

# Predictive Analysis of Transaction Type: Classification Of Financial Operations



Universidad  
Industrial de  
Santander



Daniel Alejandro  
Peña Hurtado  
2211893

Cristian Alberto  
Solano Torres  
2211906

Karen Dayana  
Mateus Gomez  
2212765

Canva

# OBJETIVO

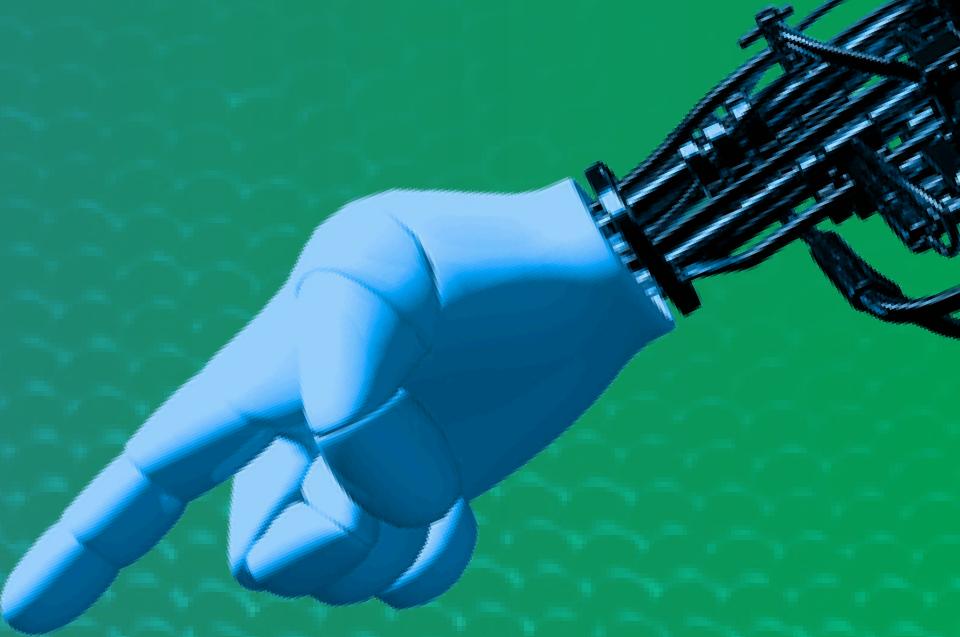
"Aplicar técnicas de clasificación para predecir el tipo de transacción financiera (TransactionType) a partir de características como el monto, saldo y edad del cliente, utilizando el TransactionType como 'ground truth' para evaluar el desempeño del modelo mediante métricas de clasificación."





## ACERCA DE DATASET

Este conjunto de datos proporciona una visión detallada del comportamiento transaccional y los patrones de actividad financiera, ideal para explorar la detección de fraudes y la identificación de anomalías. Contiene 2.512 muestras de datos de transacciones, que cubren varios atributos de transacciones, datos demográficos de los clientes y patrones de uso. Cada entrada ofrece información completa sobre el comportamiento de las transacciones, lo que permite el análisis de las aplicaciones de seguridad financiera y detección de fraude.

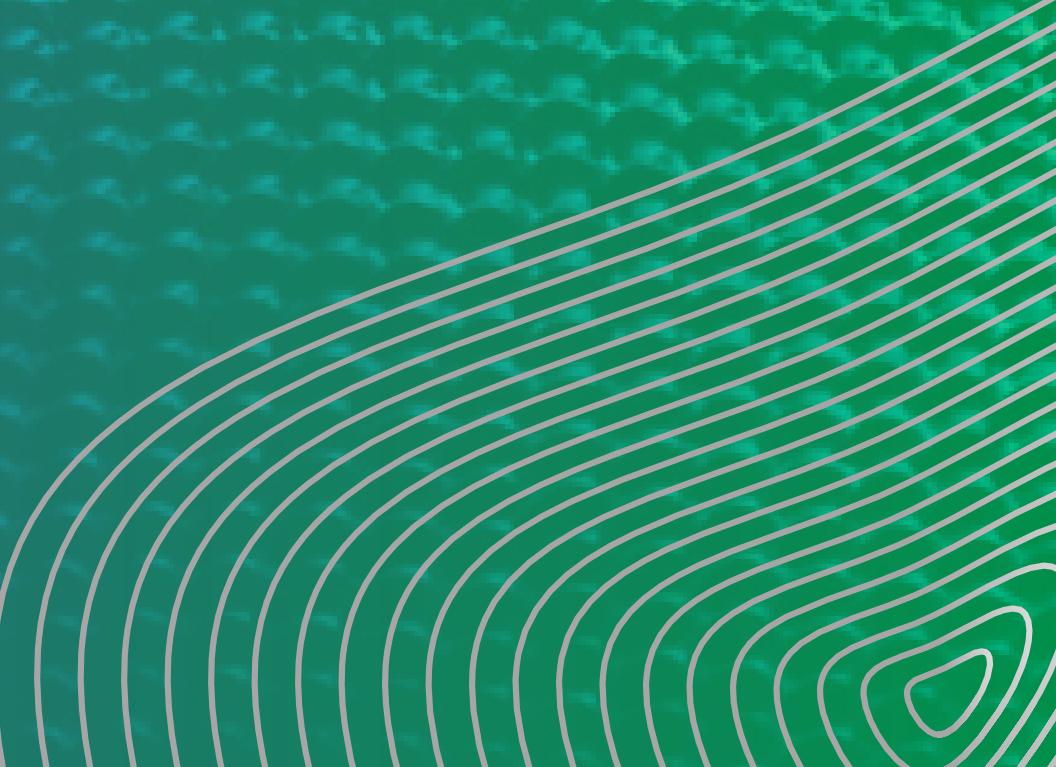


# CARACTERISTICAS CLAVES

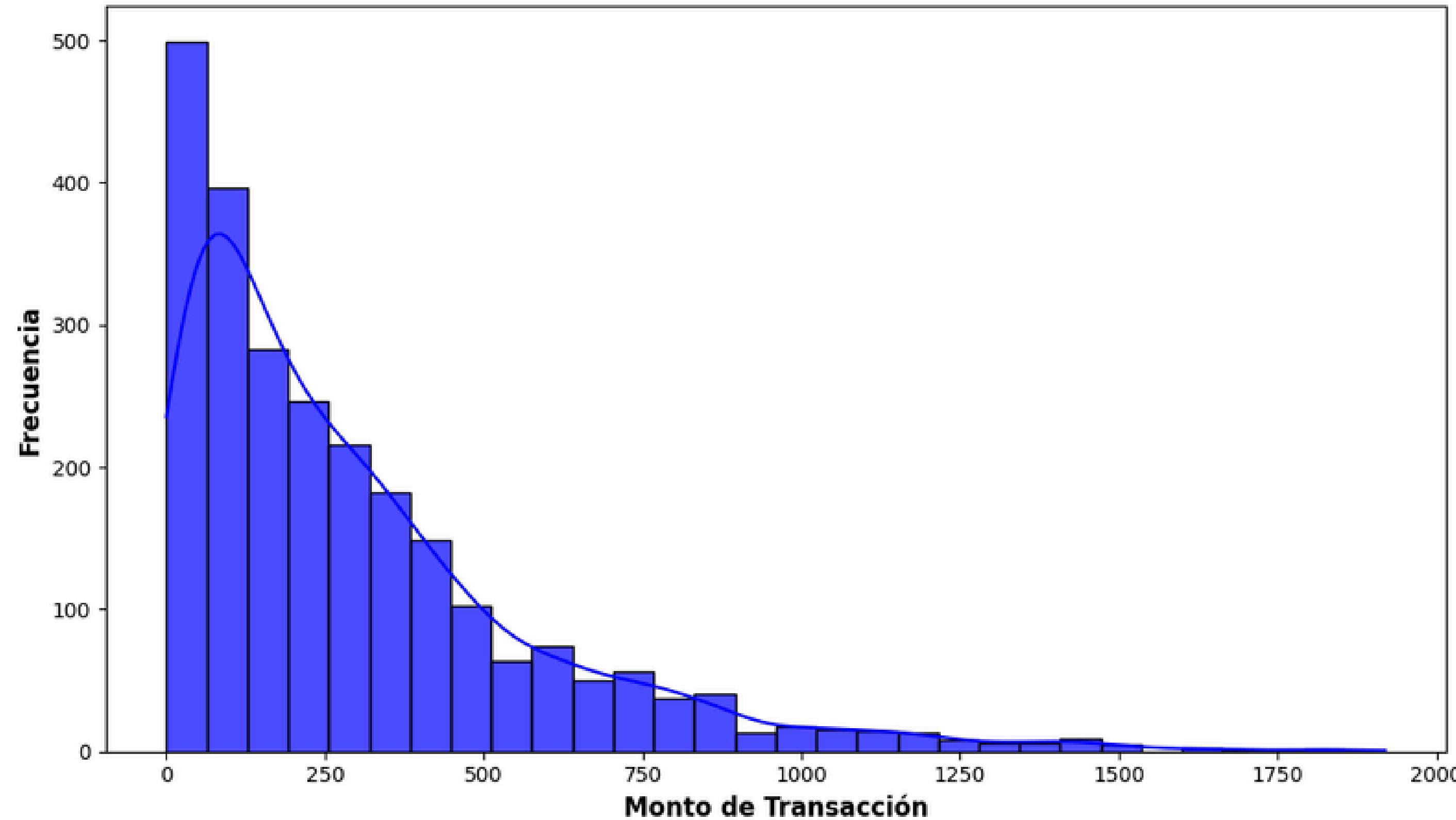
- TransactionID: ID único para cada transacción.
- AccountID: ID único para cada cuenta.
- TransactionAmount: Monto de la transacción.
- TransactionDate: Fecha y hora exacta.
- TransactionType: Crédito o Débito.
- Ubicación: Ciudad donde ocurrió la transacción.
- DeviceID: Dispositivo usado.
- Dirección IP: Dirección IP de origen.
- MerchantID: Comerciantes asociados.
- AccountBalance: Saldo tras la transacción.
- PreviousTransactionDate: Fecha previa.
- Canal: Medio (ATM, Online...).
- CustomerAge: Edad del cliente.
- CustomerOccupation: Ocupación.
- TransactionDuration: Duración de la operación.
- LoginAttempts: Intentos de login.

El dataset incluye 16 características clave que abarcan identificadores, detalles transaccionales, datos del cliente y contexto de uso, lo que permite analizar comportamientos financieros y detectar posibles fraudes.

# PRIMERA ENTREGA AVANCES DE PROYECTO

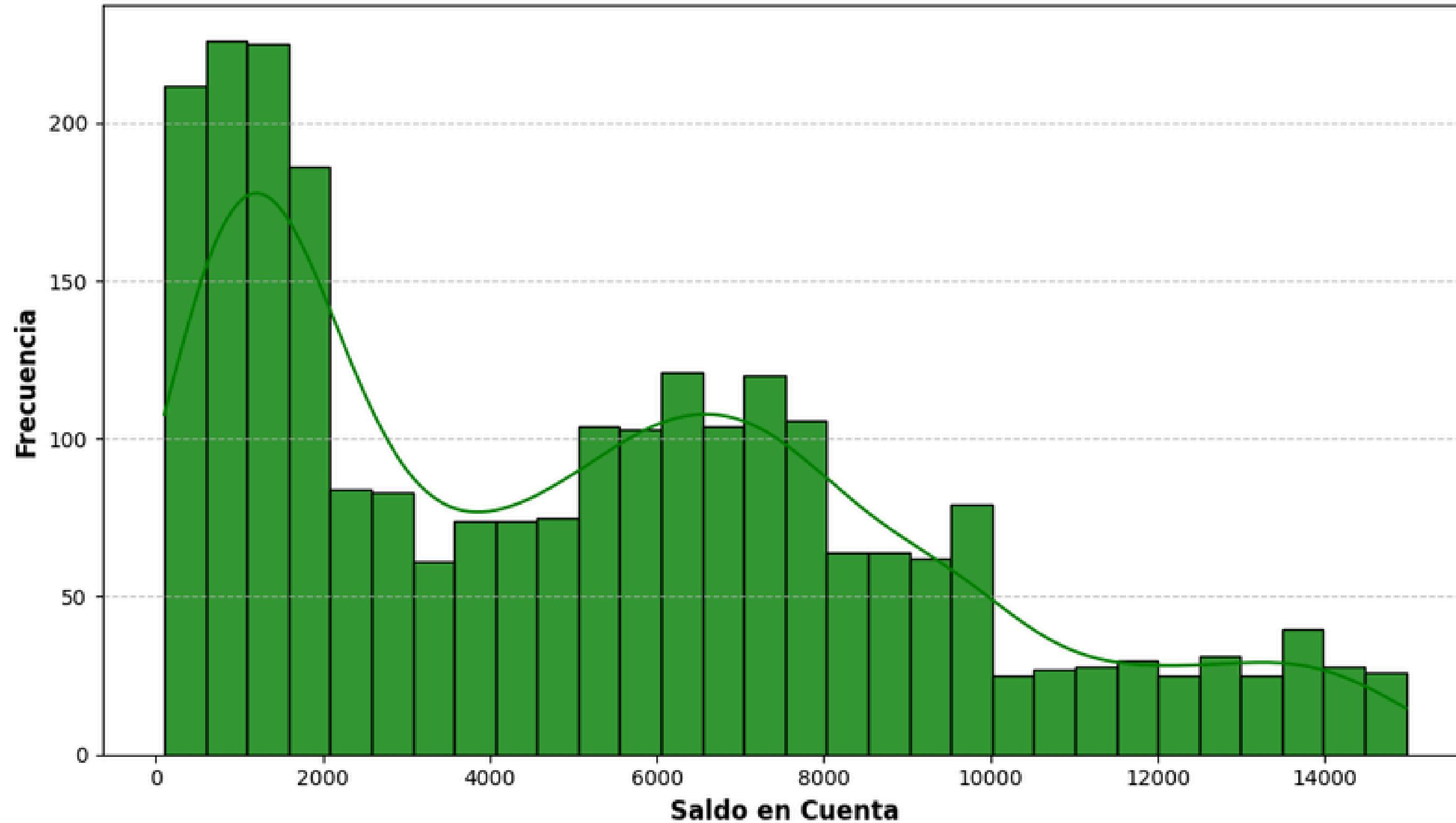


## Distribución de Montos de Transacción

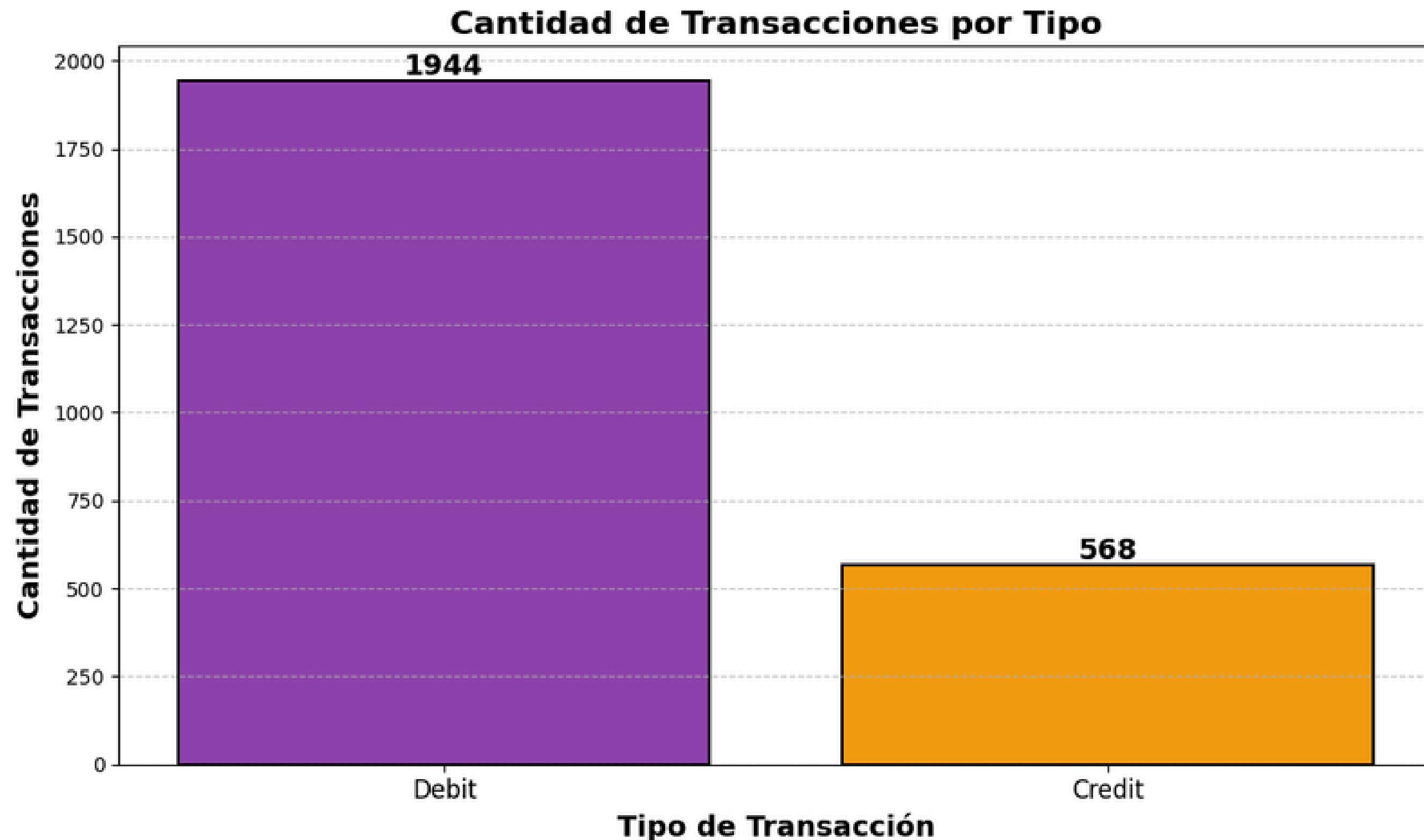


El histograma muestra la distribución de los montos de transacción, con la mayoría de los valores concentrados en el rango de 0 dólares a 250 dólares. A medida que los montos aumentan, la frecuencia de transacciones disminuye significativamente, y son muy pocos los casos con montos superiores a 1000 dólares.

## Distribución de Saldos de Cuenta

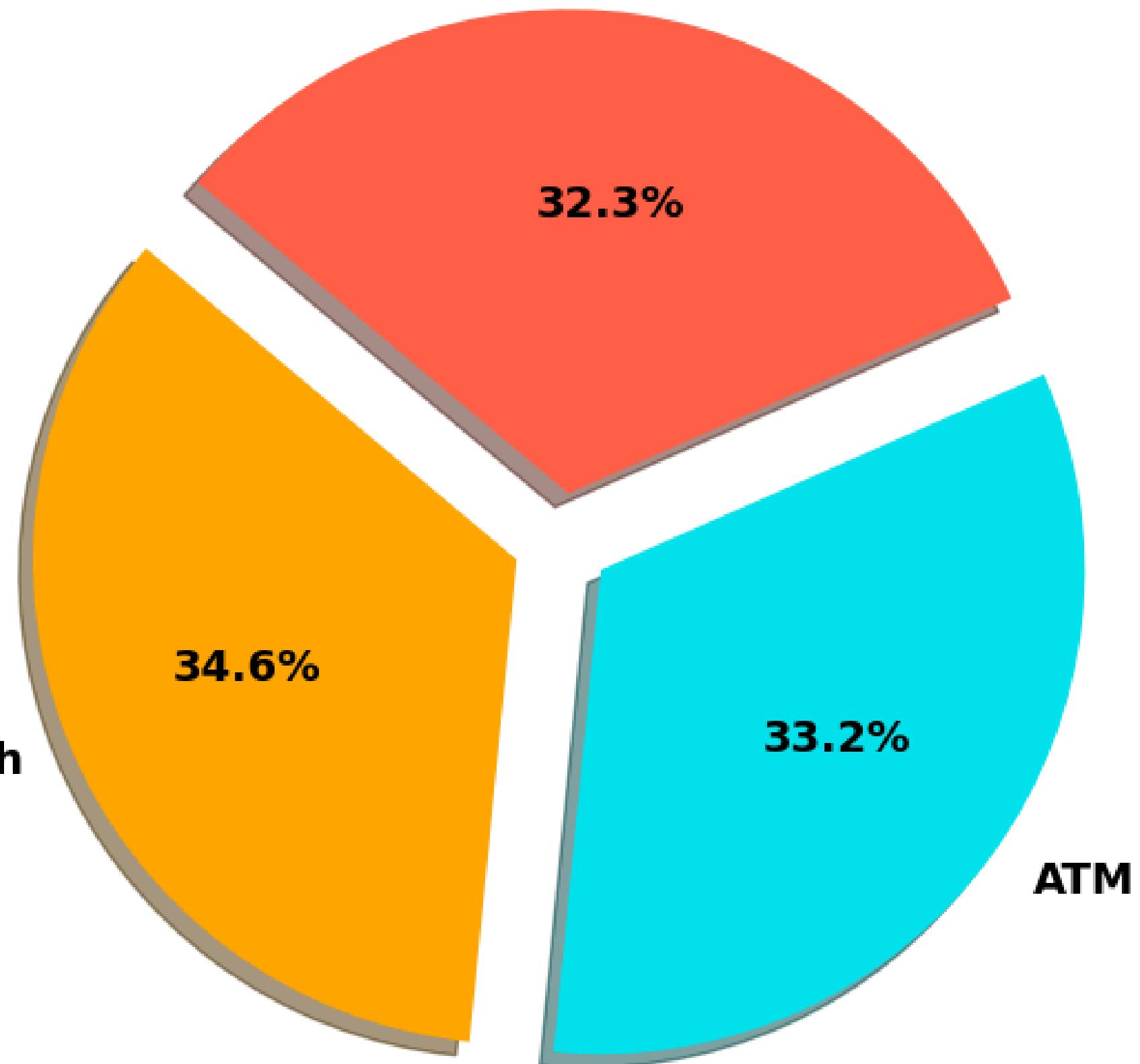


El histograma muestra la distribución de los saldos de cuenta, con la mayoría de los valores concentrados en el rango de 0 dólares a 2,000 dólares, donde la frecuencia es alta. A medida que los saldos aumentan, la frecuencia disminuye significativamente, con pocos saldos superiores a 10,000 dólares. El histograma está dividido en 30 intervalos.



El gráfico de barras muestra la cantidad de transacciones según su tipo: Debit y Credit. La barra morada, correspondiente a las transacciones de débito, es notablemente más alta, con un total de 1945 registros. En contraste, la barra amarilla, que representa las transacciones de crédito, alcanza solo 568 registros.

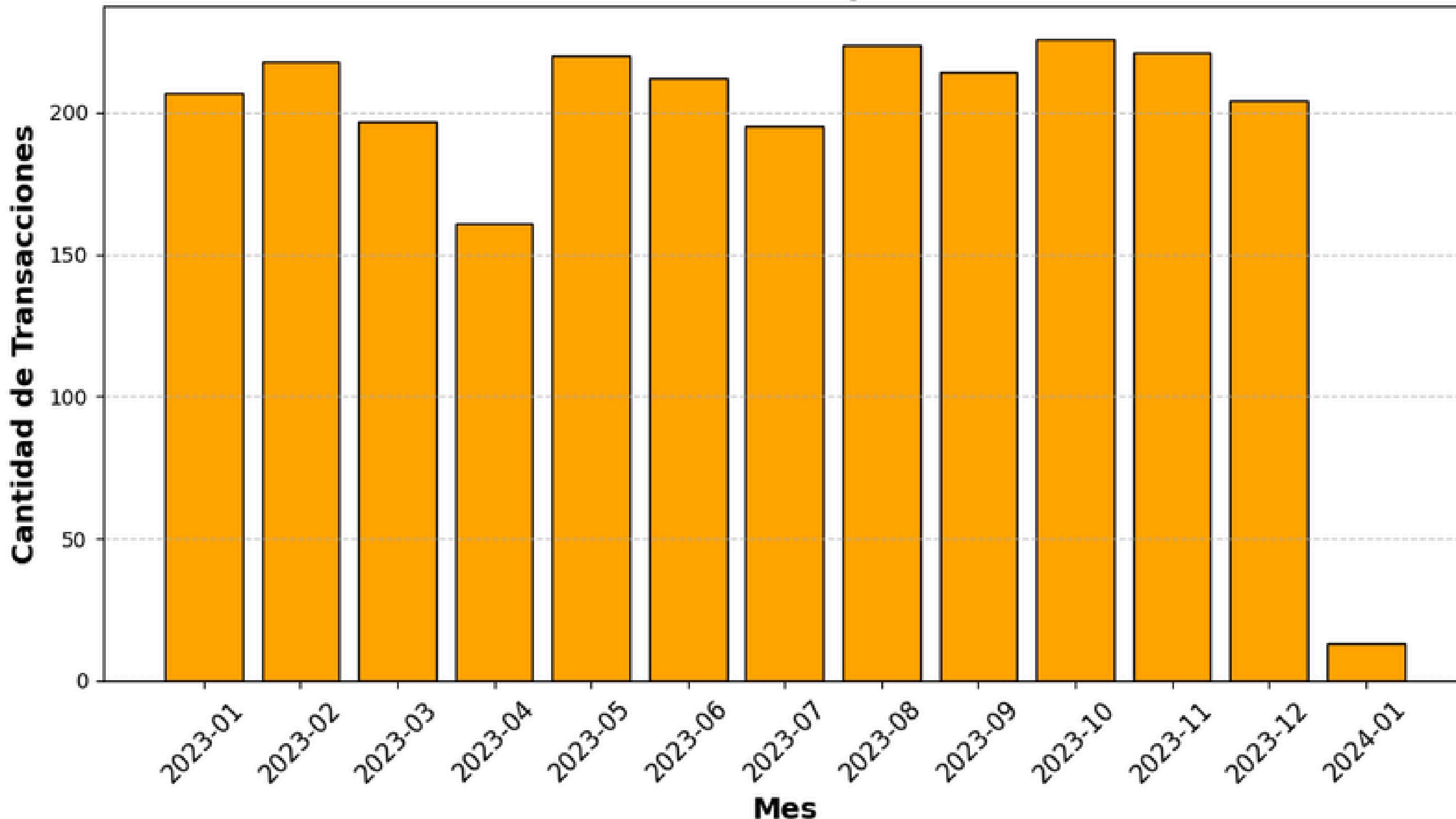
## Distribución de Canales de Transacción Online



El gráfico de sectores muestra la distribución de las transacciones según el canal utilizado. El canal Branch (sucursal) representa el 34.6% del total de transacciones, siendo el canal con mayor participación.

Le sigue el canal ATM (cajero automático) con el 33.2%, y finalmente, el canal Online (en línea) que abarca el 32.3%.

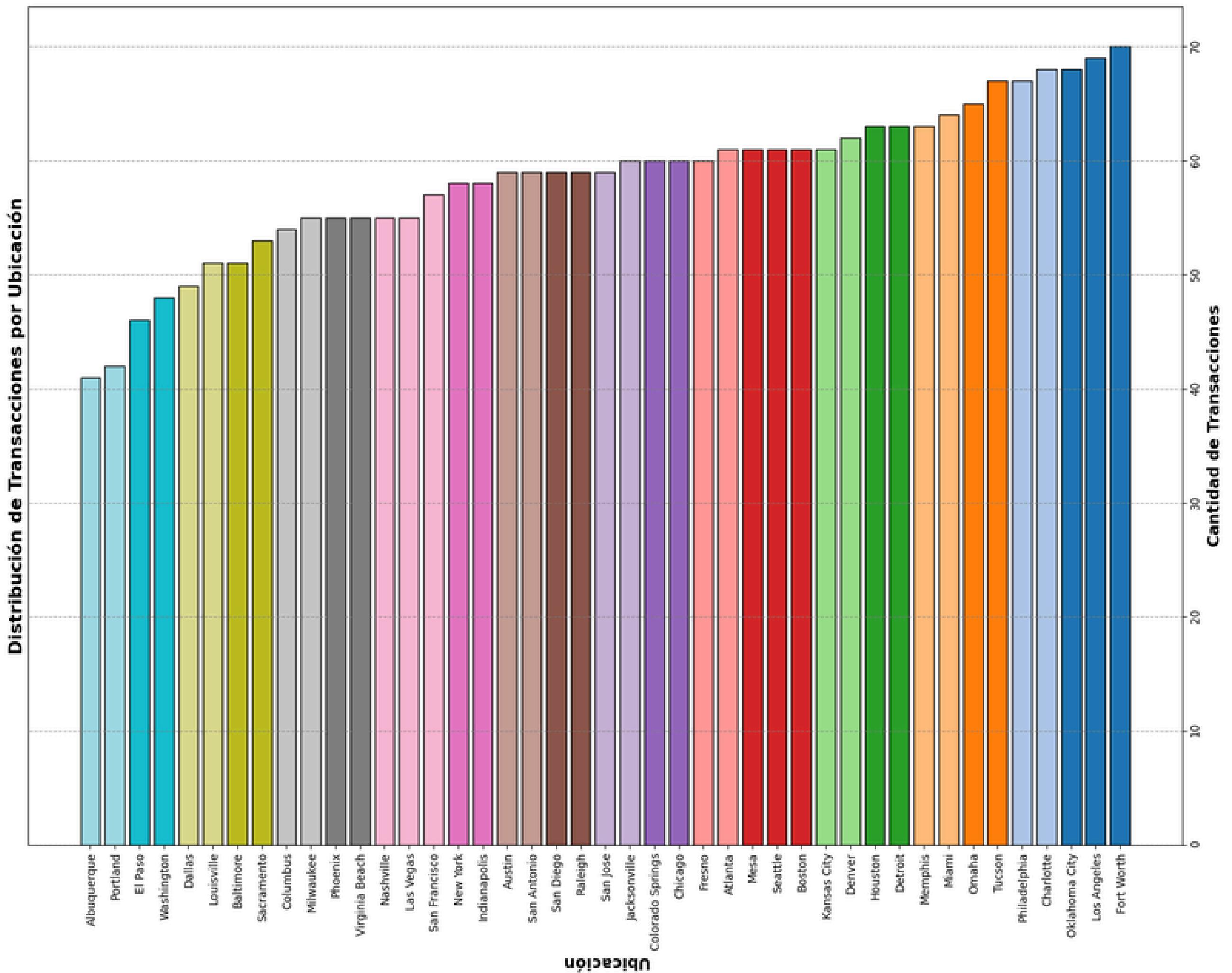
## Transacciones por Mes



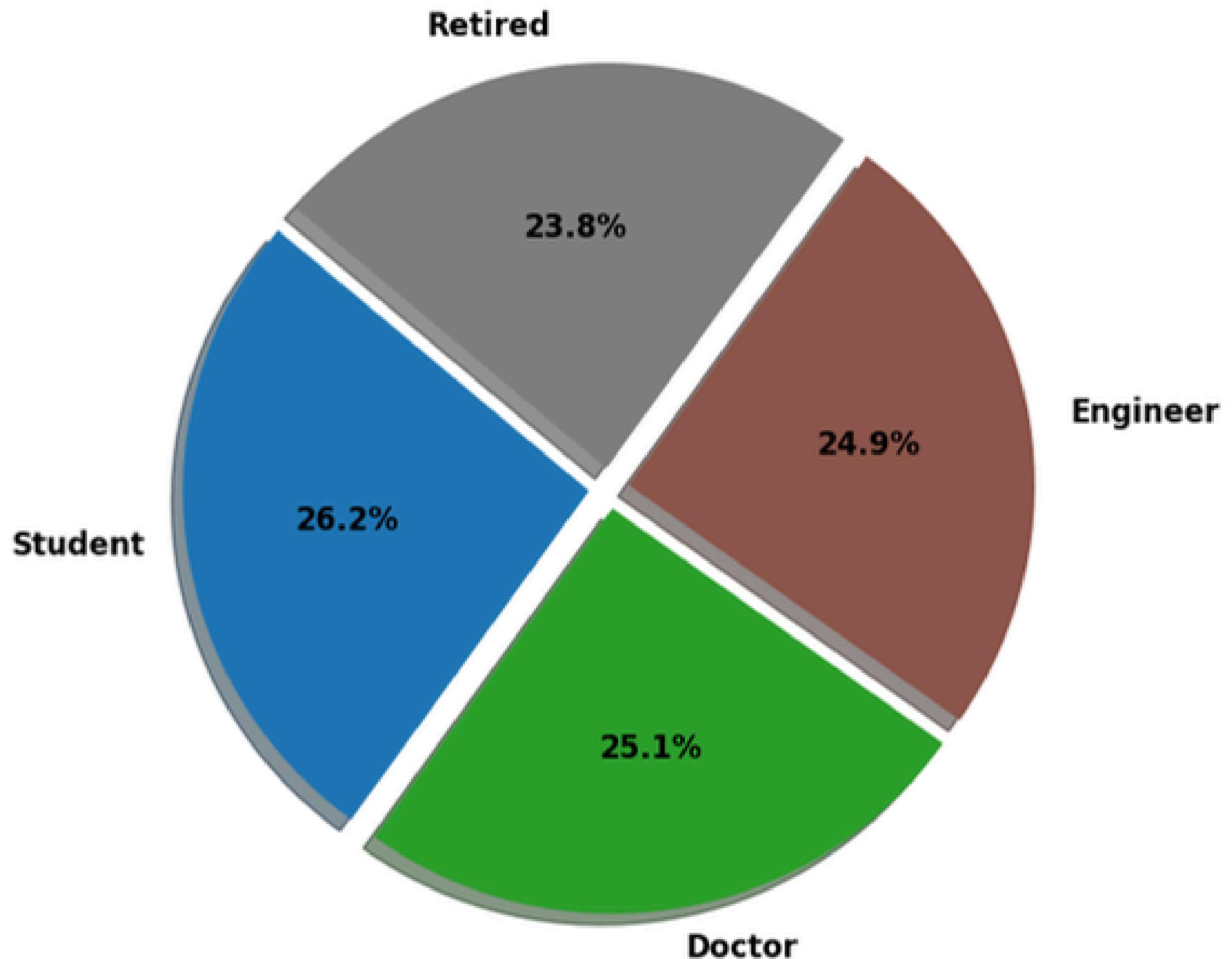
El gráfico muestra que las transacciones se mantienen estables durante el año, con aumentos en octubre y diciembre de 2023, y una caída notable en enero de 2024, el mes con menos transacciones.

El gráfico muestra la cantidad de transacciones por ciudad, destacando a Fort Worth y Los Angeles como las de mayor actividad, mientras que Albuquerque, Portland y El Paso tienen las menores.

Hay una notable variabilidad entre ciudades.



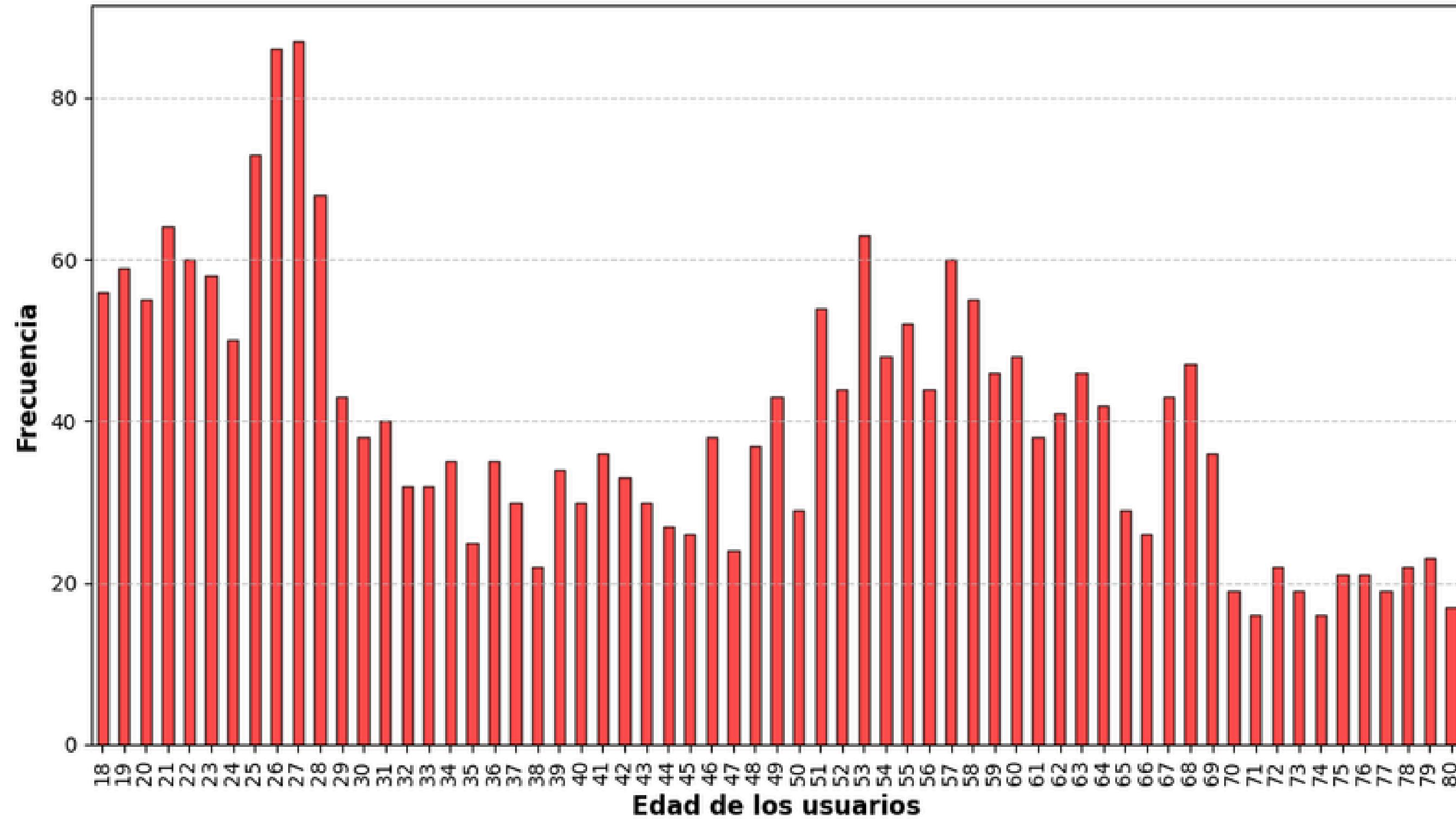
## Distribución de Profesiones



La categoría Student (Estudiante) representa el 26.2% del total, siendo la mayor proporción. Le sigue Doctor con el 25.1%, Engineer (Ingeniero) con el 24.9%, y finalmente Retired (Jubilado) con el 23.8%.

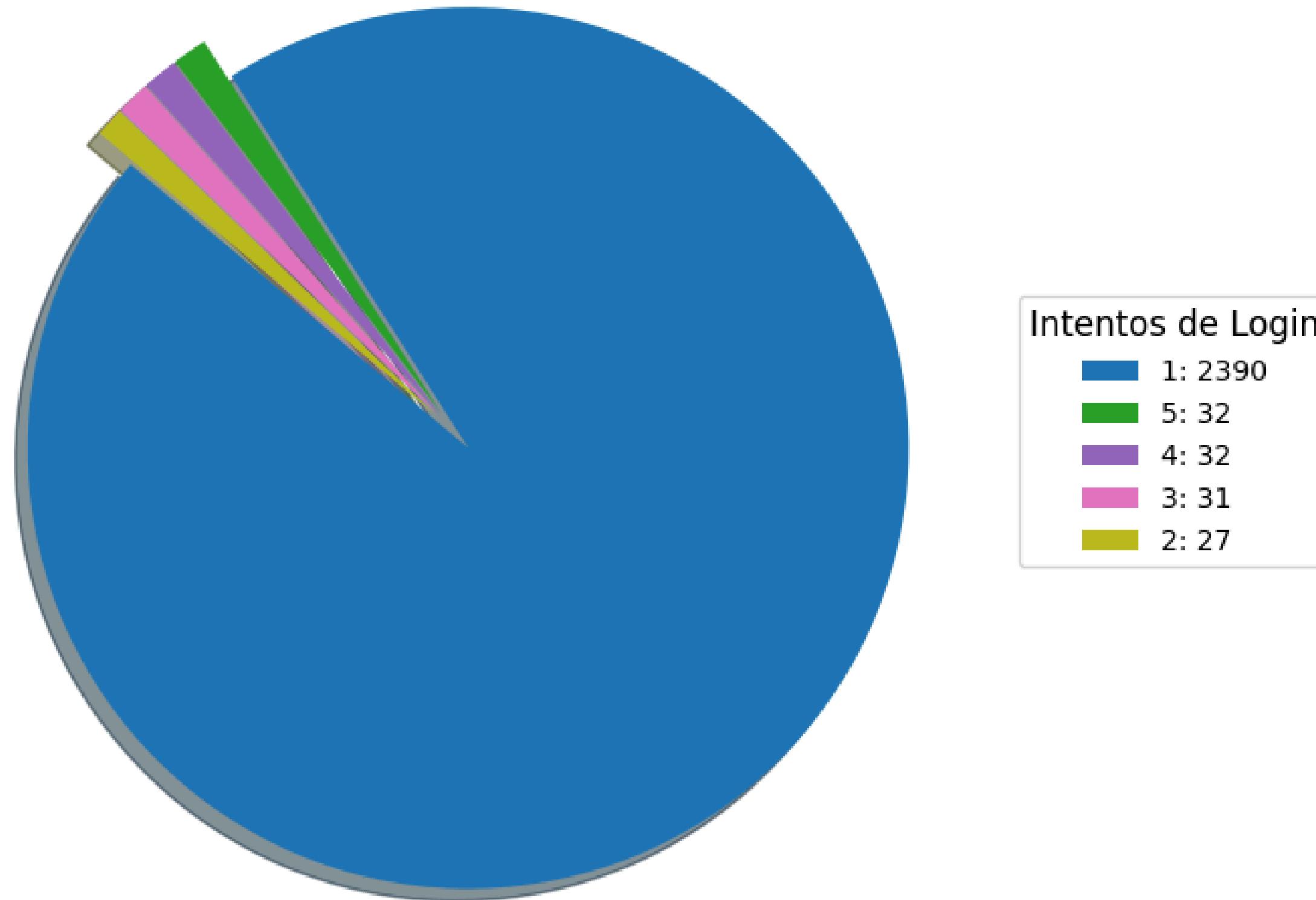
El gráfico muestra una distribución relativamente equilibrada entre las distintas profesiones, con una ligera superioridad en los estudiantes

## Distribución de Edades



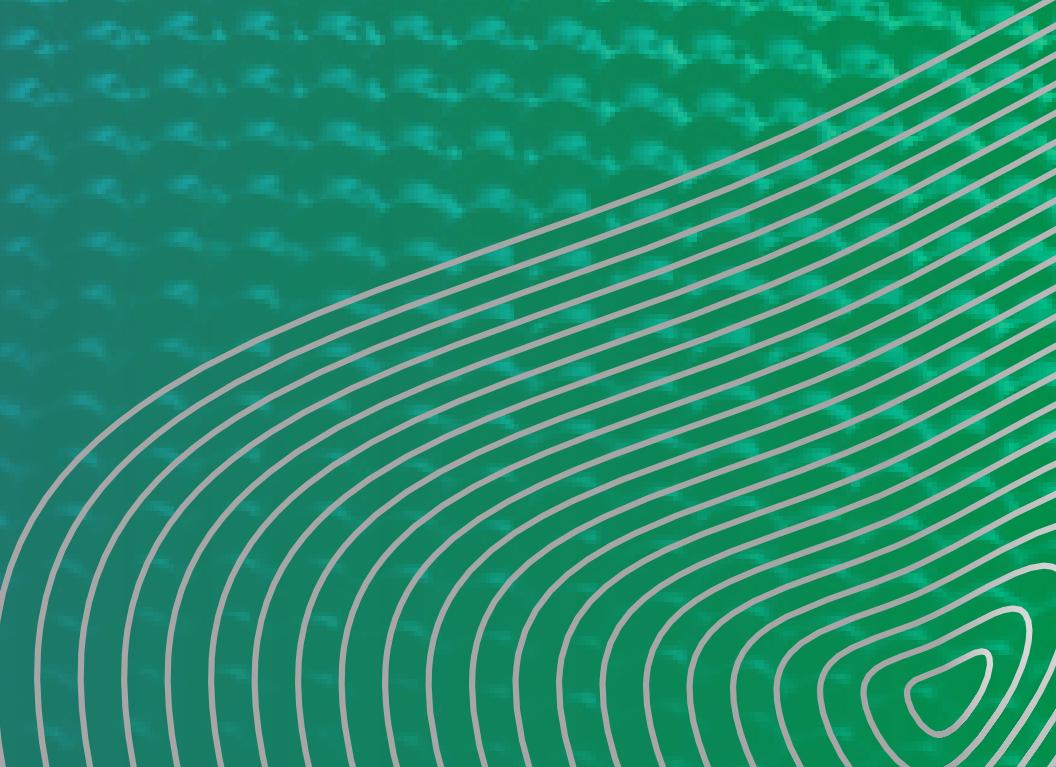
El gráfico muestra que los usuarios más activos están entre los 20 y 25 años, con alta participación también entre los 18-31 y 51-60 años. La frecuencia disminuye con la edad, especialmente después de los 60, aunque hay una leve alza entre los 67 y 69 años.

## Distribución de Intentos de Inicio de Sesión



El gráfico muestra que la mayoría de los intentos de inicio de sesión son únicos (valor 1), con 2390 casos, mientras que los intentos múltiples son poco frecuentes, con cantidades significativamente menores.

# SEGUNDA ENTREGA AVANCES DE PROYECTO



# CLASIFICACIÓN

La clasificación es adecuada para este proyecto porque el objetivo es predecir el tipo de transacción (TransactionType), una variable categórica como "Crédito" o "Débito".

Esta variable actúa como ground truth, permitiendo entrenar y evaluar el modelo mediante métricas como la precisión, a partir de características como monto, saldo y edad del cliente.



# ESTIMADORES SIN PARAMETROS

```
1 #@title Convertir Valores Categóricos en Valores Numéricos
2 from sklearn.preprocessing import LabelEncoder
3
4 a = pd.read_csv("/content/drive/MyDrive/ProyectoIA1/data/bank_transactions_data_2.csv")
5 a= a.drop(['TransactionDate', 'PreviousTransactionDate'], axis=1)
6
7 df_encoded = a.copy()
8 le = LabelEncoder()
9
10 for col in df_encoded.columns:
11     if df_encoded[col].dtype == 'object':
12         df_encoded[col] = le.fit_transform(df_encoded[col])
13
14 print(df_encoded.head())
```

CONVERTIMOS VALORES CATEGORICOS EN VALORES NUMERICOS

# ESTIMADORES SIN PARAMETROS

	TransactionID	AccountID	TransactionAmount	TransactionType	Location	\	
0	0	126	14.09	1	36		
1	1	450	376.24	1	15		
2	2	18	126.29	1	23		
3	3	68	184.50	1	33		
4	4	406	13.45	0	1		
	DeviceID	IP Address	MerchantID	Channel	CustomerAge	CustomerOccupation	\
0	365	186	14	0	70	0	
1	50	82	51	0	68	0	
2	229	343	8	2	19	3	
3	182	300	1	2	26	3	
4	298	501	90	2	26	3	
	TransactionDuration	LoginAttempts	AccountBalance				
0	81	1	5112.21				
1	141	1	13758.91				
2	56	1	1122.35				
3	25	1	8569.06				
4	198	1	7429.40				

El Label Encoding convirtió correctamente las variables categóricas en valores numéricos, facilitando su uso en modelos de machine learning. Aunque las categorías no tienen un orden natural, el dataframe ya está listo para el análisis predictivo.

# DECISION TREE CLASSIFIER (DT)

```
1 from sklearn.tree import DecisionTreeClassifier  
2 from sklearn.model_selection import train_test_split  
3 from sklearn.metrics import *  
4  
5 # --- Definimos TransactionType como ground truth  
6 X = df_encoded.drop('TransactionType', axis=1).values  
7 y = df_encoded['TransactionType'].values  
8  
9 #--- Entrenamos el modelo dejando  
10 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=21, shuffle=True)  
11 est = DecisionTreeClassifier()  
12 est.fit(X_train, y_train)  
13 print("Accuracy en test set:", accuracy_score(est.predict(X_test), y_test))
```

Accuracy en test set: 0.6699801192842942

El dataset está desbalanceado (más débitos que créditos), lo que puede sesgar al modelo. Aunque el árbol de decisión no predice solo la clase mayoritaria, su accuracy de 66.9% no es óptimo. Por ello, se recomienda usar métricas como precisión, sensibilidad y matriz de confusión para evaluar mejor su desempeño.

# RANDOM FOREST CLASSIFIER (RF)

```
1 #@title Random Forest Classifier (RF)
2 from sklearn.model_selection import train_test_split, cross_val_score, KFold
3 from sklearn.ensemble import RandomForestClassifier
4 from sklearn.metrics import accuracy_score, make_scorer
5
6 # --- Definimos TransactionType como ground truth
7 X = df_encoded.drop('TransactionType', axis=1).values
8 y = df_encoded['TransactionType'].values
9
10 #--- Entrenamos el modelo dejando
11 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=21, shuffle=True)
12 est = RandomForestClassifier()
13 est.fit(X_train, y_train)
14 print("Accuracy en test set:", accuracy_score(est.predict(X_test), y_test))
```

Accuracy en test set: 0.7852882703777336

El modelo Random Forest alcanzó una accuracy de 0.785, superando al árbol de decisión por su mayor capacidad de generalización. Sin embargo, el desbalance de clases puede hacer que esta métrica sea engañosa, favoreciendo la clase mayoritaria sin representar bien la minoritaria.

# SUPPORT VECTOR MACHINE (SVM)

```
1 #@title Support vector Machine (SVM)
2 from sklearn.model_selection import train_test_split, cross_val_score, KFold
3 from sklearn.svm import SVC
4 from sklearn.metrics import accuracy_score, make_scorer
5
6 # --- Definimos TransactionType como ground truth
7 X = df_encoded.drop('TransactionType', axis=1).values
8 y = df_encoded['TransactionType'].values
9
10 #---- Entrenamos el modelo dejando
11 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=21, shuffle=True)
12 est = SVC()
13 est.fit(X_train,y_train)
14 print("Accuracy en test set:", accuracy_score(est.predict(X_test), y_test))
```

Accuracy en test set: 0.7952286282306164

El modelo SVC alcanzó una precisión de 0.795, superando a Random Forest y Decision Tree. Aunque muestra mejor generalización, el desbalance entre clases puede hacer que el accuracy sea engañoso, ya que podría estar favoreciendo la clase mayoritaria.

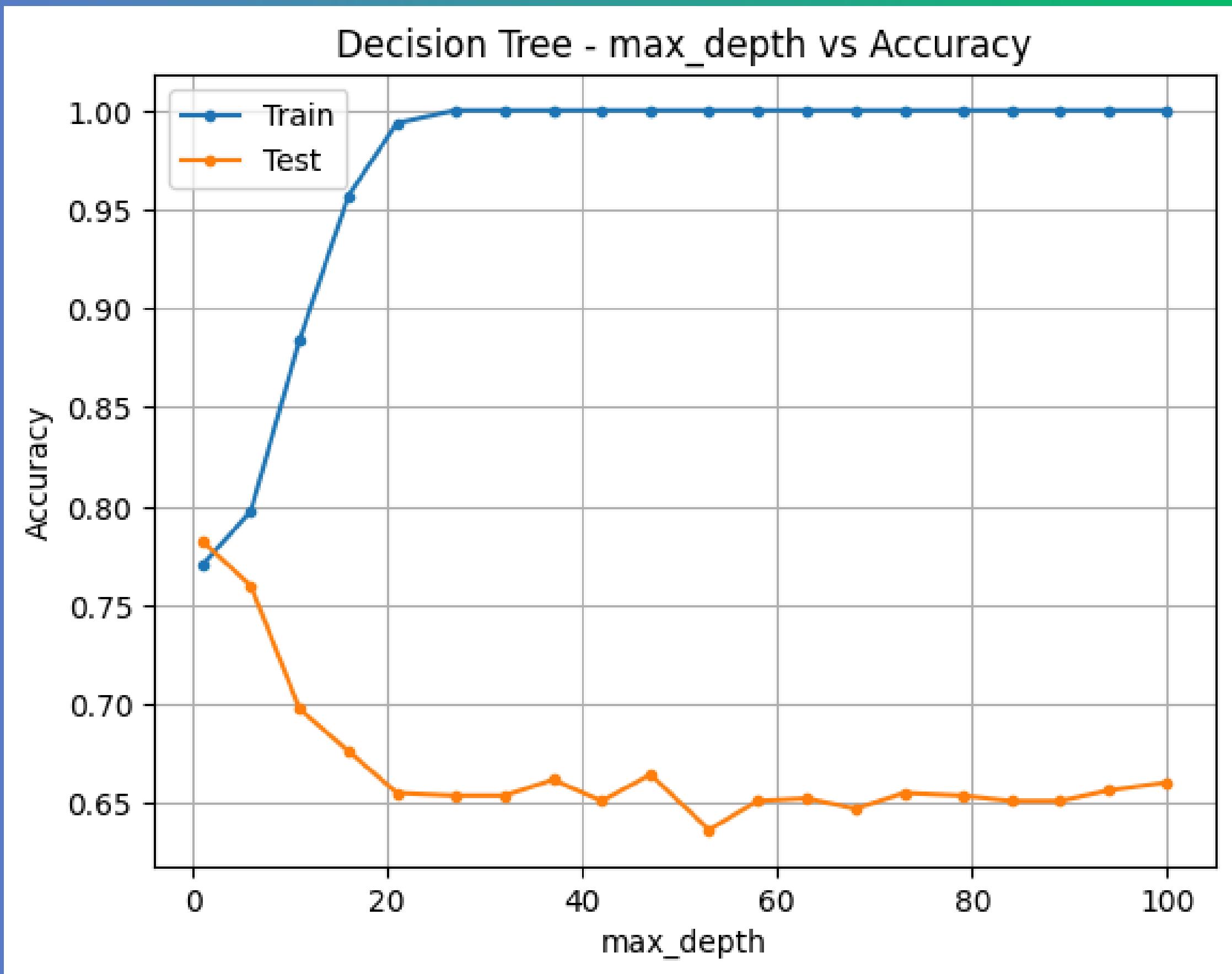
# LEARNING CURVES: DECISIONTREE, RANDOM FOREST Y SUPPORT VECTOR MACHINE



# DECISIONTREE (MAX\_DEPTH)

El gráfico muestra sobreajuste: al aumentar la profundidad, el árbol memoriza los datos y pierde capacidad de generalización. La precisión en test cae y se estabiliza cerca del 65%.

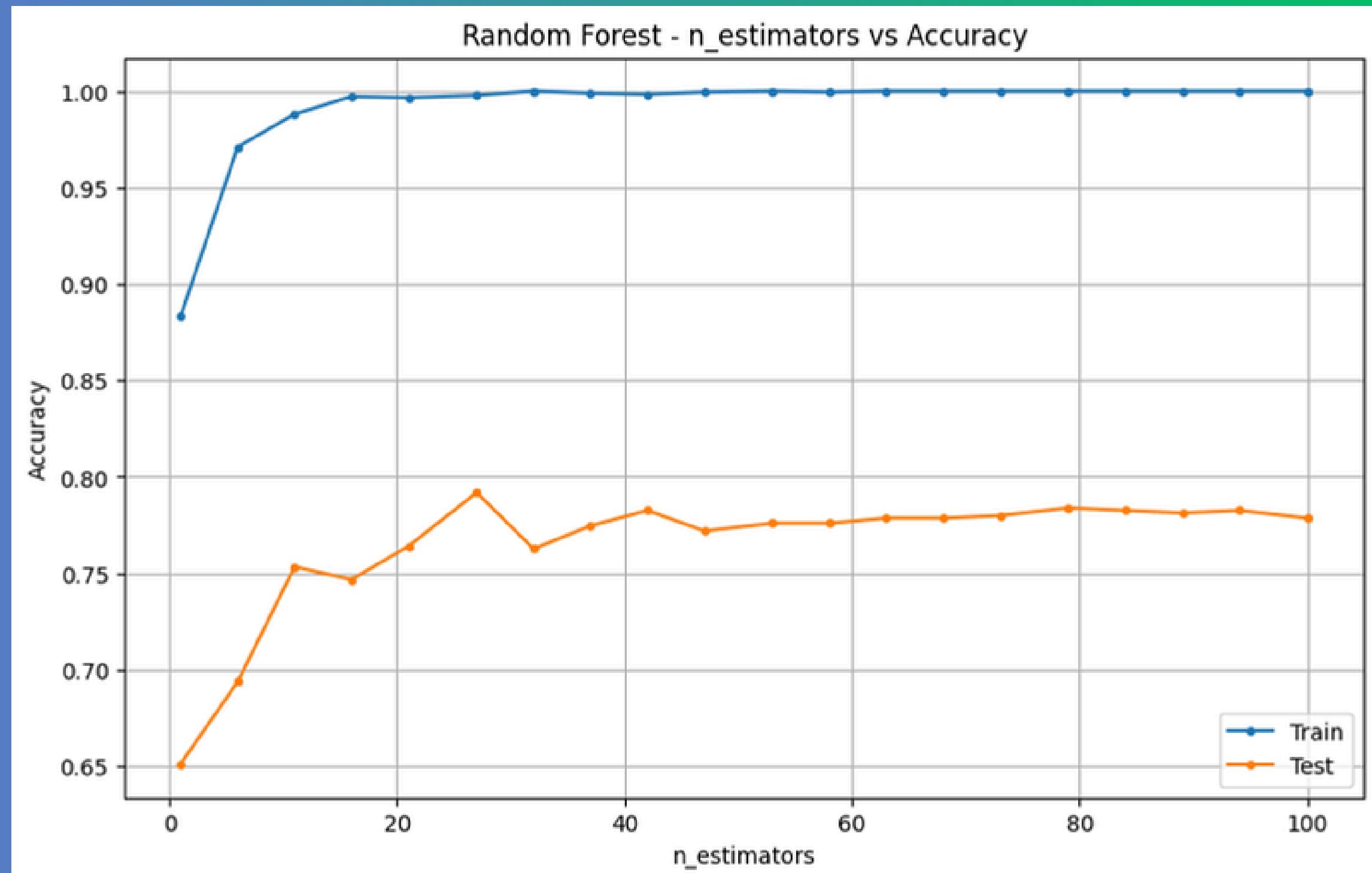
El mejor rendimiento se da con profundidades entre 3 y 5, indicando que un modelo simple es más eficaz para predecir el TransactionType.



# RANDOM FOREST (n\_estimators)

El gráfico muestra sobreajuste: al aumentar la profundidad, el árbol memoriza los datos y pierde capacidad de generalización. La precisión en test cae y se estabiliza cerca del 65%.

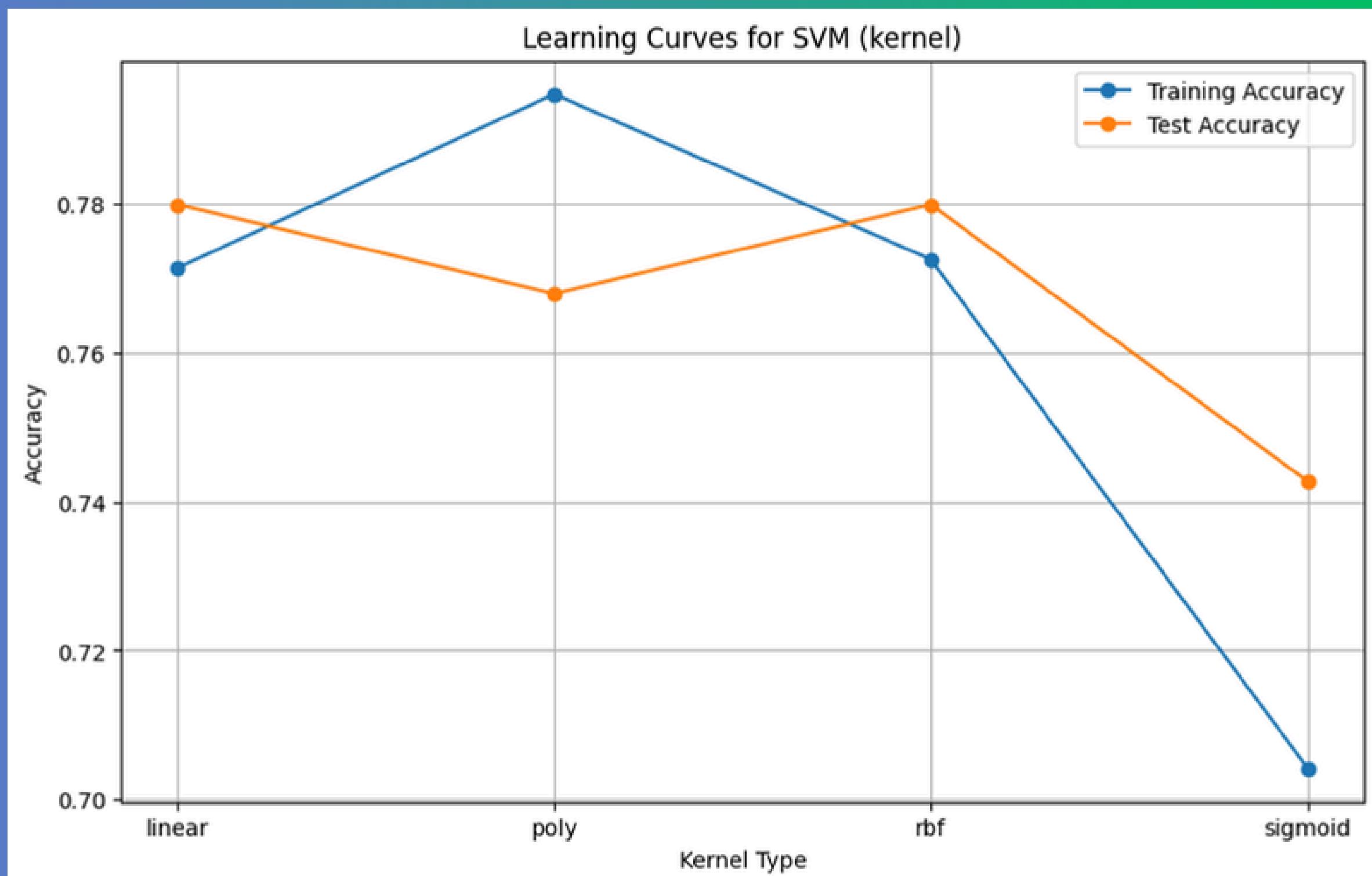
El mejor rendimiento se da con profundidades entre 3 y 5, indicando que un modelo simple es más eficaz para predecir el TransactionType.



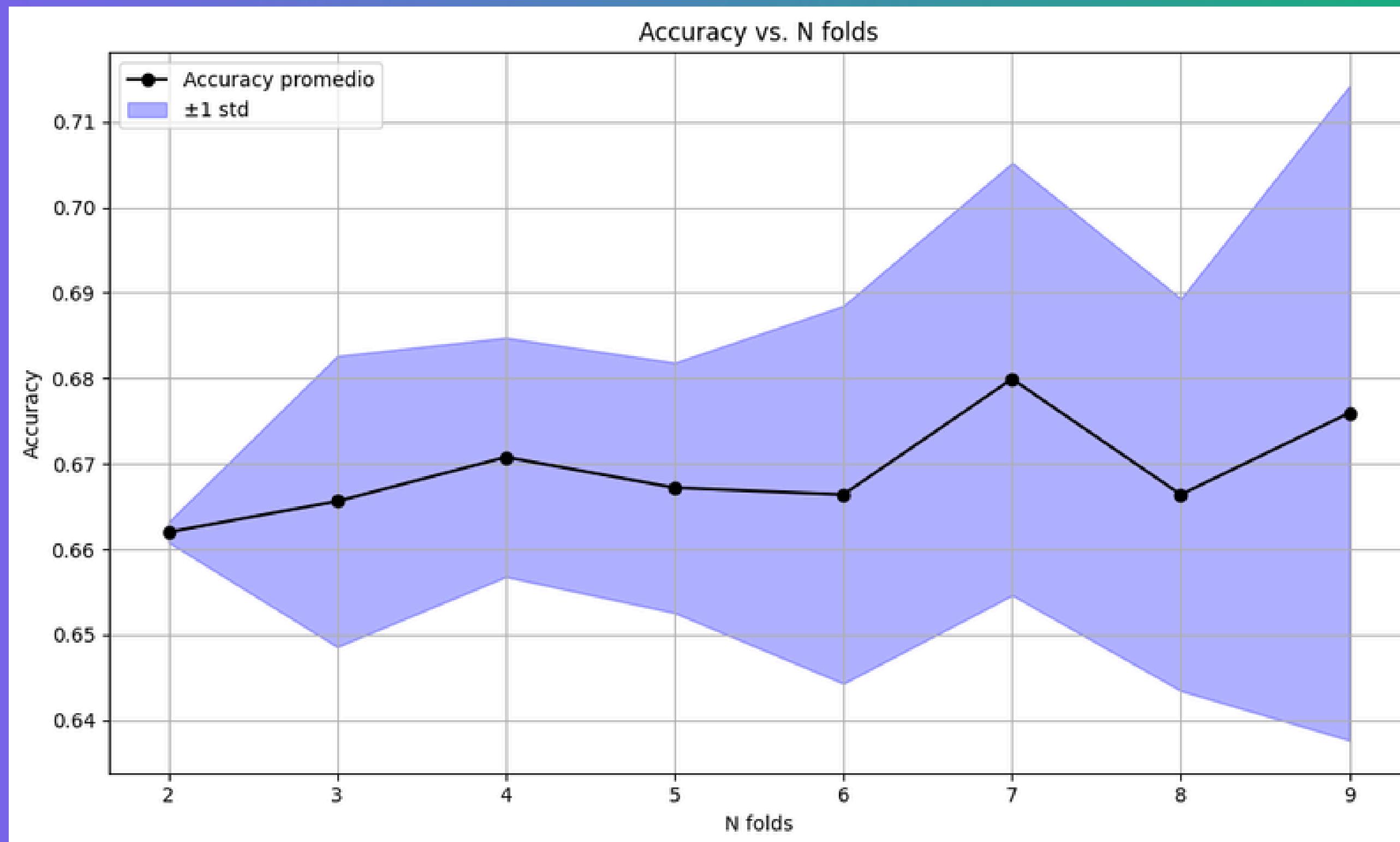
# SUPPORT VECTOR MACHINE (KERNEL)

El gráfico muestra que el kernel poly tiene la mayor precisión en entrenamiento (~80%) y una leve caída en prueba (~77%). Los kernels linear y rbf tienen un rendimiento equilibrado (~77-78% en ambos).

El kernel sigmoid presenta una precisión en prueba superior al entrenamiento (~74% vs. ~70%), sugiriendo una mejor adaptación al conjunto de prueba. El kernel rbf es el que mejor generaliza para este dataset.



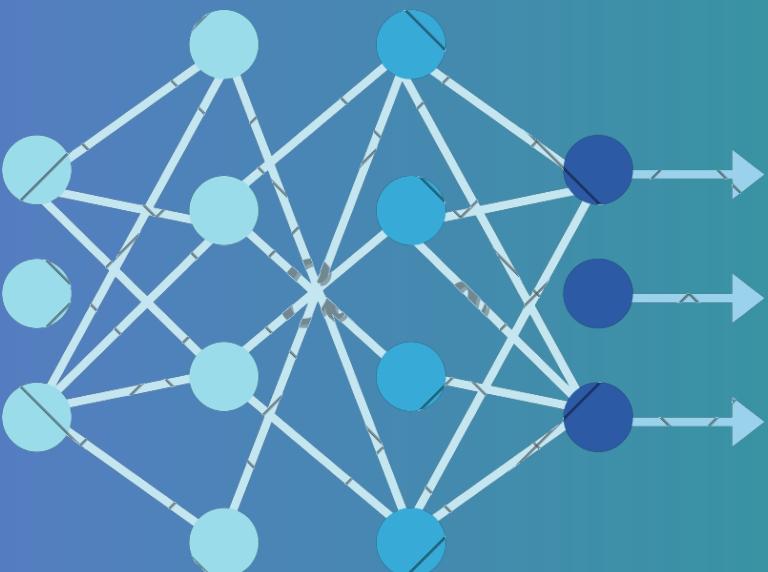
# LEARNING CURVES PARA CROSS-VALIDATION (DECISION TREE DT)



# TERCERA ENTREGA AVANCES DE PROYECTO



# IMPLEMENTACIÓN DE CLASIFICACIÓN EN UNA RED NEURONAL



Se entrenaron tres redes neuronales para clasificación, con 3, 6 y 10 capas ocultas respectivamente, todas con 128 neuronas por capa y activación ReLU.

```
1 from sklearn.model_selection import train_test_split
2 from sklearn.preprocessing import LabelEncoder, MinMaxScaler
3 import tensorflow as tf
4 from tensorflow import keras
5
6 keras.utils.set_random_seed(21)
7
8 def train_classification_model(a, target_column, layers=3, epochs=10):
9     y = a[target_column]
10    X = a.drop(columns=[target_column])
11
12    scaler = MinMaxScaler()
13    X_scaled = scaler.fit_transform(X)
14
15    label_encoder = LabelEncoder()
16    y_encoded = label_encoder.fit_transform(y)
17
18    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_encoded, test_size=0.10, random_state=21)
19
20    model = tf.keras.Sequential()
21    model.add(tf.keras.layers.Flatten(input_shape=(X_train.shape[1],)))
22
23    for _ in range(layers):
24        model.add(tf.keras.layers.Dense(128, activation='relu'))
25
26    model.add(tf.keras.layers.Dense(len(np.unique(y_encoded)), activation='softmax'))
27
28    model.compile(optimizer=tf.keras.optimizers.SGD(),
29                  loss='sparse_categorical_crossentropy',
30                  metrics=['accuracy'])
31
32    model.fit(X_train, y_train, epochs=epochs, validation_data=(X_test, y_test), verbose=1)
33
34    test_loss, test_acc = model.evaluate(X_test, y_test, verbose=0)
35
36    return model, test_acc
37
38 # Entrenando modelos con 3, 6 y 10 capas
39 model_3_layers, acc_3 = train_classification_model(df_encoded, target_column="TransactionType", layers=3, epochs=10)
40 model_6_layers, acc_6 = train_classification_model(df_encoded, target_column="TransactionType", layers=6, epochs=10)
41 model_10_layers, acc_10 = train_classification_model(df_encoded, target_column="TransactionType", layers=10, epochs=10)
42
43 # Imprimir resultados con formato bonito
44 print("\nResultados del modelo de clasificación:\n")
45 print("-" * 40)
46 print(f"Modelo con 3 capas ocultas: \n\tPrecisión de test: {acc_3 * 100:.2f}%\n")
47 print("-" * 40)
48 print(f"Modelo con 6 capas ocultas: \n\tPrecisión de test: {acc_6 * 100:.2f}%\n")
49 print("-" * 40)
50 print(f"Modelo con 10 capas ocultas: \n\tPrecisión de test: {acc_10 * 100:.2f}%\n")
51 print("-" * 40)
```

El modelo alcanza su mejor precisión (~77.78%) tras pocas epochs, por lo que 5-7 serían suficientes. Más capas no mejoran el rendimiento, sugiriendo que un modelo más simple sería más eficiente.

Aunque la pérdida disminuye, su reducción es leve; ajustar los parámetros de Adam podría mejorarla.

## Resultados del modelo de clasificación:

---

### Modelo con 3 capas ocultas:

Precisión de test: 77.78%

---

### Modelo con 6 capas ocultas:

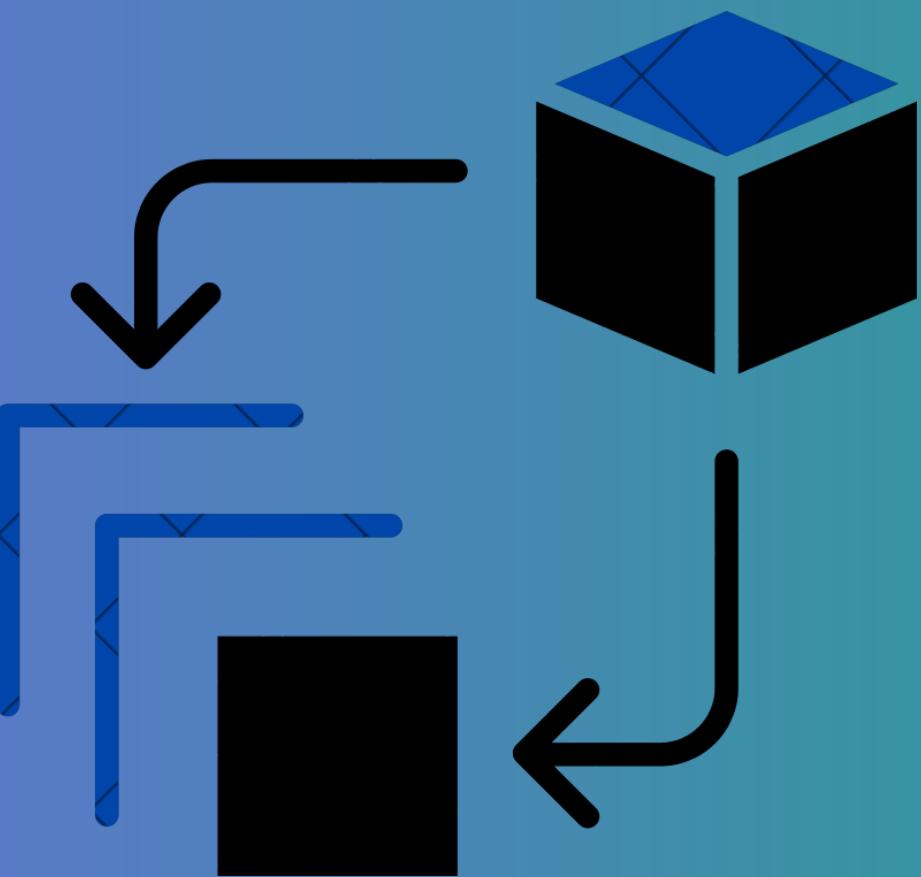
Precisión de test: 77.78%

---

### Modelo con 10 capas ocultas:

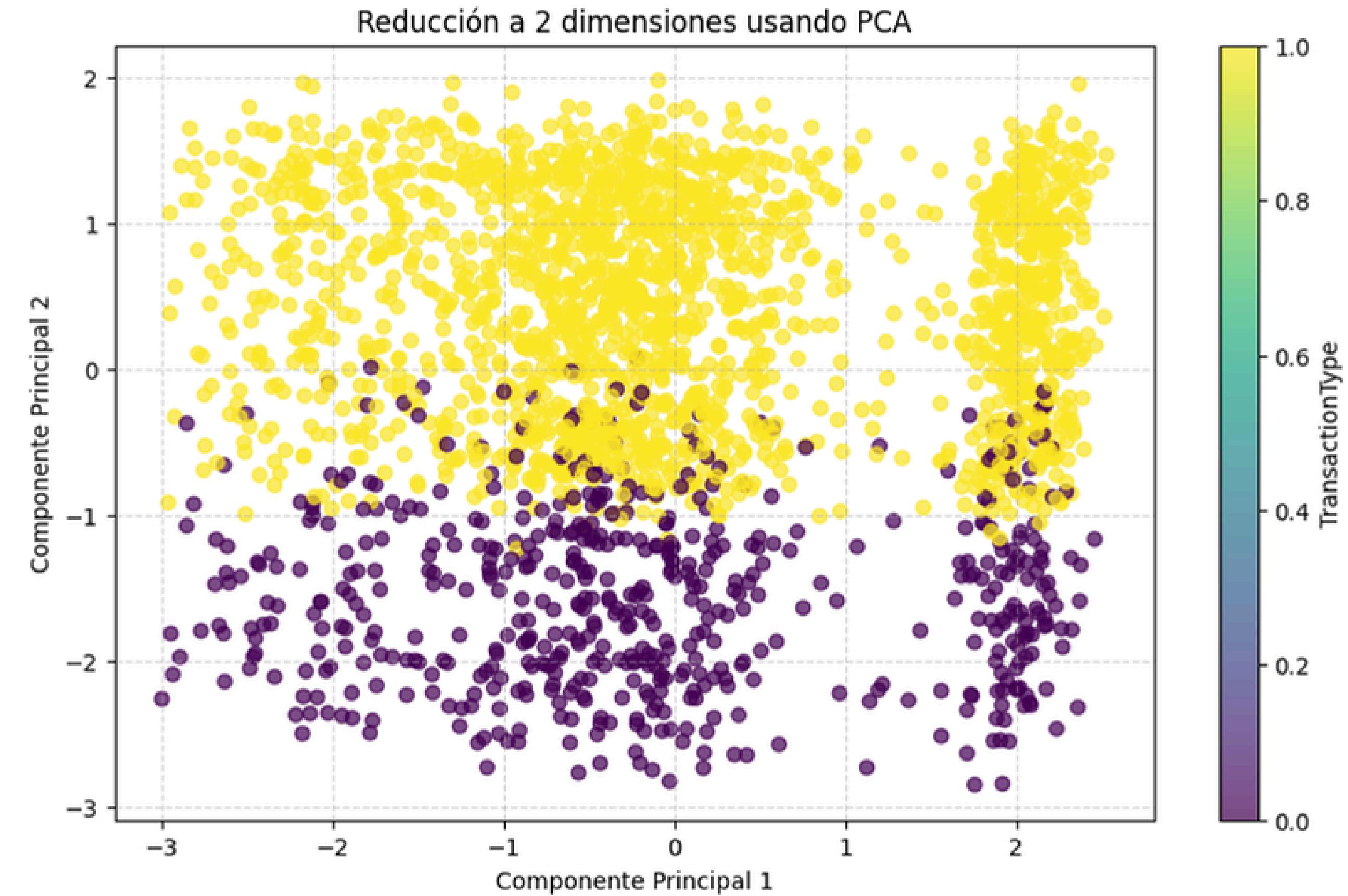
Precisión de test: 77.78%

# REDUCCIÓN DE DIMENSIONALIDAD CON PCA

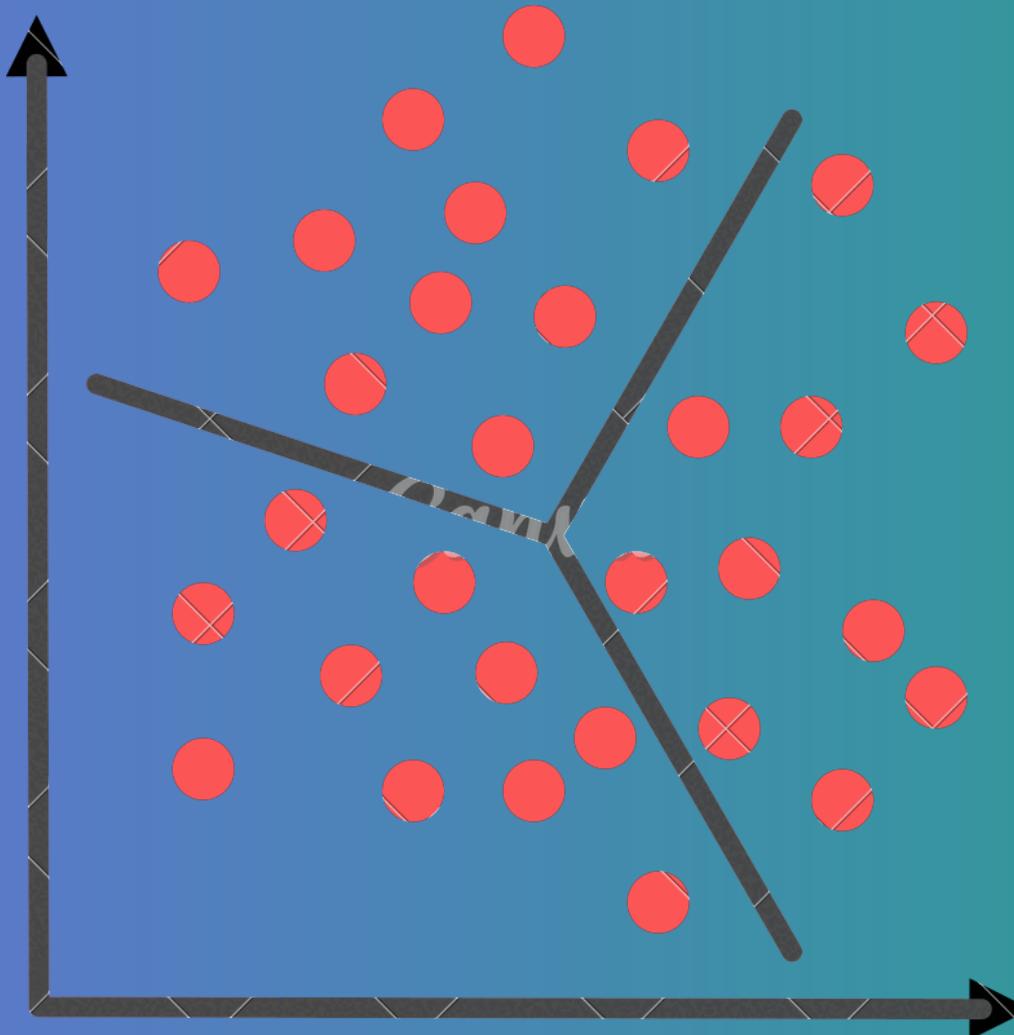


PCA reduce las 14 características a 2 para facilitar la visualización y análisis, tras estandarizar los datos.

Aunque separa bien las clases, persiste cierto solapamiento, lo que sugiere que usar más componentes podría mejorar la representación sin perder información.

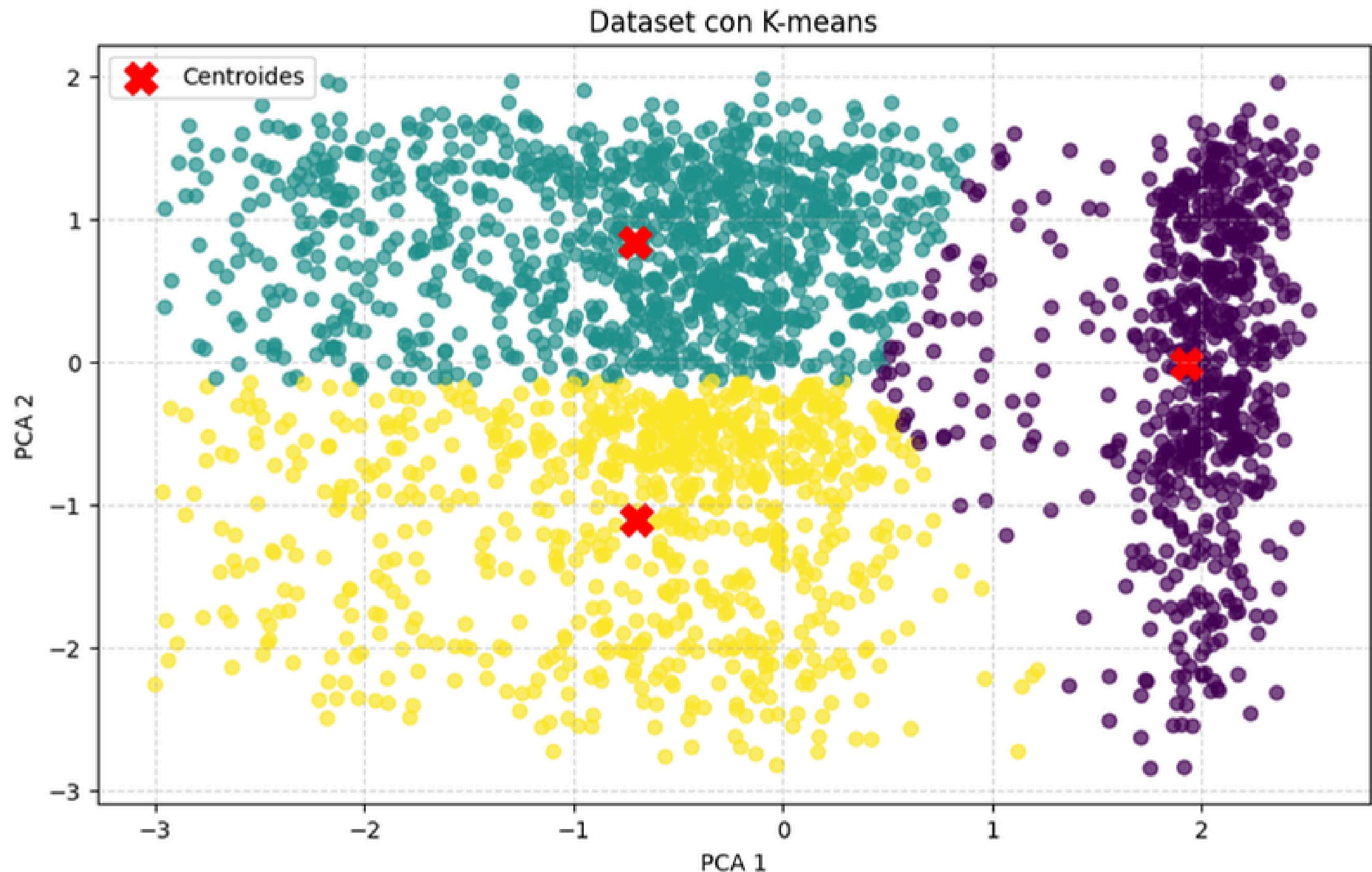


# IMPLEMENTACIÓN DE K-MEANS

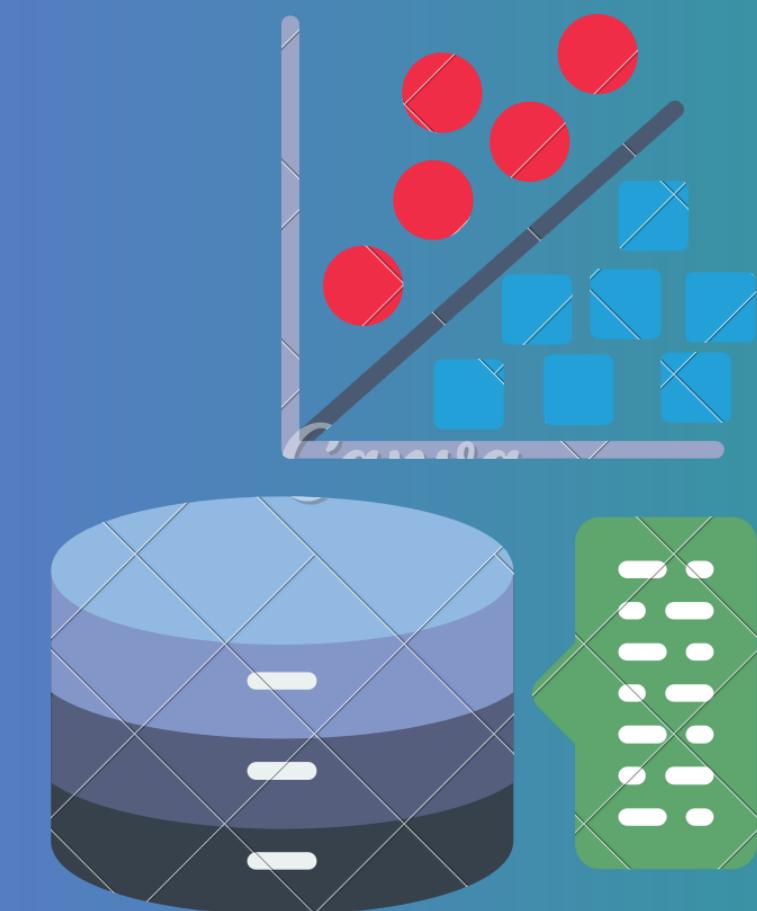


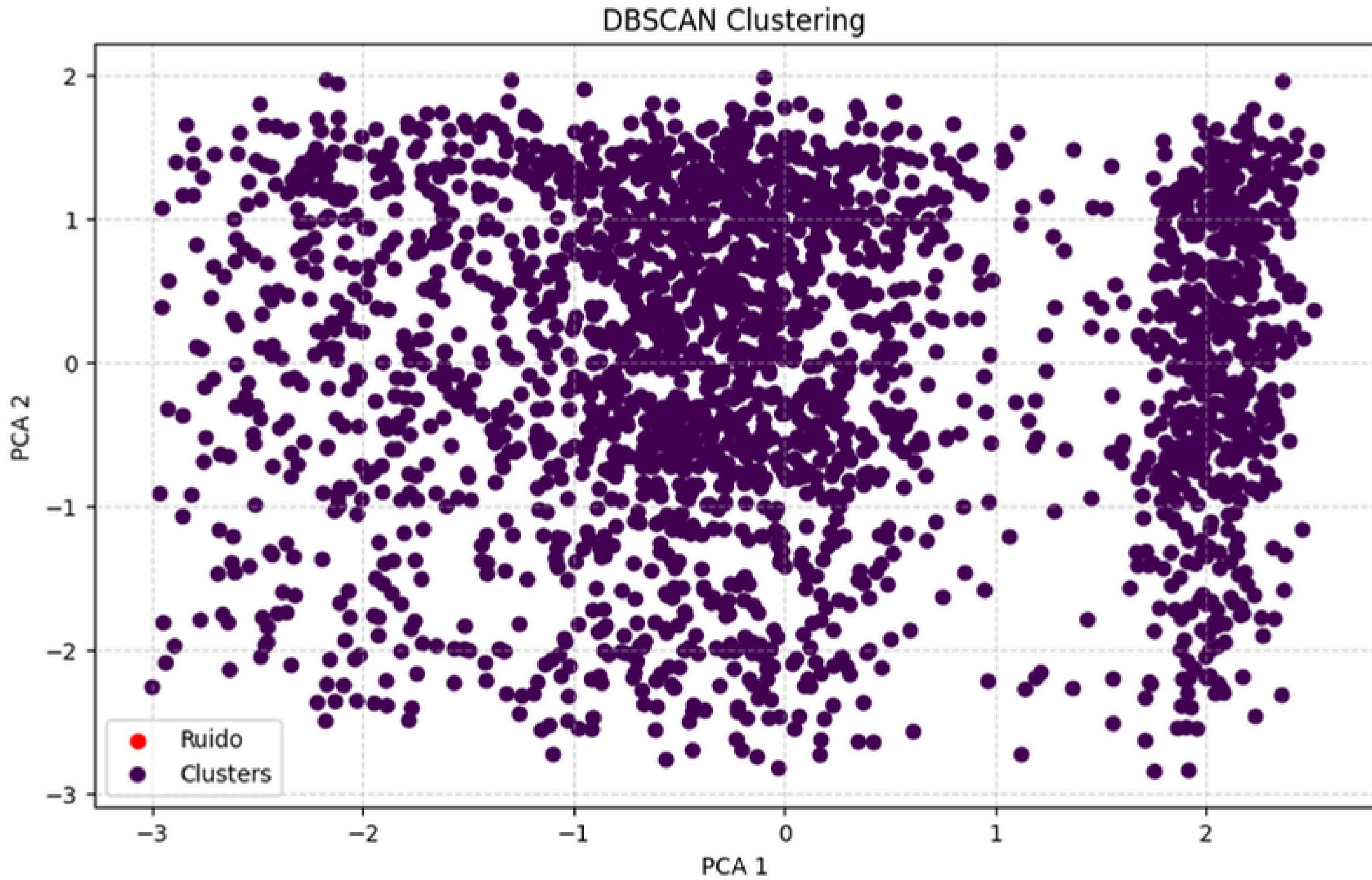
El gráfico de K-Means muestra tres clústeres definidos con sus centroides, aunque hay cierta mezcla en los bordes.

Esto indica una buena agrupación general, pero sugiere que la separación entre débito y crédito podría mejorarse ajustando los parámetros del modelo.



# IMPLEMENTACIÓN DE DBSCAN





El gráfico de DBSCAN muestra un único clúster (puntos morados) sin ruido, lo que indica que agrupó todas las transacciones juntas, posiblemente por el dominio de las transacciones de débito.

Esto sugiere que los parámetros actuales, especialmente `eps`, no permiten distinguir adecuadamente los tipos. Un ajuste fino o un enfoque supervisado podría mejorar la separación.

**GRACIAS POR SU  
ATENCIÓN**

