No description has been provided for this image No description has been provided for this image

Project: Predicting Boston Housing Prices

No description has been provided for this image

Description

This dataset is sourced from the **UCI Machine Learning Repository** and concerns housing values in the suburbs of Boston.

Content

- **Number of Instances**: 506

Attribute Information

Feature	Description
`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`	Percentage of the population with lower socioeconomic status.
`MEDV`	Median value of owner-occupied homes in $1000's (Target variable).

Usage

This dataset is often used for:

1. Regression problems to predict `MEDV` (housing price).
2. Exploratory data analysis and feature engineering practice.

Instructions

You will build a linear regression model to predict housing prices using the Boston Housing Prices dataset. Complete each task step-by-step as outlined below.

Expected Learning Outcomes

- Normalize data using `StandardScaler`.
- Perform a train-test split.
- Build a machine learning pipeline.
- Evaluate a regression model using MSE and (R^2).

In [29]:
```
# Prepare Tools

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

import sklearn.linear_model
import sklearn.metrics
import sklearn.preprocessing
import sklearn.pipeline
import sklearn.datasets

from sklearn.model_selection import train_test_split
```

Step 1: Load and Explore the Dataset

1. **Load the Boston Housing Prices dataset using `../data/housing.csv`.**
   - Display the first 5 rows of the dataset as a pandas DataFrame.
   - Assign the features to (X) and the target variable (`MEDV`) to (y).

   Hint:
   - Use `pd.DataFrame` to convert the dataset into a DataFrame.
   - The target variable can be accessed via `df.MEDV`.

In [30]:
```
# Your code here
df = pd.read_csv('../data/housing.csv')
df.head()
```

Out[30]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

1A) Assign the feature (X) and the target variable (MEDV) to (Y)

In [31]:
```
x = df.drop('MEDV', axis = 1)
y= df['MEDV']
```

In [32]:
```
x.head() #This is independet varaible (x)
```

Out[32]:

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

In [33]:
```
y.head()  #This is dependet varaible (Y)
```

Out[33]:
```
0    24.0
1    21.6
2    34.7
3    33.4
4    36.2
Name: MEDV, dtype: float64
```

Step 2: Normalize the Features

2. **Use `StandardScaler` to normalize the features in (X).**
   - Apply normalization **only after splitting** the data into training and testing sets.

   Hint:
   - First, perform train-test split using `train_test_split`.
   - Fit the scaler on the training data only and transform both training and testing data.

2A) scale the X data using standardscaler

In [34]:
```
#Normalize the Features using StandardScaler
from sklearn.preprocessing import StandardScaler

#initialize the standardscaler
scaler = StandardScaler()

#fit the scaler on the training and test data (X)
scaler.fit(x_train)
scaler.fit(x_test)

#Transform both training and testing
x_train_scaled = scaler.transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

Step 3: Split the Data

3. **Split the dataset into training and testing sets with an 80-20 split.**

- Assign the result to variables $X_{\text{train}}, X_{\text{test}}, y_{\text{train}}, y_{\text{test}}$.

  Hint:
  - Use `train_test_split` from `sklearn.model_selection`.

In [35]:
```
from sklearn.model_selection import train_test_split
#split the data into train and test
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2, random_state=44)
```

In [36]:
```
print("**This is Training set**")
print(f'This is x train {x_train.shape}')
print(f'This is y train {y_train.shape}')
print("")
print("**This is Test set**")
print(f'This is x test {x_test.shape}')
print(f'This is y test {y_test.shape}')
```

```
**This is Training set**
This is x train (404, 13)
This is y train (404,)

**This is Test set**
This is x test (102, 13)
This is y test (102,)
```

Step 4: Create a Pipeline

4. **Build a pipeline that includes:**
   - `StandardScaler` for normalization.
   - `LinearRegression` for fitting the model.
   - Fit the pipeline to the training data.

   Hint:
   - Use `Pipeline` from `sklearn.pipeline`.

4A) Create pipeline

In [37]:
```
# Create a Pipeline
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression

pipeline = Pipeline([
    ('scaler', StandardScaler()),  # First step: StandardScaler for normalization
    ('model', LinearRegression())     # Second step: LinearRegression for model fitting
])

# Fit the pipeline to the training data
pipeline.fit(x_train, y_train)
```

Out[37]:
```
Pipeline
  StandardScaler
  LinearRegression
```

4B) Show the COEF

In [38]:
```
from sklearn.linear_model import LinearRegression

# Optional: Check the model's coefficients
print("Model Coefficients:", pipeline.named_steps['model'].coef_)
```

```
Model Coefficients: [-0.90993944  1.22792296 -0.02371233  0.85736904 -2.23765701  2.20948448
  0.05613783 -3.8096152   2.67329749 -1.87344214 -2.09548097  0.90396745
 -4.16220947]
```

4C) Interpret the coef

Coefficient -1.00213533: This means that if the first feature (CRIM - crime rate per capita) increases by one unit, the house price (MEDV target) is expected to decrease by 1.002 units (in the same units as the target MEDV).

Coefficient 3.14523956: This means that if the sixth feature (RM - average number of rooms) increases by one unit, the house price is expected to increase by 3.145 units.

Step 5: Evaluate the Model

5. **Use the pipeline to make predictions on the test data.**
   - Evaluate the model using:
     - Mean Squared Error (MSE).
     - R-squared ((R^2)).

   Hint:
   - Use `mean_squared_error` and `r2_score` from `sklearn.metrics`.

5A) Evaluate the model using MSE and R2

In [39]:
```
from sklearn.metrics import mean_squared_error, r2_score

# predict on the test data
y_pred = pipeline.predict(x_test)

# Calculate the Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)

# Calculate R-squared (R²)
r2 = r2_score(y_test, y_pred)

# Print the results
print("Mean Squared Error (MSE):", mse)
print("R-squared (R²):", r2)
```

```
Mean Squared Error (MSE): 20.93275021868356
R-squared (R²): 0.7521800808693159
```

5B) Interpret the data meaning

MSE (20.93) indicates that the model still has errors in predicting house prices, but this value alone is not enough to assess how good the model is.

R^2 (0.75) shows that 75% of the variation in house prices can be explained by the model, while the remaining 25% may be influenced by other factors not captured in the model.

In summary, this model can be considered to perform reasonably well because it explains almost 75% of the variation in house prices. However, there is still 25% that cannot be explained, which could be improved by adding more data or using a more advanced model.

Step 6: Visualize the Results (Optional)

6. **Create a scatter plot comparing the predicted prices vs actual prices for the test data.**

   Hint:
   - Use `plt.scatter` from `matplotlib.pyplot`.

6A) Visualize actual and prediction line using scatter plot

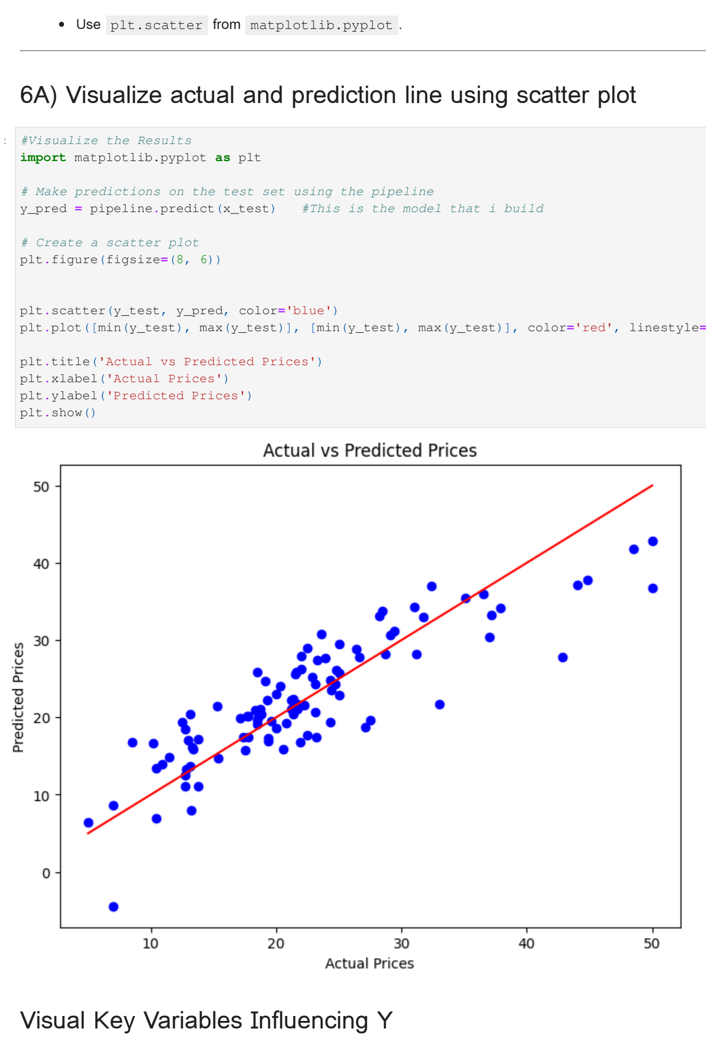
In [40]:
```
#Visualize the Results
import matplotlib.pyplot as plt

# Make predictions on the test set using the pipeline
y_pred = pipeline.predict(x_test)   #This is the model that i build

# Create a scatter plot
plt.figure(figsize=(8, 6))


plt.scatter(y_test, y_pred, color='blue')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='-')  # Line for perfect prediction

plt.title('Actual vs Predicted Prices')
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.show()
```



Visual Key Variables Influencing Y

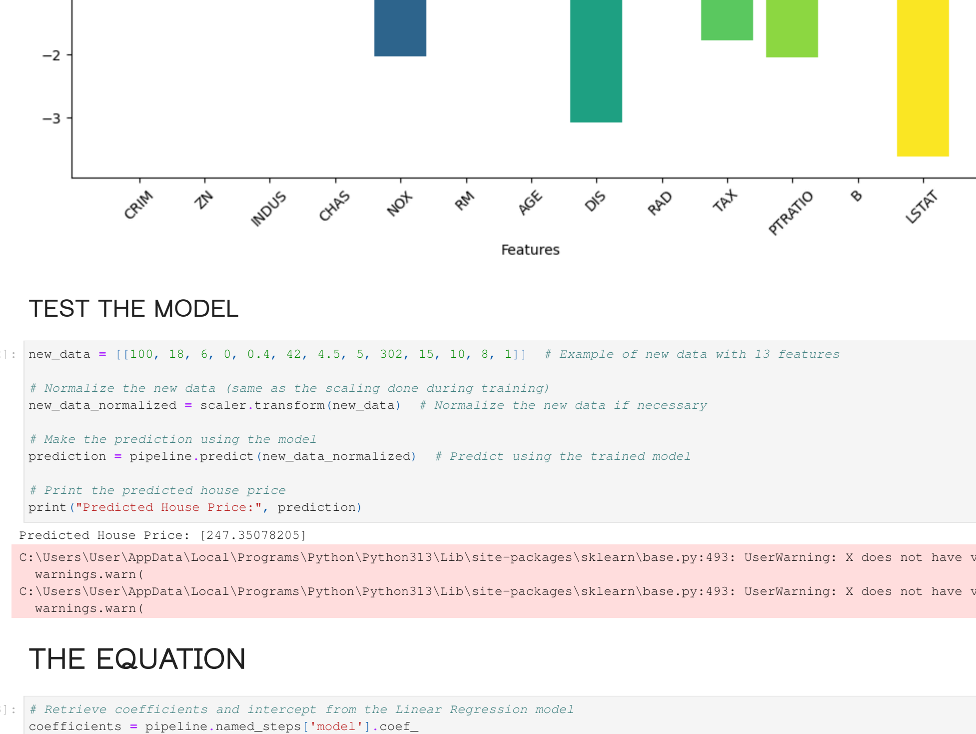
In [41]:
```
import matplotlib.pyplot as plt
import numpy as np

# Names of the features
features = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']
importances = [-1.00213533, 0.69626862, 0.27806485, 0.7187384, -2.0223194, 3.14523956,
               -0.17604788, -3.0819076, 2.25140666, -1.76701378, -2.03775151, 1.12956831, -3.61165842]

# Generate colors automatically using a colormap
colors = plt.cm.viridis(np.linspace(0, 1, len(features)))

# Plot the bar chart
plt.figure(figsize=(10, 6))
plt.bar(features, importances, color=colors)

plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Feature Importances in Dataset')
plt.xticks(rotation=45)
plt.tight_layout()  # Adjust layout for better spacing
plt.show()
```



TEST THE MODEL

In [42]:
```
new_data = [[100, 18, 6, 0, 0.4, 42, 4.5, 5, 302, 15, 10, 8, 1]]  # Example of new data with 13 features

# Normalize the new data (same as the scaling done during training)
new_data_normalized = scaler.transform(new_data)  # Normalize the new data if necessary

# Make the prediction using the model
prediction = pipeline.predict(new_data_normalized)  # Predict using the trained model

# Print the predicted house price
print("Predicted House Price:", prediction)
```

```
Predicted House Price: [247.35078205]
```

```
C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\base.py:493: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\base.py:493: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```

THE EQUATION

In [43]:
```
# Retrieve coefficients and intercept from the Linear Regression model
coefficients = pipeline.named_steps['model'].coef_
intercept = pipeline.named_steps['model'].intercept_

# Print the equation
print("Equation: y = {:.2f} + ".format(intercept) +
      " + ".join(["{:.2f}*x{}".format(coeff, i) for i, coeff in enumerate(coefficients, start=1)]))
```

```
Equation: y = 22.59 + -0.91*x1 + 1.23*x2 + -0.02*x3 + 0.86*x4 + -2.24*x5 + 2.21*x6 + 0.06*x7 + -3.81*x8 + 2.67*x9 + -1.87*x10 + -2.10*x11 + 0.90*x12 + -4.16*x13
```

SAVE THE MODEL

In [49]:
```
import joblib

# Save pipeline file
joblib.dump(pipeline, 'model_pipeline.pkl')  # can change the name file if want
print("Model has already been save")
```

```
Model has already been save
```

LOAD THE MODEL

In [50]:
```
# Load the saved model / You can also load another notebook
loaded_pipeline = joblib.load('model_pipeline.pkl')  # Make sure the filename is correct or provide the full path if necessary
print("Model has already been loaded")
```

```
Model has already been loaded
```

USE MODEL AND MAKE PREDICT

In [52]:
```
# New data for prediction
new_data = [[100, 18, 6, 0, 0.4, 42, 4.5, 5, 302, 15, 10, 8, 99]]  # Example of new data

# Make prediction using the loaded model
prediction = loaded_pipeline.predict(new_data)

# Print the prediction result
print("Predicted Value:", prediction)
```

```
Predicted Value: [172.77812893]
```

```
C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\base.py:493: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```

In [ ]:

In [ ]: