04.MultivariableRegression

April 30, 2021

1 Notebook Imports

[81]:
```
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

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

import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor

%matplotlib inline
```

2 Gather data

Source: Original Research Paper

[2]:
```
boston_dataset = load_boston()
```

[3]:
```
type(boston_dataset)
```

[3]:
```
sklearn.utils.Bunch
```

[4]:
```
dir(boston_dataset)
```

[4]:
```
['DESCR', 'data', 'feature_names', 'filename', 'target']
```

[5]:
```
print(boston_dataset.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------
```

Data Set Characteristics:

:Number of Instances: 506

:Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the target.

:Attribute Information (in order):
- CRIM   per capita crime rate by town
- ZN   proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS   proportion of non-retail business acres per town
- CHAS   Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
- NOX   nitric oxides concentration (parts per 10 million)
- RM   average number of rooms per dwelling
- AGE   proportion of owner-occupied units built prior to 1940
- DIS   weighted distances to five Boston employment centres
- RAD   index of accessibility to radial highways
- TAX   full-value property-tax rate per $10,000
- PTRATIO pupil-teacher ratio by town
- B   1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- LSTAT   % lower status of the population
- MEDV   Median value of owner-occupied homes in $1000's

:Missing Attribute Values: None

:Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/

This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics …', Wiley, 1980.   N.B. Various transformations are used in the table on pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regression
problems.

.. topic:: References

```
   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential
Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In
Proceedings on the Tenth International Conference of Machine Learning, 236-243,
University of Massachusetts, Amherst. Morgan Kaufmann.
```

2.0.1 Data points and features

[6]:
```python
boston_dataset.data.shape #chaining dot notation
```

[6]:
```
(506, 13)
```

[7]:
```python
boston_dataset.feature_names
```

[7]:
```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

[8]:
```python
# Actual prices in thousands(1000's)
boston_dataset.target
```

[8]:
```
array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9, 15. ,
       18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
       15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
       13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
       21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
       35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
       19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. ,
       20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2,
       23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8,
       33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.4,
       21.7, 22.8, 18.8, 18.7, 18.5, 18.3, 21.2, 19.2, 20.4, 19.3, 22. ,
       20.3, 20.5, 17.3, 18.8, 21.4, 15.7, 16.2, 18. , 14.3, 19.2, 19.6,
       23. , 18.4, 15.6, 18.1, 17.4, 17.1, 13.3, 17.8, 14. , 14.4, 13.4,
       15.6, 11.8, 13.8, 15.6, 14.6, 17.8, 15.4, 21.5, 19.6, 15.3, 19.4,
       17. , 15.6, 13.1, 41.3, 24.3, 23.3, 27. , 50. , 50. , 50. , 22.7,
       25. , 50. , 23.8, 23.8, 22.3, 17.4, 19.1, 23.1, 23.6, 22.6, 29.4,
       23.2, 24.6, 29.9, 37.2, 39.8, 36.2, 37.9, 32.5, 26.4, 29.6, 50. ,
       32. , 29.8, 34.9, 37. , 30.5, 36.4, 31.1, 29.1, 50. , 33.3, 30.3,
       34.6, 34.9, 32.9, 24.1, 42.3, 48.5, 50. , 22.6, 24.4, 22.5, 24.4,
       20. , 21.7, 19.3, 22.4, 28.1, 23.7, 25. , 23.3, 28.7, 21.5, 23. ,
       26.7, 21.7, 27.5, 30.1, 44.8, 50. , 37.6, 31.6, 46.7, 31.5, 24.3,
       31.7, 41.7, 48.3, 29. , 24. , 25.1, 31.5, 23.7, 23.3, 22. , 20.1,
       22.2, 23.7, 17.6, 18.5, 24.3, 20.5, 24.5, 26.2, 24.4, 24.8, 29.6,
       42.8, 21.9, 20.9, 44. , 50. , 36. , 30.1, 33.8, 43.1, 48.8, 31. ,
       36.5, 22.8, 30.7, 50. , 43.5, 20.7, 21.1, 25.2, 24.4, 35.2, 32.4,
       32. , 33.2, 33.1, 29.1, 35.1, 45.4, 35.4, 46. , 50. , 32.2, 22. ,
```

```
       20.1, 23.2, 22.3, 24.8, 28.5, 37.3, 27.9, 23.9, 21.7, 28.6, 27.1,
       20.3, 22.5, 29. , 24.8, 22. , 26.4, 33.1, 36.1, 28.4, 33.4, 28.2,
       22.8, 20.3, 16.1, 22.1, 19.4, 21.6, 23.8, 16.2, 17.8, 19.8, 23.1,
       21. , 23.8, 23.1, 20.4, 18.5, 25. , 24.6, 23. , 22.2, 19.3, 22.6,
       19.8, 17.1, 19.4, 22.2, 20.7, 21.1, 19.5, 18.5, 20.6, 19. , 18.7,
       32.7, 16.5, 23.9, 31.2, 17.5, 17.2, 23.1, 24.5, 26.6, 22.9, 24.1,
       18.6, 30.1, 18.2, 20.6, 17.8, 21.7, 22.7, 22.6, 25. , 19.9, 20.8,
       16.8, 21.9, 27.5, 21.9, 23.1, 50. , 50. , 50. , 50. , 50. , 13.8,
       13.8, 15. , 13.9, 13.3, 13.1, 10.2, 10.4, 10.9, 11.3, 12.3,  8.8,
        7.2, 10.5,  7.4, 10.2, 11.5, 15.1, 23.2,  9.7, 13.8, 12.7, 13.1,
       12.5,  8.5,  5. ,  6.3,  5.6,  7.2, 12.1,  8.3,  8.5,  5. , 11.9,
       27.9, 17.2, 27.5, 15. , 17.2, 17.9, 16.3,  7. ,  7.2,  7.5, 10.4,
        8.8,  8.4, 16.7, 14.2, 20.8, 13.4, 11.7,  8.3, 10.2, 10.9, 11. ,
        9.5, 14.5, 14.1, 16.1, 14.3, 11.7, 13.4,  9.6,  8.7,  8.4, 12.8,
       10.5, 17.1, 18.4, 15.4, 10.8, 11.8, 14.9, 12.6, 14.1, 13. , 13.4,
       15.2, 16.1, 17.8, 14.9, 14.1, 12.7, 13.5, 14.9, 20. , 16.4, 17.7,
       19.5, 20.2, 21.4, 19.9, 19. , 19.1, 19.1, 20.1, 19.9, 19.6, 23.2,
       29.8, 13.8, 13.3, 16.7, 12. , 14.6, 21.4, 23. , 23.7, 25. , 21.8,
       20.6, 21.2, 19.1, 20.6, 15.2,  7. ,  8.1, 13.6, 20.1, 21.8, 24.5,
       23.1, 19.7, 18.3, 21.2, 17.5, 16.8, 22.4, 20.6, 23.9, 22. , 11.9])
```

2.1 Data exploration with Pandas dataframes

[9]:
```
# Create a pandas data frame
data= pd.DataFrame(data=boston_dataset.data,columns=boston_dataset.
 ↪feature_names)

#Add column with the price
data['PRICE']=boston_dataset.target
```

[10]:
```
data.head() # The top rows looks like this
```

[10]:
```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

   PTRATIO       B  LSTAT  PRICE
0     15.3  396.90   4.98   24.0
1     17.8  396.90   9.14   21.6
2     17.8  392.83   4.03   34.7
3     18.7  394.63   2.94   33.4
4     18.7  396.90   5.33   36.2
```

[11]:
```
data.tail() # The bottom rows looks like this
```

[11]:
```
        CRIM   ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
501  0.06263  0.0  11.93   0.0  0.573  6.593  69.1  2.4786  1.0  273.0
502  0.04527  0.0  11.93   0.0  0.573  6.120  76.7  2.2875  1.0  273.0
503  0.06076  0.0  11.93   0.0  0.573  6.976  91.0  2.1675  1.0  273.0
504  0.10959  0.0  11.93   0.0  0.573  6.794  89.3  2.3889  1.0  273.0
505  0.04741  0.0  11.93   0.0  0.573  6.030  80.8  2.5050  1.0  273.0

     PTRATIO       B  LSTAT  PRICE
501     21.0  391.99   9.67   22.4
502     21.0  396.90   9.08   20.6
503     21.0  396.90   5.64   23.9
504     21.0  393.45   6.48   22.0
505     21.0  396.90   7.88   11.9
```

[12]:
```
data.count() # Show us the no. of rows
```

[12]:
```
CRIM       506
ZN         506
INDUS      506
CHAS       506
NOX        506
RM         506
AGE        506
DIS        506
RAD        506
TAX        506
PTRATIO    506
B          506
LSTAT      506
PRICE      506
dtype: int64
```

2.2 Cleaning Data- Check for missing values

[13]:
```
pd.isnull(data).any()
```

[13]:
```
CRIM       False
ZN         False
INDUS      False
CHAS       False
NOX        False
RM         False
AGE        False
DIS        False
RAD        False
TAX        False
PTRATIO    False
```

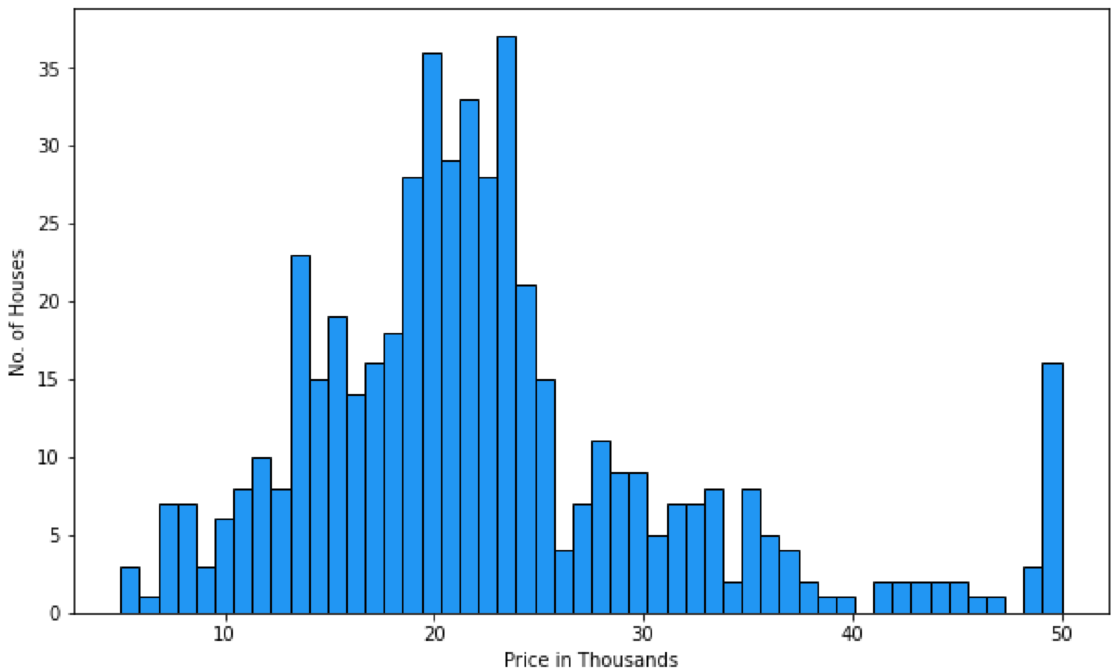
```
B          False
LSTAT      False
PRICE      False
dtype: bool
```

[14]:
```
data.info()
```

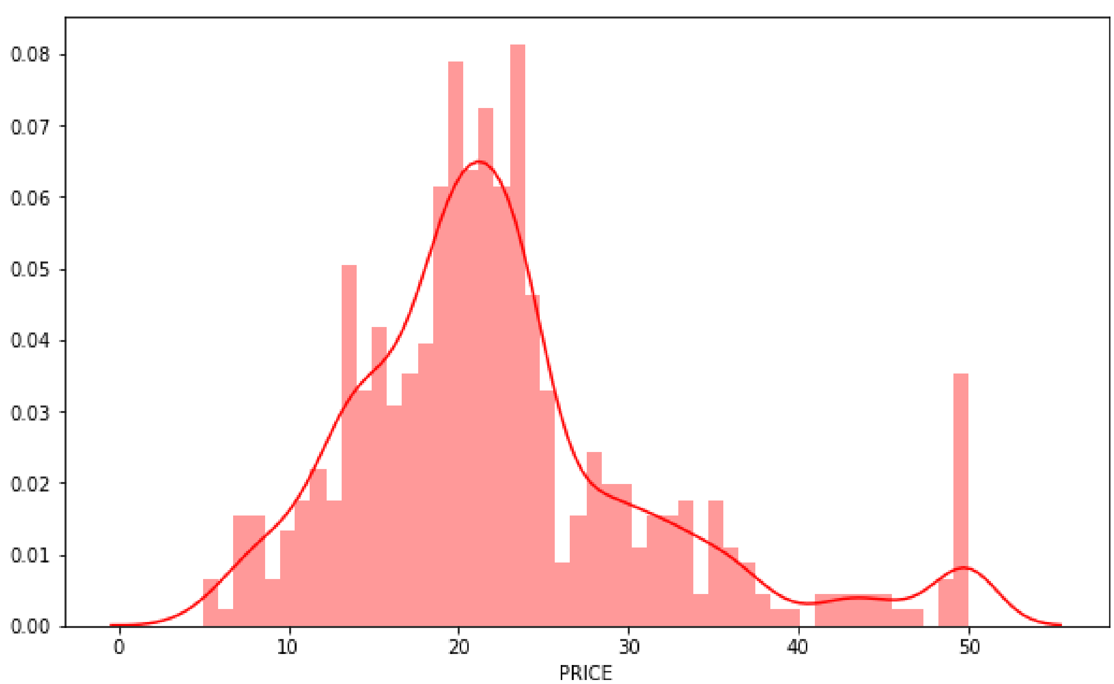
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 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    float64
 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: float64(14)
memory usage: 55.5 KB
```

2.3 Visualising Data - Histograms, Distributions and Bar Charts

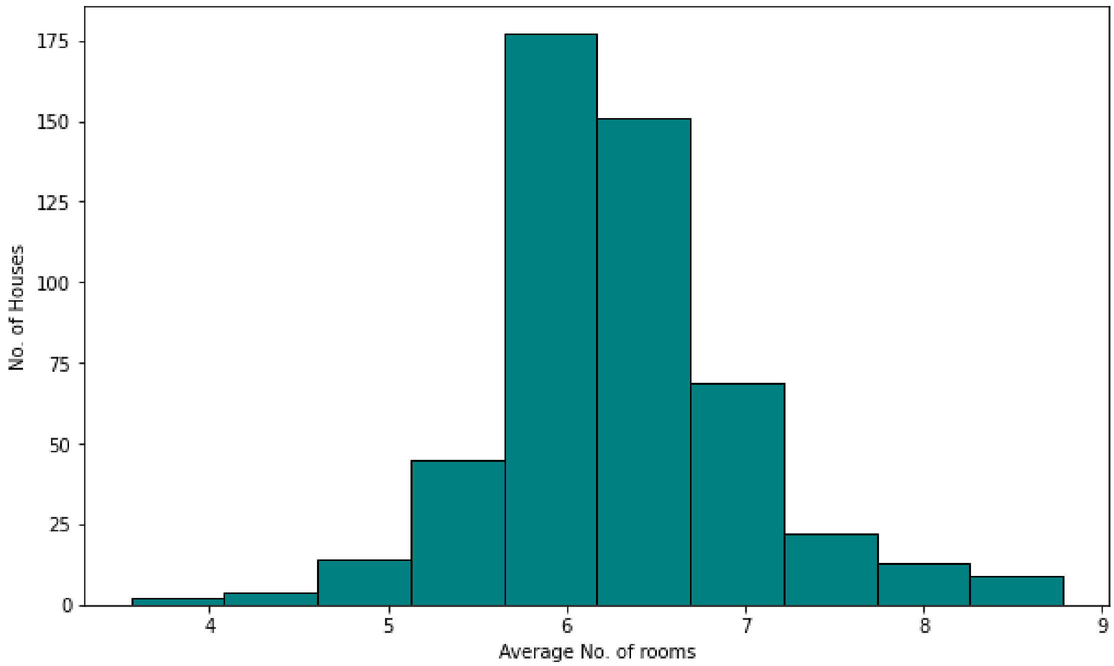
[15]:
```
plt.figure(figsize=(10,6))
plt.hist(data["PRICE"],bins=50,ec='black',color='#2196F3')
plt.xlabel('Price in Thousands')
plt.ylabel('No. of Houses')
plt.show()
```



[16]:
```
plt.figure(figsize=(10,6))
sns.distplot(data['PRICE'],bins=50,color='red')
plt.show()
```



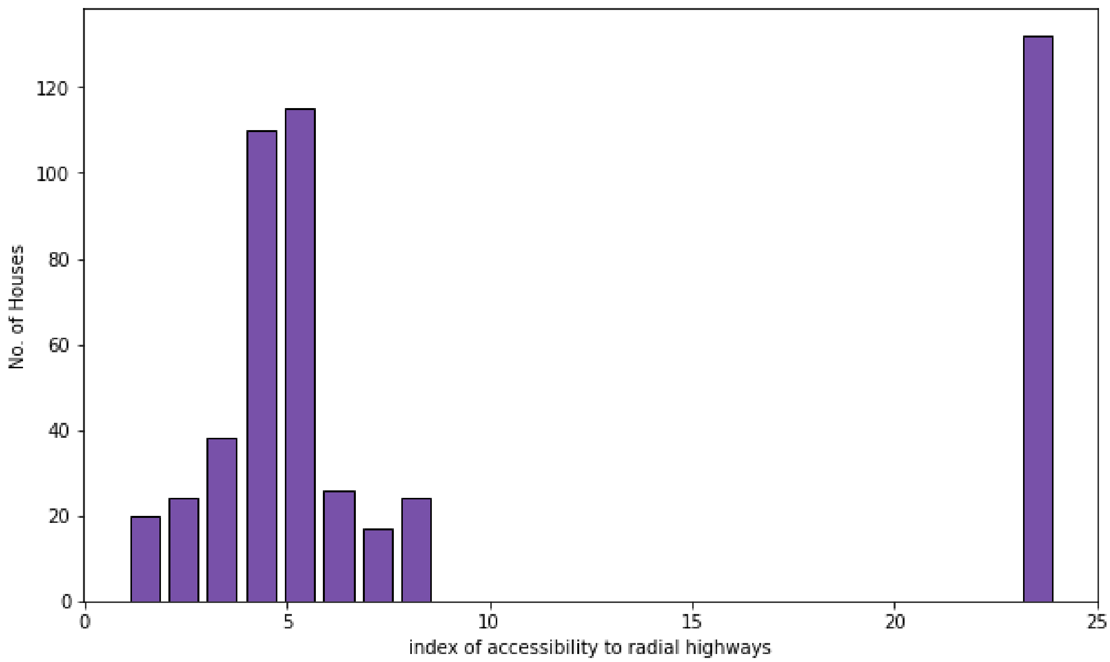
[17]:
```
plt.figure(figsize=(10,6))
plt.hist(data["RM"],ec='black',color='teal')
plt.xlabel('Average No. of rooms')
plt.ylabel('No. of Houses')
plt.show()
```



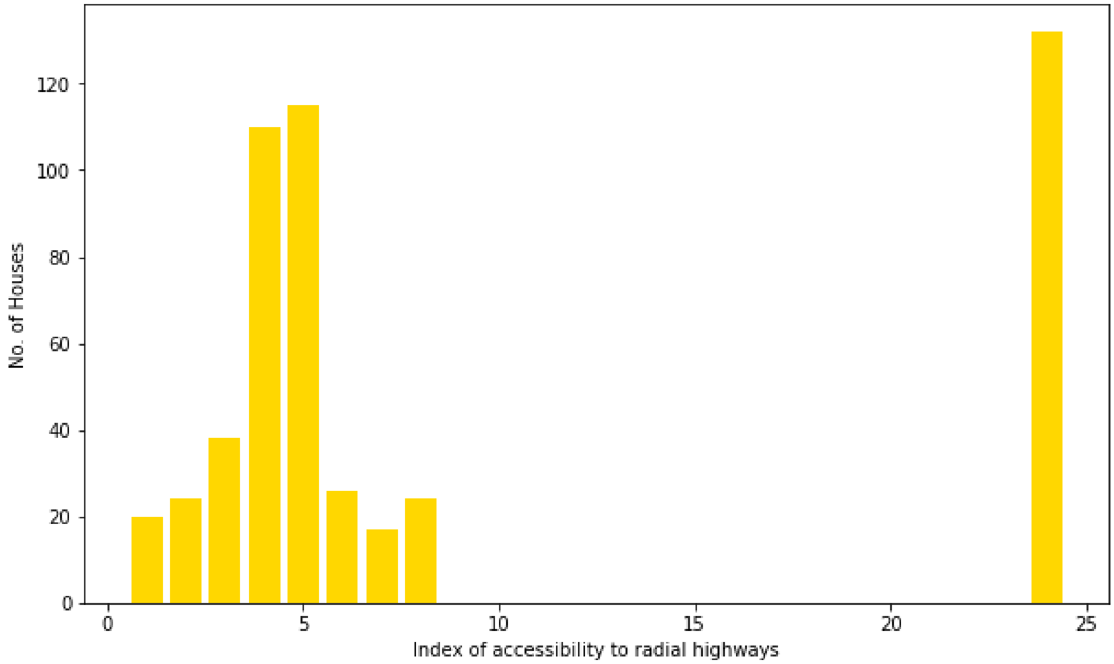
[18]:
```
data["RM"].mean()
```

[18]: 6.284634387351787

[19]:
```
# Challenge: Create a meaningful histogram for RAD using matplotlib... in royal␣
↪purple
plt.figure(figsize=(10,6))
plt.hist(data["RAD"],ec='black',color='#7851a9',bins=24,rwidth=0.75)
plt.xlabel('index of accessibility to radial highways')
plt.ylabel('No. of Houses')
plt.show()
```



[20]:
```
frequency = data["RAD"].value_counts()
# type(frequency)
# frequency.axes[0]
plt.figure(figsize=(10,6))
plt.bar(frequency.index, height=frequency,color="gold")
plt.xlabel('Index of accessibility to radial highways')
plt.ylabel('No. of Houses')
plt.show()
```



[21]:
```
data['CHAS'].value_counts() # Dummy Variable
```

[21]:
```
0.0    471
1.0     35
Name: CHAS, dtype: int64
```

2.4 Descriptive Statistics

[22]:
```
data["PRICE"].min()
```

[22]:
```
5.0
```

[23]:
```
data["PRICE"].max()
```

[23]:
```
50.0
```

[24]:
```
data.min()
```

[24]:
```
CRIM         0.00632
ZN           0.00000
INDUS        0.46000
CHAS         0.00000
NOX          0.38500
RM           3.56100
AGE          2.90000
```

```
DIS           1.12960
RAD           1.00000
TAX         187.00000
PTRATIO      12.60000
B             0.32000
LSTAT         1.73000
PRICE         5.00000
dtype: float64
```

[25]:
```python
data.max()
```

[25]:
```
CRIM        88.9762
ZN         100.0000
INDUS       27.7400
CHAS         1.0000
NOX          0.8710
RM           8.7800
AGE        100.0000
DIS         12.1265
RAD         24.0000
TAX        711.0000
PTRATIO     22.0000
B          396.9000
LSTAT       37.9700
PRICE       50.0000
dtype: float64
```

[26]:
```python
data.mean()
```

[26]:
```
CRIM         3.613524
ZN          11.363636
INDUS       11.136779
CHAS         0.069170
NOX          0.554695
RM           6.284634
AGE         68.574901
DIS          3.795043
RAD          9.549407
TAX        408.237154
PTRATIO     18.455534
B          356.674032
LSTAT       12.653063
PRICE       22.532806
dtype: float64
```

[27]:
```python
data.median()
```

[27]:

```
CRIM          0.25651
ZN            0.00000
INDUS         9.69000
CHAS          0.00000
NOX           0.53800
RM            6.20850
AGE          77.50000
DIS           3.20745
RAD           5.00000
TAX         330.00000
PTRATIO      19.05000
B           391.44000
LSTAT        11.36000
PRICE        21.20000
dtype: float64
```

[28]:

```
data.describe()
```

[28]:

```
             CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean     3.613524   11.363636   11.136779    0.069170    0.554695    6.284634
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.702617
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885500
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500
75%      3.677083   12.500000   18.100000    0.000000    0.624000    6.623500
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              AGE         DIS         RAD         TAX     PTRATIO           B  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.574901    3.795043    9.549407  408.237154   18.455534  356.674032
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.294864
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.377500
50%     77.500000    3.207450    5.000000  330.000000   19.050000  391.440000
75%     94.075000    5.188425   24.000000  666.000000   20.200000  396.225000
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

            LSTAT       PRICE
count  506.000000  506.000000
mean    12.653063   22.532806
std      7.141062    9.197104
min      1.730000    5.000000
25%      6.950000   17.025000
50%     11.360000   21.200000
75%     16.955000   25.000000
max     37.970000   50.000000
```

2.5 Correlation

2.6

$$\rho_{XY} = corr(X, Y)$$

2.7

$$-1.0 \leq \rho_{XY} \leq +1.0$$

[29]:
```
data["PRICE"].corr(data['RM'])
```

[29]:
```
0.6953599470715396
```

[30]:
```
#Challenge: Calculate the correlation between property prices and pupil teacher ratio
data['PRICE'].corr(data['PTRATIO'])
```

[30]:
```
-0.5077866855375618
```

[31]:
```
data.corr() # Pearson correlation coefficient
```

[31]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE
CRIM	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779
CHAS	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518
NOX	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470
RM	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265
AGE	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000
DIS	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881
RAD	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022
TAX	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456
PTRATIO	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515
B	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534
LSTAT	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339
PRICE	-0.388305	0.360445	-0.483725	0.175260	-0.427321	0.695360	-0.376955

	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
CRIM	-0.379670	0.625505	0.582764	0.289946	-0.385064	0.455621	-0.388305
ZN	0.664408	-0.311948	-0.314563	-0.391679	0.175520	-0.412995	0.360445
INDUS	-0.708027	0.595129	0.720760	0.383248	-0.356977	0.603800	-0.483725
CHAS	-0.099176	-0.007368	-0.035587	-0.121515	0.048788	-0.053929	0.175260
NOX	-0.769230	0.611441	0.668023	0.188933	-0.380051	0.590879	-0.427321
RM	0.205246	-0.209847	-0.292048	-0.355501	0.128069	-0.613808	0.695360
AGE	-0.747881	0.456022	0.506456	0.261515	-0.273534	0.602339	-0.376955
DIS	1.000000	-0.494588	-0.534432	-0.232471	0.291512	-0.496996	0.249929
RAD	-0.494588	1.000000	0.910228	0.464741	-0.444413	0.488676	-0.381626
TAX	-0.534432	0.910228	1.000000	0.460853	-0.441808	0.543993	-0.468536
PTRATIO	-0.232471	0.464741	0.460853	1.000000	-0.177383	0.374044	-0.507787

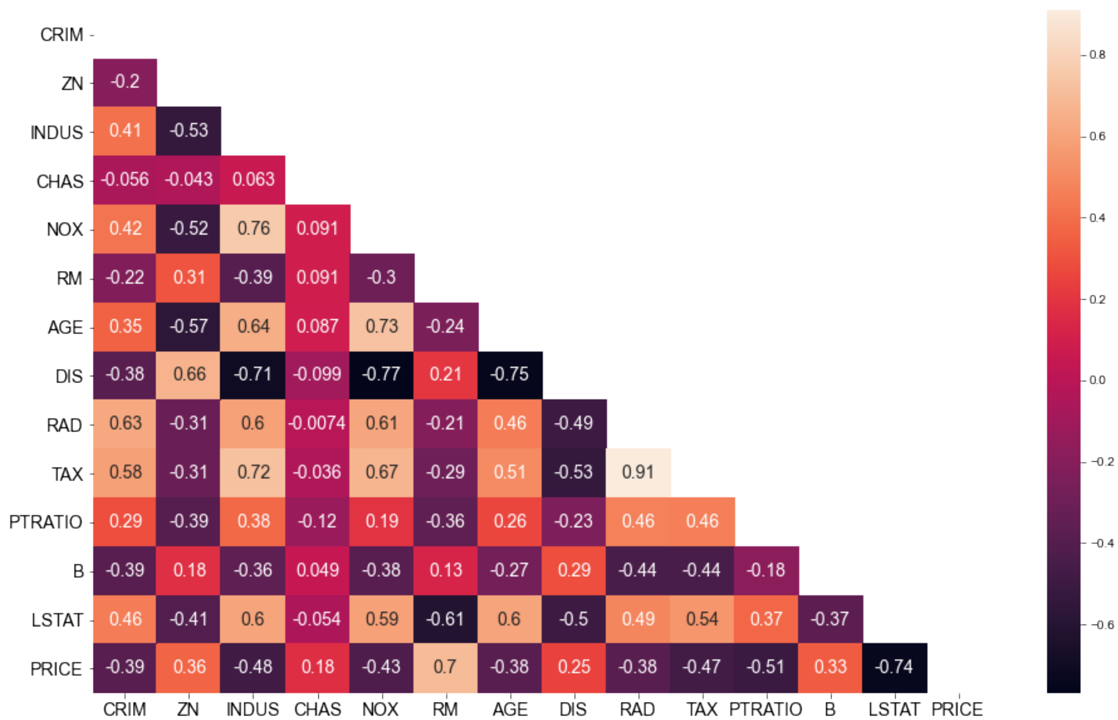
```
B         0.291512 -0.444413 -0.441808 -0.177383  1.000000 -0.366087  0.333461
LSTAT    -0.496996  0.488676  0.543993  0.374044 -0.366087  1.000000 -0.737663
PRICE     0.249929 -0.381626 -0.468536 -0.507787  0.333461 -0.737663  1.000000
```

[32]:

```
mask=np.zeros_like(data.corr())
triangle_indices=np.triu_indices_from(mask)
mask[triangle_indices]= True
```

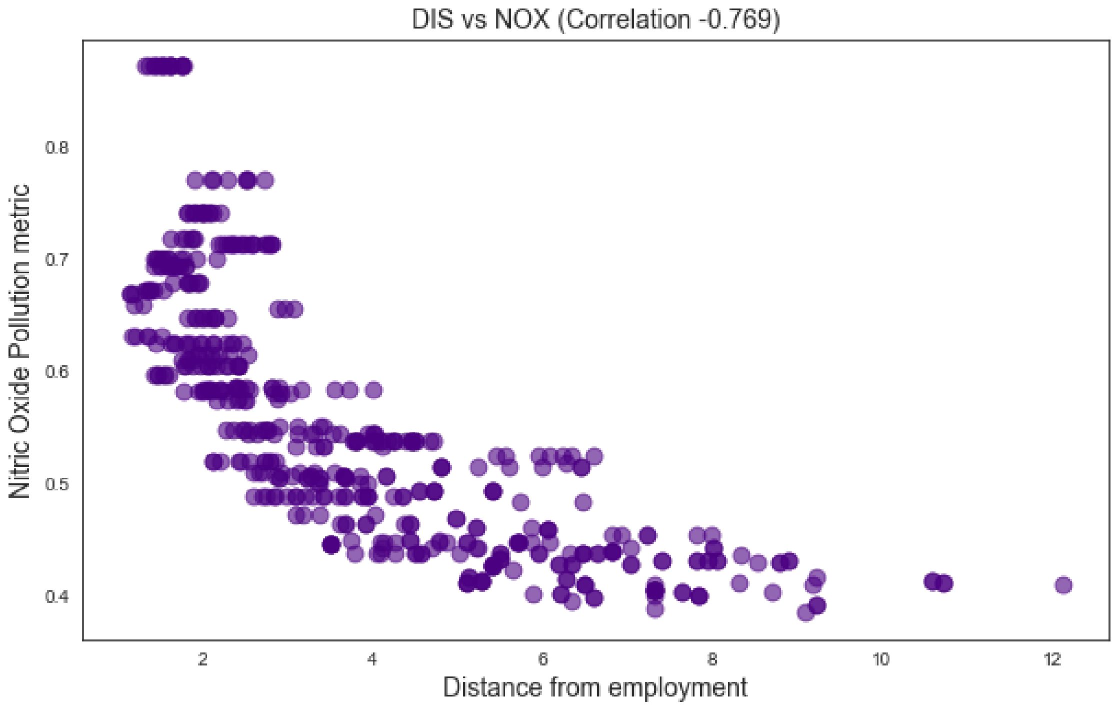
[33]:

```
plt.figure(figsize=(16,10))
sns.heatmap(data.corr(),mask=mask,annot=True,annot_kws={"size":14})
sns.set_style(style='white')
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```

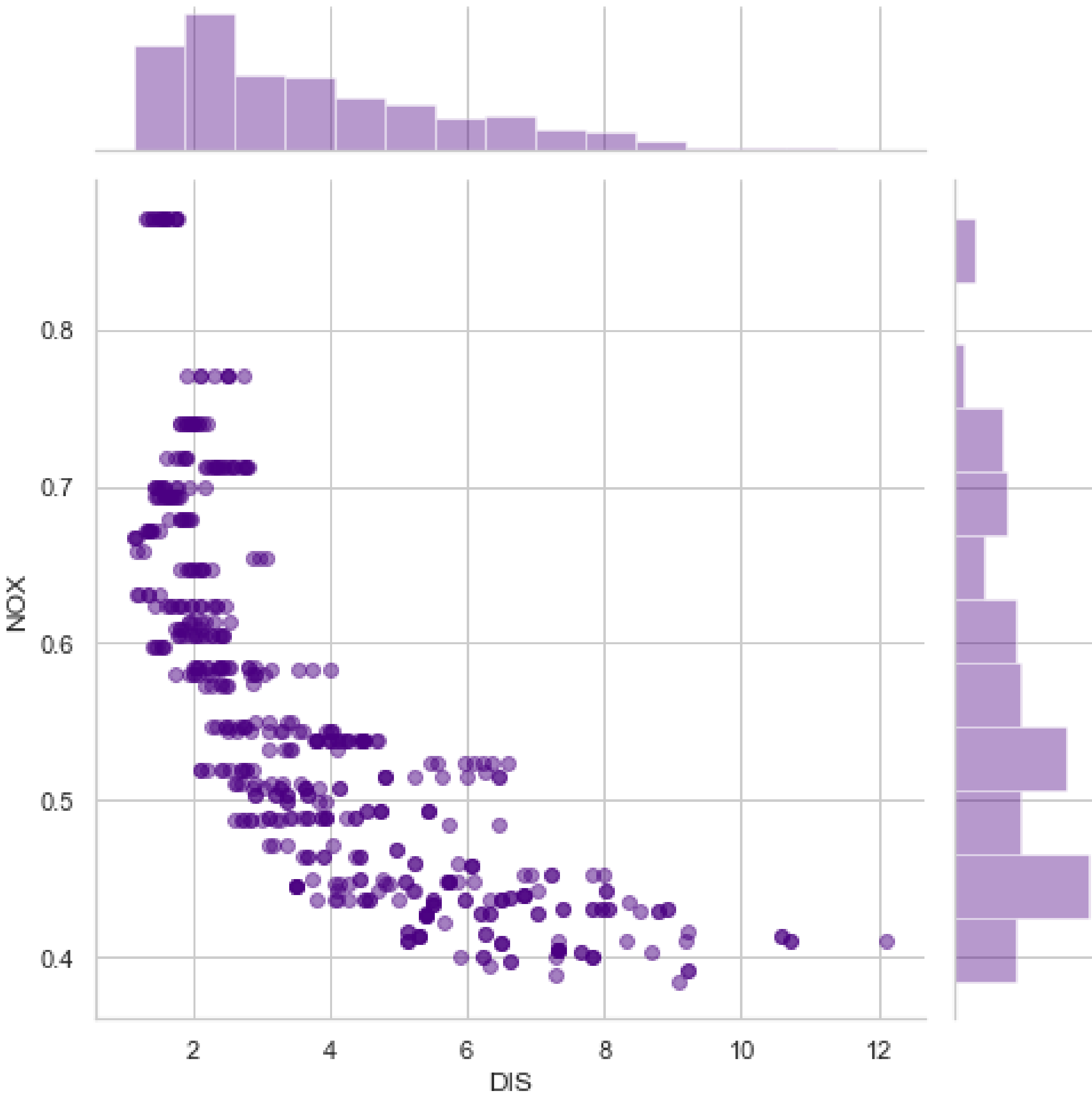


[34]:

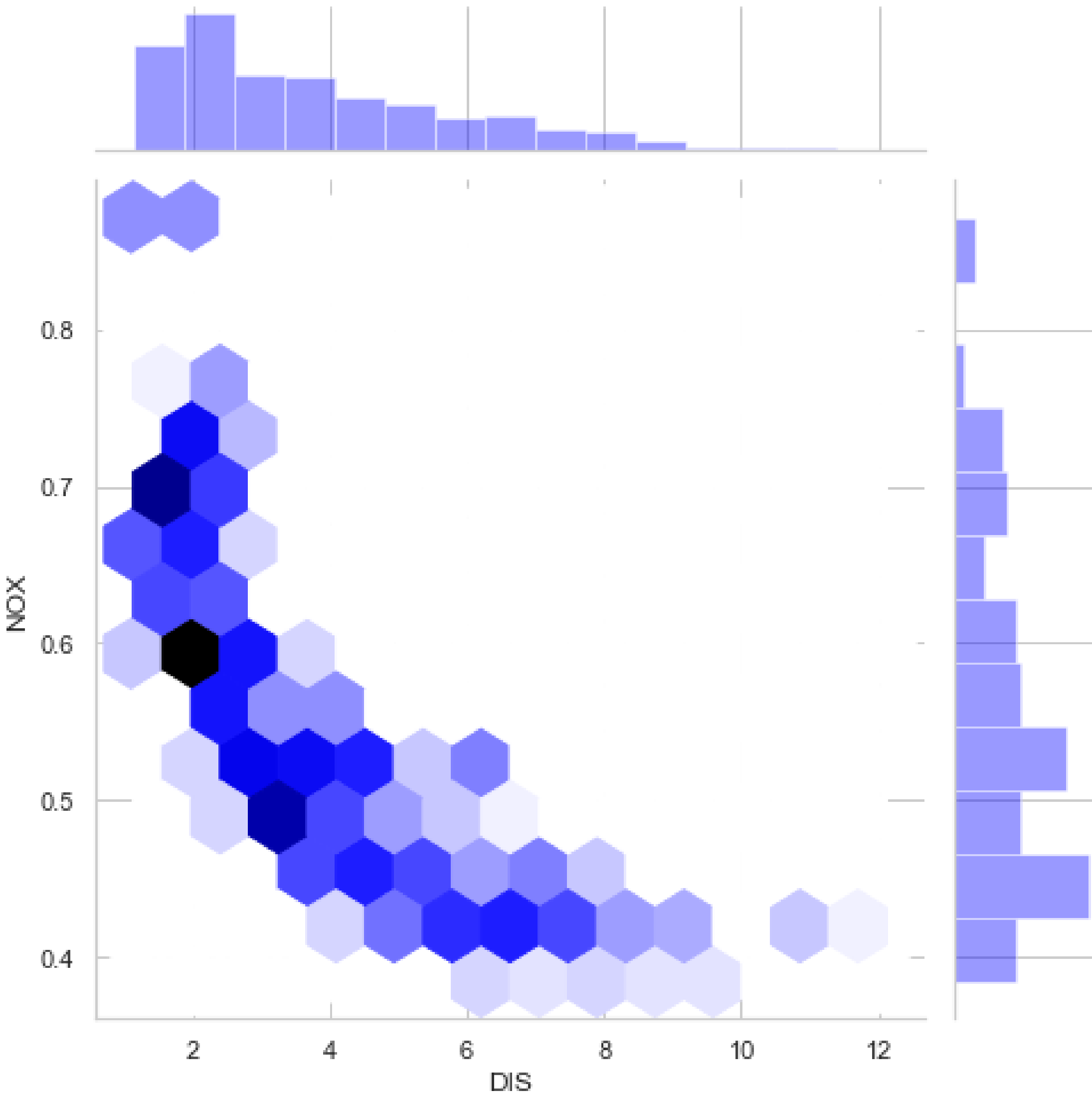
```
# Challenge: Scatter plot DIS NOX
nox_dis_corr= round(data['NOX'].corr(data['DIS']),3)
plt.figure(figsize=(10,6))
plt.scatter(data["DIS"],data["NOX"],alpha=0.6,s=80,color='indigo')
plt.title(f'DIS vs NOX (Correlation {nox_dis_corr})',fontsize=14)
plt.xlabel("Distance from employment",fontsize=14)
plt.ylabel("Nitric Oxide Pollution metric",fontsize=14)
plt.show()
```



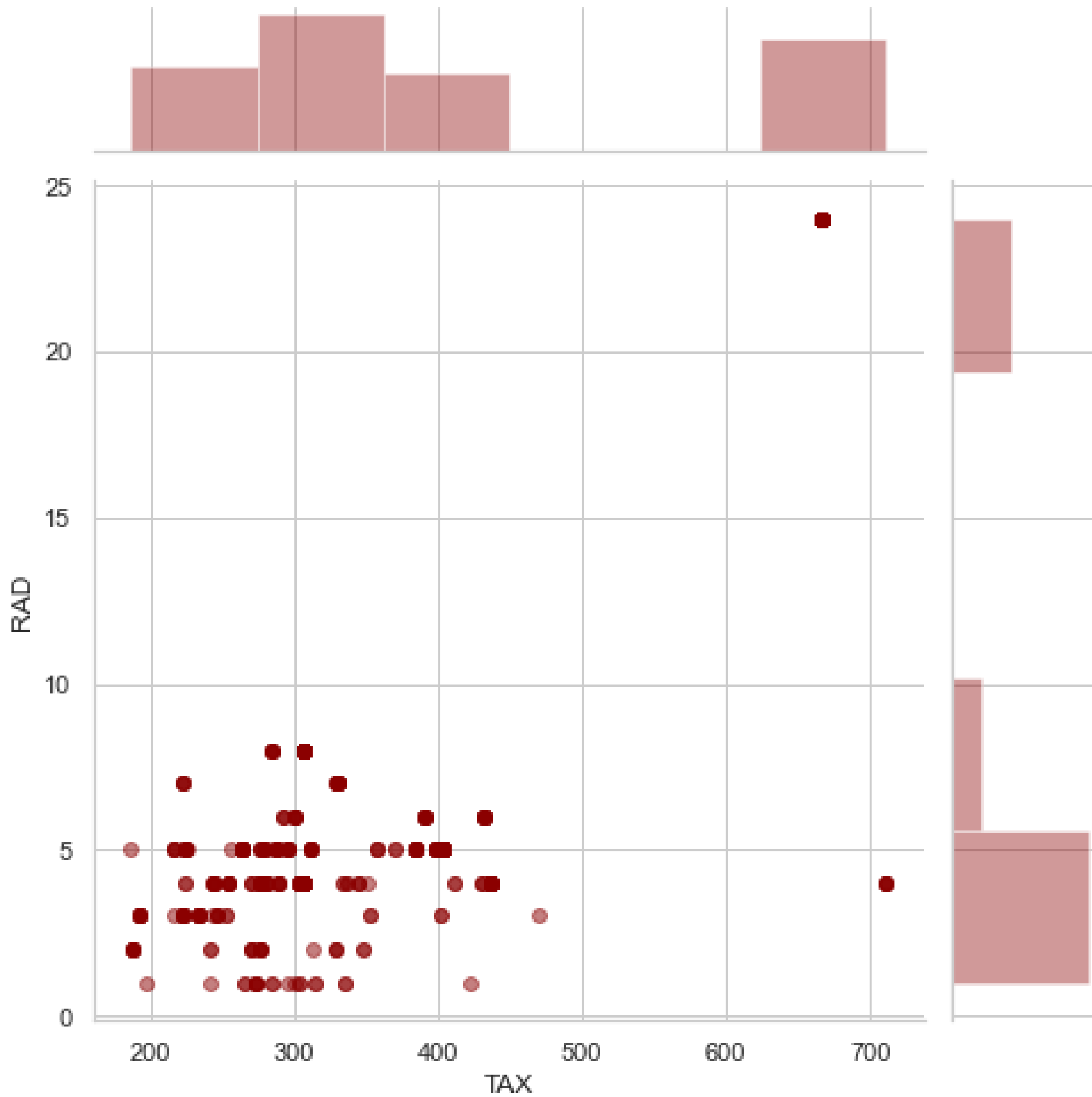
[35]:
```
sns.set_style('whitegrid')
sns.set_context('notebook')
sns.jointplot(x=data['DIS'],y=data['NOX'],height=7,joint_kws={'alpha':0.
↪5},color='indigo')
plt.show()
```



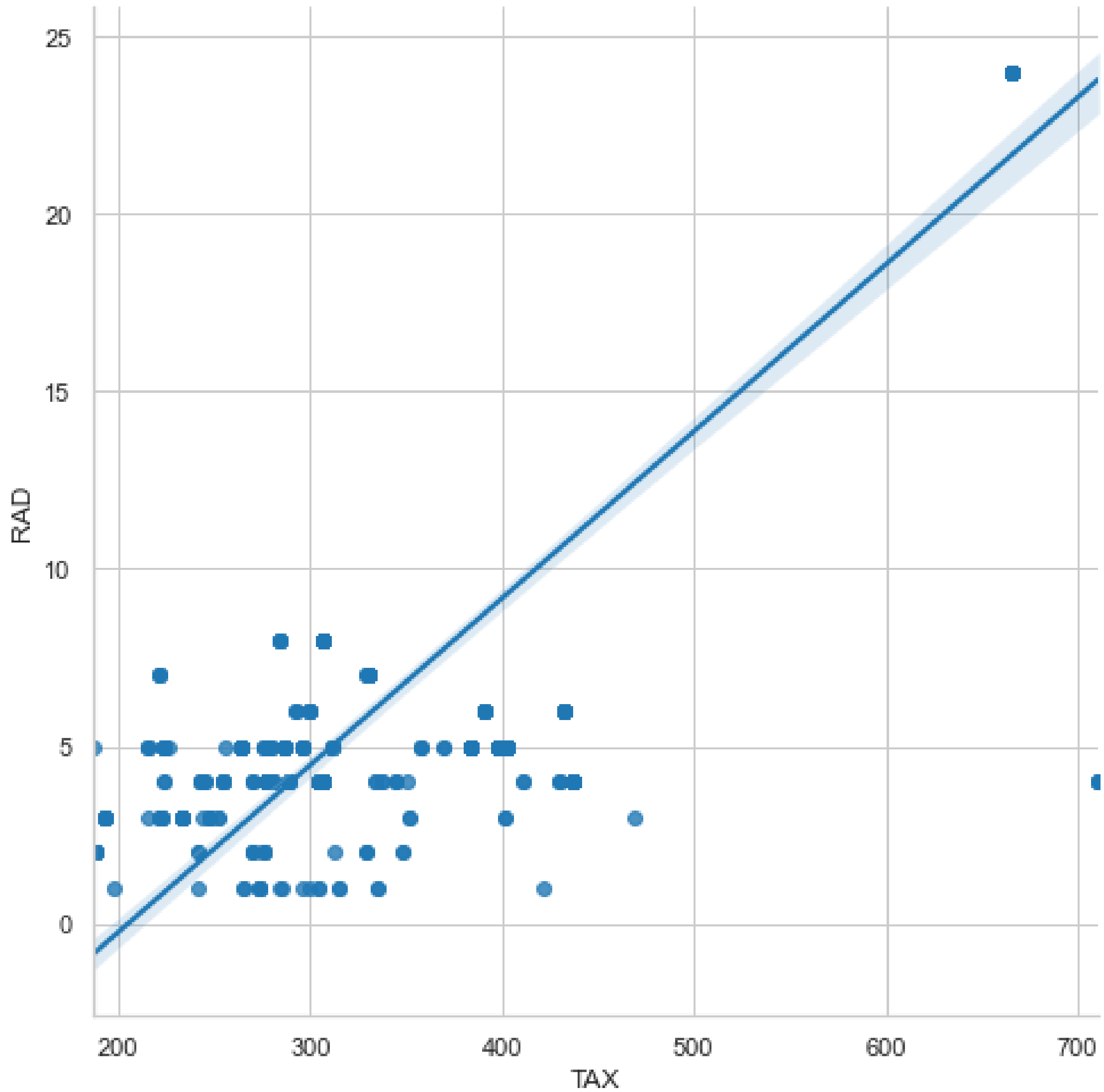
[36]:
```
sns.set_style('whitegrid')
sns.set_context('notebook')
sns.jointplot(x=data['DIS'],y=data['NOX'],height=7,kind='hex',color='blue')
plt.show()
```



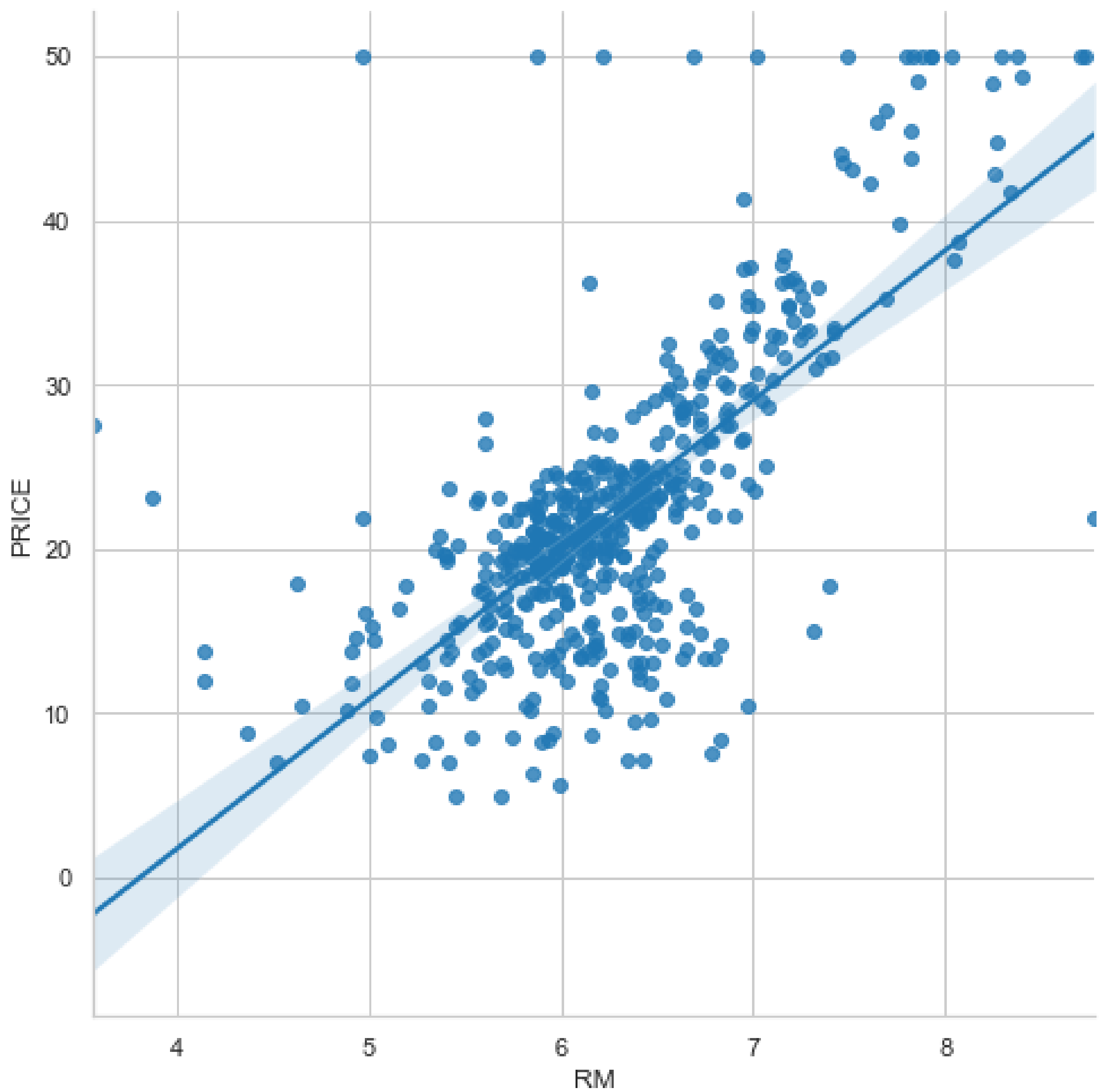
[37]:
```
sns.set_style('whitegrid')
sns.set_context('notebook')
sns.jointplot(x=data['TAX'],y=data['RAD'],height=7,joint_kws={'alpha':0.
 ↪5},color='darkred')
plt.show()
```



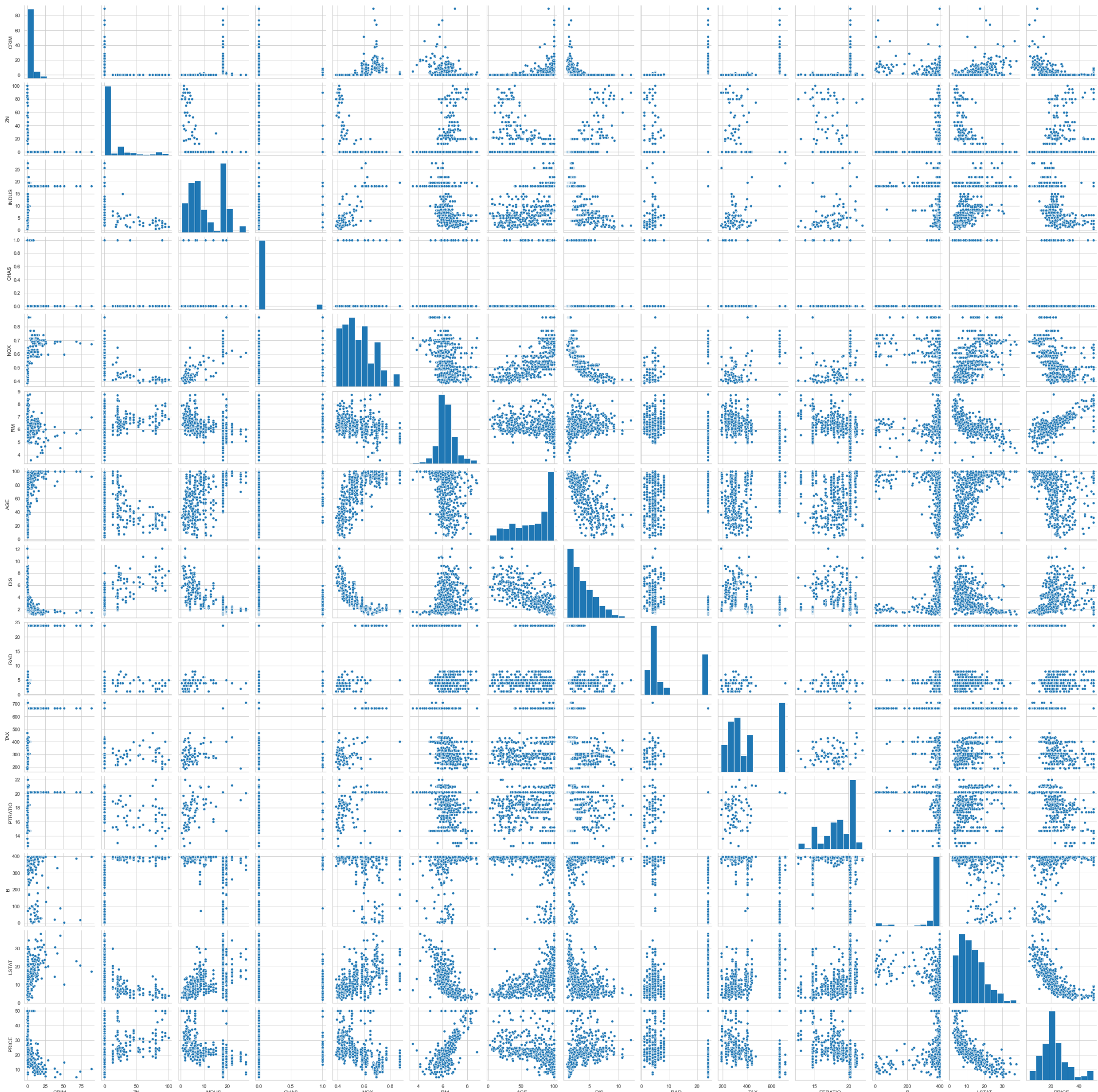
[38]:
```
sns.lmplot(x='TAX',y='RAD',data=data,height=7)
plt.show()
```



[39]:
```
#Challenge: Scatter plot RM and PRICE
sns.lmplot(x='RM',y='PRICE',data=data,height=7)
plt.show()
```

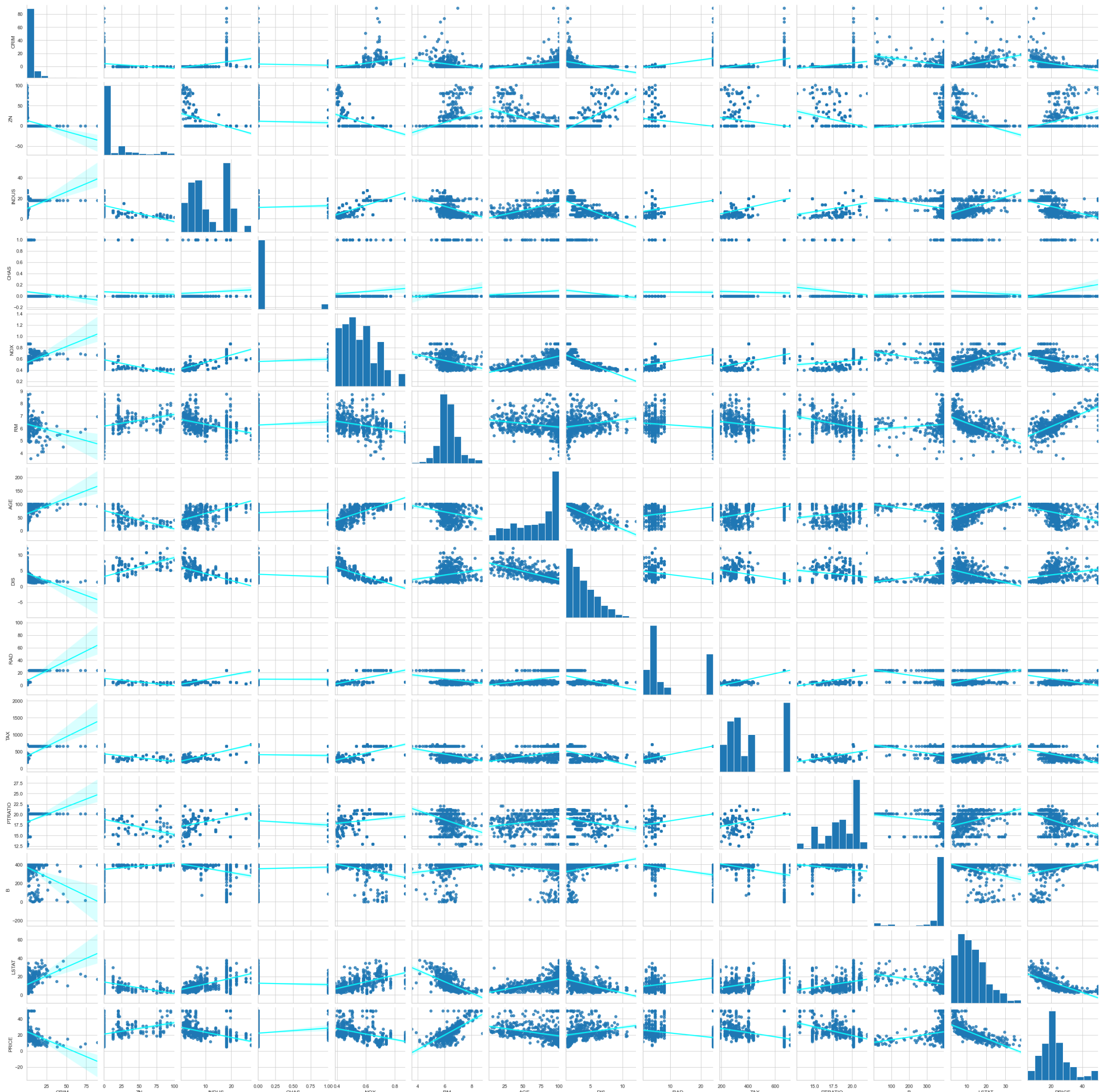


[40]:
```
%%time

sns.pairplot(data=data)
plt.show()
```



```
Wall time: 2min 11s
```

[41]:
```
%%time

sns.pairplot(data,kind='reg',plot_kws={'line_kws':{'color':'cyan'}})
plt.show()
```



Wall time: 2min 26s

2.8 Training & Test dataset split

[42]:

```
prices = data['PRICE']
features = data.drop('PRICE',axis=1)

X_train, X_test, y_train, y_test = train_test_split(features,prices,test_size=0.
 ↪2,random_state=10)

# % of training set
len(X_train)/len(features)
```

[42]: 0.7984189723320159

[43]:
```
# % of test data set
X_test.shape[0]/features.shape[0]
```

[43]: 0.2015810276679842

2.9 Multivariable Regression

[44]:
```
regr = LinearRegression()
regr.fit(X_train,y_train)

#Challenge: print out r-squared for training and test datasets
print('Training data r-squared:',regr.score(X_train,y_train))
print('Test data r-squared:',regr.score(X_test,y_test))

print('Intercept',regr.intercept_)
pd.DataFrame(data=regr.coef_,index=X_train.columns,columns=['coeff'])
```

```
Training data r-squared: 0.750121534530608
Test data r-squared: 0.6709339839115636
Intercept 36.53305138282418
```

[44]:

	coeff
CRIM	-0.128181
ZN	0.063198
INDUS	-0.007576
CHAS	1.974515
NOX	-16.271989
RM	3.108456
AGE	0.016292
DIS	-1.483014
RAD	0.303988
TAX	-0.012082
PTRATIO	-0.820306
B	0.011419
LSTAT	-0.581626

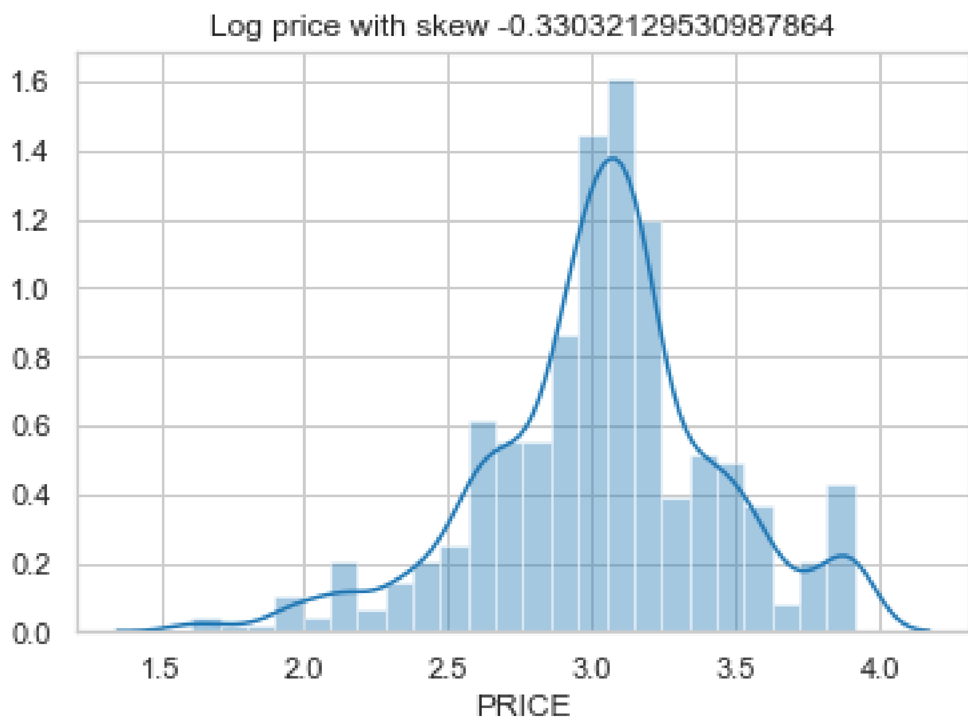
[45]:
```
y_log=np.log(data['PRICE'])
y_log.head()
y_log.tail()
```

[45]:
```
501    3.109061
502    3.025291
503    3.173878
504    3.091042
505    2.476538
Name: PRICE, dtype: float64
```

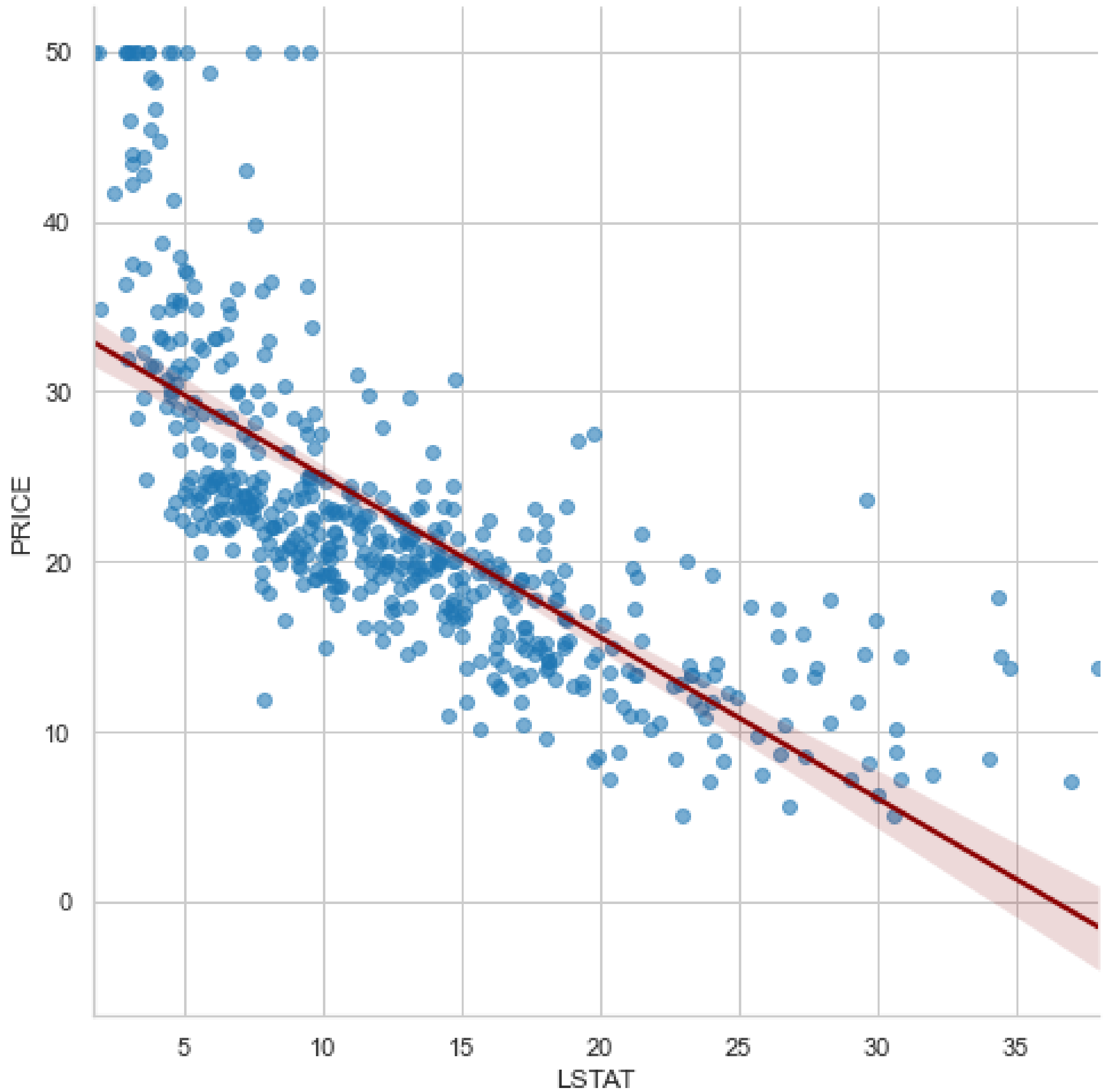
[46]:
```
y_log.skew()
```

[46]: -0.33032129530987864

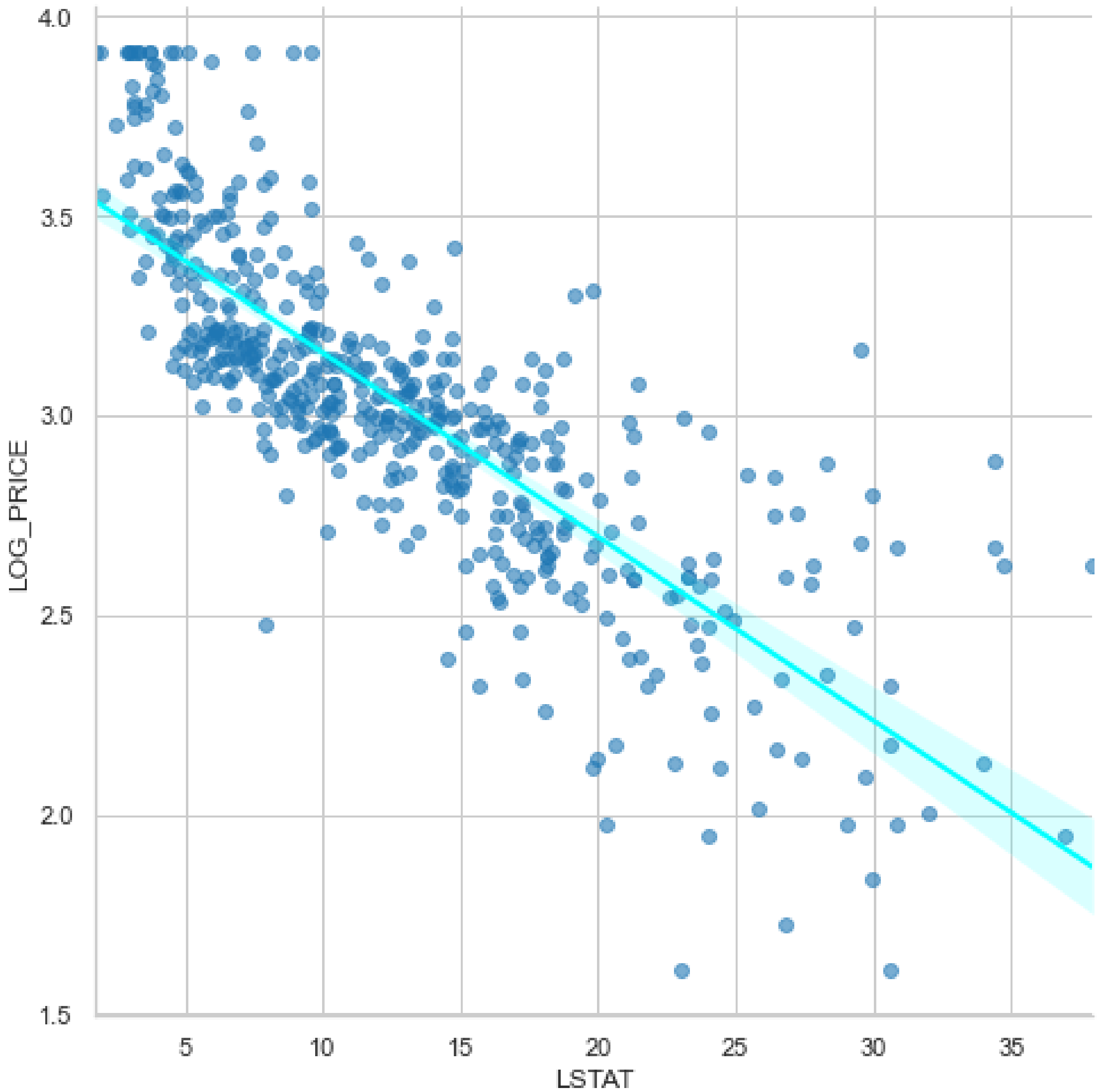
[47]:
```
sns.distplot(y_log)
plt.title(f'Log price with skew {y_log.skew()}')
plt.show()
```



[48]:
```
sns.lmplot(x='LSTAT',y='PRICE',data=data,height=7,scatter_kws={'alpha':0.
↪6},line_kws={'color':'darkred'})
plt.show()
```



[49]:
```
transformed_data = features
transformed_data['LOG_PRICE']=y_log
sns.
 ↪lmplot(x='LSTAT',y='LOG_PRICE',data=transformed_data,height=7,scatter_kws={'alpha':
 ↪0.6},line_kws={'color':'cyan'})
plt.show()
```



2.10 Regression using Log prices

[50]:
```
prices = np.log(data['PRICE'])
features = data.drop('PRICE',axis=1)

X_train, X_test, y_train, y_test = train_test_split(features,prices,test_size=0.
↪2,random_state=10)

regr = LinearRegression()
regr.fit(X_train,y_train)
```

```
print('Training data r-squared:',regr.score(X_train,y_train))
print('Test data r-squared:',regr.score(X_test,y_test))

print('Intercept',regr.intercept_)
pd.DataFrame(data=regr.coef_,index=X_train.columns,columns=['coeff'])
```

```
Training data r-squared: 0.7930234826697584
Test data r-squared: 0.7446922306260724
Intercept 4.059943871775182
```

[50]:

	coeff
CRIM	-0.010672
ZN	0.001579
INDUS	0.002030
CHAS	0.080331
NOX	-0.704068
RM	0.073404
AGE	0.000763
DIS	-0.047633
RAD	0.014565
TAX	-0.000645
PTRATIO	-0.034795
B	0.000516
LSTAT	-0.031390

[51]:
```
# Charle river property premium
np.e**0.080331
```

[51]: 1.0836456950439142

2.11 p values & Evaluating Coefficients

[52]:
```
X_incl_const = sm.add_constant(X_train)

model =sm.OLS(y_train,X_incl_const) # Ordinary Least Squares
results = model.fit()

#results.params
#results.pvalues
pd.DataFrame({'coef':results.params,'p-value':round(results.pvalues,3)})
```

[52]:

	coef	p-value
const	4.059944	0.000
CRIM	-0.010672	0.000
ZN	0.001579	0.009
INDUS	0.002030	0.445
CHAS	0.080331	0.038
NOX	-0.704068	0.000

```
RM        0.073404    0.000
AGE       0.000763    0.209
DIS      -0.047633    0.000
RAD       0.014565    0.000
TAX      -0.000645    0.000
PTRATIO  -0.034795    0.000
B         0.000516    0.000
LSTAT    -0.031390    0.000
```

2.12 Testing for Multicollinearity

$$TAX = \alpha_0 + \alpha_1 RM + \alpha_2 NOX + ... + \alpha_{12} LSTAT$$

$$VIF_{TAX} = \frac{1}{(1 - R_{TAX}^2)}$$

[53]:
```
variance_inflation_factor(exog=X_incl_const.values,exog_idx=1)
```

[53]: 1.7145250443932485

[54]:
```
# Challenge : Print the length of columns
col = X_incl_const.shape[1]
```

[55]:
```
#Challenge:write a for loop that prints out all the VIFs for all the features
for i in range(col):
    print(variance_inflation_factor(exog=X_incl_const.values,exog_idx=i))
```

```
597.5487126763895
1.7145250443932485
2.3328224265597597
3.943448822674636
1.0788133385000576
4.410320817897635
1.8404053075678573
3.3267660823099394
4.222923410477865
7.314299817005058
8.508856493040817
1.8399116326514058
1.338671325536472
2.812544292793036
```

[56]:
```
vif=[]
for i in range(col):
   vif.append(variance_inflation_factor(exog=X_incl_const.values,exog_idx=i))
print(vif)
```

```
[597.5487126763895, 1.7145250443932485, 2.3328224265597597, 3.943448822674636,
1.0788133385000576, 4.410320817897635, 1.8404053075678573, 3.3267660823099394,
```

```
4.222923410477865, 7.314299817005058, 8.508856493040817, 1.8399116326514058,
1.338671325536472, 2.812544292793036]
```

[57]:
```
vif=[ variance_inflation_factor(exog=X_incl_const.values,exog_idx=i) for i in␣
↪range(col)]
pd.DataFrame({'coef_name':X_incl_const.columns,'vif':np.round(vif,2)})
# All vif < 10 so all the props are cool with multicollinearity problem
```

[57]:

	coef_name	vif
0	const	597.55
1	CRIM	1.71
2	ZN	2.33
3	INDUS	3.94
4	CHAS	1.08
5	NOX	4.41
6	RM	1.84
7	AGE	3.33
8	DIS	4.22
9	RAD	7.31
10	TAX	8.51
11	PTRATIO	1.84
12	B	1.34
13	LSTAT	2.81

2.13 Model Simplification & the BIC

[58]:
```
# Original Model with log prices and all features

X_incl_const = sm.add_constant(X_train)

model =sm.OLS(y_train,X_incl_const)
results = model.fit()

org_coef = pd.DataFrame({'coef':results.params,'p-value':round(results.
↪pvalues,3)})

#Challenge: find and check official docs for results object and print out BIC &␣
↪r-squared
print('BIC is ',results.bic)
print('r-squared is ',results.rsquared)
```

```
BIC is  -139.74997769478898
r-squared is  0.7930234826697584
```

[59]:
```
#  reduced Model with log prices and excluding INDUS

X_incl_const = sm.add_constant(X_train)
X_incl_const = X_incl_const.drop(['INDUS'],axis=1)
```

```python
model =sm.OLS(y_train,X_incl_const)
results = model.fit()

org_minus_indus = pd.DataFrame({'coef':results.params,'p-value':round(results.
 ↪pvalues,3)})

#Challenge: find and check official docs for results object and print out BIC &␣
 ↪r-squared
print('BIC is ',results.bic)
print('r-squared is ',results.rsquared)
```

```
BIC is  -145.14508855591163
r-squared is  0.7927126289415163
```

[60]:
```python
#  reduced Model with log prices excluding age and indus

X_incl_const = sm.add_constant(X_train)
X_incl_const = X_incl_const.drop(['INDUS','AGE'],axis=1)
model =sm.OLS(y_train,X_incl_const)
results = model.fit()

reduced_coef = pd.DataFrame({'coef':results.params,'p-value':round(results.
 ↪pvalues,3)})

#Challenge: find and check official docs for results object and print out BIC &␣
 ↪r-squared
print('BIC is ',results.bic)
print('r-squared is ',results.rsquared)
```

```
BIC is  -149.49934294224678
r-squared is  0.7918657661852815
```

[61]:
```python
frames = [org_coef,org_minus_indus,reduced_coef]
pd.concat(frames,axis=1) # NaN Not a Number
```

[61]:

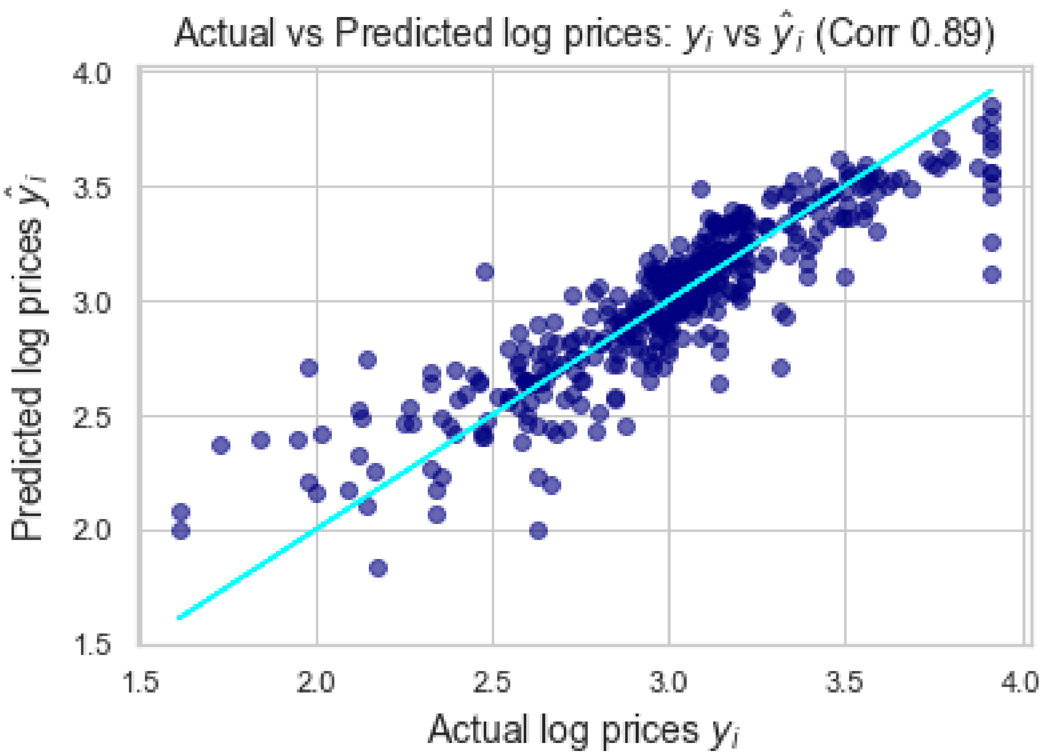
	coef	p-value	coef	p-value	coef	p-value
const	4.059944	0.000	4.056231	0.000	4.035922	0.000
CRIM	-0.010672	0.000	-0.010721	0.000	-0.010702	0.000
ZN	0.001579	0.009	0.001551	0.010	0.001461	0.014
INDUS	0.002030	0.445	NaN	NaN	NaN	NaN
CHAS	0.080331	0.038	0.082795	0.032	0.086449	0.025
NOX	-0.704068	0.000	-0.673365	0.000	-0.616448	0.000
RM	0.073404	0.000	0.071739	0.000	0.076133	0.000
AGE	0.000763	0.209	0.000766	0.207	NaN	NaN
DIS	-0.047633	0.000	-0.049394	0.000	-0.052692	0.000
RAD	0.014565	0.000	0.014014	0.000	0.013743	0.000
TAX	-0.000645	0.000	-0.000596	0.000	-0.000590	0.000
PTRATIO	-0.034795	0.000	-0.034126	0.000	-0.033481	0.000

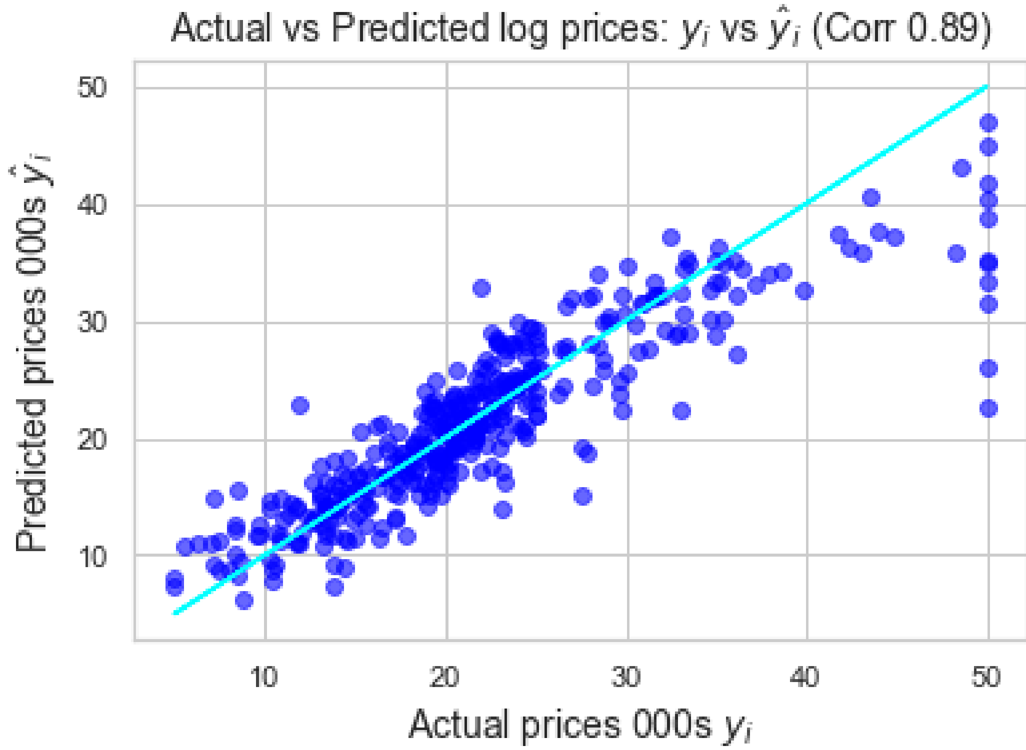
```
B         0.000516    0.000  0.000511    0.000  0.000518    0.000
LSTAT    -0.031390    0.000 -0.031262    0.000 -0.030271    0.000
```

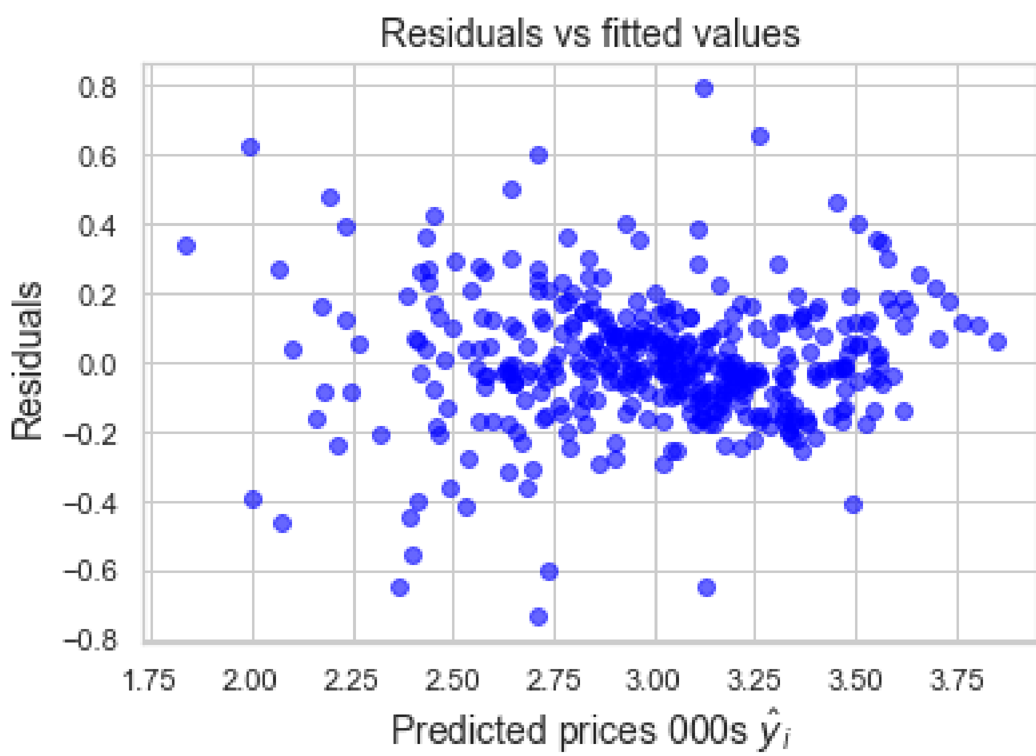
2.14 Residuals & Residual Plots

[69]:
```
# Modified model transformed using log price and simplified(dropping two␣
↪features)
prices = np.log(data['PRICE'])
features = data.drop(['PRICE','INDUS','AGE'],axis=1)

X_train, X_test, y_train, y_test = train_test_split(features,prices,test_size=0.
↪2,random_state=10)

#Using statsmodel
X_incl_const = sm.add_constant(X_train)
model =sm.OLS(y_train,X_incl_const)
results = model.fit()

#Residuals
# residuals = y_train - results.fittedvalues
# results.resid


# Graph of Actual vs Predicted prices
corr = round(y_train.corr(results.fittedvalues),2)
corr
plt.scatter(y_train,results.fittedvalues,c="navy",alpha=0.6)
plt.plot(y_train,y_train,c='cyan')
plt.xlabel('Actual log prices $y _i$',fontsize=14)
plt.ylabel('Predicted log prices $\hat y _i$',fontsize=14)
plt.title(f'Actual vs Predicted log prices: $y_i$ vs $\hat y_i$ (Corr␣
↪{corr})',fontsize=14)
plt.show()

plt.scatter(np.e**y_train,np.e**results.fittedvalues,c="blue",alpha=0.6)
plt.plot(np.e**y_train,np.e**y_train,c='cyan')
plt.xlabel('Actual prices 000s $y _i$',fontsize=14)
plt.ylabel('Predicted prices 000s $\hat y _i$',fontsize=14)
plt.title(f'Actual vs Predicted log prices: $y_i$ vs $\hat y_i$ (Corr␣
↪{corr})',fontsize=14)
plt.show()

#Residual vs Predicted values

plt.scatter(results.fittedvalues,results.resid,c="blue",alpha=0.6)
```

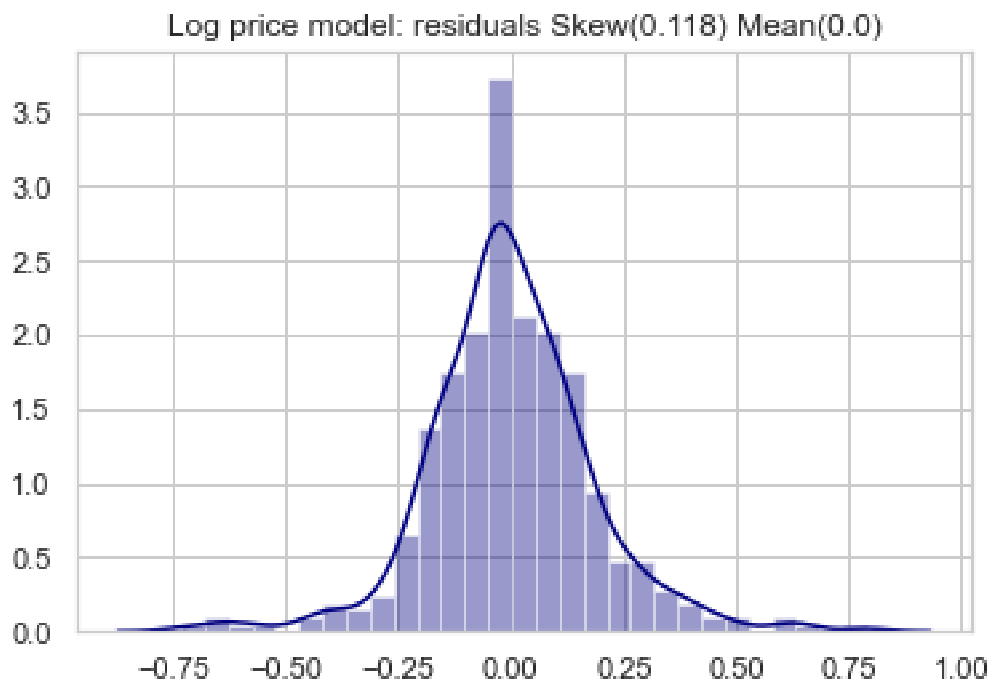
```python
plt.xlabel('Predicted prices 000s $\hat y _i$',fontsize=14)
plt.ylabel('Residuals',fontsize=14)

plt.title('Residuals vs fitted values',fontsize=14)
plt.show()

# Mean Squared error & r-squared
reduced_log_mse = round(results.mse_resid,3)
reduced_log_rsquared = round(results.rsquared,3)
```




Actual vs Predicted log prices: y_i vs $\hat{y}_i$ (Corr 0.89)
Predicted prices 000s $\hat{y}_i$
Actual prices 000s y_i
50
40
30
20
10
10
20
30
40
50

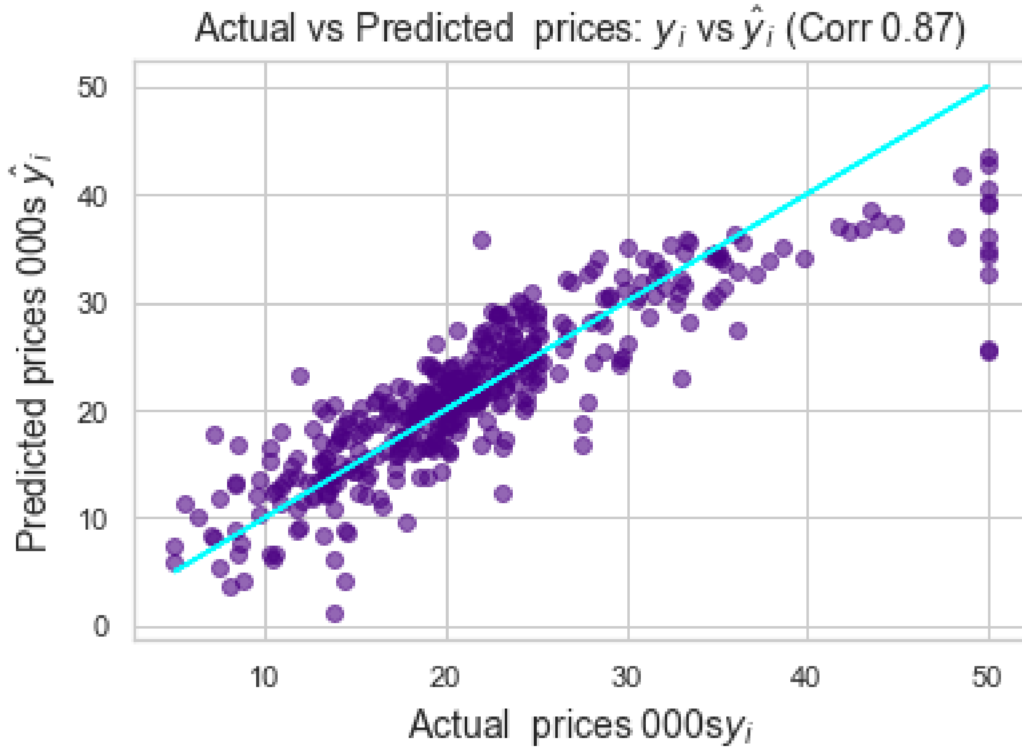

Residuals vs fitted values
Residuals
Predicted prices 000s $\hat{y}_i$
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
1.75
2.00
2.25
2.50
2.75
3.00
3.25
3.50
3.75

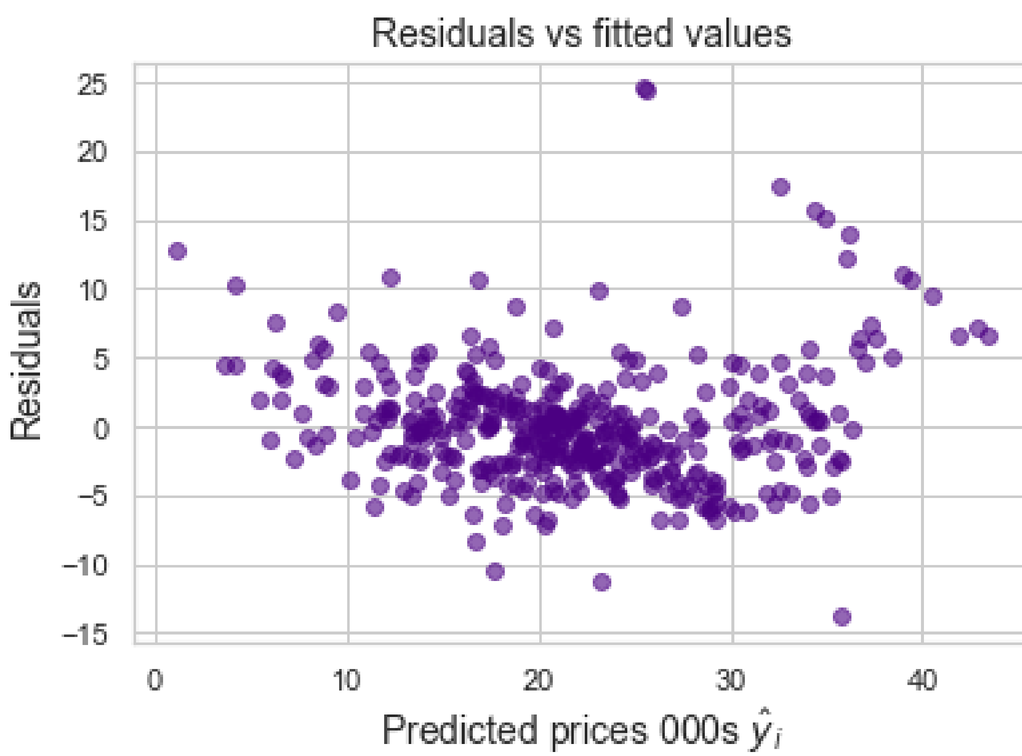
[66]:
```
# Distribution of Residuals (log prices) - checking for normality
resid_mean = round(results.resid.mean(),3)
resid_skew = round(results.resid.skew(),3)

sns.distplot(results.resid,color='navy')
plt.title(f'Log price model: residuals Skew({resid_skew}) Mean({resid_mean})')
plt.show()
```

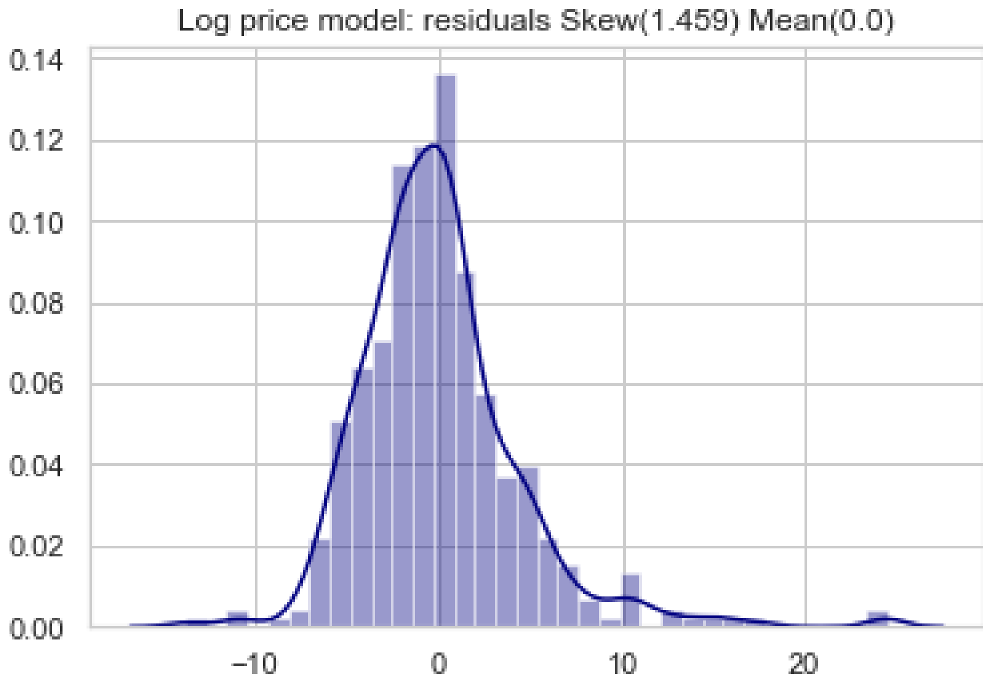


[68]:
```
# Challenge: Using the original model with all the features and normal price␣
↪generate:
# Plot of actual vs predicted prices(incl. correlation) using a different color
# Plot of residuals vs predicted prices
# Plot of distribution of residuals(incl. skew)
# Analyse the results

# Original model: normal prices and all features
prices = data['PRICE']
features = data.drop(['PRICE'],axis=1)

X_train, X_test, y_train, y_test = train_test_split(features,prices,test_size=0.
↪2,random_state=10)
```

```
X_incl_const = sm.add_constant(X_train)
model =sm.OLS(y_train,X_incl_const)
results = model.fit()




# Graph of Actual vs Predicted prices
corr = round(y_train.corr(results.fittedvalues),2)
plt.scatter(y_train,results.fittedvalues,c="indigo",alpha=0.6)
plt.plot(y_train,y_train,c='cyan')
plt.xlabel('Actual  prices 000s$y _i$',fontsize=14)
plt.ylabel('Predicted prices 000s $\hat y _i$',fontsize=14)
plt.title(f'Actual vs Predicted  prices: $y_i$ vs $\hat y_i$ (Corr␣
 ↪{corr})',fontsize=14)
plt.show()


#Residual vs Predicted values

plt.scatter(results.fittedvalues,results.resid,c="indigo",alpha=0.6)


plt.xlabel('Predicted prices 000s $\hat y _i$',fontsize=14)
plt.ylabel('Residuals',fontsize=14)

plt.title('Residuals vs fitted values',fontsize=14)
plt.show()

#Residual distribution Chart
resid_mean = round(results.resid.mean(),3)
resid_skew = round(results.resid.skew(),3)

sns.distplot(results.resid,color='navy')
plt.title(f'Log price model: residuals Skew({resid_skew}) Mean({resid_mean})')
plt.show()

# Mean Squared error & r-squared
full_normal_mse = round(results.mse_resid,3)
full_normal_rsquared = round(results.rsquared,3)
```

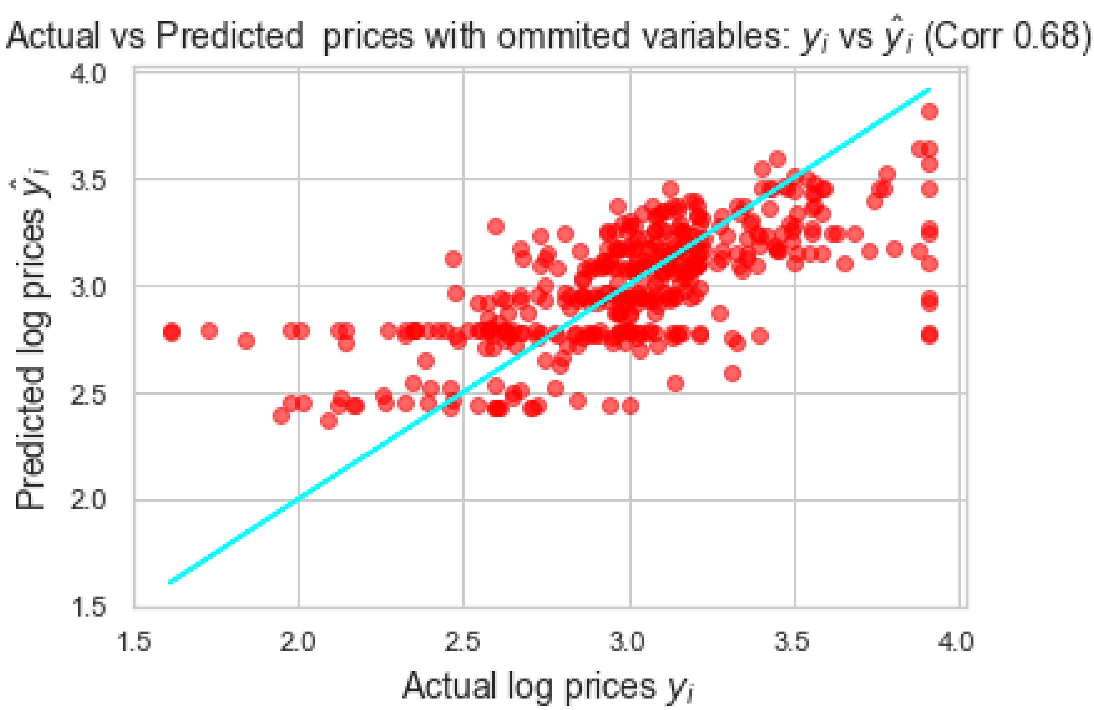

Actual vs Predicted prices: y_i vs $\hat{y}_i$ (Corr 0.87)
Predicted prices 000s $\hat{y}_i$
Actual prices 000sy_i
0
10
20
30
40
50

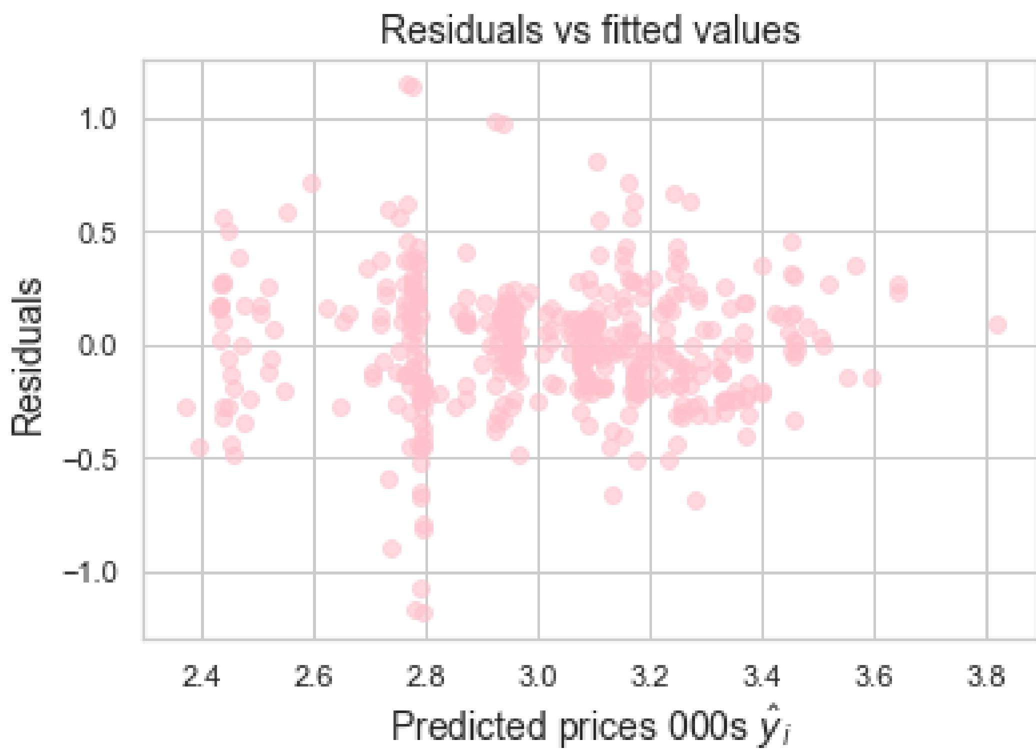

Residuals vs fitted values
Residuals
Predicted prices 000s $\hat{y}_i$
25
20
15
10
5
0
−5
−10
−15
0
10
20
30
40



[70]:
```
# Challenge: Using the original model with all the features and normal price␣
↪generate:
# Plot of actual vs predicted prices(incl. correlation) using a different color
# Plot of residuals vs predicted prices
# Plot of distribution of residuals(incl. skew)
# Analyse the results

#Model ommitting key features and using log price
prices =np.log(data['PRICE'])
features = data.drop(['PRICE','INDUS','LSTAT','AGE','RM','NOX','CRIM'],axis=1)

X_train, X_test, y_train, y_test = train_test_split(features,prices,test_size=0.
↪2,random_state=10)


X_incl_const = sm.add_constant(X_train)
model =sm.OLS(y_train,X_incl_const)
results = model.fit()
```

```
# Graph of Actual vs Predicted prices
corr = round(y_train.corr(results.fittedvalues),2)
plt.scatter(y_train,results.fittedvalues,c="red",alpha=0.6)
plt.plot(y_train,y_train,c='cyan')
plt.xlabel('Actual log prices $y _i$',fontsize=14)
plt.ylabel('Predicted log prices $\hat y _i$',fontsize=14)
plt.title(f'Actual vs Predicted  prices with ommited variables: $y_i$ vs $\hat␣
↪y_i$ (Corr {corr})',fontsize=14)
plt.show()


#Residual vs Predicted values

plt.scatter(results.fittedvalues,results.resid,c="pink",alpha=0.6)


plt.xlabel('Predicted prices 000s $\hat y _i$',fontsize=14)
plt.ylabel('Residuals',fontsize=14)

plt.title('Residuals vs fitted values',fontsize=14)
plt.show()

# Mean Squared error & r-squared
omitted_var_mse = round(results.mse_resid,3)
omitted_var_rsquared = round(results.rsquared,3)
```





[74]:
```
pd.DataFrame({'R-squared':
↪[reduced_log_rsquared,full_normal_rsquared,omitted_var_rsquared],'MSE':
↪[reduced_log_mse,full_normal_mse,omitted_var_mse],'RMSE':np.
↪sqrt([reduced_log_mse,full_normal_mse,omitted_var_mse])},index=['Reduced Log␣
↪Model','Full Normal Price Model','Omitted Variable Model'])
```

[74]:

	R-squared	MSE	RMSE
Reduced Log Model	0.792	0.035	0.187083
Full Normal Price Model	0.750	19.921	4.463295
Omitted Variable Model	0.460	0.090	0.300000

[85]:
```
#Challenge: Our estimate for a house id $30000 calculate upperbound and lower␣
↪bound for a 95% prediction interval using the reduced log model
print('1 std in log price is',np.sqrt(reduced_log_mse))
print('2 std in log price is',2*np.sqrt(reduced_log_mse))
upper_bound=np.log(30) + 2*np.sqrt(reduced_log_mse)
print('The upper bound for a 95% prediction interval in log prices␣
↪is',upper_bound)
print('The upper bound for a 95% prediction interval in normal prices is',(np.
↪e**upper_bound)*1000)
```

```
lower_bound=np.log(30) - 2*np.sqrt(reduced_log_mse)
print('The lower bound for a 95% prediction interval in log prices␣
↪is',lower_bound)
print('The lower bound for a 95% prediction interval in normal prices is',(np.
↪e**lower_bound)*1000)
```

```
1 std in log price is 0.18708286933869708
2 std in log price is 0.37416573867739417
The upper bound for a 95% prediction interval in log prices is
3.7753631203395495
The upper bound for a 95% prediction interval in normal prices is
43613.34233239937
The lower bound for a 95% prediction interval in log prices is
3.0270316429847615
The lower bound for a 95% prediction interval in normal prices is
20635.886906824155
```

[ ]:

```

```