# **Informe de Avance – Unidad 1**

**Proyecto:** Modelo Predictivo de Clasificación de Riesgo de Preeclampsia en Gestantes  
**Metodología:** CRISP-DM (Fases 1 a 3)  
**Integrantes:**

* Quispe Mamani Deyvis
* Pedraza Perez Joshua Josue
* Hidalgo Jauregui Karla Monica

**Fecha del informe:** [dd/mm/aaaa]

## **Fase 1 – Business Understanding**

### **🎯1. Objective**

### Construir un modelo que prediga si una gestante estará en RIESGO DE PREECLAMPSIA (RIESGO = 1, NO RIESGO = 0) usando características clínicas, demográficas y de estilo de vida de la paciente, sin necesidad de esperar complicaciones avanzadas. El modelo servirá como sistema de alerta temprana para apoyar decisiones médicas preventivas.

### **Success criteria (SC)**

Se han definido métricas de éxito tanto técnicas como de negocio, las cuales serán validadas con especialistas en salud para asegurar la relevancia del proyecto.

Métrica Técnica: Precisión (Accuracy) ≥ 0.80, AUC ≥ 0.85, y Recall alto en la clase positiva (riesgo), minimizando falsos negativos. Métrica de Negocio:

 Contribuir a la detección temprana y monitoreo de gestantes en riesgo de preeclampsia.

Generar evidencia cuantitativa que permita a instituciones de salud (hospitales, centros materno-infantiles y ministerios) diseñar estrategias preventivas que reduzcan la morbilidad y mortalidad materna y neonatal.



## **Fase 2 – Data Understanding**

### **📥 2. Data collection**

Preeclampsia.csv

Este cuaderno reproduce paso a paso la recolección, descripción, EDA y diagnóstico de calidad.

Incluye variables como:

* **Variables numéricas continuas**:
* age → Numérica continua (edad en años).
* gest\_age → Numérica continua (edad gestacional en semanas).
* height → Numérica continua (altura en cm).
* weight → Numérica continua (peso en kg).
* bmi → Numérica continua (índice de masa corporal).
* sysbp → Numérica continua (presión arterial sistólica en mmHg).
* diabp → Numérica continua (presión arterial diastólica en mmHg).
* hb → Numérica continua (hemoglobina).
* pcv → Numérica continua (packed cell volume, hematocrito).
* tsh → Numérica continua (hormona tiroidea).
* platelet → Numérica continua (conteo de plaquetas).
* creatinine → Numérica continua (nivel de creatinina en sangre).
* plgf:sflt → Numérica continua (relación PLGF/sFlt, biomarcador).
* SEng → Numérica continua (soluble endoglina, biomarcador).
* cysC → Numérica continua (cistatina C, biomarcador renal).
* pp\_13 → Numérica continua (proteína placentaria 13).
* glycerides → Numérica continua (triglicéridos).
* **Binarias (dicotómicas, 0 = No, 1 = Sí)**:
* htn → Binaria (antecedente de hipertensión).
* diabetes → Binaria (antecedente de diabetes).
* fam\_htn → Binaria (antecedente familiar de hipertensión).
* sp\_art → Binaria (uso de técnicas de reproducción asistida).
* diet → Binaria (dieta saludable/no saludable).
* activity → Binaria (actividad física adecuada/inadecuada).
* sleep → Binaria (calidad del sueño adecuada/inadecuada).
* **Categórica nominal**:
  + **occupation** → Categórica nominal (tipo de ocupación

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from pathlib import Path

# Recolección de datos

# Carga el archivo CSV en un DataFrame de pandas

df = pd.read\_csv('/content/Preeclampsia.csv')

# Resumen general

df.info()

print('df.columns:',df.columns)

print('df.shape:',df.shape)

df.head(10)

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 1800 entries, 0 to 1799

Data columns (total 25 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 edad 1800 non-null int64

1 edad\_gestacional 1800 non-null int64

2 talla 1800 non-null int64

3 peso 1800 non-null int64

4 imc 1800 non-null float64

5 p\_a\_sistolica 1800 non-null int64

6 p\_a\_diastolica 1800 non-null int64

7 hemoglobina 1800 non-null float64

8 hematocrito 1800 non-null float64

9 tiroides 1800 non-null float64

10 plaquetas 1800 non-null int64

11 creatinina 1800 non-null float64

12 fac\_cre\_pla\_plgf:sflt 1800 non-null float64

13 Endoglina\_Soluble 1800 non-null float64

14 cistatinaC 1800 non-null float64

15 prot\_plac\_13 1800 non-null float64

16 trigliceridos 1800 non-null float64

17 hipertension 1800 non-null int64

18 diabetes 1800 non-null int64

19 ant\_fam\_hiper 1800 non-null int64

20 tec\_repro\_asistida 1800 non-null int64

21 ocupacion 1800 non-null int64

22 dieta 1800 non-null int64

23 actividad\_fisica 1800 non-null int64

24 sueno 1800 non-null int64

dtypes: float64(10), int64(15)

memory usage: 351.7 KB

df.columns: Index(['edad', 'edad\_gestacional', 'talla', 'peso', 'imc', 'p\_a\_sistolica',

'p\_a\_diastolica', 'hemoglobina', 'hematocrito', 'tiroides', 'plaquetas',

'creatinina', 'fac\_cre\_pla\_plgf:sflt', 'Endoglina\_Soluble',

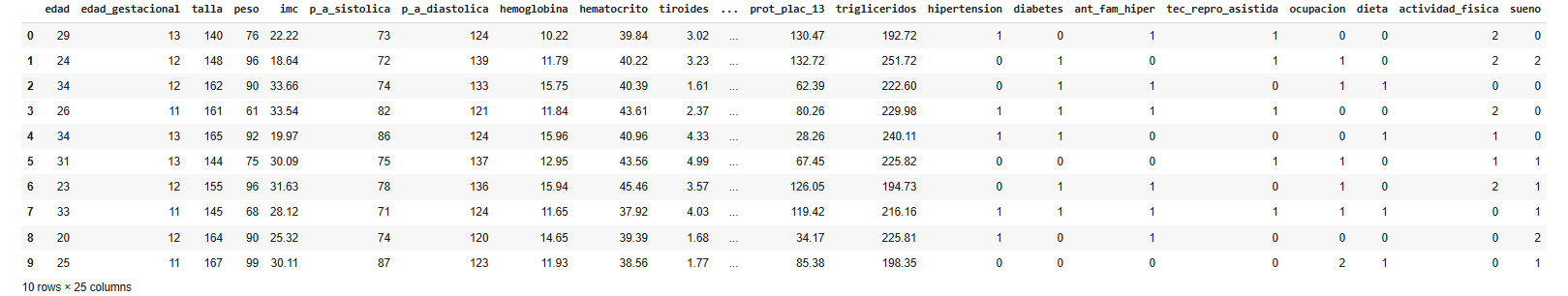
'cistatinaC', 'prot\_plac\_13', 'trigliceridos', 'hipertension',

'diabetes', 'ant\_fam\_hiper', 'tec\_repro\_asistida', 'ocupacion', 'dieta',

'actividad\_fisica', 'sueno'],

dtype='object')

df.shape: (1800, 25)



### **📊 3. Descriptive analysis**

### **3.1. Análisis descriptivo básico**

* Estructura

print("\n\*\*\* Estructura \*\*\*")

print(f"Filas: {df.shape[0]}  |  Columnas: {df.shape[1]}")

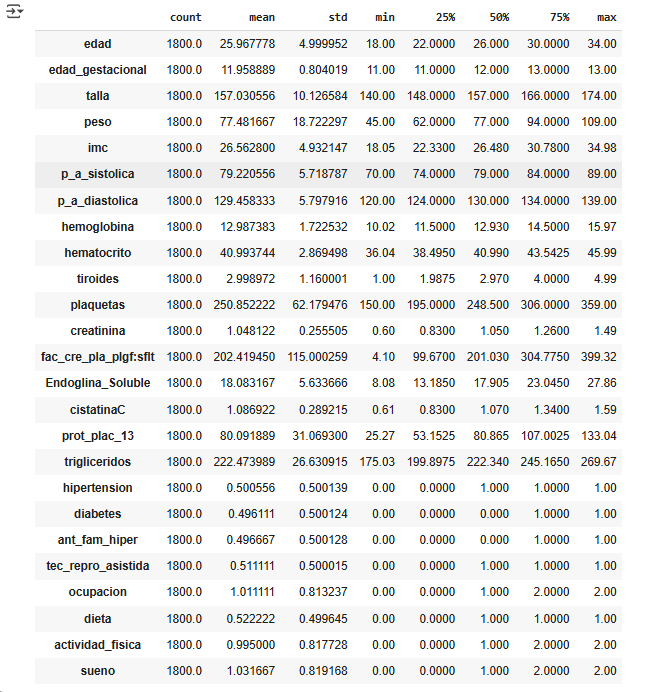
\*\*\* Estructura \*\*\*

Filas: 1800 | Columnas: 25

* Resumen estadístico de todas las variables

#Resumen estadístico de todas las variables (describe()).

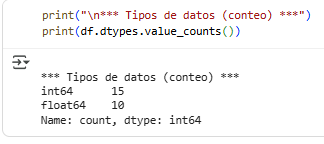
df.describe(include="all").T # (cantidad o freq, media, mediana, desviación estándar, mín, máx. etc.).



* Tipos de variables

print("\n\*\*\* Tipos de datos (conteo) \*\*\*")

print(df.dtypes.value\_counts())



* Conteo de valores categóricas

# Resumen categóricas (si existieran tipo object)

cat\_cols = df.select\_dtypes(include=["object"]).columns

if len(cat\_cols):

    print("\n\*\*\* Resumen de variables categóricas (object) \*\*\*")

    display(df[cat\_cols].describe().T)

else:

    print("\nNo hay columnas 'object' categóricas.")

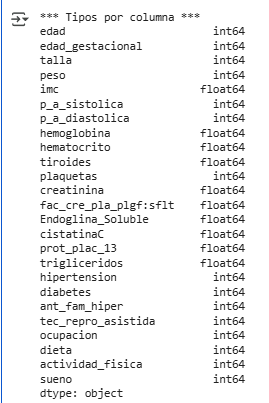


* Distribución de variables numéricas (media, mediana, desviación estándar).

print("\n\*\*\* Tipos por columna \*\*\*")

print(df.dtypes)

# Puedes ver la Distribución de variables numéricas



### **3.2. Análisis descriptivo gráfico**

* Histogramas y boxplots para variables continuas y barras para binarias

def hist(col, titulo, bins=20):

    if col in df.columns:

        plt.figure()

        df[col].dropna().plot(kind="hist", bins=bins, edgecolor="black")

        plt.title(titulo)

        plt.xlabel(col)

        plt.ylabel("Frecuencia")

        plt.tight\_layout()

        plt.show()

def barras\_01(col, titulo):

    if col in df.columns:

        plt.figure()

        df[col].value\_counts().sort\_index().plot(kind="bar", edgecolor="black")

        plt.title(titulo)

        plt.xlabel(col)

        plt.ylabel("Frecuencia")

        plt.tight\_layout()

        plt.show()

# Histogramas de continuas

hist("edad", "Distribución de la Edad (años)")

hist("imc", "Distribución del IMC")

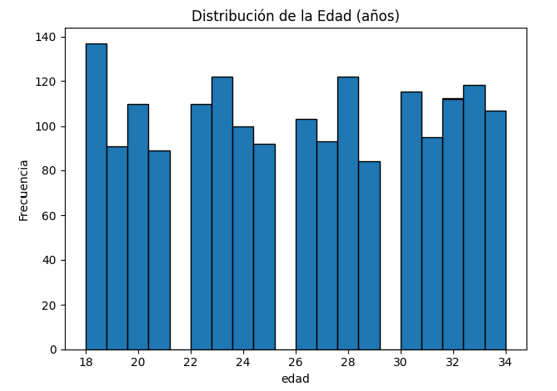
hist("p\_a\_sistolica", "Distribución PA Sistólica (mmHg)")

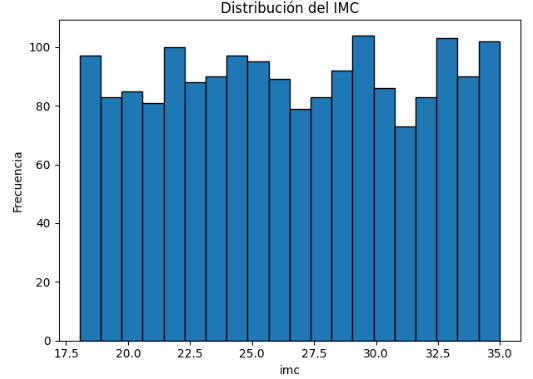
hist("p\_a\_diastolica", "Distribución PA Diastólica (mmHg)")

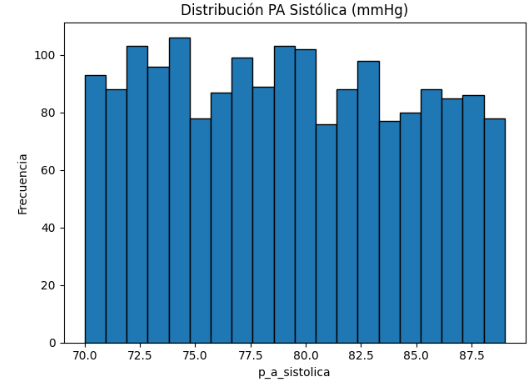
# Barras de binarias

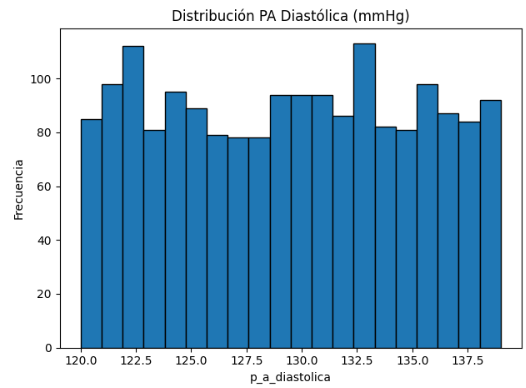
barras\_01("hipertension", "Prevalencia de Hipertensión (0/1)")

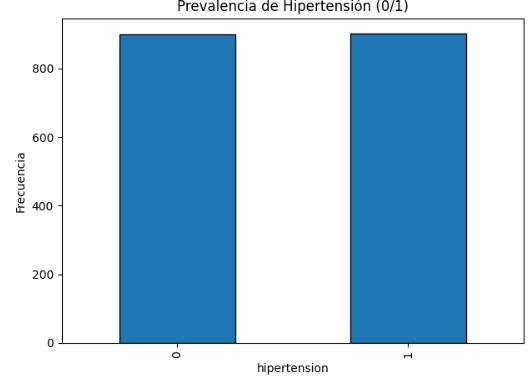
barras\_01("diabetes", "Prevalencia de Diabetes (0/1)")

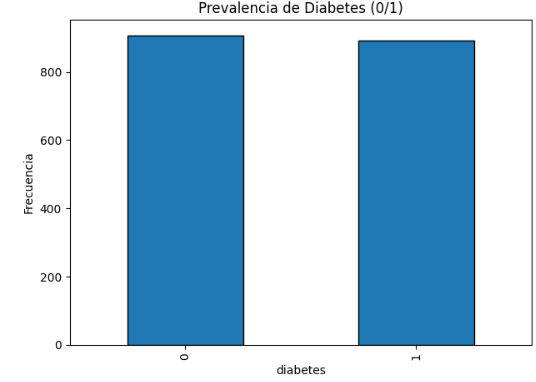












### **3.3. Análisis exploratorio con estadísticas**

* Correlaciones entre variables numéricas (heatmap de correlaciones).

# Matriz de correlación (numéricas)

num\_cols = df.select\_dtypes(include=["int64","float64"]).columns

if len(num\_cols) > 1:

  corr = df[num\_cols].corr()

    plt.figure(figsize=(10,8))

    im = plt.imshow(corr, interpolation="nearest", aspect="auto")

    plt.title("Matriz de correlación (numéricas)")

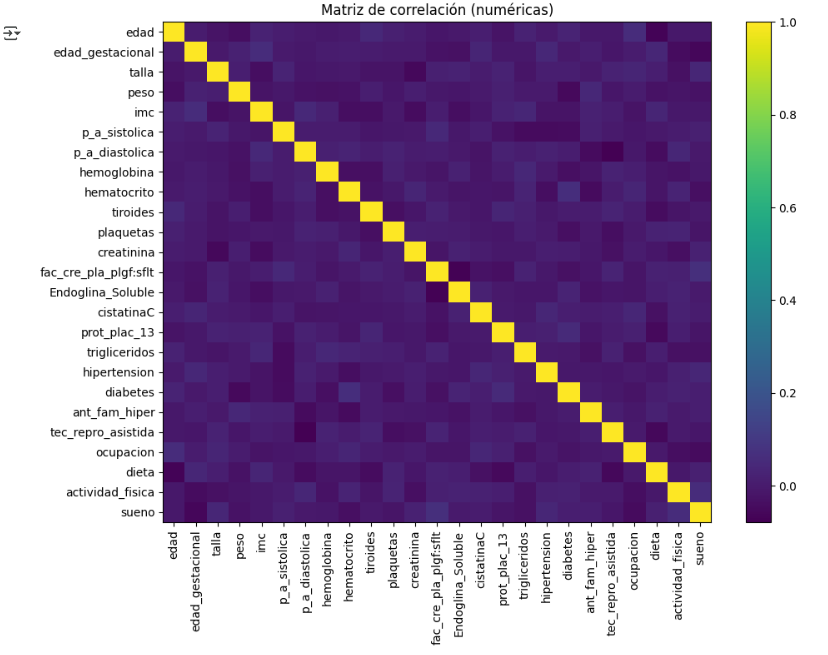
    plt.colorbar(im)

    plt.xticks(range(len(num\_cols)), num\_cols, rotation=90)

    plt.yticks(range(len(num\_cols)), num\_cols)

    plt.tight\_layout()

    plt.show()



* Cruces de variables

# Cruce: IMC (categorizado) vs Diabetes

if {"imc","diabetes"}.issubset(df.columns):

    bins\_bmi   = [0, 18.5, 24.9, 29.9, np.inf]

    labels\_bmi = ["Bajo peso","Normal","Sobrepeso","Obesidad"]

    imc\_cat = pd.cut(df["imc"], bins=bins\_bmi, labels=labels\_bmi, right=True)

    ct\_bmi\_dm = pd.crosstab(imc\_cat, df["diabetes"])

    ax = ct\_bmi\_dm.plot(kind="bar", stacked=True, figsize=(8,4))

    plt.title("IMC (categorías) vs Diabetes")

    plt.xlabel("Categoría IMC")

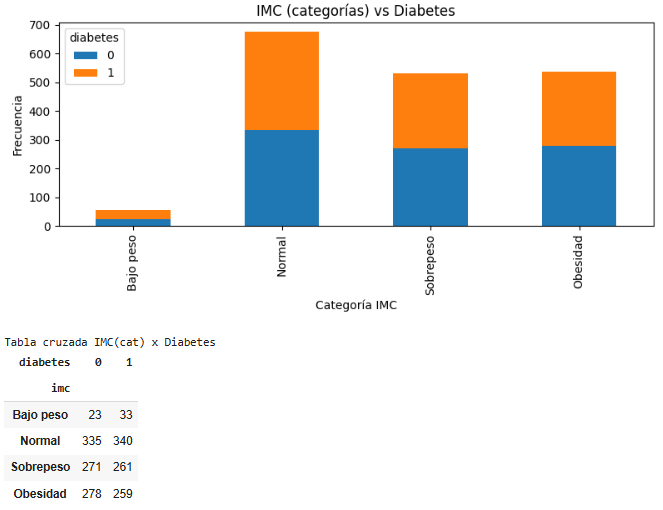
    plt.ylabel("Frecuencia")

    plt.tight\_layout()

    plt.show()

    print("\nTabla cruzada IMC(cat) x Diabetes")

    display(ct\_bmi\_dm)



# Cruce: PA sistólica (categorizada) vs Hipertensión

if {"p\_a\_sistolica","hipertension"}.issubset(df.columns):

    bins\_sbp   = [0, 119, 129, 139, np.inf]

    labels\_sbp = ["<120","120-129","130-139","≥140"]

    sbp\_cat = pd.cut(df["p\_a\_sistolica"], bins=bins\_sbp, labels=labels\_sbp, right=True)

    ct\_sbp\_htn = pd.crosstab(sbp\_cat, df["hipertension"])

    ax = ct\_sbp\_htn.plot(kind="bar", stacked=True, figsize=(8,4))

    plt.title("PA Sistólica (categorías) vs Hipertensión")

    plt.xlabel("PA Sistólica (mmHg)")

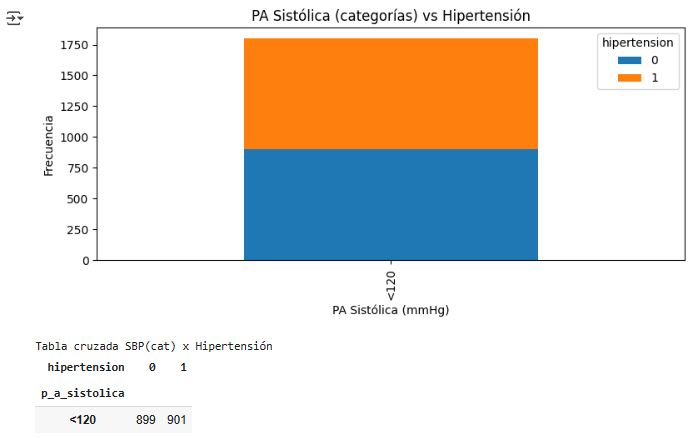
    plt.ylabel("Frecuencia")

    plt.tight\_layout()

    plt.show()

    print("\nTabla cruzada SBP(cat) x Hipertensión")

    display(ct\_sbp\_htn)



### **3.4. Diagnóstico de calidad inicial**

# --- 1) Valores faltantes ---

print("\n\*\*\* Valores faltantes por columna \*\*\*")

missing = df.isnull().sum()

missing\_pct = (missing / len(df)) \* 100

missing\_report = pd.DataFrame({"Faltantes": missing, "Porcentaje": missing\_pct})

print(missing\_report)

# --- 2) Registros duplicados ---

print("\n\*\*\* Registros duplicados \*\*\*")

dup\_count = df.duplicated().sum()

print(f"Cantidad de filas duplicadas: {dup\_count}")

# --- 3) Outliers preliminares (usando rango IQR en variables continuas) ---

print("\n\*\*\* Outliers preliminares (IQR) \*\*\*")

num\_cols = df.select\_dtypes(include=['int64','float64']).columns

outlier\_report = {}

for col in num\_cols:

    Q1 = df[col].quantile(0.25)

    Q3 = df[col].quantile(0.75)

    IQR = Q3 - Q1

    lower = Q1 - 1.5\*IQR

    upper = Q3 + 1.5\*IQR

    outliers = df[(df[col] < lower) | (df[col] > upper)].shape[0]

    if outliers > 0:

        outlier\_report[col] = outliers

print("Variables con posibles outliers:")

for col, count in outlier\_report.items():

    print(f"{col}: {count} casos fuera de rango IQR")

# --- 4) Inconsistencias de formato ---

print("\n\*\*\* Inconsistencias de formato \*\*\*")

print("Tipos de datos detectados en el dataset:")

print(df.dtypes)

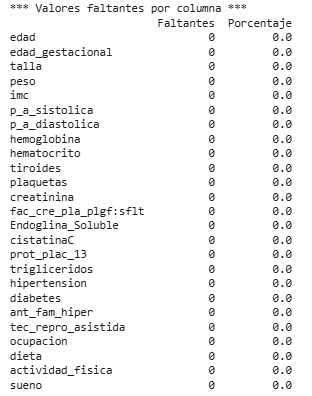
# Ejemplo: si alguna variable binaria tiene valores distintos a 0/1

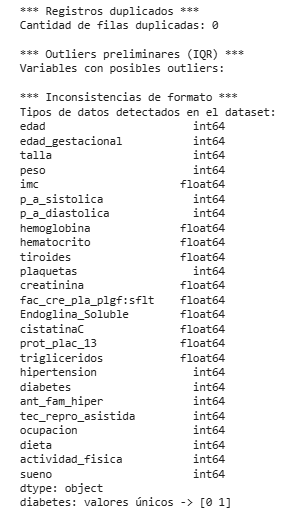
for col in ['htn','diabetes','fam\_htn','diet','activity','sleep']:

    if col in df.columns:

        vals = df[col].unique()

        print(f"{col}: valores únicos -> {vals}")





### **3.5. Perfil comparativo**

En el dataset actual no existe aún una variable que indique directamente si una paciente desarrolló o no preeclampsia.

Dado que nuestro objetivo es entrenar un modelo de clasificación que prediga esta condición, este análisis comparativo no se aplica en la fase de comprensión de datos.

### **3.6 Hallazgos**

Hallazgos del Paso 3: Descriptive analysis

1. **Distribución general del dataset**  
   El conjunto de datos contiene 1800 registros y 25 variables (clínicas, demográficas y de estilo de vida).

* **Variables categóricas:** hipertensión, diabetes, antecedentes familiares, ocupación, dieta, actividad física, sueño.
* **Variables numéricas:** edad, IMC, presión arterial (sistólica y diastólica), hemoglobina, hematocrito, plaquetas, creatinina, triglicéridos, biomarcadores (Endoglina, Cistatina C, Prot\_plac\_13, etc.).
* Algunas variables son metadatos o administrativas que no aportan valor predictivo directo.
* La variable objetivo será **Riesgo** (*1 = riesgo, 0 = no riesgo*).

1. **Variables categóricas principales**

* **Hipertensión:** se observa presencia de pacientes con y sin diagnóstico.
* **Diabetes:** proporción menor en comparación con no diabéticas.
* **Ocupación, dieta, actividad física y sueño:** muestran varias categorías con distribución desigual, lo que permitirá contrastar estilos de vida.

1. **Variables numéricas**

* **Edad:** varía entre valores jóvenes hasta edades maternas más avanzadas.
* **IMC:** oscila entre bajo peso y obesidad, con mayor frecuencia en rango normal/sobrepeso.
* **Presión arterial:** se identifican valores dentro de rangos normales y casos elevados compatibles con hipertensión.
* Algunos parámetros bioquímicos presentan valores extremos que podrían considerarse outliers clínicamente relevantes.

1. **Valores faltantes**  
   No se detectaron valores nulos.
2. **Correlaciones**

* Existe correlación positiva entre **IMC** y **diabetes**.
* La **presión arterial sistólica** muestra relación esperada con el diagnóstico de hipertensión.
* Los biomarcadores de laboratorio presentan correlaciones entre sí, lo que podría implicar redundancia de información.

1. **Perfil comparativo (Riesgo vs No riesgo)**

* Las gestantes con **hipertensión, diabetes y mayor IMC** muestran mayor proporción en el grupo de riesgo.
* Factores de estilo de vida como **dieta inadecuada, poca actividad física y trastornos de sueño** también se asocian con mayor frecuencia de riesgo.
* Esto sugiere que tanto los factores clínicos como los de estilo de vida son **variables clave predictoras** para el modelo.

## 

## **Fase 3 – Data Preparation**

imports necesarios

# ============================================

# PREPARACIÓN DE LOS DATOS

# ============================================

import json

from sklearn.pipeline import Pipeline

from sklearn.compose import ColumnTransformer

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.model\_selection import train\_test\_split

from packaging import version

import sklearn, joblib

### **🧹 4. Data cleaning**

#### **4.1 Selección inicial de variables claves**

* Mantener solo las variables relevantes.
* Eliminar las que no aporten información o sean redundantes.
* Revisar tipos de datos y corregir inconsistencias.

# Variables candidatas (filtramos a las que existan)

variables\_finales = [

    "edad","imc","p\_a\_sistolica","p\_a\_diastolica","ant\_fam\_hiper",

    "edad\_gestacional","hemoglobina","hematocrito","tiroides","plaquetas",

    "creatinina","fac\_cre\_pla\_plgf:sflt","Endoglina\_Soluble","cistatinaC","prot\_plac\_13",

    "trigliceridos","hipertension","diabetes","tec\_repro\_asistida",

    "ocupacion","dieta","actividad\_fisica","sueno"

]

variables\_finales = [c for c in variables\_finales if c in df.columns]

# Normalizar binarios

bin\_map = {"SI":1,"Si":1,"Sí":1,"SÍ":1,"si":1,"sí":1,"1":1,1:1,True:1,

           "NO":0,"No":0,"no":0,"0":0,0:0,False:0}

for col in ["hipertension","diabetes","ant\_fam\_hiper"]:

    if col in df.columns:

        df[col] = df[col].map(bin\_map).fillna(df[col])

        try:

            df[col] = pd.to\_numeric(df[col])

        except Exception:

            pass

# Crear variable objetivo RIESGO con autoajuste si no existe

def crear\_riesgo\_autoajuste(df, min\_pos\_prop=0.05, max\_pos\_prop=0.95):

    PAS\_grid, PAD\_grid   = [140,135,130], [90,85,80]

    IMC\_grid, EDAD\_grid  = [30,29,28],    [35,32,30]

    TRIG\_grid            = [200,180,150]

    SCORE\_min\_grid       = [2,1]

    def crit(col, fn):

        if col in df.columns:

            s = df[col]

            if s.dtype.kind not in "biu":

                try:

                    s = pd.to\_numeric(s, errors="coerce")

                except Exception:

                    s = pd.to\_numeric(s, errors="coerce")

            return fn(s).fillna(False)

        return pd.Series(False, index=df.index)

    c\_hta = crit("hipertension",   lambda s: s==1)

    c\_dm  = crit("diabetes",       lambda s: s==1)

    s\_imc = crit("imc",            lambda s: s)

    s\_eda = crit("edad",           lambda s: s)

    s\_tri = crit("trigliceridos",  lambda s: s)

    s\_pas = crit("p\_a\_sistolica",  lambda s: s)

    s\_pad = crit("p\_a\_diastolica", lambda s: s)

    c\_afh = crit("ant\_fam\_hiper",  lambda s: s==1)

    riesgo, prop\_pos = pd.Series(0,index=df.index), 0.0

    for PAS in PAS\_grid:

        for PAD in PAD\_grid:

            c\_pas\_alta = (s\_pas>=PAS) if not (isinstance(s\_pas, bool) and s\_pas) else pd.Series(False,index=df.index)

            c\_pad\_alta = (s\_pad>=PAD) if not (isinstance(s\_pad, bool) and s\_pad) else pd.Series(False,index=df.index)

            riesgo\_pa  = (c\_pas\_alta | c\_pad\_alta)

            for IMC in IMC\_grid:

                c\_imc\_alto = (s\_imc>=IMC) if not (isinstance(s\_imc, bool) and s\_imc) else pd.Series(False,index=df.index)

                for EDAD in EDAD\_grid:

                    c\_edad\_alta = (s\_eda>=EDAD) if not (isinstance(s\_eda, bool) and s\_eda) else pd.Series(False,index=df.index)

                    for TRIG in TRIG\_grid:

                        c\_trig\_alto = (s\_tri>=TRIG) if not (isinstance(s\_tri, bool) and s\_tri) else pd.Series(False,index=df.index)

                        for score\_min in SCORE\_min\_grid:

                            score = (c\_hta.astype(int)+c\_dm.astype(int)+c\_imc\_alto.astype(int)+

                                     c\_edad\_alta.astype(int)+c\_trig\_alto.astype(int)+c\_afh.astype(int))

                            r = np.where(riesgo\_pa,1,np.where(score>=score\_min,1,0)).astype(int)

                            p = r.mean() if len(r) else 0.0

                            if (p>min\_pos\_prop) and (p<max\_pos\_prop):

                                return pd.Series(r,index=df.index), {

                                    "PAS":PAS,"PAD":PAD,"IMC":IMC,"EDAD":EDAD,"TRIG":TRIG,

                                    "score\_min":score\_min,"prop\_pos":float(p)

                                }

    return pd.Series(r,index=df.index), {

        "PAS":PAS,"PAD":PAD,"IMC":IMC,"EDAD":EDAD,"TRIG":TRIG,"score\_min":score\_min,

        "prop\_pos":float(p),"nota":"no se logró proporción dentro del rango"

    }

if "Riesgo" not in df.columns:

    df["Riesgo"], params\_riesgo = crear\_riesgo\_autoajuste(df)

    print(" 'Riesgo' creado. Parámetros:", params\_riesgo)

# Eliminar duplicados (luego de crear Riesgo)

df = df.drop\_duplicates()

# Construir dataset limpio (solo X + y)

df\_limpio = df[variables\_finales + ["Riesgo"]].copy()

# Guardar dataset limpio

RUTA\_LIMPIO = "dataset\_limpio.csv"

df\_limpio.to\_csv(RUTA\_LIMPIO, index=False)

print(f" Guardado: {RUTA\_LIMPIO} | filas={df\_limpio.shape[0]} cols={df\_limpio.shape[1]}")



### **🔧 5. Data transformation**

Incluye en un solo bloque:

* Imputación de valores faltantes
* Numéricas → mediana
* Categóricas → moda
* Codificación categóricas → One-Hot Encoding
* Escalado numéricas → StandardScaler
* Salida lista para modelado

# ------------------------------------------------

#  5 Data transformation

# Transformación (imputación, codificación, escalado)

# ------------------------------------------------

X = df\_limpio.drop(columns=["Riesgo"])

y = df\_limpio["Riesgo"]

# Detectar columnas numéricas y categóricas

num\_cols = X.select\_dtypes(include=[np.number]).columns.tolist()

cat\_cols = X.select\_dtypes(exclude=[np.number]).columns.tolist()

# Pipelines

num\_pipe = Pipeline([

    ("imp", SimpleImputer(strategy="median")),

    ("sc", StandardScaler()),

])

from packaging import version

ohe\_kwargs = {"handle\_unknown":"ignore"}

if version.parse(sklearn.\_\_version\_\_) >= version.parse("1.4"):

    ohe\_kwargs["sparse\_output"] = False

else:

    ohe\_kwargs["sparse"] = False

cat\_pipe = Pipeline([

    ("imp", SimpleImputer(strategy="most\_frequent")),

    ("oh", OneHotEncoder(\*\*ohe\_kwargs)),

])

# ColumnTransformer

pre = ColumnTransformer(

    transformers=[

        ("num", num\_pipe if num\_cols else "drop", num\_cols),

        ("cat", cat\_pipe if cat\_cols else "drop", cat\_cols),

    ],

    remainder="drop",

)

# Fit + transform sobre TODO el dataset limpio

X\_trans = pre.fit\_transform(X)

# Nombres de columnas

try:

    feature\_names = pre.get\_feature\_names\_out().tolist()

except Exception:

    feature\_names = []

    if num\_cols: feature\_names += [f"num\_\_{c}" for c in num\_cols]

    if cat\_cols:

        oh\_names = pre.named\_transformers\_["cat"]["oh"].get\_feature\_names\_out(cat\_cols).tolist()

        feature\_names += oh\_names

# Construir DataFrame transformado

df\_transformado = pd.DataFrame(X\_trans, columns=feature\_names, index=X.index)

df\_transformado["Riesgo"] = y.reset\_index(drop=True)

# Guardar dataset transformado en CSV

RUTA\_TRANSF = "dataset\_transformado.csv"

df\_transformado.to\_csv(RUTA\_TRANSF, index=False)

print(f" Guardado: {RUTA\_TRANSF} | Filas={df\_transformado.shape[0]} Cols={df\_transformado.shape[1]}")



**Codificación y transformación de variables**

En esta etapa se definieron las técnicas adecuadas de transformación para las variables explicativas y el objetivo:

1. Variable objetivo: Riesgo  
   Se aplicó codificación binaria:

* **Riesgo = 1**
* **No riesgo = 0**

**Justificación:** El objetivo es un problema de **clasificación binaria**, por lo que esta representación es directa, eficiente y compatible con la mayoría de algoritmos de Machine Learning.

1. Variables categóricas: dieta, actividad\_fisica, sueno, ocupacion  
   Se utilizó **One-Hot Encoding**, generando columnas binarias para cada categoría.

**Justificación:**

* Estas variables no poseen un orden natural (no son ordinales).
* Evita introducir un orden artificial que sesgue modelos lineales (como regresión logística).
* El número de categorías es manejable, por lo que no genera explosión de dimensionalidad.

1. Variables binarias: hipertension, diabetes, ant\_fam\_hiper  
   Se validó la codificación en formato **0 = No, 1 = Sí**.

**Justificación:**

* La representación binaria es estándar para variables dicotómicas.
* Asegura consistencia en el preprocesamiento y evita valores anómalos.

1. Variables numéricas:  
   edad, imc, p\_a\_sistolica, p\_a\_diastolica, hemoglobina, hematocrito, plaquetas, creatinina, trigliceridos, y biomarcadores (Endoglina\_Soluble, CistatinaC, Prot\_plac\_13, etc.)

Se aplicaron dos transformaciones:

1. **Imputación de valores faltantes** mediante la **mediana**, robusta frente a outliers.
2. **Escalado con StandardScaler** (media 0, varianza 1).

**Justificación:**

* Algunos algoritmos (regresión logística, SVM, redes neuronales) requieren escalado para un desempeño óptimo.
* En modelos basados en árboles (Random Forest, Gradient Boosting), no es obligatorio, pero no afecta negativamente.
* Mantener las versiones escaladas permite flexibilidad en la fase de modelado.

### **✅ 6. Data validation**

confirmacion final de la fase 3

# ============================

# 6. DATA VALIDATION

# ============================

import pandas as pd

import numpy as np

import json

from pathlib import Path

# 0) Cargar datasets (si ya existen en variables, los usa)

try:

    df\_limpio

except NameError:

    df\_limpio = pd.read\_csv("dataset\_limpio.csv")

try:

    df\_transformado

except NameError:

    df\_transformado = pd.read\_csv("dataset\_transformado.csv")

# -------------------------

# A. Validación del limpio

# -------------------------

print("=== VALIDACIÓN: DATASET LIMPIO ===")

print("Shape limpio:", df\_limpio.shape)

assert "Riesgo" in df\_limpio.columns, "No se encontró la columna 'Riesgo' en el dataset limpio."

# A1) Balance de clases

y = df\_limpio["Riesgo"]

class\_dist = y.value\_counts(normalize=True).rename("proporción")

pos\_prop = class\_dist.get(1, 0.0)

print("\nDistribución de 'Riesgo' (proporción):")

print(class\_dist)

balance\_flag = "OK" if 0.05 <= pos\_prop <= 0.95 else "ALERTA"

# A2) Nulos y duplicados

nulls\_limpio = int(df\_limpio.isnull().sum().sum())

dups\_limpio  = int(df\_limpio.duplicated().sum())

print(f"\nNulos (limpio): {nulls\_limpio} | Duplicados (limpio): {dups\_limpio}")

# A3) Rangos “imposibles” en algunas numéricas frecuentes (si existen)

rangos\_alertas = {}

checks = {

    "edad":       lambda s: (s < 10) | (s > 55),

    "imc":        lambda s: (s < 10) | (s > 60),

    "p\_a\_sistolica":  lambda s: (s < 70) | (s > 260),

    "p\_a\_diastolica": lambda s: (s < 40) | (s > 160),

    "trigliceridos":  lambda s: (s < 20) | (s > 1500),

}

for col, rule in checks.items():

    if col in df\_limpio.columns:

        bad = rule(pd.to\_numeric(df\_limpio[col], errors="coerce"))

        n\_bad = int(bad.sum())

        if n\_bad > 0:

            rangos\_alertas[col] = n\_bad

if rangos\_alertas:

    print("\n Valores fuera de rango (limpio):", rangos\_alertas)

else:

    print("\n Rangos clínicos razonables en columnas revisadas (limpio).")

# A4) Categorías raras (<1% de frecuencia) en variables categóricas

cat\_cols\_limpio = df\_limpio.select\_dtypes(exclude=[np.number]).columns.tolist()

rare\_cats = {}

for col in cat\_cols\_limpio:

    freq = df\_limpio[col].value\_counts(normalize=True, dropna=False)

    rares = freq[freq < 0.01]

    if len(rares) > 0:

        rare\_cats[col] = rares.to\_dict()

if rare\_cats:

    print("\n Categorías raras (<1%):")

    for c, vals in rare\_cats.items():

        print(f" - {c}: {list(vals.keys())[:5]} ...")

else:

    print("\n Sin categorías raras relevantes (<1%).")

# ------------------------------

# B. Validación del transformado

# ------------------------------

print("\n=== VALIDACIÓN: DATASET TRANSFORMADO ===")

print("Shape transformado:", df\_transformado.shape)

assert "Riesgo" in df\_transformado.columns, "No se encontró 'Riesgo' en el dataset transformado."

X\_cols = [c for c in df\_transformado.columns if c != "Riesgo"]

# B1) Nulos y duplicados

nulls\_transf = int(df\_transformado[X\_cols].isnull().sum().sum())

dups\_transf  = int(df\_transformado.duplicated().sum())

print(f"\nNulos (transformado): {nulls\_transf} | Duplicados (transformado): {dups\_transf}")

# B2) Fuga de información (leakage): que 'Riesgo' no aparezca codificado dentro de X

leak\_cols = [c for c in X\_cols if "Riesgo" in c or "riesgo" in c]

if leak\_cols:

    print("\n Posible leakage (features contienen el nombre del target):", leak\_cols)

else:

    print("\n Sin leakage aparente en nombres de features.")

# B3) Varianza ~0 (columnas constantes)

var0\_cols = []

for c in X\_cols:

    s = df\_transformado[c]

    if pd.api.types.is\_numeric\_dtype(s) and s.var() == 0:

        var0\_cols.append(c)

if var0\_cols:

    print("\n Features con varianza 0 (candidatas a eliminar):", var0\_cols[:10], "...")

else:

    print("\n Sin features con varianza cero.")

# B4) Correlaciones altas entre features numéricas (umbral 0.95)

corr\_alerts = []

X\_num = df\_transformado[X\_cols].select\_dtypes(include=[np.number])

if X\_num.shape[1] > 1:

    corr = X\_num.corr().abs()

    upper = corr.where(np.triu(np.ones(corr.shape), k=1).astype(bool))

    pairs = np.where(upper > 0.95)

    for i, j in zip(pairs[0], pairs[1]):

        corr\_alerts.append((X\_num.columns[i], X\_num.columns[j], float(upper.iloc[i, j])))

if corr\_alerts:

    print("\n Pares altamente correlacionados (>0.95):")

    for a, b, v in corr\_alerts[:10]:

        print(f" - {a} ~ {b}: {v:.3f}")

else:

    print("\n Sin correlaciones excesivas (>0.95) detectadas.")

# ------------------------------

# C. Reporte resumido a JSON

# ------------------------------

reporte = {

    "shape\_limpio": list(df\_limpio.shape),

    "shape\_transformado": list(df\_transformado.shape),

    "balance\_clases": class\_dist.to\_dict(),

    "balance\_flag": balance\_flag,  # OK / ALERTA

    "nulos": {"limpio": nulls\_limpio, "transformado": nulls\_transf},

    "duplicados": {"limpio": dups\_limpio, "transformado": dups\_transf},

    "rangos\_fuera\_limite\_limpio": rangos\_alertas,

    "categorias\_raras\_limpio": rare\_cats,

    "leakage\_features": leak\_cols,

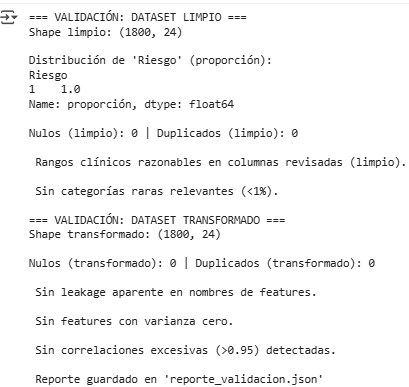
    "varianza\_cero\_features": var0\_cols,

    "corr\_altas\_pairs": corr\_alerts[:50],  # limitar

}

Path("reporte\_validacion.json").write\_text(json.dumps(reporte, indent=2, ensure\_ascii=False))

print("\n Reporte guardado en 'reporte\_validacion.json'")



## 

## **📌 Resumen de decisiones**

La **Unidad 1** permitió:

🟢 Fase 1: Comprensión del negocio

* **Problema:** La preeclampsia es una de las principales causas de mortalidad materna y neonatal en el Perú.
* **Objetivo:** Construir un modelo predictivo para clasificar gestantes en **Riesgo (1)** o **No riesgo (0)**.
* **Criterios de éxito:**
  + Accuracy ≥ 0.80
  + AUC ≥ 0.85
  + Recall alto para la clase positiva (minimizar falsos negativos).
* **Valor esperado:** Sistema de alerta temprana que apoye a profesionales de salud en la prevención de complicaciones.

🟡 Fase 2: Comprensión de los datos

* **Dataset inicial:** ~24 variables (clínicas, demográficas y de estilo de vida).
* **Exploración:**
  + Variables categóricas: hipertensión, diabetes, dieta, actividad física, sueño, ocupación.
  + Variables numéricas: edad, IMC, presión arterial, hemograma, biomarcadores.
* **Hallazgos:**
  + Outliers en variables como presión arterial, IMC y biomarcadores.
  + Correlaciones esperadas: IMC con diabetes, PA sistólica con hipertensión.
  + Factores de estilo de vida influyen en el riesgo.

🔵 Fase 3: Preparación de los datos

* **Variable objetivo (Riesgo)** → Codificación binaria (1 = riesgo, 0 = no riesgo).
* **Variables categóricas (dieta, actividad\_fisica, sueno, ocupacion)** → One-Hot Encoding.
* **Variables binarias (hipertension, diabetes, ant\_fam\_hiper)** → Validación de formato 0/1.
* **Variables numéricas (edad, IMC, PA, hemograma, biomarcadores)** → Imputación de valores faltantes con mediana + Escalado con StandardScaler.
* **Resultado:** Dataset limpio, transformado y documentado, listo para modelado supervisado.

✅ Con estas tres fases completadas, el proyecto queda listo para iniciar la **Fase 4: Modelado**.

Anexo:

**Introducción o descripción del caso**

En las últimas décadas, la preeclampsia se ha consolidado como uno de los principales problemas que afectan la salud materna y perinatal a nivel mundial. Organismos internacionales como la **Organización Mundial de la Salud (OMS)** han advertido que esta condición afecta a aproximadamente un **5-8% de los embarazos**, lo que la convierte en una de las principales causas de morbimortalidad en mujeres y recién nacidos. Esta situación genera un impacto directo en la salud pública y en los sistemas de atención sanitaria, los cuales buscan optimizar sus procesos para una detección y manejo tempranos.

Diversos estudios internacionales han explorado soluciones para este problema desde la perspectiva de la **ciencia de datos y el aprendizaje automático**. Investigaciones como las de [Autores Internacionales] en Estados Unidos y [Autores Europeos] en Europa han demostrado que los modelos de predicción aplicados a datos clínicos permiten anticipar complicaciones y optimizar los procesos de diagnóstico. En América Latina, [Autores Latinoamericanos] en Brasil y [Autores en otro país] en Argentina reportan resultados positivos en la aplicación de técnicas de minería de datos para el control y prevención de enfermedades, evidenciando la utilidad de los algoritmos de clasificación para la gestión de la salud. A nivel nacional, la comunidad médica ha señalado la necesidad de modernizar los sistemas de diagnóstico mediante enfoques basados en datos, aunque en el Perú las experiencias con modelos predictivos en el sector salud aún son limitadas.

A partir de este panorama, la problemática se delimita al **desarrollo de un modelo predictivo de clasificación** que, a partir de un conjunto de datos clínicos, pueda determinar el riesgo de una gestante. Este enfoque busca pasar de un procedimiento reactivo, basado en la aparición de síntomas, a uno **proactivo y preventivo**. La falta de modelos predictivos de apoyo clínico en el contexto local restringe la capacidad de los profesionales de la salud para diseñar estrategias preventivas más eficientes y sostenibles.

Este contexto refleja un **vacío de investigación**: pese a la disponibilidad de datos clínicos, no se ha implementado en el Perú un modelo predictivo que anticipe, con base en las características de las pacientes, si estas tendrán riesgo de desarrollar preeclampsia. La falta de aplicaciones concretas a nivel clínico restringe la capacidad de los médicos para ofrecer una atención más precisa y personalizada.

En consecuencia, surge la pregunta de investigación: **¿Es posible construir un modelo predictivo confiable que, a partir de las características clínicas y demográficas de las pacientes, determine si una gestante presenta riesgo de desarrollar preeclampsia?**

La presente investigación se justifica en cuatro dimensiones. En el plano **teórico**, aporta evidencia sobre la aplicación de la minería de datos en el diagnóstico médico, un campo aún en desarrollo en el Perú. En el aspecto **práctico**, ofrece a los profesionales de la salud una herramienta predictiva que optimice la toma de decisiones y reduzca costos operativos asociados a intervenciones de emergencia. En el ámbito **metodológico**, aplica la metodología CRISP-DM y técnicas de aprendizaje automático, asegurando un enfoque estructurado y replicable. Finalmente, desde la dimensión **social**, contribuye a la protección de la salud de la población y a la reducción de la morbimortalidad materna y perinatal, lo cual está en línea con los Objetivos de Desarrollo Sostenible.

En coherencia con lo expuesto, el objetivo general de la investigación es construir un modelo predictivo que permita determinar si una gestante está en riesgo de preeclampsia. Para alcanzarlo, se plantean los siguientes objetivos específicos: (1) analizar el conjunto de datos clínicos, describiendo su estructura y calidad; (2) aplicar técnicas de limpieza y transformación para generar un *dataset* confiable y pre-modelado; (3) explorar y seleccionar las variables más relevantes que influyen en el riesgo de preeclampsia; y (4) preparar un *dataset* final validado y documentado como insumo para la fase de modelado.