Clasificación de Revenue (Sí/No compra)

integrante: Luis Lucero Balvin

```
# === Colab: Clasificación de Revenue (Sí/No compra) ===

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

from sklearn.model_selection import train_test_split, StratifiedKFold,
cross_val_score
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (
    accuracy_score, f1_score, roc_auc_score, precision_recall_curve,
    roc_curve, average_precision_score, classification_report,
confusion_matrix
)
```

1. Carga robusta del CSV

```
# ---------- 1) Cargar CSV ----------
csv_name = "online_shoppers_intention.csv"
if not os.path.exists(csv_name):
    from google.colab import files
    print(f"No encontré '{csv_name}'. Selecciona tu CSV para
subirlo…")
    uploaded = files.upload()
    if csv_name not in uploaded:
        csv_name = list(uploaded.keys())[0]
        print(f"Usando archivo subido: {csv_name}")

df = None
for enc in ("utf-8", "latin-1", "cp1252"):
    try:
        df = pd.read_csv(csv_name, encoding=enc)
        break
    except Exception:
        pass
if df is None:
    raise RuntimeError("No pude leer el CSV. Verifica
ruta/codificación.")
```

```python
print("Dimensiones:", df.shape)
display(df.head())
```

```
Dimensiones: (12330, 18)
```

{"summary":"{\n \"name\": \"display(df\",\n \"rows\": 5,\n \"fields\": [\n   {\n   \"column\": \"Administrative\",\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\": \"Administrative_Duration\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.0,\n   \"min\": 0.0,\n   \"max\": 0.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   0.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Informational\",\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\": \"Informational_Duration\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.0,\n   \"min\": 0.0,\n   \"max\": 0.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   0.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"ProductRelated\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 3,\n   \"min\": 1,\n   \"max\": 10,\n   \"num_unique_values\": 3,\n   \"samples\": [\n   1\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"ProductRelated_Duration\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 274.5386534209701,\n   \"min\": 0.0,\n   \"max\": 627.5,\n   \"num_unique_values\": 4,\n   \"samples\": [\n   64.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"BounceRates\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.09838699100999075,\n   \"min\": 0.0,\n   \"max\": 0.2,\n   \"num_unique_values\": 4,\n   \"samples\": [\n   0.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"ExitRates\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.06496152707564687,\n   \"min\": 0.05,\n   \"max\": 0.2,\n   \"num_unique_values\": 4,\n   \"samples\": [\n   0.1\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"PageValues\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.0,\n   \"min\": 0.0,\n   \"max\": 0.0,\n

```
\"num_unique_values\": 1,\n        \"samples\": [\n          0.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"SpecialDay\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.0,\n        \"min\": 0.0,\n        \"max\": 0.0,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          0.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Month\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
1,\n        \"samples\": [\n          \"Feb\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"OperatingSystems\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1,\n        \"min\": 1,\n        \"max\": 4,\n
\"num_unique_values\": 4,\n        \"samples\": [\n          2\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Browser\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 1,\n        \"max\": 3,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          1\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Region\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 3,\n
\"min\": 1,\n        \"max\": 9,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"TrafficType\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1,\n        \"min\": 1,\n
\"max\": 4,\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          2\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"VisitorType\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          \"Returning_Visitor\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Weekend\",\n      \"properties\":
{\n        \"dtype\": \"boolean\",\n        \"num_unique_values\": 2,\
n        \"samples\": [\n          true\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Revenue\",\n      \"properties\":
{\n        \"dtype\": \"boolean\",\n        \"num_unique_values\": 1,\
n        \"samples\": [\n          false\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


2. Esquema de columnas y saneamiento de tipos

```
# ---------- 2) Columnas (según tu esquema en inglés) ----------
num_cols = [
```

```
"Administrative","Administrative_Duration","Informational","Informatio
nal_Duration",

"ProductRelated","ProductRelated_Duration","BounceRates","ExitRates","
PageValues","SpecialDay"
]
cat_cols = [

"Month","OperatingSystems","Browser","Region","TrafficType","VisitorTy
pe","Weekend"
]
target = "Revenue"

# Seguridad: coerción de tipos, mapeo booleanos 'TRUE/FALSE'
bool_map = {"TRUE": True, "FALSE": False, "True": True, "False":
False,
            "true": True, "false": False}
if "Weekend" in df.columns and df["Weekend"].dtype == object:
    df["Weekend"] = df["Weekend"].map(bool_map).fillna(df["Weekend"])
if "Revenue" in df.columns and df["Revenue"].dtype == object:
    df["Revenue"] = df["Revenue"].map(bool_map).fillna(df["Revenue"])

# Coerción numérica segura
for c in num_cols:
    if c in df.columns:
        df[c] = pd.to_numeric(df[c], errors="coerce")

# Seleccionar X, y
use_cols = [c for c in num_cols + cat_cols if c in df.columns]
missing = [c for c in num_cols + cat_cols + [target] if c not in
df.columns]
if missing:
    print("⚠ Faltan columnas en el CSV:", missing)

X = df[use_cols].copy()
y = df[target].astype(int)  # Revenue -> 0/1

# Rellenar NaN en numéricas con mediana
for c in [c for c in num_cols if c in X.columns]:
    X[c] = X[c].fillna(X[c].median())
```

3. Partición estratificada de datos

```
# ---------- 3) Split estratificado ----------
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, random_state=42, stratify=y
)
```

4. Preprocesamiento y definición de modelos

```
# ---------- 4) Preprocesamiento + Modelos ----------
numeric_features = [c for c in num_cols if c in X.columns]
categorical_features = [c for c in cat_cols if c in X.columns]

preprocess = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), numeric_features),
        ("cat", OneHotEncoder(handle_unknown="ignore"),
categorical_features),
    ],
    remainder="drop",
)

# Modelo 1: Regresión Logística
log_reg = Pipeline(steps=[
    ("prep", preprocess),
    ("clf", LogisticRegression(max_iter=1000,
class_weight="balanced"))
])

# Modelo 2: Random Forest (baseline árboles)
rf = Pipeline(steps=[
    ("prep", preprocess),
    ("clf", RandomForestClassifier(
        n_estimators=300, max_depth=None, random_state=42,
class_weight="balanced"
    ))
])

models = {
    "LogisticRegression": log_reg,
    "RandomForest": rf
}
```

5. Función de evaluación y visualización

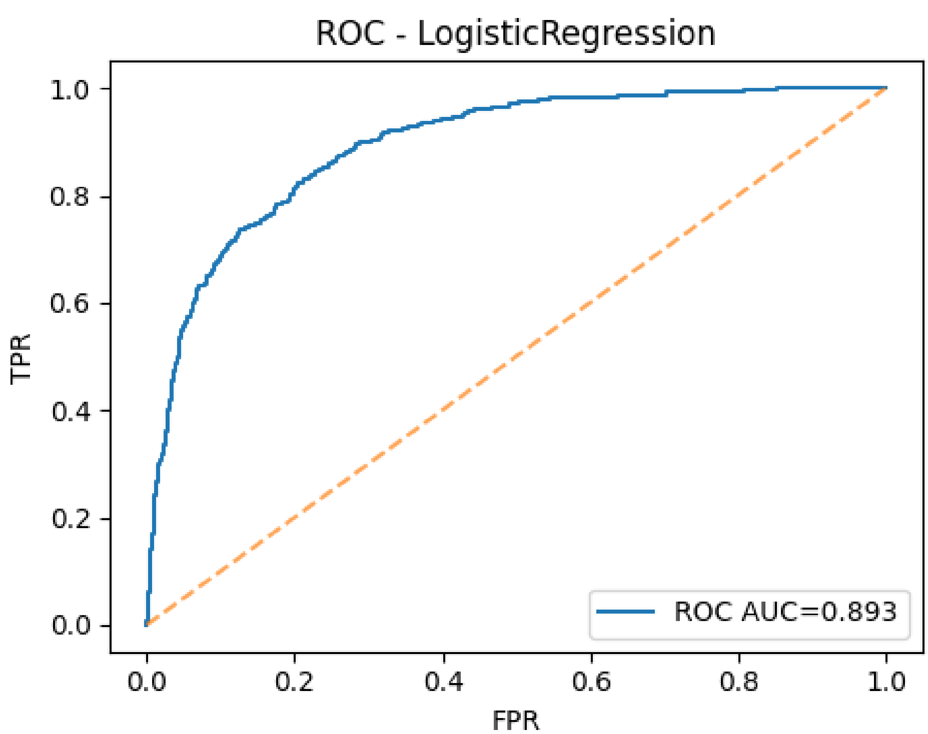
```
# ---------- 5) Entrenamiento + Métricas en Test ----------
def evaluate(model, X_train, y_train, X_test, y_test, name="model"):
    model.fit(X_train, y_train)
    proba = model.predict_proba(X_test)[:, 1]
    preds_default = (proba >= 0.5).astype(int)

    acc = accuracy_score(y_test, preds_default)
    f1 = f1_score(y_test, preds_default)
    roc = roc_auc_score(y_test, proba)
    ap = average_precision_score(y_test, proba)  # PR AUC

    print(f"\n=== {name} ===")
```

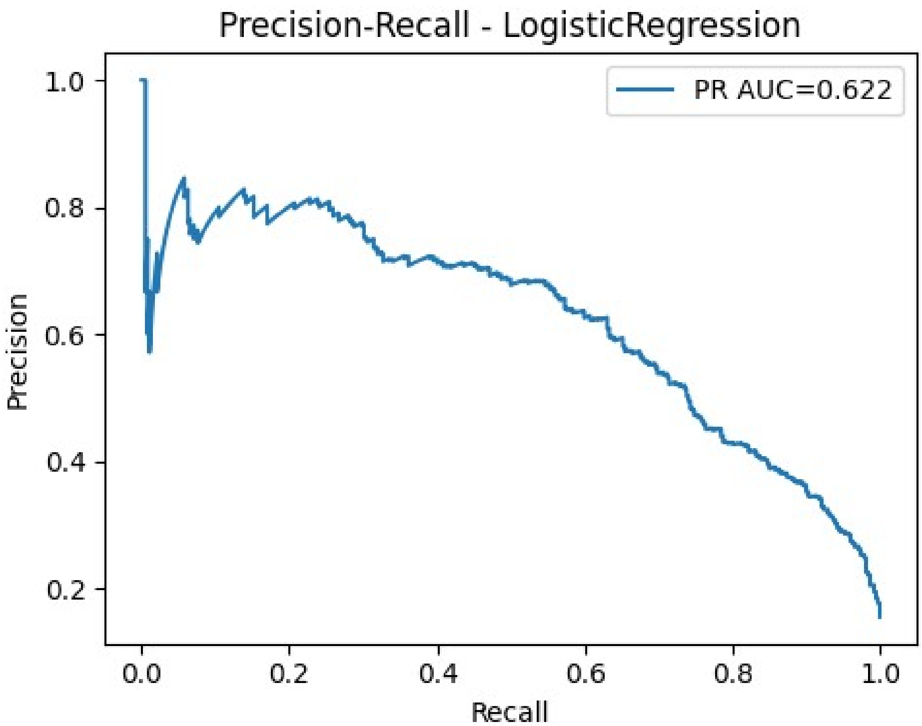
```python
    print(f"Accuracy : {acc:.4f}")
    print(f"F1       : {f1:.4f}")
    print(f"ROC AUC  : {roc:.4f}")
    print(f"PR AUC   : {ap:.4f}")
    print("\nClassification report (umbral 0.5):")
    print(classification_report(y_test, preds_default, digits=4))

    # Matriz de confusión
    cm = confusion_matrix(y_test, preds_default)
    print("Confusion matrix:\n", cm)

    # Curva ROC
    fpr, tpr, _ = roc_curve(y_test, proba)
    plt.figure(figsize=(5,4))
    plt.plot(fpr, tpr, label=f"ROC AUC={roc:.3f}")
    plt.plot([0,1],[0,1],'--', alpha=0.7)
    plt.xlabel("FPR")
    plt.ylabel("TPR")
    plt.title(f"ROC - {name}")
    plt.legend()
    plt.tight_layout()
    plt.show()

    # Curva Precision-Recall
    prec, rec, thr = precision_recall_curve(y_test, proba)
    plt.figure(figsize=(5,4))
    plt.plot(rec, prec, label=f"PR AUC={ap:.3f}")
    plt.xlabel("Recall")
    plt.ylabel("Precision")
    plt.title(f"Precision-Recall - {name}")
    plt.legend()
    plt.tight_layout()
    plt.show()

    # ---- Búsqueda de umbral: maximizar F1 ----
    best_f1, best_t = 0, 0.5
    for t in np.linspace(0.05, 0.95, 19):
        p = (proba >= t).astype(int)
        f1_t = f1_score(y_test, p)
        if f1_t > best_f1:
            best_f1, best_t = f1_t, t
    print(f"Mejor umbral por F1: t={best_t:.2f}, F1={best_f1:.4f}")

    return {"acc": acc, "f1": f1, "roc_auc": roc, "pr_auc": ap,
"best_thr": best_t, "best_f1": best_f1}
```

6. Entrenamiento y reporte por modelo

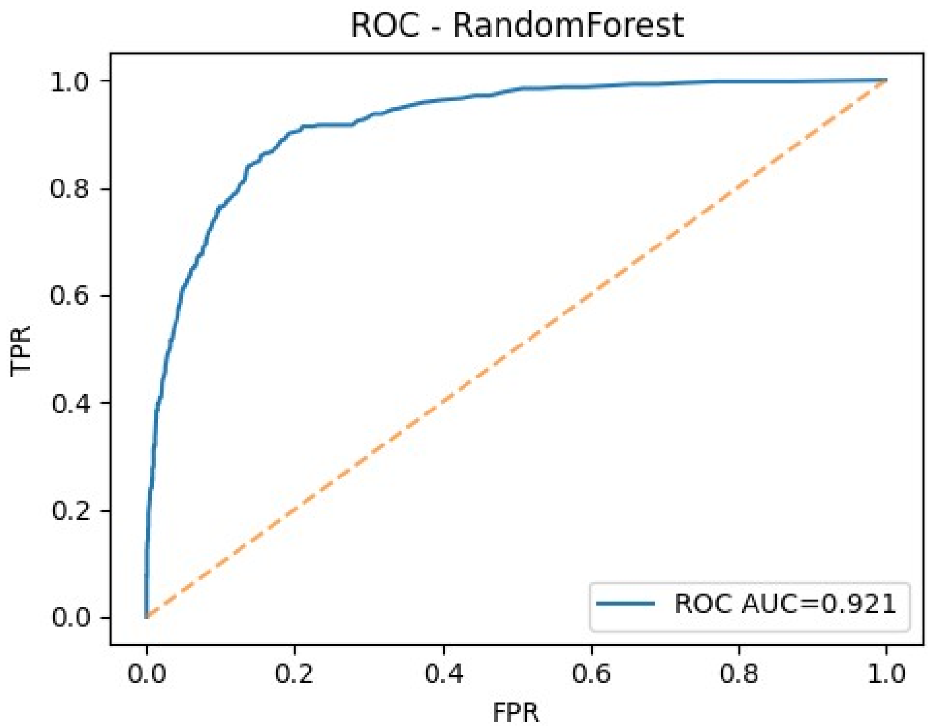
```
results = {}
for name, mdl in models.items():
    results[name] = evaluate(mdl, X_train, y_train, X_test, y_test,
name=name)
```

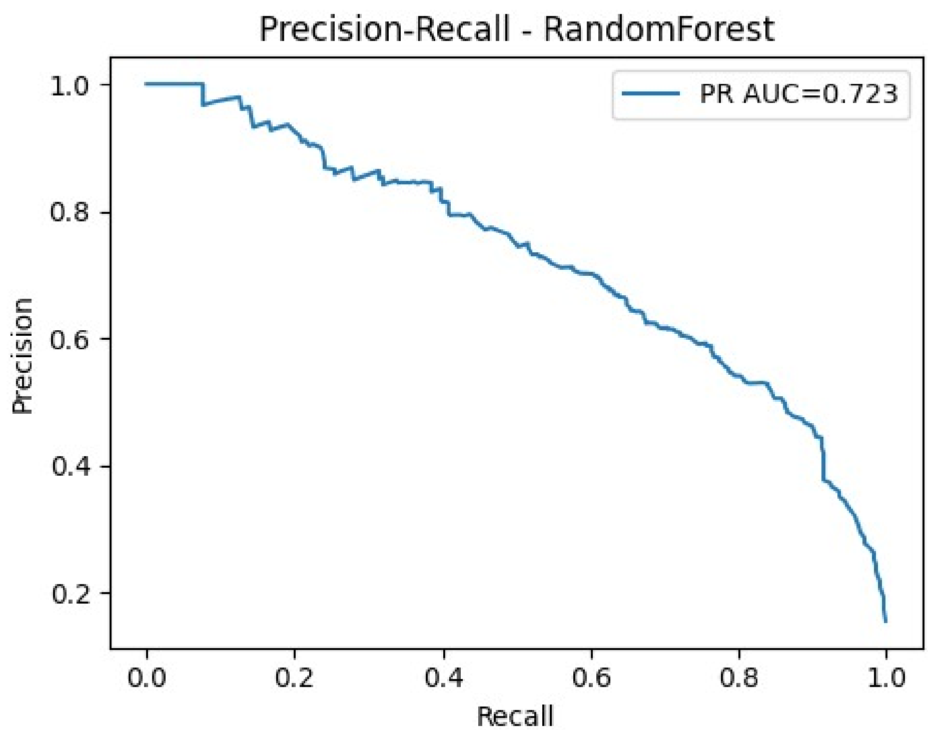
```
=== LogisticRegression ===
Accuracy : 0.8410
F1       : 0.5917
ROC AUC  : 0.8932
PR AUC   : 0.6224

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

           0     0.9481    0.8589    0.9013      2084
           1     0.4913    0.7435    0.5917       382

    accuracy                         0.8410      2466
   macro avg     0.7197    0.8012    0.7465      2466
weighted avg     0.8773    0.8410    0.8533      2466

Confusion matrix:
 [[1790  294]
 [  98  284]]
```






```
Mejor umbral por F1: t=0.60, F1=0.6183

=== RandomForest ===
Accuracy : 0.8974
F1       : 0.5965
ROC AUC  : 0.9208
PR AUC   : 0.7234

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

           0     0.9122    0.9722    0.9412      2084
           1     0.7633    0.4895    0.5965       382

    accuracy                         0.8974      2466
   macro avg     0.8377    0.7308    0.7689      2466
weighted avg     0.8891    0.8974    0.8878      2466

Confusion matrix:
 [[2026   58]
 [ 195  187]]
```






```
Mejor umbral por F1: t=0.30, F1=0.6586
```

7. Validación cruzada (ROC AUC CV5)

```
# ---------- 6) Validación cruzada (ROC AUC) ----------
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
for name, mdl in models.items():
    scores = cross_val_score(mdl, X, y, cv=cv, scoring="roc_auc")
    print(f"\n{name} - ROC AUC CV5: {scores.mean():.4f} ±
{scores.std():.4f}")
```

```
LogisticRegression - ROC AUC CV5: 0.9030 ± 0.0077

RandomForest - ROC AUC CV5: 0.9260 ± 0.0048
```

8. Interpretabilidad: coeficientes de la logística

```
# ---------- 7) (Opcional) Ver coeficientes de la logística ----------
# Para leer coeficientes de la LR, ajustamos y extraemos columnas
transformadas:
log_reg.fit(X_train, y_train)

# Obtener nombres de columnas después del preprocesamiento
ohe = log_reg.named_steps["prep"].named_transformers_["cat"]
num_names = numeric_features
cat_names = list(ohe.get_feature_names_out(categorical_features)) if
categorical_features else []
feat_names = num_names + cat_names

clf = log_reg.named_steps["clf"]
coefs = pd.Series(clf.coef_.ravel(),
index=feat_names).sort_values(key=np.abs, ascending=False)
print("\nTop coeficientes (|peso|) - LogisticRegression:")
display(coefs.head(20).to_frame("coef"))
```

```
Top coeficientes (|peso|) - LogisticRegression:
```


```
{"summary":"{\n  \"name\": \"display(coefs\",\n  \"rows\": 20,\n
\"fields\": [\n    {\n      \"column\": \"coef\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.9594114793611253,\n        \"min\": -1.2732857577802994,\n
\"max\": 2.3074413045461784,\n        \"num_unique_values\": 20,\n
\"samples\": [\n          2.3074413045461784,\n          -
0.424640468403154,\n          0.4919118142891404\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```