Javier Mombiela

Carnet: 20067

12 de abril 2024

Proyecto 2: Entrenamiento Incremental en Modelos de Deep Learning y Machine Learning

Importando librerias

```
import numpy as np
import pandas as pd
import seaborn as sns
import xgboost as xgb
import lightgbm as lgb
import matplotlib.pyplot as plt
from imblearn.over_sampling import SMOTE
from sklearn.impute import SimpleImputer
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, recall_score, f1_score,
roc_curve, auc, precision_recall_curve, accuracy_score
```

Análisis Exploratorio

El dataset contiene información detallada sobre transacciones de tarjetas de crédito, recopilada para la identificación de fraudes. Incluye datos sobre la fecha, el monto y la ubicación de las transacciones, así como información demográfica de los titulares de las tarjetas y características de las transacciones que permiten distinguir entre actividades legítimas y fraudulentas.

```
df = pd.read_csv('fraud_feature_engineering_example.csv')

df.head()
```

```
  trans_date_trans_time            cc_num
merchant  \
0   2019-01-01 00:00:18  2703186189652095          fraud_Rippin, Kub
and Mann
1   2019-01-01 00:00:44      630423337322     fraud_Heller, Gutmann
and Zieme
2   2019-01-01 00:00:51    38859492057661                fraud_Lind-
Buckridge
3   2019-01-01 00:01:16  3534093764340240  fraud_Kutch, Hermiston and
Farrell
```

```
4   2019-01-01 00:03:06   375534208663984
fraud_Keeling-Crist

        category     amt      first     last gender  \
0       misc_net    4.97   Jennifer    Banks      F
1    grocery_pos  107.23  Stephanie     Gill      F
2  entertainment  220.11     Edward  Sanchez      M
3  gas_transport   45.00     Jeremy    White      M
4       misc_pos   41.96      Tyler   Garcia      M

                         street            city  ...  \
0                561 Perry Cove  Moravian Falls  ...
1  43039 Riley Greens Suite 393          Orient  ...
2     594 White Dale Suite 530     Malad City  ...
3   9443 Cynthia Court Apt. 038        Boulder  ...
4              408 Bradley Rest        Doe Hill  ...

                          trans_num   unix_time  merch_lat  merch_long
\
0  0b242abb623afc578575680df30655b9  1325376018  36.011293  -82.048315

1  1f76529f8574734946361c461b024d99  1325376044  49.159047 -118.186462

2  a1a22d70485983eac12b5b88dad1cf95  1325376051  43.150704 -112.154481

3  6b849c168bdad6f867558c3793159a81  1325376076  47.034331 -112.561071

4  a41d7549acf90789359a9aa5346dcb46  1325376186  38.674999  -78.632459


   is_fraud amt_month amt_year amt_month_shopping_net_spend  \
0         0      4.97     4.97                          0.0
1         0    107.23   107.23                          0.0
2         0    220.11   220.11                          0.0
3         0     45.00    45.00                          0.0
4         0     41.96    41.96                          0.0

   count_month_shopping_net  first_time_at_merchant
0                       0.0                    True
1                       0.0                    True
2                       0.0                    True
3                       0.0                    True
4                       0.0                    True

[5 rows x 27 columns]
```

```python
# Información general sobre el dataframe
print("\nInformación del dataframe:")
print(df.info())
```

```
Información del dataframe:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1852394 entries, 0 to 1852393
Data columns (total 27 columns):
 #   Column                         Dtype
---  ------                         -----
 0   trans_date_trans_time          object
 1   cc_num                         int64
 2   merchant                       object
 3   category                       object
 4   amt                            float64
 5   first                          object
 6   last                           object
 7   gender                         object
 8   street                         object
 9   city                           object
 10  state                          object
 11  zip                            int64
 12  lat                            float64
 13  long                           float64
 14  city_pop                       int64
 15  job                            object
 16  dob                            object
 17  trans_num                      object
 18  unix_time                      int64
 19  merch_lat                      float64
 20  merch_long                     float64
 21  is_fraud                       int64
 22  amt_month                      float64
 23  amt_year                       float64
 24  amt_month_shopping_net_spend   float64
 25  count_month_shopping_net       float64
 26  first_time_at_merchant         bool
dtypes: bool(1), float64(9), int64(5), object(12)
memory usage: 369.2+ MB
None
```

Al revisar la información del dataset utilizando df.info(), se observa que varias columnas están codificadas como tipo object. Estas columnas incluyen nombres, direcciones y otros datos no numéricos que deberán ser transformados a un formato numérico o categórico para su procesamiento en modelos de machine learning. Se deberá realizar una adecuada codificación de estas características antes de entrenar cualquier modelo predictivo.

```python
# Verificar valores faltantes en el conjunto de datos
missing_values = df.isnull().sum()
print("Valores faltantes por columna:")
print(missing_values)
```

```
Valores faltantes por columna:
trans_date_trans_time           0
cc_num                          0
merchant                        0
category                        0
amt                             0
first                           0
last                            0
gender                          0
street                          0
city                            0
state                           0
zip                             0
lat                             0
long                            0
city_pop                        0
job                             0
dob                             0
trans_num                       0
unix_time                       0
merch_lat                       0
merch_long                      0
is_fraud                        0
amt_month                       0
amt_year                        0
amt_month_shopping_net_spend    0
count_month_shopping_net        0
first_time_at_merchant          0
dtype: int64
```

```
# Estadísticas descriptivas
print("\nEstadísticas descriptivas del dataset:")
df.describe()
```

```
Estadísticas descriptivas del dataset:

             cc_num           amt           zip           lat
long  \
count  1.852394e+06  1.852394e+06  1.852394e+06  1.852394e+06
1.852394e+06
mean   4.173860e+17  7.006357e+01  4.881326e+04  3.853931e+01 -
9.022783e+01
std    1.309115e+18  1.592540e+02  2.688185e+04  5.071470e+00
1.374789e+01
min    6.041621e+10  1.000000e+00  1.257000e+03  2.002710e+01 -
1.656723e+02
25%    1.800429e+14  9.640000e+00  2.623700e+04  3.466890e+01 -
9.679800e+01
50%    3.521417e+15  4.745000e+01  4.817400e+04  3.935430e+01 -
```

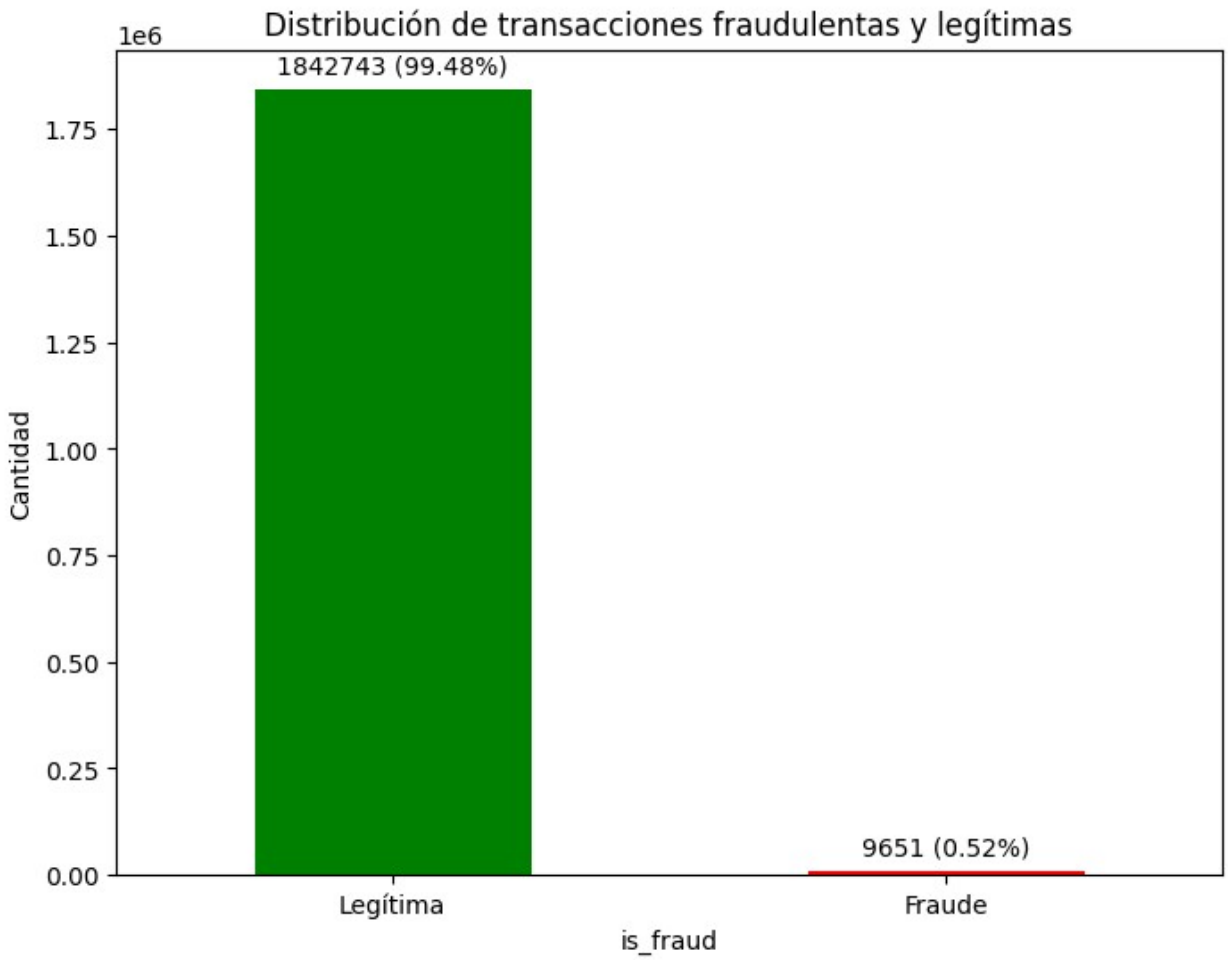
```
8.747690e+01
75%    4.642255e+15  8.310000e+01  7.204200e+04  4.194040e+01 -
8.015800e+01
max    4.992346e+18  2.894890e+04  9.992100e+04  6.669330e+01 -
6.795030e+01

           city_pop     unix_time     merch_lat    merch_long
is_fraud  \
count  1.852394e+06  1.852394e+06  1.852394e+06  1.852394e+06
1.852394e+06
mean   8.864367e+04  1.358674e+09  3.853898e+01 -9.022794e+01
5.210015e-03
std    3.014876e+05  1.819508e+07  5.105604e+00  1.375969e+01
7.199217e-02
min    2.300000e+01  1.325376e+09  1.902742e+01 -1.666716e+02
0.000000e+00
25%    7.410000e+02  1.343017e+09  3.474012e+01 -9.689944e+01
0.000000e+00
50%    2.443000e+03  1.357089e+09  3.936890e+01 -8.744069e+01
0.000000e+00
75%    2.032800e+04  1.374581e+09  4.195626e+01 -8.024511e+01
0.000000e+00
max    2.906700e+06  1.388534e+09  6.751027e+01 -6.695090e+01
1.000000e+00

          amt_month      amt_year  amt_month_shopping_net_spend  \
count  1.852394e+06  1.852394e+06                  1.852394e+06
mean   4.153689e+03  4.530560e+04                  3.762028e+02
std    3.909005e+03  3.586752e+04                  7.253531e+02
min    1.000000e+00  1.020000e+00                  0.000000e+00
25%    1.344790e+03  1.734142e+04                  9.020000e+00
50%    3.071990e+03  3.743910e+04                  7.589000e+01
75%    5.738470e+03  6.472088e+04                  4.259800e+02
max    4.326189e+04  2.190868e+05                  1.204718e+04

       count_month_shopping_net
count              1.852394e+06
mean               4.567241e+00
std                4.575502e+00
min                0.000000e+00
25%                1.000000e+00
50%                3.000000e+00
75%                7.000000e+00
max                4.800000e+01
```

```python
# Distribución de clases
class_distribution = df['is_fraud'].value_counts()
print("\nDistribución de clases:")
print(class_distribution)
```

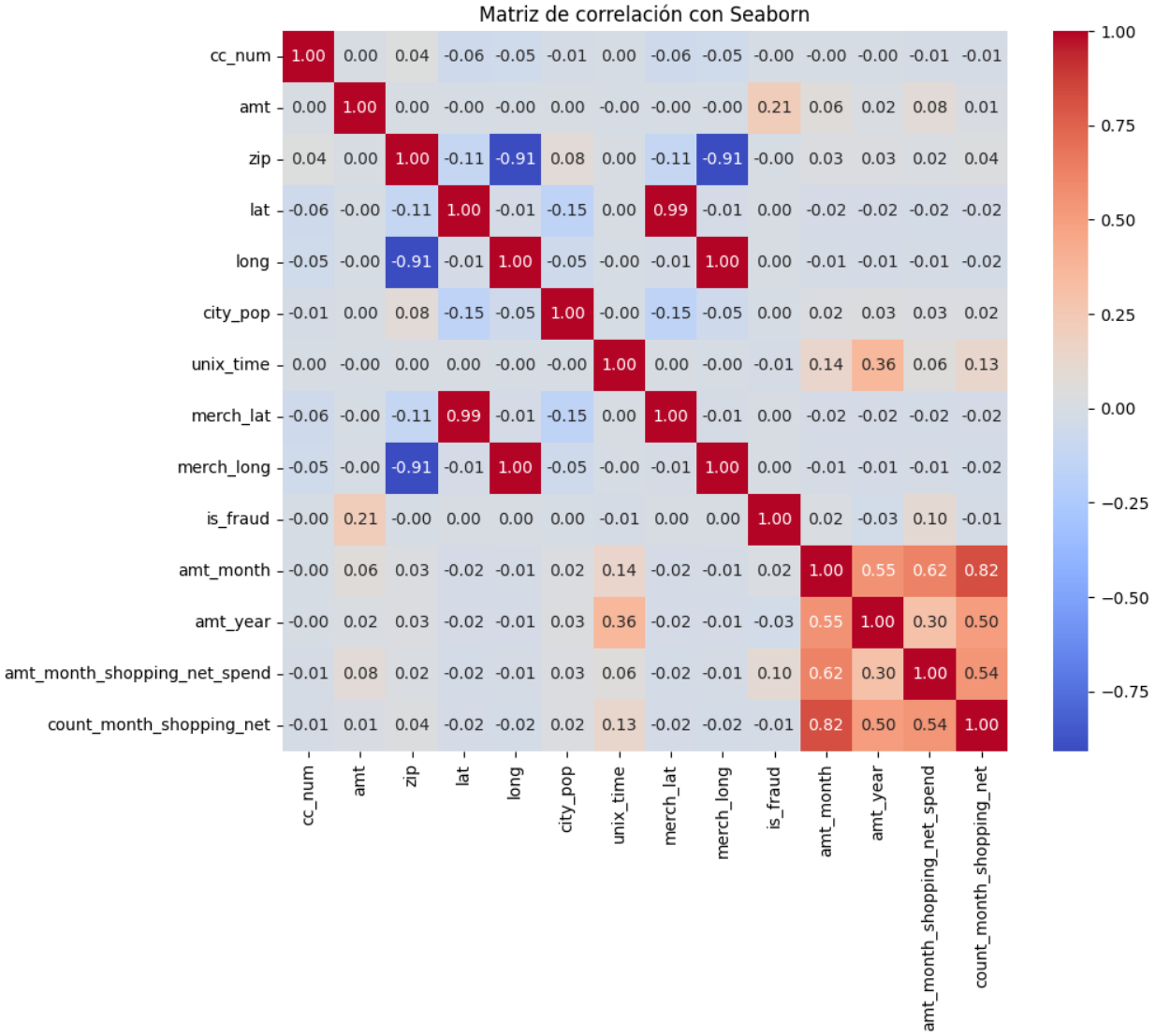
```
Distribución de clases:
is_fraud
0    1842743
1       9651
Name: count, dtype: int64
```

El conjunto de datos presenta una distribución altamente desbalanceada en la variable objetivo is_fraud. La mayoría de las transacciones están etiquetadas como no fraudulentas (clase 0), con un total de 1,842,743 instancias, mientras que solo hay 9,651 transacciones etiquetadas como fraudulentas (clase 1). Esta disparidad en el número de instancias entre las clases podría afectar el rendimiento de los modelos predictivos, lo que requerirá estrategias adecuadas de manejo de desbalance de clases durante el entrenamiento del modelo.

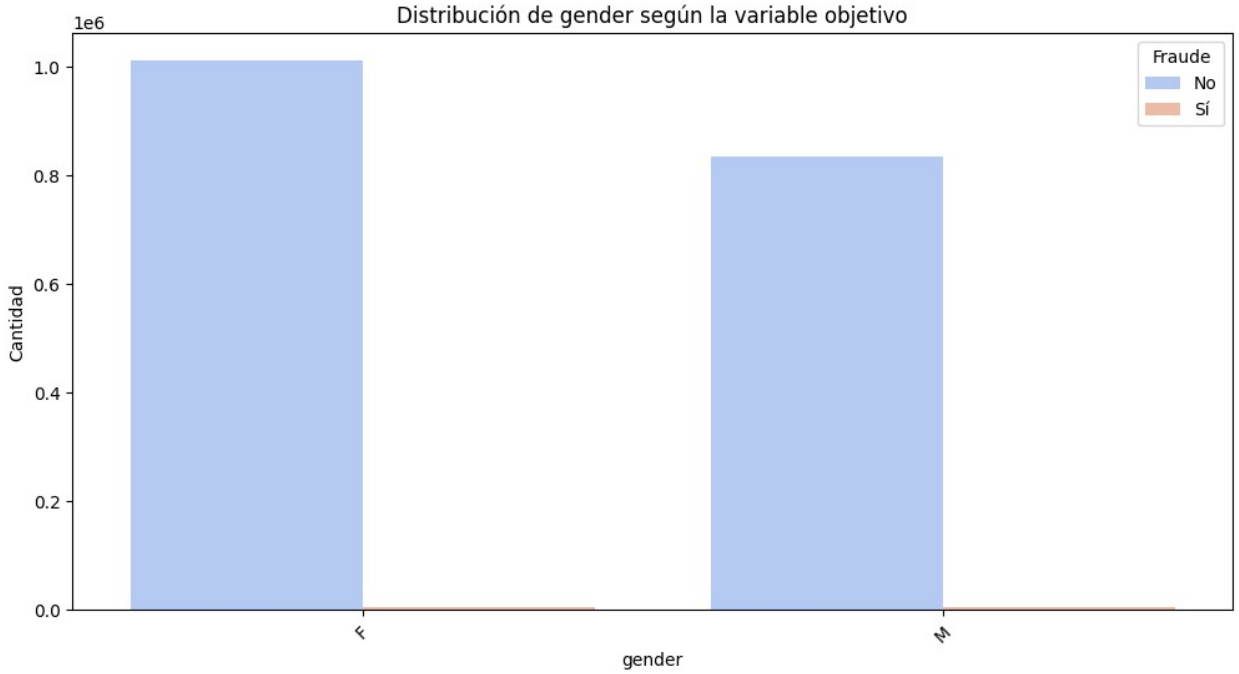
```
# Calcular la cantidad de transacciones fraudulentas y legítimas
fraud_counts = df['is_fraud'].value_counts()
total_transactions = fraud_counts.sum()

# Visualizar la distribución de la variable objetivo 'is_fraud'
plt.figure(figsize=(8, 6))
fraud_counts.plot(kind='bar', color=['green', 'red'])
plt.title('Distribución de transacciones fraudulentas y legítimas')
plt.xlabel('is_fraud')
plt.ylabel('Cantidad')
plt.xticks([0, 1], ['Legítima', 'Fraude'], rotation=0)

# Mostrar la proporción en el gráfico
for i, count in enumerate(fraud_counts):
    plt.text(i, count + total_transactions * 0.02, f"{count} ({count / total_transactions:.2%})", ha='center')

plt.show()
```

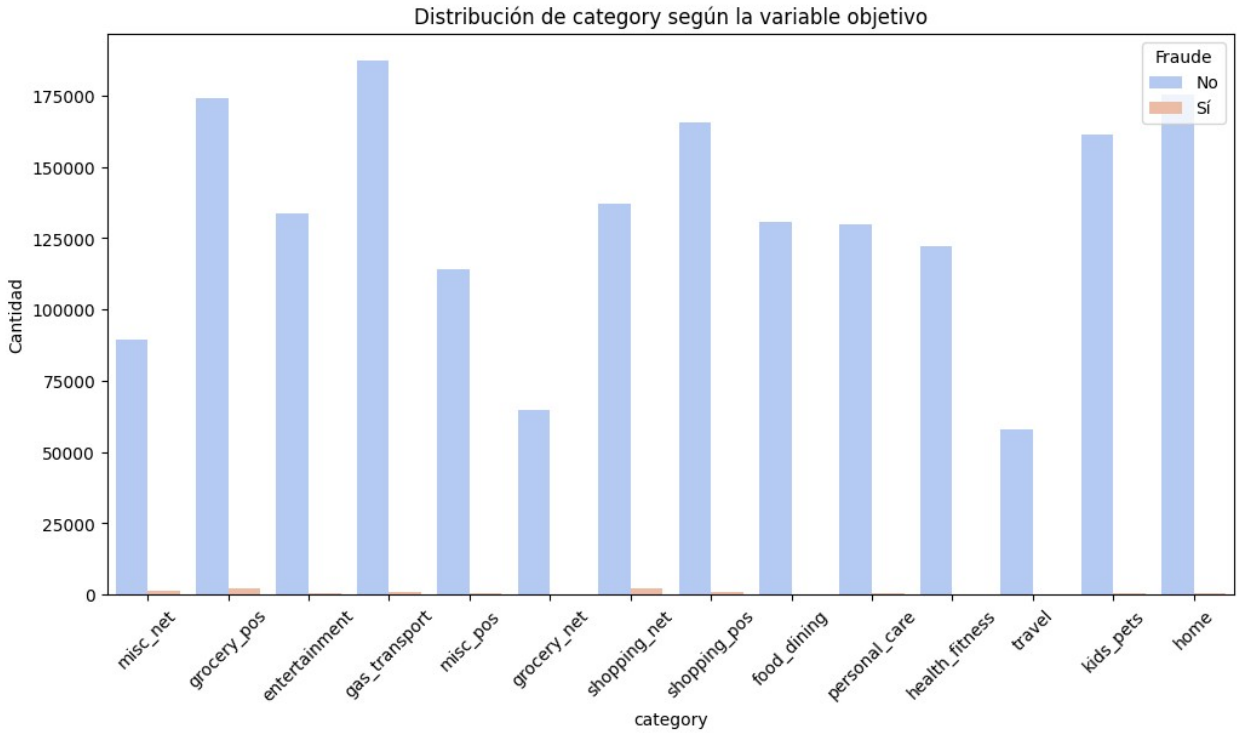


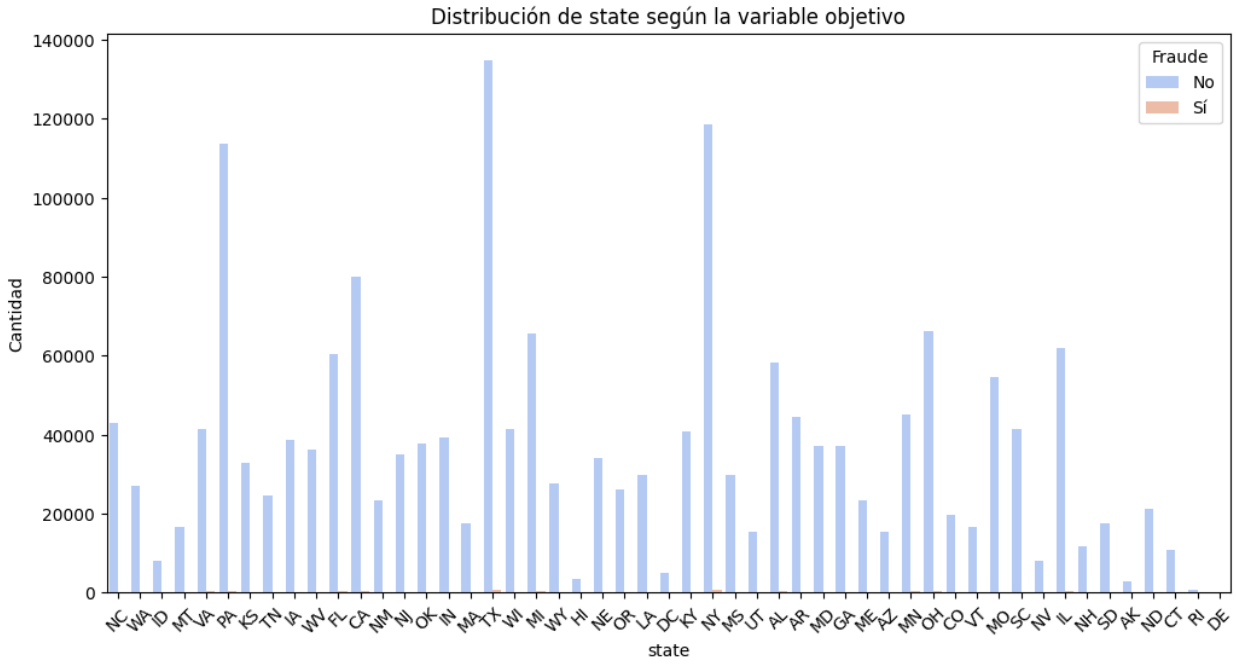

```
# Matriz de correlación
numeric_columns = df.select_dtypes(include=['float64', 'int64'])

correlation_matrix = numeric_columns.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt=".2f")
plt.title('Matriz de correlación con Seaborn')
plt.show()
```



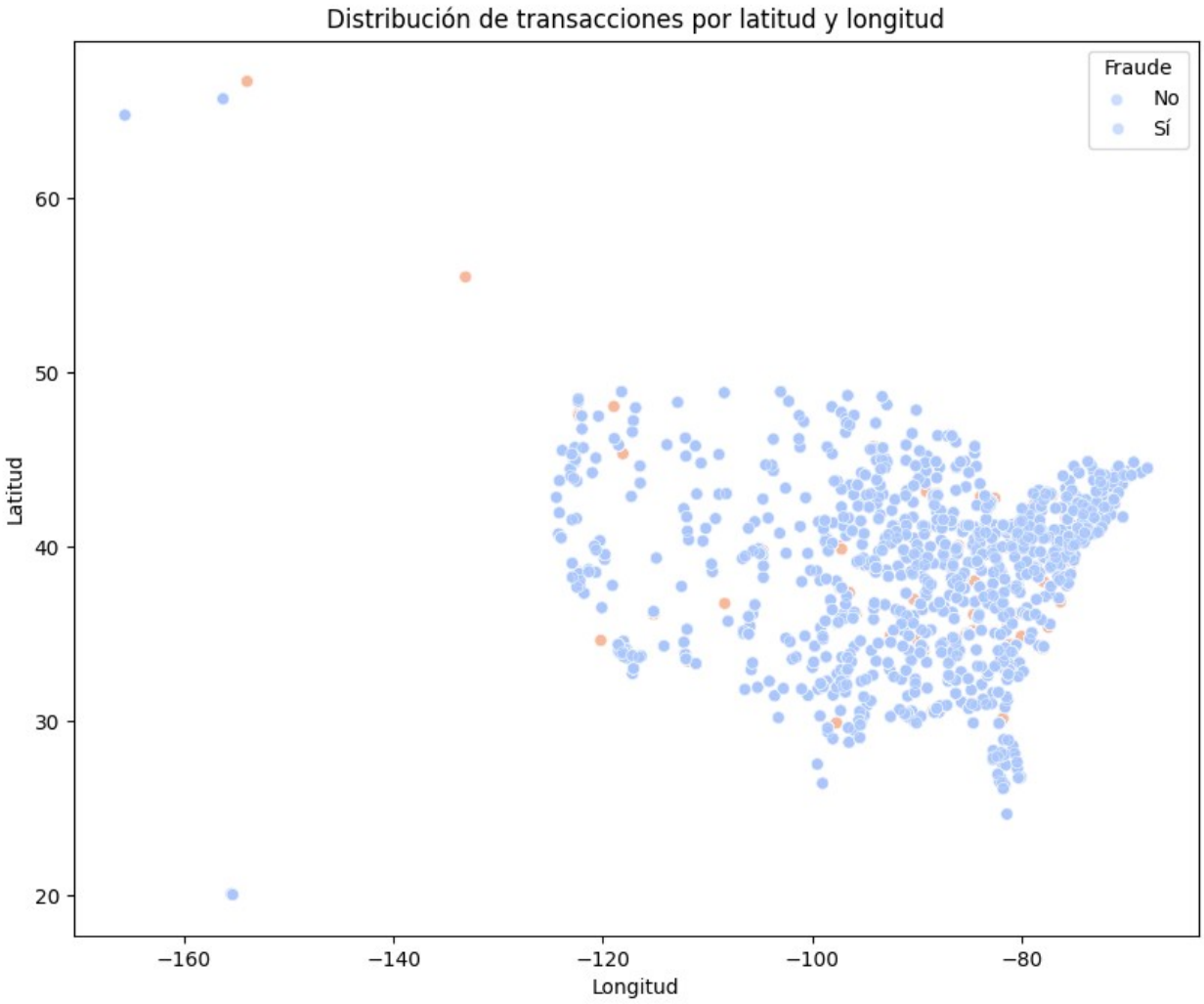

```
# Gráficos de barras para variables categóricas
categorical_variables = ['gender', 'category', 'state']  # Lista de
variables categóricas que deseas explorar

for column in categorical_variables:
    plt.figure(figsize=(12, 6))
    sns.countplot(data=df, x=column, hue='is_fraud',
palette='coolwarm')
    plt.title(f'Distribución de {column} según la variable objetivo')
    plt.xlabel(column)
    plt.ylabel('Cantidad')
    plt.xticks(rotation=45)
    plt.legend(title='Fraude', loc='upper right', labels=['No', 'Sí'])
    plt.show()
```


Distribución de gender según la variable objetivo
1e6
1.0
0.8
0.6
0.4
0.2
0.0
Cantidad
Fraude
No
Sí
F
M
gender


Distribución de category según la variable objetivo
175000
150000
125000
100000
75000
50000
25000
0
Cantidad
Fraude
No
Sí
misc_net
grocery_pos
entertainment
gas_transport
misc_pos
grocery_net
shopping_net
shopping_pos
food_dining
personal_care
health_fitness
travel
kids_pets
home
category




```
# Gráfico de densidad de dispersión
plt.figure(figsize=(10, 8))
sns.scatterplot(data=df, x='long', y='lat', hue='is_fraud',
palette='coolwarm', alpha=0.6)
plt.title('Distribución de transacciones por latitud y longitud')
plt.xlabel('Longitud')
plt.ylabel('Latitud')
plt.legend(title='Fraude', loc='upper right', labels=['No', 'Sí'])
plt.show()
```



Feature Engineering

Agregando características de la fecha

```
# Convertir la columna 'trans_date_trans_time' a formato de fecha y hora
df['trans_date_trans_time'] = pd.to_datetime(df['trans_date_trans_time'])

# Agregar características temporales
df['hour_of_day'] = df['trans_date_trans_time'].dt.hour
df['day_of_week'] = df['trans_date_trans_time'].dt.dayofweek  # Lunes=0, Domingo=6
df['month'] = df['trans_date_trans_time'].dt.month
df['year'] = df['trans_date_trans_time'].dt.year

# Mostrar las primeras filas del conjunto de datos para verificar las nuevas características
```

```
df[['trans_date_trans_time', 'hour_of_day', 'day_of_week', 'month',
'year']].head()
```

```
  trans_date_trans_time  hour_of_day  day_of_week  month  year
0   2019-01-01 00:00:18            0            1      1  2019
1   2019-01-01 00:00:44            0            1      1  2019
2   2019-01-01 00:00:51            0            1      1  2019
3   2019-01-01 00:01:16            0            1      1  2019
4   2019-01-01 00:03:06            0            1      1  2019
```

Agregando características de monto

```
# Calcular el monto promedio de transacción por cliente
avg_transaction_amt_by_customer = df.groupby('cc_num')['amt'].mean()
df['avg_transaction_amt'] =
df['cc_num'].map(avg_transaction_amt_by_customer)

# Calcular la frecuencia de transacciones por cliente
transaction_count_by_customer = df['cc_num'].value_counts()
df['transaction_count'] =
df['cc_num'].map(transaction_count_by_customer)

# Calcular la desviación estándar del monto de transacción por cliente
std_transaction_amt_by_customer = df.groupby('cc_num')['amt'].std()
df['std_transaction_amt'] =
df['cc_num'].map(std_transaction_amt_by_customer)

# Mostrar las primeras filas del conjunto de datos para verificar las
nuevas características
df[['amt', 'cc_num', 'avg_transaction_amt', 'std_transaction_amt',
'transaction_count']].head()
```

```
      amt            cc_num  avg_transaction_amt  std_transaction_amt
\
0    4.97  2703186189652095            89.408743           127.530101

1  107.23      630423337322            56.078113           159.201852

2  220.11    38859492057661            69.924272           116.688602

3   45.00  3534093764340240            80.090040           280.077880

4   41.96   375534208663984            95.341146            94.322842


   transaction_count
0               2927
1               4362
2                735
```

```
3                 743
4                2922

# Crear una nueva columna para almacenar la diversidad de comercios
visitados por cada tarjetahabiente en todo el conjunto de datos
df['unique_merchants_visited'] = df.groupby('cc_num')
['merchant'].transform('nunique')

# Mostrar las primeras filas del conjunto de datos para verificar las
nuevas características
df[['cc_num', 'merchant', 'unique_merchants_visited']].head()

              cc_num                                 merchant  \
0  2703186189652095          fraud_Rippin, Kub and Mann
1      630423337322     fraud_Heller, Gutmann and Zieme
2    38859492057661                  fraud_Lind-Buckridge
3  3534093764340240  fraud_Kutch, Hermiston and Farrell
4   375534208663984                  fraud_Keeling-Crist

   unique_merchants_visited
0                       660
1                       681
2                       431
3                       423
4                       652
```

Agregando características de localizacion

```
# Calcular la diferencia entre la latitud y la longitud del cliente y
del vendedor
df['customer_merchant_distance'] = np.sqrt((df['lat'] -
df['merch_lat'])**2 + (df['long'] - df['merch_long'])**2)

# Mostrar las primeras filas del conjunto de datos para verificar la
nueva característica
df[['lat', 'long', 'merch_lat', 'merch_long',
'customer_merchant_distance']].head()

       lat      long  merch_lat  merch_long
customer_merchant_distance
0  36.0788  -81.1781  36.011293  -82.048315
0.872830
1  48.8878 -118.2105  49.159047 -118.186462
0.272310
2  42.1808 -112.2620  43.150704 -112.154481
0.975845
3  46.2306 -112.1138  47.034331 -112.561071
0.919802
4  38.4207  -79.4629  38.674999  -78.632459
0.868505
```

Aplicando Label Encoder a las variables categóricas

```
# Inicializar LabelEncoder
label_encoder = LabelEncoder()

# Codificar variables categóricas
df['state'] = label_encoder.fit_transform(df['state'])
df['gender'] = label_encoder.fit_transform(df['gender'])
df['category'] = label_encoder.fit_transform(df['category'])
df['first_time_at_merchant'] =
label_encoder.fit_transform(df['first_time_at_merchant'])
```

Eliminando columnas innecesarias

```
# Variables a eliminar
variables_a_eliminar = ['trans_date_trans_time','trans_num', 'first',
'last', 'street', 'city', 'merchant', 'lat', 'long', 'city_pop',
'dob', 'unix_time', 'merch_lat', 'merch_long', 'job']

# Eliminar las variables del conjunto de datos
df = df.drop(variables_a_eliminar, axis=1)

# Mostrar las primeras filas del conjunto de datos para verificar los
cambios
df.head()
```

```
             cc_num  category     amt  gender  state    zip  is_fraud
\
0  2703186189652095         8    4.97       0     27  28654         0

1     630423337322         4  107.23       0     47  99160         0

2    38859492057661         0  220.11       1     13  83252         0

3  3534093764340240         2   45.00       1     26  59632         0

4   375534208663984         9   41.96       1     45  24433         0


   amt_month  amt_year  amt_month_shopping_net_spend  ...  \
0       4.97      4.97                           0.0  ...
1     107.23    107.23                           0.0  ...
2     220.11    220.11                           0.0  ...
3      45.00     45.00                           0.0  ...
4      41.96     41.96                           0.0  ...

   first_time_at_merchant  hour_of_day  day_of_week  month  year  \
0                       1            0            1      1  2019
1                       1            0            1      1  2019
2                       1            0            1      1  2019
```

```
3                       1            0            1      1  2019
4                       1            0            1      1  2019

   avg_transaction_amt  transaction_count  std_transaction_amt  \
0            89.408743               2927           127.530101
1            56.078113               4362           159.201852
2            69.924272                735           116.688602
3            80.090040                743           280.077880
4            95.341146               2922            94.322842

   unique_merchants_visited  customer_merchant_distance
0                       660                    0.872830
1                       681                    0.272310
2                       431                    0.975845
3                       423                    0.919802
4                       652                    0.868505

[5 rows x 21 columns]
```

Guardando el nuevo dataset

```
# guardar el dataframe modificado
df.to_csv('processed.csv', index=False)

df.head()

             cc_num  category     amt  gender  state    zip  is_fraud
\
0  2703186189652095         8    4.97       0     27  28654         0

1      630423337322         4  107.23       0     47  99160         0

2    38859492057661         0  220.11       1     13  83252         0

3  3534093764340240         2   45.00       1     26  59632         0

4   375534208663984         9   41.96       1     45  24433         0


   amt_month  amt_year  amt_month_shopping_net_spend  ...  \
0       4.97      4.97                           0.0  ...
1     107.23    107.23                           0.0  ...
2     220.11    220.11                           0.0  ...
3      45.00     45.00                           0.0  ...
4      41.96     41.96                           0.0  ...

   first_time_at_merchant  hour_of_day  day_of_week  month  year  \
0                       1            0            1      1  2019
1                       1            0            1      1  2019
2                       1            0            1      1  2019
```

```
3                       1            0            1      1  2019
4                       1            0            1      1  2019

   avg_transaction_amt  transaction_count  std_transaction_amt  \
0            89.408743               2927           127.530101
1            56.078113               4362           159.201852
2            69.924272                735           116.688602
3            80.090040                743           280.077880
4            95.341146               2922            94.322842

   unique_merchants_visited  customer_merchant_distance
0                       660                    0.872830
1                       681                    0.272310
2                       431                    0.975845
3                       423                    0.919802
4                       652                    0.868505

[5 rows x 21 columns]
```

SMOTE y División de Datos

```
df_processed = pd.read_csv('processed.csv')

print(df_processed.info())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1852394 entries, 0 to 1852393
Data columns (total 21 columns):
 #   Column                          Dtype
---  ------                          -----
 0   cc_num                          int64
 1   category                        int64
 2   amt                             float64
 3   gender                          int64
 4   state                           int64
 5   zip                             int64
 6   is_fraud                        int64
 7   amt_month                       float64
 8   amt_year                        float64
 9   amt_month_shopping_net_spend    float64
 10  count_month_shopping_net        float64
 11  first_time_at_merchant          int64
 12  hour_of_day                     int64
 13  day_of_week                     int64
 14  month                           int64
 15  year                            int64
 16  avg_transaction_amt             float64
 17  transaction_count               int64
 18  std_transaction_amt             float64
 19  unique_merchants_visited        int64
```

```
 20  customer_merchant_distance    float64
dtypes: float64(8), int64(13)
memory usage: 296.8 MB
None
```

La función `preprocess_and_resample` preprocesa y resamplea un dataset para la detección de fraude. Separa características y etiquetas, imputa valores faltantes, y escala las características numéricas. Luego, divide el dataset en entrenamiento, desarrollo y prueba. Utiliza SMOTE para balancear el conjunto de entrenamiento, ajustando la proporción de fraudes {0:1, 1:5}. Finalmente, retorna los conjuntos de datos resampleados y el conjunto de prueba, listos para el entrenamiento del modelo.

```
def preprocess_and_resample(df):
    # Seleccionar las características y la etiqueta
    X = df.drop(columns=['is_fraud'])
    y = df['is_fraud']

    # Imputar valores faltantes en las características numéricas
    imputer = SimpleImputer(strategy='mean')
    X_imputed = pd.DataFrame(imputer.fit_transform(X),
columns=X.columns)

    # Escalar las características numéricas para un mejor desempeño de
SMOTE
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X_imputed)

    # Dividir los datos en Train, Dev y Test
    X_train_dev, X_test, y_train_dev, y_test =
train_test_split(X_scaled, y, test_size=0.15, random_state=42)

    # Dividir los datos de Train y Dev
    X_train, X_dev, y_train, y_dev = train_test_split(X_train_dev,
y_train_dev, test_size=0.176, random_state=42)

    # Calcular la cantidad de transacciones legítimas y fraudulentas
en el conjunto de entrenamiento
    non_fraud_count = len(y_train[y_train == 0])
    fraud_count = non_fraud_count // 5

    # Definir la estrategia de muestreo para SMOTE
    sampling_strategy = {0: non_fraud_count, 1: fraud_count}
    print("Estrategia de muestreo para SMOTE:", sampling_strategy)

    # Aplicar SMOTE solo al conjunto de entrenamiento
    smote = SMOTE(sampling_strategy=sampling_strategy,
random_state=42)
    X_train_resampled, y_train_resampled = smote.fit_resample(X_train,
y_train)
```

```
    # Verificar la forma de los datos después de aplicar SMOTE
    print("Forma de X_train_resampled después de SMOTE:",
X_train_resampled.shape)
    print("Forma de y_train_resampled después de SMOTE:",
y_train_resampled.shape)

    return X_train_resampled, y_train_resampled, X_test, y_test,
X_dev, y_dev
```

Entrenamiento Inicial

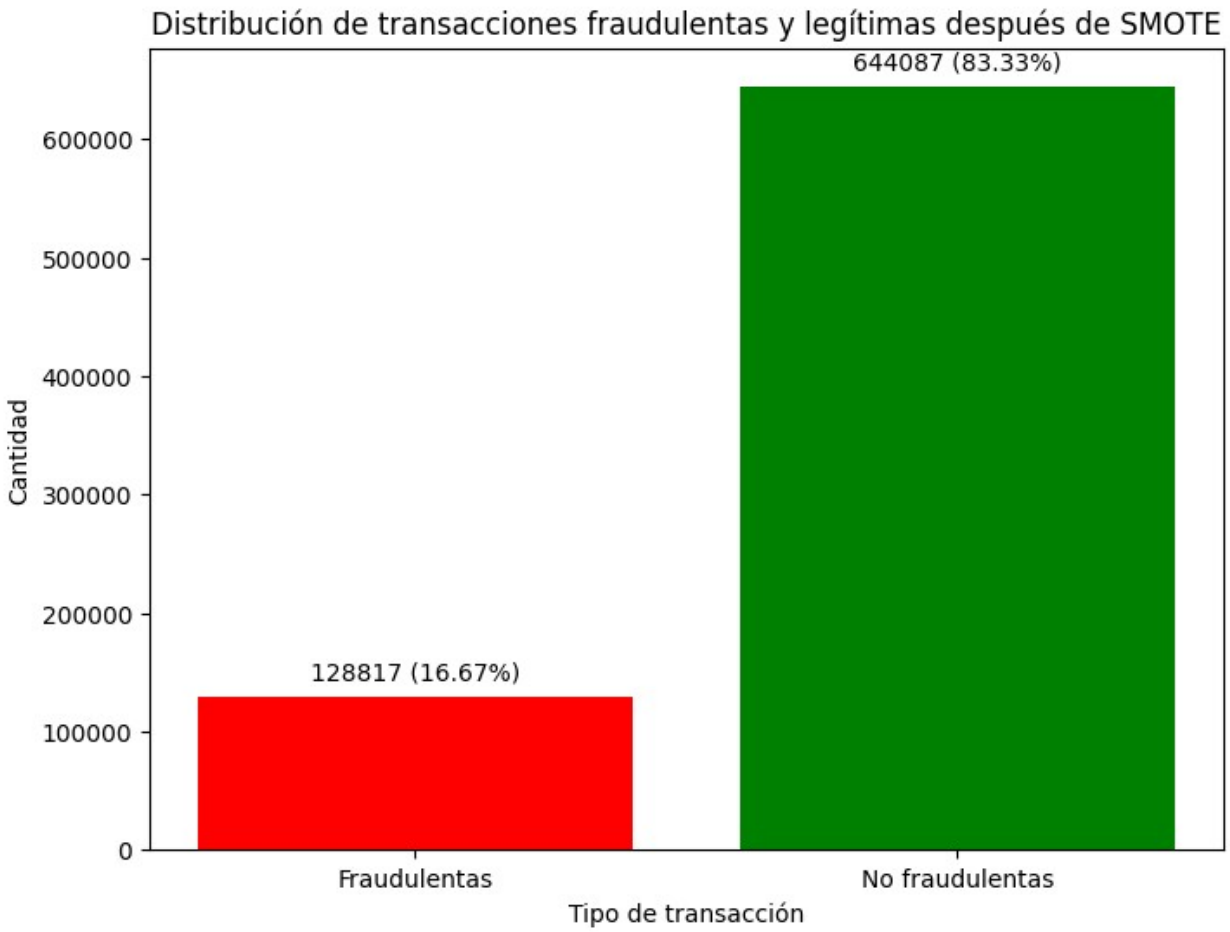
Para el entrenamiento incremental, se seleccionó todo el año 2019 como el conjunto de datos inicial. Esto se debe a que el año 2019 representa el conjunto de datos más antiguo y completo disponible al momento de iniciar el proceso de entrenamiento. Al tomar todo el año 2019 como punto de partida, el modelo tiene la oportunidad de aprender de la totalidad de los datos históricos disponibles antes de comenzar a actualizar y adaptarse a nuevos datos.

```
# Seleccionar solo los datos del año 2019
df_subset_2019 = df_processed[df_processed['year'] == 2019]

# Preprocesar y aplicar SMOTE a los datos del año 2019
X_train_resampled, y_train_resampled, X_test, y_test, X_dev, y_dev =
preprocess_and_resample(df_subset_2019)
```

```
Estrategia de muestreo para SMOTE: {0: 644087, 1: 128817}
Forma de X_train_resampled después de SMOTE: (772904, 20)
Forma de y_train_resampled después de SMOTE: (772904,)
```

```
# Calcular la cantidad de transacciones fraudulentas y legítimas
después de SMOTE
fraudulent_transactions_resampled = (y_train_resampled == 1).sum()
non_fraudulent_transactions_resampled = (y_train_resampled == 0).sum()
total_transactions_resampled = fraudulent_transactions_resampled +
non_fraudulent_transactions_resampled

# Visualizar la distribución de transacciones fraudulentas y legítimas
después de SMOTE
plt.figure(figsize=(8, 6))
plt.bar(['Fraudulentas', 'No fraudulentas'],
[fraudulent_transactions_resampled,
non_fraudulent_transactions_resampled], color=['red', 'green'])
plt.title('Distribución de transacciones fraudulentas y legítimas
después de SMOTE')
plt.xlabel('Tipo de transacción')
plt.ylabel('Cantidad')

# Mostrar la proporción en el gráfico
for i, count in enumerate([fraudulent_transactions_resampled,
non_fraudulent_transactions_resampled]):
    plt.text(i, count + total_transactions_resampled * 0.02, f"{count}
```

```
({count / total_transactions_resampled:.2%})", ha='center')

plt.show()
```



Implementación de Modelos

La función `model_metrics` evalúa el rendimiento del modelo usando un umbral basado en la curva ROC para mejorar el recall, lo que es crucial para detectar fraudes y minimizar falsos negativos. Predice las probabilidades sobre el conjunto de prueba y calcula el umbral óptimo basado en la diferencia entre la tasa de verdaderos positivos (TPR) y la tasa de falsos positivos (FPR). Luego, genera predicciones binarias y muestra métricas de evaluación como ROC-AUC, exactitud, recall y F1-score, además de graficar las curvas ROC y de precisión-recall y la matriz de confusión.

```
def model_metrics(model, dtest, y_test, model_name):

    # Predecir sobre el conjunto de prueba
    y_pred_proba = model.predict(dtest)

    # Calcular el umbral óptimo basado en la curva ROC, ya que mejora
```

```
el Recall, lo que es importante para detectar fraudes
    fpr, tpr, thresholds_roc = roc_curve(y_test, y_pred_proba)
    optimal_idx_roc = np.argmax(tpr - fpr)
    optimal_threshold_roc = thresholds_roc[optimal_idx_roc]
    # print(f"Umbral óptimo basado en ROC: {optimal_threshold_roc}")

    y_pred_proba = model.predict(dtest)
    y_pred = [1 if pred > optimal_threshold_roc else 0 for pred in
y_pred_proba]

    # Calcular y mostrar las métricas de evaluación
    print(f"Rendimiento del modelo {model_name}:")
    print("ROC-AUC:", roc_auc_score(y_test, y_pred_proba))
    # print("Precision:", precision_score(y_test, y_pred))
    print("Accuracy: ", accuracy_score(y_test, y_pred))
    # print("Accuracy: ", test_accuracy)
    print("Recall:", recall_score(y_test, y_pred))
    print("F1-score:", f1_score(y_test, y_pred))

    # Calcular la matriz de confusion
    conf_matrix = confusion_matrix(y_test, y_pred)
    print("Confusion Matrix:")
    print(conf_matrix)
    plt.figure(figsize=(6, 4))
    sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
                xticklabels=["Non-Fraud", "Fraud"],
                yticklabels=["Non-Fraud", "Fraud"])
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title(f"Confusion Matrix - {model_name}")
    plt.show()

    # Calcular la curva ROC
    fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
    roc_auc = auc(fpr, tpr)

    # Calcular la curva de precisión-recall
    precision, recall, _ = precision_recall_curve(y_test,
y_pred_proba)

    # Graficar la curva ROC
    plt.figure(figsize=(6, 4))
    plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (area =
{roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver Operating Characteristic (ROC) Curve -
```

```
{model_name}')
    plt.legend(loc="lower right")
    plt.grid(True)
    plt.show()

    # Graficar la curva de precisión-recall
    plt.figure(figsize=(6, 4))
    plt.plot(recall, precision, color='green', lw=2, label='Precision-
Recall curve')
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title(f'Precision-Recall Curve - {model_name}')
    plt.legend(loc="lower left")
    plt.grid(True)
    plt.show()
```

La función `model_recall` evalúa el recall del modelo utilizando un umbral óptimo basado en la curva ROC para mejorar la detección de verdaderos positivos. Predice las probabilidades en el conjunto de prueba, ajusta el umbral para maximizar el recall y luego muestra las métricas de evaluación, incluyendo ROC-AUC, Recall y la matriz de confusión.

```
def model_recall(model, dtest, y_test, model_name):

    # Predecir sobre el conjunto de prueba
    y_pred_proba = model.predict(dtest)

    # Calcular el umbral óptimo basado en la curva ROC
    fpr, tpr, thresholds_roc = roc_curve(y_test, y_pred_proba)
    optimal_idx_roc = np.argmax(tpr - fpr)
    optimal_threshold_roc = thresholds_roc[optimal_idx_roc]
    # print(f"Umbral óptimo basado en ROC: {optimal_threshold_roc}")

    y_pred_proba = model.predict(dtest)
    y_pred = [1 if pred > optimal_threshold_roc else 0 for pred in
y_pred_proba] # agarrar valor, area bajo la curva

    # Calcular y mostrar las métricas de evaluación
    print(f"Rendimiento del modelo {model_name}:")
    print("Recall:", recall_score(y_test, y_pred))

    # Calcular la matriz de confusion
    conf_matrix = confusion_matrix(y_test, y_pred)
    print(conf_matrix)
```

LGBM

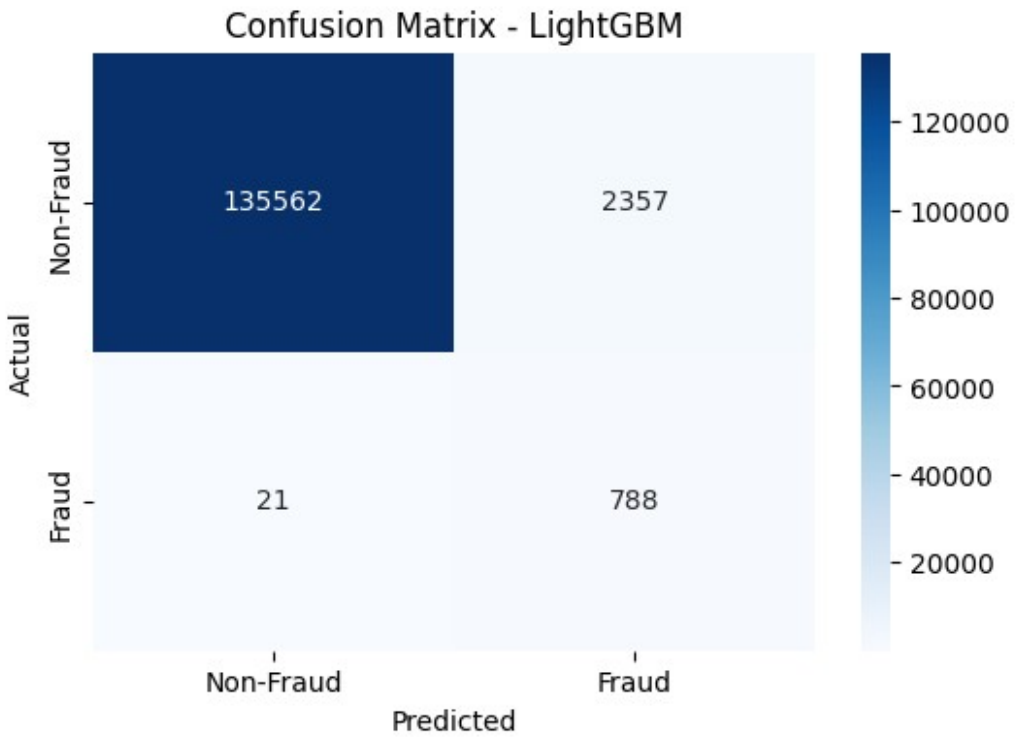
```
# Convertir los conjuntos de datos a DMatrix
train_data_batch = lgb.Dataset(X_train_resampled,
label=y_train_resampled)
```

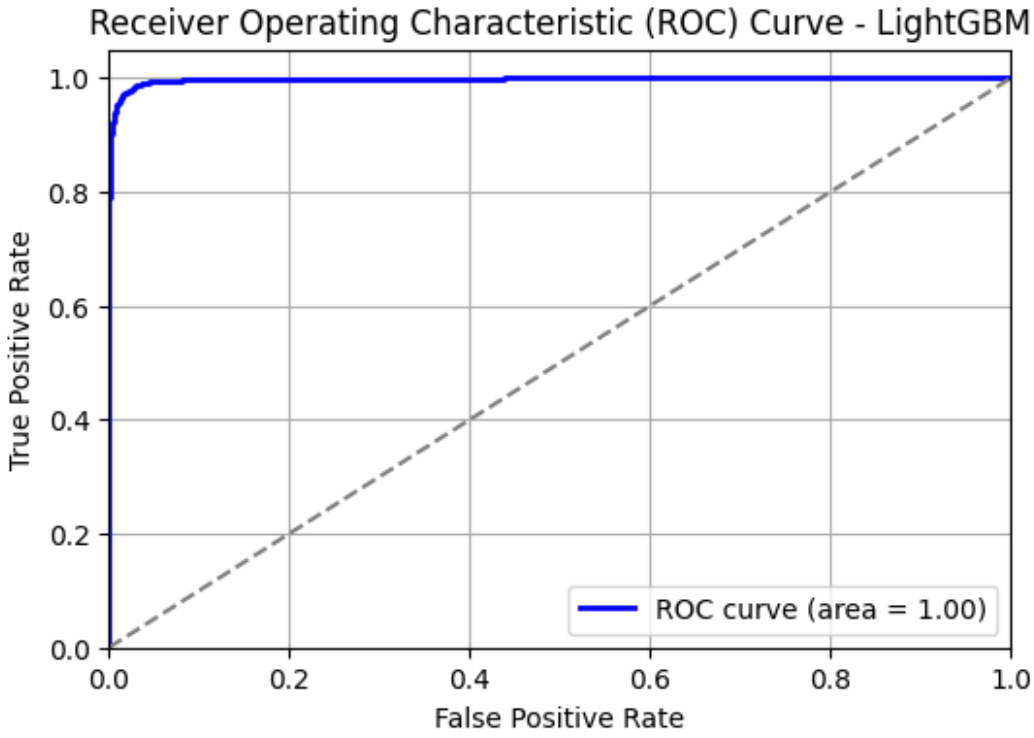
```python
# Definir los parámetros del modelo
params = {
    'objective': 'binary',
    'metric': 'binary_logloss',
    'verbose': -1,
}

# Configurar el número de rondas
num_rounds_batch = 100
lgbm_model = lgb.train(params, train_data_batch,
num_boost_round=num_rounds_batch)

# Guardar el modelo
lgbm_model.save_model('LGBM Models/lgb_model_2019.txt')
```

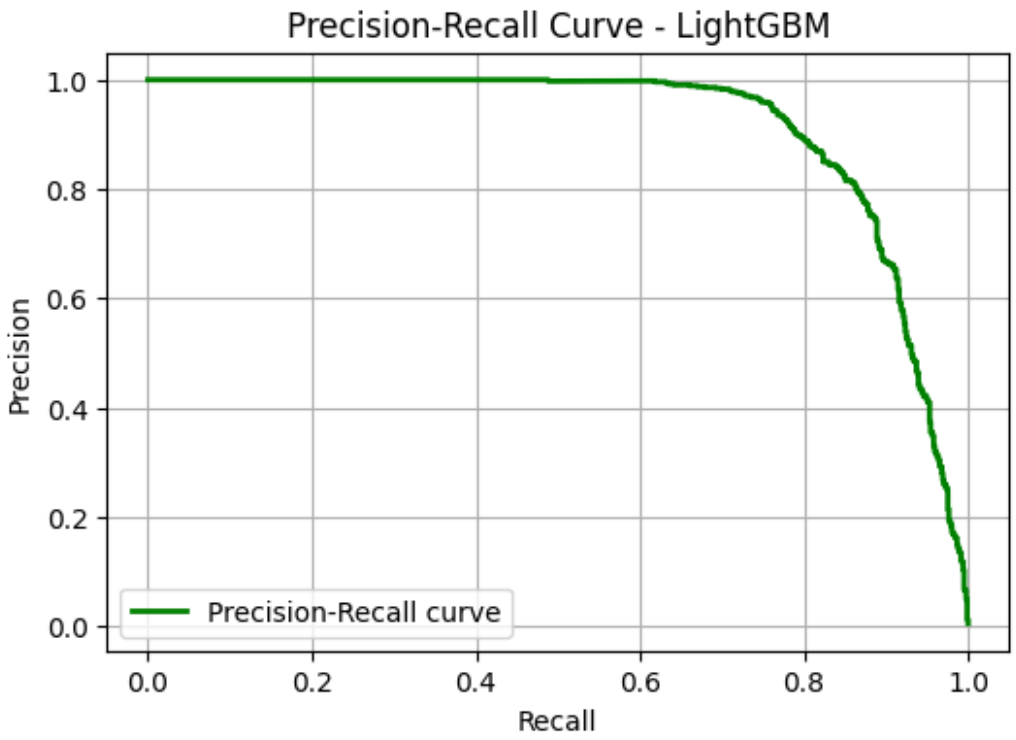
```
<lightgbm.basic.Booster at 0x24900ec7150>
```

```python
model_metrics(lgbm_model, X_test, y_test, "LightGBM")
```

```
Rendimiento del modelo LightGBM:
ROC-AUC: 0.9977734015265638
Accuracy:  0.9828585433366012
Recall: 0.9740420271940667
F1-score: 0.398583712696004
Confusion Matrix:
[[135562   2357]
 [    21    788]]
```


Confusion Matrix - LightGBM
Non-Fraud
Fraud
Actual
135562
2357
21
788
120000
100000
80000
60000
40000
20000
Non-Fraud
Fraud
Predicted


Receiver Operating Characteristic (ROC) Curve - LightGBM
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 1.00)



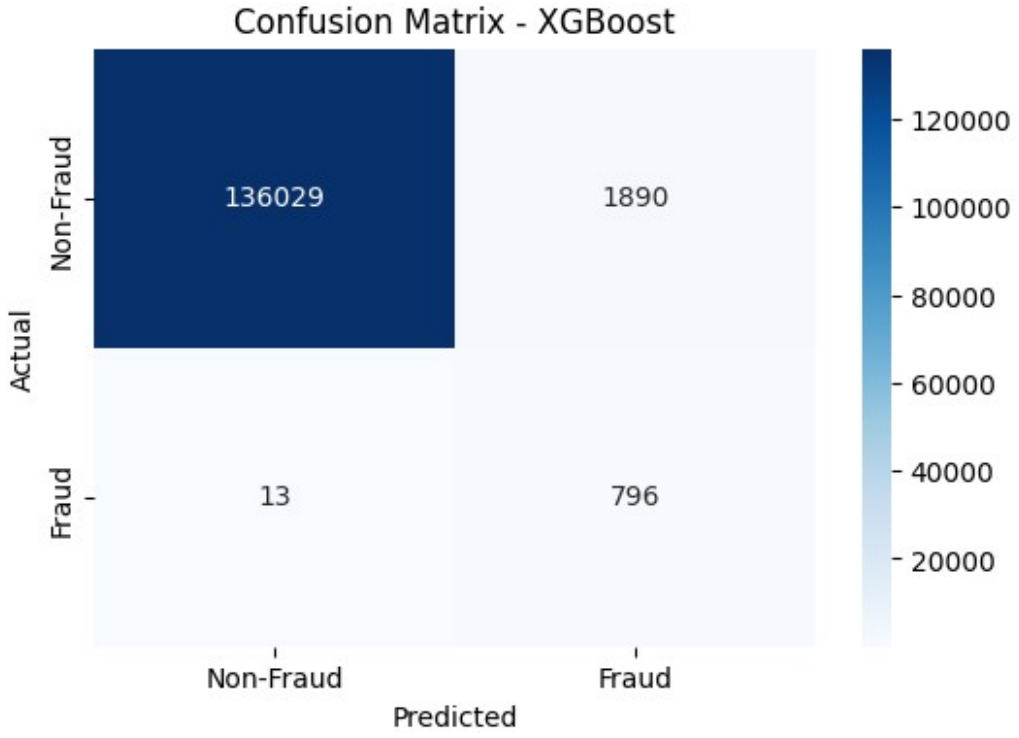
XGBoost

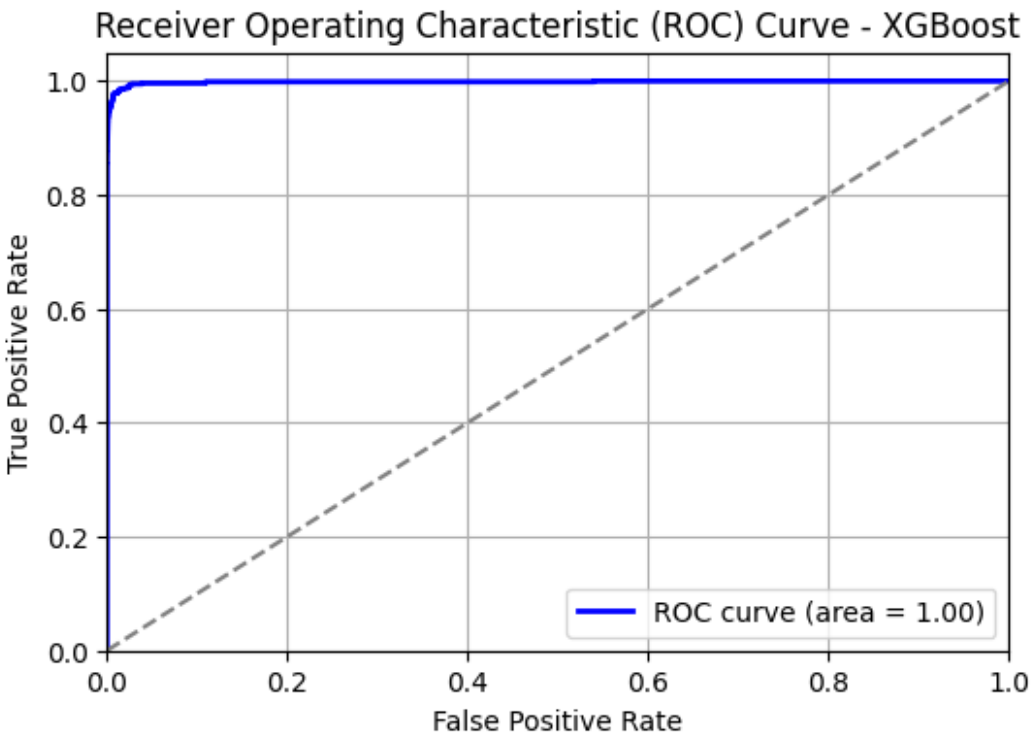
```
# Convertir los datos al formato DMatrix de XGBoost
dtrain = xgb.DMatrix(X_train_resampled, label=y_train_resampled)
dtest = xgb.DMatrix(X_test)

# Definir los parámetros del modelo
params = {
    'objective': 'binary:logistic',
    'eval_metric': 'logloss'
}

# Entrenar el modelo XGBoost
num_rounds = 100
xgb_model = xgb.train(params, dtrain, num_rounds)

# Guardar el modelo entrenado
xgb_model.save_model('XGB Models/xgb_model_2019.model')
```

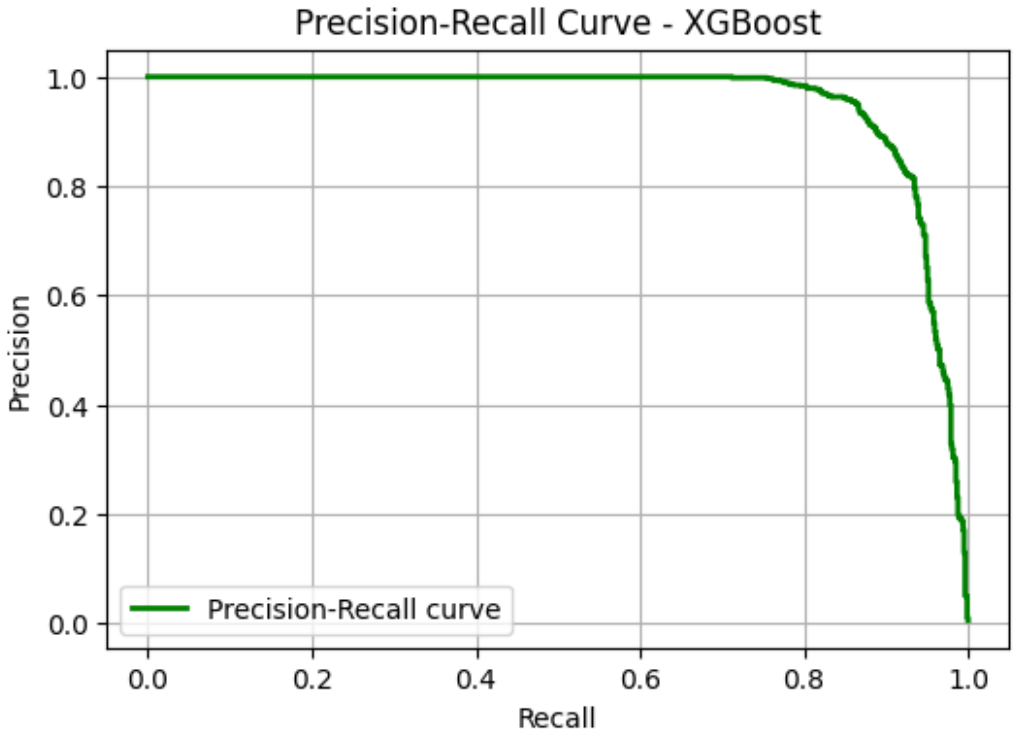
```
c:\Users\rjmom\AppData\Local\Programs\Python\Python311\Lib\site-
packages\xgboost\core.py:160: UserWarning: [12:51:52] WARNING: C:\
buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group-i-
0b3782d1791676daf-1\xgboost\xgboost-ci-windows\src\c_api\
c_api.cc:1240: Saving into deprecated binary model format, please
consider using `json` or `ubj`. Model format will default to JSON in
XGBoost 2.2 if not specified.
  warnings.warn(smsg, UserWarning)
```

```
model_metrics(xgb_model, dtest, y_test, 'XGBoost')
```

```
Rendimiento del modelo XGBoost:
ROC-AUC: 0.9984410960622693
Accuracy:  0.9862825096591892
Recall: 0.9839307787391842
F1-score: 0.45550786838340485
Confusion Matrix:
[[136029   1890]
 [    13    796]]
```







Reentrenamientos

Para llevar a cabo el reentrenamiento incremental, se emplearán las funciones `lgbm_incremental_training` y `xgb_incremental_training`, las cuales requieren como

parámetros el dataframe de datos procesados y los intervalos de tiempo (semestres o trimestres) para ejecutar los reentrenamientos de manera apropiada. Estas funciones permiten actualizar continuamente los modelos de LightGBM y XGBoost con los nuevos datos disponibles en intervalos definidos, garantizando una adaptación efectiva a los cambios en los datos a lo largo del tiempo.

```
# Función para realizar el reentrenamiento incremental semestral
def lgbm_incremental_training(df_processed, tiempos,
initial_model_path, output_model_path_prefix):

    # Definir los tiempos de reentrenamiento
    if tiempos == 'semestre':
        meses = [(1, 6), (7, 12)]
        inicial = 'S'
    elif tiempos == 'trimestre':
        meses = [(1, 3), (4, 6), (7, 9), (10, 12)]
        inicial = 'T'

    # Cargar el modelo inicial (entrenado en 2019)
    lgbm_model_incremental =
lgb.Booster(model_file=initial_model_path)

    # Extraer los parámetros del modelo inicial
    params = lgbm_model_incremental.params

    # Eliminar parámetros específicos que no son necesarios para el
reentrenamiento
    params.pop('early_stopping_round', None)

    for i, (start_month, end_month) in enumerate(meses):
        print(f"\nReentrenamiento incremental para el {inicial}
{i+1}:")

        # Seleccionar los datos del semestre actual
        df_subset = df_processed[(df_processed['year'] == 2020) &
                                 (df_processed['month'] >=
start_month) &
                                 (df_processed['month'] <= end_month)]

        # Preprocesar y aplicar SMOTE a los datos del semestre actual
        X_train_resampled_batch, y_train_resampled_batch,
X_test_batch, y_test_batch, X_dev_batch, y_dev_batch =
preprocess_and_resample(df_subset)

        # Convertir los datos al formato Dataset de LightGBM
        train_data_batch = lgb.Dataset(X_train_resampled_batch,
label=y_train_resampled_batch)

        # Continuar entrenando el modelo con los nuevos datos del
semestre
```

```
        num_rounds_batch = 100
        lgbm_model_incremental = lgb.train(params, train_data_batch,
num_boost_round=num_rounds_batch, init_model=lgbm_model_incremental)

        # Guardar el modelo actualizado después de cada semestre
        model_path = f'{output_model_path_prefix}_{inicial}{i+1}.txt'
        lgbm_model_incremental.save_model(model_path)

        print("")
        # Evaluar el modelo actualizado
        model_recall(lgbm_model_incremental, X_test_batch,
y_test_batch, f'LightGBM_{inicial}{i+1}')
        print("")

        # Cargar el modelo actualizado para el próximo semestre
        lgbm_model_incremental = lgb.Booster(model_file=model_path)

# Función para realizar el reentrenamiento incremental semestral con
XGBoost
def xgb_incremental_training(df_processed, tiempos,
initial_model_path, output_model_path_prefix):

    # Definir los tiempos de entrenamiento
    if tiempos == 'semestre':
        meses = [(1, 7), (8, 12)]
        inicial = 'S'
    elif tiempos == 'trimestre':
        meses = [(1, 3), (4, 6), (7, 9), (10, 12)]
        inicial = 'T'

    # Cargar el modelo inicial (entrenado en 2019)
    xgb_model_incremental = xgb.Booster()
    xgb_model_incremental.load_model(initial_model_path)

    for i, (start_month, end_month) in enumerate(meses):
        print(f"\nReentrenamiento incremental para el {inicial}
{i+1}:")

        # Seleccionar los datos del semestre actual
        df_subset = df_processed[(df_processed['year'] == 2020) &
                                 (df_processed['month'] >=
start_month) &
                                 (df_processed['month'] <= end_month)]

        # Preprocesar y aplicar SMOTE a los datos del semestre actual
        X_train_resampled_batch, y_train_resampled_batch,
X_test_batch, y_test_batch, X_dev_batch, y_dev_batch =
preprocess_and_resample(df_subset)

        # Convertir los datos del batch al formato DMatrix de XGBoost
```

```
        dtrain_batch = xgb.DMatrix(X_train_resampled_batch,
label=y_train_resampled_batch)

        # Entrenar el modelo XGBoost con el nuevo batch de datos
        num_rounds_batch = 100
        xgb_model_incremental.update(dtrain_batch, num_rounds_batch)

        # Guardar el modelo actualizado después de cada semestre
        model_path = f'{output_model_path_prefix}_{inicial}
{i+1}.model'
        xgb_model_incremental.save_model(model_path)

        # Convertir los datos de prueba al formato DMatrix de XGBoost
        dtest_batch = xgb.DMatrix(X_test_batch)

        print("")
        # Evaluar el modelo actualizado
        model_recall(xgb_model_incremental, dtest_batch, y_test_batch,
f'LightGBM_{inicial}{i+1}')
        print("")

        # Cargar el modelo actualizado para el próximo semestre
        xgb_model_incremental = xgb.Booster()
        xgb_model_incremental.load_model(model_path)
```

Entrenamiento incremental semestral

Para el reentrenamiento incremental semestral, se utilizarán datos del año 2020, divididos en dos partes: enero a junio y julio a diciembre. Se espera que este enfoque le permita al modelo actualizarse con una cantidad significativa de datos recientes, capturando nuevas tendencias y patrones a medida que se desarrollan a lo largo del año.

LGBM

```
# Llamar a la función para realizar el reentrenamiento semestral
lgbm_incremental_training(df_processed, 'semestre', 'LGBM
Models/lgb_model_2019.txt', 'LGBM Models/lgb_model_2020')


Reentrenamiento incremental para el S1:
Estrategia de muestreo para SMOTE: {0: 279758, 1: 55951}
Forma de X_train_resampled después de SMOTE: (335709, 20)
Forma de y_train_resampled después de SMOTE: (335709,)

c:\Users\rjmom\AppData\Local\Programs\Python\Python311\Lib\site-
packages\lightgbm\engine.py:172: UserWarning: Found `num_iterations`
in params. Will use it instead of argument
  _log_warning(f"Found `{alias}` in params. Will use it instead of
argument")
```

```
Rendimiento del modelo LightGBM_S1:
Recall: 0.9651162790697675
[[58867  1072]
 [   12   332]]


Reentrenamiento incremental para el S2:
Estrategia de muestreo para SMOTE: {0: 366729, 1: 73345}
Forma de X_train_resampled después de SMOTE: (440074, 20)
Forma de y_train_resampled después de SMOTE: (440074,)

c:\Users\rjmom\AppData\Local\Programs\Python\Python311\Lib\site-
packages\lightgbm\engine.py:172: UserWarning: Found `num_iterations`
in params. Will use it instead of argument
  _log_warning(f"Found `{alias}` in params. Will use it instead of
argument")


Rendimiento del modelo LightGBM_S2:
Recall: 0.9013605442176871
[[76215  2341]
 [   29   265]]
```

XGBoost

```
# Llamar a la función para realizar el reentrenamiento semestral con
XGBoost
xgb_incremental_training(df_processed, 'semestre', 'XGB
Models/xgb_model_2019.model', 'XGB Models/xgb_model_2020')


Reentrenamiento incremental para el S1:
Estrategia de muestreo para SMOTE: {0: 339678, 1: 67935}
Forma de X_train_resampled después de SMOTE: (407613, 20)
Forma de y_train_resampled después de SMOTE: (407613,)

c:\Users\rjmom\AppData\Local\Programs\Python\Python311\Lib\site-
packages\xgboost\core.py:160: UserWarning: [12:52:03] WARNING: C:\
buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group-i-
0b3782d1791676daf-1\xgboost\xgboost-ci-windows\src\c_api\
c_api.cc:1240: Saving into deprecated binary model format, please
consider using `json` or `ubj`. Model format will default to JSON in
XGBoost 2.2 if not specified.
  warnings.warn(smsg, UserWarning)


Rendimiento del modelo LightGBM_S1:
Recall: 0.8221153846153846
[[70023  2721]
```

```
 [   74   342]]


Reentrenamiento incremental para el S2:
Estrategia de muestreo para SMOTE: {0: 306861, 1: 61372}
Forma de X_train_resampled después de SMOTE: (368233, 20)
Forma de y_train_resampled después de SMOTE: (368233,)

c:\Users\rjmom\AppData\Local\Programs\Python\Python311\Lib\site-
packages\xgboost\core.py:160: UserWarning: [12:52:05] WARNING: C:\
buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group-i-
0b3782d1791676daf-1\xgboost\xgboost-ci-windows\src\c_api\
c_api.cc:1240: Saving into deprecated binary model format, please
consider using `json` or `ubj`. Model format will default to JSON in
XGBoost 2.2 if not specified.
  warnings.warn(smsg, UserWarning)


Rendimiento del modelo LightGBM_S2:
Recall: 0.8918918918918919
[[55737  9976]
 [   28   231]]
```

Entrenamiento incremental trimestral

En el reentrenamiento incremental trimestral, también se emplearán los datos del 2020, pero divididos en cuatro trimestres. Esta estrategia proporciona actualizaciones más frecuentes, por lo que se espera que el modelo se adapte rápidamente a cambios en los datos y de esta manera mejorar su capacidad para detectar fraudes en un entorno dinámico.

LGBM

```
# Llamar a la función para realizar el reentrenamiento trimestral con
LightGBM
lgbm_incremental_training(df_processed,'trimestre', 'LGBM
Models/lgb_model_2019.txt', 'LGBM Models/lgb_model_2020')


Reentrenamiento incremental para el T1:
Estrategia de muestreo para SMOTE: {0: 120297, 1: 24059}
Forma de X_train_resampled después de SMOTE: (144356, 20)
Forma de y_train_resampled después de SMOTE: (144356,)

c:\Users\rjmom\AppData\Local\Programs\Python\Python311\Lib\site-
packages\lightgbm\engine.py:172: UserWarning: Found `num_iterations`
in params. Will use it instead of argument
  _log_warning(f"Found `{alias}` in params. Will use it instead of
argument")
```

```
Rendimiento del modelo LightGBM_T1:
Recall: 0.9655172413793104
[[25347   406]
 [    6   168]]


Reentrenamiento incremental para el T2:
Estrategia de muestreo para SMOTE: {0: 159514, 1: 31902}
Forma de X_train_resampled después de SMOTE: (191416, 20)
Forma de y_train_resampled después de SMOTE: (191416,)

c:\Users\rjmom\AppData\Local\Programs\Python\Python311\Lib\site-
packages\lightgbm\engine.py:172: UserWarning: Found `num_iterations`
in params. Will use it instead of argument
  _log_warning(f"Found `{alias}` in params. Will use it instead of
argument")


Rendimiento del modelo LightGBM_T2:
Recall: 0.9387755102040817
[[33426   734]
 [   12   184]]


Reentrenamiento incremental para el T3:
Estrategia de muestreo para SMOTE: {0: 170237, 1: 34047}
Forma de X_train_resampled después de SMOTE: (204284, 20)
Forma de y_train_resampled después de SMOTE: (204284,)

c:\Users\rjmom\AppData\Local\Programs\Python\Python311\Lib\site-
packages\lightgbm\engine.py:172: UserWarning: Found `num_iterations`
in params. Will use it instead of argument
  _log_warning(f"Found `{alias}` in params. Will use it instead of
argument")


Rendimiento del modelo LightGBM_T3:
Recall: 0.8048780487804879
[[35548   909]
 [   32   132]]


Reentrenamiento incremental para el T4:
Estrategia de muestreo para SMOTE: {0: 196505, 1: 39301}
Forma de X_train_resampled después de SMOTE: (235806, 20)
Forma de y_train_resampled después de SMOTE: (235806,)

c:\Users\rjmom\AppData\Local\Programs\Python\Python311\Lib\site-
packages\lightgbm\engine.py:172: UserWarning: Found `num_iterations`
in params. Will use it instead of argument
```

```
  _log_warning(f"Found `{alias}` in params. Will use it instead of
argument")


Rendimiento del modelo LightGBM_T4:
Recall: 0.7777777777777778
[[41484   610]
 [   30   105]]
```

XGBoost

```
# Llamar a la función para realizar el reentrenamiento trimestral con
XGBoost
xgb_incremental_training(df_processed, 'trimestre', 'XGB
Models/xgb_model_2019.model', 'XGB Models/xgb_model_2020')


Reentrenamiento incremental para el T1:
Estrategia de muestreo para SMOTE: {0: 120297, 1: 24059}
Forma de X_train_resampled después de SMOTE: (144356, 20)
Forma de y_train_resampled después de SMOTE: (144356,)

c:\Users\rjmom\AppData\Local\Programs\Python\Python311\Lib\site-
packages\xgboost\core.py:160: UserWarning: [12:52:17] WARNING: C:\
buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group-i-
0b3782d1791676daf-1\xgboost\xgboost-ci-windows\src\c_api\
c_api.cc:1240: Saving into deprecated binary model format, please
consider using `json` or `ubj`. Model format will default to JSON in
XGBoost 2.2 if not specified.
  warnings.warn(smsg, UserWarning)


Rendimiento del modelo LightGBM_T1:
Recall: 0.9080459770114943
[[23350  2403]
 [   16   158]]


Reentrenamiento incremental para el T2:
Estrategia de muestreo para SMOTE: {0: 159514, 1: 31902}
Forma de X_train_resampled después de SMOTE: (191416, 20)
Forma de y_train_resampled después de SMOTE: (191416,)

c:\Users\rjmom\AppData\Local\Programs\Python\Python311\Lib\site-
packages\xgboost\core.py:160: UserWarning: [12:52:18] WARNING: C:\
buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group-i-
0b3782d1791676daf-1\xgboost\xgboost-ci-windows\src\c_api\
c_api.cc:1240: Saving into deprecated binary model format, please
consider using `json` or `ubj`. Model format will default to JSON in
```

```
XGBoost 2.2 if not specified.
  warnings.warn(smsg, UserWarning)


Rendimiento del modelo LightGBM_T2:
Recall: 0.7091836734693877
[[31482  2678]
 [   57   139]]


Reentrenamiento incremental para el T3:
Estrategia de muestreo para SMOTE: {0: 170237, 1: 34047}
Forma de X_train_resampled después de SMOTE: (204284, 20)
Forma de y_train_resampled después de SMOTE: (204284,)

c:\Users\rjmom\AppData\Local\Programs\Python\Python311\Lib\site-
packages\xgboost\core.py:160: UserWarning: [12:52:19] WARNING: C:\
buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group-i-
0b3782d1791676daf-1\xgboost\xgboost-ci-windows\src\c_api\
c_api.cc:1240: Saving into deprecated binary model format, please
consider using `json` or `ubj`. Model format will default to JSON in
XGBoost 2.2 if not specified.
  warnings.warn(smsg, UserWarning)


Rendimiento del modelo LightGBM_T3:
Recall: 0.6829268292682927
[[32931  3526]
 [   52   112]]


Reentrenamiento incremental para el T4:
Estrategia de muestreo para SMOTE: {0: 196505, 1: 39301}
Forma de X_train_resampled después de SMOTE: (235806, 20)
Forma de y_train_resampled después de SMOTE: (235806,)

Rendimiento del modelo LightGBM_T4:
Recall: 0.8518518518518519
[[36934  5160]
 [   20   115]]


c:\Users\rjmom\AppData\Local\Programs\Python\Python311\Lib\site-
packages\xgboost\core.py:160: UserWarning: [12:52:20] WARNING: C:\
buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group-i-
0b3782d1791676daf-1\xgboost\xgboost-ci-windows\src\c_api\
c_api.cc:1240: Saving into deprecated binary model format, please
consider using `json` or `ubj`. Model format will default to JSON in
XGBoost 2.2 if not specified.
  warnings.warn(smsg, UserWarning)
```

Reentrenamiento total

Para realizar un reentrenamiento total, utilizamos todos los datos disponibles en el dataframe, abarcando tanto el año 2019 como el año 2020. Esto permite que el modelo capture de manera efectiva tanto las tendencias históricas como las más recientes en el conjunto de datos. Al entrenar el modelo desde cero con esta información completa, se busca optimizar su rendimiento y capacidad de generalización, garantizando una mayor precisión en la detección de fraudes.

```
# Seleccionar todos los datos del dataframe
df_subset_all = df_processed.copy()

# Preprocesar y aplicar SMOTE a todos los datos
X_train_resampled_all, y_train_resampled_all, X_test_all, y_test_all,
X_dev_all, y_dev_all = preprocess_and_resample(df_subset_all)
```

```
Estrategia de muestreo para SMOTE: {0: 1290690, 1: 258138}
Forma de X_train_resampled después de SMOTE: (1548828, 20)
Forma de y_train_resampled después de SMOTE: (1548828,)
```

LGBM

```
# Convertir los conjuntos de datos a lgb.Dataset
train_data_batch = lgb.Dataset(X_train_resampled_all,
label=y_train_resampled_all)

# Definir los parámetros del modelo
params = {
    'objective': 'binary',
    'metric': 'binary_logloss',
    'verbose': -1,
}

# Configurar el número de rondas
num_rounds_batch = 100
lgbm_model_retrained = lgb.train(params, train_data_batch,
num_boost_round=num_rounds_batch)

# Guardar el modelo
lgbm_model_retrained.save_model('LGBM Models/lgb_model_retrained.txt')
```

```
<lightgbm.basic.Booster at 0x2490109fdd0>
```

```
model_recall(lgbm_model_retrained, X_test_all, y_test_all, "LightGBM
(Retrained)")
```

```
Rendimiento del modelo LightGBM (Retrained):
Recall: 0.9817320703653586
[[270945   5437]
 [    27   1451]]
```

XGBoost

```
# Convertir los datos al formato DMatrix de XGBoost
dtrain = xgb.DMatrix(X_train_resampled, label=y_train_resampled)
dtest_all = xgb.DMatrix(X_test_all)

# Definir los parámetros del modelo
params = {
    'objective': 'binary:logistic',
    'eval_metric': 'logloss'
}

# Entrenar el modelo XGBoost
num_rounds = 100
xgb_model_retrained = xgb.train(params, dtrain, num_rounds)

# Guardar el modelo entrenado
xgb_model_retrained.save_model('XGB Models/xgb_model_retrained.model')
```

```
c:\Users\rjmom\AppData\Local\Programs\Python\Python311\Lib\site-
packages\xgboost\core.py:160: UserWarning: [12:52:38] WARNING: C:\
buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group-i-
0b3782d1791676daf-1\xgboost\xgboost-ci-windows\src\c_api\
c_api.cc:1240: Saving into deprecated binary model format, please
consider using `json` or `ubj`. Model format will default to JSON in
XGBoost 2.2 if not specified.
  warnings.warn(smsg, UserWarning)
```

```
model_recall(xgb_model_retrained, dtest_all, y_test_all, 'XGBoost
(Retrained)')
```

```
Rendimiento del modelo XGBoost (Retrained):
Recall: 0.8470906630581867
[[238576  37806]
 [   226   1252]]
```

Comparando Resultados

Para comparar los modelos, se utilizó un enfoque consistente y robusto. Se empleó el mismo conjunto de prueba para evaluar todos los modelos, lo que garantiza una comparación justa y precisa de su rendimiento. Mediante funciones específicas, se calculó la media del recall para cada modelo, proporcionando así una medida agregada de su capacidad para detectar verdaderos positivos. Además, se utilizó una visualización de líneas para mostrar la tendencia del recall a lo largo de los diferentes reentrenamientos (semestrales, trimestrales y completo), lo que ayudó a identificar cualquier patrón o cambio en el rendimiento a lo largo del tiempo. Este enfoque sistemático permitió realizar una evaluación exhaustiva y comparativa de los modelos, facilitando así la identificación del mejor enfoque para el conjunto de datos y objetivos específicos.

```python
# Función para ver la matriz de confusión
def model_matrix(model, X_test, y_test, model_name):
    # Predecir sobre el conjunto de prueba
    y_pred_proba = model.predict(X_test)

    # Calcular el umbral óptimo basado en la curva ROC
    fpr, tpr, thresholds_roc = roc_curve(y_test, y_pred_proba)
    optimal_idx_roc = np.argmax(tpr - fpr)
    optimal_threshold_roc = thresholds_roc[optimal_idx_roc]

    # Aplicar el umbral óptimo basado en Precision-Recall (o puedes
usar el de ROC)
    y_pred = [1 if pred > optimal_threshold_roc else 0 for pred in
y_pred_proba]

    # Calcular y mostrar las métricas de evaluación
    print(f"Rendimiento del modelo {model_name}:")
    print("Recall:", recall_score(y_test, y_pred))

    # Calcular la matriz de confusión
    conf_matrix = confusion_matrix(y_test, y_pred)
    print(conf_matrix)
    print("")

    return recall_score(y_test, y_pred)

# Función para evaluar un modelo
def evaluate_model(model, X_test, y_test, model_name):
    y_pred_proba = model.predict(X_test)

    fpr, tpr, thresholds_roc = roc_curve(y_test, y_pred_proba)
    optimal_idx_roc = np.argmax(tpr - fpr)
    optimal_threshold_roc = thresholds_roc[optimal_idx_roc]

    y_pred = [1 if pred > optimal_threshold_roc else 0 for pred in
y_pred_proba]

    recall = recall_score(y_test, y_pred)
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

    return {
        "Model": model_name,
        "Recall": recall,
        "True Negatives": tn
    }

# Evaluar los modelos y almacenar los resultados
def evaluate_all_models(model_paths, X_test, y_test, method_name):
    results = []
    for model_path in model_paths:
```

```
        if model_path[0] == 'X':
            model = xgb.Booster()
            model.load_model(model_path)
        else:
            model = lgb.Booster(model_file=model_path)
        results.append(evaluate_model(model, X_test, y_test,
f"{method_name}_{model_path.split('/')[-1]}"))
    return results

# Función para calcular estadísticas agregadas
def calculate_aggregate_results(results_df):
    agg_results =
results_df.groupby(results_df['Model'].str.extract(r'(\w+)_')
[0]).agg({
        'Recall': ['mean'],
        'True Negatives': ['mean']
    }).reset_index()

    agg_results.columns = ['Method', 'Recall Mean', 'TN Mean']
    return agg_results

# Función para graficar la tendencia de recall para cada método de
reentrenamiento
def plot_recall_trend_by_method(results_semestral, results_trimestral,
results_completo):
    # Lista de resultados y títulos
    results_list = [
        (results_semestral, "Semestral"),
        (results_trimestral, "Trimestral"),
        (results_completo, "Completo")
    ]

    plt.figure(figsize=(12, 6))

    for results, title in results_list:
        # Convertir la lista de resultados en un DataFrame
        results_df = pd.DataFrame(results)

        # Crear la columna 'Reentreno' basada en la secuencia de
reentrenamientos
        results_df['Reentreno'] = range(1, len(results_df) + 1)

        # Graficar la tendencia de recall
        plt.plot(results_df['Reentreno'], results_df['Recall'],
marker='o', label=title)

    plt.xlabel('Reentreno')
    plt.ylabel('Recall')
    plt.title('Tendencia de Recall por Método de Reentrenamiento')
    plt.legend()
```

```
    plt.grid(True)
    plt.xticks(range(1, max(len(results_semestral),
len(results_trimestral), len(results_completo)) + 1))  # Asegurarse de
que el eje X sea entero
    plt.show()
```

LGBM

```
# Definir los modelos a evaluar
lgb_model_paths_semestral = [
    'LGBM Models/lgb_model_2019.txt',
    'LGBM Models/lgb_model_2020_S1.txt',
    'LGBM Models/lgb_model_2020_S2.txt'
]

lgb_model_paths_trimestral = [
    'LGBM Models/lgb_model_2019.txt',
    'LGBM Models/lgb_model_2020_T1.txt',
    'LGBM Models/lgb_model_2020_T2.txt',
    'LGBM Models/lgb_model_2020_T3.txt',
    'LGBM Models/lgb_model_2020_T4.txt'
]

lgb_model_paths_completo = [
    'LGBM Models/lgb_model_2019.txt',
    'LGBM Models/lgb_model_retrained.txt'
]

# Matriz de confusión para los modelos semestrales
for model_path in lgb_model_paths_semestral:
    model = lgb.Booster(model_file=model_path)
    model_matrix(model, X_test_all, y_test_all,
f'LightGBM_{model_path.split("/")[-1]}')
```

```
Rendimiento del modelo LightGBM_lgb_model_2019.txt:
Recall: 0.8903924221921515
[[247917  28465]
 [   162   1316]]

Rendimiento del modelo LightGBM_lgb_model_2020_S1.txt:
Recall: 0.8910690121786198
[[248904  27478]
 [   161   1317]]

Rendimiento del modelo LightGBM_lgb_model_2020_S2.txt:
Recall: 0.8525033829499323
[[238643  37739]
 [   218   1260]]
```

```python
# Matriz de confusión para los modelos trimestrales
for model_path in lgb_model_paths_trimestral:
    model = lgb.Booster(model_file=model_path)
    model_matrix(model, X_test_all, y_test_all,
f'LightGBM_{model_path.split("/")[-1]}')
```

```
Rendimiento del modelo LightGBM_lgb_model_2019.txt:
Recall: 0.8903924221921515
[[247917  28465]
 [   162   1316]]

Rendimiento del modelo LightGBM_lgb_model_2020_T1.txt:
Recall: 0.8775372124492558
[[246018  30364]
 [   181   1297]]

Rendimiento del modelo LightGBM_lgb_model_2020_T2.txt:
Recall: 0.8741542625169147
[[214426  61956]
 [   186   1292]]

Rendimiento del modelo LightGBM_lgb_model_2020_T3.txt:
Recall: 0.8396481732070366
[[211940  64442]
 [   237   1241]]

Rendimiento del modelo LightGBM_lgb_model_2020_T4.txt:
Recall: 0.696211096075778
[[225705  50677]
 [   449   1029]]
```

```python
# Matriz de confusión para los modelos completos
for model_path in lgb_model_paths_completo:
    model = lgb.Booster(model_file=model_path)
    model_matrix(model, X_test_all, y_test_all,
f'LightGBM_{model_path.split("/")[-1]}')
```

```
Rendimiento del modelo LightGBM_lgb_model_2019.txt:
Recall: 0.8903924221921515
[[247917  28465]
 [   162   1316]]

Rendimiento del modelo LightGBM_lgb_model_retrained.txt:
Recall: 0.9817320703653586
[[270945   5437]
 [    27   1451]]
```

```python
# Evaluar los modelos con el conjunto de prueba
lgb_results_semestral = evaluate_all_models(lgb_model_paths_semestral,
```

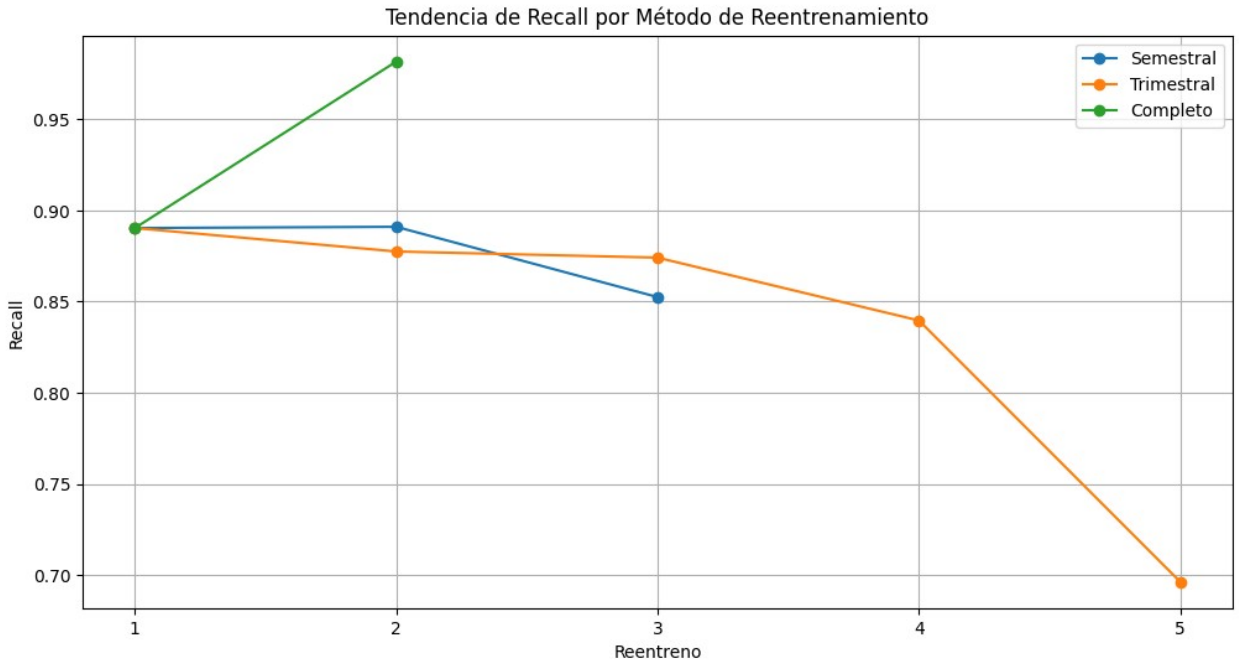
```
X_test_all, y_test_all, "Semestral")
lgb_results_trimestral =
evaluate_all_models(lgb_model_paths_trimestral, X_test_all,
y_test_all, "Trimestral")
lgb_results_completo = evaluate_all_models(lgb_model_paths_completo,
X_test_all, y_test_all, "Completo")

# Combinar todos los resultados
all_lgb_results = lgb_results_semestral + lgb_results_trimestral +
lgb_results_completo

# Convertir los resultados a un DataFrame
lgb_results_df = pd.DataFrame(all_lgb_results)
lgb_results_df
```

	Model	Recall	True Negatives
0	Semestral_lgb_model_2019.txt	0.890392	247917
1	Semestral_lgb_model_2020_S1.txt	0.891069	248904
2	Semestral_lgb_model_2020_S2.txt	0.852503	238643
3	Trimestral_lgb_model_2019.txt	0.890392	247917
4	Trimestral_lgb_model_2020_T1.txt	0.877537	246018
5	Trimestral_lgb_model_2020_T2.txt	0.874154	214426
6	Trimestral_lgb_model_2020_T3.txt	0.839648	211940
7	Trimestral_lgb_model_2020_T4.txt	0.696211	225705
8	Completo_lgb_model_2019.txt	0.890392	247917
9	Completo_lgb_model_retrained.txt	0.981732	270945

```
# Cambiar nombres de modelos para agrupación correcta
lgb_results_df['Model'] =
lgb_results_df['Model'].str.replace('Semestral_lgb_model_2019.txt',
'Semestral_lgb_model_2020_S0.txt')
lgb_results_df['Model'] =
lgb_results_df['Model'].str.replace('Trimestral_lgb_model_2019.txt',
'Trimestral_lgb_model_2020_T0.txt')
lgb_results_df['Model'] =
lgb_results_df['Model'].str.replace('Completo_lgb_model_2019.txt',
'Completo_lgb_model_2019.txt')

# Calcular estadísticas agregadas
lgb_agg_results = calculate_aggregate_results(lgb_results_df)
lgb_agg_results
```

	Method	Recall Mean	TN Mean
0	Completo_lgb_model	0.936062	259431.000000
1	Semestral_lgb_model_2020	0.877988	245154.666667
2	Trimestral_lgb_model_2020	0.835589	229201.200000

```
# Llamar a la función con los resultados
plot_recall_trend_by_method(lgb_results_semestral,
lgb_results_trimestral, lgb_results_completo)
```



XGBoost

```
# Definir los modelos a evaluar
xgb_model_paths_semestral = [
    'XGB Models/xgb_model_2019.model',
    'XGB Models/xgb_model_2020_S1.model',
    'XGB Models/xgb_model_2020_S2.model'
]

xgb_model_paths_trimestral = [
    'XGB Models/xgb_model_2019.model',
    'XGB Models/xgb_model_2020_T1.model',
    'XGB Models/xgb_model_2020_T2.model',
    'XGB Models/xgb_model_2020_T3.model',
    'XGB Models/xgb_model_2020_T4.model'
]

xgb_model_paths_completo = [
    'XGB Models/xgb_model_2019.model',
    'XGB Models/xgb_model_retrained.model'
]

# Matriz de confusión para los modelos semestrales
for model_path in xgb_model_paths_semestral:
    model = xgb.Booster()
    model.load_model(model_path)
    model_matrix(model, dtest_all, y_test_all, model_path.split('/')[-
1])
```

```
Rendimiento del modelo xgb_model_2019.model:
Recall: 0.8470906630581867
[[238576  37806]
 [   226   1252]]

Rendimiento del modelo xgb_model_2020_S1.model:
Recall: 0.803788903924222
[[221801  54581]
 [   290   1188]]

Rendimiento del modelo xgb_model_2020_S2.model:
Recall: 0.8795669824086604
[[252139  24243]
 [   178   1300]]


# Matriz de confusión para los modelos trimestrales
for model_path in xgb_model_paths_trimestral:
    model = xgb.Booster()
    model.load_model(model_path)
    model_matrix(model, dtest_all, y_test_all, model_path.split('/')[-
1])

Rendimiento del modelo xgb_model_2019.model:
Recall: 0.8470906630581867
[[238576  37806]
 [   226   1252]]

Rendimiento del modelo xgb_model_2020_T1.model:
Recall: 0.8335588633288228
[[209230  67152]
 [   246   1232]]

Rendimiento del modelo xgb_model_2020_T2.model:
Recall: 0.7341001353179973
[[249153  27229]
 [   393   1085]]

Rendimiento del modelo xgb_model_2020_T3.model:
Recall: 0.7814614343707713
[[249838  26544]
 [   323   1155]]

Rendimiento del modelo xgb_model_2020_T4.model:
Recall: 0.7625169147496617
[[257530  18852]
 [   351   1127]]


# Matriz de confusión para los modelos completos
for model_path in xgb_model_paths_completo:
```

```
    model = xgb.Booster()
    model.load_model(model_path)
    model_matrix(model, dtest_all, y_test_all, model_path.split('/')[-
1])

Rendimiento del modelo xgb_model_2019.model:
Recall: 0.8470906630581867
[[238576  37806]
 [   226   1252]]

Rendimiento del modelo xgb_model_retrained.model:
Recall: 0.8470906630581867
[[238576  37806]
 [   226   1252]]


# Evaluar los modelos con el conjunto de prueba
xgb_results_semestral = evaluate_all_models(xgb_model_paths_semestral,
dtest_all, y_test_all, "Semestral")
xgb_results_trimestral =
evaluate_all_models(xgb_model_paths_trimestral, dtest_all, y_test_all,
"Trimestral")
xgb_results_completo = evaluate_all_models(xgb_model_paths_completo,
dtest_all, y_test_all, "Completo")

# Combinar todos los resultados
all_xgb_results = xgb_results_semestral + xgb_results_trimestral +
xgb_results_completo

# Convertir los resultados a un DataFrame
xgb_results_df = pd.DataFrame(all_xgb_results)
xgb_results_df
```

	Model	Recall	True Negatives
0	Semestral_xgb_model_2019.model	0.847091	238576
1	Semestral_xgb_model_2020_S1.model	0.803789	221801
2	Semestral_xgb_model_2020_S2.model	0.879567	252139
3	Trimestral_xgb_model_2019.model	0.847091	238576
4	Trimestral_xgb_model_2020_T1.model	0.833559	209230
5	Trimestral_xgb_model_2020_T2.model	0.734100	249153
6	Trimestral_xgb_model_2020_T3.model	0.781461	249838
7	Trimestral_xgb_model_2020_T4.model	0.762517	257530
8	Completo_xgb_model_2019.model	0.847091	238576
9	Completo_xgb_model_retrained.model	0.847091	238576

```
# Cambiar nombres de modelos para agrupación correcta
xgb_results_df['Model'] =
xgb_results_df['Model'].str.replace('Semestral_xgb_model_2019.model',
'Semestral_xgb_model_2020_S0.model')
xgb_results_df['Model'] =
xgb_results_df['Model'].str.replace('Trimestral_xgb_model_2019.model',
```

```
'Trimestral_xgb_model_2020_T0.model')
xgb_results_df['Model'] =
xgb_results_df['Model'].str.replace('Completo_xgb_model_2019.model',
'Completo_xgb_model_2019.model')

# Calcular estadísticas agregadas
xgb_agg_results = calculate_aggregate_results(xgb_results_df)
xgb_agg_results
```

	Method	Recall Mean	TN Mean
0	Completo_xgb_model	0.847091	238576.000000
1	Semestral_xgb_model_2020	0.843482	237505.333333
2	Trimestral_xgb_model_2020	0.791746	240865.400000

```
# Llamar a la función con los resultados
plot_recall_trend_by_method(xgb_results_semestral,
xgb_results_trimestral, xgb_results_completo)
```