

```
!pip install -q gdown
```

## ✚ Importar librerías necesarias

Cargar todas las librerías necesarias para la manipulación de datos

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.metrics import classification_report, mean_absolute_error
from xgboost import XGBClassifier
from statsmodels.tsa.stattools import acf
from google.colab import files
import gdown
import time
from lightgbm import LGBMClassifier
```

## ✚ Datos extraídos de un Drive

```
urlclientes = "https://drive.google.com/file/d/1RpvKvh4YWUc6ETD-KUXBzDGZM3USgTTJ/view?usp=sharing"
# Extraer ID y convertirlo a un enlace descargable
file_clientes = urlclientes.split("/d/")[1].split("/")[0]
download_url1 = f"https://drive.google.com/uc?id={file_clientes}"
# Descargar el archivo
baseclientes = "base_clientes_final.csv"
gdown.download(download_url1, baseclientes, quiet=False)
urltransacciones = "https://drive.google.com/file/d/1GvPBcygM9lWfk8B4aywJY4B70c9XI5f3/view?usp=sharing"
# Extraer ID y convertirlo a un enlace descargable
file_transacciones = urltransacciones.split("/d/")[1].split("/")[0]
download_url2 = f"https://drive.google.com/uc?id={file_transacciones}"
# Descargar el archivo
basetransacciones = "base_transacciones_final.csv"
gdown.download(download_url2, basetransacciones, quiet=False)
```

```
📄 Downloading...
From: https://drive.google.com/uc?id=1RpvKvh4YWUc6ETD-KUXBzDGZM3USgTTJ
To: /content/base_clientes_final.csv
100%|██████████| 154k/154k [00:00<00:00, 35.5MB/s]
Downloading...
From: https://drive.google.com/uc?id=1GvPBcygM9lWfk8B4aywJY4B70c9XI5f3
To: /content/base_transacciones_final.csv
100%|██████████| 36.3M/36.3M [00:00<00:00, 107MB/s]
'base_transacciones_final.csv'
```

## ✚ Cargar y preparar datos

Importar la base de datos, corregir inconsistencias, unirlas y generar variables temporales

```
# Carga de bases de clientes y transacciones
clientes = pd.read_csv(baseclientes)
transacciones = pd.read_csv(basetransacciones)

# Corregir datos faltantes o inconsistentes (ejemplo: asignar género)
clientes.loc[clientes['id'] == '9980f12e32711330d5f58460e169e6207afda041', 'genero'] = 'M'

# Merge para combinar información de clientes y sus transacciones
df = pd.merge(clientes, transacciones, on='id')

# Convertir fechas a formato datetime
df['fecha'] = pd.to_datetime(df['fecha'])
df['fecha_nacimiento'] = pd.to_datetime(df['fecha_nacimiento'])

# Crear columna 'anio_mes' para agrupar por mes
df['anio_mes'] = df['fecha'].dt.to_period('M')

# Calcular edad actual aproximada del cliente
df['edad'] = (pd.Timestamp('today') - df['fecha_nacimiento']).dt.days // 365

# Limpiar datos eliminando filas con valores nulos en columnas clave
df.dropna(subset=['monto', 'fecha', 'id'], inplace=True)
```

## ✚ Detección de patrones de gasto recurrente

Agrupar los datos por cliente, comercio y mes para sumar montos y detectar si hay patrones recurrentes

```
# Agrupar transacciones por cliente, comercio y mes; sumar montos mensuales
recurrentes = df.groupby(["id", "comercio", "anio_mes"]).agg({"monto": "sum"}).reset_index()

# Crear tabla pivot con montos mensuales por cliente y comercio
pivot = recurrentes.pivot_table(index="id", "comercio", columns="anio_mes", values="monto", fill_value=0)
```

```
#Para detectar patrones: Si la serie es demasiado corta -> no recurrente
#Si alguna autocorrelación en los primeros 12 lags es > 0.5 -> es recurrente
def detectar_periodicidad(serie):
    if len(serie) < 3 or np.var(serie) == 0:
        return False
    acf_vals = acf(serie, nlags=min(12, len(serie)-1), fft=True)
    return any(acf_vals[1:] > 0.5)

# Aplicar detección de recurrencia a cada cliente-comercio
pivot["recurrente"] = pivot.apply(lambda row: detectar_periodicidad(row.values), axis=1)
df_recurrentes = pivot[pivot['recurrente']].reset_index()
recurrentes
```



	id	comercio	anio_mes	monto	
0	003d9abe467a91847d566cf455bd2d7d6c8f7e75	AMAZON	2022-01	83.28	 
1	003d9abe467a91847d566cf455bd2d7d6c8f7e75	AMAZON	2022-02	172.78	
2	003d9abe467a91847d566cf455bd2d7d6c8f7e75	AMAZON	2022-04	168.78	
3	003d9abe467a91847d566cf455bd2d7d6c8f7e75	AMAZON	2022-07	19.90	
4	003d9abe467a91847d566cf455bd2d7d6c8f7e75	AMAZON	2022-08	48.38	
...	...	...	...	...	
120579	ff67da037fae796809be0e36fb9cdd0e191c38a4	UBER EATS	2022-06	49.29	
120580	ff67da037fae796809be0e36fb9cdd0e191c38a4	UBER EATS	2022-07	65.02	
120581	ff67da037fae796809be0e36fb9cdd0e191c38a4	UBER EATS	2022-08	76.47	
120582	ff67da037fae796809be0e36fb9cdd0e191c38a4	UBER EATS	2022-10	48.81	
120583	ff67da037fae796809be0e36fb9cdd0e191c38a4	UBER EATS	2023-01	83.86	

120584 rows x 4 columns

Exportación del DataFrame con etiquetas

Se exporta el dataframe recurrentes

```
recurrentes.to_csv('recurrentes.csv', index=False)
#from google.colab import files
#files.download('recurrentes.csv')
```

Preparación de datos para modelado predictivo

Crea nuevas variables de contexto temporal y de comportamiento de gasto para entrenar modelos predictivos.

Clase 1 -> Gastos recurrentes Clase 0 -> Sin gastos

XGBoost que es demasiado robusto y tiene costo computacional "alto"

```
df = pd.read_csv("recurrentes.csv")

# Convertir 'anio_mes' a formato datetime
df["anio_mes"] = pd.to_datetime(df["anio_mes"], format="%Y-%m")

# Ordenar por cliente, comercio y fecha
df = df.sort_values(by=["id", "comercio", "anio_mes"]).reset_index(drop=True)

# Crear clave única combinando cliente y comercio
df["clave"] = df["id"] + "_" + df["comercio"]

# Calcular el mes siguiente para cada registro
df["anio_mes_next"] = df["anio_mes"] + pd.offsets.MonthBegin(1)

# Definir variables objetivo:
# 'target_ocurrencia' indica si hubo transacción en el mes siguiente (1 = sí, 0 = no)
# 'target_monto' es el monto gastado en ese mes siguiente
futuro = df[["clave", "anio_mes", "monto"]].copy()
futuro["anio_mes"] = futuro["anio_mes"] + pd.offsets.MonthBegin(1)
futuro = futuro.rename(columns={"monto": "target_monto"})
df = df.merge(futuro, on=["clave", "anio_mes"], how="left")
df["target_ocurrencia"] = (~df["target_monto"].isna()).astype(int)

# Crear variables predictivas basadas en historial de montos y frecuencia:
df["monto_mes_anterior"] = df.groupby("clave")["monto"].shift(1)
df["monto_ultimos_3"] = df.groupby("clave")["monto"].transform(lambda x: x.shift(1).rolling(3).mean())
df["frecuencia_ultimos_6"] = df.groupby("clave")["monto"].transform(lambda x: x.shift(1).rolling(6).apply(lambda y: np.sum(~np.isnan(y))))
df["inactividad_ultimos_6"] = df.groupby("clave")["monto"].transform(lambda x: x.shift(1).rolling(6).apply(lambda y: 6 - np.count_nonzero(~np.isnan(y))))
df["dias_desde_ultimo_pago"] = df.groupby("clave")["anio_mes"].transform(lambda x: (x.max() - x).dt.days)

# Rellenar valores faltantes y extraer mes para posibles patrones estacionales
df = df.fillna(0)
df["mes"] = df["anio_mes"].dt.month

# Eliminar filas sin información de ocurrencia futura para modelado
df_model = df.dropna(subset=["target_ocurrencia"])
df_model
```

		id	comercio	anio_mes	monto		clave	anio_mes_next	target_monto	target_ocur
0	003d9abe467a91847d566cf455bd2d7d6c8f7e75		AMAZON	2022-01-01	83.28	003d9abe467a91847d566cf455bd2d7d6c8f7e75_AMAZON		2022-02-01	172.78	
1	003d9abe467a91847d566cf455bd2d7d6c8f7e75		AMAZON	2022-02-01	172.78	003d9abe467a91847d566cf455bd2d7d6c8f7e75_AMAZON		2022-03-01	0.00	
2	003d9abe467a91847d566cf455bd2d7d6c8f7e75		AMAZON	2022-04-01	168.78	003d9abe467a91847d566cf455bd2d7d6c8f7e75_AMAZON		2022-05-01	0.00	
3	003d9abe467a91847d566cf455bd2d7d6c8f7e75		AMAZON	2022-07-01	19.90	003d9abe467a91847d566cf455bd2d7d6c8f7e75_AMAZON		2022-08-01	48.38	
4	003d9abe467a91847d566cf455bd2d7d6c8f7e75		AMAZON	2022-08-01	48.38	003d9abe467a91847d566cf455bd2d7d6c8f7e75_AMAZON		2022-09-01	0.00	
...	...	...	...	...	...	...	...	...	...	...
120579	ff67da037fae796809be0e36fb9cdd0e191c38a4		UBER EATS	2022-06-01	49.29	ff67da037fae796809be0e36fb9cdd0e191c38a4_UBER ...		2022-07-01	65.02	
120580	ff67da037fae796809be0e36fb9cdd0e191c38a4		UBER EATS	2022-07-01	65.02	ff67da037fae796809be0e36fb9cdd0e191c38a4_UBER ...		2022-08-01	76.47	
120581	ff67da037fae796809be0e36fb9cdd0e191c38a4		UBER EATS	2022-08-01	76.47	ff67da037fae796809be0e36fb9cdd0e191c38a4_UBER ...		2022-09-01	0.00	
120582	ff67da037fae796809be0e36fb9cdd0e191c38a4		UBER EATS	2022-10-01	48.81	ff67da037fae796809be0e36fb9cdd0e191c38a4_UBER ...		2022-11-01	0.00	
120583	ff67da037fae796809be0e36fb9cdd0e191c38a4		UBER EATS	2023-01-01	83.86	ff67da037fae796809be0e36fb9cdd0e191c38a4_UBER ...		2023-02-01	0.00	
120584 rows x 14 columns										

120584 rows x 14 columns

Entrenamiento de modelos XGBoost y Random Forest

Se entrena un modelo XGBoost para predecir si habrá un gasto el siguiente mes y Random Forest para predecir el monto

```
features = [
    "mes", "monto_mes_anterior", "monto_ultimos_3",
    "frecuencia_ultimos_6", "inactividad_ultimos_6", "dias_desde_ultimo_pago"
]
#Clasificación para saber si se realiza el pago o no y regresión para saber el monto
X = df_model[features]
y_class = df_model["target_ocurrencia"]
y_reg = df_model["target_monto"].fillna(0)

#XGBoost
X_train_c, X_test_c, y_train_c, y_test_c = train_test_split(X, y_class, test_size=0.2, random_state=42)

xgb_base = XGBClassifier(
    use_label_encoder=False,
    eval_metric='logloss',
    random_state=42
)
```

```

param_grid = {
    'max_depth': [3, 4],          # Menos niveles, suficiente para evitar árboles muy complejos
    'n_estimators': [50, 100],    # Número moderado de árboles
    'learning_rate': [0.1],       # Un solo valor razonable (puedes cambiarlo a 0.1 si quieres algo más rápido)
    'reg_alpha': [0, 0.1],        # Regularización L1 para prevenir overfitting
    'scale_pos_weight': [1, 1.5]  # Balance de clases (por si la clase 0 está poco representada)
}

grid_search = GridSearchCV(
    estimator=xgb_base,
    param_grid=param_grid,
    scoring='f1',
    cv=3,
    verbose=1,
    n_jobs=-1
)


grid_search.fit(X_train_c, y_train_c)

print("Mejores hiperparámetros:")
print(grid_search.best_params_)

# Predecir en test set
y_pred_c = grid_search.best_estimator_.predict(X_test_c)

print("\nReporte de clasificación (mejor modelo):")
print(classification_report(y_test_c, y_pred_c, digits=3))

```

 Fitting 3 folds for each of 16 candidates, totalling 48 fits  
 /usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [16:25:24] WARNING: /workspace/src/learner.cc:740:  
 Parameters: { "use\_label\_encoder" } are not used.

```

warnings.warn(msg, UserWarning)
Mejores hiperparámetros:
{'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 100, 'reg_alpha': 0, 'scale_pos_weight': 1}

Reporte de clasificación (mejor modelo):

```

	precision	recall	f1-score	support
0	0.917	0.607	0.731	8168
1	0.829	0.972	0.894	15949
accuracy			0.848	24117
macro avg	0.873	0.789	0.812	24117
weighted avg	0.858	0.848	0.839	24117

## ✓ Evaluación de regresión por comercio

Aquí se entrena un modelo de regresión por comercio para predecir montos. Se imprime el MAE por comercio.

**Este modelo no sirve porque el MAE es grande**

Se comparan los reportes de clasificación de train y test del XGBoost para verificar que el recall en la clase 0 no sea por overfitting

```

print("\nMAE por comercio:")
for comercio in df_model["comercio"].unique():
    subset = df_model[(df_model["comercio"] == comercio) & (df_model["target_ocurrencia"] == 1)]
    if len(subset) > 100:
        X_reg = subset[features]
        y_reg = subset["target_monto"]
        X_train, X_test, y_train, y_test = train_test_split(X_reg, y_reg, test_size=0.2, random_state=42)
        model = RandomForestRegressor(n_estimators=100, random_state=42)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        print(f"{comercio}: MAE = {mean_absolute_error(y_test, y_pred):.2f}")

```



```

ELEVEN: MAE = 32.03
COPPEL: MAE = 188.38
MEGACABLE: MAE = 24.14
SEARS: MAE = 178.84
SERV AGUA DREN: MAE = 31.88
VIX: MAE = 2.82
CABLEYCOMUN: MAE = 21.47
PARCO: MAE = 9.80
UBRPAGOSMEX: MAE = 129.02
ALSUPER: MAE = 151.57
GOOGLE YOUTUBEPREMIUM: MAE = 11.44
TOTAL PLAY: MAE = 25.05
AT&T: MAE = 36.93
TELEFONICA: MAE = 17.08
OPENAI: MAE = 11.52
TULOTERO: MAE = 74.78
NAYAX: MAE = 22.60
URBANI: MAE = 18.59
DISNEY PLUS: MAE = 26.45
TOTAL PASS: MAE = 16.40
RENTAMOVISTAR: MAE = 58.56
ROTOPLAS: MAE = 28.92
AUDIBLE: MAE = 6.45
APLAZO: MAE = 205.94
CALIENTE: MAE = 621.47
COSTCO GAS: MAE = 113.85
SMART: MAE = 113.80
BET365: MAE = 2047.46
CANVA: MAE = 8.83
SMARTFIT: MAE = 43.12
ALLIANZ MEXICO: MAE = 97.93

```

## ✓ Desempeño entre conjunto de entrenamiento y prueba

Para tener claro que no existe sobreajuste se comparan los reportes de clasificación en el conjunto de entrenamiento de prueba

```

y_train_pred_c = grid_search.best_estimator_.predict(X_train_c)

print("\nReporte de clasificación (Training Set):")
print(classification_report(y_train_c, y_train_pred_c, digits=3))

print("\nReporte de clasificación (Test Set):")
print(classification_report(y_test_c, y_pred_c, digits=3))

```



```

Reporte de clasificación (Training Set):
              precision    recall  f1-score   support

     0       0.923       0.624       0.744       32598
     1       0.835       0.973       0.899       63869

 accuracy                   0.855       96467
 macro avg       0.879       0.799       0.822       96467
 weighted avg    0.865       0.855       0.847       96467

```

```

Reporte de clasificación (Test Set):
              precision    recall  f1-score   support

     0       0.917       0.607       0.731       8168
     1       0.829       0.972       0.894      15949

 accuracy                   0.848      24117
 macro avg       0.873       0.789       0.812      24117
 weighted avg    0.858       0.848       0.839      24117

```

## ✓ Costo computacional del modelo XGBoost

Tiempo de entrenamiento

```

start_time = time.time()
grid_search.fit(X_train_c, y_train_c)
end_time = time.time()
print(f"Tiempo de entrenamiento: {end_time - start_time:.2f} segundos")

```



```

Fitting 3 folds for each of 16 candidates, totalling 48 fits
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [16:26:13] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

 warnings.warn(msg, UserWarning)
Tiempo de entrenamiento: 15.81 segundos

```

Tiempo de inferencia

```

start_time = time.time()
y_pred = grid_search.best_estimator_.predict(X_test_c)
end_time = time.time()
print(f"Tiempo de inferencia para {len(X_test_c)} muestras: {end_time - start_time:.4f} segundos")

```




```

Tiempo de inferencia para 24117 muestras: 0.0311 segundos

```

Uso de memoria

```
!pip install -q memory-profiler
from memory_profiler import memory_usage
def train_model():
    grid_search.fit(X_train_c, y_train_c)
mem_usage = memory_usage(train_model)
print(f"Uso máximo de memoria: {max(mem_usage):.2f} MB")
```

 Fitting 3 folds for each of 16 candidates, totalling 48 fits  
 /usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [16:26:33] WARNING: /workspace/src/learner.cc:740:  
 Parameters: { "use\_label\_encoder" } are not used.

```
warnings.warn(msg, UserWarning)
Uso máximo de memoria: 559.09 MB
```

## ✓ Modelos alternativos para la entrega

Debido al alto costo de entrenar y desplegar XGBoost, se evalúan otros modelos con menor carga computacional: Logistic Regression, LightGBM y HistGradientBoostingClassifier

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.metrics import classification_report

df = pd.read_csv("recurrentes.csv")
df["anio_mes"] = pd.to_datetime(df["anio_mes"], format="%Y-%m")
df = df.sort_values(by=["id", "comercio", "anio_mes"]).reset_index(drop=True)
df["clave"] = df["id"] + "_" + df["comercio"]
df["anio_mes_next"] = df["anio_mes"] + pd.offsets.MonthBegin(1)

futuro = df[["clave", "anio_mes", "monto"]].copy()
futuro["anio_mes"] = futuro["anio_mes"] - pd.offsets.MonthBegin(1)
futuro = futuro.rename(columns={"monto": "target_monto"})
df = df.merge(futuro, on=["clave", "anio_mes"], how="left")
df["target_ocurrencia"] = (~df["target_monto"].isna()).astype(int)

df["monto_mes_anterior"] = df.groupby("clave")["monto"].shift(1)
df["monto_ultimos_3"] = df.groupby("clave")["monto"].transform(lambda x: x.shift(1).rolling(3).mean())
df["frecuencia_ultimos_6"] = df.groupby("clave")["monto"].transform(lambda x: x.shift(1).rolling(6).apply(lambda y: np.sum(~np.isnan(y))))
df["inactividad_ultimos_6"] = df.groupby("clave")["monto"].transform(lambda x: x.shift(1).rolling(6).apply(lambda y: 6 - np.count_nonzero(~np.isnan(y))))
df["dias_desde_ultimo_pago"] = df.groupby("clave")["anio_mes"].transform(lambda x: (x.max() - x).dt.days)
df = df.fillna(0)
df["mes"] = df["anio_mes"].dt.month
df_model = df.dropna(subset=["target_ocurrencia"])

features = [
    "mes", "monto_mes_anterior", "monto_ultimos_3",
    "frecuencia_ultimos_6", "inactividad_ultimos_6", "dias_desde_ultimo_pago"
]
X = df_model[features]
y_class = df_model["target_ocurrencia"]

X_train_c, X_test_c, y_train_c, y_test_c = train_test_split(X, y_class, test_size=0.2, random_state=42)

def medir_tiempo(modelo, nombre):
    start_train = time.time()
    modelo.fit(X_train_c, y_train_c)
    end_train = time.time()

    start_pred = time.time()
    y_pred = modelo.predict(X_test_c)
    end_pred = time.time()

    print(f"\n Modelo: {nombre}")
    print(f" Tiempo de entrenamiento: {end_train - start_train:.2f} segundos")
    print(f" Tiempo de inferencia: {end_pred - start_pred:.4f} segundos")
    print(classification_report(y_test_c, y_pred, digits=3))

#Logistic Regression
log_model = LogisticRegression(max_iter=1000, random_state=42)
medir_tiempo(log_model, "Logistic Regression")

#LightGBM
lgb_model = LGBMClassifier(
    boosting_type='gbdt',
    num_leaves=31,
    learning_rate=0.1,
    n_estimators=100,
    class_weight='balanced',
    random_state=42
)
medir_tiempo(lgb_model, "LightGBM")

#HistGradientBoostingClassifier
hist_model = HistGradientBoostingClassifier(
    max_iter=100,
    learning_rate=0.1,
    max_depth=6,
    random_state=42
)
```

```
)
medir_tiempo(hist_model, "HistGradientBoosting")
```



```
Modelo: Logistic Regression
Tiempo de entrenamiento: 1.71 segundos
Tiempo de inferencia: 0.0139 segundos
```

	precision	recall	f1-score	support
0	0.793	0.509	0.620	8168
1	0.788	0.932	0.854	15949
accuracy			0.789	24117
macro avg	0.790	0.721	0.737	24117
weighted avg	0.789	0.789	0.775	24117

```
[LightGBM] [Info] Number of positive: 63869, number of negative: 32598
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.014448 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 558
[LightGBM] [Info] Number of data points in the train set: 96467, number of used features: 5
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Start training from score 0.000000
```

```
Modelo: LightGBM
Tiempo de entrenamiento: 1.38 segundos
Tiempo de inferencia: 0.2160 segundos
```

	precision	recall	f1-score	support
0	0.747	0.772	0.759	8168
1	0.881	0.866	0.873	15949
accuracy			0.834	24117
macro avg	0.814	0.819	0.816	24117
weighted avg	0.836	0.834	0.835	24117

```
Modelo: HistGradientBoosting
Tiempo de entrenamiento: 1.64 segundos
Tiempo de inferencia: 0.2280 segundos
```

	precision	recall	f1-score	support
0	0.911	0.624	0.740	8168
1	0.834	0.969	0.896	15949
accuracy			0.852	24117
macro avg	0.872	0.796	0.818	24117
weighted avg	0.860	0.852	0.843	24117

## Conclusion

Si bien XGBoost obtuvo el mejor desempeño general en términos de precisión y recall, su alto costo computacional en tiempo y memoria lo hace menos práctico en entornos de producción o cuando se requiere escalar.

El modelo que ofrece el mejor balance entre desempeño y eficiencia computacional es LightGBM, ya que mantiene un buen nivel de precisión, especialmente en la clase minoritaria, con un tiempo de entrenamiento mucho menor y un uso eficiente de recursos.

## ✓ REGRESION PARA PREDECIR EL MONTO

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from lightgbm import LGBMRegressor
from scipy.stats import zscore

# Cargar y preparar tus datos
df = pd.read_csv("recurrentes.csv")
df = df_model
df = preparar_features(df)
df_reg = df[df["target_ocurrencia"] == 1].copy()

# Aplicar z-score al target
df_reg["z_score_monto"] = zscore(df_reg["target_monto"])
df_z_filtered = df_reg[np.abs(df_reg["z_score_monto"]) < 3].copy()
df_z_filtered.drop(columns=["z_score_monto"], inplace=True)

# Definir features y target
features = ['mes', 'monto_mes_anterior', 'monto_ultimos_3', 'frecuencia_ultimos_6', 'inactividad_ultimos_6', 'dias_desde_ultimo_pago']
X = df_z_filtered[features]
y = df_z_filtered["target_monto"]

# Separar y entrenar
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)
model = LGBMRegressor(n_estimators=100, max_depth= 15, learning_rate=0.05, num_leaves=31, random_state=123, reg_alpha= 0.5, reg_lambda= 0)

model.fit(X_train, y_train)

# Predicción y métricas
y_pred = model.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

```
yz = lgb_predict(y_test, y_pred)
```

```
print(f"Evaluación del modelo con Z-score:")
print(f"MAE = {mae:.2f}")
print(f"MSE = {mse:.2f}")
print(f"R² = {r2:.2f}")
```

```

[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.006850 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 558
[LightGBM] [Info] Number of data points in the train set: 63306, number of used features: 5
[LightGBM] [Info] Start training from score 108.158325
Evaluación del modelo con Z-score:
MAE = 67.78
MSE = 16523.69
R² = 0.42

```

```

y_pred = lgb_model.predict(X_test_c)
#Los que si tienen recurrencia
y_prob = lgb_model.predict_proba(X_test_c)[:, 1]

```

```

# Df de predicciones
df_predicciones = X_test_c.copy()
df_predicciones["prediccion"] = y_pred
df_predicciones["probabilidad_ocurrencia"] = y_prob
df_predicciones["real"] = y_test_c.values

```

```

df_predicciones = df_predicciones.reset_index(drop=True)
df_predicciones.to_csv("predicciones_lightgbm.csv", index=False)

```

```
print(df_predicciones.head())
```

```

mes    monto_mes_anterior  monto_ultimos_3  frecuencia_ultimos_6 \
0         4              66.76          66.760000          0.0

```