 No description has been provided for this image

01MIAR - Estructuras de datos, +Pandas

Luis Pilaguano

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

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

Out[2]:
```
'2.3.1'
```

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	1	2	0
1	1	0	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	True	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
```

```
0
```

	A	B	C
0	False	False	False
1	False	False	False

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

Out[6]: numpy.ndarray

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

```
1
```

	A	B	C
0	False	True	False
1	False	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
2	4	5	15	3
4	9	15	1	3
3	18	11	15	0
1	12	17	1	9
0	7	19	9	1

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

	D	A	C	B
4	9	15	1	3
3	18	11	15	0
2	4	5	15	3
1	12	17	1	9
0	7	19	9	1

	D	A	C	B
2	4	5	15	3
4	9	15	1	3
3	18	11	15	0
1	12	17	1	9
0	7	19	9	1

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

	D	C	B	A
2	4	15	3	5
4	9	1	3	15
3	18	15	0	11
1	12	1	9	17
0	7	9	1	19

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

	D	A	C	B
2	4	5	15	3
3	18	11	15	0
4	9	15	1	3
1	12	17	1	9
0	7	19	9	1

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

	C	B	D	A
2	15	3	4	5
4	1	3	9	15
3	15	0	18	11
1	1	9	12	17
0	9	1	7	19

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	7	19	9	1
1	12	17	1	9
4	9	15	1	3
3	18	11	15	0
2	4	5	15	3

Ranking

- Construir un ranking de valores

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

	D	A	C	B
0	7	19	9	1
1	12	17	1	9
4	9	15	1	3
3	18	11	15	0
2	4	5	15	3

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

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

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)
```

```
15
15
9
1
1
```

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	39	29	91	19
1	77	12	62	60
2	21	94	88	10
3	34	25	88	59

	0	1	2	3
0	56	61	31	81
1	4	68	35	2
2	0	50	66	34
3	11	46	32	99

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	95	90	122	100
1	81	80	97	62
2	21	144	154	44
3	45	71	120	158

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	17	32	-60	62
1	-73	56	-27	-58
2	-21	-44	-22	24
3	-23	21	-56	40

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	39	29	91	19
1	77	12	62	60
2	21	94	88	10
3	34	25	88	59

	0	1	2
0	43	61	95
1	98	29	12
2	35	17	53

	0	1	2	3
0	82.0	90.0	186.0	NaN
1	175.0	41.0	74.0	NaN
2	56.0	111.0	141.0	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	82.0	90.0	186.0	19.0
1	175.0	41.0	74.0	60.0
2	56.0	111.0	141.0	10.0
3	34.0	25.0	88.0	59.0

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\MachineLearningPracticas\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\MachineLearningPracticas\Lib\site-packages\pandas\core\comp
utation\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\MachineLearningPracticas\Lib\site-packages\pandas\core\comp
utation\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\MachineLearningPracticas\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\MachineLearningPracticas\Lib\site-packages\pandas\core\arra
ylike.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\MachineLearningPracticas\Lib\site-packages\pandas\core\fram
e.py:7912, in DataFrame._arith_method(self, other, op)
   7910 def _arith_method(self, other, op):
   7911     if self._should_reindex_frame_op(other, op, 1, None, None):
-> 7912         return self._arith_method_with_reindex(other, op)
   7914     axis: Literal[1] = 1  # only relevant for Series other case
   7915     other = ops.maybe_prepare_scalar_for_op(other, (self.shape[axis],))

File ~\anaconda3\envs\MachineLearningPracticas\Lib\site-packages\pandas\core\fram
e.py:8044, in DataFrame._arith_method_with_reindex(self, right, op)
   8042 new_left = left.iloc[:, lcols]
   8043 new_right = right.iloc[:, rcols]
-> 8044 result = op(new_left, new_right)
   8046 # Do the join on the columns instead of using left._align_for_op
   8047 #  to avoid constructing two potentially large/sparse DataFrames
   8048 join_columns, _, _ = left.columns.join(
   8049     right.columns, how="outer", level=None, return_indexers=True
   8050 )
```

```
File ~\anaconda3\envs\MachineLearningPracticas\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\MachineLearningPracticas\Lib\site-packages\pandas\core\arra
ylike.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\MachineLearningPracticas\Lib\site-packages\pandas\core\fram
e.py:7920, in DataFrame._arith_method(self, other, op)
   7917 self, other = self._align_for_op(other, axis, flex=True, level=None)
   7919 with np.errstate(all="ignore"):
-> 7920     new_data = self._dispatch_frame_op(other, op, axis=axis)
   7921 return self._construct_result(new_data)

File ~\anaconda3\envs\MachineLearningPracticas\Lib\site-packages\pandas\core\fram
e.py:7963, in DataFrame._dispatch_frame_op(self, right, func, axis)
   7957     assert self.columns.equals(right.columns)
   7958     # TODO: The previous assertion `assert right._indexed_same(self)`
   7959     #  fails in cases with empty columns reached via
   7960     #  _frame_arith_method_with_reindex
   7961
   7962     # TODO operate_blockwise expects a manager of the same type
-> 7963     bm = self._mgr.operate_blockwise(
   7964         # error: Argument 1 to "operate_blockwise" of "ArrayManager" has
   7965         # incompatible type "Union[ArrayManager, BlockManager]"; expected
   7966         # "ArrayManager"
   7967         # error: Argument 1 to "operate_blockwise" of "BlockManager" has
   7968         # incompatible type "Union[ArrayManager, BlockManager]"; expected
   7969         # "BlockManager"
   7970         right._mgr,  # type: ignore[arg-type]
   7971         array_op,
   7972     )
   7973     return self._constructor_from_mgr(bm, axes=bm.axes)
   7975 elif isinstance(right, Series) and axis == 1:
   7976     # axis=1 means we want to operate row-by-row

File ~\anaconda3\envs\MachineLearningPracticas\Lib\site-packages\pandas\core\inte
rnals\managers.py:1511, in BlockManager.operate_blockwise(self, other, array_op)
   1507 def operate_blockwise(self, other: BlockManager, array_op) -> BlockManage
r:
   1508     """
   1509     Apply array_op blockwise with another (aligned) BlockManager.
   1510     """
-> 1511     return operate_blockwise(self, other, array_op)

File ~\anaconda3\envs\MachineLearningPracticas\Lib\site-packages\pandas\core\inte
rnals\ops.py: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(lef
t, 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\MachineLearningPracticas\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,
Any]"
--> 283     res_values = _na_arithmetic_op(left, right, op)  # type: ignore[arg-t
ype]
    285 return res_values

File ~\anaconda3\envs\MachineLearningPracticas\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 comp
lex numbers
    226         #  incorrectly, see GH#32047
--> 227         result = _masked_arith_op(left, right, op)
    228     else:
    229         raise

File ~\anaconda3\envs\MachineLearningPracticas\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

In [23]:
```
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))
```

	A	B	C	D
0	8	3	7	3
1	0	1	3	7
2	4	7	0	1

```
A    8
B    3
C    7
D    3
Name: 0, dtype: int32
pandas.core.series.Series
```

	A	B	C	D
0	0	0	0	0
1	-8	-2	-4	4
2	-4	4	-7	-2

	A	B	C	D
0	0	0	0	0
1	-8	-2	-4	4
2	-4	4	-7	-2

	A	B	C	D
0	0	-5	-1	-5
1	0	1	3	7
2	0	3	-4	-3

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
	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

Tipo	Operación	Descripción
	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

In [24]:
```
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 colu
```

	A	B	C	D
0	7	7	1	3
1	4	3	0	4
2	2	3	7	2

```
0    6
1    4
2    5
dtype: int32
```

In [26]:
```
def max_min(x):
    return x.max() - x.min()

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

```
A    5
B    4
C    7
D    2
dtype: int32
```

In [27]:
```
# 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	0	8	6	0
1	7	5	4	3
2	3	9	9	8

```
0    8
1    4
2    6
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 [28]:
```
diccionario = { "nombre" : ["Marisa","Laura","Manuel", "Carlos"], "edad" : [34,3
                "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

In [29]:
```
# 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 [30]:
```
frame.mean(numeric_only=True)
```

Out[30]:
```
edad      27.250000
puntos    69.333333
dtype: float64
```

In [31]:
```
frame.cumsum()
```

Out[31]:

	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 [32]:
```
frame.count(axis=1)
```

Out[32]:
```
0    4
1    4
2    4
3    3
dtype: int64
```

In [33]:
```
print(frame['edad'].std())
```

```
10.996211468804457
```

In [34]:
```
frame['edad'].idxmax()
```

Out[34]:
```
0
```

In [35]:
```
frame['puntos'].idxmin()
```

Out[35]:
```
1
```

In [36]:
```
# 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 [37]:
```
frame["ranking"] = frame["puntos"].rank(method='max')
```

In [38]:
```
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 [40]:
```
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

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

In [41]:
```
# si es Nan descarta la fila
df = frame.groupby('puntos').count()
display(df)
```

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

In [42]:
```
display(frame.groupby('genero').mean(numeric_only=True))
```

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

In [43]:
```
display(frame.groupby('genero').max())
```

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

In [44]:
```
# funciones de agregación de varias columnas para obtener distintos estadísticos
display(frame.groupby('genero')[['edad', 'puntos']].aggregate(['min', 'mean', 'm
```

	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

In [45]:
```
# 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[45]:

	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-datascience/post/introduction-to-correlation

Ejemplo Fuel efficiency

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

In [46]:
```
import pandas as pd
path = 'http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-m

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

```
<>:4: SyntaxWarning: invalid escape sequence '\s'
<>:4: SyntaxWarning: invalid escape sequence '\s'
C:\Users\USER\AppData\Local\Temp\ipykernel_7524\2516009719.py:4: SyntaxWarning: i
nvalid escape sequence '\s'
  mpg_data = pd.read_csv(path, sep='\s+', header=None,
```

In [47]:
```
display(mpg_data.sample(5))
```

	mpg	cilindros	desplazamiento	potencia	peso	aceleracion	año	origen	nombr
122	24.0	4	121.0	110.0	2660.0	14.0	73	2	saa 99
115	15.0	8	350.0	145.0	4082.0	13.0	73	1	chevrol mont carlo
397	31.0	4	119.0	82.0	2720.0	19.4	82	1	chevy 1
194	22.5	6	232.0	90.0	3085.0	17.6	76	1	am horn
288	18.2	8	318.0	135.0	3830.0	15.2	79	1	dodg st. reg

In [48]:
```
display(mpg_data.describe(include='all'))
```

	mpg	cilindros	desplazamiento	potencia	peso	aceleracion	
count	398.000000	398.000000	398.000000	392.000000	398.000000	398.000000	39
unique	NaN	NaN	NaN	NaN	NaN	NaN	
top	NaN	NaN	NaN	NaN	NaN	NaN	
freq	NaN	NaN	NaN	NaN	NaN	NaN	
mean	23.514573	5.454774	193.425879	104.469388	2970.424623	15.568090	7
std	7.815984	1.701004	104.269838	38.491160	846.841774	2.757689	
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	7
25%	17.500000	4.000000	104.250000	75.000000	2223.750000	13.825000	7
50%	23.000000	4.000000	148.500000	93.500000	2803.500000	15.500000	7
75%	29.000000	8.000000	262.000000	126.000000	3608.000000	17.175000	7
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	8

In [49]:
```
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 [50]:
```
mpg_data['mpg'].corr(mpg_data['peso']) # + mpg = - peso
```

Out[50]:
```
np.float64(-0.8317409332443354)
```

In [51]:
```
mpg_data['peso'].corr(mpg_data['aceleracion']) # + peso = - aceleracion
```

Out[51]:
```
np.float64(-0.41745731994039337)
```

Correlaciones entre todos los valores

In [52]:
```
mpg_data.corr(numeric_only=True)
```

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
año	0.579267	-0.348746	-0.370164	-0.416361	-0.306564	0.288137
origen	0.563450	-0.562543	-0.609409	-0.455171	-0.581024	0.205873

In [53]:
```
#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 [57]:
```
# representación gráfica matplotlib
import matplotlib.pyplot as plt
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[57], line 2
      1 # representación gráfica matplotlib
----> 2 import matplotlib.pyplot as plt

ModuleNotFoundError: No module named 'matplotlib'
```

In [58]:
```
# representación gráfica
corr_data.style.background_gradient(cmap=plt.get_cmap('RdYlGn'), axis=1)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[58], line 2
      1 # representación gráfica
----> 2 corr_data.style.background_gradient(cmap=plt.get_cmap('RdYlGn'), axis=1)

NameError: name 'plt' is not defined
```

In [59]:
```
# correlación más negativa
mpg_data.drop(['año','origen'],axis=1).corr(numeric_only=True).idxmin()
```

Out[59]:
```
mpg                 peso
cilindros            mpg
desplazamiento       mpg
potencia             mpg
peso                 mpg
aceleracion      potencia
dtype: object
```

In [60]:
```
# correlación más positiva
mpg_data.drop(['año','origen'],axis=1).corr(numeric_only=True).idxmax()  #consig
```

Out[60]:
```
mpg                       mpg
cilindros             cilindros
desplazamiento    desplazamiento
potencia                potencia
peso                        peso
aceleracion          aceleracion
dtype: object
```

In [61]:
```
# tabla similar con las correlaciones más positivas (evitar parejas del mismo va
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[61]:
```
mpg                aceleracion
cilindros          desplazamiento
desplazamiento         cilindros
potencia           desplazamiento
peso               desplazamiento
aceleracion                  mpg
dtype: object
```

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

Ejercicios

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

Git Hub Pandas

In [ ]:
```

```