



## 01MIAR - Estructuras de datos, +Pandas



Bryan Justin Antamba

```
In [1]: import pandas as pd  
import numpy as np
```

```
In [2]: pd.__version__
```

```
Out[2]: '2.3.3'
```

## Operaciones en pandas

### Búsqueda

```
In [3]: rand_matrix = np.random.randint(6,size=(2,3))  
frame = pd.DataFrame(rand_matrix , columns=list('ABC'))  
display(frame)
```

	A	B	C
0	5	4	4
1	4	1	0

```
In [4]: # buscando columnas (DataFrame como dic, busca en claves)  
'A' in frame
```

```
Out[4]: True
```

```
In [5]: # buscando valores  
display(frame.isin([3,2])) # --> mask de respuesta (valores que son 3 o 2)
```

	A	B	C
0	False	False	False
1	False	False	False

```
In [6]: # Contar el número de ocurrencias  
print(frame.isin([4]).values.sum())  
display(frame.isin([4])) #devuelve una mask con valores true si el elemento es 4  
print(frame.isin([4]).values)  
type(frame.isin([4]).values) # pandas es una capa alrededor de numpy
```

3

	A	B	C
0	False	True	True
1	True	False	False

```
[[False True True]  
 [ True False False]]
```

```
Out[6]: numpy.ndarray
```

```
In [7]: # Cuántos valores son >= 2  
mask = frame >= 2  
print(mask.values.sum())  
display(mask)
```

4

	A	B	C
0	True	True	True
1	True	False	False

## Ordenación

```
In [8]: from random import shuffle  
  
rand_matrix = np.random.randint(20,size=(5,4))  
  
indices = list(range(5))  
shuffle(indices) # mezcla indices  
  
frame = pd.DataFrame(rand_matrix , columns=list('DACB') , index=indices)  
display(frame)
```

	D	A	C	B
0	11	19	8	12
2	17	7	6	11
3	9	8	4	5
1	3	7	14	5
4	5	17	3	7

```
In [9]: # ordenar por indice
display(frame.sort_index(ascending=False))
display(frame)
```

	D	A	C	B
4	5	17	3	7
3	9	8	4	5
2	17	7	6	11
1	3	7	14	5
0	11	19	8	12

	D	A	C	B
0	11	19	8	12
2	17	7	6	11
3	9	8	4	5
1	3	7	14	5
4	5	17	3	7

```
In [10]: #ordenar por columna
display(frame.sort_index(axis=1, ascending=False))
```

	D	C	B	A
0	11	8	12	19
2	17	6	11	7
3	9	4	5	8
1	3	14	5	7
4	5	3	7	17

```
In [11]: # ordenar filas por valor en columna  
display(frame.sort_values(by='A', ascending=True))
```

	D	A	C	B
2	17	7	6	11
1	3	7	14	5
3	9	8	4	5
4	5	17	3	7
0	11	19	8	12

```
In [12]: # ordenar columnas por valor en fila  
display(frame.sort_values(by=1, axis=1, ascending=True))
```

	D	B	A	C
0	11	12	19	8
2	17	11	7	6
3	9	5	8	4
1	3	5	7	14
4	5	7	17	3

```
In [13]: # ordenar por valor en columna y guardar cambios  
frame.sort_values(by='A', ascending=False, inplace=True)  
display(frame)
```

	D	A	C	B
0	11	19	8	12
4	5	17	3	7
3	9	8	4	5
2	17	7	6	11
1	3	7	14	5

## Ranking

- Construir un ranking de valores

```
In [14]: display(frame)
```

	<b>D</b>	<b>A</b>	<b>C</b>	<b>B</b>
<b>0</b>	11	19	8	12
<b>4</b>	5	17	3	7
<b>3</b>	9	8	4	5
<b>2</b>	17	7	6	11
<b>1</b>	3	7	14	5

```
In [15]: display(frame.rank(method='max', axis=1))
```

	<b>D</b>	<b>A</b>	<b>C</b>	<b>B</b>
<b>0</b>	2.0	4.0	1.0	3.0
<b>4</b>	2.0	4.0	1.0	3.0
<b>3</b>	4.0	3.0	1.0	2.0
<b>2</b>	4.0	2.0	1.0	3.0
<b>1</b>	1.0	3.0	4.0	2.0

```
In [16]: # Imprimir, uno a uno, los valores de la columna 'C' de mayor a menor
for x in frame.sort_values(by='C', ascending=False)[ 'C' ].values:
    print(x)
```

14  
8  
6  
4  
3

## Operaciones

Operaciones matemáticas entre objetos

```
In [17]: matrixA = np.random.randint(100, size=(4,4))
matrixB = np.random.randint(100, size=(4,4))
frameA = pd.DataFrame(matrixA)
frameB = pd.DataFrame(matrixB)
display(frameA)
display(frameB)
```

	0	1	2	3
0	78	10	17	32
1	47	11	46	2
2	11	74	2	59
3	52	59	6	58

	0	1	2	3
0	66	36	64	88
1	52	9	67	41
2	53	61	69	19
3	62	8	51	32

In [18]: *# a través de métodos u operadores*

```
display(frameA + frameB == frameA.add(frameB))
display(frameA + frameB)
```

	0	1	2	3
0	True	True	True	True
1	True	True	True	True
2	True	True	True	True
3	True	True	True	True

	0	1	2	3
0	144	46	81	120
1	99	20	113	43
2	64	135	71	78
3	114	67	57	90

In [19]: `display(frameB - frameA == frameB.sub(frameA))`

```
display(frameB - frameA)
```

	0	1	2	3
0	True	True	True	True
1	True	True	True	True
2	True	True	True	True
3	True	True	True	True

	0	1	2	3
0	-12	26	47	56
1	5	-2	21	39
2	42	-13	67	-40
3	10	-51	45	-26

```
In [20]: # si los frames no son iguales, valor por defecto NaN
frameC = pd.DataFrame(np.random.randint(100,size=(3,3)))
display(frameA)
display(frameC)
display(frameC + frameA)
```

	0	1	2	3
0	78	10	17	32
1	47	11	46	2
2	11	74	2	59
3	52	59	6	58

	0	1	2
0	57	14	46
1	44	36	9
2	3	28	12

	0	1	2	3
0	135.0	24.0	63.0	NaN
1	91.0	47.0	55.0	NaN
2	14.0	102.0	14.0	NaN
3	NaN	NaN	NaN	NaN

```
In [21]: # se puede especificar el valor por defecto con el argumento fill_value
display(frameA.add(frameC, fill_value=0))
```

	0	1	2	3
0	135.0	24.0	63.0	32.0
1	91.0	47.0	55.0	2.0
2	14.0	102.0	14.0	59.0
3	52.0	59.0	6.0	58.0

Operadores aritméticos solo válidos en elementos aceptables

```
In [22]: frameD = pd.DataFrame({0: ['a', 'b'], 1:[ 'd', 'f']})  
display(frameD)  
frameA - frameD
```

	0	1
0	a	d
1	b	f

```
-----  
TypeError                                         Traceback (most recent call last)  
File ~\anaconda3\envs\jupyterlab-maquina2026\Lib\site-packages\pandas\core\ops\array  
_ops.py:218, in _na_arithmetic_op(left, right, op, is_cmp)  
  217     try:  
--> 218         result = func(left, right)  
  219     except TypeError:  
  
File ~\anaconda3\envs\jupyterlab-maquina2026\Lib\site-packages\pandas\core\computati  
on\expressions.py:242, in evaluate(op, a, b, use_numexpr)  
  240     if use_numexpr:  
  241         # error: "None" not callable  
--> 242         return _evaluate(op, op_str, a, b) # type: ignore[misc]  
  243     return _evaluate_standard(op, op_str, a, b)  
  
File ~\anaconda3\envs\jupyterlab-maquina2026\Lib\site-packages\pandas\core\computati  
on\expressions.py:73, in _evaluate_standard(op, op_str, a, b)  
    72         _store_test_result(False)  
---> 73     return op(a, b)  
  
TypeError: unsupported operand type(s) for -: 'int' and 'str'  
  
During handling of the above exception, another exception occurred:  
  
TypeError                                         Traceback (most recent call last)  
Cell In[22], line 3  
      1 frameD = pd.DataFrame({0: ['a','b'],1:['d','f']})  
      2 display(frameD)  
----> 3 frameA - frameD  
  
File ~\anaconda3\envs\jupyterlab-maquina2026\Lib\site-packages\pandas\core\ops\commo  
n.py:76, in _unpack_zerodim_and_defer.<locals>.new_method(self, other)  
    72             return NotImplemented  
    74 other = item_from_zerodim(other)  
---> 76     return method(self, other)  
  
File ~\anaconda3\envs\jupyterlab-maquina2026\Lib\site-packages\pandas\core\arraylik  
e.py:194, in OpsMixin.__sub__(self, other)  
  192 @unpack_zerodim_and_defer("__sub__")  
  193 def __sub__(self, other):  
--> 194     return self._arith_method(other, operator.sub)  
  
File ~\anaconda3\envs\jupyterlab-maquina2026\Lib\site-packages\pandas\core\frame.py:  
7927, in DataFrame._arith_method(self, other, op)  
  7925     def _arith_method(self, other, op):  
  7926         if self._should_reindex_frame_op(other, op, 1, None, None):  
-> 7927             return self._arith_method_with_reindex(other, op)  
  7929         axis: Literal[1] = 1 # only relevant for Series other case  
  7930         other = ops.maybe_prepare_scalar_for_op(other, (self.shape[axis],))  
  
File ~\anaconda3\envs\jupyterlab-maquina2026\Lib\site-packages\pandas\core\frame.py:  
8059, in DataFrame._arith_method_with_reindex(self, right, op)  
  8057     new_left = left.iloc[:, lcols]  
  8058     new_right = right.iloc[:, rcols]  
-> 8059     result = op(new_left, new_right)  
  8061 # Do the join on the columns instead of using left._align_for_op
```

```

8062 # to avoid constructing two potentially large/sparse DataFrames
8063 join_columns, _, _ = left.columns.join(
8064     right.columns, how="outer", level=None, return_indexers=True
8065 )

File ~\anaconda3\envs\jupyterlab-maquina2026\Lib\site-packages\pandas\core\ops\commo
n.py:76, in _unpack_zerodim_and_defer.<locals>.new_method(self, other)
    72         return NotImplemented
    74 other = item_from_zerodim(other)
--> 76 return method(self, other)

File ~\anaconda3\envs\jupyterlab-maquina2026\Lib\site-packages\pandas\core\arraylik
e.py:194, in OpsMixin.__sub__(self, other)
   192 @unpack_zerodim_and_defer("__sub__")
   193 def __sub__(self, other):
--> 194     return self._arith_method(other, operator.sub)

File ~\anaconda3\envs\jupyterlab-maquina2026\Lib\site-packages\pandas\core\frame.py:
7935, in DataFrame._arith_method(self, other, op)
    7932 self, other = self._align_for_op(other, axis, flex=True, level=None)
    7934 with np.errstate(all="ignore"):
-> 7935     new_data = self._dispatch_frame_op(other, op, axis=axis)
    7936 return self._construct_result(new_data)

File ~\anaconda3\envs\jupyterlab-maquina2026\Lib\site-packages\pandas\core\frame.py:
7978, in DataFrame._dispatch_frame_op(self, right, func, axis)
    7972     assert self.columns.equals(right.columns)
    7973     # TODO: The previous assertion `assert right._indexed_same(self)`
    7974     # fails in cases with empty columns reached via
    7975     # _frame_arith_method_with_reindex
    7976
    7977     # TODO operate_blockwise expects a manager of the same type
-> 7978     bm = self._mgr.operate_blockwise(
    7979         # error: Argument 1 to "operate_blockwise" of "ArrayManager" has
    7980         # incompatible type "Union[ArrayManager, BlockManager]"; expected
    7981         # "ArrayManager"
    7982         # error: Argument 1 to "operate_blockwise" of "BlockManager" has
    7983         # incompatible type "Union[ArrayManager, BlockManager]"; expected
    7984         # "BlockManager"
    7985         right._mgr, # type: ignore[arg-type]
    7986         array_op,
    7987     )
    7988     return self._constructor_from_mngr(bm, axes=bm.axes)
7990 elif isinstance(right, Series) and axis == 1:
7991     # axis=1 means we want to operate row-by-row

File ~\anaconda3\envs\jupyterlab-maquina2026\Lib\site-packages\pandas\core\internals
\managers.py:1530, in BlockManager.operate_blockwise(self, other, array_op)
    1526 def operate_blockwise(self, other: BlockManager, array_op) -> BlockManager:
    1527     """
    1528     Apply array_op blockwise with another (aligned) BlockManager.
    1529     """
--> 1530     return operate_blockwise(self, other, array_op)

File ~\anaconda3\envs\jupyterlab-maquina2026\Lib\site-packages\pandas\core\internals
\ops.py:65, in operate_blockwise(left, right, array_op)

```

```

63 res_blk: list[Block] = []
64 for lvals, rvals, locs, left_ea, right_ea, rblk in _iter_block_pairs(left, right):
--> 65     res_values = array_op(lvals, rvals)
66     if (
67         left_ea
68         and not right_ea
69         and hasattr(res_values, "reshape")
70         and not is_1d_only_ea_dtype(res_values.dtype)
71     ):
72         res_values = res_values.reshape(1, -1)

File ~/anaconda3/envs/jupyterlab-maquina2026/Lib/site-packages/pandas/core/ops/array_
_ops.py:283, in arithmetic_op(left, right, op)
279     _bool_arith_check(op, left, right) # type: ignore[arg-type]
281     # error: Argument 1 to "_na_arithmetic_op" has incompatible type
282     # "Union[ExtensionArray, ndarray[Any, Any]]"; expected "ndarray[Any, An
y]"
--> 283     res_values = _na_arithmetic_op(left, right, op) # type: ignore[arg-typ
e]
285 return res_values

File ~/anaconda3/envs/jupyterlab-maquina2026/Lib/site-packages/pandas/core/ops/array_
_ops.py:227, in _na_arithmetic_op(left, right, op, is_cmp)
219 except TypeError:
220     if not is_cmp and (
221         left.dtype == object or getattr(right, "dtype", None) == object
222     ):
(...).225         # Don't do this for comparisons, as that will handle complex
numbers
226         # incorrectly, see GH#32047
--> 227     result = _masked_arith_op(left, right, op)
228 else:
229     raise

File ~/anaconda3/envs/jupyterlab-maquina2026/Lib/site-packages/pandas/core/ops/array_
_ops.py:163, in _masked_arith_op(x, y, op)
161     # See GH#5284, GH#5035, GH#19448 for historical reference
162     if mask.any():
--> 163         result[mask] = op(xrav[mask], yrav[mask])
165 else:
166     if not is_scalar(y):

TypeError: unsupported operand type(s) for -: 'int' and 'str'

```

**Manera correcta:** La intención es realizar una resta, ambos DataFrames deben ser numéricos

In [69]:

```

import pandas as pd

frameA = pd.DataFrame({0: [5, 6], 1: [7, 8]})
frameD = pd.DataFrame({0: [1, 2], 1: [3, 4]})

resultado = frameA - frameD
display(resultado)

```

0	1
0	4 4
1	4 4

## Operaciones entre Series y DataFrames

```
In [ ]: rand_matrix = np.random.randint(10, size=(3, 4))
df = pd.DataFrame(rand_matrix, columns=list('ABCD'))
display(df)

display(df.iloc[0])
display(type(df.iloc[0]))
# uso común, averiguar la diferencia entre una fila y el resto
display(df - df.iloc[0])
display(df.sub(df.iloc[0], axis=1))
# Por columnas cómo se restaría
display(df.sub(df['A'], axis=0))
```

pandas se basa en NumPy, np operadores binarios y unarios son aceptables

Tipo	Operación	Descripción
Unario	<i>abs</i>	Valor absoluto de cada elemento
	<i>sqrt</i>	Raíz cuadrada de cada elemento
	<i>exp</i>	$e^x$ , siendo x cada elemento
	<i>log, log10, log2</i>	Logaritmos en distintas bases de cada elemento
	<i>sign</i>	Retorna el signo de cada elemento (-1 para negativo, 0 o 1 para positivo)
	<i>ceil</i>	Redondea cada elemento por arriba
	<i>floor</i>	Redondea cada elemento por abajo
	<i>isnan</i>	Retorna si cada elemento es Nan
	<i>cos, sin, tan</i>	Operaciones trigonométricas en cada elemento
	<i>arccos, arcsin, arctan</i>	Inversas de operaciones trigonométricas en cada elemento
Binario	<i>add</i>	Suma de dos arrays
	<i>subtract</i>	Resta de dos arrays
	<i>multiply</i>	Multiplicación de dos arrays
	<i>divide</i>	División de dos arrays
	<i>maximum, minimum</i>	Retorna el valor máximo/mínimo de cada pareja de elementos

Tipo	Operación	Descripción
	<i>equal, not_equal</i>	Retorna la comparación (igual o no igual) de cada pareja de elementos
	<i>greater, greater_equal, less, less_equal</i>	Retorna la comparación ( $>$ , $\geq$ , $<$ , $\leq$ respectivamente) de cada pareja de elementos

Aplicación de funciones a medida con lambda

```
In [23]: rand_matrix = np.random.randint(10, size=(3, 4))
frame = pd.DataFrame(rand_matrix, columns=list('ABCD'))
display(frame)

print(frame.apply(lambda x : x.max() - x.min(), axis = 1)) # diferencia por columna
```

	A	B	C	D
<b>0</b>	9	3	9	7
<b>1</b>	8	7	3	8
<b>2</b>	8	0	4	3
	0	6		
	1	5		
	2	8		
				dtype: int32

```
In [24]: def max_min(x):
    return x.max() - x.min()

print(frame.apply(max_min, axis = 0)) # diferencia por columna
```

A	1
B	7
C	6
D	5
	dtype: int32

```
In [25]: # diferencia entre min y max por fila (no columna)
rand_matrix = np.random.randint(10, size=(3, 4))
frame = pd.DataFrame(rand_matrix, columns=list('ABCD'))
display(frame)

print(frame.apply(lambda x : x.max() - x.min(), axis = 1)) # diferencia por fila
```

	A	B	C	D
<b>0</b>	2	2	0	1
<b>1</b>	7	7	4	9
<b>2</b>	9	0	9	2

```
0    2  
1    5  
2    9  
dtype: int32
```

## Estadística descriptiva

- Análisis preliminar de los datos
- Para Series y DataFrame

Operación	Descripción
count	Número de valores no NaN
describe	Conjunto de estadísticas sumarias
min, max	Valores mínimo y máximo
argmin, argmax	Índices posicionales del valor mínimo y máximo
idxmin, idxmax	Índices semánticos del valor mínimo y máximo
sum	Suma de los elementos
mean	Media de los elementos
median	Mediana de los elementos
mad	Desviación absoluta media del valor medio
var	Varianza de los elementos
std	Desviación estándar de los elementos
cumsum	Suma acumulada de los elementos
diff	Diferencia aritmética de los elementos

```
In [26]: diccionario = { "nombre" : ["Marisa", "Laura", "Manuel", "Carlos"], "edad" : [34,34,1  
          "puntos" : [98,12,98,np.nan], "genero": ["F", "F", "M", "M"] }  
frame = pd.DataFrame(diccionario)  
display(frame)  
display(frame.describe()) # datos generales de elementos
```

	nombre	edad	puntos	genero
0	Marisa	34	98.0	F
1	Laura	34	12.0	F
2	Manuel	11	98.0	M
3	Carlos	30	NaN	M

	edad	puntos
<b>count</b>	4.000000	3.000000
<b>mean</b>	27.250000	69.333333
<b>std</b>	10.996211	49.652123
<b>min</b>	11.000000	12.000000
<b>25%</b>	25.250000	55.000000
<b>50%</b>	32.000000	98.000000
<b>75%</b>	34.000000	98.000000
<b>max</b>	34.000000	98.000000

```
In [27]: # operadores básicos
print(frame.sum())
```

```
display(frame)
print(frame.sum(axis=1, numeric_only=True))
```

```
nombre      MarisaLauraManuelCarlos
edad                  109
puntos                 208.0
genero                  FFMM
dtype: object
```

	nombre	edad	puntos	genero
0	Marisa	34	98.0	F
1	Laura	34	12.0	F
2	Manuel	11	98.0	M
3	Carlos	30	NaN	M

```
0      132.0
1      46.0
2     109.0
3     30.0
dtype: float64
```

```
In [28]: frame.mean(numeric_only=True)
```

```
Out[28]: edad      27.250000
puntos    69.333333
dtype: float64
```

```
In [29]: frame.cumsum()
```

```
Out[29]:
```

	nombre	edad	puntos	genero
0	Marisa	34	98.0	F
1	MarisaLaura	68	110.0	FF
2	MarisaLauraManuel	79	208.0	FFM
3	MarisaLauraManuelCarlos	109	NaN	FFMM

```
In [30]:
```

```
frame.count(axis=1)
```

```
Out[30]:
```

```
0    4  
1    4  
2    4  
3    3  
dtype: int64
```

```
In [31]:
```

```
print(frame['edad'].std())
```

```
10.996211468804457
```

```
In [32]:
```

```
frame['edad'].idxmax()
```

```
Out[32]:
```

```
0
```

```
In [33]:
```

```
frame['puntos'].idxmin()
```

```
Out[33]:
```

```
1
```

```
In [34]:
```

```
# frame con las filas con los valores maximos de una columna  
print(frame['puntos'].max())
```

```
display(frame[frame['puntos'] == frame['puntos'].max()])
```

```
98.0
```

	nombre	edad	puntos	genero
0	Marisa	34	98.0	F
2	Manuel	11	98.0	M

```
In [35]:
```

```
frame["ranking"] = frame["puntos"].rank(method='max')
```

```
In [36]:
```

```
display(frame)
```

	nombre	edad	puntos	genero	ranking
0	Marisa	34	98.0	F	3.0
1	Laura	34	12.0	F	1.0
2	Manuel	11	98.0	M	3.0
3	Carlos	30	NaN	M	NaN

## Agregaciones

```
In [37]: display(frame)
df = frame.groupby('genero').count()
display(df)
```

	nombre	edad	puntos	genero	ranking
0	Marisa	34	98.0	F	3.0
1	Laura	34	12.0	F	1.0
2	Manuel	11	98.0	M	3.0
3	Carlos	30	NaN	M	NaN

	nombre	edad	puntos	ranking
genero				
F	2	2	2	2
M	2	2	1	1

```
In [38]: # si es Nan descarta la fila
df = frame.groupby('puntos').count()
display(df)
```

	nombre	edad	genero	ranking
puntos				
12.0	1	1	1	1
98.0	2	2	2	2

```
In [39]: display(frame.groupby('genero').mean(numeric_only=True))
```

	edad	puntos	ranking
genero			
F	34.0	55.0	2.0
M	20.5	98.0	3.0

```
In [40]: display(frame.groupby('genero').max())
```

	nombre	edad	puntos	ranking
genero				
F	Marisa	34	98.0	3.0
M	Manuel	30	98.0	3.0

```
In [41]: # funciones de agregación de varias columnas para obtener distintos estadísticos
display(frame.groupby('genero')[['edad', 'puntos']].aggregate(['min', 'mean', 'max'])
```

	edad			puntos		
genero	min	mean	max	min	mean	max
F	34	34.0	34	12.0	55.0	98.0
M	11	20.5	30	98.0	98.0	98.0

```
In [42]: # Filtrado de los datos en el que el conjunto no supera una media determinada
def media(x):
    return x["edad"].mean() > 30

display(frame)
frame.groupby('genero').filter(media)
```

	nombre	edad	puntos	genero	ranking
0	Marisa	34	98.0	F	3.0
1	Laura	34	12.0	F	1.0
2	Manuel	11	98.0	M	3.0
3	Carlos	30	NaN	M	NaN

```
Out[42]:      nombre  edad  puntos  genero  ranking
0   Marisa    34    98.0      F     3.0
1    Laura    34    12.0      F     1.0
```

# Correlaciones

pandas incluye métodos para analizar correlaciones

- Relación matemática entre dos variables (-1 negativamente relacionadas, 1 positivamente relacionadas, 0 sin relación)
- obj.corr(obj2) --> medida de correlación entre los datos de ambos objetos
- <https://blogs.oracle.com/ai-and-datasience/post/introduction-to-correlation>

## Ejemplo Fuel efficiency

- <https://archive.ics.uci.edu/ml/datasets/Auto+MPG>

```
In [43]: import pandas as pd
path = 'http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.csv'

mpg_data = pd.read_csv(path, sep='\s+', header=None,
                      names = ['mpg', 'cilindros', 'desplazamiento', 'potencia',
                               'peso', 'aceleracion', 'año', 'origen', 'nombre'],
                      na_values='?', engine='c')
```

```
In [44]: display(mpg_data.sample(5))
```

	mpg	cilindros	desplazamiento	potencia	peso	aceleracion	año	origen	nombre		
396	28.0	4		120.0	79.0	2625.0		18.6	82	1	ford ranger
238	33.5	4		98.0	83.0	2075.0		15.9	77	1	dodge colt m/m
184	25.0	4		140.0	92.0	2572.0		14.9	76	1	capri ii
16	18.0	6		199.0	97.0	2774.0		15.5	70	1	amc hornet
112	19.0	4		122.0	85.0	2310.0		18.5	73	1	ford pinto

```
In [45]: display(mpg_data.describe(include='all'))
```

	<b>mpg</b>	<b>cilindros</b>	<b>desplazamiento</b>	<b>potencia</b>	<b>peso</b>	<b>aceleracion</b>	
<b>count</b>	398.000000	398.000000	398.000000	392.000000	398.000000	398.000000	398.000000
<b>unique</b>	Nan	Nan	Nan	Nan	Nan	Nan	Nan
<b>top</b>	Nan	Nan	Nan	Nan	Nan	Nan	Nan
<b>freq</b>	Nan	Nan	Nan	Nan	Nan	Nan	Nan
<b>mean</b>	23.514573	5.454774	193.425879	104.469388	2970.424623	15.568090	76.01
<b>std</b>	7.815984	1.701004	104.269838	38.491160	846.841774	2.757689	3.69
<b>min</b>	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	70.00
<b>25%</b>	17.500000	4.000000	104.250000	75.000000	2223.750000	13.825000	73.00
<b>50%</b>	23.000000	4.000000	148.500000	93.500000	2803.500000	15.500000	76.00
<b>75%</b>	29.000000	8.000000	262.000000	126.000000	3608.000000	17.175000	79.00
<b>max</b>	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	82.00

In [46]: `mpg_data.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   mpg               398 non-null    float64
 1   cilindros         398 non-null    int64  
 2   desplazamiento   398 non-null    float64
 3   potencia          392 non-null    float64
 4   peso              398 non-null    float64
 5   aceleracion       398 non-null    float64
 6   año               398 non-null    int64  
 7   origen            398 non-null    int64  
 8   nombre             398 non-null    object 
dtypes: float64(5), int64(3), object(1)
memory usage: 28.1+ KB
```

## Correlaciones entre valores

In [47]: `mpg_data['mpg'].corr(mpg_data['peso']) # + mpg = - peso`

Out[47]: `np.float64(-0.8317409332443354)`

In [48]: `mpg_data['peso'].corr(mpg_data['aceleracion']) # + peso = - aceleracion`

Out[48]: `np.float64(-0.41745731994039337)`

## Correlaciones entre todos los valores

```
In [49]: mpg_data.corr(numeric_only=True)
```

Out[49]:

	mpg	cilindros	desplazamiento	potencia	peso	aceleracion	
mpg	1.000000	-0.775396	-0.804203	-0.778427	-0.831741	0.420289	0
cilindros	-0.775396	1.000000	0.950721	0.842983	0.896017	-0.505419	-0
desplazamiento	-0.804203	0.950721	1.000000	0.897257	0.932824	-0.543684	-0
potencia	-0.778427	0.842983	0.897257	1.000000	0.864538	-0.689196	-0
peso	-0.831741	0.896017	0.932824	0.864538	1.000000	-0.417457	-0
aceleracion	0.420289	-0.505419	-0.543684	-0.689196	-0.417457	1.000000	0
año	0.579267	-0.348746	-0.370164	-0.416361	-0.306564	0.288137	1
origen	0.563450	-0.562543	-0.609409	-0.455171	-0.581024	0.205873	0

```
In [50]: #año y origen no parecen correlacionables
```

```
#eliminar columnas de la correlacion
```

```
corr_data = mpg_data.drop(['año','origen'],axis=1).corr(numeric_only=True)
```

```
display(corr_data)
```

	mpg	cilindros	desplazamiento	potencia	peso	aceleracion	
mpg	1.000000	-0.775396	-0.804203	-0.778427	-0.831741	0.420289	
cilindros	-0.775396	1.000000	0.950721	0.842983	0.896017	-0.505419	
desplazamiento	-0.804203	0.950721	1.000000	0.897257	0.932824	-0.543684	
potencia	-0.778427	0.842983	0.897257	1.000000	0.864538	-0.689196	
peso	-0.831741	0.896017	0.932824	0.864538	1.000000	-0.417457	
aceleracion	0.420289	-0.505419	-0.543684	-0.689196	-0.417457	1.000000	

```
In [51]: # representación gráfica matplotlib
```

```
import matplotlib.pyplot as plt
```

```
In [52]: # representación gráfica
```

```
corr_data.style.background_gradient(cmap=plt.get_cmap('RdYlGn'), axis=1)
```

```
Out[52]:
```

	mpg	cilindros	desplazamiento	potencia	peso	aceleracion
mpg	1.000000	-0.775396	-0.804203	-0.778427	-0.831741	0.420289
cilindros	-0.775396	1.000000	0.950721	0.842983	0.896017	-0.505419
desplazamiento	-0.804203	0.950721	1.000000	0.897257	0.932824	-0.543684
potencia	-0.778427	0.842983	0.897257	1.000000	0.864538	-0.689196
peso	-0.831741	0.896017	0.932824	0.864538	1.000000	-0.417457
aceleracion	0.420289	-0.505419	-0.543684	-0.689196	-0.417457	1.000000

```
In [53]:
```

```
# correlación más negativa  
mpg_data.drop(['año','origen'],axis=1).corr(numeric_only=True).idxmin()
```

```
Out[53]:
```

```
mpg          peso  
cilindros      mpg  
desplazamiento  mpg  
potencia       mpg  
peso          mpg  
aceleracion     potencia  
dtype: object
```

```
In [54]:
```

```
# correlación más positiva  
mpg_data.drop(['año','origen'],axis=1).corr(numeric_only=True).idxmax() #consigo m
```

```
Out[54]:
```

```
mpg          mpg  
cilindros    cilindros  
desplazamiento  desplazamiento  
potencia      potencia  
peso          peso  
aceleracion    aceleracion  
dtype: object
```

```
In [55]:
```

```
# tabla similar con las correlaciones más positivas (evitar parejas del mismo valor  
positive_corr = mpg_data.drop(['año','origen'],axis=1).corr(numeric_only=True)  
np.fill_diagonal(positive_corr.values, 0)  
display(positive_corr)  
positive_corr.idxmax()
```

	mpg	cilindros	desplazamiento	potencia	peso	aceleracion
mpg	0.000000	-0.775396	-0.804203	-0.778427	-0.831741	0.420289
cilindros	-0.775396	0.000000	0.950721	0.842983	0.896017	-0.505419
desplazamiento	-0.804203	0.950721	0.000000	0.897257	0.932824	-0.543684
potencia	-0.778427	0.842983	0.897257	0.000000	0.864538	-0.689196
peso	-0.831741	0.896017	0.932824	0.864538	0.000000	-0.417457
aceleracion	0.420289	-0.505419	-0.543684	-0.689196	-0.417457	0.000000

```
Out[55]: mpg          aceleracion
          cilindros      desplazamiento
          desplazamiento  cilindros
          potencia        desplazamiento
          peso            desplazamiento
          aceleracion     mpg
          dtype: object
```

```
In [56]: positive_corr.style.background_gradient(cmap=plt.get_cmap('RdYlGn'), axis=1, vmin=-1, vmax=1)
```

```
Out[56]:
```

	mpg	cilindros	desplazamiento	potencia	peso	aceleracion
mpg	0.000000	-0.775396	-0.804203	-0.778427	-0.831741	0.420289
cilindros	-0.775396	0.000000	0.950721	0.842983	0.896017	-0.505419
desplazamiento	-0.804203	0.950721	0.000000	0.897257	0.932824	-0.543684
potencia	-0.778427	0.842983	0.897257	0.000000	0.864538	-0.689196
peso	-0.831741	0.896017	0.932824	0.864538	0.000000	-0.417457
aceleracion	0.420289	-0.505419	-0.543684	-0.689196	-0.417457	0.000000

## Ejercicios

- Ejercicios para practicar Pandas: <https://github.com/ajcr/100-pandas-puzzles/blob/master/100-pandas-puzzles.ipynb>

1.- Imprime la versión de pandas que se ha importado.

```
In [57]: pd.__version__
```

```
Out[57]: '2.3.3'
```

2.- Cree un DataFrame df a partir de los datos de este diccionario que tenga las etiquetas de índice.

```
In [58]: import numpy as np

data = {'animal': ['cat', 'cat', 'snake', 'dog', 'dog', 'cat', 'snake', 'cat', 'dog',
                  'age': [2.5, 3, 0.5, np.nan, 5, 2, 4.5, np.nan, 7, 3],
                  'visits': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
                  'priority': ['yes', 'yes', 'no', 'yes', 'no', 'no', 'no', 'yes', 'no', 'no']

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

df = pd.DataFrame(data, index=labels)
```

3.- Devuelve las primeras 3 filas del DataFrame df.

```
In [59]: # Muestra Las primeras 3 filas del DataFrame df  
df.head(3)
```

```
Out[59]: animal age visits priority
```

a	cat	2.5	1	yes
b	cat	3.0	3	yes
c	snake	0.5	2	no

4.- Seleccione los datos en las filas [3, 4, 8] y en las columnas ['animal', 'edad'].

```
In [60]: df.iloc[[3, 4, 8]][['animal', 'age']]
```

```
Out[60]: animal age
```

d	dog	NaN
e	dog	5.0
i	dog	7.0

5.- Seleccione las filas donde falta la edad, es decir, es NaN.

```
In [61]: df[df['age'].isna()]
```

```
Out[61]: animal age visits priority
```

d	dog	NaN	3	yes
h	cat	NaN	1	yes

6.- Select the rows the age is between 2 and 4 (inclusive).

```
In [62]: df[df['age'].between(2, 4)]
```

```
Out[62]: animal age visits priority
```

a	cat	2.5	1	yes
b	cat	3.0	3	yes
f	cat	2.0	3	no
j	dog	3.0	1	no

7.- Calcule la suma de todas las visitas en df (es decir, encuentre el número total de visitas).

```
In [63]: df['visits'].sum()
```

```
Out[63]: np.int64(19)
```

8.- Agregue una nueva fila 'k' a df con los valores que elija para cada columna. Luego elimine esa fila para devolver el DataFrame original.

```
In [64]: # Agregar la nueva fila 'k'  
df.loc['k'] = ['cat', 1.5, 2, 'yes']  
df
```

```
Out[64]:   animal  age  visits  priority  
a      cat    2.5      1     yes  
b      cat    3.0      3     yes  
c    snake    0.5      2     no  
d      dog    NaN      3     yes  
e      dog    5.0      2     no  
f      cat    2.0      3     no  
g    snake    4.5      1     no  
h      cat    NaN      1     yes  
i      dog    7.0      2     no  
j      dog    3.0      1     no  
k      cat    1.5      2     yes
```

```
In [65]: # Eliminar la fila 'k'  
df = df.drop('k')  
df
```

```
Out[65]:   animal  age  visits  priority  
a      cat    2.5      1     yes  
b      cat    3.0      3     yes  
c    snake    0.5      2     no  
d      dog    NaN      3     yes  
e      dog    5.0      2     no  
f      cat    2.0      3     no  
g    snake    4.5      1     no  
h      cat    NaN      1     yes  
i      dog    7.0      2     no  
j      dog    3.0      1     no
```

9.- Ordene df primero por los valores de 'edad' en orden descendente, luego por el valor de la columna 'visitas' en orden ascendente (por lo que la fila i debería ser la primera y la fila d debería ser la última).

```
In [66]: df.sort_values(by=['age', 'visits'], ascending=[False, True])
```

```
Out[66]:
```

	animal	age	visits	priority
i	dog	7.0	2	no
e	dog	5.0	2	no
g	snake	4.5	1	no
j	dog	3.0	1	no
b	cat	3.0	3	yes
a	cat	2.5	1	yes
f	cat	2.0	3	no
c	snake	0.5	2	no
h	cat	NaN	1	yes
d	dog	NaN	3	yes

10.- En la columna "animal", cambie las entradas "anaconda" a "python".

```
In [67]: # Reemplazar "anaconda" por "python" en la columna animal  
df['animal'] = df['animal'].replace('anaconda', 'python')  
df
```

```
Out[67]:
```

	animal	age	visits	priority
a	cat	2.5	1	yes
b	cat	3.0	3	yes
c	snake	0.5	2	no
d	dog	NaN	3	yes
e	dog	5.0	2	no
f	cat	2.0	3	no
g	snake	4.5	1	no
h	cat	NaN	1	yes
i	dog	7.0	2	no
j	dog	3.0	1	no

[https://github.com/BryanAntamba/machine2026-  
PrimerParcial/blob/main/SegundoParcial/PAO25\\_25\\_08\\_Python\\_%2BPandas%20.ipynb](https://github.com/BryanAntamba/machine2026-PrimerParcial/blob/main/SegundoParcial/PAO25_25_08_Python_%2BPandas%20.ipynb)

In [ ]: