beds in nearby Hospital": int(input("Enter Number␣
 ↪of Hospital Beds in nearby Hospital: ")),
        "Number of Hotel Rooms in Area": int(input("Enter Number of Hotel Rooms␣
 ↪in Area: ")),
```

```python
        "Proximity to Waterbody (1-5 scale)": int(input("Enter Proximity to␣
↪Waterbody (1-5 scale): ")),
    }
    return inputs

# Collect user inputs
user_inputs = collect_inputs()

# Format the inputs for the model (ensure 15 features)
preinput = [[
    user_inputs["Crime Rate"],
    user_inputs["Residential Area (sqft)"],
    user_inputs["Air Quality Index"],
    user_inputs["Number of Rooms"],
    user_inputs["Age of Property (years)"],
    user_inputs["Distance to City Center (km)"],
    user_inputs["Distance to Public Transport (km)"],
    user_inputs["Distance to School (km)"],
    user_inputs["Distance to Hospital (km)"],
    user_inputs["Number of Teachers in nearby School"],
    user_inputs["Proportion of Poor Families in Area (%)"],
    user_inputs["Distance to Airport (km)"],
    user_inputs["Number of Hospital Beds in nearby Hospital"],
    user_inputs["Number of Hotel Rooms in Area"],
    user_inputs["Proximity to Waterbody (1-5 scale)"]
]]

# Make predictions using the loaded model
try:
    result = load_model.predict(preinput)
    final_result = round(result[0], 0)  # Rounding to nearest whole number for␣
↪price
    print(f"The Price for the house details mentioned above is:␣
↪{int(final_result)}")
except Exception as e:
    print(f"Error in prediction: {e}")
```

```
Enter Crime Rate (scale of 1-10):  3
Enter Residential Area (sqft):  50
Enter Air Quality Index:  0.534
Enter Number of Rooms:  10
Enter Age of Property (years):  24
Enter Distance to City Center (km):  1.5
Enter Distance to Public Transport (km):  2
Enter Distance to School (km):  1
Enter Distance to Hospital (km):  2
Enter Number of Teachers in nearby School:  25
```

```
Enter Proportion of Poor Families in Area (%):  24
Enter Distance to Airport (km):  10
Enter Number of Hospital Beds in nearby Hospital:  15
Enter Number of Hotel Rooms in Area:  12
Enter Proximity to Waterbody (1-5 scale):  5

The Price for the house details mentioned above is: 20
```

[ ]:

[ ]: