Modelo de regresión

Integrante: Luis Keny Lucero Balvin

```
# Archivo: clientes_intereses_1000_targets.csv
# X: primeras 12 columnas (intereses 1–10)
# Y1: convirtio_30d (CLASIFICACIÓN)   | Y2: ordenes_90d (REGRESIÓN DE
CONTEO)

import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression, PoissonRegressor
from sklearn.metrics import (
    accuracy_score, f1_score, roc_auc_score, classification_report,
confusion_matrix
)
```

1. Dependencias para VIF

```
# Para VIF
!pip -q install statsmodels
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import
variance_inflation_factor
```

2. Parámetro de entrada (ruta del CSV)

```
CSV = "clientes_intereses_1000_targets.csv"
```

3. Carga del dataset

```
# 1) Carga
if not os.path.exists(CSV):
    from google.colab import files
    print(f"No encontré '{CSV}'. Súbelo ahora…")
    uploaded = files.upload()
    if CSV not in uploaded:
        CSV = list(uploaded.keys())[0]
        print("Usando:", CSV)

df = pd.read_csv(CSV)
```

4. Esquema de columnas esperadas

```
# 2) Columnas esperadas
feature_cols = [

"importancia_garantia","sensibilidad_precio","rapidez_envio","calidad_
producto",

"devolucion_flexible","confianza_reviews","sostenibilidad","marca_impo
rtancia",

"variedad_metodos_pago","personalizacion","soporte_postventa","fidelid
ad_programa"
]
y_cls = "convirtio_30d"
y_cnt = "ordenes_90d"

missing = [c for c in feature_cols+[y_cls, y_cnt] if c not in
df.columns]
if missing:
    raise ValueError(f"Faltan columnas en el CSV: {missing}")
```

5. Limpieza mínima y coerción de tipos

```
# 3) Limpieza rápida
X = df[feature_cols].copy()
y1 = df[y_cls].copy()
y2 = df[y_cnt].copy()

# Forzar numérico y tratar NaN por si acaso
for c in feature_cols:
    X[c] = pd.to_numeric(X[c], errors="coerce")
y1 = pd.to_numeric(y1, errors="coerce")
y2 = pd.to_numeric(y2, errors="coerce")
```

6. Diagnóstico de faltantes y relleno

```
# Reporte de faltantes
null_report = pd.DataFrame({
    "faltantes_X": X.isna().sum(),
})
null_report.loc["convirtio_30d","faltantes_Y1"] = y1.isna().sum()
null_report.loc["ordenes_90d","faltantes_Y2"] = y2.isna().sum()
print("=== Faltantes ===")
display(null_report.fillna(0).astype(int))

# Relleno conservador con medianas
X = X.fillna(X.median())
```

```
y1 = y1.fillna(y1.median())
y2 = y2.fillna(y2.median())
```

```
=== Faltantes ===
```

{"summary":"{\n \"name\": \"y2 = y2\",\n \"rows\": 14,\n \"fields\": [\n   {\n   \"column\": \"faltantes_X\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 0,\n   \"max\": 0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"faltantes_Y1\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 0,\n   \"max\": 0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"faltantes_Y2\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 0,\n   \"max\": 0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

7. Columnas constantes o casi constantes

```
# 4) Columnas constantes o con varianza ~0
nuniq = X.nunique()
const_cols = nuniq[nuniq <= 1].index.tolist()
print("Columnas constantes/varianza cero:", const_cols if const_cols
else "Ninguna")
```

```
Columnas constantes/varianza cero: Ninguna
```

8. Chequeo de rangos esperados (1–10)

```
# 5) Rango de features (esperado 1–10)
print("\n=== Rango observado de features ===")
display(pd.DataFrame({"min": X.min(), "max": X.max()}))
```

```
=== Rango observado de features ===
```

{"summary":"{\n \"name\": \"display(pd\",\n \"rows\": 12,\n \"fields\": [\n   {\n   \"column\": \"min\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 0,\n   \"max\": 3,\n   \"num_unique_values\": 4,\n   \"samples\": [\n   2,\n   0,\n   3\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"max\",\n   \"properties\": {\n   \"dtype\": \"number\",\n

\"std\": 0,\n   \"min\": 8,\n   \"max\": 10,\n   \"num_unique_values\": 3,\n   \"samples\": [\n   9,\n   10,\n   8\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

9. Multicolinealidad (VIF)

```
# 6) Multicolinealidad (VIF)
# Escalamos y añadimos constante para VIF
Xs = pd.DataFrame(StandardScaler().fit_transform(X),
columns=feature_cols)
Xs_const = sm.add_constant(Xs, has_constant="add")
vif = pd.Series(
    [variance_inflation_factor(Xs_const.values, i) for i in
range(Xs_const.shape[1])],
    index=["const"] + feature_cols,
    name="VIF"
)
print("\n=== VIF (multicolinealidad) ===")
display(vif.to_frame())
```

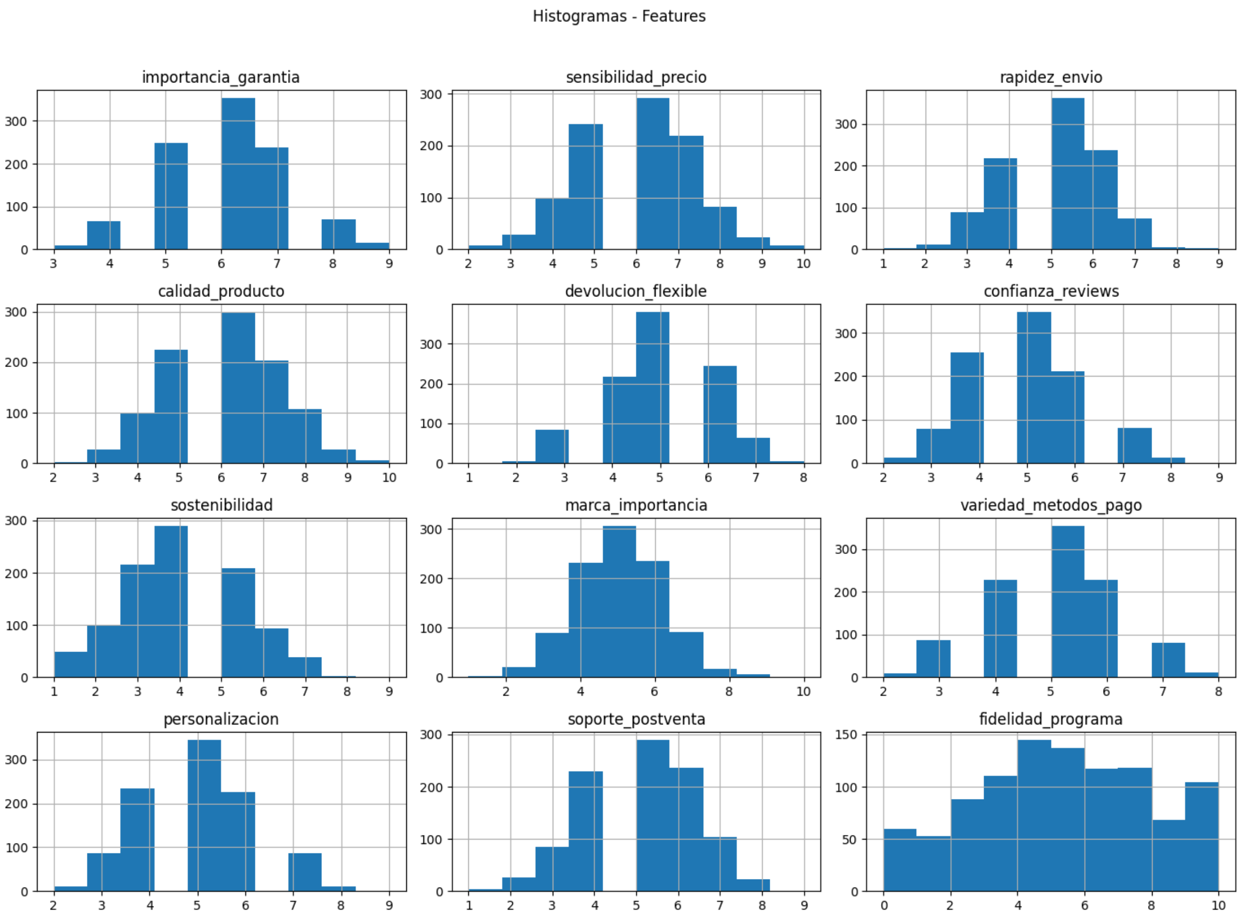
```
=== VIF (multicolinealidad) ===
```

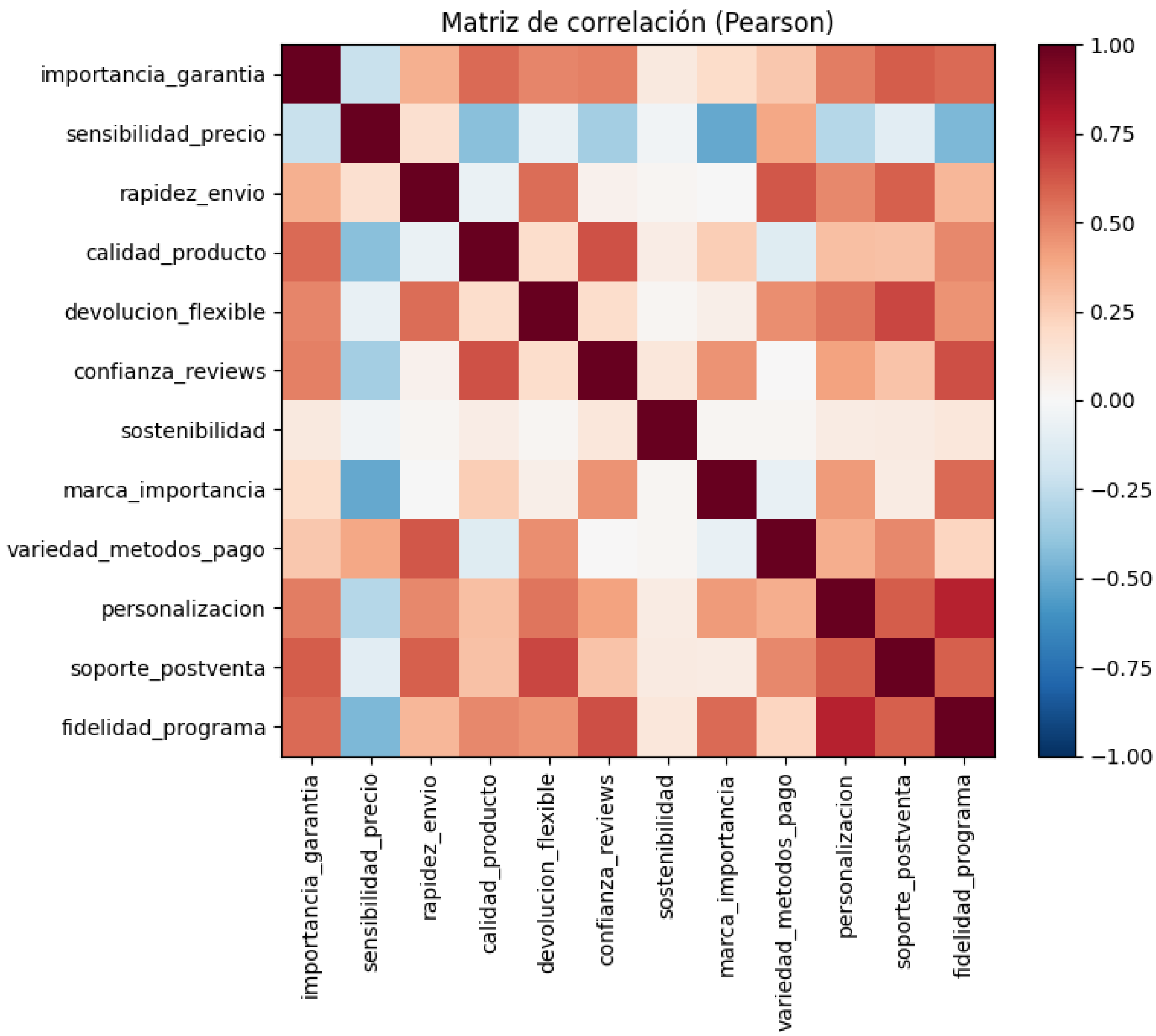
{"summary":"{\n \"name\": \"display(vif\",\n \"rows\": 13,\n \"fields\": [\n   {\n   \"column\": \"VIF\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.9984657436771056,\n   \"min\": 0.9999999999999998,\n   \"max\": 4.994573684954479,\n   \"num_unique_values\": 13,\n   \"samples\": [\n   3.1149816909519226,\n   2.2486339122331422,\n   0.9999999999999998\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

10. Exploratorio: histogramas y correlación

```
# 7) Distribuciones y correlación rápida
axes = X.hist(bins=10, figsize=(14,10))
plt.suptitle("Histogramas - Features", y=1.02); plt.tight_layout();
plt.show()

corr = X.corr(method="pearson")
plt.figure(figsize=(9,7))
im = plt.imshow(corr, cmap="RdBu_r", vmin=-1, vmax=1)
plt.xticks(range(len(corr.columns)), corr.columns, rotation=90)
plt.yticks(range(len(corr.index)), corr.index)
plt.colorbar(im, fraction=0.046, pad=0.04)
plt.title("Matriz de correlación (Pearson)")
plt.tight_layout(); plt.show()
```


Histogramas - Features
importancia_garantia
sensibilidad_precio
rapidez_envio
calidad_producto
devolucion_flexible
confianza_reviews
sostenibilidad
marca_importancia
variedad_metodos_pago
personalizacion
soporte_postventa
fidelidad_programa



11. Naturaleza de los objetivos (Y)

```
# 8) Naturaleza de las Y
print("\n=== Naturaleza de las Y ===")
cls_rate = y1.mean()
zeros_rate = (y2 == 0).mean()
print(f"convirtio_30d -> tasa de positivos = {cls_rate:.3f}")
print(f"ordenes_90d   -> prop. de ceros   = {zeros_rate:.3f},
media={y2.mean():.3f}, var={y2.var():.3f}")
```

```
=== Naturaleza de las Y ===
convirtio_30d -> tasa de positivos = 0.450
ordenes_90d   -> prop. de ceros   = 0.455, media=0.966, var=1.578
```

12. Señal base (clasificación)

```
# 9) Señal base rápida
# Clasificación: Regresión Logística
X_tr, X_te, y1_tr, y1_te = train_test_split(X, y1, test_size=0.2,
random_state=42, stratify=y1)
scaler = StandardScaler().fit(X_tr)
X_tr_s, X_te_s = scaler.transform(X_tr), scaler.transform(X_te)

lr = LogisticRegression(max_iter=1000, class_weight="balanced")
lr.fit(X_tr_s, y1_tr)
p = lr.predict_proba(X_te_s)[:,1]
pred = (p >= 0.5).astype(int)
```

13. Métricas y reporte (clasificación)

```
# === Línea base: Clasificación (Logistic Regression) ===
print("\n=== Línea base: Clasificación (Logistic Regression) ===")

# Probabilidades y predicción por defecto (umbral 0.5)
p = lr.predict_proba(X_te_s)[:, 1]
pred = (p >= 0.5).astype(int)

# Métricas (¡ojo! usar formato o round(), no .round() sobre float)
roc = roc_auc_score(y1_te, p)
f1  = f1_score(y1_te, pred)
acc = accuracy_score(y1_te, pred)

print(f"ROC AUC: {roc:.3f}")
print(f"F1      : {f1:.3f}")
print(f"Accuracy: {acc:.3f}")

print("\nClassification report (umbral 0.5):")
print(classification_report(y1_te, pred, digits=3))

cm = confusion_matrix(y1_te, pred)
print("Confusion matrix:\n", cm)
```

```
=== Línea base: Clasificación (Logistic Regression) ===
ROC AUC: 0.701
F1     : 0.606
Accuracy: 0.655

Classification report (umbral 0.5):
              precision    recall  f1-score   support

           0      0.678     0.709     0.693       110
           1      0.624     0.589     0.606        90
```

```
    accuracy                           0.655       200
   macro avg      0.651     0.649     0.650       200
weighted avg      0.654     0.655     0.654       200

Confusion matrix:
 [[78 32]
 [37 53]]
```

14. Señal base (conteos Poisson)

```
# Conteos: Poisson
X_tr2, X_te2, y2_tr, y2_te = train_test_split(X, y2, test_size=0.2,
random_state=42)
scaler2 = StandardScaler().fit(X_tr2)
X_tr2_s, X_te2_s = scaler2.transform(X_tr2), scaler2.transform(X_te2)

pr = PoissonRegressor(alpha=1e-6, max_iter=1000)  # base
pr.fit(X_tr2_s, y2_tr)
y2_pred = pr.predict(X_te2_s)
```

15. Métricas de conteo y sugerencia de familia

```
rmse = np.sqrt(np.mean((y2_te - y2_pred)**2))
mae  = np.mean(np.abs(y2_te - y2_pred))
print("\n=== Línea base: Regresión de conteo (Poisson) ===")
print("RMSE:", round(rmse,3), " | MAE:", round(mae,3))
print("Media y2_test:", round(y2_te.mean(),3), " | Var y2_test:",
round(y2_te.var(),3))
print("Sugerencia modelo de conteo:",
      "Negativo Binomial/Zero-Inflated" if (y2.var() > y2.mean()*2 or
zeros_rate > 0.4) else "Poisson parece razonable")


=== Línea base: Regresión de conteo (Poisson) ===
RMSE: 0.965  | MAE: 0.711
Media y2_test: 1.075  | Var y2_test: 1.477
Sugerencia modelo de conteo: Negativo Binomial/Zero-Inflated
```