Detección de fraudes con tarjetas de crédito

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

Importa las bibliotecas necesarias

In [5]:
```
# 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")
```

Importa y organiza el dataset

In [7]:
```
# Organizar los datos en un dataframe
# Importamos la librería pandas
import pandas as pd
```

```
# Cargamos el archivo CSV en un DataFrame. Asegúrate de sustituir "ruta_al_archi
data = pd.read_csv("creditcard.csv")

# Mostramos las primeras 10 filas del DataFrame para revisar cómo se cargaron lo
# Esto es útil para obtener una vista preliminar de los datos.
print(data.head(10))
```

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

         V8        V9  ...       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -0.206010
5  0.260314 -0.568671  ... -0.208254 -0.559825 -0.026398 -0.371427 -0.232794
6  0.081213  0.464960  ... -0.167716 -0.270710 -0.154104 -0.780055  0.750137
7 -3.807864  0.615375  ...  1.943465 -1.015455  0.057504 -0.649709 -0.415267
8  0.851084 -0.392048  ... -0.073425 -0.268092 -0.204233  1.011592  0.373205
9  0.069539 -0.736727  ... -0.246914 -0.633753 -0.120794 -0.385050 -0.069733

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0
5  0.105915  0.253844  0.081080    3.67      0
6 -0.257237  0.034507  0.005168    4.99      0
7 -0.051634 -1.206921 -1.085339   40.80      0
8 -0.384157  0.011747  0.142404   93.20      0
9  0.094199  0.246219  0.083076    3.68      0

[10 rows x 31 columns]
```

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

Limpia los datos

a. Valores perdidos

In [11]:
```
# Contamos los valores nulos en cada columna utilizando isnull() y sum()
# El resultado será un resumen de la cantidad de valores nulos por columna
print(data.isnull().sum())

# Eliminar filas que tengan cualquier valor nulo
data_cleaned = data.dropna()
print(data_cleaned.head(10))
```

```
# Reemplazar los valores nulos con la media de cada columna numérica
data_filled = data.fillna(data.mean())
print(data_filled.head(10))
```

```
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
```

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

	V8	V9	...	V21	V22	V23	V24	V25
0	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539
1	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170
2	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642
3	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376
4	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010
5	0.260314	-0.568671	...	-0.208254	-0.559825	-0.026398	-0.371427	-0.232794
6	0.081213	0.464960	...	-0.167716	-0.270710	-0.154104	-0.780055	0.750137
7	-3.807864	0.615375	...	1.943465	-1.015455	0.057504	-0.649709	-0.415267
8	0.851084	-0.392048	...	-0.073425	-0.268092	-0.204233	1.011592	0.373205
9	0.069539	-0.736727	...	-0.246914	-0.633753	-0.120794	-0.385050	-0.069733

	V26	V27	V28	Amount	Class
0	-0.189115	0.133558	-0.021053	149.62	0
1	0.125895	-0.008983	0.014724	2.69	0
2	-0.139097	-0.055353	-0.059752	378.66	0

```
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0
5  0.105915  0.253844  0.081080    3.67      0
6 -0.257237  0.034507  0.005168    4.99      0
7 -0.051634 -1.206921 -1.085339   40.80      0
8 -0.384157  0.011747  0.142404   93.20      0
9  0.094199  0.246219  0.083076    3.68      0

[10 rows x 31 columns]
   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
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941
5   2.0 -0.425966  0.960523  1.141109 -0.168252  0.420987 -0.029728  0.476201
6   4.0  1.229658  0.141004  0.045371  1.202613  0.191881  0.272708 -0.005159
7   7.0 -0.644269  1.417964  1.074380 -0.492199  0.948934  0.428118  1.120631
8   7.0 -0.894286  0.286157 -0.113192 -0.271526  2.669599  3.721818  0.370145
9   9.0 -0.338262  1.119593  1.044367 -0.222187  0.499361 -0.246761  0.651583

         V8        V9  ...       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -0.206010
5  0.260314 -0.568671  ... -0.208254 -0.559825 -0.026398 -0.371427 -0.232794
6  0.081213  0.464960  ... -0.167716 -0.270710 -0.154104 -0.780055  0.750137
7 -3.807864  0.615375  ...  1.943465 -1.015455  0.057504 -0.649709 -0.415267
8  0.851084 -0.392048  ... -0.073425 -0.268092 -0.204233  1.011592  0.373205
9  0.069539 -0.736727  ... -0.246914 -0.633753 -0.120794 -0.385050 -0.069733

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0
5  0.105915  0.253844  0.081080    3.67      0
6 -0.257237  0.034507  0.005168    4.99      0
7 -0.051634 -1.206921 -1.085339   40.80      0
8 -0.384157  0.011747  0.142404   93.20      0
9  0.094199  0.246219  0.083076    3.68      0

[10 rows x 31 columns]
```

► Haz clic aquí para obtener una pista

b. Datos duplicados

In [14]:
```
# Contamos el número de filas duplicadas en el DataFrame
# La función duplicated() devuelve un booleano por fila, y sum() cuenta los valo
num_duplicados = data.duplicated().sum()
print(f"Número de filas duplicadas: {num_duplicados}")

# Eliminar las filas duplicadas del DataFrame, dejando solo las filas únicas
data_cleaned = data.drop_duplicates()
```

```
# Mostrar las primeras 10 filas del DataFrame limpio
print("Primeras 10 filas del DataFrame sin duplicados:")
print(data_cleaned.head(10))

# Verificar si 'columna_1' existe en el DataFrame
if 'columna_1' in data.columns:
    # Eliminar filas duplicadas basadas en la columna 'columna_1'
    data_cleaned_columna = data.drop_duplicates(subset=["columna_1"])

    # Mostrar las primeras 10 filas del DataFrame limpio basado en la columna 'c
    print("Primeras 10 filas del DataFrame sin duplicados en 'columna_1':")
    print(data_cleaned_columna.head(10))
else:
    print("La columna 'columna_1' no existe en el DataFrame.")
```

```
Número de filas duplicadas: 1081
Primeras 10 filas del DataFrame sin duplicados:
   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
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941
5   2.0 -0.425966  0.960523  1.141109 -0.168252  0.420987 -0.029728  0.476201
6   4.0  1.229658  0.141004  0.045371  1.202613  0.191881  0.272708 -0.005159
7   7.0 -0.644269  1.417964  1.074380 -0.492199  0.948934  0.428118  1.120631
8   7.0 -0.894286  0.286157 -0.113192 -0.271526  2.669599  3.721818  0.370145
9   9.0 -0.338262  1.119593  1.044367 -0.222187  0.499361 -0.246761  0.651583

         V8        V9  ...       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -0.206010
5  0.260314 -0.568671  ... -0.208254 -0.559825 -0.026398 -0.371427 -0.232794
6  0.081213  0.464960  ... -0.167716 -0.270710 -0.154104 -0.780055  0.750137
7 -3.807864  0.615375  ...  1.943465 -1.015455  0.057504 -0.649709 -0.415267
8  0.851084 -0.392048  ... -0.073425 -0.268092 -0.204233  1.011592  0.373205
9  0.069539 -0.736727  ... -0.246914 -0.633753 -0.120794 -0.385050 -0.069733

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0
5  0.105915  0.253844  0.081080    3.67      0
6 -0.257237  0.034507  0.005168    4.99      0
7 -0.051634 -1.206921 -1.085339   40.80      0
8 -0.384157  0.011747  0.142404   93.20      0
9  0.094199  0.246219  0.083076    3.68      0

[10 rows x 31 columns]
La columna 'columna_1' no existe en el DataFrame.
```

▶ Haz clic aquí para obtener una pista

Analiza los datos

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

In [18]:
```
# Calcula el porcentaje de transacciones fraudulentas
# Contamos cuántas transacciones son fraudulentas (donde 'Class' es igual a 1)
fraudulent_transactions = data['Class'].sum()

# Contamos el número total de transacciones
total_transactions = len(data)


# Muestra el porcentaje de transacciones fraudulentas
# Calculamos el porcentaje de transacciones fraudulentas
fraudulent_percentage = (fraudulent_transactions / total_transactions) * 100

# Imprimimos el resultado
print(f"El porcentaje de transacciones fraudulentas es: {fraudulent_percentage:.
```

```
El porcentaje de transacciones fraudulentas es: 0.17%
```

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

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

In [21]:
```
# Calcula el importe medio de las transacciones fraudulentas
# Filtrar las transacciones fraudulentas (donde 'Class' es igual a 1)
fraudulent_data = data[data['Class'] == 1]

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


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

```
El importe medio de las transacciones fraudulentas es: 122.21
```

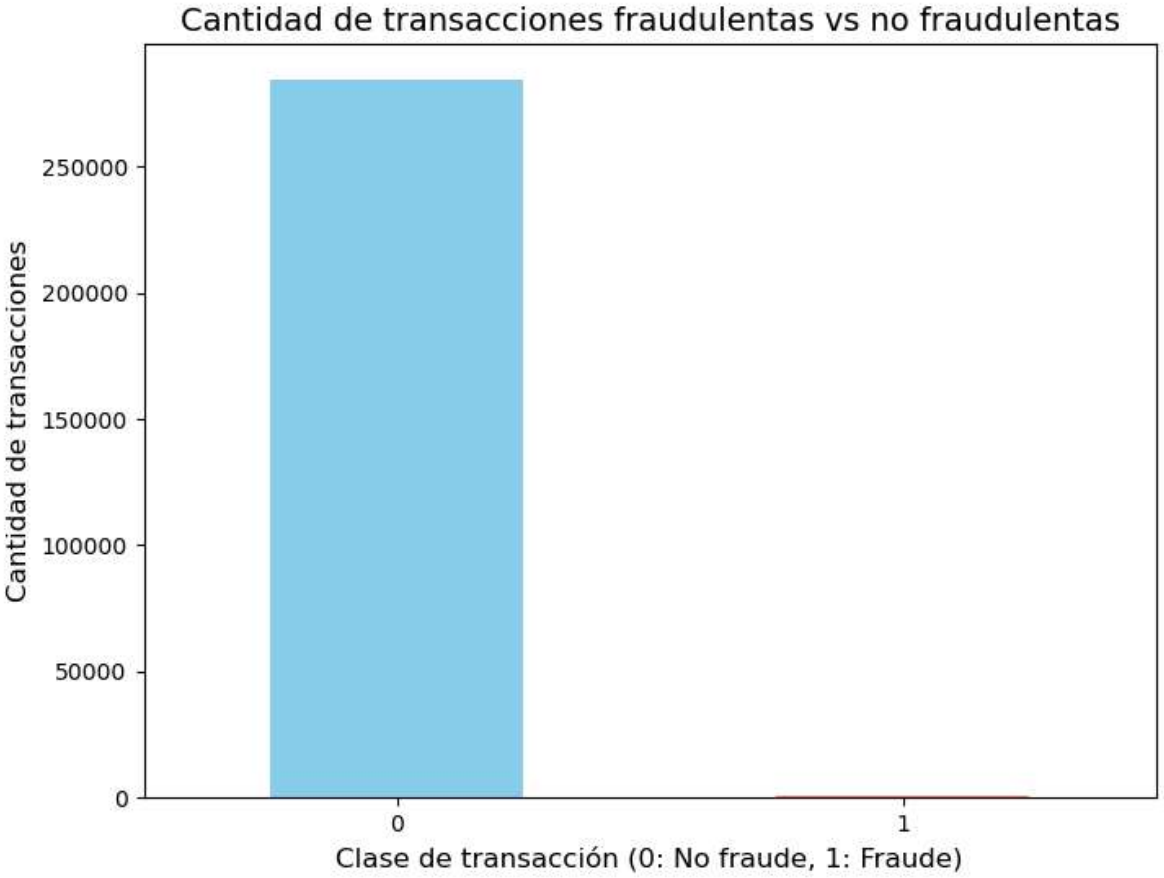
▶ **Haz clic aquí para obtener una pista**

Visualiza los datos

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

In [25]:
```
# Cuenta el número de transacciones fraudulentas y no fraudulentes

import matplotlib.pyplot as plt

# Contamos las transacciones fraudulentas (Class = 1) y no fraudulentas (Class =
transaction_counts = data['Class'].value_counts()


# Muestra la distribución de las traducciones fraudulentas con respecto de las n
# Creamos el gráfico de barras
plt.figure(figsize=(8, 6))
transaction_counts.plot(kind='bar', color=['skyblue', 'salmon'])

# Añadimos etiquetas y título al gráfico
```

```
plt.title('Cantidad de transacciones fraudulentas vs no fraudulentas', fontsize=
plt.xlabel('Clase de transacción (0: No fraude, 1: Fraude)', fontsize=12)
plt.ylabel('Cantidad de transacciones', fontsize=12)

# Mostramos el gráfico
plt.xticks(rotation=0)  # Para que las etiquetas en el eje x no estén rotadas
plt.show()
```

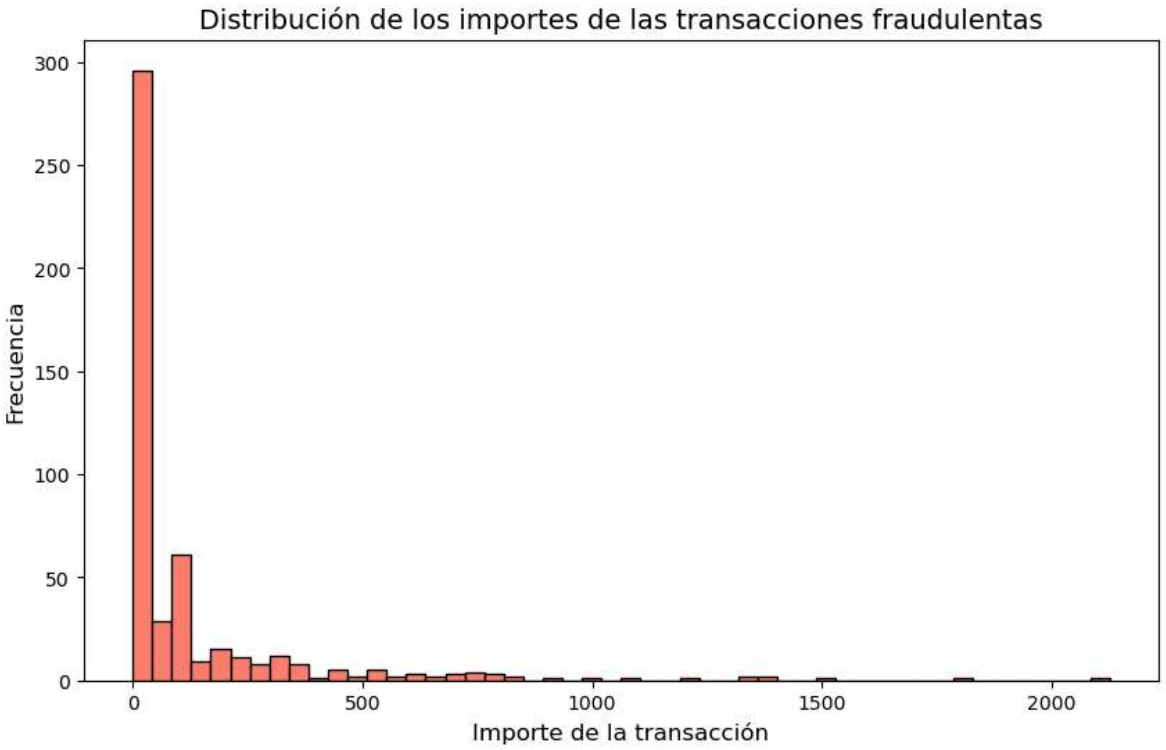


▶ **Haz clic aquí para obtener una pista**

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

In [28]:
```
# Separa los datos de transacciones fraudulentas

import matplotlib.pyplot as plt

# Filtrar las transacciones fraudulentas (donde 'Class' es igual a 1)
fraudulent_data = data[data['Class'] == 1]

# Muestra la distribución de los importes de las transacciones fraudulentas

# Crear el histograma para la columna 'Amount' de las transacciones fraudulentas
plt.figure(figsize=(10, 6))
plt.hist(fraudulent_data['Amount'], bins=50, color='salmon', edgecolor='black')

# Añadir título y etiquetas al gráfico
plt.title('Distribución de los importes de las transacciones fraudulentas', font
plt.xlabel('Importe de la transacción', fontsize=12)
plt.ylabel('Frecuencia', fontsize=12)
```

```
# Mostrar el gráfico
plt.show()
```



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

Desarrollo y evaluación de modelos

Separa del dataset

In [32]:
```
# Separa los datos de entrenamiento y evaluación

from sklearn.model_selection import train_test_split

# Crear X (todas las columnas excepto 'Class') y y (solo la columna 'Class')
X = data.drop(columns=['Class'])
y = data['Class']

# Dividir el dataset en grupos de entrenamiento y evaluación (80% entrenamiento,
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Verificar las dimensiones de los datos resultantes
print(f"Dimensiones de X_train: {X_train.shape}")
print(f"Dimensiones de X_test: {X_test.shape}")
print(f"Dimensiones de y_train: {y_train.shape}")
print(f"Dimensiones de y_test: {y_test.shape}")
```

```
Dimensiones de X_train: (227845, 30)
Dimensiones de X_test: (56962, 30)
Dimensiones de y_train: (227845,)
Dimensiones de y_test: (56962,)
```

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

Crea y evalúa los modelos

In [35]:
```
# Importar las bibliotecas necesarias
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score

# Crear una instancia del clasificador RandomForest con los hiperparámetros max_
rf_model = RandomForestClassifier(max_depth=150, random_state=42)

# Entrenar el modelo con los datos de entrenamiento
rf_model.fit(X_train, y_train)

# Hacer predicciones sobre el conjunto de evaluación
y_pred = rf_model.predict(X_test)

# Mostrar el reporte de clasificación (precisión, recall, f1-score por cada clas
print("Reporte de Clasificación:\n")
print(classification_report(y_test, y_pred))

# Calcular la exactitud del modelo
accuracy = accuracy_score(y_test, y_pred)

# Mostrar la exactitud del modelo en porcentaje
print(f"Exactitud del modelo: {accuracy * 100:.2f}%")
```

```
Reporte de Clasificación:

              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

Exactitud del modelo: 99.96%
```

In [ ]:
```
```

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