TOULOUSE LAUTREC

APRENDIZAJE AUTOMATICO CON PYTHON

METODOS DE REGRESION

Ing. Alexander Valdez

Curso 2290, Clases Lunes y Miercoles 20:00-22:30pm

Segunda Clase

Boston Dataset

- Contains information about different houses in Boston.
- There are 506 samples and 13 feature variables in this dataset.
- Maintained at Carnegie Mellon University.
- This is a copy of UCI ML housing dataset.

We want to predict the value of prices of the house using the given features.

In [ ]:
```
import pandas as pd
boston_data = pd.read_csv("https://raw.githubusercontent.com/sebastianVP/ML_PROJECT/mast
boston_data.head()
```

Out[ ]:

	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

In [ ]:
```
boston_data.values[0,:-1]
```

Out[ ]:
```
array([6.320e-03, 1.800e+01, 2.310e+00, 0.000e+00, 5.380e-01, 6.575e+00,
       6.520e+01, 4.090e+00, 1.000e+00, 2.960e+02, 1.530e+01, 3.969e+02,
```

```
       4.980e+00])
```

In [ ]:
```
bos_pd = pd.DataFrame(boston_data.values[:,:-1])
bos_pd.head()
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

Attribute Information:

Acronym	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	roportion 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

In [ ]:
```
print("Keys: ", boston_data.keys())
```

```
Keys:  Index(['crim', 'zn', 'indus', 'chas', 'nox', 'rm', 'age', 'dis', 'rad', 'tax',
       'ptratio', 'b', 'lstat', 'medv'],
      dtype='object')
```

In [ ]:
```
bos_pd.columns = ['CRIM','ZN','INDUS','CHAS','NOX','RM','AGE','DIS','RAD','TAX','PTRATIO
bos_pd.head()
```

Out[ ]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [ ]:
```
bos_pd['PRICE']=boston_data.values[:,-1]
bos_pd.head()
```

Out[ ]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

Pass the data into a Pandas dataframe

Exploratory Data Analysis

- Important step before training the model.
- We use statistical analysis and visualizations to understand the relationship of the target variable with other features.

Check Missing Values

It is a good practice to see if there are any missing values in the data.

Count the number of missing values for each feature

In [ ]:
```
bos_pd.isnull().sum()
```

Out[ ]:
```
CRIM        0
ZN          0
INDUS       0
CHAS        0
NOX         0
RM          0
AGE         0
DIS         0
RAD         0
TAX         0
PTRATIO     0
B           0
LSTAT       0
PRICE       0
dtype: int64
```

Obtain basic statistics on the data

In [ ]:
```
bos_pd
```

Out[ ]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7

3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88	11.9

506 rows × 14 columns

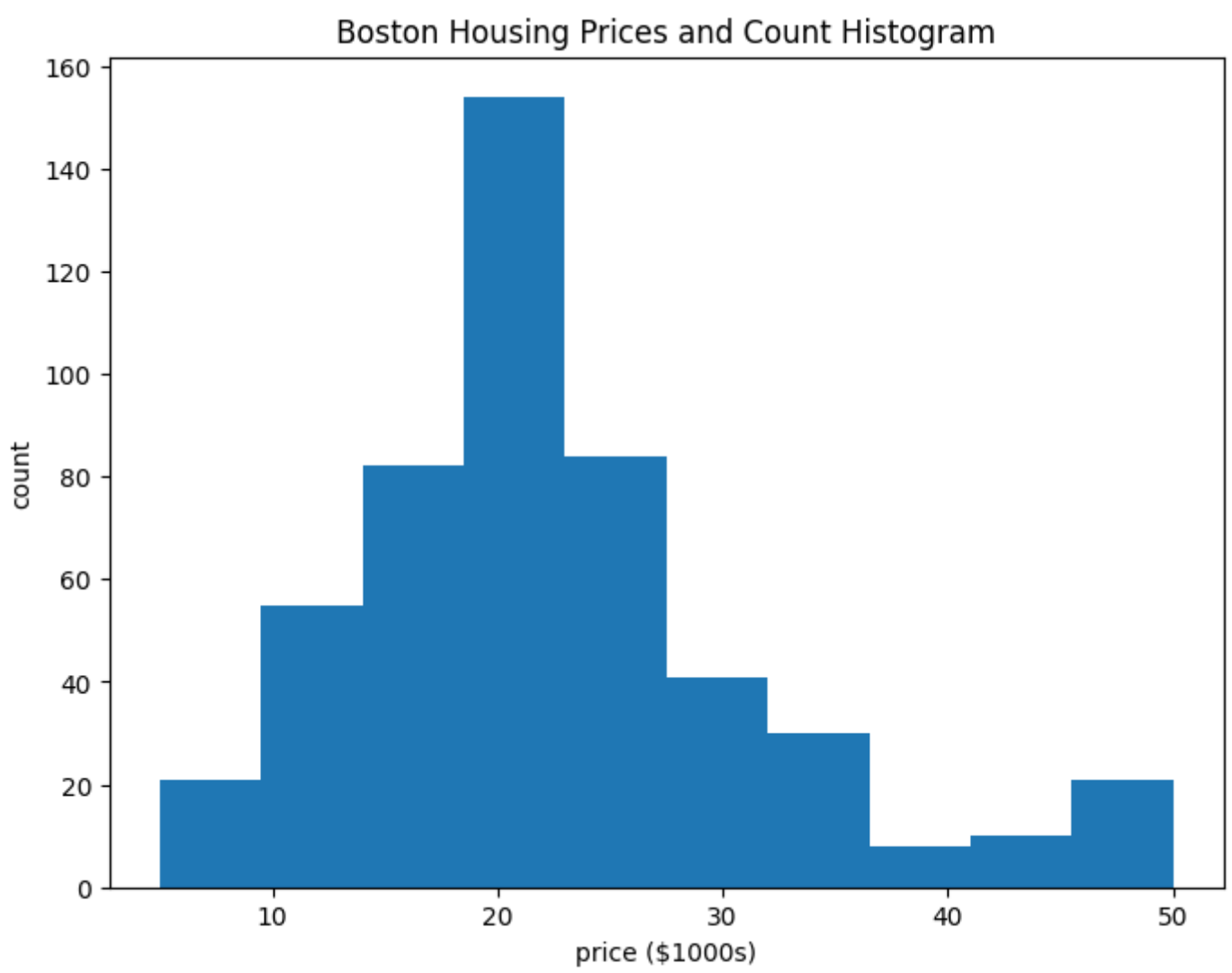
In [ ]:
```
bos_pd.describe().transpose()
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	max
CRIM	506.0	3.613524	8.601545	0.00632	0.082045	0.25651	3.677083	88.9762
ZN	506.0	11.363636	23.322453	0.00000	0.000000	0.00000	12.500000	100.0000
INDUS	506.0	11.136779	6.860353	0.46000	5.190000	9.69000	18.100000	27.7400
CHAS	506.0	0.069170	0.253994	0.00000	0.000000	0.00000	0.000000	1.0000
NOX	506.0	0.554695	0.115878	0.38500	0.449000	0.53800	0.624000	0.8710
RM	506.0	6.284634	0.702617	3.56100	5.885500	6.20850	6.623500	8.7800
AGE	506.0	68.574901	28.148861	2.90000	45.025000	77.50000	94.075000	100.0000
DIS	506.0	3.795043	2.105710	1.12960	2.100175	3.20745	5.188425	12.1265
RAD	506.0	9.549407	8.707259	1.00000	4.000000	5.00000	24.000000	24.0000
TAX	506.0	408.237154	168.537116	187.00000	279.000000	330.00000	666.000000	711.0000
PTRATIO	506.0	18.455534	2.164946	12.60000	17.400000	19.05000	20.200000	22.0000
B	506.0	356.674032	91.294864	0.32000	375.377500	391.44000	396.225000	396.9000
LSTAT	506.0	12.653063	7.141062	1.73000	6.950000	11.36000	16.955000	37.9700
PRICE	506.0	22.532806	9.197104	5.00000	17.025000	21.20000	25.000000	50.0000

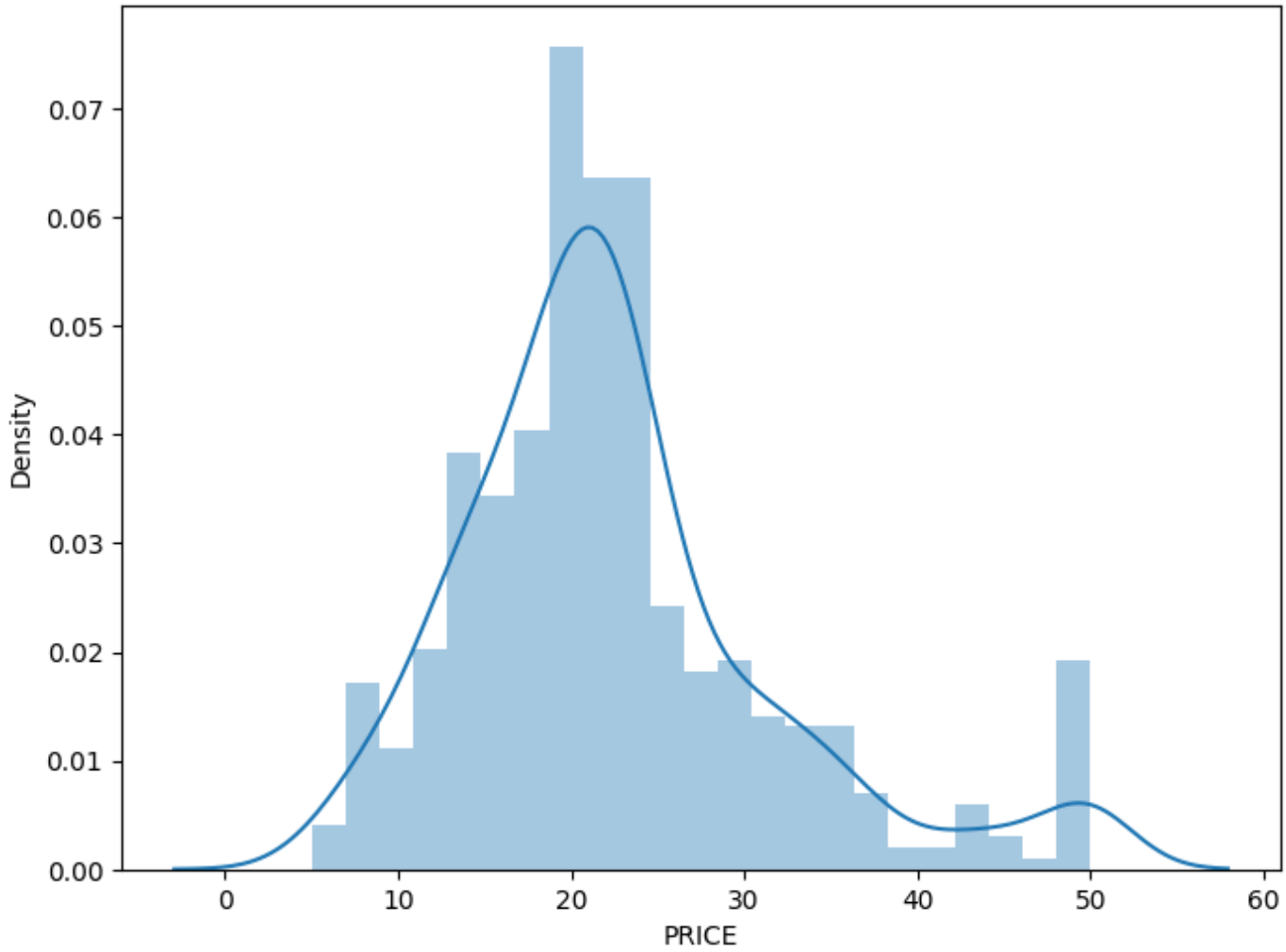
Distribution of the target variable

In [ ]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6));
plt.hist(bos_pd['PRICE']);
plt.title('Boston Housing Prices and Count Histogram');
plt.xlabel('price ($1000s)');
plt.ylabel('count');
plt.show();
```



In [ ]:
```
import seaborn as sns

plt.figure(figsize=(8, 6));
sns.distplot(bos_pd['PRICE']);
```

```
<ipython-input-14-b7b47e3d823d>:4: UserWarning:

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

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

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

  sns.distplot(bos_pd['PRICE']);
```

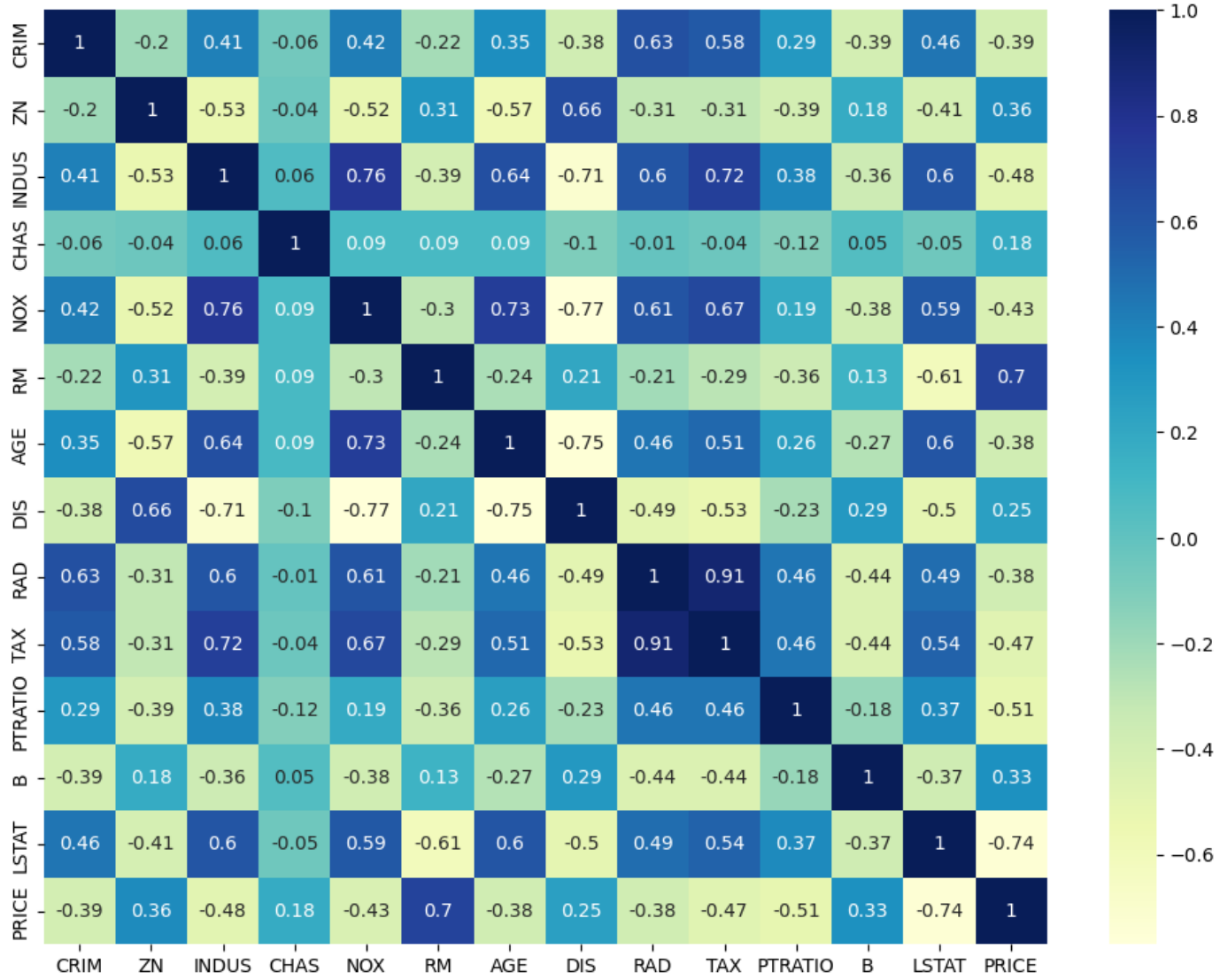


From the above output we can see that the values of PRICE is normally distributed with some of the outliers.

Heatmap: Two-Dimensional Graphical Representation

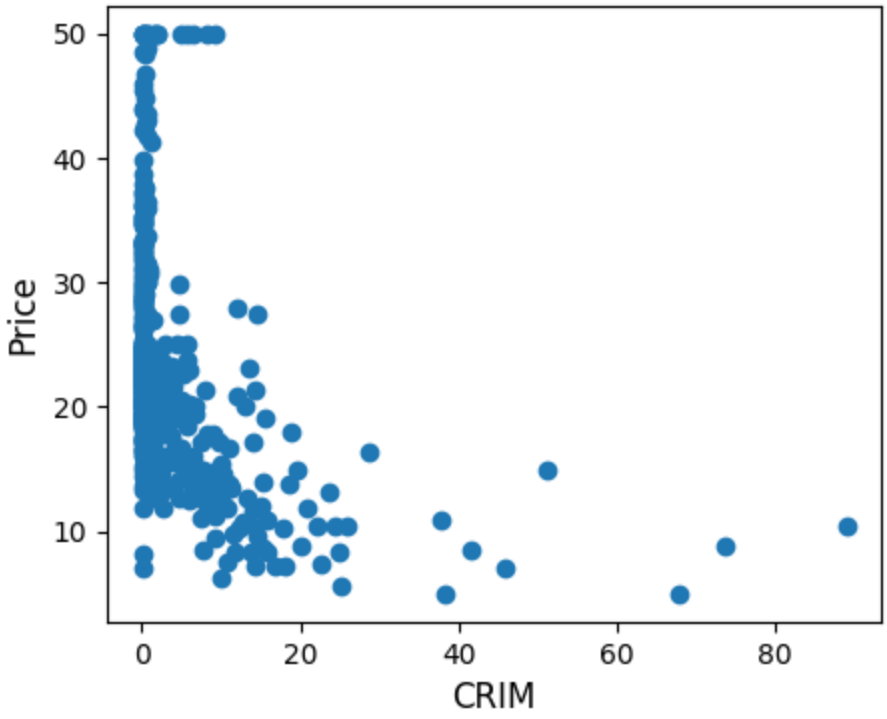
- Represent the individual values that are contained in a matrix as colors.
- Create a correlation matrix that measures the linear relationships between the variables.

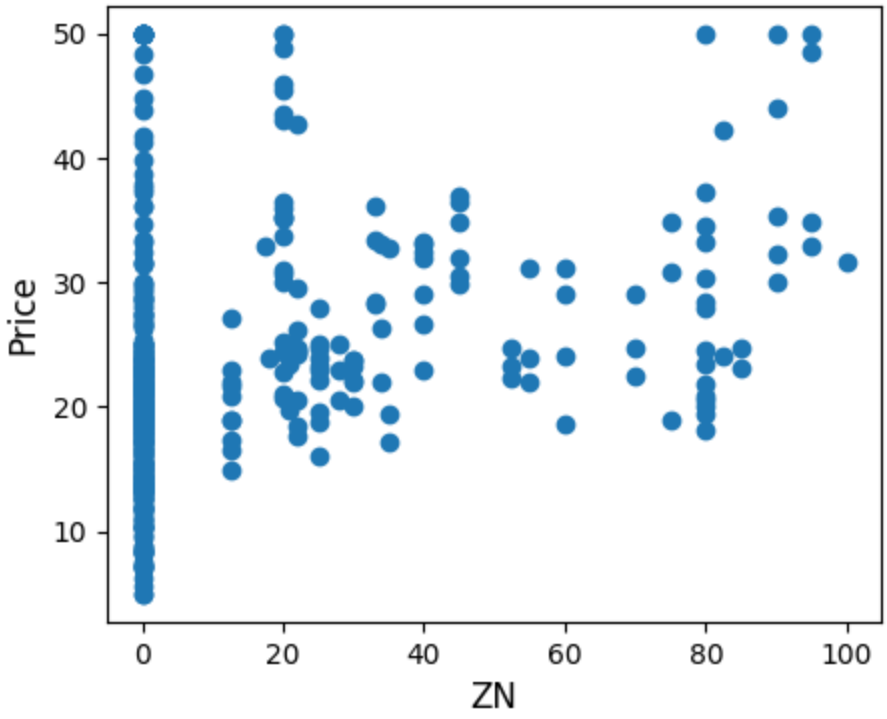
In [ ]:
```
plt.figure(figsize=(12, 9));
correlation_matrix = bos_pd.corr().round(2);
sns.heatmap(correlation_matrix, cmap="YlGnBu", annot=True);
```

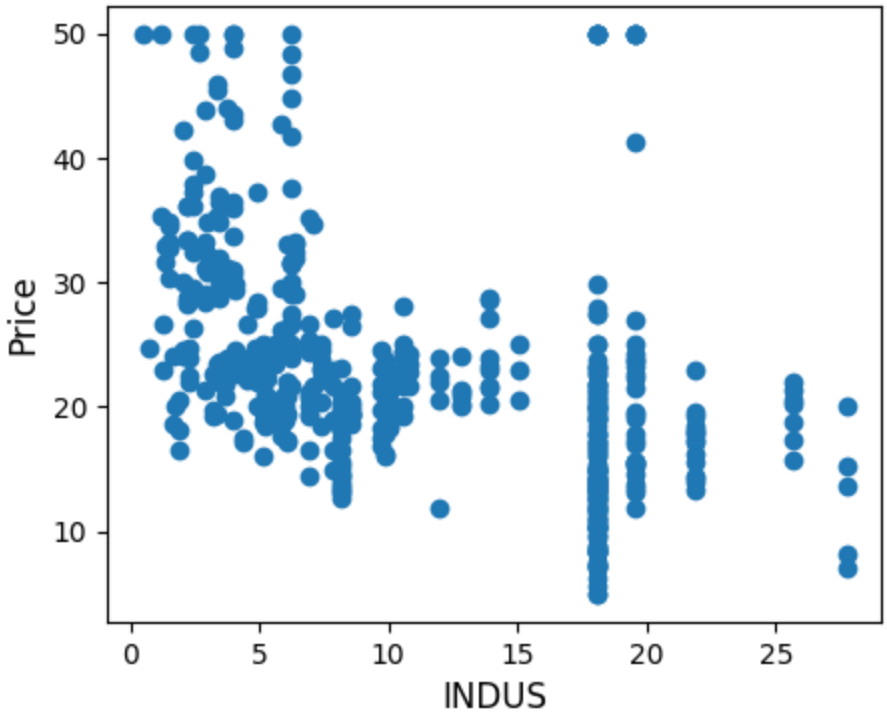


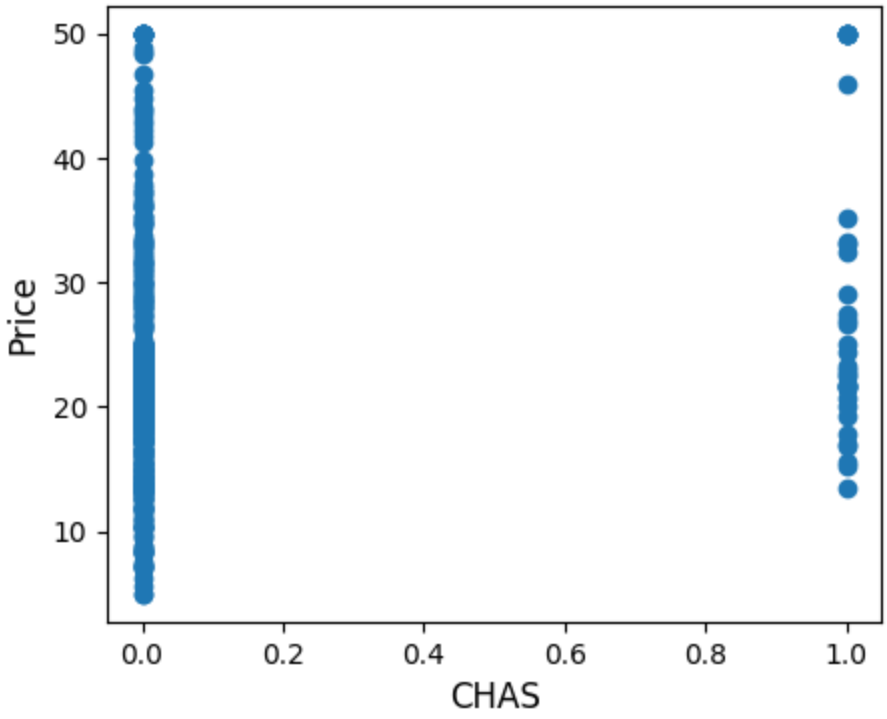
- **RM** tiene una fuerte correlación positiva con **PRICE** (0,7), mientras que **LSTAT tiene** una alta correlación negativa con **PRICE** (-0,74).
- Las características **RAD**, **TAX** tienen una correlación de 0,91. Estos pares de características están fuertemente correlacionados entre sí. Esto puede afectar el modelo. Lo mismo ocurre con las características **DIS** y **AGE** que tienen una correlación de -0,75.
- Las variables predictoras como **CRIM**, **INDUS**, **NOX**, **AGE**, **RAD**, **TAX**, **PTRATIO**, **LSTAT** tiene una correlación negativa con el objetivo. El aumento de cualquiera de ellos conlleva la bajada del precio de la vivienda.
- Las variables predictivas como **ZN**, **RM**, **DIS**, **B** tienen una buena correlación positiva con el objetivo. El aumento de cualquiera de ellos conlleva el aumento del precio de la vivienda.

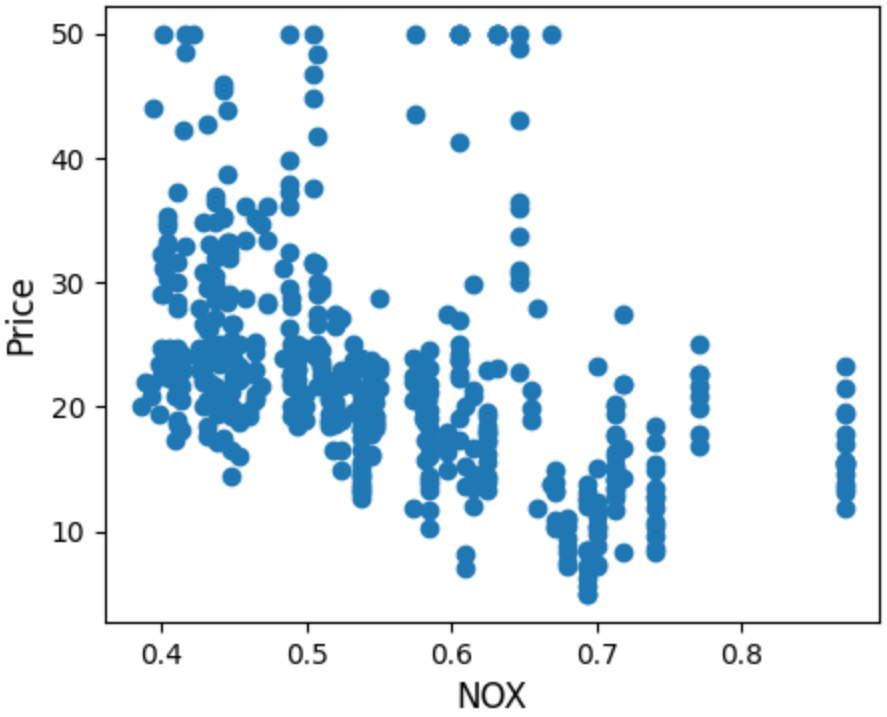
In [ ]:
```
#for feature_name in housing.feature_names:
for feature_name in bos_pd.columns:

    plt.figure(figsize=(5, 4));
    plt.scatter(bos_pd[feature_name], bos_pd['PRICE']);
    plt.ylabel('Price', size=12);
    plt.xlabel(feature_name, size=12);
plt.show();
```

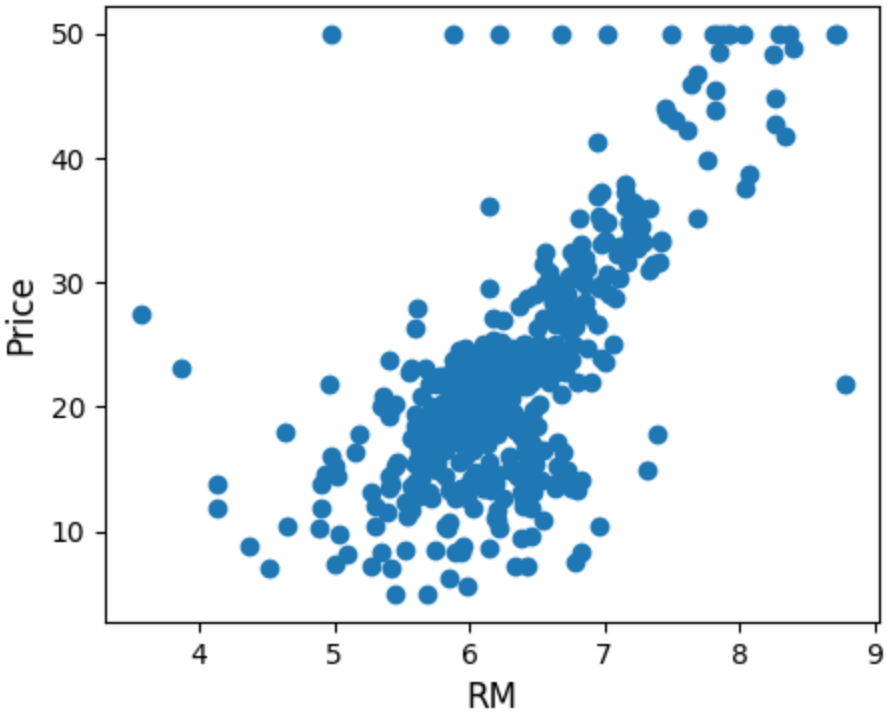

Price
CRIM
50
40
30
20
10
0
20
40
60
80

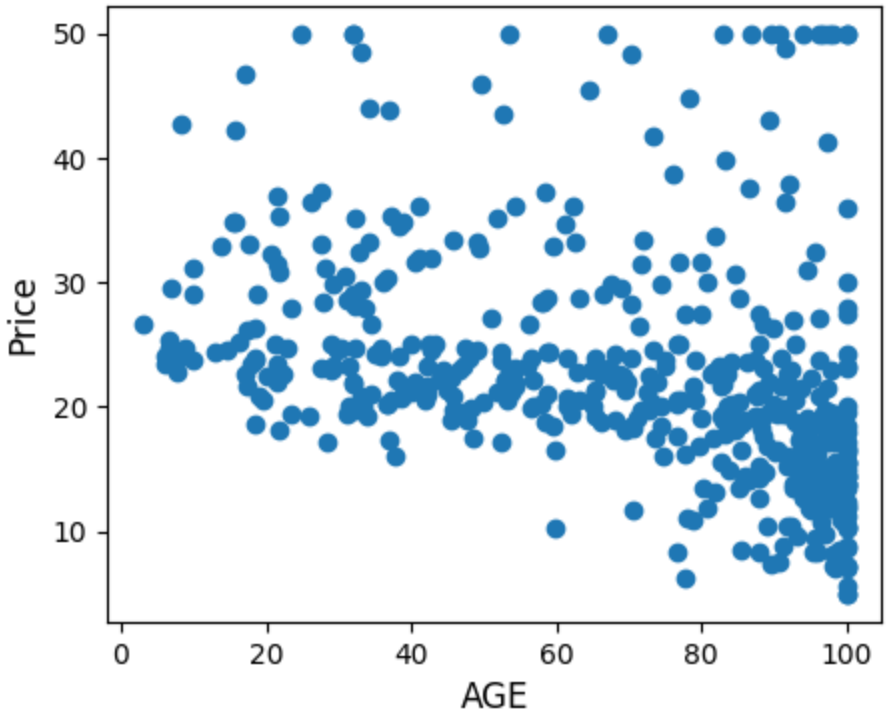

Price
ZN
50
40
30
20
10
0
20
40
60
80
100

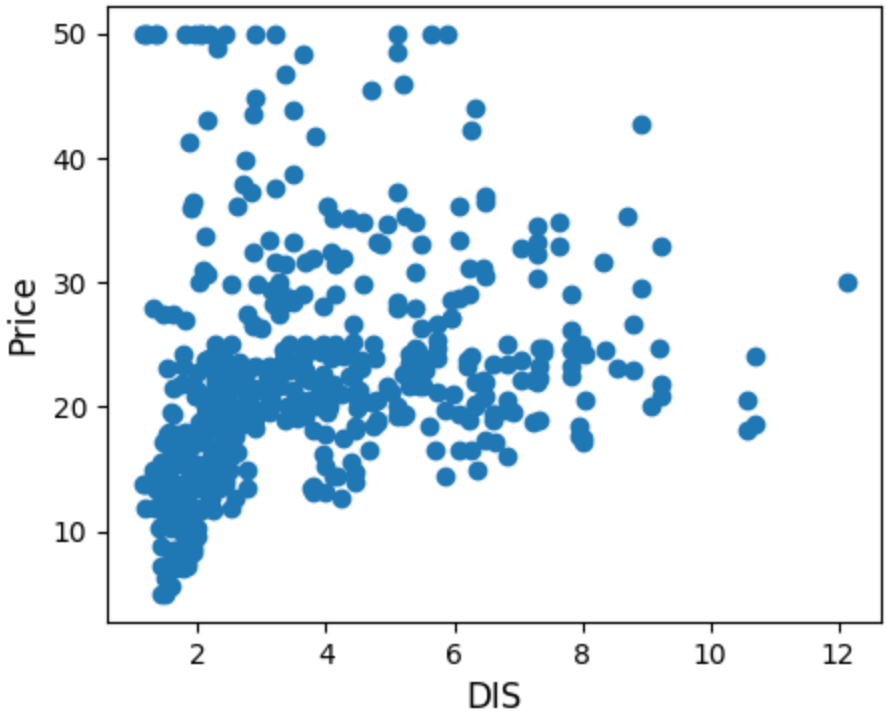

Price
INDUS
50
40
30
20
10
0
5
10
15
20
25

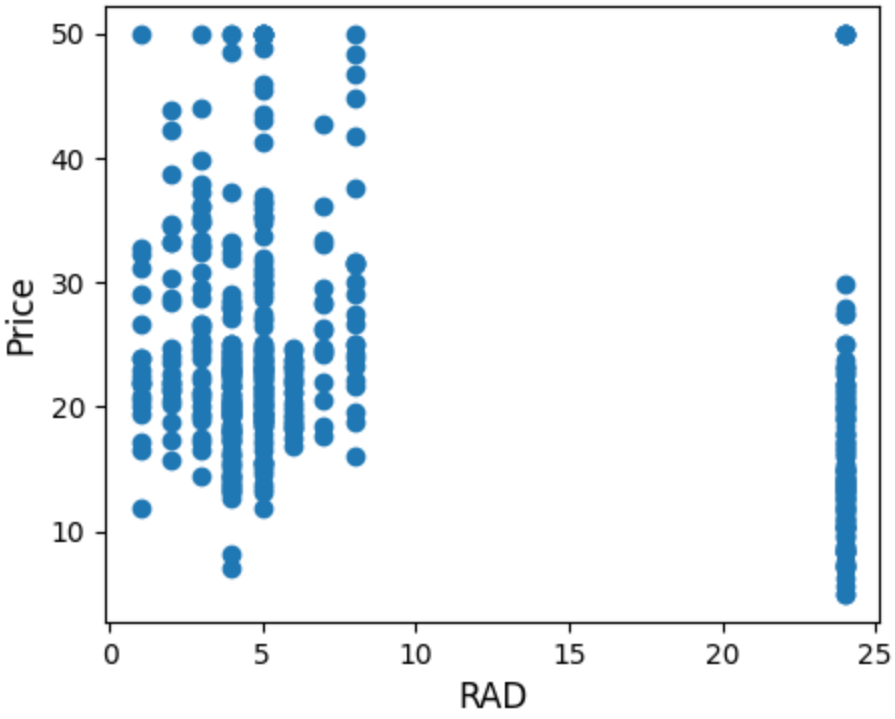

Price
CHAS
50
40
30
20
10
0.0
0.2
0.4
0.6
0.8
1.0

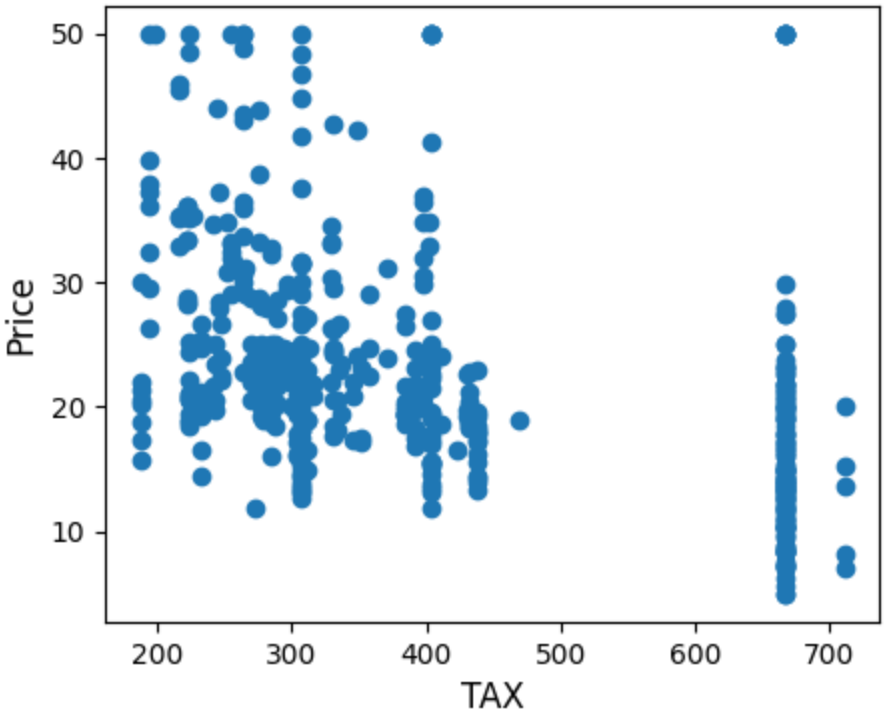

Price
NOX
50
40
30
20
10
0.4
0.5
0.6
0.7
0.8

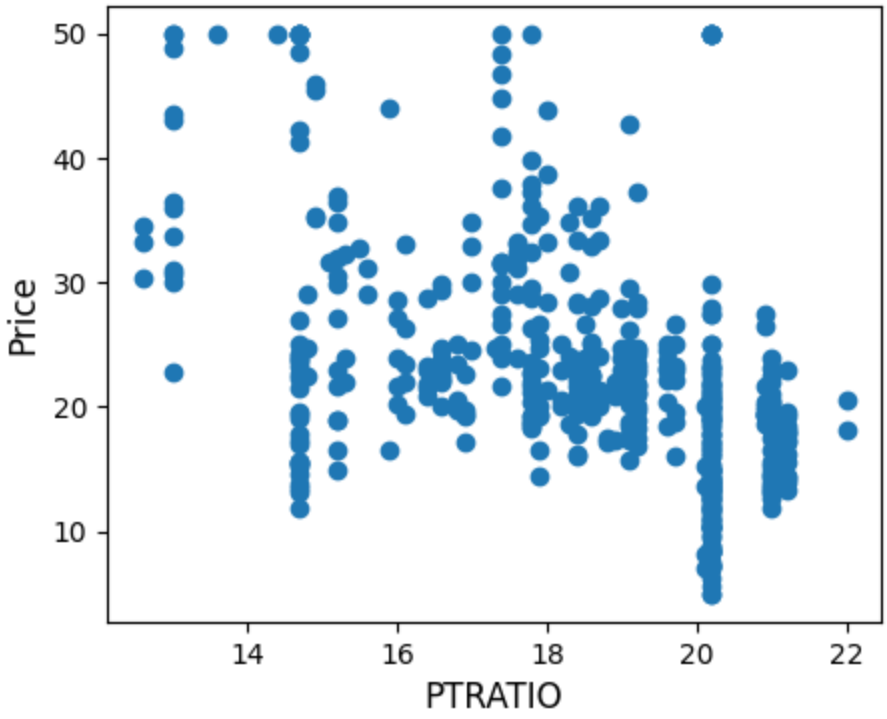

Price
RM
50
40
30
20
10
4
5
6
7
8
9

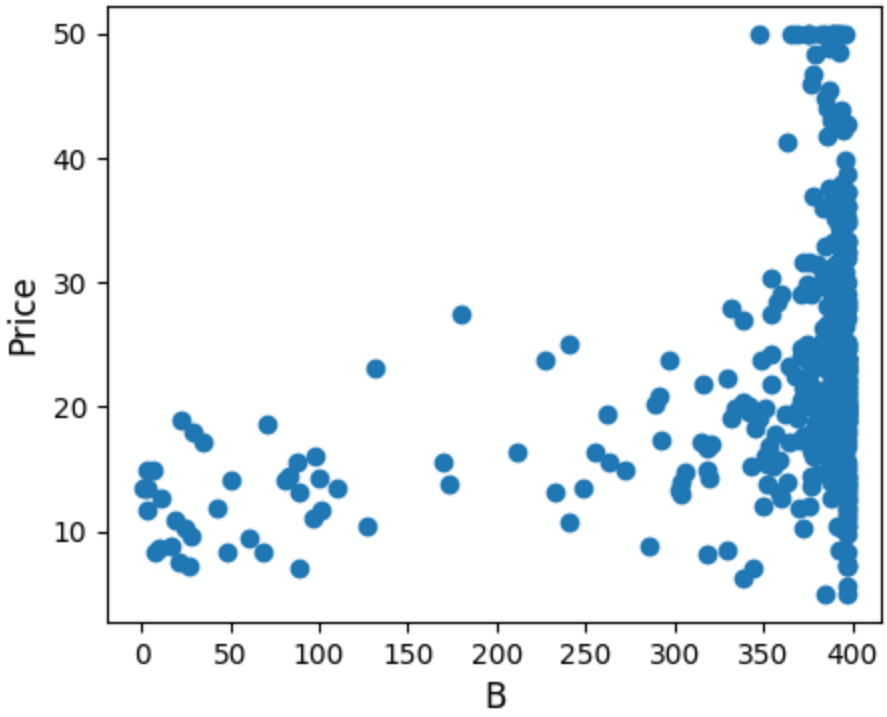

Price
AGE
50
40
30
20
10
0
20
40
60
80
100

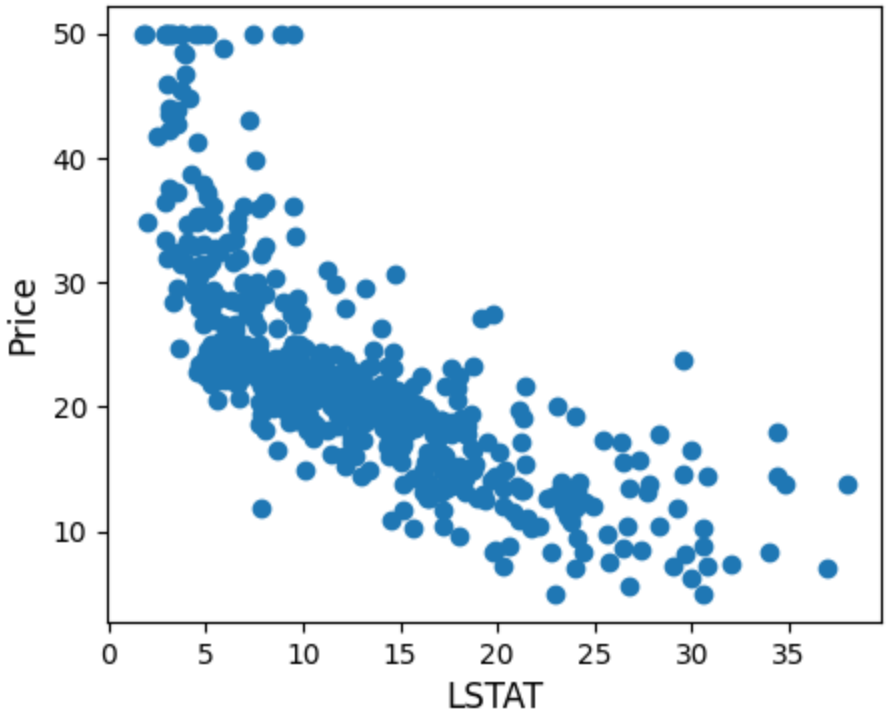

Price
DIS
50
40
30
20
10
2
4
6
8
10
12

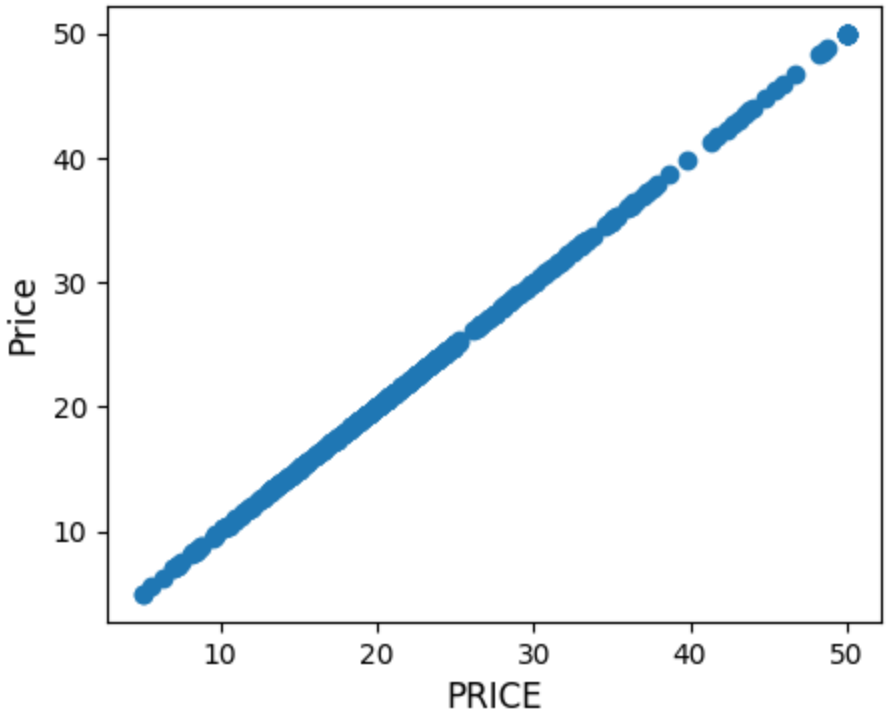

Price
RAD
50
40
30
20
10
0
5
10
15
20
25


Price
TAX
50
40
30
20
10
200
300
400
500
600
700


Price
PTRATIO
50
40
30
20
10
14
16
18
20
22

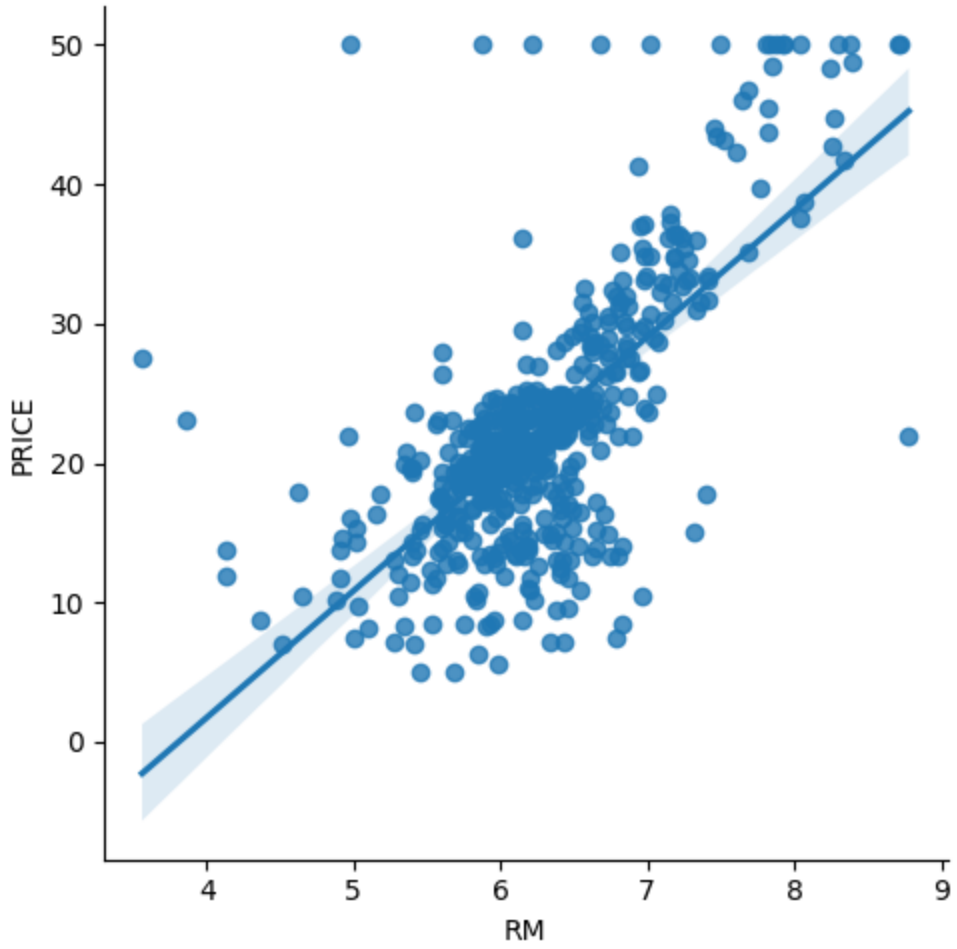

Price
B
50
40
30
20
10
0
50
100
150
200
250
300
350
400





- Los precios aumentan a medida que el valor de RM aumenta linealmente. Hay pocos valores atípicos y los datos parecen tener un límite de 50.
- Los precios tienden a bajar con un aumento de LSTAT. Aunque no parece seguir exactamente una línea lineal.

In [ ]:
```
sns.lmplot(x = 'RM', y = 'PRICE', data = bos_pd);
```



Model Selection Process

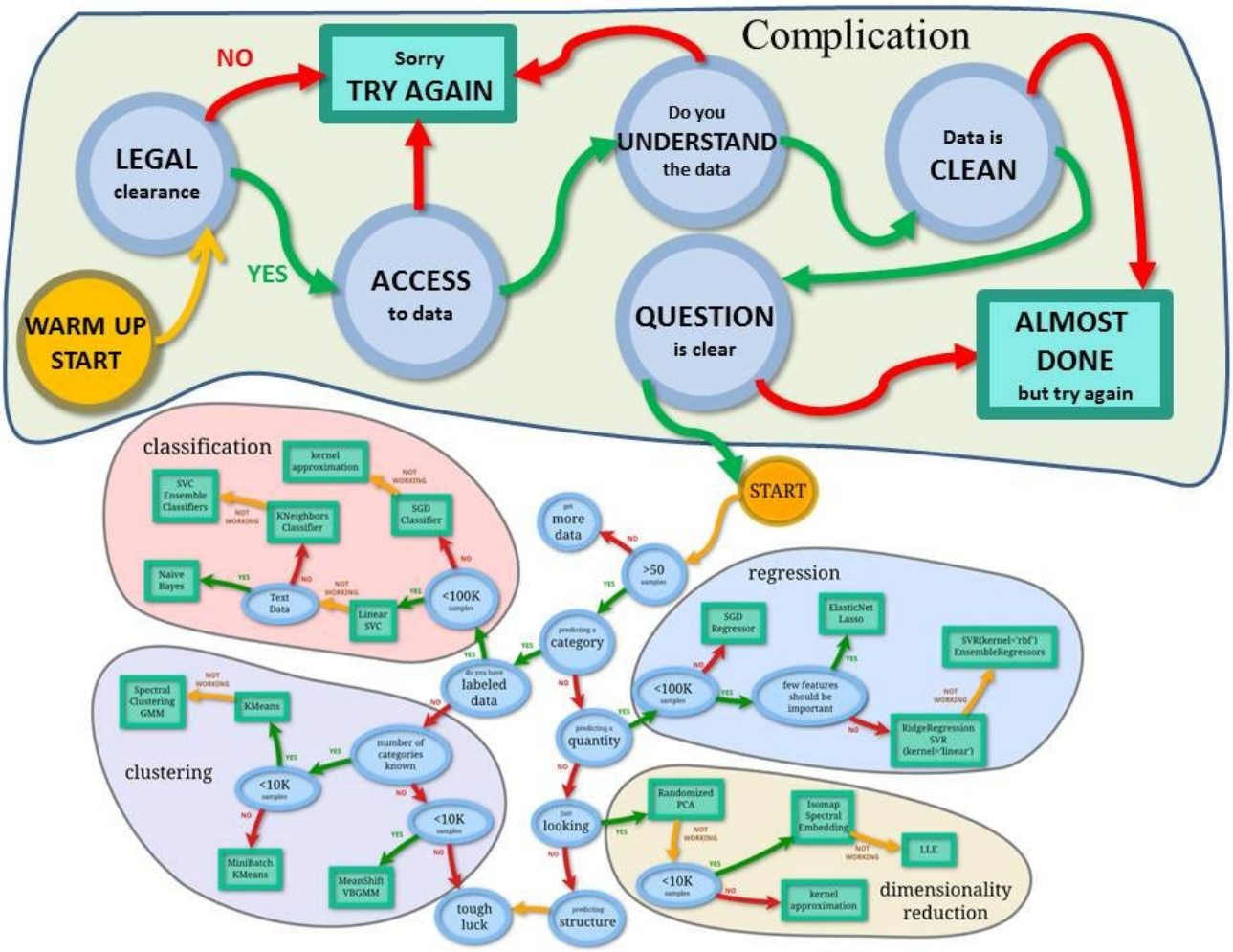


Image Source: Christophe Bourguignat

Es necesario entrenar un algoritmo de aprendizaje automático con un conjunto de datos para aprender las relaciones entre diferentes características y cómo estas características afectan la variable objetivo.

- Necesitamos dividir todo el conjunto de datos en dos conjuntos:
  - Conjunto de entrenamiento sobre el que vamos a entrenar nuestro algoritmo para construir un modelo.
  - Conjunto de pruebas en el que probaremos nuestro modelo para ver qué tan precisas son sus predicciones.

Antes de crear los dos conjuntos, necesitamos identificar el algoritmo que usaremos para nuestro modelo. Podemos usar el mapa como una hoja de trucos para seleccionar los algoritmos que podemos probar para construir nuestro modelo de predicción. Usando la lista de verificación, veamos en qué categoría se encuentra nuestro conjunto de datos actual:

- Disponemos de 506 muestras: >50? (**Sí**)
- ¿Estamos prediciendo una categoría? (**No**)
- ¿Estamos prediciendo una cantidad? (**Sí**)

Con base en la lista de verificación que preparamos anteriormente y siguiendo el `machine_learning_map`, podemos probar **métodos de regresión** como:

- Linear Regression
- Lasso
- ElasticNet Regression

- Ridge Regression:
- K Neighbors Regressor
- Decision Tree Regressor
- Simple Vector Regression (SVR)
- Ada Boost Regressor
- Gradient Boosting Regressor
- Random Forest Regression
- Extra Trees Regressor

Consulte los siguientes documentos sobre regresión: Supervised learning--scikit-learn, Learn regression algorithms using Python and scikit-learn, Non-Linear Regression Trees with scikit-learn.

Simple Linear Model

- Si es difícil visualizar las múltiples funciones.
- Queremos predecir el precio de la vivienda con una sola variable y luego pasar a la regresión con todas las características.
- Debido a que **RM** muestra una correlación positiva con los **Precios de la vivienda**, usaremos **RM** para el modelo.

In [ ]:
```
import numpy as np
```

In [ ]:
```
X_rooms = bos_pd.RM
y_price = bos_pd.PRICE
X_rooms.shape


X_rooms = np.array(X_rooms).reshape(-1,1)
y_price = np.array(y_price).reshape(-1,1)

print(X_rooms.shape)
print(y_price.shape)
```

```
(506, 1)
(506, 1)
```

Splitting the data into training and testing sets

- Dividimos los datos en conjuntos de entrenamiento y prueba.
- Entrenamos el modelo con el 80% de las muestras y probamos con el 20% restante.
- Hacemos esto para evaluar el rendimiento del modelo en datos invisibles.

In [ ]:
```
from sklearn.model_selection import train_test_split
```

In [ ]:
```
X_train_1, X_test_1, Y_train_1, Y_test_1 =   train_test_split(X_rooms, y_price, test_siz

print(X_train_1.shape)
print(Y_train_1.shape)
print(X_test_1.shape)
print(Y_test_1.shape)
```

```
(404, 1)
(404, 1)
(102, 1)
(102, 1)
```

Training and testing the model

- Usamos LinearRegression de scikit-learn para entrenar nuestro modelo tanto en el entrenamiento como en los conjuntos de prueba.
- Comprobamos el rendimiento del modelo en el conjunto de datos del tren.

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

In [ ]:
```
reg_1 = LinearRegression()
reg_1.fit(X_train_1, Y_train_1)

y_train_predict_1 = reg_1.predict(X_train_1)
rmse = (np.sqrt(metrics.mean_squared_error(Y_train_1, y_train_predict_1)))
r2 = round(reg_1.score(X_train_1, Y_train_1),2)

print(f"The model performance for training set")
print(f"--------------------------------------")
print(f'RMSE is {rmse}')
print(f'R2 score is {r2}')
```

```
The model performance for training set
--------------------------------------
RMSE is 6.972277149440585
R2 score is 0.43
```

Model Evaluation for Test Set

In [ ]:
```
y_pred_1 = reg_1.predict(X_test_1)
rmse = (np.sqrt(metrics.mean_squared_error(Y_test_1, y_pred_1)))
r2 = round(reg_1.score(X_test_1, Y_test_1),2)

print(f"The model performance for training set")
print(f"--------------------------------------")
print(f"Root Mean Squared Error: {rmse}")
print(f"R^2: {r2}")
```

```
The model performance for training set
--------------------------------------
Root Mean Squared Error: 4.895963186952216
R^2: 0.69
```

The coefficient of determination: 1 is perfect prediction

In [ ]:
```
print(f'Coefficient of determination: {metrics.r2_score(Y_test_1, y_pred_1) :.4f}')
```

```
Coefficient of determination: 0.6938
```

45-Degree Plot

In [ ]:
```
plt.figure(figsize=(8, 5));
plt.scatter(Y_test_1, y_pred_1);
plt.plot([0, 50], [0, 50], '--k');
plt.axis('tight');
plt.xlabel("Actual House Prices ($1000)");
plt.ylabel("Predicted House Prices: ($1000)");
#plt.xticks(range(0, int(max(y_test)),2));
#plt.yticks(range(0, int(max(y_test)),2));
plt.title("Actual Prices vs Predicted prices");
plt.tight_layout();
```

Modelo de regresión lineal con todas las variables

- Queremos crear un modelo considerando todas las características del conjunto de datos.

Creacion del modelo

In [ ]:
```
X = bos_pd.drop('PRICE', axis = 1)
y = bos_pd['PRICE']
```

In [ ]:
```
X
```

Out[ ]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33
...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88

506 rows × 13 columns

In [ ]:
```
y
```

Out[ ]:
```
0      24.0
1      21.6
2      34.7
3      33.4
4      36.2
       ...
501    22.4
502    20.6
503    23.9
504    22.0
505    11.9
Name: PRICE, Length: 506, dtype: float64
```

- Utilice `train_test_split` para dividir los datos en subconjuntos aleatorios de tren y prueba.
- Cada vez que lo ejecutes sin especificar `random_state`, obtendrás un resultado diferente.
- Si usa `random_state=some_number`, puede garantizar que la división será siempre la misma.
- No importa cuál sea el valor de `random_state`: 42, 0, 21, ...
- Esto es útil si desea resultados reproducibles.

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=42)
```

El modelo de regresion lineal:

In [ ]:
```
reg_all = LinearRegression()
reg_all.fit(X_train, y_train)
```

Out[ ]:
```
▾ LinearRegression
LinearRegression()
```

Evaluación del modelo para el conjunto de entrenamiento

In [ ]:
```
y_train_predict = reg_all.predict(X_train)
```

In [ ]:
```
rmse = (np.sqrt(metrics.mean_squared_error(y_train, y_train_predict)))
r2 = round(reg_all.score(X_train, y_train),2)

print(f"The model performance for training set")
print(f"--------------------------------------")
print(f'RMSE is {rmse}')
print(f'R2 score is {r2}')
```

```
The model performance for training set
--------------------------------------
RMSE is 4.6520331848801675
R2 score is 0.75
```

Evaluación del modelo para el conjunto de pruebas

In [ ]:
```
y_pred = reg_all.predict(X_test)
```

In [ ]:
```
rmse = (np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
r2 = round(reg_all.score(X_test, y_test),2)
```

```
print(f"The model performance for training set")
print(f"--------------------------------------")
print(f"Root Mean Squared Error: {rmse}")
print(f"R^2: {r2}")
```

```
The model performance for training set
--------------------------------------
Root Mean Squared Error: 4.928602182665332
R^2: 0.67
```

El coeficiente de determinación: 1 es predicción perfecta.

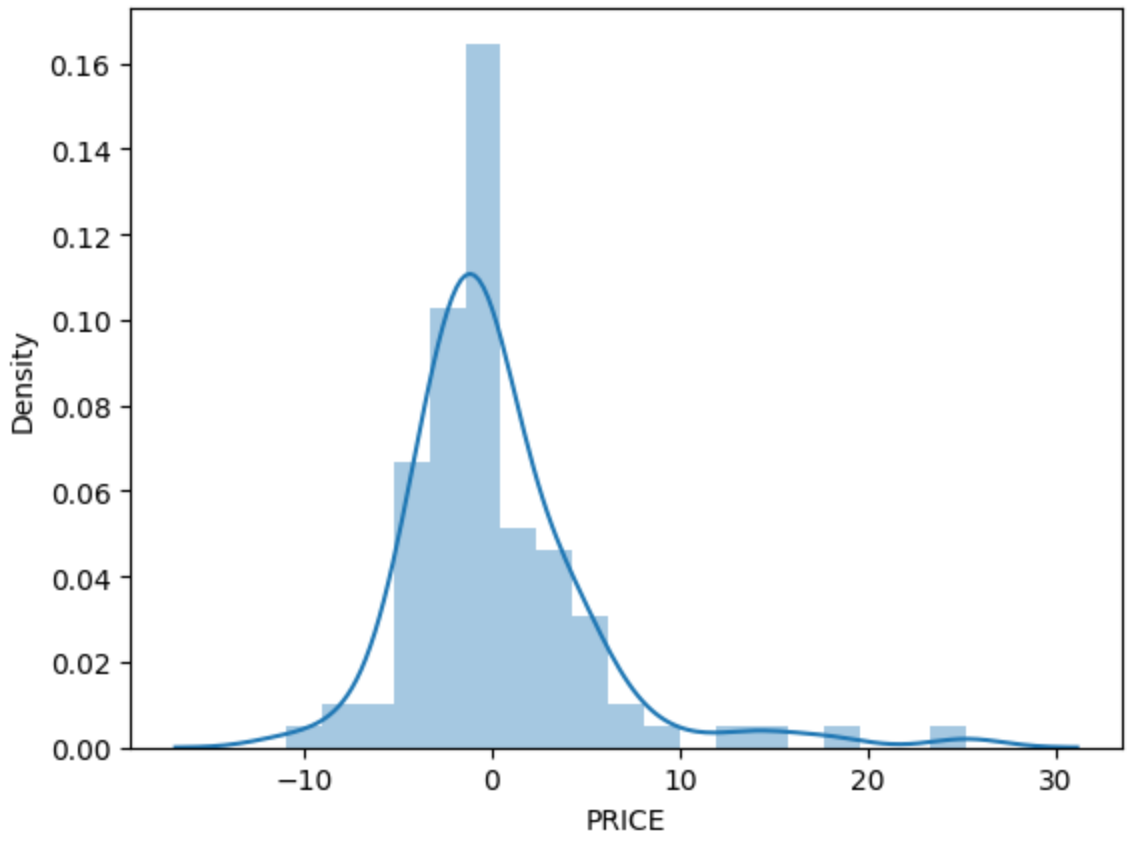
In [ ]:
```
print(f'Coefficient of determination: {metrics.r2_score(y_test, y_pred) :.4f}')
```

```
Coefficient of determination: 0.6688
```

Error Distribution

In [ ]:
```
sns.distplot(y_test - y_pred);
```

```
<ipython-input-40-5219a9e270f7>:1: UserWarning:

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

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

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

  sns.distplot(y_test - y_pred);
```



45-Degree plot

In [ ]:
```
plt.figure(figsize=(8, 5));
```

```
plt.scatter(y_test, y_pred);
plt.plot([0, 50], [0, 50], '--k');
plt.axis('tight');
plt.xlabel("Actual House Prices ($1000)");
plt.ylabel("Predicted House Prices: ($1000)");
#plt.xticks(range(0, int(max(y_test)),2));
#plt.yticks(range(0, int(max(y_test)),2));
plt.title("Actual Prices vs Predicted prices");
plt.tight_layout();
```

In [ ]:
```
print("RMS: %r " % np.sqrt(np.mean((y_test - y_pred) ** 2)))
```

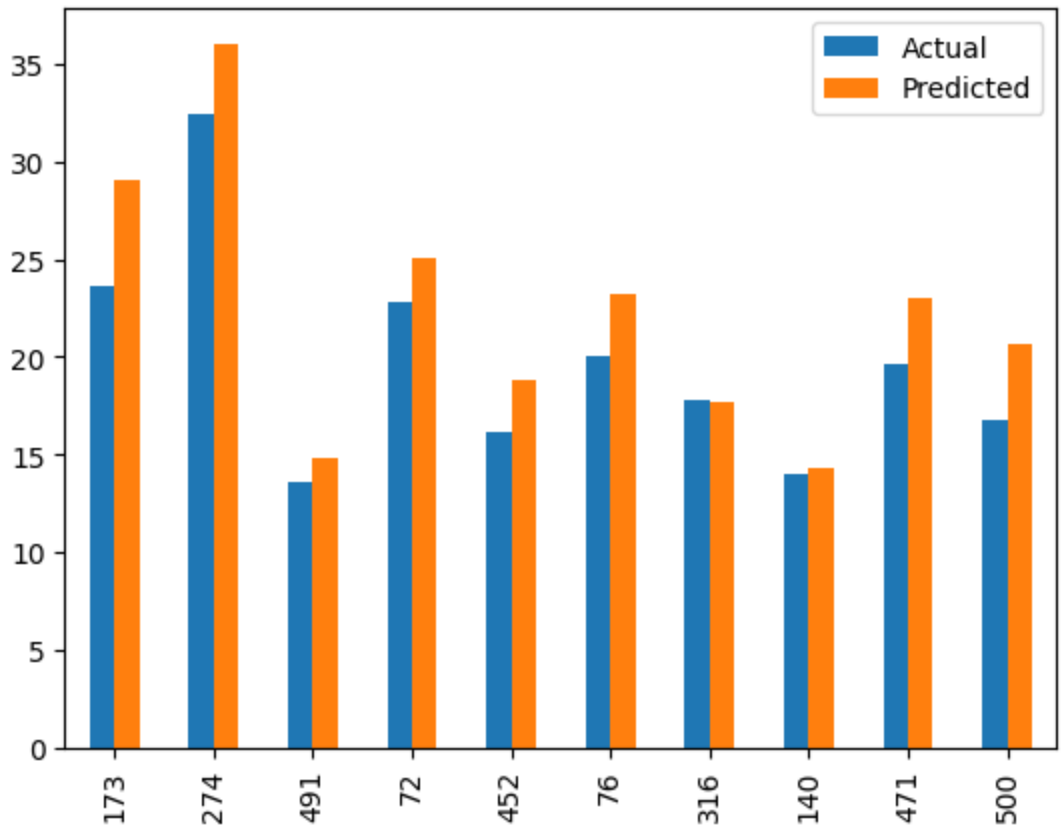
```
RMS: 4.928602182665332
```

In [ ]:
```
df1 = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df2 = df1.head(10)
df2
```

Out[ ]:

	Actual	Predicted
173	23.6	28.996724
274	32.4	36.025565
491	13.6	14.816944
72	22.8	25.031979
452	16.1	18.769880
76	20.0	23.254429
316	17.8	17.662538
140	14.0	14.341190
471	19.6	23.013207
500	16.8	20.632456

In [ ]:
```
df2.plot(kind='bar');
```



Eleccion del mejor modelo: k-Fold Cross-Validation

- La validación cruzada es un procedimiento de remuestreo que se utiliza para evaluar modelos de aprendizaje automático en una muestra de datos limitada.
- Se utiliza principalmente en el aprendizaje automático aplicado para estimar la habilidad de un modelo de aprendizaje automático en datos invisibles.
- Usamos una muestra limitada para estimar cómo se espera que funcione el modelo en general cuando se usa para hacer predicciones sobre datos no utilizados durante el entrenamiento del modelo.
- La mayor ventaja de este método es que cada punto de datos se utiliza para la validación exactamente una vez y para el entrenamiento k-1 veces.
- Para elegir el modelo final a utilizar, seleccionamos el que tiene el menor error de validación.\

El procedimiento general es el siguiente:

1. Mezcla el conjunto de datos aleatoriamente.
2. Dividir el conjunto de datos en k grupos
3. Para cada grupo único:\ 3.1 Tome el grupo como un conjunto de datos de prueba o de reserva\ 3.2 Tome los grupos k-1 restantes como un conjunto de datos de entrenamiento\ 3.3 Ajustar un modelo en el conjunto de entrenamiento y evaluarlo en el conjunto de prueba\ 3.4 Conservar la puntuación de la evaluación y descartar el modelo.
4. Resuma las habilidades del modelo utilizando la muestra de puntajes de evaluación del modelo.

¿Cómo elegir **k**?

- Un valor mal elegido para **k** puede dar lugar a una idea errónea de la habilidad del modelo, como una puntuación con una varianza alta o un sesgo alto.

- La elección de **k** suele ser 5 o 10, pero no existe una regla formal. A medida que **k** aumenta, la diferencia de tamaño entre el conjunto de entrenamiento y los subconjuntos de remuestreo se reduce. A medida que esta diferencia disminuye, el sesgo de la técnica se vuelve menor.
- Un valor de **k=10** es muy común en el campo del aprendizaje automático aplicado y se recomienda si tiene dificultades para elegir un valor para su conjunto de datos.

A continuación se muestra la visualización de una validación k veces cuando k = 5.

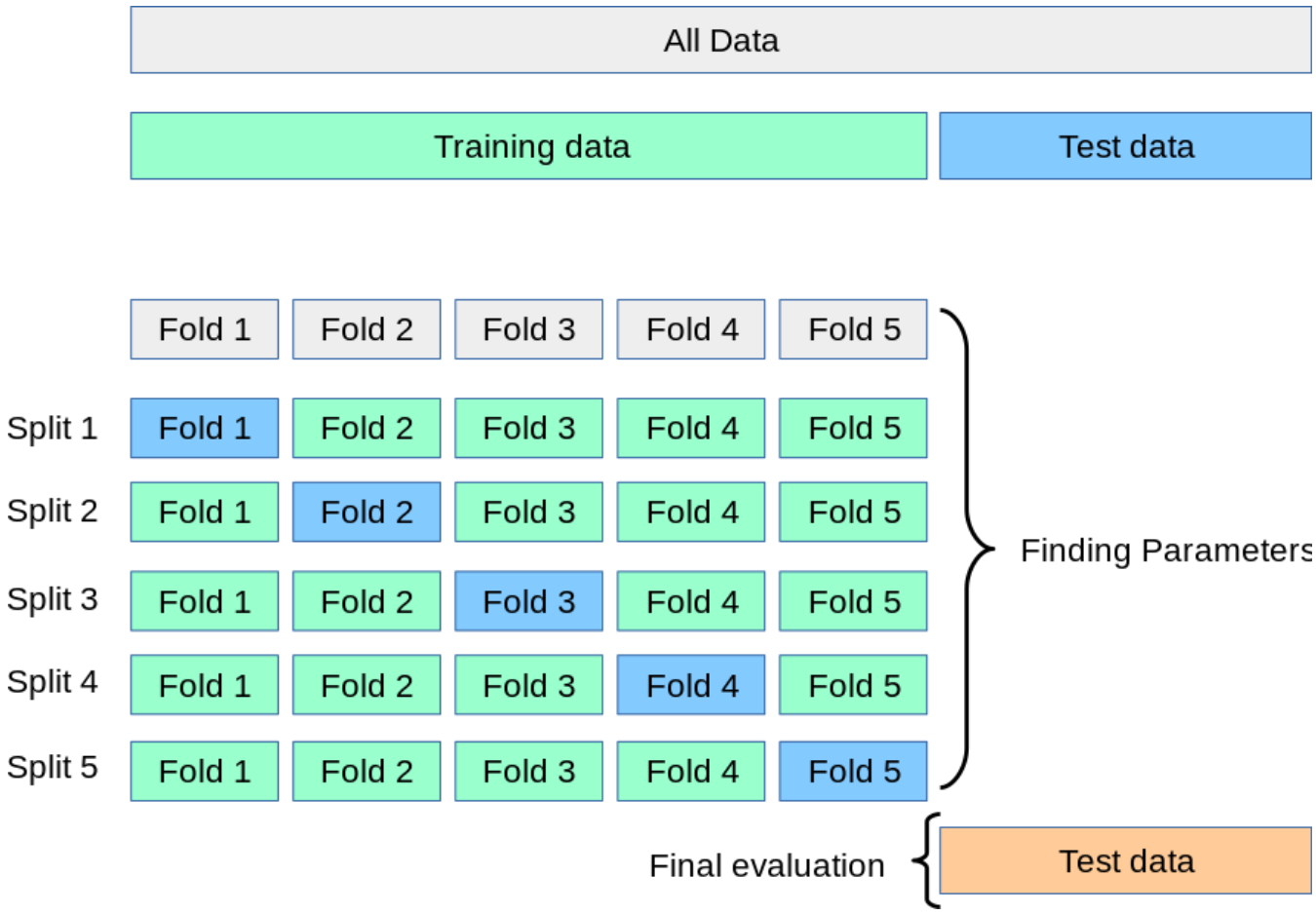


Image Source: https://scikit-learn.org/

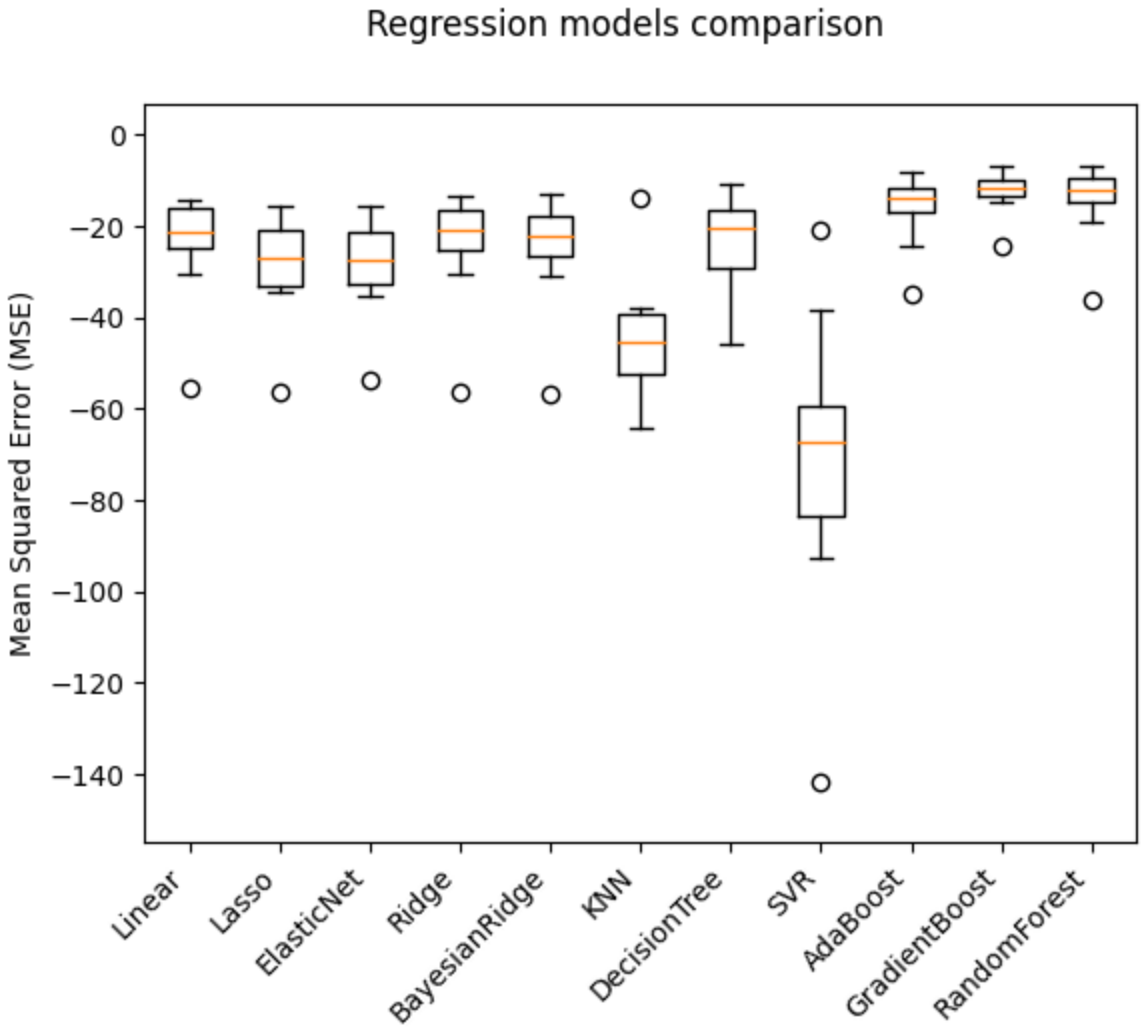
In [ ]:
```
np.sqrt(24.07)
```

Out[ ]:
```
4.906118628814432
```

In [ ]:
```
# import warnings filter
from warnings import simplefilter
# ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import Ridge
from sklearn.linear_model import BayesianRidge
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
```

```
from sklearn.ensemble import RandomForestRegressor

# variables de usuario para ajustar
seed    = 9
folds   = 10
metric  = "neg_mean_squared_error"

# mantener diferentes modelos de regresión en un solo diccionario
models = dict()
models["Linear"]        = LinearRegression()
models["Lasso"]         = Lasso()
models["ElasticNet"]    = ElasticNet()
models["Ridge"]         = Ridge()
models["BayesianRidge"] = BayesianRidge()
models["KNN"]           = KNeighborsRegressor()
models["DecisionTree"]  = DecisionTreeRegressor()
models["SVR"]           = SVR()
models["AdaBoost"]      = AdaBoostRegressor()
models["GradientBoost"] = GradientBoostingRegressor()
models["RandomForest"]  = RandomForestRegressor()

# Validación cruzada 10 veces para cada modelo
model_results = list()
model_names   = list()
for model_name in models:
    model   = models[model_name]
    k_fold  = KFold(n_splits=folds, random_state=seed,shuffle=True)
    results = cross_val_score(model, X_train, y_train, cv=k_fold, scoring=metric)
    #print(results)
    model_results.append(results)
    model_names.append(model_name)
    print("{:>20}: {:.2f}, {:.2f}".format(model_name, round(results.mean(), 3),
                                round(results.std(), 3)))

# box-whisker plot to compare regression models
import matplotlib.pyplot as plt
figure = plt.figure();
figure.suptitle('Regression models comparison');
ax = figure.add_subplot(111);
plt.boxplot(model_results);
ax.set_xticklabels(model_names, rotation = 45, ha="right");
ax.set_ylabel("Mean Squared Error (MSE)");
plt.margins(0.05, 0.1);
#plt.savefig("model_mse_scores.png")
plt.show();
#plt.clf()
#plt.close()
```

```
              Linear: -24.07, 11.53
               Lasso: -28.88, 10.90
          ElasticNet: -28.89, 10.17
               Ridge: -24.23, 11.80
       BayesianRidge: -24.88, 11.69
                 KNN: -44.99, 13.80
        DecisionTree: -23.17, 9.98
                 SVR: -71.30, 31.05
            AdaBoost: -16.15, 7.78
       GradientBoost: -12.29, 4.78
        RandomForest: -14.12, 8.09
```



Según la comparación anterior, podemos ver que el modelo "Regresión de aumento de gradiente" supera a todos los demás modelos de regresión.

Model with Gradient Boosted Tree

In [ ]:
```
gbr = GradientBoostingRegressor()
gbr.fit(X_train, y_train)

gbr_predicted = gbr.predict(X_test)
gbr_expected = y_test
```

Root Mean Square Error:

In [ ]:
```
print("RMS: %r " % np.sqrt(np.mean((gbr_predicted - gbr_expected) ** 2)))
```

```
RMS: 2.495003006968167
```

The coefficient of determination: (1 is perfect prediction)

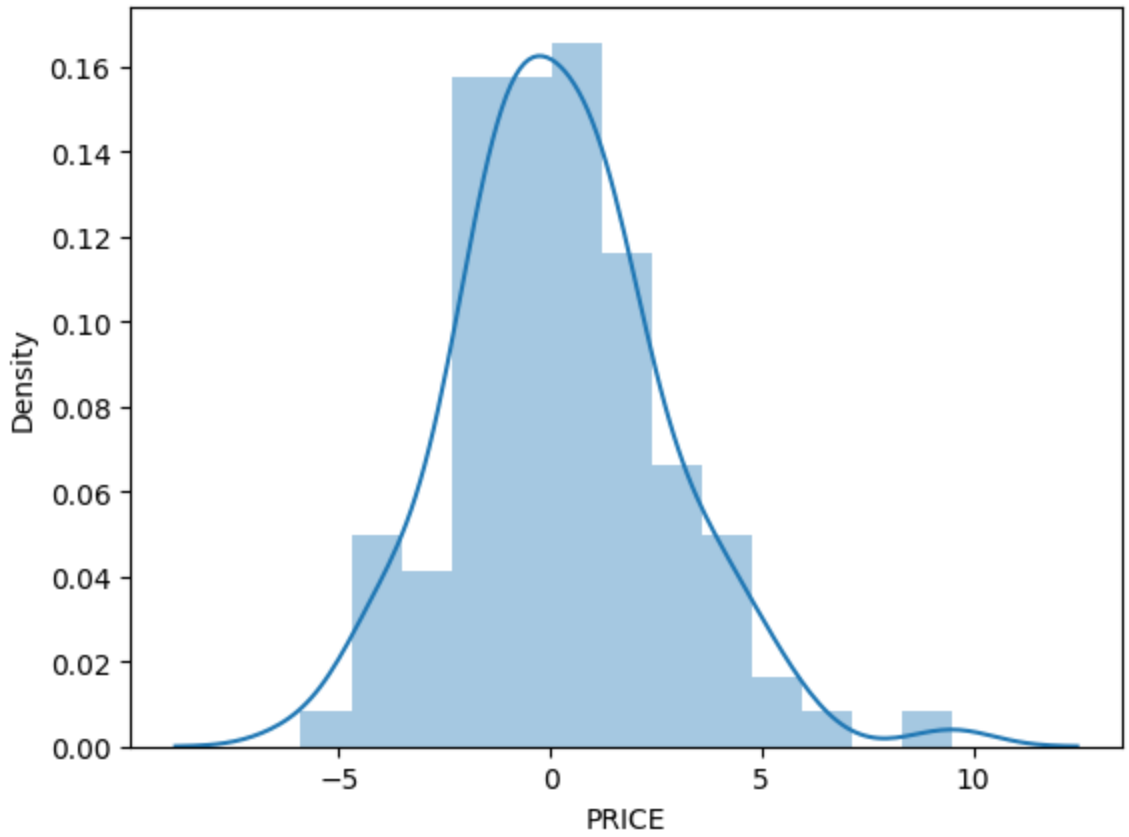
In [ ]:
```
print('Coeff of determination: {:.4f}'.format(metrics.r2_score(gbr_expected, gbr_predict
```

```
Coeff of determination: 0.9151
```

Error Distribution

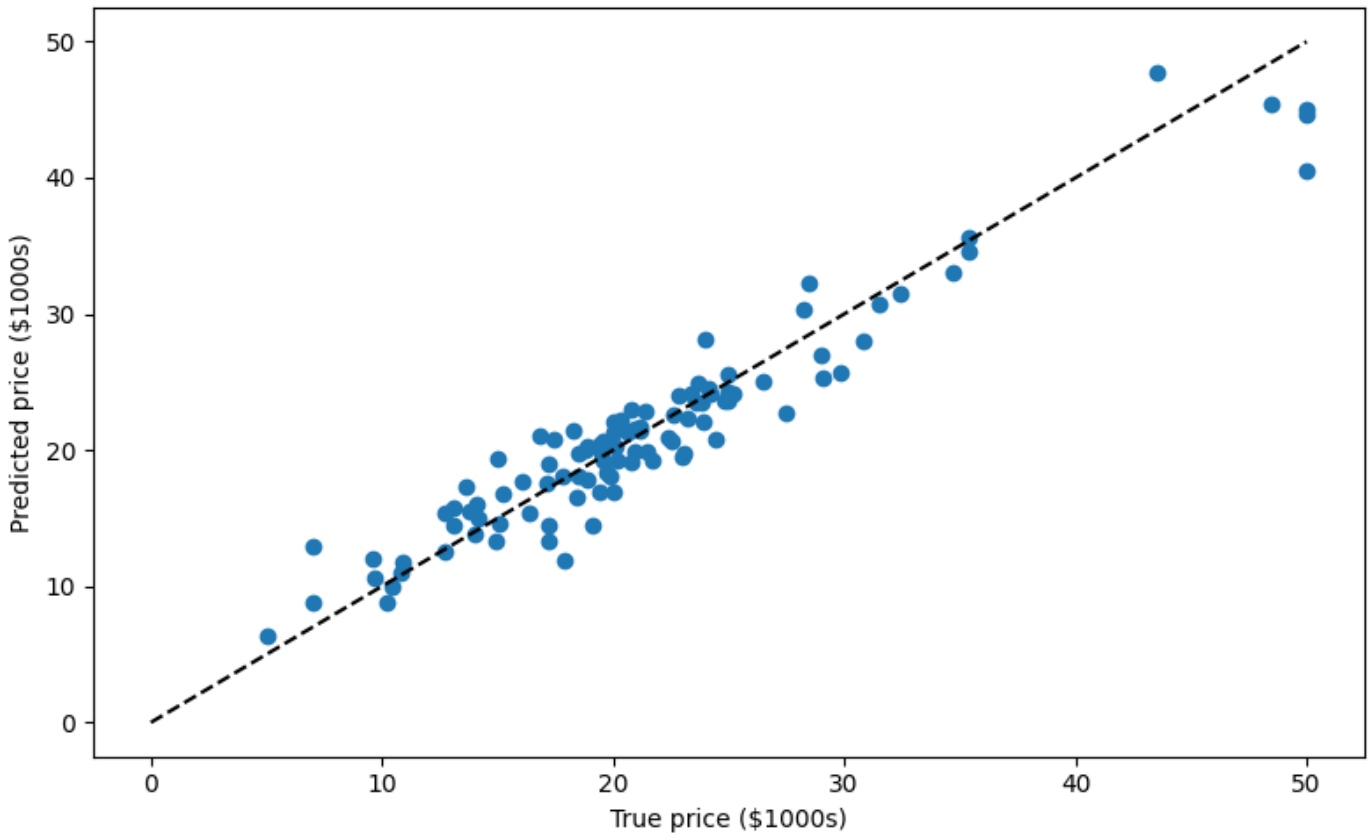
In [ ]:
```
sns.distplot(gbr_expected - gbr_predicted);
```

```
<ipython-input-52-c11227365f33>:1: UserWarning:

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

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

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

  sns.distplot(gbr_expected - gbr_predicted);
```



45-Degree Plot

In [ ]:
```
plt.figure(figsize=(8, 5));
plt.scatter(gbr_expected, gbr_predicted)
plt.plot([0, 50], [0, 50], '--k');
plt.axis('tight');
plt.xlabel('True price ($1000s)');
plt.ylabel('Predicted price ($1000s)');
plt.tight_layout();
```



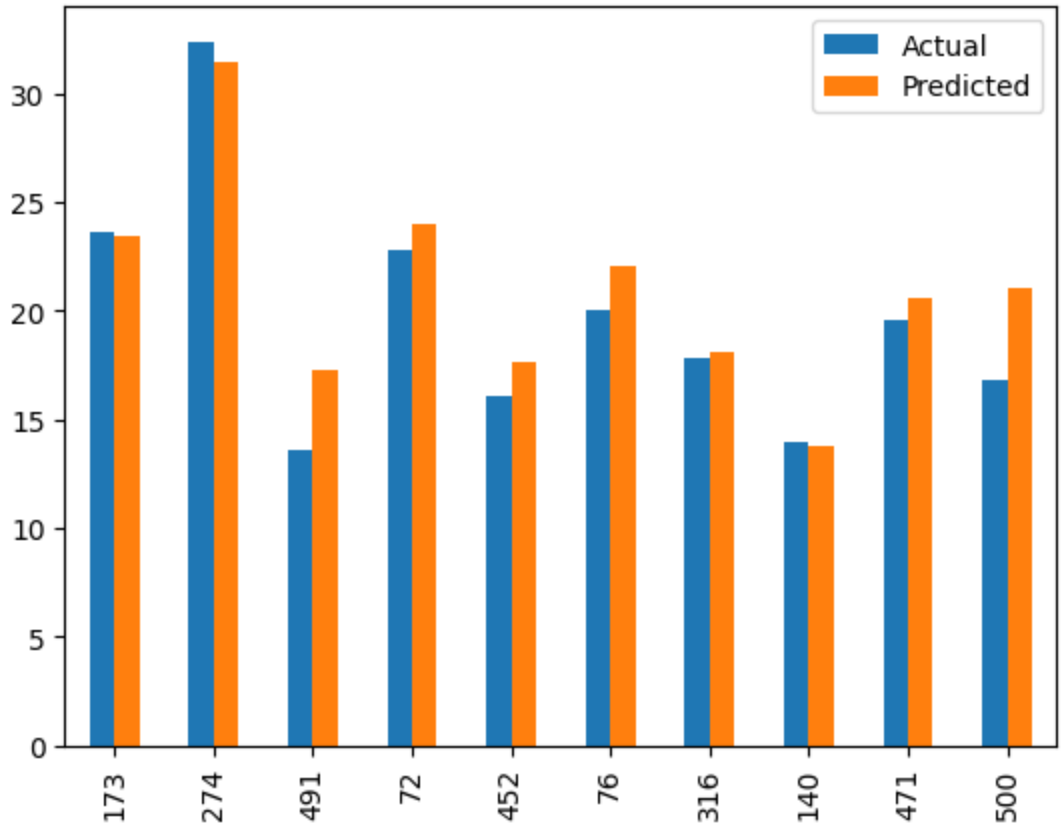
Zoom in:

In [ ]:
```
df1 = pd.DataFrame({'Actual': gbr_expected, 'Predicted': gbr_predicted})
df2 = df1.head(10)
df2
```

Out[ ]:

	Actual	Predicted
173	23.6	23.449761
274	32.4	31.461360
491	13.6	17.269762
72	22.8	23.974831
452	16.1	17.681144
76	20.0	22.031128
316	17.8	18.149588
140	14.0	13.830452
471	19.6	20.616193
500	16.8	21.042857

In [ ]:
```
df2.plot(kind='bar');
```



Fin del Cuaderno