End to End Machine Learning project

¿Cómo realizo mi proyecto de Machine Learning?

Trabajando con datos reales

- Un ejemplo que nos permite saber cómo llevar a cabo un proyecto de Machine Learning
- Necesario seguir una serie de pasos
- Datos reales y no solo datos artificiales
- Datos de repositorios
- Datos de sus empresas

aws

UCI Machine Learning Repository

kaggle

OPEN DATA MONITOR

Pasos principales para realizar un proyecto de Machine Learning

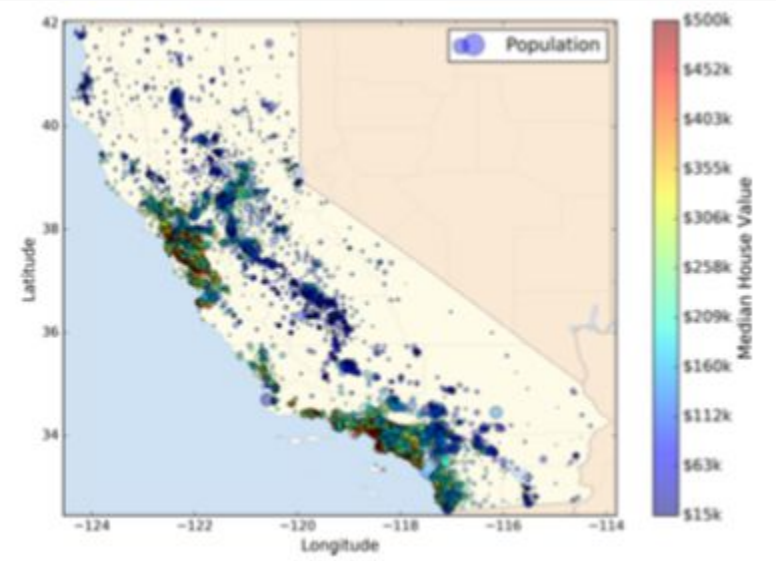
1. Observar el problema completo
2. Conseguir los datos
3. Descubrir y visualizar los datos para ganar percepción
4. Preparar los datos para los algoritmos de Machine Learning
5. Seleccionar y entrenar el modelo
6. Ajustar el modelo
7. Lanzar, monitorear y mantener el sistema

Ejemplo

Para ejemplificar este tema, se toma el problema y dataset de *the California Housing Price*.

Se desea predecir el precio medio de una casa de acuerdo a sus características y al historial de otras casas.

Siguiendo los pasos...



1. Observar el problema completo

- Enmarcar el problema:

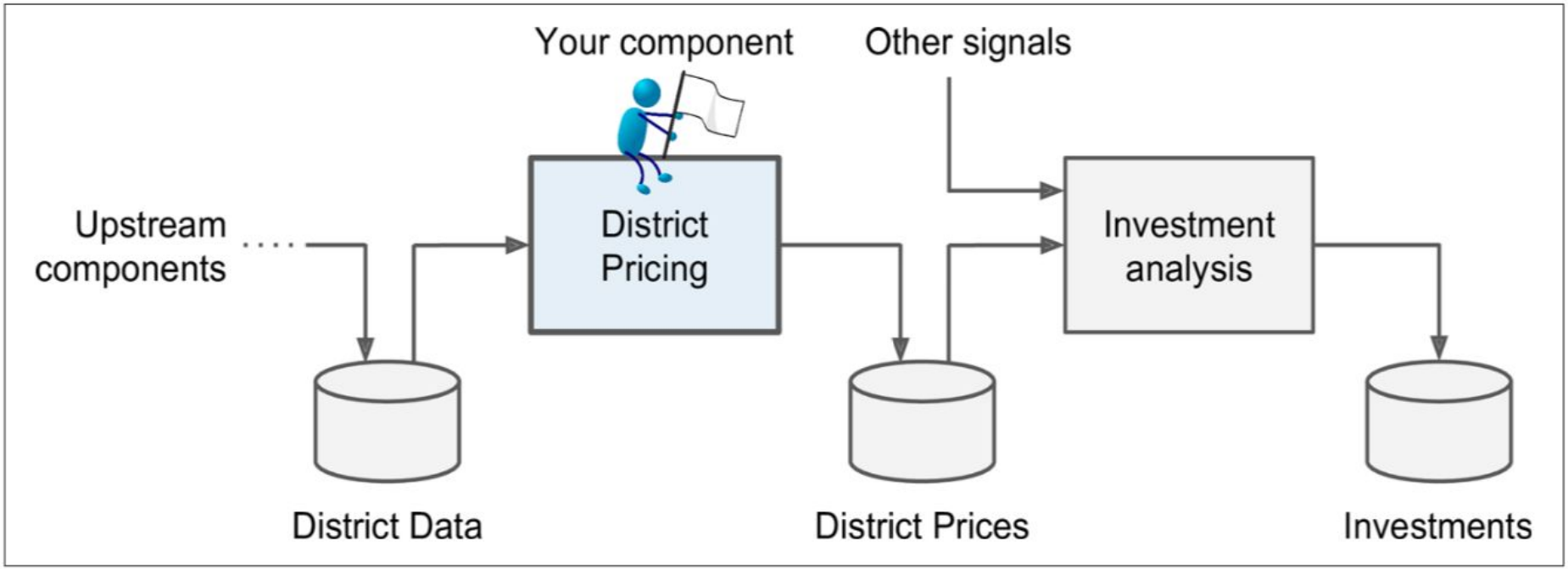
  ¿Cúal es exactamente el objetivo del proyecto?

  ¿Qué beneficio se espera de hacer un modelo ML?

  La respuesta a estas preguntas, es la salida de nuestro modelo.

- Seleccionar medida de desempeño
- Revisar suposiciones

Enmarcar el problema



Enmarcar el problema

- Conocer si hay guías para comparar desempeño
- Encontrar margen de error típico al predecir
- ¿Qué tipo de problema es?

  Tip : El aprendizaje supervisado tiene ***etiquetas***
- ¿Batch o Online?

Volviendo a nuestro problema:

Media housing prices, es aprendizaje supervisado de regresión multivariable.

Seleccionar medida de desempeño

- Medir que tan buena es la salida de nuestro sistema
- Comparar lo que se está prediciendo con los resultados reales
- Métricas para medir la distancia entre dos vectores, predicción y salida real

Error cuadrático medio (Root Mean Square Error), mide la desviación estándar

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

Error absoluto medio (Mean Absolute Error)

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

Revisar suposiciones

- Enlistar y verificar las suposiciones que se hacen
- Tener conciencia de que el sistema entregará lo que necesitamos

Si todo es correcto, ¡podemos comenzar!

2. Conseguir los datos

- Crear el espacio de trabajo
- Descargar los datos
- Observar la estructura de los datos
- Crear conjuntos de prueba

Crear el espacio de trabajo

Se crea el directorio para el proyecto

```
$ export ML_PATH="$HOME/ml"      # You can change the path if you prefer
$ mkdir -p $ML_PATH
```

Actualizar el módulo pip para instalar los demás módulos necesarios

```
$ pip3 install --upgrade pip
```

Como buena práctica se trabaja en un ambiente aislado

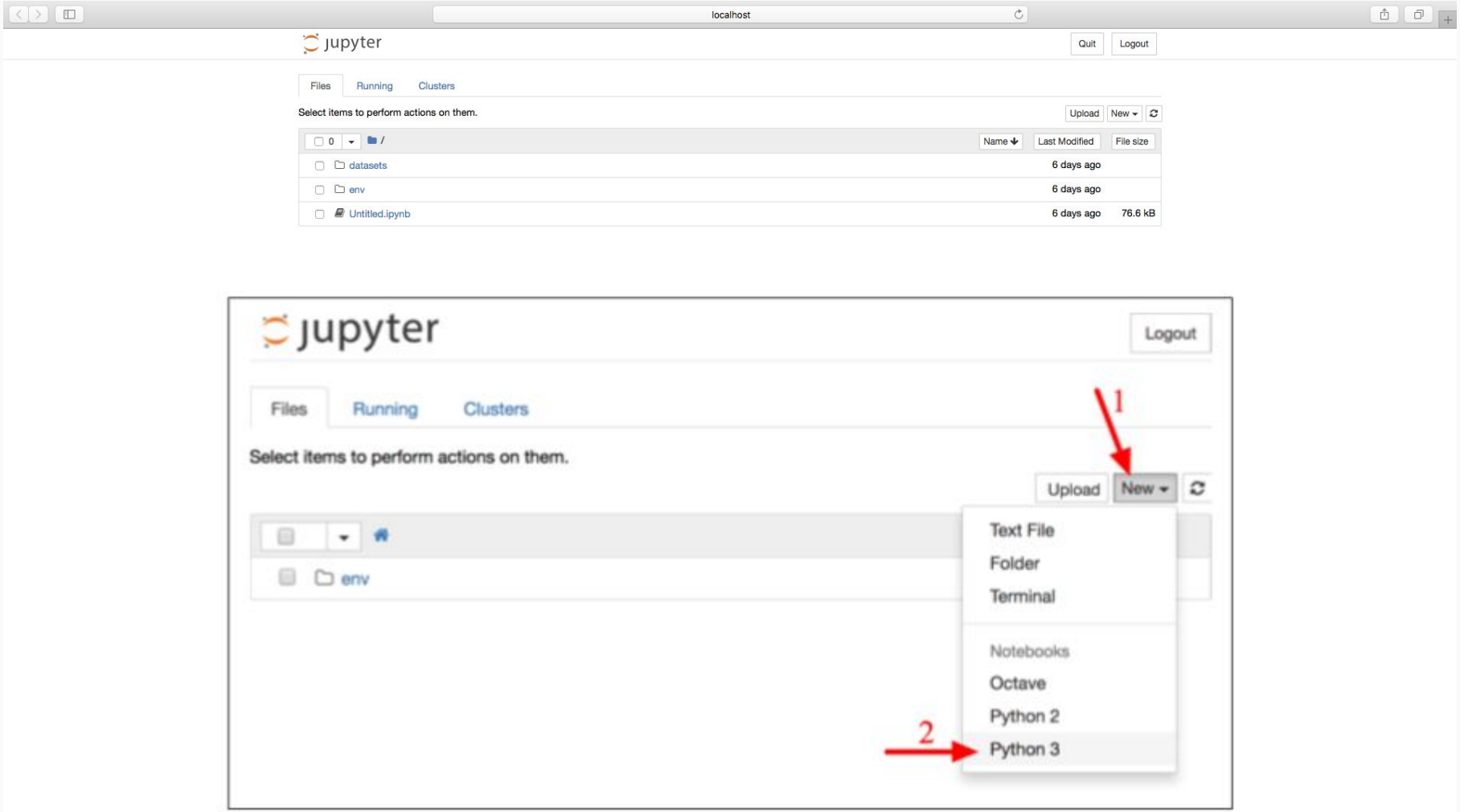
```
$ pip3 install --user --upgrade virtualenv
$ cd $ML_PATH
$ virtualenv env
$ cd $ML_PATH
$ source env/bin/activate
```

Instalar los módulos necesarios, como Panda, Jupyter, Numpy, Scikit… etc

```
$ pip3 install --upgrade jupyter matplotlib numpy pandas scipy scikit-learn
$ python3 -c "import jupyter, matplotlib, numpy, pandas, scipy, sklearn"
```

Iniciar Jupyter

```
$ jupyter notebook
```


localhost
jupyter
Quit
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
Name
Last Modified
File size
datasets
6 days ago
env
6 days ago
Untitled.ipynb
6 days ago
76.6 kB
jupyter
Logout
Files
Running
Clusters
Select items to perform actions on them.
1
Upload
New
Text File
Folder
Terminal
Notebooks
Octave
Python 2
2
Python 3
env

Descargar los datos

Dos opciones:

- Descargar los datos de forma manual
- Crear una función que los descargue por nosotros

Para nuestro ejemplo, se usa el dataset que está en la dirección:

https://raw.githubusercontent.com/ageron/handson-ml/master/datasets/housing/housing.tgz En particular el archivo *housing.csv*

En el archivo de Jupyter se crea la función para descargar el dataset

In [1]:
```
import os
import tarfile
from six.moves import urllib

DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml/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):
    if not os.path.isdir(housing_path):
        os.makedirs(housing_path)
    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()
```

In [3]:
```
fetch_housing_data()
```

Descargar los datos

Ahora se cargan los datos para poder utilizarlos a través de Pandas, con la siguiente función

In [4]:
```
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)
```

Esta función crea un *DataFrame* de Pandas para trabajar en Python con la información contenida en el archivo *.csv*

Observar la estructura de los datos

Observaremos la estructura de los primeros 5 datos almacenados

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

Out[5]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

Notamos que cada instancia (renglón), tiene 10 atributos (columnas)

Observar la estructura de los datos

```
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
```

Observar la estructura de los datos

Conocer el número de instancias con el mismo atributo:

```
In [7]: housing["ocean_proximity"].value_counts()

Out[7]: <1H OCEAN     9136
        INLAND        6551
        NEAR OCEAN    2658
        NEAR BAY      2290
        ISLAND           5
        Name: ocean_proximity, dtype: int64
```

Para conocer las propiedades de cada atributo:

```
In [9]: housing.describe()
```

Out[9]:

click to expand output; double click to hide output

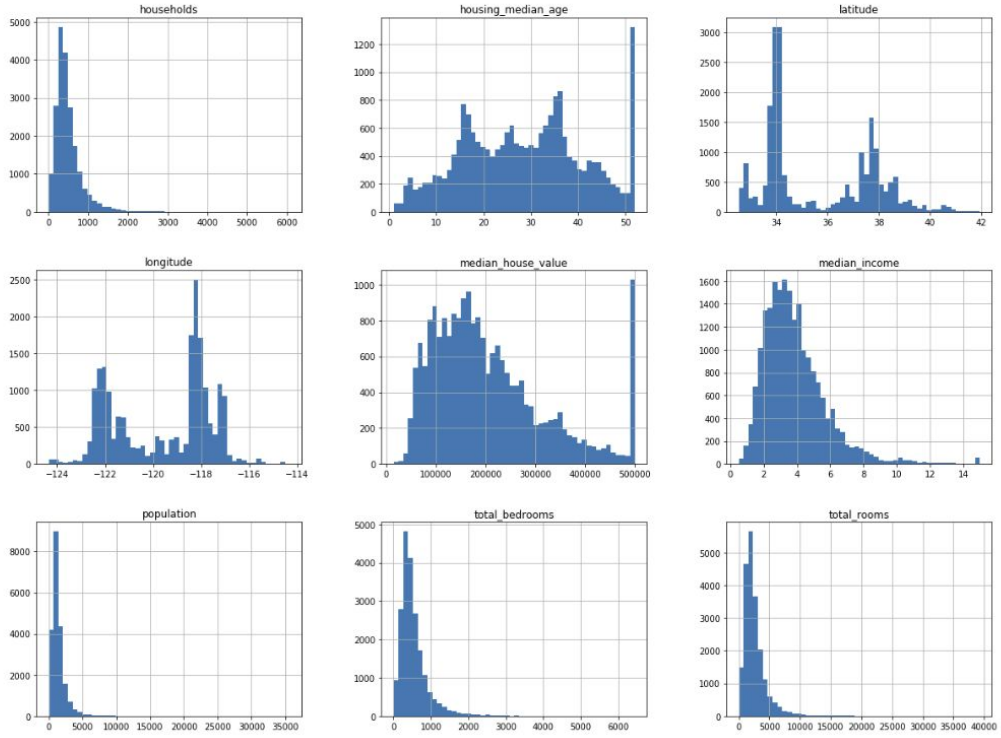
	atitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	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

Observar la estructura de los datos

Otra forma de visualizar el tipo de datos es a través de los histogramas

In [10]:
```
%matplotlib inline

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



Crear conjuntos de prueba

Consiste en separar un conjunto de datos del dataset completo

Evitar: **Overfitting**, volverse muy optimista en la estimación

Para crear un conjunto de prueba, se elige aleatoriamente el 20% de las instancias de nuestro dataset

In [11]:
```
import numpy as np
```

In [12]:
```
def split_train_test(data, test_ratio):
    np.random.seed(42)
    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]
```

In [13]:
```
train_set, test_set = split_train_test(housing, 0.2)
print(len(train_set), "train +", len(test_set), "test")
```

¿Por qué es una buena técnica para elegir aleatoriamente el conjunto de prueba y cuando va a fallar?

Crear conjuntos de prueba

- Uso de identificadores únicos de cada instancia.
  - Cálculo del *hash* a cada instancia.

Función para calcular el hash de las instancias:

In [14]:
```
import hashlib

def test_set_check (identifier, test_ratio, hash):
    return hash(np.int64(identifier)).digest()[-1] < 256*test_ratio

def split_train_test_by_id(data, test_ratio, id_column, hash=hashlib.md5):
    ids = data[id_column]
    in_test_set = ids.apply(lambda id_: test_set_check(id_, test_ratio, hash))
    return data.loc[~in_test_set], data.loc[in_test_set]
```

Dado que nuestras instancias no tienen un identificador, es necesario crearlo

In [15]:
```
housing_with_id = housing.reset_index()
train_set, test_set = split_train_test_by_id(housing_with_id, 0.2, "index")
```

Crear conjuntos de prueba

¿Cómo luce ahora nuestro dataset?

```
In [16]: housing_with_id.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 11 columns):
index                 20640 non-null int64
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), int64(1), object(1)
memory usage: 1.7+ MB
```

Cuidar que no haya duplicidad
Nuevas instancias al final del dataset para conservar identificadores constantes
Buscar un identificador estable

```
In [17]: housing_with_id["id"] = housing["longitude"] * 1000 + housing["latitude"]
         train_set, test_set = split_train_test_by_id(housing_with_id, 0.2, "id")
```

Crear conjuntos de prueba

Scikit-Learn provee funciones para dividir datasets en múltiples conjuntos de diferentes formas.

Una forma muy simple es la función **train_test_split,** que devuelve el conjunto de entrenamiento y el conjunto de prueba.

```
In [18]: from sklearn.model_selection import train_test_split
         train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)

In [19]: train_set.info()
         test_set.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16512 entries, 14196 to 15795
Data columns (total 10 columns):
longitude             16512 non-null float64
latitude              16512 non-null float64
housing_median_age    16512 non-null float64
total_rooms           16512 non-null float64
total_bedrooms        16512 non-null float64
population            16512 non-null float64
households            16512 non-null float64
median_income         16512 non-null float64
median_house_value    16512 non-null float64
ocean_proximity       16512 non-null object
dtypes: float64(9), object(1)
memory usage: 1.4+ MB
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4128 entries, 20046 to 3665
Data columns (total 10 columns):
longitude             4128 non-null float64
latitude              4128 non-null float64
housing_median_age    4128 non-null float64
total_rooms           4128 non-null float64
total_bedrooms        3921 non-null float64
population            4128 non-null float64
households            4128 non-null float64
median_income         4128 non-null float64
median_house_value    4128 non-null float64
ocean_proximity       4128 non-null object
dtypes: float64(9), object(1)
memory usage: 354.8+ KB
```

Crear conjuntos de prueba

Contar con un número grande de instancias para cada estrato que es de importancia

En caso de tener pocos datos, se generan categorías donde múltiples instancias caen en una misma categoría

```
In [20]: housing["income_cat"] = np.ceil(housing["median_income"] / 1.5)
         housing["income_cat"].where(housing["income_cat"] < 5, 5.0, inplace=True)
```

Out[24]:

gitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity	income_cat
122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY	5.0
122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY	5.0
122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY	5.0
122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY	4.0
122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY	3.0

Crear conjuntos de prueba

Las categorías permiten que el muestreo sea estratificado para valores continuos.
Nuevamente usando Scikit-Learn

```
In [25]: 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]
```

Visualizando la proporción por categoría en el dataset

```
In [26]: housing["income_cat"].value_counts() / len(housing)

Out[26]: 3.0    0.350581
         2.0    0.318847
         4.0    0.176308
         5.0    0.114438
         1.0    0.039826
         Name: income_cat, dtype: float64
```

Ejercicio: Eliminar el atributo ***income_cat*** para volver a nuestros datos originales

3. Descubrir y visualizar los datos para generar percepción

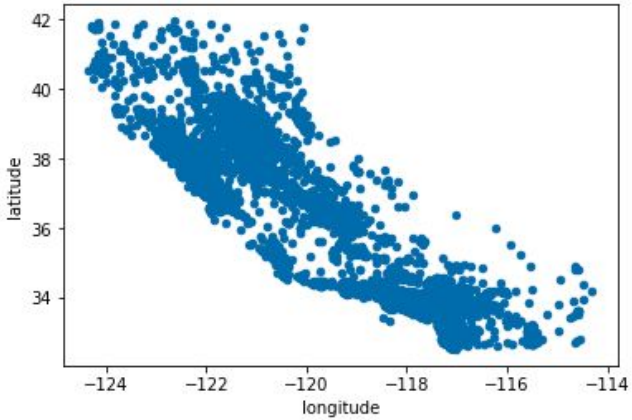
Una vez que se tiene una idea general de los datos, es tiempo de profundizar en ellos.

- Visualizar datos geográficos
- Buscar correlaciones
- Experimentar con combinación de atributos

Visualizar datos geográficos

Observar donde se encuentran los datos con los que se trabaja

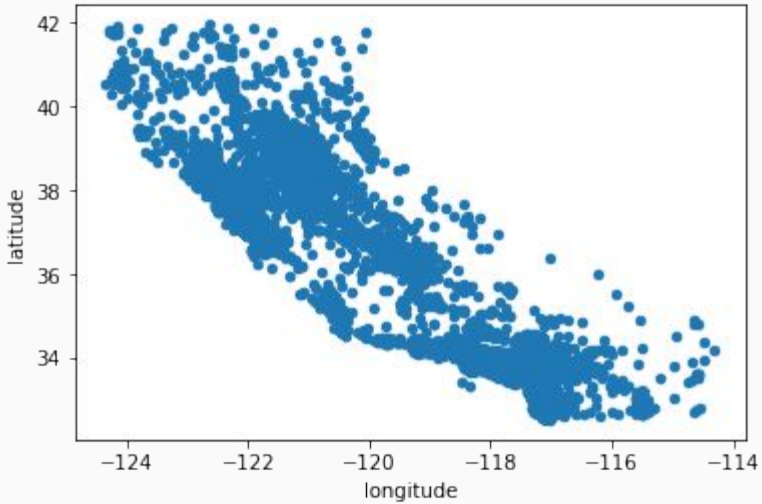
```
In [69]: housing.plot(kind="scatter", x="longitude", y="latitude")
Out[69]: <matplotlib.axes._subplots.AxesSubplot at 0x13096a588>
```

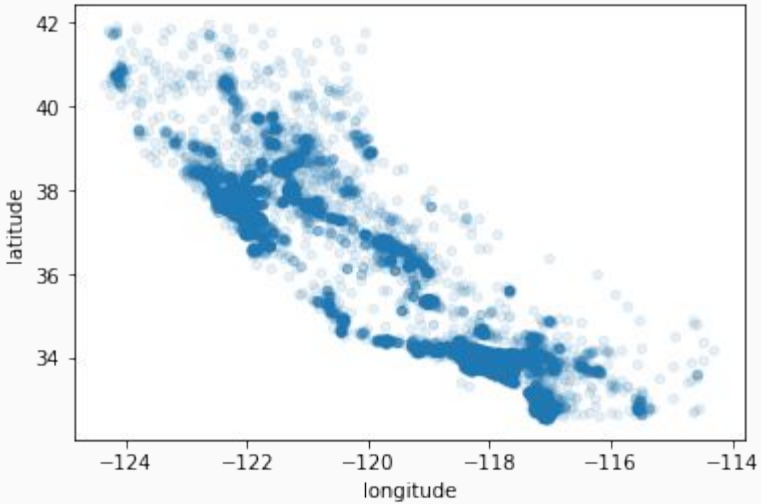


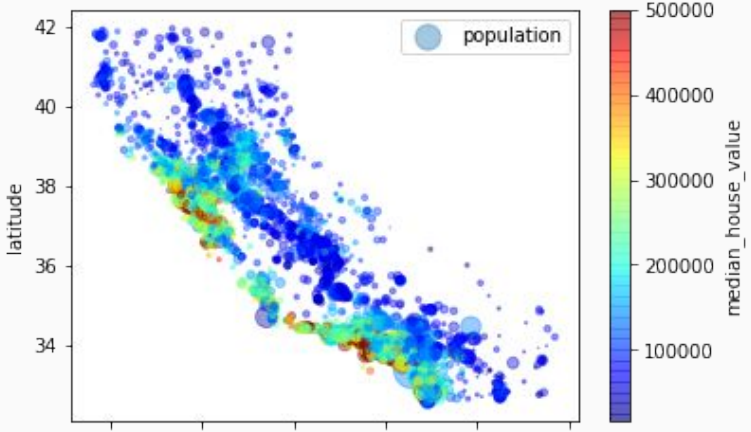
Ver los datos de forma más clara

```
In [70]: housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
```

```
In [28]: housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4, s=housing["population"]/100, label="population",
                      c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True,
                      )
         plt.legend()
```


latitude
longitude
42
40
38
36
34
−124
−122
−120
−118
−116
−114


latitude
longitude
42
40
38
36
34
−124
−122
−120
−118
−116
−114


population
latitude
42
40
38
36
34
500000
400000
300000
200000
100000
median_house_value

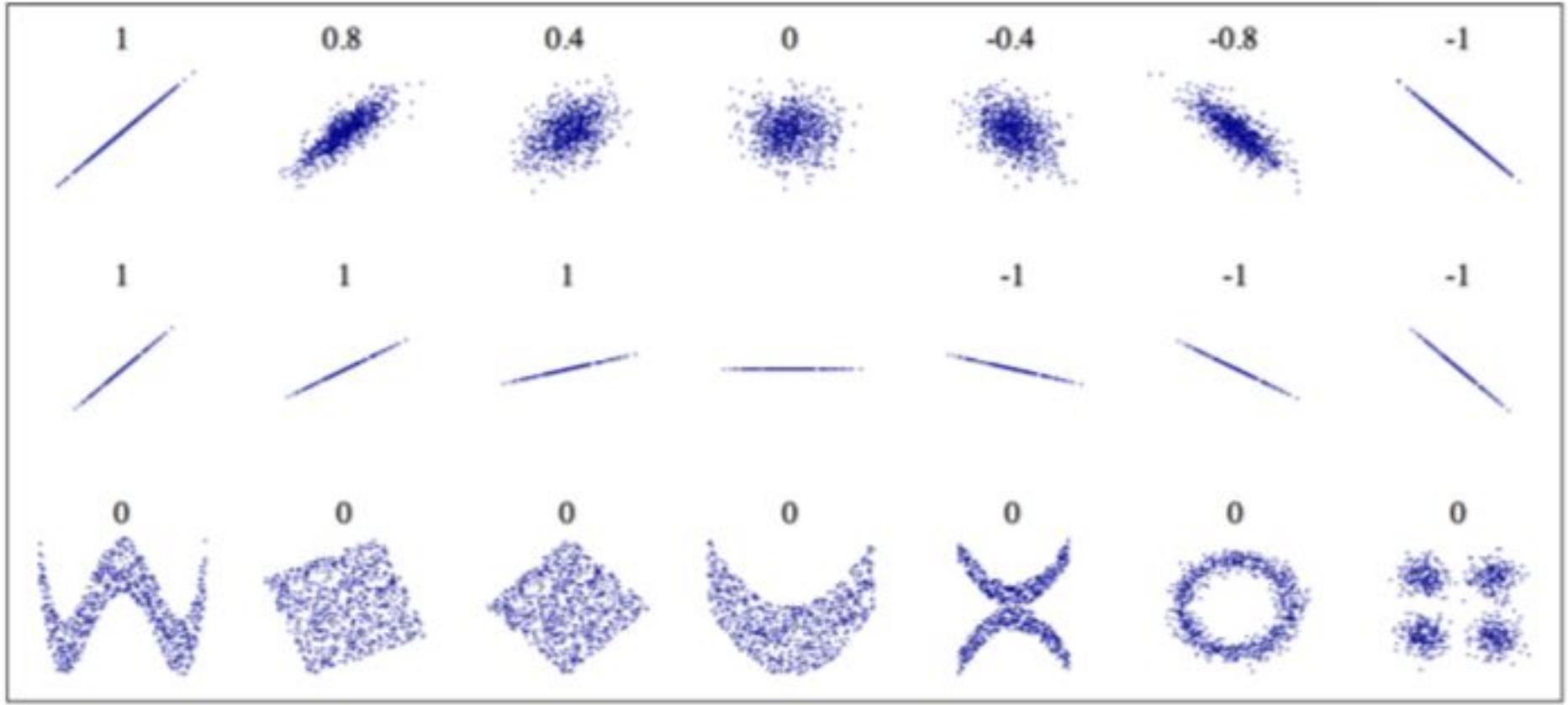
Buscar correlaciones

En conjuntos de datos pequeños es posible hacer correlación a pares de cada atributo

```
In [29]: corr_matrix = housing.corr()
```

Los coeficientes de correlación sólo mide correlación lineal

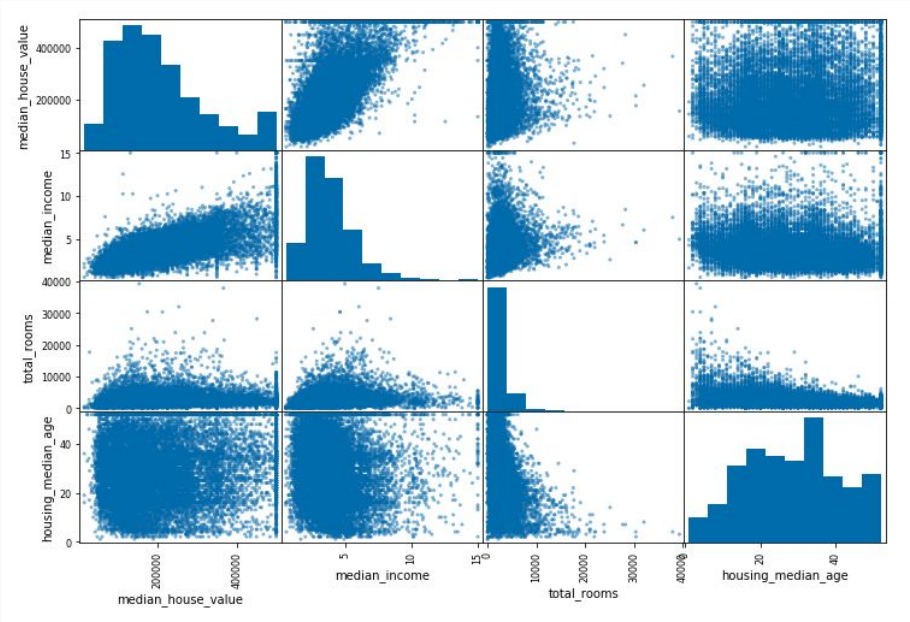
- Entrega valores entre 1 y -1
  - **Correlación 1** significa una fuerte correlación positiva
  - **Correlación -1** significa una fuerte correlación negativa
- Si hay correlación 0, realmente significa que no hay correlación


1
0.8
0.4
0
-0.4
-0.8
-1
1
1
1
-1
-1
-1
0
0
0
0
0
0
0

Buscar correlaciones

Pandas brinda la función ***scatter_matrix*** que permite ver la correlación de atributos

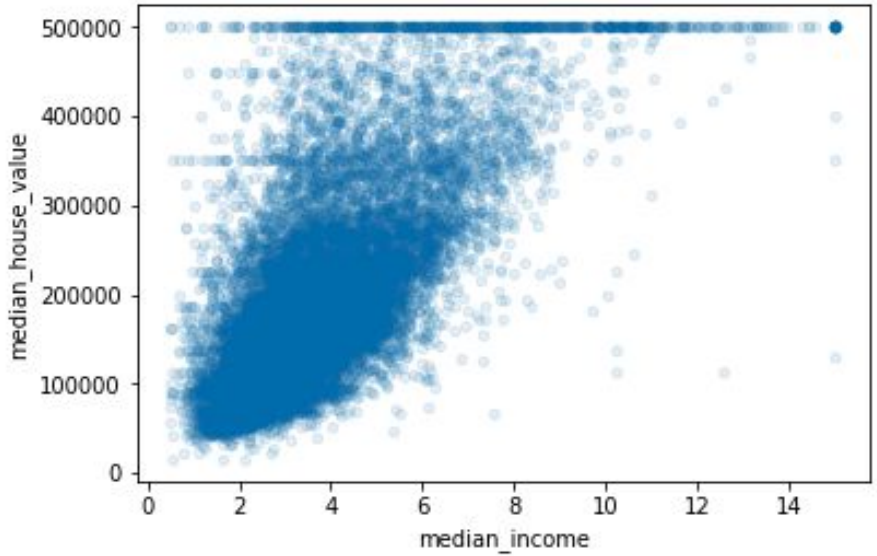
```
In [33]: #from pandas.tools.plotting import scatter_matrix  conflicto con conda
         from pandas.plotting import scatter_matrix
         attributes = ["median_house_value", "median_income", "total_rooms", "housing_median_age"]
         scatter_matrix(housing[attributes], figsize=(12, 8))
```



Buscar correlaciones

El atributo más prometedor para predecir el valor medio de la casa es ***median_income***

```
In [34]: housing.plot(kind="scatter", x="median_income", y="median_house_value", alpha=0.1)
Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0x11c32a438>
```



Experimentar con combinación de atributos

- Atributos por sí solos pueden no dar suficiente información
- Transformar ciertos atributos, incrementa la correlación con nuestro objetivo
- Esta combinación dependerá y cambiará en cada proyecto
- Es una forma de preparar los datos para los algoritmos de Machine learning
- Probar con algunas combinaciones:
  rooms_per_household
  bedrooms_per_rom
  population_per_household

In [35]:
```
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"]
```

4. Preparar los datos para los algoritmos de Machine Learning

- Limpieza de datos
- Manejo de textos y atributos categóricos
- Transformadores personalizados
- Escalado de características
- Tuberías de transformación

Primer paso es separar los atributos de las etiquetas

```
In [37]: housing = strat_train_set.drop("median_house_value", axis=1)
         housing_labels = strat_train_set["median_house_value"].copy()
```

Limpieza de datos

La mayoría de los algoritmos de ML no trabajan con datos faltantes o perdidos, para solucionar este problema hay tres opciones:

- Deshacerse de los distritos correspondiente (estancias)
- Deshacerse del atributo
- Fijar el valor a algún valor (cero, la media, la mediana… etc.)

```
In [55]: 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"]= housing["total_bedrooms"].fillna(median)
```

Scikit-Learn provee una clase que hace esta acción: **Imputer**

- Solo trabaja con valores numéricos
- Permite trabajar con diferentes estrategias (media, mediana o una constante)
- Se elige la media

```
In [71]: from sklearn.preprocessing import Imputer
         imputer = Imputer(strategy="median")
```

```
In [72]: imputer.fit(housing_num)
```

Manejo de textos y atributos categóricos

- En mensajes de texto no es posible calcular su media
- Preciso convertir texto a números para trabajar en ML
- Scikit-Learn provee un transformador llamado LabelEncoder

```
In [77]: from sklearn.preprocessing import LabelEncoder
         encoder = LabelEncoder()
         housing_cat = housing["ocean_proximity"]
         housing_cat_encoded = encoder.fit_transform(housing_cat)
         housing_cat_encoded
         print(encoder.classes_)

         ['<1H OCEAN' 'INLAND' 'ISLAND' 'NEAR BAY' 'NEAR OCEAN']
```

Una solución que permite diferenciar categorías con mejor distancia entre ellas, es el uso de atributos binarios.

Scikit-Learn provee, también, un transformador OneHotEncoder

Manejo de textos y atributos categóricos

```
In [78]: from sklearn.preprocessing import OneHotEncoder
         encoder = OneHotEncoder()
         housing_cat_1hot = encoder.fit_transform(housing_cat_encoded.reshape(-1,1))
         housing_cat_1hot
```

- Ejecutar estas líneas da como salida una matriz dispersa de SciPy
- Para cambiarlo a un arreglo, basta correr **housing_cat_1hot.toarray()**

Así mismo se puede convertir categorías numéricas a **one-hot vectors** utilizando la clase **LabelBinarizer**

```
In [80]: from sklearn.preprocessing import LabelBinarizer
         encoder = LabelBinarizer()
         housing_cat_1hot = encoder.fit_transform(housing_cat)
         housing_cat_1hot
```

Transformadores personalizados

A pesar de Scikit-Learn provee transformadores útiles, en ocasiones es necesario crear alguno propio para tareas específicas

Por ejemplo:

In [48]:
```
from sklearn.base import BaseEstimator, TransformerMixin
```

In [49]:
```
rooms_ix, bedrooms_ix, population_ix, household_ix = 3, 4, 5, 6
```

In [50]:
```
class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room = True): # no *args or **kargs
        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, y=None):
        rooms_per_household = X[:, rooms_ix] / X[:, household_ix]
        population_per_household = X[:, population_ix] / X[:, household_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]
```

In [51]:
```
attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
housing_extra_attribs = attr_adder.transform(housing.values)
```

Escalado de características

- Machine learning no se desempeña bien con escalas diferentes
- Existen dos formas para obtener atributos con la misma escala:
  - Min-Max scaling
  - Standardization

Min-max scaling: los valores son escalados en un rango de 0 a 1

$$MinMax = \frac{X - min}{max - min}$$

Standardization: los valores son escalados con base en el valor medio, pero no son acotados

$$Std = \frac{X - \bar{X}}{\sigma^2}$$

Scikit-Learn cuenta con ***MinMaxScaler*** y ***StandardScaler*** para escalar los datos

Tuberías de transformación

Diferentes transformaciones son necesarias de seguir en un orden correcto

Scikit-Learn provee la clase Pipeline para ayudar con esa secuencia de transformación

In [52]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([
            ('imputer', Imputer(strategy="median")),
            ('attribs_adder', CombinedAttributesAdder()),
            ('std_scaler', StandardScaler()),
        ])
housing_num_tr = num_pipeline.fit_transform(housing_num)
```

5. Seleccionar y entrenar el modelo

Una vez enmarcado el problema, teniendo los datos en conjuntos de prueba y entrenamiento y transformados, es tiempo de elegir el modelo de Machine Learning a trabajar.

- Entrenamiento y evaluación en el conjunto de entrenamiento
- Mejor evaluación usando evaluación cruzada

Entrenamiento y evaluación en el conjunto de entrenamiento

- Elegir el modelo de Machine learning a aprender
  - Regresión lineal
  - Árboles de decisión
  - Random Forests Regressors

Funciones ***LinearRegression***, ***DecisionTreeRegressor*** y ***RandomForestRegresor***

In [57]:
```
from sklearn.linear_model import LinearRegression

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

In [60]:
```
from sklearn.tree import DecisionTreeRegressor

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

In [65]:
```
from sklearn.ensemble import RandomForestRegressor
forest_reg = RandomForestRegressor()
forest_reg.fit(housing_prepared, housing_labels)
```

Entrenamiento y evaluación en el conjunto de entrenamiento

Tomando unos datos y el modelo de regresión se observa que la predicción no es muy precisa

```
Predictions:     [210644.60459286 317768.80697211 210956.43331178  59218.98886849
 189747.55849879]
Labels:          [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]
```

¿Cómo evaluar el desempeño de nuestro modelo de regresión lineal?

RMSE

```
In [59]: 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

Out[59]: 68628.19819848922
```

Hacer la misma evaluación para los Árboles de decisión, da un error de 0.0

¿Por lo tanto…?

Mejorar evaluación usando validación cruzada

- Utilizar un conjunto de validación
- K-fold cross-validation: 10 subconjuntos del conjunto de entrenamiento a fin de evaluar el modelo 10 veces.
- Resultado: 10 resultados de evaluación

In [62]:
```
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)
rmse_scores = np.sqrt(-scores)
```

In [63]:
```
def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())

display_scores(rmse_scores)
```

```
Scores: [69327.01708558 65486.39211857 71358.25563341 69091.37509104
 70570.20267046 75529.94622521 69895.20650652 70660.14247357
 75843.74719231 68905.17669382]
Mean: 70666.74616904806
Standard deviation: 2928.322738055112
```

Mejorar evaluación usando validación cruzada

Ejercicio: Hacer la validación cruzada de los tres modelos para saber cual es el “mejor”

Guardar el modelo favorito

In [67]:
```
from sklearn.externals import joblib
joblib.dump(forest_reg, "my_model.pkl")
# and later...
my_model_loaded = joblib.load("my_model.pkl")
```

6. Ajustar el modelo

De los modelos más prometedores, ajustarlos a su mejor desempeño

- Búsqueda de cuadrícula
- Búsqueda aleatoria
- Métodos de ensamble
- Analizar los mejores modelos y sus errores
- Evaluar el sistema en el conjunto de prueba

Búsqueda de cuadrícula

Se asigna un conjunto de parámetros en los cuales se evalúa el modelo para encontrar el mejor desempeño

In [68]:
```
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')

grid_search.fit(housing_prepared, housing_labels)
```

In [69]:
```
grid_search.best_params_
```

Out[69]:
```
{'max_features': 6, 'n_estimators': 30}
```

Búsqueda aleatoria y Métodos se ensamble

Para listas grandes de hiperparametros, ***RandomizedSearchCV,*** busca aleatoriamente los valores para que el modelo trabaje mejor

- En las iteraciones se busca combinaciones de valores a cada hiperparámetro.
- Se controla la cantidad de cómputo para cada hiperparámetro a través de las iteraciones

Métodos de Ensamble:

- Combinar los modelos con mejor desempeño
- Busca conjuntar desempeño de distintos modelos

Analizar los mejores modelos y sus errores y Evaluar el sistema en el conjunto de prueba

- Buscar pistas para resolver el problema a través de revisar los mejores modelos
- El modelo puede indicar la importancia de cada atributo a la hora de predecir
- Eliminar atributos no relevantes

- Evaluar el sistema en el conjunto de pruebas es la evaluación final al modelo
- Sólo es hacer la predicción en el conjunto de pruebas y evaluar el modelo
- El desempeño es usualmente peor que el de validación

In [75]:
```
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)
```

7. Lanzar, monitorear y mantener el sistema

Hora de colocar la entrada de datos al sistema

Escribir un código de monitoreo

Revisar el desempeño del sistema

Evaluar la calidad de los datos de entrada al sistema

Reentrenar con datos frescos, cada 6 meses regularmente