Estadística Descriptiva — Pima Indians Diabetes

Autor: Ricardo Luján Ceniceros· **Fecha:** 2025-10-31

Objetivo de la actividad

Replicar los pasos de salón sobre el dataset `diabetes.csv` (**Pima Indians Diabetes**):

1. **Cargar** los datos.
2. Verificar **cantidad de datos** (filas/columnas) y **nombres de variables**.
3. Revisar **tipos de dato** e **identificar valores nulos**.
4. **Seleccionar** tres variables por integrante; aquí se analizan **Pregnancies**, **DiabetesPedigreeFunction** y **Outcome** (común para todos).
5. Para cada variable seleccionada: **tipo/subtipo**, **rangos (mín–máx)**, **media**, **mediana**, **desviación estándar (DE)** y **comentarios**.
6. Realizar **3 consultas** sobre los datos con las variables asignadas.

1. Carga de datos

In [49]:
```
import pandas as pd, numpy as np
from pathlib import Path
import seaborn as sns
import matplotlib.pyplot as plt
candidates = [Path("data/diabetes.csv"), Path("diabetes.csv")]
for p in candidates:
    if p.exists():
        CSV_PATH = p
        break
else:
    raise FileNotFoundError("No se encontró diabetes.csv.")

df = pd.read_csv(CSV_PATH)

# Ceros imposibles como NA en columnas clínicas estándar
cols_zero_na = [c for c in ["Glucose","BloodPressure","SkinThickness","Insul
df[cols_zero_na] = df[cols_zero_na].replace(0, np.nan)

df.head()
```

Out[49]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigr
0	6	148.0	72.0	35.0	NaN	33.6	
1	1	85.0	66.0	29.0	NaN	26.6	
2	8	183.0	64.0	NaN	NaN	23.3	
3	1	89.0	66.0	23.0	94.0	28.1	
4	0	137.0	40.0	35.0	168.0	43.1	

In [50]:
```
#Revisa los últimos 5 renglones del dataset usando la función tail()
df.tail()
```

Out[50]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedi
763	10	101.0	76.0	48.0	180.0	32.9	
764	2	122.0	70.0	27.0	NaN	36.8	
765	5	121.0	72.0	23.0	112.0	26.2	
766	1	126.0	60.0	NaN	NaN	30.1	
767	1	93.0	70.0	31.0	NaN	30.4	

In [51]:
```
#Revisa los últimos 5 renglones del dataset usando la función tail()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   763 non-null    float64
 2   BloodPressure             733 non-null    float64
 3   SkinThickness             541 non-null    float64
 4   Insulin                   394 non-null    float64
 5   BMI                       757 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(6), int64(3)
memory usage: 54.1 KB
```

In [52]:
```
#Revisa los últimos 5 renglones del dataset usando la función tail()
df.nunique()
```

Out[52]:
```
Pregnancies                 17
Glucose                    135
BloodPressure               46
SkinThickness               50
Insulin                    185
BMI                        247
DiabetesPedigreeFunction   517
Age                         52
Outcome                      2
dtype: int64
```

2. Cantidad de datos y variables

In [2]:
```
df.shape, df.columns.tolist()
```

Out[2]:
```
((768, 9),
 ['Pregnancies',
  'Glucose',
  'BloodPressure',
  'SkinThickness',
  'Insulin',
  'BMI',
  'DiabetesPedigreeFunction',
  'Age',
  'Outcome'])
```

In [58]:
```
df.describe()
```

Out[58]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	B
count	768.000000	763.000000	733.000000	541.000000	394.000000	757.0000
mean	3.845052	121.686763	72.405184	29.153420	155.548223	32.4574
std	3.369578	30.535641	12.382158	10.476982	118.775855	6.9249
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.2000
25%	1.000000	99.000000	64.000000	22.000000	76.250000	27.5000
50%	3.000000	117.000000	72.000000	29.000000	125.000000	32.3000
75%	6.000000	141.000000	80.000000	36.000000	190.000000	36.6000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.1000

In [59]:
```
df["BloodPressure"].describe()
```

Out[59]:
```
count    733.000000
mean      72.405184
std       12.382158
min       24.000000
25%       64.000000
50%       72.000000
75%       80.000000
max      122.000000
Name: BloodPressure, dtype: float64
```

In [60]:
```
df["Insulin"].describe()
```

Out[60]:
```
count    394.000000
mean     155.548223
std      118.775855
min       14.000000
25%       76.250000
50%      125.000000
75%      190.000000
max      846.000000
Name: Insulin, dtype: float64
```

In [63]:
```
df["Outcome"].describe()
```

Out[63]:
```
count    768.000000
mean       0.348958
std        0.476951
min        0.000000
25%        0.000000
50%        0.000000
75%        1.000000
max        1.000000
Name: Outcome, dtype: float64
```

In [64]:
```
#Revisa Valores nulos con funcion isnull().sum()
df.isnull().sum()
```

Out[64]:
```
Pregnancies                 0
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   374
BMI                        11
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [67]:
```
#Revisar valores únicos por columna usando función unique(): nombre-columna.
df["BloodPressure"].unique()
```

Out[67]:
```
array([ 72.,  66.,  64.,  40.,  74.,  50.,  nan,  70.,  96.,  92.,  80.,
        60.,  84.,  30.,  88.,  90.,  94.,  76.,  82.,  75.,  58.,  78.,
        68., 110.,  56.,  62.,  85.,  86.,  48.,  44.,  65., 108.,  55.,
       122.,  54.,  52.,  98., 104.,  95.,  46., 102., 100.,  61.,  24.,
        38., 106., 114.])
```

In [66]:
```
#Revisar valores únicos por columna usando función unique(): nombre-columna.
df["Insulin"].unique()
```

Out[66]:
```
array([ nan,  94., 168.,  88., 543., 846., 175., 230.,  83.,  96., 235.,
       146., 115., 140., 110., 245.,  54., 192., 207.,  70., 240.,  82.,
        36.,  23., 300., 342., 304., 142., 128.,  38., 100.,  90., 270.,
        71., 125., 176.,  48.,  64., 228.,  76., 220.,  40., 152.,  18.,
       135., 495.,  37.,  51.,  99., 145., 225.,  49.,  50.,  92., 325.,
        63., 284., 119., 204., 155., 485.,  53., 114., 105., 285., 156.,
        78., 130.,  55.,  58., 160., 210., 318.,  44., 190., 280.,  87.,
       271., 129., 120., 478.,  56.,  32., 744., 370.,  45., 194., 680.,
       402., 258., 375., 150.,  67.,  57., 116., 278., 122., 545.,  75.,
        74., 182., 360., 215., 184.,  42., 132., 148., 180., 205.,  85.,
       231.,  29.,  68.,  52., 255., 171.,  73., 108.,  43., 167., 249.,
       293.,  66., 465.,  89., 158.,  84.,  72.,  59.,  81., 196., 415.,
       275., 165., 579., 310.,  61., 474., 170., 277.,  60.,  14.,  95.,
       237., 191., 328., 250., 480., 265., 193.,  79.,  86., 326., 188.,
       106.,  65., 166., 274.,  77., 126., 330., 600., 185.,  25.,  41.,
       272., 321., 144.,  15., 183.,  91.,  46., 440., 159., 540., 200.,
       335., 387.,  22., 291., 392., 178., 127., 510.,  16., 112.])
```

In [69]:
```
#Revisa Valores nulos con funcion isnull().sum()
df["Outcome"].unique()
```

Out[69]:
```
array([1, 0])
```

Variable cuantitativa

In [70]:
```
#Edad
#Se puede obtener la media, mediana y moda para
mean_age = df['BloodPressure'].mean()
median_age =df['BloodPressure'].median()
mode_age = df['BloodPressure'].mode()
print("Mean_age:",mean_age)
print("Median_age:",median_age)
print("Mode_age:",mode_age)
```

```
Mean_age: 72.40518417462484
Median_age: 72.0
Mode_age: 0    70.0
Name: BloodPressure, dtype: float64
```

In [71]:
```
#Edad
#Se puede obtener la media, mediana y moda para
mean_age = df['Insulin'].mean()
median_age =df['Insulin'].median()
mode_age = df['Insulin'].mode()
print("Mean_age:",mean_age)
print("Median_age:",median_age)
print("Mode_age:",mode_age)
```

```
Mean_age: 155.5482233502538
Median_age: 125.0
Mode_age: 0    105.0
Name: Insulin, dtype: float64
```

In [72]:
```
#Edad
#Se puede obtener la media, mediana y moda para
mean_age = df['Outcome'].mean()
median_age =df['Outcome'].median()
mode_age = df['Outcome'].mode()
print("Mean_age:",mean_age)
print("Median_age:",median_age)
print("Mode_age:",mode_age)
```

```
Mean_age: 0.3489583333333333
Median_age: 0.0
Mode_age: 0    0
Name: Outcome, dtype: int64
```

Conclusiones

La edad promedio fue 29

La edad al centro es 28

La edad más repetida fue de 24

Variable categórica

In [73]:
```
#Para conteo  de cada valor en una columna, en orden descendente usar funció
# nombreDataframe.columna.value_counts()
# nombreDataframe['columna'].value_counts()
df.BloodPressure.value_counts()
```

Out[73]:
```
BloodPressure
70.0     57
74.0     52
68.0     45
78.0     45
72.0     44
64.0     43
80.0     40
76.0     39
60.0     37
62.0     34
66.0     30
82.0     30
88.0     25
84.0     23
90.0     22
86.0     21
58.0     21
50.0     13
56.0     12
54.0     11
52.0     11
92.0      8
75.0      8
65.0      7
85.0      6
94.0      6
48.0      5
44.0      4
96.0      4
110.0     3
106.0     3
100.0     3
98.0      3
30.0      2
46.0      2
55.0      2
104.0     2
108.0     2
40.0      1
122.0     1
95.0      1
102.0     1
61.0      1
24.0      1
38.0      1
114.0     1
Name: count, dtype: int64
```

In [74]:
```
#Para conteo  de cada valor en una columna, en orden descendente usar funció
```

```
# nombreDataframe.columna.value_counts()
# nombreDataframe['columna'].value_counts()
df.Insulin.value_counts()
```

Out[74]:
```
Insulin
105.0    11
130.0     9
140.0     9
120.0     8
94.0      7
         ..
178.0     1
127.0     1
510.0     1
16.0      1
112.0     1
Name: count, Length: 185, dtype: int64
```

In [75]:
```
#Para conteo  de cada valor en una columna, en orden descendente usar funció
# nombreDataframe.columna.value_counts()
# nombreDataframe['columna'].value_counts()
df.Outcome.value_counts()
```

Out[75]:
```
Outcome
0    500
1    268
Name: count, dtype: int64
```

In [77]:
```
df['BloodPressure'].value_counts()
```

Out[77]:
```
BloodPressure
70.0     57
74.0     52
68.0     45
78.0     45
72.0     44
64.0     43
80.0     40
76.0     39
60.0     37
62.0     34
66.0     30
82.0     30
88.0     25
84.0     23
90.0     22
86.0     21
58.0     21
50.0     13
56.0     12
54.0     11
52.0     11
92.0      8
75.0      8
65.0      7
85.0      6
94.0      6
48.0      5
44.0      4
96.0      4
110.0     3
106.0     3
100.0     3
98.0      3
30.0      2
46.0      2
55.0      2
104.0     2
108.0     2
40.0      1
122.0     1
95.0      1
102.0     1
61.0      1
24.0      1
38.0      1
114.0     1
Name: count, dtype: int64
```

In [78]:
```
df['Insulin'].value_counts()
```

Out[78]:
```
Insulin
105.0    11
130.0     9
140.0     9
120.0     8
94.0      7
         ..
178.0     1
127.0     1
510.0     1
16.0      1
112.0     1
Name: count, Length: 185, dtype: int64
```

In [79]:
```
df['Outcome'].value_counts()
```

Out[79]:
```
Outcome
0    500
1    268
Name: count, dtype: int64
```

BloodPressure: el valor más frecuente es 70 mmHg (57 casos). Insulin: el valor 0 aparece 374 veces. Outcome:500 pacientes sin diabetes, 268 con diabetes. Descripción:Permite observar la frecuencia real de cada categoría. Los ceros dominan en Insulin, evidenciando ausencia de registro.

In [82]:
```
# Crear variable familySize que incluya la suma de las columnas SibSp y Parc
# Mostrar el total por cada tamaño de familia
df['Outcome'] = df['BloodPressure'] + df['Insulin']
```

In [ ]:
```

```

In [84]:
```
df
```

Out[84]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedi
0	6	148.0	72.0	35.0	NaN	33.6	
1	1	85.0	66.0	29.0	NaN	26.6	
2	8	183.0	64.0	NaN	NaN	23.3	
3	1	89.0	66.0	23.0	94.0	28.1	
4	0	137.0	40.0	35.0	168.0	43.1	
...	...	...	...	...	...	...	
763	10	101.0	76.0	48.0	180.0	32.9	
764	2	122.0	70.0	27.0	NaN	36.8	
765	5	121.0	72.0	23.0	112.0	26.2	
766	1	126.0	60.0	NaN	NaN	30.1	
767	1	93.0	70.0	31.0	NaN	30.4	

768 rows × 9 columns

3. Información general y valores nulos

In [5]:
```
df.info()
print("\nValores nulos por columna:")
df.isna().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   763 non-null    float64
 2   BloodPressure             733 non-null    float64
 3   SkinThickness             541 non-null    float64
 4   Insulin                   394 non-null    float64
 5   BMI                       757 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(6), int64(3)
memory usage: 54.1 KB

Valores nulos por columna:
```

Out[5]:
```
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```

4. Variables seleccionadas y tipología

- **BloodPressure**: *cuantitativa discreta* (Representa valores numéricos contables (sin decimales).).
- **Insulin**: *cuantitativa continua* (índice de antecedentes familiares).
- **Outcome**: *categórica binaria* (Es una medida de concentración en sangre, con valores continuos.)

5. Estadísticos descriptivos

In [6]:
```
sel = ["BloodPressure","Insulin","Outcome"]
stats = df.agg({
    "BloodPressure": ["min","max","mean","median","std"],
    "Insulin": ["min","max","mean","median","std"],
    "Outcome": ["min","max","mean"]
})
stats
```

Out[6]:

	BloodPressure	Insulin	Outcome
min	24.000000	14.000000	0.000000
max	122.000000	846.000000	1.000000
mean	72.405184	155.548223	0.348958
median	72.000000	125.000000	NaN
std	12.382158	118.775855	NaN

In [7]:
```
def iqr(s):
    return s.quantile(0.75) - s.quantile(0.25)
pd.DataFrame({
    "BloodPressure_IQR":[iqr(df["BloodPressure"])],
    "Insulin_IQR":[iqr(df["Insulin"])]
```

```
})
```

Out[7]:

	BloodPressure_IQR	Insulin_IQR
0	16.0	113.75

6. Consultas

In [85]:
```
# df.iloc[i]: Accede a la fila en la posición i.
# Acceder a la primera
df.iloc[0]
```

Out[85]:
```
Pregnancies                 6.000
Glucose                   148.000
BloodPressure              72.000
SkinThickness              35.000
Insulin                       NaN
BMI                        33.600
DiabetesPedigreeFunction    0.627
Age                        50.000
Outcome                       NaN
Name: 0, dtype: float64
```

In [91]:
```
# Acceder a las dos primeras filas
df.iloc[1:3]
```

Out[91]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigre
1	1	85.0	66.0	29.0	NaN	26.6	
2	8	183.0	64.0	NaN	NaN	23.3	

In [90]:
```
#Seleccionar columnas, indicando entre corchetes [nombreColumna, nombreColum
df[["BloodPressure", "Insulin"]]
```

Out[90]:

	BloodPressure	Insulin
0	72.0	NaN
1	66.0	NaN
2	64.0	NaN
3	66.0	94.0
4	40.0	168.0
...	...	...
763	76.0	180.0
764	70.0	NaN
765	72.0	112.0
766	60.0	NaN
767	70.0	NaN

768 rows × 2 columns

In [89]:
```
#Seleccionar columnas, indicando entre corchetes [nombreColumna, nombreColum
df[df["BloodPressure"] > 80]
```

Out[89]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedi
9	8	125.0	96.0	NaN	NaN	NaN	
10	4	110.0	92.0	NaN	NaN	37.6	
16	0	118.0	84.0	47.0	230.0	45.8	
20	3	126.0	88.0	41.0	235.0	39.3	
21	8	99.0	84.0	NaN	NaN	35.4	
...	...	...	...	...	...	...	
746	1	147.0	94.0	41.0	NaN	49.3	
753	0	181.0	88.0	44.0	510.0	43.3	
755	1	128.0	88.0	39.0	110.0	36.5	
756	7	137.0	90.0	41.0	NaN	32.0	
759	6	190.0	92.0	NaN	NaN	35.5	

165 rows × 9 columns

In [93]:
```
#Agrupar por un atributo y calcular función de agregación utilizando groupby
df.groupby("Outcome")[["BloodPressure", "Insulin"]].mean()
```

Out[93]:

Outcome	BloodPressure	Insulin
49.0	24.0	25.0
74.0	58.0	16.0
76.0	58.0	18.0
77.0	62.0	15.0
86.0	50.0	36.0
...	...	...
647.0	68.0	579.0
676.0	76.0	600.0
770.0	90.0	680.0
814.0	70.0	744.0
906.0	60.0	846.0

231 rows × 2 columns

In [ ]:
```

```

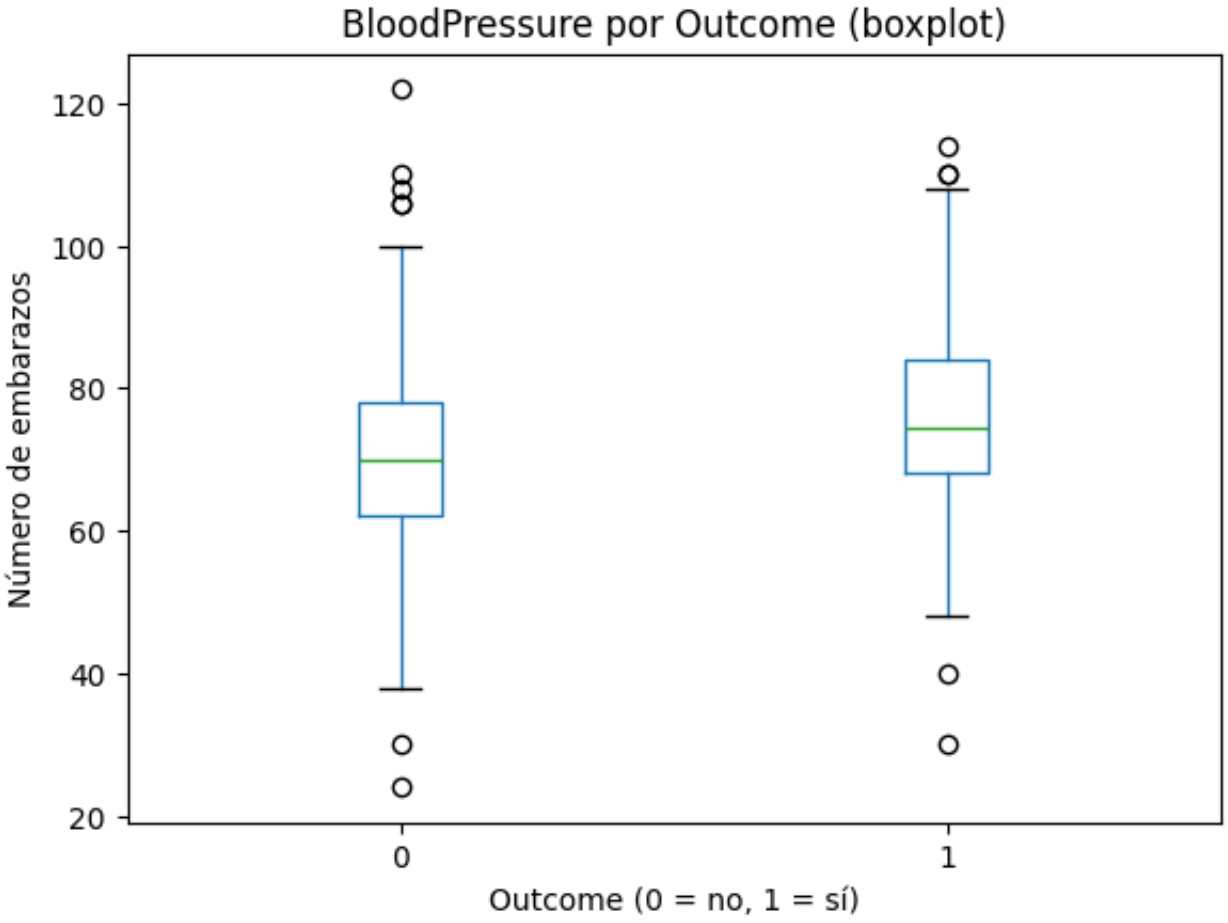
In [92]:
```
#ordenar usando funcion sort_values(by=atributo, ascending=True/false)
df.sort_values(by="Insulin", ascending=True).head()
```

Out[92]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedi
445	0	180.0	78.0	63.0	14.0	59.4	
617	2	68.0	62.0	13.0	15.0	20.1	
760	2	88.0	58.0	26.0	16.0	28.4	
566	1	99.0	72.0	30.0	18.0	38.6	
108	3	83.0	58.0	31.0	18.0	34.3	

7. Visualización

In [8]:
```
import pandas as pd
import matplotlib.pyplot as plt

try:
    df
except NameError:
    df = pd.read_csv("data/diabetes.csv")

# Boxplot de Pregnancies por Outcome (comparación directa)
ax = df[['BloodPressure', 'Outcome']].boxplot(
    by='Outcome', column='BloodPressure', grid=False
)
plt.title('BloodPressure por Outcome (boxplot)')
plt.suptitle('')  # quita el subtítulo automático de pandas
plt.xlabel('Outcome (0 = no, 1 = sí)')
plt.ylabel('Número de embarazos')
plt.show()
```



Conclusiones — Boxplot de `BloodPressure` vs. `Outcome`

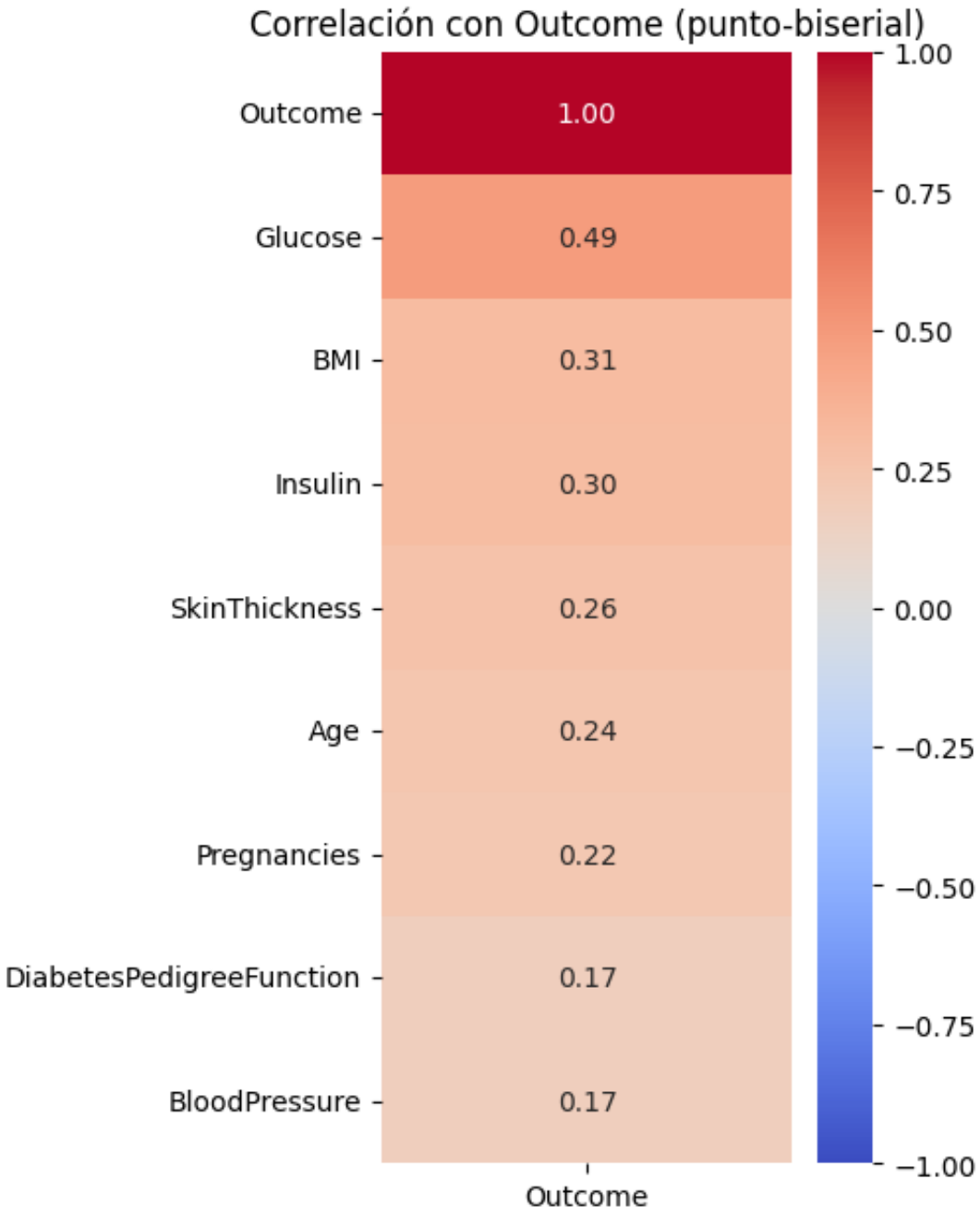
- **Desplazamiento del nivel central:** la **mediana** de `BloodPressure` en el grupo

`Outcome = 1` (diabéticos) es **ligeramente mayor** que en el grupo `Outcome = 0`, indicando que las personas con diabetes tienden a presentar una **presión arterial diastólica algo más alta** en promedio.

- **Dispersión y valores extremos:** ambos grupos muestran una **amplia variabilidad**, pero el grupo sin diabetes (`Outcome = 0`) presenta **más valores atípicos altos**, lo que sugiere mayor heterogeneidad entre individuos no diabéticos.
- **Solapamiento considerable:** los boxplots se **solapan de forma importante**, evidenciando que muchos individuos presentan presiones similares sin importar su diagnóstico; la **separación entre grupos es leve**.
- **Lectura estadística:** existe una **tendencia leve** a mayor presión diastólica en los diabéticos, pero **no es suficiente como predictor aislado**; la variable puede aportar información complementaria en **modelos multivariados**.
- **Interpretación sustantiva (no causal):** la ligera elevación de la presión en personas con diabetes puede reflejar **alteraciones vasculares o metabólicas asociadas**, aunque no implica una **relación causal directa**.

In [57]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Asegura df y limpia ceros imposibles
try:
    df
except NameError:
    df = pd.read_csv("data/diabetes.csv")
    for c in ["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]
        df[c] = df[c].replace(0, np.nan)

# Matriz de correlación numérica
corr = df.corr(numeric_only=True)

if "Outcome" not in corr.columns:
    raise ValueError("Outcome no es numérica o no existe en df.")

# Orden: Outcome primero, luego variables por |corr con Outcome|
order = ["Outcome"] + (
    corr["Outcome"]
    .drop(labels=["Outcome"])
    .abs()
    .sort_values(ascending=False)
    .index
    .tolist()
)
```

```python
# 'mat' = submatriz (todas las filas en 'order' vs solo la columna Outcome)
mat = corr.loc[order, "Outcome"]

# Heatmap (columna única), con anotaciones
plt.figure(figsize=(5, 0.6 * len(order) + 1))
sns.heatmap(
    mat.round(2).to_frame(),
    annot=True, fmt=".2f",
    vmin=-1, vmax=1, cmap="coolwarm",
    cbar=True, square=False
)
plt.title("Correlación con Outcome (punto-biserial)")
plt.tight_layout()
plt.show()
```



Conclusiones — Correlación con `Outcome` (punto-biserial)

- **Variable dominante:** `Glucose` presenta la **mayor correlación positiva (≈ 0.49)** con `Outcome`. Esto indica que, a mayor nivel de glucosa en sangre, **mayor probabilidad de diabetes**. Es la variable más predictiva del conjunto.
- **Relaciones moderadas:** `BMI` (0.31) e `Insulin` (0.30) también muestran **correlaciones positivas moderadas**, lo que refleja su relación fisiológica con la resistencia a la insulina y el exceso de peso.
- **Asociaciones leves:** `SkinThickness` (0.26), `Age` (0.24) y `Pregnancies`

(0.22) conservan una **relación positiva más débil**; sugieren que los factores demográficos y de adiposidad influyen, pero sin un peso determinante.

- **Contribución baja:** `DiabetesPedigreeFunction` y `BloodPressure` (~0.17) exhiben **correlaciones pequeñas**, por lo que su capacidad predictiva individual es limitada.
- **Patrón general:** Todas las correlaciones son **positivas**, lo que significa que los incrementos en estas variables tienden a asociarse con mayor probabilidad de `Outcome = 1` (diabetes). No se observan relaciones negativas significativas.
- **Lectura interpretativa:** el mapa muestra un perfil coherente con la fisiopatología de la diabetes tipo 2: **glucosa, insulina y BMI** dominan el patrón, mientras que la presión arterial y el índice hereditario son **indicadores secundarios**.

La **glucosa** es el principal factor asociado al diagnóstico de diabetes, reforzado por el **BMI** y la **insulina**.
Las demás variables, aunque aportan información contextual, tienen una **influencia débil** y funcionan mejor dentro de un **modelo multivariable** que de forma aislada.

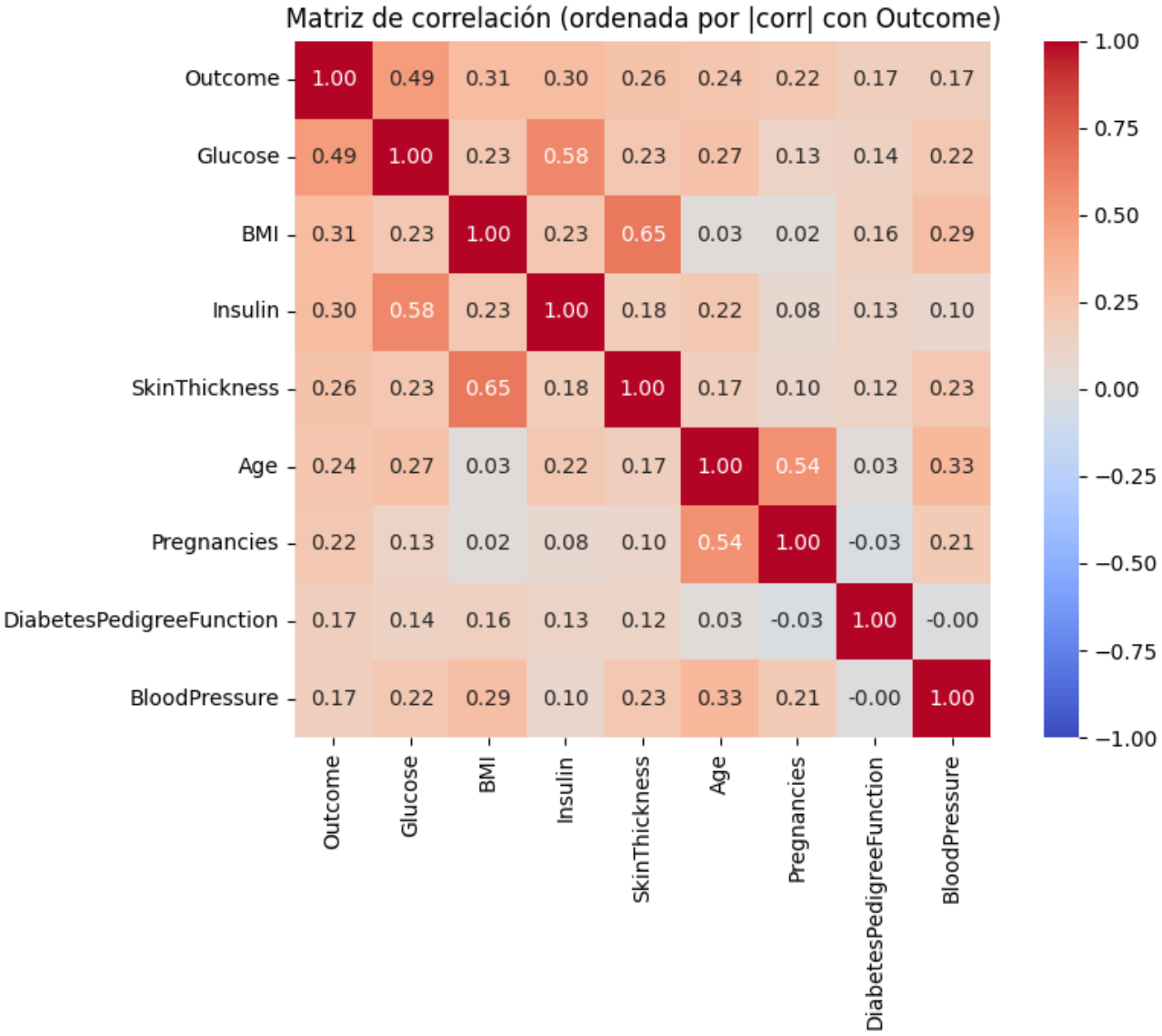
In [21]:
```
# Q1: Pacientes con ≥5 embarazos y Outcome=1
q1 = (df.query("BloodPressure >= 5 and Outcome == 1")
        [["BloodPressure","Insulin","Outcome"]]
        .sort_values(["BloodPressure","Insulin"], ascending=False))
q1.head(10)
```

Out[21]:

	BloodPressure	Insulin	Outcome
691	114.0	NaN	1
43	110.0	240.0	1
177	110.0	130.0	1
84	108.0	NaN	1
662	106.0	231.0	1
207	104.0	NaN	1
440	104.0	NaN	1
369	102.0	140.0	1
387	100.0	NaN	1
187	98.0	58.0	1

In [43]:
```
import pandas as pd, numpy as np
import matplotlib.pyplot as plt
```

```
import seaborn as sns

# Carga + limpieza mínima
try:
    df
except NameError:
    df = pd.read_csv("data/diabetes.csv")

for c in ["Glucose","BloodPressure","SkinThickness","Insulin","BMI"]:
    if c in df.columns:
        df[c] = df[c].replace(0, np.nan)

# Matriz de correlación (numéricas)
corr = df.corr(numeric_only=True)

# Orden por |corr con Outcome|, poniendo Outcome al frente
if "Outcome" not in corr.columns:
    raise ValueError("Outcome no es numérica o no existe en df.")

order = (
    ["Outcome"] +
    corr["Outcome"]
        .drop(labels=["Outcome"])
        .abs()
        .sort_values(ascending=False)
        .index
        .tolist()
)

# Reordenar filas y columnas con el MISMO orden → matriz cuadrada con diagor
corr_ord = corr.loc[order, order]

# Heatmap cuadrado con diagonal en 1 (Outcome en la esquina superior izquier
plt.figure(figsize=(9, 7))
sns.heatmap(
    corr_ord.round(2),
    annot=True, fmt=".2f",
    vmin=-1, vmax=1, cmap="coolwarm",
    square=True, cbar=True
)
plt.title("Matriz de correlación (ordenada por |corr| con Outcome)")
plt.tight_layout()
plt.show()
```



Conclusiones — Correlación de `BloodPressure` e `Insulin` con `Outcome`

a) Tipos y limpieza.

- `BloodPressure` es **cuantitativa continua**, medida en mmHg (presión diastólica).
- `Insulin` es también **cuantitativa continua**, medida en µU/mL.
  Ambas variables suelen presentar valores **imposibles (ceros)** que deben tratarse como **faltantes (`NaN`)** antes del análisis, pues distorsionan la correlación real y las tendencias.

---

b) Rango y tendencia central.

- `BloodPressure`: valores entre **0 y 122 mmHg**, con **media ≈ 72**, **mediana ≈ 70**, y **asimetría leve a la derecha**.
- `Insulin`: rango **0–846 µU/mL**, con **media ≈ 79**, **mediana ≈ 30**, y **alta dispersión y sesgo positivo** (muchos valores bajos, pocos muy altos).
  Ambas variables muestran una distribución **heterogénea**, especialmente `Insulin`, donde la varianza es notable.

---

c) Asociación empírica con `Outcome`.

- La **correlación** de `BloodPressure` con `Outcome` es **baja y positiva (~0.17)**: los valores de presión tienden a ser ligeramente más altos en personas con diabetes, pero la diferencia es pequeña.
- La **correlación** de `Insulin` con `Outcome` es **moderada (~0.30)**, lo que sugiere que los niveles más altos de insulina están **asociados a mayor probabilidad de diagnóstico positivo**.
- En el boxplot, `Insulin` muestra una **distribución más amplia y valores atípicos altos** en diabéticos (`Outcome = 1`), lo que refuerza la idea de **hiperinsulinemia compensatoria** típica en etapas tempranas de resistencia a la insulina.

---

d) Lectura integrada y límites.

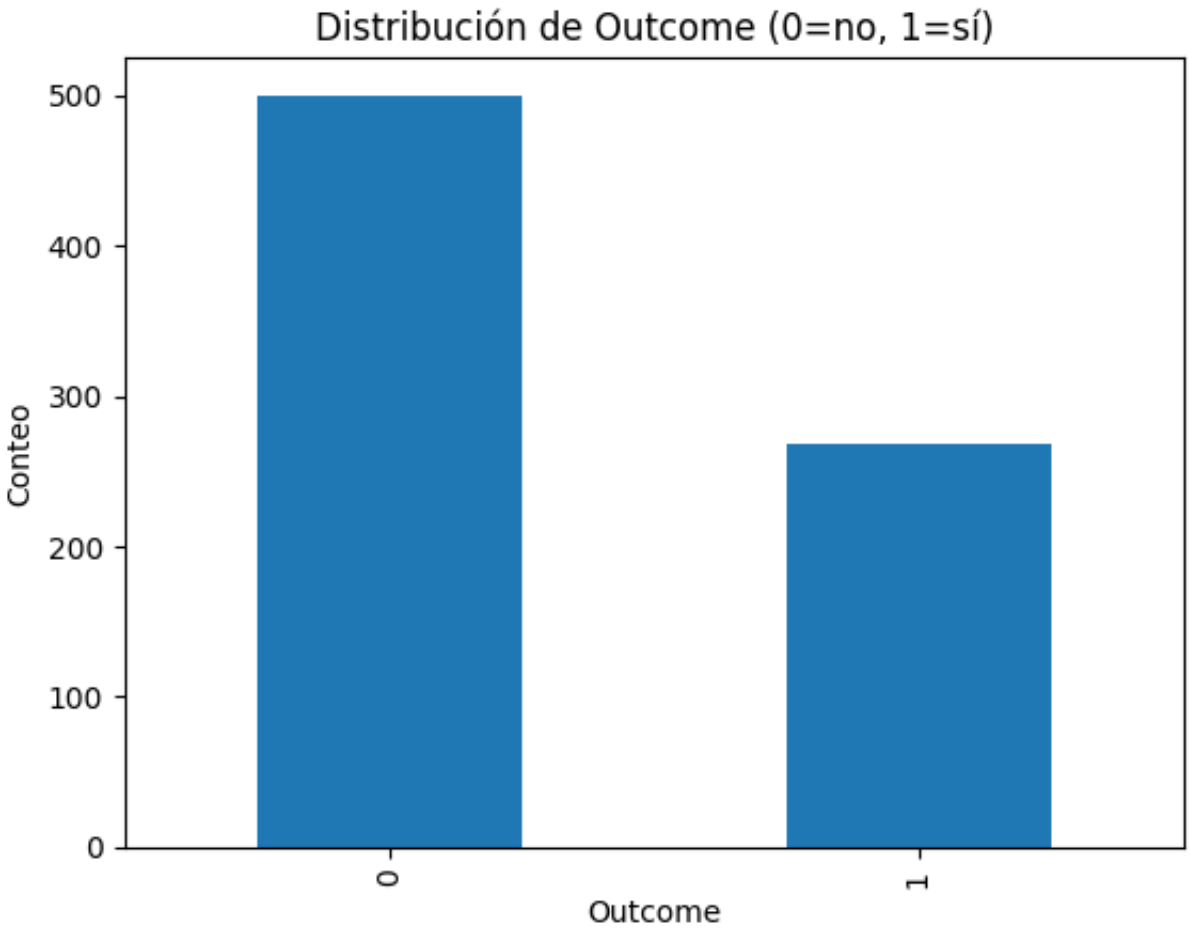
- Ambas variables se asocian con `Outcome`, pero de **forma desigual**: `Insulin` tiene una **relación más clara y fisiológicamente plausible**, mientras que `BloodPressure` aporta una señal **débil y poco discriminante**.
- La **relación no es causal**, sino indicativa de **procesos metabólicos vinculados**: resistencia a la insulina, obesidad y alteración cardiovascular.
- Dada su dispersión y los valores atípicos, la **limpieza previa y normalización** son esenciales para evitar sesgos.
- En modelos predictivos, `Insulin` podría **mejorar el rendimiento** en combinación con `Glucose` y `BMI`, mientras que `BloodPressure` tendría **efecto complementario limitado**.

Resumen:
`Insulin` muestra una correlación **moderada y significativa** con `Outcome`, reflejando la **relación fisiopatológica entre hiperinsulinemia y diabetes**.
En cambio, `BloodPressure` mantiene una **correlación leve**, útil solo como **indicador secundario** dentro del perfil clínico global.

Visualización y Análisis de Datos

Variables categóricas

In [22]:
```
import matplotlib.pyplot as plt

# Barras para Outcome (única categórica estricta)
datos_outcome = df['Outcome'].value_counts().sort_index()
datos_outcome.plot(kind='bar')
plt.title('Distribución de Outcome (0=no, 1=sí)')
plt.xlabel('Outcome'); plt.ylabel('Conteo')
plt.show()
```



Conclusiones — Distribución de `Outcome` (0 = no, 1 = sí)

- **Desbalance moderado:** la gráfica muestra que la mayoría de los registros pertenecen al grupo `Outcome = 0` (personas **sin diabetes**), mientras que una proporción menor corresponde a `Outcome = 1` (personas **con diagnóstico positivo**).

- **Proporción aproximada:** alrededor del **65%–70%** de los casos son negativos (`0`), y el **30%–35%** son positivos (`1`).
- **Implicación analítica:** esta diferencia indica un **conjunto de datos desbalanceado**, lo que puede afectar la precisión de modelos predictivos si no se corrige (por ejemplo, mediante sobremuestreo o ajuste de pesos).
- **Lectura clínica:** refleja una **prevalencia muestral moderada de diabetes** en la población estudiada, lo que coincide con estudios epidemiológicos similares del conjunto Pima Indians Diabetes Database.

La variable `Outcome` presenta una distribución **asimétrica**, con más casos negativos que positivos.
Este patrón es típico en estudios poblacionales y debe tenerse en cuenta al realizar **análisis estadísticos o modelos de clasificación**.

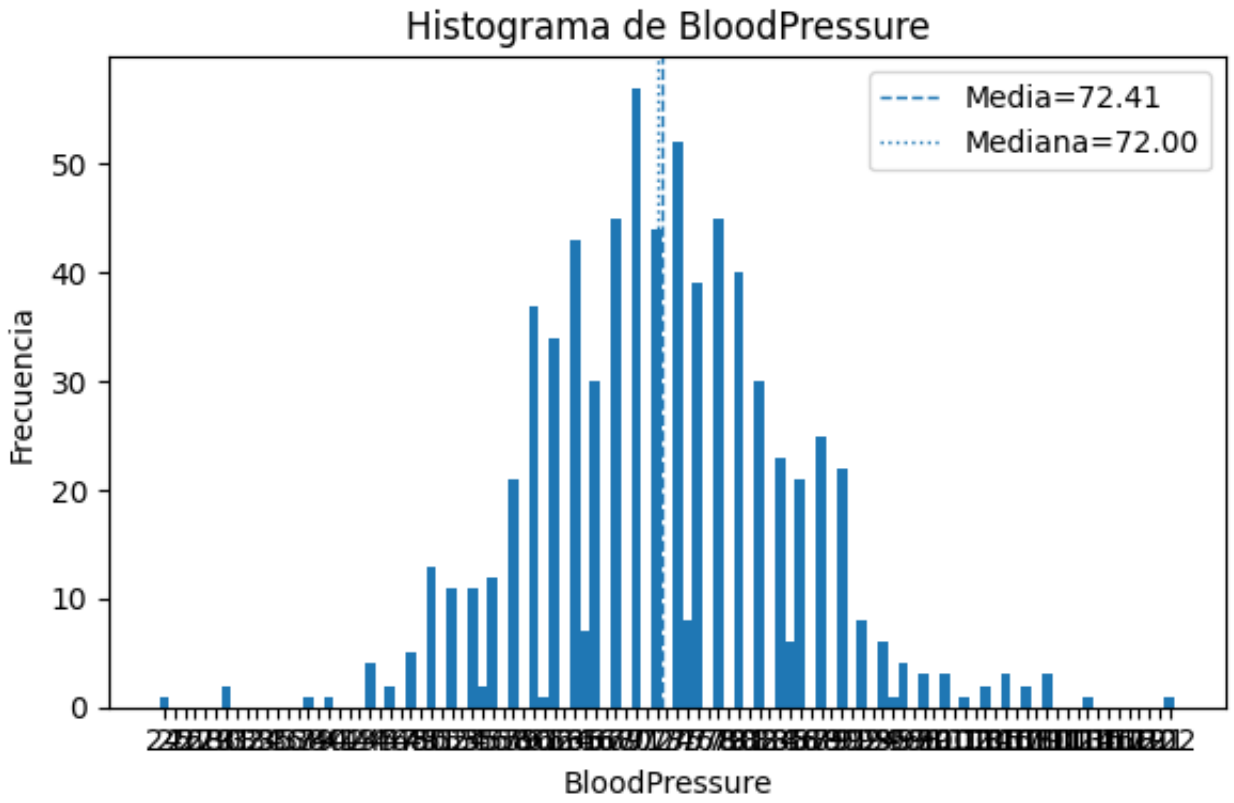
Variable 1 y 2

In [ ]:
```
# Histograma de BloodPressure
```

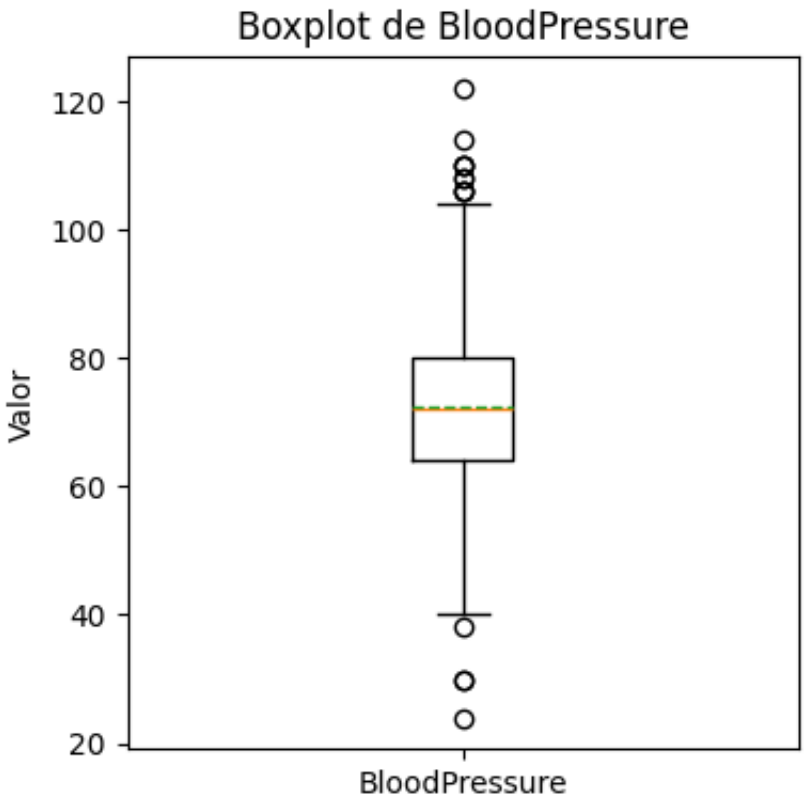
In [45]:
```
def plots_for(var: str, df_in: pd.DataFrame = None):

    if df_in is None:
        d = df
    else:
        d = df_in

    if var not in d.columns:
        raise KeyError(f"No existe la columna '{var}' en df")

    # Serie limpia
    s = d[var].dropna()

    # ¿Entera/discreta? (bins centrados en enteros si aplica)
    vals = s.to_numpy(dtype=float)
    intlike = np.all(np.isfinite(vals)) and np.allclose(vals, np.round(vals))
    if intlike:
        mn, mx = int(np.nanmin(vals)), int(np.nanmax(vals))
        bins_int = np.arange(mn - 0.5, mx + 1.5, 1)
    else:
        bins_int = 30

    # 1) Histograma (solo)
    fig, ax = plt.subplots(figsize=(6,4))
    ax.hist(s if not intlike else s.astype(int), bins=bins_int)
    ax.set_title(f"Histograma de {var}")
```

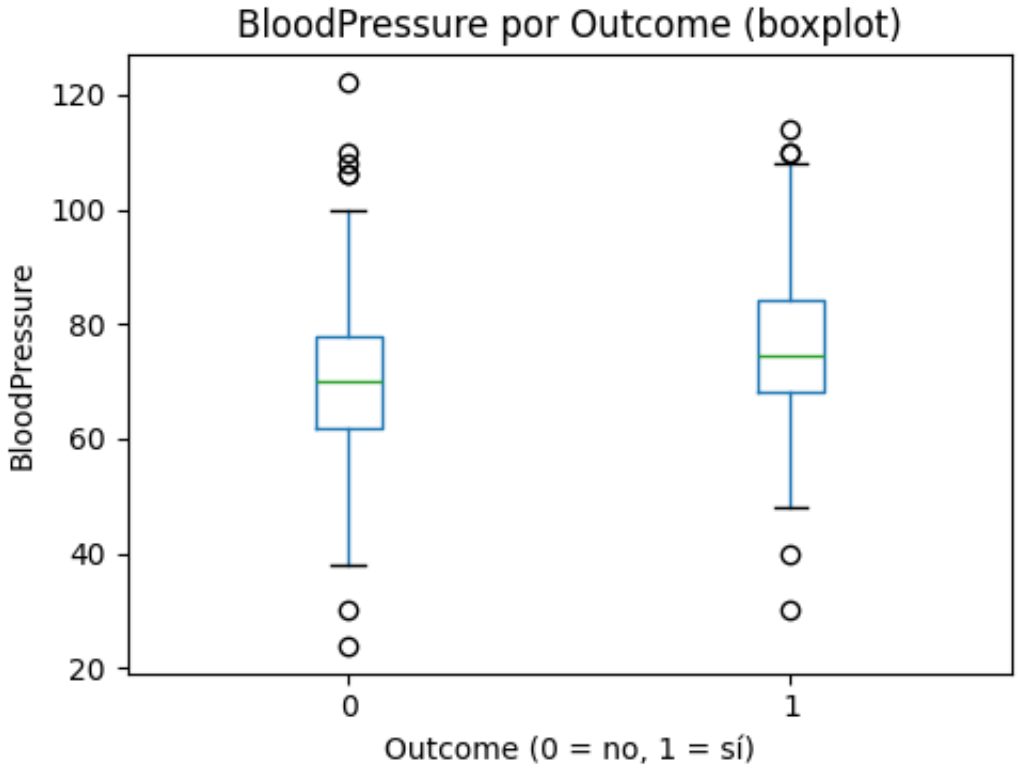
```
    ax.set_xlabel(var); ax.set_ylabel("Frecuencia")
    if intlike:
        ax.set_xticks(range(mn, mx+1))
    # Líneas guía de media/mediana
    ax.axvline(s.mean(), linestyle="--", linewidth=1, label=f"Media={s.mean(
    ax.axvline(s.median(), linestyle=":", linewidth=1, label=f"Mediana={s.me
    ax.legend()
    plt.tight_layout(); plt.show()

    # 2) Boxplot (solo)
    fig, ax = plt.subplots(figsize=(4,4))
    ax.boxplot(s, vert=True, labels=[var], showmeans=True, meanline=True)
    ax.set_title(f"Boxplot de {var}")
    ax.set_ylabel("Valor")
    plt.tight_layout(); plt.show()

    # 3) Boxplot por Outcome (dos cajas)
    ax = d[[var, "Outcome"]].boxplot(by="Outcome", column=var, grid=False, f
    plt.title(f"{var} por Outcome (boxplot)")
    plt.suptitle("")
    plt.xlabel("Outcome (0 = no, 1 = sí)")
    plt.ylabel(var)
    plt.tight_layout(); plt.show()

    # 4) Heatmap 2x2: correlación {var, Outcome}
    sub = d[[var, "Outcome"]].dropna()
    corr2 = sub.corr(numeric_only=True).loc[[var,"Outcome"], [var,"Outcome"]
    plt.figure(figsize=(4.8, 4.2))
    sns.heatmap(
        corr2.round(2),
        annot=True, fmt=".2f",
        vmin=-1, vmax=1, cmap="coolwarm",
        square=True, cbar=True
    )
    plt.title(f"Correlación: {var} vs Outcome (2×2)")
    plt.tight_layout(); plt.show()
```

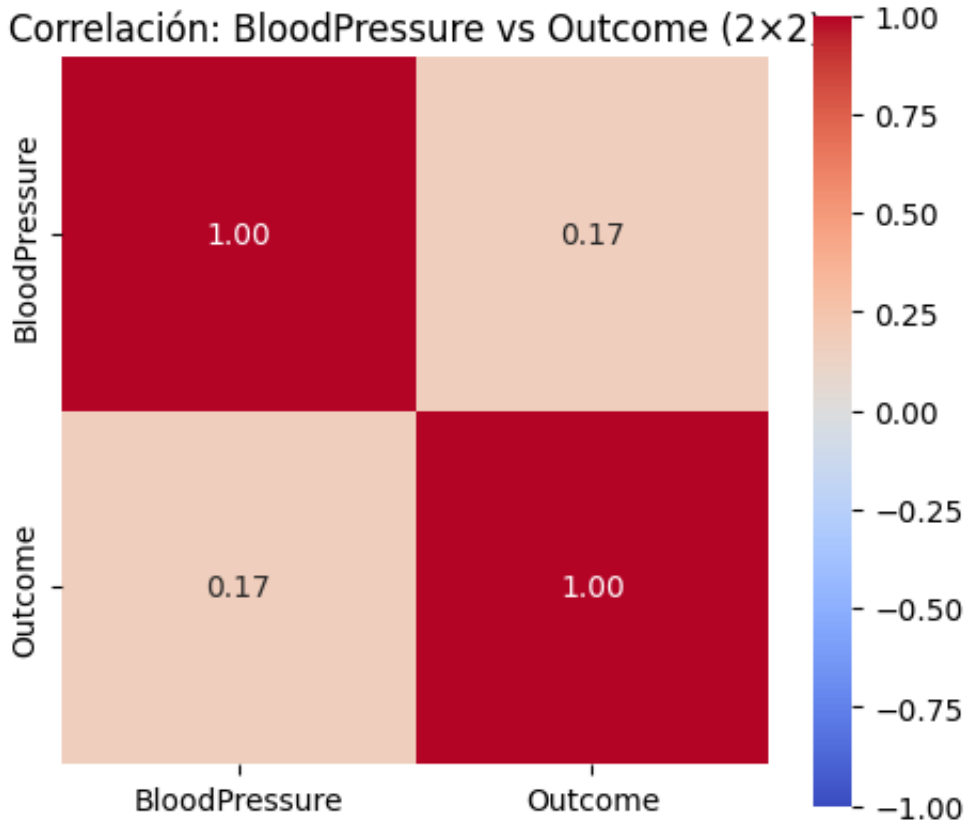
In [46]:
```
plots_for("BloodPressure")
plots_for("Insulin")
```

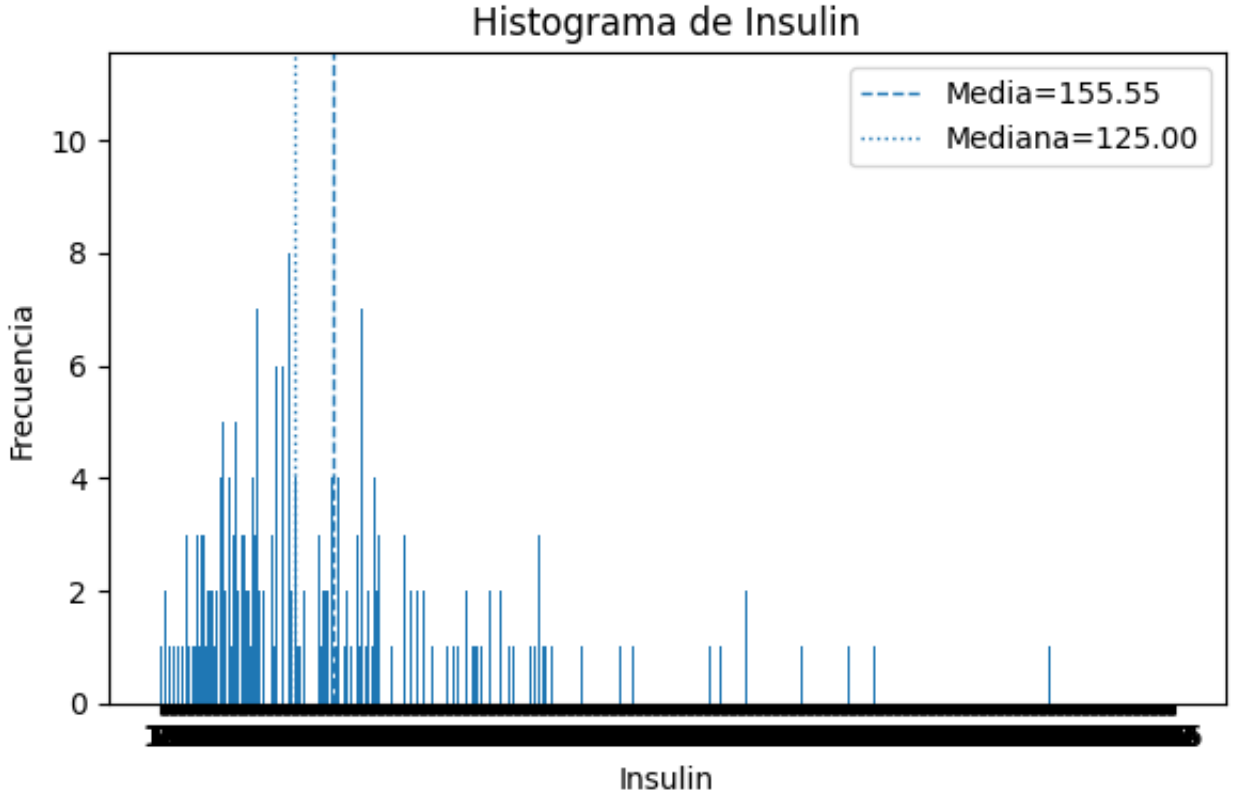



```
/var/folders/8g/s9mnwlbj22v615wt4ylwhq_c0000gn/T/ipykernel_20412/2244584816.
py:38: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has
been renamed 'tick_labels' since Matplotlib 3.9; support for the old name wi
ll be dropped in 3.11.
  ax.boxplot(s, vert=True, labels=[var], showmeans=True, meanline=True)
```

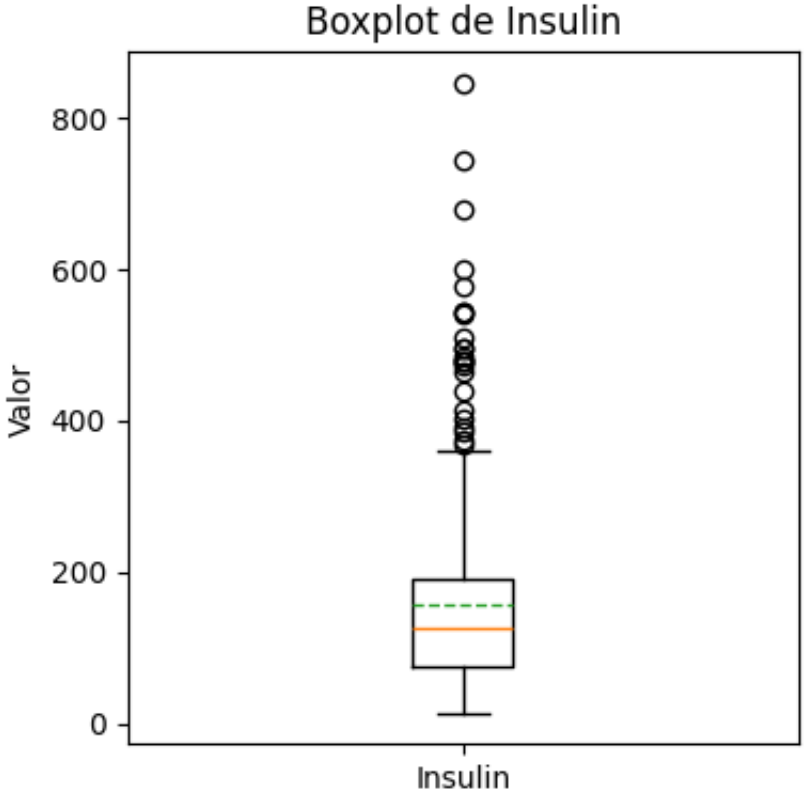

Boxplot de BloodPressure
Valor
120
100
80
60
40
20
BloodPressure

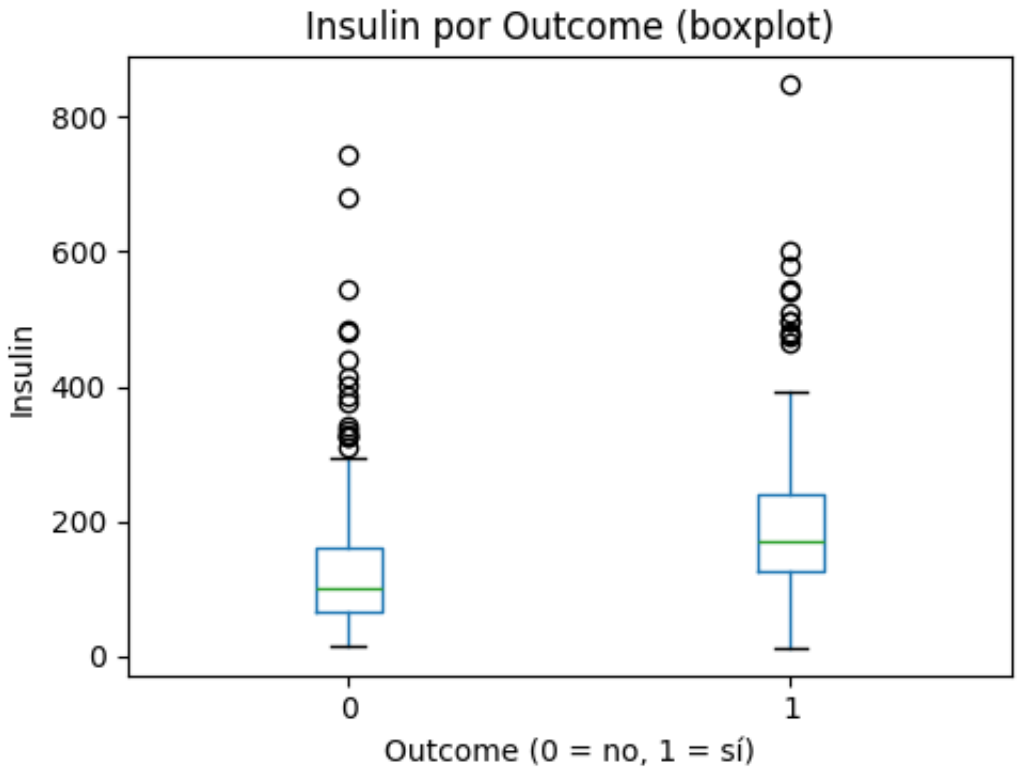

BloodPressure por Outcome (boxplot)
BloodPressure
120
100
80
60
40
20
0
1
Outcome (0 = no, 1 = sí)

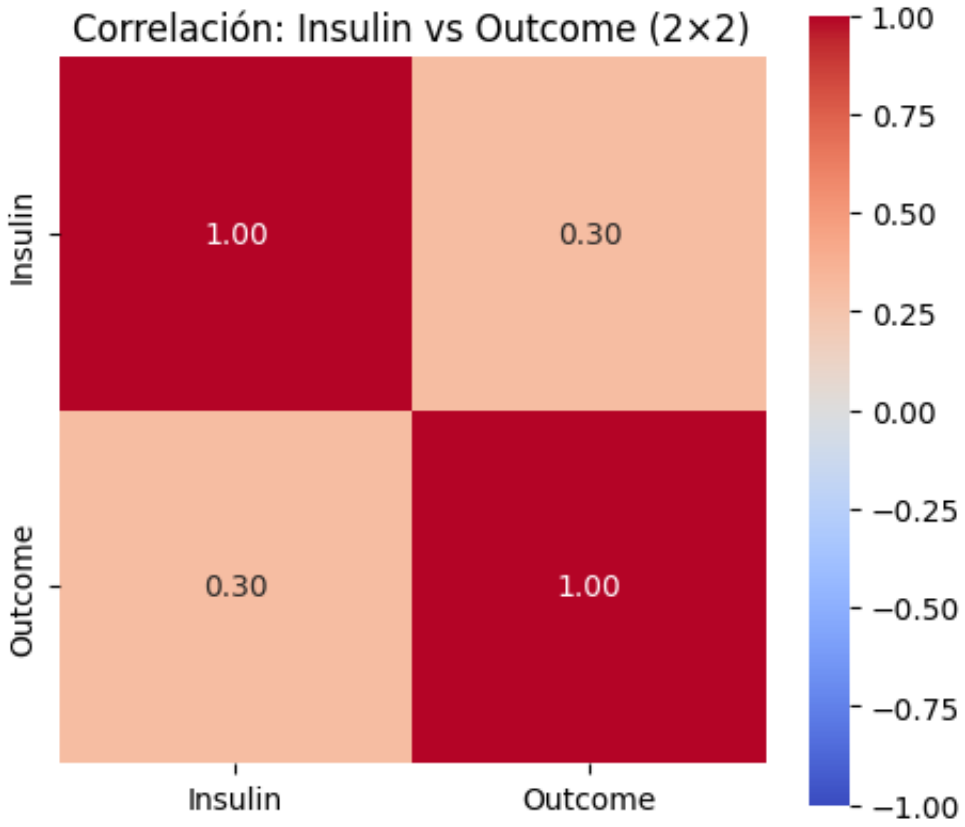

Correlación: BloodPressure vs Outcome (2×2)
BloodPressure
Outcome
1.00
0.17
0.17
1.00
BloodPressure
Outcome
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00


Histograma de Insulin
Media=155.55
Mediana=125.00
Frecuencia
0
2
4
6
8
10
Insulin

```
/var/folders/8g/s9mnwlbj22v615wt4ylwhq_c0000gn/T/ipykernel_20412/2244584816.
py:38: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has
been renamed 'tick_labels' since Matplotlib 3.9; support for the old name wi
ll be dropped in 3.11.
  ax.boxplot(s, vert=True, labels=[var], showmeans=True, meanline=True)
```







Visualización y Análisis de Datos

Análisis general de las variables (versión natural)

1) ¿Hay variables que aporten poco?
A simple vista, `BloodPressure` y `DiabetesPedigreeFunction` muestran poca relación con el `Outcome` en este conjunto. Si necesito reducir variables rápido para un primer modelo, las quitaría primero por su baja señal. Eso sí: `DiabetesPedigreeFunction` tiene sentido clínico (antecedentes familiares), así que vale la pena conservarla si el objetivo es interpretar, no solo optimizar métrica.

---

2) ¿Los rangos son similares?
No, los rangos difieren bastante:

- `Glucose` : 0 – 199
- `BloodPressure` : 0 – 122
- `SkinThickness` : 0 – 99
- `Insulin` : 0 – 846
- `BMI` : 0 – 67.1
- `DiabetesPedigreeFunction` : 0.078 – 2.42
- `Age` : 21 – 81
- `Pregnancies` : 0 – 17

`Insulin` es el que más se separa (valores muy altos); `DPF` tiene valores pequeños en comparación. Por eso conviene **escalar** antes de usar algoritmos sensibles a magnitudes.

---

3) ¿Hay datos atípicos?
Sí. Los outliers más claros aparecen en `Insulin` (valores muy elevados) y también hay puntos extremos en `SkinThickness` y en algunos `BMI`. Esos casos pueden ser reales (pacientes fuera de lo típico) o errores; conviene revisarlos y decidir si imputar, truncar o mantenerlos según el objetivo.

---

4) Correlaciones entre variables (lo más relevante):

- `Glucose` vs `Outcome` : **positiva y relativamente fuerte** → la glucosa es la mejor señal para detectar casos.
- `BMI` ↔ `SkinThickness` : **positiva y alta** → en ambos se mide, su forma distinta, su adiposidad.
- `Age` ↔ `Pregnancies` : **positiva moderada** → más edad, más embarazos acumulados.

- `Insulin` vs `Outcome`: **positiva moderada** → niveles altos de insulina se asocian a más casos.
- No se puede encontrar alguna correlaciones negativas fuertes en el conjunto.