DATA WRANGLING

M3: Fundamentos de programación en Java

|AE5: APLICAR TÉCNICAS DE DATA WRANGLING PARA LA MANIPULACIÓN, ENRIQUECIMIENTO Y TRANSFORMACIÓN DE DATOS EN UNA ESTRUCTURA DE TIPO SERIE O DATAFRAME PARA LA RESOLUCIÓN DE UN PROBLEMA.

alkemy

Introducción👩‍💻

En el análisis de datos, uno de los mayores desafíos es garantizar que la información esté limpia, estructurada y lista para su uso en procesos analíticos o modelos predictivos. Data Wrangling es el conjunto de técnicas utilizadas para transformar datos crudos en conjuntos organizados y funcionales, permitiendo su correcta manipulación y exploración.

Las fuentes de datos pueden presentar inconsistencias, valores faltantes, formatos incorrectos y duplicaciones, lo que puede distorsionar los resultados y dificultar la toma de decisiones. Utilizando Pandas, una de las bibliotecas más poderosas en Python, es posible realizar operaciones avanzadas como filtrado, agrupación, reestructuración y optimización de DataFrames, facilitando así el trabajo con grandes volúmenes de información.

Este proceso es clave en áreas como ciencia de datos, inteligencia de negocios y análisis financiero, donde la calidad de los datos impacta directamente en la precisión de los modelos y reportes. Con Data Wrangling, los datos se convierten en activos confiables, asegurando que sean interpretados y utilizados de manera eficiente.

Aprendizaje esperado

Cuando finalices la lección serás capaz de:

1. **Comprender el concepto y la importancia de Data Wrangling** en el proceso de análisis de datos.
2. **Aplicar herramientas de Pandas** para limpiar, transformar y optimizar datasets de diversas fuentes.
3. **Manejar valores nulos y registros duplicados**, garantizando la calidad y coherencia de los datos.
4. **Utilizar funciones avanzadas de filtrado, agregación y manipulación de columnas** en un **DataFrame**.
5. **Reestructurar conjuntos de datos mediante pivotado y fusiones**, mejorando la accesibilidad y el análisis de información.
6. **Exportar los datos transformados en formatos adecuados**, asegurando su disponibilidad para análisis posteriores o integraciones con otras herramientas.

DATA WRANGLING

1. QUÉ ES DATA WRANGLING

El Data Wrangling es el proceso de transformar y estructurar datos en un formato adecuado para el análisis. Este proceso incluye la limpieza, normalización y reestructuración de los datos para que sean más accesibles y comprensibles. Su objetivo principal es mejorar la calidad de los datos, reducir errores y asegurar que se encuentren en un formato adecuado para su uso en modelos analíticos.

El Data Wrangling no solo implica eliminar valores erróneos, sino también transformar los datos en un formato adecuado para el análisis. Algunas de las tareas más importantes dentro del Data Wrangling incluyen:

- Conversión de tipos de datos: Ajustar los datos al tipo correcto para su procesamiento.
- Corrección de valores erróneos: Identificar y modificar valores incorrectos.
- Estandarización de datos: Unificar unidades de medida, formatos de fecha y convenciones de nombres.
- Detección y eliminación de valores atípicos: Identificar valores extremos que puedan sesgar el análisis.
- Normalización y escalado: Ajustar los valores para que estén en un rango comparable.

Ejemplo de transformación de datos en Pandas:

```python
import pandas as pd

# Crear un DataFrame con datos desorganizados
datos = {'Fecha': ['2023-01-10', '2023/02/15', 'March 20, 2023', '10-04-2023'],
         'Precio': ['$1,200', '1.500$', 'USD 1,350', '€1.400']}

df = pd.DataFrame(datos)

# Normalizar formatos de fecha y eliminar caracteres especiales en el precio
df['Fecha'] = pd.to_datetime(df['Fecha'], errors='coerce')
df['Precio'] = df['Precio'].replace({'\$': '', '€': '', 'USD ': '', ',': ''},
                                    regex=True).astype(float)

print(df)
```

En este ejemplo, se corrigen formatos de fecha inconsistentes y se convierten los precios a valores numéricos.

2. PRINCIPALES TAREAS DE DATA WRANGLING

El Data Wrangling implica diversas tareas esenciales que permiten mejorar la calidad y estructura de los datos. Algunas de las más importantes incluyen:

1. Ordenamiento y manipulación de datos: Reorganización de los datos para facilitar su análisis.
2. Eliminación de registros duplicados: Identificación y eliminación de registros repetidos que pueden afectar la calidad del análisis.
3. Reemplazo de valores: Sustitución de valores erróneos o inconsistentes para estandarizar la información.
4. Discretización y binning: Agrupación de valores continuos en categorías para facilitar la interpretación.
5. Enriquecimiento de datos: Aplicación de funciones y transformaciones para agregar valor a los datos.
6. Manipulación de la estructura de un DataFrame: Ajuste del formato de los datos para mejorar su usabilidad.

ORDENAMIENTO Y MANIPULACIÓN DE DATOS

3. MUESTREOS ALEATORIOS Y PERMUTACIÓN DE LA DATA

El muestreo aleatorio es una técnica utilizada para seleccionar subconjuntos de datos de manera aleatoria. Esto es útil cuando se trabaja con grandes volúmenes de datos y se desea obtener una muestra representativa para el análisis. La permutación, por otro lado, permite reorganizar aleatoriamente los datos en un orden diferente, lo cual es útil en validaciones cruzadas o en experimentos donde se desea eliminar el sesgo del orden original de los datos.

Ejemplo de muestreo aleatorio en Pandas:

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

# Crear un DataFrame con datos de ejemplo
df = pd.DataFrame({'ID': np.arange(1, 101), 'Valor': np.random.randint(1, 100, 100)})

# Obtener una muestra aleatoria de 10 filas
df_muestra = df.sample(n=10, random_state=42)
print(df_muestra)
```

También es posible realizar una permutación aleatoria de los datos, es decir, reorganizar el orden de las filas:

```python
# Reordenar aleatoriamente todas las filas
df_permutado = df.sample(frac=1, random_state=42)
print(df_permutado.head())
```

Este proceso es útil para mezclar datos antes de realizar entrenamientos de modelos de machine learning o validaciones cruzadas.

4. ORDENAMIENTO DE LA DATA

El ordenamiento de los datos permite organizar la información según valores específicos. En Pandas, se puede ordenar de forma ascendente o descendente con sort_values().

Ejemplo de ordenamiento por una columna:

```python
# Ordenar por la columna 'Valor' de forma ascendente
df_ordenado = df.sort_values(by='Valor', ascending=True)
print(df_ordenado.head())
```

También es posible ordenar por múltiples columnas:

```python
# Ordenar primero por 'Valor' y luego por 'ID'
df_ordenado = df.sort_values(by=['Valor', 'ID'], ascending=[True, False])
```

Este método es útil cuando se necesita priorizar ciertos criterios dentro de un conjunto de datos.

5. DETECCIÓN Y ELIMINACIÓN DE REGISTROS DUPLICADOS

Los registros duplicados pueden generar sesgos en el análisis. Pandas ofrece métodos sencillos para detectarlos y eliminarlos.

Ejemplo de detección de duplicados:

```
# Identificar registros duplicados
duplicados = df.duplicated()
print(duplicados.sum())  # Número de registros duplicados
```

Para eliminarlos:

```
# Eliminar registros duplicados
df_sin_duplicados = df.drop_duplicates()
print(df_sin_duplicados.shape)
```

Esto asegura que el conjunto de datos contenga únicamente información única y sin redundancias.

6. REEMPLAZO DE VALORES

En ocasiones, los conjuntos de datos contienen valores incorrectos, mal escritos o inconsistentes que deben corregirse. Pandas ofrece funciones para reemplazar valores de manera eficiente.

Ejemplo de reemplazo de valores:

```
# Crear un DataFrame con valores erróneos
df = pd.DataFrame({'Producto': ['Laptop', 'Tablet', 'Celular', 'Lapto', 'Tablet'],
                   'Precio': [1000, 500, 800, 950, 520]})

# Reemplazar valores incorrectos
df['Producto'].replace({'Lapto': 'Laptop'}, inplace=True)
print(df)
```

También es posible usar expresiones regulares para reemplazos más avanzados:

```
# Reemplazar valores que contengan un patrón específico
df['Producto'] = df['Producto'].str.replace(r'Lapto', 'Laptop', regex=True)
```

El reemplazo de valores es fundamental para estandarizar los datos y evitar errores en el análisis posterior.

7. DISCRETIZACIÓN Y BINNING

La discretización es un proceso que convierte valores continuos en categorías o intervalos (bins). Esto es útil cuando se quiere agrupar datos numéricos en rangos específicos.

Ejemplo de binning en Pandas:

```
# Crear un DataFrame con datos continuos
df = pd.DataFrame({'Edad': [22, 25, 30, 35, 40, 45, 50, 55, 60]})

# Definir los rangos (bins)
bins = [20, 30, 40, 50, 60]
etiquetas = ['Joven', 'Adulto Joven', 'Adulto', 'Mayor']

# Aplicar discretización
df['Grupo Edad'] = pd.cut(df['Edad'], bins=bins, labels=etiquetas)
print(df)
```

Este método facilita la segmentación y agrupación de datos en análisis exploratorios y visualizaciones.

ENRIQUECIMIENTO DE LA DATA

8. UTILIZACIÓN DE FUNCIONES Y MAPEOS

El enriquecimiento de datos implica transformar la información existente para agregar valor al conjunto de datos. Una técnica clave es el uso de funciones y mapeos para modificar columnas o generar nuevas variables en un DataFrame.

Pandas ofrece el método .map() para aplicar una función a cada elemento de una serie. Este método es útil cuando se necesita reemplazar valores en una columna o realizar conversiones simples.

Ejemplo de uso de .map() en una columna categórica:

```
import pandas as pd

# Crear un DataFrame
df = pd.DataFrame({'Producto': ['Laptop', 'Tablet', 'Celular'],
                   'Categoría': ['Electrónica', 'Electrónica', 'Comunicación']})

# Crear un diccionario de mapeo
mapeo = {'Electrónica': 'Tecnología', 'Comunicación': 'Telecomunicaciones'}

# Aplicar mapeo
df['Nueva Categoría'] = df['Categoría'].map(mapeo)
print(df)
```

El método .map() permite transformar valores en base a reglas definidas, facilitando la categorización y estandarización de datos.

9. APLICACIÓN DE FUNCIONES DENTRO DE UNA SERIE DE DATOS (APPLY)

Cuando se requieren transformaciones más complejas, el método .apply() es una opción poderosa. Permite aplicar funciones personalizadas a cada valor de una serie.

Ejemplo de .apply() en una columna numérica:

```
# Crear un DataFrame
df = pd.DataFrame({'Precio': [1000, 500, 800, 1500]})

# Función para calcular impuestos
def calcular_impuesto(precio):
    return precio * 1.21  # Aplicar 21% de IVA

# Aplicar la función a la columna 'Precio'
df['Precio con IVA'] = df['Precio'].apply(calcular_impuesto)
print(df)
```

Este método es útil cuando se necesita realizar cálculos en cada valor de una columna.

10. APLICACIÓN DE FUNCIONES DENTRO DE UN DATAFRAME (APPLY)

El método .apply() también puede utilizarse en un DataFrame completo para aplicar funciones a filas o columnas específicas.

Ejemplo de .apply() en múltiples columnas:

```
# Crear un DataFrame
df = pd.DataFrame({'Nombre': ['Ana', 'Luis', 'Carlos'],
                   'Edad': [25, 30, 28],
                   'Salario': [50000, 54000, 58000]})

# Aplicar una función a varias columnas
def categorizar_fila(fila):
    if fila['Salario'] > 55000:
        return 'Alto'
    else:
        return 'Medio'

# Aplicar función a cada fila
df['Nivel Salarial'] = df.apply(categorizar_fila, axis=1)
print(df)
```

Este método permite realizar transformaciones en base a múltiples columnas, enriqueciendo los datos con nueva información derivada.

11. UTILIZACIÓN DE EXPRESIONES LAMBDA EN LA APLICACIÓN DE FUNCIONES

Las expresiones lambda permiten definir funciones en una sola línea y aplicarlas a una columna o DataFrame sin necesidad de definir funciones separadas.

Ejemplo de expresión lambda para calcular descuentos:

```
# Crear un DataFrame
df = pd.DataFrame({'Precio': [1000, 500, 800, 1500]})

# Aplicar descuento del 10%
df['Precio con Descuento'] = df['Precio'].apply(lambda x: x * 0.90)
print(df)
```

Las funciones lambda son una opción eficiente cuando se requieren transformaciones simples sin necesidad de escribir funciones extensas.

MANIPULACIÓN DE LA ESTRUCTURA DE UN DATAFRAME

12. AGREGAR O ELIMINAR COLUMNAS

En Pandas, se pueden agregar o eliminar columnas fácilmente utilizando las funciones assign(), drop() y la asignación directa de valores.

Ejemplo de agregar una nueva columna:

```
# Crear un DataFrame
df = pd.DataFrame({'Nombre': ['Ana', 'Luis', 'Carlos'],
                   'Edad': [25, 30, 28]})

# Agregar una nueva columna
df['Ciudad'] = ['Madrid', 'Barcelona', 'Sevilla']
print(df)
```

Para eliminar una columna:

```
# Eliminar la columna 'Ciudad'
df = df.drop(columns=['Ciudad'])
print(df)
```

13. REDIMENSIÓN DE UN DATAFRAME

El tamaño de un DataFrame puede modificarse mediante la selección de filas y columnas específicas.

Ejemplo de selección de un subconjunto de columnas:

```
# Seleccionar solo las columnas 'Nombre' y 'Edad'
df_reducido = df[['Nombre', 'Edad']]
print(df_reducido)
```

También es posible seleccionar filas específicas mediante índices:

```
# Seleccionar las primeras dos filas
df_subset = df.iloc[:2]
print(df_subset)
```

14. CONVERTIR EL TIPO DE DATO DE UNA COLUMNA

A veces, es necesario cambiar el tipo de datos de una columna para su correcta manipulación.

Ejemplo de conversión de tipos:

```
# Crear un DataFrame con datos en formato de texto
df = pd.DataFrame({'ID': ['1', '2', '3'], 'Precio': ['1000', '500', '800']})

# Convertir a tipos numéricos
df['ID'] = df['ID'].astype(int)
df['Precio'] = df['Precio'].astype(float)
print(df.dtypes)
```

15. DEFINIR Y RESETEAR ÍNDICES

El índice en Pandas se puede definir y modificar para mejorar la organización de los datos.

Ejemplo de definición de índice:

```
# Definir la columna 'ID' como índice
df.set_index('ID', inplace=True)
print(df)
```

Para resetear el índice:

```
# Resetear el índice
df.reset_index(inplace=True)
print(df)
```

16. RENOMBRAR COLUMNAS E ÍNDICES

Pandas permite renombrar columnas e índices utilizando .rename().

Ejemplo de renombrar columnas:

```
# Renombrar columnas
df = df.rename(columns={'Precio': 'Costo'})
print(df)
```

17. REMOVER DUPLICADOS

Los datos pueden contener registros duplicados que deben eliminarse para evitar errores en el análisis.

Ejemplo de eliminación de duplicados:

```
# Crear un DataFrame con datos duplicados
df = pd.DataFrame({'Nombre': ['Ana', 'Luis', 'Ana', 'Carlos'],
                   'Edad': [25, 30, 25, 28]})

# Eliminar duplicados
df_sin_duplicados = df.drop_duplicates()
print(df_sin_duplicados)
```

Cierre ✍️

El Data Wrangling es un proceso esencial en el análisis de datos, ya que permite transformar conjuntos de datos crudos en información estructurada, limpia y lista para el análisis. A lo largo de este manual, hemos explorado diferentes técnicas que facilitan la manipulación, enriquecimiento y optimización de los datos utilizando la biblioteca Pandas en Python.

Desde la limpieza de registros hasta la reestructuración de DataFrames, cada técnica presentada contribuye a mejorar la calidad y confiabilidad de los datos. La correcta aplicación de estos métodos no solo optimiza el análisis estadístico y la modelización de datos, sino que también garantiza la precisión de los resultados y la toma de decisiones informadas.

La capacidad de transformar y preparar datos de manera eficiente es una de las habilidades más valoradas en el mundo del análisis de datos y la ciencia de datos. Dominar las herramientas de Data Wrangling permite a los analistas e investigadores desarrollar procesos de análisis más sólidos, reducir errores y automatizar tareas repetitivas.

Referencias📚

1. Pandas Documentation - https://pandas.pydata.org/docs/
2. Python for Data Analysis - Wes McKinney (2017).
3. Data Science from Scratch - Joel Grus (2019).
4. Machine Learning with Python - Andreas C. Müller, Sarah Guido (2017).
5. Hands-On Data Analysis with Pandas - Stefanie Molin (2019).
6. The Elements of Statistical Learning - Trevor Hastie, Robert Tibshirani, Jerome Friedman (2009).
7. An Introduction to Statistical Learning - Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani (2013).

¡Muchas gracias!

Nos vemos en la próxima lección