

Navegación

Ir a la sección:

- ☐ Exploración Inicial
- ☒ Limpieza y Validación
- ☐ Indicadores y Documentación
- ☐ EDA Avanzado & Dashboards
- ☐ Modelado de Machine Learning



# Análisis y Calidad de Datos de Pacientes de Hospital

Esta aplicación realiza un análisis exhaustivo de la calidad de los datos de pacientes, seguido de procesos de limpieza, validación, generación de KPIs, EDA avanzado y un modelo de Machine Learning.

## 2. 🧹 Limpieza y Validación

Aplicación de un proceso de limpieza para resolver los problemas identificados y validaciones cruzadas.

### 2.1. Proceso de Limpieza

#### Limpieza de `sexo`

```
# Convertir a cadena y a minúsculas para un manejo consistente
df_cleaned['sexo'] = df_cleaned['sexo'].astype(str).str.lower()

# Mapear valores a 'Female', 'Male' o np.nan para no mapeados
sex_mapping = {
    'f': 'Female',
    'female': 'Female',
    'm': 'Male',
    'male': 'Male'
}

df_cleaned['sexo'] = df_cleaned['sexo'].map(sex_mapping) # Esto generará np.nan para valores no

# Finalmente, reemplazar np.nan con None (Python None)
df_cleaned.loc[df_cleaned['sexo'].isna(), 'sexo'] = None
```

Valores de `sexo` después de la normalización y mapeo:

sexo	count
Male	
Female	
None	

**Justificación:** Los valores de la columna `sexo` se normalizan a minúsculas y luego se mapean explícitamente a `'Female'` o `'Male'`. Cualquier valor que no coincida con estas categorías mapeadas (incluyendo cadenas vacías, `'nan'`, o `'Other'`) se convierte a `None` (nulo), asegurando una consistencia total para análisis y filtros. Se utiliza `numpy.nan` para el manejo intermedio de nulos, que es la forma estándar de Pandas.

#### Limpieza y Cálculo de `fecha_nacimiento` y `edad`

```
# Convertir 'fecha_nacimiento' a datetime, forzando nulos si el formato es inválido
df_cleaned['fecha_nacimiento'] = pd.to_datetime(df_cleaned['fecha_nacimiento'], errors='coerce')

# Calcular 'edad' para nulos o valores inconsistentes
current_date = date.today() # Fecha actual
df_cleaned['edad_calculada'] = df_cleaned['fecha_nacimiento'].apply(lambda dob: calculate_age_fr
```

```
# Priorizar edad calculada si fecha_nacimiento es válida, de lo contrario usar existente o None
df_cleaned['edad'] = df_cleaned.apply(
    lambda row: row['edad_calculada'] if pd.notna(row['fecha_nacimiento']) else row['edad'], axis=0
)
df_cleaned['edad'] = df_cleaned['edad'].astype('Int64') # Int64 para permitir NaNs y mantenerlo
```

Valores nulos en `edad` después de la limpieza: 2

Valores nulos en `fecha_nacimiento` después de la limpieza: 4

#### Justificación:

- `fecha_nacimiento` se convierte a tipo `datetime`, convirtiendo formatos inválidos a `NaT` (Not a Time).
- `edad` se recalcula basándose en `fecha_nacimiento` si es válida. Esta edad calculada se prioriza si está disponible. Si `fecha_nacimiento` es `NaT`, se mantiene la `edad` original.
- Asegura que la edad sea un entero no negativo. Se usa `Int64` para manejar nulos en columnas numéricas.

## Limpieza de `telefono`

```
# Eliminar caracteres no numéricos
df_cleaned['telefono'] = df_cleaned['telefono'].astype(str).str.replace(r'[^\d-]', '', regex=True)
# Reemplazar cadenas vacías (o solo espacios) con None
df_cleaned.loc[df_cleaned['telefono'].str.strip() == '', 'telefono'] = None
```

Ejemplos de `telefono` después de la limpieza:

	telefono
0	3429501064
1	None
2	3157898999
3	None
4	None

**Justificación:** Se eliminan caracteres no numéricos del teléfono para estandarizar el formato. Las cadenas vacías o aquellas con solo espacios se convierten a `None`.

## 2.2. Validaciones Cruzadas

Se aplican reglas para asegurar la consistencia lógica entre columnas.

### Validación: `edad` consistente con `fecha_nacimiento`

No se encontraron inconsistencias significativas entre `edad` y `fecha_nacimiento` después de la limpieza.

### Validación: `email` con formato válido

Se encontraron 2506 registros con correo electrónico inválido después de la limpieza.

	id_paciente		email
1		2	None
9		10	None
21		22	None
22		23	None
23		24	None

**Regla de Validación:** El campo `email` debe seguir un formato de correo electrónico estándar ( `texto@texto.dominio` ).  
**Acción:** Se identifican pero no se modifican automáticamente, ya que esto requeriría inferencia o interacción manual.

**Validación:** `telefono` contiene solo dígitos (después de la limpieza)

Todos los teléfonos contienen solo dígitos o son nulos después de la limpieza.

## 2.4. Detección y Manejo de Duplicados

Identifica y gestiona registros duplicados en el dataset.

Selecciona las columnas para detectar duplicados:

id\_paciente x

nombre x

fecha\_nacimiento x

telefono x

x v

Buscar Duplicados y Aplicar Acción

## 2.5. Gestión de Valores Atípicos (Outliers)

Identifica y opcionalmente maneja los valores atípicos en columnas numéricas.

Selecciona una columna numérica para detectar outliers:

id\_paciente v

Límites de Detección de Outliers (IQR) para 'id\_paciente':

Q1: 1251.25, Q3: 3750.75, IQR: 2499.50

Límite Inferior: -2498.00, Límite Superior: 7500.00

No se encontraron valores atípicos en la columna 'id\_paciente' usando el método IQR.

## 2.6. Análisis de Completitud por Umbral

Verifica qué columnas cumplen con un umbral de datos no nulos.

Porcentaje mínimo de completitud deseado (% no nulos):



	Porcentaje No Nulo (%)	Cumple Umbral
id_paciente	100	
nombre	100	
edad	99.9601	
fecha_nacimiento	99.9202	
ciudad	83.493	
sexo	79.5808	
telefono	66.7066	
email	49.98	

Las siguientes columnas no cumplen con el umbral de 90% de completitud:

	Porcentaje No Nulo (%)	Cumple Umbral
ciudad	83.493	
sexo	79.5808	
telefono	66.7066	
email	49.98	

## 2.7. Validación de Rangos Numéricos

Verifica si los valores de una columna numérica están dentro de un rango aceptable.

Selecciona una columna numérica para validar el rango:

id\_paciente

Valor mínimo aceptable para id\_paciente:

0

Valor máximo aceptable para id\_paciente:

5000

Todos los valores en 'id\_paciente' están dentro del rango [0, 5000].

## 2.8. DataFrame Después de la Limpieza

Las primeras 10 filas del DataFrame limpio:

	id_paciente	nombre	fecha_nacimiento	edad	sexo	email
0	1	Claudia Torres	1954-01-08 00:00:00	71	Female	user1@example.com
1	2	Carlos Gómez	1965-01-01 00:00:00	60	Female	None
2	3	Carlos Gómez	2009-03-08 00:00:00	16	None	user3@example.com
3	4	Andrea López	1951-11-18 00:00:00	73	Female	user4@example.com
4	5	Juan Gómez	1961-09-05 00:00:00	63	Female	user5@example.com
5	6	María López	1966-10-26 00:00:00	58	Male	user6@example.com
6	7	María Torres	1954-03-16 00:00:00	71	None	user7@example.com
7	8	Andrea López	2004-02-12 00:00:00	21	Male	user8@example.com
8	9	María Torres	1974-02-04 00:00:00	51	Female	user9@example.com
9	10	Juan López	1961-04-28 00:00:00	64	Male	None

Información del DataFrame limpio:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5010 entries, 0 to 5009
Data columns (total 8 columns):
#   Column      Non-Null Count  Dtype
---  ---
0  id_paciente  5010 non-null  int64
1  nombre       5010 non-null  object
2  fecha_nacimiento  5006 non-null  datetime64[ns]
3  edad         5008 non-null  Int64
4  sexo         3987 non-null  object
5  email        2504 non-null  object
6  telefono     3342 non-null  object
7  ciudad       4183 non-null  object
dtypes: Int64(1), datetime64[ns](1), int64(1), object(5)
memory usage: 318.1+ KB
```