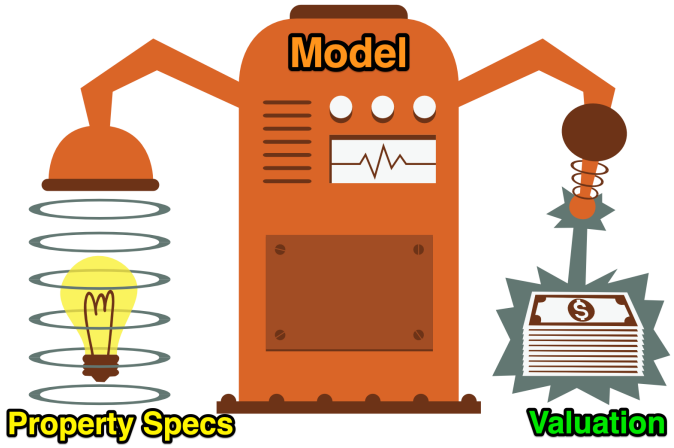
Setup and Context

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

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.

```
%pip install --upgrade plotly
```

```
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (5.15.0)
Collecting plotly
  Downloading plotly-5.18.0-py3-none-any.whl (15.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 15.6/15.6 MB 15.9 MB/s eta 0:00:00
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly) (8.2.3)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from plotly) (23.2)
Installing collected packages: plotly
  Attempting uninstall: plotly
    Found existing installation: plotly 5.15.0
    Uninstalling plotly-5.15.0:
      Successfully uninstalled plotly-5.15.0
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency conflicts.
lida 0.0.10 requires fastapi, which is not installed.
lida 0.0.10 requires kaleido, which is not installed.
lida 0.0.10 requires python-multipart, which is not installed.
lida 0.0.10 requires uvicorn, which is not installed.
Successfully installed plotly-5.18.0
```

Import Statements

```
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
```

Notebook Presentation

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

Load the Data

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

```
data = pd.read_csv('boston.csv', index_col=0)
```

Understand the Boston House Price Dataset

Characteristics:

```
:Number of Instances: 506

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

: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.

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?

```
data.shape # 506 data points
```

```
(506, 14)
```

```
data.columns # column names
```

```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT', 'PRICE'],
      dtype='object')
```

```
data.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
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	4.98	24.00
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	9.14	21.60
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	4.03	34.70
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	2.94	33.40
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	5.33	36.20

```
data.tail()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
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	9.67	22.40
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	9.08	20.60
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	5.64	23.90
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	6.48	22.00
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	7.88	11.90

```
data.count() # number of rows
```

```
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
```

Data Cleaning - Check for Missing Values and Duplicates

```
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
```

```
print(f'Any NaN values? {data.isna().values.any()}')
```

```
Any NaN values? False
```

```
print(f'Any duplicates? {data.duplicated().values.any()}')
```

```
Any duplicates? False
```

There are no null (i.e., NaN) values. Fantastic!

Descriptive Statistics

Challenge

- 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?

```
data.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
count	506.00	506.00	506.00	506.00	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	18.46	356.67	12.65	22.53
std	8.60	23.32	6.86	0.25	0.12	0.70	28.15	2.11	8.71	168.54	2.16	91.29	7.14	9.20
min	0.01	0.00	0.46	0.00	0.39	3.56	2.90	1.13	1.00	187.00	12.60	0.32	1.73	5.00
25%	0.08	0.00	5.19	0.00	0.45	5.89	45.02	2.10	4.00	279.00	17.40	375.38	6.95	17.02
50%	0.26	0.00	9.69	0.00	0.54	6.21	77.50	3.21	5.00	330.00	19.05	391.44	11.36	21.20
75%	3.68	12.50	18.10	0.00	0.62	6.62	94.07	5.19	24.00	666.00	20.20	396.23	16.96	25.00
max	88.98	100.00	27.74	1.00	0.87	8.78	100.00	12.13	24.00	711.00	22.00	396.90	37.97	50.00

`CHAS` shows whether the home is next to the Charles River or not. As such, it only has the value 0 or 1. This kind of feature is also known as a dummy variable.

The average price of a Boston home in the 1970s was 22.53 or $22,530. We've experienced a lot of inflation and house price appreciation since then!

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:

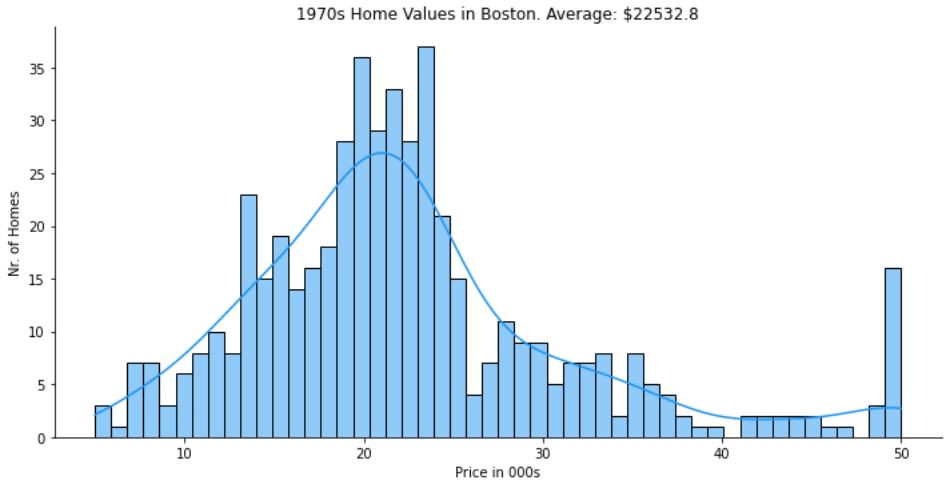
- PRICE: The home price in thousands.
- RM: the average number of rooms per owner unit.
- DIS: the weighted distance to the 5 Boston employment centres i.e., the estimated length of the commute.
- RAD: the index of accessibility to highways.

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

What do you notice in the distributions of the data?

House Prices 💰

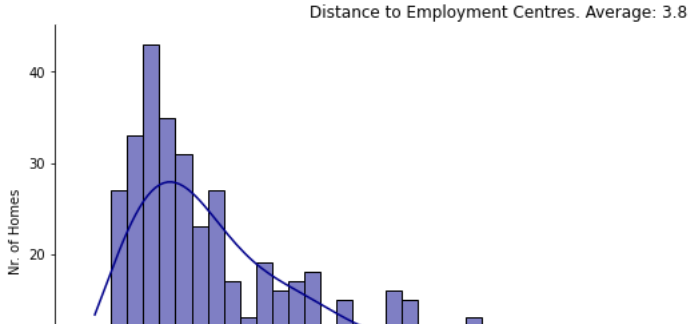
```
sns.displot(data['PRICE'],
            bins=50,
            aspect=2,
            kde=True,
            color='#2196f3')

plt.title(f'1970s Home Values in Boston. Average: ${(1000*data.PRICE.mean()):.6}')
plt.xlabel('Price in 000s')
plt.ylabel('Nr. of Homes')

plt.show()
```



Note there is a spike in the number homes at the very right tail at the $50,000 mark. 🤔

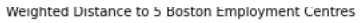
Distance to Employment - Length of Commute 🚙

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

plt.title(f'Distance to Employment Centres. Average: {(data.DIS.mean()):.2}')
plt.xlabel('Weighted Distance to 5 Boston Employment Centres')
plt.ylabel('Nr. of Homes')

plt.show()
```



Most homes are about 3.8 miles away from work. There are fewer and fewer homes the further out we go.

Number of Rooms




```
sns.displot(data.RM,
            aspect=2,
            kde=True,
            color='#00796b')

plt.title(f'Distribution of Rooms in Boston. Average: {data.RM.mean():.2}')
plt.xlabel('Average Number of Rooms')
plt.ylabel('Nr. of Homes')

plt.show()
```

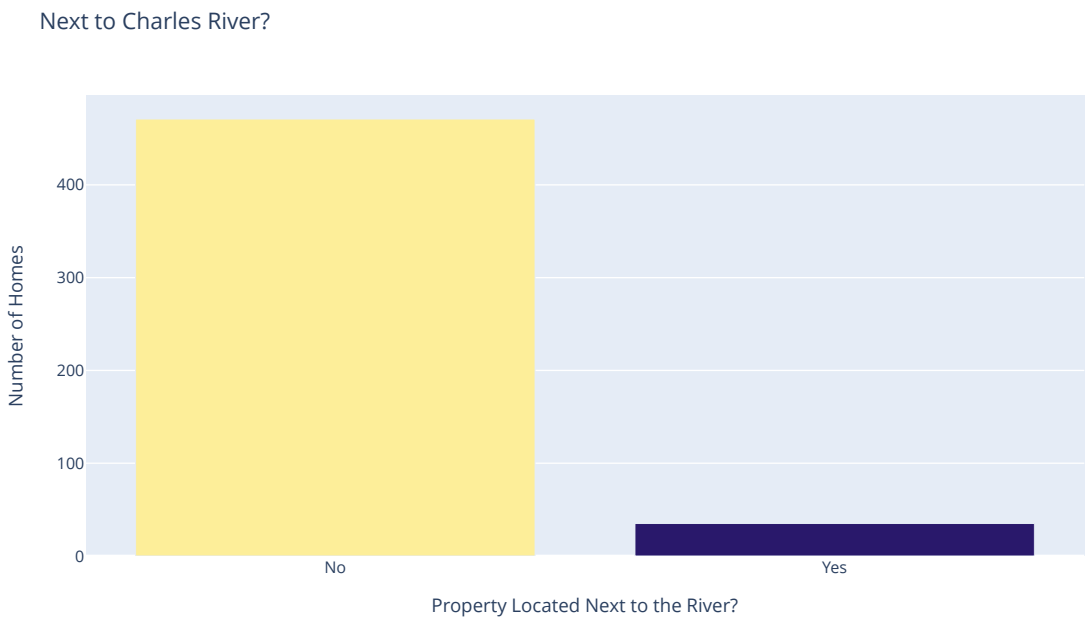
Access to Highways

```
plt.figure(figsize=(10, 5), dpi=200)

plt.hist(data['RAD'],
         bins=24,
         ec='black',
         color='#7b1fa2',
         rwidth=0.5)

plt.xlabel('Accessibility to Highways')
plt.ylabel('Nr. of Houses')
plt.show()
```

RAD is an index of accessibility to roads. Better access to a highway is represented by a higher number. There's a big gap in the values of the index.

Double-click (or enter) to edit

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

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()
```



We see that out of the total number of 506 homes, only 35 are located next to the Charles River.

Understand the Relationships in the Data

Run a Pair Plot

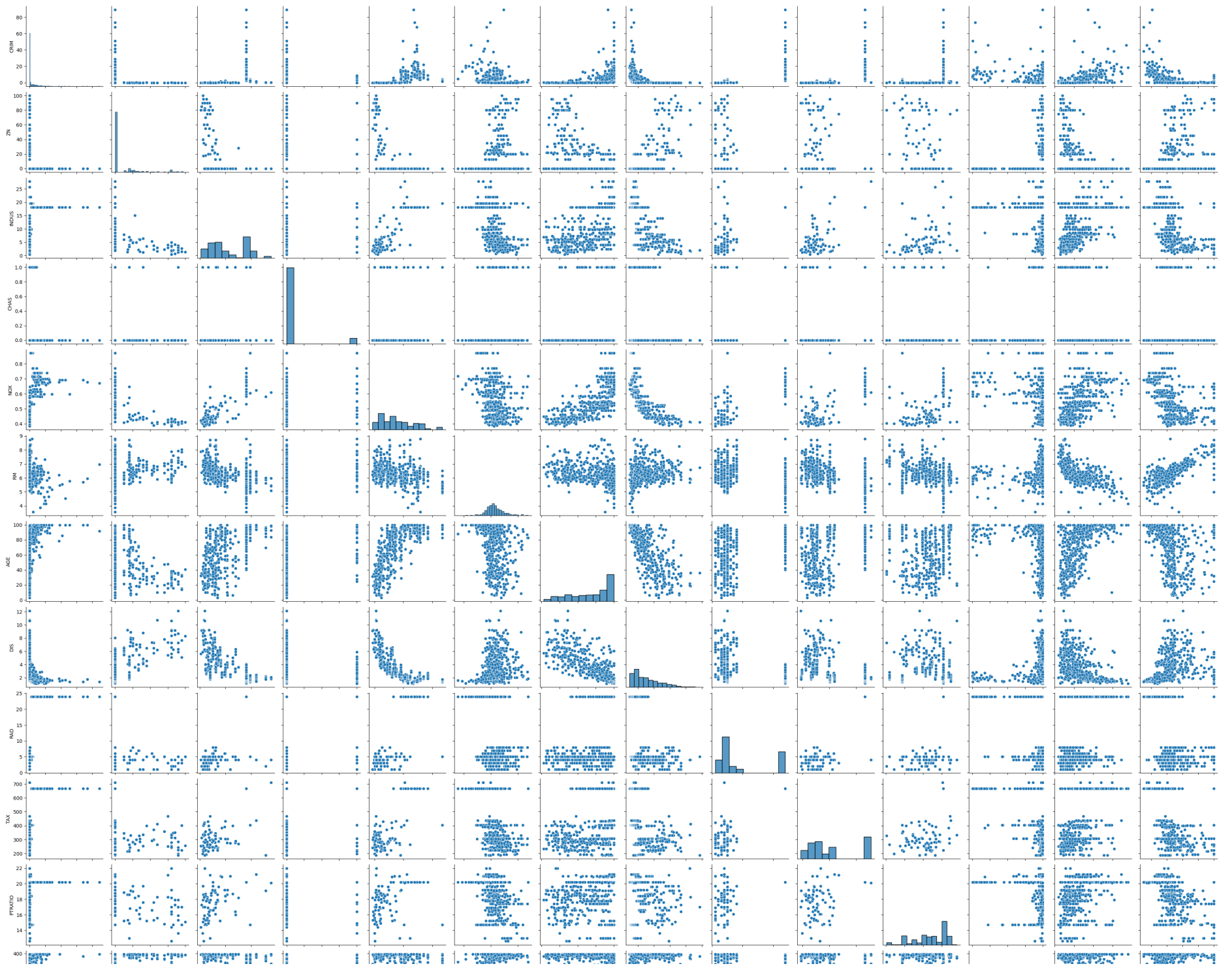
Challenge

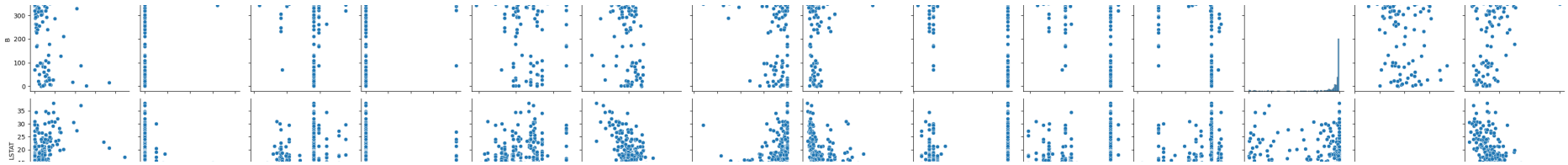
There might be some relationships in the data that we should know about. Before you run the code, make some predictions:

- 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`.

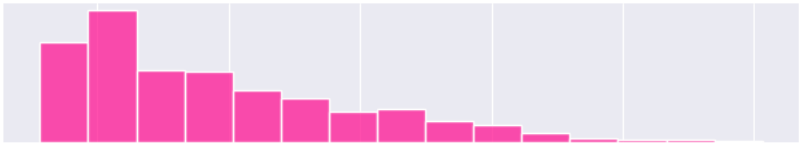
```
sns.pairplot(data)

# You can even include a regression line
# sns.pairplot(data, kind='reg', plot_kws={'line_kws':{'color': 'cyan'}})
plt.show()
```


CRIM
ZN
INDUS
CHAS
NOX
RM
AGE
DIS
RAD
TAX
PTRATIO



We see that we get back a grid. You might have to zoom in or squint a bit, but there are scatterplots between all the columns in our dataset. And down the diagonal in the middle, we get histograms for all our columns.

```
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()
```



We see that pollution goes down as we go further and further out of town. This makes intuitive sense. However, even at the same distance of 2 miles to employment centres, we can get very different levels of pollution. By the same token, DIS of 9 miles and 12 miles have very similar levels of pollution.

Proportion of Non-Retail Industry 🏭 🏭 🏭 versus Pollution

```
with sns.axes_style('darkgrid'):
  sns.jointplot(x=data.NOX,
                y=data.INDUS,
                # kind='hex',
                height=7,
                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?

```
with sns.axes_style('darkgrid'):
  sns.jointplot(x=data['LSTAT'],
                y=data['RM'],
                # kind='hex',
                height=7,
                color='orange',
                joint_kws={'alpha':0.5})
plt.show()
```

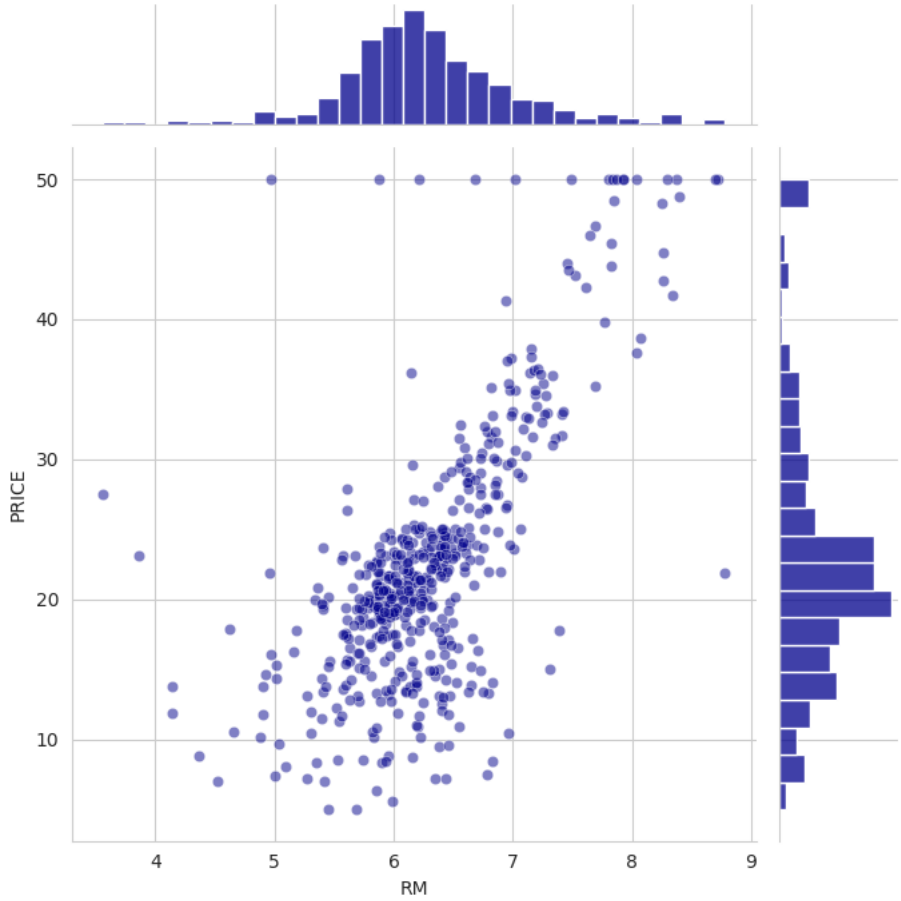
In the top left corner we see that all the homes with 8 or more rooms, LSTAT is well below 10%.

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. 😊

```
with sns.axes_style('whitegrid'):
  sns.jointplot(x=data.RM,
                y=data.PRICE,
                height=7,
                color='darkblue',
                joint_kws={'alpha':0.5})
plt.show()
```



Again, we see those homes at the $50,000 mark all lined up at the top of the chart. Perhaps there was some sort of cap or maximum value imposed during data collection.

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.

```
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)
```

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

# % of test data set
test_pct = 100*X_test.shape[0]/features.shape[0]
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%.
```

Run Your First Regression

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

```
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
```

0.75 is a very high r-squared!

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).

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

```
regr_coef
```

	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

```
# Premium for having an extra room
premium = regr_coef.loc['RM'].values[0] * 1000  # i.e., ~3.11 * 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
```

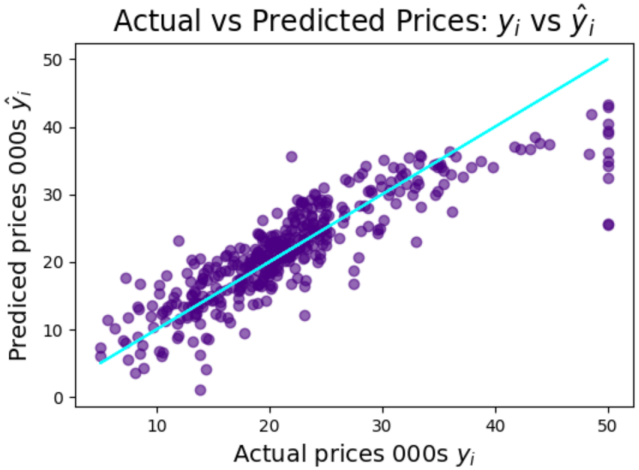
Analyse the Estimated Values & Regression Residuals

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)
```

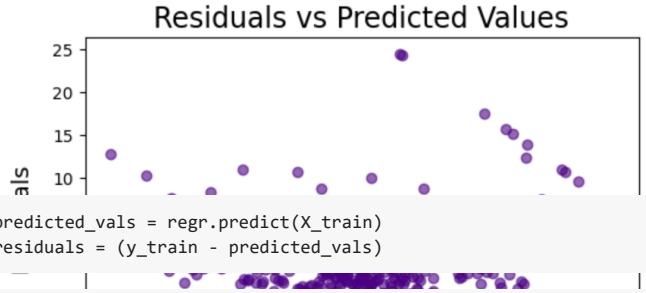
Challenge: Create two scatter plots.

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



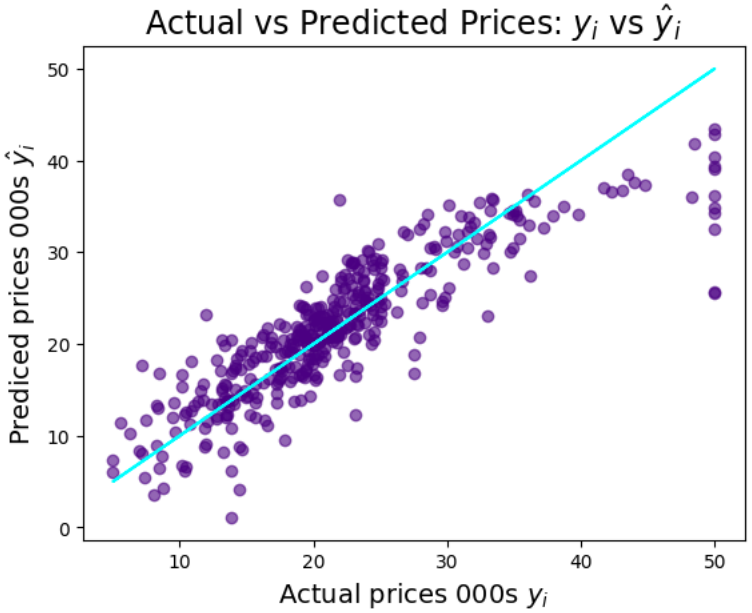
The cyan line in the middle shows `y_train` against `y_train`. If the predictions had been 100% accurate then all the dots would be on this line.
The further away the dots are from the line, the worse the prediction was. That makes the distance to the cyan line, you guessed it, our residuals
😊

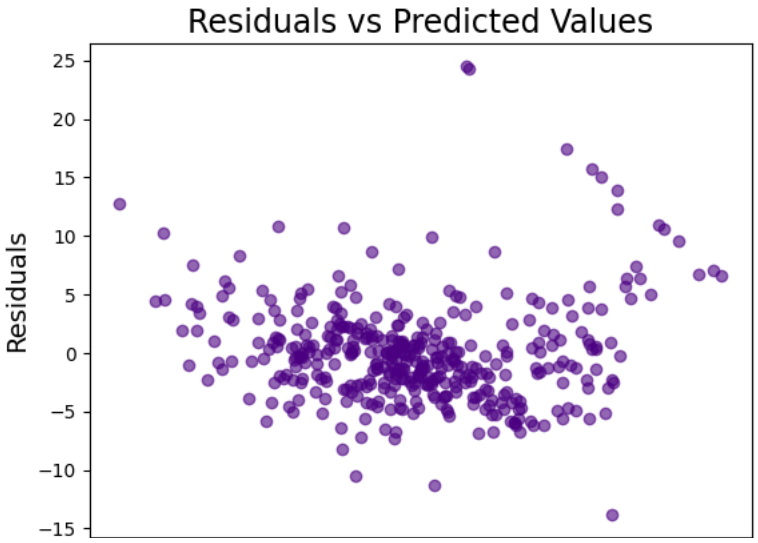
The second plot should be the residuals against the predicted prices. Here's what we're looking for:




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

```
# Original Regression of Actual vs. Predicted Prices
plt.figure(dpi=100)
plt.scatter(x=y_train, y=predicted_vals, 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()

# Residuals vs Predicted values
plt.figure(dpi=100)
plt.scatter(x=predicted_vals, 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()
```

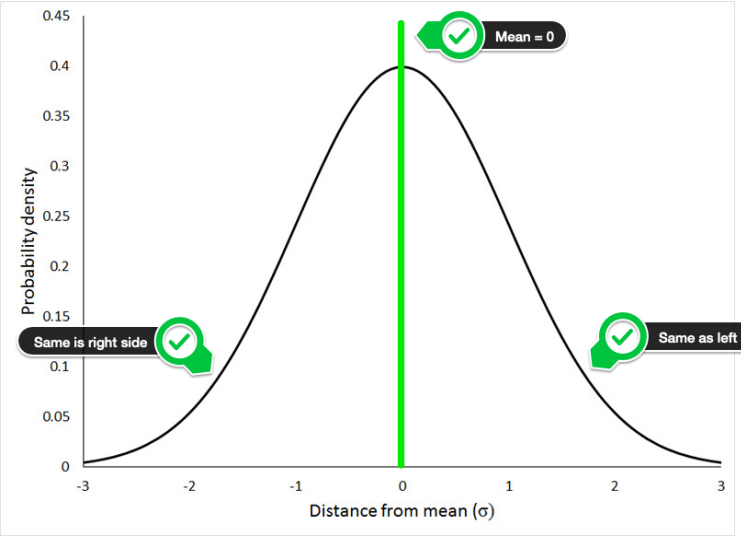




Why do we want to look at the residuals? We want to check that they look random. Why? The residuals represent the errors of our model. If there's a pattern in our errors, then our model has a systematic bias.

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

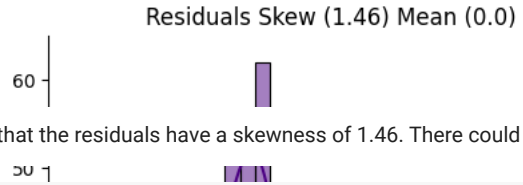
In an ideal case, what we want is something close to a normal distribution. A normal distribution has a skewness of 0 and a mean of 0. A skew of 0 means that the distribution is symmetrical - the bell curve is not lopsided or biased to one side. Here's what a normal distribution looks like:



Challenge

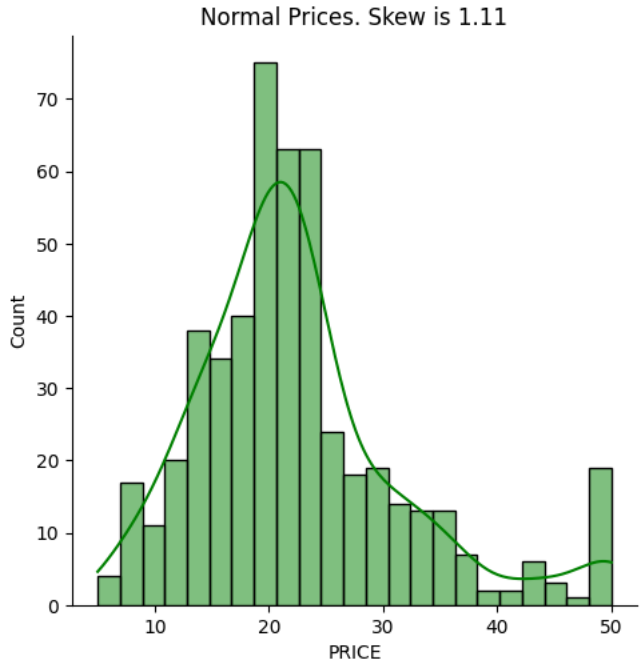
- Calculate the mean and the skewness of the residuals.
- Again, use Seaborn's `.displot()` to create a histogram and superimpose the Kernel Density Estimate (KDE)

```
# 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 ({resid_mean})')
plt.show()
```

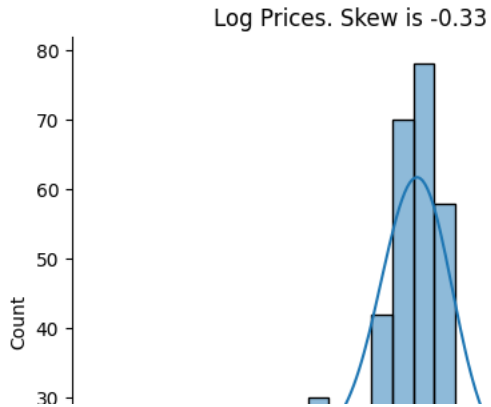


We see that the residuals have a skewness of 1.46. There could be some room for improvement here.

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




```
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()
```



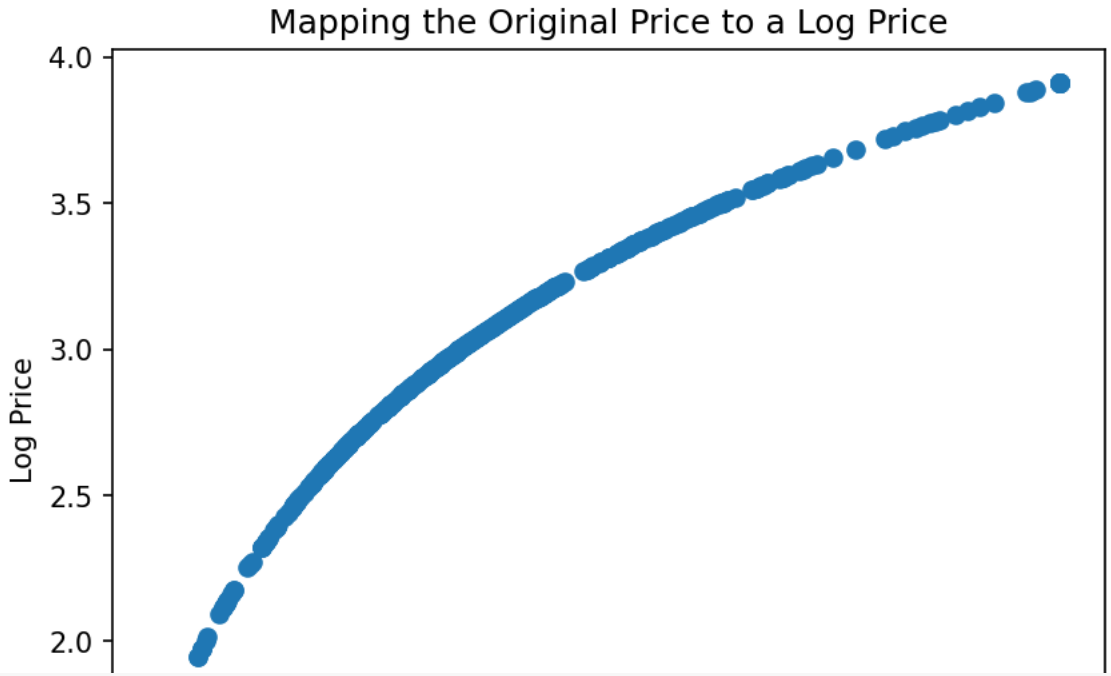
The log prices have a skew that's closer to zero. This makes them a good candidate for use in our linear model. Perhaps using log prices will improve our regression's r-squared and our model's residuals.

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.

```
plt.figure(dpi=150)
plt.scatter(data.PRICE, np.log(data.PRICE))

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




```
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
```

This time we got an r-squared of 0.79 compared to 0.75. This looks like a promising improvement.

Evaluating Coefficients with Log Prices

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

	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

So how can we interpret the coefficients? The key thing we look for is still the sign - being close to the river results in higher property prices because CHAS has a coefficient greater than zero. Therefore property prices are higher next to the river.

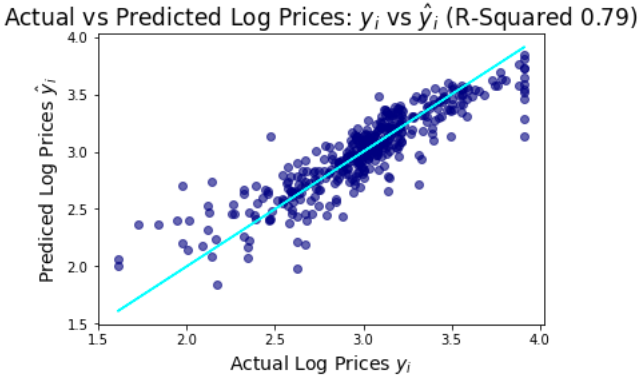
More students per teacher - a higher PTRATIO - is a clear negative. Smaller classroom sizes are indicative of higher quality education, so have a negative coefficient for PTRATIO.

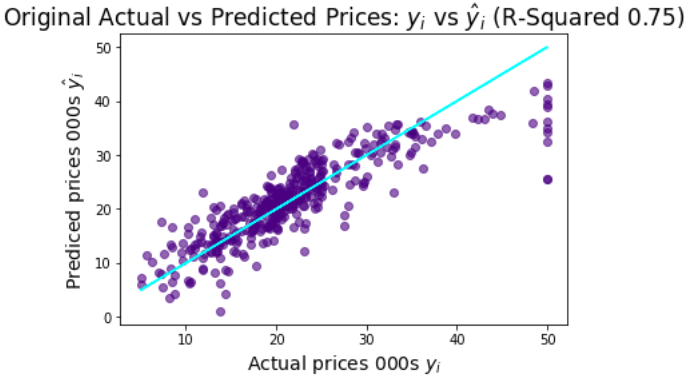
Regression with Log Prices & Residual Plots

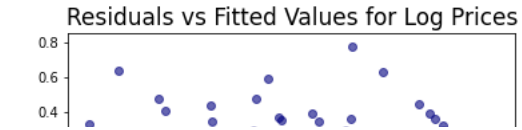
Challenge:

- 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.

```
# 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_vals, 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_vals, 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()
```



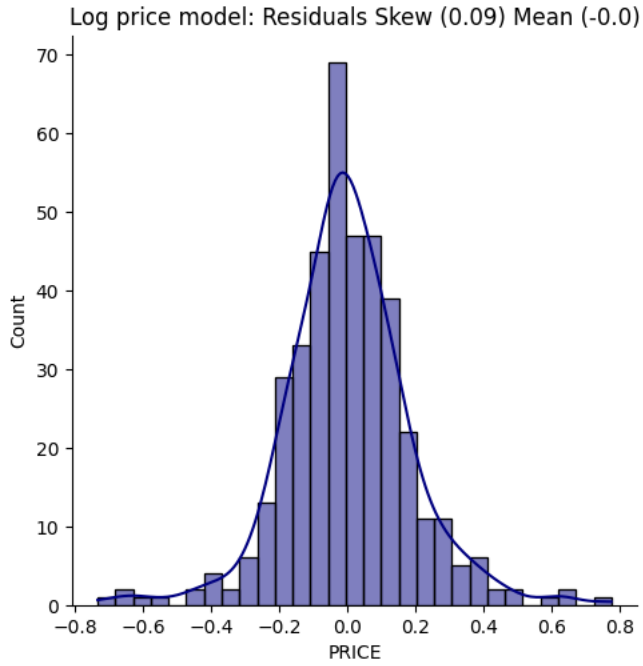


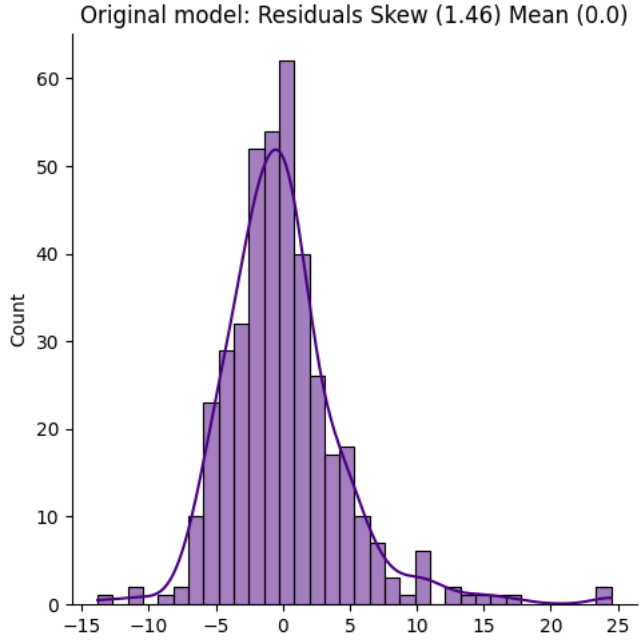


It's hard to see a difference here just by eye. The predicted values seems slightly closer to the cyan line, but eyeballing the charts is not terribly helpful in this case.

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?

```
# 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()
```





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.

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.

```
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
```

By definition, the model has not been optimised for the testing data. Therefore performance will be worse than on the training data. However, our r-squared still remains high, so we have built a useful model.

Predict a Property's Value using the Regression Coefficients

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

$$\log(\hat{PRICE}) = \theta_0 + \theta_1 RM + \theta_2 NOX + \theta_3 DIS + \theta_4 CHAS + \ldots + \theta_{13} LSTAT$$

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

```
# 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
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
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	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.

```
# 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
# 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
```