Information about data

In [1]:
```
import pandas as pd
```

In [2]:
```
housing = pd.read_csv("data.csv")
```

In [3]:
```
housing.head()
```

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33

In [4]:
```
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: 55.4 KB
```

In [5]:
```
housing["CHAS"].value_counts()
```

Out[5]:
```
0    471
1     35
Name: CHAS, dtype: int64
```

In [6]:
```
housing.describe()
```

Out[6]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	
count	506.000000	506.000000	506.000000	506.000000	506.000000	501.000000	506.000000	506
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284341	68.574901	3
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.705587	28.148861	2
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.884000	45.025000	2
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208000	77.500000	3
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.625000	94.075000	5
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12

Splitting data into train and test

In [7]:
```
from sklearn.model_selection import train_test_split
trainset, testset = train_test_split(housing, test_size=0.2, random_state=42
print(f"No. of rows in train set: {len(trainset)}\nNo. of rows in test set:

```

```
No. of rows in train set: 404
No. of rows in test set:  102
```

In [8]:
```
from sklearn.model_selection import StratifiedShuffleSplit
spt = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in spt.split(housing, housing["CHAS"]):
    strat_trainset = housing.loc[train_index]
    strat_testset = housing.loc[test_index]
strat_trainset["CHAS"].value_counts()

```

Out[8]:
```
0    376
1     28
Name: CHAS, dtype: int64
```

In [9]:
```
strat_testset["CHAS"].value_counts()
```

Out[9]:
```
0    95
1     7
Name: CHAS, dtype: int64
```

In [10]:
```
housing = strat_trainset.copy()
```

In [11]:
```
housing.shape
```

Out[11]:
```
(404, 14)
```

In [12]:
```
housing_labels = strat_trainset["MEDV"].copy()
```

Looking for correlations

In [13]:
```
corr_matrix = housing.corr()
corr_matrix["MEDV"].sort_values(ascending = False)
```

Out[13]:
```
MEDV       1.000000
RM         0.680857
B          0.361761
ZN         0.339741
DIS        0.240451
CHAS       0.205066
AGE       -0.364596
RAD       -0.374693
CRIM      -0.393715
NOX       -0.422873
TAX       -0.456657
INDUS     -0.473516
PTRATIO   -0.493534
LSTAT     -0.740494
Name: MEDV, dtype: float64
```

Creating pipeline

In [14]:
```
housing_data = housing.drop("MEDV", axis = 1)
```

In [15]:
```
housing_data.shape
```

Out[15]:
```
(404, 13)
```

In [16]:
```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
my_pipeline = Pipeline ([
    ('imputer', SimpleImputer(strategy="median")),
    ('std_scaler', StandardScaler()),
])

```

In [17]:
```
housing_tr = my_pipeline.fit_transform(housing_data)
```

In [18]:
```
housing_tr.shape
```

Out[18]:
```
(404, 13)
```

Choosing the best model

In [19]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
model_rfr = RandomForestRegressor()
model_lr = LinearRegression()
model_dtr = DecisionTreeRegressor()
model_rfr.fit(housing_tr, housing_labels)
```

Out[19]: `RandomForestRegressor()`

In [20]:
```
model_lr.fit(housing_tr, housing_labels)
```

Out[20]: `LinearRegression()`

In [21]:
```
model_dtr.fit(housing_tr, housing_labels)
```

Out[21]: `DecisionTreeRegressor()`

In [22]:
```
import numpy as np
from sklearn.metrics import mean_squared_error
housing_predictions_rfr = model_rfr.predict(housing_tr)
mse_rfr = mean_squared_error(housing_labels, housing_predictions_rfr)
rmse_rfr = np.sqrt(mse_rfr)
print(f"mse_rfr: {mse_rfr}\nrmse_rfr: {rmse_rfr}")
```

```
mse_rfr: 1.4579957351485127
rmse_rfr: 1.2074749418304764
```

In [23]:
```
housing_predictions_lr = model_lr.predict(housing_tr)
mse_lr  = mean_squared_error(housing_labels, housing_predictions_lr )
rmse_lr  = np.sqrt(mse_lr )
print(f"mse_lr: {mse_lr}\nrmse_lr: {rmse_lr}")
```

```
mse_lr: 23.380136328422374
rmse_lr: 4.835301058716238
```

In [24]:
```
housing_predictions_dtr = model_dtr.predict(housing_tr)
mse_dtr = mean_squared_error(housing_labels, housing_predictions_dtr)
rmse_dtr = np.sqrt(mse_dtr)
print(f"mse_dtr: {mse_dtr}\nrmse_dtr: {rmse_dtr}")  # this is known as over
```

```
mse_dtr: 0.0
rmse_dtr: 0.0
```

In [25]:
```
from sklearn.model_selection import cross_val_score
import numpy as np
scores_lr = cross_val_score(model_lr, housing_tr, housing_labels, scoring="n
rmse_scores_lr = np.sqrt(-scores_lr)
```

In [26]:
```
scores_dtr = cross_val_score(model_dtr, housing_tr, housing_labels, scoring=
rmse_scores_dtr = np.sqrt(-scores_dtr)
```

In [27]:
```
scores_rfr = cross_val_score(model_rfr, housing_tr, housing_labels, scoring=
rmse_scores_rfr = np.sqrt(-scores_rfr)
```

In [28]:
```
def print_scores(scores):
    print("Scores:", scores)
    print("Mean: ", scores.mean())
    print("RMS: ", np.sqrt((scores**2).mean()))
    print("Standard deviation: ", scores.std())
```

In [29]:
```
print_scores(rmse_scores_lr)
```

```
Scores: [4.22235612 4.26438649 5.09424333 3.83081183 5.37600331 4.41092152
 7.47272243 5.48554135 4.14606627 6.0717752 ]
Mean:  5.037482786117751
RMS:  5.147683188192044
Standard deviation:  1.0594382405606955
```

In [30]:
```
print_scores(rmse_scores_dtr)
```

```
Scores: [3.94696549 4.43582973 5.04235717 4.13833948 4.08071685 2.98257439
 5.11131099 4.0234003  3.21939435 3.70482793]
Mean:  4.068571668485012
RMS:  4.12016797550069
Standard deviation:  0.6500067113057745
```

In [31]:
```
print_scores(rmse_scores_rfr) # chosing this model because of low mean and s
```

```
Scores: [2.85559876 2.83036399 4.57049673 2.56482946 3.68750516 2.62305342
 4.93372897 3.29772233 3.48475141 3.26600872]
Mean:  3.4114058945648558
RMS:  3.4948284378318273
Standard deviation:  0.7590359888741811
```

Creating a suitable model

In [32]:
```
from joblib import dump, load
dump(model_rfr, "house_predictor.joblib")
```

Out[32]:
```
['house_predictor.joblib']
```

Testing the test data

In [33]:
```
test_data = strat_testset.drop("MEDV", axis = 1)
actual_test_output = strat_testset["MEDV"].copy()
prepared_test_data = my_pipeline.transform(test_data )
```

In [34]:
```
from sklearn.metrics import mean_squared_error
housing_predictions = model_rfr.predict(housing_tr)
mse = mean_squared_error(housing_labels, housing_predictions)
rmse = np.sqrt(mse)
```

In [35]:
```
our_test_output = model_rfr.predict(prepared_test_data)
final_mse = mean_squared_error(actual_test_output, our_test_output)
final_rmse = np.sqrt(final_mse)
print(f"final rmse: {final_rmse}") # final rmse is less than RMS:3.3632
```

```
final rmse: 2.9784042147879566
```

In [40]:
```
a = len(our_test_output)
print(a)
j =1;
for i in our_test_output:
    print(f"{j}:{i}", end= ', ')
    j = j+1
```

```
102
1:25.17900000000001, 2:11.791999999999994, 3:25.456000000000014, 4:21.756999999
999998, 5:18.16099999999999, 6:15.116999999999997, 7:19.708999999999993, 8:14.5
66999999999993, 9:31.605, 10:41.70700000000002, 11:19.796000000000003, 12:12.10
599999999999, 13:24.888999999999996, 14:27.333, 15:19.577999999999996, 16:10.79
8999999999994, 17:31.473, 18:14.174000000000005, 19:23.700999999999986, 20:17.9
1, 21:19.961, 22:17.477999999999998, 23:16.637999999999995, 24:22.0329999999999
98, 25:18.35899999999999, 26:31.181000000000008, 27:16.144999999999996, 28:32.5
95000000000006, 29:8.793999999999997, 30:33.20799999999999, 31:23.5459999999999
96, 32:21.34399999999999, 33:22.69499999999998, 34:11.30199999999999, 35:20.924
99999999999, 36:11.083, 37:42.86000000000002, 38:24.615, 39:23.144999999999996,
40:42.43000000000002, 41:24.041999999999998, 42:30.48700000000001, 43:20.206000
000000017, 44:20.866000000000007, 45:18.767999999999997, 46:33.52699999999999,
47:45.42100000000005, 48:20.363999999999997, 49:20.391999999999992, 50:21.57099
999999999, 51:21.697999999999993, 52:14.424, 53:21.099000000000004, 54:15.04299
9999999996, 55:25.37, 56:33.14299999999998, 57:41.37300000000001, 58:28.7539999
9999999, 59:19.495000000000008, 60:20.96500000000001, 61:46.78800000000002, 62:
9.491999999999996, 63:18.883999999999993, 64:24.677999999999997, 65:14.67699999
999999, 66:33.167999999999985, 67:19.563, 68:18.060000000000006, 69:19.16100000
0000005, 70:34.507000000000005, 71:25.21499999999999, 72:22.955999999999975, 7
3:21.279999999999994, 74:22.309999999999995, 75:34.90300000000001, 76:12.953, 7
7:16.118999999999986, 78:20.01400000000001, 79:20.758999999999997, 80:21.508999
999999993, 81:22.215999999999998, 82:21.001000000000005, 83:14.110000000000005,
84:23.587999999999987, 85:20.822, 86:21.023000000000003, 87:13.657999999999998,
88:21.127000000000006, 89:21.684000000000005, 90:23.80900000000001, 91:18.631,
92:27.218000000000004, 93:7.329, 94:26.891, 95:18.571999999999996, 96:29.474999
99999998, 97:19.58599999999999, 98:30.96999999999998, 99:14.722999999999999, 10
0:27.34599999999999, 101:21.571000000000005, 102:20.447999999999993,
```

In [42]:
```
a = len(actual_test_output)
print(a)
j =1;
for i in actual_test_output:
    print(f"{j}:{i}", end= ', ')
    j = j+1
```

```
102
1:16.5, 2:10.2, 3:30.1, 4:23.0, 5:14.4, 6:15.6, 7:19.4, 8:14.1, 9:30.3, 10:35.
2, 11:23.1, 12:13.8, 13:25.0, 14:27.9, 15:19.5, 16:12.3, 17:32.2, 18:13.5, 19:2
3.8, 20:21.7, 21:19.2, 22:19.5, 23:10.4, 24:23.2, 25:18.6, 26:28.5, 27:15.2, 2
8:32.0, 29:7.2, 30:34.6, 31:20.1, 32:20.6, 33:23.6, 34:13.1, 35:23.8, 36:12.7,
37:43.1, 38:24.7, 39:22.2, 40:44.0, 41:28.1, 42:31.0, 43:21.7, 44:23.4, 45:19.
5, 46:33.1, 47:41.7, 48:18.7, 49:19.9, 50:20.6, 51:21.2, 52:13.6, 53:20.3, 54:1
7.8, 55:27.1, 56:31.5, 57:50.0, 58:29.1, 59:18.9, 60:20.4, 61:50.0, 62:7.2, 63:
17.2, 64:36.2, 65:14.6, 66:33.2, 67:23.8, 68:19.9, 69:21.5, 70:37.3, 71:27.0, 7
2:22.0, 73:24.3, 74:19.8, 75:33.3, 76:7.0, 77:19.4, 78:20.9, 79:21.1, 80:20.4,
81:22.2, 82:11.9, 83:11.7, 84:21.6, 85:19.7, 86:23.0, 87:16.7, 88:21.7, 89:20.
6, 90:23.3, 91:19.6, 92:28.0, 93:5.0, 94:24.4, 95:20.8, 96:24.8, 97:21.8, 98:2
3.6, 99:19.0, 100:25.0, 101:20.3, 102:21.5,
```

Using the model for unknown data and predicting the price

In [38]:
```
from joblib import dump, load
model = load("house_predictor.joblib")
features = np.array([[0.02187,  60.00,   2.930, 0,  0.4010,  66.8000,   34.9
model.predict(features)
```

Out[38]:
```
array([21.455])
```