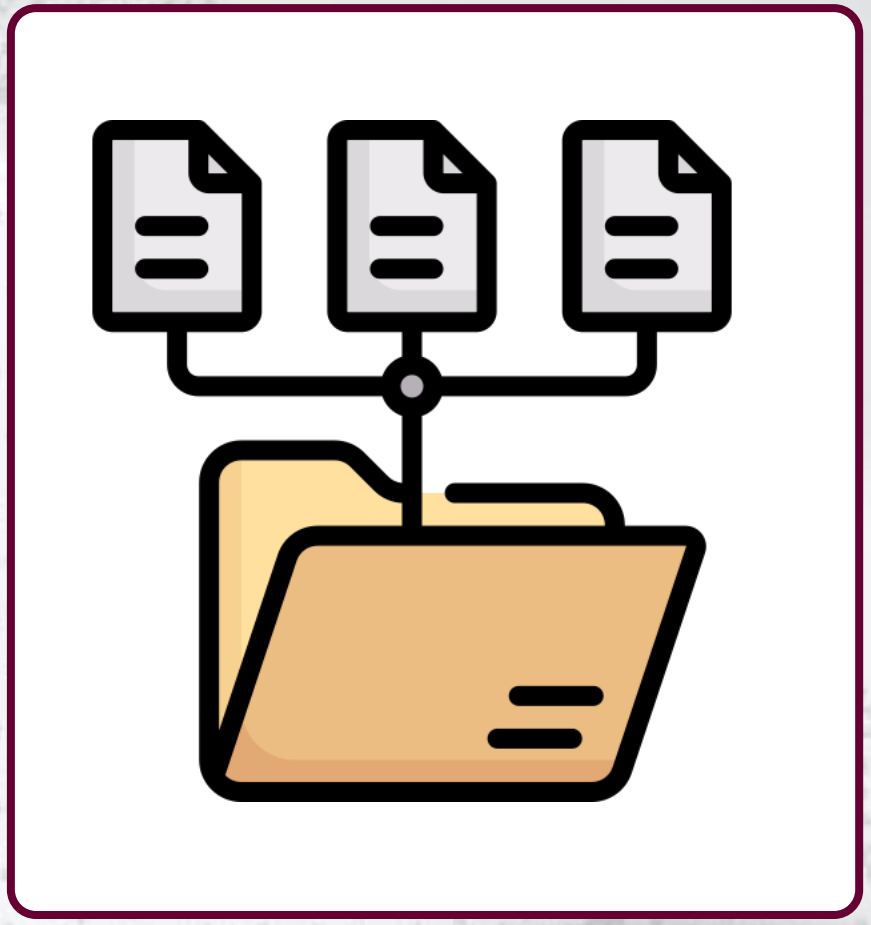
KIBERNUM
IT Academy

Módulo 2 – Obtención y Preparación de Datos

Agrupamiento de Datos

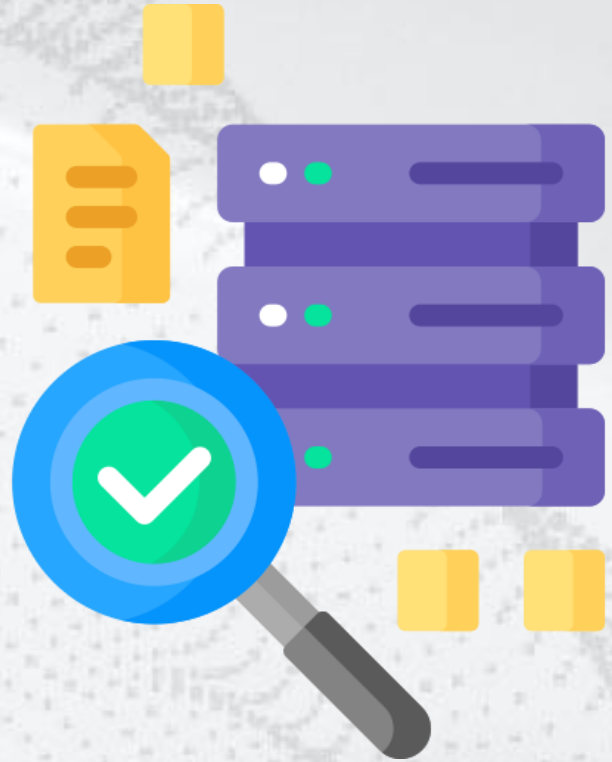
Ciencia de Datos

Agrupamiento de Datos

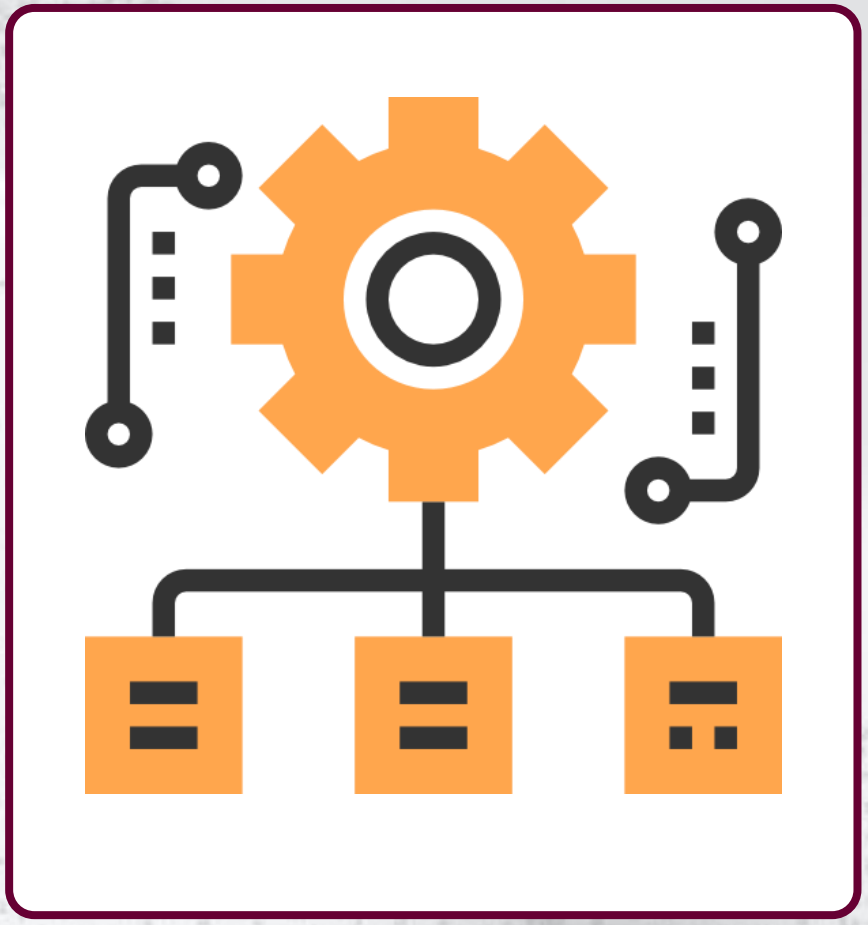


¿Qué exploraremos?

- Índices y Multi-índices.
- Agrupamiento.
- Pivoteo de tablas.
- Despivoteo de tablas.



Índices y Multi-índices



KIBERNUM

¿Qué es un índice?

En un DataFrame de Pandas, el índice es una estructura que permite **etiquetar y acceder a las filas de manera eficiente**. Es similar a la columna de etiquetas de una tabla en una base de datos relacional. Cada fila del DataFrame tiene una etiqueta única asociada en el índice, lo que facilita la identificación y recuperación de datos.

DataFrame Index

```
In [40]: data.head()
```

Out[40]:

	region	cases	cases_per_million	cases_7_days	cases_24_hours	deaths	deaths_per_million	deaths_7_days	deaths_24_hours	transmission_type
0	NaN	81947503	10497.596952	3717947	470046	1808041	231.61274	70443	9921	NaN
1	Americas	19346790	58449.050000	1035385	0	335789	1014.46000	12262	0	Community transmission
2	South-East Asia	10286709	7454.110000	139864	20035	148994	107.97000	1902	256	Clusters of cases
3	Americas	7619200	35845.040000	253683	55649	193875	912.10000	4655	1194	Community transmission
4	Europe	3186336	21834.020000	193630	27039	57555	394.39000	3896	536	Clusters of cases

¿Qué es un índice?

El índice puede ser una secuencia de números enteros (por defecto), etiquetas personalizadas, fechas u otros tipos de datos que proporcionen una identificación única para cada fila del DataFrame. Además, el índice también puede ser jerárquico, lo que significa que puede tener múltiples niveles de etiquetas.

El uso del índice en un DataFrame permite realizar operaciones de selección, indexación y alineación de datos de manera eficiente, lo que hace que trabajar con conjuntos de datos sea más intuitivo y poderoso en Pandas.

		Science			Math		
		mean	min	max	mean	min	max
Zone	School						
East	Shermer	55.5	45	66	78.0	67	89
North	Rushmore	70.0	70	70	78.0	78	78
South	Bayside	68.0	68	68	76.0	76	76
	Rydell	90.0	90	90	56.0	56	56
West	Hogwarts	65.0	32	98	65.5	55	76
	North Shore	70.0	70	70	79.0	79	79
	Ridgemont	89.0	89	89	100.0	100	100

Índices jerárquicos

Un índice jerárquico permite tener múltiples niveles de indexación en un mismo eje.

State	Year	Expenses
California	2010	37253956
New York	2010	19378102
New York	2000	18976457
Texas	2000	20851820
California	2000	29483772
Chicago	2010	34888922
Los Angeles	2010	24877673
Texas	2010	23098724
California	2005	30477622

```
import pandas as pd
import io
```

```
datos = '''State,Year,Expenses
California,2010,37253956
New York,2010,19378102
New York,2000,18976457
Texas,2000,20851820
California,2000,29483772
Chicago,2010,34888922
Los Angeles,2010,24877673
Texas,2010,23098724
California,2005,30477622
'''
```

```
data_io = io.StringIO(datos)
df = pd.read_csv(data_io)
df
```

	State	Year	Expenses
0	California	2010	37253956
1	New York	2010	19378102
2	New York	2000	18976457
3	Texas	2000	20851820
4	California	2000	29483772
5	Chicago	2010	34888922
6	Los Angeles	2010	24877673
7	Texas	2010	23098724
8	California	2005	30477622

Índices jerárquicos

```
# setear indices en mas de un nivel en las filas
# ojo con inplace
df.set_index(['State', 'Year'], inplace=True)
```

```
# Ahora el dataset tiene los dos incides definidos
df
```

		Expenses
State	Year	
California	2010	37253956
New York	2010	19378102
	2000	18976457
Texas	2000	20851820
California	2000	29483772
Chicago	2010	34888922
Los Angeles	2010	24877673
Texas	2010	23098724
California	2005	30477622


```
# setear indices en mas de un nivel en las filas
# ojo con inplace
df.set_index(['State', 'Year'], inplace=True)
```

```
# Ahora el dataset tiene los dos incides definidos
df
```


		Expenses
State	Year	
California	2010	37253956
New York	2010	19378102
	2000	18976457
Texas	2000	20851820
California	2000	29483772
Chicago	2010	34888922
Los Angeles	2010	24877673
Texas	2010	23098724
California	2005	30477622

Índices jerárquicos

```
# ahora verifiquemos los indices que tiene el dataframe
df.index
```

```
MultiIndex(levels=[['California', 'Chicago', 'Los Angeles', 'New York', 'Texas'], [2000, 2005, 2010]],
           labels=[[0, 3, 3, 4, 0, 1, 2, 4, 0], [2, 2, 0, 0, 0, 2, 2, 2, 1]],
           names=['State', 'Year'])
```

```
# Cambiemosle nombre a los indices
df.index.names = ['Estado','Año']
df
```

		Expenses
Estado	Año	
California	2010	37253956
New York	2010	19378102
	2000	18976457
Texas	2000	20851820
California	2000	29483772
Chicago	2010	34888922
Los Angeles	2010	24877673
Texas	2010	23098724
California	2005	30477622

Índices jerárquicos

```
# Sumario de estadisticas por nivel
df.sum(level=1)
```

Año	Expenses
2000	69312049
2005	30477622
2010	139497377

```
df.sum(level='Año')
```

Año	Expenses
2000	69312049
2005	30477622
2010	139497377

```
df.sum(level='Estado')
```

Estado	Expenses
California	97215350
Chicago	34888922
Los Angeles	24877673
New York	38354559
Texas	43950544

Índices jerárquicos

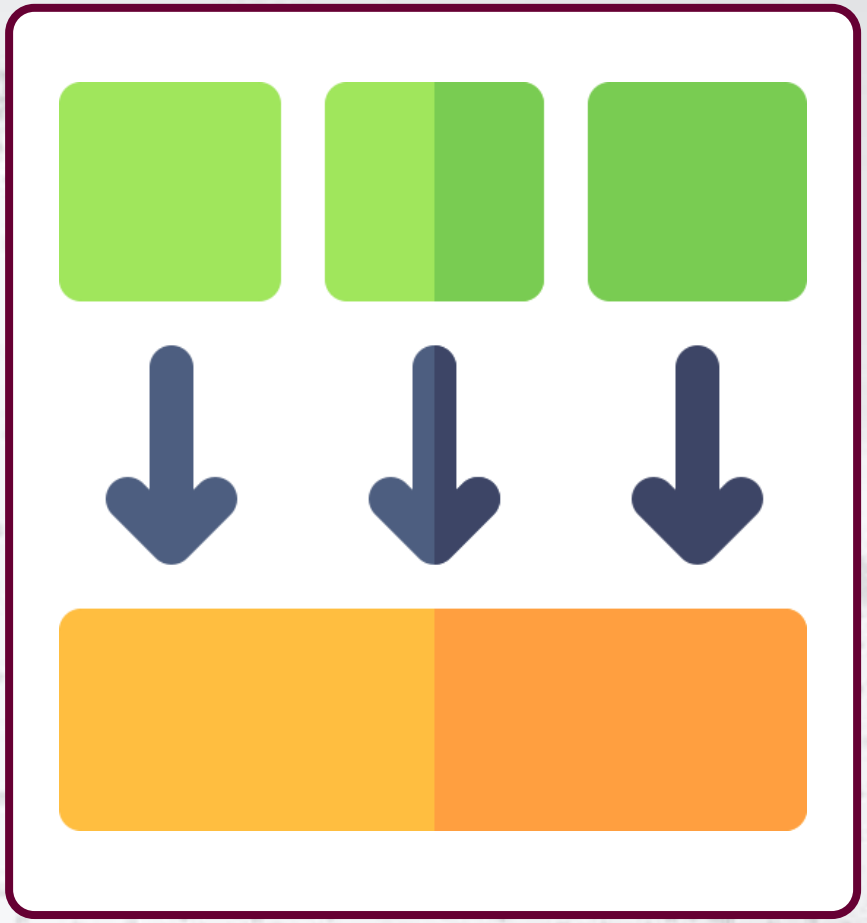
```
# Reordenando los niveles de indices
df.swaplevel('Año','Estado')
```

		Expenses
Año	Estado	
2010	California	37253956
	New York	19378102
2000	New York	18976457
	Texas	20851820
	California	29483772
2010	Chicago	34888922
	Los Angeles	24877673
	Texas	23098724
2005	California	30477622

```
# Ordenando los niveles de indices
df.sort_index(level=1, ascending=True)
```

		Expenses
Estado	Año	
California	2000	29483772
New York	2000	18976457
Texas	2000	20851820
California	2005	30477622
	2010	37253956
Chicago	2010	34888922
Los Angeles	2010	24877673
New York	2010	19378102
Texas	2010	23098724

Agrupamiento



¿Qué es Agrupamiento?

El agrupamiento es una operación en la que los datos se dividen en grupos basados en algún criterio común y luego se aplican operaciones o transformaciones a cada grupo por separado. Es una técnica fundamental en el análisis de datos y se utiliza ampliamente en diferentes áreas, como la estadística, la ciencia de datos y la ingeniería.

	Name	Team	Position	Age	Weight
0	Avery Bradly	Boston Celtics	PG	25.0	180.0
1	Jae Crowder	Boston Celtics	SF	25.0	235.0
2	John Holland	Boston Celtics	SG	27.0	205.0
3	R.j. Hunter	Boston Celtics	SG	22.0	185.0
4	Sergey Karasev	Brooklyn Nets	SG	22.0	208.0
5	sean Kilpatrick	Brooklyn Nets	SG	26.0	219.0
6	Shane Larkin	Brooklyn Nets	PG	23.0	175.0
7	Brook Lopez	Brooklyn Nets	C	28.0	275.0
8	Chris Johnson	Utah Jazz	SF	26.0	206.0
9	Trey Lyles	Utah Jazz	PF	20.0	234.0
10	Shelvin Mack	Utah Jazz	PG	26.0	203.0
11	Raul Pleiss	Utah Jazz	PG	24.0	179.0

Boston Celtics
Boston Celtics
Boston Celtics
Boston Celtics

Brooklyn Nets
Brooklyn Nets
Brooklyn Nets
Brooklyn Nets

Utah Jazz
Utah Jazz
Utah Jazz
Utah Jazz

¿Cuál es el promedio de edad de los jugadores de Boston Celtics?

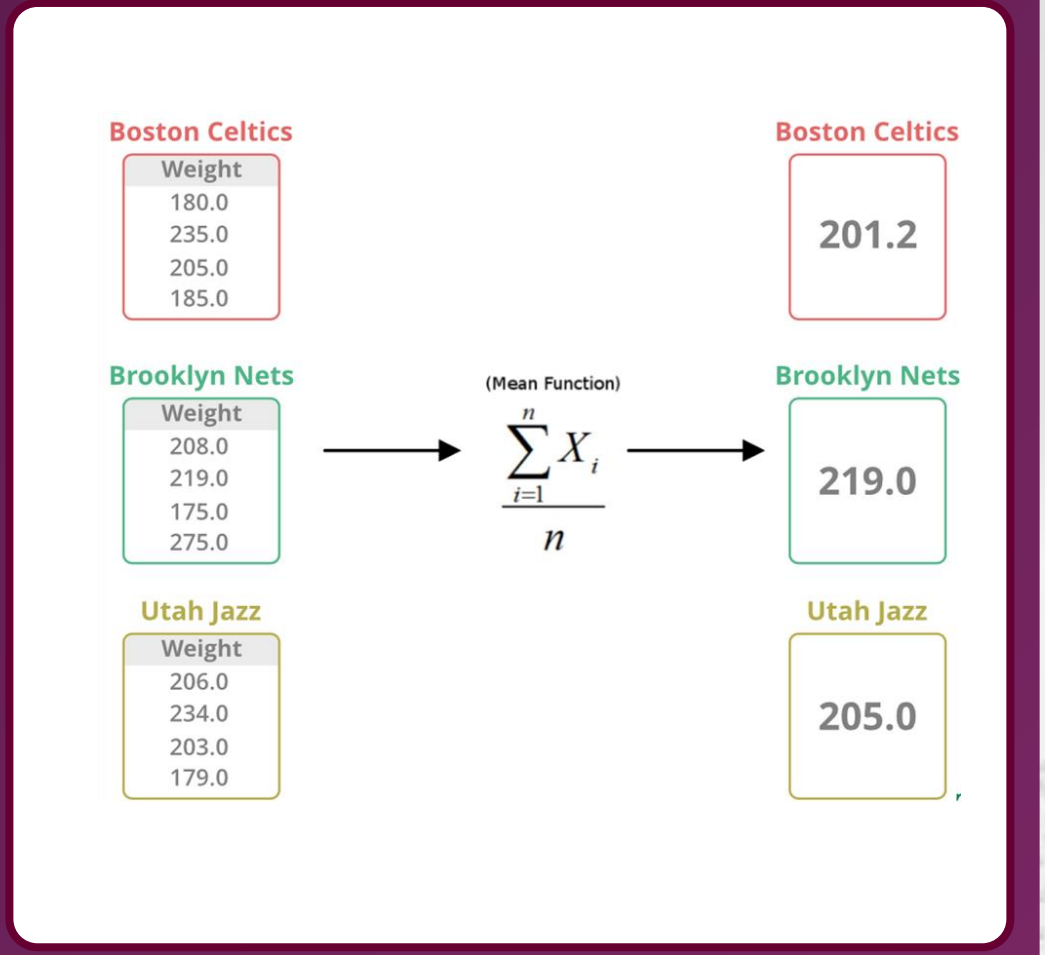
¿Cuál es el peso máximo de los jugadores de Utah Jazz?

¿Cuál la edad mínima de los jugadores de Brooklyn Nets?

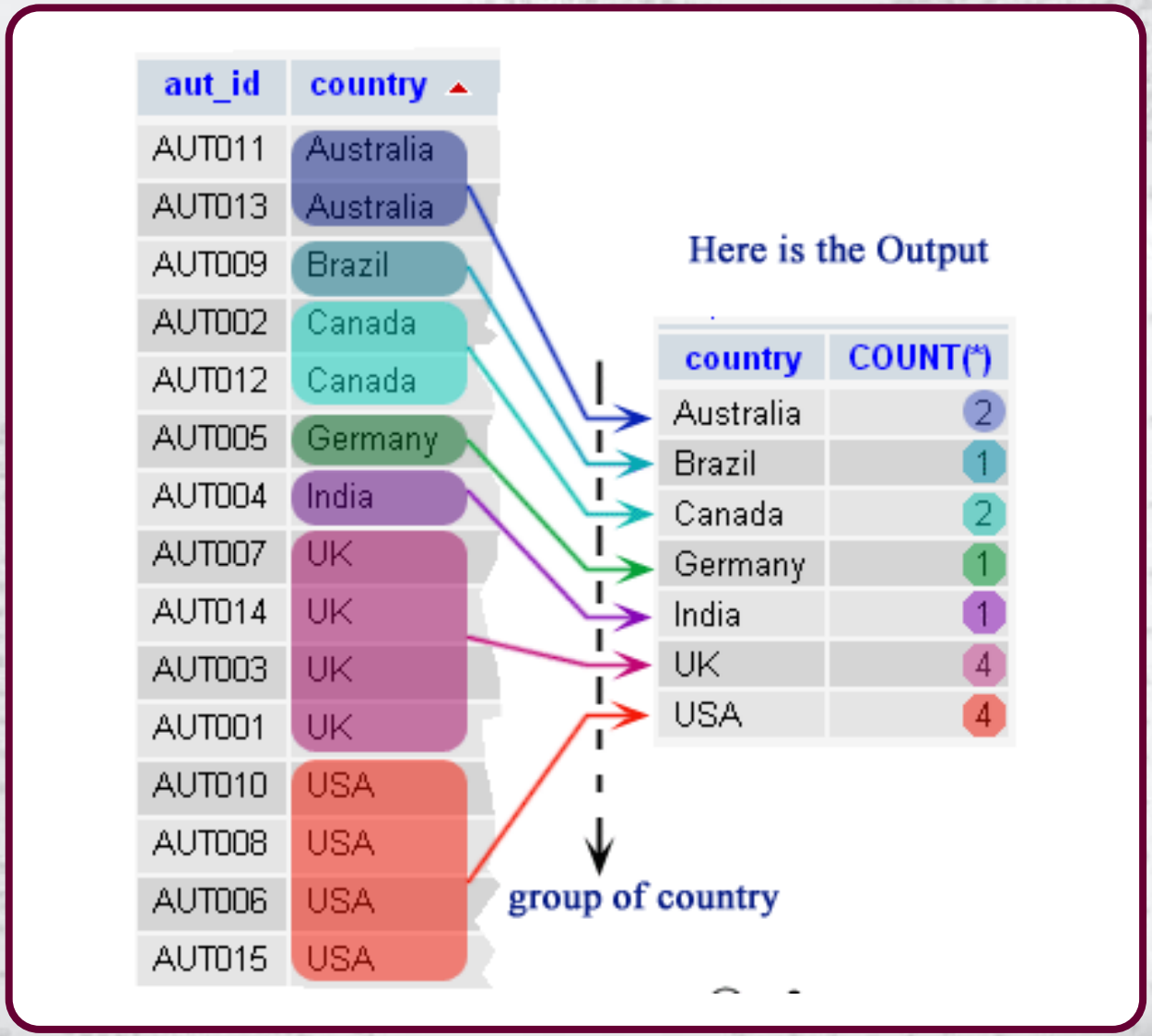
¿Qué es Agrupamiento?

En el contexto de Pandas, el agrupamiento se realiza utilizando el método groupby(), que divide un DataFrame en grupos basados en los valores de una o más columnas. Luego, se pueden aplicar funciones de agregación, como sumas, medias, conteos, etc., a cada grupo por separado utilizando métodos como sum(), mean(), count(), etc.

El agrupamiento es útil para resumir y analizar datos de manera más eficiente, ya que permite examinar las características de subconjuntos específicos de datos de forma separada. Por ejemplo, se puede agrupar un conjunto de datos de ventas por categoría de producto y luego calcular la suma de ventas para cada categoría por separado. Esto proporciona información útil sobre el rendimiento de cada categoría de producto individualmente.



Agrupamiento



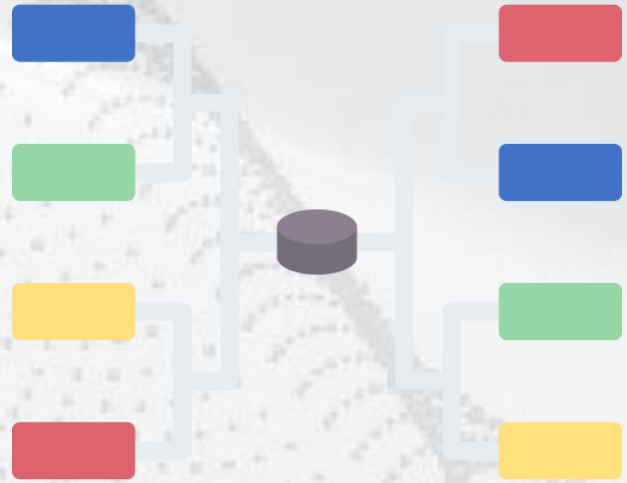
Agrupamiento

En primer lugar, cargaremos el set de datos.

```
import pandas as pd
import numpy as np
```

```
df = pd.read_csv('state-expenses.csv')
df
```

	State	Year	Expenses
0	California	2010	37253956
1	New York	2010	19378102
2	New York	2000	18976457
3	Texas	2000	20851820
4	California	2000	29483772
5	Chicago	2010	34888922
6	Los Angeles	2010	24877673
7	Texas	2010	23098724
8	California	2005	30477622



Agrupando datos

Definiremos una agrupación de acuerdo con un criterio y se lo asignaremos a una variable:

```
byYear = df.groupby('Year')
```

```
type(byYear)
```

```
pandas.core.groupby.generic.DataFrameGroupBy
```

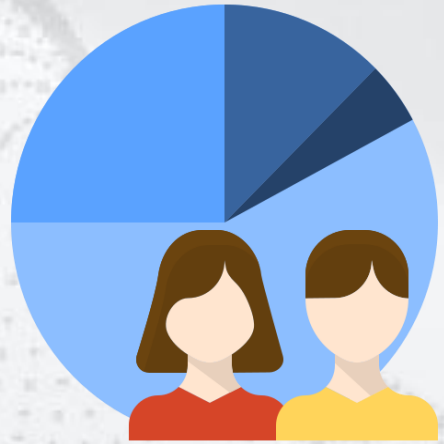
Ahora podemos realizar operaciones de agrupación:

```
byYear.count()
```

Year	State	Expenses
2000	3	3
2005	1	1
2010	5	5

```
byYear.sum()
```

Year	Expenses
2000	69312049
2005	30477622
2010	139497377



KIBERNUM

Forma abreviada

Forma abreviada de consultar una operación de agrupación en un dataframe:

```
df.groupby('State').sum()
```

State	Year	Expenses
California	6015	97215350
Chicago	2010	34888922
Los Angeles	2010	24877673
New York	4010	38354559
Texas	4010	43950544

El resultado es un dataframe con los valores agrupados

Forma abreviada

El resultado de una operación de agrupamiento es un dataframe, por lo tanto, lo podemos operar como tal:

```
res = df.groupby('State').sum()
type(res)
```

```
pandas.core.frame.DataFrame
```

```
res['Year']
```

```
State
California      6015
Chicago         2010
Los Angeles     2010
New York        4010
Texas           4010
Name: Year, dtype: int64
```

```
res.loc['Chicago']
```

```
Year            2010
Expenses    34888922
Name: Chicago, dtype: int64
```

Métodos de agregación

Agrupar y buscar el valor mínimo:

```
df.groupby('State').min()
```

State	Year	Expenses
California	2000	29483772
Chicago	2010	34888922
Los Angeles	2010	24877673
New York	2000	18976457
Texas	2000	20851820

Agrupar y buscar el valor máximo:

```
df.groupby('State').max()
```

State	Year	Expenses
California	2010	37253956
Chicago	2010	34888922
Los Angeles	2010	24877673
New York	2010	19378102
Texas	2010	23098724

Métodos de agregación

Media de un grupo:

```
df.groupby('State').mean()
```

State	Year	Expenses
California	2005.0	3.240512e+07
Chicago	2010.0	3.488892e+07
Los Angeles	2010.0	2.487767e+07
New York	2005.0	1.917728e+07
Texas	2005.0	2.197527e+07

Desviación estándar de un grupo:

```
df.groupby('State').std()
```

State	Year	Expenses
California	5.000000	4.228518e+06
Chicago	NaN	NaN
Los Angeles	NaN	NaN
New York	7.071068	2.840059e+05
Texas	7.071068	1.588801e+06

Métodos de agregación

Media de un grupo:

```
df.groupby('State').median()
```

State	Year	Expenses
California	2005.0	30477622.0
Chicago	2010.0	34888922.0
Los Angeles	2010.0	24877673.0
New York	2005.0	19177279.5
Texas	2005.0	21975272.0

Quantil de un grupo:

```
df.groupby('State').quantile(q=0.5)
```

State	Year	Expenses
California	2005.0	30477622.0
Chicago	2010.0	34888922.0
Los Angeles	2010.0	24877673.0
New York	2005.0	19177279.5
Texas	2005.0	21975272.0

Describiendo un grupo

Se puede obtener de forma rápida la información de sumarización de un grupo:

```
df.groupby('State').describe()
```

	Year								Expenses							
	count	mean	std	min	25%	50%	75%	max	count	mean	std	min	25%	50%	75%	
State																
California	3.0	2005.0	5.000000	2000.0	2002.5	2005.0	2007.5	2010.0	3.0	3.240512e+07	4.228518e+06	29483772.0	29980697.00	30477622.0	33865789	
Chicago	1.0	2010.0	NaN	2010.0	2010.0	2010.0	2010.0	2010.0	1.0	3.488892e+07	NaN	34888922.0	34888922.00	34888922.0	34888922	
Los Angeles	1.0	2010.0	NaN	2010.0	2010.0	2010.0	2010.0	2010.0	1.0	2.487767e+07	NaN	24877673.0	24877673.00	24877673.0	24877673	
New York	2.0	2005.0	7.071068	2000.0	2002.5	2005.0	2007.5	2010.0	2.0	1.917728e+07	2.840059e+05	18976457.0	19076868.25	19177279.5	19277690	
Texas	2.0	2005.0	7.071068	2000.0	2002.5	2005.0	2007.5	2010.0	2.0	2.197527e+07	1.588801e+06	20851820.0	21413546.00	21975272.0	22536998	

Agrupamiento por más de un nivel

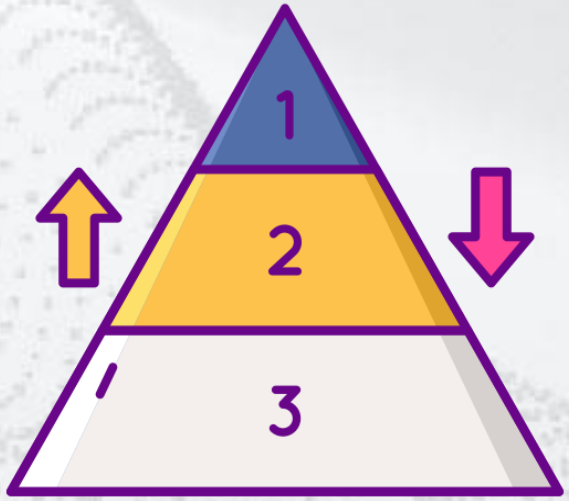
Podemos establecer más de una variable de agrupamiento (índice).

```
df.groupby(['State','Year']).sum()
```

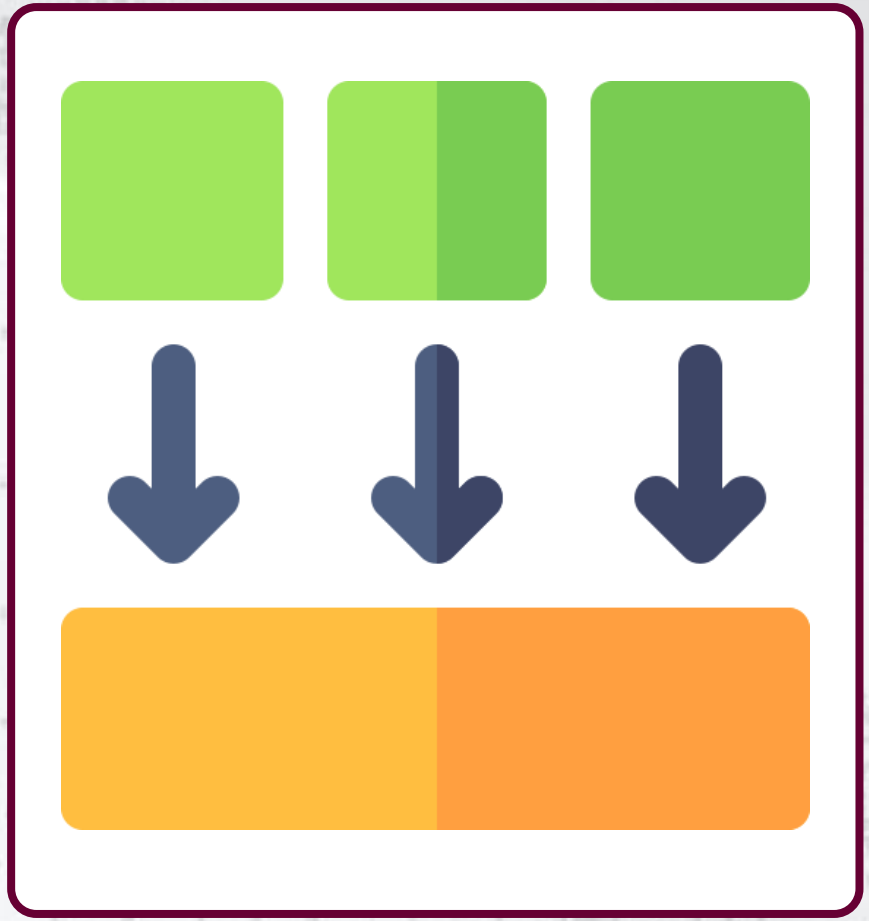
		Expenses
State	Year	
California	2000	29483772
	2005	30477622
	2010	37253956
Chicago	2010	34888922
Los Angeles	2010	24877673
New York	2000	18976457
	2010	19378102
Texas	2000	20851820
	2010	23098724

```
df.groupby(['Year','State']).sum()
```

		Expenses
Year	State	
2000	California	29483772
	New York	18976457
	Texas	20851820
2005	California	30477622
2010	California	37253956
	Chicago	34888922
	Los Angeles	24877673
	New York	19378102
	Texas	23098724

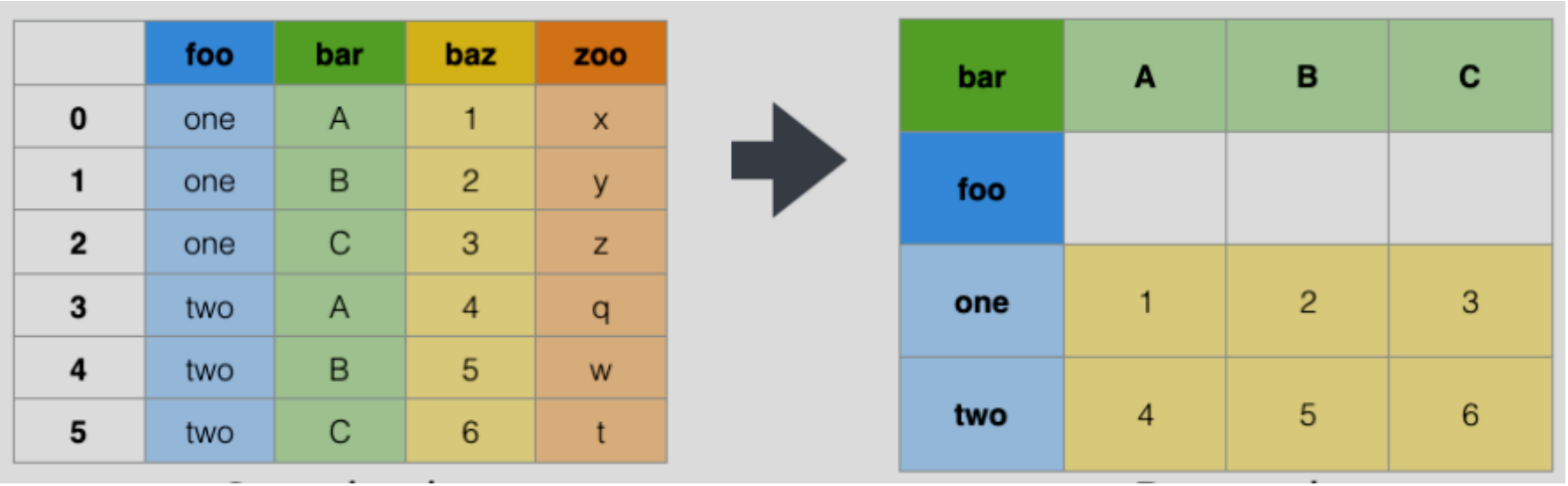


Tablas Pivoteadas



¿Qué es una tabla Pivoteada?

La característica principal de una tabla pivote es que permite reorganizar y resumir datos de una manera más estructurada y legible. Esto se logra al pivotar (rotar) los datos originales de modo que los valores de una columna se conviertan en las columnas de la tabla pivotada y los valores de otra columna se conviertan en las filas, con operaciones de agregación aplicadas a esos valores.



	foo	bar	baz	zoo
0	one	A	1	x
1	one	B	2	y
2	one	C	3	z
3	two	A	4	q
4	two	B	5	w
5	two	C	6	t

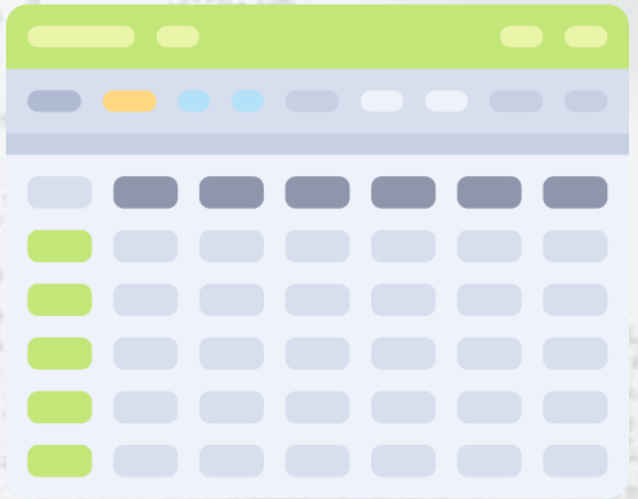
bar	A	B	C
foo			
one	1	2	3
two	4	5	6

Pivotes

Permite hacer tablas definiendo pivotes, similares a las tablas dinámicas de Excel. Para esto, tomemos el siguiente dataframe:

```
df = pd.read_csv('state-expenses.csv')
# incorporamos una variable adicional al dataset
df['Investment'] = np.random.randint(1e7,9e7,9)
df
```

	State	Year	Expenses	Investment
0	California	2010	37253956	10778591
1	New York	2010	19378102	60824702
2	New York	2000	18976457	49477721
3	Texas	2000	20851820	36604653
4	California	2000	29483772	61432718
5	Chicago	2010	34888922	70699986
6	Los Angeles	2010	24877673	55260190
7	Texas	2010	23098724	63629831
8	California	2005	30477622	34859841



Pivotes

Ahora apliquemos la función **pivot_table()**:

Columna con el valor a operar

Columnas que compondrán el índice

Columnas que quedarán como tales

Función de agregación

```
# Puede aplicarse funciones de agregación tales como sum, count, mean, min, max
df.pivot_table(values='Expenses', index='State', columns='Year', aggfunc='sum', fill_value=0)
```

Rellena valores

Year	2000	2005	2010
State			
California	29483772	30477622	37253956
Chicago	0	0	34888922
Los Angeles	0	0	24877673
New York	18976457	0	19378102
Texas	20851820	0	23098724

Puede aplicarse funciones de agregación tales como sum, count, mean, min, max

Pivotes

En este caso, se especifica un listado de columnas en el campo values:

```
df.pivot_table(values=['Expenses','Investment'], index='State', columns='Year', aggfunc='sum', fill_value=0)
```

	Expenses			Investment		
Year	2000	2005	2010	2000	2005	2010
State						
California	29483772	30477622	37253956	61432718	34859841	10778591
Chicago	0	0	34888922	0	0	70699986
Los Angeles	0	0	24877673	0	0	55260190
New York	18976457	0	19378102	49477721	0	60824702
Texas	20851820	0	23098724	36604653	0	63629831

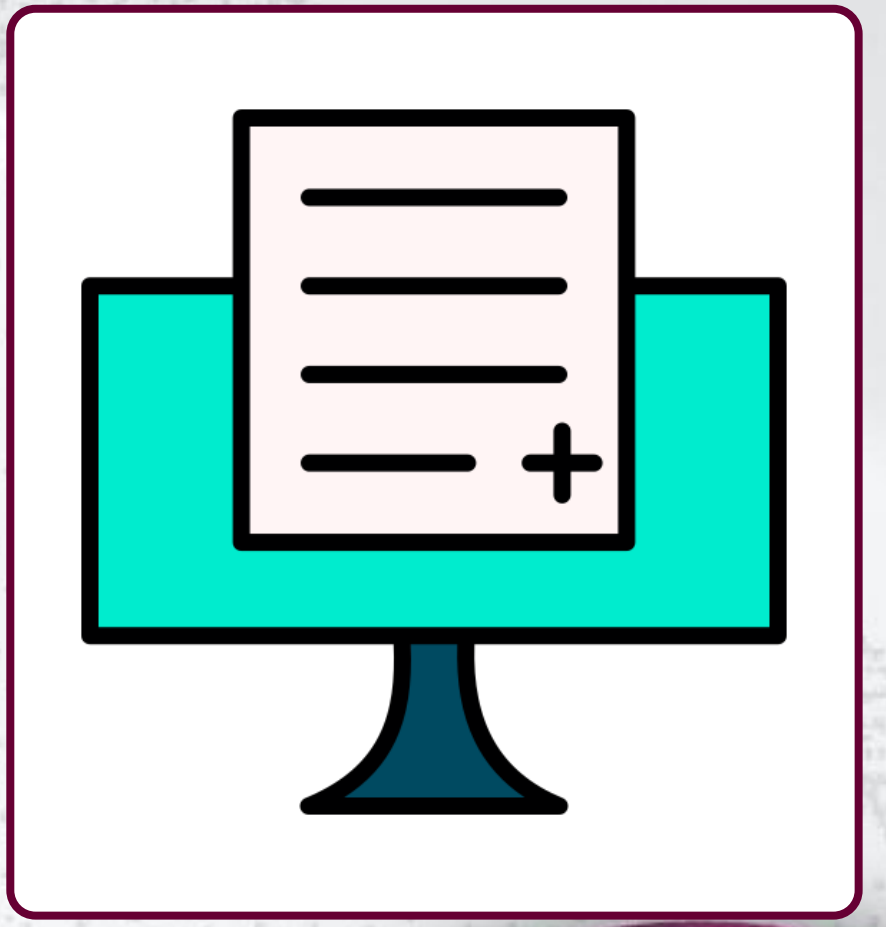
Pivotes

Si no se especifican columnas, entonces el reporte es similar a uno de agrupación (groupby)

```
df.pivot_table(values=['Expenses','Investment'], index=['State','Year'], aggfunc='sum', fill_value=0)
```

		Expenses	Investment
State	Year		
California	2000	29483772	61432718
	2005	30477622	34859841
	2010	37253956	10778591
Chicago	2010	34888922	70699986
Los Angeles	2010	24877673	55260190
New York	2000	18976457	49477721
	2010	19378102	60824702
Texas	2000	20851820	36604653
	2010	23098724	63629831

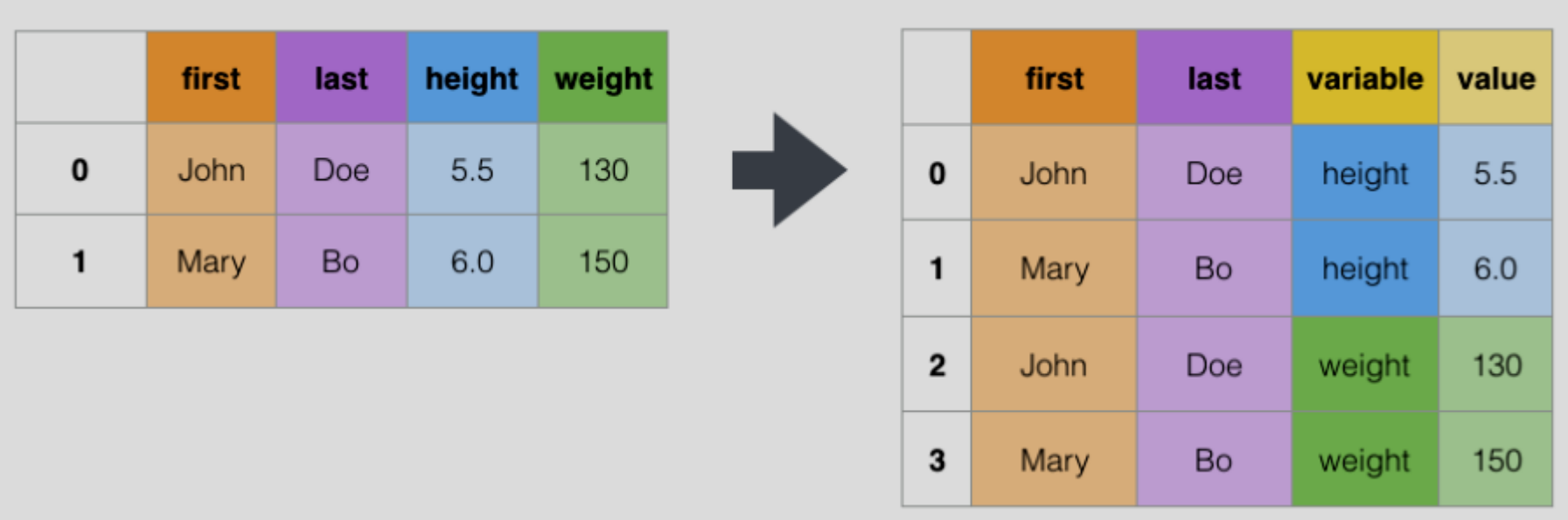
Despivoteo de Tablas



KIBERNUM

¿Qué es Despivoteo?

El "despivoteo", es el proceso inverso a la creación de una tabla pivotada. Mientras que una tabla pivotada organiza los datos de manera estructurada para facilitar el análisis y la visualización, el despivoteo se refiere a la acción de revertir esta transformación y volver a los datos en su formato original o en un formato diferente.



	first	last	height	weight
0	John	Doe	5.5	130
1	Mary	Bo	6.0	150

	first	last	variable	value
0	John	Doe	height	5.5
1	Mary	Bo	height	6.0
2	John	Doe	weight	130
3	Mary	Bo	weight	150

Despivoteo

A veces, se requiere despivotear una tabla que ya viene con una forma de pivote para dejarla al estilo “planilla”.

```
url = 'https://www.eia.gov/dnav/ng/hist/rngwhhdM.htm'
tables = pd.read_html(url)
```

```
# Explorando, la tabla 4 es la que tiene la información requerida
df = tables[4].copy()
```

```
df.head()
```

	Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
0	1997.0	3.45	2.15	1.89	2.03	2.25	2.20	2.19	2.49	2.88	3.07	3.01	2.35
1	1998.0	2.09	2.23	2.24	2.43	2.14	2.17	2.17	1.85	2.02	1.91	2.12	1.72
2	1999.0	1.85	1.77	1.79	2.15	2.26	2.30	2.31	2.80	2.55	2.73	2.37	2.36
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	2000.0	2.42	2.66	2.79	3.04	3.59	4.29	3.99	4.43	5.06	5.02	5.52	8.90

Despivoteo

Aplicamos previamente algunas técnicas de limpieza de datos.

Al parecer, hay lineas en blanco (espaciadoras), entonces removemos las filas con más de 10 espacios en blanco

```
df.dropna(thresh=10,inplace=True)
```

Cambiamos el tipo de dato del año a *int*, ya que fue reconocido como *float* al cargar la data

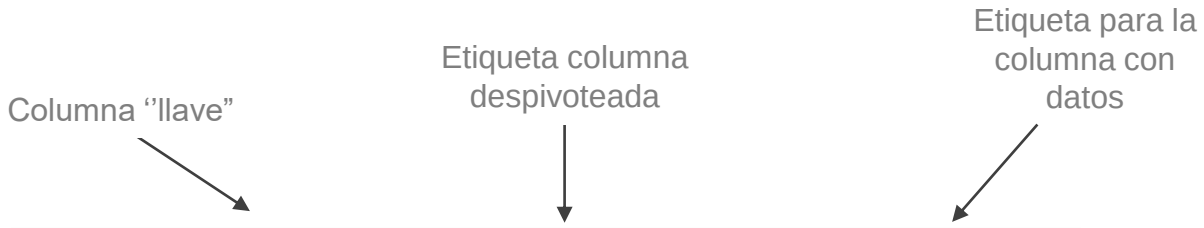
```
df['Year'] = df['Year'].astype(int)
```

```
df
```

	Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
0	1997	3.45	2.15	1.89	2.03	2.25	2.20	2.19	2.49	2.88	3.07	3.01	2.35
1	1998	2.09	2.23	2.24	2.43	2.14	2.17	2.17	1.85	2.02	1.91	2.12	1.72
2	1999	1.85	1.77	1.79	2.15	2.26	2.30	2.31	2.80	2.55	2.73	2.37	2.36
4	2000	2.42	2.66	2.79	3.04	3.59	4.29	3.99	4.43	5.06	5.02	5.52	8.90
5	2001	8.17	5.61	5.23	5.19	4.19	3.72	3.11	2.97	2.19	2.46	2.34	2.30
6	2002	2.32	2.32	3.03	3.43	3.50	3.26	2.99	3.09	3.55	4.13	4.04	4.74
7	2003	5.43	7.71	5.93	5.26	5.81	5.82	5.03	4.99	4.62	4.63	4.47	6.13
8	2004	6.14	5.37	5.39	5.71	6.33	6.27	5.93	5.41	5.15	6.35	6.17	6.58
10	2005	6.15	6.14	6.96	7.16	6.47	7.18	7.63	9.53	11.75	13.42	10.30	13.05

Despivoteo

La función **melt()** permite “despivotear” la tabla.




```
df.melt(id_vars='Year', var_name='Month', value_name='Precio')
```

	Year	Month	Precio
0	1997	Jan	3.45
1	1998	Jan	2.09
2	1999	Jan	1.85
3	2000	Jan	2.42
4	2001	Jan	8.17
...	...	...	...
271	2015	Dec	1.93
272	2016	Dec	3.59
273	2017	Dec	2.82
274	2018	Dec	4.04
275	2019	Dec	2.22

Dudas y consultas

KIBERNUM

Fin presentación

KIBERNUM