

Inteligencia Artificial

Machine Learning - (Aprendizaje Automático)
Un caso de estudio



Machine Learning - La importancia de los datos



The Unreasonable Effectiveness of Data

Alon Halevy, Peter Norvig, and Fernando Pereira, Google

Eugene Wigner's article "The Unreasonable Effectiveness of Mathematics in the Natural Sciences"¹ examines why so much of physics can be neatly explained with simple mathematical formulas

such as $f = ma$ or $e = mc^2$. Meanwhile, sciences that involve human beings rather than elementary particles have proven more resistant to elegant mathematics. Economists suffer from physics envy over

behavior. So, this corpus could serve as the basis of a complete model for certain tasks—if only we knew how to extract the model from the data.

Learning from Text at Web Scale

The biggest successes in natural-language-related machine learning have been statistical speech recognition and statistical machine translation. The reason for these successes is not that these tasks are easier than other tasks; they are in fact much harder

En muchos problemas de IA, tener más y mejores datos es más importante que usar modelos más complejos

A. Halevy, P. Norvig and F. Pereira, "The Unreasonable Effectiveness of Data," in *IEEE Intelligent Systems*, vol. 24, no. 2, pp. 8-12, March-April 2009, doi: 10.1109/MIS.2009.36.

Machine Learning - La importancia de los datos

Quizás el ejemplo más famoso de sesgo de muestreo ocurrió durante las elecciones presidenciales de EE. UU. en 1936, en las que se enfrentaron Landon y Roosevelt: la revista **Literary Digest** realizó una encuesta muy grande, enviando cuestionarios por correo a unas **10 millones de personas**. Obtuvo **2,4 millones de respuestas** y predijo con gran confianza que **Landon obtendría el 57 %** de los votos. Sin embargo, **Roosevelt ganó con el 62 %**.

El error estuvo en el método de muestreo de Literary Digest:

Fuentes de direcciones sesgadas: Para obtener las direcciones a las que enviar las encuestas, Literary Digest usó directorios telefónicos, listas de suscriptores de revistas, listas de miembros de clubes, etc. Todas estas listas tendían a favorecer a personas con cierta posición económica, quienes tenían mayor probabilidad de votar al Partido Republicano (y, por ende, a Landon).

Baja tasa de respuesta (menos del 25 %): Menos de una cuarta parte de las personas encuestadas respondió. Esto introdujo un sesgo adicional, conocido como sesgo por no respuesta (**nonresponse bias**), ya que se excluía potencialmente a personas que no estaban interesadas en la política, que no simpatizaban con Literary Digest u otros grupos clave.

Machine Learning - Acerca de los modelos

El Teorema **No Free Lunch (NFL)** enunciado por *David Wolpert* en 1996, establece que no existe un modelo de aprendizaje automático que sea el mejor para todos los problemas. Cada modelo funciona bien solo en ciertos tipos de datos y mal en otros. Por ejemplo, un modelo lineal es adecuado si la relación entre las variables es lineal, pero no lo será si la relación es no lineal y compleja.

En la práctica, esto significa que **siempre estamos haciendo suposiciones implícitas sobre la naturaleza de los datos al elegir un modelo**. Como no es posible evaluar todos los modelos posibles, lo habitual es probar solo algunos que resulten razonables para el problema.

Machine Learning - Proceso de modelado y utilización

1. **Problem Definition:** Definir claramente qué problema se quiere resolver, los objetivos y las métricas a utilizar.
2. **Data Collection:** Obtener los datos necesarios desde las fuentes disponibles.
3. **Data Cleaning and Preprocessing:** Corregir errores, manejar valores faltantes, normalizar y transformar los datos para que el modelo los pueda usar.
4. **Exploratory Data Analysis:** Analizar visual y estadísticamente los datos para entender patrones, tendencias y posibles problemas.
5. **Feature Engineering and Selection:** Crear nuevas características relevantes y seleccionar las más útiles para mejorar el rendimiento del modelo.
6. **Model Selection:** Elegir el tipo de modelo que probablemente funcione mejor según el problema y los datos.
7. **Model Training:** Entrenar el modelo con los datos de entrenamiento para que aprenda patrones.
8. **Model Evaluation and Tuning:** Evaluar el rendimiento en datos de prueba y ajustar hiperparámetros para optimizarlo.
9. **Model Deployment and Maintenance:** Poner el modelo en producción para que empiece a usarse.

Machine Learning - Recolección de Datos

En esta etapa nos concentraremos en reunir de forma sistemática los conjuntos de datos que servirán como materia prima para entrenar el modelo. Durante la recolección, es fundamental garantizar que los datos **sean relevantes para el problema**, y que contengan **todas las características necesarias**.

Aspectos clave de la Recolección de Datos:

Relevancia: Reunir datos directamente relacionados con el problema incluyendo las características esenciales.

Calidad: Mantener la precisión, consistencia y cumplir con **estándares éticos**.

Cantidad: Asegurar un volumen suficiente de datos para entrenar un modelo.

Diversidad: Incluir conjuntos de datos variados para capturar una amplia gama de escenarios y patrones.

Machine Learning - Sanitización de Datos

Una vez recolectados los datos es necesario un proceso de sanitización, que involucra su estructuración, limpieza (descartando partes irrelevantes para el modelo), proveer consistencia, detectar datos faltantes, casos atípicos (outliers), etc.

Limpieza de Datos: Abordar problemas como valores faltantes, valores atípicos e inconsistencias en los datos.

Pre-procesamiento de Datos: Estandarizar formatos, escalar valores y codificar variables categóricas para lograr coherencia.

Calidad de Datos: Asegurar que los datos estén bien organizados y listos para un análisis significativo.

Machine Learning - Exploración de Datos

Para descubrir información y comprender la estructura del conjunto de datos, por ejemplo, encontrar patrones y características ocultas en los datos, se utiliza el Análisis Exploratorio de Datos. Las visualizaciones ayudan a presentar la información de forma comprensible.

Exploración: Utilizar herramientas estadísticas y visuales para explorar patrones en los datos.

Patrones y Tendencias: Identificar patrones subyacentes, tendencias y posibles desafíos dentro del conjunto de datos.

Perspectivas: Obtener información valiosa para la toma de decisiones en etapas posteriores, por ejemplo la selección de modelos.

Machine Learning - Selección e ingeniería de Características

Esta etapa consiste en seleccionar únicamente las características relevantes para la predicción del modelo. La ingeniería de características implica seleccionar características relevantes o crear nuevas características transformando las existentes para la predicción con el fin de mejorar la precisión y minimizar la complejidad computacional.

Ingeniería de Características: Crear nuevas características o transformar las existentes para capturar mejor patrones y relaciones.

Selección de Características: Identificar el subconjunto de características más relevantes para el modelo.

Optimización: Equilibrar el conjunto de características para maximizar la precisión y minimizar la complejidad computacional.

Machine Learning - Selección del modelo

La selección del modelo es una decisión clave que determina el marco algorítmico para la predicción. La elección depende de la naturaleza de los datos, la complejidad del problema y los resultados deseados.

Alineación: Elegir un modelo que se ajuste al problema definido y a las características del conjunto de datos.

Complejidad: Considerar la complejidad del problema y la naturaleza de los datos al seleccionar un modelo.

Factores de decisión: Evaluar aspectos como el rendimiento y la escalabilidad al elegir un modelo.

Experimentación: Probar diferentes modelos para encontrar el que mejor se adapte al problema.

Machine Learning - Entrenamiento

El entrenamiento del modelo es un **proceso iterativo** en el que el algoritmo ajusta sus parámetros para minimizar errores y mejorar la precisión predictiva. Un proceso de entrenamiento exitoso asegura que el modelo funcione bien con datos nuevos, garantizando predicciones confiables en escenarios reales.

Datos de entrenamiento: Determinar estrategias y porciones de datos para el entrenamiento.

Proceso iterativo: Entrenar el modelo de forma iterativa, ajustando parámetros para minimizar errores y mejorar la precisión (control de underfitting y overfitting).

Validación: Contrastar rigurosamente el modelo para asegurar su precisión con datos nuevos (generalización).

Machine Learning - Evaluación

La evaluación es fundamental para obtener información sobre las fortalezas y debilidades del modelo. Si el modelo no alcanza los niveles de rendimiento deseados, puede ser necesario volver a ajustarlo y modificar sus hiperparámetros para mejorar su precisión predictiva. Este ciclo iterativo de evaluación y ajuste es crucial para lograr el nivel de robustez y fiabilidad deseado en el modelo.

Métricas de evaluación: Utilizar métricas como MSE, RMSE, precisión, exhaustividad, etc. para evaluar el rendimiento del modelo, identificando fortalezas y debilidades.

Mejora iterativa: Realizar ajustes en los hiperparámetros para mejorar el modelo hasta alcanzar los niveles de confiabilidad deseados.

Machine Learning - Implementación (Deploy)

Una vez obtenido el modelo con una evaluación exitosa, el modelo está listo para su despliegue en aplicaciones del mundo real. El despliegue del modelo implica integrarlo en los sistemas existentes, permitiendo que las organizaciones utilicen sus predicciones para tomar decisiones informadas.

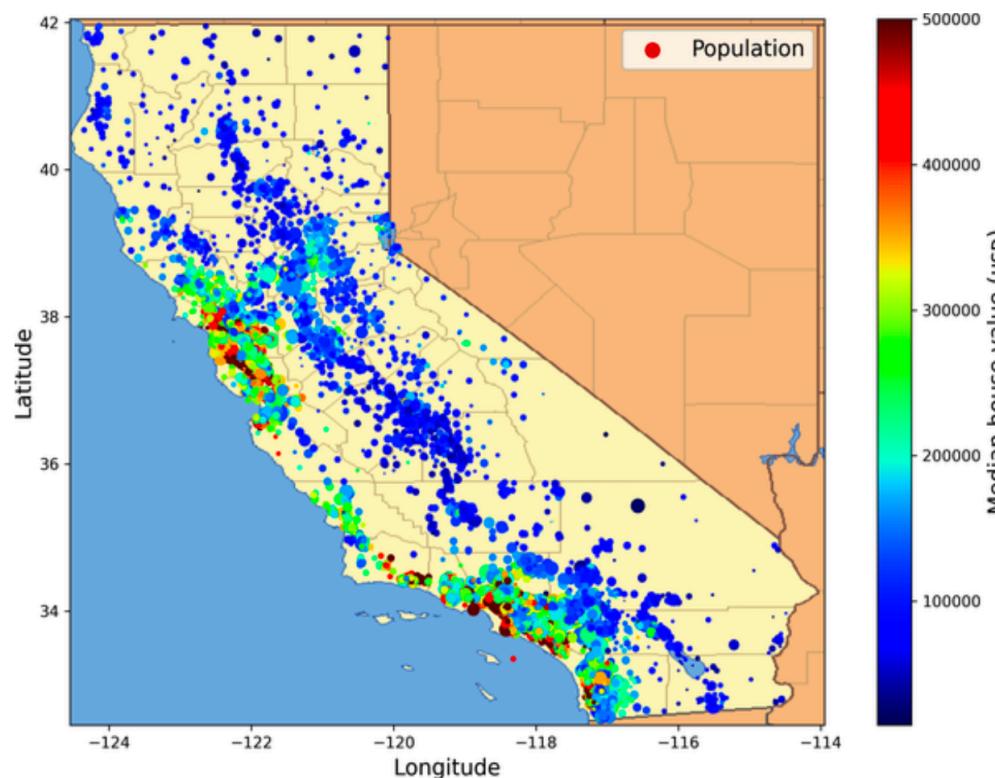
Integración: Incorporar el modelo entrenado en sistemas o procesos existentes para su uso en entornos reales.

Uso del modelo: Utilizar las predicciones del modelo para fundamentar decisiones informadas.

Mejora continua: Supervisar el rendimiento del modelo y realizar ajustes según sea necesario para mantener su eficacia a lo largo del tiempo.

Caso de Estudio: California Housing Prices

Se dispone de datos acerca del costo de propiedades (casas) en California y se espera crear un modelo que pueda predecir los valores de las casas según sus características, como lugar (geo-posición), salario promedio, dimensiones, valor, etc.



Caso de Estudio: California Housing Prices

Es un problema de **regresión múltiple** ya que el sistema utilizará **múltiples características** para realizar una predicción. También es un problema de regresión **univariante**, ya que **solo intentamos predecir el valor** para cada distrito. Si intentáramos predecir múltiples valores por distrito, sería un problema de regresión multivariante.

Una decisión a tomar es cuál será la métrica para determinar el aprendizaje:

Root Mean Squared Error (RMSE): También denominada norma euclídea o norma l_2

$$RMSE(X, h) = \sqrt{\frac{1}{m} \sum_1^m (h(X_i) - y_i)^2}$$

Mean Absolute Error (MAE): También denominada norma de manhattan o norma l_1

$$MAE(X, h) = \frac{1}{m} \sum_1^m |h(X_i) - y_i|$$

En general, la norma l_k de un vector se define como $l_k = (|v_0|^k + \dots + |v_n|^k)^{1/k}$. Normas más grandes ponen más énfasis en los valores grandes y menos en los valores pequeños (más propenso a ser afectado por valores atípicos).

Caso de Estudio: California Housing Prices

Comencemos las etapas de obtención de datos, su análisis, exploración y sanitización. Vamos utilizar un repositorio que cuenta con la información en formato csv (comma separated values).

```
from pathlib import Path
import pandas as pd
import tarfile
import urllib.request

def load_housing_data():
    tarball_path = Path("datasets/housing.tgz")
    if not tarball_path.is_file():
        Path("datasets").mkdir(parents=True, exist_ok=True)
        url = "https://github.com/ageron/data/raw/main/housing.tgz"
        urllib.request.urlretrieve(url, tarball_path)
    with tarfile.open(tarball_path) as housing_tarball:
        housing_tarball.extractall(path="datasets")
    return pd.read_csv(Path("datasets/housing/housing.csv"))

housing = load_housing_data()
```

longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY

Caso de Estudio: California Housing Prices

longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY

Cada fila es la información de un distrito que contiene 10 características:

- **longitude**: Longitud geográfica del distrito.
- **latitude**: Latitud geográfica del distrito.
- **housing_median_age**: Mediana sobre la edad de las viviendas.
- **total_rooms**: Número total de habitaciones en el distrito.
- **total_bedrooms**: Número total de dormitorios en el distrito.
- **population**: Población total del distrito.
- **households**: Número total de hogares en el distrito.
- **median_income**: Ingreso medio por hogar en el distrito.
- **median_house_value**: **Valor medio de las viviendas (objetivo de la predicción)**.
- **ocean_proximity**: Proximidad al océano (característica por categorías).

Mediana: se ordenan los valores y se toma el central o el promedio de los valores centrales. Es más representativa en presencia de valores atípicos (outliers)

Caso de Estudio: California Housing Prices

longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY

Algunas observaciones:

- **Información faltante**: Por ejemplo para la cantidad de dormitorios
- **ocean_proximity**: Proximidad al océano (característica por categorías) no tiene valores numéricos, sino un **enumerado**: *1H OCEAN, INLAND, NEAR OCEAN, NEAR BAY, ISLAND*. En estas situaciones podemos codificarlos para un modelo usando:
 - **One-hot encoding** (crear variables binarias para cada categoría).
 - **Ordinal encoding** (si hubiera un orden lógico, que aquí no lo hay).

Caso de Estudio: California Housing Prices

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.816909
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.615874
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

Promedio (media): Suma de todos los valores dividido entre el número de datos. La media es sensible a la distribución es simétrica, pero es sensible a los outliers.

Desviación estándar: Mide cuánto se dispersan los datos alrededor de la media. La desviación estándar es sensible a que los valores están muy alejados de la media.

Mínimo y máximo: Valores más pequeño y más grande en el conjunto de datos.

Cuartiles (25, 50, 75): Dividen los datos ordenados en cuatro partes iguales.

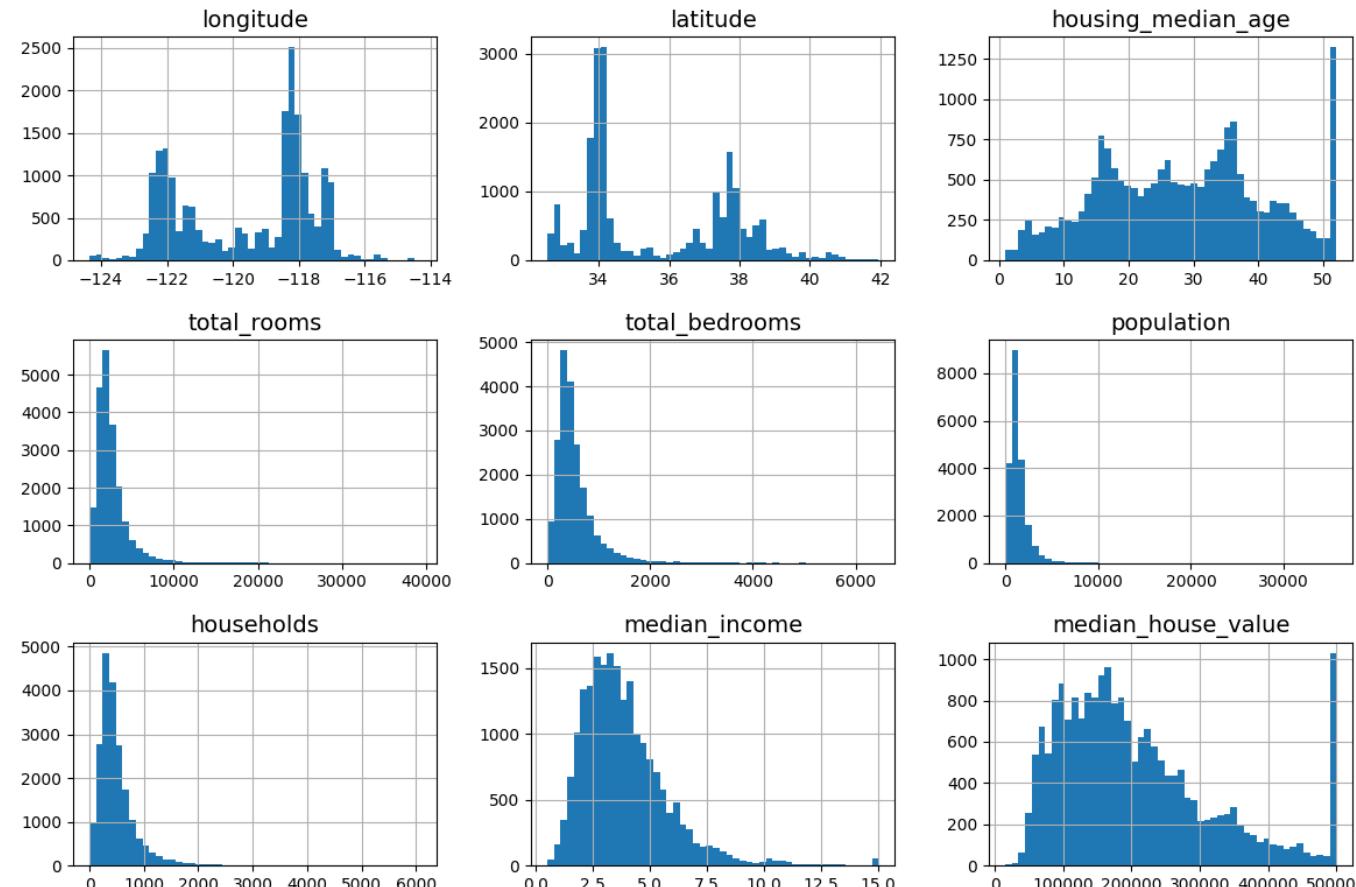
Q1 (25%): Valor por debajo del cual está el 25% de los datos.

Q2 (50%): Valor central que separa en dos mitades los datos.

Q3 (75%): Valor por debajo del cual está el 75% de los datos.

- Faltan algunos datos para la feature total_bedrooms
- *median_income*: está expresado en decenas de miles de dólares, y restringido a un rango entre 0,5 y 15
- *median_age* y *median_house_value* también están restringidos
 - *median_house_value* es lo que queremos predecir y nuestro algoritmo podría aprender que nunca puede superar los \$500.000
- Si queremos predecir mejor valores > \$500.000, tenemos dos opciones:
 - Mejorar la recolección de datos
 - Eliminar los distritos con valor \$500.000 del conjunto de datos
- Las características tienen escalas muy diferentes

Caso de Estudio: California Housing Prices



```
import matplotlib.pyplot as plt  
housing.hist(bins=50, figsize=(12, 8))  
plt.show()
```

- Las características tienen escalas muy diferentes
- Los histogramas se extienden más hacia la derecha de la media que hacia la izquierda. Esto podría hacer que algunos algoritmos de ML tengan dificultad en aprender patrones
- Podemos **escalar** las características para que tengan una distribución normal

Caso de Estudio: California Housing Prices

Una vez analizados los datos y sus características, dependiendo de la metodología de aprendizaje evaluación, debemos separar aquellos datos que vamos a utilizar para evaluar el/los modelos aprendidos.

Un error común es inspeccionar o seleccionar “manualmente” dicho conjunto. Esto derivará seguramente en un sesgo ya que podemos suponer ciertos patrones o criterios por conocer los datos de evaluación (**data snooping bias**).

Para este caso de estudio, podemos utilizar por ejemplo un **muestreo aleatorio**:

```
import numpy as np

def shuffle_and_split_data(data, test_ratio):
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
    return data.iloc[train_indices],
           data.iloc[test_indices]

train_set, test_set = shuffle_and_split_data(housing, 0.2)
len(train_set)
len(test_set)
```

Una forma similar pero con algunas mejoras es utilizar:

```
from sklearn.model_selection import
train_test_split

train_set, test_set =
train_test_split(housing,
                test_size=0.2, random_state=42)
```

Caso de Estudio: California Housing Prices

Si el conjunto de datos no es lo suficientemente grande, corremos el riesgo de cometer **sesgo por muestreo** en nuestra selección, por ejemplo que en la selección aleatoria, no se corresponda adecuadamente (con cierta proporción) la cantidad de muestras respecto a ciertas características. Para minimizar los riegos de sesgo por muestreo (selección de datos) podemos utilizar **muestreo estratificado**.

Es una técnica utilizada para garantizar que una muestra (como el conjunto de prueba) refleje la composición general de la población en relación con ciertas características importantes.

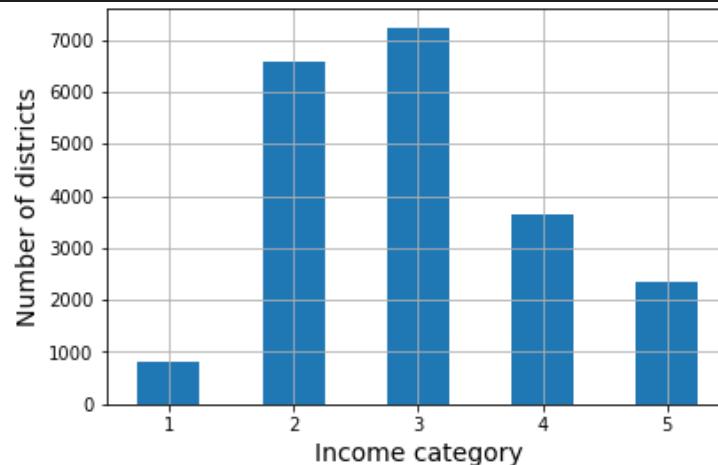
- La población se divide en **subgrupos homogéneos** llamados **estratos**.
- De cada estrato, se toma un **número proporcional de instancias**, de manera que la muestra conserve la misma distribución que el conjunto de datos completo.

Caso de Estudio: California Housing Prices

Por ejemplo, respecto de la categoría de sueldo promedio (relevante para nuestro problema), podemos categorizar (estratificar) en 5 categorías:

```
housing["income_cat"] = pd.cut(housing["median_income"],  
                                bins=[0., 1.5, 3.0, 4.5, 6., np.inf],  
                                labels=[1, 2, 3, 4, 5])
```

```
housing["income_cat"].value_counts().sort_index().plot.bar(rot=0,  
grid=True)  
plt.xlabel("Income category")  
plt.ylabel("Number of districts")  
plt.show()
```



Caso de Estudio: California Housing Prices

Para utilizar estas categorías en la selección del conjunto de prueba podemos utilizar:

```
strat_train_set, strat_test_set = train_test_split(  
    housing, test_size=0.2, stratify=housing["income_cat"], random_state=42)
```

Comparamos los dos tipos de selecciones focalizados en las categorías creadas

Income Category	Overall %	Stratified %	Random %	Strat. Error %	Rand. Error %
1	3.98	4.00	4.24	0.36	6.45
2	31.88	31.88	30.74	-0.02	-3.59
3	35.06	35.05	34.52	-0.01	-1.53
4	17.63	17.64	18.41	0.03	4.42
5	11.44	11.43	12.09	-0.08	5.63

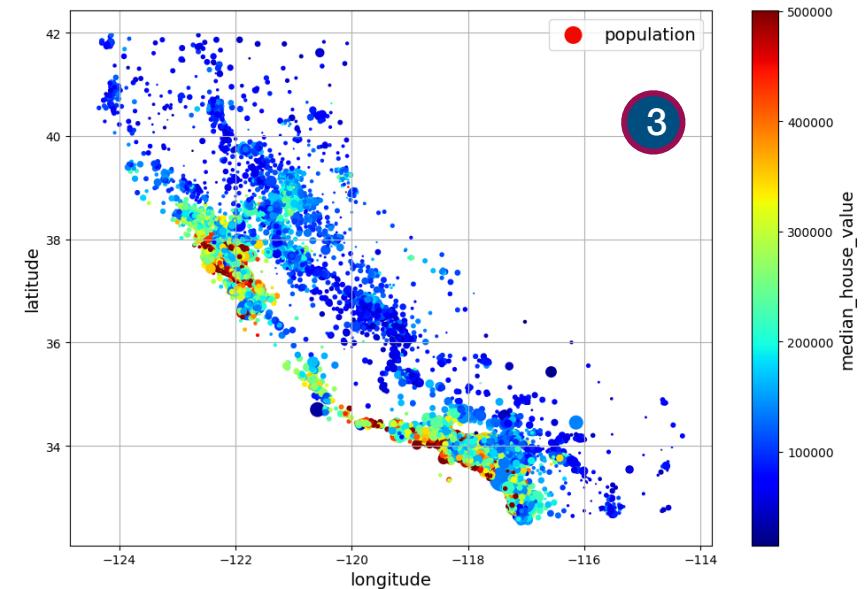
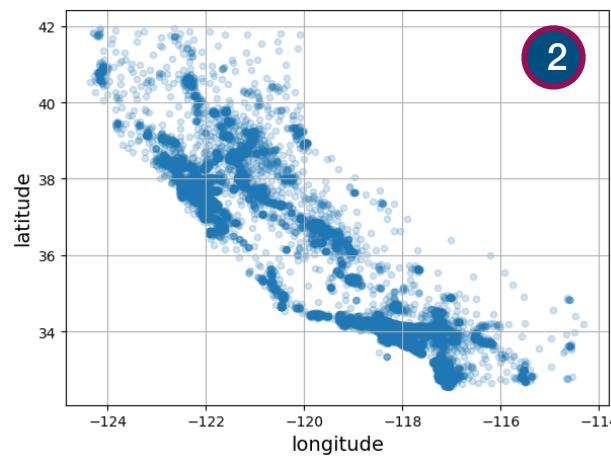
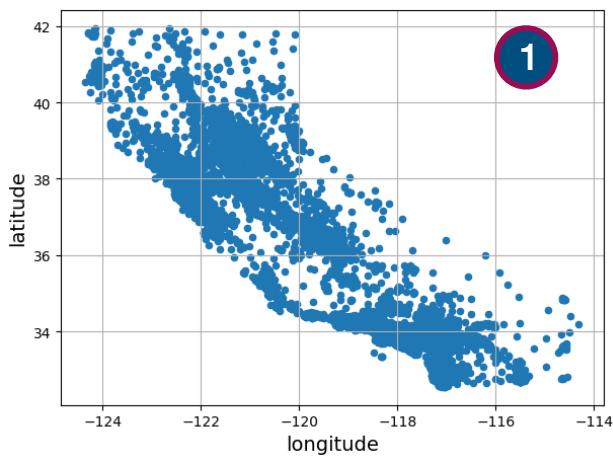
Una vez elegidos organizados ambos conjuntos, debemos quitar esa característica creada sólo al efecto de dividir los conjuntos de aprendizaje y prueba.

```
for set_ in (strat_train_set, strat_test_set):  
    set_.drop("income_cat", axis=1, inplace=True)
```

Caso de Estudio: California Housing Prices

En todo momento podemos visualizar la información:

- 1 housing.plot(kind="scatter", x="longitude", y="latitude", grid=True)
- 2 housing.plot(kind="scatter", x="longitude", y="latitude", grid=True, alpha=0.2)
- 3 housing.plot(kind="scatter", x="longitude", y="latitude", grid=True,
s=housing["population"] / 100, label="population",
c="median_house_value", cmap="jet", colorbar=True,
legend=True, sharex=False, figsize=(10, 7))
plt.show()

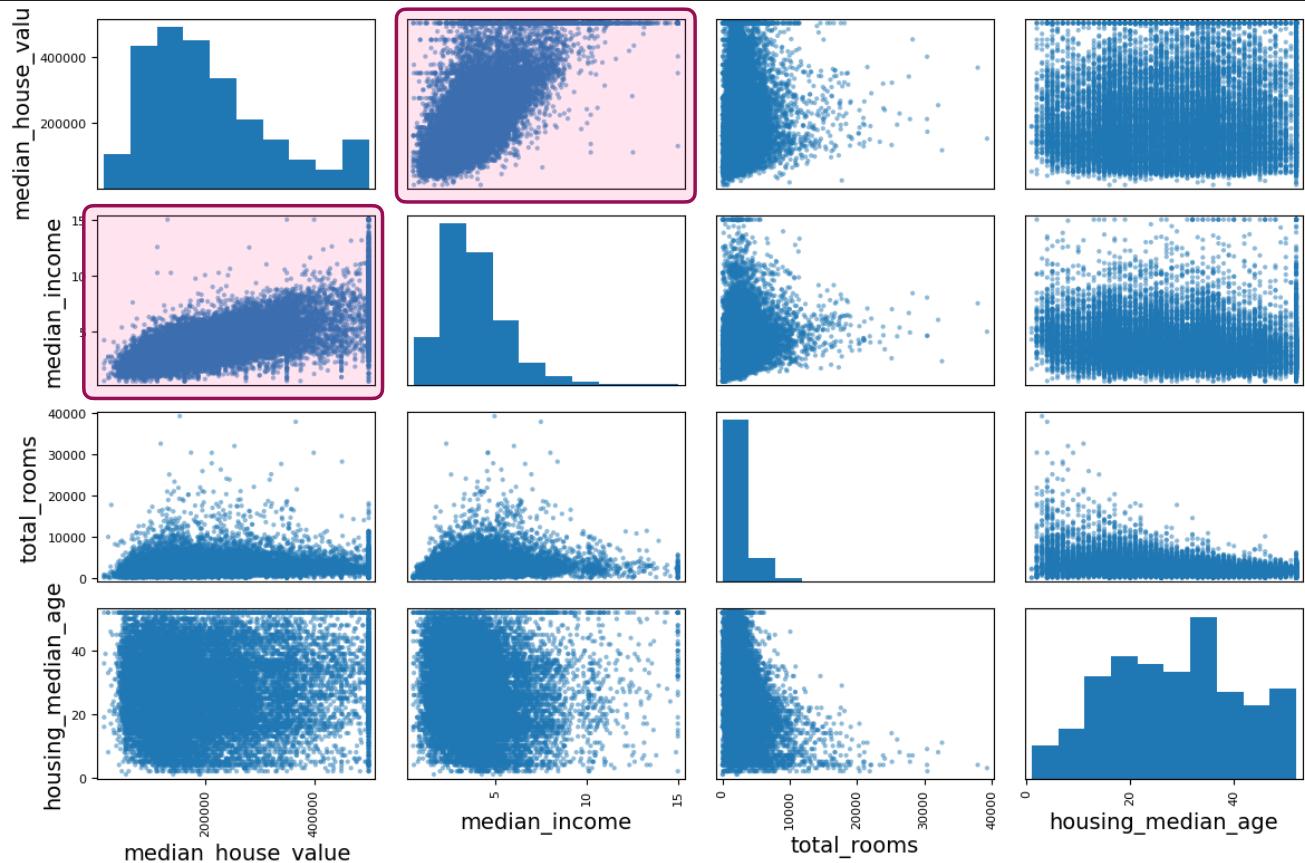


Caso de Estudio: California Housing Prices

```
from pandas.plotting import scatter_matrix

attributes = ["median_house_value", "median_income", "total_rooms", "housing_median_age"]
scatter_matrix(housing[attributes], figsize=(12, 8))
plt.show()
```

Otra mirada exploratoria es la búsqueda de **correlaciones**, por ejemplo, **Pearson's r** (coeficiente de correlación de Pearson) es una medida estadística que indica la fuerza y dirección de la relación lineal entre dos variables cuantitativas. Podemos contrastar esta medida entre las características de los datos a modelar.



Caso de Estudio: California Housing Prices

Esta medida nos da valores entre **-1** y **1** con la siguiente interpretación:

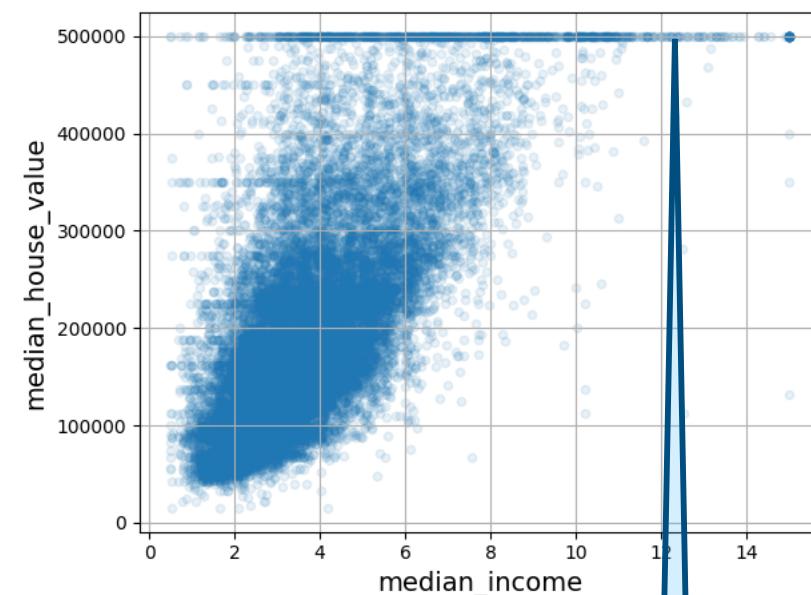
$r = 1$: correlación positiva perfecta (una sube, la otra sube proporcionalmente).

$r = -1$: correlación negativa perfecta (una sube, la otra baja proporcionalmente).

$r = 0$: no hay correlación lineal.

```
corr_matrix = housing.corr(numeric_only=True)
corr_matrix[ "median_house_value" ].sort_values(ascending=False)
```

	median_house_value
median_house_value	1.000000
median_income	0.688075
total_rooms	0.134153
housing_median_age	0.105623
households	0.065843
total_bedrooms	0.049686
population	-0.024650
longitude	-0.045967
latitude	-0.144160



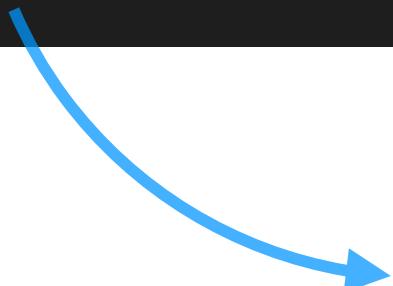
Los datos presentan límites y valores repetidos (p. ej., \$500.000, \$450.000, \$350.000) que evidencian ruido y podrían requerir la eliminación de distritos afectados o la mejora en su recolección para evitar que el modelo aprenda estos patrones.

Caso de Estudio: California Housing Prices

Durante el proceso de exploración, podemos derivar más características, por ejemplo:

```
housing[ "rooms_per_house" ] = housing[ "total_rooms" ] / housing[ "households" ]
housing[ "bedrooms_ratio" ] = housing[ "total_bedrooms" ] / housing[ "total_rooms" ]
housing[ "people_per_house" ] = housing[ "population" ] / housing[ "households" ]

corr_matrix = housing.corr(numeric_only=True)
corr_matrix[ "median_house_value" ].sort_values(ascending=False)
```



	median_house_value
median_house_value	1.000000
median_income	0.688380
rooms_per_house	0.143663
total_rooms	0.137455
housing_median_age	0.102175
households	0.071426
total_bedrooms	0.054635
population	-0.020153
people_per_house	-0.038224
longitude	-0.050859
latitude	-0.139584
bedrooms_ratio	-0.256397

Caso de Estudio: California Housing Prices

Una vez explorados los datos, comenzamos su preparación previa al entrenamiento.

Habíamos notado la ausencia del valor **total_bedrooms** en algunos distritos.

Tenemos tres opciones:

- Eliminar los distritos correspondientes
- Eliminar el atributo completo
- Asignarle algún valor a los distritos con problemas (cero, la media, la mediana, etc.)

```
housing.dropna(subset=[ "total_bedrooms" ], inplace=True)      # option 1
housing.drop("total_bedrooms", axis=1)                      # option 2
median = housing[ "total_bedrooms" ].median()    # option 3
housing[ "total_bedrooms" ].fillna(median, inplace=True)
```

Otra forma más eficiente es utilizando las implementaciones provistas por **Scikit-Learn**, que facilitan el reuso de las *transformaciones*, y sus sucesivas aplicaciones a distintos datasets (ej. a partes del dataset, al test set, etc.)

Caso de Estudio: California Housing Prices

Podemos usar el *transformer* Imputer de Scikit-Learn para llenar los datos faltantes con la **media**.

Primero eliminaremos el atributo *ocean_proximity* porque la media no se puede computar para atributos enumerados, luego creamos un Imputer que compute las medias del dataset con el método *fit()*. Finalmente aplicamos la transformación reemplazando los valores faltantes por la **media**.

```
from sklearn.impute import SimpleImputer

housing_num = housing.select_dtypes(include=[np.number]) #drop ocean_proximity

imputer = SimpleImputer(strategy="median")

imputer.fit(housing_num)

imputer.statistics_

X = imputer.transform(housing_num)
```

Scikit-Learn: API

Estimadores: Son objetos que aprenden parámetros a partir de un conjunto de datos usando el método `fit()`. Si es aprendizaje supervisado, reciben también las etiquetas. Los hiperparámetros de configuración (por ejemplo, `strategy` en `SimpleImputer`) y se definen al crear el objeto.

Transformadores: Estimadores que además pueden transformar datos mediante `transform()`, usando los parámetros aprendidos con `fit()`. Tienen el método `fit_transform()` para hacer ambos pasos de forma optimizada.

Predictores: Estimadores que pueden generar predicciones con `predict()` y evaluar su desempeño con `score()`.

Buenas prácticas de diseño:

- Los hiperparámetros y parámetros aprendidos son accesibles como atributos públicos (los aprendidos llevan guion bajo, ej. `statistics_`).
- Los datos se representan con arrays NumPy o matrices SciPy, no con clases personalizadas.
- Se fomenta la composición, como crear **pipelines** que encadenen transformadores y un estimador final.
- Se ofrecen valores por defecto razonables para facilitar la creación rápida de modelos funcionales.

Caso de Estudio: California Housing Prices

Dado que todos los algoritmos de Machine Learning funcionan mejor con números, debemos tratar la característica *ocean_proximity*, que es un enumerado.

En general tenemos dos opciones:

- Asignarle un valor entero diferente a cada valor enumerado, esto viene implementado por el transformador **OrdinalEncoder** de Scikit-Learn. Debemos tener precaución de no introducir alguna correlación o alteración a los algoritmos.
- Codificar los valores del atributo enumerado como **bitsets**, esto viene implementado por el transformador **OneHotEncoder** de Scikit-Learn.

```
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import OneHotEncoder

ordinal_encoder = OrdinalEncoder()
housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)

cat_encoder = OneHotEncoder()
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
```

Caso de Estudio: California Housing Prices

Una de las transformaciones más importantes en el preprocesamiento de datos es la normalización o escalado de características (**feature scaling**). La mayoría de los algoritmos de ML funcionan mal si las variables numéricas tienen **escalas muy diferentes**.

En el ejemplo de datos de viviendas, el número total de habitaciones varía de aproximadamente 6 a 39.320, mientras que el ingreso medio solo va de 0 a 15 (**Sin escalado, muchos modelos tenderán a ignorar el ingreso medio y dar más peso al número de habitaciones**).

Las dos técnicas más comunes para igualar la escala de todas las variables son:

- **Min-max scaling (normalización)**: ajusta los valores para que queden dentro de un rango fijo, normalmente [0, 1]. Mantiene la forma de la distribución original y es útil para algoritmos que asumen rangos acotados (ej. redes neuronales). Sin embargo, es muy sensible a *outliers*: un solo valor extremo cambia la escala de todo.
- **Estandarización (standardization)**: transforma los datos para que tengan media 0 y desviación estándar 1, restando a cada valor la media del conjunto de datos y lo divide por la desviación estándar. Si bien su rango no es fijo, no se ve afectada por la escala original, es más robusta ante valores atípicos y es más útil para algoritmos que asumen datos centrados (SVM, PCA, regresión logística, redes neuronales).

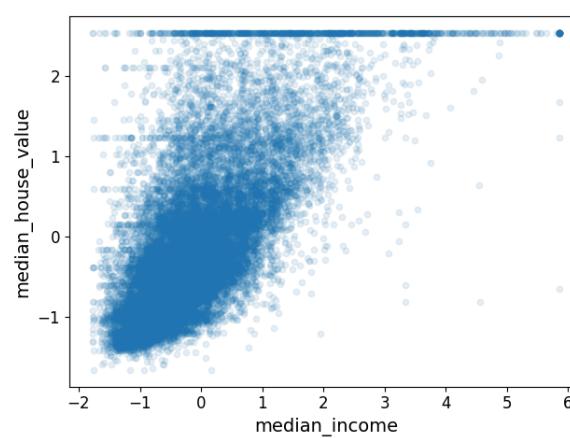
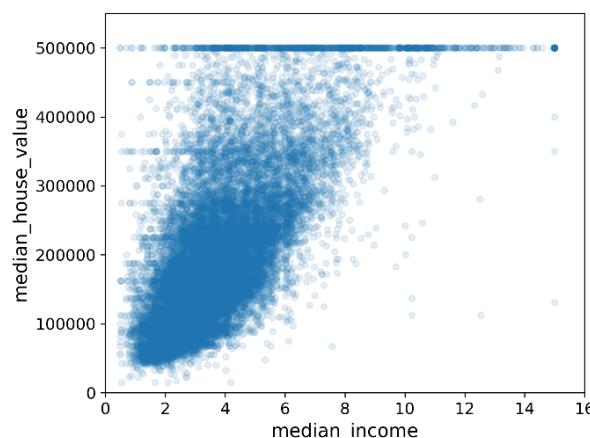
Caso de Estudio: California Housing Prices

Podemos utilizar los métodos provistos por Scikit-Learn para aplicar diferentes tipos de escalados necesario en nuestro conjunto de datos:

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler

min_max_scaler = MinMaxScaler(feature_range=(-1, 1))
housing_num_min_max_scaled = min_max_scaler.fit_transform(housing_num)

std_scaler = StandardScaler()
housing_num_std_scaled = std_scaler.fit_transform(housing_num)
```



Scikit-Learn: Pipelines de Transformación

Como vimos, hay muchos pasos de transformación de datos que deben ejecutarse en el orden correcto. Scikit-Learn proporciona la clase **Pipeline** para ayudar con este tipo de secuencias de transformaciones.

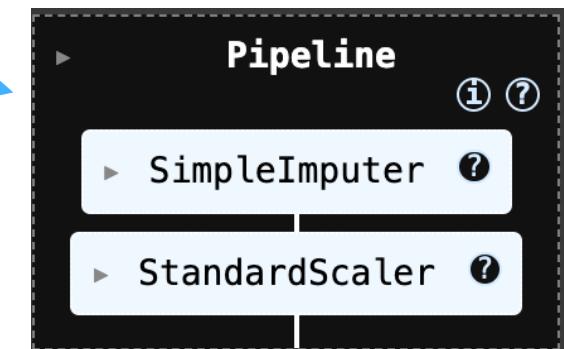
```
from sklearn.pipeline import make_pipeline
from sklearn import set_config
from sklearn.compose import ColumnTransformer

num_pipeline = make_pipeline(SimpleImputer(strategy="median"), StandardScaler())
set_config(display='diagram')
num_pipeline

num_attribs = ["longitude", "latitude", "housing_median_age", "total_rooms",
              "total_bedrooms", "population", "households", "median_income"]
cat_attribs = ["ocean_proximity"]

cat_pipeline = make_pipeline(
    SimpleImputer(strategy="most_frequent"),
    OneHotEncoder(handle_unknown="ignore"))

preprocessing = ColumnTransformer([
    ("num", num_pipeline, num_attribs),
    ("cat", cat_pipeline, cat_attribs),
])
housing_prepared = preprocessing.fit_transform(housing)
```



Datos listos para el aprendizaje

Caso de Estudio: California Housing Prices

Una vez preparados los datos, es hora del entrenamiento:

```
from sklearn.linear_model import LinearRegression  
  
lin_reg = make_pipeline(preprocessing, LinearRegression())  
lin_reg.fit(housing, housing_labels)
```

Podemos probar sus predicciones contrastándolas con los valores de entrenamiento:

```
housing_predictions = lin_reg.predict(housing)  
housing_predictions[:5].round(-2) # -2 = rounded to the nearest hundred  
housing_labels.iloc[:5].values  
  
lin_rmse = root_mean_squared_error(housing_labels, housing_predictions)  
lin_rmse
```

```
array([242800., 375900., 127500., 99400., 324600.])
```

```
array([458300., 483800., 101700., 96100., 361800.])
```

```
-47.0%, -22.3%, 25.4%, 3.4%, -10.3% !!
```

```
68647.95 !!
```

Demasiado “lejos” seguramente
estamos en presencia de
underfitting

Caso de Estudio: California Housing Prices

Probemos con otro modelo:

```
from sklearn.tree import DecisionTreeRegressor  
  
tree_reg = make_pipeline(preprocessing, DecisionTreeRegressor(random_state=42))  
tree_reg.fit(housing, housing_labels)
```

Podemos probar sus predicciones contrastándolas con los valores de entrenamiento:

```
housing_predictions = tree_reg.predict(housing)  
tree_rmse = root_mean_squared_error(housing_labels, housing_predictions)  
tree_rmse
```

0.0 !!

Demasiado “cerca” seguramente
estamos en presencia de **overfitting**

Caso de Estudio: California Housing Prices

Probemos entrenando y validando con **Cross-Validation**:

```
from sklearn.model_selection import cross_val_score

tree_rmses = -cross_val_score(tree_reg, housing, housing_labels, scoring="neg_root_mean_squared_error", cv=10)
lin_rmses = -cross_val_score(lin_reg, housing, housing_labels, scoring="neg_root_mean_squared_error", cv=10)
pd.Series(tree_rmses).describe()
pd.Series(lin_rmses).describe()
```

En Scikit-Learn, el RMSE se devuelve como valor negativo porque la validación cruzada espera que valores mayores sean mejores. Por eso, hay que invertir el signo para obtener el RMSE real.

$k = 10$

	count	10.000000	count	10.000000
mean	66366.983603		mean	69847.923224
std	1976.844743		std	4078.407329
min	63557.655007		min	65659.761079
25%	65004.623899		25%	68088.799156
50%	65886.897085		50%	68697.591463
75%	68129.026040		75%	69800.966364
max	69530.301101		max	80685.254832

!!

Sigue siendo un modelo demasiado impreciso. Además confirma que el modelo “tree” entendido estaba en presencia de overfitting

Caso de Estudio: California Housing Prices

Una más y no jodemos más:

Analizaremos en detalles estos modelos

```
rom sklearn.ensemble import RandomForestRegressor  
  
forest_reg = make_pipeline(preprocessing,  
                           RandomForestRegressor(random_state=42))  
forest_rmses = -cross_val_score(forest_reg, housing, housing_labels,  
                               scoring="neg_root_mean_squared_error", cv=10)  
  
pd.Series(forest_rmses).describe()
```

count	10.000000
mean	46938.209246
std	1018.397196
min	45522.649195
25%	46291.334639
50%	47021.703303
75%	47321.521991
max	49140.832210

```
forest_reg.fit(housing, housing_labels)  
housing_predictions = forest_reg.predict(housing)  
forest_rmse = root_mean_squared_error(housing_labels, housing_predictions)  
forest_rmse
```

17521.56

Claramente este modelo es mejor que los anteriores

Caso de Estudio: California Housing Prices

Como lo mencionamos, entrenar un modelo para que funcione adecuadamente puede requerir de una gran cantidad de pruebas en búsqueda del mismo. Más aún, para cada modelo bajo prueba, pueden existir diferentes hiperparámetros.

Una vez seleccionados algunos modelos potencialmente adecuados, es hora del **Fine Tuning**. Esto es ajustar los hiperparámetros para lograr el mejor ajuste posible. Una forma de sistematizar este proceso de búsqueda que provee Scikit-Learn es GridSearchCV.

```
from sklearn.model_selection import GridSearchCV

full_pipeline = Pipeline([
    ("preprocessing", preprocessing),
    ("random_forest", RandomForestRegressor(random_state=42)),
])
param_grid = [
    {'preprocessing_geo_n_clusters': [5, 8, 10],
     'random_forest_max_features': [4, 6, 8]},
    {'preprocessing_geo_n_clusters': [10, 15],
     'random_forest_max_features': [6, 8, 10]},
]
grid_search = GridSearchCV(full_pipeline, param_grid, cv=3,
                           scoring='neg_root_mean_squared_error')
grid_search.fit(housing, housing_labels)
grid_search.best_params_
```

Valores para los **hiperparámetros**.

Esta funcionalidad realiza una **validación cruzada con todas las combinaciones de los valores propuestos y devuelve los hiperparámetros que obtuvieron el mejor rendimiento**

{'preprocessing_geo_n_clusters': 15,
 'random_forest_max_features': 6}

Caso de Estudio: California Housing Prices

Una vez obtenido un modelo, debemos probar su funcionamiento con los datos de prueba.

```
x_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].copy()

final_predictions = final_model.predict(X_test)

final_rmse = root_mean_squared_error(y_test, final_predictions)
print(final_rmse)
```

41556.05

Otra medida interesante a analizar es el **Intervalo de Confianza**, que nos informa el rango del valor bajo una cierta probabilidad (expectativa) de **confianza**.

```
from scipy import stats

def rmse(squared_errors):
    return np.sqrt(np.mean(squared_errors))

confidence = 0.95
squared_errors = (final_predictions - y_test) ** 2
boot_result = stats.bootstrap([squared_errors], rmse,
                               confidence_level=confidence, random_state=42)
rmse_lower, rmse_upper = boot_result.confidence_interval
```

[39629.60, 43816.66]

Caso de Estudio: California Housing Prices

Monitoreo del modelo

- Luego del deployment, idealmente se deben monitorear las predicciones que hace el modelo de ML sobre nuevos datos, y detectar rápidamente degradaciones en el rendimiento.
- Es posible que los datos con los que se consulta el modelo vayan cambiando con el paso del tiempo (data drift). Por ejemplo, una aplicación que detecta patrones en fotografías puede degradar su performance por diversos motivos: nuevas cámaras de mayor resolución, nuevos filtros, nuevos objetos de moda, etc.

Gestión y calidad de datos

- A medida que van ingresando nuevos datos para hacer consultas al modelo, almacenarlos y usarlos para reentrenar periódicamente.
- Si el rendimiento se degrada con los sucesivos entrenamientos, esto puede indicar problemas con la recolección de datos. También pueden indicar problemas en la recolección: detección de datos faltantes, outliers cada vez más frecuentes, etc.

Reentrenamiento y validación

- Antes del deployment de un modelo entrenado con nuevos datos, evaluar el rendimiento del modelo previo y del nuevo con el dataset completo. Sólo publicar si el nuevo modelo supera en performance al anterior.
- Crear scripts para automatizar esta tarea.

Seguridad y respaldo

- Hacer backups de los modelos anteriores para restaurarlos rápidamente si se detectan problemas (ej. una degradación) en el modelo actual.