IMT2200

INTRODUCCIÓN A CIENCIA DE DATOS

2022-2

COMBINACIÓN Y AGRUPACIÓN DE DATOS UNIONES ESPACIALES

CLASE 13

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

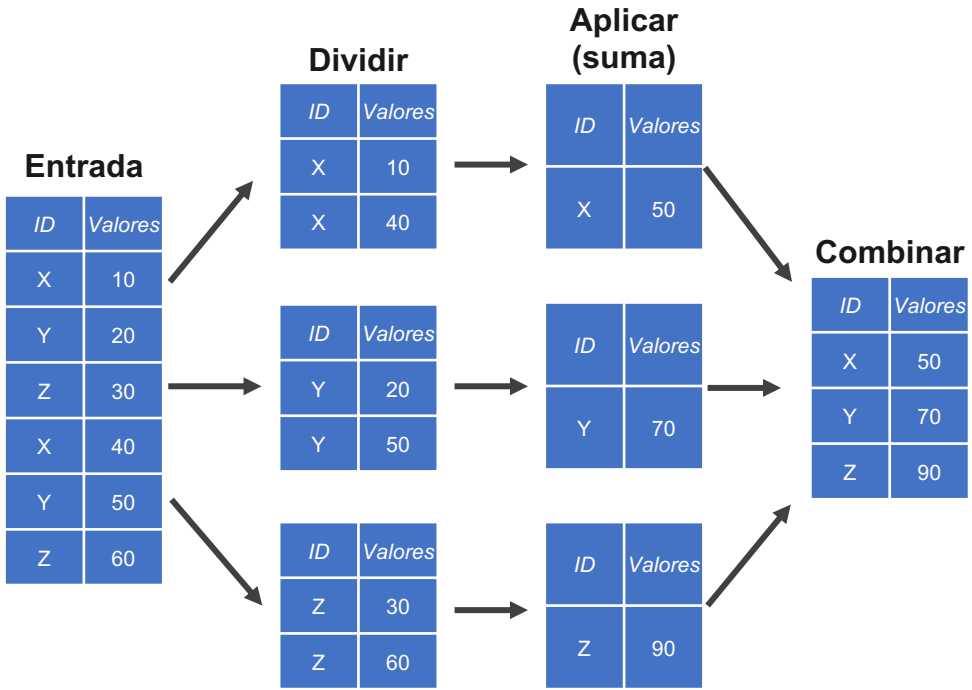
	Rut	DV	Fecha	Monto
0	18015713	1	2020-01-10 03:57:47.889908256	61256
1	18015713	1	2020-01-05 01:19:15.963302752	49697
2	18017388	7	2020-01-03 00:00:00.000000000	44901
3	18017388	7	2020-01-09 11:00:33.027522935	84978
4	18017935	4	2020-01-02 03:18:09.908256880	45243
5	18017935	4	2020-01-09 22:14:18.715596330	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

09:55

ESQUEMA DE OPERACIONES DE AGRUPACIÓN DE DATOS



Fuente: Adaptado de w3resource.com.de Pandas groupby split apply combine. *w3resource.com*

MÉTODO *groupby*

pandas.DataFrame.groupby

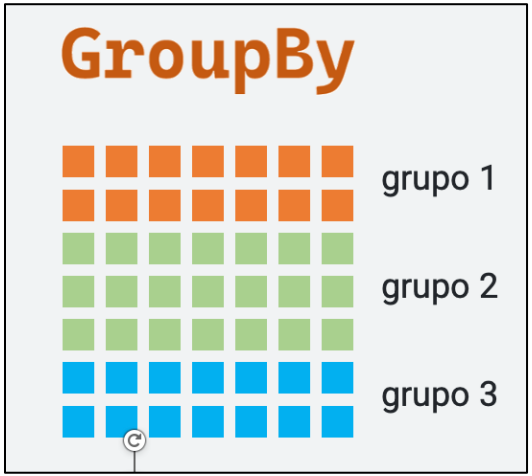
```
DataFrame.groupby(by=None, axis=0, level=None, as_index=True, sort=True,
group_keys=_NoDefault.no_default, squeeze=_NoDefault.no_default,
observed=False, dropna=True)
```

[source]

Group DataFrame using a mapper or by a Series of columns.

A groupby operation involves some combination of splitting the object, applying a function, and combining the results. This can be used to group large amounts of data and compute operations on these groups.

Parameters: **by** : *mapping, function, label, or list of labels*



Returns: **DataFrameGroupBy**

Returns a groupby object that contains information about the groups.

09:55

MÉTODO *groupby*

Para aplicar funciones sobre cada uno de los grupos, se pueden usar distintos métodos:

- Aplicar una función predefinida sobre todas las columnas:

```
compras_clientes.groupby(by=['Rut'],as_index=False).sum()
```

```
compras_clientes.groupby(by=['Rut'],as_index=False).mean()
```

- Aplicar distintas funciones para distintas columnas: `.agg()`

```
compras.groupby(by=['Rut']).agg({'Edad':'min','Monto':['sum','mean']})
```

MÉTODO *groupby*

Para aplicar funciones sobre cada uno de los grupos, se pueden usar distintos métodos:

- Aplicar una función cualquiera: **`.apply()`**

```
1  def as_perc(value, total):
2      return value/float(total)
```

```
1  total=compras_clientes.Monto.sum()
2  compras_clientes[['Monto','Rut']].groupby(by=['Rut']).Monto.apply(as_perc,total=total)
3
```

TABLAS PIVOTE (TABLA DINÁMICA)

Herramienta de resumen de datos encontrada en software de planilla de cálculo.

Consiste en agregar un conjunto de datos a partir de una o más llaves.

Estructurando los datos en forma rectangular con **algunas llaves en el eje de las filas, y otras a lo largo de las columnas**.

MÉTODO *pivot_table*

Principales argumentos

Dataframe

A agregar

Index

Columna o lista de columnas para agrupar los datos

Values

Columnas sobre las que se calcularán valores agregados

Aggfunc

Para definir funciones a aplicar sobre cada grupo

MÉTODO *pivot_table*

pandas.pivot_table

```
pandas.pivot_table(data, values=None, index=None, columns=None,
aggfunc='mean', fill_value=None, margins=False, dropna=True,
margins_name='All', observed=False, sort=True)
```

[source]

Create a spreadsheet-style pivot table as a DataFrame.

The levels in the pivot table will be stored in MultiIndex objects (hierarchical indexes) on the index and columns of the result DataFrame.

Parameters:

data : *DataFrame*

values : *column to aggregate, optional*

index : *column, Grouper, array, or list of the previous*

If an array is passed, it must be the same length as the data. The list can contain any of the other types (except list). Keys to group by on the pivot table index. If an array is passed, it is being used as the same manner as column values.

columns : *column, Grouper, array, or list of the previous*

If an array is passed, it must be the same length as the data. The list can contain any of the other types (except list). Keys to group by on the pivot table column. If an array is passed, it is being used as the same manner as column values.

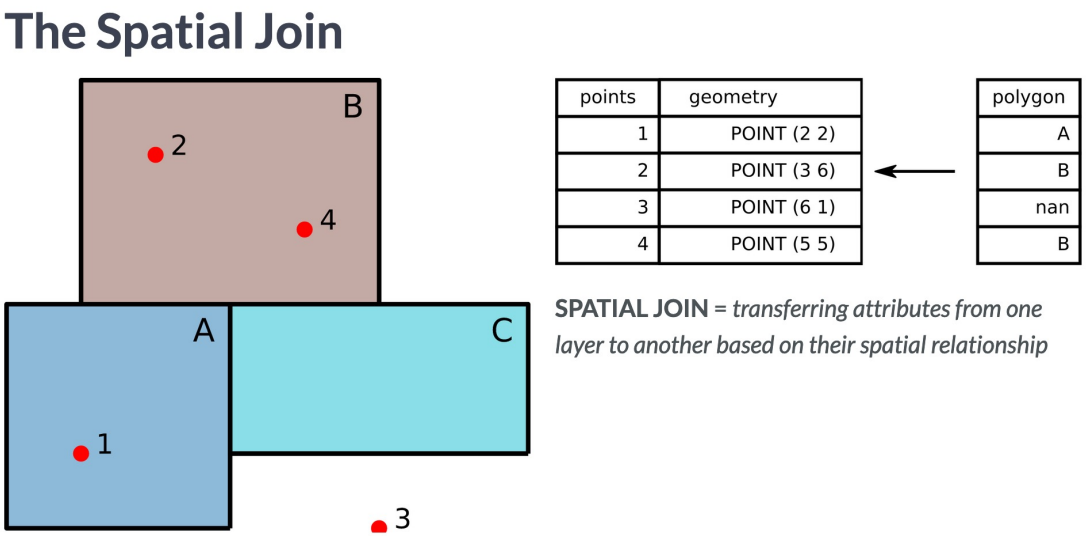
aggfunc : *function, list of functions, dict, default numpy.mean*

If list of functions passed, the resulting pivot table will have hierarchical columns whose top level are the function names (inferred from the function objects themselves) If dict is passed, the key is column to aggregate and value is function or list of functions.

COMBINACIÓN DE DATASETS: UNIONES ESPACIALES (`spatial join`)

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

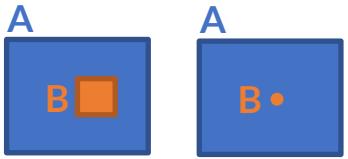
- Ej: left join (mantengo orden y filas del dataframe de de la izquierda)



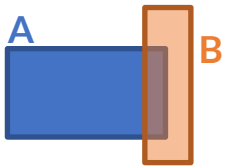
points	geometry
1	POINT (2 2)
2	POINT (3 6)
3	POINT (6 1)
4	POINT (5 5)

polygon
A
B
nan
B

SPATIAL JOIN = *transferring attributes from one layer to another based on their spatial relationship*



A `contains` B
B `within` A
A `intersects` B



A `intersects` B

09:55

COMBINACIÓN DE DATASETS: UNIONES ESPACIALES (`sjoin`)

geopandas.sjoin

**`geopandas.sjoin(`*`left_df, right_df, how='inner', predicate='intersects',`*
`lsuffix='left', rsuffix='right', **kwargs``)`**

Spatial join of two GeoDataFrames.

See the User Guide page Merging Data for details.

Parameters: **left_df, right_df** : *GeoDataFrames*

how : *string, default 'inner'*

The type of join:

- 'left': use keys from left_df; retain only left_df geometry column
- 'right': use keys from right_df; retain only right_df geometry column
- 'inner': use intersection of keys from both dfs; retain only left_df geometry column

predicate : *string, default 'intersects'*

Binary predicate. Valid values are determined by the spatial index used. You can check the valid values in left_df or right_df as `left_df.sindex.valid_query_predicates` or `right_df.sindex.valid_query_predicates` Replaces deprecated `op` parameter.

lsuffix : *string, default 'left'*

Suffix to apply to overlapping column names (left GeoDataFrame).

rsuffix : *string, default 'right'*

Suffix to apply to overlapping column names (right GeoDataFrame).

COMBINACIÓN DE DATASETS: UNIONES ESPACIALES (`spatial join`)

- 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)

cities (points)
countries (polygon)

```
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

09:55

COMBINACIÓN DE DATASETS: UNIONES ESPACIALES (`sjoin`)

- Ej: `left join` (mantengo orden y filas del dataframde 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

09:55