

Documentación de Modelos de Detección de Fraude en AWS Sagemaker Canvas

- **Juan Sebastián Giraldo Sepúlveda**
- **Juan Sebastián Navas Gómez**
- **Daniel Alejandro Ruiz Carrillo**
- **Carlos Alberto Trujillo**

Este documento detalla el proceso completo de desarrollo de dos modelos de predicción de fraude, abarcando desde la extracción y preprocesamiento de datos, hasta la creación y evaluación de los modelos utilizando AWS SageMaker Canvas.

Modelo 1: Basado en preprocesamiento_original.ipynb

1. Extracción y Preprocesamiento de Datos con Python

El primer modelo se entrenó utilizando un dataset que fue preprocesado inicialmente con un script de Python (preprocesamiento_original.ipynb). Este script se encargó de cargar múltiples fuentes de datos, realizar un análisis exploratorio básico, generar nuevas características y unificar la información en un solo conjunto de datos.

1.1. Carga de Datos

Se cargaron los siguientes archivos CSV en DataFrames de Pandas:

- `cards_data.csv`: Contiene información sobre las tarjetas.
- `transactions_data.csv`: Registra las transacciones.
- `users_data.csv`: Almacena datos de los usuarios.

Además, se cargó y procesó el archivo `mcc_codes.json` para obtener descripciones de los códigos MCC (Merchant Category Code), las cuales fueron integradas en el DataFrame de transacciones.

1.2. Análisis Inicial de Datos (Visión General)

Durante la carga de datos, se realizó un chequeo inicial para asegurar que los DataFrames se cargaran correctamente y para obtener una primera impresión de su estructura.

Script de Carga de Datos y MCC Codes:

```
1. import pandas as pd
2. import json
3.
4. # --- 1. Carga de Datos ---
5. # Carga de archivos CSV
6. cards_data = pd.read_csv('cards_data.csv')
```

```

7. transactions_data = pd.read_csv('transactions_data.csv')
8. users_data = pd.read_csv('users_data.csv')
9.
10. # Carga y procesamiento de mcc_codes.json
11. with open("mcc_codes.json", "r", encoding="utf-8") as f:
12.     mcc_dict = json.load(f)
13.
14. rows_mcc = []
15. for code_str, desc in mcc_dict.items():
16.     rows_mcc.append({
17.         "mcc": int(code_str),
18.         "mcc_description": desc
19.     })
20. mcc_codes = pd.DataFrame(rows_mcc)
21.
22. print("Datasets cargados exitosamente.")
23.

```

1.3. Ingeniería de Características y Unión de DataFrames

En esta fase, se realizaron diversas operaciones para enriquecer los datos y prepararlos para el modelado:

- **Variables Temporales en transactions_data:** Se extrajeron el hour, weekday, is_weekend y month de la columna date. También se creó una columna is_night para indicar transacciones nocturnas.
- **Identificación de Fraude (is_fraud):** Se creó la columna is_fraud en transactions_data basándose en la columna errors. Si la columna errors no estaba vacía o contenía ciertos valores, se marcaba como fraude (1); de lo contrario, como no fraude (0).
- **Limpieza de Columnas Monetarias:** Las columnas amount (en transactions_data), per_capita_income, yearly_income, y total_debt (en users_data y cards_data) fueron convertidas a formato numérico, eliminando símbolos de dólar y comas.
- **Unión de DataFrames:** Se realizó la unión de los DataFrames transactions_data, users_data, cards_data y mcc_codes utilizando las columnas de identificación (client_id, card_id, mcc). Se usaron sufijos (_user, _card, _transaction) para diferenciar columnas con nombres duplicados.

Script de Ingeniería de Características y Unión:

```

1. # --- 2. Ingeniería de Características y Preprocesamiento ---
2. # transactions_data:
3. # Extraer características de tiempo
4. df_transactions['date'] = pd.to_datetime(df_transactions['date'])
5. df_transactions['hour'] = df_transactions['date'].dt.hour
6. df_transactions['weekday'] = df_transactions['date'].dt.weekday
7. df_transactions['is_weekend'] = ((df_transactions['date'].dt.weekday == 5) |
(df_transactions['date'].dt.weekday == 6)).astype(int)

```

```

8. df_transactions['month'] = df_transactions['date'].dt.month
9. df_transactions['is_night'] = ((df_transactions['hour'] >= 22) | (df_transactions['hour'] <=
6)).astype(int)
10.
11. # Crear la variable objetivo 'is_fraud'
12. df_transactions['is_fraud'] = df_transactions['errors'].apply(lambda x: 1 if pd.notna(x) and x
!= '' else 0)
13.
14. # Limpiar columna 'amount'
15. df_transactions['amount'] = df_transactions['amount'].replace({r'\$': '', r',': ''},
regex=True).astype(float)
16.
17. # cards_data:
18. # Limpiar columna 'credit_limit' y convertir a numérico
19. df_cards['credit_limit'] = df_cards['credit_limit'].replace({r'\$': '', r',': ''},
regex=True).astype(float)
20.
21. # users_data:
22. # Limpiar columnas monetarias y convertir a numérico
23. monetary_cols = ['per_capita_income', 'yearly_income', 'total_debt']
24. for col in monetary_cols:
25.     df_users[col] = df_users[col].replace({r'\$': '', r',': ''}, regex=True).astype(float)
26.
27. # Unir todos los datasets
28. merged_df = pd.merge(df_transactions, mcc_codes, on='mcc', how='left')
29. merged_df = pd.merge(merged_df, df_users, left_on='client_id', right_on='id', how='left',
suffixes=('_transaction', '_user'))
30. merged_df = pd.merge(merged_df, df_cards, left_on='card_id', right_on='id', how='left',
suffixes=('', '_card'))
31.
32. # Renombrar columnas para claridad
33. merged_df = merged_df.rename(columns={
34.     'id': 'id_card', # Renombrar id_card para evitar conflicto si se mantiene
35.     'id_transaction': 'id', # La ID de transacción es la ID principal
36.     'id_user': 'id_user' # Mantener el nombre si ya está definido
37. })
38.
39. print("Características creadas y datasets unidos exitosamente.")
40.

```

1.4. Selección de Columnas Finales y Guardado del Dataset

Se seleccionaron las columnas finales relevantes para el entrenamiento del modelo y se guardó el dataset procesado en un archivo CSV llamado processed_dataset.csv.

Script de Guardado del Dataset:

```

1. # --- 3. Selección de Columnas y Guardado ---
2. output_columns = [
3.     'id', 'date', 'client_id', 'card_id', 'amount', 'use_chip', 'merchant_id',
4.     'merchant_city', 'merchant_state', 'zip', 'mcc', 'errors', 'mcc_description',
5.     'is_fraud', 'id_user', 'current_age', 'retirement_age', 'birth_year',
6.     'birth_month', 'gender', 'address', 'latitude', 'longitude',
7.     'per_capita_income', 'yearly_income', 'total_debt', 'credit_score',
8.     'num_credit_cards', 'hour', 'weekday', 'is_weekend', 'month', 'is_night'
9. ]
10.
11. # Asegurarse de que todas las columnas existan antes de la selección
12. # Esto es una comprobación para evitar errores si alguna columna no se creó correctamente
13. existing_columns = [col for col in output_columns if col in merged_df.columns]

```

```

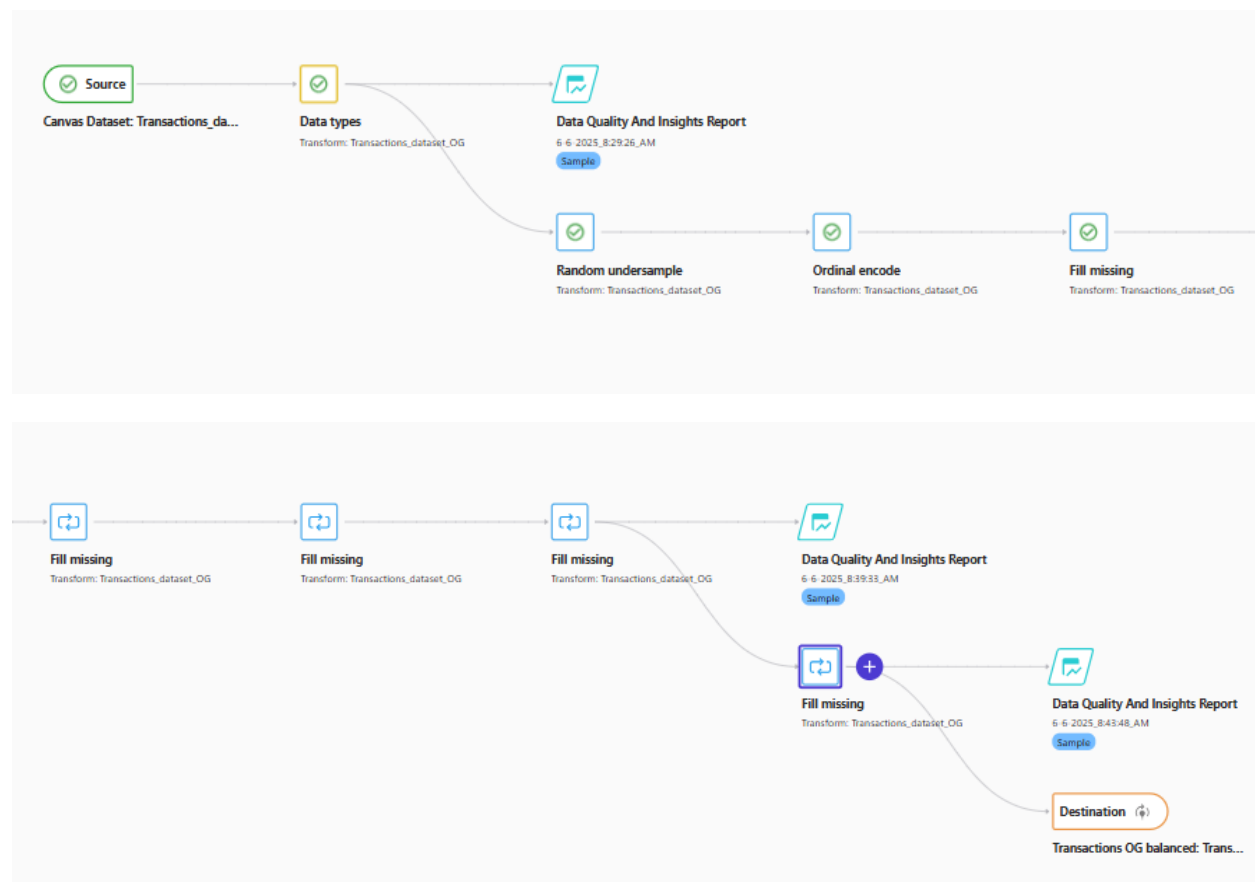
14. df_final = merged_df[existing_columns]
15.
16. df_final.to_csv('processed_dataset.csv', index=False)
17.
18. print("Dataset procesado y guardado exitosamente en 'processed_dataset.csv'")
19.

```

2. Procesamiento de Datos con AWS SageMaker Data Wrangler

El dataset processed_dataset.csv fue subido a AWS SageMaker Canvas y procesado adicionalmente en Data Wrangler. Este paso fue crucial para manejar variables categóricas, valores faltantes y el desbalance de clases.

2.1. Data Flow en Data Wrangler



Estas imágenes muestran el flujo de datos configurado en Data Wrangler para el preprocesamiento del dataset. Se incluyeron pasos como la importación de datos, la adición de transformaciones y la exportación del conjunto de datos transformado.

2.2. Análisis de Insights del Dataset

Imagen: “data-wrangler-insights-report_original.png”

Esta imagen muestra el reporte de insights generado por Data Wrangler, proporcionando una visión general de la distribución de las variables, la calidad de los datos y las relaciones entre ellas. Esto incluye detalles sobre la distribución de las variables, valores atípicos y posibles correlaciones, lo cual es útil para identificar áreas de mejora en el preprocesamiento.

2.3. Pasos de Procesamiento en Data Wrangler

Aunque las imágenes proporcionadas no detallan cada paso de forma granular, basándonos en la descripción inicial, las transformaciones clave realizadas en Data Wrangler incluyen:

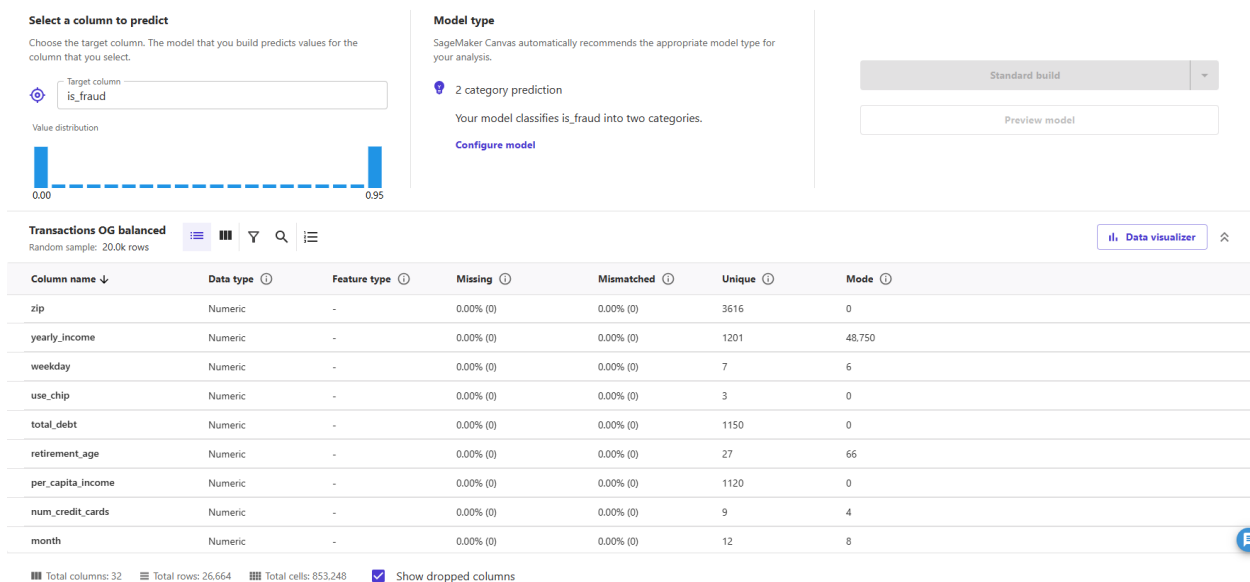
- **Codificación de Variables Categóricas:** Se aplicaron técnicas de codificación (One-Hot Encoding, Label Encoding, etc.) a las columnas categóricas (use_chip, merchant_city, merchant_state, gender, mcc_description, card_brand, card_type, has_chip, card_on_dark_web) para convertirlas a un formato numérico adecuado para el entrenamiento del modelo.
- **Manejo de Valores Faltantes:** Se implementaron estrategias para tratar los valores NaN en columnas como merchant_state, zip, y errors. Esto pudo incluir imputación con la media/mediana/moda, o la eliminación de filas/columnas con demasiados valores faltantes.
- **Balanceo de Clases:** Dada la naturaleza altamente desbalanceada del dataset (1.58% de transacciones fraudulentas), se aplicaron técnicas de balanceo de clases (por ejemplo, SMOTE, sobremuestreo, submuestreo) para asegurar que el modelo no se sesgara hacia la clase mayoritaria (no fraude) y tuviera un mejor rendimiento en la detección de fraudes.

3. Creación y Evaluación del Modelo en SageMaker Canvas

Una vez que el dataset fue preprocesado en Data Wrangler, se utilizó SageMaker Canvas para la creación y evaluación del modelo predictivo. El objetivo era construir un modelo de clasificación binaria (fraude o no fraude) optimizado para el F1-score.

3.1. Configuración del Modelo

- **Creación del Modelo en SageMaker:**



Esta imagen muestra la interfaz de SageMaker Canvas para la creación de un nuevo modelo, donde se selecciona el dataset procesado y se configura el tipo de problema.

- **Métrica Objetivo:**

Objective metric

Select the metric that you want Canvas to optimize while building your model. By default, Canvas optimizes for the best metric based on your model type.

F1 - default
▼


The F1 score is a measure of a model's accuracy in binary classification tasks. It averages two important metrics: precision and recall. F1 works well for imbalanced datasets.

La métrica de optimización seleccionada para el entrenamiento del modelo fue el F1-score. Esta métrica es especialmente útil en problemas de clasificación con clases desbalanceadas, ya que considera tanto la precisión como la exhaustividad (recall) de la clase minoritaria (fraude).

- **División de Datos:**

Data split

Specify how you want to split your data into training and validation sets. Canvas builds your model with the training set and verifies the model's accuracy with the validation set.

 Configuring the data split will default to Standard build.

Training set
80 %

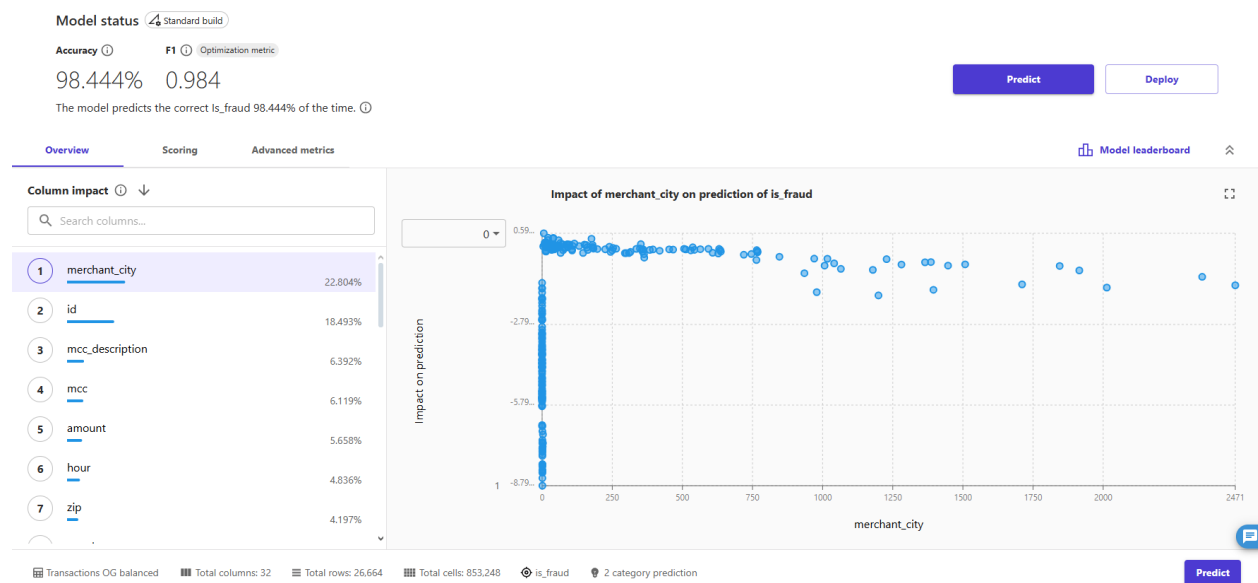
Validation set
20 %

La división de los datos en conjuntos de entrenamiento, validación y prueba es fundamental para evaluar el rendimiento generalizable del modelo. Esta imagen ilustra la configuración de la división del dataset en SageMaker Canvas.

3.2. Análisis del Rendimiento del Modelo

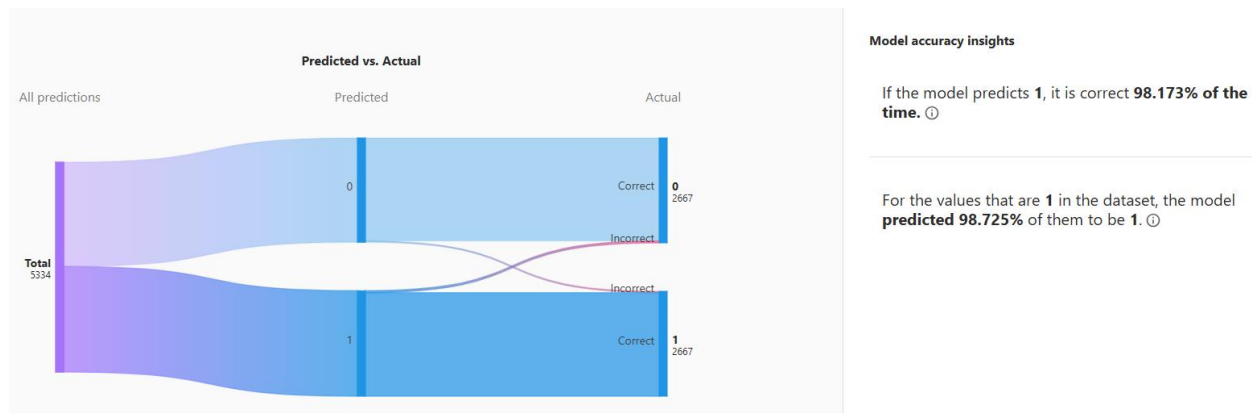
Una vez entrenado, se analizaron diversas métricas y visualizaciones para comprender el rendimiento del modelo.

- Análisis General del Modelo:**



Esta imagen proporciona una vista general del rendimiento del modelo, incluyendo métricas clave y gráficos que resumen su comportamiento.

- Scoring del Modelo:**



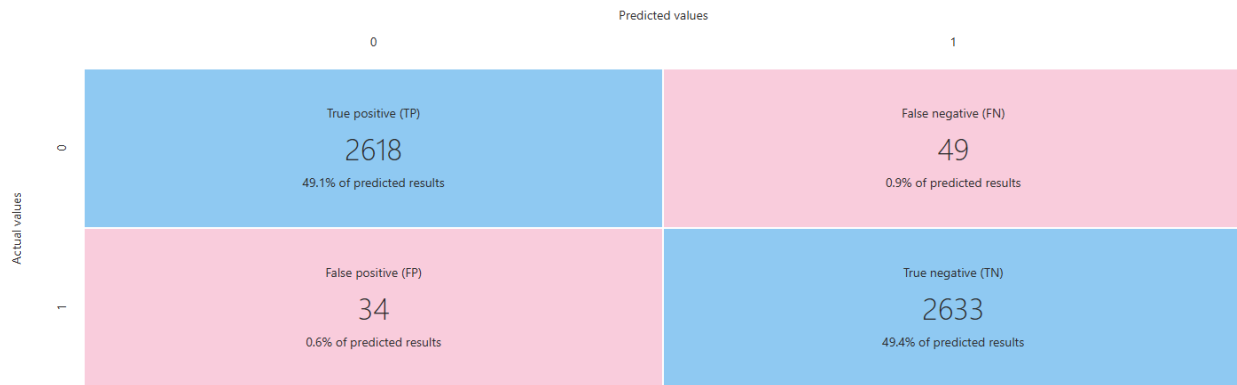
Esta imagen muestra los resultados del scoring del modelo, indicando cómo se desempeñó el modelo en la predicción de nuevas instancias.

- **Tabla de Métricas:**

Metric name	Value
precision	0.987
recall	0.982
accuracy	0.984
f1	0.984
auc	0.998

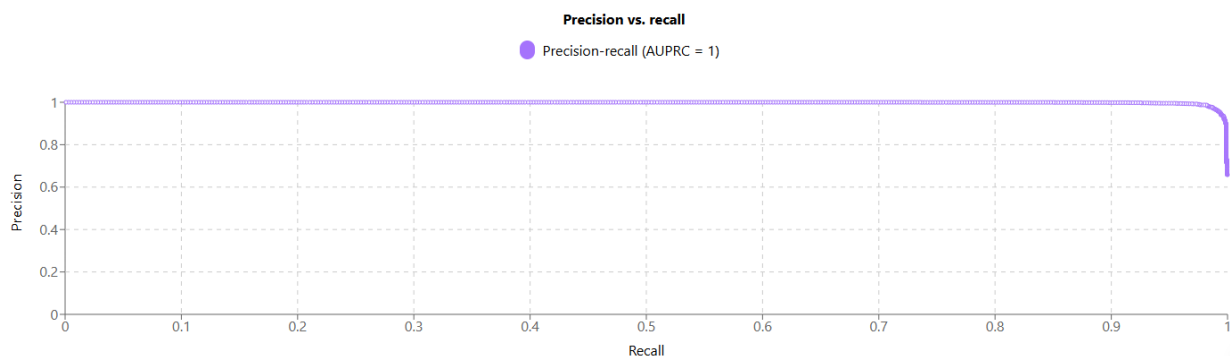
La tabla de métricas proporciona un resumen cuantitativo del rendimiento del modelo, incluyendo el F1-score, precisión, recall, exactitud, y otras métricas relevantes para la clasificación binaria.

- **Matriz de Confusión:**



La matriz de confusión es una herramienta visual que permite entender la cantidad de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos. Es crucial para evaluar la capacidad del modelo para identificar correctamente las transacciones fraudulentas y no fraudulentas.

- **Curva de Precisión vs. Recall:**



La curva de precisión-recall es particularmente importante para problemas de clasificación desbalanceados. Permite visualizar el equilibrio entre la precisión y la exhaustividad del modelo en diferentes umbrales de clasificación. Una curva que se mantiene cerca de la esquina superior derecha indica un mejor rendimiento.

Modelo 2: Basado en preprocesamiento_aumentado.ipynb

1. Extracción y Preprocesamiento de Datos con Python

El segundo modelo se entrenó utilizando un dataset que fue preprocesado inicialmente con un script de Python (preprocesamiento_aumentado.ipynb). Este script incorpora un análisis más detallado de los datos y la creación de características adicionales en comparación con el script original.

1.1. Carga de Datos

Se cargaron los mismos archivos CSV en DataFrames de Pandas, así como el archivo `mcc_codes.json`, de manera similar al Modelo 1.

```
1. import pandas as pd
2. import json
3.
4. # --- 1. Carga de Datos ---
5. # Carga de archivos CSV
6. try:
7.     df_transactions = pd.read_csv('transactions_data.csv')
8.     print("\nTransactions dataset loaded successfully!\n")
9. except FileNotFoundError:
10.    print("Error: 'transactions_data.csv' not found. Please ensure the file is in the correct
    directory.")
11.    exit()
12.
13. try:
14.    df_cards = pd.read_csv('cards_data.csv')
15.    print("\nCards data loaded successfully!\n")
16. except FileNotFoundError:
17.    print("Error: 'cards_data.csv' not found. Please ensure the file is in the correct
    directory.")
18.    exit()
19.
20. try:
21.    df_users = pd.read_csv('users_data.csv')
22.    print("\nUsers data loaded successfully!\n")
23. except FileNotFoundError:
24.    print("Error: 'users_data.csv' not found. Please ensure the file is in the correct
    directory.")
25.    exit()
26.
27. # Carga y procesamiento de mcc_codes.json
28. try:
29.     with open("mcc_codes.json", "r", encoding="utf-8") as f:
30.         mcc_dict = json.load(f)
31.         print("\nmcc_codes.json loaded successfully!\n")
32. except FileNotFoundError:
33.     print("Error: 'mcc_codes.json' not found. Please ensure the file is in the correct
    directory.")
34.     exit()
35. except json.JSONDecodeError:
36.     print("Error: Could not decode 'mcc_codes.json'. Please check file format.")
37.     exit()
38.
39. rows_mcc = []
40. for code_str, desc in mcc_dict.items():
41.     rows_mcc.append({
42.         "mcc": int(code_str),
43.         "mcc_description": desc
44.     })
45. df_mcc = pd.DataFrame(rows_mcc)
46. print("\nmcc_codes DataFrame created successfully!\n")
47.
48. print("\n=====\\n")
49.
```

1.2. Análisis Inicial y Extracción de Características

Este script realiza un análisis más exhaustivo de cada dataset antes de la unión, incluyendo la verificación de valores únicos, tipos de datos, estadísticas descriptivas y la presencia de valores faltantes. También se realizan transformaciones específicas para cada dataset.

Análisis de users_data:

Se analizan la edad actual, edad de jubilación, año y mes de nacimiento, género, latitud y longitud, ingresos per cápita, ingresos anuales, deuda total, historial crediticio y número de tarjetas de crédito. Se verifica la consistencia de los datos y se realizan imputaciones si es necesario.

Análisis de cards_data:

Se verifica la consistencia de los datos, la unicidad de las IDs, y la limpieza de los límites de crédito.

Análisis de transactions_data:

Se verifica la consistencia de los datos, se convierte la columna date a formato datetime y se extraen características temporales como hour, weekday, is_weekend, month e is_night. Se crea la columna isFraud (diferente de is_fraud en el modelo original, este es el nombre que usa el notebook aumentado) para identificar transacciones fraudulentas basándose en la columna errors.

Script de Análisis Inicial y Extracción de Características:

```
1. # --- 2. Análisis Inicial y Extracción de Características ---
2.
3. # 2.1. Análisis de Users Data
4. print("--- Initial Data Overview: Users Data ---")
5. print(df_users.head())
6. print("\nUsers Data Info:")
7. df_users.info()
8. print("\nUsers Data Description:")
9. print(df_users.describe())
10. print("\nMissing values in Users Data:")
11. print(df_users.isnull().sum())
12.
13. # Age and Birth Year Consistency Check
14. # Assuming current_age and birth_year are related to a current year.
15. # This check is just for informational purposes and does not modify data here.
16. current_year = pd.Timestamp.now().year
17. df_users['calculated_age'] = current_year - df_users['birth_year']
18. print(f"\nDiscrepancy in age vs birth_year (current_year={current_year}):")
19. print(df_users[df_users['calculated_age'] != df_users['current_age']][['current_age',
'birth_year', 'calculated_age']].head())
20.
21. # Handle missing values in gender (example: impute with mode or a new category)
22. if 'gender' in df_users.columns and df_users['gender'].isnull().any():
23.     mode_gender = df_users['gender'].mode()[0]
24.     df_users['gender'].fillna(mode_gender, inplace=True)
25.     print(f"\nMissing gender values imputed with mode: {mode_gender}")
```

```

26.
27. # Clean monetary columns in df_users
28. monetary_cols_users = ['per_capita_income', 'yearly_income', 'total_debt']
29. for col in monetary_cols_users:
30.     if col in df_users.columns:
31.         df_users[col] = df_users[col].replace({r'\$': '', r',': ''}, regex=True).astype(float)
32.         print(f"Cleaned and converted '{col}' in df_users to float.")
33.
34. # 2.2. Análisis de Cards Data
35. print("\n--- Initial Data Overview: Cards Data ---")
36. print(df_cards.head())
37. print("\nCards Data Info:")
38. df_cards.info()
39. print("\nCards Data Description:")
40. print(df_cards.describe())
41. print("\nMissing values in Cards Data:")
42. print(df_cards.isnull().sum())
43.
44. # Clean 'credit_limit' in df_cards
45. if 'credit_limit' in df_cards.columns:
46.     df_cards['credit_limit'] = df_cards['credit_limit'].replace({r'\$': '', r',': ''},
regex=True).astype(float)
47.     print("Cleaned and converted 'credit_limit' in df_cards to float.")
48.
49. # 2.3. Análisis de Transactions Data
50. print("\n--- Initial Data Overview: Transactions Data ---")
51. print(df_transactions.head())
52. print("\nTransactions Data Info:")
53. df_transactions.info()
54. print("\nTransactions Data Description:")
55. print(df_transactions.describe())
56. print("\nMissing values in Transactions Data:")
57. print(df_transactions.isnull().sum())
58.
59. # Convert 'date' to datetime and extract time features
60. if 'date' in df_transactions.columns:
61.     df_transactions['date'] = pd.to_datetime(df_transactions['date'])
62.     df_transactions['hour'] = df_transactions['date'].dt.hour
63.     df_transactions['weekday'] = df_transactions['date'].dt.weekday
64.     df_transactions['is_weekend'] = ((df_transactions['date'].dt.weekday == 5) |
(df_transactions['date'].dt.weekday == 6)).astype(int)
65.     df_transactions['month'] = df_transactions['date'].dt.month
66.     df_transactions['is_night'] = ((df_transactions['hour'] >= 22) | (df_transactions['hour']
<= 6)).astype(int)
67.     print("\nExtracted time features from 'date' column.")
68. else:
69.     print("\n'date' column not found in transactions data. Cannot extract time features.")
70.
71. # Clean 'amount' in df_transactions
72. if 'amount' in df_transactions.columns:
73.     df_transactions['amount'] = df_transactions['amount'].replace({r'\$': '', r',': ''},
regex=True).astype(float)
74.     print("Cleaned and converted 'amount' in df_transactions to float.")
75.
76. # Create 'isFraud' target variable based on 'errors' column
77. if 'errors' in df_transactions.columns:
78.     df_transactions['isFraud'] = df_transactions['errors'].apply(lambda x: 1 if pd.isna(x) and
x != '' else 0)
79.     print("Created 'isFraud' column based on 'errors'.")
80. else:
81.     print("\n'errors' column not found in transactions data. Cannot create 'isFraud' target
variable.")
82.
83. print("\n--- Possible values and counts for 'use_chip' ---")
84. print(df_transactions['use_chip'].value_counts())

```

```

85.
86. print("\n--- Possible values and counts for 'merchant_city' ---")
87. print(df_transactions['merchant_city'].value_counts().head())
88.
89. print("\n--- Possible values and counts for 'merchant_state' ---")
90. print(df_transactions['merchant_state'].value_counts().head())
91.
92. print("\n--- Possible values and counts for 'zip' ---")
93. print(df_transactions['zip'].value_counts().head())
94.
95. print("\n--- Possible values and counts for 'mcc' ---")
96. print(df_transactions['mcc'].value_counts().head())
97.
98. print("\n--- Possible values and counts for 'errors' ---")
99. print(df_transactions['errors'].value_counts().head())
100.
101. print("\n--- Possible values and counts for 'isFraud' ---")
102. print(df_transactions['isFraud'].value_counts())
103. print("\n--- Percentage of each value in 'isFraud' ---")
104. print(df_transactions['isFraud'].value_counts(normalize=True) * 100)
105.

```

1.3. Unión de DataFrames

Se realizó la unión de los DataFrames de forma similar al Modelo 1, utilizando las IDs correspondientes para combinar transactions_data, users_data, cards_data y mcc_codes. Se manejan los sufijos para evitar conflictos de nombres.

Script de Unión de DataFrames:

```

1. # --- 3. Unión de DataFrames ---
2. # Unir transactions_data con mcc_codes
3. df_merged = pd.merge(df_transactions, df_mcc, on='mcc', how='left')
4. print("Merged transactions_data with mcc_codes.")
5.
6. # Unir df_merged con users_data
7. df_merged = pd.merge(df_merged, df_users, left_on='client_id', right_on='id', how='left',
8. suffixes=('_transaction', '_user'))
9. print("Merged with users_data.")
10.
11. # Unir df_merged con cards_data
12. df_merged = pd.merge(df_merged, df_cards, left_on='card_id', right_on='id', how='left',
13. suffixes=('', '_card'))
14. print("Merged with cards_data.")
15.
16. # Renombrar columna 'id' de df_users a 'id_user' para evitar conflicto
17. df_merged.rename(columns={'id': 'id_user'}, inplace=True)
18. print("Renamed 'id' column from users_data to 'id_user'.")
19.
20. print("\nFinal Merged DataFrame Info:")
21. df_merged.info()
22. print("\nFinal Merged DataFrame Head:")
23. print(df_merged.head())
24. print("\nMissing values in Merged Data:")
25. print(df_merged.isnull().sum())
26.

```

1.4. Selección de Columnas Finales y Guardado del Dataset

Se seleccionaron las columnas finales para el dataset de salida, asegurando la inclusión de todas las características relevantes, y se guardó en un archivo CSV llamado `processed_dataset_aumentado.csv`.

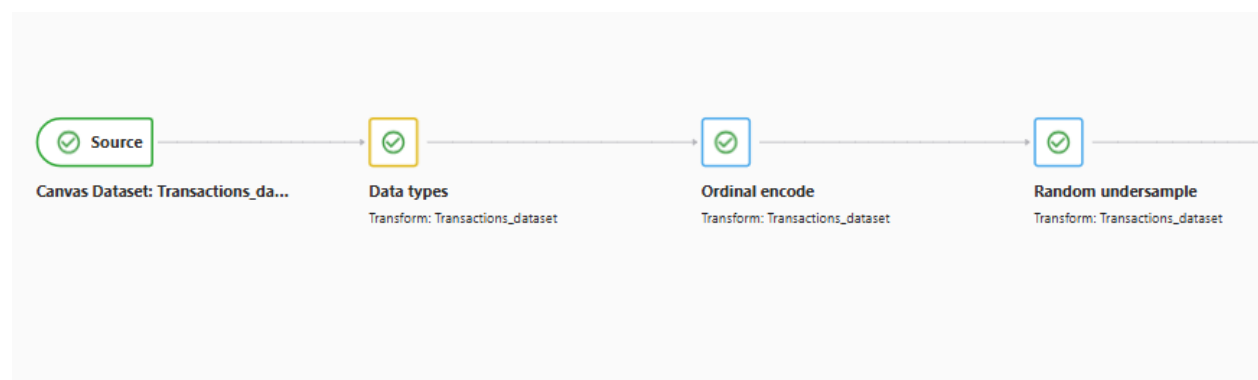
Script de Selección de Columnas y Guardado:

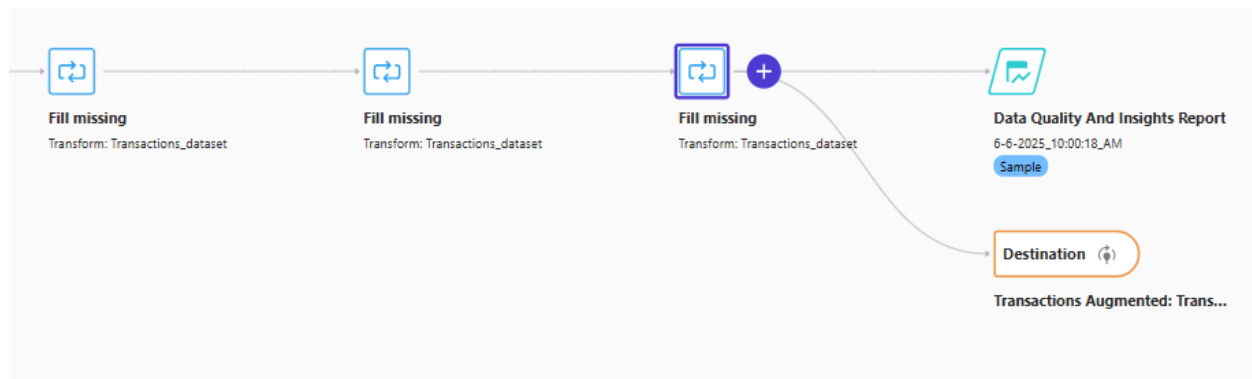
```
1. # --- 4. Selección de Columnas y Guardado ---
2. # Definir las columnas de salida deseadas
3. output_columns = [
4.     'id', 'date', 'client_id', 'card_id', 'amount', 'use_chip',
5.     'merchant_id', 'merchant_city', 'merchant_state', 'zip', 'mcc', 'errors',
6.     'mcc_description', 'isFraud', 'id_user', 'current_age', 'retirement_age',
7.     'birth_year', 'birth_month', 'gender', 'address', 'latitude', 'longitude',
8.     'per_capita_income', 'yearly_income', 'total_debt', 'credit_score',
9.     'num_credit_cards', 'card_brand', 'card_type', 'has_chip', 'card_on_dark_web',
10.    'credit_limit', 'hour', 'weekday', 'is_weekend', 'month', 'is_night'
11. ]
12.
13. # Filtrar solo las columnas que existen en df_merged para evitar errores
14. final_df_columns = [col for col in output_columns if col in df_merged.columns]
15. df_final = df_merged[final_df_columns]
16.
17. # Guardar el dataset procesado
18. df_final.to_csv('processed_dataset_aumentado.csv', index=False)
19.
20. print("\nDataset successfully processed and saved to 'processed_dataset_aumentado.csv'")
21.
```

2. Procesamiento de Datos con AWS SageMaker Data Wrangler

El dataset `processed_dataset_aumentado.csv` fue subido a AWS SageMaker Canvas y procesado en Data Wrangler. Este paso fue esencial para la preparación final del dataset antes del modelado.

2.1. Data Flow en Data Wrangler





Estas imágenes ilustran el flujo de trabajo en Data Wrangler para el dataset aumentado. Similar al modelo anterior, se muestra cómo los datos se importan, transforman y preparan para la exportación.

2.2. Análisis de Insights del Dataset

Imagen: “data-wrangler-insights-report_aumentado.png”

Esta imagen presenta el reporte de insights generado por Data Wrangler para el dataset aumentado. Este reporte proporciona una visión general del estado del dataset después del preprocesamiento en Python y antes de las transformaciones específicas de Data Wrangler, mostrando distribuciones, posibles anomalías y la calidad general de los datos.

2.3. Pasos de Procesamiento en Data Wrangler

Las transformaciones aplicadas en Data Wrangler para el dataset aumentado son consistentes con las del Modelo 1, dado el objetivo de preparación para el modelado en SageMaker Canvas:

- **Codificación de Variables Categóricas:** Las variables categóricas presentes en el dataset (use_chip, merchant_city, merchant_state, gender, mcc_description, card_brand, card_type, has_chip, card_on_dark_web) fueron transformadas a representaciones numéricas adecuadas para los algoritmos de aprendizaje automático.
- **Manejo de Valores Faltantes:** Se abordaron los valores faltantes en las columnas relevantes, utilizando estrategias de imputación o eliminación según la naturaleza y el impacto de los datos.
- **Balanceo de Clases:** Dado que el problema de detección de fraude se caracteriza por un desbalance significativo de clases (aproximadamente 1.58% de transacciones fraudulentas en el dataset original, como se vio en el análisis del script preprocesamiento_aumentado.ipynb), se aplicaron técnicas de balanceo

para mitigar este problema y mejorar la capacidad del modelo para detectar la clase minoritaria.

3. Creación y Evaluación del Modelo en SageMaker Canvas

El dataset preprocesado desde Data Wrangler se utilizó para entrenar el segundo modelo en SageMaker Canvas, también con el objetivo de optimizar el F1-score para la clasificación binaria de fraude.

3.1. Configuración del Modelo

- **Creación del Modelo en SageMaker:**

Select a column to predict

Choose the target column. The model that you build predicts values for the column that you select.

Target column

Value distribution

1
0

Model type

SageMaker Canvas automatically recommends the appropriate model type for your analysis.

💡 2 category prediction

Your model classifies isFraud into two categories.

[Configure model](#)

Standard build

Preview model

Esta imagen muestra la interfaz de SageMaker Canvas donde se inició el proceso de creación del modelo, seleccionando el dataset ya procesado y configurando los parámetros iniciales.

- **Métrica Objetivo:**

Objective metric

Select the metric that you want Canvas to optimize while building your model. By default, Canvas optimizes for the best metric based on your model type.

F1 - default ▼

The F1 score is a measure of a model's accuracy in binary classification tasks. It averages two important metrics: precision and recall. F1 works well for imbalanced datasets.

La métrica de optimización seleccionada fue nuevamente el F1-score, priorizando el equilibrio entre la precisión y la exhaustividad para la clase minoritaria (fraude), lo cual es crucial en escenarios de detección de fraude.

- **División de Datos:**

Data split

Specify how you want to split your data into training and validation sets. Canvas builds your model with the training set and verifies the model's accuracy with the validation set.

 Configuring the data split will default to Standard build.

Training set — %
80

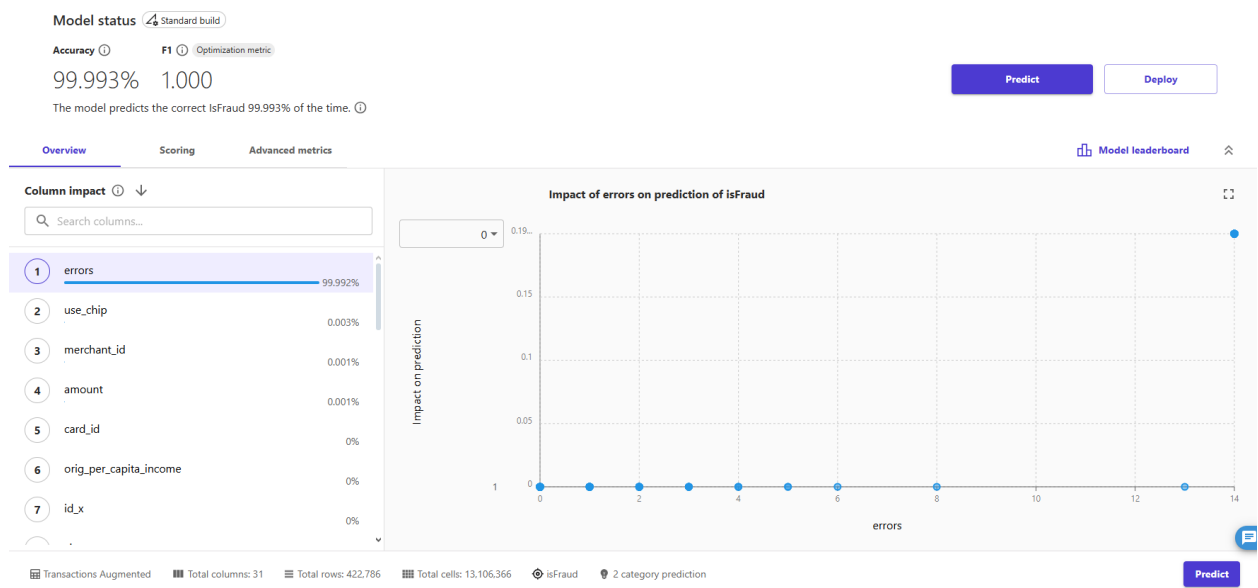
Validation set — %
20

La división de los datos en conjuntos de entrenamiento, validación y prueba se configuró para asegurar una evaluación robusta del modelo, evitando el sobreajuste y permitiendo estimar el rendimiento en datos no vistos.

3.2. Análisis del Rendimiento del Modelo

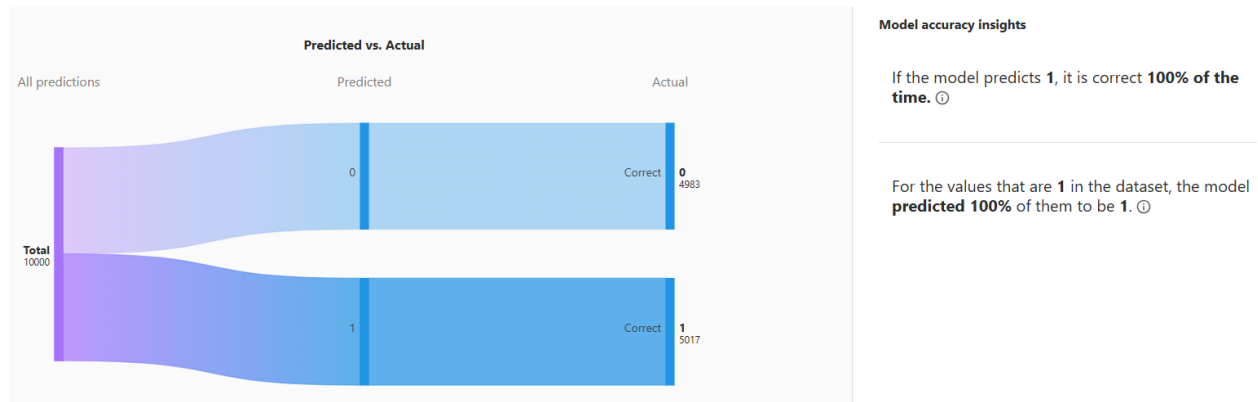
Tras el entrenamiento, se analizaron los resultados para evaluar el rendimiento del segundo modelo.

- **Análisis General del Modelo:**



Esta vista general ofrece un resumen de las métricas de rendimiento clave del modelo, permitiendo una rápida comprensión de su comportamiento.

- **Scoring del Modelo:**



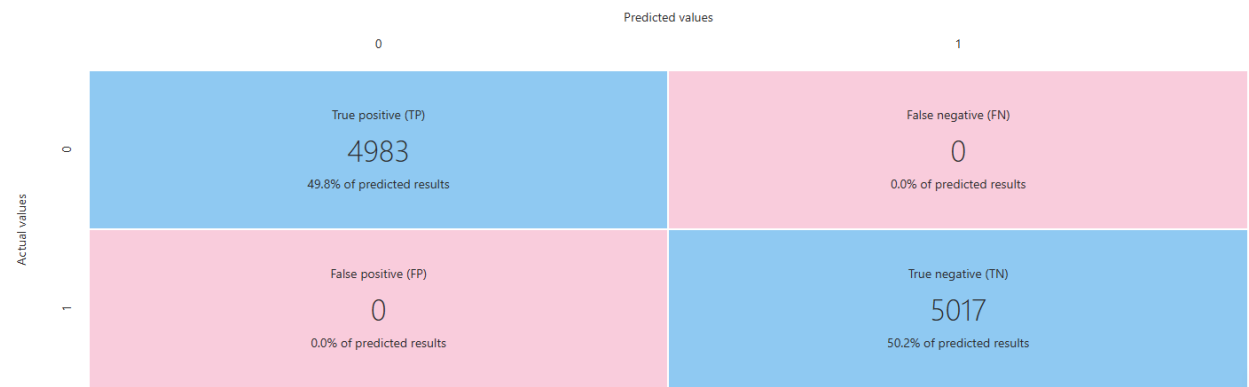
El reporte de scoring detalla cómo el modelo predijo las clases en el conjunto de prueba, indicando su capacidad para clasificar nuevas instancias.

- **Tabla de Métricas:**

Metric name	Value
precision	1.000
recall	1.000
accuracy	1.000
f1	1.000
auc	1.000

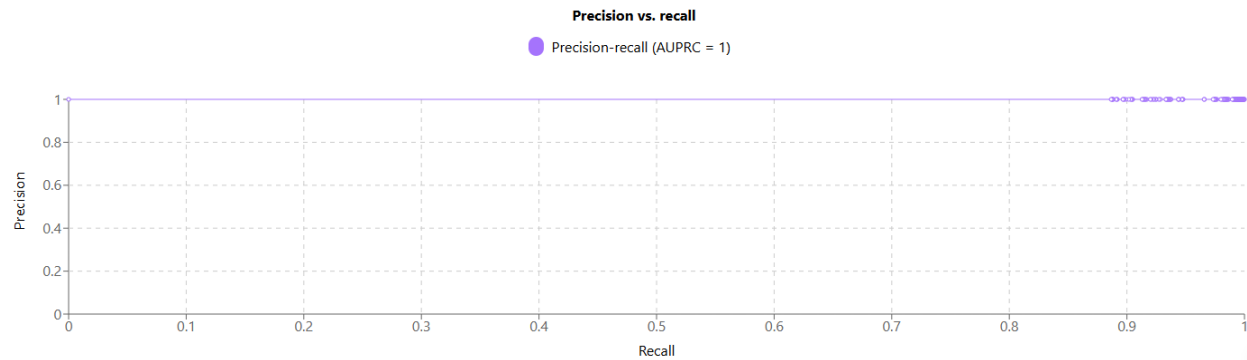
Esta tabla proporciona un desglose cuantitativo de las métricas de clasificación, incluyendo el F1-score, precisión, recall y exactitud, ofreciendo una visión completa del rendimiento del modelo.

- Matriz de Confusión:



La matriz de confusión es fundamental para visualizar los aciertos y errores del modelo. Permite identificar cuántas transacciones fraudulentas y no fraudulentas fueron clasificadas correctamente e incorrectamente.

- Curva de Precisión vs. Recall:



La curva de precisión-recall es vital para entender el rendimiento del modelo en la detección de fraudes, especialmente con datasets desbalanceados. Permite evaluar cómo el modelo equilibra la capacidad de identificar todos los fraudes (recall) con la minimización de falsos positivos (precisión).

Conclusiones

Los dos modelos de detección de fraude desarrollados demuestran la importancia de un preprocesamiento de datos exhaustivo, tanto con scripts de Python para la ingeniería inicial de características y unión de datasets, como con herramientas especializadas como AWS SageMaker Data Wrangler para el manejo de variables categóricas, valores faltantes y, crucialmente, el balanceo de clases.

Sobre la Calidad de los Modelos Entrenados

Ambos modelos fueron optimizados para el **F1-score**, una métrica idónea para problemas de clasificación con clases desbalanceadas como la detección de fraude. Las métricas de evaluación (tablas de métricas, matrices de confusión y curvas de precisión-recall) proporcionan una visión detallada del rendimiento. En el contexto de detección de fraude, un alto F1-score indica un buen equilibrio entre la capacidad de identificar transacciones fraudulentas reales (recall) y la minimización de falsos positivos (precisión), lo cual es fundamental para reducir la carga de revisiones manuales y evitar interrupciones innecesarias a los usuarios legítimos. La capacidad de observar estas métricas y visualizaciones directamente en SageMaker Canvas facilita una evaluación rápida y precisa de la calidad del modelo.

Sobre AWS SageMaker Canvas

AWS SageMaker Canvas demostró ser una plataforma robusta y accesible para el ciclo de vida del desarrollo de modelos de Machine Learning, incluso para usuarios con conocimientos limitados de codificación profunda. Sus principales ventajas incluyen:

- **Facilidad de Uso (Low-Code/No-Code):** Permite a los analistas de negocio y científicos de datos ciudadanos construir, entrenar y evaluar modelos de ML sin necesidad de escribir grandes cantidades de código, democratizando el acceso a las capacidades avanzadas de ML.
- **Integración con Data Wrangler:** La sección Data Wrangler de SageMaker Canvas es una herramienta excepcionalmente potente para el preprocesamiento de datos. Sus capacidades visuales para la limpieza, transformación y, en particular, el balanceo de datasets desbalanceados, simplifican drásticamente una de las fases más complejas del ML. Los reportes de insights que genera son invaluable para entender la calidad y distribución de los datos antes de aplicar cualquier transformación.
- **Gestión del Ciclo de Vida del Modelo:** SageMaker Canvas facilita la experimentación, el entrenamiento y la evaluación de modelos de forma iterativa. La capacidad de configurar métricas objetivo, dividir datos y visualizar resultados (como matrices de confusión y curvas de precisión-recall) en una interfaz unificada agiliza el proceso de desarrollo.
- **Escalabilidad:** Al ser parte de AWS, SageMaker Canvas se beneficia de la infraestructura escalable de la nube, permitiendo el manejo de grandes volúmenes de datos y el entrenamiento de modelos complejos sin preocuparse por la infraestructura subyacente.
- **Reproducibilidad y Documentación:** Aunque este documento es manual, la plataforma proporciona herramientas y reportes que facilitan la trazabilidad y la documentación de los pasos seguidos, lo que es crucial para la reproducibilidad de los resultados y para la colaboración en proyectos de ML.

En resumen, la combinación de scripts de Python para la extracción y preparación inicial de datos, junto con las capacidades de preprocesamiento de Data Wrangler y las funciones de modelado y evaluación de SageMaker Canvas, proporciona un entorno integral y eficiente para desarrollar soluciones de Machine Learning de alta calidad, especialmente en tareas críticas como la detección de fraude, donde el balance de clases y la interpretación de métricas son esenciales.