# Multivariable_Regression_and_Valuation_Model

November 19, 2025

## 1 Setup and Context

#### 1.0.1 Introduction

Welcome to Boston Massachusetts in the 1970s! Imagine you're working for a real estate development company. Your company wants to value any residential project before they start. You are tasked with building a model that can provide a price estimate based on a home's characteristics like: * The number of rooms * The distance to employment centres * How rich or poor the area is * How many students there are per teacher in local schools etc

To accomplish your task you will:

1. Analyse and explore the Boston house price data
2. Split your data for training and testing
3. Run a Multivariable Regression
4. Evaluate how your model's coefficients and residuals
5. Use data transformation to improve your model performance
6. Use your model to estimate a property price

#### 1.0.2 Upgrade plotly (only Google Colab Notebook)

Google Colab may not be running the latest version of plotly. If you're working in Google Colab, uncomment the line below, run the cell, and restart your notebook server.

[43]:
```
# pip install --upgrade plotly
```

#### 1.0.3 Import Statements

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

import seaborn as sns
import plotly.express as px
import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

### 1.0.4 Notebook Presentation

[2]:
```python
pd.options.display.float_format = '{:,.2f}'.format
```

## 2 Load the Data

The first column in the .csv file just has the row numbers, so it will be used as the index.

[3]:
```python
data = pd.read_csv('boston.csv', index_col=0)
```

### 2.0.1 Understand the Boston House Price Dataset

---

**Characteristics:**

```
:Number of Instances: 506

:Number of Attributes: 13 numeric/categorical predictive. The Median Value (attribute 14) is th

:Attribute Information (in order):
    1. CRIM     per capita crime rate by town
    2. ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
    3. INDUS    proportion of non-retail business acres per town
    4. CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
    5. NOX      nitric oxides concentration (parts per 10 million)
    6. RM       average number of rooms per dwelling
    7. AGE      proportion of owner-occupied units built prior to 1940
    8. DIS      weighted distances to five Boston employment centres
    9. RAD      index of accessibility to radial highways
    10. TAX      full-value property-tax rate per $10,000
    11. PTRATIO  pupil-teacher ratio by town
    12. B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
    13. LSTAT    % lower status of the population
    14. PRICE     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. This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University. You can find the original research paper here.

## 3 Preliminary Data Exploration

**Challenge**

- What is the shape of `data`?
- How many rows and columns does it have?
- What are the column names?

- Are there any NaN values or duplicates?

[4]:
```
data.shape
```

[4]:
```
(506, 14)
```

[5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 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: 59.3 KB
```

[6]:
```
data.head()
```

[6]:
```
   CRIM    ZN  INDUS  CHAS  NOX   RM   AGE  DIS  RAD    TAX  PTRATIO      B  \
0  0.01 18.00   2.31  0.00 0.54 6.58 65.20 4.09 1.00 296.00    15.30 396.90
1  0.03  0.00   7.07  0.00 0.47 6.42 78.90 4.97 2.00 242.00    17.80 396.90
2  0.03  0.00   7.07  0.00 0.47 7.18 61.10 4.97 2.00 242.00    17.80 392.83
3  0.03  0.00   2.18  0.00 0.46 7.00 45.80 6.06 3.00 222.00    18.70 394.63
4  0.07  0.00   2.18  0.00 0.46 7.15 54.20 6.06 3.00 222.00    18.70 396.90

   LSTAT  PRICE
0   4.98  24.00
1   9.14  21.60
2   4.03  34.70
3   2.94  33.40
4   5.33  36.20
```

[7]:
```
data.tail()
```

```
[7]:       CRIM   ZN  INDUS  CHAS  NOX   RM   AGE  DIS  RAD    TAX  PTRATIO      B  \
     501  0.06 0.00  11.93  0.00 0.57 6.59 69.10 2.48 1.00 273.00    21.00 391.99
     502  0.05 0.00  11.93  0.00 0.57 6.12 76.70 2.29 1.00 273.00    21.00 396.90
     503  0.06 0.00  11.93  0.00 0.57 6.98 91.00 2.17 1.00 273.00    21.00 396.90
     504  0.11 0.00  11.93  0.00 0.57 6.79 89.30 2.39 1.00 273.00    21.00 393.45
     505  0.05 0.00  11.93  0.00 0.57 6.03 80.80 2.50 1.00 273.00    21.00 396.90

          LSTAT  PRICE
     501   9.67  22.40
     502   9.08  20.60
     503   5.64  23.90
     504   6.48  22.00
     505   7.88  11.90
```

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

```
[8]: 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
```

### 3.1 Data Cleaning - Check for Missing Values and Duplicates

```
[21]: print(f'Are there any NaN values? : {data.isna().values.any()}')
```

```
Are there any NaN values? : False
```

```
[22]: print(f'Are there any duplicated values? : {data.duplicated().values.any()}')
```

```
Are there any duplicated values? : False
```

### 3.2 Descriptive Statistics

- How many students are there per teacher on average?
- What is the average price of a home in the dataset?
- What is the `CHAS` feature?
- What are the minimum and the maximum value of the `CHAS` and why?
- What is the maximum and the minimum number of rooms per dwelling in the dataset?

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

```
[23]:         CRIM     ZN  INDUS   CHAS    NOX     RM    AGE    DIS    RAD    TAX  \
count  506.00 506.00 506.00 506.00 506.00 506.00 506.00 506.00 506.00 506.00
mean     3.61  11.36  11.14   0.07   0.55   6.28  68.57   3.80   9.55 408.24
std      8.60  23.32   6.86   0.25   0.12   0.70  28.15   2.11   8.71 168.54
min      0.01   0.00   0.46   0.00   0.39   3.56   2.90   1.13   1.00 187.00
25%      0.08   0.00   5.19   0.00   0.45   5.89  45.02   2.10   4.00 279.00
50%      0.26   0.00   9.69   0.00   0.54   6.21  77.50   3.21   5.00 330.00
75%      3.68  12.50  18.10   0.00   0.62   6.62  94.07   5.19  24.00 666.00
max     88.98 100.00  27.74   1.00   0.87   8.78 100.00  12.13  24.00 711.00

       PTRATIO      B  LSTAT  PRICE
count   506.00 506.00 506.00 506.00
mean     18.46 356.67  12.65  22.53
std       2.16  91.29   7.14   9.20
min      12.60   0.32   1.73   5.00
25%      17.40 375.38   6.95  17.02
50%      19.05 391.44  11.36  21.20
75%      20.20 396.23  16.96  25.00
max      22.00 396.90  37.97  50.00
```

```
[32]: print(f'There are {round(data.PTRATIO.mean())} teachers per student on average')
```

```
There are 18 teachers per student on average
```

```
[34]: print(f'The average price of a home is ${data.PRICE.mean() * 1000:.2f} ')
```

```
The average price of a home is $22532.81
```

CHAS is a dummy variable that shows whether the home is next to the Charles River or not. As such, it only has the value 0 or 1.

```
[83]: print(f'The maximum number of rooms per dwelling is {data.RM.max()}')
```

```
The maximum number of rooms per dwelling is 8.78
```

```
[84]: print(f'The minimum number of rooms per dwelling is {data.RM.min()}')
```

```
The minimum number of rooms per dwelling is 3.561
```

### 3.3 Visualise the Features

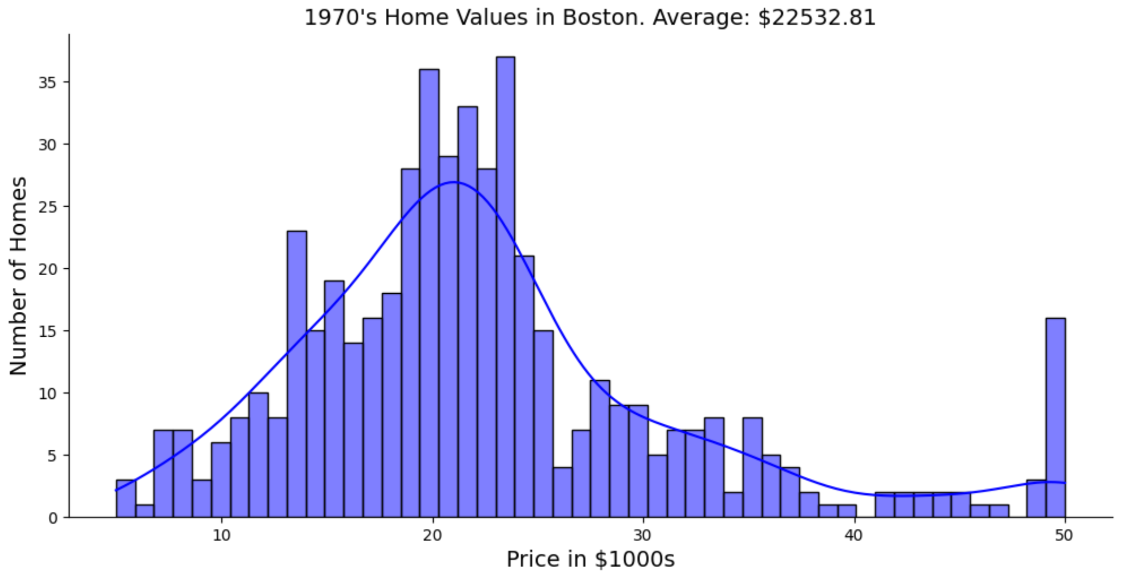
**Challenge**: Having looked at some descriptive statistics, visualise the data for your model. Use Seaborn's `.displot()` to create a bar chart and superimpose the Kernel Density Estimate (KDE) for the following variables: * PRICE: The home price in thousands. * RM: the average number of rooms per owner unit. * DIS: the weighted distance to the 5 Boston employment centres i.e., the estimated length of the commute. * RAD: the index of accessibility to highways.

Try setting the `aspect` parameter to `2` for a better picture.

What do you notice in the distributions of the data?

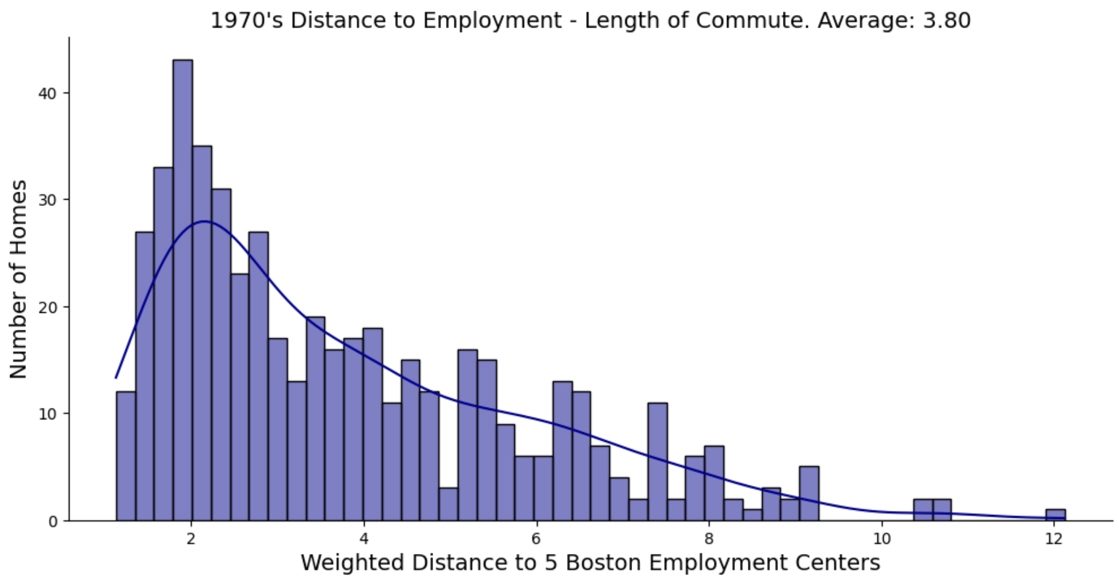
**House Prices**

[85]:
```
sns.displot(data.PRICE,
            bins = 50,
            aspect = 2,
            kde = True,
            color = 'blue')

plt.title(f"1970's Home Values in Boston. Average: ${data.PRICE.mean() * 1000:.
  ↪2f}", fontsize = 14)
plt.xlabel(f'Price in $1000s', fontsize = 14)
plt.ylabel(f'Number of Homes', fontsize = 14)

plt.show()
```



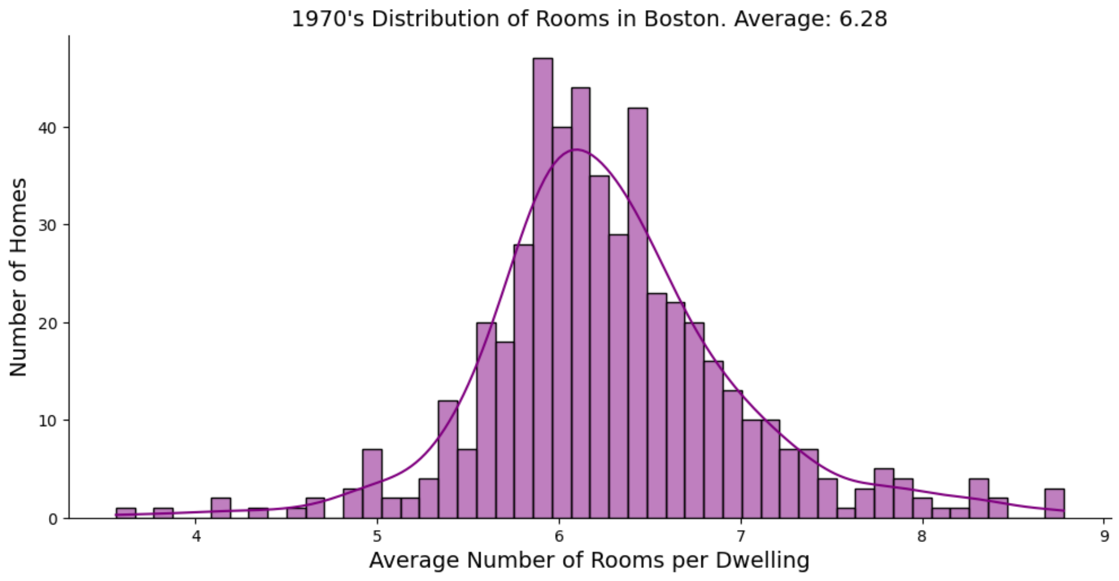
**Distance to Employment - Length of Commute**

[87]:
```
sns.displot(data.DIS,
            bins = 50,
            aspect = 2,
            kde = True,
            color = 'darkblue')

plt.title(f"1970's Distance to Employment - Length of Commute. Average: {data.
  ↪DIS.mean():.2f}", fontsize = 14)
plt.xlabel(f'Weighted Distance to 5 Boston Employment Centers', fontsize = 14)
plt.ylabel(f'Number of Homes', fontsize = 14)
plt.show()
```



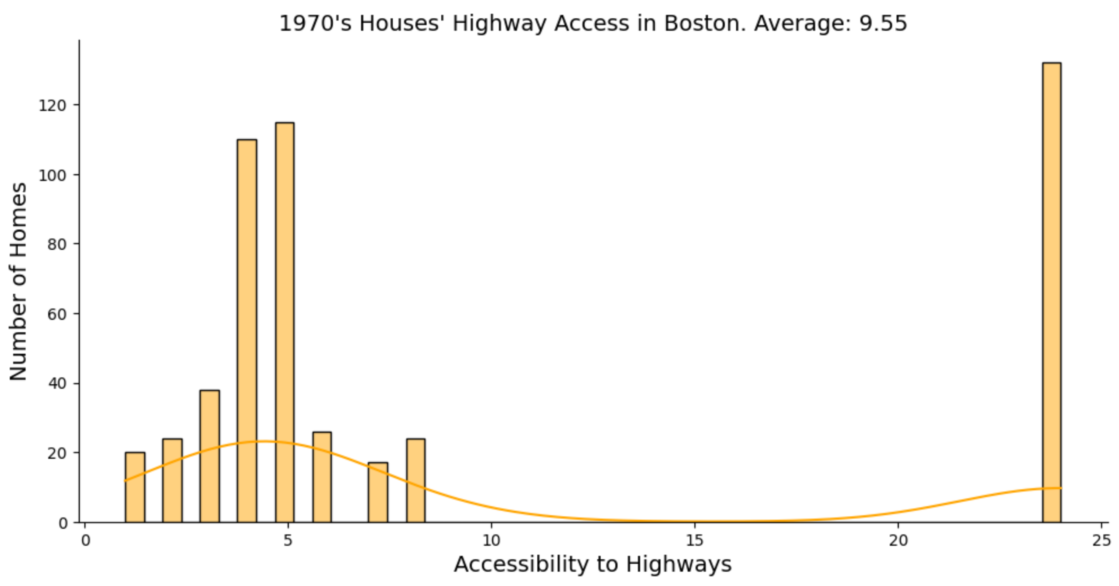
### Number of Rooms

[94]:
```
sns.displot(data.RM,
            bins = 50,
            aspect = 2,
            kde = True,
            color = 'purple')

plt.title(f"1970's Distribution of Rooms in Boston. Average: {data.RM.mean():.
 ↪2f}", fontsize = 14)
plt.xlabel(f'Average Number of Rooms per Dwelling', fontsize = 14)
plt.ylabel(f'Number of Homes', fontsize = 14)

plt.show()
```



**Access to Highways**

[96]:
```
sns.displot(data.RAD,
            bins = 50,
            aspect = 2,
            kde = True,
            color = 'orange')

plt.title(f"1970's Houses' Highway Access in Boston. Average: {data.RAD.mean():.
 ↪2f}", fontsize = 14)
plt.xlabel(f'Accessibility to Highways', fontsize = 14)
plt.ylabel(f'Number of Homes', fontsize = 14)

plt.show()
```



**Next to the River? Challenge**

Create a bar chart with plotly for CHAS to show many more homes are away from the river versus next to it. The bar chart should look something like this:

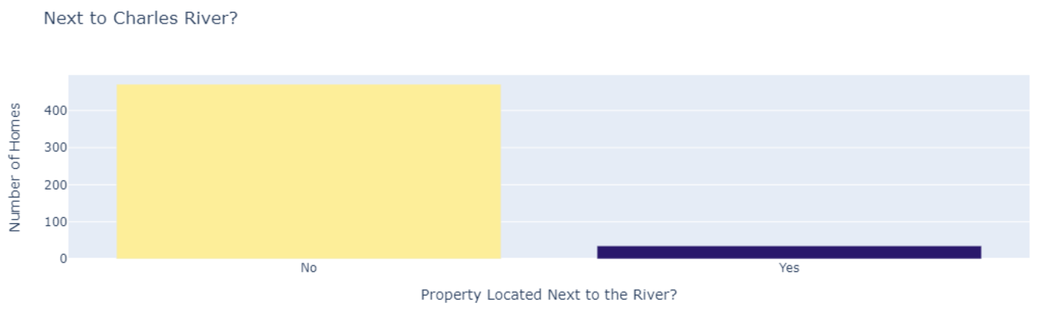
You can make your life easier by providing a list of values for the x-axis (e.g., `x=['No', 'Yes']`)

[97]:
```
river_access = data['CHAS'].value_counts()
```

[98]:
```
river_access
```

[98]:
```
0.00    471
1.00     35
Name: CHAS, dtype: int64
```

[99]:
```
bar = px.bar(x=['No', 'Yes'],
             y=river_access.values,
             color=river_access.values,
             color_continuous_scale=px.colors.sequential.haline,
             title='Next to Charles River?')

bar.update_layout(xaxis_title='Property Located Next to the River?',
                  yaxis_title='Number of Homes',
                  coloraxis_showscale=False)
bar.show()
```



## 4 Understand the Relationships in the Data

#### 4.0.1 Run a Pair Plot

**Challenge**

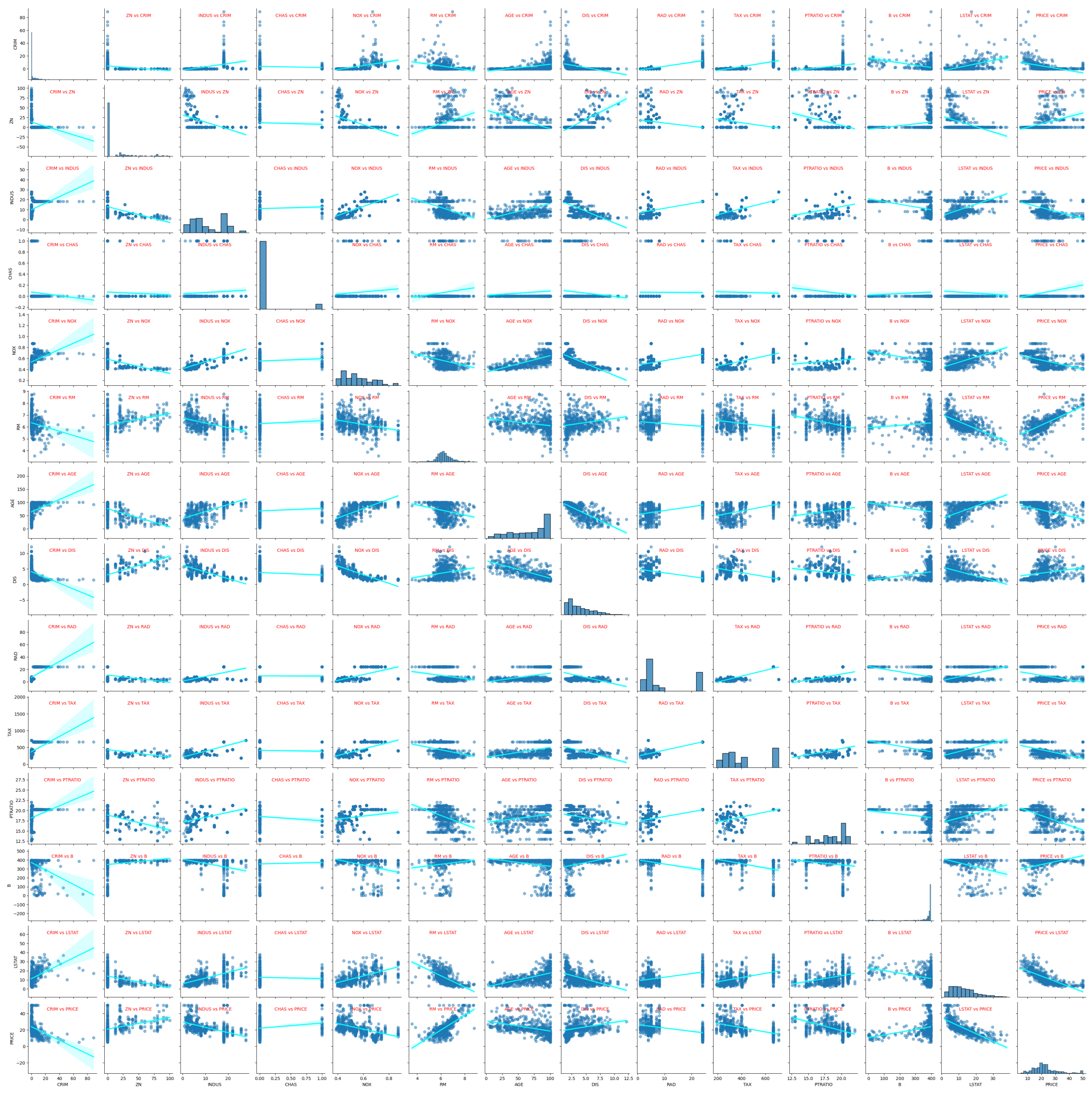
There might be some relationships in the data that we should know about. Before you run the code, make some predictions:

- What would you expect the relationship to be between pollution (NOX) and the distance to employment (DIS)?
- What kind of relationship do you expect between the number of rooms (RM) and the home value (PRICE)?
- What about the amount of poverty in an area (LSTAT) and home prices?

Run a Seaborn `.pairplot()` to visualise all the relationships at the same time. Note, this is a big task and can take 1-2 minutes! After it's finished check your intuition regarding the questions above on the `pairplot`.

[109]:
```python
# Create the pairplot
pairplot = sns.pairplot(data, kind='reg', plot_kws={'scatter_kws':{'alpha': 0.
 ↪5}, 'line_kws':{'color': 'cyan'}})

# Iterate over the axes in the pairplot to add annotations
for i in range(len(pairplot.axes)):  # Loop over rows
    for j in range(len(pairplot.axes[i])):  # Loop over columns
        if i != j:  # Skip the histogram plots on the diagonal
            # Add text annotation by using the column and row indices to get␣
 ↪the variable names
            pairplot.axes[i, j].text(0.5, 0.9, f'{pairplot.x_vars[j]} vs␣
 ↪{pairplot.y_vars[i]}',
                                     horizontalalignment='center',␣
 ↪verticalalignment='center',
                                     transform=pairplot.axes[i, j].transAxes,␣
 ↪color='red')
```

```
plt.show()
```



## Challenge

Use Seaborn's `.jointplot()` to look at some of the relationships in more detail. Create a jointplot for:

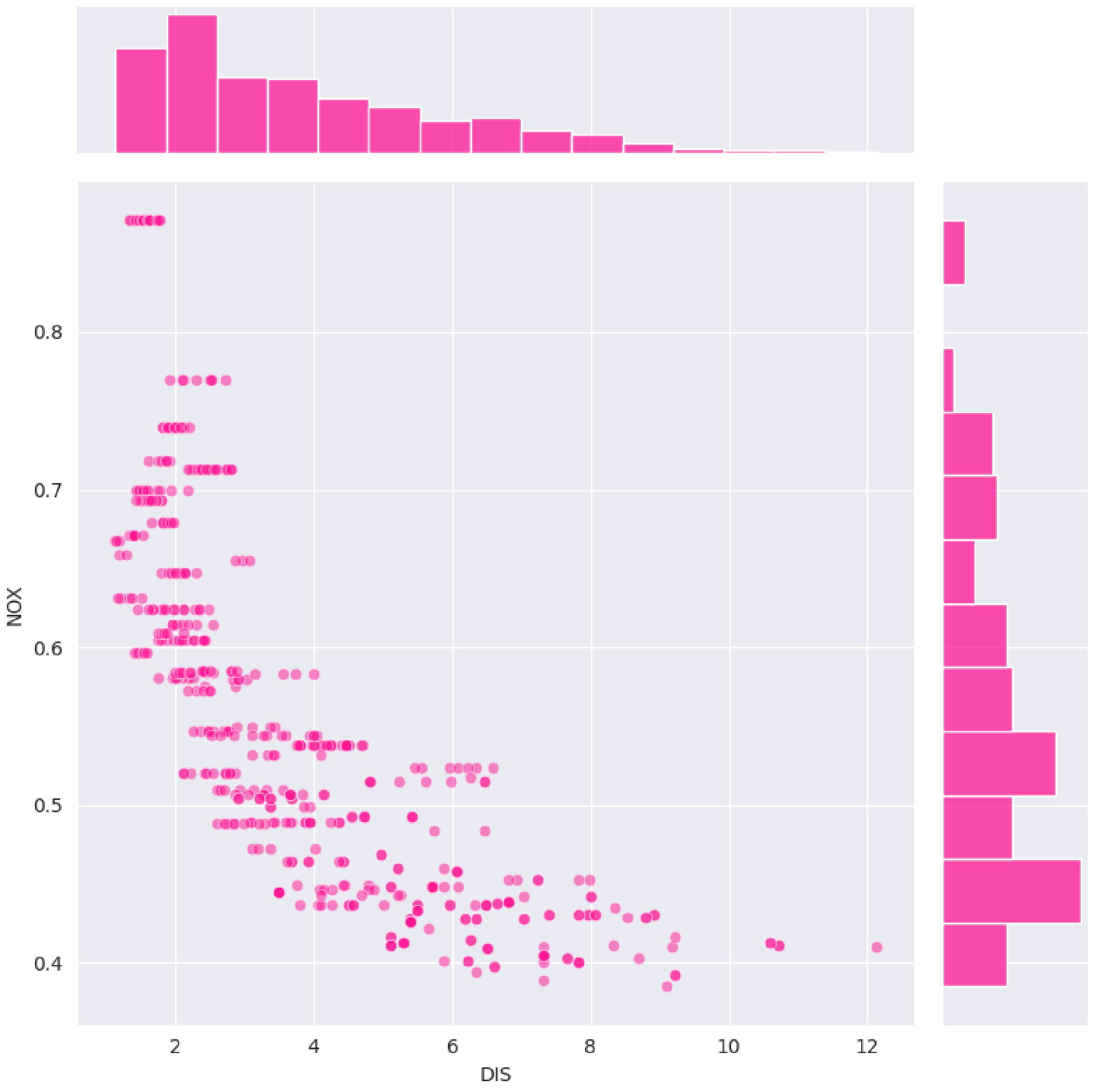
- DIS and NOX
- INDUS vs NOX
- LSTAT vs RM
- LSTAT vs PRICE
- RM vs PRICE

Try adding some opacity or `alpha` to the scatter plots using keyword arguments under `joint_kws`.

**Distance from Employment vs. Pollution  Challenge**:

Compare DIS (Distance from employment) with NOX (Nitric Oxide Pollution) using Seaborn's `.jointplot()`. Does pollution go up or down as the distance increases?

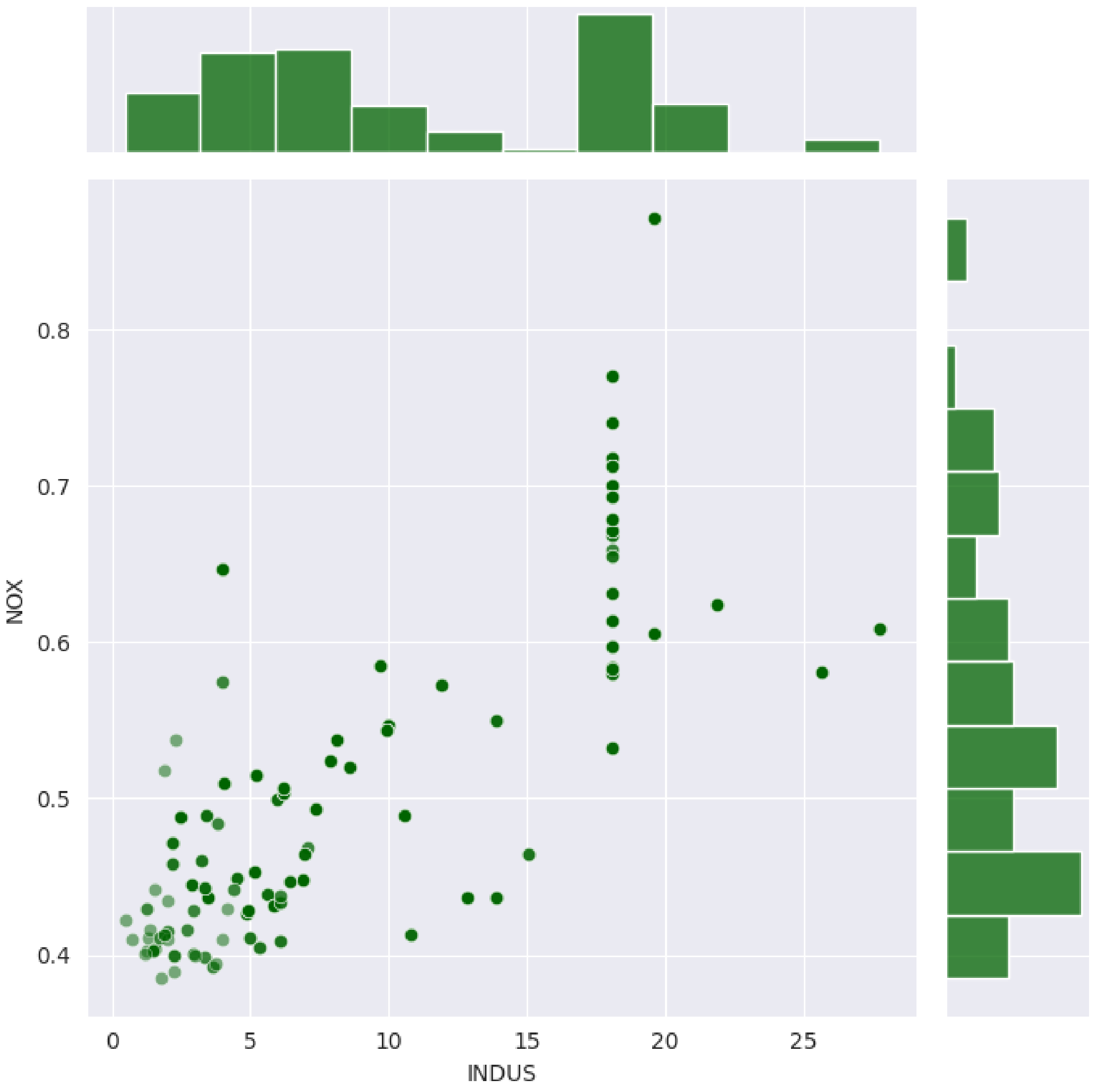
[110]:
```
with sns.axes_style('darkgrid'):
    sns.jointplot(x = data['DIS'],
                  y = data['NOX'],
                  height = 8,
                  kind = 'scatter',
                  color = 'deeppink',
                  joint_kws = {'alpha' : 0.5})
plt.show()
```



**Proportion of Non-Retail Industry versus Pollution Challenge**:

Compare INDUS (the proportion of non-retail industry i.e., factories) with NOX (Nitric Oxide Pollution) using Seaborn's `.jointplot()`. Does pollution go up or down as there is a higher proportion of industry?

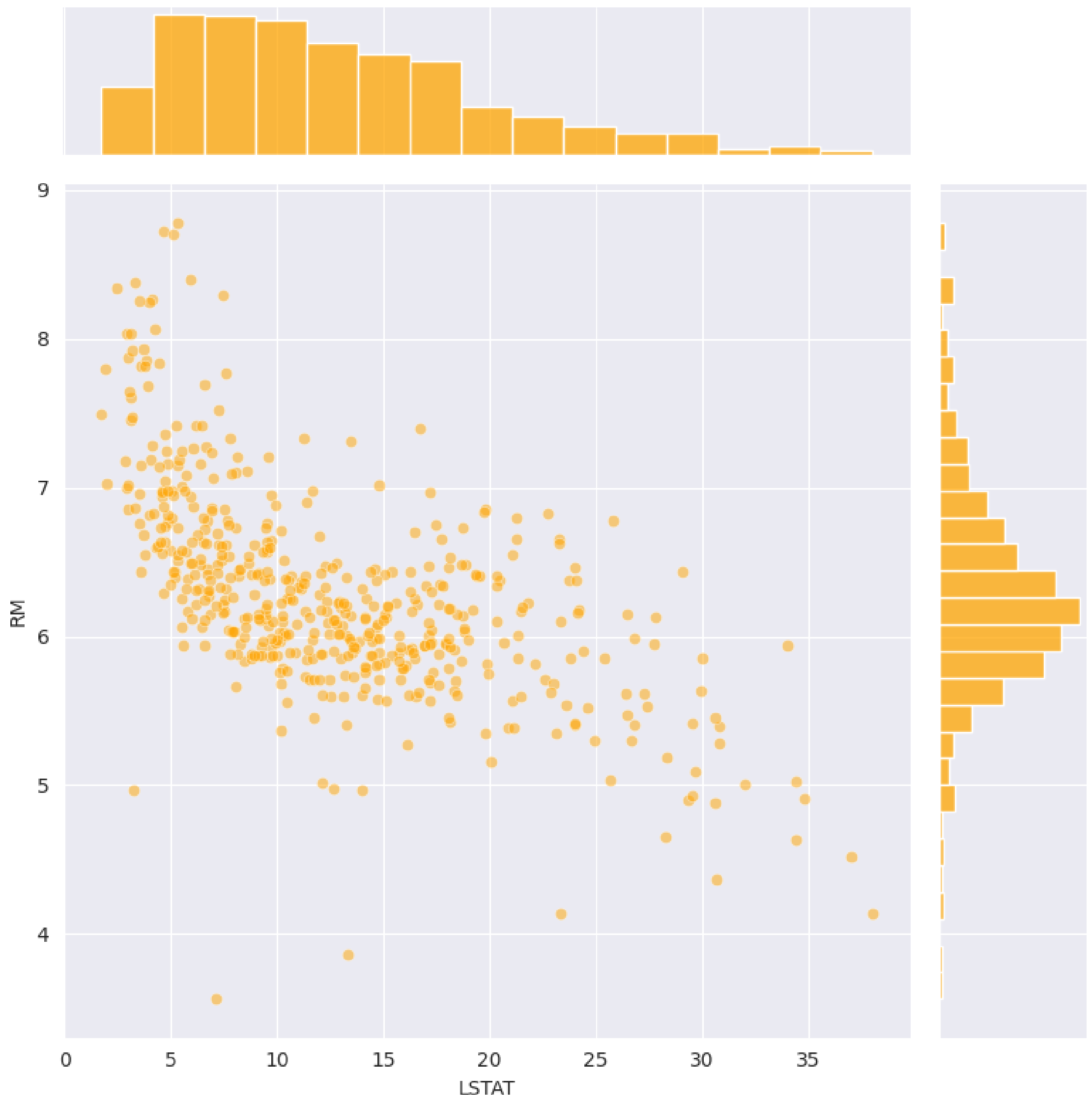
```
[111]: with sns.axes_style('darkgrid'):
           sns.jointplot(x = data['INDUS'],
                         y = data['NOX'],
                         height = 7,
                         kind = 'scatter',
                         color = 'darkgreen',
                         joint_kws = {'alpha' : 0.5})
       plt.show()
```



### % of Lower Income Population vs Average Number of Rooms  Challenge

Compare LSTAT (proportion of lower-income population) with RM (number of rooms) using Seaborn's `.jointplot()`. How does the number of rooms per dwelling vary with the poverty of area? Do homes have more or fewer rooms when LSTAT is low?

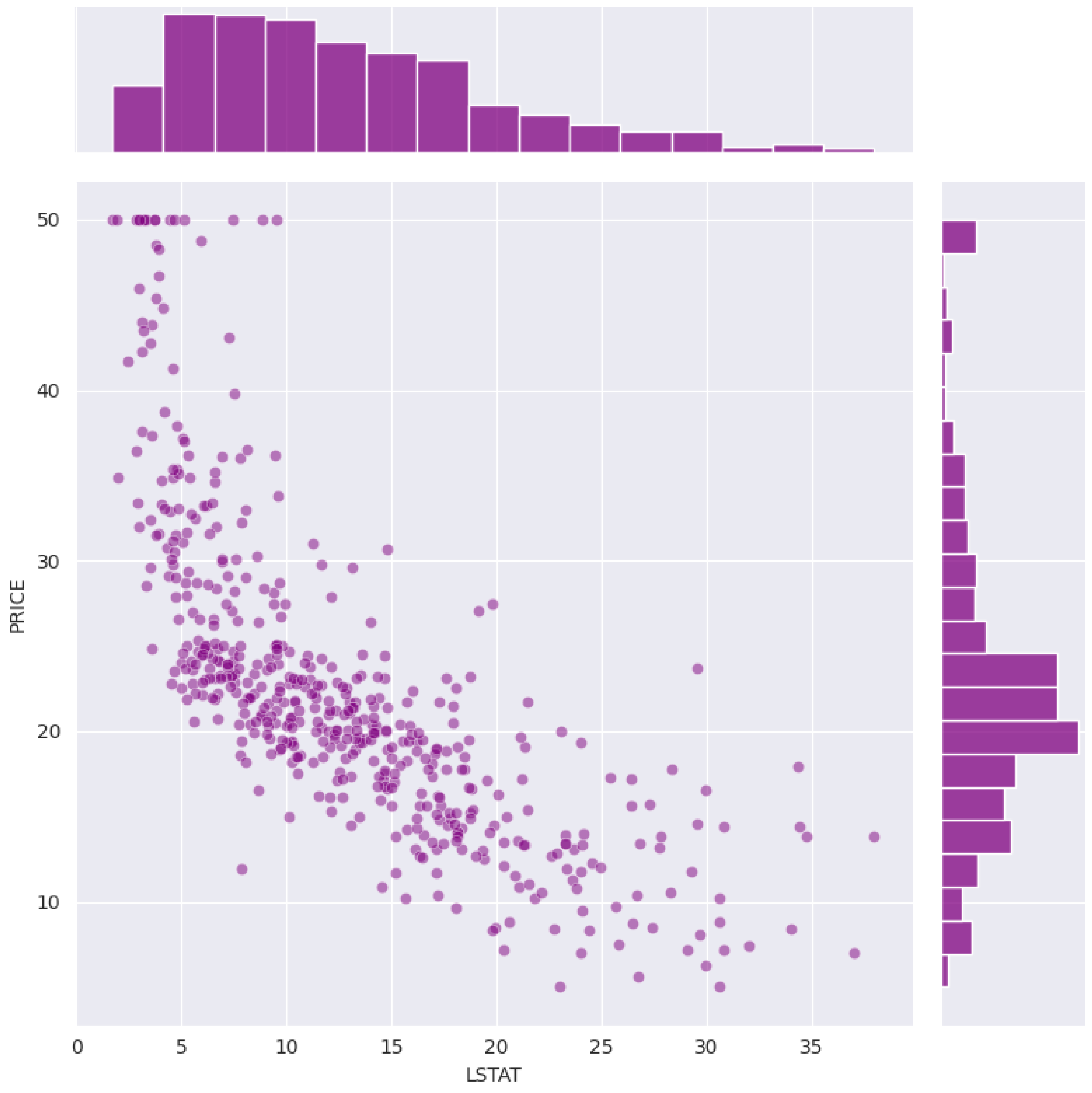
```
[112]: with sns.axes_style('darkgrid'):
           sns.jointplot(x = data['LSTAT'],
                         y = data['RM'],
                         height = 8,
                         kind = 'scatter',
                         color = 'orange',
                         joint_kws = {'alpha' : 0.5})
       plt.show()
```



### % of Lower Income Population versus Home Price   Challenge

Compare LSTAT with PRICE using Seaborn's `.jointplot()`. How does the proportion of the lower-income population in an area affect home prices?

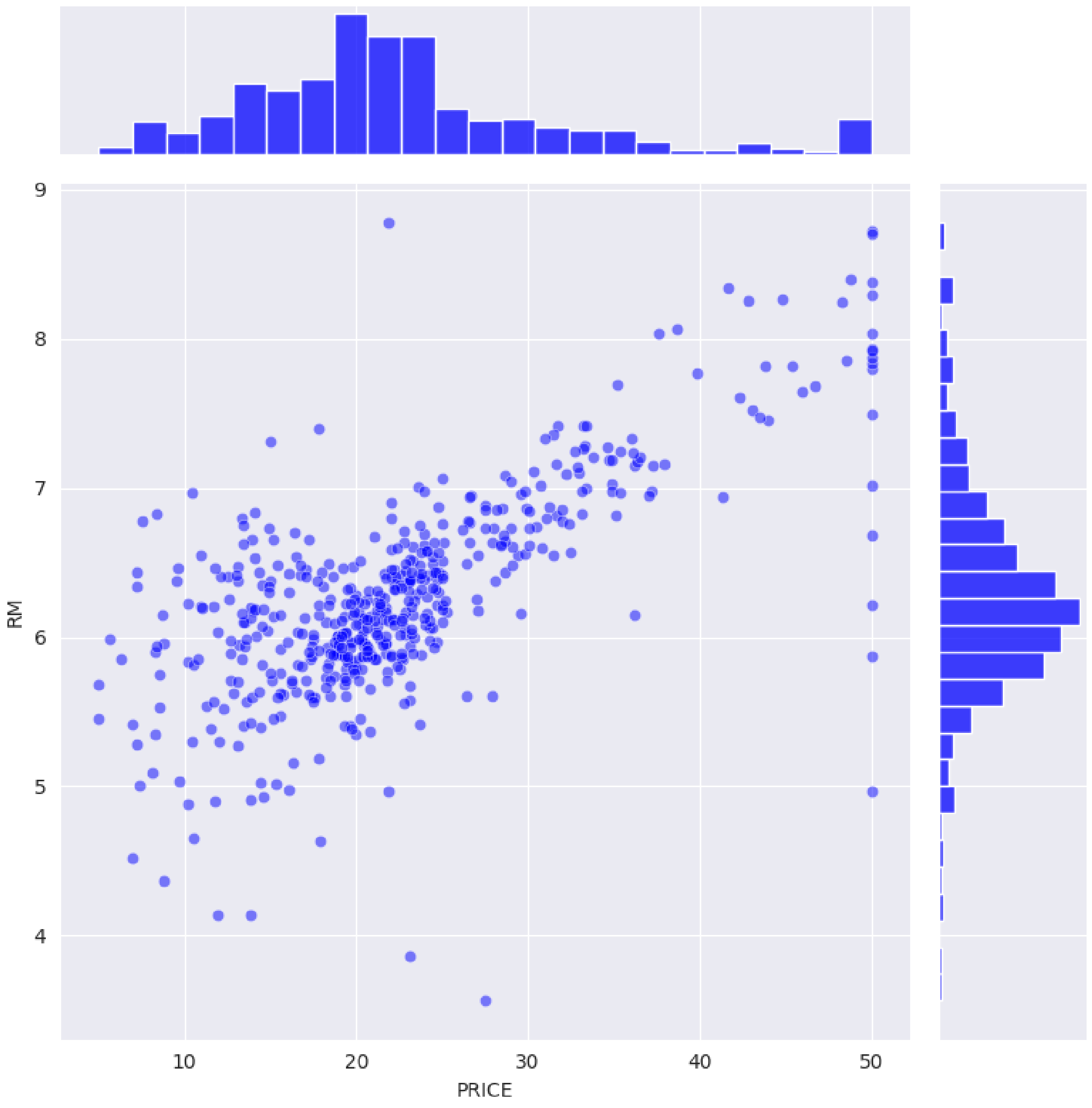
[113]:
```
with sns.axes_style('darkgrid'):
    sns.jointplot(x = data['LSTAT'],
                  y = data['PRICE'],
                  height = 8,
                  kind = 'scatter',
                  color = 'purple',
                  joint_kws = {'alpha' : 0.5})
plt.show()
```



**Number of Rooms versus Home Value   Challenge**

Compare RM (number of rooms) with PRICE using Seaborn's `.jointplot()`. You can probably guess how the number of rooms affects home prices.

```
[114]: with sns.axes_style('darkgrid'):
           sns.jointplot(x = data['PRICE'],
                         y = data['RM'],
                         height = 8,
                         kind = 'scatter',
                         color = 'blue',
                         joint_kws = {'alpha' : 0.5})
       plt.show()
```



## 5 Split Training & Test Dataset

We *can't* use all 506 entries in our dataset to train our model. The reason is that we want to evaluate our model on data that it hasn't seen yet (i.e., out-of-sample data). That way we can get a better idea of its performance in the real world.

**Challenge**

- Import the `train_test_split()` function from sklearn
- Create 4 subsets: X_train, X_test, y_train, y_test
- Split the training and testing data roughly 80/20.
- To get the same random split every time you run your notebook use `random_state=10`. This helps us get the same results every time and avoid confusion while we're learning.

Hint: Remember, your **target** is your home PRICE, and your **features** are all the other columns you'll use to predict the price.

[116]:
```
target = data['PRICE']
features = data.drop('PRICE', axis=1)

X_train, X_test, y_train, y_test = train_test_split(features,
                                                    target,
                                                    test_size=0.2,
                                                    random_state=10)
```

[117]:
```
X_train
```

[117]:
```
      CRIM    ZN  INDUS  CHAS  NOX   RM    AGE  DIS   RAD    TAX  PTRATIO  \
50    0.09 21.00   5.64  0.00 0.44 5.96  45.70 6.81  4.00 243.00    16.80
367  13.52  0.00  18.10  0.00 0.63 3.86 100.00 1.51 24.00 666.00    20.20
34    1.61  0.00   8.14  0.00 0.54 6.10  96.90 3.76  4.00 307.00    21.00
78    0.06  0.00  12.83  0.00 0.44 6.23  53.70 5.01  5.00 398.00    18.70
172   0.14  0.00   4.05  0.00 0.51 5.57  88.50 2.60  5.00 296.00    16.60
..     …     …      …     …    …    …      …    …     …      …
320   0.17  0.00   7.38  0.00 0.49 6.43  52.30 4.54  5.00 287.00    19.60
15    0.63  0.00   8.14  0.00 0.54 5.83  56.50 4.50  4.00 307.00    21.00
484   2.38  0.00  18.10  0.00 0.58 5.87  41.90 3.72 24.00 666.00    20.20
125   0.17  0.00  25.65  0.00 0.58 5.99  88.40 1.99  2.00 188.00    19.10
265   0.76 20.00   3.97  0.00 0.65 5.56  62.80 1.99  5.00 264.00    13.00

          B  LSTAT
50   395.56  13.45
367  131.42  13.33
34   248.31  20.34
78   386.40  12.34
172  396.90  14.69
```

```
..      …      …
320  396.90   7.20
15   395.62   8.47
484  370.73  13.34
125  385.02  14.81
265  392.40  10.45

[404 rows x 13 columns]
```

[118]:
```
features
```

[118]:
```
     CRIM    ZN  INDUS  CHAS  NOX   RM   AGE  DIS  RAD    TAX  PTRATIO      B  \
0    0.01 18.00   2.31  0.00 0.54 6.58 65.20 4.09 1.00 296.00    15.30 396.90
1    0.03  0.00   7.07  0.00 0.47 6.42 78.90 4.97 2.00 242.00    17.80 396.90
2    0.03  0.00   7.07  0.00 0.47 7.18 61.10 4.97 2.00 242.00    17.80 392.83
3    0.03  0.00   2.18  0.00 0.46 7.00 45.80 6.06 3.00 222.00    18.70 394.63
4    0.07  0.00   2.18  0.00 0.46 7.15 54.20 6.06 3.00 222.00    18.70 396.90
..    …     …      …     …   …    …     …   …    …      …        …      …
501  0.06  0.00  11.93  0.00 0.57 6.59 69.10 2.48 1.00 273.00    21.00 391.99
502  0.05  0.00  11.93  0.00 0.57 6.12 76.70 2.29 1.00 273.00    21.00 396.90
503  0.06  0.00  11.93  0.00 0.57 6.98 91.00 2.17 1.00 273.00    21.00 396.90
504  0.11  0.00  11.93  0.00 0.57 6.79 89.30 2.39 1.00 273.00    21.00 393.45
505  0.05  0.00  11.93  0.00 0.57 6.03 80.80 2.50 1.00 273.00    21.00 396.90

     LSTAT
0     4.98
1     9.14
2     4.03
3     2.94
4     5.33
..     …
501   9.67
502   9.08
503   5.64
504   6.48
505   7.88

[506 rows x 13 columns]
```

[119]:
```
len(X_test)
```

[119]:
```
102
```

[120]:
```
# % of training set
train_pct = len(X_train)/len(features) * 100
print(f'Training data is {train_pct:.3}% of the total data.')
```

```
# % of test data set
test_pct = len(X_test)/len(features) * 100
print(f'Test data makes up the remaining {test_pct:0.3}%.')
```

```
Training data is 79.8% of the total data.
Test data makes up the remaining 20.2%.
```

## 6 Multivariable Regression

In a previous lesson, we had a linear model with only a single feature (our movie budgets). This time we have a total of 13 features. Therefore, our Linear Regression model will have the following form:

$$P\hat{RIC}E = \theta_0 + \theta_1 RM + \theta_2 NOX + \theta_3 DIS + \theta_4 CHAS + ... + \theta_{13} LSTAT$$

#### 6.0.1 Run Your First Regression

**Challenge**

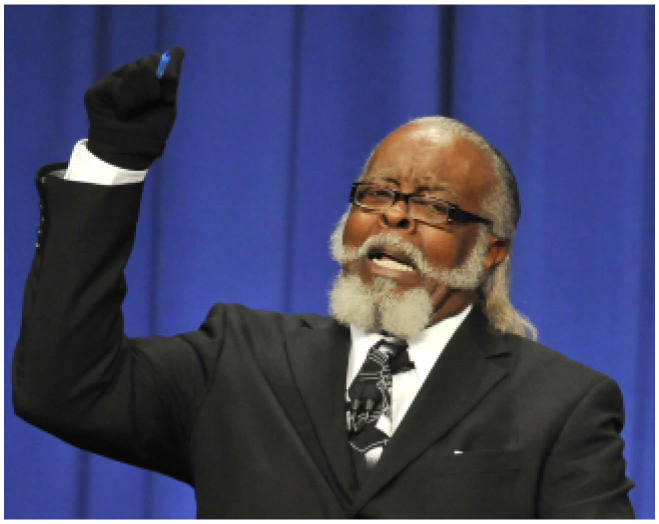
Use sklearn to run the regression on the training dataset. How high is the r-squared for the regression on the training data?

[121]:
```
regr = LinearRegression()
regr.fit(X_train, y_train)
rsquared = regr.score(X_train, y_train)

print(f'Training data r-squared: {rsquared:.2}')
```

```
Training data r-squared: 0.75
```

[123]:
```
print(f'Our features describe the target by {round(rsquared * 100)}% which is␣
↪too damn high')
```

```
Our features describe the target by 75% which is too damn high
```



### 6.0.2 Evaluate the Coefficients of the Model

Here we do a sense check on our regression coefficients. The first thing to look for is if the coefficients have the expected sign (positive or negative).

**Challenge** Print out the coefficients (the thetas in the equation above) for the features. Hint: You'll see a nice table if you stick the coefficients in a DataFrame.

- We already saw that RM on its own had a positive relation to PRICE based on the scatter plot. Is RM's coefficient also positive?
- What is the sign on the LSAT coefficient? Does it match your intuition and the scatter plot above?
- Check the other coefficients. Do they have the expected sign?
- Based on the coefficients, how much more expensive is a room with 6 rooms compared to a room with 5 rooms? According to the model, what is the premium you would have to pay for an extra room?

[124]:
```
regr_coef = pd.DataFrame(data=regr.coef_, index=X_train.columns,␣
 ↪columns=['Coefficient'])
regr_coef
```

[124]:
```
         Coefficient
CRIM           -0.13
ZN              0.06
INDUS          -0.01
CHAS            1.97
NOX           -16.27
RM              3.11
AGE             0.02
DIS            -1.48
RAD             0.30
TAX            -0.01
PTRATIO        -0.82
B               0.01
LSTAT          -0.58
```

[130]:
```
regr_coef.loc['RM'].values[0]
```

[130]:
```
3.1084562454033025
```

[132]:
```
premium = regr_coef.loc['RM'].values[0] * 1000
print(f'The price premium for having an extra room is ${premium:.5}')
```

```
The price premium for having an extra room is $3108.5
```

### 6.0.3 Analyse the Estimated Values & Regression Residuals

The next step is to evaluate our regression. How good our regression is depends not only on the r-squared. It also depends on the **residuals** - the difference between the model's predictions ($\hat{y}_i$) and the true values ($y_i$) inside `y_train`.

```
predicted_values = regr.predict(X_train)
residuals = (y_train - predicted_values)
```

**Challenge**: Create two scatter plots.

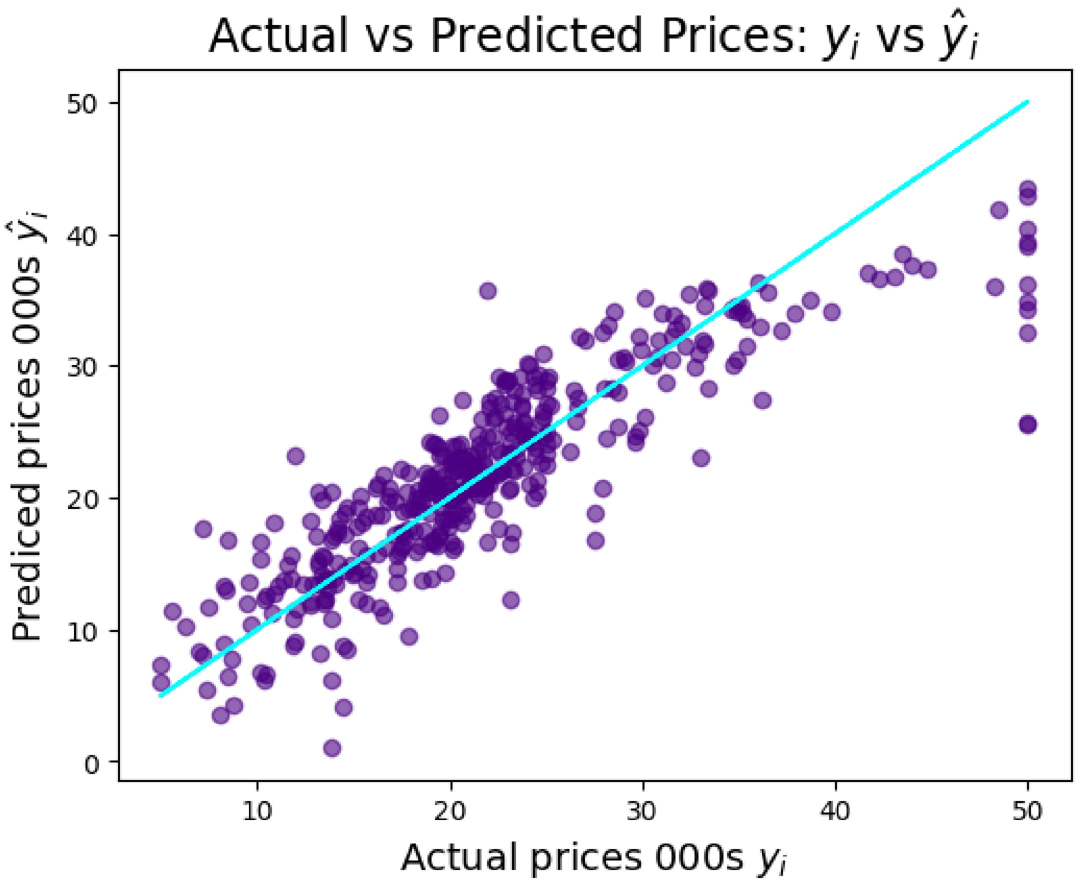
The first plot should be actual values (`y_train`) against the predicted value values:

The cyan line in the middle shows `y_train` against `y_train`. If the predictions had been 100% accurate then all the dots would be on this line. The further away the dots are from the line, the worse the prediction was. That makes the distance to the cyan line, you guessed it, our residuals

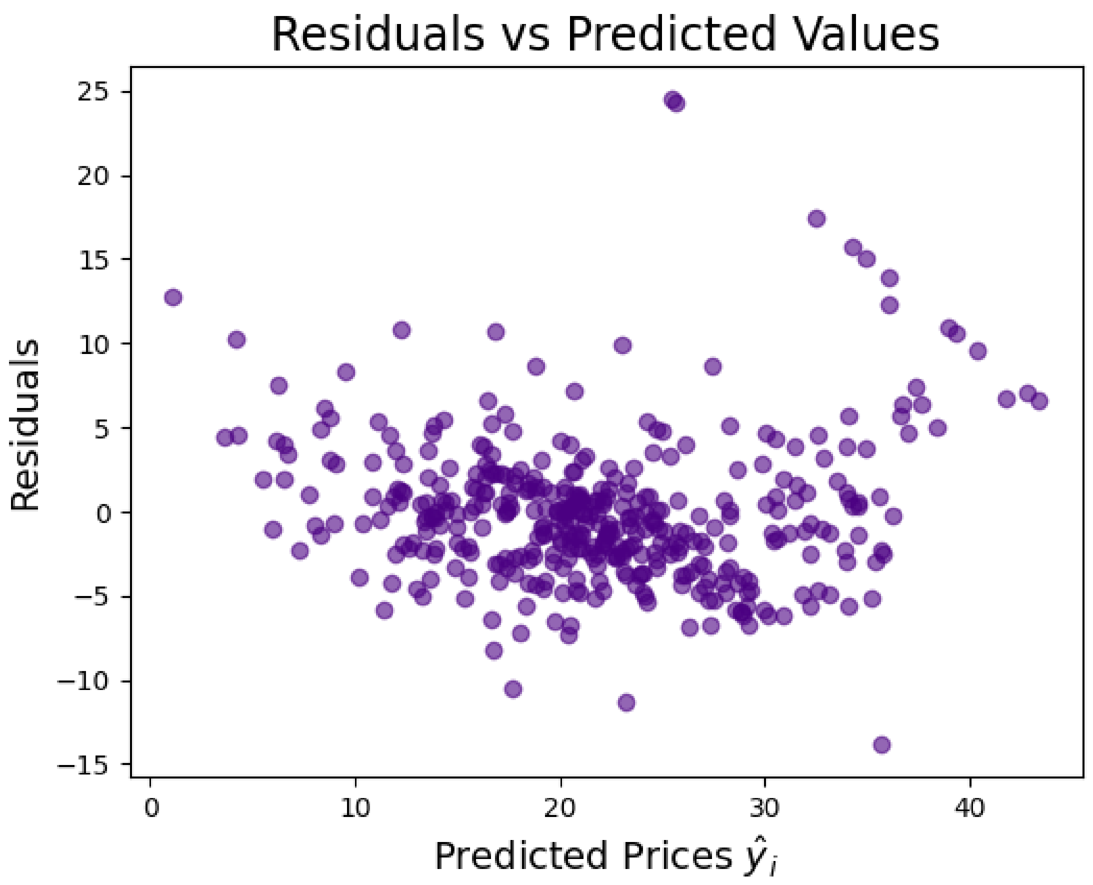
The second plot should be the residuals against the predicted prices. Here's what we're looking for:

[133]:
```
predicted_values = regr.predict(X_train)
residuals = (y_train - predicted_values)
```

[162]:
```
# Original Regression of Actual vs. Predicted Prices
plt.figure(dpi=100)
plt.scatter(x=y_train, y=predicted_values, c='indigo', alpha=0.6)
plt.plot(y_train, y_train, color='cyan')
plt.title(f'Actual vs Predicted Prices: $y _i$ vs $\hat y_i$', fontsize=17)
plt.xlabel('Actual prices 000s $y _i$', fontsize=14)
plt.ylabel('Prediced prices 000s $\hat y _i$', fontsize=14)
plt.show()
```



[163]:
```
# Residuals vs Predicted values
plt.figure(dpi=100)
plt.scatter(x=predicted_values, y=residuals, c='indigo', alpha=0.6)
plt.title('Residuals vs Predicted Values', fontsize=17)
plt.xlabel('Predicted Prices $\hat y _i$', fontsize=14)
plt.ylabel('Residuals', fontsize=14)
plt.show()
```



Why do we want to look at the residuals? We want to check that they look random. Why? The residuals represent the errors of our model. If there's a pattern in our errors, then our model has a systematic bias.

We can analyse the distribution of the residuals. In particular, we're interested in the **skew** and the **mean**.

In an ideal case, what we want is something close to a normal distribution. A normal distribution has a skewness of 0 and a mean of 0. A skew of 0 means that the distribution is symmetrical - the bell curve is not lopsided or biased to one side. Here's what a normal distribution looks like:

**Challenge**

- Calculate the mean and the skewness of the residuals.
- Again, use Seaborn's `.displot()` to create a histogram and superimpose the Kernel Density Estimate (KDE)
- Is the skewness different from zero? If so, by how much?
- Is the mean different from zero?

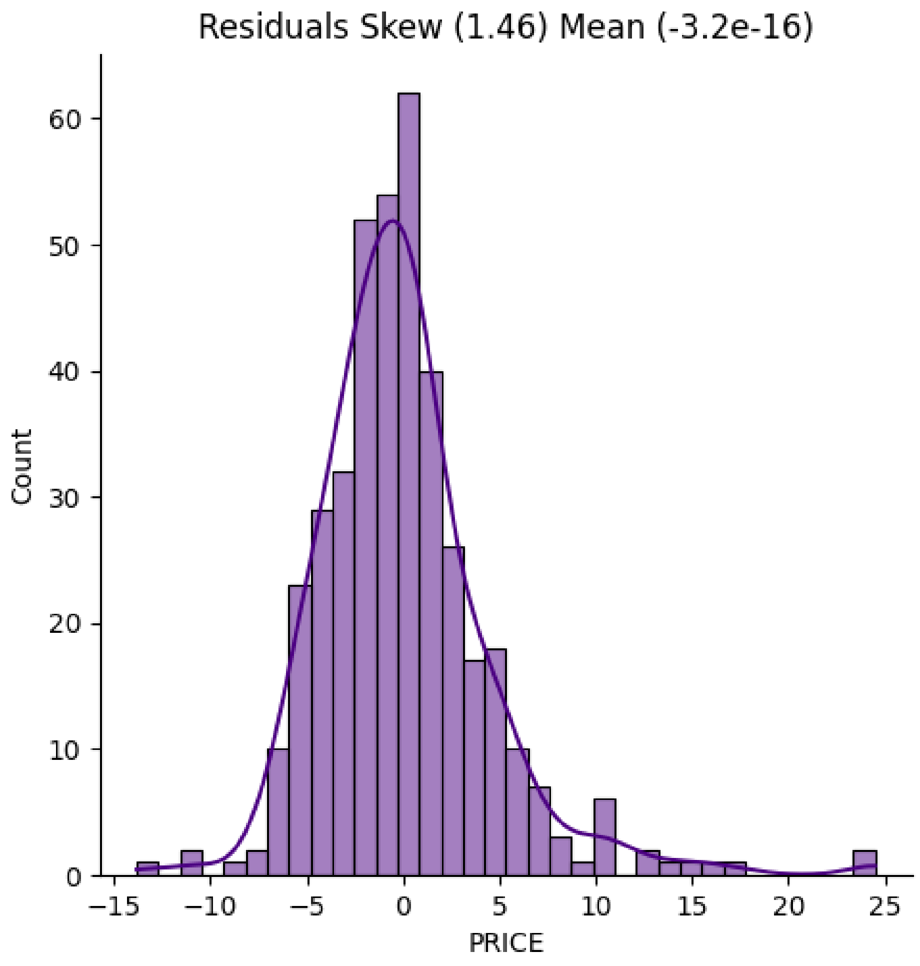
```
[164]: residuals.mean()
```

```
[164]: -3.165784466257872e-16
```

```
[165]: residuals.skew()
```

```
[165]: 1.4594428196679634
```

```
[166]: # Residual Distribution Chart
       resid_mean = round(residuals.mean(),2)
       resid_skew = round(residuals.skew(), 2)

       sns.displot(residuals, kde=True, color='indigo')
       plt.title(f'Residuals Skew ({resid_skew}) Mean ({residuals.mean():.2})')
       plt.show()
```



Needs improvement. Change our model completely or transform data to fit better with our linear model.

### 6.0.4 Data Transformations for a Better Fit

We have two options at this point:

1. Change our model entirely. Perhaps a linear model is not appropriate.
2. Transform our data to make it fit better with our linear model.

Let's try a data transformation approach.

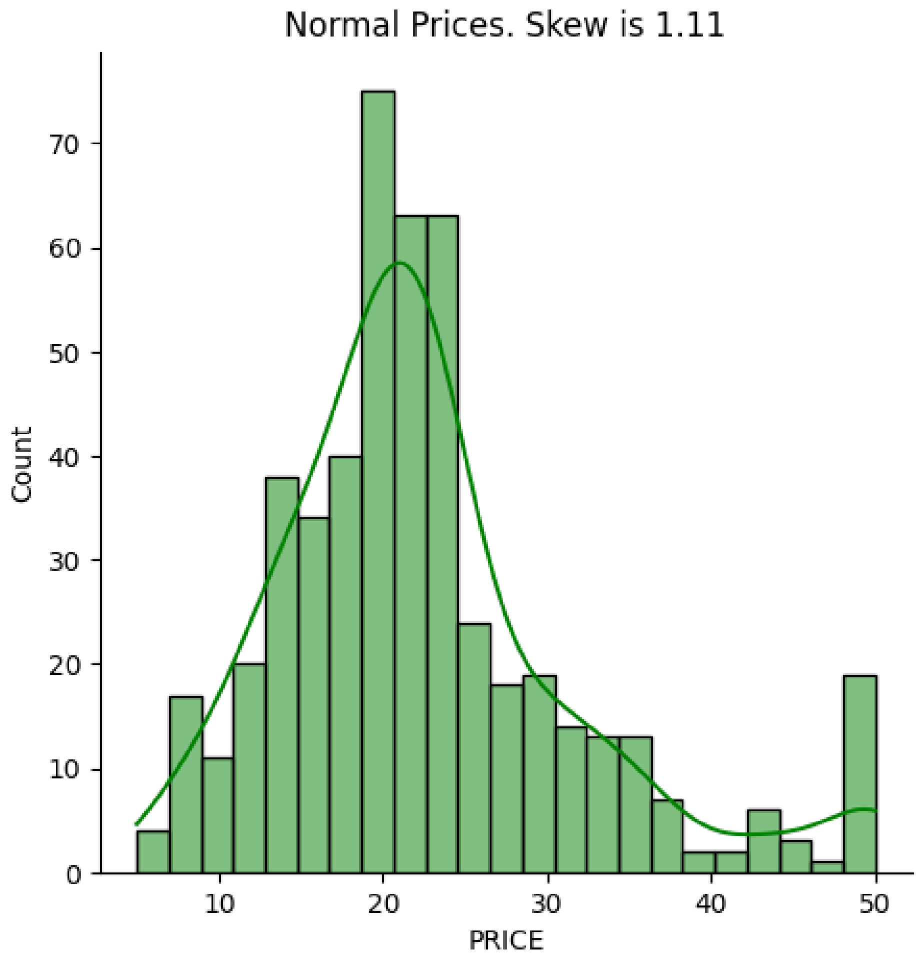
**Challenge**

Investigate if the target `data['PRICE']` could be a suitable candidate for a log transformation.

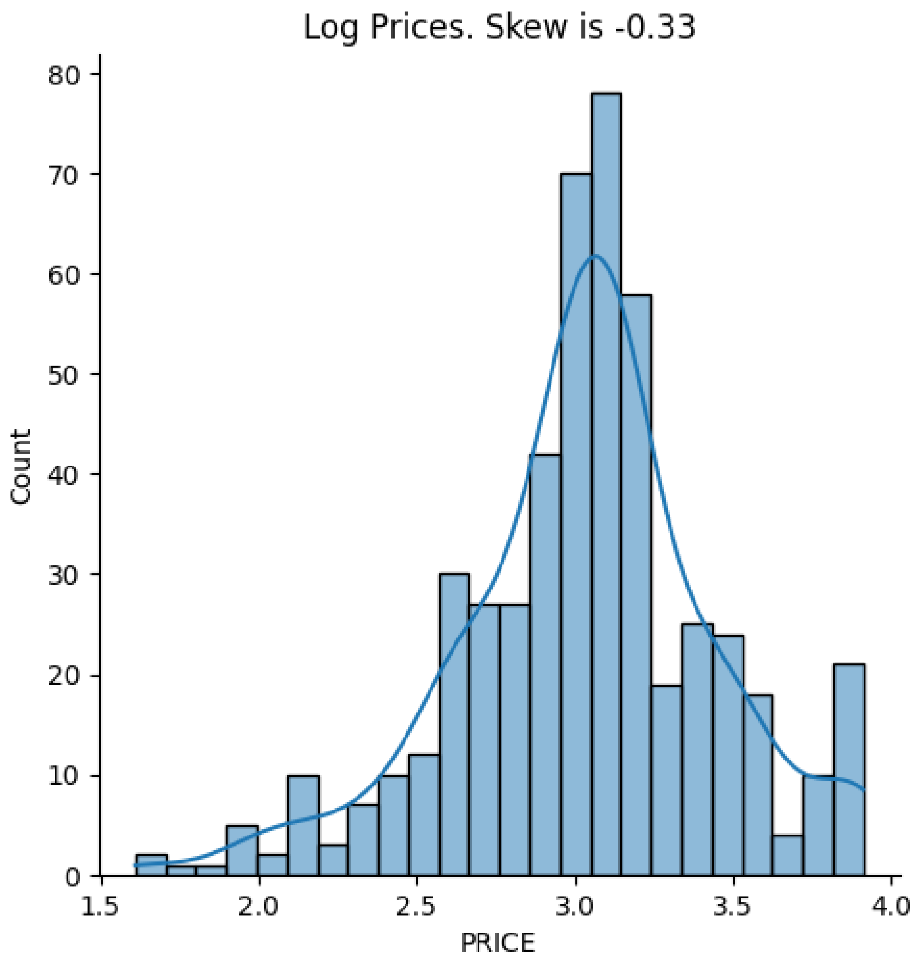
- Use Seaborn's `.displot()` to show a histogram and KDE of the price data.
- Calculate the skew of that distribution.
- Use NumPy's `log()` function to create a Series that has the log prices
- Plot the log prices using Seaborn's `.displot()` and calculate the skew.

- Which distribution has a skew that's closer to zero?

[167]:
```python
tgt_skew = data['PRICE'].skew()
sns.displot(data['PRICE'], kde=True , color='green')
plt.title(f'Normal Prices. Skew is {tgt_skew:.3}')
plt.show()
```



[168]:
```python
y_log = np.log(data['PRICE'])
sns.displot(y_log, kde=True)
plt.title(f'Log Prices. Skew is {y_log.skew():.3}')
plt.show()
```

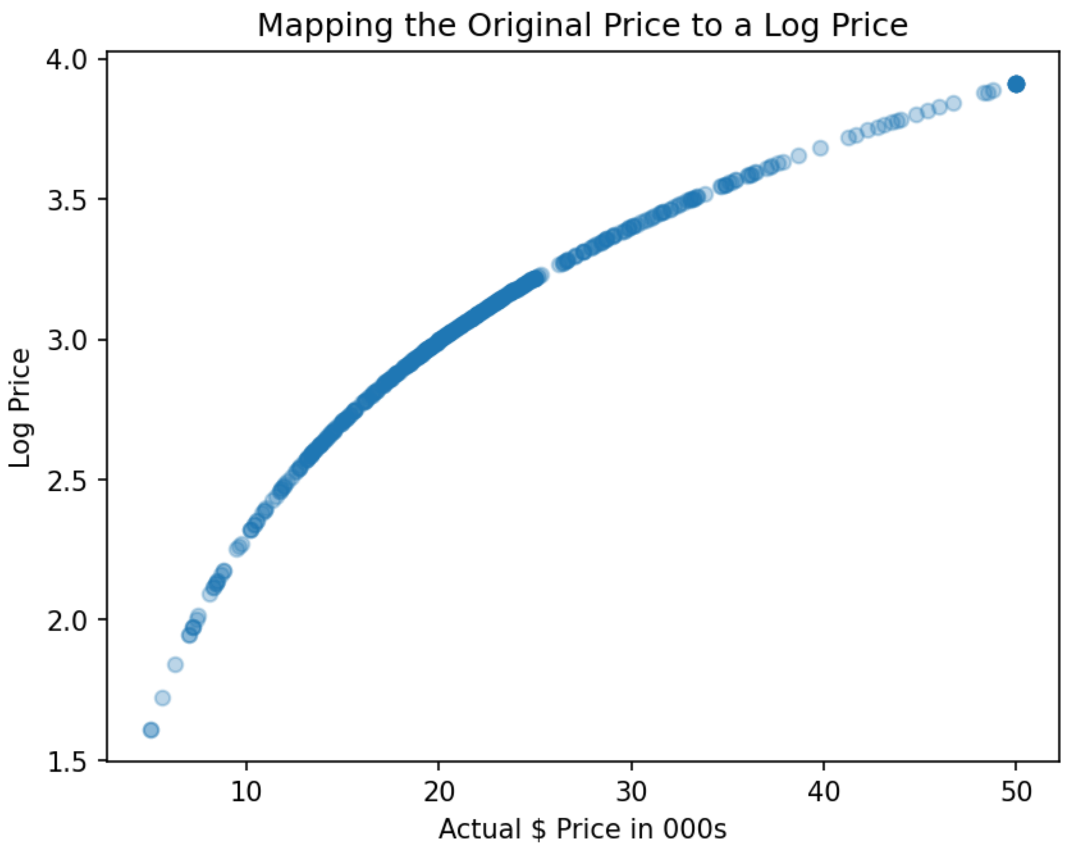


The log prices have a skew that's closer to zero. This makes them a good candidate for use in our linear model. Perhaps using log prices will improve our regression's r-squared and our model's residuals.

**How does the log transformation work?** Using a log transformation does not affect every price equally. Large prices are affected more than smaller prices in the dataset. Here's how the prices are "compressed" by the log transformation:

We can see this when we plot the actual prices against the (transformed) log prices.

[148]:
```
plt.figure(dpi=150)
plt.scatter(data.PRICE, y_log, alpha=0.3, s=30)

plt.title('Mapping the Original Price to a Log Price')
plt.ylabel('Log Price')
plt.xlabel('Actual $ Price in 000s')
plt.show()
```



### 6.1 Regression using Log Prices

Using log prices instead, our model has changed to:

$$\log(PR\hat{I}CE) = \theta_0 + \theta_1 RM + \theta_2 NOX + \theta_3 DIS + \theta_4 CHAS + ... + \theta_{13} LSTAT$$

**Challenge**:

- Use `train_test_split()` with the same random state as before to make the results comparable.
- Run a second regression, but this time use the transformed target data.
- What is the r-squared of the regression on the training data?
- Have we improved the fit of our model compared to before based on this measure?

[169]:
```
new_target = np.log(data['PRICE']) # Use log prices
features = data.drop('PRICE', axis=1)

X_train, X_test, log_y_train, log_y_test = train_test_split(features,
                                                    new_target,
                                                    test_size=0.2,
                                                    random_state=10)

log_regr = LinearRegression()
log_regr.fit(X_train, log_y_train)
log_rsquared = log_regr.score(X_train, log_y_train)

log_predictions = log_regr.predict(X_train)
log_residuals = (log_y_train - log_predictions)

print(f'Training data r-squared: {log_rsquared:.2}')
```

```
Training data r-squared: 0.79
```

## 6.2 Evaluating Coefficients with Log Prices

**Challenge**: Print out the coefficients of the new regression model.

- Do the coefficients still have the expected sign?
- Is being next to the river a positive based on the data?
- How does the quality of the schools affect property prices? What happens to prices as there are more students per teacher?

Hint: Use a DataFrame to make the output look pretty.

[170]:
```
df_coef = pd.DataFrame(data=log_regr.coef_, index=X_train.columns,␣
 ↪columns=['coef'])
df_coef
```

[170]:
```
         coef
CRIM    -0.01
ZN       0.00
INDUS    0.00
CHAS     0.08
NOX     -0.70
RM       0.07
AGE      0.00
DIS     -0.05
RAD      0.01
TAX     -0.00
PTRATIO -0.03
B        0.00
LSTAT   -0.03
```

## 6.3 Regression with Log Prices & Residual Plots

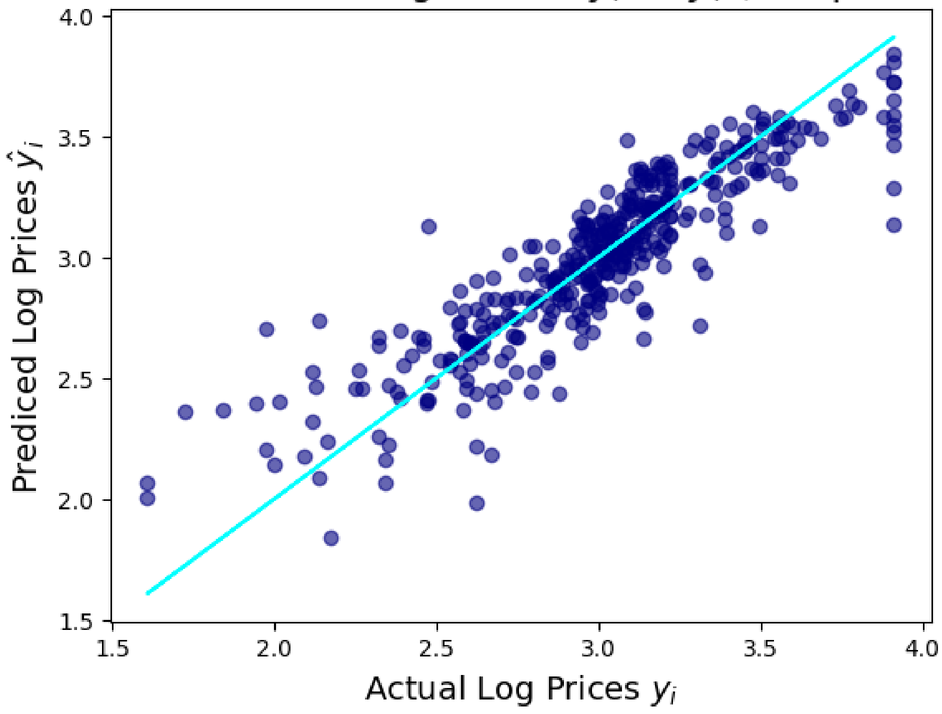
**Challenge**:

- Copy-paste the cell where you've created scatter plots of the actual versus the predicted home prices as well as the residuals versus the predicted values.

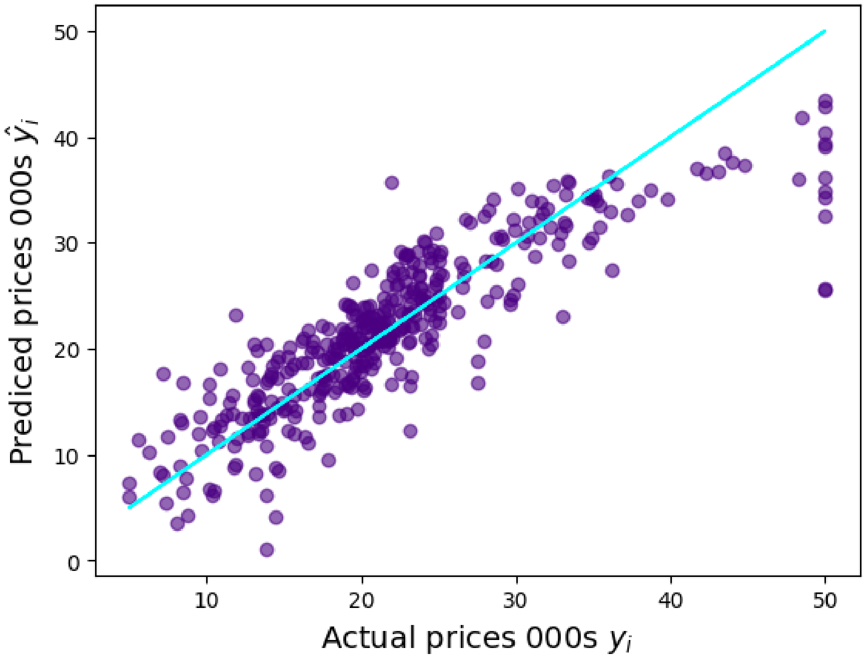
- Add 2 more plots to the cell so that you can compare the regression outcomes with the log prices side by side.
- Use `indigo` as the colour for the original regression and `navy` for the color using log prices.

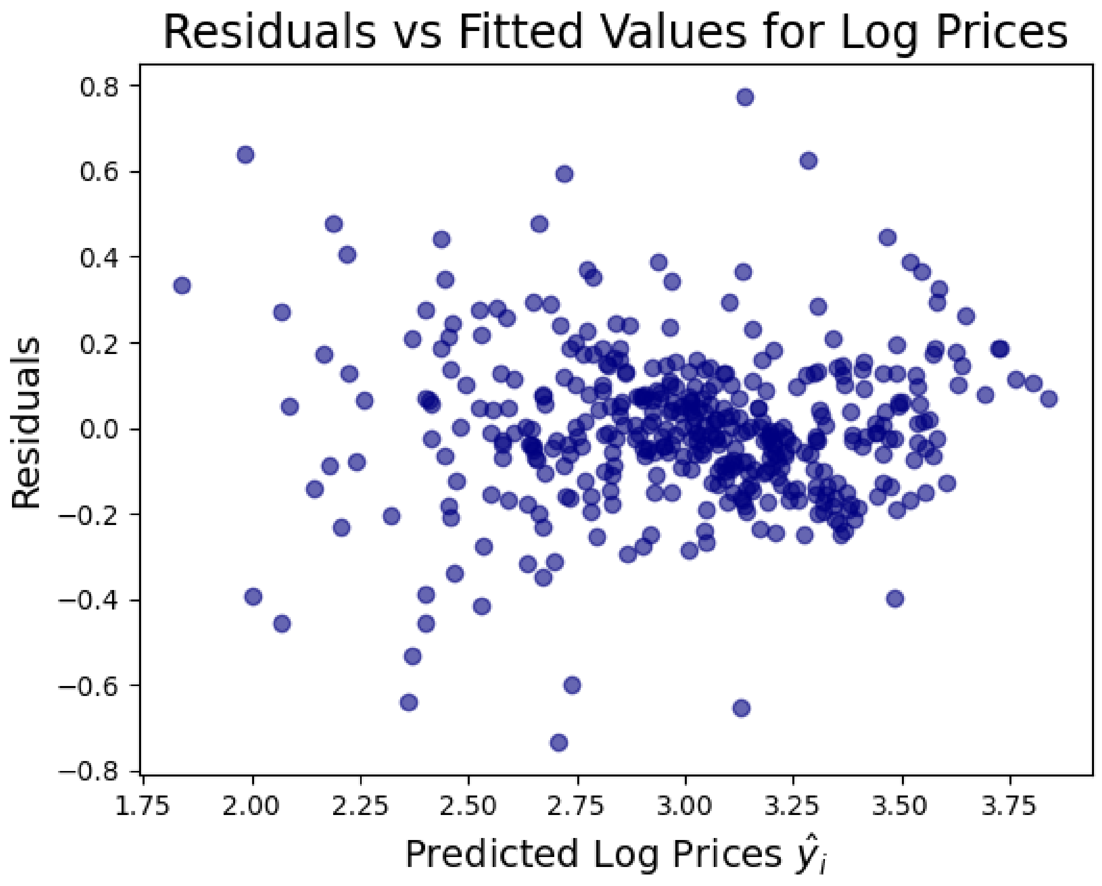
[171]:
```
# Graph of Actual vs. Predicted Log Prices
plt.scatter(x=log_y_train, y=log_predictions, c='navy', alpha=0.6)
plt.plot(log_y_train, log_y_train, color='cyan')
plt.title(f'Actual vs Predicted Log Prices: $y _i$ vs $\hat y_i$ (R-Squared␣
  ↪{log_rsquared:.2})', fontsize=17)
plt.xlabel('Actual Log Prices $y _i$', fontsize=14)
plt.ylabel('Prediced Log Prices $\hat y _i$', fontsize=14)
plt.show()

# Original Regression of Actual vs. Predicted Prices
plt.scatter(x=y_train, y=predicted_values, c='indigo', alpha=0.6)
plt.plot(y_train, y_train, color='cyan')
plt.title(f'Original Actual vs Predicted Prices: $y _i$ vs $\hat y_i$␣
  ↪(R-Squared {rsquared:.3})', fontsize=17)
plt.xlabel('Actual prices 000s $y _i$', fontsize=14)
plt.ylabel('Prediced prices 000s $\hat y _i$', fontsize=14)
plt.show()

# Residuals vs Predicted values (Log prices)
plt.scatter(x=log_predictions, y=log_residuals, c='navy', alpha=0.6)
plt.title('Residuals vs Fitted Values for Log Prices', fontsize=17)
plt.xlabel('Predicted Log Prices $\hat y _i$', fontsize=14)
plt.ylabel('Residuals', fontsize=14)
plt.show()

# Residuals vs Predicted values
plt.scatter(x=predicted_values, y=residuals, c='indigo', alpha=0.6)
plt.title('Original Residuals vs Fitted Values', fontsize=17)
plt.xlabel('Predicted Prices $\hat y _i$', fontsize=14)
plt.ylabel('Residuals', fontsize=14)
plt.show()
```

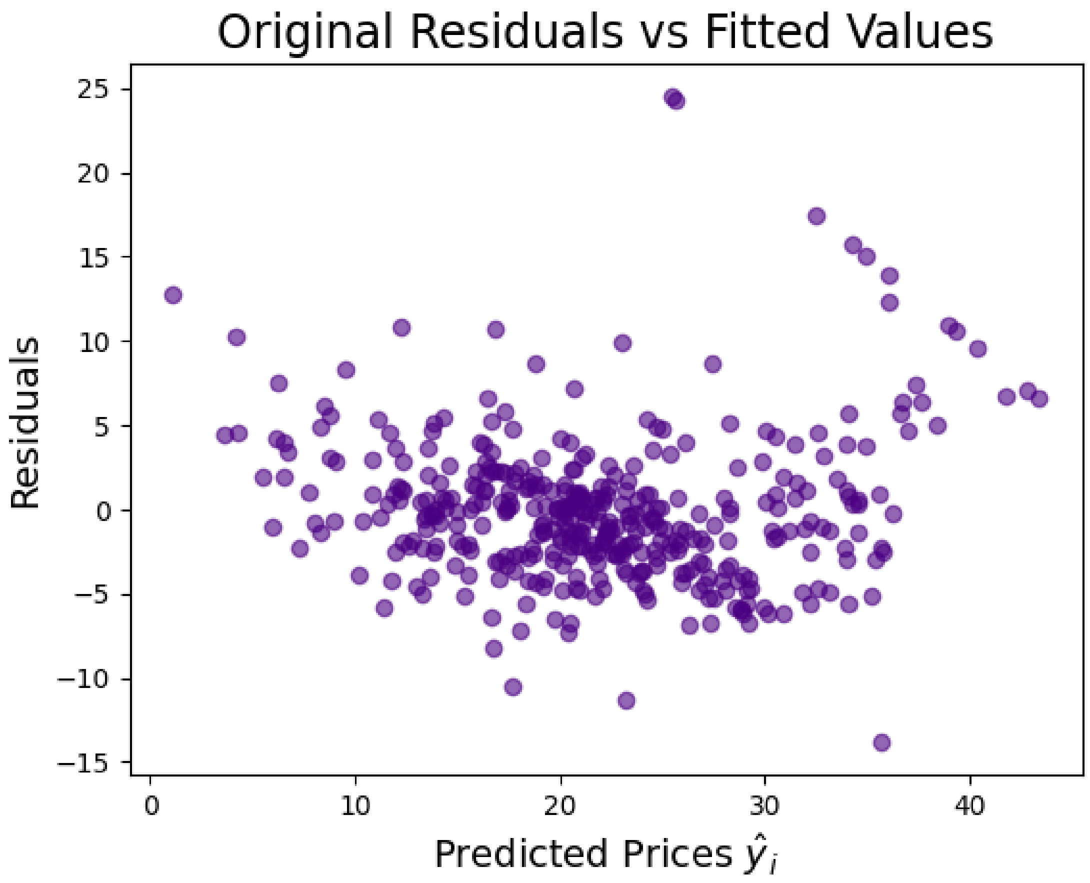
Actual vs Predicted Log Prices: $y_i$ vs $\hat{y}_i$ (R-Squared 0.79)



Original Actual vs Predicted Prices: $y_i$ vs $\hat{y}_i$ (R-Squared 0.75)



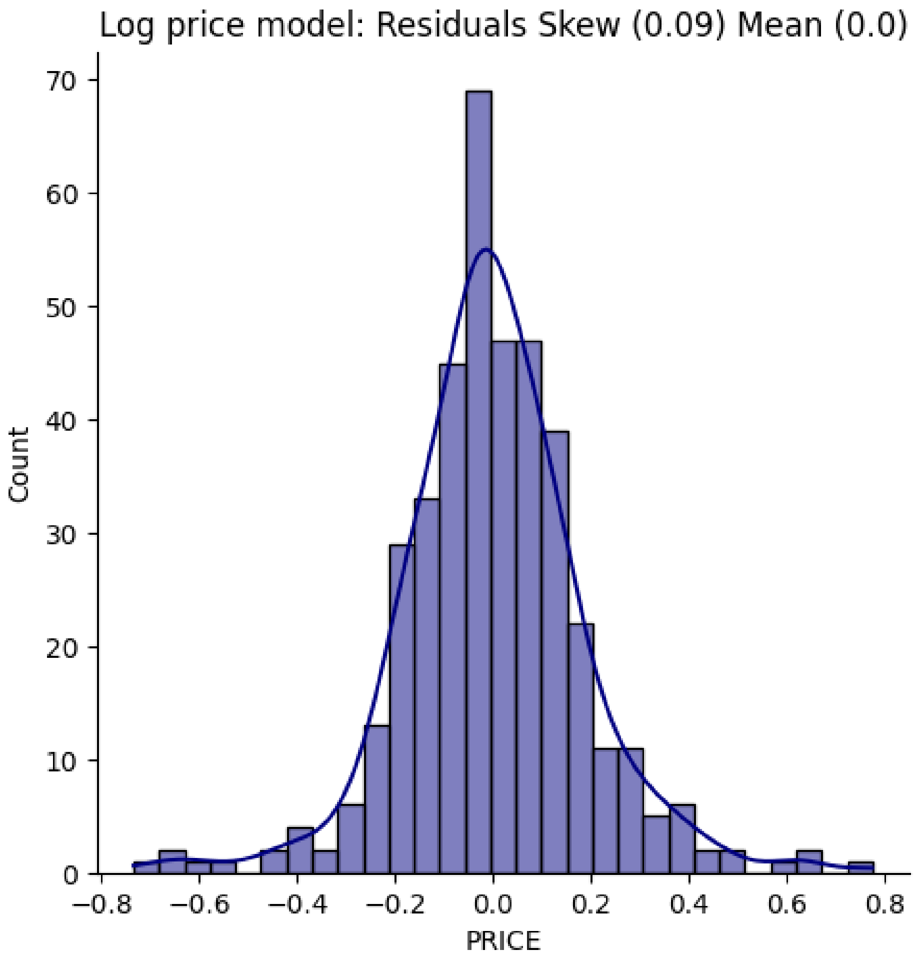

Residuals vs Fitted Values for Log Prices
Residuals
Predicted Log Prices $\hat{y}_i$
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
1.75
2.00
2.25
2.50
2.75
3.00
3.25
3.50
3.75

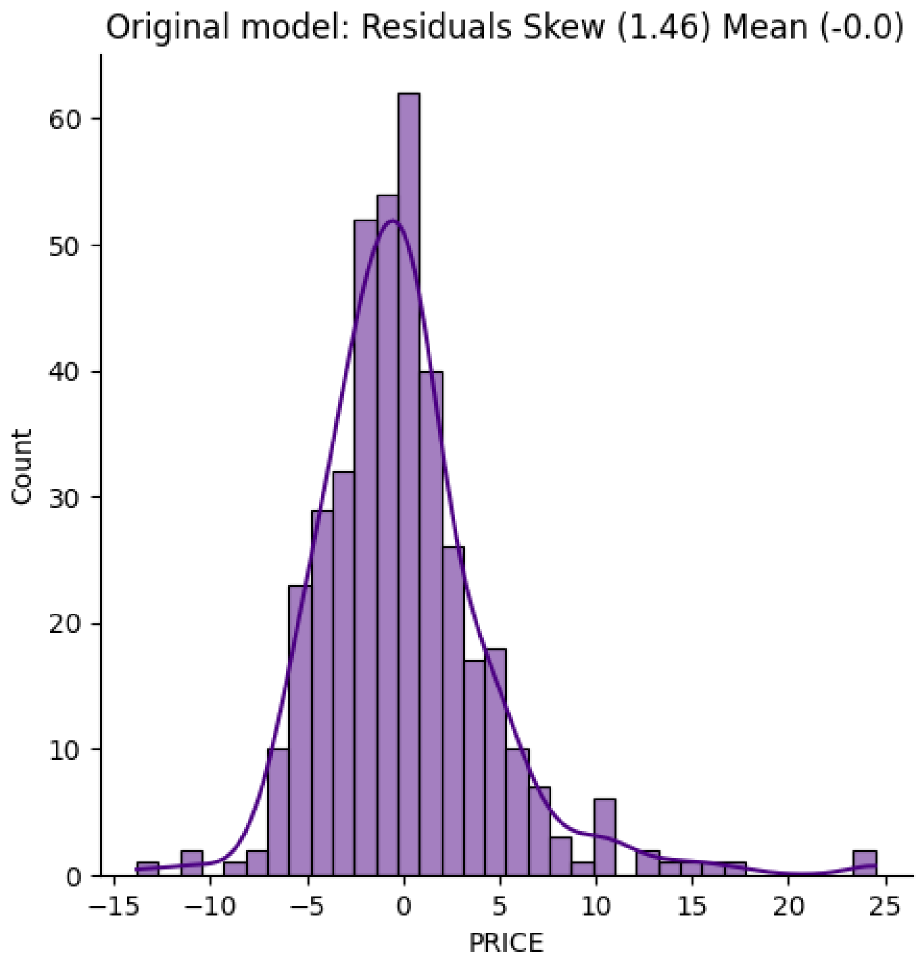


**Challenge**:

Calculate the mean and the skew for the residuals using log prices. Are the mean and skew closer to 0 for the regression using log prices?

```
[172]: # Distribution of Residuals (log prices) - checking for normality
log_resid_mean = round(log_residuals.mean(), 2)
log_resid_skew = round(log_residuals.skew(), 2)

sns.displot(log_residuals, kde=True, color='navy')
plt.title(f'Log price model: Residuals Skew ({log_resid_skew}) Mean␣
 ↪({log_resid_mean})')
plt.show()

sns.displot(residuals, kde=True, color='indigo')
plt.title(f'Original model: Residuals Skew ({resid_skew}) Mean ({resid_mean})')
plt.show()
```


Log price model: Residuals Skew (0.09) Mean (0.0)
Count
PRICE
0
10
20
30
40
50
60
70
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8



Our new regression residuals have a skew of 0.09 compared to a skew of 1.46. The mean is still around 0. From both a residuals perspective and an r-squared perspective we have improved our model with the data transformation.

## 7 Compare Out of Sample Performance

The *real* test is how our model performs on data that it has not “seen” yet. This is where our `X_test` comes in.

**Challenge**

Compare the r-squared of the two models on the test dataset. Which model does better? Is the r-squared higher or lower than for the training dataset? Why?

[173]:
```
print(f'Original Model Test Data r-squared: {regr.score(X_test, y_test):.2}')
print(f'Log Model Test Data r-squared: {log_regr.score(X_test, log_y_test):.2}')
```

```
Original Model Test Data r-squared: 0.67
```

```
Log Model Test Data r-squared: 0.74
```

## 8 Predict a Property's Value using the Regression Coefficients

Our preferred model now has an equation that looks like this:

$$\log(PR\hat{I}CE) = \theta_0 + \theta_1 RM + \theta_2 NOX + \theta_3 DIS + \theta_4 CHAS + ... + \theta_{13} LSTAT$$

The average property has the mean value for all its charactistics:

[177]:
```
# Starting Point: Average Values in the Dataset
features = data.drop(['PRICE'], axis=1)
average_vals = features.mean().values
property_stats = pd.DataFrame(data=average_vals.reshape(1, len(features.
  ↪columns)),
                              columns=features.columns)
property_stats
```

[177]:
```
   CRIM    ZN  INDUS  CHAS  NOX   RM   AGE  DIS  RAD    TAX  PTRATIO      B  \
0  3.61 11.36  11.14  0.07 0.55 6.28 68.57 3.80 9.55 408.24    18.46 356.67

   LSTAT
0  12.65
```

### Challenge

Predict how much the average property is worth using the stats above. What is the log price estimate and what is the dollar estimate? You'll have to reverse the log transformation with `.exp()` to find the dollar value.

[178]:
```
# Make prediction
log_estimate = log_regr.predict(property_stats)[0]
print(f'The log price estimate is ${log_estimate:.3}')

# Convert Log Prices to Actual Dollar Values
dollar_est = np.e**log_estimate * 1000
# or use
dollar_est = np.exp(log_estimate) * 1000
print(f'The property is estimated to be worth ${dollar_est:.6}')
```

```
The log price estimate is $3.03
The property is estimated to be worth $20703.2
```

### Challenge

Keeping the average values for CRIM, RAD, INDUS and others, value a property with the following characteristics:

[179]:
```python
# Define Property Characteristics
next_to_river = True
nr_rooms = 8
students_per_classroom = 20
distance_to_town = 5
pollution = data.NOX.quantile(q=0.75) # high
amount_of_poverty =  data.LSTAT.quantile(q=0.25) # low
```

[180]:
```python
# Solution
# Set Property Characteristics
property_stats['RM'] = nr_rooms
property_stats['PTRATIO'] = students_per_classroom
property_stats['DIS'] = distance_to_town

if next_to_river:
    property_stats['CHAS'] = 1
else:
    property_stats['CHAS'] = 0

property_stats['NOX'] = pollution
property_stats['LSTAT'] = amount_of_poverty
```

[181]:
```python
# Make prediction
log_estimate = log_regr.predict(property_stats)[0]
print(f'The log price estimate is ${log_estimate:.3}')

# Convert Log Prices to Acutal Dollar Values
dollar_est = np.e**log_estimate * 1000
print(f'The property is estimated to be worth ${dollar_est:.6}')
```

```
The log price estimate is $3.25
The property is estimated to be worth $25792.0
```