TOULOUSE LAUTREC

APRENDIZAJE AUTOMATICO CON PYTHON

MACHINE LEARNING

Ing. Alexander Valdez

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

Primera Clase

---

1 ¿Que es Aprendizaje Automatico?

Antes de echar un vistazo a los detalles de varios métodos de aprendizaje automático, comencemos por ver qué es y qué no es el aprendizaje automático. El aprendizaje automático a menudo se clasifica como un subcampo de la inteligencia artificial, pero encuentro que la categorización puede ser engañosa. El estudio del aprendizaje automático ciertamente surgió de la investigación en este contexto, pero en la aplicación de métodos de aprendizaje automático a la ciencia de datos, es más útil pensar en el aprendizaje automático como un medio para construir modelos de datos.

En este contexto, el "aprendizaje" entra en juego cuando damos a estos modelos parámetros ajustables que pueden adaptarse a los datos observados; de esta manera se puede considerar que el programa está "aprendiendo" de los datos. Una vez que estos modelos se han ajustado a los datos vistos anteriormente, se pueden utilizar para predecir y comprender aspectos de los datos recién observados. Dejaré al lector la digresión más filosófica sobre hasta qué punto este tipo de "aprendizaje" matemático basado en modelos es similar al "aprendizaje" exhibido por el cerebro humano.

Comprender la configuración de problemas en el aprendizaje automático es esencial para utilizar estas herramientas de manera efectiva, por lo que comenzaremos con algunas categorizaciones amplias de los tipos de enfoques que discutiremos aquí.

2 Categorias de Aprendizaje Automatico

El aprendizaje automático se puede clasificar en dos tipos principales: aprendizaje supervisado y aprendizaje no supervisado.

El aprendizaje supervisado implica de alguna manera modelar la relación entre las características medidas de los datos y algunas etiquetas asociadas con los datos; Una vez que se determina este modelo, se puede utilizar para aplicar etiquetas a datos nuevos y desconocidos. A veces, esto se subdivide en tareas de clasificación y tareas de regresión: en la clasificación, las etiquetas son categorías discretas, mientras que en la regresión, las etiquetas son cantidades continuas. Verá ejemplos de ambos tipos de aprendizaje supervisado en la siguiente sección.

El aprendizaje no supervisado implica modelar las características de un conjunto de datos sin referencia a ninguna etiqueta. Estos modelos incluyen tareas como agrupación y reducción de dimensionalidad. Los algoritmos de agrupamiento identifican distintos grupos de datos, mientras que los algoritmos de reducción de dimensionalidad buscan representaciones más concisas de los datos. También verá ejemplos de ambos tipos de aprendizaje no supervisado en la siguiente sección.

Además, existen los llamados métodos de aprendizaje semisupervisados, que se sitúan a medio camino entre el aprendizaje supervisado y el no supervisado. Los métodos de aprendizaje semisupervisados suelen ser útiles cuando solo se dispone de etiquetas incompletas

3. Scikit-Learn

- Scikit-learn es una biblioteca gratuita de aprendizaje automático para Python.
- Proporciona una selección de herramientas eficientes para el aprendizaje automático y el modelado estadístico, que incluyen::
    - **Classification:** Identificar a qué categoría pertenece un objeto. Ejemplo: detección de spam
    - **Regression:** Predicción de una variable continua basada en variables independientes relevantes. Ejemplo: predicciones del precio de las acciones
    - **Clustering:** Agrupación automática de objetos similares en diferentes grupos. Ejemplo: segmentación de clientes
    - **Dimensionality Reduction:** Busca reducir el número de variables de entrada en los datos de entrenamiento preservando las relaciones destacadas en los datos
- Presenta varios algoritmos como máquina de vectores de soporte, bosques aleatorios y k-vecinos.
- Admite bibliotecas científicas y numéricas de Python como NumPy y SciPy.

Algunos grupos populares de modelos proporcionados por scikit-learn incluyen:

- **Clustering:** Agrupa datos sin etiquetar, como KMeans.
- **Cross Validation:** Estime el rendimiento de modelos supervisados con datos no vistos.
- **Datasets:** para conjuntos de datos de prueba y para generar conjuntos de datos con propiedades específicas para investigar el comportamiento del modelo.
- **Dimensionality Reduction:** reduzca la cantidad de atributos en los datos para el resumen, la visualización y la selección de características, como el análisis de componentes principales.
- **Ensemble Methods:** combine las predicciones de múltiples modelos supervisados.
- **Feature Extraction:** defina atributos en datos de imagen y texto.
- **Feature Selection:** Identifique atributos significativos a partir de los cuales crear modelos supervisados.
- **Parameter Tuning:** Aprovecha al máximo los modelos supervisados..

- **Manifold Learning:** Resumir y representar datos multidimensionales complejos.
- **Supervised Models:** Una amplia gama que no se limita a modelos lineales generalizados, análisis discriminante, bayes ingenuos(naive bayes), métodos perezosos, redes neuronales, máquinas de vectores de soporte y árboles de decisión.
- **Unsupervised Learning Algorithms:** Incluyen clustering, análisis factorial, PCA (Análisis de componentes principales), redes neuronales no supervisadas.

Image Source: ulhpc-tutorials.readthedocs.io

4 Paquetes Requeridos

- Numpy
- scipy
- matplotlib
- pandas
- seaborn
- scikit-learn

5 PRIMEROS PASOS

Numerical Data

California Dataset

- the dataset contains 20,640 samples and 8 features;
- all features are numerical features encoded as floating number;
- there is no missing values.

Obtain the Dataset

In [ ]:
```
from sklearn.datasets import fetch_california_housing
import pandas as pd
housing = fetch_california_housing(as_frame=True)
```

In [ ]:
```
housing.frame.head()
```

Out[ ]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

In [ ]:
```
print(housing.DESCR)
```

```
.. _california_housing_dataset:

California Housing dataset
--------------------------

**Data Set Characteristics:**

    :Number of Instances: 20640

    :Number of Attributes: 8 numeric, predictive attributes and the target

    :Attribute Information:
        - MedInc        median income in block group
        - HouseAge      median house age in block group
        - AveRooms      average number of rooms per household
        - AveBedrms     average number of bedrooms per household
        - Population    block group population
        - AveOccup      average number of household members
        - Latitude      block group latitude
        - Longitude     block group longitude

    :Missing Attribute Values: None

This dataset was obtained from the StatLib repository.
https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html

The target variable is the median house value for California districts,
expressed in hundreds of thousands of dollars ($100,000).

This dataset was derived from the 1990 U.S. census, using one row per census
block group. A block group is the smallest geographical unit for which the U.S.
Census Bureau publishes sample data (a block group typically has a population
of 600 to 3,000 people).

A household is a group of people residing within a home. Since the average
number of rooms and bedrooms in this dataset are provided per household, these
columns may take surprisingly large values for block groups with few households
and many empty houses, such as vacation resorts.

It can be downloaded/loaded using the
:func:`sklearn.datasets.fetch_california_housing` function.

.. topic:: References

    - Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,
      Statistics and Probability Letters, 33 (1997) 291-297
```

Features of the Dataset

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

```
Keys:  dict_keys(['data', 'target', 'frame', 'target_names', 'feature_names', 'DESCR'])
```

In [ ]:
```
print("target_names: ", housing.target_names)
```

```
target_names:  ['MedHouseVal']
```

In [ ]:
```
print("Shape: ", housing.data.shape)
```

```
Shape:  (20640, 8)
```

In [ ]:
```
print("Feature Names: ", housing.feature_names)
```

```
Feature Names:  ['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccu
p', 'Latitude', 'Longitude']
```

Attribute Information:

Acronym	Description
MedInc	median income in block group
HouseAge	median house age in block group
AveRooms	average number of rooms per household
AveBedrms	average number of bedrooms per household
Population	block group population
AveOccup	average number of household members
Latitude	block group latitude
Longitude	block group longitude

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

Extract Data

Pass the data into a Pandas dataframe

In [ ]:
```
housing_pd = pd.DataFrame(housing.data)
housing_pd.head()
```

Out[ ]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25

In [ ]:
```
housing_pd.columns= housing.feature_names
housing_pd.head()
```

Out[ ]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25

Add home prices to the Pandas dataframe

In [ ]:
```
housing.target[:5]
```

Out[ ]:
```
0    4.526
1    3.585
2    3.521
```

```
3    3.413
4    3.422
Name: MedHouseVal, dtype: float64
```

In [ ]:
```
print("Shape of the target data: ", housing.target.shape)
```

```
Shape of the target data:  (20640,)
```

In [ ]:
```
housing.target_names
```

Out[ ]:
```
['MedHouseVal']
```

In [ ]:
```
housing_pd['PRICE']=housing.target
housing_pd.head()
```

Out[ ]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	PRICE
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

Check the types of features:

In [ ]:
```
housing_pd.dtypes
```

Out[ ]:
```
MedInc        float64
HouseAge      float64
AveRooms      float64
AveBedrms     float64
Population    float64
AveOccup      float64
Latitude      float64
Longitude     float64
PRICE         float64
dtype: object
```

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()
```

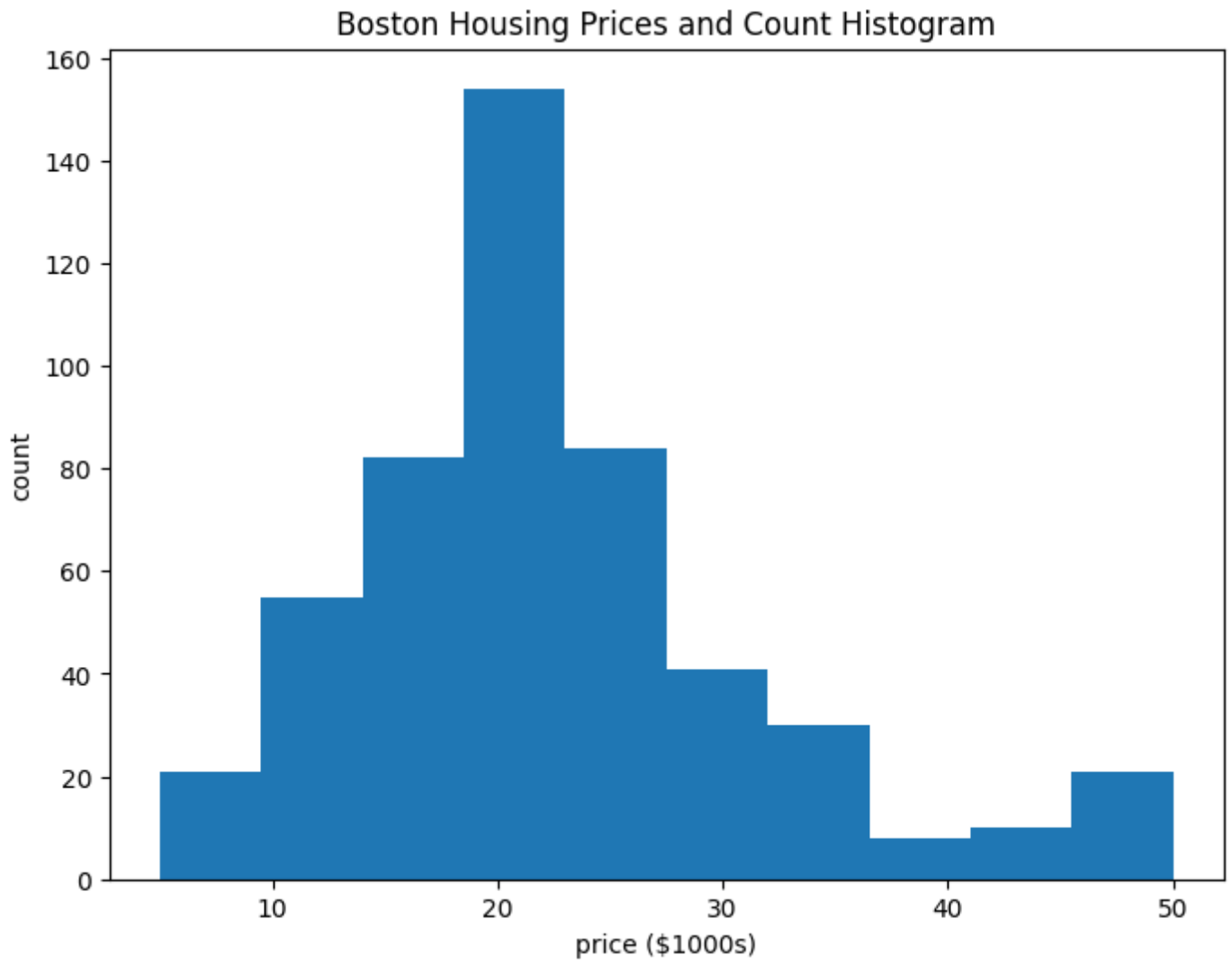
Out[ ]:

	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

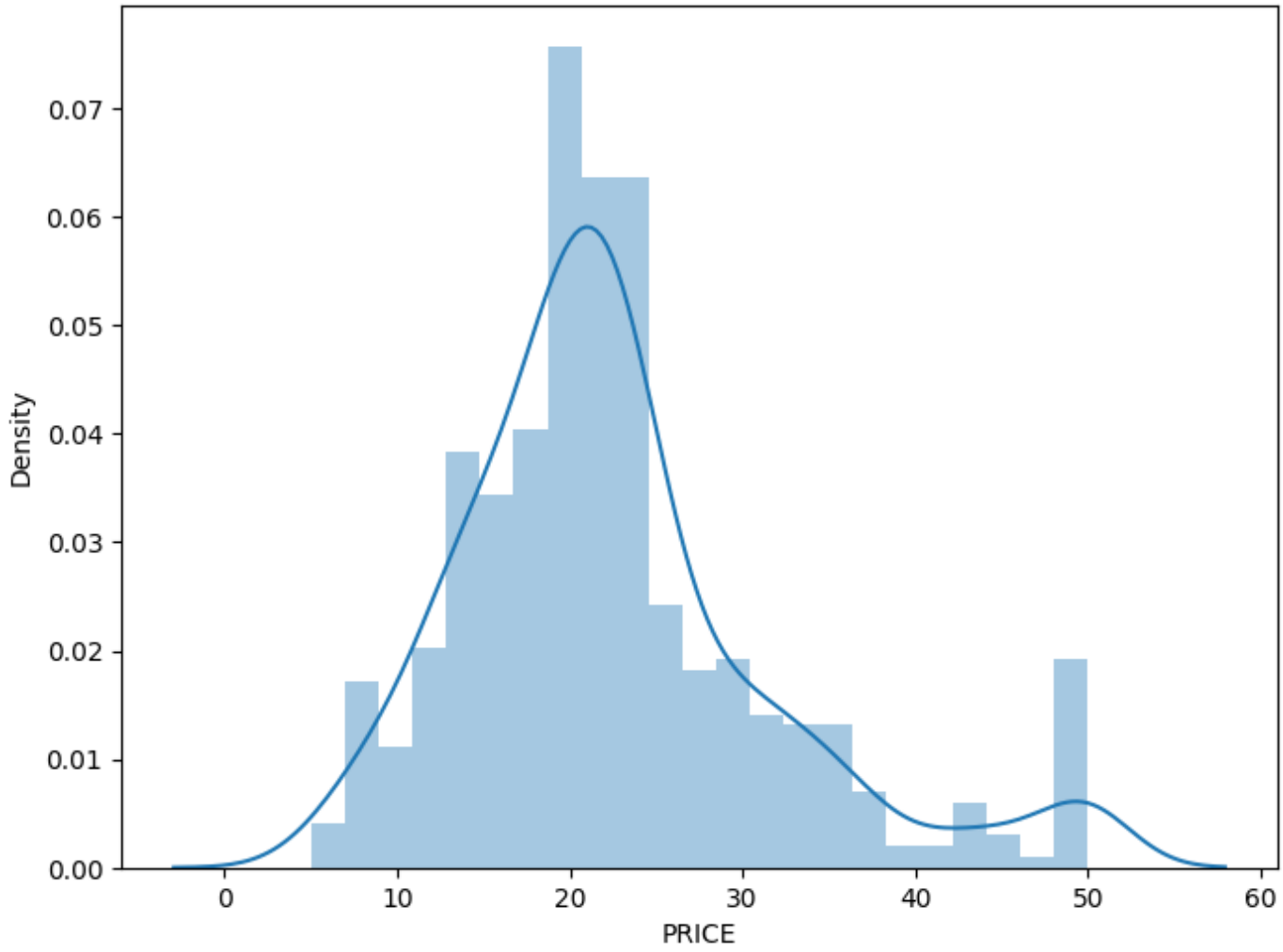
In [ ]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6));
plt.hist(bos_pd['PRICE']);
plt.title('Boston Housing Prices and Count Histogram');
plt.xlabel('price ($1000s)');
plt.ylabel('count');
plt.show();
```



In [ ]:
```
import seaborn as sns

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

```
<ipython-input-52-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']);
```

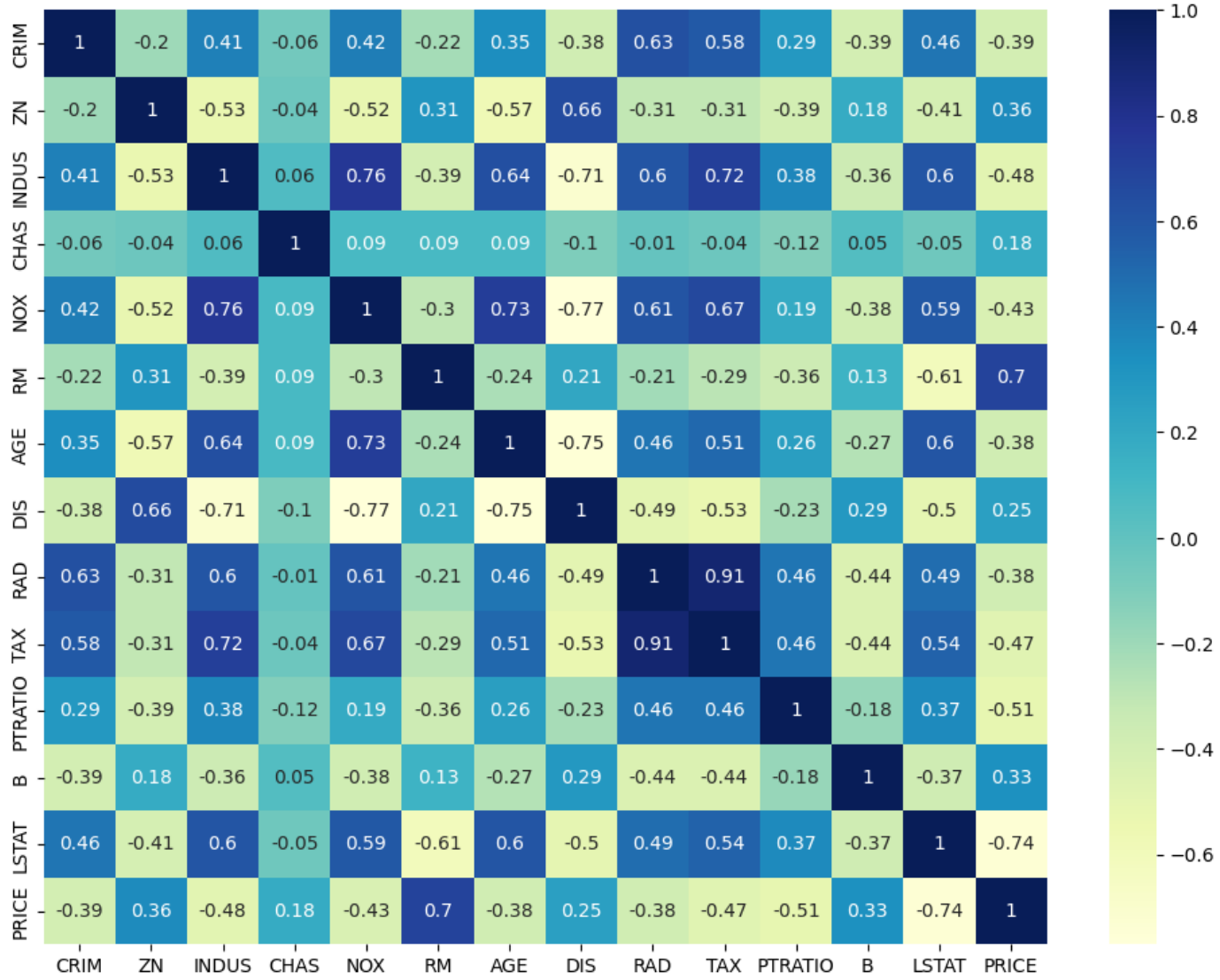


From the above output we can see that the values of PRICE is normally distributed with some of the outliers.

Heatmap: Two-Dimensional Graphical Representation

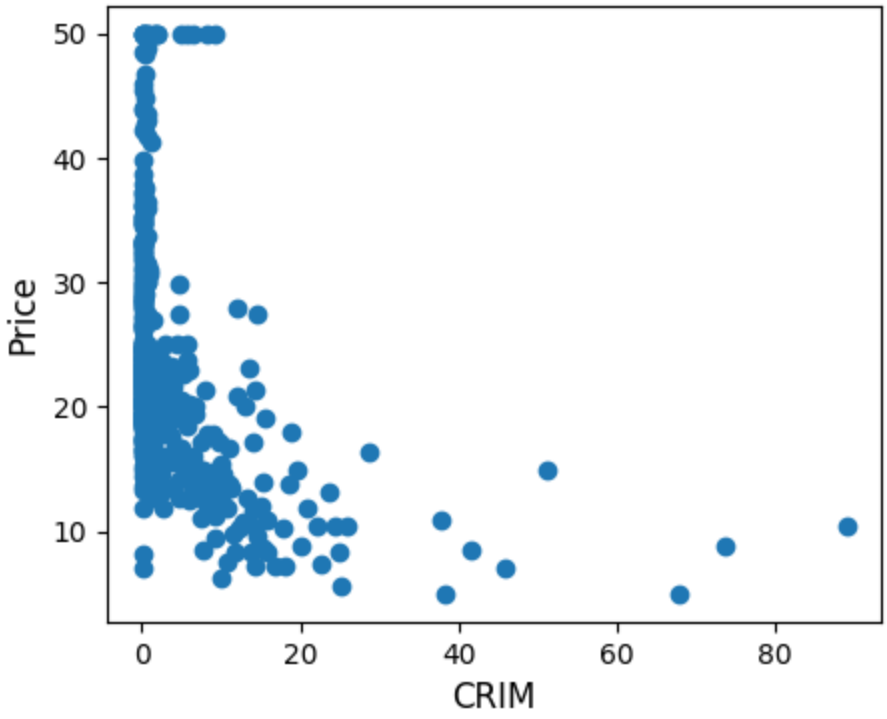
- Represent the individual values that are contained in a matrix as colors.
- Create a correlation matrix that measures the linear relationships between the variables.

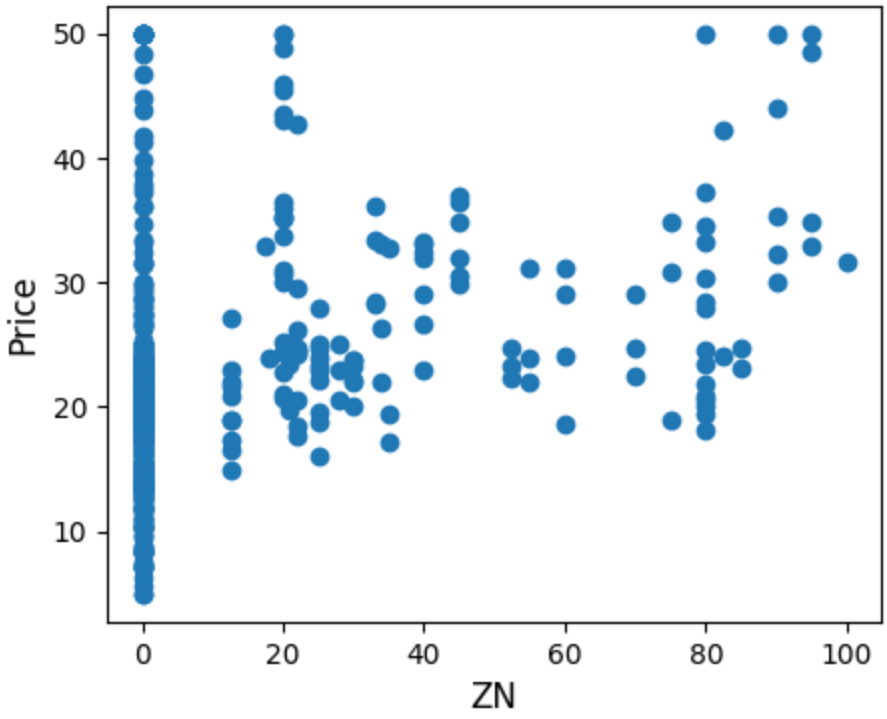
In [ ]:
```
plt.figure(figsize=(12, 9));
correlation_matrix = bos_pd.corr().round(2);
sns.heatmap(correlation_matrix, cmap="YlGnBu", annot=True);
```

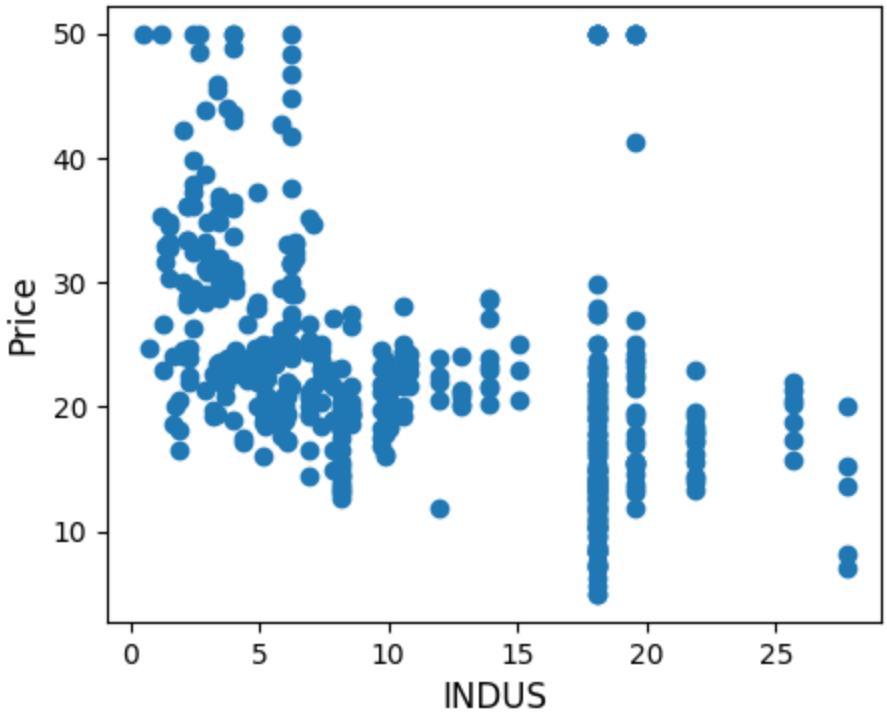


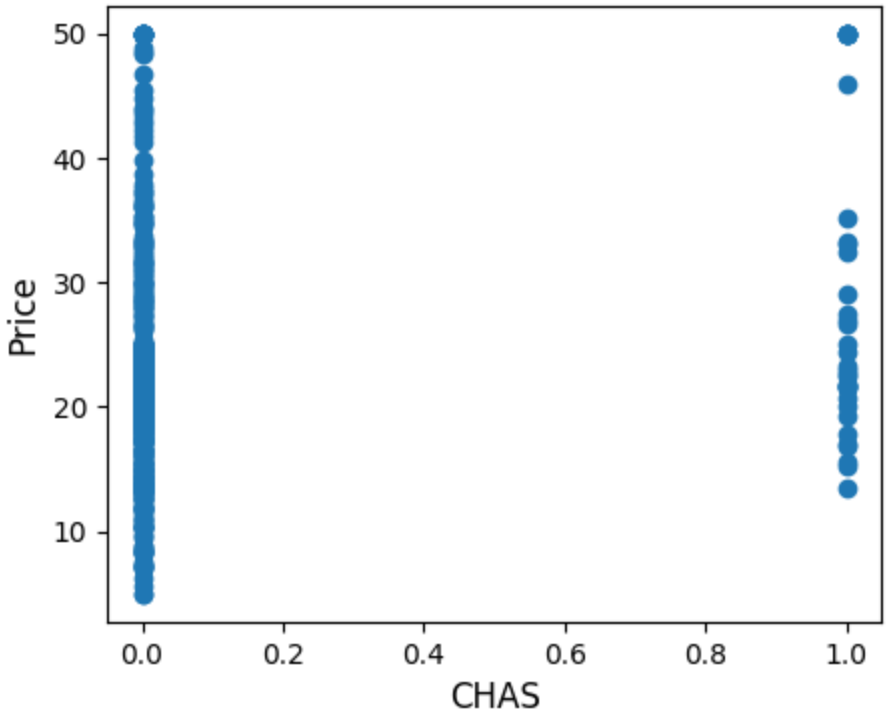
- **RM** tiene una fuerte correlación positiva con **PRICE** (0,7), mientras que **LSTAT tiene** una alta correlación negativa con **PRICE** (-0,74).
- Las características **RAD**, **TAX** tienen una correlación de 0,91. Estos pares de características están fuertemente correlacionados entre sí. Esto 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.

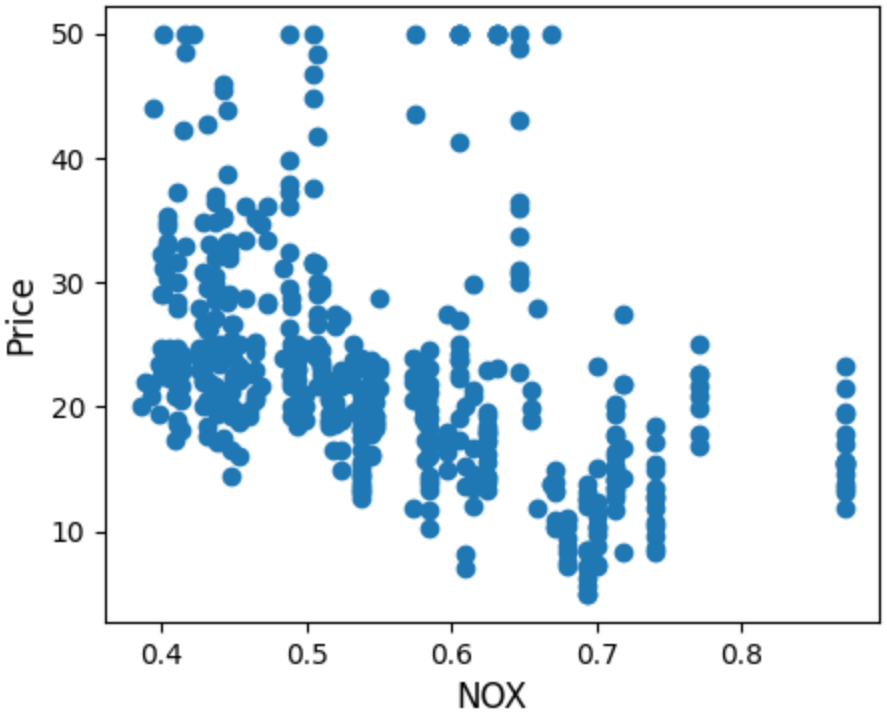
In [ ]:
```
#for feature_name in housing.feature_names:
for feature_name in bos_pd.columns:

    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();
```

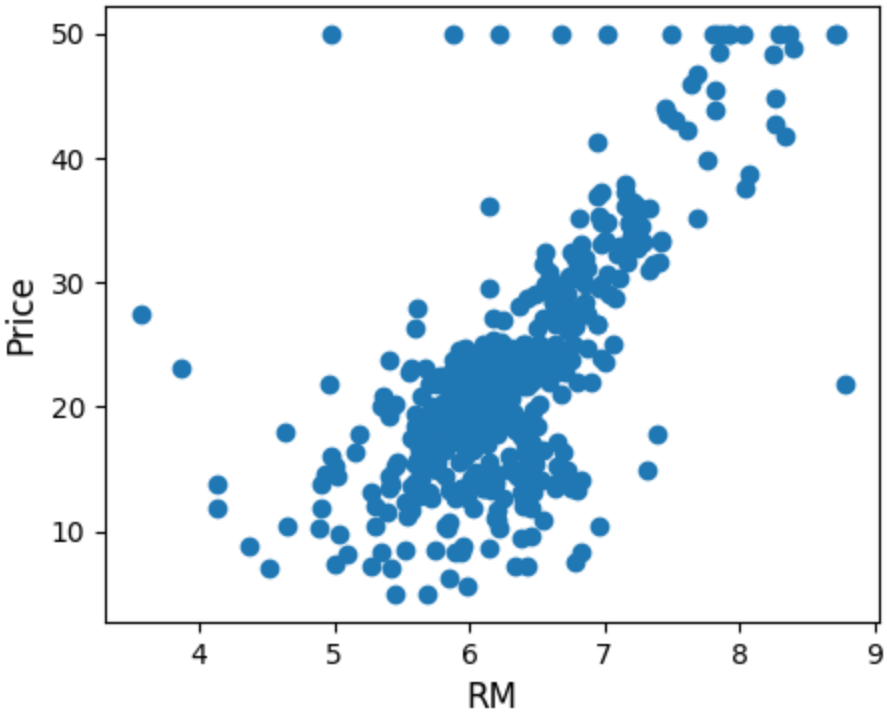

Price
CRIM
50
40
30
20
10
0
20
40
60
80

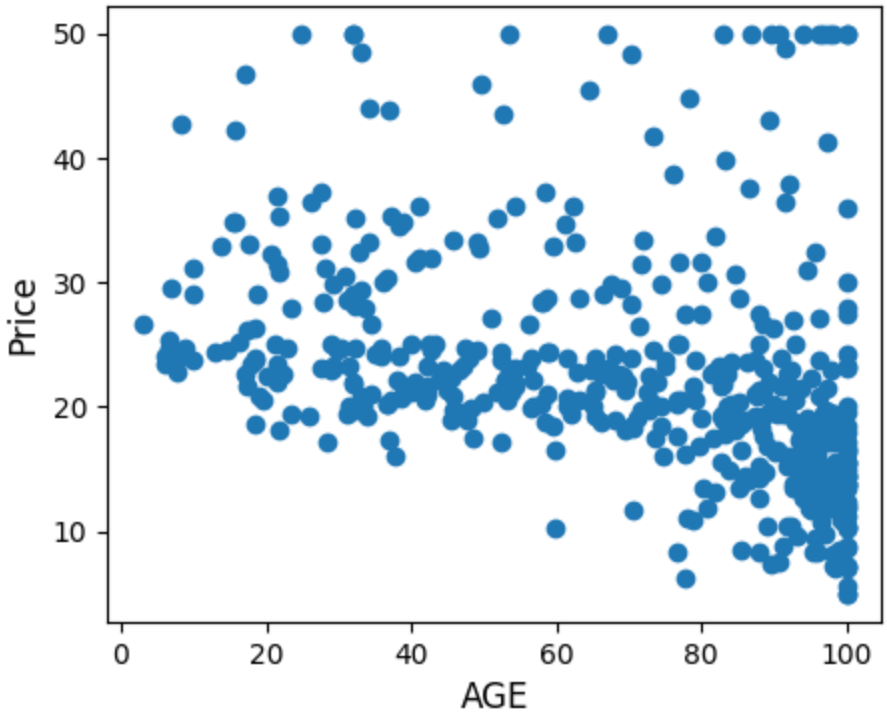

Price
ZN
50
40
30
20
10
0
20
40
60
80
100

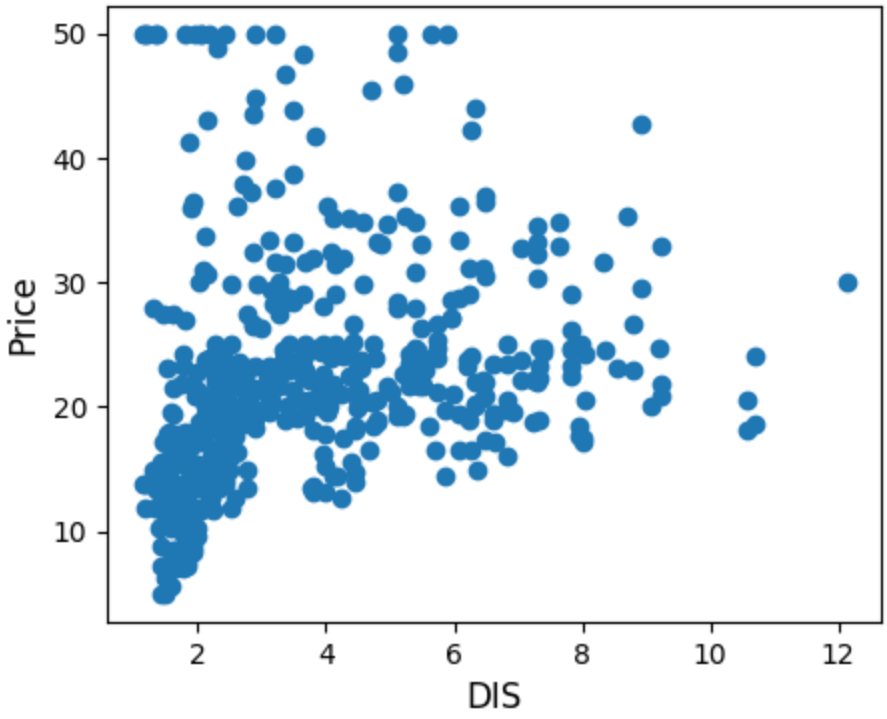

Price
INDUS
50
40
30
20
10
0
5
10
15
20
25

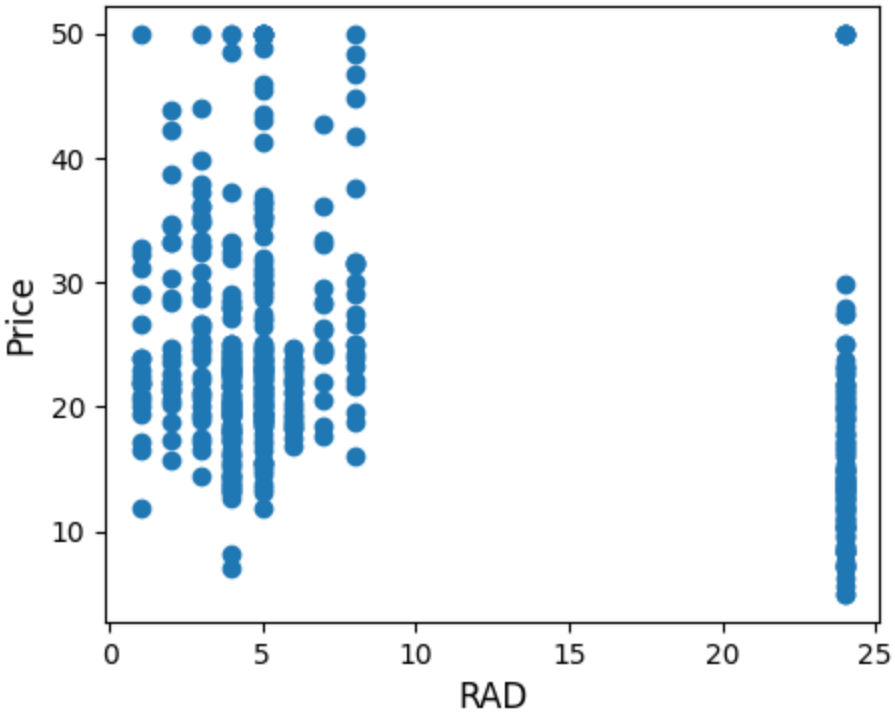

Price
CHAS
50
40
30
20
10
0.0
0.2
0.4
0.6
0.8
1.0

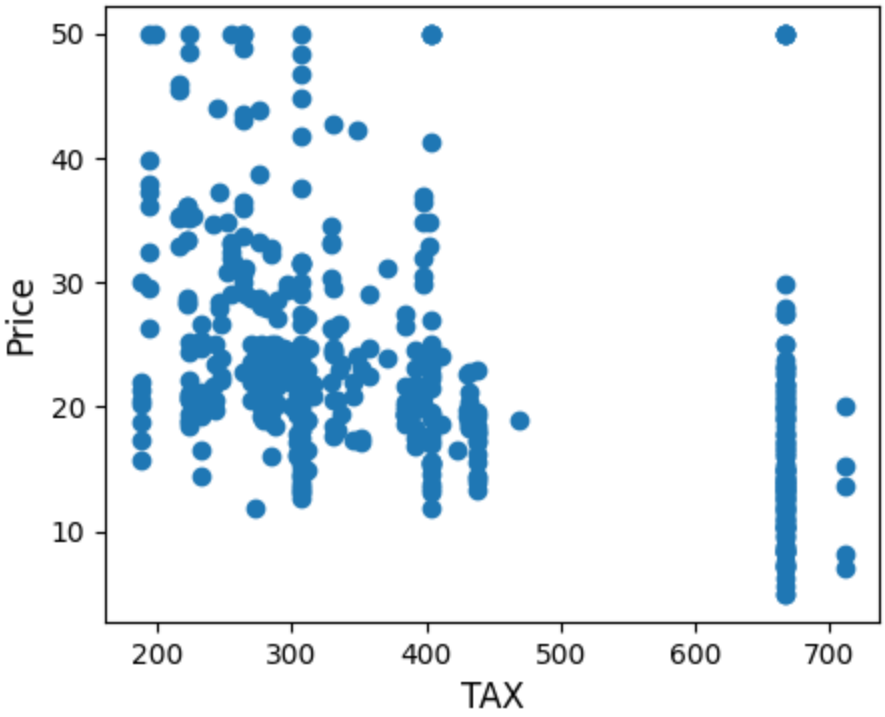

Price
NOX
50
40
30
20
10
0.4
0.5
0.6
0.7
0.8

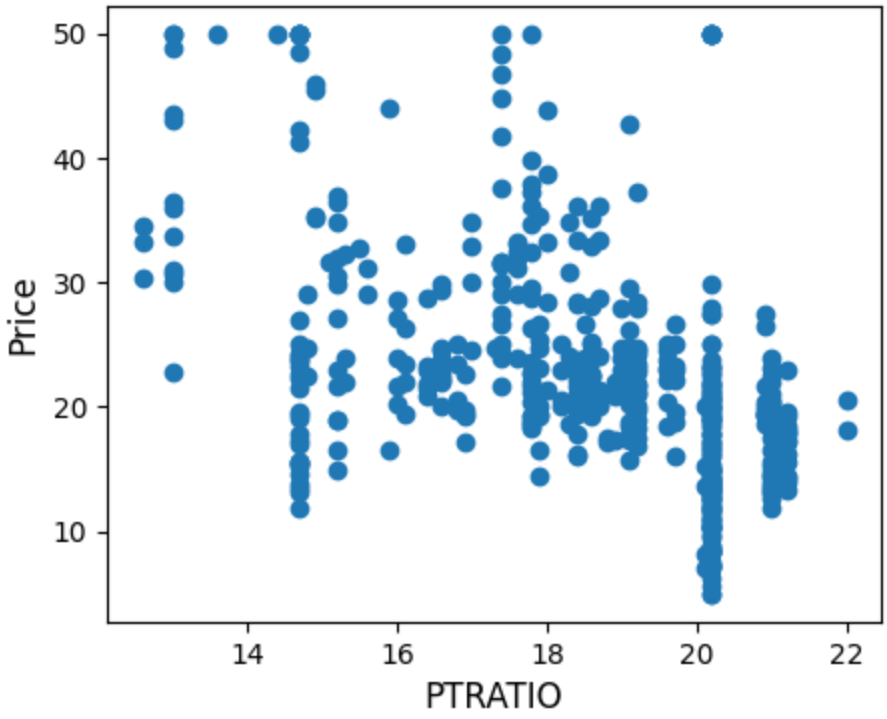

Price
RM
50
40
30
20
10
4
5
6
7
8
9

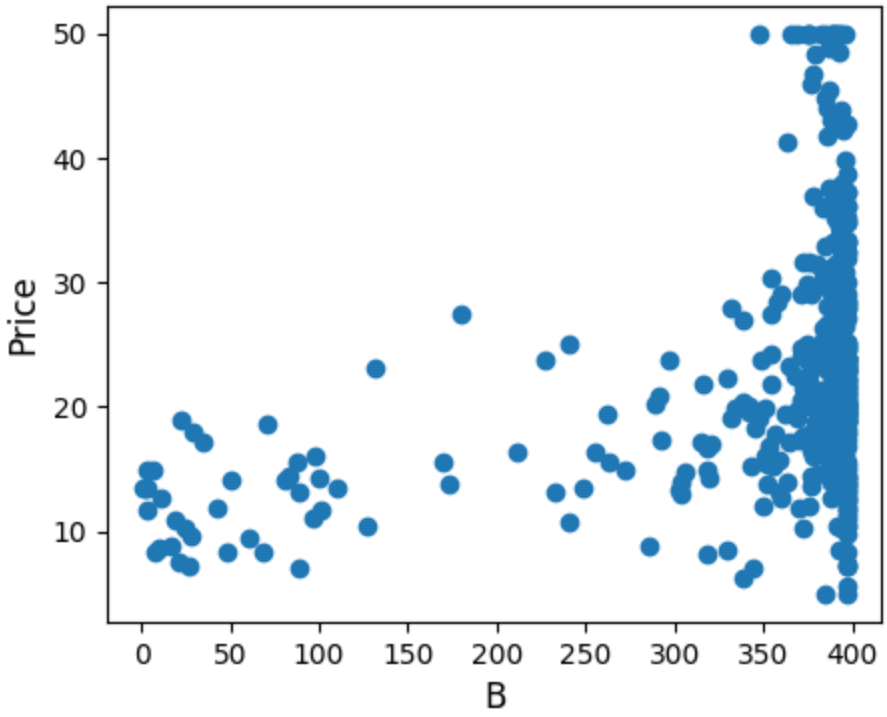

Price
AGE
50
40
30
20
10
0
20
40
60
80
100

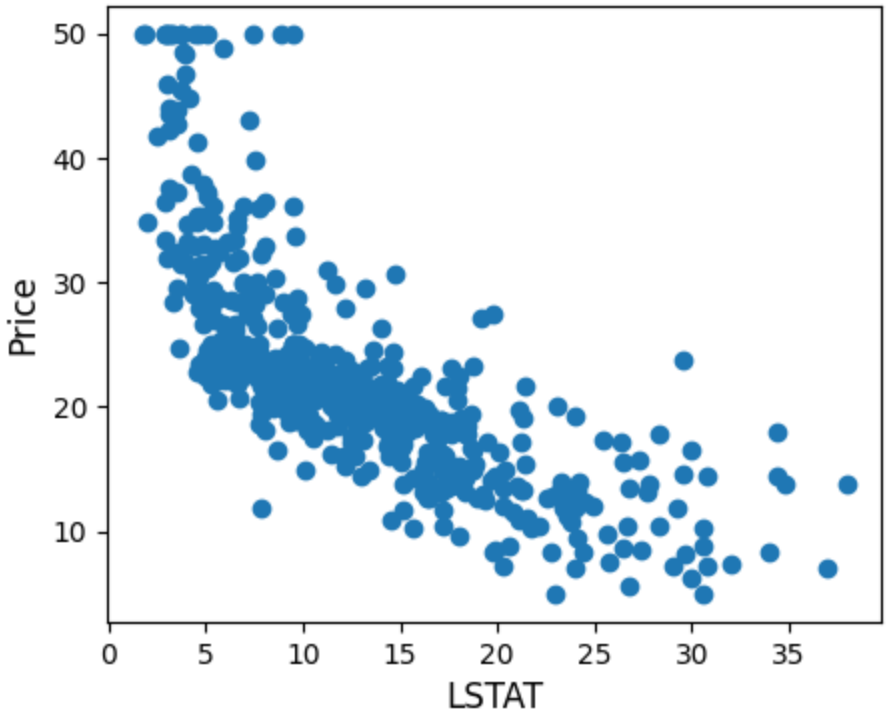

Price
DIS
50
40
30
20
10
2
4
6
8
10
12

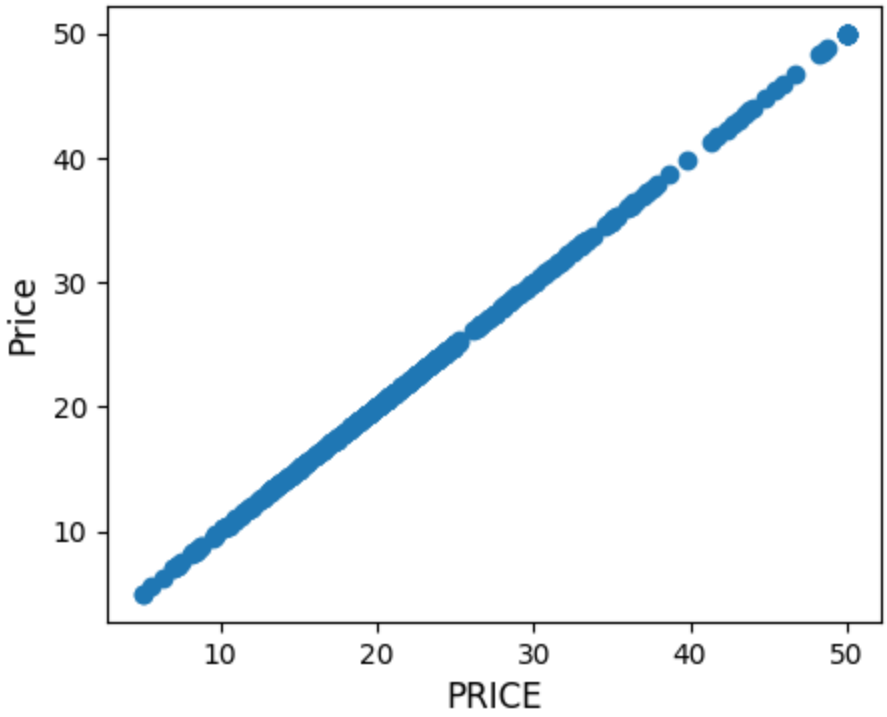

Price
RAD
50
40
30
20
10
0
5
10
15
20
25


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

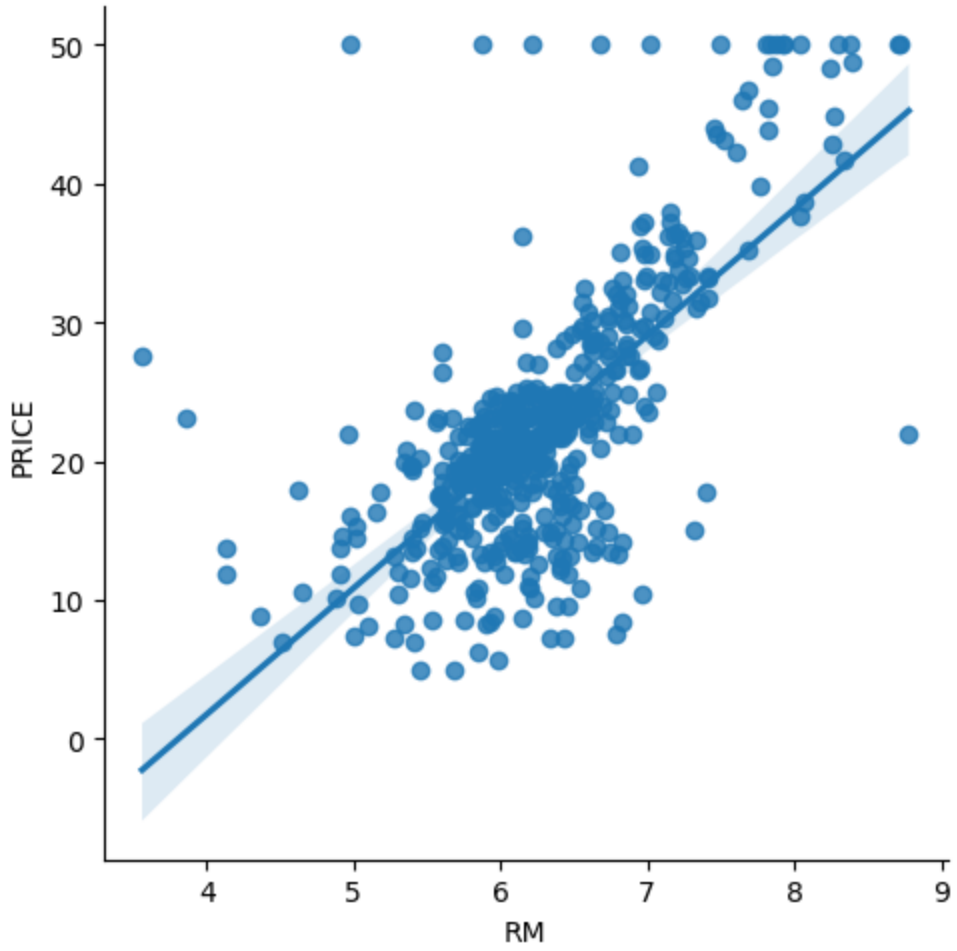

Price
B
50
40
30
20
10
0
50
100
150
200
250
300
350
400





- 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

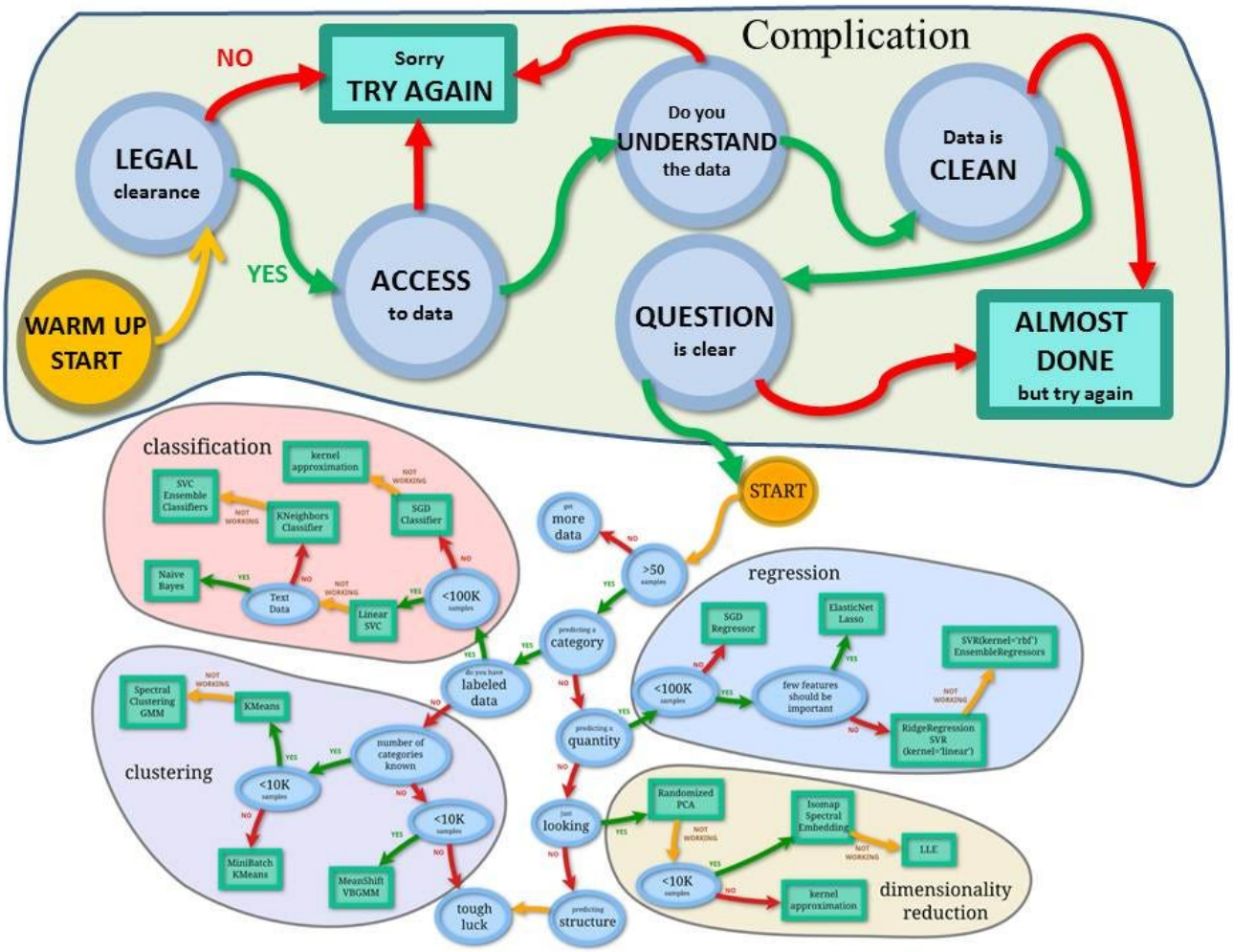


Image Source: Christophe Bourguignat

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
```

In [ ]:
```

```