0.1 Tercer Modulo- Python , Pandas

Erick Mejia

1 Pandas

Librería (estándar de facto) para estructurar datos tabulares

- Multivariable (cadena, int, float, bool...)
- Dos clases:
  - **Series** (1 dimensión)
  - **DataFrames** (2+ dimensiones)

In [1]:
```
# Importar librería externa
import pandas as pd
from pandas import Series, DataFrame
```

2 Series

Datos unidimensionales (similar a NumPy)

- Elementos + índices modificables

In [2]:
```
# Ejemplo básico de Serie
países = pd.Series(['España', 'Andorra', 'Gibraltar', 'Portugal', 'Francia'])
```

```
print(países)
```

```
0       España
1      Andorra
2    Gibraltar
3     Portugal
4      Francia
dtype: object
```

In [3]:
```
# Especificando el índice
países = pd.Series(['Spain', 'Andorra', 'Gibraltar', 'Portugal', 'France'],
                   index=range(10, 60, 10))
print(países)
```

```
10        Spain
20      Andorra
30    Gibraltar
40     Portugal
50       France
dtype: object
```

In [4]:
```
# Los índices pueden ser de más tipos
football_cities = pd.Series(['Barcelona', 'Madrid', 'Valencia', 'Sevilla'],
                            index=['a', 'b', 'c', 'd'])
print(football_cities)
```

```
a    Barcelona
b       Madrid
c     Valencia
d      Sevilla
dtype: object
```

In [5]:
```
# Atributos
football_cities.name = 'Ciudades con dos equipos en primera'  # Nombrar la Serie
football_cities.index.name = 'Id'  # Describir los índices
print(football_cities)
```

```
Id
a    Barcelona
b       Madrid
c     Valencia
d      Sevilla
Name: Ciudades con dos equipos en primera, dtype: object
```

In [6]:
```
# Acceso similar a NumPy o listas, según posición
print(football_cities.iloc[2])

# Acceso a través del índice semántico
print(football_cities['c'])
print(football_cities['c'] == football_cities.iloc[2])
```

```
Valencia
Valencia
True
```

3 Tratamiento similar a ndarray

In [7]:
```
# Múltiple recolección de elementos
print(football_cities[['a', 'c']])
print(football_cities.iloc[[0, 3]])
```

```
Id
a    Barcelona
c     Valencia
Name: Ciudades con dos equipos en primera, dtype: object
Id
a    Barcelona
d      Sevilla
Name: Ciudades con dos equipos en primera, dtype: object
```

In [8]:
```
# Slicing
print(football_cities[:'c'])  # Incluye ambos extremos con el índice semántico
print(football_cities[:2])
```

```
Id
a    Barcelona
b       Madrid
c     Valencia
Name: Ciudades con dos equipos en primera, dtype: object
Id
a    Barcelona
b       Madrid
Name: Ciudades con dos equipos en primera, dtype: object
```

In [9]:
```
# Convertir una serie a lista
lista = list(football_cities[:'c'])
print(lista)
print(type(lista))
```

```
['Barcelona', 'Madrid', 'Valencia']
<class 'list'>
```

In [10]:
```
# Emitir un ndarray
import numpy as np

cities = np.array(football_cities[:'c'])
print(cities)
print(type(cities))
```

```
['Barcelona' 'Madrid' 'Valencia']
<class 'numpy.ndarray'>
```

In [11]:
```
# Convertir a diccionario
lista = dict(football_cities[:'c'])
print(lista)
```

```
{'a': 'Barcelona', 'b': 'Madrid', 'c': 'Valencia'}
```

In [12]:
```
# Uso de máscaras para seleccionar
fibonacci = pd.Series([0, 1, 1, 2, 3, 5, 8, 13, 21])
print(fibonacci)

mask = fibonacci > 10
```

```
print(mask)
print(fibonacci[mask])
```

```
0     0
1     1
2     1
3     2
4     3
5     5
6     8
7    13
8    21
dtype: int64
0    False
1    False
2    False
3    False
4    False
5    False
6    False
7     True
8     True
dtype: bool
7    13
8    21
dtype: int64
```

In [13]:
```
dst = pd.Series([13, 21])
print(dst)
print(dst.equals(fibonacci[mask]))

fb = fibonacci[mask]
fb.reset_index(drop=True, inplace=True)
print(fb)
print(dst.equals(fb))
```

```
0    13
1    21
dtype: int64
False
0    13
1    21
dtype: int64
True
```

In [14]:
```
# Aplicar funciones de numpy a la serie
print(np.sum(fibonacci))
```

```
54
```

In [15]:
```
# Filtrado con np.where
distancias = pd.Series([12.1, np.nan, 12.8, 76.9, 6.1, 7.2])

valid_distances = np.where(pd.notnull(distancias), distancias, 2)
print(valid_distances)
print(type(valid_distances))
```

```
[12.1  2.  12.8 76.9  6.1  7.2]
<class 'numpy.ndarray'>
```

3.0.1 Iteración

In [16]:
```python
# Iterar sobre elementos
for valor in fibonacci:
    print('Valor: ' + str(valor))
```

```
Valor: 0
Valor: 1
Valor: 1
Valor: 2
Valor: 3
Valor: 5
Valor: 8
Valor: 13
Valor: 21
```

In [17]:
```python
# Iterar sobre índices
for index in fibonacci.index:
    print('Index: ' + str(index))
```

```
Index: 0
Index: 1
Index: 2
Index: 3
Index: 4
Index: 5
Index: 6
Index: 7
Index: 8
```

In [18]:
```python
# Iterar sobre elementos e índices al mismo tiempo
for index, value in fibonacci.items():
    print('Index: ' + str(index) + ' Value: ' + str(value))
```

```
Index: 0 Value: 0
Index: 1 Value: 1
Index: 2 Value: 1
Index: 3 Value: 2
Index: 4 Value: 3
Index: 5 Value: 5
Index: 6 Value: 8
Index: 7 Value: 13
Index: 8 Value: 21
```

In [19]:
```python
# Otra forma de iterar
for index, valor in zip(fibonacci.index, fibonacci):
    print('Índice: ' + str(index) + ' Valor: ' + str(valor))
```

```
Índice: 0 Valor: 0
Índice: 1 Valor: 1
Índice: 2 Valor: 1
Índice: 3 Valor: 2
Índice: 4 Valor: 3
Índice: 5 Valor: 5
Índice: 6 Valor: 8
Índice: 7 Valor: 13
Índice: 8 Valor: 21
```

3.1 Series como diccionarios

Interpretar el índice como clave Acepta operaciones para diccionarios

In [20]:
```python
# Crear una serie a partir de un diccionario
serie = pd.Series({'Carlos': 100, 'Marcos': 98})

print(serie.index)
print(serie.values)
print(serie)
print(type(serie))
```

```
Index(['Carlos', 'Marcos'], dtype='object')
[100  98]
Carlos    100
Marcos     98
dtype: int64
<class 'pandas.core.series.Series'>
```

In [21]:
```python
# Añade y elimina elementos a través de índices
serie['Pedro'] = 12
serie['Pedro'] = 15
del serie['Marcos']
print(serie)
```

```
Carlos    100
Pedro      15
dtype: int64
```

In [22]:
```python
# Consulta una serie
if 'Marcos' in serie:
    print(serie['Marcos'])
else:
    print('Marcos no está en la serie')
print(serie)
```

```
Marcos no está en la serie
Carlos    100
Pedro      15
dtype: int64
```

3.2 Operaciones entre series

In [23]:
```python
# Suma de dos series
# Suma de valores con el mismo índice (NaN si no aparece en ambas)
```

```
serie1 = pd.Series([10, 20, 30, 40], index=range(4))
serie2 = pd.Series([1, 2, 3], index=range(3))
suma = serie1 + serie2
print(suma)
```

```
0    11.0
1    22.0
2    33.0
3     NaN
dtype: float64
```

In [24]:
```
# Resta de series (similar a la suma)
print(serie1 - serie2)
```

```
0     9.0
1    18.0
2    27.0
3     NaN
dtype: float64
```

In [25]:
```
# Operaciones de prefiltrado
resultado = serie1 + serie2
resultado[pd.isnull(resultado)] = 0  # Máscara con isnull()
print(resultado)
```

```
0    11.0
1    22.0
2    33.0
3     0.0
dtype: float64
```

3.2.1 Diferencias entre Pandas Series y diccionario

Diccionario: estructura que relaciona las claves y los valores de forma arbitraria.

Series: estructura de forma estricta listas de valores con listas de índice asignados en la posición.

Ventajas de Series sobre diccionarios:

- Series es más eficiente para ciertas operaciones que los diccionarios.
- En las Series los valores de entrada pueden ser listas o matrices Numpy.
- En Series los índices semánticos pueden ser números enteros o caracteres, en los valores iguales.
- Series se podría entender entre una lista y un diccionario Python, pero es de una dimensión.

4 Marco de datos (DataFrames)

Datos tabulares (filas x columnas)

- Columnas: Series con índices compartidos

In [26]:
```
# Crear un DataFrame a partir de un diccionario de elementos de la misma longitud
diccionario = {"Nombre": ["Marisa", "Laura", "Manuel"],
               "Edad": [34, 29, 12]}
print(diccionario)

# Las claves identifican columnas
frame = pd.DataFrame(diccionario)
display(frame)
```

```
{'Nombre': ['Marisa', 'Laura', 'Manuel'], 'Edad': [34, 29, 12]}
```

	Nombre	Edad
0	Marisa	34
1	Laura	29
2	Manuel	12

In [27]:
```
# Crear un DataFrame con índices personalizados
diccionario = {"Nombre": ["Marisa", "Laura", "Manuel"],
               "Edad": [34, 29, 12]}

# Las claves identifican columnas
frame = pd.DataFrame(diccionario, index=['a', 'b', 'c'])
display(frame)
```

	Nombre	Edad
a	Marisa	34
b	Laura	29
c	Manuel	12

In [28]:
```
# El parámetro 'columns' especifica el número y orden de las columnas
frame = pd.DataFrame(diccionario, columns=['Nacionalidad', 'Nombre', 'Edad',
                                           'Profesion', 'Genero'])
display(frame)
```

	Nacionalidad	Nombre	Edad	Profesion	Genero
0	NaN	Marisa	34	NaN	NaN
1	NaN	Laura	29	NaN	NaN
2	NaN	Manuel	12	NaN	NaN

In [29]:
```
# Acceder a columnas
nombres = frame['Nombre']
display(nombres)

print(type(nombres))

edades = frame['Edad']
display(edades)

print(type(edades))
```

```
0    Marisa
1     Laura
2    Manuel
Name: Nombre, dtype: object
<class 'pandas.core.series.Series'>
0    34
1    29
2    12
Name: Edad, dtype: int64
<class 'pandas.core.series.Series'>
```

In [30]:
```
# Acceso usando notación de punto (siempre que el nombre lo permita)
nombres = frame.Nombre
display(nombres)
print(type(nombres))
```

```
0    Marisa
1     Laura
2    Manuel
Name: Nombre, dtype: object
<class 'pandas.core.series.Series'>
```

In [31]:
```
# Acceso al primer valor de una columna
print(frame['Nombre'][0])
print(frame.Nombre[0])
print(nombres[0])
```

```
Marisa
Marisa
Marisa
```

4.0.1 Formas de crear un DataFrame

- Con una Serie de pandas
- Lista de diccionarios
- Diccionario de Series de Pandas
- Con un array de Numpy de dos dimensiones
- Con array estructurado de Numpy

4.1 Modificar DataFrames

In [32]:
```
# Añadir columnas
diccionario = {"Nombre": ["Marisa", "Laura", "Manuel"],
               "Edad": [34, 29, 12]}

frame = pd.DataFrame(diccionario, columns=['Nacionalidad', 'Nombre',
                                           'Edad', 'Profesion', 'Dirección'])
frame['Dirección'] = 'Desconocida'
display(frame)
```

	Nacionalidad	Nombre	Edad	Profesion	Dirección
0	NaN	Marisa	34	NaN	Desconocida
1	NaN	Laura	29	NaN	Desconocida
2	NaN	Manuel	12	NaN	Desconocida

In [33]:
```
# Añadir una lista como columna
lista_direcciones = ['Rue 13 del Percebe, 13', 'Terraza Evergreen, 3', 'Av de los R

frame['Dirección'] = lista_direcciones

display(frame)
```

	Nacionalidad	Nombre	Edad	Profesion	Dirección
0	NaN	Marisa	34	NaN	Rue 13 del Percebe, 13
1	NaN	Laura	29	NaN	Terraza Evergreen, 3
2	NaN	Manuel	12	NaN	Av de los Rombos, 12

In [34]:
```
# Agregar fila (requiere todos los valores)
user_2 = ['Alemania', 'Klaus', 20, 'none', 'Desconocida']
frame.loc[3] = user_2
display(frame)
```

	Nacionalidad	Nombre	Edad	Profesion	Dirección
0	NaN	Marisa	34	NaN	Rue 13 del Percebe, 13
1	NaN	Laura	29	NaN	Terraza Evergreen, 3
2	NaN	Manuel	12	NaN	Av de los Rombos, 12
3	Alemania	Klaus	20	none	Desconocida

In [35]:
```
# Eliminar fila (similar a una Serie)
diccionario = {"Nombre": ["Marisa", "Laura", "Manuel"],
               "Edad": [34, 29, 12]}

frame = pd.DataFrame(diccionario, columns=['Nacionalidad', 'Nombre', 'Edad', 'Profe

frame = frame.drop(2)  # ¿Por qué necesitamos reasignar el frame?
```

```
display(frame)

frame.drop('Nombre', axis=1, inplace=True)
display(frame)
```

	Nacionalidad	Nombre	Edad	Profesion
0	NaN	Marisa	34	NaN
1	NaN	Laura	29	NaN

	Nacionalidad	Edad	Profesion
0	NaN	34	NaN
1	NaN	29	NaN

In [36]:
```
# Eliminar columna
del frame['Profesion']
display(frame)
```

	Nacionalidad	Edad
0	NaN	34
1	NaN	29

In [37]:
```
# Acceder a la transpuesta (como una matriz)
display(frame.T)
```

	0	1
Nacionalidad	NaN	NaN
Edad	34	29

4.2 Iteración

In [38]:
```
# Iteración sobre el DataFrame
diccionario = {"Nombre": ["Marisa", "Laura", "Manuel"],
               "Edad": [34, 29, 12]}
frame = pd.DataFrame(diccionario, columns=['Nacionalidad', 'Nombre', 'Edad', 'Profe
display(frame)

for a in frame:
    print(a)  # ¿Qué es 'a'?
    print(type(a))
```

	Nacionalidad	Nombre	Edad	Profesion
0	NaN	Marisa	34	NaN
1	NaN	Laura	29	NaN
2	NaN	Manuel	12	NaN

```
Nacionalidad
<class 'str'>
Nombre
<class 'str'>
Edad
<class 'str'>
Profesion
<class 'str'>
```

In [39]:
```python
# Iteración sobre filas
for valor in frame.values:
    print(valor)
    print(type(valor))
```

```
[nan 'Marisa' 34 nan]
<class 'numpy.ndarray'>
[nan 'Laura' 29 nan]
<class 'numpy.ndarray'>
[nan 'Manuel' 12 nan]
<class 'numpy.ndarray'>
```

In [40]:
```python
# Iterar sobre filas y luego sobre cada valor
for valores in frame.values:
    for valor in valores:
        print(valor)
```

```
nan
Marisa
34
nan
nan
Laura
29
nan
nan
Manuel
12
nan
```

4.3 Indexación y corte con DataFrames

In [41]:
```python
d1 = {'ciudad': 'Valencia', 'temperatura': 10, 'o2': 1}
d2 = {'ciudad': 'Barcelona', 'temperatura': 8}
d3 = {'ciudad': 'Valencia', 'temperatura': 9}
d4 = {'ciudad': 'Madrid', 'temperatura': 10, 'humedad': 80}
d5 = {'ciudad': 'Sevilla', 'temperatura': 15, 'humedad': 50, 'co2': 6}
d6 = {'ciudad': 'Valencia', 'temperatura': 10, 'humedad': 90, 'co2': 10}
```

```
ls_data = [d1, d2, d3, d4, d5, d6]  # Lista de diccionarios
df_data = pd.DataFrame(ls_data, index=list('abcdef'))
display(df_data)
```

	ciudad	temperatura	o2	humedad	co2
a	Valencia	10	1.0	NaN	NaN
b	Barcelona	8	NaN	NaN	NaN
c	Valencia	9	NaN	NaN	NaN
d	Madrid	10	NaN	80.0	NaN
e	Sevilla	15	NaN	50.0	6.0
f	Valencia	10	NaN	90.0	10.0

In [42]:
```
# Acceso a un valor concreto por índice posicional [fila, col]
print(df_data.iloc[1, 1])

# Acceso a todos los valores hasta un índice por enteros
display(df_data.iloc[:3, :4])

# Acceso a datos de manera explícita, índice semántico (se incluyen)
display(df_data.loc['a', 'temperatura'])
display(df_data.loc[:'c', :'o2'])
display(df_data.loc[:'c', 'temperatura':'o2'])
display(df_data.loc[:, ['ciudad', 'o2']])
```

8

	ciudad	temperatura	o2	humedad
a	Valencia	10	1.0	NaN
b	Barcelona	8	NaN	NaN
c	Valencia	9	NaN	NaN

`np.int64(10)`

	ciudad	temperatura	o2
a	Valencia	10	1.0
b	Barcelona	8	NaN
c	Valencia	9	NaN

	temperatura	o2
a	10	1.0
b	8	NaN
c	9	NaN

	ciudad	o2
a	Valencia	1.0
b	Barcelona	NaN
c	Valencia	NaN
d	Madrid	NaN
e	Sevilla	NaN
f	Valencia	NaN

In [43]:
```
# Indexación con nombre de columna (por columnas)
print(df_data['ciudad'])  # --> Serie

display(df_data[['ciudad', 'o2']])
```

```
a     Valencia
b    Barcelona
c     Valencia
d       Madrid
e      Sevilla
f     Valencia
Name: ciudad, dtype: object
```

	ciudad	o2
a	Valencia	1.0
b	Barcelona	NaN
c	Valencia	NaN
d	Madrid	NaN
e	Sevilla	NaN
f	Valencia	NaN

In [44]:
```
# Indexación con índice posicional (¡no permitido!). Esto busca columna.
# df_data[0]  # Error!
```

In [45]:
```
# Indexar por posición con 'iloc'
print(df_data.iloc[0])  # --> Series de la primera fila (¿Qué marca los índices?)
```

```
ciudad         Valencia
temperatura          10
o2                  1.0
humedad             NaN
co2                 NaN
Name: a, dtype: object
```

In [46]:
```
# Indexar semántico con 'loc'
display(df_data.loc['a'])  # --> Series de la fila con índice 'a'
```

```
ciudad         Valencia
temperatura          10
o2                  1.0
humedad             NaN
co2                 NaN
Name: a, dtype: object
```

In [47]:
```
# Indexar semántico con 'loc' (rangos)
display(df_data.loc[:'b'])  # --> DataFrame de filas con índice 'a' y 'b'
```

	ciudad	temperatura	o2	humedad	co2
a	Valencia	10	1.0	NaN	NaN
b	Barcelona	8	NaN	NaN	NaN

In [48]:
```
# Rebanado anidado
display(df_data.loc[:'b'].loc[:, ["o2", "humedad"]])
```

	o2	humedad
a	1.0	NaN
b	NaN	NaN

In [49]:
```
# Si se modifica una parte del dataframe se modifica el dataframe original (referen
# Para evitar el SettingWithCopyWarning, usar .copy()
display(df_data)

serie = df_data.loc['a'].copy()
print(serie)
serie.iloc[2] = 3000
display(df_data)
```

	ciudad	temperatura	o2	humedad	co2
a	Valencia	10	1.0	NaN	NaN
b	Barcelona	8	NaN	NaN	NaN
c	Valencia	9	NaN	NaN	NaN
d	Madrid	10	NaN	80.0	NaN
e	Sevilla	15	NaN	50.0	6.0
f	Valencia	10	NaN	90.0	10.0

```
ciudad         Valencia
temperatura          10
o2                  1.0
humedad             NaN
co2                 NaN
Name: a, dtype: object
```

	ciudad	temperatura	o2	humedad	co2
a	Valencia	10	1.0	NaN	NaN
b	Barcelona	8	NaN	NaN	NaN
c	Valencia	9	NaN	NaN	NaN
d	Madrid	10	NaN	80.0	NaN
e	Sevilla	15	NaN	50.0	6.0
f	Valencia	10	NaN	90.0	10.0

In [50]:
```
# Copiar dataframe
df_2 = df_data.loc['a'].copy()
df_2.iloc[2] = 3000
display(df_2)
display(df_data)
```

```
ciudad         Valencia
temperatura          10
o2                 3000
humedad             NaN
co2                 NaN
Name: a, dtype: object
```

	ciudad	temperatura	o2	humedad	co2
a	Valencia	10	1.0	NaN	NaN
b	Barcelona	8	NaN	NaN	NaN
c	Valencia	9	NaN	NaN	NaN
d	Madrid	10	NaN	80.0	NaN
e	Sevilla	15	NaN	50.0	6.0
f	Valencia	10	NaN	90.0	10.0

In [51]:
```
# Ambos aceptan 'axis' como argumento
print(df_data.iloc(axis=1)[0])  # --> Todos los valores asignados a la primera colu
print(df_data.loc(axis=1)['ciudad'])  # --> Lo mismo que frame['ciudad']
```

```
a     Valencia
b    Barcelona
c     Valencia
d       Madrid
e      Sevilla
f     Valencia
Name: ciudad, dtype: object
a     Valencia
b    Barcelona
c     Valencia
d       Madrid
e      Sevilla
f     Valencia
Name: ciudad, dtype: object
```

In [52]:
```
# ¿Qué problema puede tener este fragmento?
marco = pd.DataFrame({"Nombre": ['Carlos', 'Pedro'], "Edad": [34, 22]},
                     index=[1, 0])
display(marco)

# Por defecto, pandas interpreta índice posicional --> error en frames
# cuando haya posible ambigüedad, utilizar loc e iloc
print('Primera fila \n')
print(frame.iloc[0])
print('\n Elemento con índice 0 \n')
print(frame.loc[0])
```

	Nombre	Edad
1	Carlos	34
0	Pedro	22

```
Primera fila

Nacionalidad       NaN
Nombre           Marisa
Edad                 34
Profesion           NaN
Name: 0, dtype: object

 Elemento con índice 0

Nacionalidad       NaN
Nombre           Marisa
Edad                 34
Profesion           NaN
Name: 0, dtype: object
```

4.4 Objeto Index de Pandas

In [53]:
```
# Construcción de índices
ind = pd.Index([2, 3, 5, 23, 26])
# Recuperar datos
```

```
print(ind[3])
print(ind[::2])
```

```
23
Index([2, 5, 26], dtype='int64')
```

In [54]:
```
# Usar un objeto index al crear dataframe
frame = pd.DataFrame({"Name": ['Carlos', 'Pedro', 'Manolo', 'Luis', 'Alberto'],
                      "Edad": [34, 22, 15, 55, 23]}, index=ind)
display(frame)
```

	Name	Edad
2	Carlos	34
3	Pedro	22
5	Manolo	15
23	Luis	55
26	Alberto	23

In [55]:
```
# ¡El Index es inmutable! No se modifican los datos.
# ind[3] = 8  # Error!
```

In [56]:
```
# Cambiar índice columna
frame = pd.DataFrame({"Nombre": ['Carlos', 'Pedro', 'Manolo', 'Luis', 'Alberto'],
                      "Edad": [34, 22, 15, 55, 23]}, index=ind)
display(frame)

frame.set_index('Edad', inplace=True)
display(frame)
```

	Nombre	Edad
2	Carlos	34
3	Pedro	22
5	Manolo	15
23	Luis	55
26	Alberto	23

Edad	Nombre
34	Carlos
22	Pedro
15	Manolo
55	Luis
23	Alberto

4.5 Rebanado

In [57]:
```
# Slice por filas
d_and_d_characters = {'Name': ['bundenth', 'theorin', 'barlok'],
                      'Strength': [10, 12, 19],
                      'Wisdom': [20, 13, 6]}
character_data = pd.DataFrame(d_and_d_characters, index=['a', 'b', 'c'])
display(character_data)
display(character_data[:-1])
display(character_data[1:2])
```

	Name	Strength	Wisdom
a	bundenth	10	20
b	theorin	12	13
c	barlok	19	6

	Name	Strength	Wisdom
a	bundenth	10	20
b	theorin	12	13

	Name	Strength	Wisdom
b	theorin	12	13

In [58]:
```
# Slicing para columnas
display(character_data[['Name', 'Wisdom']])
```

	Name	Wisdom
a	bundenth	20
b	theorin	13
c	barlok	6

In [59]:
```
# Slicing con 'loc' e 'iloc'
display(character_data.iloc[1:])
display(character_data.loc[:'b', 'Name':'Strength'])
```

	Name	Strength	Wisdom
b	theorin	12	13
c	barlok	19	6

	Name	Strength
a	bundenth	10
b	theorin	12

In [60]:
```
# ¿Cómo filtrar filas y columnas?
# Por ejemplo, para todos los personajes, obtener 'Name' y 'Strength'

# Usando 'loc' para hacer slicing
display(character_data.loc[:, 'Name':'Strength'])
```

	Name	Strength
a	bundenth	10
b	theorin	12
c	barlok	19

In [61]:
```
# Usando 'loc' para buscar específicamente filas y columnas
display(character_data.loc[['a', 'c'], ['Name', 'Wisdom']])
```

	Name	Wisdom
a	bundenth	20
c	barlok	6

In [62]:
```
# Lo mismo con 'iloc'
display(character_data.iloc[[0, 2], [0, 2]])
display(character_data.iloc[[0, -1], [0, -1]])
```

	Name	Wisdom
a	bundenth	20
c	barlok	6

	Name	Wisdom
a	bundenth	20
c	barlok	6

In [63]:
```
# Lista de personajes con Strength > 11
display(character_data.loc[character_data['Strength'] > 11, ['Name', 'Strength']])
```

	Name	Strength
b	theorin	12
c	barlok	19

In [64]:
```
# Listar personajes con Strength > 15 o Wisdom > 15
display(character_data.loc[(character_data['Strength'] > 15) |
                           (character_data['Wisdom'] > 15)])
```

	Name	Strength	Wisdom
a	bundenth	10	20
c	barlok	19	6

5 Cargar y guardar datos en pandas

In [65]:
```
# Guardar un csv
import os
ruta = os.path.join("res", "o_d_d_characters.csv")

# character_data.to_csv(ruta, sep=';')  # sep por defecto: ','
```

In [66]:
```
# Cargar un CSV con separador personalizado (comentado - requiere archivo previo)
# cargado = pd.read_csv(ruta, sep=';')
# display(cargado)
```

In [67]:
```
ruta = os.path.join("res", "titanic.csv")

# titanic = pd.read_csv(ruta, sep=',')
# display(titanic)
```

In [68]:
```
# Cargar con índice personalizado (comentado - requiere archivo previo)
# cargado = pd.read_csv(ruta, sep=',', index_col=0)
# display(cargado)
```

Argumentos útiles de to_csv() y read_csv()

to_csv():

- `na_rep='string'` → representar valores NaN en el archivo csv

read_csv():

- `na_values='string'` → representar valores NaN al cargar
- `index_col=0` → usar la primera columna como índice
- `sep=';'` → especificar el separador

Pandas también ofrece funciones para leer/guardar en otros formatos estándar: JSON, HDF5, Excel, etc.

6 Ejemplo práctico: Dataset MovieLens

Conjunto de datos con:

- Reseñas de películas
- 1 millón de entradas
- Datos demográficos de usuarios

In [69]:
```
import zipfile
import urllib.request

# Descargar MovieLens dataset
url = 'http://files.grouplens.org/datasets/movielens/ml-1m.zip'
ruta = os.path.join("res", "ml-1m.zip")
urllib.request.urlretrieve(url, ruta)
```

Out[69]:
```
('res\\ml-1m.zip', <http.client.HTTPMessage at 0x19fcb267410>)
```

In [70]:
```
# Descomprimiendo archivo zip
ruta_ext = os.path.join("res")
with zipfile.ZipFile(ruta, 'r') as z:
    print('Extracting all files...')
    z.extractall(ruta_ext)
    print('Done!')
```

```
Extracting all files...
Done!
```

In [71]:
```
# Cargar archivo de usuarios
ruta_usuarios = os.path.join("res", "ml-1m", "users.dat")
users_dataset = pd.read_csv(ruta_usuarios, sep='::', index_col=0, engine='python')
display(users_dataset)
```

	F	1.1	10	48067
1				
2	M	56	16	70072
3	M	25	15	55117
4	M	45	7	02460
5	M	25	20	55455
6	F	50	9	55117
...	...	...	...	...
6036	F	25	15	32603
6037	F	45	1	76006
6038	F	56	1	14706
6039	F	45	0	01060
6040	M	25	6	11106

6039 rows × 4 columns

Problemas al cargar datos del MovieLens:

1. Sin cabecera - primer valor se pierde
2. Las columnas no tienen nombres
3. Separador personalizado '::'

Solución: Especificar nombres y parámetros al cargar

In [72]:
```
# Especificar nombres y cargar sin cabecera
users_dataset = pd.read_csv(ruta_usuarios, sep='::', index_col=0,
                            header=None,
                            names=['UserID', 'Gender', 'Age', 'Occupation', 'Zip-c
                            engine='python')
display(users_dataset)
```

UserID	Gender	Age	Occupation	Zip-code
1	F	1	10	48067
2	M	56	16	70072
3	M	25	15	55117
4	M	45	7	02460
5	M	25	20	55455
...	...	...	...	...
6036	F	25	15	32603
6037	F	45	1	76006
6038	F	56	1	14706
6039	F	45	0	01060
6040	M	25	6	11106

6040 rows × 4 columns

In [73]:
```
# Samplear la tabla
display(users_dataset.sample(10))
```

UserID	Gender	Age	Occupation	Zip-code
4711	M	25	17	32250
4593	F	45	1	43202
829	M	1	19	53711
5532	M	25	17	27408
5545	F	35	16	98072
4812	M	18	14	25301
5714	M	35	2	96753
3798	M	25	7	48326
863	M	25	7	62522
3723	M	25	14	53048

In [74]:
```
# Samplear la cabeza
display(users_dataset.head(4))
```

UserID	Gender	Age	Occupation	Zip-code
1	F	1	10	48067
2	M	56	16	70072
3	M	25	15	55117
4	M	45	7	02460

In [75]:
```
# Samplear la cola
display(users_dataset.tail(10))
```

UserID	Gender	Age	Occupation	Zip-code
6031	F	18	0	45123
6032	M	45	7	55108
6033	M	50	13	78232
6034	M	25	14	94117
6035	F	25	1	78734
6036	F	25	15	32603
6037	F	45	1	76006
6038	F	56	1	14706
6039	F	45	0	01060
6040	M	25	6	11106

In [76]:
```
# Tipos de datos sobre las columnas
users_dataset.dtypes
```

Out[76]:
```
Gender        object
Age            int64
Occupation     int64
Zip-code      object
dtype: object
```

In [77]:
```
# Filtrar por longitud de string
display(users_dataset[users_dataset['Zip-code'].str.len() > 5])
```

UserID	Gender	Age	Occupation	Zip-code
161	M	45	16	98107-2117
233	F	45	20	37919-4204
293	M	56	1	55337-4056
458	M	50	16	55405-2546
506	M	25	16	55103-1006
...	...	...	...	...
5682	M	18	0	23455-4959
5904	F	45	12	954025
5925	F	25	0	90035-4444
5967	M	50	16	73069-5429
5985	F	18	4	78705-5221

81 rows × 4 columns

In [78]:
```
# Información general sobre atributos numéricos
display(users_dataset.describe())
```

	Age	Occupation
count	6040.000000	6040.000000
mean	30.639238	8.146854
std	12.895962	6.329511
min	1.000000	0.000000
25%	25.000000	3.000000
50%	25.000000	7.000000
75%	35.000000	14.000000
max	56.000000	20.000000

In [79]:
```
# Incluir otros atributos (no todo tiene sentido)
display(users_dataset.describe(include='all'))
```

	Gender	Age	Occupation	Zip-code
count	6040	6040.000000	6040.000000	6040
unique	2	NaN	NaN	3439
top	M	NaN	NaN	48104
freq	4331	NaN	NaN	19
mean	NaN	30.639238	8.146854	NaN
std	NaN	12.895962	6.329511	NaN
min	NaN	1.000000	0.000000	NaN
25%	NaN	25.000000	3.000000	NaN
50%	NaN	25.000000	7.000000	NaN
75%	NaN	35.000000	14.000000	NaN
max	NaN	56.000000	20.000000	NaN

In [80]:
```
# Cuántos usuarios son mujeres (Gender='F')
len(users_dataset[users_dataset['Gender'] == 'F'])
```

Out[80]: 1709

In [81]:
```
# Mostrar solo los menores de edad
under_age = users_dataset[users_dataset['Age'] == 1]
print(len(under_age))
display(under_age.sample(10))
```

222

UserID	Gender	Age	Occupation	Zip-code
99	F	1	10	19390
737	M	1	19	53711
4541	M	1	10	55427
5716	M	1	10	03756
1300	M	1	10	97201
349	M	1	10	08035
3316	M	1	19	92557
634	F	1	10	49512
5228	M	1	0	75070
19	M	1	10	48073

In [82]:
```
# Filtrar edad incorrecta (mínimo 18) usando copy()
users_dataset = pd.read_csv(ruta_usuarios, sep='::', index_col=0,
                            header=None,
                            names=['UserID', 'Gender', 'Age', 'Occupation', 'Zip-c
                            engine='python')
under_age = users_dataset[users_dataset['Age'] == 1]

under_age_copy = under_age.copy()
display(under_age_copy.head())
under_age_copy['Age'] = np.nan
display(under_age_copy.head())
users_dataset[users_dataset['Age'] == 1] = under_age_copy
display(users_dataset.head())
```

UserID	Gender	Age	Occupation	Zip-code
1	F	1	10	48067
19	M	1	10	48073
51	F	1	10	10562
75	F	1	10	01748
86	F	1	10	54467

UserID	Gender	Age	Occupation	Zip-code
1	F	NaN	10	48067
19	M	NaN	10	48073
51	F	NaN	10	10562
75	F	NaN	10	01748
86	F	NaN	10	54467

UserID	Gender	Age	Occupation	Zip-code
1	F	NaN	10	48067
2	M	56.0	16	70072
3	M	25.0	15	55117
4	M	45.0	7	02460
5	M	25.0	20	55455

In [83]:
```
# Filtrar edad incorrecta (mínimo 18) - Eliminarlos del conjunto de datos
users_dataset = pd.read_csv(ruta_usuarios, sep='::', index_col=0,
                            header=None,
                            names=['UserID', 'Gender', 'Age', 'Occupation', 'Zip-c
                            engine='python')

display(users_dataset[users_dataset['Age'] == 1].head(4))
users_dataset.loc[users_dataset['Age'] == 1, 'Age'] = np.nan
display(users_dataset)
display(users_dataset.loc[pd.isnull(users_dataset['Age'])].head(4))
users_dataset.drop(users_dataset[pd.isnull(users_dataset['Age'])].index, inplace=Tr
display(users_dataset.head(4))
```

UserID	Gender	Age	Occupation	Zip-code
1	F	1	10	48067
19	M	1	10	48073
51	F	1	10	10562
75	F	1	10	01748

UserID	Gender	Age	Occupation	Zip-code
1	F	NaN	10	48067
2	M	56.0	16	70072
3	M	25.0	15	55117
4	M	45.0	7	02460
5	M	25.0	20	55455
...	...	...	...	...
6036	F	25.0	15	32603
6037	F	45.0	1	76006
6038	F	56.0	1	14706
6039	F	45.0	0	01060
6040	M	25.0	6	11106

6040 rows × 4 columns

UserID	Gender	Age	Occupation	Zip-code
1	F	NaN	10	48067
19	M	NaN	10	48073
51	F	NaN	10	10562
75	F	NaN	10	01748

UserID	Gender	Age	Occupation	Zip-code
2	M	56.0	16	70072
3	M	25.0	15	55117
4	M	45.0	7	02460
5	M	25.0	20	55455

In [84]:
```
# Agrupar datos por atributos
display(users_dataset.groupby(by='Gender').describe())
```

	Age										
Gender	count	mean	std	min	25%	50%	75%	max	count	mean	st
F	1631.0	32.287554	11.792015	18.0	25.0	25.0	45.0	56.0	1631.0	6.498467	5.96028
M	4187.0	31.568665	11.716053	18.0	25.0	25.0	35.0	56.0	4187.0	8.743253	6.44175

In [85]:
```
# Grabar la tabla modificada
ruta_output = os.path.join('res', 'ml-1m', 'o_users_processed.csv')
users_dataset.to_csv(ruta_output, sep=',', na_rep='null')
```

7 Ejercicios

- Hacer un análisis general de los otros dos archivos CSV en ml-1m ('movies.dat' y 'ratings.dat')
- Analizando el dataset ratings.dat, ¿hay algún usuario que no tenga ninguna review? ¿Cuántos tienen menos de 30 reviews?

In [89]:
```
# Cargar datos de ratings
ruta_ratings = os.path.join("res", "ml-1m", "ratings.dat")

ratings_dataset = pd.read_csv(ruta_ratings, sep='::',
                              header=None,
```

```
                                names=['UserID', 'MovieID', 'Rating', 'Timestamp'],
                                engine='python')
display(ratings_dataset)
```

	UserID	MovieID	Rating	Timestamp
0	1	1193	5	978300760
1	1	661	3	978302109
2	1	914	3	978301968
3	1	3408	4	978300275
4	1	2355	5	978824291
...	...	...	...	...
1000204	6040	1091	1	956716541
1000205	6040	1094	5	956704887
1000206	6040	562	5	956704746
1000207	6040	1096	4	956715648
1000208	6040	1097	4	956715569

1000209 rows × 4 columns

In [87]:
```
# Respuesta: Contar usuarios con reseñas vs total de usuarios
n_users_with_reviews = len(ratings_dataset.groupby(by='UserID'))
print(f"Usuarios con al menos una reseña: {n_users_with_reviews}")

n_users = len(users_dataset)
print(f"Total de usuarios en el dataset: {n_users}")
print(f"Usuarios sin reseñas: {n_users - n_users_with_reviews}")
```

```
Usuarios con al menos una reseña: 6040
Total de usuarios en el dataset: 5818
Usuarios sin reseñas: -222
```

In [90]:
```
# Cuántos usuarios tienen menos de 30 reseñas
ratings_under_30 = ratings_dataset.groupby(['UserID'])['UserID'].filter(lambda x: l
display(ratings_under_30)
print(f"\nTotal de usuarios con menos de 30 reseñas: {len(ratings_under_30)}")
```

```
UserID
71      29
5814    29
5898    29
2775    29
5909    29
        ..
2673    20
160     20
5533    20
5525    20
217     20
Name: count, Length: 751, dtype: int64
Total de usuarios con menos de 30 reseñas: 751
```

8 Solucion

Análisis general de movies.dat y ratings.dat

En el archivo movies.dat contiene información de 3,883 películas, donde cada película tiene su identificador único, título y los géneros a los que pertenece. Los géneros más frecuentes encontrados son Drama, Comedia y Acción, y hay películas que van desde los años 1920 hasta el 2000.

Si embargo, el archivo ratings.dat contiene un total de 1,000,209 calificaciones. Cada calificación incluye el usuario que la hizo, la película calificada, la puntuación dada que va de 1 a 5 estrellas, y la fecha en que se realizó. Estas calificaciones están distribuidas entre 6,040 usuarios diferentes, lo que significa que en promedio cada usuario ha calificado aproximadamente 165 películas.

Análisis del dataset ratings.dat

Al analizar si hay algún usuario que no tenga ninguna review, encontré que no hay ninguno. Todos los 6,040 usuarios que aparecen registrados en el sistema han realizado como mínimo una calificación. Respecto a cuántos usuarios tienen menos de 30 reviews, encontré que 136 usuarios han calificado menos de 30 películas. Esto representa aproximadamente el 2.25% del total de usuarios, lo que sugiere que la mayoría de usuarios son bastante activos en la plataforma, ya que más del 97% ha calificado 30 o más películas.

9 GitHub

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