Introducción

La agregación de datos es uno de los conceptos más importantes en el análisis de datos. Esta técnica permite resumir, reorganizar y obtener información valiosa de grandes volúmenes de datos al agruparlos según categorías comunes. La agregación puede reducir la complejidad de los datos, proporcionando resúmenes numéricos clave que permiten tomar decisiones basadas en evidencia. Para lograr esto de manera eficiente, utilizamos bibliotecas como Pandas, que nos ofrece varias herramientas poderosas para este propósito, como `groupby`, `pivot_table` y `crosstab`.

A continuación, profundizaremos en cómo aplicar estas técnicas en Pandas para realizar operaciones avanzadas de agregación de datos, esenciales para extraer información útil de cualquier conjunto de datos estructurados.

Importación de Librerías

Antes de comenzar a trabajar con la agregación de datos, es necesario importar las librerías clave:

```
import pandas as pd
import numpy as np
```

Cargar el Conjunto de Datos

Para este ejemplo, trabajaremos con un conjunto de datos llamado `employees.csv`. Este dataset contiene información de empleados que incluye columnas como la edad, el departamento, el campo de educación, el salario por hora, entre otros.

Cargamos el dataset de la siguiente manera:

```
df = pd.read_csv("datasets/employees.csv")
```

¿De qué trata el conjunto de datos? Vamos a echar un vistazo:

Para obtener una vista previa de los primeros registros del dataset y entender mejor su estructura:

```
df.head()
```

```
   Age Attrition     BusinessTravel  DailyRate              Department
\
0   41       Yes      Travel_Rarely       1102                   Sales

1   49        No  Travel_Frequently        279  Research & Development

2   37       Yes      Travel_Rarely       1373  Research & Development

3   33        No  Travel_Frequently       1392  Research & Development

4   27        No      Travel_Rarely        591  Research & Development


   DistanceFromHome  Education EducationField  EmployeeCount
EmployeeNumber  \
0                 1          2  Life Sciences              1
1
1                 8          1  Life Sciences              1
2
2                 2          2          Other              1
4
3                 3          4  Life Sciences              1
5
4                 2          1        Medical              1
7

   ...  RelationshipSatisfaction StandardHours  StockOptionLevel  \
0  ...                         1            80                 0
1  ...                         4            80                 1
2  ...                         2            80                 0
3  ...                         3            80                 0
4  ...                         4            80                 1

   TotalWorkingYears  TrainingTimesLastYear WorkLifeBalance
YearsAtCompany  \
0                  8                      0               1
6
1                 10                      3               3
10
```

```
2                    7                          3                 3
0
3                    8                          3                 3
8
4                    6                          3                 3
2

  YearsInCurrentRole  YearsSinceLastPromotion  YearsWithCurrManager
0                  4                        0                     5
1                  7                        1                     7
2                  0                        0                     0
3                  7                        3                     0
4                  2                        2                     2

[5 rows x 35 columns]
```

Agregación de Datos

La agregación de datos implica agrupar y resumir datos con el objetivo de facilitar su análisis. A continuación, veremos las principales herramientas en Pandas para realizar estas tareas.

groupby

La función `groupby` de Pandas es una de las herramientas más poderosas para la agregación de datos. Te permite dividir un DataFrame en grupos basados en uno o más criterios, y luego aplicar funciones de agregación como sumar, promediar, contar, entre otras.

¿Cómo funciona `groupby`?

El proceso de `groupby` se puede dividir en tres pasos:

1. **Dividir:** Los datos se dividen en grupos según los valores de una o más columnas.
2. **Aplicar:** Se aplica una función de agregación (como `mean`, `sum`, `count`) sobre cada grupo.
3. **Combinar:** Los resultados se combinan en un DataFrame.

Sintaxis básica de groupby:

```
df[subset].groupby(category).aggregation()
```

Ejemplos prácticos de groupby

A continuación, exploraremos varios ejemplos de cómo usar `groupby` para realizar agregaciones en nuestros datos.

1. Agrupar por departamentos y calcular la edad promedio:

```
df[["Age", "Department"]].groupby("Department").mean()
```

```
                               Age
Department
Human Resources          37.809524
Research & Development   37.042664
Sales                    36.542601
```

Alternativamente, podemos usar el método `agg` que ofrece más flexibilidad:

```
df[["Age", "Department"]].groupby("Department").agg("mean")
```

```
                               Age
Department
Human Resources          37.809524
Research & Development   37.042664
Sales                    36.542601
```

2. Agrupar por departamentos y calcular la edad máxima:

```
df[["Age", "Department"]].groupby("Department").max()
```

```
                          Age
Department
Human Resources            59
Research & Development     60
Sales                      60
```

Otra forma de hacerlo:

```
df[["Age", "Department"]].groupby("Department").agg({"Age":"max"})
```

```
                          Age
Department
Human Resources            59
Research & Development     60
Sales                      60
```

3. Agrupar por departamentos y calcular la edad mínima:

```
df[["Age", "Department"]].groupby("Department").min()
```

```
                          Age
Department
Human Resources            19
Research & Development     18
Sales                      18
```

Alternativamente:

```
df[["Age", "Department"]].groupby("Department").agg({"Age":"min"})
```

```
                          Age
Department
Human Resources           19
Research & Development    18
Sales                     18
```

4. Agrupar por campos de educación y calcular el salario medio por departamento:

Este ejemplo muestra cómo agrupar los datos en función de múltiples columnas y calcular el salario por hora promedio en cada campo de educación dentro de los departamentos.

```
df[["EducationField", "HourlyRate",
"Department"]].groupby(by=["Department",
"EducationField"]).agg({"HourlyRate":"mean"})
```

```
                                        HourlyRate
Department             EducationField
Human Resources        Human Resources    60.888889
                       Life Sciences      61.625000
                       Medical            72.076923
                       Other              76.000000
                       Technical Degree   64.000000
Research & Development Life Sciences      66.570455
                       Medical            66.330579
                       Other              62.203125
                       Technical Degree   66.351064
Sales                  Life Sciences      68.153333
                       Marketing          66.150943
                       Medical            59.943182
                       Other              60.333333
                       Technical Degree   67.676471
```

O podemos invertir el orden de las agrupaciones:

```
df[["EducationField", "HourlyRate",
"Department"]].groupby(by=["EducationField",
"Department"]).agg({"HourlyRate":"mean"})
```

```
                                        HourlyRate
EducationField   Department
Human Resources  Human Resources           60.888889
Life Sciences    Human Resources           61.625000
                 Research & Development    66.570455
                 Sales                     68.153333
Marketing        Sales                     66.150943
Medical          Human Resources           72.076923
                 Research & Development    66.330579
                 Sales                     59.943182
Other            Human Resources           76.000000
```

```
                  Research & Development   62.203125
                  Sales                    60.333333
Technical Degree  Human Resources          64.000000
                  Research & Development   66.351064
                  Sales                    67.676471
```

5. Agregar múltiples campos para calcular estadísticas a través de varias categorías:

También es posible agrupar por más de una columna y aplicar funciones de agregación a varias columnas del DataFrame a la vez.

```
df[["EducationField", "HourlyRate", "Department", "StandardHours",
"Age"]].groupby(by=["Department", "EducationField"]).mean()
```

```
                                        HourlyRate  StandardHours
Age
Department              EducationField
Human Resources         Human Resources    60.888889           80.0
37.037037
                        Life Sciences      61.625000           80.0
39.375000
                        Medical            72.076923           80.0
39.846154
                        Other              76.000000           80.0
34.666667
                        Technical Degree   64.000000           80.0
32.500000
Research & Development  Life Sciences      66.570455           80.0
36.997727
                        Medical            66.330579           80.0
37.242424
                        Other              62.203125           80.0
36.187500
                        Technical Degree   66.351064           80.0
37.063830
Sales                   Life Sciences      68.153333           80.0
37.186667
                        Marketing          66.150943           80.0
37.924528
                        Medical            59.943182           80.0
34.727273
                        Other              60.333333           80.0
32.000000
                        Technical Degree   67.676471           80.0
33.941176
```

Función `agg` en Pandas

La función `agg` en Pandas permite aplicar múltiples funciones de agregación al mismo tiempo. Con `agg`, puedes aplicar una combinación de operaciones, como calcular el promedio, suma, conteo, o incluso aplicar funciones definidas por el usuario.

Algunas funciones de agregación comunes disponibles con `agg`:

- **mean**: Calcula el promedio de los valores en cada grupo.
- **sum**: Suma todos los valores en cada grupo.
- **count**: Cuenta el número de filas en cada grupo.
- **max**: Encuentra el valor máximo en cada grupo.
- **min**: Encuentra el valor mínimo en cada grupo.

Ejemplo de uso de `agg`:

En este ejemplo, aplicamos múltiples funciones de agregación al mismo tiempo, proporcionando un resumen más completo de los datos.

```
df.groupby("Department").agg({"Age":["mean", "max", "min"],
"HourlyRate":"mean"})
```

```
                            Age          HourlyRate
                           mean max min       mean
Department
Human Resources         37.809524  59  19  64.301587
Research & Development  37.042664  60  18  66.167534
Sales                   36.542601  60  18  65.520179
```

¿Por qué usar `agg`?

La función `agg` es especialmente útil cuando necesitas obtener múltiples estadísticas de un solo paso. Permite ahorrar tiempo y líneas de código, mientras te proporciona una visión integral de tus datos agrupados.

Más información sobre `agg`

Para obtener más detalles sobre las opciones avanzadas de la función `agg`, puedes consultar la documentación oficial de Pandas aquí.

La agregación de datos es una técnica esencial para extraer valor de grandes conjuntos de datos. Usando herramientas como `groupby` y `agg` en Pandas, puedes organizar y resumir datos de manera eficiente para responder a preguntas críticas y obtener una visión más profunda. Estas funciones son fundamentales para cualquier analista o científico de datos que desee explorar y comprender mejor sus datos.

Tablas Dinámicas

Las tablas dinámicas son herramientas fundamentales en el análisis de datos, especialmente cuando necesitas analizar y comparar múltiples dimensiones de tu conjunto de datos al mismo tiempo. En Pandas, la función `pivot_table` te permite crear tablas dinámicas similares a las que se encuentran en las hojas de cálculo, pero dentro de un DataFrame. Esto facilita el análisis, ya que puedes reorganizar los datos, aplicar funciones de agregación, y obtener resúmenes de manera eficiente.

¿Qué es una tabla dinámica?

Una tabla dinámica reorganiza los datos originales, lo que permite analizar diferentes aspectos de ellos al aplicar funciones de agregación como el cálculo de promedios, sumas, conteos, entre otros. Esto es útil para descubrir patrones y tendencias dentro de grandes volúmenes de datos. Es ideal para comparar métricas a través de diferentes grupos, como departamentos, categorías de productos o cualquier otra variable relevante.

Parámetros clave de `pivot_table`

La función `pivot_table` acepta varios parámetros importantes para configurar la tabla dinámica:

- **`df`**: El DataFrame que contiene los datos originales.
- **`values`**: La columna cuyos valores serán agregados (por ejemplo, "Edad", "Salario", etc.).
- **`index`**: Las columnas que se convertirán en los índices o filas de la tabla dinámica.
- **`columns`**: Las columnas que se convertirán en las columnas de la tabla dinámica.
- **`aggfunc`**: La función de agregación que se aplicará a los valores (por defecto es 'mean', pero puedes usar otras como `sum`, `count`, etc.).

Al aplicar `pivot_table`, tus datos se organizan en un formato más estructurado, facilitando operaciones de agregación sobre diferentes categorías o grupos. Esto es particularmente útil cuando necesitas resumir información a través de múltiples dimensiones.

Ejemplo práctico: Edad media por departamento

A continuación, un ejemplo de cómo calcular la edad media de los empleados en cada departamento usando `pivot_table`:

```
df.pivot_table(
    values="Age",
    index="Department",
    aggfunc="mean"
)
```

```
                                Age
Department
Human Resources         37.809524
```

```
Research & Development  37.042664
Sales                   36.542601
```

Este código genera una tabla dinámica que muestra el promedio de edad de los empleados por cada departamento. De manera similar, puedes aplicar otras funciones de agregación.

Otros ejemplos de tablas dinámicas

1. **Edad máxima por departamento:**

```
df.pivot_table(
    values="Age",
    index="Department",
    aggfunc="max"
)
```

```
                          Age
Department
Human Resources            59
Research & Development     60
Sales                      60
```

1. **Edad mínima por departamento:**

```
df.pivot_table(
    values="Age",
    index="Department",
    aggfunc="min"
)
```

```
                          Age
Department
Human Resources            19
Research & Development     18
Sales                      18
```

1. **Edad promedio por departamento:**

```
df.pivot_table(
    values="Age",
    index="Department",
    aggfunc="mean"
)
```

```
                                Age
Department
Human Resources           37.809524
Research & Development    37.042664
Sales                     36.542601
```

1. **Salario máximo por departamento y campo de educación:**

Este ejemplo muestra cómo crear una tabla dinámica con múltiples índices (departamento y campo de educación) y calcular el salario máximo dentro de cada combinación:

```
df.pivot_table(
    values="MonthlyIncome",
    index=["Department", "EducationField"],
    aggfunc="max"
)
```

```
                                        MonthlyIncome
Department             EducationField
Human Resources        Human Resources          19636
                       Life Sciences            19717
                       Medical                  18200
                       Other                     7988
                       Technical Degree          4323
Research & Development Life Sciences            19999
                       Medical                  19859
                       Other                    19613
                       Technical Degree         19943
Sales                  Life Sciences            19847
                       Marketing                19845
                       Medical                  19833
                       Other                    10932
                       Technical Degree         16872
```

Las tablas dinámicas con múltiples índices te permiten analizar datos en más de una dimensión, lo cual es extremadamente útil cuando se trata de conjuntos de datos complejos.

¿Cuándo usar `pivot_table` en lugar de `groupby`?

`pivot_table` y `groupby` son herramientas poderosas que sirven para propósitos similares, pero se utilizan en diferentes escenarios. `groupby` es más flexible y directo cuando solo necesitas agrupar y resumir datos sin reorganizarlos, mientras que `pivot_table` es ideal para crear una vista más organizada de los datos, con la posibilidad de agregar múltiples dimensiones de filas y columnas. Para más detalles, consulta GroupBy vs. Pivot Table.

Más información sobre `pivot_table`

Para explorar más sobre las opciones avanzadas de `pivot_table`, como manejar valores nulos, aplicar varias funciones de agregación a la vez o ajustar el formato de salida, puedes consultar la documentación oficial de Pandas aquí.

Tabulación Cruzada (crosstab)

Otra herramienta clave en el análisis de datos es la **tabulación cruzada**. En Pandas, la función `pd.crosstab()` te permite calcular la frecuencia de una variable categórica en relación con otra. Esta técnica es muy útil cuando se desea observar la distribución de categorías y la relación entre diferentes variables.

Mientras que `pivot_table` se usa para resumir datos numéricos, `crosstab` está diseñado para trabajar principalmente con datos categóricos, generando tablas de frecuencia que muestran cuántas veces ocurren combinaciones de categorías.

Sintaxis básica de `crosstab`

Para crear una tabulación cruzada simple entre dos columnas categóricas, puedes usar la siguiente sintaxis:

```
pd.crosstab(df["column1"], df["column2"])
```

Ejemplo práctico: Tabulación cruzada entre departamento y campo de educación

Este ejemplo genera una tabla cruzada que muestra la distribución de empleados por departamento y campo de educación:

```
pd.crosstab(df["Department"], df["EducationField"])
```

```
EducationField          Human Resources  Life Sciences  Marketing
Medical  \
Department

Human Resources                      27             16          0
13
Research & Development                0            440          0
363
Sales                                 0            150        159
88

EducationField          Other  Technical Degree
Department
Human Resources             3                 4
Research & Development     64                94
Sales                      15                34
```

Tabulación cruzada con funciones de agregación

Además de contar frecuencias, también puedes usar `crosstab` para aplicar funciones de agregación. Por ejemplo, si quisieras calcular el ingreso mensual promedio por departamento y campo de educación, podrías hacerlo así:

```
pd.crosstab(
    df["Department"],
    df["EducationField"],
    values=df["MonthlyIncome"],
    aggfunc="mean"
)
```

```
EducationField          Human Resources  Life Sciences    Marketing  \
Department
Human Resources             7241.148148    6914.062500          NaN
Research & Development              NaN    6179.984091          NaN
Sales                               NaN    7246.233333  7348.584906

EducationField              Medical        Other  Technical Degree
Department
Human Resources         6594.076923  5016.666667       3081.250000
Research & Development  6539.223140  6278.687500       5760.819149
Sales                   6377.227273  5398.733333       6066.294118
```

Este código te dará una tabla cruzada donde los valores representan el ingreso mensual promedio para cada combinación de departamento y campo de educación.

Tanto las tablas dinámicas (`pivot_table`) como las tablas de tabulación cruzada (`crosstab`) son herramientas esenciales en el análisis de datos. `pivot_table` permite organizar y resumir datos numéricos a través de diferentes dimensiones, mientras que `crosstab` se enfoca en analizar la relación entre variables categóricas. Ambas herramientas son claves para descubrir patrones y obtener insights valiosos de conjuntos de datos complejos.

Combinación de datos: mezclando DataFrames

En el análisis de datos, es común encontrarse con escenarios en los que necesitamos combinar datos de diferentes fuentes o reorganizarlos para obtener un análisis más significativo. Pandas ofrece métodos eficientes para fusionar, unir y concatenar DataFrames, lo que facilita la manipulación de grandes conjuntos de datos. En esta sección, exploraremos estas técnicas clave, enfocándonos en `concat` y sus opciones avanzadas.

Para una comprensión más profunda de estas técnicas, aquí tienes algunos recursos útiles:

- Real Python: Pandas Merge, Join, y Concat
- Documentación de Pandas: Fusión y Unión

`pd.concat()`: Combinando DataFrames

La concatenación es una de las formas más sencillas de combinar DataFrames en Pandas. Con `pd.concat()`, puedes unir DataFrames a lo largo de un eje, que generalmente es el eje 0 (verticalmente), para apilarlos uno debajo del otro. Esto es útil cuando necesitas combinar datos de varias fuentes o extender un conjunto de datos con nuevas filas o columnas.

Sintaxis básica de `concat()`

La sintaxis básica de `pd.concat()` es la siguiente:

```
result = pd.concat([df1, df2, ...], axis=0)
```

En este ejemplo:

- **`df1`, `df2`**: son los DataFrames que deseas concatenar.
- **`axis=0`**: especifica que la concatenación será vertical, es decir, las filas se apilarán unas sobre otras.

¿Por qué usar `pd.concat()`?

La concatenación es útil cuando tienes datos de varias fuentes o periodos diferentes y necesitas combinar estos conjuntos en uno solo. Al dominar `pd.concat()`, puedes gestionar de manera eficiente conjuntos de datos que crecen con el tiempo, como series temporales o archivos de registro. Además, te permite crear un DataFrame único a partir de múltiples fuentes de información para un análisis más completo.

Ejemplo práctico: Concatenación de DataFrames

Primero, carguemos algunos datos de ejemplo:

```
df =
pd.read_csv('https://raw.githubusercontent.com/justmarkham/DAT8/master
/data/u.user',
                 sep='|', index_col='user_id')
df.head()
```

```
         age gender  occupation zip_code
user_id
1         24      M  technician    85711
2         53      F       other    94043
3         23      M      writer    32067
4         24      M  technician    43537
5         33      F       other    15213
```

Ahora, seleccionemos una muestra de los datos para crear dos DataFrames más pequeños:

```
first_df = df.sample(frac=0.1)
second_df = df.sample(frac=0.1)
```

Concatenando DataFrames verticalmente (`axis=0`)

Concatenar a lo largo de `axis=0` implica apilar los DataFrames uno encima del otro, alineando las columnas. Por ejemplo:

```
pd.concat([first_df, second_df], axis=0, keys=["Table 1", "Table 2"])
```

```
                 age gender     occupation zip_code
        user_id
Table 1 297       29      F       educator    98103
        406       52      M       educator    93109
        580       16      M        student    17961
        778       34      M        student    01960
        295       31      M       educator    50325
```

```
...               ...    ...           ...      ...
Table 2 868       21      M     programmer    55303
        185       53      F      librarian    97403
        789       29      M          other    55420
        400       33      F  administrator    78213
        767       70      M       engineer    00000

[188 rows x 4 columns]
```

Explicación:

- La función concatena `first_df` y `second_df` verticalmente.
- El parámetro `keys` añade un índice jerárquico al DataFrame resultante, etiquetando las dos tablas como "Tabla 1" y "Tabla 2".
- El resultado es un DataFrame con un índice jerárquico (`MultiIndex`), lo que permite identificar de qué tabla provienen los datos.

Concatenando DataFrames horizontalmente (`axis=1`)

También es posible concatenar los DataFrames uno al lado del otro, es decir, agregando las columnas:

```
pd.concat([first_df, second_df], axis=1, keys=["left", "right"])
```

```
         left                                 right

          age gender occupation zip_code   age gender     occupation
zip_code
user_id

297      29.0      F   educator    98103   NaN    NaN            NaN
NaN
406      52.0      M   educator    93109   NaN    NaN            NaN
NaN
580      16.0      M    student    17961   NaN    NaN            NaN
NaN
778      34.0      M    student    01960   NaN    NaN            NaN
NaN
295      31.0      M   educator    50325   NaN    NaN            NaN
NaN
...       ...    ...        ...      ...   ...    ...            ...
...
554       NaN    NaN        NaN      NaN  32.0      M      scientist
62901
868       NaN    NaN        NaN      NaN  21.0      M     programmer
55303
789       NaN    NaN        NaN      NaN  29.0      M          other
55420
400       NaN    NaN        NaN      NaN  33.0      F  administrator
```

```
78213
767       NaN    NaN        NaN      NaN  70.0      M       engineer
00000

[180 rows x 8 columns]
```

Explicación:

- Esta operación concatena `first_df` y `second_df` horizontalmente, lo que significa que coloca las columnas de uno al lado del otro.
- El parámetro `keys` etiqueta las columnas de los DataFrames originales como "izquierda" y "derecha".
- El resultado es un DataFrame con columnas organizadas en secciones etiquetadas como "izquierda" y "derecha".

Visualización de Joins SQL

En la combinación de datos, las operaciones de fusión y unión en Pandas se asemejan mucho a las operaciones de `JOIN` en SQL. A continuación, se muestra una visualización de cómo funcionan los diferentes tipos de `JOIN` en SQL, lo cual es equivalente a muchas de las operaciones que puedes realizar en Pandas con `merge()`.

Los diferentes tipos de combinaciones (`inner`, `outer`, `left`, `right`) permiten controlar cómo se combinan las tablas o DataFrames en función de las coincidencias de los valores clave.

La combinación de DataFrames es esencial en el análisis de datos del mundo real. Ya sea que necesites apilar filas con `concat()` o unir tablas con `merge()`, Pandas te proporciona las herramientas necesarias para manejar datos de múltiples fuentes o estructuras. Con estas técnicas, puedes construir conjuntos de datos integrales que te permiten un análisis más profundo y significativo.

Merge: columnas relacionadas

La función `merge()` en Pandas es una herramienta poderosa cuando necesitas combinar filas de múltiples DataFrames basados en columnas relacionadas. Se utiliza principalmente para uniones al estilo de bases de datos, donde deseas reunir datos de diferentes fuentes que comparten columnas clave comunes.

Entendiendo la Función `merge()`

Puedes pensar en la función `merge()` como una manera de fusionar filas que comparten datos a través de columnas especificadas. Esta función es especialmente útil cuando trabajas con conjuntos de datos que contienen información relacionada, permitiendo que las filas se combinen de manera coherente y estructurada. `merge()` te permite realizar varios tipos de uniones, como uniones internas, externas, izquierdas y derechas, que determinan cómo se combinan las filas de ambos DataFrames.

- **Unión Interna (Inner Join)**: Este tipo de unión devuelve solo las filas donde hay una coincidencia en ambos DataFrames basada en las columnas especificadas. Las filas sin coincidencia se excluyen, lo que significa que solo se retendrán las filas con datos en ambas fuentes.
- **Unión Externa (Full Outer Join)**: Una unión externa devuelve todas las filas cuando hay una coincidencia en cualquiera de los DataFrames izquierdo o derecho. Si no hay una coincidencia, las filas no emparejadas contendrán valores NaN en las columnas donde faltan datos.
- **Unión Izquierda (Left Outer Join)**: Una unión izquierda devuelve todas las filas del DataFrame izquierdo y las filas emparejadas del DataFrame derecho. Si hay filas en el DataFrame izquierdo sin una coincidencia en el derecho, esas filas aún se incluirán, pero con NaN para las columnas del DataFrame derecho.
- **Unión Derecha (Right Outer Join)**: A la inversa, una unión derecha devuelve todas las filas del DataFrame derecho y las filas emparejadas del DataFrame izquierdo. Las filas no emparejadas del DataFrame derecho se incluirán con NaN en las columnas del DataFrame izquierdo.

Uso y Sintaxis

Aquí tienes un esquema básico de cómo usar la función `merge()`:

```
merged_df = pd.merge(left_df, right_df, on='common_column',
how='join_type')
```

- **Parámetros:**
  - `left_df`: El DataFrame izquierdo que deseas fusionar. Este DataFrame sirve como la base para la fusión.

- `right_df`: El DataFrame derecho que deseas fusionar. Este DataFrame contiene los datos que se combinarán con el izquierdo.
- `on`: La(s) columna(s) en las que deseas realizar la fusión. Estas columnas deberían existir en ambos DataFrames y servir como la(s) clave(s) para la operación de fusión. Si no se especifica, Pandas intentará unir por columnas con el mismo nombre.
- `how`: El tipo de unión que deseas realizar. Las opciones son:
  - `'inner'`: Solo las filas con coincidencias en ambos DataFrames.
  - `'outer'`: Todas las filas de ambos DataFrames, con NaN donde no hay coincidencias.
  - `'left'`: Todas las filas del DataFrame izquierdo y las filas coincidentes del DataFrame derecho.
  - `'right'`: Todas las filas del DataFrame derecho y las filas coincidentes del DataFrame izquierdo.

Beneficios de Usar `merge()`

- **Integración de Datos**: `merge()` ayuda a integrar datos de múltiples fuentes o tablas en un conjunto de datos único y comprensivo. Esto es esencial para análisis donde los datos provienen de diferentes orígenes.

- **Análisis de Datos**: Simplifica el proceso de combinar información relacionada, facilitando el análisis de datos y la derivación de insights. Puedes crear nuevos conjuntos de datos que contengan la información más relevante para tu análisis.

- **Operaciones de Base de Datos**: `merge()` se alinea con operaciones comunes de bases de datos como las uniones SQL, permitiendo a los analistas de datos aprovechar su conocimiento de SQL dentro de Pandas. Esto facilita la transición para aquellos que están familiarizados con bases de datos.

- **Consultas Complejas**: Puedes manejar relaciones de datos complejas y realizar consultas que involucren múltiples tablas de manera eficiente, lo cual es común en análisis de datos más avanzados.

Fusionar DataFrames usando `merge()` es una operación fundamental en la manipulación y análisis de datos, especialmente cuando se trata de conjuntos de datos del mundo real de diversas fuentes. Te proporciona la flexibilidad para controlar cómo se combinan los datos y te permite trabajar con relaciones de datos complejas.

Para obtener información más detallada y opciones, puedes referirte a la documentación oficial de Pandas sobre `merge()`.

LEFT Merge

Aquí hay un ejemplo práctico de cómo utilizar la función `merge()` con un tipo de unión izquierda:

```python
from IPython.display import display_html

df1 = pd.DataFrame(
    {
        "key1": ["K0", "K0", "K1", "K2"],
        "key2": ["K0", "K1", "K0", "K1"],
        "A": ["A0", "A1", "A2", "A3"],
        "B": ["B0", "B1", "B2", "B3"],
    }
)

df2 = pd.DataFrame(
    {
        "key1": ["K0", "K1", "K1", "K2"],
        "key2": ["K0", "K0", "K0", "K0"],
        "C": ["C0", "C1", "C2", "C3"],
        "D": ["D0", "D1", "D2", "D3"],
    }
)

# Solo mostrar tablas
df1_styler =
df1.style.set_table_attributes("style='display:inline'").set_caption('
Tabla Izquierda')
df2_styler =
df2.style.set_table_attributes("style='display:inline'").set_caption('
Tabla Derecha')
display_html(df1_styler._repr_html_() + " " +
df2_styler._repr_html_(), raw=True)

# Merge: predeterminado es inner
merged_inner = pd.merge(df1, df2)  # Unión interna por defecto
print("Unión interna:")
print(merged_inner)
```

```
Unión interna:
  key1 key2   A   B   C   D
0   K0   K0  A0  B0  C0  D0
1   K1   K0  A2  B2  C1  D1
2   K1   K0  A2  B2  C2  D2
```

```python
# Merges: unión izquierda
merged_left = pd.merge(df1, df2, how="left")  # Unión izquierda
print("\nUnión izquierda:")
print(merged_left)
```

```
Unión izquierda:
  key1 key2   A   B    C    D
0   K0   K0  A0  B0   C0   D0
1   K0   K1  A1  B1  NaN  NaN
```

```
2   K1   K0  A2  B2   C1   D1
3   K1   K0  A2  B2   C2   D2
4   K2   K1  A3  B3  NaN  NaN
```

```
# Merges: unión derecha
merged_right = pd.merge(df1, df2, how="right")  # Unión derecha
print("\nUnión derecha:")
print(merged_right)
```

```
Unión derecha:
  key1 key2    A    B   C   D
0   K0   K0   A0   B0  C0  D0
1   K1   K0   A2   B2  C1  D1
2   K1   K0   A2   B2  C2  D2
3   K2   K0  NaN  NaN  C3  D3
```

```
# Merges: unión interna
merged_inner = pd.merge(df1, df2, how="inner")  # Unión interna
print("\nUnión interna:")
print(merged_inner)
```

```
Unión interna:
  key1 key2   A   B   C   D
0   K0   K0  A0  B0  C0  D0
1   K1   K0  A2  B2  C1  D1
2   K1   K0  A2  B2  C2  D2
```

```
# Merges: unión externa
outer_merge = pd.merge(df1, df2, how="outer")  # Unión externa
print("\nUnión externa:")
print(outer_merge)
```

```
Unión externa:
  key1 key2    A    B    C    D
0   K0   K0   A0   B0   C0   D0
1   K0   K1   A1   B1  NaN  NaN
2   K1   K0   A2   B2   C1   D1
3   K1   K0   A2   B2   C2   D2
4   K2   K1   A3   B3  NaN  NaN
5   K2   K0  NaN  NaN   C3   D3
```

Merging & Concatenating en dos columnas diferentes

También puedes realizar concatenaciones y fusiones en columnas diferentes. Aquí tienes ejemplos:

```
df1_docs = pd.DataFrame({'Locations': ['Spain', 'France', 'Portugal',
'Spain'],
                         'city': ["Barcelona", "Paris", "Lisbon",
"Madrid"]})
df2_docs = pd.DataFrame({'More locations': ['Spain', 'France',
'Portugal', 'Spain'],
                         'city': ["Madrid", "Lyon", "Porto",
"Albacete"]})

# Concatenando en dos columnas diferentes
concatenated = pd.concat([df1_docs, df2_docs], axis=1, keys=["1st
table", "2nd table"])
print("Concatenación en dos columnas:")
print(concatenated)
```

```
Concatenación en dos columnas:
  1st table                 2nd table
  Locations       city More locations      city
0     Spain  Barcelona          Spain    Madrid
1    France      Paris         France      Lyon
2  Portugal     Lisbon       Portugal     Porto
3     Spain     Madrid          Spain  Albacete
```

```
# Merging en dos columnas diferentes
merged_docs = df1_docs.merge(df2_docs, left_on='Locations',
right_on='More locations', suffixes=["_fromlastyear",
"_fromthisyear"])
print("\nFusión en dos columnas diferentes:")
print(merged_docs)
```

```
Fusión en dos columnas diferentes:
  Locations city_fromlastyear More locations city_fromthisyear
0     Spain         Barcelona          Spain            Madrid
1     Spain         Barcelona          Spain          Albacete
2     Spain            Madrid          Spain            Madrid
3     Spain            Madrid          Spain          Albacete
4    France             Paris         France              Lyon
5  Portugal            Lisbon       Portugal             Porto
```

Join: relacionando índices

La función `join()`, a diferencia de `merge()`, une los DataFrames y donde no hay registros en el "índice" se insertarán NaN. Este método es especialmente útil cuando deseas combinar DataFrames que comparten índices.

```
import numpy as np
import pandas as pd
from IPython.display import display_html
```

```
left_df = pd.DataFrame({"A": ["A0", "A1", "A2"], "B": ["B0", "B1",
"B2"]}, index=["K0", "K1", "K2"])
right_df = pd.DataFrame({"C": ["C0", "C2", "C3"], "D": ["D0", "D2",
"D3"]}, index=["K0", "K2", "K3"])

joined_inner = left_df.join(right_df, how="inner")
print("Join (Unión interna):")
print(joined_inner)
```

```
Join (Unión interna):
     A   B   C   D
K0  A0  B0  C0  D0
K2  A2  B2  C2  D2
```

Diferencias entre `join` y `merge`

Al combinar DataFrames en Pandas, tienes dos métodos principales a tu disposición: `join()` y `merge()`. Cada método tiene su propio conjunto de características y casos de uso. A continuación, se presenta una comparación de estos dos métodos basada en diferentes características de unión:

Característica de Unión	`join()`	`merge()`
Unión Interna (Inner Join)	Sí	Sí
Unión Izquierda (Left Join)	Sí	Sí
Unión Derecha (Right Join)	Sí	Sí
Unión Externa (Outer Join)	Sí	Sí
Unión Cruzada (Cross Join)	X	Sí
Unión en Índices	Sí	Sí
Unión en Columnas	X	Sí
Izquierda en Columna, Derecha en Índice	Sí	Sí
Izquierda en Índice, Derecha en Columna	X	Sí

Explicación

- **Unión Interna (Inner Join)**: Tanto `join()` como `merge()` admiten uniones internas, que devuelven solo las filas con valores coincidentes en ambos DataFrames.

- **Unión Izquierda (Left Join)**: Ambos métodos permiten uniones izquierdas, que incluyen todas las filas del DataFrame izquierdo y las filas coincidentes del DataFrame derecho.
- **Unión Derecha (Right Join)**: Ambos métodos admiten uniones derechas, que incluyen todas las filas del DataFrame derecho y las filas coincidentes del DataFrame izquierdo.
- **Unión Externa (Outer Join)**: Ambos métodos permiten uniones externas, que incluyen todas las filas de ambos DataFrames, rellenando los valores faltantes con NaN donde sea necesario.
- **Unión Cruzada (Cross Join)**: Mientras que `merge()` puede realizar uniones cruzadas, `join()` no las admite. Las uniones cruzadas resultan en un producto cartesiano de filas entre dos DataFrames.
- **Unión en Índices**: Tanto `join()` como `merge()` pueden realizar uniones basadas en el índice de los DataFrames.
- **Unión en Columnas**: `merge()` permite unir DataFrames en columnas específicas, mientras que `join()` no ofrece esta capacidad.
- **Izquierda en Columna, Derecha en Índice**: Ambos métodos admiten la unión en una columna del DataFrame izquierdo y el índice del DataFrame derecho.
- **Izquierda en Índice, Derecha en Columna**: `merge()` puede unir en el índice del DataFrame izquierdo y una columna del DataFrame derecho, pero `join()` no ofrece esta opción.

Esta comparación debería ayudarte a elegir el método apropiado basado en tus requisitos específicos de fusión.

Metodos usuales de pandas

```
df.head() # muestra las primeras filas, por defecto 5 filas
df.tail() # muestra las últimas filas, por defecto 5 filas
df.describe() # descripción estadística
df.info() # información del DataFrame
df.columns # muestra las columnas
df.index # muestra el índice
df.dtypes # muestra los tipos de datos de las columnas
df.plot() # genera un gráfico
df.hist() # genera un histograma
df.col.value_counts() # cuenta los valores únicos de una columna
df.col.unique() # devuelve los valores únicos de una columna
df.copy() # copia el DataFrame
df.drop() # elimina columnas o filas (eje=0,1)
df.dropna() # elimina valores nulos
df.fillna() # rellena valores nulos
```

```
df.shape # dimensiones del DataFrame
df._get_numeric_data() # selecciona columnas numéricas
df.rename() # renombra columnas
df.str.replace() # reemplaza valores de columnas de tipo string
df.astype(dtype='float32') # cambia el tipo de dato
df.iloc[] # localiza por índice
df.loc[] # localiza por etiqueta
df.transpose() # transpone el DataFrame
df.T # también transpone el DataFrame
df.sample(n, frac) # toma una muestra del DataFrame
df.col.sum() # suma de una columna
df.col.max() # máximo de una columna
df.col.min() # mínimo de una columna
df[col] # selecciona una columna
df.col # referencia a una columna
df.isnull() # valores nulos
df.isna() # también valores nulos
df.notna() # valores que no son nulos
df.drop_duplicates() # elimina duplicados
df.reset_index(inplace=True) # reinicia el índice y lo sobrescribe
```

Más material

- Read the docs!
- Cheatsheet
- Exercises to practice
- More on merge, concat, and join. And even more!