



## MÁSTER EN DATA SCIENCE Y BUSINESS ANALYTICS ONLINE

# Agente Predictor de Variables Económicas y Análisis de Noticias con IA.

TFM elaborado por: Josué Quirós Quirós

Tutor de TFM: Juan Manuel Moreno Lampero

- San José, 29 de mayo de 2025 -



## RESUMEN

Con el campo de la Inteligencia Artificial en pleno apogeo, impulsado por los avances de los últimos años en el Procesamiento de Lenguaje Natural a manos de los Grandes Modelos de Lenguaje (LLMs, por sus siglas en inglés) cada año a partir de la salida de ChatGPT ha sido catalogado basado en la tecnología o aplicación que se le dará a este tipo de modelos, ofreciendo un cambio de paradigma sobre cómo desarrollamos nuestras actividades diarias. Este 2025 no es la excepción y muchos expertos lo han nombrado el año de los agentes.

Apalancado en los avances de los primeros meses del año en este tema, este trabajo de fin de máster consiste en el desarrollo de un agente que, mediante la obtención y preprocesamiento de datos asociados a distintas variables representativas de la economía costarricense, contribuya al análisis predictivo y prospectivo del futuro económico de este país centroamericano. Además, aprovechando las funcionalidades presentes para la obtención de información actualizada, se realiza una búsqueda automatizada de las principales noticias asociadas a los indicadores de interés, las cuales se sumarán a las proyecciones ya realizadas para servir de combustible a un LLM que realizará un análisis especializado.

Finalmente, de manera automatizada, a partir de los distintos insumos obtenidos y contruidos se generará un reporte que funcionará de insumo para distintos análisis técnicos, todo esto con la facilidad de estar integrado en una aplicación web amigable con el usuario.

## Índice de contenido

1.	Introducción y Antecedentes .....	7
1.1	Del Aprendizaje Automático Clásico a los Agentes de IA.....	7
1.1.1	Aprendizaje Automático .....	7
1.1.2	Aprendizaje Profundo .....	9
1.1.3	Procesamiento de Lenguaje Natural .....	11
1.1.4	Agentes Inteligentes.....	13
2.	Objetivos del Proyecto .....	16
3.	Materiales y Métodos .....	17
3.1	Librerías utilizadas.....	17
3.2	Metodología y Servicios Usados .....	20
4.	Resultados.....	28
5.	Limitaciones .....	46
6.	Conclusiones.....	47
7.	Referencias Bibliográficas.....	48
8.	Anexos.....	50
8.1	Repositorio del Proyecto.....	50
8.2	Código archivo funciones.py .....	51
8.3	Código archivo App.py .....	63

## Tabla de Ilustraciones

Figura 1. Tipos de aprendizaje. ....	8
Figura 2. Estructura clásica de una red neuronal artificial. ....	9
Figura 3. Ilustración de Red Neuronal Convolutacional. ....	10
Figura 4. Celda de una Red Neuronal Recurrente. ....	11
Figura 5. Arquitectura del Transformer. ....	12
Figura 6. Estructura de ejemplo de un Agente de IA. ....	14
Figura 7. Librerías para la estructuración del agente. ....	19
Figura 8. Librerías para el desarrollo de la herramienta interactiva. ....	19
Figura 9. Estructura script funciones.py. ....	20
Figura 10. Importación de las funciones Multiagente. ....	21
Figura 11. Formulario Suscripción de Usuario. ....	22
Figura 12. Modelos disponibles en GroqCloud. ....	23
Figura 13. Comparativa a nivel de Benchmarks. ....	24
Figura 14. Límites de uso para DeepSeek-R1. ....	24
Figura 15. Archivos para la Dockerización. ....	25
Figura 16. Estructura del Proyecto a nivel de archivos. ....	26
Figura 17. Contenedor creado y activo. ....	26
Figura 18. Detalle configuración Plan App Service. ....	26
Figura 19. Despliegue de imagen en App Service. ....	27
Figura 20. Grupo de Recursos creado. ....	27
Figura 21. Interfaz del Agente, vista preliminar. ....	28
Figura 22. Despliegue de variables disponibles. ....	29
Figura 23. Inclusión de API Key de Groq. ....	29
Figura 24. Información requerida para análisis autorregresivo. ....	30
Figura 25. Definición base de periodos a proyectar. ....	30
Figura 26. Información requerida para análisis regresión múltiple. ....	31
Figura 27. Indicadores disponibles para funcionar como predictores. ....	31
Figura 28. Método __init__ para la definición de atributos de la clase IndicatorData. ....	32
Figura 29. Método __init__ para la definición de atributos de la clase IndicatorDataM. ....	32
Figura 30. Método run para ejecución de la funcionalidad de IndicatorData. ....	33
Figura 31. Estructura inicial del Reporte Final. ....	33
Figura 32. Definición del diccionario con las variables seleccionadas. ....	34

Figura 33. Proceso de Iteración para obtención de datos. ....	34
Figura 34. Estructura inicial del Reporte Final del Análisis de Regresión Múltiple. ....	34
Figura 35. Funcionalidad PredictionAgent. ....	35
Figura 36. Sección Nro.1 de la Función PredictionAgentM. ....	36
Figura 37. Sección Nro.2 de la Función PredictionAgentM. ....	37
Figura 38. Sección Nro.3 de la Función PredictionAgentM, Secuencia Mensual. ....	38
Figura 39. Generación de proyecciones finales. ....	38
Figura 40. Primera Sección Reporte Análisis Autorregresivo. ....	39
Figura 41. Primera Sección Reporte Análisis Regresión Múltiple. ....	39
Figura 42. Funcionalidad ChartAgent. ....	40
Figura 43. Funcionalidad ChartAgentM. ....	40
Figura 44. Conversión del objeto fig a base64. ....	41
Figura 45. Comportamiento Real vs Proyección. ....	41
Figura 46. Definición y Ejecución de la Búsqueda de Noticias. ....	42
Figura 47. Definición Salida Estructurada. ....	42
Figura 48. User prompt. ....	43
Figura 49. Generación de Respuesta. ....	44
Figura 50. Estructura Parte Final del Reporte. ....	44
Figura 51. Análisis de Noticias del Agente. ....	45

# 1. Introducción y Antecedentes

En cada área del conocimiento, en cada espacio donde la automatización de tareas, la asistencia en la resolución de problemas y la creación de sistemas avanzados sea una posibilidad para mejorar el rendimiento personal y grupal, la Inteligencia Artificial (IA) se ha convertido en la herramienta que muchos quieren implementar. Desde la obtención de resultados desde un enfoque predictivo, hasta la generación de contenido mediante un enfoque generativo, los avances recientes en este campo de las ciencias de la computación han incrementado el interés de gran parte de la sociedad por entenderla, usarla y beneficiarse.

Con la ola más reciente de los agentes inteligentes, en este 2025 se han venido diseñando variedad de herramientas que permiten el desarrollo de estos asistentes avanzados que prometen ser el ayudante perfecto en distintas tareas, pero que también exponen ciertas profesiones a una completa automatización. No obstante, se encuentra más en peligro aquel individuo que no aprovecha estos nuevos avances para mejorar sus capacidades y ofrecer mejores resultados.

Una de las tantas tareas que puede verse favorecida es el análisis económico. Tradicionalmente, esta labor se ha basado en modelos matemáticos/estadísticos, experimentando una transformación significativa con la incorporación de herramientas basadas en Inteligencia Artificial. La capacidad reciente de procesar grandes volúmenes de datos prácticamente en tiempo real, así como identificar patrones complejos y generar proyecciones con altos grados de precisión ha llevado a la automatización y optimización de la toma de decisiones económicas, permitiendo la evaluación dinámica de variables macro y microeconómicas.

Estas herramientas no solo han permitido agilizar el proceso analítico, sino que también en la actualidad democratizan el acceso a modelos avanzados, facilitando su uso a profesionales y organizaciones que buscan mejorar su capacidad predictiva y competitividad en el mercado. Es por lo anterior que a través del desarrollo de este trabajo final de máster se busca disponibilizar un agente que se convierta en una herramienta más que continúe favoreciendo la toma de decisiones a partir del entendimiento profundo del comportamiento de las variables económicas y cómo estas se relacionan, potenciado por los avances actuales en los modelos generativos de Inteligencia Artificial.

## 1.1 Del Aprendizaje Automático Clásico a los Agentes de IA

### 1.1.1 Aprendizaje Automático

Los modelos clásicos de Aprendizaje Automático o Machine Learning desde hace varios años han sido el motor de muchas herramientas con las cuales hemos interactuado sin comprender con claridad

que se trataba de aplicaciones de Inteligencia Artificial, como los sistemas de recomendación y aplicaciones predictivas que permitían establecer como, por ejemplo, necesidades futuras de inventario o de mantenimiento preventivo de maquinaria.

Este tipo de IA implica la identificación de patrones a partir de datos históricos mediante técnicas estadísticas y computacionales para realizar predicciones o tomar decisiones sin necesidad de programación explícita. Abarcando diversas técnicas, como el análisis de regresión, los árboles de decisión, los bosques aleatorios y el aumento de gradiente, su principal fortaleza es el manejo de datos tabulares/estructurados, es decir, datos que pueden organizarse en filas y columnas de una hoja de cálculo o una tabla de base de datos. (Ramakrishnan, 2025)

Las tareas típicas que son cubiertas con este tipo de modelos son calificadas de acuerdo con el conocimiento que se tiene del resultado que se espera sea arrojado, por lo que si este es conocido son catalogados como aprendizaje supervisado y cuando se desconoce, pero se busca obtener algún tipo de comprensión sobre las relaciones o fenómenos que podrían explicar los datos, son llamados como aprendizaje no supervisado.

Asimismo, también existe una clasificación adicional, llamada aprendizaje por refuerzo, que básicamente representa un conjunto de técnicas para resolver problemas de decisión secuenciales de forma iterativa. En este tipo de aprendizaje un agente interactúa con un entorno mediante tres señales de estado, la primera describe el estado del entorno, la segunda permite al agente influenciar sobre el entorno y la tercera ofrece lo que comúnmente se conoce como recompensa, que determina la calidad de la acción realizada sobre el entorno. (Escandell Montero, 2014)

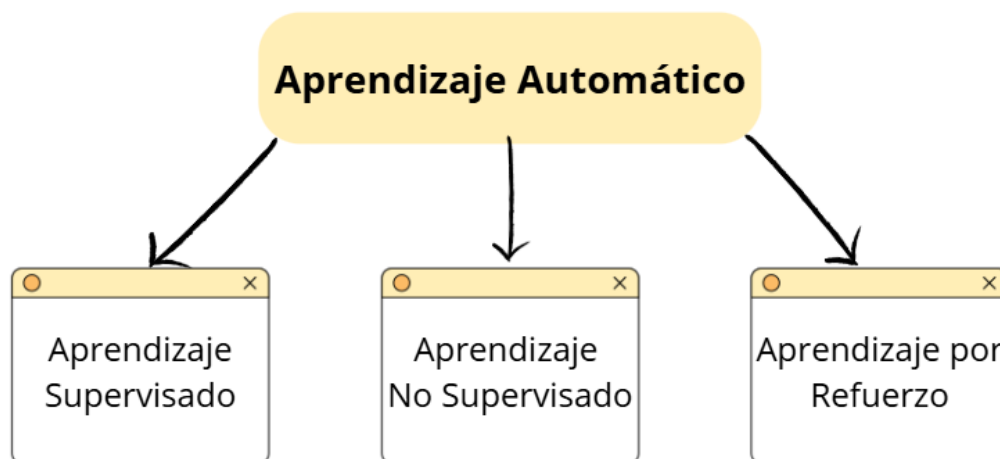


Figura 1. Tipos de aprendizaje. Fuente: Elaboración propia.

Dentro de las principales aplicaciones de los modelos de Aprendizaje Automático en lo que respecta al análisis económico y financiero se pueden destacar los siguientes:



- Predicción de indicadores macroeconómicos.
- Segmentación de consumidores.
- Detección de anomalías y crisis financieras.
- Optimización de Portafolios Financieros.

### 1.1.2 Aprendizaje Profundo

A partir del aumento en el volumen de los datos generados, disminución en los costos para su almacenaje y una mejora considerada en las capacidades de procesamiento, algoritmos que requerían grandes cantidades de datos empezaron a convertirse en la mejor opción para representar aquellos fenómenos o relaciones donde el nivel de precisión de los algoritmos clásicos era deficiente. Estos nuevos esquemas matemáticos fueron bautizados como redes neuronales artificiales por basarse en la estructura y el proceso de aprendizaje de sus homólogos presentes en el cerebro humano.

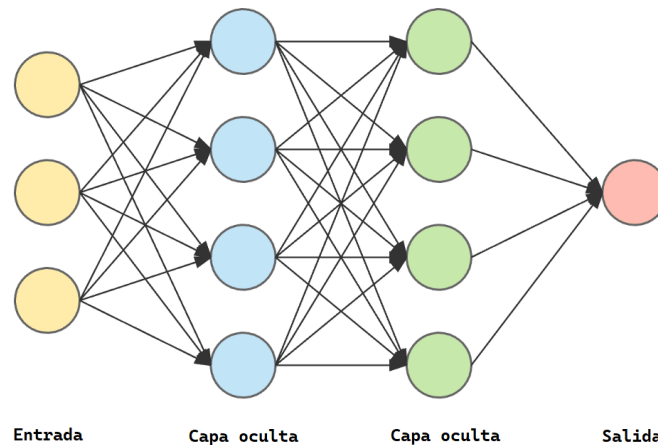


Figura 2. Estructura clásica de una red neuronal artificial. Fuente: Huet, 2023.

El Aprendizaje Profundo o Deep Learning, llamado así por el uso de capas ocultas, representa un avance fundamental en las capacidades analíticas. Los modelos de aprendizaje profundo pueden procesar datos no estructurados, como imágenes, audio y lenguaje natural, sin necesidad de procesamiento manual previo, lo que facilita numerosos casos de uso. Su capacidad para gestionar datos estructurados y no estructurados lo hace especialmente valioso para tareas donde los datos de entrada aparecen de forma natural en diferentes modalidades. Un modelo para la detección de enfermedades, por ejemplo, debería ser capaz de procesar datos de imágenes (como exploraciones radiológicas) junto con datos tabulares, como los resultados de pruebas de pacientes. (Ramakrishnan, 2025)

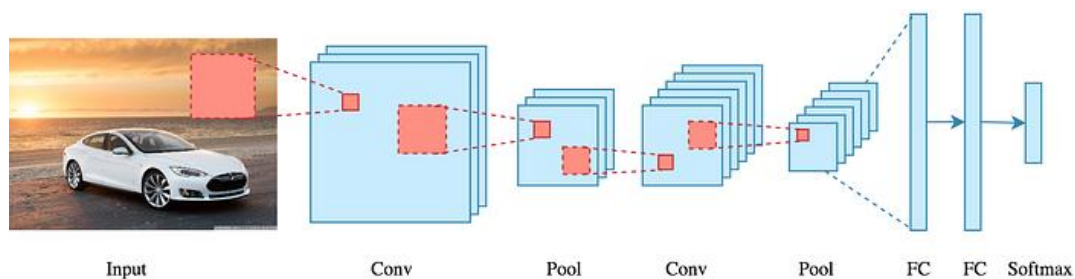


Figura 3. Ilustración de Red Neuronal Convolutiva. Fuente: Silva & Freire, 2019.

Así como en el Machine Learning clásico existen ciertos algoritmos comúnmente conocidos y de frecuente aplicación, en el Deep Learning podemos encontrar tres tipos de estructuras más utilizadas:

- **Perceptrón Multicapa (MLP, por sus siglas en inglés):** este tipo de estructura de red neuronal artificial que consiste en múltiples capas de neuronas interconectadas, donde cada neurona aplica una función de activación no lineal. Se entrena mediante retropropagación del error (backpropagation) y se usa comúnmente para problemas de clasificación y regresión. (Rumelhart, Hinton, & Williams, 1986)

Un ejemplo visual se puede observar con claridad en la figura 2 anterior.

- **Redes Neuronales Convolucionales (CNN, por sus siglas en inglés):** este tipo de arquitectura neuronal fue diseñada para el procesamiento de datos con estructura de cuadrícula, como imágenes. Utilizan capas convolucionales, que aplican filtros para extraer características espaciales, seguidas de capas de pooling para reducir la dimensionalidad. Se han convertido en el estándar en tareas de visión por computadora. (LeCun, Bottou, Bengio, & Haffner, 1998)

Un ejemplo visual se puede observar con claridad en la figura 3 anterior

- **Redes Neuronales Recurrentes (RNN, por sus siglas en inglés):** este tipo de redes neuronales se diseñaron para modelar datos secuenciales. Se caracterizan por tener conexiones recurrentes, lo que permite que la información de estados previos influya en el procesamiento de estados futuros, como se requiere en el procesamiento de lenguaje natural. Sin embargo, sufren problemas de desvanecimiento del gradiente, lo que llevó al desarrollo de variantes como LSTM y GRU. (Hochreiter & Schmidhuber, 1997)

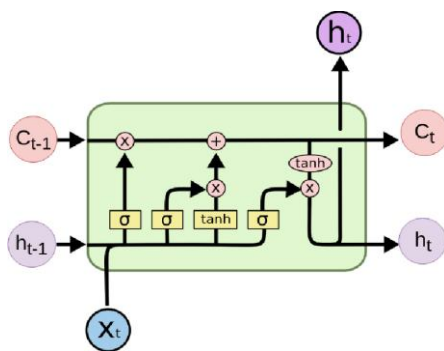


Figura 4. Celda de una Red Neuronal Recurrente. Fuente: Ferro et al., 2020.

Aunque esta última estructura empezó a demostrar grandes capacidades en el desarrollo de aplicaciones para el procesamiento de lenguaje natural, no fue sino a partir del 2017 que una nueva arquitectura de red neuronal empezó a llamar la atención principalmente por este mismo aspecto, el mecanismo de atención que ofrecía. Los Transformers hoy en día son considerados como el tipo de red neuronal que más ha impulsado el crecimiento de este campo de las ciencias computacionales y su uso universal para cualquier usuario, indistintamente de su capacidad técnica. Se ampliará al respecto en el siguiente apartado.

### 1.1.3 Procesamiento de Lenguaje Natural

El Procesamiento de Lenguaje Natural (NLP, por sus siglas en inglés) es un subcampo de la Inteligencia Artificial que se centra en la interacción entre los equipos computacionales y el lenguaje humano. Esta disciplina combina técnicas de lingüística computacional con aprendizaje automático para modelar la estructura y significado del texto que se desea procesar. (Jurafsky & Martin, 2021)

Desde hace muchos años el NLP ha venido transformando diversas actividades cotidianas, permitiendo una interacción más natural con la tecnología y mejorando la eficiencia en diversas áreas, donde sus avances han sido utilizados para el desarrollo de asistentes virtuales, la traducción automática, la corrección y generación de texto, el análisis de sentimientos y entidades, en sistemas de recomendación y más notablemente por la población en general como parte de la automatización del servicio al cliente mediante los muy conocidos Chatbots.

Tal como ya fue mencionado, por mucho tiempo las arquitecturas más utilizadas en este campo fueron las Redes Neuronales Recurrentes y sus variantes. Sin embargo, estas redes tenían problemas para manejar secuencias largas debido a la dificultad de capturar dependencias a largo plazo y otros inconvenientes propios de este tipo de estructuras.

Por lo tanto, con la salida de los Transformers, introducidos en el artículo "Attention Is All You Need" (Vaswani et al., 2017), se presentó una verdadera revolución del NLP al emplear mecanismos de

atención autorregresiva (Self-Attention). Este enfoque permite que el modelo procese simultáneamente todas las palabras de una secuencia, en lugar de hacerlo secuencialmente como en las RNNs, mejorando la capacidad de capturar relaciones contextuales a largo plazo en los textos.

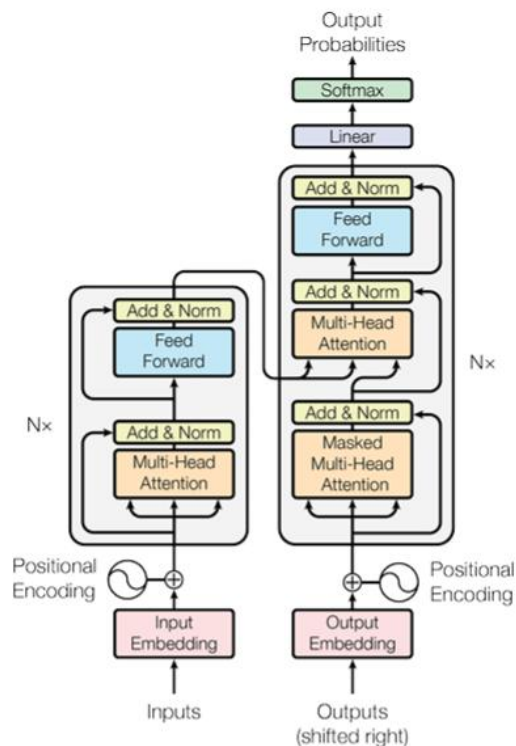


Figura 5. Arquitectura del Transformer. Fuente: Vaswani et al., 2017.

El Transformer es un modelo de red neuronal basado en el mecanismo de autoatención y capas completamente conectadas para procesar datos secuenciales de manera paralela. Se compone de bloques de atención multi-cabeza, capas feedforward y mecanismos de normalización que le permiten capturar relaciones complejas en textos largos con mucha mayor eficiencia. (Vaswani et al., 2017)

Esta arquitectura se compone en términos generales por dos bloques, el Encoder (lado izquierdo de la estructura) y el Decoder (lado derecho de la estructura).

El Encoder procesa la secuencia de entrada en paralelo usando autoatención multi-cabeza, permitiendo que cada palabra se relacione con todas las demás sin depender del orden secuencial. Además, emplea una red feedforward y normalización de capa para mejorar la estabilidad y el aprendizaje profundo. Este diseño supera a modelos anteriores como RNNs y LSTMs en eficiencia y capacidad de capturar dependencias a largo plazo.

El Decoder tiene una estructura bastante similar, pero con dos diferencias claves: incluye una autoatención enmascarada, que evita que los tokens vean el futuro en la generación de texto, y una

atención cruzada con las representaciones del Encoder, permitiendo generar texto basándose en la información codificada. Esta combinación facilita tareas como traducción automática, resumen de textos y generación de contenido, aplicación ampliamente utilizada en la actualidad por muchas aplicaciones tipo.

Al menos en los últimos 3 años los Transformers han sido fundamentales en el desarrollo de los Grandes Modelos de Lenguaje (LLMs, por sus siglas en inglés) como GPT, Gemini, Llama y muchos otros que han llegado para incrementar exponencialmente las capacidades de interacción y utilidad de la IA. Estos modelos han sido entrenados con volúmenes masivos de texto, en el orden de billones de palabras o tokens, utilizando técnicas de aprendizaje autosupervisado y aprendizaje por refuerzo, lo cual les ha permitido comprender y generar texto con un alto grado de coherencia y fluidez.

El impacto de los LLMs supera al de otras tecnologías previas de Inteligencia Artificial porque han hecho posible la creación de modelos con una gran escalabilidad, accesibles a cualquier público y, por lo tanto, personalizables para distintos dominios como la medicina, el derecho y el desarrollo de software. En este último campo es donde en los últimos meses ha resonado con mayor fuerza, dada su capacidad para comprender, generar y completar código, lo cual ha permitido acelerar tareas que antes requerían mucho tiempo y experiencia técnica.

Este éxito que han demostrado los LLMs en el desarrollo de software es justo lo que ha manifestado su capacidad para ser utilizados como principal componente de los agentes de IA, funcionando como el cerebro que combina comprensión de lenguaje natural con capacidades de razonamiento y generación de código para interactuar como asistentes virtuales avanzados.

#### 1.1.4 Agentes Inteligentes

Se podría definir como Agente de IA aquel sistema capaz de percibir su entorno, razonar sobre él y actuar de manera autónoma para lograr los objetivos por los cuales fue diseñado, en muchos casos con capacidad de adaptarse y aprender de manera continua. En el contexto actual, estos agentes se apoyan en modelos avanzados como los LLMs para interpretar el lenguaje humano, planificar tareas y ejecutar acciones en entornos digitales o físicos. Los desarrolladores de estos asistentes integran componentes de razonamiento simbólico, herramientas externas como capacidades de navegación y llamadas a API's y memoria a largo plazo. (Liu, Lou, Jiao, & Zhang, 2024)

El aumento en las capacidades que se logró con la llegada de los LLMs fue el principal impulsor de esta nueva tendencia, que aumenta las expectativas de contar con un asistente con tal capacidad, que le permita delegar al ser humano ciertas tareas que por sus características requieren recursos importantes, pero que su aportación de valor no es tan significativa. Pero no solo ha sido esto, sino

que también se ha demostrado que estos sistemas son capaces de desarrollar tareas fundamentales como la atención al cliente y el desarrollo de software con niveles importantes de rendimiento.

Hugging Face, como parte de su curso sobre agentes de IA, presenta una estructura resumida en dos partes:

1. El Cerebro (modelo de IA)
2. El Cuerpo (capacidades y herramientas)

En el primero es donde se desarrolla todo el pensamiento. El modelo de IA se encarga de gestionar el razonamiento y la planificación para cumplir con las actividades que se le han solicitado, decidiendo que acciones realizar según la situación. En el segundo se representa todo lo que el agente está equipado para hacer, o sea, todas las herramientas con las que se ha dotado al modelo IA para accionar la planificación previamente realizada con base en la solicitud del usuario.



Figura 6. Estructura de ejemplo de un Agente de IA. Fuente: González, 2024.

Esta simple estructura permite comprender a grandes rasgos el funcionamiento de un agente de IA, el cual podrá cumplir cualquier tarea para la cual se le otorgó capacidades en su desarrollo. Por lo tanto, el diseño de las herramientas es crucial e incide completamente en la calidad del agente para cumplir sus objetivos. Según sea su caso de uso serán requeridas herramientas muy específicas que requerirán el despliegue de servicios codificados o herramientas de propósito general como la búsqueda web. (Hugging Face, s.f)

Son muchas las opciones en las que se puede pensar a la hora de dar ejemplos de este tipo de agentes virtuales, pero algunos de los casos que pueden ser más fácilmente aplicables son los siguientes:

1. **Asistente de Call Center:** en este caso el agente de IA puede ser diseñado para interactuar directamente con personas a través del lenguaje natural, simulando una interacción humana mediante herramientas de Speech-to-Text (conversión de voz en texto) y Text-to-Speech (conversión de texto en voz). Estos asistentes pueden contestar preguntas frecuentes, guiar a los usuarios y realizar escalamientos cuando así lo consideren, ofreciendo un servicio más cercano, lo cual ha sido una queja recurrente respecto a los antiguos Chatbots basados en reglas.
2. **Agente de Recomendación:** Este tipo de agente, mediante el análisis de datos relacionados con el comportamiento y las preferencias del usuario, no solo puede predecir sus necesidades o deseos, sino que podrá ofrecer y hasta solicitar contenido personalizado y en tiempo real, con una interacción más oportuna con relación de los mecanismos de recomendación actuales.
3. **Asistente de Negociación Automática:** este caso de uso ofrece un agente autónomo que no solo analiza las variables actuales del mercado para efectuar una transacción informada, sino que también controla la compra y venta de activos en procesos de negociación como subastas electrónicas o trading financiero. Se emplean algoritmos de aprendizaje por refuerzo y teoría de juegos para tomar decisiones en tiempo real, basados en los objetivos que le han sido transmitidos, el comportamiento de otros agentes y condiciones del entorno.
4. **Agente de Análisis Crediticio:** diseñado para evaluar automáticamente el riesgo asociado al otorgamiento de créditos, este agente emite análisis con base en la información histórica disponible de los clientes como sus transacciones, ingresos y comportamiento de pago, Adicionalmente, puede ser dotado de capacidades de comprensión sobre la realidad actual de la organización y su capacidad para asumir el riesgo que pueda representar el otorgamiento de cada solicitud individual de forma agrupada por distintas segmentaciones demográficas.

## 2. Objetivos del Proyecto

Este trabajo tiene como objetivo general el desarrollo de un agente encargado de la generación de un reporte automático que incluya información básica sobre el indicador o indicadores económicos seleccionados, así como proyecciones sobre sus valores esperados a partir de su comportamiento histórico y de otras variables seleccionadas.

Además, a partir de servicios de API disponibles, se realiza una búsqueda automática de noticias relacionadas recientes con el o los indicadores seleccionados, creando un pequeño corpus de información que será enviado a un modelo de generación de texto que se encargará de realizar un resumen técnico y analítico de las mismas y de las proyecciones obtenidas, con el objetivo de ofrecer conclusiones sobre el comportamiento esperado de los indicadores. Los objetivos específicos del proyecto son los siguientes:

- Obtener la información histórica de los indicadores económicos publicados por el Banco Central de Costa Rica en su servicio web, aplicando las modificaciones necesarias a los datos para su uso posterior.
- Generar proyecciones sobre el indicador económico seleccionado, a partir de su comportamiento histórico y su relación con otras variables disponibles.
- Obtener de manera automatizada noticias relevantes y actuales relacionadas con el indicador de interés que ofrezca información importante para análisis posteriores.
- Generar informes técnicos y estructurados apalancados por grandes modelos de lenguaje (LLMs) que ofrezcan análisis resumidos respecto a la información acumulada hasta el momento (noticias, comportamiento histórico y proyecciones).



### 3. Materiales y Métodos

Para cumplir con los objetivos propuestos, se han utilizado diversas tecnologías y metodologías que combinan el desarrollo de código tradicional, el uso de modelos grandes de lenguaje y herramientas que han sido puestas a disposición del público mediante APIs.

El código se ha desarrollado utilizando Visual Studio Code (VSC) como Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés). Primeramente, se creó un entorno virtual utilizando la versión de Python 3.10.11 para asegurar una adecuada gestión de las librerías utilizadas y evitar problemas por dependencias o compatibilidad. La creación del entorno se ha configurado empleando la librería venv y su gestión se ha realizado mediante la extensión de VSC llamada Python Environment Manager.

Habiéndose activado el entorno virtual creado, se procedió con la instalación de las librerías mediante el comando pip, comúnmente utilizado para el proceso de importación de librerías en Python.

A continuación, se presenta un detalle de cada una de las librerías utilizadas en este proyecto, brindando una pequeña descripción de su funcionalidad y para que fue utilizada. Se hará una excepción para aquellas librerías que son consideradas de uso general y que usualmente ya se incluyen en la instalación estándar de Python, como es el caso de os, requests, PIL, warnings y datetime.

#### 3.1 Librerías utilizadas

##### **Pandas**

Librería esencial para manipular y analizar estructuras de datos como DataFrames y Series de forma eficiente. Utilizada en el proyecto para la manipulación y formateo de los datos recibidos de las variables económicas consultadas.

##### **Numpy**

Ofrece soporte para arrays multidimensionales y funciones matemáticas de alto rendimiento. Se utilizó para algunos procesos que implicaban cambios solo sobre ciertas series de datos.

##### **Streamlit**

Framework para construir aplicaciones web interactivas de manera rápida. Librería principal para el desarrollo de la interfaz web para que interactuar de forma sencilla con el agente creado

##### **Matplotlib**

Biblioteca de gráficos 2D que permite crear visualizaciones personalizadas como líneas, barras, histogramas, etc. Se ha utilizado para el desarrollo de las visualizaciones del comportamiento de la variable objetivo a lo largo de periodo establecido.

### **Smolagents**

Librería ligera de Hugging Face para construir agentes inteligentes que razonan paso a paso y pueden interactuar con herramientas. Para el proyecto se utilizó como herramienta principal para desarrollar las capacidades del agente y su interacción.

### **Prophet**

Herramienta de Facebook para pronósticos de series temporales. Diseñada para manejar estacionalidades y tendencias con facilidad. Librería utilizada para el desarrollo de las proyecciones bajo los distintos escenarios planteados.

### **Groq**

Cliente para conectarse con los modelos LLM (Large Language Models) alojados en la plataforma de Groq, optimizados para velocidad. Se utilizó para contar con las capacidades de un LLM con capacidad de razonamiento en los análisis planteados.

### **Markdown2**

Convierte texto en formato Markdown a HTML de manera sencilla. Librería utilizada para la conversión del producto generado por el agente a un formato fácilmente compatible.

### **io**

Módulo estándar de Python para manejar flujos de entrada/salida (como archivos en memoria). Para el proyecto se utilizó de manera conjunta con Base64 para la conversión de los gráficos a un formato de fácil inclusión en un formato markdown.

### **Base64**

Permite codificar y decodificar datos en base64, útil para enviar datos binarios como imágenes a través de texto plano. Para el proyecto se utilizó de manera conjunta con io para la conversión de los gráficos a un formato de fácil inclusión en un formato markdown.

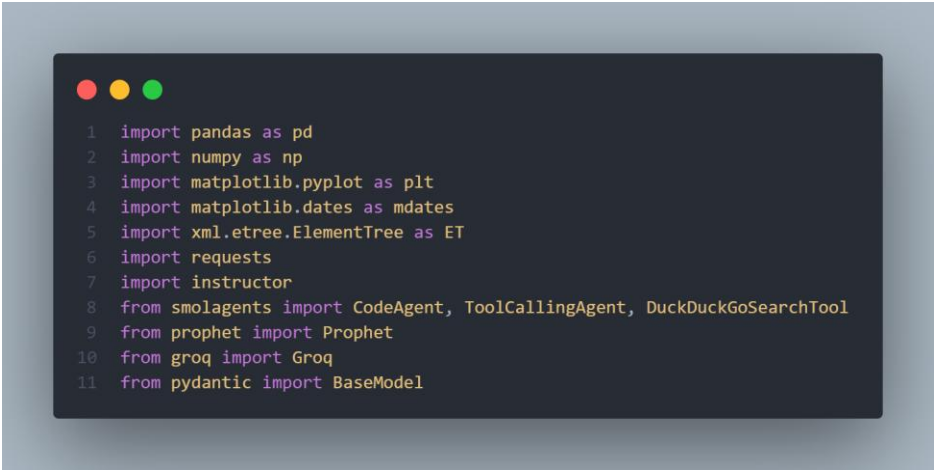
### **Instructor**

Permite estructurar y validar salidas de modelos LLM usando clases de Pydantic, asegurando que las respuestas cumplan un formato definido. Se utilizó para permitir al LLM utilizado mediante Groq generar salidas estructuradas, interpretando la salida como un archivo tipo JSON.

## Pydantic

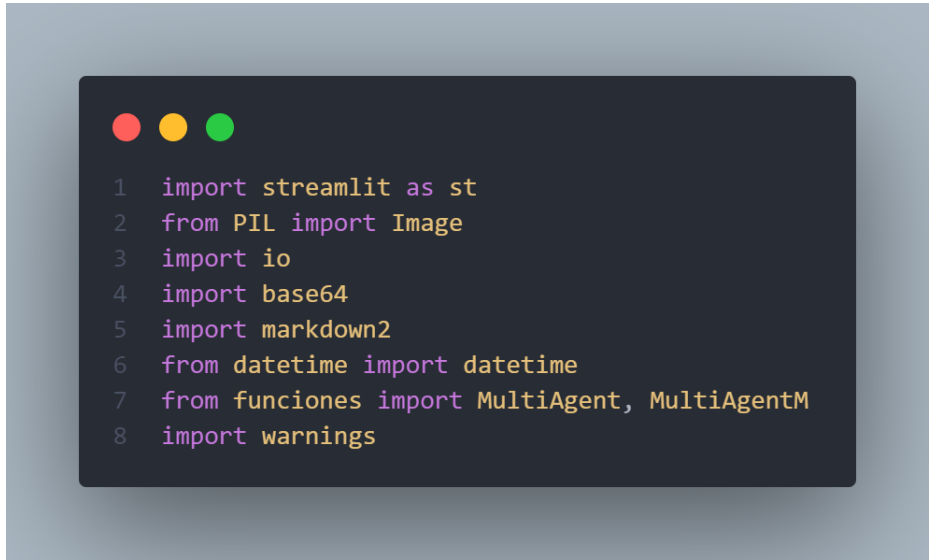
Permite definir clases de datos con validación automática de tipos. Muy usado para modelos de entrada/salida en APIs y en combinación con LLMs. Esta librería se utilizó para estructurar las salidas esperadas del modelo, lo cual permite que los resultados del análisis sean específicas y manipulables con mayor facilidad.

Se pueden agrupar estas librerías en dos grupos de acuerdo con el uso dado, el primero, donde se ha estructurado el agente mediante clases, el segundo, donde se ha realizado el desarrollo de la herramienta interactiva:

A screenshot of a code editor with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The code is written in a light-colored font and lists imports for various Python libraries used for structuring an agent.

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import matplotlib.dates as mdates
5 import xml.etree.ElementTree as ET
6 import requests
7 import instructor
8 from smolagents import CodeAgent, ToolCallingAgent, DuckDuckGoSearchTool
9 from prophet import Prophet
10 from groq import Groq
11 from pydantic import BaseModel
```

Figura 7. Librerías para la estructuración del agente. Fuente: Elaboración Propia.

A screenshot of a code editor with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The code is written in a light-colored font and lists imports for various Python libraries used for developing an interactive tool.

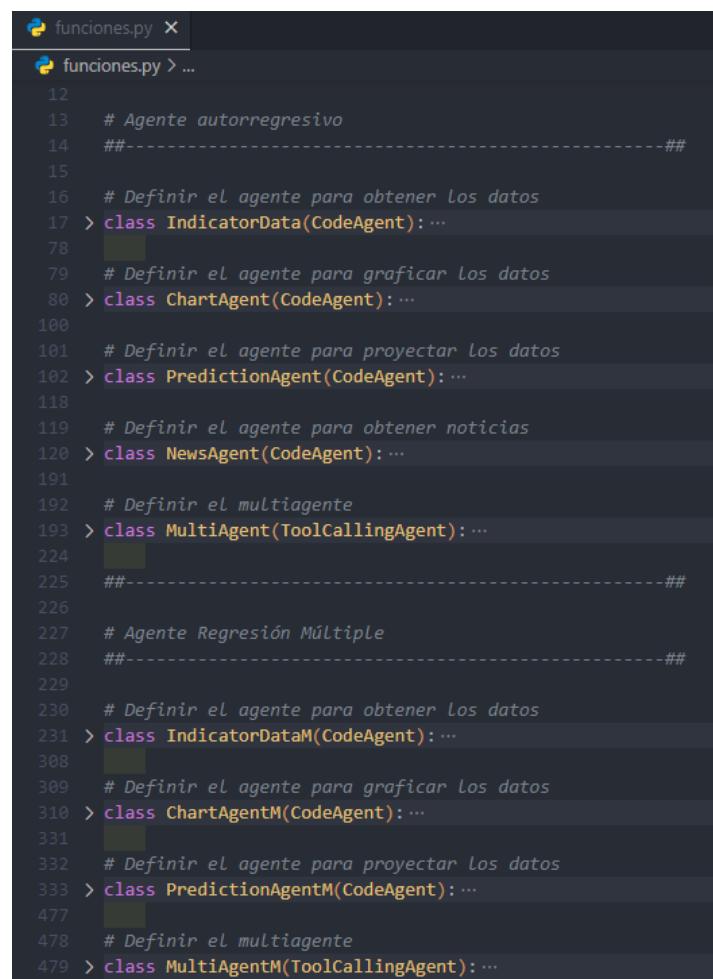
```
1 import streamlit as st
2 from PIL import Image
3 import io
4 import base64
5 import markdown2
6 from datetime import datetime
7 from funciones import MultiAgent, MultiAgentM
8 import warnings
```

Figura 8. Librerías para el desarrollo de la herramienta interactiva. Fuente: Elaboración Propia.

## 3.2 Metodología y Servicios Usados

Como parte de la metodología de trabajo, se desarrollaron dos scripts de Python como la base fundamental del proyecto: funciones.py y App.py, donde el primero corresponde al desarrollo de cada una de las partes que comprenden el agente y el segundo es propiamente donde se desarrolló la codificación de la aplicación web con Streamlit.

Para el desarrollo del agente, se dividieron sus funciones en sus dos trabajos principales. Primeramente, el análisis autorregresivo, en el cual para realizar las proyecciones se basa únicamente en el comportamiento histórico de la variable objetivo. Como segunda parte se estableció el análisis de regresión múltiple donde, además de considerar el comportamiento propio de la variable objetivo, se considera el comportamiento histórico de las variables adicionales que hayan sido seleccionadas como regresores. Lo anterior fue plasmado de esa manera a nivel de código con cada una de las clases creadas:



```
funciones.py x
funciones.py > ...
12
13 # Agente autorregresivo
14 ##-----##
15
16 # Definir el agente para obtener Los datos
17 > class IndicatorData(CodeAgent): ...
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79 # Definir el agente para graficar Los datos
80 > class ChartAgent(CodeAgent): ...
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101 # Definir el agente para proyectar Los datos
102 > class PredictionAgent(CodeAgent): ...
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119 # Definir el agente para obtener noticias
120 > class NewsAgent(CodeAgent): ...
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192 # Definir el multiagente
193 > class MultiAgent(ToolCallingAgent): ...
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225 ##-----##
226
227 # Agente Regresión Múltiple
228 ##-----##
229
230 # Definir el agente para obtener Los datos
231 > class IndicatorDataM(CodeAgent): ...
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309 # Definir el agente para graficar Los datos
310 > class ChartAgentM(CodeAgent): ...
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332 # Definir el agente para proyectar Los datos
333 > class PredictionAgentM(CodeAgent): ...
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428 # Definir el multiagente
429 > class MultiAgentM(ToolCallingAgent): ...
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
```

Figura 9. Estructura script funciones.py. Fuente: Elaboración Propia.

Cada función que tendría el agente fue creada con base en las clases de la librería Smolagents utilizadas: CodeAgent y ToolCallingAgent. Como se puede observar en la figura anterior, utilizando CodeAgent se crearon las cuatro funciones o capacidades que tiene el agente:

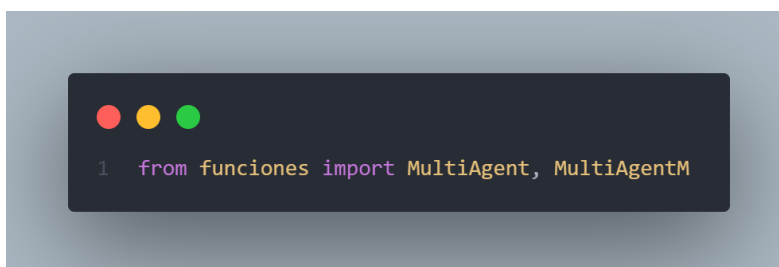
- Obtener la información desde la fuente: IndicatorData e IndicatorDataM.
- Crear el gráfico donde se refleje el comportamiento real y proyectado de la variable objetivo: ChartAgent y ChartAgentM.
- Generar las proyecciones a partir de los datos obtenidos: PredictionAgent y PredictionAgentM.
- Obtener las noticias relacionadas al indicador objetivo, pasarlas al LLM para su análisis y generación de conclusiones: NewsAgent.

En este último caso, solo fue necesario la creación de una única función, ya que, a diferencia de las otras tres capacidades, los datos obtenidos mantienen el mismo formato independientemente del tipo de análisis realizado (autorregresivo o regresión múltiple).

De forma posterior, habiéndose creado las capacidades anteriormente mencionadas, se procede con su agrupación utilizando ToolCallingAgent, creándose las dos agrupaciones ya dichas:

- MultiAgent para el desarrollo de las tareas autorregresivas, y
- MultiAgentM para el desarrollo de las tareas de regresión múltiple.

Por otro lado, en el caso del aplicativo web, dentro del script App.py además de las librerías antes mencionadas, se importan las clases creadas en el script funciones.py, para disponibilizar las funciones como parte de la interfaz diseñada. Esto se realiza a partir de la siguiente línea de código:

A screenshot of a code editor with a dark background and light blue text. At the top left, there are three colored circles (red, yellow, green) representing window controls. The code shown is a single line: `1 from funciones import MultiAgent, MultiAgentM`.

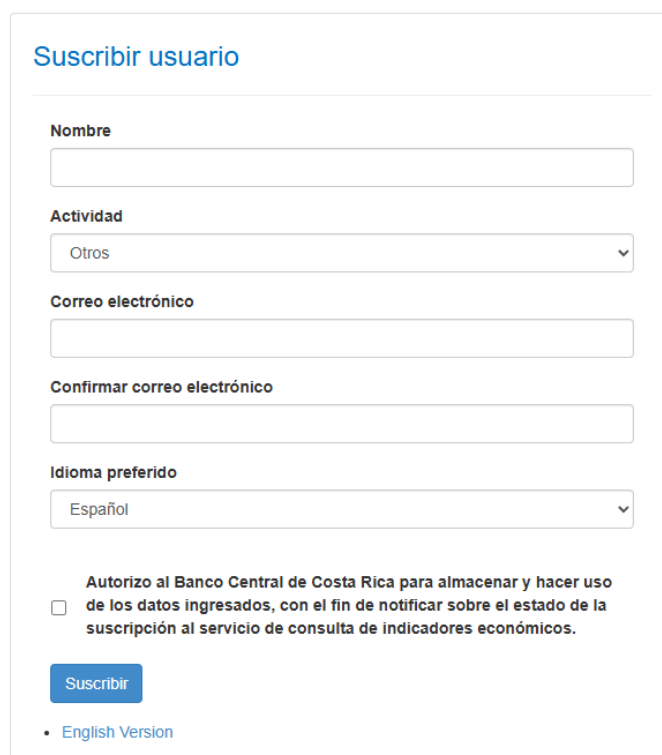
```
1 from funciones import MultiAgent, MultiAgentM
```

Figura 10. Importación de las funciones Multiagente. Fuente: Elaboración Propia.

Con las importaciones necesarias ya realizadas, se procede con la programación de la interfaz web, como los estilos deseados y las funcionalidades requeridas para la inclusión de la información por parte del usuario, siendo estos aspectos fundamentales como claves y usuarios requeridos para la obtención de la información.

En este punto es donde se vuelve importante aclarar sobre dos servicios web utilizados y que son cruciales para el funcionamiento del agente, el que permite la obtención de los datos de la variable económica de interés y de las variables regresoras en el caso del análisis de regresión múltiple, y el que permite acceder a las funcionalidades del LLM para la realización de los análisis previamente explicados.

Como primer servicio se ha utilizado la API que el Banco Central de Costa Rica pone a disposición del público general para consultar y autoabastecerse de los datos históricos de las variables asociadas a la economía costarricense. La solicitud de acceso es fácil realización, solo basta el llenado de un formulario en su página web y en segundos se recibirá en el correo electrónico el token necesario para realizar las consultas deseadas. Además, han puesto a disposición una guía de uso y un catálogo con la numeración asignada a cada indicador.



El formulario, titulado "Suscribir usuario", contiene los siguientes campos:

- Nombre:** Un campo de texto para ingresar el nombre del usuario.
- Actividad:** Un menú desplegable con la opción "Otros" seleccionada.
- Correo electrónico:** Un campo de texto para ingresar el correo electrónico.
- Confirmar correo electrónico:** Un campo de texto para confirmar el correo electrónico.
- Idioma preferido:** Un menú desplegable con la opción "Español" seleccionada.
- Consentimiento:** Una casilla de verificación (checkbox) que, al ser marcada, autoriza al Banco Central de Costa Rica para almacenar y hacer uso de los datos ingresados, con el fin de notificar sobre el estado de la suscripción al servicio de consulta de indicadores económicos.
- Botón:** Un botón azul con el texto "Suscribir".
- Enlace:** Un enlace azul que dice "English Version".

Figura 11. Formulario Suscripción de Usuario. Fuente: Página web oficial BCCR.

El nombre, el correo y el token serán necesarios para realizar cualquier nueva consulta, por eso es de suma importancia su adecuado almacenamiento y respaldo. Además, es importante mantener como personal dichos datos información para evitar que sean utilizados con fines malintencionados sobre este punto de acceso.

Como segundo servicio, ya se había explicado la necesidad de acceder a un LLM que se encargue de realizar la revisión de las noticias y de las proyecciones, realizando así un análisis al respecto que sirva como fundamentación sobre la estimación final realizada. Aunque se contaba con la capacidad instalada de manera local para ejecutar un LLM de pocos parámetros, dada la facilidad actual de acceder a estos servicios mediante API y con la posibilidad de utilizar modelos mucho más avanzados puestos a disposición en la nube, se decidió utilizar el servicio de Groq.

Groq es una empresa que principalmente se dedica a la fabricación de chips de Inteligencia Artificial. En específico, Groq desarrolla los llamados Language Processing Units (LPS), siendo estos chips especializados para el proceso de inferencia de los Grandes Modelos de Lenguaje. Pero la empresa no se queda solo en la fabricación, sino que también mediante GroqCloud ofrece acceso a diferentes LLMs como la serie Llama, DeepSeek y otros de código abierto.

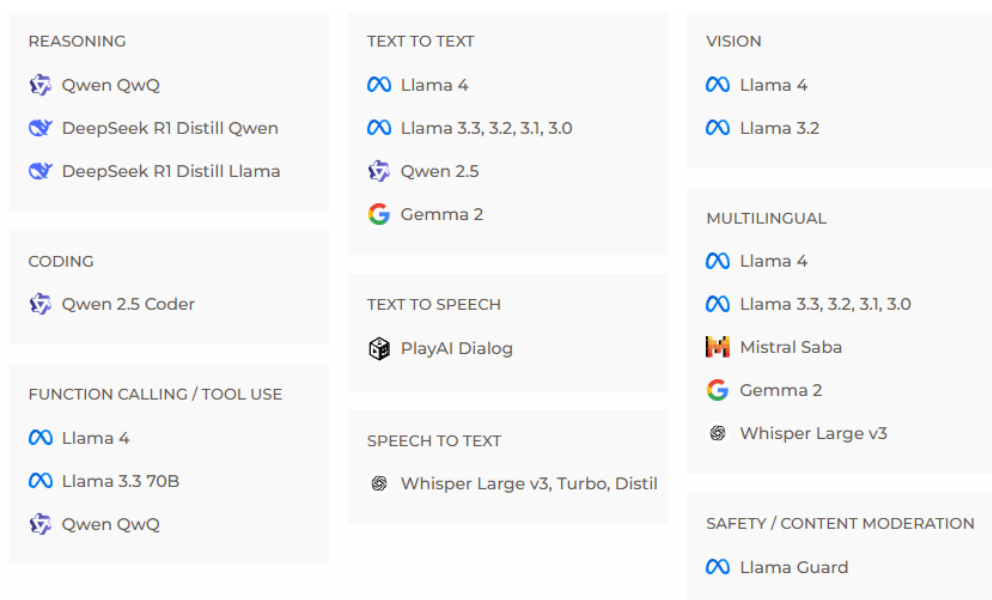


Figura 12. Modelos disponibles en GroqCloud. Fuente: Página web oficial Groq.

Groq no solo ofrece un plan a un costo muy competitivo, sino que también cuenta con una opción gratuita que establece límites considerados altos en comparación con otras ofertas del mercado, siendo esto fundamental para seleccionar este servicio para el desarrollo del proyecto. Además, en GroqCloud se puede encontrar uno de los modelos razonadores que más revuelo crearon a inicios de este 2025, como lo fue DeepSeek-R1. Este aspecto es fundamental ya que como parte del proceso de análisis del agente se desea mostrar la cadena de pensamiento que un modelo de tipo razonador utiliza para finalmente emitir sus conclusiones.

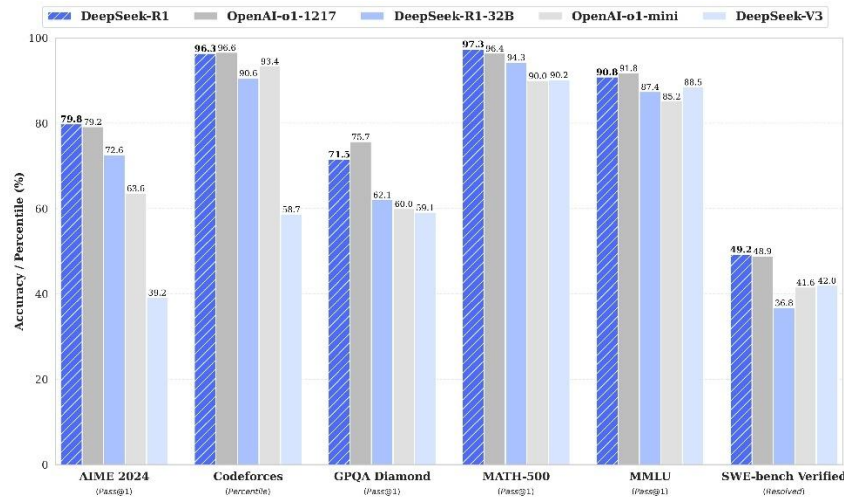


Figura 13. Comparativa a nivel de Benchmarks. Fuente: DeepSeek-AI, 2025.

DeepSeek-R1 es un modelo de pesos abiertos desarrollado por la empresa China DeepSeek que demostró capacidades de state-of-the-art en el campo de los modelos de lenguaje razonadores, con mejoras significativas en su proceso de entrenamiento y en su estructura que representaron eficiencias a nivel de recursos requeridos importantes en comparación con otros modelos cerrados como los de OpenAI.

Al ser un nivel gratuito el utilizado en GroqCloud, existen ciertas limitantes que debían ser consideradas en relación con la cantidad de consultas y tokens por minuto, los cuales se detallan en la siguiente imagen. No obstante, dadas las características del producto generado a partir del proyecto, la cantidad de generaciones quedarían bastante por debajo de los límites establecidos, siendo aún más correcta la selección de este proveedor.

MODEL ID	RPM	RPD	TPM
deepseek-r1-distill-llama-70b	30	1,000	6,000

- **RPM:** Solicitudes por minuto
- **RPD:** Solicitudes por día
- **TPM:** Tokens por minuto

Figura 14. Límites de uso para DeepSeek-R1. Fuente: Página web oficial Groq.

Para su puesta a disposición fuera de solo el entorno local, se desarrolló una estructuración para crear un contenedor en Docker con lo necesario para que la aplicación web sea funcional en cualquier equipo y más que eso, ya que, utilizando la nube de Microsoft Azure se ha desplegado una copia del contenedor y se ha creado un punto de acceso mediante el servicio de App Service para que cualquiera pueda interactuar con el agente.



Para lograr lo anterior, además de los archivos previamente explicados, se crearon tres archivos adicionales necesarios para el levantamiento del contenedor:

- Requirements.txt: archivo de texto donde se incluyen las librerías requeridas para la correcta ejecución de las funcionalidades codificadas.
- Dockerfile: archivo de texto donde se incluyen las instrucciones paso a paso para la construcción de la imagen personalizada para la adecuada configuración y posterior funcionamiento del contenedor.
- Docker\_compose.yml: archivo de configuración para automatizar la ejecución del contenedor basado en la imagen configurada en el archivo Dockerfile.
- .dockerignore: archivo de texto donde se indican los archivos del directorio actual que se desean excluir para la creación del contenedor.



Figura 15. Archivos para la Dockerización. Fuente: Elaboración Propia.

Con estos últimos archivos ya creados, la estructura del proyecto se encuentra completa y lista para su despliegue, primeramente, creando el contenedor en el entorno local y, seguidamente, incluyéndolo en Azure. Esta sería la composición final del proyecto a nivel de archivos:

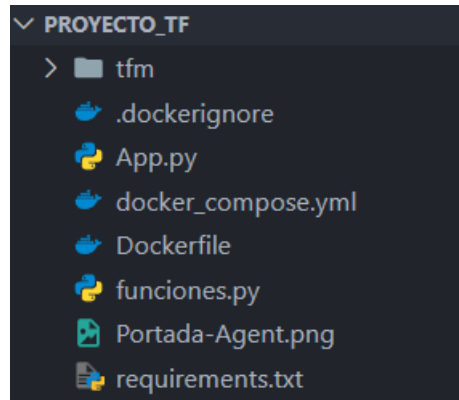


Figura 16. Estructura del Proyecto a nivel de archivos. Fuente: Elaboración Propia.

Los aspectos más específicos del código de App.py y funciones.py serán explicados en el apartado de Resultados a como se vaya explicando la funcionalidad del agente. En lo que respecta al proceso de dockerización, los pasos realizados son de suma sencillez. Utilizando la opción “Compose up” del archivo docker\_compose.yml se inicia el proceso de creación del contenedor con la configuración establecida. Ya finalizado el proceso, dentro de la aplicación Docker Desktop se observa el contenedor creado y en ejecución:

<input type="checkbox"/>	Name	Container ID <small>↑</small>	Image	Port(s)	CPU (%)	Last started	Actions
<input type="checkbox"/>	▼ <span style="color: green;">●</span> proyecto_tf	-	-	-	0%	13 minutes ago	<span style="color: blue;">■</span> <span style="color: blue;">⋮</span> <span style="color: red;">🗑</span>
<input type="checkbox"/>	<span style="color: green;">●</span> tfm-1	effce44d9395	proyecto_tf-tfm	8082:8501 <a href="#">↗</a>	0%	13 minutes ago	<span style="color: blue;">■</span> <span style="color: blue;">⋮</span> <span style="color: red;">🗑</span>

Figura 17. Contenedor creado y activo. Fuente: Elaboración Propia.

Con este proceso completado se asegura que el agente funcionará correctamente en cualquier otro equipo tal como lo hace en el entorno local. De manera posterior, ya se puede proceder con su carga o exportación a la nube, donde como se indicó se pone a disposición de cualquier usuario que cuente con el enlace del servicio. Respecto a la configuración realizada en Azure, se explicará en términos generales los pasos realizados para disponibilizar el agente:

1. Se crea un nuevo grupo de recursos donde se agrupan los servicios requeridos.
2. Se crea un nuevo plan de App Service asociado al grupo de recursos creado.

Detalles	
Suscripción	Azure for Students
Grupo de recursos	tfm_agente
Nombre	tfm-agente
Sistema operativo	Linux
Región	East US 2
SKU	Básico
Tamaño	Pequeño
ACU	Total de ACU: 100
Memoria	1.75 GB de memoria

Figura 18. Detalle configuración Plan App Service. Fuente: Elaboración Propia.

3. Utilizando la extensión de Docker en VSCode, se consultan las imágenes disponibles y sobre la imagen creada proyecto\_tf-tfm, seleccionando la opción push, se procede con la exportación a la nube mediante la creación de un registro de Azure Container.
4. Como último punto, para finalizar el proceso de despliegue del agente, se procede con la creación del App Service. De igual manera, aprovechando las funcionalidades de VSCode asociado a Azure, al consultar las opciones disponibles para el registro creado se utiliza la opción “Deploy Image to Azure Service App” a partir de la cual al completar una serie de sencillos pasos realiza la creación de la app web.

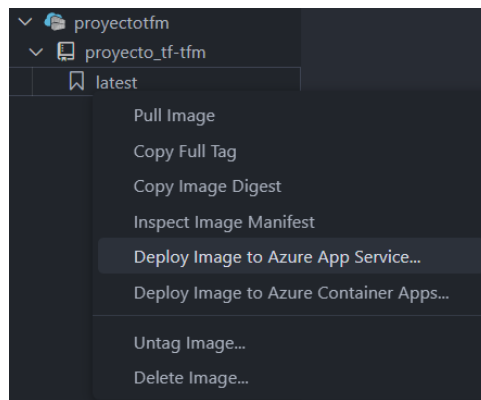


Figura 19. Despliegue de imagen en App Service. Fuente: Elaboración Propia.

Al finalizar el proceso de despliegue, ya se cuenta con el servicio creado y en funcionamiento. El costo asociado a la app web no supera los US\$ 13 mensuales y puede llegar a ser inferior en caso de no tener un uso excesivo como es de esperar dadas las características del agente creado. En la siguiente imagen se resumen los componentes desplegados en la nube de Azure.





<input type="checkbox"/> Nombre ↑↓	Tipo ↑↓
<input type="checkbox"/>  proyecto-tfmds	App Service
<input type="checkbox"/>  proyectotfm	Container registry
<input type="checkbox"/>  proyectotfmds3005e1 (proyectotfm/proyectotfmds3005e1)	Container registry webhook
<input type="checkbox"/>  tfm-agente	Plan de App Service

Figura 20. Grupo de Recursos creado. Fuente: Elaboración Propia.

## 4. Resultados

Explicado el proceso llevado a cabo para la creación del agente, incluyendo las tecnologías y servicios utilizados, se profundizará en el agente desarrollado y las capacidades que le han sido otorgadas. Lo anterior se realizará a partir de la app web ya desplegado, mostrando su funcionamiento y el código que fue desarrollado para alcanzar el logro de los objetivos establecidos.



Figura 21. Interfaz del Agente, vista preliminar. Fuente: Elaboración Propia.

Como se observa en la imagen anterior, la vista inicial de la interfaz del agente muestra la solicitud al usuario de la variable económica que se desea analizar. En este caso el agente está configurado para realizar su trabajo sobre las siguientes variables:

- Tipo de cambio de compra.
- Tipo de cambio de venta.
- Inflación.
- PIB (Producto Interno Bruto del país).
- Tasa de Empleo.
- IMAE (Índice Mensual de Actividad Económica).
- TBP (Tasa Básica Pasiva).
- TRI 6M (Tasa de Referencia Interbancaria a 6 meses).

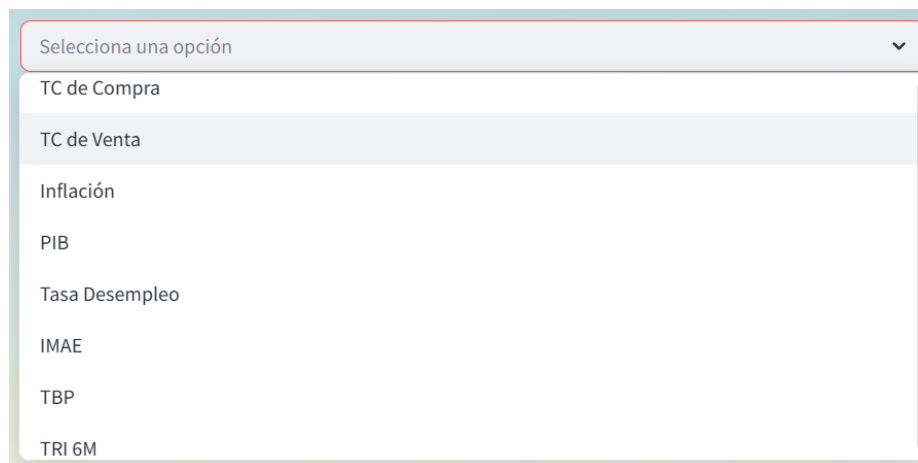


Figura 22. Despliegue de variables disponibles. Fuente: Elaboración Propia.

Además, en el sidebar ubicado en la columna lateral izquierda, se ha incluido el espacio para la inclusión de la API Key de GroqCloud, necesaria para la utilización de las capacidades del LLM, de acuerdo con lo indicado en la sección anterior de Materiales y Metodología. Para facilidad del usuario, también se comparte un tutorial que explica cómo obtener acceso a GroqCloud de forma sencilla.

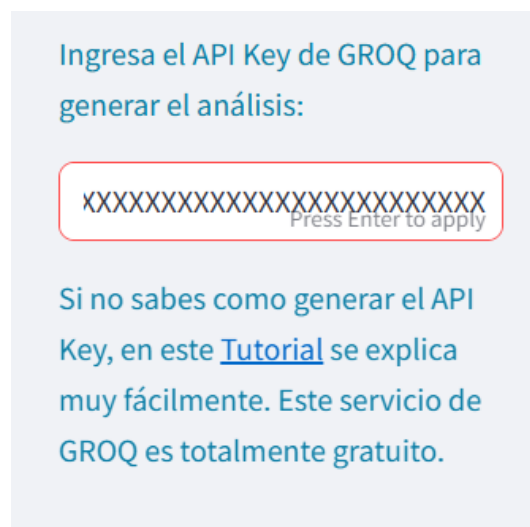


Figura 23. Inclusión de API Key de Groq. Fuente: Elaboración Propia.

Con la selección de la variable de interés ya realizada, se habilita la casilla para seleccionar el tipo de análisis de acuerdo con lo indicado anteriormente: Autorregresivo o Regresión Múltiple. Con base en esta selección se habilitan diferentes entradas requeridas para completar las tareas:

Soy un Agente creado para ayudarte a analizar variables económicas.

¿Cuál variable te interesa analizar?

TC de Compra

Periodicidad: Diaria

Selecciona el tipo de análisis:

Autorregresivo

Completa lo siguiente para obtener los datos del [BCCR Servicio Web](#)

Nombre:

Correo Electrónico:

Token BCCR:

Fecha Inicial: 2020/01/01 Fecha Final: 2025/04/20

Número de periodos a predecir: 180

Generar Reporte

Figura 24. Información requerida para análisis autorregresivo. Fuente: Elaboración Propia.

Dado que para el análisis autorregresivo solamente se toma la información histórica del indicador seleccionado, únicamente se solicita la información requerida por el Servicio Web del BCCR para la obtención de los datos, el rango deseado para la generación de la base histórica y los periodos que se desean proyectar. Acá es importante aclarar que de forma predeterminada se establecieron valores para proyectar de acuerdo con la periodicidad del indicador seleccionado, con la posibilidad de ser cambiado por el usuario:

```

1  if variable != None:
2      freq = dicc_indicador[variable][1]
3      if freq == "D":
4          periodicidad = "Diaria"
5          value = 180
6      elif freq == "M":
7          periodicidad = "Mensual"
8          value = 6
9      elif freq == "Q":
10         periodicidad = "Trimestral"
11         value = 2
12     else:
13         periodicidad = "Anual"
14         value = 1

```

Figura 25. Definición base de periodos a proyectar. Fuente: Elaboración Propia.

Soy un Agente creado para ayudarte a analizar variables económicas.

¿Cuál variable te interesa analizar?

TC de Compra

Periodicidad: Diaria

Selecciona el tipo de análisis: ↔

Regresión Múltiple

Completa lo siguiente para obtener los datos del [BCCR Servicio Web](#)

Nombre:

Correo Electrónico:

Token BCCR:

Fecha Inicial: 2020/01/01 Fecha Final: 2025/04/20

Selecciona las variables predictoras:

Choose an option

Número de periodos a predecir:

180

Figura 26. Información requerida para análisis regresión múltiple. Fuente: Elaboración Propia.

En el caso del análisis de regresión múltiple, además de la información del Servicio Web del BCCR previamente explicado, el rango de fechas y los periodos a proyectar, también deben ser seleccionadas las variables predictoras, siendo estas las mismas explicadas con anterioridad, exceptuando por supuesto la seleccionada para realizar el análisis.

Selecciona las variables predictoras:

Choose an option

TC de Venta

Inflación

PIB

Tasa Desempleo

IMAE

TBP

TRI 6M

Figura 27. Indicadores disponibles para funcionar como predictores. Fuente: Elaboración Propia.

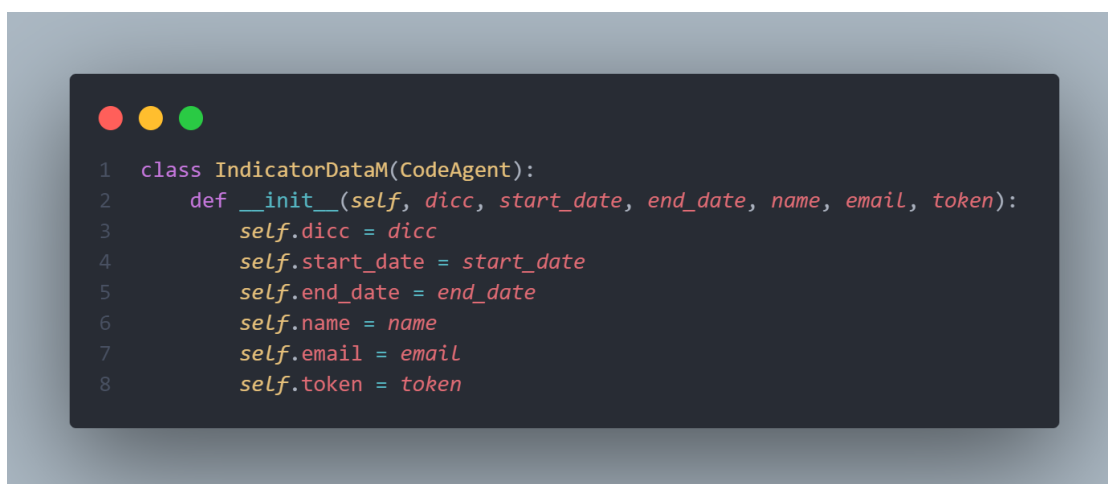
Cuando cada uno de los campos requeridos ya han sido definidos, es posible la generación del reporte mediante el botón disponible al final de la pantalla, lo cual permite que cada una de las funciones del agente tome los datos suministrados y se empiecen a desarrollar las tareas para las cuales fue creado, iniciando por la obtención de datos.

IndicatorData e IndicatorDataM se encargan de la obtención y preparación de los datos históricos del indicador seleccionado y en el caso del análisis de regresión múltiple, de los regresores escogidos como tales. Para ello, primero se definen los datos ingresados como atributos que serán utilizadas mediante el método `__init__` para crear la nueva instancia:



```
1 class IndicatorData(CodeAgent):
2     def __init__(self, variable, indicator, start_date, end_date, name, email, token):
3         self.variable = variable
4         self.indicator = indicator
5         self.start_date = start_date
6         self.end_date = end_date
7         self.name = name
8         self.email = email
9         self.token = token
```

Figura 28. Método `__init__` para la definición de atributos de la clase IndicatorData. Fuente: Elaboración Propia.



```
1 class IndicatorDataM(CodeAgent):
2     def __init__(self, dicc, start_date, end_date, name, email, token):
3         self.dicc = dicc
4         self.start_date = start_date
5         self.end_date = end_date
6         self.name = name
7         self.email = email
8         self.token = token
```

Figura 29. Método `__init__` para la definición de atributos de la clase IndicatorDataM. Fuente: Elaboración Propia.

En IndicatorDataM se utiliza un diccionario justo por la necesidad de agrupar más de una variable, a diferencia de la funcionalidad IndicatorData. Posterior a esta definición, mediante el método `run` se ejecuta la tarea principal de esta funcionalidad, realizando la solicitud al Servicio Web del BCCR:



```

1 def run(self):
2     # Definir La URL del servicio SOAP
3     url = "https://gee.bccr.fi.cr/Indicadores/Suscripciones/WS/wsindicadoreseconomicos.asmx"
4     # Definir Los parámetros SOAP
5     soap_request = f"""<?xml version="1.0" encoding="utf-8"?>
6     <soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
7     xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
8     <soap:Body>
9     <ObtenerIndicadoresEconomicos xmlns="http://ws.sdde.bccr.fi.cr">
10    <Indicador>{self.indicator}</Indicador>
11    <FechaInicio>{self.start_date}</FechaInicio>
12    <FechaFinal>{self.end_date}</FechaFinal>
13    <Nombre>{self.name}</Nombre>
14    <SubNiveles>N</SubNiveles>
15    <CorreoElectronico>{self.email}</CorreoElectronico>
16    <Token>{self.token}</Token>
17    </ObtenerIndicadoresEconomicos>
18    </soap:Body>
19    </soap:Envelope>"""
20     # Headers necesarios para la solicitud SOAP
21     headers = {
22         'Content-Type': 'text/xml; charset=utf-8',
23         'SOAPAction': 'http://ws.sdde.bccr.fi.cr/ObtenerIndicadoresEconomicos'
24     }
25     # Realizar la solicitud POST
26     response = requests.post(url, data=soap_request, headers=headers)
27     # Parsear el XML de la respuesta
28     root = ET.fromstring(response.content)
29     # Extraer los datos de interés
30     data = []
31     for item in root.findall(".//INGC011_CAT_INDICADORECONOMIC"):
32         cod_ind = item.find("COD_INDICADORINTERNO").text
33         fecha = item.find("DES_FECHA").text.replace("T00:00:00-06:00", "") # Limpiar la fecha
34         valor = float(item.find("NUM_VALOR").text) # Convertir a flotante
35     # Agregar a la lista de datos
36     data.append({
37         "CÓDIGO_IND": cod_ind,
38         "FECHA": fecha,
39         "VALOR": valor,
40         "INDICADOR": self.variable
41     })
42     # Crear el DataFrame
43     df_ind = pd.DataFrame(data)

```

Figura 30. Método run para ejecución de la funcionalidad de IndicatorData. Fuente: Elaboración Propia.

En este extracto de código es fácilmente observable el proceso de obtención de los datos y su posterior transformación en un dataframe para facilitar su manipulación. Además, dentro de este mismo método se incluyen los primeros extractos del reporte que resultará de la ejecución de tareas del agente.

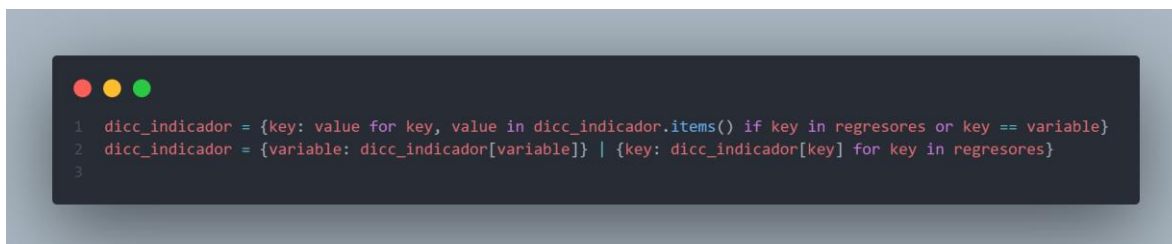
```

1 # Imprimir detalles
2 report = f"## Datos del indicador {df_ind['INDICADOR'][0]}"
3 report += f"\n\n{self.variable} al Cierre:** {df_ind['VALOR'].iloc[-1]}"
4 report += "\n\n**Período seleccionado:**"
5 report += f"\n\n**Fecha inicial:** {self.start_date}"
6 report += f"\n\n**Fecha final:** {self.end_date}\n\n"
7 report += f"\n\n**Cantidad de registros base:** {len(df_ind)}"
8 report += f"\n\n**Datos Importantes de la Proyección:**"
9 return report, df_ind

```

Figura 31. Estructura inicial del Reporte Final. Fuente: Elaboración Propia.

En cuanto a IndicatorDataM, la principal diferencia es la necesidad de un bucle for que obtenga los datos para cada una de las variables, recorriendo el diccionario que se forma a partir de las selecciones realizadas por el usuario en la app web.

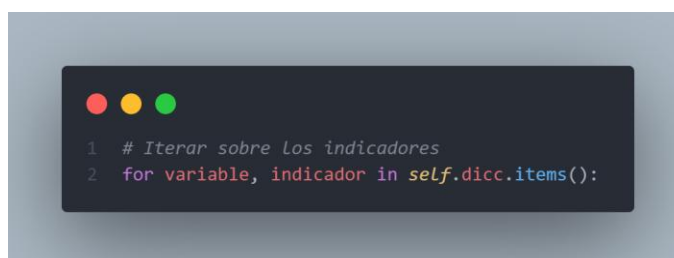


```

1  dicc_indicador = {key: value for key, value in dicc_indicador.items() if key in regresores or key == variable}
2  dicc_indicador = {variable: dicc_indicador[variable]} | {key: dicc_indicador[key] for key in regresores}
3

```

Figura 32. Definición del diccionario con las variables seleccionadas. Fuente: Elaboración Propia.



```

1  # Iterar sobre Los indicadores
2  for variable, indicador in self.dicc.items():

```

Figura 33. Proceso de Iteración para obtención de datos. Fuentes: Elaboración Propia.

Al igual que con la funcionalidad para la obtención de datos del análisis autorregresivo, en IndicatorDataM se incluye la primera sección del reporte que será generado al finalizar el trabajo por parte del agente.



```

1  report = f"## Datos del indicador {df_Ind_s.columns[1]}"
2  report += f"\n\n**{df_Ind_s.columns[1]} al Cierre:** {df_report.iloc[-1, 1]:.2f}"
3  report += f"\n\n**Fecha de último dato disponible:** {df_report['FECHA'].iloc[-1]}"
4  report += "\n\n**Período seleccionado:**"
5  report += f"\n\n**Fecha inicial:** {self.start_date}"
6  report += f"\n\n**Fecha final:** {self.end_date}"
7  report += f"\n\n**Cantidad de registros base:** {len(df_report)}"
8  report += f"\n\n**Datos Importantes de la Proyección:**"
9  report += f"\n\n**Cantidad de Predictores adicionales:** {len(df_Ind_s.columns[2:])}"
10 report += f"\n\n**Predictores adicionales:** {columns}"
11 return report, df_Ind_s

```

Figura 34. Estructura inicial del Reporte Final del Análisis de Regresión Múltiple. Fuente: Elaboración Propia.

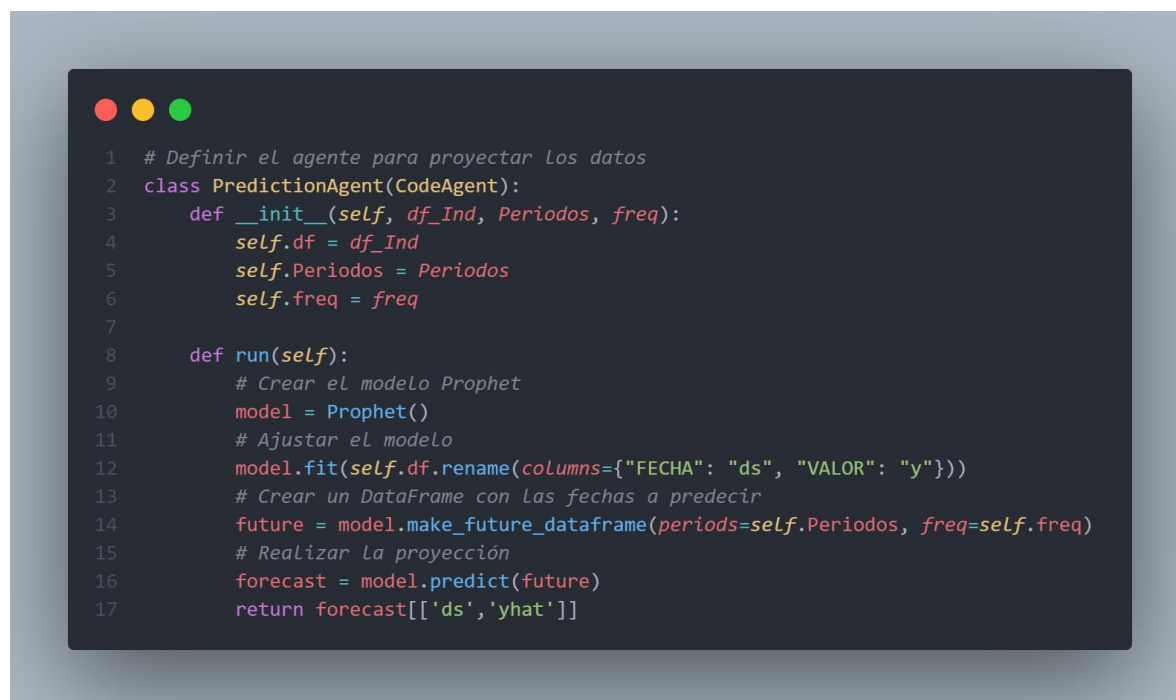
Explicado el trabajo de ambas funcionalidades, se puede proceder con las dos funcionalidades siguientes: la encargada de la generar las proyecciones para cada tipo de análisis y con la cual el

agente es capaz de generar el objeto visual donde se presenta el comportamiento real del indicador en contraposición de las proyecciones generadas.

PredictionAgent y PredictionAgentM, a partir de los datos obtenidos y depurados, tomando los periodos a proyectar seleccionados por el usuario y utilizando la librería prophet para crear el modelo, generan las proyecciones de la variable objetivo. En el caso de PredictionAgent al ser un trabajo de proyección autorregresivo su desarrollo fue más sencillo, implican solo los 4 pasos básicos de este tipo de proyecciones:

- Creación del modelo.
- Entrenamiento del modelo con los datos históricos
- Creación del dataframe con las fechas que serán proyectadas, y
- Propiamente el desarrollo de las proyecciones

A partir del desarrollo de estos pasos se obtiene una nueva serie de datos con las proyecciones obtenidas. En la siguiente ilustración se puede apreciar con la claridad el desarrollo de esta funcionalidad para el análisis autorregresivo.



```
1  # Definir el agente para proyectar Los datos
2  class PredictionAgent(CodeAgent):
3      def __init__(self, df_Ind, Periodos, freq):
4          self.df = df_Ind
5          self.Periodos = Periodos
6          self.freq = freq
7
8      def run(self):
9          # Crear el modelo Prophet
10         model = Prophet()
11         # Ajustar el modelo
12         model.fit(self.df.rename(columns={"FECHA": "ds", "VALOR": "y"}))
13         # Crear un DataFrame con las fechas a predecir
14         future = model.make_future_dataframe(periods=self.Periodos, freq=self.freq)
15         # Realizar la proyección
16         forecast = model.predict(future)
17         return forecast[['ds', 'yhat']]
```

Figura 35. Funcionalidad PredictionAgent. Fuente: Elaboración Propia

En el caso de la funcionalidad PredictionAgentM, dadas las condiciones propias de incluir como regresores en el desarrollo de las proyecciones algunas variables que pueden contar con diferentes periodicidades, fue necesario el desarrollo de una depuración más profunda sobre los datos, ya que

antes de crear las proyecciones se volvía requisito empatar la cantidad de datos de los regresores con la cantidad de datos disponibles del indicador objetivo que es definido al inicio de la interacción con el agente. Por lo tanto, el trabajo realizado para esta funcionalidad se puede segmentar en tres secciones:

1. Creación del modelo para proyectar cada variable utilizada como regresor, depuración de los datos y generación de proyecciones para cada variable, indistintamente su periodicidad.
2. Formatear el dataframe de proyecciones de acuerdo con la periodicidad de la variable objetivo.
3. Inclusión de las proyecciones de los regresores en el dataframe que será utilizado para la generación de las proyecciones de la variable objetivo.

Como parte del primer paso, dado que las variables utilizadas como regresores claramente no cuentan con datos en el futuro, fue necesario generar las proyecciones para cada una a partir de su comportamiento histórico. Posteriormente, estos datos se convierten en parte del conjunto de datos utilizado para proyectar la variable objetivo.



```
1 # Crear dataframe de proyecciones
2 df_proy = pd.DataFrame()
3 # Crear modelos para cada indicador
4 for col in self.df.columns[2:]:
5     # Crear el modelo Prophet
6     model = Prophet()
7     # Mantener dataframe sin NA's
8     df_n = self.df[["FECHA", col]].dropna()
9     # Ajustar el modelo
10    model.fit(df_n.rename(columns={"FECHA": "ds", col: "y"}))
11    # Crear un DataFrame con las fechas a predecir
12    if self.dicc[col][1] == 'D':
13        future = model.make_future_dataframe(
14            periods=int(self.Periodos * (1 if self.freq == 'D' else 30 if self.freq == 'M' else 90)), freq='D')
15    elif self.dicc[col][1] == 'M':
16        future = model.make_future_dataframe(
17            periods=int(self.Periodos * (1 if self.freq == 'M' else 1/30 if self.freq == 'D' else 3)), freq='M')
18    elif self.dicc[col][1] == 'Q':
19        future = model.make_future_dataframe(
20            periods=int(self.Periodos * (1 if self.freq == 'Q' else 1/90 if self.freq == 'D' else 1/3)), freq='Q')
21    # Realizar la proyección
22    forecast = model.predict(future)
23    # Agregar a dataframe
24    df_proy_1 = pd.DataFrame()
25    df_proy_1[f"{col}_proy"] = forecast["yhat"]
26    df_proy = pd.concat([df_proy, df_proy_1], axis=1)
```

Figura 36. Sección Nro.1 de la Función PredictionAgentM. Fuente: Elaboración Propia

Cada variable es proyectada de acuerdo con la cantidad de periodos seleccionados y ajustada de acuerdo con su periodicidad de actualización. Por ejemplo, al seleccionar un indicador objetivo con actualización diaria implicaría que para proyectar una variable regresora con periodicidad mensual,

la cantidad de periodos a proyectar debería ser dividida entre al menos 30 días para obtener la cantidad de meses, tal como se puede apreciar en la figura anterior.

Posterior a la unificación de los datos proyectados de cada variable regresora con el dataframe original, será necesaria una depuración adicional, dado que, desde un inicio se tienen datos faltantes o datos adicionales que no serán requeridos de acuerdo con la periodicidad de la variable objetivo. Volviendo al ejemplo, al ser la variable objetivo de periodicidad diaria, tanto los datos de la variable regresora como sus proyecciones serán únicamente para las fechas correspondientes al cierre de cada mes, por lo que será necesario completar los valores para las fechas faltantes con los datos disponibles. De lo anterior se encarga la segunda y la tercera sección de esta función.



```
1 # Formatear dataframe de acuerdo con peridiocidad
2 df_or = self.df.copy()
3 if self.dicc[df_or.columns[1]][1] == 'D':
4     # Completar NA's
5     df_or.fillna(method='ffill', inplace=True)
6     # Borrar NA's restantes
7     df_or.dropna(inplace=True)
8 elif self.dicc[df_or.columns[1]][1] == 'M':
9     # Borrar NA's de la segunda columna
10    df_or.dropna(subset=[list(self.dicc.keys())[0]], inplace=True)
11    # Completar NA's
12    df_or.fillna(method='ffill', inplace=True)
13    # Borrar NA's restantes
14    df_or.dropna(inplace=True)
15 elif self.dicc[df_or.columns[1]][1] == 'Q':
16     # Borrar NA's
17     df_or.dropna(inplace=True)
```

Figura 37. Sección Nro.2 de la Función PredictionAgentM. Fuente: Elaboración Propia

El código anterior hace lo correspondiente sobre el dataframe con los datos históricos, haciendo la depuración necesaria o completando los valores faltantes. El siguiente código hace este mismo trabajo, pero sobre las proyecciones creadas para cada regresor seleccionado, basándose en la frecuencia de actualización de la variable objetivo. Dado lo extenso del código de esta sección, se presenta únicamente el caso de periodicidad mensual, ya que implicó una carga considerable de trabajo por la cantidad de aspectos involucrados.

```

1 elif self.freq == 'M':# TODO: Listo
2     if self.dicc[col][1] == 'M':
3         df_proy_3 = df_proy.copy()
4         df_proy_3.dropna(inplace=True) # !Cambio fundamental
5         future[col] = pd.concat([df_or[col], df_proy_3[f"{col}_proy"][-self.Periodos:]], ignore_index=True)
6     elif self.dicc[col][1] == 'D':
7         factor = self.Periodos*30
8         df_proy_2[f"{col}_proy"] = df_proy[f"{col}_proy"][-factor:].values
9         dates_next_months = [(last_date + pd.DateOffset(months=i)).to_period("M").end_time.normalize() for i in range(1, self.Periodos+1)]
10        star_date = pd.to_datetime(self.end_date)
11        dates_next = pd.date_range(start=star_date + pd.Timedelta(days=1), periods=factor, freq='D')
12        df_proy_2['Fecha'] = dates_next
13        df_proy_2 = df_proy_2[df_proy_2['Fecha'].isin(dates_next_months)]
14        future[col] = pd.concat([df_or[col], df_proy_2[f"{col}_proy"]], ignore_index=True)
15        df_proy_2 = pd.DataFrame()
16    elif self.dicc[col][1] == 'Q':
17        factor = 3
18        df_proy_3 = df_proy.copy()
19        df_proy_3.dropna(inplace=True) # !Cambio fundamental
20        df_proy_2[f"{col}_proy"] = np.repeat(df_proy_3[f"{col}_proy"].values, factor)
21        future[col] = pd.concat([df

```

Figura 38. Sección Nro.3 de la Función PredictionAgentM, Secuencia Mensual. Fuente: Elaboración Propia

Con estas tres tareas finalizadas, el dataset base se encuentra listo para realizar las proyecciones indicadas del indicador objetivo. Esto se realiza de forma fácil con el módulo .predict de model. Las funciones PredictionAgent y PredictionAgentM retornan solamente las proyecciones generadas.

```

1 # Realizar la proyección
2 future = future.dropna()
3 forecast = model.predict(future)
4 # Retornar proyecciones
5 return forecast[['ds','yhat']]

```

Figura 39. Generación de proyecciones finales. Fuente: Elaboración Propia

Hasta el momento ya la herramienta cuenta con todos los datos suficientes para generar la primera parte del reporte, tal como se visualiza en las siguientes imágenes, tanto para el análisis autorregresivo como para el análisis de regresión múltiple.

## Reporte de Análisis: TC de Compra

### Datos del indicador TC de Compra

**TC de Compra al Cierre:** 498.68

**Periodo seleccionado:**

**Fecha inicial:** 2020/01/01

**Fecha final:** 2025/04/17

**Cantidad de registros base:** 1934

**Datos Importantes de la Proyección:**

**Fecha final de proyección:** 2025/10/14

**Predicción del indicador:** 510.34

Figura 40. Primera Sección Reporte Análisis Autorregresivo. Fuente: Elaboración Propia

## Reporte de Análisis: TC de Compra

### Datos del indicador TC de Compra

**TC de Compra al Cierre:** 504.30

**Fecha de último dato disponible:** 2025-05-22

**Periodo seleccionado:**

**Fecha inicial:** 2020/01/01

**Fecha final:** 2025/05/22

**Cantidad de registros base:** 1969

**Datos Importantes de la Proyección:**

**Cantidad de Predictores adicionales:** 2

**Predictores adicionales:** TBP - IMAE

**Fecha final de proyección:** 2025/11/18

**Predicción del indicador:** 509.92

Figura 41. Primera Sección Reporte Análisis Regresión Múltiple. Fuente: Elaboración Propia

Con los datos históricos depurados y las proyecciones generadas, las funcionalidades ChartAgent y ChartAgentM toman las salidas anteriores y desarrollan el gráfico con el comparativo de los datos reales vs los datos proyectados. Ambos retornan la figura lista para su inclusión en el reporte.



```

1 # Definir el agente para graficar Los datos
2 class ChartAgent(CodeAgent):
3     def __init__(self, df_Ind, forecast):
4         self.df = df_Ind
5         self.forecast = forecast
6
7     def run(self):
8         # Convertir la columna FECHA a datetime
9         self.df["FECHA"] = pd.to_datetime(self.df["FECHA"])
10        # Crear la gráfica
11        fig, ax = plt.subplots(figsize=(10, 6))
12        ax.plot(self.df["FECHA"], self.df["VALOR"], color="green", label=f"{self.df['INDICADOR'][0]}")
13        ax.plot(self.forecast["ds"], self.forecast["yhat"], color="red", linestyle="--", label="Predicción")
14        ax.set_title(f"Indicador {self.df['INDICADOR'][0]}")
15        ax.set_xlabel("Fecha")
16        ax.set_ylabel("Valor")
17        ax.xaxis.set_major_formatter(mdates.DateFormatter('%m-%Y'))
18        ax.xaxis.set_major_locator(plt.MaxNLocator(10))
19        plt.xticks(rotation=45)
20        plt.legend()
21        return fig

```

Figura 42. Funcionalidad ChartAgent. Fuente: Elaboración Propia

```

1 # Definir el agente para graficar Los datos
2 class ChartAgentM(CodeAgent):
3     def __init__(self, df_Ind, forecast):
4         self.df = df_Ind
5         self.forecast = forecast
6
7     def run(self):
8         # Convertir la columna FECHA a datetime
9         self.df['FECHA'] = pd.to_datetime(self.df['FECHA'])
10        # Crear la gráfica
11        fig, ax = plt.subplots(figsize=(10, 6))
12        df_chart = self.df.dropna(subset=self.df.columns[1])
13        ax.plot(df_chart["FECHA"], df_chart[df_chart.columns[1]], color="green", label=f"{df_chart.columns[1]}")
14        ax.plot(self.forecast["ds"], self.forecast["yhat"], color="red", linestyle="--", label="Predicción")
15        ax.set_title(f"Indicador {df_chart.columns[1]}")
16        ax.set_xlabel("Fecha")
17        ax.set_ylabel("Valor")
18        ax.xaxis.set_major_formatter(mdates.DateFormatter('%m-%Y'))
19        ax.xaxis.set_major_locator(plt.MaxNLocator(10))
20        plt.xticks(rotation=45)
21        plt.legend()
22        return fig

```

Figura 43. Funcionalidad ChartAgentM. Fuente: Elaboración Propia

La inclusión de este gráfico en el reporte no se realiza directamente en el código de cada funcionalidad, sino en el desarrollo como tal de la aplicación. Esto fue necesario dado que se deseaba generar un archivo html como producto final del agente. Para ello, se realiza una conversión del objeto fig a un formato de imagen admisible.





Figura 44. Conversión del objeto fig a base64. Fuente: Elaboración Propia

Ya con esta conversión se logra presentar el gráfico no solo en la herramienta, sino también en el archivo HTML final.

#### Gráfica del indicador

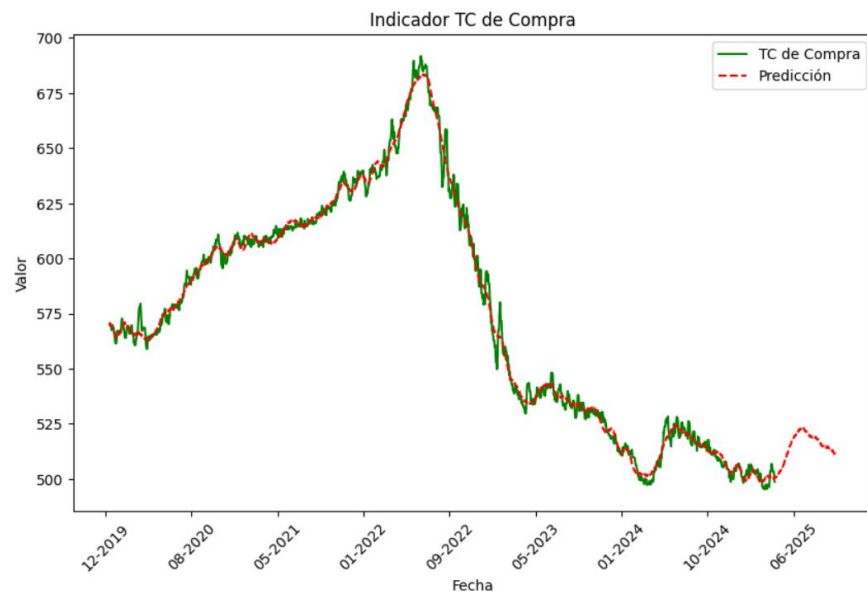


Figura 45. Comportamiento Real vs Proyección. Fuente: Elaboración Propia

Ya el agente ha desarrollado tres de sus cuatro funcionalidades, generando las 2/3 partes del reporte. Con este trabajo ya completado, la cuarta funcionalidad entra en funcionamiento, NewsAgent. A diferencia de las funcionalidades anteriores, para esta última no fue necesario crear una diferente para cada tipo de análisis, dado que el trabajo a realizar es indistinto del tipo de regresión utilizado.

Esta funcionalidad desarrolla dos actividades principales:

- Búsqueda en la Web de noticias relacionadas con el indicador objetivo en la fecha cercana al último dato disponible, utilizando para ello el servicio de DuckDuckGo.

- Toma las noticias obtenidas y se las envía al Modelo de Lenguaje disponible en la nube de Groq junto con la proyección final obtenida para el respectivo análisis. Para ello también se incluyen tres aspectos fundamentales: el system prompt que define el trabajo a realizar por el modelo, el user prompt que introduce las noticias obtenidas y el formato deseado para la salida que nos dará el Modelo, siendo definido mediante el módulo BaseModel de la librería pydantic. Con este último aspecto se asegura la estandarización de la respuesta obtenida.

A screenshot of a code editor with a dark background and light-colored text. The code is in Python and defines a search tool and a query. It includes comments in Spanish. The code is as follows:

```
1 # Definir herramienta de búsqueda
2 search_tool = DuckDuckGoSearchTool()
3 # Definir el tema de la búsqueda
4 query = f"Noticias de Costa Rica de {self.variable} cercanas al {self.end_date}"
5 # Realizar la búsqueda
6 results = search_tool(query)
7
```

Figura 46. Definición y Ejecución de la Búsqueda de Noticias. Fuente: Elaboración Propia

Dada la extensión típica de la cadena de razonamiento del modelo DeepSeek-R1, se consideró que como parte del informe final solamente se incluya un resumen de las principales diez ideas arrojadas por el modelo, además de la conclusión general del análisis desarrollado. Lo anterior se logra con la definición de la salida estructurada, tal como se mencionó anteriormente.

A screenshot of a code editor with a dark background and light-colored text. The code is in Python and defines a class named NewsResume that inherits from BaseModel. It lists attributes for structured output, including the number of news items and ten chains of thought, followed by the news analysis. The code is as follows:

```
1 # Definir salida estructurada
2 class NewsResume(BaseModel):
3     cantidad_noticias: int
4     cadena_pensamiento_1: str
5     cadena_pensamiento_2: str
6     cadena_pensamiento_3: str
7     cadena_pensamiento_4: str
8     cadena_pensamiento_5: str
9     cadena_pensamiento_6: str
10    cadena_pensamiento_7: str
11    cadena_pensamiento_8: str
12    cadena_pensamiento_9: str
13    cadena_pensamiento_10: str
14    Analisis_noticias: str
```

Figura 47. Definición Salida Estructurada. Fuente: Elaboración Propia

El system prompt proporciona al LLM las instrucciones que debe cumplir para el desarrollo del análisis, lo cual delimita el trabajo del modelo y contribuye a una mayor alineación entre el resultado que se desea y el que se obtiene. El siguiente fragmento de texto corresponde al system prompt:

*“Eres un asistente virtual especializado en el análisis de variables económicas. A partir de un conjunto de noticias que se te proporcionan, tu tarea es:*

- 1. Identificar y analizar de forma individual cómo cada noticia se relaciona con una variable económica específica.*
- 2. Posteriormente, integrar la información de las distintas noticias para detectar patrones, validar supuestos y generar una base sólida para el análisis.*
- 3. Evaluar una proyección que se te indica respecto a la variable económica, determinando en qué medida es consistente o factible en función de la evidencia encontrada en las noticias.*
- 4. Elaborar una conclusión argumentada sobre lo que puede esperarse en los próximos meses para la variable económica analizada, basándote exclusivamente en la información contenida en las noticias.*

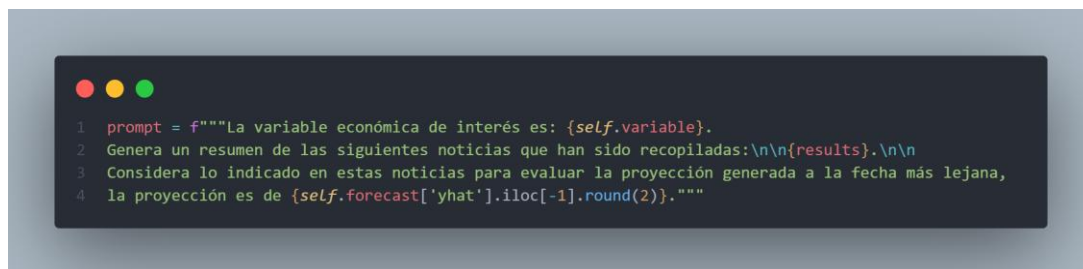
*Aspectos estructurales a considerar:*

*\* Toda tu Cadena de Pensamiento debe estar en español, al igual que tu respuesta final.*

*\* El análisis debe ser riguroso, ordenado y con enfoque crítico.*

*\* Justifica claramente tus evaluaciones y conclusiones, especificando los elementos que respaldan tu juicio sobre la proyección dada.”*

El user prompt le realiza una indicación simple al modelo, donde indica la variable de interés, la proyección obtenida e incluye las noticias obtenidas de la búsqueda web realizada.



```
1 prompt = f"""La variable económica de interés es: {self.variable}.
2 Genera un resumen de las siguientes noticias que han sido recopiladas:\n\n{results}.\n\n
3 Considera lo indicado en estas noticias para evaluar la proyección generada a la fecha más lejana,
4 la proyección es de {self.forecast['yhat'].iloc[-1].round(2)}."""
```

Figura 48. User prompt. Fuente: Elaboración Propia

Con cada aspecto definido, se procede con la generación de la respuesta del LLM, a partir de la estructura estandarizada para la interacción con estos modelos.



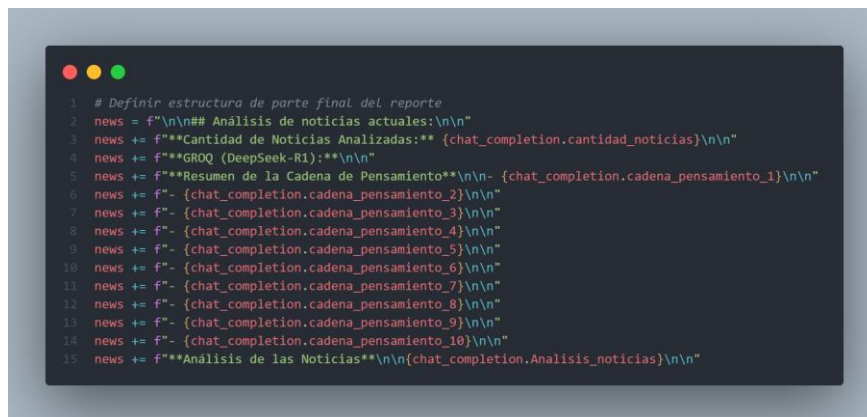
```

1 chat_completion = self.client.chat.completions.create(
2     messages=[
3         {
4             "role": "system",
5             "content": system_prompt
6         },
7         {
8             "role": "user",
9             "content": prompt
10        }
11    ],
12    model="deepseek-r1-distill-llama-70b",
13    response_model=NewsResume
14 )

```

Figura 49. Generación de Respuesta. Fuente: Elaboración Propia

Con la respuesta del LLM generada, ya se cuentan con todos los insumos necesarios para la sección final del reporte del agente. Acá se incluyen las principales cadenas de pensamiento del modelo resumidas, así como la conclusión respecto al análisis de las noticias y la proyección obtenida.



```

1 # Definir estructura de parte final del reporte
2 news = f"\n\n## Análisis de noticias actuales:\n\n"
3 news += f"**Cantidad de Noticias Analizadas:** {chat_completion.cantidad_noticias}\n\n"
4 news += f"**GROQ (DeepSeek-R1):**\n\n"
5 news += f"**Resumen de la Cadena de Pensamiento**\n\n- {chat_completion.cadena_pensamiento_1}\n\n"
6 news += f"- {chat_completion.cadena_pensamiento_2}\n\n"
7 news += f"- {chat_completion.cadena_pensamiento_3}\n\n"
8 news += f"- {chat_completion.cadena_pensamiento_4}\n\n"
9 news += f"- {chat_completion.cadena_pensamiento_5}\n\n"
10 news += f"- {chat_completion.cadena_pensamiento_6}\n\n"
11 news += f"- {chat_completion.cadena_pensamiento_7}\n\n"
12 news += f"- {chat_completion.cadena_pensamiento_8}\n\n"
13 news += f"- {chat_completion.cadena_pensamiento_9}\n\n"
14 news += f"- {chat_completion.cadena_pensamiento_10}\n\n"
15 news += f"**Análisis de las Noticias**\n\n{chat_completion.Analisis_noticias}\n\n"

```

Figura 50. Estructura Parte Final del Reporte. Fuente: Elaboración Propia

Finalizada la definición de la función NewsAgent, ya se ha completado la definición de las capacidades del agente y con ello se ha completado la estructura del reporte final.

## Análisis de noticias actuales:

**Cantidad de Noticias Analizadas:** 10

**GROQ (DeepSeek-R1):**

### Resumen de la Cadena de Pensamiento

- La primera noticia informa sobre la caída del tipo de cambio en Costa Rica a 513 colones por dólar, con un análisis de tendencias y recomendaciones para 2025. Esto indica una disminución reciente en el valor del dólar, lo que podría afectar la variable de TC de Compra.
- La segunda noticia menciona las proyecciones de economistas sobre un tipo de cambio promedio entre ₡498 y ₡500 para el primer trimestre de 2025, lo que sugiere estabilidad o leve depreciación del colón.
- La tercera noticia se centra en el análisis de Javier Cortés sobre los macroprecios clave, incluyendo el tipo de cambio, lo que ofrece una visión integral del panorama económico para 2025.
- La cuarta noticia es un enlace general de economía sin detalles específicos, por lo que no aporta información directa sobre el TC de Compra.
- La quinta noticia proporciona datos históricos del tipo de cambio de compra y venta del dólar, lo que es útil para analizar tendencias a largo plazo.
- La sexta noticia incluye un análisis actualizado del tipo de cambio, mencionando un valor de compra de ₡506.41, lo que indica una tendencia estable.
- La séptima noticia repite el análisis de Javier Cortés, reforzando la importancia de considerar los macroprecios en el análisis económico.
- La octava noticia cita a Daniel Suchar, quien predice que el tipo de cambio oscilará entre ₡550 y ₡580 en 2025, lo que sugiere un posible aumento en el valor del dólar.
- La novena noticia informa sobre el comportamiento reciente del tipo de cambio, con un incremento en la compra a ₡502.58, rompiendo una tendencia a la baja.
- La décima noticia menciona el presupuesto 2025, con un tipo de cambio previsto por debajo de ₡520, lo que indica expectativas de estabilidad o leve disminución del valor del dólar.

### Análisis de las Noticias

La mayoría de las noticias sugieren que el tipo de cambio de compra del dólar en Costa Rica se mantendrá estable o experimentará una leve variación en 2025. Las proyecciones de los economistas y los datos recientes indican que el valor del dólar podría oscilar en un rango cercano a los ₡500-₡520. La proyección de 509.92 está dentro de este rango y es consistente con las tendencias actuales y las expectativas de los expertos.

Figura 51. Análisis de Noticias del Agente. Fuente: Elaboración Propia

La definición del código para la interfaz web, principalmente implicó el desarrollo de los aspectos ya visualizados en imágenes anteriores, como entradas de datos, cajas de selección, botones de acción y otras estructuras propias del desarrollo de aplicaciones utilizando Streamlit, por lo tanto, no serán abarcados en esta sección de resultados, no obstante, será incluido como un anexo, además de ponerlo a disposición en un repositorio público.

El agente ha sido desarrollado por completo, ofreciendo una interfaz web para facilitar la interacción, capaz de obtener la información necesaria, generar las proyecciones, analizarlas junto con noticias relacionadas con el indicador de interés y ofrecer un análisis autoevaluativo de grado profesional para la toma de decisiones. El mismo se encontrará disponible durante el período de evaluación de este trabajo en el siguiente enlace: <https://proyecto-tfmds.azurewebsites.net/>

## 5. Limitaciones

A pesar de contar con un agente funcional que permite cumplir con los objetivos definidos, se identificaron una serie de limitaciones que pueden afectar la generación del reporte tal como ha sido diseñado. Es importante tomar en consideración estos aspectos al utilizar este agente.

### 1. Cantidad de consultas disponibles del servicio DuckDuckGo

El servicio DuckDuckGo tiene un número limitado de consultas diarias disponibles por usuario dada su característica de funcionalidad gratuita. Por lo tanto, al utilizar el agente se debe considerar que un número continuo de generaciones puede bloquear el servicio y ser necesario esperar varias horas para utilizarlo de nuevo. No obstante, la finalidad del agente no es generar análisis de forma continua, sino ser utilizado de manera puntual para apoyar la toma de decisiones.

### 2. Indicadores disponibles para consulta

Actualmente el agente solo permite la consulta de 8 indicadores asociados a la economía costarricense, por lo tanto, en caso de que se desee desarrollar un análisis de otra variable, deberá incluirse desde cero en las capacidades del agente, específicamente dentro de su capacidad de obtención de datos.

### 3. Límites de uso de la API de GroqCloud

El servicio gratuito de la nube de Groq, a pesar de ser una opción sumamente útil por poner a disposición Modelos de Lenguaje con grandes capacidades, cuenta con medidas de control para regular la frecuencia con la que los usuarios y las aplicaciones pueden acceder a la API dentro de plazos específicos. Lo anterior implica un bloqueo en cuanto a la cantidad de tokens generados y consultas que se pueden realizar, no obstante, durante las distintas generaciones realizadas no se tuvo problemas con el servicio, pero existe la posibilidad de que en el futuro si se presenten inconvenientes.

## 6. Conclusiones

El proyecto demostró la viabilidad de construir un agente inteligente que es capaz de automatizar el análisis de variables económicas mediante el uso de modelos de lenguaje avanzados (LLMs) y herramientas de aprendizaje automático. La arquitectura modular, el uso de bibliotecas como SmolAgents y Prophet, y la integración con servicios externos como GroqCloud y el servicio web del BCCR, permitieron desarrollar una solución funcional.

El agente logra automatizar varias tareas que con anterioridad requerían mayor intervención humana, como la recopilación de datos económicos, la generación de proyecciones y el análisis cualitativo de noticias. Esto no solo reduce el tiempo requerido para obtener datos económicos de valor, sino que mejora la objetividad y trazabilidad de los análisis que son realizados.

Uno de los puntos más fuertes es su interfaz web que permite a cualquier usuario, incluso sin conocimientos técnicos avanzados, interactuar con el agente, seleccionar las variables económicas de interés, definir los periodos de análisis y obtener un reporte robusto de manera sencilla. Esto apertura el acceso a modelos complejos de análisis económico.

La estructura del código desarrollado y los servicios utilizados permiten ampliar con facilidad el agente a nuevas variables, fuentes de datos o modelos más avanzados. Adicionalmente, la capacidad de despliegue en contenedores y su integración con servicios en la nube de Azure dan garantía para que el agente pueda ser replicado y utilizado en distintos entornos operativos sin necesidad de ajustes mayores.

Ya para finalizar, es cierto que se identificaron ciertas limitaciones, como el número limitado de indicadores soportados, el uso gratuito restringido de APIs o la dependencia de servicios externos; sin embargo, estas no comprometen la funcionalidad general del agente. Estos aspectos son fácilmente abordables con actualizaciones futuras, acceso a versiones de pago o mejoras en la arquitectura del desarrollada.

## 7. Referencias Bibliográficas

- Ramakrishnan, R. (2025, marzo 24). *When to use GenAI versus predictive AI*. MIT Sloan Management Review. <https://sloanreview.mit.edu/article/when-to-use-genai-versus-predictive-ai/>
- Escandell Montero, P. (2014). *Aprendizaje por refuerzo en espacios continuos: algoritmos y aplicación al tratamiento de la anemia renal* [Tesis doctoral, Universitat de València]. RODERIC. <https://roderic.uv.es/rest/api/core/bitstreams/b58898d5-57e9-4b01-8917-06649854b35c/content>
- Huet, P. (2023, abril 13). *Qué son las redes neuronales y sus aplicaciones*. OpenWebinars. <https://openwebinars.net/blog/que-son-las-redes-neuronales-y-sus-aplicaciones/>
- Silva, S., & Freire, E. (2019, noviembre 23). *Intro a las redes neuronales convolucionales*. Bootcamp AI. <https://bootcampai.medium.com/redes-neuronales-convolucionales-5e0ce960caf8>
- Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). *Learning representations by back-propagating errors*. Nature, **323**(6088), 533-536. <https://www.nature.com/articles/323533a0>
- LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). *Gradient-based learning applied to document recognition*. Proceedings of the IEEE, **86**(11), 2278-2324. <https://ieeexplore.ieee.org/document/726791>
- Hochreiter, S., & Schmidhuber, J. (1997). *Long short-term memory*. Neural Computation, **9**(8), 1735-1780. <https://direct.mit.edu/neco/article-abstract/9/8/1735/6109/Long-Short-Term-Memory?redirectedFrom=fulltext>
- Ferro, C., Celis Mayorga, N., & Casallas García, A. (2020). *Llenado de series de datos de 2014 a 2019 de PM2.5 por medio de una red neuronal y una regresión lineal* [Figura 3]. ResearchGate. [https://www.researchgate.net/figure/Celda-de-una-red-neuronal-recurrente-LSTM-Donde-los-valores-de-C-son-los\\_fig3\\_343450066](https://www.researchgate.net/figure/Celda-de-una-red-neuronal-recurrente-LSTM-Donde-los-valores-de-C-son-los_fig3_343450066)
- Jurafsky, D., & Martin, J. H. (2021). *Speech and Language Processing* (3rd ed.). Prentice Hall. <https://web.stanford.edu/~jurafsky/slp3/>
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). *Attention is All You Need*. Advances in Neural Information Processing Systems (NeurIPS). <https://arxiv.org/abs/1706.03762>



- Liu, X., Lou, X., Jiao, J., & Zhang, J. (2024). *Position: Foundation agents as the paradigm shift for decision making*. arXiv. <https://doi.org/10.48550/arXiv.2405.17009>
- DeepSeek-AI. (2025). *DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning*. Hugging Face. <https://huggingface.co/deepseek-ai/DeepSeek-R1>
- González, M. L. (17 de mayo de 2024). ¿Qué son y cómo funcionan los ChatBots y Agentes IA? *ITCL Centro Tecnológico*. <https://itcl.es/blog/que-son-y-como-funcionan-los-chatbots-y-los-agentes-ia/>
- Hugging Face. (s.f.). ¿Qué es un agente? *Curso de Agentes de Hugging Face*. <https://huggingface.co/learn/agents-course/unit1/what-are-agents>

## 8. Anexos

### 8.1 Repositorio del Proyecto

[https://github.com/Cheski1610/proyecto\\_tfm](https://github.com/Cheski1610/proyecto_tfm)

## 8.2 Código archivo funciones.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import xml.etree.ElementTree as ET
import requests
import instructor
from smolagents import CodeAgent, ToolCallingAgent, DuckDuckGoSearchTool
from prophet import Prophet
from groq import Groq
from pydantic import BaseModel

# Agente autorregresivo
##-----##

# Definir el agente para obtener los datos
class IndicatorData(CodeAgent):
    def __init__(self, variable, indicator, start_date, end_date, name, email,
token):
        self.variable = variable
        self.indicator = indicator
        self.start_date = start_date
        self.end_date = end_date
        self.name = name
        self.email = email
        self.token = token

    def run(self):
        # Definir la URL del servicio SOAP
        url =
"https://gee.bccr.fi.cr/Indicadores/Suscripciones/WS/wsindicadoreseconomicos.asmx"
        # Definir los parámetros SOAP
        soap_request = f'<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<ObtenerIndicadoresEconomicos xmlns="http://ws.sdde.bccr.fi.cr">
<Indicador>{self.indicator}</Indicador>
<FechaInicio>{self.start_date}</FechaInicio>
<FechaFinal>{self.end_date}</FechaFinal>
<Nombre>{self.name}</Nombre>
<SubNiveles>N</SubNiveles>
<CorreoElectronico>{self.email}</CorreoElectronico>
<Token>{self.token}</Token>
</ObtenerIndicadoresEconomicos>
</soap:Body>
</soap:Envelope>'
        # Headers necesarios para la solicitud SOAP
        headers = {
```

```

        'Content-Type': 'text/xml; charset=utf-8',
        'SOAPAction': 'http://ws.sdde.bccr.fi.cr/ObtenerIndicadoresEconomicos'
    }
    # Realizar la solicitud POST
    response = requests.post(url, data=soap_request, headers=headers)
    # Parsear el XML de la respuesta
    root = ET.fromstring(response.content)
    # Extraer los datos de interés
    data = []
    for item in root.findall("./INGC011_CAT_INDICADORECONOMIC"):
        cod_ind = item.find("COD_INDICADORINTERNO").text
        fecha = item.find("DES_FECHA").text.replace("T00:00:00-06:00", "") #
Limpiar la fecha
        valor = float(item.find("NUM_VALOR").text) # Convertir a flotante
    # Agregar a la lista de datos
    data.append({
        "CÓDIGO_IND": cod_ind,
        "FECHA": fecha,
        "VALOR": valor,
        "INDICADOR": self.variable
    })
    # Crear el DataFrame
    df_Ind = pd.DataFrame(data)
    # Imprimir detalles
    report = f"## Datos del indicador {df_Ind['INDICADOR'][0]}"
    report += f"\n\n**{self.variable} al Cierre:** {df_Ind['VALOR'].iloc[-1]}"
    report += "\n\n**Período seleccionado:**"
    report += f"\n\n**Fecha inicial:** {self.start_date}"
    report += f"\n\n**Fecha final:** {self.end_date}\n\n"
    report += f"\n\n**Cantidad de registros base:** {len(df_Ind)}"
    report += f"\n\n**Datos Importantes de la Proyección:**"
    return report, df_Ind

# Definir el agente para graficar los datos
class ChartAgent(CodeAgent):
    def __init__(self, df_Ind, forecast):
        self.df = df_Ind
        self.forecast = forecast

    def run(self):
        # Convertir la columna FECHA a datetime
        self.df["FECHA"] = pd.to_datetime(self.df["FECHA"])
        # Crear la gráfica
        fig, ax = plt.subplots(figsize=(10, 6))
        ax.plot(self.df["FECHA"], self.df["VALOR"], color="green",
label=f"{self.df['INDICADOR'][0]}")
        ax.plot(self.forecast["ds"], self.forecast["yhat"], color="red",
linestyle="--", label="Predicción")
        ax.set_title(f"Indicador {self.df['INDICADOR'][0]}")
        ax.set_xlabel("Fecha")

```

```

        ax.set_ylabel("Valor")
        ax.xaxis.set_major_formatter(mdates.DateFormatter('%m-%Y'))
        ax.xaxis.set_major_locator(plt.MaxNLocator(10))
        plt.xticks(rotation=45)
        plt.legend()
        return fig

# Definir el agente para proyectar los datos
class PredictionAgent(CodeAgent):
    def __init__(self, df_Ind, Periodos, freq):
        self.df = df_Ind
        self.Periodos = Periodos
        self.freq = freq

    def run(self):
        # Crear el modelo Prophet
        model = Prophet()
        # Ajustar el modelo
        model.fit(self.df.rename(columns={"FECHA": "ds", "VALOR": "y"}))
        # Crear un DataFrame con las fechas a predecir
        future = model.make_future_dataframe(periods=self.Periodos, freq=self.freq)
        # Realizar la proyección
        forecast = model.predict(future)
        return forecast[['ds', 'yhat']]

# Definir el agente para obtener noticias
class NewsAgent(CodeAgent):
    def __init__(self, variable, end_date, forecast, api_key):
        self.variable = variable
        self.end_date = end_date
        self.forecast = forecast
        self.client = instructor.from_groq(Groq(api_key=api_key),
mode=instructor.Mode.JSON) # Patch Groq() con instructor

    def run(self):
        # Definir herramienta de búsqueda
        search_tool = DuckDuckGoSearchTool()
        # Definir el tema de la búsqueda
        query = f"Noticias de Costa Rica de {self.variable} cercanas al
{self.end_date}"
        # Realizar la búsqueda
        results = search_tool(query)
        # Definir salida estructurada
        class NewsResume(BaseModel):
            cantidad_noticias: int
            cadena_pensamiento_1: str
            cadena_pensamiento_2: str
            cadena_pensamiento_3: str
            cadena_pensamiento_4: str
            cadena_pensamiento_5: str

```

```

cadena_pensamiento_6: str
cadena_pensamiento_7: str
cadena_pensamiento_8: str
cadena_pensamiento_9: str
cadena_pensamiento_10: str
Analisis_noticias: str
# Definir system prompt
system_prompt = """
Eres un asistente virtual especializado en el análisis de variables
económicas. A partir de un conjunto de noticias que se te proporcionan, tu tarea
es:

1. Identificar y analizar de forma individual cómo cada noticia se
relaciona con una variable económica específica.
2. Posteriormente, integrar la información de las distintas noticias para
detectar patrones, validar supuestos y generar una base sólida para el análisis.
3. Evaluar una proyección que se te indica respecto a la variable
económica, determinando en qué medida es consistente o factible en función de la
evidencia encontrada en las noticias.
4. Elaborar una conclusión argumentada sobre lo que puede esperarse en los
próximos meses para la variable económica analizada, basándote exclusivamente en la
información contenida en las noticias.

Aspectos estructurales a considerar:

* Toda tu Cadena de Pensamiento debe estar en español, al igual que tu
respuesta final.
* El análisis debe ser riguroso, ordenado y con enfoque crítico.
* Justifica claramente tus evaluaciones y conclusiones, especificando los
elementos que respaldan tu juicio sobre la proyección dada.
"""

# Crear el resumen de las noticias con GROQ
prompt = f"""La variable económica de interés es: {self.variable}.
Genera un resumen de las siguientes noticias que han sido
recopiladas:\n\n{results}.\n\n Considera lo indicado
en estas noticias para evaluar la proyección generada a la fecha más
lejana,
la proyección es de {self.forecast['yhat'].iloc[-1].round(2)}."""
chat_completion = self.client.chat.completions.create(
    messages=[
        {
            "role": "system",
            "content": system_prompt
        },
        {
            "role": "user",
            "content": prompt
        }
    ],
    model="deepseek-r1-distill-llama-70b",

```

```

        response_model=NewsResume
    )
    # Definir estructura de parte final del reporte
    news = f"\n\n## Análisis de noticias actuales:\n\n"
    news += f"***Cantidad de Noticias Analizadas:**"
    {chat_completion.cantidad_noticias}\n\n"
    news += f"***GROQ (DeepSeek-R1):**\n\n"
    news += f"***Resumen de la Cadena de Pensamiento**\n\n"
    {chat_completion.cadena_pensamiento_1}\n\n"
    news += f"- {chat_completion.cadena_pensamiento_2}\n\n"
    news += f"- {chat_completion.cadena_pensamiento_3}\n\n"
    news += f"- {chat_completion.cadena_pensamiento_4}\n\n"
    news += f"- {chat_completion.cadena_pensamiento_5}\n\n"
    news += f"- {chat_completion.cadena_pensamiento_6}\n\n"
    news += f"- {chat_completion.cadena_pensamiento_7}\n\n"
    news += f"- {chat_completion.cadena_pensamiento_8}\n\n"
    news += f"- {chat_completion.cadena_pensamiento_9}\n\n"
    news += f"- {chat_completion.cadena_pensamiento_10}\n\n"
    news += f"***Análisis de las"
    Noticias**\n\n{chat_completion.Analisis_noticias}\n\n"
    return news

# Definir el multiagente
class MultiAgent(ToolCallingAgent):
    def __init__(self, variable, indicator, start_date, end_date, name, email,
token, Periodos, freq, api_key):
        self.variable = variable
        self.indicator = indicator
        self.start_date = start_date
        self.end_date = end_date
        self.name = name
        self.email = email
        self.token = token
        self.Periodos = Periodos
        self.freq = freq
        self.api_key = api_key

    def run(self):
        # Crear el agente para obtener los datos
        agent1 = IndicatorData(self.variable, self.indicator, self.start_date,
self.end_date, self.name, self.email, self.token)
        report, df_Ind = agent1.run()
        # Crear el agente para predecir los datos
        agent3 = PredictionAgent(df_Ind, self.Periodos, self.freq)
        forecast = agent3.run()
        # Crear el agente para graficar los datos
        agent2 = ChartAgent(df_Ind, forecast)
        fig = agent2.run()
        # crear el agente para obtener noticias
        agent4 = NewsAgent(self.variable, self.end_date, forecast, self.api_key)

```

```

        resume = agent4.run()
        # Crear el reporte final
        report += f"\n\n**Fecha final de proyección:**"
        {forecast['ds'].iloc[-1].strftime('%Y/%m/%d'))"
        report += f"\n\n**Predicción del indicador:**"
        {forecast['yhat'].iloc[-1].round(2)}"
        report += "\n\n## Gráfica del indicador"
        return report, fig, resume

##-----##

# Agente Regresión Múltiple
##-----##

# Definir el agente para obtener los datos
class IndicatorDataM(CodeAgent):
    def __init__(self, dicc, start_date, end_date, name, email, token):
        self.dicc = dicc
        self.start_date = start_date
        self.end_date = end_date
        self.name = name
        self.email = email
        self.token = token

    def run(self):
        # Definir la URL del servicio SOAP
        url =
"https://gee.bccr.fi.cr/Indicadores/Suscripciones/WS/wsindicadoreseconomicos.asmx"
        # Crear un DataFrame vacío
        df_Ind = pd.DataFrame()
        # Iterar sobre los indicadores
        for variable, indicador in self.dicc.items():
            # Definir los parámetros SOAP
            soap_request = f"""<?xml version="1.0" encoding="utf-8"?>
            <soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
            <soap:Body>
            <ObtenerIndicadoresEconomicos xmlns="http://ws.sdde.bccr.fi.cr">
            <Indicador>{indicador[0]}</Indicador>
            <FechaInicio>{self.start_date}</FechaInicio>
            <FechaFinal>{self.end_date}</FechaFinal>
            <Nombre>{self.name}</Nombre>
            <SubNiveles>N</SubNiveles>
            <CorreoElectronico>{self.email}</CorreoElectronico>
            <Token>{self.token}</Token>
            </ObtenerIndicadoresEconomicos>
            </soap:Body>
            </soap:Envelope>"""
            # Headers necesarios para la solicitud SOAP

```



```

        headers = {
            'Content-Type': 'text/xml; charset=utf-8',
            'SOAPAction':
'http://ws.sdde.bccr.fi.cr/ObtenerIndicadoresEconomicos'
        }
        # Realizar la solicitud POST
        response = requests.post(url, data=soap_request, headers=headers)
        # Parsear el XML de la respuesta
        root = ET.fromstring(response.content)
        # Extraer los datos de interés
        data = []
        for item in root.findall(".//INGC011_CAT_INDICADORECONOMIC"):
            cod_ind = item.find("COD_INDICADORINTERNO").text
            fecha = item.find("DES_FECHA").text.replace("T00:00:00-06:00", "")
# Limpiar la fecha
            valor = float(item.find("NUM_VALOR").text) # Convertir a flotante
            # Agregar a la lista de datos
            data.append({
                "CÓDIGO_IND": cod_ind,
                "FECHA": fecha,
                "VALOR": valor,
                "INDICADOR": variable
            })
        # Crear el DataFrame
        df = pd.DataFrame(data)
        # Concatenar los DataFrames
        df_Ind = pd.concat([df_Ind, df])
        # Formatear dataframe
        df_pivot = df_Ind.pivot(index="FECHA", columns="INDICADOR",
values="VALOR").reset_index()
        # Reordenar las columnas según el diccionario
        columnas = ['FECHA'] + [col for col in self.dicc.keys() if col in
df_pivot.columns]
        df_pivot = df_pivot[columnas]
        df_Ind_s = df_pivot.copy()
        columns = " - ".join(df_Ind_s.columns[2:].values.tolist())
        # Imprimir detalles
        df_report = df_Ind_s.dropna(subset=df_Ind_s.columns[1])
        report = f"## Datos del indicador {df_Ind_s.columns[1]}"
        report += f"\n\n**{df_Ind_s.columns[1]} al Cierre:** {df_report.iloc[-1,
1]:.2f}"
        report += f"\n\n**Fecha de último dato disponible:**
{df_report['FECHA'].iloc[-1]}"
        report += "\n\n**Período seleccionado:**"
        report += f"\n\n**Fecha inicial:** {self.start_date}"
        report += f"\n\n**Fecha final:** {self.end_date}"
        report += f"\n\n**Cantidad de registros base:** {len(df_report)}"
        report += f"\n\n**Datos Importantes de la Proyección:**"
        report += f"\n\n**Cantidad de Predictores adicionales:**
{len(df_Ind_s.columns[2:])}"

```

```

        report += f"\n\n**Predictores adicionales:** {columns}"
        return report, df_Ind_s

# Definir el agente para graficar los datos
class ChartAgentM(CodeAgent):
    def __init__(self, df_Ind, forecast):
        self.df = df_Ind
        self.forecast = forecast

    def run(self):
        # Convertir la columna FECHA a datetime
        self.df['FECHA'] = pd.to_datetime(self.df['FECHA'])
        # Crear la gráfica
        fig, ax = plt.subplots(figsize=(10, 6))
        df_chart = self.df.dropna(subset=self.df.columns[1])
        ax.plot(df_chart["FECHA"], df_chart[df_chart.columns[1]], color="green",
label=f"{df_chart.columns[1]}")
        ax.plot(self.forecast["ds"], self.forecast["yhat"], color="red",
linestyle="--", label="Predicción")
        ax.set_title(f"Indicador {df_chart.columns[1]}")
        ax.set_xlabel("Fecha")
        ax.set_ylabel("Valor")
        ax.xaxis.set_major_formatter(mdates.DateFormatter('%m-%Y'))
        ax.xaxis.set_major_locator(plt.MaxNLocator(10))
        plt.xticks(rotation=45)
        plt.legend()
        return fig

# Definir el agente para proyectar los datos
class PredictionAgentM(CodeAgent):
    def __init__(self, df_Ind, Periodos, dicc, freq, end_date):
        self.df = df_Ind
        self.Periodos = Periodos
        self.dicc = dicc
        self.freq = freq
        self.end_date = end_date

    def run(self):
        # Crear dataframe de proyecciones
        df_proy = pd.DataFrame()
        # Crear modelos para cada indicador
        for col in self.df.columns[2:]:
            # Crear el modelo Prophet
            model = Prophet()
            # Mantener dataframe sin NA's
            df_n = self.df[["FECHA", col]].dropna()
            # Ajustar el modelo
            model.fit(df_n.rename(columns={"FECHA": "ds", col: "y"}))
            # Crear un DataFrame con las fechas a predecir
            if self.dicc[col][1] == 'D':

```

```

        future = model.make_future_dataframe(
            periods=int(self.Periodos * (1 if self.freq == 'D' else 30 if
self.freq == 'M' else 90)), freq='D')
        elif self.dicc[col][1] == 'M':
            future = model.make_future_dataframe(
                periods=int(self.Periodos * (1 if self.freq == 'M' else 1/30 if
self.freq == 'D' else 3)), freq='M')
        elif self.dicc[col][1] == 'Q':
            future = model.make_future_dataframe(
                periods=int(self.Periodos * (1 if self.freq == 'Q' else 1/90 if
self.freq == 'D' else 1/3)), freq='Q')
        # Realizar la proyección
        forecast = model.predict(future)
        # Agregar a dataframe
        df_proy_1 = pd.DataFrame()
        df_proy_1[f"{col}_proy"] = forecast["yhat"]
        df_proy = pd.concat([df_proy, df_proy_1], axis=1)

# Formatear dataframe de acuerdo con peridicidad
df_or = self.df.copy()
if self.dicc[df_or.columns[1]][1] == 'D':
    # Completar NA's
    df_or.fillna(method='ffill', inplace=True)
    # Borrar NA's restantes
    df_or.dropna(inplace=True)
elif self.dicc[df_or.columns[1]][1] == 'M':
    # Borrar NA's de la segunda columna
    df_or.dropna(subset=[list(self.dicc.keys())[0]], inplace=True)
    # Completar NA's
    df_or.fillna(method='ffill', inplace=True)
    # Borrar NA's restantes
    df_or.dropna(inplace=True)
elif self.dicc[df_or.columns[1]][1] == 'Q':
    # Borrar NA's
    df_or.dropna(inplace=True)

#Obtener última fecha
last_date = pd.to_datetime(df_or["FECHA"].iloc[-1])
# Cambiar nombres dataframe
df_or.columns = ['ds', 'y'] + [col for col in df_or.columns[2:]]
# Crear el modelo Prophet
model = Prophet()
# Agregar regresores
for col in df_or.columns[2:]:
    model.add_regressor(col)
# Ajustar el modelo
model.fit(df_or)
# Crear un DataFrame con las fechas a predecir
future = model.make_future_dataframe(periods=self.Periodos, freq=self.freq)
df_proy_2 = pd.DataFrame()

```

```

        for col in df_or.columns[2:]:
            if self.freq == 'D': # TODO: Listo
                if self.dicc[col][1] == 'D':
                    future[col] = pd.concat([df_or[col],
df_proy[f"{col}_proy"][-self.Periodos:]], ignore_index=True)
                elif self.dicc[col][1] == 'M':
                    factor = 30
                    df_proy_3 = df_proy.copy()
                    df_proy_3.dropna(inplace=True) # !Cambio fundamental
                    df_proy_2[f"{col}_proy"] =
np.repeat(df_proy_3[f"{col}_proy"].values, factor)
                    future[col] = pd.concat([df_or[col],
df_proy_2[f"{col}_proy"][-self.Periodos:]], ignore_index=True)
                    df_proy_2 = pd.DataFrame()
                elif self.dicc[col][1] == 'Q':
                    factor = 90
                    df_proy_3 = df_proy.copy()
                    df_proy_3.dropna(inplace=True) # !Cambio fundamental
                    df_proy_2[f"{col}_proy"] =
np.repeat(df_proy_3[f"{col}_proy"].values, factor)
                    future[col] = pd.concat([df_or[col],
df_proy_2[f"{col}_proy"][-self.Periodos:]], ignore_index=True)
                    df_proy_2 = pd.DataFrame()
            elif self.freq == 'M': # TODO: Listo
                if self.dicc[col][1] == 'M':
                    df_proy_3 = df_proy.copy()
                    df_proy_3.dropna(inplace=True) # !Cambio fundamental
                    future[col] = pd.concat([df_or[col],
df_proy_3[f"{col}_proy"][-self.Periodos:]], ignore_index=True)
                elif self.dicc[col][1] == 'D':
                    factor = self.Periodos*30
                    df_proy_2[f"{col}_proy"] =
df_proy[f"{col}_proy"][-factor:].values
                    dates_next_months = [(last_date +
pd.DateOffset(months=i)).to_period("M").end_time.normalize() for i in range(1,
self.Periodos+1)]
                    star_date = pd.to_datetime(self.end_date)
                    dates_next = pd.date_range(start=star_date +
pd.Timedelta(days=1), periods=factor, freq='D')
                    df_proy_2['Fecha'] = dates_next
                    df_proy_2 =
df_proy_2[df_proy_2['Fecha'].isin(dates_next_months)]
                    future[col] = pd.concat([df_or[col], df_proy_2[f"{col}_proy"]],
ignore_index=True)
                df_proy_2 = pd.DataFrame()
            elif self.dicc[col][1] == 'Q':
                factor = 3
                df_proy_3 = df_proy.copy()
                df_proy_3.dropna(inplace=True) # !Cambio fundamental
                df_proy_2[f"{col}_proy"] =

```

```

np.repeat(df_proy_3[f"{col}_proy"].values, factor)
        future[col] = pd.concat([df_or[col],
df_proy_2[f"{col}_proy"][-self.Periodos:]], ignore_index=True)
        df_proy_2 = pd.DataFrame()
        elif self.freq == 'Q':# TODO: Listo
            dates_next_quarters = [(last_date +
pd.DateOffset(months=i*3)).to_period("Q").end_time.normalize() for i in range(1,
self.Periodos+1)]
            if self.dicc[col][1] == 'Q':
                df_proy_3 = df_proy.copy()
                df_proy_3.dropna(inplace=True) # !Cambio fundamental
                future[col] = pd.concat([df_or[col],
df_proy_3[f"{col}_proy"][-self.Periodos:]], ignore_index=True)
            elif self.dicc[col][1] == 'D':
                factor = self.Periodos*90
                df_proy_2[f"{col}_proy"] =
df_proy[f"{col}_proy"][-factor:].values
                star_date = pd.to_datetime(self.end_date)
                dates_next = pd.date_range(start=star_date +
pd.Timedelta(days=1), periods=factor, freq='D')
                df_proy_2['Fecha'] = dates_next
                df_proy_2 =
df_proy_2[df_proy_2['Fecha'].isin(dates_next_quarters)]
                future[col] = pd.concat([df_or[col], df_proy_2[f"{col}_proy"]],
ignore_index=True)
                df_proy_2 = pd.DataFrame()
            elif self.dicc[col][1] == 'M':
                factor = self.Periodos*3
                df_proy_3 = df_proy.copy()
                df_proy_3.dropna(inplace=True) # !Cambio fundamental
                df_proy_2[f"{col}_proy"] =
df_proy_3[f"{col}_proy"][-factor:].values
                star_date = pd.to_datetime(self.end_date)
                dates_next = pd.date_range(start=star_date +
pd.Timedelta(days=1), periods=factor, freq='M')
                df_proy_2['Fecha'] = dates_next
                df_proy_2 =
df_proy_2[df_proy_2['Fecha'].isin(dates_next_quarters)]
                future[col] = pd.concat([df_or[col], df_proy_2[f"{col}_proy"]],
ignore_index=True)
                df_proy_2 = pd.DataFrame()
            # Realizar la proyección
            future = future.dropna()
            forecast = model.predict(future)
            # Retornar proyecciones
            return forecast[['ds','yhat']]

# Definir el multiagente
class MultiAgentM(ToolCallingAgent):
    def __init__(self, dicc, start_date, end_date, name, email, token, Periodos,

```

```

freq, api_key):
    self.dicc = dicc
    self.start_date = start_date
    self.end_date = end_date
    self.name = name
    self.email = email
    self.token = token
    self.Periodos = Periodos
    self.freq = freq
    self.api_key = api_key

    def run(self):
        # Crear el agente para obtener los datos
        agent1 = IndicatorDataM(self.dicc, self.start_date, self.end_date,
self.name, self.email, self.token)
        report, df_Ind_2 = agent1.run()
        # Crear el agente para predecir los datos
        agent3 = PredictionAgentM(df_Ind_2, self.Periodos, self.dicc, self.freq,
self.end_date)
        forecast = agent3.run()
        # Crear el agente para graficar los datos
        agent2 = ChartAgentM(df_Ind_2, forecast)
        fig = agent2.run()
        # Crear el agente para obtener noticias
        agent4 = NewsAgent(df_Ind_2.columns[1], self.end_date, forecast,
self.api_key)
        resume = agent4.run()
        # Crear el reporte final
        report += f"\n\n**Fecha final de proyección:**"
{forecast['ds'].iloc[-1].strftime('%Y/%m/%d')}
        report += f"\n\n**Predicción del indicador:**"
{forecast['yhat'].iloc[-1].round(2)}
        report += "\n\n## Gráfica del indicador"
        return report, fig, resume

```

## 8.3 Código archivo App.py

```
# Importar Librerías
import streamlit as st
from PIL import Image
import io
import base64
import markdown2
from datetime import datetime
from funciones import MultiAgent, MultiAgentM
import warnings

warnings.filterwarnings("ignore")

# Importar imagen
imagen_1 = Image.open("Portada-Agent.png")

# Configuración de la página

def run():

    # Configurar sidebar
    st.sidebar.markdown("<h7 style='color:#097da6;'>Ingresa el API Key de GROQ para  
generar el análisis:</h7>", unsafe_allow_html=True)
    api_key = st.sidebar.text_input("API Key", value="",
    label_visibility="collapsed")
    st.sidebar.markdown("<h7 style='color:#097da6; text-align: justify;'>Si no  
sabes como generar el API Key, en este  
[Tutorial](https://www.youtube.com/watch?v=YPghgcC4p-E) se explica muy fácilmente.  
Este servicio de GROQ es totalmente gratuito.</h7>", unsafe_allow_html=True)

    # Cambiar el color de fondo de la página
    page_bg_img = ""
    <style>
    [data-testid="stAppViewContainer"] {
        background:
            radial-gradient(circle at 20% 20%, rgba(255, 255, 255, 0.5), transparent),
            radial-gradient(circle at 80% 80%, rgba(255, 255, 255, 0.3), transparent),
            linear-gradient(to bottom, #2596BE, #f5e6d3);
    }

    [data-testid="stHeader"] {
        background: rgba(0, 0, 0, 0);
    }
    </style>
    ""

    st.markdown(page_bg_img, unsafe_allow_html=True)

    st.image(imagen_1, use_container_width='always') #imagen de portada
```

```

    st.markdown("<h4 style='color:#097da6; text-align: justify;*>Soy un Agente  

creado para ayudarte a analizar variables económicas.</h4>",  

unsafe_allow_html=True)

    st.markdown("<h5 style='color:#097da6;*>¿Cuál variable te interesa analizar?  

</h5>", unsafe_allow_html=True)

    # Crear un menú de selección

    variable = st.selectbox("", ["TC de Compra", "TC de Venta", "Inflación", "PIB",  

"Tasa Desempleo", "IMAE", "TBP", "TRI 6M"],  

index=None, placeholder="Selecciona una opción",  

label_visibility="collapsed")

    dicc_indicador = {"TC de Compra": ["317", "D"], "TC de Venta": ["318", "D"],  

"Inflación": ["89635", "M"], "PIB": ["90633", "Q"],  

"Tasa Desempleo": ["23630", "M"], "IMAE": ["87764", "M"],  

"TBP": ["423", "D"], "TRI 6M": ["41206", "D"]}

    if variable != None:  

        freq = dicc_indicador[variable][1]  

        if freq == "D":  

            periodicidad = "Diaria"  

            value = 180  

        elif freq == "M":  

            periodicidad = "Mensual"  

            value = 6  

        elif freq == "Q":  

            periodicidad = "Trimestral"  

            value = 2  

        else:  

            periodicidad = "Anual"  

            value = 1

    st.markdown(f"<h7 style='color:#097da6;*>***Periodicidad**:  

{periodicidad}</h7>", unsafe_allow_html=True)

    st.markdown("<h5 style='color:#097da6;*>Selecciona el tipo de  

análisis:</h5>", unsafe_allow_html=True)

    tipo_analisis = st.selectbox("", ["Autorregresivo", "Regresión Múltiple"],  

index=None, placeholder="Selecciona una  

opción", label_visibility="collapsed")

    # TODO: Análisis Autoregresivo  

    if tipo_analisis == "Autorregresivo":  

        st.markdown("<h7 style='color:#097da6;*>Completa lo siguiente para  

obtener los datos del [BCCR Servicio  

Web](https://www.bCCR.fi.cr/indicadores-economicos/servicio-web)</h7>",  

unsafe_allow_html=True)

```



```

Name = st.text_input("**Nombre:**", "")
Email = st.text_input("**Correo Electrónico:**", "")
Token = st.text_input("**Token BCCR:**", "")
Indicador = dicc_indicador[variable][0]

col1, col2 = st.columns(2, gap="small")
with col1:
    fecha_inicio = st.date_input("**Fecha Inicial:**", datetime(2020,
1, 1))
    with col2:
        fecha_fin = st.date_input("**Fecha Final:**",
datetime.today().date())

        fecha_inicio = fecha_inicio.strftime("%Y/%m/%d")
        fecha_fin = fecha_fin.strftime("%Y/%m/%d")

        periodos = st.number_input("**Número de periodos a predecir:**",
min_value=1, max_value=180, value=value)

        if st.button("Generar Reporte", type="primary"):
            # Crear el multiagente
            agent = MultiAgent(variable, Indicador, fecha_inicio, fecha_fin,
Name, Email, Token, periodos, freq, api_key)
            report, fig, resume = agent.run()

            # Mostrar el reporte
            st.markdown("<h5 style='color:#097da6;'>REPORTE</h5>",
unsafe_allow_html=True)
            st.markdown(report, unsafe_allow_html=True)
            st.pyplot(fig)
            st.markdown(resume, unsafe_allow_html=True)

            # Convertir la imagen a base64
            buffer = io.BytesIO()
            fig.savefig(buffer, format="png", bbox_inches="tight")
            buffer.seek(0)
            img_str = base64.b64encode(buffer.read()).decode('utf-8')

            # Crear el contenido HTML
            html_content = f"""
<!DOCTYPE html>
<html>
<head>
    <title>Reporte - {variable}</title>
    <style>
        body {{ font-family: Arial, sans-serif; margin: 40px; }}
        h1, h2, h3, h4, h5 {{ color: #097da6; }}
        .content {{ margin-bottom: 20px; }}
    </style>

```

```

</head>
<body>
  <h1>Reporte de Análisis: {variable}</h1>
  <div class="content">
    {markdown2.markdown(report)}
  </div>
  <div class="content">
    
  </div>
  <div class="content">
    {markdown2.markdown(resume)}
  </div>
</body>
</html>
"""

# Crear botón de descarga
st.download_button(
    label="Descargar Reporte",
    data=html_content,
    file_name=f"Reporte_{variable}.html",
    mime="text/html",
)

# TODO: Análisis Multivariable
elif tipo_analisis == "Regresión Múltiple":
    st.markdown("<h7 style='color:#097da6;'>Completa lo siguiente para
obtener los datos del [BCCR Servicio
Web](https://www.bCCR.fi.cr/indicadores-economicos/servicio-web)</h7>",
unsafe_allow_html=True)

    Name = st.text_input("**Nombre:**", "")
    Email = st.text_input("**Correo Electrónico:**", "")
    Token = st.text_input("**Token BCCR:**", "")
    Indicador = dicc_indicador[variable][0]

    col1, col2 = st.columns(2, gap="small")
    with col1:
        fecha_inicio = st.date_input("**Fecha Inicial:**", datetime(2020,
1, 1))
    with col2:
        fecha_fin = st.date_input("**Fecha Final:**",
datetime.today().date())

    fecha_inicio = fecha_inicio.strftime("%Y/%m/%d")
    fecha_fin = fecha_fin.strftime("%Y/%m/%d")

    regresores = st.multiselect("**Selecciona las variables
predictoras:**",

```

```

[var for var in ["TC de Compra", "TC de
Venta", "Inflación", "PIB", "Tasa Desempleo", "IMAE", "TBP", "TRI 6M"] if var !=
variable])

dicc_indicador = {key: value for key, value in dicc_indicador.items()
if key in regresores or key == variable}
dicc_indicador = {variable: dicc_indicador[variable]} | {key:
dicc_indicador[key] for key in regresores}

periodos = st.number_input("**Número de periodos a predecir:**",
min_value=1, max_value=180, value=value)

if st.button("Generar Reporte", type="primary"):
    # Crear el multiagente
    agent = MultiAgentM(dicc_indicador, fecha_inicio, fecha_fin, Name,
Email, Token, periodos, freq, api_key)
    # report, fig, resume = agent.run()
    report, fig, resume = agent.run()

    # Mostrar el reporte
    st.markdown("<h5 style='color:#097da6;'>REPORTE</h5>",
unsafe_allow_html=True)
    st.markdown(report, unsafe_allow_html=True)
    st.pyplot(fig)
    st.markdown(resume, unsafe_allow_html=True)

    # Convertir la imagen a base64
    buffer = io.BytesIO()
    fig.savefig(buffer, format="png", bbox_inches="tight")
    buffer.seek(0)
    img_str = base64.b64encode(buffer.read()).decode('utf-8')

    # Crear el contenido HTML
    html_content = f"""
<!DOCTYPE html>
<html>
<head>
    <title>Reporte - {variable}</title>
    <style>
        body {{ font-family: Arial, sans-serif; margin: 40px; }}
        h1, h2, h3, h4, h5 {{ color: #097da6; }}
        .content {{ margin-bottom: 20px; }}
    </style>
</head>
<body>
    <h1>Reporte de Análisis: {variable}</h1>
    <div class="content">
        {markdown2.markdown(report)}
    </div>
    <div class="content">

```

```

        
    </div>
    <div class="content">
        {markdown2.markdown(resume)}
    </div>
</body>
</html>
"""

# Crear botón de descarga
st.download_button(
    label="Descargar Reporte",
    data=html_content,
    file_name=f"Reporte_{variable}.html",
    mime="text/html",
)

if __name__ == "__main__":
    run()

```