Consulting Challenge ‘24

Sebastian Marquez, Sebastián Higueras, Miguel Ángel Alcón, Guillermo García - Máster en Business Analytics

RETO

pwc

Tabla de Contenido

Introducción

La detección de fraude en transacciones con tarjetas de crédito es un desafío constante para las instituciones financieras. El uso de técnicas de machine learning puede ser fundamental para identificar patrones y anomalías en los datos que podrían indicar actividades fraudulentas.

Nuestro grupo, formado por Sebastian Marquez, Sebastián Higueras, Miguel Angel Galán y Guillermo García, decidimos afrontar este reto con optimismo, ganas de aprender y con el objetivo de elaborar el mejor modelo posible para este caso de uso.

Objetivo

Este informe describe los pasos esenciales, que realizaremos sobre el dataset (descrita en el punto A) para predecir el fraude en transacciones con tarjetas de crédito utilizando técnicas de machine learning con diferentes herramientas, las mismas que se justifican por la premura en la entrega de presente informe, entre otras:

1. RapidMiner
2. Python
3. Cloud
4. Firebase

Procedimiento

A. Adquisición de Datos

Nos entregan un dataset con transacciones de tarjetas de crédito, no hay fechas en las transacciones, incluyen características como cantidad, tipo de transacción, entre otros.

Descripción del dataset original del reto PwC

- Nombre: full_dataset.csv
- Tipo: CSV
- Registros: 6.362.620 registros
- Atributos: 11 atributos.

Limpieza y preprocesamiento: No hay datos duplicados ni valores faltantes. Transformar

Name	Type	Missing
amount	Real	0
oldbalanceOrg	Real	0
newbalanceOrig	Real	0
oldbalanceDest	Real	0
newbalanceDest	Real	0
step	Integer	0
isFlaggedFraud	Integer	0
Label isFraud	Nominal	0
type	Nominal	0
nameOrig	Nominal	0
nameDest	Nominal	0

variables si es necesario (normalización, codificación de variables categóricas).

B. Análisis Exploratorio de Datos (EDA)

En este paso se busca entender correctamente todas las características presentes en la base de datos para comenzar a aproximarse a posibles soluciones y poder proceder al uso de la misma de cara a la generación de modelos de clasificación que permitan la detección de fraude en las transacciones.

Análisis de los datos

De primera instancia, a través de una exploración con Rapidminer, se pudo detectar ciertas características importantes para decidir cómo proceder en la resolución del caso de uso como las siguientes:

- Millones de transacciones
- 5 tipos distintos de transacciones: CASH_IN (21,99%), PAYMENT (33,81%), TRANSFER (8,38%%), CASH_OUT (35,17%), DEBIT (0,65%).
    - Podemos suponer en base a los datos que se puede tratar de movimientos realizados con tarjetas de crédito.
- **8.213 transacciones** registradas como Fraude de 6.362.620 registros.

```
0    99.870918
1     0.129082
Name: isFraud, dtype: float64
```

- A penas un 0,13% de las observaciones están clasificadas como fraude
- Mayor volumen de transacciones realizadas a través de Transfer y Cash out en relación a fraude.
- 5 Variables que entregan valor para identificar un Fraude.
- Cash out (4116) y Transfer (4097) presentan mayor cantidad de casos de Fraude. (TABLA 1)
- Es más común que sea fraude cuando la transacción sea mayor (variable amount)
- Transacciones con valores en “0” identificadas como fraude
- Gran desviación de los datos. Podría ser conveniente la eliminación de los datos outliers.

- Ni una sola operación de fraude en Payment (transacciones físicas, poca probabilidad de fraude en estas transacciones).
- ha de ser eliminada ya que no nos aportará nada en la predicción del fraude, esto se debe a que su representación es casi nula.

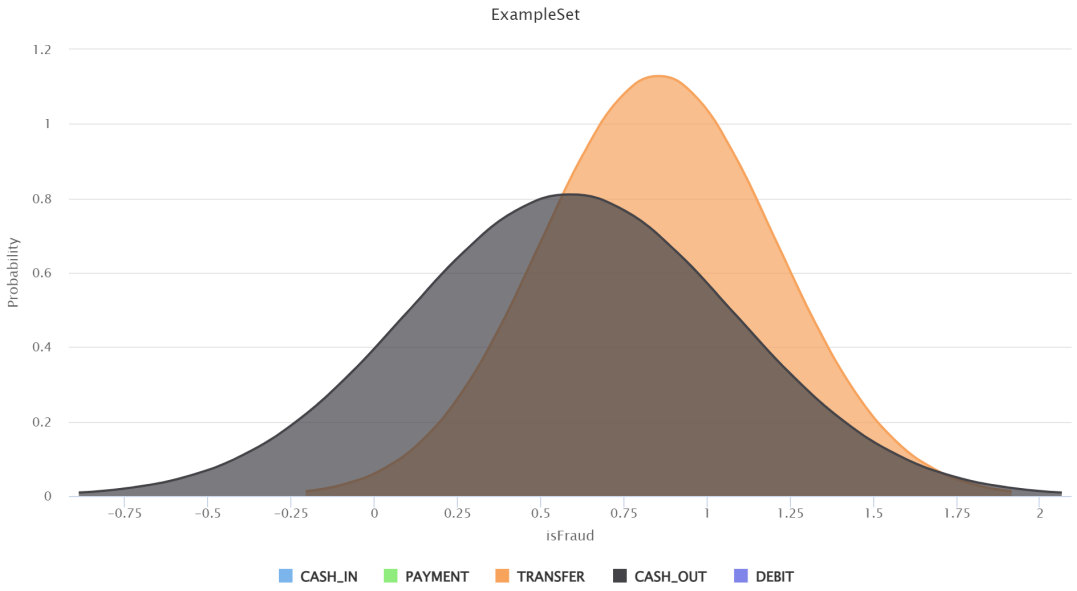
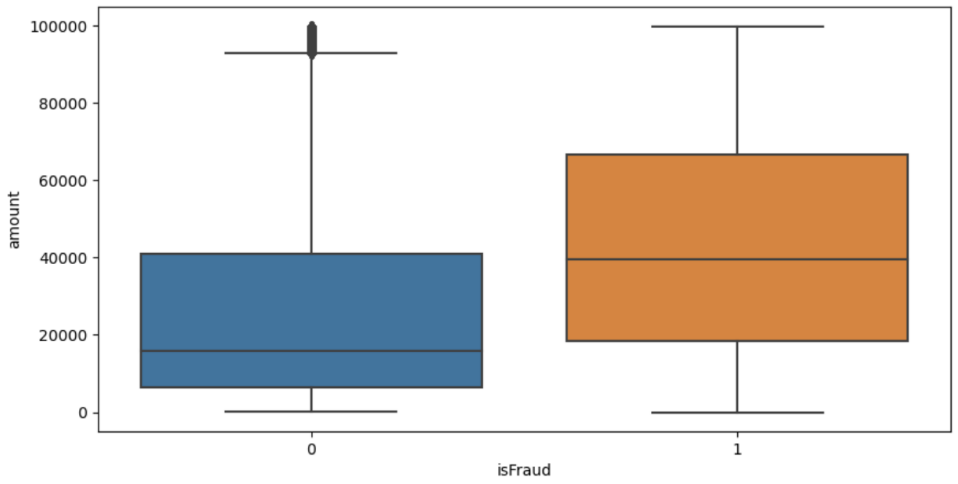
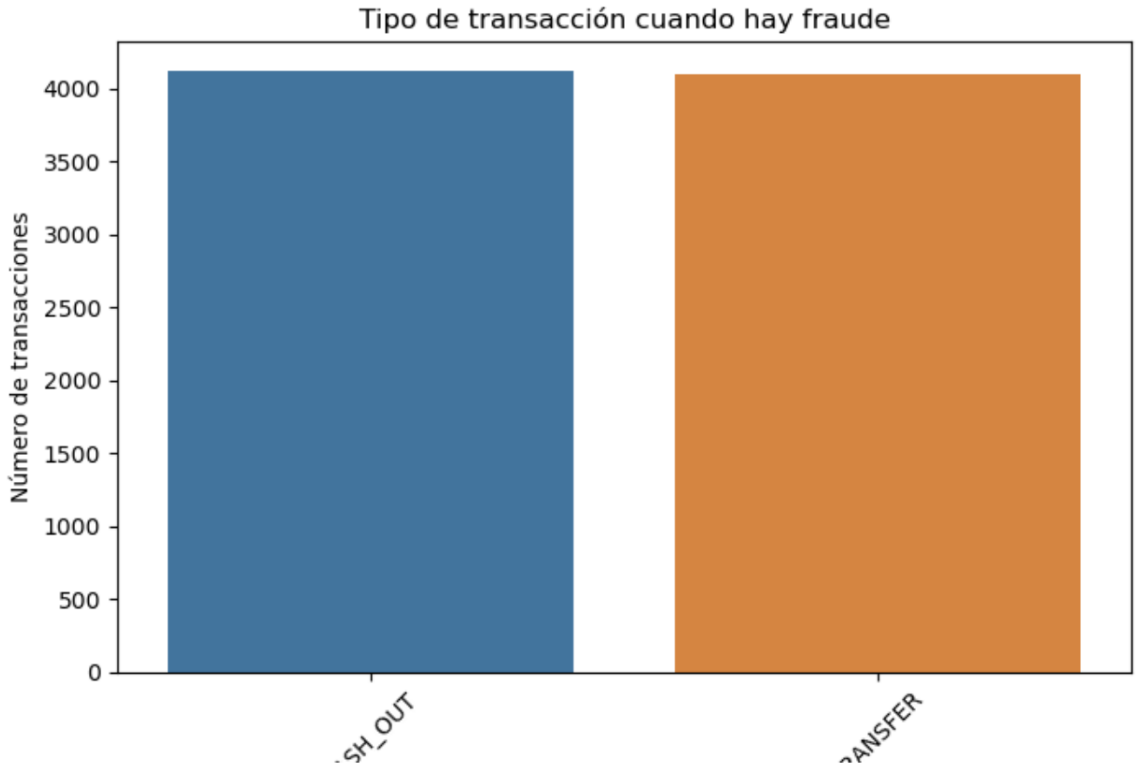


TABLA 1



Se observa que ocurren más fraudes cuando la variable amount es mayor.

TABLA 2



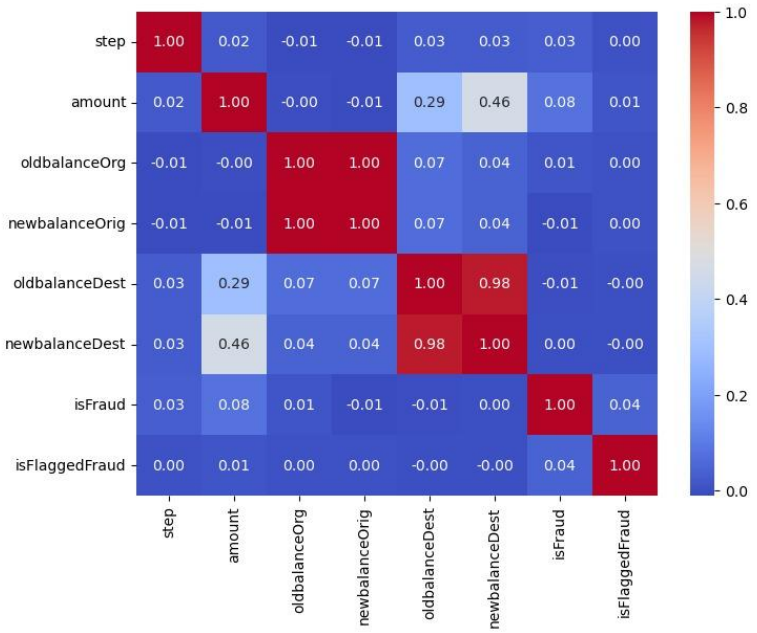
Distribución de variables

La desigualdad en la distribución de clases en las variables "No fraude"(6.3M) y "Fraude"(8213), puede ser problemática para el entrenamiento de modelos por las siguientes razones:

- **Sesgo del modelo:** Si el modelo se entrena en un conjunto de datos desequilibrado, podría desarrollar sesgos hacia la clase mayoritaria, es decir, a los “No Fraude”. Esto significa que el modelo podría tener dificultades para generalizar y detectar patrones en la clase minoritaria “Fraude”.
- **Baja sensibilidad al fraude:** Dado que la clase de fraude es significativamente más pequeña, el modelo podría tener dificultades para aprender patrones específicos de esta clase, lo que resulta en una baja sensibilidad o capacidad para identificar fraudes.

Correlaciones

Para determinar las correlaciones entre variables, decidimos optar por realizar una matriz de correlación a través de Python:



Como se puede observar en el gráfico, existe una correlación total entre las variables **oldbalanceorg** y **newbalanceorg**, así como **oldbalanceDest** y **newbalanceDest.** Existe gran influencia entre estas variables. Si una persona envía dinero a un destinatario, ve su capital reducido, mientras que el de la otra persona aumenta. Esto da lugar a redundancia entre las variables. Optamos por no eliminar ninguna de las variables debido a la falta de información acerca de la base de datos y su origen, así como de la relevancia que tiene para ellos estas variables.

C. Ingeniería de Características

Selección de características

La variable de mayor peso y la más importante para la correcta consecución de los modelos es “isFraud”, siendo la variable dependiente, ya que es la variable objetivo de la que obtendremos los resultados esperados. Sin embargo, y en adición a esta variable, se procederá a ordenar de mayor a menor importancia el resto de variables independientes, siendo aquellas que ofrecen algún tipo de valor a la clasificación.

```
# Suponiendo que ya has entrenado tu modelo de Random Forest y lo has asignado a model_rf
feature_importance = model_dt.feature_importances_

# Ahora puedes imprimir o procesar la importancia de las características
print(feature_importance)

#["amount","oldbalanceOrg","newbalanceOrig","oldbalanceDest","newbalanceDest", 'CASH_OUT','TRANSFER']
```

```
[0.15177777 0.34607048 0.36785836 0.0075471  0.00680441 0.05303464
 0.06690725]
```

Variables	Importancia
OldbalanceOrg (balance inicial previo a la transacción)	0.37
Amount (cantidad de la transacción))	0.1687
NewbalanceDest (balance final del receptor después de la transacción)	0.0029
NewbalanceOrig(balance final después de la transacción)	0.42
OldbalanceDest (balance inicial del receptor antes de la transacción)	0.0026

Podemos apreciar como la variable OldbalanceOrg, que hace referencia al balance inicial previo a la transacción, es con mucha diferencia la variable con un mayor nivel de importancia para el modelo.

Creación de nuevas características

Eliminamos las variables categóricas, además creamos variables dummy de la variable type para evaluar de mejor manera los modelos.

Eliminación de variables categóricas: Type, nameorigin y namedest

- Muchos algoritmos de aprendizaje automático no pueden manejar directamente variables categóricas. Es por eso, que es necesario cambiarlas a tipo numérico para poder utilizar dichas variables.

Creación de variables dummy:

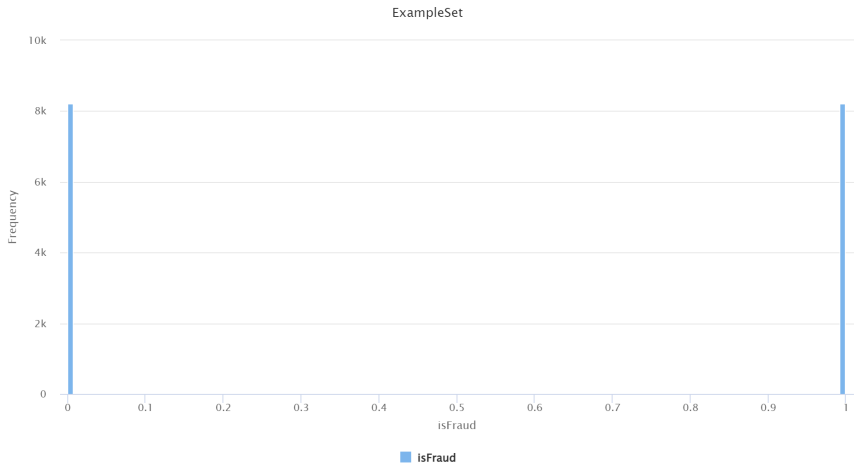
- Las variables dummy, las utilizamos para representar información categórica como variables binarias (0 o 1). Así podemos utilizar estas variables de una manera más efectiva. Dividimos la variable Type en CASH IN, CASH OUT, TRANSFER, DEBIT y PAYMENT. Para el modelo solo utilizamos Cash out y Transfer .

Razones para realizar estos pasos en modelos predictivos de fraude:

- Mejora de la precisión del modelo: convirtiendo las variables categóricas en numéricas,, el algoritmo puede identificar patrones más fácilmente y mejorar la precisión del modelo.
- Evitar sesgos: Al trabajar con variables dummy en lugar de simplemente asignar números a las categorías, se evitan sesgos que podrían surgir al asignar un orden numérico arbitrario a las categorías.
- Compatibilidad con algoritmos: Muchos algoritmos de aprendizaje automático, como regresión logística o máquinas de soporte vectorial, requieren que las variables de entrada sean numéricas.

División de datos:

Se apreció un desequilibrio significativo en la proporción de los datos. Con tan solo 8213 casos de fraude, en contraposición a los otros 6 millones de casos de no fraude. Debido a ello el modelo terminaría identificando todos los casos como casos de no fraude, haciendo del todo inservible el modelo. A su vez, trabajar con un conjunto de datos tan inmenso podría derivar en problemas de eficiencia desde un punto de vista computacional. Es por ello que se decidió realizar un **undersampling** del conjunto de datos. Reduciendo así tanto los tiempos de entrenamiento del modelo, como los recursos requeridos para ello, así como disponer de un conjunto de datos armonizado y balanceado con el que poder entrenar al modelo con las máximas garantías.



Aplicando Undersampling

```
from collections import Counter
from imblearn.under_sampling import RandomUnderSampler
from sklearn.model_selection import train_test_split
```

```
X = X_orig.copy()
y = y_orig.copy()

under_sample = RandomUnderSampler(sampling_strategy='majority')

X_under, y_under = under_sample.fit_resample(X, y)
# summarize class distribution
print(Counter(y_under))
```

```
y_under.value_counts()
```

```
# Divide los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X_under, y_under, train_size=0.7, test_size=0.3, random_state=10

# Entrena un modelo de árbol de decisión
model_dt = DecisionTreeClassifier()
model_dt.fit(X_train, y_train)
```

```
model_dt.score(X_test, y_test)
```

```
print("score en data de entrenamiento: {:.4f}".format(model_dt.score(X_train, y_train)))
print("score en data de validacion: {:.4f}".format(model_dt.score(X_test, y_test)))
```

```
# Realiza predicciones en el conjunto de prueba
predicciones = model_dt.predict(X_test)
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# Calcula y muestra la matriz de confusión
matriz_confusion = confusion_matrix(y_test, predicciones)
plt.figure(figsize=(8, 6))
sns.heatmap(matriz_confusion, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
# Making predictions
y_pred_dt = model_dt.predict(X_test)

# Printing classification report
print(classification_report(y_test, predicciones))
```

```
y_prob = model_dt.predict_proba(X_test)[:, 1]  # Selecciona las probabilidades de la clase positiva

# Calcula el AUC-ROC Score
auc_roc = roc_auc_score(y_test, y_prob)

print("AUC-ROC Score:", auc_roc)
```

```
#kfold_validation=KFold(10)

#results=cross_val_score(model_dt,X,y,cv=kfold_validation)
#print(results)
#print(np.mean(results))
```

```
#df_nom.to_excel('data encriptada.xlsx')
#df_nom2.to_excel('data clusterizada.xlsx')
```

El conjunto de datos se vió dividido en dos partes bien diferenciadas, una primera parte con un 70% de los datos, y otra parte con un 30% de los mismos, distribuidos de forma heterogénea:

- Ese 30% de los datos permite evaluar el desempeño del modelo con datos desconocidos. Fundamental para cerciorarse de si el modelo ha aprendido patrones generales o de si ha memorizado los datos de entrenamiento.
- Si la accuracy en la muestra de entrenamiento es mucho mayor que en la muestra de prueba. Se puede inferir que el modelo está memorizando en lugar de generalizando.

D. Selección y Entrenamiento del Modelo

A la hora de seleccionar el modelo, era óptimo realizar pruebas con diferentes modelos para poder analizar la confianza de cada uno de ellos y así elegir el que nos otorgara una mayor precisión. En nuestro caso, elegimos 3 modelos que consideramos potencialmente beneficiosos para este caso: Decision Tree (sin balancear), Decision Tree (Random Undersampling), Random Forest .

Árboles de decisión:

Este modelo toma decisiones mediante la construcción de una estructura de árbol. Cada nodo interno del árbol representa una decisión basada en un atributo (ej: fraude), y cada hoja representa la clase o el valor de salida. Los árboles de decisión son fáciles de entender e interpretar, lo que los hace útiles para la toma de decisiones explicables. Sin embargo, pueden ser propensos al sobreajuste (overfitting) si no se controla adecuadamente. En primer lugar probamos el modelo sin balancear para base de datos a modo de prueba aunque la precisión era menor debido a la dispersión de los datos.

Random Forest:

Random Forest es una extensión de los árboles de decisión que construye múltiples árboles y los combina para obtener un modelo más robusto y preciso. Cada árbol se entrena en una muestra aleatoria del conjunto de datos y luego vota sobre la clasificación final. Al combinar múltiples árboles, el Random Forest tiende a ser más resistente al sobreajuste y proporciona una mejor generalización. Es especialmente útil cuando se trabaja con conjuntos de datos grandes y complejos como es nuestro caso, a pesar de que nosotros hemos optado por un undersampling debido a la falta de capacidad de computación.

Motivo de elección de los modelos:

- **Manejo de características desbalanceadas**: Los conjuntos de datos de fraudes suelen tener una proporción desigual entre las clases normales y fraudulentas. Los modelos de árboles y Random Forest pueden manejar desequilibrios en los datos de manera efectiva.
- **Capacidad para manejar características no lineales**: Los fraudes pueden manifestarse de maneras no lineales y complejas. Los árboles de decisión y Random Forest pueden capturar patrones no lineales en los datos, lo que los hace adecuados para la detección de fraudes.
- **Interpretabilidad:** En muchos casos, es crucial comprender cómo se toma una decisión. Los árboles de decisión son interpretables, lo que significa que se puede seguir la lógica de las decisiones tomadas en cada nodo. Esto puede ser valioso para entender y explicar por qué se clasifica una transacción como fraudulenta.
- **Manejo de grandes volúmenes de datos**: Random Forest es especialmente efectivo en grandes conjuntos de datos, lo que puede ser relevante en la detección de fraudes, donde se pueden tener grandes cantidades de transacciones para analizar
- **Entrenamiento del modelo:** Utilizar el conjunto de entrenamiento para entrenar el modelo seleccionado.

E. Evaluación del Modelo

Metodologías utilizadas para la validación del modelo:

```
# Printing classification report
print(classification_report(y_test, predicciones))
```

```
              precision    recall  f1-score   support

           0       0.99      0.93      0.96      2438
           1       0.93      0.99      0.96      2490

    accuracy                           0.96      4928
   macro avg       0.96      0.96      0.96      4928
weighted avg       0.96      0.96      0.96      4928
```

```
# Calcula el AUC-ROC Score
auc_roc = roc_auc_score(y_test, predicciones)

print("AUC-ROC Score:", auc_roc)
```

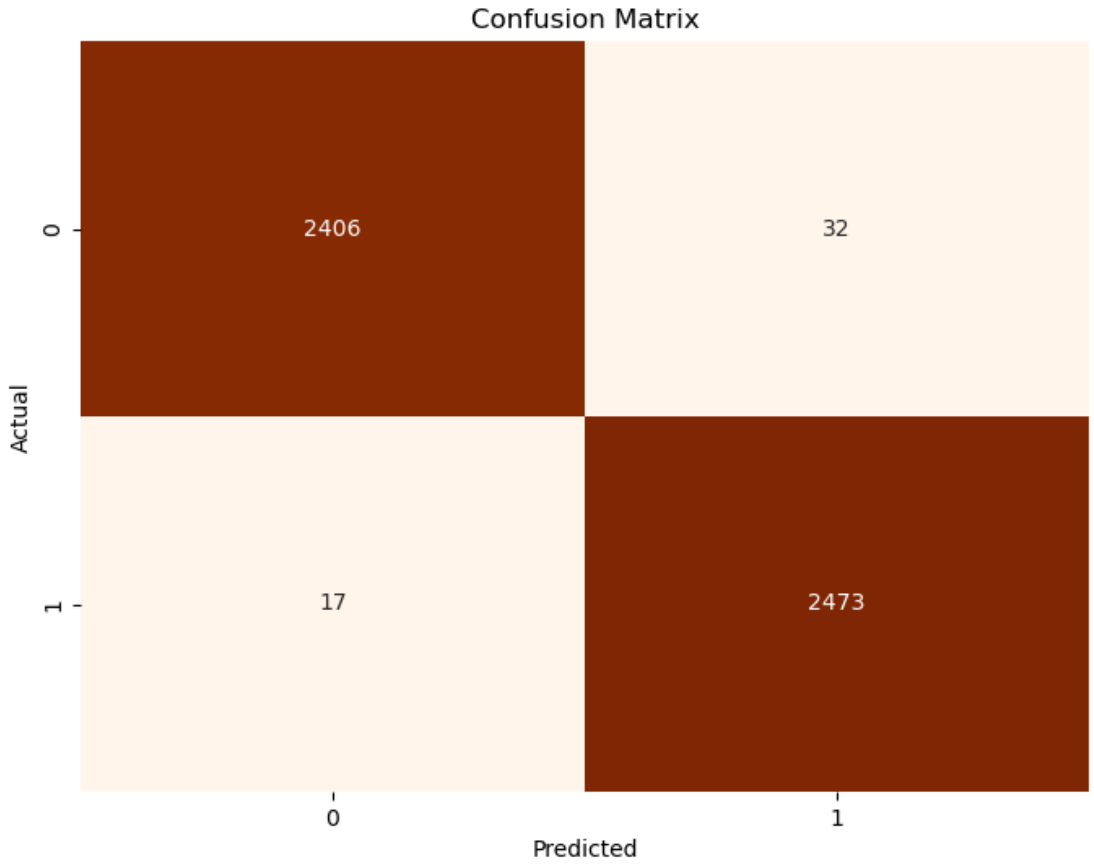
```
AUC-ROC Score: 0.9617088534614255
```

Precisión: es la proporción de instancias positivas que fueron correctamente clasificadas con respecto a todas las instancias clasificadas como positivas (verdaderos positivos divididos por la suma de verdaderos positivos y falsos positivos). **Mide la exactitud de las predicciones positivas del modelo.** Es útil cuando se busca minimizar los falsos positivos.

Exhaustividad (Recall o Sensibilidad): Exhaustividad es la proporción de instancias positivas que fueron correctamente clasificadas con respecto a todas las instancias que realmente son positivas (verdaderos positivos divididos por la suma de verdaderos positivos y falsos negativos). **Mide la capacidad del modelo para identificar todas las instancias positivas**. Es importante cuando se busca minimizar los falsos negativos.

F1-Score: El F1-Score es la media armónica de precisión y exhaustividad. Ofrece un equilibrio entre precisión y exhaustividad. Es útil cuando hay un desequilibrio entre las clases.

Matriz de Confusión: La matriz de confusión es una tabla que muestra las predicciones correctas e incorrectas del modelo, clasificadas en verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos. Proporciona una visión detallada del rendimiento del modelo para cada clase. Esencial para entender cómo el modelo clasifica cada tipo de instancia.



ROC-AUC (Área bajo la Curva ROC): La curva ROC representa la tasa de verdaderos positivos frente a la tasa de falsos positivos en varios umbrales de clasificación. El AUC (Área bajo la Curva) mide la integral debajo de la curva ROC. Evalúa la capacidad del modelo para discriminar entre clases. Un AUC cercano a 1 indica un buen rendimiento, mientras que un AUC cercano a 0.5 indica un rendimiento similar al azar.

F. Despliegue y Monitoreo

Implementación del modelo: Integrar el modelo en un entorno de producción para monitorear transacciones en tiempo real, ya que en la actualidad solo podrá predecir entorno simulados o contratos, no en tiempo real.

Monitoreo continuo: Supervisar el rendimiento del modelo y realizar actualizaciones según sea necesario para mantener su efectividad.

Conclusiones & Consideraciones Finales

La predicción de fraude en tarjetas de crédito mediante técnicas de machine learning es fundamental para mitigar riesgos financieros. Los pasos mencionados constituyen un marco sólido para desarrollar un sistema efectivo de detección de fraudes. En nuestro caso, hemos determinado que el modelo Árbol de Decisiones ya que a pesar de no ser el más acertado, nos ofrece una precisión cercana al 97%. Random Forest sería óptimo ya que presenta aún mayor precisión pero necesita mayor capacidad de computación.

La actualización constante del modelo es crucial para adaptarse a nuevos patrones de fraude. La colaboración con expertos en seguridad financiera es esencial para mejorar la precisión y eficacia del modelo.

PRESENTACIÓN: https://www.canva.com/design/DAElv5BK27Y/Gc9ldgImkOPb_09jfL0W2w/edit
ENLACE GITHUB:

https://github.com/GuillermoGG0102/PwC-Grupo1/blob/main/PWC_Challenge.ipynb