deteccion_fraude_transacciones

September 26, 2025

1 Detección de Fraudes en Transacciones Financieras

Materia: Herramientas para el Almacenamiento de Grandes Volúmenes
Estudiante: Daniel Cureño Martínez
Semestre: 5to
Fecha: 24 de septiembre de 2025

1.1 Objetivo

Identificar transacciones potencialmente fraudulentas aplicando reglas de negocio predefinidas, independientemente de la divisa.

[1]:
```
# Importar librerías necesarias
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Confihuración estilos y opciones de visualización
plt.style.use('seaborn-v0_8')
sns.set_palette("Set2")
pd.set_option('display.max_columns', None)
```

1.2 Carga de datos

Cargamos el archivo CSV con las transacciones simuladas y mostramos las primeras filas para verificar la estructura.

[2]:
```
# Cargar el DataSet
df = pd.read_csv('transacciones_simuladas.csv')
# Mostrar las primeras filas
df.head()
```

[2]:

	id_transaccion	id_cliente	nombre_cliente	cuenta_origen	\
0	1	2783	Philip Knight	ES72380307822087479772	
1	2	3696	William Harmon	ES74956814146097211403	
2	3	6403	Amanda Lewis	ES94278160534459740595	
3	4	1156	Victoria Burton	ES18259905541180196560	
4	5	2653	Todd Young	ES59522414203568104424	

```
        fecha     monto divisa tipo_transaccion     ciudad  \
0  13/09/2025  31551.32    USD    transferencia  Queretaro
1  16/09/2025  36889.11    MXN           retiro     Juarez
2  04/07/2025  30644.10    MXN           retiro      Paris
3  06/09/2025   7869.29    USD           retiro     Juarez
4  03/04/2025  37940.37    MXN           retiro     Puebla

            cuenta_destino categoria_comercio
0  ES1994692018319175947 4               NaN
1                      NaN               NaN
2                      NaN               NaN
3                      NaN               NaN
4                      NaN               NaN
```

1.3 Preprocesamiento

- Conversión de la columna monto a tipo numérico.
- Definición de ciudades mexicanas para identificar operaciones en el extranjero (extendida con ciudades comunes en México).
- Creación de la bandera es_extranjera (con manejo case-insensitive para robustez).
- Reemplazo de NaN en categoria_comercio por cadena vacía para evitar errores en comparaciones.

[4]:
```
# Convertir monto a numérico
df['monto'] = pd.to_numeric(df['monto'], errors='coerce')

# Lista de ciudades mexicanas (case-insensitive)
ciudades_mexicanas = {
'Queretaro', 'Juarez', 'Monterrey', 'Puebla', 'Merida', 'Leon', 'Torreon',
'CDMX', 'Guadalajara', 'Tijuana', 'Hermosillo', 'Cancun', 'Veracruz',
'Oaxaca', 'Chihuahua'
}

df['ciudad'] = df['ciudad'].str.lower() # Convertir a minúsculas para matching
df['es_extranjera'] = ~df['ciudad'].isin([c.lower() for c in␣
 ↪ciudades_mexicanas])

# Reemplazar NaN en categoria_comercio
df['categoria_comercio'] = df['categoria_comercio'].fillna('')

# Imprimir ciudades extranjeras detectadas
print("Ciudades extranjeras detectadas:", df[df['es_extranjera']]['ciudad'].
 ↪unique())
```

```
Ciudades extranjeras detectadas: ['paris' 'madrid' 'brasil' 'barcelona' 'los
angeles' 'sidney' 'london'
 'bogota' 'boston' 'new york' 'buenos aires' 'montreal' 'israel' 'cali'
 'india' 'china' 'nigeria' 'ucrania' 'peru' 'libia' 'texas' 'milan'
```

```
'moscu']
```

1.4 Aplicación de reglas de fraude

Se implementan las 4 reglas del enunciado:

1. **Regla 1**: `monto > 45000`
2. **Regla 2**: `retiro` en ciudad extranjera y `monto > 25000`
3. **Regla 3**: `transferencia` y (`monto > 30000` **o** `ciudad extranjera`)
4. **Regla 4**: `compra` en categoría `"viajes"` y `monto > 35000`

Calculamos el total de fraudes y el conteo por regla (considerando overlaps).

[7]:
```
# Definir las reglas
regla1 = df['monto'] > 45000
regla2 = (df['tipo_transaccion'] == 'retiro') & df['es_extranjera'] &␣
 ↪(df['monto'] > 25000)
regla3 = (df['tipo_transaccion'] == 'transferencia') & ((df['monto'] > 30000) |␣
 ↪df['es_extranjera'])
regla4 = (df['tipo_transaccion'] == 'compra') & (df['categoria_comercio'] ==␣
 ↪'viajes') & (df['monto'] > 35000)

# Crear columna de fraude
df['es_fraude'] = regla1 | regla2 | regla3 | regla4

# Calcular fraudes únicos por regla (sin contar overlaps en otros)
fraudes_regla1 = df[regla1 & ~(regla2 | regla3 | regla4)]['es_fraude'].sum()
fraudes_regla2 = df[regla2 & ~(regla1 | regla3 | regla4)]['es_fraude'].sum()
fraudes_regla3 = df[regla3 & ~(regla1 | regla2 | regla4)]['es_fraude'].sum()
fraudes_regla4 = df[regla4 & ~(regla1 | regla2 | regla3)]['es_fraude'].sum()

# Total general
total_fraudes = df['es_fraude'].sum()

# Imprimir resultados
print(f"Total de fraudes detectados: {total_fraudes}")
print(f"Regla 1 (monto > 45,000): {regla1.sum()}")
print(f"Regla 2 (retiro en extranjero > 25k): {regla2.sum()}")
print(f"Regla 3 (transferencia en extranjero o > 30k): {regla3.sum()}")
print(f"Regla 4 (compra en 'viajes' > 35k): {regla4.sum()}")
```

```
Total de fraudes detectados: 2075
Regla 1 (monto > 45,000): 990
Regla 2 (retiro en extranjero > 25k): 10
Regla 3 (transferencia en extranjero o > 30k): 1301
Regla 4 (compra en 'viajes' > 35k): 158
```

1.5 Visualización: Fraudes por tipo de transacción

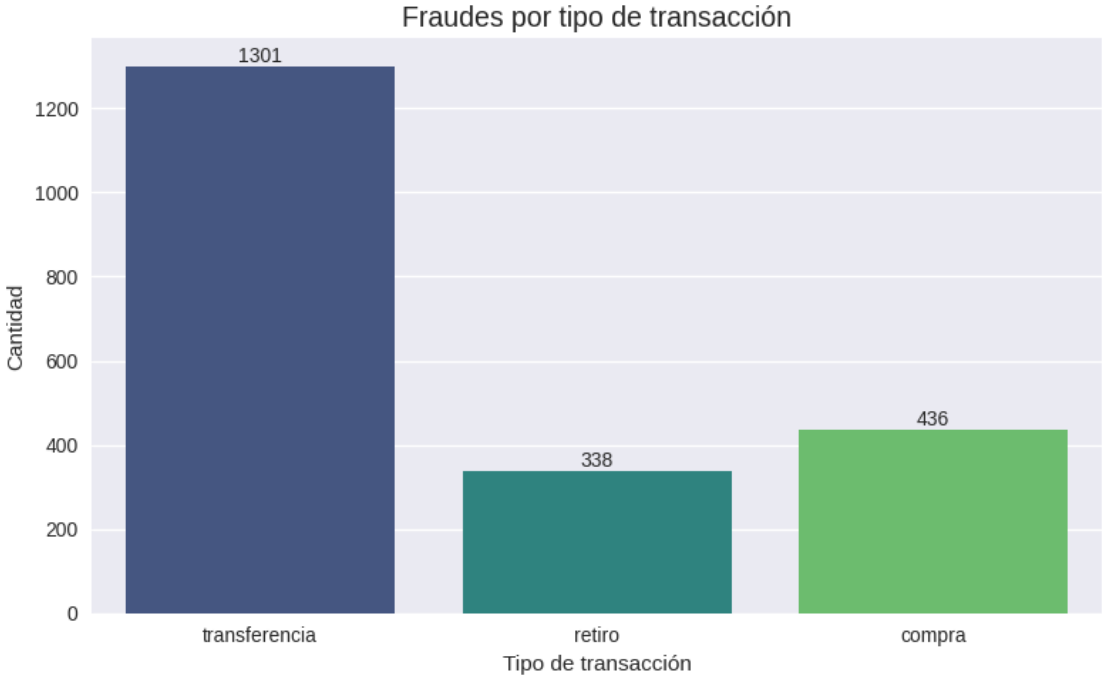
Visualizamos la cantidad de fraudes por tipo de transacción, con etiquetas numéricas para mayo

[12]:
```
# Filtrar solo las transacciones fraudulentas
fraudes = df[df['es_fraude']]

# Crear gráfico de barras
plt.figure(figsize=(8, 5))
ax = sns.countplot(data=fraudes, x='tipo_transaccion', palette='viridis')
plt.title('Fraudes por tipo de transacción', fontsize=14)
plt.xlabel('Tipo de transacción')
plt.ylabel('Cantidad')

# Añadir etiquetas numéricas a las barras
for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() / 2., p.
 ↪get_height()),
                ha='center', va='bottom')

plt.tight_layout()
plt.show()
```

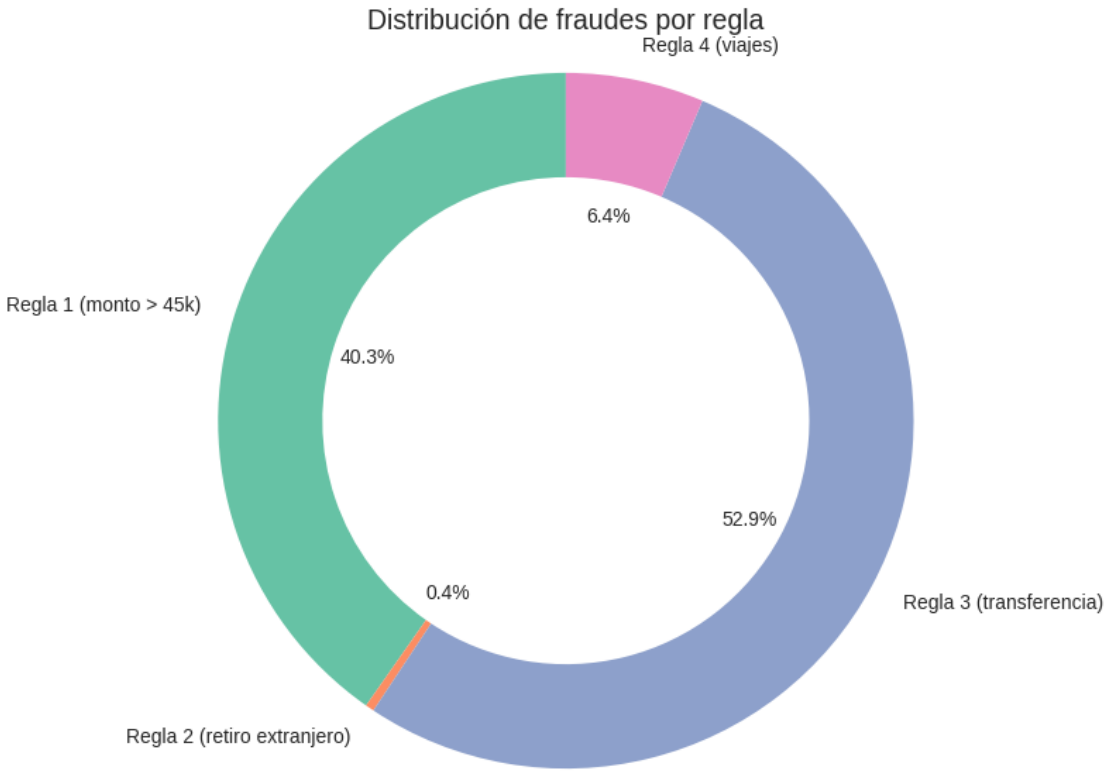


1.6 Visualización: Distribución de fraudes por regla

Gráfico de pastel para mostrar la contribución relativa de cada regla (basado en conteos totales por regla).

[13]:
```
# Conteos por regla
regla_counts = [regla1.sum(), regla2.sum(), regla3.sum(), regla4.sum()]
regla_labels = ['Regla 1 (monto > 45k)', 'Regla 2 (retiro extranjero)',
'Regla 3 (transferencia)', 'Regla 4 (viajes)']

# Gráfico de pastel
plt.figure(figsize=(7, 7))
plt.pie(regla_counts, labels=regla_labels, autopct='%1.1f%%', startangle=90,
wedgeprops=dict(width=0.3), colors=sns.color_palette('Set2'))
plt.title('Distribución de fraudes por regla', fontsize=14)
plt.axis('equal')
plt.show()
```



1.7 Resultados: Transacciones fraudulentas

Mostramos las primeras y últimas 10 transacciones fraudulentas, junto con estadísticas básicas de los montos

[14]:
```
# Mostrar primeras y últimas 10 transacciones fraudulentas
print("Primeras 10 transacciones fraudulentas:")
print(fraudes[['id_transaccion', 'monto', 'tipo_transaccion', 'ciudad',␣
 ↪'categoria_comercio']].head(10))
print("\nÚltimas 10 transacciones fraudulentas:")
print(fraudes[['id_transaccion', 'monto', 'tipo_transaccion', 'ciudad',␣
 ↪'categoria_comercio']].tail(10))

# Estadísticas de montos fraudulentos
print("\nEstadísticas básicas de montos fraudulentos:")
print(fraudes['monto'].describe())
```

```
Primeras 10 transacciones fraudulentas:
    id_transaccion     monto tipo_transaccion     ciudad categoria_comercio
0                1  31551.32    transferencia  queretaro
2                3  30644.10           retiro      paris
8                9  41811.10    transferencia     merida
9               10  37271.83    transferencia     puebla
11              12  35360.16    transferencia       leon
12              13  48471.27    transferencia  monterrey
13              14  48485.84           compra  queretaro        restaurante
18              19  49904.69           retiro    torreon
20              21  49727.53           retiro    tijuana
29              30  39122.25    transferencia  queretaro

Últimas 10 transacciones fraudulentas:
      id_transaccion     monto tipo_transaccion       ciudad  \
9945            9946  45547.82           retiro         cdmx
9954            9955  40547.18    transferencia       juarez
9957            9958  45926.29    transferencia      tijuana
9963            9964  48264.94    transferencia    monterrey
9964            9965  33143.85    transferencia         cdmx
9966            9967  43362.94    transferencia  guadalajara
9967            9968  38800.11    transferencia       puebla
9974            9975  30203.27    transferencia         cdmx
9981            9982  47405.66           compra         cdmx
9994            9995  45492.74           retiro       merida

     categoria_comercio
9945
9954
9957
9963
9964
9966
9967
9974
9981        restaurante
```

```
9994

Estadísticas básicas de montos fraudulentos:
count     2075.000000
mean     42420.692723
std       6003.177361
min       2854.430000
25%      37803.980000
50%      44399.040000
75%      47333.755000
max      49995.630000
Name: monto, dtype: float64
```

1.8 Análisis por regla de detección

A continuación, se muestra cuántos fraudes activó cada regla (una transacción puede activar más de una).

[15]:
```python
# Imprimir conteos totales por regla
print(f"Regla 1 (monto > 45,000): {regla1.sum()}")
print(f"Regla 2 (retiro en extranjero > 25k): {regla2.sum()}")
print(f"Regla 3 (transferencia en extranjero o > 30k): {regla3.sum()}")
print(f"Regla 4 (compra en 'viajes' > 35k): {regla4.sum()}")
```

```
Regla 1 (monto > 45,000): 990
Regla 2 (retiro en extranjero > 25k): 10
Regla 3 (transferencia en extranjero o > 30k): 1301
Regla 4 (compra en 'viajes' > 35k): 158
```

1.9 Conclusiones

- Se detectaron 2,075 transacciones fraudulentas, con la mayoría activadas por la Regla 3 (transferencias $>$ 30k o extranjeras), que contribuyó con 1,301 casos.
- La Regla 3 incluye transferencias desde ciudades extranjeras con montos bajos, lo que genera un alto número de detecciones y potenciales falsos positivos. Se recomienda ajustar el umbral de monto o validar manualmente estos casos.
- La Regla 4 detectó 158 fraudes en la categoría 'viajes' con montos $>$ 35k, corrigiendo la suposición inicial de ausencia de esta categoría.
- El enfoque basado en reglas es efectivo para escenarios predefinidos, pero para grandes volúmenes, se sugiere integrar modelos predictivos (ej. Random Forest) y usar herramientas como Apache Spark para escalabilidad.
- Este análisis refleja comprensión técnica y crítica, con propuestas para mejorar la precisión en la detección de fraudes.

[ ]:
```python

```