UNIR

Máster Oficial en Visual Analytics & Big Data

Limpieza y preparación de datos.

Análisis integrado del sector turístico.

Santiago Ríos Benjumea

Asignatura: *Bases de Datos para el Big Data*

Docente: *Marlon Cárdenas*

2025

Índice

1. Problemas encontrados

1.1. Tipo de error #1: Valores nulos

Descripción: En el `dataset 201132-0-museos.csv` se identificaron varias columnas con valores faltantes. Dos de ellas (`ESCALERAS` y `PUERTA`) presentaban el 100% de registros nulos (69 de 69). Otras columnas como `PLANTA, ORIENTACION, FAX, EQUIPAMIENTO` y DESCRIPCION también tenían un porcentaje muy alto de valores nulos (superior al 75%).

Cómo se detectó: Se utilizó la función `.isnull().sum()` y su variante porcentual con `.isnull().mean() * 100` en pandas para calcular la cantidad y proporción de valores faltantes por columna.

Por qué es un problema: Las columnas completamente vacías no aportan información al análisis. Además, las columnas con más del 75% de valores nulos generan ruido.

Forma de solucionarlo: Se eliminaron las columnas con el 100% de valores nulos, asimismo, las que tenían más del 75% de valores nulos. Para columnas con una proporción más moderada de nulos y tipo de dato texto, como `DESCRIPCION-ENTIDAD, HORARIO, EMAIL y TELEFONO`, se cambiaron los valores faltantes con la palabra "`Desconocido`".

Justificación de la solución: La eliminación de columnas vacías o prácticamente vacías optimiza la estructura de la base de datos, mejora el rendimiento en el análisis de datos y evita interpretaciones sesgadas o erróneas. El reemplazo de nulos en columnas relevantes por el valor "`Desconocido`" permite mantener la consistencia del dataset.

1.2. Tipo de error #2: Inconsistencia de tipo de dato

Descripción: En el dataset `201132-0-museos.csv`, se detectó que la columna `ACCESIBILIDAD` estaba almacenada como texto (`object`) en lugar de un tipo numérico. Esto se debía a que los valores usaban comas (,) como separador decimal, lo que impide su conversión automática a números por parte de pandas.

Cómo se detectó: Se utilizó el método .dtypes de pandas para revisar los tipos de datos por columna. Al notar que `ACCESIBILIDAD` figuraba como object, se inspeccionaron los valores, confirmando que contenían comas como separador decimal (por ejemplo, "0,4").

Por qué es un problema: Los valores numéricos escritos con coma decimal no pueden ser procesados matemáticamente si están en formato texto. Esto limita operaciones como agrupaciones, estadísticas, visualización y modelos que dependen de tipos numéricos.

Forma de solucionarlo: Primero se reemplazaron las comas por puntos usando `.str.replace(",", ".")`. Luego, se convirtió la columna al tipo `float` usando `pd.to_numeric(..., errors="coerce")`.

Justificación de la solución: La conversión al formato numérico adecuado permite realizar análisis cuantitativo, operaciones estadísticas y visualizaciones correctas. Además, al usar `errors="coerce"`, se previenen errores durante la conversión y se identifican automáticamente los valores no válidos como `NaN`.

1.3. Tipo de error #3: Error de formato

Descripción: En el dataset `alojamientos_turisticos.csv`, la columna planta tenía valores con formatos inconsistentes: algunos registros incluían el símbolo "º" junto al número de planta (ej. "1º", "2º"), mientras que otros contenían solo el número (ej. "1", "2"). Esta mezcla de formatos dificulta el análisis y la agrupación coherente de los datos.

Cómo se detectó: Se revisaron los valores únicos de la columna usando `.unique()` en pandas, lo que permitió identificar la presencia de variantes inconsistentes del mismo dato.

Por qué es un problema: Estas diferencias de formato impiden realizar agrupamientos o filtrados correctos, ya que los valores son tratados como distintos, aunque semánticamente sean equivalentes.

Forma de solucionarlo: Se eliminó el símbolo "º" de todos los valores en la columna planta utilizando el método `.str.replace()` de pandas. Los valores textuales no numéricos, como "`Bajo`" o "`Sótano`", se conservaron tal como estaban, al no presentar inconsistencia.

Justificación de la solución: La limpieza del formato permite que los valores numéricos sean homogéneos y fácilmente comparables.

1.4. Tipo de error #4: Errores semánticos o de interpretación

Descripción: En el dataset `alojamientos_turisticos.csv`, la columna `NUMERO` presenta valores que no corresponden a números de portal válidos, como "`S/Nº`", "`3 DUPL`", "`13 BIS`", "`53-55`" o "`68.000`", que reflejan errores de escritura, formatos inconsistentes o información no estandarizada.

Cómo se detectó: Se utilizó el método `.unique()` de pandas para revisar los valores distintos de la columna. Al intentar convertir los valores a formato numérico con

`pd.to_numeric(df_almacenamiento[“numero”], errors="coerce")`, se detectaron múltiples entradas no convertibles que no representaban datos válidos.

Por qué es un problema: Estos valores impiden la estandarización de direcciones, afectan el análisis geográfico y pueden causar errores en la unión con otras bases de datos.

Forma de solucionarlo: Se estandarizaron variantes de "sin número" (como "`S/Nº`", "`SN`", etc.) en "`S/N`" y se reemplazaron los valores no numéricos que no aportaban información (como "`3 DUPL`", "`68.000`", etc.) con `NaN`.

Justificación de la solución: Limpiar esta columna permite realizar análisis y comparaciones correctas sobre direcciones. El uso de `NaN` para valores no válidos evita distorsionar estadísticas o mapas, y mantiene la integridad del dataset.

1.5. Tipo de error #5: Columna irrelevante o vacía

Descripción: En el dataset `data_act_01.csv`, se identificó que la columna `Range` está completamente vacía, es decir, no contiene ningún valor útil en ninguno de sus `10.051` registros.

Cómo se detectó: Se utilizó el método `.isnull().mean()` de pandas para calcular el porcentaje de valores nulos por columna. La columna `Range` presentó un 100% de valores faltantes, confirmando su inutilidad.

Por qué es un problema: Las columnas completamente vacías ocupan memoria innecesariamente, complican la estructura del dataset y no aportan información útil para el análisis.

Forma de solucionarlo: Se eliminó la columna `Range` usando el método `.drop(columns=["Range"])`.

Justificación de la solución: Eliminar columnas vacías optimiza el rendimiento y facilita el análisis al dejar solo los campos que contienen datos relevantes.

1.6. Tipo de error #6: Incoherencia temporal

Descripción: En el dataset data_act_01.csv, se identificó un caso en el que la fecha del delito (`OffenseDate`) era posterior al momento de la llamada (`CallDateTime`). Esta incoherencia temporal indica un error en los datos, ya que una llamada no puede registrarse antes de que ocurra el evento.

Cómo se detectó: Se convirtieron ambas columnas a tipo `datetime` con `pd.to_datetime()`. Luego se compararon los valores fila a fila con `OffenseDate > CallDateTime`, lo que permitió identificar una fila incoherente.

Por qué es un problema: Una secuencia temporal incorrecta afecta la integridad del análisis cronológico. Aunque el impacto de un solo registro es bajo, su presencia puede generar errores en cálculos de tiempos de respuesta o análisis de tendencias temporales.

Forma de solucionarlo: Dado que no existía información adicional para corregir el valor, se optó por eliminar el único registro inconsistente usando una condición lógica. Se usó el código `df_data = df_data[df_data['OffenseDate_parsed'] <= df_data['CallDateTime_parsed']]`

Justificación de la solución: Eliminar registros con errores temporales sin posibilidad de validación asegura la coherencia del dataset. En este caso, se trataba de un único caso entre más de 10.000 registros, por lo que no afecta la representatividad de los datos.

2. Datos en JSON

Los tres datasets limpios (`museos`, `alojamientos_turisticos` e `incidentes`) fueron convertidos a un único archivo JSON llamado `RIOS_BENJUMEA_SANTIAGO_actividad_1.json`.

Cada conjunto de datos se incluyó como una lista de registros (`orient="records"`) bajo una clave distinta, manteniendo toda la información de cada fila del CSV original.

Antes de la conversión:

- Se eliminaron columnas vacías y valores inconsistentes.
- Se corrigieron errores de formato, tipos de datos y texto.
- Se estandarizaron todos los campos para asegurar coherencia.

La conversión se realizó con Python utilizando `pandas` y `json`.

3. Metodología aplicada

1. Cargar los datos desde los archivos fuente.
2. Revisar estructura, tipos de datos y contenido general.
3. Detectar y tratar valores nulos, eliminando o completando según el caso.
4. Corregir tipos de datos (números, fechas, texto, etc.).

5. Normalizar textos (espacios, mayúsculas, errores comunes).
6. Eliminar duplicados y columnas irrelevantes.
7. Validar coherencia de los datos (rangos, relaciones, categorías).
8. Guardar el dataset limpio en un nuevo archivo.

4. Propuesta de mejora para la calidad de datos

4.1. Validar los datos desde el origen

Muchos errores se podrían evitar si los datos se controlaran correctamente al momento de su captura. Se recomienda:

- Hacer obligatorios los campos importantes (como dirección o categoría).
- Usar listas desplegables para evitar que los usuarios escriban libremente (por ejemplo, en tipo de alojamiento, localidad, estado).
- Validar automáticamente formatos de fechas, códigos postales y números.

4.2. Unificar formatos y nombres

Se encontraron muchas formas diferentes de escribir lo mismo (como `"S/Nº"`, `"s/n"`, `"S. N."`, etc.). Para evitar esto:

- Establecer una lista de opciones válidas para cada campo.
- Usar códigos estandarizados (por ejemplo, `"1-HOTEL"`, `"2-PENSION"`) en lugar de texto libre.
- Documentar bien las reglas de escritura para que todos los datos se ingresen igual.

Estas dos mejoras ayudarían a reducir errores, facilitar el análisis y evitar tener que limpiar los datos manualmente cada vez.