Documentación librería Pandas

Básicos de Pandas

Pandas es una librería fundamental en Python para la manipulación y análisis de datos. Su estructura principal es el **DataFrame**, una tabla de datos con etiquetas en las filas e identificadores en las columnas. Permite visualizar, explorar y describir datos de forma muy eficiente.

```
1. import pandas as pd
 2. import numpy as np
 3.
 4. dataframe = pd.DataFrame([[1,2,3],[4,5,6],[7,8,9],[10,11,12]], columns=["A","B","C"],
index=["x","y","z","zz"])
 5.
 6. display(dataframe)
 7.
 8. dataframe.head(2)
 9.
10. dataframe.tail(3)
11.
12. dataframe.columns
13.
14. dataframe.index.tolist()
15.
16. dataframe.info()
17.
18. dataframe.describe()
19.
20. dataframe.nunique()
21.
22. dataframe['A'].unique()
23.
24. dataframe.shape
25.
26. dataframe.size
27.
```

Explicación:

- pd.DataFrame(...): crea un DataFrame a partir de una lista de listas. Se definen nombres de columnas (columns) y nombres de filas (index).
- display(dataframe): muestra el DataFrame completo (especialmente útil en Jupyter).
- dataframe.head(n): muestra las **primeras n filas** del DataFrame.
- dataframe.tail(n): muestra las **últimas n filas**.
- dataframe.columns: lista las columnas del DataFrame.
- dataframe.index.tolist(): convierte los índices (nombres de filas) en una lista.
- dataframe.info(): muestra un resumen general de la estructura: cantidad de datos, tipos, columnas, etc.
- dataframe.describe(): genera estadísticas descriptivas (media, desviación estándar, min, max...) para las columnas numéricas.
- dataframe.nunique(): muestra cuántos **valores únicos** hay por columna.

- dataframe['A'].unique(): devuelve los valores únicos de la columna "A".
- dataframe.shape: muestra una tupla con (n_filas, n_columnas).
- dataframe.size: devuelve el número total de elementos (filas × columnas).

Estas operaciones son esenciales para comenzar cualquier análisis exploratorio de datos y entender rápidamente la estructura de un conjunto de datos.

Carga de ficheros a DataFrames con Pandas

Pandas permite cargar datos desde múltiples formatos de archivo directamente en **DataFrames**, lo que facilita su análisis y manipulación. Los formatos más comunes son CSV, Excel y Parquet.

```
1. fichero_coffe = pd.read_csv("coffee.csv")
 2. bios = pd.read_csv('bios.csv')
 3. display(fichero_coffe)
 4.
 5. fichero_olimpics = pd.read_excel("olympics-data.xlsx", sheet_name="results")
 6. display(fichero_olimpics)
 7.
 8. fichero_parquet = pd.read_parquet("results.parquet")
 9. display(fichero_parquet)
10.
```

Explicación:

- pd.read_csv("archivo.csv"): carga un archivo CSV (valores separados por comas) en un DataFrame.
- pd.read_excel("archivo.xlsx", sheet_name="nombre_hoja"): carga un archivo de Excel, especificando el nombre de la hoja que se quiere leer.
- pd.read_parquet("archivo.parquet"): carga un archivo en formato **Parquet**, un formato binario eficiente para almacenamiento columnar, común en big data.
- display(...): muestra visualmente el contenido del DataFrame (en notebooks o entornos interactivos).

Estas funciones son fundamentales para importar datos reales desde fuentes externas y comenzar el análisis de manera inmediata.

Formas de indexar en Pandas

Pandas ofrece múltiples formas de **acceder, seleccionar y modificar** datos dentro de un DataFrame, ya sea mediante índices numéricos (iloc), etiquetas (loc), o directamente por nombre de columna.

```
1. fichero_coffe.sample(3)
```

- Devuelve **3 filas aleatorias** del DataFrame.

```
1. fichero_coffe.loc[2]
```

- Accede a la **fila con índice 2** usando el índice por etiqueta (loc).

```
1. fichero_coffe.loc[[0,1,5]]
2.
```

- Accede a múltiples filas específicas mediante una **lista de etiquetas**.

```
1. fichero_coffe.loc[5:9, ["Day", "Units Sold"]]
2.
```

- Accede a un **rango de filas** del índice 5 al 9 y solo a las columnas "Day" y "Units Sold".

```
1. dataframe.loc['x']
2.
```

- Accede a una fila con índice personalizado (etiqueta 'x'), como en el DataFrame creado anteriormente.

```
1. fichero_coffe.iloc[5]
2.
```

- Accede a la **fila número 5** (posición numérica) usando iloc.

```
1. fichero_coffe.iloc[:, [0,2]]
```

- Accede a **todas las filas** y a las **columnas 0 y 2** por posición.

```
1. fichero_coffe.index = fichero_coffe["Day"]
2.
```

- Cambia el **índice del DataFrame** para que sea la columna "Day".

```
1. fichero_coffe.loc["Monday":"Wednesday"]
2.
```

- Selecciona un **rango de filas** entre "Monday" y "Wednesday" (cuando el índice es la columna "Day").

```
1. fichero_coffe.loc["Monday":"Wednesday", "Units Sold"] = 10
2.
```

- **Modifica valores** en la columna "Units Sold" solo para los días "Monday" a "Wednesday".

```
1. fichero_coffe.Day
2. fichero_coffe["Units Sold"]
3.
```

- Acceso directo a columnas por **nombre**. Ambas formas son equivalentes, aunque la notación con corchetes es más flexible y segura.

```
1. fichero_coffe.sort_values(["Units Sold"], ascending=False)
2.
```

- Ordena el DataFrame de forma **descendente** según los valores de la columna "Units Sold".

Estas técnicas permiten manipular y consultar los datos de forma precisa, facilitando análisis avanzados y modificaciones sobre subconjuntos de información.

Iterar sobre un DataFrame en Pandas

Aunque en Pandas se recomienda trabajar con operaciones vectorizadas por eficiencia, también es posible **recorrer fila por fila** un DataFrame usando iterrows(). Esto es útil cuando se necesita trabajar con cada fila de forma individual.

```
1. for index, row in fichero_coffe.iterrows():
2.   print(f"Café con índice: {int(index)} tiene los valores de: {row}")
3.   print("\n\n")
```

```
4.
```

Explicación:

- fichero_coffe.iterrows(): devuelve un generador que permite **iterar por filas** del DataFrame.
- En cada iteración, se obtienen dos valores:
    - index: el índice de la fila (puede ser numérico o una etiqueta personalizada).
    - row: un objeto tipo Series que contiene los **datos de la fila completa**.
- Se usa int(index) para asegurarse de que el índice se imprima como número entero (si corresponde).

Este método es adecuado para inspecciones, transformaciones personalizadas o creación de estructuras basadas en condiciones por fila.

Operaciones con los datos en Pandas

Pandas permite aplicar filtros, crear nuevas columnas, unir distintos DataFrames y realizar tareas de limpieza de datos de manera muy eficiente y expresiva.

Filtros y condiciones

```
 1. bios.head()
 2.
 3. bios.loc[bios["height_cm"] > 215]
 4.
 5. bios.loc[bios["height_cm"] > 215, ["name", "height_cm"]]
 6.
 7. bios[bios["height_cm"] > 215][["name", "height_cm"]]
 8.
 9. bios[(bios["height_cm"] > 215) & (bios["born_country"] == "USA")]
10.
```

- Se aplican condiciones para **filtrar filas** del DataFrame.
- Se pueden usar múltiples condiciones combinadas con & (AND) o | (OR).
- Se accede a columnas específicas después de filtrar.

Filtrar usando strings

```
1. bios[bios['name'].str.contains("keith", case=False)]
2.
3. bios.query('born_country == "USA" and born_city == "Seattle"')
4.
```

- .str.contains(): permite **buscar texto** dentro de una columna tipo string.
- .query(): permite aplicar condiciones como si se escribieran en SQL.

Añadir y quitar columnas

```
1. fichero_coffe.head()
 2.
 3. fichero_coffe["price"] = 4.56
 4.
```

```
 5. fichero_coffe['price_new'] = np.where(fichero_coffe['Coffee Type'] == 'Espresso', 3.15,
3.50)
 6.
 7. fichero_coffe.drop(columns=['price'], inplace=True)
 8.
 9. fichero_coffe = fichero_coffe[['Day', 'Coffee Type', 'Units Sold', 'price_new']]
10.
11. fichero_coffe['beneficio'] = fichero_coffe['price_new'] * fichero_coffe['Units Sold']
12.
13. fichero_coffe.rename(columns={'price_new': 'precio'}, inplace=True)
15. fichero_coffe_copy = fichero_coffe.copy()
```

- Se crean nuevas columnas usando constantes o condiciones (np.where).
- Se eliminan columnas con .drop().
- Se renombra una columna con .rename().
- Se usa .copy() para crear una copia del DataFrame.

Unión de información de distintos DataFrames

```
1. bios = pd.read_csv('bios.csv')
 2. noc = pd.read_csv('noc_regions.csv')
 3.
 4. bios_ampliada = pd.merge(bios, noc, left_on="born_country", right_on="NOC", how="left")
 5.
 6. bios_ampliada.rename(columns={'region': 'pais_nacimiento'}, inplace=True)
 7.
 8. francia = bios[bios['born_country'] == "FRA"]
 9. inglaterra = bios[bios['born_country'] == "USA"]
10.
11. francia_inglaterra = pd.concat([inglaterra, francia])
12.
```

- pd.merge(): une dos DataFrames por columnas comunes.
- how='left': conserva todos los registros de la izquierda, y añade datos coincidentes de la derecha.
- pd.concat([...]): concatena dos DataFrames verticalmente (uno debajo del otro).

Limpieza de valores nulos

```
 1. dataframe = pd.DataFrame({'A': [1,2,None,4], 'B':[None,2,3,None], 'C': [1,2,3,4]})
 2. dataframe = dataframe.interpolate()
 4.
 5. dataframe = dataframe.drop_duplicates()
 6.
 7. print(dataframe.isnull().sum())
 8.
 9. print(dataframe.notnull())
10.
11. dataframe_no_nulos = dataframe.dropna()
12.
```

- .interpolate(): **rellena los valores nulos** mediante interpolación.
- .drop_duplicates(): elimina **filas duplicadas**.
- .isnull(): devuelve una máscara booleana para detectar NaN.
- .notnull(): devuelve el opuesto de .isnull().

- .dropna(): elimina **filas con valores nulos**.

Estas operaciones son fundamentales en cualquier flujo de trabajo de análisis y transformación de datos con Pandas.