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.

3. 📈 Indicadores de Calidad y Documentación

Resumen de indicadores de calidad antes y después de la limpieza, junto con la documentación.

3.1. Indicadores de Calidad de Datos

Comparación de Valores Faltantes (%)

Columna	Original (%)	Limpio (%)
edad	32.8743	
fecha_nacimiento	0	
telefono	33.2934	
sexo	20.4192	

Observaciones:

- Se espera una **reducción significativa** en el porcentaje de nulos en `edad` si `fecha_nacimiento` estaba disponible y era válida.
- `fecha_nacimiento` puede mostrar un aumento de nulos si los formatos originales eran inválidos y se convirtieron a `NaT`.
- `telefono` puede tener nulos si quedaron cadenas vacías después de limpiar caracteres no numéricos.
- `sexo` podría tener nulos si había valores vacíos o no estandarizables.

Comparación de Tipos de Datos

Tipos de datos originales:


```
{
  "id_paciente" : "int64"
  "nombre" : "object"
  "fecha_nacimiento" : "object"
  "edad" : "float64"
  "sexo" : "object"
  "email" : "object"
  "telefono" : "object"
  "ciudad" : "object"
}
```


Tipos de datos después de la limpieza:


```
{
  "id_paciente" : "int64"
  "nombre" : "object"
```

```
  "fecha_nacimiento" : "datetime64[ns]"
  "edad" : "Int64"
  "sexo" : "object"
  "email" : "object"
  "telefono" : "object"
  "ciudad" : "object"
}
```

Observaciones:

- `fecha_nacimiento` debería cambiar de `object` (cadena) a `datetime64[ns]` (tipo fecha y hora).
- `edad` debería cambiar de `object` (si contenía nulos o estaba mezclado) o `float64` (si se infirió numérico) a `Int64` (entero con soporte para nulos).
- `telefono` y `email` idealmente deberían permanecer como `object` (cadena) pero con formato validado.

Indicadores de Consistencia y Unicidad

`sexo` **- Unicidad de Categorías:**

Original: ['Male', None, 'F', 'Female', 'M']

Limpio: ['Male', 'Female', None]

Observación: Se espera que el número de categorías únicas y sus nombres se normalicen después de la limpieza (ej., solo 'Female', 'Male' y `None`).

`email` **- Patrón de Formato (Conteo de Inválidos):**

Correos inválidos (Original): **2506**

Correos inválidos (Limpio): **2506**

Observación: Aunque la limpieza no los altera, se validó su formato. Este indicador muestra si persisten correos con formato no estándar.

`telefono` **- Contiene solo dígitos (después de la limpieza):**

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

3.2. Documentación del Proceso

Supuestos Adoptados Durante la Limpieza:

- **Fuente Única para Edad:** Se asume que `fecha_nacimiento` es la fuente más confiable para determinar la `edad`. Si `fecha_nacimiento` es válida, se **prioriza el cálculo de la edad a partir de ella** sobre el valor `edad` existente si este es nulo o inconsistente. La edad se calcula como la diferencia en años a la fecha actual, ajustando por mes y día.
- **Formato de `sexo`:** Se asume que los valores `Female`, `female`, `Male`, `male`, `F`, `f`, `M`, `m` y sus variaciones deben ser estandarizados a `Female` y `Male`. Cualquier otro valor (`NaN`, vacío, o no reconocido) se convierte a `None`.
- **Formato de `telefono`:** Se asume que los números de teléfono deben contener solo dígitos. Cualquier otro carácter (guiones, espacios, paréntesis, etc.) es **removido**. Las cadenas vacías o que solo consisten en espacios resultantes de esta limpieza se interpretan como nulas (`None`).
- **Coherencia de Fechas:** Se asume que las fechas de nacimiento no pueden ser en el futuro ni excesivamente antiguas (la edad se calcula en relación con la fecha actual y las edades negativas se descartan, convirtiéndolas a `None`).
- **ID de Paciente:** Se asume que `id_paciente` es el identificador **único** de cada paciente y no se espera que tenga problemas de calidad (duplicados, nulos).

Reglas de Validación Implementadas:

- **Validación de `fecha_nacimiento`:** Se verifica que la columna pueda ser convertida a tipo `datetime`. Los valores que no cumplen con este formato se marcan como `NaT` (Not a Time).
- **Validación de `edad`:**
  - Debe ser un entero no negativo.
  - Debe ser **consistente** con `fecha_nacimiento`: la `edad` calculada a partir de `fecha_nacimiento` debe ser cercana a la `edad` reportada (se permite una tolerancia de 1 año para posibles discrepancias de actualización de fechas en los datos originales).
- **Validación de `sexo`:** Los valores deben estar dentro de un conjunto predefinido de categorías estandarizadas (`Female`, `Male` o `None`).
- **Validación de `email`:** Se verifica que el formato siga una expresión regular básica (`[^@]+@[^@]+\.[^@]+`) para asegurar que contenga un `@` y al menos un `.` en el dominio. Esta es una validación de patrón, no de existencia.
- **Validación de `telefono`:** Se verifica que, después de la limpieza, la columna contenga solo caracteres numéricos (o sea nula).

Recomendaciones de Mejora para Asegurar la Calidad Futura de los Datos:

1. **Validación en Origen:** Implementar validaciones a nivel de entrada de datos (ej., formularios web, bases de datos) para `fecha_nacimiento`, `sexo`, `email` y `telefono`.
   - `fecha_nacimiento`: Usar selectores de fecha para prevenir entradas manuales erróneas y asegurar formato `AAAA-MM-DD`.
   - `sexo`: Usar listas desplegables con opciones predefinidas (`Female`, `Male`) para evitar inconsistencias de capitalización o errores tipográficos.
   - `email`: Implementar validación de formato de correo electrónico en tiempo real en la entrada de datos y, si es posible, una verificación de dominio.
   - `telefono`: Forzar la entrada de solo dígitos o un formato específico (ej., con máscaras de entrada) dependiendo del país, y validar longitud mínima/máxima.
2. **Estandarización de `ciudad`:** Implementar un catálogo o lista maestra de ciudades/municipios para asegurar consistencia y evitar variaciones en los nombres de ciudades (ej., "Barranquilla" vs "barranquilla", o errores tipográficos).
3. **Definición de Campos Obligatorios:** Establecer claramente qué campos son obligatorios (ej., `id_paciente`, `nombre`, `fecha_nacimiento`, `sexo`) en la base de datos o sistema de entrada para reducir la aparición de valores nulos críticos.
4. **Auditorías Regulares de Datos:** Realizar auditorías periódicas de la base de datos para identificar nuevos patrones de error o degradación de la calidad de los datos con el tiempo.
5. **Documentación de Metadatos:** Mantener un `diccionario de datos` actualizado que defina claramente cada campo, su tipo de dato esperado, formato, reglas de validación y significado, accesible para todo el equipo.
6. **Sistema de Reporte de Errores:** Establecer un mecanismo para que los usuarios (personal del hospital, médicos) reporten inconsistencias o errores en los datos cuando los detecten, con un flujo claro para su corrección.
7. **Capacitación del Personal:** Asegurar que el personal encargado de la entrada de datos esté continuamente capacitado en las mejores prácticas de entrada de datos y comprenda la importancia de la calidad de los datos para la toma de decisiones y la atención al paciente.

3.3. Bonus (Opcional)

Implementación de Pruebas Automáticas

Para implementar pruebas automáticas para la calidad de los datos, se podrían usar frameworks como **Pytest** o **Great Expectations**.

Ejemplo conceptual con Pytest (en un archivo `tests/test_data_quality.py`):

```
# Este código es conceptual y no forma parte de app.py
# Deberías tener tus funciones de limpieza y validación en un módulo separado para importarlas a
import pandas as pd
import pytest
```

```
from datetime import date, datetime
# from your_project.data_quality_functions import clean_patient_data, calculate_age_from_dob # E

# Asegúrate de que calculate_age_from_dob sea accesible si no la importas desde un módulo
def calculate_age_from_dob(row_dob, current_date):
    if pd.isna(row_dob):
        return None
    else:
        if isinstance(row_dob, str):
            try:
                row_dob = datetime.strptime(row_dob, '%Y-%m-%d').date()
            except ValueError:
                return None
        elif isinstance(row_dob, pd.Timestamp):
            row_dob = row_dob.date()

        age = current_date.year - row_dob.year - ((current_date.month, current_date.day) < (row_
        return age if age >= 0 else None

@pytest.fixture
def sample_patient_data():
    # Datos de prueba con casos conocidos para verificar la limpieza
    data = {
        "pacientes": [
            {"id_paciente": 1, "nombre": "Claudia Torres", "fecha_nacimiento": "1954-01-08", "ed
            {"id_paciente": 2, "nombre": "Pedro Gomez", "fecha_nacimiento": "1980-05-15", "edad"
            {"id_paciente": 3, "nombre": "Ana Smith", "fecha_nacimiento": "2025-01-01", "edad":
            {"id_paciente": 4, "nombre": "Luis Lopez", "fecha_nacimiento": "1990-11-20", "edad":
            {"id_paciente": 5, "nombre": "Maria Paz", "fecha_nacimiento": None, "edad": 30, "sex
            {"id_paciente": 6, "nombre": "Carlos", "fecha_nacimiento": "1970-07-08", "edad": 50,
            {"id_paciente": 7, "nombre": "Laura", "fecha_nacimiento": "1995-03-20", "edad": None
            {"id_paciente": 8, "nombre": "Diego", "fecha_nacimiento": "1988-09-10", "edad": None
        ]
    }
    return pd.json_normalize(data['pacientes'])

# Esta sería la función de limpieza que probarías, adaptada de tu app.py
def clean_patient_data_for_test(df_raw):
    df_cleaned_test = df_raw.copy()
    current_date_for_test = date(2025, 7, 9) # Fecha fija para las pruebas de edad

    # Sexo
    df_cleaned_test['sexo'] = df_cleaned_test['sexo'].astype(str).str.lower()
    sex_mapping = {
        'f': 'Female',
        'female': 'Female',
        'm': 'Male',
        'male': 'Male'
    }
    df_cleaned_test['sexo'] = df_cleaned_test['sexo'].map(sex_mapping)
    df_cleaned_test.loc[df_cleaned_test['sexo'].isna(), 'sexo'] = None


    # Fecha Nacimiento y Edad
    df_cleaned_test['fecha_nacimiento'] = pd.to_datetime(df_cleaned_test['fecha_nacimiento'], er
    df_cleaned_test['edad_calculada_test'] = df_cleaned_test['fecha_nacimiento'].apply(lambda do
    df_cleaned_test['edad'] = df_cleaned_test.apply(
        lambda row: row['edad_calculada_test'] if pd.notna(row['edad_calculada_test']) else row[
    )
    df_cleaned_test['edad'] = df_cleaned_test['edad'].astype('Int64')
    df_cleaned_test = df_cleaned_test.drop(columns=['edad_calculada_test'])

    # Telefono
    df_cleaned_test['telefono'] = df_cleaned_test['telefono'].astype(str).str.replace(r'[^0-9]',
    df_cleaned_test.loc[df_cleaned_test['telefono'].str.strip() == '', 'telefono'] = None
```

```
    return df_cleaned_test

def test_sexo_standardization(sample_patient_data):
    df_cleaned = clean_patient_data_for_test(sample_patient_data)
    assert all(s in ['Female', 'Male', None] for s in df_cleaned['sexo'].unique()), "Los valores
    assert pd.isna(df_cleaned.loc[df_cleaned['id_paciente'] == 6, 'sexo'].iloc[0]), "El valor 'O
    assert df_cleaned.loc[df_cleaned['id_paciente'] == 7, 'sexo'].iloc[0] == 'Female', "El valor
    assert df_cleaned.loc[df_cleaned['id_paciente'] == 8, 'sexo'].iloc[0] == 'Male', "El valor '


def test_age_calculation_and_validation(sample_patient_data):
    df_cleaned = clean_patient_data_for_test(sample_patient_data)
    assert all(df_cleaned['edad'].dropna() >= 0), "Las edades calculadas no deben ser negativas.
    # Verificar edad para id_paciente 1 (1954-01-08) -> 2025-1954 = 71
    assert df_cleaned.loc[df_cleaned['id_paciente'] == 1, 'edad'].iloc[0] == 71, "La edad para e
    # Verificar que la fecha futura (2025-01-01) resulta en edad nula/None
    assert pd.isna(df_cleaned.loc[df_cleaned['id_paciente'] == 3, 'edad'].iloc[0]), "La edad par
    # Verificar que si fecha_nacimiento es nulo pero la edad existe, se mantiene (id_paciente 5)
    assert df_cleaned.loc[df_cleaned['id_paciente'] == 5, 'edad'].iloc[0] == 30, "La edad para e


def test_email_format_after_cleaning(sample_patient_data):
    df_cleaned = clean_patient_data_for_test(sample_patient_data) # No hay limpieza directa de e
    invalid_emails_in_cleaned = df_cleaned[~df_cleaned['email'].astype(str).str.match(r'[^@]+@[^
    # Esperamos que 'pedro@example' siga siendo inválido
    assert 'pedro@example' in invalid_emails_in_cleaned['email'].values, "El email 'pedro@exampl
    # No esperamos que se añadan nuevos invalidos, y su numero debe ser consistente con los orig
    assert len(invalid_emails_in_cleaned) == 1, "Se detecto un numero inesperado de correos elec


def test_telefono_numeric_after_cleaning(sample_patient_data):
    df_cleaned = clean_patient_data_for_test(sample_patient_data)
    assert all(df_cleaned['telefono'].dropna().apply(lambda x: x.isdigit())), "El campo telefono
    assert pd.isna(df_cleaned.loc[df_cleaned['id_paciente'] == 2, 'telefono'].iloc[0]), "El tele
```

Para ejecutar Pytest, necesitas:

1. `pytest` instalado: `pip install pytest`
2. Guardar el código de prueba en un archivo como `tests/test_data_quality.py` (o similar) en una carpeta `tests/`.
3. **Importante:** Refactorizar tus funciones de limpieza y validación de `app.py` en un módulo de Python separado (ej., `procesamiento_datos.py`) para que puedas importarlas en las pruebas. O, para esta demostración, puedes copiar y adaptar las funciones de limpieza dentro del propio archivo de prueba como se muestra arriba.
4. Ejecutar `pytest` en tu terminal desde la raíz de tu proyecto.

Simulación de Migración de Datos Limpios a una Estructura Objetivo

Una vez que los datos han sido limpiados y validados, el siguiente paso lógico en una tubería de datos es cargarlos en una estructura objetivo, como un Data Warehouse o una base de datos analítica. Formatos como **Parquet** son ideales para esto debido a su naturaleza columnar, compresión eficiente y capacidad para manejar esquemas complejos.

Aquí simulamos la descarga de los datos limpios en formatos comunes para la migración.

Simulación de Migración de Datos Limpios

Descargar Datos Limpios (CSV)

Descargar Datos Limpios (Parquet)

Justificación de la Migración: La migración de datos limpios a un Data Warehouse (DW) típicamente implica:

1. **Extract (Extraer):** Obtener datos de las fuentes.
2. **Transform (Transformar):** Los datos son limpiados, estandarizados, validados y preparados para ajustarse al esquema del DW. Esta es la fase que hemos detallado en esta aplicación.

3. **Load (Cargar):** Los datos transformados se cargan en las tablas dimensionales y de hechos del DW. Los formatos como CSV son universales, pero Parquet es preferido en entornos de Big Data y DW por su eficiencia. La simulación de descarga CSV/Parquet representa la salida de este proceso de transformación, lista para ser cargada en un sistema optimizado para consultas analíticas.