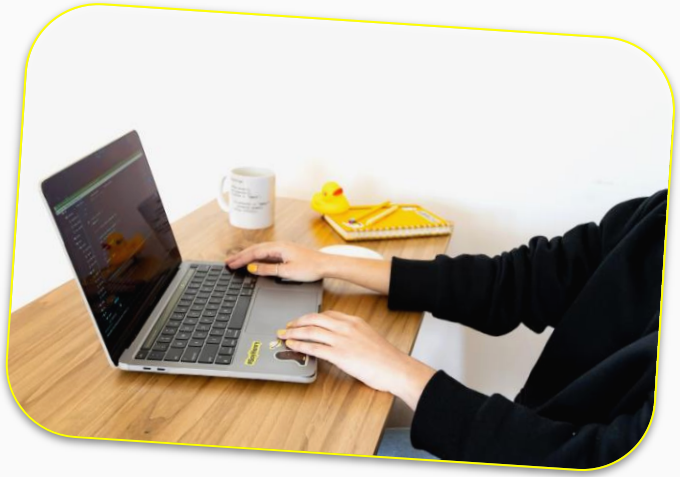
HENRY

Pandas

Data Science

Agenda



- Tipos de datos de Pandas
- Series en Pandas
- DataFrame en Pandas

OBJETIVOS DE LA CLASE

Al finalizar esta lecture estarás en la capacidad de...

→ Usar la librería Pandas para manejo de conjuntos de datos

Pandas

¿Qué es?

Pandas es una librería de Python especializada en el manejo y análisis de estructuras de datos.

Principales características

- Estructuras de datos basadas en los arrays de la librería NumPy pero con nuevas funcionalidades.
- Permite leer y escribir fácilmente archivos en formato CSV, Excel y bases de datos SQL.
- Permite acceder a los datos mediante índices o nombres para filas y columnas.
- Ofrece métodos para reordenar, dividir y combinar conjuntos de datos.
- Permite trabajar con series temporales.
- Realiza todas estas operaciones de manera muy eficiente.

Tipos de datos

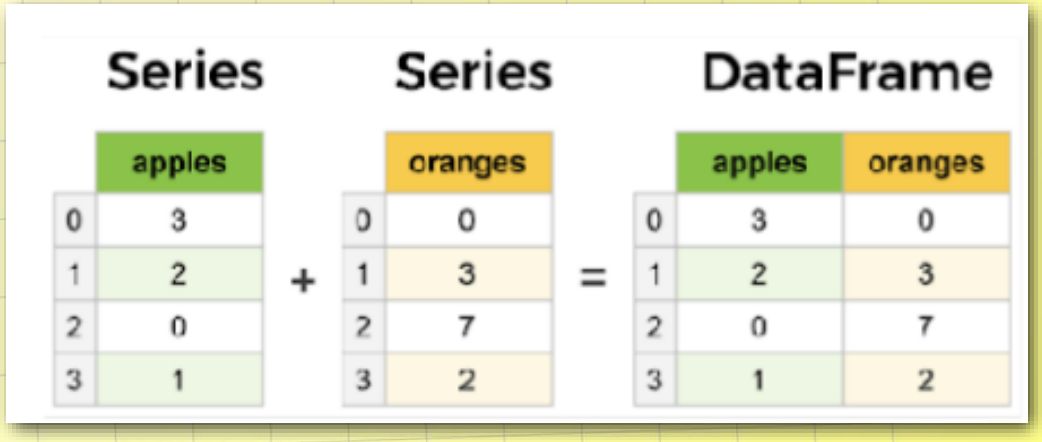
Pandas dispone de estructuras de datos diferentes:

Series

Estructura de una dimensión.

DataFrame

Estructura de dos dimensiones (tablas)



Series en **Pandas**

¿Qué son?

- Estructuras similares a los arrays de una dimensión.
- Sus elementos tienen que ser del mismo tipo, y su tamaño es inmutable, es decir, no se puede cambiar, aunque sí su contenido.
- Dispone de un índice que asocia un nombre a cada elemento de la serie, a través de la cuál se accede al elemento.

Crear series

```
>>> import pandas as pd
>>> s = pd.Series(['Matemáticas', 'Historia', 'Economía', 'Programación', 'Inglés'], dtype='string')
>>> print(s)
0     Matemáticas
1        Historia
2        Economía
3    Programación
4          Inglés
dtype: string
```

A partir de una lista

A partir de un diccionario

```
>>> import pandas as pd
>>> s = pd.Series({'Matemáticas': 6.0,  'Economía': 4.5, 'Programación': 8.5})
>>> print(s)
Matemáticas     6.0
Economía        4.5
Programación    8.5
dtype: float64
```

Atributos de una serie

→ s.size: Número de elementos de la serie s

→ s.dtype: Tipo de datos de los elementos de la serie s

```
>>> import pandas as pd
>>> s = pd.Series([1, 2, 2, 3, 3, 3, 4, 4, 4, 4])
>>> s.size
10
>>> s.dtype
dtype('int64')
```

Acceso a los elementos de una serie

Acceso por posición

```
>>> s[1:3]
Economía        4.5
Programación    8.5
dtype: float64
```

Acceso por índice

```
>>> s['Economía']
4.5
>>> s[['Programación', 'Matemáticas']]
Programación    8.5
Matemáticas     6.0
dtype: float64
```

Resumen descriptivo de una serie

- s.count(): Número de elementos que no son nulos ni NaN en la serie s.
- s.sum(): Suma de los datos de la serie s para datos numéricos, o concatenación si son tipo str.
- s.cumsum(): Devuelve serie con la suma acumulada de los datos de la serie s cuando son datos numéricos.

```
>>> import pandas as pd
>>> s = pd.Series([1, 1, 1, 1, 2, 2, 2, 3, 3, 4])
>>> s.count()  # Tamaño muestral
10
>>> s.sum()  # Suma
20
>>> s.cumsum()  # Suma acumulada
0     1
1     2
2     3
3     4
4     6
5     8
6    10
7    13
8    16
9    20
dtype: int64
```

Resumen descriptivo de una serie

- s.value_counts(): Serie con la frecuencia de cada valor de la serie s.
- s.min(): Devuelve el menor de los datos de la serie s.
- s.max(): Devuelve el mayor de los datos de la serie s.

```
>>> import pandas as pd
>>> s = pd.Series([1, 1, 1, 1, 2, 2, 2, 3, 3, 4])
>>> s.value_counts()  # Frecuencias absolutas
1    4
2    3
3    2
4    1
dtype: int64
>>> s.value_counts(normalize=True)  # Frecuencias relativas
1    0.4
2    0.3
3    0.2
4    0.1
dtype: float64
>>> s.min()  # Mínimo
1
>>> s.max()  # Máximo
4
```

Resumen descriptivo de una serie

- s.mean(): Devuelve la media de los datos de la serie s cuando son de un tipo numérico.
- s.std(): Devuelve la desviación típica de los datos de la serie s cuando los datos son numéricos.
- s.describe(): Serie con resumen descriptivo (número de datos, su suma, el mínimo, el máximo, la media, la desviación típica y los cuartiles.

```
>>> import pandas as pd
>>> s = pd.Series([1, 1, 1, 1, 2, 2, 2, 3, 3, 4])
>>> s.mean()  # Media
2.0
>>> s.var()  # Varianza
1.1111111111111112
>>> s.std()  # Desviación típica
1.0540925533894598
>>> s.describe()  # Resume descriptivo
count    10.000000
mean      2.000000
std       1.054093
min       1.000000
25%       1.000000
50%       2.000000
75%       2.750000
max       4.000000
dtype: float64
```

Operaciones con series

Los operadores binarios (+, *, /, etc.) pueden utilizarse con una serie, y devuelve otra serie con el resultado de aplicar la operación a cada elemento de la serie.

Por ejemplo:

```
>>> import pandas as pd
s = pd.Series([1, 2, 3, 4])
>>> s * 2
0    2
1    4
2    6
3    8
dtype: int64
```

```
>>> s % 2
0    1
1    0
2    1
3    0
dtype: int64
```

```
>>> s = pd.Series(['a', 'b', 'c'])
>>> s * 5
0    aaaaa
1    bbbbb
2    ccccc
dtype: object
```

Funciones con series

Es posible aplicar una función a cada elemento de la serie mediante el siguiente método:

s.apply(f): Devuelve una serie con el resultado de aplicar la función f a cada uno de los elementos de la serie s.

```
>>> import pandas as pd
>>> from math import log
>>> s = pd.Series([1, 2, 3, 4])
>>> s.apply(log)
0    0.000000
1    0.693147
2    1.098612
3    1.386294
dtype: float64
>>> s = pd.Series(['a', 'b', 'c'])
>>> s.apply(str.upper)
0    A
1    B
2    C
dtype: object
```

Filtrar una serie

Para filtrar una serie y quedarse con los valores que cumplen una determinada condición se utiliza el siguiente método:

s[condicion]: Devuelve una serie con los elementos de la serie s que se corresponden con el valor True de la lista booleana condición (debe ser una lista de valores booleanos de la misma longitud que la serie

```
>>> import pandas as pd
>>> s = pd.Series({'Matemáticas': 6.0,  'Economía': 4.5, 'Programación': 8.5})
>>> print(s[s > 5])
Matemáticas     6.0
Programación    8.5
dtype: float64
```

Ordenar una serie

- s.sort_values(ascending=booleano): Serie que resulta de ordenar los valores la serie s. Si ascending es True el orden es creciente y si es False decreciente.

- s.sort_index(ascending=booleano): Serie que resulta de ordenar el índice de la serie s. Si ascending es True el orden es creciente y si es False decreciente.

```
>>> import pandas as pd
>>> s = pd.Series({'Matemáticas': 6.0,
                   'Economía': 4.5,
                   'Programación': 8.5})
>>> print(s.sort_values())
Economía        4.5
Matemáticas     6.0
Programación    8.5
dtype: float64
>>> print(s.sort_index(ascending = False))
Programación    8.5
Matemáticas     6.0
Economía        4.5
dtype: float64
```

Eliminar datos desconocidos en una serie

Los datos desconocidos se representan por NaN y los nulos por None.
Para eliminarlos de una serie se utiliza el siguiente método:

→ s.dropna(): Elimina los datos desconocidos o nulos de la serie s.

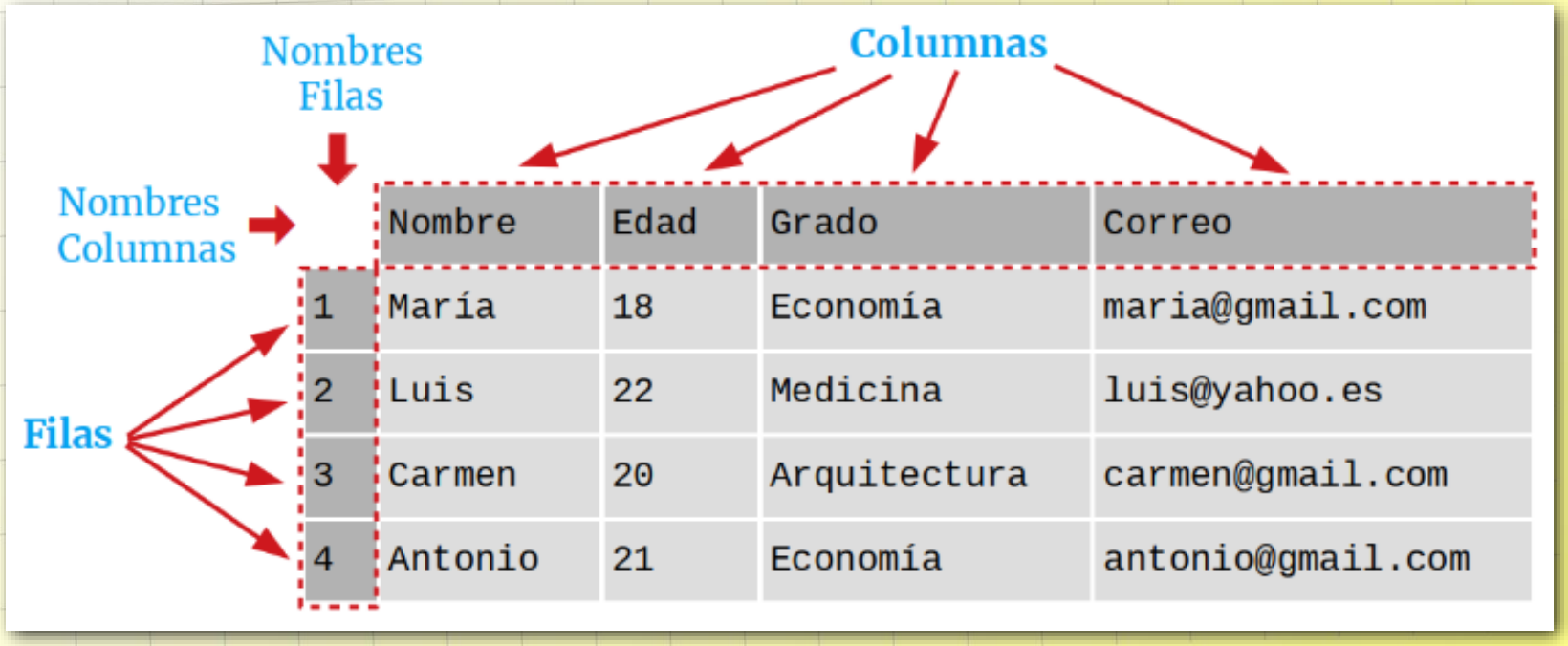
```
>>> s = pd.Series(['a', 'b', None, 'c', np.NaN,  'd'])
>>> s
0       a
1       b
2    None
3       c
4     NaN
5       d
dtype: object
>>> s.dropna()
0    a
1    b
3    c
5    d
dtype: object
```

Dataframe en **Pandas**

¿Qué es?

- Define un conjunto de datos estructurado en forma de tabla donde cada columna es un objeto de tipo Series, y las filas son registros que pueden contener datos de distintos tipos.
- Un DataFrame contiene dos índices, uno para las filas y otro para las columnas.
- Se puede acceder a sus elementos mediante los nombres de las filas y las columnas.

Ejemplo de DataFrame



	Nombre	Edad	Grado	Correo
1	María	18	Economía	maria@gmail.com
2	Luis	22	Medicina	luis@yahoo.es
3	Carmen	20	Arquitectura	carmen@gmail.com
4	Antonio	21	Economía	antonio@gmail.com

Crear DataFrames

A partir de un diccionario

```
>>> import pandas as pd
>>> datos = {'nombre':['María', 'Luis', 'Carmen', 'Antonio'],
... 'edad':[18, 22, 20, 21],
... 'grado':['Economía', 'Medicina', 'Arquitectura', 'Economía'],
... 'correo':['maria@gmail.com', 'luis@yahoo.es', 'carmen@gmail.com', 'antonio@gmail.com']
... }
>>> df = pd.DataFrame(datos)
>>> print(df)
    nombre  edad         grado             correo
0    María    18       Economía    maria@gmail.com
1     Luis    22       Medicina      luis@yahoo.es
2   Carmen    20  Arquitectura   carmen@gmail.com
3  Antonio    21       Economía  antonio@gmail.com
```

Crear DataFrames

```
>>> import pandas as pd
>>> df = pd.DataFrame([['María', 18], ['Luis', 22], ['Carmen', 20]], columns=['Nombre', 'Edad'])
>>> print(df)
   Nombre   Edad
0   María     18
1    Luis     22
2  Carmen     20
```

A partir de una lista

A partir de una lista de diccionarios

```
>>> import pandas as pd
>>> df = pd.DataFrame([{'Nombre':'María', 'Edad':18}, {'Nombre':'Luis', 'Edad':22}, {'Nombre':'Carmen'}])
>>> print(df)
0   María  18.0
1    Luis  22.0
2  Carmen   NaN
```

Crear DataFrames

A partir de un array

```
>>> import pandas as pd
>>> df = pd.DataFrame(np.random.randn(4, 3), columns=['a', 'b', 'c'])
>>> print(df)
          a         b         c
0 -1.408238  0.644706  1.077434
1 -0.279264 -0.249229  1.019137
2 -0.805470 -0.629498  0.935066
3  0.236936 -0.431673 -0.177379
```

Crear DataFrames

A partir de un archivo CSV o Excel

```
>>> df = pd.read_csv('colesterol.csv', sep=';', decimal=',')
>>> print(df.head())
                          nombre  edad sexo   peso  altura  colesterol
0      José Luis Martínez Izquierdo    18    H   85.0    1.79       182.0
1                  Rosa Díaz Díaz    32    M   65.0    1.73       232.0
2           Javier García Sánchez    24    H    NaN    1.81       191.0
3             Carmen López Pinzón    35    M   65.0    1.70       200.0
4            Marisa López Collado    46    M   51.0    1.58       148.0
```

- → read_csv(fichero.csv, sep=separador, header=n, index_col=m, na_values=no-validos, decimal=separador-decimal)
- → read_excel(fichero.xlsx, sheet_name=hoja, header=n, index_col=m, na_values=no-validos, decimal=separador-decimal)

Exportación de DataFrames

→ df.to_csv(fichero.csv, sep=separador, columns=booleano, index=booleano)

→ df.to_excel(fichero.xlsx, sheet_name = hoja, columns=booleano, index=booleano)

En ambos casos:

- Se usa como separador de los datos la cadena separador.
- Si se pasa True al parámetro columns se exporta también la fila con los nombres de columnas.
- Si se pasa True al parámetro index se exporta también la columna con los nombres de las filas.

Atributos de un DataFrame

```
>>> df = pd.read_csv(
'https://raw.githubusercontent.com/asalber/manual-python/master/datos/colesterol.csv')
```

```
>>> df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14 entries, 0 to 13
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   nombre      14 non-null     object
 1   edad        14 non-null     int64
 2   sexo        14 non-null     object
 3   peso        13 non-null     float64
 4   altura      14 non-null     float64
 5   colesterol  13 non-null     float64
dtypes: float64(3), int64(1), object(2)
memory usage: 800.0+ bytes
>>> df.shape
(14, 6)
>>> df.size
84
```

- → df.info(): Información (número de filas, número de columnas, índices, tipo de las columnas y memoria usado).
- → df.shape: Devuelve una tupla con el número de filas y columnas.
- → s.size: Número de elementos.

Atributos de un DataFrame

```
>>> df = pd.read_csv(
'https://raw.githubusercontent.com/asalber/manual-python/master/datos/colesterol.csv')
```

```
>>> df.columns
Index(['nombre', 'edad', 'sexo', 'peso', 'altura', 'colesterol'], dtype='object')
```

- df.columns: Lista con los nombres de las columnas.
- df.index: Lista con los nombres de las filas.
- s.dtype: Serie con los tipos de datos de las columnas.

```
>>> df.index
RangeIndex(start=0, stop=14, step=1)
>>> df.dtypes
nombre         object
edad            int64
sexo           object
peso          float64
altura        float64
colesterol    float64
dtype: object
```

Atributos de un DataFrame

→ df.head(n): Devuelve las n primeras filas.

Ejemplo: df.head(3)
(devuelve las 3 primeras filas)

→ df.tail(n): Devuelve las n últimas filas.

Ejemplo: df.tail(2)
(devuelve las 2 últimas filas)

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
5	25.29	4.71	Male	No	Sun	Dinner	4
6	8.77	2.00	Male	No	Sun	Dinner	2
7	26.88	3.12	Male	No	Sun	Dinner	4
8	15.04	1.96	Male	No	Sun	Dinner	2
9	14.78	3.23	Male	No	Sun	Dinner	2
10	10.27	1.71	Male	No	Sun	Dinner	2
11	35.26	5.00	Female	No	Sun	Dinner	4

Operaciones sobre un DataFrame

Agregar columnas

```
>>> df['diabetes'] = pd.Series([False, False, True, False, True])
>>> print(df)
                  nombre  edad sexo    peso  altura  colesterol diabetes
0 Luis Martínez Izquierdo    18    H    85.0    1.79       182.0    False
1          Rosa Díaz Díaz    32    M    65.0    1.73       232.0    False
2   Javier García Sánchez    24    H   NaN.0    1.81       191.0     True
3     Carmen López Pinzón    35    M    65.0    1.70       200.0    False
4    Marisa López Collado    46    M    51.0    1.58       148.0     True
5       Antonio Ruiz Cruz    68    H    66.0    1.74       249.0      NaN
```

- → df[nombre] = lista:
  Añade a df una nueva columna nombre y los valores de la lista.

- → df[nombre] = serie:
  Añade a df una nueva columna nombre y los valores de la serie.

Operaciones sobre un DataFrame

Operaciones sobre columnas

Los datos de una misma columna de un DataFrame son del mismo tipo, entonces se puede aplicar la misma operación a todos los elementos de la columna.

Por ejemplo

```
>>> import pandas as pd
>>> df = pd.read_csv('colesterol.csv')
>>> print(df['altura']*100)
0     179
1     173
2     181
...
>>> print(df['sexo']=='M')
0     False
1      True
2     False
...
```

Operaciones sobre un DataFrame

Aplicar funciones a columnas

Se puede aplicar funciones a todos los elementos de una columna utilizando el método:

- df[columna].apply(f): Serie con los valores que resulta de aplicar la función f a los elementos de la columna del DataFrame df.

```
>>> import pandas as pd
>>> from math import log
>>> df = pd.read_csv('colesterol.csv')
>>> print(df['altura'].apply(log))
0     0.582216
1     0.548121
2     0.593327
...
```

Por ejemplo

Operaciones sobre un DataFrame

Convertir una columna al tipo datetime

Para convertir cadenas al tipo datetime se utiliza el siguiente método:

→ to_datetime(columna, formato): Serie que resulta de convertir las cadenas de columna en fechas del tipo datetime con el formato.

```
>>> import pandas as pd
>>> df = pd.DataFrame({'Name': ['María', 'Carlos', 'Carmen'], 'Nacimiento':['05-03-2000', '20-05-2001', '10-12-1999']})
>>> print(pd.to_datetime(df.Nacimiento, format = '%d-%m-%Y'))
0   2000-03-05
1   2001-05-20
2   1999-12-10
Name: Nacimiento, dtype: datetime64[ns]
```

Operaciones sobre un DataFrame

Renombrar filas y columnas

→ df.rename(columns=columnas, index=filas):

DataFrame que resulta de renombrar las columnas indicadas en las claves del diccionario columnas con sus valores y las filas indicadas en las claves del diccionario filas con sus valores en el DataFrame df.

```
>>> import pandas as pd
>>> df = pd.read_csv('colesterol.csv')
>>> print(df.rename(columns={'nombre':'nombre y apellidos', 'altura':'estatura'},
                    index={0:1000, 1:1001, 2:1002}))
                    nombre y apellidos  edad sexo    peso  estatura    colesterol
1000      José Luis Martínez Izquierdo    18    H    85.0      1.79         182.0
1001                 Rosa Díaz Díaz       32    M    65.0      1.73         232.0
1002           Javier García Sánchez      24    H     NaN      1.81         191.0
3                Carmen López Pinzón      35    M    65.0      1.70         200.0
4                Marisa López Collado     46    M    51.0      1.58         148.0
...
```

Operaciones sobre un DataFrame

Reindexar

Para reordenar los índices de las filas y las columnas de un DataFrame, o eliminar índices, se utiliza el siguiente método:

```
>>> import pandas as pd
>>> df = pd.read_csv('colesterol.csv')
>>> print(df.reindex(index=[4, 3, 1],
                     columns=['nombre', 'tensión', 'colesterol']))
                  nombre  tensión  colesterol
4   Marisa López Collado      NaN       148.0
3    Carmen López Pinzón      NaN       200.0
1         Rosa Díaz Díaz      NaN       232.0
```

→ df.reindex(index=filas, columns=columnas, fill_value=relleno):
Devuelve el DataFrame que resulta de tomar de df las filas con nombres en la lista filas y las columnas con nombres en la lista columnas.

Operaciones sobre un DataFrame

Eliminar columnas

- del d[nombre]: Elimina la columna 'nombre'.
- df.pop(nombre): Elimina la columna 'nombre' y la devuelve como una serie.

```
print(edad)
0     18
1     32
2     24
...
```

```
>>> import pandas as pd
>>> df = pd.read_csv('colesterol.csv')
>>> edad = df.pop('edad')
>>> print(df)
                    nombre    sexo  peso  altura  colesterol
0    Luis Martínez Izquierdo     H    85.0    1.79       182.0
1             Rosa Díaz Díaz     M    65.0    1.73       232.0
2      Javier García Sánchez     H
NaN    1.81         191.0
...
```

Operaciones sobre un DataFrame

Eliminar filas

- df.drop(filas): Devuelve el DataFrame que resulta de eliminar las filas con los nombres indicados en la lista 'filas'.

```
>>> import pandas as pd
>>> df = pd.read_csv('colesterol.csv')
>>> print(df.drop([1, 3]))
                          nombre  edad sexo   peso  altura  colesterol
0      José Luis Martínez Izquierdo    18    H   85.0    1.79       182.0
2            Javier García Sánchez    24    H    NaN    1.81       191.0
4             Marisa López Collado    46    M   51.0    1.58       148.0
...
```

Operaciones sobre un DataFrame

Eliminar filas con datos desconocidos

- df.dropna(subset=columnas)drop(filas): Elimina las filas que contienen algún dato desconocido o nulo en las columnas de la lista 'columna'. Si no se pasa un argumento al parámetro subset se aplica a todas las columnas.

```
>>> import pandas as pd
>>> df = pd.read_csv('colesterol.csv')
>>> print(df.dropna())
                          nombre  edad sexo   peso  altura  colesterol
0       José Luis Martínez Izquierdo    18    H   85.0    1.79       182.0
1                  Rosa Díaz Díaz    32    M   65.0    1.73       232.0
3              Carmen López Pinzón    35    M   65.0    1.70       200.0
4              Marisa López Collado    46    M   51.0    1.58       148.0
...
```

Operaciones sobre un DataFrame

Ordenar DataFrame

- df.sort_values(columna, ascending=booleano): Ordena las filas de df según los valores de la 'columna'. Si ascending es True el orden es creciente y si es False decreciente.
- df.sort_index(ascending=booleano): Ordena las filas de df según los nombres de las 'filas'.

```
>>> import pandas as pd
>>> df = pd.read_csv('colesterol.csv')
>>> print(df.sort_values('colesterol'))
                          nombre  edad sexo   peso  altura  colesterol
4              Marisa López Collado    46    M   51.0    1.58       148.0
0       José Luis Martínez Izquierdo    18    H   85.0    1.79       182.0
2              Javier García Sánchez    24    H    NaN    1.81       191.0
13             Carolina Rubio Moreno    20    M   61.0    1.77       194.0
...
```

Operaciones sobre un DataFrame

Filtrar filas

→ df.[condicion]: Devuelve un DataFrame con las filas del DataFrame df que se corresponden con el valor True de la lista booleana 'condicion'.

```
>>> import pandas as pd
>>> df = pd.read_csv('colesterol.csv')
>>> print(df[(df['sexo']=='H') & (df['colesterol'] > 260)])
                    nombre  edad sexo    peso  altura    colesterol
6    Antonio Fernández Ocaña    51    H    62.0    1.72         276.0
9    Santiago Reillo Manzano    46    H    75.0    1.85         280.0
```

Acceso sobre el DataFrame

Mediante posiciones

- df.iloc[i, j]: Devuelve el elemento que se encuentra en la fila i y la columna j del DataFrame df.
- df.iloc[filas, columnas]: Devuelve un DataFrame con los elementos de las filas y las columnas.
- df.iloc[i]: Devuelve una serie con los elementos de la fila i del DataFrame df.

```
>>> import pandas as pd
>>> df = pd.read_csv('colesterol.csv')
>>> print(df.iloc[1, 3])
65
>>> print(df.iloc[1, :2])
nombre     Rosa Díaz Díaz
edad                   32
```

Acceso sobre el DataFrame

Mediante nombres

- → df.loc[fila, columna]: Devuelve el elemento que se encuentra en la fila y la columna del DataFrame df.
- → df[columna]: Devuelve una serie con los elementos de la columna del DataFrame df.
- → df.columna: Devuelve una serie con los elementos de la columna de df. Sólo funciona cuando el nombre de la columna no tiene espacios en blanco.

```
>>> import pandas as pd
>>> df = pd.read_csv('colesterol.csv')
>>> print(df.loc[2, 'colesterol'])
191
>>> print(df.loc[:3, ('colesterol','peso')])
     colesterol    peso
1         232.0    65.0
2         191.0     NaN
3         200.0    65.0
>>> print(df['colesterol'])
0     182.0
1     232.0
2     191.0
3     200.0
...
```

Resumen descriptivo del DataFrame

- df.count(): Devuelve una serie con el número de elementos que no son nulos ni NaN en cada columna de df.

```
>>> import pandas as pd
>>> df = pd.read_csv('colesterol.csv')
>>> df.edad.count()
 14
```

- df.sum(): Devuelve una serie con la suma de los datos de las columnas del DataFrame df cuando los datos son de un tipo numérico, o la concatenación de ellos cuando son del tipo cadena str.

- df.cumsum(): Devuelve un DataFrame con la suma acumulada de los datos de las columnas del DataFrame df cuando los datos son de un tipo numérico.

Resumen descriptivo del DataFrame

- df.mean(): Devuelve una serie con las medias de los datos de las columnas numéricas del DataFrame df.
- df.std(): Devuelve una serie con las desviaciones típicas de los datos de las columnas numéricas del DataFrame df.
- df.describe(include = tipo): Devuelve un DataFrame con un resumen estadístico de las columnas del DataFrame df del tipo 'tipo'.

```
>>> import pandas as pd
>>> df = pd.read_csv('colesterol.csv')
>>> print(df.edad.mean())
38.214285714285715
>>> print(df.edad.std())
15.62137870123737
>>> print(df.describe())
            edad        peso     altura  colesterol
count  14.000000   13.000000  14.000000   13.000000
mean   38.214286   70.923077   1.768571  220.230769
std    15.621379   16.126901   0.115016   39.847948
min    18.000000   51.000000   1.580000  148.000000
25%    24.750000   61.000000   1.705000  194.000000
50%    35.000000   65.000000   1.755000  210.000000
75%    49.750000   78.000000   1.840000  249.000000
max    68.000000  109.000000   1.980000  280.000000
```

Reagrupar un DataFrame

Dividir un DataFrame en grupos

- df.groupby(columnas).groups: Devuelve un diccionario cuyas claves son las tuplas que resultan de todas las combinaciones de los valores de las columnas, y valores las listas de los nombres de las filas que contienen esos valores en las correspondientes columnas del DataFrame df.

```
>>> import pandas as pd
>>> df = pd.read_csv('colesterol.csv')
>>> print(df.groupby('sexo').groups)
{'H': Int64Index([0, 2, 5, 6, 8, 9, 11, 12], dtype='int64'),
 'M': Int64Index([1, 3, 4, 7, 10, 13], dtype='int64')}
```

Reagrupar un DataFrame

Aplicar una función de agregación por grupos

➜ df.groupby(columnas).agg(funciones): Devuelve un DataFrame con el resultado de aplicar las funciones de agregación a cada uno de los DataFrames que resultan de dividir el DataFrame según las columnas.

Una función de agregación toma como argumento una lista y devuelve un único valor.
Algunas de las más comunes son: np.min, np.max, np.count_nonzero, np.sum, np.mean, np.std.

```
>>> import pandas as pd
>>> df = pd.read_csv('colesterol.csv')
>>> print(df.groupby('sexo').agg(np.mean))
            edad       peso    altura  colesterol
sexo
H      40.875000  80.714286  1.837500     228.375
M      34.666667  59.500000  1.676667     207.200
```

Reagrupar un DataFrame

Convertir un DataFrame a formato ancho

- df.pivot(index= filas, columns=columna, values=valores): Devuelve el DataFrame que resulta de convertir df de formato largo a formato ancho. Se crean tantas columnas nuevas como valores distintos haya en columna. Los nombres de estas nuevas columnas son los valores de la columna columna mientras que sus valores se toman de la columna valores. Los nombres del índice del nuevo DataFrame se toman de los valores de la columna filas.


HENRY