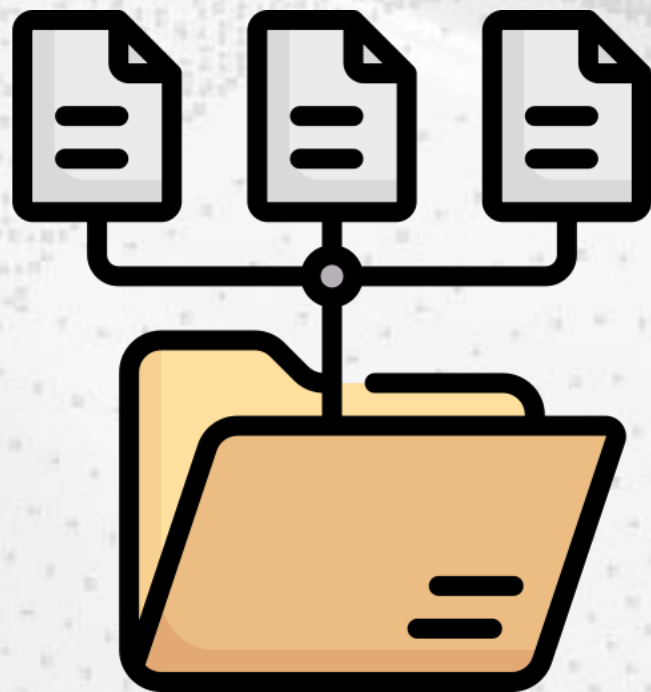


Módulo 8  
Clase 1

# Análisis Exploratorio de Datos



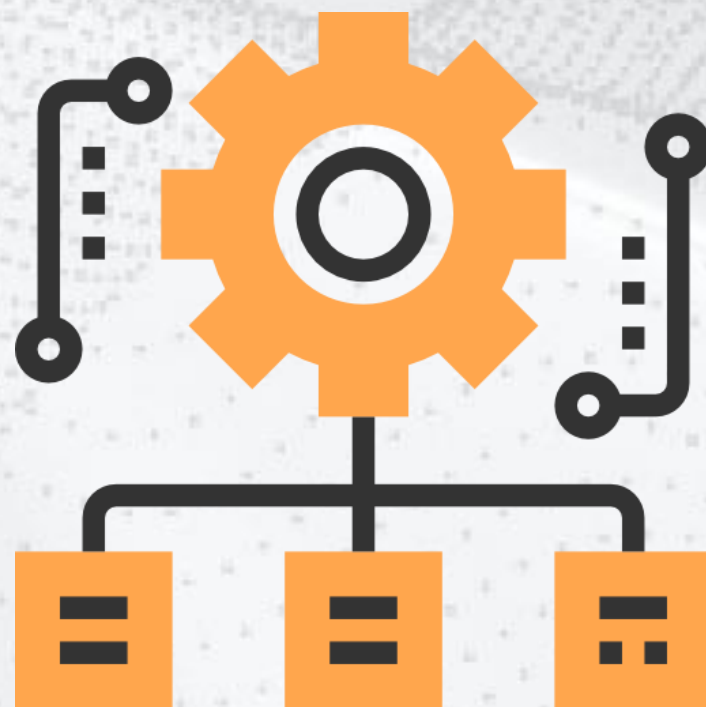
Agrupamiento de Datos

# ¿Qué cubriremos?

- **Multi-índices**
- **Agrupamiento**
- **Pivoteo de tablas**
- **Despivoteo de tablas**







Índices Jerárquicos

# 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

	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

```
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
```

# 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 indices 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 indices 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 índices 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 índices  
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 estadísticas por nivel  
df.sum(level=1)
```

Expenses	
Año	
2000	69312049
2005	30477622
2010	139497377

```
df.sum(level='Año')
```

Expenses	
Año	
2000	69312049
2005	30477622
2010	139497377

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

Expenses	
Estado	
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

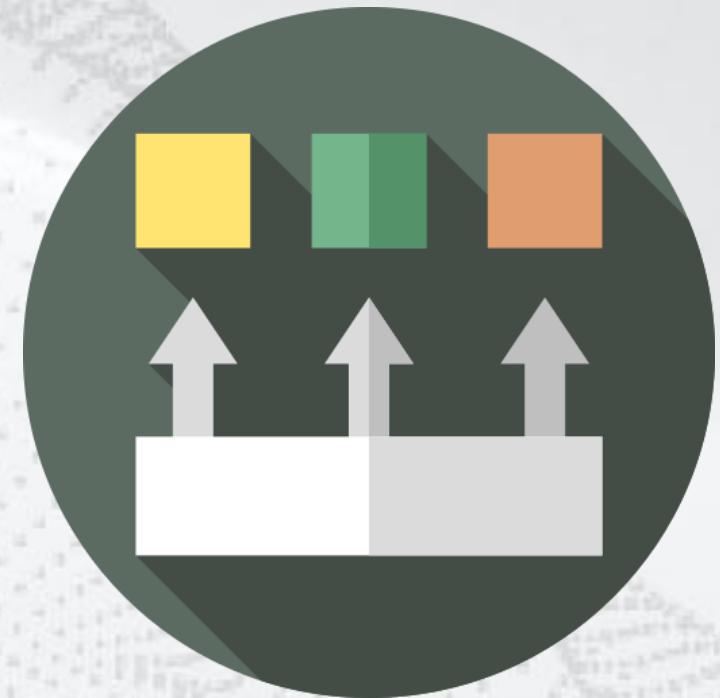
# Agrupamiento

aut_id	country ▲
AUT011	Australia
AUT013	Australia
AUT009	Brazil
AUT002	Canada
AUT012	Canada
AUT005	Germany
AUT004	India
AUT007	UK
AUT014	UK
AUT003	UK
AUT001	UK
AUT010	USA
AUT008	USA
AUT006	USA
AUT015	USA

Here is the Output

country	COUNT(*)
Australia	2
Brazil	1
Canada	2
Germany	1
India	1
UK	4
USA	4

group of country



# Agrupamiento

Primeramente, 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 a un criterio y se lo asignaremos a una variable:

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

```
type(byYear)
```

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

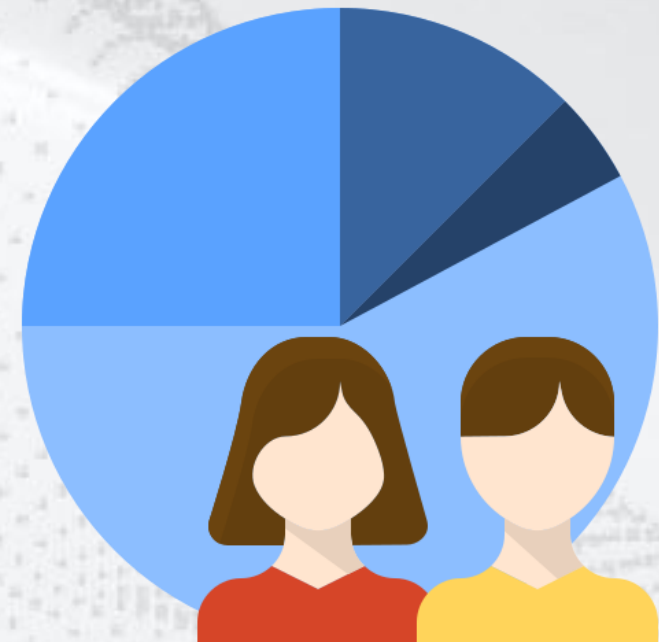
Ahora podemos realizar operaciones de agrupación:

```
byYear.count()
```

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

```
byYear.sum()
```

	Expenses
Year	
2000	69312049
2005	30477622
2010	139497377



# Forma abreviada

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

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

	Year	Expenses
State		
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 Agrupamiento

**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 Estadísticos

## Media de un grupo:

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

	Year	Expenses
State		
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()
```

	Year	Expenses
State		
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 Estadísticos

## 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 sumariaización de un grupo:

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

State	Year								Expenses						
	count	mean	std	min	25%	50%	75%	max	count	mean	std	min	25%	50%	75%
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

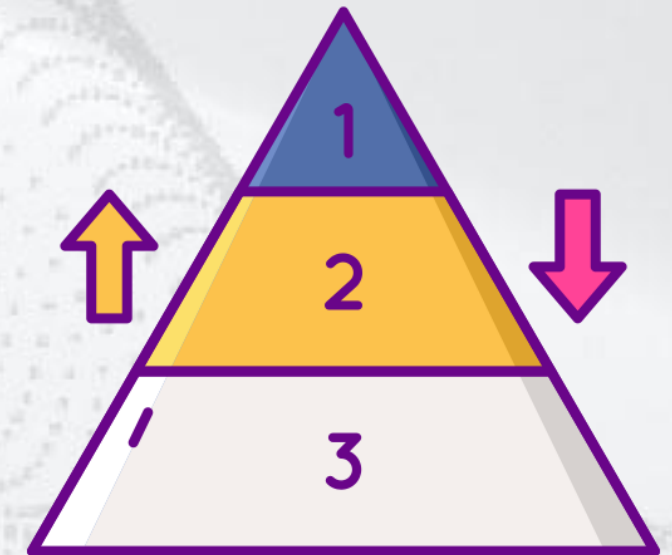
Podemos establecer más de una variable de agrupamiento (índice).

```
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







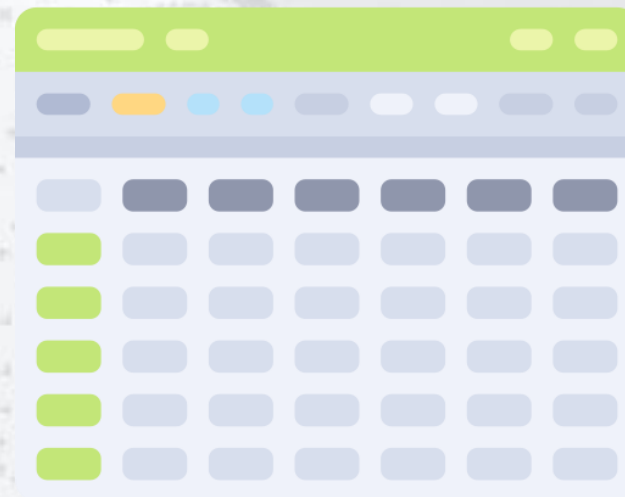
Tablas Pivoteadas

# 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)
```

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

Rellena valores



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)
```

Year	Expenses			Investment		
	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

# Despivotado

A veces se requiere despivotar 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 líneas 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

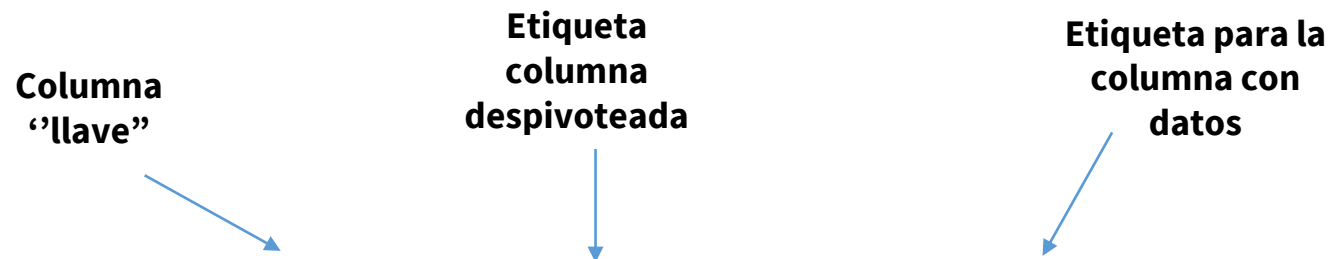
	Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
0	1997	3.45	2.15	1.89	2.03	2.25	2.20	2.19	2.49	2.88	3.07	3.01	2.35
1	1998	2.09	2.23	2.24	2.43	2.14	2.17	2.17	1.85	2.02	1.91	2.12	1.72
2	1999	1.85	1.77	1.79	2.15	2.26	2.30	2.31	2.80	2.55	2.73	2.37	2.36
4	2000	2.42	2.66	2.79	3.04	3.59	4.29	3.99	4.43	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.

**Columna  
“llave”**                      **Etiqueta  
columna  
despivoteada**                      **Etiqueta para la  
columna con  
datos**



```
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



Gracias