Técnicas de limpieza y perfilado de datos

Técnicas de limpieza de datos

Tratamiento de los valores perdidos

- **Identificar los valores perdidos**

In [ ]:

```
import pandas as pd
df = pd.read_csv('../../data/housing.csv')
df.info()
df.describe()
```

In [ ]:

```
rows_with_nan = df[df.isnull().any(axis=1)]
print(rows_with_nan)
print('\nFilas con nan:\n', rows_with_nan.count())
```

- **Opción de eliminar entrada**

In [ ]:

```
df.dropna(inplace=True)
```

- **O intentar imputar un valor razonable para ponerlo en su lugar.**

In [ ]:

```
df.fillna(df.mean(numeric_only=True), inplace=True)
```

Detección y tratamiento de valores atípicos

Detección

In [4]:

```
# Datos de muestra y adición de outliers
import numpy as np

np.random.seed(0)
data = np.random.randint(low=0, high=11, size=1000)
data[0] = 100
data[1] = -100
```

- **Calcular los z-scores (https://en.wikipedia.org/wiki/Standard_score)**
- **Identificar valores atípicos según z-cores (ej. 3)**

In [ ]:

```
z_scores = (data - np.mean(data)) / np.std(data)

threshold = 3
outliers = np.where(np.abs(z_scores) > threshold)[0]
print(data[outliers])
```

- **Otro método consiste en calcular el rango intercuartílico (IQR: https://en.wikipedia.org/wiki/Interquartile_range) de la distribución y clasificar cualquier valor que sea Q1-(1,5 x IQR) o Q3 + (1,5 x IQR) como valores atípicos potenciales.**

In [ ]:

```
# Calcular IQR
q1 = np.percentile(data, 25)
q3 = np.percentile(data, 75)
iqr = q3 - q1

# Identificar outliers según IQR
lower_bound = q1 - (1.5 * iqr)
upper_bound = q3 + (1.5 * iqr)
outliers = np.where((data < lower_bound) | (data > upper_bound))[0]
print(data[outliers])
```

Tratamiento

- **Si se determinó que el valor atípico se debe a un error, simplemente corregir el error para resolver el valor atípico.**
- **En otros casos, eliminar el valor atípico del conjunto de datos o sustituirlo por un valor menos extremo que conserve la forma general de la distribución.**
- **La limitación: es un método en el que estableces un tope, o umbral, en la distribución de los datos y se sustituye cualquier valor fuera de esos límites por un valor especificado.**

In [ ]:

```
# Datos de ejemplo
data = {'A': [100, 90, 85, 88, 110, 115, 120, 130, 140],
        'B': [1, 2, 3, 4, 5, 6, 7, 8, 9]}
df = pd.DataFrame(data)

# Umbrales inferior y superior para la limitación (aquí los percentiles 5 y 95)
umbral_inferior = df.quantile(0.05, numeric_only=True)
umbral_superior = df.quantile(0.95, numeric_only=True)
print('umbral_inferior:\n', umbral_inferior)
print('umbral_superior:\n', umbral_superior)

# Limitar valores atípicos
capped_df = df.clip(lower=umbral_inferior, upper=umbral_superior, axis=1)
print("DataFrame original:")
print(df)
print("\nDataFrame limitado:")
print(capped_df)
```

- **En algunos casos, se puede transformar los datos de forma que los valores atípicos tengan menos impacto, como una transformación de raíz cuadrada o una transformación logarítmica.**

Precauciones

- **Comprende la distribución de datos subyacente: Antes de aplicar cualquier transformación, es importante comprender la distribución de tus datos y cómo influirán en ella determinadas transformaciones.**
- **Elige una transformación adecuada: Selecciona un método de transformación adecuado a tu distribución de datos.**
- **Maneja ceros y valores negativos: Algunas transformaciones pueden no ser adecuadas para datos que contengan ceros o valores negativos. Añadir una pequeña constante puede ayudar a evitar problemas al tomar logaritmos, por ejemplo.**
- **Valida los datos transformados: Tras aplicar las transformaciones, valida los datos transformados para asegurarte de que la distribución resultante cumple los supuestos de tu análisis.**
- **Considera la interpretabilidad: Los datos transformados pueden no ser tan fácilmente interpretables como los datos originales. Asegúrate de que las partes interesadas comprenden las implicaciones de la transformación en la interpretación de los resultados.**

Lecturas recomendadas:

- **https://github.com/francomanca93/fundamentos-de-estadistica-con-python**
- **https://datos.gob.es/sites/default/files/doc/file/guia_eda_python.pdf**

Tratamiento de duplicados

- **Utilizando el método duplicated() de la biblioteca pandas de Python, puedes identificar fácilmente las filas duplicadas en un DataFrame para examinarlas.**

In [ ]:

```
df = pd.read_csv('../data/housing.csv')
df = pd.concat([df, df.iloc[[1, 60, 6]]])
duplicate_rows = df[df.duplicated()]
print(duplicate_rows)
```

- **La mayoría de los duplicados pueden ser copias exactas y pueden eliminarse simplemente utilizando el método drop_duplicates() de Pandas:**

In [ ]:

```
cleaned_df = df.drop_duplicates()
duplicate_rows = cleaned_df[cleaned_df.duplicated()]
print(duplicate_rows)
```

- **En algunos casos, puede ser más apropiado fusionar registros duplicados, agregando información.**
  - **Por ejemplo, si los duplicados representan varias entradas para la misma entidad, podemos fusionarlos utilizando funciones de agregación:**

In [ ]:

```
data = {'customer_id': [102, 102, 101, 103, 102], 'product_id': [
    'A', 'B', 'A', 'C', 'B'], 'quantity_sold': [5, 3, 2, 1, 4]}
df = pd.DataFrame(data)
print(df, '\n')

merged_df = df.groupby(['customer_id', 'product_id']).agg(
    {'quantity_sold': 'sum'}).reset_index()
print(merged_df)
```

Tratar las incoherencias

- **Los distintos tipos de incoherencias requerirán soluciones diferentes. Las incoherencias derivadas de la introducción de datos incorrectos o de erratas pueden tener que ser corregidas por una fuente experta. Otra posibilidad es sustituir los datos incorrectos mediante imputación, como si se tratara de un valor omitido, o eliminarlos por completo del conjunto de datos, según las circunstancias.**
- **Las incoherencias en el formato de los datos pueden corregirse utilizando algunos métodos de normalización. Para eliminar los espacios iniciales y finales de una cadena, puedes utilizar el método .strip(). Los métodos .upper() y .lower() normalizarán las mayúsculas y minúsculas en las cadenas. Y la conversión de fechas a datetimes mediante pd.to_datetime normalizará el formato de las fechas.**
- **También se debe asegurar de que cada valor de la columna es del mismo tipo usando: .astype().**
- **Otras correcciones de incoherencias de formato que se pueden necesitar llevar a cabo son:**
  - **Conversión de unidades**
  - **Normalización del correo electrónico, el teléfono y la dirección**
  - **Eliminar la puntuación de las cadenas**
  - **Utilizar el mapeo de valores para tratar las abreviaturas comunes**

In [ ]:

```
value_mapping = {'M': 'Male', 'F': 'Female'}
standardized_value = value_mapping.get('M', 'Unknown')
print(standardized_value)
```

Perfilado de datos

Exploratory Data Analysis (EDA)

El análisis EDA puede servir para muchos propósitos, pero algunos de los más destacados son preparar los datos para el modelado, intentar encontrar algunas tendencias y responder a las preguntas iniciales.

Al tener una visión general de nuestros datos, podemos ver si estamos yendo por el camino correcto.

Entendiendo los datos (básico):

- **Información general con** `data.info()`
- **Manipular columnas con** `data.columns`
- **Comprobación de las dimensiones de los datos con** `data.shape`

- **Comprobación de las dimensiones de los datos con** `data.shape`
- **Obtener el tipo de datos de cada atributo con** `data.dtypes`
- **Observar los datos en bruto** `data.head()` **,** `data.tail()`
- **Resumen estadístico de datos con** `data.describe()`
- **Revisar la distribución de clases con** `data.groupby()`
- **Revisión de la correlación entre atributos con** `data.corr(method='pearson')`
- **Revisar el sesgo de la distribución de atributos con** `data.skew()`

Asimismo, podemos mejorar la calida de los datos:

- **Actualizar tipos y limpiar caracteres**
- **Revisar calidad del dato y limpiar**

In [ ]:
```
df.info()
```

In [ ]:
```
df.head()
df.tail()
```

In [ ]:
```
df.shape
```

In [ ]:
```
df.dtypes
```

In [ ]:
```
df.describe()
```

In [ ]:
```
df_group = df.groupby('ocean_proximity')['population'].mean()
df_group.name='media_population'
df_group
```

In [ ]:
```
df.corr(method='pearson',numeric_only=True)
```

In [ ]:
```
df.skew(numeric_only=True)
```

Pandas-profiling

Pandas Profiling es una librería que genera informes desde un DataFrame de pandas.

Hace poco cambió el nombre de paquete a:ydata-profiling

Presenta en un informe interactivo las siguientes estadísticas para cada columna.

- **Inferencia de tipo: detectar los tipos de columnas en un dataframe.**
- **Esenciales: tipo, valores únicos, valores perdidos**
- **Estadísticas de cantidad como valor mínimo, Q1, mediana, Q3, máximo, rango, rango intercuartíl**
- **Estadísticas descriptivas como la media, la moda, la desviación estándar, la suma, la mediana de la desviación absoluta, el coeficiente de variación, la curtosis, la asimetría**
- **Los valores más frecuentes**
- **Histogramas**
- **Correlaciones destacadas de variables altamente correlacionadas, matrices de Spearman, Pearson y Kendall**
- **Matriz de valores perdidos, recuento, mapa de calor y dendrograma de valores perdidos**
- **Análisis de texto aprende sobre las categorías (Mayúsculas, Espacio), guiones (Latín, Cirílico) y bloques (ASCII) de datos de texto.**
- **Análisis de archivos e imágenes extrae los tamaños de los archivos, las fechas de creación y las dimensiones y escanea las imágenes truncadas o que contienen información EXIF.**

In [ ]:
```
# !pip install ydata-profiling
```

In [2]:

```
from ydata_profiling import ProfileReport
profile = ProfileReport(df, title="Pandas Profiling Report")
```

In [ ]:

```
profile.to_notebook_iframe()
```

In [ ]:

```
profile.to_file("profile_report.html")
```