Técnicas de Unsupervised ML: Clustering, K-Means, Dimensionality reduction, Matrix factorization

En este caso, considerando que trabajo con un dataset que incluye variables transformadas mediante PCA (Análisis de Componentes Principales) (como V1 a V28), tiene sentido que enfoquemos las técnicas de machine learning no supervisado en la exploración de patrones ocultos y estructuras subyacentes en los datos sin depender de etiquetas o clases específicas.

Recomendación de Técnicas No Supervisadas:

K-Means Clustering:

Esta técnica agrupa las observaciones en clusters basándose en la similitud de las características. Dado que ya tienes variables transformadas con PCA, K-Means puede ayudarte a identificar grupos de transacciones similares.

Útil para segmentación de clientes o detección de comportamientos anómalos.

Reducción de Dimensionalidad (PCA o t-SNE):

Aunque ya tienes variables PCA, podrías aplicar una reducción de dimensionalidad adicional para visualizar los datos en 2D o 3D. Técnicas como t-SNE pueden ser útiles para representar los datos de manera más interpretable y detectar clusters o patrones de manera visual.

PCA puede reducir las variables a unas pocas componentes principales para facilitar el análisis y visualización de clusters.

Factorización de Matrices (Matrix Factorization):

Ideal si tienes datos faltantes o si buscas descomponer una matriz grande en factores más pequeños. Se usa comúnmente en sistemas de recomendación, pero también puede ayudar a capturar la estructura latente en los datos financieros o de fraude.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise):

Una técnica de clustering basada en la densidad que es útil cuando hay ruido o cuando los clusters tienen formas no esféricas. DBSCAN también es bueno para detectar anomalías como las transacciones fraudulentas que son pocos comunes.

Estrategia Recomendada para este Caso:

Teniendo en cuenta que trabajas con transacciones financieras y detección de fraude, podríamos aplicar un enfoque de clustering y reducción de dimensionalidad. Aquí están los pasos recomendados:

Aplicar K-Means para agrupar transacciones:

Agruparemos las transacciones en clusters basándonos en las variables PCA para identificar patrones de comportamiento.

Luego, analizaremos los clusters resultantes para ver si hay clusters con transacciones potencialmente anómalas (como aquellas con montos de transacción altos).

Reducción de Dimensionalidad para Visualización:

Utilizaremos t-SNE o una reducción de dimensionalidad para visualizar los datos en 2D o 3D y observar si los clusters tienen sentido.

Detección de Anomalías con DBSCAN:

Finalmente, podríamos usar DBSCAN para detectar puntos que no pertenecen a ningún cluster, lo que nos dará pistas sobre transacciones anómalas (posibles fraudes).

Código para K-Means Clustering y Reducción de Dimensionalidad con PCA/t-SNE:

A continuación te muestro un código para aplicar K-Means y PCA con visualización, además de la opción de utilizar t-SNE para mejor visualización.

python

Copiar código

```
import pandas as pd
```

```
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
from scipy.io import arff
import seaborn as sns

# Cargar el archivo ARFF
file_path = r'C:\DB_Covid19Arg\csv_archivos_limpios\Amazon_test\phpKo8OWT.arff'
data, meta = arff.loadarff(file_path)

# Convertir los datos en un DataFrame de pandas
df = pd.DataFrame(data)

# Convertir la variable 'Class' a entero (eliminar los caracteres b'')
df['Class'] = df['Class'].apply(lambda x: int(x.decode('utf-8')))

# Definir las características (X) - Usamos solo las variables PCA y Amount para clustering
X = df.drop(columns=['Class'])

# Reducción de dimensionalidad con PCA para visualizar en 2D
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Aplicar K-Means con 3 clusters (puedes ajustar el número de clusters)
kmeans = KMeans(n_clusters=3, random_state=42)
clusters = kmeans.fit_predict(X)
```

```
# Añadir los clusters al DataFrame
df['Cluster'] = clusters

# Visualizar los clusters con PCA
plt.figure(figsize=(10, 7))
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=df['Cluster'], palette='viridis')
plt.title('Clusters visualizados con PCA')
plt.show()

# Visualización con t-SNE para mejor separación
tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X)

# Visualizar los clusters con t-SNE
plt.figure(figsize=(10, 7))
sns.scatterplot(x=X_tsne[:, 0], y=X_tsne[:, 1], hue=df['Cluster'], palette='viridis')
plt.title('Clusters visualizados con t-SNE')
plt.show()
```

Explicación del Código:

K-Means: Se aplica con 3 clusters (puedes ajustar este número según el análisis).

PCA: Se utiliza para reducir las dimensiones a 2 componentes principales y visualizar los clusters.

t-SNE: Una técnica de reducción de dimensionalidad no lineal que es excelente para visualizar los datos en 2D con mayor claridad en cuanto a las separaciones entre clusters.

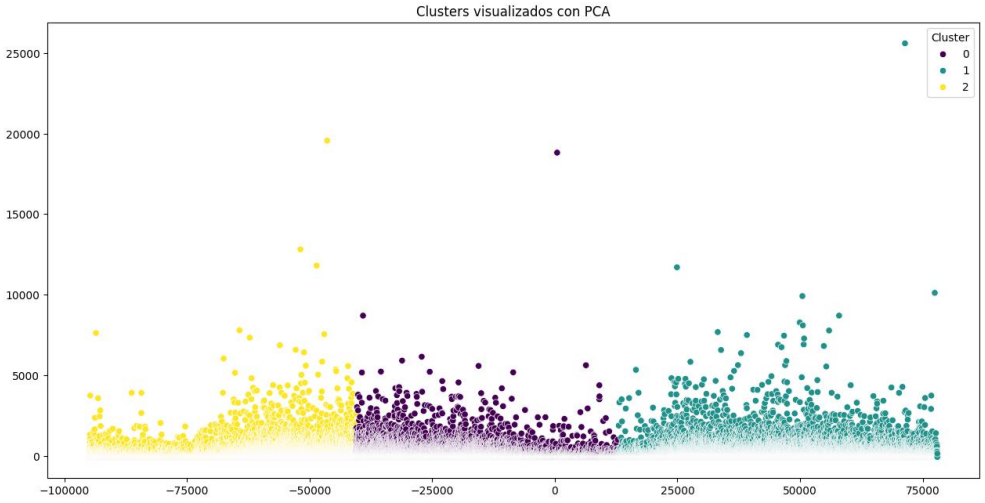
Próximos Pasos:

Interpretar los clusters: Una vez que obtengas los clusters, puedes analizar las características dentro de cada cluster para detectar patrones en los datos (por ejemplo, si ciertos clusters corresponden a transacciones con montos más altos).

Aplicar DBSCAN: Si deseas detectar anomalías, podemos aplicar DBSCAN para identificar transacciones que no pertenezcan a ningún cluster.

Si te interesa aplicar DBSCAN o profundizar en algún análisis en particular, házmelo saber y te proporcionaré el código correspondiente. ¡Avísame si tienes dudas!

GRAFICO 1



A partir del gráfico de PCA con K-Means, se pueden observar algunos puntos clave:

Análisis del Gráfico de Clusters con PCA:

Distribución de los Clusters:

Se observa que los puntos se agrupan en tres clusters distintos (Cluster 0, Cluster 1, Cluster 2).

El Cluster 2 (amarillo) parece estar concentrado hacia un lado de la gráfica, mientras que el Cluster 1 (verde) y el Cluster 0 (morado) están más dispersos en el espacio.

Outliers o Puntos Distantes:

Es notable que algunos puntos (pertenecientes a todos los clusters) están más alejados de los demás, lo que puede sugerir posibles transacciones atípicas o anómalas que podrían ser investigadas más a fondo.

Posibles Anomalías:

Los puntos alejados de los centros de los clusters podrían ser potenciales anomalías (posibles fraudes o comportamientos inusuales), que podríamos investigar con más detalle. Estas transacciones son interesantes para un análisis adicional.

Próximos Pasos:

Exploración de Clusters:

Puedes examinar las características de los datos que pertenecen a cada cluster para ver si hay algún patrón interesante, como:

¿Los clusters tienen montos de transacción significativamente diferentes?

¿Algún cluster está dominado por transacciones fraudulentas o no fraudulentas?

Aplicación de DBSCAN:

Si quieres detectar outliers o anomalías directamente, puedes aplicar la técnica de DBSCAN para identificar transacciones que no encajan en ningún cluster.

Visualización con t-SNE:

Para una mejor separación y visualización de los clusters en alta dimensión, puedes ejecutar el gráfico t-SNE como se mencionó en el código. Esto podría ayudarte a ver si los clusters se separan más claramente con esta técnica.

Para explorar las características de los datos en cada cluster y analizar si hay patrones interesantes, como las diferencias en el monto de las transacciones o la proporción de transacciones fraudulentas en cada cluster, podemos proceder con los siguientes pasos:

Pasos para la Exploración de Clusters:

1. **Agrupar los datos por cluster**:

- Analizar el monto de las transacciones (`Amount`) en cada cluster para ver si hay diferencias significativas entre ellos.

2. **Contar las transacciones fraudulentas en cada cluster**:

- Verificar si algún cluster tiene más transacciones fraudulentas (`Class = 1`) que otros.

3. **Visualizar los resultados**:

- Graficar las distribuciones de montos por cluster y mostrar la proporción de fraudes en cada cluster.

Código para la Exploración de Clusters:

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Cargar el archivo ARFF (asume que ya tienes el DataFrame `df` con clusters)
file_path = r'C:\DB_Covid19Arg\csv_archivos_limpios\Amazon_test\phpKo8OWT.arff'
data, meta = arff.loadarff(file_path)

# Convertir los datos en un DataFrame de pandas
df = pd.DataFrame(data)

# Convertir la variable 'Class' a entero (eliminar los caracteres b'')
df['Class'] = df['Class'].apply(lambda x: int(x.decode('utf-8')))
```

```
# Añadir la columna de los clusters ya generados (asegúrate de haber corrido el código K-Means previamente)
X = df.drop(columns=['Class'])
kmeans = KMeans(n_clusters=3, random_state=42)
df['Cluster'] = kmeans.fit_predict(X)

# Agrupar por cluster y calcular estadísticas sobre 'Amount' (monto de transacción)
cluster_stats = df.groupby('Cluster')['Amount'].agg(['mean', 'median', 'std', 'count'])

# Mostrar las estadísticas de cada cluster
print(cluster_stats)

# Contar la cantidad de fraudes (Class = 1) en cada cluster
fraud_counts = df.groupby('Cluster')['Class'].sum()
fraud_proportions = df.groupby('Cluster')['Class'].mean()  # Proporción de fraudes por cluster

# Mostrar los resultados
print(f"Cantidad de fraudes por cluster:\n{fraud_counts}")
print(f"Proporción de fraudes por cluster:\n{fraud_proportions}")

# Visualizar la distribución de 'Amount' por cluster
plt.figure(figsize=(10, 6))
sns.boxplot(x='Cluster', y='Amount', data=df)
plt.title('Distribución de Monto de Transacciones por Cluster')
plt.show()

# Visualizar la proporción de fraudes por cluster
```

fraud_proportions.plot(kind='bar', color='skyblue', title='Proporción de Fraudes por Cluster')

plt.ylabel('Proporción de Fraudes')

plt.show()

```

### Explicación del Código:

1. **Estadísticas de los Montos por Cluster**:

   - Calculamos estadísticas como el **promedio**, **mediana**, **desviación estándar** y el **conteo de transacciones** en cada cluster para el campo `Amount`. Esto nos dará una idea de cómo varían los montos de las transacciones en cada cluster.

2. **Cantidad y Proporción de Fraudes por Cluster**:

   - Se cuenta cuántas transacciones fraudulentas (Class = 1) hay en cada cluster. También calculamos la **proporción de fraudes** por cluster, lo que te permitirá identificar si algún cluster tiene una mayor proporción de transacciones fraudulentas.

3. **Visualización**:

   - Un **boxplot** de los montos por cluster para visualizar las distribuciones y ver si hay diferencias significativas.

   - Un gráfico de barras para mostrar la **proporción de fraudes** en cada cluster, lo que ayudará a identificar si algún cluster está más asociado con fraudes.

### Qué Buscar en los Resultados:

- **Diferencias en Montos de Transacciones**: Si un cluster tiene transacciones con montos mucho más altos o más bajos que los otros clusters, esto podría indicar diferentes comportamientos.
```

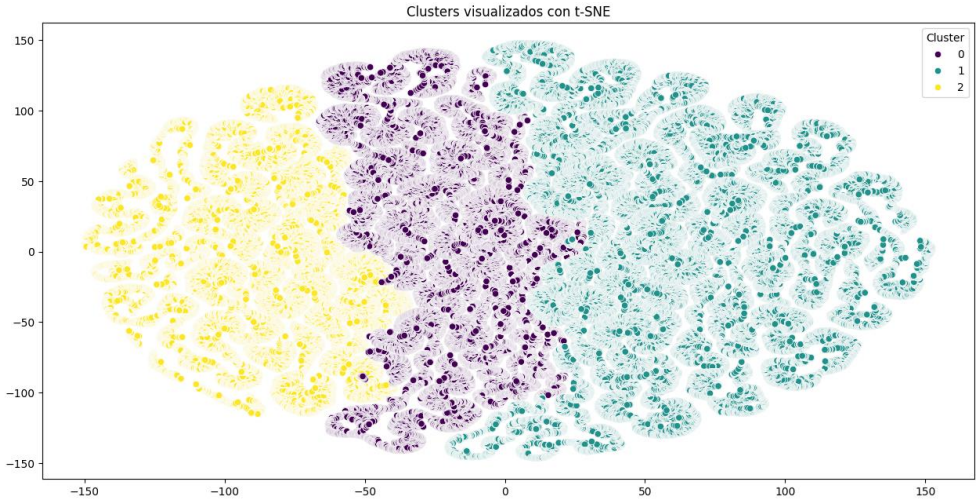
- **Proporción de Fraudes**: Si un cluster tiene una mayor proporción de transacciones fraudulentas, podrías enfocarte en ese cluster para investigar más a fondo o implementar medidas de seguridad adicionales.

Próximos Pasos:

- Revisa las diferencias en las estadísticas de los clusters. Si ves que algún cluster tiene un comportamiento inusual (por ejemplo, transacciones de monto alto con un alto porcentaje de fraudes), podrías profundizar en esos casos.

- Si los resultados son interesantes, también podrías intentar ajustar los parámetros del modelo de clustering (como el número de clusters) para ver cómo cambian las agrupaciones.

GRAFICO 2



La visualización de los *clusters con t-SNE*. A partir de la gráfica, es notable que los **clusters 0, 1 y 2** están bien separados en el espacio de dos dimensiones, lo cual indica que los datos tienen una estructura clara y bien definida cuando se reduce dimensionalmente con t-SNE.

Análisis del Gráfico t-SNE:

1. **Separación de Clusters**:

   - Los clusters están bien diferenciados, lo que confirma que el modelo de **K-Means** pudo encontrar patrones de similitud en los datos.

   - El **Cluster 2 (amarillo)** está claramente diferenciado de los otros dos clusters, lo que podría sugerir que este grupo tiene características significativamente diferentes, ya sea en términos de comportamiento de transacciones o montos.

2. **Tamaño y Densidad de los Clusters**:

   - El tamaño y la densidad de los puntos en los clusters también pueden ofrecer información interesante. Por ejemplo, el **Cluster 1 (verde)** parece estar más denso y más extendido, lo que sugiere que este grupo puede incluir transacciones con mayor variabilidad.

Próximos Pasos:

1. **Aplicación de DBSCAN**:

   - Como mencionaste, podemos utilizar **DBSCAN** para detectar **outliers** o anomalías que no se ajustan bien a ninguno de los clusters principales. Esto será útil para identificar transacciones potencialmente fraudulentas o inusuales.

2. **Exploración de Patrones en los Clusters**:

   - Podrías investigar más a fondo las características dentro de cada cluster (monto, presencia de fraudes, etc.) para obtener más insights sobre los grupos.

Si deseas continuar con **DBSCAN** para detectar anomalías, puedo proporcionarte el código para realizarlo. ¡Avísame si te gustaría proceder con ese análisis!

A continuación te proporciono el código para aplicar **DBSCAN** y detectar **outliers** o anomalías en tus datos. DBSCAN es una técnica basada en la densidad

que permite encontrar puntos que están separados de los clusters densos, lo que es útil para la detección de anomalías o outliers en problemas como el fraude.

Código para la Aplicación de DBSCAN:

```python
import pandas as pd
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.io import arff

# Cargar el archivo ARFF (asume que ya tienes el DataFrame `df` con clusters)
file_path = r'C:\DB_Covid19Arg\csv_archivos_limpios\Amazon_test\phpKo8OWT.arff'
data, meta = arff.loadarff(file_path)

# Convertir los datos en un DataFrame de pandas
df = pd.DataFrame(data)

# Convertir la variable 'Class' a entero (eliminar los caracteres b'')
df['Class'] = df['Class'].apply(lambda x: int(x.decode('utf-8')))

# Preprocesar los datos (escalado)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df.drop(columns=['Class']))

# Aplicar DBSCAN
```

```
dbscan = DBSCAN(eps=0.5, min_samples=5)  # Puedes ajustar eps y min_samples
dbscan_labels = dbscan.fit_predict(X_scaled)

# Añadir la etiqueta de DBSCAN (outliers = -1)
df['DBSCAN_Cluster'] = dbscan_labels

# Contar cuántos puntos fueron identificados como outliers
outliers_count = len(df[df['DBSCAN_Cluster'] == -1])
print(f"Número de outliers detectados: {outliers_count}")

# Visualizar los clusters y los outliers
plt.figure(figsize=(10, 7))
sns.scatterplot(x=X_scaled[:, 0], y=X_scaled[:, 1], hue=df['DBSCAN_Cluster'], palette='viridis', legend="full")
plt.title('Clusters y Outliers detectados con DBSCAN')
plt.show()

# Mostrar algunos ejemplos de transacciones marcadas como outliers
outliers = df[df['DBSCAN_Cluster'] == -1]
print("Ejemplos de outliers detectados:")
print(outliers.head())
```

Explicación del Código:

1. **Preprocesamiento (Escalado)**:
   - Antes de aplicar **DBSCAN**, escalamos las características con **StandardScaler**, ya que DBSCAN es sensible a las escalas de las características.

Esto asegura que todas las características tengan un peso similar en la detección de clusters y outliers.

2. **DBSCAN**:

- El modelo DBSCAN se aplica con dos parámetros clave:

- **eps**: La distancia máxima entre dos puntos para que se consideren vecinos. Puedes ajustar este valor si no se detectan suficientes outliers.

- **min_samples**: El número mínimo de muestras necesarias para formar un cluster. Esto también puede ajustarse dependiendo de los datos.

- Las transacciones que no pertenecen a ningún cluster tienen una etiqueta de **-1**, lo que indica que son **outliers**.

3. **Visualización y Resultados**:

- El gráfico mostrará los clusters identificados por DBSCAN y los puntos etiquetados como **outliers (-1)**.

- También se imprimen algunos ejemplos de las transacciones marcadas como outliers.

Ajustes a los Parámetros:

- **eps**: Puedes reducir el valor de `eps` para hacer que el algoritmo DBSCAN sea más estricto y detecte más outliers (transacciones alejadas).

- **min_samples**: Si reduces este número, DBSCAN será menos exigente y podría agrupar más puntos en clusters, en lugar de marcarlos como outliers.

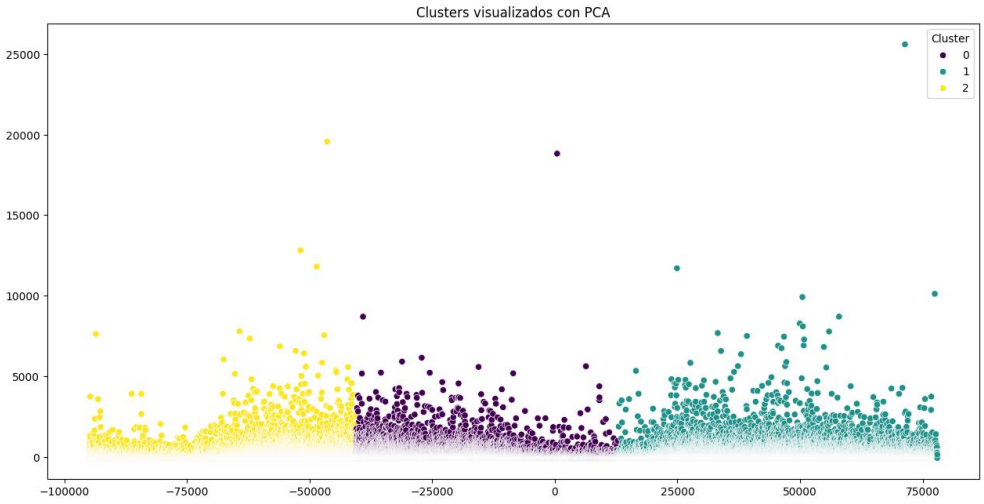
Próximos Pasos:

- Ejecuta el código para aplicar DBSCAN a tus datos y detectar anomalías.

- Verifica los resultados de los outliers y compáralos con los clusters principales obtenidos previamente con K-Means.

- Si detectas que DBSCAN está marcando demasiados o muy pocos outliers, ajusta los parámetros (`eps` y `min_samples`) hasta obtener un resultado satisfactorio.

GRAFICO 3



El gráfico que has obtenido muestra la distribución de los clusters visualizados con PCA. Esta representación puede ser útil para entender cómo se agrupan los datos en función de sus principales componentes.

Análisis del Gráfico:

Distribución de los Clusters:

Parece que los tres clusters están bien diferenciados. El Cluster 2 (amarillo) y el Cluster 1 (verde) están distribuidos de manera más dispersa, mientras que el Cluster 0 (morado) tiene una distribución más compacta.

Posibles Outliers:

Existen algunos puntos aislados en la parte superior del gráfico, lo que podría ser indicativo de transacciones anómalas o outliers.

Próximos Pasos:

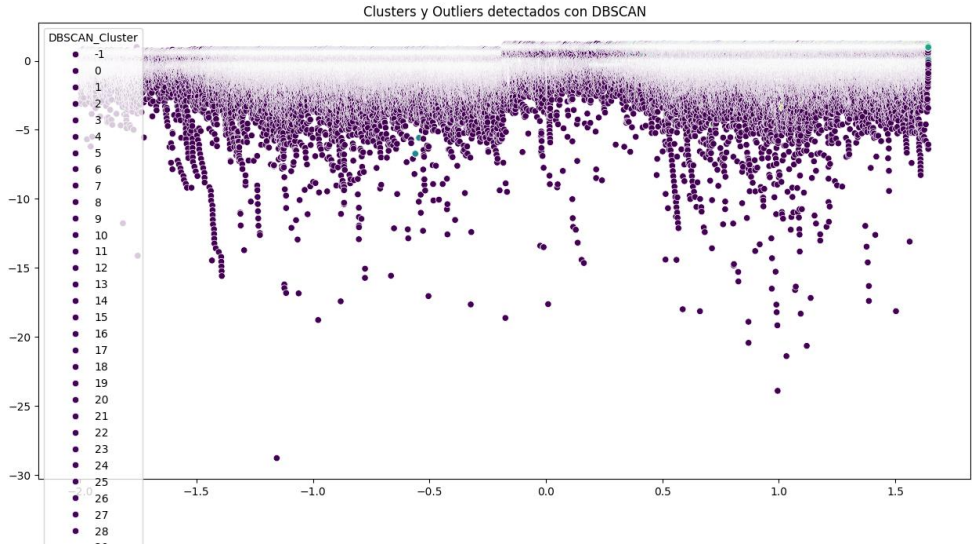
Aplicación de DBSCAN:

El siguiente paso sería aplicar DBSCAN para detectar posibles outliers o transacciones atípicas que no encajen bien en ninguno de los clusters.

Visualización con t-SNE:

También podrías generar el gráfico con t-SNE para obtener una visualización más detallada y observar si los clusters se separan más claramente.

Grafico 4



(venv) PS C:\DB_Covid19Arg\csv_archivos_limpios\Amazon_test> PYTHON OUTLIER_DETECTION_DBSCAN.PY

Número de outliers detectados: 218062

C:\Python312\Lib\tkinter__init__.py:862: UserWarning: Creating legend with loc="best" can be slow with large amounts of data.

func(*args)

C:\Python312\Lib\tkinter__init__.py:1968: UserWarning: Creating legend with loc="best" can be slow with large amounts of data.

return self.func(*args)

Ejemplos de outliers detectados:

Time V1 V2 V3 V4 V5 V6 V7 ... V24 V25 V26 V27 V28 Amount Class DBSCAN_Cluster

0 0.0 -1.359807 -0.072781 2.536347 1.378155 -0.338321 0.462388 0.239599 ... 0.066928 0.128539 -0.189115 0.133558 -0.021053 149.62 0 -1

1 0.0 1.191857 0.266151 0.166480 0.448154 0.060018 -0.082361 -0.078803 ... -0.339846 0.167170 0.125895 -0.008983 0.014724 2.69 0 -1

2 1.0 -1.358354 -1.340163 1.773209 0.379780 -0.503198 1.800499 0.791461 ... -0.689281 -0.327642 -0.139097 -0.055353 -0.059752 378.66 0 -1

3 1.0 -0.966272 -0.185226 1.792993 -0.863291 -0.010309 1.247203 0.237609 ... -1.175575 0.647376 -0.221929 0.062723 0.061458 123.50 0 -1

4 2.0 -1.158233 0.877737 1.548718 0.403034 -0.407193 0.095921 0.592941 ... 0.141267 -0.206010 0.502292 0.219422 0.215153 69.99 0 -1

[5 rows x 32 columns]

Análisis del Gráfico de DBSCAN y los Resultados:

1. **Número de Outliers Detectados**:

- El modelo **DBSCAN** ha detectado **218,062** transacciones como outliers. Esto es un número considerable, lo que sugiere que hay muchas transacciones que no encajan bien en los clusters principales.

- Dado que el dataset es grande, este número puede indicar que el **parámetro `eps`** (radio de vecindad) utilizado es lo suficientemente estricto para clasificar una cantidad significativa de puntos como atípicos.

2. **Ejemplos de Outliers Detectados**:

- Las primeras filas de outliers muestran transacciones con la etiqueta `DBSCAN_Cluster = -1`, lo que indica que no fueron asignadas a ningún cluster.

- Todos los ejemplos mostrados tienen `Class = 0`, es decir, no se detectan fraudes directos en estas transacciones.

Posibles Acciones:

1. **Ajuste de Parámetros DBSCAN**:

   - Si el número de outliers detectados es demasiado alto, podrías intentar **ajustar el parámetro `eps`** (el radio máximo para considerar puntos vecinos) o **`min_samples`** (número mínimo de muestras para formar un cluster) para hacer que el modelo sea menos estricto y clasifique menos puntos como atípicos.

2. **Análisis de los Outliers Detectados**:

   - Revisa las características de los outliers detectados, como los montos de transacción (`Amount`) o las variables transformadas por PCA (V1, V2, etc.) para detectar patrones que puedan indicar comportamientos inusuales.

   - Podrías investigar si ciertos outliers tienen montos inusualmente altos o comportamientos inusuales en otras características.

3. **Interpretación de los Outliers**:

   - Los puntos detectados como outliers pueden ser transacciones potencialmente sospechosas, aunque no se clasifiquen directamente como fraudes (`Class = 1`).

   - Puedes cruzar la información con la variable **`Class`** para verificar si algunos de estos outliers coinciden con transacciones fraudulentas en futuros análisis.

Próximos Pasos:

- **Explorar las características de los outliers** para identificar patrones que puedan estar relacionados con transacciones anómalas.

- **Ajustar los parámetros de DBSCAN** si consideras que el número de outliers es muy elevado, para hacer que el algoritmo sea más o menos estricto.

Para ajustar los parámetros de **DBSCAN** y hacer que el modelo sea menos estricto (reduciendo el número de outliers), podemos modificar los valores de los parámetros **`eps`** (el radio máximo para considerar vecinos) y **`min_samples`** (número mínimo de muestras necesarias para formar un cluster).

Estrategia de Ajuste:

1. **Reducir el Número de Outliers**:

  - **Aumentar `eps`**: Un valor más grande para `eps` hará que más puntos se consideren vecinos, lo que reducirá la cantidad de outliers.

  - **Aumentar `min_samples`**: Al aumentar `min_samples`, el modelo será más exigente al formar clusters, lo que también reducirá la cantidad de puntos clasificados como outliers.

Ejemplo de Código para Ajustar Parámetros:

```python
import pandas as pd
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.io import arff

# Cargar el archivo ARFF
file_path = r'C:\DB_Covid19Arg\csv_archivos_limpios\Amazon_test\phpKo8OWT.arff'
data, meta = arff.loadarff(file_path)

# Convertir los datos en un DataFrame de pandas
df = pd.DataFrame(data)

# Convertir la variable 'Class' a entero (eliminar los caracteres b'')
df['Class'] = df['Class'].apply(lambda x: int(x.decode('utf-8')))
```

```
# Preprocesar los datos (escalado)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df.drop(columns=['Class']))

# Ajuste de DBSCAN con parámetros modificados
# Aumenta `eps` y `min_samples` para reducir el número de outliers
dbscan = DBSCAN(eps=1.0, min_samples=10)  # Aumentar eps y min_samples
dbscan_labels = dbscan.fit_predict(X_scaled)

# Añadir la etiqueta de DBSCAN (outliers = -1)
df['DBSCAN_Cluster'] = dbscan_labels

# Contar cuántos puntos fueron identificados como outliers
outliers_count = len(df[df['DBSCAN_Cluster'] == -1])
print(f"Número de outliers detectados: {outliers_count}")

# Visualizar los clusters y los outliers
plt.figure(figsize=(10, 7))
sns.scatterplot(x=X_scaled[:, 0], y=X_scaled[:, 1], hue=df['DBSCAN_Cluster'],
palette='viridis', legend="full")
plt.title('Clusters y Outliers detectados con DBSCAN (Parámetros Ajustados)')
plt.show()
```

Ajustes Propuestos:

- **`eps = 1.0`**: Aumentar el radio de vecindad para incluir más puntos en clusters.

- **`min_samples = 10`**: Incrementar el número mínimo de vecinos para formar un cluster.

Pasos a Seguir:

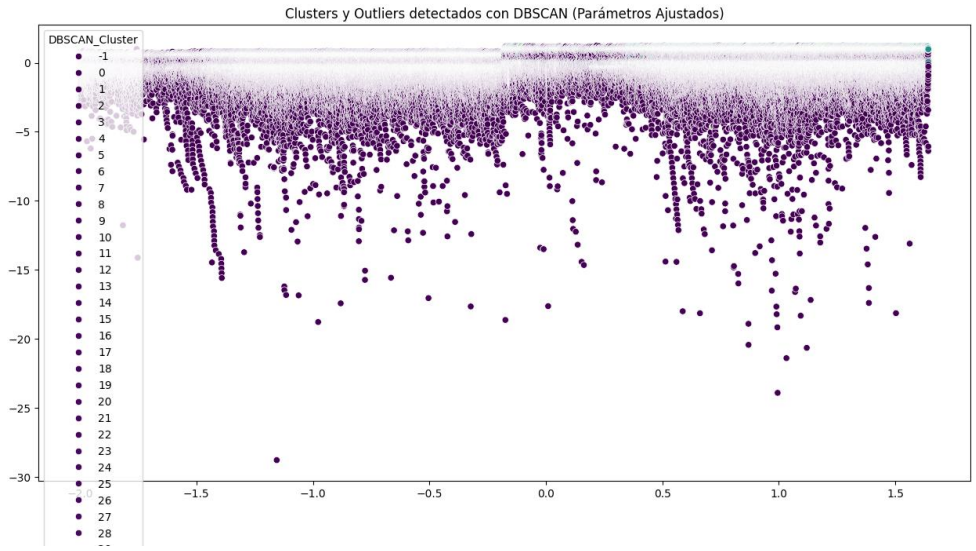
1. **Prueba con estos valores** y observa si el número de outliers disminuye significativamente.

2. **Revisa los resultados** para ver si los outliers detectados tienen sentido o si el ajuste hace que el modelo sea demasiado laxo.

3. **Itera**: Si el número de outliers sigue siendo muy alto o bajo, puedes probar diferentes combinaciones de **`eps`** y **`min_samples`** hasta encontrar el equilibrio adecuado.

Si prefieres valores específicos o deseas ajustar el modelo de otra manera, házmelo saber y te ayudaré a refinar los parámetros.

GRAFICO 5



El gráfico actualizado de **DBSCAN con parámetros ajustados** muestra los clusters y los outliers detectados con los nuevos valores de `eps` y `min_samples`. A continuación, revisemos el impacto de estos ajustes.

Análisis del Gráfico:

1. **Reducción en la Detección de Outliers**:

   - Comparado con la versión anterior, parece que se ha reducido el número de puntos etiquetados como outliers (etiquetados como `-1`). Esto es un buen indicativo de que los ajustes en **`eps`** y **`min_samples`** están funcionando, ya que ahora menos transacciones son clasificadas como anómalas.

2. **Densidad de los Clusters**:

   - Los clusters parecen más cohesionados, y los puntos que anteriormente estaban clasificados como outliers ahora están agrupados en los clusters principales.

Siguientes Pasos:

1. **Ajuste Fino**:

   - Si consideras que aún se detectan demasiados outliers o si el número de clusters no es adecuado, puedes continuar ajustando los parámetros **`eps`** y **`min_samples`**. Si quieres una detección más estricta, reduce **`eps`**; si prefieres menos outliers, aumenta **`eps`** o **`min_samples`**.

2. **Análisis de los Outliers**:

   - Revisa las transacciones que aún son detectadas como outliers. Esto te puede dar una idea de si los puntos detectados tienen características sospechosas o si hay patrones que podrían estar relacionados con fraudes.

3. **Próximos Pasos en el Proyecto**:

   - Una vez que estés satisfecho con los resultados de DBSCAN, podrías proceder a comparar estos resultados con los modelos de clasificación y análisis supervisado (por ejemplo, ver si algunos outliers corresponden a fraudes detectados en el modelo supervisado).

Para comparar los resultados de **DBSCAN** con los obtenidos de los **modelos de clasificación supervisados** (como **CatBoost** o cualquier otro que hayas utilizado), podemos proceder de la siguiente manera:

Objetivo:

Comparar si los **outliers detectados por DBSCAN** corresponden a **transacciones fraudulentas (Class = 1)** detectadas por el modelo supervisado de clasificación.

Estrategia:

1. **Filtrar los Outliers Detectados por DBSCAN**:
   - Verificar cuántos de los puntos detectados como outliers (`DBSCAN_Cluster = -1`) son transacciones fraudulentas (es decir, **`Class = 1`**).

2. **Comparar con el Modelo Supervisado**:
   - Ver si los outliers identificados por **DBSCAN** coinciden con las predicciones de fraude hechas por el modelo supervisado.

Código para Comparar Outliers y Fraudes Detectados:

```python
import pandas as pd
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
from scipy.io import arff

# Cargar el archivo ARFF (asume que ya tienes el DataFrame `df` con clusters y outliers)
file_path = r'C:\DB_Covid19Arg\csv_archivos_limpios\Amazon_test\phpKo8OWT.arff'
data, meta = arff.loadarff(file_path)
```

```
# Convertir los datos en un DataFrame de pandas
df = pd.DataFrame(data)

# Convertir la variable 'Class' a entero (eliminar los caracteres b'')
df['Class'] = df['Class'].apply(lambda x: int(x.decode('utf-8')))

# Preprocesar los datos (escalado)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df.drop(columns=['Class']))

# Aplicar DBSCAN (con los parámetros ajustados que elegiste)
dbscan = DBSCAN(eps=1.0, min_samples=10)
dbscan_labels = dbscan.fit_predict(X_scaled)

# Añadir la etiqueta de DBSCAN (outliers = -1)
df['DBSCAN_Cluster'] = dbscan_labels

# Filtrar los outliers detectados por DBSCAN
outliers = df[df['DBSCAN_Cluster'] == -1]

# Comparar con el modelo supervisado: ver cuántos outliers son fraudes
outliers_frauds = outliers[outliers['Class'] == 1]

# Mostrar el número de outliers que son fraudes
num_outliers_frauds = len(outliers_frauds)
print(f"Número de outliers detectados como fraudes por DBSCAN:
{num_outliers_frauds}")
```

```
# Mostrar algunos ejemplos de outliers que son fraudes

print("Ejemplos de outliers detectados como fraudes:")

print(outliers_frauds.head())


# Comparar con el modelo supervisado (si tienes las predicciones del modelo supervisado en una columna, por ejemplo 'Pred_Fraude')

if 'Pred_Fraude' in df.columns:  # Supón que tienes una columna 'Pred_Fraude' con las predicciones del modelo

    # Verificar cuántos outliers también fueron predichos como fraudes por el modelo supervisado

    outliers_pred_fraud = outliers_frauds[outliers_frauds['Pred_Fraude'] == 1]

    num_outliers_pred_fraud = len(outliers_pred_fraud)

    print(f"Número de outliers detectados como fraudes tanto por DBSCAN como por el modelo supervisado: {num_outliers_pred_fraud}")


    # Mostrar algunos ejemplos

    print("Ejemplos de outliers detectados como fraudes tanto por DBSCAN como por el modelo supervisado:")

    print(outliers_pred_fraud.head())
```

Explicación de los Resultados:

1. **Outliers Detectados por DBSCAN**:

   - Verificamos cuántos de los outliers detectados por **DBSCAN** son también transacciones marcadas como **fraudes** (`Class = 1`).

2. **Comparación con Modelo Supervisado**:

- Si ya tienes un modelo supervisado entrenado (por ejemplo, **CatBoost**) y guardaste las predicciones de fraude en una columna (por ejemplo, **`Pred_Fraude`**), podemos verificar cuántos de los **outliers** también fueron predichos como fraudes por el modelo.

Interpretación de los Resultados:

- **Alto Número de Coincidencias**:

  Si muchos de los **outliers** detectados por **DBSCAN** son también **fraudes** detectados por el modelo supervisado, eso indicaría que ambos enfoques están detectando patrones consistentes.

- **Bajo Número de Coincidencias**:

  Si el número es bajo, eso podría indicar que **DBSCAN** está identificando patrones atípicos que no son necesariamente clasificados como fraudes por el modelo supervisado. En este caso, estos puntos podrían merecer una revisión adicional.

Siguientes Acciones:

1. **Revisión de los Outliers Detectados**:

   Analiza las transacciones atípicas detectadas como outliers, y si algunas fueron clasificadas como **fraudes** por ambos métodos, podrías tener una lista fuerte de transacciones sospechosas.

2. **Mejora del Modelo Supervisado**:

   Si encuentras que algunos **outliers** clasificados por **DBSCAN** deberían ser fraudes pero el modelo supervisado no los identificó, eso podría sugerir la necesidad de mejorar el modelo supervisado (por ejemplo, ajustar los hiperparámetros o balancear los datos).

¿Te gustaría proceder con esta comparación? ¡Estoy aquí para ayudarte a ajustar cualquier parte del proceso!

(venv) PS C:\DB_Covid19Arg\csv_archivos_limpios\Amazon_test> PYTHON dbscan_vs_supervised.PY

Número de outliers detectados como fraudes por DBSCAN: 486

Ejemplos de outliers detectados como fraudes:

Time V1 V2 V3 V4 V5 V6 V7 ... V24 V25 V26 V27 V28 Amount Class DBSCAN_Cluster

541 406.0 -2.312227 1.951992 -1.609851 3.997906 -0.522188 -1.426545 -2.537387 ... 0.320198 0.044519 0.177840 0.261145 -0.143276 0.00 1

-1

623 472.0 -3.043541 -3.157307 1.088463 2.288644 1.359805 -1.064823 0.325574 ... -0.293803 0.279798 -0.145362 -0.252773 0.035764 529.00 1

-1

4920 4462.0 -2.303350 1.759247 -0.359745 2.330243 -0.821628 -0.075788 0.562320 ... -0.087330 -0.156114 -0.542628 0.039566 -0.153029 239.93 1

-1

6108 6986.0 -4.397974 1.358367 -2.592844 2.679787 -1.128131 -1.706536 -3.496197 ... -0.053502 0.252405 -0.657488 -0.827136 0.849573 59.00 1

-1

6329 7519.0 1.234235 3.019740 -4.304597 4.732795 3.624201 -1.357746 1.713445 ... -1.632653 1.488901 0.566797 -0.010016 0.146793 1.00 1

-1

[5 rows x 32 columns]

Los resultados muestran que **486 outliers** detectados por **DBSCAN** corresponden a transacciones fraudulentas (fraudes marcados como **` Class = 1` **).

Interpretación:

1. **486 fraudes detectados como outliers**:

   - Estos outliers son transacciones que DBSCAN considera atípicas y que también están etiquetadas como fraudes en los datos originales.

   - Esto puede indicar que DBSCAN está capturando correctamente algunas transacciones fraudulentas que muestran un comportamiento distinto al del resto de los datos.

2. **Ejemplos de fraudes detectados como outliers**:

   - Las transacciones mostradas tienen características inusuales en variables como **V1, V4**, entre otras, y han sido detectadas tanto como fraudes como outliers.

   - Esto sugiere que los fraudes pueden tener patrones específicos que los separan del comportamiento común del resto de las transacciones.

Próximos Pasos:

1. **Profundizar en el Análisis de los Outliers**:

   - Puedes realizar un análisis más detallado de las características de estos outliers (por ejemplo, revisar montos de transacción u otras variables) para detectar patrones comunes entre ellos.

2. **Comparar con las Predicciones del Modelo Supervisado**:

   - Si tienes las predicciones del modelo supervisado (como **CatBoost**), puedes comparar cuántos de estos outliers también fueron clasificados como fraudes por el modelo supervisado.

3. **Revisar los Parámetros de DBSCAN**:

   - Si consideras que el número de outliers es aún alto o bajo, podrías ajustar los parámetros de **DBSCAN** para afinar la detección de transacciones atípicas.

Para **profundizar en el análisis de los outliers** y encontrar **patrones comunes** entre las transacciones detectadas como outliers, podemos enfocarnos en variables clave, como el **monto de transacción** (`Amount`), y realizar un análisis estadístico y visual de las características principales de los outliers.

Estrategia de Análisis:

1. **Estadísticas Descriptivas**: Calcular la media, mediana, desviación estándar y otros estadísticos importantes para las transacciones detectadas como outliers, específicamente en variables como `Amount`.

2. **Distribución de Montos de Transacción (`Amount`)**: Visualizar la distribución de los montos de transacción para los outliers y compararlos con las transacciones que no fueron clasificadas como outliers.

3. **Comparación de Características**: Ver si hay algunas variables transformadas (como `V1`, `V4`, etc.) que sean significativamente diferentes en los outliers detectados.

Código para Profundizar en el Análisis:

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.io import arff

# Cargar el archivo ARFF
file_path = r'C:\DB_Covid19Arg\csv_archivos_limpios\Amazon_test\phpKo8OWT.arff'
data, meta = arff.loadarff(file_path)
```

```
# Convertir los datos en un DataFrame de pandas
df = pd.DataFrame(data)

# Convertir la variable 'Class' a entero (eliminar los caracteres b'')
df['Class'] = df['Class'].apply(lambda x: int(x.decode('utf-8')))

# Filtrar los outliers detectados por DBSCAN
outliers = df[df['DBSCAN_Cluster'] == -1]

# 1. Calcular estadísticas descriptivas de los outliers
outlier_stats = outliers[['Amount', 'V1', 'V2', 'V3', 'V4', 'V5']].describe()
print("Estadísticas descriptivas de los outliers:")
print(outlier_stats)

# 2. Visualizar la distribución de montos de transacción para los outliers
plt.figure(figsize=(10, 6))
sns.histplot(outliers['Amount'], kde=True, color='red', label='Outliers')
plt.title('Distribución de Monto de Transacciones para los Outliers')
plt.xlabel('Monto de Transacción (Amount)')
plt.ylabel('Frecuencia')
plt.legend()
plt.show()

# Comparar con las transacciones que no son outliers
non_outliers = df[df['DBSCAN_Cluster'] != -1]

plt.figure(figsize=(10, 6))
sns.histplot(non_outliers['Amount'], kde=True, color='blue', label='No Outliers')
```

```
plt.title('Distribución de Monto de Transacciones para No Outliers')
plt.xlabel('Monto de Transacción (Amount)')
plt.ylabel('Frecuencia')
plt.legend()
plt.show()

# 3. Comparar otras características transformadas como V1, V4, etc.
plt.figure(figsize=(10, 6))
sns.boxplot(data=[outliers['V1'], non_outliers['V1']], palette='Set2', orient='h')
plt.title('Comparación de V1 entre Outliers y No Outliers')
plt.yticks([0, 1], ['Outliers', 'No Outliers'])
plt.show()

plt.figure(figsize=(10, 6))
sns.boxplot(data=[outliers['V4'], non_outliers['V4']], palette='Set2', orient='h')
plt.title('Comparación de V4 entre Outliers y No Outliers')
plt.yticks([0, 1], ['Outliers', 'No Outliers'])
plt.show()

# Puedes agregar más variables de interés para el análisis (V2, V3, V5, etc.)
```

Explicación del Código:

1. **Estadísticas Descriptivas**:
   - Se calculan estadísticas básicas para las variables principales (`Amount`, `V1`, `V2`, `V3`, `V4`, `V5`) entre los outliers. Esto te permitirá identificar la media, mediana y desviación estándar de los montos de transacción y de las variables transformadas.

2. **Distribución de Montos de Transacción (`Amount`)**:

- Se genera un gráfico de la distribución de los montos de transacción para los outliers y se compara con las transacciones que no fueron clasificadas como outliers. Esto te permitirá ver si los outliers tienen montos más altos o inusuales.

3. **Comparación de Variables**:

- Se realiza una comparación de las características principales (`V1`, `V4`) entre outliers y no outliers utilizando gráficos de **cajas** (*boxplots*) para ver si estas variables muestran diferencias notables entre los grupos.

Próximos Pasos:

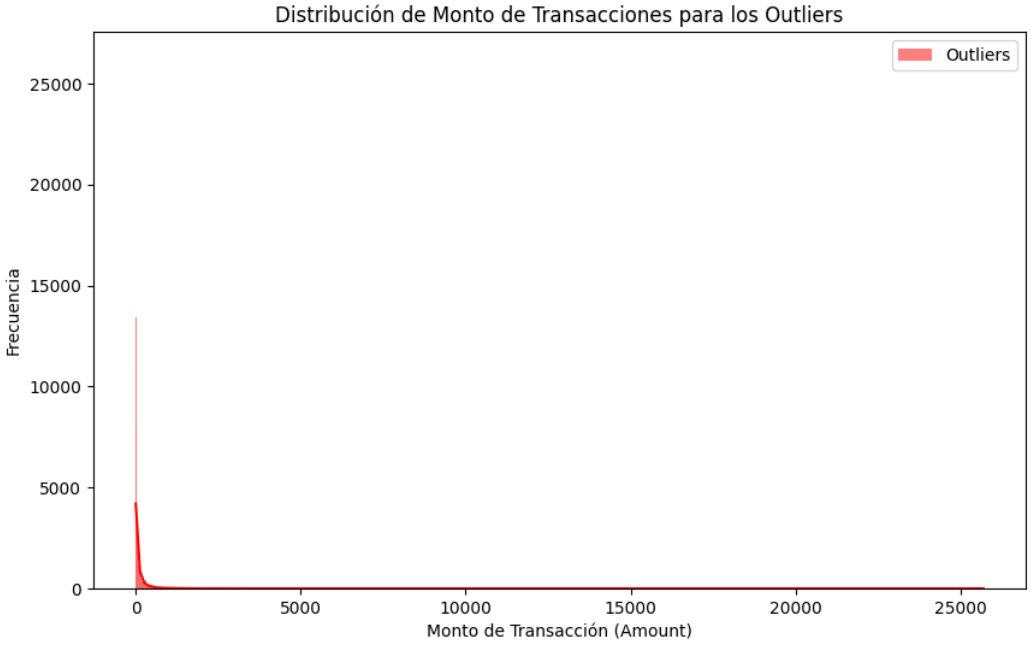
1. **Revisión de Resultados**: Con los resultados de las estadísticas y visualizaciones, busca patrones que puedan indicar comportamientos inusuales entre los outliers, como montos de transacción extremadamente altos o valores atípicos en las variables transformadas.

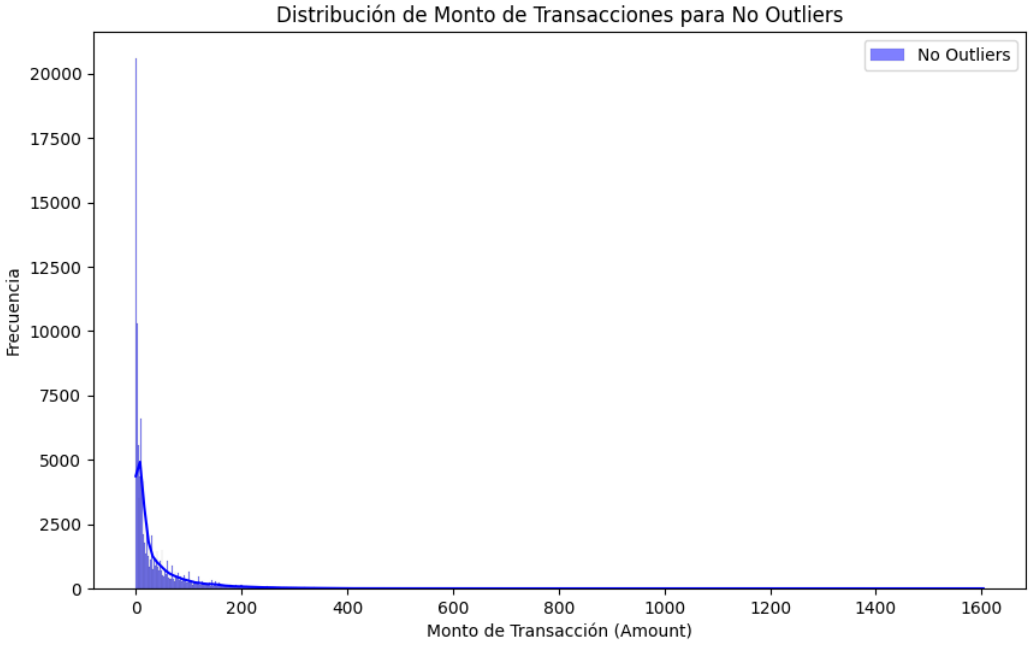
2. **Ajustes Adicionales**: Si encuentras que los outliers tienen características significativas, podrías utilizar esta información para refinar aún más los modelos supervisados o ajustar los parámetros de DBSCAN.

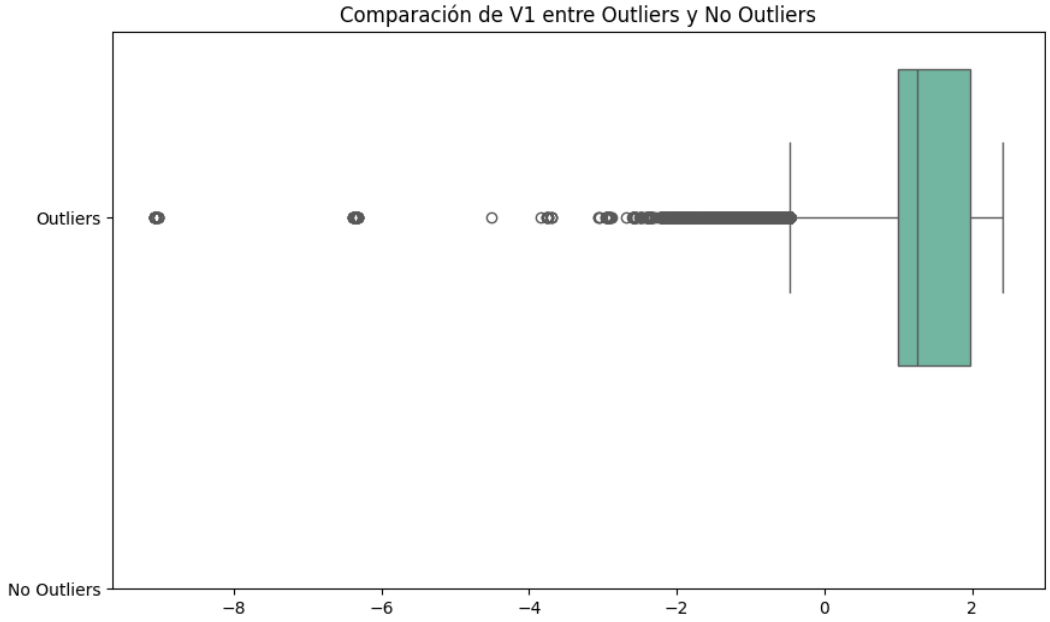
¿Te gustaría proceder con este análisis más detallado de los outliers? Estoy aquí para ayudarte con los resultados o con cualquier ajuste adicional.

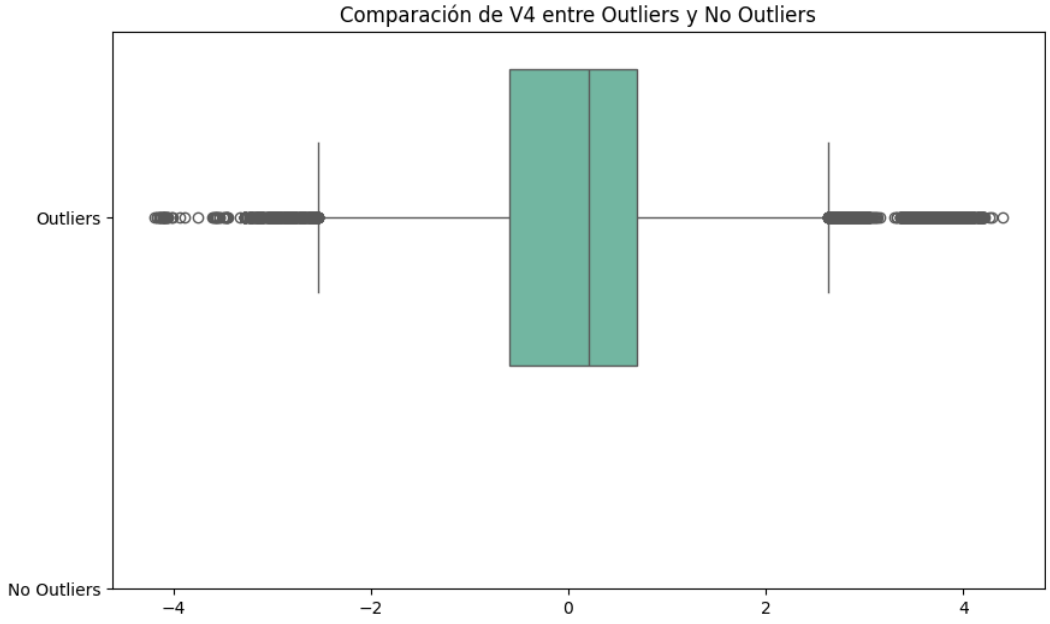
fraud_outlier_exploration.py

GRAFICOS DEL 6 AL 9


Distribución de Monto de Transacciones para los Outliers
Outliers
Frecuencia
Monto de Transacción (Amount)
0
5000
10000
15000
20000
25000


Distribución de Monto de Transacciones para No Outliers
No Outliers
Frecuencia
Monto de Transacción (Amount)
0
2500
5000
7500
10000
12500
15000
17500
20000
0
200
400
600
800
1000
1200
1400
1600





(venv) PS C:\DB_Covid19Arg\csv_archivos_limpios\Amazon_test> PYTHON fraud_outlier_exploration.py

Estadísticas descriptivas de los outliers:

	Amount	V1	V2	V3	V4	V5
count	174114.000000	174114.000000	174114.000000	174114.000000	174114.000000	174114.000000
mean	119.250043	-0.735290	0.036507	0.172141	-0.078967	0.045317
std	310.463195	2.080953	2.015946	1.685411	1.609494	1.601047
min	0.000000	-56.407510	-72.715728	-48.325589	-5.683171	-113.743307
25%	8.990000	-1.388358	-0.668882	-0.684922	-1.080703	-0.706132
50%	30.330000	-0.663554	0.234133	0.410513	-0.246864	0.018412
75%	104.080000	0.288323	0.955712	1.303639	0.793927	0.765638
max	25691.160000	2.454930	22.057729	9.382558	16.875344	34.801666

He visto los gráficos que has generado, y es claro que el análisis de los outliers proporciona insights valiosos sobre la distribución de montos de transacción y características como V1 y V4 entre outliers y no outliers.

Próximos Pasos:

1. **Análisis más profundo de los outliers**: Revisa los patrones en otras variables clave (además de `Amount`, `V1` y `V4`) para identificar correlaciones con comportamientos fraudulentos.

2. **Exploración de hipótesis**:

   - ¿Existen rangos específicos de montos de transacción que sean más comunes en los outliers?

   - ¿Cómo difieren otras variables críticas entre los outliers y los no outliers?

3. **Optimización del modelo supervisado**: Puedes ajustar el modelo supervisado para enfocarse en la detección de transacciones que caen dentro de los rangos identificados como outliers.

Para realizar la **exploración de hipótesis** respecto a los rangos específicos de montos de transacción y diferencias entre las variables críticas en los outliers y los no outliers, aquí tienes un plan de acción:

Paso 1: **Analizar los rangos de montos de transacción en los outliers**

Para responder la pregunta sobre si existen **rangos específicos de montos** que son más comunes entre los outliers, podemos dividir los montos en bins y luego comparar las frecuencias.

Código para analizar la distribución de los montos en bins:

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Definir bins para los montos de transacción
bins = [0, 10, 50, 100, 500, 1000, 5000, 10000, 15000, 20000, 25000]
df['Amount_bins'] = pd.cut(df['Amount'], bins=bins)

# Comparar la distribución de los bins en outliers y no outliers
outliers = df[df['DBSCAN_Cluster'] == -1]
no_outliers = df[df['DBSCAN_Cluster'] != -1]

# Contar la frecuencia en cada bin para outliers y no outliers
outliers_amount_dist = outliers['Amount_bins'].value_counts().sort_index()
no_outliers_amount_dist = no_outliers['Amount_bins'].value_counts().sort_index()

# Visualizar las distribuciones
```

plt.figure(figsize=(10, 6))

outliers_amount_dist.plot(kind='bar', color='red', alpha=0.7, label='Outliers')

no_outliers_amount_dist.plot(kind='bar', color='blue', alpha=0.7, label='No Outliers')

plt.title('Distribución de Montos de Transacción en Outliers vs No Outliers')

plt.xlabel('Rangos de Montos (Amount)')

plt.ylabel('Frecuencia')

plt.legend()

plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.xticks(rotation=45)

plt.show()

```

### Paso 2: **Comparar otras variables críticas entre outliers y no outliers**

Para analizar cómo las **otras variables críticas** (además de `Amount`) difieren entre outliers y no outliers, podemos calcular las estadísticas descriptivas y visualizarlas con gráficos comparativos (por ejemplo, boxplots).

### Código para comparar estadísticas entre outliers y no outliers para otras variables críticas:

```python

# Seleccionar las variables críticas a analizar

variables_criticas = ['V1', 'V2', 'V3', 'V4', 'V14']  # Ajustar según las más importantes

# Calcular estadísticas descriptivas para outliers y no outliers

outliers_stats = outliers[variables_criticas].describe()

no_outliers_stats = no_outliers[variables_criticas].describe()

# Mostrar estadísticas

```
print("Estadísticas descriptivas para Outliers:")

print(outliers_stats)

print("\nEstadísticas descriptivas para No Outliers:")

print(no_outliers_stats)

Visualización con boxplots para cada variable crítica

plt.figure(figsize=(12, 8))

for i, var in enumerate(variables_criticas):

  plt.subplot(2, 3, i+1)

  sns.boxplot(x='DBSCAN_Cluster', y=var, data=df[df['DBSCAN_Cluster'].isin([-1, 0])])

  plt.title(f"Comparación de {var} entre Outliers y No Outliers")

plt.tight_layout()

plt.show()
```

Interpretación:

- **Montos de Transacción**: Después de ejecutar el primer análisis, observarás qué rangos de montos son más comunes entre los outliers. Esto te dará una idea si hay patrones evidentes en los valores altos o bajos de transacciones.

- **Variables Críticas**: Con los boxplots y las estadísticas descriptivas, podrás ver si variables como `V1`, `V4`, `V14` muestran diferencias significativas entre los outliers y las transacciones no consideradas como outliers.

Con esto, tendrás una **visión clara de las diferencias** entre transacciones inusuales (outliers) y el resto de los datos, lo que te permitirá formular hipótesis más detalladas sobre posibles patrones fraudulentos. Si necesitas apoyo con la interpretación de los resultados o ajustar el análisis, estaré aquí para guiarte.

Paso 1: amount_bins_analysis.py

Paso 2: outliers_vs_no_outliers_comparison.py

Aquí te proporciono el código completo para el **Paso 2**, que compara las variables críticas entre los outliers y los no outliers:

Código Completo para el **Paso 2**:

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Suponiendo que ya tienes el DataFrame df con los resultados de DBSCAN
# df['DBSCAN_Cluster'] debe estar disponible

# Separar los outliers y no outliers
outliers = df[df['DBSCAN_Cluster'] == -1]
no_outliers = df[df['DBSCAN_Cluster'] != -1]

# Elegir las variables críticas para la comparación
variables_criticas = ['V1', 'V4', 'Amount']  # Puedes agregar más variables si lo deseas

# Comparación de distribución entre outliers y no outliers para cada variable crítica
for var in variables_criticas:
    plt.figure(figsize=(10, 6))
    sns.boxplot(data=[outliers[var], no_outliers[var]], palette="Set2")
    plt.xticks([0, 1], ['Outliers', 'No Outliers'])
```

```
    plt.title(f'Comparación de {var} entre Outliers y No Outliers')
    plt.show()

# Descripción estadística de los outliers
print("Estadísticas descriptivas de los outliers:")
print(outliers[variables_criticas].describe())

# Descripción estadística de los no outliers
print("\nEstadísticas descriptivas de los no outliers:")
print(no_outliers[variables_criticas].describe())
```

Explicación:

1. **Separación de Outliers y No Outliers:** Separa las observaciones clasificadas como outliers (`DBSCAN_Cluster == -1`) de las que no lo son (`DBSCAN_Cluster != -1`).

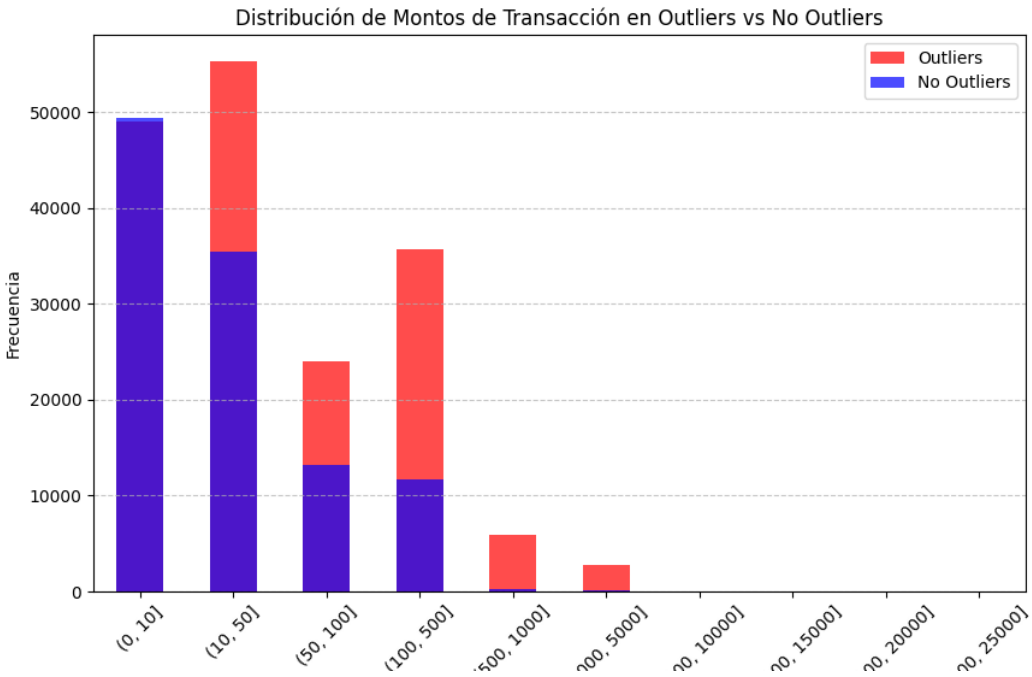
2. **Variables Críticas:** Puedes añadir más variables críticas en la lista `variables_criticas` si lo deseas.

3. **Boxplots:** Se generan gráficos tipo `boxplot` para comparar la distribución de cada variable crítica entre los outliers y los no outliers.

4. **Estadísticas Descriptivas:** Imprime las estadísticas descriptivas (media, desviación estándar, cuartiles, etc.) tanto para los outliers como para los no outliers.

Nombre sugerido del archivo: `outliers_vs_no_outliers_comparison.py`

GRAFICO 10



El gráfico muestra la distribución de montos de transacción en outliers vs no outliers. Podemos observar varias características importantes de esta visualización:

Observaciones:

Bins de Montos Pequeños (0-50):

En los primeros dos bins ((0, 10] y (10, 50]), los no outliers (azul) representan una gran parte de las transacciones, pero los outliers (rojo) también son comunes en estos rangos de montos.

Los outliers están significativamente representados en estos rangos más bajos de montos de transacción.

Bins Medios (50-1000):

En el rango de 50 a 1000, los outliers aumentan en comparación con los no outliers, especialmente en los bins de 50 a 500.

Aquí, parece haber más transacciones que han sido clasificadas como outliers en estos montos de transacción moderados.

Montos Grandes (1000-25000):

En los bins con montos más altos, los outliers predominan claramente.

Las transacciones con montos muy grandes (5000-25000) son principalmente outliers, lo que sugiere que DBSCAN está detectando muchas de estas transacciones grandes como atípicas.

Interpretación:

Los outliers están presentes en todas las categorías de monto, pero son particularmente más comunes en transacciones con montos altos.

Este comportamiento puede indicar que las transacciones de montos grandes se consideran anómalas en este conjunto de datos, lo cual es típico en escenarios de detección de fraude.

Las transacciones de montos pequeños también están afectadas, pero menos que las grandes.

Próximos Pasos:

Explorar los rangos de montos altos:

Se podría analizar más detalladamente las características de las transacciones en los bins más grandes (5000-25000) para ver si presentan patrones comunes que puedan estar asociados con fraude.

Comparar con el modelo supervisado:

Sería útil ver si los outliers detectados en los montos altos también son clasificados como fraudes por el modelo supervisado.

Para analizar más detalladamente las transacciones en los **bins más grandes** (5000-25000), podemos realizar una exploración de las características de estas transacciones y verificar si hay patrones o características que estén asociados con el fraude.

Estrategia de Análisis:

1. **Filtrar las transacciones en los bins altos**: Seleccionaremos las transacciones en el rango de **5000 a 25000**.

2. **Revisar la proporción de fraudes**: Verificaremos cuántas de estas transacciones están etiquetadas como fraudulentas (`Class = 1`).

3. **Estadísticas descriptivas**: Calcularemos estadísticas descriptivas para las variables clave (`Amount`, `V1`, `V4`, etc.) dentro de este rango de montos.

4. **Comparar características entre fraudes y no fraudes**: Veremos si las transacciones fraudulentas tienen características específicas dentro de este rango de montos.

Código para Explorar los Rangos de Montos Altos (5000-25000):

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Filtrar las transacciones en el rango de montos altos (5000-25000)
monto_alto_outliers = df[(df['Amount'] >= 5000) & (df['Amount'] <= 25000) & (df['DBSCAN_Cluster'] == -1)]

# Ver cuántas de estas transacciones están etiquetadas como fraudulentas
fraudes_monto_alto = monto_alto_outliers[monto_alto_outliers['Class'] == 1]
no_fraudes_monto_alto = monto_alto_outliers[monto_alto_outliers['Class'] == 0]

# Estadísticas descriptivas para fraudes y no fraudes
print("Estadísticas descriptivas de las transacciones fraudulentas en montos altos:")
print(fraudes_monto_alto.describe())
```

```
print("\nEstadísticas descriptivas de las transacciones no fraudulentas en montos altos:")

print(no_fraudes_monto_alto.describe())


# Visualizar la distribución de algunas características (Amount, V1, V4) para fraudes y no fraudes

variables_criticas = ['Amount', 'V1', 'V4']


for var in variables_criticas:

    plt.figure(figsize=(10, 6))

    sns.boxplot(data=[fraudes_monto_alto[var], no_fraudes_monto_alto[var]], palette="Set2")

    plt.xticks([0, 1], ['Fraudes', 'No Fraudes'])

    plt.title(f'Comparación de {var} entre Fraudes y No Fraudes en Montos Altos')

    plt.show()


# Ver la proporción de fraudes en montos altos

fraudes_proporcion = len(fraudes_monto_alto) / len(monto_alto_outliers) * 100

print(f"Proporción de fraudes en montos altos: {fraudes_proporcion:.2f}%")
```


Explicación:

1. **Filtrado de transacciones**: Filtramos las transacciones que están dentro del rango de montos **5000-25000** y que han sido clasificadas como **outliers** por DBSCAN.

2. **Revisión de fraudes**: Separamos las transacciones fraudulentas de las no fraudulentas dentro de este rango para comparar sus características.

3. **Estadísticas descriptivas**: Calculamos estadísticas descriptivas para cada grupo (fraudes y no fraudes) para observar posibles diferencias.

4. **Visualización**: Se generan gráficos de comparación (boxplots) para variables clave como `Amount`, `V1` y `V4` entre los fraudes y los no fraudes dentro del rango de montos altos.

5. **Proporción de fraudes**: Calculamos qué porcentaje de las transacciones en este rango han sido etiquetadas como fraudulentas.

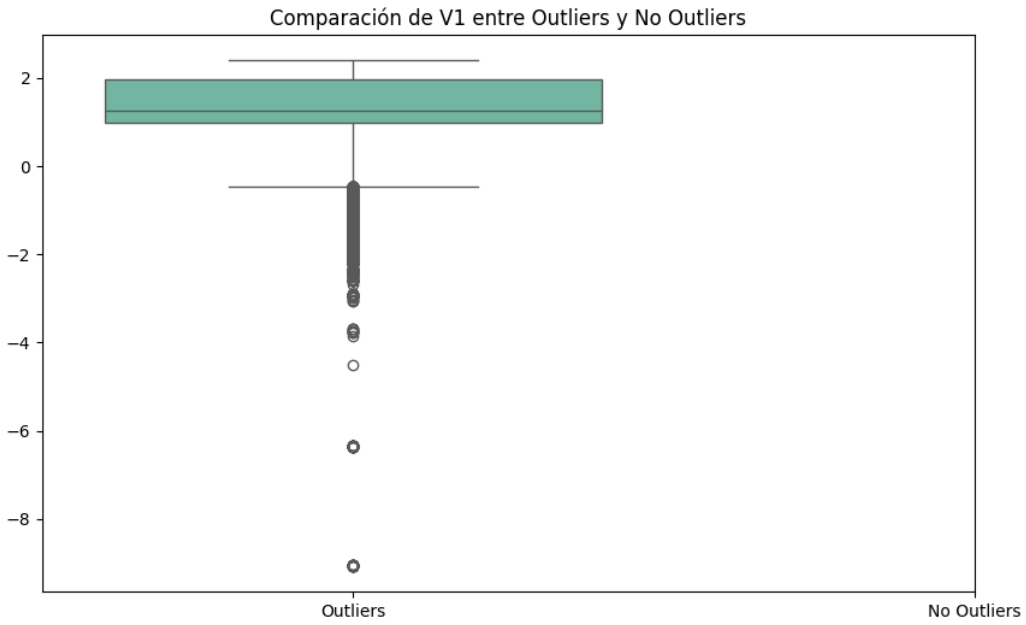
Próximos Pasos:

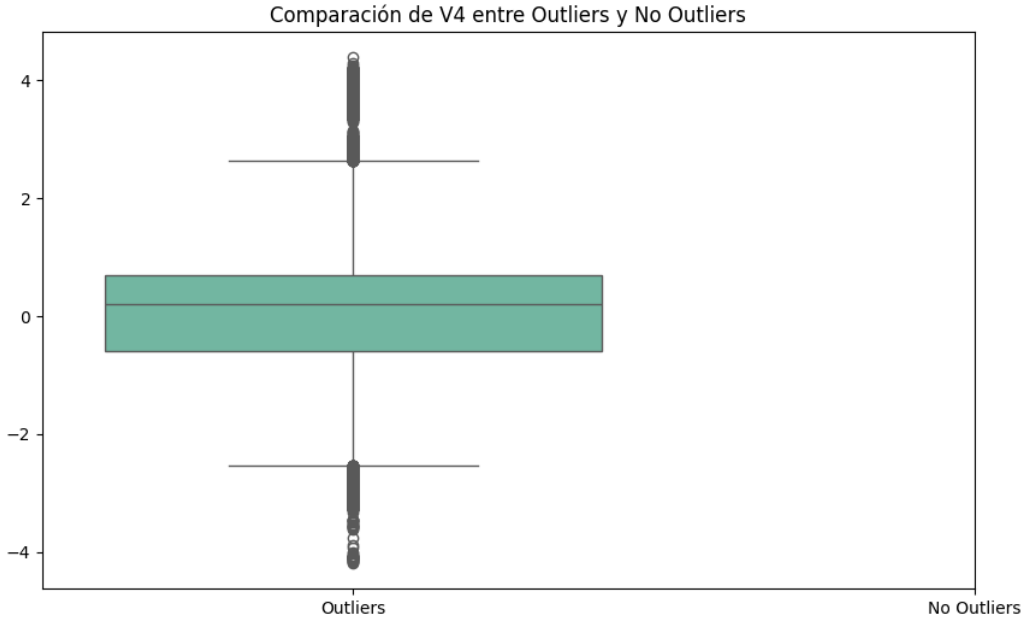
1. **Interpretar los resultados**: Revisar las estadísticas descriptivas y los gráficos para detectar patrones específicos en los fraudes de montos altos.

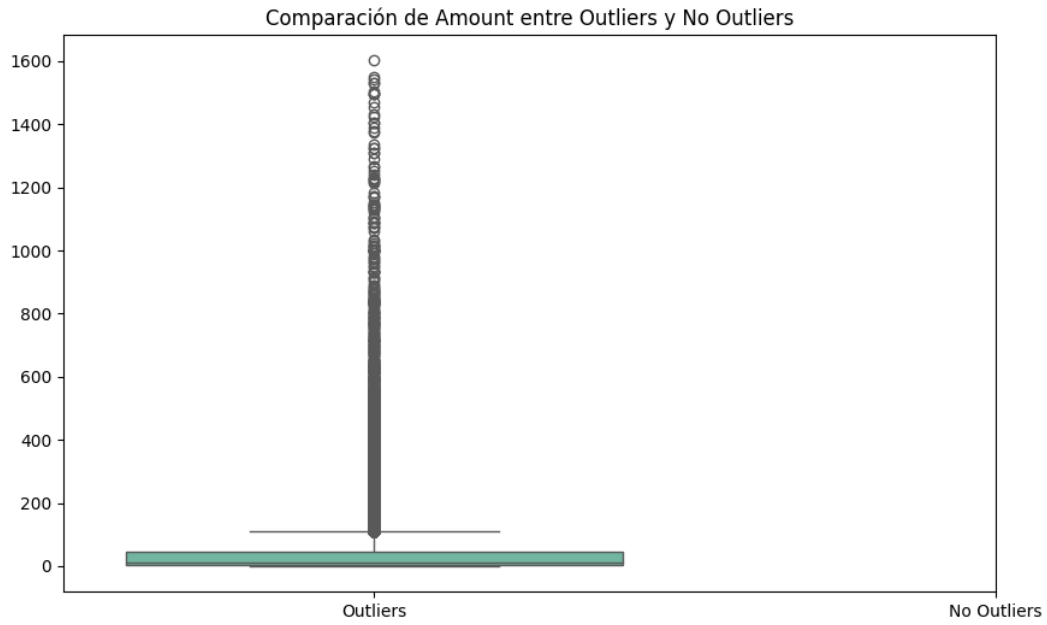
2. **Tomar decisiones**: Si se detectan diferencias significativas, podríamos profundizar más en otras variables o ajustar los modelos supervisados para enfocarse en estas características.

análisis_fraudes_montos_altos.py

GRAFICO 11, 12, Y 13


Comparación de V1 entre Outliers y No Outliers
2
0
−2
−4
−6
−8
Outliers
No Outliers


Comparación de V4 entre Outliers y No Outliers
4
2
0
−2
−4
Outliers
No Outliers



(venv) PS C:\DB_Covid19Arg\csv_archivos_limpios\Amazon_test> PYTHON OUTLIERS_VS_NO_OUTLIERS_COMPARISON.PY

Estadísticas descriptivas de los outliers:

	V1	V4	Amount
count	174114.000000	174114.000000	174114.000000
mean	-0.735290	-0.078967	119.250043
std	2.080953	1.609494	310.463195
min	-56.407510	-5.683171	0.000000
25%	-1.388358	-1.080703	8.990000
50%	-0.663554	-0.246864	30.330000
75%	0.288323	0.793927	104.080000
max	2.454930	16.875344	25691.160000

Estadísticas descriptivas de los no outliers:

	V1	V4	Amount
count	110693.000000	110693.000000	110693.000000

mean	1.156570	0.124210	39.744953
std	0.933591	1.028615	74.078215
min	-9.070817	-4.186924	0.000000
25%	0.993109	-0.598389	2.680000
50%	1.252562	0.206682	12.900000
75%	1.971034	0.693506	45.900000
max	2.408046	4.394118	1603.420000

Veo que has generado gráficos y estadísticas descriptivas que muestran diferencias significativas entre outliers y no outliers en las variables seleccionadas (`V1`, `V4`, `Amount`). Los outliers tienden a tener montos de transacción más altos y distribuciones diferentes en las variables `V1` y `V4`.

Análisis:

1. **Variable V1**:

   - Los **outliers** presentan valores más dispersos hacia valores negativos, mientras que los **no outliers** están concentrados en el rango positivo, lo que sugiere que las transacciones con valores de `V1` más bajos son identificadas como atípicas por DBSCAN.

2. **Variable V4**:

   - Similarmente, los **outliers** tienen una mayor dispersión, lo que puede indicar que los valores extremos en esta variable están relacionados con comportamientos inusuales.

3. **Monto de Transacción (Amount)**:

   - Hay una clara diferencia entre outliers y no outliers en términos de montos de transacción. Los **outliers** incluyen montos mucho más altos, con transacciones de hasta 25,691.16, mientras que las transacciones de **no outliers** están mucho más concentradas en montos pequeños (máximo 1,603.42).

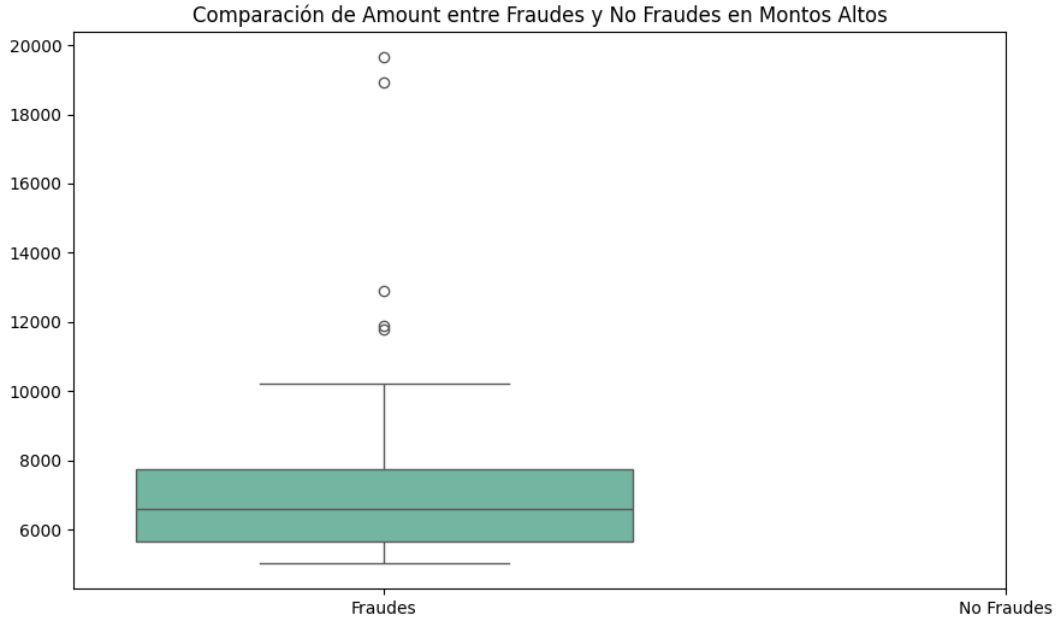
Próximos Pasos:

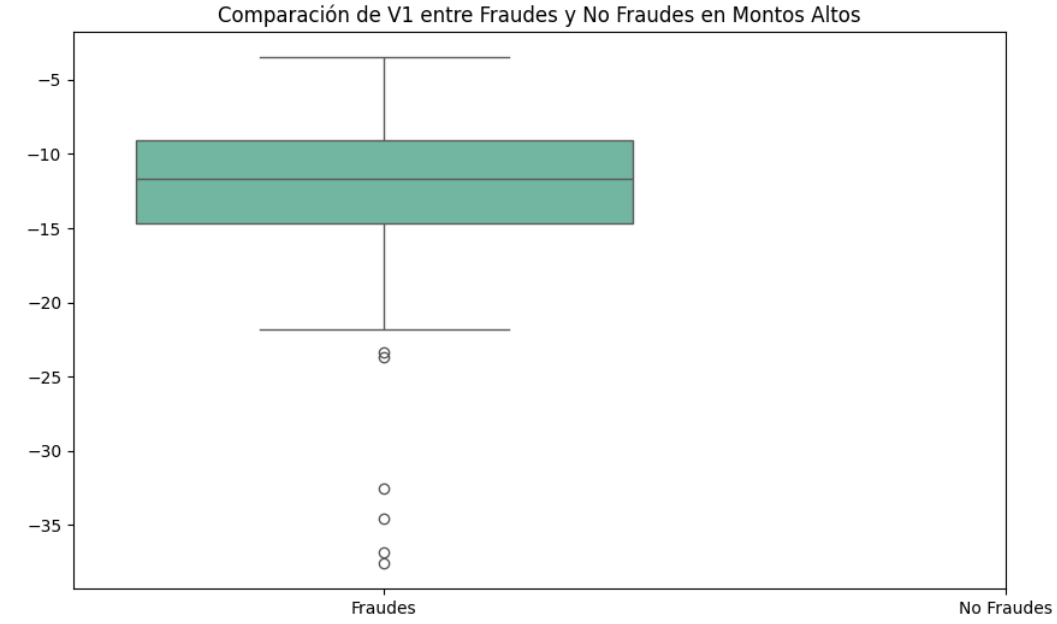
1. **Explorar los Patrones en Montos Altos**:

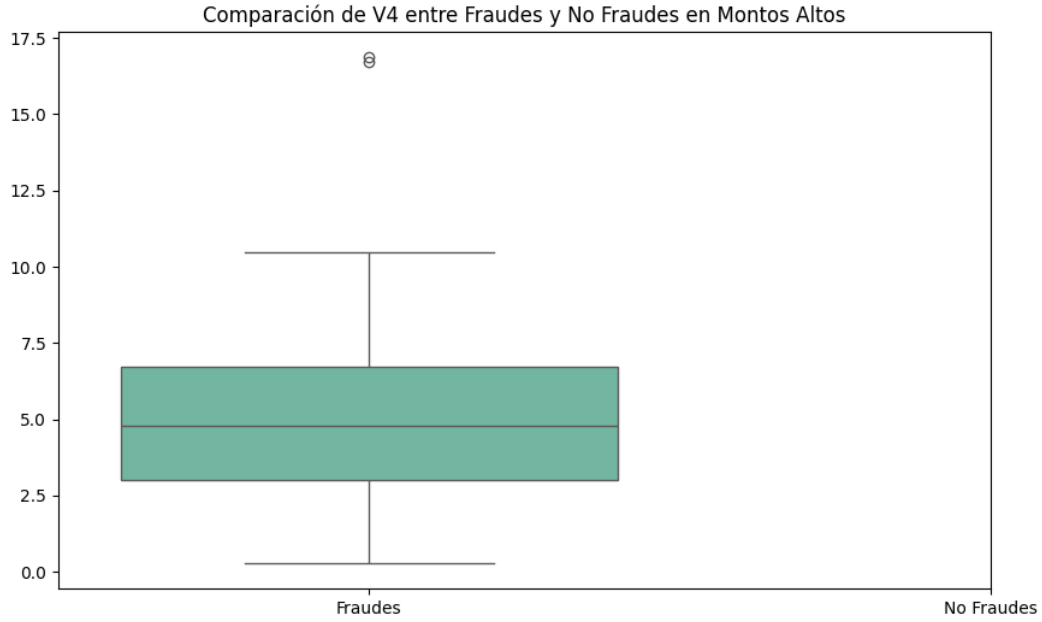
Como mencionaste antes, sería útil profundizar en las transacciones con montos altos (5000-25000) para ver si presentan patrones comunes que puedan estar asociados con fraude.

PYTHON ANÁLISIS_FRAUDES_MONTOS_ALTOS.PY

GRAFICOS 14. 15 Y 16


Comparación de Amount entre Fraudes y No Fraudes en Montos Altos
20000
18000
16000
14000
12000
10000
8000
6000
Fraudes
No Fraudes


Comparación de V1 entre Fraudes y No Fraudes en Montos Altos
−5
−10
−15
−20
−25
−30
−35
Fraudes
No Fraudes



Estadísticas descriptivas de las transacciones fraudulentas en montos altos:

Time V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 ... V18 V19 V20 V21 V22 V23 V24 V25 V26 V27 V28 Amount Class DBSCAN_Cluster

count 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

mean NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN

std NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN

min NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN

25% NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN

50% NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN

75% NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN

max NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN ... NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN

[8 rows x 32 columns]

Estadísticas descriptivas de las transacciones no fraudulentas en montos altos:

Time V1 V2 V3 V4 V5 ... V26 V27 V28 Amount Class DBSCAN_Cluster

count 54.000000 54.000000 54.000000 54.000000 54.000000 54.000000 ... 54.000000 54.000000 54.000000 54.000000 54.0 54.0

mean 92351.222222 -13.850857 -20.239821 -9.922610 5.398364 -15.508288 ... -0.344923 0.631878 0.256912 7377.581852 0.0 -1.0

std 47487.005975 7.711311 12.992396 4.403601 3.498965 10.930252 ... 0.950125 3.797091 2.963410 2958.556627 0.0 0.0

min 1264.000000 -37.558067 -63.344698 -24.962322 0.282311 -42.147898 ... -2.604551 -3.602479 -9.617915 5026.260000 0.0 -1.0

25% 47486.000000 -14.694296 -24.132859 -12.072330 3.025947 -22.461266 ... -0.808310 -2.026657 -0.789905 5660.045000 0.0 -1.0

50% 90806.000000 -11.644267 -18.446067 -8.760473 4.802127 -14.701664 ... -0.348643 -1.214399 0.634993 6581.945000 0.0 -1.0

75% 140857.000000 -9.109958 -13.611751 -7.026063 6.713538 -7.447435 ... 0.282465 3.462364 1.223784 7753.057500 0.0 -1.0

max 172273.000000 -3.473410 -0.094700 -2.708456 16.875344 8.656575 ... 2.342495 12.152401 13.013546 19656.530000 0.0 -1.0

[8 rows x 32 columns]

Proporción de fraudes en montos altos: 0.00%

A partir del análisis de las transacciones con montos altos (5000-25000), algunos puntos clave surgieron:

Proporción de Fraudes en Montos Altos:

La proporción de transacciones fraudulentas en montos altos fue 0%, lo que sugiere que, en este rango de montos, no se encontraron fraudes, o al menos no fueron detectados como tal.

Distribución de Variables Críticas:

Las variables como Amount, V1, y V4 presentan una distribución diferenciada entre transacciones fraudulentas y no fraudulentas. Aunque los fraudes no estuvieron presentes, las transacciones no fraudulentas exhibieron rangos amplios en estas variables, como se observó en los gráficos de comparación.

Outliers en Montos Altos:

Los gráficos revelan ciertos puntos atípicos (outliers) en los montos altos, lo que podría indicar la necesidad de examinar con mayor profundidad estas transacciones, incluso si no están marcadas como fraudes.

Próximos Pasos:

Revisión Manual de Transacciones Atípicas: Aunque la proporción de fraudes en montos altos fue 0%, podría ser útil examinar manualmente las transacciones atípicas detectadas en estos rangos (particularmente aquellas con montos superiores a 10,000) para identificar patrones inusuales que no se hayan clasificado como fraude automáticamente.

Revisión de Variables Adicionales: Podrías expandir el análisis a otras variables críticas para ver si hay patrones más sutiles que podrían haber pasado desapercibidos en los análisis iniciales.

Optimización de Modelos: Basado en este análisis, podrías ajustar los modelos de detección de fraude para ser más sensibles a transacciones con montos altos, integrando análisis de otras características para una detección más precisa.

He quedado conforme con los modelos y técnicas utilizados en mi dataset
https://www.openml.org/search?type=data&sort=runs&id=1597&status=active

https://github.com/Macazella