IMT2200

INTRODUCCIÓN A CIENCIA DE DATOS

2022-2

# LIMPIEZA DE DATOS
# COMBINACIÓN Y AGRUPACIÓN DE DATOS

CLASE 11

# TAREAS COMUNES DE LIMPIEZA DE DATOS

- ✓ Eliminar y renombrar columnas
- ✓ Conversión de tipos de datos
- ✓ Identificación y eliminación/agrupación de duplicados
- ✓ Datos categóricos

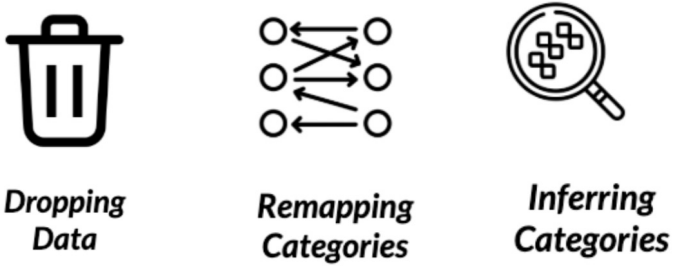
- ✓ Valores faltantes
- ✓ Limpieza de textos
- ✓ Valores fuera de rango

- ○ Validaciones de valores entre campos

# DATOS CATEGÓRICOS

- Variables que sólo pueden adoptar un conjunto acotado de valores.
- Es frecuente codificarlas en categorías numéricas para su uso en modelos de machine learning.
- Pueden existir datos con valores inconsistentes con las categorías definidas, lo cual requiere:
  - Eliminar datos
  - Remapear categorías
  - Inferir y asignar categorías
- La identificación y corrección de inconsistencias es particular a cada dataset.

| | id | price | cancellation_policy |
|---|---|---|---|
| 0 | 14576 | $81,730.00 | moderate |
| 1 | 30950 | $19,615.00 | flexible |
| 2 | 47936 | $44,952.00 | flexible |
| 3 | 49392 | $44,952.00 | moderate |
| 4 | 50466 | $69,471.00 | flexible |

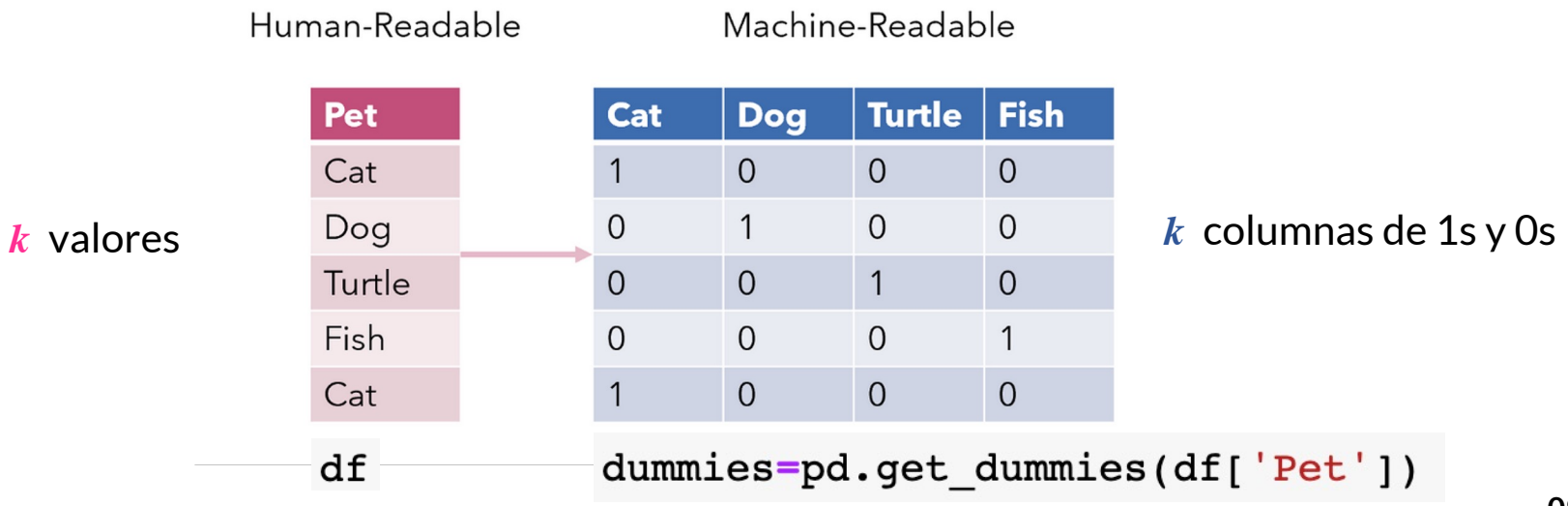



**Funciones útiles:** `pd.unique()`, `pd.isin()`, `pd.map()`, `pd.drop()`, `pd.replace()`, `string methods`

09:54

# CÁLCULO DE VARIABLES INDICATIVAS O “DUMMIES”

- Para ciertas aplicaciones de machine learning, se **requiere transformar variables categóricas en una matriz de “dummies”** o “indicadores”.
- Dummy ➔ variable binaria que indica si
- También llamado: “one hot encoding”
- Es un método utilizado para variables con




| Pet |
|---|
| Cat |
| Dog |
| Turtle |
| Fish |
| Cat |

| Cat | Dog | Turtle | Fish |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 |

# DATOS FALTANTES

- Ocurre cuando no se ha almacenado el valor de una variable en una observación.
- Típicamente se debe a errores humanos o técnicos.

- El valor faltante puede representarse como: **`NA, NaN, 0, …, ., …`** o estar **`vacío`**

- **`pandas`** interpreta los campos vacíos como **`NaN`**
  - *Not a Number*, heredado de **`numpy (np.nan)`**

- Otras representaciones, pueden convertirse a **`np.nan`** usando la función **`df.replace()`**

- Para identificar valores nulos en un DataFrame: **`df.isnull(),df.isna()`** (idénticas)

# DATOS FALTANTES: EJEMPLO

```
1 df=pd.read_csv('data.csv')
```

data.csv

```
col1, col2, col3, col4
100,a, ,5.4
200,b,5.6,NaN
300,c,10.5,100.4
400,,12.2,30.2
500,e,None,40.5
600,f,*,40.5
700,g,45.7,-999
```

| | col1 | col2 | col3 | col4 |
|---|---|---|---|---|
| **0** | 100 | a | | 5.4 |
| **1** | 200 | b | 5.6 | NaN |
| **2** | 300 | c | 10.5 | 100.4 |
| **3** | 400 | NaN | 12.2 | 30.2 |
| **4** | 500 | e | None | 40.5 |
| **5** | 600 | f | * | 40.5 |
| **6** | 700 | g | 45.7 | -999.0 |

```
1 df.isna()
```

| | col1 | col2 | col3 | col4 |
|---|---|---|---|---|
| **0** | False | False | False | False |
| **1** | False | False | False | True |
| **2** | False | False | False | False |
| **3** | False | True | False | False |
| **4** | False | False | False | False |
| **5** | False | False | False | False |
| **6** | False | False | False | False |

# DATOS FALTANTES: EJEMPLO

data.csv

```
col1, col2, col3, col4
100,a, ,5.4
200,b,5.6,NaN
300,c,10.5,100.4
400,,12.2,30.2
500,e,None,40.5
600,f,*,40.5
700,g,45.7,-999
```

```
1  df=pd.read_csv('data.csv')
```

| | col1 | col2 | col3 | col4 |
|---|---|---|---|---|
| **0** | 100 | a | | 5.4 |
| **1** | 200 | b | 5.6 | NaN |
| **2** | 300 | c | 10.5 | 100.4 |
| **3** | 400 | NaN | 12.2 | 30.2 |
| **4** | 500 | e | None | 40.5 |
| **5** | 600 | f | * | 40.5 |
| **6** | 700 | g | 45.7 | -999.0 |

```
1  df=df.replace('*',np.nan)
2  df=df.replace('None',np.nan)
3  df=df.replace(' ',np.nan)
4  df=df.replace(-999,np.nan)
```

```
1  df.isna()
```

| | col1 | col2 | col3 | col4 |
|---|---|---|---|---|
| **0** | False | False | True | False |
| **1** | False | False | False | True |
| **2** | False | False | False | False |
| **3** | False | True | False | False |
| **4** | False | False | True | False |
| **5** | False | False | True | False |
| **6** | False | False | False | True |

# DATOS FALTANTES

- Es importante analizar la data faltante para identificar **problemas de recolección de datos** o potenciales **sesgos.**

- Para encontrar datos nulos: **`pd.isnull(), pd.notnull()`**

- Para rellenar los valores faltantes, se pueden aplicar distintas estrategias, dependiendo de la naturaleza de los datos:
  - Eliminar los registros
  - Un valor constante
  - Un valor constante por columna
  - Valor medio, mediana
  - Interpolación
  - Algoritmos avanzados de imputación.

| Argument | Description |
|---|---|
| `dropna` | Filter axis labels based on whether values for each label have missing data, with varying thresholds for how much missing data to tolerate. |
| `fillna` | Fill in missing data with some value or using an interpolation method such as `'ffill'` or `'bfill'`. |
| `isnull` | Return boolean values indicating which values are missing/NA. |
| `notnull` | Negation of `isnull`. |

# DATOS FALTANTES

- El método de reemplazo a utilizar depende del conocimiento que se tenga de los datos, y las características de los datos faltantes.

- Aleatorio ⇒ missing completely at random (MCAR)
  - Los datos faltantes están distribuidos de forma totalmente aleatoria
  - Ej: errores en el ingreso de datos

- Semi-aleatorio ⇒ Missing at random (MAR)
  - Hay una relación sistemática entre datos faltantes, y *otra variable* observada

- No aleatorio ⇒ Missing not at random (MNAR)
  - Hay una relación sistemática entre datos faltantes, y valores no observados.
  - Ej: datos de satisfacción de clientes ⇒ no hay datos para clientes insatisfechos

# LIMPIEZA DE TEXTOS

- Muchos datos tipo **`str`** pueden requerir un proceso de limpieza o estandarización.
- Python ofrece múltiples funciones para la manipulación de strings.
- **`pandas`** implementa muchas de estas funciones en forma **vectorizada**
  - ➢ Se aplican sobre toda una Serie o columna del DataFrame
  - ➢ Omiten datos NaN

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.str.html

## String methods

Series and Index are equipped with a set of string processing methods that make it easy to operate on each element of the array. Perhaps most importantly, these methods exclude missing/NA values automatically. These are accessed via the `str` attribute and generally have names matching the equivalent (scalar) built-in string methods:

```
In [24]: s = pd.Series(
   ....:     ["A", "B", "C", "Aaba", "Baca", np.nan, "CABA", "dog", "cat"], dtype="string"
   ....: )
   ....:

In [25]: s.str.lower()
Out[25]:
0       a
1       b
2       c
3    aaba
4    baca
5    <NA>
6    caba
7     dog
8     cat
dtype: string
```

# LIMPIEZA DE TEXTOS

*Table 7-3. Python built-in string methods*

| Method | Description |
|---|---|
| `count` | Return the number of non-overlapping occurrences of substring in the string. |
| `endswith` | Returns `True` if string ends with suffix. |
| `startswith` | Returns `True` if string starts with prefix. |
| `join` | Use string as delimiter for concatenating a sequence of other strings. |
| `index` | Return position of first character in substring if found in the string; raises `ValueError` if not found. |
| `find` | Return position of first character of *first* occurrence of substring in the string; like `index`, but returns –1 if not found. |
| `rfind` | Return position of first character of *last* occurrence of substring in the string; returns –1 if not found. |

# LIMPIEZA DE TEXTOS

| | |
|---|---|
| `replace` | Replace occurrences of string with another string. |
| `strip`, `rstrip`, `lstrip` | Trim whitespace, including newlines; equivalent to `x.strip()` (and `rstrip, lstrip`, respectively) for each element. |
| `split` | Break string into list of substrings using passed delimiter. |
| `lower` | Convert alphabet characters to lowercase. |
| `upper` | Convert alphabet characters to uppercase. |
| `casefold` | Convert characters to lowercase, and convert any region-specific variable character combinations to a common comparable form. |
| `ljust, rjust` | Left justify or right justify, respectively; pad opposite side of string with spaces (or some other fill character) to return a string with a minimum width. |

# LIMPIEZA DE TEXTOS

- Muchos datos tipo `str` pueden requerir un proceso de limpieza o estandarización.
- Python ofrece múltiples funciones para la manipulación de strings.
- `pandas` implementa muchas de estas funciones en forma **vectorizada**
  - ➢ Se aplican sobre toda una Serie o columna del DataFrame
  - ➢ Omiten datos NaN

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.str.html

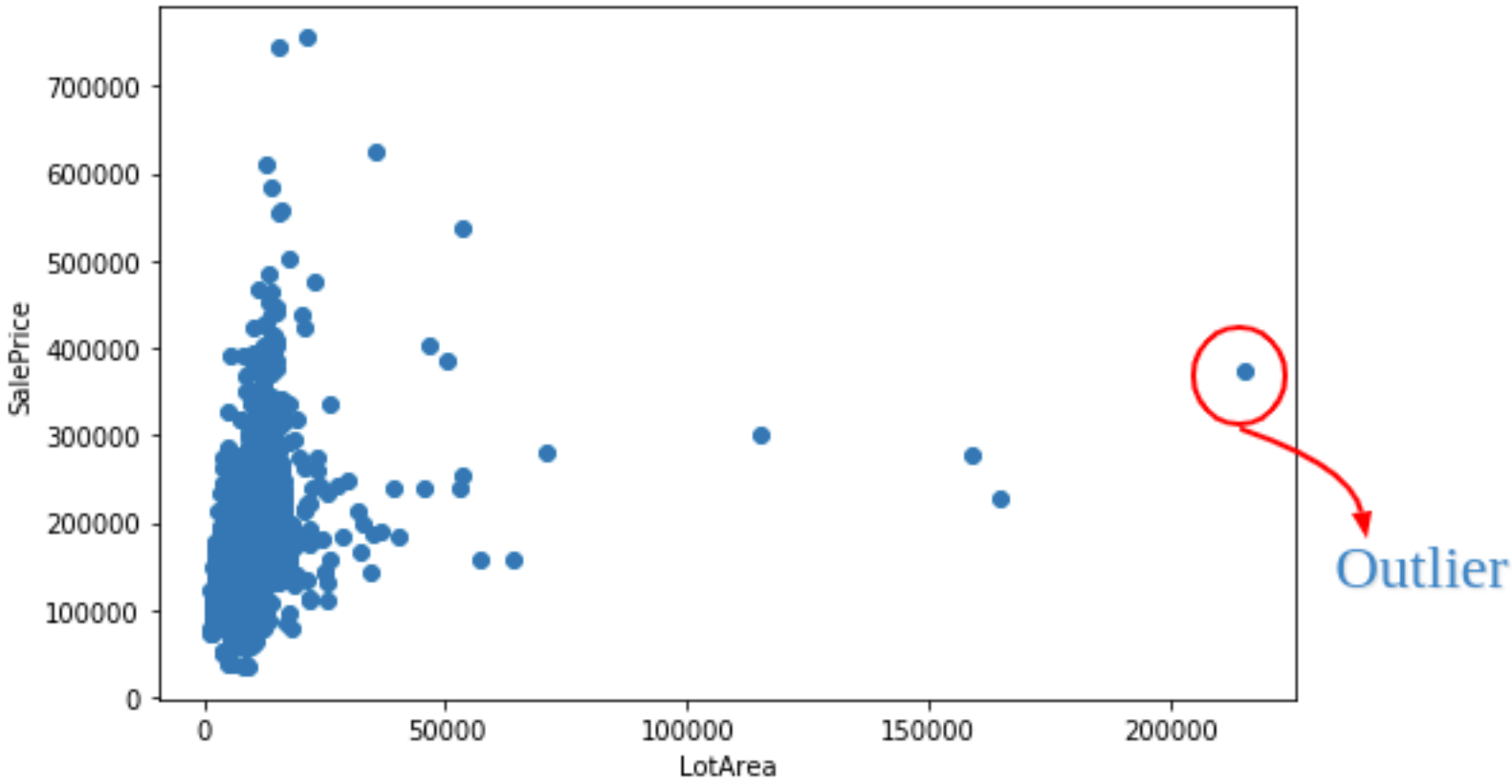
| | |
|---|---|
| `count` | Return the number of non-overlapping occurrences of substring in the string. |
| `endswith` | Returns `True` if string ends with suffix. |
| `startswith` | Returns `True` if string starts with prefix. |
| `join` | Use string as delimiter for concatenating a sequence of other strings. |
| `index` | Return position of first character in substring if found in the string; raises `ValueError` if not found. |
| `find` | Return position of first character of *first* occurrence of substring in the string; like `index`, but returns –1 if not found. |
| `rfind` | Return position of first character of *last* occurrence of substring in the string; returns –1 if not found. |
| `replace` | Replace occurrences of string with another string. |
| `strip, rstrip, lstrip` | Trim whitespace, including newlines; equivalent to `x.strip()` (and `rstrip, lstrip`, respectively) for each element. |
| `split` | Break string into list of substrings using passed delimiter. |
| `lower` | Convert alphabet characters to lowercase. |
| `upper` | Convert alphabet characters to uppercase. |
| `casefold` | Convert characters to lowercase, and convert any region-specific variable character combinations to a common comparable form. |
| `ljust, rjust` | Left justify or right justify, respectively; pad opposite side of string with spaces (or some other fill character) to return a string with a minimum width. |

# VALORES FUERA DE RANGO

- Las variables numéricas de un dataset pueden contener valores fuera de rango. Esto incluye:

  - Valores que podemos catalogar como **inverosímiles** a partir de nuestro conocimiento de los datos, y las entidades reales que representan.

# VALORES FUERA DE RANGO



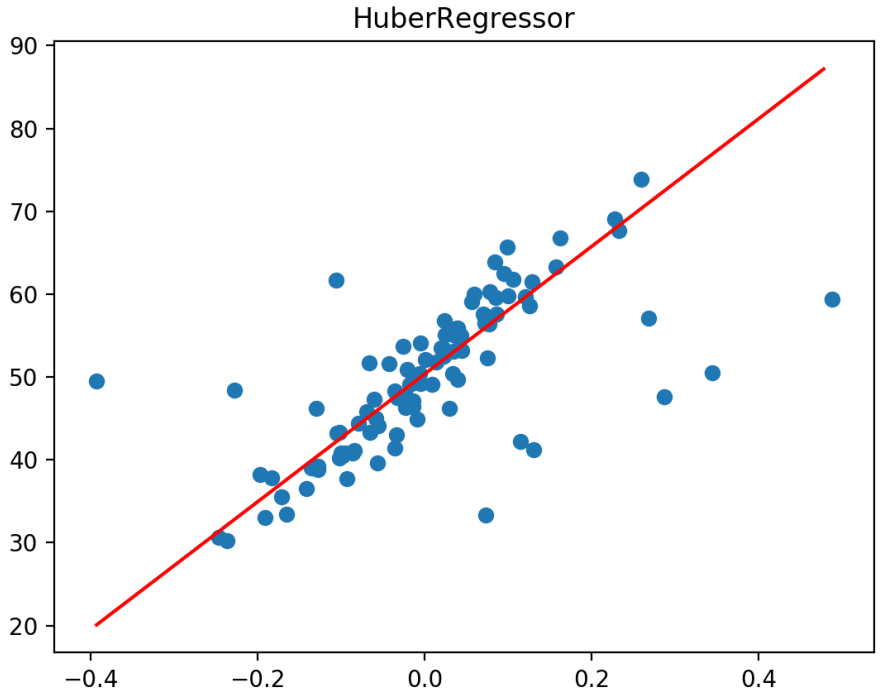
# VALORES FUERA DE RANGO

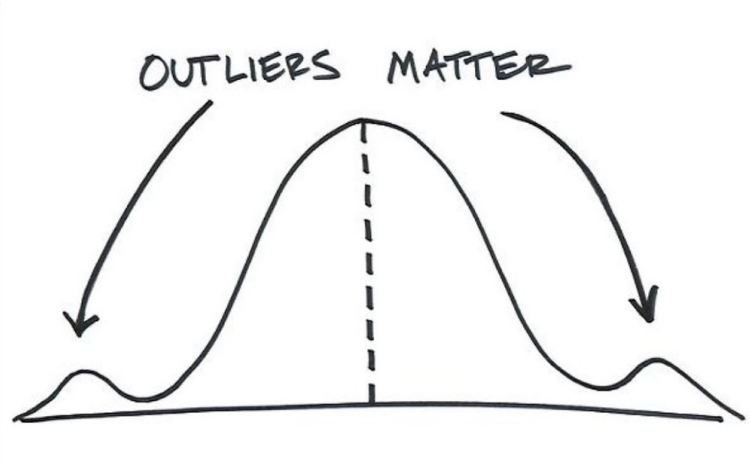
- Las variables numéricas de un dataset pueden contener valores fuera de rango. Esto incluye:

  - Valores que podemos catalogar como **inverosímiles** a partir de nuestro conocimiento de los datos, y las entidades reales que representan.

  - Valores que pueden (o no) se legítimos, pero que **escapan de la norma de la serie de datos.**
  - En este caso se requiere un análisis estadístico más detallado para determinar qué es realmente un valor fuera de rango.

# VALORES FUERA DE RANGO




09:54

# OTRAS TÉCNICAS ÚTILES PARA LA LIMPIEZA DE DATOS

- Validación de valores entre distintos campos
  - Uso de múltiples columnas del dataset para validar la integridad de los datos y detectar inconsistencias/errores.

  - Ejemplos:
    - edad vs. fecha de nacimiento
    - total población vs. subtotales por sexo/edad/región, etc.

- Chequear uniformidad de los datos
  - Formatos
  - Unidades

# COMBINACIÓN Y AGRUPACIÓN DE DATASETS

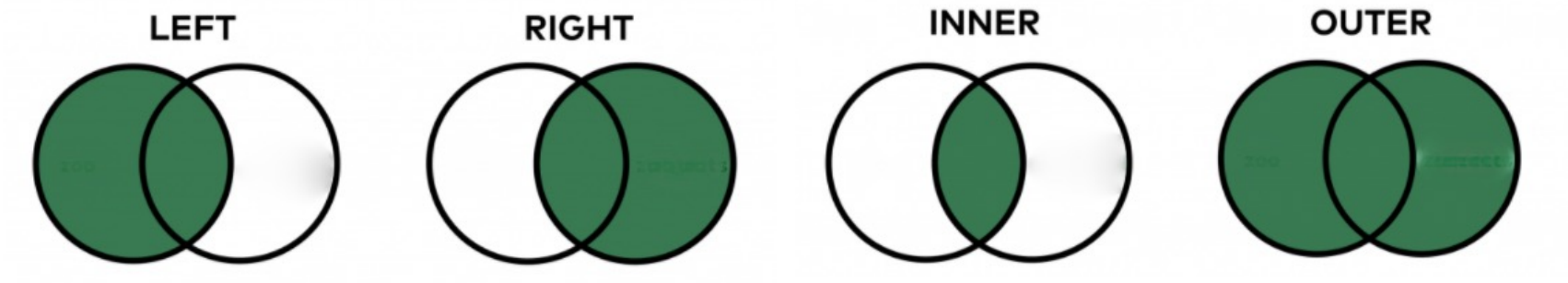
- Normalmente, los datos de interés pueden estar contenidos en dos o más tablas que deben ser combinadas, ya sea horizontal o verticalmente.
  - **`pd.concat`** ➔ concatena o apila ("stack") distintos objetos a lo largo de un eje determinado
  - **`pd.merge`** ➔ conecta filas en distintos DataFrames en base a uno o más campos o keys.
    - Función más general para todo tipo de combinación de DataFrames
    - Análogo a **`join`** en lenguaje de bases de datos (Ej.J SQL)

- También puede ser necesario agrupar por columnas o filas.
  - **`pd.groupby()`**
  - **`pd.pivot_table()`**
  - **`Pd.crosstable()`**

# COMBINACIÓN DE DATASETS: TIPOS DE UNIONES

Normalmente, los DataFrames a combinar contienen distintos conjuntos de valores para el key o campo de unión.

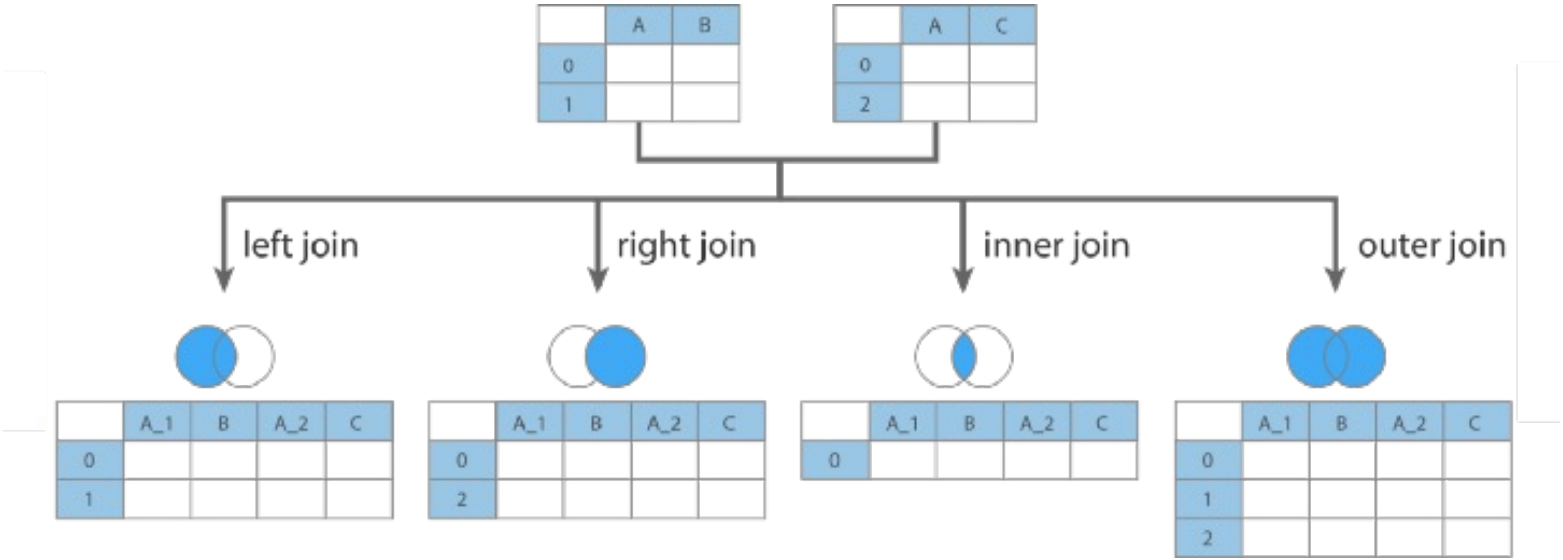
Dependiendo del conjunto de keys que se utilice como referencia para construir el DataFrame combinado, usaremos distintos tipos de uniones.



# COMBINACIÓN DE DATASETS: TIPOS DE UNIONES

Normalmente, los DataFrames a combinar contienen distintos conjuntos de valores para el key o campo de unión.

Dependiendo del conjunto de keys que se utilice como referencia para construir el DataFrame combinado, usaremos distintos tipos de uniones.



# COMBINACIÓN DE DATASETS: CONCATENACIÓN

| | colA | colB | colC | colD |
|---|---|---|---|---|
| 0 | A0 | B0 | C0 | D0 |
| 1 | A1 | B1 | C1 | D1 |
| 2 | A2 | B2 | C2 | D2 |
| 3 | A3 | B3 | C3 | D3 |

**df1**

| | colA | colB | colC | colD |
|---|---|---|---|---|
| 4 | A4 | B4 | C4 | D4 |
| 5 | A5 | B5 | C5 | D5 |
| 6 | A6 | B6 | C6 | D6 |
| 7 | A7 | B7 | C7 | D7 |

**df2**

| | colA | colB | colC | colD |
|---|---|---|---|---|
| 0 | A8 | B8 | C8 | D8 |
| 1 | A9 | B9 | C9 | D9 |
| 2 | A10 | B10 | C10 | D10 |
| 3 | A11 | B11 | C11 | D11 |

**df3**

**axis=0**

```
1 pd.concat([df1, df2, df3],axis=0)
```

| | colA | colB | colC | colD |
|---|---|---|---|---|
| 0 | A0 | B0 | C0 | D0 |
| 1 | A1 | B1 | C1 | D1 |
| 2 | A2 | B2 | C2 | D2 |
| 3 | A3 | B3 | C3 | D3 |
| 3 | A4 | B4 | C4 | D4 |
| 4 | A5 | B5 | C5 | D5 |
| 5 | A6 | B6 | C6 | D6 |
| 6 | A7 | B7 | C7 | D7 |
| 0 | A8 | B8 | C8 | D8 |
| 1 | A9 | B9 | C9 | D9 |
| 2 | A10 | B10 | C10 | D10 |
| 3 | A11 | B11 | C11 | D11 |

# COMBINACIÓN DE DATASETS: CONCATENACIÓN

| | colA | colB | colC | colD |
|---|---|---|---|---|
| 0 | A0 | B0 | C0 | D0 |
| 1 | A1 | B1 | C1 | D1 |
| 2 | A2 | B2 | C2 | D2 |
| 3 | A3 | B3 | C3 | D3 |

df1

| | colA | colB | colC | colD |
|---|---|---|---|---|
| 4 | A4 | B4 | C4 | D4 |
| 5 | A5 | B5 | C5 | D5 |
| 6 | A6 | B6 | C6 | D6 |
| 7 | A7 | B7 | C7 | D7 |

df2

| | colA | colB | colC | colD |
|---|---|---|---|---|
| 0 | A8 | B8 | C8 | D8 |
| 1 | A9 | B9 | C9 | D9 |
| 2 | A10 | B10 | C10 | D10 |
| 3 | A11 | B11 | C11 | D11 |

df3

axis=1

```
1  pd.concat([df1, df2, df3],axis=1)
```

| | colA | colB | colC | colD | colA | colB | colC | colD | colA | colB | colC | colD |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | A0 | B0 | C0 | D0 | NaN | NaN | NaN | NaN | A8 | B8 | C8 | D8 |
| 1 | A1 | B1 | C1 | D1 | NaN | NaN | NaN | NaN | A9 | B9 | C9 | D9 |
| 2 | A2 | B2 | C2 | D2 | NaN | NaN | NaN | NaN | A10 | B10 | C10 | D10 |
| 3 | A3 | B3 | C3 | D3 | A4 | B4 | C4 | D4 | A11 | B11 | C11 | D11 |
| 4 | NaN | NaN | NaN | NaN | A5 | B5 | C5 | D5 | NaN | NaN | NaN | NaN |
| 5 | NaN | NaN | NaN | NaN | A6 | B6 | C6 | D6 | NaN | NaN | NaN | NaN |
| 6 | NaN | NaN | NaN | NaN | A7 | B7 | C7 | D7 | NaN | NaN | NaN | NaN |

# COMBINACIÓN DE DATASETS: CONCATENACIÓN

| | colA | colB | colC | colD |
|---|---|---|---|---|
| 0 | A0 | B0 | C0 | D0 |
| 1 | A1 | B1 | C1 | D1 |
| 2 | A2 | B2 | C2 | D2 |
| 3 | A3 | B3 | C3 | D3 |

df1

| | colA | colB | colC | colD |
|---|---|---|---|---|
| 4 | A4 | B4 | C4 | D4 |
| 5 | A5 | B5 | C5 | D5 |
| 6 | A6 | B6 | C6 | D6 |
| 7 | A7 | B7 | C7 | D7 |

df2

| | colA | colB | colC | colD |
|---|---|---|---|---|
| 0 | A8 | B8 | C8 | D8 |
| 1 | A9 | B9 | C9 | D9 |
| 2 | A10 | B10 | C10 | D10 |
| 3 | A11 | B11 | C11 | D11 |

df3

axis=1

```
1 pd.concat([df1, df2, df3],axis=1)
```

| | colA | colB | colC | colD | colA | colB | colC | colD | colA | colB | colC | colD |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | A0 | B0 | C0 | D0 | NaN | NaN | NaN | NaN | A8 | B8 | C8 | D8 |
| 1 | A1 | B1 | C1 | D1 | NaN | NaN | NaN | NaN | A9 | B9 | C9 | D9 |
| 2 | A2 | B2 | C2 | D2 | NaN | NaN | NaN | NaN | A10 | B10 | C10 | D10 |
| 3 | A3 | B3 | C3 | D3 | A4 | B4 | C4 | D4 | A11 | B11 | C11 | D11 |
| 4 | NaN | NaN | NaN | NaN | A5 | B5 | C5 | D5 | NaN | NaN | NaN | NaN |
| 5 | NaN | NaN | NaN | NaN | A6 | B6 | C6 | D6 | NaN | NaN | NaN | NaN |
| 6 | NaN | NaN | NaN | NaN | A7 | B7 | C7 | D7 | NaN | NaN | NaN | NaN |

```
1 pd.concat([df1, df2, df3],axis=1,join='inner')
```

| | colA | colB | colC | colD | colA | colB | colC | colD | colA | colB | colC | colD |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | A3 | B3 | C3 | D3 | A4 | B4 | C4 | D4 | A11 | B11 | C11 | D11 |

09:54

# COMBINACIÓN DE DATASETS: CONCAT

**pandas.concat(*objs, axis=0, join='outer', ignore_index=False, keys=None, levels=None, names=None, verify_integrity=False, sort=False, copy=True*)** [source]

Concatenate pandas objects along a particular axis with optional set logic along the other axes.

Can also add a layer of hierarchical indexing on the concatenation axis, which may be useful if the labels are the same (or overlapping) on the passed axis number.

**Parameters:** **objs : *a sequence or mapping of Series or DataFrame objects***

If a mapping is passed, the sorted keys will be used as the *keys* argument, unless it is passed, in which case the values will be selected (see below). Any None objects will be dropped silently unless they are all None in which case a ValueError will be raised.

**axis : *{0/'index', 1/'columns'}, default 0***

The axis to concatenate along.

**join : *{'inner', 'outer'}, default 'outer'***

How to handle indexes on other axis (or axes).

# COMBINACIÓN DE DATASETS: MERGE

```
DataFrame.merge(right, how='inner', on=None, left_on=None, right_on=None,
left_index=False, right_index=False, sort=False, suffixes=('_x', '_y'), copy=True,
indicator=False, validate=None)                                          [source]
```

| | Rut | DV | Edad |
|---|---|---|---|
| 0 | 18015713 | 1 | 46 |
| 1 | 18017388 | 7 | 66 |
| 2 | 18017935 | 4 | 72 |
| 3 | 18021517 | 6 | 84 |
| 4 | 18048671 | 0 | 68 |
| 5 | 18053032 | 3 | 48 |
| 6 | 18070156 | 4 | 32 |
| 7 | 18081204 | 3 | 80 |
| 8 | 18084804 | 7 | 20 |
| 9 | 18086300 | 2 | 84 |

**clientes**

| | Rut | DV | Fecha | Monto |
|---|---|---|---|---|
| 0 | 18015713 | 1 | 2020-01-10 03:57:47.889908256 | 61256 |
| 1 | 18015713 | 1 | 2020-01-05 01:19:15.963302752 | 49697 |
| 2 | 18017388 | 7 | 2020-01-03 00:00:00.000000000 | 44901 |
| 3 | 18017388 | 7 | 2020-01-09 11:00:33.027522935 | 84978 |
| 4 | 18017935 | 4 | 2020-01-02 03:18:09.908256880 | 45243 |
| 5 | 18017935 | 4 | 2020-01-09 22:14:18.715596330 | 54903 |
| 6 | 18017935 | 4 | 2020-01-11 12:46:14.311926605 | 56797 |
| 7 | 18017935 | 4 | 2020-01-07 18:16:30.825688073 | 28328 |
| 8 | 18017935 | 4 | 2020-01-07 17:50:05.504587156 | 69288 |
| 9 | 18017935 | 4 | 2020-01-09 16:30:49.541284403 | 48746 |

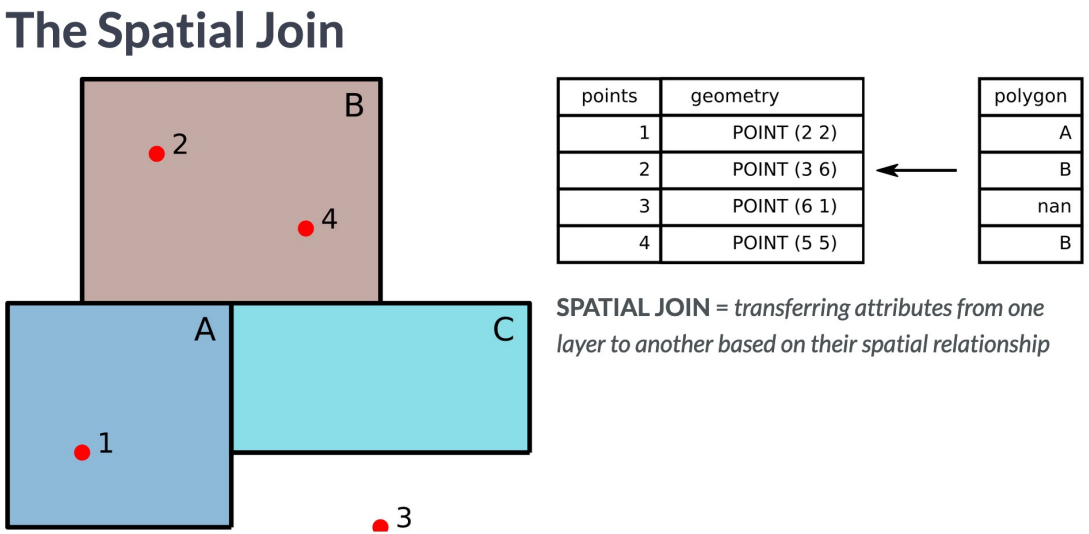
**transacciones**

- Ejemplo: datos de transacciones de varios clientes

# COMBINACIÓN DE DATASETS: UNIONES ESPACIALES (`sjoin`)

- Podemos combinar (join) data de distintos dataframes, y unir atributos en base a una **relación espacial**.

- Ej: left join (mantengo orden y filas del dataframe de de la izquierda)



# COMBINACIÓN DE DATASETS: UNIONES ESPACIALES (`sjoin`)

- Ej: `left join` (mantengo orden y filas del dataframe de la izquierda) ➔ el orden de los argumentos es importante

- Puedo utilizar cualquier método geométrico para definir la relación espacial.


```
joined = geopandas.sjoin(cities,
                         countries[['name', 'geometry']],
                         op="within")
```

```
joined.head()
```

| | name_left | geometry | name_right |
|---|---|---|---|
| 0 | Vatican City | POINT (12.45338654497177 41.90328217996012) | Italy |
| 1 | San Marino | POINT (12.44177015780014 43.936095834768) | Italy |
| 226 | Rome | POINT (12.481312562874 41.89790148509894) | Italy |
| 2 | Vaduz | POINT (9.516669472907267 47.13372377429357) | Austria |