Ejercicios Pandas

January 21, 2026

0.1 01MIAR - Estructuras de datos, +Pandas

0.2 Nombre: *Adriel Bedoya*

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

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

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

1 Operaciones en pandas

Búsqueda

[4]:
```
#Generamos una matriz con números random enteros hasta el 6, con un tamaño de␣
 ↪matriz de 2 filas x 3 col
rand_matrix = np.random.randint(6,size=(2,3))
#entramos con el método de dataframe, hacemos referencia a la matríz y␣
 ↪almacenamos los names de las col como A , B , C
frame = pd.DataFrame(rand_matrix , columns=list('ABC'))
#MOSTRAMOS
display(frame)
```

```
   A  B  C
0  5  0  4
1  1  4  5
```

[5]:
```
# buscando columnas (DataFrame como dic, busca en claves)
'A' in frame
```

[5]:
```
True
```

[8]:
```
# buscando valores
#isin() comprueba celda por celda si el valor está dentro de la lista [3, 2]
#Saldrá true si la celda contiene 3 o 2
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
```

[10]:
```
# Contar el número de ocurrencias

#Crea una mask bool dará true si es 4, sino false, values convierte en array de␣
 ↪numpy y sum() cuenta cuántas veces aparece el valor 4 en todo el DataFrame.
print(frame.isin([4]).values.sum())
print("----------------------------------------")
#devuelve una mask con valores true si el elemento es 4
display(frame.isin([4]))
print("----------------------------------------")
#Imprime el array de NumPy que hay debajo del DataFrame.
print(frame.isin([4]).values)
print("----------------------------------------")
type(frame.isin([4]).values) # pandas es una capa alrededor de numpy
```

```
2
----------------------------------------
       A      B      C
0  False  False   True
1  False   True  False
----------------------------------------
[[False False  True]
 [False  True False]]
----------------------------------------
```

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

[10]:
```
numpy.ndarray
```

[11]:
```
# Cuántos valores son >= 2

#Creamos la condición para el dataframe, todos los valores mayores o iguales a 2
mask = frame >= 2
#Ponemos mask y que vaya viendo cuántos números cumplen la condición
print(mask.values.sum())
#print
display(mask)
```

```
4
```

	A	B	C
0	True	False	True
1	False	True	True

Ordenación

[13]:
```
from random import shuffle #shuffle es el módulo estándar random

#Genera una matriz NumPy de tamaño 5×4 y los valores son enteros aleatorios␣
 ↪entre 0 y 19.
rand_matrix = np.random.randint(20,size=(5,4))
#Creamos una lista de índices del 0 al 4
indices = list(range(5))
#Mezcla la lista indices directamente, No crea una nueva lista.
shuffle(indices)

#inserta los datos para el dataframe, crea las columnas con DACB y asigna el␣
 ↪índice mezclado
frame = pd.DataFrame(rand_matrix , columns=list('DACB'), index=indices)
#print
display(frame)
```

	D	A	C	B
3	4	2	10	14
4	18	3	1	1
2	18	19	7	4
0	0	9	18	9
1	8	10	6	6

[16]:
```
# ordenar por índice

#Ordenamos los índices de manera descendente
```

```
display(frame.sort_index(ascending=False))
print("-------------------------------------")
#Como estaría sin ordenar
display(frame)
```

```
    D   A   C   B
4  18   3   1   1
3   4   2  10  14
2  18  19   7   4
1   8  10   6   6
0   0   9  18   9

-------------------------------------

    D   A   C   B
3   4   2  10  14
4  18   3   1   1
2  18  19   7   4
0   0   9  18   9
1   8  10   6   6
```

[18]:
```
#ordenar por columna
#Recordemos que axis = 0 es para filas, axis = 1 es para col y axis = 2 es para␣
 ↪los bloques
#me da tok ver así, le pondré en ascendente
display(frame.sort_index(axis=1, ascending=True))
```

```
    A   B   C   D
3   2  14  10   4
4   3   1   1  18
2  19   4   7  18
0   9   9  18   0
1  10   6   6   8
```

[19]:
```
# ordenar filas por valor en columna

#Toda la columna "A" estará ordenada de manera ascendente
display(frame.sort_values(by='A', ascending=True))
```

```
    D   A   C   B
3   4   2  10  14
4  18   3   1   1
0   0   9  18   9
1   8  10   6   6
2  18  19   7   4
```

[20]:
```
# ordenar columnas por valor en fila
#Ordena valores en un DataFrame.
#las columnas se reordenan, no las filas.
```

```
#by=1 indica que el criterio de ordenación es la fila con índice 1.

display(frame.sort_values(by=1, axis=1, ascending=True))
```

```
    B   C   D   A
3  14  10   4   2
4   1   1  18   3
2   4   7  18  19
0   9  18   0   9
1   6   6   8  10
```

[21]:
```
# ordenar por valor en columna y guardar cambios
frame.sort_values(by='A', ascending=False, inplace=True)
display(frame)
```

```
    D   A   C   B
2  18  19   7   4
1   8  10   6   6
0   0   9  18   9
4  18   3   1   1
3   4   2  10  14
```

Ranking

- Construir un ranking de valores

[22]:
```
display(frame)
```

```
    D   A   C   B
2  18  19   7   4
1   8  10   6   6
0   0   9  18   9
4  18   3   1   1
3   4   2  10  14
```

[23]:
```
#rank() asigna rangos numéricos a los valores del DataFrame según su orden.
#rank(min_rango, max_rango)
#Indica que el ranking se hace por filas.
#Cada fila se procesa independientemente y las columnas de una misma fila se␣
 ↪comparan entre sí.
#Si fuera axis=0 (default), el ranking sería por columnas.
display(frame.rank(method='max', axis=1))
```

```
     D    A    C    B
2  3.0  4.0  2.0  1.0
1  3.0  4.0  2.0  2.0
0  1.0  3.0  4.0  3.0
4  4.0  3.0  2.0  2.0
3  2.0  1.0  3.0  4.0
```

[24]:
```
# 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)
```

```
18
10
7
6
1
```

2 Operaciones

Operaciones matemáticas entre objetos

[25]:
```
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	42	34	46	99
1	36	87	22	91
2	93	64	93	99
3	64	89	59	37

	0	1	2	3
0	17	30	33	39
1	96	64	94	83
2	10	64	23	90
3	31	19	95	27

[26]:
```
# a través de métodos u operadores
#Compara si sumar frameA + frameB es igual al frameA.add(frameB)
display(frameA + frameB == frameA.add(frameB)) #Verdadero

display(frameA + frameB) #Suma
```

	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	59	64	79	138
1	132	151	116	174
2	103	128	116	189
3	95	108	154	64

[27]:
```
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	-25	-4	-13	-60
1	60	-23	72	-8
2	-83	0	-70	-9
3	-33	-70	36	-10

[28]:
```
# 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	42	34	46	99
1	36	87	22	91
2	93	64	93	99
3	64	89	59	37

	0	1	2
0	9	8	78
1	20	94	15
2	84	51	22

	0	1	2	3
0	51.0	42.0	124.0	NaN
1	56.0	181.0	37.0	NaN
2	177.0	115.0	115.0	NaN
3	NaN	NaN	NaN	NaN

[29]:
```
# se puede especificar el valor por defecto con el argumento fill_value
display(frameA.add(frameC, fill_value=0))
```

	0	1	2	3
0	51.0	42.0	124.0	99.0
1	56.0	181.0	37.0	91.0
2	177.0	115.0	115.0	99.0
3	64.0	89.0	59.0	37.0

Operadores aritméticos solo válidos en elementos aceptables

[30]:
```
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\jupyter_pdf\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\jupyter_pdf\Lib\site-packages\pandas\core\computation\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\jupyter_pdf\Lib\site-packages\pandas\core\computation\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[30], line 3
      1 frameD = pd.DataFrame({0: ['a','b'],1:['d','f']})
      2 display(frameD)
----> 3 frameA - frameD

File ~\anaconda3\envs\jupyter_pdf\Lib\site-packages\pandas\core\ops\common.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\jupyter_pdf\Lib\site-packages\pandas\core\arraylike.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\jupyter_pdf\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\jupyter_pdf\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\jupyter_pdf\Lib\site-packages\pandas\core\ops\common.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\jupyter_pdf\Lib\site-packages\pandas\core\arraylike.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\jupyter_pdf\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\jupyter_pdf\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" ha
   7980         # incompatible type "Union[ArrayManager, BlockManager]"; expect d
   7981         # "ArrayManager"
   7982         # error: Argument 1 to "operate_blockwise" of "BlockManager" ha
   7983         # incompatible type "Union[ArrayManager, BlockManager]"; expect d
   7984         # "BlockManager"
   7985         right._mgr,  # type: ignore[arg-type]
   7986         array_op,
   7987     )
   7988     return self._constructor_from_mgr(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\jupyter_pdf\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\jupyter_pdf\Lib\site-packages\pandas\core\internals\ops.p :
 ↪65, in operate_blockwise(left, right, array_op)
     63 res_blks: 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\jupyter_pdf\Lib\site-packages\pandas\core\ops\array_ops.p :
 ↪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 ␣
 ↪Any]"
--> 283     res_values = _na_arithmetic_op(left, right, op)  # type:␣
 ↪ignore[arg-type]
```

```
    285 return res_values

File ~\anaconda3\envs\jupyter_pdf\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\jupyter_pdf\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'
```

Operaciones entre Series y DataFrames

[ ]:
```
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	*abs*	Valor absoluto de cada elemento
	sqrt	Raíz cuadrada de cada elemento

Tipo	Operación	Descripción
	exp	e^x, siendo x cad elemento
	log, log10, log2	Logaritmos en distintas bases de cada elemento
	sign	Retorna el signo de cada elemento (-1 para negativo, 0 o 1 para positivo)
	ceil	Redondea cada elemento por arriba
	floor	Redondea cada elemento por abajo
	isnan	Retorna si cada elemento es Nan
	cos, sin, tan	Operaciones trigonométricas en cada elemento
	arccos, arcsin, arctan	Inversas de operaciones trigonométricas en cada elemento
Binario	*add*	Suma de dos arrays
	substract	Resta de dos arrays
	multiply	Multiplicación de dos arrays
	divide	División de dos arrays
	maximum, minimum	Retorna el valor máximo/mínimo de cada pareja de elementos
	equal, not_equal	Retorna la comparación (igual o no igual) de cada pareja de elementos
	greater, greater_equal, less, less_equal	Retorna la comparación (>, >=, <, <= respectivamente) de cada pareja de elementos

Aplicación de funciones a medida con lambda

[31]:
```
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
```

```
0  4  6  4  2
1  7  0  9  7
2  9  1  9  9
```

```
0    4
1    9
2    8
dtype: int32
```

[32]:
```
def max_min(x):
    return x.max() - x.min()

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

```
A    5
B    6
C    5
D    7
dtype: int32
```

[33]:
```
# 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
0  9  9  6  1
1  8  5  9  1
2  3  6  8  0
```

```
0    8
1    8
2    8
dtype: int32
```

3 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

Operación	Descripción
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

[34]:
```
diccionario = { "nombre" : ["Marisa","Laura","Manuel", "Carlos"], "edad" :␣
↪[34,34,11, 30],
               "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
count   4.000000   3.000000
mean   27.250000  69.333333
std    10.996211  49.652123
min    11.000000  12.000000
25%    25.250000  55.000000
50%    32.000000  98.000000
75%    34.000000  98.000000
max    34.000000  98.000000
```

[35]:
```
# 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
```

[36]:
```
frame.mean(numeric_only=True)
```

[36]:
```
edad      27.250000
puntos    69.333333
dtype: float64
```

[37]:
```
frame.cumsum()
```

[37]:
```
                    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
```

[38]:
```
frame.count(axis=1)
```

[38]:
```
0    4
1    4
2    4
3    3
dtype: int64
```

[39]:
```
print(frame['edad'].std())
```

```
10.996211468804457
```

[40]:
```
frame['edad'].idxmax()
```

[40]:
```
0
```

[41]:
```
frame['puntos'].idxmin()
```

[41]:
```
1
```

[42]:
```
# 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
```

[43]:
```
frame["ranking"] = frame["puntos"].rank(method='max')
```

[44]:
```
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
```

3.1 Agregaciones

[45]:
```
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
```

[46]:
```
# 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
```

[47]:
```
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
```

[48]:
```
display(frame.groupby('genero').max())
```

```
        nombre  edad  puntos  ranking
genero
```

```
F       Marisa    34    98.0      3.0
M       Manuel    30    98.0      3.0
```

[49]:
```
# funciones de agregación de varias columnas para obtener distintos estadísticos
display(frame.groupby('genero')[['edad', 'puntos']].aggregate(['min', 'mean',␣
 ↪'max']))
```

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

[50]:
```
# 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
```

[50]:
```
    nombre  edad  puntos genero  ranking
 0  Marisa    34    98.0      F      3.0
 1   Laura    34    12.0      F      1.0
```

3.2 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-datascience/post/introduction-to-correlation

3.2.1 Ejemplo Fuel efficiency

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

[51]:
```
import pandas as pd
path = 'http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/
 ↪auto-mpg.data'

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

[52]:
```
display(mpg_data.sample(5))
```

```
      mpg  cilindros  desplazamiento  potencia    peso  aceleracion  año  \
245  36.1          4            98.0      66.0  1800.0         14.4   78
106  12.0          8           350.0     180.0  4499.0         12.5   73
250  19.4          8           318.0     140.0  3735.0         13.2   78
337  32.4          4           107.0      72.0  2290.0         17.0   80
44   13.0          8           400.0     175.0  5140.0         12.0   71

     origen                    nombre
245       1                ford fiesta
106       1  oldsmobile vista cruiser
250       1             dodge diplomat
337       3               honda accord
44        1        pontiac safari (sw)
```

[53]:
```
display(mpg_data.describe(include='all'))
```

```
                mpg   cilindros  desplazamiento    potencia         peso  \
count   398.000000  398.000000      398.000000  392.000000   398.000000
unique         NaN         NaN             NaN         NaN          NaN
top            NaN         NaN             NaN         NaN          NaN
freq           NaN         NaN             NaN         NaN          NaN
mean     23.514573    5.454774      193.425879  104.469388  2970.424623
std       7.815984    1.701004      104.269838   38.491160   846.841774
min       9.000000    3.000000       68.000000   46.000000  1613.000000
25%      17.500000    4.000000      104.250000   75.000000  2223.750000
50%      23.000000    4.000000      148.500000   93.500000  2803.500000
75%      29.000000    8.000000      262.000000  126.000000  3608.000000
max      46.600000    8.000000      455.000000  230.000000  5140.000000

        aceleracion         año      origen      nombre
count    398.000000  398.000000  398.000000         398
unique          NaN         NaN         NaN         305
top             NaN         NaN         NaN  ford pinto
freq            NaN         NaN         NaN           6
mean      15.568090   76.010050    1.572864         NaN
std        2.757689    3.697627    0.802055         NaN
min        8.000000   70.000000    1.000000         NaN
25%       13.825000   73.000000    1.000000         NaN
50%       15.500000   76.000000    1.000000         NaN
75%       17.175000   79.000000    2.000000         NaN
max       24.800000   82.000000    3.000000         NaN
```

[54]:
```
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
```

3.2.2 Correlaciones entre valores

[55]:
```
mpg_data['mpg'].corr(mpg_data['peso']) # + mpg = - peso
```

[55]:
```
np.float64(-0.8317409332443354)
```

[56]:
```
mpg_data['peso'].corr(mpg_data['aceleracion']) # + peso = - aceleracion
```

[56]:
```
np.float64(-0.41745731994039337)
```

3.2.3 Correlaciones entre todos los valores

[57]:
```
mpg_data.corr(numeric_only=True)
```

[57]:

	mpg	cilindros	desplazamiento	potencia	peso	aceleracion	año	origen
mpg	1.000000	-0.775396	-0.804203	-0.778427	-0.831741	0.420289	0.579267	0.563450
cilindros	-0.775396	1.000000	0.950721	0.842983	0.896017	-0.505419	-0.348746	-0.562543
desplazamiento	-0.804203	0.950721	1.000000	0.897257	0.932824	-0.543684	-0.370164	-0.609409
potencia	-0.778427	0.842983	0.897257	1.000000	0.864538	-0.689196	-0.416361	-0.455171
peso	-0.831741	0.896017	0.932824	0.864538	1.000000	-0.417457	-0.306564	-0.581024
aceleracion	0.420289	-0.505419	-0.543684	-0.689196	-0.417457	1.000000	0.288137	0.205873
año	0.579267	-0.348746	-0.370164	-0.416361	-0.306564	0.288137	1.000000	0.180662
origen	0.563450	-0.562543	-0.609409	-0.455171	-0.581024	0.205873	0.180662	1.000000

[58]:
```
#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  \
mpg             1.000000  -0.775396       -0.804203 -0.778427 -0.831741
cilindros      -0.775396   1.000000        0.950721  0.842983  0.896017
desplazamiento -0.804203   0.950721        1.000000  0.897257  0.932824
potencia       -0.778427   0.842983        0.897257  1.000000  0.864538
peso           -0.831741   0.896017        0.932824  0.864538  1.000000
aceleracion     0.420289  -0.505419       -0.543684 -0.689196 -0.417457

                aceleracion
mpg                0.420289
cilindros         -0.505419
desplazamiento    -0.543684
potencia          -0.689196
peso              -0.417457
aceleracion        1.000000
```

[60]:
```
# representación gráfica matplotlib
import matplotlib.pyplot as plt
```

[61]:
```
# representación gráfica
corr_data.style.background_gradient(cmap=plt.get_cmap('RdYlGn'), axis=1)
```

[61]:
```
<pandas.io.formats.style.Styler at 0x1e4fc0ec6d0>
```

[62]:
```
# correlación más negativa
mpg_data.drop(['año','origen'],axis=1).corr(numeric_only=True).idxmin()
```

[62]:
```
mpg                 peso
cilindros            mpg
desplazamiento       mpg
potencia             mpg
peso                 mpg
aceleracion      potencia
dtype: object
```

[63]:
```
# correlación más positiva
mpg_data.drop(['año','origen'],axis=1).corr(numeric_only=True).idxmax() ␣
↪#consigo misma....
```

[63]:
```
mpg                       mpg
cilindros             cilindros
desplazamiento    desplazamiento
potencia                potencia
```

```
peso                         peso
aceleracion           aceleracion
dtype: object
```

[64]:
```
# 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  \
mpg             0.000000  -0.775396       -0.804203 -0.778427 -0.831741
cilindros      -0.775396   0.000000        0.950721  0.842983  0.896017
desplazamiento -0.804203   0.950721        0.000000  0.897257  0.932824
potencia       -0.778427   0.842983        0.897257  0.000000  0.864538
peso           -0.831741   0.896017        0.932824  0.864538  0.000000
aceleracion     0.420289  -0.505419       -0.543684 -0.689196 -0.417457

                aceleracion
mpg                0.420289
cilindros         -0.505419
desplazamiento    -0.543684
potencia          -0.689196
peso              -0.417457
aceleracion        0.000000
```

[64]:
```
mpg                   aceleracion
cilindros          desplazamiento
desplazamiento          cilindros
potencia           desplazamiento
peso               desplazamiento
aceleracion                   mpg
dtype: object
```

[65]:
```
positive_corr.style.background_gradient(cmap=plt.get_cmap('RdYlGn'), axis=1,vmin=-1.0, vmax=1.0)
```

[65]:
```
<pandas.io.formats.style.Styler at 0x1e4fc260510>
```

3.3 Ejercicios

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

[66]:
```
#2. Print the version of pandas that has been imported.
version = pd.__version__
print(version)
```

2.3.3

[68]:
```
#4. Create a DataFrame df from this dictionary data which has the index labels.

import numpy as np

data = {'animal': ['cat', 'cat', 'snake', 'dog', 'dog', 'cat', 'snake', 'cat',␣
 ↪'dog', '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)
display(df)
```

	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

[70]:
```
#6. Return the first 3 rows of the DataFrame df.
df.head(3) #Las 3 primeras filas
```

[70]:

	animal	age	visits	priority
a	cat	2.5	1	yes
b	cat	3.0	3	yes
c	snake	0.5	2	no

[75]:
```
#8. Select the data in rows [3, 4, 8] and in columns ['animal', 'age'].
#filas 3,4,8 ; col 'animal' , 'age'
df.index = range(len(df))  # Obtenemos los índices de forma numérica
df.loc[[3, 4, 8], ['animal', 'age']] #Ahora ya podemos hacer loc
```

[75]:

	animal	age
3	dog	NaN
4	dog	5.0
8	dog	7.0

[76]:
```
#10. Select the rows where the age is missing, i.e. it is NaN.
df[df['age'].isna()]
```

[76]:

	animal	age	visits	priority
3	dog	NaN	3	yes
7	cat	NaN	1	yes

[81]:
```
#12. Select the rows the age is between 2 and 4 (inclusive).
df[(df['age'] >= 2) & (df['age'] <= 4)]
```

[81]:

	animal	age	visits	priority
0	cat	2.5	1	yes
1	cat	3.0	3	yes
5	cat	2.0	3	no
9	dog	3.0	1	no

[82]:
```
#14. Calculate the sum of all visits in df (i.e. find the total number of␣
↪visits).
total_visits = df['visits'].sum()
print(total_visits)
```

```
19
```

[83]:
```
#16. Append a new row 'k' to df with your choice of values for each column.␣
↪Then delete that row to return the original DataFrame.

# Crear la nueva fila como un diccionario
new_row = {'animal': 'rabbit', 'age': 3, 'visits': 1, 'priority': 'high'}

# Añadir la fila al DataFrame
df.loc['k'] = new_row

# Mostrar el DataFrame con la nueva fila
display(df)
```

	animal	age	visits	priority
0	cat	2.5	1	yes
1	cat	3.0	3	yes
2	snake	0.5	2	no
3	dog	NaN	3	yes
4	dog	5.0	2	no
5	cat	2.0	3	no
6	snake	4.5	1	no
7	cat	NaN	1	yes
8	dog	7.0	2	no
9	dog	3.0	1	no
k	rabbit	3.0	1	high

[84]:
```
#18. Sort df first by the values in the 'age' in decending order, then by the␣
 ↪value in the 'visits' column in ascending order (so row i should be first,␣
 ↪and row d should be last).
df_sorted = df.sort_values(
    by=['age', 'visits'],      # columnas para ordenar
    ascending=[False, True]    # False → descendente, True → ascendente
)

display(df_sorted)
```

	animal	age	visits	priority
8	dog	7.0	2	no
4	dog	5.0	2	no
6	snake	4.5	1	no
9	dog	3.0	1	no
k	rabbit	3.0	1	high
1	cat	3.0	3	yes
0	cat	2.5	1	yes
5	cat	2.0	3	no
2	snake	0.5	2	no
7	cat	NaN	1	yes
3	dog	NaN	3	yes

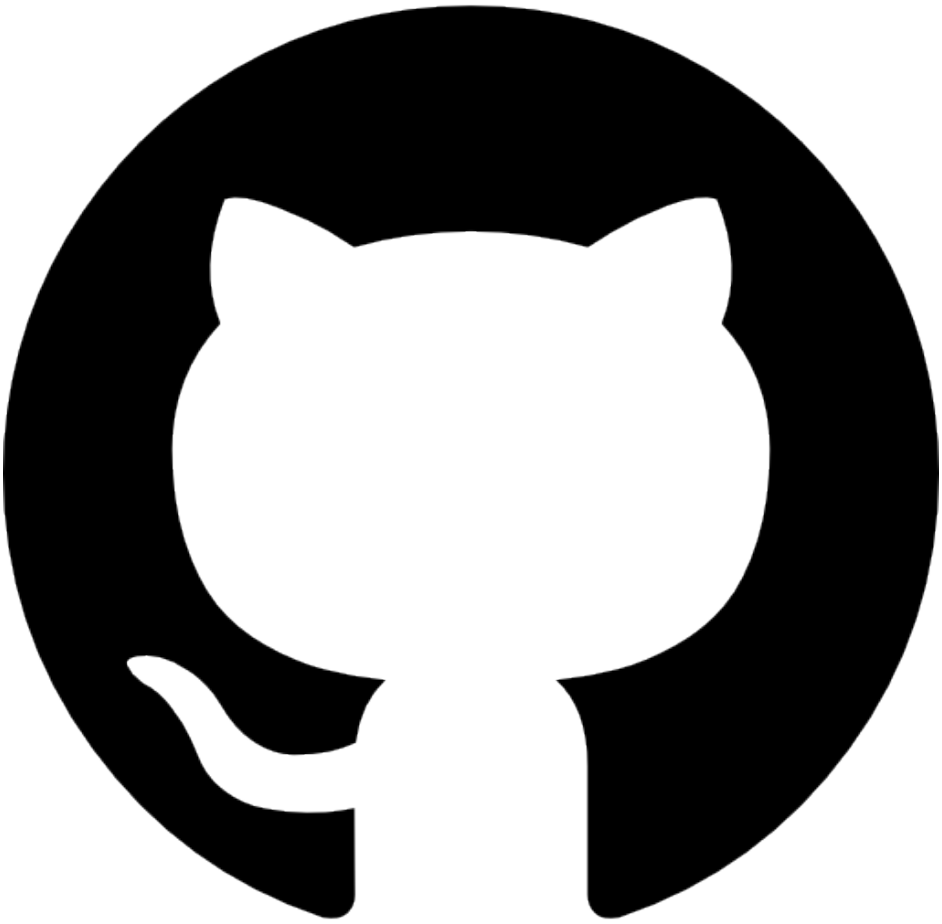
[87]:
```
#20. In the 'animal' column, change the 'snake' entries to 'python'.

#Hice 2, pero la primera me pareció más fácil
#1
df_20 = df['animal'] = df['animal'].replace('snake', 'python')
#2
#df['animal'] = df['animal'].replace('snake', 'python')

display(df_20)
```

```
0       cat
1       cat
2    python
3       dog
4       dog
5       cat
6    python
7       cat
8       dog
9       dog
k    rabbit
Name: animal, dtype: object
```

4 Github



4.0.1 Click aquí para ver el repositorio