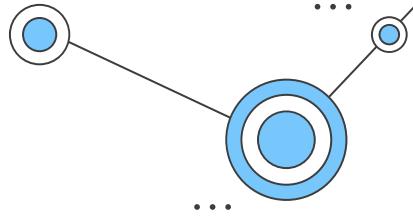


Proyecto de Machine Learning de principio a fin



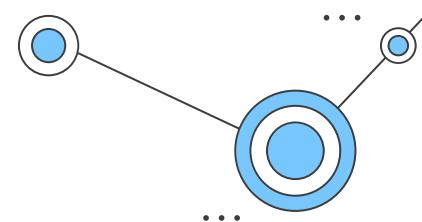
UNIVERSIDAD
CATÓLICA
DE CÓRDOBA
JESUITAS

Dr. Francisco Arduh
2023



Pasos

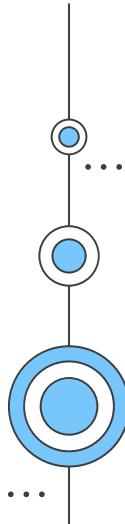
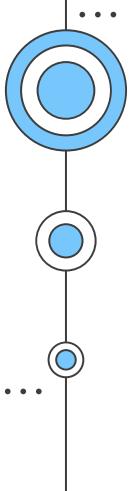
1. Mirar el panorama general.
2. Conseguir los datos.
3. Descubrir y visualizar los datos para ganar entendimiento.
4. Preparar los datos para los algoritmos de Machine Learning.
5. Seleccionar el modelo y entrenarlo.
6. Ajuste del modelo.
7. Presentar la solución.
8. Lanzarla, monitorearla y mantenerla.



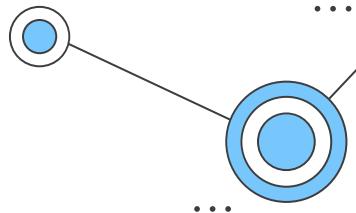
Pasos (CRISP-DM)

1. Mirar el panorama general. (Entendimiento del Negocio)
2. Obtener los datos.
3. Descubrir y visualizar los datos para ganar entendimiento.(Entendimiento de los datos)
4. Preparar los datos para los algoritmos de Machine Learning. (Preparación de los datos)
5. Seleccionar el modelo y entrenarlo.(Modelado)
6. Ajuste del modelo. (Evaluación)
7. Presentar la solución. (Despliegue)
8. Lanzarla, monitorearla y mantenerla. (Despliegue)

Panorama general

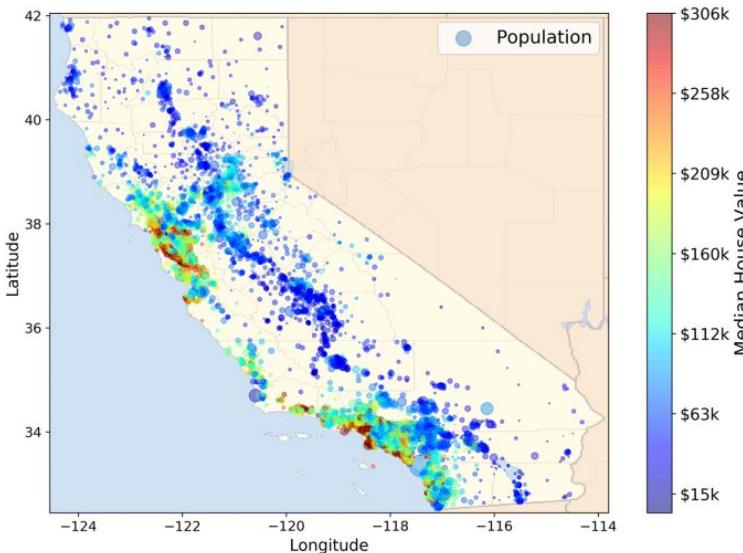


Mirar el panorama general

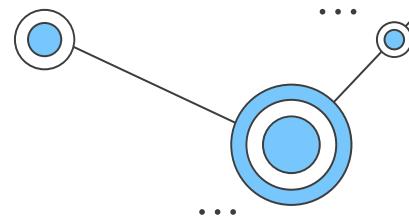


Se necesita construir un modelo de los precios de las casas en el estado de California a partir de los datos generados del censo.

- La unidad más pequeña que el US Census Bureau publica los resultados es de grupos de entre 600 y 3000 personas, a lo que los llamaremos distrito.
- Nuestro modelo debería poder predecir el precio medio en cualquier distrito.

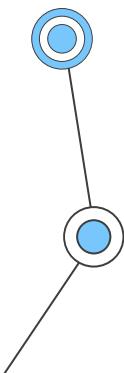


Mirar el panorama general: Definir el problema



Necesitamos saber:

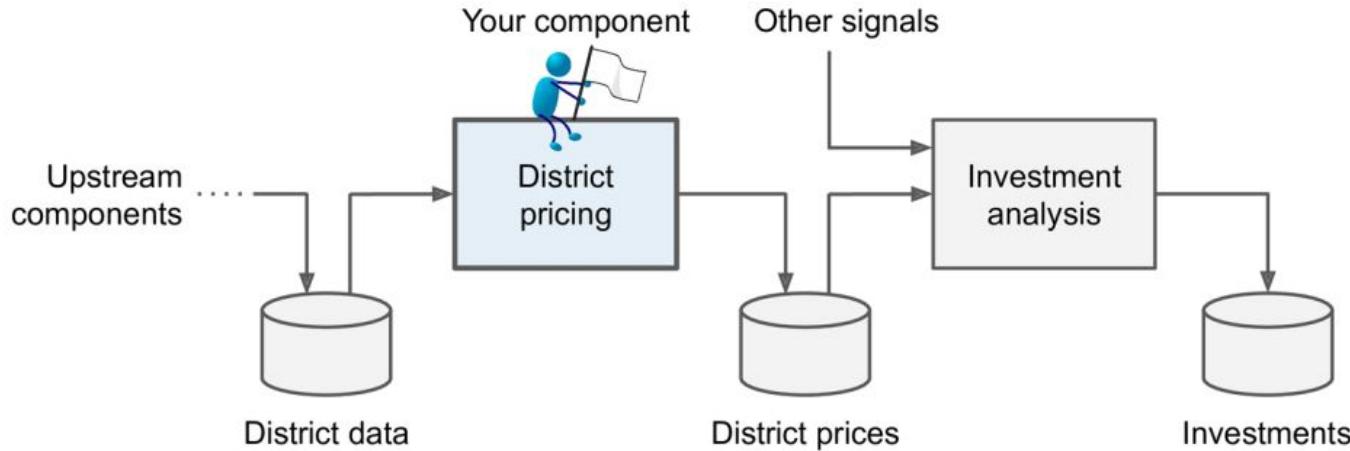
- ¿Cómo la compañía espera usar y beneficiarse de nuestro modelo?
- ¿Qué se utiliza actualmente? ¿Cuál es el desempeño de este método?
- ¿En qué categorías cae el algoritmo a utilizar?



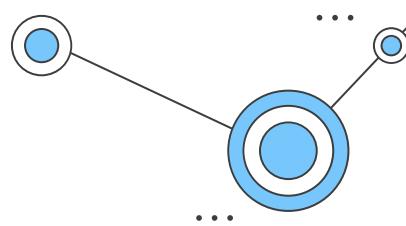
Mirar el panorama general: Definir el problema

¿Cómo la compañía espera usar y beneficiarse de nuestro modelo?

Nuestro modelo va a alimentar a otro modelo con los precios para saber si es conveniente invertir o no.

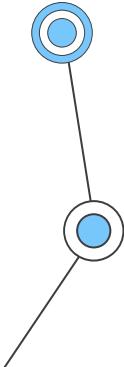


Mirar el panorama general: Definir el problema



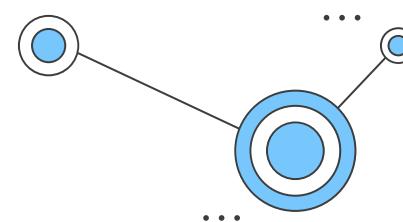
¿Qué se utiliza actualmente? ¿Cuál es el desempeño de este método?

Actualmente se utilizan expertos que calculan el precio de forma manual que suelen tener un error de alrededor de 20%.

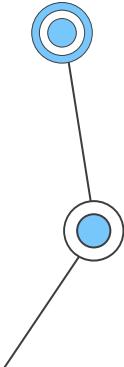


Mirar el panorama general: Definir el problema

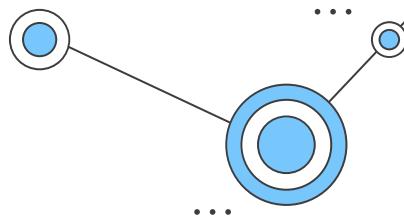
¿En qué categorías cae el algoritmo a utilizar?



- La información etiquetada → supervisado
- Se espera un valor real positivo → regresión (univariada)
- Se cuenta con muchas variables → regresión múltiple
- No contamos con un flujo constante de datos y la cantidad de datos no es muy grande → batch learning



Mirar el panorama general: Selección de medida desempeño



En caso de regresión las medidas típicas son:

- RMSE (Root Mean Square Error):

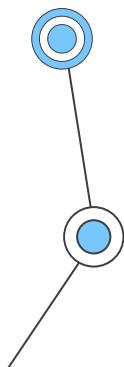
$$\text{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m} \sum_{i=1}^m (h(\mathbf{x}^{(i)}) - y^{(i)})^2}$$

- MAE (Mean absolute error)

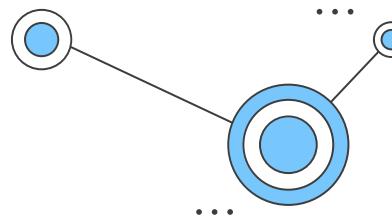


$$\text{MAE}(\mathbf{X}, h) = \frac{1}{m} \sum_{i=1}^m |h(\mathbf{x}^{(i)}) - y^{(i)}|$$

Nota: Es posible usar otro tipo de normas, pero a medida que crezca el su valor se tiene más en cuenta los valores atípicos. Por lo que MAE (que corresponde a L1) va a tener menos en cuenta los valores atípicos que RMSE (L2)



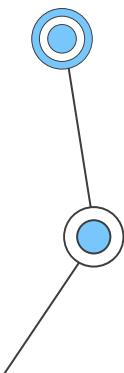
Mirar el panorama general: Chequeo de suposiciones



Es importante chequear suposiciones antes de empezar para poder evitar posibles inconvenientes más adelante.

En este ejemplo se podría dar el caso que el sistema al que le vamos a pasar el precio espere una categoría (barato, mediano o caro) en vez del precio.

Es necesario estar en contacto con el resto de los equipos para dispersar este tipo de dudas.



Configuración



Scikit-learn

- Librería creada por David Cournapeau
- Herramientas simples y eficientes para el análisis predictivo.
- Accesible y reutilizable.
- Construido sobre Numpy, SciPy y matplotlib
- Open source, comercialmente utilizable.

Link a la librería: <https://scikit-learn.org/stable/index.html>

Setear un ambiente virtual

Crear ambiente virtual

```
$ cd $ML_PATH  
$ python3 -m virtualenv my_env
```

Activarlo:

```
$ cd $ML_PATH  
$ source my_env/bin/activate # on Linux or macOS  
$ .\my_env\Scripts\activate # on Windows
```

Instalar librerías necesarias

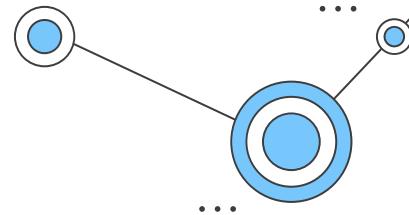
```
$ python3 -m pip install -U jupyter matplotlib numpy pandas scipy scikit-learn
```

Registrar jupyter en el ambiente virtual:

```
$ python3 -m ipykernel install --user --name=python3
```

Levanto jupyter:

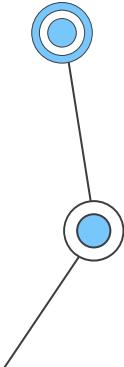
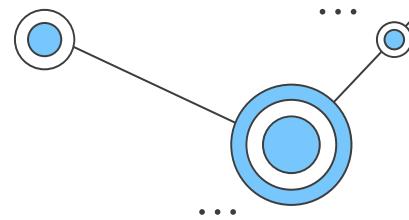
```
$ jupyter notebook
```



Obtener los datos

Obtener los datos

- Esto podría estar en base de datos, documentos/tablas/archivos.
- Datos privados, se necesitará autorización.
- En este ejemplo los datos son públicos y están en una tabla dentro de un archivo “.csv”.



Obtener los datos: descargar datos

```
import os
import tarfile
import urllib

DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
HOUSING_PATH = os.path.join("datasets", "housing")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"

def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):
    os.makedirs(housing_path, exist_ok=True)
    tgz_path = os.path.join(housing_path, "housing.tgz")
    urllib.request.urlretrieve(housing_url, tgz_path)
    housing_tgz = tarfile.open(tgz_path)
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()
```

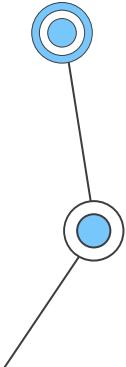
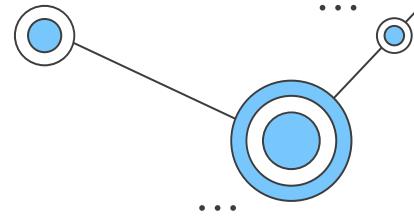
Obtener los datos: cargar datos

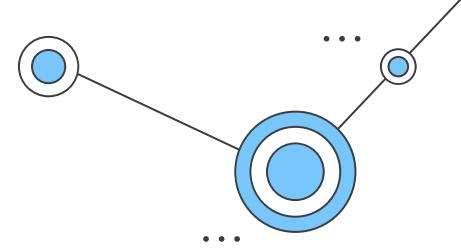
```
import pandas as pd

def load_housing_data(housing_path=HOUSING_PATH):
    csv_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(csv_path)
```

Obtener los datos: Mirar rápidamente la estructura de los datos

Se pueden utilizar métodos de pandas como `.head()`, `.info()`, `.describe()`, `value_counts()`, incluso `hist()` para ver su distribuciones de las variables





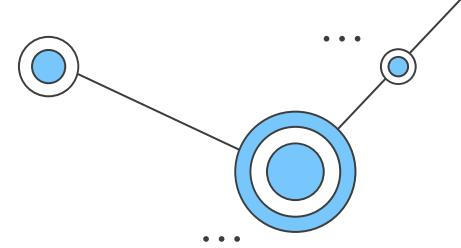
Obtener los datos: Mirar rápidamente la estructura de los datos

Se pueden utilizar métodos de pandas como .head(), .info(), .describe(), value_counts(), incluso hist() para ver su distribuciones de las variables

```
In [5]: housing = load_housing_data()  
housing.head()
```

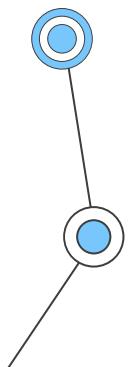
Out[5]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
0	-122.23	37.88	41.0	880.0	129.0	322.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0
3	-122.25	37.85	52.0	1274.0	235.0	558.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0



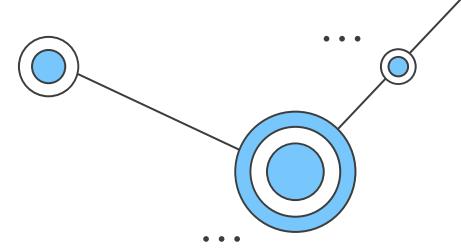
Obtener los datos: Mirar rápidamente la estructura de los datos

Se pueden utilizar métodos de pandas como .head(), .info(), .describe(), value_counts(), incluso hist() para ver su distribuciones de las variables



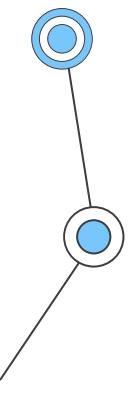
```
In [6]: housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude           20640 non-null float64
latitude            20640 non-null float64
housing_median_age  20640 non-null float64
total_rooms          20640 non-null float64
total_bedrooms       20433 non-null float64
population          20640 non-null float64
households          20640 non-null float64
median_income        20640 non-null float64
median_house_value   20640 non-null float64
ocean_proximity      20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```



Obtener los datos: Mirar rápidamente la estructura de los datos

Se pueden utilizar métodos de pandas como .head(), .info(), .describe(), value_counts(), incluso hist() para ver su distribuciones de las variables



```
>>> housing["ocean_proximity"].value_counts()  
<1H OCEAN      9136  
INLAND         6551  
NEAR OCEAN     2658  
NEAR BAY        2290  
ISLAND           5  
Name: ocean_proximity, dtype: int64
```

Obtener los datos: Mirar rápidamente la estructura de los datos

Se pueden utilizar métodos de pandas como .head(), .info(), .describe(), value_counts(), incluso hist() para ver su distribuciones de las variables

In [8]: `housing.describe()`

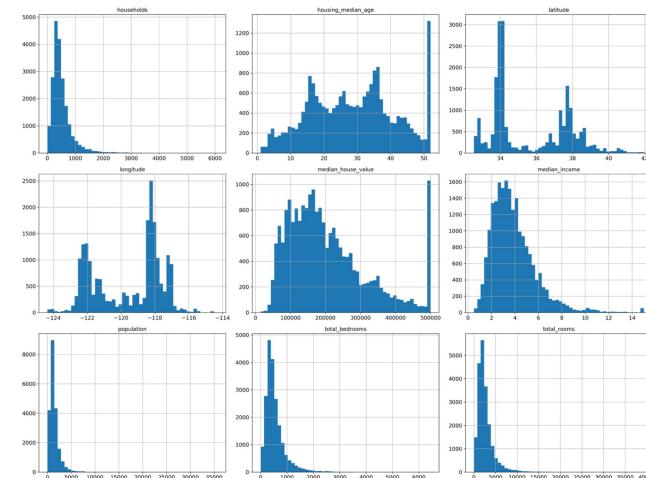
Out[8]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553
std	2.003532	2.135952	12.585558	2181.615252	421.385070
min	-124.350000	32.540000	1.000000	2.000000	1.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000

Obtener los datos: Mirar rápidamente la estructura de los datos

```
import matplotlib.pyplot as plt  
housing.hist(bins=50, figsize=(20,15))  
plt.show()
```

- Mirar escalas y formas, ¿qué información no proporcionan?

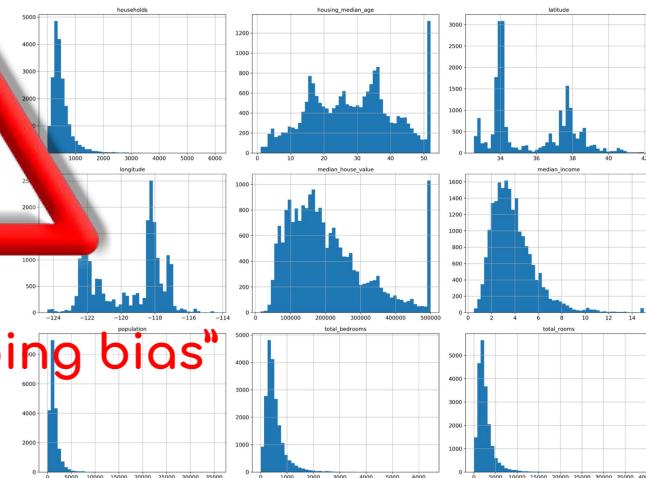
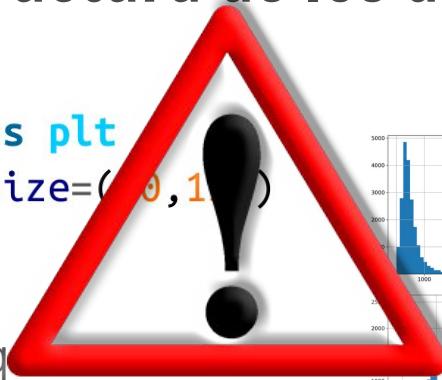


Obtener los datos: Mirar rápidamente la estructura de los datos

```
import matplotlib.pyplot as plt  
housing.hist(bins=50, figsize=(10,10))  
plt.show()
```

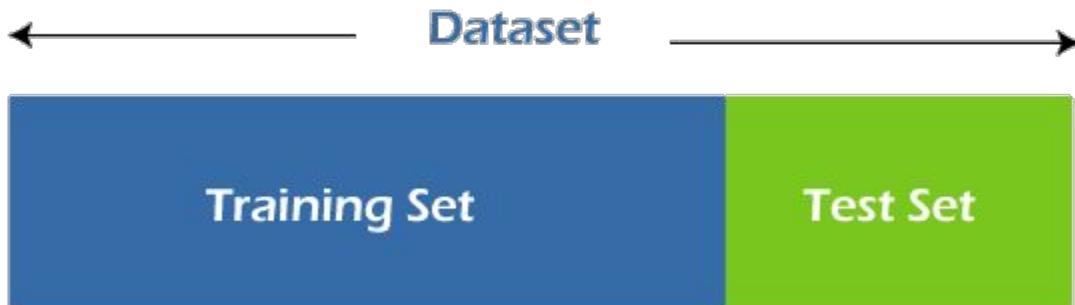
- Mirar escalas y formas, ¿qué información no proporcionan?

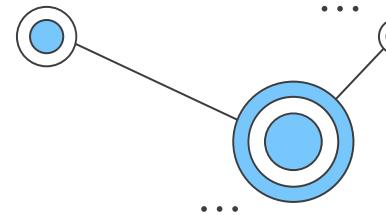
Cuidado con “snooping bias”



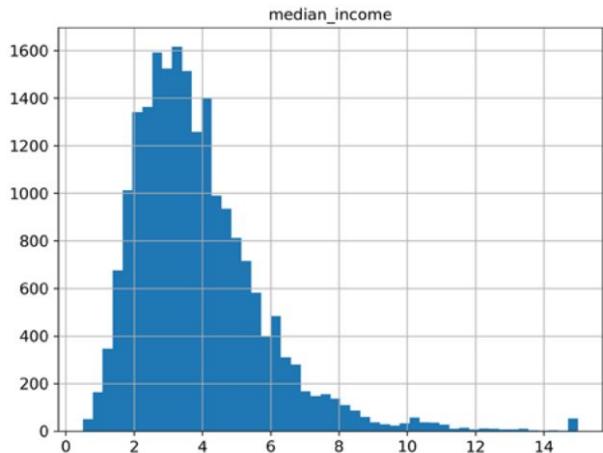
Creación un dataset de evaluación (test set)

```
from sklearn.model_selection import train_test_split  
  
train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
```



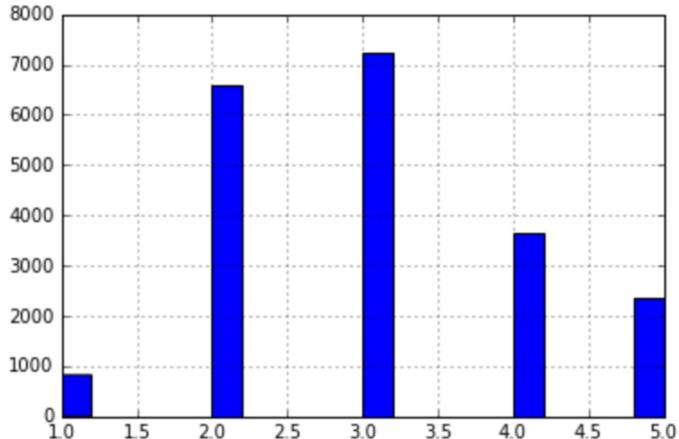


Creación un dataset de evaluación (test set) estratificado



```
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])
```

```
from sklearn.model_selection import StratifiedShuffleSplit
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing["income_cat"]):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]
```



Creación un dataset de evaluación (test set) estratificado

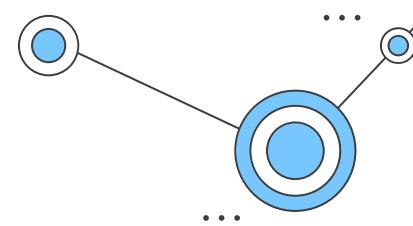
	Overall	Stratified	Random	Rand. %error	Strat. %error
1	0.039826	0.039729	0.040213	0.973236	-0.243309
2	0.318847	0.318798	0.324370	1.732260	-0.015195
3	0.350581	0.350533	0.358527	2.266446	-0.013820
4	0.176308	0.176357	0.167393	-5.056334	0.027480
5	0.114438	0.114583	0.109496	-4.318374	0.127011

Eliminamos la columna que utilizamos para el procedimiento

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

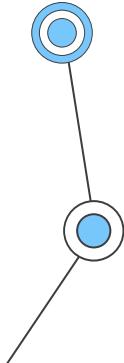
Obtener los datos

Descubrir y visualizar los datos para ganar entendimiento



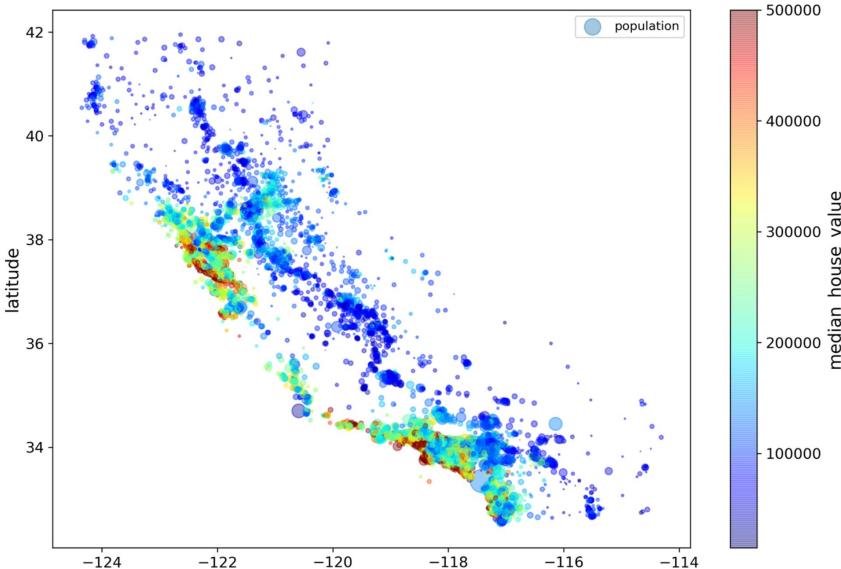
- Para no modificar el training set, lo copiamos.
- Si el training set es muy grande para esta tarea podría utilizar un subset

```
housing = strat_train_set.copy()
```

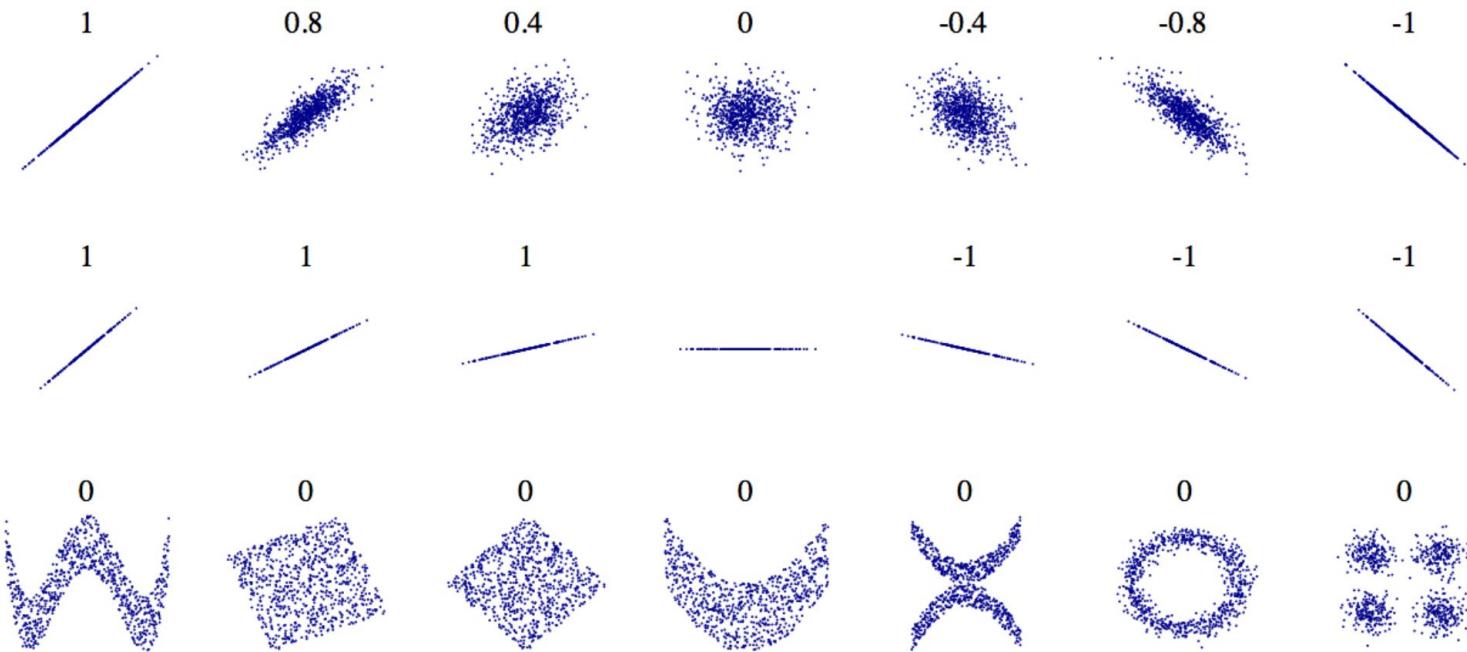
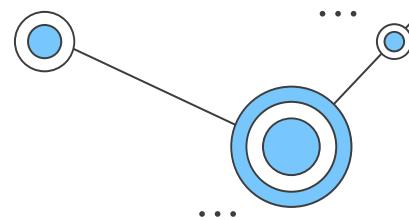


Visualización de datos geográficos

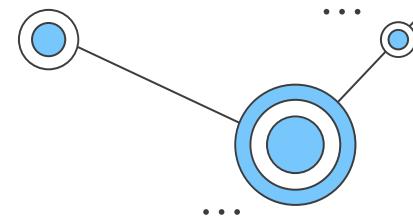
```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,
             s=housing["population"]/100, label="population", figsize=(10,7),
             c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True,
)
plt.legend()
```



Correlación Lineal (Pearson)



Correlaciones Lineal (Pearson) en el ejemplo



```
corr_matrix = housing.corr()
```

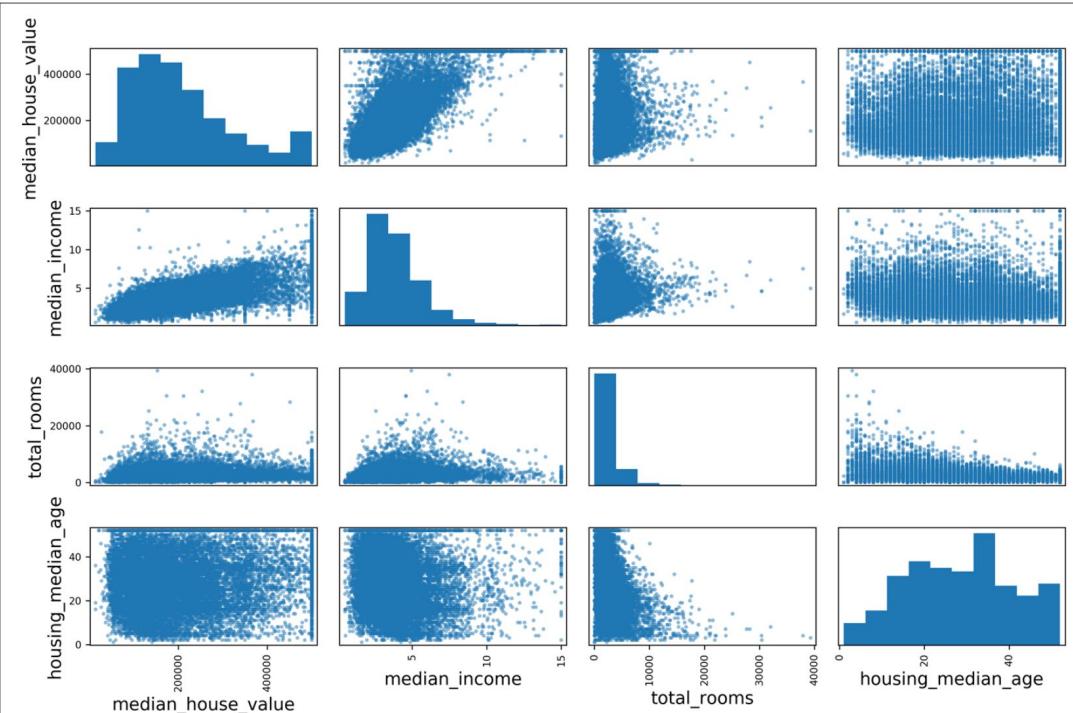
```
corr_matrix["median_house_value"].sort_values(ascending=False)
```

median_house_value	1.000000
median_income	0.687160
total_rooms	0.135097
housing_median_age	0.114110
households	0.064506
total_bedrooms	0.047689
population	-0.026920
longitude	-0.047432
latitude	-0.142724



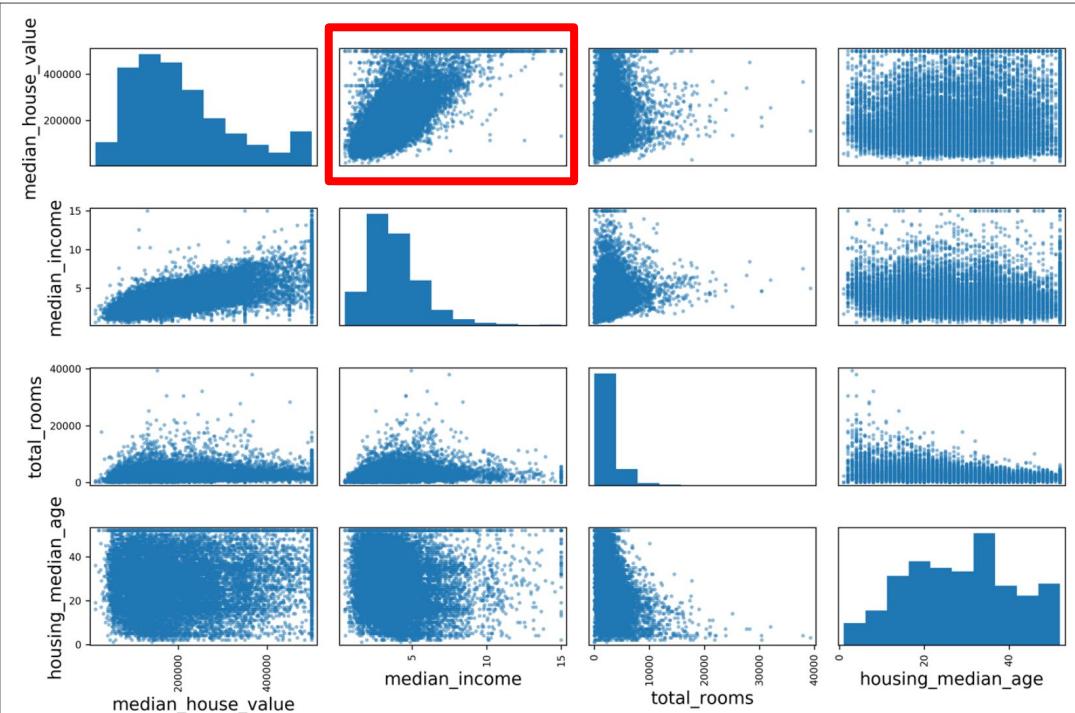
Función: scatter_matrix

```
from pandas.plotting import scatter_matrix  
  
attributes = ["median_house_value", "median_income", "total_rooms",  
              "housing_median_age"]  
scatter_matrix(housing[attributes], figsize=(12, 8))
```

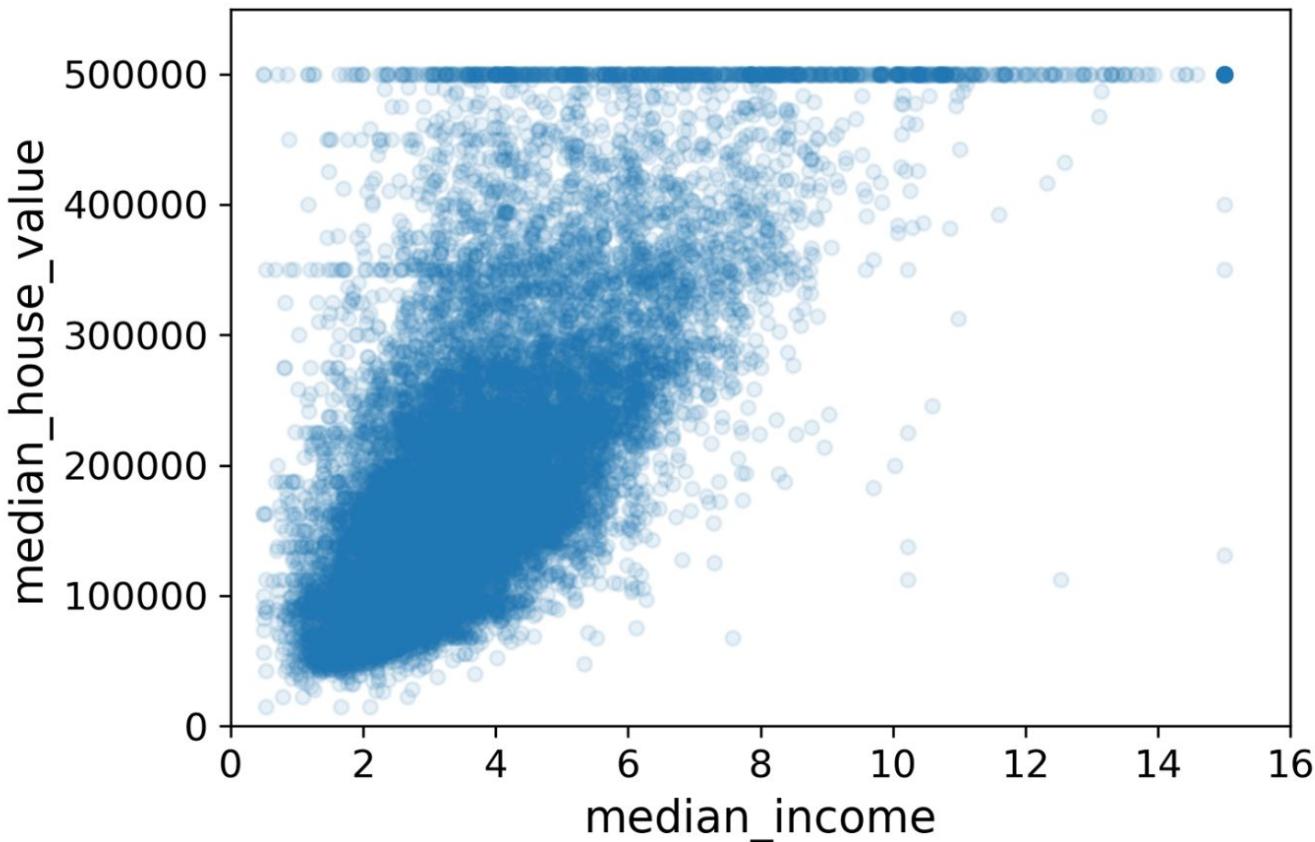


Función: scatter_matrix

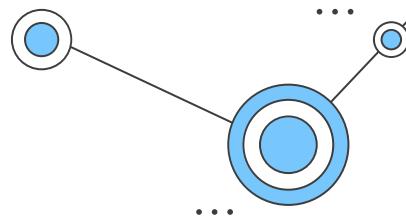
```
from pandas.plotting import scatter_matrix  
  
attributes = ["median_house_value", "median_income", "total_rooms",  
              "housing_median_age"]  
scatter_matrix(housing[attributes], figsize=(12, 8))
```



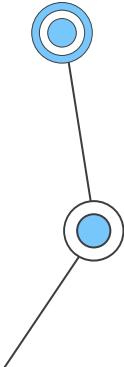
Correlación más fuerte



Experimentando con combinaciones de atributos



```
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"] = housing["population"]/housing["households"]
```



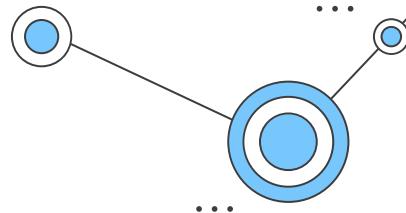
Experimentando con combinaciones de atributos

```
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"] = housing["population"]/housing["households"]
```

```
>>> corr_matrix = housing.corr()
>>> corr_matrix["median_house_value"].sort_values(ascending=False)
median_house_value           1.000000
median_income                 0.687160
rooms_per_household          0.146285
total_rooms                   0.135097
housing_median_age            0.114110
households                     0.064506
total_bedrooms                  0.047689
population_per_household      -0.021985
population                      -0.026920
longitude                       -0.047432
latitude                        -0.142724
bedrooms_per_room                -0.259984
Name: median_house_value, dtype: float64
```

Preparar los datos para algoritmos de Machine Learning

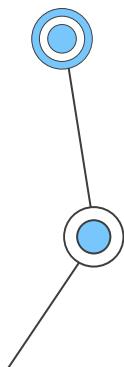
Preparando los datos



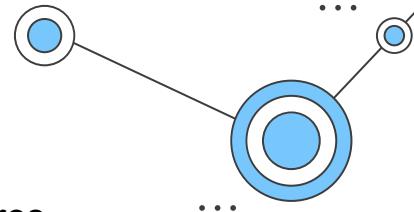
En vez de preparar los manualmente es mejor hacerlo en funciones por lo siguiente:

- Te permite reproducir las transformaciones fácilmente en cualquier conjunto de datos.
- Vas a construir gradualmente una librería de transformación de funciones qué puedes utilizar en otros proyectos de machine learning.
- Puedes utilizar esas funciones una vez deployado el algoritmo para transformar la nueva información.
- Te facilitará probar varias transformaciones para entender qué combinación es mejor.

```
housing = strat_train_set.drop("median_house_value", axis=1)  
housing_labels = strat_train_set["median_house_value"].copy()
```

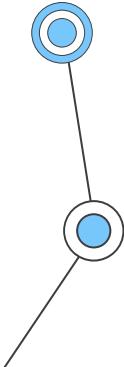


Limpieza de datos

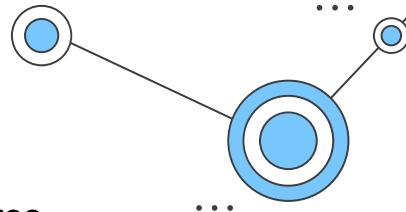


Cuando nos topamos con valores faltantes en nuestros dataset tenemos tres opciones.

- Deshacerse de filas correspondientes.
- Deshacerse todo el atributo. (columna)
- Reemplazar los valores faltantes con otro valor (media, mediana, cero, etc).



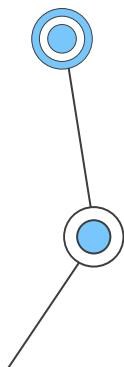
Limpieza de datos



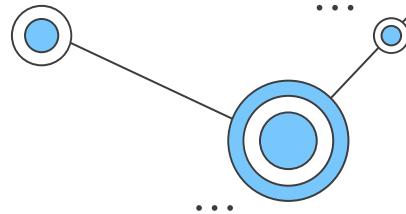
Cuando nos topamos con valores faltantes en nuestros dataset tenemos tres opciones.

- Deshacerse de filas correspondientes.
- Deshacerse todo el atributo. (columna)
- Reemplazar los valores faltantes con otro valor (media, mediana, cero, etc).

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



SimpleImputer



```
from sklearn.impute import SimpleImputer
```

```
imputer = SimpleImputer(strategy="median")
```

Remuevo variables categóricas:

```
housing_num = housing.drop("ocean_proximity", axis=1)
```

“Ajusto” a el dataset housing_num:

```
imputer.fit(housing_num)
```

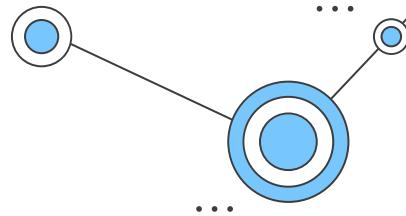
```
>>> imputer.statistics_
array([-118.51, 34.26, 29., 2119.5, 433., 1164., 408., 3.5409])
```

Transformo el dataframe:

```
X = imputer.transform(housing_num)
```

```
housing_tr = pd.DataFrame(X, columns=housing_num.columns,
                           index=housing_num.index)
```

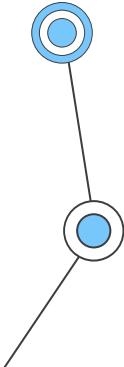
Tratando con texto y variables categóricas



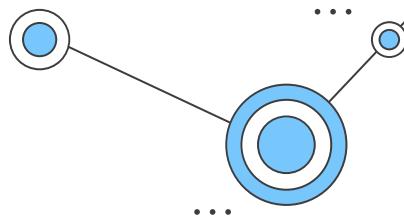
La mayoría de algoritmos de machine learning espera como atributos variables numéricas. Para tratar estos casos se pueden utilizar clases de sklearn en función de la variable categórica con la que estamos tratando.

```
>>> from sklearn.preprocessing import OrdinalEncoder  
>>> ordinal_encoder = OrdinalEncoder()  
>>> housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)  
>>> housing_cat_encoded[:10]  
array([[0.],  
       [0.],  
       [4.],  
       [1.],  
       [0.],  
       [1.],  
       [0.],  
       [1.],  
       [0.],  
       [0.]])
```

```
>>> housing["ocean_proximity"].value_counts()  
<1H OCEAN      9136  
INLAND          6551  
NEAR OCEAN     2658  
NEAR BAY        2290  
ISLAND            5  
Name: ocean_proximity, dtype: int64
```

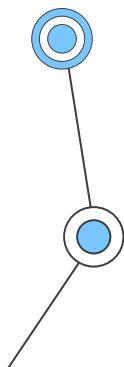


Tratando con texto y variables categóricas



En el caso del ejemplo es una variable categórica (sin orden)

```
>>> from sklearn.preprocessing import OneHotEncoder  
>>> cat_encoder = OneHotEncoder()  
>>> housing_cat_1hot = cat_encoder.fit_transform(housing_cat)  
>>> housing_cat_1hot  
<16512x5 sparse matrix of type '<class 'numpy.float64'>'  
    with 16512 stored elements in Compressed Sparse Row format>  
>>> housing_cat_1hot.toarray()  
array([[1., 0., 0., 0., 0.],  
       [1., 0., 0., 0., 0.],  
       [0., 0., 0., 0., 1.],  
       ...,  
       [0., 1., 0., 0., 0.],  
       [1., 0., 0., 0., 0.],  
       [0., 0., 0., 1., 0.]])  
>>> cat_encoder.categories_  
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],  
      dtype=object)]
```



Transformaciones personalizadas

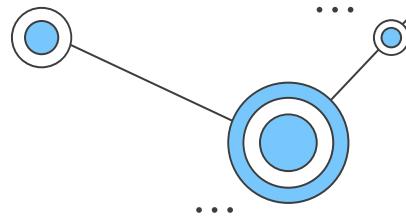
```
from sklearn.base import BaseEstimator, TransformerMixin

rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room = True): # no *args or **kwargs
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self # nothing else to do
    def transform(self, X):
        rooms_per_household = X[:, rooms_ix] / X[:, households_ix]
        population_per_household = X[:, population_ix] / X[:, households_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household,
                        bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]

attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
housing_extra_attribs = attr_adder.transform(housing.values)
```

Escalado de características (Scaler)



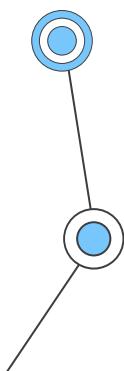
Muchos de los algoritmos esperan características con escalas similares, existen dos modos de escalar las características:

- Utilizando la clase **MinMaxScaler**, esta clase espera un rango de inicialización, por defecto (0, 1), y va a escalar la variable al rango predefinido por las variables. Internamente:

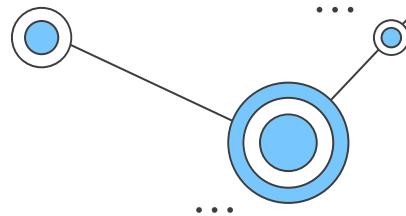
```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
X_scaled = X_std * (max - min) + min
```

- Utilizando la clase **StandardScaler**, va a sustraer el valor medio de la muestra y dividirla por la desviación estándar. Este tipo de escalado es menos sensible a valor atípicos.

$$z = (x - \mu) / s$$



Escalado de características (Scaler)



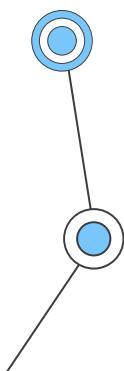
Muchos de los algoritmos esperan características con escalas similares, existen dos modos de escalar las características:

- Utilizando la clase **MinMaxScaler**, esta clase espera un rango de inicialización, por defecto (0, 1), y va a escalar la variable al rango predefinido por las variables. Internamente:

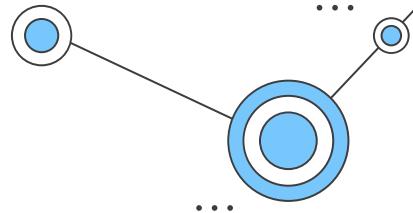
```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
X_scaled = X_std * (max - min) + min
```

- Utilizando la clase **StandardScaler**, va a sustraer el valor medio de la muestra y dividirla por la desviación estándar. Este tipo de escalado es menos sensible a valor atípicos.

$$z = (x - \mu) / s$$



Pipelines de transformación



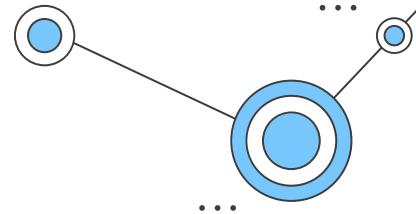
Cuando se realizan múltiples transformaciones sobre las mismas columnas utilizamos la clase Pipeline

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    ('atribus_adder', CombinedAttributesAdder()),
    ('std_scaler', StandardScaler()),
])

housing_num_tr = num_pipeline.fit_transform(housing_num)
```

Uniendo múltiples Pipelines



Para unir múltiples pipelines que operan sobre distintas características utilizamos ColumnTransformer

```
from sklearn.compose import ColumnTransformer

num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]

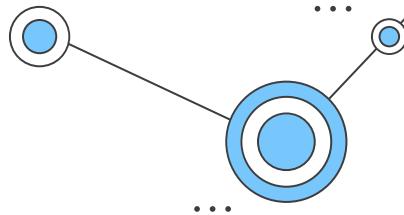
full_pipeline = ColumnTransformer([
    ("num", num_pipeline, num_attribs),
    ("cat", OneHotEncoder(), cat_attribs),
])

housing_prepared = full_pipeline.fit_transform(housing)
```

Nota: al momento de realizar la union de los output de los dos pipelines, se mide la densidad de ceros en la matriz resultante para saber si devolver una matriz densa o un sparse_matrix

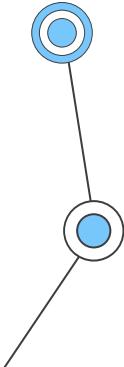
Seleccionar el modelo y entrenarlo

Entrenando el modelo

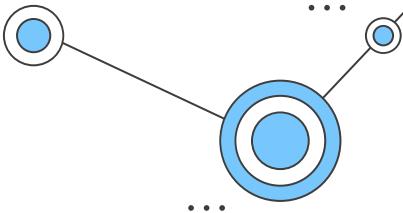


Una vez realizado todos los pasos anteriores solo queda elegir el modelo qué vamos a utilizar. Podríamos empezar por probar un modelo simple como es una regresión lineal.

```
from sklearn.linear_model import LinearRegression  
  
lin_reg = LinearRegression()  
lin_reg.fit(housing_prepared, housing_labels)
```



Entrenando el modelo

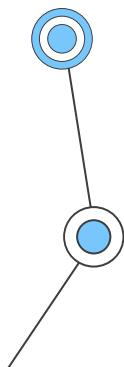


Una vez realizado todos los pasos anteriores solo queda elegir el modelo qué vamos a utilizar. Podríamos empezar por probar un modelo simple como es una regresión lineal.

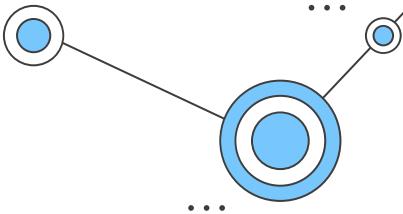
```
from sklearn.linear_model import LinearRegression
```

```
lin_reg = LinearRegression()  
lin_reg.fit(housing_prepared, housing_labels)
```

```
>>> some_data = housing.iloc[:5]  
>>> some_labels = housing_labels.iloc[:5]  
>>> some_data_prepared = full_pipeline.transform(some_data)  
>>> print("Predictions:", lin_reg.predict(some_data_prepared))  
Predictions: [ 210644.6045  317768.8069  210956.4333  59218.9888  189747.5584]  
>>> print("Labels:", list(some_labels))  
Labels: [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]
```



Entrenando el modelo

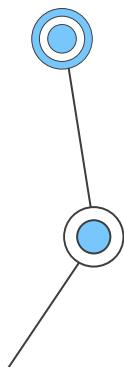


Una vez realizado todos los pasos anteriores solo queda elegir el modelo qué vamos a utilizar. Podríamos empezar por probar un modelo simple como es una regresión lineal.

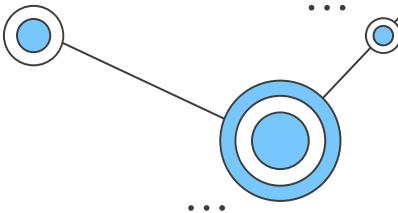
```
from sklearn.linear_model import LinearRegression  
  
lin_reg = LinearRegression()  
lin_reg.fit(housing_prepared, housing_labels)
```

```
>>> from sklearn.metrics import mean_squared_error  
>>> housing_predictions = lin_reg.predict(housing_prepared)  
>>> lin_mse = mean_squared_error(housing_labels, housing_predictions)  
>>> lin_rmse = np.sqrt(lin_mse)  
>>> lin_rmse  
68628.19819848922
```

Teniendo en cuenta qué los precios de la casas van entre 120000 y 265000 la error en la estimación es bastante grande



Entrenando el modelo



Podríamos estar frente a un caso de underfitting, podríamos probar un modelo más complejo. Probemos con DesitionTreeRegressor

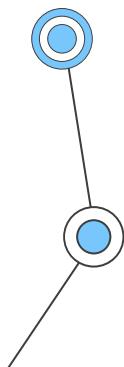
```
from sklearn.tree import DecisionTreeRegressor
```

```
tree_reg = DecisionTreeRegressor()  
tree_reg.fit(housing_prepared, housing_labels)
```

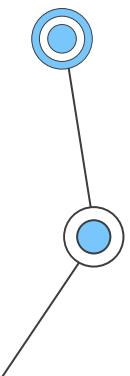
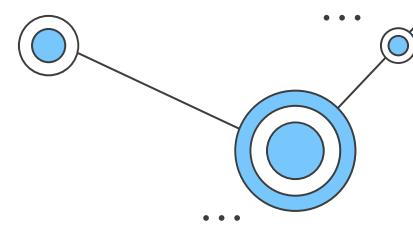
Evaluamos:

```
>>> housing_predictions = tree_reg.predict(housing_prepared)  
>>> tree_mse = mean_squared_error(housing_labels, housing_predictions)  
>>> tree_rmse = np.sqrt(tree_mse)  
>>> tree_rmse  
0.0
```

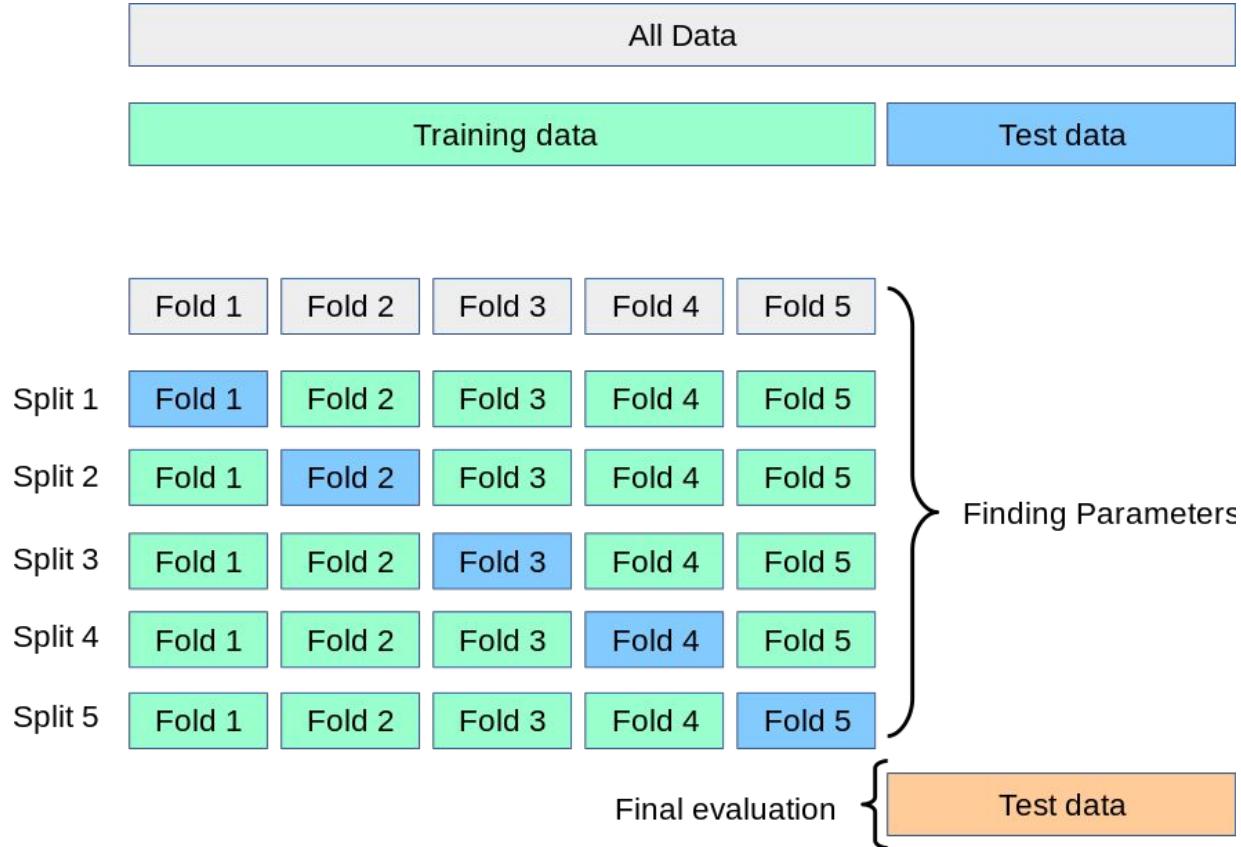
En este caso, podríamos estar frente a un caso overfitting



Validación cruzada



Validación cruzada



Validación cruzada

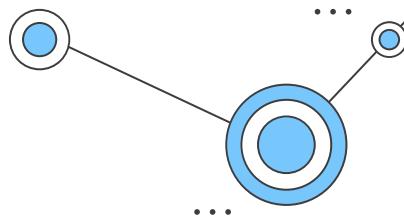
```
: from sklearn.model_selection import cross_val_score  
  
scores = cross_val_score(tree_reg, housing_prepared, housing_labels,  
                        scoring="neg_mean_squared_error", cv=10)  
tree_rmse_scores = np.sqrt(-scores)
```

```
:  
def display_scores(scores):  
    print("Scores:", scores)  
    print("Mean:", scores.mean())  
    print("Standard deviation:", scores.std())  
  
display_scores(tree_rmse_scores)
```

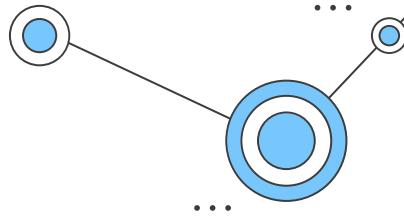
```
Scores: [70194.33680785 66855.16363941 72432.58244769 70758.73896782  
71115.88230639 75585.14172901 70262.86139133 70273.6325285  
75366.87952553 71231.65726027]  
Mean: 71407.68766037929  
Standard deviation: 2439.4345041191004
```

```
: lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels,  
                           scoring="neg_mean_squared_error", cv=10)  
lin_rmse_scores = np.sqrt(-lin_scores)  
display_scores(lin_rmse_scores)
```

```
Scores: [66782.73843989 66960.118071 70347.95244419 74739.57052552  
68031.13388938 71193.84183426 64969.63056405 68281.61137997  
71552.91566558 67665.10082067]  
Mean: 69052.46136345083  
Standard deviation: 2731.674001798342
```



Validación cruzada



```
:     from sklearn.model_selection import cross_val_score  
  
    scores = cross_val_score(tree_reg, housing_prepared, housing_labels,  
                            scoring="neg_mean_squared_error", cv=10)  
    tree_rmse_scores = np.sqrt(-scores)
```

```
:  
def display_scores(scores):  
    print("Scores:", scores)  
    print("Mean:", scores.mean())  
    print("Standard deviation:", scores.std())  
  
display_scores(tree_rmse_scores)
```

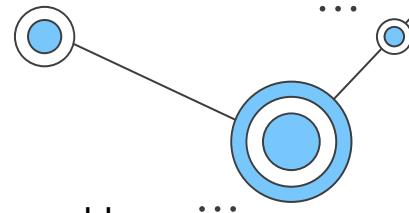
```
Scores: [70194.33680785 66855.16363941 72432.58244769 70758.73896782  
71115.88230639 75585.14172901 70262.86139133 70273.6325285  
75366.87952553 71231.65726027]  
Mean: 71407.68766037929  
Standard deviation: 2439.4345041191004
```

```
:  
lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels,  
                           scoring="neg_mean_squared_error", cv=10)  
lin_rmse_scores = np.sqrt(-lin_scores)  
display_scores(lin_rmse_scores)
```

```
Scores: [66782.73843989 66960.118071 70347.95244419 74739.57052552  
68031.13388938 71193.84183426 64969.63056405 68281.61137997  
71552.91566558 67665.10082067]  
Mean: 69052.46136345083  
Standard deviation: 2731.674001798342
```

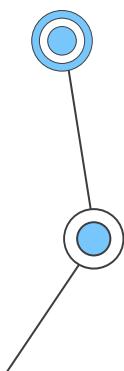


Validación cruzada



Probemos un modelo qué cae dentro de la categoría de algoritmos de ensamble RandomForestRegressor. Los algoritmos de ensamble se construyen en base a algoritmos más simples. En este caso RandomForest se construye en base a múltiples DesitionTree

```
>>> from sklearn.ensemble import RandomForestRegressor  
>>> forest_reg = RandomForestRegressor()  
>>> forest_reg.fit(housing_prepared, housing_labels)  
>>> [...]  
>>> forest_rmse  
18603.515021376355  
>>> display_scores(forest_rmse_scores)  
Scores: [49519.80364233 47461.9115823 50029.02762854 52325.28068953  
49308.39426421 53446.37892622 48634.8036574 47585.73832311  
53490.10699751 50021.5852922 ]  
Mean: 50182.303100336096  
Standard deviation: 2097.0810550985693
```

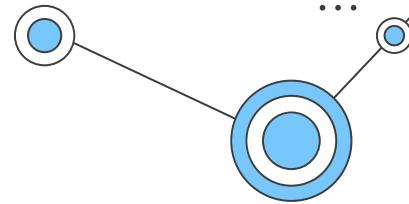


Nota, buenas prácticas

Una buena práctica es ir guardando los modelos que vamos generando para poder más tarde compararlos.

```
import joblib  
  
joblib.dump(my_model, "my_model.pkl")  
# and later...  
my_model_loaded = joblib.load("my_model.pkl")
```

Fine-tuning del modelo



Finalmente una vez qué acotó la lista de modelos qué mejor se ajustan a los datos, se necesita hallar los hiperparámetros óptimos del modelo. Scikit learn ofrece algunos métodos para encontrarlos.

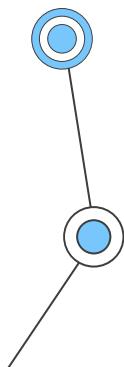
```
from sklearn.model_selection import GridSearchCV

param_grid = [
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
]

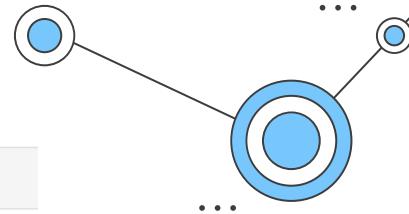
forest_reg = RandomForestRegressor()

grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)

grid_search.fit(housing_prepared, housing_labels)
```



Fine-tuning del modelo



```
grid_search.best_params_
{'max_features': 8, 'n_estimators': 30}

grid_search.best_estimator_
RandomForestRegressor(max_features=8, n_estimators=30, random_state=42)
```

Let's look at the score of each hyperparameter combination tested during the grid search:

```
cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)

63669.11631261028 {'max_features': 2, 'n_estimators': 3}
55627.099719926795 {'max_features': 2, 'n_estimators': 10}
53384.57275149205 {'max_features': 2, 'n_estimators': 30}
60965.950449450494 {'max_features': 4, 'n_estimators': 3}
52741.04704299915 {'max_features': 4, 'n_estimators': 10}
50377.40461678399 {'max_features': 4, 'n_estimators': 30}
58663.93866579625 {'max_features': 6, 'n_estimators': 3}
52006.19873526564 {'max_features': 6, 'n_estimators': 10}
50146.51167415009 {'max_features': 6, 'n_estimators': 30}
57869.25276169646 {'max_features': 8, 'n_estimators': 3}
51711.127883959234 {'max_features': 8, 'n_estimators': 10}
49682.273345071546 {'max_features': 8, 'n_estimators': 30}
62895.06951262424 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
54658.176157539405 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
59470.40652318466 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
52724.9822587892 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
57490.5691951261 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
51009.495668875716 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```



Fine-tuning del modelo

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

param_distributions = {
    'n_estimators': randint(low=1, high=200),
    'max_features': randint(low=1, high=8),
}

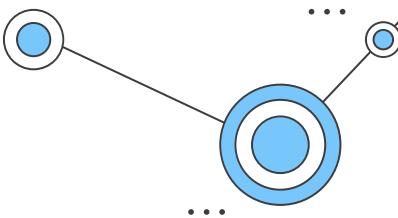
forest_reg = RandomForestRegressor(random_state=42)
rnd_search = RandomizedSearchCV(forest_reg, param_distributions=param_distributions,
                                 n_iter=10, cv=5, scoring='neg_mean_squared_error', random_state=42)
rnd_search.fit(housing_prepared, housing_labels)
```

```
RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(random_state=42),
                    param_distributions={'max_features': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7fd1b96682d0>,
                                         'n_estimators': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7fd1b9668b10>},
                    random_state=42, scoring='neg_mean_squared_error')
```

```
cvres = rnd_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
49150.70756927707 {'max_features': 7, 'n_estimators': 180}
51389.889203389284 {'max_features': 5, 'n_estimators': 15}
50796.155224308866 {'max_features': 3, 'n_estimators': 72}
50835.13360315349 {'max_features': 5, 'n_estimators': 21}
49280.9449827171 {'max_features': 7, 'n_estimators': 122}
50774.90662363929 {'max_features': 3, 'n_estimators': 75}
50682.78888164288 {'max_features': 3, 'n_estimators': 88}
49608.99608105296 {'max_features': 5, 'n_estimators': 100}
50473.61930350219 {'max_features': 3, 'n_estimators': 150}
64429.84143294435 {'max_features': 5, 'n_estimators': 2}
```

Evaluar el sistema en el test set



```
final_model = grid_search.best_estimator_
X_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].copy()

X_test_prepared = full_pipeline.transform(X_test)
final_predictions = final_model.predict(X_test_prepared)

final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
```

```
final_rmse
```

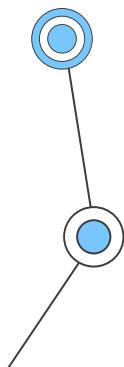
```
47730.22690385927
```

We can compute a 95% confidence interval for the test RMSE:

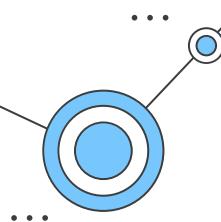
```
from scipy import stats

confidence = 0.95
squared_errors = (final_predictions - y_test) ** 2
np.sqrt(stats.t.interval(confidence, len(squared_errors) - 1,
                        loc=squared_errors.mean(),
                        scale=stats.sem(squared_errors)))
```

```
array([45685.10470776, 49691.25001878])
```

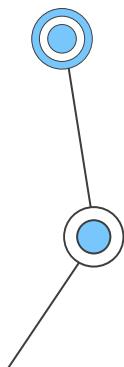


Presentación de resultados finales

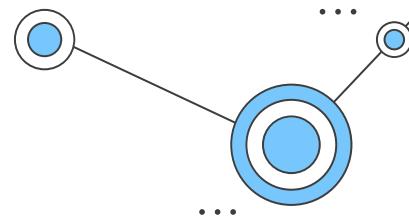


Una vez terminado la fase de prelanzamiento, es necesario:

- presentar la solución propuesta (señalar qué funcionó y qué no, qué suposiciones se realizaron, cuales son las limitaciones),
- documentar todo se realizó
- Crear visualizaciones
- Conclusión o comentarios finales (el sistema no mejora a las estimaciones de los expertos, pero podría utilizarse para liberarlos de esta tarea).



Algunas recomendaciones para presentar resultados



- Si se muestra un gráfico o tabla, explicar qué contiene.
- Numerar las diapositivas.
- Entender a quien va dirigida la presentación
- Si la presentación está dirigida al equipo técnico, puede ser buena idea presentar en un jupyter notebook qué te permite poner texto, código y visualizaciones.

