House Price Prediction

In [17]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount,
call drive.mount("/content/drive", force_remount=True).
```

Load Dataset

Dataset: Boston housing dataset UCI ML repository

In [18]:
```
import pandas as pd
housing = pd.read_csv("/content/drive/MyDrive/ML LAB/Project/data.csv")
```

In [19]:
```
housing.head()
```

Out[19]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	L
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	

In [ ]:
```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    int64  
 4   NOX      506 non-null    float64
 5   RM       501 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64  
 9   TAX      506 non-null    int64  
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(11), int64(3)
memory usage: 55.5 KB
```

In [ ]:
```
housing['CHAS'].value_counts()
```

Out[7]:
```
0    471
1     35
Name: CHAS, dtype: int64
```

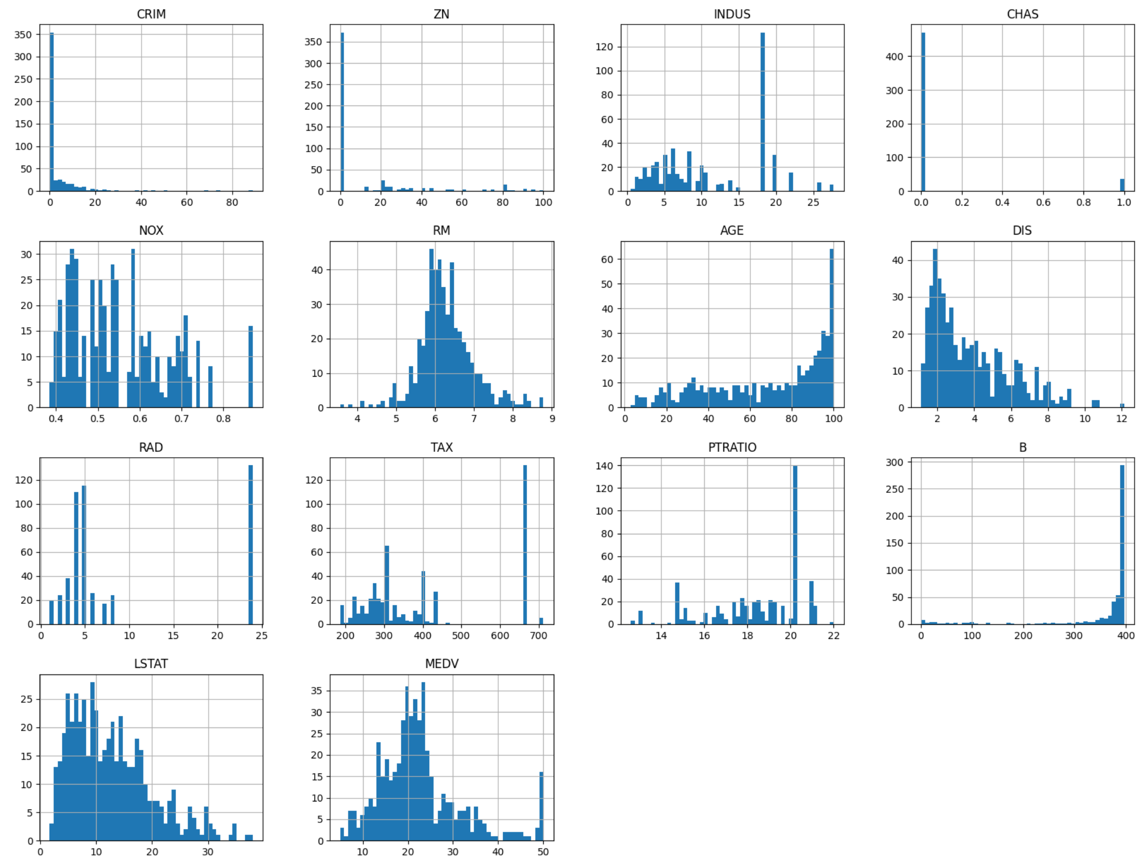
In [ ]:
```
housing.describe()
```

Out[8]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE
count	506.000000	506.000000	506.000000	506.000000	506.000000	501.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284341	68.574901
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.705587	28.148861
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.884000	45.025000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208000	77.500000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.625000	94.075000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000

In [ ]:
```
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
```

Out[12]:
```
array([[<Axes: title={'center': 'CRIM'}>, <Axes: title={'center': 'ZN'}>,
        <Axes: title={'center': 'INDUS'}>,
        <Axes: title={'center': 'CHAS'}>],
       [<Axes: title={'center': 'NOX'}>, <Axes: title={'center': 'RM'}>,
        <Axes: title={'center': 'AGE'}>, <Axes: title={'center': 'DIS'}
>],
       [<Axes: title={'center': 'RAD'}>, <Axes: title={'center': 'TAX'}>,
        <Axes: title={'center': 'PTRATIO'}>,
        <Axes: title={'center': 'B'}>],
       [<Axes: title={'center': 'LSTAT'}>,
        <Axes: title={'center': 'MEDV'}>, <Axes: >, <Axes: >]],
      dtype=object)
```



Train Test Split

In [20]:
```
import numpy as np
def split_train_test(data, test_ratio):
    np.random.seed(42)
    shuffled = np.random.permutation(len(data))
    print(shuffled)
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled[:test_set_size]
    train_indices = shuffled[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]
```

In [21]:
```
train_set, test_set = split_train_test(housing, 0.2)
```

```
[173 274 491  72 452  76 316 140 471 500 218   9 414  78 323 473 124 388
 195 448 271 278  30 501 421 474  79 454 210 497 172 320 375 362 467 153
   2 336 208  73 496 307 204  68  90 390  33  70 470   0  11 281  22 101
 268 485 442 290  84 245  63  55 229  18 351 209 395  82  39 456  46 481
 444 355  77 398 104 203 381 489  69 408 255 392 312 234 460 324  93 137
 176 417 131 346 365 132 371 412 436 411  86  75 477  15 332 423  19 325
 335  56 437 409 334 181 227 434 180  25 493 238 244 250 418 117  42 322
 347 182 155 280 126 329  31 113 148 432 338  57 194  24  17 298  66 211
 404  94 154 441  23 225 433 447   5 116  45  16 468 360   3 405 185  60
 110 321 265  29 262 478  26   7 492 108  37 157 472 118 114 175 192 272
 144 373 383 356 277 220 450 141 369  67 361 168 499 394 400 193 249 109
 420 145  92 152 222 304  83 248 165 163 199 231  74 311 455 253 119 284
 302 483 357 403 228 261 237 386 476  36 196 139 368 247 287 378  59 111
  89 266   6 364 503 341 158 150 177 397 184 318  10 384 103  81  38 317
 167 475 299 296 198 377 146 396 147 428 289 123 490  96 143 239 275  97
 353 122 183 202 246 484 301 354 410 399 286 125 305 223 422 219 129 424
 291 331 380 480 358 297 294 370 438 112 179 310 342 333 487 457 233 314
 164 136 197 258 232 115 120 352 224 406 340 127 285 415 107 374 449 133
 367  44 495  65 283  85 242 186 425 159  12  35  28 170 142 402 349 221
  95  51 240 376 382 178  41 440 391 206 282 254 416   4 256 453 100 226
 431 213 426 171  98 292 215  61  47  32 267 327 200 451  27 393 230 260
 288 162 429 138  62 135 128 482   8 326 469  64 300  14 156  40 379 465
 407 216 279 439 504 337 236 207 212 295 462 251 494 464 303 350 269 201
 161  43 217 401 190 309 259 105  53 389   1 446 488  49 419  80 205  34
 430 263 427 366  91 339 479  52 345 264 241  13 315  88 387 273 166 328
 498 134 306 486 319 243  54 363  50 461 174 445 189 502 463 187 169  58
  48 344 235 252  21 313 459 160 276 443 191 385 293 413 343 257 308 149
 130 151 359  99 372  87 458 330 214 466 121 505  20 188  71 106 270 348
 435 102]
```

In [ ]:
```
print(f"Rows in train set: {len(train_set)}\nRows in test set: {len(test_se
```

```
Rows in train set: 405
Rows in test set: 101
```

Stratifies Split on CHAS Column

In [22]:
```
import pandas as pd
import numpy as np

# Assuming you have a DataFrame 'housing' with a 'CHAS' column
# and you want to split it into train and test sets stratified by the 'CHAS

# Define your test_size and random_state
test_size = 0.2
random_state = 42

# Create an empty DataFrame for stratified train and test sets
strat_train_set = pd.DataFrame()
strat_test_set = pd.DataFrame()

# Group the data by the 'CHAS' column
groups = housing.groupby('CHAS')

# Iterate over the groups
for group_name, group_data in groups:
    # Calculate the number of samples to include in the test set
    num_samples = int(len(group_data) * test_size)

    # Use random_state to ensure reproducibility
    np.random.seed(random_state)

    # Randomly shuffle the indices of the group
    shuffled_indices = np.random.permutation(len(group_data))

    # Select the first 'num_samples' indices for the test set
    test_indices = shuffled_indices[:num_samples]

    # Select the remaining indices for the train set
    train_indices = shuffled_indices[num_samples:]

    # Add the selected rows to the stratified train and test sets
    strat_test_set = pd.concat([strat_test_set, group_data.iloc[test_indice
    strat_train_set = pd.concat([strat_train_set, group_data.iloc[train_inc
```

In [23]:
```
import pandas as pd

def custom_value_counts(data, column_name):
    if column_name not in data.columns:
        raise ValueError(f"Column '{column_name}' not found in the DataFran

    counts = data[column_name].value_counts()
    return counts

# Example usage:
# Assuming you have a DataFrame 'strat_test_set' and you want to get the vc
value_counts_result = custom_value_counts(strat_test_set, 'CHAS')
print(value_counts_result)
```

```
0    94
1     7
Name: CHAS, dtype: int64
```

In [24]:
```
value_counts_result = custom_value_counts(strat_train_set, 'CHAS')
print(value_counts_result)
```

```
0    377
1     28
Name: CHAS, dtype: int64
```

In [ ]:
```
7/94
```

Out[45]:
```
0.07446808510638298
```

In [ ]:
```
28/377
```

Out[46]:
```
0.07427055702917772
```

In [26]:
```
housing = strat_train_set.copy()
```

Looking for correlations

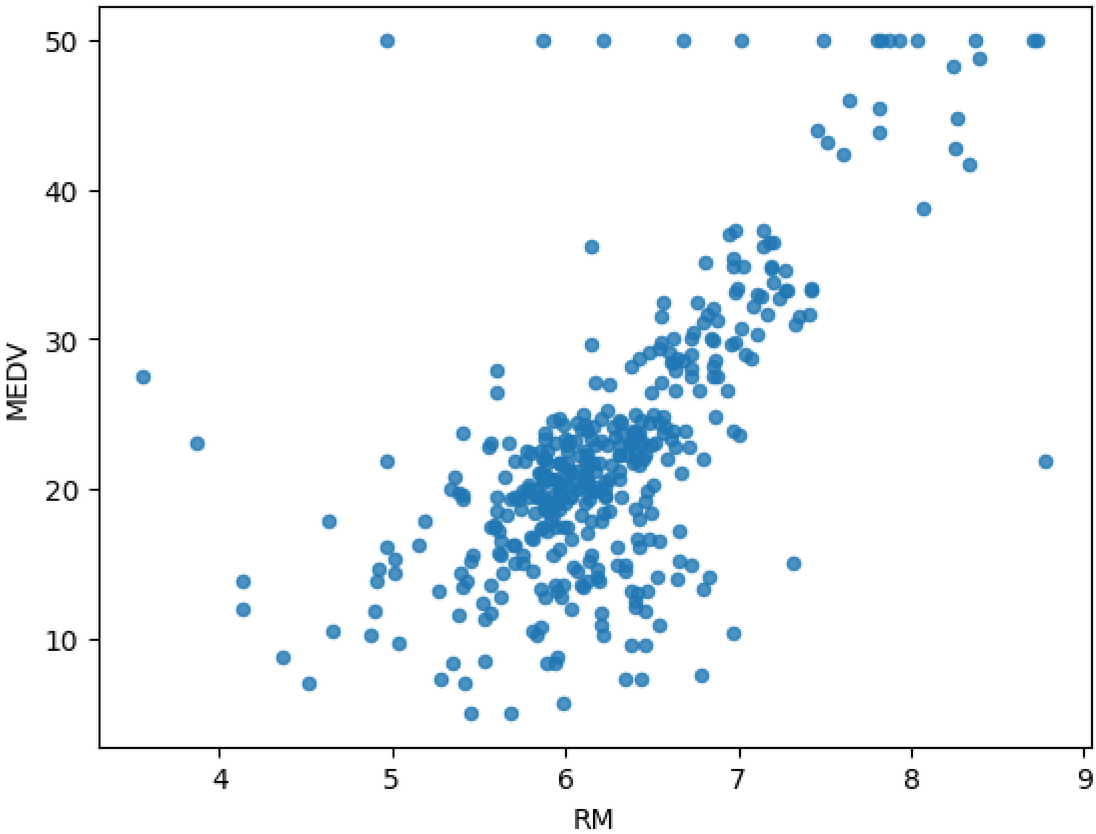
In [25]:
```
corr_matrix = housing.corr()
corr_matrix['MEDV'].sort_values(ascending=False)
```

Out[25]:
```
MEDV       1.000000
RM         0.696169
ZN         0.360445
B          0.333461
DIS        0.249929
CHAS       0.175260
AGE       -0.376955
RAD       -0.381626
CRIM      -0.388305
NOX       -0.427321
TAX       -0.468536
INDUS     -0.483725
PTRATIO   -0.507787
LSTAT     -0.737663
Name: MEDV, dtype: float64
```

In [ ]:
```
# from pandas.plotting import scatter_matrix
# attributes = ["MEDV", "RM", "ZN", "LSTAT"]
# scatter_matrix(housing[attributes], figsize = (12,8))
```

In [ ]:
```
housing.plot(kind="scatter", x="RM", y="MEDV", alpha=0.8)
```

Out[60]: <Axes: xlabel='RM', ylabel='MEDV'>



Trying out Attribute combinations

In [27]:
```
housing["TAXRM"] = housing['TAX']/housing['RM']
```

In [ ]:
```
housing.head()
```

Out[62]:

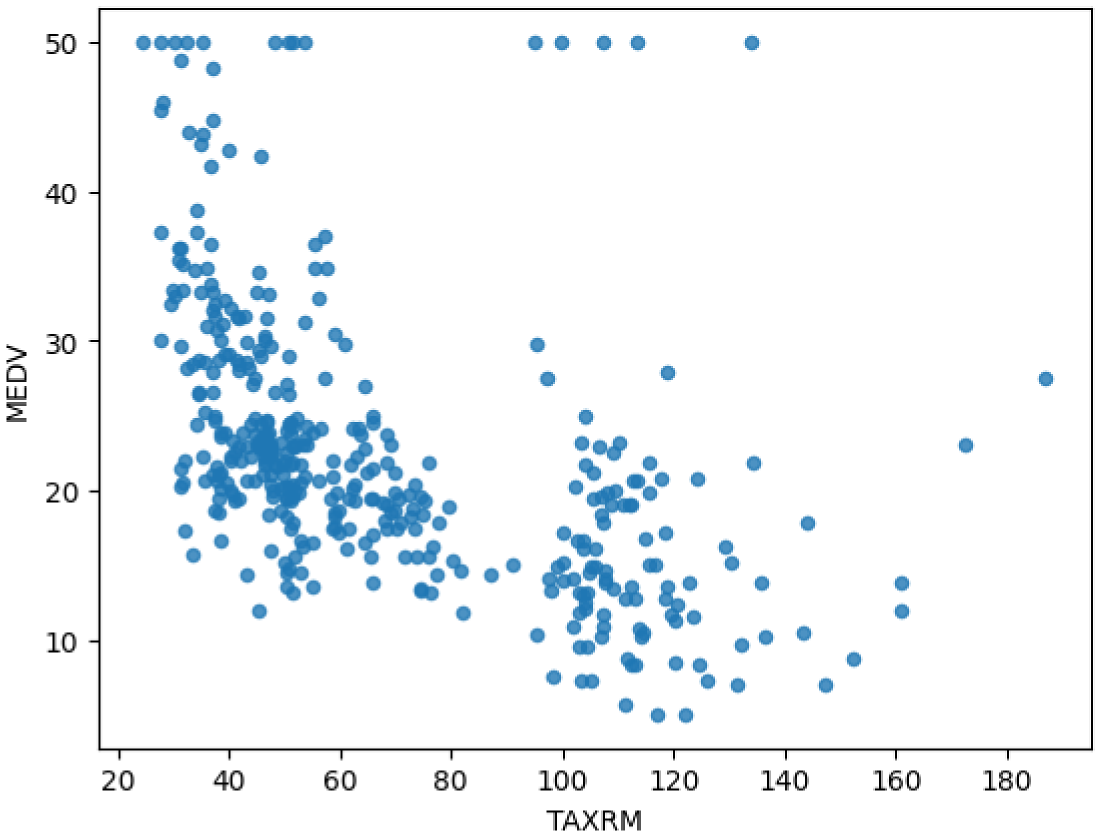
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
449	7.52601	0.0	18.10	0	0.7130	6.417	98.3	2.1850	24	666	20.2	304.21
332	0.03466	35.0	6.06	0	0.4379	6.031	23.3	6.6407	1	304	16.9	362.25
22	1.23247	0.0	8.14	0	0.5380	6.142	91.7	3.9769	4	307	21.0	396.90
334	0.03738	0.0	5.19	0	0.5150	6.310	38.5	6.4584	5	224	20.2	389.40
46	0.18836	0.0	6.91	0	0.4480	5.786	33.3	5.1004	3	233	17.9	396.90

In [28]:
```
corr_matrix = housing.corr()
corr_matrix['MEDV'].sort_values(ascending=False)
```

Out[28]:
```
MEDV       1.000000
RM         0.677626
ZN         0.331789
B          0.319503
DIS        0.243090
CHAS       0.158192
RAD       -0.362749
AGE       -0.371203
CRIM      -0.386091
NOX       -0.416177
TAX       -0.436108
INDUS     -0.463648
PTRATIO   -0.481774
TAXRM     -0.508594
LSTAT     -0.729320
Name: MEDV, dtype: float64
```

In [ ]:
```
housing.plot(kind="scatter", x="TAXRM", y="MEDV", alpha=0.8)
```

Out[64]:
```
<Axes: xlabel='TAXRM', ylabel='MEDV'>
```



In [29]:
```
housing = strat_train_set.drop("MEDV", axis=1)
housing_labels = strat_train_set["MEDV"].copy()
```

Missing attributes

In [ ]:
```
housing.describe() # before we started filling missing attributes
```

Out[67]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE
count	405.000000	405.000000	405.000000	405.000000	405.000000	401.000000	405.000000
mean	3.820903	11.622222	11.023753	0.069136	0.554263	6.259828	69.357037
std	9.281857	23.492715	6.853784	0.253999	0.116508	0.721169	27.669496
min	0.009060	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000
25%	0.079780	0.000000	5.130000	0.000000	0.449000	5.876000	46.700000
50%	0.253560	0.000000	8.560000	0.000000	0.538000	6.167000	78.700000
75%	3.673670	12.500000	18.100000	0.000000	0.624000	6.616000	94.300000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000

In [30]:
```
import pandas as pd

# Assuming you have a DataFrame named 'df'
# Check for missing values in each column
missing_values = housing.isna().sum()

# If you prefer to use 'isnull()' method, you can do so as follows:
# missing_values = df.isnull().sum()

# Display the columns with missing values
print(missing_values)
```

```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         4
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
dtype: int64
```

Creating Pipeline

In [31]:
```
import numpy as np
import pandas as pd

# Define your DataFrame 'data'
# For demonstration purposes, let's assume you have a DataFrame named 'data

# Step 1: Handle missing values (Imputation with median)
data=housing
for column in data.columns:
    if data[column].isnull().any():
        median = data[column].median()
        data[column].fillna(median, inplace=True)

# Step 2: Standardization
for column in data.columns:
    if data[column].dtype in [np.float64, np.float32, np.int64, np.int32]:
        mean = data[column].mean()
        std = data[column].std()
        data[column] = (data[column] - mean) / std

# Your data is now processed.

# Example usage:
# Assuming you have a DataFrame 'data', you can use this custom preprocessi
```

In [32]:
```
missing_values = housing.isna().sum()
print(missing_values)
```

```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
dtype: int64
```

In [ ]:
```
data.shape
```

Out[73]:
```
(405, 13)
```

In [42]:
```
data.describe()
```

Out[42]:

	CRIM	ZN	INDUS	CHAS	NOX	R
count	4.050000e+02	4.050000e+02	4.050000e+02	4.050000e+02	4.050000e+02	4.050000e+(
mean	3.070246e-17	6.908054e-17	1.600914e-16	7.017706e-17	3.070246e-17	3.157968e-
std	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+(
min	-4.106768e-01	-4.947160e-01	-1.541302e+00	-2.721896e-01	-1.452803e+00	-3.759374e+(
25%	-4.030576e-01	-4.947160e-01	-8.599269e-01	-2.721896e-01	-9.034853e-01	-5.321698e-(
50%	-3.843351e-01	-4.947160e-01	-3.594734e-01	-2.721896e-01	-1.395907e-01	-1.280725e-(
75%	-1.586247e-02	3.736383e-02	1.032458e+00	-2.721896e-01	5.985548e-01	4.836471e-(
max	9.174381e+00	3.761923e+00	2.438981e+00	3.664838e+00	2.718577e+00	3.512983e+(

Linear Regression Model

In [36]:
```
import numpy as np

class LinearRegression:
    def __init__(self, learning_rate=0.01, n_iterations=1000):
        self.learning_rate = learning_rate
        self.n_iterations = n_iterations
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        # Gradient Descent
        for _ in range(self.n_iterations):
            y_predicted = np.dot(X, self.weights) + self.bias

            # Compute gradients
            dw = (1 / n_samples) * np.dot(X.T, (y_predicted - y))
            db = (1 / n_samples) * np.sum(y_predicted - y)

            # Update weights and bias
            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

    def predict(self, X):
        y_predicted = np.dot(X, self.weights) + self.bias
        return y_predicted
```

In [38]:
```
# Create and train the linear regression model
model = LinearRegression()
model.fit(data, housing_labels)

# Make predictions
predictions = model.predict(data)

print("Predictions:", predictions)
```

```
Predictions: [17.69883554 23.90847408 16.23933297 21.94981502 20.4489512
14.64865458
 12.70427166 18.68371699  8.14161549 32.0642296  30.42187284 20.12566328
 13.31612432 16.20114181 17.40927297 25.12638598 32.68361165 22.28213751
 18.51560278 24.98248677 22.02561668 26.53775637 23.64899262 20.01945262
 25.80557294 25.65244494  7.60441713 22.39132379 31.05940808 35.30224078
 31.00902959 21.89878227 24.38252133 32.374873   25.74189466 24.18281836
 22.12980148 21.15131984 26.07270452 23.86588128 29.09511191 18.63675755
 17.89007086 21.23989356 24.13959677 27.96018431 21.35022929 30.46233597
 21.70585377 16.06645072 20.01394356 23.09339856 23.10153543 27.7927839
 16.56187334 21.07096624 25.84290101 13.27735088 18.53192417 25.87615678
 27.51037089 20.55493493 30.87650074 20.52306952 18.99037307 16.34126934
  4.25578667 20.42591827  8.33911234 26.63265574 14.5523689  32.1038494
 34.48134195 20.26079308 17.85238281 16.5055381  25.07358574 14.36744872
 16.38457484 22.74942307 24.99693896 11.74362012 22.94803311 10.90983353
 18.21832238 26.10618762 22.83956573 33.67901172 18.91000135 21.62022301
 13.55143473 31.30992373 18.46643488 14.72586666 36.95976619 24.25743949
 27.87148691 19.5903512  29.38833524 22.2746841  16.77429071 20.60509283
 31.05632923 25.82940855 19.12749983 21.0462142  27.29902226 26.82673298
 42.71063852 23.42038629 19.26967397 18.75898788 25.45569731 19.0377698
 22.20057723 31.0525352  18.77969379 16.41918915 20.85374524 26.90008752
 23.00102287 -4.08297763 26.5401184  20.75246072 28.67388285 22.59068895
 29.43296823 21.91556134 14.04782576 29.18114488 31.14449478  9.21589893
 22.67592653 24.46637971 10.43033093 30.76150059 27.74210059  1.634481
 21.69657127 28.56565035 14.49964195 13.72397834 25.03346795  9.39802971
 17.76385537 15.04924852 19.98615712 36.03158947 17.89644309 18.33052124
 14.24158884 23.21153443 12.66830712 18.74023039 31.526348   10.91846347
 34.76295321 22.03768022 30.74908002 38.02925203 25.64217579 16.0423854
 21.16717853 28.74924434 13.64301878 30.23687361 20.64904297 12.11577077
 27.5914285  18.18575028 28.3585178  14.47685618 21.61416201  6.27144849
 14.3008485  17.60960951 32.14032552 15.97375629 28.63644156 12.43538129
 25.445817   21.46164853 35.4186058   6.7445703  19.16289362 19.44512133
 14.14590782  6.21482798 30.86097924 25.40134788 24.40749386 16.29076358
 41.17054643 27.70655995 25.86640096 21.14569862 20.1747717  15.78914686
 16.97757819  9.79107179 29.01810156 15.54958938 23.25978631 18.70566114
 21.1395693   3.5326092  13.49554462 16.30583271 20.35557545 23.12054389
 17.26759218 29.96614264 18.50481728 27.88965016 40.52553365 31.79783735
 37.19844649 21.32159372 24.62477723 20.07466143 29.30497785 12.82186482
 16.84900659 23.71301589 23.69368089 28.9183429  23.9858783  34.33596104
  6.38326425 10.24442928 24.33304692 28.22552684 19.7396079  37.9723081
 23.65473686 34.44220209 10.17548463 28.41678568 33.75216598 17.83030843
 25.13777994 20.72685589  5.28891023 33.11006515 31.48360121 35.40941496
 21.6342371  36.97820716 18.3869299  18.15119099  9.20192582 24.41817615
 15.14741427 17.80315475 13.25068045 15.23487626 13.27233013 24.76390923
 26.67236724 20.82636435 26.72595186 14.9870082  14.14293313 23.99555585
 17.06403893 19.29858122 21.59845718 12.02859481 24.94558228 23.16645854
 24.16949181 19.78162606 19.83449069 36.77012729 33.13981191 17.13351665
 13.58719288 25.13194758 34.12782758 11.98587675 13.17960978 24.91941016
 36.09073485 39.3990428  24.33999857 23.13145918 18.49101853 27.75764671
 17.56224194 14.84428455 23.61581938  8.30131094 27.45222753 25.09915303
 26.43018249 24.82191912 32.68339558 32.66818348 20.23649569 19.94918628
 18.91772924 24.10809788 14.67903309 25.21349724 19.93518231 10.47787276
 17.46143982 28.34756842 29.48704437 14.03135058 31.83110308 18.06148269
 27.64389358  0.85749444 23.76770514 27.70742059 14.37813339 32.7920111
 36.68503034 20.19834458 24.68040293 30.96910616 18.65370515 30.96918329
 29.04848433 17.41578298 21.83629687 13.75063432 22.53704723 21.72974869
 16.69841591 33.94343633 16.51539927  6.74435629 21.31043858 27.16800214
 36.13640113 22.31941401 31.23861032 25.8731239  21.98082547  9.25743649
 16.96237182 22.12087177 32.85247526 18.10008727 21.38244506 23.21737586
 32.54864533 19.38106919 34.70552148 20.0020818  25.0914011  18.02477427
 15.94831383 31.09107051 20.90018268 21.57044048 20.47743489 18.01075641
```

```
 18.26845655 32.5270932  20.07725359 25.82175828 21.17757963 23.43999808
 37.90893939 20.69523382 22.34410078 12.8852981  39.68848588 21.9901138
 17.59461292 19.72206539 13.34694097 20.49031502 20.29118556 26.50000368
 17.22651701 23.85858047 22.61488163 35.0114147  15.66626159 33.09645864
 32.29398238 20.70181752 40.97691331 22.94563588 21.30723562 23.54499475
 33.09562628 21.26168727 21.26694975 35.20000667 38.31977337 17.23502973
 35.54359068 31.97964239 40.14334257 42.28726378 25.37680362 26.51754921
 23.84285133 29.60524951 23.9008274 ]
```

Model Evaluation

In [39]:

```
import numpy as np

# Calculate the squared differences
squared_errors = (housing_labels - predictions) ** 2

# Calculate the mean squared error (MSE)
mse = np.mean(squared_errors)

# Calculate the root mean squared error (RMSE)
rmse = np.sqrt(mse)

print("RMSE:", rmse)
```

```
RMSE: 4.945529902802092
```

In [ ]:

```
# Other Evaluation techniques
```

Decision Tree Regression Model

In [71]:
```
import numpy as np

class DecisionTreeRegressor:
    def __init__(self, max_depth=None, min_samples_split=2):
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split

    def fit(self, X, y):
        self.tree = self._build_tree(X, y, depth=0)

    def predict(self, X):
        return [self._predict_single(x, self.tree) for x in X]

    def _build_tree(self, X, y, depth):
        n_samples, n_features = X.shape
        impurity, threshold, split_index = self._find_best_split(X, y)

        if impurity <= 0.0 or (self.max_depth and depth >= self.max_depth)
            return np.mean(y)

        left_indices = np.where(X[:, split_index] <= threshold)[0]
        right_indices = np.where(X[:, split_index] > threshold)[0]

        left_tree = self._build_tree(X[left_indices], y[left_indices], dept
        right_tree = self._build_tree(X[right_indices], y[right_indices], d

        return (split_index, threshold, left_tree, right_tree)


    def _find_best_split(self, X, y):
        n_samples, n_features = X.shape
        if n_samples <= 1:
            return float('inf'), None, None

        impurity_parent = self._calculate_mse(y)

        best_impurity = float('inf')
        best_threshold = None
        best_split_index = None

        for feature_index in range(n_features):
            feature_values = X[:, feature_index]
            unique_values = np.unique(feature_values)

            for threshold in unique_values:
                left_indices = np.where(feature_values <= threshold)[0]
                right_indices = np.where(feature_values > threshold)[0]

                if len(left_indices) == 0 or len(right_indices) == 0:
                    continue

                impurity_left = self._calculate_mse(y[left_indices])
                impurity_right = self._calculate_mse(y[right_indices])
                weighted_impurity = (len(left_indices) / n_samples) * impur

                if weighted_impurity < best_impurity:
                    best_impurity = weighted_impurity
                    best_threshold = threshold
                    best_split_index = feature_index

        return best_impurity, best_threshold, best_split_index
```

```
    def _calculate_mse(self, y):
        if len(y) == 0:
            return 0
        mean = np.mean(y)
        mse = np.mean((y - mean) ** 2)
        return mse

    def _predict_single(self, x, tree):
        if isinstance(tree, (int, float)):
            return tree

        split_index, threshold, left_tree, right_tree = tree
        if x[split_index] <= threshold:
            return self._predict_single(x, left_tree)
        else:
            return self._predict_single(x, right_tree)
```

In [75]:
```
# Create and train the Desicion Tree regression model
model = DecisionTreeRegressor(max_depth=5, min_samples_split=2)
model.fit(np.array(data), np.array(housing_labels))

# # Make predictions
predictions = model.predict(np.array(data))

print("Predictions:", predictions)
```

```
Predictions: [13.036842105263158, 22.065384615384612, 15.534782608695652,
19.89074074074074, 22.065384615384612, 19.018181818181816, 15.53478260869
5652, 19.018181818181816, 9.471428571428572, 32.40833333333333, 27.915384
615384617, 19.89074074074074, 13.036842105263158, 15.534782608695652, 19.
89074074074074, 22.065384615384612, 27.915384615384617, 22.06538461538461
2, 19.89074074074074, 22.065384615384612, 23.27777777777778, 22.065384615
384612, 22.065384615384612, 19.89074074074074, 22.065384615384612, 22.065
384615384612, 9.471428571428572, 23.27777777777778, 32.40833333333333, 3
3.58888888888889, 32.40833333333333, 19.89074074074074, 22.06538461538461
2, 32.40833333333333, 27.915384615384617, 22.065384615384612, 22.06538461
5384612, 19.89074074074074, 26.42777777777778, 50.0, 32.40833333333333,
9.471428571428572, 19.89074074074074, 19.89074074074074, 27.9153846153846
17, 26.42777777777778, 26.42777777777778, 32.40833333333333, 22.065384615
384612, 19.89074074074074, 20.488235294117647, 19.89074074074074, 19.8907
4074074074, 22.065384615384612, 14.940000000000001, 19.018181818181816, 2
2.065384615384612, 15.534782608695652, 13.036842105263158, 26.42777777777
778, 23.75, 13.036842105263158, 27.915384615384617, 22.065384615384612, 1
9.018181818181816, 15.534782608695652, 15.534782608695652, 19.89074074074
074, 9.471428571428572, 22.065384615384612, 13.036842105263158, 32.408333
33333333, 33.58888888888889, 20.488235294117647, 9.471428571428572, 15.53
4782608695652, 22.065384615384612, 15.534782608695652, 14.94000000000000
1, 19.89074074074074, 22.065384615384612, 14.940000000000001, 22.06538461
5384612, 19.89074074074074, 15.534782608695652, 22.065384615384612, 19.89
074074074074, 32.40833333333333, 19.89074074074074, 22.065384615384612, 1
3.036842105263158, 26.42777777777778, 22.065384615384612, 19.890740740740
74, 42.55, 22.065384615384612, 26.42777777777778, 15.534782608695652, 22.
065384615384612, 22.065384615384612, 15.534782608695652, 13.0368421052631
58, 27.915384615384617, 22.065384615384612, 15.534782608695652, 19.890740
74074074, 26.42777777777778, 22.065384615384612, 49.85, 22.06538461538461
2, 15.534782608695652, 22.065384615384612, 22.065384615384612, 22.0653846
15384612, 22.065384615384612, 32.40833333333333, 20.488235294117647, 9.47
1428571428572, 19.89074074074074, 26.42777777777778, 22.065384615384612,
9.471428571428572, 22.065384615384612, 15.534782608695652, 22.06538461538
4612, 22.065384615384612, 26.42777777777778, 22.065384615384612, 20.48823
5294117647, 22.065384615384612, 32.40833333333333, 15.534782608695652, 2
2.065384615384612, 22.065384615384612, 15.534782608695652, 32.40833333333
333, 22.065384615384612, 14.940000000000001, 22.065384615384612, 27.91538
4615384617, 15.534782608695652, 13.036842105263158, 22.065384615384612, 1
5.534782608695652, 19.018181818181816, 19.89074074074074, 19.890740740740
74, 38.7, 19.018181818181816, 22.065384615384612, 15.534782608695652, 22.
065384615384612, 15.534782608695652, 15.534782608695652, 32.4083333333333
3, 20.488235294117647, 32.40833333333333, 22.065384615384612, 33.58888888
888889, 46.0, 22.065384615384612, 15.534782608695652, 22.065384615384612,
27.915384615384617, 15.534782608695652, 27.915384615384617, 22.0653846153
84612, 9.471428571428572, 23.27777777777778, 14.940000000000001, 22.06538
4615384612, 15.534782608695652, 19.89074074074074, 9.471428571428572, 19.
89074074074074, 15.534782608695652, 32.40833333333333, 19.89074074074074,
22.065384615384612, 13.036842105263158, 23.27777777777778, 15.53478260869
5652, 46.0, 9.471428571428572, 19.89074074074074, 19.89074074074074, 15.5
34782608695652, 13.036842105263158, 42.55, 26.42777777777778, 22.06538461
5384612, 15.534782608695652, 49.85, 22.065384615384612, 32.4083333333333
3, 15.534782608695652, 22.065384615384612, 13.036842105263158, 22.0653846
15384612, 9.471428571428572, 32.40833333333333, 15.534782608695652, 22.06
5384615384612, 14.940000000000001, 19.89074074074074, 9.471428571428572,
13.036842105263158, 15.534782608695652, 13.036842105263158, 22.0653846153
84612, 14.940000000000001, 23.27777777777778, 22.065384615384612, 26.4277
777777778, 49.85, 32.40833333333333, 49.85, 20.488235294117647, 22.06538
4615384612, 19.89074074074074, 27.915384615384617, 15.534782608695652, 1
4.940000000000001, 32.40833333333333, 22.065384615384612, 26.427777777777
78, 22.065384615384612, 33.58888888888889, 9.471428571428572, 9.471428571
428572, 22.065384615384612, 32.40833333333333, 27.5, 46.0, 22.06538461538
```

```
4612, 32.40833333333333, 9.471428571428572, 32.40833333333333, 32.4083333
3333333, 13.036842105263158, 26.42777777777778, 22.065384615384612, 9.471
428571428572, 32.40833333333333, 26.42777777777778, 46.0, 22.065384615384
612, 46.0, 19.89074074074074, 20.488235294117647, 19.018181818181816, 22.
065384615384612, 19.89074074074074, 20.488235294117647, 15.53478260869565
2, 15.534782608695652, 9.471428571428572, 23.27777777777778, 22.065384615
384612, 22.065384615384612, 22.065384615384612, 9.471428571428572, 15.534
782608695652, 19.89074074074074, 15.534782608695652, 15.534782608695652,
19.89074074074074, 15.45, 32.40833333333333, 19.89074074074074, 26.427777
77777778, 22.065384615384612, 19.89074074074074, 49.85, 32.4083333333333
3, 15.534782608695652, 13.036842105263158, 22.065384615384612, 32.4083333
3333333, 9.471428571428572, 9.471428571428572, 50.0, 46.0, 46.0, 22.06538
4615384612, 22.065384615384612, 19.89074074074074, 22.065384615384612, 1
9.018181818181816, 19.018181818181816, 22.065384615384612, 9.471428571428
572, 22.065384615384612, 22.065384615384612, 22.065384615384612, 22.06538
4615384612, 32.40833333333333, 33.58888888888889, 19.89074074074074, 22.0
65384615384612, 20.488235294117647, 22.065384615384612, 14.94000000000000
1, 22.065384615384612, 15.534782608695652, 15.534782608695652, 20.4882352
94117647, 27.915384615384617, 26.42777777777778, 15.534782608695652, 32.4
0833333333333, 9.471428571428572, 22.065384615384612, 9.471428571428572,
19.89074074074074, 23.27777777777778, 9.471428571428572, 32.4083333333333
3, 33.58888888888889, 19.89074074074074, 22.065384615384612, 23.75, 19.89
074074074074, 32.40833333333333, 22.065384615384612, 23.27777777777778, 2
2.065384615384612, 15.534782608695652, 23.27777777777778, 19.890740740740
74, 22.065384615384612, 33.58888888888889, 19.89074074074074, 9.471428571
428572, 22.065384615384612, 22.065384615384612, 32.40833333333333, 19.890
74074074074, 27.915384615384617, 22.065384615384612, 19.89074074074074, 1
5.45, 9.471428571428572, 22.065384615384612, 32.40833333333333, 19.890740
74074074, 19.89074074074074, 22.065384615384612, 32.40833333333333, 14.94
0000000000001, 33.58888888888889, 13.036842105263158, 22.065384615384612,
13.036842105263158, 9.471428571428572, 33.58888888888889, 19.890740740740
74, 22.065384615384612, 19.89074074074074, 15.534782608695652, 9.47142857
1428572, 32.40833333333333, 19.89074074074074, 22.065384615384612, 22.065
384615384612, 19.89074074074074, 46.0, 22.065384615384612, 19.89074074074
074, 15.534782608695652, 49.85, 22.065384615384612, 20.488235294117647, 2
0.488235294117647, 13.036842105263158, 19.018181818181816, 19.89074074074
074, 22.065384615384612, 20.488235294117647, 20.488235294117647, 20.48823
5294117647, 50.0, 15.534782608695652, 27.915384615384617, 26.427777777777
78, 13.036842105263158, 49.85, 20.488235294117647, 22.065384615384612, 1
5.534782608695652, 50.0, 19.018181818181816, 19.89074074074074, 32.408333
33333333, 21.9, 20.488235294117647, 32.40833333333333, 26.42777777777778,
46.0, 49.85, 22.065384615384612, 50.0, 22.065384615384612, 22.06538461538
4612, 19.89074074074074]
```

Model Evaluation

In [76]:
```
import numpy as np

# Calculate the squared differences
squared_errors = (housing_labels - predictions) ** 2

# Calculate the mean squared error (MSE)
mse = np.mean(squared_errors)

# Calculate the root mean squared error (RMSE)
rmse = np.sqrt(mse)

print("RMSE:", rmse)
```

```
RMSE: 2.8162598634670104
```

RandomForestRegressor()

In [86]:
```
import numpy as np

class RandomForestRegressor:
    def __init__(self, n_estimators=100, max_depth=None, min_samples_split=
        self.n_estimators = n_estimators
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.max_features = max_features
        self.trees = []

    def fit(self, X, y):
        for _ in range(self.n_estimators):
            sample_indices = np.random.choice(len(X), len(X), replace=True)
            if self.max_features:
                feature_indices = np.random.choice(X.shape[1], self.max_fea
                X_subsample = X[sample_indices][:, feature_indices]
            else:
                X_subsample = X[sample_indices]
            y_subsample = y[sample_indices]

            tree = DecisionTreeRegressor(self.max_depth, self.min_samples_s
            tree.fit(X_subsample, y_subsample)
            self.trees.append(tree)

    def predict(self, X):
        predictions = [tree.predict(X) for tree in self.trees]
        return np.mean(predictions, axis=0)

class DecisionTreeRegressor:
    def __init__(self, max_depth=None, min_samples_split=2):
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split

    def fit(self, X, y):
        self.tree = self._build_tree(X, y, depth=0)

    def predict(self, X):
        return [self._predict_single(x, self.tree) for x in X]

    def _build_tree(self, X, y, depth):
        n_samples, n_features = X.shape
        impurity, threshold, split_index = self._find_best_split(X, y)

        if impurity <= 0.0 or split_index is None or threshold is None or (
            return np.mean(y)

        left_indices = np.where(X[:, split_index] <= threshold)[0]
        right_indices = np.where(X[:, split_index] > threshold)[0]

        if len(left_indices) == 0 or len(right_indices) == 0:
            return np.mean(y)

        left_tree = self._build_tree(X[left_indices], y[left_indices], dept
        right_tree = self._build_tree(X[right_indices], y[right_indices], d

        return (split_index, threshold, left_tree, right_tree)

    def _find_best_split(self, X, y):
        n_samples, n_features = X.shape
        if n_samples <= 1:
            return float('inf'), None, None
```

```
        impurity_parent = self._calculate_mse(y)

        best_impurity = float('inf')
        best_threshold = None
        best_split_index = None

        for feature_index in range(n_features):
            feature_values = X[:, feature_index]
            unique_values = np.unique(feature_values)

            for threshold in unique_values:
                left_indices = np.where(feature_values <= threshold)[0]
                right_indices = np.where(feature_values > threshold)[0]

                if len(left_indices) == 0 or len(right_indices) == 0:
                    continue

                impurity_left = self._calculate_mse(y[left_indices])
                impurity_right = self._calculate_mse(y[right_indices])
                weighted_impurity = (len(left_indices) / n_samples) * impur

                if weighted_impurity < best_impurity:
                    best_impurity = weighted_impurity
                    best_threshold = threshold
                    best_split_index = feature_index

        return best_impurity, best_threshold, best_split_index

    def _calculate_mse(self, y):
        if len(y) == 0:
            return 0
        mean = np.mean(y)
        mse = np.mean((y - mean) ** 2)
        return mse

    def _predict_single(self, x, tree):
        if isinstance(tree, (int, float)):
            return tree

        split_index, threshold, left_tree, right_tree = tree
        if x[split_index] <= threshold:
            return self._predict_single(x, left_tree)
        else:
            return self._predict_single(x, right_tree)
```

In [87]:
```
# Create and train the Desicion Tree regression model
model = RandomForestRegressor()
model.fit(np.array(data), np.array(housing_labels))

# # Make predictions
predictions = model.predict(np.array(data))

print("Predictions:", predictions)
```

```
Predictions: [13.37440952 19.67719286 15.40424881 21.41999675 20.43701429
19.72006111
 16.28093571 19.67165    12.7577     31.13086667 31.35878333 19.05816515
  9.9091     16.05478333 17.86734444 19.85127698 29.33688333 22.00858571
 16.44735397 23.23010476 22.93714286 24.86700952 23.5623381  20.68701429
 20.87589863 24.0471     7.79935238 25.11106667 32.58265556 34.97131548
 33.44088333 19.9749619  22.38246667 35.17288413 27.29648333 20.87312857
 19.93889127 22.92362857 24.05105    46.33772587 33.983      10.93180549
 18.97037857 21.45089372 23.50665952 25.16786071 21.68826667 34.33283333
 21.27444524 16.93541944 23.97395833 20.76860238 20.10014762 22.19866667
 14.3177119  20.11398333 19.52296667 13.57430952 11.68016667 28.47397857
 25.93791667 14.19417857 29.28109048 20.62363571 18.66198095 14.66025714
 14.55086905 20.13378571  6.57437619 24.43069762 11.80104524 31.28225238
 29.97021667 19.85978016 13.17131905 15.8802316  23.5408     14.48455476
 12.49121667 22.25388182 22.92028961 15.7474     21.47710476 22.59035
 15.48947698 24.12911667 21.5502619  36.2973487  20.82022857 20.15998413
 12.46356905 28.80632024 18.58737857 28.6029619  42.93779719 22.07238333
 27.39660238 13.546125   27.38266667 20.62759524 17.02150952 13.67485476
 31.2313381  23.41288571 16.13447143 19.90347381 23.96721429 22.71255119
 48.76570589 21.97915952 17.38632619 19.60206667 24.42979048 19.41533333
 25.04922143 30.32441905 16.80745833 12.54920238 19.56119048 24.86631667
 23.72055476  7.0470881  21.9582     16.34964167 23.39788247 22.42730476
 28.4427619  24.15810476 19.15505    23.36744913 30.14578889 14.55071667
 18.5982     22.30931667 16.31265    34.29695556 22.8529     16.39458333
 20.97982143 28.03955952 17.15118571 11.59016667 21.47394524 12.94906667
 19.9914596  17.41860238 21.75258333 41.90826905 18.11690152 18.52795556
 14.60025    21.09435    14.16226667 19.14630866 35.07369762 22.30506667
 34.54828095 21.58962294 29.64594762 45.50738711 22.06408333 14.16095
 23.56873333 25.64343333 16.31735    27.74229167 20.3072      6.2112119
 24.53658333 14.48593571 24.83036667 15.05032327 17.95824524  8.61127619
 13.38088526 15.06070675 31.04452857 21.681      25.7608     11.24375238
 23.3093     19.0330881  46.90467209  9.66095476 19.16001032 19.10687857
 14.81476786 13.02898571 43.74859127 21.26925714 20.38564524 17.29675952
 47.98800337 22.27427857 31.52018571 18.79705476 21.63189048 14.14597937
 18.10804524  7.74566429 31.4795119  16.32184881 19.73335    17.07033333
 20.1271      8.59485476 14.82728175 17.9753     13.82145714 21.40975
 17.55037857 24.19791667 17.69342976 26.75540476 47.48003144 30.92090952
 45.32651577 21.50360628 20.06198333 18.90608333 24.36211667 14.90730303
 17.32858333 27.65153333 20.5226     27.77923333 20.84375714 33.22129048
  8.95062381  9.74234048 28.3896     27.73283333 23.94737857 45.38952198
 20.55058315 33.82745952  8.72418571 35.45765238 34.87442619 14.98483889
 26.4577369  23.33985952  8.82759524 33.18064524 29.40212381 43.82304469
 21.2167619  46.55178321 17.0046619  17.60430833 14.12467143 22.06235238
 18.79126111 21.38596667 13.77093333 15.25757619  9.12285952 24.40691667
 23.09016905 17.748925   22.57788333 10.77272381 14.50995238 22.72658452
 17.58726667 18.08104286 21.47606905 16.90047024 30.06256667 20.03408333
 32.63675    21.81598095 18.70576825 48.84803763 34.72175897 17.62425303
  8.83187857 23.28748571 36.86483175 10.59556429 10.37151667 40.96333929
 44.48111453 46.74804139 23.09771429 21.05754267 20.29299762 24.48616667
 21.74087778 17.42078095 21.01486818  8.49699286 26.42491667 26.32826667
 23.17752857 24.31351905 33.89522643 31.07452381 19.48370952 21.19886667
 19.68149683 22.3151     12.86506429 21.83422381 15.78188398 10.47625238
 19.13974048 27.75675833 28.18027381 14.11505476 29.51011667  8.56466429
 22.48806818 12.85883571 22.14936429 25.22137619 13.7973924  35.94252564
 42.18765076 20.45744505 19.63744286 26.78537222 20.87016667 26.81735714
 23.5704     21.50805476 21.84062468 15.67704351 22.63143452 21.15032143
 18.00148333 31.66448095 18.95973571  5.85312619 20.34515    30.96026667
 34.35413333 19.45284029 28.648975   22.98472619 23.13318333 15.79374524
 11.07284286 20.23350714 32.17507619 19.16829286 19.30347381 22.30624762
 33.46343333 14.91233571 33.46578333 15.61056667 23.68333333 14.14877937
 12.01442692 31.83446905 20.64585714 20.91296667 19.14167619 18.04299286
```

```
 11.98308333 34.55049603 29.20959762 22.1995881  21.28117143 24.33959167
 44.72914681 21.11319405 15.22894744 14.19215476 47.05178173 21.91131667
 19.14229683 20.51505    12.13753333 19.30668889 18.92449524 22.4824619
 19.037475   20.17596461 20.86331818 46.12964437 13.84047381 28.32404167
 28.191725   16.39619603 48.13382486 21.92379913 17.3947119  17.1607
 45.5581092  16.3556881  17.84317857 33.17009603 31.27091337 19.53961786
 31.64511667 28.33528333 45.86244656 48.73462486 20.89923333 45.86658929
 23.3450461  22.59943095 21.11721667]
```

Model Evaluation

In [88]:

```
import numpy as np

# Calculate the squared differences
squared_errors = (housing_labels - predictions) ** 2

# Calculate the mean squared error (MSE)
mse = np.mean(squared_errors)

# Calculate the root mean squared error (RMSE)
rmse = np.sqrt(mse)

print("RMSE:", rmse)
```

```
RMSE: 1.3823386843106842
```

Saving Model

In [89]:

```
from joblib import dump, load
dump(model, 'Dragon.joblib')
```

Out[89]:

```
['Dragon.joblib']
```

Testing model on Test data

In [90]:

```
X_test = strat_test_set.drop("MEDV", axis=1)
Y_test = strat_test_set["MEDV"].copy()
```

In [92]:
```
import numpy as np
import pandas as pd

# Define your DataFrame 'data'
# For demonstration purposes, let's assume you have a DataFrame named 'data

# Step 1: Handle missing values (Imputation with median)
data=X_test
for column in data.columns:
    if data[column].isnull().any():
        median = data[column].median()
        data[column].fillna(median, inplace=True)

# Step 2: Standardization
for column in data.columns:
    if data[column].dtype in [np.float64, np.float32, np.int64, np.int32]:
        mean = data[column].mean()
        std = data[column].std()
        data[column] = (data[column] - mean) / std

# Your data is now processed.

# Example usage:
# Assuming you have a DataFrame 'data', you can use this custom preprocessi
```

In [93]:
```
predictions = model.predict(np.array(data))
```

In [96]:
```
print("Predictions:", predictions)
```

```
Predictions: [33.37963333 21.53815476 15.68332143 30.5084     15.55442698
25.13641071
 45.762587   19.65289794  8.34413095 23.69480476 20.70781515 12.31303333
 10.05234762 20.8503961  19.778075   19.52492381 20.0744     23.30486667
 17.54694286 20.43906667 24.77433333 20.86728214 19.42313333 41.84642013
 20.11675    46.26648906 15.47141905 21.46098333 15.6793381  26.31057262
 18.94994524 34.82857738 24.55901905 10.04423095 19.28619524 26.0368
 12.30560714 19.59622857 19.46081746 15.89235    22.73672857 47.5408416
 25.561525   41.26206905 16.7873746  19.75784167 22.69570476 22.48845
 21.70916818 17.69078333 12.84852381 32.87117222 46.99008149 11.53108095
 23.07726667 19.13770278 19.69628077 16.74739048 20.26815    14.64947684
 19.08449762 21.32020628 19.76535079 33.46662937 23.09968452 18.83505556
 24.77495476 11.65398333 26.6037     11.0357381  30.98277381 19.05882601
 36.37528452  9.45133333 18.80737619  9.35717857 21.15775952 22.35828571
 21.10966172  9.8847619  22.56717619 23.39753333 23.78887143  9.20767619
 16.18855     7.93722143  7.03665476 24.43679167 22.71143333 18.69187381
 18.0238     19.01161905 15.20986984 25.13193929 45.24507251 19.91465
 30.24998333 41.34521905 27.73470833 20.11639286 23.41199762]
```

Model Evaluation

In [95]:
```
import numpy as np

# Calculate the squared differences
squared_errors = (Y_test - predictions) ** 2

# Calculate the mean squared error (MSE)
mse = np.mean(squared_errors)

# Calculate the root mean squared error (RMSE)
rmse = np.sqrt(mse)

print("RMSE:", rmse)
```

```
RMSE: 2.8166770346677232
```

Using the model

In [97]:
```
from joblib import dump, load
import numpy as np
model = load('Dragon.joblib')
features = np.array([[-5.43942006, 4.12628155, -1.6165014, -0.67288841, -1.
       -11.44443979304, -49.31238772,  7.61111401, -26.0016879 , -0.577819
       -0.97491834,  0.41164221, -66.86091034]])
model.predict(features)
```

Out[97]:
```
array([23.74960476])
```