LA LIBRERÍA PANDAS

M3: OBTENCIÓN Y PREPARACIÓN DE DATOS

|AE2: UTILIZAR MÉTODOS BÁSICOS DE EXPLORACIÓN EN UN SET DE DATOS UTILIZANDO ESTRUCTURAS DE TIPO SERIE Y DATAFRAME DE LA LIBRERÍA PANDAS PARA LA SELECCIÓN, FILTRADO Y SUMARIZACIÓN DE LOS DATOS PARA LA RESOLUCIÓN DE UN PROBLEMA.

Introducción👩‍💻

Pandas es una de las bibliotecas más utilizadas en Python para el análisis y manipulación de datos. Su versatilidad y facilidad de uso la convierten en una herramienta fundamental para científicos de datos, analistas y programadores que trabajan con grandes volúmenes de información.

Este manual tiene como objetivo proporcionar una guía completa sobre el uso de Pandas, desde los conceptos básicos hasta técnicas avanzadas de manipulación de datos. Se explorarán los principales tipos de datos que maneja Pandas, como las Series y los DataFrames, y se detallarán sus métodos más utilizados para la exploración y transformación de conjuntos de datos.

A lo largo de este documento, se abordarán ejemplos prácticos y explicaciones detalladas para que el usuario pueda aplicar los conocimientos adquiridos en contextos reales. Al finalizar, el lector estará preparado para utilizar Pandas de manera eficiente en análisis de datos, machine learning y otras áreas de la ciencia de datos.

Pandas se ha convertido en una herramienta indispensable dentro del ecosistema de Python debido a su integración con bibliotecas como NumPy, Matplotlib y Scikit-learn, lo que facilita su aplicación en análisis exploratorio, modelado de datos y visualización.

Aprendizaje esperado

Cuando finalices la lección serás capaz de:

1. Comprender la importancia y funcionalidad de Pandas en el análisis de datos en Python.
2. Trabajar con estructuras de datos de Pandas, incluyendo Series y DataFrames.
3. Aplicar métodos para la manipulación y exploración de datos.
4. Realizar operaciones sobre Series y DataFrames de manera eficiente.
5. Extraer información relevante utilizando métodos básicos de exploración y sumarización.
6. Aplicar filtros y selecciones condicionales sobre los datos.
7. Manejar métodos de agrupación y transformación de datos para su posterior análisis.

LA LIBRERÍA PANDAS

1. RESEÑA DE LA LIBRERÍA PANDAS

Pandas es una biblioteca de código abierto construida sobre NumPy y diseñada para facilitar el manejo y análisis de datos estructurados en Python. Fue desarrollada originalmente por Wes McKinney en 2008 y desde entonces se ha convertido en un estándar en el ámbito del análisis de datos.

A diferencia de NumPy, que se centra en arreglos numéricos multidimensionales, Pandas ofrece estructuras de datos más flexibles y expresivas, diseñadas para facilitar el manejo de datos tabulares y heterogéneos. Su principal ventaja es la capacidad de procesar grandes volúmenes de datos de manera eficiente y realizar transformaciones de manera sencilla.

Pandas permite cargar, limpiar, analizar y visualizar datos de diferentes fuentes como archivos CSV, Excel, bases de datos SQL y JSON. Gracias a su integración con otras bibliotecas como Matplotlib y Seaborn, se puede complementar el análisis de datos con visualizaciones detalladas y representativas.

2. PARA QUÉ SE UTILIZA

Pandas es ampliamente utilizada en la manipulación y análisis de datos provenientes de diferentes fuentes, como archivos CSV, Excel, bases de datos SQL y JSON. Entre sus principales usos se encuentran:

1. Carga y limpieza de datos: Permite leer y escribir datos desde múltiples formatos, facilitando la conversión y depuración de información. Esto es clave en la etapa de preprocesamiento de datos antes de su análisis o modelado.
2. Transformación de datos: Ofrece herramientas para modificar la estructura de los datos, incluyendo la eliminación de valores nulos, la normalización y el manejo de datos categóricos, lo que permite adaptar la información a diferentes necesidades.
3. Análisis estadístico: Proporciona métodos para calcular estadísticas descriptivas, identificar patrones y resumir datos mediante funciones como mean(), median(), std(), entre otras.

4. Filtrado y selección de datos: Permite realizar consultas avanzadas sobre conjuntos de datos de manera rápida y eficiente, optimizando procesos en grandes volúmenes de información.
5. Visualización de datos: Se integra con bibliotecas de gráficos como Matplotlib y Seaborn para generar representaciones gráficas de los datos, facilitando su interpretación.
6. Preparación de datos para machine learning: Es utilizada para la manipulación de datasets antes de aplicar modelos de aprendizaje automático, permitiendo normalizar, escalar y transformar características en los datos de entrenamiento.

Gracias a estas funcionalidades, Pandas se ha convertido en una de las herramientas más utilizadas en la ciencia de datos, proporcionando una manera eficiente y flexible de trabajar con información estructurada.

EL TIPO DE DATO SERIE

3. CARACTERÍSTICAS DEL TIPO DE DATO SERIE

Una Serie en Pandas es una estructura de datos unidimensional que puede almacenar cualquier tipo de datos, incluyendo números, texto y fechas. Se asemeja a una columna de una hoja de cálculo o a un array de NumPy, pero con la ventaja de poseer un índice asociado a cada elemento.

Las principales características de una Serie son:

- Posee un índice único que permite acceder a cada elemento.
- Puede contener datos de diferentes tipos, incluyendo valores numéricos y categóricos.
- Admite operaciones matemáticas y de transformación sobre sus elementos, facilitando la manipulación de datos sin necesidad de bucles.
- Se puede crear a partir de listas, diccionarios o arrays de NumPy, lo que permite una integración fluida con otras bibliotecas de Python.
- Permite filtrar y seleccionar datos de manera eficiente mediante índices y condiciones lógicas.

Las Series son fundamentales en el análisis de datos, ya que representan variables individuales dentro de un conjunto más grande y facilitan cálculos estadísticos y exploraciones de datos.

4. CREACIÓN DE UNA SERIE

Se puede crear una Serie en Pandas a partir de diferentes fuentes, como listas, diccionarios o arrays de NumPy.

```
import pandas as pd

# Crear una Serie a partir de una lista
serie = pd.Series([10, 20, 30, 40, 50])
print(serie)
```

También se puede asignar un índice personalizado a cada elemento:

```
serie_indexada = pd.Series([100, 200, 300], index=['a', 'b', 'c'])
print(serie_indexada)
```

En este caso, cada elemento de la Serie está asociado a una etiqueta en lugar de un índice numérico, lo que facilita el acceso y la manipulación de datos.

5. OBTENCIÓN DE DATOS DE UNA SERIE

Una Serie en Pandas permite acceder a elementos específicos mediante su índice. Existen varias formas de extraer datos de una Serie, incluyendo el uso de índices numéricos y etiquetas.

Para obtener un solo elemento, se puede utilizar el índice correspondiente:

```
import pandas as pd
serie = pd.Series([10, 20, 30, 40], index=['a', 'b', 'c', 'd'])
print(serie['b'])  # Output: 20
```

También es posible seleccionar múltiples elementos utilizando listas de índices:

```
print(serie[['a', 'c']])  # Output: Valores en los índices 'a' y 'c'
```

El método .iloc[] permite acceder a los elementos de la Serie mediante su posición numérica:

```
print(serie.iloc[2])  # Output: 30
```

El slicing se puede utilizar para obtener rangos de datos:

```
print(serie[1:3])  # Output: Elementos en las posiciones 1 y 2
```

Si se desea extraer valores que cumplan ciertas condiciones, se puede aplicar filtrado condicional:

```
print(serie[serie > 20])  # Output: Elementos mayores a 20
```

6. OPERACIONES SOBRE UNA SERIE

Las Series permiten realizar operaciones matemáticas y de transformación de manera eficiente. Se pueden realizar cálculos entre Series y escalares, así como aplicar funciones estadísticas y de agregación.

Operaciones aritméticas con escalares

Se pueden realizar operaciones directamente sobre la Serie, y los valores se modificarán elemento por elemento:

```
print(serie * 2)  # Multiplica cada valor por 2
print(serie + 5)  # Suma 5 a cada valor
```

Operaciones entre series

Si se realizan operaciones entre dos Series, estas se aplican a los elementos correspondientes. Si los índices no coinciden, Pandas rellena los valores faltantes con NaN.

```
serie2 = pd.Series([5, 10, 15, 20], index=['a', 'b', 'c', 'd'])
print(serie + serie2)  # Suma elemento a elemento
```

Aplicación de funciones matemáticas

Pandas proporciona funciones matemáticas para operar sobre una Serie sin necesidad de bucles explícitos:

```
print(serie.mean())  # Media de los valores
print(serie.sum())  # Suma de los valores
print(serie.std())  # Desviación estándar
```

También se pueden aplicar funciones personalizadas mediante apply():

```
print(serie.apply(lambda x: x ** 2))  # Eleva al cuadrado cada elemento
```

Las operaciones sobre Series permiten transformar y analizar datos de manera eficiente sin necesidad de realizar iteraciones manuales.

EL TIPO DE DATO DATAFRAME

7. CARACTERÍSTICAS DEL TIPO DE DATO DATAFRAME

Un DataFrame en Pandas es una estructura de datos bidimensional, similar a una hoja de cálculo o a una tabla en una base de datos. A diferencia de una Serie, un DataFrame puede contener múltiples columnas con diferentes tipos de datos.

Las principales características de un DataFrame son:

- Posee etiquetas en las filas y columnas, facilitando la selección y manipulación de datos.
- Permite almacenar datos de diferentes tipos en una misma estructura, como números, texto y fechas.
- Se puede crear a partir de listas, diccionarios, archivos CSV, bases de datos SQL, entre otros.
- Soporta operaciones avanzadas como filtrado, agrupación y transformación de datos, lo que permite analizar grandes volúmenes de información de manera eficiente.
- Se integra con otras bibliotecas para análisis y visualización de datos, como NumPy, Matplotlib y Seaborn.

Los DataFrames son el tipo de dato más utilizado en Pandas debido a su flexibilidad y capacidad para manejar datos estructurados de diferentes fuentes. Su estructura tabular facilita la aplicación de operaciones matemáticas, estadísticas y de transformación de datos.

8. CREACIÓN DE UN DATAFRAME

Un DataFrame se puede crear de varias maneras, incluyendo listas de listas, diccionarios y archivos externos.

```
import pandas as pd

# Creación de un DataFrame desde un diccionario
datos = {
    'Nombre': ['Ana', 'Luis', 'Carlos', 'Sofía'],
    'Edad': [28, 35, 22, 30],
    'Ciudad': ['Madrid', 'Barcelona', 'Valencia', 'Sevilla']
}

df = pd.DataFrame(datos)
print(df)
```

También se puede leer un archivo CSV para crear un DataFrame, lo que facilita la manipulación de datos provenientes de diversas fuentes:

```
df_csv = pd.read_csv('archivo.csv')
print(df_csv.head())
```

Los DataFrames permiten una fácil manipulación y transformación de datos, lo que los hace fundamentales en el análisis de información estructurada. Son utilizados en múltiples aplicaciones, como análisis financiero, estudios científicos, inteligencia de negocios y aprendizaje automático.

9. SELECCIÓN DE FILAS O COLUMNAS EN UN DATAFRAME

Para seleccionar una columna específica, se puede utilizar su nombre como clave:

```
print(df['Nombre'])  # Devuelve la columna 'Nombre'
```

Para seleccionar varias columnas:

```
print(df[['Nombre', 'Edad']])
```

Las filas pueden seleccionarse usando loc[] para etiquetas o iloc[] para posiciones numéricas:

```
print(df.loc[1])  # Devuelve la fila con índice 1
print(df.iloc[2])  # Devuelve la tercera fila (posición 2)
```

Este tipo de selección es útil cuando se trabaja con grandes volúmenes de datos y se requiere acceder a subconjuntos específicos para su análisis o visualización.

10. SELECCIÓN DE ELEMENTOS EN UN DATAFRAME

Para acceder a un elemento específico, se combinan loc[] o iloc[] con nombres de columnas:

```
print(df.loc[0, 'Ciudad'])  # Ciudad de la primera fila
print(df.iloc[2, 1])  # Elemento en la tercera fila y segunda columna
```

También se pueden seleccionar subconjuntos de datos, lo que facilita el análisis y la manipulación de información relevante:

```
print(df.loc[0:2, ['Nombre', 'Edad']])  # Primeras tres filas
```

11. SELECCIÓN CONDICIONAL EN UN DATAFRAME

Los DataFrames permiten filtrar datos con condiciones lógicas, lo que resulta muy útil para análisis exploratorio y segmentación de datos.

```
print(df[df['Edad'] > 25])  # Filtra personas mayores de 25 años
```

También se pueden combinar condiciones con operadores & y |, lo que permite realizar consultas más complejas:

```
print(df[(df['Edad'] > 25) & (df['Ciudad'] == 'Madrid')])  # Mayores de 25 en Madrid
```

La selección condicional es una de las funcionalidades más utilizadas en Pandas, ya que permite extraer información relevante de grandes conjuntos de datos.

MÉTODOS BÁSICOS DE EXPLORACIÓN

12. MÉTODOS BÁSICOS DE EXPLORACIÓN (HEAD, TAIL, INFO, DESCRIBE)

Los DataFrames incluyen métodos para obtener información rápida sobre los datos:

```
print(df.head(3))  # Muestra las primeras 3 filas
print(df.tail(2))  # Muestra las últimas 2 filas
print(df.info())  # Información general del DataFrame
print(df.describe())  # Estadísticas generales del DataFrame
```

Estos métodos permiten entender la estructura y contenido de los datos antes de realizar análisis más detallados. info() proporciona información sobre los tipos de datos y valores nulos, mientras que describe() devuelve estadísticas generales sobre las columnas numéricas.

13. MÉTODOS BÁSICOS DE SUMARIZACIÓN (MIN, MAX, COUNT, MEAN, MEDIAN, SUM)

Para obtener resúmenes estadísticos de las columnas numéricas, Pandas ofrece métodos que facilitan la exploración y comprensión de los datos:

```
print(df['Edad'].min())  # Edad mínima
print(df['Edad'].max())  # Edad máxima
print(df['Edad'].count())  # Número de elementos
print(df['Edad'].mean())  # Promedio
print(df['Edad'].median())  # Mediana
print(df['Edad'].sum())  # Suma total
```

Estos métodos son clave en el análisis de datos, ya que permiten identificar tendencias y patrones de manera rápida y eficiente.

14. MÉTODOS UNIQUE, NUNIQUE, VALUE_COUNTS

Para analizar valores únicos y sus frecuencias, Pandas proporciona métodos que ayudan a explorar la distribución de los datos:

```
print(df['Ciudad'].unique())  # Valores únicos en la columna 'Ciudad'
print(df['Ciudad'].nunique())  # Cantidad de valores únicos
print(df['Ciudad'].value_counts())  # Frecuencia de cada valor
```

Estos métodos son útiles para comprender la diversidad y composición de los datos, lo que facilita la toma de decisiones basada en información estructurada.

Cierre ✍️

Pandas es una de las herramientas más poderosas dentro del ecosistema de Python para la manipulación y análisis de datos. A lo largo de este manual, se han abordado sus características esenciales, incluyendo la creación de estructuras de datos como Series y DataFrames, la selección y filtrado de información, y la exploración y agregación de datos mediante distintos métodos. Su flexibilidad y facilidad de uso han hecho que se convierta en una herramienta indispensable para analistas, científicos de datos y desarrolladores.

El conocimiento adquirido sobre Pandas permite gestionar grandes volúmenes de datos de manera eficiente, facilitando la transformación, limpieza y análisis de información con un código conciso y optimizado. Su integración con bibliotecas como NumPy, Matplotlib, Seaborn y Scikit-learn amplía aún más sus aplicaciones, permitiendo el desarrollo de flujos de trabajo completos para la exploración y modelado de datos.

Referencias📚

1. Pandas Documentation - https://pandas.pydata.org/docs/
2. Python for Data Analysis - Wes McKinney (2017).
3. Machine Learning with Python - Andreas C. Müller, Sarah Guido (2017).
4. Hands-On Data Analysis with Pandas - Stefanie Molin (2019).

¡Muchas gracias!

Nos vemos en la próxima lección