untitled44

April 24, 2024

[1]:
```
import numpy as np
import pandas as pd
```

[4]:
```
from sklearn.datasets import fetch_openml

# Load the Boston housing dataset
boston = fetch_openml(data_id=531)

# Access the data and target attributes
X = boston.data
y = boston.target

# Print the shape of the data and target
print("Shape of data:", X.shape)
print("Shape of target:", y.shape)
```

```
E:\Bandicam\anaconda\Lib\site-packages\sklearn\datasets\_openml.py:968:
FutureWarning: The default value of `parser` will change from `'liac-arff'` to
`'auto'` in 1.4. You can set `parser='auto'` to silence this warning. Therefore,
an `ImportError` will be raised from 1.4 if the dataset is dense and pandas is
not installed. Note that the pandas parser may return different data types. See
the Notes Section in fetch_openml's API doc for details.
  warn(

Shape of data: (506, 13)
Shape of target: (506,)
```

[ ]:
```
```

[7]:
```
data = pd.DataFrame(boston.data)
```

[8]:
```
data.head()
```

[8]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	\
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	

```
4  0.06905   0.0   2.18    0  0.458  7.147   54.2  6.0622   3  222.0     18.7

        B  LSTAT
0  396.90   4.98
1  396.90   9.14
2  392.83   4.03
3  394.63   2.94
4  396.90   5.33
```

[9]:
```
data.columns = boston.feature_names
```

[10]:
```
data['PRICE'] = boston.target
```

[11]:
```
data.head(n=10)
```

[11]:
```
      CRIM    ZN  INDUS CHAS    NOX     RM    AGE     DIS RAD    TAX  PTRATIO  \
0  0.00632  18.0   2.31    0  0.538  6.575   65.2  4.0900   1  296.0     15.3
1  0.02731   0.0   7.07    0  0.469  6.421   78.9  4.9671   2  242.0     17.8
2  0.02729   0.0   7.07    0  0.469  7.185   61.1  4.9671   2  242.0     17.8
3  0.03237   0.0   2.18    0  0.458  6.998   45.8  6.0622   3  222.0     18.7
4  0.06905   0.0   2.18    0  0.458  7.147   54.2  6.0622   3  222.0     18.7
5  0.02985   0.0   2.18    0  0.458  6.430   58.7  6.0622   3  222.0     18.7
6  0.08829  12.5   7.87    0  0.524  6.012   66.6  5.5605   5  311.0     15.2
7  0.14455  12.5   7.87    0  0.524  6.172   96.1  5.9505   5  311.0     15.2
8  0.21124  12.5   7.87    0  0.524  5.631  100.0  6.0821   5  311.0     15.2
9  0.17004  12.5   7.87    0  0.524  6.004   85.9  6.5921   5  311.0     15.2

        B  LSTAT  PRICE
0  396.90   4.98   24.0
1  396.90   9.14   21.6
2  392.83   4.03   34.7
3  394.63   2.94   33.4
4  396.90   5.33   36.2
5  394.12   5.21   28.7
6  395.60  12.43   22.9
7  396.90  19.15   27.1
8  386.63  29.93   16.5
9  386.71  17.10   18.9
```

[12]:
```
print(data.shape)
```

```
(506, 14)
```

[14]:
```
data.isnull().sum()
```

[14]:
```
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
PRICE      0
dtype: int64
```

[15]:
```
data.describe()
```

[15]:

	CRIM	ZN	INDUS	NOX	RM	AGE
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.554695	6.284634	68.574901
std	8.601545	23.322453	6.860353	0.115878	0.702617	28.148861
min	0.006320	0.000000	0.460000	0.385000	3.561000	2.900000
25%	0.082045	0.000000	5.190000	0.449000	5.885500	45.025000
50%	0.256510	0.000000	9.690000	0.538000	6.208500	77.500000
75%	3.677083	12.500000	18.100000	0.624000	6.623500	94.075000
max	88.976200	100.000000	27.740000	0.871000	8.780000	100.000000

	DIS	TAX	PTRATIO	B	LSTAT	PRICE
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.795043	408.237154	18.455534	356.674032	12.653063	22.532806
std	2.105710	168.537116	2.164946	91.294864	7.141062	9.197104
min	1.129600	187.000000	12.600000	0.320000	1.730000	5.000000
25%	2.100175	279.000000	17.400000	375.377500	6.950000	17.025000
50%	3.207450	330.000000	19.050000	391.440000	11.360000	21.200000
75%	5.188425	666.000000	20.200000	396.225000	16.955000	25.000000
max	12.126500	711.000000	22.000000	396.900000	37.970000	50.000000

[16]:
```
data.info()
```

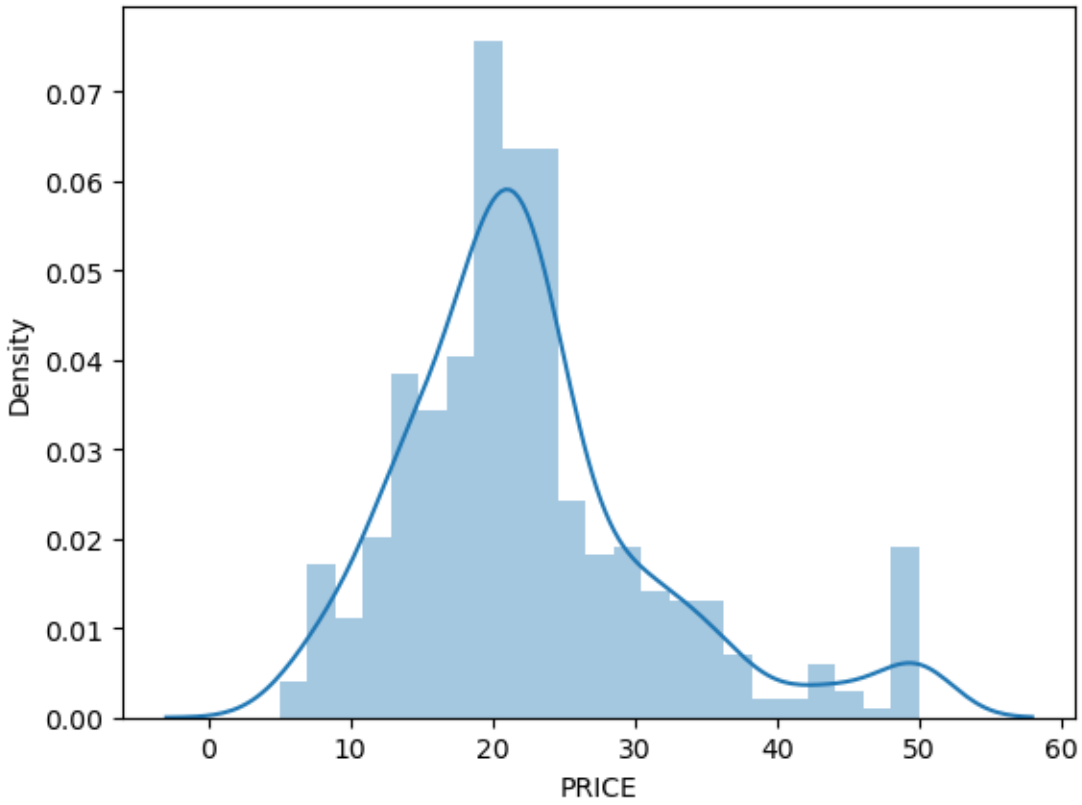
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    category
```

```
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    category
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  PRICE    506 non-null    float64
dtypes: category(2), float64(12)
memory usage: 49.0 KB
```

[21]:
```
import seaborn as sns
sns.distplot(data.PRICE)
```

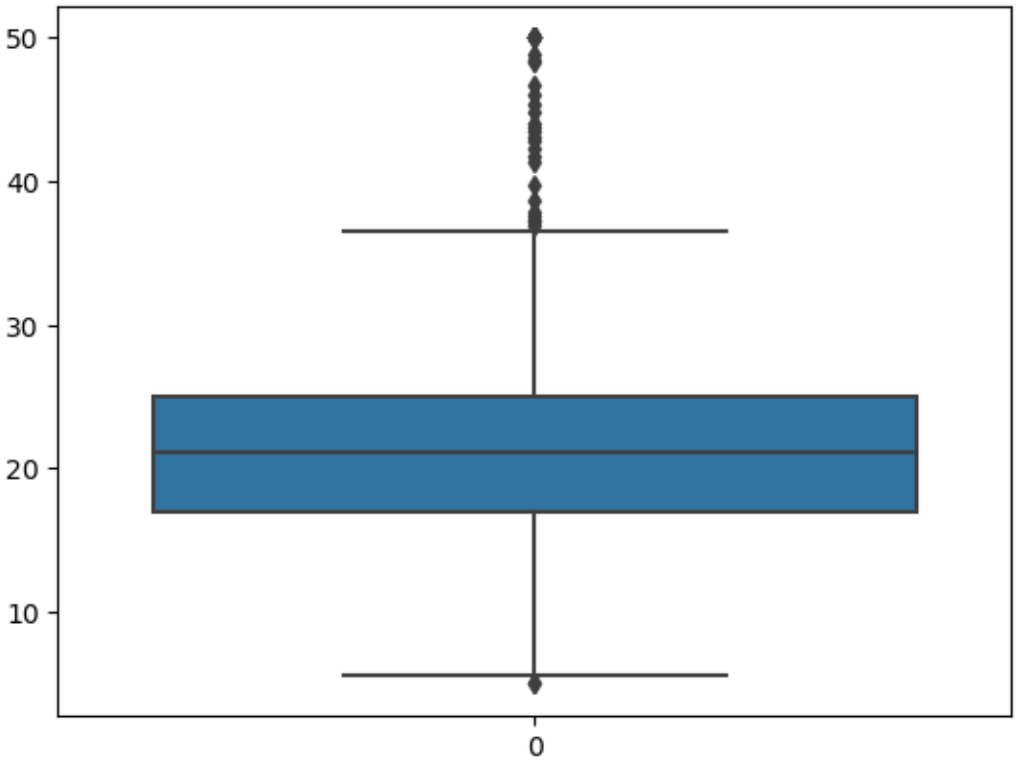
```
C:\Users\dell\AppData\Local\Temp\ipykernel_12392\3137224507.py:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data.PRICE)
E:\Bandicam\anaconda\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

[21]: <Axes: xlabel='PRICE', ylabel='Density'>



[22]:
```
sns.boxplot(data.PRICE)
```

[22]: <Axes: >



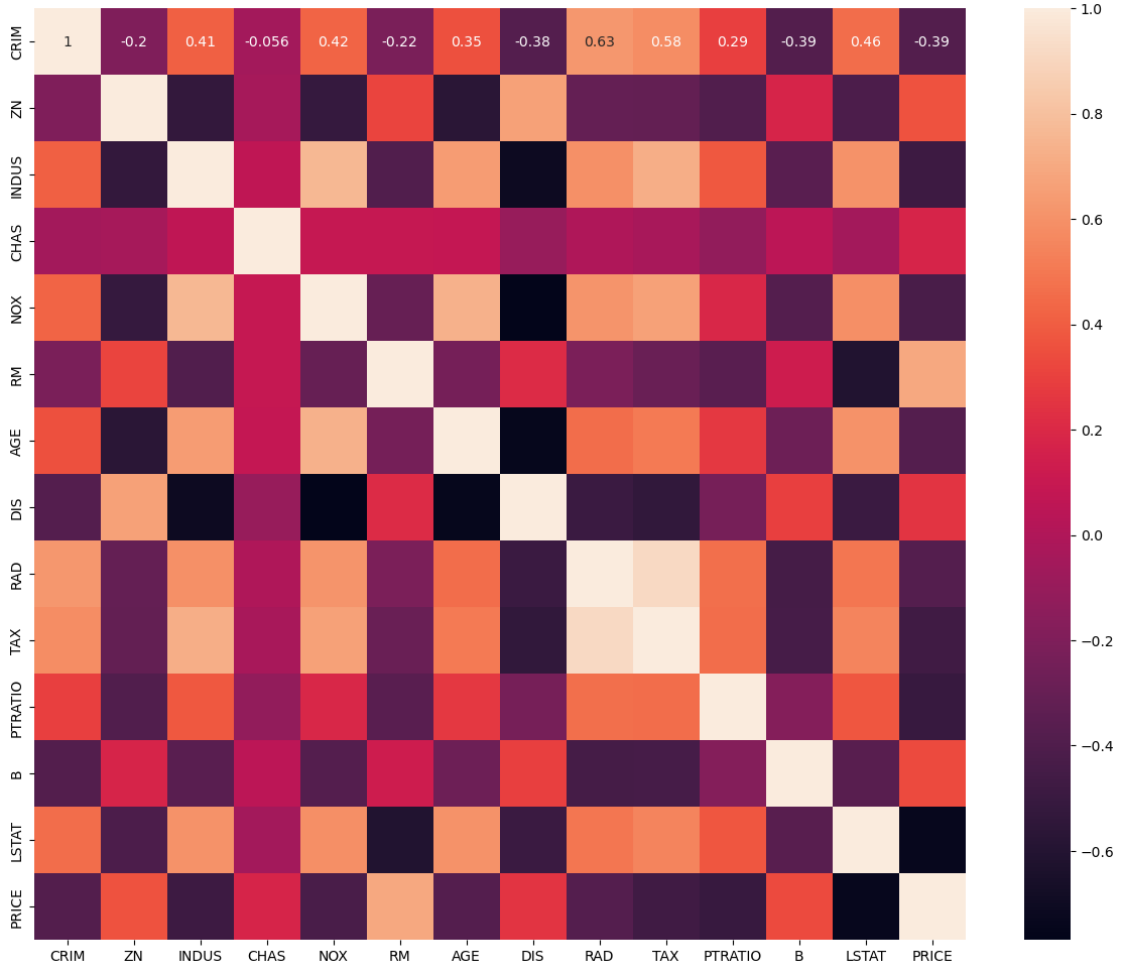
[24]:
```
#Checking the correlation of the independent feature with the dependent feature
# Correlation is a statistical technique that can show whether and how strongly
#pairs of variables are related.An intelligent correlation analysis can lead to␣
 ↪a
#greater understanding of your data
#checking Correlation of the data
correlation = data.corr()
correlation.loc['PRICE']
```

[24]:
```
CRIM      -0.388305
ZN         0.360445
INDUS     -0.483725
CHAS       0.175260
NOX       -0.427321
RM         0.695360
AGE       -0.376955
DIS        0.249929
RAD       -0.381626
TAX       -0.468536
PTRATIO   -0.507787
B          0.333461
LSTAT     -0.737663
```

```
PRICE      1.000000
Name: PRICE, dtype: float64
```

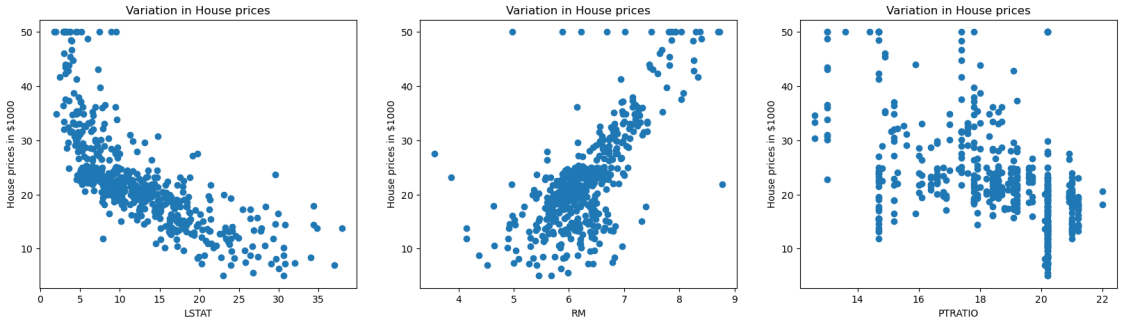
[27]:
```
# plotting the heatmap
import matplotlib.pyplot as plt
fig,axes = plt.subplots(figsize=(15,12))
sns.heatmap(correlation,square = True,annot = True)
# By looking at the correlation plot LSAT is negatively correlated with -0.75␣
 ↪and
#RM is positively correlated to the price and PTRATIO is correlated negatively
#with -0.51
```

[27]:
```
<Axes: >
```



[29]:
```
# Checking the scatter plot with the most correlated features
plt.figure(figsize=(20, 5))
features = ['LSTAT', 'RM', 'PTRATIO']
```

```
for i, col in enumerate(features):
    plt.subplot(1, len(features), i + 1)
    x = data[col]
    y = data.PRICE
    plt.scatter(x, y, marker='o')
    plt.title("Variation in House prices")
    plt.xlabel(col)
    plt.ylabel('House prices in $1000')
```



[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: