PROGRAMA DE INTELIGENCIA ARTIFICIAL | IBM SkillUp 2024

Proyecto Final de Inteligencia Artificial: Detección de fraudes con tarjetas de crédito.

Autor: Antonio Fernández Salcedo / linkedin

Sitio Web personal

GitHub

Hugging Face

Trabajando: Canal de Isabel II (https://www.canaldeisabelsegunda.es)

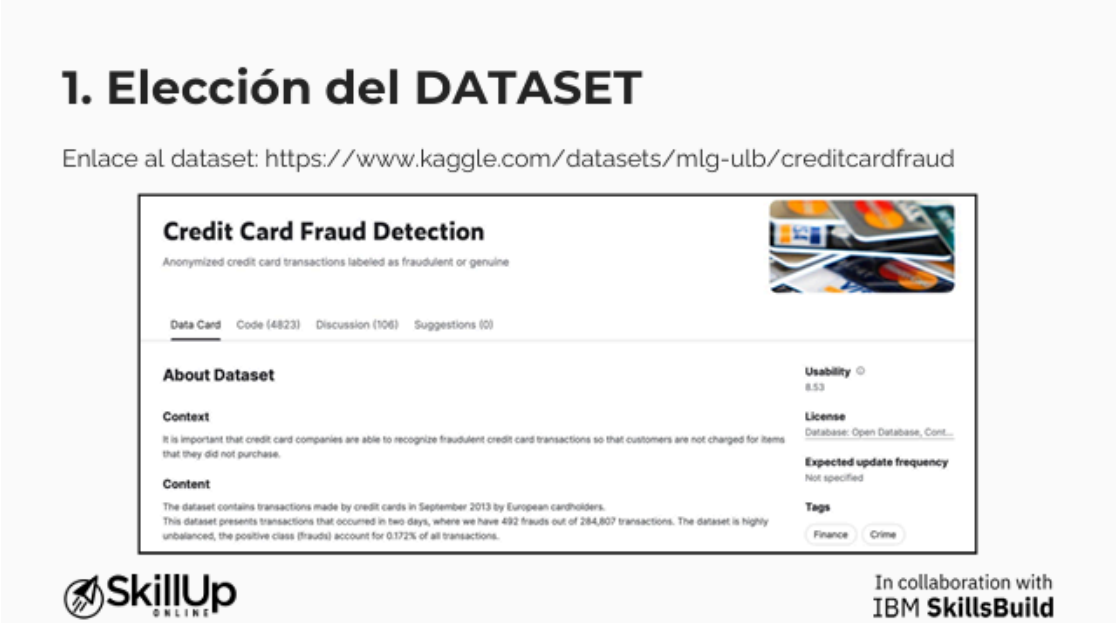
Formación: Grado Ingeniería Informática (https://www.uoc.edu/es)

Contacto: afernandezsalc@uoc.edu

Detección de fraudes con tarjetas de crédito

Elección del dataset

Enlace al dataset: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud



Importa las bibliotecas necesarias

In [1]:
```
# Importa la bibliotecas necesarias
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Importa y organiza el dataset

Importar la librería `kagglehub`. Esta librería actúa como un puente entre tu código Python y la plataforma de **Kaggle**.

In [3]:
```
import kagglehub

# Download latest version.
path = kagglehub.dataset_download("mlg-ulb/creditcardfraud")

print("Path to dataset files:", path)
```

```
Downloading from https://www.kaggle.com/api/v1/datasets/download/mlg-ulb/creditcardfraud?dataset_version_number=3...
100%|██████████| 66.0M/66.0M [00:02<00:00, 23.9MB/s]
Extracting files...

Path to dataset files: /root/.cache/kagglehub/datasets/mlg-ulb/creditcardfraud/versions/3
```

In [4]:
```
# Organizar los datos en un dataframe
filepath = "/content/drive/MyDrive/IBM/CursoIA/creditcard.csv"
```

```
data = pd.read_csv(filepath, header=0)
data.head(10)
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.
6	4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.
7	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.
8	7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.
9	9.0	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.

10 rows × 31 columns

▶ **Haz clic aquí para obtener una pista**

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

Limpia los datos

a. Valores perdidos

In [6]:
```
#Escribe tu código aquí
valores_nulos = data.isnull().sum()
print(valores_nulos)
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

▶ **Haz clic aquí para obtener una pista**

b. Datos duplicados

In [7]:
```
#Escribe tu código aquí
# Contar las filas duplicadas
filas_duplicadas = data.duplicated().sum()
print("Número de filas duplicadas:", filas_duplicadas)
```

```
Número de filas duplicadas: 1081
```

▶ **Haz clic aquí para obtener una pista**

Qué hacer con las filas duplicadas

La decisión de qué hacer con las filas duplicadas depende del contexto de tus datos y del objetivo de tu análisis.

1. Eliminar las filas duplicadas:

```
data = data.drop_duplicates()
```

1. Mantener todas las filas:

Si cada fila representa una observación única, incluso si los valores son idénticos, se pueden mantener todas las filas. Esto podría ser útil porque los duplicados tienen un significado específico en tu contexto. Pueden ser operaciones que se han repetido.

1. Investigar las duplicadas:

Antes de tomar una decisión, es recomendable investigar por qué existen duplicados:

- Errores en la recopilación de datos: Si los duplicados son errores, pueden ser eliminarlos.
- Múltiples observaciones de la misma entidad: Si los duplicados representan múltiples observaciones de la misma entidad (por ejemplo, múltiples registros de un mismo cliente), puedes agruparlos y calcular estadísticas agregadas.
- Datos duplicados intencionales: Si los duplicados son intencionales (por ejemplo, para fines de validación cruzada), puedes mantenerlos.

1. Marcar las filas duplicadas:

Si se quiere conservar toda la información, pero se necesita identificar las filas duplicadas para análisis posteriores, se puede crear una nueva columna que indique si una fila es duplicada o no:

`data['is_duplicate'] = data.duplicated()`

Impacto en el análisis:

Evalúa cómo la eliminación de duplicados podría afectar tus resultados. Si esta construyendo un modelo de machine learnin y la eliminación de duplicados reduciría el tamaño de tu conjunto de entrenamiento minimamente y no va afectar el rendimiento del modelo.

In [8]:
```
# total filas duplicadas
total_filas = len(data)
total_filas
```

Out[8]:
```
284807
```

In [9]:
```
# Porcentaje de filas duplicadas
porcentaje_duplicados = (filas_duplicadas / total_filas) * 100
print("Porcentaje de filas duplicadas:", porcentaje_duplicados, "%")
```

```
Porcentaje de filas duplicadas: 0.379555277784605 %
```

IMPORTANTE: NO SE ELIMINAN DUPLICADAS

UN EXTRA AL FINAL, TRATANDO EL TEMA DE FILAS DUPLICADAS QUE TIENE QUE VER CON LAS TRANSACIONES FRAUDALENTAS.

Porcentaje de transacciones fraudulentas: `0.1727485630620034 %`

ES UN PORCENTAJE ALTO, POR ESO DETERMINO NO ELIMINAR ESAS FILAS:

Porcentaje de transacciones duplicadas/fraudulentas con respecto al numero de transacciones fraudulentas: `3.861788617886179 %`

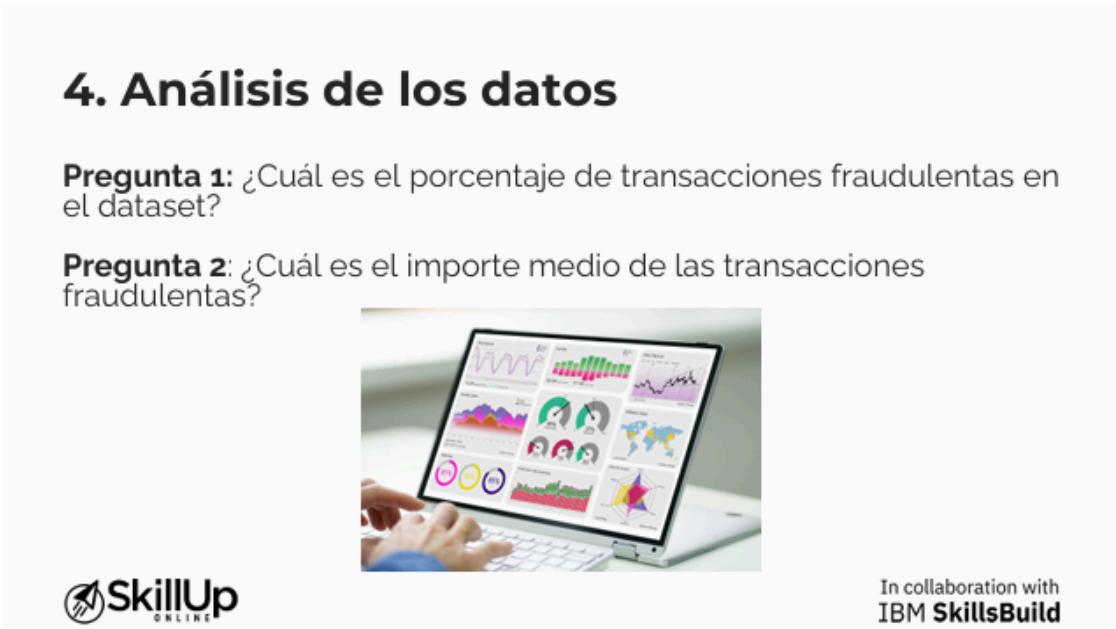
Hay una relación muy alta de que si las filas son duplicadas y el numero de transsaciones fraudulentas, el porcentaje aumenta un **22.36 veces**

En cualquier caso, lo que es evidente es que las transaciones duplicadas no es un error de duplicidad de texto, es que han repetido la transacion, y cuando repiten la transaciones aumenta un **2236 %** de que sea fraudulenta.

En el caso de eliminar duplicadas:

In [10]:
```
# data = data.drop_duplicates()
# data
```

Analiza los datos



Pregunta 1: ¿Cuál es el porcentaje de transacciones fraudulentas en el dataset?

In [11]:
```
# Calcula el porcentaje de transacciones fraudulentas
# Contar las transacciones fraudulentas
# Filtra el DataFrame para obtener solo las filas donde la columna 'Class
# shape[0]: Devuelve el número de filas (transacciones) en el DataFrame f
num_transacciones_fraudulentas = data[data['Class'] == 1].shape[0]
num_transacciones_fraudulentas
```

Out[11]:
```
492
```

In [12]:
```
# Contar el total de transacciones
total_transacciones = data.shape[0]
total_transacciones
```

Out[12]:
```
284807
```

In [13]:
```
# Calcular el porcentaje de transacciones fraudulentas
porcentaje_fraude = (num_transacciones_fraudulentas / total_transacciones


# Muestra el porcentaje de transacciones fraudulentas
print("Porcentaje de transacciones fraudulentas:", porcentaje_fraude, "%"
```

```
Porcentaje de transacciones fraudulentas: 0.1727485630620034 %
```

La mayoría de las transacciones no son fraudulentas. Si usamos este marco de datos como base para nuestros modelos predictivos y análisis, podríamos obtener muchos errores y nuestros algoritmos probablemente se sobreajustarán, ya que "asumirán" que la mayoría de las transacciones no son fraude. Esta fuera de estudio en este caso se ajusta el entrenamiento a las especificaciones dadas.

▶ **Haz clic aquí para obtener una pista**

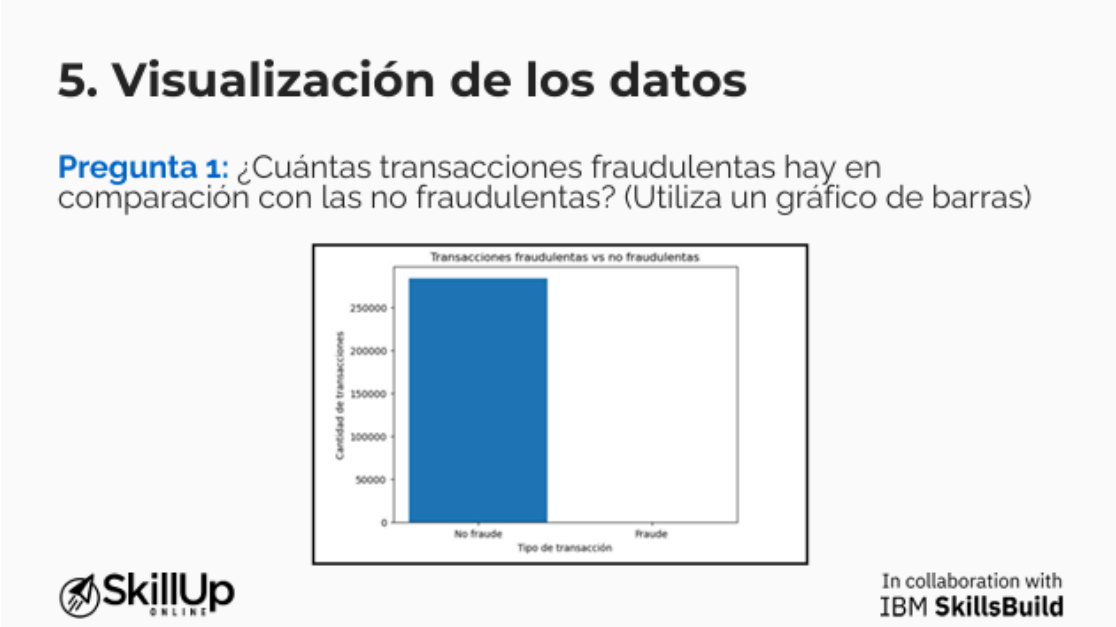
Pregunta 2: ¿Cuál es el importe medio de las transacciones fraudulentas?

In [14]:
```
# Calcula el importe medio de las transacciones fraudulentas
# Filtrar las transacciones fraudulentas
transacciones_fraudulentas = data[data['Class'] == 1]

# Calcular el importe medio de las transacciones fraudulentas
importe_medio_fraude = transacciones_fraudulentas['Amount'].mean()

# Muestra el importe medio de las transacciones fraudulentas
print(f"El importe medio de las transacciones fraudulentas es: {importe_me
```

```
El importe medio de las transacciones fraudulentas es: 122.211321
```

▶ **Haz clic aquí para obtener una pista**

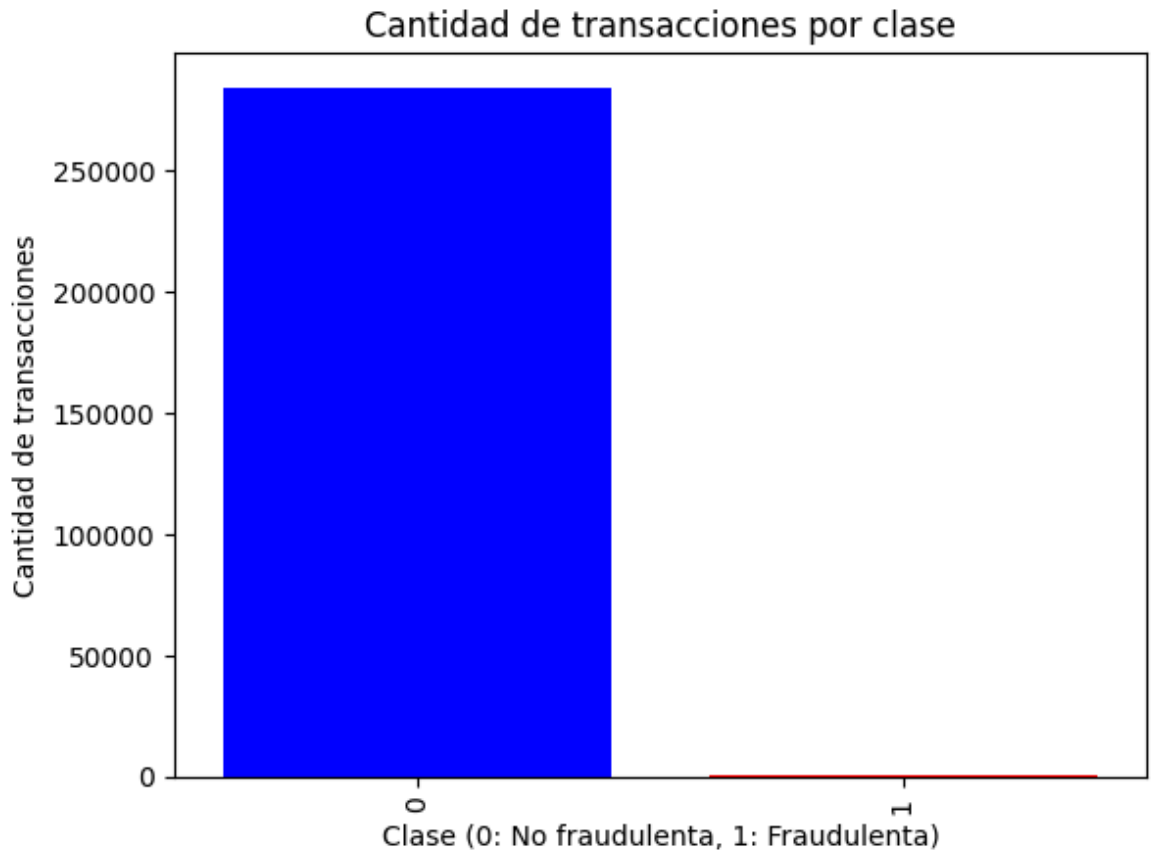
Visualiza los datos



Pregunta 1: ¿Cuántas transacciones fraudulentas hay en comparación con las no fraudulentas? (Utiliza un gráfico de barras)

In [15]:
```
# Cuenta el número de transacciones fraudulentas y no fraudulentes
conteo_clases = data['Class'].value_counts()

# Muestra la distribución de las traducciones fraudulentas con respecto de
conteo_clases.plot(kind='bar')

# Crear una lista de colores
colores = ['blue', 'red']

# Asignar los colores a cada barra
plt.bar(conteo_clases.index, conteo_clases.values, color=colores)

# Agregar título y etiquetas a los ejes
plt.title('Cantidad de transacciones por clase')
plt.xlabel('Clase (0: No fraudulenta, 1: Fraudulenta)')
```

```
plt.ylabel('Cantidad de transacciones')

# Mostrar el gráfico
plt.show()
```

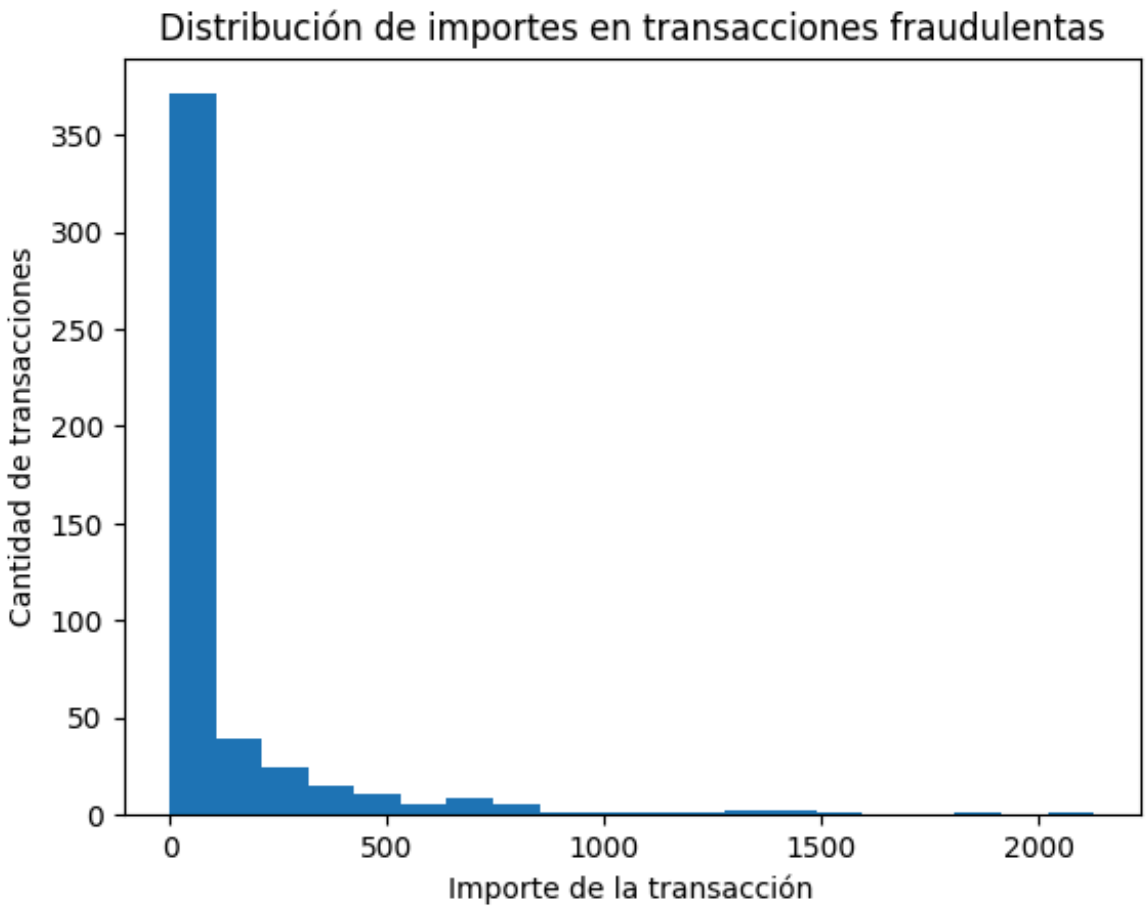


Distribuciones: Al ver las distribuciones podemos tener una idea de que están sesgadas.

▶ **Haz clic aquí para obtener una pista**

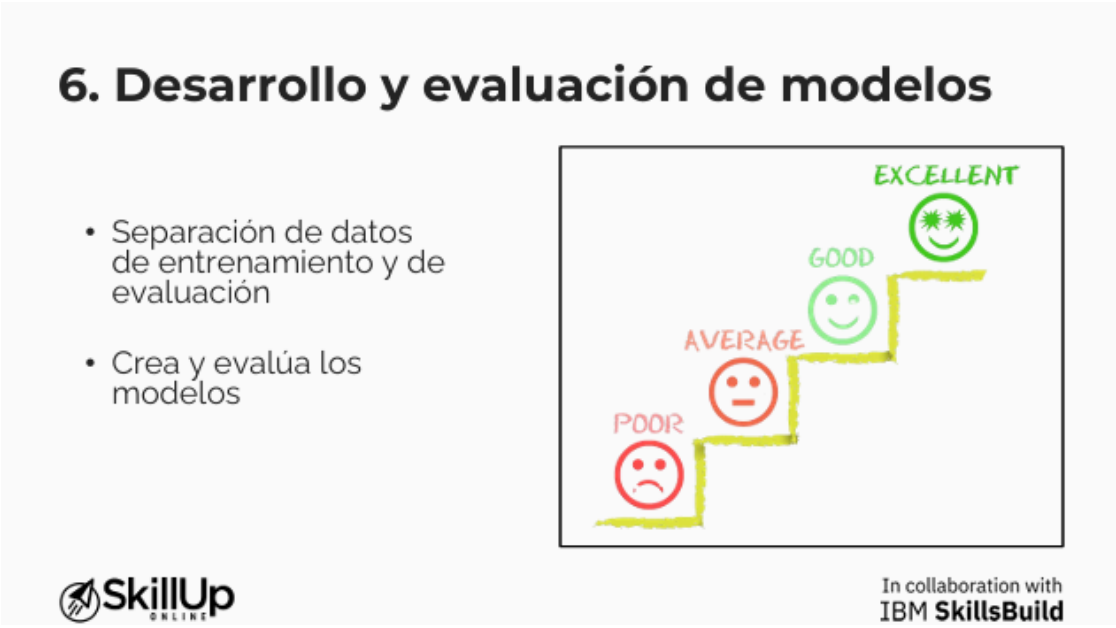
Pregunta 2: ¿Cuál es la distribución de los importes de las transacciones fraudulentas? (Utiliza un histograma)

In [16]:
```
# Separa los datos de transacciones fraudulentas
# Filtrar las transacciones fraudulentas
transacciones_fraudulentas = data[data['Class'] == 1]

# Muestra la distribución de los importes de las transacciones fraudulento
# Crear el histograma
plt.hist(transacciones_fraudulentas['Amount'], bins=20)

# Agregar título y etiquetas a los ejes
plt.title('Distribución de importes en transacciones fraudulentas')
plt.xlabel('Importe de la transacción')
plt.ylabel('Cantidad de transacciones')

# Mostrar el gráfico
plt.show()
```



▶ **Haz clic aquí para obtener una pista**

Desarrollo y evaluación de modelos



Separa del dataset

In [17]:
```
# Separa los datos de entrenamiento y evaluación
from sklearn.model_selection import train_test_split

# Separar las características (X) de la variable objetivo (y)
X = data.drop('Class', axis=1)  # Todas las columnas excepto 'Class'
y = data['Class']

# Dividir los datos en conjuntos de entrenamiento y evaluación
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

print("Forma del conjunto de entrenamiento:", X_train.shape, y_train.shape
print("Forma del conjunto de evaluación:", X_test.shape, y_test.shape)
```

```
Forma del conjunto de entrenamiento: (227845, 30) (227845,)
Forma del conjunto de evaluación: (56962, 30) (56962,)
```

▶ **Haz clic aquí para obtener una pista**

Crea y evalúa los modelos

1. Importar las bibliotecas.
2. Crear una instancia del clasificador: Creamos una instancia de RandomForestClassifier y configuramos los hiperparámetros max_depth (profundidad máxima de los árboles) y random_state (semilla para la reproducibilidad).
3. Entrenar el modelo: Utilizamos el método fit() para entrenar el modelo con los datos de entrenamiento X_train e y_train.
4. Hacer predicciones: Utilizamos el método predict() para hacer predicciones sobre los datos de prueba X_test y almacenamos las predicciones en y_pred.
5. Evaluar el modelo:

- classification_report: Genera un informe detallado con métricas como precisión, recall, F1-score para cada clase.
- accuracy_score: Calcula la precisión general del modelo, es decir, la proporción de predicciones correctas.

1. Imprimir resultados: Imprimimos el informe de clasificación y la precisión del modelo en un formato legible.

In [18]:
```
#Escribe tu código aquí
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score

# Crear una instancia del clasificador Random Forest
clf = RandomForestClassifier(max_depth=150, random_state=42)

# Entrenar el modelo
clf.fit(X_train, y_train)
```

Out[18]:
```
RandomForestClassifier(max_depth=150, random_state=42)
```

In [19]:
```
# Hacer predicciones en el conjunto de prueba
y_pred = clf.predict(X_test)

# Evaluar el modelo
print(classification_report(y_test, y_pred))

# Calcular la precisión
accuracy = accuracy_score(y_test, y_pred)
print("Precisión del modelo:", accuracy * 100, "%")
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.97      0.77      0.86        98

    accuracy                           1.00     56962
   macro avg       0.99      0.88      0.93     56962
weighted avg       1.00      1.00      1.00     56962

Precisión del modelo: 99.95611109160492 %
```

▶ **Haz clic aquí para obtener una pista**

Análisis de las Métricas del Modelo

Las métricas obtenidas proporcionan una visión bastante clara del rendimiento del modelo en la detección de transacciones fraudulentas. Vamos a analizarlas en detalle, teniendo en cuenta el contexto del problema (bajo porcentaje de transacciones fraudulentas):

Interpretación de las Métricas

1. Precisión (precision):

Clase 0 (No fraudulenta): Una precisión de 1.00 indica que todas las transacciones clasificadas como no fraudulentas realmente lo eran. Esto es muy bueno y sugiere que el modelo es muy bueno en identificar transacciones legítimas.

Clase 1 (Fraudulenta): Una precisión de 0.97 significa que el 97% de las transacciones clasificadas como fraudulentas realmente lo eran. Aunque es un buen valor, indica que hay un pequeño margen de error en la clasificación de las transacciones fraudulentas.

1. Recall (exhaustividad):

Clase 0: Un recall de 1.00 significa que el modelo identificó todas las transacciones no fraudulentas.

Clase 1: Un recall de 0.77 indica que el modelo solo identificó el 77% de las transacciones fraudulentas. Esto sugiere que el modelo podría estar pasando por alto algunas transacciones fraudulentas.

1. F1-score:

El F1-score es la media armónica de precisión y recall, proporcionando un equilibrio entre ambos. Un F1-score de 0.86 para la clase 1 indica un buen rendimiento general, pero sugiere que aún hay margen de mejora en la detección de todas las transacciones fraudulentas.

1. Accuracy:

La precisión general del modelo es extremadamente alta (99.95%). Sin embargo, debido al **desbalance de clases** (muy pocas transacciones fraudulentas), esta métrica puede ser engañosa. Es decir, el modelo podría estar clasificando casi todas las transacciones como no fraudulentas y aún así obtener una alta precisión.

Conclusiones y Consideraciones

1. Desbalance de clases: El principal desafío en este tipo de problemas es el desbalance de clases. El modelo está muy bien en identificar transacciones legítimas, pero tiene dificultades para identificar las fraudulentas, que son mucho menos frecuentes.
2. Recall como métrica clave: En este contexto, el recall es una métrica más importante que la precisión. Queremos asegurarnos de identificar la mayor

cantidad posible de transacciones fraudulentas, incluso si eso significa algunos falsos positivos.

Posible mejora:

Sobremuestreo o submuestreo: Aumentar el número de ejemplos de la clase minoritaria (fraudulenta) o reducir el número de ejemplos de la clase mayoritaria (no fraudulenta) para equilibrar el dataset.

En resumen, el modelo presenta un excelente rendimiento en la identificación de transacciones legítimas, pero podría mejorar en la detección de transacciones fraudulentas. Dada la importancia de identificar todas las transacciones fraudulentas, es crucial enfocarse en mejorar el recall para la clase minoritaria.

Resumen comparativo:

Precisión del modelo **SIN** duplicados: `99.95418179254926 %`

Precisión del modelo **CON** duplicados: `99.95611109160492 %`

1. Baja tasa de fraude: el porcentaje de transacciones fraudulentas es bajo, lo cual es típico en conjuntos de datos de fraude.
2. Análisis de duplicados: Se ha detectado relación entre transacciones duplicadas y fraude, lo cual es un hallazgo interesante.
3. Cálculo de porcentajes: Se han cuantificadon las relaciones entre las variables, lo que facilita la interpretación de los resultados.
4. La presión del modelo es practicamente el mismo, con o sin duplicados, con lo cual no va a variar en las prediciones del modelo.
5. Creación de reglas: Se pueden establecer reglas para identificar transacciones potencialmente fraudulentas basadas en la duplicidad de datos. Por ejemplo, si se detecta una transacción duplicada, se puede marcar como sospechosa y someterla a una revisión manual.
6. Mejora de modelos predictivos: La duplicidad puede incorporarse como una característica adicional en los modelos de aprendizaje automático para mejorar su capacidad de detectar fraudes. Por ejemplo, se puede crear una variable binaria que indique si una transacción es duplicada o no, y utilizarla como entrada en el modelo.
7. el conocimiento de que las filas duplicadas están asociadas con un mayor riesgo de fraude proporciona una valiosa información que puede ser

utilizada para mejorar los sistemas de detección de fraude, reducir las pérdidas financieras y proteger a los clientes.

Extra:

In [20]:
```
# Incluimos dos nuevas columas'
data['is_duplicate'] = data.duplicated()
data['is_fraud'] = data['Class'] == 1
data
```

Out[20]:

	Time	V1	V2	V3	V4	V5	V6	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.2
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.(
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.7
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.2
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.5
...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.9
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.(
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.2
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.(
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.5

284807 rows × 33 columns

Examinar las trasaciones fraudulentas y duplicadas:

In [21]:
```
# Filtrando por transacciones fraudulentas y duplicadas
fraud_dup_counts = ((data['is_fraud'] == True) & (data['is_duplicate'] ==
fraud_dup_counts
```

Out[21]:

	count
False	284788
True	19

dtype: int64

Hay 19 transaciones que coinciden entre duplicadas y fraudulentas.

In [22]:
```
tabla= pd.crosstab(data['is_duplicate'], data['is_fraud'])
tabla
```

Out[22]:

is_fraud	False	True
is_duplicate		
False	283253	473
True	1062	19

In [23]:
```
num_transacciones_fraudulentas
```

Out[23]:
```
492
```

In [24]:
```
filas_duplicadas
```

Out[24]:
```
1081
```

In [25]:
```
# Filtrando por transacciones fraudulentas y duplicadas
fraud_dup_counts2 = ((data['is_fraud'] == True) & (data['is_duplicate'] ==
fraud_dup_counts2
```

Out[25]:
```
19
```

In [26]:
```
# Calcular el porcentaje de transacciones duplicadas/fraudulentas con resp
porcentaje_fraude2 = ( fraud_dup_counts2 / num_transacciones_fraudulentas)

# Muestra el porcentaje de transacciones duplicadas/fraudulentas con respe
print("Porcentaje de transacciones duplicadas/fraudulentas con respecto al
```

```
Porcentaje de transacciones duplicadas/fraudulentas con respecto al numer
o de transacciones fraudulentas: 3.861788617886179 %
```

In [27]:
```
# Calcular el porcentaje de transacciones duplicadas/fraudulentas con resp
porcentaje_fraude3 = (fraud_dup_counts2  / filas_duplicadas) * 100

# Muestra el porcentaje de transacciones duplicadas/fraudulentas con respe
print("Porcentaje de transacciones duplicadas/fraudulentas con respecto al
```

```
Porcentaje de transacciones duplicadas/fraudulentas con respecto al numer
o de filas duplicadas: 1.757631822386679 %
```

In [28]:
```
# Calcular el porcentaje de transacciones duplicadas/fraudulentas con resp
porcentaje_fraude4 = ( fraud_dup_counts2 / total_filas) * 100

# Muestra el porcentaje de transacciones duplicadas/fraudulentas con respe
print("Porcentaje de transacciones duplicadas/fraudulentas con respecto al
```

```
Porcentaje de transacciones duplicadas/fraudulentas con respecto al numer
o de transacciones totales: 0.006671184345890375 %
```

In [29]:
```
# Calcular el porcentaje de transacciones fraudulentas
porcentaje_fraude = (num_transacciones_fraudulentas / total_transacciones)
# Muestra el porcentaje de transacciones fraudulentas
print("Porcentaje de transacciones fraudulentas:", porcentaje_fraude, "%")
```

```
Porcentaje de transacciones fraudulentas: 0.1727485630620034 %
```

In [30]:
```
pd.crosstab(data['is_duplicate'], data['is_fraud'])
sns.countplot(x='is_duplicate', hue='is_fraud', data=data)
plt.show()
```