```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import fetch_california_housing
housing = fetch_california_housing(as_frame = True)
print(housing)
```

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

       Longitude
0        -122.23
1        -122.22
2        -122.24
3        -122.25
4        -122.25
...          ...
20635    -121.09
20636    -121.21
20637    -121.22
20638    -121.32
20639    -121.24

[20640 rows x 8 columns], 'target': 0        4.526
1        3.585
2        3.521
```

```
3        3.413
4        3.422
         ...
20635    0.781
20636    0.771
20637    0.923
20638    0.847
20639    0.894
Name: MedHouseVal, Length: 20640, dtype: float64, 'frame':
MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude
\
0      8.3252      41.0  6.984127   1.023810       322.0  2.555556
37.88
1      8.3014      21.0  6.238137   0.971880      2401.0  2.109842
37.86
2      7.2574      52.0  8.288136   1.073446       496.0  2.802260
37.85
3      5.6431      52.0  5.817352   1.073059       558.0  2.547945
37.85
4      3.8462      52.0  6.281853   1.081081       565.0  2.181467
37.85
...       ...       ...       ...        ...         ...       ...
...
20635  1.5603      25.0  5.045455   1.133333       845.0  2.560606
39.48
20636  2.5568      18.0  6.114035   1.315789       356.0  3.122807
39.49
20637  1.7000      17.0  5.205543   1.120092      1007.0  2.325635
39.43
20638  1.8672      18.0  5.329513   1.171920       741.0  2.123209
39.43
20639  2.3886      16.0  5.254717   1.162264      1387.0  2.616981
39.37

       Longitude  MedHouseVal
0        -122.23        4.526
1        -122.22        3.585
2        -122.24        3.521
3        -122.25        3.413
4        -122.25        3.422
...          ...          ...
20635    -121.09        0.781
20636    -121.21        0.771
20637    -121.22        0.923
20638    -121.32        0.847
20639    -121.24        0.894

[20640 rows x 9 columns], 'target_names': ['MedHouseVal'],
'feature_names': ['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms',
```

```
'Population', 'AveOccup', 'Latitude', 'Longitude'], 'DESCR': '..
_california_housing_dataset:\n\nCalifornia Housing dataset\
n--------------------------\n\n**Data Set Characteristics:**\n\
n    :Number of Instances: 20640\n\n    :Number of Attributes: 8
numeric, predictive attributes and the target\n\n    :Attribute
Information:\n        - MedInc        median income in block group\n
- HouseAge      median house age in block group\n        - AveRooms
average number of rooms per household\n        - AveBedrms     average
number of bedrooms per household\n        - Population    block group
population\n        - AveOccup      average number of household
members\n        - Latitude      block group latitude\n        -
Longitude     block group longitude\n\n    :Missing Attribute Values:
None\n\nThis dataset was obtained from the StatLib repository.\
nhttps://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html\n\nThe
target variable is the median house value for California districts,\
nexpressed in hundreds of thousands of dollars ($100,000).\n\nThis
dataset was derived from the 1990 U.S. census, using one row per
census\nblock group. A block group is the smallest geographical unit
for which the U.S.\nCensus Bureau publishes sample data (a block group
typically has a population\nof 600 to 3,000 people).\n\nA household is
a group of people residing within a home. Since the average\nnumber of
rooms and bedrooms in this dataset are provided per household, these\
ncolumns may take surprisingly large values for block groups with few
households\nand many empty houses, such as vacation resorts.\n\nIt can
be downloaded/loaded using the\
n:func:`sklearn.datasets.fetch_california_housing` function.\n\n..
topic:: References\n\n    - Pace, R. Kelley and Ronald Barry, Sparse
Spatial Autoregressions,\n      Statistics and Probability Letters, 33
(1997) 291-297\n'}
```

```
housing['data'].head()
```

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

   Longitude
0    -122.23
1    -122.22
2    -122.24
```

```
3    -122.25
4    -122.25
```

```
housing['target'].head()
```

```
0    4.526
1    3.585
2    3.521
3    3.413
4    3.422
Name: MedHouseVal, dtype: float64
```

```
df = pd.DataFrame(housing['data'])
df
```

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

       Longitude
0        -122.23
1        -122.22
2        -122.24
3        -122.25
4        -122.25
...          ...
20635    -121.09
20636    -121.21
20637    -121.22
```

```
20638    -121.32
20639    -121.24

[20640 rows x 8 columns]

df['Price'] = housing['target']
df

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

       Longitude  Price
0        -122.23  4.526
1        -122.22  3.585
2        -122.24  3.521
3        -122.25  3.413
4        -122.25  3.422
...          ...    ...
20635    -121.09  0.781
20636    -121.21  0.771
20637    -121.22  0.923
20638    -121.32  0.847
20639    -121.24  0.894

[20640 rows x 9 columns]

df.info()
```

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

```
df.isna().sum()
```

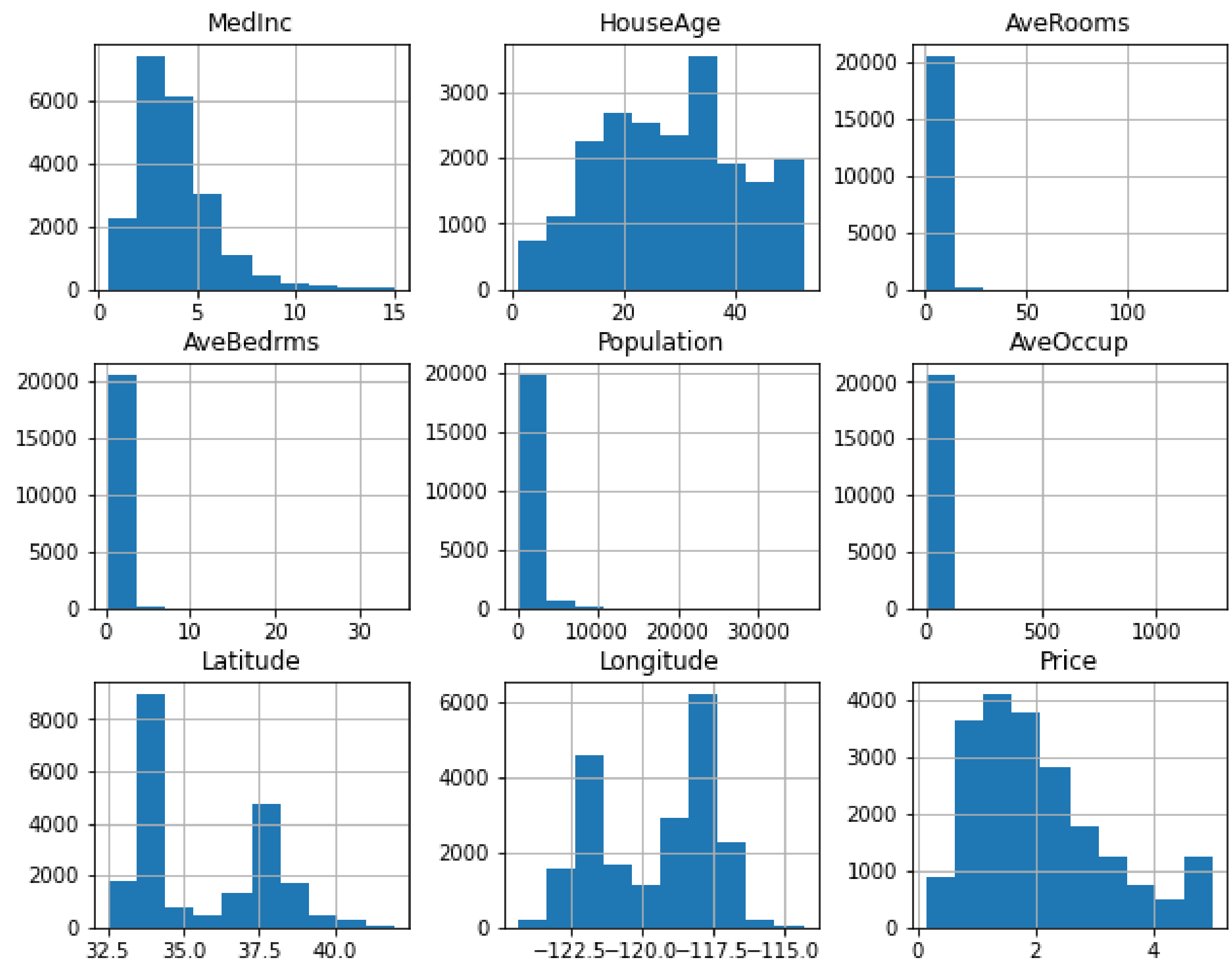
```
MedInc        0
HouseAge      0
AveRooms      0
AveBedrms     0
Population    0
AveOccup      0
Latitude      0
Longitude     0
Price         0
dtype: int64
```

```
df.describe().T
```

```
              count         mean          std         min          25%  \
MedInc      20640.0     3.870671     1.899822    0.499900     2.563400   

HouseAge    20640.0    28.639486    12.585558    1.000000    18.000000   

AveRooms    20640.0     5.429000     2.474173    0.846154     4.440716   

AveBedrms   20640.0     1.096675     0.473911    0.333333     1.006079   

Population  20640.0  1425.476744  1132.462122    3.000000   787.000000   

AveOccup    20640.0     3.070655    10.386050    0.692308     2.429741   

Latitude    20640.0    35.631861     2.135952   32.540000    33.930000   

Longitude   20640.0  -119.569704     2.003532 -124.350000  -121.800000   

Price       20640.0     2.068558     1.153956    0.149990     1.196000   
```

	50%	75%	max
MedInc	3.534800	4.743250	15.000100
HouseAge	29.000000	37.000000	52.000000
AveRooms	5.229129	6.052381	141.909091
AveBedrms	1.048780	1.099526	34.066667
Population	1166.000000	1725.000000	35682.000000
AveOccup	2.818116	3.282261	1243.333333
Latitude	34.260000	37.710000	41.950000
Longitude	-118.490000	-118.010000	-114.310000
Price	1.797000	2.647250	5.000010

```
df.hist(figsize=(10,8))
plt.show()
```

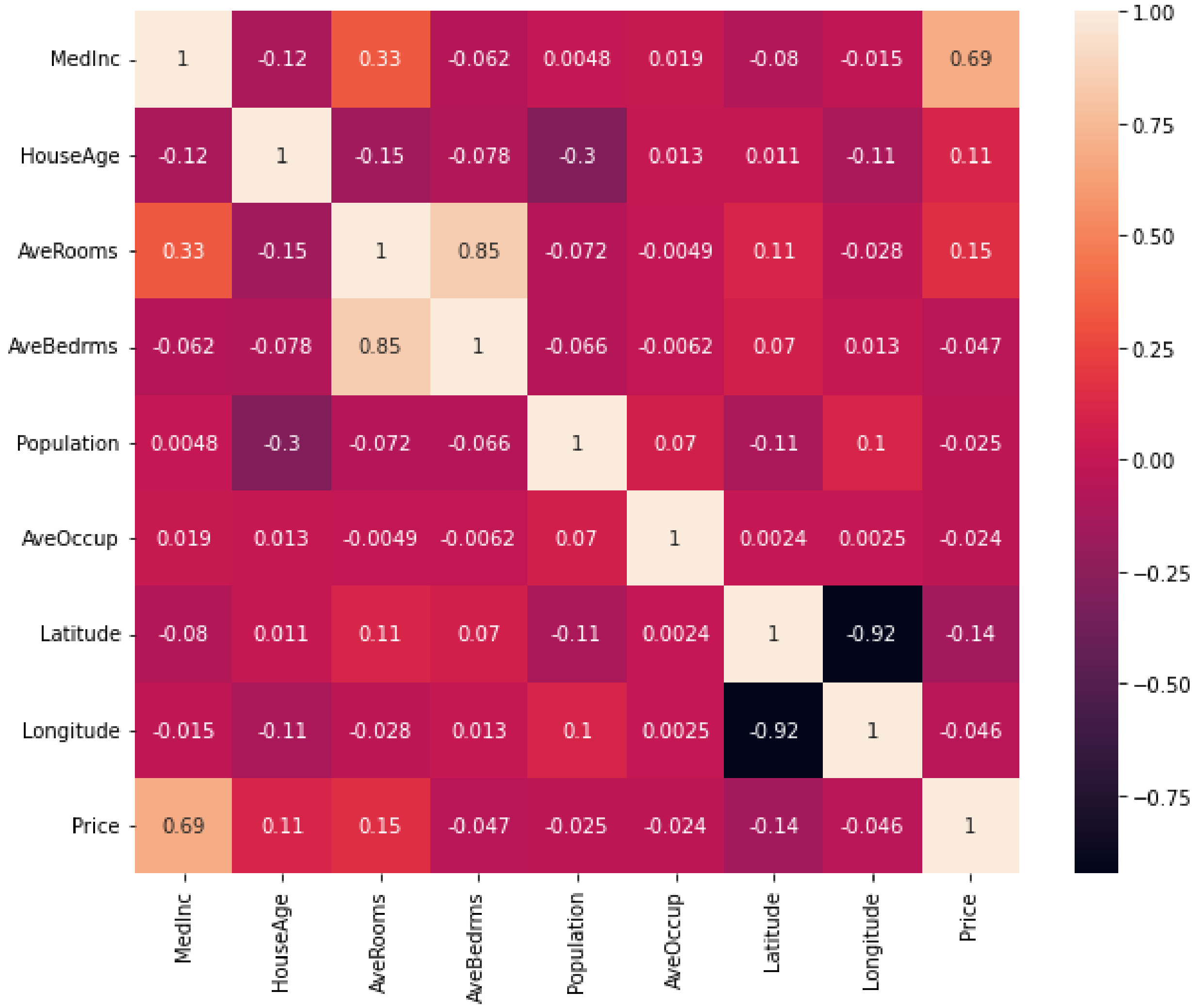



```
corr = df.corr()
corr
```

```
              MedInc  HouseAge  AveRooms  AveBedrms  Population  
AveOccup  \
MedInc      1.000000 -0.119034  0.326895  -0.062040    0.004834  
0.018766   
HouseAge   -0.119034  1.000000 -0.153277  -0.077747   -0.296244  
0.013191   
AveRooms    0.326895 -0.153277  1.000000   0.847621   -0.072213 -
0.004852   
AveBedrms  -0.062040 -0.077747  0.847621   1.000000   -0.066197 -
0.006181   
Population  0.004834 -0.296244 -0.072213  -0.066197    1.000000  
0.069863   
AveOccup    0.018766  0.013191 -0.004852  -0.006181    0.069863  
1.000000   
Latitude   -0.079809  0.011173  0.106389   0.069721   -0.108785  
0.002366   
Longitude  -0.015176 -0.108197 -0.027540   0.013344    0.099773  
0.002476   
Price       0.688075  0.105623  0.151948  -0.046701   -0.024650 -
0.023737   

            Latitude  Longitude     Price  
MedInc     -0.079809  -0.015176  0.688075  
HouseAge    0.011173  -0.108197  0.105623  
AveRooms    0.106389  -0.027540  0.151948  
AveBedrms   0.069721   0.013344 -0.046701  
Population -0.108785   0.099773 -0.024650  
AveOccup    0.002366   0.002476 -0.023737  
Latitude    1.000000  -0.924664 -0.144160  
Longitude  -0.924664   1.000000 -0.045967  
Price      -0.144160  -0.045967  1.000000  
```

```
plt.figure(figsize=(10,8))
sns.heatmap(corr,annot=True)
```

```
<AxesSubplot:>
```




```
X = df.drop('Price',axis=1)
y = df['Price']

X
```

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

```
...
20635  1.5603      25.0  5.045455   1.133333       845.0  2.560606
39.48
20636  2.5568      18.0  6.114035   1.315789       356.0  3.122807
39.49
20637  1.7000      17.0  5.205543   1.120092      1007.0  2.325635
39.43
20638  1.8672      18.0  5.329513   1.171920       741.0  2.123209
39.43
20639  2.3886      16.0  5.254717   1.162264      1387.0  2.616981
39.37

       Longitude
0        -122.23
1        -122.22
2        -122.24
3        -122.25
4        -122.25
...          ...
20635    -121.09
20636    -121.21
20637    -121.22
20638    -121.32
20639    -121.24

[20640 rows x 8 columns]
```

```
y
```

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

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size =
0.25,random_state = 42)
```

```
df.shape
```

```
(20640, 9)
```

```
X_train
```

```
       MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup
Latitude  \
8158   4.2143      37.0  5.288235   0.973529       860.0  2.529412
33.81
18368  5.3468      42.0  6.364322   1.087940       957.0  2.404523
37.16
19197  3.9191      36.0  6.110063   1.059748       711.0  2.235849
38.45
3746   6.3703      32.0  6.000000   0.990196      1159.0  2.272549
34.16
13073  2.3684      17.0  4.795858   1.035503       706.0  2.088757
38.57
...       ...       ...       ...        ...         ...       ...
...
11284  6.3700      35.0  6.129032   0.926267       658.0  3.032258
33.78
11964  3.0500      33.0  6.868597   1.269488      1753.0  3.904232
34.02
5390   2.9344      36.0  3.986717   1.079696      1756.0  3.332068
34.03
860    5.7192      15.0  6.395349   1.067979      1777.0  3.178891
37.58
15795  2.5755      52.0  3.402576   1.058776      2619.0  2.108696
37.77

       Longitude
8158     -118.12
18368    -121.98
19197    -122.69
3746     -118.41
13073    -121.33
...          ...
11284    -117.96
11964    -117.43
5390     -118.38
860      -121.96
15795    -122.42

[15480 rows x 8 columns]

X_test

       MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup
Latitude  \
20046  1.6812      25.0  4.192201   1.022284      1392.0  3.877437
36.06
3024   2.5313      30.0  5.039384   1.193493      1565.0  2.679795
35.14
15663  3.4801      52.0  3.977155   1.185877      1310.0  1.360332
37.80
```

```
20484  5.7376      17.0  6.163636   1.020202      1705.0  3.444444
34.28
9814   3.7250      34.0  5.492991   1.028037      1063.0  2.483645
36.62
...       ...       ...       ...        ...          ...       ...
...
5363   6.6260      51.0  5.532213   0.974790       771.0  2.159664
34.04
19755  2.1898      30.0  4.509091   0.945455       410.0  2.484848
40.18
4885   2.1667      37.0  3.272152   1.056962      2173.0  4.584388
34.02
13043  6.8869       6.0  7.382385   1.030075      2354.0  2.528464
38.51
8583   6.6321      36.0  5.734644   1.056511      1033.0  2.538084
33.89

       Longitude
20046    -119.01
3024     -119.46
15663    -122.44
20484    -118.72
9814     -121.93
...          ...
5363     -118.42
19755    -122.21
4885     -118.26
13043    -121.06
8583     -118.40

[5160 rows x 8 columns]
```

```
y_train.shape
```

```
(15480,)
```

```
y_test.shape
```

```
(5160,)
```

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train,y_train)
```

```
LinearRegression()
```

```
y_pred = model.predict(X_test)
y_pred
```

```
array([0.72412832, 1.76677807, 2.71151581, ..., 1.72382152,
2.34689276,
       3.52917352])
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
print(mean_squared_error(y_pred,y_test))
```

```
0.541128747847068
```

```
print(mean_absolute_error(y_pred,y_test))
```

```
0.529696401291945
```

```
r2_score(y_test,y_pred)
```

```
0.5910509795491358
```

Make Prediction

```
new_data = [[7.2574,52.0,8.288136,1.073446,496.0,2.802260,37.85,-
122.24]]
model.predict(new_data)
```

```
C:\Users\piyus\anaconda3\lib\site-packages\sklearn\base.py:464:
UserWarning: X does not have valid feature names, but LinearRegression
was fitted with feature names
  warnings.warn(
```

```
array([3.66368929])
```

MSE

1. Mean Squared Error (MSE): MSE is a measure of the average squared difference between the predicted values and the actual values. It gives more weight to larger errors. The formula for MSE is:

   MSE = (1/n) * Σ(yi - ŷi)^2

   Where:

   - n is the number of data points.
   - yi represents the actual values.
   - ŷi represents the predicted values.

   Let's illustrate this with an example:

   Actual values (yi): [2, 3, 5, 7, 10] Predicted values (ŷi): [1.5, 2.5, 4.5, 7.5, 9.5]

   MSE = (1/5) * [(2 - 1.5)^2 + (3 - 2.5)^2 + (5 - 4.5)^2 + (7 - 7.5)^2 + (10 - 9.5)^2]
   MSE = (1/5) * [0.25 + 0.25 + 0.25 + 0.25 + 0.25] MSE = 0.25

```
import numpy as np
y_actual = np.array([2, 3, 5, 7, 10])
y_pred = np.array([1.5, 2.5, 4.5, 7.5, 9.5])
print(mean_squared_error(y_actual,y_pred))

0.25
```

MAE

1. Mean Absolute Error (MAE): MAE is a measure of the average absolute difference between the predicted values and the actual values. It gives equal weight to all errors. The formula for MAE is:

   MAE = (1/n) * Σ|yi - ŷi|

   Using the same example:

   MAE = (1/5) * [|2 - 1.5| + |3 - 2.5| + |5 - 4.5| + |7 - 7.5| + |10 - 9.5|] MAE = (1/5) * [0.5 + 0.5 + 0.5 + 0.5 + 0.5] MAE = 0.5

```
print(mean_absolute_error(y_actual,y_pred))

0.5
```

R-squared (R^2) Score

1. R-squared (R^2) Score: R-squared is a measure of how well the regression model fits the data. It represents the proportion of the variance in the dependent variable (y) that is explained by the independent variables (X). The formula for R^2 is:

   R^2 = 1 - (MSE(model) / MSE(mean))

   Where:

   - MSE(model) is the mean squared error of the model.
   - MSE(mean) is the mean squared error of the mean of the actual values.

   In this example, we already calculated MSE as 0.25. Let's assume the mean of the actual values is 5:

   MSE(mean) = (1/5) * [(2 - 5)^2 + (3 - 5)^2 + (5 - 5)^2 + (7 - 5)^2 + (10 - 5)^2] MSE(mean) = (1/5) * [9 + 4 + 0 + 4 + 25] MSE(mean) = 8.4

   Now, calculate R^2:

   R^2 = 1 - (0.25 / 8.4) R^2 ≈ 0.9702 An R^2 score close to 1 indicates that the model explains a high proportion of the variance in the data, while lower values suggest that the model does not fit the data well. In this example, the R^2 score of approximately 0.9702 indicates a good fit between the model and the data.

```
r2_score(y_actual,y_pred)
```

```
0.9696601941747572
```