# **Data Wrangling**

El Data Wrangling es el proceso de limpiar, transformar y estructurar los datos para su uso en el análisis. Es la fase en la que se realizan las tareas necesarias para preparar los datos para su análisis posterior. Esto incluye la eliminación de datos duplicados, la transformación de datos en diferentes formatos, el relleno de datos faltantes, la normalización de los datos y la conversión de datos categóricos a numéricos, entre otras tareas.

## Tareas básicas en el análisis de datos

Data Wrangling, también conocido como preparación de datos, es el proceso de limpiar y transformar los datos para que sean más útiles y fáciles de analizar. Dentro del Data Wrangling, hay tareas básicas que se realizan para comenzar a explorar y analizar el conjunto de datos importado.

Una de las tareas más importantes en la preparación de datos es la eliminación de valores faltantes. A menudo, los datos que se recopilan pueden tener valores faltantes que deben ser manejados antes de que se puedan analizar. Esto se puede hacer eliminando filas o columnas que tienen valores faltantes, o imputando los valores faltantes con una media, una mediana o una moda.

Otra tarea común es el formateo de datos para estandarizarlos y hacerlos consistentes. Por ejemplo, si una columna tiene valores en diferentes unidades (por ejemplo, kilómetros y millas), se pueden convertir todos los valores a una unidad común (por ejemplo, kilómetros) para facilitar el análisis.

También es común la conversión de variables categóricas en variables numéricas cuantitativas. Por ejemplo, si una columna tiene categorías como "alto", "medio" y "bajo", estas categorías se pueden convertir en valores numéricos (por ejemplo, 3 para "alto", 2 para "medio" y 1 para "bajo") para facilitar el análisis estadístico.

Además, antes de realizar un análisis de datos más detallado, es importante comprender la estructura y el contenido de los datos. Esto implica realizar un análisis exploratorio de los datos (EDA), que consiste en examinar visualmente y estadísticamente los datos para detectar patrones, relaciones y posibles errores.

Por último, es importante poder manejar datos en diferentes formatos, como texto, CSV, Excel, etc. Dependiendo del formato de los datos, se pueden utilizar diferentes técnicas y herramientas de Data Wrangling.

En resumen, las tareas básicas del Data Wrangling son esenciales para comenzar a explorar y analizar los datos importados, y son críticas para garantizar la calidad y confiabilidad de los resultados del análisis.

### Manejo de valores faltantes en los datos

El manejo de datos faltantes es una tarea importante en el análisis de datos, ya que los datos incompletos pueden afectar la precisión y confiabilidad de los resultados. Los datos faltantes pueden ser el resultado de diversas razones, como errores en la recolección de datos, fallas en el proceso de entrada de datos, o simplemente porque la información no está disponible.

Existen diversas estrategias para manejar datos faltantes, algunas de las cuales se describen a continuación:

#### Eliminación de datos faltantes

Esta estrategia consiste en eliminar las filas o columnas que contienen datos faltantes. Sin embargo, esta estrategia puede ser arriesgada, ya que puede llevar a la pérdida de información valiosa y a un análisis sesgado.

Ejemplo:

| import pandas as pd data = {'A': [1, 2, 3, None, 5],  'B': [6, 7, None, 9, 10],  'C': [11, 12, 13, 14, 15]} df = pd.DataFrame(data) # Eliminar filas con datos faltantes df = df.dropna() print(df) |
| --- |
| A B C 0 1.0 6.0 11 1 2.0 7.0 12 4 5.0 10.0 15 |

#### Imputación de valores faltantes

Otra técnica para manejar valores atípicos es la imputación de valores. Esto implica reemplazar los valores atípicos por un valor estimado. La imputación se puede hacer de diferentes maneras, dependiendo del tipo de datos y del objetivo del análisis. Algunas técnicas comunes de imputación son:

* Reemplazo por la media: en este enfoque, los valores atípicos se reemplazan por la media de los valores no atípicos de la misma variable. Este enfoque funciona bien cuando los valores atípicos son aleatorios y no representan una tendencia en los datos.
* Reemplazo por la mediana: en este enfoque, los valores atípicos se reemplazan por la mediana de los valores no atípicos de la misma variable. Este enfoque funciona bien cuando los valores atípicos son una representación de la tendencia central de los datos.
* Imputación basada en modelos: este enfoque implica utilizar un modelo para estimar los valores faltantes en función de las variables disponibles. Por ejemplo, se puede usar un modelo de regresión para predecir los valores faltantes en una variable en función de otras variables disponibles.

Veamos un ejemplo de cómo se puede realizar la imputación de valores utilizando la biblioteca scikit-learn en Python:

| from sklearn.impute import SimpleImputer import numpy as np import pandas as pd  # Creamos un DataFrame con valores faltantes df = pd.DataFrame({'A': [1, 2, np.nan, 4, 5],  'B': [np.nan, 7, 8, 9, 10],  'C': [11, 12, 13, 14, 15]}) print(df) |
| --- |
| A B C 0 1.0 NaN 11 1 2.0 7.0 12 2 NaN 8.0 13 3 4.0 9.0 14 4 5.0 10.0 15 |
| # Creamos un objeto SimpleImputer para reemplazar los valores faltantes con la media imputer = SimpleImputer(strategy='mean')  # Imputamos los valores faltantes df\_imputed = pd.DataFrame(imputer.fit\_transform(df), columns=df.columns) print(df\_imputed) |
| A B C 0 1.0 8.5 11.0 1 2.0 7.0 12.0 2 3.0 8.0 13.0 3 4.0 9.0 14.0 4 5.0 10.0 15.0 |

En este ejemplo, creamos un DataFrame con algunos valores faltantes en las columnas A y B. Luego, creamos un objeto SimpleImputer que utiliza la media como estrategia de imputación. Finalmente, imputamos los valores faltantes y guardamos el resultado en un nuevo DataFrame llamado df\_imputed. El resultado muestra que los valores faltantes se han reemplazado por la media de cada columna.

#### Eliminar valores atípicos usando Z-score

El método Z-score es una forma común de detectar y eliminar valores atípicos en una distribución normal. El método consiste en calcular la puntuación Z para cada valor en el conjunto de datos y eliminar cualquier valor que esté por encima de un cierto umbral.

Aquí hay un ejemplo de cómo eliminar valores atípicos usando Z-score en Python:

| import pandas as pd import numpy as np  # Creamos un DataFrame de ejemplo df = pd.DataFrame({'A': [1, 2, 3, 4, 5],  'B': [10, 20, 30, 40, 50]}) # Agregamos un valor atípico df = df.append({'A': 6, 'B': 500}, ignore\_index=True) # Calculamos la puntuación Z para cada valor en la columna 'B' z\_scores = (df['B'] - df['B'].mean()) / df['B'].std() # Eliminamos cualquier valor que esté por encima de un umbral de 3 df = df.drop(df[z\_scores > 3].index)  print(df) |
| --- |
| A B 0 1 10 1 2 20 2 3 30 3 4 40 4 5 50 5 6 500 |

Este código creará un DataFrame de ejemplo con una columna llamada "B" que tiene un valor atípico (500 en lugar de los valores cercanos a 30). Luego, calculamos la puntuación Z para cada valor en la columna "B" y eliminamos cualquier valor que esté por encima de un umbral de 3 desviaciones estándar. En este caso, el valor atípico se eliminará y el DataFrame resultante solo tendrá los valores normales de "B".

#### Eliminar valores atípicos usando el rango intercuartil (IQR)

El método del rango intercuartil (IQR) es otra forma común de detectar y eliminar valores atípicos. Este método utiliza la diferencia entre el tercer cuartil (Q3) y el primer cuartil (Q1) de los datos para calcular un umbral para los valores atípicos.

Aquí hay un ejemplo de cómo eliminar valores atípicos usando IQR en Python:

| import pandas as pd import numpy as np  # Creamos un DataFrame de ejemplo df = pd.DataFrame({'A': [1, 2, 3, 4, 5],  'B': [10, 20, 30, 40, 50]}) # Agregamos un valor atípico df = df.append({'A': 6, 'B': 500}, ignore\_index=True) # Calculamos el rango intercuartil (IQR) para la columna 'B' q1 = df['B'].quantile(0.25) q3 = df['B'].quantile(0.75) iqr = q3 - q1 # Calculamos el umbral para los valores atípicos upper\_bound = q3 + (1.5 \* iqr) lower\_bound = q1 - (1.5 \* iqr) # Eliminamos cualquier valor que esté por encima del umbral superior o por debajo del umbral inferior df = df.drop(df[(df['B'] > upper\_bound) | (df['B'] < lower\_bound)].index)  print(df) |
| --- |
| A B 0 1 10 1 2 20 2 3 30 3 4 40 4 5 50 |

En resumen, el manejo de datos faltantes es un paso importante en el análisis de datos y puede ser abordado de diferentes maneras, dependiendo del conjunto de datos y del objetivo del análisis. Es importante evaluar cuidadosamente las opciones disponibles para garantizar que se tomen decisiones informadas y se obtengan resultados precisos y confiables.

### Formateo de datos

El formateo de datos es una tarea fundamental en el análisis de datos, ya que permite estandarizar y normalizar los datos para hacerlos consistentes y comparables. La estandarización se refiere a la conversión de los datos a una escala común, mientras que la normalización se refiere a la transformación de los datos para que tengan una distribución normal.

#### Estandarización:

La estandarización de datos es un proceso de transformación en el que se escala y se desplaza los valores de una variable para que tenga una media de cero y una desviación estándar de uno. Este proceso es útil cuando los datos de diferentes variables tienen diferentes escalas y no se pueden comparar fácilmente. La estandarización permite comparar las variables en una escala común.

A continuación se presenta un ejemplo práctico de cómo realizar la estandarización de datos en Python utilizando la biblioteca NumPy:

| import numpy as np  # Creamos un array con valores aleatorios datos = np.array([10, 20, 30, 40, 50])  print("Datos originales:", datos) |
| --- |
| > Datos originales: [10 20 30 40 50] |
| # Calculamos la media y desviación estándar de los datos media = np.mean(datos) std = np.std(datos) # Realizamos la estandarización de los datos datos\_estandarizados = (datos - media) / std  print("Datos estandarizados:", datos\_estandarizados) |
| > Datos estandarizados: [-1.41421356 -0.70710678 0. 0.70710678 1.41421356] |

En este ejemplo, creamos un array con valores aleatorios y luego calculamos la media y desviación estándar de los datos. Luego, aplicamos la fórmula de estandarización para obtener un nuevo array de datos estandarizados. En este nuevo array, podemos observar que la media es cero y la desviación estándar es uno.

Como analogía, podemos imaginar que tenemos dos personas que miden su altura en diferentes unidades: una persona mide su altura en metros y la otra en pies. Para comparar sus alturas de manera justa, debemos convertir ambas unidades a una misma escala. De esta manera, podemos estandarizar los datos para poder compararlos de manera adecuada.

#### Normalización:

La normalización es otra técnica común en el preprocesamiento de datos. Consiste en escalar los datos a un rango común, generalmente de 0 a 1. La normalización es útil cuando los valores de las características (features) tienen diferentes rangos y unidades, y queremos asegurarnos de que todas tengan el mismo impacto en el modelo.

Por ejemplo, si tenemos un conjunto de datos que contiene información sobre el peso de personas en libras y su altura en pulgadas, es posible que queramos normalizar los datos para que los valores de peso y altura tengan el mismo impacto en el modelo. Para hacerlo, podemos utilizar la fórmula de la normalización min-max:

| x\_norm = (x - min(x)) / (max(x) - min(x)) |
| --- |

Donde x es el valor de la característica, min(x) es el valor mínimo de la característica y max(x) es el valor máximo de la característica.

A continuación, un ejemplo práctico utilizando Python y la librería Scikit-Learn:

| from sklearn.preprocessing import MinMaxScaler import pandas as pd  # Creamos un DataFrame con algunas características data = {'edad': [25, 30, 35, 40, 45],  'ingresos': [50000, 70000, 90000, 110000, 130000],  'deuda': [10000, 20000, 30000, 40000, 50000]} df = pd.DataFrame(data)  # Normalizamos los datos scaler = MinMaxScaler() df\_norm = pd.DataFrame(scaler.fit\_transform(df), columns=df.columns)  print(df\_norm) |
| --- |
| edad ingresos deuda 0 0.00 0.00 0.00 1 0.25 0.25 0.25 2 0.50 0.50 0.50 3 0.75 0.75 0.75 4 1.00 1.00 1.00 |

En este ejemplo, creamos un DataFrame con tres características: edad, ingresos y deuda. A continuación, utilizamos el objeto MinMaxScaler de Scikit-Learn para normalizar los datos. El resultado es un nuevo DataFrame df\_norm con los mismos datos que df, pero normalizados entre 0 y 1.

La normalización de los datos también se puede visualizar mediante un gráfico. A continuación, un ejemplo que muestra la diferencia entre los datos originales y los datos normalizados:

| import matplotlib.pyplot as plt  # Graficamos los datos originales y normalizados fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(10, 4))  df.plot(ax=ax1) ax1.set\_title('Datos originales')  df\_norm.plot(ax=ax2) ax2.set\_title('Datos normalizados') |
| --- |
|  |

En este ejemplo, utilizamos la librería Matplotlib para graficar los datos originales y normalizados. El resultado es un gráfico que muestra la diferencia entre los dos conjuntos de datos. Se puede observar que los datos normalizados están todos en el rango de 0 a 1, mientras que los datos originales tienen diferentes escalas.

En resumen, tanto la estandarización como la normalización son técnicas útiles en el pre-procesamiento de datos que pueden ayudar a mejorar el rendimiento de los modelos de aprendizaje automático. Es importante tener en cuenta que estas técnicas deben aplicarse después de la limpieza y manipulación de los datos faltantes o inconsistentes.

### Variables Categóricas y variables numéricas cuantitativas.

### Análisis de datos raw.

### Manejo de datos en diferentes formatos

Cuando trabajamos con datos, a menudo nos encontramos con diferentes formatos de archivo que contienen la información que necesitamos. Es importante saber cómo importar estos archivos y cómo manejarlos en Python para poder trabajar con ellos.

Una de las librerías más utilizadas para importar datos en Python es pandas. Pandas nos permite leer archivos en varios formatos, como texto, CSV, Excel y muchos otros. A continuación, se presentan algunos ejemplos de cómo podemos importar diferentes tipos de archivos con pandas:

| import pandas as pd df = pd.read\_csv('datos.txt', sep='\t') |
| --- |

En este ejemplo, estamos importando un archivo de texto llamado "datos.txt" que está separado por tabulaciones. El método read\_csv de pandas nos permite especificar el separador que se utiliza en el archivo.

| import pandas as pd df = pd.read\_csv('datos.csv') |
| --- |

En este ejemplo, estamos importando un archivo CSV llamado "datos.csv". Pandas reconoce automáticamente que el archivo está separado por comas.

| import pandas as pd df = pd.read\_excel('datos.xlsx', sheet\_name='Hoja1') |
| --- |

En este ejemplo, estamos importando un archivo Excel llamado "datos.xlsx" y especificando que queremos importar la hoja llamada "Hoja1".

Una vez que hemos importado los datos, podemos realizar diferentes tareas de data wrangling para limpiar y transformar los datos. Por ejemplo, podemos eliminar filas o columnas que no sean relevantes para nuestro análisis, cambiar el tipo de datos de una columna, o unir dos DataFrames. Pandas tiene muchas funciones y métodos que nos permiten realizar estas tareas de manera sencilla y eficiente.

En resumen, saber cómo importar y manejar datos en diferentes formatos es esencial para cualquier análisis de datos. Pandas es una herramienta muy útil para trabajar con datos en Python, y nos permite importar y manipular datos en diferentes formatos de manera sencilla y eficiente.

En conclusión, el módulo de Data Wrangling se podría comparar con la tarea de preparar una cena. Al igual que en el proceso de preparar una cena, en Data Wrangling se deben realizar algunas tareas básicas para comenzar a explorar y analizar el conjunto de datos importado, como la limpieza, el filtrado y la selección de los datos relevantes.

Al igual que en una cena, a veces los ingredientes no están disponibles o están dañados, lo que significa que no se pueden utilizar en el plato. Del mismo modo, los datos pueden contener valores faltantes o errores, lo que requiere una manipulación cuidadosa para que se ajusten al conjunto de datos. Además, al igual que en la cocina, es importante estandarizar y hacer consistentes los datos para asegurar que el análisis sea preciso y coherente.

En general, el Data Wrangling es un proceso importante y crítico en el análisis de datos, ya que garantiza que los datos sean limpios, precisos y adecuados para su uso en el análisis y la toma de decisiones.