Optimización del tratamiento de datos en información de facturación y recaudo mediante frameworks de big data y aplicación de análisis predictivo: caso de estudio en una empresa de aseo domiciliario

Carlos Mario Cordero Rodríguez

Asesor

Jorge Luis Quintero López

Universidad Nacional Abierta y a Distancia UNAD

Escuela de Ciencias Básicas, Tecnología e Ingeniería ECBTI

Especialización en Ciencia de Datos y Analítica

2025

Resumen

El presente proyecto se enfoca en el análisis de herramientas de Big Data para la gestión de la información de suscriptores en una empresa de aseo domiciliario. La empresa enfrenta el desafío de procesar grandes volúmenes de datos de facturación provenientes de la empresa de energía con la cual realiza la facturación conjunta de sus servicios. El análisis de estos datos se ve limitado por el uso de herramientas tradicionales como Excel y Power BI, lo que dificulta la conciliación de los montos facturados y la detección de posibles errores.

Para abordar esta problemática, se propone analizar las herramientas y tecnologías de Big Data disponibles, con el fin de identificar las más efectivas para analizar la información y optimizar el proceso de tratamiento de los datos. Se realizará una revisión de literatura científica y casos de estudio para analizar las características y funcionalidades de las principales herramientas de Big Data, evaluar su capacidad para procesar grandes volúmenes de datos y generar reportes a partir de la demanda de los datos, y diseñar una propuesta de implementación de una herramienta de Big Data adaptada a las necesidades de la empresa.

***Palabras claves*:** Big Data, framework, escalabilidad, eficiencia, optimización.

Abstract

This project focuses on the analysis of Big Data tools for the management of subscriber information in a home cleaning company. The company faces the challenge of processing large volumes of billing data from the energy company with which it performs the joint billing of its services. The analysis of this data is limited to using traditional tools such as Excel and Power BI, which makes it difficult to reconcile the invoiced amounts and detect possible errors.

To address this problem, we propose analyzing the Big Data tools and technologies available, in order to identify the most effective ones to reconcile the amounts billed with the energy company and optimize the collection process. A review of scientific literature and case studies will be conducted to analyze the characteristics and functionalities of the main Big Data tools, evaluate their capacity to process large volumes of data and generate reconciliation reports, and design a proposal for the implementation of a Big Data tool adapted to the company's needs.

Keywords**:** Big Data, framework, scalability, efficiency, optimization.

Tabla de Contenido

Lista de Tablas

Lista de Figuras

Introducción

Actualmente, la gestión de información se ha convertido en una tarea esencial para las empresas, principalmente en el sector de servicios donde la satisfacción del cliente juega un papel principal (Haro Sarango et al., 2023). Desde este punto de vista, una empresa de aseo domiciliario en la costa colombiana presenta conflictos al analizar grandes cantidades datos relacionados a su facturación. Esta actividad alusiva a la entrega tarifaria a sus clientes, en su gran mayoría se realiza de manera conjunta con la empresa de energía, lo que implica recibir mensualmente reportes con centenas de millares de registros.

El procesamiento y análisis de esta cantidad de datos se ha convertido en un obstáculo para la empresa, partiendo que, utilizan comúnmente herramientas de ofimática para sus actividades. Si bien es cierto que estas herramientas como lo pueden ser Excel o incluso Power BI suelen ser muy eficientes para el análisis de conjuntos de datos pequeños, la carga de trabajo que genera procesar tanta información supera sus límites (Gagliordi, 2023). Esta situación evita la detección de irregularidades en el recaudo y posibles pérdidas económicas causantes de no poder detectar errores que se puedan encontrar en la conciliación de los montos facturados con la empresa de energía, además, impide que se desarrollen análisis profundos sobre el comportamiento de suscriptores e impide tomar decisiones estratégicas para mejorar el servicio (Lapiedra et al., 2021).

La empresa necesita de una solución que le permita manejar datos en un entorno de Big Data para su procesamiento, tratamiento y análisis eficiente de los datos masivos presentados en la facturación de sus clientes. Una herramienta que permita automatizar el proceso de extracción de indicadores que generen un valor significativo en la detección de patrones en los pagos,

montos facturados e identificación de inconsistencias los cuales permitan tomar decisiones acertadas en momentos críticos (Çelik, 2024).

Bajo esta primicia surge el planteamiento de la siguiente pregunta:

¿Cuáles son las herramientas y/o tecnologías de Big Data adecuadas que permitan analizar los montos facturados con la empresa de energía, optimicen el proceso de recaudo y que pueda gestionar eficientemente la información de suscriptores en empresas de aseo domiciliario?

Justificación

La creciente digitalización y el incremento constante de grandes volúmenes de datos son un obstáculo para las empresas de sectores servicios en general y particularmente en información de facturación para la industria de los servicios (Goh et al., 2024). En este sentido, una empresa de aseo doméstico de la costa de Colombia necesita procesar y analizar mensualmente las cantidades de información que recibe de la empresa de energía que, en conjunto, son responsables de efectuar la facturación de los servicios ofrecidos relacionados a la recolección de residuos. La gran cantidad de datos involucrada supera la capacidad y los recursos que emplea la empresa, de forma que no es posible analizar los datos, detectar inconsistencias e imposibilita la capacidad de generar insights para la toma de decisiones. (Zaki et al., 2024).

Para esta problemática se propone el uso de herramientas y tecnologías de Big Data para el procesamiento de datos masivos con el fin de optimizar y estandarizar el proceso de análisis (Mohanty et al., 2024; Wang et al., 2024). De esta manera, el propósito de la investigación se centra en analizar las herramientas de Big Data más populares, con el fin de identificar las más efectivas para analizar la información entregada por la empresa de energía y optimizar el tratamiento de los datos.

Este estudio es aplicable en la actual situación del sector de servicios, puesto que, la implantación de nuevas tecnologías de gestión de la información se hace cada vez más necesaria (Arganza Salcedo & Arroyo López, 2019). Este trabajo proporciona información sobre las herramientas de Big Data que son adecuadas para la industria del aseo domiciliario que podrán ser de referencia para otras empresas que presenten problemas similares. Aparte de analizar las herramientas disponibles en mercado, este trabajo tiene un carácter aplicable y práctico en la medida en que estipula diseñar una propuesta de implantación acorde a las necesidades de la

empresa de aseo domiciliario. Esto le proporcionará a la empresa una guía concreta para la implantación de estas tecnologías y la optimización de sus procesos, posibilitando mejoras en sus procesos, evitando posibles pérdidas financieras.

Objetivos

Objetivo General

Diseñar e implementar un procedimiento técnico basado en herramientas de Big Data que permita optimizar el tratamiento de datos de recaudo y facturación en una empresa de aseo domiciliario, incorporando modelos predictivos que generen métricas clave para la toma de decisiones.

Objetivos Específicos

Analizar distintos frameworks de Big Data para la gestión de datos estructurados, con el fin de seleccionar el más adecuado en términos de escalabilidad, integración y rendimiento para el entorno empresarial.

Desarrollar un flujo automatizado de tratamiento de datos que integre fuentes externas, gestione grandes volúmenes de información y consolide datos de recaudo y facturación en un entorno distribuido.

Construir modelos de aprendizaje supervisado y no supervisado que permitan predecir el riesgo de mora y segmentar usuarios según sus patrones de comportamiento, evaluando su desempeño mediante métricas de clasificación y validación cruzada.

Marco Conceptual y Teórico

Conceptos y Desafíos del Big Data

El Big Data ha emergido como un fenómeno transformador en la era digital, caracterizado por conjuntos de datos de gran volumen, alta velocidad de generación, variabilidad de formatos y veracidad que encierran un enorme valor, estos elementos hacen parte de las 5V's del Big Data (Amalina et al., 2020; Mayer-Schönberger & Cukier, 2013). Estos datos, que desbordan la capacidad de las herramientas de procesamiento tradicionales, presentan desafíos significativos para las organizaciones (Gagliordi, 2023).

El BDA (Big Data Análisis) es un proceso que busca extraer información valiosa y conocimiento de grandes conjuntos de datos que se ha convertido en una actividad esencial en diversas industrias, donde el volumen de datos ha crecido exponencialmente en presencia de varios de formatos (Hussain et al., 2023; Reis & Housley, 2022). En este contexto, el análisis de Big Data presenta desafíos importantes, como la integración y funcionamiento armonioso de las 5V's, la necesidad de procesar grandes volúmenes de datos a alta velocidad con una gran variedad de fuentes y formatos de datos, garantizando la calidad y la confiabilidad (Amalina et al., 2020). Además, la seguridad y la privacidad de la información son aspectos críticos que deben abordarse en el análisis de Big Data, concretamente en el ámbito del gobierno y ética de datos (Hussain et al., 2023).

Tecnologías de Big Data

Para abordar los desafíos del Big Data, se han desarrollado diversas tecnologías, entre las que destacan Hadoop y Spark ya que permiten procesar estos datos y obtener información relevante para la toma de decisiones (Syed et al., 2021). Hadoop, un framework de código abierto para el procesamiento distribuido de grandes conjuntos de datos, se basa en el paradigma

MapReduce. Ofrece un sistema de archivos distribuido (HDFS) para el almacenamiento de datos y un sistema de gestión de recursos (YARN) para la asignación de recursos y tareas (García Núñez & Olmedo Flores, 2021).

Una infraestructura de almacenamiento de datos construida sobre Hadoop es Hive, esta proporciona capacidades de análisis de datos mediante un lenguaje de consulta similar a SQL (HiveQL) la cual facilita el análisis de datos para los desarrolladores familiarizados con datos estructurados, permitiendo la creación de almacenes de datos sobre Hadoop (Gunay et al., 2019).

Spark, otro framework de código abierto para el procesamiento distribuido, se destaca por su modelo de programación basado en la memoria, que lo hace más rápido para aplicaciones que requieren procesamiento iterativo (Montoya Suárez & Gil Restrepo, 2018). Spark ofrece una serie de bibliotecas para el análisis de datos, el aprendizaje automático y el procesamiento de grafos, lo que lo convierte en una herramienta versátil para diversas aplicaciones de Big Data (García Núñez & Olmedo Flores, 2021).

Apache Spark se ha convertido en un estándar en la industria debido a su velocidad, generalidad y escalabilidad (Tang et al., 2022). Spark se basa en un modelo de programación flexible llamado RDD (Resilient Distributed Dataset), que permite a los usuarios procesar datos de forma eficiente y personalizada (Tang et al., 2022). A diferencia de otros frameworks como MapReduce, Spark puede soportar computación por lotes, interactiva, iterativa y de streaming en el mismo tiempo de ejecución, lo que lo hace adecuado para una amplia gama de aplicaciones (Tang et al., 2022).

Hadoop MapReduce y Spark son dos frameworks de computación distribuida que se utilizan ampliamente para el análisis de Big Data (Ketu et al., 2020). Hadoop MapReduce es un framework basado en disco, mientras que Spark es un framework basado en memoria. Spark es

generalmente más rápido que Hadoop MapReduce para aplicaciones que requieren procesamiento iterativo, pero Hadoop MapReduce puede ser más eficiente para aplicaciones que requieren un solo paso de procesamiento (Ketu et al., 2020).

Partición de Datos

La partición de datos es una técnica fundamental para el procesamiento eficiente de grandes volúmenes de datos en entornos distribuidos (Ponnusamy & Gupta, 2024). Esta técnica permite dividir los datos en subconjuntos más pequeños que se pueden procesar en paralelo, lo que aumenta la escalabilidad y reduce el tiempo de procesamiento (Ponnusamy & Gupta, 2024). Existen diferentes técnicas de partición de datos, como la partición basada en hash, la partición basada en rango y la partición dinámica (Ponnusamy & Gupta, 2024). La elección de la técnica de partición adecuada depende de factores como el tipo de datos, la carga de trabajo y los requisitos de la aplicación (Ponnusamy & Gupta, 2024).

Sistemas SQL Escalables

Los sistemas SQL escalables, como Apache Drill, Apache Hive, Apache Spark SQL y Trino, permiten ejecutar consultas SQL en datos almacenados sistemas de ficheros distribuidos y con ayuda de Kubernetes, se simplifica la implementación y el escalado de estas aplicaciones (Cardas et al., 2023). Usando el benchmark TPC-H para evaluar el rendimiento de los sistemas SQL en un clúster de Hadoop con Kubernetes, Cardas y compañía en 2023 obtuvo resultados de un estudio comparativo mostrando que el rendimiento de los diferentes sistemas SQL puede variar significativamente en Kubernetes (Cardas et al., 2023).

La seguridad es un aspecto crítico en el análisis de Big Data, ya que se almacenan grandes cantidades de información sensible y los ataques de inyección SQL (SQLIA) son una

amenaza importante para las bases de datos, es fundamental contar con mecanismos para detectarlos y prevenirlos (Shareef et al., 2023).

Un método para detectar SQLIA es utilizar la regresión logística con la biblioteca Spark ML, el cual ha demostrado una alta precisión (99.04%) y un bajo tiempo de respuesta (0.05 s) en la detección de ataques (Shareef et al., 2023).

LotusSQL es un motor SQL de alto rendimiento para sistemas de Big Data que se basa en el motor de computación en paralelo Lotus y utiliza Calcite como front-end para el procesamiento de consultas (Li et al., 2021). Para optimizar el rendimiento de las consultas, LotusSQL utiliza técnicas como la fusión de operadores y la estimación de costos, los resultados muestran tiene un rendimiento superior al de Spark SQL en el benchmark TPC-H (Li et al., 2021).

Gestión de Recursos en la Nube

La gestión eficiente de recursos en entornos de nube es crucial para el procesamiento de Big Data(Lattuada et al., 2022). Las aplicaciones Spark, que se utilizan comúnmente para el análisis de Big Data, pueden requerir una cantidad significativa de recursos, y es importante optimizar su uso para minimizar los costos y maximizar el rendimiento (Lattuada et al., 2022). Lattuada y compañía en 2022 proponen políticas de optimización para la gestión de recursos en tiempo de ejecución de aplicaciones Spark en la nube, utilizando técnicas de predicción del rendimiento como el aprendizaje automático y la simulación.

La planificación de la capacidad es un aspecto crítico en la implementación de aplicaciones de Big Data en la nube (Gianniti et al., 2021). Es importante determinar la configuración óptima del clúster para minimizar los costos y cumplir con los requisitos de calidad de servicio (Gianniti et al., 2021). Gianniti y compañía en 2021 proponen una

herramienta que utiliza técnicas de simulación y optimización para resolver este problema. Los resultados experimentales muestran que la herramienta puede reducir significativamente los costos de las aplicaciones de Big Data en la nube sin comprometer el rendimiento.

Aplicaciones del Big Data

El análisis de Big Data se ha vuelto esencial en diversos campos, incluyendo el análisis del comportamiento humano en redes sociales (Tariq et al., 2021), el análisis de videos de tráfico para la detección de accidentes (Perafan-Villota et al., 2022), y la detección de noticias falsas en tiempo real (Babar et al., 2024). Para procesar eficientemente estos grandes volúmenes de datos, se utilizan sistemas de procesamiento en paralelo basados en frameworks como Hadoop y Spark (Perafan-Villota et al., 2022). Estos sistemas permiten dividir los datos en partes y analizarlas en paralelo, utilizando técnicas como las redes neuronales convolucionales (CNN) y los filtros de Kalman para la detección y seguimiento de objetos (Perafan-Villota et al., 2022).

Un ejemplo de la aplicación de Big Data Analytics en el análisis de datos a gran escala es el estudio de María Teresa Cuomo y compañia en 2023, que utiliza un conjunto de datos de casi 20 millones de personas ahorradoras para realizar una segmentación de clientes. En este estudio, se describe el proceso ETL (Extract, Transform, Load) para la recolección y preparación de los datos, y se utiliza el algoritmo K-means para la segmentación de los ahorradores (Cuomo et al., 2023). Los autores destacan la eficiencia y escalabilidad del algoritmo K-means para grandes conjuntos de datos y mencionan la posibilidad de implementar el algoritmo en frameworks como Hadoop y Spark (Cuomo et al., 2023).

El análisis de Big Data y el aprendizaje automático se están utilizando para mejorar los servicios médicos en el deporte (Zhao et al., 2024). Un ejemplo es el marco integrado propuesto por Zhao y compañía en 2024, que combina el análisis de Big Data basado en Spark con el

algoritmo XGBoost para la monitorización de la salud y la recomendación en deportes. Spark se utiliza para gestionar grandes volúmenes de datos generados durante el entrenamiento y las competiciones, mientras que XGBoost se utiliza para la minería de datos y la generación de recomendaciones (Zhao et al., 2024). Los resultados empíricos demuestran mejoras sustanciales en los servicios médicos deportivos mediante la aplicación del marco propuesto (Zhao et al., 2024).

Metodología

Este proyecto sigue la metodología CRISP-DM (Cross-Industry Standard Process for Data Mining) porque cuenta con un enfoque iterativo que se ajusta con flexibilidad a las diversas etapas del proyecto y que deja cabida a la adquisición de nuevos conocimientos. CRISP-DM se ajusta relativamente bien a proyectos de análisis de datos y minería de datos como este en que se pretende obtener información útil de grandes conjuntos de datos y proveer soluciones a problemas concretos.

Comprensión del Problema y Preparación de los Datos

Esta fase se centra en la compresión en profundidad del problema relacionados al tratamiento y normalización de datos para posterior análisis. La información entregada por la empresa de energía es de carácter estructurado y en formato desconocido, esta se encuentra dispersa en muchos archivos (ver ***Figura 1***).

Figura 1

Muestra de Archivos Iniciales

Nombre	Tipo	Tamaño
CATASTRO_USUARIOS_SV400_CERETE_ZONA_4_AS003_20241130	Archivo	1 KB
CATASTRO_USUARIOS_SV400_CERETE_ZONA_4_AS073_20241130	Archivo	6.281 KB
CATASTRO_USUARIOS_SV400_CERETE_ZONA_4_AS076_20241130	Archivo	1 KB
CATASTRO_USUARIOS_SV400_CERETE_ZONA_8_AS073_20241130	Archivo	3 KB
CATASTRO_USUARIOS_SV400_CERETE_ZONA_9_AS073_20241130	Archivo	1.482 KB
CONV_SV400_ECOSTAA_CORDOBA_CERETE4_AS073_20241130	Archivo	316 KB
CONV_SV400_ECOSTAA_CORDOBA_CERETE9_AS073_20241130	Archivo	56 KB
FACT_SV400_ECOSTAA_CORDOBA_CERETE4_AS003_20241130	Archivo	1 KB
FACT_SV400_ECOSTAA_CORDOBA_CERETE4_AS073_20241130	Archivo	12.294 KB
FACT_SV400_ECOSTAA_CORDOBA_CERETE8_AS073_20241130	Archivo	3 KB
FACT_SV400_ECOSTAA_CORDOBA_CERETE9_AS073_20241130	Archivo	2.374 KB
IC_ASEO_CERETE__CORDOBA__AS003_20241130	Archivo	3 KB
IC_ASEO_CERETE__CORDOBA__AS073_20241130	Archivo	50 KB
IC_SERVICIOS_CORDOBA_CERETE___AS003_20241130	Archivo	1 KB
IC_SERVICIOS_CORDOBA_CERETE___AS073_20241130	Archivo	3 KB
IC_SERVICIOS_CORDOBA_CERETE__20241130	Archivo	3 KB
NOVEDADES_CORDOBA_CERETE_20241130	Archivo	413 KB
NOVEDADES_CORDOBA_CERETE_AS073_20241130	Archivo	12 KB
RECA_SV400_ECOSTAA_CORDOBA_CERETE4_AS003_20241130	Archivo	1 KB
RECA_SV400_ECOSTAA_CORDOBA_CERETE4_AS073_20241130	Archivo	9.011 KB
RECA_SV400_ECOSTAA_CORDOBA_CERETE8_AS073_20241130	Archivo	3 KB
RECA_SV400_ECOSTAA_CORDOBA_CERETE9_AS073_20241130	Archivo	1.418 KB
REFA_SV400_ECOSTAA_CORDOBA_CERETE4_AS073_20241130	Archivo	125 KB
REFA_SV400_ECOSTAA_CORDOBA_CERETE9_AS073_20241130	Archivo	12 KB

Nota. Muestra de los archivos iniciales sin extensión definida entregada por la empresa de energía.

Transformación y Organización de Archivos

Utilizando el símbolo del sistema de Windows (CMD) se convierte en masa estos archivos a texto plano mediante un simple código mostrado en la ***Figura 2***. Este paso es necesario para poder integrar los datos fácilmente con Python o con cualquier otro programa, y así evitar conflictos por formatos desconocidos.

Figura 2

Conversión de Archivos a Formato Texto

Nota: Asignación de extensión de los archivos a txt por carpeta usando CMD.

Partiendo que la información es suministrada mensualmente y no se encuentra clasificada, la primera etapa consta de organizar los archivos según sus características en un formato de texto. Como la finalidad de este proyecto es el análisis de facturación y recaudo, se tendrá en cuenta los archivos que contengan los siguientes acrónimos:

- FACT: Facturación
- RECA: Recaudo
- REFA: Refacturación

Los archivos seleccionados son organizados por carpetas con el nombre respectivo de cada acrónimo.

Normalización y Homogenización de los Datos

En esta fase se pretende concatenar los archivos según sus características y crear un archivo único con el fin de tener mejor estructurada la información. Aplicando el lenguaje de programación Python con las bibliotecas de *Pandas*, *Glob* y *Path* se unifica y combina según criterios de facturación, refacturación y recaudo.

Seleccionando los archivos según sus cualidades, estos con almacenados en una carpeta para que por medio de un script iterativo se puedan concatenar los archivos que pertenezcan a dicha carpeta. La estructura de los 3 archivos seleccionados (facturación, refacturación y recaudo) presentan las mismas características y el mismo tipo de variables (ver ***Tabla 1***) a excepción de refacturación que presenta una menos (Consumo).

Tabla 1

Tipos de Variables Presentes en los Datos

Nombre de la columna	Tipo de dato
Fecha_actual	str
Co_operacion	object
Nic	str
Nis_rad	str
Sec_Nis	str
Sec_rec	str
F_FACT	str
Tip_Cli	object
TARIFA	object
EST_CONTRATO	str
CONCEPTO	object
Importe	float64
F_adicional	str
Valor_recibo	float64

Nombre de la columna	Tipo de dato
SECTOR	str
Codigo_municipio	str
Codigo_corregimiento	str
Codigo_departamento	str
Simbolo_Var	str
Consumo	object

Nota. Variables presentes en los archivos con su respectivo tipo de dato.

Las primeras lineas de codigo del script mostrado en la ***Figura 3*** hacen parte de la importacion de librerias y variables preliminares como la vigencia (año y mes donde pertenecen los archivos) y la ruta donde se encuentran los archivos. Seguido se encuetran los encabezados que serán asignados a cada dataframe y su respectivo tipo de variable.

La función llamada *cargar_archivos* desarrolla la concatenacion de cada archivo de la carpeta *(ruta)* de forma iterativa con el ciclo *for.* La funcion va a recorrer cada archivo teniendo en cuenta que en cada uno de ellos las columnas están delimitadas por tabulaciones, se le asignan los nombres de las columnas y los tipos de datos como se muestra en la **Tabla 1** con codificación 'utf-8' para la interpretación de caracteres. Adicionalmente a los datos presentes, se agrega una columna que indica el tipo de informacion del registro (RECA, FACT, REFA) y para que los datos de refacturacion contenga el mismo numero de columnas, se agrega "Consumo" con campos vacios.

Finalmente, de concatena facturación y refacturación para obtener un solo archivo que contenga la facturacion de cada usuario (FACT) con los ajustes tarifarios generados por diversos motivos (REFA).

Figura 3

Script de Concatenación de Archivos en Python

```python
import pandas as pd
import glob
from pathlib import Path

# Parámetros
vigencia = "202411"
ruta = Path(r'D:\202411\txt')

# Columnas y tipos de datos
columnas = [
    "Fecha_actual", "Co_operacion", "Nic", "Nis_rad", "Sec_Nis",
    "Sec_rec", "F_FACT", "Tip_Cli", "TARIFA", "EST_CONTRATO",
    "CONCEPTO", "Importe", "F_adicional", "Valor_recibo", "SECTOR",
    "Codigo_municipio", "Codigo_corregimiento", "Codigo_departamento",
    "Simbolo_Var", "Consumo"
]

dtypes = {
    "Fecha_actual": "str", "Co_operacion": "object", "Nic": "str",
    "Nis_rad": "str", "Sec_Nis": "str", "Sec_rec": "str",
    "F_FACT": "str", "Tip_Cli": "object", "TARIFA": "object",
    "EST_CONTRATO": "str", "CONCEPTO": "object", "Importe": "float64",
    "F_adicional": "str", "Valor_recibo": "float64", "SECTOR": "str",
    "Codigo_municipio": "str", "Codigo_corregimiento": "str",
    "Codigo_departamento": "str", "Simbolo_Var": "str", "Consumo": "object"
}

def cargar_archivos(ruta_patron, encoding='utf-8', add_column=None, drop_empty=True):
    """Carga archivos de texto, aplica nombres de columnas y tipos de datos."""
    archivos = glob.glob(str(ruta_patron))
    dataframes = []

    for archivo in archivos:
        df = pd.read_csv(archivo, sep='\t', names=columnas, dtype=dtypes, encoding=encoding,
low_memory=False)

        if add_column:
            df["Tipo"] = add_column  # Agrega la columna con el tipo de archivo

        dataframes.append(df)

    return pd.concat(dataframes, ignore_index=True) if dataframes else
pd.DataFrame(columns=columnas)

# Carga de archivos
df_reca = cargar_archivos(ruta / "RECA" / "RECA*.txt", add_column="RECA")
df_prefact = cargar_archivos(ruta / "FACT" / "FACT*.txt", add_column="FACT")
df_refa = cargar_archivos(ruta / "REFA" / "REFA*.txt", encoding='latin1', add_column="REFA")

# Ajuste especial para df_refa
df_refa["Consumo"] = ""  # Agregar columna "Consumo"
df_refa = df_refa.astype(dtypes, errors='ignore')  # Convertir tipos sin errores

# Concatenar Prefacturación (Facturación inicial) y Refacturación para obtener la Facturación final
df_fact = pd.concat([df_prefact, df_refa])

# Guardar archivos procesados
df_reca.to_csv(ruta / f"db_Reca_{vigencia}.csv", index=False)
df_fact.to_csv(ruta / f"db_Fact_{vigencia}.csv", index=False)

print("¡Script Ejecutado!")
```

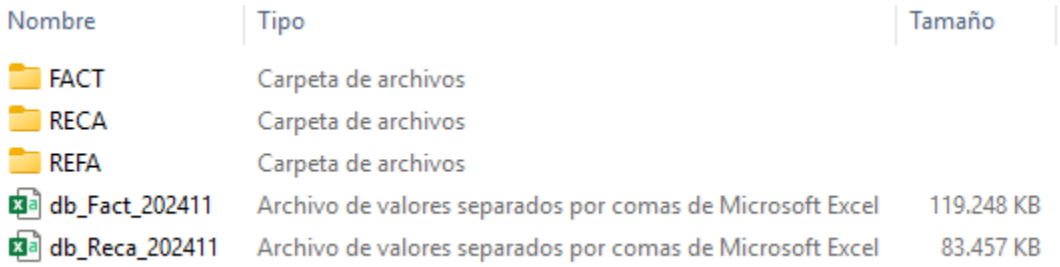
Nota: Procedimiento iterativo en Python para la concatenación de los archivos txt transformándolos en un solo archivo csv teniendo en cuenta criterios de FACT, RECA, REFA.

Los archivos csv obtenidos como resultado del script se observan en la ***Figura 4***, este proceso se aplica para cada fecha de recepción (mensual) y es compartido a las partes interesadas por medio de una carpeta de *SharePoint*. Se toma la decisión de generar los archivos con

extensión *csv* para que otros empleados puedan acceder a los datos fácilmente sin necesidad de tener conocimientos en programación o de programas de gestión de datos en específico.

Figura 4

Contenido de la Carpeta con la Concatenación en Archivos csv



Nota: Muestra ilustrativa de los resultados obtenidos del script para la concatenación de los archivos txt.

Este ciclo de ETL (Extract, Transform and Load) ayuda a organizar, normalizar la información y a reducir miles de archivos en tres tablas principales que representen la información reportada por la empresa de energía. Con esta normalización se tendrán a disposición un conjunto de datos de gran volumen que serán la primicia para las siguientes etapas del proyecto.

Exploración de Frameworks de Big Data

Dado que el análisis de grandes volúmenes de datos es computacionalmente costoso para equipos convencionales, se requiere la aplicación de técnicas de Big Data, en este sentido, se explorarán los frameworks más populares con un enfoque en el análisis de datos estructurados. A partir de una revisión de bibliografía se comparan las fortalezas y las debilidades de cada una de las herramientas de Big Data para analizar cómo se ajusta cada una a la situación y poder evaluar la mejor a partir de las condiciones y recursos de la empresa.

En la actualidad, la producción en serie de datos ha modificado cómo las empresas y las instituciones operan con información. Al aumentar la popularidad de las tecnologías en Big Data, se volvió necesario utilizar herramientas que puedan procesar, analizar y obtener conocimiento de grandes sets de datos de manera eficaz. A tal propósito se hace necesaria la evaluación y consideración de las herramientas más idóneas para aplicabilidad empresarial y otras índoles.

El caso en consideración se enfoca en la transformación y análisis en masa en una empresa de aseo domiciliario donde la exactitud y eficiencia en procesamiento de datos desempeñan un papel crucial. Emplear de una herramienta de Big Data adecuada garantizará que el sistema tenga la capacidad de procesar volúmenes de información extensos, realizar análisis en un corto tiempo y optimizar el rendimiento computacional. Para tal fin se comparan diversas tecnologías con base en criterios como escalabilidad, eficiencia computacional, tolerancia a fallas y facilidad de integración con infraestructuras empresariales con entornos tradicionales (Ketu et al., 2020; Khalid & Yousaf, 2021).

A lo largo de los últimos años, las arquitecturas Big Data se han ido desarrollando desde las soluciones de disco basadas en Hadoop MapReduce hacia arquitecturas optimizadas en memoria (RAM) como Apache Spark. Aunque Hadoop señaló una de las primeras soluciones del procesamiento distribuido de datos, su modelo de lectura y escritura en disco ha sido superada en rendimiento por tecnologías más recientes (Montoya Suárez & Gil Restrepo, 2018). En este trabajo se busca analizar y comparar estos frameworks para definir cuál de estos ofrece la mejor solución para el caso de estudio.

Importancia del Análisis Comparativo

Comparar frameworks de Big Data es crucial para la selección de la tecnología más adecuada en función del entorno y del tipo de datos que se van a procesar. Cada una de ellas

cuenta con características que la convierten en más eficiente para ciertas aplicaciones. Por ejemplo, Apache Spark es una buena opción en aquellos casos en que se necesita procesamiento en memoria de manera rápida, mientras que Apache Flink se debe usar en aplicaciones que necesitan análisis en tiempo real con bajo nivel de latencia (Gunay et al., 2019; Inoubli et al., 2018).

Es crucial tener un framework que se pueda escalarse eficazmente. Tecnologías como PrestoDB y Apache Spark pueden manipular grandes volúmenes de información en entornos distribuidos con buen rendimiento incluso con volúmenes de datos que alcanzan petabytes (Li et al., 2021). Por otro lado, herramientas como Apache Hive están desarrolladas específicamente para consultas SQL en grandes conjuntos de datos almacenados en Hadoop, sin embargo, los tiempos de procesamiento también suelen ser altos, al igual que Hadoop, estos están basados en un modelo persistente de almacenamiento en disco (Cardas et al., 2023).

Algo a considerar es la interacción con infraestructuras en la nube. La mayoría de las empresas prefieren emplear sistemas centrales basados en AWS, Azure o Google Cloud para gestionar el procesamiento y almacenamiento óptimo de datos. Al respecto, los frameworks Apache Spark y Delta Lake brindan una integración nativa con la nube, lo que resulta en la implementación accesible y mayor flexibilidad en la administración de datos (Lattuada et al., 2022). Este hecho es crítico para el proyecto en cuestión, ya que la solución optada debe implementarse en un entorno donde no se comprometa el costo computacional y la instalación en las nubes ofrecen un amplio margen de configuración a costos moderados, ajustándose de acuerdo con las demandas del usuario.

Propósito del Análisis

Una de las líneas de investigación de este proyecto es identificar cuál de los entornos Big Data se ajusta a las condiciones de los datos de facturación y recaudo en la empresa ofreciendo una gran capacidad de procesamiento y de tratamiento. El propósito del framework consta de poder implementar adecuadamente las 5V´s del Big Data (Volumen, Velocidad, Veracidad, Variedad, Valor) y para ello se hará un estudio comparativo evaluando el rendimiento, escalabilidad e integración en sistemas locales, en la nube o mixtos.

Dentro de los criterios de evaluación para los frameworks de estudio se tendrá en cuenta los recursos y los tipos de datos (estructurados) presentes en la empresa.

Como primicia se tendrá en cuenta el modelo de procesamiento de cada framework validando su capacidad de procesamiento en batch (lote), streaming (tiempo real) o ambos con el propósito de ofrecer resultados en tiempos adecuados. Un aspecto crucial es la compatibilidad a formatos como txt, csv, json o Parquet ya que la información se encuentra almacenada en archivos y no en base de datos SQL. De la misma forma se tendrá en cuenta la eficiencia en el uso de recursos en hardware de cada ambiente, debido a lo computacionalmente costo que puede ser trabajar con datos masivos, es necesario tener en cuenta este aspecto por las limitaciones en insumos y equipo computacional. Sin dejar cavidad a fallos, el framework debe ser tolerante a ellos manejándolos eficientemente con la seguridad de que se puedan recuperar datos en casos extremos. Como la escalabilidad es un factor presente en los datos, se debe estudiar la compatibilidad en plataformas en la nube como AWS, Azure, Google Cloud u Oracle evitando que los procesos se vean comprometidos por las limitantes del equipo local en un futuro.

La correcta selección del framework asegurará una eficaz ejecución de las actividades de conciliación de facturación y una reducción de costos computacionales y optimización del uso de recursos (Belov et al., 2021; Ullah et al., 2024).

El análisis de frameworks de Big Data permite identificar cuál de ellos se adapta mejor a los requisitos del proyecto. Evaluando varios frameworks disponibles para tratar información estructurada, se busca desarrollar un comparativo de cada herramienta teniendo en cuenta los criterios de mencionados anteriormente los cuales constan del procesamiento, escalabilidad, eficiencia, tolerancia a fallos y compatibilidad en la nube.

Apache Spark es un framework el cual se adapta fácilmente ya que permite modelar por lotes y en streaming, además, es compatible con varios lenguajes de programación como Python, Java y Scala permitiendo su implementación en diferentes proyectos con trata de datos. Este framework es utilizado hoy en día por la comunidad por su buen rendimiento en procesamiento de información, con ayuda del almacenamiento en memoria RAM, los datos van a mayor velocidad en comparación del disco como lo utiliza Hadoop con MapReduce (Tekdogan & Cakmak, 2022). Spark se consolida como una de las mejores debido a su eficiencia en el procesamiento en memoria, su capacidad de escalabilidad y su integración con servicios en la nube (Ketu et al., 2020; Khalid & Yousaf, 2021; Montoya Suárez & Gil Restrepo, 2018).

En cuanto al procesamiento de datos en tiempo real se tiene frameworks como Apache Flink donde una de sus fortalezas es trabajar en baja latencia, estas características lo convierten en un recurso valioso en ambientes críticos con respuestas inmediatas (Gunay et al., 2019). En comparación con Apache Spark, Flink es altamente reactivo en circunstancias donde se requieran resultados al instante, sin embargo, la configuración respecto a su instalación y mantenimiento

son muy exigentes y requieren de más conocimiento respecto a otros framewoks más amigables al usuario (Jaime & Hasperué, 2021).

Teniendo en cuenta que la información es estructurada y en formato tabular, utilizar herramientas que sean amigables con motor SQL es una opción tentativa dentro de las características que deba tener el framework. Dentro de este perfil se nombran ambientes como PrestoDB y ClickHouse, estos resultan ser intuitivos dentro del ecosistema Big Data gracias a su familiar motor de consulta (Zeydan & Mangues-Bafalluy, 2022) sin dejar de lado los tiempos de respuesta para exploración y analítica de datos (Belov et al., 2021; Cardas et al., 2023; Li et al., 2021). Sin embargo, su alcance es limitado en términos de procesamiento distribuido y escalabilidad en comparación con soluciones como Spark o Flink (Gu et al., 2023; Mostafa et al., 2022).

Apache Hive sigue siendo una opción relevante para entornos basados en Hadoop, proporcionando una capa de abstracción SQL sobre HDFS (Hadoop Distributed File System). Sin embargo, su alto consumo de recursos y tiempos de respuesta más lentos en comparación con tecnologías más recientes lo hacen menos eficiente para aplicaciones de Big Data que requieren procesamiento ágil y escalable (Cardas et al., 2023).

En términos de almacenamiento y procesamiento distribuido, Hadoop MapReduce sigue siendo relevante para tareas donde la persistencia y tolerancia a fallos son primordiales. A pesar de su menor eficiencia en comparación con Spark, sigue siendo una opción viable en entornos donde la confiabilidad y la consistencia de los datos son más importantes que la velocidad de procesamiento (Ullah et al., 2024).

La elección de Apache Spark como framework principal se justifica por su equilibrio entre eficiencia, escalabilidad y compatibilidad con diversos entornos como se observa en la

Tabla 2. Su adopción en la industria sigue creciendo, respaldada por una amplia comunidad de desarrolladores y una evolución continua de la plataforma. Su capacidad de integrarse con servicios en la nube y su rendimiento optimizado para análisis de datos estructurados lo convierten en la mejor opción para el tratamiento y análisis de datos masivos en una empresa de aseo domiciliario, con su implementación se garantiza un procesamiento eficiente, rápido y escalable para en este tipo de proyectos (Lattuada et al., 2022; Ullah et al., 2024).

Tabla 2

Comparativo Frameworks

Framework	Procesamiento	Tipo de Datos	Escalabilidad	Tolerancia	Lenguajes	Eficiencia	Nube	Uso
Apache Spark[a,b,c]	Batch y Streaming	TXT, CSV, Parquet	Alta (Clúster/Nube)	Alta	Python, Scala, Java	Procesamiento en memoria	AWS, Azure, GCP, Databricks	ETL, ML, Análisis Masivo
Apache Flink[b,d]	Streaming y Batch	TXT, CSV, JSON	Alta (Tiempo Real)	Alta	Java, Scala, Python	Procesamiento distribuido	AWS Kinesis, GCP Dataflow	Procesamiento en Tiempo Real
PrestoDB[e,f]	SQL Distribuido	CSV, JSON, HDFS	Alta	Media	SQL	Optimizado para consultas SQL	AWS Athena, Azure Synapse	Consultas SQL Interactivas
Apache Hive[g]	Batch (SQL en Hadoop)	TXT, CSV en HDFS	Alta	Alta	SQL	Alto consumo de recursos	AWS EMR, Azure HDInsight	ETL y Reportes en Hadoop

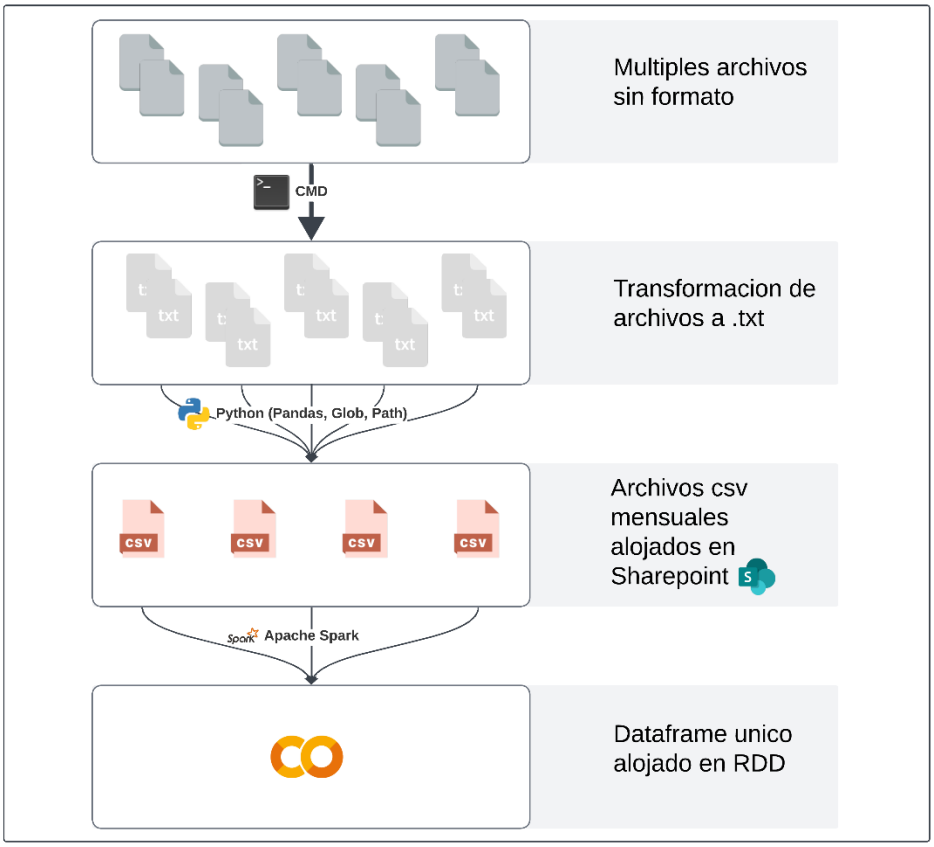
Framework	Procesamiento	Tipo de Datos	Escalabilidad	Tolerancia	Lenguajes	Eficiencia	Nube	Uso
Delta Lake[h]	Batch y Streaming	Parquet, JSON	Alta	Alta	Scala, Java, Python	ACID Transactions sobre Spark	AWS, Azure, Databricks	Data Lakes y Gestión de Datos
ClickHouse[i]	OLAP (Análisis)	TXT, CSV, Parquet	Alta	Media	SQL	Ultra rápido para consultas analíticas	ClickHouse Cloud, AWS, Azure	Análisis OLAP en tiempo real
Apache Hadoop (MapReduce)[j]	Batch	TXT, CSV, JSON	Alta	Alta	Java	Procesamiento en disco basado en MapReduce	AWS EMR, Azure HDInsight	Procesamiento distribuido a gran escala

Nota. Evaluación comparativa de los frameworks que mejor de adapten al conjunto de datos y a los recursos de la empresa. Adaptado de [a]Khalid & Yousaf, 2021; [b]Ketu et al., 2020; [c]Montoya Suarez & Gil Restrepo, 2018; [d]Inoubli et al., 2018; [e]Gunay et al., 2022; [f]Li et al., 2021; [g]Cardas et al., 2023; [h]Lattuada et al., 2022; [i]Belov et al., 2021; [j]Ullah et al., 2024

Utilizando Spark como el framework seleccionado, se completa el proceso de preparación de los datos como se muestra en la ***Figura 5***. La fase final del esquema consta del uso de Spark para la concatenación de los archivos csv en un dataframe de Spark y este es almacenado en Resilient Distributed Dataset (RDD) contemplando la computación en servicios en la nube.

Figura 5

Esquema Ilustrativo de la Preparación de los Datos



Nota: Descripción ilustrativa de la transformación de los datos por fases con las herramientas utilizadas.

Modelado y Evaluación

Acceso a los Datos Alojados en SharePoint

Para obtener acceso a los archivos csv almacenados en SharePoint, se requieren de credenciales que permitan lectura y escritura de los datos desde Python. Utilizando una aplicación dedicada a la seguridad y permisos de usuarios llamada *Microsoft Entra,* se crea en primera instancia una aplicación la cual será la conexión entre el script y los servicios de Microsoft (Subbarao et al., 2023). La aplicación creada con nombre "*Python SharePoint API"* suministra lo siguiente:

- **CLIENT_ID:** Identificador único de la aplicación creada en Microsoft Entra el cual permite que Microsoft sepa desde que aplicación se está haciendo la autenticación.
- **TENANT_ID:** Identifica el directorio de Microsoft Entra el cual indica desde que inquilino (Organización) pertenece la autorización.
- **CLIENT_SECRET:** Es una clave generada automáticamente desde la aplicación *Python SharePoint API* que permite autenticarse de forma privada y segura.

Con estos identificadores se obtiene el token de acceso usando *Client Credencials Flow* (autenticación servidor a servidor sin usuario) como se observa en la ***Figura 6***.

Una vez el token es almacenado en una variable, este se utiliza para construir los encabezados de la solicitud HTTP enviando un *Bearer Token* con el *Dominio de SharePoint* y el *Sitio* al cual se necesita el acceso. Usando una petición GET a la URL del sitio de SharePoint, devuelve como respuesta un JSON con información sobre el sitio, como su ID, nombre, URL y fecha de creación si la autenticación es exitosa y los permisos son adecuados (***Figura 7***).

Una vez el token es almacenado en una variable, este se utiliza para construir los encabezados de la solicitud HTTP enviando un *Bearer Token* con el *Dominio de SharePoint* y el

Sitio al cual se necesita el acceso. Usando una petición GET a la URL del sitio de SharePoint, devuelve como respuesta un JSON con información sobre el sitio, como su ID, nombre, URL y fecha de creación si la autenticación es exitosa y los permisos son adecuados (***Figura 7***).

Figura 6

Token de Acceso en Microsoft Graph

```
5

import os
import msal
import requests
from dotenv import load_dotenv

# Copiar el archivo .env desde DBFS a un directorio temporal accesible
dbutils.fs.cp("dbfs:/FileStore/tables/.env", "file:/tmp/.env", True)

# Cargar variables desde el archivo copiado
load_dotenv("/tmp/.env")

# Datos de la aplicación en Microsoft Entra
CLIENT_ID = os.getenv("CLIENT_ID")
TENANT_ID = os.getenv("TENANT_ID")
CLIENT_SECRET = os.getenv("CLIENT_SECRET")

AUTHORITY = f"https://login.microsoftonline.com/{TENANT_ID}"
SCOPES = ["https://graph.microsoft.com/.default"]

# Autenticación con Client Secret
app = msal.ConfidentialClientApplication(CLIENT_ID, authority=AUTHORITY, client_credential=CLIENT_SECRET)
token = app.acquire_token_for_client(SCOPES)

if "access_token" in token:
    access_token = token["access_token"]
    print("✅ Token de acceso obtenido correctamente")
else:
    print("❌ Error al obtener el token:", token.get("error_description"))

✅ Token de acceso obtenido correctamente
```

Nota: AUTHORITY es la URL de autenticación para el inquilino y *SCOPES* define los permisos para acceder a *Microsoft Graph API.*

Figura 7

Acceso al Sitio de SharePoint

```
6

SHAREPOINT_DOMAIN = os.getenv("SHAREPOINT_DOMAIN")
SHAREPOINT_SITE = os.getenv("SHAREPOINT_SITE")

site_url = f"https://graph.microsoft.com/v1.0/sites/{SHAREPOINT_DOMAIN}:/sites/{SHAREPOINT_SITE}"
headers = {"Authorization": f"Bearer {access_token}"}

response = requests.get(site_url, headers=headers)
#print(response.status_code, response.text)
site_data = response.json()
print("✅ Información del site obtenida!")

✅ Información del site obtenida!
```

Nota: Obtención del ID del sitio para acceder a la biblioteca de documentos de SharePoint.

Con la información obtenida del sitio, se accede a la biblioteca de documentos (drive) con el ID del sitio y armando la URL para hacer la solicitud HTTP. Con respuesta exitosa, la consulta arroja un diccionario JSON que muestra las bibliotecas de documentos disponibles en el sitio.

Figura 8

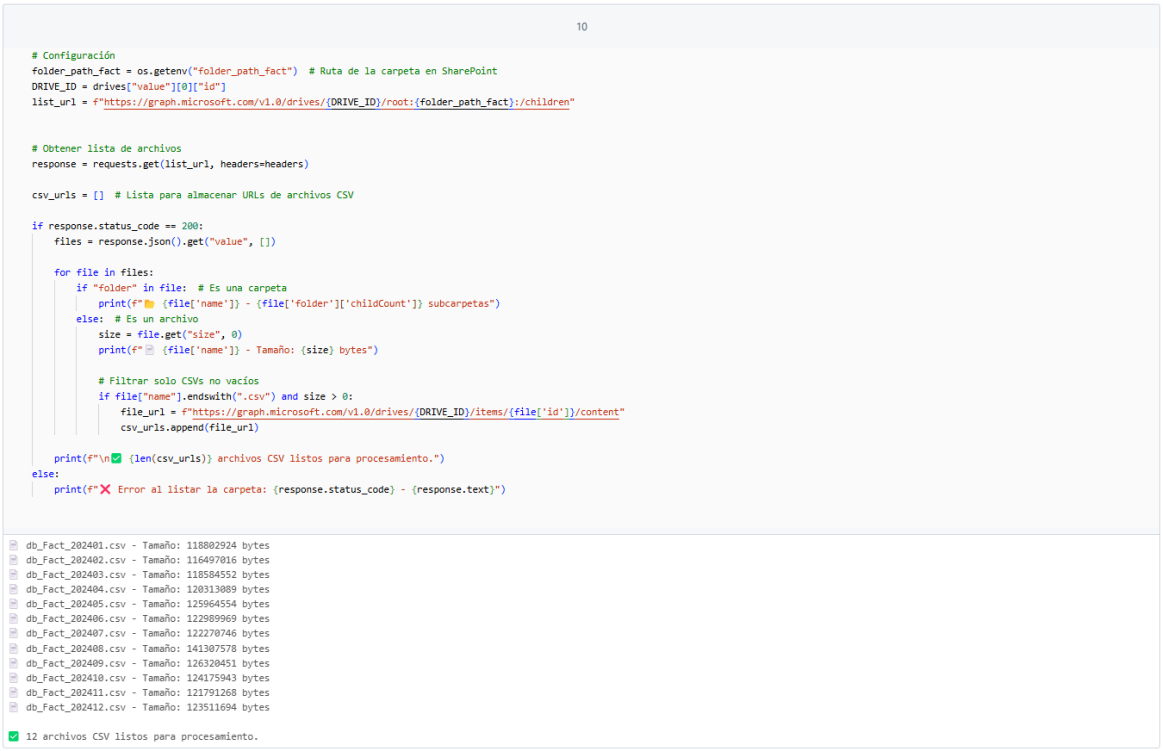
Acceso a los Documentos del Sitio en SharePoint

Nota: Acceso al ID de la biblioteca de documentos para obtener las URL de los archivos csv.

Por último, usando el ID de la biblioteca de documentos, se crea la URL con la *ruta de la carpeta* en la cual se hará la solicitud GET para obtener la lista de archivos. Con una respuesta exitosa, se guarda la información suministrada por un JSON en una lista llamada *csv_urls* que contenga la URL de cada archivo y posteriormente se imprime cada nombre con su tamaño correspondiente. Este método se utiliza tanto para las carpetas de facturación como recaudo.

Figura 9

Obtención de las URL de Cada Archivo Alojado en SharePoint



```
10
```

```
# Configuración
folder_path_fact = os.getenv("folder_path_fact")  # Ruta de la carpeta en SharePoint
DRIVE_ID = drives["value"][0]["id"]
list_url = f"https://graph.microsoft.com/v1.0/drives/{DRIVE_ID}/root:{folder_path_fact}:/children"


# Obtener lista de archivos
response = requests.get(list_url, headers=headers)

csv_urls = []  # Lista para almacenar URLs de archivos CSV

if response.status_code == 200:
    files = response.json().get("value", [])

    for file in files:
        if "folder" in file:  # Es una carpeta
            print(f"📁 {file['name']} - {file['folder']['childCount']} subcarpetas")
        else:  # Es un archivo
            size = file.get("size", 0)
            print(f"📄 {file['name']} - Tamaño: {size} bytes")

            # Filtrar solo CSVs no vacíos
            if file["name"].endswith(".csv") and size > 0:
                file_url = f"https://graph.microsoft.com/v1.0/drives/{DRIVE_ID}/items/{file['id']}/content"
                csv_urls.append(file_url)

    print(f"\n✅ {len(csv_urls)} archivos CSV listos para procesamiento.")
else:
    print(f"❌ Error al listar la carpeta: {response.status_code} - {response.text}")
```

```
📄 db_Fact_202401.csv - Tamaño: 118802924 bytes
📄 db_Fact_202402.csv - Tamaño: 116497016 bytes
📄 db_Fact_202403.csv - Tamaño: 118584552 bytes
📄 db_Fact_202404.csv - Tamaño: 120313089 bytes
📄 db_Fact_202405.csv - Tamaño: 125964554 bytes
📄 db_Fact_202406.csv - Tamaño: 122989969 bytes
📄 db_Fact_202407.csv - Tamaño: 122270746 bytes
📄 db_Fact_202408.csv - Tamaño: 141307578 bytes
📄 db_Fact_202409.csv - Tamaño: 126320451 bytes
📄 db_Fact_202410.csv - Tamaño: 124175943 bytes
📄 db_Fact_202411.csv - Tamaño: 121791268 bytes
📄 db_Fact_202412.csv - Tamaño: 123511694 bytes

✅ 12 archivos CSV listos para procesamiento.
```

Nota: Se extrae la URL de cada archivo y se almacena en una variable llamada *csv_urls.*

Implementación

Con la importación de los archivos almacenados en SharePoint, se inicia la etapa de lectura de los datos en un solo dataframe. Este proceso radica en concatenar todos los archivos csv y para su desarrollo se crea en primera instancia una función llamada *procesar_csv* donde se descarga cada archivo desde la URL con una solicitud GET, se lee el csv en memoria con pandas y se transforma a un dataframe de spark.

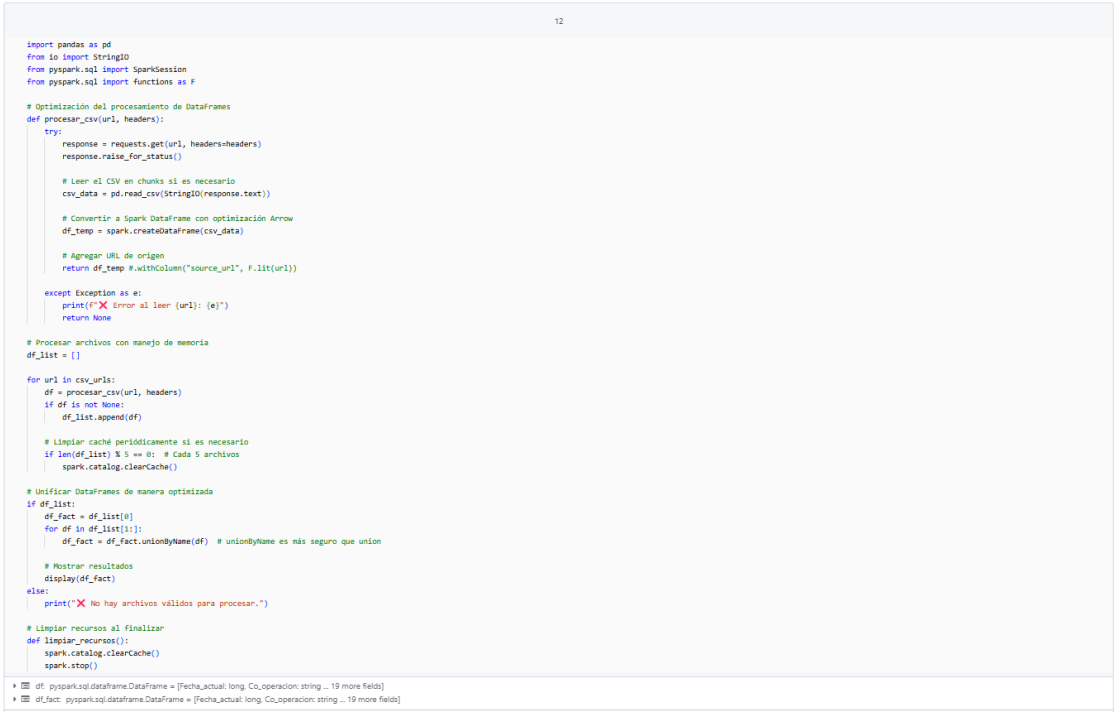
Usando un ciclo *for* para iterar la agregación de cada dataframe en una lista *df_list*, en este se llama a la función *procesar_csv* teniendo en cuenta que las URL de cada archivo fueron obtenidas en el paso anterior almacenándose en la variable *csv_urls* (**Figura 9**). Además, para

optimizar el uso de memoria, cada vez que la lista alcanza 5 elementos, se limpia la caché de Spark con *spark.catalog.clearCache.*

Por último, se toma el primer dataframe como guía y se unen los demás usando *unionByName.* Estos pasos se pueden observar en el código mostrado en la ***Figura 10***.

Figura 10

Importación de los Datos de Facturación



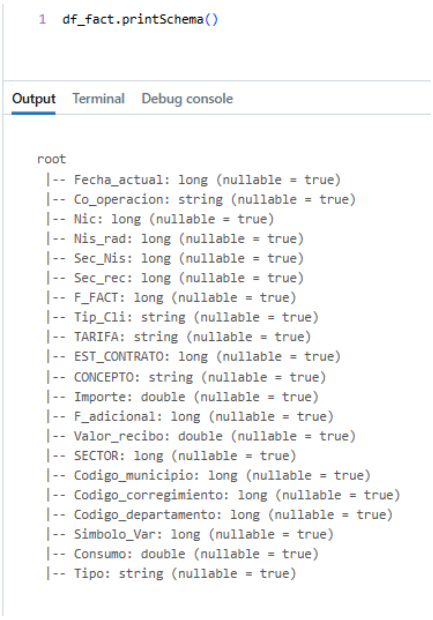
```
12

import pandas as pd
from io import StringIO
from pyspark.sql import SparkSession
from pyspark.sql import functions as F

# Optimización del procesamiento de DataFrames
def procesar_csv(url, headers):
    try:
        response = requests.get(url, headers=headers)
        response.raise_for_status()

        # Leer el CSV en chunks si es necesario
        csv_data = pd.read_csv(StringIO(response.text))

        # Convertir a Spark DataFrame con optimización Arrow
        df_temp = spark.createDataFrame(csv_data)

        # Agregar URL de origen
        return df_temp #.withColumn("source_url", F.lit(url))

    except Exception as e:
        print(f"❌ Error al leer {url}: {e}")
        return None

# Procesar archivos con manejo de memoria
df_list = []

for url in csv_urls:
    df = procesar_csv(url, headers)
    if df is not None:
        df_list.append(df)

    # Limpiar caché periódicamente si es necesario
    if len(df_list) % 5 == 0:  # Cada 5 archivos
        spark.catalog.clearCache()

# Unificar DataFrames de manera optimizada
if df_list:
    df_fact = df_list[0]
    for df in df_list[1:]:
        df_fact = df_fact.unionByName(df)  # unionByName es más seguro que union

    # Mostrar resultados
    display(df_fact)
else:
    print("❌ No hay archivos válidos para procesar.")

# Limpiar recursos al finalizar
def limpiar_recursos():
    spark.catalog.clearCache()
    spark.stop()

▸ df: pyspark.sql.dataframe.DataFrame = [Fecha_actual: long, Co_operacion: string ... 19 more fields]
▸ df_fact: pyspark.sql.dataframe.DataFrame = [Fecha_actual: long, Co_operacion: string ... 19 more fields]
```

Nota: Concatenación de los csv de facturación a un dataframe.

De la misma manera como se ve en la ***Figura 10*** se transforman los datos de recaudo en un dataframe de Spark y se obtienen como resultado las métricas iniciales observadas en la ***Figura 12***.

Figura 11

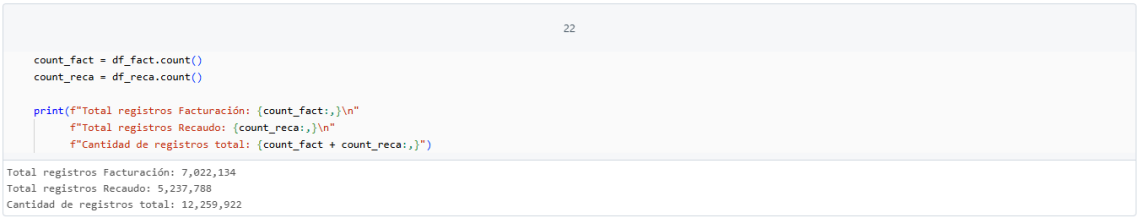
Esquema de Variables Presentes en el Dataframe de Facturación



Nota: Muestra de las variables de db_fact.

Figura 12

Tamaño de los Dataframes de Facturación y Recaudo



Nota: Cantidad de registros relacionados a la información ofrecida por la empresa de energía en el año 2024 con separadores de miles.

Para posteriores análisis se unifican los datasets como se observa en ***Figura 13***, manteniendo variables relevantes.

Figura 13

Concatenación de Datasets

```
# Concatenar FACTURACIÓN y RECAUDO
df_conjunta = df_fact.unionByName(df_reca)

# FACTURACION y RECAUDO

variables = ["Vigencia", "Nic", "Nis_rad", "Tip_Cli", "TARIFA",
             "EST_CONTRATO", "SECTOR", "Codigo_municipio",
             "Codigo_corregimiento", "Codigo_departamento",
             "Simbolo_Var"
            ]

df_gconjunta = df_conjunta.groupBy(variables).pivot("Tipo", ["FACT", "RECA", "REFA"]) \
               .agg(sum("Importe"))


```

Nota: Concatenación de los datasets sin las variables Fecha_actual, Sec_Nis, Sec_rec, F_FACT, CONCEPTO, F_adicional, Valor_recibo.

Adicionalmente se aplicaron transformaciones al dataframe df_conjunta donde se omitieron registros anteriores a la vigencia 202401 ya que el caso de estudio consta de 2024. Si bien es cierto que existen datos de recaudo o refacturaciones de los usuarios de facturas antiguas pagadas en 2024, la información de facturación para esas fechas no está presente lo que puede causar desbalance en los tipos de datos y fallos en métricas calculadas.

Aplicando la agrupación por la variable Tipo es posible calcular nuevas medidas como CARTERA para identificar los saldos pendientes de cada factura por usuario, el método de calcular esta nueva variable parte de usar FACT + REFA – RECA, donde básicamente se puede resumir como cobro menos pagos.

El comportamiento ideal de pagos por los usuarios debería ser que cancelen el total facturado por el mes, sin embargo, hay casos donde el recaudo recibido por el usuario es mayor a

su facturación y por ende la CARTERA es negativa, en este caso omitiremos esos registros donde el recaudo es mayor que la facturación (ver ***Figura 14***).

Teniendo en cuenta que Simbolo_Var es un identificativo único para las facturas de cada usuario, donde los primeros 7 dígitos son el Nic (identificativo usuario) más un secuencial de 3 dígitos al final, se detectaron valores duplicados para este identificativo con un total de 20 registros, al ser una cantidad mínima se descartan igualmente del dataset.

Figura 14

Detección de Duplicados por Simbolo_Var

```
[ ] print(f"Cantidad de usuarios con saldo a favor: {df_saldo_favor.count()}")

Cantidad de usuarios con saldo a favor: 805

from pyspark.sql.functions import count
df_saldo_favor.groupBy("VIGENCIA").agg(count("*").alias("Conteo")).show()

+--------+------+
|VIGENCIA|Conteo|
+--------+------+
|  202401|    23|
|  202402|    26|
|  202409|    30|
|  202408|    15|
|  202405|   570|
|  202404|    21|
|  202411|     8|
|  202410|    11|
|  202403|    26|
|  202406|    47|
|  202407|    27|
|  202412|     1|
+--------+------+
```

Nota: Se detectaron un total de 805 registros con saldo a favor una fracción minina (0.04% < 1%).

En última instancia se crea una variable booleana llamada SALDO, si el valor es 1 quiere decir que el usuario pago la factura, si es 0 tiene saldo pendiente por pagar.

Resultados

Análisis Descriptivo

El análisis descriptivo es el primer paso para entender de manera ordenada cómo se comportan los usuarios del servicio. En esta etapa se revisaron tanto las variables numéricas como las categóricas, con el propósito de conocer su distribución, valores centrales, dispersión y si existen datos fuera de lo común.

Se utilizaron métricas estadísticas y visuales como histogramas, curvas de densidad, gráficos de violín y tablas de frecuencia. estas métricas permiten identificar puntos críticos, como altos niveles de morosidad, usuarios que no pagan con frecuencia o zonas donde se concentra un gran número de registros según el tipo de cliente.

Estadísticas Descriptivas y Percentiles de Variables Numéricas

Este análisis descriptivo de las variables numéricas permitió interpretar en detalle el comportamiento de los usuarios con el servicio como se nota en la ***Figura 15***. En primer lugar, la variable FACT tuvo una distribución amplia, con una mediana de $14.310 y un percentil 75 de $21.420, y esto explica el hecho que la mayoría de los usuarios tiene un nivel moderado del mismo. Aun así, se apreciaron valores atípicos importantes, con un límite inferior de -$1.519.620 y un límite superior de $3.085.810. Dichas cantidades extremas se podrían atribuir a error de digitación, diferencias contables revertidas, o a casos poco usuales de endeudamientos.

En cuanto a la variable RECA, que se refiere a la cantidad efectivamente recaudada por usuario en cada mes, los datos registraron una clara conglomeración en valores bajos. El 25% de los usuarios no hizo pagos durante el periodo analizado (percentil 25 = 0) y la mediana se encontró en solo $9.190. Si bien hubo algunos usuarios que efectuaron pagos mayores al millón de pesos, la relación media-mediana revela una distribución sesgada, prevaleciendo un gran

grupo de usuarios con pagos nulos o bajos y un conjunto que, siendo menor, tiene aportes notablemente altos que distorsiona el promedio.

Por otro lado, la variación REFA encuentra que los percentiles 25, 50 y 75 fueron todos equivalentes a cero, ya que, en la mayoría de las observaciones, no se hicieron estimaciones. Sin embargo, el intervalo total era significativo, con valores que iban del - $413.980 hasta el + $413.900. Esto se explica porque, aunque son aislados, las estimaciones pueden tener un peso significativo en el monto final del endeudamiento algunos usuarios.

Finalmente, se estudió la variable CARTERA, como la diferencia entre el facturado (con los ajustes) y el efectivamente recaudado, se utiliza para estimar el saldo pendiente a la clausura del mes. Por lo menos el 50% de los clientes no tenía deuda (mediana = 0), la cual es un signo positivo del pago. Sin embargo, el percentil 75% con el mayor nivel de deuda rebasaba los $7.540. Las máximas coinciden con las cuantías máximas facturadas, verificando que para esta factura no hubo recaudo.

Figura 15

Descripción de Variables Financieras

```
Total de registros: 1785633
+-------+------------------+------------------+------------------+-----------------+
|summary|              FACT|              RECA|              REFA|          CARTERA|
+-------+------------------+------------------+------------------+-----------------+
|  count|           1785633|           1785633|           1785633|          1785633|
|   mean| 17898.02531651241|12408.995436038645|-8.001280218275536|5481.018733412745|
| stddev|21884.755250477312|17447.070413021313| 2216.809814525685|17233.52898429241|
|    min|        -1519620.0|        -1519620.0|         -413900.0|              0.0|
|    max|         3085810.0|         1354910.0|          413900.0|          3085810|
+-------+------------------+------------------+------------------+-----------------+

Percentiles de FACT: 25%=8680.0, 50%=14310.0, 75%=21420.0
Percentiles de RECA: 25%=0.0, 50%=9190.0, 75%=19090.0
Percentiles de REFA: 25%=0.0, 50%=0.0, 75%=0.0
Percentiles de CARTERA: 25%=0.0, 50%=0.0, 75%=7540.0
```

Nota: Comportamiento descriptivo de las variables financieras (numéricas) del dataset.

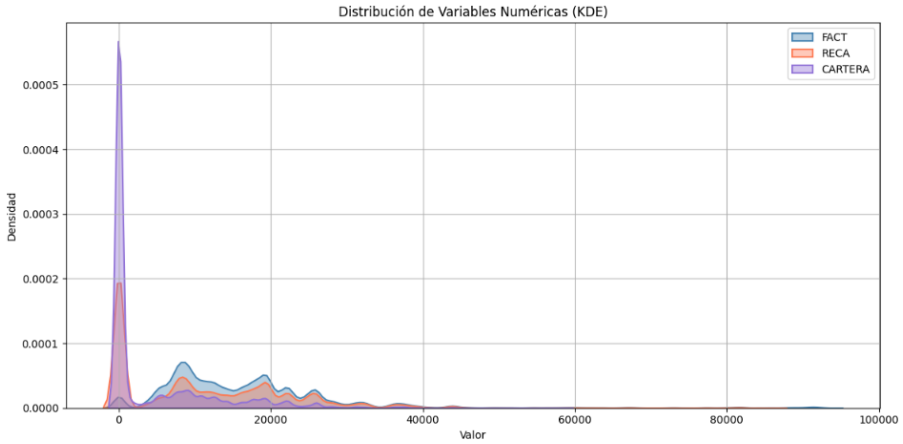
El análisis gráfico a través de curvas de densidad (KDE) de la ***Figura 16*** refuerza y representa con mayor claridad los hallazgos anteriores en cuanto a la distribución de las variables financieras. Para excluir distorsiones producto de valores atípicos, se utilizó un filtro eliminando los extremos dentro del percentil 1 y percentil 99.

Para la variable FACT, se encontró un clúster medio entre los $10.000 y los $25.000, con múltiples máximos que pueden corresponder a diferencias en valores tarifarios facturados. En el caso de la variable RECA, se halló una mayor aglomeración en valores cerca del cero, lo que apoya la interpretación del grupo de usuarios que no efectúa pagos mensuales, pero sí se encuentran rangos típicos para el pago dentro del $10.000 y el $20.000.

En relación con REFA, no se pudo graficar su distribución debido a la escasa variabilidad de los datos, la mayor parte de ellos se mantuvo en el nivel del cero. Por último, la variable CARTERA presentó una distribución intensamente sesgada a la izquierda y, la mayoría del universo poseen saldos pendientes bajos o nulos, y un reducido grupo tiene niveles relativos a la deuda acumulada altos.

Figura 16

Curva de Densidad de Variables Financieras



Nota: Distribución de las variables numéricas respecto a la densidad de sus datos.

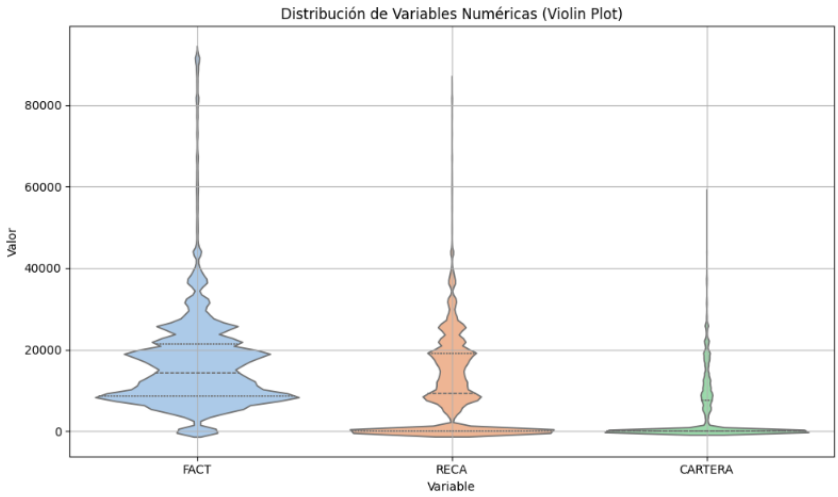
En el análisis visual a través de los gráficos de violín (***Figura 17***) el cual permite investigar en conjunto la distribución de densidad, la mediana y la dispersión de las variables numéricas destacadas donde su aplicación es particularmente destacable para la búsqueda de patrones en general, outliers y estructuras internas dentro del conjunto de datos.

La FACT, tiene una clara apariencia multimodal, con presencia de varianza de clientes respecto a su facturación, típicamente procedentes de diferentes estratos o clases de servicio. El máximo se encuentra en el límite $5.000-$25.000, con mediana en torno a los $12.000. El hecho de apreciar largas colas hacia el límite superior acentúa la presencia de unos clientes con valores superiores con poca densidad.

En el caso del recaudo mensual, hay una distribución sesgada a valores mínimos, con un incremento acentuado en densidad en el entorno del cero. Esto induce a que hay un gran número de clientes que no efectuó pagos durante el período que se analiza. Si bien se detectan registros con pagos altos, se trata de menos observaciones que en el caso de la facturación, que encaja con la hipótesis de morosidad en un segmento del conjunto. Por último, la variación CARTERA, que refleja una distribución altamente comprimida en torno del cero, lo que quiere decir que la mayoría de los clientes están actualizados o adeudan cuantías bajas. Sin embargo, la existencia de una cola acotada representa la presencia de clientes con cuantías pendientes altas, acumuladas quizás a lo largo del tiempo.

Figura 17

Distribución de las Variables Financieras (Grafico de Violín)

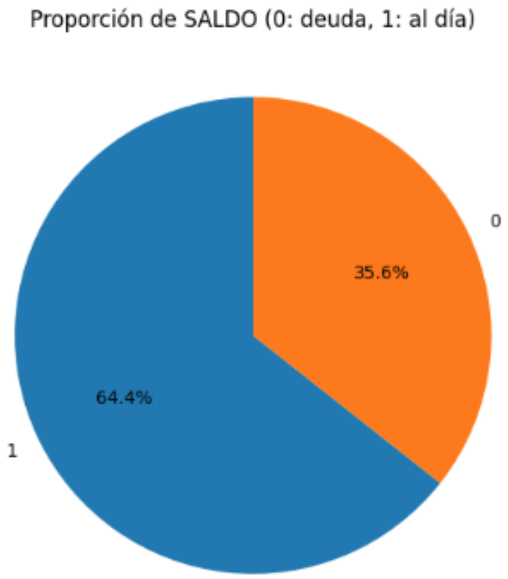


Nota: Representación conjunta de la dispersión, densidad y valores centrales de las variables numéricas.

La Variable SALDO, que mide el nivel mensual del cliente según el cumplimiento del pago, caracteriza una distribución dicotómica en la que el 64,4% son para clientes “al día” (valor 1), mientras el 35,6% está caracterizado por referencias a deuda (valor 0), como se aprecia en ***Figura 18***. Esa proporción refleja una mayoría de clientes que están actualizados en sus pagos, algo positivo en perspectiva financiera. Por el contrario, el resto del percentil, equivalentemente a tener a más de 630 mil registros, es un gran percentil de impagos recurrentes, algo que justifica la utilización del análisis predictor y permita predecir el riesgo de impago. Es una variable determinante en modelado supervisado, donde se tiene como target SALDO para analizar el comportamiento financiero del cliente según sus antecedentes históricos de recaudo.

Figura 18

Distribución de la Variable SALDO



Nota: Distribución binaria de la variable SALDO, utilizada para clasificar el estado mensual del usuario con respecto al cumplimiento de pago.

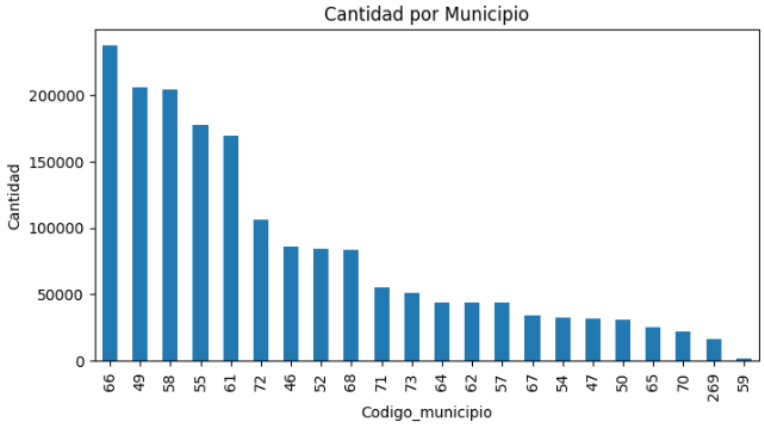
Análisis de Distribución Categórica

Al analizar la variable Código_municipio en la ***Figura 19***, se encontró que en los municipios con los códigos 66, 49 y 58 tienen muchos más registros que otros superando un umbral de 200.000 registros cada uno, la explicación de este segmento puede interpretarse a que una gran parte de los datos está concentrada en ciertas zonas.

Desde el municipio 72 en adelante, la cantidad de registros presentaba una baja considerable quedando por debajo de los 100.000 y algunos municipios incluso tienen menos de 20.000. Esto puede pasar por tener menos usuarios, menos cobertura o haber empezado a usar el sistema hace poco.

Figura 19

Cantidad de Registros por Municipio



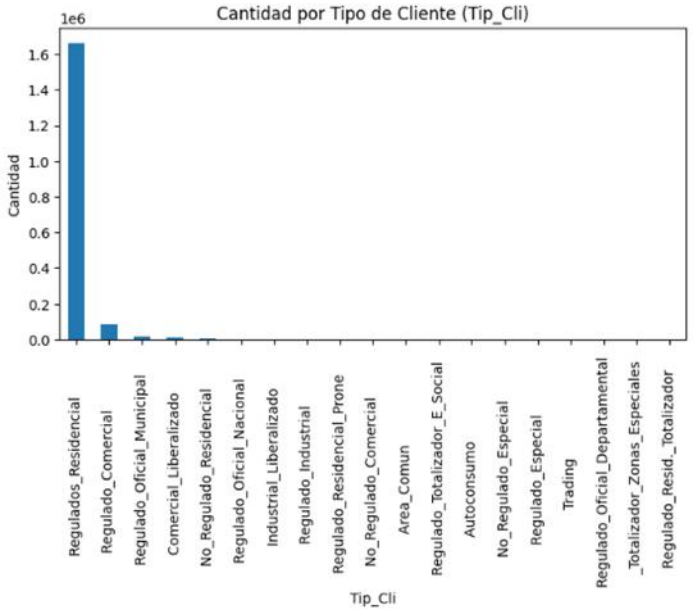
Nota: Frecuencia de registros por código de municipio.

Analizando la frecuencia de datos por Tip_Cli como se observa en la ***Figura 20***, la mayoría de los registros están en el grupo “Regulado_Residencial”, representando aproximadamente 90% del total de los registros.

Otro tipo de clientes como “Regulado_Comercial” y “Regulado_Oficial_Municipal” son menos frecuentes con cantidades de 100.000 y 20.000 registros, respectivamente. Por último, se detectaron categorías poco frecuentes, como “Trading”, “Totalizador_Zonas_Especiales” y “Regulado_Resid_Totalizador” con menos de 50 registros, estas pueden estar relacionadas con clientes especiales, actividades industriales o con tarifas particulares que no se aplican de forma regular.

Figura 20

Cantidad de Registros por Tip_Cli

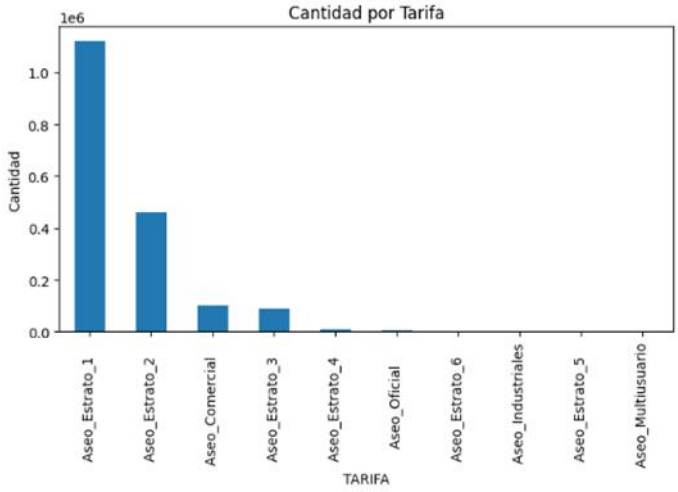


Nota: Frecuencia de registros por tipo cliente.

Al revisar la variable TARIFA con el grafico de barras mostrado en la Figura 21, se ve que la mayoría de los registros están en los estratos más bajos. La categoría Estrato 1 tiene más de 1.121.000 casos, y Estrato 2 cerca de 460.000. Otras tarifas, como Comercial y Estrato 3 también tienen cierta presencia, pero en menor cantidad con 100.000 y 84.000 registros respectivamente. Desde Estrato 4 en adelante los números bajan considerablemente donde categorías como Multiusuario o Estrato 5, apenas aparecen y no superan los 100 registros.

Figura 21

Cantidad de Registros por Tarifa



Nota: Frecuencia de registros por tipo cliente.

Análisis Predictivo

Después de entender cómo se comportan los datos en general por medio del análisis descriptivo, el siguiente paso es construir modelos que ayuden a predecir lo que podría pasar con los usuarios en el futuro. El enfoque de uno de esos análisis es el supervisado el cual busca responder a la pregunta de si un usuario estará al día con sus facturas o no, teniendo en cuenta los históricos de pagos de cada usuario.

Para eso se usó Random Forest, que es un modelo que funciona bastante bien cuando la variable objetivo es booleana, este modelo es confiable y es resistente a desbalances y sesgos en los datos. Además, también se aplica K-means, un modelo no supervisado comúnmente utilizado para agrupar usuarios según similitudes en su comportamiento, sin saber de antemano a qué grupo pertenecen.

Random Forest

Para detectar el comportamiento de pagos por usuario se crea una nueva tabla donde se observa el recaudo mes a mes (ver ***Figura 22***). Para ello se aplica un pivot de vigencia donde los valores sean del recudo, agrupándolos por las variables categóricas. Seguido de esta transformación se agregan las variables numéricas de FACT, REFA, RECA, CARTERA.

Figura 22

Dataframe de Comportamientos de Pagos por Usuario

```
[ ] var = ["Nic", "Codigo_municipio", "Codigo_corregimiento", "Codigo_departamento"]

    df_hist_reca = df_gconjunta.groupBy(var) \
        .pivot("Vigencia") \
        .agg(sum("RECA"))

    df_hist_reca = df_hist_reca.fillna(0.00)
    df_hist_reca.show()
```

```
+-------+----------------+--------------------+--------------------+------+------+------+------+------+------+------+------+------+------+------+------+
|    Nic|Codigo_municipio|Codigo_corregimiento|Codigo_departamento|202401|202402|202403|202404|202405|202406|202407|202408|202409|202410|202411|202412|
+-------+----------------+--------------------+--------------------+------+------+------+------+------+------+------+------+------+------+------+------+
|1064617|              66|                 859|                   2|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|
|1088725|              52|                 554|                   2|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|
|4401932|              55|                 633|                   2|11230.0|11240.0|12640.0|13040.0|14300.0|15640.0|15840.0|16630.0|17470.0|17450.0|17460.0|17490.0|
|4405353|              68|                 896|                   2|11680.0|11490.0|12380.0|12640.0|13740.0|14200.0|14500.0|13680.0|   0.0|14810.0|   0.0|14820.0|
|4405927|              55|                 626|                   2|   0.0|   0.0|   0.0| 9190.0|10130.0|10110.0|10280.0|10110.0|11190.0|10910.0|10910.0|   0.0|
|4409407|              70|                 921|                   2|12350.0|12510.0|12560.0|12560.0|12650.0|12690.0|12900.0|12200.0|13650.0|13710.0|13720.0|   0.0|
|4409583|              70|                 921|                   2|17300.0|18020.0|18100.0|18670.0|18210.0|18240.0|18440.0|17150.0|18550.0|18560.0|18580.0|   0.0|
|4415375|              61|                 774|                   2|19040.0|21490.0|19850.0|19820.0|19820.0|19820.0|21040.0|19080.0|19030.0|19280.0|19370.0|19370.0|
|4415707|              61|                 774|                   2|25060.0|27990.0|25810.0|25780.0|25970.0|25780.0|27330.0|24940.0|24640.0|24960.0|25170.0|25090.0|
|4417300|              61|                 774|                   2|19020.0|21600.0|19850.0|19820.0|20000.0|19820.0|21040.0|19090.0|19030.0|19270.0|19480.0|19370.0|
|4417848|              54|                 604|                   2|27390.0|30030.0|30030.0|30180.0|30160.0|30160.0|30200.0|28760.0|28160.0|28150.0|28160.0|28260.0|
|4420783|              64|                 818|                   2|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|
|4421109|              64|                 818|                   2| 7280.0| 7720.0| 7980.0| 7590.0| 7500.0| 7640.0| 7560.0| 7140.0| 7310.0| 7380.0| 7380.0| 7390.0|
|4455813|              55|                 633|                   2|14060.0|14050.0|15830.0|16360.0|17940.0|23440.0|23750.0|24860.0|26200.0|26190.0|26190.0|26190.0|
|4457609|              55|                 633|                   2|14170.0|14060.0|15830.0|16350.0| 3914.0|   0.0|   0.0|   0.0|26190.0|26496.0|   0.0|   0.0|
|4464727|              49|                 507|                   2|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|
|4465747|              49|                 510|                   2|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|   0.0|
|4470194|              49|                 495|                   2|20620.0|22320.0|22620.0|22340.0|22430.0|22470.0|22930.0|24600.0|25890.0|25960.0|25890.0|   0.0|
|4474567|              52|                 554|                   2|40420.0|72430.0|73040.0|72850.0|73940.0|73400.0|74680.0|77930.0|87890.0|87950.0|87900.0|   0.0|
|4476052|              52|                 554|                   2| 7280.0| 7780.0| 7800.0| 7790.0| 7860.0| 7860.0| 7950.0| 8550.0| 9180.0| 9180.0| 9190.0| 9180.0|
+-------+----------------+--------------------+--------------------+------+------+------+------+------+------+------+------+------+------+------+------+
only showing top 20 rows
```

Nota: Creación del dataframe df_hist_reca como base para observar el comportamiento de pagos por usuario.

Se agregaron columnas adicionales al dataframe df_hist_reca como Total_RECA, Total_FACT, Total_CARTERA (ver ***Figura 23***) de enero a noviembre 2024 para predecir la variable objetivo SALDO_202412 que determina si un usuario está al día con sus pagos a fecha de diciembre 202412 (variable booleana SALDO).

Figura 23

Ajustes del Dataframe Previos al Entrenamiento

```
from pyspark.sql.functions import col, sum, when

# 1. Crear copia sin alterar df_gconjunta
df_gc_temp = df_gconjunta.withColumn("Vigencia_int", col("Vigencia").cast("int"))

# 2. Filtrar entre 202401 y 202411
df_gc_filtrado = df_gc_temp.filter((col("Vigencia_int") >= 202401) & (col("Vigencia_int") <= 202411))

# 3. FACT_REAL = FACT + REFA
df_gc_filtrado = df_gc_filtrado.withColumn("FACT_REAL", col("FACT") + col("REFA"))

# 4. Agrupar por Nic y calcular Total_FACT y Total_RECA
totales = df_gc_filtrado.groupBy("Nic").agg(
    sum("FACT_REAL").alias("Total_FACT"),
    sum("RECA").alias("Total_RECA")
)

# 5. Eliminar columna 202412
columnas_validas = [c for c in df_hist_reca.columns if c not in ("202412")]
df_hist_reca_filtrado = df_hist_reca.select(*columnas_validas)

# 6. Unir con los totales nuevos
df_train_rf = df_hist_reca_filtrado.join(totales, on="Nic", how="left")

# 7. Calcular Total_CARTERA y generar variable SALDO_202412
df_train_rf = df_train_rf.withColumn("Total_CARTERA", col("Total_FACT") - col("Total_RECA"))

df_train_rf = df_train_rf.withColumn(
    "SALDO_202412",
    when(col("Total_CARTERA") <= 100, 1).otherwise(0)
)
```

Nota: Transformaciones adicionales para predecir el estado de cuenta de cada usuario.

Adicionalmente se detectaron campos nulos en algunas variables y Nics (identificativo usuario) duplicados que se descartaron para no generar ruido en el entrenamiento (ver ***Figura 24***).

Figura 24

Limpieza de df_train_rf

```
from pyspark.sql.functions import col, sum as spark_sum

df_train_rf.select([
    spark_sum(col(c).isNull().cast("int")).alias(c)
    for c in df_train_rf.columns
]).show()
```

```
+---+----------------+------------------+-------------------+------+------+------+------+------+------+------+------+------+------+------+----------+----------+-------------+------------+
|Nic|Codigo_municipio|Codigo_corregimiento|Codigo_departamento|202401|202402|202403|202404|202405|202406|202407|202408|202409|202410|202411|Total_FACT|Total_RECA|Total_CARTERA|SALDO_202412|
+---+----------------+------------------+-------------------+------+------+------+------+------+------+------+------+------+------+------+----------+----------+-------------+------------+
|  0|               0|                 0|                  0|     0|     0|     0|     0|     0|     0|     0|     0|     0|     0|     0|       638|       638|          638|           0|
+---+----------------+------------------+-------------------+------+------+------+------+------+------+------+------+------+------+------+----------+----------+-------------+------------+
```

```
# Eliminar filas con valores nulos
df_train_rf = df_train_rf.dropna()
```

```
from pyspark.sql.functions import count

df_train_rf.groupBy("Nic") \
    .agg(count("*").alias("conteo")) \
    .filter(col("conteo") > 1) \
    .show()
```

```
+-------+------+
|    Nic|conteo|
+-------+------+
|7037723|     2|
|6289118|     2|
|7917081|     2|
|4436788|     2|
+-------+------+
```

```
# 2. Eliminar duplicados por Nic
df_train_rf = df_train_rf.dropDuplicates(["Nic"])
```

Nota: Limpieza del dataframe de valores nulos y duplicados.

Para generar un resultado más robusto en el modelo de clasificación se agregan nuevas variables estadísticas para que dicho modelo pueda interpretar de mejor forma el comportamiento de los datos por usuario (ver ***Figura 25***). Dichas variables se describen de la siguiente forma:

- **RECA_media:** Promedio mensual de recaudo por usuario de enero a noviembre 2024.
- **RECA_desv:** Desviación estándar de los pagos mensuales, identificando la variación de los pagos de cada usuario.
- **RECA_tendencia:** Diferencia entre el pago de enero y noviembre 2024 para interpretar el avance o declive de los pagos de inicio a fin.
- **RECA_cumplimiento:** Es una proporción de meses en los que el usuario pagó más de $5000 para determinar una entrada de dinero fija.
- **RECA_ultimos3:** Promedio de los pagos de los últimos tres meses (202409, 202410, 202411) para capturar el comportamiento más reciente.
- **RECA_ratio_FACT:** Es una proporción entre el total recaudado (Total_RECA) y el total facturado (Total_FACT) para detectar si el usuario tiende a pagar completamente su factura.

Figura 25

Feature Engineering al Dataframe

```
[ ] from pyspark.sql.functions import col, expr, when, lit
    from functools import reduce

    # 1. Columnas RECA de enero a noviembre
    reca_cols = [f"2024{str(m).zfill(2)}" for m in range(1, 12)]  # 202401 a 202411

    # 2. Crear una copia del DataFrame base para no sobrescribir
    df_ext = df_train_rf

    # 3. Promedio mensual de pagos
    reca_sum = reduce(lambda a, b: a + b, [col(c) for c in reca_cols])
    df_ext = df_ext.withColumn("RECA_media", reca_sum / len(reca_cols))

    # 4. Desviación estándar (manual)
    expr_suma = " + ".join([f"POWER({c} - RECA_media, 2)" for c in reca_cols])
    expr_desv = f"SQRT( ({expr_suma}) / {len(reca_cols)} )"
    df_ext = df_ext.withColumn("RECA_desv", expr(expr_desv))

    # 5. Tendencia: último - primero
    df_ext = df_ext.withColumn("RECA_tendencia", col("202411") - col("202401"))

    # 6. Cumplimiento mensual (>5000)
    expr_cumplimiento = " + ".join([f"(CASE WHEN {c} > 5000 THEN 1 ELSE 0 END)" for c in reca_cols])
    df_ext = df_ext.withColumn("RECA_cumplimiento", expr(f"({expr_cumplimiento}) / {len(reca_cols)}"))

    # 7. Promedio últimos 3 meses
    df_ext = df_ext.withColumn("RECA_ultimos3", (col("202409") + col("202410") + col("202411")) / 3)

    # 8. Proporción entre Total_RECA y Total_FACT
    df_ext = df_ext.withColumn("RECA_ratio_FACT", col("Total_RECA") / (col("Total_FACT") + lit(1e-6)))

    # 9. Variables nuevas
    vars_nuevas = [
        "RECA_media", "RECA_desv", "RECA_tendencia",
        "RECA_cumplimiento", "RECA_ultimos3", "RECA_ratio_FACT"
    ]

    # 10. Crear df_modelo_ext con columnas originales + nuevas
    input_cols_ext = ["Nic"] + reca_cols + [
        "Codigo_municipio", "Codigo_corregimiento", "Codigo_departamento"
    ] + vars_nuevas + ["SALDO_202412"]

    df_modelo_ext = df_ext.select(*input_cols_ext)

    # 11. Validar estructura y primeras filas
    df_modelo_ext.printSchema()
    df_modelo_ext.select(vars_nuevas + ["SALDO_202412"]).show(5)
```

```
root
 |-- Nic: long (nullable = true)
 |-- 202401: double (nullable = false)
 |-- 202402: double (nullable = false)
 |-- 202403: double (nullable = false)
 |-- 202404: double (nullable = false)
 |-- 202405: double (nullable = false)
 |-- 202406: double (nullable = false)
 |-- 202407: double (nullable = false)
 |-- 202408: double (nullable = false)
 |-- 202409: double (nullable = false)
 |-- 202410: double (nullable = false)
 |-- 202411: double (nullable = false)
 |-- Codigo_municipio: long (nullable = true)
 |-- Codigo_corregimiento: long (nullable = true)
 |-- Codigo_departamento: long (nullable = true)
 |-- RECA_media: double (nullable = true)
 |-- RECA_desv: double (nullable = true)
 |-- RECA_tendencia: double (nullable = false)
 |-- RECA_cumplimiento: double (nullable = true)
 |-- RECA_ultimos3: double (nullable = true)
 |-- RECA_ratio_FACT: double (nullable = true)
 |-- SALDO_202412: integer (nullable = false)

+------------------+------------------+--------------+-----------------+------------------+------------------+------------+
|        RECA_media|         RECA_desv|RECA_tendencia|RECA_cumplimiento|     RECA_ultimos3|   RECA_ratio_FACT|SALDO_202412|
+------------------+------------------+--------------+-----------------+------------------+------------------+------------+
| 5697.272727272727|196708.72729814556|        9480.0|              1.0|            9360.0|0.9999999999840434|           1|
| 6349.090909090909| 196056.9091164119|        1130.0|              1.0|            6860.0|0.9999999999856817|           1|
|22249.090909090908|180156.90911866268|       37330.0|              1.0|37353.333333333336| 0.999999999995914|           1|
|               0.0|202406.00002470284|           0.0|              1.0|               0.0|               0.0|           1|
|           25690.0|  176716.000028294|         390.0|              1.0|24896.666666666668|0.9999999999964613|           1|
+------------------+------------------+--------------+-----------------+------------------+------------------+------------+
only showing top 5 rows
```

Nota: Creación de variables estadísticas para ayudar a la precisión del modelo.

Con el propósito de predecir el cumplimiento de pago para los usuarios en el mes de diciembre 2024 por medio de la variable objetivo SALDO_202412 se implementó el modelo de clasificación Random Forest mostrado en la ***Figura 26***. Este modelo se destaca por su clasificación binaria cuya técnica de aprendizaje supervisado es robusta para datos estructurados con resistencia a sobreajuste y sesgos por los datos (Dangeti, 2017).

Las variables mostradas en la ***Figura 25*** son integradas por medio de VectorAssembler de Pyspark el cual junta todas las variables numéricas en un solo vector y usando un pipeline se encapsula el modelo con dicho ensamblador. El modelo fue configurado con una cantidad de 50 árboles (numTrees = 50) y profundidad de 10 (maxDepth = 10) y para evitar sobreajuste se realizó validación cruzada con 3 particiones (numFolds = 3) sobre un conjunto de entrenamiento del 80% del dataframe propuesto. Por último, el modelo fue evaluado bajo curva ROC (areaUnderROC) midiendo el rendimiento del modelo para discriminar entre usuarios al día y deudores o morosos.

Luego de evaluar el modelo con el conjunto de prueba, se revisaron las probabilidades de predicción para los usuarios que estaban al día, es decir, la clase positiva. Con esa información se construyó una curva de precisión frente a exhaustividad (conocida como Precision–Recall), que es especialmente útil cuando las clases están desbalanceadas (ver ***Figura 27***).

A partir de esta curva se identificó el mejor punto de corte, es decir, el umbral de probabilidad que permite obtener el mayor F1-score. Esta métrica define qué tan precisa es la clasificación de casos positivos y cuántos de esos positivos reales se lograron captar lo cual permite ajustar el modelo de forma más fina, en lugar de quedarse con el típico umbral de 0.5.

Figura 26

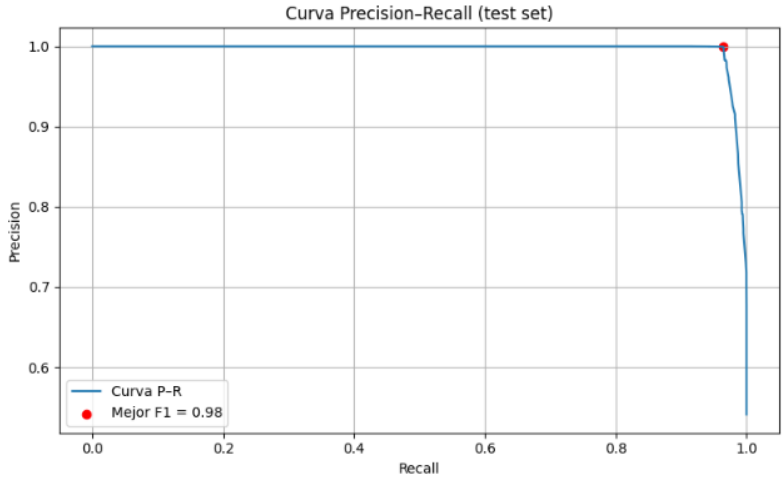
Entrenamiento del Modelo de Clasificación Random Forest

```
[ ] from pyspark.ml.classification import RandomForestClassifier
    from pyspark.ml.feature import VectorAssembler
    from pyspark.ml import Pipeline
    from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
    from pyspark.ml.evaluation import BinaryClassificationEvaluator
    from pyspark.sql.functions import col
    from sklearn.metrics import precision_recall_curve, classification_report, confusion_matrix, roc_curve, roc_auc_score
    import matplotlib.pyplot as plt
    import seaborn as sns
    import numpy as np

    # 0. Dividir el dataset (80% train, 20% test)
    train_df, test_df = df_modelo_ext.randomSplit([0.8, 0.2], seed=42)

    # 1. Columnas seleccionadas (meses + derivadas + geográficas)
    features_cols_ext = [
        "202401", "202402", "202403", "202404", "202405", "202406",
        "202407", "202408", "202409", "202410", "202411",
        "Codigo_municipio", "Codigo_corregimiento", "Codigo_departamento",
        "RECA_media", "RECA_desv", "RECA_tendencia",
        "RECA_cumplimiento", "RECA_ultimos3", "RECA_ratio_FACT"
    ]

    # 2. VectorAssembler
    assembler_ext = VectorAssembler(inputCols=features_cols_ext, outputCol="features")

    # 3. Modelo Random Forest
    rf = RandomForestClassifier(labelCol="SALDO_202412", featuresCol="features", seed=42)

    # 4. Pipeline
    pipeline = Pipeline(stages=[assembler_ext, rf])

    # 5. Hiperparámetros
    paramGrid = ParamGridBuilder() \
        .addGrid(rf.numTrees, [50]) \
        .addGrid(rf.maxDepth, [10]) \
        .build()

    # 6. CrossValidator (solo sobre train_df)
    evaluator = BinaryClassificationEvaluator(labelCol="SALDO_202412", metricName="areaUnderROC")
    cv = CrossValidator(estimator=pipeline,
                        estimatorParamMaps=paramGrid,
                        evaluator=evaluator,
                        numFolds=3,
                        seed=42)

    # 7. Entrenamiento con validación cruzada
    cv_model = cv.fit(train_df)

    # 8. Evaluación sobre test_df
    predictions = cv_model.transform(test_df)
    pdf_preds = predictions.select("SALDO_202412", "probability").toPandas()
    pdf_preds["prob_1"] = pdf_preds["probability"].apply(lambda x: float(x[1]))
    pdf_preds.rename(columns={"SALDO_202412": "SALDO"}, inplace=True)

    # 9. Umbral óptimo
    precision, recall, thresholds = precision_recall_curve(pdf_preds["SALDO"], pdf_preds["prob_1"])
    f1_scores = 2 * (precision * recall) / (precision + recall + 1e-6)
    best_idx = np.argmax(f1_scores)
    best_thr = thresholds[best_idx]
    best_f1 = f1_scores[best_idx]
    print(f"Umbral óptimo según F1 = {best_f1:.3f} → Threshold = {best_thr:.3f}")

    # 10. Clasificación ajustada
    pdf_preds["y_pred"] = (pdf_preds["prob_1"] >= best_thr).astype(int)
```

Nota: Entrenamiento del modelo Random Forest a partir del dataframe df_modelo_ext.

Figura 27

Curva Precision-Recall

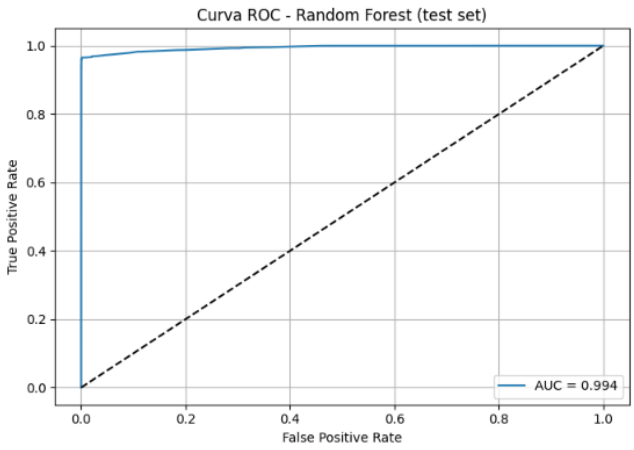


Nota: Curva de Precision-Recall para identificar el mejor umbral para el modelo.

La curva ROC mostrada en la ***Figura 28*** muestra un AUC cercano a 1, a lo que indica que el modelo tiene muy buena capacidad de diferenciar entre las dos clases, es decir, logra distinguir entre usuarios al día y morosos.

Figura 28

Curva ROC



Nota: Eficiencia del modelo para identificar clases.

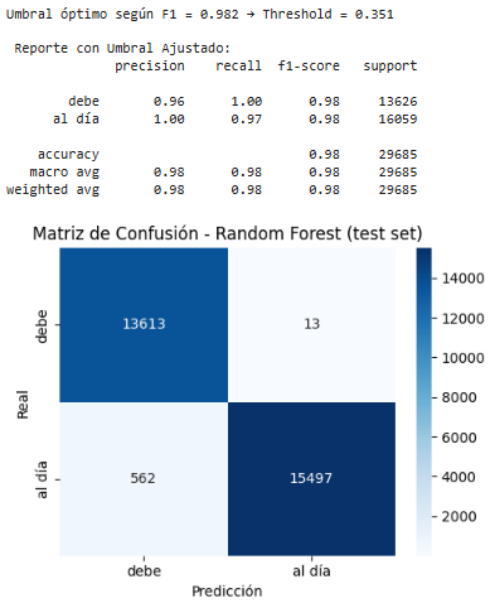
Con el modelo ya entrenado y usando un umbral de 0.351 (mejor F1-score: 0.982), la matriz de confusión (ver ***Figura 29***) mostró que se clasificaron correctamente 13.613 usuarios con deuda y 15.497 al día con una precisión general fue del 98%, concluyendo que el modelo distingue bien entre usuarios que pagan y los que no.

El F1-score obtenido en ambas clases fue de 0.98, lo cual refleja un buen balance entre precisión y recall. En la clase morosa el recall llegó al 100% por lo que el modelo logró detectar a los usuarios con deuda. En el grupo “al día”, la precisión fue también del 100%, aunque el recall bajó un poco con un resultado del 97%.

Por último, al revisar los promedios macro y ponderado de precisión, recall y F1-score, todos quedaron en 0.98 demostrando que el modelo funciona bien incluso si las clases no están balanceadas.

Figura 29

Reporte de Clasificación y Matriz de Confusión de los de datos de prueba.

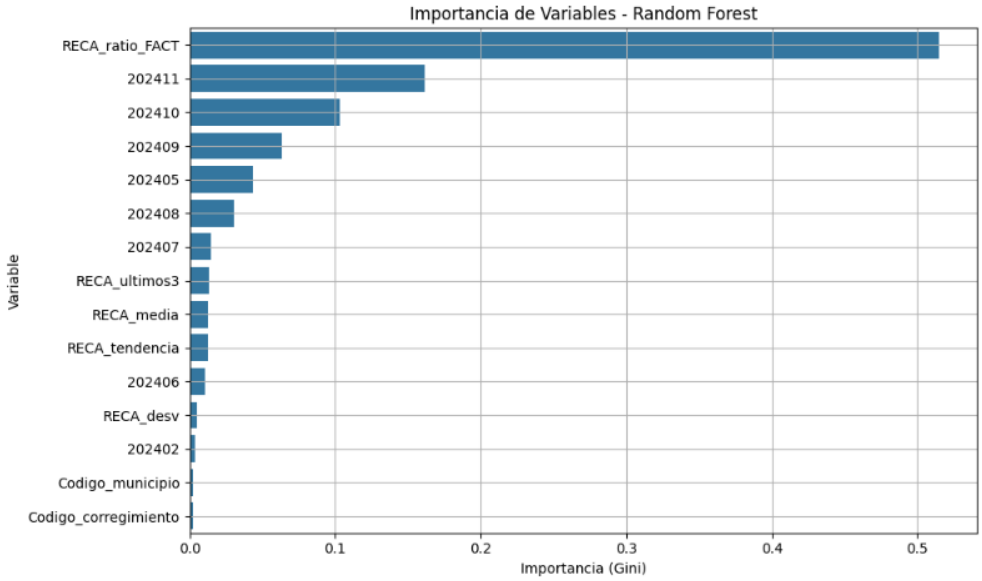


Nota: Resultados obtenidos del modelo de clasificación Random Forest con métricas de clasificación.

Para evaluar la posibilidad de sobreajuste del modelo, se evaluó la importancia de las variables predictoras sobre la variable objetivo. La variable RECA_ratio_FACT se destaca por superar en más de un 50% en la decisión del modelo por lo que es una variable altamente influyente en el resultado (ver ***Figura 30***). Como segunda instancia se aplicó el modelo en los conjuntos de entrenamiento y prueba evaluando similitudes como se muestra en la ***Figura 31***, los resultados muestran una alta similitud por lo que se descarta sobreajuste por su buena capacidad de generalización al no haber memorizado solo los datos de entrenamiento y al aprender patrones que se mantienen en los datos nuevos.

Figura 30

Importancia de las Variables



Nota: Nivel de importancia de las variables predictoras sobre la variable objetivo en el modelo de clasificación.

Figura 31

Comparativo de los Datos de Entrenamiento y Prueba

```
[ ] from sklearn.metrics import f1_score, accuracy_score, roc_auc_score
    from pyspark.sql.functions import col
    import pandas as pd

    def evaluar_rendimiento(model, df_spark, umbral):
        # Generar predicciones
        pred_df = model.transform(df_spark).select("SALDO_202412", "probability")
        pdf = pred_df.toPandas()
        pdf["prob_1"] = pdf["probability"].apply(lambda x: float(x[1]))
        pdf["y_true"] = pdf["SALDO_202412"]
        pdf["y_pred"] = (pdf["prob_1"] >= umbral).astype(int)

        # Calcular métricas
        acc = accuracy_score(pdf["y_true"], pdf["y_pred"])
        f1 = f1_score(pdf["y_true"], pdf["y_pred"])
        auc = roc_auc_score(pdf["y_true"], pdf["prob_1"])
        return acc, f1, auc

    # Umbral óptimo ya obtenido anteriormente
    umbral_optimo = best_thr

    # Evaluar en train y test
    acc_train, f1_train, auc_train = evaluar_rendimiento(cv_model, train_df, umbral_optimo)
    acc_test, f1_test, auc_test = evaluar_rendimiento(cv_model, test_df, umbral_optimo)

    # Consolidar en DataFrame
    df_eval = pd.DataFrame({
        "Dataset": ["Entrenamiento", "Test"],
        "Accuracy": [acc_train, acc_test],
        "F1 Score": [f1_train, f1_test],
        "AUC": [auc_train, auc_test]
    })

    print(df_eval)

         Dataset  Accuracy  F1 Score       AUC
0  Entrenamiento  0.981357  0.982499  0.994440
1           Test  0.980630  0.981786  0.994214
```

Nota: Comparativo de resultados obtenido entre datos de prueba y entrenamiento.

Posteriormente al entrenamiento, se aplicó el modelo hacia nuevos datos para predecir si un usuario está al día o presenta deudas para enero de 2025 a partir de un histórico de pagos y facturación de todo el año 2024, los resultados arrojados se observan en la **Figura 32**.

Figura 32

Predicciones del Modelo para enero 2025

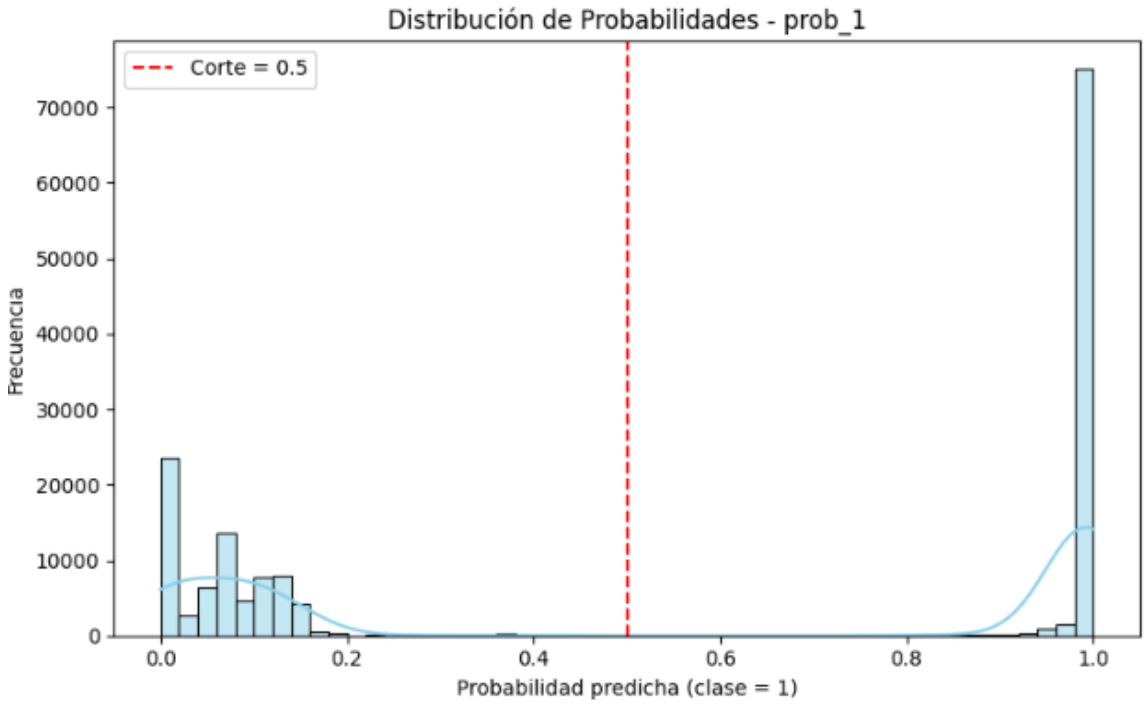
	pred_ajustada	confianza	cantidad
2	1	alta	78572
3	1	baja	298
4	1	media	41
0	0	alta	71442
1	0	baja	13

Nota: Predicciones obtenidas para nuevos datos.

Partiendo de los del histograma observado en la **Figura 33**, se muestra una distribución binomial con altas concentraciones cercas de 0 y 1. Esto demuestra que existe separación entre ambas clases y que hay poca incertidumbre en la predicción del modelo, demostrando que el modelo está bien entrenado y que no está generando datos aleatorios.

Figura 33

Distribución de Resultados en Datos Nuevos

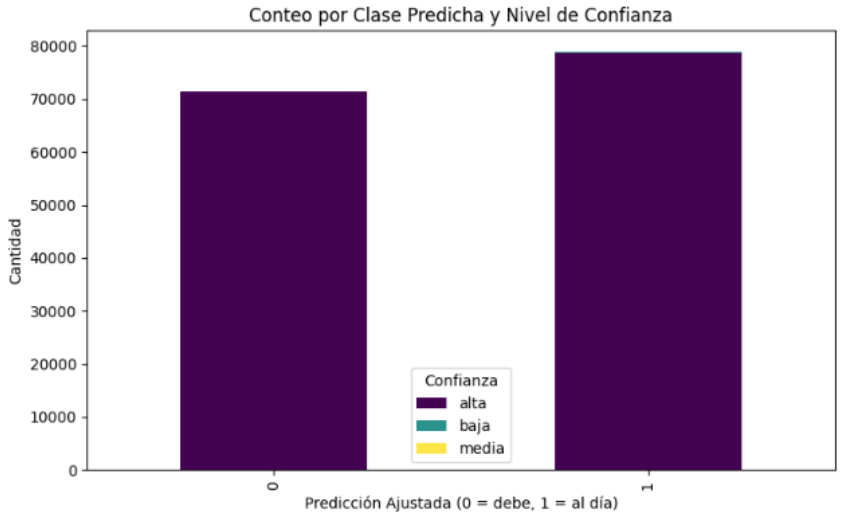


Nota: Distribución de la clase 1 (al día) en datos nuevo.

Los resultados mostrados en la ***Figura 34*** evidencian que además de poder predecir entre clases sin sesgos, el modelo tiene alta confiabilidad en sus resultados con buen balance.

Figura 34

Nivel de Confianza del Modelo con Datos Nuevos



Nota: Nivel de confianza entre clases para los nuevos datos.

K–means

Para aplicar el modelo no supervisado de clusterización K-means en primera instancia se aplicó limpieza de los datos eliminando registros vacíos, se indexaron variables categóricas como lo son Tip_Cli y TARIFA para posteriormente ser implementados en un ensamblador por medio de VectorAssembler y así aplicar un escalado de las variables usando StandarScaler como se observa en la ***Figura 35***. Finalmente, el modelo pasa por un evaluador cuya métrica es "Silhouette" para evidenciar que tan bien formados están los clusters creados por el modelo dependiendo del número de agrupamiento asignado.

Figura 35

Preparación de los Datos para Modelo de Agrupamiento

```
[ ] from pyspark.sql.functions import col
    from pyspark.ml.feature import StringIndexer, VectorAssembler, StandardScaler
    from pyspark.ml.clustering import KMeans
    from pyspark.ml.evaluation import ClusteringEvaluator
    import matplotlib.pyplot as plt

    # DATASET
    df_sample = df_gconjunta

    # Cast y limpieza
    df_sample = df_sample.withColumn("CARTERA", col("CARTERA").cast("double"))
    df_sample = df_sample.dropna(subset=["FACT", "RECA", "REFA", "CARTERA", "Tip_Cli", "TARIFA"])

    # Indexar
    index_tipcli = StringIndexer(inputCol="Tip_Cli", outputCol="Tip_Cli_indexed", handleInvalid="keep").fit(df_sample)
    index_tarifa = StringIndexer(inputCol="TARIFA", outputCol="TARIFA_indexed", handleInvalid="keep").fit(df_sample)

    df_indexed = index_tipcli.transform(df_sample)
    df_indexed = index_tarifa.transform(df_indexed)

    # Assembler
    assembler = VectorAssembler(
        inputCols=["FACT", "RECA", "REFA", "CARTERA", "Tip_Cli_indexed", "TARIFA_indexed"],
        outputCol="features_unscaled"
    )
    df_assembled = assembler.transform(df_indexed)

    # Escalar
    scaler = StandardScaler(inputCol="features_unscaled", outputCol="features", withMean=True, withStd=True)
    scaler_model = scaler.fit(df_assembled)
    df_scaled = scaler_model.transform(df_assembled)
```

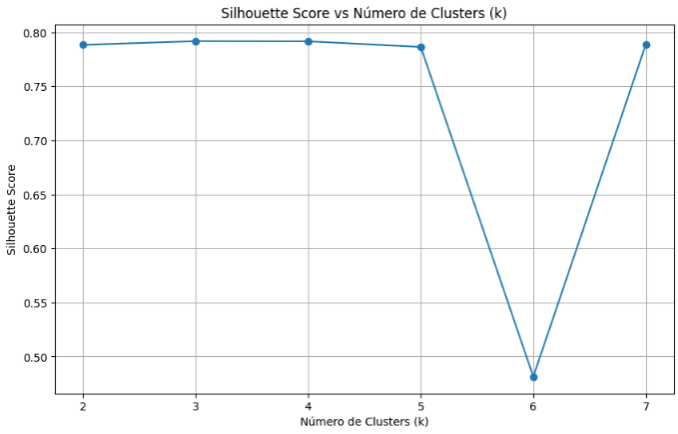
```
[ ] evaluator = ClusteringEvaluator(featuresCol="features", predictionCol="prediction", metricName="silhouette", distanceMeasure="squaredEuclidean")
```

Nota: Transformación y parametrización de variables para modelo no supervisado.

Se aplicaron agrupamientos con valores de k del 2 al 7 con el fin de evidenciar gráficamente cuantos clusters son los adecuados para el modelo de agrupamiento (ver **Figura 36**). Los resultados obtenidos muestran una muy leve preferencia hacia 3 clusters.

Figura 36

Métrica de Evaluación Silhouette para K-means



Nota: Evaluación de clusters para el modelo de agrupamiento.

Conociendo el número de clusters optimo se entrena el modelo no supervisado de K-means y posteriormente se muestran los resultados por medio de un gráfico de dispersión como se observa en el código de la ***Figura 37*** y ***Figura 38***.

Figura 37

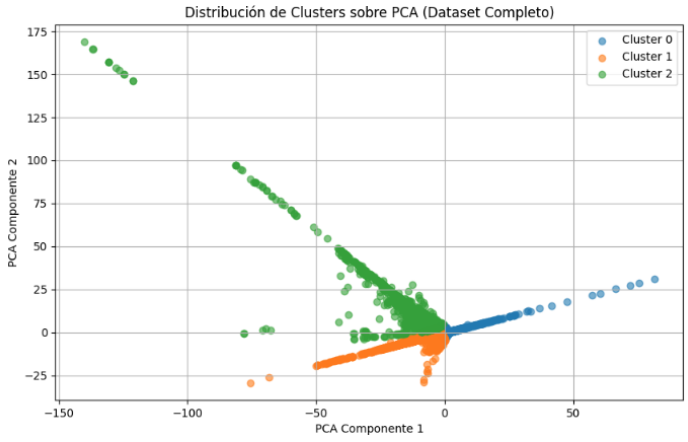
Aplicación del Modelo K-means con k Optimo

```
# 4. Aplicar PCA
from pyspark.ml.feature import PCA

pca = PCA(k=2, inputCol="features", outputCol="pca_features")
pca_model = pca.fit(df_scaled)
df_pca = pca_model.transform(df_scaled)

# 5. Entrenar KMeans final con k óptimo
kmeans_final = KMeans(featuresCol="pca_features", predictionCol="cluster", k=best_k, seed=42)
modelo_final = kmeans_final.fit(df_pca)
resultados_final = modelo_final.transform(df_pca)

print("Modelo final entrenado sobre todo el dataset")

# 6. Visualizar Clusters
import pandas as pd

pdf_resultados = resultados_final.select("pca_features", "cluster").toPandas()

pdf_resultados["PCA1"] = pdf_resultados["pca_features"].apply(lambda x: x[0])
pdf_resultados["PCA2"] = pdf_resultados["pca_features"].apply(lambda x: x[1])

plt.figure(figsize=(10, 6))
for cluster in sorted(pdf_resultados["cluster"].unique()):
    subset = pdf_resultados[pdf_resultados["cluster"] == cluster]
    plt.scatter(subset["PCA1"], subset["PCA2"], label=f"Cluster {cluster}", alpha=0.6)

plt.title("Distribución de Clusters sobre PCA (Dataset Completo)")
plt.xlabel("PCA Componente 1")
plt.ylabel("PCA Componente 2")
plt.legend()
plt.grid(True)
plt.show()
```

Nota: Entrenamiento del modelo con numero de clusters optimo.

Figura 38

Scatterplot del Modelo K-means



Nota: Representación gráfica del modelo K-means.

En la ***Figura 39*** se puede observar el comportamiento de cada cluster según sus características, el cluster 0 presenta la mayor cantidad de registros con el promedio más bajo en facturación y promedios cartera moderada, este comportamiento es razonable debido a que la clasificación predominante de este grupo es los estratos 1.

El cluster 1 donde predominan los usuarios estrato 3, manifiesta una cantidad de registros moderada respecto a las demás con un promedio de facturación superior al cluster 0 pero con un promedio de cartera menor. Este comportamiento evidencia que a pesar de tener mayor facturación que el cluster 0, los pagos realizados por este grupo son condescendientes a su facturación en su mayoría. En la distribución de la variable SALDO mostrada en la ***Figura 40***, denota que este cluster tiene un porcentaje del 92% de saldos al día, es el grupo con mayor predominancia hacia la categoría 1.

Por último, en el cluster 2, la cantidad de registros es mucho menor a los demás grupos, pero con una alta facturación y un promedio de cartera alto. Estos resultados están relacionados al tipo de tarifa dominante (comercial) donde los cobros para este tipo de usuarios son mucho mayores a los residenciales, sin embargo, este grupo ofrece una muy baja proporción de usuarios al día por lo que sería un sector crítico por su baja tasa de pagos.

Figura 39

Comportamiento de cada cluster

cluster	Cantidad_Facturas	Promedio_FACT	Promedio_CARTERA	Tarifa_Dominante	
1	146711	45587.682927660506	1540.2428720409512	Aseo_Estrato_3	...
0	1620067	14356.46910899364	4695.9219655730285	Aseo_Estrato_1	...
2	18855	106743.47653142402	103601.65160434898	Aseo_Comercial	...

Nota: Características de cada cluster según sus variables.

Figura 40

Distribución de la Variable SALDO en cada cluster

cluster	SALDO	Cantidad
0	0	606228
0	1	1013839
1	0	11214
1	1	135497
2	0	18670
2	1	185

Nota: Comportamiento del estado de cuenta de los usuarios.

Conclusiones

Utilizando Python como ecosistema principal con soporte de frameworks como Spark y con ayuda de computación en la nube como Google Colab, se logró optimizar el tratamiento de datos masivos para el caso de la empresa de aseo domiciliario. Se aplicaron técnicas de Big Data para el procesamiento de los datos, aplicando concatenaciones de archivos por medio de conexiones a SharePoint, implementando ETL para la limpieza de los datos y EDA donde los análisis descriptivo y predictivo prevalecen en este apartado.

Se liberó carga de trabajo al utilizar computación en la nube, eliminando la dependencia de hardware requerido para la trata y procesamiento de grandes volúmenes de datos, los cuales son computacionalmente costoso sobre todo cuando se aplican técnicas de análisis predictivo para entrenar modelos. La mejora de los procesos mejoro sustancialmente en consideración de las herramientas y técnicas utilizadas actualmente por la empresa que no alcanzan a cubrir sus necesidades evidenciando una clara mejora en la eficiencia de capacidad de análisis.

En el análisis exploratorio se descubrieron diferentes patrones de los usuarios respecto al comportamiento de su facturación, pagos y cartera donde las densidades en cartera son altas en valores cercanos a 0 pero en recaudo (pago de usuarios) con el mismo comportamiento. Se encontró alta variabilidad en valores facturados consecuencia de los diferentes tipos de tarifas presentes en los datos, pero con una proporción positiva de usuarios al día con sus facturas (64%).

Se identificó alta concentración de usuarios en ciertos municipios de la región producto de alta densidad poblacional respecto a otras zonas donde la gran mayoría de usuarios pertenecen a predios residenciales destacándose los estratos más bajos (1 y 2) en comparación con las demás categorías.

En los análisis predictivos se aplicó un modelo supervisado (Random Forest) y no supervisado (K-means) para detectar el comportamiento de usuarios con saldos al día y su agrupación bajo las características de sus variables. Los resultados del Random Forest muestran que tiene una alta capacidad en detectar las clases (al día, morosos) sin dejase influenciar de sesgos en los datos, obteniendo un f1-score del 98% y un AUC muy cercano a 1. Se descartó la posibilidad sobreajuste al comparar resultados entre los conjuntos de entrenamiento y prueba que resultaron ser muy similares, y que, al aplicar el modelo con nuevos datos, la distribución de probabilidades se encuentra cercanas a 0 y 1 lo que evidencia bidimensionalidad sin resultados aleatorios, además de una alta confianza en las predicciones.

En el modelo de clusterización K-means se detectó que el número de agrupamiento ideal para el conjunto de datos es 3 por medio de la métrica Silhouette, dentro de los grupos se logra destacar cluster 1 el cual tiene una alta tasa de facturas al día con valores moderados en su facturación y dominancia en tarifas estrato 3.

A partir de estos hallazgos, se propone la implementación de un sistema de alerta temprana que, utilizando el modelo Random Forest entrenado, permita identificar mensualmente a los usuarios con alto riesgo de morosidad y activar estrategias diferenciadas de cobranza preventiva. También se recomienda institucionalizar el uso de reportes automáticos en Power BI, integrando las predicciones y agrupamientos para facilitar la toma de decisiones. Finalmente, se sugiere escalar el modelo a producción mediante servicios en la nube y bases de datos actualizadas periódicamente, estableciendo un ciclo de monitoreo predictivo continuo para la gestión de cartera.

Recomendaciones

Si bien es cierto que se obtuvieron resultados positivos dentro de los modelos predictivos, es necesario implementar datos recientes y antiguos para detectar el comportamiento de los usuarios a lo largo de los años. Incluir estos nuevos datos permite interpretar mejor el comportamiento de los usuarios al tener un histórico más prolongado en el tiempo respecto a las variables financieras. Esto puede dar mayor robustez al modelo supervisado y abre puertas a nuevos comportamientos como los de usuarios responsables con sus pagos y que en algún momento dejaron de hacerlo. Tener en cuenta que, al aumentar el volumen de los datos, lo más probable es que se requieran de servicios extras en la nube que requieran de un coste adicional.

Respecto a los resultados obtenidos en el modelo de agrupamiento K-means, el cluster 2 presenta una tasa demasiado alta de morosos en 2024 (99%), por lo que resulta conveniente hacer seguimiento a este grupo para detectar las razones de impago.

En el modelo supervisado Random Forest es imprescindible comparar los resultados de las predicciones con los datos reales, si bien es cierto que el modelo obtuvo buenos resultados, no es un ultimátum para determinar si un usuario paga o no ya que pueden existir otras variables que influencien este comportamiento que no se tienen en cuenta por su ausencia como lo puede ser variables sociodemográficas o estados financieros.

Adicionalmente, se recomienda establecer un plan de implementación progresiva para el despliegue del modelo predictivo en ambientes operativos reales. Este plan debe contemplar etapas como: (1) preparación tecnológica y migración de datos históricos, (2) entrenamiento y ajuste del modelo en ciclos mensuales, (3) integración con sistemas de reporte como Power BI para visualización de resultados, y (4) evaluación de impacto mediante indicadores clave como reducción de cartera vencida, precisión del modelo y mejora en la eficiencia operativa. También

será fundamental considerar los recursos tecnológicos necesarios, como instancias en la nube o conectores de integración, así como los recursos humanos requeridos para su mantenimiento y supervisión.

Referencias Bibliográficas

Amalina, F., Targio Hashem, I. A., Azizul, Z. H., Fong, A. T., Firdaus, A., Imran, M., & Anuar, N. B. (2020). Blending big data analytics: Review on challenges and a recent study. *IEEE Access*, *8*, 3629–3645. https://doi.org/10.1109/ACCESS.2019.2923270

Arganza Salcedo, R., & Arroyo López, M. (2019). Big data: aplicaciones de la gestión del dato en las distintas etapas del funnel de conversión. *Revista de Marketing y Publicidad*, *1*, 39–68. https://doi.org/10.51302/marketing.2019.684

Babar, M., Ahmad, A., Tariq, M. U., & Kaleem, S. (2024). Real-time fake news detection using big data analytics and deep neural network. *IEEE Transactions on Computational Social Systems*, *11*(4), 5189–5198. https://doi.org/10.1109/TCSS.2023.3309704

Belov, V., Tatarintsev, A., & Nikulchev, E. (2021). Comparative characteristics of big data storage formats. *Journal of Physics: Conference Series*, *1727*(1), 012005. https://doi.org/10.1088/1742-6596/1727/1/012005

Cardas, C., Aldana-Martín, J. F., Burgueño-Romero, A. M., Nebro, A. J., Mateos, J. M., & Sánchez, J. J. (2023). On the performance of SQL scalable systems on Kubernetes: a comparative study. *Cluster Computing*, *26*(3), 1935–1947. https://doi.org/10.1007/s10586-022-03718-9

Çelik, C. (2024). Small to medium-sized enterprises data perception and applications. *Yönetim ve Ekonomi Araştırmaları Dergisi*, *22*(1), 154–170. https://doi.org/10.11611/yead.1410770

Cuomo, M. T., Tortora, D., Colosimo, I., Ricciardi Celsi, L., Genovino, C., Festa, G., & La Rocca, M. (2023). Segmenting with big data analytics and Python: A quantitative

exploratory analysis of household savings. *Technological Forecasting and Social Change*, *191*, 122431. https://doi.org/10.1016/j.techfore.2023.122431

Dangeti, P. (2017). *Statistics for machine learning*. Packt Publishing.

Gagliordi, N. (2023). *Diez riesgos comunes de las hojas de cálculo y soluciones para empresas*. Oracle. https://www.oracle.com/co/business-analytics/spreadsheet-risks/

García Núñez, A., & Olmedo Flores, J. L. (2021). Arquitectura distribuida de alta disponibilidad para la detección de fraude. *Revista Cubana de Ciencias Informáticas*, *15*, 199–224.

Gianniti, E., Ciavotta, M., & Ardagna, D. (2021). Optimizing quality-aware big data applications in the cloud. *IEEE Transactions on Cloud Computing*, *9*(2), 737–752. https://doi.org/10.1109/TCC.2018.2874944

Goh, B. W., Li, N., & Ranasinghe, T. (2024). Big data analytics and management forecasting behavior. *Accounting Horizons*, *38*(3), 59–76. https://doi.org/10.2308/HORIZONS-2020-145

Gu, R., Zhang, Y., Yin, L., Song, L., Huang, W., Yuan, C., Wang, Z., Zhu, G., & Huang, Y. (2023). Coral: Federated query join order optimization based on deep reinforcement learning. *World Wide Web*, *26*(5), 3093–3118. https://doi.org/10.1007/s11280-023-01156-0

Gunay, M., Ince, M. N., & Cetinkaya, A. (2019). Apache Hive performance improvement techniques for relational data. In *2019 International Artificial Intelligence and Data Processing Symposium (IDAP)* (pp. 1–6). IEEE. https://doi.org/10.1109/IDAP.2019.8875898

Haro Sarango, A. F., Martínez Yacelga, A. P., Nuela Sevilla, R. M., Criollo Sailema, M. E., & Pico Lescano, J. C. (2023). Inteligencia de negocios en la gestión empresarial: un análisis a las investigaciones científicas mundiales. *LATAM Revista Latinoamericana de Ciencias Sociales y Humanidades*, *4*(1), 3367–3382. https://doi.org/10.56712/latam.v4i1.493

Hussain, F., Nauman, M., Alghuried, A., Alhudhaif, A., & Akhtar, N. (2023). Leveraging big data analytics for enhanced clinical decision-making in healthcare. *IEEE Access*, *11*, 127817–127836. https://doi.org/10.1109/ACCESS.2023.3332030

Inoubli, W., Aridhi, S., Mezni, H., Maddouri, M., & Mephu Nguifo, E. (2018). An experimental survey on big data frameworks. *Future Generation Computer Systems*, *86*, 546–564. https://doi.org/10.1016/j.future.2018.04.032

Jaime, R. A., & Hasperué, W. (2021). *Estudio comparativo entre Apache Flink y Apache Spark*. [Trabajo de grado, Universidad Nacional de La Plata]. Universidad Nacional de La Plata.

Ketu, S., Mishra, P. K., & Agarwal, S. (2020). Performance analysis of distributed computing frameworks for big data analytics: Hadoop vs. Spark. *Computación y Sistemas*, *24*(2), 669–686. https://doi.org/10.13053/CyS-24-2-3401

Khalid, M., & Yousaf, M. M. (2021). A comparative analysis of big data frameworks: An adoption perspective. *Applied Sciences*, *11*(22), 11033. https://doi.org/10.3390/app112211033

Lapiedra, R., Forés, B., Puig-Denia, A., & Martínez-Cháfer, L. (2021). *Introducción a la gestión de sistemas de información en las empresas*. Universitat Jaume I. https://doi.org/10.6035/sapientia178

Lattuada, M., Barbierato, E., Gianniti, E., & Ardagna, D. (2022). Optimal resource allocation of cloud-based Spark applications. *IEEE Transactions on Cloud Computing*, *10*(2), 1301–1316. https://doi.org/10.1109/TCC.2020.2985682

Li, X., Yu, B., Feng, G., Wang, H., & Chen, W. (2021). LotusSQL: SQL engine for high-performance big data systems. *Big Data Mining and Analytics*, *4*(4), 252–265. https://doi.org/10.26599/BDMA.2021.9020009

Mayer-Schönberger, V., & Cukier, K. (2013). Big data: A revolution that will transform how we live, work, and think. *Houghton Mifflin Harcourt*.

Mohanty, A., Ramasamy, A. K., Verayiah, R., Bastia, S., Swaroop Dash, S., Soudagar, M. E. M., Yunus Khan, T. M., & Cuce, E. (2024). Smart grid and application of big data: Opportunities and challenges. *Sustainable Energy Technologies and Assessments*, *71*, 104011. https://doi.org/10.1016/j.seta.2024.104011

Montoya Suárez, L. M., & Gil Restrepo, G. A. (2018). Actualidad e importancia de la implementación de Big Data utilizando las herramientas Hadoop y Spark. *Lámpsakos*, *19*, 67–72. https://doi.org/10.21501/21454086.2403

Mostafa, J., Wehbi, S., Chilingaryan, S., & Kopmann, A. (2022). SciTS: A benchmark for time-series databases in scientific experiments and industrial Internet of Things. In *ACM International Conference Proceeding Series* (pp. 1–11). Association for Computing Machinery. https://doi.org/10.1145/3538712.3538723

Perafan-Villota, J. C., Mondragon, O. H., & Mayor-Toro, W. M. (2022). Fast and precise: Parallel processing of vehicle traffic videos using big data analytics. *IEEE Transactions on Intelligent Transportation Systems*, *23*(8), 12064–12073. https://doi.org/10.1109/TITS.2021.3109625

Ponnusamy, S., & Gupta, P. (2024). Scalable data partitioning techniques for distributed data processing in cloud environments: A review. *IEEE Access*, *12*, 26735–26746. https://doi.org/10.1109/ACCESS.2024.3365810

Reis, J., & Housley, M. (2022). *Fundamentals of data engineering*. O'Reilly Media, Inc.

Shareef, O. S. F., Hasan, R. F., & Farhan, A. H. (2023). Analyzing SQL payloads using logistic regression in a big data environment. *Journal of Intelligent Systems*, *32*(1), 20230063. https://doi.org/10.1515/jisys-2023-0063

Subbarao, D., Raju, B., Anjum, F., Rao, C. venkateswara, & Reddy, B. M. (2023). Microsoft Azure Active Directory for next-level authentication to provide a seamless single sign-on experience. *Applied Nanoscience (Switzerland)*, *13*(2), 1655–1664. https://doi.org/10.1007/s13204-021-02021-0

Syed, D., Zainab, A., Ghrayeb, A., Refaat, S. S., Abu-Rub, H., & Bouhali, O. (2021). Smart grid big data analytics: Survey of technologies, techniques, and applications. *IEEE Access*, *9*, 59564–59585. https://doi.org/10.1109/ACCESS.2020.3041178

Tang, S., He, B., Yu, C., Li, Y., & Li, K. (2022). A survey on Spark ecosystem: Big data processing infrastructure, machine learning, and applications. *IEEE Transactions on Knowledge and Data Engineering*, *34*(1), 71–91. https://doi.org/10.1109/TKDE.2020.2975652

Tariq, M. U., Babar, M., Poulin, M., Khattak, A. S., Alshehri, M. D., & Kaleem, S. (2021). Human behavior analysis using intelligent big data analytics. *Frontiers in Psychology*, *12*, 686610. https://doi.org/10.3389/fpsyg.2021.686610

Tekdogan, T., & Cakmak, A. (2022). Benchmarking Apache Spark and Hadoop MapReduce on big data classification. In *Proceedings of the 2022 International*

Conference on Data Management, Analytics and Innovation (ICDMAI) (pp. 176–181). ACM. https://doi.org/10.1145/3481646.3481649

Ullah, F., Dhingra, S., Xia, X., & Babar, M. A. (2024). Evaluation of distributed data processing frameworks in hybrid clouds. *Journal of Network and Computer Applications*, *224*, 103837. https://doi.org/10.1016/j.jnca.2024.103837

Wang, L., Pertheban, T. R. A., Li, T., & Zhao, L. (2024). Application of business intelligence based on big data in e-commerce data evaluation. *Heliyon*, *10*(2), e38768. https://doi.org/10.1016/j.heliyon.2024.e38768

Zaki, U. H. H., Kamsani, I. I., Ahmad Fadzil, A. F., Idrus, Z., & Kandogan, E. (2024). Big data: Issues and challenges in clustering data visualization. *Journal of Advanced Research in Applied Sciences and Engineering Technology*, *51*(1), 150–159. https://doi.org/10.37934/araset.51.1.150159

Zeydan, E., & Mangues-Bafalluy, J. (2022). Recent advances in data engineering for networking. *IEEE Access*, *10*, 34449–34496. https://doi.org/10.1109/ACCESS.2022.3162863

Zhao, Y., Ramos, Ma. F., & Li, B. (2024). Integrated framework to integrate Spark-based big data analytics and for health monitoring and recommendation in sports using XGBoost algorithm. *Soft Computing*, *28*(2), 1585–1608. https://doi.org/10.1007/s00500-023-09450-9