



Análisis de datos con Pandas

Ing. Agustín Ullón Ramírez





Datos

Datos

- Los datos son hechos, eventos, transacciones, etc, que han sido registrados
- Detrás de cada aplicación hay una base de datos



A hand in a blue suit sleeve holds a black funnel. Above the funnel, a large cloud of colorful business icons floats, including gears, a clock, a magnifying glass, a smartphone, a laptop, a target, a shopping cart, a padlock, a globe, a location pin, a calendar, a speech bubble, a document, a key, a floppy disk, and a small tablet. The funnel directs these icons into a computer monitor at the bottom, which displays a dashboard with a bar chart and a line graph.

Una nueva tendencia: Storytelling

① DATOS



② LIMPIOS EN UNA BASE DE DATOS



③ ANALIZADOS

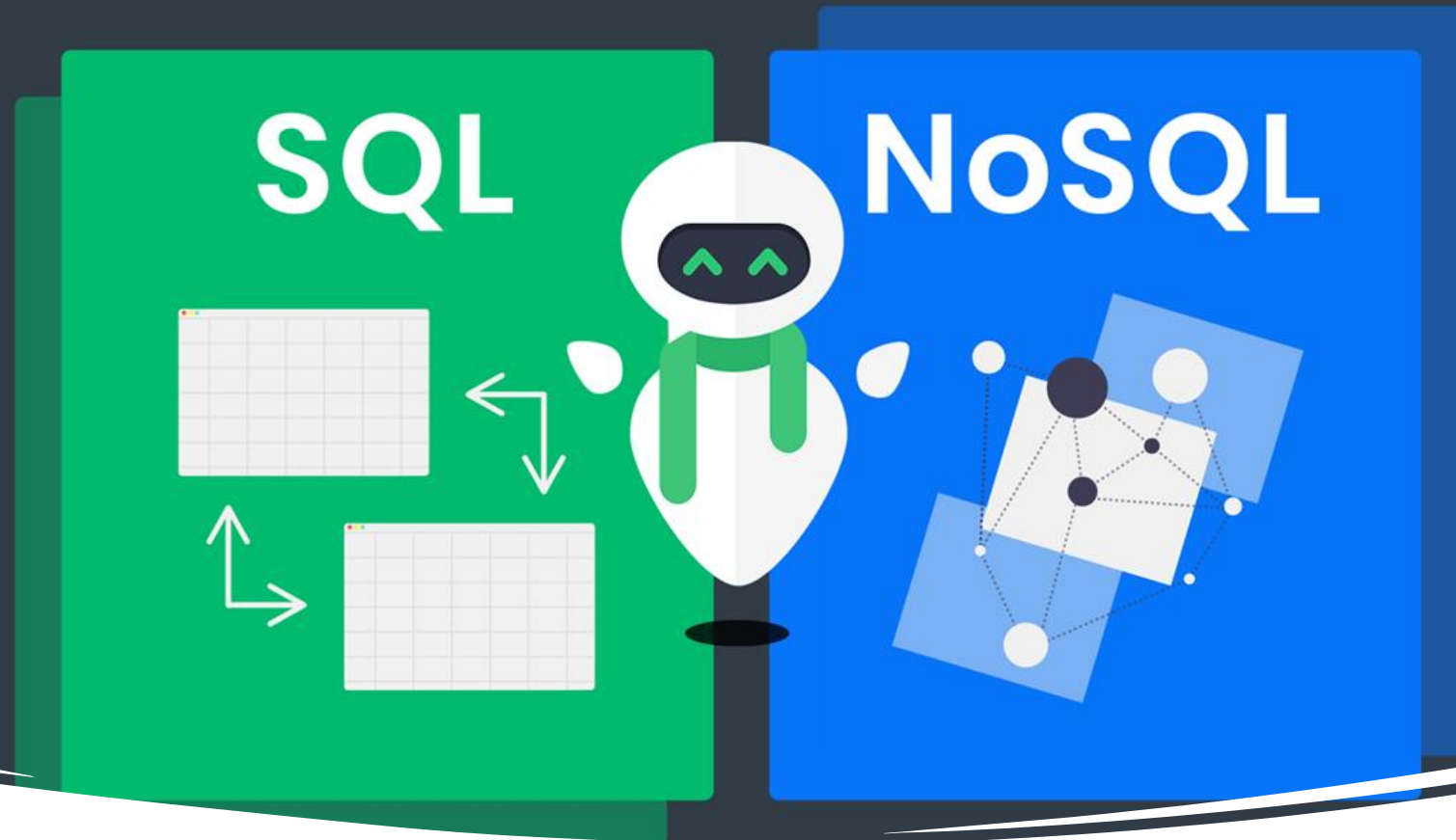


④ PRESENTADOS DE
FORMA VISUAL



⑤ EXPLICADOS CON
UNA HISTORIA

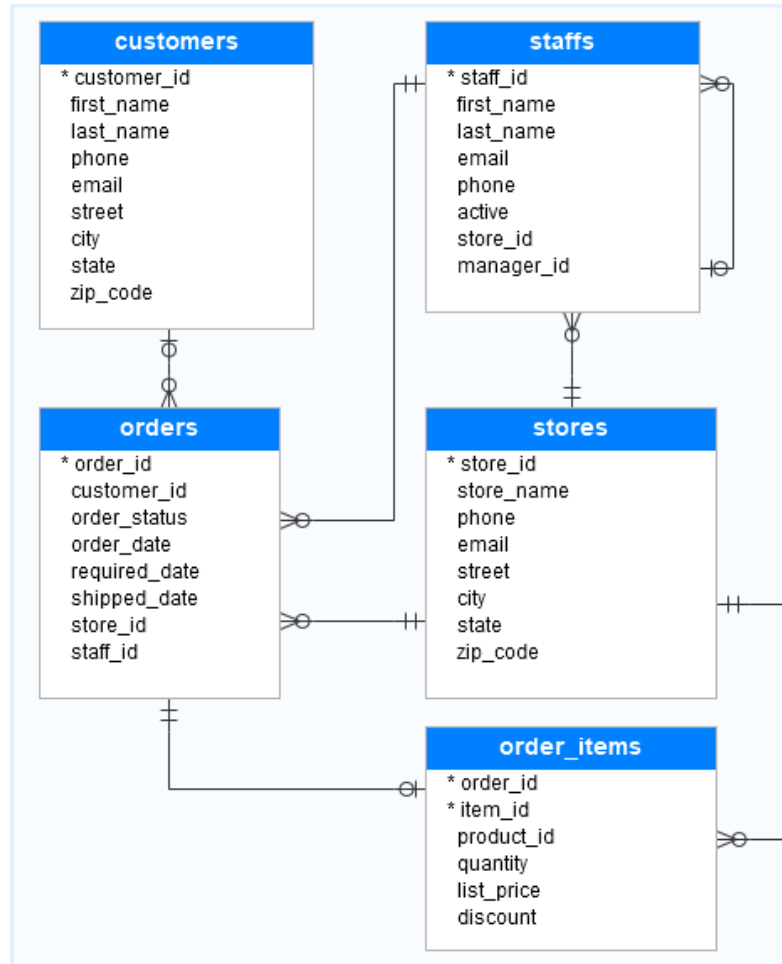




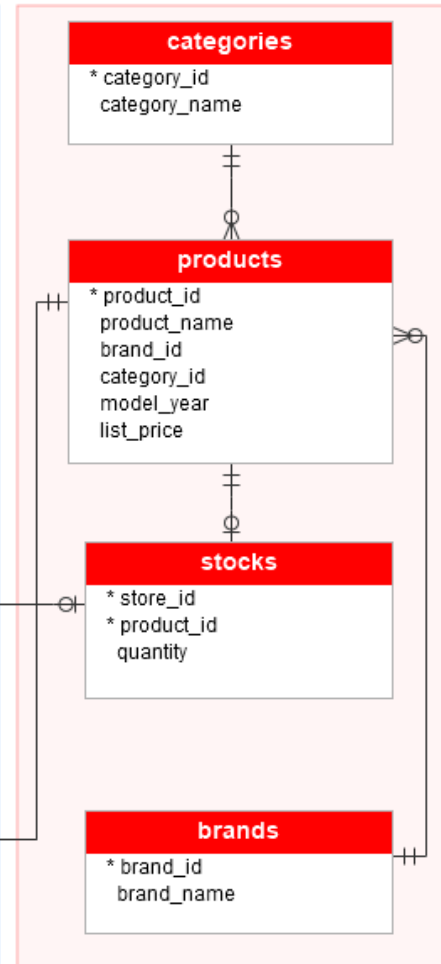
Bases de datos

- Un conjunto de datos ordenados
- Se gestionan bajo modelos:
 - Relacional (SQL)
 - No relacional (NSQL)

Sales



Production



SQL

Tablas

Column (attribute) Table (relation)

Row (tuple)

Primary key


Data value

CustomerID	FirstName	LastName	Birthdate
XY001	John	Doe	April 18, 1929
BR092	Mary	Green	March 4, 1980
PD500	Francesca	de la Gillebert	September 12, 1959
WI308	John	Green	March 4, 1980

Pandas

Es una librería basada en Numpy que ofrece estructuras de datos y herramientas para realizar operaciones básicas de análisis de datos, tales como su obtención, indexación, limpieza, transformación, combinación y selección.

- Pandas es una librería que proporciona herramientas analíticas y estructura de datos con alto rendimiento y facilidad de uso.
- En particular, la clase DataFrame es útil para representación y manipulación de datos tabulados (hojas de cálculo, tabla SQL, etc.)
- Posee herramientas robustas de lectura/escritura de datos desde ficheros con formatos conocidos como: CSV, XLS, SQL, HDF5, entre otros.
- Brinda diversas funciones de tratamiento de datos que nos permiten filtrar, agregar o eliminar datos.
- Se importa de la siguiente manera:



```
import pandas as pd
```

Series

- ▶ Una serie se compone de una secuencia de valores e índices, que se pueden acceder con los atributos `values` e `index`.
- ▶ Por defecto, el índice se crea automáticamente empezando en 0 y con paso 1.
- ▶ Los elementos de una serie se pueden acceder con el índice usando `[]`.
- ▶ Al crear una serie, se pueden especificar sus índices.

```
In [3]: import pandas as pd

In [11]: s = pd.Series([1, "b", 3.14, 4])
s
Out[11]: 0    1
         1    b
         2   3.14
         3    4
         dtype: object

In [10]: s = pd.Series([5, 12, 34.0, 4])
s
Out[10]: 0    5.0
         1   12.0
         2   34.0
         3    4.0
         dtype: float64

In [12]: s = pd.Series([54, 62, 73, 87],
index = ['Ene', 'Feb', 'Mar', 'Abr'])

In [13]: s
Out[13]: Ene    54
         Feb    62
         Mar    73
         Abr    87
         dtype: int64
```

Diagram labels: Índices, Valores, Tipo

Series

- Creación de una serie a partir de Numpy Array:
 - `pd.Series(np.arange(3))`
 - `pd.Series(np.arange(3), index=['a', 'b', 'c'])`

a	0
b	1
c	2

- ▶ Como en las secuencias, se pueden seleccionar elementos desde entre índices. El resultado es otra serie.
- ▶ También es posible seleccionar usando condiciones booleanas.

```
In [8]: s['Feb':'Abr']
Out[8]: Feb      62
        Mar      73
        Abr      87
        dtype: int64
```

```
In [7]: s[(s > 60) & (s < 80)]
Out[7]: Feb      62
        Mar      73
        dtype: int64
```

Series

- ▶ Los objetos series cuentan con métodos para cálculo estadístico como `mean`, `median`, `std`...
- ▶ El resultado de una operación aritmética (`+`, `-`, `/`, `*`, `...`):
 - Las series se alinean automáticamente según en sus etiquetas.
 - Cuando no hay alineación total, se obtiene la unión de todos los índices.
 - Se usa `NaN` para las etiquetas que no estén presentes en todas las series



Tablas en pandas

Series

	apples
0	3
1	2
2	0
3	1

+

Series

	oranges
0	0
1	3
2	7
3	2

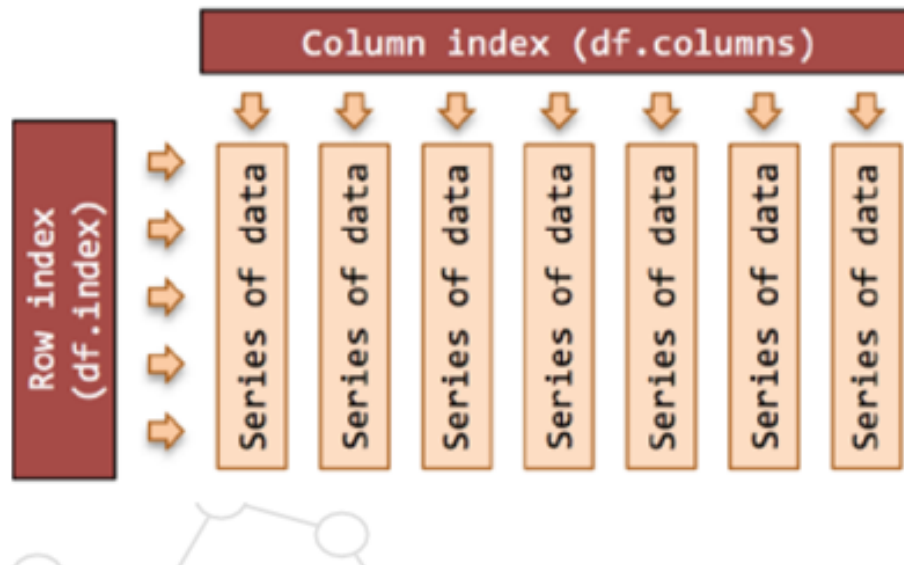
=

DataFrame

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

Data Frames

- El Data Frame permite almacenar y manipular datos tabulados en filas de observaciones y columnas de variables.



		Columns			
		Name	Score	Attempts	Qualify
Rows	0	Anastasia	12.5	1	yes
	1	Dima	9.0	3	no
	2	Katherine	16.5	2	yes
	3	James	NaN	3	no
	4	Emily	9.0	2	no
		Data			

Data Frames

Es una estructura de datos de 2 dimensiones de distinto tipos de datos, un data frame puede venir de las siguientes estructuras de datos:

NumPy Array, Listas, Diccionarios, Series, 2D NumPyArray.

- Data Frames que provienen de diccionarios de series.

```
d = {'c1': pd.Series(['A', 'B', 'C']), 'c2': pd.Series([1, 2., 3., 4.])}
```

```
df = pd.DataFrame(d)
```

- Data Frames que provienen de diccionarios de listas.

```
d = {'c1': ['A', 'B', 'C', 'D'], 'c2': [1, 2.0, 3.0, 4.0]}
```

```
df = pd.DataFrame(d)
```

Data Frames

- ▶ En su creación se pueden especificar etiquetas en las filas (índices) y en las columnas:
 - Usa una secuencia de enteros positivos por defecto.

```
In [11]: pd.DataFrame(np.random.rand(4, 2),  
                      columns=['valor1', 'valor2'],  
                      index=['a', 'b', 'c', 'd'])
```

Out[11]:

	valor1	valor2
a	0.959328	0.442022
b	0.329641	0.399674
c	0.365967	0.126291
d	0.928304	0.215927

- **columns**: devuelve los nombres de las columnas.
- **values**: devuelve un *array* con los valores.
- **shape**: devuelve las dimensiones.

Data Frames

- ▶ Cuando se definen, los nombres de las columnas de un DF se convierten en atributos.
 - Se pueden usar para seleccionar columnas (series).
- ▶ Usando el nombre dentro de corchetes [] también se puede seleccionar.
 - Si se usan dobles corchetes, [[]], en lugar de una serie devuelve un DF.

```
In [46]: df= pd.DataFrame(np.random.rand(4, 2),  
columns=['valor1', 'valor2'],  
index=['a', 'b', 'c', 'd'])  
  
In [47]: df.valor1  
Out[47]: a    0.658438  
b    0.442153  
c    0.937371  
d    0.377319  
Name: valor1, dtype: float64  
  
In [48]: df['valor1']  
Out[48]: a    0.658438  
b    0.442153  
c    0.937371  
d    0.377319  
Name: valor1, dtype: float64  
  
In [49]: df[['valor1']]  
Out[49]:
```

	valor1
a	0.658438
b	0.442153
c	0.937371
d	0.377319

Data Frames

► Funciones adicionales usadas frecuentemente

- Transposición (`df.T`)
- Mostrar índice (`df.index`)
- Mostrar columnas (`df.columns`)
- Mostrar valores en crudo (`df.values`)
- Ordenar por índice (`df.sort_index(...)`)
- Ordenar por columna (`sort_values(...)`)
- Primeros 5 elementos (`df.head()`)
- Últimos 5 elementos (`df.tail()`)



Data Frames

- ▶ Con el operador `iloc` se usan posiciones para seleccionar.

iloc	[<selección de fila> ,	<selección de columna>]
	Lista de enteros: [0,2,4]	Lista de enteros: [1,3,5]
	Secuencia de filas: [3:6]	Secuencia de columnas: [2:5]
	Una fila: [4]	Una columna: [4]
	Todas las filas: :	Todas las columnas: :

```
print(df.iloc[3])
print(df.iloc[3:5,0:2])
print(df.iloc[[1,2,4],[0,2]])
```

```
A    1.843015
B    0.782642
C   -0.789224
D   -0.394010
Name: 2013-01-04 00:00:00, dtype: float64
```

	A	B
2013-01-04	1.843015	0.782642
2013-01-05	-0.627971	-0.006470

	A	C
2013-01-02	0.160351	-0.073032
2013-01-03	1.917280	0.795580
2013-01-05	-0.627971	-0.384700

Data Frames

- ▶ Con el operador `loc` se usan índices y etiquetas para seleccionar.

loc	[<selección de fila> ,	<selección de columna>]
	Etiqueta/índice: "Ene"	Nombre de columna: "valor1"
	Lista de índices: ["Ene", "Feb"]	Lista de nombres de columnas: ["valor1", "valor2"]
	Secuencia de índices ["Ene": "Jun"]	Secuencia de nombres de columnas: ["valor2": "valor5"]
	Todas las filas: :	Todas las columnas: :
	Expresión booleana: <code>df.valor1 > 10</code> o <code>df['valor1']</code>	Todas las columnas: :

```
print(df)
print(df.loc[dates[1]])
print(df.loc['2013-01-02'])
print(df.loc[:, ['A', 'B']]) #selecciona todas las filas y las columnas A y B
print(df.loc['2013-01-01': '2013-01-02', ['A', 'B']])
#selecciona las filas '2013-01-01' a '2013-01-02' y las columnas A y B
```

	A	B	C	D
2013-01-01	0.294724	0.519698	-0.156688	1.283312
2013-01-02	0.160351	0.489060	-0.073032	-1.042722
2013-01-03	1.917280	-0.283947	0.795580	0.052200
2013-01-04	1.843015	0.782642	-0.789224	-0.394010
2013-01-05	-0.627971	-0.006470	-0.384700	1.029400
2013-01-06	1.271107	0.349719	-0.816250	0.485858

Realizando agrupaciones

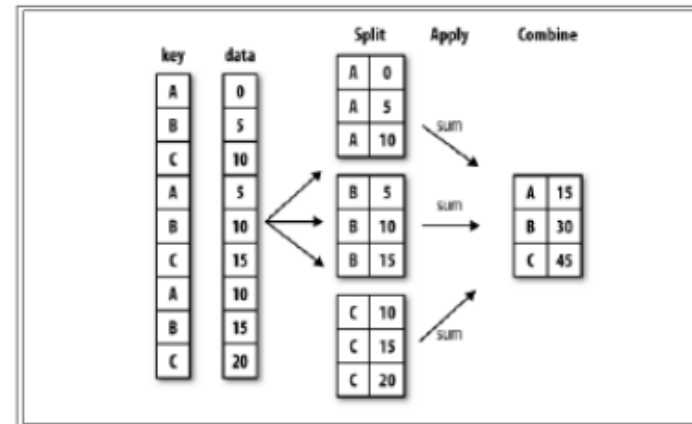
- ▶ El método `groupby` permite hacer agregaciones de datos y, después, operaciones en base a esas agregaciones:

- Sum.
- Mean.
- Median.
- Std.
- Size.
- Describe.
- Quantile.

- ▶ Permite aplicar la estrategia *split-transform-combine* a DFs y series.

```
iris.groupby('Species').mean()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Species				
setosa	5.006	3.428	1.462	0.246
versicolor	5.936	2.770	4.260	1.326
virginica	6.588	2.974	5.552	2.026



Análisis exploratorio inicial

- ▶ Calcula **estadísticos significativos** que resumen la tendencia central, la dispersión y al forma del set de datos. Excluye valores nulos.
- ▶ Para **datos de tipo objeto** (*strings* o *timestamps*) presenta el total de valores, el número de valores únicos, el valor más común (top) y su frecuencia (freq)
- ▶ Para datos numéricos o mixtos.

```
In [52]: iris['Species'].describe()
Out[52]: count          150
         unique           3
         top      versicolor
         freq           50
         Name: Species, dtype: object
```


```
In [49]: iris.describe()
Out[49]:
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Importar y exportar datos con pandas

https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html

Format Type	Data Description	Reader	Writer
text	CSV	<code>read_csv</code>	<code>to_csv</code>
text	JSON	<code>read_json</code>	<code>to_json</code>
text	HTML	<code>read_html</code>	<code>to_html</code>
text	Local clipboard	<code>read_clipboard</code>	<code>to_clipboard</code>
binary	MS Excel	<code>read_excel</code>	<code>to_excel</code>
binary	HDF5 Format	<code>read_hdf</code>	<code>to_hdf</code>
binary	Feather Format	<code>read_feather</code>	<code>to_feather</code>
binary	Msgpack	<code>read_msgpack</code>	<code>to_msgpack</code>
binary	Stata	<code>read_stata</code>	<code>to_stata</code>
binary	SAS	<code>read_sas</code>	
binary	Python Pickle Format	<code>read_pickle</code>	<code>to_pickle</code>
SQL	SQL	<code>read_sql</code>	<code>to_sql</code>
SQL	Google Big Query	<code>read_gbq</code>	<code>to_gbq</code>

- 
- ▶ Estas funciones son muy versátiles ya que cuentan con docenas de parámetros opcionales que permiten definir cómo se van a cargar los datos.

Importar y exportar datos con pandas

- ▶ Importar datos en formato csv.

```
d = pd.read_csv('Data/students.csv')
```

- ▶ Importar datos en formato xls.

```
d = pd.read_excel('Data/students.xls')
```

Exportar datos de una data frame a xls y csv

In [56]:

```
# Exportar datos de un data frame a un xls  
d.to_excel('E:/JFB/Python/PYTHON_1/DATOS/sample_data.xls')
```

In [57]:

```
# Exportar datos de un data frame a un csv  
d.to_csv('E:/JFB/Python/PYTHON_1/DATOS/sample_data.csv')
```



Importar datos en formato JSON

JSON (JavaScript Object Notation) es un formato para el intercambios de datos, básicamente JSON describe los datos con una sintaxis dedicada que se usa para identificar y gestionar los datos.

Importar el paquete JSON.

- ▶ Importar el paquete JSON.

```
import json
```

```
json_data = open('Data/students.json')
```

```
data = json.load(json_data)
```

```
json_data.close()
```