INSTITUTO SUPERIOR TECNOLÓGICO QUITO

Excelencia en Educación Superior

0.1 Tercer Modulo- Python , Pandas

python™

Erick Mejia

In [38]:
```
import pandas as pd
import numpy as np
```

In [39]:
```
pd.__version__
```

Out[39]:
```
'2.3.3'
```

Operaciones en pandas

Búsqueda

In [40]:
```
rand_matrix = np.random.randint(6,size=(2,3))
frame = pd.DataFrame(rand_matrix , columns=list('ABC'))
display(frame)
```

	A	B	C
0	4	3	2
1	1	5	5

In [41]:
```
# buscando columnas (DataFrame como dic, busca en claves)
'A' in frame
```

Out[41]: True

In [42]:
```
# buscando valores
display(frame.isin([3,2])) # --> mask de respuesta (valores que son 3 o 2)
```

	A	B	C
0	False	True	True
1	False	False	False

In [43]:
```
# 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
```

```
1
```

	A	B	C
0	True	False	False
1	False	False	False

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

Out[43]: numpy.ndarray

In [44]:
```
# Cuántos valores son >= 2
mask = frame >= 2
print(mask.values.sum())
display(mask)
```

```
5
```

	A	B	C
0	True	True	True
1	False	True	True

Ordenación

In [45]:
```
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	9	2	13	13
4	12	18	8	4
1	10	7	1	12
0	11	6	7	9
3	11	16	6	9

In [46]:
```
# ordenar por índice
display(frame.sort_index(ascending=False))
display(frame)
```

	D	A	C	B
4	12	18	8	4
3	11	16	6	9
2	9	2	13	13
1	10	7	1	12
0	11	6	7	9

	D	A	C	B
2	9	2	13	13
4	12	18	8	4
1	10	7	1	12
0	11	6	7	9
3	11	16	6	9

In [47]:
```
#ordenar por columna
display(frame.sort_index(axis=1, ascending=False))
```

	D	C	B	A
2	9	13	13	2
4	12	8	4	18
1	10	1	12	7
0	11	7	9	6
3	11	6	9	16

In [48]:
```
# ordenar filas por valor en columna
display(frame.sort_values(by='A', ascending=True))
```

	D	A	C	B
2	9	2	13	13
0	11	6	7	9
1	10	7	1	12
3	11	16	6	9
4	12	18	8	4

In [49]:
```
# ordenar columnas por valor en fila
display(frame.sort_values(by=1, axis=1, ascending=True))
```

	C	A	D	B
2	13	2	9	13
4	8	18	12	4
1	1	7	10	12
0	7	6	11	9
3	6	16	11	9

In [50]:
```
# ordenar por valor en columna y guardar cambios
frame.sort_values(by='A', ascending=False, inplace=True)
display(frame)
```

	D	A	C	B
4	12	18	8	4
3	11	16	6	9
1	10	7	1	12
0	11	6	7	9
2	9	2	13	13

Ranking

- Construir un ranking de valores

In [51]:
```
display(frame)
```

	D	A	C	B
4	12	18	8	4
3	11	16	6	9
1	10	7	1	12
0	11	6	7	9
2	9	2	13	13

In [52]:
```
display(frame.rank(method='max', axis=1))
```

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

In [53]:
```
# 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)
```

```
13
8
7
6
1
```

Operaciones

Operaciones matemáticas entre objetos

In [54]:
```
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	6	77	75	79
1	73	60	86	54
2	88	53	59	87
3	49	61	30	10

	0	1	2	3
0	38	63	35	67
1	73	3	31	94
2	24	99	67	60
3	76	62	16	34

In [55]:
```
# 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	44	140	110	146
1	146	63	117	148
2	112	152	126	147
3	125	123	46	44

In [56]:
```
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	32	-14	-40	-12
1	0	-57	-55	40
2	-64	46	8	-27
3	27	1	-14	24

In [57]:
```
# 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	6	77	75	79
1	73	60	86	54
2	88	53	59	87
3	49	61	30	10

	0	1	2
0	3	38	24
1	53	69	79
2	90	27	74

	0	1	2	3
0	9.0	115.0	99.0	NaN
1	126.0	129.0	165.0	NaN
2	178.0	80.0	133.0	NaN
3	NaN	NaN	NaN	NaN

In [58]:
```
# se puede especificar el valor por defecto con el argumento fill_value
display(frameA.add(frameC, fill_value=0))
```

	0	1	2	3
0	9.0	115.0	99.0	79.0
1	126.0	129.0	165.0	54.0
2	178.0	80.0	133.0	87.0
3	49.0	61.0	30.0	10.0

Operadores aritméticos solo válidos en elementos aceptables

In [59]:
```
import pandas as pd

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

print(frameA - frameD)
```

```
   0  1
0  0  2
1  1  2
```

Operaciones entre Series y DataFrames

In [60]:
```
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	5	6	2	3
1	0	7	2	9
2	2	2	8	4

```
A    5
B    6
C    2
D    3
Name: 0, dtype: int32
pandas.core.series.Series
```

	A	B	C	D
0	0	0	0	0
1	-5	1	0	6
2	-3	-4	6	1

	A	B	C	D
0	0	0	0	0
1	-5	1	0	6
2	-3	-4	6	1

	A	B	C	D
0	0	1	-3	-2
1	0	7	2	9
2	0	0	6	2

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

Tipo	Operación	Descripción
	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

In [61]:
```
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	6	4	8	2
1	9	5	8	3
2	1	2	9	5

```
0    6
1    6
2    8
dtype: int32
```

In [62]:
```
def max_min(x):
    return x.max() - x.min()

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

```
A    8
B    3
C    1
D    3
dtype: int32
```

In [63]:
```python
# 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	5	3	4	6
1	9	4	5	9
2	6	3	3	4

```
0    3
1    5
2    3
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

Operación	Descripción
diff	Diferencia aritmética de los elementos

In [64]:
```
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
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 [65]:
```
# 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 [66]:
```
frame.mean(numeric_only=True)
```

Out[66]:
```
edad      27.250000
puntos    69.333333
dtype: float64
```

In [67]:
```
frame.cumsum()
```

Out[67]:

	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 [68]:
```
frame.count(axis=1)
```

Out[68]:
```
0    4
1    4
2    4
3    3
dtype: int64
```

In [69]:
```
print(frame['edad'].std())
```

```
10.996211468804457
```

In [70]:
```
frame['edad'].idxmax()
```

Out[70]:
```
0
```

In [71]:
```
frame['puntos'].idxmin()
```

Out[71]:
```
1
```

In [72]:
```
# 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 [73]:
```
frame["ranking"] = frame["puntos"].rank(method='max')
```

In [74]:
```
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 [75]:
```
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 [76]:
```
# 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 [77]:
```
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 [78]:
```
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 [79]:
```
# 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

In [80]:
```
# 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[80]:

	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 [81]:
```
import pandas as pd
path = 'http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.

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 [82]:
```
display(mpg_data.sample(5))
```

	mpg	cilindros	desplazamiento	potencia	peso	aceleracion	año	origen	nombre
363	22.4	6	231.0	110.0	3415.0	15.8	81	1	buick century
274	20.3	5	131.0	103.0	2830.0	15.9	78	2	audi 5000
272	23.8	4	151.0	85.0	2855.0	17.6	78	1	oldsmobile starfire sx
204	32.0	4	85.0	70.0	1990.0	17.0	76	3	datsun b-210
33	19.0	6	232.0	100.0	2634.0	13.0	71	1	amc gremlin

In [83]:
```
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	398.00
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	76.01
std	7.815984	1.701004	104.269838	38.491160	846.841774	2.757689	3.69
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	70.00
25%	17.500000	4.000000	104.250000	75.000000	2223.750000	13.825000	73.00
50%	23.000000	4.000000	148.500000	93.500000	2803.500000	15.500000	76.00
75%	29.000000	8.000000	262.000000	126.000000	3608.000000	17.175000	79.00
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	82.00

In [84]:
```
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 [85]:
```
mpg_data['mpg'].corr(mpg_data['peso']) # + mpg = - peso
```

Out[85]:
```
np.float64(-0.8317409332443354)
```

In [86]:
```
mpg_data['peso'].corr(mpg_data['aceleracion']) # + peso = - aceleracion
```

Out[86]:
```
np.float64(-0.41745731994039337)
```

Correlaciones entre todos los valores

In [87]:
```
mpg_data.corr(numeric_only=True)
```

Out[87]:

	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 [88]:
```
#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 [89]:
```
# representación gráfica matplotlib
import matplotlib.pyplot as plt
```

In [90]:
```
# representación gráfica
corr_data.style.background_gradient(cmap=plt.get_cmap('RdYlGn'), axis=1)
```

Out[90]:

	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 [91]:
```
# correlación más negativa
mpg_data.drop(['año','origen'],axis=1).corr(numeric_only=True).idxmin()
```

Out[91]:
```
mpg                peso
cilindros           mpg
desplazamiento      mpg
potencia            mpg
peso                mpg
aceleracion    potencia
dtype: object
```

In [92]:
```
# correlación más positiva
mpg_data.drop(['año','origen'],axis=1).corr(numeric_only=True).idxmax()  #consigo m
```

Out[92]:
```
mpg                     mpg
cilindros           cilindros
desplazamiento  desplazamiento
potencia            potencia
peso                    peso
aceleracion      aceleracion
dtype: object
```

In [93]:
```
import numpy as np

# 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[93]:
```
mpg                aceleracion
cilindros          desplazamiento
desplazamiento         cilindros
potencia           desplazamiento
peso               desplazamiento
aceleracion                   mpg
dtype: object
```

In [94]:
```
positive_corr.style.background_gradient(cmap=plt.get_cmap('RdYlGn'), axis=1, vmin=-
```

Out[94]:

	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

2. Imprima la versión de pandas que se ha importado.

In [95]:
```
# Importamos pandas
import pandas as pd

# Mostrar qué versión tengo instalada
print("Versión de pandas:", pd.__version__)
```

```
Versión de pandas: 2.3.3
```

4 Cree un DataFrame df a partir de estos datos del diccionario que tienen las etiquetas de índice.

In [96]:
```
# Importamos numpy y pandas
import numpy as np
import pandas as pd

# Datos de animales
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'

# Etiquetas para las filas
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

# Crear la tabla con los datos y las etiquetas
df = pd.DataFrame(data, index=labels)

# Mostrar la tabla
print(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
```

6. Devuelve las primeras 3 filas del DataFrame df.

In [97]:
```
print(df.head(3))
```

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

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

In [98]:
```
# Importamos pandas
import pandas as pd

# Crear la tabla con datos de animales
tabla = pd.DataFrame({
    'animal': ['cat', 'dog', 'snake', 'cow', 'horse', 'sheep', 'lion', 'tiger', 'be
    'age': [2, 5, 3, 7, 4, 6, 8, 9, 10]
})

# Seleccionar filas 3, 4 y 8, con las columnas animal y age
resultado = tabla.loc[[3, 4, 8], ['animal', 'age']]

# Mostrar el resultado
print(resultado)
```

```
  animal  age
3    cow    7
4  horse    4
8   bear   10
```

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

In [99]:
```
df[df['age'].isna()]
```

Out[99]:

	animal	age	visits	priority
d	dog	NaN	3	yes
h	cat	NaN	1	yes

12. Seleccione las filas cuya edad esté entre 2 y 4 (inclusive).

In [100…
```
df[df['age'].between(2, 4)]
```

Out[100…

	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

14. Calcula la suma de todas las visitas en gl (es decir, encuentra el número total de visitas).

In [101…
```
# Traer pandas y numpy
import pandas as pd
import numpy as np

# Crear tabla con información de animales
```

```
tabla = pd.DataFrame({
    'animal': ['cat', 'dog', 'snake', 'cow', 'horse', 'sheep', 'lion', 'tiger', 'be
    'age': [2, 5, 3, 7, 4, 6, 8, 9, 10],
    'visits': [1, 2, 3, 4, 5, 6, 7, 8, 9]
})

# Sumar todas las visitas
total_visitas = int(tabla['visits'].sum())

# Mostrar el total
print("Total de visitas:", total_visitas)
```

```
Total de visitas: 45
```

16. Añada una nueva fila 'k' a df con los valores que elija para cada columna. Luego, elimine esa fila para devolver el DataFrame original.

In [102…

```
# Agregar una nueva fila con índice 'k'
tabla.loc['k'] = ['elephant', 12, 5]
print("Tabla con la nueva fila:")
print(tabla)

# Eliminar la fila 'k' para volver a la tabla original
tabla = tabla.drop('k')
print("\nTabla sin la fila 'k':")
print(tabla)
```

```
Tabla con la nueva fila:
     animal  age  visits
0       cat    2       1
1       dog    5       2
2     snake    3       3
3       cow    7       4
4     horse    4       5
5     sheep    6       6
6      lion    8       7
7     tiger    9       8
8      bear   10       9
k  elephant   12       5

Tabla sin la fila 'k':
  animal  age  visits
0    cat    2       1
1    dog    5       2
2  snake    3       3
3    cow    7       4
4  horse    4       5
5  sheep    6       6
6   lion    8       7
7  tiger    9       8
8   bear   10       9
```

18. Ordene df primero por los valores en la columna 'edad' en orden descendente, luego por el valor en la columna 'visitas' en orden ascendente (por lo que la fila i debe ser la

primera y la fila d debe ser la última).

In [103…
```
df = df.sort_values(by=['age', 'visits'], ascending=[False, True])
print(df)
```

```
  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
```

20. En la columna "animal", cambie las entradas "serpiente" a "pitón".

In [104…
```
df['animal'] = df['animal'].replace('serpiente', 'pitón')
print(df)
```

```
  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
```

GitHub

https://github.com/Erick-305/machine-learning.git