



MÁSTER EN DATA SCIENCE Y BUSINESS ANALYTICS

**USO DE MODELOS DE APRENDIZAJE  
AUTOMÁTICO PARA LA DETECCIÓN DE  
TRANSACCIONES FRAUDULENTAS CON  
TARJETAS DE CRÉDITO**

TFM elaborado por: Sara Estefanía Chamseddine Fajardo

Tutor de TFM: Abel Ángel Soriano Vázquez

- Madrid. Noviembre, 2023 -

# Índice

<i>Resumen .....</i>	<b>11</b>
<b>1. Introducción.....</b>	<b>12</b>
1.1. Justificación .....	12
1.2. Antecedentes .....	12
1.3. Objetivos y Alcance.....	14
1.3.1. Objetivo General .....	14
1.3.2. Objetivos específicos.....	15
<b>2. Marco Teórico .....</b>	<b>16</b>
2.1. Detección de anomalías .....	16
2.2. Técnicas de preprocesamiento de datos .....	16
2.2.1. Balanceo de clases .....	16
2.2.2. Normalización .....	17
2.2.3. Estandarización .....	18
2.2.4. Codificación de variables categóricas .....	18
2.3. Tipos de modelos de ML.....	19
2.3.1. Regresión Logística.....	19
2.3.2. Support Vector Machines (SVM) o Máquinas de Soporte Vectorial .....	20
2.3.3. Naive Bayes .....	22
2.3.4. Isolation Forest.....	23
2.4. Técnicas de evaluación de modelos de ML .....	24
2.4.1. Matriz de confusión .....	24

2.4.2.	Métricas de rendimiento en problemas de clasificación binaria.....	25
2.4.3.	Característica operativa del receptor o Receiver operating characteristic (ROC).....	25
<b>2.5.</b>	<b>Técnicas de validación y optimización de modelos de aprendizaje supervisado.....</b>	<b>26</b>
2.5.1.	<i>k</i> -fold Cross Validation o Validación Cruzada de <i>k</i> -pliegues .....	26
2.5.2.	Búsqueda de los mejores hiperparámetros con Búsqueda en Cuadrícula o Grid Search .....	27
2.5.3.	Bootstrap.....	27
2.5.4.	Selección de características con la Prueba Chi-Cuadrada ( $\chi^2$ ) .....	28
2.5.5.	Reducción de Dimensionalidad con PCA o Análisis de Componente Principales .....	28
<b>2.6.</b>	<b>Herramientas y librerías .....</b>	<b>30</b>
<b>2.7.</b>	<b>Particionamiento en Spark .....</b>	<b>30</b>
2.7.1.	Data Skew o sesgo de datos.....	30
2.7.2.	Sugerencias para elegir el número de particiones.....	34
<b>3.</b>	<b>Metodología .....</b>	<b>36</b>
<b>3.1.</b>	<b>Datos utilizados .....</b>	<b>36</b>
3.1.1.	Transacciones (credit_card_transactions-ibm_v2.csv).....	37
3.1.2.	Tarjetas (sd254_cards.csv).....	38
3.1.3.	Usuarios (sd254_users.csv).....	39
3.1.4.	MCC Codes (mcc_codes.csv).....	40
<b>3.2.</b>	<b>Herramientas.....</b>	<b>41</b>
3.2.1.	Kaggle .....	41
3.2.2.	Microsoft Azure Cloud .....	42
3.2.3.	Databricks.....	42
3.2.4.	Python / PySpark .....	43
3.2.5.	Scala .....	43

3.2.6. SQL.....	44
<b>3.3. Configuración del entorno de trabajo .....</b>	<b>44</b>
3.3.1. Creación de Instancia de Azure Databricks.....	44
3.3.2. Creación de Clúster con Apache Spark en Databricks .....	48
3.3.3. Librería y Credenciales de Kaggle.....	55
<b>4. Implementación .....</b>	<b>58</b>
4.1. Extracción de datos.....	58
4.2. Análisis Exploratorio, transformación y feature engineering.....	61
4.2.1. Transacciones.....	62
4.2.2. Tarjetas.....	86
4.2.3. Usuarios.....	95
4.2.4. Tabla final para el modelo.....	99
4.3. Modelado de Aprendizaje Automático .....	105
4.3.1. Preparación de datos .....	110
4.3.2. Reducción de dimensionalidad .....	118
4.3.3. Entrenamiento de modelos .....	120
<b>5. Resultados .....</b>	<b>128</b>
5.1. SVM .....	129
5.2. Regresión Logística .....	131
5.3. Isolation Forest.....	135
<b>6. Conclusiones .....</b>	<b>137</b>
6.1. Análisis de resultados .....	137

6.2.	Áreas de mejora y trabajo futuro.....	139
7.	<i>Referencias</i> .....	142
8.	<i>Anexos</i> .....	147
8.1.	1_extraccion .....	147
8.2.	2_1_eda_transformacion_transactions.....	147
8.3.	2_2_eda_transformacion_cards .....	147
8.4.	2_3_eda_transformacion_users .....	147
8.5.	3_1_preparacion_dataset_modelo .....	147
8.6.	3_2_preparacion_dataset_modelo_mllib.....	147
8.7.	3_3_reduccion_dimensionalidad .....	147
8.8.	4_1_entrenamiento_con_svm.....	147
8.9.	4_2_entrenamiento_con_Regresion_Logistica .....	147
8.10.	4_3_entrenamiento_con_isolation_forest .....	147
8.11.	5_metricas_validacion .....	147

# Índice de Figuras

Figura 1 – Definición de la matriz de confusión binaria .....	24
Figura 2 – Diagnosticar el sesgo visualizando el conteo de datos dentro de las particiones.....	31
Figura 3 – Conteo de registros por partición .....	32
Figura 4 – Conjunto de datos sin sesgo .....	32
Figura 5 – Conjunto de datos con sesgo.....	32
Figura 6 – Columna de salt para particionar los datos.....	33
Figura 7 – Visualización del resultado del reparticionamiento .....	34
Figura 8 - Ejemplo del contenido del archivo de transacciones (transactions) del dataset. ....	37
Figura 9 - Ejemplo del contenido del archivo de tarjetas (cards) del dataset. ....	38
Figura 10 - Ejemplo del contenido del archivo de usuarios (users) del dataset. ....	39
Figura 11 - Ejemplo del contenido del archivo de MCC Codes.....	41
Figura 12 - Creación de un grupo de recursos (1) .....	44
Figura 13 - Creación de un grupo de recursos (2) .....	45
Figura 14 - Creación de un grupo de recursos (3) .....	45
Figura 15 – Creación de instancia de Databricks en Azure (1) .....	45
Figura 16 – Creación de instancia de Databricks en Azure (2) .....	46
Figura 17 – Creación de instancia de Databricks en Azure (3) .....	46
Figura 18 – Creación de instancia de Databricks en Azure (4) .....	47
Figura 19 – Acceso a la instancia de Databricks .....	47
Figura 20 – Clústers usados en la tesis .....	48
Figura 21 – Resumen del clúster para ML.....	48
Figura 22 – Clúster para ML .....	49
Figura 23 - Resumen del clúster para Feature Engineering.....	50
Figura 24 – Cluster para Feature Engineering .....	50
Figura 25 – Resumen del clúster para DataViz .....	51
Figura 26 - Resumen del clúster para DataViz.....	52
Figura 27 – Resumen del clúster para Dimensionality Reduction .....	53
Figura 28 - Resumen del clúster para Dimensionality Reduction .....	53
Figura 29 – Resumen del clúster para EDA .....	54
Figura 30 - Resumen del clúster para EDA .....	54
Figura 31 – Instalación de librería de Kaggle en el clúster de Databricks (1).....	55
Figura 32 - Instalación de librería de Kaggle en el clúster de Databricks (2). .....	55
Figura 33 – Generación de archivo de credenciales de Kaggle (1) .....	56
Figura 34 - Generación de archivo de credenciales de Kaggle (2).....	56
Figura 35 - Generación de archivo de credenciales de Kaggle (3).....	57
Figura 36 - Generación de archivo de credenciales de Kaggle (4).....	57
Figura 37 - Generación de archivo de credenciales de Kaggle (5).....	57
Figura 38 – Creación de base de datos en Databricks. .....	58
Figura 39 – Lectura del archivo de credenciales de Kaggle .....	58
Figura 40 - Almacenamiento de credenciales de Kaggle como variables de entorno .....	59
Figura 41 – Descarga y descompresión del dataset.....	59

Figura 42 – Almacenamiento de archivos del dataset en el DBFS.....	60
Figura 43 – Creación de raw delta tables.....	61
Figura 44 – Crear visualizaciones y perfiles de datos en Databricks .....	62
Figura 45 – Lectura de la tabla raw_transactions .....	62
Figura 46 – Tabla de transacciones: crear columna de fecha. ....	63
Figura 47 – Perfil de datos numéricos de la tabla de transacciones .....	63
Figura 48 – Perfil de datos categóricos de la tabla de transacciones.....	65
Figura 49 – Distribución Logarítmica de Transacciones Fraudulentas y No Fraudulentas .....	66
Figura 50 – Diagrama de barras: tipo de transacción vs es fraude .....	67
Figura 51 – Diagrama de barras: tipo de error vs es fraude.....	67
Figura 52 – Eliminación de duplicados para la tabla de transacciones .....	68
Figura 53 – Esquema de la tabla de transacciones antes de la transformación de datos .....	68
Figura 54 – Tabla de transacciones: cambio del tipo de dato de la columna is_fraud .....	68
Figura 55 – Eliminación de variables innecesarias en la tabla de transacciones.....	69
Figura 56 – Transformación de la variable amount.....	69
Figura 57 – Creación de histograma para la variable amount .....	69
Figura 58 – Histograma: distribución de los valores de “amount” .....	70
Figura 59 – Tabla de transacciones: cantidad de devoluciones.....	70
Figura 60 - Creación de histograma para la variable amount en las transacciones fraudulentas .....	71
Figura 61 - Histograma para la variable amount en las transacciones fraudulentas.....	71
Figura 62 – Boxplot para el monto de las transacciones fraudulentas por tipo de transacción .....	72
Figura 63 – Cálculos estadísticos para transacciones fraudulentas y no fraudulentas que usan o no usan chip .....	73
Figura 64 – Creación de Boxplot con los montos de las transacciones según si fue fraude o no .....	74
Figura 65 - Boxplot con los montos de las transacciones según si fue fraude o no .....	75
Figura 66 – Creación de gráficas con el comportamiento de transacciones en el tiempo .....	75
Figura 67 – Gráfica con el comportamiento de transacciones no fraudulentas en el tiempo .....	76
Figura 68 - Gráfica con el comportamiento de transacciones fraudulentas en el tiempo .....	76
Figura 69 – Enriquecimiento del dataset con los nombres de los MCC .....	77
Figura 70 – Transacciones fraudulentas por MCC .....	78
Figura 71 – Distribución de los valores de la variable errors .....	79
Figura 72 – One Hot Encoding a la columna de errors .....	80
Figura 73 – Cantidad de transacciones fraudulentas y no fraudulentas por error.....	81
Figura 74 – Distribución de los valores de la variable use_chip .....	82
Figura 75 - Distribución de los valores de la variable merchant_city .....	82
Figura 76 – Transacciones con merchant_city online y use_chip diferente de online .....	83
Figura 77 – Solución a calidad de datos en la variable use_chip .....	83
Figura 78 – Imputación de datos para la variable merchant_state.....	84
Figura 79 – Diagrama de barras: transacciones fraudulentas por merchant_state.....	85
Figura 80 - Eliminación de variables innecesarias en la tabla de transacciones.....	85
Figura 81 – Esquema final de la tabla de transacciones en staging .....	86
Figura 82 – Visualización del dataset de tarjetas (cards).....	86
Figura 83 – Perfil de datos numéricos para tabla de tarjetas .....	87
Figura 84 - Perfil de datos categóricos para tabla de tarjetas .....	88

Figura 85 - Diagrama de barras: cantidad de tarjetas por marca .....	89
Figura 86 – Diagrama de barras: tipo de tarjeta por marca .....	89
Figura 87 - Diagrama de barras: tarjetas con chip y sin chip por marca .....	90
Figura 88 - Diagrama de barras: cantidad de tarjetas emitidas.....	90
Figura 89 – Eliminación de duplicados en la tabla de tarjetas .....	91
Figura 90 – Esquema inicial de la tabla de tarjetas.....	91
Figura 91 – Eliminación de columnas innecesarias de la tabla de tarjetas .....	92
Figura 92 – Reemplazo de la variable has_chip por ceros y unos .....	92
Figura 93 – Transformación de la columna credit_limit de la tabla de tarjetas.....	93
Figura 94 – Cambio de formato para las columnas acct_open_date y year_pin_last_changed .....	93
Figura 95 – Renombramiento de la columna card_index .....	94
Figura 96 – Esquema final de la tabla de tarjetas .....	94
Figura 97 – Guardar la tabla de tarjetas en staging.....	94
Figura 98 – Validación de la creación de la tabla de tarjetas .....	94
Figura 99 – Perfil de datos para las columnas categóricas de la tabla de usuarios.....	95
Figura 100 – Perfil de datos para las columnas numéricas de la tabla de usuarios.....	96
Figura 101 - Mapa de usuarios por género .....	97
Figura 102 – Distribución de FICO Score para los usuarios .....	97
Figura 103 – Distribución de edad de los usuarios.....	98
Figura 104 – Distribución de la edad de retiro por género .....	98
Figura 105 – Cantidad de usuarios por género .....	98
Figura 106 – Lectura de tabla de staging de transacciones.....	99
Figura 107 - Lectura de tabla de staging de tarjetas .....	99
Figura 108 – Join entre las tablas de transacciones y tarjetas.....	100
Figura 109 – Diagrama de barras: transacciones fraudulentas por marca de tarjeta .....	101
Figura 110 – Diagrama de barras: transacciones fraudulentas que usan o no chip .....	101
Figura 111 – Diagrama de barras: transacciones fraudulentas por tipo de tarjeta.....	101
Figura 112 – Cálculo de la antigüedad del usuario en años al momento de la transacción.....	102
Figura 113 - Cálculo de la antigüedad del usuario en años al momento de la transacción .....	102
Figura 114 – Eliminación de las columnas innecesarias del dataset final .....	103
Figura 115 – Cantidad de transacciones fraudulentas recibidas por cada año de antigüedad .....	103
Figura 116 – Transacciones fraudulentas por años transcurridos desde el último cambio de PIN .....	104
Figura 117 – Esquema final de la tabla de transacciones para el modelo. .....	104
Figura 118 – Escritura de la tabla final de transacciones en producción.....	105
Figura 119 – Renombre de la columna is_fraud a label .....	110
Figura 120 – Reparticionamiento del conjunto de datos para el modelo .....	110
Figura 121 – Reparticionamiento de los datos para el modelo.....	111
Figura 122 – Eliminación de las columnas de usuario y tarjeta .....	111
Figura 123 – Creación de features de fecha.....	112
Figura 124 – Codificación de las variables de fecha .....	113
Figura 125 – Análisis de variables con muchas categorías únicas .....	113
Figura 126 – Porcentaje de categorías únicas para merchant_city .....	114
Figura 127 – Eliminación de la columna merchant_city .....	114
Figura 128 – StringIndexer a las variables merchant_state y mcc_category.....	115

Figura 129 – StringIndexer a las variables categóricas .....	115
Figura 130 – Unir variables en un vector con VectorAssembler.....	116
Figura 131 – División estratificada de los datos en train y test .....	117
Figura 132 – Construcción de los conjuntos de entrenamiento y prueba .....	118
Figura 133 – Lectura del dataset para aplicar PCA .....	118
Figura 134 - Implementación de PCA al modelo .....	119
Figura 135 – Ajustar y transformar PCA.....	119
Figura 136 – Almacenar conjunto de train y test con PCA. ....	120
Figura 137 – Lectura y particionamiento del dataset.....	120
Figura 138 – Cálculo de ratio entre clases.....	121
Figura 139 – Duplicar clase minoritaria .....	121
Figura 140 – Parámetros para SVM .....	122
Figura 141 – Entrenamiento del modelo SVM.....	123
Figura 142 – Parámetros para regresión logística.....	124
Figura 143 – Entrenamiento del modelo de regresión logística .....	124
Figura 144 – Lectura y particionamiento del dataset en Scala .....	125
Figura 145 – Isolation Forest en Scala sin bootstrapping.....	126
Figura 146 - Isolation Forest en Scala con bootstrapping .....	127
Figura 147 – Cálculo de métricas de validación de modelos .....	128
Figura 148 – Resultados del experimento SVM en Databricks.....	129
Figura 149 – Valores de ROC para el modelo SVM .....	130
Figura 150 – Lectura del resultado del modelo SVM.....	130
Figura 151 – Mostrar los resultados de las predicciones para SVM .....	131
Figura 152 – Cálculo de métricas de validación para SVM .....	131
Figura 153 - Resultados del experimento de regresión logística en Databricks.....	132
Figura 154 – ROC del modelo de regresión logística.....	133
Figura 155 – Lectura del resultado del modelo de Regresión Logística .....	133
Figura 156 - Mostrar los resultados de las predicciones para regresión logística .....	134
Figura 157 – Graficar curva ROC del modelo de regresión logística .....	134
Figura 158 – Curva ROC del modelo de regresión logísticas .....	135
Figura 159 - Lectura del resultado del modelo de Isolation Forest .....	135
Figura 160 - Mostrar los resultados de las predicciones para Isolation Forest.....	136
Figura 161 – Cálculo de métricas para Isolation Forest .....	136
Figura 162 – Resultados de los modelos .....	137

# Índice de Tablas

Tabla 1 - Ficha técnica del dataset “Credit Card Transaction”.	36
Tabla 2 - Ficha técnica del dataset “MCC Codes”.	37
Tabla 3 - Columnas del archivo de transacciones (transactions) del dataset.	37
Tabla 4 - Columnas del archivo de tarjetas (cards) del dataset.	39
Tabla 5 - Columnas del archivo de usuarios (users) del dataset.	40
Tabla 6 - Columnas del archivo de MCC Codes.	41
Tabla 7 – Experimento #1.....	105
Tabla 8 – Experimento #2.....	106
Tabla 9 – Experimento #3.....	106
Tabla 10 – Experimento #4.....	107
Tabla 11 – Experimento #5.....	108
Tabla 12 – Experimento #6.....	108
Tabla 13 – Características del experimento final.....	109
Tabla 14 – Resultados de las métricas de los modelos implementados.....	138

## Resumen

Esta tesis se enfoca en la detección de anomalías, específicamente en la detección de fraudes en transacciones con tarjetas de crédito en entornos distribuidos. El objetivo principal es implementar y evaluar varios modelos de aprendizaje automático para identificar cuál ofrece el mejor rendimiento en esta tarea crítica.

Los objetivos específicos incluyen la extracción y procesamiento de datos de transacciones, un análisis exploratorio de los datos, la transformación de los datos para que se adapten al caso de uso, la realización de feature engineering para mejorar la calidad de los datos, la codificación de variables para su utilización en los modelos, el equilibrio del conjunto de datos para dividirlo en conjuntos de entrenamiento y prueba, la reducción de la dimensionalidad y, finalmente, la implementación y entrenamiento de tres modelos de aprendizaje automático. Para todo este proceso se utiliza Databricks en Azure Cloud, con Python.

Es importante destacar que el énfasis está en no pasar por alto transacciones fraudulentas. Dado el desequilibrio en los datos, se prioriza la sensibilidad (recall) como métrica clave. La sensibilidad mide la capacidad del modelo para identificar correctamente todas las transacciones fraudulentas, minimizando así los falsos negativos, es decir, las transacciones fraudulentas que no se detectan. Este enfoque es esencial cuando la prioridad es evitar errores graves.

Los resultados se analizan en profundidad para tres modelos: SVM, Regresión Logística e Isolation Forest. Dado que la prioridad es minimizar la cantidad de transacciones fraudulentas pasadas por alto, SVM resulta ser la mejor opción entre estos tres modelos para este caso, debido a su alta sensibilidad. Sin embargo, también se debe considerar que la precisión de SVM es baja, lo que significa que puede generar más falsos positivos.

# 1. Introducción

En la era actual, con el auge de las nuevas tecnologías, las tarjetas de crédito se han convertido en un componente crucial de la vida cotidiana. Esto se debe a su versatilidad para su uso en una amplia gama de transacciones, tanto en compras en línea como en establecimientos físicos. Sin embargo, este progreso tecnológico y financiero también ha dado lugar a un aumento en la creatividad de individuos maliciosos que buscan aprovecharse de estos avances para su propio beneficio. Se ha vuelto más evidente que nunca el robo y la falsificación de tarjetas de crédito.

## 1.1. Justificación

Por lo tanto, es de suma importancia implementar nuevos procedimientos destinados a la identificación de transacciones potencialmente fraudulentas. Este enfoque resulta fundamental para combatir esta creciente amenaza. Al hacerlo, podemos salvaguardar los intereses tanto de los consumidores como de las entidades financieras, evitando pérdidas financieras significativas. Asimismo, contribuye a fortalecer y consolidar la confianza en el sistema financiero en su conjunto, fomenta el cumplimiento de la normativa vigente y permite una utilización responsable de las avanzadas tecnologías disponibles en la actualidad.

## 1.2. Antecedentes

Construir modelos de Machine Learning para fraude financiero, representa un gran desafío pues normalmente se disponen de conjuntos de datos muy grandes que tienen pocas transacciones fraudulentas y esto genera un desequilibrio de clases. Cuando esto ocurre, los algoritmos

normales de ML “tienden a maximizar la precisión general (exactitud) tendiendo a clasificar todas las observaciones como instancias de la clase mayoritaria. Esto se traduce en una baja capacidad de predicción (recall) para la clase de nuestro interés, que en el caso concreto de fraudes financieros corresponde a la clase minoritaria”, es decir, todas aquellas que son fraudulentas” (Frola, Alvez, & Chesñevar, 2020).

El propósito central de esta tesis es abordar el desafío de equilibrar las clases en el conjunto de datos para permitir un entrenamiento efectivo de modelos de detección de fraudes en un entorno de procesamiento distribuido, especialmente diseñado para gestionar grandes volúmenes de datos (Big Data). Como referencia para elaborar esta tesis, se toman como base los siguientes proyectos:

- **Un primer acercamiento a un modelo predictivo ajustable por umbrales para detección de fraudes financieros:** El fraude en las transacciones financieras con tarjetas de crédito y débito es un problema estudiado debido a su impacto económico. Abordar este desafío con machine learning es complicado debido a la falta de etiquetas en los datos y el desequilibrio entre clases. En este trabajo, se propone un enfoque innovador que ajusta el umbral de probabilidad de detección de fraude. A través de experimentos, se demuestra que esta estrategia es eficaz y ofrece una alternativa válida para detectar fraudes de manera efectiva (Frola, Alvez, & Chesñevar, 2020).
- **Mapeo de las Técnicas de Aprendizaje Automático Usados en Fraudes con Tarjeta De Crédito:** El fraude financiero es un delito que afecta a personas y organizaciones, especialmente en el caso de tarjetas de crédito, donde las pérdidas representan alrededor del 5% de los ingresos anuales. Detectar y prevenir el fraude de manera temprana es crucial para mantener la confianza de los clientes. Este capítulo de investigación analiza técnicas de aprendizaje automático aplicadas al fraude con tarjetas de crédito. Se concluye

que estas técnicas, como redes neuronales, regresión logística y árboles de decisión, han contribuido significativamente a la detección efectiva y oportuna del fraude en este contexto (Gutiérrez-Portela, Perdomo-Guerrero , Mario Heimer, Hernández-Aros , & Quiceno-Castañeda, 2020).

- **Detección de Fraude en Tarjetas de Crédito:** Este trabajo de memoria propone una metodología para predecir fraudes en transacciones con tarjetas de crédito en el sector minorista. Dado el aumento del fraude en ventas minoristas y la falta de etiquetas de fraude en las bases de datos, se etiquetaron manualmente transacciones fraudulentas. Se construyeron modelos de clasificación, incluyendo support vector machines, redes neuronales, árboles de decisión y regresión logística, para asignar probabilidades de fraude a cada transacción. Los modelos más complejos, como SVM y redes neuronales, superaron a la regresión logística. La segmentación por categorías mostró mejores resultados, destacando la utilidad de múltiples modelos especializados. Reducir la proporción de transacciones fraudulentas mejoró la precisión (Cepeda García, 2012).

### **1.3. Objetivos y Alcance**

#### **1.3.1. Objetivo General**

Implementar y evaluar múltiples modelos de aprendizaje automático en un entorno distribuido para determinar cuál proporciona un rendimiento óptimo en la detección de anomalías en transacciones con tarjetas de crédito.

### 1.3.2. Objetivos específicos

- Extraer y procesar un conjunto de datos con transacciones con tarjetas de crédito.
- Realizar el análisis exploratorio del conjunto de datos.
- Transformar el conjunto de datos para adaptarlo a las necesidades del caso de uso.
- Implementar feature engineering en el conjunto de datos.
- Codificar las variables del conjunto de datos para ser utilizadas por el modelo.
- Dividir el conjunto de datos de manera balanceada para train y test.
- Reducir la dimensionalidad del conjunto de datos.
- Entrenar tres (3) diferentes modelos de aprendizaje automático.
- Evaluar los diferentes modelos de aprendizaje automático implementados y determinar el mejor.

## 2. Marco Teórico

### 2.1. Detección de anomalías

El problema de la detección de fraudes es parte de un proceso llamado detección de anomalías.

La detección de anomalías se define como el proceso de identificar puntos de datos en un set de datos o sistema fuera de los comunes. Estas anomalías pueden aparecer como datos atípicos, cambios inesperados o errores, dependiendo del conjunto de datos y los tipos de datos analizados. Este proceso es de suma utilidad pues permite abordar problemas potenciales o amenazas con rapidez y de forma eficiente para mantener la integridad de los sistemas. Además de la detección de fraudes, existen otras aplicaciones en otras áreas como la ciberseguridad para la detección de intrusos en redes y el monitoreo de sistemas y aplicaciones para evitar interrupciones en los sistemas (Elastic, 2021).

### 2.2. Técnicas de preprocesamiento de datos

Existen múltiples técnicas de procesamiento de datos. A continuación, se explican algunas de las más usadas.

#### 2.2.1. Balanceo de clases

La detección de anomalías nos enfrenta a nuevos desafíos y limitaciones. Uno de ellos que es de suma importancia cuando se usan modelos de Machine Learning es la cantidad insuficiente de datos etiquetados para su detección (Elastic, 2021). Cuando esto sucede, se dice que se tiene un

conjunto de datos desequilibrado. Un conjunto de datos desequilibrado, en problemas de clasificación, es un conjunto de datos donde los ejemplos de una clase son significativamente mayores que los ejemplos de la otra clase, lo que provoca que los modelos de clasificación tiendan a predecir de forma ciega nuevas observaciones como si fueran observaciones de la clase dominante (Wan, 2020). Para poder enfrentar estas limitaciones, podemos valernos de distintas técnicas de preprocesamiento de datos. Algunas de ellas son:

- **Oversampling y Subsampling:** Estas técnicas permiten balancear el número de observaciones de cada clase, ya sea duplicando las observaciones de la clase minoritaria para que estén al mismo nivel que la clase mayoritaria (oversampling), o eliminando observaciones de la clase mayoritaria para que estén al nivel de la clase minoritaria (undersampling) (Wan, 2020).
- **SMOTE (Synthetic Minority Over-sampling Technique):** Esta es una técnica más sofisticada de Oversampling que permite generar nuevas observaciones sintéticas en vez de duplicar las observaciones (oversampling con reemplazo) a partir de un proceso basado en  $k$  vecinos más cercanos. (Chawla, Bowyer, Hall, & Kegelmeyer, 2002).

### **2.2.2. Normalización**

“La normalización busca ajustar los datos a un rango específico o a una escala definida” (Urrego, 2022). La normalización se aplica cuando:

- El modelo a aplicar está basado en distancia, como por ejemplo, k-NN y clustering.
- Se necesita una escala comparable.
- Cuando se necesitan datos cercanos a una distribución normal, para aplicar modelos como las redes neuronales.

- Cuando se necesitan ajustar los datos a un rango específico, como por ejemplo, en el procesamiento de imágenes o señales.

### **2.2.3. Estandarización**

“La estandarización se enfoca en la transformación de los datos para que tengan una media de cero y una desviación estándar de uno” (Urrego, 2022). Estandarizar los datos permite:

- Garantizar una comparabilidad precisa y equilibrada entre diferentes variables. Puede ser útil al momento de aplicar técnicas como PCA.
- Equilibrar el impacto de las características para mejorar modelos sensibles a las escalas de las variables (Ej.: regresión logística, SVM, clustering, entre otros).
- Aplicar una escala común a variables con unidades de medida diferentes.

### **2.2.4. Codificación de variables categóricas**

Todos los modelos usados en la tesis requieren que las variables categóricas se codifiquen a variables numéricas. Para ello, se hacen uso de dos técnicas distintas:

#### **2.2.4.1. One-Hot Encoding**

Esta técnica permite codificar las variables categóricas a partir de un vector cuyo tamaño es equivalente al número de categorías presentes en el conjunto de datos. Para cada observación, si dicha observación pertenece a la  $i$ -ésima categoría, la  $i$ -ésima componente del vector adquiere un

valor de 1 mientras el resto de las posiciones se establecen con un valor de 0 (DeepAI, s.f.). Por ejemplo, si una variable categórica  $X$  se establece como:  $X = [blanco \quad negro \quad gris]$

El proceso de codificación con One-Hot Encoding crearía 3 posibles vectores para cada observación de acuerdo con su posición en  $X$ : Si  $X = blanco$ :  $[1 \quad 0 \quad 0]$  ; Si  $X = negro$ :  $[0 \quad 1 \quad 0]$  ; Si  $X = gris$ :  $[0 \quad 0 \quad 1]$

#### **2.2.4.2. StringIndexer en PySpark**

Este procedimiento está disponible en PySpark como parte de MLlib. Permite codificar una variable categórica mapeando sus valores con los valores de otra columna de índices cuyos valores se encuentran dentro del rango [0, número de categorías). Por defecto, la columna de índices se ordena de acuerdo con la frecuencia de las categorías, siendo 0 la categoría más frecuente (Apache Spark, 2021).

### **2.3. Tipos de modelos de ML**

Existen numerosos tipos de modelos de clasificación, a continuación, se muestran los que se prueban en la tesis:

#### **2.3.1. Regresión Logística**

Este modelo es una extensión del modelo de Regresión Lineal que se diferencia por modelar la probabilidad de que la clase Y pertenezca a una categoría específica. Para ello, usa la función

$$\text{logística que se define como: } p(X) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}}$$

Esta función, a diferencia de la función lineal, permite obtener valores en el rango de 0 a 1 para todos los valores de X (James, Witten, Hastie, Tibshirani, & Taylor, 2023).

Para poder calcular los coeficientes  $\beta_0$  y  $\beta_1$  se usa la fórmula de la verosimilitud (o *likelihood function*) que se define como:

$$\ell(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i':y'_i=0} (1 - p(x_{i'}))$$

Donde se busca que los valores estimados  $\hat{\beta}_0$  y  $\hat{\beta}_1$  maximicen la fórmula de la verosimilitud (James, Witten, Hastie, Tibshirani, & Taylor, 2023).

Por defecto, la Regresión Logística no funciona bien para problemas de clasificación con clases desequilibradas. Para ello, se puede modificar el modelo para tomar en cuenta el sesgo en la distribución de los datos especificando un peso a las clases para poder influenciar la cantidad de actualización que sufren los coeficientes de la regresión durante el entrenamiento (Brownlee, 2020).

### **2.3.2. Support Vector Machines (SVM) o Máquinas de Soporte Vectorial**

“Son un conjunto de métodos de aprendizaje supervisado usados para clasificación, regresión y detección de anomalías” (Scikit-Learn, 2023). Estos modelos amplian el espacio de características usando kernels, que son funciones que permiten cuantificar la similitud entre dos observaciones, para poder ajustar un límite no lineal entre las clases (James, Witten, Hastie, Tibshirani, & Taylor, 2023). Este modelo funciona originalmente para problemas de clasificación binaria.

Las ventajas de las Máquinas de Soporte Vectorial según Scikit Learn son (Scikit-Learn, 2023):

- Son efectivas para espacios de altas dimensiones.
- Efectivas en casos donde el número de dimensiones es mayor que el número de muestras.

- Usa un subconjunto de puntos de entrenamiento para la función de decisión (llamados vectores soporte), lo que igual lo hace eficiente en memoria.

- Es versátil, pues puede usar distintas funciones de decisión a partir de Kernels.

Sus desventajas son:

- Si el número de variables es mayor que el número de muestras, seleccionar un Kernel y un término de regularización es crucial para evitar sobre-ajuste u overfitting.
- No provee directamente las probabilidades estimadas.

La implementación de MLlib es conocida como Linear Support Vector Classifier o Clasificador Lineal de Soporte Vectorial. Este clasificador crea un hiperplano de separación entre clases que maximice la distancia más cercana a las observaciones de entrenamiento, permitiendo que ciertas observaciones puedan clasificarse de forma errónea. Por eso, se dice que este hiperplano o margen es *suave* o *soft*. (James, Witten, Hastie, Tibshirani, & Taylor, 2023).

### **Representación matemática del clasificador lineal de soporte vectorial**

Resulta que la solución al problema del clasificador de soporte vectorial involucra solamente el producto interno entre sus observaciones (James, Witten, Hastie, Tibshirani, & Taylor, 2023). El producto interno de dos  $r$ -vectores  $a$  y  $b$  se define como  $\langle a, b \rangle = \sum_i^r a_i b_i$ . Por lo que el producto interno entre dos observaciones  $x_i, x_{i'}$  se representa como  $\langle x_i, x_{i'} \rangle = \sum_{j=1}^p x_{ij} x_{i'j}$

Donde  $p$  es el número de variables o características. Por lo que se puede demostrar que el clasificador lineal de soporte vectorial se puede representar como:  $f(x) = \beta_0 + \sum_{i=1}^n \alpha_i \langle x, x_i \rangle$

Donde hay  $n$  parámetros  $\alpha_i, i = 1, \dots, n$ , uno para cada observación de entrenamiento. Pero resulta que  $\alpha_i$  es no-cero solamente en los vectores soporte. Así que si  $S$  es la colección de índices de estos puntos soporte, la función anterior se puede reescribir como:  $f(x) = \beta_0 + \sum_{i \in S} \alpha_i \langle x, x_i \rangle$

Que típicamente involucra menos términos que la función original. Generalizando el producto interno como una función Kernel, esta se representa como:  $K(x_i, x_{i'})$

Que para el clasificador lineal de soporte vectorial sería equivalente a:

$$K(x_i, x_{i'}) = \sum_{j=1}^p x_{ij} x_{i'j}$$

Esta función se le conoce como *kernel lineal*, la cual calcula la similitud entre dos observaciones usando la correlación estándar de Pearson. (James, Witten, Hastie, Tibshirani, & Taylor, 2023)

### 2.3.3. Naive Bayes

Este clasificador tiene su origen en el teorema de Bayes, la cuál provee una expresión para la probabilidad posterior  $p_k(x) = \Pr(Y = k | X = x)$ . Este clasificador supone de forma “ingenua” (o *naive*, en inglés) que **los  $p$  predictores son independientes dentro de la  $k$ -ésima clase** (James, Witten, Hastie, Tibshirani, & Taylor, 2023). El teorema de Bayes se define como:

$$\Pr(Y = k | X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^K \pi_l f_l(x)}$$

Que representa la probabilidad *posterior* que una observación  $X = x$  pertenezca a la  $k$ -ésima clase, o dicho de otra forma, la probabilidad de que la observación pertenezca a la  $k$ -ésima clase, dado el valor del predictor de esa observación.  $\pi_k$  represenla la probabilidad *priori* de que una observación aleatoria provenga de la  $k$ -ésima clase, y  $f_k(X) \equiv \Pr(X|Y = k)$  representa la función de densidad de  $X$  para una observación que proviene de la  $k$ -ésima clase.

La suposición del modelo (los  $p$  predictores son independientes dentro de la  $k$ -ésima clase) se expresa matemáticamente como:  $f_k(x) = f_{k1}(x_1) \times f_{k2}(x_2) \times \dots f_{kp}(x_p)$

Donde  $f_{kj}$  es la función de densidad del  $j$ -ésimo predictor entre las observaciones de la  $k$ -ésima clase. Esta suposición disminuye la varianza introduciendo algo de sesgo, que en la práctica funciona bien. Esta suposición también elimina la necesidad de preocuparse por la asociación entre los  $p$  predictores porque se asume que no existe ningún tipo de asociación entre ellos.

Finalmente, la función del teorema de Bayes se pueden describir como:

$$\Pr(Y = k | X = x) = \frac{\pi_k \times f_{k1}(x_1) \times f_{k2}(x_2) \times \dots f_{kp}(x_p)}{\sum_{l=1}^K \pi_l \times f_{l1}(x_1) \times f_{l2}(x_2) \times \dots f_{lp}(x_p)} \text{ Para } k = 1, \dots, K.$$

Para el caso de datasets con clases imbalanceadas, existe el algoritmo *Complement naive Bayes* (CNB). Este algoritmo es una variante del algoritmo estándar multinomial naive Bayes (MNB), y se caracteriza por usar estadísticos del *complemento* de cada clase para calcular los pesos del modelo. Los inventores de este algoritmo demostraron empíricamente que los parámetros estimados por CNB son más estables que aquellos calculados con MNB (Scikit-Learn, 2023).

### 2.3.4. Isolation Forest

Este algoritmo fue propuesto por Liu, Ting y Zhou en 2008, y se especializa en la detección de anomalías. La base sobre la que se fundamenta este algoritmo es que las anomalías son fáciles de separar o aislar del resto de las observaciones. Al estar basado en bosques aleatorios o Random Forest, este algoritmo asume que el particionamiento de las instancias genera un camino mucho más corto para obtener las observaciones que son anomalías (Liu, Ting, & Zhou, 2008).

## 2.4. Técnicas de evaluación de modelos de ML

### 2.4.1. Matriz de confusión

Esta “es una herramienta que permite visualizar el desempeño de un algoritmo de aprendizaje supervisado” (Barrios Arce, 2019). En esta matriz se pueden comparar los valores reales vs. los valores predichos por el modelo:

*Figura 1 – Definición de la matriz de confusión binaria*

	VALORES PREDICCIÓN	
Verdaderos positivos		Falsos Positivos
Falsos Negativos		Verdaderos Negativos

VALORES REALES

*Fuente: (Barrios Arce, 2019)*

Los 4 posibles valores de la matriz son:

- Verdaderos positivos o True Positives ( $tp$ ): Es el número de valores positivos clasificados como positivos.
- Falsos positivos o False Positives ( $fp$ ): Es el número de valores negativos clasificados como positivos. Según Barrios Arce (Barrios Arce, 2019), en estadística a este valor se le conoce como error tipo I.

- Falsos negativos o False Negatives ( $fn$ ): Es el número de valores positivos clasificados como negativos. Según Barrios Arce (Barrios Arce, 2019), en estadística a este valor se le conoce como error tipo II.
- Verdaderos negativos o True Negatives ( $tn$ ): Es el número de valores negativos clasificados como negativos.

#### **2.4.2. Métricas de rendimiento en problemas de clasificación binaria**

Para evaluar los modelos de clasificación binaria existen distintas métricas según Scikit-Learn (Scikit-Learn, 2021), siendo las más comunes:

La precisión, que mide la proporción de valores clasificados de forma correcta respecto al número de predicciones realizadas:  $precision = \frac{tp}{tp+fp}$

El recall o sensibilidad, que mide la proporción de valores clasificados como positivos respecto al número de observaciones positivas existentes:  $recall = \frac{tp}{tp+fn}$

Y el F1-score, que se interpreta como “la media armónica de la precisión y el recall, donde un F1 score alcanza su mejor en 1 y su peor valor en 0. La contribución de la precisión y el recall al F1 score es equitativa” (Scikit-Learn, 2021):  $F_1 = 2 \frac{precision \times recall}{precision + recall}$

#### **2.4.3. Característica operativa del receptor o Receiver operating characteristic (ROC)**

Según Scikit-Learn (Scikit-Learn, 2021), la curva ROC es una representación gráfica que ilustra el rendimiento de un clasificador binario a partir de la fracción de verdaderos positivos

(TPR o True Positive Rate) contra la fracción de falsos positivos (FPR o False Positive Rate) con distintos umbrales.

## 2.5. Técnicas de validación y optimización de modelos de aprendizaje supervisado

### 2.5.1. *k*-fold Cross Validation o Validación Cruzada de *k*-pliegues

Esta técnica de entrenamiento permite disminuir el error de prueba de un modelo de aprendizaje supervisado. Consiste en dividir de forma aleatoria el conjunto de observaciones en  $k$  grupos o pliegues de aproximadamente el mismo tamaño. Cada  $k$  pliegue es usado como un conjunto de validación del modelo, y el resto de  $k - 1$  pliegues se usan para entrenar el modelo. Este proceso se repite de forma iterativa  $k$  veces, calculando en cada iteración el error medio cuadrático o *Mean Squared Error* (MSE) sobre el conjunto de datos de validación. Este procedimiento resulta entonces en  $k$  estimaciones del error de prueba  $MSE_1, MSE_2, \dots, MSE_k$ , y con estos valores se estima finalmente el promedio del error de prueba de la validación cruzada por medio de la fórmula (James, Witten, Hastie, Tibshirani, & Taylor, 2023):  $CV_{(k)} =$

$$\frac{1}{k} \sum_{i=1}^k MSE_i$$

Elegir el número de  $k$  conlleva también un compromiso entre la varianza y el sesgo. Un valor muy bajo de  $k$  aumenta el sesgo derivado de la cantidad de datos que se usan para probar el modelo. Sin embargo, un valor muy alto de  $k$  aumenta la varianza debido a que el modelo se entrena con conjuntos de datos que son muy parecidos entre sí. Según (James, Witten, Hastie, Tibshirani, & Taylor (James, Witten, Hastie, Tibshirani, & Taylor, 2023), “generalmente se lleva a

cabo validación cruzada de  $k$  pliegues usando  $k = 5$  o  $k = 10$ , ya que estos valores han demostrado de forma empírica obtener estimados del error de prueba que no sufren de excesivo sesgo ni de alta varianza”.

### **2.5.2. Búsqueda de los mejores hiperparámetros con Búsqueda en Cuadrícula o Grid Search**

Aunado al proceso de validación cruzada de  $k$  pliegues, dicho proceso también puede ser usado para encontrar los mejores hiperparámetros del modelo que también contribuyan a disminuir el error de prueba.

Los hiperparámetros son parámetros de los modelos que no son ajustados dentro del proceso de aprendizaje. Estos valores son pasados directamente a los modelos, por ejemplo, en la Regresión Logística existe el hiperparámetro de regularización para controlar el ajuste que sufren los coeficientes. Para poder encontrar los mejores valores de los hiperparámetros generalmente se crea una cuadrícula o *grid* donde se establecen distintos valores para los distintos hiperparámetros, y cada posible combinación de valores se prueba como parte del proceso de validación cruzada de  $k$  pliegues. A este proceso se le conoce como Búsqueda en Cuadrícula o Grid Search.

### **2.5.3. Bootstrap**

Esta es una herramienta estadística ampliamente usada para cuantificar la incertidumbre asociada con un estimador o un modelo. Consiste en emular el proceso de obtener nuevos conjuntos de muestra a partir de muestrear de forma iterativa observaciones del conjunto de datos original. Este muestreo se lleva a cabo con reemplazo, lo que significa que las observaciones

pueden aparecer más de una vez en los distintos conjuntos de datos creados (James, Witten, Hastie, Tibshirani, & Taylor, 2023).

#### **2.5.4. Selección de características con la Prueba Chi-Cuadrada ( $\chi^2$ )**

“Una prueba Chi-cuadrada se usa en estadística para probar la independencia entre dos eventos” (Gajawada, 2019). La prueba Chi-cuadrada se usa también para seleccionar las características que son más dependientes de la variable de respuesta. Un valor bajo de Chi-cuadrada significa que una variable predictora y la variable de la respuesta pueden ser independientes entre sí, mientras un valor alto de Chi-cuadrada significa que una variable predictora y la variable de la respuesta pueden ser dependientes entre sí (Gajawada, 2019).

#### **2.5.5. Reducción de Dimensionalidad con PCA o Análisis de Componente Principales**

##### **2.5.5.1. Generalización de los métodos de reducción de dimensionalidad**

En general, los métodos de reducción de dimensionalidad transforman las variables originales del conjunto de datos y luego ajustan un modelo de mínimos cuadrados con las variables transformadas.

Definiendo  $Z_1, Z_2, \dots, Z_M$  como las  $M < p$  combinaciones líneas de las  $p$  variables originales. Esto es:  $Z_m = \sum_{j=1}^p \phi_{jm} X_j$

Donde  $\phi_{1m}, \phi_{2m}, \dots, \phi_{pm}$  son distintas constantes, y  $m = 1, \dots, M$ . Se puede entonces ajustar un modelo de regresión lineal con mínimos cuadrados:  $y_i = \theta_0 + \sum_{m=1}^M \theta_m z_{im} + \epsilon_i$ ,  $i = 1, \dots, n$ .

Donde  $\theta_0, \theta_1, \dots, \theta_M$  representan los coeficientes de regresión. Si las constantes  $\phi_{1m}, \phi_{2m}, \dots, \phi_{pm}$  se eligen de forma sabia, entonces este nuevo modelo de regresión lineal con mínimos cuadrados puede mejorar el resultado del modelo de regresión lineal con mínimos cuadrados usando las variables originales.

El término *reducción de dimensionalidad* proviene del hecho que este enfoque reduce el problema de estimar los  $p + 1$  coeficientes  $\beta_0, \beta_1, \dots, \beta_p$  del modelo de regresión lineal original a estimar los  $M + 1$  coeficientes  $\theta_0, \theta_1, \dots, \theta_M$ , donde  $M < p$ . En otras, palabras, la dimensión del problema se redujo de  $p + 1$  a  $M + 1$  (James, Witten, Hastie, Tibshirani, & Taylor, 2023).

### 2.5.5.2. Análisis de Componentes Principales o PCA

El Análisis de Componentes Principales (Principal Component Analysis o PCA) es una técnica de aprendizaje no supervisado que calcula las  $M$  componentes principales  $Z_1, Z_2, \dots, Z_M$  para poder obtener una representación de un conjunto de datos de muchas dimensiones en un espacio de menor número de dimensiones manteniendo la mayor cantidad de información o varianza del conjunto de datos original. La idea detrás de PCA recae en que las  $n$  observaciones se encuentran en un espacio  $p$ -dimensional, pero no todas las dimensiones son igual de importantes, midiendo la importancia en términos de la variación de las observaciones sobre cada dimensión (James, Witten, Hastie, Tibshirani, & Taylor, 2023).

## 2.6. Herramientas y librerías

Para llevar a cabo el modelado se usa la librería MLlib de Spark. Esta librería aprovecha el funcionamiento distribuido de Spark para entrenar modelos escalables de forma fácil.

Esta librería provee de todas las herramientas necesarias para facilitar el entrenamiento de modelos de Machine Learning como (Apache Spark, 2021):

- Incluye algoritmos de ML comunes para tareas de clasificación, regresión, clustering y filtro colaborativo.
- Herramientas para la extracción y transformación de características, reducción de dimensionalidad.
- Creación de pipelines para facilitar el entrenamiento con distintas etapas.
- Entre otros.

Para poder crear un modelo con Isolation Forest se usa la implementación creada por (Verbus, 2019) del Anti-Abuse AI Team de LinkedIn. Este algoritmo está implementado como un algoritmo de aprendizaje no supervisado en Scala y se integra de forma sencilla con MLlib de Spark para poder usar el resto de sus características.

## 2.7. Particionamiento en Spark

### 2.7.1. Data Skew o sesgo de datos

El sesgo o Data Skew es una medida estadística que se usa para referirse a la asimetría de una distribución de datos. Indica la tendencia de los datos a inclinarse hacia un lado u otro de la distribución (Yadav, 2021).

En Spark, este concepto aparece cuando los datos no se partitionan de forma balanceada entre los nodos Worker del clúster, como resultado de operaciones que requieren reparticionar los datos (Las Transformaciones Anchas o Wide Transformations) como lo son las operaciones de Join y agrupamiento (GroupBy) (Berk, 2022).

El impacto que esto puede generar sobre el rendimiento del clúster es de suma importancia:

- Ciertas operaciones tomarán más tiempo en terminar de ejecutarse pues habrá nodos Worker que estén trabajando con muchos datos.
- Si los datos no caben en la memoria de algún nodo Worker comenzarán a escribirse en disco, lo que puede generar demoras significativas en las operaciones.
- Un nodo Worker puede quedarse sin memoria ni almacenamiento, lo que podría ocasionar errores de tipo **Out of Memory**.

Existen distintas técnicas empíricas que permiten diagnosticar y solucionar el problema del sesgo de datos en PySpark, como por ejemplo, a través de la visualización del conteo de datos dentro de las particiones. En este caso, la función `spark_partition_id` puede usarse para agrupar los registros por el ID de la partición a la que pertenecen para posteriormente diagnosticar si existe sesgo de forma visual. Un código de ejemplo del uso de esta función se presenta a continuación:

*Figura 2 – Diagnosticar el sesgo visualizando el conteo de datos dentro de las particiones*

```
from pyspark.sql.functions import sum

prod_transactions.groupBy(spark_partition_id()).count().agg(sum("count")).display()
```

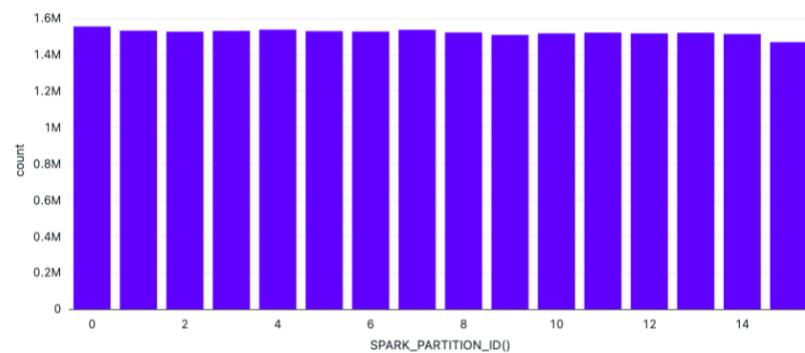
En el ejemplo, el contenido del DataFrame de PySpark `prod_transactions` se agrupa usando el ID de las particiones para obtener el conteo de registros dentro de cada partición.

**Figura 3 – Conteo de registros por partición**

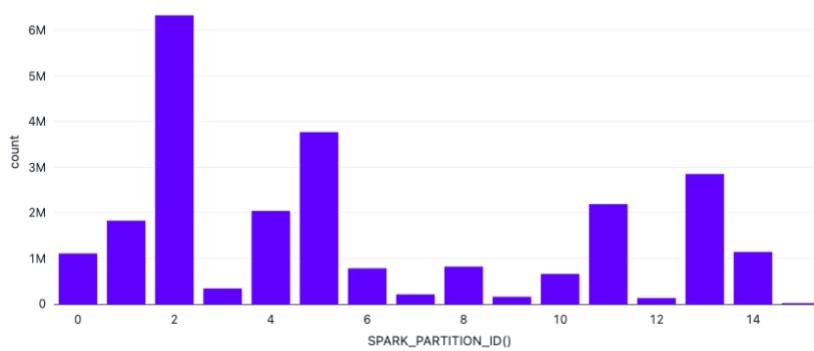
	SPARK_PARTITION_ID()	count
2	1	1533369
3	2	1527957
4	3	1532153
5	4	1539113
6	5	1530598
7	6	1528522
8	7	1529112

También se puede usar para crear una visualización que nos permita diagnosticar de forma sencilla si existe sesgo o no en la distribución de las particiones. A continuación, se muestran dos ejemplos, uno con sesgo y otro sin sesgo.

**Figura 4 – Conjunto de datos sin sesgo**



**Figura 5 – Conjunto de datos con sesgo.**



Existen 2 técnicas muy sencillas para poder solventar este problema que involucran reparticionar el conjunto de datos:

- **Reparticionar el conjunto de datos usando una columna:** Reparticionar el conjunto de datos usando la columna o columnas usadas para las operaciones de agrupamiento/joining para mejorar la eficiencia de las operaciones de shuffling o movimiento de datos entre particiones.
- **Crear una columna salt para reparticionar el conjunto de datos:** Esta es una técnica muy efectiva para lograr una distribución uniforme de los datos en distintas particiones. Consiste en crear una columna **salt**, la cual contiene números aleatorios de una distribución uniforme obtenidas con la función **rand**, y usar esa columna para reparticionar el conjunto de datos.

Un código de ejemplo se presenta a continuación:

**Figura 6 – Columna de salt para particionar los datos**

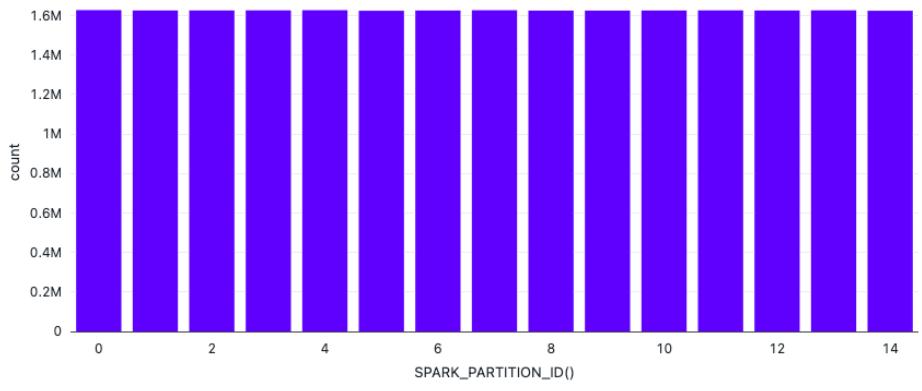
```
# Usar una columna salt para reparticionar los datos.
from pyspark.sql.functions import rand

prod_transactions = prod_transactions.withColumn("salt", rand()).repartition(15, "salt")

new_transactions.groupBy(spark_partition_id()).count().display()
```

Como se mencionó anteriormente, la función **rand** permite crear la columna **salt**. El resultado del reparticionamiento se visualiza en la siguiente imagen (Berk, 2022):

**Figura 7 – Visualización del resultado del reparticionamiento**



### 2.7.2. Sugerencias para elegir el número de particiones

Elegir el número de particiones adecuado es un tema empírico que en la literatura es difícil de encontrar. Existen algunas sugerencias que se pueden seguir para elegir dicho valor (ProjectPro, 2023):

- El número de particiones siempre se basa de acuerdo con la configuración del clúster y los requerimientos de la aplicación.
- Existe un compromiso entre tener un número muy bajo de particiones vs. Tener un número muy alto de particiones:
  - Un número muy bajo de particiones puede subutilizar los recursos del clúster, incluso dejando que algunos nodos Worker no se utilicen en lo absoluto.
  - Un número muy alto de particiones puede crear particiones con pocos datos o incluso particiones vacías. De igual forma, el tiempo de programación de las tareas puede ser mayor que el tiempo de ejecución de las mismas.
  - Siempre se sugiere iniciar con un número de particiones que sea igual al número de núcleos disponibles en el clúster para los nodos Worker. El número de particiones

puede variar en x2 o x3 veces dicho número, pero manteniendo un valor proporcional al número de núcleos.

- Como límite superior, se puede tomar como referencia que las tareas no deben exceder un tiempo de ejecución mayor a 100ms. Un tiempo de ejecución mayor puede ser signo de un tiempo de programación muy alto.

Este tema de particionamiento resulta de suma importancia para la presente tesis, pues las distintas etapas del proyecto se desarrollan usando distintos clústers de distintos tamaños. Más adelante se explicará con mayor detalle cada uno de los clústers y su propósito.

### 3. Metodología

#### 3.1. Datos utilizados

Para llevar a cabo el modelo, se utiliza principalmente un conjunto de datos (dataset) de Kaggle con transacciones realizadas con tarjetas de crédito. Adicionalmente, se enriquece el conjunto de datos con información obtenida de una fuente externa. Esta información adicional incluye el nombre completo del MCC (Merchant Category Code), en lugar de solo su código numérico. A continuación, se pueden encontrar las fichas técnicas de los datasets utilizados.

**Tabla 1 - Ficha técnica del dataset “Credit Card Transaction”.**

<b>Nombre del dataset</b>	Credit Card Transaction (en español: Transacciones con tarjeta de crédito).
<b>Descripción</b>	Transacciones generadas a partir de una simulación realizada por IBM. Los datos abarcan 2000 consumidores (sintéticos) residentes en Estados Unidos, pero que viajan por todo el mundo. Los datos abarcan también décadas de compras e incluyen varias tarjetas de muchos de los consumidores (IBM, 2021).
<b>Fuente</b>	Kaggle (el dataset se puede descargar a través del siguiente enlace: <a href="https://www.kaggle.com/datasets/ealtman2019/credit-card-transactions">https://www.kaggle.com/datasets/ealtman2019/credit-card-transactions</a> ).
<b>Formato</b>	Cuatro (4) archivos CSV: <ul style="list-style-type: none"> <li>• credit_card_transactions-ibm_v2.csv</li> <li>• sd254_cards.csv</li> <li>• sd254_users.csv</li> <li>• User0_credit_card_transactions.csv</li> </ul>
<b>Tamaño</b>	Aproximadamente 20 millones de registros para transacciones.
<b>Licencia</b>	Apache License Version 2.0 (open-source que permite usar, reproducir y distribuir el conjunto de datos, incluyendo fines educativos e investigativos).
<b>Calidad de los datos</b>	Usabilidad del 8.53/10. Siendo su única anotación que este dataset no se actualiza con frecuencia. Sin embargo, esto no debe ser un impedimento para el caso de estudio con fines educativos.

Para el objetivo del estudio, se excluye el archivo User0\_credit\_card\_transactions.csv, pues este solo contiene una muestra de la información para el usuario con ID número 0.

**Tabla 2 - Ficha técnica del dataset “MCC Codes”.**

<b>Nombre del dataset</b>	MCC Codes
<b>Descripción</b>	Repositorio público de MCC (Código de Categoría Mercante) en diferentes formatos de fácil lectura (Knaddison, Martins Pereira, & Lewis, 2020).
<b>Fuente</b>	GitHub (el dataset se puede descargar a través del siguiente enlace: <a href="https://github.com/greggles/mcc-codes/blob/main/mcc_codes.csv">https://github.com/greggles/mcc-codes/blob/main/mcc_codes.csv</a> ).
<b>Formato</b>	CSV
<b>Tamaño</b>	982 registros / 92.2 KB.
<b>Licencia</b>	Unlicense

### 3.1.1. Transacciones (credit\_card\_transactions-ibm\_v2.csv)

Este CSV contiene toda la información de transacciones realizadas con tarjetas de crédito que contiene el dataset.

**Figura 8 - Ejemplo del contenido del archivo de transacciones (transactions) del dataset.**

```
[2]: 1 df = pd.read_csv('credit_card_transactions-ibm_v2.csv')
2 df.head()

[2]:   User  Card  Year  Month  Day  Time  Amount      Use Chip  Merchant Name  Merchant City  Merchant State  Zip  MCC  Errors?  Is Fraud?
  0    0     0  2002       9     1  06:21  $134.09  Swipe Transaction  3527213246127876953  La Verne          CA  91750.0  5300    NaN    No
  1    0     0  2002       9     1  06:42   $38.48  Swipe Transaction -727612092139916043  Monterey Park        CA  91754.0  5411    NaN    No
  2    0     0  2002       9     2  06:22  $120.34  Swipe Transaction -727612092139916043  Monterey Park        CA  91754.0  5411    NaN    No
  3    0     0  2002       9     2  17:45  $128.95  Swipe Transaction  3414527459579106770  Monterey Park        CA  91754.0  5651    NaN    No
  4    0     0  2002       9     3  06:23  $104.71  Swipe Transaction  5817218446178736267  La Verne          CA  91750.0  5912    NaN    No
```

**Tabla 3 - Columnas del archivo de transacciones (transactions) del x dataset.**

User	ID del usuario que realiza la transacción. Corresponde a un registro del archivo de “users”.
------	--

<b>Card</b>	ID de la tarjeta de crédito utilizada en la transacción. Corresponde a un registro del archivo de “cards”.
<b>Year</b>	Año en que se realizó la transacción.
<b>Month</b>	Mes del año en que se realizó la transacción.
<b>Day</b>	Día del mes en que se realizó la transacción.
<b>Time</b>	Hora de la transacción.
<b>Amount</b>	Valor de la transacción.
<b>Use Chip</b>	Tipo de transacción: Chip Transaction, Online Transaction o Swipe Transaction.
<b>Merchant Name</b>	Nombre del comerciante al que se le realizó la transacción. Se muestra como una especie de ID con números.
<b>Merchant City</b>	Ciudad donde se encuentra registrado el comerciante.
<b>Merchant State</b>	Estado donde se encuentra registrado el comerciante.
<b>Zip</b>	Código postal de donde se encuentra registrado el comerciante.
<b>MCC</b>	Merchant Category Code (en español: Código de Categoría del Comerciante), “define los valores de código utilizados para permitir la clasificación de los comerciantes en categorías específicas basadas en el tipo de negocio, comercio o servicios suministrados” (ISO, 2003).
<b>Errors?</b>	Si la transacción tuvo algún error, se muestra el tipo de error.
<b>Is Fraud?</b>	Si la transacción es fraudulenta o no (Yes / No).

### 3.1.2. Tarjetas (sd254\_cards.csv)

Este CSV contiene toda la información de las tarjetas de crédito por usuario. Se asocia una tarjeta de crédito a cada transacción, pero un usuario puede tener más de una tarjeta.

*Figura 9 - Ejemplo del contenido del archivo de tarjetas (cards) del dataset.*

[3]:	1 df1 = pd.read_csv('sd254_cards.csv')
[3]:	2 df1.head()
	User CARD INDEX Card Brand Card Type Card Number Expires CVV Has Chip Cards Issued Credit Limit Acct Open Date Year PIN last Changed Card on Dark Web
0	0 0 Visa Debit 4344676511950444 12/2022 623 YES 2 \$24295 09/2002 2008 No
1	0 1 Visa Debit 4956965974959986 12/2020 393 YES 2 \$21968 04/2014 2014 No
2	0 2 Visa Debit 4582313478255491 02/2024 719 YES 2 \$46414 07/2003 2004 No
3	0 3 Visa Credit 4879494103069057 08/2024 693 NO 1 \$12400 01/2003 2012 No
4	0 4 Mastercard Debit (Prepaid) 5722874738736011 03/2009 75 YES 1 \$28 09/2008 2009 No

**Tabla 4 - Columnas del archivo de tarjetas (cards) del dataset.**

User	ID del usuario propietario de la tarjeta
CARD INDEX	ID de la tarjeta.
Card Brand	Marca de la tarjeta.
Card Type	Tipo de tarjeta (Debit, Credit, Debit (Prepaid)).
Card Number	Número de la tarjeta.
Expires	Fecha de vencimiento de la tarjeta.
CVV	Código de seguridad de la tarjeta.
Has Chip	Si la tarjeta tiene chip (YES / NO).
Cards Issued	Número de veces que la tarjeta ha sido emitida.
Credit Limit	Límite de crédito.
Acct Open Date	Mes y año en el que se abrió la cuenta bancaria asociada a la tarjeta.
Year PIN last Changed	Año en que el PIN se cambió por última vez.
Card on Dark Web	Si la tarjeta de encuentra en la dark web (Yes / No).

### 3.1.3. Usuarios (sd254\_users.csv)

Este CSV contiene de los usuarios considerados en la lista de transacciones y sus respectivos datos.

**Figura 10 - Ejemplo del contenido del archivo de usuarios (users) del dataset.**

[4]:	Person	Current Age	Retirement Age	Birth Year	Birth Month	Gender	Address	Apartment	City	State	Zipcode	Latitude	Longitude	Per Capita Income - Zipcode	Yearly Income - Person	Total Debt	FICO Score	Num Credit Cards	
0	Hazel Robinson	53	66	1966	11	Female	462 Rose Lane		NaN	La Verne	CA	91750	34.15	-117.76	\$29278	\$59696	\$127613	787	5
1	Sasha Sadr	53	68	1966	12	Female	3506 Federal Boulevard		NaN	Little Neck	NY	11363	40.76	-73.74	\$37891	\$777254	\$191349	701	5
2	Saami Lee	81	67	1938	11	Female	766 Third Drive		NaN	West Covina	CA	91792	34.02	-117.89	\$22681	\$33483	\$196	698	5
3	Everlee Clark	63	63	1957	1	Female	3 Madison Street		NaN	New York	NY	10069	40.71	-73.99	\$163145	\$249925	\$202328	722	4
4	Kyle Peterson	43	70	1976	9	Male	9620 Valley Stream Drive		NaN	San Francisco	CA	94117	37.76	-122.44	\$53797	\$109687	\$183855	675	1

**Tabla 5 - Columnas del archivo de usuarios (users) del dataset.**

<b>Person</b>	Nombre del usuario.
<b>Current Age</b>	Edad actual del usuario.
<b>Retirement Age</b>	Edad de retiro.
<b>Birth Year</b>	Año de nacimiento.
<b>Birth Month</b>	Mes de nacimiento.
<b>Gender</b>	Género del usuario.
<b>Address</b>	Dirección de residencia del usuario.
<b>Apartment</b>	Número de apartamento, si aplica.
<b>City</b>	Ciudad de residencia.
<b>State</b>	Estado de residencia.
<b>Zipcode</b>	Código postal de la residencia del usuario.
<b>Latitude</b>	Latitud.
<b>Longitude</b>	Longitud.
<b>Per Capita Income – Zipcode</b>	Ingresos por persona en el área.
<b>Yearly Income – Person</b>	Ingresos anuales del usuario.
<b>Total Debt</b>	Deuda total.
<b>FICO Score</b>	“Los puntajes FICO son el estándar para los puntajes de crédito, utilizados por el 90% de los principales prestamistas”. Este “informa a los prestamistas sobre su solvencia crediticia (qué tan probable es que pague un préstamo en función de su historial de crédito)” (myFICO, 2023).
<b>Num Credit Cards</b>	Cantidad de tarjetas que el usuario posee.

### 3.1.4. MCC Codes (mcc\_codes.csv)

Este CSV contiene información de los MCC (Código de Categoría Mercante), y se utiliza para mapear los códigos a los nombres de las categorías del dataset de transacciones.

**Figura 11 - Ejemplo del contenido del archivo de MCC Codes.**

mcc	edited_description	combined_description	usda_description	irs_description	irs_reportable
742	Veterinary Services	Veterinary Services	Veterinary Services	Veterinary Services	Yes
763	Agricultural Co-operatives	Agricultural Co-operatives	Agricultural Co-operatives	Agricultural Cooperative	Yes
780	Horticultural Services, Landscaping Services	Horticultural Services, Landscaping Services	Horticultural Services	Landscaping Services	Yes
1520	General Contractors-Residential and Commercial	General Contractors-Residential and Commercial	General Contractors-Residential and Commercial	General Contractors	Yes
1711	Air Conditioning Contractors – Sales and Installation, Heating Contractors – Sales, Service, Installation	Air Conditioning Contractors – Sales and Installation, Heating Contractors – Sales, Service, Installation	Air Conditioning Contractors – Sales and Installation	Heating, Plumbing, A/C	Yes
1731	Electrical Contractors	Electrical Contractors	Electrical Contractors	Electrical Contractors	Yes
1740	Insulation – Contractors, Masonry, Stonework Contractors, Plastering Contractors, Stonework and Masonry Contractors, Tile Settings Contractors	Insulation – Contractors, Masonry, Stonework Contractors, Plastering Contractors, Stonework and Masonry Contractors, Tile Settings Contractors	Insulation – Contractors	Masonry, Stonework, and Plaster	Yes
1750	Carpentry Contractors	Carpentry Contractors	Carpentry Contractors	Carpentry Contractors	Yes
1761	Roofing – Contractors, Sheet Metal Work – Contractors, Siding – Contractors	Roofing – Contractors, Sheet Metal Work – Contractors, Siding – Contractors	Roofing - Contractors	Roofing/Siding, Sheet Metal	Yes
1771	Contractors – Concrete Work	Contractors – Concrete Work	Contractors – Concrete Work	Concrete Work Contractors	Yes
1799	Contractors – Special Trade, Not Elsewhere Classified	Contractors – Special Trade, Not Elsewhere Classified	Contractors – Special Trade, Not Elsewhere Classified	Special Trade Contractors	Yes
2741	Miscellaneous Publishing and Printing	Miscellaneous Publishing and Printing	Miscellaneous Publishing and Printing	Miscellaneous Publishing and Printing	Yes
2791	Typeetting, Plate Making, & Related Services	Typeetting, Plate Making, & Related Services	Typeetting, Plate Making, & Related Services	Typeetting, Plate Making, and Related Services	Yes
284	Specialty Cleaning, Polishing, and Sanitation Preparations	Specialty Cleaning, Polishing, and Sanitation Preparations	Specialty Cleaning, Polishing, and Sanitation Preparations	Specialty Cleaning	Yes
3000	UNITED AIRLINES	UNITED AIRLINES	UNITED AIRLINES	Airlines	Yes
3001	AMERICAN AIRLINES	AMERICAN AIRLINES	AMERICAN AIRLINES	Airlines	Yes
3002	PAN AMERICAN	PAN AMERICAN	PAN AMERICAN	Airlines	Yes
3003	Airlines	Airlines	null	Airlines	Yes
3004	TRANS WORLD AIRLINES	TRANS WORLD AIRLINES	TRANS WORLD AIRLINES	Airlines	Yes

**Tabla 6 - Columnas del archivo de MCC Codes.**

<b>mcc</b>	Código de Categoría Mercante (MCC).
<b>edited_description</b>	Texto moderno para describir el MCC.
<b>combined_description</b>	Utiliza la descripción del USDA o del IRS.
<b>usda_description</b>	Descripción del USDA (Departamento de Agricultura de los Estados Unidos).
<b>irs_description</b>	Descripción del IRS (Servicio Interno de Impuestos).
<b>irs_reportable</b>	Reportable bajo 6041/6041A y Autoridad para Excepciones.

## 3.2. Herramientas

### 3.2.1. Kaggle

Kaggle es una comunidad dedicada a la ciencia de datos, donde científicos de datos de todo el mundo pueden abordar problemas relacionados con este campo, así como participar en competiciones, compartir código y conjuntos de datos. También es un lugar donde los profesionales pueden colaborar y aprender unos de otros en el ámbito de la ciencia de datos, el aprendizaje automático y el análisis predictivo (Kaggle, 2017). Particularmente en este proyecto,

se utiliza Kaggle como la fuente de información de la que se descargará el dataset, utilizando su API de Python CLI (Kaggle, 2023).

### **3.2.2. Microsoft Azure Cloud**

Esta es la nube seleccionada para llevar a cabo el proyecto. “Azure es la plataforma pública en la nube de Microsoft. Azure ofrece una amplia gama de servicios, lo que incluye las funcionalidades de plataforma como servicio (PaaS), infraestructura como servicio (IaaS) y servicio de base de datos administrado” (Microsoft, 2023). En este caso, se utiliza para crear una instancia de Databricks gratuita, sobre la cual se implementa el análisis, procesamiento de datos y los modelos.

### **3.2.3. Databricks**

“Azure Databricks es una plataforma de análisis unificada y abierta para crear, implementar, compartir y mantener soluciones de datos, análisis e IA de nivel empresarial a escala”. Se utilizará para “procesar, almacenar, limpiar, compartir, analizar, modelar y monetizar sus conjuntos de datos con soluciones desde la BI al aprendizaje automático” (Microsoft, 2023).

La elección de utilizar Databricks se justifica por la disponibilidad de todas las herramientas esenciales para el procesamiento de datos en su entorno, lo que elimina la necesidad de crear una infraestructura de datos en la nube desde cero. Además, Databricks ofrece la capacidad de llevar a cabo un procesamiento distribuido para manejar grandes volúmenes de datos, como los que se encuentran en el conjunto de datos que se desea procesar. Algunas de las cosas que se pueden realizar en Databricks, según su sitio web oficial (Microsoft, 2023) son:

- Programación y administración de flujos de procesamiento de datos.
- SQL.
- Creación de visualizaciones.
- Ingesta de datos.
- Exploración de datos.
- Administración de procesos.
- Modelado y seguimiento de Machine Learning (ML).

Databricks admite diferentes lenguajes de programación: Python, Scala, R y SQL.

### **3.2.4. Python / PySpark**

Databricks es una plataforma de trabajo que admite la ejecución de varios lenguajes de programación, como se mencionó anteriormente. En este contexto, se utilizará Python, específicamente en su variante PySpark. Como lo menciona Apache Spark en su documentación (Apache Spark, 2023), PySpark es una interfaz para Apache Spark en Python, que permite un análisis interactivo de los datos en un entorno distribuido. Admite la mayoría de las características de Spark, como lo pueden ser SQL, DataFrame, Streaming, MLlib, Spark Core, entre otras.

### **3.2.5. Scala**

“Scala es un lenguaje de programación hermoso y expresivo, con una sintaxis limpia y moderna, que soporta programación funcional (PF) y programación orientada a objetos (POO), y que proporciona un sistema de tipos estático seguro” (Scala, 2021). Las razones por las que usar Scala es una buena idea, residen en que “Scala es 10 veces más rápido que Python, produce un

código de menor tamaño que Java, ofrece capacidades de programación más robustas que C++ y combina las ventajas de dos grandes paradigmas de programación, lo que lo hace único frente a otros muchos lenguajes de programación” (Raval, 2023).

### 3.2.6. SQL

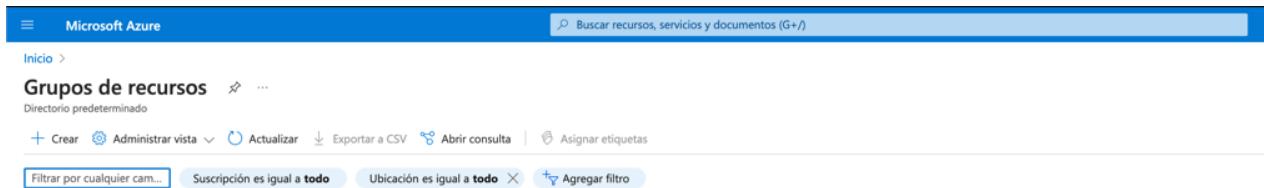
“SQL es un lenguaje estándar para almacenar, manipular y recuperar datos en bases de datos” (W3Schools, 2023). Ya que Databricks también permite SQL, se utilizará para hacer algunas consultas a las tablas que se creen.

## 3.3. Configuración del entorno de trabajo

### 3.3.1. Creación de Instancia de Azure Databricks

Para la creación de la instancia de Azure Databricks, se asume que se posee una cuenta válida y una suscripción de Azure. Posteriormente se procede a la creación de un grupo de recursos.

*Figura 12 - Creación de un grupo de recursos (1)*



**Figura 13 - Creación de un grupo de recursos (2)**

Microsoft Azure

Buscar recursos, servicios y documentos (G+/-)

Inicio > Grupos de recursos >

Crear un grupo de recursos ...

Datos básicos Etiquetas Revisar y crear

Grupo de recursos - Contenedor que incluye los recursos relacionados para una solución de Azure. El grupo de recursos puede contener todos los recursos de la solución o solamente los recursos que quiere administrar en grupo. Debe decidir cómo quiere asignar los recursos a los grupos de recursos según lo que resulte más pertinente para su organización. [Más información](#)

Detalles del proyecto

Suscripción \* ⓘ Azure subscription 1 ✓

Grupo de recursos \* ⓘ IMF\_Smart\_Education ✓

Detalles del recurso

Región \* ⓘ (US) East US ✓

**Figura 14 - Creación de un grupo de recursos (3)**

Grupos de recursos	
Directorio predeterminado	
+ Crear	Administrador vista ⓘ
Suscripción es igual a todo	Ubicación es igual a todo ⓘ
Nombre ⓘ	Suscripción ⓘ
IMF_Smart_Education	Azure subscription 1
	Ubicación ⓘ
	East US

Posteriormente, se busca el servicio de Databricks en Azure y se crea una instancia de Databricks en Azure dentro del grupo de recursos. Para ello se proporcionaron los detalles como el nombre de la instancia, la región de implementación, otras opciones relevantes.

**Figura 15 – Creación de instancia de Databricks en Azure (1)**

Azure Databricks	
Directorio predeterminado	
+ Crear	Administrador vista ⓘ
Suscripción es igual a todo	Grupo de recursos es igual a todo ⓘ
Ubicación es igual a todo ⓘ	Agregar filtro
Filtrar por cualquier campo...	

*Figura 16 – Creación de instancia de Databricks en Azure (2)*

Microsoft Azure

Buscar recursos, servicios y documentos

Inicio > Azure Databricks > Creación de un área de trabajo de Azure Databricks

Datos básicos Redes Cifrado Etiquetas Revisar y crear

Detalles del proyecto

Seleccione la suscripción para administrar recursos implementados y los costes. Use los grupos de recursos como carpetas para organizar y administrar todos los recursos.

Suscripción \* ⓘ Azure subscription 1

Grupo de recursos \* ⓘ IMF\_Smart\_Education [Crear nuevo](#)

Detalles de instancia

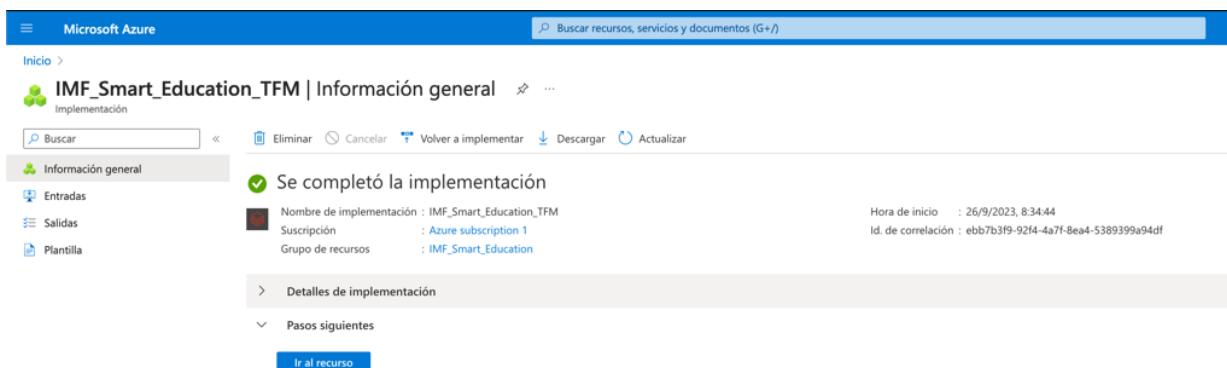
Nombre del área de trabajo \* TFM

Región \* East US

Plan de tarifa \* ⓘ Evaluación (Premium: DBU gratis durante 14 días)

Nombre del grupo de recursos administrados Enter name for managed resource group

*Figura 17 – Creación de instancia de Databricks en Azure (3)*



*Figura 18 – Creación de instancia de Databricks en Azure (4)*

The screenshot shows the Microsoft Azure portal interface for managing a Databricks service. The top navigation bar includes 'Microsoft Azure', a search bar, and several icons. The main content area is titled 'TFM | Información general' under 'Servicio de Azure Databricks'. It shows basic details like 'Estado: Active', 'Grupo de recursos: IMF\_Smart\_Education', 'Ubicación: East US', and 'Suscripción: Azure subscription 1'. A large red icon of three stacked cubes is prominently displayed. Below the main details are sections for 'Información esencial', 'Configuración', 'Automation', and 'Ayuda'. On the right, there are buttons for 'Iniciar área de trabajo' and 'Actualizar a Premium'. At the bottom, there are six cards: 'Documentación', 'Introducción', 'Importar datos de archivo', 'Importar datos de Azure Storage', 'Guía de administración', and 'Vincular área de trabajo de Azure ML'.

Una vez creada la instancia de Databricks, ya se puede acceder a ella

*Figura 19 – Acceso a la instancia de Databricks*

The screenshot shows the Databricks workspace interface. The left sidebar contains navigation links for 'Nuevo' (Workspace, Recientes, Catálogo, Flujos de trabajo, Cómputo), 'SQL' (Editor de SQL, Consultas, Tableros, Alertas, Historial, Warehouses SQL), 'Ingeniería de datos' (Ejecuciones, Ingesta de datos, Tablas Delta Live), 'Machine Learning' (Experimentos, Características, Modelos, Servicio). The main area is titled 'Empezar con Databricks' and includes sections for 'Estamos preparando el warehouse de SQL' (status: 00:51 Haciendo start en el warehouse), 'Explorar proyectos de muestra' (with a preview of a sample query and visualizations), and 'Importe sus datos' (with a 'Upload de datos' button and a link to 'Ver todas las fuentes de datos').

### 3.3.2. Creación de Clúster con Apache Spark en Databricks

Para llevar a cabo las distintas etapas del proyecto se crean 5 distintos clústeres de distintos tamaños. Esto con el objetivo de ahorrar dinero y tener el poder de cómputo adecuado para cada uno de los procesos que se llevan a cabo.

*Figura 20 – Clústers usados en la tesis*

Estado	Nombre	Directriz
■	ML Training	-
■	Feature Engineering	-
■	DataViz	-
■	Dimensionality Reduction	-
■	EDA	-

- **Clúster ML Training:** a continuación muestran sus características.

*Figura 21 – Resumen del clúster para ML*

1-5 workers	28-140 GB de memoria 4-20 núcleos
1 driver	28 GB de memoria y 4 núcleos
Runtime	13.3.x-cpu-ml-scala2.12

Standard\_DS12\_v2    2-6 DBU/h

**Figura 22 – Clúster para ML**

The screenshot shows the 'ML Training' configuration page in the Databricks interface. The top navigation bar includes 'Cómputo > Vista previa de la IU' and 'Enviar comentarios'. Below the navigation is a header with 'ML Training' and a blue 'Run' button. A horizontal menu bar includes 'Configuración' (which is underlined), 'Cuadernos (0)', 'Bibliotecas', 'Log de eventos', 'IU de Spark', 'Logs del driver', 'Métricas', and 'Aplicaciones'.

**Directriz**: Sin restricciones

**Modo de acceso**: Compartido sin aislamiento

**Rendimiento**

Versión de Databricks Runtime: 13.3 LTS ML (includes Apache Spark 3.4.1, Scala 2.12)

Utilizar aceleración Photon

**Tipo de worker**: Standard\_DS12\_v2 (28 GB de memoria y 4 núcleos) | Workers mín. 1 | Workers máx. 5 |  Instancias de spot

**Tipo de driver**: Standard\_DS12\_v2 (28 GB de memoria y 4 núcleos)

Habilitar la auto expansión

Terminar después de 10 minutos de inactividad

**Etiquetas**: Sin etiquetas personalizadas

> Etiquetas añadidas automáticamente

**Opciones avanzadas**

Transferencia de credenciales de Azure Data Lake Storage

Habilitar traspaso de credenciales para acceso a datos a nivel de usuario

**Spark** (selected), Registro, Scripts de inicio, JDBC/ODBC, Permisos

Configuración de Spark:

```
spark.databricks.delta.preview.enabled true
spark.driver.memory 15g
spark.driver.maxResultSize 15g
```

Variables de entorno

No hay variables de entorno

Este clúster está compuesto de 1 nodo Driver y 1 a 5 nodos Worker, teniendo un total de 6 computadoras. Cada una con 4 núcleos y 28 GB de RAM. Los nodos Worker escalan de forma automática cuando el procesamiento lo requiere.

En total, todos los nodos Worker suman 20 núcleos y 140 GB de RAM. Este clúster es el más grande de todos, y es el que se usa para el entrenamiento de los distintos modelos.

- **Clúster Feature Engineering:** a continuación se muestran sus características.

**Figura 23 - Resumen del clúster para Feature Engineering**



**Figura 24 – Cluster para Feature Engineering**

Cómputo > Vista previa de la IU Enviar comentarios

### Feature Engineering

- Configuración
- Cuadernos (0)
- Bibliotecas
- Log de eventos
- IU de Spark
- Logs del driver
- Métricas
- Aplicaciones

**Directriz** Sin restricciones

Multi-nodo  Nodo único

**Modo de acceso** Compartido sin aislamiento

**Rendimiento**

Versión de Databricks Runtime  
13.3 LTS ML (includes Apache Spark 3.4.1, Scala 2.12)

Utilizar aceleración Photon

**Tipo de worker** Standard\_DS3\_v2 14 GB de memoria y 4 núcleos 1 2  Instancias de spot

**Tipo de driver** Standard\_DS3\_v2 14 GB de memoria y 4 núcleos

Habilitar la auto expansión  
 Terminar después de 20 minutos de inactividad

**Etiquetas**

Sin etiquetas personalizadas  
Etiquetas añadidas automáticamente

▶ Opciones avanzadas

Este clúster está compuesto de 1 nodo Driver y 1 a 2 nodos Worker, teniendo un total de 3 computadoras. Cada una con 4 núcleos y 14 GB de RAM. Los nodos Worker escalan de forma automática cuando el procesamiento lo requiere. En total, todos los nodos Worker suman 8 núcleos y 28 GB de RAM. Este clúster es el que se usa para la preparación del dataset usado para el entrenamiento de los modelos.

- **Clúster DataViz:** Este clúster está compuesto solamente de 1 nodo con 16 núcleos y 56 GB de RAM.

*Figura 25 – Resumen del clúster para DataViz*



Este clúster se usa para crear visualizaciones con librerías como Matplotlib y Seaborn, pues estas librerías requieren que el DataFrame de PySpark se convierta en uno de Pandas, y esta operación requiere mover los datos de los Workers a la memoria del nodo Driver. Esta operación es más eficiente cuando los Workers y el Driver viven en la misma instancia.

**Figura 26 - Resumen del clúster para DataViz**

The screenshot shows the 'Configuración' tab of a Databricks cluster named 'DataViz'. The interface includes sections for 'Directriz', 'Modo de acceso', 'Rendimiento', 'Etiquetas', and 'Opciones avanzadas'.

- Directriz:** Sin restricciones
- Modo de acceso:** Compartido sin aislamiento
- Rendimiento:**
  - Versión de Databricks Runtime: 13.3 LTS ML (includes Apache Spark 3.4.1, Scala 2.12)
  - Utilizar aceleración Photon
  - Tipo de nodo: Standard\_DS5\_v2 (56 GB de memoria y 16 núcleos)
  - Terminar después de 20 minutos de inactividad
- Etiquetas:** Sin etiquetas personalizadas
- Opciones avanzadas:**
  - Transferencia de credenciales de Azure Data Lake Storage: Habilitar traspaso de credenciales para acceso a datos a nivel de usuario
  - Spark: Registro, Scripts de inicio, JDBC/ODBC, Permisos
  - Configuración de Spark:
    - spark.databricks.delta.preview.enabled true
    - spark.master local[\*, 4]
    - spark.databricks.cluster.profile singleNode
  - Variables de entorno: No hay variables de entorno

- **Clúster Dimensionality Reduction:** Este clúster está compuesto solamente de 1 nodo con 4 núcleos y 28 GB de RAM.

**Figura 27 – Resumen del clúster para Dimensionality Reduction**



**Figura 28 - Resumen del clúster para Dimensionality Reduction**

The screenshot shows the configuration page for a Dimensionality Reduction cluster. The top navigation bar includes links for Cómputo, Vista previa de la IU, Enviar comentarios, Configuration, Cuadernos (0), Bibliotecas, Log de eventos, IU de Spark, Logs del driver, and Help.

**Configuration**

- Directriz:** Sin restricciones
- Modo de acceso:** Compartido sin aislamiento

**Rendimiento**

- Versión de Databricks Runtime: 13.3 LTS ML (includes Apache Spark 3.4.1, Scala 2.12)
- Utilizar aceleración Photon
- Tipo de nodo:** Standard\_DS12\_v2, 28 GB de memoria y 4 núcleos
- Terminar después de 10 minutos de inactividad

**Etiquetas**

- Sin etiquetas personalizadas
- > Etiquetas añadidas automáticamente

**Opciones avanzadas**

**Transferencia de credenciales de Azure Data Lake Storage**

- Habilitar traspaso de credenciales para acceso a datos a nivel de usuario

**Spark**

- Configuración de Spark:**

```
spark.databricks.delta.preview.enabled true
spark.master local[*, 4]
spark.databricks.cluster.profile singleNode
```
- Variables de entorno:** No hay variables de entorno

Este clúster se usa para ejecutar el proceso de reducción de dimensionalidad con PCA, ya que este proceso no requiere de muchos recursos de cómputo, y dicho proceso se lleva a cabo en un Notebook independiente.

- **Clúster EDA:** Las características de este clúster ser muestran en la siguiente imagen:

**Figura 29 – Resumen del clúster para EDA**



**Figura 30 - Resumen del clúster para EDA**

Cómputo > Vista previa de la IU Enviar comentarios

**EDA** Ver más ... Iniciar Editar

**Configuración** Cuadernos (0) Bibliotecas Log de eventos IU de Spark Logs del driver Métricas Aplicaciones IU de cómputo Spark - Máster ▾

Sin restricciones

○ Multi-nodo ○ Nodo único

Modo de acceso

Compartido sin aislamiento

**Rendimiento**

Versión de Databricks Runtime

13.3 LTS (includes Apache Spark 3.4.1, Scala 2.12)

Utilizar aceleración Photon

Tipo de nodo

Standard\_DS3\_v2 14 GB de memoria y 4 núcleos

Terminar después de 60 minutos de inactividad

**Etiquetas**

Sin etiquetas personalizadas

> Etiquetas añadidas automáticamente

**Opciones avanzadas**

Transferencia de credenciales de Azure Data Lake Storage

Habilitar traspaso de credenciales para acceso a datos a nivel de usuario

**Spark** Registro Scripts de inicio JDBC/ODBC Permisos

Configuración de Spark

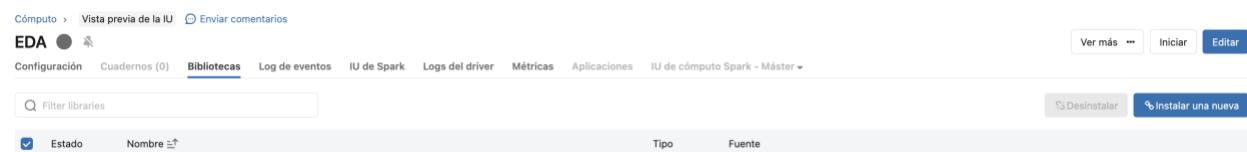
```
spark.databricks.delta.preview.enabled true
spark.master local[*]
spark.databricks.cluster.profile singleNode
```

Este clúster está compuesto solamente de 1 nodo con 4 núcleos y 14 GB de RAM. Y a diferencia de los otros clústeres, este utiliza aceleración Photon para acelerar y mejorar el rendimiento de las consultas. Este clúster se usa para ejecutar los procesos de Análisis Exploratorio o EDA (Exploratory Data Analysis) sobre los 3 distintos conjuntos de datos.

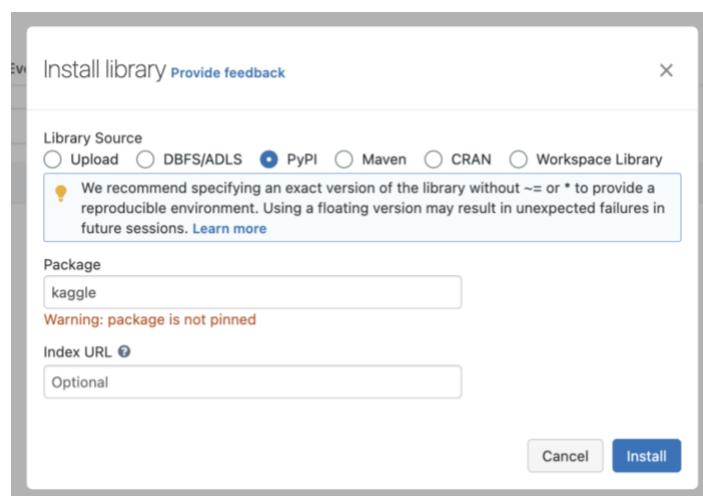
### 3.3.3. Librería y Credenciales de Kaggle

Después de la creación de los clústeres, se instala la biblioteca de Kaggle en el clúster de análisis exploratorio para su uso más eficiente en los notebooks, eliminando la necesidad de reinstalarla cada vez que se inicie el clúster.

*Figura 31 – Instalación de librería de Kaggle en el clúster de Databricks (1).*

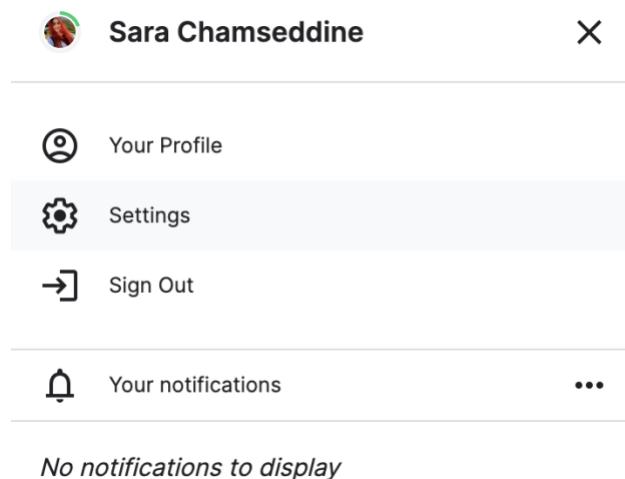


*Figura 32 - Instalación de librería de Kaggle en el clúster de Databricks (2).*



Después de la instalación de la biblioteca, será esencial configurar las credenciales de la cuenta de Kaggle que se utilizarán para la descarga del conjunto de datos a través del código y la API. Para ello se debe ingresar a la sección de “Settings” de la cuenta de Kaggle y generar un nuevo API token con la opción “Create new token”.

*Figura 33 – Generación de archivo de credenciales de Kaggle (1)*



*Figura 34 - Generación de archivo de credenciales de Kaggle (2)*

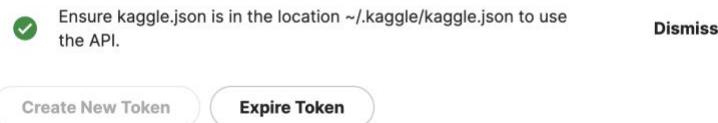
The screenshot shows the "Settings" page for a Kaggle account. At the top, there is a search bar and a "Your profile" link. Below the search bar, the word "Settings" is prominently displayed in large, bold letters. Underneath "Settings", there are two tabs: "Account" and "Notifications", with "Account" being the active tab. In the "Account" section, there is a "Phone verification" section. It states: "Your account is not verified. Verifying your account with a phone number allows you to do more on Kaggle, and helps prevent spam and other abuse." Below this text is a blue "Phone verify" button. To the right of this section, there are two account details: "Your username" (sarachamseddine (not editable)) and "Your account number" (4362907). Further down, there is an "API" section. It says: "Using Kaggle's beta API, you can interact with Competitions and Datasets to download data, make submissions, and more via the command line. [Read the docs](#)". Below this text are two buttons: "Create New Token" and "Expire Token". The overall layout is clean and organized, typical of a web-based application settings page.

Este proceso generará un archivo de credenciales para Kaggle en formato .JSON, el cual constará de dos variables: “username” y “key”.

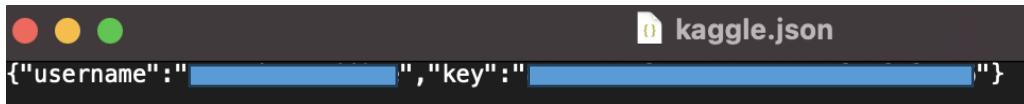
**Figura 35 - Generación de archivo de credenciales de Kaggle (3)**

### API

Using Kaggle's beta API, you can interact with Competitions and Datasets to download data, make submissions, and more via the command line. [Read the docs](#)

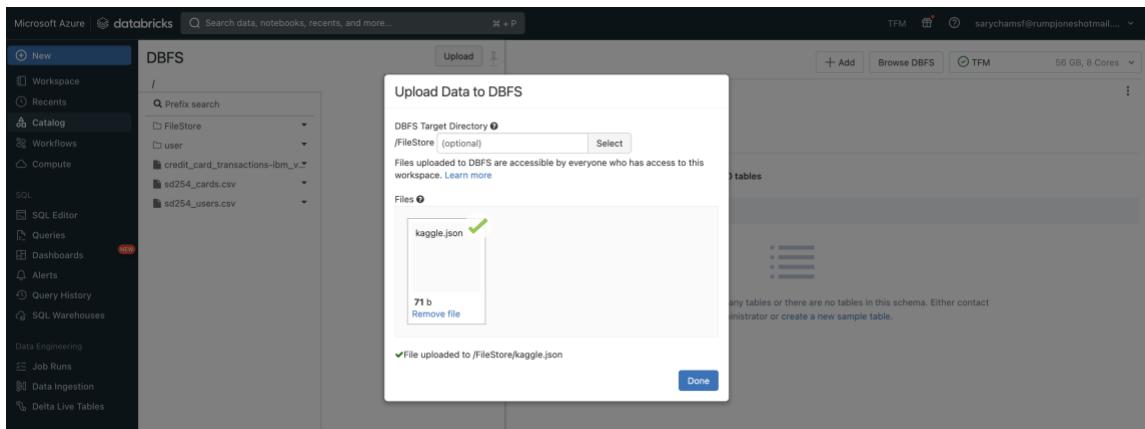


**Figura 36 - Generación de archivo de credenciales de Kaggle (4)**



El archivo .JSON de credenciales debe ser cargado en el sistema de archivos distribuido de Databricks (DBFS). Es fundamental conocer su ubicación en el DBFS para poder posteriormente referenciarlo en el notebook al realizar la descarga del conjunto de datos.

**Figura 37 - Generación de archivo de credenciales de Kaggle (5)**



## 4. Implementación

### 4.1. Extracción de datos

Se elabora un notebook que realiza el proceso de extracción de datos (ver anexo 8.1 1\_extracciondbc). En este, primero se crea una base de datos para trabajar:

*Figura 38 – Creación de base de datos en Databricks.*

```
1 %sql
2 CREATE DATABASE fraud_detection;
```

Luego se obtienen las credenciales del archivo JSON con las credenciales de Kaggle descargadas anteriormente (ver: 3.3.3 Librería y Credenciales de Kaggle) y se guardan como variables de entorno.

*Figura 39 – Lectura del archivo de credenciales de Kaggle*

```
1 # Ruta al archivo JSON en DBFS.
2 ruta_archivo = '/dbfs/FileStore/kaggle.json'
3
4 # Abrir el archivo y cargar su contenido JSON
5 with open(ruta_archivo, 'r') as archivo:
6     credenciales_kaggle = json.load(archivo)
```

Show result

Cmd 9

```
1 # Asignar las variables de entorno.
2 os.environ['KAGGLE_USERNAME'] = credenciales_kaggle["username"]
3 os.environ['KAGGLE_KEY'] = credenciales_kaggle["key"]
```

**Figura 40 - Almacenamiento de credenciales de Kaggle como variables de entorno**

```

1 %sh
2 kaggle config set -n username -v os.environ['KAGGLE_USERNAME']
3 kaggle config set -n key -v os.environ['KAGGLE_KEY']

```

Utilizando la API de Kaggle y las credenciales almacenadas como variables de entorno, se descarga el archivo con los datos. El resultado es un archivo comprimido en formato .zip, que se debe descomprimir para obtener todos los archivos del dataset.

**Figura 41 – Descarga y descompresión del dataset**

```

1 %sh
2 kaggle datasets download -d ealtman2019/credit-card-transactions
3 chmod 777 credit-card-transactions.zip
4 ls -l

```

Show result

Cmd 14

```

1 %sh
2 unzip credit-card-transactions.zip
3 ls -l

Archive: credit-card-transactions.zip
inflating: User0_credit_card_transactions.csv
inflating: credit_card_transactions-ibm_v2.csv
inflating: sd254_cards.csv
inflating: sd254_users.csv
total 2569248
-rw-r--r-- 1 root root 1899258 Oct 14 2021 User0_credit_card_transactions.csv
drwxr-xr-x 2 root root 4096 Oct 2 20:54 azure
drwxr-xr-x 2 root root 4096 Oct 2 20:54 conf
-rwxrwxrwx 1 root root 276210511 Oct 14 2021 credit-card-transactions.zip
-rw-r--r-- 1 root root 2350744057 Oct 14 2021 credit_card_transactions-ibm_v2.csv
drwxr-xr-x 3 root root 4096 Oct 2 20:58 eventlogs
-r-xr-xr-x 1 root root 2755 Oct 2 20:54 hadoop_accessed_config.lst
drwxr-xr-x 2 root root 4096 Oct 2 22:18 logs
-r-xr-xr-x 1 root root 1306936 Oct 2 20:54 preload_class.lst
-rw-r--r-- 1 root root 487120 Oct 14 2021 sd254_cards.csv
-rw-r--r-- 1 root root 224394 Oct 14 2021 sd254_users.csv

```

Se mueven estos archivos descargados de la máquina o entorno de control de Databricks al sistema de archivos distribuido de Databricks (DBFS). Esto va a permitir que los archivos estén disponibles para su procesamiento.

**Figura 42 – Almacenamiento de archivos del dataset en el DBFS**

```

1 # Se almacenan los nombres de los archivos en una lista.
2 file_names = ["credit_card_transactions-ibm_v2.csv", "sd254_cards.csv", "sd254_users.csv"]

Command took 0.11 seconds -- by sarychamsf@rumpjoneshotmail.onmicrosoft.com at 3/10/2023, 12:42:25 a. m. on TFM
Cmd 19

1 for f in file_names:
2     dbutils.fs.mv(f"file:/databricks/driver/{f}", f"dbfs:/{f}")

Command took 11.36 seconds -- by sarychamsf@rumpjoneshotmail.onmicrosoft.com at 3/10/2023, 12:42:27 a. m. on TFM
Cmd 20

1 %fs
2 ls dbfs:/
```

Table		New result table: OFF		
	path	name	size	modificationTir
1	dbfs:/FileStore/	FileStore/	0	169628589000
2	dbfs:/Volume/	Volume/	0	0
3	dbfs:/Volumes/	Volumes/	0	0
4	dbfs:/credit_card_transactions-ibm_v2.csv	credit_card_transactions-ibm_v2.csv	2350744057	169628655800
5	dbfs:/databricks-datasets/	databricks-datasets/	0	0
6	dbfs:/databricks-results/	databricks-results/	0	0

Finalmente, se procede a la lectura de los archivos y se almacena su contenido en tablas Delta de Databricks. Se crearán tres tablas en total, cada una correspondiente a los datos descargados. Estas tablas llevarán el prefijo 'raw\_' para indicar que los datos no han experimentado modificaciones con respecto a los datos originales. Las tablas se denominarán 'raw\_transactions', 'raw\_cards' y 'raw\_users'. Como un paso adicional, se desarrolla un método destinado a mejorar los nombres de las columnas en el DataFrame que servirá como base para cada tabla. Esto implica la sustitución de espacios por guiones bajos y la eliminación de caracteres especiales o inapropiados en los nombres de las columnas.

*Figura 43 – Creación de raw delta tables*

```

1 """
2 Este método mejora los nombres de las columnas del dataframe
3 sustituyendo los espacios por guiones bajos y eliminando
4 los signos de interrogación.
5
6 Input:
7   df: DataFrame
8
9 """
10 def fix_column_names(df):
11     for col_name in df.columns:
12         new_col_name = col_name.lower().replace(" ", "_").replace("?", "")
13         df = df.withColumnRenamed(col_name, new_col_name)
14     return df

```

Command took 0.13 seconds -- by sarychamsf@rumpjoneshotmail.onmicrosoft.com at 3/10/2023, 12:42:55 a. m. on TFM

Cmd 23

```

1 database = "fraud_detection"
2 table_names = ["transactions", "cards", "users"]

```

Command took 0.11 seconds -- by sarychamsf@rumpjoneshotmail.onmicrosoft.com at 3/10/2023, 12:46:33 a. m. on TFM

Cmd 24

```

1 # Se escriben los DataFrames a Delta tables.
2 for index, f in enumerate(file_names):
3     # Ruta al archivo CSV en DBFS.
4     archivo_csv_dbfs = f"dbfs:/{{f}}"
5     # Leer el archivo CSV y crear un DataFrame.
6     df = spark.read.csv(archivo_csv_dbfs, header=True, inferSchema=True)
7     df = fix_column_names(df)
8     # Guardar el DataFrame como una tabla Delta.
9     df.write.format("delta").mode('overwrite').saveAsTable(f"{{database}}.raw_{{table_names[index]}}")
10    print(f"Tabla creada: {{database}}.raw_{{table_names[index]}}")

```

▶ (24) Spark Jobs

▶ df: pyspark.sql.DataFrame = [person: string, current\_age: integer ... 16 more fields]

Tabla creada: fraud\_detection.raw\_transactions

Tabla creada: fraud\_detection.raw\_cards

Tabla creada: fraud\_detection.raw\_users

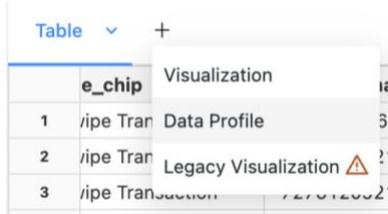
## 4.2. Análisis Exploratorio, transformación y feature engineering

En esta sección, se utiliza PySpark en Databricks para llevar a cabo un análisis exploratorio de los datos con el propósito de comprender su naturaleza, identificar patrones y abordar posibles problemas de calidad de datos. Este proceso es fundamental para preparar los datos y mejorar su

idoneidad para la construcción de modelos de aprendizaje automático. Además, se realizan transformaciones en los datos y se crean nuevas características (features) que resultan relevantes y beneficiosas para los modelos de aprendizaje automático que se desarrollan posteriormente.

En muchos casos, para la visualización y el perfilado de los datos, se empleará la interfaz de Databricks. Esto se logra al utilizar el método ".display()" en un DataFrame y hacer clic en el botón de adición (+) para acceder a las herramientas de visualización disponibles. Se muestra un ejemplo:

**Figura 44 – Crear visualizaciones y perfiles de datos en Databricks**



#### 4.2.1. Transacciones

A continuación se mostrará el paso a paso realizado para el notebook de transacciones (para más información, se puede revisar el anexo 8.2 2\_1\_eda\_transformacion\_transactionsdbc). Primero, se debe leer la tabla raw creada en el proceso de extracción.

**Figura 45 – Lectura de la tabla raw\_transactions**

```
1 # Se lee la tabla raw.
2 raw_transactions = spark.read.table(f'{database}.raw_transactions')
```

En este caso, no se aplicarán particiones, pues el clúster es suficiente para hacer el análisis exploratorio y estamos tratando con raw data. Posteriormente, se procede a hacer un análisis

exploratorio parcial del dataset. Iniciando con crear una columna de fecha con las columnas de year, month y day.

**Figura 46 – Tabla de transacciones: crear columna de fecha.**

```
2   raw_transactions = raw_transactions.withColumn(
3     "transaction_date", to_date(concat_ws("-", col("year"), col("month"), col("day")))
4   )
```

Se crea el perfil de datos para variables numéricas y categóricas.

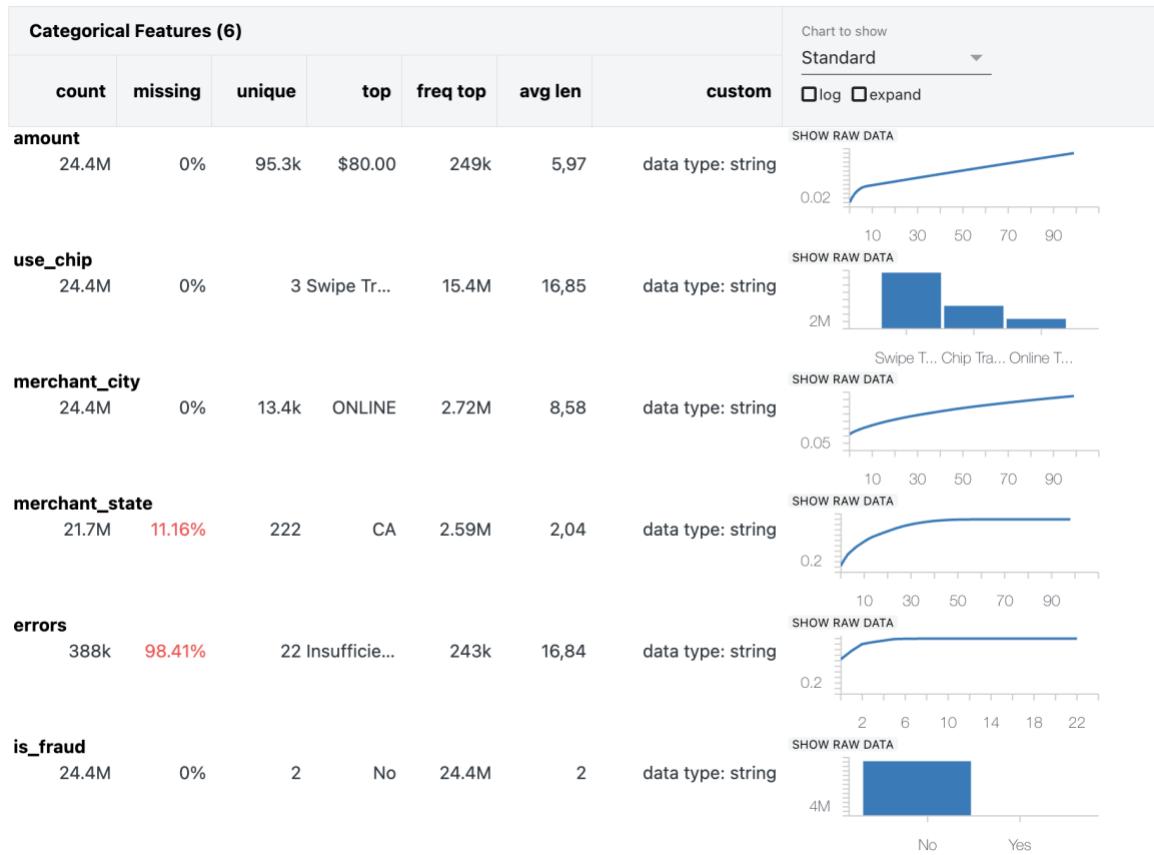
**Figura 47 – Perfil de datos numéricos de la tabla de transacciones**



Como podemos observar:

- Hay algunas variables que, aunque son numéricas, no tiene mucho sentido analizar porque se comportan como categóricas. Por ejemplo: user, card, zip, merchant\_name, mcc.
- A medida que pasa el tiempo, incrementa el número de transacciones (columna year y transaction\_date).
- Los usuarios tienen al menos una tarjeta y la mayoría de ellos tiene solo una. Se observa también que disminuyen paulatinamente el número de transacciones por segunda, tercera o más tarjetas por usuario por esta razón.
- Hay mayor cantidad de transacciones en los primeros dos meses del año y los últimos dos meses del año. Permaneciendo más o menos constante el resto del tiempo.
- Hay mayor cantidad de transacciones en los últimos días del mes, pero en general se mantiene constante.
- La variable time es agregada por Databricks con la hora en que se procesó el dataset. No es relevante para el análisis.
- Sólo existen valores nulos para la columna zip.
- Las columnas user y card contiene ceros. Esto se debe a que se utiliza el cero como ID para ambas columnas. No afecta el análisis.
- Hay transacciones para 2000 usuarios. Se poseen registros desde 1991 hasta 2020.
- Una alta desviación estándar indica que los valores están más dispersos alrededor de la media. Esto sugiere que los datos son más variables y menos predecibles. Sin embargo, en este caso no se puede analizar para la mayoría de las variables porque son realmente categóricas y esto sólo se puede calcular para las variables numéricas.

*Figura 48 – Perfil de datos categóricos de la tabla de transacciones*

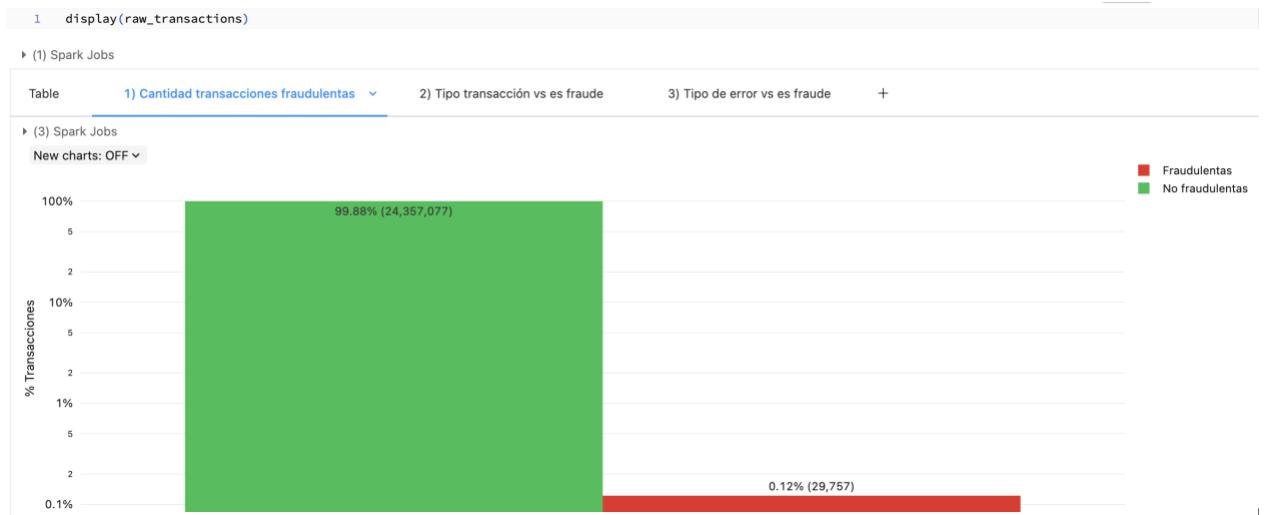


- La variable amount debería ser numérica, por lo que se deberá cambiar el tipo de dato.
- La variable use\_chip tiene 3 posibles valores: Swipe, Chip y Online.
- La mayoría de las transacciones se llevan a cabo Online.
- Hay valores faltantes para merchant\_state y errors, habrá que analizar la razón y darles tratamiento respectivamente.
- La variable a predecir es is\_fraud. Podemos observar que la mayoría de las transacciones no son fraudulentas. Sin embargo, las clases YES o NO están bastante desproporcionadas. Esto puede añadir complejidad al modelo, pero este tema se tratará posteriormente.

Se incluyen adicionalmente las siguientes visualizaciones:

- **Cantidad transacciones fraudulentas:** diagrama de barras en escala logarítmica para representar el porcentaje y las cantidades de transacciones fraudulentas y no fraudulentas. Se observa que las transacciones fraudulentas son sustancialmente menos frecuentes. Esta disparidad sugiere que el dataset presenta desequilibrio de clases, y será necesario aplicar enfoques especiales para detectar y abordar estas anomalías de manera efectiva.
- **Tipo transacción vs es fraude:** diagrama de barras en escala de porcentaje para analizar la relación entre el tipo de transacción y su carácter fraudulento. Este análisis revela que las transacciones en línea son las más susceptibles a ser fraudulentas.
- **Tipo de error vs es fraude:** muestra el porcentaje de transacciones fraudulentas por cada tipo de error presentado. Se puede observar que el error más frecuente es la combinación de “Bad Expiration, Technical Glitch”. Resulta un poco difícil analizar los datos en este formato, por lo que se les dará tratamiento posteriormente.

**Figura 49 – Distribución Logarítmica de Transacciones Fraudulentas y No Fraudulentas**



*Figura 50 – Diagrama de barras: tipo de transacción vs es fraude*



*Figura 51 – Diagrama de barras: tipo de error vs es fraude*



Algunos tipos de datos pueden no ser la elección adecuada para representar su contenido, y es posible que ciertas columnas en la base de datos carezcan de relevancia. Con el objetivo de simplificar el análisis del conjunto de datos a través de visualizaciones y para determinar si se necesitan mejoras en la calidad de los datos, se realizan validaciones y transformaciones básicas. Además, se llevarán a cabo más visualizaciones para profundizar en la comprensión del dataset.

Se inicia validando si hay duplicados, eliminándolos de ser necesario y además visualizando los tipos de datos de las columnas para tener claras las transformaciones a aplicar.

**Figura 52 – Eliminación de duplicados para la tabla de transacciones**

```

2   cant_transactions = raw_transactions.count()
3   print(cant_transactions)

▶ (2) Spark Jobs
24386900
1   # Validar si hay transacciones duplicadas y eliminarlas.
2   num_duplicates = cant_transactions - raw_transactions.dropDuplicates().count()
3
4   if num_duplicates > 0:
5       raw_transactions = raw_transactions.dropDuplicates()
6
7   print(
8       f"Duplicados encontrados y eliminados de la tabla de transacciones: {num_duplicates}"
9   )

▶ (3) Spark Jobs
▶ raw_transactions: pyspark.sql.dataframe.DataFrame = [user: integer, card: integer ... 14 more fields]
Duplicados encontrados y eliminados de la tabla de transacciones: 66

```

**Figura 53 – Esquema de la tabla de transacciones antes de la transformación de datos**

```

2   raw_transactions.printSchema()

root
 |-- user: integer (nullable = true)
 |-- card: integer (nullable = true)
 |-- year: integer (nullable = true)
 |-- month: integer (nullable = true)
 |-- day: integer (nullable = true)
 |-- time: timestamp (nullable = true)
 |-- amount: string (nullable = true)
 |-- use_chip: string (nullable = true)
 |-- merchant_name: long (nullable = true)
 |-- merchant_city: string (nullable = true)
 |-- merchant_state: string (nullable = true)
 |-- zip: double (nullable = true)
 |-- mcc: integer (nullable = true)
 |-- errors: string (nullable = true)
 |-- is_fraud: string (nullable = true)
 |-- transaction_date: date (nullable = true)

```

Se inicia cambiando el tipo de dato de la variable target: is\_fraud. Como es una variable binaria, se asignará 1 para “yes” y 0 para “no”, convirtiéndola en numérica.

**Figura 54 – Tabla de transacciones: cambio del tipo de dato de la columna is\_fraud**

```

2   stg_transactions = raw_transactions.withColumn(
3       "is_fraud", when(col("is_fraud") == "Yes", 1).when(col("is_fraud") == "No", 0)
4   )

```

Se eliminan las columnas de fecha no necesarias.

**Figura 55 – Eliminación de variables innecesarias en la tabla de transacciones**

```
2   stg_transactions = stg_transactions.drop(*["year", "month", "day"])
```

Para visualizar cómo se distribuye la variable *amount*, se cambiará el tipo de dato de string a decimal, y así se podrá realizar un histograma. Para ello se reemplaza el signo \$ en la columna de *amount*. El signo \$ es un carácter especial, por lo que se debe usar un código de escape. A su vez, se convierte la columna a tipo decimal. Para saber qué cantidad de bins colocar al histograma, se tomará en cuenta la regla de Sturges que sugiere utilizar  $\text{ceil}(\log_2(n) + 1)$  bins, donde "n" es el número de datos. Esta regla tiende a funcionar bien para tamaños de muestra grandes.

Como el dataset es muy grande, en este caso se utiliza la librería de Seaborn, pues Databricks no logra graficar todos los datos de manera óptima.

**Figura 56 – Transformación de la variable amount**

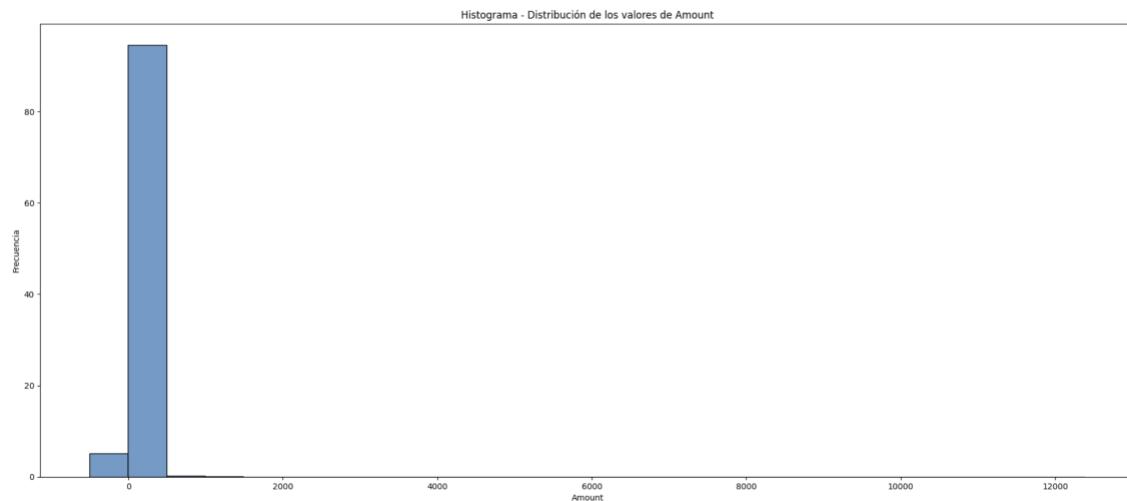
```
3   stg_transactions = stg_transactions.withColumn(
4     "amount",
5     regexp_replace(stg_transactions["amount"], "\\$", "").cast(DecimalType(10, 2)),
6   )
```

**Figura 57 – Creación de histograma para la variable amount**

```
2   bins_transactions = math.ceil(math.log2(cant_transactions) + 1)
3   print(bins_transactions)

2   amounts = stg_transactions.select("amount").rdd.flatMap(lambda x: x).collect()
1   # Se crea un histograma utilizando seaborn
2   plt.figure(figsize=(24, 10))
3
4   sns.histplot(amounts, bins=bins_transactions, color="#3A7AB7", stat="percent")
5
6   # Se añaden las etiquetas y título.
7
8   plt.xlabel("Amount")
9   plt.ylabel("Frecuencia")
10  plt.title("Histograma - Distribución de los valores de Amount")
11
12  # Se muestra el histograma
13  plt.show()
```

**Figura 58 – Histograma: distribución de los valores de “amount”**



Se puede observar que la mayoría de las transacciones representan valores menores a \$500.

También hay una gran cantidad de valores negativos. Estos podrían representar devoluciones.

¿Valdría la pena filtrarlas? ¿Se puede considerar una devolución como fraudulenta?

**Figura 59 – Tabla de transacciones: cantidad de devoluciones.**

```
1  print(f"Cantidad de devoluciones: {stg_transactions.filter(col('amount') < 0).count()}")
▶ (3) Spark Jobs
Cantidad de devoluciones: 1244676
```

El fraude en devoluciones podría abarcar diversas tácticas, como devoluciones ficticias, manipulación de montos de devoluciones, el uso de tarjetas robadas o comprometidas, colusión interna y el abuso de políticas de devolución laxas. Estas estrategias fraudulentas involucran a estafadores que buscan obtener reembolsos injustificados a expensas de las empresas. Por lo tanto, no se realizará un filtrado de estos datos, pues un 5% del dataset son devoluciones.

Se analizará también la distribución de montos de las transacciones fraudulentas usando el mismo método.

**Figura 60 - Creación de histograma para la variable amount en las transacciones fraudulentas**

```

1  transacciones_fraudulentas = stg_transactions.filter(col("is_fraud") == 1)
2  cant_fraud = transacciones_fraudulentas.count()
3
4  # Aplicar la función ceil(log2(cant) + 1) al número de registros
5  bins_fraud = math.ceil(math.log2(cant_fraud) + 1)
6  print(bins_fraud)
7
8  print(f"Número de bin para transacciones fraudulentas: {bins_fraud}")

```

▶ (3) Spark Jobs

▶ pyspark.sql.dataframe.DataFrame transacciones\_fraudulentas: pyspark.sql.dataframe.DataFrame = [user: integer, card: integer ... 11 more fields]

16  
Número de bin para transacciones fraudulentas: 16

Command took 4.89 seconds -- by sarychamsf@rumpjoneshotmail.onmicrosoft.com at 12/10/2023, 8:28:49 p. m. on Sar

---

Cmd 43

```

1  # Selecciona la columna amount para el histograma
2  amounts_fraud = (
3  |   transacciones_fraudulentas.select("amount").rdd.flatMap(lambda x: x).collect()
4  )

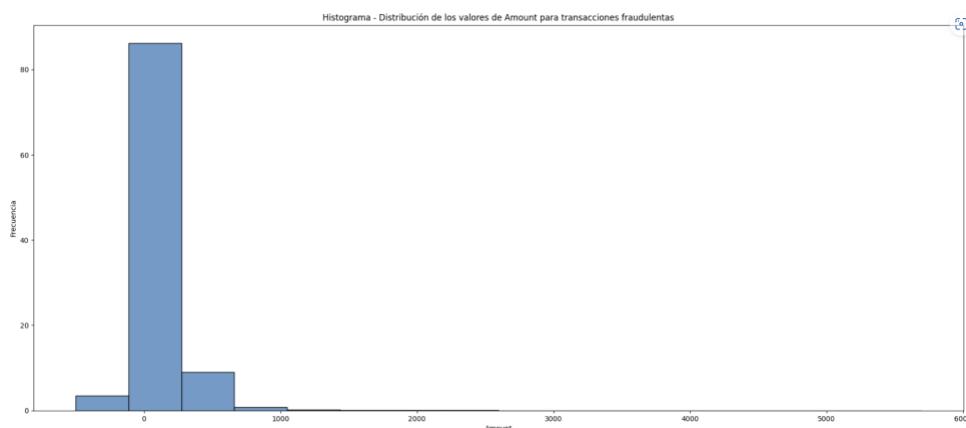
```

```

1  # Se crea un histograma utilizando seaborn
2  plt.figure(figsize=(24, 10))
3
4  sns.histplot(amounts_fraud, bins=bins_fraud, color="#3A7AB7", stat="percent")
5
6  # Se añaden las etiquetas y título.
7
8  plt.xlabel("Amount")
9  plt.ylabel("Frecuencia")
10 plt.title(
11 |   "Histograma - Distribución de los valores de Amount para transacciones fraudulentas"
12 )
13
14 # Se muestra el histograma
15 plt.show()

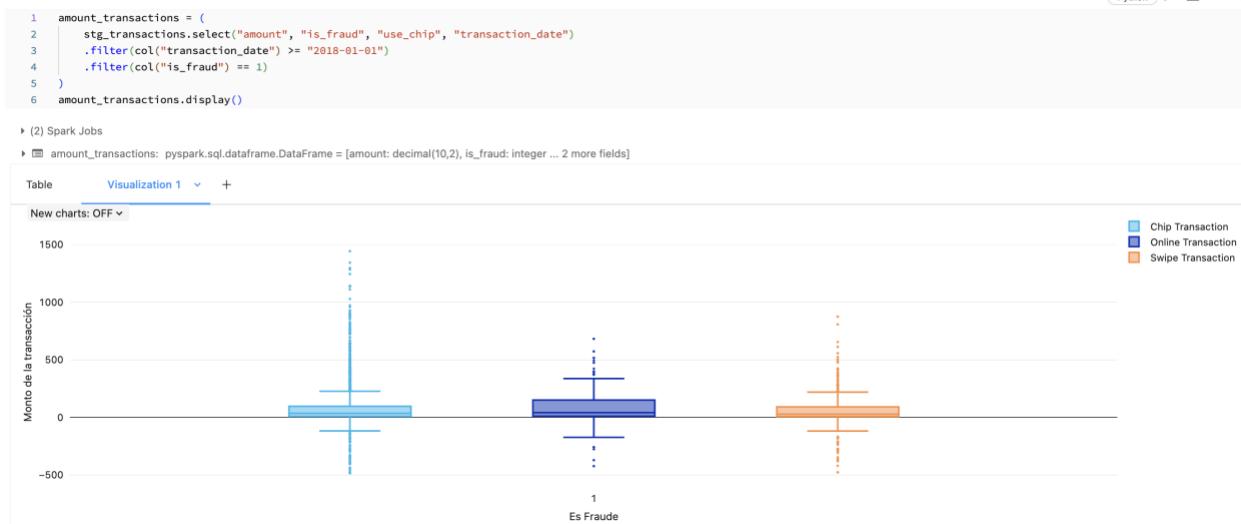
```

**Figura 61 - Histograma para la variable amount en las transacciones fraudulentas**



Se puede observar que las transacciones fraudulentas suelen darse en montos más pequeños principalmente, y algunas veces, incluso en devoluciones. Se puede hacer un análisis también para ver cómo se distribuyen los montos de las transacciones si estas son fraudulentas, y tomando en cuenta el tipo de transacción. Como son muchos los datos, se filtrarán los últimos 3 años (desde el 2018 en adelante).

**Figura 62 – Boxplot para el monto de las transacciones fraudulentas por tipo de transacción**



En general los montos medios son similares entre los diferentes tipos de transacciones. Sin embargo, se puede apreciarse que para las transacciones de chip existe una cantidad muy grande de outliers. Se pueden analizar las agregaciones para todas las transacciones, teniendo en cuenta si fueron o no fraudulentas y qué tipo de transacción fue (si usó chip o no).

**Figura 63 – Cálculos estadísticos para transacciones fraudulentas y no fraudulentas que usan o no usan chip**

```

1 # Realizar la agregación
2 aggregated_data = (
3     stg_transactions.groupBy("is_fraud", "use_chip")
4     .agg(
5         mean("amount").alias("mean_amount"),
6         percentile_approx("amount", 0.25).alias("q1_amount"),
7         percentile_approx("amount", 0.5).alias("median_amount"),
8         percentile_approx("amount", 0.75).alias("q3_amount"),
9         min("amount").alias("min_amount"),
10        max("amount").alias("max_amount"),
11    )
12    .orderBy("is_fraud")
13 )
14
15 # Mostrar el DataFrame con los resultados agrupados
16 display(aggregated_data)

```

▶ (3) Spark Jobs  
▶ aggregated\_data: pyspark.sql.dataframe.DataFrame = [is\_fraud: integer, use\_chip: string ... 6 more fields]

Table +											
	is_fraud	use_chip	mean_amount	q1_amount	median_amount	q3_amount	min_amount	max_amount			
1	0	Online Transaction	56.298458	21.84	34.73	56.19	-500.00	12390.50			
2	0	Swipe Transaction	42.337719	8.01	29.19	67.00	-500.00	6820.20			
3	0	Chip Transaction	41.067045	7.58	27.47	64.30	-500.00	5712.06			
4	1	Swipe Transaction	99.938036	18.22	66.20	136.19	-492.00	2259.66			
5	1	Chip Transaction	84.679518	9.42	40.41	102.64	-484.00	1442.63			
6	1	Online Transaction	117.992027	24.92	81.70	164.26	-500.00	5694.44			

Se puede observar a través de los resultados anteriores que:

- Las transacciones fraudulentas tienden a tener montos promedio más altos en comparación con las no fraudulentas. Por ejemplo, las transacciones fraudulentas tienen un promedio de alrededor de \$99.94 (Swipe) y \$117.99 (Online), mientras que las no fraudulentas tienen promedios más bajos, alrededor de \$42.34 (Swipe) y \$41.07 (Chip).
- Las transacciones con chip tienden a tener montos promedio más bajos en comparación con las transacciones online y swipe, tanto para transacciones fraudulentas como no fraudulentas.
- El cuartil 1 (q1\_amount) muestra el 25% inferior de las transacciones. Las transacciones fraudulentas tienen cuartiles 1 más bajos en comparación con las no fraudulentas. Lo

Lo mismo pasa con los otros cuartiles, como el cuartil 3 (q3\_amount) que muestra el 75% inferior de las transacciones.

- Las transacciones fraudulentas tienden a tener medianas más altas que las no fraudulentas.
- Las transacciones fraudulentas tienen valores mínimos y máximos que varían ampliamente, con algunos valores atípicos (como -492 y 2259.66 para Swipe).
- Las transacciones no fraudulentas también tienen valores mínimos y máximos bajos y altos, respectivamente, pero en general, son menos extremos que las transacciones fraudulentas.
- El dataset es disperso porque la mediana y la media son alejadas entre sí.

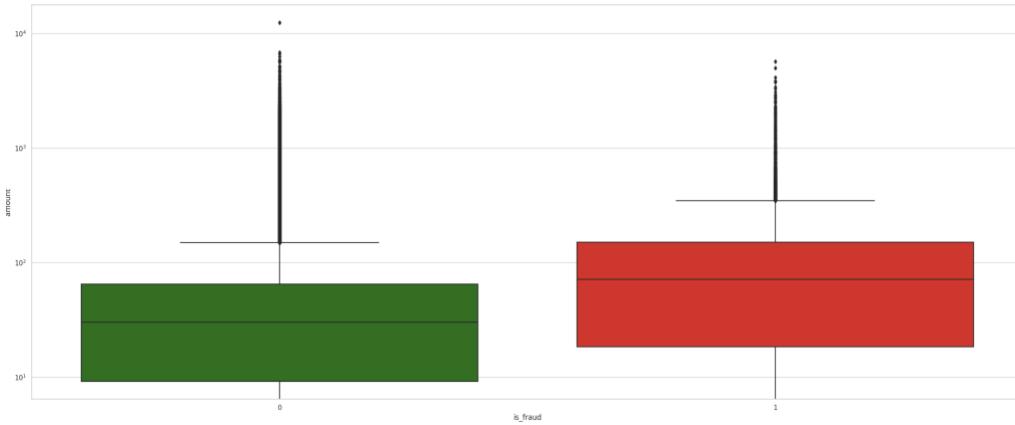
En resumen, este análisis sugiere que las transacciones fraudulentas tienden a involucrar montos más altos en comparación con las transacciones no fraudulentas. Además, las transacciones con chip tienden a tener montos promedio más bajos en general. Esto se puede visualizar también con un diagrama de caja y bigotes.

**Figura 64 – Creación de Boxplot con los montos de las transacciones según si fue fraude o no**

```

1 # Convertir la columna "amount" a tipo de dato double
2 amount_transactions = stg_transactions.select("amount", "is_fraud").withColumn(
3     "amount", stg_transactions["amount"].cast("double")
4 )
5
6 # Especificar colores para las categorías de "is_fraud"
7 colors = {0: "green", 1: "red"}
8
9 # Crear un boxplot utilizando seaborn y matplotlib
10 plt.figure(figsize=(24, 10))
11 sns.set(style="whitegrid")
12 sns.boxplot(
13     x="is_fraud", y="amount", data=amount_transactions.toPandas(), palette=colors
14 )
15 plt.yscale("log")
16
17 # Mostrar el gráfico
18 plt.tight_layout()
19 plt.show()
```

*Figura 65 - Boxplot con los montos de las transacciones según si fue fraude o no*



Se realizan unas gráficas que permiten conocer el comportamiento de las transacciones fraudulentas y no fraudulentas a través del tiempo. Para ello se crea un método que muestra el recuento de transacciones (fraudulentas o no fraudulentas) por año-mes.

*Figura 66 – Creación de gráficas con el comportamiento de transacciones en el tiempo*

```

17 def transactions_date_agg(
18     stg_transactions: DataFrame, meses: DataFrame, is_fraud: int
19 ) -> DataFrame:
20     # Filtrar las transacciones dependiendo de si son o no fraudulentas y agregar el recuento.
21     transactions = (
22         stg_transactions.filter(f"is_fraud = {is_fraud}")
23         .withColumn("transaction_month", month("transaction_date"))
24         .withColumn("transaction_year", year("transaction_date"))
25         .groupBy("transaction_year", "transaction_month")
26         .agg(count("*").alias("count"))
27     )
28
29     # Realizar un left join para incluir los meses sin transacciones no fraudulentas
30     transactions_agg = (
31         meses.join(
32             transactions, on=["transaction_year", "transaction_month"], how="left"
33         )
34         .fillna(0, subset=["count"])
35         .orderBy("transaction_year", "transaction_month")
36         .withColumn(
37             "year_month",
38             concat(
39                 col("transaction_year"),
40                 lit("-"),
41                 lpad(col("transaction_month"), 2, "0"),
42             ),
43         )
44     )
45
46     return transactions_agg

```

```

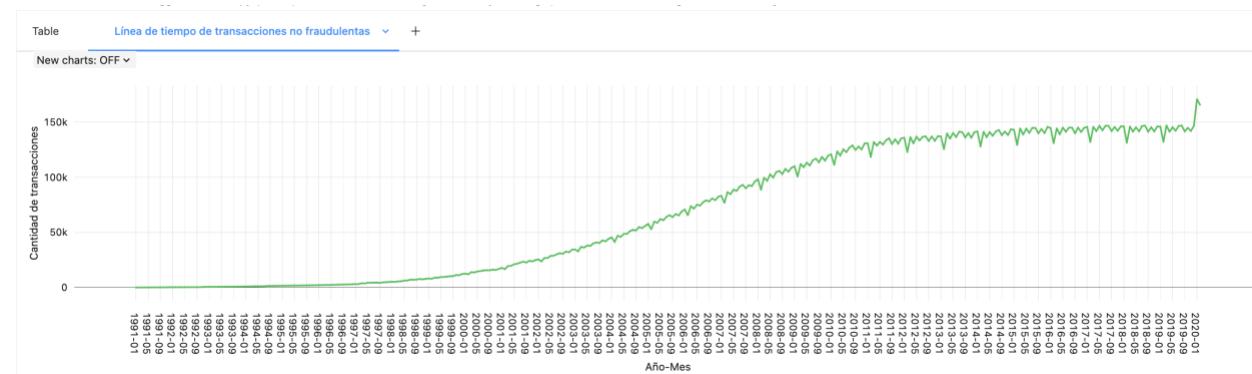
1 # Crear un DataFrame con todos los meses posibles
2 meses = stg_transactions.select(
3     year("transaction_date").alias("transaction_year"),
4     month("transaction_date").alias("transaction_month"),
5 ).distinct()

▶ [meses: pyspark.sql.dataframe.DataFrame = [transaction_year: integer, transaction_month: integer]
Command took 0.20 seconds -- by sarychamsf@rumpjoneshotmail.onmicrosoft.com at 12/10/2023, 8:28:49 p.
Cmd 55

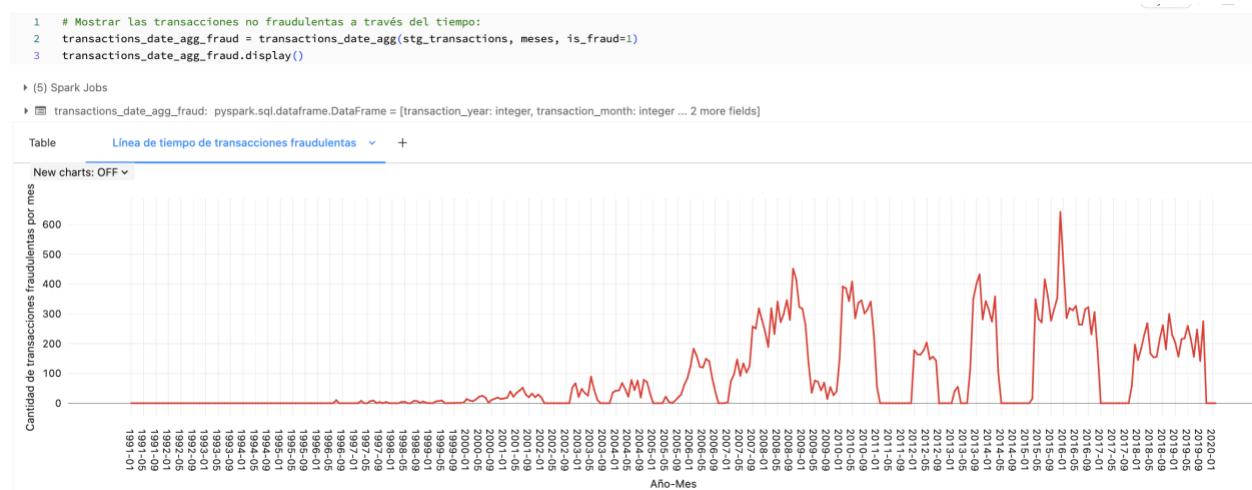
1 # Mostrar las transacciones no fraudulentas a través del tiempo:
2 transactions_date_agg_non_fraud = transactions_date_agg(
3     stg_transactions, meses, is_fraud=0
4 )
5 transactions_date_agg_non_fraud.display()

```

*Figura 67 – Gráfica con el comportamiento de transacciones no fraudulentas en el tiempo*



*Figura 68 - Gráfica con el comportamiento de transacciones fraudulentas en el tiempo*



Se puede observar que la cantidad de transacciones en general aumenta con el pasar del tiempo.

Las transacciones fraudulentas aumentan también cuando hay más transacciones, pero no se observa un patrón específico en ellas. Sin embargo, el enfoque del proyecto será en detección de anomalías, no en series de tiempo.

Ahora se analiza la variable "MCC," que representa el código de categoría de comercio. Es posible que ciertas categorías de comercio sean más susceptibles a la presencia de actividades fraudulentas. Para mejorar el análisis, se enriquece el conjunto de datos con información adicional obtenida de una fuente externa. Esta información adicional incluye el nombre completo del MCC (Merchant Category Code), en lugar de su código numérico. El proceso implica leer el CSV con estos códigos descrito en la sección “3.1.4 MCC Codes (mcc\_codes.csv)”.

**Figura 69 – Enriquecimiento del dataset con los nombres de los MCC**

```

1 # URL del archivo CSV
2 url = "https://raw.githubusercontent.com/greggles/mcc-codes/main/mcc_codes.csv"
3
4 # Realizar una solicitud GET para descargar el contenido del archivo CSV
5 response = requests.get(url)
6
7 # Leer el contenido descargado en un DataFrame de Spark
8 mcc_data = response.text
9 mcc_data = spark.read.csv(
10 |   spark.sparkContext.parallelize(mcc_data.splitlines()), header=True, inferSchema=True
11 )
12
13 mcc_data = mcc_data.select("mcc", col("edited_description").alias("mcc_category"))
14
15 # Mostrar el DataFrame (si lo deseas)
16 mcc_data.display()

▶ (4) Spark Jobs
▶ mcc_data: pyspark.sql.dataframe.DataFrame = [mcc: integer, mcc_category: string]

Table 981 rows | 2.98 seconds runtime

+-----+-----+
| mcc | mcc_category |
+-----+-----+
| 1   | Veterinary Services |
| 2   | Agricultural Co-operatives |
| 3   | Horticultural Services, Landscaping Services |
| 4   | General Contractors-Residential and Commercial |
| 5   | Air Conditioning Contractors – Sales and Installation, Heating Contractors – Sales, Service, Installation |
| 6   | Electrical Contractors |
+-----+-----+
| 981 rows | 2.98 seconds runtime

1 # Agregar la categoría al DataFrame de transacciones
2 stg_transactions = stg_transactions.join(mcc_data, on="mcc", how="left").withColumn(
3 |   "mcc", col("mcc").cast("string")
4 )

```

```

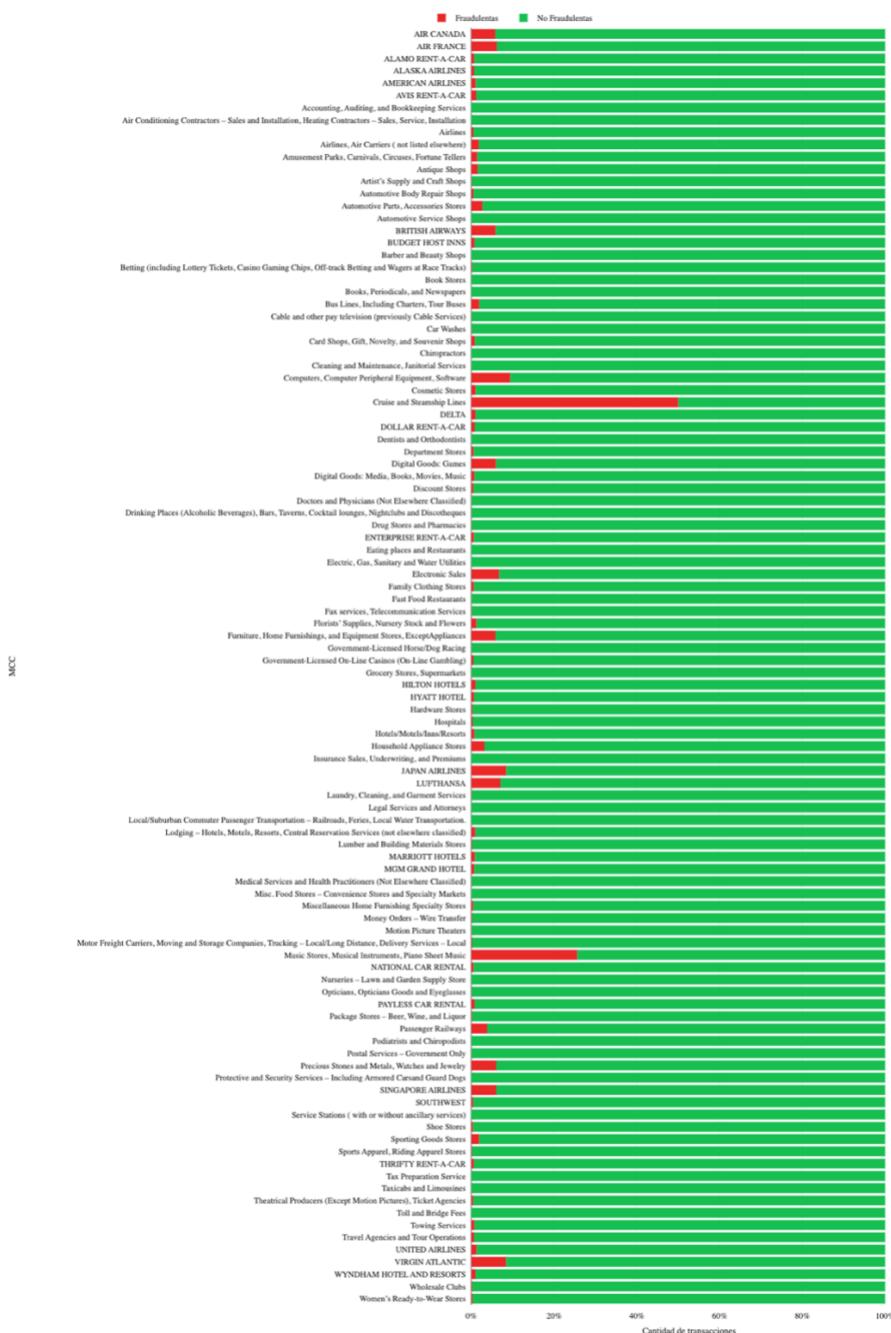
1  # Se valida que no haya categorías de comercio vacías luego de hacer merge con la nueva fuente de datos.
2  stg_transactions.filter(col("mcc_category").isNull()).count()

▶ (4) Spark Jobs
0

1  mcc_transactions = stg_transactions.groupBy("mcc_category", "is_fraud").agg(
2  |   count("*").alias("cantidad")
3  )
4  mcc_transactions.display()

```

**Figura 70 – Transacciones fraudulentas por MCC**



Podemos observar que los tipos de comercio en donde ocurren más fraudes se relacionan a cruceros, aerolíneas, software, videojuegos, música, ventas electrónicas, artículos de hogar y joyería.

Se realizará feature engineering con la columna "errors", separando todos sus valores en diferentes columnas y asignando 0 o 1 si un error específico ocurrió en una transacción (One-hot Encoding).

**Figura 71 – Distribución de los valores de la variable errors**

```

1 # Distribución de los valores de la variable errors.
2 stg_transactions.groupBy("errors").agg(count("*").alias("count")).orderBy(
3   desc("count")
4 ).display()

```

▶ (4) Spark Jobs

	errors	count
1	null	23998449
2	Insufficient Balance	242737
3	Bad PIN	58918
4	Technical Glitch	48157
5	Bad Card Number	13321
6	Bad CVV	10740
7	Bad Expiration	10716
8	Bad Zipcode	2079
9	Bad PIN,Insufficient Balance	581
10	Insufficient Balance,Technical Glitch	457
11	Bad PIN,Technical Glitch	128
12	Bad Card Number,Insufficient Balance	122
13	Bad CVV,Insufficient Balance	89
14	Bad Expiration,Insufficient Balance	78
15	Bad Card Number,Bad CVV	60
16	Bad Card Number,Bad Expiration	54
17	Bad Expiration,Bad CVV	47
18	Bad Expiration,Technical Glitch	32
19	Bad Card Number,Technical Glitch	25
20	Bad CVV,Technical Glitch	21
21	Bad Zipcode,Insufficient Balance	13
22	Bad Zipcode,Technical Glitch	7
23	Bad Card Number,Bad Expiration,Insufficient Balance	2
24	Bad Card Number,Bad Expiration,Technical Glitch	1

Las causas del fallo de una transacción pueden ser varias a la vez. Es mejor separarlas, pues sólo hay 8 diferentes. Las demás son sólo combinaciones de las 8 anteriores.

**Figura 72 – One Hot Encoding a la columna de errors**

```

1  # Dividir la columna 'errors' en múltiples columnas usando la coma como separador.
2  stg_transactions = stg_transactions.withColumn("error_list", split(col("errors"), ","))
3
4  # Obtener la lista de tipos de errores únicos.
5  error_types = (
6      stg_transactions.selectExpr("explode(error_list) as error")
7      .distinct()
8      .rdd.map(lambda row: row.error)
9      .collect()
10 )
11
12 # Crear columnas separadas para cada tipo de error y asignar 1 si está presente o 0 si no.
13 for error_type in error_types:
14     stg_transactions = stg_transactions.withColumn(
15         error_type, when(array_contains(col("error_list"), error_type), 1).otherwise(0)
16     )
17
18 # Eliminar la columna 'error_list' si ya no es necesaria.
19 stg_transactions = stg_transactions.drop("error_list")
20 stg_transactions = stg_transactions.drop("errors")
21
22 # Arreglar los nombres de las nuevas columnas.
23 stg_transactions = fix_column_names(stg_transactions)

```

```
1  stg_transactions.columns
```

```
['mcc',
 'user',
 'card',
 'time',
 'amount',
 'use_chip',
 'merchant_name',
 'merchant_city',
 'merchant_state',
 'zip',
 'is_fraud',
 'transaction_date',
 'mcc_category',
 'technical_glitch',
 'insufficient_balance',
 'bad_card_number',
 'bad_pin',
 'bad_cvv',
 'bad_expiration',
 'bad_zipcode']
```

Este método se utiliza para visualizar los tipos de errores presentes en transacciones fraudulentas y no fraudulentas. En primer lugar, se crea un DataFrame utilizando el método `stack()`, que permite apilar categorías relacionadas con posibles errores en las transacciones, junto con el recuento de cuántas veces ocurre cada error. Posteriormente, se agrupan estos datos en función de si la transacción es fraudulenta o no, lo que facilita la comparación y la identificación de patrones en los errores asociados a ambos tipos de transacciones.

**Figura 73 – Cantidad de transacciones fraudulentas y no fraudulentas por error**



Se puede observar que la mayoría de las veces los errores que más se repiten para transacciones fraudulentas y no fraudulentas, son “insufficient\_balance” y “bad\_pin”.

Se procede a hacer un análisis de las variables categóricas *use\_chip* y *merchant\_city*.

*Figura 74 – Distribución de los valores de la variable use\_chip*

```
2   stg_transactions.groupBy("use_chip").agg(count("*").alias("count")).orderBy(
3     desc("count")
4   ).display()
```

► (4) Spark Jobs

Table ▾ +

	use_chip	count
1	Swipe Transaction	15386030
2	Chip Transaction	6287584
3	Online Transaction	2713220

*Figura 75 - Distribución de los valores de la variable merchant\_city*

```
1  # Distribución de los valores de la variable merchant_city.
2  stg_transactions.groupBy("merchant_city").agg(count("*").alias("count")).orderBy(
3    desc("count")
4  ).display()
```

► (4) Spark Jobs

Table ▾ +

	merchant_city	count
1	ONLINE	2720821
2	Houston	246036
3	Los Angeles	180496
4	Miami	178653
5	Brooklyn	155425
6	Chicago	136165

La cantidad de veces que una transacción es parte de la ciudad "ONLINE", no coincide con la cantidad de veces que la variable *use\_chip* es "Online Transaction". Por ello procedemos a hacer un análisis para determinar si existen problemas en la calidad de datos y cómo solucionarlo. Se

filtran todas aquellas transacciones con ciudad ONLINE que no tengan *use\_chip* como Online Transaction.

**Figura 76 – Transacciones con merchant\_city online y use\_chip diferente de online.**

```

1 stg_transactions.filter(
2   "merchant_city = 'ONLINE' and use_chip != 'Online Transaction'"
3 ).select(
4   "user",
5   "card",
6   "transaction_date",
7   "merchant_city",
8   "use_chip",
9   "is_fraud",
10  "merchant_state",
11  "zip",
12 ).display()

```

▶ (3) Spark Jobs

	user	card	transaction_date	merchant_city	use_chip	is_fraud	merchant_state	zip
1	606	2	2017-12-11	ONLINE	Chip Transaction	0	null	null
2	96	1	2015-11-18	ONLINE	Chip Transaction	0	null	null
3	111	1	2017-02-20	ONLINE	Chip Transaction	0	null	null
4	377	3	2017-04-22	ONLINE	Chip Transaction	0	null	null
5	449	2	2016-04-07	ONLINE	Chip Transaction	0	null	null
6	155	4	2015-10-30	ONLINE	Chip Transaction	0	null	null

Se visualiza que los campos de *state* y *zip* que corresponden a la ubicación de la transacción, también están vacíos y que por lógica no se puede utilizar el chip en una transacción en línea. Por lo que este se considera un error de calidad de datos. Se procede a cambiar *use\_chip* por Online Transaction para todas aquellas que no tengan una ciudad, estado o zip específicos.

**Figura 77 – Solución a calidad de datos en la variable use\_chip**

```

1 # Filtrar las filas y actualizar la columna use_chip
2 stg_transactions = stg_transactions.withColumn(
3   "use_chip",
4   when(
5     (stg_transactions["merchant_city"] == "ONLINE")
6     & (stg_transactions["use_chip"] != "Online Transaction"),
7     "Online Transaction",
8   ).otherwise(stg_transactions["use_chip"]),
9 )

```

Se pueden analizar los datos para la ciudad y el estado de las transacciones. Se inicia revisando si todas las transacciones ONLINE (merchant\_city) tienen estado (merchant\_state) null y viceversa. Como todos los estados null pertenecen a compras online, se hará una imputación de datos se cambiará el valor null por ONLINE también.

**Figura 78 – Imputación de datos para la variable merchant\_state**

```
1 stg_transactions.filter(col("merchant_city") == "ONLINE").select(
2   | "merchant_city", "merchant_state"
3 ).distinct().display()
```

▶ (2) Spark Jobs

Table +

	merchant_city	merchant_state
1	ONLINE	null

↓ 1 row | 2.28 seconds runtime

Command took 2.28 seconds -- by sarychamsf@rumpjoneshotmail.onmicrosoft.com at 12/10/2023, 8:28:5

Cmd 84

```
1 stg_transactions.filter(col("merchant_state").isNull()).select(
2   | "merchant_city", "merchant_state"
3 ).distinct().display()
```

▶ (2) Spark Jobs

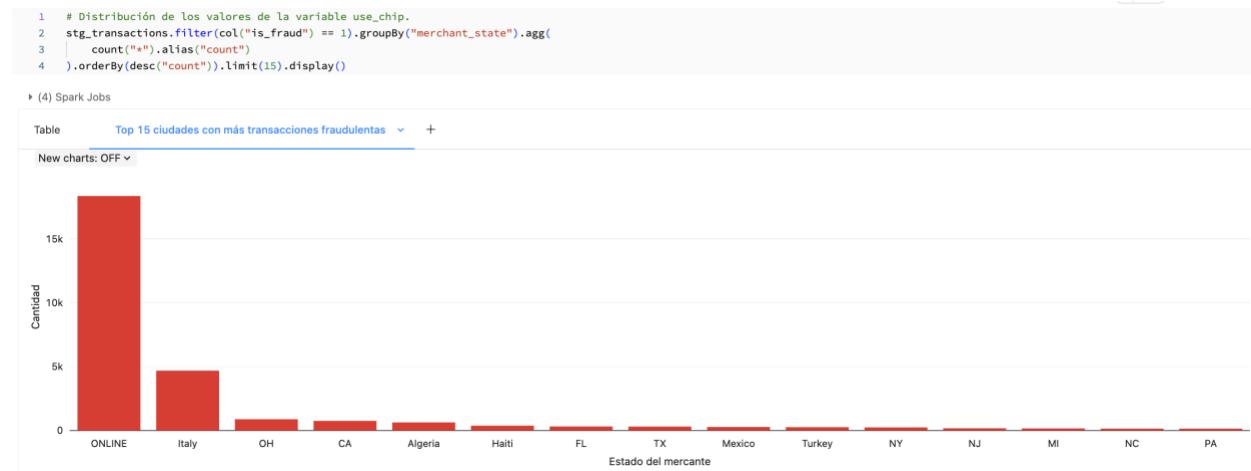
Table +

	merchant_city	merchant_state
1	ONLINE	null

↓ 1 row | 1.90 seconds runtime

```
1 # Se eemplazan los valores nulos en "merchant_state" con "ONLINE"
2 stg_transactions = stg_transactions.withColumn(
3   | "merchant_state",
4   | when(stg_transactions["merchant_state"].isNull(), "ONLINE").otherwise(
5   |   stg_transactions["merchant_state"]
6   ),
7 )
```

*Figura 79 – Diagrama de barras: transacciones fraudulentas por merchant\_state*



Es evidente que la mayoría de las transacciones fraudulentas ocurren en plataformas en línea, y además, se observa que algunas de ellas se llevan a cabo en países diferentes a Estados Unidos, a pesar de que los titulares de las tarjetas de crédito del conjunto de datos residen en este país. También es importante destacar que se registran casos de fraude incluso dentro de Estados Unidos. Se eliminan algunas variables innecesarias.

*Figura 80 - Eliminación de variables innecesarias en la tabla de transacciones*

```
1 # Se eliminan algunas variables innecesarias
2 stg_transactions = stg_transactions.drop(*["merchant_name", "mcc", "time", "zip"])
```

Se muestra el esquema final del dataset luego de hacer todas las transformaciones y luego se escribe en una tabla final de staging.

**Figura 81 – Esquema final de la tabla de transacciones en staging**

```

user: string (nullable = true)
card: string (nullable = true)
amount: decimal(10,2) (nullable = true)
use_chip: string (nullable = true)
merchant_city: string (nullable = true)
merchant_state: string (nullable = true)
is_fraud: integer (nullable = true)
transaction_date: string (nullable = true)
mcc_category: string (nullable = true)
technical_glitch: integer (nullable = true)
insufficient_balance: integer (nullable = true)
bad_card_number: integer (nullable = true)
bad_pin: integer (nullable = true)
bad_cvv: integer (nullable = true)
bad_expiration: integer (nullable = true)
bad_zipcode: integer (nullable = true)
card_brand: string (nullable = true)
card_type: string (nullable = true)
has_chip: integer (nullable = true)
credit_limit: decimal(10,2) (nullable = true)
anios_antiguedad: integer (nullable = true)

```

#### 4.2.2. Tarjetas

Primero, se hace la lectura de la tabla creada en el paso de extracción de datos y se muestran los registros. Spark suele adicionar por defecto una partición al dataset que cree que puede optimizar su uso. En este caso, como el dataset es pequeño, no se aplicará ninguna repartición nueva o adicional.

**Figura 82 – Visualización del dataset de tarjetas (cards)**

```

1  # Se lee la tabla raw.
2  raw_cards = spark.read.table(f"{{database}}.raw_cards")

> raw_cards: pyspark.sql.dataframe.DataFrame = [user:integer, card_index:integer ... 11 more fields]
Command took 0.28 seconds -- by sarychansf@rumpjoneshotmail.onmicrosoft.com at 12/18/2023, 9:18:41 p. m. on Sara Chamseddine's Personal Compute Cluster
Cnd 8

1  raw_cards.display()

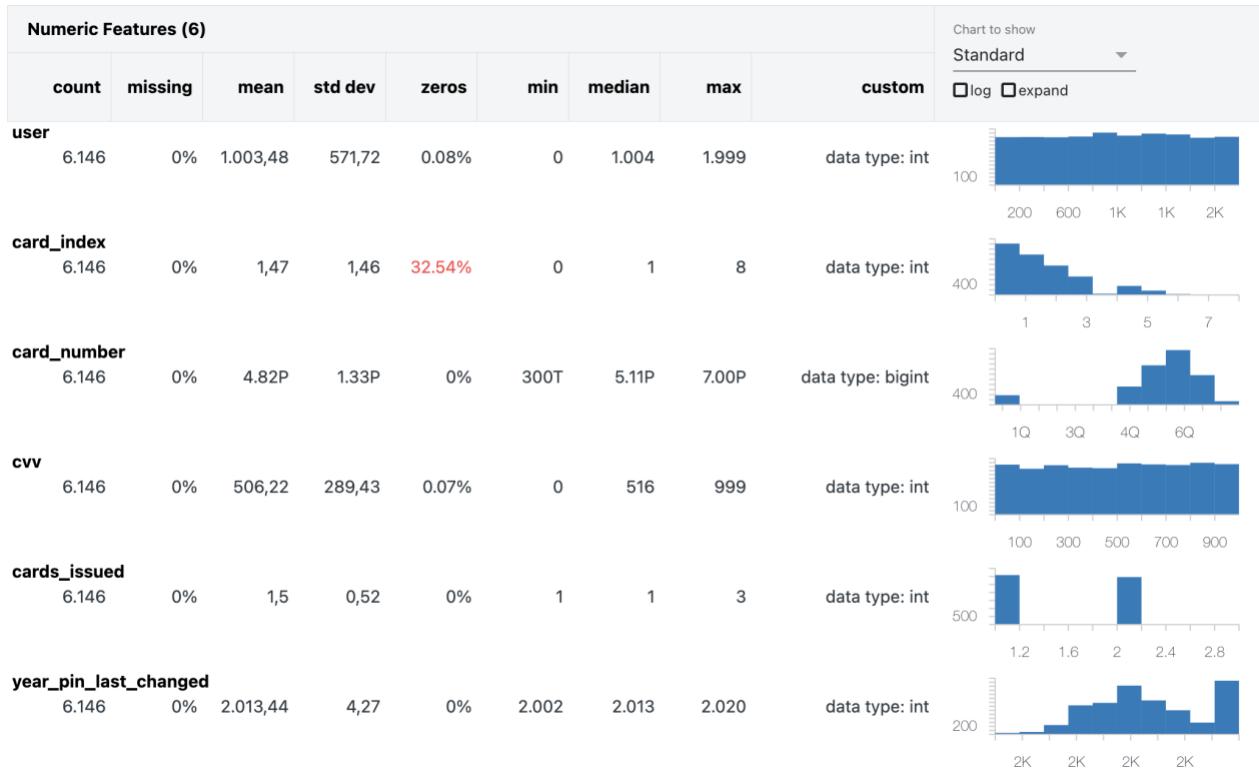
> (2) Spark Jobs
Table + New result t

```

	user	card_index	card_brand	card_type	card_number	expires	cvv	has_chip	cards_issued	credit_limit	acct_open_date	year_pin_last_changed	card_on_dark_web
1	0	0	Visa	Debit	4344676511950444	12/2022	623	YES	2	\$24295	09/2002	2008	No
2	0	1	Visa	Debit	4956965974959986	12/2020	393	YES	2	\$21968	04/2014	2014	No
3	0	2	Visa	Debit	4582313478255491	02/2024	719	YES	2	\$46414	07/2003	2004	No
4	0	3	Visa	Credit	4879494103069057	08/2024	693	NO	1	\$12400	01/2003	2012	No
5	0	4	Mastercard	Debit (Prepaid)	5722874738736011	03/2009	75	YES	1	\$28	09/2008	2009	No
6	1	0	Visa	Credit	4404898874682993	09/2003	736	YES	1	\$27500	09/2003	2012	No

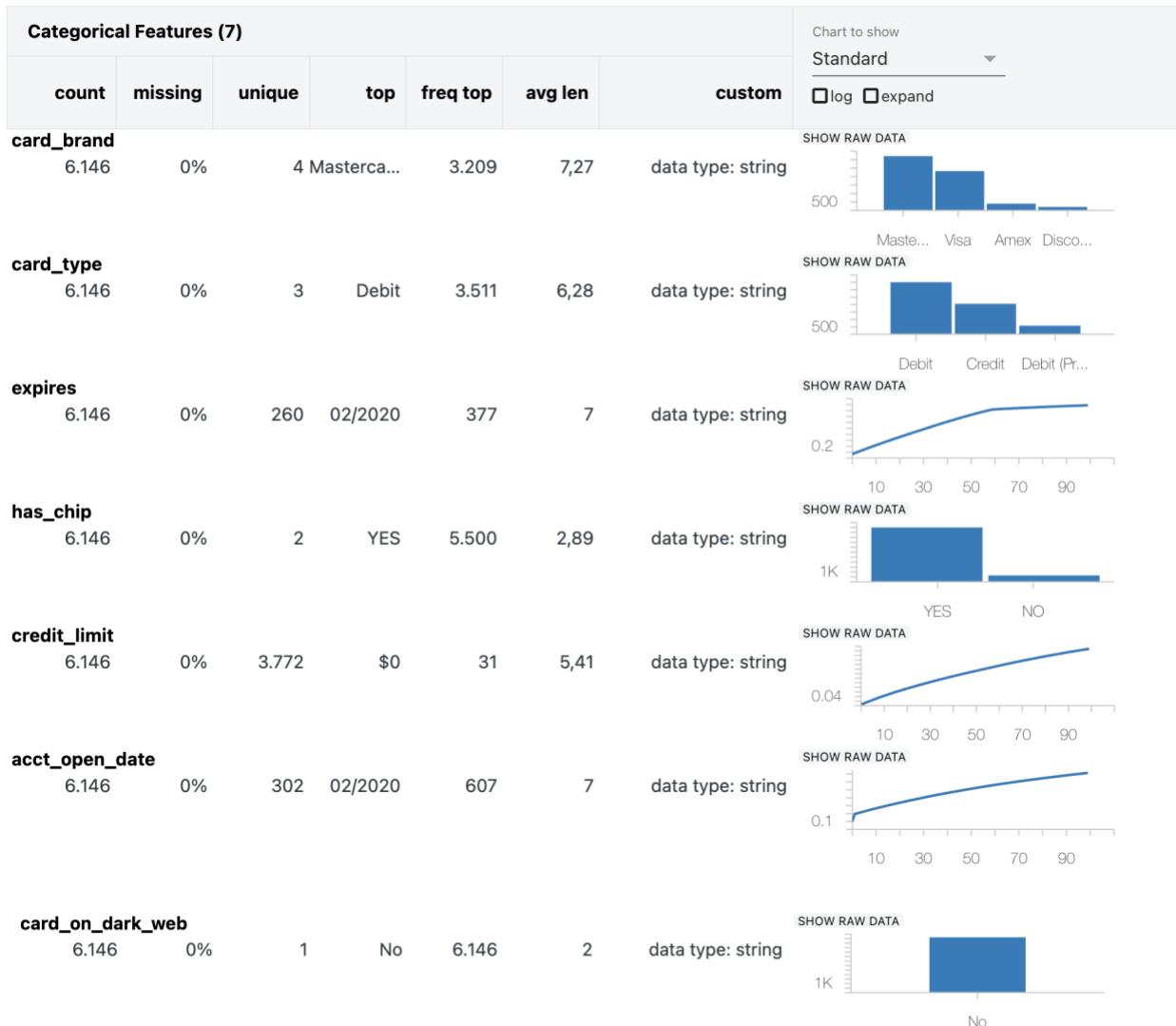
Se hace uso de la opción de visualizaciones de Databricks para crear un Perfil de datos.

**Figura 83 – Perfil de datos numéricos para tabla de tarjetas**



- Muchas de las variables numéricas realmente son categóricas, habrá que darles un tratamiento específico.
- No hay valores faltantes en ninguna columna.
- La presencia de ceros se debe al ID de las tarjetas.
- Vemos que hay muchas variables que contienen datos personales que no se necesitan, como por ejemplo: cvv, expires y card\_number.

*Figura 84 - Perfil de datos categóricos para tabla de tarjetas*

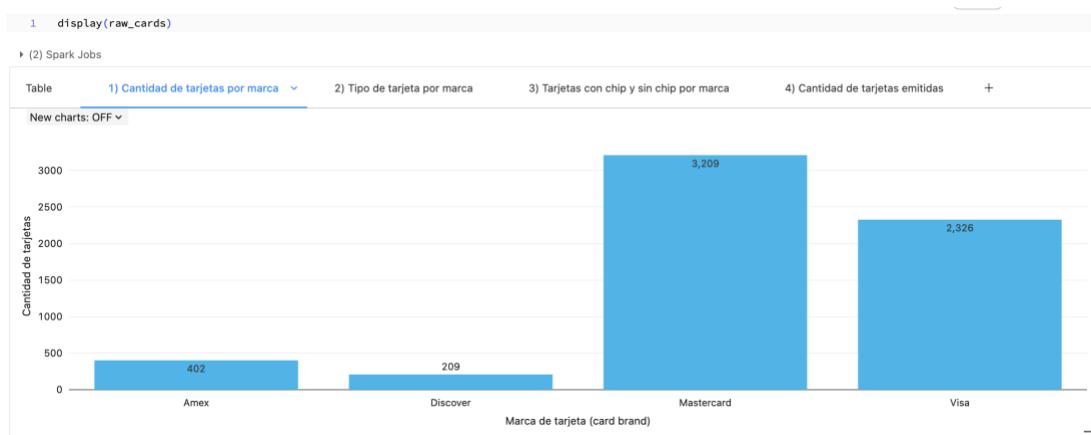


- La mayoría de tarjetas utilizadas para transacciones son Mastercard, débito y utilizan chip.
- Ninguna tarjeta se encuentra en la dark web.
- La columna credit\_limit debería ser numérica, por eso será necesario hacer una transformación de datos.

Se proceden a realizar algunas visualizaciones de los datos iniciales.

- **Cantidad de tarjetas por marca:** diagrama de barras que muestra cuántas tarjetas en el dataset pertenecen a cada tipo de marca.
- **Tipo de tarjeta por marca:** porcentaje de cada tipo de tarjeta por marca.
- **Tarjetas con chip y sin chip por marca:** muestra el porcentaje de tarjetas que tienen chip y las que no por cada marca.
- **Cantidad de tarjetas emitidas:** muestra la cantidad de tarjetas por las veces que ha sido emitida.

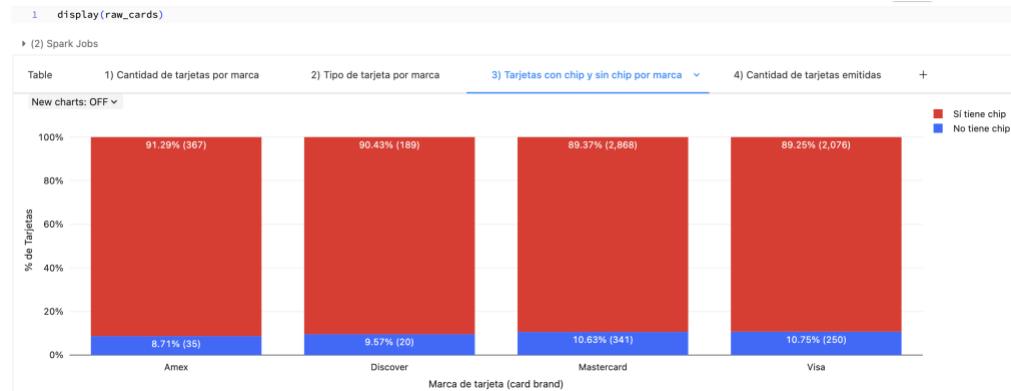
**Figura 85 - Diagrama de barras: cantidad de tarjetas por marca**



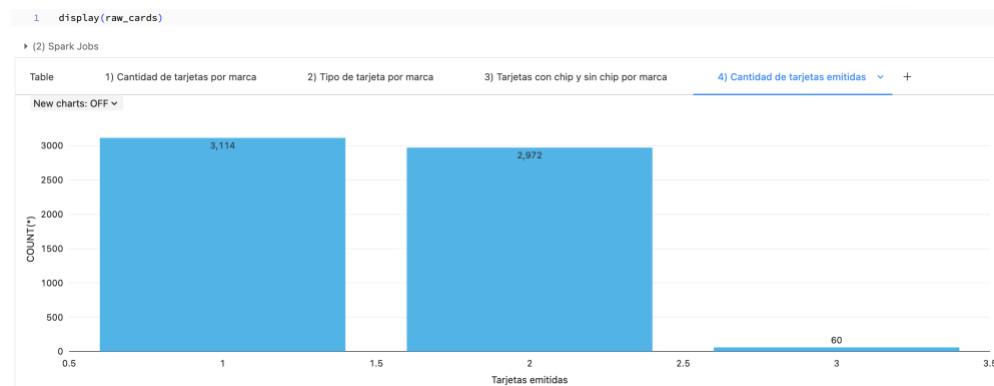
**Figura 86 – Diagrama de barras: tipo de tarjeta por marca**



**Figura 87 - Diagrama de barras: tarjetas con chip y sin chip por marca**



**Figura 88 - Diagrama de barras: cantidad de tarjetas emitidas**



Algunos tipos de datos pueden no ser la elección adecuada para representar su contenido, y es posible que ciertas columnas en la base de datos carezcan de relevancia. Con el objetivo de simplificar el análisis del conjunto de datos a través de visualizaciones y para determinar si se necesitan mejoras en la calidad de los datos, se realizan validaciones y transformaciones básicas. Además, se llevarán a cabo más visualizaciones para profundizar en la comprensión del dataset.

Primero se eliminan duplicados, si los hay:

**Figura 89 – Eliminación de duplicados en la tabla de tarjetas**

```

1 # Contar la cantidad total de tarjetas.
2 cant_cards = raw_cards.count()
3 print(cant_cards)

▶ (3) Spark Jobs
6146
Command took 45.36 seconds -- by sarychamsf@rumpjoneshotmail.onmicrosoft.com at 12/10/2023, 9:25:54 p. m. on Sara Cham
Cmd 27

1 # Validar si hay tarjetas duplicadas y eliminarlas.
2 num_duplicates = cant_cards - raw_cards.dropDuplicates().count()
3
4 if num_duplicates > 0:
5     raw_transactions = raw_cards.dropDuplicates()
6
7 print(
8     f"Duplicados encontrados y eliminados de la tabla de tarjetas: {num_duplicates}"
9 )

```

▶ (3) Spark Jobs

Duplicados encontrados y eliminados de la tabla de tarjetas: 0

Se muestra el esquema del DataFrame para analizar qué tipos de datos deben ser cambiados y qué columnas eliminadas.

**Figura 90 – Esquema inicial de la tabla de tarjetas**

```

2 raw_cards.printSchema()

root
|-- user: integer (nullable = true)
|-- card_index: integer (nullable = true)
|-- card_brand: string (nullable = true)
|-- card_type: string (nullable = true)
|-- card_number: long (nullable = true)
|-- expires: string (nullable = true)
|-- cvv: integer (nullable = true)
|-- has_chip: string (nullable = true)
|-- cards_issued: integer (nullable = true)
|-- credit_limit: string (nullable = true)
|-- acct_open_date: string (nullable = true)
|-- year_pin_last_changed: integer (nullable = true)
|-- card_on_dark_web: string (nullable = true)

```

Se eliminarán las columnas que no serán utilizadas en el modelo, como aquellas que incluyen datos personales y que no proporcionan información valiosa.

**Figura 91 – Eliminación de columnas innecesarias de la tabla de tarjetas**

```

1  stg_cards = raw_cards.drop(*["card_number", "expires", "cvv", "card_on_dark_web", "cards_issued"])
2  stg_cards.printSchema()

▶ [stg_cards: pyspark.sql.dataframe.DataFrame = [user: integer, card_index: integer ... 6 more fields]
root
|-- user: integer (nullable = true)
|-- card_index: integer (nullable = true)
|-- card_brand: string (nullable = true)
|-- card_type: string (nullable = true)
|-- has_chip: string (nullable = true)
|-- credit_limit: string (nullable = true)
|-- acct_open_date: string (nullable = true)
|-- year_pin_last_changed: integer (nullable = true)

```

Se procede a reemplazar la variable has\_chip por ceros y unos, para tratarla como un número entero y no como una variable categórica, esto será útil al momento de implementar el modelo.

**Figura 92 – Reemplazo de la variable has\_chip por ceros y unos**

```

1  # Reemplazar "yes" con 1 y "no" con 0 en la columna 'has_chip'
2  stg_cards = stg_cards.withColumn(
3      "has_chip",
4      when(col("has_chip") == "YES", 1)
5      .when(col("has_chip") == "NO", 0)
6      .otherwise(-1)
7  )
8
9  stg_cards.display()

```

▶ (2) Spark Jobs  
▶ [stg\_cards: pyspark.sql.dataframe.DataFrame = [user: integer, card\_index: integer ... 6 more fields]

Table										
user	card_index	card_brand	card_type	has_chip	credit_limit	acct_open_date	year_pin_last_changed			
1	0	1	Visa	Debit	1	\$21968	04/2014	2014		
2	1	1	Visa	Debit	1	\$28508	02/2011	2011		
3	2	1	Mastercard	Debit	0	\$27480	03/2002	2008		
4	3	1	Mastercard	Debit (Prepaid)	1	\$62	02/2007	2007		
5	5	1	Visa	Debit	1	\$21587	09/2007	2007		
6	6	1	Mastercard	Credit	1	\$11200	09/2010	2010		

Se cambia también el tipo de dato de la columna credit limit para que sea numérica en vez de categórica, y eliminando a su vez el signo \$.

**Figura 93 – Transformación de la columna credit\_limit de la tabla de tarjetas**

```

3  stg_cards = stg_cards.withColumn(
4      "credit_limit",
5      (regexp_replace(stg_cards["credit_limit"], "\$\s", "").cast(DecimalType(10, 2)))
6  )
7
8  display(stg_cards)

```

▶ (2) Spark Jobs  
▶ stg\_cards: pyspark.sql.dataframe.DataFrame = [user: integer, card\_index: integer ... 6 more fields]

Table +

	user	card_index	card_brand	card_type	has_chip	credit_limit	acct_open_date	year_pin_last_changed
1	0	1	Visa	Debit	1	21968.00	04/2014	2014
2	1	1	Visa	Debit	1	28508.00	02/2011	2011
3	2	1	Mastercard	Debit	0	27480.00	03/2002	2008
4	3	1	Mastercard	Debit (Prepaid)	1	62.00	02/2007	2007
5	5	1	Visa	Debit	1	21587.00	09/2007	2007
6	6	1	Mastercard	Credit	1	11200.00	09/2010	2010

Las variables de acct\_open\_date y year\_pin\_last\_changed pueden ser útiles para feature engineering posteriormente, por ello se mantendrán, pero se cambiará su formato.

**Figura 94 – Cambio de formato para las columnas acct\_open\_date y year\_pin\_last\_changed**

```

1  # Convertir la columna "acct_open_date" a formato de fecha
2  stg_cards = stg_cards.withColumn("acct_open_date", to_date("acct_open_date", "MM/yyyy"))
3
4  # Formatear la columna "fecha" en estilo "año-mes"
5  stg_cards = stg_cards.withColumn("acct_open_date", date_format("acct_open_date", "yyyy-MM"))
6
7  display(stg_cards)

```

▶ (2) Spark Jobs  
▶ stg\_cards: pyspark.sql.dataframe.DataFrame = [user: integer, card\_index: integer ... 6 more fields]

Table +

	user	card_index	card_brand	card_type	has_chip	credit_limit	acct_open_date	year_pin_last_changed
1	0	1	Visa	Debit	1	21968.00	2014-04	2014
2	1	1	Visa	Debit	1	28508.00	2011-02	2011
3	2	1	Mastercard	Debit	0	27480.00	2002-03	2008
4	3	1	Mastercard	Debit (Prepaid)	1	62.00	2007-02	2007
5	5	1	Visa	Debit	1	21587.00	2007-09	2007
6	6	1	Mastercard	Credit	1	11200.00	2010-09	2010

Se cambia el nombre de la columna de card\_index para que coincida con el de transacciones y el join que se realizará posteriormente sea más sencillo.

*Figura 95 – Renombramiento de la columna card\_index*

```
1 stg_cards = stg_cards.withColumnRenamed("card_index", "card")  
▶ stg_cards: pyspark.sql.dataframe.DataFrame = [user: integer, card: integer ... 6 more fields]
```

Se muestra el esquema final luego de todas las transformaciones implementadas y se guarda la tabla en staging.

*Figura 96 – Esquema final de la tabla de tarjetas*

```
1   stg_cards.printSchema()
```

root

```
|-- user: integer (nullable = true)
|-- card: integer (nullable = true)
|-- card_brand: string (nullable = true)
|-- card_type: string (nullable = true)
|-- has_chip: integer (nullable = false)
|-- credit_limit: decimal(10,2) (nullable = true)
|-- acct_open_date: string (nullable = true)
|-- year_pin_last_changed: string (nullable = true)
```

**Figura 97 – Guardar la tabla de tarjetas en staging**

```
1 stg_cards.write.partitionBy("card").format("delta").mode("overwrite").option(  
2   "overwriteSchema", True  
3 ).saveAsTable(f"{{database}}.stg_cards")
```

Se valida que la tabla haya cargado haciendo una consulta a ella con SQL.

*Figura 98 – Validación de la creación de la tabla de tarjetas*

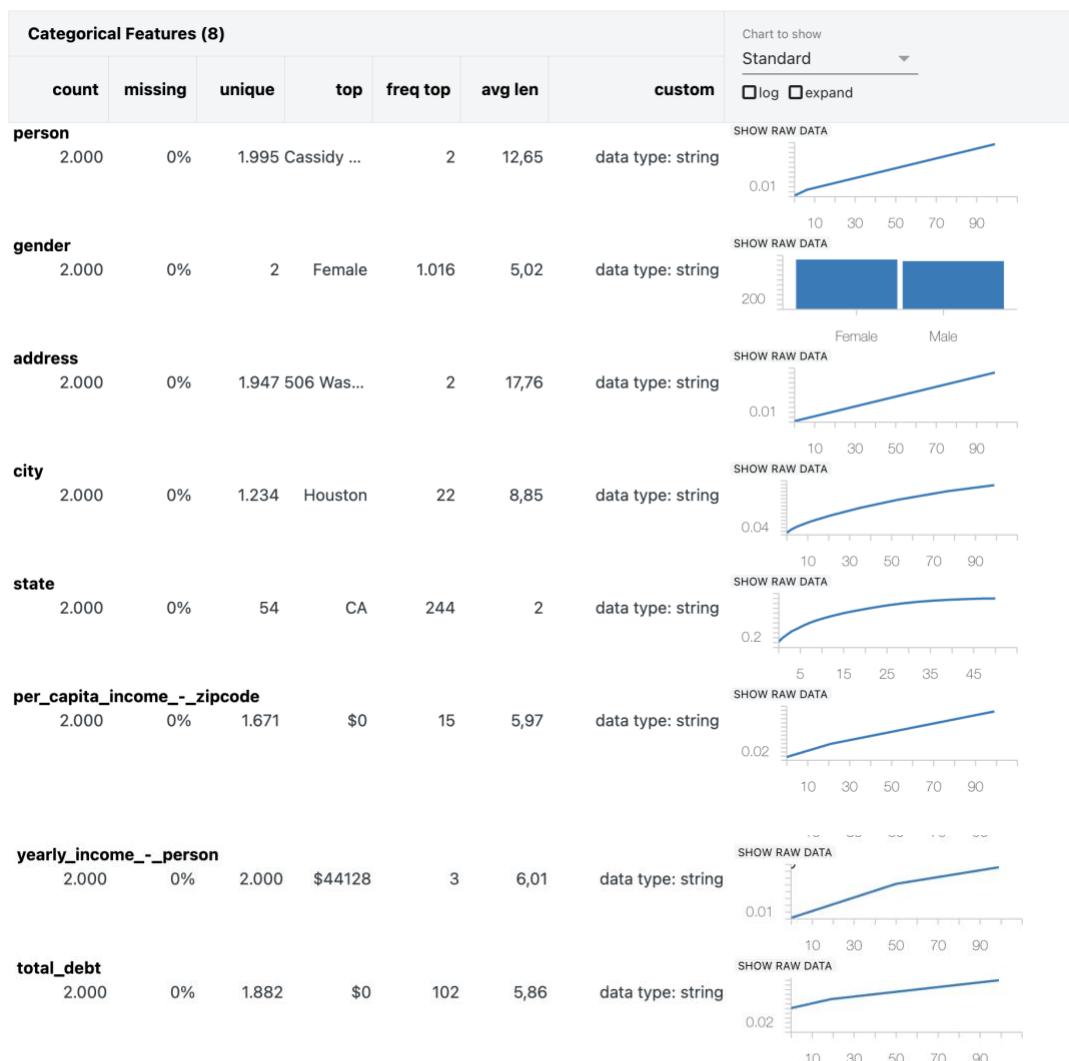
	user	card	card_brand	card_type	has_chip	credit_limit	acct_open_date	year_pin_last_changed
1	0	0	Visa	Debit	1	24295.00	2002-09	2002
2	1	0	Visa	Credit	1	27500.00	2003-09	2003
3	2	0	Mastercard	Debit	1	31599.00	2009-10	2009
4	3	0	Visa	Credit	1	98100.00	2011-01	2011
5	4	0	Mastercard	Debit	1	34900.00	1999-12	1999
6	5	0	Visa	Credit	1	9900.00	2002-01	2002

### 4.2.3. Usuarios

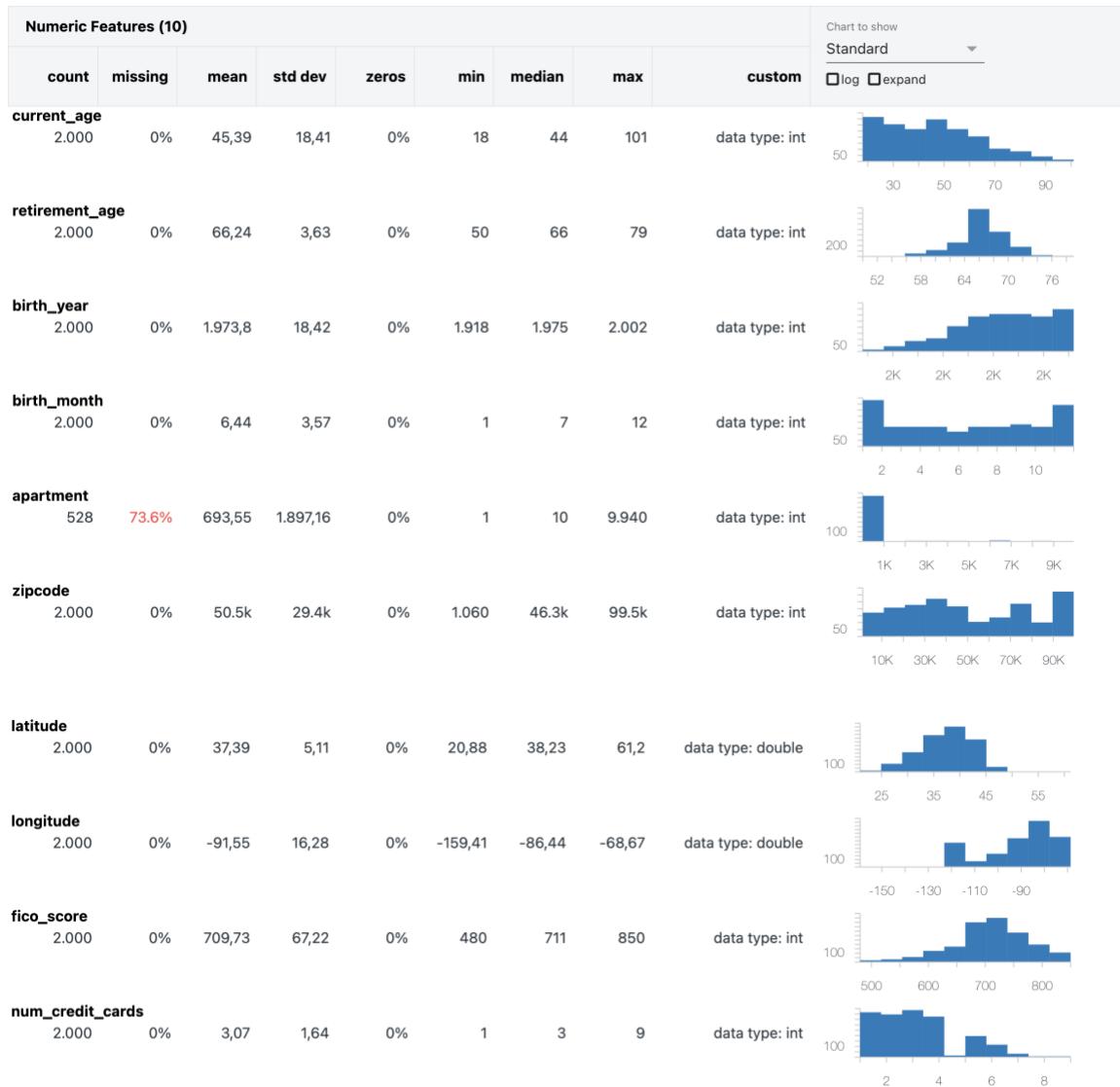
En este caso, como el dataset es pequeño, no se aplicará ninguna repartición nueva o adicional.

Luego se procede muestra parte de del contenido de la tabla de tarjetas para visualizar la forma de los datos utilizando la opción de "Data Profile" que Databricks posee para hacer un análisis de los datos de la tabla.

**Figura 99 – Perfil de datos para las columnas categóricas de la tabla de usuarios.**



*Figura 100 – Perfil de datos para las columnas numéricas de la tabla de usuarios.*

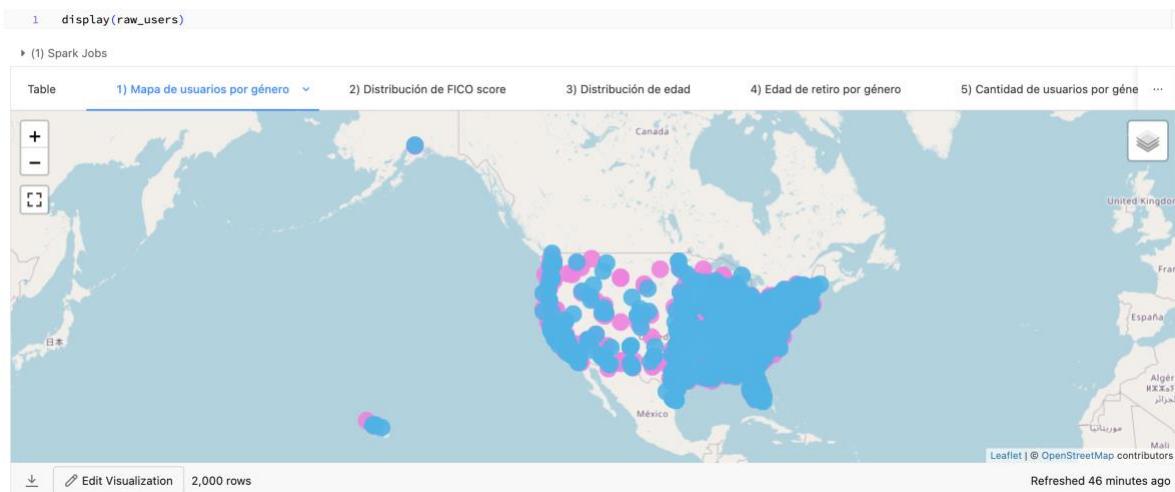


Se proceden a realizar algunas visualizaciones para hacer análisis exploratorio.

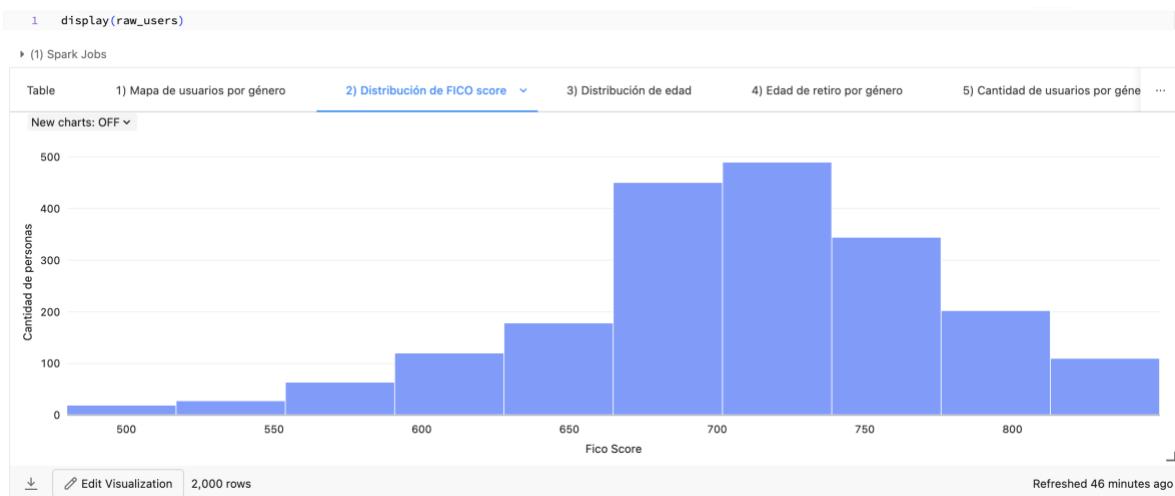
- **Mapa de usuarios por género:** Muestra la ubicación geográfica de los usuarios del dataset, marcando con el color su género. Se puede observar que todos los usuarios del dataset viven en Estados Unidos.
- **Distribución de FICO Score:** Distribución del FICO Score (puntaje de crédito) para los usuarios del dataset. Se observa que la mayoría tienen un puntaje entre 665 y 739.

- **Distribución de edad:** distribución de edad de los usuarios del dataset. La mayoría de los usuarios son menores a 68 años, sin embargo son mayores los usuarios entre 18 y 26.
- **Edad de retiro por género:** muestra la distribución de edad de retiro por género. La mayoría de los usuarios se retiran entre los 65 y los 67 años de edad.
- **Cantidad de usuarios por género:** se muestra la cantidad de usuarios que tiene el dataset por género. Se observa que hay ligeramente más mujeres que hombres.

**Figura 101 - Mapa de usuarios por género**



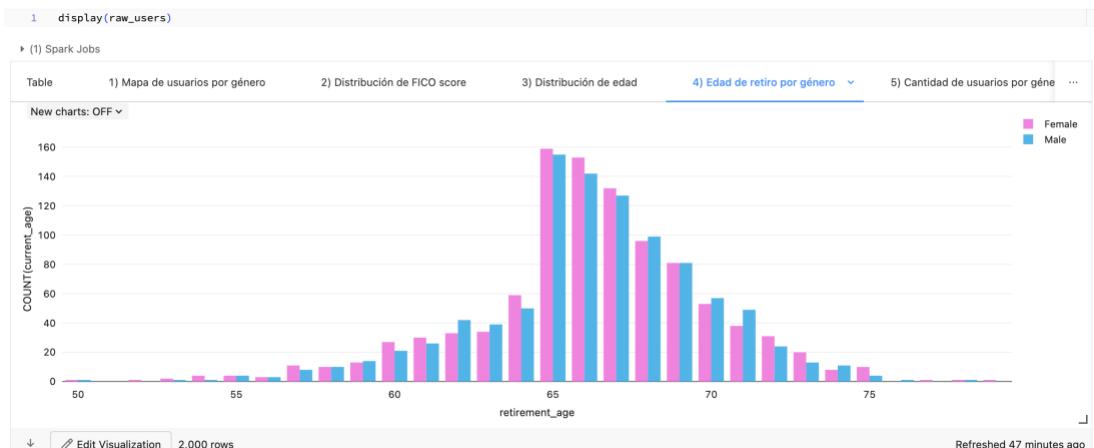
**Figura 102 – Distribución de FICO Score para los usuarios**



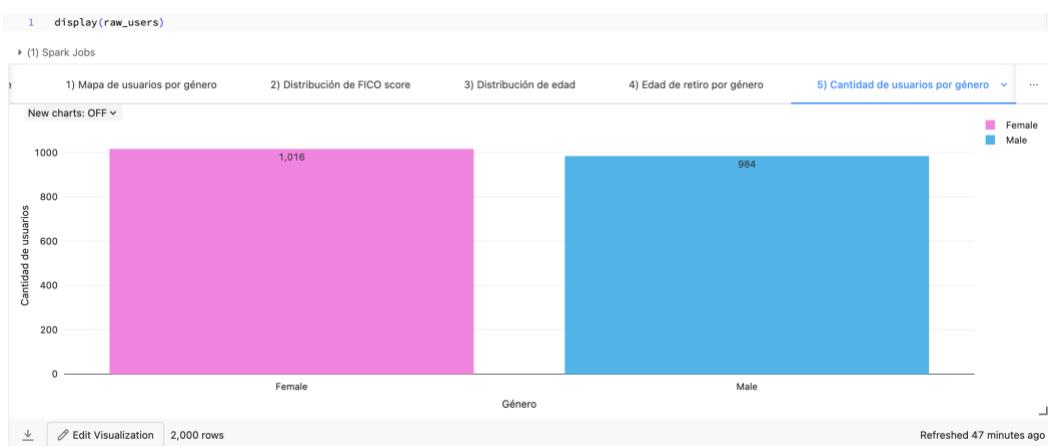
**Figura 103 – Distribución de edad de los usuarios**



**Figura 104 – Distribución de la edad de retiro por género**



**Figura 105 – Cantidad de usuarios por género**



En lo que respecta a los usuarios, se ha determinado que la información personal de un usuario no debería tener influencia en la detección de la autenticidad de una transacción. Por esta razón, no se incorpora esta información en el modelo, evitando así cualquier posible sesgo.

#### 4.2.4. Tabla final para el modelo

Para utilizar al momento de crear los modelos, se utilizará una única tabla con toda la información relevante. Para ello se utilizarán las tablas de transacciones y de tarjetas. Se dejará por fuera la información personal del usuario. Siendo así, se leen ambas tablas y se hace join de las mismas.

**Figura 106 – Lectura de tabla de staging de transacciones**

```
2   stg_transactions = spark.read.table(f"database.stg_transactions")
3
4   stg_transactions.display()
```

▶ (3) Spark Jobs  
▶ stg\_transactions: pyspark.sql.DataFrame = [user: integer, card: integer ... 14 more fields]

	user	card	transaction_date	amount	merchant_city	merchant_state	use_chip	mcc_category
1	932	1	2011-02-19	42.93	Upper Marlboro	MD	Swipe Transaction	Taxis and Limousines
2	932	1	2011-04-05	30.38	Upper Marlboro	MD	Swipe Transaction	Grocery Stores, Supermarkets
3	932	1	2011-05-04	57.69	Upper Marlboro	MD	Swipe Transaction	Taxis and Limousines
4	932	1	2011-08-20	8.38	Lothian	MD	Swipe Transaction	Drinking Places (Alcoholic Beverages), Bars, Taverns,
5	932	1	2011-08-31	8.82	Lothian	MD	Swipe Transaction	Drinking Places (Alcoholic Beverages), Bars, Taverns,
6	932	1	2011-11-15	46.57	Upper Marlboro	MD	Swipe Transaction	Taxis and Limousines

**Figura 107 - Lectura de tabla de staging de tarjetas**

```
1   # Se lee la tabla raw.
2   stg_cards = spark.read.table(f"database.stg_cards")
3
4   stg_cards.display()
```

▶ (2) Spark Jobs  
▶ stg\_cards: pyspark.sql.DataFrame = [user: integer, card: integer ... 6 more fields]

	user	card	card_brand	card_type	has_chip	credit_limit	acct_open_date	year_pin_last_changed
1	0	0	Visa	Debit	1	24295.00	2002-09	2002
2	1	0	Visa	Credit	1	27500.00	2003-09	2003
3	2	0	Mastercard	Debit	1	31599.00	2009-10	2009
4	3	0	Visa	Credit	1	98100.00	2011-01	2011
5	4	0	Mastercard	Debit	1	34900.00	1999-12	1999
6	5	0	Visa	Credit	1	9900.00	2002-01	2002

**Figura 108 – Join entre las tablas de transacciones y tarjetas**

```

1 prod_transactions = stg_transactions.join(
2     broadcast(stg_cards), on=["user", "card"], how="inner"
3 )
4
5 display(prod_transactions)

```

▶ (2) Spark Jobs

▶ prod\_transactions: pyspark.sql.dataframe.DataFrame [user: integer, card: integer ... 20 more fields]

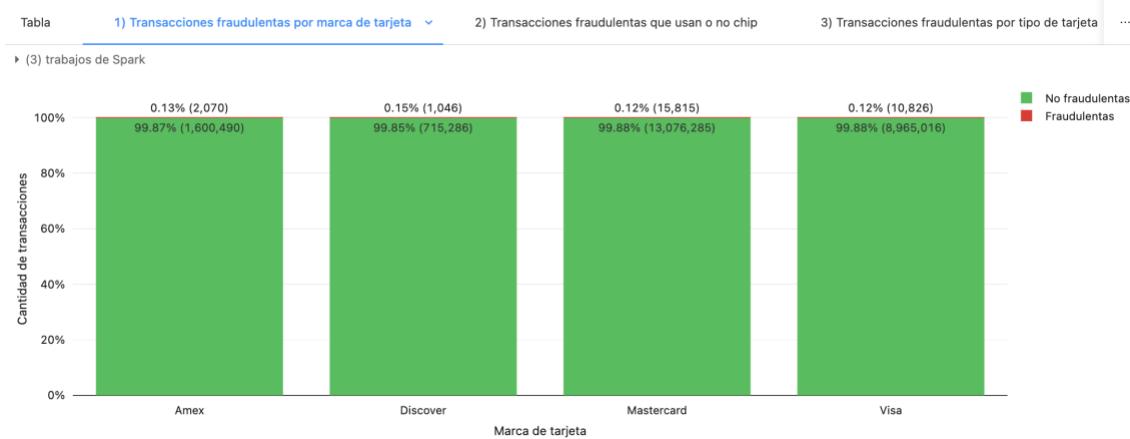
Table + New result table: OFF ▾

user	card	transaction_date	amount	merchant_city	merchant_state	use_chip	mcc_category	
1	1036	2	2013-05-13	69.28	Far Rockaway	NY	Swipe Transaction	Automotive Service Shops
2	1036	2	2013-11-26	2.05	Lincoln	KS	Swipe Transaction	Fast Food Restaurants
3	1036	2	2014-03-16	2.28	Bennett	CO	Swipe Transaction	Service Stations ( with or without ancillary services)
4	1036	2	2014-11-25	68.85	Lexington	SC	Swipe Transaction	Automotive Service Shops
5	1036	2	2016-07-30	158.34	Woodmere	NY	Chip Transaction	Wholesale Clubs
6	1036	2	2017-01-15	19.29	Woodmere	NY	Chip Transaction	Discount Stores

Se proceden a realizar algunas visualizaciones de los datos iniciales para hacer análisis exploratorio. Esto debido a que ahora tenemos todos los datos en una misma tabla y se pueden descubrir cosas nuevas que relacionen ambas tablas.

- **Transacciones fraudulentas por marca de tarjeta:** muestra el porcentaje de transacciones fraudulentas por marca de tarjeta. Se utilizó escala logarítmica. La gráfica muestra que la mayor cantidad de transacciones fraudulentas se realizan con tarjetas Discover, seguida por Amex.
- **Transacciones fraudulentas que usan o no chip:** muestra el porcentaje de transacciones fraudulentas que usan chip y las que no usan chip. Se utilizó escala logarítmica. La gráfica muestra que la mayor cantidad de transacciones fraudulentas se realizan con tarjetas que sí tienen chip.
- **Transacciones fraudulentas por tipo de tarjeta:** muestra el porcentaje de transacciones fraudulentas por tipo de tarjeta utilizada para la transacción. Se utilizó escala logarítmica. La gráfica muestra que la mayor cantidad de transacciones fraudulentas se realizan con tarjetas de débito prepagadas, seguida de las transacciones con tarjeta de crédito, y por último, las de débito.

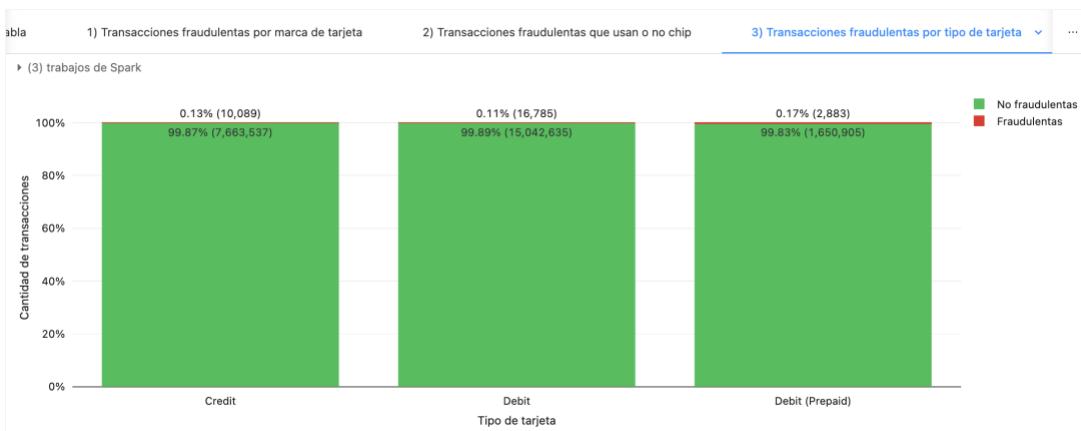
**Figura 109 – Diagrama de barras: transacciones fraudulentas por marca de tarjeta**



**Figura 110 – Diagrama de barras: transacciones fraudulentas que usan o no chip**



**Figura 111 – Diagrama de barras: transacciones fraudulentas por tipo de tarjeta**



Se crearán algunas features para enriquecer el dataset y facilitar el posterior entrenamiento del modelo. Por ejemplo, una puede ser la antigüedad del usuario al momento de realizar la transacción. Para calcular esta feature, se utilizarán los campos de transaction\_date y acct\_open\_date.

**Figura 112 – Cálculo de la antigüedad del usuario en años al momento de la transacción**

```

1 # Calcula la antigüedad del usuario al momento de la transacción en años
2 prod_transactions = prod_transactions.withColumn(
3     "anios_antiguedad", year("transaction_date") - year("acct_open_date")
4 )
5
6 prod_transactions.display()

```

user_id	card	transaction_date	amount	merchant	anios_antiguedad
1	2	2013-05-13	69.28	Far Rockaway	1
2	2	2013-11-26	2.05	Lincoln	1
3	2	2014-03-16	2.28	Bennett	2
4	2	2014-11-25	68.85	Lexington	2
5	2	2016-07-30	158.34	Woodmere	4
6	2	2017-01-15	19.29	Woodmere	5

También se calculan los años transcurridos desde la última vez que se cambió el PIN de la tarjeta y la transacción. Para esta feature, se utilizarán los campos de transaction\_date y year\_pin\_last\_changed. Luego, se eliminan las columnas innecesarias.

**Figura 113 - Cálculo de la antigüedad del usuario en años al momento de la transacción**

```

2 prod_transactions = prod_transactions.withColumn(
3     "anios_ultimo_cambio_pin", year("transaction_date") - year("year_pin_last_changed")
4 )
5
6 prod_transactions.display()

```

user_id	card	transaction_date	amount	merchant	anios_ultimo_cambio_pin
1	2	2013-05-13	69.28	Far Rockaway	1
2	2	2013-11-26	2.05	Lincoln	1
3	2	2014-03-16	2.28	Bennett	1
4	2	2014-11-25	68.85	Lexington	1
5	2	2016-07-30	158.34	Woodmere	2
6	2	2017-01-15	19.29	Woodmere	2

**Figura 114 – Eliminación de las columnas innecesarias del dataset final**

```
prod_transactions = prod_transactions.drop(*["year_pin_last_changed", "acct_open_date"])
```

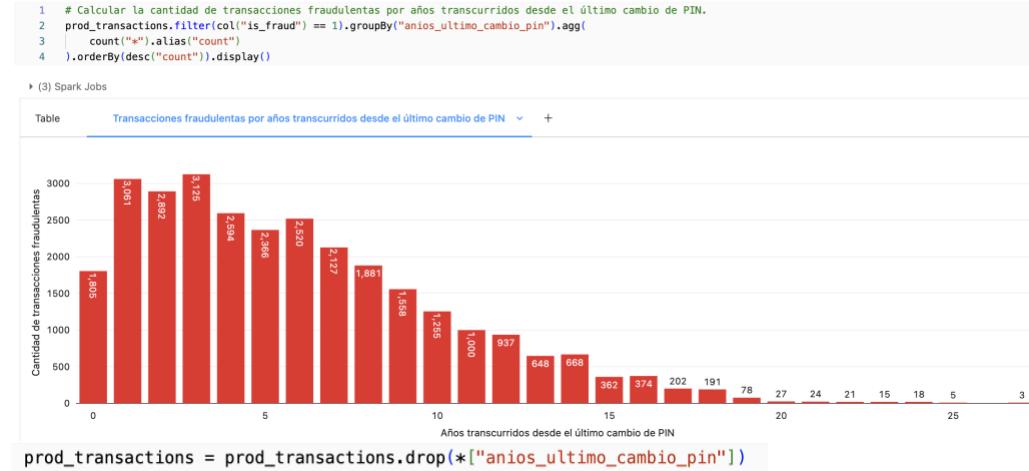
Se realiza una visualizaciones adicional para ver cómo se distribuye la cantidad de transacciones fraudulentas por antigüedad de usuario al momento de la transacción. Se puede observar que mientras los usuarios son más recientes, mayor es la probabilidad de fraude, que inicia aproximadamente luego del primer año a incrementar.

**Figura 115 – Cantidad de transacciones fraudulentas recibidas por cada año de antigüedad**



Se realiza también otra visualización con la cantidad de transacciones fraudulentas por años transcurridos desde el último cambio de PIN. Al analizar este conjunto de datos, se puede observar que los usuarios no han realizado cambios en sus PIN desde la creación de sus cuentas, ya que los valores de estas dos variables son idénticos. A partir de esta observación, se concluye que no tiene sentido utilizar la característica "años\_ultimo\_cambio\_pin", ya que no aporta ningún valor significativo a nuestro análisis.

**Figura 116 – Transacciones fraudulentas por años transcurridos desde el último cambio de PIN**



"Las columnas "user" y "card" se han convertido al tipo de dato string con el fin de tratarlas como variables categóricas en lugar de numéricas. Esto asegura que estas variables no influyan en el modelo con un peso numérico innecesario al ser introducidas en él. Además, dado que el modelo no puede manejar directamente variables de tipo date, se ha realizado una conversión a tipo string en la columna "transaction\_date" para adecuarla al formato compatible con el modelo."

**Figura 117 – Esquema final de la tabla de transacciones para el modelo.**

```

1 prod_transactions = prod_transactions.withColumn("user", col("user").cast("string"))
2 prod_transactions = prod_transactions.withColumn("card", col("card").cast("string"))
3 prod_transactions = prod_transactions.withColumn("transaction_date", col("transaction_date").cast("string"))
4 prod_transactions.printSchema()
5
6 prod_transactions: pyspark.sql.dataframe.DataFrame = [user: string, card: string ... 19 more fields]
7
8
9 |-- user: string (nullable = true)
10 |-- card: string (nullable = true)
11 |-- transaction_date: string (nullable = true)
12 |-- amount: decimal(10,2) (nullable = true)
13 |-- merchant_city: string (nullable = true)
14 |-- merchant_state: string (nullable = true)
15 |-- use_chip: string (nullable = true)
16 |-- mcc_category: string (nullable = true)
17 |-- technical_glitch: integer (nullable = true)
18 |-- insufficient_balance: integer (nullable = true)
19 |-- bad_card_number: integer (nullable = true)
20 |-- bad_pini: integer (nullable = true)
21 |-- bad_cvv: integer (nullable = true)
22 |-- bad_expiration: integer (nullable = true)
23 |-- bad_zipcode: integer (nullable = true)
24 |-- is_fraud: integer (nullable = true)
25 |-- card_brand: string (nullable = true)
26 |-- card_type: string (nullable = true)
27 |-- has_chip: integer (nullable = true)
28 |-- credit_limit: decimal(10,2) (nullable = true)
29 |-- anios_antiguedad: integer (nullable = true)

```

*Figura 118 – Escritura de la tabla final de transacciones en producción*

```
prod_transactions.write.format("delta").mode("overwrite").option(
    "overwriteSchema", True
).saveAsTable(f"{database}.prod_transactions")
```

### 4.3. Modelado de Aprendizaje Automático

Se llevaron a cabo diversas tareas que involucraron la creación de características, la aplicación de transformaciones, codificaciones, reducción de dimensiones y selección de variables. El proceso para obtener un modelo adecuado implicó probar distintas combinaciones de parámetros, modelos, codificadores, transformaciones y variables en distintos experimentos. Esto quiere decir que la creación del modelo es un proceso iterativo. Algunos modelos no exitosos aplicados consistieron en las siguientes transformaciones:

*Tabla 7 – Experimento #1*

<b>Particionamiento</b>	Particionamiento con una columna de salt generada con números aleatorios de una distribución uniforme.
<b>Transformaciones básicas adicionales</b>	Sin transformaciones básicas adicionales a las aplicadas en el análisis exploratorio y feature engineering.
<b>Codificador de variables</b>	StringIndexer.
<b>Selector de variables</b>	No aplicado.
<b>Reducción de dimensionalidad</b>	No aplicado.
<b>Unión de variables en un vector</b>	VectorAssembler.
<b>Separación train-test</b>	randomSplit de todo el dataset.
<b>Modelo</b>	LogisticRegression
<b>Ejecución del modelo</b>	ParamGridBuilder y CrossValidator.

El modelo tuvo un ROC muy alto, de aproximadamente 0.9. Sin embargo, predijo todos los valores como no fraude (falsos negativos). Debido al desbalance de clases, el modelo no fue capaz de detectar las anomalías. Es decir, la sensibilidad del modelo fue muy baja.

**Tabla 8 – Experimento #2**

<b>Particionamiento</b>	Particionamiento con una columna de salt generada con números aleatorios de una distribución uniforme.
<b>Transformaciones básicas adicionales</b>	Sin transformaciones básicas adicionales a las aplicadas en el análisis exploratorio y feature engineering.
<b>Codificador de variables</b>	StringIndexer
<b>Selector de variables</b>	ChiSqSelector, UnivariateFeatureSelector
<b>Reducción de dimensionalidad</b>	No aplicado.
<b>Unión de variables en un vector</b>	VectorAssembler
<b>Separación train-test</b>	randomSplit de todo el dataset.
<b>Modelo</b>	LogisticRegression
<b>Ejecución del modelo</b>	ParamGridBuilder y CrossValidator.

Para el experimento #2 no fue posible aplicar el selector de variable de ChiSqSelector, pues la variable merchant\_city cuenta con más de 10,000 y esto supera los límites del método para variables categóricas. Para el selector de variables UnivariateFeatureSelector, el modelo tuvo un ROC muy alto, de aproximadamente 0.9. Pero de nuevo, debido al desbalance de clases, el modelo no fue capaz de detectar las anomalías. Es decir, la sensibilidad del modelo fue muy baja.

**Tabla 9 – Experimento #3**

<b>Particionamiento</b>	Particionamiento con una columna de salt generada con números aleatorios de una distribución uniforme.
<b>Transformaciones básicas adicionales</b>	Sin transformaciones básicas adicionales a las aplicadas en el análisis exploratorio y feature engineering.

<b>Codificador de variables</b>	StringIndexer
<b>Selector de variables</b>	UnivariateFeatureSelector
<b>Reducción de dimensionalidad</b>	PCA
<b>Unión de variables en un vector</b>	VectorAssembler
<b>Separación train-test</b>	randomSplit de todo el dataset.
<b>Modelo</b>	LogisticRegression
<b>Ejecución del modelo</b>	ParamGridBuilder y CrossValidator.

Para el experimento #3 la combinación de funciones no permitió que la ejecución terminara, ni aumentando la cantidad de recursos del clúster.

**Tabla 10 – Experimento #4**

<b>Particionamiento</b>	Particionamiento con una columna de salt generada con números aleatorios de una distribución uniforme.
<b>Transformaciones básicas adicionales</b>	Sin transformaciones básicas adicionales a las aplicadas en el análisis exploratorio y feature engineering.
<b>Codificador de variables</b>	StringIndexer
<b>Selector de variables</b>	No aplicado.
<b>Reducción de dimensionalidad</b>	PCA.
<b>Unión de variables en un vector</b>	VectorAssembler
<b>Separación train-test</b>	randomSplit de todo el dataset.
<b>Modelo</b>	LinearSVC
<b>Ejecución del modelo</b>	ParamGridBuilder y CrossValidator.

Para el experimento #4, el modelo SVM tuvo un ROC muy bajo, de 0.53. Por lo que el modelo no funciona mejor que elegir los resultados al azar.

**Tabla 11 – Experimento #5**

<b>Particionamiento</b>	Particionamiento con una columna de salt generada con números aleatorios de una distribución uniforme.
<b>Transformaciones básicas adicionales</b>	Sin transformaciones básicas adicionales a las aplicadas en el análisis exploratorio y feature engineering.
<b>Codificador de variables</b>	StringIndexer
<b>Selector de variables</b>	No aplicado.
<b>Reducción de dimensionalidad</b>	PCA.
<b>Unión de variables en un vector</b>	VectorAssembler
<b>Separación train-test</b>	randomSplit de todo el dataset.
<b>Modelo</b>	Regresión logística ponderada con LogisticRegression
<b>Ejecución del modelo</b>	ParamGridBuilder y CrossValidator.

Para el experimento #5, la Regresión Logística Ponderada tuvo un ROC muy bajo, de 0.5. Por lo que el modelo no funciona mejor que elegir los resultados al azar.

**Tabla 12 – Experimento #6**

<b>Particionamiento</b>	Particionamiento con una columna de salt generada con números aleatorios de una distribución uniforme.
<b>Transformaciones básicas adicionales</b>	Sin transformaciones básicas adicionales a las aplicadas en el análisis exploratorio y feature engineering.
<b>Codificador de variables</b>	StringIndexer
<b>Selector de variables</b>	No aplicado.
<b>Reducción de dimensionalidad</b>	PCA.
<b>Estandarización de datos</b>	MinMaxScaler.
<b>Unión de variables en un vector</b>	VectorAssembler.
<b>Separación train-test</b>	randomSplit de todo el dataset.
<b>Modelo</b>	Complement Naive Bayes con NaiveBayes
<b>Ejecución del modelo</b>	ParamGridBuilder y CrossValidator.

Para el experimento #6 la combinación de funciones no permitió que la ejecución terminara, ni aumentando la cantidad de recursos del clúster.

Sin embargo, como esos modelos no fueron óptimos, el enfoque que se detallará a continuación resultó ser el experimento más efectivo y que se tomará como resultado final.

**Tabla 13 – Características del experimento final**

<b>Particionamiento</b>	Particionamiento con una columna de salt generada con números aleatorios de una distribución uniforme.
<b>Transformaciones básicas adicionales</b>	<ul style="list-style-type: none"> <li>• Eliminación de variables que generan ruido: user_id y card_id.</li> <li>• Crear nuevas features de fecha: month, day, day_of_year, day_of_month, day_of_week.</li> <li>• Eliminación de la variable merchant_city, pues esta tenía más de 10,000 valores, se podía utilizar el merchant_state que tenía menos valores.</li> </ul>
<b>Codificador de variables</b>	StringIndexer y OneHotEncoder (dependiendo de la cantidad de posibles valores de la columna).
<b>Selector de variables</b>	No aplicado.
<b>Reducción de dimensionalidad</b>	PCA.
<b>Estandarización de datos</b>	No aplicado.
<b>Unión de variables en un vector</b>	VectorAssembler.
<b>Separación train-test</b>	División estratificada con randomSplit. Se aplicó adicionalmente Oversampling para equilibrar las clases.
<b>Modelo</b>	LinearSVC, Logistic Regression e IsolationForest.
<b>Ejecución del modelo</b>	ParamGridBuilder, CrossValidator.

A continuación se explica de manera más detallada el experimento que resultó más exitoso, para posteriormente analizar los resultados de sus modelos y determinar el mejor para este caso.

### 4.3.1. Preparación de datos

A continuación, se describe el proceso de preparación final de datos para el modelo, que inicia con el cambio de nombre de la variable is\_fraud a label.

*Figura 119 – Renombre de la columna is\_fraud a label*

```

1  database = "fraud_detection"
2
3  # Cargar tu DataFrame de entrada
4  prod_transactions = spark.read.table(f"{database}.prod_transactions").withColumnRenamed("is_fraud", "label")
5  display(prod_transactions)

```

#### 4.3.1.1. Reparticionamiento

*Figura 120 – Reparticionamiento del conjunto de datos para el modelo*

```

4  prod_transactions = prod_transactions.withColumn("salt", rand()).repartition(8, "salt").drop("salt").cache()
5
6  # Contamos el numero de registros por particion
7  prod_transactions.groupBy(spark_partition_id()).count().display()

```

- ▶ (3) Spark Jobs
- ▶ prod\_transactions: pyspark.sql.dataframe.DataFrame = [user: string, card: string ... 19 more fields]

Table		Registros por partición (Original)		+ New res
	SPARK_PARTITION_ID()	count		
1	0	3045246		
2	1	3048954		
3	2	3050743		
4	3	3049919		
5	4	3047155		
6	5	3046134		
7	6	3049000		
8	7	3049683		

**Figura 121 – Reparticionamiento de los datos para el modelo**

```

1 # Numero de particiones original del conjunto de datos
2 prod_transactions.rdd.getNumPartitions()

4
Command took 0.17 seconds -- by sarychamsf@rumpjoneshotmail.onmicrosoft.com at 16/10/2023, 12:50:36 a. m. on Feature Engineering
Cmd 7

1 # Contamos el numero de registros por particion
2 prod_transactions = prod_transactions.withColumn("salt", rand()).repartition(8, "salt").drop("salt").cache() #
3 prod_transactions.groupBy(spark_partition_id()).count().display()

▶ (3) Spark Jobs
▶ prod_transactions: pyspark.sql.dataframe.DataFrame = [user: string, card: string ... 19 more fields]

Table      Registro por partición (Original)  +
New result table

SPARK_PARTITION_ID()  count
1 0 3045246
2 1 3048954
3 2 3050743
4 3 3049919
5 4 3047155
6 5 3046134
7 6 3049000
8 7 3049683

```

#### 4.3.1.2. Tratamiento de variables

Las columnas de ID de usuario e ID de tarjeta no contienen información directamente utilizable para el modelado predictivo. Se opta por eliminar o excluir estas variables.

**Figura 122 – Eliminación de las columnas de usuario y tarjeta**

```

1 prod_transactions = prod_transactions.drop("user", "card")

```

Extraer las variables de fecha para que el modelo tenga más información para detectar patrones importantes.

*Figura 123 – Creación de features de fecha*

```

1 prod_transactions = (
2     prod_transactions.withColumn("year", year("transaction_date"))
3     .withColumn("month", month("transaction_date"))
4     .withColumn("day", day("transaction_date"))
5     .withColumn("day_of_year", dayofyear("transaction_date"))
6     .withColumn("day_of_month", dayofmonth("transaction_date"))
7     .withColumn("day_of_week", dayofweek("transaction_date"))
8     .drop("transaction_date")
9 )
10
11 display(prod_transactions)

```

▶ (1) Spark Jobs  
▶ prod\_transactions: pyspark.sql.dataframe.DataFrame = [amount: decimal(10,2), merchant\_city: string ... 22 more fields]

Table +													New result table: OFF ✓
	credit_limit	anios_antiguedad	year	month	day	day_of_year	day_of_month	day_of_week					
1	14624.00	6	2017	9	22	265	22	6					
2	9094.00	7	2000	2	9	40	9	4					
3	9094.00	7	2000	12	30	365	30	7					
4	9094.00	9	2002	11	22	326	22	6					
5	9094.00	10	2003	2	1	32	1	7					
6	9094.00	10	2003	6	10	161	10	3					

#### 4.3.1.3. Codificación de variables

Las variables de año, mes y día se utilizarán como variables categóricas. Las trataremos con One-Hot-Encoding. La razón principal es que el año no tiene un orden intrínseco y tratarlo como una variable numérica implicaría que los años más altos tienen un valor mayor que los años más bajos, lo cual no tiene sentido en este caso. Esto ayudará a evitar que el modelo asigne un significado numérico incorrecto o infiera un orden inexistente entre los valores de día y mes.

*Figura 124 – Codificación de las variables de fecha*

```

1 # Lista auxiliar para unir el vector al final
2 dates_onehot = ["year_onehot", "month_onehot", "day_onehot"]
3
4 string_indexer = StringIndexer(inputCols=["year", "month", "day"], outputCols=["year_indexed", "month_indexed",
5 "day_indexed"])
6 one_hot_encoder = OneHotEncoder(inputCols=["year_indexed", "month_indexed", "day_indexed"], outputCols=dates_onehot)
7
8 dates_pipeline = Pipeline(stages=[string_indexer, one_hot_encoder])

Command took 0.10 seconds -- by sarychamsf@rumpjoneshotmail.onmicrosoft.com at 16/10/2023, 12:50:36 a. m. on Feature Engineering
Cmd 15

1 prod_transactions = dates_pipeline.fit(prod_transactions).transform(prod_transactions).drop("year", "month", "day",
2 "year_indexed", "month_indexed", "day_indexed")
2 display(prod_transactions)

▶ (12) Spark Jobs
▼ (1) MLflow run
    Logged 1 run to an experiment in MLflow. Learn more
    prod_transactions: pyspark.sql.dataframe.DataFrame = [amount: decimal(10,2), merchant_city: string ... 22 more fields]

Table + New result table: OFF ▾


|   | year_onehot                                                              | month_onehot                                                             | day_onehot            |
|---|--------------------------------------------------------------------------|--------------------------------------------------------------------------|-----------------------|
| 1 | ↳ {"vectorType": "sparse", "length": 29, "indices": [1], "values": [1]}  | ↳ {"vectorType": "sparse", "length": 11, "indices": [7], "values": [1]}  | ↳ {"vectorType": [1]} |
| 2 | ↳ {"vectorType": "sparse", "length": 29, "indices": [20], "values": [1]} | ↳ {"vectorType": "sparse", "length": 11, "indices": [10], "values": [1]} | ↳ {"vectorType": [1]} |
| 3 | ↳ {"vectorType": "sparse", "length": 29, "indices": [20], "values": [1]} | ↳ {"vectorType": "sparse", "length": 11, "indices": [1], "values": [1]}  | ↳ {"vectorType": [1]} |
| 4 | ↳ {"vectorType": "sparse", "length": 29, "indices": [17], "values": [1]} | ↳ {"vectorType": "sparse", "length": 11, "indices": [5], "values": [1]}  | ↳ {"vectorType": [1]} |


```

Se aplican transformaciones sobre las variables con muchas categorías únicas.

*Figura 125 – Análisis de variables con muchas categorías únicas*

```

1 # Variable merchant_city
2 prod_transactions.select("merchant_city").distinct().count()

▶ (3) Spark Jobs
13429
Command took 1.57 seconds -- by sarychamsf@rumpjoneshotmail.onmicrosoft.com at 16/10/2023, 12:50:37 a. m. on Feature Engineering
Cmd 18

1 # Variable merchant_state
2 prod_transactions.select("merchant_state").distinct().count()

▶ (3) Spark Jobs
224
Command took 1.00 second -- by sarychamsf@rumpjoneshotmail.onmicrosoft.com at 16/10/2023, 12:50:37 a. m. on Feature Engineering
Cmd 19

1 # Variable mcc_category
2 prod_transactions.select("mcc_category").distinct().count()

▶ (3) Spark Jobs
104

```

Se realiza un análisis de los porcentajes de cada categoría única para pensar una forma más sencilla de tratar las variable con alta dimensionalidad.

**Figura 126 – Porcentaje de categorías únicas para merchant\_city**

```

1 prod_transactions.groupBy("merchant_city").agg(
2   | (count("merchant_city") / prod_transactions.count() * 100).alias("percentage")
3   ).orderBy("percentage", ascending=False).display()

```

▶ (4) Spark Jobs

Table +

	merchant_city	percentage
1	ONLINE	11.1569259051831
2	Houston	1.0088886486864184
3	Los Angeles	0.7401370756039919
4	Miami	0.7325797190402001
5	Brooklyn	0.6373316027820586
6	Chicago	0.5583545613177996
7	Dallas	0.5528556925429516
8	Indianapolis	0.4957429078329725
9	Orlando	0.49523443674566364
10	San Antonio	0.47745435098299355
11	Philadelphia	0.4682608656785871
12	Atlanta	0.4641602923938384
13	Tucson	0.44219352130743994
14	Minneapolis	0.42142821819347276
15	Louisville	0.4088025530497317
16	Bronx	0.4052801605981326

↓ ▾ 10,000 rows | Truncated data | 1.90 seconds runtime

Se va a eliminar la variable merchant\_city por 4 motivos:

- Su cardinalidad es muy grande y se vuelve muy difícil de manejar.
- La dispersión de las categorías tiende a ser mucho mayor para la categoría `ONLINE`, como se puede ver en la tabla anterior.
- No se necesita un nivel tan detallado como la ciudad, con utilizar el state es suficiente.

**Figura 127 – Eliminación de la columna merchant\_city**

```

1 prod_transactions = prod_transactions.drop("merchant_city")
2 display(prod_transactions)

```

Las variables “merchant\_state” y “mcc\_category” se tratarán con StringIndexer antes de utilizarlas en un modelo de clasificación o aplicar PCA para reducir dimensionalidad. StringIndexer asigna un número único a cada categoría en función de su frecuencia, y esto permite que los algoritmos de aprendizaje automático trabajen con variables categóricas.

*Figura 128 – StringIndexer a las variables merchant\_state y mcc\_category*

```

1 # Lista auxiliar para unir el vector al final
2 big_categories = ["merchant_state_indexed", "mcc_category_indexed"]
3
4 string_indexer = StringIndexer(inputCols=["merchant_state", "mcc_category"], outputCols=["merchant_state_indexed",
5 "mcc_category_indexed"])

```

Command took 0.07 seconds -- by sarychamsf@rumpjoneshotmail.onmicrosoft.com at 16/10/2023, 12:50:38 a. m. on Feature Engineering

Cmd 26

```

1 prod_transactions = (
2     string_indexer.fit(prod_transactions)
3     .transform(prod_transactions)
4     .drop(
5         "merchant_state",
6         "mcc_category"
7     )
8 )

```

Las variables más pequeñas igual se trabajarán con One-Hot-Encoder.

*Figura 129 – StringIndexer a las variables categóricas*

```

1 # Lista auxiliar para unir el vector al final
2 small_categories = ["use_chip_onehot", "card_brand_onehot", "card_type_onehot"]
3
4 string_indexer = StringIndexer(
5     inputCols=["use_chip", "card_brand", "card_type"],
6     outputCols=["use_chip_indexed", "card_brand_indexed", "card_type_indexed"],
7 )
8 one_hot_encoder = OneHotEncoder(
9     inputCols=["use_chip_indexed", "card_brand_indexed", "card_type_indexed"],
10    outputCols=small_categories,
11 )
12
13 small_categories_onehot = Pipeline(stages=[string_indexer, one_hot_encoder])

```

Command took 0.09 seconds -- by sarychamsf@rumpjoneshotmail.onmicrosoft.com at 16/10/2023, 12:50:39 a. m. on Feature Engineering

Cmd 32

```

1 prod_transactions = (
2     small_categories_onehot.fit(prod_transactions)
3     .transform(prod_transactions)
4     .drop(
5         "use_chip",
6         "card_brand",
7         "card_type",
8         "use_chip_indexed",
9         "card_brand_indexed",
10        "card_type_indexed",
11     )
12 )
13
14 display(prod_transactions)

```

El día de la semana en número se puede tratar como una variable numérica sin problemas.

Como el valor numérico del día de la semana tiene un orden inherente, no es necesario codificarlo

como una variable categórica. En este caso, se utiliza el valor numérico directamente en el modelo y el algoritmo debería reconocer que estos valores tienen un orden lógico y podrá capturar cualquier patrón o correlación relacionada con el día de la semana en el conjunto de datos.

Respecto al día del año y el día del mes, ambos se pueden tratar como variables numéricas en un modelo de clasificación. Pues, a diferencia del caso del año, estos valores tienen un orden intrínseco y tienen un significado numérico.

Para el día del año, se puede usar directamente el número del día, que va del 1 al 365 (o 366 en años bisiestos) como una variable numérica continua. En cuanto al día del mes, se puede usar el número del día, que va desde 1 hasta 28, 30 o 31, dependiendo del mes.

#### 4.3.1.4. Unir variables en un vector

Cualquier tipo de modelo con MLlib debe tener las entradas de datos como vector.

**Figura 130 – Unir variables en un vector con VectorAssembler**

```

1 # Columnas numéricas
2 columnas_numericas = ["amount", "technical_glitch", "insufficient_balance", "bad_card_number", "bad_pin", "bad_cvv",
3 "bad_expiration", "bad_zipcode", "has_chip", "credit_limit", "anios_antiguedad", "day_of_year", "day_of_month",
4 "day_of_week"]
5
6 # Listas auxiliares
7 columnas_categoricas = dates_onehot + big_categories + small_categories
8
9 # Todas las columnas features
10 columnas_todas = columnas_numericas + columnas_categoricas
11
12 # Crear 2 vectores diferentes con las variables numericas y categoricas
13 vector_assembler = VectorAssembler(inputCols=columnas_todas, outputCol="features")

Command took 0.11 seconds -- by sarychams@rumpjones@hotmail.onmicrosoft.com at 16/10/2023, 1:06:15 a. m. on Feature Engineering
Cnd: 36

1 prod_transactions = vector_assembler.transform(prod_transactions).drop(*columnas_todas)
2 prod_transaction = prod_transactions.cache()
3 display(prod_transactions)

▶ (1) Spark Jobs
▶ [prod_transactions: pyspark.sql.dataframe.DataFrame = [label: integer, features: udt]
▶ [prod_transaction: pyspark.sql.dataframe.DataFrame = [label: integer, features: udt]

Table v +
New result table: O1

```

label	features
1 0	[({"vectorType": "sparse", "length": 93, "indices": [0, 8, 9, 10, 11, 12, 13, 15, 50, 64, 84, 85, 87, 88, 91], "values": [161.32, 1, 14624, 6, 265, 22, 6, 1, 1, 1, 24, 35, 1, 1]}), {"vectorType": "sparse", "length": 93, "indices": [0, 8, 9, 10, 11, 12, 13, 34, 53, 76, 84, 85, 86, 88, 91], "values": [65.46, 1, 9094, 7, 40, 9, 4, 1, 1, 1, 9, 9, 1, 1]}), {"vectorType": "sparse", "length": 93, "indices": [0, 8, 9, 10, 11, 12, 13, 34, 44, 83, 85, 88, 91], "values": [44.53, 1, 9094, 7, 365, 30, 7, 1, 1, 1, 6, 1, 1]}), {"vectorType": "sparse", "length": 93, "indices": [0, 8, 9, 10, 11, 12, 13, 31, 48, 64, 84, 85, 86, 88, 91], "values": [11.72, 1, 9094, 9, 326, 22, 6, 1, 1, 1, 9, 4, 1, 1]}), {"vectorType": "sparse", "length": 93, "indices": [0, 8, 9, 10, 11, 12, 13, 30, 53, 57, 85, 88, 91], "values": [31.62, 1, 9094, 10, 32, 1, 7, 1, 1, 1, 6, 1, 1]}]

Con las transformaciones se termina con 93 variables en total. Por ello, posteriormente, se llevará a cabo una reducción de dimensionalidad para que entrenar el modelo sea más sencillo.

#### 4.3.1.5. Dividir conjunto de datos para entrenamiento y prueba

Como se ha visualizado a lo largo del análisis exploratorio, el conjunto de datos tiene un desequilibrio de clases. Así que, se hace necesario hacer una división del conjunto de datos en entrenamiento y pruebas teniendo en cuenta esto. Para ello, se lleva a cabo una división estratificada o división balanceada para las transacciones fraudulentas y legítimas, manteniendo la proporción de clases en ambos conjuntos. Esto va a permitir que, tanto el conjunto de datos como el de prueba, contengan una representación adecuada de transacciones fraudulentas y legítimas.

**Figura 131 – División estratificada de los datos en train y test**

```

1 # Separamos los registros datos de los legítimos
2 is_fraud = prod_transactions.filter("label = 1")
3 is_legit = prod_transactions.filter("label = 0")

▶ is_fraud: pyspark.sql.dataframe.DataFrame = [label: integer, features: udt]
▶ is_legit: pyspark.sql.dataframe.DataFrame = [label: integer, features: udt]
Command took 0.23 seconds -- by sarychamsf@rumpjoneshotmail.onmicrosoft.com at 16/10/2023, 1:10:
:md 40

1 seed = 1234
2
3 # Usamos 70% para entrenamiento y 30% para prueba
4 (fraud_train, fraud_test) = is_fraud.randomSplit([0.7, 0.3], seed=seed)
5 (legit_train, legit_test) = is_legit.randomSplit([0.7, 0.3], seed=seed)
6
7 print("Fraud: ", fraud_train.count(), fraud_test.count())
8 print("Legit: ", legit_train.count(), legit_test.count())

▶ (9) Spark Jobs
▶ fraud_train: pyspark.sql.dataframe.DataFrame = [label: integer, features: udt]
▶ fraud_test: pyspark.sql.dataframe.DataFrame = [label: integer, features: udt]
▶ legit_train: pyspark.sql.dataframe.DataFrame = [label: integer, features: udt]
▶ legit_test: pyspark.sql.dataframe.DataFrame = [label: integer, features: udt]
Fraud: 20913 8844
Legit: 17050801 7306276

```

**Figura 132 – Construcción de los conjuntos de entrenamiento y prueba**

```

1 # Unir los DataDrames para construir los conjuntos de entrenamiento-prueba
2 train_data = fraud_train.union(legit_train)
3 test_data = fraud_test.union(legit_test)
4
5 print(train_data.count(), test_data.count())

```

▶ (4) Spark Jobs

- ▶ train\_data: pyspark.sql.dataframe.DataFrame = [label: integer, features: udt]
- ▶ test\_data: pyspark.sql.dataframe.DataFrame = [label: integer, features: udt]

17071714 7315120

```

1 # Almacenamos ambos conjuntos de datos en el Lakehouse
2 train_data.write.format("delta").mode("overwrite").option("overwriteSchema", True).saveAsTable(f"{database}.train_data")
3 test_data.write.format("delta").mode("overwrite").option("overwriteSchema", True).saveAsTable(f"{database}.test_data")

```

### 4.3.2. Reducción de dimensionalidad

Como método de reducción de dimensionalidad se utiliza PCA (Análisis de Componentes Principales). Esto con el fin de simplificar el análisis de los datos, eliminar multicolinealidad, preservar de la información importante, y mejorar el rendimiento del algoritmo.

**Figura 133 – Lectura del dataset para aplicar PCA**

```

1 database = "fraud_detection"
2
3 # Cargar tu DataFrame de entrada
4 train_data = spark.read.table(f"{database}.train_data").withColumn("salt", rand()).repartition(4, "salt").drop("salt").cache()
5 train_data.groupBy(spark_partition_id()).count().display()

```

Ahora se cuenta con 93 variables numéricas, ya no se tienen variables categóricas porque todas fueron convertidas a números en los pasos anteriores de feature engineering, tratamiento de variables y codificación.

Ejecutamos un PCA con un valor de k = 93 para analizar la varianza explicada de cada variable. Es decir, cuánta información es capaz de explicar cada nuevo factor. El total de la suma

de la varianza explicada de todos los factores es igual a 1 porque es equivalente a tener toda el conjunto de entrenamiento completo.

**Figura 134 - Implementación de PCA al modelo**

```

1  pca = PCA(k=93, inputCol="features", outputCol="features_pca")
2
3  model = pca.fit(train_data)
4  model.explainedVariance

▶ (6) Spark Jobs
▼ (1) MLflow run
  Logged 1 run to an experiment in MLflow. Learn more

DenseVector([0.9999, 0.0001, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

Con esta reducción de dimensionalidad podemos visualizar que el factor 1 es capaz de explicar el 99.99% de los datos. Esto reduce el espacio de dimensiones de 93 variables a una sola. Se guardan los conjuntos de datos de prueba y test con PCA aplicado.

Es importante tener en cuenta que se ajusta con el modelo de entrenamiento y no con el de prueba. PCA se ajusta con el método .fit(), y luego que ha aprendido la estructura de los datos de entrenamiento, se pueden entrenar para transformar tanto los de prueba como los de entrenamiento usando .transform().

**Figura 135 – Ajustar y transformar PCA.**

```

2  pca = PCA(k=1, inputCol="features", outputCol="features_pca").fit(train_data)
1  train_data_pca = pca.transform(train_data)
2  test_data_pca = pca.transform(test_data)
3
4  display(train_data_pca)
```

Se escriben los dos dataset a la base de datos para un acceso más sencillo a través de otros notebooks.

**Figura 136 – Almacenar conjunto de train y test con PCA.**

```
1 train_data_pca.write.format("delta").mode("overwrite").option("overwriteSchema", True).saveAsTable(f"{database}.train_data_pca")
2 test_data_pca.write.format("delta").mode("overwrite").option("overwriteSchema", True).saveAsTable(f"{database}.test_data_pca")
```

### 4.3.3. Entrenamiento de modelos

Antes de realizar el entrenamiento de cualquier modelo, primero se realiza la lectura de los datasets almacenados anteriormente. Como el clúster cuenta con 16 núcleos, se particionarán ambos conjuntos de datos con este número. Esto permitirá aprovechar los recursos al máximo.

**Figura 137 – Lectura y particionamiento del dataset**

```
1 database = "fraud_detection"
2
3 # Cargar los conjuntos de dato
4 train_data = (
5     spark.read.table(f"{database}.train_data_pca")
6     .withColumn("salt", rand())
7     .repartition(16, "salt")
8     .drop("salt")
9 )
10 test_data = (
11     spark.read.table(f"{database}.test_data_pca")
12     .withColumn("salt", rand())
13     .repartition(16, "salt")
14     .drop("salt")
15 )
```

Para implementar el algoritmo de SVM y detectar las anomalías de manera exitosa, se utilizará la técnica de oversampling. Esta permite aumentar el número de ejemplos de la clase que tiene una representación minoritaria o insuficiente.

En este caso, se aumentará el número de transacciones fraudulentas. Esto permitirá proporcionar mayor información al modelo para mejorar su capacidad de detectar patrones en la clase minoritaria. Su ventaja es que no se perderá información.

Existen varias maneras de realizar oversampling, pero en este caso se van a duplicar las instancias de datos para la clase minoritaria (transacciones fraudulentas). Para ello, se crea una columna “dummy” que contiene un arreglo con valores del 0 al  $ratio - 1$  (factor de duplicación), y luego con la función “explode” se duplica cada fila del DataFrame por el número de veces específico de dummy y luego se borra esta columna.

**Figura 138 – Cálculo de ratio entre clases**

```

1 # Separamos por clase.
2 fraud_train = train_data.filter("label == 1")
3 legit_train = train_data.filter("label == 0")
4
5 # Ratio entre cada clase
6 ratio = int(legit_train.count() / fraud_train.count())
7
8 print("count", fraud_train.count(), legit_train.count())
9 print("ratio", ratio)

```

```

▶ (12) Spark Jobs
▶ └─ fraud_train: pyspark.sql.dataframe.DataFrame = [label: integer, features: udt ... 1 more field]
▶ └─ legit_train: pyspark.sql.dataframe.DataFrame = [label: integer, features: udt ... 1 more field]
count 20913 17050801
ratio 815

```

**Figura 139 – Duplicar clase minoritaria**

```

1 # Duplicar la clase minoritaria
2 ratio_range = range(ratio)
3
4 # Duplicar la clase minoritaria
5 oversampling = fraud_train.withColumn("dummy", explode(array(lit(x) for x in ratio_range))).drop("dummy")
6
7 # Confirmar que efectivamente se duplicaron los registros
8 print(oversampling.count(), fraud_train.count() * ratio)

```

```

▶ (6) Spark Jobs
17044095 17044095
Command took 9.55 seconds -- by sarychamsf@rumpjones@hotmail.onmicrosoft.com at 16/10/2023, 5:52:10 a. m. on ML Training
Cmd 10

```

```

1 # Combinar el conjunto oversampled de la clase minoritaria con la clase mayoritaria
2 oversampled_train_data = legit_train.union(oversampling)
3 oversampled_train_data.groupBy("label").count().display()

```

```

▶ (4) Spark Jobs
▶ └─ oversampled_train_data: pyspark.sql.dataframe.DataFrame = [label: integer, features: udt ... 1 more field]

```

Table		New result table: OFF	
	label	count	
1	0	17050801	
2	1	17044095	
↓ 2 rows   3.00 seconds runtime			Refreshed 7 days ago

De esta forma, se procede a implementar 3 diferentes modelos.

#### 4.3.3.1. SVM

Se entrena el modelo usando distintas combinaciones de datos con K-fold Cross Validation y un valor de  $k = 5$ . Esto, pues “existe un intercambio de sesgo-varianza asociado con la elección de  $k$  en la validación cruzada de  $k$  veces. Normalmente, dadas estas consideraciones, se realiza una validación cruzada  $k$  veces usando  $k = 5$  o  $k = 10$ , ya que se ha demostrado empíricamente que estos valores producen estimaciones de la tasa de error de prueba que no sufren ni un sesgo excesivamente alto ni una varianza muy alta” (James, Witten, Hastie, Tibshirani, & Taylor, 2023).

El primero paso, consiste en definir los diferentes valores que se considerarán para la lista de parámetros de SVM.

**Figura 140 – Parámetros para SVM**

```

1 # Parametros del entrenamiento
2 PARALLELISM = 16
3 K = 5
4
5 # Parametros para SVM
6 reg_params = [0.0, 0.01, 0.1, 1.0]
7 standardization_params = [True, False]
8
9 # Parametros del evaluador
10 evaluator = BinaryClassificationEvaluator()

Command took 0.13 seconds --- by sarychamsf@rumpjoneshotmail.onmicrosoft
Cmd 14

1 # Numero de iteraciones para este modelo
2 K * len(reg_params) * len(standardization_params)

```

40

Luego, se entrena el modelo tomando las diferentes combinaciones entre los parámetros definidos anteriormente, para esto se utilizan ParamGridBuilder y CrossValidator. Se ajusta el modelo a los datos de entrenamiento, y se almacenan los resultados del modelo para que sean accedidos más fácilmente en la base de datos sin ejecutarlo nuevamente.

**Figura 141 – Entrenamiento del modelo SVM**

```

1 # Guardamos en cache el conjunto de datos usado
2 oversampled_train_data = oversampled_train_data.persist(StorageLevel.MEMORY_AND_DISK)
3
4 # Crear el modelo SVM
5 svm = LinearSVC(featuresCol="features", labelCol="label")
6
7 # Definir una cuadrícula de parámetros para el ajuste del modelo
8 param_grid = (
9     ParamGridBuilder()
10    .addGrid(svm.regParam, reg_params)
11    .addGrid(svm.standardization, standardization_params)
12    .build()
13 )
14
15 # Utilizar CrossValidator para ajustar el modelo
16 model_validator = CrossValidator(
17     estimator=svm,
18     estimatorParamMaps=param_grid,
19     evaluator=evaluator,
20     parallelism=PARALLELISM,
21     numFolds=K,
22     seed=SEED,
23 )
24
25 # Ajustar el modelo a los datos de entrenamiento
26 model = model_validator.fit(oversampled_train_data)
27
28 # AVG ROC
29 print(model.avgMetrics)
30
31 # Guardamos el modelo en el DBFS
32 model.write().save("dbfs:/FileStore/ml/svm/oversampled-features_pca")
33
34 # Quitamos de la cache del conjunto de datos usado
35 oversampled_train_data = oversampled_train_data.unpersist()

```

▶ (40) Spark Jobs  
 ▶ (8) MLflow runs  
 Logged 8 runs to an experiment in MLflow. [Learn more](#)  
 ▶ oversampled\_train\_data: pyspark.sql.dataframe.DataFrame = [label: integer, features: udt ... 1 more field]  
 [0.9170135155563065, 0.9170134567962839, 0.9147664885817486, 0.9112965271184041, 0.9133008127694117, 0.8856848230306449, 0.88981256622828  
 8, 0.8153004713610758]

### 4.3.3.2. Regresión Logística

Se entrena el modelo usando distintas combinaciones de datos con K-fold Cross Validatio.

El primero paso, consiste en definir los diferentes valores que se considerarán para la lista de parámetros del modelo de regresión logística.

**Figura 142 – Parámetros para regresión logística**

```

1 # Parametros del entrenamiento
2 PARALLELISM = 16
3 K = 5
4
5 # Parametros para la Regresión Logística
6 reg_params = [0.0, 0.01, 0.1, 1.0]
7 elasticNet_params = [0.0, 0.01, 0.5, 1.0]
8
9 # Parametros del evaluador
10 evaluator = BinaryClassificationEvaluator()

Command took 0.10 seconds -- by sarychamsf@rumpjoneshotmail.onmicrosoft.com at 16/10/2023, 6:03:47 a. m. on ML Training
md 14

1 # Numero de iteraciones para este modelo
2 K * len(reg_params) * len(elasticNet_params)

80

```

Luego, se entrena el modelo tomando las diferentes combinaciones entre los parámetros definidos anteriormente, para esto se utilizan ParamGridBuilder y CrossValidator. Se ajusta el modelo a los datos de entrenamiento, y se almacenan los resultados del modelo para que sean accedidos más fácilmente en la base de datos sin ejecutarlo nuevamente.

**Figura 143 – Entrenamiento del modelo de regresión logística**

```

1 # Guardamos en cache el conjunto de datos usado
2 oversampled_train_data = oversampled_train_data.persist(StorageLevel.MEMORY_AND_DISK)
3
4 # Crear el modelo de regresión logística
5 lr = LogisticRegression(featuresCol="features_pca", labelCol="label")
6
7 # Definir una cuadrícula de parámetros para el ajuste del modelo
8 param_grid = (
9     ParamGridBuilder()
10    .addGrid(lr.regParam, reg_params)
11    .addGrid(lr.elasticNetParam, elasticNet_params)
12    .build()
13 )
14
15 # Utilizar CrossValidator para ajustar el modelo
16 model_validator = CrossValidator(
17     estimator=lr,
18     estimatorParamMaps=param_grid,
19     evaluator=evaluator,
20     parallelism=PARALLELISM,
21     numFolds=K,
22     seed=SEED,
23 )
24
25 # Ajustar el modelo a los datos de entrenamiento
26 model = model_validator.fit(oversampled_train_data)
27
28 # AVG ROC
29 print(model.avgMetrics)
30
31 # Guardamos el modelo en el DBFS
32 model.write().save("dbfs:/FileStore/ml/logistic_regression/oversampled-features_pca")
33
34 # Quitamos de la cache del conjunto de datos usado
35 oversampled_train_data = oversampled_train_data.unpersist()

▶ (23) Spark Jobs
▼ (16) MLflow runs
    Logged 16 runs to an experiment in MLflow. Learn more
    ▶ oversampled_train_data: pyspark.sql.dataframe.DataFrame = [label: integer, features: udt ... 1 more field]
[0.5387764741931956, 0.5387739358643693, 0.5387768129003503, 0.5387745229569589, 0.5387727039542476, 0.5387731365311069, 0.5387763056809884, 0.538774208071
3329, 0.5387751615751906, 0.5387730751516486, 0.5, 0.5, 0.5387723396240418, 0.5387741036976536, 0.5, 0.5]

```

### 4.3.3.3. Isolation Forest

Este modelo se implementa con Scala, en vez de Python. Esto se debe a que no hay muchos modelos en PySpark con MLlib para tratar con este tipo de caso tan desproporcionados. Sin embargo se llevan a cabo operaciones similares.

Primero se realiza la lectura de los datasets almacenados anteriormente. Como el clúster cuenta con 20 núcleos, se partitionan ambos conjuntos de datos con este número. Esto permite aprovechar los recursos al máximo.

*Figura 144 – Lectura y particionamiento del dataset en Scala*

```

1  val database = "fraud_detection"
2
3  // Cargar los conjuntos de datos
4  var train_data = spark.read.table(s"$database.train_data_pca")
5    .withColumn("salt", rand())
6    .repartition(20, col("salt"))
7    .drop("salt")
8
9  var test_data = spark.read.table(s"$database.test_data_pca")
10   .withColumn("salt", rand())
11   .repartition(20, col("salt"))
12   .drop("salt")

▶ [train_data: org.apache.spark.sql.DataFrame = [label: integer, features: udt ... 1 more field]
▶ [test_data: org.apache.spark.sql.DataFrame = [label: integer, features: udt ... 1 more field]

database: String = fraud_detection
train_data: org.apache.spark.sql.DataFrame = [label: int, features: vector ... 1 more field]
test_data: org.apache.spark.sql.DataFrame = [label: int, features: vector ... 1 more field]

Command took 5.40 seconds -- by sarychamsf@rumpjoneshotmail.onmicrosoft.com at 16/10/2023, 9:30:34 p. m. on ML Training
:md 6

1  // Numero de filas de ambos
2  print(train_data.count(), test_data.count())

▶ (8) Spark Jobs
(17071714, 7315120)

```

Luego, se entrena el modelo tomando en cuenta el valor de contaminación como el parámetro que determina la proporción esperada de anomalías o valores atípicos en un conjunto de datos. Y así también, se toman en cuenta dos valores diferentes para el parámetro de Bootstrap, que indica si se lleva a cabo muestreo con reemplazo o no al momento de tomar las muestras aleatorias.

**Figura 145 – Isolation Forest en Scala sin bootstrapping**

```

1 // Guardamos en cache el conjunto de datos usado
2 train_data = train_data.persist(StorageLevel.MEMORY_AND_DISK)
3
4 // Crear el modelo Isolation Forest
5 var iForest = new IsolationForest()
6   .setMaxSamples(0.8)
7   .setBootstrap(false)
8   .setFeaturesCol("features_pca")
9   .setPredictionCol("prediction")
10  .setContamination(0.01)
11  .setContaminationError(0.0001)
12  .setRandomSeed(SEED)
13
14 // Ajustar el modelo a los datos de entrenamiento
15 var model = iForest.fit(train_data)
16
17 // Guardamos el modelo en el DBFS
18 model.write.overwrite.save("dbfs:/FileStore/ml/iforest/train_data-features_pca")
19
20 // Quitamos de la cache del conjunto de datos usado
21 train_data = train_data.unpersist()
```

► (1) Spark Jobs

```

train_data: org.apache.spark.sql.DataFrame = [label: int, features: vector ... 1 more field]
iForest: com.linkedin.relevance.isolationforest.IsolationForest = isolation-forest_1bb44ca6ab70
model: com.linkedin.relevance.isolationforest.IsolationForestModel = isolation-forest_1bb44ca6ab70
train_data: org.apache.spark.sql.DataFrame = [label: int, features: vector ... 1 more field]

Command took 1.05 hours -- by sarychamsf@rumpjoneshotmail.onmicrosoft.com at 16/10/2023, 6:54:08 p. m. on ML Training
```

---

d 11

```

1 // Realizar predicciones en el conjunto de prueba
2 var predictions = model.transform(test_data)
3
4 // Mostrar las predicciones
5 display(predictions.select("label", "prediction"))
```

► (2) Spark Jobs

► predictions: org.apache.spark.sql.DataFrame = [label: integer, features: udt ... 3 more fields]

---

Table ▾ +

	label	prediction
1	0	0
2	0	0

**Figura 146 - Isolation Forest en Scala con bootstrapping**

```

1 // Guardamos en cache el conjunto de datos usado
2 train_data = train_data.persist(StorageLevel.MEMORY_AND_DISK)
3
4 // Crear el modelo Isolation Forest
5 var iForest = new IsolationForest()
6   .setMaxSamples(0.8)
7   .setBootstrap(true)
8   .setFeaturesCol("features_pca")
9   .setPredictionCol("prediction")
10  .setContamination(0.01)
11  .setContaminationError(0.0001)
12  .setRandomSeed(SEED)
13
14 // Ajustar el modelo a los datos de entrenamiento
15 var model = iForest.fit(train_data)
16
17 // Guardamos el modelo en el DBFS
18 model.write.overwrite.save("dbfs:/FileStore/ml/iforest_bootstrap/train_data-features_pca")
19
20 // Quitamos de la cache del conjunto de datos usado
21 train_data = train_data.unpersist()
```

▶ (12) Spark Jobs

```

rain_data: org.apache.spark.sql.DataFrame = [label: int, features: vector ... 1 more field]
Forest: com.linkedin.relevance.isolationforest.IsolationForest = isolation-forest_d3928f7708cc
odel: com.linkedin.relevance.isolationforest.IsolationForestModel = isolation-forest_d3928f7708cc
rain_data: org.apache.spark.sql.DataFrame = [label: int, features: vector ... 1 more field]

ommand took 1.05 hours -- by sarychamsf@rumpjoneshotmail.onmicrosoft.com at 16/10/2023, 9:31:12 p. m. on ML Training
```

15

```

1 // Realizar predicciones en el conjunto de prueba
2 var predictions = model.transform(test_data)
3
4 // Mostrar las predicciones
5 display(predictions.select("label", "prediction"))
```

Una vez entrenado el modelo, se almacenan sus resultados para que sean accedidos más fácilmente en la base de datos sin ejecutarlo nuevamente.

## 5. Resultados

Se procede a realizar las validaciones de los modelos implementados anteriormente y se muestran los resultados obtenidos (métricas de los modelos).

Para esto, se crea un método que permita calcular las métricas de validación de un modelo: true positives, false negatives, false positives, true negatives, FRR, sensibilidad (recall), precisión, exactitud y F1-Score.

*Figura 147 – Cálculo de métricas de validación de modelos*

```

1  def calcular_metricas(predictions):
2
3      # Calcular True Positives (TP)
4      TP = predictions.filter((col("label") == 1) & (col("prediction") == 1)).count()
5
6      # Calcular False Negatives (FN)
7      FN = predictions.filter((col("label") == 1) & (col("prediction") == 0)).count()
8
9      # Calcular False Positives (FP)
10     FP = predictions.filter((col("label") == 0) & (col("prediction") == 1)).count()
11
12     # Calcular True Negatives (TN)
13     TN = predictions.filter((col("label") == 0) & (col("prediction") == 0)).count()
14
15     # Calcular Sensibilidad (Recall) o TPR
16     recall = TP / (TP + FN)
17
18     # FPR
19     FPR = FP / (FP + TN)
20
21     # Precisión (Precision)
22     precision = TP / (TP + FP)
23
24     # F1-Score
25     f1 = 2 * (precision * recall) / (precision + recall)
26
27     # Exactitud (Accuracy)
28     accuracy = (TP + TN) / (TP + TN + FP + FN)
29
30     return TP, FN, FP, TN, FPR, recall, precision, f1, accuracy
31

```

## 5.1. SVM

Para conocer cuáles fueron los mejores parámetros del modelo y otros detalles del experimento, se puede acceder a él desde la sección de Machine Learning / Experiments en Databricks.

**Figura 148 – Resultados del experimento SVM en Databricks**

The screenshot shows the Databricks interface with the sidebar navigation open. The 'Experiments' section is selected under 'Machine Learning'. The main content area displays the details of a completed experiment run. Key information includes:

- Run ID: 5f2fce0089734a45bc98fdf290e0b3e
- Date: 2023-10-16 06:04:19
- User: sarychamsf@rumpjoneshotmail.onmicrosoft.com
- Duration: 1.8h
- Lifecycle Stage: active
- Status: FINISHED
- Source: 4\_1\_entrenamiento\_con\_svm

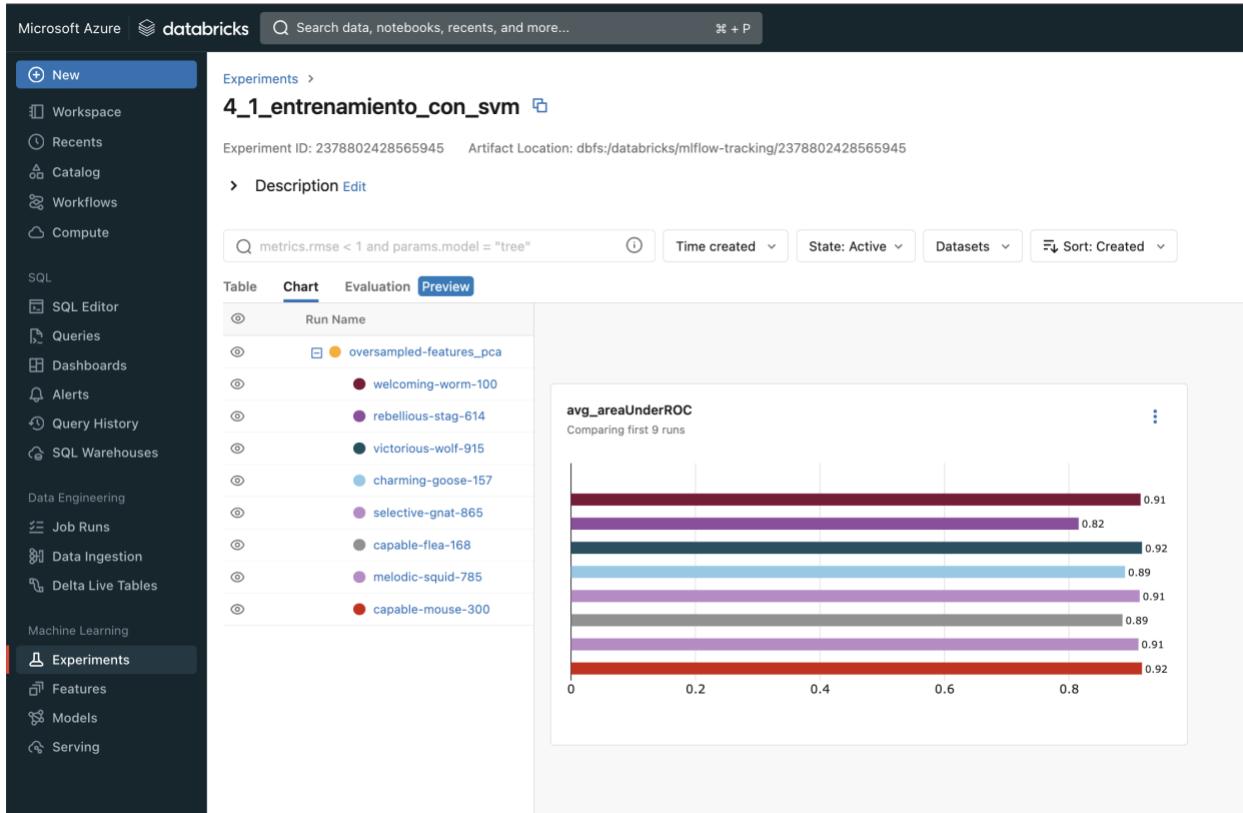
The 'Artifacts' section is expanded, showing the contents of the 'best\_model' directory. The 'best\_parameters.json' file is highlighted and its content is displayed:

```
{
  "LinearSVC.regParam": 0,
  "LinearSVC.standardization": true
}
```

- **regParam:** como el parámetro de regularización es igual a 0, no hay regularización, y el modelo se ajustará a los datos de entrenamiento sin restricciones.
- **standardization:** como el parámetro es True, se aplicará una normalización a los datos para que se encuentren en la misma escala y así sea más fácil comparar y analizar las variables.

También se pueden visualizar los valores de ROC si se visualizan las gráficas generadas por el modelo.

**Figura 149 – Valores de ROC para el modelo SVM**



Como se puede observar con los resultados anteriores, el mejor valor de ROC es de 0.92. Lo cual es un valor muy bueno. Luego, se procede a leer la tabla que se obtuvo como resultado del modelo para luego calcular las métricas.

**Figura 150 – Lectura del resultado del modelo SVM**

```

1  svm = spark.read.table("database.predictions_svm").withColumn("salt", rand()).repartition(16, "salt").drop("salt")
2  svm.display()

▶ (2) Spark Jobs
  Table + New result table: OFF ▾
  label  features  features_pca  rawPrediction
  1    0   ↗ ("vectorType": "sparse", "length": 93, "indices": [0, 1, 8, 9, 10, 11, 12, 13, 14, 44, 73, 84, 85, 87, 92], "values": [0.39, 1, 1, 7900, 18, 337, 3, 3, 1, 1, 1, 7, 1, 1, 1])
  2    0   ↗ ("vectorType": "sparse", "length": 93, "indices": [0, 1, 8, 9, 10, 11, 12, 13, 14, 47, 78, 84, 85, 87, 89, 91], "values": [23.66, 1, 1, 32269, 5, 186, 5, 6, 1, 1, 1, 23, 17, 1, 1, 1])
  3    0   ↗ ("vectorType": "sparse", "length": 93, "indices": [0, 1, 8, 9, 10, 11, 12, 13, 24, 47, 61, 84, 85, 86, 88, 91], "values": [14982.964615214027])

```

**Figura 151 – Mostrar los resultados de las predicciones para SVM**

```
1 # Mostrar las predicciones
2 svm_metricas = svm.select("label", "prediction").groupBy("label", "prediction").count().withColumnRenamed("count", "count_svm")
3 display(svm_metricas)
```

▶ (2) Spark Jobs

svm\_metricas: pyspark.sql.dataframe.DataFrame = [label: integer, prediction: double ... 1 more field]

Table +

	label	prediction	count_svm
1	1	0	1162
2	0	0	6265219
3	1	1	7682
4	0	1	1041057

**Figura 152 – Cálculo de métricas de validación para SVM**

```
1 recall, precision, f1, accuracy = calcular_metricas(svm)
2
3 # Mostrar las métricas calculadas
4 print(f"Sensibilidad (Recall): {recall}")
5 print(f"Precisión (Precision): {precision}")
6 print(f"Exactitud (Accuracy): {accuracy}")
7 print(f"F1-Score: {f1}")
```

▶ (12) Spark Jobs

Sensibilidad (Recall): 0.8686114880144731  
 Precisión (Precision): 0.00732498743729374  
 Exactitud (Accuracy): 0.8575253721059941  
 F1-Score: 0.014527464983835784

Como el enfoque del experimento es la sensibilidad (recall), podemos observar que el modelo responde muy bien al valor, aunque la precisión no es muy óptima.

## 5.2. Regresión Logística

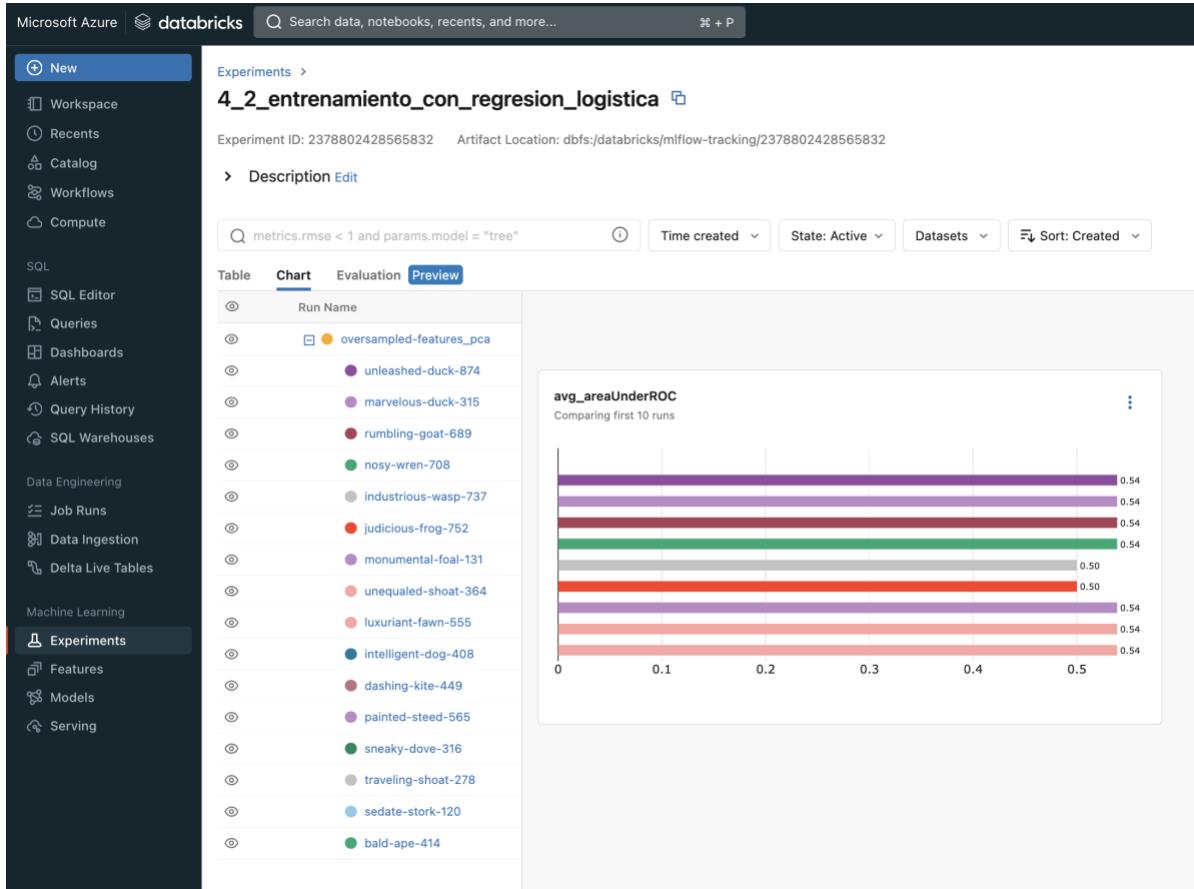
Para conocer cuáles fueron los mejores parámetros del modelo y otros detalles del experimento, se puede acceder a él desde la sección de Machine Learning / Experiments en Databricks.

**Figura 153 - Resultados del experimento de regresión logística en Databricks**

- **regParam:** como el parámetro de regularización es igual a 0, no hay regularización, y el modelo se ajustará a los datos de entrenamiento sin restricciones.
- **elasticNetParam:** es un parámetro controla la cantidad de regularización L1 (Lasso) y regularización L2 (Ridge) aplicada al modelo. Si su valor es 1 significa que utiliza la regularización de Lasso, si su valor es 0 utiliza Ridge. En este caso el valor intermedio de 0.5 indica que se ajusta a la combinación de ambas regularizaciones (Elastic Net) y esto evita el sobreajuste y permite seleccionar las mejores características.

También se pueden visualizar los valores de ROC si se visualizan las gráficas generadas por el modelo.

**Figura 154 – ROC del modelo de regresión logística**



Como se evidencia en los resultados anteriores, el valor de ROC óptimo es 0.54. Sin embargo, este valor es insatisfactorio, ya que sugiere que el modelo no es capaz de realizar predicciones significativas y, en cambio, se comporta como si eligiera valores al azar. Luego, se procede a leer la tabla que se obtuvo como resultado del modelo para luego calcular las métricas.

**Figura 155 – Lectura del resultado del modelo de Regresión Logística**

```

1  regresion_logistica = spark.read.table("database.predictions_logistic_regression").withColumn("salt", rand()).repartition(16, "salt").drop("salt")
2  regresion_logistica.display()

```

(3) Spark Jobs

regresion\_logistica: pyspark.sql.dataframe.DataFrame = [label: integer, features: udt ... 4 more fields]

label	features	features_pca	rawPrediction
1	0 ↳ {"vectorType": "sparse", "length": 93, "indices": [0, 1, 8, 9, 10, 11, 12, 13, 16, 50, 57, 85, 88, 91], "values": [22.31, 1, 1, 21284, 20, 244, 1, 7, 1, 1, 1, 8, 1, 1]}	↳ {"vectorType": "dense", "length": 1, "values": [-21284.00824996661]}	↳ {"vectorType": "dense", "length": 1, "values": [-33080.03786638581]}
2	0 ↳ {"vectorType": "sparse", "length": 93, "indices": [0, 1, 8, 9, 10, 11, 12, 13, 16, 65, 85, 89, 91], "values": [84.72, 1, 1, 33080, 10, 103, 13, 6, 1, 1, 16, 1, 1]}	↳ {"vectorType": "dense", "length": 1, "values": [-33080.03786638581]}	↳ {"vectorType": "dense", "length": 1, "values": [-33080.03786638581]}

**Figura 156 - Mostrar los resultados de las predicciones para regresión logística**

```

1 # Mostrar las predicciones
2 rl_metricas = regresion_logistica.select("label", "prediction").groupBy("label", "prediction").count().withColumnRenamed("count", "count_lr")
3 display(rl_metricas)

▶ (3) Spark Jobs
▶ rl_metricas: pyspark.sql.dataframe.DataFrame = [label: integer, prediction: double ... 1 more field]

Table + 
+---+---+---+---+
|label|prediction|count_lr|
+---+---+---+---+
| 1   | 0         | 3506    |
| 0   | 0         | 3264981 |
| 1   | 1         | 5338    |
| 0   | 1         | 4041295 |
+---+---+---+---+
4 rows | 2.85 seconds runtime

Command took 2.85 seconds -- by sarychamsf@rumpjones@hotmail.onmicrosoft.com at 22/10/2023, 10:44:35 p. m. on DataViz
Cmd 7

1 TP, FN, FP, TN, FPR, recall, precision, f1, accuracy = calcular_metricas(regresion_logistica)
2
3 # Mostrar las métricas calculadas
4 print(f"Sensibilidad (Recall): {recall}")
5 print(f"Precisión (Precision): {precision}")
6 print(f"Exactitud (Accuracy): {accuracy}")
7 print(f"F1-Score: {f1}")

▶ (12) Spark Jobs
Sensibilidad (Recall): 0.6035730438715513
Precisión (Precision): 0.0013191213534807828
Exactitud (Accuracy): 0.44706293266549285
F1-Score: 0.0026324893471224224

```

Siendo el enfoque del experimento la sensibilidad (recall), podemos observar que el modelo tiene un valor bastante regular de 0.6, pero tampoco son muy buenos los valores de precisión, exactitud o F1-Score.

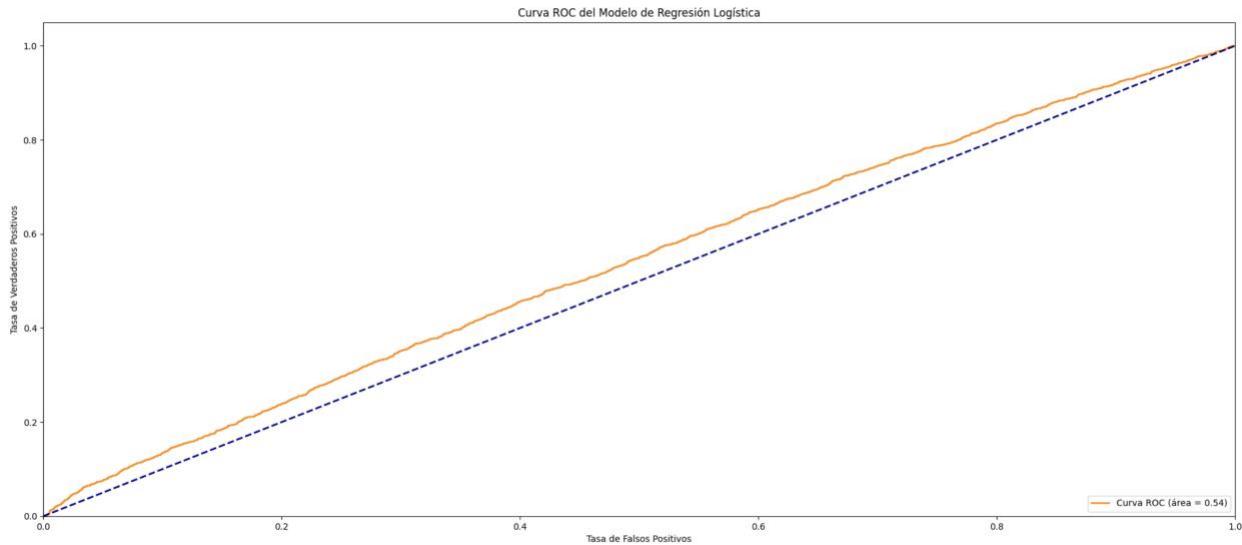
**Figura 157 – Graficar curva ROC del modelo de regresión logística**

```

1 # Obtener las probabilidades de clase positiva (label=1)
2 y_true = regresion_logistica.select("label")
3 y_scores = regresion_logistica.select("probability").rdd.map(lambda row: row[0][1])
4
5 # Calcular la curva ROC
6 binary_evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")
7 roc_auc = binary_evaluator.evaluate(regresion_logistica)
8
9 fpr, tpr, thresholds = roc_curve(y_true.collect(), y_scores.collect())
10
11 # Graficar la curva ROC
12 plt.figure(figsize=(24, 10))
13 plt.plot(fpr, tpr, color='darkorange', lw=2, label='Curva ROC (área = %0.2f)' % roc_auc)
14 plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
15 plt.xlim([0.0, 1.0])
16 plt.ylim([0.0, 1.05])
17 plt.xlabel('Tasa de Falsos Positivos')
18 plt.ylabel('Tasa de Verdaderos Positivos')
19 plt.title('Curva ROC del Modelo de Regresión Logística')
20 plt.legend(loc="lower right")
21 plt.show()

```

**Figura 158 – Curva ROC del modelo de regresión logísticas**



En la gráfica anterior de ROC, se observa con más detalle que el modelo es prácticamente aleatorio.

### 5.3. Isolation Forest

Este modelo es no supervisado, pero como realmente sí tenemos las variables de salida, excepcionalmente se pueden calcular las métricas.

**Figura 159 - Lectura del resultado del modelo de Isolation Forest**

```

1 isolation_forest = spark.read.table(f"database.predictions_iforest").withColumn("salt", rand()).repartition(16, "salt").drop("salt")
2 isolation_forest.display()

```

(2) Spark Jobs

isolation\_forest: pyspark.sql.dataframe.DataFrame = [label: integer, features: udt ... 3 more fields]

label	features	features_pca	outlierScore
1 0	>{"vectorType": "sparse", "length": 93, "indices": [0, 1, 8, 9, 10, 11, 12, 13, 17, 50, 68, 84, 85, 87, 88, 91], "values": [4.61, 1, 1, 20354, 4, 261, 17, 7, 1, 1, 14, 22, 1, 1, 1]}	>{"vectorType": "dense", "length": 1, "values": [-20353.99946256583]}	0.499176490138
2 0	>{"vectorType": "sparse", "length": 93, "indices": [0, 1, 8, 9, 10, 11, 12, 13, 21, 51, 57, 84, 85, 86, 89, 91], "values": [7.42, 1, 1, 14392, 2, 61, 1, 5, 1, 1, 14, 2, 1, 1, 1]}	>{"vectorType": "dense", "length": 1, "values": [-14392.001571473886]}	0.446182955212
3 0	>{"vectorType": "sparse", "length": 93, "indices": [0, 2, 8, 9, 10, 11, 12, 13, 15, 50, 82, 85, 89, 92], "values": [96.8, 1, 1, 6100, 13, 272, 29, 6, 1, 1, 11, 1, 11]}	>{"vectorType": "dense", "length": 1, "values": [-6100.046982264983]}	0.420472909530

*Figura 160 - Mostrar los resultados de las predicciones para Isolation Forest*

```

1 # Mostrar las predicciones
2 i_forest_metrics = (
3     isolation_forest.select("label", "prediction")
4     .groupBy("label", "prediction")
5     .count()
6     .withColumnRenamed("count", "count_i_forest")
7     display(i_forest_metrics)

```

▶ (2) Spark Jobs

▶ i\_forest\_metrics: pyspark.sql.dataframe.DataFrame = [label: integer, pre

Table +

	label	prediction	count_i_forest
1	1	0	7981
2	0	0	7230387
3	1	1	863
4	0	1	75889

↓ 4 rows | 0.95 seconds runtime

*Figura 161 – Cálculo de métricas para Isolation Forest*

```

1 recall, precision, f1, accuracy = calcular_metricas(isolation_forest)
2
3 # Mostrar las métricas calculadas
4 print(f"Sensibilidad (Recall): {recall}")
5 print(f"Precisión (Precision): {precision}")
6 print(f"Exactitud (Accuracy): {accuracy}")
7 print(f"F1-Score: {f1}")

```

▶ (12) Spark Jobs

Sensibilidad (Recall): 0.09758028041610131  
 Precisión (Precision): 0.01124400667083594  
 Exactitud (Accuracy): 0.9885347061975743  
 F1-Score: 0.02016449366792841

Si se considera que el enfoque principal de este experimento es la sensibilidad (recall), es evidente que el modelo presenta un valor muy bajo en este aspecto. La única métrica que muestra un valor significativamente alto es la precisión, pero es importante destacar que esta no es la prioridad.

## 6. Conclusiones

### 6.1. Análisis de resultados

Es importante tener en cuenta que el enfoque de este modelo es detectar anomalías, por ello se considera más prioritario no pasar por alto alguna transacción fraudulenta que detectar como fraudulenta una transacción que no lo es.

En estos casos “Analizar los indicadores de precisión general del modelo predictivo puede no ser lo más conveniente para escenarios desbalanceados” (Frola, Alvez, & Chesñevar, 2020). Es decir, que para el caso de uso, la métrica más adecuada es la sensibilidad (recall o tasa de verdaderos positivos), ya que esta mide la capacidad del modelo para identificar correctamente todas las transacciones fraudulentas en comparación de las transacciones fraudulentas reales.

En este contexto, los verdaderos positivos son las transacciones fraudulentas que el modelo detecta correctamente, y los falsos negativos son las transacciones fraudulentas que el modelo no logra identificar como tal. Maximizar la sensibilidad asegura que se minimicen los casos en los que se pasa por alto una transacción fraudulenta, lo que es crucial cuando la prioridad es evitar estos errores. Después de obtener todas las métricas para los tres modelos aplicados, se obtienen los siguientes valores:

**Figura 162 – Resultados de los modelos**

```

1  metricas = rl_metricas.join(svm_metricas, on=["label", "prediction"]).join(i_forest_metricas, on=["label", "prediction"])
2  display(metricas)

```

▶ (6) Spark Jobs

Table +

	label	prediction	count_lr	count_svm	count_i_forest
1	1	0	3506	1162	7981
2	0	0	3264981	6265219	7230387
3	1	1	5338	7682	863
4	0	1	4041295	1041057	75889

**Tabla 14 – Resultados de las métricas de los modelos implementados**

Métrica	SVM	Regresión Logística	Isolation Forest
<b>ROC</b>	0.92	0.54	-
<b>Sensibilidad</b>	0.8686	0.6035	0.0975
<b>Precisión</b>	0.0073	0.0013	0.0112
<b>Exactitud</b>	0.8575	0.4470	0.9885
<b>F1-Score</b>	0.0145	0.0026	0.0201

También se muestran los resultados de los modelos en cuanto al label y la predicción (falsos positivos, falsos negativos, verdaderos positivos y verdaderos negativos).

- SVM tiene la sensibilidad más alta (0.8686), lo que significa que es el mejor modelo para identificar transacciones fraudulentas. Sin embargo, la precisión (0.0073) es baja, lo que indica que puede haber un número significativo de falsos positivos.
- La regresión logística tiene una sensibilidad (0.6035) menor en comparación con SVM, pero aun así es significativa. La precisión (0.0013) es aún más baja, lo que sugiere una mayor cantidad de falsos positivos.
- Para el caso de Isolation Forest, aunque este modelo es no supervisado, se poseen las etiquetas de los datos, por lo que solo para este caso es posible calcular las métricas que normalmente no serían posibles. Siendo así, el modelo tiene una sensibilidad muy baja (0.0975), lo que indica que es menos efectivo en la detección de transacciones fraudulentas. Sin embargo, la precisión (0.0112) es más alta en comparación con los otros modelos.

Dado que la prioridad es minimizar la cantidad de transacciones fraudulentas pasadas por alto, SVM es la mejor opción entre estos tres modelos debido a su alta sensibilidad. Sin embargo,

también se debe considerar que la precisión de SVM es baja, lo que significa que puede generar más falsos positivos.

Podemos ver cómo el aplicar oversampling al conjunto de datos permite obtener resultados más favorables. Este enfoque tiene el beneficio de minimizar la omisión de transacciones fraudulentas (falsos negativos) al tiempo que puede llevar a un aumento en los falsos positivos (transacciones legítimas identificadas como sospechosas). Sin embargo, el énfasis está en generar alertas para detectar comportamientos sospechosos en lugar de centrarse exclusivamente en las transacciones fraudulentas, por lo que es un buen enfoque.

Además, es interesante notar que Databricks ofrece un entorno de trabajo robusto y versátil que es especialmente útil para abordar los desafíos relacionados con el procesamiento y análisis de conjuntos de datos voluminosos. Al permitir la creación de clústers y la partición de datos, Databricks facilita la manipulación de grandes volúmenes de información, así como la capacitación y evaluación de modelos de aprendizaje automático en un entorno unificado.

Así también, la capacidad de acceder a datos almacenados en data lakes y bases de datos desde el mismo entorno de trabajo de Databricks es una ventaja significativa, ya que simplifica la integración y la disponibilidad de datos necesarios para la detección de fraudes en transacciones con tarjetas de crédito. Esto contribuye a la eficiencia en el manejo de Big Data y facilita la implementación y pruebas de modelos de detección de fraudes. Es por esto que fue muy beneficioso utilizar Databricks para la realización del proyecto.

## 6.2. Áreas de mejora y trabajo futuro

A lo largo del proceso de investigación y desarrollo de esta tesis, se han identificado ciertos aspectos que han presentado desafíos y que pueden servir como puntos de referencia para futuros trabajos:

- Si bien la utilización de Databricks y su librería MLlib ha sido beneficiosa en términos de manejo de datos y análisis, es importante destacar que puede presentar limitaciones, como costos significativos y el uso intensivo de recursos al ejecutar modelos con grandes conjuntos de datos. En futuras investigaciones, se sugiere considerar el uso de librerías como TensorFlow, diseñadas para abordar volúmenes de datos aún mayores, lo que podría mejorar la eficiencia en la implementación de modelos de detección de fraudes.
- La librería MLlib en Python, aunque valiosa, puede tener modelos limitados para aplicar en entornos distribuidos. Por lo tanto, se recomienda explorar otras librerías y lenguajes de programación que ofrezcan un espectro más amplio de modelos y técnicas de aprendizaje automático. Esta diversificación puede enriquecer las opciones disponibles para la detección de anomalías en transacciones con tarjetas de crédito.
- Una línea prometedora para investigaciones futuras podría ser la implementación de modelos de aprendizaje automático no supervisados. Estos modelos podrían ofrecer ventajas en la detección de anomalías al no depender de etiquetas de datos y permitir la identificación de patrones inusuales de manera más flexible. Explorar enfoques no supervisados podría llevar a un mayor rendimiento en la detección de transacciones fraudulentas.
- El modelo mostró un excelente rendimiento en cuanto al recall. Sin embargo, se observó un desempeño menos satisfactorio en precisión y exactitud. Por lo tanto, se sugiere la búsqueda de un equilibrio más adecuado entre estas otras métricas.
- El conjunto de datos utilizado en esta tesis fue generado de manera sintética, lo que podría haber influido en la idoneidad de algunas de sus características. En futuros

estudios, se sugiere considerar el uso de conjuntos de datos reales para validar si los resultados obtenidos se mantienen o difieren en un entorno más representativo.

## 7. Referencias

- IBM. (29 de Julio de 2021). *Kaggle*. Obtenido de Synthetic Credit Card Transaction Data for Fraud Detection: <https://www.kaggle.com/datasets/ealtman2019/credit-card-transactions>
- ISO. (2003). *ISO 18245:2003 Retail financial services — Merchant category codes*. Obtenido de ISO: <https://www.iso.org/standard/33365.html>
- myFICO. (2023). *What is a Credit Score?* Obtenido de myFICO: <https://www.myfico.com/credit-education/credit-scores>
- Microsoft. (2023). *¿Cómo funciona Azure?* Obtenido de Microsoft Learn: <https://learn.microsoft.com/es-es/azure/cloud-adoption-framework/get-started/what-is-azure>
- Microsoft. (2023). *¿Qué es Azure Databricks?* Obtenido de Microsoft Learn: <https://learn.microsoft.com/es-es/azure/databricks/introduction/>
- Apache Spark. (2023). *PySpark Documentation*. Obtenido de Apache Spark: <https://spark.apache.org/docs/3.3.1/api/python/index.html>
- W3Schools. (2023). *SQL Tutorial*. Obtenido de W3Schools: <https://www.w3schools.com/sql/>
- Kaggle. (2017). *What is Kaggle, Why I Participate, What is the Impact?* Obtenido de Kaggle: <https://www.kaggle.com/discussions/getting-started/44916>
- Kaggle. (2023). *How to Use Kaggle*. Obtenido de Kaggle: <https://www.kaggle.com/docs/api>
- Knaddison, G., Martins Pereira, J. S., & Lewis, E. (2020). *MCC Codes [GitHub repository]*. Obtenido de GitHub: <https://github.com/greggles/mcc-codes>
- James, G., Witten, D., Hastie, T., Tibshirani, R., & Taylor, J. (2023). *An Introduction to Statistical Learning with Applications in Python*. Springer.

- Frola, F., Alvez, C., & Chesñevar, C. (2020). *Un primer acercamiento a un modelo predictivo ajustable por umbrales para detección de fraudes financieros*. Obtenido de Simposio Argentino de Inteligencia Artificial: <https://49jaiio.sadio.org.ar/pdfs/asai/ASAI-09.pdf?fbclid=IwAR2rkuA5s3zkHnOXI7UNNxRKcw56LI50kMC7sNjreCpLthayNPlucr6qw5U>
- Cepeda García, D. (Julio de 2012). *Detección de Fraude en Tarjetas de Crédito*. Obtenido de Repositorio Académico - Universidad de Chile:  
<https://repositorio.uchile.cl/bitstream/handle/2250/110943/Deteccion-de-fraude-en-tarjetas-de-cr%C3%A9dito.pdf?sequence=3&isAllowed=y&fbclid=IwAR3eW-koIer1KvuiNvKQeeg57H2m-qF2a7iz3QbNUovO5wdbC7C3aDWEGpQ>
- Gutiérrez-Portela, F., Perdomo-Guerrero , C., Mario Heimer, F.-G., Hernández-Aros , L., & Quiceno-Castañeda, D. (2020). *Mapeo de las Técnicas de Aprendizaje Automático Usados en Fraudos con Tarjeta de Crédito*. Obtenido de Alianza de Investigadores Internacionales S.A.S Alinin: [https://alinin.org/wp-content/uploads/2020/10/ten\\_inv\\_uni\\_ix\\_59\\_71.pdf?fbclid=IwAR1ITabuXapix-Z7uDauEm4p4GwNtDSwazGATNe7ZzvTYLb9tjWsRiNhRg0](https://alinin.org/wp-content/uploads/2020/10/ten_inv_uni_ix_59_71.pdf?fbclid=IwAR1ITabuXapix-Z7uDauEm4p4GwNtDSwazGATNe7ZzvTYLb9tjWsRiNhRg0)
- Wan, J. (9 de Febrero de 2020). *Oversampling and Undersampling with PySpark*. Obtenido de Medium: <https://medium.com/@junwan01/oversampling-andundersampling-with-pyspark-5dbc25cdf253>
- Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research* 16, 321-357.

- Brownlee, J. (26 de October de 2020). *Cost-Sensitive Logistic Regression for Imbalanced Classification*. Obtenido de Machine Learning Mastery:  
<https://machinelearningmastery.com/cost-sensitive-logistic-regression/>
- Elastic. (2021). *¿Qué es la detección de anomalías?* Obtenido de Elastic B.V.:  
<https://www.elastic.co/es/what-is/anomaly-detection>
- Liu, F. T., Ting, K. M., & Zhou, Z. H. (2008). Isolation Forest. *2008 Eighth IEEE International Conference on Data Mining*, (págs. 413-422). Pisa, Italia.
- Verbus, J. (13 de Agosto de 2019). *Detecting and preventing abuse on LinkedIn using isolation forests*. Obtenido de LinkedIn Engineering:  
<https://engineering.linkedin.com/blog/2019/isolation-forest>
- Yadav, V. (27 de Agosto de 2021). *Optimizing the Skew in Spark*. Obtenido de Clairvoyant:  
<https://www.clairvoyant.ai/blog/optimizing-the-skew-in-spark>
- Berk, M. (10 de Mayo de 2022). *PySpark Data Skew in 5 Minutes*. Obtenido de Towards Data Science: <https://towardsdatascience.com/data-skew-in-pyspark-783d529a9dd7>
- Urrego, N. (2022). *Transformando datos en oro: Cómo la estandarización y normalización mejoran tus resultados*. Obtenido de Medium:  
<https://medium.com/@nicolasurrego/transformando-datos-en-oro-c%C3%B3mo-la-estandarizaci%C3%B3n-y-normalizaci%C3%B3n-mejoran-tus-resultados-fbe0840d2b94#:~:text=La%20estandarizaci%C3%B3n%20se%20 enfoca%20en,espec%C3%A9fico%20o%20una%20escala%20definida.>
- ProjectPro. (12 de Octubre de 2023). *How Data Partitioning in Spark helps achieve more parallelism?* Obtenido de ProjectPro: <https://www.projectpro.io/article/how-data-partitioning-in-spark-helps-achieve-more-parallelism/297>

Scala. (2021). *Scala*. Obtenido de Introduction: <https://docs.scala-lang.org/scala3/book/introduction.html>

Scikit-Learn. (2021). *Binary classification*. Obtenido de Scikit-Learn: [https://scikit-learn.org/stable/modules/model\\_evaluation.html#binary-classification](https://scikit-learn.org/stable/modules/model_evaluation.html#binary-classification)

Scikit-Learn. (2023). *Complement Naive Bayes*. Obtenido de Scikit-Learn: [https://scikit-learn.org/stable/modules/naive\\_bayes.html#complement-naive-bayes](https://scikit-learn.org/stable/modules/naive_bayes.html#complement-naive-bayes)

Scikit-Learn. (2023). *Support Vector Machines*. Obtenido de Scikit Learn: <https://scikit-learn.org/stable/modules/svm.html#svm-outlier-detection>

Scikit-Learn. (2021). *F1-Score*. Obtenido de Scikit-Learn: [https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1\\_score.html](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html)

Raval, P. (2023). *ProjectPro*. Obtenido de How to Learn Scala for Data Engineering?: <https://www.projectpro.io/article/learn-scala-for-data-engineering/602>

Scikit-Learn. (2021). *Receiver operating characteristic (ROC)*. Obtenido de Scikit-Learn: [https://scikit-learn.org/stable/modules/model\\_evaluation.html#receiver-operating-characteristic-roc](https://scikit-learn.org/stable/modules/model_evaluation.html#receiver-operating-characteristic-roc)

Barrios Arce, J. I. (26 de Julio de 2019). *La matriz de confusión y sus métricas*. Obtenido de Health Big Data: <https://www.juanbarrios.com/la-matriz-de-confusion-y-sus-metricas/>

Apache Spark. (2021). *Machine Learning Library (MLlib) Guide*. Obtenido de Spark 3.5.0 documentation: <https://spark.apache.org/docs/latest/ml-guide.html>

Apache Spark. (2021). *StringIndexer*. Obtenido de PySpark 3.5.0 documentation: <https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.StringIndexer.html>

Gajawada, S. K. (4 de Octubre de 2019). *Chi-Square Test for Feature Selection in Machine learning*. Obtenido de Towards Data Science: <https://towardsdatascience.com/chi-square-test-for-feature-selection-in-machine-learning-206b1f0b8223>

DeepAI. (s.f.). *One-Hot Encoding*. Obtenido de DeepAI: The front page of AI: <https://deepai.org/machine-learning-glossary-and-terms/one-hot-encoding>

## **8. Anexos**

**8.1. 1\_extracciondbc**

**8.2. 2\_1\_eda\_transformacion\_transactionsdbc**

**8.3. 2\_2\_eda\_transformacion\_cardsdbc**

**8.4. 2\_3\_eda\_transformacion\_usersdbc**

**8.5. 3\_1\_preparacion\_dataset\_modelodbc**

**8.6. 3\_2\_preparacion\_dataset\_modelo\_mllibdbc**

**8.7. 3\_3\_reduccion\_dimensionalidaddbc**

**8.8. 4\_1\_entrenamiento\_con\_svmdbc**

**8.9. 4\_2\_entrenamiento\_con\_Regresion\_logisticadbc**

**8.10. 4\_3\_entrenamiento\_con\_isolation\_forestdbc**

**8.11. 5\_metricas\_validaciondbc**