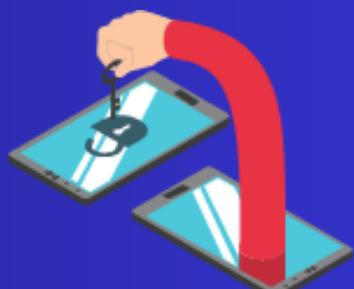


FRAUDE FINANCIERO



Análisis de Transacciones

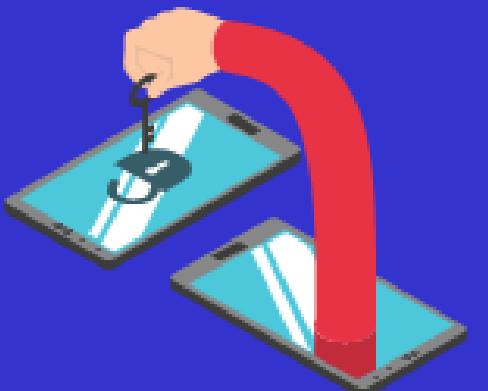


INTRODUCCIÓN

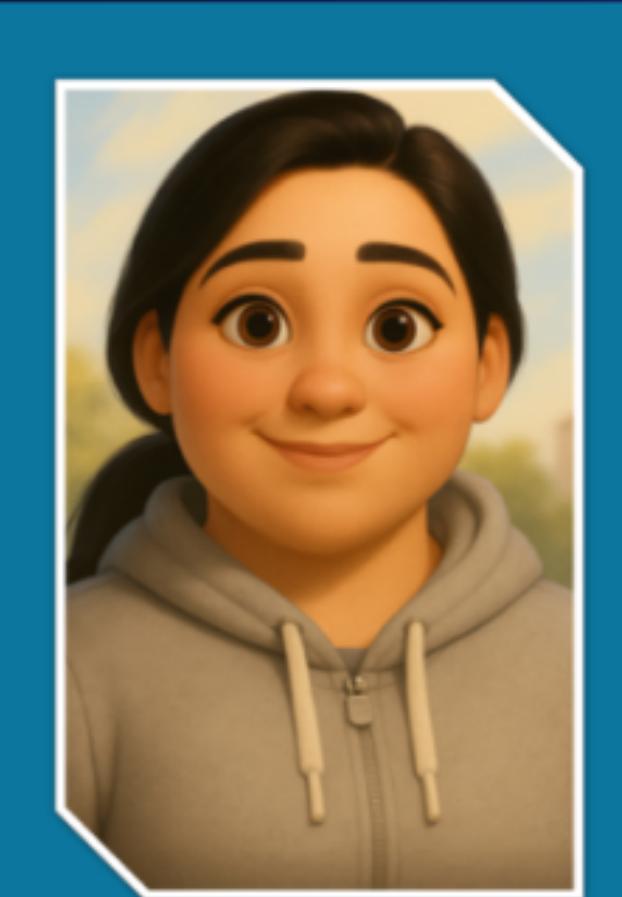


Los delitos asociados a transacciones bancarias fraudulentas no solo generan pérdidas económicas considerables para las entidades financieras y los usuarios, sino que también representan un desafío constante para la seguridad digital.

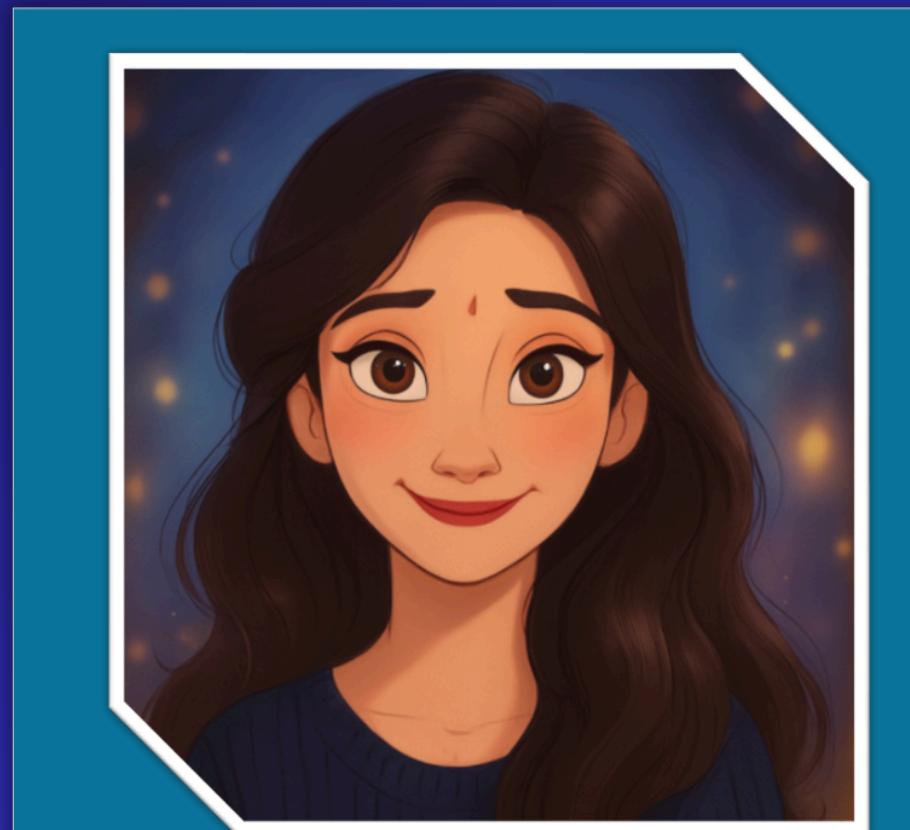
A través de técnicas de análisis exploratorio y modelos de detección de anomalías, buscamos comprender los patrones detrás de estas operaciones y proponer soluciones que permitan identificar y prevenir fraudes de manera eficiente.



Nuestro Equipo Data Science



Vanesa Carpio
Ing Sistemas



Anne Kathrin Huber
Ing Ambiental

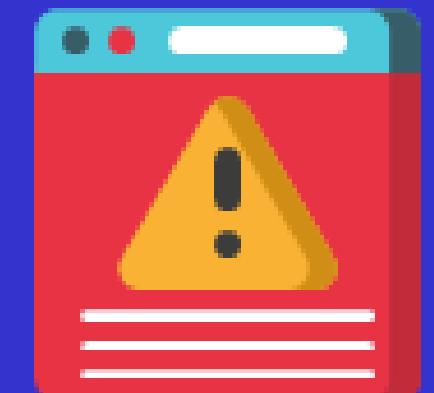


Maria Florencia Siarri
Ing Industrial

INDICE



- **Dataset**
- **Preprocesamiento y limpieza**
- **Selección y evaluación de modelo**
- **Preparación de datos para Machine Learning**
- **Principales hallazgos**
- **Conclusión**





OBJETIVO PRINCIPAL

Analizar y detectar patrones de fraude en transacciones financieras a través del análisis exploratorio de datos.



IMPORTANCIA

El fraude financiero genera pérdidas millonarias anualmente en el sector. La detección temprana es crucial para mitigar riesgos.



METODOLOGÍA

Análisis exploratorio para identificar patrones, correlaciones y anomalías en las transacciones que puedan indicar actividad fraudulenta.

¿Por qué usar modelos de datos ?



Protección de información sensible y privacidad de clientes reales



Generación de escenarios controlados de fraude



Facilita compartir datos para investigación sin riesgos legales



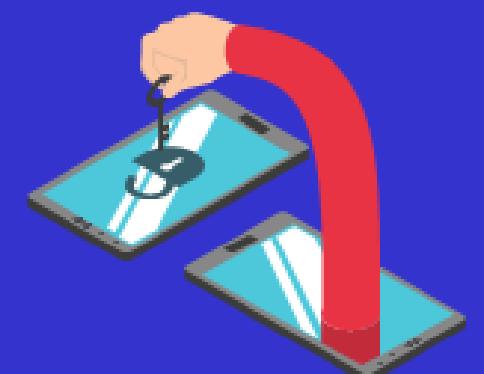
Mantiene características estadísticas de los datos reales



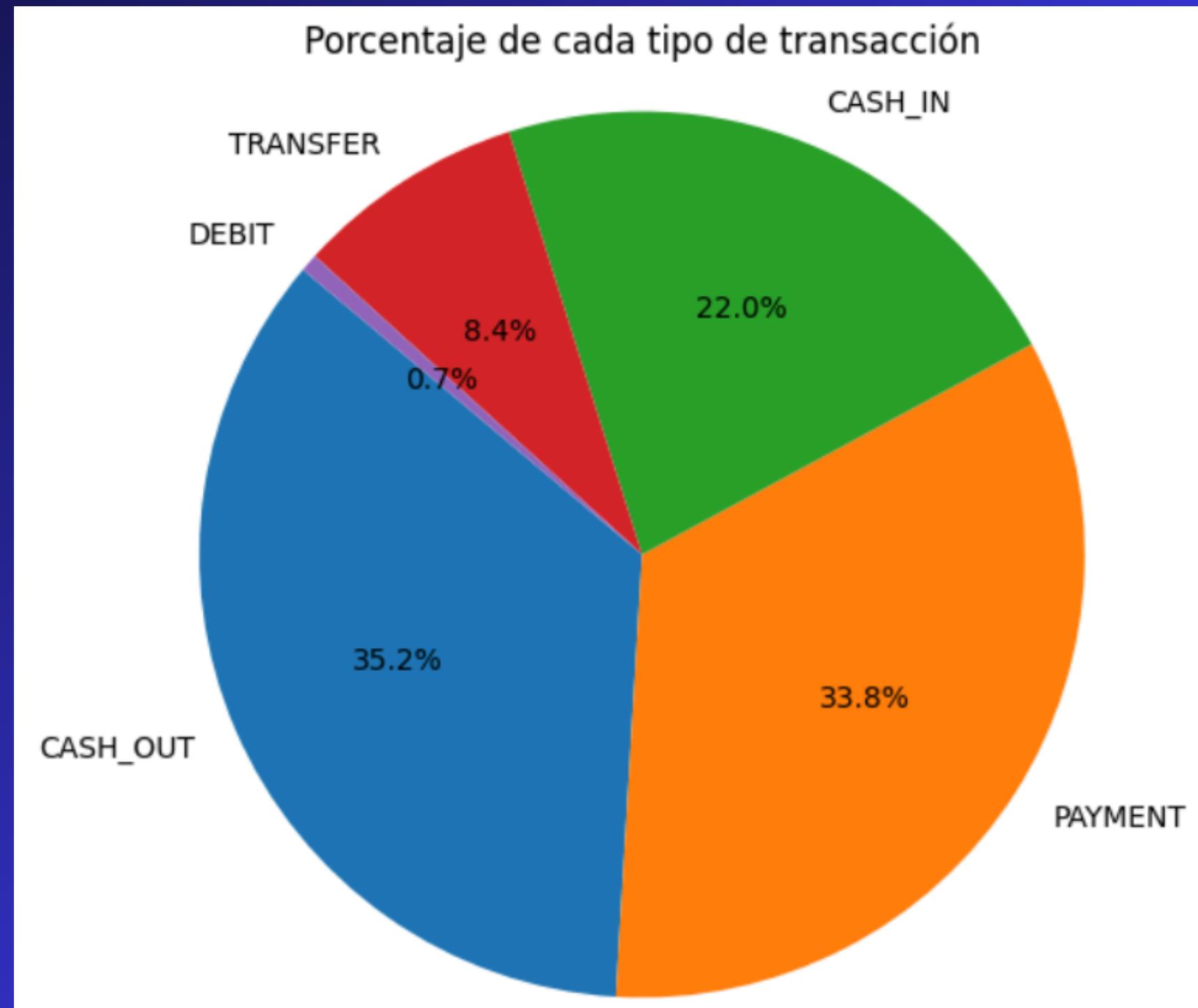
Explicación de nuestras base de datos

Columnas Principales	Columnas de Balance y Fraude
step Unidad de tiempo (1 step = 1 hora). Total: 744 steps (31 días).	oldbalanceOrg / newbalanceOrig Saldo inicial y nuevo del cliente origen.
type Tipo de transacción: CASH-IN, CASH-OUT, DEBIT, PAYMENT, TRANSFER.	oldbalanceDest / newbalanceDest Saldo inicial y nuevo del cliente destino. Sin info para Comerciantes (M).
amount Monto de la transacción en moneda local.	isFraud Transacciones fraudulentas (1) donde se intenta vaciar fondos de cuentas.
nameOrig / nameDest Identificadores de cliente origen y destino. Prefijos C: Cliente, M: Comerciante.	isFlaggedFraud Sistema de detección para transferencias mayores a 200,000

QR dataset:



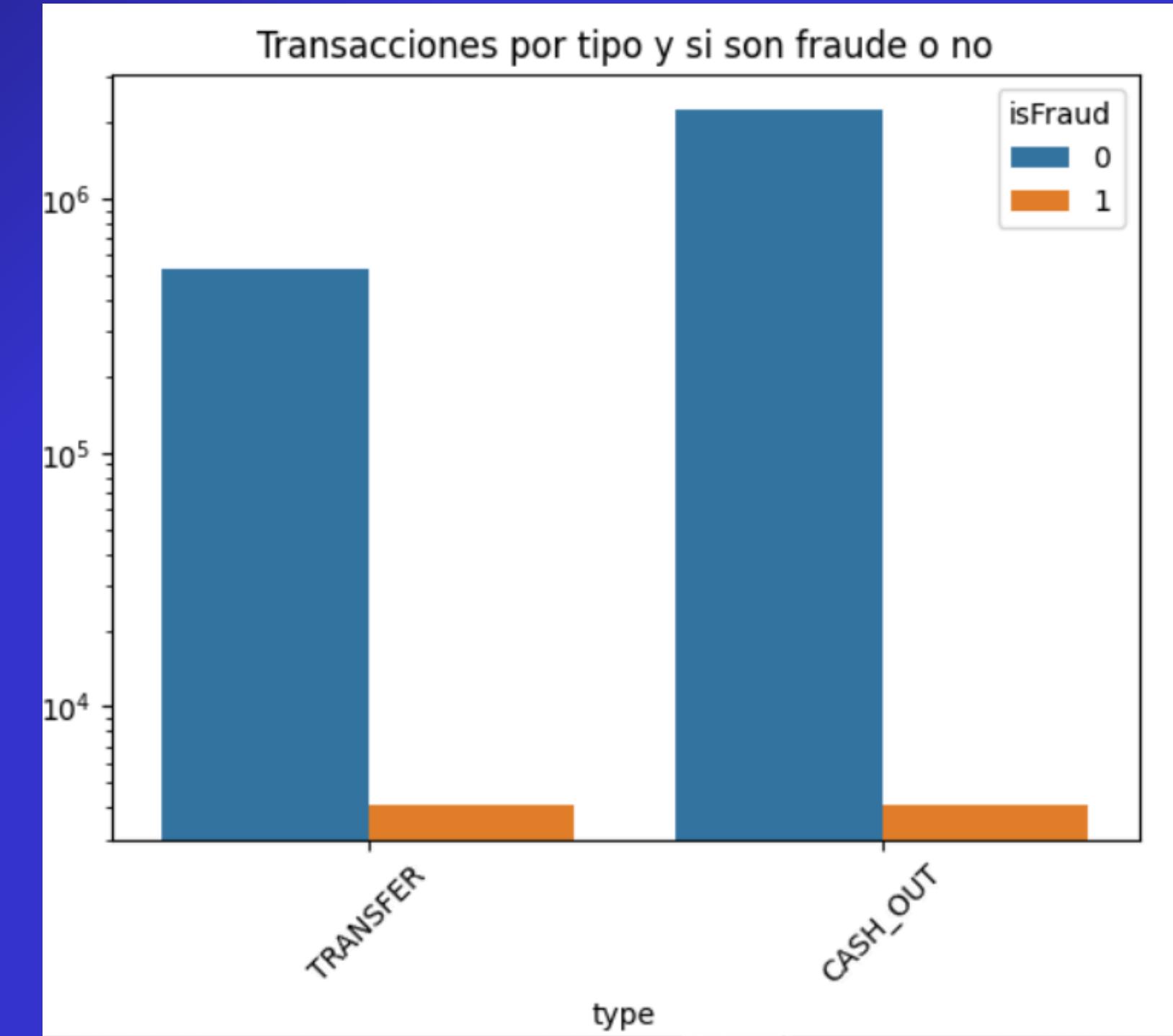
Distribución porcentual de las transacciones



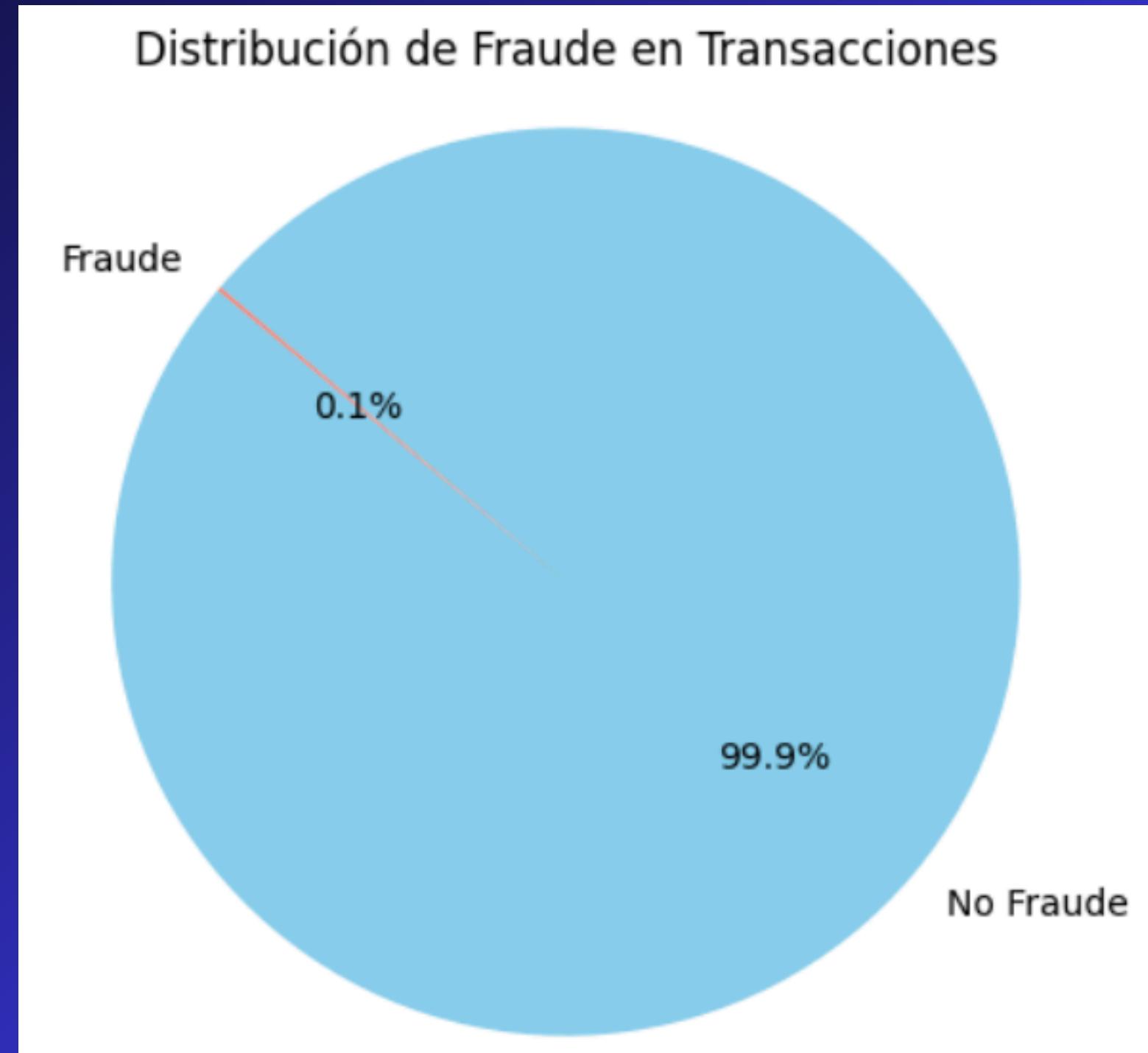
Tipo de transacciones con/sin fraude

Siempre tenemos que tener presente que ningún valor de balance es confiable para predecir el fraude, ya que fue alterado por el simulador después de la detección del fraude

Detectamos que hay un desbalance considerable entre lo que esta etiquetado como fraude y no fraude.

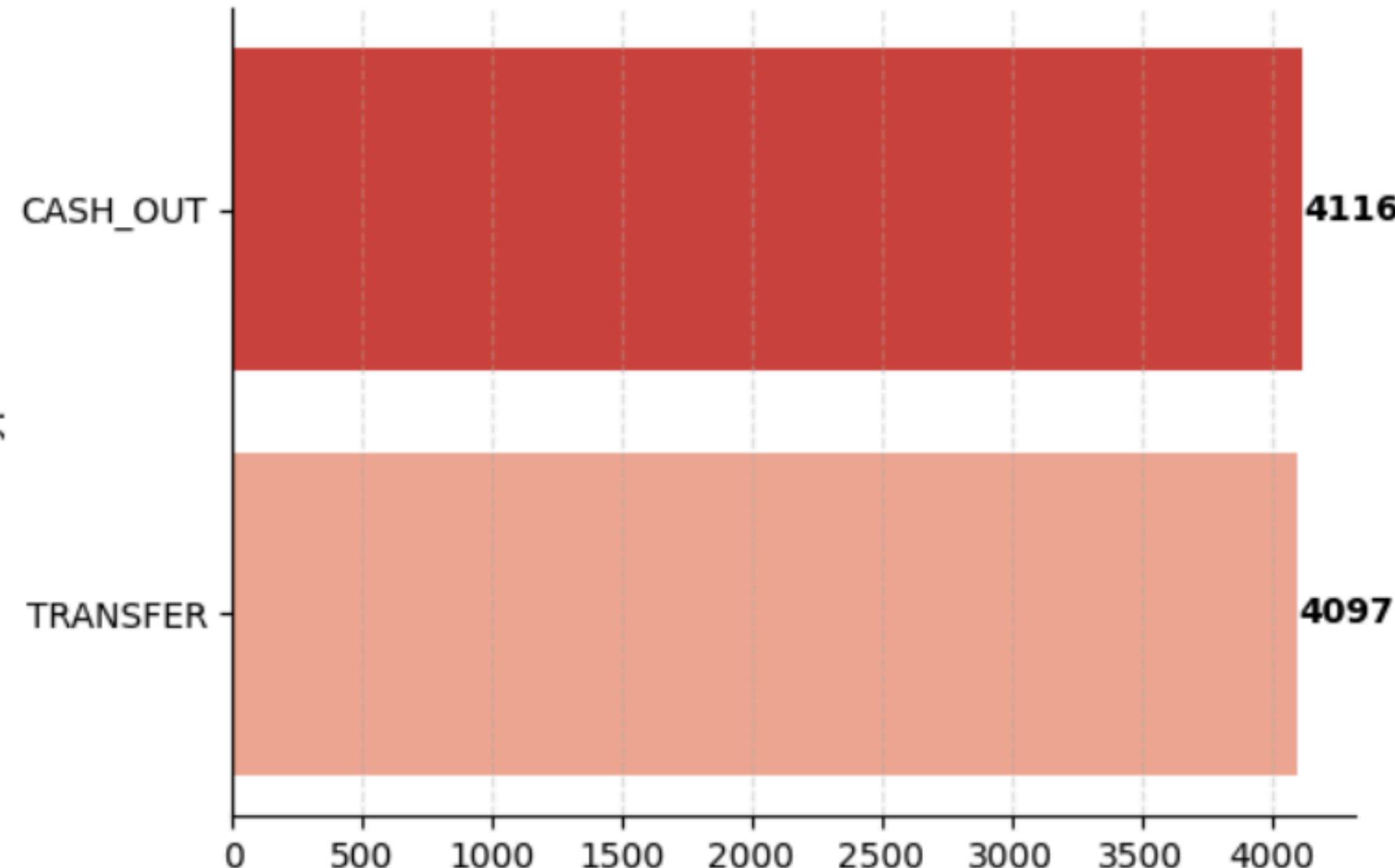


Distribución de los features desbalanceados

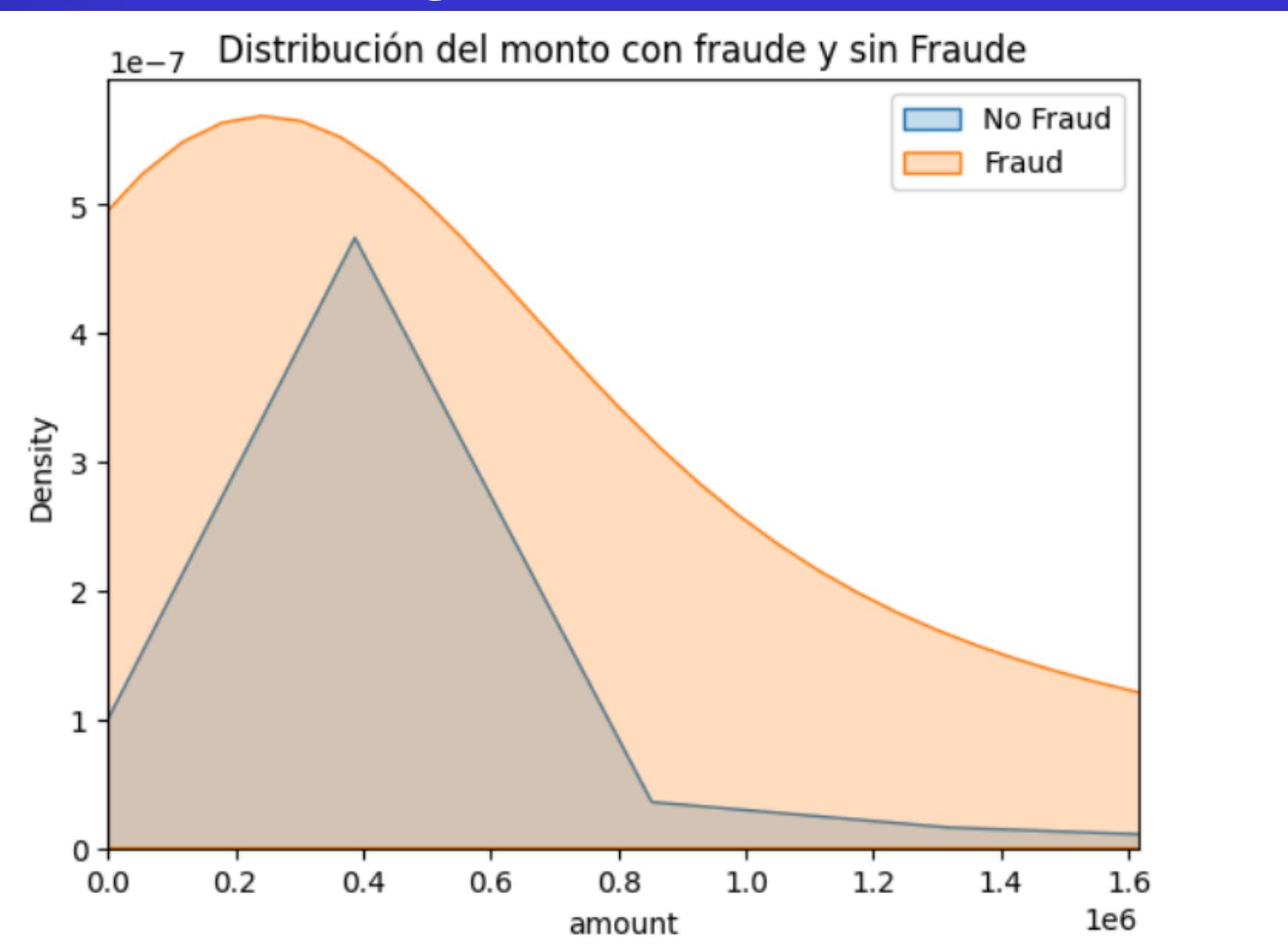


Categorías que alojan la mayor cantidad de Fraudes

Cantidad de fraudes por tipo de transacción



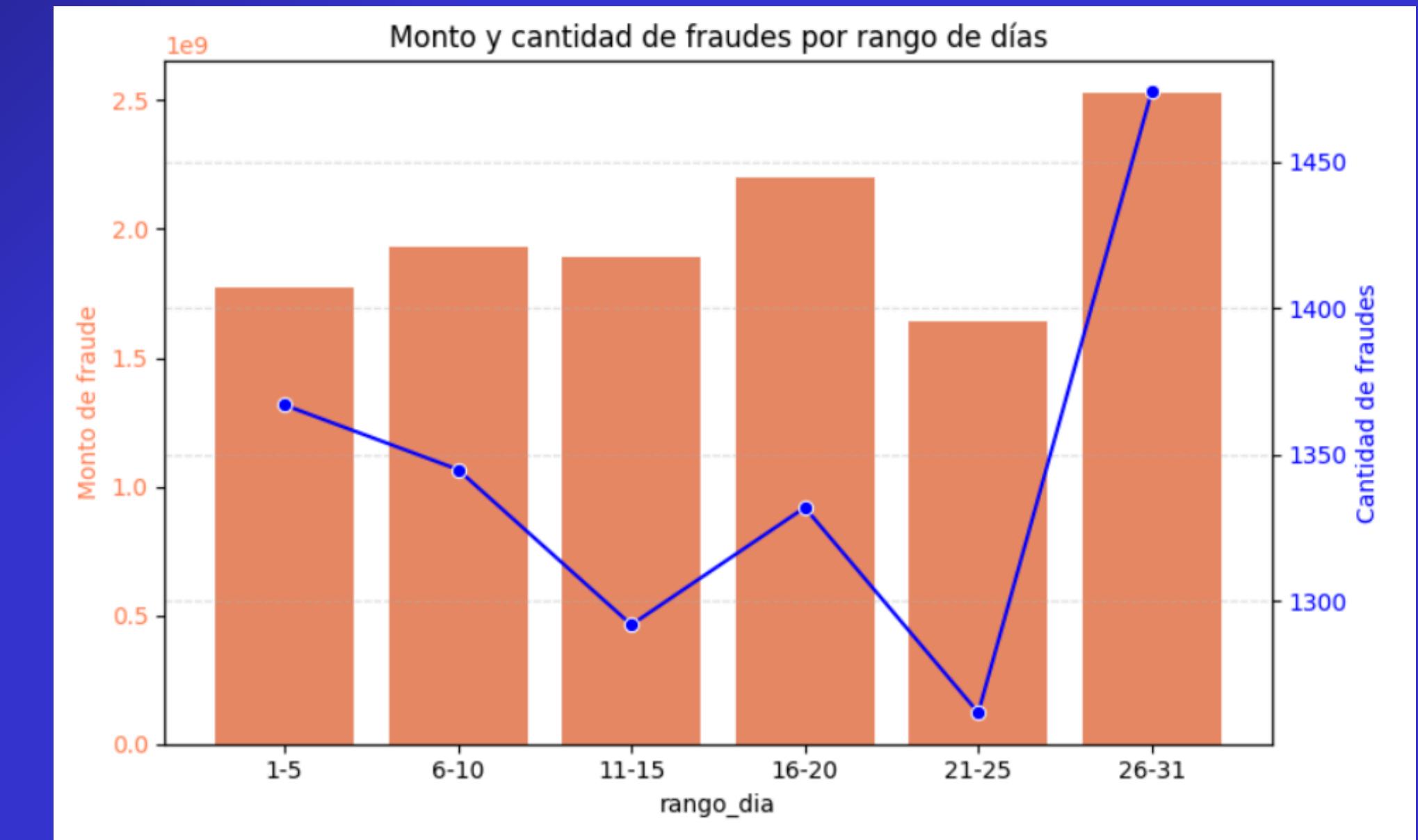
Montos donde observamos cuando se originan los Fraudes



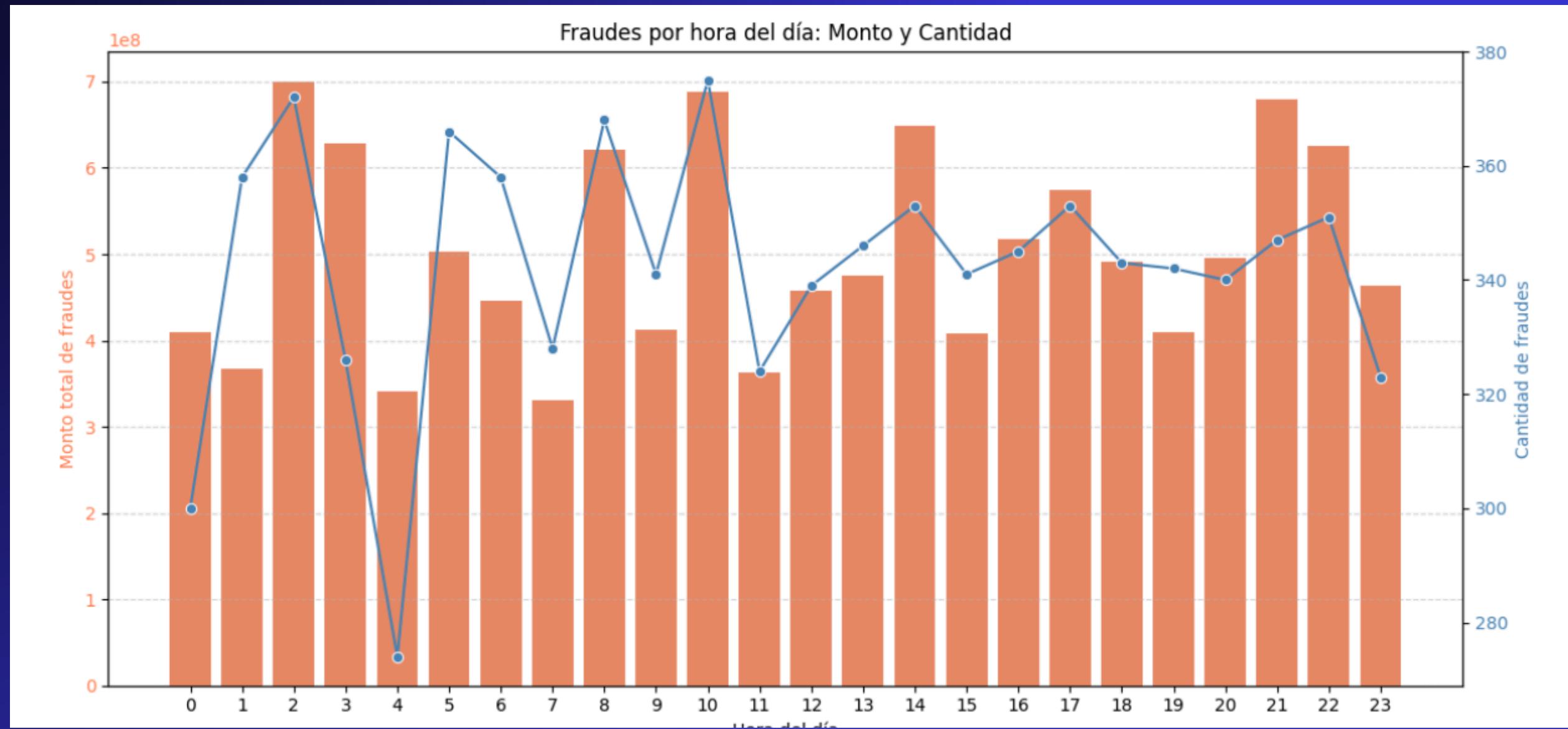
Cantidad de Fraudes y montos en función del tiempo

Utilizamos Groupby ya que se puede aplicar para Dataset con grandes volúmenes de datos para obtener una versión condensada y significativa, permitiéndonos combinar cantidad de montos y fraude por día. Así obtenemos nuevas variables (features) que ayudan al modelo a detectar patrones fraudulentos.

Montos
Cantidades



Realizamos un análisis por variable y horas del mes.



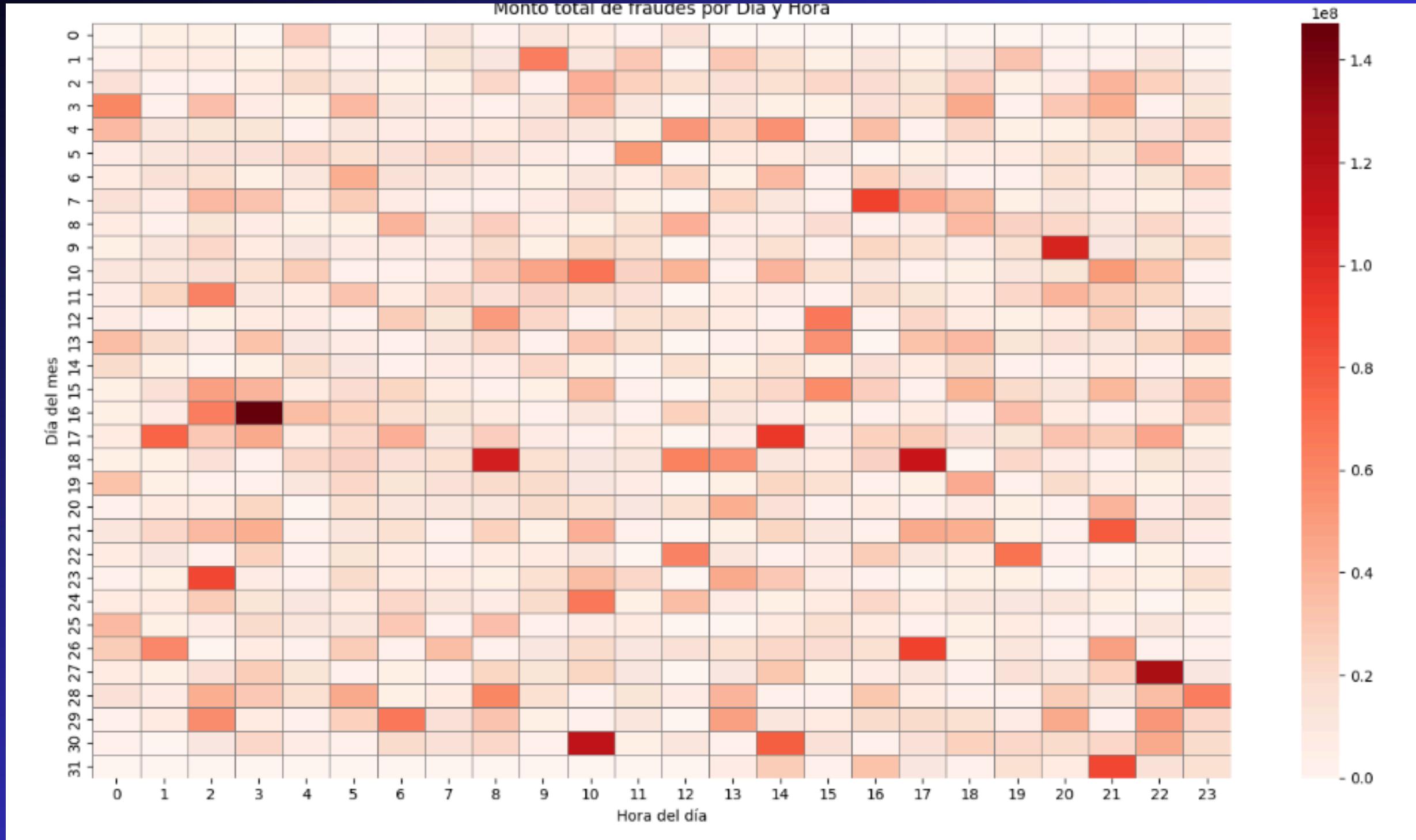
⌚ Horas críticas por frecuencia

La cantidad de fraudes se distribuye a lo largo del día con mayor concentración en los rangos de **1-2h, 5-10h y 13-22h**, siendo más frecuentes durante horario laboral.

💰 Horas críticas por monto

Las horas con mayor monto de fraude son las **2, 3, 8, 10, 14, 21 y 22**, con picos especialmente significativos durante la madrugada (2-3h) y la noche (21-22h).

HeadMaps Fraude Día y Hora



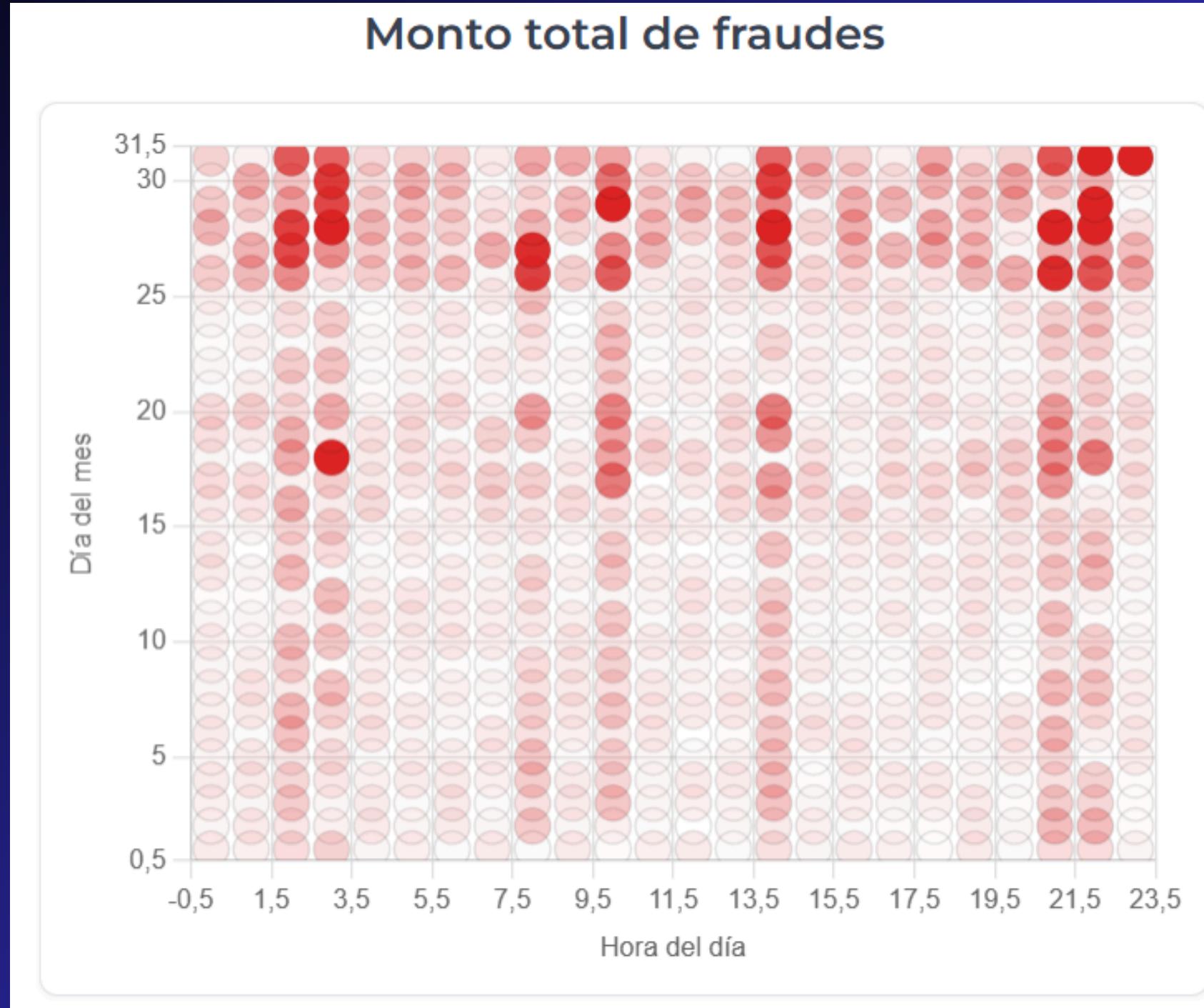
Patrones identificados

- Mayor cantidad de fraude entre los días 16-20 del mes.
- Concentración de fraude en las horas 2,3,8,10,14,21,22

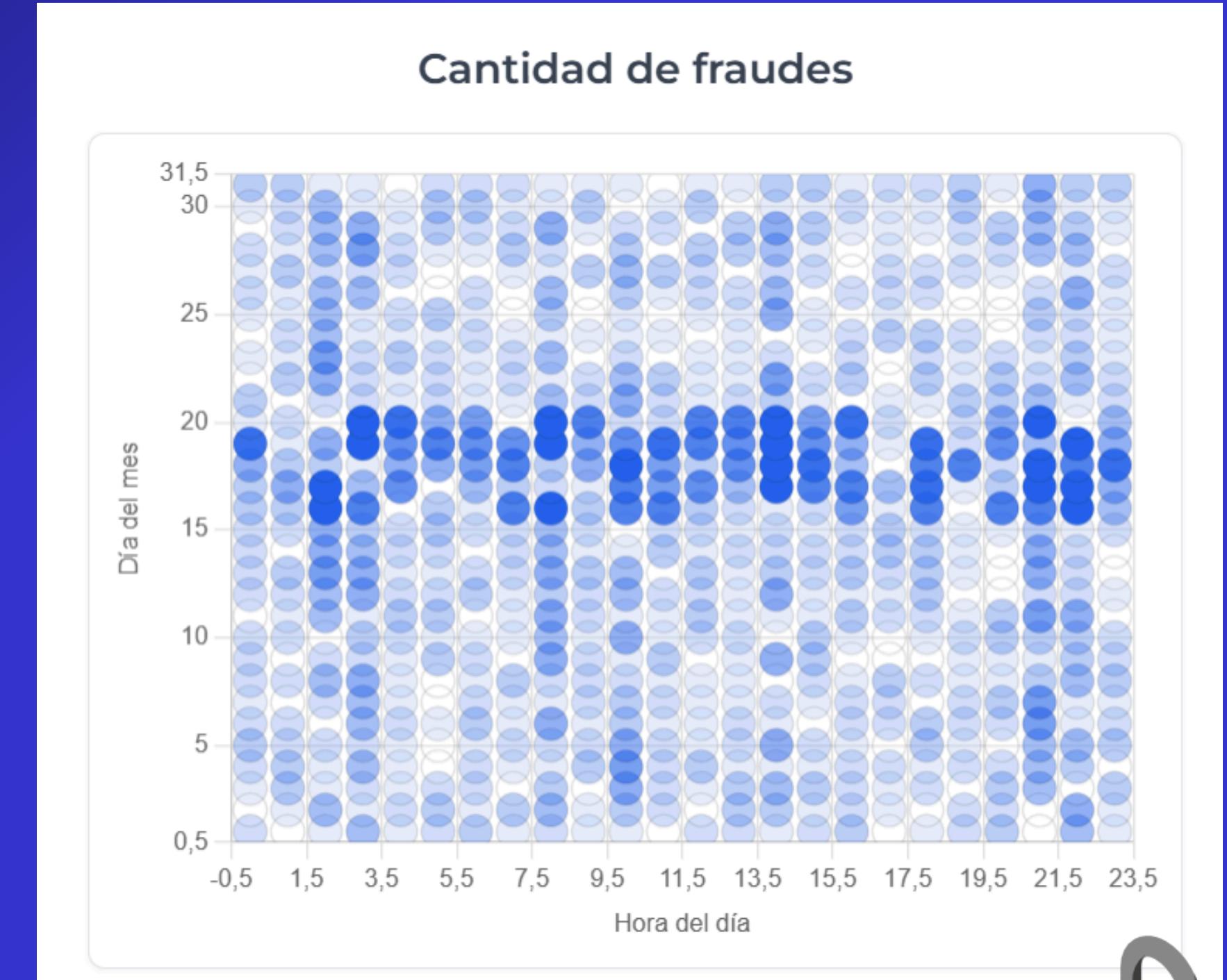


HeadMaps Fraude Día y Hora

Monto total de fraudes



Cantidad de fraudes



¿Porque utilizamos OneHotEncoder?

En los proyecto de detección de fraude, el uso de OneHotEncoder es esencial para transformar datos categóricos en un formato que los modelos de machine learning puedan interpretar y procesar correctamente.

Como mencionamos anteriormente el código OneHotEncoder convierte variables categóricas en una representación numérica binaria, creando una columna por cada categoría y asignando:

- 1 si la fila pertenece a esa categoría
- 0 si no



Preparación de datos para Machine Learning



Eliminación de columnas no útiles

Eliminamos las columnas de balance (oldbalanceOrg, newbalanceOrig, oldbalanceDest, newbalanceDest) ya que no son confiables para la detección de fraudes. En transacciones fraudulentas, estos valores fueron modificados por el simulador.



Filtrado por tipos relevantes

Nos enfocamos en los únicos tipos de transacciones donde se detectaron fraudes: TRANSFER y CASH_OUT. Esto reduce el dataset de 6.3M a 2.7M registros, manteniendo el 100% de los casos de fraude.



Transformación de variables categóricas

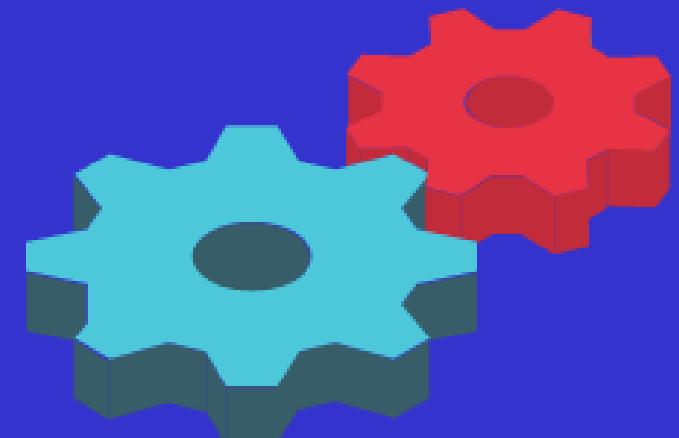
Aplicamos OneHotEncoding a la variable categórica 'type', generando las columnas 'type_CASH_OUT' y 'type_TRANSFER'. También eliminamos columnas de identificación de usuarios que no aportan valor predictivo.



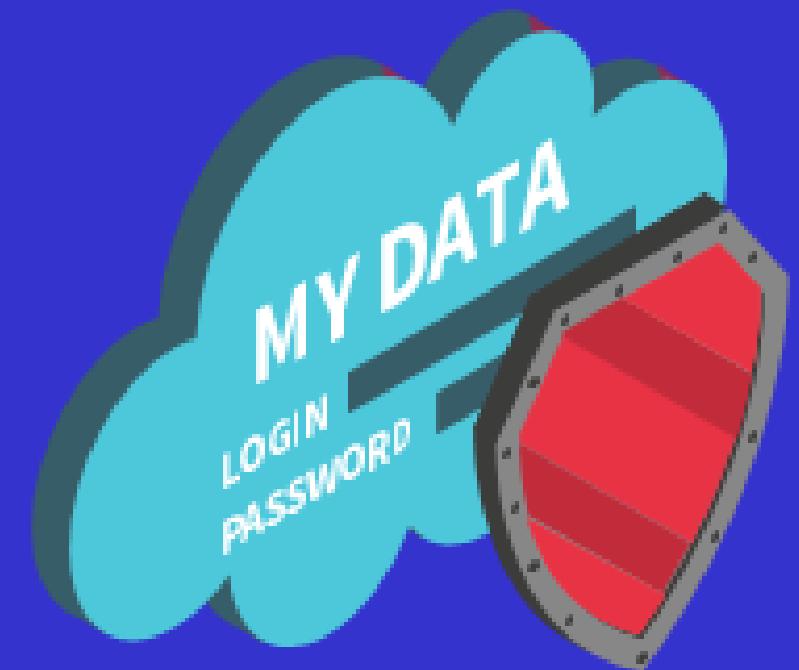
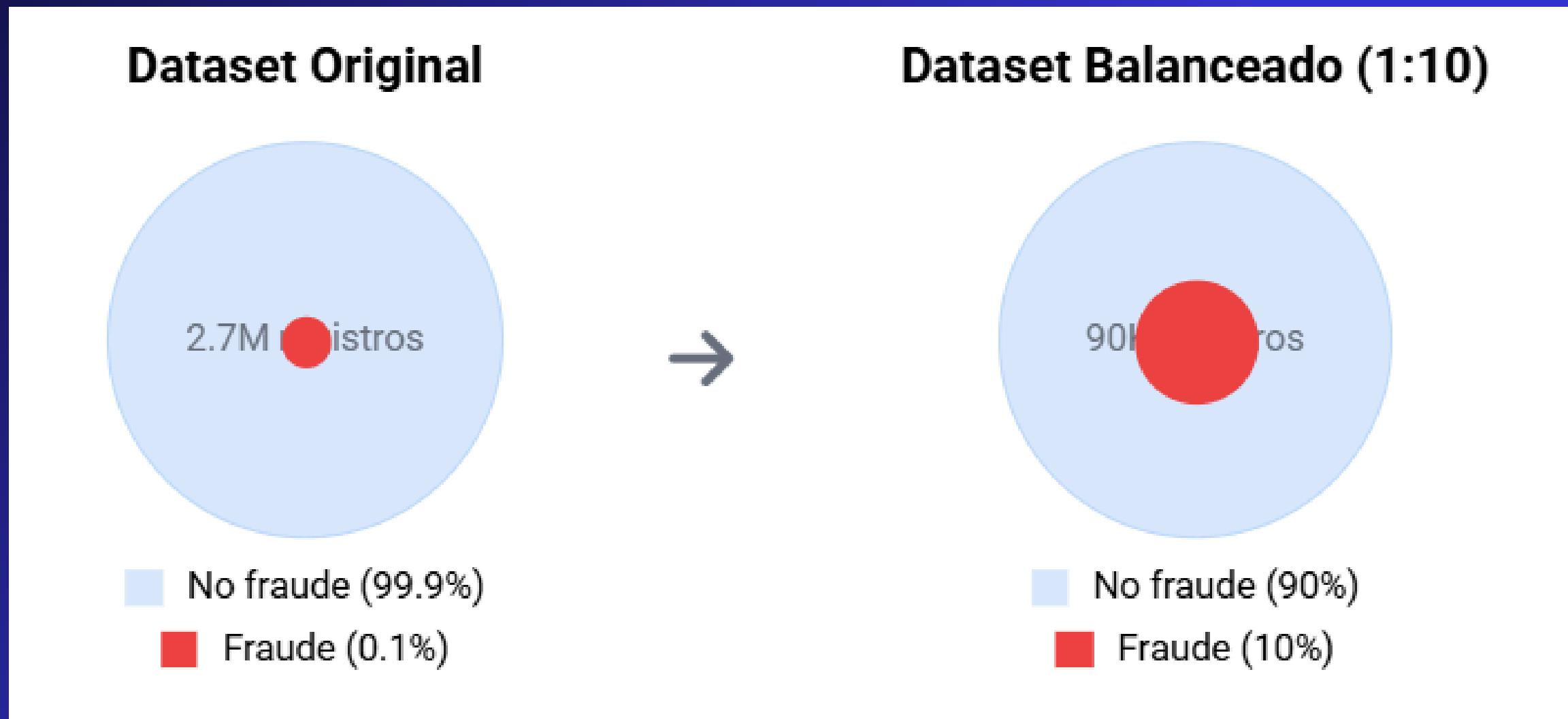
Dataset transformado

Estructura del Dataset (df_final)

Variable	Descripción
amount	Monto de la transacción
isFraud	Indicador de fraude (1) o no fraude (0)
type_CASH_OUT	Indicador de retiro de efectivo
type_DEBIT	Indicador de cargo a cuenta
type_PAYMENT	Indicador de pago realizado
type_TRANSFER	Indicador de transferencia
isFlaggedFraud	Transacción marcada como sospechosa



Dataset Balanceado vs Dataset Original



Como balanceamos un Dataset

- █ **Submuestreo:** RandomUnderSampler , NearMiss
- █ **Sobremuestreo:** SMOTE , ADASYN , RandomOverSampler

	Submuestro (Undersampling)	Sobremuestre (Oversampling)
¿Qué hace?	Elimina ejemplos de la clase mayoritaria	Duplica o genera ejemplo de la clase minoritaria
Objetivo	Reduce el dominio de la clase más común	Aumenta la representación de la clase menos común
Tamaño del Dataset	Redice el tamaño del conjunto	Aumenta el tamaño del conjunto
Costo Computacional	Bajo (menos datos)	Mayor (más datos procesados)
Riesgo	Perder información útil	Overfitting si se duplican datos
¿Modelos recomendados?	Rápidos o con datasets muy grandes	Modelos más complejos o datasets pequeños
Aplicaciones comunes	BigData,tiempo real	Problemas medios,fraude.



Estrategias de Balanceo: Submuestreo



¿Qué es el submuestreo?

Técnica que reduce la cantidad de instancias de la clase mayoritaria para equilibrar el dataset, manteniendo todas las instancias de la clase minoritaria



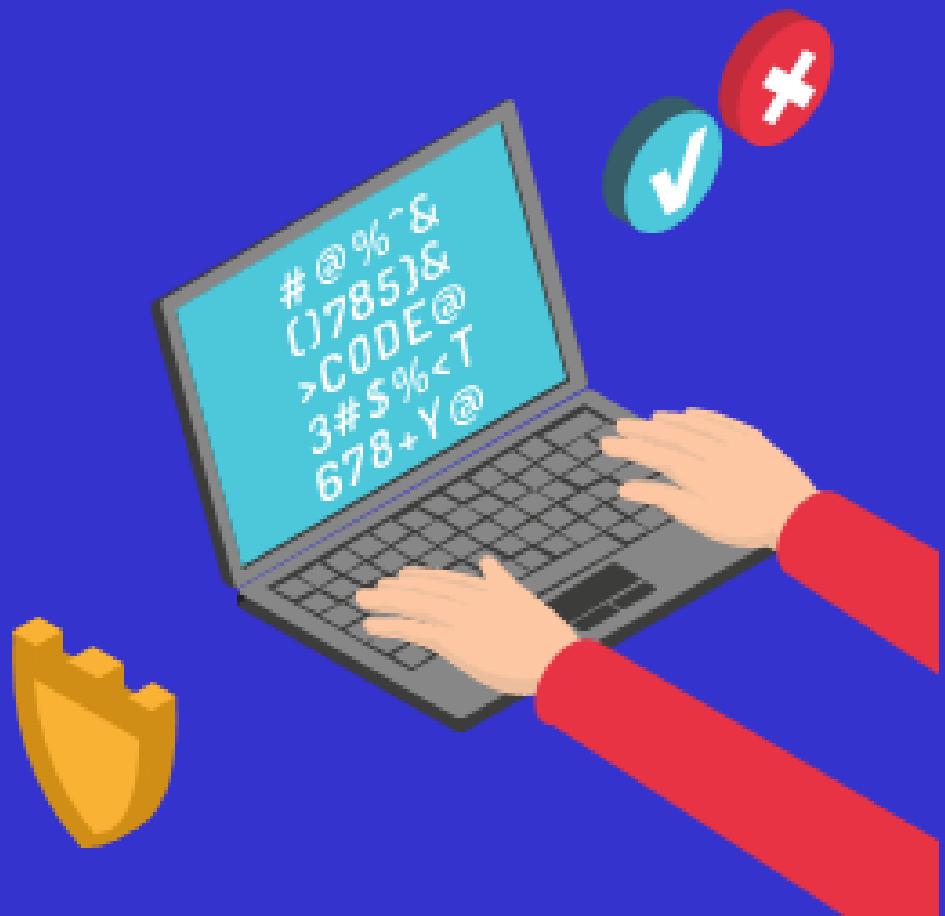
Fortalezas

- Reduce el costo computacional del entrenamiento
- Evita el sobreajuste hacia la clase mayoritaria
- Método simple y directo de implementar
- Mejora el recall de la clase minoritaria (fraudes)



Limitaciones

- Descarta información potencialmente valiosa
- Puede reducir la precisión general del modelo
- Riesgo de perder patrones importantes de la clase mayoritaria
- El modelo puede no generalizar bien a datos nuevos



Submuestreo

```
# Vamos a usar una relación de 1:10 --> 1 fraude == 10 No Fraude
from sklearn.utils import resample

# Dividimos
df_fraude = df_final[df_final.isFraud == 1]
df_no_fraude = df_final[df_final.isFraud == 0]

# Submuestreo
df_no_fraude_bal = resample(df_no_fraude,
                             replace=False,
                             n_samples=len(df_fraude) * 10, # relación 1:10
                             random_state=42)

# Dataset balanceado
df_balanceado = pd.concat([df_fraude, df_no_fraude_bal])
```

Estrategias de Balanceo: Sobre muestreo (SMOTE y ADASYN)



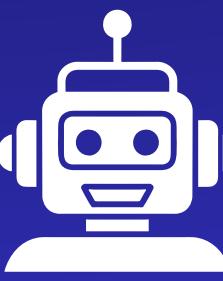
Sobre muestreo# (Oversampling)

Consiste en aumentar el número de ejemplos de la clase minoritaria (fraudes) para equilibrar el dataset sin perder información de la clase mayoritaria.



SMOTE# (Synthetic Minority Oversampling Technique)

Genera ejemplos sintéticos interpolando características entre ejemplos de fraude existentes, creando nuevos casos en el espacio de características.

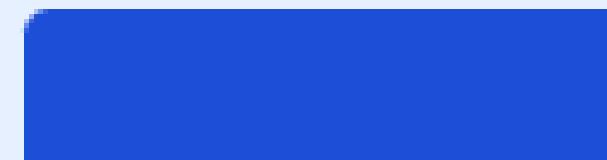


ADASYN #(Adaptive Synthetic Sampling)

Evolución de SMOTE que genera más ejemplos sintéticos para los casos de fraude más difíciles de clasificar, concentrándose en las fronteras de decisión.

MODELO ANTES vs SMOTE

Antes del balanceo



No Fraude: 99.7%

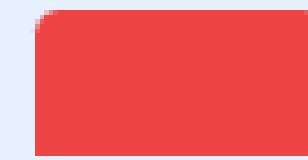


Fraude: 0.3%

Después de SMOTE



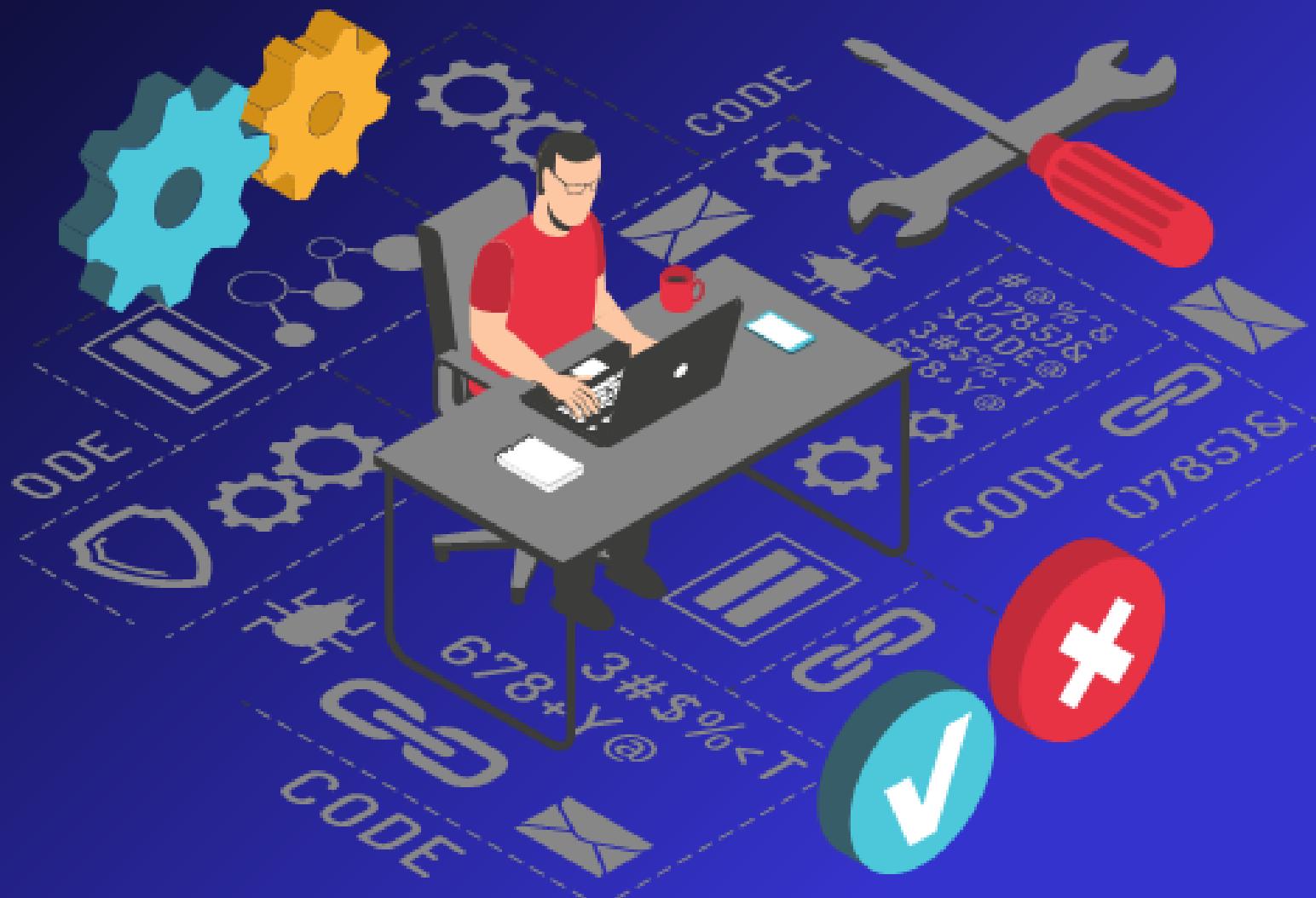
No Fraude: 50%



Fraude: 50%



Entrenamos los Modelos Base



RANDOM FOREST

XGBOOST



¿Porque utilizar Random Forest Classifier?

#En que modelos pordemos aplicar
Random Forest Classifier

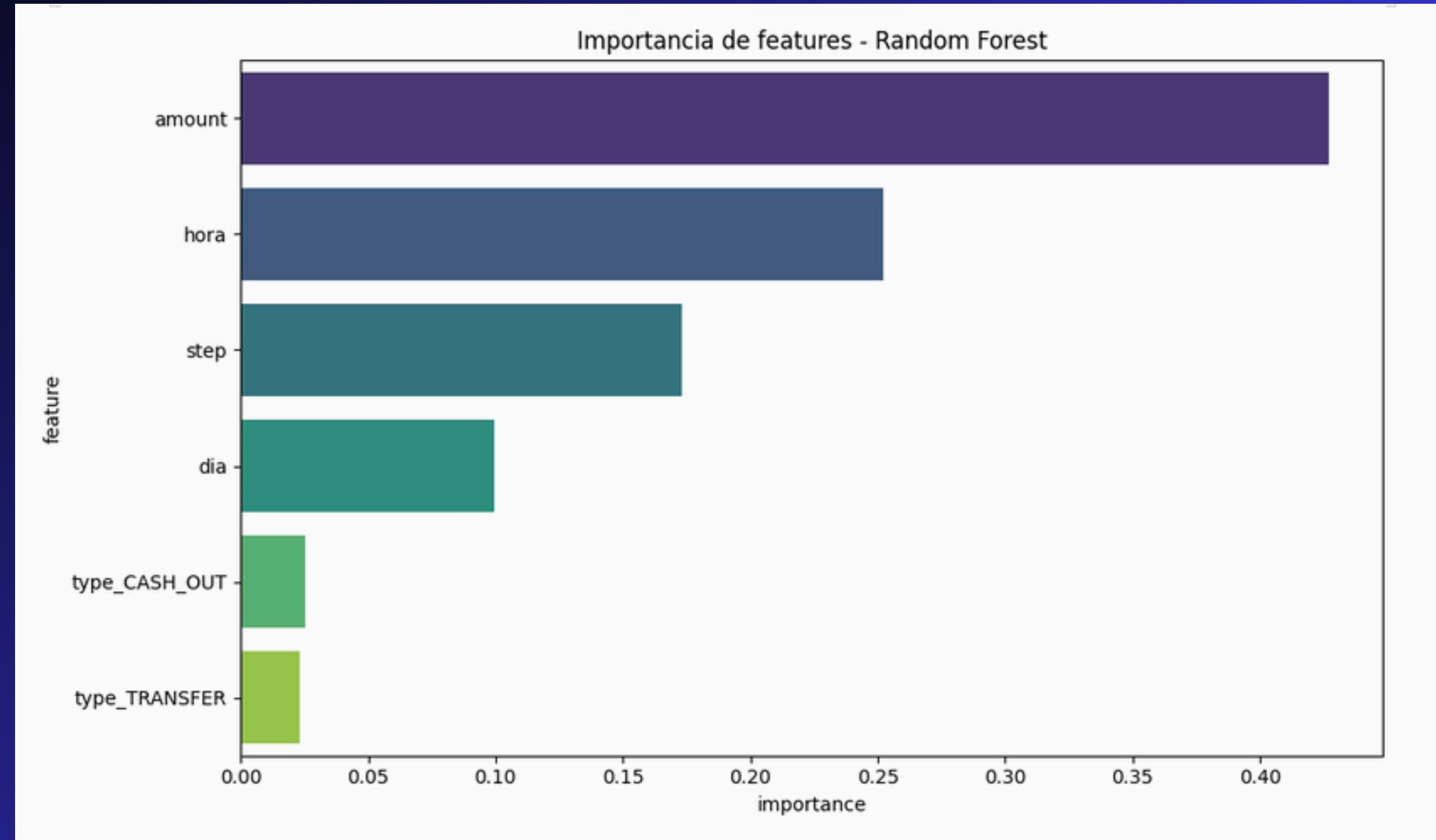
Tipo de tarea	Ejemplos típicos
Clasificación	Fraude/no fraude, spam/no spam, enfermedad/sana
Regresión	Predecir precios, tiempos, cantidades
Feature importance	Saber qué variables son más relevantes
Datos ruidosos	Manejar datos con errores o inconsistencias
Conjuntos grandes	Modelar problemas con muchas columnas (features)

#		precision	recall	f1-score	support
0		0.9772	0.9841	0.9806	36959
1		0.7334	0.6554	0.6922	2464
	accuracy			0.9636	39423
	macro avg	0.8553	0.8198	0.8364	39423
	weighted avg	0.9620	0.9636	0.9626	39423

Para el modelo Random Forest vemos que el valor obtenido de recall está muy lejos del valor óptimo.



Aplicamos “feature importances”



El feature más importante es “monto” (“amount”)

XGBoost

```
# Crear el modelo
model = XGBClassifier()

# Entrenar
model.fit(X_train, y_train)

# Predecir
y_pred = model.predict(X_test)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred, digits=4))
```

	precision	recall	f1-score	support
0	0.9984	0.9999	0.9992	828659
1	0.9335	0.4732	0.6281	2464
accuracy			0.9983	831123
macro avg	0.9660	0.7366	0.8136	831123
weighted avg	0.9982	0.9983	0.9981	831123

Para el modelo XGBoost vemos que el valor de recall sigue siendo bajo.



Realizamos varias pruebas

Según los resultados necesitamos detectar la mayor cantidad de fraudes (**Alto Recall**), el mejor modelo hasta ahora es Random Forest + SMOTE

Modelo	Recall Fraude	Precision Fraude	¿Más conservador?
RandomForest	<input checked="" type="checkbox"/> 72%	71%	No (más detecta, más se equivoca)
XGBoost	70.8%	<input checked="" type="checkbox"/> 83%	Sí (menos detecciones, pero más seguras)

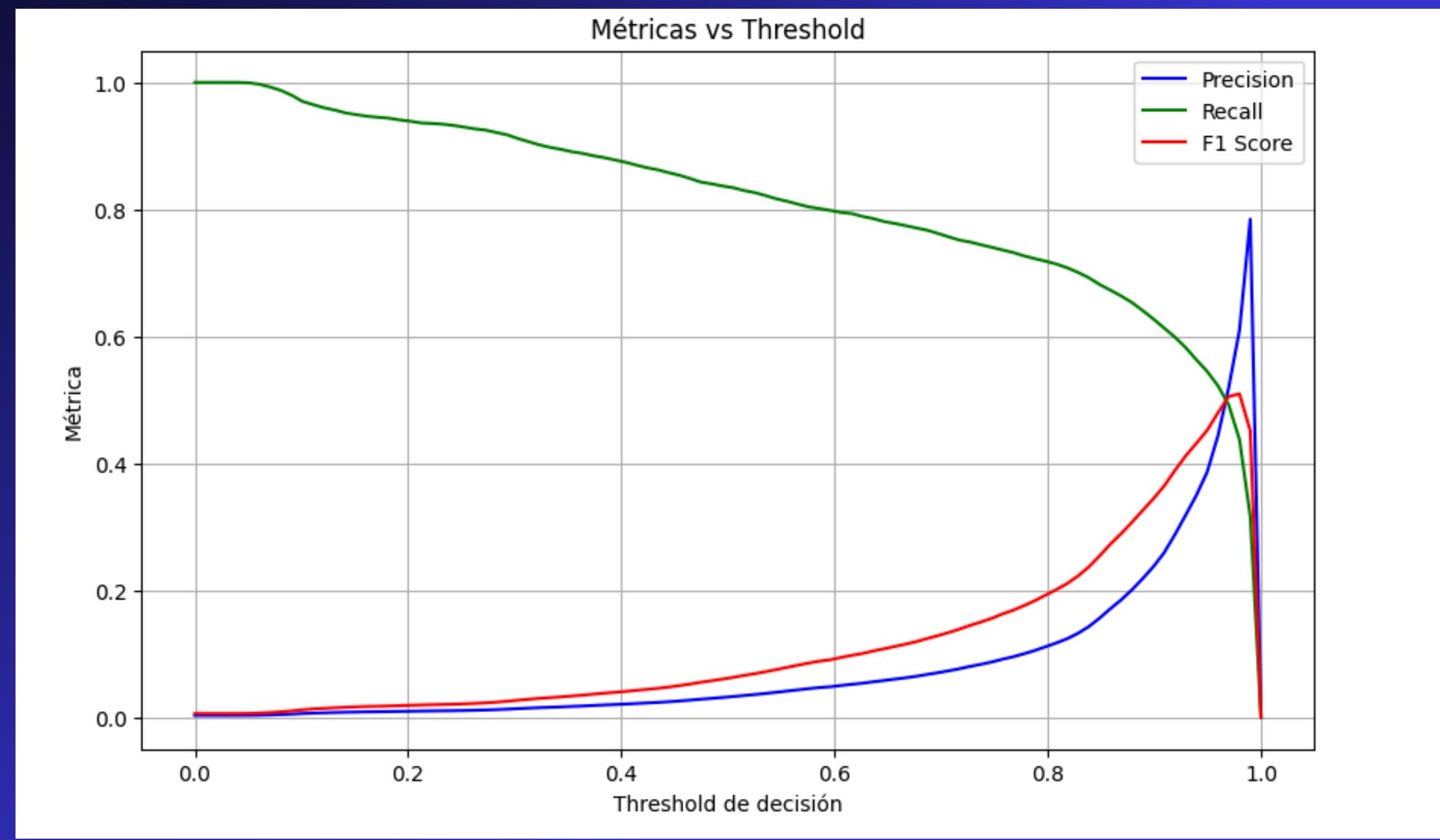
XGBoost

	precision	recall	f1-score
0	0.9712	0.9858	0.9784
1	0.8325	0.7079	0.7651

Random Forest después de
aplicar SMOTE

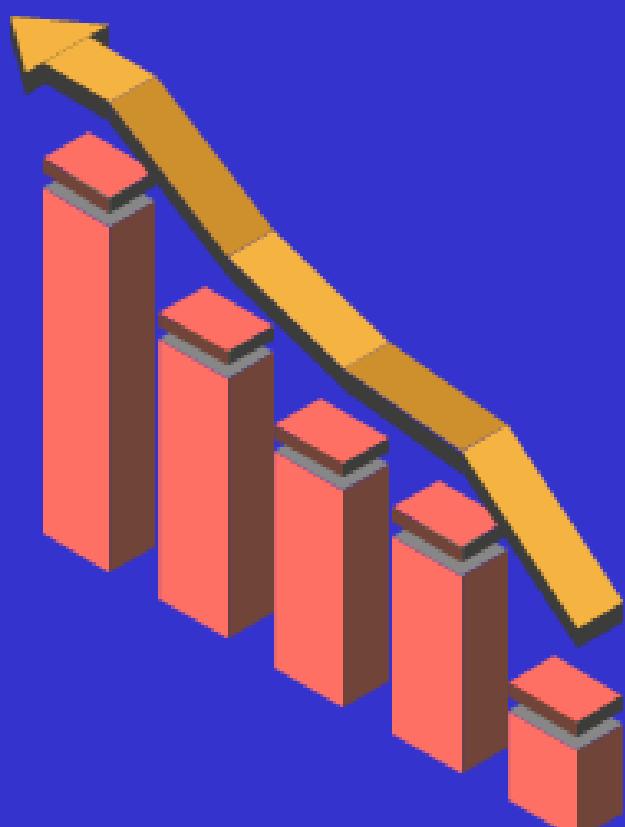
	precision	recall	f1-score	support
0	0.9720	0.9714	0.9717	16426
1	0.7163	0.7206	0.7184	1643

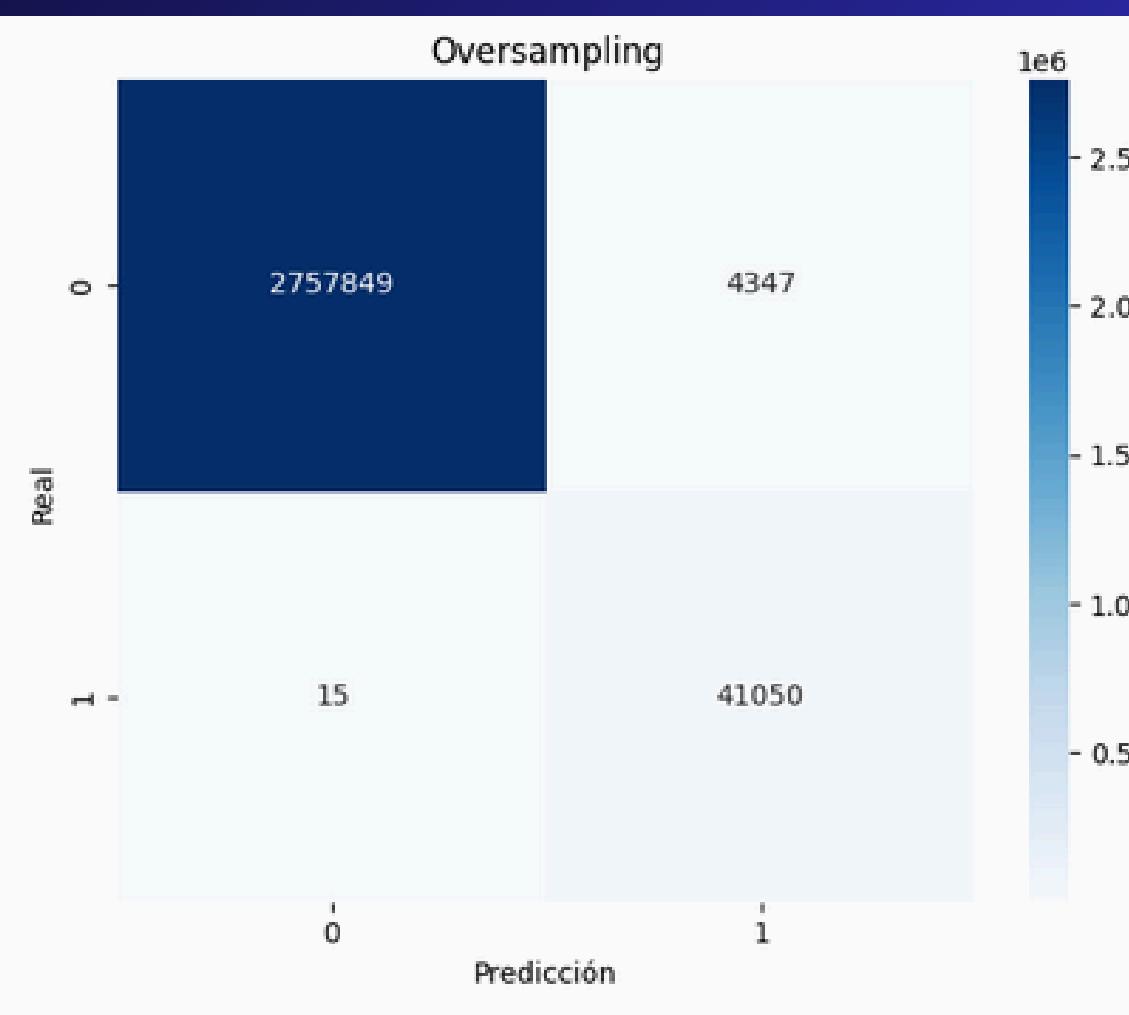
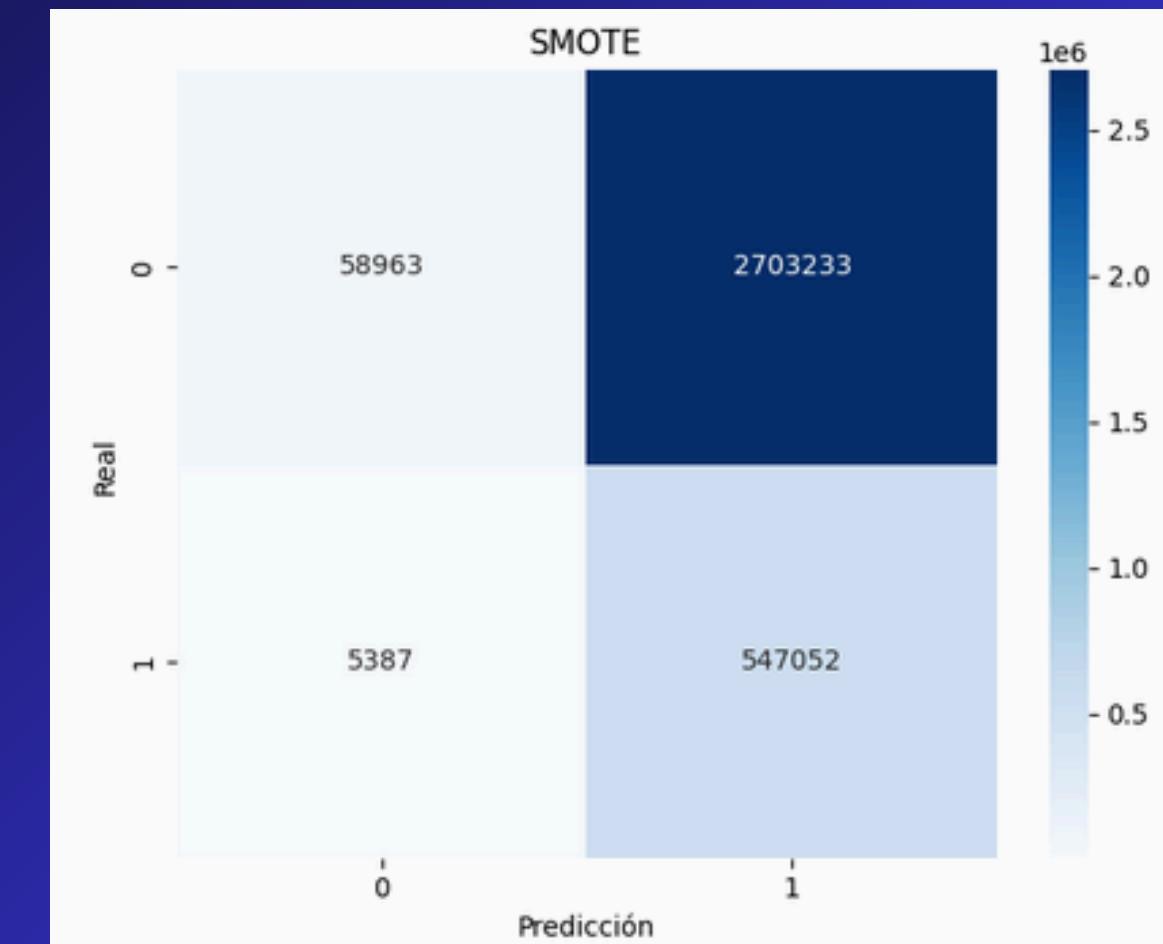
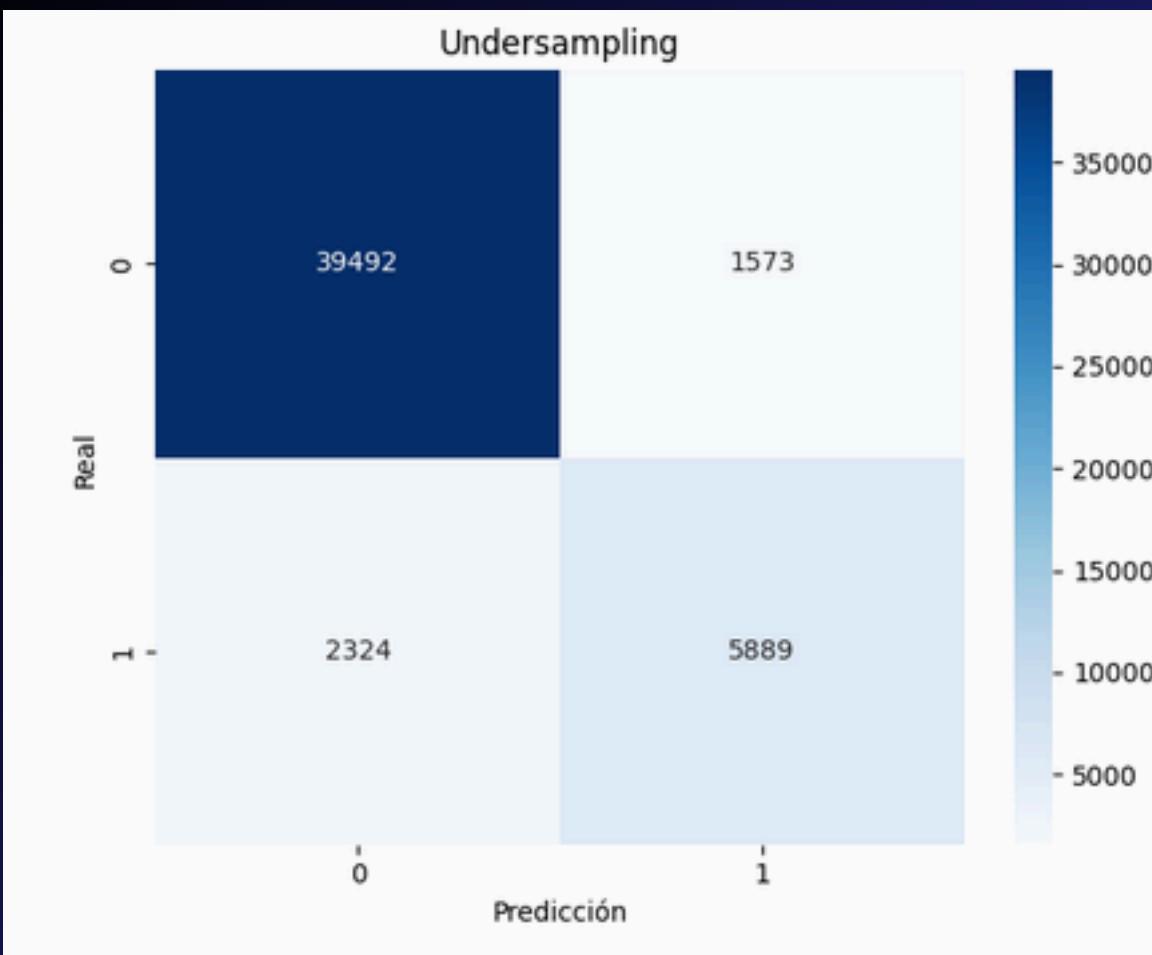
El fin de obtener el mejor **THRESHOLD** te permite maximizar el desempeño del modelo según la métrica y el objetivo del negocio o aplicación.



Umbral de decisión que tiene un valor de corte para variables binarias

Mejor threshold para F1:
0,98





	F1	Precision	Recall
Undersampling	0.751388	0.789199	0.717034
Oversampling	0.949550	0.904245	0.999635
SMOTE	0.287716	0.168309	0.990249

Modelos NO supervisados

Comenzamos a separar las columnas continuas de las binarias para proceder a su escalado.

Separación de variable

Escalado

Reducción PCA

Usamos todas las columnas menos el target para el entrenamiento del modelo.

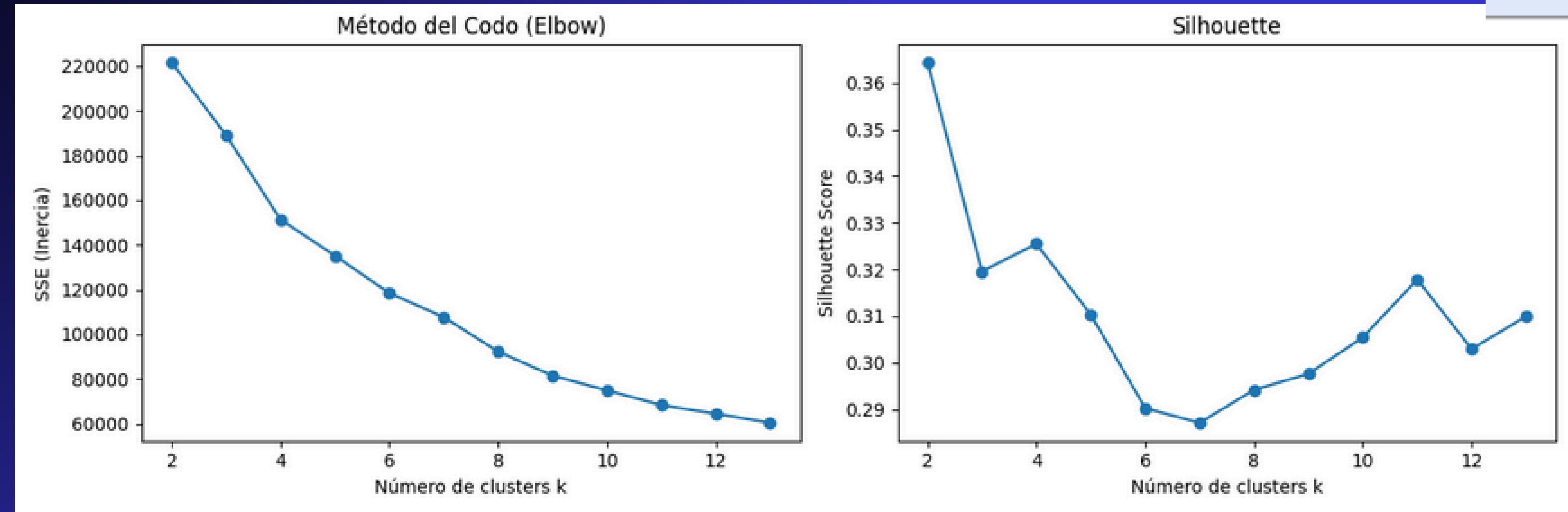
```
# Ejemplo de código de preprocessamiento
col_continuas = ['amount', 'step', 'dia', 'hora']
col_binarias = ['type_CASH_OUT', 'type_TRANSFER']

scaler = StandardScaler()
X_scaled = scaler.transform(X[col_continuas])

pca = PCA(n_components=0.95, random_state=42)
X_pca = pca.fit_transform(X_scaled)
X_feat = np.hstack([X_pca, X[col_binarias].values])
```



Número óptimo de clusters



- **Método de elbow: 3**

0.62
Precisión

0.15
Recall

0.24
F1-Score

- **Método de silhouette: 3**



K Means vs DBSCAN

K Means

K-Means:	precision	recall	f1-score	support
0	0.92	0.99	0.95	77924
1	0.62	0.15	0.24	8213

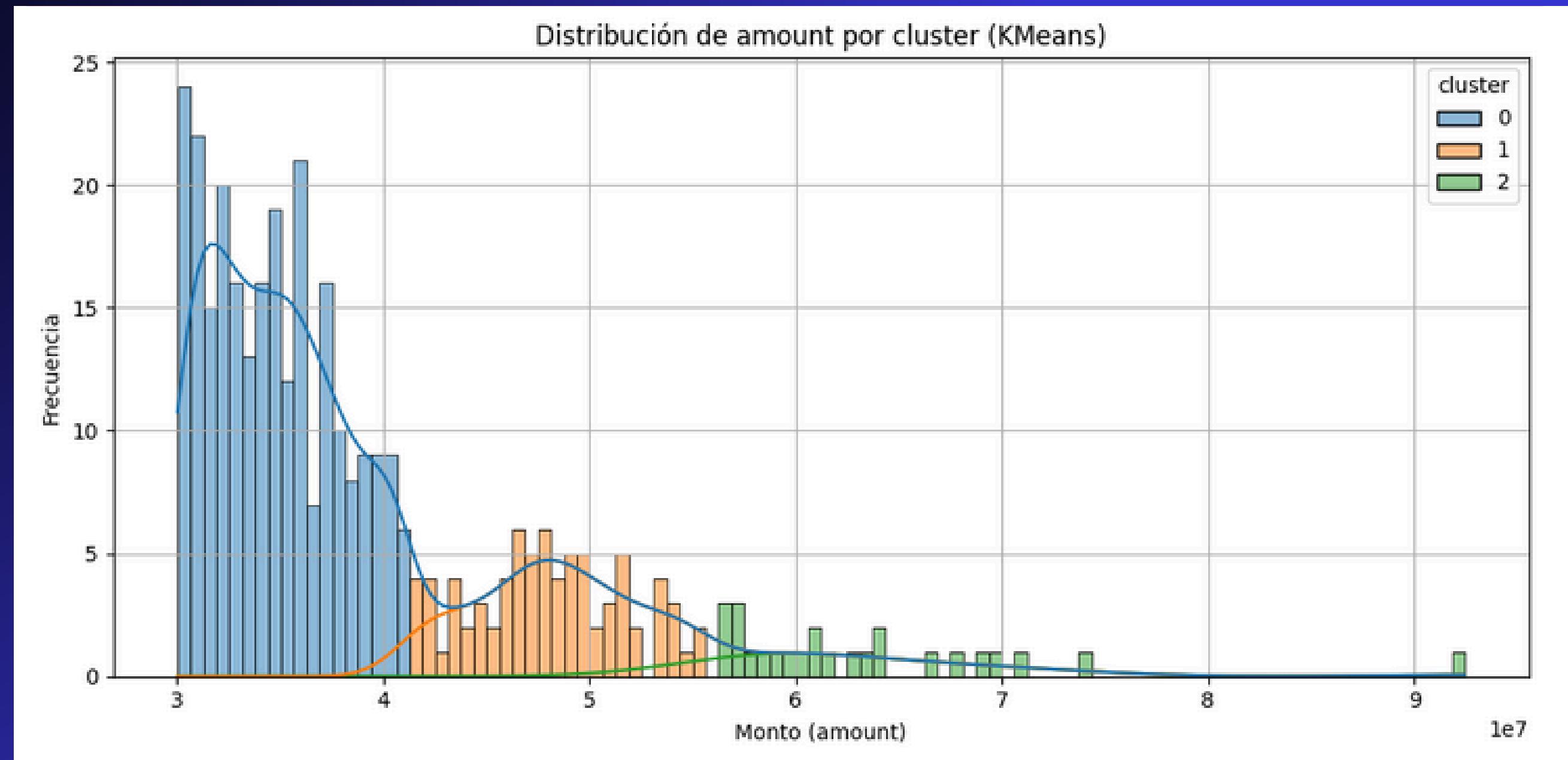
- Alta precisión
- Bajo recall
- Buen detector de casos extremos

DBSCAN

DBSCAN:	precision	recall	f1-score	support
0	0.91	1.00	0.95	77924
1	0.84	0.04	0.07	8213

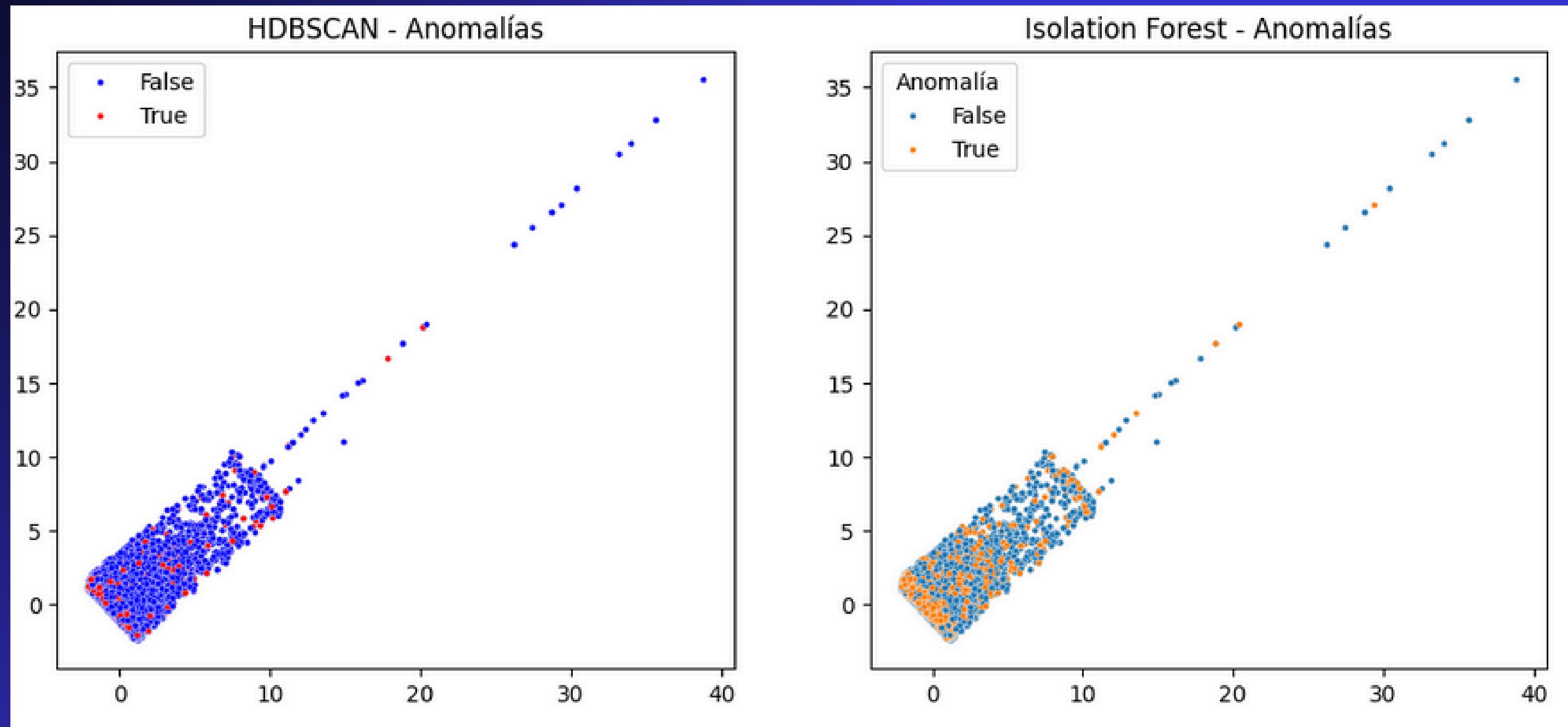
- Alta precisión
- Bajo recall
- Efectivo para outliers

Distribución de monto por clusters KMeans



- Mayor frecuencia de ocurrencias. Rango bajo de montos.
- Frecuencia media de ocurrencias. Rango medio de montos.
- Frecuencia baja. Rango alto de montos.

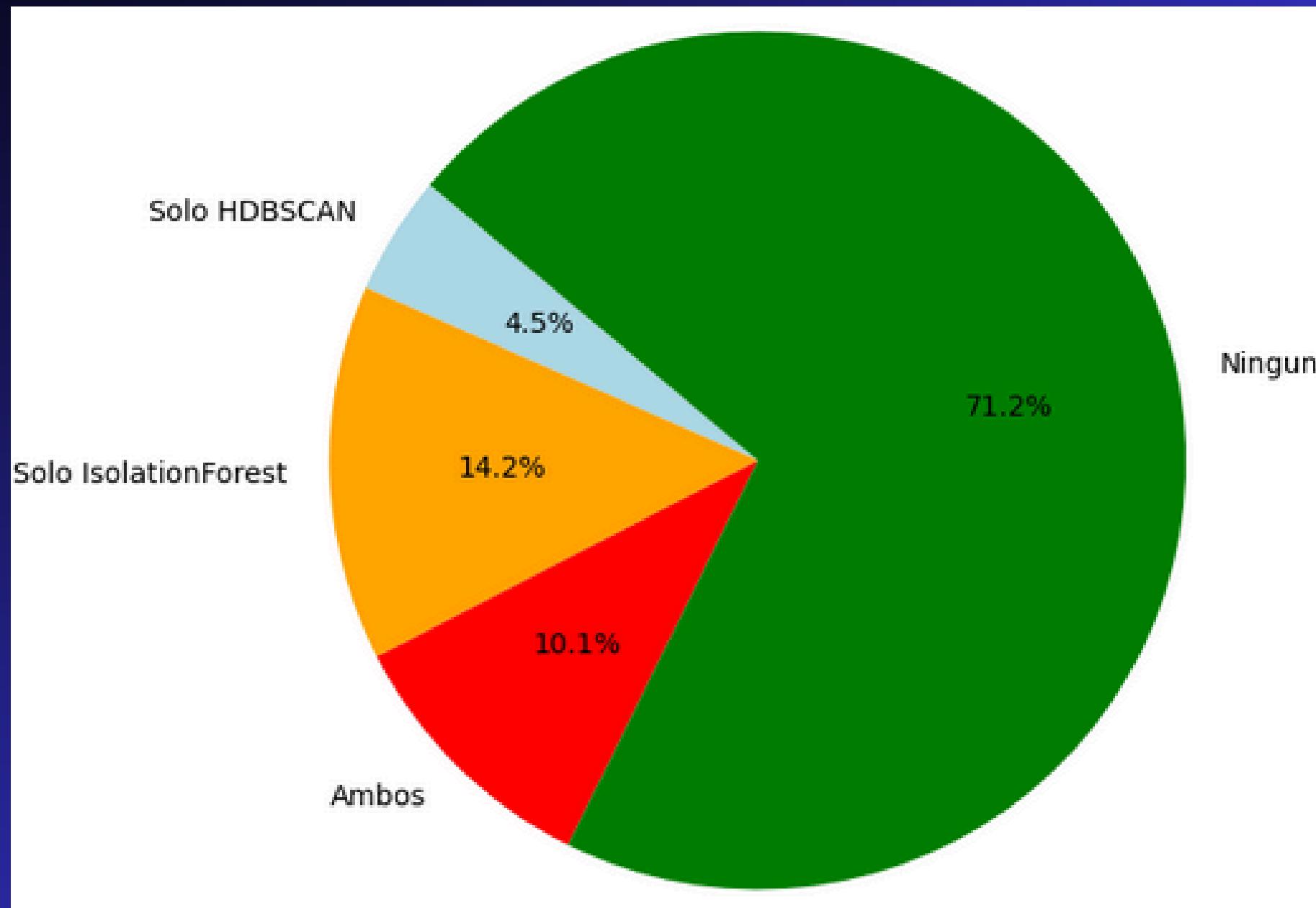
Anomalías



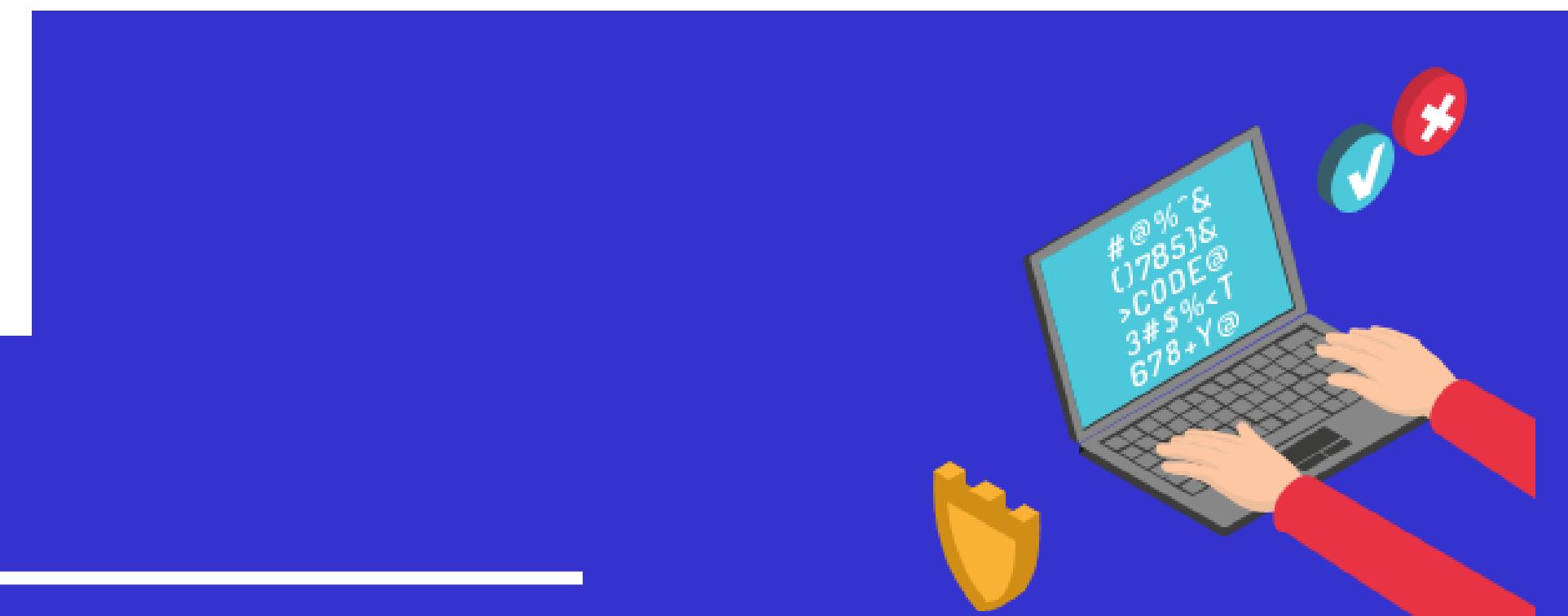
- Menos cantidad de anomalías
- Dispersas
- Mayor cantidad de anomalías
- Distribuidas a lo largo de los datos



Distribución de detección de anomalías



Detección	Cantidad	Porcentaje	Significado
<input checked="" type="checkbox"/> Ambos métodos	1013	10.13%	Anomalías muy confiables (alta certeza)
<input checked="" type="checkbox"/> Solo HDBSCAN	451	4.51%	Detectadas por densidad, no por aislamiento
<input checked="" type="checkbox"/> Solo Isolation Forest	1416	14.16%	Más agresivo, muchos potenciales falsos positivos
<input type="checkbox"/> Ninguno (normal)	7120	71.20%	Consenso fuerte de que son normales



Diferencias entre los métodos aplicados



Isolation Forest:

- **Más rápido**
- **Mejor recall**

HDBCAN

- **Mayor precisión**

CONCLUSIONES



La combinación entre HDBSCAN e Isolation Forest ofrece la mayor efectividad en detección de anomalías (FRAUDES) con un 10.3% de coincidencia.



El método K-Means con K=3 identifico patrones de fraude basados en montos, pero con menor presición que lo métodos especializados en anomalías.



Isolarion Forest resulta más eficiente con data sets masivo y con alta efectividad.



Recomendaciones



Implementar un sistema que combine Isolation Forest para una detección más precisa con la combinación de HDBSCAN que el análisis es más profundo en casos de FRUADES.



Actualizar periodicamente los modelos con nuevos datos para mantener su efectividad frente a nuevas tecnicas de fraudes.



La clave esta es seguir invirtiendo en técnicas de análisis exploratorio y modelos en detección de anomalías.



MUCHAS GRACIAS



FUNDACION YPF - INGENIAS + 2025