SESIÓN LIMPIEZA Y TRANSFORMACIÓN DE DATOS CON PANDAS

CONTENIDOS:

- Qué es un valor perdido.
    - Identificación de valores perdidos.
    - Filtrado de la data perdida.
    - Imputación de datos.
    - Imputación de valores cualitativos.
- Qué es un outlier.
    - Detección y filtrado de outliers.
- Qué es data wrangling.
    - Principales tareas de data wrangling.
- Ordenamiento y manipulación de datos:
    - Muestreos aleatorios.
    - Permutación de la data.
    - Ordenamiento de la data.
- Detección y eliminación de registros duplicados.
- Reemplazo de valores.
- Discretización y binning (contenido opcional)
- Enriquecimiento de la data:
    - Utilización de funciones y mapeos.
    - Aplicación de funciones dentro de una serie de datos (apply).
    - Aplicación de funciones dentro de un dataframe (apply).
    - Utilización de expresiones lambda en la aplicación de funciones.
- Manipulación de la estructura de un dataframe:

- Agregar o eliminar columnas.
- Redimensión de un dataframe.
- Convertir el tipo de dato de una columna.
- Definir y resetear índices.
- Renombrar columnas e índices.
- Remover duplicados.

QUÉ ES UN VALOR PERDIDO

La limpieza y transformación de datos, también conocida como data wrangling, es una etapa fundamental en cualquier proceso de análisis de datos. Los datos en bruto suelen contener errores, valores faltantes, duplicados o inconsistencias que pueden afectar los resultados del análisis.

¿Qué es un Valor Perdido?

Un valor perdido (o missing value) es un dato ausente en un conjunto de datos. En Pandas, los valores perdidos se representan con NaN (Not a Number).

Los valores perdidos pueden surgir debido a:

- Errores en la recolección de datos (por ejemplo, sensores defectuosos).
- Registros incompletos (campos no ingresados en encuestas).
- Conversión de formatos (al importar datos de distintas fuentes).
- Eliminación accidental de valores.

Identificación de Valores Perdidos

Para detectar valores perdidos en un DataFrame, se usa isnull(), que devuelve un DataFrame booleano donde True indica valores perdidos:

```python
import pandas as pd

df = pd.DataFrame({"Nombre": ["Ana", "Carlos", "Luis", None],
                   "Edad": [25, 30, None, 40]})

print(df.isnull())  # Muestra un DataFrame con True donde hay valores nulos
print(df.isnull().sum())  # Cuenta los valores nulos por columna
```

Ilustración 1 Ejemplo isnull()

Otra alternativa es notnull(), que muestra los valores NO nulos.

Filtrado de la Data Perdida

Si los valores perdidos son pocos y no afectan significativamente el análisis, pueden eliminarse con dropna():

```python
df_limpio = df.dropna()  # Elimina todas las filas con al menos un NaN
df_sin_columnas_nulas = df.dropna(axis=1)  # Elimina columnas con NaN
```

Ilustración 2 Ejemplo dropna()

- axis=0: Elimina filas (por defecto).
- axis=1: Elimina columnas.
- how="any": Elimina si hay al menos un NaN en la fila/columna.
- how="all": Solo elimina si todas las celdas son NaN.

Imputación de Datos

En lugar de eliminar datos, podemos reemplazar los valores perdidos con fillna():

```
df["Edad"].fillna(df["Edad"].mean(), inplace=True)  # Rellena con la media
df["Nombre"].fillna("Desconocido", inplace=True)  # Rellena valores de texto
```

Ilustración 3 Ejemplo fillna()

Imputación de Valores Cualitativos

Para datos categóricos, se puede usar la moda (valor más frecuente):

```
df["Categoría"].fillna(df["Categoría"].mode()[0], inplace=True)
```

Ilustración 4 Ejemplo imputación de valores cualitativos

QUÉ ES UN OUTLIER

Un outlier o valor atípico es un dato que se aleja significativamente del resto de los valores en un conjunto de datos. Puede deberse a:

- Errores en la medición.
- Variabilidad natural en los datos.
- Problemas en la entrada de datos.

Los outliers pueden distorsionar análisis estadísticos y modelos de Machine Learning, por lo que es importante detectarlos y decidir si deben eliminarse o ajustarse.

Detección y Filtrado de Outliers

Este método se utiliza para identificar outliers analizando la dispersión de los datos a través de los cuartiles. Un valor se considera atípico si se encuentra fuera del rango esperado, determinado por el rango intercuartílico (IQR).

El IQR se calcula como la diferencia entre el tercer cuartil (Q3) y el primer cuartil (Q1):

$$IQR = Q3 - Q1$$

Ilustración 5 función para calcular IQR

Un dato se considera un outlier si está fuera del siguiente rango:

$$Q1 - 1.5 \times IQR \quad \text{o} \quad Q3 + 1.5 \times IQR$$

Ilustración 6 Función para identificar outlier

¿Qué son los cuartiles?

Los cuartiles son valores que dividen un conjunto de datos ordenados en cuatro partes iguales, cada una con aproximadamente el 25% de los datos. Se utilizan en estadística para analizar la distribución de los datos y detectar valores atípicos.

Principales cuartiles:

- Primer cuartil (Q1): Es el valor por debajo del cual se encuentra el 25% de los datos. También se conoce como percentil 25.
- Segundo cuartil (Q2): Es la mediana, que divide el conjunto en dos mitades, dejando el 50% de los datos por debajo.

- Tercer cuartil (Q3): Es el valor por debajo del cual está el 75% de los datos. También se conoce como percentil 75.

Este método es ampliamente utilizado en análisis de datos para detectar valores extremos y mejorar la calidad de los análisis estadísticos.

Ejemplo en Python

```python
import numpy as np

Q1 = df["Edad"].quantile(0.25)  # Primer cuartil
Q3 = df["Edad"].quantile(0.75)  # Tercer cuartil
IQR = Q3 - Q1

# Filtramos los datos dentro del rango permitido
df_filtrado = df[(df["Edad"] >= Q1 - 1.5*IQR) & (df["Edad"] <= Q3 + 1.5*IQR)]
```

Ilustración 7 Ejemplo de detección y filtrado de outliers

QUÉ ES DATA WRANGLING

El data wrangling es el proceso de transformar, limpiar y estructurar datos en un formato adecuado para el análisis.

Principales Tareas de Data Wrangling

- Limpieza de datos: Eliminación de valores faltantes, duplicados y errores.
- Transformación de datos: Cambio de formatos, creación de nuevas variables.
- Integración de datos: Combinación de diferentes fuentes de datos.

ORDENAMIENTO Y MANIPULACIÓN DE DATOS

Se refiere a las técnicas utilizadas para organizar, modificar y estructurar conjuntos de datos con el objetivo de facilitar su análisis y procesamiento. Estas operaciones son esenciales en la ciencia de datos, la estadística y el aprendizaje automático, ya que permiten extraer información útil de los datos de manera eficiente.

Muestreos Aleatorios

Para seleccionar una muestra aleatoria de un DataFrame, usamos sample(), lo que permite extraer una fracción o una cantidad específica de datos de manera aleatoria. Esto es útil en análisis exploratorio y modelado estadístico.

```
df_muestra = df.sample(n=5)  # Selecciona 5 registros aleatorios
```

Ilustración 8 Ejemplo muestreos aleatorios

Permutación de la Data

Para reordenar aleatoriamente las filas de un DataFrame, usamos np.random.permutation(). Esto es útil cuando queremos reorganizar los datos antes de aplicar técnicas de aprendizaje automático o dividir el conjunto en entrenamiento y prueba.

```
df = df.iloc[np.random.permutation(len(df))]
```

Ilustración 9 Ejemplo permutación de la data

Ordenamiento de la data

Para ordenar el DataFrame según una o más columnas, usamos sort_values():

```
df_ordenado = df.sort_values(by="Edad", ascending=False)
```

Ilustración 10 Ejemplo ordenamiento de la data

DETECCIÓN Y ELIMINACIÓN DE REGISTROS DUPLICADOS

Los registros duplicados son filas en un DataFrame que tienen exactamente los mismos valores en todas las columnas o en un subconjunto de columnas. Estos duplicados pueden surgir por errores en la recolección de datos o durante la integración de múltiples fuentes de datos.

Identificación de Duplicados

Para detectar registros duplicados en un DataFrame, usamos duplicated(), que devuelve True para filas duplicadas:

```
df = pd.DataFrame({
    'A': [1, 2, 2, 3],
    'B': [4, 5, 5, 6]
})
print(df.duplicated())
```

Ilustración 11 Ejemplo duplicated()

Para contar los duplicados en una columna específica:

```
df["Nombre"].duplicated().sum()
```

Ilustración 12 Ejemplo para contar duplicados en una columna especifica

Eliminación de Duplicados

Para eliminar duplicados, usamos drop_duplicates():

```
df_sin_duplicados = df.drop_duplicates()
print(df_sin_duplicados)
```

Ilustración 13 Ejemplo drop_duplicates()

Parámetros de drop_duplicates()

- subset: Especifica las columnas a considerar para identificar duplicados.

```
df_sin_duplicados = df.drop_duplicates(subset=['A'])
print(df_sin_duplicados)
```

Ilustración 14 Ejemplo subset

- keep: Controla qué duplicados mantener
  - first: Mantiene la primera ocurrencia (por defecto).
  - last: Mantiene la última ocurrencia.
  - False: Elimina todas las filas duplicadas.

```
df_sin_duplicados = df.drop_duplicates(keep='last')
print(df_sin_duplicados)
```

Ilustración 15 Ejemplo de keep

REEMPLAZO DE VALORES

En el análisis y manipulación de datos, a menudo es necesario modificar valores en un conjunto de datos. Con la función replace() de Pandas, podemos reemplazar valores específicos en un DataFrame o Serie de manera sencilla.

Podemos reemplazar valores específicos con replace():

```
df["Categoría"] = df["Categoría"].replace({"M": "Masculino", "F": "Femenino"})
```

Ilustración 16 Ejemplo replace()

También podemos reemplazar múltiples valores:

```
df.replace(["Desconocido", "N/A"], np.nan, inplace=True)
```

Ilustración 17 Ejemplo de reemplazo de múltiples valores

DISCRETIZACIÓN Y BINNING

Discretización y binning son técnicas utilizadas para transformar variables continuas (números que pueden tomar cualquier valor dentro de un rango) en variables discretas (valores finitos o categorías). Esto es especialmente útil en el análisis de datos, donde simplificar los datos puede facilitar su comprensión, visualización y modelado.

Discretización:

Es el proceso de convertir una variable continua en una variable discreta, es decir, transformar un conjunto de valores numéricos continuos en categorías o intervalos. Por ejemplo, en lugar de trabajar con edades exactas (25.5, 30.1, 40.2), podríamos agruparlas en rangos, como "18-30", "31-40", "41-50", etc. Este proceso facilita el análisis de datos al reducir la variabilidad de los valores continuos.

Binning (o agrupar en "bins"):

Es un tipo específico de discretización que consiste en dividir los valores de una variable continua en intervalos o "bins". Estos intervalos pueden ser de igual tamaño o basarse en criterios específicos. Cada valor de la variable continua se asigna al intervalo que le corresponde. Por ejemplo, si tenemos una lista de sueldos y los agrupamos en "bajos", "medios" y "altos", estamos realizando un binning.

Ventajas de la discretización y el binning:

- **Simplificación:** Reduce la complejidad de los datos continuos y hace que sean más fáciles de interpretar.
- **Facilita el análisis:** Algunos algoritmos de aprendizaje automático y análisis de datos funcionan mejor con variables discretas que con continuas.
- **Detección de patrones:** Puede ayudar a descubrir patrones o tendencias en los datos que no son evidentes cuando los datos son continuos.

Ejemplo de binning:

Supongamos que tenemos una lista de edades: [18, 25, 34, 42, 57, 63, 72]. Si queremos agrupar estas edades en intervalos de 10 años, podemos crear los siguientes bins:

- "18-28"
- "29-38"
- "39-48"
- "49-58"
- "59-68"
- "69+"

En este caso, estamos aplicando binning para discretizar las edades en intervalos de 10 años.

Sintaxis básica:

```
pd.cut(x, bins, labels=False, include_lowest=False)
```

Ilustración 18 Sintaxis básica de pd.cut()

- x: los datos numéricos que se van a discretizar.
- bins: el número de intervalos o los límites de los intervalos en los que se dividirán los datos.
- labels: si se establecen etiquetas, asignará una etiqueta a cada intervalo. Si se establece en False, devolverá los intervalos en formato numérico.
- include_lowest: si se establece en True, incluirá el límite inferior del primer intervalo.

Ejemplo en Python:

Para agrupar valores numéricos en intervalos, usamos pd.cut():

```
df["Rango_Edad"] = pd.cut(df["Edad"], bins=[0, 18, 35, 60, 100], labels=["Joven", "Adulto Joven", "Adulto", "Mayor"])
```

Ilustración 19 Ejemplo pd.cut()

ENRIQUECIMIENTO DE LA DATA

El enriquecimiento de datos consiste en mejorar un conjunto de datos mediante la aplicación de funciones, transformaciones y combinaciones con otras fuentes de información.

Aplicación de Funciones

En Pandas, podemos aplicar funciones a los datos de forma eficiente mediante el método apply(), que nos permite ejecutar una función sobre cada elemento de una columna o fila.

Ejemplo básico de apply() en una Serie:

```python
import pandas as pd

df = pd.DataFrame({"Nombre": ["Ana", "Carlos", "Luis"],
                   "Edad": [25, 30, 40]})

# Aplicar una función para calcular la edad en 10 años
df["Edad_Futura"] = df["Edad"].apply(lambda x: x + 10)
print(df)
```

Ilustración 20 Ejemplo básico de apply() en una serie

Aplicación de Funciones en un DataFrame

Podemos aplicar funciones personalizadas sobre filas o columnas utilizando apply().

Ejemplo:

```python
def clasificar_edad(edad):
    if edad < 30:
        return "Joven"
    else:
        return "Adulto"

df["Categoria_Edad"] = df["Edad"].apply(clasificar_edad)
print(df)
```

Ilustración 21 Ejemplo utilizando apply()

Expresiones Lambda

Las expresiones lambda son funciones anónimas que permiten realizar operaciones rápidas sin necesidad de definir una función tradicional. Son especialmente útiles en apply().

Ejemplo:

```
df["Nombre_Mayus"] = df["Nombre"].apply(lambda x: x.upper())
print(df)
```

Ilustración 22 Ejemplo expresiones lambda

MANIPULACIÓN DE LA ESTRUCTURA DE UN DATAFRAME

Pandas ofrece herramientas para modificar la estructura de un DataFrame, lo que permite adaptar los datos a diferentes formatos de análisis.

Agregar o Eliminar Columnas

Podemos agregar una nueva columna asignando valores directamente:

```
df["Salario"] = [5000, 6000, 7000]  # Agregar columna
```

Ilustración 23 Ejemplo para agregar una nueva columna

Para eliminar columnas, usamos drop():

```
df = df.drop(columns=["Salario"])  # Eliminar columna
```

Ilustración 24 Ejemplo de drop()

Redimensión de un dataframe

La función melt() permite transformar datos anchos en formato largo, útil para análisis estadísticos y visualización.

Ejemplo:

```
df_melt = df.melt(id_vars=["Nombre"], value_vars=["Edad", "Edad_Futura"], var_name="Variable", value_name="Valor")
print(df_melt)
```

Ilustración 25 Ejemplo de la función melt()

Convertir el tipo de dato de una columna

En ocasiones, es necesario cambiar el tipo de dato de una columna para optimizar el rendimiento o asegurar la compatibilidad con otras herramientas. La función astype() permite realizar esta conversión, ajustando el tipo de dato de una columna según sea necesario.

Ejemplo:

```
df["Edad"] = df["Edad"].astype(float)  # Convertir a flotante
```

Ilustración 26 Ejemplo de la función astype()

Definir y resetear índices

Podemos definir una columna como índice con set_index() y restaurar el índice original con reset_index().

Ejemplo:

```
df = df.set_index("Nombre")  # Usar "Nombre" como índice
df = df.reset_index()  # Volver al índice original
```

Ilustración 27 Ejemplo de set_index() y reset_index()

Renombrar columnas e índices

Para mejorar la legibilidad, podemos renombrar columnas o índices con rename().

Ejemplo:

```
df = df.rename(columns={"Edad": "Años", "Categoria_Edad": "Grupo_Etario"})
print(df)
```

Ilustración 28 Ejemplo rename()

Remover duplicados

Los datos pueden contener filas duplicadas que deben ser eliminadas con drop_duplicates().

Ejemplo:

```
df = df.drop_duplicates()
```

Ilustración 29 Ejemplo drop_duplicates()