UNIVERSIDAD NACIONAL DE SAN JUAN
FACULTAD DE CIENCIAS EXACTAS FÍSICAS Y NATURALES
DEPARTAMENTO DE INFORMÁTICA

LICENCIATURA EN SISTEMAS DE INFORMACIÓN

INFORME DE TRABAJO FINAL DE GRADO

Apache Spark en un Contexto de Analítica e Inteligencia de Negocios:

Análisis de Datos de Transporte Público de Pasajeros de Corta Distancia de la Provincia de San Juan

AUTOR: Álvarez, Germán

ASESORA: Migani, Silvina **COASESORA: Balmaceda, Silvina**

SAN JUAN
2026

Agradecimientos

Quiero expresar mis mayores agradecimientos a mi familia, que siempre ha sido mi motor no solo en esta etapa, sino a lo largo de mi vida. De igual manera, a mi pareja y amigos más íntimos, por su apoyo y presencia constante sobre todo en los momentos de mayor dificultad. Extiendo este reconocimiento a mis compañeros y amistades de la facultad, por brindarme motivación y buenos momentos, haciendo que este trayecto sea satisfactorio. Por último, también quiero agradecer a mis dos asesoras Silvina Migani y Silvina Balmaceda, por su disposición no solo para resolver todas mis dudas, sino por su gran colaboración para desarrollar este trabajo.

Índice de Contenidos

Índice de Figuras

Índice de Tablas

Capítulo 1: Introducción

En la actualidad, tanto las empresas como los organismos gubernamentales se enfrentan a la creciente necesidad de analizar grandes y variados volúmenes de datos. En este contexto, la Analítica e Inteligencia de Negocios (ABI, por sus siglas en inglés de Analytics and Business Intelligence) adquiere un rol fundamental al permitir obtener información y conocimiento oportuno que asista la toma de decisiones estratégicas [1] [2]. De manera particular, en el ámbito empresarial la ABI promueve la eficiencia operativa y la competitividad en entornos dinámicos; mientras que, en el ámbito público, contribuye a optimizar la gestión y mejorar la calidad de los servicios ofrecidos a los ciudadanos [3] [4].

Por su parte, la Secretaría de Tránsito y Transporte de la Provincia de San Juan (STyT) posee un gran volumen de datos provenientes del Sistema Único de Boleto Electrónico (SUBE) [5], que necesita aprovechar para optimizar el uso de los recursos estatales e incrementar la satisfacción de sus usuarios, en otras palabras, lograr la mejora constante de la gestión del transporte público en la provincia. Sin embargo, la principal dificultad que enfrenta es el manejo masivo de estos datos, que requieren el uso de plataformas robustas, tanto para el almacenamiento como el procesamiento de los mismos [6].

En ese contexto, el presente trabajo propone desarrollar una solución ABI que fortalezca la capacidad de diagnóstico y toma de decisiones estratégicas de la Secretaría de Tránsito y Transporte, utilizando Apache Spark y servicios de Google Cloud Platform.

Esta tesis se desarrolló en el marco del Proyecto PIC “Analítica e Inteligencia de Negocios (ABI): Diseño de un modelo de proceso de negocio”, llevado a cabo en la Facultad de Ciencias Exactas, Físicas y Naturales de la Universidad Nacional de San Juan, aprobado en la Convocatoria 2022-2023, con Resolución N° 1501/23-R.

1.1. Antecedentes

Distintos estudios validan que la aplicación conjunta de la ABI y del Big Data en el dominio del transporte público ha crecido considerablemente en los últimos años, permitiendo así, la mejora continua en la planificación, operación y toma de decisiones.

A continuación, se presentan los trabajos más representativos relacionados a este trabajo.

El artículo de Welch & Widita [7] ofrece una revisión de las principales fuentes y técnicas de big data aplicadas al transporte público. Destaca la importancia de contar con registros masivos de datos, como los capturados por las tarjetas inteligentes y los desafíos asociados al volumen,

la calidad y la privacidad de los datos. Si bien brinda un marco conceptual del uso de datos masivos en movilidad, no implementa una solución tecnológica concreta.

En el artículo de Mobility Analytics [8] se presenta una solución de ABI tradicional para analizar datos operativos mediante la construcción de un Data Warehouse y paneles de control en Power BI. Muestra cómo las herramientas ABI facilitan la gestión del transporte, sin embargo, se basa en arquitecturas tradicionales.[1]

El estudio de Bańka [9] plantea el uso de herramientas ABI para mejorar el transporte público de pasajeros sobre datos de opinión de los usuarios (subjetivos), provenientes de encuestas digitales, comentarios de pasajeros, formularios web, etc., con el propósito de capturar, procesar y clasificar información cualitativa vinculada a la satisfacción del usuario, la percepción del servicio y los reclamos recurrentes.

Por último, el trabajo Martínez Soler [10] desarrolla una solución de análisis de datos abiertos publicados por el gobierno de Chicago, con foco en información vinculada a seguridad ciudadana y criminalidad. Para ello utiliza diversas fuentes oficiales que incluyen registros de víctimas de homicidios y tiroteos no fatales, datos de estaciones de policía, de arrestos, etc. El proyecto implementa un proceso completo de extracción, limpieza, integración y análisis utilizando Apache Spark, Hive y Power BI. Las principales problemáticas identificadas son la dispersión de los datos en múltiples archivos sin un esquema unificado, la dificultad para analizar eficientemente grandes volúmenes de información bajo métodos tradicionales y la adopción de una arquitectura que corre en un entorno de cómputo local. El trabajo demuestra cómo las tecnologías de Big Data permiten transformar datasets heterogéneos y voluminosos en información estratégica para la toma de decisiones en materia de seguridad pública. Si bien constituye el antecedente más cercano en cuanto al uso de tecnologías, ya que comparte la orientación tecnológica con el presente trabajo, se diferencia claramente en el dominio analizado y en la arquitectura final adoptada, lo que valida la pertinencia y originalidad del presente proyecto.

1.2. Formulación del Problema y Justificación

El sistema SUBE constituye la principal fuente de información transaccional sobre la movilidad urbana en el transporte público de pasajeros en Argentina [5]. A nivel nacional, los datos se

[1] En este informe se utilizarán indistintamente los términos ‘dashboard’ y ‘panel de control’ para referirse a paneles interactivos de visualización de indicadores de negocio.

procesan de manera centralizada y se ponen a disposición de las provincias mediante paneles de control que facilitan la visualización de información relevante. Además, mensualmente se envían tanto los datos detallados de cada transacción como resúmenes de los mismos, utilizados principalmente en la elaboración de dichos paneles.

En la Provincia de San Juan, la STyT emplea algunos informes locales elaborados a partir de datos resumidos y, en gran medida, depende de los paneles elaborados a nivel nacional. Si bien estas visualizaciones resultan útiles, no pueden ser modificadas, ampliadas, ni personalizadas según las necesidades de la provincia, ya que su estructura, niveles de detalle e indicadores son definidos centralmente a nivel nacional. Esta limitación impide realizar análisis específicos orientados al contexto local dificultando la obtención de información estratégica ajustada a las prioridades de la secretaría provincial.

Este trabajo presenta el desarrollo de una solución ABI específica para la STyT de la provincia de San Juan construida a partir de la totalidad de los datos transaccionales y de geolocalización disponibles. El gran volumen de datos hace necesaria la utilización de técnicas y herramientas de Big Data para procesar, integrar y visualizar la información mediante plataformas como Apache Spark, BigQuery y Looker Studio. De esta manera, la propuesta permite mejorar la capacidad analítica y la toma de decisiones en la secretaría, y al mismo tiempo brinda al becario un caso real donde aplicar metodologías y herramientas actuales de ABI y Big Data, reforzando la comprensión de los procesos necesarios para transformar datos complejos en información estratégica.

*1.3. **Objetivos***

Objetivo General

Desarrollar y evaluar una solución ABI basada en Apache Spark que permita integrar, procesar y visualizar datos del transporte público de la provincia de San Juan, demostrando su viabilidad técnica y su potencial aporte al análisis estratégico.

Objetivos Específicos

- Comprender la temática ABI y los principios de procesamiento de grandes volúmenes de datos.

- Analizar las características y capacidades de plataformas para el procesamiento y almacenamiento de grandes volúmenes de datos, especialmente relacionadas a Apache Spark. En primera instancia se pretende construir una pequeña solución de prueba en entorno de trabajo local, y luego explorar hacia entornos basados en la nube.
- Desarrollar e implementar una solución ABI que integre, procese y almacene datos transaccionales y de geolocalización, utilizando Apache Spark y BigQuery.
- Generar dashboards interactivos en Looker Studio que permitan visualizar indicadores relevantes del servicio de transporte público, facilitando la exploración y análisis por parte de la STyT de la provincia de San Juan.

1.4. Metodología

El presente trabajo se enmarca en una investigación de carácter aplicado, orientada al diseño y desarrollo de una solución de ABI para el análisis de datos del transporte público de pasajeros de corta distancia en la provincia de San Juan. El estudio toma como caso concreto los datos provenientes del sistema SUBE provistos por la Secretaría de Tránsito y Transporte (STyT), en el marco institucional que regula la gestión del transporte público a nivel nacional y provincial.

El enfoque metodológico es fundamentalmente cuantitativo, dado que se basa en el procesamiento masivo de registros digitales generados por el sistema en su operación habitual. El conjunto de datos analizado comprende información transaccional (validaciones de boletos) y registros de geolocalización GPS de las unidades de transporte, correspondientes a los años 2023 y 2024 (hasta marzo). Se trabaja con la totalidad de los registros suministrados para dicho intervalo, por lo que no se aplican técnicas de muestreo.

Las unidades de análisis están constituidas por cada registro individual generado por el sistema, tanto transacciones como posiciones geográficas. A partir de estos registros se definieron variables de interés tales como fecha y franja horaria, línea o recorrido, tipo de transacción, beneficios aplicados y ubicación espacial, entre otras, que permiten caracterizar el comportamiento de la demanda y la dinámica operativa del servicio.

El trabajo se desarrolló mediante una serie de etapas sucesivas. En primer lugar, se realizó un análisis de los archivos de datos con el fin de comprender su estructura, semántica, volumen y calidad. Posteriormente, se diseñó un esquema de procesamiento orientado a la integración,

limpieza y transformación de la información. Sobre esta base, se implementaron procesos de agregación y cálculo para la generación de indicadores analíticos relevantes.

La solución fue desarrollada utilizando tecnologías orientadas al procesamiento distribuido de grandes volúmenes de información, particularmente Apache Spark, lo que permitió gestionar eficientemente la escala y complejidad de los datos analizados. Finalmente, se construyeron visualizaciones y representaciones analíticas destinadas a facilitar la interpretación de los resultados y la exploración de patrones significativos en la operación del sistema.

Capítulo2: Marco Teórico

El marco teórico de este trabajo se organiza a partir de un conjunto de ejes conceptuales, que se enuncian y explican a continuación.

2.1. Analítica e Inteligencia de Negocios (ABI)

Según la definición presentada por Gartner [11], la Analítica e Inteligencia de Negocios (ABI) constituye un término amplio que engloba aplicaciones, infraestructura, herramientas y prácticas destinadas a facilitar el acceso y análisis de la información con el fin de optimizar la toma de decisiones y el desempeño organizacional. En la actualidad, las organizaciones generan y almacenan grandes volúmenes de datos que, cuando se gestionan adecuadamente, pueden transformarse en una fuente significativa de información y conocimiento. Esto otorga a los directivos una visión más clara y precisa sobre el funcionamiento de la organización y les permite fundamentar sus decisiones en evidencias [1] [2].

2.1.1. Proceso de Construcción

Una vez presentada la definición conceptual de ABI, resulta pertinente profundizar en el proceso y los componentes tecnológicos que permiten transformar datos operacionales en información/conocimiento útil para el análisis táctico y estratégico.

En este sentido, el proceso de construcción de un entorno ABI se apoya en un conjunto de actividades que, de manera coordinada, habilitan el flujo de datos desde los sistemas de origen hasta los mecanismos de explotación analítica. El proceso inicia fundamentalmente en los sistemas operacionales, donde se registran las transacciones propias del negocio. Estos datos se caracterizan por su granularidad fina, su actualización frecuente y su orientación eminentemente operativa. A través de un proceso sistemático de extracción, limpieza, validación y agregación, los datos “brutos” son transformados para adquirir una estructura adecuada para su uso analítico [12] [13] [14] [15]. Este flujo conceptual queda representado en la Figura 1, que sintetiza de manera abstracta las etapas fundamentales presentes en la mayoría de las soluciones ABI, independientemente de la tecnología empleada [16].

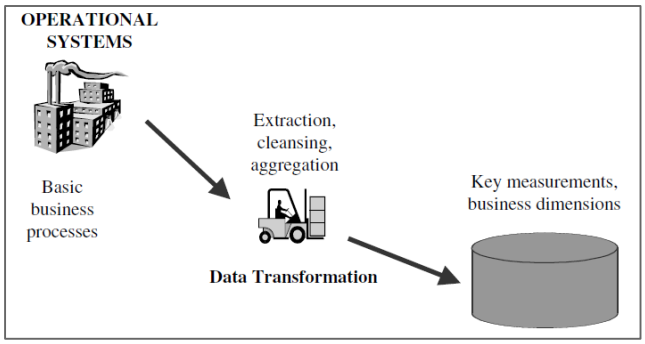


Figura 1: Proceso Elemental dentro de soluciones ABI (Extraído de Data Warehousing Fundamentals: A Comprehensive Guide for IT Professionals. Paulraj Ponniah)

A partir de ese esquema general, resulta pertinente profundizar en los procesos que permiten implementar la integración de datos dentro de un entorno analítico. Tradicionalmente, las tareas implicadas se han estructurado bajo el modelo de procesamiento ETL (Extract, Transform, Load), en el cual los datos se extraen desde las fuentes, se someten a procesos de depuración y transformación en un área de staging, y posteriormente se cargan en el repositorio analítico (Data Warehouse o Almacén de Datos). Este enfoque ha sido el predominante por décadas y sus principios continúan siendo fundamentales para comprender el movimiento y la preparación de datos en este contexto [12] [13] [14].

Sin embargo, la evolución de las plataformas tecnológicas ha dado lugar a una variante denominada ELT (Extract, Load, Transform), donde en este caso los datos "brutos" se cargan inicialmente en almacén (generalmente un Data Warehouse, Data Lake o entorno de almacenamiento escalable) y las transformaciones se ejecutan directamente dentro del motor de procesamiento. Este cambio en el orden de las etapas responde a las capacidades modernas de cómputo distribuido y almacenamiento masivo, que permiten realizar transformaciones complejas y operaciones analíticas directamente sobre grandes volúmenes de datos de manera eficiente. A pesar de sus diferencias operativas, ETL y ELT coexisten plenamente en la actualidad, y su adopción depende del tipo de arquitectura, las herramientas disponibles y los requisitos de cada organización. Ambos modelos comparten un mismo propósito: garantizar la calidad, consistencia y organización de los datos para su explotación analítica [17].

2.1.2. Arquitectura

La Figura 2 presenta una arquitectura conceptual ampliamente aceptada en la literatura, propuesta por Ponniah [16], cuyo valor reside en la identificación de funciones esenciales de un entorno ABI, más que en la descripción de una implementación tecnológica específica. Esta representación incluye las principales áreas funcionales (fuentes de datos, zona de staging, repositorio analítico, metadatos y capa de entrega de información) y es lo suficientemente general como para reflejar tanto arquitecturas basadas en ETL como enfoques modernos sustentados en ELT o procesamiento distribuido.

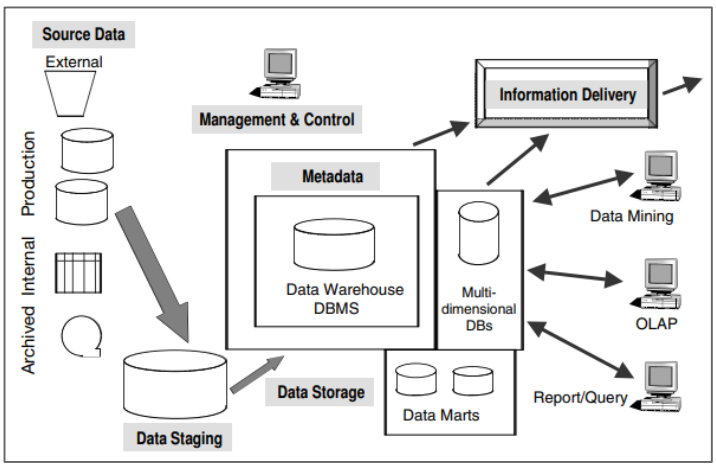


Figura 2: Componentes fundamentales de un entorno ABI (Extraído de Data Warehousing Fundamentals: A Comprehensive Guide for IT Professionals. Paulraj Ponniah)

Dentro de esta arquitectura se pueden distinguir las siguientes capas y elementos:

- ***Capa Fuentes de Datos***

Incluyen sistemas transaccionales, archivos, sensores, APIs externas y cualquier otro repositorio que genere o almacene información relevante. Estas fuentes suelen presentar heterogeneidad, distintos niveles de calidad y frecuencias variables de actualización, lo que motiva la existencia de procesos de integración específicos.

- ***Capa de Ingesta e Integración***

Los procesos ETL (Extract–Transform–Load) o ELT (Extract–Load–Transform) constituyen el mecanismo mediante el cual los datos se extraen de las fuentes, se depuran, transforman y unifican, y finalmente se cargan en estructuras optimizadas para análisis. Esta etapa resulta crítica, ya que determina la consistencia, confiabilidad y trazabilidad del dato analítico.

- ***Capa de Almacenamiento (Almacenes de Datos/Data Warehouse y Data Marts)***

Una vez integrados, los datos se organizan en un Data Warehouse o en Data Marts temáticos, siguiendo principios ampliamente difundidos en la literatura [12] [13] [14]. Estos repositorios posibilitan consultas eficientes, análisis histórico y consolidación de información proveniente de múltiples sistemas. Aquí cobra relevancia el modelado multidimensional, ya que, mediante la identificación de hechos, dimensiones, jerarquías y medidas, se representan las actividades del negocio de manera intuitiva y orientada al análisis [15].

Los hechos conforman eventos u objetos principales de análisis, donde las dimensiones representan categorías o conceptos asociados a esos hechos y por último las medidas conforman valores cuantificables de análisis. Para poner un ejemplo, se podría tomar como hecho las ventas de una empresa, con dimensiones cliente y producto, cuya medida puede ser total de ventas.

A la hora de diseñar un almacén de datos, existen dos modelos de diseño que se diferencian tanto en normalización como también en complejidad de las consultas.

- **Esquema Estrella:** Modelo desnormalizado que se enfoca en el diseño de menos relaciones, dando como resultado consultas de menor complejidad y más rápidas.
- **Esquema Copo de Nieve:** Modelo normalizado que se enfoca en evitar redundancia de datos al costo de complejizar las consultas. La jerarquía entre dimensiones queda explícita entre las tablas de una misma dimensión (relación de padre-hijo).

En el Modelo Estrella las tablas de dimensión podrían no estar en 3FN, y por ende, existen datos redundantes. El modelo copo de nieve resuelve estos problemas de normalización, aunque sea más complejo de consultar. Las Figuras 3a y 3b muestran un ejemplo de ambos esquemas.

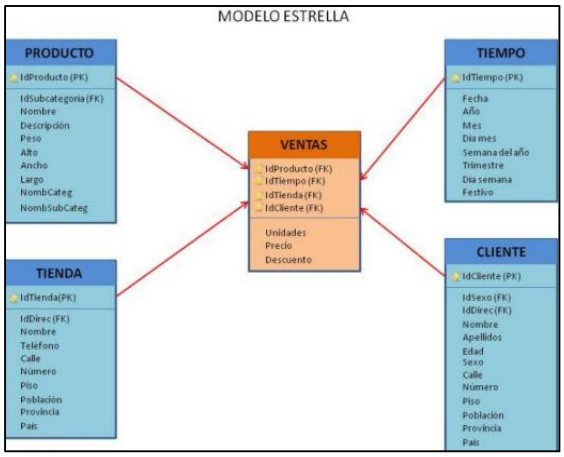


Figura 3a: Ejemplo de modelo estrella (Extraído de Desarrollo de una herramienta Big Data para el análisis de Chicago Open Data)

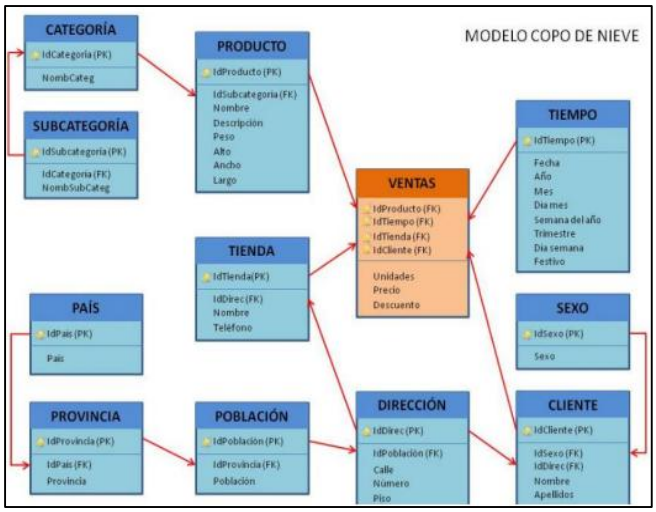


Figura 3b: Ejemplos de modelo copo de nieve mundoDB (Extraído de Desarrollo de una herramienta Big Data para el análisis de Chicago Open Data)

A la hora de diseñar un almacén de datos, se debe optar por el esquema que se adecue mejor a las necesidades del proyecto [18].

- ***Capa Analítica***

El procesamiento analítico puede adoptar modalidades diversas:

- **OLAP (Online Analytical Processing)** para análisis multidimensional.
- **Consultas ad-hoc** para exploración puntual de datos.
- **Modelos estadísticos o de machine learning** para análisis avanzado. Este nivel permite realizar comparaciones, tendencias, segmentaciones y análisis complejos de forma ágil.

Es importante tener en cuenta que el procesamiento de la información puede realizarse de dos formas: en tiempo real (Stream Processing) o en lotes (Batch Processing). La diferencia radica en si se desea priorizar mayor velocidad para tiempos de respuesta, por menor volumen de datos, tal como lo refleja el primer caso, en contraste con el procesamiento en lotes.

- ***Capa de Visualización y Consumo***

Esta capa constituye el punto de interacción entre la solución ABI y los usuarios finales, siendo responsable de presentar la información de manera comprensible, accesible y orientada a la toma de decisiones. Esta capa incluye:

- Métricas cuantitativas que permiten monitorear el progreso respecto a objetivos estratégicos.
- Dashboards o Paneles de Control, que integran múltiples métricas, gráficos y tablas, diseñadas para facilitar la lectura rápida y la toma de decisiones.
- Reportes interactivos orientados a análisis específicos.

2.1.3. Tipos de Analítica

Se reconocen distintos tipos de analítica según el objetivo del análisis y el tipo de conocimiento que se busca obtener a partir de los datos, a saber:

- **Descriptiva**: Responde a la pregunta ¿Qué ha ocurrido? Resume los datos históricos (por ej. mediante informes, gráficos o dashboards) para identificar patrones y comportamientos. En este contexto, los Indicadores Clave de Desempleo, mencionados comúnmente como KPIs (por sus siglas en inglés) son métricas diseñadas para medir de

manera objetiva y cuantificable el rendimiento de un proceso, sistema u organización, frente a metas definidas. Su objetivo es transformar grandes volúmenes de datos en información sintética y comprensible, que permita monitorear tendencias, detectar anomalías y apoyar la toma de decisiones estratégicas.

- **Predictiva**: Response a la pregunta ¿Qué podría ocurrir? Utiliza modelos estadísticos y algoritmos de aprendizaje automático (machine learning) para predecir resultados futuros o comportamientos con base en datos históricos.
- **Prescriptiva**: Responde a la pregunta ¿Qué deberíamos hacer? A partir de predicciones y del entendimiento del negocio, genera recomendaciones específicas de acción para alcanzar objetivos definidos (por ejemplo: asignar recursos de forma diferente, lanzar determinada campaña, optimizar un proceso).

Cabe mencionar que algunas fuentes también distinguen una cuarta categoría, la analítica diagnóstica, que busca explicar por qué ocurrió algo, situándola conceptualmente entre la analítica descriptiva y la predictiva [2].

2.1.4. Enfoques y Técnicas

En el marco de la ABI, existen distintas disciplinas, enfoques y técnicas que contribuyen al análisis avanzado de datos, que se complementan dentro de soluciones analíticas modernas. Se destacan los siguientes:

- **Data Mining**: Disciplina que se ocupa de explorar grandes conjuntos de datos con el objetivo de identificar patrones, relaciones, regularidades o estructuras relevantes, apoyándose en técnicas estadísticas, algoritmos computacionales y métodos de aprendizaje automático [17].
- **Machine Learning**: Disciplina que desarrolla algoritmos capaces de aprender patrones a partir de los datos y mejorar su desempeño en una tarea específica sin ser programados explícitamente, apoyándose en datos históricos, modelos estadísticos y procesos iterativos de entrenamiento y validación [17].

2.2. Big Data

Desde la perspectiva de la ABI, el término Big Data refiere a conjuntos de datos cuya escala, complejidad y dinámica exceden la capacidad de las arquitecturas y herramientas tradicionales de gestión y procesamiento de datos, requiriendo enfoques distribuidos, escalables y tolerantes a fallos para su almacenamiento, integración y análisis. No se trata únicamente de grandes volúmenes de información, sino también de la diversidad de fuentes, la velocidad de generación y la complejidad inherente a los datos que deben ser gestionados y procesados [17].

En un sentido amplio, Big Data describe el contexto tecnológico y conceptual que posibilita transformar datos masivos y heterogéneos en información y conocimiento útil, habilitando la toma de decisiones basada en evidencia empírica, patrones detectados y modelos analíticos. En dominios intensivos en datos, como el transporte público, este enfoque permite analizar millones de transacciones, recorridos y eventos geográficos con el objetivo de mejorar la eficiencia del sistema, apoyar la planificación y optimizar la experiencia de los usuarios.

2.2.1. Origen y Evolución

El concepto de Big Data surge a comienzos del siglo XXI como respuesta al crecimiento exponencial de la información digital generada por internet, sensores, dispositivos móviles y redes sociales. Aunque el término fue popularizado en 2001 por Doug Laney, analista de Gartner, quien formuló el famoso modelo de las tres “V”, cuyos fundamentos ya se desarrollaban desde la década de 1990 cuando las bases de datos relacionales comenzaron a quedar limitadas para manejar datos no estructurados y flujos continuos de información.

El modelo de las tres “V” se refieren a **Volumen** donde el tamaño y la cantidad de datos es importante; **Velocidad** que hace referencia al ritmo al que se reciben los datos; y por último la **Variedad** que hace referencia a los tipos de datos disponibles (estructurado, no estructurado) [19].

Con el tiempo, este modelo se amplió incorporando otras dimensiones:

- **Veracidad**: Evalúa la calidad y confiabilidad de los datos.
- **Valor**: Hace referencia a la utilidad real de la información analizada.

El auge del Big Data coincidió con la aparición de plataformas distribuidas como Hadoop (2006), que introdujo un paradigma revolucionario: en lugar de concentrar toda la información en un único servidor de gran capacidad, distribuir los datos y el procesamiento entre cientos o

miles de nodos interconectados. Este modelo de cómputo distribuido abrió la puerta a la gestión eficiente de volúmenes masivos de información, permitiendo la evolución hacia arquitecturas escalables y tolerantes a fallos.

2.2.2. Tecnologías de Big Data

El ecosistema de Big Data está compuesto por un stack de tecnologías que cubren distintas etapas del ciclo de vida de los datos. Estas herramientas evolucionaron con el objetivo de manejar la complejidad, el tamaño y la velocidad de la información moderna [20]. A continuación, se describen las principales categorías tecnológicas: La tabla 1 describe las principales categorías tecnológicas:

Categorías	**Descripción de Tecnologías Big Data**
Ingesta y Almacenamiento distribuido	- HDFS es el mencionado sistema de archivos distribuido de Hadoop. Una de las primeras tecnologías tolerante a fallos y escalable. - Apache Kafka para la ingesta en tiempo real de flujos de datos (streaming), donde la información llega desde múltiples fuentes. - Bases de Datos NoSQL que almacena datos semiestructurados o no estructurados, con alta disponibilidad y escalabilidad horizontal.
Procesamiento masivo distribuido	- MapReduce es el modelo de programación de Hadoop que se usa para procesamiento en lote, limitado en flexibilidad y latencia. - Spark es la evolución de MapReduce, al utilizar un modelo basado en memoria. Ofrece una API que se integra con múltiples elementos.
Integración y Orquestación	- Apache Airflow o Oozie para gestionar flujos de trabajo (ETL pipelines), control de dependencias y ejecución de tareas. - Google Cloud Dataproc, AWS EMR y Azure Synapse son servicios administrados que facilitan la creación y ejecución de clusters Spark o Hadoop sin configuración manual de infraestructura.
Almacenamiento Analítico y Data Warehouse	- Google BigQuery y Amazon Redshift son almacenes de datos analíticos que permiten correr consultas SQL sobre millones de registros. - Data Lakes son repositorios basados en almacenamiento de objetos (como Google Cloud Storage) que permiten realizar diferentes tipos de análisis posterior su almacenamiento.
Visualización	- Looker Studio, Tableau y Power BI son plataformas de visualización que permiten transformar resultados analíticos en dashboards interactivos para la toma de decisiones.

Tabla 1: Descripción de tecnologías Big Data

2.2.3. Apache Spark como tecnología de Big Data

Apache Spark es un sistema informático distribuido de código abierto que se caracteriza por mejorar la velocidad y el rendimiento de aplicaciones Big Data. En 2009 surgió como un proyecto open source en los laboratorios de investigación de la Universidad de Berkeley, donde desde 2013 comenzó a tomar popularidad.

Es una tecnología revolucionaria en el ecosistema Big Data que aborda los principales desafíos del manejo y análisis de grandes volúmenes de datos en contextos donde el volumen, la velocidad y la variedad de los datos son factores críticos, Spark ofrece una solución eficiente, escalable y versátil para una amplia gama de casos de uso [19].

Su popularidad se debe a un modelo de procesamiento persistente en memoria que mejora la velocidad y eficiencia de las tareas, integrando funcionalidades para cargas de trabajo por lotes, streaming y aprendizaje automático, todo ello desde una arquitectura unificada, conformando una solución integral para las necesidades de Big Data.

Ecosistema y Componentes

Algunos de los módulos más importantes que se integran con Spark son:

- **Spark Core Engine:** Es el núcleo del ecosistema Spark, siendo el motor responsable de la planificación, distribución y monitorización de aplicaciones ejecutadas en un entorno computacional, sea clúster o máquina local.
- **Spark SQL:** Es el componente que da soporte a consultas interactivas sobre conjuntos de datos por medio del lenguaje SQL.
- **Spark Streaming:** Es un componente que permite el procesamiento de aplicaciones y análisis de datos en tiempo real, dividiendo los datos en pequeños micro-lotes (DStreams) para su procesamiento rápido. Fue reemplazado por su sucesor Structured Streaming.
- **MLlib:** Es la librería de aprendizaje automático de Spark. Permite entrenar modelos a altas velocidades y desarrollar algoritmos escalables para tareas como clasificación, regresión, agrupación y filtrado colaborativo.
- **GraphX:** Es una API de procesamiento distribuido en estructura de datos de tipo grafo, permitiendo a los usuarios trabajar con algoritmos de grafos paralelos.

- **SparkR:** Es el paquete que integra el lenguaje estadístico R con Spark, lo que permite que científicos de datos con experiencia en R interactúen con Spark para realizar análisis de datos y demás trabajos estadísticos.

Lenguajes soportados y APIs

Además, es compatible con varios lenguajes de programación incluyendo Scala, Java y Python. La elección del lenguaje a utilizar depende de dos enfoques: Si se busca un mejor rendimiento, las mejores opciones son Scala o Java pues son los lenguajes nativos de Spark, teniendo la integración más estrecha sin ninguna sobrecarga adicional. Sin embargo, si se busca desarrollar una solución de Spark sin una curva de aprendizaje compleja, la mejor opción es Python por su facilidad de uso.

Relación con Apache Hadoop

Desde sus inicios Spark ha extendido el stack del Big Data tradicional superando las limitaciones de su antecesor Apache Hadoop, que al igual que Spark, es un motor de código abierto que almacena y procesa grandes conjuntos de datos, con limitaciones en tiempos de respuesta y flexibilidad.

Ambos representan marcos de trabajo para Big Data, pero están destinadas a propósitos diferentes: Hadoop se centra principalmente en el procesamiento por lotes y cuenta con un ecosistema bien establecido de tecnologías, mientras que Spark admite procesamiento por lotes y en tiempo real, siendo mucho más rápido pues mantiene los datos en memoria en lugar de leerlos desde el disco. Spark cuenta con un ecosistema más limitado, pero puede integrarse con mayores componentes.

Spark es un excelente motor de procesamiento para grandes volúmenes y es considerado el sucesor de Hadoop, sin embargo, es importante tener en cuenta que Spark no almacena los datos que lee, solo trabaja con ellos en memoria lo que lo hace más rápido [10]. Si lo que se desea es contar con un sistema de almacenamiento robusto o procesar volúmenes que no caben en memoria, la mejor opción es trabajar con Apache Hadoop. Resulta muy eficiente combinar el modelo de procesamiento de Spark con todo el ecosistema de Hadoop, aprovechando las capacidades de cada uno para conformar un ecosistema de Big Data sumamente robusto y eficiente.

A continuación, se destacan los siguientes componentes de Hadoop que pueden integrarse con Spark:

- **MapReduce:** Es el modelo de programación que procesa datos de forma distribuida y paralela, estando conformado por dos fases: Map y Reduce. La fase de map se encarga de recoger y mapear pares de datos (clave-valor) de los distintos nodos para que luego la fase Reduce, reduzca estos datos de entrada a una única salida, aprovechando al máximo las capacidades de distribución y paralelismo. Este modelo fue sustituido por Spark ya que almacena los datos en memoria en lugar de en disco.
- **HDFS:** Es el sistema de archivos distribuido de Hadoop que gestiona todo el almacenamiento, particionando el conjunto de datos a resguardar entre las distintas máquinas de forma distribuida a través de una red. Es un sistema tolerante a fallos y trabaja de forma eficiente minimizando a una sola escritura y múltiples lecturas.
- **YARN:** Es el componente que se encarga de mejorar la gestión de recursos de clúster, pudiendo escalar la cantidad de nodos gestionando eficientemente la asignación y el compartimiento de los recursos.

Arquitectura de Spark

La arquitectura de Spark se basa en un modelo de Driver y Worker Nodes que colaboran para ejecutar aplicaciones de forma distribuida, permitiendo procesar grandes cantidades de datos de manera eficiente [21]. Los componentes de esta arquitectura se muestran en la Figura 4:

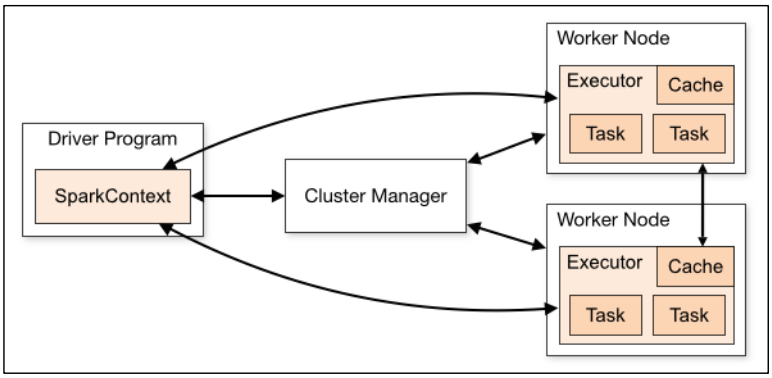


Figura 4: Arquitectura de Apache Spark (Extraído de ¿Cómo es la Arquitectura de Apache Spark? Keep coding Blog)

- **Application:** Programa codificado en un lenguaje que está construido sobre Spark y consta de un Driver Program y Executors.

- **Driver Program:** Proceso que ejecuta la función main() de la aplicación y crea el SparkContext. Este es el corazón de la aplicación, ya que planifica la ejecución de las tareas en el clúster.
- **Cluster Manager:** Servicio externo como Yarn o Kubernetes que se encarga de adquirir y gestionar los recursos necesarios para la aplicación en el clúster. El Driver Program interactúa con el Cluster Manager para solicitar recursos.
- **Worker Node:** Son los nodos esclavos o trabajadores del clúster que ejecutan el código de la aplicación.
- **Executor:** Proceso lanzado en un Worker Node para la aplicación específica. Los Executors son responsables de ejecutar las Tasks y mantener los datos en memoria o en disco a través del Caché. Cada aplicación tiene sus propios Executors.
- **Task:** Es la unidad de trabajo más pequeña que se envía a un Executor para ser ejecutada.

Manejo de datos en Spark

La principal funcionalidad de Spark es la creación y gestión de datos distribuidos y resilientes (RDD), que son colecciones de datos con tolerancia a fallos que se distribuyen entre los nodos del clúster y trabajan en paralelo, constituyendo particiones de datos. Son inmutables ya que no se pueden modificar luego de crearse y son resilientes, pues ante fallos se regeneran automáticamente. Su funcionamiento se basa en la transformación de un RDD para la obtención de un nuevo RDD [22].

Los RDD tienen un ciclo de vida conformado por transformaciones y luego acciones que son puntos finales de procesamiento, donde los datos se almacenan en una fuente externa dado por terminado el proceso ETL (Figura 5).

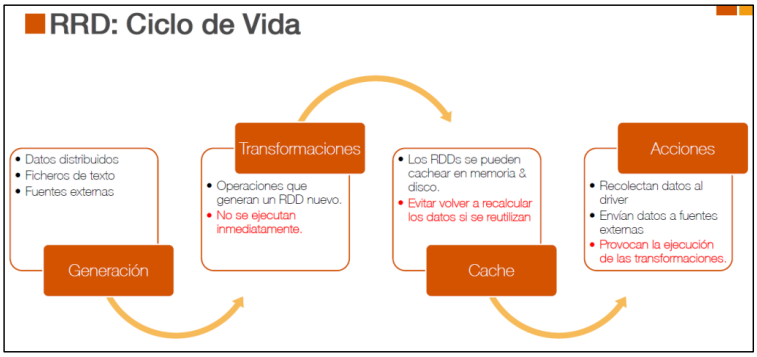


Figura 5: Ciclo de vida de RDD (Extraído de ¿Qué son los Resilient Distributed Datasets? Keep coding blog)

Sin embargo, desde la creación de Spark, las colecciones de datos también han ido evolucionando (Figura 6).

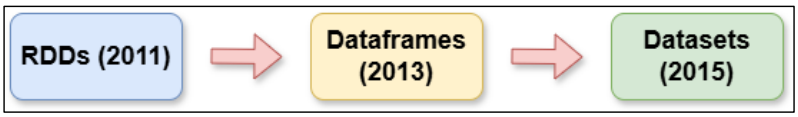


Figura 6: Evolución de las colecciones de datos en Apache Spark.

En 2013 aparecen los famosos Dataframes como una mejora de los RDD en términos de optimización, inferencia de esquema y organización basada en columnas. Los Dataframes poseen una API para realizar operaciones de agregación y se destacan por ser la colección más rápida.

En 2015 surgieron los Datasets como una extensión de los Dataframes con esquema del tipo clave-valor, con más características de seguridad de tipo de datos y una interfaz orientada a objetos. Utiliza el motor SQL para inferir el esquema y resultan más rápidos que los RDDs pero menos que los Dataframes.

Es recomendable usar dataframes o datasets cuando se requiere una semántica más rica, abstracciones de alto nivel y seguridad de tipo de datos. Sin embargo si se necesita una aplicación menos compleja con más énfasis en el pre procesamiento, conviene utilizar RDD.

Procesamiento distribuido y particionamiento de datos

Una partición en Spark representa una unidad lógica de distribución de datos, donde cada dataframe se divide internamente en particiones que se procesan de forma independiente en distintos nodos del clúster, de manera que cada uno ejecuta una fracción del trabajo total en paralelo [23].

El particionamiento de los datos constituye un aspecto central del modelo de ejecución de Spark, ya que determina el grado de paralelismo y la eficiencia en el uso de los recursos computacionales. Si hay muchas particiones pequeñas, se origina una sobrecarga de coordinación (por exceso de tareas y comunicaciones), mientras que, si hay pocas y muy grandes particiones, algunos nodos quedarán ociosos mientras otros soportan la carga total, sin aprovechar correctamente los recursos. Para ajustar manualmente la cantidad de particiones se utilizan operaciones como repartition o coalesce que permiten equilibrar el trabajo y optimizar los recursos del clúster. Constituyen mecanismos para controlar el paralelismo y la granularidad del procesamiento:

- **repartition**: Crea un nuevo dataframe con el número de particiones especificado, redistribuyendo los datos de forma aleatoria o basándose en una columna clave de partición. Este tipo de redistribución garantiza que las filas con la misma clave estén en la misma partición, lo que minimiza el shuffle posterior durante el join.
- **coalesce**: Por el contrario esta operación reduce el número de particiones existentes, fusionando particiones adyacentes. Es útil en etapas finales como antes de escribir el resultado en BigQuery, reduciendo la cantidad de particiones de salida sin incurrir a una gran sobrecarga de recursos.

Operaciones de Shuffle

El término shuffle describe el proceso mediante el cual Spark redistribuye datos entre particiones y nodos, por lo general como consecuencia de transformaciones que requieren agrupar o combinar información (groupBy, join, distinct, reduceByKey, entre otras) [24].

El shuffle se origina cuando los datos intermedios permanecen en memoria y se transfieren por red hacia otras particiones, generando un costo elevado en tiempo y recursos de E/S. Para mitigar su impacto, se aplican las siguientes estrategias:

- Filtrado y proyección temprana de columnas antes del join, reduciendo el volumen de datos que se redistribuyen.

- Uso de particiones por clave de unión para evitar redistribuciones innecesarias.
- Broadcast de tablas pequeñas, que eliminan completamente la necesidad de shuffle y redistribución.

Comprender el mecanismo de shuffle y controlarlo adecuadamente es fundamental para alcanzar un rendimiento óptimo en jobs distribuidos.

Transformaciones y Acciones

Las operaciones sobre dataframes se clasifican en transformaciones o acciones. Una transformación implica crear un nuevo dataframe a partir de otro, usando funciones como select, filter, join, groupBy o withColumn. No ejecutan inmediatamente el cálculo, sino que construyen un plan lógico DAG (grafo acíclico dirigido de operaciones) que Spark evalua más adelante, conocido como mecanismo de evaluación perezosa. Por otro lado, las acciones son operaciones que disparan la ejecución real del DAG, por medio de funciones como count(), show(), write().

El DAG se ejecuta al desencadenar una acción, interviniendo el optimizador Catalyst para optimizar el plan de ejecución físico, reorganizando las transformaciones e intentando minimizar el shuffle.

En resumen, las transformaciones definen el plan de lo que se debe hacer, las acciones desencadenan el plan, y el Catalyst decide la mejor manera de ejecutar el plan, pudiendo manipular particiones, shuffles y broadcasts.

Broadcast y Broadcast Join

El broadcast es una operación mediante la cual Spark envía una copia de un conjunto de datos pequeño a cada nodo del clúster. De esta forma, los ejecutores pueden acceder localmente a esa información sin necesidad de realizar transferencias o reparticiones [25].

Su uso más común es en los broadcast joins, donde una tabla pequeña (por ej., el archivo GPS o el índice IPC) se replica en memoria en todos los nodos y se une con una tabla mucho mayor (las transacciones).

Esto convierte en algunos casos al join como una operación local (map-side join), eliminando el costoso shuffle y acelerando significativamente la ejecución.

Cache y Persist

Operaciones que permiten almacenar en memoria los resultados intermedios de un dataframe, evitando que Spark vuelva a recalcularlos en caso de ser reutilizados. Es posible utilizarlas cuando se trabaja con el mismo dataframe varias veces y se debe solicitar una acción como count, show o write, donde Spark debería recalcular toda la transformación en cada caso, generando mayores costos y redundancia de cómputo.

Sin embargo presentan una diferencia: El método cache() almacena los datos únicamente en memoria (MEMORY_ONLY). Mientras que persist() permite especificar otros niveles de almacenamiento, como MEMORY_AND_DISK, que guarda en disco los datos que no caben en memoria. En ambos casos cuando ya no sea necesario almacenar los datos intermedios, se debe liberar memoria usando el método unpersist() [26].

User Defined Function (UDF)

Las UDFs extienden la funcionalidad de Spark definiendo funciones personalizadas que se aplican sobre columnas de un dataframe, agregando lógica personalizada además de usar funciones de Pyspark [27].

Ofrecen flexibilidad, pero deben usarse con moderación, ya que introducen una barrera de serialización entre el núcleo de Spark (JVM) y el intérprete de Python, pudiendo afectar el rendimiento. Por ello en la mayor parte del código se prioriza el uso de funciones nativas de PySpark, reservando los UDFs solo para casos en los que es imprescindible usar bibliotecas externas.

Serialización de Datos

Dentro del contexto de procesamiento distribuido de PySpark, la serialización es un proceso fundamental que consiste en convertir los objetos o estructuras de datos en un formato binario que puede ser transmitido entre los distintos nodos del clúster o almacenado temporalmente en memoria o disco.

En un entorno distribuido, los datos y funciones deben moverse continuamente entre el driver y los executors. Sin embargo, estos entornos no comparten directamente el mismo espacio de memoria; por lo tanto, cada vez que se envía una función, variable o partición desde el driver hacia los ejecutores, Spark necesita serializar esos objetos para su transporte y posteriormente deserializarlos cuando llegan al destino.

La eficiencia del proceso de serialización impacta directamente en el rendimiento global del job, ya que una serialización ineficiente puede incrementar el uso de CPU y memoria, además del tiempo de transferencia de datos entre los nodos.

Por esta razón, se utiliza la siguiente configuración de Spark:

spark.conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")

El KryoSerializer desarrollado en Java, es considerablemente más rápido y compacto que el serializer predeterminado (JavaSerializer), siendo ideal para entornos de alta concurrencia o cuando se manipulan objetos complejos [28].

Para el caso particular de UDFs, la serialización cobra aún mayor relevancia. Cada vez que se aplica una UDF escrita en Python, Spark debe serializar tanto la función como los datos que entran y salen de ella, enviándolos al intérprete Python y devolviendo los resultados a la JVM. Este intercambio constante puede generar una sobrecarga considerable si las UDFs se utilizan masivamente o sobre grandes volúmenes de datos. Por este motivo, siempre se recomienda priorizar el uso de funciones nativas de Spark SQL, que operan internamente sin necesidad de serialización entre lenguajes.

La serialización cumple un rol esencial en la arquitectura distribuida de PySpark, al permitir la comunicación entre procesos y la movilidad de los datos dentro del clúster. No obstante, también representa un punto crítico de optimización: una serialización adecuada puede reducir los tiempos de ejecución y el consumo de recursos, mientras que un manejo ineficiente puede transformarse en un cuello de botella dentro del proceso ETL.

Catalyst Optimizer y AQE

Spark incorpora un componente interno denominado Optimizador Catalyst que analiza el plan lógico de transformaciones, aplica reglas de optimización (como las optimizaciones mencionadas de shuffle y eliminación de pasos redundantes), generando un plan físico eficiente [29].

Complementariamente, la funcionalidad Adaptive Query Execution (AQE) permite ajustar dinámicamente el número de particiones y estrategias de join en tiempo de ejecución, en función de las estadísticas reales observadas durante el procesamiento. Esta configuración se activa durante el desarrollo, pudiendo adaptar los jobs a variaciones de tamaño en función de los períodos procesados.

La combinación de todas las operaciones anteriores permite construir un proceso ETL organizado, eficiente y escalable. Aplicar estas optimizaciones asegura que cada job mensual se ejecute de forma rápida, evitando cuellos de botella en red, disco o memoria.

Flujo de Ejecución de un Job

Cuando se lanza una aplicación de Spark, el Driver Program se inicia y se conecta al Cluster Manager. El Driver es quien orquesta la ejecución y solicita recursos al Cluster Manager, descomponiendo un trabajo grande (Job) en partes manejables (Stages y Tasks) que son distribuidas y ejecutadas por los Executors de los Worker Nodes, todo ello bajo la supervisión del Cluster Manager (Figura 7).

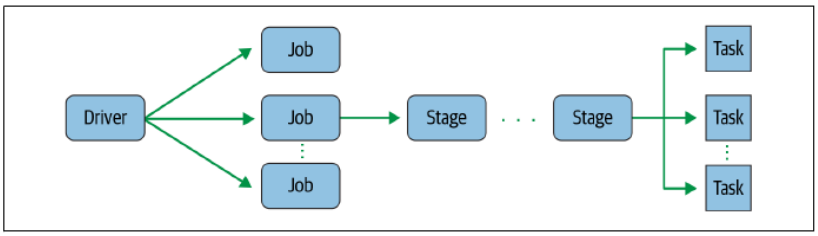


Figura 7: Aplicación, job, stage y task en Apache Spark (Extraído de stack overflow)

La imagen describe como es el flujo de ejecución de un Job en spark, siguiendo la siguiente secuencia [30]:

1. El Driver recibe acciones de la aplicación Spark (como save o collect) y genera Jobs. El job es una unidad de trabajo grande que representa la ejecución de múltiples tareas en paralelo. El driver es el responsable de dividir el trabajo en jobs.
2. Cada Job se divide en conjuntos más pequeños de tareas llamadas Stages. Las Stages dependen unas de otras de manera similar a las etapas de map y reduce en MapReduce. El Driver planifica las Stages de cada Job.
3. Finalmente cada Stage se divide en Tasks, que representa la unidad de trabajo más pequeña a ejecutar. El Driver envía estas Tasks a los Executors para su ejecución en paralelo, lo que permite el procesamiento de datos a gran escala.

Esta arquitectura modular y distribuida es lo que hace a Spark tan potente para el procesamiento de Big Data.

Aplicación de Spark con ABI

La integración de Apache Spark con ABI es una combinación poderosa que permite a las empresas extraer valor de grandes volúmenes de datos de manera eficiente y en poco tiempo.

La principal aplicación de Spark es el procesamiento de datos a gran escala, resultando mucho más rápido que las herramientas tradicionales de BI, gracias a su arquitectura en memoria y procesamiento paralelo. Se adapta a volúmenes de datos crecientes y puede ejecutarse en clusters distribuidos.

Otros casos de uso comunes son para la visualización o analítica avanzada. Desarrollar una solución de visualización es posible a la integración de Spark con herramientas de visualización como Power BI, Tableau o QlikView, permitiendo crear paneles interactivos y visualizar los resultados de los análisis. Por otro lado para Analítica avanzada se puede utilizar Spark SQL, MLlib para aprendizaje automático o GraphX para efectuar análisis relacionados con redes sociales o sistemas de recomendación.

2.2.4. Google Cloud Platform

Google Cloud Platform (GCP) es la plataforma de computación en la nube de Google que ofrece infraestructura global, servicios administrados y herramientas para cómputo, almacenamiento, redes, análisis de datos, Inteligencia Artificial/Machine Learning y más. Está diseñada para permitir a organizaciones mover cargas de trabajo desde infraestructuras on-premise hacia servicios administrados (serverless o gestionados), aprovechando la red global con baja latencia para obtener un rendimiento rápido, coherente y escalable. Facilita el camino hacia la modernización digital para todo tipo de organizaciones, mejorando sus sistemas y procesos actuales sin tener que preocuparse de cómo deben administrar o configurar los servicios requeridos.

GCP agrupa productos en familias (Compute Engine, Storage, Databases, Data Analytics, AI/ML, Networking, Developer Tools, etc.), facilitando la integración entre servicios y poniendo énfasis en seguridad, gestión de identidades (IAM) y cumplimiento.

Esta plataforma usa un modelo de pago por uso, pero ofrece una prueba gratuita para nuevos clientes que incluye $300 USD en créditos para usar durante 90 días, además de un nivel gratuito sobre determinados productos que requieren seguir sus límites mensuales. Para activar la prueba, es necesario una cuenta de Google y una tarjeta de crédito para la verificación de

identidad. No se cobra nada hasta que se superen los límites de la prueba o se active voluntariamente a un plan de pago. Por otro lado, el modelo de pago de uso se basa principalmente en el pago por uso**,** donde el usuario sólo paga por la cantidad de recursos utilizados, abarcando solo los periodos de tiempo donde realmente ese recurso y/o servicio está activo. No se paga ningún servicio de forma mensual, ni cargos por adelantado o cancelación.

2.2.5. Dataproc: Plataforma gestionada para Apache Spark y Hadoop

Dataproc es un servicio totalmente gestionado de Google Cloud para ejecutar clústeres y cargas de trabajo de Apache Spark, Apache Hadoop y otros frameworks de código abierto. Permite crear clústeres rápidamente, ejecutar trabajos Spark/Hadoop y escalar recursos para optimizar costes y rendimiento. Dataproc incorpora aceleradores (p. ej. Lightning Engine para acelerar Spark sobre Compute Engine), además siendo posible integrarlo con otros servicios de metadatos y gobernanza de datos [31].

La esencia de este servicio es la creación rápida y flexible de clústeres con imágenes preconstruidas que incluyen Spark, Hadoop, Hive, Jupyter, etc. Estas imágenes abarcan sistemas operativos como Ubuntu, siendo posible crear una nueva imagen totalmente personalizada abarcando solo los componentes y conectores que se requieran.

Soporta más de 30 frameworks como MapReduce, YARN o Hive y se integra con Dataproc Metastore. También es posible utilizarlo con conectores oficiales como BigQuery, Cloud Storage y conectores externos como paquetes de python o dependencias nativas, que pueden ser instalados desde archivos de inicialización o configurando las propiedades del clúster.

Ofrece un alto grado de seguridad mediante la configuración de una red privada, reglas de firewall, control de acceso por IAM, VPC Service Controls y CMEK para encriptación administrada por cliente.

Ofrece un modelo de pago por uso optimizado, donde los costos se desglosan como: vCPUs de Compute Engine, tarifa de servicio por VCPU-hora, discos persistentes y autoescalado. Soporta clústeres persistentes o clústeres efímeros que se utilizan por menor tiempo.

2.2.6. Google Cloud Storage: Buckets

Un bucket es el contenedor básico de objetos en Google Cloud Storage (GCS). GCS es un servicio de object storage diseñado para durabilidad, escalabilidad y accesible mediante APIs desde servicios como Dataproc, Compute Engine, Dataflow, etc.

Representa una implementación práctica de un Data Lake, ya que actúa como una fuente de repositorio centralizado y persistente de datos crudos (raw data) previo a su procesamiento.

La creación y gestión de un bucket se ve influenciado por varias características que influyen en su rendimiento y costes: clase de almacenamiento, ubicación y control de acceso [32]. En primer lugar, un bucket puede ser Standard, Nearline, Codline o Archive, variando en compromisos y propósitos, coste por acceso y duración mínima recomendada.

Un factor super importante es determinar la ubicación del contenedor, pues esta influye en la latencia, disponibilidad de los datos y coste por transferencia. A estos fines, resulta importante planificar la misma región para todos los servicios utilizados, como bucket y dataproc, pudiendo reducir los costes y la latencia. Por último y no menos importante, este servicio ofrece un alto grado en el control de accesos a nivel de buckets y objetos. Estos controles pueden ser a nivel IAM, acceso uniforme a nivel de bucket y mediante prevención de acceso público. Además, los contenedores son cifrados por defecto por Google, pero también es posible utilizar CMEK u otros servicios de encriptación.

2.2.7. BigQuery: Almacén de Datos Serverless

BigQuery es un almacén de datos completamente administrado que permite almacenar y/o analizar grandes volúmenes de datos mediante el uso de APIs sin aprovisionamiento de infraestructura. BigQuery separa almacenamiento y procesamiento en dos capas independientes, permitiendo escalar cada una de forma diferente, siendo además posible optimizar el almacenamiento por columnas y el procesamiento mediante asignación dinámica de recursos, simplificando las operaciones que conlleva este servicio [33].

La ingesta de la información puede realizarse por lotes o en tiempo real dependiendo de las necesidades, siendo desde Cloud Storage u otros orígenes para ingesta en batch o Streaming inserts y Storage Write API para ingesta en tiempo real. Por otro lado soporta el uso de APIs y consultas SQL, además de conectores ODBC/JDBC para integrarse con distintas aplicaciones.

BigQuery ofrece optimización para Inteligencia de Negocios mediante vistas materializadas y BI Engine (caché en memoria) para acelerar consultas de dashboards. Se integra nativamente con herramientas BI, destacando Google Looker Studio.

Su modelo de precios es bajo demanda (on-demand) que depende de los bytes procesados por consulta y GB de almacenamiento. Sin embargo, BigQuery como servicio ofrece un plan gratuito mensual que cubre 10 GB de almacenamiento activo y 1 TB de procesamiento de consultas.

2.2.8. Looker Studio: Visualización y Dashboards

Looker Studio es la herramienta de visualización y reporting gratuito de Google que permite crear informes y dashboards interactivos mediante un editor drag-and-drop. Soporta múltiples conectores (BigQuery, Cloud Storage, Google Sheets, bases de datos, conectores comunitarios) y opciones para compartir/colaborar en tiempo real. Looker Studio se integra estrechamente con BigQuery y puede acelerarse con BI Engine [34].

Looker Studio se conecta directamente a BigQuery accediendo a las tablas y vistas requeridas. Para dashboards con requisitos de baja latencia y uso intensivo, conviene usar BI Engine (caché de consultas en memoria) o pre-agregar datos en BigQuery. También soporta visualización de tipos geoespaciales.

A fines del trabajo de investigación se crea un panel de control interactivo y dinámico para visualizar los resultados de los indicadores KPIs propuestos, ayudando a obtener información valiosa y por ende facilitar la toma de decisiones.

2.3. Transporte Público de Pasajeros de Corta Distancia de la provincia de San Juan

La provincia de San Juan comenzó a utilizar el Sistema Único de Boleto Electrónico (SUBE) en 2015, y en 2016 su uso fue obligatorio, lo que permitió registrar de manera sistemática los viajes de los usuarios del transporte público de corta distancia. Actualmente, aunque existen otros medios de pago, los datos de SUBE continúan siendo una fuente significativa para el análisis del flujo de pasajeros. Este sistema brinda beneficios como la agilidad en el pago, descuentos automáticos y aplicación de tarifas sociales, y, desde el punto de vista investigativo,

ofrece la oportunidad de extraer datos que permitan construir indicadores y realizar análisis orientados a mejorar políticas públicas de transporte [5]. Por ello, este trabajo se centra en el análisis de los datos de la tarjeta SUBE como base para la demostración de metodologías de analítica y Big Data.

Capítulo 3: Solución ABI para la STyT

Para el desarrollo de la solución ABI se seleccionaron Apache Spark como motor de procesamiento y Google Cloud Platform como infraestructura en la nube. La elección se fundamentó en las capacidades esenciales de una plataforma ABI, que incluyen integración de datos, transformación, análisis avanzado, visualización y generación de indicadores [35]. Apache Spark permitió procesar los datos de transporte público de manera distribuida, realizar análisis avanzados y generar dashboards interactivos, mientras que GCP proporcionó la infraestructura necesaria para manejar grandes volúmenes de información de forma escalable y eficiente.

3.1. Arquitectura Elegida

Se propuso una arquitectura ABI secuencial, en la cual los datos transaccionales SUBE de los años 2023 y 2024 (hasta marzo) se almacenan inicialmente en formato CSV en un bucket de Cloud Storage. Mediante Dataproc, se creó un clúster efímero de Spark para ejecutar múltiples jobs ETL, transformando la información y cargándola en BigQuery, que funcionó como almacén de datos centralizado. Posteriormente, los datos se visualizaron a través de Looker Studio en dashboards interactivos. La Figura 8 la describe:

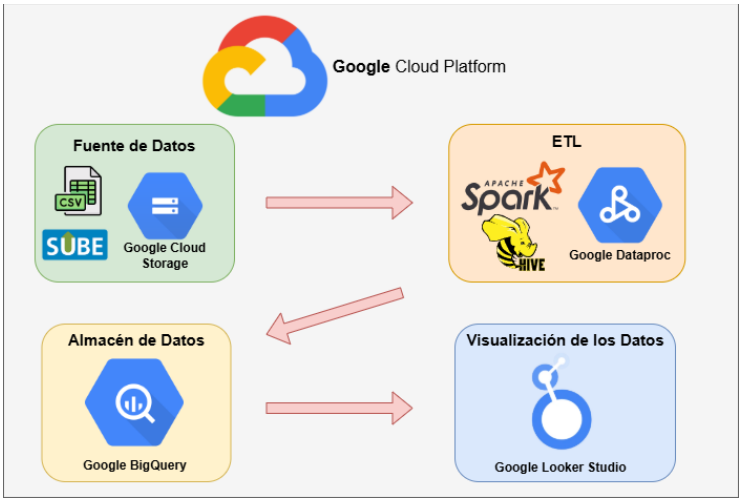


Figura 8: Arquitectura ABI utilizada usando el ecosistema de Google Cloud con Spark

3.1.1. Google Cloud Storage: Almacenamiento de los Datos Fuente

Dentro de la arquitectura, el almacenamiento de los datos de entrada y de los scripts de procesamiento se resolvieron mediante la creación de un bucket en Google Cloud Storage (GCS). Este bucket constituyó el punto inicial del flujo ETL y actúa como repositorio central para los archivos requeridos por el clúster de procesamiento en Dataproc.

La elección de GCS responde a su integración nativa con los servicios utilizados en el entorno GCP, lo que evita transferencias innecesarias entre plataformas y simplifica la administración operativa. Asimismo, permite definir clases de almacenamiento acordes a la frecuencia de acceso prevista, favoreciendo la optimización de costos sin comprometer la durabilidad de la información.

La organización interna del bucket se estructuró mediante prefijos que simulan una jerarquía de directorios, diferenciando claramente los scripts de procesamiento de los datos de entrada. La estructura adoptada se presenta a continuación:

gs://bucket-sube/

├── scripts/
│ ├── Main.py (script principal)
│ ├── CreateExtraction.py
│ ├── CreateGPS.py
│ └── Transformation.py (Devuelve join de t1 y t2 con datos válidos)
│ ├── GetCities.py
│ ├── GestionCalendarios.py
│ ├── IPC.py
├── inputs/
│ ├── Extracción/ (csv)
│ ├── GPS/ (csv)
│ ├── IPC Indec (csv)
│ ├── Coordenadas-ciudades-departamentos (csv)

El directorio inputs concentra las fuentes de información desde donde se inicia el proceso ETL. Los datos transaccionales del pasaje y los registros de geolocalización se almacenan en subdirectorios independientes, lo que permite tratarlos de manera diferenciada durante la etapa de transformación. Ambos conjuntos se organizan en archivos particionados por período mes-año, estrategia que facilita el reprocesamiento ante fallos y mejora la modularidad del pipeline.

Las Figuras 9 y 10 ilustran la visualización de los scripts y de los archivos de entrada dentro del bucket.

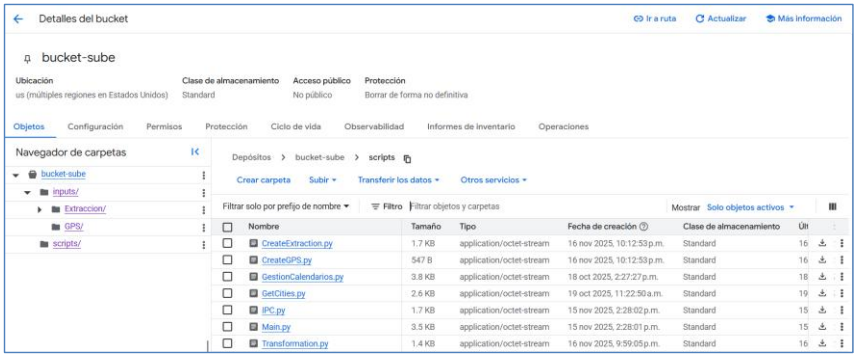


Figura 9: Visualización demostrativa de los scripts del Bucket "bucket-sube"

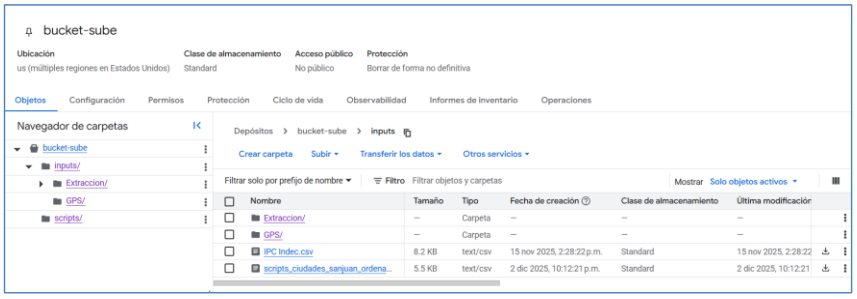


Figura 10: Visualización demostrativa de los archivos entrantes del bucket.

En cuanto a la configuración regional, el bucket fue creado en una región de Estados Unidos, manteniendo coherencia con el resto de los servicios utilizados, con el objetivo de reducir latencia y asegurar compatibilidad operativa.

Finalmente, el acceso al bucket se restringió mediante políticas IAM, manteniéndolo como un contenedor privado dentro del proyecto y garantizando el control sobre los permisos de lectura y escritura durante el desarrollo de la solución.

3.1.2. Dataproc con Spark: Procesamiento de los Datos

El procesamiento de los datos se implementó mediante un clúster administrado en Google Cloud Dataproc, utilizando Apache Spark como motor de ejecución. Se optó por usar un clúster temporal (creado exclusivamente para cada ciclo de procesamiento y destruido posteriormente) para la ejecución de los jobs correspondientes a cada período mes-año y eliminado una vez finalizada la carga. Esta decisión permitió reducir costos operativos, dado que la demanda del proceso no justificaba un clúster permanente ni la configuración de políticas de autoescalado.

El proceso ETL fue desarrollado en PySpark, permitiendo estructurar la lógica de transformación mediante código Python. Para cada período mensual se ejecutaron múltiples jobs independientes, favoreciendo la modularidad del pipeline y facilitando el reprocesamiento ante eventuales fallos. La información resultante se cargó posteriormente en el almacén de datos.

La estrategia adoptada estuvo condicionada por el uso de la prueba gratuita de GCP, que impone límites de recursos (250 GB de almacenamiento en disco, 24 GB de RAM y 6 vCPU). En este contexto, el código fue ajustado para aprovechar de manera eficiente la capacidad disponible del clúster.

Para la creación del clúster se consideran los siguientes aspectos técnicos y de configuración:

- **Región**: us-central1, con zona de preferencia “a” pero puede ser cualquiera. En todos los servicios se utiliza la misma región EEUU para reducir la latencia (BigQuery y GCS).
- **Tipo de Clúster**: Estándar con un nodo máster y dos nodos workers (trabajadores).
- **Control de versiones**: La última versión a la fecha, 2.3-ubuntu 22 (con Hadoop 3.3, Spark 3.5 y conector BigQuery-Spark integrados).
- **Mejoras en el rendimiento de Spark**: Para optimizar la forma en que Spark ejecuta las cargas de trabajo, se aplican las tres configuraciones:
  - **Optimizaciones avanzadas:** Mejora el rendimiento de las cargas de trabajo que involucren dataframes y consultas complejas SQL.

    - **Capa de ejecución avanzada:** Aumento significativo de velocidad para la ejecución de operaciones intensivas, como agregaciones y shuffles.
    - **Almacenamiento en caché:** Los datos leídos de GCS se pueden almacenar localmente en los nodos del clúster, lo que reduce la latencia y mejora la velocidad de ejecución de cargas de trabajo.
- **Configuración de Red**: Red principal default. Subred default (10.128.0.0/20). En la sección de apéndice se detalla la necesidad de definir reglas de firewall entrantes y salientes para el manejo seguro de los datos.
- **Configuración de los Nodos**: La limitación de los recursos del clúster debe distribuirse adecuadamente entre todos sus nodos.
    - **Nodo Master**: Máquina n2-standard-2; SSD Persistent Disk; Interfaz de SSD Local NVMe; 1 SSD local de 375 GB y 80 GB disco principal.
    - **Nodos Worker (2)**: Máquina n2-standard-2; SSD Persistent Disk; Interfaz de SSD Local NVMe; 1 SSD local de 375 GB y 60 GB disco principal.
- **Propiedades del Clúster**: Se utiliza la biblioteca de Python workalendar para el manejo de calendarios nacionales. Dado que el entorno de Python en Dataproc se basa en Conda, la dependencia se instala mediante pip utilizando la propiedad dataproc:pip.packages, con el valor workalendar==17.0.0, siendo necesario indicar la versión requerida.
- **Instancias con direcciones IP internas:** Se gestiona la opción de que las máquinas virtuales del clúster tengan acceso desde internet, permitiendo habilitar o deshabilitar direcciones IP privadas dentro de la red de nube privada virtual (VPC). Para este caso, se deshabilita esta opción pues se busca acceder a internet para actualizar repositorios de la imagen Ubuntu e instalar la dependencia de python.
- **Acceso al proyecto**: Se habilita el alcance de la plataforma cloud para el clúster.

La creación y el monitoreo del cluster se llevó a cabo desde la sección de clústeres Dataproc en la Consola de Google Cloud. En él, se visualizan gráficos de monitoreo sobre el rendimiento del clúster y YARN en general (Figuras 11 y 12).

Figura 11: Monitoreo del trabajo ejecutado en Dataproc

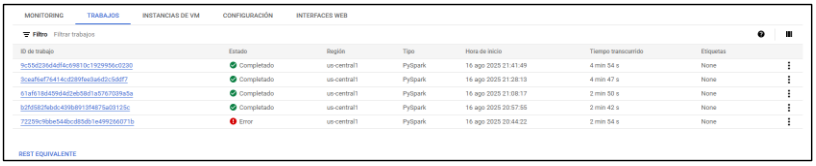


Figura 12: Lista de trabajos ejecutados desde el cluster Dataproc.

Las pantallas de arriba presentan información que hicieron posible el seguimiento del clúster, a través de datos relativos a la memoria YARN en uso, a la memoria YARN pendiente requerida, al uso de CPU y disco. Este monitoreo permitió identificar posibles cuellos de botella y evaluar, en caso necesario, la ampliación de recursos o nodos de procesamiento.

3.1.3. Google Cloud SDK: Gestión Operativa del Clúster

Para la interacción operativa con el entorno GCP se utilizó la terminal de Google Cloud SDK, desde la cual se gestionó la autenticación, la vinculación del proyecto y la ejecución de los jobs sobre el clúster Dataproc. Esta herramienta permitió centralizar las acciones necesarias para el despliegue y monitoreo del procesamiento sin depender exclusivamente de la interfaz gráfica de la consola web.

En el contexto de este trabajo, la terminal fue empleada principalmente para lanzar los jobs de PySpark y consultar los logs de ejecución en tiempo real. A continuación, se describen los comandos utilizados durante el desarrollo:

- **Vincular cuenta personal**: Con el comando *gcloud auth login* se abre una ventana emergente para la autenticación de la cuenta de google cloud.
- **Vincular proyecto**: Con el comando *gcloud config set project [ID_DE_PROYECTO]* el proyecto y la cuenta de servicio ya quedan autenticadas (Figura 13).

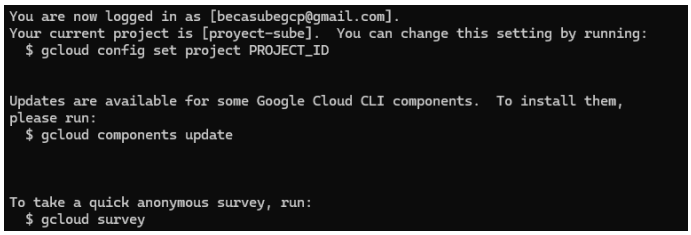


Figura 13: Vinculación del proyecto desde la terminal de Google Cloud SDK

- **Verificar acceso al bucket**: Mediante el comando gsutil ls gs://bucket-sube se verifica que la terminal tenga acceso al bucket. Para operar sobre objetos de Cloud Storage, es necesario que la cuenta de servicio cuente con los roles y permisos correspondientes.
- **Lanzamiento de un job**: Mediante el comando gcloud dataproc jobs submit pyspark [PARÁMETROS] se inicia la ejecución de un job de tipo PySpark sobre el clúster previamente creado y activo. Los parámetros en concreto se mencionan más adelante, en la sección 3.2.4 “Implementación del ETL”.

3.1.4. BigQuery: Persistencia del Almacén de Datos

Una vez finalizado el proceso ETL en Dataproc, la información resultante fue almacenada en BigQuery, utilizado como almacén de datos dentro de la arquitectura implementada. Para este trabajo se empleó el plan gratuito del servicio, considerando las limitaciones asociadas al volumen de almacenamiento y procesamiento disponibles en dicho esquema [36].

La carga de datos se realizó mediante el conector Spark–BigQuery integrado en el clúster Dataproc, permitiendo escribir directamente las tablas generadas por los jobs de PySpark. A partir de allí, los datos quedaron disponibles para su consulta mediante SQL estándar y para su posterior conexión con Looker Studio en la etapa de visualización.

Cada consulta SQL tiene asociado un tamaño de procesamiento y almacenamiento, esto debido a que BigQuery desencadena un proceso de segmentación de la consulta en sus nodos internos

de procesamiento. Este proceso divide la consulta en tareas más pequeñas en base a las columnas y tablas involucradas, donde cada tarea se ejecuta en paralelo y luego se combinan para el resultado final, resultando en un procedimiento muy similar a MapReduce.

El programa principal "main" fue el responsable de materializar la tabla hecho_viajes (mostrada en el Figura 14). Ese programa define parámetros necesarios como: proyecto creado, dataset o conjunto de datos, tabla resultante y el bucket utilizado para almacenar temporalmente los datos procesados de Dataproc que serán escritos en BigQuery, optimizando costos y tiempos de transferencia de información.

Es importante tener en cuenta que el conjunto de datos creado manualmente se alojó en la misma región tanto para el bucket, cluster dataproc como BigQuery, evitando mayores costos por latencia.

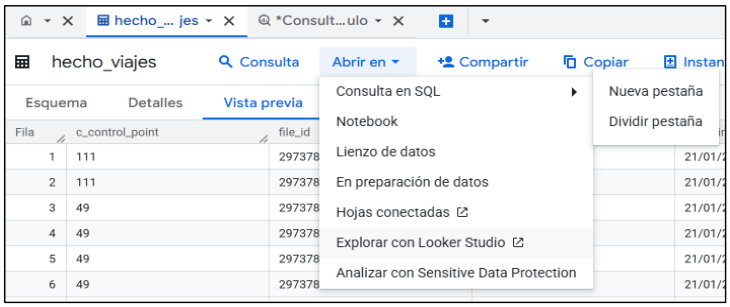


Figura 14: Vista previa de la tabla resultante "hecho viajes" desde BigQuery

3.1.5. Looker Studio: Visualización

La etapa de visualización se implementó utilizando Looker Studio como herramienta para la construcción de paneles interactivos conectados a BigQuery. A partir de las tablas generadas en el almacén de datos, se diseñaron informes que permitieron explorar indicadores definidos.

Looker Studio fue seleccionado por su integración directa con BigQuery y por tratarse de una herramienta accesible dentro del entorno de trabajo adoptado. Si bien el servicio no impone restricciones propias significativas para la elaboración de informes, las consultas ejecutadas desde los paneles generan procesamiento en BigQuery. En consecuencia, cada actualización de los dashboards implica la ejecución de consultas SQL sobre el almacén de datos.

En el marco de esta tesis, se desarrolló un panel de control compuesto por distintos dashboards orientados a la visualización de indicadores clave (KPIs). Estos tableros permiten aplicar filtros

dinámicos para analizar la información desde múltiples perspectivas y facilitar la interpretación de los resultados obtenidos. Un ejemplo del panel desarrollado se presenta en la Figura 15.

Figura 15: Visualización de una página del panel de control de Looker Studio

Cabe mencionar que además de visualizar, de manera directa, datos de columnas de tablas en BigQuery, se definieron y usaron datos calculados dentro de Looker Studio para ciertos indicadores. Esta estrategia permitió realizar transformaciones simples en la capa de visualización, evitando incrementar innecesariamente el procesamiento sobre el almacén de datos.

3.2. Proceso de Desarrollo

3.2.1. Análisis de los Datos Fuente

Los datos recibidos de la STyT se organizaron en dos archivos CSV por cada período mes-año, abarcando desde enero de 2023 hasta marzo de 2024.

El primero, denominado “extracción”, contiene datos asociados a cada transacción (viaje) de transporte público: empresa, línea, identificación de tarjeta, saldo, tipo de transacción, fecha y hora, contrato, monto y descuento aplicado.

El segundo archivo, denominado “gps”, contiene datos de geolocalización del sistema de transporte, entre los cuales se encuentran los pares de coordenadas latitud-longitud que indican

la ubicación espacial de las unidades (colectivos). Algunos de estos registros se encuentran asociados al momento de validación de la tarjeta por parte del usuario (subida a la unidad), mientras que otros corresponden a puntos intermedios generados durante el seguimiento en tiempo real de los recorridos. En consecuencia, el volumen de registros del archivo "gps" es considerablemente superior al del archivo de extracción.

Previo a la ejecución del proceso ETL y a la carga en el almacén de datos, se llevó a cabo un análisis detallado los datos de ambos archivos, con el fin de comprender su estructura, validar la calidad de la información e identificar posibles inconsistencias o ambigüedades.

3.2.2. Diseño del Almacén de Datos (Datawarehouse)

A partir del análisis de las fuentes disponibles, se procedió al diseño del almacén de datos bajo un enfoque bottom-up, dado que la estructura del modelo y la definición de los indicadores surgieron del análisis exploratorio de los datos efectivamente accesibles y no de un conjunto formal de requerimientos definidos por los usuarios finales. En función de dicha exploración, se identificaron métricas potencialmente relevantes para la organización. Asimismo, el alcance de la solución estuvo condicionado por el acceso efectivo a las fuentes disponibles, lo que limitó la incorporación de otras variables que podrían haber ampliado el análisis.

3.2.2.1. Diseño Conceptual

El diseño conceptual del almacén de datos se definió en torno al hecho de negocio "viaje", cuya granularidad corresponde a cada transacción individual registrada en el sistema de transporte público. Entre las métricas asociadas a cada viaje se encuentran el monto abonado y el descuento aplicado.

La caracterización completa del hecho requirió integrar conceptualmente la información proveniente de los dos archivos disponibles ("extracción" y "gps"), ya que ambos aportan información complementaria sobre el evento de validación desde perspectivas diferentes. Mientras el archivo de extracción proporciona los datos operativos de la transacción, el archivo "gps" incorpora la componente de geolocalización, enriqueciendo el análisis del viaje.

A partir de la identificación del hecho principal, se definieron las dimensiones necesarias para su contextualización: tiempo, línea, empresa, tipo de transacción y ubicación geográfica. Estas dimensiones constituyen los ejes de análisis a través de los cuales pueden examinarse las métricas vinculadas al hecho de viaje.

La Figura 18 presenta el Diagrama/Modelo Conceptual Dimensional correspondiente:

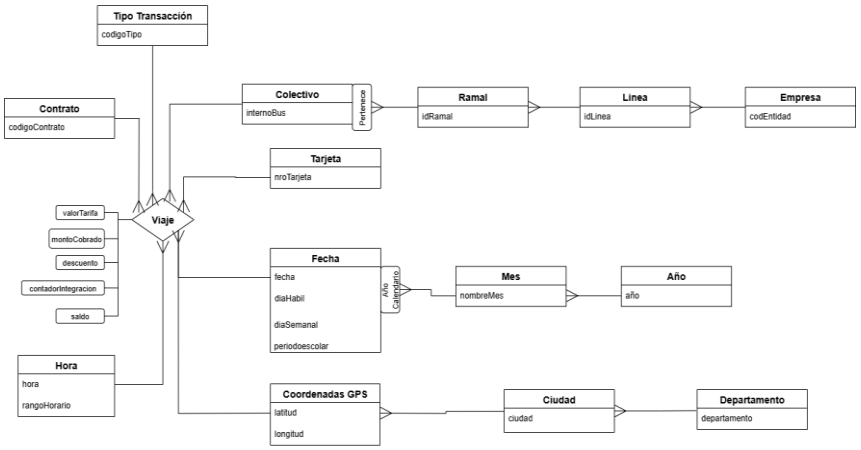


Figura 18: Modelo Dimensional del Almacen de Datos (Simbología Vaisman & Zimányi, 2014)

A continuación, se describen los principales componentes del Modelo Conceptual Dimensional desarrollado:

- El hecho VIAJE contiene las medidas asociadas a cada transacción registrada en el sistema: valor de la tarifa, monto cobrado, descuento aplicado, saldo y un identificador de combinación para el conteo de trasbordos. Las medidas referidas a importes, tales como tarifa, monto, descuento y saldo, necesitaron un proceso extra, se aplicó un algoritmo en Spark que permitió ajustarlos según los Índices de Precios al Consumidor (IPC) del INDEC Argentina, obteniendo valores conceptualmente comparables.

  La transacción posee un tipo que identifica si corresponde a boleto común o a trasbordo, lo cual influye en el monto de la tarifa. El tipo de contrato es otro aspecto que influye en el monto, puede ser común, pensión, beneficiario, beca estudiantil, entre otros.

- En cuanto a las dimensiones, se identificaron varias. Algunas contienen atributos descriptivos, mientras que otras se limitan a un único atributo identificador, conocidas en la bibliografía como dimensiones degeneradas.

  Las dimensiones identificadas son: COLECTIVO, RAMAL, LÍNEA, EMPRESA, FECHA (se extrae del campo fecha/hora desde los datos fuente, y se modela como una dimensión separada para facilitar el análisis por fecha y hora de manera independiente),

MES, AÑO, COORDENADAS, CIUDAD, DEPARTAMENTO, HORA, TARJETA, CONTRATO, TIPO TRANSACCION.

Algunas dimensiones están vinculadas directamente con el hecho central, mientras que otras se relacionan con dimensiones existentes, generando jerarquías (comúnmente vinculadas con una relación n-1). Esas jerarquías permiten analizar los hechos a diferente nivel de detalle, según se "camine" entre los niveles dentro de una jerarquía.

Se identificaron las siguientes:

- COLECTIVO, RAMAL, LÍNEA y EMPRESA.
- FECHA, MES, AÑO.
- COORDENADAS, CIUDAD, DEPARTAMENTO.

Se observan que todas las jerarquías identificadas son estrictas (la multiplicidad de las vinculaciones entre ellas son n-1), y simétricas (todos los niveles de la jerarquía son obligatorios).

3.2.2.2. Diseño Lógico

A partir del diseño conceptual realizado, se definió el diseño lógico e implementación del almacén de datos según el paradigma relacional, generando así, tablas relacionales.

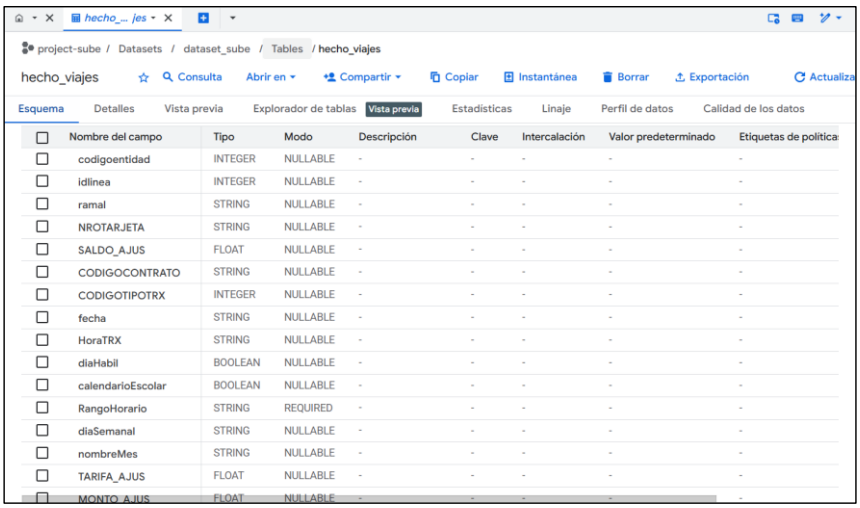
En principio, se optó por un esquema en estrella frente al de copo de nieve, ya que este último, al normalizar las dimensiones en múltiples tablas, complejiza las consultas y aumenta los tiempos de respuesta, aunque se eliminan redundancias. Por el contrario, el esquema en estrella sacrifica cierto nivel de normalización (aunque en este caso no sucede), en pos de ofrecer una estructura más simple y eficiente para las consultas, aspecto fundamental en el manejo de grandes volúmenes de datos.

A posteriori, se tomó otra decisión estratégica de implementación, consolidar todos los datos una única tabla, centralizando todos los datos, con el propósito de simplificar su explotación analítica y conservar la semántica dimensional definida en el diseño conceptual. La decisión pudo realizarse dado que la estructura del modelo conceptual es sencilla, sus jerarquías son estrictas y simétricas [15].

Al no tener las tablas de dimensión separadas se optimiza significativamente el rendimiento de las consultas, conformando una implementación escalable, con un excelente desempeño y

claridad analítica, aprovechando al máximo las capacidades de BigQuery para el manejo de grandes volúmenes de información y la generación de reportes en Looker Studio.

La estructura de la tabla se expone en la Figura 19:



Nombre del campo	Tipo	Modo	Descripción	Clave	Intercalación	Valor predeterminado	Etiquetas de política
codigoentidad	INTEGER	NULLABLE	-	-	-	-	-
idlinea	INTEGER	NULLABLE	-	-	-	-	-
ramal	STRING	NULLABLE	-	-	-	-	-
NROTARJETA	STRING	NULLABLE	-	-	-	-	-
SALDO_AJUS	FLOAT	NULLABLE	-	-	-	-	-
CODIGOCONTRATO	STRING	NULLABLE	-	-	-	-	-
CODIGOTIPOTRX	INTEGER	NULLABLE	-	-	-	-	-
fecha	STRING	NULLABLE	-	-	-	-	-
HoraTRX	STRING	NULLABLE	-	-	-	-	-
diaHabil	BOOLEAN	NULLABLE	-	-	-	-	-
calendarioEscolar	BOOLEAN	NULLABLE	-	-	-	-	-
RangoHorario	STRING	REQUIRED	-	-	-	-	-
diaSemanal	STRING	NULLABLE	-	-	-	-	-
nombreMes	STRING	NULLABLE	-	-	-	-	-
TARIFA_AJUS	FLOAT	NULLABLE	-	-	-	-	-
MONTO_AJUS	FLOAT	NULLABLE	-	-	-	-	-

Figura 19: Esquema de la tabla resultante en BigQuery

Abajo, se muestra la vista previa de la tabla resultante (Figura 20).

Fila	codigoentidad	idlinea	interno	NROTARJETA	SALDO	SALDO_AJUS	CODIGOCON...	CODIGOTIPO...	fecha	HoraTRX	diaHabil	calenda...
1	525	3347	92	1837627046	69.6	433.48	621	12	12/01/2023	07:20:17	true	false
2	525	3347	92	1825013402	1141.65	7110.42	621	12	12/01/2023	07:20:11	true	false
3	525	3347	92	1819441479	289.55	1803.37	621	12	12/01/2023	07:20:03	true	false
4	525	3347	92	1808439691	429.09	2672.46	621	12	12/01/2023	07:19:58	true	false
5	251	3397	46	1811240279	372.14	2317.76	621	12	12/01/2023	09:37:30	true	false
6	251	3397	46	1813082998	327.89	2042.16	621	12	12/01/2023	09:36:48	true	false
7	251	3397	46	1844158495	166.6	1037.62	621	12	12/01/2023	09:36:45	true	false
8	254	3436	64	1816310246	632.62	3940.08	621	12	09/01/2023	18:20:47	true	false
9	254	3436	64	1839028424	-27.93	-173.95	602	12	09/01/2023	18:09:13	true	false
10	254	3436	64	1824557558	460.84	2870.2	602	12	09/01/2023	18:09:07	true	false
11	254	3436	64	1846007360	207.52	1292.48	621	12	09/01/2023	18:08:56	true	false
12	254	3436	64	1846007360	258.6	1610.61	621	12	09/01/2023	18:08:51	true	false
13	254	3436	64	1830630528	654.54	4076.6	621	12	09/01/2023	18:03:20	true	false
14	254	3436	64	1830630528	722.04	4497.01	621	12	09/01/2023	18:03:16	true	false
15	251	3397	46	1846234918	661.4	4119.33	621	12	12/01/2023	10:14:05	true	false
16	251	3397	46	1809038902	213.38	1328.97	621	12	12/01/2023	10:14:00	true	false
17	251	3397	46	1809038902	258.61	1610.67	621	12	12/01/2023	10:13:56	true	false
18	251	3397	46	1813082451	5.86	36.5	621	12	12/01/2023	10:13:51	true	false
19	251	3397	46	1842095039	181.27	1128.99	621	12	12/01/2023	10:13:46	true	false
20	251	3397	46	1824222394	615.09	3830.9	621	12	12/01/2023	10:13:39	true	false
21	251	3397	46	1808885660	671.22	4180.49	602	12	12/01/2023	09:41:40	true	false

Figura 20: Vista previa de la tabla resultante en BigQuery.

3.2.3. Definición de los KPIs

Dentro del contexto de la STyT, los KPIs inherentes debieran permitir entender el comportamiento de la demanda, los patrones de movilidad, la eficiencia de los servicios y el impacto de políticas públicas como tarifas diferenciadas o subsidios.

Los KPIs identificados en este proyecto, permiten responder preguntas clave como:

- ¿Cuándo y cómo se viaja más?
- ¿Qué ciudades, líneas o empresas concentran la mayor demanda?
- ¿Cómo inciden factores contextuales como los días hábiles, los periodos escolares o los feriados?
- ¿Cuál es el impacto de los trasbordos en el sistema?
- ¿Cómo han ido evolucionando las tarifas y saldos promedio con el tiempo?

A continuación, se mencionan los KPIs identificados:

Categoría	Nombre	Descripción	Justificación
Indicadores de Viajes	**Cantidad de Viajes por Periodo**	Cuadro de resultado que muestra la cantidad de viajes realizados en un periodo de tiempo determinado.	Conocer la totalidad de viajes en un rango temporal en específico.
	Cantidad de Viajes por Rango Horario	Agrupar los viajes según el rango horario (mañana, tarde o noche).	Permite identificar los picos de demanda diarios, útil para conocer la frecuencia de colectivos.
	Cantidad de Viajes por Día Semanal	Organizar la cantidad de viajes por día de la semana.	Identifica los picos de demanda semanal, resultando muy útil en conjunto con el indicador anterior.
	Cantidad de Viajes por Tipo de Transacción	Conteo de los viajes por forma de viaje (boleto común o trasbordo).	Permite analizar cómo se distribuyen los dos tipos de viaje.
	Cantidad de Viajes por Día Hábil	Agrupa la cantidad de viajes por día hábil y no hábil (feriados, fin de semana).	Permite cuantificar la diferencia entre viajes realizados en días laborables y no laborables, esencial para entender el comportamiento de

			la movilidad frente al calendario.
	Cantidad de Viajes por Periodo Escolar	Agrupa la cantidad de viajes en función del periodo escolar, según el calendario provincial.	Identifica la incidencia del calendario escolar en el sistema de transporte, evaluando cuántos viajes dependen de la actividad educativa.
Indicadores de Trasbordos	**Tasa de Trasbordos**	Proporción de viajes con trasbordos respecto del total de viajes. Se calcula como: $\frac{\textbf{Viajes con contadorIntegracion} > \textbf{0}}{\textbf{Total de Viajes}}$	Es un indicador de eficiencia: una tasa elevada muestra dependencia en combinaciones, lo que puede reflejar carencias en rutas directas. Podría indicar la necesidad de generar nuevos subsidios por trasbordo.
	Cantidad de Trasbordos por Rango Horario	Conteo de trasbordos agrupados por rango horario.	Permite analizar en qué rangos horarios los usuarios deciden tomar más trasbordos, lo que señala posibles áreas para mejorar la conectividad directa.
	Cantidad de Trasbordos por Día Semanal	Agrupa el total de trasbordos realizados según el día de la semana en el que se efectuó la transacción.	Permite identificar los días de mayor demanda de trasbordos. Es crucial para la planificación operativa, ya que los días con más trasbordos pueden requerir una mejor coordinación de rutas y frecuencias entre líneas.
	Cantidad de Trasbordos por Departamento	Organizar la cantidad de trasbordos por departamento.	Analizar qué departamentos tienen mayor y menor demanda de trasbordos.
	Cantidad de Trasbordos por Ciudad	Agrupar la cantidad de trasbordos por ciudad.	Identificar en qué ciudades los usuarios toman más trasbordos, lo que posibilita analizar mejoras en recorridos.
	Cantidad de	Muestra la proporción de	Ayuda a medir el impacto de la

	Trasbordos por Periodo Escolar	trasbordos que se realizan dentro y fuera del período escolar.	población estudiantil en la demanda de trasbordos. Esta información es vital para realizar ajustes durante los ciclos académicos.
Indicadores de Geolocalización	**Monto Total Recaudado por Departamento y Ciudad**	Calcula la suma total del dinero recaudado desglosado por departamento y ciudad de los viajes.	Identifica las zonas geográficas que generan el mayor volumen de ingresos, resultando interesante para la toma de decisiones financieras.
	Monto Total Descontado por Departamento y Ciudad	Calcula la suma total de los descuentos aplicados, desglosado por departamento y ciudad.	Permite visualizar la distribución geográfica de los beneficios y subsidios aplicados. Importante para evaluar el impacto social y económico de las políticas tarifarias.
	Cantidad de Viajes por Ciudad	Conteo de viajes agrupados por las ciudades de la provincia, obtenidas a partir de las coordenadas GPS.	Ofrece una visión territorial sobre qué ciudades concentran la mayor movilidad. Esencial para analizar la distribución geográfica de la demanda y priorizar inversiones por ej. en infraestructura y subsidios.
	Cantidad de Viajes por Departamento	Agrupa la cantidad de viajes por departamento de la provincia.	Conocer qué departamentos regulan la mayor demanda de circulación.
	Cantidad de Contratos por Departamento	Muestra el número de contratos registrados, agrupados por el departamento donde se efectúa la transacción.	Ayuda a entender la distribución de los contratos por zona geográfica. Es clave para estrategias de fidelización o expansión de contratos.
	Saldo Promedio por Periodo	Cálculo del saldo promedio de los usuarios en un periodo de tiempo determinado.	Información valiosa para conocer el capital promedio por periodo de tiempo, lo que puede conllevar a cambios en la administración financiera del servicio de transporte.

Indicadores de Perfil de Usuario	**Saldo Promedio por Tipo de Contrato**	Se calcula el saldo promedio de los usuarios agrupados por tipo de contrato.	Permite evaluar el comportamiento financiero de los usuarios: algunos grupos pueden mantener saldos altos como respaldo de seguridad, otros pueden recargar con más frecuencia. Esta información puede orientar estrategias de recarga y promociones.
	Saldo Promedio en el tiempo	Calcula el saldo promedio de las tarjetas de los usuarios a lo largo de un período de tiempo determinado.	Permite analizar la tendencia del saldo de los usuarios e inferir el comportamiento de recarga. Los picos o caídas pueden correlacionarse con eventos específicos, promociones o momentos de mayor uso.
	Saldo Promedio por Día Semanal	Calcula el saldo promedio de las tarjetas de los usuarios agrupado por el día de la semana.	Ayuda a identificar patrones de recarga o uso intensivo por día. Por ejemplo, si el saldo promedio es mayor los lunes, podría indicar recargas de inicio de semana.
	Saldo Promedio por Departamento y Ciudad	Saldo promedio de los usuarios agrupado por departamento y ciudad.	Muestra diferencias territoriales en el uso de la tarjeta SUBE. Una ciudad con saldos bajos podría reflejar un mayor grado de vulnerabilidad económica. Indicador que podría ayudar al ofrecimiento de beneficios en aquellas ciudades con saldos bajos.
	Saldo Promedio por Entidad	Muestra el saldo promedio de las tarjetas, desglosado por Empresas que gestionan el servicio de transporte.	Facilita el análisis del comportamiento de recarga/uso entre las diferentes empresas de transporte. Es útil para la evaluación de la gestión por entidad.
	Cantidad de Viajes por	Agrupa la cantidad de viajes por ramal y línea.	Provee un nivel de detalle operativo para evaluar el rendimiento de cada

Indicadores de Entidades	**Línea y Ramal**		ramal específico dentro de una línea, útil para analizar la demanda y el rendimiento de las líneas.
	Cantidad de Trasbordos por Línea y Ramal	Agrupa el total de trasbordos realizados, desglosado por la Línea y el Ramal específico de la ruta de transporte.	Identifica las líneas y tramos más utilizados para transbordar. Es esencial para optimizar la interconexión de las rutas, asegurar la fluidez de los trasbordos y mejorar la calidad del servicio en esos puntos clave.
	Cantidad de Viajes por Entidad	Contar la cantidad de viajes realizados por código de empresa.	Facilita monitorear qué empresas poseen mayor demanda de viajes y analizar la posibilidad de distribuir subsidios o beneficios.
	Monto Total Recaudado y Descontado por Entidad	Muestra la suma total del dinero recaudado y el total de descuentos aplicados, agrupados por la empresa de transporte.	Permite una visión completa de los ingresos brutos y netos por cada empresa. Fundamental para evaluar el rendimiento económico de las entidades y el impacto de los descuentos.
Indicadores de Montos	**Monto Total Recaudado por Periodo**	Presenta la cifra total y consolidada del dinero recaudado en un período de tiempo seleccionado.	Es un indicador de resumen de alto nivel para evaluar el desempeño económico general. Proporciona un vistazo rápido al volumen total de ingresos.
	Monto Total Recaudado a través del tiempo	Agrupar los montos totales recaudados y descontados por periodo mes-año, abarcando 2023 hasta marzo 2024.	Permite seguir la evolución mensual de la recaudación y descuentos a través del tiempo.
	Monto Total Recaudado y Descontado por Contrato	Muestra el total del dinero recaudado y el total de descuentos aplicados, desglosados por el Tipo de Contrato.	Permite analizar la rentabilidad y el impacto de los descuentos de cada tipo de contrato. Crucial para la gestión de tarifas y la evaluación de subsidios o beneficios por contrato.

	Monto Total Recaudado por Rango Horario	Calcula el monto total de dinero recaudado agrupado por el rango horario.	Identifica los períodos del día que generan mayor ingreso. Esto puede estar relacionado con los picos de demanda y es útil para mejorar la gestión del servicio en horas de mayor actividad.
	Monto Total Recaudado por Día Semanal	Agrupar por día de la semana el monto total cobrado, tomando valores de monto ajustados según Indec.	Visión de cuánto se recauda por día. Es útil para detectar patrones de consumo y planificar ingresos.

Tabla 2: Descripción de los KPIs

3.2.4. Implementación del ETL

3.2.4.1. Diseño General

El proceso ETL es el que convierte los datos fuente en información confiable y útil para la toma de decisiones. En este caso, el ETL se implementó usando Pyspark, la API de Python de Apache Spark, que permite escribir código con sintaxis Python y aprovechar toda la potencia del procesamiento distribuido de Spark.

El flujo completo se ejecutó desde el cluster Dataproc, corriendo múltiples jobs por cada periodo mes-año, tomando los archivos del bucket, procesándolos y escribiendo los resultados en BigQuery.

Para llevar a cabo todo este objetivo, es importante elaborar un diseño de implementación adecuado, que enumere los pasos principales y estos conforman a la vez el proceso ETL. Por ello, el flujo general del diseño es el siguiente:

1. Lectura y preprocesamiento del archivo de transacciones “Extracción”.
2. Lectura y preprocesamiento del archivo de transacciones “GPS”.
3. Operación join entre tablas, validaciones y filtrado para reducir datos innecesarios.
4. Cálculo de dimensiones de fecha para determinar días hábiles y escolares mediante el uso de una librería externa de Python (workalendar).
5. Detección geográfica de ciudades y departamentos a partir de coordenadas de latitud y longitud.

6. Escritura final en BigQuery usando el conector integrado Spark-BigQuery.

La siguiente figura 21 ilustra las etapas generales mencionadas:

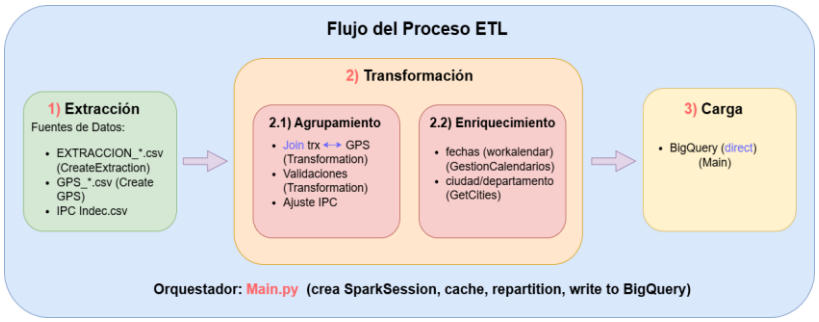


Figura 21: Etapas del Proceso ETL implementadas.

El script **Main.py** actúa como orquestador de la solución:

- Crea la SparkSession con configuraciones optimizadas para el cluster.
- Llama a los módulos que leen, transforman y enriquecen los datos.
- Escribe el dataframe final en BigQuery

La SparkSession es el punto inicial de la aplicación Spark. La configuración usada se observa en la Figura 22.

```
spark = SparkSession.builder \
    .appName("SUBE Solution Optimized") \
    .config("spark.sql.adaptive.enabled", "true") \
    .config("spark.sql.adaptive.coalescePartitions.enabled", "true") \
    .config("spark.sql.adaptive.skewJoin.enabled", "true") \
    .config("spark.sql.adaptive.localShuffleReader.enabled", "true") \
    .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer") \
    .config("spark.sql.execution.arrow.pyspark.enabled", "true") \
    .config("spark.sql.broadcastTimeout", "36000") \
    .config("spark.sql.debug.maxToStringFields", "1000") \
    .config("spark.sql.autoBroadcastJoinThreshold", "50m") \
    .config("spark.kryoserializer.buffer.max", "512m") \
    .config("spark.sql.adaptive.advisoryPartitionSizeInBytes", "128MB") \
    .getOrCreate()
```

Figura 22: Configuración del SparkSession.

- Las primeras cuatro se refieren al control de operaciones shuffle y optimización AQE (Adaptive Query Execution) que permite ajustar el tamaño de particiones y estrategias de join en tiempos de ejecución, resultando muy útil en el manejo de múltiples particiones en la red.
- El mecanismo de serialización aplicado es Kryo para mejorar el rendimiento en la transmisión de objetos entre nodos.
- Se hace uso de Apache Arrow para acelerar el uso de funciones definidas por el usuario aplicadas a dataframes (UDFs).
- Se aumenta el tiempo de espera y el tamaño máximo para efectuar operaciones broadcast join a todos los executors.

Para la lectura de los dos archivos csv se utilizan funciones especiales. La Figura 23 muestra el uso de la función cache() debido a la reutilización de ambos dataframes, evitando relecturas en los nodos.

```
1  df_gps = creategps(spark)
2  df_trx = createextration(spark)
3  df_gps.cache()
4  df_trx.cache()
```

Figura 23: Lectura y persistencia de los archivos de entrada.

En la Figura 24 se muestran las llamadas a las funciones optimizadas de los otros scripts auxiliares (CreateExtraction.py, CreateGPS.py, Transformation.py, etc.). Cada función recibe como parámetro el objeto spark creado en Main.py, lo que permite generar broadcasts variables para dataframes pequeños, como los índices IPC o los diccionarios de ciudades y departamentos. Esta estrategia optimiza los joins al reducir el shuffle, asegurando que cada executor tenga una copia local del dataframe pequeño y mejorando significativamente el rendimiento del cluster.

```
df_trx_final = jointables(df_gps, df_trx)
df_trx_final = obtener_fechas(df_trx_final)
df_trx_final = obtener_departamento_por_coordenadas(df_trx_final, spark)
```

Figura 24: Invocación de funciones optimizadas.

En cuanto a la selección de columnas finales y casteo, se asegura la compatibilidad de esquema con BigQuery, evitando errores por tipos de datos (Figura 25).

```
df_trx_final = df_trx_final.withColumn("c_control_point", col("c_control_point").cast("string")) \
                           .withColumn("file_id", col("file_id").cast("string")) \
                           .withColumn("NROTARJETA", col("NROTARJETA").cast("string")) \
                           .withColumn("ramal", col("ramal").cast("string")) \
                           .withColumn("CODIGOCONTRATO", col("CODIGOCONTRATO").cast("string"))

columnas = ['codigoentidad', 'idlinea', 'ramal', 'NROTARJETA', 'SALDO_AJUS',
            'CODIGOCONTRATO', 'CODIGOTIPOTRX', 'fecha', 'HoraTRX', 'diaHabil', 'calendarioEscolar', 'RangoHorario', 'diaSemanal', 'nombreMes',
            'TARIFA_AJUS', 'MONTO_AJUS',
            'DESCUENTO_AJUS', 'ID_COMBINACION', 'longitude', 'latitude', 'ciudad', 'departamento'
            ]
df_trx_final = df_trx_final.select(*columnas)
```

Figura 25: Casteo y selección de columnas finales.

3.2.4.2. Extracción de Datos

La etapa de extracción consistió en leer los dos archivos CSV a través de dos scripts: CreateExtraction.py, y CreateGPS.py. Durante esta fase se seleccionaron únicamente las columnas relevantes para el análisis propuesto. Además, se normalizaron nombres y formatos de las columnas desde el inicio, asegurando consistencia para los joins posteriores y facilitando las etapas siguientes del ETL. Se detallan los scripts que conforman la extracción:

Script de creación Primer Dataframe de Extracción

Se encarga de leer el csv de extracciones transaccionales en un determinado periodo, transformando el primer dataframe de extracción a un formato válido antes de aplicar la operación de join. Se hace uso de funciones nativas de Spark (split, withColumn, select) para explotar el paralelismo nativo. La lectura del archivo csv se demuestra en la siguiente Figura 26:

```
index = '01'
file = f'gs://bucket-sube/inputs/Extraccion/EXTRACCION_Prov-San-Juan_j{index}2023.csv'
df_trx = spark.read.csv(file, sep=';', header=True, inferSchema=True)
```

Figura 26: Importación del primer dataframe.

Se selecciona un archivo período mes-año (01/2023 en este caso) del bucket y se importa como dataframe de Spark. Se detalla el separador de las columnas, con encabezado incluido e inferencia del esquema, lo que agiliza la ejecución del job.

```
df_trx = df_trx.withColumn('HoraTRX', F.split('FECHATRX', ' ')[1].cast(StringType())) \
               .withColumn('fecha', F.split('FECHATRX', ' ')[0].cast(StringType())) \
               .withColumnRenamed('ID_POSICIONAMIENTO', 'c_control_point') \
               .withColumnRenamed('IDARCHIVOINTERCAMBIO', 'file_id') \
               .withColumnRenamed('CODIGOENTIDAD', 'codigoentidad') \
               .withColumnRenamed('IDLINEA', 'idlinea')
```

Figura 27: Modificaciones en las columnas del primer dataframe.

Se usa la función nativa split para descomponer la columna fechaHoraTrx en dos nuevas columnas: fecha y hora de la transacción por separado. Además, se renombran las columnas para que tengan los mismos nombres respecto al otro dataframe y efectuar un join consistente (c_control_point y file_id) (Figura 27).

```
df_ipc = spark.read.csv('gs://bucket-sube/inputs/IPC Indec.csv', sep=';', header=True, inferSchema=True)
df_trx = calculos_ipc(df_trx, df_ipc)
```

Figura 28: Importación del dataframe IPC.

Se lee el archivo IPC Indec.csv y se llama a la función calculos_ipc. En dicha función se aplican ajustes de tarifa por inflación generando nuevas columnas con valores ajustados. Esto permite realizar una comparación de tarifas de transporte público en la actualidad (Figura 28).

```
columns_needed = ['codigoentidad', 'idlinea', 'NROTARJETA', 'SALDO',
                  'CODIGOCONTRATO', 'CODIGOTIPOTRX', 'fecha', 'HoraTRX',
                  'VALOR_TARIFA', 'MONTO', 'DESCUENTO', 'ID_COMBINACION',
                  'c_control_point', 'file_id', 'TARIFA_AJUS', 'DESCUENTO_AJUS',
                  'MONTO_AJUS', 'SALDO_AJUS']
df_trx = df_trx.select(*columns_needed)
df_trx = df_trx.repartition(4, col("c_control_point"))
```

Figura 29: Selección de columnas finales para primer dataframe.

Finalmente se seleccionan el conjunto de columnas requeridas, reduciendo el tamaño del dataframe que ayudan también a reducir el ancho de banda y memoria usada por los nodos.

Por otro lado, se realiza una repartición por una columna clave que reduce el shuffle antes de la operación join, evitando un alto costo de cómputo (Figura 29). Se generan cuatro particiones que se distribuyen a los núcleos de los nodos, aumentando el nivel de paralelismo.

Script de creación Segundo Dataframe GPS

Se lee el segundo archivo csv de transacciones “gps” en el mismo periodo mes-año del otro archivo csv, transformando este dataframe al formato válido antes del join. La lectura se realiza de la manera como demuestra la figura 30:

```
index = '01'
file = f'gs://bucket-sube/inputs/GPS/GPS_SAN_JUAN_{index}2023.csv'
df_gps = spark.read.csv(file, sep=';', header=True, inferSchema=True)
```

Figura 30: Importación del segundo dataframe.

Se toma un archivo por periodo del bucket y se importa como dataframe, utilizando la misma configuración con esquema inferido.

Figura 31: Selección de columnas requeridas y repartición del segundo dataframe.

Por último, se renombra una columna y se seleccionan sólo el conjunto de columnas requeridas, reduciendo el tamaño del dataframe resultante, como demuestra la Figura 31.

Además, se aplica la repartición por la columna clave tal como lo hace el dataframe de extracción. Esto ayuda a que cuando se haga el join por c_control_point y file_id, los datos ya estén particionados por la clave, reduciendo el coste de shuffle.

3.2.4.3. Transformación de Datos

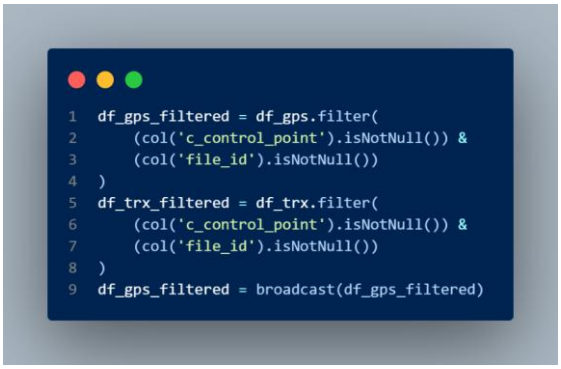
La transformación es el núcleo del proceso ETL, combinando las dos fuentes (transacciones + GPS), aplicando validaciones y generando columnas derivadas necesarias para el análisis. Se prioriza la eficiencia en los joins proyectando solo columnas esenciales y eliminando registros nulos de entrada, además, se utiliza la estrategia de Broadcast Join tanto para la integración de transacciones con GPS como para el ajuste monetario por IPC. Esto permite que las tablas más pequeñas se distribuyan a todos los nodos, minimizando el intercambio de datos (shuffle) y reduciendo drásticamente la latencia. Finalmente, todas las validaciones se consolidan en una sola operación lógica para que el motor de ejecución realice una única pasada sobre los datos.

Por otro lado, el enriquecimiento contextual añade valor analítico mediante la generación de dimensiones temporales y geográficas. A través de funciones de usuario (UDFs) y la librería workalendar, el sistema identifica periodos críticos como días hábiles y el calendario escolar de

San Juan. Asimismo, se implementa una lógica de geolocalización que traduce coordenadas en ciudades y departamentos específicos. Este proceso transforma datos brutos en columnas derivadas (horarios, meses, ubicación administrativa) que son fundamentales para el análisis posterior en la solución de inteligencia de negocios. Esta fase implementa la lógica de los siguientes scripts:

Script que Une los dos Archivos Fuente

El script tiene como objetivo realizar el join optimizado entre ambos dataframes transaccionales y luego aplicar una serie de validaciones sobre las columnas, transformando el dataframe objetivo a un formato válido.



```
df_gps_filtered = df_gps.filter(
    (col('c_control_point').isNotNull()) &
    (col('file_id').isNotNull())
)
df_trx_filtered = df_trx.filter(
    (col('c_control_point').isNotNull()) &
    (col('file_id').isNotNull())
)
df_gps_filtered = broadcast(df_gps_filtered)
```

Figura 32: Filtrado temprano y broadcast del segundo dataframe.

Se aplica un filtrado temprano sobre las columnas clave relacionadas para efectuar el join, tomando solo aquellas transacciones que sí se encuentran relacionadas entre ambas partes (Figura 32).

Además, se aplica el broadcast sobre el dataframe más pequeño, haciendo que cada ejecutor tenga dicho dataframe. Esta decisión de rendimiento es muy importante porque optimiza la aplicación del join evitando una mayor sobrecarga del clúster.

```
df_join = df_gps_filtered.join(
    df_trx_filtered,
    on=['c_control_point', 'file_id'],
    how='inner'
)
```

Figura 33: Join entre los dos dataframes de entrada.

En la figura 33 se efectúa la operación inner join (join interno) sobre las dos claves y luego se aplica la función transformation para validar los campos y formatos del dataframe.

```
df_trx_final = df_trx.filter(
    (col('longitude').isNotNull()) & (col('latitude').isNotNull()) &
    (col('longitude').cast(DoubleType()).between(-180, 180)) &
    (col('latitude').cast(DoubleType()).between(-90, 90)) &
    (col('codigoentidad').isNotNull()) & (col('codigoentidad').cast(IntegerType()) > 0) &
    (col('idlinea').isNotNull()) & (col('idlinea').cast(IntegerType()) > 0) &
    ((col('ID_COMBINACION').isNull()) | (col('ID_COMBINACION').cast(IntegerType()).between(1, 3)))
)
```

Figura 34: Filtrado de columnas del dataframe resultante.

La función de transformación aplica en una sola pasada el filtro que valida el formato de algunas columnas requeridas. Esto hace que si una fila (transacción) no cumple con alguna condición, no se considere válida para el dataframe resultante (Figura 34).

Se valida que los pares de coordenadas se encuentren dentro del rango permitido, que los códigos de entidad, interno y línea sean valores numéricos positivos, además que el identificador de trasbordos también se encuentre dentro del rango válido.

Esta forma de evaluar los valores de la columna hace que se reduzca el volumen del dataframe además de aplicar las pasadas una sola vez evitando sobrecargar el clúster.

Enriquecimiento Analítico

Las siguientes transformaciones permitieron materializar las métricas monetarias, además de las dimensiones temporales y geográficas definidas en el diseño conceptual del almacén.

Script que Ajusta los Importes

El script implementa la lógica de ajuste monetario (indexación por Índices de Precios Consumidor) de los montos transaccionales, tomando como base la categoría Transporte en la región Cuyo. Los cálculos se aplican sobre las columnas valor tarifa, monto, descuento y saldo, que expresan valores monetarios nominales que deben ser ajustados para mantener coherencia económica, generando cuatro nuevas columnas: TARIFA_AJUS, DESCUENTO_AJUS, MONTO_AJUS y SALDO_AJUS.

Se detalla la fórmula IPC que se utiliza:

$$Valor\ Tarifa\ Real\ (En\ Marzo\ 2025) = \frac{IPC\ Marzo\ 2025}{IPC\ FechaTransacción} * Valor\ Tarifa\ Fecha\ Transacción$$

Si se toma el valor de tarifa común de Agosto 2023 con valor de $150.00 y se compara con el de Marzo 2025 que equivale a $616,22, se aprecia el impacto de la inflación nacional en el tiempo, donde el poder adquisitivo de la moneda ha disminuido.

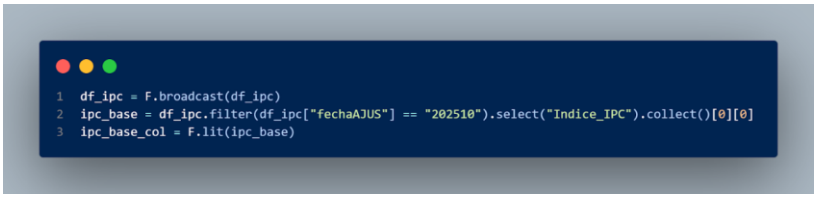
Para llevar a cabo todo esto, es necesario desarrollar en Pyspark:

Primera etapa que consiste en la preparación del dataframe IPC. Se explica con mayor detalle la generación de este archivo en la sección de Apéndice.

```
df_ipc = df_ipc.withColumnRenamed('Periodo', 'fechaAJUS') \
               .withColumn("Indice_IPC", F.regexp_replace(F.col("Indice_IPC"), ",", ".")) \
               .withColumn("Indice_IPC", F.col("Indice_IPC").cast(DoubleType()))
```

Figura 35: Modificaciones del dataframe IPC.

En la figura 35 se muestra cómo se renombra la columna clave para hacer el join con el dataframe de transacciones “extracción”. Además, se normaliza el formato numérico pues el csv entrante define al separador decimal como “,”.



```
df_ipc = F.broadcast(df_ipc)
ipc_base = df_ipc.filter(df_ipc["fechaAJUS"] == "202510").select("Indice_IPC").collect()[0][0]
ipc_base_col = F.lit(ipc_base)
```

Figura 36: Broadcast del dataframe IPC.

Antes de realizar la operación de join se definen cuestiones a tener en cuenta: Primero se aplica la operación de broadcast, que sugiere al optimizador de Spark que trate df_ipc como una tabla pequeña que puede ser copiada a cada executor (Figura 36).

Luego se aplican operaciones para obtener el índice base IPC en un periodo de referencia hardcodeado. Esto se hace filtrando por ejemplo un periodo base “202510” (añomes) y usar la función collect para traer ese valor al driver. Se envuelve el valor con la operación lit para usarlo como literal en las expresiones del cálculo de la nueva columna en Spark (ipc_base / indice_ipc_mes). A propósitos del trabajo de investigación y periodo de prueba gratuita de Dataproc, se toma como periodo base final “202510” ya que corresponde al último al momento de la construcción de la solución.

```
df_tarifas = df_trx_enero.withColumn(
    'fechaAJUS',
    F.concat(
        F.split(df_trx_enero['fecha'], '/').getItem(2).cast(StringType()),
        F.split(df_trx_enero['fecha'], '/').getItem(1).cast(StringType())
    )
).withColumn("VALOR_TARIFA", F.col("VALOR_TARIFA").cast(DoubleType())) \
 .withColumn("DESCUENTO", F.col("DESCUENTO").cast(DoubleType())) \
 .withColumn("MONTO", F.col("MONTO").cast(DoubleType())) \
 .withColumn("SALDO", F.regexp_replace(F.col("SALDO"), ",", ".").cast(DoubleType()))

df_join = df_tarifas.join(df_ipc, on="fechaAJUS", how="left")
```

Figura 37: Casteo de columnas y left-join del dataframe IPC.

Lo último que se hace antes del join es crear la columna fechaAJUS con el formato requerido yyyyMM para alinearse con la otra columna clave del df_ipc y castear las columnas de valores

monetarios para las posteriores operaciones matemáticas que conformarán las nuevas columnas (Figura 37).

Se ejecuta un left join teniendo en cuenta que algunas transacciones pueden no tener índice IPC aplicable. Por el broadcast anterior se ejecuta el join como broadcast join, aprovechando que df_ipc es pequeño evitando shuffle.

Figura 38: Creación de nuevas columnas con valores ajustados IPC.

La figura 38 expone operaciones donde la primera, aplica el cálculo de las nuevas columnas usando una sola transformación, generando los valores ajustados según la ecuación mencionada. Utiliza la función round para redondear a dos decimales.

La segunda operación genera la última columna a partir del cálculo de las columnas creadas en la operación anterior, evitando recalcular expresiones pesadas.

Script que Agrega Datos de Fechas

El script tiene como objetivo definir un calendario argentino personalizado que se extiende a partir de la librería de python workalendar, para incluir los feriados provinciales, como el día del maestro y la fundación de San Juan. La librería permite determinar a partir de una fecha si corresponde a un día laborable, feriado, semana santa, fin de semana entre otras cosas.

Se busca mejorar este comportamiento contemplando lo provincial, generando nuevas columnas para día hábil y calendario escolar, que son cuestiones importantes que influyen en el servicio de transporte público.

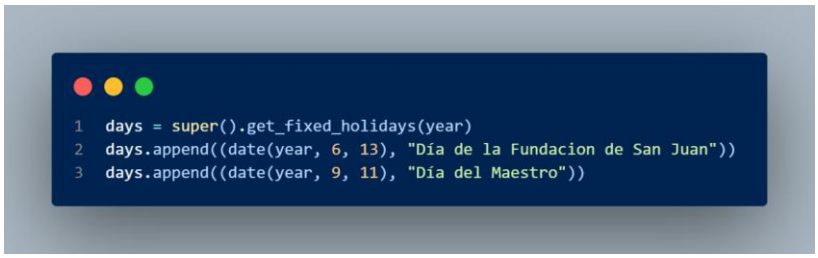


Figura 39: Definición de feriados provinciales.

Se crea una clase personalizada extendida del calendario de Argentina donde se le añaden dos feriados provinciales, únicamente para los años específicos de uso (Figura 39).

Figura 40: Modificaciones para la gestión de calendarios.

Antes de aplicar las funciones definidas, se debe modificar la columna fecha a un formato Date válido para ser evaluado, ya que la librería workalendar trabaja sobre objetos Date, no cadenas de texto, tal como demuestra la Figura 40.

Se opta por utilizar UDF ya que la lógica exige definir nuevas funciones personalizadas y complejas que no están integradas en Spark, teniendo que aplicarlas en cada fila del dataframe. Su desventaja es que no son funciones muy optimizadas como las funciones nativas de Spark.

En este caso, ambas funciones UDF implementan lógica personalizada haciendo uso de funciones nativas de Spark, con lo cual se obtiene un buen equilibrio en rendimiento.

Al definir las UDFs se detalla el tipo de dato de retorno, que es el tipo de dato de las nuevas columnas (Figura 41).

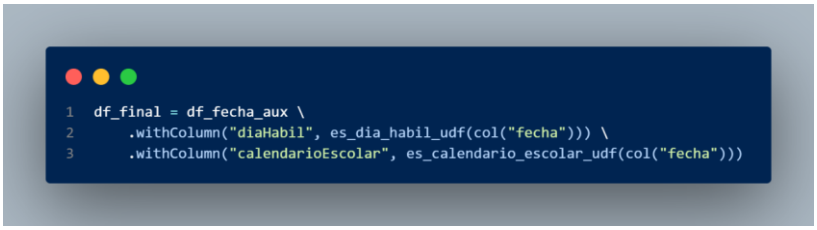


Figura 41: Invocación de las UDFs para la gestión de calendarios.

Finalmente se vuelve a cambiar el formato de fecha usando la función date_format para que vuelva a su tipo de dato String (Figura 42).

Figura 42: Casteo a formato string luego de la gestión calendarios.

Script que Agrega Datos de Ciudades

Script que se encarga de la creación de nuevas columnas: ciudad y departamento de la provincia a partir del par de coordenadas de la transacción. Para ello se lee un archivo csv con los límites mínimos y máximos de los pares de coordenadas (bounding boxes), resguardandose en una estructura de datos de tipo diccionario, para que luego se implemente un UDF con la lógica de obtención de ciudades y departamentos.

El archivo CSV es construido manualmente con asistencia de sitios web oficiales como OpenStreetMap, Google Maps, Wikipedia y City Population [37] [38]. Tiene como encabezado el nombre de la ciudad, departamento, latitud mínima, latitud máxima, longitud mínima y longitud máxima. Además, es ordenado ascendentemente primero por rangos abarcativos de coordenadas geográficas y luego por población.

Se importa el archivo csv sabiendo que es un dataframe pequeño que puede ser trabajado para mayor optimización como diccionario (Figura 43).

```
df_ciudades = spark.read.csv('gs://bucket-sube/scripts/ciudades_sanjuan_ordenado_por_area2.csv',
sep=",", header=True, inferSchema=True)
ciudades_dict = {
    row['ciudad']: {
        'departamento': row['departamento'],
        'min_lat': row['min_latitud'],
        'max_lat': row['max_latitud'],
        'min_lon': row['min_longitud'],
        'max_lon': row['max_longitud']
    }
    for row in df_ciudades.collect()
}
```

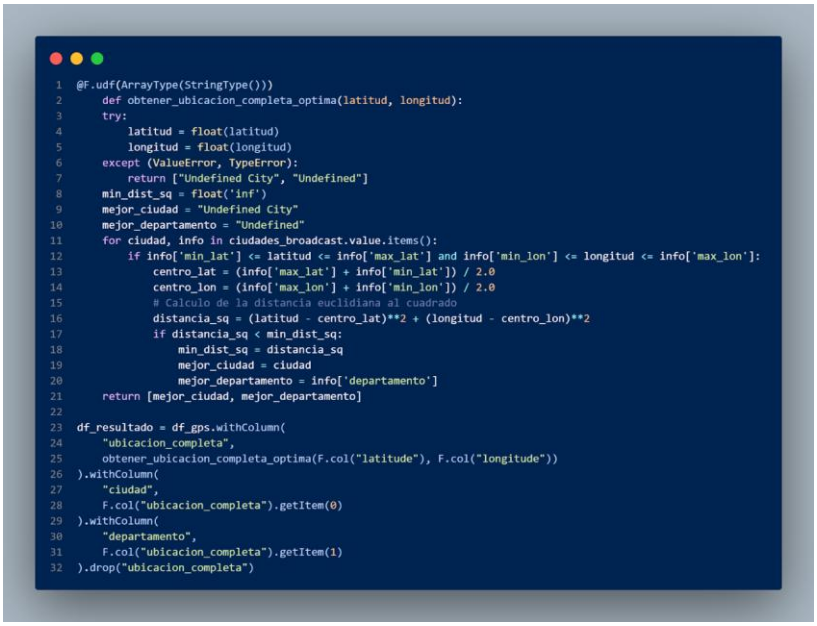
Figura 43: Importación del dataframe para la gestión de ciudades.

La Figura 44 muestra cómo se crea su variable broadcast aprovechando que es una estructura pequeña y permitiendo que cada executor posea una copia local del diccionario.

```
ciudades_broadcast = spark.sparkContext.broadcast(ciudades_dict)
```

Figura 44: Broadcast del dataframe de gestión de ciudades.

El UDF itera sobre todos los límites geográficos y aplica un filtro inicial de pertenencia a rango para descartar rápidamente las ciudades no candidatas. Para los bounding boxes que contienen la coordenada, se calcula la distancia euclidiana al cuadrado entre el punto de entrada y el centro geográfico de la ciudad (Figura 45).



```python
@F.udf(ArrayType(StringType()))
    def obtener_ubicacion_completa_optima(latitud, longitud):
    try:
        latitud = float(latitud)
        longitud = float(longitud)
    except (ValueError, TypeError):
        return ["Undefined City", "Undefined"]
    min_dist_sq = float('inf')
    mejor_ciudad = "Undefined City"
    mejor_departamento = "Undefined"
    for ciudad, info in ciudades_broadcast.value.items():
        if info['min_lat'] <= latitud <= info['max_lat'] and info['min_lon'] <= longitud <= info['max_lon']:
            centro_lat = (info['max_lat'] + info['min_lat']) / 2.0
            centro_lon = (info['max_lon'] + info['min_lon']) / 2.0
            # Calculo de la distancia euclidiana al cuadrado
            distancia_sq = (latitud - centro_lat)**2 + (longitud - centro_lon)**2
            if distancia_sq < min_dist_sq:
                min_dist_sq = distancia_sq
                mejor_ciudad = ciudad
                mejor_departamento = info['departamento']
    return [mejor_ciudad, mejor_departamento]

df_resultado = df_gps.withColumn(
    "ubicacion_completa",
    obtener_ubicacion_completa_optima(F.col("latitude"), F.col("longitude"))
).withColumn(
    "ciudad",
    F.col("ubicacion_completa").getItem(0)
).withColumn(
    "departamento",
    F.col("ubicacion_completa").getItem(1)
).drop("ubicacion_completa")
```

Figura 45: UDF para la gestión de ciudades y departamentos.

Los puntos dados son las coordenadas que llegan como parámetros, mientras que el centro del bounding box de la ciudad se calcula como indica la ecuación.

$$Distancia^2 = (\, lat_{punto} - lat_{centro})^2 + (\, lon_{punto} - lon_{centro})^2$$

$$lat_{centro} = \frac{maxLatitud + minLatitud}{2} \qquad lon_{centro} = \frac{maxLongitud + minLongitud}{2}$$

El resultado final corresponde a la ciudad y departamento asociados a la distancia mínima encontrada, asegurando así la asignación más precisa cuando existen rangos geográficos superpuestos. Esta metodología asegura eficiencia al reducir los cálculos de distancia sólo a los rangos pertinentes, y precisión en la geocodificación.

Se retorna un ArrayType que encapsula los dos datos, obteniendo ambos valores en una sola pasada de datos, evitando una mayor sobrecarga de distribución.

Finalmente se hace uso de una columna auxiliar temporal “ubicación_completa” para garantizar que la invocación del UDF costoso se realice una sola vez, extrayendo los ítems requeridos en las nuevas columnas resultantes.

Resulta importante detallar, la existencia de alternativas a este método, que requieren el uso de servicios externos o APIs dedicadas que imponen una serie de desafíos operativos y de rendimiento considerables. Específicamente, cada solicitud de resolución de coordenadas requiere una llamada o consumo de estos servicios o APIs. En el contexto de un cluster de procesamiento distribuido, como el utilizado, esto significa que cada par de coordenadas geográficas distinto debe ser enviado, procesado y devuelto por el servicio externo antes de que el procesamiento interno pueda continuar.

Este patrón de consumo genera una sobrecarga de latencia considerable. La latencia se acumula por el tiempo de viaje de ida y vuelta de la solicitud a través de la red y el tiempo que le toma al proveedor del servicio procesar y responder a la consulta. Cuando se manejan grandes volúmenes de datos o se requiere una respuesta en tiempo casi real, la suma de estas latencias para múltiples consultas puede degradar seriamente el rendimiento general del sistema.

Además, el uso constante de servicios externos implica un mayor consumo de recursos dentro del cluster. No solo se consumen recursos de red para gestionar las comunicaciones externas, sino que también se requieren recursos de CPU y memoria para la serialización, deserialización y manejo de las respuestas de las APIs, lo que resulta en una huella de procesamiento más grande para una tarea que, idealmente, debería resolverse de manera eficiente y local. Finalmente, la dependencia de servicios de terceros introduce una capa de riesgo operativo y, potencialmente, costes asociados al volumen de peticiones.

3.2.4.4. Carga de los Datos (Almacén) en BigQuery

La etapa final consiste en escribir el dataframe ya transformado y enriquecido en una tabla de BigQuery. Para ello se utiliza el conector integrado de BigQuery-Spark que agiliza este tipo de operación costosa. La configuración de escritura en modo directo hace que los datos se transmitan de Dataproc a BigQuery evitando pasos intermedios costosos que degradan el rendimiento del clúster. Usar el método “append” hace que se vayan guardando los datos en el orden de llegada, teniendo en cuenta que se ejecutan múltiples jobs independientes por mes-año.

El programa Main.py crea la SparkSession con configuraciones optimizadas para el cluster, llama a los módulos que leen, transforman y enriquecen los datos y finalmente escribe en BigQuery.

Como la creación del clúster Dataproc está limitado a los recursos de la prueba gratuita, el procesamiento del volumen se lleva a cabo tomando periodos mes-año, ejecutando un job por cada periodo. Cada job tiene como programa principal a Main.py que va aplicando todo el flujo descrito hasta realizar la carga incremental a BigQuery.

La ejecución de cada job requierió tengan que modificarse manualmente los archivos entrantes de extracción y gps.

A continuación, se detalla el comando que desencadena la ejecución de un job por medio de la terminal SDK de Google Cloud:

gcloud dataproc jobs submit pyspark gs://bucket-sube/scripts/Main.py --cluster=cluster-sube --region=us-central1 --py-files=gs://bucket-sube/scripts/IPC.py,gs://bucket-sube/scripts/CreateExtraction.py,gs://bucket-sube/scripts/CreateGPS.py,gs://bucket-sube/scripts/Transformation.py,gs://bucket-sube/scripts/GetCities.py,gs://bucket-sube/scripts/GestionCalendarios.py

Se detalla el tipo de job a enviar al clúster (Pyspark), siguiendo del programa principal, nombre, región del clúster y finalmente la serie de archivos de python que son invocados desde el programa principal.

Además, dentro de Main.py se realiza la escritura del dataframe resultante con las columnas casteadas y seleccionadas explícitamente. La configuración se observa en la Figura 46:

Figura 46: Configuración de la escritura a BigQuery.

- Se define que la escritura del dataframe se hizo por medio de la API de BigQuery, haciendo uso del conector Spark-BigQuery que viene integrado en el clúster.
- Se detalla la ruta absoluta de la tabla destino en BigQuery, detallando el identificador del proyecto, del conjunto de datos y el nombre de la tabla.
- Método de escritura directa donde los datos se escriben directamente en el storage de BigQuery, sin depender de otro proceso de carga, ofreciendo un mejor rendimiento gracias a la integración directa.
- A pesar de que la escritura sea directa se utiliza un bucket temporal que ayuda a BigQuery en la escritura del gran volumen de información, donde los datos pueden ubicarse en este sector como área de stage o preparación.
- Se agregan los parámetros "CREATE_IF_NEEDED" para crear la tabla si no existe y "WRITE_APPEND" para añadir datos del dataframe al final de la tabla existente en BigQuery. El modo de escritura es siempre append debido a los múltiples jobs que se ejecutan.
- Opciones ante la modificación del esquema: BigQuery permite que se puedan añadir las nuevas columnas durante el proceso de escritura. Por otra parte permite "relajar" las restricciones de las columnas existentes ya que algunas pueden aceptar valores nulos. Estas opciones aceptan cambios menores en el esquema sin fallar la carga, permitiendo un pipeline tolerante a cambios incrementales del esquema de salida.

3.2.5. Control de versiones y automatización del despliegue

Con el objetivo de garantizar la integridad del código y el seguimiento estructurado de los cambios, se implementó un flujo de trabajo basado en control de versiones y automatización del despliegue, alineado con prácticas de Integración y Despliegue Continuo (CI/CD).

Este enfoque permitió reducir errores asociados a la manipulación manual de archivos, mejorar la trazabilidad del desarrollo y asegurar la consistencia entre el entorno de desarrollo y el entorno de ejecución en la nube.

Git y GitHub: Gestión del Código

El desarrollo de los scripts de PySpark se gestionó mediante Git como sistema de control de versiones. Esta herramienta permite mantener un historial detallado de las modificaciones

Comentado [MR1]: Donde lo colocariamos?

Comentado [MR2R1]: El punto posterior habla de sincronizacion, no se si refiere a lo mismo...

Comentado [GA3R1]: Este texto es introductorio al tema. Los subtemas son Git/GitHub y GitHub Actions

Comentado [MR4R1]: quiero q ud lo ubique/cambie, porque no quiero meter la pata. forma parte de la arquitectura? o es una aclaracion sobre el codigo empleado? donde? para la construccion del ETL? entonces, donde lo podriamos colocar? tal vez como un texto sin titulo??? no se... pero a mi entender no es componente de la arquitectura, po lo q no debiera estar como un subitem directo dentro de la arquitectura...

realizadas sobre el código, facilitando la identificación de cambios y la reversión ante errores durante el desarrollo.

Como repositorio remoto se utilizó GitHub, que actúa como servidor centralizado para el almacenamiento del código fuente en la nube. La sincronización entre el entorno de desarrollo local y el repositorio remoto asegura la persistencia del trabajo realizado, así como su disponibilidad para revisión técnica, auditoría y futuras extensiones del proyecto.

El uso combinado de Git y GitHub favorece además la adopción de buenas prácticas de ingeniería de software, tales como el versionado explícito del código, la documentación de cambios mediante mensajes de commit y la organización estructurada de los scripts del proceso ETL (Figura 16).

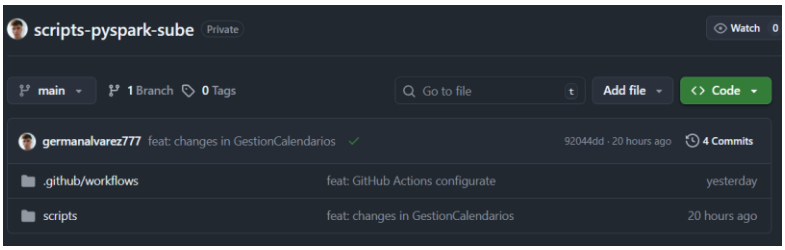


Figura 16: Repositorio remoto de GitHub

GitHub Actions: Automatización de la Sincronización

Los scripts de PySpark utilizados por el clúster Dataproc se almacenaron en el bucket configurado en GCS. Para evitar la actualización manual de estos archivos y mantener coherencia entre el repositorio y el entorno de ejecución, se implementó un mecanismo de sincronización automática mediante GitHub Actions.

Se configuró un workflow que se ejecuta automáticamente ante cada evento de push sobre la rama principal del repositorio.

Dicho flujo de trabajo consta de las siguientes etapas:

1. **Autenticación**: El workflow se autentica contra GCP mediante una cuenta de servicio dedicada, cuyas credenciales se gestionan a través de Secrets cifrados provistos por GitHub. Este mecanismo garantiza la seguridad del acceso a la infraestructura en la nube y evita la exposición de credenciales sensibles en el código fuente.

2. **Sincronización de objetos**: Una vez autenticado, el workflow ejecuta el comando *gsutil rsync*, que compara el contenido del directorio de scripts del repositorio, con el subdirectorio correspondiente en el bucket (*bucket-sube/inputs/scripts*).
3. **Actualización incremental de scripts**: Solo los archivos que presentan modificaciones son transferidos al bucket, asegurando una sincronización eficiente y manteniendo el entorno de ejecución en la nube como un espejo en tiempo real del código versionado en el repositorio.

La figura 17 muestra cómo se desencadena la ejecución del workflow luego de un evento de push, desde GitHub Actions.

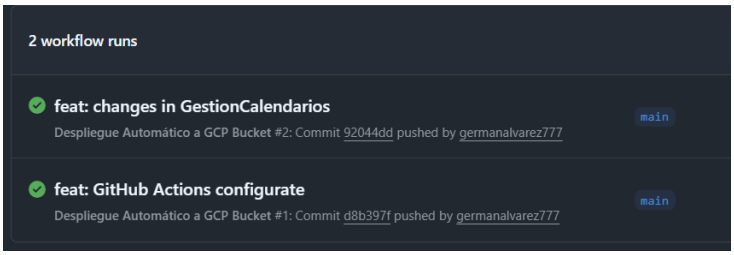


Figura 17: Workflows de sincronización en GitHub Actions

Este mecanismo de automatización establece un puente directo entre el desarrollo del código y su disponibilidad en el entorno de GCP, optimizando los tiempos de despliegue.

3.2.6. Diseño e Implementación del Panel de Control

Una vez completada la carga de los datos en el almacén de datos (BigQuery), se avanzó con el diseño y la construcción del panel de control, orientado a la visualización y exploración dinámica de la información procesada. Para ello, se utilizó Looker Studio, debido a su integración nativa con BigQuery y a la flexibilidad de su entorno visual. Esta herramienta permite conectarse directamente a las tablas, estableciendo una conexión segura mediante credenciales de Google Cloud. Además, permite que el panel puede compartirse mediante un enlace público o restringido, garantizando la accesibilidad desde cualquier navegador y manteniendo la trazabilidad de la información.

El proceso de importación de datos tomó como fuente principal la tabla resultante del procesamiento realizado en PySpark. A partir de esta información se configuraron los distintos

elementos del panel de control, dando lugar a visualizaciones interactivas e interpretables por los usuarios.

La Figura 47 expone la importación de la tabla, denominada “viajes final”.

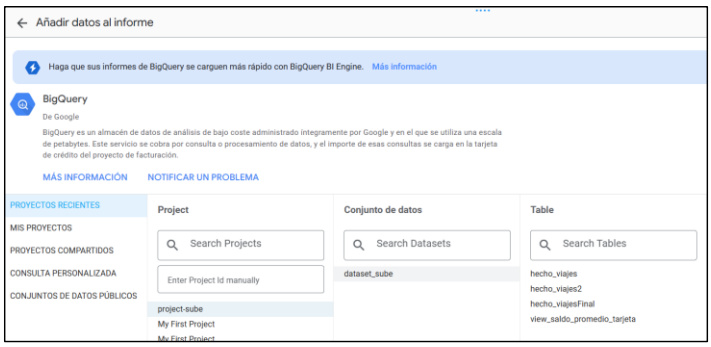


Figura 47: Importación de la tabla “viajes final”.

Una de las principales ventajas de Looker Studio es su interfaz intuitiva y adaptable, que permite crear gráficos, tablas, indicadores y filtros sin necesidad de programación adicional. Cada componente visual se encuentra vinculado en tiempo real con BigQuery, lo que posibilita la exploración directa de los datos sobre la infraestructura en la nube. No obstante, dado que cada interacción genera una consulta SQL hacia BigQuery, implicando consumo de recursos de procesamiento, se decidió que la primera versión del panel se construyera utilizando un único período mes-año. Esta decisión permitió optimizar el rendimiento y controlar costos durante la etapa de pruebas y validación del modelo visual.

Asimismo, se incorporaron nuevos datos (campos calculados personalizados) que permiten transformar o enriquecer la información visualizada. Por ejemplo, algunos valores enteros o codificados son convertidos en descripciones textuales más legibles para el usuario final, aplicando funciones lógicas y expresiones de transformación directamente desde la interfaz de Looker Studio. Con ello la lectura de los gráficos resulta más comprensible y accesible, sin necesidad de modificar la tabla fuente.

El panel de control fue diseñado bajo un enfoque modular e interactivo, compuesto por cinco páginas o secciones principales que representan una categoría analítica específica:

1. Indicadores de Viajes.
2. Indicadores de Trasbordos.

3. Indicadores de Geolocalización.
4. Indicadores de Perfil de Usuario.
5. Indicadores de Entidades.
6. Indicadores de Montos.

Cada página incorpora filtros dinámicos de control que permiten seleccionar los datos a analizar. Ellos representan dimensiones del modelo definido (meses, ciudades, departamentos, tipos de contrato, línea de transporte, etc.). Es decir, los filtros no responden a selecciones arbitrarias de datos, sino que se sustentan directamente en la estructura dimensional del almacén, permitiendo analizar los hechos registrados desde múltiples perspectivas. Esta alineación de la visualización con el modelo dimensional de los datos asegura estabilidad en los cálculos, claridad conceptual en los indicadores y consistencia en los distintos niveles de análisis.

Diseño Analítico de Visualizaciones

La construcción de las visualizaciones en Looker Studio se realizó en coherencia con el modelo dimensional previamente definido. Cada gráfico se estructura a partir de dimensiones, que establecen el criterio de agrupamiento, y métricas, que expresan los valores agregados objeto de análisis. A continuación, se describen los principales tipos de visualización implementados y su configuración dentro del panel.

Gráficos de Barras

Para la visualización del comportamiento económico por segmentos se implementó un gráfico de barras configurado con las dimensiones “Tipo de Contrato” y “Período de Fecha”, y la métrica “Saldo Promedio”.

“Saldo Promedio” constituye un campo calculado definido específicamente para esta visualización, obtenido como la media aritmética de la columna “saldo ajustado”. Esta configuración permite analizar comparativamente el nivel promedio de saldo entre categorías contractuales y su evolución dentro del período seleccionado en el panel.

Usar este tipo de gráficos, que proporcionan una estructura clara y jerarquizada, permiten identificar rápidamente los contrastes entre categorías, resultando especialmente útiles para analizar indicadores económicos o de desempeño operativo. La Figura 48 muestra el grafico y su implementación:

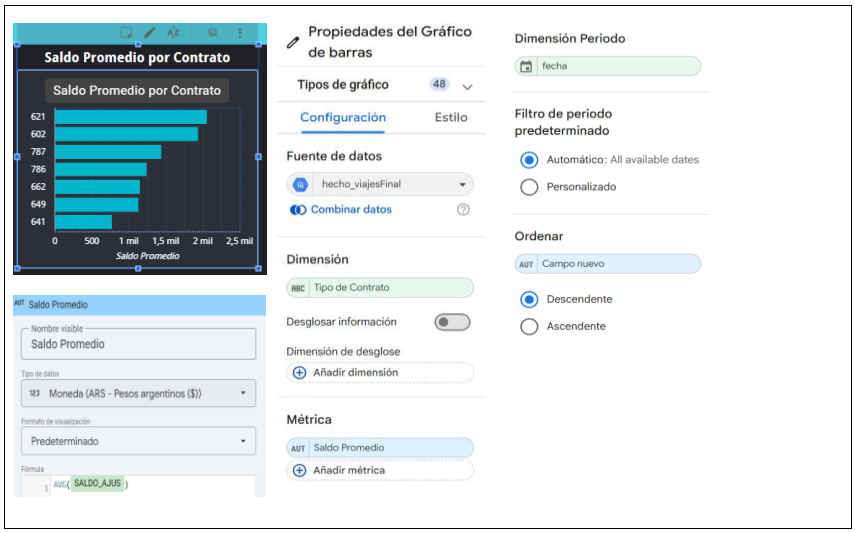


Figura 48: Detalle y configuración de gráficos de barra en Looker Studio.

Gráficos Circulares

Otro tipo de grafico utilizado fue el gráfico circular. Se usó para analizar la cantidad de viajes según las dimensiones establecidas. La métrica asociada en este caso es la Cantidad de Viajes, que se calcula como el conteo total de registros agrupados por cada categoría de las dimensiones. En este caso, la segmentación se realizó utilizando una dimensión derivada como campo calculado, lo que permitió clasificar los registros según el criterio de día hábil o no hábil con mayor claridad interpretativa.

La Figura 49 muestra el grafico obtenido, y la definición correspondiente:

Comentado [MR5]: Creo q no lo pondria...

Comentado [GA6R5]: La mención de dimensión derivada es respecto a la figura 49

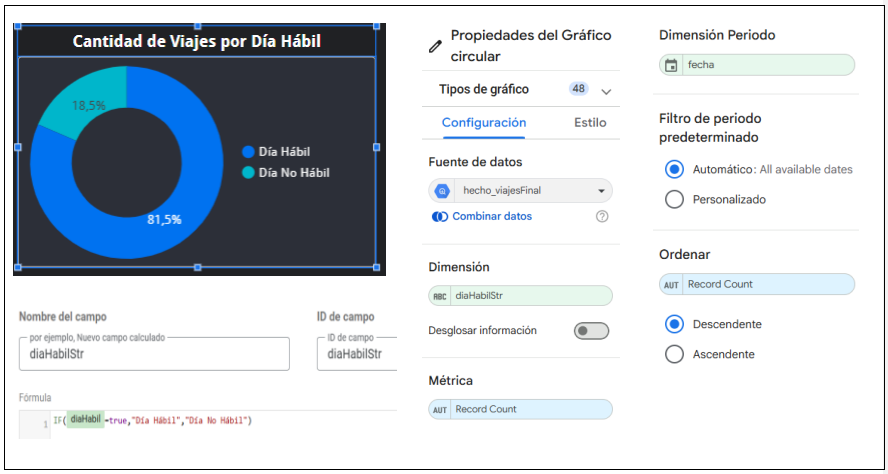


Figura 49: Detalle y configuración de gráficos circulares en Looker Studio.

Como puede observarse, este tipo de gráfico facilita una interpretación visual inmediata de la distribución porcentual de los datos, mostrando de forma intuitiva el peso relativo de cada segmento dentro del total de viajes registrados en un período determinado. Es una herramienta eficaz para comparar proporciones o evaluar la representatividad de distintas categorías operativas.

Mapas de Calor

Se implementó un mapa de calor para representar la cantidad de viajes y la cantidad de trasbordos por línea y ramal, mostrado en la Figura 50. La visualización adopta un formato matricial, en el que cada fila corresponde a una combinación de estas variables. Los valores se muestran en forma numérica y se acompañan de una escala de intensidad cromática. El uso del color permite identificar de manera inmediata qué combinaciones concentran mayores o menores valores. Las tonalidades más intensas indican cantidades relativamente más altas, mientras que las más atenuadas señalan magnitudes inferiores, facilitando la comparación entre líneas y ramales según los filtros aplicados.

Desde el punto de vista analítico, esta dinámica permite explorar distintos niveles de agregación. Al aplicar filtros más específicos se profundiza en el detalle; al ampliar la selección se obtiene una visión más agregada. Este comportamiento puede compararse conceptualmente con las operaciones de drill-down y drill-up propias de las herramientas OLAP, aunque en este

caso se logra mediante la combinación de filtros y no a través de una jerarquía navegable explícita.

Como en todos los gráficos del panel, se utiliza la dimensión temporal “Período de Fecha” para contextualizar los resultados y mantener coherencia temporal entre las distintas visualizaciones. La Figura 50 muestra además la configuración del mapa de calor, donde se observan las dimensiones y la medida (métrica) seleccionadas para su construcción.

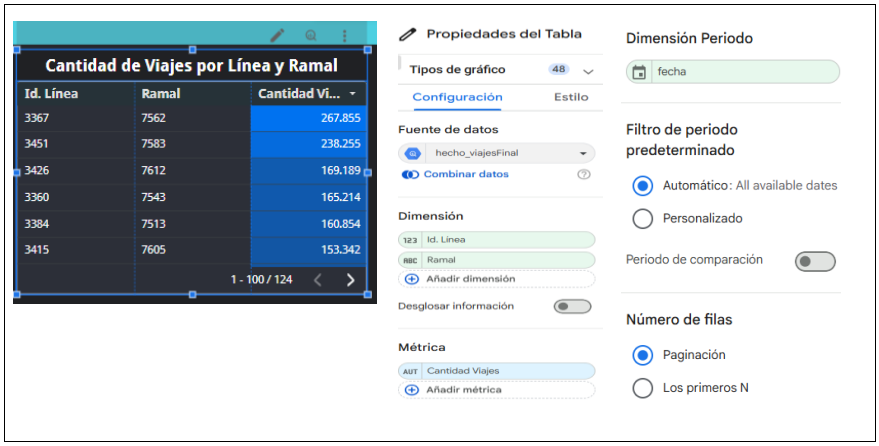


Figura 50: Detalle y configuración de mapas de calor en Looker Studio.

Indicadores de Resultados

También se incorporaron indicadores presentados a través de cuadros de resultados, diseñados para resumir métricas clave de manera inmediata y destacada, complementados con un minigráfico (sparkline) que muestra su evolución temporal. Estos elementos son particularmente útiles para resaltar indicadores estratégicos vinculados al desempeño operativo.

Concretamente, tal como lo muestra la Figura 51, se implementó un cuadro de resultados que muestra la Tasa de Trasbordos (expresado como porcentaje). Su formulación corresponde al cociente entre la cantidad de viajes con identificador de combinación distinto de nulo y el total de viajes registrados:

$$Tasa\ de\ Transbordos\ = \frac{Viajes\ con\ Id\_combinación \neq NULL}{Total\ de\ viajes}$$

Para su visualización, se utiliza la dimensión temporal “fecha”, que alimenta el minigráfico incorporado en el cuadro de resultados y permite observar la evolución del indicador a lo largo del período seleccionado.

Este componente cumple una función sintética dentro del tablero: permite conocer de manera inmediata el valor actual del indicador y, simultáneamente, analizar su comportamiento temporal.

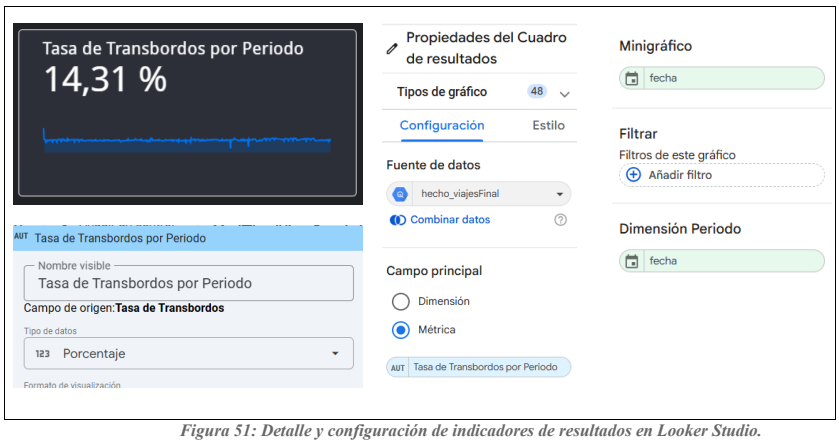


Figura 51: Detalle y configuración de indicadores de resultados en Looker Studio.

Campos Calculados Personalizados

Además de los campos calculados mencionados anteriormente, tal como la Tasa de Trasbordos, se incorporaron otros campos calculados personalizados orientados a la transformación y clasificación de dimensiones. Estos fueron definidos directamente en Looker Studio, sin modificar la tabla de origen almacenada en BigQuery.

Estos campos son expresiones creadas mediante fórmulas lógicas que permiten generar nuevas variables derivadas, tanto numéricas como textuales. Su aplicación fue esencial para mejorar la legibilidad y comprensión de los datos, ya que muchos campos originales provenían del sistema transaccional con valores codificados o numéricos.

Por ejemplo, algunos campos como el Código de Contrato o el Tipo de Transacción contenían valores enteros que no resultaban fácilmente interpretables. Para mejorar su presentación, se creó un nuevo campo derivado que convierte esos valores en etiquetas textuales descriptivas (por ejemplo, “Boleto Común” o “Trasbordo”), mediante el uso de una fórmula condicional basada en la función CASE WHEN.

La Figura 52 muestra el detalle de su configuración:

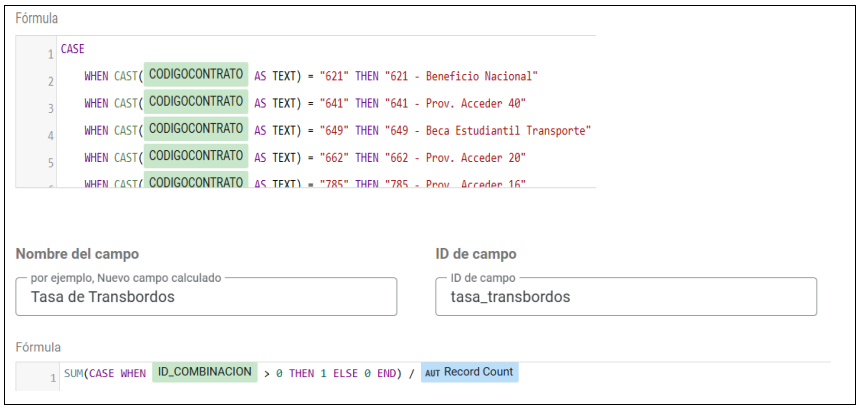


Figura 52: Detalle y configuración de campos personalizados en Looker Studio.

Además de mejorar la presentación visual, los campos calculados amplían las posibilidades analíticas, ya que pueden utilizarse como dimensiones en gráficos, integrarse en filtros de control y combinarse con otras métricas. Esto permite segmentar la información de manera más comprensible y alineada con la terminología operativa del dominio propio de la STyT.

3.3. Presentación y Análisis del Panel de Control Desarrollado

Esta sección describe y muestra tal como las ve el usuario final, las distintas páginas que componen el panel de control, presentando los principales indicadores y su aporte al análisis del sistema de transporte público. El objetivo es mostrar cómo la información se organiza en distintos enfoques temáticos y cómo cada conjunto de visualizaciones contribuye a una comprensión integral del comportamiento operativo y económico del sistema.

Cada página aborda un aspecto específico del análisis, como por ejemplo, movilidad general, trasbordos, distribución territorial, perfil de usuario, desempeño de entidades y dimensión económica; manteniendo coherencia en el uso de filtros y en la selección temporal. La navegación permite explorar el mismo conjunto de datos desde diferentes perspectivas analíticas, conservando consistencia en los criterios de segmentación y agregación.

*3.3.1. Página **Indicadores** de Viajes*

La página “Indicadores de Viajes” ofrece una caracterización integral del volumen y los patrones de movilidad registrados en el período seleccionado, como puede observarse en la Figura 53.

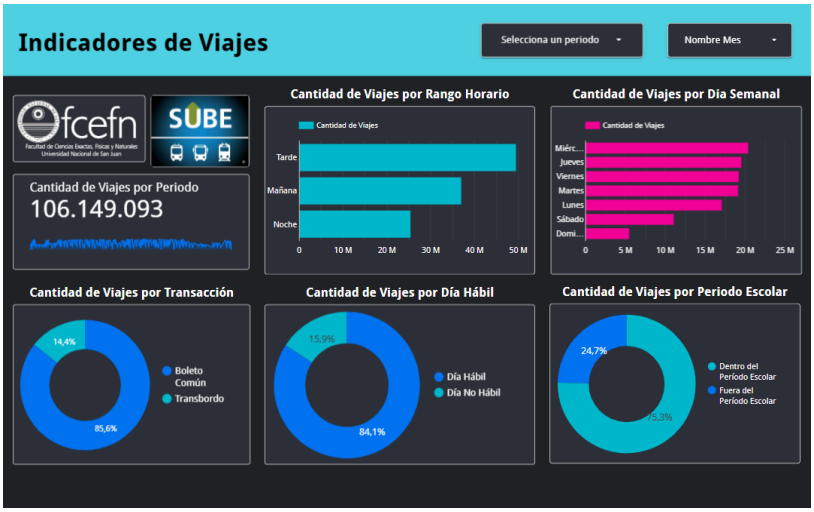


Figura 53: Indicadores de Viajes.

En la parte superior se disponen los filtros temporales, tales como período y mes, que permiten contextualizar todos los indicadores dentro del mismo marco temporal, garantizando consistencia en la interpretación de los resultados.

El indicador principal presenta la cantidad total de viajes del período, que en el caso ilustrado asciende a 106.149.093. Este valor sintetiza la magnitud global de la demanda y constituye la base cuantitativa sobre la cual se estructuran las demás segmentaciones.

El análisis por rango horario muestra cómo se distribuye la demanda a lo largo del día. En el panel se observa que la franja de la tarde concentra la mayor cantidad de viajes, seguida por la mañana y, finalmente, la noche. Esta configuración resulta coherente con dinámicas habituales de movilidad urbana, en las que los intervalos asociados al inicio y finalización de actividades concentran mayores flujos de desplazamiento.

En cuanto a la distribución por día de la semana, se observa una mayor intensidad de uso del transporte público de pasajeros en las jornadas intermedias, particularmente miércoles y jueves,

y una disminución marcada el día domingo. Este comportamiento refleja variaciones sistemáticas en la demanda a lo largo del ciclo semanal.

La segmentación por tipo de transacción indica que la mayoría de los viajes corresponde a boleto común, mientras que una proporción menor implica trasbordos. Esta relación pone de manifiesto que el desplazamiento directo constituye la modalidad predominante, aunque los viajes con combinación representan un componente operativo relevante dentro del sistema.

El eje de análisis que refleja si el viaje corresponde a un día hábil o no, permite visualizar una concentración ampliamente mayoritaria de viajes en jornadas laborables. Este resultado confirma la estrecha vinculación entre la demanda del servicio y las actividades laborales de la población usuaria.

Finalmente, la comparación entre período escolar y no escolar evidencia una mayor participación de viajes durante el ciclo lectivo, lo que sugiere una incidencia significativa del calendario escolar en el volumen total de desplazamientos.

*3.3.2. **Página** Indicadores de Trasbordos*

Esta página aborda el análisis de los viajes según una dimensión relevante para el sistema de transporte, los trasbordos (mostrada en la Figura 54).

Figura 54: Indicadores de Trasbordos.

En la parte superior se dispone el indicador de tasa de trasbordos por período, que expresa la proporción de viajes que requieren al menos una combinación respecto del total de viajes registrados. Este indicador sintetiza el peso relativo de los trasbordos dentro de la estructura general de movilidad.

La distribución por rango horario muestra que la mayor cantidad de trasbordos se concentra en la franja de la tarde, seguida por la mañana y, en menor medida, por la noche. Este comportamiento resulta consistente con la mayor intensidad general de viajes observada en esos intervalos, lo que sugiere que la necesidad de combinación se incrementa en los momentos de mayor demanda.

El análisis por día de la semana evidencia una mayor cantidad de trasbordos en los días intermedios de la semana (miércoles, jueves y viernes) y una reducción marcada durante el fin de semana. Esta evolución acompaña el patrón general de movilidad, reflejando que la actividad interlíneas se intensifica en días laborables.

Los indicadores por departamento y ciudad permiten identificar la distribución territorial de los trasbordos. Se observa una mayor concentración en el departamento Capital y en las ciudades de Concepción y el Gran San Juan, mientras que otras jurisdicciones presentan valores significativamente menores. Estos resultados sugieren que las áreas centrales del sistema concentran una parte relevante de las combinaciones, consistente con su rol como nodos de conexión dentro de la red.

Finalmente, la segmentación por período escolar muestra una mayor participación de trasbordos durante el ciclo lectivo en comparación con el período no escolar. Esta diferencia indica que la actividad educativa requiere fuertemente combinación entre líneas.

3.3.3. Pagina Indicadores de Geolocalización

Esta página (Figura 55) se centra en el análisis de los datos a través de la dimensión de geolocalización, permitiendo explorar cómo se distribuyen los hechos a lo largo del territorio provincial y detectar patrones espaciales relevantes.

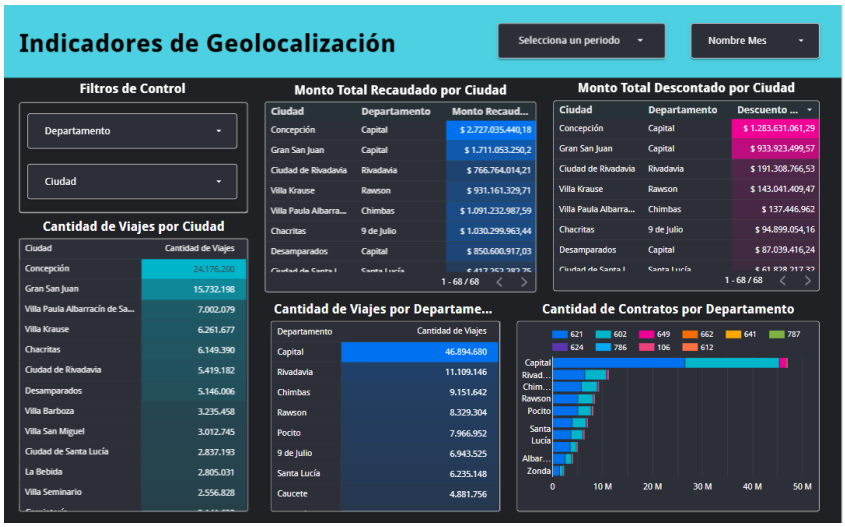


Figura 55: Indicadores de Geolocalización.

El indicador de monto total recaudado por ciudad y departamento muestra la concentración territorial de los ingresos generados por el sistema de transporte en el período seleccionado. Se observa que determinadas jurisdicciones concentran valores más elevados, lo que refleja su mayor participación relativa dentro del volumen total de viajes registrados.

El monto total descontado por ciudad y departamento complementa este análisis al exhibir la distribución geográfica de los beneficios tarifarios aplicados. En términos generales, las jurisdicciones con mayor nivel de recaudación presentan también mayores montos descontados, lo que indica que la aplicación de beneficios acompaña la intensidad de uso del servicio.

La cantidad de viajes por ciudad y por departamento permite identificar los principales focos de movilidad dentro del sistema. Se advierte una concentración significativa en algunas ciudades y departamentos, mientras que otras jurisdicciones presentan participaciones menores. Esta segmentación territorial facilita la comparación relativa de la demanda entre distintas zonas.

El indicador de cantidad de contratos por departamento incorpora la dimensión vinculada a los tipos de beneficios registrados. Se observa que los departamentos con mayor volumen de viajes concentran también mayor cantidad de contratos. Entre los tipos de contrato con mayor representación se encuentran el boleto común (código 602) y el beneficio nacional identificado como 621, los cuales reúnen una proporción significativa de los registros analizados.

En conjunto, esta página integra la dimensión territorial con variables de demanda y económicas, permitiendo visualizar la estructura espacial del sistema y la distribución geográfica de su actividad.

3.3.4. Página Indicadores de Perfil de Usuario

Esta página (Figura 56) permite el análisis del comportamiento económico asociado al uso del sistema, tomando como variable central el saldo disponible en las tarjetas.

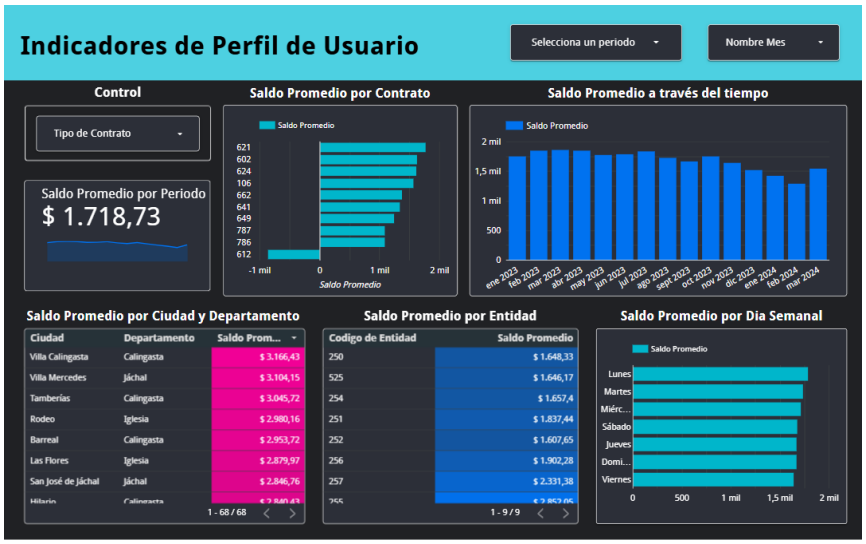


Figura 56: Indicadores de Perfil de Usuario.

El indicador de saldo promedio por período permite observar la evolución temporal del importe disponible en las tarjetas durante el intervalo seleccionado. La serie muestra variaciones moderadas a lo largo de los meses, lo que sugiere cierta estabilidad en los patrones de recarga y uso del sistema.

El saldo promedio por tipo de contrato permite identificar diferencias entre segmentos de usuarios. Se observa que los contratos correspondientes a boleto común y a determinados beneficios nacionales presentan saldos promedio más elevados, mientras que otros tipos registran valores menores e incluso negativos. Esta segmentación permite detectar comportamientos diferenciados en la gestión del crédito disponible.

El gráfico de saldo promedio a través del tiempo complementa el análisis anterior al visualizar la tendencia mensual desde 2023 hasta marzo de 2024. Las fluctuaciones observadas pueden asociarse a variaciones en la intensidad de uso del servicio o en los momentos de recarga, sin evidenciar cambios abruptos que indiquen alteraciones estructurales en el comportamiento general.

La distribución del saldo promedio por día semanal muestra diferencias acotadas entre jornadas, aunque se observa una leve tendencia a mayores saldos al inicio de la semana y menores valores hacia el final, lo cual podría vincularse con ciclos habituales de recarga y consumo.

Finalmente, la segmentación por ciudad, departamento y entidad permite comparar diferencias territoriales y empresariales. Los departamentos más alejados presentan, en algunos casos, saldos promedio superiores, mientras que otras jurisdicciones registran valores menores. Estas variaciones pueden estar asociadas a diferencias en patrones de movilidad, frecuencia de uso o características socioeconómicas de la población usuaria.

3.3.5. Página Indicadores de Entidades (Empresas)

En este caso, como se observa en la Figura 57, el foco está puesto en el análisis operativo desde la perspectiva de las empresas prestadoras del servicio.

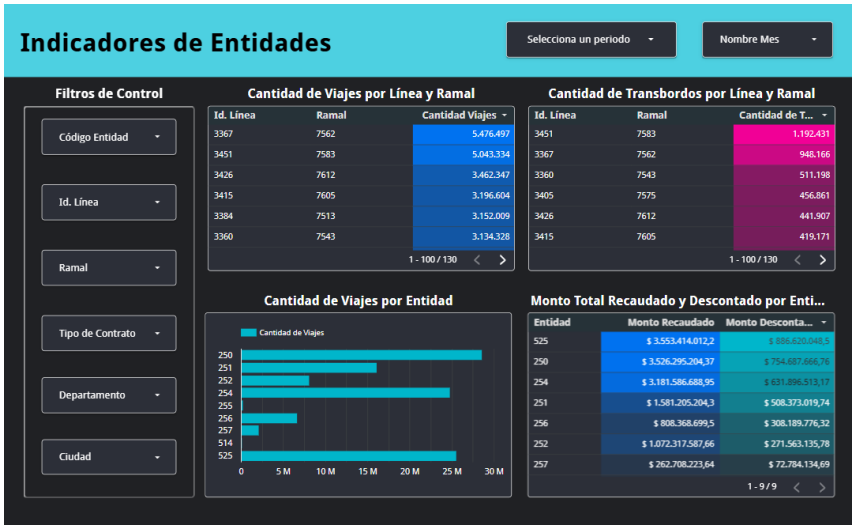


Figura 57: Indicadores de Entidades.

La cantidad de viajes por línea y ramal permite identificar los recorridos con mayor demanda dentro del período seleccionado. Las líneas que concentran mayor volumen de viajes evidencian su relevancia estructural dentro de la red de transporte, mientras que aquellas con menor participación pueden ser objeto de evaluación en términos de adecuación de oferta.

En cuanto a la cantidad de trasbordos por línea y ramal complementa este análisis, al mostrar qué recorridos participan con mayor frecuencia en combinaciones interlíneas. Un volumen elevado de trasbordos puede indicar nodos de conexión relevantes dentro del sistema.

El indicador de cantidad de viajes por entidad permite observar cómo se distribuye la demanda entre las distintas empresas operadoras. Esta información resulta útil para analizar la concentración del servicio y el peso relativo de cada entidad dentro del sistema de transporte.

Por último, el monto total recaudado y el monto total descontado por entidad permiten evaluar la dimensión económica asociada a cada empresa. La comparación entre ambos valores facilita el análisis del impacto de los beneficios tarifarios sobre los ingresos brutos de cada operador y aporta una visión integrada entre desempeño operativo y resultado económico.

3.3.6. Página Indicadores de Montos

Esta última página (Figura 58) aborda la dimensión económica del sistema, considerando valores monetarios ajustados conforme a índices oficiales, lo que permite realizar comparaciones homogéneas a lo largo del tiempo.

Figura 58: Indicadores de Montos.

El indicador de monto total recaudado por período ofrece una visión consolidada de los ingresos generados en el intervalo seleccionado. Este valor sintetiza el desempeño económico global del sistema para el período analizado.

El gráfico de monto total recaudado y descontado a través del tiempo permite observar la evolución mensual, diferenciando entre ingresos brutos y descuentos aplicados. La visualización facilita la identificación de variaciones temporales y la comparación proporcional entre ambos componentes.

El análisis por tipo de contrato muestra la distribución de montos recaudados y descontados según los distintos segmentos de usuarios. Esta desagregación permite identificar cuáles contratos concentran mayor volumen económico y cuáles presentan mayor peso relativo en términos de beneficios tarifarios.

El monto total recaudado por rango horario aproxima el análisis económico a la dinámica operativa diaria, mostrando qué franjas concentran mayor generación de ingresos. En términos generales, las franjas de mayor movilidad coinciden con mayores niveles de recaudación.

Finalmente, el monto total recaudado por día semanal permite observar la distribución de ingresos a lo largo de la semana. Se verifica una mayor concentración en jornadas laborables y

una disminución durante el fin de semana, en coherencia con el comportamiento general de la demanda.

Capítulo 4: Resultados, Conclusiones y Trabajo Futuro

El presente capítulo presenta los resultados alcanzados durante el desarrollo del proyecto, junto con las conclusiones y posibles líneas de trabajo futuro.

*4.1. **Resultados***

La construcción de la solución ABI permitió integrar Apache Spark como motor de procesamiento distribuido con servicios gestionados de Google Cloud Platform, en particular Cloud Storage, Dataproc, BigQuery y Looker Studio. La arquitectura desarrollada resolvió las etapas de extracción, transformación y carga de grandes volúmenes de datos transaccionales y georreferenciados del sistema de transporte público de la provincia de San Juan.

Durante el procesamiento se trabajó en la organización y preparación de la información dentro de Spark, dejándola lista para su carga en el entorno analítico. En BigQuery, el modelo implementado facilitó la construcción de consultas destinadas a la generación de indicadores y visualizaciones dinámicas.

A partir del análisis se identificaron patrones espaciales y temporales acordes con la dinámica de movilidad provincial, con concentraciones de demanda en zonas urbanas centrales —especialmente en el Gran San Juan— y diferencias entre días hábiles y fines de semana. Asimismo, el estudio de los hábitos de recarga y del saldo promedio evidenció comportamientos diferenciados entre usuarios, lo que permite profundizar en el análisis de las condiciones de acceso al servicio en distintos sectores.

Además, los indicadores vinculados a los montos recaudados y el volumen de viajes ofrecieron una visión integrada del funcionamiento operativo del sistema. Esta información pudo explorarse a través del panel de control, mediante filtros temporales, geográficos y operativos, lo que evidenció la coherencia entre el diseño del almacén de datos y las necesidades de análisis.

*4.2. **Conclusiones***

El desarrollo de esta tesis permitió cumplir con el objetivo planteado: se diseñó e implementó una solución ABI basada en Apache Spark capaz de integrar, procesar y visualizar datos del transporte público provincial, constituyendo una herramienta valiosa para la toma de decisiones. La arquitectura propuesta mostró un funcionamiento consistente, articulando el

procesamiento distribuido con el almacenamiento analítico en la nube, adecuado para las características del dominio de aplicación.

El proceso de construcción implicó profundizar en el tratamiento de grandes volúmenes de datos, analizar alternativas tecnológicas y poner en práctica un flujo completo de analítica, desde el proceso ETL hasta la visualización de la información. Esto permitió evidenciar que una solución ABI no se limita simplemente al desarrollo de paneles de control, sino que requiere una arquitectura robusta que asegure la integridad, coherencia y fiabilidad de los datos a lo largo de todo el proceso, desde los datos crudos hasta la información consolidada.

Por otra parte, desde el punto de vista académico, el alumno se enfrentó a conceptos y tecnologías poco abordados durante la carrera, ampliando sus conocimientos y familiarizándose con herramientas de Big Data y Cloud Computing, valiosas para su futuro desarrollo profesional.

*4.3. **Trabajo Futuro***

Como línea de mejora, una implementación futura podría ampliar la capacidad de procesamiento y almacenamiento más allá de las limitaciones del período de prueba de GCP. Esto permitiría trabajar con un clúster Dataproc de mayor capacidad, incorporar más períodos históricos y aumentar la frecuencia de actualización de la información.

Otra mejora importante sería la incorporación de un esquema de ingesta continua de datos. Esto implicaría automatizar la incorporación de los datos transaccionales y de fuentes externas (como los índices de ajuste del INDEC) mediante mecanismos de procesamiento en streaming orientados principalmente a la carga y consolidación de los datos. De este modo, la información se mantendría actualizada sin depender de cargas manuales.

Estas implementaciones habilitarían el análisis de series temporales a largo plazo y la identificación de patrones estructurales de movilidad que en este trabajo no pudieron ser contemplados.

Por último, sería valioso incorporar también mecanismos orientados a la gobernanza de los datos, entendida como prácticas de control de calidad, trazabilidad y seguimiento de la información. Podrían incluir validaciones automáticas durante los procesos ETL, detección de inconsistencias y registro de las transformaciones realizadas. La adopción de estas prácticas

permitiría contar con datos más confiables y consistentes, fortaleciendo la arquitectura y la calidad de la información disponible para análisis.

5. Referencias Bibliográficas

[1] Chen, H., Chiang, R. H. L., & Storey, V. C. (2012). *Business intelligence and analytics: From big data to big impact.* MIS Quarterly, 36(4), 1165–1188.

[2] Sharda, R., Delen, D., & Turban, E. (2020). *Analytics, data science, & artificial intelligence: Systems for decision support* (11th ed.). Pearson.

[3] OECD. (2015). *Data-driven innovation: Big data for growth and well-being.* OECD Publishing. Available: https://doi.org/10.1787/9789264229358-en

[4] Batini, C., Cappiello, C., Francalanci, C., & Maurino, A. (2009). *Methodologies for data quality assessment and improvement.* ACM Computing Surveys, 41(3), 1–52. Available: https://doi.org/10.1145/1541880.1541883

[5] Argentina.gob.ar. (2024). *SUBE*. Available: https://www.argentina.gob.ar/sube

[6] Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., … Stoica, I. (2016). *Apache Spark: A unified engine for big data processing.* Communications of the ACM, 59(11), 56–65. Available: https://doi.org/10.1145/2934664

[7] Welch, T. F., & Widita, A. (2019). *Big data in public transportation: A review of sources and methods*. *Transport Reviews, 39*(6), 795–818. Available: https://doi.org/10.1080/01441647.2019.1616849

[8] Ferreira, R. (2020). Mobility analytics: plan and improve public transportation services though a business intelligence model and dashboards. Run.unl.pt. Available: http://hdl.handle.net/10362/104278.

[9] Bańka, M., Daniłowski, J., Czerliński, M., Murawski, J., Żochowska, R., & Sobota, A. (2022). A Feedback Analysis Automation Using Business Intelligence Technology in Companies Organizing Urban Public Transport. Sustainability, 14(18), 11740. Available: https://doi.org/10.3390/su141811740.

[10] Martínez Soler, M. (2025). Desarrollo de una herramienta BigData para el análisis de Chicago Open Data. Rua.ua.es. Available: https://rua.ua.es/dspace/handle/10045/145619.

[11] Gartner. (2019). Analytics And Business Intelligence (abi). Gartner. Available: ttps://www.gartner.com/en/information-technology/glossary/business-intelligence-bi.

[12] Kimball, R., & Ross, M. (1996). *The data warehouse toolkit: Practical techniques for building dimensional data warehouses.* John Wiley & Sons.

[13] Inmon, W. H. (2005). *Building the data warehouse* (4th ed.). John Wiley & Sons.

[14] Kimball, R., Ross, M., Thornthwaite, W., Mundy, J., & Becker, B. (2013). *The data warehouse toolkit: The definitive guide to dimensional modeling* (3rd ed.). John Wiley & Sons.

[15] Vaisman, A., & Zimányi, E. (2014). *Data warehouse systems: Design and implementation* (2nd ed.). Springer.

[16] Ponniah, P. (n.d.). *Data warehousing fundamentals: A comprehensive guide for IT professionals*. Wiley. Available:https://anuradhasrinivas.wordpress.com/wp-content/uploads/2013/03/data-warehousing-fundamentals-by-paulraj-ponniah.pdf

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

[18] García, R. G. (2025). *Desarrollo de una herramienta Big Data para el análisis de Chicago Open Data*. Repositorio Institucional RUA. Available: https://rua.ua.es/dspace/handle/10045/145619

[19] Macías et al. (2016) presentan los fundamentos de Apache Spark, describiendo su arquitectura y los principales componentes para el procesamiento distribuido de datos. Available: https://elibro.net/es/ereader/bibliounsj/58454

[20] Curto, J. (2020). *Introducción al business intelligence y big data: Generando valor a partir de los datos*. Universitat Oberta de Catalunya. Available: https://openaccess.uoc.edu/server/api/core/bitstreams/610864ce-0af3-4816-a83f-2735e527f312/content

[21] KeepCoding Bootcamps. (2022, June 7). *¿Cómo es la arquitectura de Apache Spark?* Available: https://keepcoding.io/blog/arquitectura-apache-spark/

[22] KeepCoding Bootcamps. (2025, November 20). *¿Qué son los RDD o Resilient Distributed Datasets?* Available: https://keepcoding.io/blog/rdd-resilient-distributed-datasets/

[23] Scaibu. (2024, October 16). *Spark partitioning: Unlocking big data performance*. Medium. Available: https://scaibu.medium.com/spark-partitioning-unlocking-big-data-performance-e08f6891135d

[24] Aronovich, D., & Mizrachi, E. (2025, February 26). Understanding Apache Spark Shuffle for Better Performance. Substack.com; Big Data Performance Weekly. Available: https://bigdataperformance.substack.com/p/understanding-apache-spark-shuffle

[25] Canadian Data Guy. (2025, May 14). Spark Join Strategies Explained: Broadcast Hash Join [Review of Spark Join Strategies Explained: Broadcast Hash Join]. Canadian Data Guy. Available: https://www.canadiandataguy.com/p/spark-join-strategies-explained-broadcast.

[26] Kayalvizhi. (2024, November 9). Spark Optimization for a large datasets: Medium. Available: https://medium.com/@vtrkayalrajan/spark-optimization-for-a-large-datasets-8a045986660e.

[27] IBM Developer. (2025). Ibm.com. Available: https://developer.ibm.com/blogs/spark-performance-optimization-guidelines/

[28] Pepperdata. (2025, March 12). *Why is Spark so slow? 5 ways to optimize Spark*. Available: https://www.pepperdata.com/blog/why-is-spark-so-slow

[29] Databricks. (2018, May 18). *What is the Catalyst Optimizer?* Available: https://www.databricks.com/glossary/catalyst-optimizer

[30] Stack Overflow. (n.d.). *What is the concept of application, job, stage and task in Spark?* Available: https://stackoverflow.com/questions/42263270/what-is-the-concept-of-application-job-stage-and-task-in-spark

[31] Google Cloud. (2024). *Apache Spark y Hadoop gestionados con Google Dataproc*. Available: https://cloud.google.com/dataproc?hl=es

[32] Google Cloud. (2025). *Acerca de los buckets de Cloud Storage*. Available: https://cloud.google.com/storage/docs/buckets?hl=es-419

[33] Google Cloud Documentation. (2025b). *Descripción general de BigQuery*. Available: https://docs.cloud.google.com/bigquery/docs/introduction?hl=es-419

[34] Google Cloud Documentation. (2025c). *Te damos la bienvenida a Looker Studio*. Available: https://docs.cloud.google.com/looker/docs/studio?hl=es

[35] Gartner. (2022). Business Intelligence (BI) Tools Reviews 2020 | Gartner Peer Insights. Gartner. Available: https://www.gartner.com/reviews/market/analytics-business-intelligence-platforms.

[36] Google Cloud Documentation. (2025d). Free cloud features and trial offer: BigQuery. Available: https://docs.cloud.google.com/free/docs/free-cloud-features#bigquery

[37] OpenStreetMap contributors. (2025). *OpenStreetMap*. Available: https://www.openstreetmap.org

[38] Brinkhoff, T. (2022). *San Juan (Argentina): Localidades y departamentos: Estadísticas de población, gráficos y mapas*. City Population. Available: https://www.citypopulation.de/es/argentina/sanjuan/

[39] Google Cloud Documentation. (2025d). Free cloud features and trial offer. Available: https://docs.cloud.google.com/free/docs/free-cloud-features?_gl=1

[40] Google Cloud Documentation. (2025e). Descripción general de las cuentas de servicio. Available: https://cloud.google.com/iam/docs/service-account-overview?hl=es-419

[41] Google Cloud Documentation. (2025f). Reglas de firewall de VPC. Available: https://cloud.google.com/firewall/docs/firewalls?hl=es-419

[42] Rawat, D. (2024, January 5). *A step-by-step guide to installing PySpark on Windows*. Medium. Available: https://medium.com/@deepaksrawat1906/a-step-by-step-guide-to-installing-pyspark-on-windows-3589f0139a30

[43] Nelamali, N. (2024, January 7). *Spark | PySpark versions supportability matrix - Spark by {Examples}*. Spark by {Examples}. Available: https://sparkbyexamples.com/spark/spark-versions-supportability-matrix/

[44] INDEC: Instituto Nacional de Estadística y Censos de la República Argentina. (2025). *Www.indec.gob.ar*. Available: https://www.indec.gob.ar/indec/web/Nivel4-Tema-3-5-31

6. Apéndice

El apéndice tiene como propósito documentar detalladamente apartados técnicos y adicionales que, si bien son fundamentales para la realización del proyecto, exceden el alcance del análisis principal en las secciones de desarrollo y resultados. En el mismo se detalla el proceso de instalación de Apache Spark en un entorno de trabajo local, además de cómo se obtiene y genera el archivo csv de índices inflacionarios utilizados durante el proceso ETL.

6.1. Proceso de configuración de Google Cloud Platform como entorno basado en la nube

El presente trabajo de investigación propone el desarrollo de una solución ABI utilizando Google Cloud Platform (GCP) como entorno principal de procesamiento y análisis de datos. La elección de una plataforma basada en la nube responde a la necesidad de contar con un ecosistema escalable, rápido de configurar y altamente disponible, que permita ejecutar componentes críticos del proceso analítico, tales como almacenamiento distribuido, procesamiento masivo de datos y visualización avanzada.

Para construir esta solución utilizando los servicios Google Cloud Storage (GCS), Dataproc, BigQuery y Looker Studio, es fundamental realizar una configuración inicial adecuada de la plataforma. Esta configuración garantiza el acceso correcto a los recursos, habilita las APIs necesarias y establece los permisos y reglas de seguridad para operar sin inconvenientes.

A continuación, se detalla el proceso completo para preparar GCP como entorno cloud del proyecto:

Creación de una cuenta y proyecto

El punto de partida consiste en disponer de una cuenta de Google. Es posible crear una nueva cuenta o iniciar sesión con una existente.

Una vez dentro de la consola de GCP, se procede a crear un proyecto, que funciona como contenedor lógico de todos los recursos de la solución; en este caso, el proyecto se denomina project-sube.

Este proyecto es el espacio donde se almacenarán los buckets de GCS, los clústeres de Dataproc, los datasets de BigQuery y las fuentes conectadas a Looker Studio.

Acceso a la prueba gratuita y configuración de facturación

Google Cloud ofrece una prueba gratuita de USD $300 con una duración de 90 días, que habilita el uso de más de 20 productos y servicios. Esta prueba es especialmente útil en proyectos académicos o pilotos de investigación, ya que permite experimentar con servicios avanzados sin generar costos iniciales [39].

Para activar la prueba solo se requiere registrar una tarjeta de débito o crédito válida, utilizada exclusivamente para validar la identidad. En la práctica, si el consumo no supera los USD 300 o el período de 90 días, Google no realiza ningún cargo.

La activación se realiza desde: Facturación → Administrar cuenta de facturación → Habilitar facturación. La siguiente Figura 59 muestra un pantallazo de la descripción de costos.

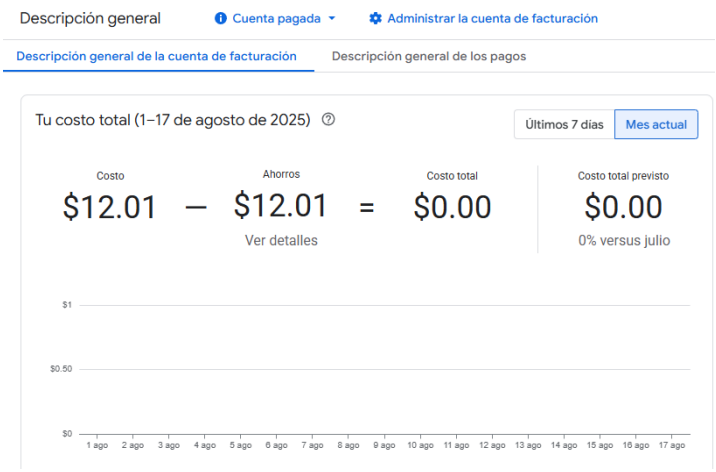


Figura 59: Detalle de facturación con el periodo de prueba gratuito.

Es importante tener presente que, al alcanzar el límite de crédito o los 90 días, todo servicio en ejecución comienza a generar cargos. Por eso para evitar costos posteriores, es posible deshabilitar la cuenta de facturación desde: Facturación → Configuración de pago → Administrador de cuentas → Inhabilitar cuenta de facturación.

Esto detiene todos los recursos asociados y previene nuevos cargos. Los costos pendientes se procesarán únicamente el primer día hábil del mes siguiente.

Habilitación de APIs

Los servicios utilizados en este proyecto (Figura 60), requieren la habilitación de diversas APIs. Si bien muchos de estos servicios se activan automáticamente al usarlos por primera vez, es conveniente verificar manualmente que todas las APIs críticas están habilitadas.

Figura 60: Servicios utilizados desde el ecosistema de Google Cloud Platform.

Las APIs relevantes para esta solución incluyen:

- **Cloud Resource Manager API** para la administración del proyecto y sus permisos.
- **Cloud Storage API** para almacenamiento estándar de bucket.
- **Dataproc API** para la creación y gestión semiautomática del clúster con Hadoop y Spark.
- **Compute Engine API** para las instancias de máquinas virtuales que componen Dataproc.
- **BigQuery API** y **BigQuery Storage API** para la gestión y carga de datos en el data warehouse.

Las activaciones pueden gestionarse desde: APIs y Servicios → APIs y servicios habilitados.

La mayoría de APIs ya están habilitadas por defecto, pero son necesarias habilitar manualmente: Dataproc API, Cloud Resource Manager API, BigQuery API y Cloud Storage API.

Este paso es fundamental para asegurar que todos los componentes de la solución funcionen correctamente sin interrupciones.

Cuenta de Servicio y asignación de roles

Una cuenta de servicio en GCP es una entidad principal del proyecto que administra el acceso a recursos y servicios de la plataforma. Es utilizada para asignar y gestionar roles en los diferentes servicios objetivos.

Esto significa que para trabajar con servicios específicos, es necesario que la cuenta de servicio cumpla con ciertos roles, de manera que se garantice un comportamiento adecuado y controlado.

Para ver información de la cuenta de servicio: IAM y administración → Cuentas de Servicio [40]. Más detalle en la siguiente Figura 61:

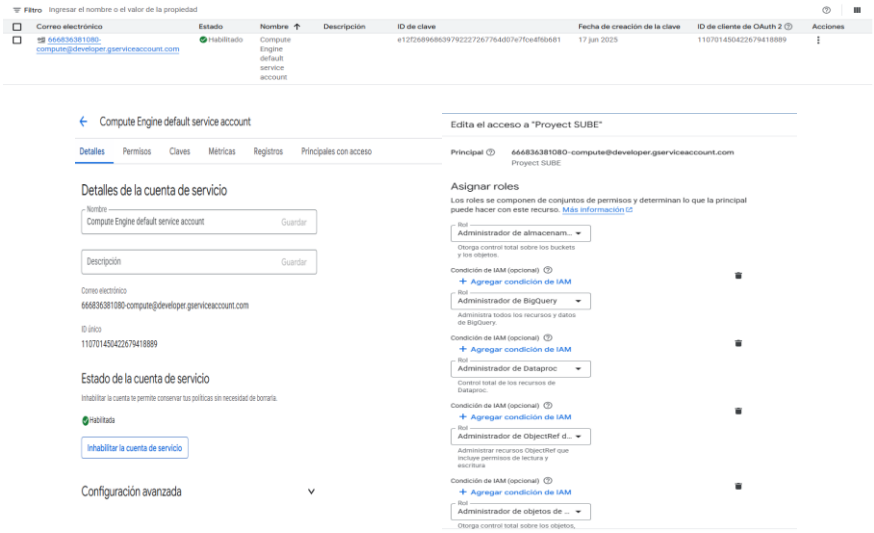


Figura 61: Configuración y roles asignados desde la cuenta de servicio.

Los roles mínimos recomendados para los servicios del proyecto incluyen:

- **General:** Editor (permite crear, modificar y eliminar recursos del proyecto).
- **Cloud Storage:** Administrador de almacenamiento, Administrador de objetos de Storage y Visualizador de objetos de almacenamiento.

- **Dataproc:** Administrador de Dataproc, Editor de Dataproc y Trabajador de Dataproc.
- **BigQuery:** Administrador de BigQuery, Administrador de ObjectRef de BigQuery, Editor de datos de BigQuery, Propietario de datos de BigQuery y Usuario de trabajo de BigQuery.

Reglas de Firewall

La creación de un clúster Dataproc implica el despliegue de varias máquinas virtuales dentro de una red virtual privada (VPC) que gestiona GCP. Para que estas instancias puedan comunicarse entre sí y con servicios externos como BigQuery, deben configurarse reglas de firewall que controlen el tráfico entrante y saliente por medio de protocolos y puertos específicos.

A continuación se detalla precisamente qué tipos de reglas se definieron para el cluster: Red VPC → Firewall [41]. La siguiente Figura 62 muestra cómo debe hacerse la configuración:

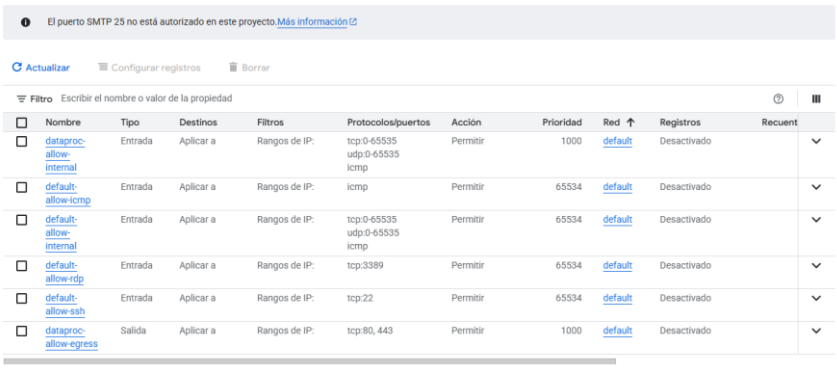


Figura 62: Configuración de las reglas de firewall.

Reglas Entrantes:

- dataproc-allow-internal (creada manualmente): Rango de IP 10.128.0.0/9; Protocolos tcp:0-65535, udp:0-65535, icmp; Red default.
- default-allow-icmp.
- default-allow-internal.
- default-allow-rdp.

- default-allow-ssh.

Regla Saliente:

- dataproc-allow-egress (creada manualmente): Rango de IP 0.0.0.0/0; Protocolo tcp:80 y 443; Red default.

Todas las reglas de firewall tienen como destino “Todas las instancias de la red”.

Las únicas reglas creadas manualmente son las mencionadas, pues era necesario establecer una regla de salida para la conexión a BigQuery y descarga de dependencias pip. El resto de reglas fueron únicamente ajustadas para restringir el acceso según el rango de IP utilizado.

6.2. Proceso de Instalación de Apache Spark en entorno local

La solución ABI basada en Apache Spark mediante PySpark también ofrece una alternativa completamente local, permitiendo ejecutar el procesamiento en una computadora personal sin depender de servicios de la nube. Esta modalidad garantiza que los datos se mantengan exclusivamente en el equipo del usuario, evitando cualquier transferencia a sistemas externos y reduciendo la exposición a riesgos de seguridad o costos asociados.

Sin embargo, operar en un entorno local implica una serie de desafíos:

- **Instalación manual de componentes:** Spark no funciona de manera autónoma: requiere la instalación de Java Development Kit (JDK), Hadoop, variables de entorno configuradas correctamente, y en algunos casos incluso herramientas adicionales como Python virtual environments. Esto aumenta la complejidad del setup inicial.
- **Recursos limitados de hardware:** El rendimiento del procesamiento depende exclusivamente del equipo personal (CPU, RAM y almacenamiento). Esto puede ser un inconveniente significativo si se trabaja con volúmenes de datos grandes, dado que Spark está diseñado para operar de forma distribuida.
- **Mantenimiento y actualización manual:** Cada componente requiere mantenimiento, control de versiones y compatibilidad entre herramientas. Cualquier inconsistencia, por ejemplo, entre la versión de Spark y Hadoop, puede generar errores difíciles de diagnosticar.

- **Mayor curva de aprendizaje:** Se debe comprender a detalle cómo funciona la ejecución distribuida, los archivos de configuración y la estructura interna del sistema Hadoop/Spark.

En pocas palabras, optar por la opción de GCP es más adecuada para proyectos reales, escalables o colaborativos. La opción de Spark local es ideal para pruebas, entornos educativos o análisis a pequeña escala.

Configuración del entorno local

Lo primero que debe configurarse es la Máquina Virtual de Java (JVM) y el runtime de Python. Se descarga el JDK como requisito fundamental porque el core de Apache Spark está escrito en Scala y se ejecuta sobre la JVM, siendo esta la base sobre la que PySpark se comunica con los clusters o el modo local. Luego se continúa con la instalación de Python y la descarga de la librería PySpark desde el entorno pip. Esto permite interactuar con la funcionalidad de Spark simulando un entorno distribuido para el procesamiento de los datos, por medio de recursos como procesador y memoria RAM [42].

Se detalla la serie de pasos para la configuración correcta:

1. **Instalación del JDK**

   Lo primero a instalar para ejecutar Spark es el Java Development Kit. Es posible descargarlo desde el sitio oficial de Oracle o OpenJDK. En particular, se descarga la versión **JDK 8** la cual no presentó problemas para la ejecución de Spark por medio de su compatibilidad [43]. Es posible verificar la instalación entrando a la terminal y escribiendo “java -version”.

2. **Descargar y descomprimir Apache Spark**

   Se descarga la versión precompilada de Spark con Hadoop, específicamente las versión Spark 3.5.4 con Hadoop 3. Luego se descomprime el archivo descargado en una ubicación determinada como C:\Spark.

3. **Descargar y configurar winutils.exe**

   Se debe descargar un archivo binario necesario para Hadoop con Windows llamado winutils. El mismo se descarga desde un repositorio de GitHub con la versión específica de Hadoop:

   https://github.com/steveloughran/winutils/blob/master/hadoop-3.0.0/bin/winutils.exe

4. **Descargar e instalar Python**

   Se descarga Python 3.9 desde su página oficial, siguiendo las instrucciones para su instalación. Desde la terminal escribiendo el comando “python -v” o “python --version” es posible chequear la instalación correcta.

5. **Instalar PySpark y findspark**

   Desde la terminal por medio del entorno pip de python se ejecuta el siguiente comando: pip install pyspark. Se descargan todas las dependencias requeridas para utilizar PySpark. Es posible verificar la instalación por medio del comando: pyspark, el cual muestra un mensaje informativo.

   El segundo comando de pip findspark es recomendable para importar a Spark como una librería estándar al programa de Python que se esté desarrollando.

6. **Configurar las variables de entorno del sistema**
   - Variable JAVA_HOME con valor a la ruta de instalación de java, por ej. C:\java
   - SPARK_HOME que apunta al directorio donde se descomprime Spark.
   - HADOOP_HOME con valor a la ruta de ubicación de winutils, por ej. C:\winutils
   - Se configura la variable PYTHON_PARK con valores “C:\spark-3.5.4-bin-hadoop3\python” y “C:\spark-3.5.4-bin-hadoop3\python\lib\py4j-0.10.9.7-src.zip”.
   - Finalmente se edita el PATH del sistema agregando al final: %JAVA_HOME%\bin %SPARK_HOME%\bin y %HADOOP_HOME%\bin

PySpark permite el desarrollo de programas en Python que no necesariamente se ejecutan de forma secuencial, debido a que Spark puede contener código que solo se ejecuta una sola vez y no puede ser nuevamente depurado. Debido a su gran capacidad de procesamiento y análisis, puede ser necesario depurar y visualizar los resultados que se van obteniendo cada cierto tiempo, haciendo que el trabajo de análisis de la información sea más comprensible y eficiente. Esto se logra mediante entornos interactivos como Jupyter Notebooks y Polyglot Notebooks que permiten combinar código, texto y visualizaciones en un solo archivo, facilitando el desarrollo y la exploración de datos. La flexibilidad de ejecutar celdas de código individualmente o en secuencia, junto con la capacidad de documentar el proceso, los convierte en herramientas indispensables para el análisis de datos a gran escala con Spark.

Para ello se recomienda instalar la extensión de Visual Studio Code denominada “Polyglot Notebooks”, que permite ejecutar bloques de código Pyspark.

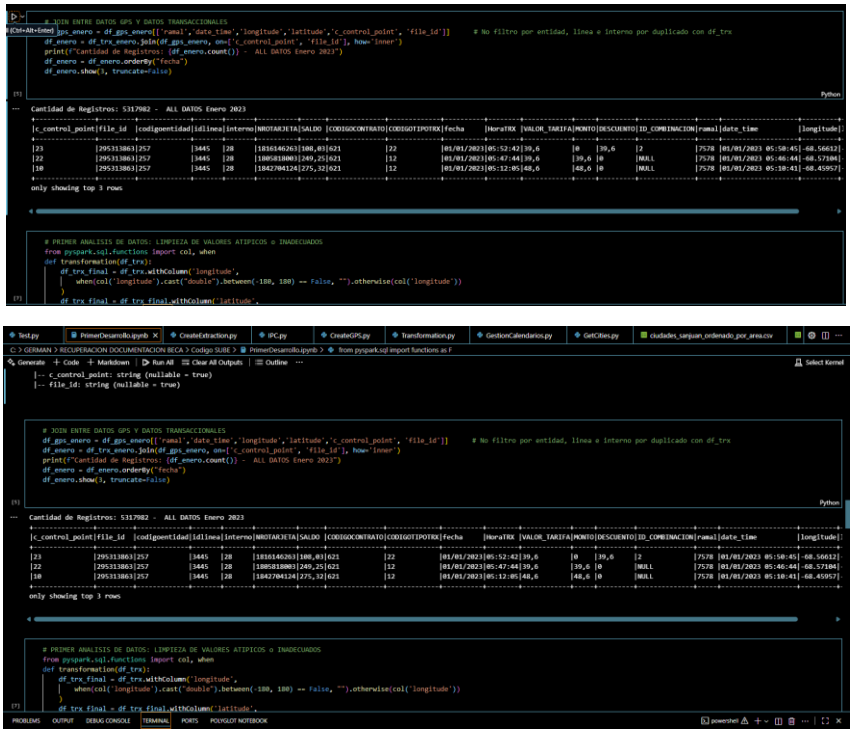


Figura 63: Código de desarrollo y pruebas de Pyspark usando Polyglot Notebooks.

En la Figura 63 se visualizan bloques de códigos que se fueron probando en las primeras etapas de aprendizaje y experimentación de Apache Spark utilizando Pyspark, aplicado a este contexto de transporte público con los datos suministrados por la STyT.

En esta etapa el código fue segmentado en celdas o bloques,cada uno con lógica particular y autocontenida. A pesar de la segmentación, todas las celdas están interrelacionadas y forman parte del proceso integral de desarrollo y solución final.

Esta estructura facilita la visualización, la ejecución interactiva y la documentación progresiva del flujo de trabajo de PySpark, desde la ingesta de datos hasta el análisis.

6.3. Construcción del archivo de Índices inflacionarios Indec

Un componente esencial para el análisis de Inteligencia de Negocios aplicado al transporte público es la inclusión de los índices de precios al consumidor (IPC) correspondientes a la división "Transporte", publicados por el Instituto Nacional de Estadística y Censos (INDEC). Dado que la proyección del análisis abarca el presente año 2025 y los datos oficiales de inflación del INDEC están disponibles para períodos previos, la construcción del archivo de datos para el año proyectado se realizó mediante un proceso semimanual de recopilación y filtrado de la información.

Este procedimiento puede resultar laborioso pues se debe acceder a las series históricas del Indec, identificar las publicaciones más recientes y extraer únicamente la información relevante para el contexto de esta investigación. En este caso, se utilizó como fuente la publicación del 12 de noviembre de 2025, correspondiente a los datos del período octubre 2025.

Se detalla la metodología empleada para estimar los valores mensuales del IPC de Transporte para 2025. El proceso incluye la generación del archivo CSV que es utilizado posteriormente en los scripts de PySpark para el cálculo de montos ajustados en función de dichos índices. Este nivel de detalle se incluye con el propósito de ofrecer transparencia y trazabilidad sobre la naturaleza de los datos inflacionarios utilizados, los cuales constituyen una construcción hipotética necesaria para la simulación de escenarios BI.

Para la obtención de los datos base, el primer paso consiste en acceder al portal del INDEC y descargar el archivo CSV titulado: "Índices y variaciones porcentuales mensuales e interanuales según divisiones de la canasta, bienes y servicios, y clasificación de grupos. Diciembre de 2016-octubre de 2025" [44]**.**

Este archivo contiene información de los índices de precios al consumidor organizados por división, período temporal y región geográfica. Para estos fines, se filtra exclusivamente la información correspondiente a la división "Transporte" y la región "Cuyo".

Las columnas incluidas en el archivo original son:

Código del IPC, Descripción de la división, Clasificador, Período, Índices (tres columnas) y Región.

Para facilitar su manipulación y filtrado, el archivo CSV se convierte en una hoja de cálculo sobre la cual se aplica la función FILTRAR, que permite extraer un subconjunto de filas que cumplen con una o más condiciones lógicas.

La sintaxis de la función es la siguiente:

$$= \boldsymbol{FILTRAR}(\backslash\boldsymbol{text}\{\boldsymbol{matriz}\};\ \backslash\boldsymbol{text}\{\boldsymbol{incluir}\};\ [\backslash\boldsymbol{text}\{\boldsymbol{si}\backslash\_\boldsymbol{vacio}\}])$$

Donde los parámetros que la conforman son:

- **Matriz (obligatorio):** rango de datos que se desea filtrar.
- **Incluir (obligatorio):** matriz de condiciones lógicas (VERDADERO/FALSO) con la misma altura que la matriz de datos. Cada condición con valor verdadero VERDADERO indica que la fila correspondiente debe incluirse en el resultado.
- **si_vacio (opcional):** valor que se devuelve si ninguna fila cumple las condiciones (no se utiliza en este contexto).

Con ello, la función aplicada en este caso es:

=FILTRAR('C:\File-Path\[serie_ipc_divisiones.csv]serie_ipc_divisiones'!B2:HMAX;

('C:\File-Path\[serie_ipc_divisiones.csv]serie_ipc_divisiones'!B2:BMAX = "Transporte")*

('C:\File-Path\[serie_ipc_divisiones.csv]serie_ipc_divisiones'!H2:HMAX = "Cuyo"))

El primer parámetro especifica la ruta absoluta del archivo csv que contiene el rango completo de datos (desde la columna B, fila 2, hasta la columna H y la última fila con datos, llamada en este caso MAX). El segundo parámetro contiene dos condiciones combinadas con el operador lógico AND (representado por el asterisco *):

- La columna B debe contener el valor “Transporte”.
- La columna H debe contener el valor “Cuyo”.

Solo las filas que cumplan ambas condiciones son incluidas en el resultado final.

El resultado de la función Filtrar produce una matriz dinámica con los registros relevantes, la cual se guarda como un nuevo archivo CSV.

Este archivo constituye la fuente de datos utilizada por los scripts desarrollados en PySpark, donde se emplea para calcular los montos ajustados según los índices de inflación del sector transporte.

Con esta metodología se garantiza que los valores proyectados del IPC utilizados en los modelos ABI sean coherentes, trazables y reproducibles, facilitando su interpretación y validación dentro del contexto analítico del proyecto.