# **Pandas para análisis de datos**

Empecemos con la creación de un DataFrame a partir de un diccionario. Un DataFrame es una estructura de datos tabular, similar a una hoja de cálculo en Excel. Podemos pensar en un diccionario como un conjunto de pares clave-valor, donde cada clave tiene un valor asociado. En pandas, podemos convertir un diccionario en un DataFrame utilizando la función "DataFrame", que toma el diccionario como argumento.

En el siguiente ejemplo, se crea un diccionario con tres claves: "nombre", "edad" y "sexo". Cada clave tiene una lista como valor, y cada lista contiene valores correspondientes a cada persona en nuestros datos. Luego, convertimos este diccionario en un DataFrame usando la función "DataFrame" de pandas. Finalmente, imprimimos el DataFrame para visualizarlo en la consola.

| import pandas as pd diccionario = {'nombre': ['Ana', 'Pedro', 'María', 'Juan'],  'edad': [25, 34, 18, 42],  'sexo': ['F', 'M', 'F', 'M']} df = pd.DataFrame(diccionario) print(df) |
| --- |
| nombre edad sexo 0 Ana 25 F 1 Pedro 34 M 2 María 18 F 3 Juan 42 M |

## Métodos útiles de un DataFrame Pandas

Ahora, pasemos a los métodos útiles de un DataFrame Pandas. Pandas proporciona una amplia variedad de métodos para manipular y analizar datos. Algunos métodos comunes son :.

### .head(), .tail() y .sample()

Son métodos de Pandas que se utilizan para ver algunas filas. Por ejemplo, si tenemos un DataFrame llamado *df*, y queremos ver las primeras 2 filas

| df.head(2) |
| --- |
| nombre edad sexo 0 Ana 25 F 1 Pedro 34 M |

Si queremos ver las últimas 2 filas

| df.tail(2) |
| --- |
| nombre edad sexo 2 María 18 F 3 Juan 42 M |

Y si queremos ver 2 filas o registros aleatorios

| df.sample(2) |
| --- |
| nombre edad sexo 3 Juan 42 M 1 Pedro 34 M |

### .info()

Es un método de Pandas que nos proporciona información sobre nuestro DataFrame, como el número de filas y columnas, los tipos de datos de las columnas y si hay valores faltantes. Por ejemplo, si queremos obtener la información sobre un DataFrame df, podemos usar

| df.info() |
| --- |
| <class 'pandas.core.frame.DataFrame'> RangeIndex: 4 entries, 0 to 3 Data columns (total 3 columns):  # Column Non-Null Count Dtype  --- ------ -------------- -----   0 nombre 4 non-null object  1 edad 4 non-null int64   2 sexo 4 non-null object dtypes: int64(1), object(2) memory usage: 224.0+ bytes |

### .describe()

Es un método de Pandas que nos proporciona estadísticas descriptivas para las columnas numéricas de nuestro DataFrame, como la media, la mediana, la desviación estándar y los valores mínimo y máximo. Por ejemplo, si queremos obtener estadísticas descriptivas de las columnas de un DataFrame df con columnas numéricas, podemos usar

| df.describe() |
| --- |
| count 4.000000 mean 29.750000 std 10.468206 min 18.000000 25% 23.250000 50% 29.500000 75% 36.000000 max 42.000000 |

### .value\_counts()

Es un método de Pandas que nos permite contar la frecuencia de valores únicos en una columna determinada. Por ejemplo, si tenemos un DataFrame df con una columna "sexo", podemos usar df['sexo'].value\_counts() para obtener la frecuencia de los valores únicos en la columna "sexo".

| df['sexo'].value\_counts() |
| --- |
| F 2 M 2 Name: sexo, dtype: int64 |

### .pivot\_table()

Es un método de Pandas que nos permite crear una tabla dinámica a partir de nuestro DataFrame, donde podemos agrupar los datos por una o varias columnas y calcular las estadísticas en función de una o varias columnas numéricas. Por ejemplo, si tenemos un DataFrame df con columnas "sexo", "edad" y "salario", podemos usar

| df.pivot\_table(index='sexo', values=['edad'], aggfunc='mean') |
| --- |
| edad sexo   F 21.5  M 38.0 |

para crear una tabla dinámica que muestre la media de la edad y el salario para cada valor único en la columna "sexo".

### Indexación y re-indexación

Otra operación fundamental en la manipulación de DataFrames es la selección de datos a través de la indexación y re-indexación. La indexación se refiere a la selección de una o varias columnas de un DataFrame y la re-indexación se refiere a la selección de filas o columnas por etiqueta o posición.

La indexación en Pandas es una forma de seleccionar un subconjunto de filas y columnas de un DataFrame. Los índices se utilizan para identificar y acceder a los datos en un DataFrame. En un DataFrame, el índice puede ser numérico o alfanumérico y se utiliza para etiquetar las filas de la tabla. Por defecto, los DataFrames de Pandas tienen un índice numérico que comienza en 0 y se incrementa en 1 para cada fila. La indexación se realiza mediante la notación de corchetes []. Para seleccionar una sola columna, se puede hacer referencia a ella por su nombre, por ejemplo:

| df['nombre'] |
| --- |
| 0 Ana 1 Pedro 2 María 3 Juan Name: nombre, dtype: object |

Para seleccionar varias columnas, se deben pasar los nombres de las columnas como una lista, por ejemplo:

| df[['nombre','edad']] |
| --- |
| nombre edad 0 Ana 25 1 Pedro 34 2 María 18 3 Juan 42 |

La re-indexación, por otro lado, se realiza con el método .loc[] y se utiliza para seleccionar filas o columnas por posición. Por ejemplo:

| df.loc[1] |
| --- |
| nombre Pedro edad 34 sexo M Name: 1, dtype: object |

Para seleccionar varias filas o columnas, se pueden pasar las posiciones como una lista, por ejemplo:

| df.loc[[2,0]] |
| --- |
| nombre edad sexo 2 María 18 F 0 Ana 25 F |

La re-indexación también se puede realizar utilizando la función .reindex(). Esta función nos permite crear un nuevo DataFrame con un índice diferente y rellenar los valores faltantes con un valor específico. Por ejemplo:

| import pandas as pd  # Creamos un DataFrame data = {'nombre': ['Juan', 'Pedro', 'Luis', 'María'],  'edad': [25, 30, 20, 27],  'ciudad': ['Madrid', 'Barcelona', 'Valencia', 'Sevilla']} df = pd.DataFrame(data)  # Reindexamos el DataFrame nuevo\_indice = [3, 2, 1, 0] df\_reindexado = df.reindex(nuevo\_indice) |
| --- |

En este ejemplo, hemos creado un nuevo índice y hemos reordenado el DataFrame según este índice utilizando la función .reindex(). El nuevo DataFrame df\_reindexado tendrá los mismos valores que el DataFrame original df, pero con el orden de las filas cambiado.

La función .reindex() también nos permite especificar cómo queremos manejar los valores faltantes utilizando el parámetro fill\_value. Por ejemplo:

| import pandas as pd  # Creamos un DataFrame data = {'nombre': ['Juan', 'Pedro', 'Luis', 'María'],  'edad': [25, 30, 20, 27],  'ciudad': ['Madrid', 'Barcelona', 'Valencia', 'Sevilla']} df = pd.DataFrame(data) # Reindexamos el DataFrame y llenamos los valores faltantes con ceros nuevo\_indice = [0, 1, 2, 3, 4] df\_reindexado = df.reindex(nuevo\_indice, fill\_value=0) |
| --- |

En este ejemplo, hemos creado un nuevo índice que incluye una fila adicional (4) que no existe en el DataFrame original df. Al especificar fill\_value=0, hemos llenado el valor faltante con ceros.

Los contenedores de secuencia son tipos de datos que pueden almacenar múltiples valores en una sola variable, como listas, tuplas y cadenas. Cada valor en un contenedor de secuencia tiene un índice, que es una posición numérica que comienza desde cero.

La indexación de contenedores de secuencia nos permite acceder a un valor específico en una determinada posición de la secuencia. Por ejemplo, para acceder al tercer elemento de una lista llamada my\_list, podemos usar la siguiente sintaxis:

| my\_list = [10, 20, 30, 40, 50]  print(my\_list[2]) |
| --- |
| > 30 |

En este caso, my\_list[2] accede al tercer elemento de la lista my\_list (el índice 2), que es el valor 30.

La indexación también puede ser negativa, lo que significa que comenzamos a contar desde el final de la secuencia. Por ejemplo, para acceder al último elemento de la lista my\_list, podemos usar la siguiente sintaxis:

| print(my\_list[-1]) |
| --- |
| > 50 |

Aquí, my\_list[-1] accede al último elemento de la lista my\_list, que es el valor 50.

En Pandas, los DataFrames son objetos bidimensionales que nos permiten trabajar con datos estructurados en forma de filas y columnas. La reindexación de DataFrames nos permite cambiar el índice de fila y / o el nombre de las columnas.

Por ejemplo, consideremos el siguiente DataFrame df:

| import pandas as pd  data = {'nombre': ['Juan', 'María', 'Pedro', 'Sofía'],  'edad': [28, 25, 19, 32]} df = pd.DataFrame(data) print(df) |
| --- |
| nombre edad 0 Juan 28 1 María 25 2 Pedro 19 3 Sofía 32 |

Aquí, los índices de fila son los números enteros predeterminados del 0 al 3. Si queremos cambiar el índice de fila a los nombres de las personas, podemos usar el método set\_index() de Pandas:

| df = df.set\_index('nombre') print(df) |
| --- |
| edad nombre  Juan 28 María 25 Pedro 19 Sofía 32 |

Aquí, hemos reindexado el DataFrame df con el índice de fila como los nombres de las personas. También podemos reindexar el DataFrame con un nuevo conjunto de nombres de columna utilizando el método rename():

| df = df.rename(columns={'edad': 'años'}) print(df) |
| --- |
| años nombre  Juan 28 María 25 Pedro 19 Sofía 32 |

Aquí, hemos cambiado el nombre de la columna 'edad' a 'años'. La reindexación es útil cuando necesitamos cambiar la forma en que se accede a los datos en un DataFrame

Además de los índices de rango, los dataframes también pueden ser indexados utilizando etiquetas. Esto se hace mediante el método .loc[], el cual permite seleccionar filas y columnas basándose en etiquetas de fila y columna.

Por ejemplo, si tenemos el siguiente dataframe:

| A B C  2010-01-01 1 2 3  2010-01-02 4 5 6  2010-01-03 7 8 9 |
| --- |

Podemos seleccionar una fila usando su etiqueta de la siguiente manera:

| df.loc['2010-01-02'] |
| --- |
| > A 4  B 5  C 6 Name: 2010-01-02, dtype: int64 |

También podemos seleccionar una columna usando su etiqueta:

| df.loc[:, 'B'] |
| --- |
| > 2010-01-01 2  2010-01-02 5  2010-01-03 8 Name: B, dtype: int64 |

Y por último, podemos seleccionar un valor específico utilizando la etiqueta de fila y columna:

| df.loc['2010-01-02', 'B'] |
| --- |
| > 5 |

Es importante notar que la etiqueta de fila y columna deben ser exactamente iguales a las que aparecen en el índice y las columnas del dataframe. Si intentamos utilizar una etiqueta que no existe en el dataframe, obtendremos un error.

En resumen, la indexación de dataframes es una herramienta fundamental en el análisis de datos. A través de ella podemos seleccionar y manipular los datos que nos interesan, ya sea basándonos en su posición en el dataframe o en etiquetas específicas. El conocimiento de estas técnicas nos permitirá realizar un análisis más preciso y eficiente de nuestros datos.

## Filtrado de datos en un DataFrame Pandas

Ahora, hablemos sobre el filtrado de datos en un DataFrame Pandas. El filtrado nos permite seleccionar un subconjunto de filas o columnas de un DataFrame basándonos en ciertas condiciones. Es una de las tareas más comunes en el análisis de datos y nos sirve para encontrar los valores que nos interesan. En Pandas, podemos filtrar datos utilizando la sintaxis de corchetes [] y los operadores lógicos & (and) y | (or).

Por ejemplo:

| filtro = (DataFrame['edad'] > 21) & (DataFrame['sexo'] == 'M') resultados = DataFrame[filtro] print(resultados) |
| --- |
| > nombre edad sexo  1 Pedro 34 M  3 Juan 42 M |

Este código filtrará los datos del DataFrame df para encontrar a las personas mayores de 21 años y de sexo masculino, y devolverá un nuevo DataFrame con los resultados.

## Ordenación de valores en un DataFrame Pandas

La ordenación de los valores en un DataFrame de Pandas es una tarea importante en el análisis de datos. A menudo, es necesario clasificar los datos en orden ascendente o descendente según un valor específico o una columna. Pandas ofrece el método "sort\_values()" para realizar esta tarea. Este método ordena los valores en función de la columna especificada. El parámetro "ascending" se utiliza para especificar si el orden debe ser ascendente o descendente.

| df = pd.DataFrame({'Nombre':['Pedro', 'Juan', 'Lucía', 'María'],  'Edad':[25, 31, 28, 22],   'Salario':[50000, 60000, 45000, 70000]})  print("DataFrame original:") print(df) |
| --- |
| > DataFrame original: > Nombre Edad Salario  0 Pedro 25 50000   1 Juan 31 60000  2 Lucía 28 45000  3 María 22 70000 |
| df\_ordenado = df.sort\_values(by='Salario', ascending=False) print("DataFrame ordenado por salario en orden descendente:") print(df\_ordenado) |
| > DataFrame ordenado por salario en orden descendente: > Nombre Edad Salario  3 María 22 70000  1 Juan 31 60000  0 Pedro 25 50000  2 Lucía 28 45000 |

En este ejemplo, creamos un DataFrame con tres columnas: "Nombre", "Edad" y "Salario". Luego, utilizamos el método "sort\_values()" para ordenar los valores según la columna "Salario" en orden descendente. El resultado es un nuevo DataFrame llamado "df\_ordenado" que se muestra en la pantalla.

## Crear columnas en un DataFrame

Otro aspecto importante de Pandas es la creación de nuevas columnas a partir de las existentes. Podemos crear una nueva columna en nuestro DataFrame df que sea el doble de la columna age, por ejemplo, usando:

| df['age\_double'] = df[Edad] \* 2 |
| --- |
| > Nombre Edad Salario age\_double 0 Pedro 25 50000 50 1 Juan 31 60000 62 2 Lucía 28 45000 56 3 María 22 70000 44 |

En Pandas, también podemos crear nuevas columnas a partir de funciones lambda. Las funciones lambda son funciones anónimas que se utilizan para realizar tareas específicas. Podemos utilizar estas funciones para crear nuevas columnas en un DataFrame de Pandas. Por ejemplo:

| df = pd.DataFrame({'Nombre': ['Pedro', 'Juan', 'Lucía', 'María'],   'Edad': [25, 31, 28, 22]}) df['Cuadrado de Edad'] = df['Edad'].apply(lambda x: x\*\*2) print(df) |
| --- |
| Nombre Edad Cuadrado de Edad 0 Pedro 25 625 1 Juan 31 961 2 Lucía 28 784 3 María 22 484 |

En este ejemplo, creamos un DataFrame con dos columnas: "Nombre" y "Edad". Luego, utilizamos el método "apply()" para aplicar una función lambda a la columna "Edad" y crear una nueva columna llamada "Cuadrado de Edad". La función lambda toma un valor "x" y devuelve el cuadrado de ese valor. El resultado es un nuevo DataFrame con tres columnas

Otra manera de crear columnas es a partir de funciones condicionales. Por ejemplo, si queremos crear una columna category que tenga los valores 'young' para edades menores a 30 y 'old' para edades mayores o iguales a 30, podemos usar:

| df['category'] = df['Edad'].apply(lambda x: 'young' if x < 30 else 'old') print(df) |
| --- |
| Nombre Edad Cuadrado de Edad category 0 Pedro 25 625 young 1 Juan 31 961 old 2 Lucía 28 784 young 3 María 22 484 young |

En algunos casos, es necesario crear nuevas columnas en un DataFrame de Pandas que contengan cadenas de texto. Para hacer esto, podemos utilizar la notación de corchetes y asignar un valor de cadena de texto a la nueva columna. Por ejemplo:

| import pandas as pd  df = pd.DataFrame({'Nombre': ['Pedro', 'Juan', 'Lucía', 'María'],  'Edad': [25, 31, 28, 22]}) df['Ciudad'] = ['Madrid', 'Barcelona', 'Sevilla', 'Valencia'] print(df) |
| --- |
| Nombre Edad Ciudad 0 Pedro 25 Madrid 1 Juan 31 Barcelona 2 Lucía 28 Sevilla 3 María 22 Valencia |

En este ejemplo, creamos un DataFrame con dos columnas: "Nombre" y "Edad". Luego, utilizamos la notación de corchetes para crear una nueva columna llamada "Ciudad" y le asignamos una lista de valores de cadena de texto. El resultado es un nuevo DataFrame con tres columnas: "Nombre", "Edad" y "Ciudad".

## Renombrar y ordenar columnas de un DataFrame Pandas

A veces, es posible que necesitemos cambiar el nombre de una columna en un DataFrame. Podemos hacer esto usando el método rename().

| import pandas as pd  ventas = pd.DataFrame({  'Producto': ['A', 'B', 'C', 'D'],  'Ventas': [100, 200, 300, 400],  'Beneficio': [50, 100, 150, 200] })  # Renombrar la columna 'Ventas' como 'Unidades vendidas' ventas = ventas.rename(columns={'Ventas': 'Unidades vendidas'}) print(ventas) |
| --- |
| Producto Unidades vendidas Beneficio 0 A 100 50 1 B 200 100 2 C 300 150 3 D 400 200 |

También podemos cambiar el orden de las columnas en un DataFrame mediante el método reindex().

| # Cambiar el orden de las columnas ventas = ventas.reindex(columns=['Producto', 'Unidades vendidas', 'Beneficio']) print(ventas) |
| --- |
| Producto Unidades vendidas Beneficio 0 A 100 50 1 B 200 100 2 C 300 150 3 D 400 200 |

En este ejemplo, creamos un DataFrame llamado ventas con tres columnas: "Producto", "Ventas" y "Beneficio". Luego, renombramos la columna "Ventas" como "Unidades vendidas" y cambiamos el orden de las columnas para que queden en el orden "Producto", "Unidades vendidas" y "Beneficio".

## Uso de groupby en Pandas

En Pandas, también es muy común utilizar la función groupby, que nos permite agrupar filas por valores de una o más columnas y aplicar funciones a esas agrupaciones. Por ejemplo, si queremos agrupar nuestro DataFrame df por la columna gender y calcular la media de la columna age para cada grupo, podemos usar:

| df.groupby('gender')['age'].mean() |
| --- |

## Concatenación de DataFrames

Finalmente, podemos unir dos DataFrames usando la función concat. Si tenemos dos DataFrames df1 y df2 con las mismas columnas y queremos unirlos por filas, podemos usar

| df\_concat = pd.concat([df1, df2]) |
| --- |

Si queremos unirlos por columnas, podemos usar:

| df\_concat = pd.concat([df1, df2], axis=1) |
| --- |

En conclusión, en este primer tema hemos aprendido las bases del uso de Pandas para el análisis de datos en Python. Hemos visto cómo crear DataFrames a partir de diccionarios, y cómo manipular los datos usando métodos útiles como loc, iloc, y at. También hemos aprendido a filtrar y ordenar datos, crear columnas nuevas a partir de funciones lambda y condicionales, y renombrar y ordenar columnas. Además, hemos explorado el uso de groupby para agrupar y resumir datos, y la concatenación de DataFrames.

El uso de Pandas es fundamental para cualquier proyecto de análisis de datos, y estos conceptos básicos que hemos cubierto en este módulo son esenciales para cualquier analista de datos que quiera dominar el uso de Pandas. En el siguiente módulo, continuaremos avanzando en nuestro aprendizaje, enfocándonos en la limpieza y preparación de los datos para su análisis, lo que es crucial para obtener resultados precisos y confiables en el análisis de datos.