m0upmaxyh

December 6, 2024

1 Detección de fraudes con tarjetas de crédito

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

1.0.1 Importamos las bibliotecas necesarias

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

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

1.0.2 Importamos y organizamos el dataset

[2]:
```
# Organizar los datos en un dataframe
df = pd.read_csv("creditcard.csv")
df.head()
```

[2]:

	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	

	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	

	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 rows x 31 columns]
```

1.0.3 Limpiamos los datos

a. Valores perdidos

[3]:
```
lost = df.isnull().sum()
```

b. Datos duplicados

[4]:
```
duplicate = df.duplicated().sum()
print (f'Hay {duplicate} valores duplicados')
```

```
Hay 1081 valores duplicados
```

[5]:
```
# Limpiamos los datos duplicados
df_clean = df.drop_duplicates()
df_clean
```

[5]:

	Time	V1	V2	V3	V4	V5	\
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	
…	…	…	…	…	…	…	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	

	V6	V7	V8	V9	…	V21	V22	\
0	0.462388	0.239599	0.098698	0.363787	…	-0.018307	0.277838	
1	-0.082361	-0.078803	0.085102	-0.255425	…	-0.225775	-0.638672	
2	1.800499	0.791461	0.247676	-1.514654	…	0.247998	0.771679	
3	1.247203	0.237609	0.377436	-1.387024	…	-0.108300	0.005274	
4	0.095921	0.592941	-0.270533	0.817739	…	-0.009431	0.798278	
…	…	…	…	…	…	…	…	
284802	-2.606837	-4.918215	7.305334	1.914428	…	0.213454	0.111864	
284803	1.058415	0.024330	0.294869	0.584800	…	0.214205	0.924384	
284804	3.031260	-0.296827	0.708417	0.432454	…	0.232045	0.578229	
284805	0.623708	-0.686180	0.679145	0.392087	…	0.265245	0.800049	
284806	-0.649617	1.577006	-0.414650	0.486180	…	0.261057	0.643078	

```
              V23       V24       V25       V26       V27       V28  Amount  \
0       -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62
1        0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69
2        0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66
3       -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50
4       -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153   69.99
…             …         …         …         …         …         …     …
284802  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731    0.77
284803  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527   24.79
284804 -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561   67.88
284805 -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533   10.00
284806  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649  217.00

        Class
0           0
1           0
2           0
3           0
4           0
…       …
284802      0
284803      0
284804      0
284805      0
284806      0

[283726 rows x 31 columns]
```

[6]:
```python
# Comprobamos la limpieza del DataFrame
df_clean.dropna(inplace=True)
nan_count = df_clean.isnull().sum().sum()
print(f'Hay {nan_count} valores nulos en el DataFrame')
```

```
Hay 0 valores nulos en el DataFrame
```

1.0.4 Analizamos los datos

Porcentaje de transacciones fraudulentas en el dataset

[7]:
```python
# Calcular el porcentaje de transacciones fraudulentas
fraud = (df_clean['Class']==1).sum()
total = df_clean['Class'].count()

# Mostrar el porcentaje de transacciones fraudulentas
print(f"El porcentaje de transacciones fraudulentas es: {(fraud/total)*100}")
```

```
El porcentaje de transacciones fraudulentas es: 0.1667101358352777
```

Importe medio de las transacciones fraudulentas

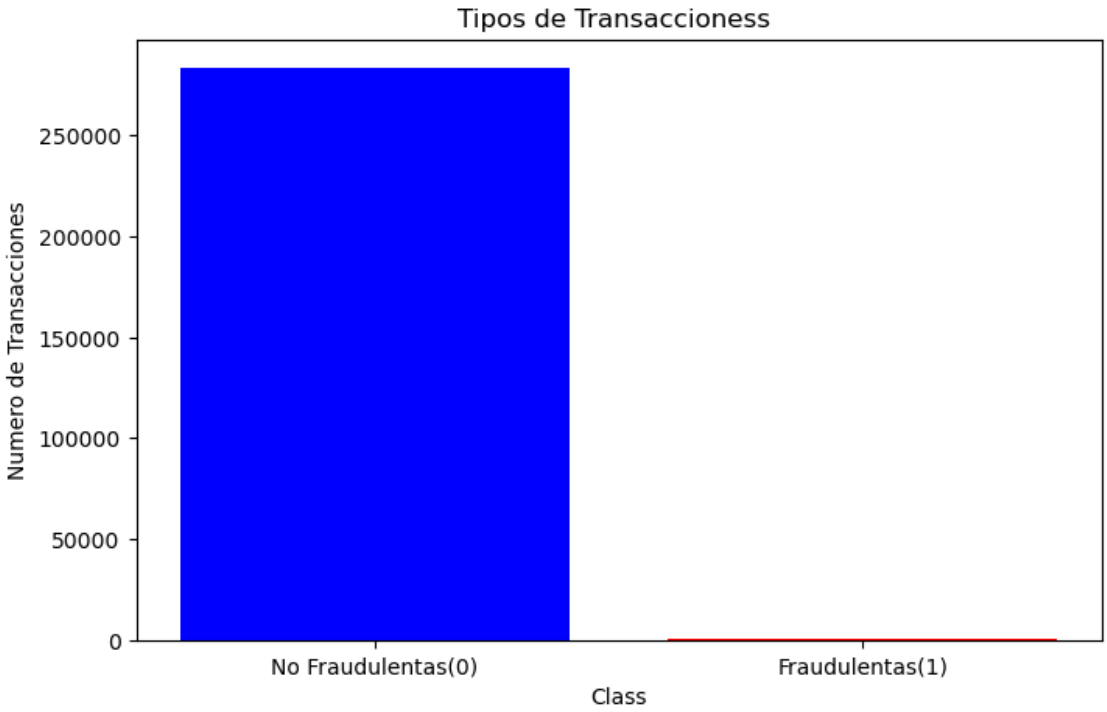
[8]:
```
# Calcular el importe medio de las transacciones fraudulentas
df_fraud = df_clean[(df_clean['Class'] == 1)]
fraud_mean = df_fraud['Amount'].mean()

# Mostrar el importe medio de las transacciones fraudulentas
print(f"Importe medio de las transacciones fraudulentas: {round(fraud_mean,␣
↪2)}$")
```

```
Importe medio de las transacciones fraudulentas: 123.87$
```

1.0.5 Visualizamos los datos

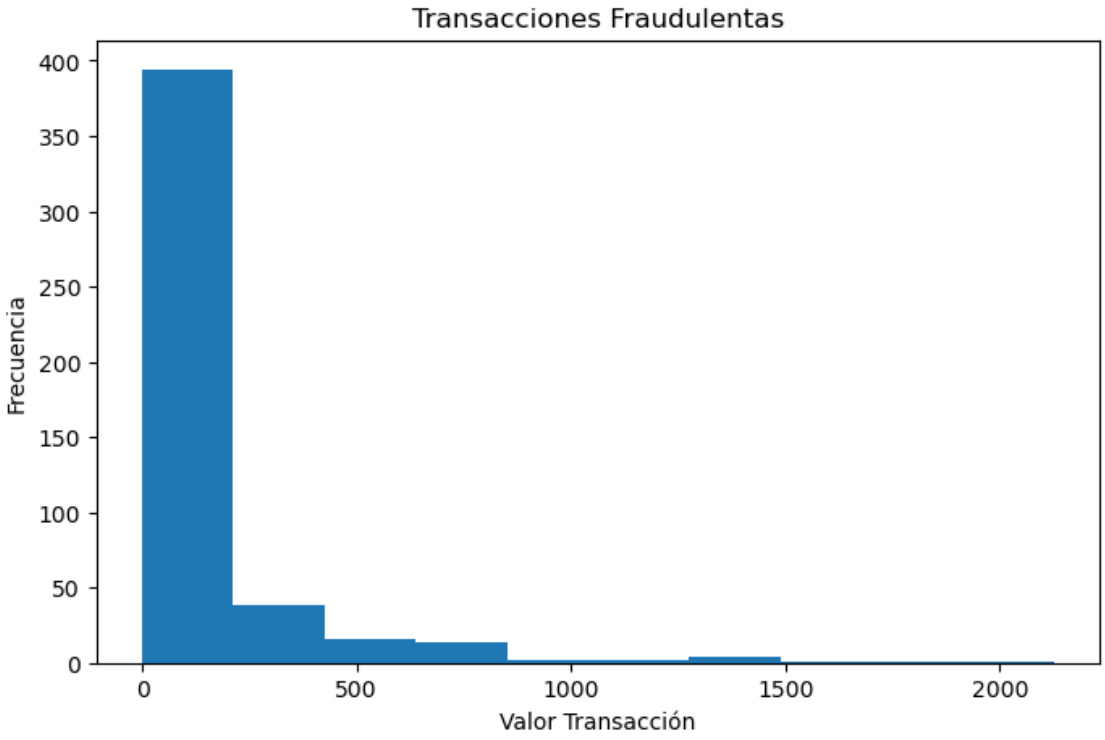
Utilizamos un gráfico de barras para comparar las transacciones fraudulentas con las no fraudulentas

[9]:
```
# Cuenta el número de transacciones fraudulentas y no fraudulentas
classC = df_clean['Class'].value_counts()
plt.figure(figsize=(8, 5))
plt.bar(classC.index, classC.values, color=['blue','red'])
plt.xlabel('Class')
plt.ylabel('Numero de Transacciones')
plt.title('Tipos de Transaccioness ')
plt.xticks([0,1], ['No Fraudulentas(0)', 'Fraudulentas(1)'])

# Muestra la distribución de las traducciones fraudulentas con respecto de las␣
↪no fraudulentas
plt.show()
```



Ahora nos valemos de un histograma para visualizar distribución de los importes de las transacciones fraudulentas

[10]:
```
# Separa los datos de transacciones fraudulentas
df_fraud = df_clean[(df_clean['Class'] == 1)]

# Muestra la distribución de los importes de las transacciones fraudulentas
plt.figure(figsize=(8, 5))
plt.hist(df_fraud['Amount'])
plt.xlabel('Valor Transacción')
plt.ylabel('Frecuencia')
plt.title('Transacciones Fraudulentas')
plt.show()
```



1.1 Desarrollo y evaluación de modelos

1.1.1 Preparamos el dataset

[11]:
```
# Separamos los datos de entrenamiento y evaluación
from sklearn.model_selection import train_test_split

X = df_clean.drop('Class', axis =1)
y = df_clean['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

1.1.2 Creamos y evaluamos los modelos

[12]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report,␣
 ↪accuracy_score,precision_score,recall_score,f1_score

model = RandomForestClassifier( n_estimators=5, max_depth=150, random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
y_pred
```

[12]:
```
array([0, 0, 0, …, 0, 0, 0], dtype=int64)
```

1.1.3 Evaluamos la precisión general

Utilizamos la función accuracy_score(), que calcula la proporción de predicciones correctas en relación con el total de predicciones. La precisión indica el porcentaje de todas las transacciones correctamente clasificadas (fraudulentas y no fraudulentas)

[13]:
```
precision = accuracy_score(y_test, y_pred)
print(f"La precisión general del modelo es de {round(precision * 100, 2)}%")
```

```
La precisión general del modelo es de 99.95%
```

1.1.4 Reporte de clasificación

La función classification_report() genera un reporte detallado con varias métricas de rendimiento, como la precisión, el recall y el F1-score, para cada clase (fraudulenta y no fraudulenta).

- Precision: La proporción de verdaderos positivos sobre el total de predicciones positivas. Es decir, de todas las veces que el modelo predijo que una transacción era fraudulenta, cuántas realmente lo eran.
- Recall: La proporción de verdaderos positivos sobre el total de casos positivos reales. Es decir, de todas las transacciones fraudulentas, cuántas fueron correctamente identificadas por el modelo.
- F1-score: La media armónica entre la precisión y el recall, que da un valor más equilibrado cuando se tiene un conjunto de clases desequilibradas.

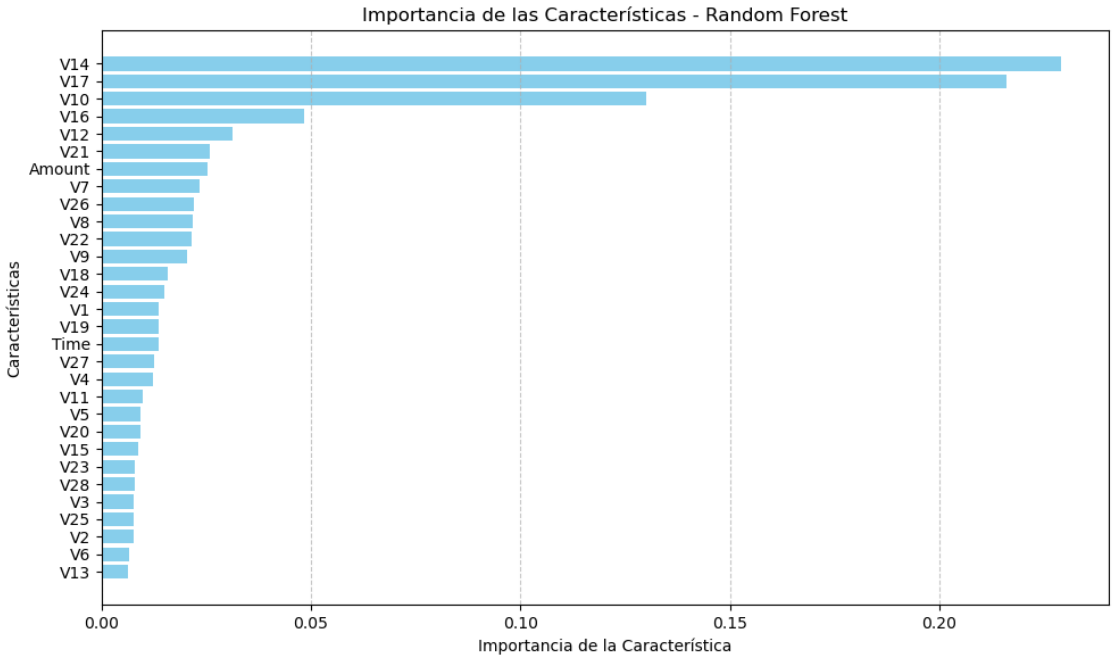
[14]:
```
report = classification_report(y_test, y_pred)
print("Reporte de clasificación:\n", report)
```

```
Reporte de clasificación:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56656
           1       0.94      0.71      0.81        90

    accuracy                           1.00     56746
   macro avg       0.97      0.86      0.90     56746
weighted avg       1.00      1.00      1.00     56746
```

1.1.5 Importancia de las características

Este gráfico añade una capa de interpretabilidad al proyecto, ayudando a entender qué aspectos del dataset están impulsando las decisiones del modelo, ya que las características con mayor importancia están más correlacionadas con la capacidad predictiva del modelo.

[15]:
```
importances = model.feature_importances_
features = X.columns

sorted_idx = importances.argsort()
sorted_features = features[sorted_idx]
sorted_importances = importances[sorted_idx]

plt.figure(figsize=(10, 6))
plt.barh(sorted_features, sorted_importances, color='skyblue')
plt.xlabel('Importancia de la Característica')
plt.ylabel('Características')
plt.title('Importancia de las Características - Random Forest')
plt.grid(axis='x', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

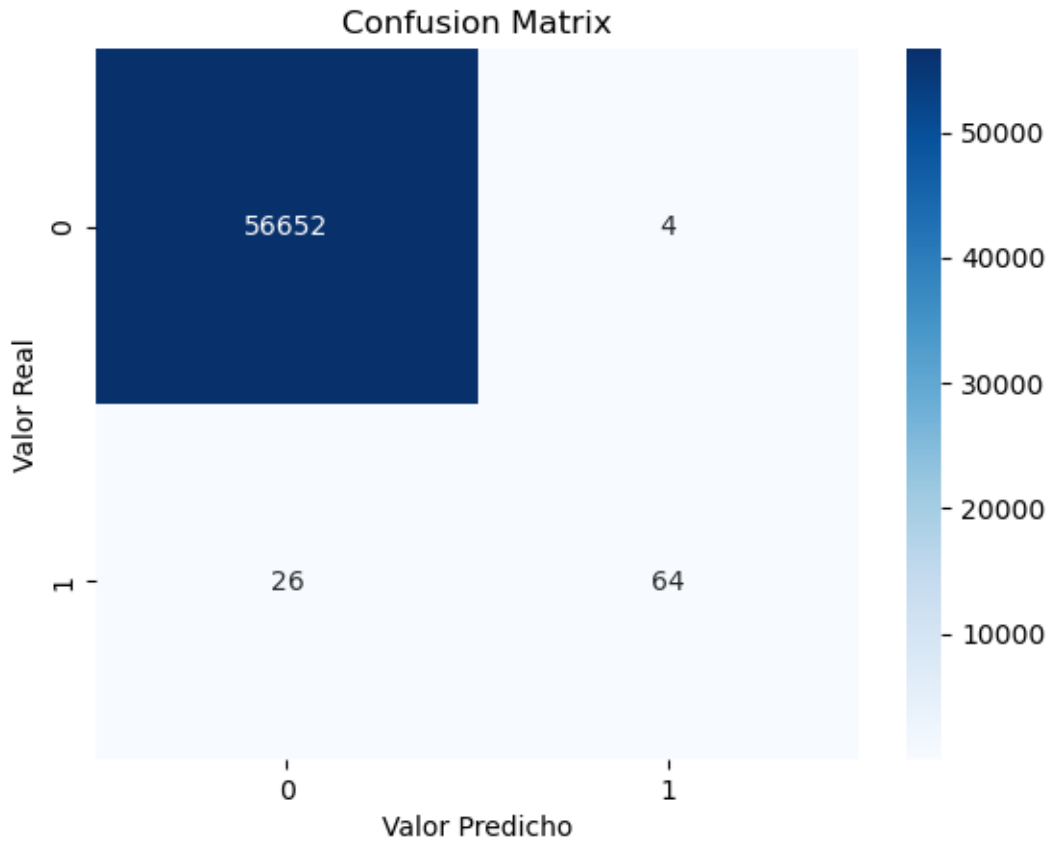


1.1.6 Matriz de confusión

La matriz de confusión es una herramienta visual excelente para visualizar el rendimiento del modelo, especialmente en problemas de clasificación.

[16]:
```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm, annot=True,fmt='d', cmap='Blues')
```

```
plt.xlabel('Valor Predicho')
plt.ylabel('Valor Real')
plt.title('Confusion Matrix')

plt.show()

FP = cm[0][1]  # Falsos Positivos
FN = cm[1][0]  # Falsos Negativos
print(f"Error Tipo I (Falsos Positivos): {FP}")
print(f"Error Tipo II (Falsos Negativos): {FN}")
```

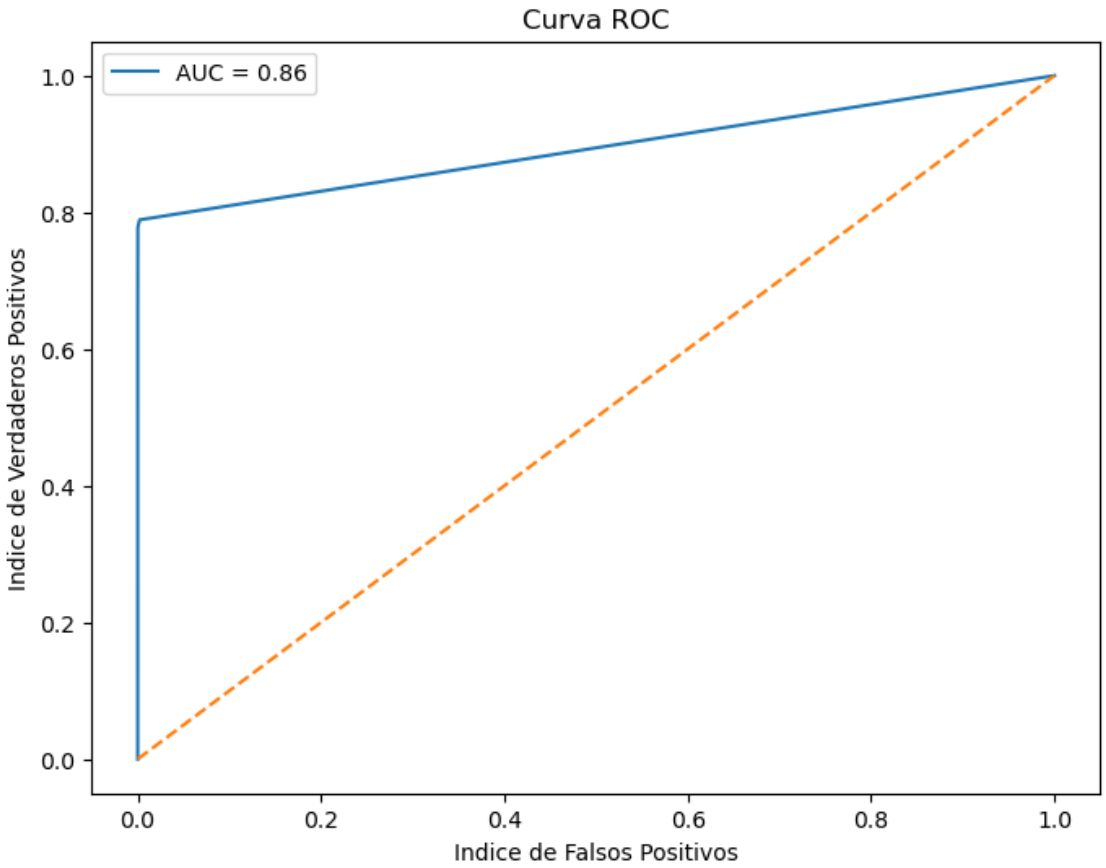



```
Error Tipo I (Falsos Positivos): 4
Error Tipo II (Falsos Negativos): 26
```

1.1.7 Curva ROC y AUC

La curva ROC muestra el rendimiento del modelo para todos los umbrales posibles, y el AUC es el área bajo la curva, que mide la capacidad del modelo para distinguir entre clases positivas y negativas. Un valor de AUC cercano a 1 indica un buen rendimiento.

[17]:
```python
from sklearn.metrics import roc_curve, roc_auc_score

fpr, tpr, thresholds = roc_curve(y_test, model.predict_proba(X_test)[:, 1])
auc = roc_auc_score(y_test, y_pred)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'AUC = {auc:.2f}')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('Indice de Falsos Positivos')
plt.ylabel('Indice de Verdaderos Positivos')
plt.title('Curva ROC')
plt.legend()

plt.show()
```

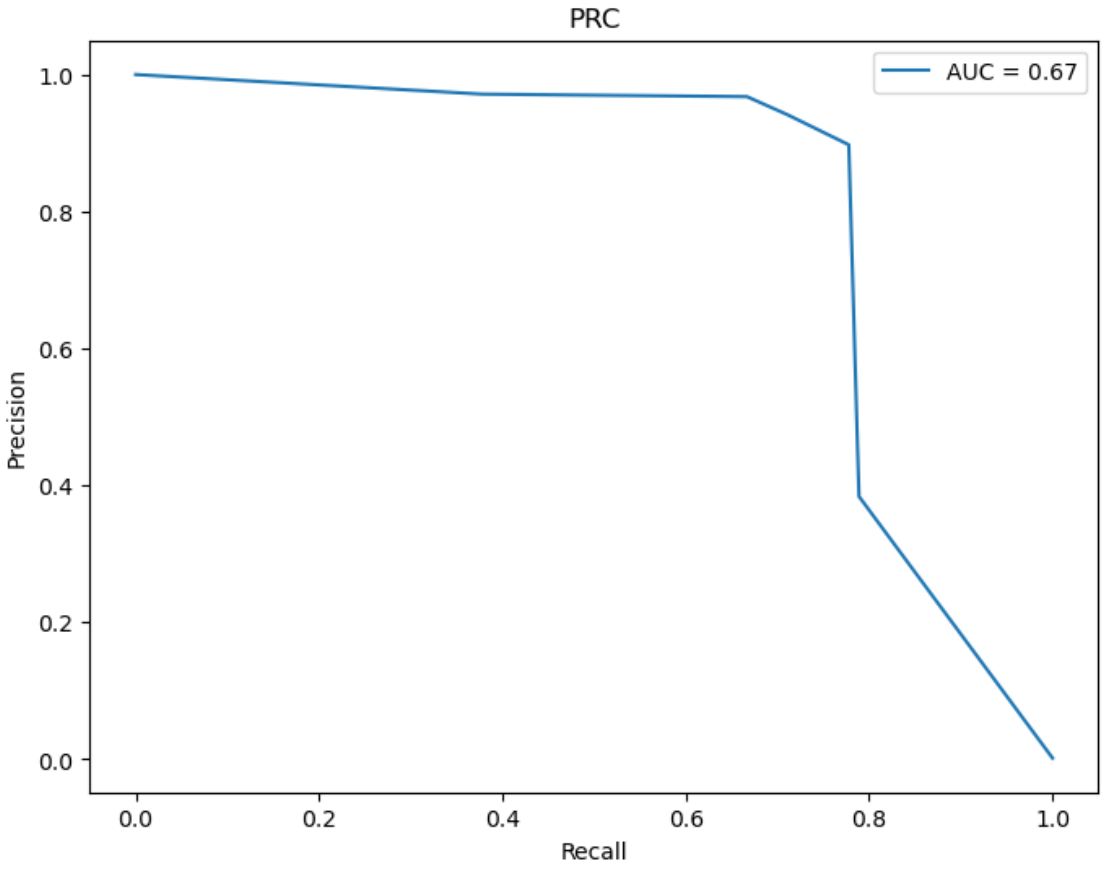


1.1.8 Curva Precision-Recall (PRC) y AUC

Similar a la curva ROC, la curva de precisión-recall muestra la relación entre la precisión y el recall para diferentes umbrales de clasificación. Es especialmente útil en problemas desbalanceados, ya que se enfoca más en la clase minoritaria (en este caso, las transacciones fraudulentas).

Cómo se interpreta: Un área bajo la curva PRC mayor indica un mejor rendimiento del modelo en cuanto a la detección de la clase positiva (fraudulenta).

[18]:
```
from sklearn.metrics import precision_recall_curve, average_precision_score

precision, recall, _ = precision_recall_curve(y_test, model.
  ↪predict_proba(X_test)[:, 1])
pr_auc = average_precision_score(y_test, y_pred)

plt.figure(figsize=(8, 6))
plt.plot(recall, precision, label=f'AUC = {pr_auc:.2f}')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('PRC')
plt.legend()

plt.show()
```



1.2 Visualizaciones interactivas

Utilizamos los módulos Plotly y Cufflinks para mejorar la interactividad y visualización del proyecto, haciéndolo más atractivo y fácil de analizar.

1.2.1 Configuración inicial

[19]:
```
import plotly.express as px
import cufflinks as cf

cf.go_offline()
cf.set_config_file(offline=False, world_readable=True)
```

1.2.2 Curvas de Aprendizaje

Una gráfica de curvas de aprendizaje interactiva permite inspeccionar cada punto y observar los valores de precisión en diferentes tamaños de entrenamiento.

[20]:
```
from sklearn.model_selection import learning_curve

train_sizes, train_scores, test_scores = learning_curve(model, X, y, cv=5,␣
 ↪n_jobs=-1)
train_scores_mean = train_scores.mean(axis=1)
test_scores_mean = test_scores.mean(axis=1)

fig = px.line(x=train_sizes, y=[train_scores_mean, test_scores_mean],␣
 ↪labels={'x': 'Training Size', 'y': 'Score'},
              title='Learning Curve', markers=True)

fig.data[0].name = 'Training Score'
fig.data[1].name = 'Validation Score'
fig.update_layout(legend_title_text='Score Type')
fig.show()
```

1.2.3 Distribución de Importe por Tipo de Transacción

Utilizamos una gráfica de tipo Box Plot para identificar la distribución de los importes en transacciones fraudulentas y no fraudulentas.

[21]:
```
fig = px.box(df_clean, x='Class', y='Amount', points="all", labels={'Class':␣
 ↪'Transaction Type', 'Amount': 'Transaction Amount'},
             title='Distribución de Importe por Tipo de Transacción')
fig.update_layout(xaxis=dict(tickvals=[0, 1], ticktext=['No Fraud', 'Fraud']))
fig.show()
```

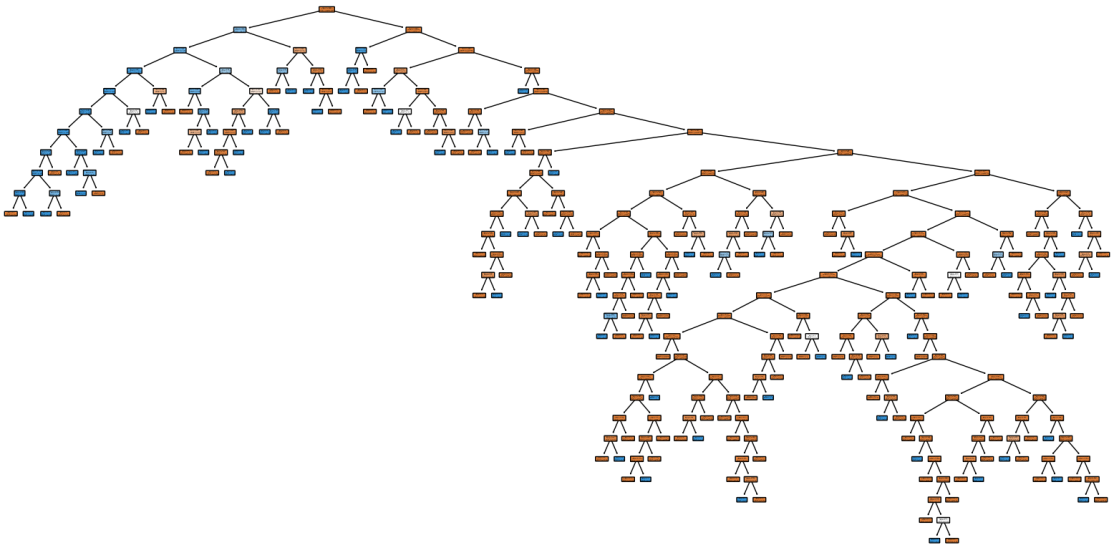
Ahora nos valemos de un gráfico KDE para comparar cómo se distribuye el importe entre transacciones fraudulentas y no fraudulentas. Este tipo de gráfico ayuda a visualizar la densidad de probabilidad de los datos.

[22]:
```
fig = px.density_contour(df_clean, x="Amount", color="Class",
                         labels={'Amount': 'Amount (in USD)', 'Class': 'Fraud'},
                         title="Distribución del Importe (Amount) por Clase de Fraude",
                         marginal_x="histogram",
                         category_orders={"Class": [0, 1]},
                         )
fig.show()
```

1.2.4 Visualización de los árboles de decisión

Por último, obtenemos una visualización de los árboles de decisión que ha implementado el modelo

[23]:
```
from sklearn.tree import plot_tree

# Seleccionar un árbol del modelo RandomForest
tree = model.estimators_[0]  # Extrae el primer árbol del Random Forest

plt.figure(figsize=(20, 10))
plot_tree(tree, filled=True, feature_names=X.columns, class_names=['No Fraude', 'Fraude'], rounded=True)
plt.show()
```



Reducimos el parámetro max_depth para reducir la complejidad del árbol y mejorar la visualización.

[24]:
```
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
```

```
# Crear el modelo de árbol de decisión con un max_depth ajustado
tree_model = DecisionTreeClassifier(max_depth=5, random_state=42)  # Ajusta␣
 ↪max_depth según sea necesario
tree_model.fit(X_train, y_train)

plt.figure(figsize=(20, 10))
plot_tree(tree_model, filled=True, feature_names=X.columns, class_names=['No␣
 ↪Fraude', 'Fraude'], rounded=True)
plt.show()
```