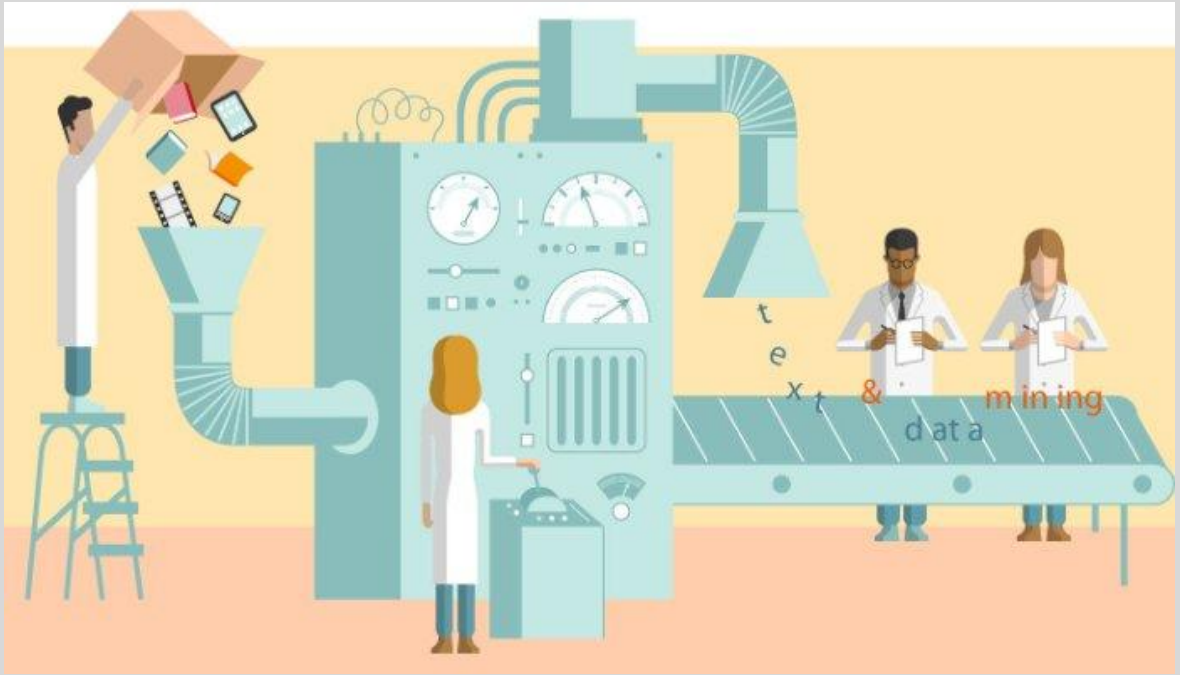


DigitalHouse > Coding School

DATA SCIENCE

MÓDULO 2

Joins con Pandas

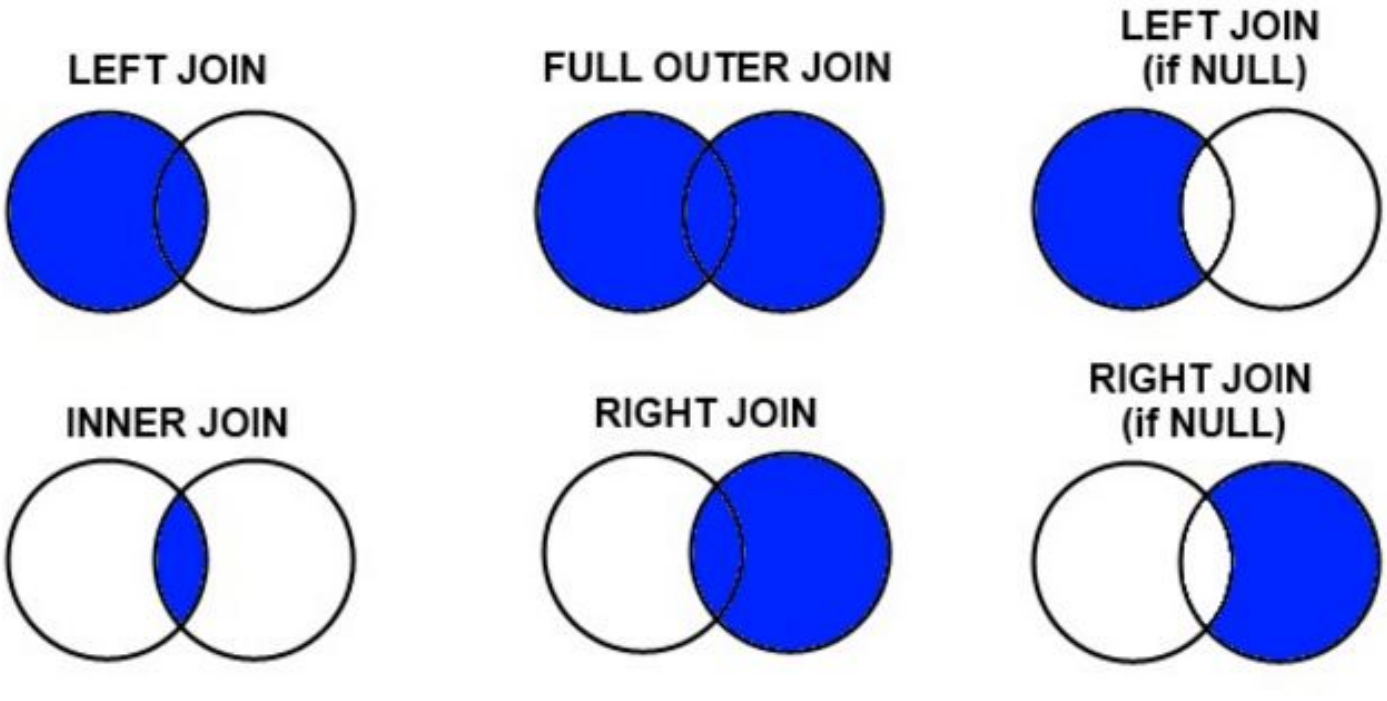
Joins con Pandas

1. Aprender a unir distintas fuentes de información con Pandas

2. Aprender los distintos tipos de Joins, para qué sirven y cómo implementarlos con Pandas

3. Aprender a manejar la función shift() para generar rezagos sobre los datos

JOINS con Diagramas de Venn



COMPARACIONES ENTRE PANDAS Y SQL

- Por ser un lenguaje de consulta y manipulación de datos, Pandas presenta muchas similitudes con SQL.

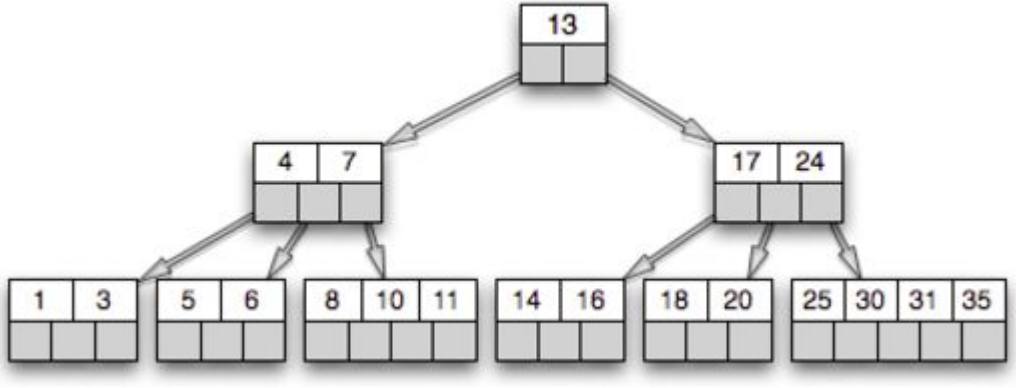
- Pandas fue diseñado para **cubrir y extender** la funcionalidad de SQL.

- Una diferencia importante entre Pandas y SQL, es que las bases de datos trabajan “en disco”, los datos residen en el disco rígido y de allí se leen o se modifican.

- Pandas, en cambio, trabaja con los datos en memoria RAM.
  Algunas opciones para persistir en disco los datos trabajados con Pandas son escribir un csv o conectarse a una base de datos para guardar los resultados.

ÍNDICES EN PANDAS Y SQL

- Tanto en SQL como en Pandas, es importante utilizar índices.
- Los índices son una versión ordenada de una columna o un grupo de columnas.
- Son una forma de transformar las búsquedas desordenadas en búsquedas ordenadas.
- En Pandas y SQL los índices modifican la performance de las sentencias de tipo JOIN.



SENTENCIA SELECT

Para completar la funcionalidad de la sentencia SELECT, las columnas se puede seleccionar con “fancy indexing” y la cantidad de registros deseada con la función head().

```sql
SELECT total_bill, tip, smoker, time
FROM tips
LIMIT 5;
```

```
In [6]: tips[['total_bill', 'tip', 'smoker', 'time']].head(5)
Out[6]:
   total_bill   tip smoker    time
0       16.99  1.01     No  Dinner
1       10.34  1.66     No  Dinner
2       21.01  3.50     No  Dinner
3       23.68  3.31     No  Dinner
4       24.59  3.61     No  Dinner
```

SENTENCIA WHERE

La forma de aplicar filtros sobre Dataframes es con boolean indexing.

```sql
SELECT *
FROM tips
WHERE time = 'Dinner'
LIMIT 5;
```

```
In [7]: tips[tips['time'] == 'Dinner'].head(5)
Out[7]:
   total_bill   tip     sex smoker  day    time  size
0       16.99  1.01  Female     No  Sun  Dinner     2
1       10.34  1.66    Male     No  Sun  Dinner     3
2       21.01  3.50    Male     No  Sun  Dinner     3
3       23.68  3.31    Male     No  Sun  Dinner     2
4       24.59  3.61  Female     No  Sun  Dinner     4
```

SENTENCIA WHERE

Los filtros pueden hacerse sobre una o varias columnas.
En el caso de Pandas, se puede utilizar series externas también, en tanto se genere una máscara booleana del tamaño del DataFrame.

```sql
-- tips of more than $5.00 at Dinner meals
SELECT *
FROM tips
WHERE time = 'Dinner' AND tip > 5.00;
```

```
# tips of more than $5.00 at Dinner meals
In [11]: tips[(tips['time'] == 'Dinner') & (tips['tip'] > 5.00)]
Out[11]:
     total_bill    tip     sex smoker  day    time  size
23         39.42   7.58    Male     No  Sat  Dinner     4
44         30.40   5.60    Male     No  Sun  Dinner     4
47         32.40   6.00    Male     No  Sun  Dinner     4
52         34.81   5.20  Female     No  Sun  Dinner     4
59         48.27   6.73    Male     No  Sat  Dinner     4
116        29.93   5.07    Male     No  Sun  Dinner     4
155        29.85   5.14  Female     No  Sun  Dinner     5
170        50.81  10.00    Male    Yes  Sat  Dinner     3
172         7.25   5.15    Male    Yes  Sun  Dinner     2
181        23.33   5.65    Male    Yes  Sun  Dinner     2
183        23.17   6.50    Male    Yes  Sun  Dinner     4
211        25.89   5.16    Male    Yes  Sat  Dinner     4
212        48.33   9.00    Male     No  Sat  Dinner     4
214        28.17   6.50  Female    Yes  Sat  Dinner     3
239        29.03   5.92    Male     No  Sat  Dinner     3
```

SENTENCIA WHERE

Pandas también ofrece métodos para crear filtros sobre valores nulos transformándolos en una máscara booleana.

```sql
SELECT *
FROM frame
WHERE col2 IS NULL;
```

```
In [15]: frame[frame['col2'].isnull()]
Out[15]:
  col1 col2
1    B  NaN
```

```sql
SELECT *
FROM frame
WHERE col1 IS NOT NULL;
```

```
In [16]: frame[frame['col1'].notnull()]
Out[16]:
  col1 col2
0    A    F
1    B  NaN
3    C    H
4    D    I
```

SENTENCIA GROUP BY

El groupby de pandas ofrece varios métodos para realizar agrupamientos y transformar cada uno de los grupos. .

```sql
SELECT sex, count(*)
FROM tips
GROUP BY sex;
/*
Female     87
Male      157
*/
```

El equivalente a la sentencia count() de SQL, no es count() sino size() ya que count() cuenta la cantidad de **valores no nulos**, de cada columna por cada grupo.

```
In [17]: tips.groupby('sex').size()
Out[17]:
sex
Female     87
Male      157
dtype: int64
```

```
In [18]: tips.groupby('sex').count()
Out[18]:
        total_bill  tip  smoker  day  time  size
sex
Female          87   87      87   87    87    87
Male           157  157     157  157   157   157
```

SENTENCIA GROUP BY CON AGREGACIÓN

Al igual que en SQL, las funciones de agregación se aplican columna por columna. Una ventaja con respecto a SQL estándar es que podemos crear nuestras propias funciones de agregación. Algunas implementaciones de SQL también lo permiten.

```sql
SELECT day, AVG(tip), COUNT(*)
FROM tips
GROUP BY day;
/*
Fri   2.734737   19
Sat   2.993103   87
Sun   3.255132   76
Thur  2.771452   62
*/
```

```
In [20]: tips.groupby('day').agg({'tip': np.mean, 'day': np.size})
Out[20]:
          tip  day
day
Fri   2.734737   19
Sat   2.993103   87
Sun   3.255132   76
Thur  2.771452   62
```

LEFT OUTER JOIN

```sql
-- show all records from df1
SELECT *
FROM df1
LEFT OUTER JOIN df2
  ON df1.key = df2.key;
```

```
# show all records from df1
In [27]: pd.merge(df1, df2, on='key', how='left')
Out[27]:
  key   value_x   value_y
0   A  0.116174       NaN
1   B -0.318214  0.543581
2   C  0.285261       NaN
3   D  2.169960 -0.426067
4   D  2.169960  1.138079
```

FULL OUTER JOIN

```sql
-- show all records from both tables
SELECT *
FROM df1
FULL OUTER JOIN df2
  ON df1.key = df2.key;
```

```
# show all records from both frames
In [29]: pd.merge(df1, df2, on='key', how='outer')
Out[29]:
  key   value_x   value_y
0   A  0.116174       NaN
1   B -0.318214  0.543581
2   C  0.285261       NaN
3   D  2.169960 -0.426067
4   D  2.169960  1.138079
5   E       NaN  0.086073
```

UNION Y UNION ALL

DigitalHouse> Coding School

UNION ALL: con repetidos

```
SELECT city, rank
FROM df1
UNION ALL
SELECT city, rank
FROM df2;
/*
         city  rank
      Chicago     1
San Francisco     2
New York City     3
      Chicago     1
       Boston     4
  Los Angeles     5
*/
```

```
In [32]: pd.concat([df1, df2])
Out[32]:
           city  rank
0       Chicago     1
1  San Francisco     2
2  New York City     3
0       Chicago     1
1        Boston     4
2   Los Angeles     5
```

UNION: sin repetidos

```
SELECT city, rank
FROM df1
UNION
SELECT city, rank
FROM df2;
-- notice that there is only one Chicago record this time
/*
         city  rank
      Chicago     1
San Francisco     2
New York City     3
       Boston     4
  Los Angeles     5
*/
```

In pandas, you can use **concat()** in conjunction with **drop_duplicates()**.

```
In [33]: pd.concat([df1, df2]).drop_duplicates()
Out[33]:
           city  rank
0       Chicago     1
1  San Francisco     2
2  New York City     3
1        Boston     4
2   Los Angeles     5
```

PANDAS BENCHMARKS

pandas vs R merge benchmarks

Many-to-one

	pandas	data.table	plyr	base::merge
inner	1	5.905	6.35	13.29
outer	1	10.05	9.209	20.25
left	1	2.849	5.918	14.93
right	1	2.05	2.923	16.91

Many-to-many

	pandas	data.table	plyr	base::merge
inner	1	5.194	5.223	18.87
outer	1	10	6.903	33.75
left	1	2.528	4.688	24.46
right	1	1.681	2.05	25.24

Pandas vs sqlite3

	sqlite3	pandas
inner	0.02328	0.01799
outer	0.02324	0.01943
left	0.02324	0.01923

Práctica Guiada (Parte I y II)

Lab