

ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

RAG aplicado a business intelligence para la gestión eficiente de insights dentro de una institución financiera ecuatoriana.

PROYECTO DE TITULACIÓN

Previo la obtención del Título de:

Magister en Ciencias de Datos

Presentado por:

David Alfonso Argüello Fiallos

Melanie Melissa Villalta Báez

GUAYAQUIL - ECUADOR

Año: 2024

DEDICATORIA

David Argüello:

A Dios, que me guía y me da fuerzas.

A Juan, que no me ha dejado solo ningún día.

A Rosa y Galo, que me han cuidado toda mi vida.

A Fanny, que me ha amado contra viento y marea, desde siempre y para siempre.

A Melanie, que es la razón de mis éxitos y mi soporte en mis fracasos.

Melanie Villalta:

A Dios, por ser el soporte para no rendirme en este camino.

A mis padres, por estar siempre acompañándome y alentándome en todo lo que me propongo.

A mis hermanos y sobrina, por su apoyo incondicional.

A David, por impulsarme a ser mejor.

AGRADECIMIENTOS

A la comunidad politécnica que nos ha acompañado en este grado, a los equipos de Analítica de Banco Guayaquil que fueron un soporte para este proyecto con ideas y recomendaciones, y a los docentes que nos han compartido su experiencia y conocimientos.

DECLARACIÓN EXPRESA

Nosotros, David Alfonso Argüello Fiallos y Melanie Melissa Villalta Báez, acordamos y reconocemos que: La titularidad de los derechos patrimoniales de autor (derechos de autor) del proyecto de graduación corresponderá al autor o autores, sin perjuicio de lo cual la ESPOL recibe en este acto una licencia gratuita de plazo indefinido para el uso no comercial y comercial de la obra con facultad de sublicenciar, incluyendo la autorización para su divulgación, así como para la creación y uso de obras derivadas. En el caso de usos comerciales se respetará el porcentaje de participación en beneficios que corresponda a favor del autor o autores. El o los estudiantes deberán procurar en cualquier caso de cesión de sus derechos patrimoniales incluir una cláusula en la cesión que proteja la vigencia de la licencia aquí concedida a la ESPOL.

La titularidad total y exclusiva sobre los derechos patrimoniales de patente de invención, modelo de utilidad, diseño industrial, secreto industrial, secreto empresarial, derechos patrimoniales de autor sobre software o información no divulgada que corresponda o pueda corresponder respecto de cualquier investigación, desarrollo tecnológico o invención realizada por nosotros durante el desarrollo del proyecto de graduación, pertenecerán de forma total, exclusiva e indivisible a la ESPOL, sin perjuicio del porcentaje que me/nos corresponda de los beneficios económicos que la ESPOL reciba por la explotación de mi/nuestra innovación, de ser el caso.

En los casos donde la Oficina de Transferencia de Resultados de Investigación (OTRI) de la ESPOL comunique a los autores que existe una innovación potencialmente patentable sobre los resultados del proyecto de graduación, no se realizará publicación o divulgación alguna, sin la autorización expresa y previa de la ESPOL.

Guayaquil, 22 de noviembre del 2024.

David Alfonso

Argüello Fiallos

Melanie Melissa

Villalta Báez

EVALUADORES

.....
PhD. Eduardo Cruz

PROFESOR TUTOR

.....
MsC. Allan Avendaño

PROFESOR REVISOR

RESUMEN

El proyecto "RAG aplicado a business intelligence para la gestión eficiente de insights dentro de una institución financiera ecuatoriana" busca desarrollar un sistema de inteligencia artificial basado en la Generación Aumentada por Recuperación (RAG) para optimizar la gestión de datos. El objetivo es permitir a los usuarios comerciales realizar consultas en lenguaje natural a través de un portal de autoservicio, reduciendo los tiempos de respuesta y la carga de trabajo del área de analítica. La solución se justifica por la necesidad de mejorar el acceso a la información y acelerar la toma de decisiones en la institución.

Para el desarrollo, se utilizó un backend con DuckDB y modelos de OpenAI que traducen las consultas en SQL, extrayendo la información relevante, la cual se presenta en una interfaz visual de Streamlit. Se realizaron pruebas empíricas para ajustar la precisión y el rendimiento del sistema.

Los resultados muestran una reducción de hasta el 98% en el tiempo de respuesta respecto al proceso tradicional y una satisfacción del 78% entre los usuarios, validando la eficacia de la solución. El proyecto concluye que la combinación de RAG y BI mejora significativamente la accesibilidad y eficiencia en la gestión de información.

Palabras clave: Generación Aumentada por Recuperación, Business Intelligence, Modelos de Lenguaje, Gestión de Información.

ABSTRACT

The project "RAG applied to business intelligence for the efficient management of insights in an Ecuadorian financial institution" aims to develop an AI system integrating Retrieval-Augmented Generation (RAG) with Business Intelligence tools. The goal is to enable commercial users to perform natural language queries through a self-service portal, reducing response times and workload for the analytics team. The solution addresses the need to improve information accessibility and decision-making processes.

The system uses a DuckDB backend and OpenAI models to translate queries into SQL statements and present the results through a Streamlit interface. Empirical testing was conducted to fine-tune system performance.

The results show a 98% reduction in response times compared to the traditional process and a 78% satisfaction rate among users, validating the solution's effectiveness. The project concludes that combining RAG with BI significantly enhances data accessibility and operational efficiency.

Keywords: *Retrieval-Augmented Generation, Business Intelligence, Large Language Models, Data Management.*

ÍNDICE GENERAL

EVALUADORES.....	5
RESUMEN.....	I
<i>ABSTRACT</i>	II
ÍNDICE GENERAL	III
ABREVIATURAS.....	VI
ÍNDICE DE FIGURAS.....	VII
ÍNDICE DE TABLAS.....	VIII
CAPÍTULO 1.....	9
1. Planteamiento del problema	9
1.1 Descripción del problema	9
1.2 Justificación del problema	10
1.3 Objetivos	12
1.3.1 Objetivo General.....	12
1.3.2 Objetivos Específicos.....	12
1.4 Metodología.....	12
1.5 Resultados esperados.....	13
1.6 Dataset.....	13
1.6.1 tb_consumos_fact.....	14
1.6.2 comercio_dim.....	14
1.6.3 tarjeta_dim	15
1.6.4 cliente_dim.....	15
CAPÍTULO 2.....	16
2. Estado del arte.....	16
2.1 Modelos de Lenguaje de gran escala (LLM)	16

2.1.1	Familias de LLMs.....	16
2.2	Modelo Retrieval-Augmented Generation (RAG)	18
2.2.1	Uso de RAG con LLM	19
2.3	Prompt Engineering.....	21
2.4	Estudios y artículos sobre la utilización de RAG como solución de negocio	22
2.4.1	Business-RAG: Information Extraction for Business Insights.....	22
2.4.2	Improving Retrieval for RAG based Question Answering Models on Financial Documents	23
2.4.3	Financial Report Chunking for Effective Retrieval Augmented Generation	24
2.5	Soluciones de Analítica y Aprendizaje modernas.....	25
2.5.1	Generación de Consultas SQL con OpenAI.....	25
2.5.2	Ejecución de Consultas en DuckDB	25
2.5.3	Visualización con Streamlit	25
2.5.4	SQL tradicional en bases de datos relacionales (MySQL, PostgreSQL)	26
CAPÍTULO 3.....		27
3.	Diseño e implementación.....	27
3.1	Extracción y unificación de datos	27
3.2	Diseño del Sistema.....	27
3.2.1	Diseño de funcionalidades	27
3.2.2	Flujograma de proceso del sistema	29
3.3	Simulación	33
3.4	Métricas y Comunicación de Resultados.....	43
CAPÍTULO 4.....		44
4.	Análisis de Resultados.....	44
CONCLUSIONES.....		47
RECOMENDACIONES.....		48
BIBLIOGRAFÍA.....		49

APÉNDICES.....51

ABREVIATURAS

ESPOL	Escuela Superior Politécnica del Litoral
RAG	Retrieval-Augmented Generation
SSAS	SQL Server Analysis Services
SSRS	SQL Server Reporting Services
SQL	Structured Query Language
BI	Business Intelligence
API	Application Programming Interface
RUC	Registro Único de Contribuyente
LLM	Large Language Model
RLHF	Reinforcement Learning from Human Feedback
IE	Information Extraction

ÍNDICE DE FIGURAS

Figura 1.1 Diseño de esquema en estrella	14
Figura 2.1 Familia de LLMs	17
Figura 2.2 Comparación de desempeño de GPT4 vs GPT3.5	18
Figura 2.3 Ejemplo de RAG en AWS.....	21
Figura 3.1 Interfaz de usuario.....	29
Figura 3.2 Flujograma de proceso del sistema.....	29
Figura 3.3 Ingreso de pregunta del usuario	33
Figura 3.4 Validación de pregunta.....	33
Figura 3.5 Contexto consolidado de tarjetas, clientes y comercios	34
Figura 3.6 Contexto de Tarjetas	35
Figura 3.7 Contexto de Clientes	36
Figura 3.8 Contexto de Comercios	37
Figura 3.9 Contexto, metadata y pregunta del usuario previo a la generación del Query	38
Figura 3.10 Generación del Query	39
Figura 3.11 Validación de Integridad	39
Figura 3.12 Llamar a la función para ejecutar consulta	40
Figura 3.13 Ejecutar consulta	40
Figura 3.14 Llamar a la función para seleccionar gráfico	41
Figura 3.15 Seleccionar gráfico	42
Figura 3.16 Presentación de resultados	42
Figura 3.17. Ejemplo de la toma de encuesta	43

ÍNDICE DE TABLAS

Tabla 4.1. Tickets resueltos.....	44
Tabla 4.2 Precisión del sistema RAG por sprint por oficial comercial.....	45
Tabla 4.3 Medidas de tendencia de tiempos entre flujos.....	45

CAPÍTULO 1

1. PLANTEAMIENTO DEL PROBLEMA

En el contexto bancario ecuatoriano, la gestión y disponibilidad de información se enfrenta a desafíos significativos debido a la naturaleza delicada y confidencial de los datos manejados. Esta restricción impide que todos los miembros de la institución tengan acceso total a la información, provocando largos tiempos de espera hasta obtener la data necesaria. Este problema es común en muchas instituciones financieras, donde la eficiencia en la gestión de información se ve afectada por la seguridad de los datos y los sistemas de control de acceso (Kimball & Ross, 2019).

1.1 Descripción del problema

A continuación, se detallan los problemas convencionales a los que se enfrentan los oficiales comerciales dentro de una institución financiera teniendo como enfoque la obtención de datos para cubrir necesidades del negocio:

Las restricciones en el acceso a la información afectan a los oficiales comerciales de tal manera que no siempre pueden llegar al dato que necesitan, pues si la búsqueda tiene un enfoque distinto al área donde se desenvuelven, este no tiene los accesos ni permisos para identificarlo. Por ejemplo, un oficial de un área en específico (establecimientos) no puede ver información sobre su mismo negocio relacionado a otra área (seguros).

La burocracia es otro dolor provocado por estas necesidades de información. Los oficiales comerciales deben hacer solicitudes específicas de información que sirven para un solo uso, lo que conlleva a abrir múltiples solicitudes a lo largo del tiempo generando más carga operativa al área encargada, generando cuellos de botella, los cuales provocan que se dilaten los tiempos de entrega.

A pesar de que el banco ha desarrollado mediante diversas herramientas de BI soluciones para solventar estos requerimientos, tales como cubos de información en SSAS, reportes web en SSRS, dashboards en Power BI y reportes manuales en Excel, los cuales consolidan la información proveniente de los aplicativos internos; la institución se enfrenta al problema de que cada nueva solución viene con la barrera de las limitaciones en el “know-how” de la herramienta.

Los obstáculos mencionados anteriormente promueven a los oficiales comerciales a forzar la información; a modo de ejemplo se pueden mencionar los casos en los que se toman datos de una cierta fecha en el tiempo, pero los completan con datos de otras fechas basándose en estimaciones, creencias, instinto o cualquier otra fuente subjetiva, que finalmente generan descuadres o ambigüedades contra el registro contable.

Por último, estas ambigüedades son consecuencia de simplemente “anotar el dato”, mas no entenderlo. El dato que el oficial necesita puede venir de diferentes fuentes como marcos de tiempo, fechas contables, definiciones de negocio, o algún contexto subyacente; los cuales deben ser claros y precisos en el momento de definir el requerimiento.

1.2 Justificación del problema

Los problemas de accesibilidad, permisos, carga operativa, desconocimiento de herramientas y metodologías, y de completar información con datos extraoficiales que sufren los oficiales comerciales dentro de una institución financiera son importantes de resolver ya que todo esto converge en largos tiempos de respuesta para la obtención de datos necesarios para la toma de decisiones en el negocio.

En una era donde distintas industrias tienen el mismo enfoque: Data Driven Decision Making; es demasiado valioso el tiempo que transcurre desde que surge una duda, pregunta o necesidad en el negocio hasta que damos una respuesta.

Según Marikar & Bandara (2020) las organizaciones están incorporando herramientas de análisis de datos y Business Intelligence (BI) con el fin de analizar datos históricos, prever eventos futuros y acceder a información precisa y confiable que facilite la toma de decisiones.

Es por esto que la solución propone un sistema de inteligencia artificial basado en RAG que sea capaz de contestar las preguntas elaboradas por los oficiales comerciales, abordando directamente la necesidad crítica de disminuir los tiempos de respuesta. Para Jurafsky & Martin (2000) responder preguntas más complejas puede implicar la necesidad de integrar y resumir datos provenientes de varias fuentes. Y es en este punto, donde se evidencia la importancia del uso del modelo RAG para que el sistema sea eficiente.

Este sistema se convertirá en una pieza clave para solucionar los problemas que enfrentan los oficiales comerciales, puesto que, acompañado con un correcto manejo de perfiles de acceso, se podrá dejar de consultar las herramientas tradicionales cuando ellos necesiten un dato en específico y pedirlo directamente en sus propias palabras y obtener los resultados en cuestión de pocos segundos.

Además, la solución al problema busca mitigar costos de licenciamientos varios, pues se reducirían la cantidad de licencias de SSAS o PowerBI, disminuirá la carga operativa para el área encargada de datos puesto que existiría una metodología de autogestión del área comercial y, por último, los oficiales no requerirán adquirir conocimientos sobre un aplicativo, ya que se enfoca únicamente en la correcta definición de los requerimientos.

1.3 Objetivos

1.3.1 Objetivo General

Desarrollar un sistema de inteligencia artificial basado en RAG que utilice tecnologías de generación de respuestas mejoradas por recuperación para la gestión de información en una institución financiera ecuatoriana.

1.3.2 Objetivos Específicos

1. Integrar y procesar las fuentes de información transaccional o core bancario aplicando una estrategia de gobernanza de datos, para que sirvan de insumo para el contexto que se use dentro del modelo.
2. Desarrollar un prototipo basado en los principios de RAG para mejorar los tiempos de consultas de información de la fuerza comercial del banco.
3. Evaluar la precisión y coherencia de las respuestas del sistema mediante pruebas empíricas utilizando métricas de rendimiento.

1.4 Metodología

Se propone desarrollar un sistema de inteligencia artificial basado en RAG, que tenga la capacidad de responder las consultas formuladas por los oficiales comerciales.

El estudio de Lewis et al. (2020) sugiere que la integración de capacidades de recuperación de información con generación de lenguaje natural, a través de modelos como Retrieval-Augmented Generation (RAG), podría mejorar la precisión y relevancia de las respuestas generadas, a la vez que disminuye el tiempo de procesamiento.

Por las limitaciones con las áreas como seguridad de la información, por lo delicado de los datos, el flujo que vamos a usar no puede exponer los datos en crudo hacia la API de OpenAI. Nuestra metodología se basa en desarrollar un proceso robusto que reciba una pregunta de usuario, por dentro vamos a

validar si la pregunta está relacionada al negocio, cuáles son los criterios involucrados dentro de la pregunta, como contexto de tipos de clientes, tipos de comercios, tipos de tarjetas y al final, con el contexto y la metadata de las tablas, poder generar un query que conteste la pregunta del usuario y una visualización efectiva que conteste la pregunta del usuario.

Para esto, vamos a desarrollar el front-end de la aplicación en Streamlit, para las consultas vamos a manejarlas usando DuckDB con archivos parquet como tablas, y para el LLM vamos a usar el modelo GPT4o-Mini de OpenAI.

1.5 Resultados esperados

El presente proyecto tendrá como resultados los siguientes entregables:

- Back-end del aplicativo, que incluya toda la lógica de ejecución del flujo y las reglas comerciales
- Front-end de la interfaz de usuario que sea amigable e intuitiva con el área comercial

1.6 Dataset

Como se ilustra en la Figura 1.1, se va a diseñar un esquema en estrella, como si fuese un cubo de información, con una tabla fact como el centro, y tablas dimensionales alrededor. Esta información corresponde al negocio de medios de pago, que va a ser el área en gestión durante este proyecto.

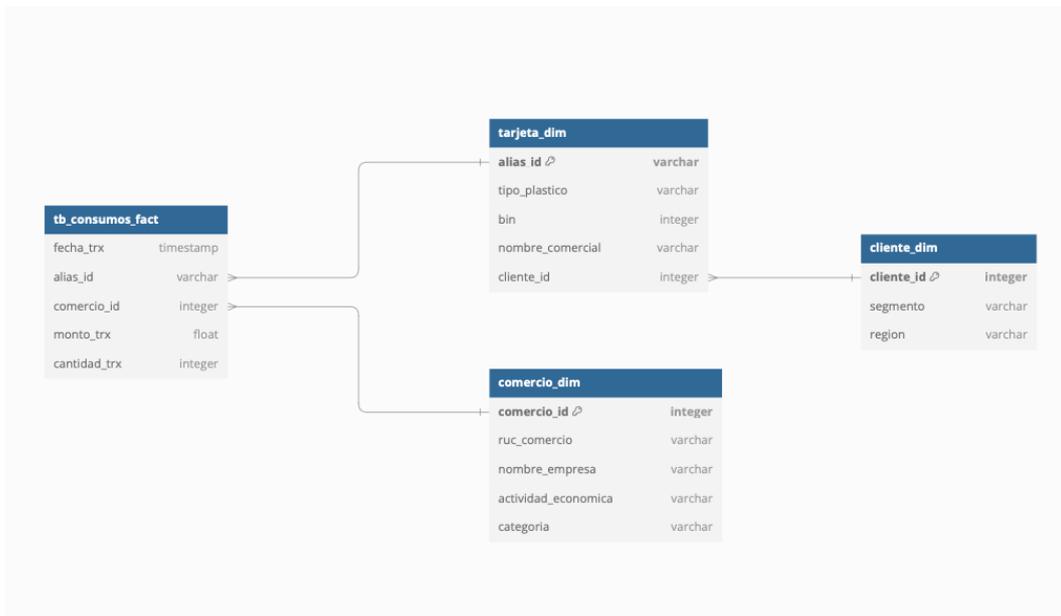


Figura 1.1 Diseño de esquema en estrella

Elaboración propia

Bajo esta información, vamos a contestar las preguntas comerciales que solicita el negocio de forma recurrente hacia el área de analítica ya que, del lado de ellos, el aplicativo que usan les permite ver los detalles de cada tarjeta de un cliente, pero no pueden ver el acumulado de una situación.

1.6.1 tb_consumos_fact

Esta es la tabla fact, que contiene todos los consumos hechos con tarjetas.

Campos:

- fecha_trx: es la fecha en la que se realizó el consumo.
- alias_id: es el número de tarjeta enmascarado.
- comercio_id: es un código asignado a un comercio que pertenece a alguna empresa.
- monto_trx: es el monto total del consumo.
- cantidad_trx: es la cantidad de transacciones.

1.6.2 comercio_dim

Esta tabla es la dimensional de comercios, donde está la información respectiva al comercio.

Campos:

- comercio_id: es un código asignado a un comercio que pertenece a alguna empresa.
- ruc_comercio: es el RUC de la empresa dueña de ese código de comercio.
- nombre_empresa: es el nombre del comercio como tal, digase la razón social del RUC.
- actividad_economica: este campo tiene el detalle de la actividad económica a la que se dedica la empresa.
- categoria: este campo hace referencia a un giro/clasificación de negocio de esta empresa.

1.6.3 tarjeta_dim

Esta tabla es la dimensional de las tarjetas, donde está la información relacionada al plástico.

Campos:

- alias_id: es un string con un número enmascarado de la tarjeta real.
- tipo_plastico: es una marca para saber si ese plástico corresponde a una tarjeta adicional o una tarjeta principal.
- bin: es el bin de la tarjeta (Bank Identification Number).
- nombre_comercial: es el nombre comercial de la tarjeta.
- cliente_id: es un código interno del banco para un cliente.

1.6.4 cliente_dim

Esta tabla es la dimensional de los clientes, donde está su información demográfica.

Campos:

- cliente_id: es un código interno del banco para un cliente.
- segmento: es el segmento estratégico del cliente, según la segmentación interna del banco.
- region: es la región del cliente.

CAPÍTULO 2

2. ESTADO DEL ARTE

En este capítulo se destaca la importancia de analizar el estado del arte en la aplicación de Business Intelligence (BI) en instituciones financieras, enfocado en la gestión eficiente de grandes volúmenes de datos. Se enfatiza la revisión de soluciones actuales y la exploración de innovaciones como la tecnología RAG, que mejora la generación y ejecución de consultas SQL mediante herramientas como OpenAI, DuckDB y Streamlit. Este análisis justifica la elección de estas tecnologías y su potencial para optimizar la toma de decisiones en el área comercial.

2.1 Modelos de Lenguaje de gran escala (LLM)

De acuerdo con la investigación de Minaee et al. (2024) los modelos de lenguaje de gran escala (LLMs, por sus siglas en inglés) son modelos estadísticos preentrenados a gran escala basados en redes neuronales. El desarrollo de los LLMs es el resultado de años de investigación en modelos de lenguaje, los cuales han atravesado cuatro fases principales, cada una con distintos tiempos de inicio y ritmos de progreso: modelos de lenguaje estadísticos, modelos de lenguaje neuronales, modelos de lenguaje preentrenados y los LLMs actuales.

2.1.1 Familias de LLMs

Para Minaee et al. (2024) los modelos de lenguaje a gran escala (LLMs) se refieren principalmente a modelos preentrenados basados en transformadores (PLMs) que cuentan con decenas o incluso cientos de miles de millones de parámetros. En comparación con los PLMs, los LLMs no solo son mucho más grandes en términos de tamaño, sino que también destacan por su mayor capacidad de comprensión y generación de lenguaje, además de exhibir habilidades emergentes que no se observan en modelos de menor escala. La Figura 2.1 proporciona una representación visual de las familias de LLMs.

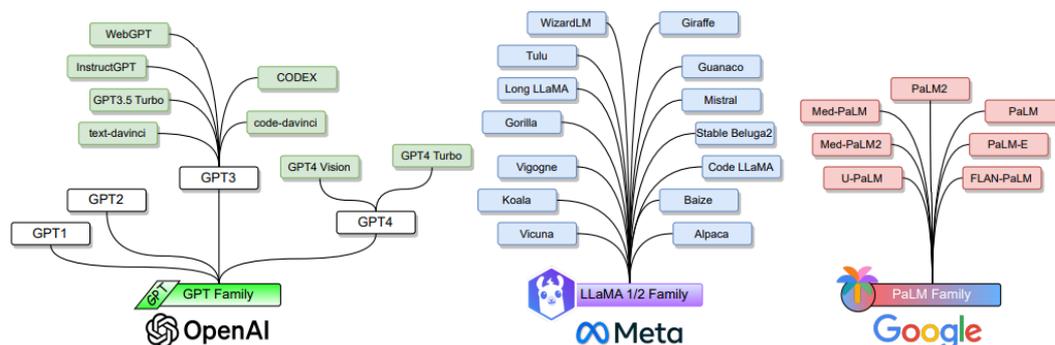


Figura 2.1 Familia de LLMs

Reproducido de "Large Language Models: A Survey," por S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, & J. Gao, 2024, Preprint, arXiv. <https://arxiv.org/abs/2402.06196>.

Como lo expone Minaee et al. (2024) en su estudio, GPT-4 es el modelo de lenguaje más reciente y avanzado de la familia GPT, lanzado en marzo de 2023. Se trata de un modelo multimodal con las capacidades de recibir imágenes y texto como entradas y generar texto como salida. Aunque aún tiene limitaciones frente a los humanos en escenarios complejos del mundo real, GPT-4 ha demostrado un rendimiento comparable al humano en varios estándares profesionales y académicos. Un ejemplo de esto es su desempeño en un examen de abogacía simulado, en el que se ubicó entre el 10% superior de los participantes, como se observa en la Figura 2.2.

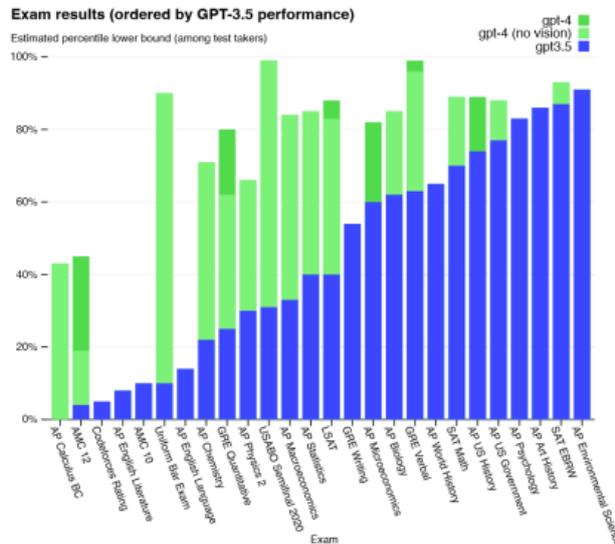


Figura 2.2 Comparación de desempeño de GPT4 vs GPT3.5

Reproducido de "Large Language Models: A Survey," por S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, & J. Gao, 2024, Preprint, arXiv. <https://arxiv.org/abs/2402.06196>.

Minaee et al. (2024) argumentan que, al igual que los primeros modelos GPT, GPT-4 fue preentrenado inicialmente para predecir los siguientes tokens en grandes conjuntos de datos textuales, y posteriormente fue ajustado mediante aprendizaje por refuerzo con retroalimentación humana (RLHF, por sus siglas en inglés de Reinforcement Learning from Human Feedback) para que su comportamiento se alinee con las expectativas humanas.

2.2 Modelo Retrieval-Augmented Generation (RAG)

El modelo Retrieval-Augmented Generation (RAG) integra la recuperación de información con la generación de lenguaje natural, lo que permite que los sistemas no solo recuperen datos relevantes, sino que también generen respuestas detalladas y precisas (Lewis, et al., 2020). La integración de RAG en sistemas de BI incrementa notablemente la precisión y relevancia de las respuestas a consultas complejas, facilitando una mayor comprensión del contexto y una mejor interpretación de los datos.

2.2.1 Uso de RAG con LLM

De acuerdo con la información publicada en el sitio web de AWS (What is RAG (Retrieval-Augmented Generation?)) se obtiene lo siguiente:

Los LLM representan una tecnología fundamental en inteligencia artificial (IA), impulsando chatbots inteligentes y otras aplicaciones de procesamiento del lenguaje natural (PLN). Su propósito es desarrollar bots capaces de responder a preguntas de los usuarios en distintos contextos, utilizando referencias cruzadas de fuentes de conocimiento autorizadas. Sin embargo, esta tecnología presenta un grado de imprevisibilidad en las respuestas generadas. Además, los datos con los que se entrenan los LLM son estáticos, lo que limita el conocimiento a una fecha determinada.

RAG es una solución que aborda varios de estos retos al redirigir el LLM para recuperar información relevante de fuentes autorizadas y predefinidas. Esto otorga a las organizaciones un mayor control sobre el contenido generado, y permite a los usuarios entender mejor el proceso mediante el cual el LLM produce las respuestas. (AWS AMAZON, s.f.)

Con RAG, se incorpora un componente de recuperación de información que emplea la entrada del usuario para extraer datos de una fuente nueva. Tanto la consulta del usuario como la información relevante se envían al LLM, que combina estos nuevos datos con su entrenamiento previo para generar respuestas más precisas. (AWS AMAZON, s.f.)

Según la información proporcionada en el sitio web de AWS (What is RAG (Retrieval-Augmented Generation?)) las siguientes secciones proporcionan una descripción general del proceso:

- Crear datos externos

Los datos que están fuera del conjunto de entrenamiento original del LLM se conocen como datos externos. Estos pueden provenir de diversas fuentes, como API, bases de datos o repositorios de documentos, y presentarse en distintos formatos, como archivos, registros de bases de datos o textos extensos. Una técnica adicional de IA, denominada “incorporación de modelos

de lenguaje” transforma estos datos en representaciones numéricas, que luego se almacenan en una base de datos vectorial. Este procedimiento genera una biblioteca de conocimientos accesible para los modelos generativos de IA.

- Recuperar información relevante

El siguiente paso que describe el artículo de AWS (What is RAG (Retrieval-Augmented Generation)?) es realizar una búsqueda basada en la relevancia. La consulta del usuario se convierte en una representación vectorial y se compara con las bases de datos vectoriales. Por ejemplo, un chatbot inteligente que responda preguntas sobre recursos humanos en una organización podría utilizar este enfoque. Si un empleado consulta "¿Cuántos días de vacaciones tengo?", el sistema recuperará documentos como la política de vacaciones anuales y el historial de vacaciones del empleado. Estos documentos se devuelven por su alta relevancia respecto a la consulta, una relevancia determinada mediante representaciones vectoriales y cálculos matemáticos.

- Aumentar el mensaje de LLM

El modelo RAG enriquece la entrada del usuario al agregar los datos relevantes que han sido recuperados en el contexto de la consulta. Este proceso emplea técnicas de ingeniería de prompts para optimizar la comunicación con el LLM. Al aumentar el mensaje con esta información adicional, los LLMs pueden generar respuestas más precisas a las preguntas de los usuarios. (AWS AMAZON, s.f.)

- Actualizar datos externos

Una posible pregunta sería: ¿qué ocurre si los datos externos se vuelven obsoletos? Para garantizar que la información recuperada esté actualizada, es necesario actualizar tanto los documentos como sus representaciones incrustadas de manera asincrónica. Esto puede lograrse mediante procesos automatizados en tiempo real o mediante procesamiento periódico por lotes. Este reto es común en el análisis de datos, y se pueden emplear diferentes

enfoques de ciencia de datos para gestionar el cambio de manera efectiva. (AWS AMAZON, s.f.)

Esto se explica en mayor detalle en la Figura 2.3, donde se observa un diagrama con el flujo conceptual del uso de RAG con LLM.

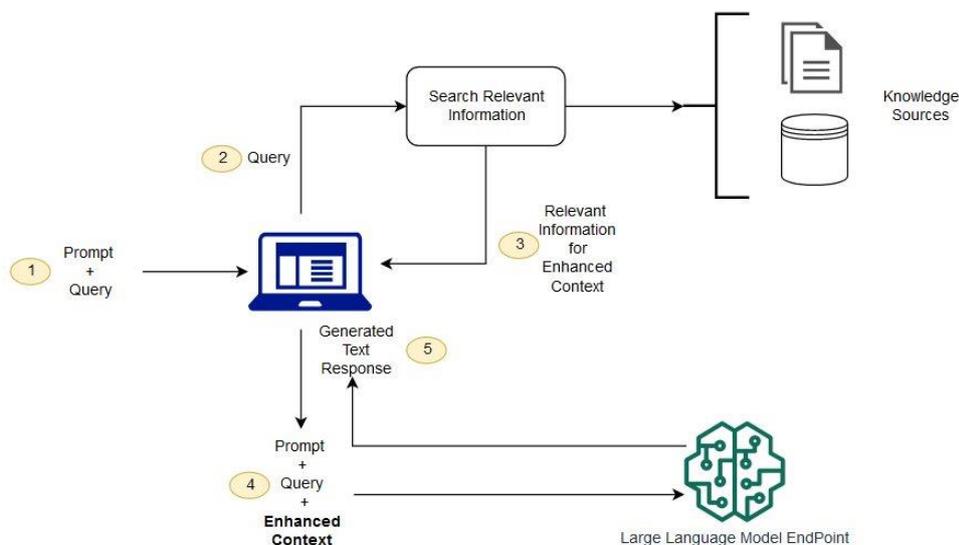


Figura 2.3 Ejemplo de RAG en AWS

Adaptado de "What is RAG (Retrieval-Augmented Generation)?", por AWS AMAZON, s.f., AWS Amazon Website. <https://aws.amazon.com/what-is/retrieval-augmented-generation/>

2.3 Prompt Engineering

Como lo exponen Marvin, Hellen, Jjingo, & Nakatumba-Nabende (2023), el prompt engineering es una técnica fundamental para interactuar con modelos de lenguaje, ya que define cómo se estructuran las entradas que recibe el sistema para asegurar respuestas precisas y útiles. Señalan también que, con el avance acelerado de la IA, especialmente de los LLMs, la ingeniería de prompts se ha vuelto una habilidad esencial para la comunicación eficaz con herramientas basadas en lenguaje, como ChatGPT. Esta técnica permite establecer reglas y automatizar procesos, garantizando tanto la calidad como la cantidad de los resultados generados por los LLMs. No es sorprendente

que las instrucciones generadas automáticamente ofrezcan un rendimiento igual o superior a las anotadas manualmente por humanos, superando puntos de referencia de los LLMs. Esto convierte a la ingeniería de prompts en una especie de programación, utilizada para personalizar las respuestas e interacciones de los LLMs.

2.4 Estudios y artículos sobre la utilización de RAG como solución de negocio

2.4.1 Business-RAG: Information Extraction for Business Insights

Arslan & Cruz (2024) presentan un sistema llamado Business-RAG que combina LLMs) con la técnica de RAG para mejorar la Extracción de Información (IE según sus siglas en inglés de Information Extraction) en el ámbito empresarial.

El objetivo principal de los autores es optimizar la extracción de información a partir de grandes volúmenes de datos comerciales, como informes financieros, artículos de noticias, y documentos legales. Esta mejora permite que las empresas obtengan insights clave para la toma de decisiones en diversas áreas, como pronósticos financieros, evaluación competitiva, y análisis de la opinión de los clientes.

Los autores implementaron un modelo que integra recuperación de documentos y generación de lenguaje natural utilizando un pipeline de procesamiento. El sistema emplea técnicas avanzadas de machine learning para analizar preguntas en lenguaje natural, generar las consultas necesarias, y extraer información específica del dominio empresarial.

Los autores concluyeron que Business-RAG ofrece una solución eficaz y adaptable para la extracción de información empresarial, al permitir que las organizaciones procesen grandes volúmenes de datos de manera más rápida y precisa. Además, destacaron la importancia de continuar investigando y mejorando este enfoque para abordar los desafíos específicos del dominio empresarial, como la precisión en consultas especializadas.

2.4.2 Improving Retrieval for RAG based Question Answering Models on Financial Documents

Setty, Thakkar, Lee, Chung, & Vidra (2024) presentan un enfoque detallado para mejorar los sistemas de preguntas y respuestas (QA) basados en RAG, aplicados a documentos financieros.

El principal objetivo de los autores fue mejorar la fase de recuperación en los sistemas RAG cuando se aplican a documentos financieros. Dado que estos documentos contienen terminología técnica y estructuras complejas, los modelos estándar pueden tener dificultades para recuperar la información más relevante y generar respuestas adecuadas.

Los autores implementaron varias mejoras en la fase de recuperación del sistema RAG para ajustarlo mejor al dominio financiero. Específicamente, el estudio involucró:

- Ajustaron los algoritmos de recuperación para trabajar con la estructura de los documentos financieros, dividiéndolos en fragmentos manejables.
- Utilizaron una base extensa de informes financieros, informes de ganancias y otros documentos del ámbito financiero, para que el modelo pudiera reconocer mejor los patrones y términos específicos de este dominio.
- Midieron la precisión de las respuestas utilizando métricas estándar como precisión, recuperación y F1-score, evaluando cuán bien el sistema recuperaba la información correcta y cuán coherentes eran las respuestas generadas.

Los resultados de las pruebas mostraron que las mejoras introducidas en el sistema RAG, específicamente en la fase de recuperación, condujeron a un aumento significativo en la precisión y coherencia de las respuestas generadas.

Los autores concluyeron que la mejora en los modelos de recuperación dentro del sistema RAG es esencial para manejar la complejidad de los documentos

financieros. Estas mejoras permitieron a los modelos no solo recuperar la información relevante de manera más efectiva, sino también generar respuestas más precisas y contextualizadas, alineadas con los estándares requeridos en el sector financiero. Además, subrayaron que esta metodología podría ser aplicada en otros dominios con documentos complejos y especializados, adaptando el enfoque de recuperación a los contextos específicos.

2.4.3 Financial Report Chunking for Effective Retrieval Augmented Generation

Yepes, You, Milczek, Laverde, & Li (2024) evaluaron la eficacia de segmentar informes financieros como una estrategia para mejorar la generación de respuestas mediante la técnica de RAG. Utilizaron métodos de chunking para dividir los informes en partes más manejables, lo que facilitó la recuperación de información relevante y precisa en tiempo real. Para la evaluación, emplearon un conjunto de datos de informes financieros y compararon el rendimiento del sistema basado en RAG con otras técnicas tradicionales de generación de respuestas.

Los autores desarrollaron un modelo basado en inteligencia artificial que combina técnicas de procesamiento de lenguaje natural con un sistema de recuperación de información. La evaluación se realizó utilizando métricas estándar en tareas de recuperación y generación, midiendo la precisión y el tiempo de respuesta.

Concluyeron que el sistema propuesto mejoró significativamente la precisión de las respuestas y redujo los tiempos de consulta en comparación con enfoques anteriores. El uso del chunking ayudó a que el sistema manejara grandes volúmenes de información financiera de manera eficiente, permitiendo una mejor toma de decisiones basada en datos consolidados y oportunos.

2.5 Soluciones de Analítica y Aprendizaje modernas

Las soluciones modernas de BI han progresado notablemente, facilitando una integración más eficiente de herramientas y tecnologías que optimizan el análisis de datos. En este contexto, el uso de la API de OpenAI se destaca como una herramienta innovadora para la generación de consultas en lenguaje natural que podrían ser traducidas a SQL y ejecutadas para obtener información relevante de una tabla maestra de clientes (Brown, et al., 2020). Además, OpenAI ha logrado que los LLMs como chatgpt-4 o chatgpt-4o-mini sean más accesibles y eficientes, lo que ha generado una amplia aceptación entre los usuarios debido a sus resultados efectivos.

2.5.1 Generación de Consultas SQL con OpenAI

La API de OpenAI facilita la creación de consultas SQL a partir de descripciones en lenguaje natural, ya que, con un adecuado prompt engineering, el modelo estará entrenado y tendrá el contexto adecuado y necesario para generar respuestas coherentes y alineadas con la realidad. Esto simplifica significativamente el proceso de consulta para usuarios que no están familiarizados con SQL (Brown, et al., 2020); esta innovación ofrece la posibilidad de reducir la barrera técnica para el acceso a información estratégica dentro de instituciones financieras.

2.5.2 Ejecución de Consultas en DuckDB

DuckDB es una base de datos embebida optimizada para el análisis de datos, diseñada específicamente para procesar eficientemente datos en formato parquet (Raasveldt & Mühleisen, 2019). Su capacidad para ejecutar consultas SQL sobre grandes volúmenes de datos en memoria y en almacenamiento comprimido ofrece una mejora significativa en el rendimiento y la velocidad de las consultas en comparación con las bases de datos tradicionales.

2.5.3 Visualización con Streamlit

Streamlit es una biblioteca de código abierto en Python que facilita la creación de aplicaciones web para compartir resultados analíticos, desarrollar experiencias interactivas complejas y construir modelos de aprendizaje

automático. A diferencia de otras herramientas de visualización que suelen ser lentas, no permiten la interacción del usuario o no son ideales para respaldar la toma de decisiones en ciencia de datos, Streamlit destaca por ser un marco que permite crear aplicaciones web en Python de manera eficiente. Ofrece métodos integrados para manejar entradas de usuario, como texto y fechas, así como para visualizar gráficos interactivos utilizando las principales bibliotecas de gráficos de Python. (Richards, 2023)

2.5.4 SQL tradicional en bases de datos relacionales (MySQL, PostgreSQL)

Las bases de datos relacionales tradicionales como MySQL y PostgreSQL son conocidas por su robustez, fiabilidad y amplio soporte. No obstante, tienen ciertas limitaciones en la gestión de grandes volúmenes de datos y, en comparación con DuckDB, pueden ser menos eficientes para ejecutar consultas sobre datos almacenados en formatos optimizados, como Parquet. (Suehring, 2001).

CAPÍTULO 3

3. DISEÑO E IMPLEMENTACIÓN

Este capítulo describe la consolidación de la información transaccional y la arquitectura del sistema de inteligencia artificial basado en RAG aplicado hacia el área comercial de Medios de Pago de una institución financiera, utilizando el modelo de lenguaje “GPT-4o mini” para generar consultas y Streamlit para la visualización de resultados.

3.1 Extracción y unificación de datos

Por ser información transaccional, se tuvo que realizar una réplica de la información desde la trama de la transacción, que es donde se guarda todo el registro de la transacción, pero para hacer más eficiente las consultas, no se trajo toda la información, sino solo los campos que sirven en base a nuestro diseño de estrella. Para esto se usó Python y se hicieron consultas usando pandas y se guardó en archivos parquet usando pyarrow. Estas consultas se hacen a día vencido para no sobrecargar el servidor transaccional, de esta forma apuntar nuestro proyecto a un respaldo que sea más robusto y no interfiera con las operaciones diarias.

3.2 Diseño del Sistema

El diseño del sistema se basa en un enfoque orientado a la interacción dinámica entre el usuario y el modelo de lenguaje GPT4o- mini, a través de una interfaz gráfica desarrollada en Streamlit. El proceso comienza cuando el usuario introduce una consulta en la interfaz, que es procesada por el sistema mediante un flujo de datos predefinido y acaba cuando se le contesta al usuario con una visualización acorde a la necesidad de la pregunta.

3.2.1 Diseño de funcionalidades

Para este proyecto, se ha seguido principios SOLID; según el Principio de Responsabilidad Única (SRP, según sus siglas en inglés), se crearon las funcionalidades de forma externa al aplicativo “main”. Se separó en tres apartados:

- Las funcionalidades relacionadas a la generación de la consulta en SQL.
- Las funcionalidades relacionadas a la visualización de los resultados.
- El aplicativo en streamlit que sería el “main” que consume los paquetes de funcionalidades.

Para usar de forma eficiente LangChain y OpenAI, las funcionalidades van con el decorador @tool para poder usar agent_scratch y function calling dentro de los agentes.

3.2.1.1 Funcionalidades de consulta

En las funcionalidades de consulta, se embebe todo el flujo desde que se recibe la pregunta del usuario hasta la generación y ejecución del query que responda la pregunta del usuario. Dentro de las funciones de consulta, se generan los prompts, se conecta a la API de OpenAI y esto se lo hace mediante cadenas simples o compuestas de LangChain para poder seguir la trazabilidad del flujo en LangSmith.

3.2.1.2 Funcionalidades gráficas

En las funcionalidades gráficas, se diseñaron las funciones de los gráficos que se podrían usar para transmitir la información. Según (Nussbaume, 2015) hay un puñado de gráficos que abarcan casi la totalidad de situaciones cuando se trata de transmitir un dato, en base a esto se decidió implementar:

- Gráficos de barras: para mostrar distribuciones.
- Gráficos de líneas: para ver evoluciones en el tiempo.
- Gráficos de dispersión: para ver relaciones/correlaciones.
- Tablas: para mostrar datos detallados.

3.2.1.3 Aplicativo principal

Dentro del aplicativo principal, está el desarrollo del front-end de nuestro portal de autoatención, para esto usamos streamlit. Tal como se aprecia en la Figura 3.1, diseñamos un portal web que tenga pocos elementos para que los usuarios comerciales no puedan perderse dentro; este portal incluye una barra

de texto donde se escribe la pregunta comercial, y un botón de búsqueda, tratando de imitar lo que ofrece Google debido a su alta acogida además de la familiaridad que ya siente el público.

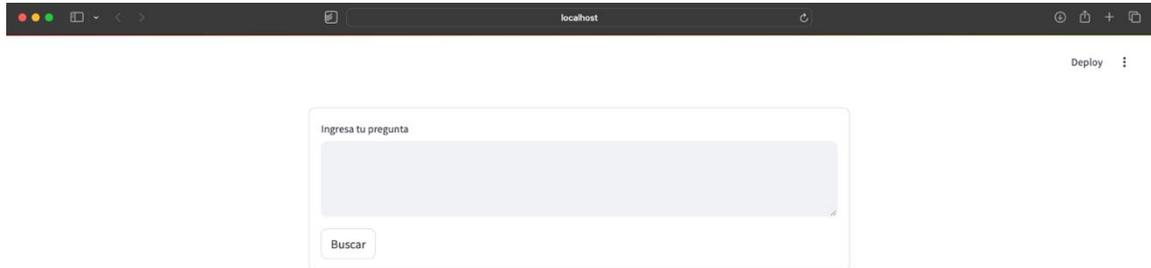


Figura 3.1 Interfaz de usuario

Elaboración propia

3.2.2 Flujograma de proceso del sistema

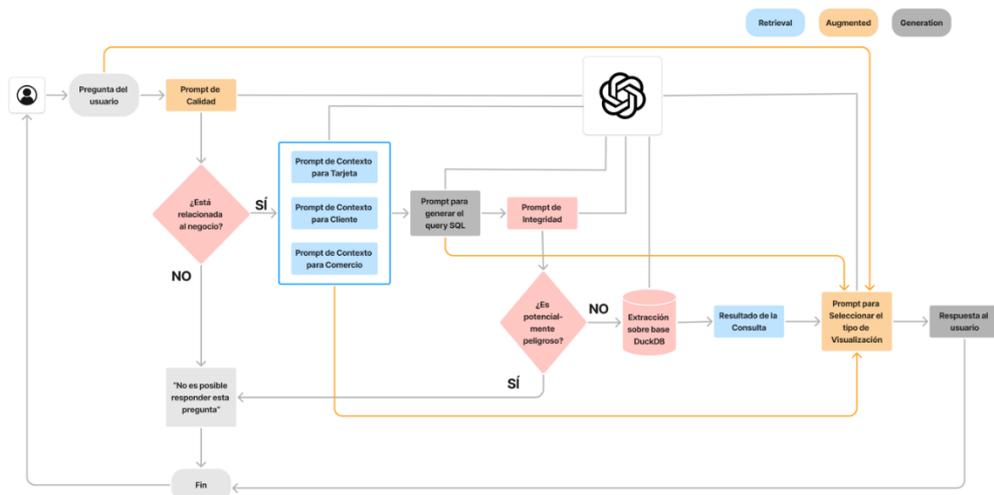


Figura 3.2 Flujograma de proceso del sistema

Elaboración propia

Una vez que el usuario ingresa su pregunta en la barra de texto y le da al botón de “buscar” comienza el flujo. El flujo ilustrado en la Figura 3.2 lo hemos diagramado de tal forma que se pueda asociar cada paso a su respectiva etapa de RAG a través de colores; así como su interacción con los agentes LLM.

1. Prompt de calidad. Se usa un prompt template de LangChain para crear un prompt dinámico donde definimos las instrucciones del sistema, incluyendo de qué trata el negocio de Medios de Pago, y dejamos una variable que sea “user_prompt” donde se va a ir reemplazando a medida que se haga cada pregunta en el sistema.
2. Validación comercial. Se invoca al LLM y se pregunta: ¿Esta pregunta tiene que ver con temas del negocio de Medios de Pago? Y se le pide que la respuesta solo pueda tomar dos valores: “Comercial” o “No Comercial”.
 - a. Si la pregunta es “Comercial” se continua con el flujo
 - b. Si la pregunta no es comercial, ahí acaba el flujo y se muestra “No es posible responder esta pregunta”
3. Suponiendo que la pregunta es comercial, se usa una cadena con ramas paralelas que obtiene el contexto en base a las dimensionales de las tablas. Se le especifica en cada rama que no debe enfocarse en la intención de la pregunta, sino en identificar datos relacionados a las tablas dimensionales.
 - a. Prompt de contexto para tarjetas. Se crea una plantilla donde se especifica en las instrucciones del sistema los tipos de tarjeta que hay, y los nombres comerciales que debe identificar dentro de la pregunta del usuario.
 - b. Prompt de contexto para clientes. Se crea una plantilla donde se especifica en las instrucciones del sistema que tipo de clientes

maneja el banco, comentándole las posibles regiones y los posibles segmentos estratégicos que pueden tener los clientes, de tal manera que se familiarice con la segmentación específica del banco.

- c. Prompt de contexto para comercios. Se crea una plantilla donde se especifica en las instrucciones del sistema información relevante a los comercios, como cuáles son los posibles giros de negocio que pueda haber (categoría en la tabla de comercio_dim), para que pueda identificar si la pregunta es relacionada a alguna actividad comercial específica o para identificar correctamente qué comercio es el adecuado si es que existieran dos que se llamen igual pero se dediquen a actividades distintas.
4. Prompt para generar el query SQL. Se crea una plantilla que contiene la metadata de las tablas y aparte se incluye el contexto obtenido. Se le especifica que va a recibir una pregunta del usuario y que debe generar un query que sea compatible con DuckDB que pueda contestar la pregunta del usuario, ya que estamos seguros de que la pregunta sí es comercial y además se le da detalles adicionales para que pueda interactuar con los filtros de forma más específica.
 5. Prompt de integridad. Como filtro de seguridad, se crea una plantilla donde se le especifica la arquitectura en la que trabaja, y se le pasa el query que se generó en el paso previo dejando a un lado la intención comercial y el contexto de negocio y su instrucción de que evalúe si este query puede afectar a las tablas relacionadas.
 6. Validación de integridad. Se invoca el LLM con el query obtenido en el paso 4 y se le pide que la respuesta solo pueda tomar dos valores: “Seguro” o “Peligroso”.
 - a. Si el query es “Seguro” se continúa con el flujo

- b. Si el query es peligroso, ahí acaba el flujo y se muestra “No es posible responder esta pregunta”
7. Con el query validado, se invoca ahora a un agente y se le da como herramientas una función de “ejecutar_query”, que está dentro de las funcionalidades de consultas, para que en base al query obtenido del punto 4, lo ejecute y pueda llevar la trazabilidad de donde se está almacenando la consulta.
8. Prompt para seleccionar el tipo de visualización. Se crea una plantilla donde se menciona en las instrucciones del sistema, cuáles son las mejores prácticas para escoger gráficos, y para que sea dinámico se crean variables de “user_prompt”, “query” para que sepa cual fue la pregunta del usuario, cuál fue el query utilizado y conozca como podría venir la información.
9. Respuesta al usuario. En este paso, se invoca otro agente y se le provee las herramientas (tools) de las funcionalidades gráficas, para que escoja la mejor visualización y la renderice en streamlit.

Agrupando nuestro flujo en las etapas de RAG,

- Retrieval. Esta etapa agrupa los pasos donde se obtiene información, que son la parte de obtener contexto, y la generación del query.
- Augmented. En base a nuestro retrieval, tenemos dos etapas de augmented en cada funcionalidad. En las consultas, el contexto de tarjetas, comercios, clientes y pregunta del usuario nos permiten dar más detalles al prompt para que el LLM pueda dar un query más efectivo y en la parte gráfica, se puede ofrecer más contexto con la pregunta del usuario, el query y las mejores prácticas de visualización.

- Generation. Encontramos dos etapas de generation, cuando se genera el query final para contestar la pregunta del usuario, y cuando se genera la visualización adecuada para contestarle al usuario.

3.3 Simulación

Se puede observar en la Figura 3.3 que se va a realizar la simulación de un usuario preguntando “¿Cómo se distribuye el monto total de consumos por segmento durante el 2018?” donde se va a usar la interfaz de LangSmith para ver cómo funciona el flujo por detrás.

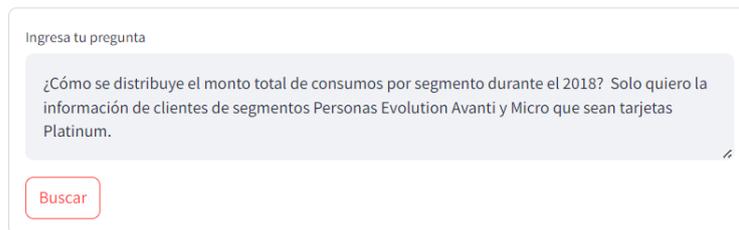


Figura 3.3 Ingreso de pregunta del usuario

Elaboración propia

1. En la Figura 3.4 se ilustra la validación de la pregunta para identificar si es o no es comercial.

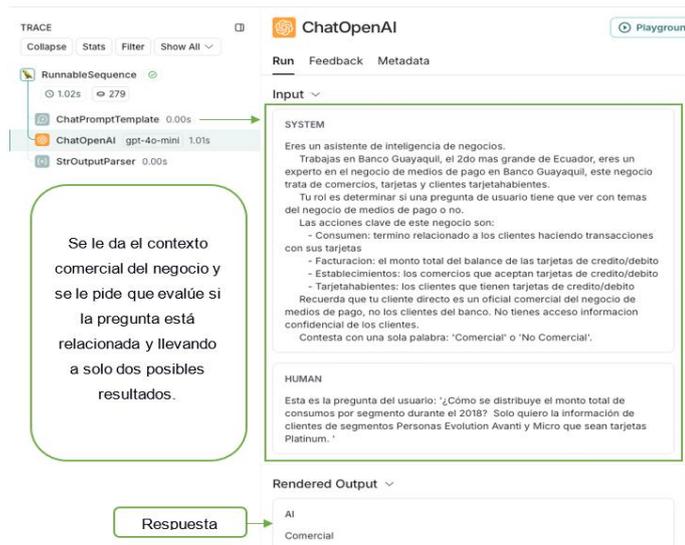


Figura 3.4 Validación de pregunta

Elaboración propia

2. La Figura 3.5 demuestra cómo se obtiene el contexto relacionado, en donde se va a buscar qué se especifica sobre clientes, sobre comercios y sobre tarjetas.

The screenshot displays a workflow trace for a 'RunnableSequence' task. The trace shows a 'RunnableParallel<branches>' task that splits into three parallel branches: 'map:key:clientes', 'AgentExecutor', and 'map:key:tarjetas'. Each branch contains a 'RunnableLambda' task followed by a 'ChatOpenAI' task. The 'Rendered Output' section shows the consolidated results for 'Clientes' and 'Comercios'. A callout box labeled 'Ramas de contexto' points to the three parallel branches. Another callout box states 'Se obtiene el contexto consolidado de las tres ramas'.

```
commercial_question: "¿Cómo se distribuye el monto total de consumos por segmento durante el 2018? Solo quiero la información de clientes de segmentos Personas Evolution Avanti y Micro que sean tarjetas Platinum."
```

```
Clientes://nsegmento personas, segmento evolution, segmento avanti, segmento micro//n/nTarjetas://nVISA PLATINUM, MASTERCARD PLATINUM, AMEX PERSONAL PLATINUM, AMEX PLATINUM TACA.//n/nComercios://nLa pregunta no tiene relación con comercios, por lo que no se puede generar una consulta SQL. La respuesta es en blanco.
```

Figura 3.5 Contexto consolidado de tarjetas, clientes y comercios

Elaboración propia

- a. Contexto de tarjetas. En la Figura 3.6 se visualiza cómo se le da el contexto sobre las posibles tarjetas y los tipos de tarjeta que hay y se le da consideraciones. Esperando que conteste con los nombres y tipos adecuados según la pregunta.

The image displays a LangChain trace and a ChatOpenAI playground interface. The trace on the left shows a sequence of steps including RunnableSequence, RunnableParallel, ChatOpenAI, and StrOutputParser. The playground on the right shows the input context, including a list of card names and instructions. A callout box points to the 'Rendered Output' section, which contains the AI response: 'VISA PLATINUM, MASTERCARD PLATINUM, AMEX PERSONAL PLATINUM, AMEX PLATINUM TACA.'

Input

SYSTEM

'AMEX MALL DEL RIO',
'VISA ORO',
'MASTERCARD ORO',
'AMEX QUICENTRO SHOPPING',
'AMEX BTA',
'AMEX BUSINESS LINK',
'AMEX PLATINUM CREDITO',
'AMEX BLUE BOX CREDITO',
'AMEX CUPO CERO GARANTIA',
'AMEX BLUE JOVENES',
'VISA TACA CLASICA',
'AMEX CHEVROLET CLASICA',
'AMEX BUSINESS PERSONA',
'AMEX SUPERMAXI CLASICA',

Estos nombres son compuestos MARCA + * * + NOMBRE COMERCIAL
Tarea:
Tu objetivo es analizar la pregunta del usuario comercial, y enfocarte unicamente en la parte de la pregunta que se relacione a las tarjetas como tal
Se espera que contestes con una referencia de a qué se pueda referir el usuario con su pregunta

Consideraciones:

- * Si la pregunta no tiene nada relacionado a una consulta de tarjetas, devuélvela en blanco
- * Los terminos en los nombres comerciales son bilingües, recuerda que pueden decirte "oro" pero el nombre real es "gold" y debes buscar ambas coincidencias
- * Solo se tienen 3 marcas, AMEX - VISA - MASTERCARD
- * Toda la respuesta, de existir, debe venir en mayúscula
- * Recuerda incluir todos los nombres relacionados

Ejemplo:
usuario: "cuantos clientes del portafolio platinum consumen en los supermercados la favorita"
idea: no tienes que enfocarte en la pregunta completa, solo identificar lo que puede estar relacionado a tarjetas que es "portafolio platinum"
en base a la lista de tarjetas que conoces, devuelve todos los nombres platinum indistintamente si es credito/personal o la marca como tal, sino todas las que mencionen platinum

HUMAN

Esta es la pregunta comercial {'commercial_question': '¿Cómo se distribuye el monto total de consumos por segmento durante el 2018? Solo quiero la información de clientes de segmentos Personas Evolution Avanti y Micro que sean tarjetas Platinum. '}

Rendered Output

AI

VISA PLATINUM, MASTERCARD PLATINUM, AMEX PERSONAL PLATINUM, AMEX PLATINUM TACA.

Solo devuelve los nombres "Platinum"

Figura 3.6 Contexto de Tarjetas

Elaboración propia

- b. Contexto para clientes. En la Figura 3.7 se observa cómo se le da el contexto del tipo de clientes que se maneja en el negocio, incluyendo los catálogos de segmentos y regiones.

The image shows a workflow trace on the left and a ChatOpenAI interface on the right. The trace includes steps like 'RunnableSequence', 'RunnableParallel', 'map:key:clientes', 'ChatOpe...', 'StrOutputParser', 'RunnableLambda', 'AgentExecutor', 'RunnableSequ...', 'RunnableAs...', 'Runnable...', 'ChatPromptTem...', 'ChatO...', 'ToolsAgentOut...', 'map:key:tarjetas', 'RunnableLambda', 'ChatOpe...', 'StrOutputParser', and 'RunnableLambda'. The ChatOpenAI interface shows the 'Input' section with a 'SYSTEM' prompt. The prompt includes a role description, context about the business (Banco Guayaquil), classification criteria for clients (segmento and región), and a task description. Below the prompt is an example question and its answer. The 'Rendered Output' section shows the AI response: 'segmento personas, segmento evolution, segmento avanti, segmento micro'. A green box highlights the 'Rendered Output' section, and a callout box points to it with the text: 'Se obtiene los segmentos que se especifican en la pregunta'.

Se obtiene los segmentos que se especifican en la pregunta

Figura 3.7 Contexto de Clientes

Elaboración propia

- c. Contexto para comercios. Con la Figura 3.8 se demuestra cómo usamos un agente para ir a conseguir los comercios relacionados a la pregunta, con la instrucción de no forzarlo, sino hay nada, devuelve una respuesta alineada a la situación

The screenshot displays the ChatOpenAI interface with a trace of an agent's execution. The trace shows a sequence of steps including RunnableSequence, RunnableParallel, map, RunnableLambda, ChatOpenAI, StrOutputParser, AgentExecutor, and ToolsAgentOutputParser. The ChatOpenAI step is highlighted, showing the input prompt and the rendered output.

Input

SYSTEM

Eres un experto en base de datos y en buscar los comercios relacionados en base a la pregunta de un usuario. Dado el contexto de la pregunta, debes encontrar el/los comercio/s que sean necesarios para contestar la pregunta a través de consultas SQL.

Tienes esta metadata:

Tabla: comercio_dim

Descripción: esta tabla contiene la información relacionada a los comercios

Campo:

- ruc_comercio: VARCHAR = Este campo contiene el RUC de la empresa (Registro Único de Contribuyente)
- nombre_empresa: VARCHAR = Este campo contiene el nombre de la empresa (todo en mayúsculas) se puede considerar la razón social
- categoria: VARCHAR = Este campo hace referencia a un giro/clasificación de esta empresa.

Los posibles valores de clasificación son:

['SERVICIOS FINANCIEROS', 'COMERCIO EN GENERAL', 'COMERCIO / SUPERMERCADOS', 'OTROS SERVICIOS', 'TRANSPORTE DE PASAJEROS', 'INDUSTRIA FARMACEUTICA', 'TURISMO-HOTELES-RESTAURANTES', 'TELECOMUNICACIONES', 'COMERCIO / ARTICULOS IMPORTADOS', 'OTROS TRANSPORTE', 'COMERCIO / ELECTRODOMESTICOS Y EQUIPOS', 'CONSTRUCCION E INFRAESTRUCTURA', 'TRANSPORTE Y ALMACENAMIENTO', 'MAQUINARIA Y EQUIPO AGRICOLA E INDUSTRIAL', 'EQUIPOS Y MATERIALES DE CONSTRUCCION', 'SERVICIOS DE SALUD', 'OTRAS INDUSTRIAS MANUFACTURERAS', 'SERVICIOS DE ENTRETENIMIENTO', 'COMERCIO / REPUESTOS AUTOMOTRIZ', 'PETROLEO Y GAS', 'PRODUCTOS ALIMENTICIOS', 'INSUMOS AGRICOLAS', 'TRANSPORTE DE CARGA', 'METALES Y PRODUCTOS DEL METAL', 'MADERA', 'OTROS CULTIVOS', 'TRANSPORTE NAVIERO/ PUERTOS', 'MAQUINARIA Y EQUIPO AGRICOLA E INDUSTRIAL', 'CAMARON', 'PROVEEDORES DEL ESTADO EN SERVICIOS', 'MINAS Y CANTERAS', 'CAÑA DE AZUCAR', 'INDUSTRIA DE BEBIDAS', 'AVICOLA', 'INDUSTRIA FLORICOLA', 'INDUSTRIA DE LA PESCA', 'CACAO', 'GANADERO', 'ARROZ', 'BANANO Y CADENA DE ABASTECIMIENTO']

-actividad_economica: VARCHAR = Este campo tiene el detalle de la actividad económica a la que se dedica la empresa

Consideraciones:

- * El query que generes debe funcionar en DuckDB

HUMAN

Esta es la pregunta del usuario ('comercial_question': '¿Cómo se distribuye el monto total de consumos por segmento durante el 2018? Solo quiero la información de clientes de segmentos Personas Evolution Avanti y Micro que sean tarjetas Platinum.')

AI

Estoy procesando la información...

AI

[]

Rendered Output

AI

La pregunta no tiene relación con comercios, por lo que no se puede generar una consulta SQL. La respuesta es en blanco.

Se identifica que no hay nada relacionado a comercios en la pregunta

Figura 3.8 Contexto de Comercios

Elaboración propia

3. Se genera el query. En esta etapa se da la metadata de las tablas, el contexto que se generó con las ramas de contexto, y la pregunta del usuario, tal como se ilustra en la Figura 3.9.

La generación del Query se demuestra en la Figura 3.10.

The screenshot shows the ChatOpenAI interface with the following content in the 'Input' section:

```

SYSTEM
# Contexto
Eres un experto en SQL y en análisis de datos. Tu tarea es generar consultas SQL basadas en preguntas en lenguaje natural, utilizando la estructura de datos proporcionada y el contexto de la pregunta.

# Estructura de la base de datos
Tabla      : tb_consumos_fact
Descripción : Esta tabla contiene todos los consumos hechos con tarjeta de crédito
Campos:
  * fecha_trx      : (DATETIME) es la fecha en la que se realizó el consumo
  * alias_id       : (VARCHAR) es el número de tarjeta enmascarado
  * comercio_id    : (INT) es un código asignado a un comercio que pertenece a alguna empresa
  * monto_trx      : (FLOAT) es el monto total del consumo
  * cantidad_trx   : (FLOAT) es la cantidad de transacciones
Relaciones:
  - tb_consumos_fact.alias_id = tarjeta_dim.alias_id
  - tb_consumos_fact.comercio_id = comercio_dim.comercio_id

Tabla      : comercio_dim
Descripción : Esta tabla es la dimensional de comercios, donde está la información respectiva al comercio
Campos:
  * comercio_id    : (INT) es un código asignado a un comercio que pertenece a alguna empresa
  * ruc_comercio   : (VARCHAR) es el ruc de la empresa dueña de ese código de comercio
  * nombre_empresa : (VARCHAR) es el nombre del comercio como tal, dígame la razón social del RUC (en el caso de Ecuador, tenemos la compañía "La Favorita")
  * actividad_economica : (VARCHAR) Este campo tiene el detalle de la actividad económica a la que se dedica la empresa
  * categoria      : (VARCHAR) Este campo hace referencia a un giro/clasificación de negocio de esta empresa
Relaciones:
  - comercio_dim.comercio_id = tb_consumos_fact.comercio_id

Tabla      : tarjeta_dim
Descripción : Esta tabla es la dimensional de las tarjetas, donde está la información relacionada al plástico y el id del dueño de la tarjeta
Campos:
  * alias_id       : (VARCHAR) es un string con un número enmascarado de la tarjeta real
  * tipo_plastico  : (VARCHAR) es una marca para saber si ese plástico corresponde a una tarjeta adicional o una tarjeta principal

HUMAN
Esta es la pregunta del usuario ¿Cómo se distribuye el monto total de consumos por segmento durante el 2018? Solo quiero la información de clientes de segmentos Personas Evolution Avanti y Micro que sean tarjetas Platinum.

HUMAN
Este es el contexto Clientes://nsegmento personas, segmento evolution, segmento avanti, segmento micro//n/nTarjetas://nVISA PLATINUM, MASTERCARD PLATINUM, AMEX PERSONAL PLATINUM, AMEX PLATINUM TACA.//n/nComercios://nLa pregunta no tiene relación con comercios, por lo que no se puede generar una consulta SQL. La respuesta es en blanco.
  
```

Annotations in the image point to the following parts of the input:

- Metadata de las tablas**: Points to the system prompt section describing the database structure (tb_consumos_fact, comercio_dim, tarjeta_dim).
- Pregunta del usuario**: Points to the first human prompt: "Esta es la pregunta del usuario ¿Cómo se distribuye el monto total de consumos por segmento durante el 2018? Solo quiero la información de clientes de segmentos Personas Evolution Avanti y Micro que sean tarjetas Platinum."
- Contexto**: Points to the second human prompt: "Este es el contexto Clientes://nsegmento personas, segmento evolution, segmento avanti, segmento micro//n/nTarjetas://nVISA PLATINUM, MASTERCARD PLATINUM, AMEX PERSONAL PLATINUM, AMEX PLATINUM TACA.//n/nComercios://nLa pregunta no tiene relación con comercios, por lo que no se puede generar una consulta SQL. La respuesta es en blanco."

Figura 3.9 Contexto, metadata y pregunta del usuario previo a la generación del Query

Elaboración propia

Rendered Output ▾

```
AI
{
  'logica_usada': 'Se interpreta la pregunta como una solicitud para obtener la distribución del monto total de consumos por segmento durante el año 2018, considerando solamente los segmentos Personas, Evolution, Avanti y Micro, así como las tarjetas Platinum. Para esto, se utilizarán las tablas cliente_dim, tarjeta_dim y tb_consumos_fact. Se aplicará un filtro para el año 2018 y para los segmentos y tipos de tarjeta especificados.',
  'query': "
-- Seleccionamos el monto total de consumos por segmento para el año 2018
SELECT cd.segmento, SUM(cf.monto_trx) AS monto_total
FROM cliente_dim cd
JOIN tarjeta_dim td ON cd.cliente_id = td.cliente_id
JOIN tb_consumos_fact cf ON td.alias_id = cf.alias_id
WHERE strftime('%Y', cf.fecha_trx) = '2018'
AND cd.segmento IN ('PERSONAS', 'EVOLUTION', 'AVANTI', 'MICRO')
AND td.nombre_comercial IN ('VISA PLATINUM', 'MASTERCARD PLATINUM', 'AMEX PERSONAL PLATINUM', 'AMEX PLATINUM TACA')
GROUP BY cd.segmento;
"
}
```

Query generado

Figura 3.10 Generación del Query

Elaboración propia

4. Validación de integridad. La Figura 3.11 representa cómo se valida si el query es peligroso para la integridad de las tablas de tal forma que las pudiera alterar.

Se especifica que es "peligroso" para evaluar el query

Evaluación del query

ChatOpenAI

Run Feedback Metadata

Input ▾

SYSTEM

Eres un experto en SQL. Tu rol es determinar si un query es peligroso para la integridad de las tablas, el query solo puede ser de consulta, no debería tener truncates, delete, updates, etc... Contesta con una sola palabra: 'Peligroso' o 'Seguro'.

HUMAN

Esta es el query: {
'logica_usada': 'Se interpreta la pregunta como una solicitud para obtener la distribución del monto total de consumos por segmento durante el año 2018, considerando solamente los segmentos Personas, Evolution, Avanti y Micro, así como las tarjetas Platinum. Para esto, se utilizarán las tablas cliente_dim, tarjeta_dim y tb_consumos_fact. Se aplicará un filtro para el año 2018 y para los segmentos y tipos de tarjeta especificados.',
'query': "
-- Seleccionamos el monto total de consumos por segmento para el año 2018
SELECT cd.segmento, SUM(cf.monto_trx) AS monto_total
FROM cliente_dim cd
JOIN tarjeta_dim td ON cd.cliente_id = td.cliente_id
JOIN tb_consumos_fact cf ON td.alias_id = cf.alias_id
WHERE strftime('%Y', cf.fecha_trx) = '2018'
AND cd.segmento IN ('PERSONAS', 'EVOLUTION', 'AVANTI', 'MICRO')
AND td.nombre_comercial IN ('VISA PLATINUM', 'MASTERCARD PLATINUM', 'AMEX PERSONAL PLATINUM', 'AMEX PLATINUM TACA')
GROUP BY cd.segmento;
"
}"

Rendered Output ▾

AI

Seguro

Figura 3.11 Validación de Integridad

Elaboración propia

5. Ejecución sobre base. Se usa un agente con una función de consulta predefinida con las rutas de las tablas tal como se observa en la Figura 3.12.

La ejecución de la consulta se ilustra en la Figura 3.13.

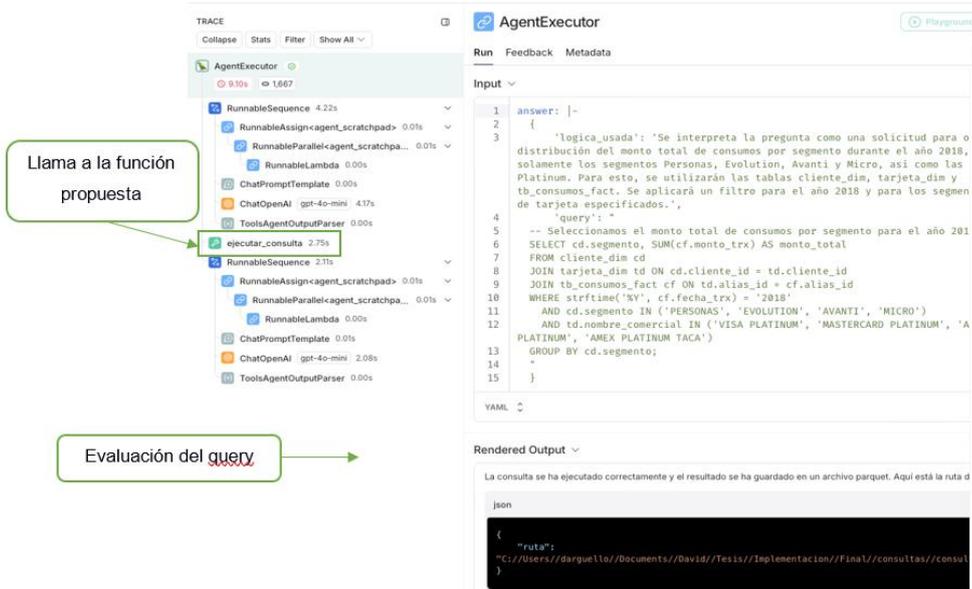


Figura 3.12 Llamar a la función para ejecutar consulta

Elaboración propia

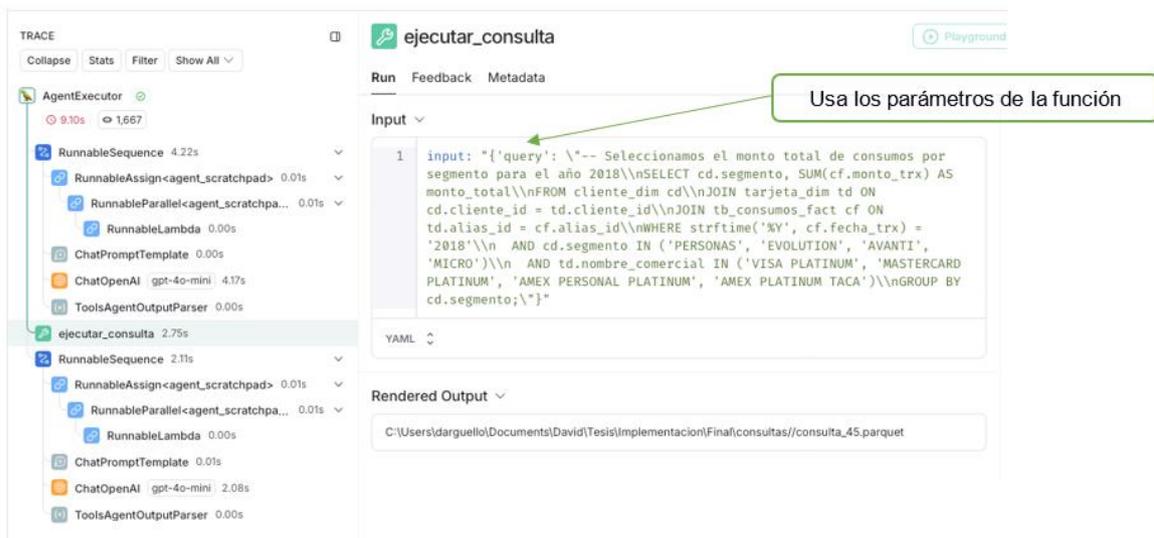


Figura 3.13 Ejecutar consulta

Elaboración propia

6. Selección de gráfico. Con la Figura 3.14 se observa cómo se usa un agente con las funciones gráficas disponibles, se le incluye la pregunta de usuario, el query para que infiera el esquema de los datos y la lógica con que se resuelve la pregunta para asimilar con las mejores prácticas de visualización.

En la Figura 3.15 se demuestra la selección del gráfico.

The screenshot displays the ChatOpenAI interface with a trace of an agent's execution. The trace shows the following steps:

- AgentExecutor (2.23s, 547)
- RunnableSequence (2.17s)
 - RunnableAssign<agent_scratchpad> (0.01s)
 - RunnableParallel<agent_scratchpa...> (0.01s)
 - RunnableLambda (0.00s)
 - ChatPromptTemplate (0.00s)
 - ChatOpenAI (gpt-4o-mini, 2.14s)
 - ToolsAgentOutputParser (0.00s)
 - mostrar_grafico_barras (0.06s)

The 'Functions & Tools' section lists the following functions:

- mostrar_tabla
- mostrar_numero_crudo
- mostrar_grafico_barras (CALLED)
- mostrar_grafico_dispersion
- mostrar_grafico_lineas

The 'Input' section contains the following text:

SYSTEM

Tienes una consulta de usuario, la lógica que se usó, el query que se usó para generar la consulta, la ruta de donde está el archivo
Tu rol es poder definir cual es la mejor manera de mostrar los resultados basado en las mejores prácticas de StoryTelling
Pistas:
Si el dato es un numero crudo, muéstralo como un numero
Si se piden distribuciones, lo más recomendado son barras
Si se piden evoluciones en el tiempo, lo más recomendado son líneas

Importantel
* Solo haz una llamada de funciones, la que creas mas adecuada pero SOLO UNA FUNCION UNA SOLA VEZ.

HUMAN

Esta es la pregunta del usuario ¿Cómo se distribuye el monto total de consumos por segmento durante el 2018? Solo quiero la información de clientes de segmentos Personas Evolution Avanti y Micro que sean tarjetas Platinum., este es la respuesta: {
"logica_usada": "Se interpreta la pregunta como una solicitud para obtener la distribución del monto total de consumos por segmento durante el año 2018, considerando solamente los segmentos Personas, Evolution, Avanti y Micro, así como las tarjetas Platinum. Para esto, se utilizarán las tablas cliente_dim, tarjeta_dim y tb_consumos_fact. Se aplicará un filtro para el año 2018 y para los segmentos y tipos de tarjeta especificados.",
"query": "
-- Seleccionamos el monto total de consumos por segmento para el año 2018
SELECT cd.segmento, SUM(cf.monto_trx) AS monto_total
FROM cliente_dim cd
JOIN tarjeta_dim td ON cd.cliente_id = td.cliente_id
JOIN tb_consumos_fact cf ON td.alias_id = cf.alias_id
WHERE strftime('%Y', cf.fecha_trx) = '2018'
AND cd.segmento IN ('PERSONAS', 'EVOLUTION', 'AVANTI', 'MICRO')
AND td.nombre_comercial IN ('VISA PLATINUM', 'MASTERCARD PLATINUM', 'AMEX PERSONAL PLATINUM', 'AMEX PLATINUM TACA')
GROUP BY cd.segmento;
"}
} y esta es la ruta del archivo La consulta se ha ejecutado correctamente y el resultado se ha guardado en un archivo parquet. Aquí está la ruta del archivo:
{}
"ruta":
"C://Users//darguello//Documents//David//Tesis//Implementacion//Final//consultas//consulta_45.parquet"
} ...

Annotations in the image:

- A box labeled "Mejores prácticas de visualización" points to the system prompt.
- A box labeled "Contexto" points to the human query.
- A box labeled "Funciones disponibles" points to the list of functions.
- An arrow points from the "mostrar_grafico_barras" function in the list to its corresponding step in the trace.

Figura 3.14 Llamar a la función para seleccionar gráfico

Elaboración propia

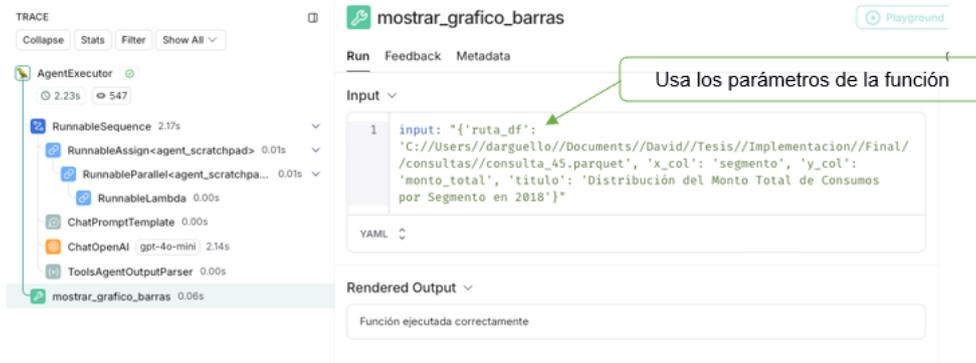


Figura 3.15 Selección gráfico

Elaboración propia

7. Presentación de resultados. Según la ejecución del agente, los gráficos se muestran usando la renderización de nativa de streamlit, así como se ilustra en la Figura 3.16.

Distribución del Monto Total de Consumos por Segmento en 2018

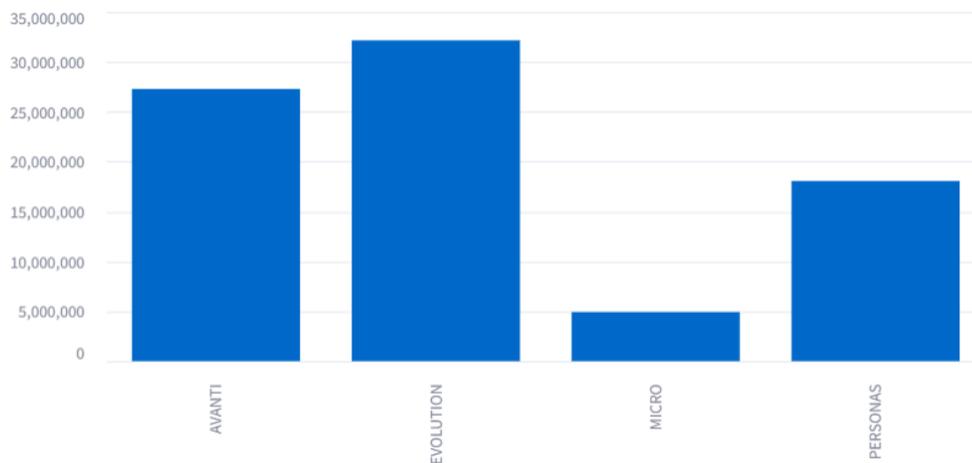


Figura 3.16 Presentación de resultados

Elaboración propia

3.4 Métricas y Comunicación de Resultados

Para evaluar este proceso, se van a usar las siguientes métricas:

- Diferencia entre tiempos de respuesta. Cuánto varía el tiempo de respuesta con el flujo actual contra cuánto demoraría el flujo propuesto.
- Precisión de las consultas generadas. Vamos a evaluar si el query que genera el proceso tiene el mismo objetivo que el query que generaría un oficial de analítica comercial que fuera a contestar la pregunta del usuario.
- Satisfacción del usuario. Si el usuario se siente satisfecho con la respuesta (en términos de claridad de la respuesta y visualización empleada). La Figura 3.17 demuestra un ejemplo de la toma de encuesta utilizada.

Te sientes satisfecho con el resultado?

Selecciona una opción:

- Sí
 No

Enviar respuesta de satisfacción

Figura 3.17. Ejemplo de la toma de encuesta

Elaboración propia

CAPÍTULO 4

4. ANÁLISIS DE RESULTADOS

El área de analítica, actualmente se organiza por Sprints donde cada área comercial tiene una cantidad de story points basados en la capacidad del equipo. Se tomaron datos de los sprints del mes de septiembre de 2024 (que son de martes a martes, por lo que abarca del martes 3 de septiembre hasta el martes 1 de octubre).

Enfocándonos en el equipo de Medios de Pago, se tomaron los datos de sus requerimientos a través del portal (SIGD) y los datos de tiempo de duración de sus tickets abiertos.

Oficial	Sprint1	Sprint2	Sprint3	Sprint4	Total
Oficial 1	2	5	3	1	11
Oficial 2	1	3	4	1	9
Oficial 3	3	2	5	2	12
Oficial 4	5	1	4	3	13
Total	11	11	16	7	45

Tabla 4.1. Tickets resueltos

Elaboración propia

En la Tabla 4.1 hemos recogido cuántos requerimientos se han resuelto para cada oficial del equipo de medios de pago, son cuatro oficiales por cada sprint del estudio. En total hemos recogido 45 pedidos de información.

Para medir la precisión de las consultas generadas, hemos excluido siete consultas que sufrían de ambigüedad por parte del usuario, dígame que había que indagar más en la pregunta que puso en el ticket para poder entender qué realmente necesitaba; para el resto de consultas se evaluó si el Query generado por el sistema coincidía con el Query creado por un oficial de analítica comercial.

En la Tabla 4.2 mostramos los resultados de precisión:

Oficial	Sprint1	Sprint2	Sprint3	Sprint4	Total
Oficial 1	50%	80%	100%	100%	82%
Oficial 2	100%	100%	75%	100%	89%
Oficial 3	67%	100%	100%	100%	92%
Oficial 4	20%	100%	100%	100%	69%
Total	45%	91%	94%	100%	82%

Tabla 4.2 Precisión del sistema RAG por sprint por oficial comercial

Elaboración propia

Aquí podemos ver que el flujo tiene una precisión global del 82%, lo que significa que 8 de cada 10 tickets pudieran ser resueltos de forma correcta usando este sistema propuesto basado en RAG.

Para tomar el tiempo, al momento de cerrar el ticket con el equipo comercial, decidimos hacer la misma consulta con nuestro sistema y llevar registro de cuánto tomó cerrar el ticket desde su apertura y cuánto tomó resolver la pregunta comercial con nuestro sistema, ambos los tabulamos en minutos para poder comparar los resultados.

Tendencia	Flujo Actual (minutos)	Flujo Propuesto (minutos)	Variación (%)
promedio	1.671	1,61	-99,90%
mediana	1.699	1,83	-99,89%
mínimo	176	0,05	-99,97%
máximo	2.797	2,90	-99,90%

Tabla 4.3 Medidas de tendencia de tiempos entre flujos

Elaboración propia

En base a los resultados obtenidos, la Tabla 4.3 muestra las medidas de tendencia de los tiempos de respuesta en flujo actual contrastados contra el flujo propuesto. Como los tickets se atienden por prioridad FIFO y hay más áreas aparte de medios de pago, puede que ingresen un ticket pero no se le de prioridad hasta acabar el que estaba en proceso, el flujo actual tiene un tiempo promedio de 1.671 minutos (~1día) mientras con el flujo propuesto toma 1,61 minutos el proceso de auto atenderse en promedio.

El sistema propuesto tiene una reducción del 99.9% del tiempo ya que caería en una categoría de autogestión, donde se saca de la ecuación el proceso de abrir un ticket, y ser sujetos de la disponibilidad de carga del equipo de analítica comercial.

CONCLUSIONES

Durante este proyecto, se consolidó la información transaccional de tarjetas en un diagrama estrella usando una perspectiva de crear un cubo de información, como los que se implementan en SSAS, pero se almacenó en archivos parquet para aprovechar sus beneficios de robustez de la mano de DuckDB. Lo cual fue útil para poder correr sentencias de SQL directamente a los archivos parquet.

En base los tiempos de respuesta y la comparación de los mismos entre los flujos, el sistema basado en RAG que se desarrolló reduce el tiempo de espera en un 99.9% de los oficiales comerciales, permitiéndoles autoatenderse sobre sus dudas sobre información relacionadas al área de Medios de Pago.

A través de los resultados observados también hemos medido la precisión de las respuestas, y hemos visto que en 8 de cada 10 veces, el sistema basado en RAG llega a la respuesta correcta, lo que da más confianza al área comercial sobre los resultados obtenidos. Este proyecto propone una alternativa para las herramientas de Business Intelligence tradicionales, donde usando LLM reducimos la burocracia actual democratizando la información hacia el área comercial y reduciendo los tiempos de espera.

RECOMENDACIONES

Para mantener este sistema basado en RAG durante el tiempo, se recomienda que el cubo de información se actualice de manera diaria, se pueden usar herramientas orquestadoras como airflow para ir insertando día a día las transacciones a los archivos parquet, de la mano de ir validando si los metadatos y el ETL que se ejecuta durante el tiempo se lo tiene que ir modificando por nuevos campos en la tabla transaccional, o nuevas consideraciones que afecten la validez de los datos de las tablas del diagrama estrella. Para el flujo como tal, el apartado de text-to-sql puede sacar mucho provecho de las nuevas implementaciones que haga LangChain, ya que no hace mucho no existían los agentes y con la implementación de estos se pudo cambiar por completo el flujo, así mismo pueden sacar nuevas versiones y es recomendable estar al pendiente y las compatibilidades con la API de OpenAI. Por último también ir revisando constantemente los queries que se generan para ir dando un double check a la información que la red comercial está usando para tomar sus decisiones y estar atentos cuando haya un error comunicarlo lo más pronto posible para no actuar de forma tardía.

BIBLIOGRAFÍA

- Arslan, M., & Cruz, C. (2024). Business-RAG: Information Extraction for Business Insights. *Proceedings of the International Conference on Smart Business Technologies (ICSBT 2024)*, (págs. 88-94). Obtenido de <https://www.scitepress.org/Papers/2024/128128/128128.pdf>
- AWS AMAZON. (s.f.). *What is RAG (Retrieval-Augmented Generation)?* Obtenido de AWS AMAZON WEBSITE: <https://aws.amazon.com/what-is/retrieval-augmented-generation/>
- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., . . . Amodei, D. (2020). Language Models are Few-Shot Learners. arXiv. doi:arXiv:2005.14165
- Jurafsky, D., & Martin, J. H. (2000). *Speech and Language Processing*. Prentice Hall.
- Kimball, R., & Ross, M. (2019). *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*. Wiley.
- Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., . . . Kiela, D. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems.*, 33, 9459-9474.
- Marikar, M. S., & Bandara, H. D. (2020). An Analysis of Data Driven, Decision-Making Capabilities. Sri Lanka: arXiv. doi:arXiv:2007.01862
- Marvin, G., Hellen, N., Jjingo, D., & Nakatumba-Nabende, J. (2023). Prompt Engineering in Large Language. *International Conference on Data Intelligence and Cognitive Informatics* (pp. 387-402). Singapore: Springer Nature Singapore. Retrieved from https://www.researchgate.net/publication/377214553_Prompt_Engineering_in_Large_Language_Models
- Minaee, S., Mikolov, T., Nikzad, N., Chenaghlu, M., Socher, R., Amatriain, X., & Gao, J. (2024). Large Language Models: A Survey. arXiv. doi:arXiv:2402.06196
- Raasveldt, M., & Mühleisen, H. (2019). DuckDB: an Embeddable Analytical Database. *Proceedings of the 2019 ACM SIGMOD International Conference on Management of Data*. Amsterdam. Obtenido de <https://mytherin.github.io/papers/2019-duckdbdemo.pdf>
- Richards, T. (2023). *Streamlit for Data Science: Create interactive data apps in Python* (2nd ed.). Packt Publishing Ltd.

Setty, S., Thakkar, H., Lee, A., Chung, E., & Vidra, N. (2024). Improving Retrieval for RAG based Question Answering Models on Financial Documents. arXiv. doi:arXiv:2404.07221

Suehring, S. (2001). *MySQL Bible*. Wiley.

Yepes, A. J., You, Y., Milczek, J., Laverde, S., & Li, L. (2024). Financial Report Chunking for Effective Retrieval Augmented Generation. Sacramento, USA: arXiv. doi:arXiv:2402.05131

APÉNDICES

Funcionalidades de consulta

```
# Cargar las variables de entorno
```

```
from dotenv import load_dotenv
```

```
load_dotenv()
```

```
# Importar las librerías necesarias de LangChain y OpenAI
```

```
from langchain.prompts import ChatPromptTemplate
```

```
from langchain.schema.output_parser import StrOutputParser
```

```
from langchain.schema.runnable import RunnableParallel, RunnableLambda
```

```
from langchain_openai import ChatOpenAI
```

```
from langchain.agents import AgentExecutor, create_tool_calling_agent
```

```
from langchain.tools import tool
```

```
import os
```

```
import pandas as pd
```

```
import duckdb
```

```
# Crear el modelo ChatOpenAI
```

```
model_4omini = ChatOpenAI(model="gpt-4o-mini")
```

```
# Combine pros and cons into a final review
```

```
def combinar_contexto(clientes, tarjetas, comercios):
```

```
    return f"Clientes://{clientes}\\n\\nTarjetas://{tarjetas}\\n\\nComercios://{comercios}"
```

```
@tool
```

```
def ejecutar_consulta(query):
```

```
    """
```

```
    Esta función recibe un query compatible con DuckDB, lo ejecuta, deposita la consulta en un archivo parquet y devuelve la ruta donde se guardó la data
```

```
    params:
```

```
    * query:str -> el query que se quiere ejecutar
```

returns:

* ruta:str -> la ruta donde se depositó la consulta

"""

```
query_final = (query
                .replace("tb_consumos_fact",
"C://Users//darguello//Documents//David//Tesis//Implementacion//Final//tb_fact.parquet"
                )
                .replace("comercio_dim",
"C://Users//darguello//Documents//David//Tesis//Implementacion//Final//tb_comercios.p
arquet")
                .replace("tarjeta_dim",
"C://Users//darguello//Documents//David//Tesis//Implementacion//Final//tb_tarjetas.parq
uet")
                .replace("cliente_dim",
"C://Users//darguello//Documents//David//Tesis//Implementacion//Final//tb_clientes.par
quet")
                )
```

Define la ruta de la carpeta

carpeta

=

```
r"C:\Users\darguello\Documents\David\Tesis\Implementacion\Final\consultas"
```

Lista los archivos de la carpeta

```
archivos = os.listdir(carpeta)
```

Filtra los archivos que tienen el prefijo "consulta_" y terminan con ".parquet"

```
archivos_consulta = [archivo for archivo in archivos if archivo.startswith("consulta_")
and archivo.endswith(".parquet")]
```

Extrae el número secuencial de cada archivo

```
numeros = [int(archivo.split("_")[1].split(".")[0]) for archivo in archivos_consulta]
```

```

# Determina el siguiente número secuencial
if numeros:
    siguiente_numero = max(numeros) + 1
else:
    siguiente_numero = 1

# Genera el siguiente nombre de archivo secuencial
nuevo_archivo = f"{carpeta}//consulta_{siguiente_numero}.parquet"

(duckdb.sql(query_final).df()
.to_parquet(nuevo_archivo, index = False))
return nuevo_archivo

```

validar_pregunta = [

("system", ""

Eres un asistente de inteligencia de negocios.

Trabajas en Banco Guayaquil, el 2do mas grande de Ecuador, eres un experto en el negocio de medios de pago en Banco Guayaquil, este negocio trata de comercios, tarjetas y clientes tarjetahabientes.

Tu rol es determinar si una pregunta de usuario tiene que ver con temas del negocio de medios de pago o no.

Las acciones clave de este negocio son:

- Consumen: termino relacionado a los clientes haciendo transacciones con sus tarjetas

- Facturacion: el monto total del balance de las tarjetas de credito/debito

- Establecimientos: los comercios que aceptan tarjetas de credito/debito

- Tarjetahabientes: los clientes que tienen tarjetas de credito/debito

Recuerda que tu cliente directo es un oficial comercial del negocio de medios de pago, no los clientes del banco. No tienes acceso informacion confidencial de los clientes.

Contesta con una sola palabra: 'Comercial' o 'No Comercial'.

"""),

```
("human", "Esta es la pregunta del usuario: '{user_prompt}"),  
]
```

```
plantilla_validar_pregunta = ChatPromptTemplate.from_messages(validar_pregunta)
```

```
# Creo la cadena inicial para validar la pregunta del usuario
```

```
step1_chain = plantilla_validar_pregunta | model_4omini | StrOutputParser()
```

```
prompt_tabla_clientes = """
```

```
Eres un experto en atención a clientes comerciales, dominas la información del negocio  
de Medios de Pago en Banco Guayaquil
```

```
Dado el contexto de la pregunta, debes encontrar los clientes relacionados para contestar  
la pregunta
```

```
Tienes esta información a la mano:
```

```
* A los clientes se los clasifica por segmento y región del cliente
```

```
* Solo existen tres regiones para los clientes: "SUR", "CENTRO", "NORTE"
```

```
* Solo existen estos segmentos estratégicos del banco: 'MICRO', 'PERSONAS',  
'EVOLUTION', 'AVANTI', 'PYME', 'ADVANCE EMPRESAS', 'EMPRESARIAL',  
'CORPORATIVO', 'INSTITUCIONAL'
```

```
Tarea:
```

```
Tu objetivo es analizar la pregunta del usuario comercial, y enfocarte unicamente en la  
parte de la pregunta que se relacione a los clientes como tal
```

```
Se espera que contestes con una referencia de a qué se pueda referir el usuario con su  
pregunta
```

```
Consideraciones:
```

```
* Si la pregunta no tiene nada relacionado a una consulta de clientes, devuélvela en  
blanco
```

```
Ejemplo:
```

usuario: "cuantos clientes avanti del portafolio platinum consumen en los supermercados la favorita"

idea: no tienes que enfocarte en la pregunta completa, solo identificar lo que puede estar relacionado a clientes que es "avanti"

en base a las regiones y segmentos que conoces, devuelve los datos relacionados a los clientes que sería: "segmento avanti"

"""

Define cons analysis step

```
def conseguir_clientes(comercial_input):
    clientes_template = ChatPromptTemplate.from_messages(
        [
            ("system", prompt_tabla_clientes),
            ("human", "Esta es la pregunta comercial {comercial_input}"),
        ]
    )
    return clientes_template.format_prompt(comercial_input=comercial_input)
```

prompt_tabla_tarjetas = """

Eres un experto en atención a clientes comerciales, dominas la información del negocio de Medios de Pago en Banco Guayaquil

Dado el contexto de la pregunta, debes encontrar la/las tarjetas relacionadas para contestar la pregunta

Tienes esta información a la mano:

* Solo existen dos tipos de tarjetas: "A" de adicionales y "P" de principales

* Solo existen estas tarjetas dentro del parque de clientes:

```
'AMEX CREDITO ORO',
'AMEX CREDITO VERDE',
'VISA PLATINUM',
'AMEX BUSINESS',
'MASTERCARD PLATINUM',
```

'GRUPO AFINIDAD IDE',
'AMEX LIFEMILES CLASICA',
'VISA SIGNATURE LIFEMILES',
'AMEX PERSONAL PLATINUM',
'AMEX LIFEMILES GOLD',
'AMEX PERSONAL PLATINUM - METAL',
'AMEX PORTA CLASICA',
'VISA TACA ORO',
'VISA SIGNATURE',
'VISA EMPRESARIAL PERSONA',
'MASTERCARD PRIMA',
'MASTERCARD CLASICA',
'AMEX PERSONAL GREEN',
'AMEX PLATINUM TACA',
'AMEX CREDITO BLUE',
'AMEX PORTA ORO',
'AMEX TACA ELITE',
'VISA EMPRESARIAL',
'AMEX SUPERMAXI PLATINUM',
'AMEX PERSONAL ORO',
'VISA CLASICA',
'VISA',
'VISA MANCHESTER UNITED',
'AMEX AMEX AADVANTAGE ELITE',
'AMEX AMEX AADVANTAGE PLATINUM',
'AMEX SUPER PUNTOS',
'AMEX MALL DEL RIO',
'VISA ORO',
'MASTERCARD ORO',
'AMEX QUICENTRO SHOPPING',
'AMEX BTA',
'AMEX BUSINESS LINK',
'AMEX PLATINUM CREDITO',

'AMEX BLUE BOX CREDITO',
'AMEX CUPO CERO GARANTIA',
'AMEX BLUE JOVENES',
'VISA TACA CLASICA',
'AMEX CHEVROLET CLASICA',
'AMEX BUSINESS PERSONA',
'AMEX SUPERMAXI CLASICA',
'VISA TACA PLATINUM',
'AMEX CHEVROLET PLATINUM',
'VISA ZERO'

Estos nombres son compuestos MARCA + " " + NOMBRE COMERCIAL

Tarea:

Tu objetivo es analizar la pregunta del usuario comercial, y enfocarte unicamente en la parte de la pregunta que se relacione a las tarjetas como tal

Se espera que contestes con una referencia de a qué se pueda referir el usuario con su pregunta

Consideraciones:

- * Si la pregunta no tiene nada relacionado a una consulta de tarjetas, devuelvela en blanco
- * Los terminos en los nombres comerciales son bilingues, recuerda que pueden decirte "oro" pero el nombre real es "gold" y debes buscar ambas coincidencias
- * Solo se tienen 3 marcas, AMEX - VISA - MASTERCARD
- * Toda la respuesta, de existir, debe venir en mayúscula
- * Recuerda incluir todos los nombres relacionados

Ejemplo:

usuario: "cuantos clientes del portafolio platinum consumen en los supermercados la favorita"

idea: no tienes que enfocarte en la pregunta completa, solo identificar lo que puede estar relacionado a tarjetas que es "portafolio platinum"

en base a la lista de tarjetas que conoces, devuelve todos los nombres platinum indistintamente si es credito/personal o la marca como tal, sino todas las que mencionen platinum

```
"""
```

```
# Define cons analysis step
```

```
def conseguir_tarjetas(comercial_input):
```

```
    tarjetas_template = ChatPromptTemplate.from_messages(
```

```
        [
```

```
            ("system", prompt_tabla_tarjetas),
```

```
            ("human", "Esta es la pregunta comercial {comercial_input}"),
```

```
        ]
```

```
    )
```

```
    return tarjetas_template.format_prompt(comercial_input=comercial_input)
```

```
prompt_tabla_comercios = """
```

Eres un experto en base de datos y en buscar los comercios relacionados en base a la pregunta de un usuario.

Dado el contexto de la pregunta, debes encontrar el/los comercio/s que sean necesarios para contestar la pregunta a través de consultas SQL

Tienes esta metadata:

Tabla: comercio_dim

Descripción: esta tabla contiene la información relacionada a los comercios

Campos:

-ruc_comercio: VARCHAR = Este campo contiene el RUC de la empresa (Registro Único de Contribuyente)

-nombre_empresa: VARCHAR = Este campo contiene el nombre de la empresa (todo en mayúsculas) se puede considerar la razón social

-categoria: VARCHAR = Este campo hace referencia a un giro/clasificación de esta empresa.

- Los posibles valores de clasificación son:

['SERVICIOS FINANCIEROS', 'COMERCIO EN GENERAL',
'COMERCIO / SUPERMERCADOS', 'OTROS SERVICIOS',
'TRANSPORTE DE PASAJEROS', 'INDUSTRIA FARMACEUTICA',
'TURISMO-HOTELES-RESTAURANTES', 'TELECOMUNICACIONES',
'COMERCIO / ARTICULOS IMPORTADOS', 'OTROS TRANSPORTE',
'COMERCIO / ELECTRODOMESTICOS Y EQUIPOS',
'CONSTRUCCION E INFRAESTRUCTURA', 'TRANSPORTE Y
ALMACENAMIENTO',
'MAQUINARIA Y EQUIPO AGRICOLA E INDUSTRIAL ',
'EQUIPOS Y MATERIALES DE CONSTRUCCION', 'SERVICIOS DE SALUD',
'OTRAS INDUSTRIAS MANUFACTURERAS', 'SERVICIOS DE
ENTRETENIMIENTO',
'COMERCIO / REPUESTOS AUTOMOTRIZ', nan, 'AUTOMOTRIZ',
'PETROLEO Y GAS', 'PRODUCTOS ALIMENTICIOS', 'INSUMOS AGRICOLAS',
'TRANSPORTE DE CARGA', 'METALES Y PRODUCTOS DEL METAL', 'MADERA',
'OTROS CULTIVOS', 'TRANSPORTE NAVIERO/ PUERTOS',
'MAQUINARIA Y EQUIPO AGRICOLA E INDUSTRIAL', 'CAMARON',
'PROVEEDORES DEL ESTADO EN SERVICIOS', 'MINAS Y CANTERAS',
'CAÑA DE AZUCAR', 'INDUSTRIA DE BEBIDAS', 'AVICOLA',
'INDUSTRIA FLORICOLA', 'INDUSTRIA DE LA PESCA', 'CACAO',
'GANADERO', 'ARROZ', 'BANANO Y CADENA DE ABASTECIMIENTO']

-actividad_economica: VARCHAR = Este campo tiene el detalle de la actividad económica a la que se dedica la empresa

Consideraciones:

* El query que generes debe funcionar en DuckDB

* Tu respuesta debe ser el/los comercios relacionados a la consulta

* Si la pregunta no tiene nada relacionado a una consulta de comercios, devuélvela en blanco

* Si un RUC se repite, solo escoge un registro por RUC

Ejemplo:

usuario: "cuantos clientes consumen en los supermercados la favorita"

idea: no tienes que enfocarte en la pregunta, solo identificar lo que puede estar relacionado a comercios que es "supermercados la favorita"

Comienza siempre a nivel del nombre de la empresa:

el posible query que generes seria "select distinct ruc_comercio, nombre_empresa, actividad_economica, categoria FROM comercio_dim WHERE nombre_empresa like '%FAVORITA%'"

Luego valida el tema del contexto adicional como categoria o actividad economica a ver si esta relacionada y en base a las respuestas mira si hay algo relacionado a supermercados

Se necesita que la respuesta de existir venga relacionada a un RUC

Valida que en la respuesta no venga repetido el ruc, de ser este el caso, toma cualquier registro que contenga los campos llenos y muestra una sola respuesta por ruc

```
"""
```

```
@tool()
```

```
def consultar_base_comercios(query:str) -> dict:
```

```
    """
```

```
    Esta función recibe un query para hacer una consulta en la base de comercios en parquet usando duckdb
```

```
    params:
```

```
    * query:str -> el query que se quiere ejecutar
```

```
    returns:
```

```
    * respuesta:dict -> el resultado del query en formato diccionario
```

```
    """
```

```
    query_consulta
```

```
=
```

```
    query.replace('comercio_dim', "C://Users//darguello//Documents//David//Tesis//Implementacion//Final//tb_comercios.parquet")
```

```
    respuesta = duckdb.sql(query_consulta).df().to_dict()
```

```
    return respuesta
```

```

messages = [
    ("system", prompt_tabla_comercios),
    ("human", "Esta es la pregunta del usuario {commercial_question}"),
    ("assistant", "Estoy procesando la información..."),
    ("assistant", "{agent_scratchpad}")
]

prompt_template_comercios = ChatPromptTemplate.from_messages(messages)

tools = [consultar_base_comercios]
# This function sets up an agent capable of calling tools based on the provided prompt.
agent_comercios = create_tool_calling_agent(
    llm=model_4omini, # Language model to use
    tools=tools, # List of tools available to the agent
    prompt=prompt_template_comercios, # Prompt template to guide the agent's
responses
)

# Create the agent executor
agent_executor_comercios = AgentExecutor.from_agent_and_tools(
    agent=agent_comercios, # The agent to execute
    tools=tools, # List of tools available to the agent
    verbose=True, # Enable verbose logging
    handle_parsing_errors=True, # Handle parsing errors gracefully
)

def obtener_contexto(pregunta):
    """
    Esta función recibe la pregunta del usuario y devuelve el contexto necesario
    relacionado en términos de tarjetas, clientes y comercios
    params:

```

* user_prompt:str -> la pregunta del usuario comercial relacionada al negocio de medios de pago

returns:

* contexto:str -> el contexto obtenido relacionado a tarjetas, clientes y comercios

"""

Simplify branches with LCEL

```
clientes_branch_chain = (  
    RunnableLambda(lambda x: conseguir_clientes(x) | model_4omini |  
StrOutputParser()  
)
```

```
tarjetas_branch_chain = (  
    RunnableLambda(lambda x: conseguir_tarjetas(x) | model_4omini |  
StrOutputParser()  
)
```

```
comercios_branch_chain = (  
    RunnableLambda(lambda x:  
agent_executor_comercios.invoke({"commercial_question": x}))  
)
```

Create the combined chain using LangChain Expression Language (LCEL)

```
chain = (  
    RunnableParallel(branches={"clientes": clientes_branch_chain, "tarjetas":  
tarjetas_branch_chain, "comercios": comercios_branch_chain})  
    | RunnableLambda(lambda x: combinar_contexto(x["branches"]["clientes"],  
x["branches"]["tarjetas"], x["branches"]["comercios"]['output']))  
)
```

Run the chain

```
result = chain.invoke({"commercial_question": pregunta})
```

return result

prompt_text_to_sql = ""

Contexto

Eres un experto en SQL y en análisis de datos. Tu tarea es generar consultas SQL basadas en preguntas en lenguaje natural, utilizando la estructura de datos proporcionada y el contexto de la pregunta.

Estructura de la base de datos

Tabla : tb_consumos_fact

Descripción : Esta tabla contiene todos los consumos hechos con tarjeta de crédito

Campos:

* fecha_trx : (DATETIME) es la fecha en la que se realizó el consumo

* alias_id : (VARCHAR) es el numero de tarjeta enmascarado

* comercio_id : (INT) es un codigo asignado a un comercio que pertenece a alguna empresa

* monto_trx : (FLOAT) es el monto total del consumo

* cantidad_trx : (FLOAT) es la cantidad de transacciones

Relaciones:

- tb_consumos_fact.alias_id = tarjeta_dim.alias_id

- tb_consumos_fact.comercio_id = comercio_dim.comercio_id

Tabla : comercio_dim

Descripción : Esta tabla es la dimensional de comercios, donde esta la informacion respectiva al comercio

Campos:

* comercio_id : (INT) es un codigo asignado a un comercio que pertenece a alguna empresa

* ruc_comercio : (VARCHAR) es el ruc de la empresa dueña de ese codigo de comercio

* nombre_empresa : (VARCHAR) es el nombre del comercio como tal, digase la razón social del RUC (en el caso de Ecuador, tenemos la compañía "La Favorita")

* actividad_economica : (VARCHAR) Este campo tiene el detalle de la actividad económica a la que se dedica la empresa

* categoria : (VARCHAR) Este campo hace referencia a un giro/clasificación de negocio de esta empresa

Relaciones:

- comercio_dim.comercio_id = tb_consumos_fact.comercio_id

Tabla : tarjeta_dim

Descripción : Esta tabla es la dimensional de las tarjetas, donde está la información relacionada al plástico y el id del dueño de la tarjeta

Campos:

* alias_id : (VARCHAR) es un string con un numero enmascarado de la tarjeta real

* tipo_plastico : (VARCHAR) es una marca para saber si ese plástico corresponde a una tarjeta adicional o una tarjeta principal

- Valores posibles:

+ "P" -> para tarjetas principales

+ "A" -> para tarjetas adicionales

* bin : (INT) es el bin de la tarjeta (Bank Identification Number)

* nombre_comercial : (VARCHAR) es el nombre comercial de la tarjeta (ejemplo: AMEX PERSONAL PLATINUM, etc...)

* cliente_id : (INT) es un codigo interno del banco para un cliente

Relaciones:

- tarjeta_dim.alias_id = tb_consumos_fact.alias_id

Tabla : cliente_dim

Descripción : Esta tabla es la dimensional de los clientes, donde esta su informacion demografica

Campos:

* cliente_id : (INT) es un codigo interno del banco para un cliente

* segmento : (VARCHAR) es el segmento estrategico del cliente, bajo el modelo de atencion del banco, se segmenta a los clientes año a año en base a su potencial

- valores posibles:

- + PERSONAS
- + EVOLUTION
- + AVANTI
- + MICRO
- + PYME
- + ADVANCE EMPRESAS
- + EMPRESARIAL
- + INSTITUCIONAL
- + CORPORATIVO

* region : (VARCHAR) es la region del cliente

- Valores posibles:

- + NORTE
- + CENTRO
- + SUR

* cupo : (FLOAT) es el cupo de tarjeta asignado al cliente

* per_aper : (INT) numero que hacer referencia al año y mes en que se le abrio la tarjeta a cliente por primera vez (YYYYMM)

Relaciones:

- cliente_dim.cliente_id = tarjeta_dim.cliente_id

Observaciones importantes

* El datamart solo contiene información de los años 2018 y 2019.

* Si no se especifica una fecha y la pregunta está relacionada con consumos, utiliza el último periodo disponible (es decir, el último periodo de facturación en formato YYYYMM).

* Si la respuesta a la pregunta no se puede obtener de las tablas disponibles, indica que no puedes contestar esa pregunta.

Instrucciones

1. Lee cuidadosamente la pregunta en lenguaje natural.

2. Verifica si la pregunta puede ser respondida con los datos disponibles.
3. Si la pregunta no puede ser respondida, indica claramente que no es posible contestarla con la información disponible.
4. Si la pregunta puede ser respondida, procede con los siguientes pasos:
 - a. Identifica las tablas y campos relevantes para la consulta.
 - b. Determina las relaciones necesarias entre las tablas.
 - c. Construye una consulta SQL que responda a la pregunta.
 - d. Asegúrate de incluir las condiciones necesarias en la cláusula WHERE.
 - e. Si es necesario, utiliza funciones de agregación (COUNT, SUM, AVG, etc.) y agrupa los resultados adecuadamente.
 - f. Si la pregunta está relacionada con consumos y no se especifica una fecha, incluye una subconsulta para obtener el último periodo de facturación disponible.

Formato de respuesta

La respuesta debe ser un diccionario con dos claves:

1. 'logica_usada': Explicación de cómo se interpretó la pregunta y cómo se va a contestar usando las tablas empleadas.
2. 'query': La consulta SQL que responde a la pregunta del usuario.

Ejemplo de formato de respuesta (devuelve un diccionario):

```
'logica_usada': 'Se interpretó la pregunta como... Para responderla, se utilizarán las tablas... Se aplicarán los siguientes filtros...',  
'query': 'SELECT ... FROM ... WHERE ...'
```

Si la pregunta no puede ser respondida, el diccionario debe tener este formato:

```
'logica_usada': 'No es posible responder a esta pregunta porque...',  
'query': None
```

Ejemplo de pregunta

"Cuántos clientes de la región sur consumieron en comisariatos el mes pasado?"

Ejemplo de respuesta

'logica_usada': 'Se interpreta la pregunta como una solicitud para contar clientes únicos de la región sur que han realizado consumos en comisariatos durante el último periodo de facturación disponible. Para esto, se utilizarán las tablas cliente_dim, tarjeta_dim, tb_consumos_fact y comercio_dim. Se aplicará un filtro para la región sur, otro para la categoría de comisariato, y se determinará el último periodo de facturación disponible.'

'query': "

-- Primero, obtenemos el último periodo de facturación disponible (formato YYYYMM)

WITH ultimo_periodo AS (

 SELECT MAX(strftime('%Y%m', fecha_trx)) AS max_periodo

 FROM tb_consumos_fact

)

-- Contamos clientes únicos de la región sur que han consumido en comisariatos en el último periodo

SELECT COUNT(DISTINCT cd.clienteid) AS clientes_sur_comisariato

FROM cliente_dim cd

JOIN tarjeta_dim td ON cd.cliente_id = td.cliente_id

JOIN tb_consumos_fact cf ON td.alias_id = cf.alias_id

JOIN comercio_dim cmd ON cf.comercio_id = cmd.comercio_id

CROSS JOIN ultimo_periodo up

WHERE cd.region = 'SUR'

 AND cmd.categoria = 'COMISARIATO'

 AND strftime('%Y%m', cf.fecha_trx) = up.max_periodo;

"

Tu tarea

Entender la pregunta del usuario usar referencialmente el contexto y egenerar un query adecuado que corra en DuckDB

Recuerda:

- Siempre usa la función obtener_contexto para que tengas referencia de los datos de la pregunta relacionados a tarjetas, clientes y comercios
 - Si la pregunta no puede ser respondida con los datos disponibles, indica claramente por qué en la 'logica_usada' y establece 'query' como None.
 - Si no se especifica una fecha para consumos, utiliza el último periodo disponible.
 - Los datos solo cubren los años 2018 y 2019.
 - La respuesta debe venir en formato diccionario, con las llaves al comienzo y al final
 - En la logica de la respuesta, no uses comillas para especificar detalles como "avanti", sino ponlas como avanti, aplica lo mismo para tarjeatas o nombres de comercio, sin comillas. Pero si comienza y acaba el string de la logica usando comillas simples
- """

```
messages = [
```

```
    ("system", prompt_text_to_sql),
```

```
    ("human", "Esta es la pregunta del usuario {commercial_question}"),
```

```
    ("human", "Este es el contexto {contexto}"),
```

```
]
```

```
text_to_sql_template = ChatPromptTemplate.from_messages(messages)
```

```
validar_integridad = [
```

```
    ("system", """
```

```
    Eres un experto en SQL.
```

Tu rol es determinar si un query es peligroso para la integridad de las tablas, el query solo puede ser de consulta, no debería tener truncates, delete, updates, etc...

Contesta con una sola palabra: 'Peligroso' o 'Seguro'.

```
"""),
("human", "Esta es el query: '{gpt_query}")),
]
```

```
plantilla_validar_query = ChatPromptTemplate.from_messages(validar_integridad)
```

```
# Creo la cadena inicial para validar la pregunta del usuario
```

```
integrity_chain = plantilla_validar_query | model_4omini | StrOutputParser()
```

```
def ejecutar_query(query):
```

```
    prompt_ejecucion_query = ""
```

```
    Eres un experto en SQL y DuckDB. Tu rol es extraer el query de un string y validar que funcione con DuckDB.
```

```
    Luego ejecuta la consulta.
```

```
    Devuelve un diccionario con la ruta de la consulta, con las llaves al comienzo y al final
```

```
    Ejemplo:
```

```
"ruta": "C://Users//darguello//Documents//David//Tesis//Implementacion//Final//consultas/
/consulta_5.parquet"
```

```
"""
```

```
messages = [
```

```
    ("system", prompt_ejecucion_query),
```

```
    ("human", "Este es el string {answer}"),
```

```
    ("assistant", "Estoy procesando la información..."),
```

```
    ("assistant", "{agent_scratchpad}")
```

```
]
```

```
prompt_template = ChatPromptTemplate.from_messages(messages)
```

```
tools = [ejecutar_consulta]
```

```
# This function sets up an agent capable of calling tools based on the provided prompt.
agent = create_tool_calling_agent(
    llm=model_4omini, # Language model to use
    tools=tools, # List of tools available to the agent
    prompt=prompt_template, # Prompt template to guide the agent's responses
)
```

```
# Create the agent executor
```

```
agent_executor = AgentExecutor.from_agent_and_tools(
    agent=agent, # The agent to execute
    tools=tools, # List of tools available to the agent
    verbose=True, # Enable verbose logging
    handle_parsing_errors=True, # Handle parsing errors gracefully
)
```

```
# Run the agent with a test query
```

```
response = agent_executor.invoke({"answer": query})
return response
```

```
# pregunta = 'Cuantos clientes Avanti o Evolution consumen en El Rosado con sus
tarjetas platinum'
```

```
def text_to_sql(pregunta):
```

```
    # Se valida si la pregunta es comercial o no:
```

```
    validez_pregunta = step1_chain.invoke({"user_prompt": pregunta})
```

```
    print(validez_pregunta)
```

```
    ruta_respuesta = "
```

```
    # Si la pregunta es valida, sigue el flujo
```

```
    if validez_pregunta.lower() == 'comercial':
```

```
        contexto = obtener_contexto(pregunta)
```

```
text_to_sql_prompt =
text_to_sql_template.invoke({"commercial_question":pregunta, "contexto":contexto})
result_sql_query = model_4omini.invoke(text_to_sql_prompt)
integridad = integrity_chain.invoke({'gpt_query':result_sql_query.content})
print(integridad)
if integridad.lower() == 'seguro':
    ruta_respuesta = ejecutar_query(result_sql_query.content)
else:
    ruta_respuesta = 'No se puede contestar esta pregunta. Reformular'
else:
    ruta_respuesta = 'No se puede contestar esta pregunta. Reformular'

return ruta_respuesta
```

Funcionalidades gráficas

```
# Cargar las variables de entorno
from dotenv import load_dotenv
load_dotenv()

# Importar las librerías necesarias de LangChain y OpenAI
from langchain.prompts import ChatPromptTemplate
from langchain.schema.output_parser import StrOutputParser
from langchain.schema.runnable import RunnableParallel, RunnableLambda
from langchain_openai import ChatOpenAI
from langchain.agents import AgentExecutor, create_tool_calling_agent
from langchain.tools import tool
import os

import pandas as pd
import duckdb
import streamlit as st

# Crear el modelo ChatOpenAI
model_4omini = ChatOpenAI(model="gpt-4o-mini")
### Funcionalidades de graficos

# Función para mostrar una tabla en Streamlit
@tool
def mostrar_tabla(ruta_df, titulo):
    """
    Muestra una tabla en Streamlit.

    Args:
        ruta_df (str): Ruta del archivo en formato parquet a leer.
        titulo (str): Título de la tabla que se mostrará en la visualización.
    """
    df = pd.read_parquet(ruta_df)
```

```
st.write(f"### {titulo}")
st.dataframe(df)
return 'Función ejecutada correctamente'
```

Función para mostrar un número en crudo en Streamlit

@tool

```
def mostrar_numero_crudo(dato, titulo):
```

```
    """
```

Muestra un número crudo en Streamlit.

Args:

dato: Valor que se quiere mostrar.

titulo (str): Título a mostrar sobre el dato.

```
    """
```

```
st.write(f"### {titulo}")
```

```
st.write(f"{dato}")
```

```
return 'Función ejecutada correctamente'
```

Función para mostrar un gráfico de barras en Streamlit

@tool

```
def mostrar_grafico_barras(ruta_df, x_col, y_col, titulo, color_col=None):
```

```
    """
```

Muestra un gráfico de barras en Streamlit.

Args:

ruta_df (str): Ruta del archivo en formato parquet a leer.

x_col (str): Columna del DataFrame para el eje X.

y_col (str): Columna del DataFrame para el eje Y.

titulo (str): Título del gráfico.

color_col (str, opcional): Columna para distinguir colores en el gráfico.

```
    """
```

```
df = pd.read_parquet(ruta_df)
```

```
st.write(f"### {titulo}")
```

```
if color_col:
    st.write(f"Color por: {color_col}")
st.bar_chart(df.set_index(x_col)[y_col])
return 'Función ejecutada correctamente'
```

Función para mostrar un gráfico de dispersión en Streamlit

@tool

```
def mostrar_grafico_dispersion(ruta_df, x_col, y_col, titulo, color_col=None):
```

```
    """
```

Muestra un gráfico de dispersión en Streamlit.

Args:

ruta_df (str): Ruta del archivo en formato parquet a leer.

x_col (str): Columna del DataFrame para el eje X.

y_col (str): Columna del DataFrame para el eje Y.

titulo (str): Título del gráfico.

color_col (str, opcional): Columna para distinguir colores en el gráfico.

```
    """
```

```
df = pd.read_parquet(ruta_df)
```

```
st.write(f"### {titulo}")
```

```
st.vega_lite_chart(df, {
```

```
    'mark': 'point',
```

```
    'encoding': {
```

```
        'x': {'field': x_col, 'type': 'quantitative'},
```

```
        'y': {'field': y_col, 'type': 'quantitative'}
```

```
    }
```

```
})
```

```
return 'Función ejecutada correctamente'
```

Función para mostrar un gráfico de líneas en Streamlit

@tool

```
def mostrar_grafico_lineas(ruta_df, x_col, y_col, titulo, color_col=None):
```

```
"""
```

Muestra un gráfico de líneas en Streamlit.

Args:

ruta_df (str): Ruta del archivo en formato parquet a leer.

x_col (str): Columna del DataFrame para el eje X.

y_col (str): Columna del DataFrame para el eje Y.

titulo (str): Título del gráfico.

color_col (str, opcional): Columna para distinguir colores en el gráfico.

```
"""
```

```
df = pd.read_parquet(ruta_df)
print(df.head())
st.write(f"### {titulo}")
st.line_chart(df.set_index(x_col)[y_col])
return 'Función ejecutada correctamente'
```

```
tools = [mostrar_tabla, mostrar_numero_crudo, mostrar_grafico_barras,
mostrar_grafico_dispersion, mostrar_grafico_lineas]
```

```
prompt_grafico = """
```

Tienes una consulta de usuario, la lógica que se usó, el query que se usó para generar la consulta, la ruta de donde está el archivo

Tu rol es poder definir cual es la mejor manera de mostrar los resultados basado en las mejores prácticas de StoryTelling

Pistas:

Si el dato es un numero crudo, muéstralo como un numero

Si te piden distribuciones, lo más recomendado son barras

Si te piden evoluciones en el tiempo, lo más recomendado son líneas

Importante!

* Solo haz una llamada de funciones, la que creas mas adecuada pero SOLO UNA FUNCION UNA SOLA VEZ.

```
"""
```

```
messages = [  
    ("system", prompt_grafico),  
    ("human", "Esta es la pregunta del usuario {commercial_question}, este es la  
respuesta: {answer} y esta es la ruta del archivo {ruta} "),  
    ("assistant", "Estoy procesando la información..."),  
    ("assistant", "{agent_scratchpad}")  
]
```

```
prompt_template_graficos = ChatPromptTemplate.from_messages(messages)
```

```
# This function sets up an agent capable of calling tools based on the provided prompt.
```

```
agent_comercios = create_tool_calling_agent(  
    llm=model_4omini, # Language model to use  
    tools=tools, # List of tools available to the agent  
    prompt=prompt_template_graficos, # Prompt template to guide the agent's responses  
)
```

```
# Create the agent executor
```

```
agent_executor_graficos = AgentExecutor.from_agent_and_tools(  
    agent=agent_comercios, # The agent to execute  
    tools=tools, # List of tools available to the agent  
    verbose=True, # Enable verbose logging  
    handle_parsing_errors=True, # Handle parsing errors gracefully,  
    max_iterations = 1  
)
```

```
def seleccionar_graficos(pregunta_usuario, respuesta_gpt, ruta):
```

```
# Run the agent with a test query
```

```
response = agent_executor_graficos.invoke({"commercial_question":
pregunta_usuario,
"answer": respuesta_gpt,
"ruta": ruta
})
return response
```

Aplicativo Principal

```
import streamlit as st
import time
from funciones_consultas import text_to_sql
from funciones_graficos import seleccionar_graficos
import pandas as pd

# Función para simular la escritura gradual de texto en Streamlit
def stream_data(logic_applied: str):
    for word in logic_applied.split(" "):
        for letter in word:
            yield letter
            time.sleep(0.05)
        yield " "
        time.sleep(0.1)

# Inicializar variables de estado
if 'satisfaccion_respondida' not in st.session_state:
    st.session_state['satisfaccion_respondida'] = False

if 'respuesta_mostrada' not in st.session_state:
    st.session_state['respuesta_mostrada'] = False

# Crear el formulario para ingresar la pregunta del usuario
form = st.form(key='text_input_form')
user_input = form.text_area(label='Ingresa tu pregunta',
                             disabled=st.session_state['respuesta_mostrada'])
submit_button = form.form_submit_button(label='Buscar',
                                         disabled=st.session_state['respuesta_mostrada'])

# Ejecutar la lógica solo si no se ha mostrado la respuesta y se hace clic en buscar
if submit_button and not st.session_state['respuesta_mostrada']:
    time.sleep(2)
```

```

st.write_stream(stream_data("Buscando información..."))

# Ejecutar la consulta y mostrar la respuesta
respuesta = text_to_sql(user_input)
# st.write(respuesta)
respuesta_graficos = seleccionar_graficos(user_input, respuesta['answer'],
respuesta['output'])
# st.write(respuesta_graficos)

# Almacenar que la respuesta ha sido mostrada
st.session_state['respuesta_mostrada'] = True

# Mostrar la pregunta de satisfacción después de que se muestre la respuesta
if st.session_state['respuesta_mostrada'] and not
st.session_state['satisfaccion_respondida']:
    st.write("Te sientes satisfecho con el resultado?")
    respuesta_satisfaccion = st.radio("Selecciona una opción:", ("Sí", "No"))

# Botón para confirmar la respuesta de satisfacción
if st.button("Enviar respuesta de satisfacción"):
    st.session_state['satisfaccion_respondida'] = True
    st.write(f"Gracias por tu respuesta: {respuesta_satisfaccion}")

# Permitir una nueva consulta al resetear las variables de estado
st.session_state['respuesta_mostrada'] = False
st.session_state['satisfaccion_respondida'] = False

# Limpiar el formulario para la siguiente consulta
st.rerun()

```

