Dataframes, tablas de datos, pandas

Un dataframe es una estructura bidimensional, con filas y columnas, esto es, una tabla de datos. El *dataframe* es el tipo de datos esencial de la librería `pandas`.

Te ofrezco un recorrido por esta librería con el siguiente contenido:

- Operaciones básicas
- Ejemplos de aplicación
- Series

En el apartado *Operaciones básicas*, usamos tablas pequeñas, de juguete, no muy realistas pero claras y que fácilmente manejables, a mano, para facilitar tu experimentación.

En el segundo apartado, *Tablas de mayor tamaño*, cargamos algunas tablas desde un archivo de disco para ofrecer un repaso del manejo de tablas más grandes y poner ejemplos de manejo de datos un poco nás realista.

El tercer apartado trata de una clase especial de tablas, las *series*.

Para trabajar con estas tablas, se ha de empezar cargando esta librería, y es habitual hacerlo con el alias `pd`:

In [1]:
```
import pandas as pd
```

1. Operaciones básicas

1.a. Construcción de tablas

Veamos algunas operaciones básicas con tablas pequeñas.

Podemos construir un `DataFrame` de muchas formas. Empezamos creando tablas pequeñas, a mano. Los ejemplos siguientes hablan pos sí mismos.

In [2]:
```
# Creación de una tabla o dataframe
# dando los nombres de sus columnas y valores a sus elementos:

tabla_a = pd.DataFrame({"A": [1, 3, 5], "B": [2, 4, 6]})
print(tabla_a)
```

```
   A  B
0  1  2
1  3  4
2  5  6
```

In [3]:
```
# El valor `None` permite forzar un dato missing (`NaN`, *Not A Number*

tabla_b = pd.DataFrame({"B": [2, 4], "C": [20, None]})
print(tabla_b)
```

```
   B     C
0  2  20.0
1  4   NaN
```

In [4]:
```
# A veces, necesitaremos crear un dataframe con números aleatorios:

import numpy as np
np.random.seed(0) # damos una semilla concreta para que la ejecución sea

# Se generan 3x4 números aleatorios N(0, 1):

tabla_c = pd.DataFrame(np.random.randn(3, 4), columns=["A", "B", "C", "D
tabla_c
```

Out[4]:

	A	B	C	D
0	1.764052	0.400157	0.978738	2.240893
1	1.867558	-0.977278	0.950088	-0.151357
2	-0.103219	0.410599	0.144044	1.454274

In [5]:
```
# Creación de un dataframe a partir de una lista de tuplas
# y sin dar los nombres de las columnas:

tuplas = [
    (1, 2, 3),
    (4, 5, 6)
]
tabla_d = pd.DataFrame(tuplas)
tabla_d
```

Out[5]:

	0	1	2
0	1	2	3
1	4	5	6

In [6]:
```
# Ahora podemos renombramos las columnas:

tabla_d.columns = ["Una", "Dos", "Tres"]
tabla_d
```

Out[6]:

	Una	Dos	Tres
0	1	2	3
1	4	5	6

In [7]:
```
# Los dos pasos anteriores a un tiempo:
# - Creación de la tabla (con una lista de listas esta vez),
# - y nombrabdo las columnas desde el principio:

listas = [
    [1, 2, 3],
    [4, 5, 6]
]

tabla_e = pd.DataFrame(listas, columns=["Una", "Dos", "Tres"])

tabla_e
```

Out[7]:

	Una	Dos	Tres
0	1	2	3
1	4	5	6

1.b. Modificación de tablas

In [8]:
```
# Añadir (o modificar) una fila:

otra_fila = {"Una": 100, "Dos": 200, "Nueva": 300}
tabla_e.loc[2] = otra_fila
tabla_e
```

Out[8]:

	Una	Dos	Tres
0	1	2	3.0
1	4	5	6.0
2	100	200	NaN

In [9]:
```
# Modificar el valor de una celda:

tabla_e.at[1, "Dos"] = 1000
tabla_e
```

Out[9]:

	Una	Dos	Tres
0	1	2	3.0
1	4	1000	6.0
2	100	200	NaN

In [10]:
```
# Modificar los valores de una fila:

tabla_e.loc[1] = [-1, -2, -3]
tabla_e
```

Out[10]:

	Una	Dos	Tres
0	1	2	3.0
1	-1	-2	-3.0
2	100	200	NaN

In [11]:
```
# Modificar los valores de una columna:

tabla_e["Dos"] = [20, 50, 90]
tabla_e
```

Out[11]:

	Una	Dos	Tres
0	1	20	3.0
1	-1	50	-3.0
2	100	90	NaN

In [12]:
```
# Añadir una columna:

tabla_e["Nueva"] = [-2, -3, -5]
tabla_e
```

Out[12]:

	Una	Dos	Tres	Nueva
0	1	20	3.0	-2
1	-1	50	-3.0	-3
2	100	90	NaN	-5

In [13]:
```
# Suprimir una columna:

del tabla_e["Tres"]
tabla_e
```

Out[13]:

	Una	Dos	Nueva
0	1	20	-2
1	-1	50	-3
2	100	90	-5

In [14]:
```
# Otra manera de suprimir columnas...

tabla_e.drop(["Una", "Dos"], axis="columns")

# Pero OJO...
```

Out[14]:

	Nueva
0	-2
1	-3
2	-5

In [15]:
```
# Cuidado, se ha calculado la eliminación de la columna,
# pero la operación no se ha realizado in place:

tabla_e
```

Out[15]:

	Una	Dos	Nueva
0	1	20	-2
1	-1	50	-3
2	100	90	-5

In [16]:
```
tabla_e.drop(["Una", "Dos"], axis="columns", inplace=True)
tabla_e
```

Out[16]:

	Nueva
0	-2
1	-3
2	-5

1.c. Operaciones *in place*

In [17]:
```
# Una tabla para los ejemplos de este apartado:

tabla = pd.DataFrame({"A": [1, 3, 5, 7, 9, 11], "B": [2, 4, 6, 1, 2, 3]})
tabla
```

Out[17]:

	A	B
0	1	2
1	3	4
2	5	6
3	7	1
4	9	2
5	11	3

In [18]:
```
# Ordenamos las filas de un dataframe

tabla.sort_values("B")
tabla

# OJO: no parece haber funcionado
```

Out[18]:

	A	B
0	1	2
1	3	4
2	5	6
3	7	1
4	9	2
5	11	3

In [19]:
```
# La operación, en realidad, sí funciona...

tabla.sort_values("B")
```

Out[19]:

	A	B
3	7	1
0	1	2
4	9	2
5	11	3
1	3	4
2	5	6

In [20]:
```
# Pero para que la tabla se actualice...

tabla.sort_values("B", inplace=True)
tabla
```

Out[20]:

	A	B
3	7	1
0	1	2
4	9	2
5	11	3
1	3	4
2	5	6

In [21]:
```
# Y para que el índice también se reinicie:

tabla.reset_index(inplace=True)

tabla
```

Out[21]:

	index	A	B
0	3	7	1
1	0	1	2
2	4	9	2
3	5	11	3
4	1	3	4
5	2	5	6

1.d. Modificación genérica de columnas

In [22]:
```
# Vamos a usar esta pequeña tabla como ejemplo:

viviendas = pd.DataFrame(
    {'Zona': ['B.Pilar', 'Chueca', 'Moncloa', 'B.Pilar', 'Chueca', 'Chue
     'Dorms': [3, 2, 4, 4, 3, 1, 2],
     'Precio': [450, 600, 750, 550, 850, 300, 500]})

viviendas
```

Out[22]:

	Zona	Dorms	Precio
0	B.Pilar	3	450
1	Chueca	2	600
2	Moncloa	4	750
3	B.Pilar	4	550
4	Chueca	3	850
5	Chueca	1	300
6	Moncloa	2	500

In [23]:
```
# Modificamos la columna "Precio", subiendo un 10%:

viviendas["Precio"] = viviendas["Precio"] * 1.10

viviendas
```

Out[23]:

	Zona	Dorms	Precio
0	B.Pilar	3	495.0
1	Chueca	2	660.0
2	Moncloa	4	825.0
3	B.Pilar	4	605.0
4	Chueca	3	935.0
5	Chueca	1	330.0
6	Moncloa	2	550.0

In [24]:
```
# De otro modo, modificamos la columna "Precio", subiendo un 10%:

viviendas["Precio"] = viviendas["Precio"].apply(lambda valor: valor*1.10

viviendas
```

Out[24]:

	Zona	Dorms	Precio
0	B.Pilar	3	544.5
1	Chueca	2	726.0
2	Moncloa	4	907.5
3	B.Pilar	4	665.5
4	Chueca	3	1028.5
5	Chueca	1	363.0
6	Moncloa	2	605.0

In [25]:
```
# Observa que, sin la asignación, parece que la tabla se actualiza...

viviendas["Precio"].apply(lambda valor: 0)
```

Out[25]:
```
0    0
1    0
2    0
3    0
4    0
5    0
6    0
Name: Precio, dtype: int64
```

In [26]:
```
# Pero no, la operación no es in place y por eso hace falta realizar la

viviendas
```

Out[26]:

	Zona	Dorms	Precio
0	B.Pilar	3	544.5
1	Chueca	2	726.0
2	Moncloa	4	907.5
3	B.Pilar	4	665.5
4	Chueca	3	1028.5
5	Chueca	1	363.0
6	Moncloa	2	605.0

In [27]:
```
# Modificación condicional:
# Los inmuebles de Chueca bajan su precio en 100 euros exactamente:

viviendas["Precio"] = viviendas.loc[viviendas["Zona"] == "Chueca", ["Pre
   viviendas["Precio"] - 100

viviendas
```

Out[27]:

	Zona	Dorms	Precio
0	B.Pilar	3	444.5
1	Chueca	2	626.0
2	Moncloa	4	807.5
3	B.Pilar	4	565.5
4	Chueca	3	928.5
5	Chueca	1	263.0
6	Moncloa	2	505.0

In [28]:
```
# Modificación condicional:

aumento_de_precio = lambda precio : precio + 100 if precio >= 500 else p

viviendas["Precio"] = viviendas["Precio"].apply(aumento_de_precio)

viviendas
```

Out[28]:

	Zona	Dorms	Precio
0	B.Pilar	3	494.5
1	Chueca	2	726.0
2	Moncloa	4	907.5
3	B.Pilar	4	665.5
4	Chueca	3	1028.5
5	Chueca	1	313.0
6	Moncloa	2	605.0

In [29]:
```
# Cálculo de una columna nueva con una fórmula:

viviendas["P.p.d."] = viviendas["Precio"] / viviendas["Dorms"]

viviendas
```

Out[29]:

	Zona	Dorms	Precio	P.p.d.
0	B.Pilar	3	494.5	164.833333
1	Chueca	2	726.0	363.000000
2	Moncloa	4	907.5	226.875000
3	B.Pilar	4	665.5	166.375000
4	Chueca	3	1028.5	342.833333
5	Chueca	1	313.0	313.000000
6	Moncloa	2	605.0	302.500000

In [30]:
```
# Modificación de una columna:

viviendas["P.p.d."] = round(viviendas["P.p.d."], 2)

viviendas
```

Out[30]:

	Zona	Dorms	Precio	P.p.d.
0	B.Pilar	3	494.5	164.83
1	Chueca	2	726.0	363.00
2	Moncloa	4	907.5	226.88
3	B.Pilar	4	665.5	166.38
4	Chueca	3	1028.5	342.83
5	Chueca	1	313.0	313.00
6	Moncloa	2	605.0	302.50

1.e. Agrupamiento de datos

In [31]:
```
viviendas = pd.DataFrame(
    {'Zona': ['B.Pilar', 'Chueca', 'Moncloa', 'B.Pilar', 'Chueca', 'Chue
     'Dorms': [3, 2, 4, 4, 3, 1, 2],
     'Precio': [450, 600, 750, 550, 850, 300, 500]})

viviendas
```

Out[31]:

	Zona	Dorms	Precio
0	B.Pilar	3	450
1	Chueca	2	600
2	Moncloa	4	750
3	B.Pilar	4	550
4	Chueca	3	850
5	Chueca	1	300
6	Moncloa	2	500

In [32]:
```
viviendas.groupby("Zona").mean()
```

Out[32]:

Zona	Dorms	Precio
B.Pilar	3.5	500.000000
Chueca	2.0	583.333333
Moncloa	3.0	625.000000

In [33]:
```
viviendas.groupby("Zona").count()
```

Out[33]:

Zona	Dorms	Precio
B.Pilar	2	2
Chueca	3	3
Moncloa	2	2

In [34]:
```
# Una manera de evitar la duplicidad de columnas anterior:

viviendas.groupby("Zona").size()
```

Out[34]:
```
Zona
B.Pilar    2
Chueca     3
Moncloa    2
dtype: int64
```

In [35]:
```
# Otra manera de evitar la duplicidad de columnas:

props = viviendas[["Zona", "Dorms"]]
props.groupby("Zona").count()
```

Out[35]:

Zona	Dorms
B.Pilar	2
Chueca	3
Moncloa	2

In [36]:
```
viviendas.groupby(["Zona", "Dorms"]).mean()
```

Out[36]:

Zona	Dorms	Precio
B.Pilar	**3**	450.0
	4	550.0
Chueca	**1**	300.0
	2	600.0
	3	850.0
Moncloa	**2**	500.0
	4	750.0

In [37]:
```
viviendas.groupby(["Zona", "Dorms"]).agg(["mean", "count"])
```

Out[37]:

		Precio	
		mean	count
Zona	Dorms		
B.Pilar	3	450.0	1
	4	550.0	1
Chueca	1	300.0	1
	2	600.0	1
	3	850.0	1
Moncloa	2	500.0	1
	4	750.0	1

1.f. Estadísticos básicos

In [38]:
```
# Estadísticos básicos con datos cuantitativos:

viviendas.describe()

# Obsérvese que la columna "Zona" no se está incluyendo
```

Out[38]:

	Dorms	Precio
count	7.000000	7.000000
mean	2.714286	571.428571
std	1.112697	184.519969
min	1.000000	300.000000
25%	2.000000	475.000000
50%	3.000000	550.000000
75%	3.500000	675.000000
max	4.000000	850.000000

In [39]:
```
# Si deseamos limitarnos a una columna:

viviendas["Dorms"].describe()
```

Out[39]:
```
count    7.000000
mean     2.714286
std      1.112697
min      1.000000
25%      2.000000
50%      3.000000
75%      3.500000
max      4.000000
Name: Dorms, dtype: float64
```

In [40]:
```
# Con datos cualitativos o categóricos:

viviendas["Zona"].describe()

# Obtenemos: cuantas filas tenemos en total, cuantas zonas distintas,
# la más frecuente (la moda) y su frecuencia
```

Out[40]:
```
count          7
unique         3
top       Chueca
freq           3
Name: Zona, dtype: object
```

In [41]:
```
# Estadística completa, datos cuantotativos y cualitativos:

viviendas.describe(include="all")

# Obvérvense los valores NaN
```

Out[41]:

	Zona	Dorms	Precio
count	7	7.000000	7.000000
unique	3	NaN	NaN
top	Chueca	NaN	NaN
freq	3	NaN	NaN
mean	NaN	2.714286	571.428571
std	NaN	1.112697	184.519969
min	NaN	1.000000	300.000000
25%	NaN	2.000000	475.000000
50%	NaN	3.000000	550.000000
75%	NaN	3.500000	675.000000
max	NaN	4.000000	850.000000

Explicación rápida:

- Con datos cualitativos:
  - `count` : cuántas filas tiene la tabla
  - `unique` : cuántos valores distintos
  - `top` : la moda
  - `freq` : frecuencia absoluta de la moda
- Con datos cuantitativos:
  - `mean` : media
  - `std` : desviación típica
  - `min` , `max` : mínimo y máximo
  - 25%, 50%, 75%: cuartiles

Obsérvese cada valor `NaN` inadecuado con datos cuantitativos y cualitativos.

In [42]:
```
# Podríamos haber obtenido estos estadísticos por separado:

print(viviendas["Dorms"].count())
print(viviendas["Dorms"].mean())
print(viviendas["Dorms"].min())
print(viviendas["Dorms"].quantile(q=0.25))
print(viviendas["Dorms"].quantile(q=0.50))
print(viviendas["Dorms"].quantile(q=0.75))
print(viviendas["Dorms"].max())
```

```
7
2.7142857142857144
1
2.0
3.0
3.5
4
```

1.g. Combinación de tablas

In [43]:
```
# Combinación vertical de dos tablas:

tabla1 = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6]})
print(tabla1, "\n")

tabla2 = pd.DataFrame({"A": [7, 8, 9], "B": [10, 11, 12]})
print(tabla2, "\n")

tabla3 = pd.concat([tabla1, tabla2], ignore_index=True)
print(tabla3)
```

```
   A  B
0  1  4
1  2  5
2  3  6

   A   B
0  7  10
1  8  11
2  9  12

   A   B
0  1   4
1  2   5
2  3   6
3  7  10
4  8  11
5  9  12
```

In [44]:

```
# Otra situación:

tabla1 = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6]})
print(tabla1, "\n")

tabla2 = pd.DataFrame({"B": [7, 8, 9], "C": [10, 11, 12]})
print(tabla2, "\n")

tabla3 = pd.concat([tabla1, tabla2], ignore_index=True)
print(tabla3)


# Observa que, al no coincidir el nombre de *todas* las columnas,
# la operación completa los valores desconocidos,
# sin tener en cuenta si hay valores coincidentes.
```

```
   A  B
0  1  4
1  2  5
2  3  6

   B   C
0  7  10
1  8  11
2  9  12

     A  B     C
0  1.0  4   NaN
1  2.0  5   NaN
2  3.0  6   NaN
3  NaN  7  10.0
4  NaN  8  11.0
5  NaN  9  12.0
```

In [45]:

```
# Merge:

tabla1 = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6]})
print(tabla1, "\n")

tabla2 = pd.DataFrame({"B": [4, 5, 6], "C": [7, 8, 9]})
print(tabla2, "\n")

tabla3 = pd.merge(tabla1, tabla2, on="B")

print(tabla3)
```

```
   A  B
0  1  4
1  2  5
2  3  6

   B  C
0  4  7
1  5  8
2  6  9

   A  B  C
0  1  4  7
1  2  5  8
2  3  6  9
```

In [46]:

```
# Es el join interno de SQL.
# Observa el funcionamiento cuando algún valor no coincide:

tabla1 = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6]})
print(tabla1, "\n")

tabla2 = pd.DataFrame({"B": [4, 5, 60], "C": [7, 8, 9]})
print(tabla2, "\n")

tabla3 = pd.merge(tabla1, tabla2, on="B")

print(tabla3)
```

```
   A  B
0  1  4
1  2  5
2  3  6

    B  C
0   4  7
1   5  8
2  60  9

   A  B  C
0  1  4  7
1  2  5  8
```

La columna que juega el papel de bisagra se llama *pivote*. En este ejemplo, ha sido la coumna `B`. Esta columna no se duplica. En esta unión, únicamente se tiene en cuenta las filas de `tabla1` y `tabla2` cuyos valores coinciden en la columna `B`.

En resumen, y en el lenguaje de las bases de datos, la combinación de tablas por defecto es el `inner join` clásico, la unión interna. Pero hay otras pobibilidades:

In [47]:

```
tabla4 = tabla1.merge(tabla2, how="outer")
print(tabla4)

tabla5 = tabla1.merge(tabla2, how="left")
print(tabla5)

tabla6 = tabla1.merge(tabla2, how="right")
## Aplicación: *realstate*
print(tabla6)
```

```
     A   B    C
0  1.0   4  7.0
1  2.0   5  8.0
2  3.0   6  NaN
3  NaN  60  9.0
   A  B    C
0  1  4  7.0
1  2  5  8.0
2  3  6  NaN
     A   B  C
0  1.0   4  7
1  2.0   5  8
2  NaN  60  9
```

2. Ejemploa de aplicación

2.1. Agencia inmobiliaria *Realstate*

Vamos con un ejemplo un poco más realista. Cargamos ahora una tabla a partir de un archivo `cvs`, que contiene datos de una inmobiliaria.

In [48]:
```
realestate = pd.read_csv('./data/realestate.csv')
realestate
```

Out[48]:

	street	city	zip	state	beds	baths	sq__ft	type	sale_
0	3526 HIGH ST	SACRAMENTO	95838	CA	2	1	836	Residential	Wed 00:(EDT
1	51 OMAHA CT	SACRAMENTO	95823	CA	3	1	1167	Residential	Wed 00:(EDT
2	2796 BRANCH ST	SACRAMENTO	95815	CA	2	1	796	Residential	Wed 00:(EDT
3	2805 JANETTE WAY	SACRAMENTO	95815	CA	2	1	852	Residential	Wed 00:(EDT
4	6001 MCMAHON DR	SACRAMENTO	95824	CA	2	1	797	Residential	Wed 00:(EDT
...	...	...	...	...	...	...	...	...	
980	9169 GARLINGTON CT	SACRAMENTO	95829	CA	4	3	2280	Residential	Thu 00:(EDT
981	6932 RUSKUT WAY	SACRAMENTO	95823	CA	3	2	1477	Residential	Thu 00:(EDT
982	7933 DAFFODIL WAY	CITRUS HEIGHTS	95610	CA	3	2	1216	Residential	Thu 00:(EDT
983	8304 RED FOX WAY	ELK GROVE	95758	CA	4	2	1685	Residential	Thu 00:(EDT
984	3882 YELLOWSTONE LN	EL DORADO HILLS	95762	CA	3	2	1362	Residential	Thu 00:(EDT

985 rows × 12 columns

In [49]:
```
type(realestate)
```

Out[49]:
```
pandas.core.frame.DataFrame
```

In [50]:
```
realestate.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 985 entries, 0 to 984
Data columns (total 12 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   street     985 non-null    object 
 1   city       985 non-null    object 
 2   zip        985 non-null    int64  
 3   state      985 non-null    object 
 4   beds       985 non-null    int64  
 5   baths      985 non-null    int64  
 6   sq__ft     985 non-null    int64  
 7   type       985 non-null    object 
 8   sale_date  985 non-null    object 
 9   price      985 non-null    int64  
 10  latitude   985 non-null    float64
 11  longitude  985 non-null    float64
dtypes: float64(2), int64(5), object(5)
memory usage: 92.5+ KB
```

Vamos a comentar a continuación cómo podemos acceder a los diferentes elementos de la tabla

Acceso a columnas

In [51]:
```
realestate['price']
```

Out[51]:
```
0       59222
1       68212
2       68880
3       69307
4       81900
        ...
980    232425
981    234000
982    235000
983    235301
984    235738
Name: price, Length: 985, dtype: int64
```

In [52]:
```
realestate[['price','zip']]
```

Out[52]:

	price	zip
0	59222	95838
1	68212	95823
2	68880	95815
3	69307	95815
4	81900	95824
...	...	...
980	232425	95829
981	234000	95823
982	235000	95610
983	235301	95758
984	235738	95762

985 rows × 2 columns

In [53]:
```
selcols = realestate.columns[2:6]
selcols
```

Out[53]:
```
Index(['zip', 'state', 'beds', 'baths'], dtype='object')
```

In [54]:
```
realestate[selcols]
```

Out[54]:

	zip	state	beds	baths
0	95838	CA	2	1
1	95823	CA	3	1
2	95815	CA	2	1
3	95815	CA	2	1
4	95824	CA	2	1
...	...	...	...	...
980	95829	CA	4	3
981	95823	CA	3	2
982	95610	CA	3	2
983	95758	CA	4	2
984	95762	CA	3	2

985 rows × 4 columns

In [55]:
```
mixed = list(selcols)+['price']
mixed
```

Out[55]:
```
['zip', 'state', 'beds', 'baths', 'price']
```

In [56]:
```
realestate[mixed]
```

Out[56]:

	zip	state	beds	baths	price
0	95838	CA	2	1	59222
1	95823	CA	3	1	68212
2	95815	CA	2	1	68880
3	95815	CA	2	1	69307
4	95824	CA	2	1	81900
...	...	...	...	...	...
980	95829	CA	4	3	232425
981	95823	CA	3	2	234000
982	95610	CA	3	2	235000
983	95758	CA	4	2	235301
984	95762	CA	3	2	235738

985 rows × 5 columns

Acceso a filas

In [57]:
```
realestate.iloc[3]
```

Out[57]:
```
street                    2805 JANETTE WAY
city                             SACRAMENTO
zip                                   95815
state                                    CA
beds                                      2
baths                                     1
sq__ft                                  852
type                            Residential
sale_date    Wed May 21 00:00:00 EDT 2008
price                                 69307
latitude                          38.616835
longitude                       -121.439146
Name: 3, dtype: object
```

In [58]:
```
realestate.iloc[[3,6]]
```

Out[58]:

	street	city	zip	state	beds	baths	sq__ft	type	sale_date	p
3	2805 JANETTE WAY	SACRAMENTO	95815	CA	2	1	852	Residential	Wed May 21 00:00:00 EDT 2008	69
6	6048 OGDEN NASH WAY	SACRAMENTO	95842	CA	3	2	1104	Residential	Wed May 21 00:00:00 EDT 2008	90

In [59]:
```
realestate.iloc[3:6]
```

Out[59]:

	street	city	zip	state	beds	baths	sq__ft	type	sale_date
3	2805 JANETTE WAY	SACRAMENTO	95815	CA	2	1	852	Residential	Wed May 21 00:00:00 EDT 2008
4	6001 MCMAHON DR	SACRAMENTO	95824	CA	2	1	797	Residential	Wed May 21 00:00:00 EDT 2008
5	5828 PEPPERMILL CT	SACRAMENTO	95841	CA	3	1	1122	Condo	Wed May 21 00:00:00 EDT 2008

In [60]:
```
list(range(3,8,2))
```

Out[60]:
```
[3, 5, 7]
```

In [61]:
```
realestate.iloc[range(3,8,2)]
```

Out[61]:

	street	city	zip	state	beds	baths	sq__ft	type	sale_date
3	2805 JANETTE WAY	SACRAMENTO	95815	CA	2	1	852	Residential	Wed May 21 00:00:00 EDT 2008
5	5828 PEPPERMILL CT	SACRAMENTO	95841	CA	3	1	1122	Condo	Wed May 21 00:00:00 EDT 2008
7	2561 19TH AVE	SACRAMENTO	95820	CA	3	1	1177	Residential	Wed May 21 00:00:00 EDT 2008

In [62]:
```
realestate.iloc[3]['city']
```

Out[62]:
```
'SACRAMENTO'
```

In [63]:
```
realestate.iloc[3][1]
```

Out[63]:
```
'SACRAMENTO'
```

In [64]:
```
realestate[['price','zip']][3:8]
```

Out[64]:

	price	zip
3	69307	95815
4	81900	95824
5	89921	95841
6	90895	95842
7	91002	95820

Operaciones globales

El módulo `pandas` nos afrece una amplísima variedad de funciones para trabajar con los `DataFrame` , a continuación, a modo de ejemplo te mostramos algunas.

In [65]:
```
realestate.sort_values(by='price', ascending=False)
```

Out[65]:

	street	city	zip	state	beds	baths	sq__ft	type	s
864	9401 BARREL RACER CT	WILTON	95693	CA	4	3	4400	Residential	
863	2982 ABERDEEN LN	EL DORADO HILLS	95762	CA	4	3	0	Residential	
334	3935 EL MONTE DR	LOOMIS	95650	CA	4	4	1624	Residential	
157	315 JUMEL CT	EL DORADO HILLS	95762	CA	6	5	0	Residential	

In [66]:
```
realestate.describe()
```

Out[66]:

	zip	beds	baths	sq__ft	price	latitude	
count	985.000000	985.000000	985.000000	985.000000	985.000000	985.000000	9
mean	95750.697462	2.911675	1.776650	1314.916751	234144.263959	38.607732	-1
std	85.176072	1.307932	0.895371	853.048243	138365.839085	0.145433	
min	95603.000000	0.000000	0.000000	0.000000	1551.000000	38.241514	-1
25%	95660.000000	2.000000	1.000000	952.000000	145000.000000	38.482717	-1
50%	95762.000000	3.000000	2.000000	1304.000000	213750.000000	38.626582	-1
75%	95828.000000	4.000000	2.000000	1718.000000	300000.000000	38.695589	-1
max	95864.000000	8.000000	5.000000	5822.000000	884790.000000	39.020808	-1

In [67]:
```
cond = realestate['baths'] == 2
cond
```

Out[67]:
```
0      False
1      False
2      False
3      False
4      False
       ...
980    False
981     True
982     True
983     True
984     True
Name: baths, Length: 985, dtype: bool
```

In [68]:
```
realestate[cond]
```

Out[68]:

	street	city	zip	state	beds	baths	sq__ft	type	sale_
6	6048 OGDEN NASH WAY	SACRAMENTO	95842	CA	3	2	1104	Residential	Wed 00:(EDT
8	11150 TRINITY RIVER DR Unit 114	RANCHO CORDOVA	95670	CA	2	2	941	Condo	Wed 00:(EDT
9	7325 10TH ST	RIO LINDA	95673	CA	3	2	1146	Residential	Wed 00:(EDT
10	645 MORRISON AVE	SACRAMENTO	95838	CA	3	2	909	Residential	Wed 00:(EDT
11	4085 FAWN CIR	SACRAMENTO	95823	CA	3	2	1289	Residential	Wed 00:(EDT
...	...	...	...	...	...	...	...	...	
979	1909 YARNELL WAY	ELK GROVE	95758	CA	3	2	1262	Residential	Thu 00:(EDT
981	6932 RUSKUT WAY	SACRAMENTO	95823	CA	3	2	1477	Residential	Thu 00:(EDT
982	7933 DAFFODIL WAY	CITRUS HEIGHTS	95610	CA	3	2	1216	Residential	Thu 00:(EDT
983	8304 RED FOX WAY	ELK GROVE	95758	CA	4	2	1685	Residential	Thu 00:(EDT
984	3882 YELLOWSTONE LN	EL DORADO HILLS	95762	CA	3	2	1362	Residential	Thu 00:(EDT

544 rows × 12 columns

In [69]:

```
two_baths = realestate['baths'] == 2
three_beds = realestate['beds'] == 3
realestate[two_baths & three_beds]
```

Out[69]:

	street	city	zip	state	beds	baths	sq__ft	type	sale_
6	6048 OGDEN NASH WAY	SACRAMENTO	95842	CA	3	2	1104	Residential	Wed 00:(EDT
9	7325 10TH ST	RIO LINDA	95673	CA	3	2	1146	Residential	Wed 00:(EDT
10	645 MORRISON AVE	SACRAMENTO	95838	CA	3	2	909	Residential	Wed 00:(EDT
11	4085 FAWN CIR	SACRAMENTO	95823	CA	3	2	1289	Residential	Wed 00:(EDT
19	113 LEEWILL AVE	RIO LINDA	95673	CA	3	2	1356	Residential	Wed 00:(EDT
...	...	...	...	...	...	...	...	...	
976	2400 INVERNESS DR	LINCOLN	95648	CA	3	2	1358	Residential	Thu 00:(EDT
979	1909 YARNELL WAY	ELK GROVE	95758	CA	3	2	1262	Residential	Thu 00:(EDT
981	6932 RUSKUT WAY	SACRAMENTO	95823	CA	3	2	1477	Residential	Thu 00:(EDT
982	7933 DAFFODIL WAY	CITRUS HEIGHTS	95610	CA	3	2	1216	Residential	Thu 00:(EDT
984	3882 YELLOWSTONE LN	EL DORADO HILLS	95762	CA	3	2	1362	Residential	Thu 00:(EDT

311 rows × 12 columns

In [70]:
```
realestate[two_baths & three_beds]['zip'].value_counts()
```

Out[70]:
```
zip
95823    33
95828    19
95843    17
95838    16
95758    14
95757    14
95835    12
95678    12
95621    11
95632     9
95822     9
95624     8
95832     8
95660     8
95834     8
95608     7
95670     7
95833     6
95762     6
95630     6
95648     6
95673     5
95610     5
95842     5
95667     5
95826     5
95827     4
95662     4
95747     4
95682     3
95626     3
95829     3
95628     3
95765     2
95831     2
95677     2
95661     2
95663     1
95614     1
95825     1
95864     1
95821     1
95619     1
95633     1
95817     1
95635     1
95819     1
95623     1
95815     1
95820     1
95841     1
95655     1
95726     1
95693     1
95824     1
Name: count, dtype: int64
```

In [71]:
```
realestate.groupby(['zip','baths']).size()
```

Out[71]:
```
zip    baths
95603  2          2
       3          3
95608  1          4
       2         15
       3          1
                 ..
95843  2         24
       3          8
95864  1          3
       2          1
       3          1
Length: 185, dtype: int64
```

In [72]:
```
realestate.groupby(['zip','baths']).size().unstack()
```

Out[72]:

baths	0	1	2	3	4	5
zip						
95603	NaN	NaN	2.0	3.0	NaN	NaN
95608	NaN	4.0	15.0	1.0	NaN	NaN
95610	NaN	NaN	5.0	1.0	1.0	NaN
95614	NaN	NaN	1.0	NaN	NaN	NaN
95619	NaN	NaN	1.0	NaN	NaN	NaN
...	...	...	...	...	...	...
95838	1.0	11.0	24.0	1.0	NaN	NaN
95841	NaN	4.0	3.0	NaN	NaN	NaN
95842	NaN	7.0	14.0	1.0	NaN	NaN
95843	NaN	1.0	24.0	8.0	NaN	NaN
95864	NaN	3.0	1.0	1.0	NaN	NaN

68 rows × 6 columns

In [73]:
```
# Mejor sin NaN (que son ceros realmente) y con tipo de datos entero:

tabla = realestate.groupby(['zip','baths']).size().unstack()
tabla = tabla.fillna(0)
tabla[tabla.columns] = tabla[tabla.columns].astype(int)
tabla
```

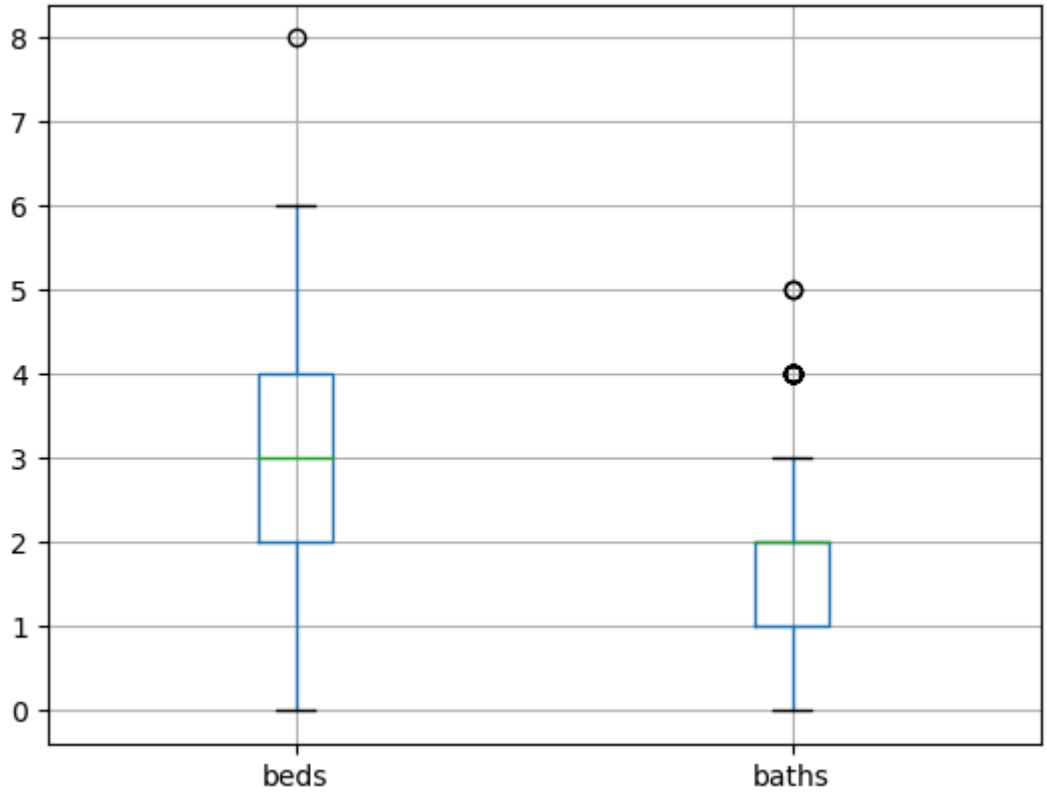
Out[73]:

baths	0	1	2	3	4	5
zip						
95603	0	0	2	3	0	0
95608	0	4	15	1	0	0
95610	0	0	5	1	1	0
95614	0	0	1	0	0	0
95619	0	0	1	0	0	0
...	...	...	...	...	...	...
95838	1	11	24	1	0	0
95841	0	4	3	0	0	0
95842	0	7	14	1	0	0
95843	0	1	24	8	0	0

Visualización

In [74]:
```
realestate[['beds','baths']].boxplot()
```

Out[74]: <Axes: >



In [75]:
```
realestate[['beds','baths']].boxplot(by="beds")
```

Out[75]:
```
<Axes: title={'center': 'baths'}, xlabel='[beds]'>
```

Matplotlib

In [76]:
```
tabla_habs_precios = realestate[['beds','price']]
tabla_habs_precios
```

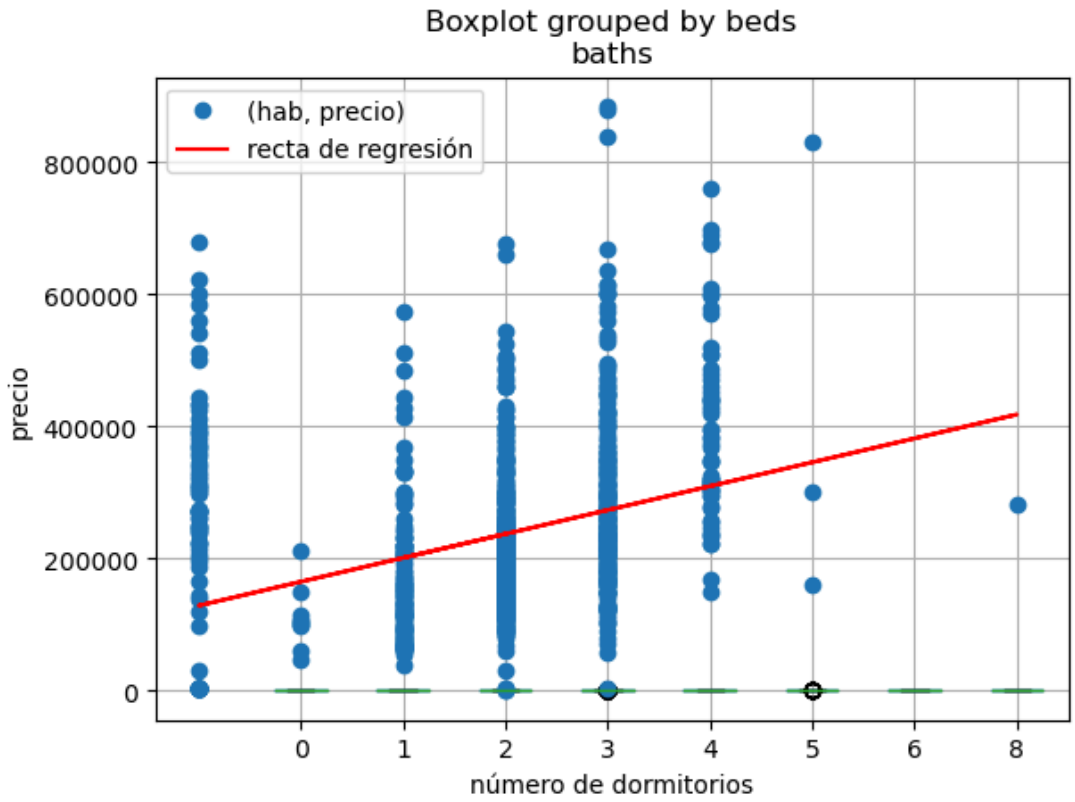
Out[76]:

	beds	price
0	2	59222
1	3	68212
2	2	68880
3	2	69307
4	2	81900
...	...	...
980	4	232425
981	3	234000
982	3	235000
983	4	235301
984	3	235738

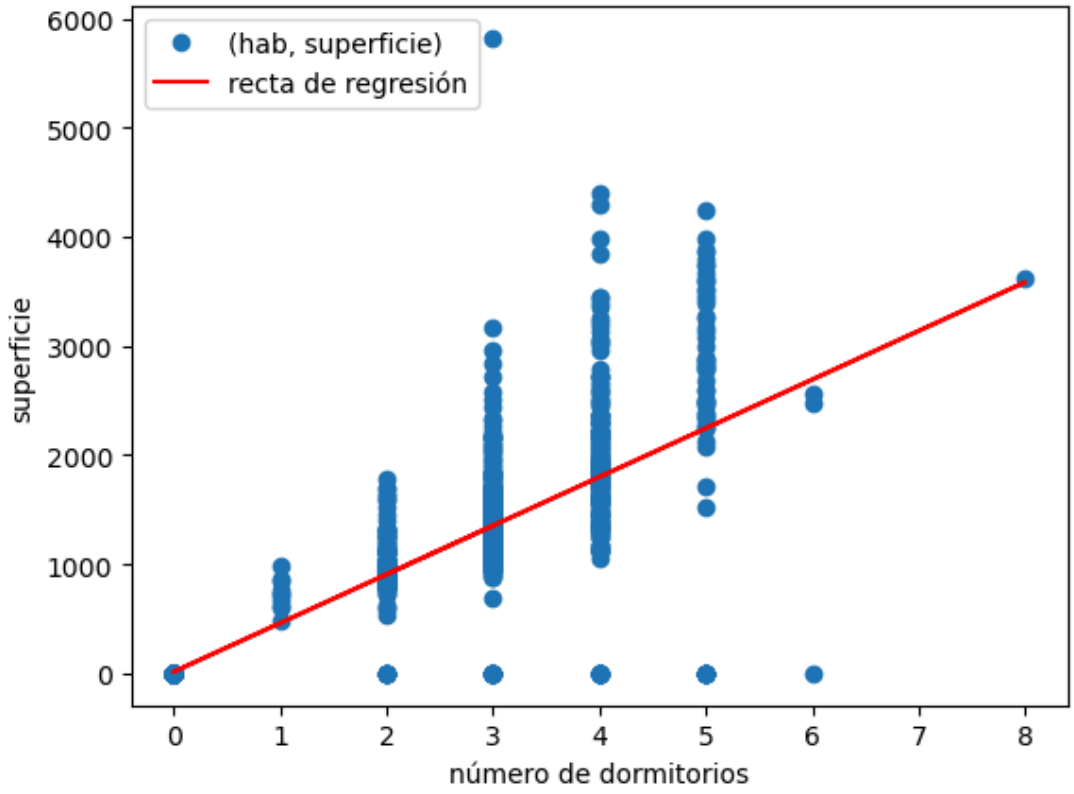
985 rows × 2 columns

In [77]:

```
from scipy import stats
import numpy as np
import matplotlib.pyplot as plt

xs = tabla_habs_precios['beds']
ys = tabla_habs_precios['price']
slope, intercept, r_value, p_value, std_err = stats.linregress(xs, ys)
recta_regres = lambda x: intercept + slope*x

plt.plot(xs, ys, 'o', label='(hab, precio)')
plt.plot(xs, recta_regres(xs), 'r', label='recta de regresión')
plt.legend(loc = 'upper left')
plt.xlabel('número de dormitorios')
plt.ylabel('precio')
plt.show()
```



In [78]:
```
tabla_habs_superficie = realestate[['beds','sq__ft']]
xs = tabla_habs_superficie['beds']
ys = tabla_habs_superficie['sq__ft']
slope, intercept, r_value, p_value, std_err = stats.linregress(xs, ys)
recta_regres = lambda x: intercept + slope*x

plt.plot(xs, ys, 'o', label='(hab, superficie)')
plt.plot(xs, recta_regres(xs), 'r', label='recta de regresión')
plt.legend(loc = 'upper left')
plt.xlabel('número de dormitorios')
plt.ylabel('superficie')
plt.show()
```



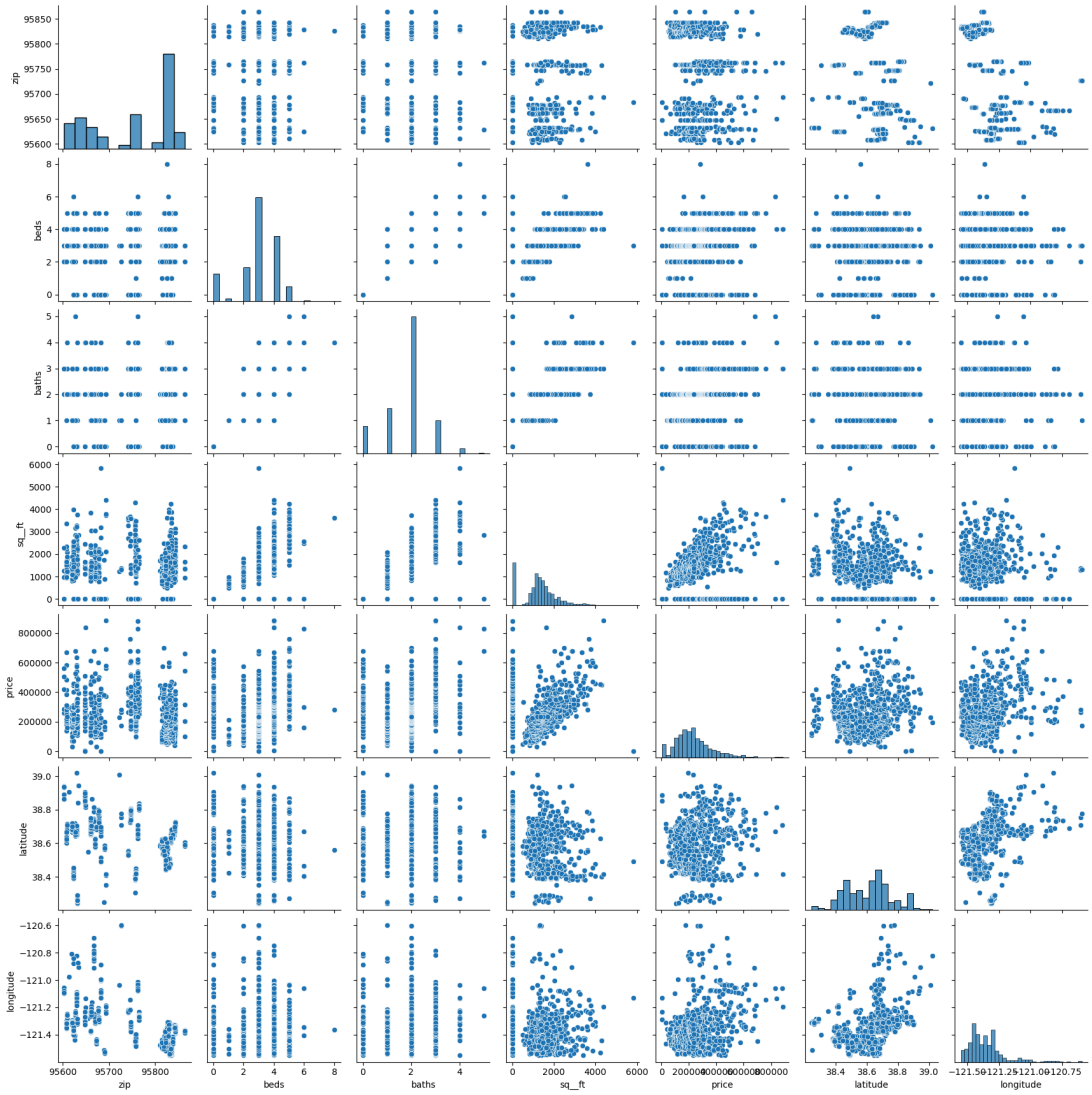
Seaborn

https://seaborn.pydata.org/ (https://seaborn.pydata.org/)

In [79]:
```
import seaborn

import warnings
warnings.filterwarnings('ignore')
```

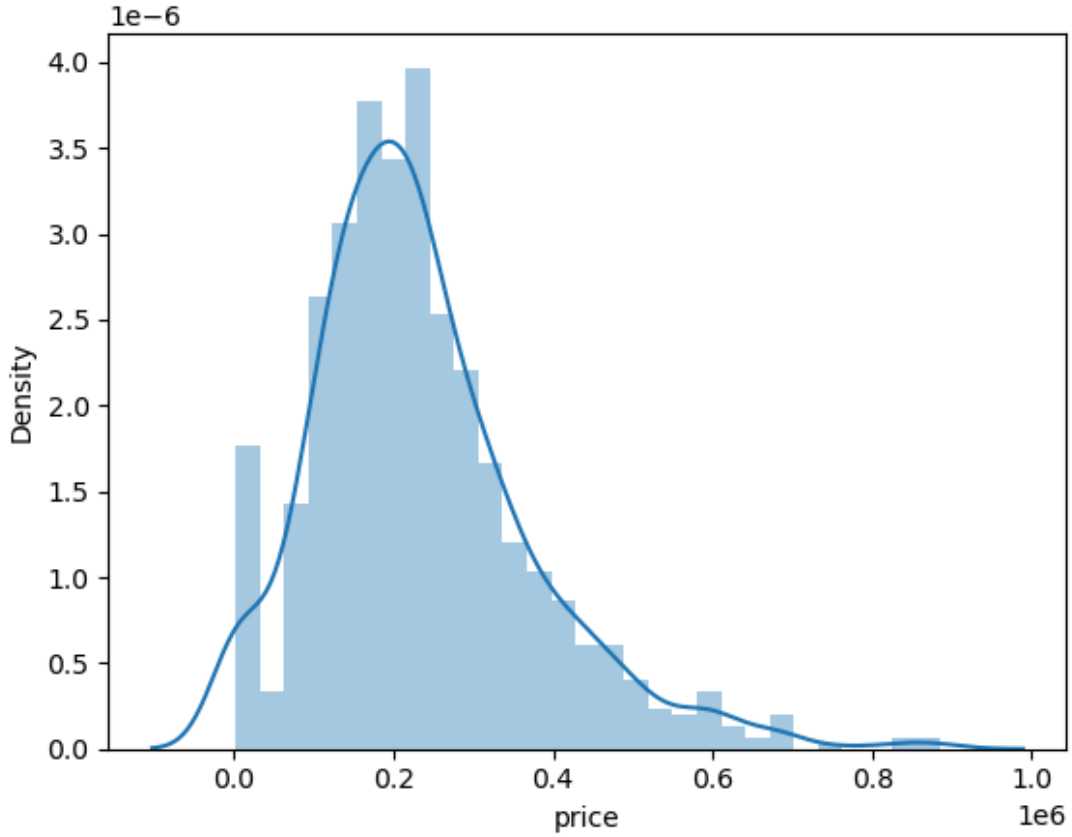
In [80]:
```
%matplotlib inline

seaborn.pairplot(realestate)
```

Out[80]: <seaborn.axisgrid.PairGrid at 0x2a4cd91c210>



In [81]:
```
price = realestate["price"]
seaborn.distplot(price)
```

Out[81]:
```
<Axes: xlabel='price', ylabel='Density'>
```



2.2. Películas

Vamos a plantear un ejemplo con una tabla de películas:

In [ ]:
```
```

In [82]:

```
# Carga de un dataframe desde un archivo csv

peliculas = pd.read_csv("data/film.csv", encoding="latin1", delimiter=";

# Observa los parámetros usados: la codificación y el delimitador

peliculas
```

Out[82]:

	Year	Length	Title	Subject	Actor	Actress	Director	Popularity	Awa
0	1990	111.0	Tie Me Up! Tie Me Down!	Comedy	Banderas, Antonio	Abril, Victoria	Almodóvar, Pedro	68.0	
1	1991	113.0	High Heels	Comedy	Bosé, Miguel	Abril, Victoria	Almodóvar, Pedro	68.0	
2	1983	104.0	Dead Zone, The	Horror	Walken, Christopher	Adams, Brooke	Cronenberg, David	79.0	
3	1979	122.0	Cuba	Action	Connery, Sean	Adams, Brooke	Lester, Richard	6.0	
4	1978	94.0	Days of Heaven	Drama	Gere, Richard	Adams, Brooke	Malick, Terrence	14.0	
...	...	...	...	...	...	...	...	...	
1654	1932	226.0	Shadow of the Eagle, The	Action	Wayne, John	NaN	NaN	19.0	
1655	1989	103.0	Blood & Guns	Action	Welles, Orson	NaN	NaN	43.0	
1656	1988	78.0	Hot Money	Drama	Welles, Orson	NaN	NaN	19.0	
1657	1977	75.0	Comedy Tonight	Comedy	Williams, Robin	NaN	NaN	18.0	
1658	1991	65.0	Robin Williams	Comedy	Williams, Robin	NaN	NaN	4.0	

1659 rows × 9 columns

In [83]:
```
# Si un dataframe es muy largo, puede interesarnos ver sólo un fragmento

peliculas.head()
```

Out[83]:

	Year	Length	Title	Subject	Actor	Actress	Director	Popularity	Awards
0	1990	111.0	Tie Me Up! Tie Me Down!	Comedy	Banderas, Antonio	Abril, Victoria	Almodóvar, Pedro	68.0	No
1	1991	113.0	High Heels	Comedy	Bosé, Miguel	Abril, Victoria	Almodóvar, Pedro	68.0	No
2	1983	104.0	Dead Zone, The	Horror	Walken, Christopher	Adams, Brooke	Cronenberg, David	79.0	No
3	1979	122.0	Cuba	Action	Connery, Sean	Adams, Brooke	Lester, Richard	6.0	No
4	1978	94.0	Days of Heaven	Drama	Gere, Richard	Adams, Brooke	Malick, Terrence	14.0	No

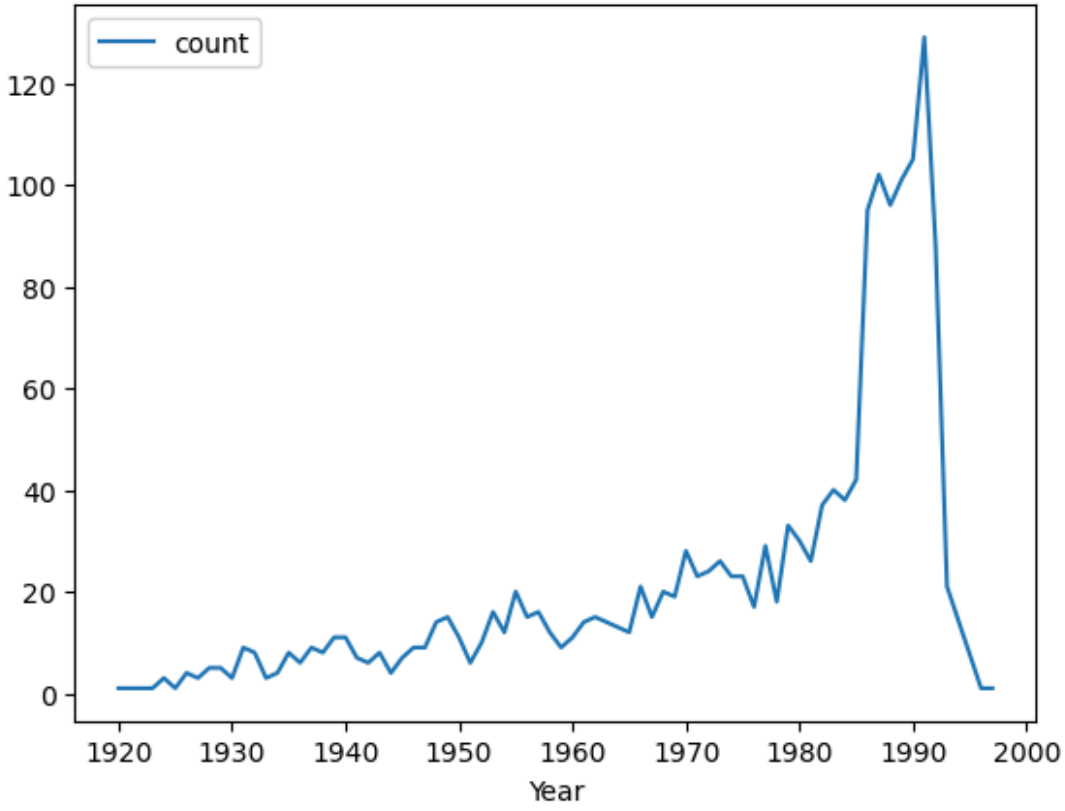
In [84]:

```
import matplotlib.pyplot as plt
%matplotlib inline

peliculas_por_annos = peliculas["Year"].value_counts().reset_index()

print(peliculas_por_annos)

peliculas_por_annos.sort_values("Year", inplace=True)
print(peliculas_por_annos)

peliculas_por_annos.plot(x="Year", y="count")
```

```
    Year  count
0   1991    129
1   1990    105
2   1987    102
3   1989    101
4   1988     96
..   ...    ...
69  1925      1
70  1996      1
71  1923      1
72  1997      1
73  1920      1

[74 rows x 2 columns]
    Year  count
73  1920      1
71  1923      1
65  1924      3
69  1925      1
64  1926      4
..   ...    ...
0   1991    129
6   1992     88
22  1993     21
70  1996      1
72  1997      1

[74 rows x 2 columns]
```

Out[84]:

```
<Axes: xlabel='Year'>
```



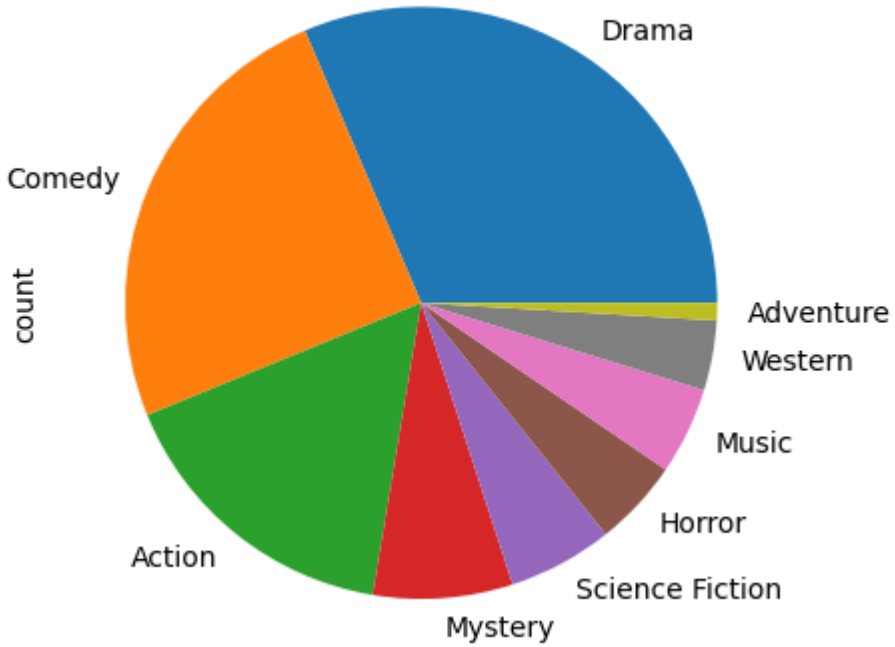
In [85]:
```
peliculas_1990 = peliculas[peliculas["Year"] == 1990]

peliculas_1990.head()
```

Out[85]:

	Year	Length	Title	Subject	Actor	Actress	Director	Popularity	Awar
0	1990	111.0	Tie Me Up! Tie Me Down!	Comedy	Banderas, Antonio	Abril, Victoria	Almodóvar, Pedro	68.0	
9	1990	149.0	Camille Claudel	Drama	Depardieu, Gérard	Adjani, Isabelle	Nuytten, Bruno	32.0	
25	1990	97.0	Zandalee	Drama	Cage, Nicolas	Anderson, Erika	Pillsbury, Sam	80.0	
118	1990	123.0	Misery	Horror	Caan, James	Bates, Kathy	Reiner, Rob	48.0	
121	1990	101.0	Act of Piracy	Mystery	Busey, Gary	Bauer, Belinda	NaN	74.0	

In [86]:
```
peliculas_1990 = peliculas[peliculas["Year"] == 1990]

peliculas_1990_tipos = peliculas_1990["Subject"].value_counts()

peliculas_1990_tipos.plot.pie()
```

Out[86]: <Axes: ylabel='count'>



Referencias

Uno de los objetivos de pandas es dar una funcionalidad de análisis de datos parecida a la proporcionada por R. Podemos encontrar una comparación en la documentación de pandas: https://pandas.pydata.org/pandas-docs/stable/comparison_with_r.html (https://pandas.pydata.org/pandas-docs/stable/comparison_with_r.html)

- http://pandas.pydata.org/pandas-docs/stable/ (http://pandas.pydata.org/pandas-docs/stable/)

3. Series

Ejemplo 1 de series

Una serie es un caso particular de un dataframe, en el que únicamente tenemos una columna de datos.

In [87]:
```
# Ejemplo 1, trivial

notas = [3.5, 5.0, 4.1, 4.5, 6.8, 8.0, 6.0, 8.9, 7.5, 9.0, 10.0, 8.5, 9.

serie_de_notas = pd.Series(notas)

print(serie_de_notas)
```

```
0      3.5
1      5.0
2      4.1
3      4.5
4      6.8
5      8.0
6      6.0
7      8.9
8      7.5
9      9.0
10    10.0
11     8.5
12     9.0
13     8.5
14     9.5
dtype: float64
```

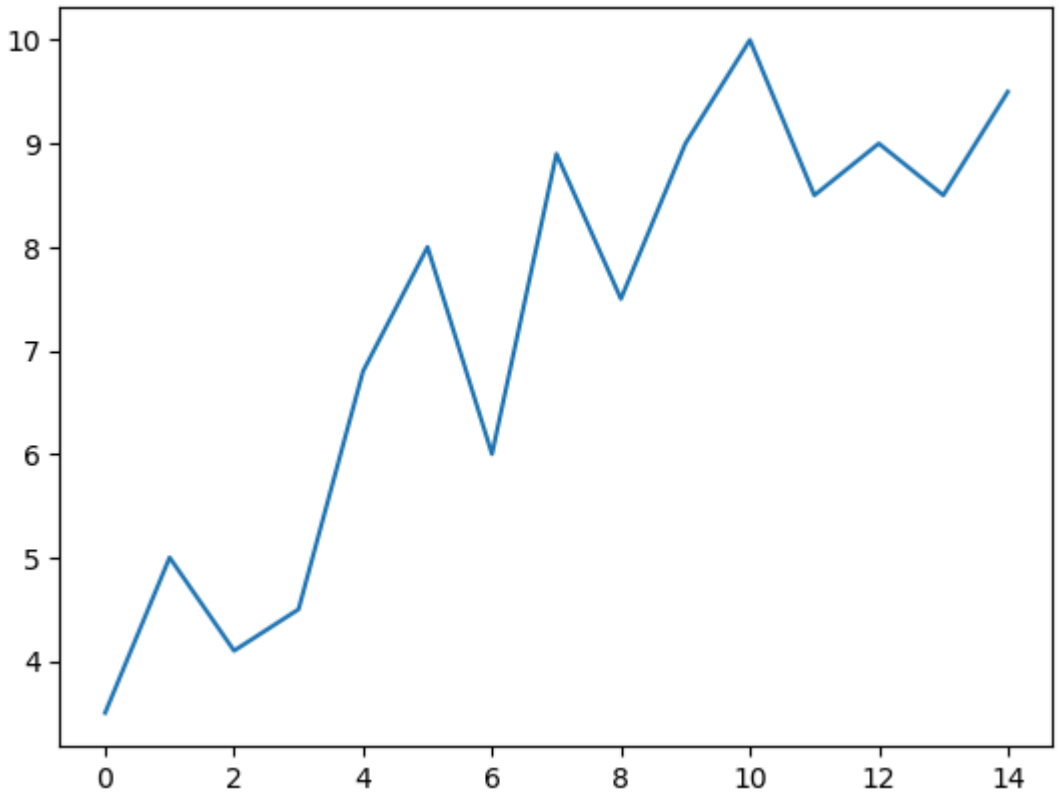
In [88]:
```
print("La primera nota: ", serie_de_notas[0])
num_notas =  serie_de_notas.count()
print("Cuantas notas tengo: ", num_notas)
print("La última nota: ", serie_de_notas[num_notas-1])

serie_de_notas.describe()
```

```
La primera nota:  3.5
Cuantas notas tengo:  15
La última nota:  9.5
```

Out[88]:
```
count    15.000000
mean      7.253333
std       2.131353
min       3.500000
25%       5.500000
50%       8.000000
75%       8.950000
max      10.000000
dtype: float64
```

In [89]:
```
serie_de_notas.plot()
```

Out[89]: <Axes: >



In [90]:
```
serie_de_notas.to_csv("./data/notas.csv", sep = ';')
```

He aquí la imagen del archivo generado, `notas.csv` , visto desde *excel*:

A	B
	0
0	3.5
1	5.0
2	4.1
3	4.5
4	6.8
5	8.0
6	6.0
7	8.9
8	7.5
9	9.0
10	10.0
11	8.5
12	9.0
13	8.5
14	9.5

In [91]:
```
# Lectura de la tabla de datos, seleccionando sólo la segunda columna:

notas_cargadas = pd.Series(pd.read_csv("./data/notas.csv", sep = ';', he
notas_cargadas
```

Out[91]:
```
0      3.5
1      5.0
2      4.1
3      4.5
4      6.8
5      8.0
6      6.0
7      8.9
8      7.5
9      9.0
10    10.0
11     8.5
12     9.0
13     8.5
14     9.5
Name: 1, dtype: float64
```

Ejemplo 2 de series, con un índice definido por el usuario

In [92]:

```
import matplotlib.pyplot as plt

# https://www.ine.es
madrid = {
    1998: 5091336,
    1999: 5145325,
    2000: 5205408,
    2001: 5372433,
    2002: 5527152,
    2003: 5718942,
    2004: 5804829,
    2005: 5964143,
    2006: 6008183,
    2007: 6081689,
    2008: 6271638,
    2009: 6386932,
    2010: 6458684,
    2011: 6489680,
    2012: 6498560,
    2013: 6495551,
    2014: 6454440,
    2015: 6436996,
    2016: 6466996,
    2017: 6507184,
    2018: 6578079,
    2019: 6663394,
    2020: 6779888
}

madrid_pob = pd.Series(madrid)

print(madrid_pob)
```

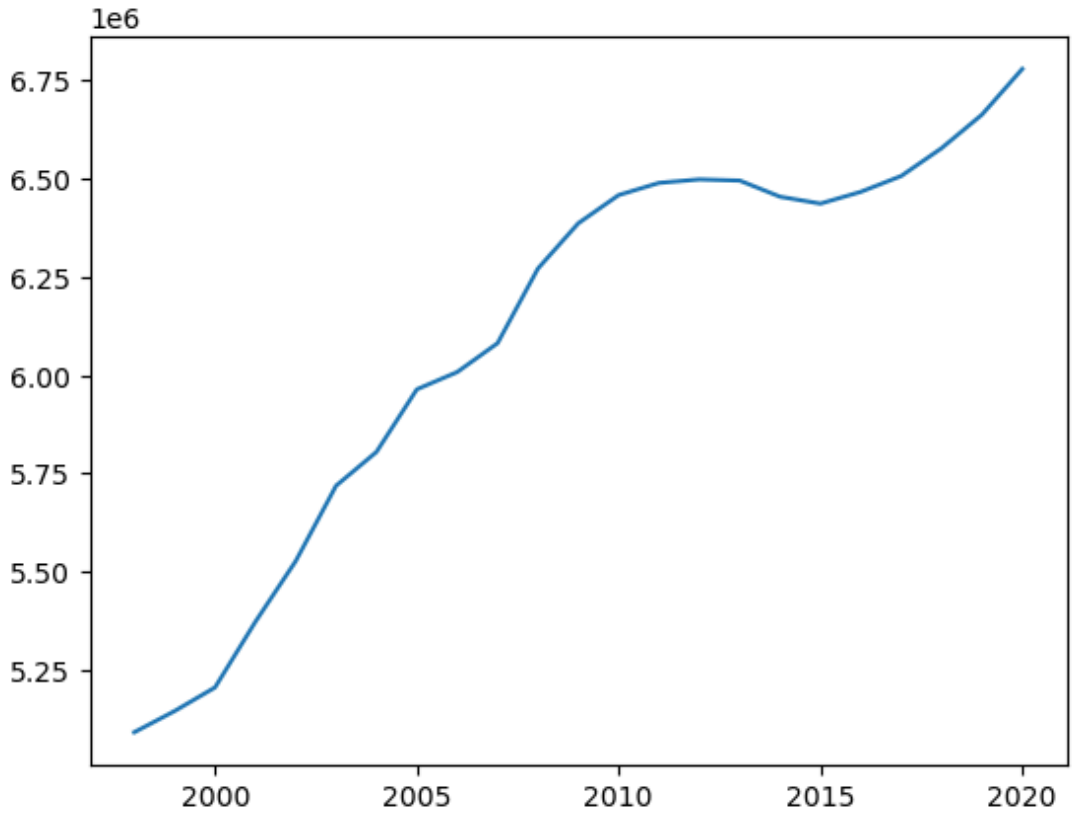
```
1998    5091336
1999    5145325
2000    5205408
2001    5372433
2002    5527152
2003    5718942
2004    5804829
2005    5964143
2006    6008183
2007    6081689
2008    6271638
2009    6386932
2010    6458684
2011    6489680
2012    6498560
2013    6495551
2014    6454440
2015    6436996
2016    6466996
2017    6507184
2018    6578079
2019    6663394
2020    6779888
dtype: int64
```

In [93]:
```
madrid_pob.describe()
# El año se asume como índice, y no se tiene en cuenta para las estadíst
```

Out[93]:
```
count    2.300000e+01
mean     6.104672e+06
std      5.266173e+05
min      5.091336e+06
25%      5.761886e+06
50%      6.386932e+06
75%      6.492616e+06
max      6.779888e+06
dtype: float64
```

In [94]:
```
madrid_pob.plot()
```

Out[94]: <Axes: >



In [95]:
```
# Crecimiento medio:

crecimiento_medio = (madrid_pob[2020] - madrid_pob[1998]) / (2020 - 1998
print(crecimiento_medio)
```

```
76752.36363636363
```

In [96]:
```
# Buscamos el año de mayor crecimiento:

# Calculamos en primer lugar la lista de crecimientos en cada año:

crecimientos = [(i, madrid_pob[i+1] - madrid_pob[i]) for i in range(1998

# Seleccionamos el par (año, crecimiento) de mayor crecimiento (segunda
# Y de él, nos interesa en año (primera componente)

max(crecimientos, key=lambda par: par[1])[0]
```

Out[96]: 2002