

LABORATORIO

Solicitar a un modelo IBM
Granite que clasifique y
resuma datos

Contenido

Introducción	2
Requisitos de software	2
Objetivo	2
Pasos del laboratorio	2
Duración estimada para completarlo	3
Escenario	4
Antecedentes	4
Desafío	4
Solución.....	4
Paso 1: Crear una cuenta de GitHub	6
Descripción general	6
Instrucciones	6
Paso 2: Crear una cuenta de Replicate	11
Descripción general	11
Instrucciones	11
Paso 3: Registrarse en Google Colab	17
Descripción general	17
Instrucciones	17
Paso 4: Proporcionar reseñas de clientes para su clasificación y probar la solicitud inicial	29
Descripción general	29
Instrucciones	29
Paso 5: Refinar la solicitud para mejorar el resultado de la clasificación	32
Descripción general	32
Instrucciones	32
Paso 6: Proporcionar una transcripción de la reunión para su resumen y probar la solicitud inicial	42
Descripción general	42
Instrucciones	42
Paso 7: Refinar la solicitud para mejorar el resultado del resumen.....	46
Descripción general	46
Instrucciones	46
Conclusión	53

Introducción

En este laboratorio, asumirás la función de gerente de producto en una empresa de teléfonos inteligentes, donde tus responsabilidades fundamentales serán analizar las reseñas de los clientes y resumir las reuniones multifuncionales. Tus tareas incluyen clasificar las reseñas según el sentimiento (positivo, negativo o mixto) y etiquetar aspectos prioritarios como la duración de la batería, la calidad de la pantalla y el rendimiento. Además, resumirás las reuniones semanales en información estructurada y práctica para las partes interesadas.

Al utilizar los modelos de IBM Granite en Google Colab con Replicate, aprenderás a automatizar estas tareas, refinar las solicitudes para mejorar la precisión y la estructura, y ofrecer resultados claros y confiables que ahorren tiempo y mejoren la toma de decisiones.

Requisitos de software

Para completar este laboratorio, debes tener acceso a lo siguiente:

- **Cuenta de Replicate:** Crea una cuenta de Replicate y obtén un token de API.
- **Google Colab:** Crea una cuenta de Google para utilizar Colab para ejecutar el cuaderno.
- **Paquetes Python:** Instala los paquetes necesarios (como langchain_community, sistema operativo, replicate, google.colab).

Objetivo

Después de completar este laboratorio, deberás ser capaz de:

- Solicitar a un modelo IBM Granite que clasifique y resuma datos

Pasos del laboratorio

Este laboratorio requiere que completes los siguientes pasos:

- Paso 1: Crear una cuenta de GitHub
- Paso 2: Configurar tu entorno e inicializar el modelo IBM Granite
- Paso 3: Registrarse en Google Colab
- Paso 4: Proporcionar reseñas de clientes para su clasificación y probar la solicitud inicial
- Paso 5: Refinar la solicitud para mejorar el resultado de la clasificación

- Paso 6: Proporcionar una transcripción de la reunión para su resumen y probar la solicitud inicial
- Paso 7: Refinar la solicitud para mejorar el resultado del resumen

Duración estimada para completarlo

30 minutos

Escenario

Antecedentes

SmartTechX, un fabricante de teléfonos inteligentes de tamaño mediano, depende en gran medida de decisiones basadas en datos para mejorar sus productos estrella. La empresa genera 500 millones de dólares en ingresos anuales y emplea a 2000 personas; el conocimiento de los clientes y los resúmenes operativos desempeñan un papel fundamental en la configuración de sus estrategias.

Como gerente de producto, tus responsabilidades incluyen supervisar el análisis de comentarios sobre el producto y facilitar una comunicación clara entre equipos multifuncionales. Las siguientes son dos tareas principales de tu función:

- Analizar las reseñas de los clientes para clasificar el sentimiento (positivo, negativo o mixto) y etiquetar los aspectos prioritarios (duración de la batería, calidad de la pantalla o rendimiento)
- Resumir las reuniones multifuncionales semanales para brindar a las partes interesadas información útil para la toma de decisiones

Emily Carter, jefa de desarrollo de productos, confía en tu análisis para priorizar las actualizaciones para la próxima iteración del producto. Asimismo, Carlos Rivera, director de marketing, utiliza los resúmenes de reuniones para alinear las estrategias de marketing con los objetivos de la organización. Proporcionar información estructurada y confiable es fundamental para garantizar que SmartTechX cumpla sus objetivos estratégicos.

Desafío

Los procesos actuales de SmartTechX presentan dos desafíos fundamentales:

- **Clasificación de las reseñas de los clientes:** El enfoque manual para analizar y categorizar reseñas por sentimiento y etiquetar aspectos prioritarios es lento, inconsistente y propenso a errores. Con miles de revisiones que llegan mensualmente, las observaciones a menudo se demoran o se pasan por alto por completo.
- **Resumen de la reunión:** Resumir las reuniones multifuncionales semanales, que suelen durar 60 minutos, requiere mucho tiempo y atención a los detalles. Las partes interesadas exigen resúmenes que sean concisos, estructurados y centrados en información útil, como las decisiones tomadas, las prioridades y los plazos. El proceso manual, que toma más de tres horas cada semana, no deja tiempo para el trabajo estratégico.

Solución

SmartTechX quiere automatizar estos procesos con modelos IBM Granite. La solución implica utilizar **granite-3.0-8b-instruct**, un modelo optimizado para tareas de clasificación y resumen

de texto con matices. Al refinar las solicitudes y automatizar los flujos de trabajo, la empresa apunta a ofrecer resultados estructurados y útiles para ambas tareas.

- Para la **clasificación**, categorizarás las reseñas por sentimiento y etiquetarás los aspectos prioritarios, lo que garantizará información consistente y confiable para Emily Carter y su equipo.
- Para el **resumen**, condensarás las reuniones estratégicas semanales en resúmenes claros y prácticos que Carlos Rivera y otras partes interesadas puedan revisar fácilmente.

Los resultados esperados incluyen una mayor eficiencia, mayor precisión y una toma de decisiones más rápida en todos los departamentos. Como gerente de producto, implementarás y probarás estas soluciones en este laboratorio.

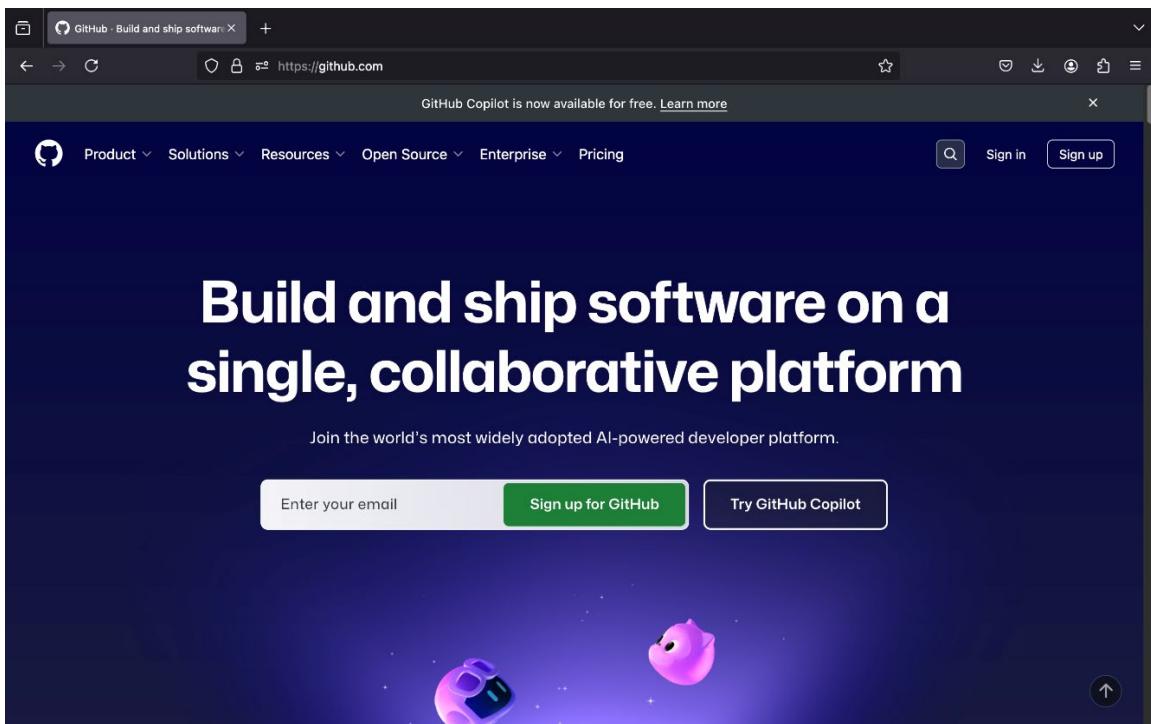
Paso 1: Crear una cuenta de GitHub

Descripción

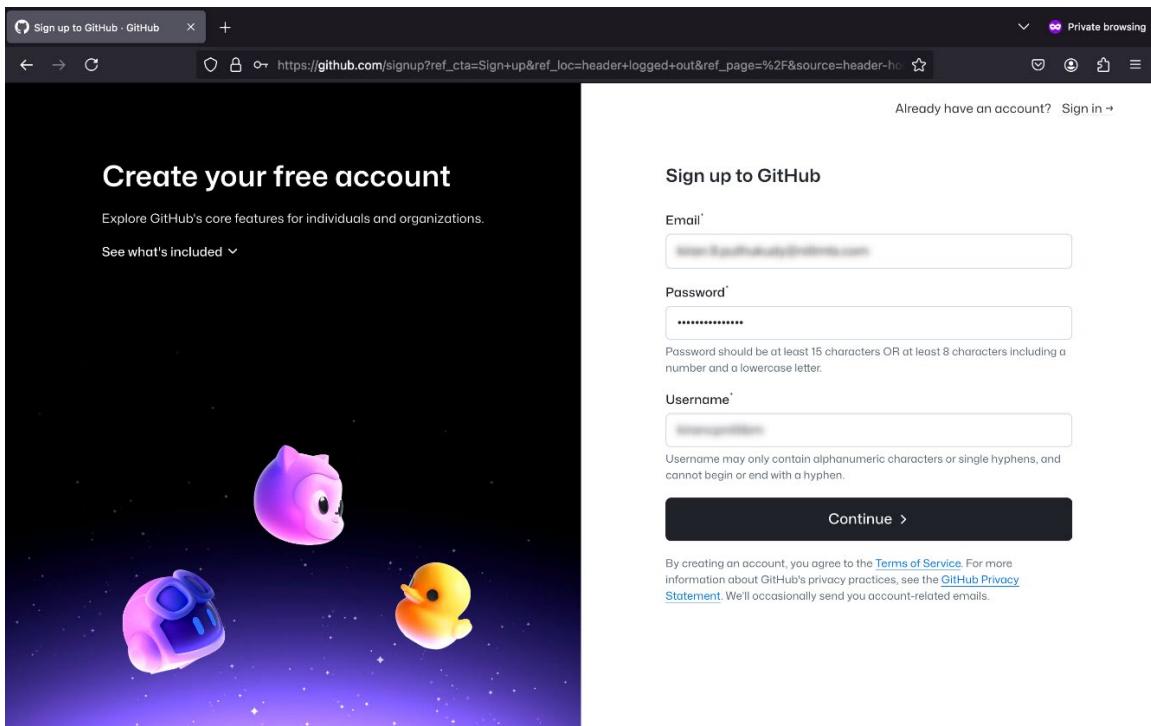
En este paso, crearás una cuenta de GitHub para registrarte en una cuenta de Replicate. GitHub es una plataforma que ayuda a los desarrolladores a almacenar, administrar y compartir código, al mismo tiempo que admite la colaboración a través de herramientas como control de versiones, seguimiento de errores y gestión de tareas. Esta configuración garantiza que tengas acceso al entorno de nube de Replicate necesario para completar el laboratorio de manera eficiente.

Instrucciones

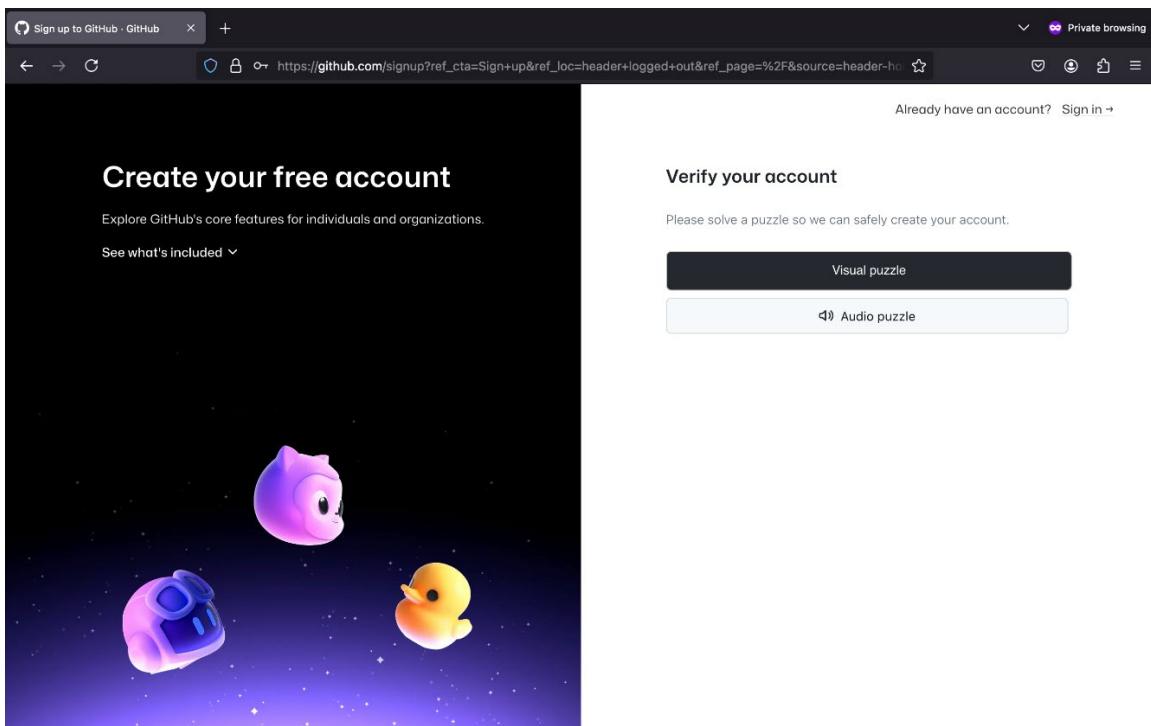
1. Ve al sitio web de [GitHub](https://github.com) para crear una cuenta de GitHub y selecciona **Registrarse**.



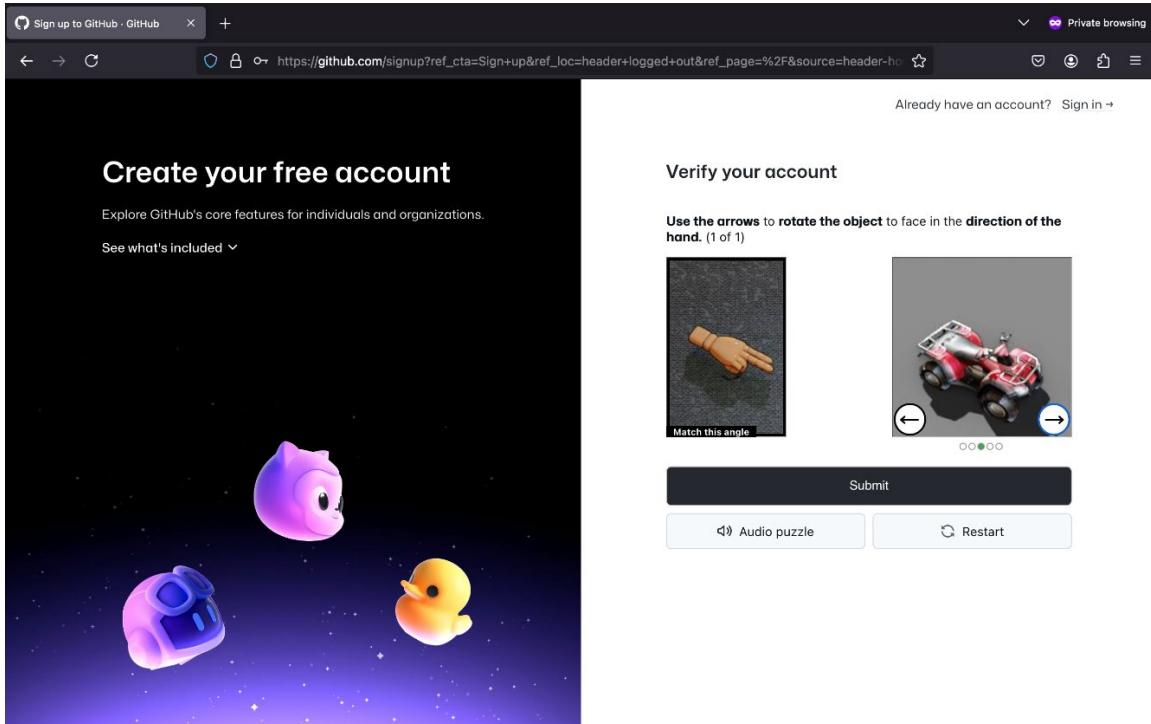
2. Escribe tus datos en los campos **Correo electrónico, Contraseña y Nombre de usuario**. Luego, selecciona **Continuar**.



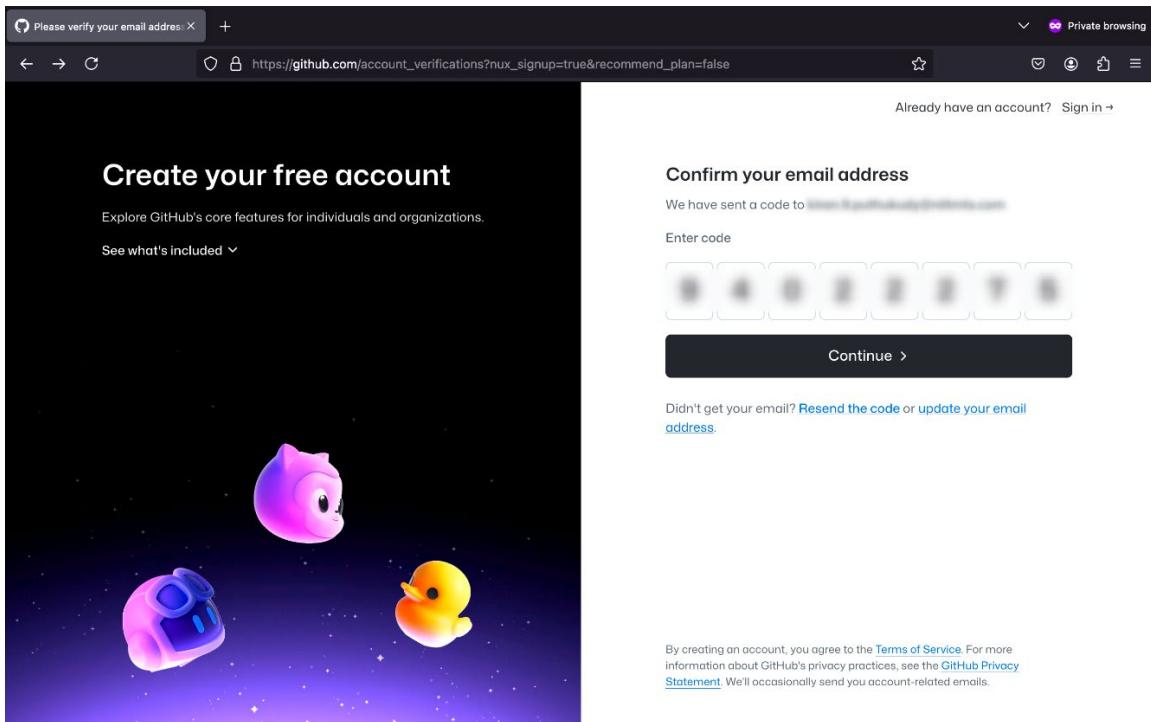
3. Selecciona la opción **Rompecabezas visual** para verificar tu cuenta.



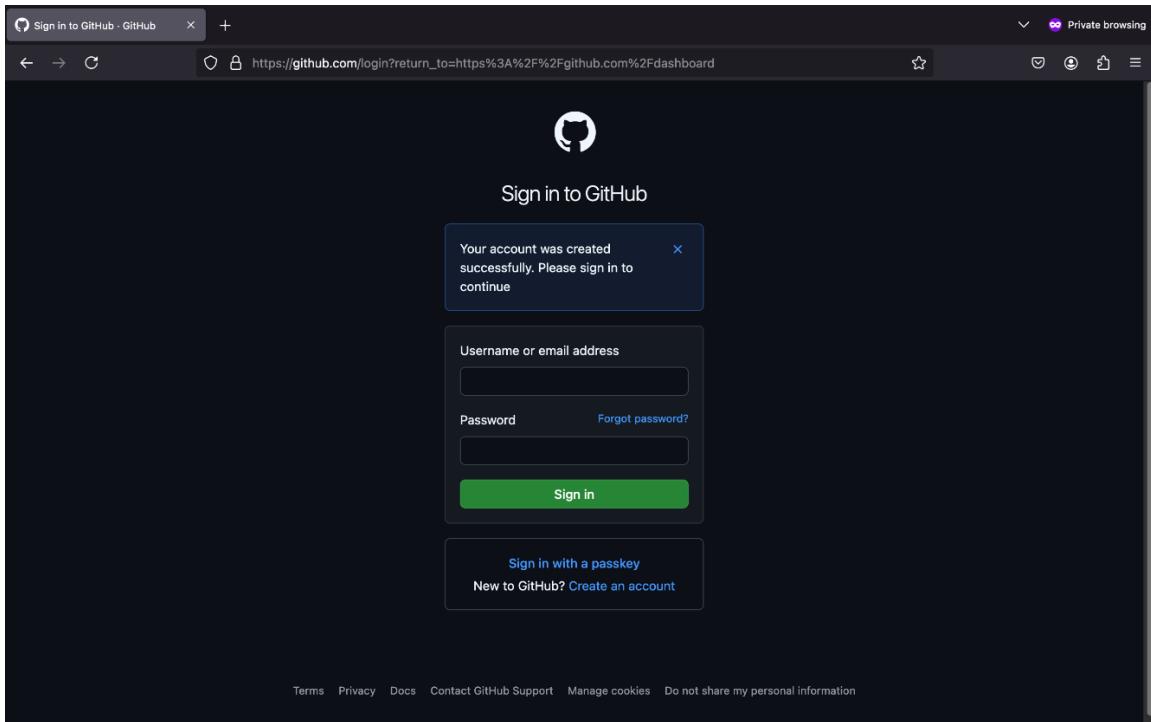
4. Resuelve el rompecabezas para verificar tu cuenta y selecciona **Comprobar**.



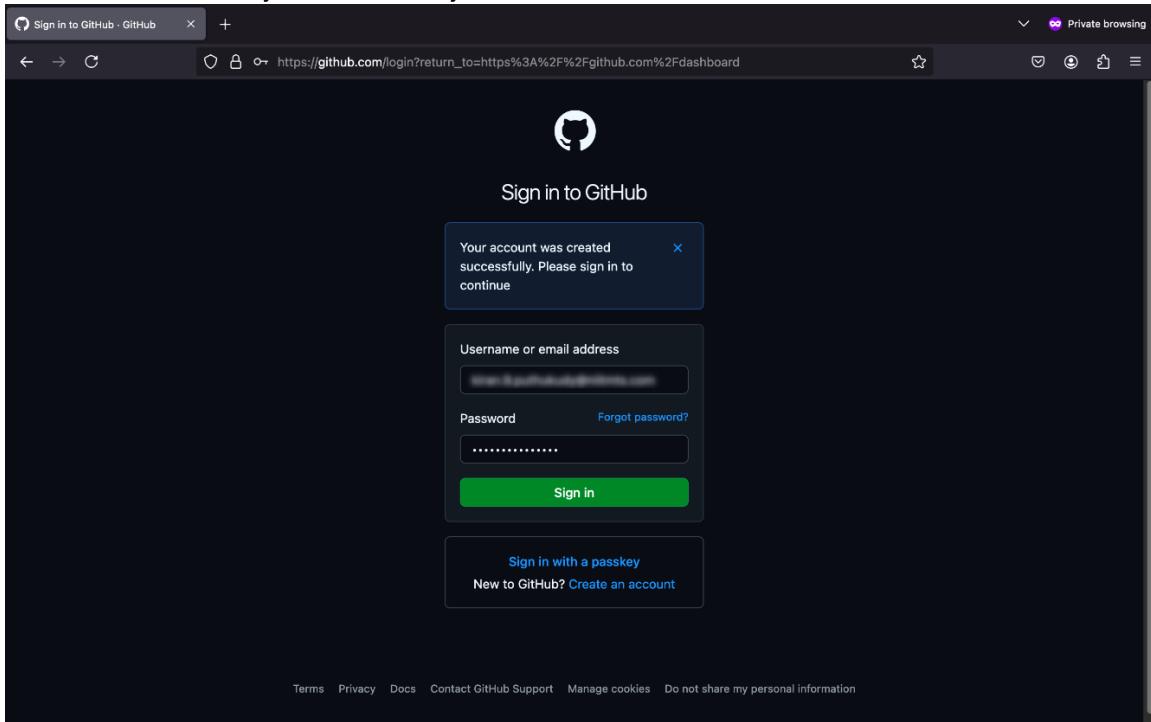
5. Escribe el código de confirmación en el campo **Ingresar código** para confirmar tu ID de correo electrónico y selecciona **Continuar**.



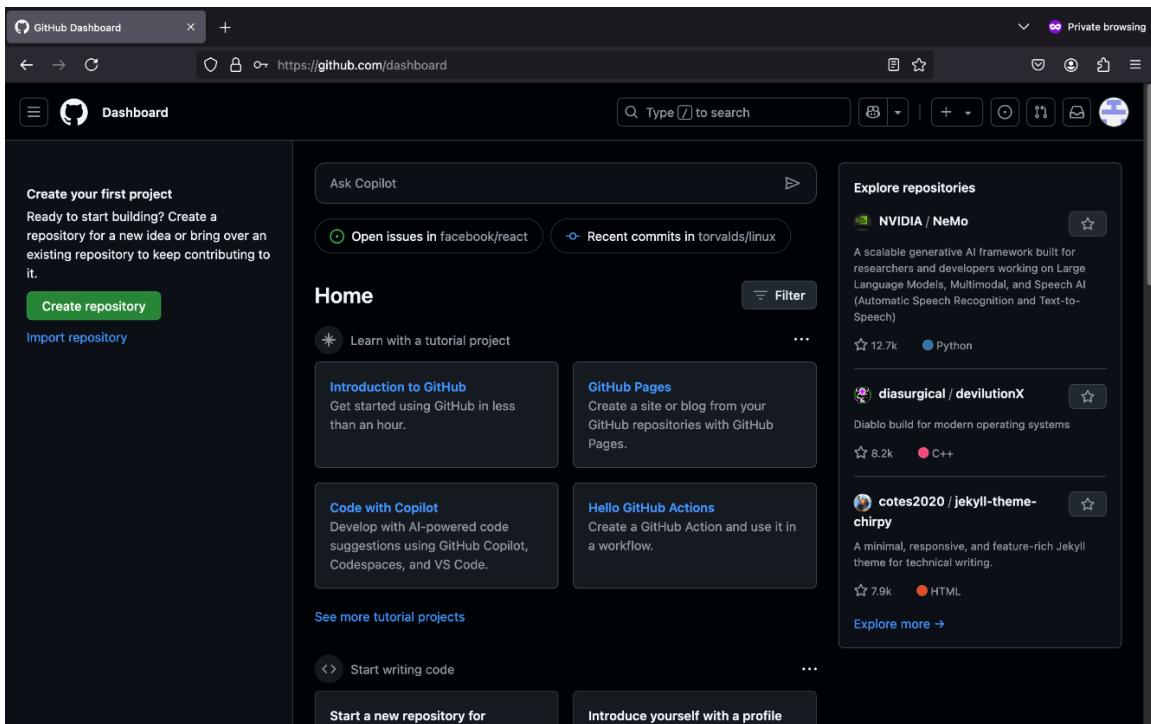
6. Verás un mensaje de confirmación cuando hayas completado con éxito la creación de tu cuenta de GitHub.



7. Escribe tus credenciales en los campos **Nombre de usuario o dirección de correo electrónico** y **Contraseña** y selecciona **Iniciar sesión**.



8. Se muestra el panel de GitHub para indicar que has iniciado sesión correctamente en tu cuenta de GitHub.



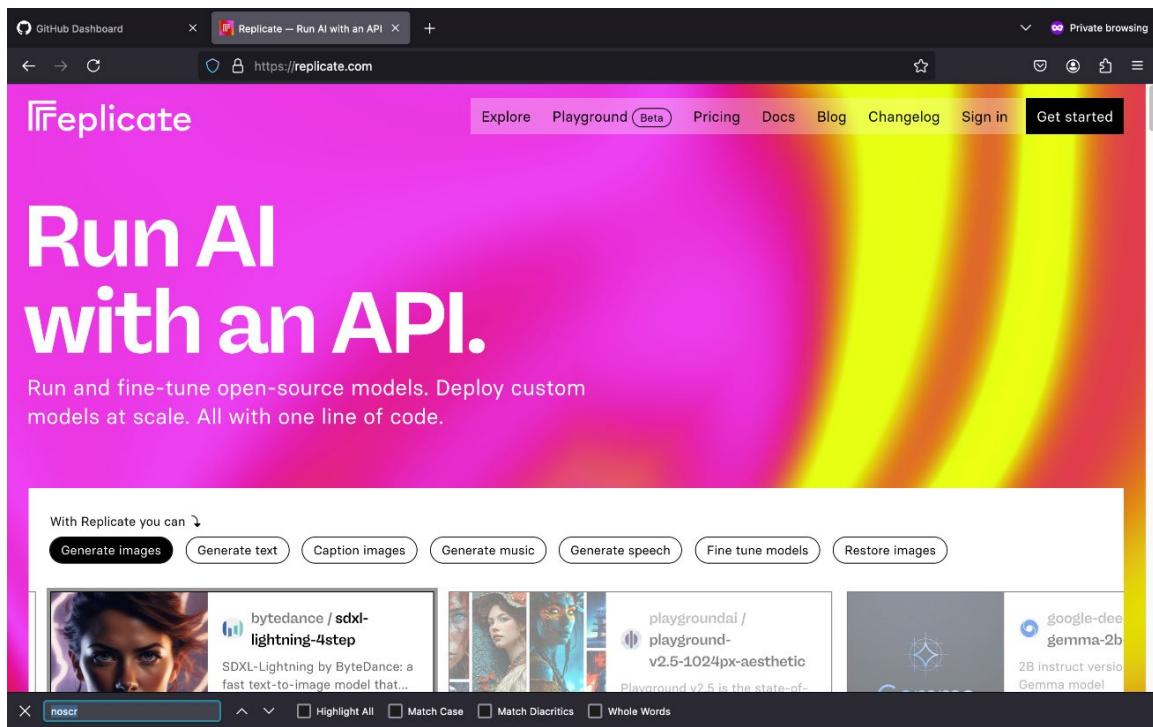
Paso 2: Crear una cuenta de Replicate

Descripción

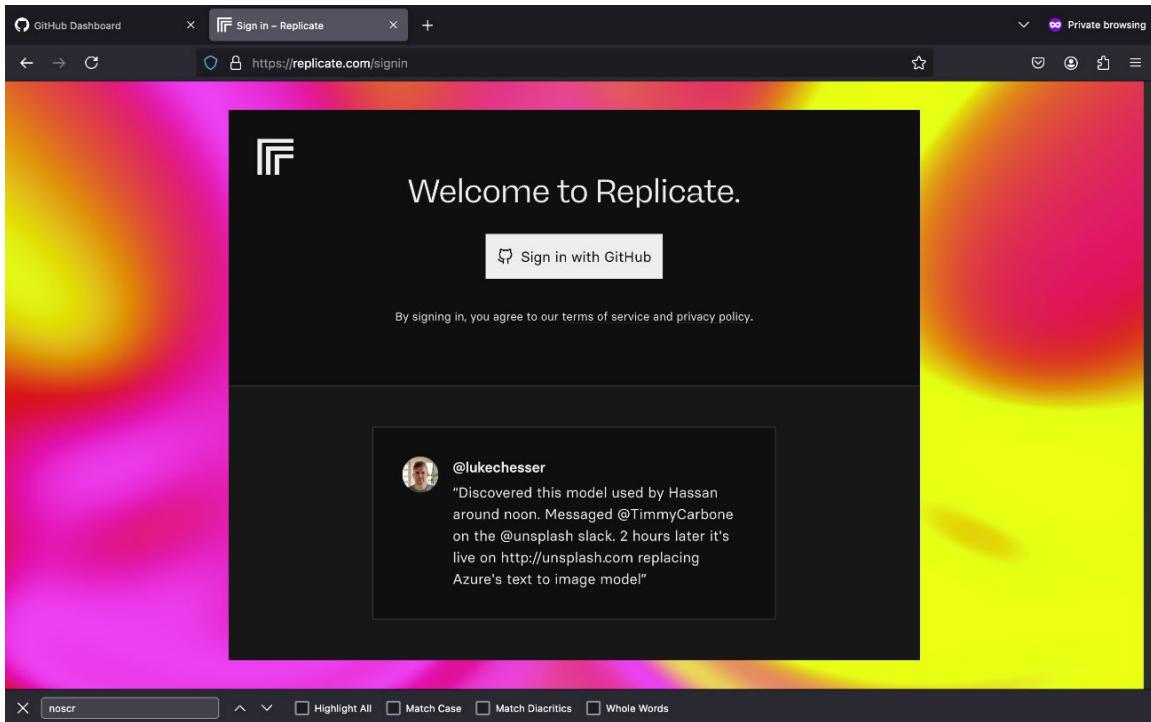
En este paso, usarás tu cuenta de GitHub para registrarte en una cuenta de Replicate. Replicate es una plataforma basada en la nube que te permite utilizar modelos de IA sin necesidad de hardware complejo. Como parte de este paso, crearás un token de Replicate. Un token es como una clave digital segura que permite al laboratorio conectarse a tu cuenta de Replicate y acceder a las herramientas necesarias para ejecutar el laboratorio en Google Colab.

Instrucciones

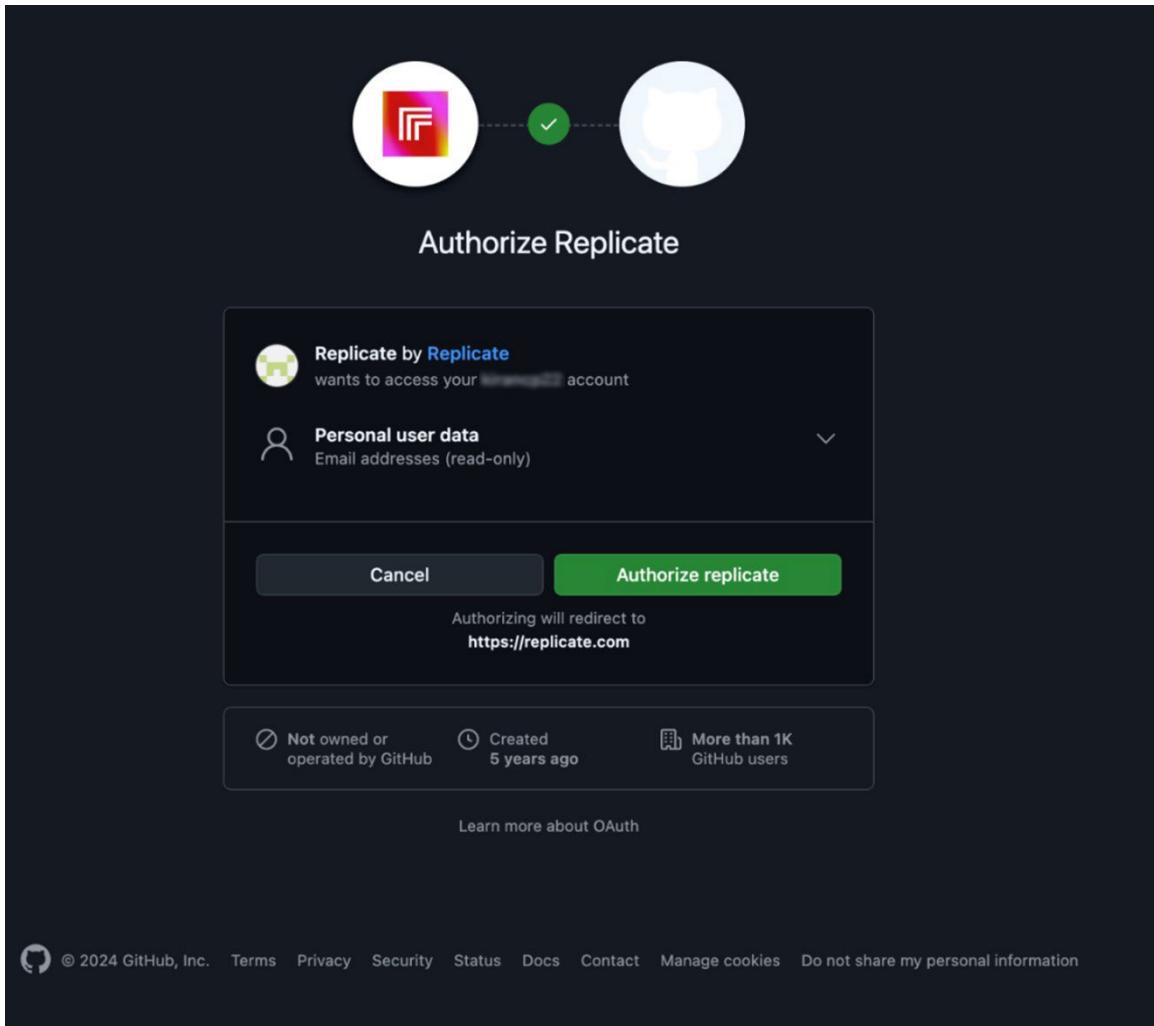
1. Ve al sitio web de [Replicate](https://replicate.com) y selecciona **Empezar**.



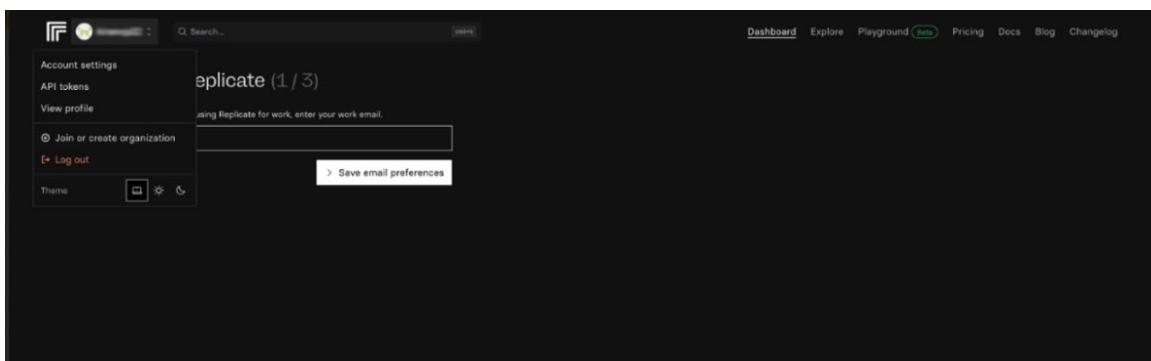
2. Selecciona **Iniciar sesión con GitHub** en la página Bienvenido a Replicate.



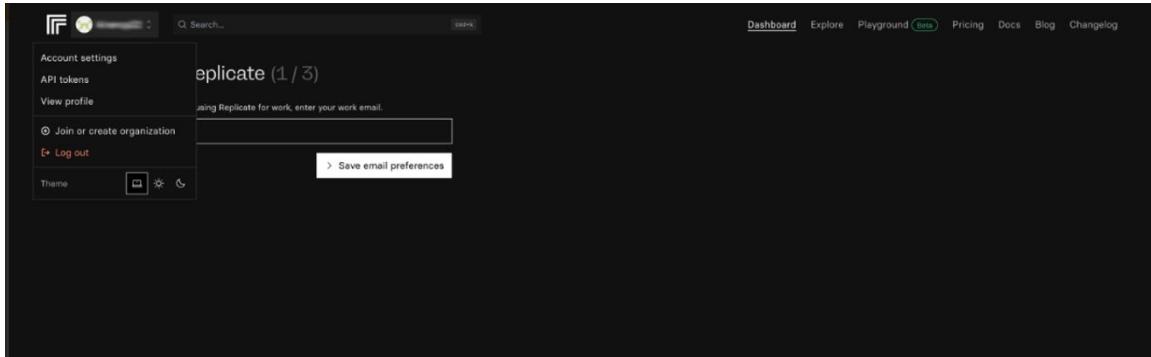
3. Selecciona **Autorizar Replicate** para continuar.



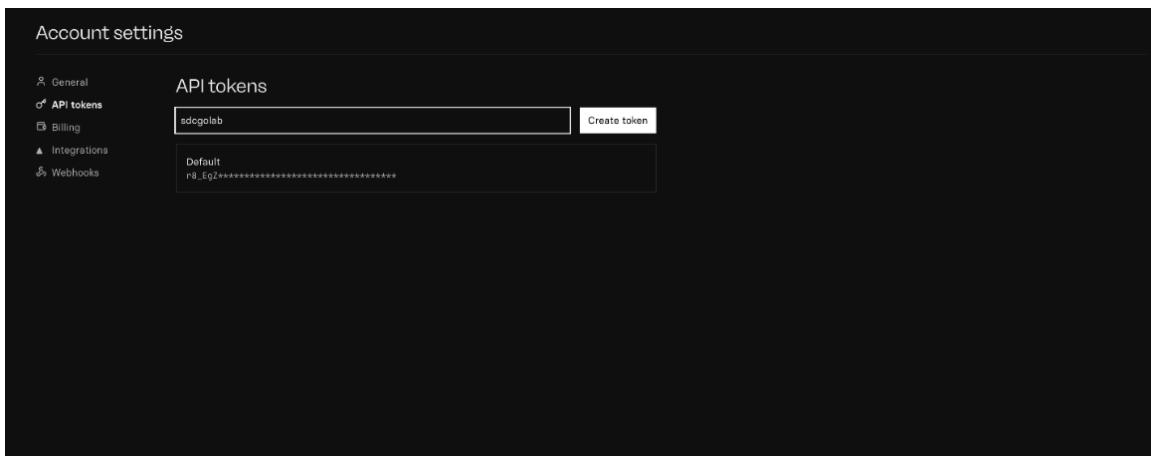
4. Para crear un token de Replicate, selecciona el ícono **configuración de la cuenta** en la barra de navegación.



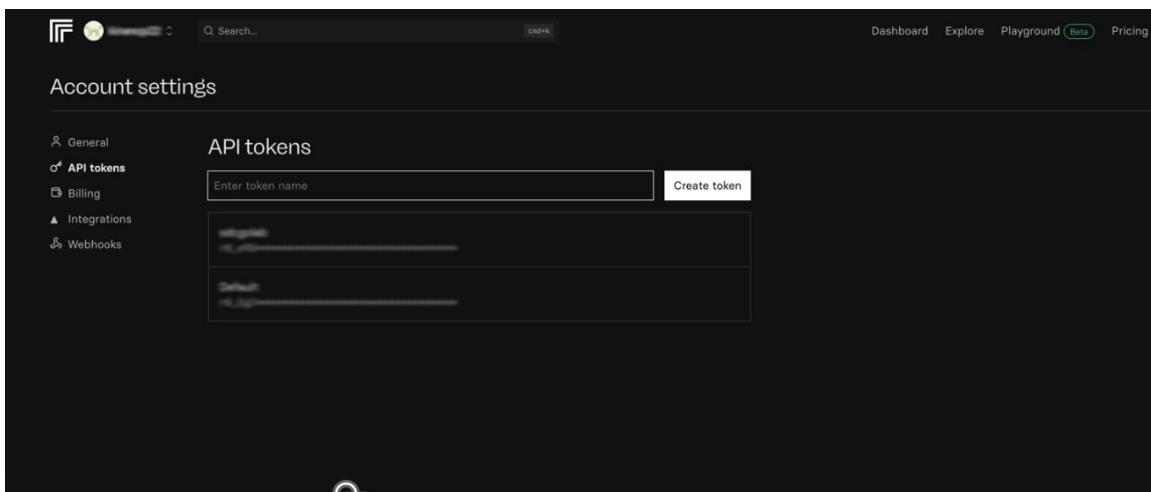
5. Selecciona **Tokens de API** en el menú "Configuración de la cuenta".



6. Escribe un nombre para el token en el campo **Tokens de API** y selecciona **Crear token**.

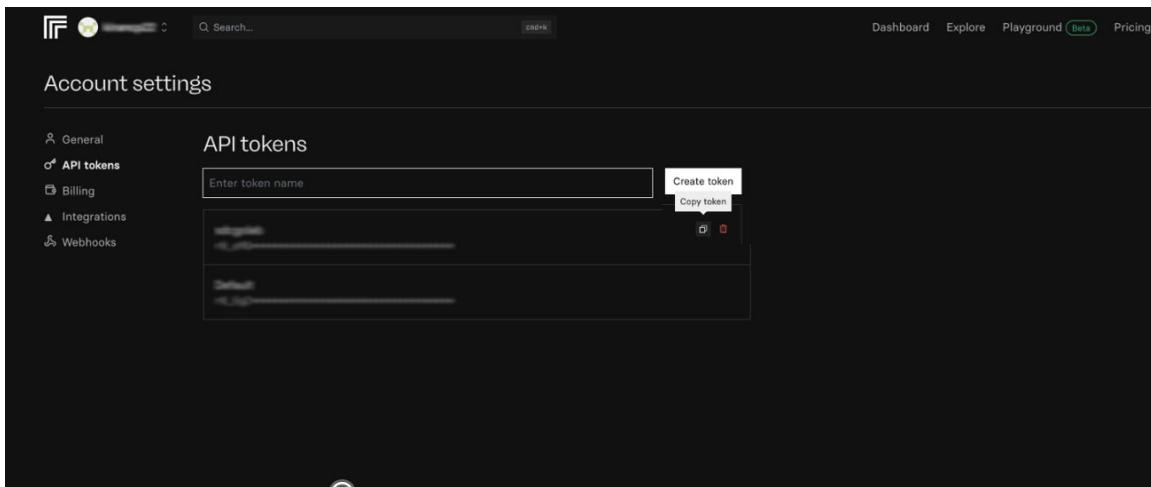


7. Tu token de API se muestra en la pantalla después del campo **Tokens de API**.

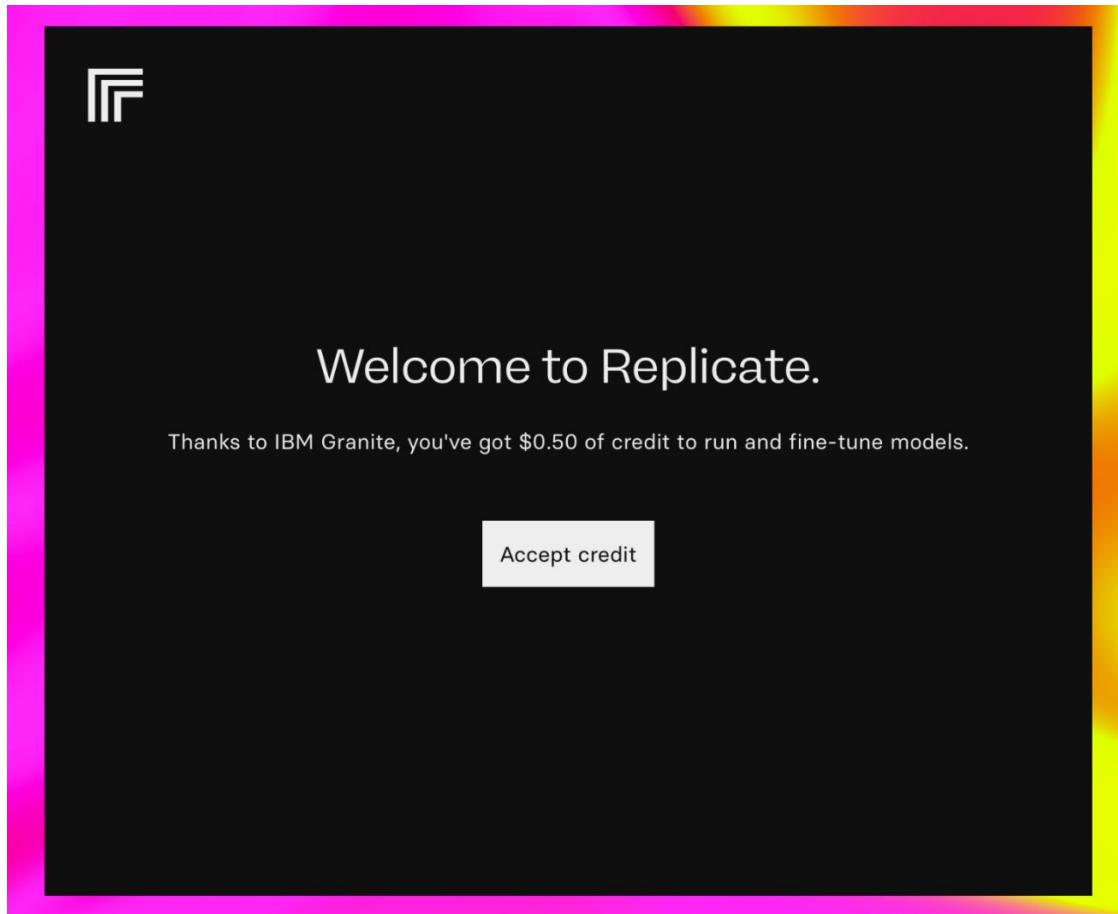


8. Selecciona el ícono **Copiar token** para copiar el token API de Replicate.

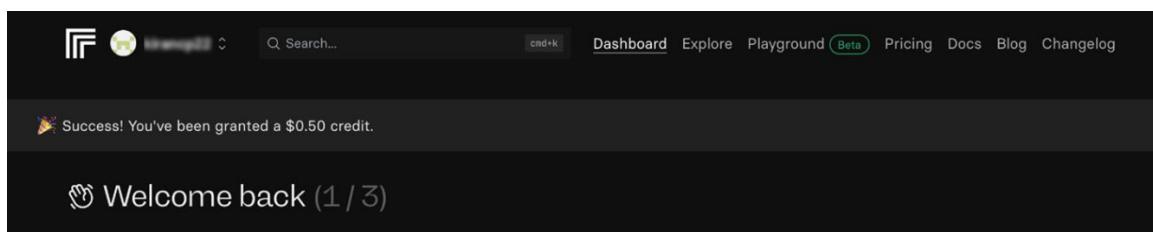
Nota: Guarda el token de Replicate porque necesitarás el token API de Replicate para ejecutar la instancia de Google Colab más adelante en este laboratorio.



- Ve al enlace [Invitar a Replicate](#) para reclamar 50 centavos en créditos Replicate, que usarás para ejecutar el laboratorio. Selecciona **Aceptar crédito** para reclamar los créditos.



- Verás un mensaje de confirmación indicando que se han agregado 50 centavos de crédito a tu cuenta de Replicate de la siguiente manera.



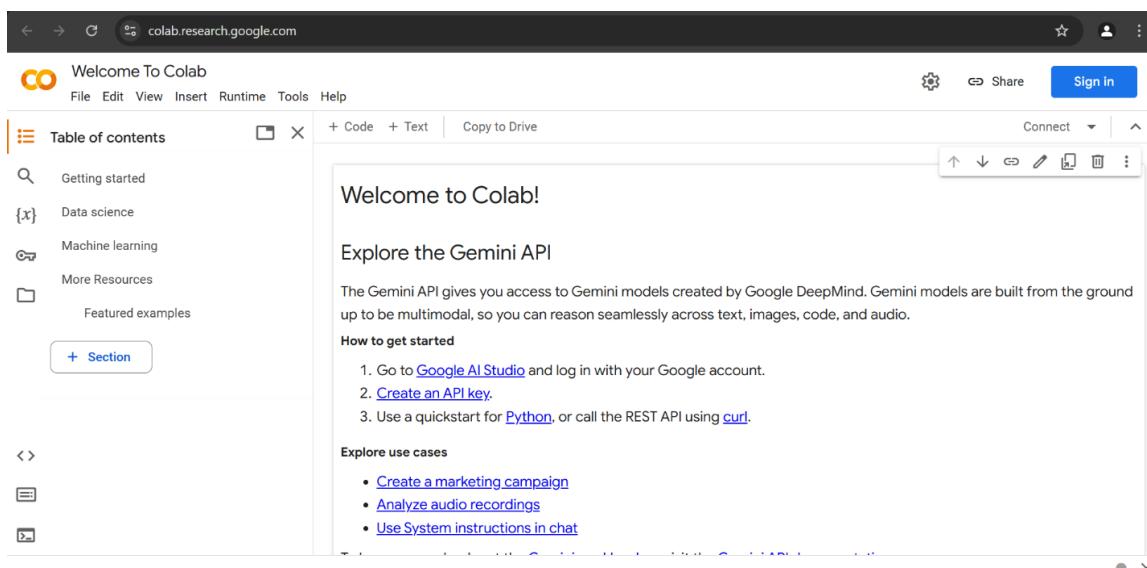
Paso 3: Registrarse en Google Colab

Descripción

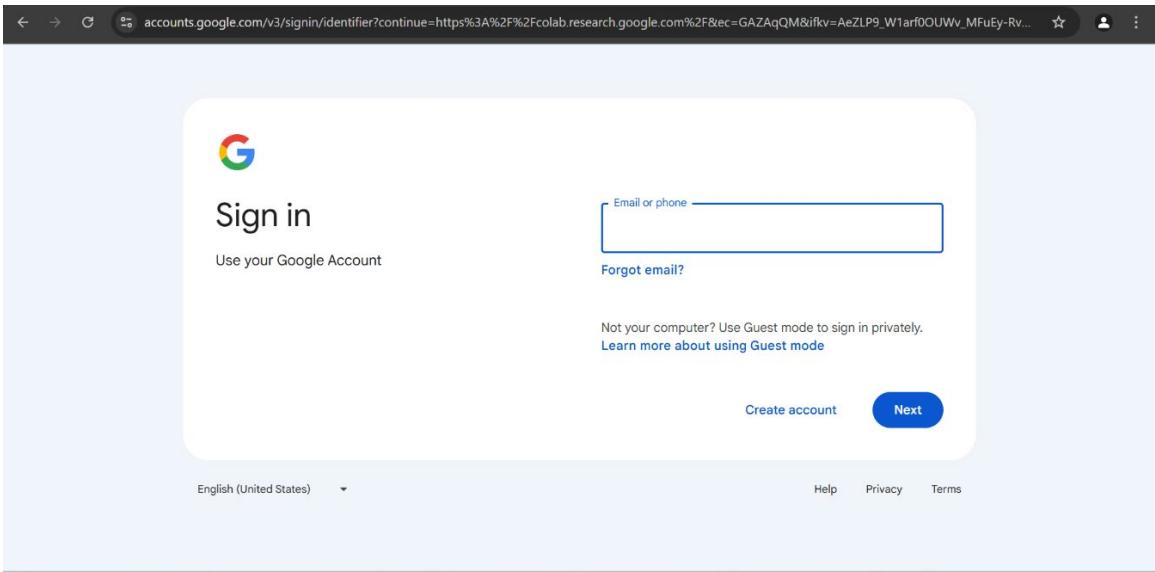
En este paso, configurarás una cuenta de Google Colab. Google Colab es una plataforma en la nube gratuita que te permite ejecutar código en cuadernos, que se utilizan normalmente para tareas de aprendizaje automático, ciencia de datos e inteligencia artificial. Esta cuenta te permite instalar y utilizar las herramientas necesarias para completar el laboratorio.

Instrucciones

1. Ve al sitio web de [Google Colab](#) para registrarte en una cuenta de Google Colab y selecciona **Iniciar sesión**.

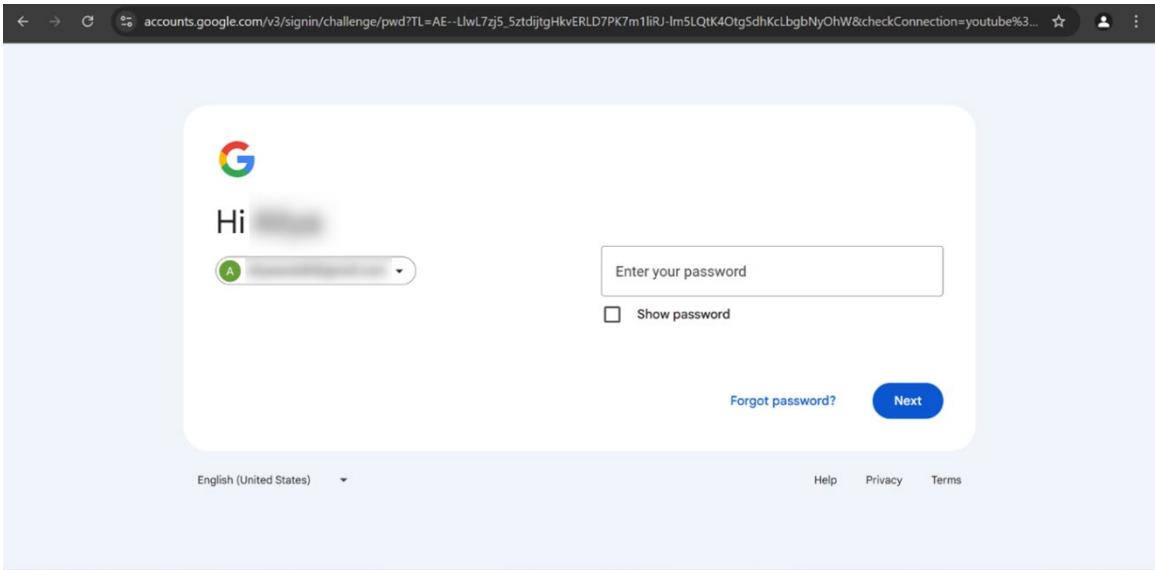


2. Escribe tu correo electrónico de Google o tu número de teléfono en el campo **Correo electrónico o teléfono** y selecciona **Siguiente**.



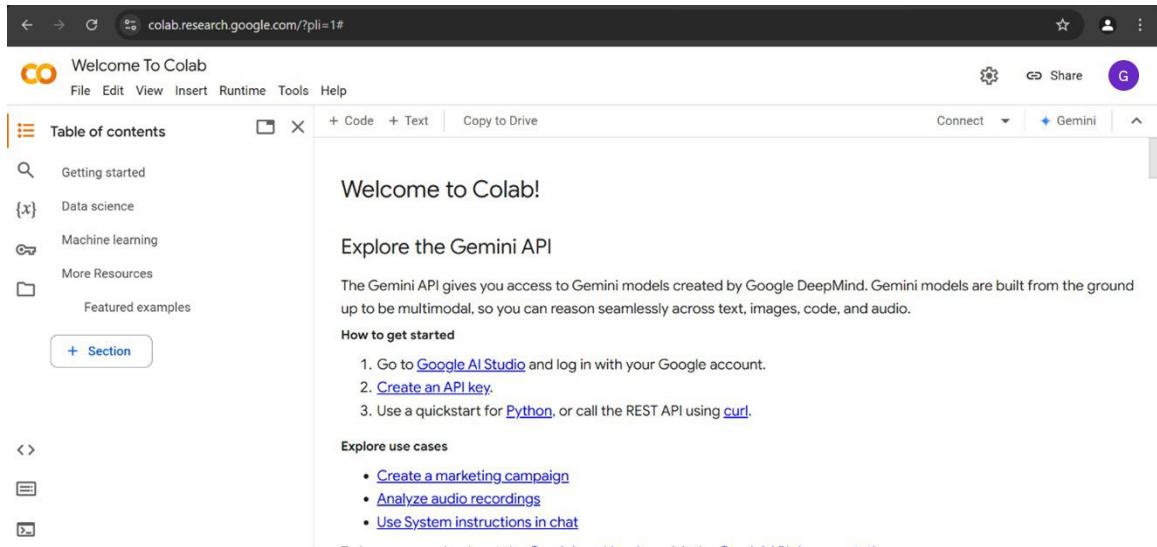
The screenshot shows the Google Sign-in page. At the top, there's a navigation bar with back, forward, and search icons. The URL in the address bar is `accounts.google.com/v3/signin/identifier?continue=https%3A%2F%2Fcolab.research.google.com%2F&ec=GAZaQMQ&ifkv=AeZLP9_W1arf0OUWv_MFuEy-Rv...`. Below the address bar is a large white sign-in box. It features the Google 'G' logo and the text 'Sign in'. Below that is the placeholder 'Use your Google Account'. To the right of the account field is a blue-bordered input field labeled 'Email or phone'. Below this input field is a link 'Forgot email?'. Further down, there's a note 'Not your computer? Use Guest mode to sign in privately.' followed by a link 'Learn more about using Guest mode'. At the bottom of the sign-in box are two buttons: 'Create account' and a blue 'Next' button. At the very bottom of the page, there are links for 'English (United States)', 'Help', 'Privacy', and 'Terms'.

3. Escribe tu contraseña en el campo **Ingresá tu contraseña** y selecciona **Siguiente**.



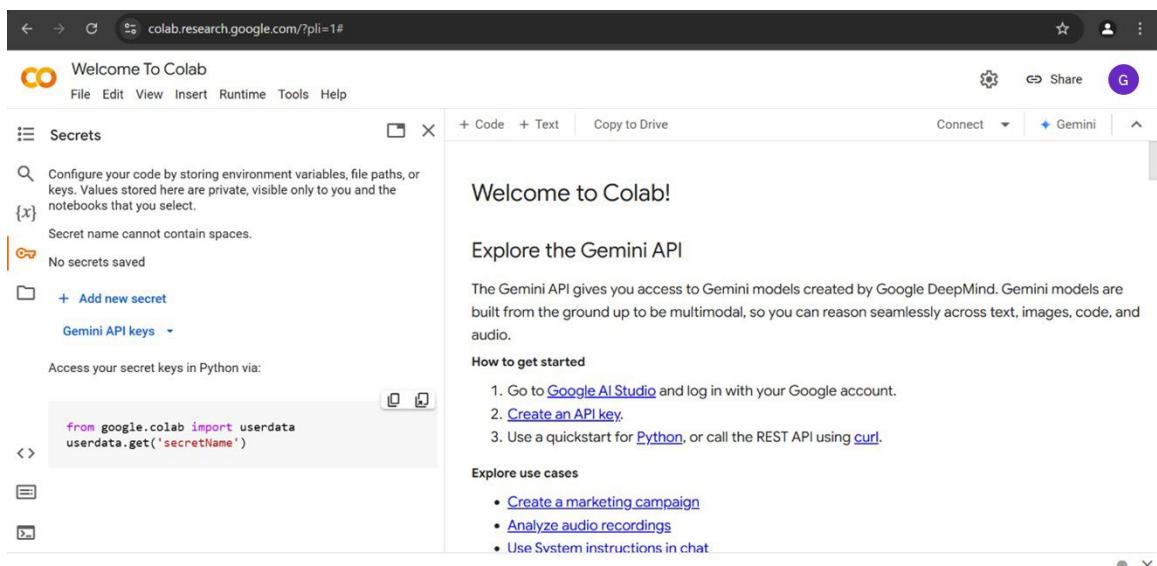
The screenshot shows the Google Sign-in page after entering an email or phone number. The page now displays a message 'Hi [REDACTED]' where [REDACTED] is the user's email address. Below this is a dropdown menu with the letter 'A' and a blurred background. To the right is a blue-bordered input field labeled 'Enter your password'. Below this input field is a checkbox labeled 'Show password'. At the bottom of the sign-in box are two buttons: 'Forgot password?' and a blue 'Next' button. At the very bottom of the page, there are links for 'English (United States)', 'Help', 'Privacy', and 'Terms'.

4. Una vez que hayas iniciado sesión, selecciona el ícono de la **Llave** para guardar tu token API de Replicate en Secretos de Google Colab.



The screenshot shows the Google Colab interface with the URL `colab.research.google.com/?pli=1#`. The left sidebar is titled "Table of contents" and includes sections like "Getting started", "Data science", "Machine learning", "More Resources", and "Featured examples". A button labeled "+ Section" is visible. The main content area displays the "Welcome to Colab!" page, which introduces the Gemini API. It states that the Gemini API provides access to multimodal models created by Google DeepMind. The "How to get started" section lists three steps: 1. Go to [Google AI Studio](#) and log in with your Google account. 2. [Create an API key](#). 3. Use a quickstart for [Python](#), or call the REST API using [curl](#). Below this, the "Explore use cases" section lists three items: "Create a marketing campaign", "Analyze audio recordings", and "Use System instructions in chat".

5. Selecciona **Añadir secreto nuevo**.



The screenshot shows the Google Colab interface with the URL `colab.research.google.com/?pli=1#`. The left sidebar is titled "Secrets" and shows a note about configuring code by storing environment variables, file paths, or keys. It also indicates that secret names cannot contain spaces and that no secrets have been saved. A button labeled "+ Add new secret" is present. The main content area displays the "Welcome to Colab!" page, identical to the one in the previous screenshot, with the Gemini API documentation and its use cases.

6. Escribe **api_token** en el campo **Nombre**.

The screenshot shows the Google Colab interface. On the left, there is a sidebar titled 'Secrets' with a table header 'Notebook access' and columns 'Name', 'Value', and 'Actions'. A single row is present with a delete icon in the Actions column. Below the table, there are buttons for '+ Add new secret' and 'Gemini API keys'. A code cell below the sidebar contains Python code to access a secret key:

```
from google.colab import userdata
userdata.get('secretName')
```

The main content area displays the 'Welcome to Colab!' message and information about the Gemini API.

7. Pega tu token API de Replicate en el campo **Valor**.

The screenshot shows the Google Colab interface. The 'Secrets' sidebar now has a row where 'Name' is 'api_token' and 'Value' is filled with a long string of characters. The rest of the interface is identical to the previous screenshot, showing the 'Welcome to Colab!' message and Gemini API information.

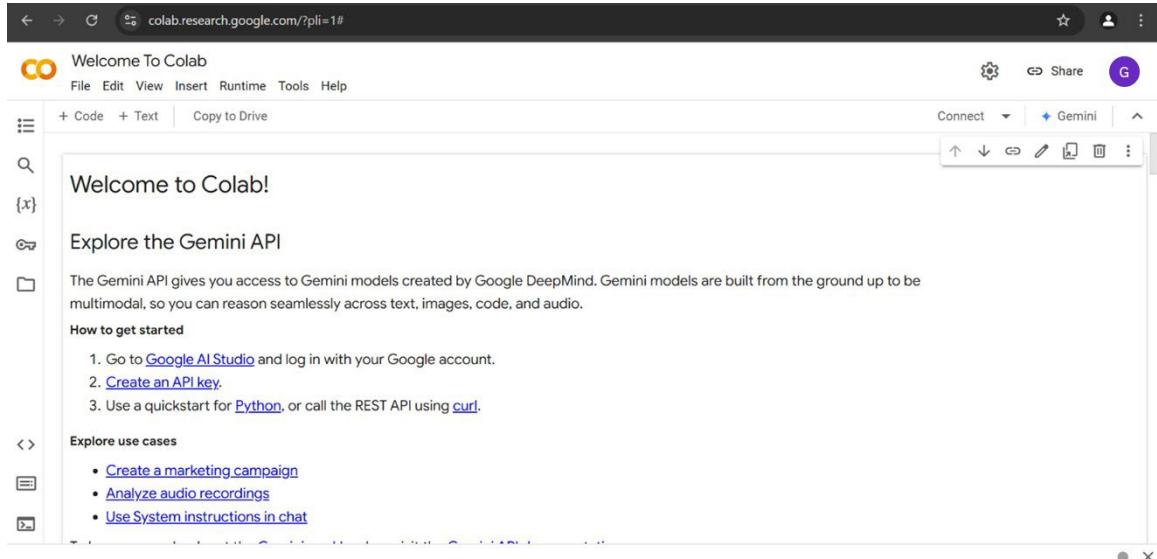
8. Selecciona el botón de **alternancia** para habilitar **el acceso al cuaderno**.

The screenshot shows the Google Colab interface. On the left, there is a sidebar titled "Secrets" which lists a single entry: "Notebook access" with "Name" set to "api_token" and "Value" set to ".....". Below this, there is a code cell containing Python code to access secret keys. On the right, the main workspace displays the "Welcome to Colab!" page, which includes sections for "Explore the Gemini API" and "Explore use cases".

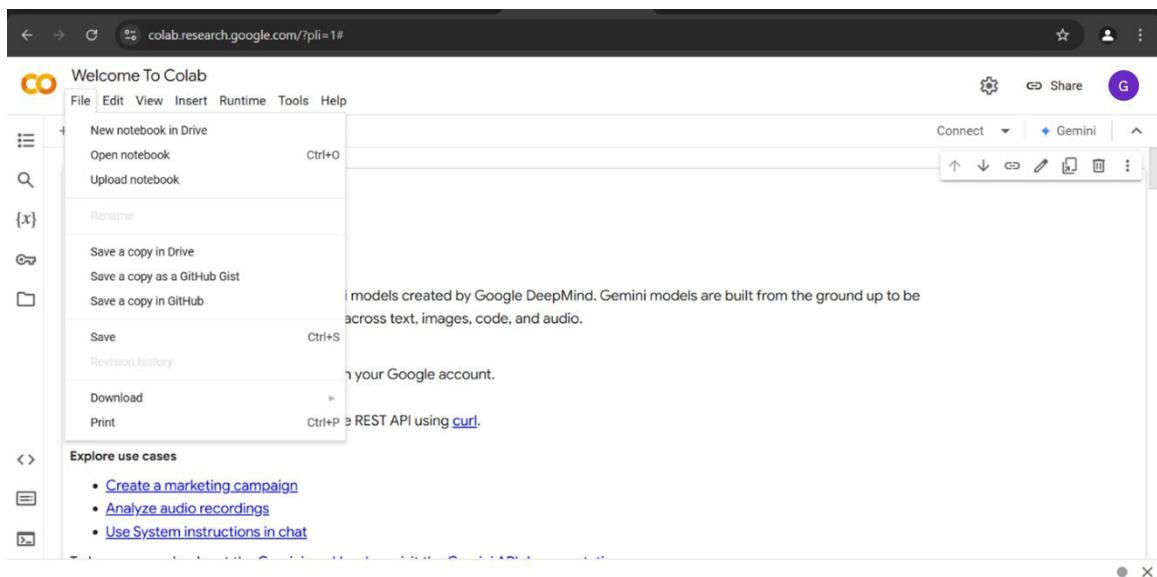
9. Selecciona **Cerrar (X)** para salir de la configuración.

This screenshot is identical to the one above, but the "Notebook access" entry in the "Secrets" sidebar now has a checked checkbox next to it, indicating that the "access" button has been selected.

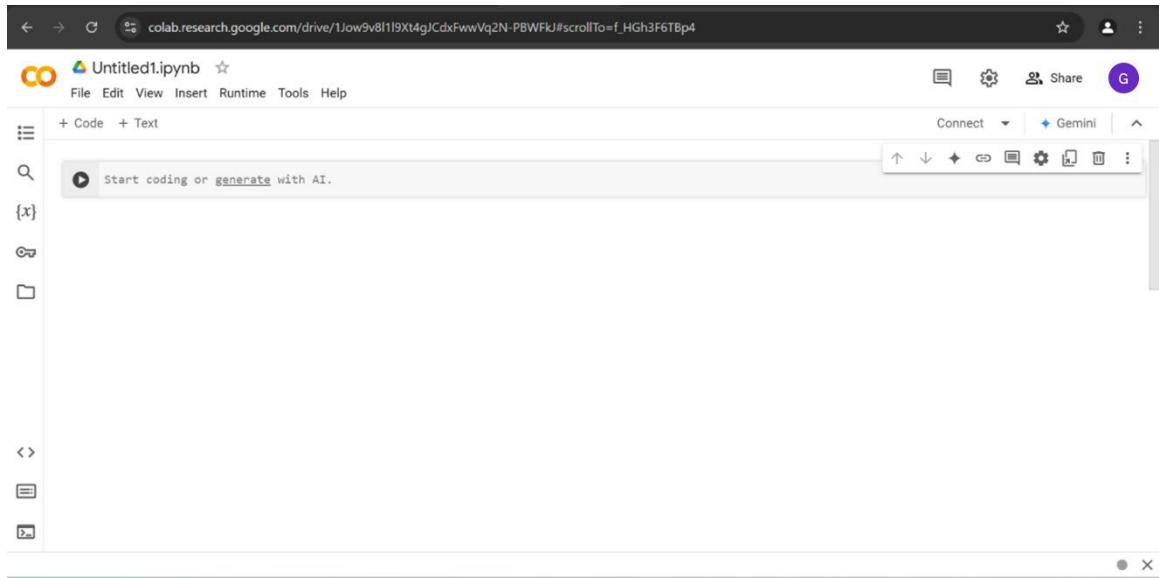
10. Selecciona **Archivo** para abrir un nuevo cuaderno.



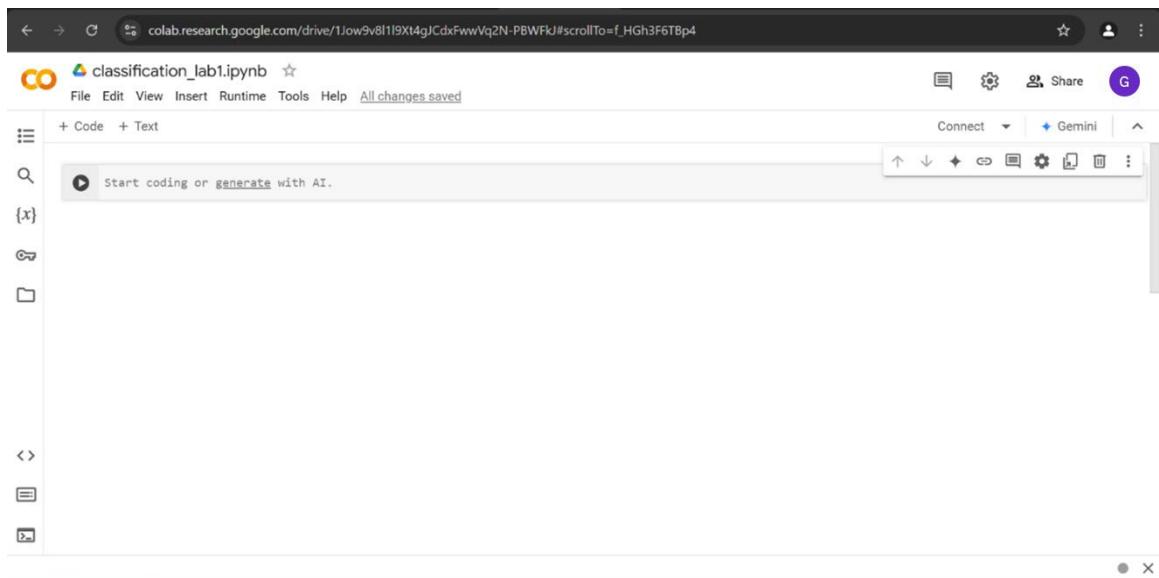
11. Selecciona **Cuaderno nuevo en Drive**.



12. Selecciona **Untitled1.ipynb** y escribe el nombre de archivo **classification_lab1** para cambiar el nombre del cuaderno.

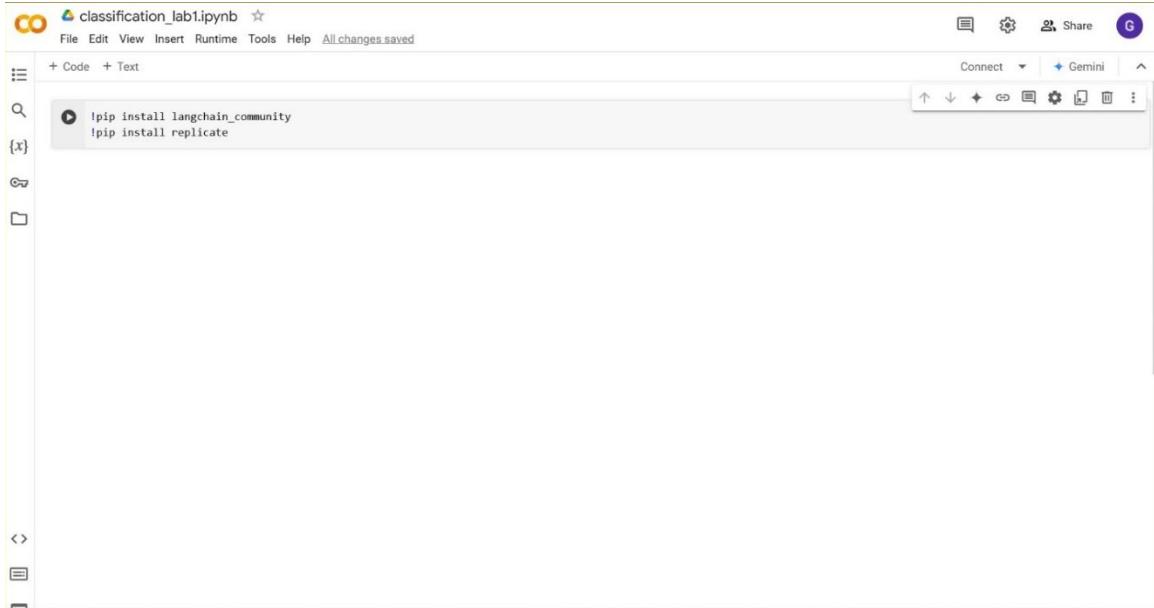


13. Has configurado el archivo correctamente. Ahora estás listo para instalar los paquetes y las bibliotecas necesarios.



14. Para instalar los paquetes Python necesarios, escribe este código en el campo **Instrucción**:

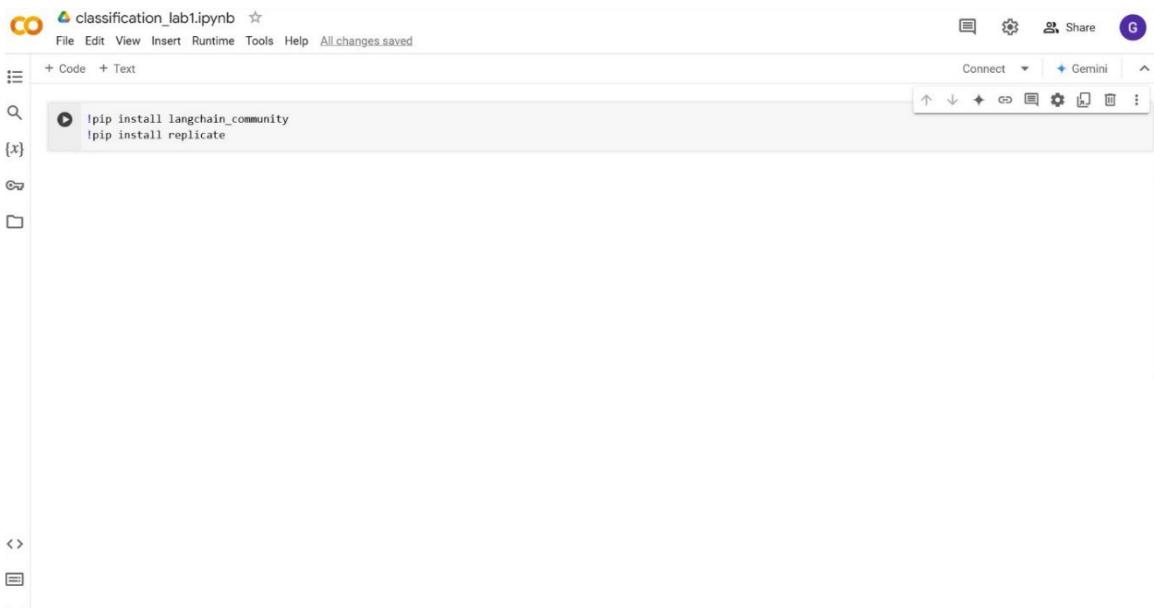
```
!pip install langchain_community  
!pip install replicate
```



A screenshot of a Jupyter Notebook interface. The title bar shows "classification_lab1.ipynb". The code cell contains the following text:

```
!pip install langchain_community  
!pip install replicate
```

15. Selecciona el botón **Reproducir** para ejecutar la solicitud inicial.



A screenshot of a Jupyter Notebook interface, identical to the previous one, but with a play button icon preceding the code cell, indicating it is ready to be executed.

16. El modelo genera el resultado. Ten en cuenta que el resultado muestra la instalación correcta de los paquetes.

```

Attempting uninstall: langchain-core
Found existing installation: langchain-core 0.3.24
Uninstalling langchain-core-0.3.24:
Successfully uninstalled langchain-core-0.3.24

Attempting uninstall: langchain-text-splitters
Found existing installation: langchain-text-splitters 0.3.2
Uninstalling langchain-text-splitters-0.3.2:
Successfully uninstalled langchain-text-splitters-0.3.2

Attempting uninstall: langchain
Found existing installation: langchain 0.3.11
Uninstalling langchain-0.3.11:
Successfully uninstalled langchain-0.3.11

Successfully installed dataclasses-json-0.6.7 htxx-sse-0.4.0 langchain-0.3.12 langchain-core-0.3.25 langchain-text-splitters-0.3.3 langchain_community-0.3.12 marshma
Collecting replicate
    Downloading replicate-1.0.4-py3-none-any.whl.metadata (29 kB)
Requirement already satisfied: htxx<1,>=0.21.0 in /usr/local/lib/python3.10/dist-packages (from replicate) (0.28.1)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from replicate) (24.2)
Requirement already satisfied: pydantic>1.10.7 in /usr/local/lib/python3.10/dist-packages (from replicate) (2.10.3)
Requirement already satisfied: typing_extensions>4.5.0 in /usr/local/lib/python3.10/dist-packages (from replicate) (4.12.2)
Requirement already satisfied: aioio in /usr/local/lib/python3.10/dist-packages (from htxx<1,>=0.21.0->replicate) (3.7.1)
Requirement already satisfied: httpcore<1.* in /usr/local/lib/python3.10/dist-packages (from htxx<1,>=0.21.0->replicate) (2024.8.30)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from htxx<1,>=0.21.0->replicate) (1.0.7)
Requirement already satisfied: idna in /usr/local/lib/python3.10/dist-packages (from htxx<1,>=0.21.0->replicate) (3.10)
Requirement already satisfied: htxx<0.15,>=0.13 in /usr/local/lib/python3.10/dist-packages (from htxx<1,>=0.21.0->replicate) (0.14.0)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.10/dist-packages (from pydantic>1.10.7->replicate) (0.7.0)
Requirement already satisfied: pydantic-core=>2.27.1 in /usr/local/lib/python3.10/dist-packages (from pydantic>1.10.7->replicate) (2.27.1)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.10/dist-packages (from aioio->htxx<1,>=0.21.0->replicate) (1.3.1)
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from aioio->htxx<1,>=0.21.0->replicate) (1.2.2)
Downloading replicate-1.0.4-py3-none-any.whl (48 kB)

48.0/48.0 kB 3.6 MB/s eta 0:00:00

Installing collected packages: replicate
Successfully installed replicate-1.0.4

```

17. Selecciona los tres puntos (puntos suspensivos verticales) en la esquina superior derecha de la celda de resultados para borrar el resultado.

```

Attempting uninstall: langchain-core
Found existing installation: langchain-core 0.3.24
Uninstalling langchain-core-0.3.24:
Successfully uninstalled langchain-core-0.3.24

Attempting uninstall: langchain-text-splitters
Found existing installation: langchain-text-splitters 0.3.2
Uninstalling langchain-text-splitters-0.3.2:
Successfully uninstalled langchain-text-splitters-0.3.2

Attempting uninstall: langchain
Found existing installation: langchain 0.3.11
Uninstalling langchain-0.3.11:
Successfully uninstalled langchain-0.3.11

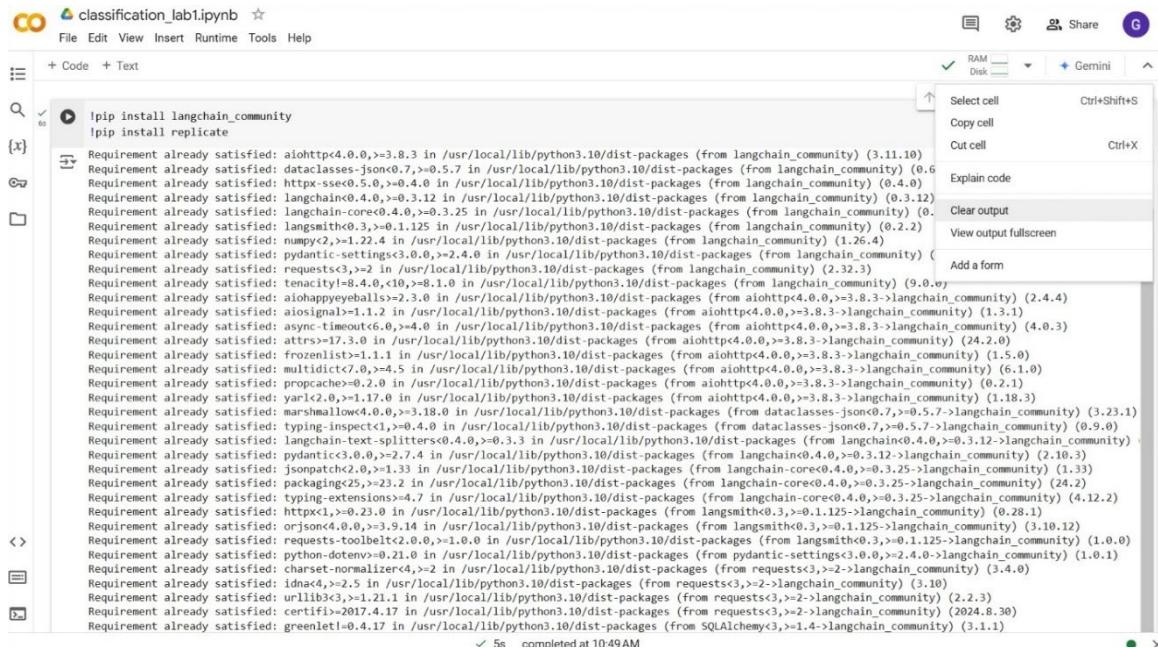
Successfully installed dataclasses-json-0.6.7 htxx-sse-0.4.0 langchain-0.3.12 langchain-core-0.3.25 langchain-text-splitters-0.3.3 langchain_community-0.3.12 marshma
Collecting replicate
    Downloading replicate-1.0.4-py3-none-any.whl.metadata (29 kB)
Requirement already satisfied: htxx<1,>=0.21.0 in /usr/local/lib/python3.10/dist-packages (from replicate) (0.28.1)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from replicate) (24.2)
Requirement already satisfied: pydantic>1.10.7 in /usr/local/lib/python3.10/dist-packages (from replicate) (2.10.3)
Requirement already satisfied: typing_extensions>4.5.0 in /usr/local/lib/python3.10/dist-packages (from replicate) (4.12.2)
Requirement already satisfied: aioio in /usr/local/lib/python3.10/dist-packages (from htxx<1,>=0.21.0->replicate) (3.7.1)
Requirement already satisfied: httpcore<1.* in /usr/local/lib/python3.10/dist-packages (from htxx<1,>=0.21.0->replicate) (2024.8.30)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from htxx<1,>=0.21.0->replicate) (1.0.7)
Requirement already satisfied: idna in /usr/local/lib/python3.10/dist-packages (from htxx<1,>=0.21.0->replicate) (3.10)
Requirement already satisfied: htxx<0.15,>=0.13 in /usr/local/lib/python3.10/dist-packages (from htxx<1,>=0.21.0->replicate) (0.14.0)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.10/dist-packages (from pydantic>1.10.7->replicate) (0.7.0)
Requirement already satisfied: pydantic-core=>2.27.1 in /usr/local/lib/python3.10/dist-packages (from pydantic>1.10.7->replicate) (2.27.1)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.10/dist-packages (from aioio->htxx<1,>=0.21.0->replicate) (1.3.1)
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from aioio->htxx<1,>=0.21.0->replicate) (1.2.2)
Downloading replicate-1.0.4-py3-none-any.whl (48 kB)

48.0/48.0 kB 3.6 MB/s eta 0:00:00

Installing collected packages: replicate
Successfully installed replicate-1.0.4

```

18. Selecciona **Borrar resultado** desde el menú desplegable. Esta acción elimina el resultado actual de la celda.

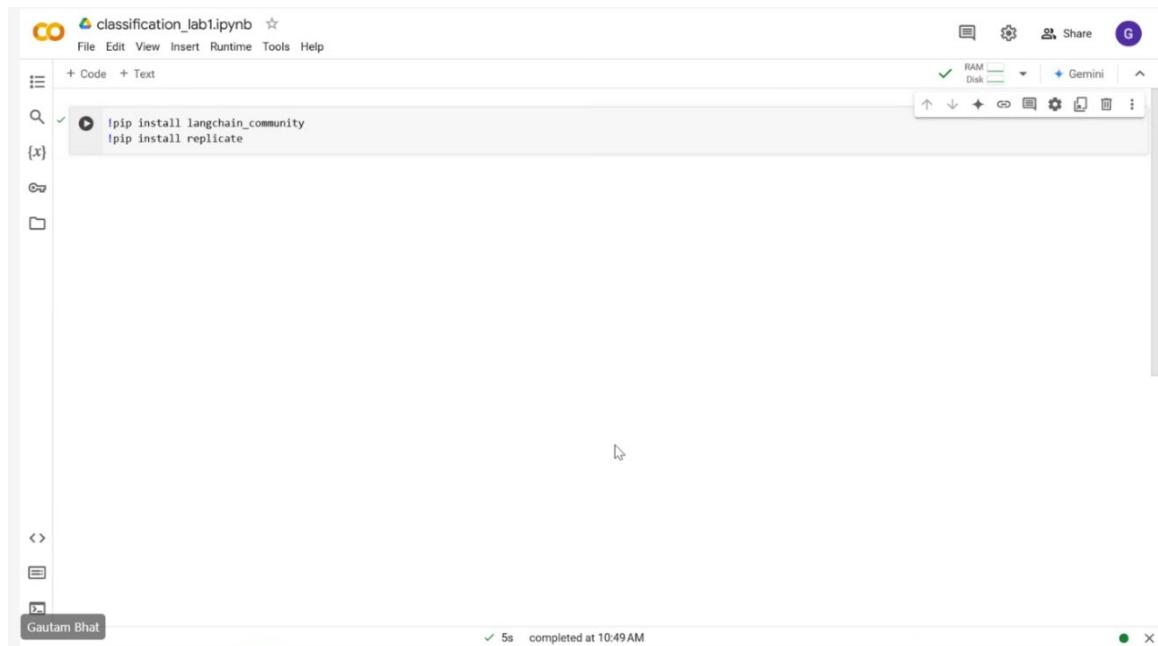


```

classification_lab1.ipynb
File Edit View Insert Runtime Tools Help
+ Code + Text
[ ] ⓘ
!pip install langchain_community
!pip install replicate
Requirement already satisfied: aiohttp<4.0.0,>~=3.8.3 in /usr/local/lib/python3.10/dist-packages (from langchain_community) (3.11.10)
Requirement already satisfied: dataclasses-json<0.7,>~=0.5.7 in /usr/local/lib/python3.10/dist-packages (from langchain_community) (0.6)
Requirement already satisfied: httpx-sse<5.0.0,>~=0.4.0 in /usr/local/lib/python3.10/dist-packages (from langchain_community) (0.4.0)
Requirement already satisfied: langchain<0.4.0,>~=0.3.12 in /usr/local/lib/python3.10/dist-packages (from langchain_community) (0.3.12)
Requirement already satisfied: langchain-core<0.4.0,>~=0.3.25 in /usr/local/lib/python3.10/dist-packages (from langchain_community) (0.3.25)
Requirement already satisfied: langsmith<0.3,>~=0.1.125 in /usr/local/lib/python3.10/dist-packages (from langchain_community) (0.2.2)
Requirement already satisfied: numpy<1.22.4,>~=1.22.4 in /usr/local/lib/python3.10/dist-packages (from langchain_community) (1.26.4)
Requirement already satisfied: pydantic-settings<3.0.0,>~=4.0 in /usr/local/lib/python3.10/dist-packages (from langchain_community) (4.0)
Requirement already satisfied: requests<,>~=2.0 in /usr/local/lib/python3.10/dist-packages (from langchain_community) (2.32.3)
Requirement already satisfied: tenacity<4.0.0,>~=0.1.0 in /usr/local/lib/python3.10/dist-packages (from langchain_community) (9.0.0)
Requirement already satisfied: aioappy eyeballs<2.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>~=3.8.3>langchain_community) (2.4.4)
Requirement already satisfied: aiosignal<1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>~=3.8.3>langchain_community) (1.3.1)
Requirement already satisfied: async-timeout<6.0,>~=4.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>~=3.8.3>langchain_community) (4.0.3)
Requirement already satisfied: attrs<17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>~=3.8.3>langchain_community) (24.2.0)
Requirement already satisfied: frozenlist<1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>~=3.8.3>langchain_community) (1.5.0)
Requirement already satisfied: multidict<7.0,>~=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>~=3.8.3>langchain_community) (6.1.0)
Requirement already satisfied: proprache<0.2.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>~=3.8.3>langchain_community) (0.2.1)
Requirement already satisfied: yarl<2.0,>~=1.17.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>~=3.8.3>langchain_community) (1.18.3)
Requirement already satisfied: marshmallow<4.0.0,>~=3.18.0 in /usr/local/lib/python3.10/dist-packages (from dataclasses-json<0.7,>~=0.5.7>langchain_community) (3.23.1)
Requirement already satisfied: typing-inspect<,>~=0.4.0 in /usr/local/lib/python3.10/dist-packages (from dataclasses-json<0.7,>~=0.5.7>langchain_community) (0.9.0)
Requirement already satisfied: langchain-text-splitters<0.4.0,>~=0.3.3 in /usr/local/lib/python3.10/dist-packages (from langchain<0.4.0,>~=0.3.12>langchain_community) (0.3.3)
Requirement already satisfied: pydantic<3.0.0,>~=7.4 in /usr/local/lib/python3.10/dist-packages (from langchain<0.4.0,>~=0.3.12>langchain_community) (2.10.3)
Requirement already satisfied: jsonpatch<2.0,>~=1.33 in /usr/local/lib/python3.10/dist-packages (from langchain-core<0.4.0,>~=0.3.25>langchain_community) (1.33)
Requirement already satisfied: packaging<25,>~=23.2 in /usr/local/lib/python3.10/dist-packages (from langchain-core<0.4.0,>~=0.3.25>langchain_community) (24.2)
Requirement already satisfied: typing-extensions<4.7 in /usr/local/lib/python3.10/dist-packages (from langchain-core<0.4.0,>~=0.3.25>langchain_community) (4.12.2)
Requirement already satisfied: httpx<1.20.0 in /usr/local/lib/python3.10/dist-packages (from langsmith<0.3,>~=0.1.125>langchain_community) (0.28.1)
Requirement already satisfied: orjson<4.0.0,>~=3.9.10 in /usr/local/lib/python3.10/dist-packages (from langsmith<0.3,>~=0.1.125>langchain_community) (3.10.12)
Requirement already satisfied: requests-toolbelt<2.0.0,>~=1.0.0 in /usr/local/lib/python3.10/dist-packages (from langsmith<0.3,>~=0.1.125>langchain_community) (1.0.0)
Requirement already satisfied: python-dotenv<0.21.0 in /usr/local/lib/python3.10/dist-packages (from requests<,>~=2>langchain_community) (3.10)
Requirement already satisfied: charset-normalizer<4,>~=2 in /usr/local/lib/python3.10/dist-packages (from requests<,>~=2>langchain_community) (3.4.0)
Requirement already satisfied: idna<,>~=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<,>~=2>langchain_community) (3.10)
Requirement already satisfied: urllib3<3,>~=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<,>~=2>langchain_community) (2.2.3)
Requirement already satisfied: certifi<2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<,>~=2>langchain_community) (2024.8.30)
Requirement already satisfied: greenlet<0.4.17 in /usr/local/lib/python3.10/dist-packages (from SQLAlchemy<3,>~=1.4>langchain_community) (3.1.1)

```

19. Selecciona **+Código** para agregar una nueva celda de código a tu cuaderno. Esta acción te permite escribir y ejecutar código Python adicional en una celda nueva y vacía.



```

classification_lab1.ipynb
File Edit View Insert Runtime Tools Help
+ Code + Text
[ ] ⓘ
!pip install langchain_community
!pip install replicate

```

20. Configura el entorno importando las bibliotecas necesarias, extrayendo el token de API, configurándolo como una variable de entorno, definiendo el modelo y creando una instancia del modelo Replicate.

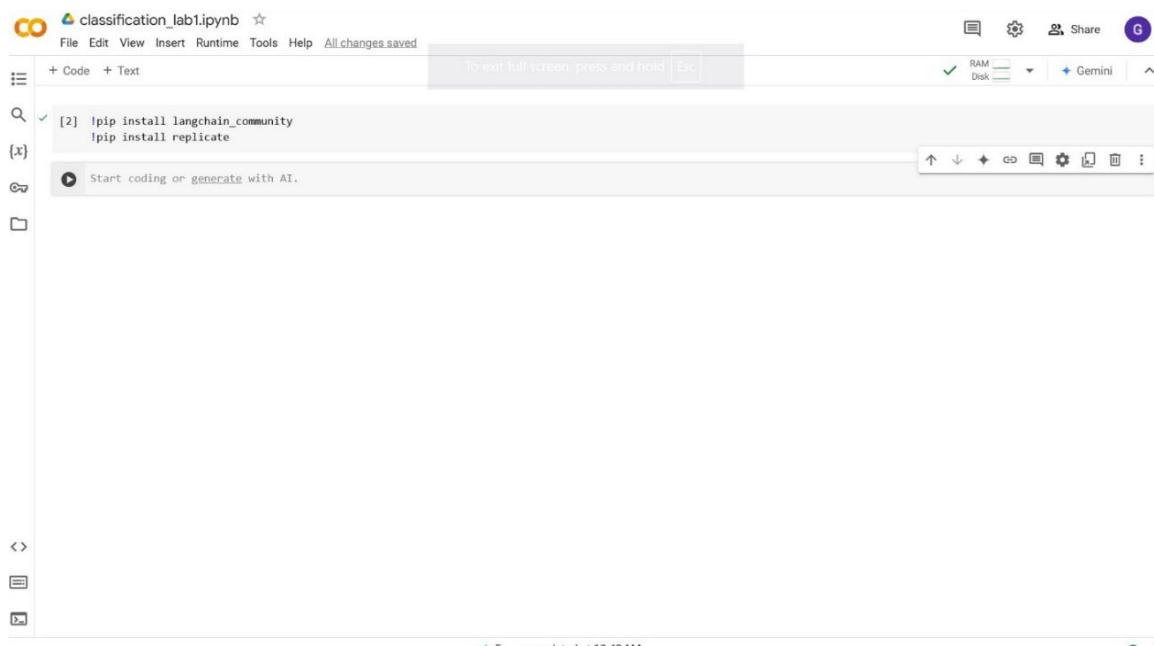
Escribe este código en el campo **Instrucción**:

```
desde langchain_community.llms importar Replicate
importar sistema operativo
desde google.colab importar userdatas
```

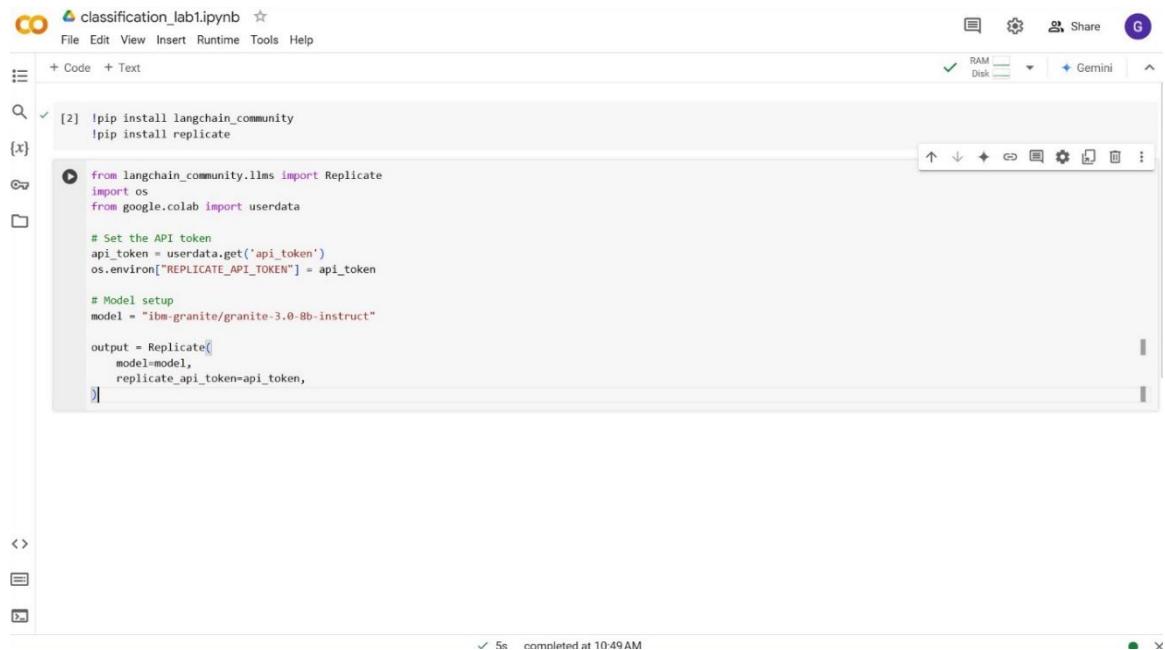
```
# Establecer el token de API
api_token = userdatas.get('api_token')
os.environ["REPLICATE_API_TOKEN"] = api_token

# Configuración del modelo
modelo = "ibm-granite/granite-3.0-8b-instruct"

resultado = Replicate(
    modelo=modelo,
    replicate_api_token=api_token,
)
```



21. Has configurado el modelo y ahora estás listo para comenzar a clasificar las reseñas de los clientes.



The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** classification_lab1.ipynb
- Toolbar:** File, Edit, View, Insert, Runtime, Tools, Help
- Code Cell:** [2] !pip install langchain_community
!pip install replicate
- Code Block:** from langchain_community.llms import Replicate
import os
from google.colab import userdata

Set the API token
api_token = userdata.get('api_token')
os.environ["REPLICATE_API_TOKEN"] = api_token

Model setup
model = "ibm-granite/granite-3.0-8b-instruct"

output = Replicate(
 model=model,
 replicate_api_token=api_token,
)
- Runtime Status:** ✓ 5s completed at 10:49 AM
- Runtime Selection:** RAM, Disk, Gemini
- Share Button:** Share

Paso 4: Proporcionar reseñas de clientes para su clasificación y probar la solicitud inicial**Descripción**

En este paso, ingresarás las reseñas de los clientes en el modelo, ejecutarás una solicitud simple para clasificar las reseñas y analizarás el resultado para señalar posibles mejoras.

Instrucciones

1. Ingresá muestras de reseñas de clientes para probar el modelo. Escribe estas opiniones de clientes en el campo **Instrucción**:

```
# Definir las opiniones de los clientes
customer_reviews = [
    "La batería dura todo el día y el rendimiento es excelente".
    "La pantalla es demasiado tenue en exteriores, pero me
    encantan los colores en interiores".
    "Este teléfono es lento y falla cuando abro ciertas
    aplicaciones".
]

# Refinar la solicitud para incluir reseñas
reviews_text = "\n".join([f"Review {i+1}: {review}" for i, review
in enumerate(customer_reviews)])

prompt = f"""
Clasifica estas reseñas como positivas, negativas o mixtas:

{reviews_text}
"""

# Invocar el modelo con la solicitud de ejemplo
respuesta = output.invoke(prompt)

# Imprimir la respuesta
print("Granite Model Response:\n")

print(response)
```

```
classification_lab1.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
os.environ["REPLICATE_API_TOKEN"] = api_token
# Model setup
model = "ibm-granite/granite-3.0-8b-instruct"
output = Replicate(
    model=model,
    replicate_api_token=api_token,
)
# Define the customer reviews
customer_reviews = [
    "The battery lasts all day, but the phone gets hot during gaming.",
    "The screen is too dim outdoors, but I love the colors indoors.",
    "This phone is fast, but it keeps crashing when I open certain apps."
]
# Refine the prompt to include reviews
reviews_text = "\n".join([f"Review {i+1}: {review}" for i, review
in enumerate(customer_reviews)])
prompt = """
Classify these reviews as positive, negative, or mixed:
{reviews_text}
"""
# Invoke the model with the example prompt
response = output.invoke(prompt)
# Print the response
print("Granite Model Response:\n")
print(response)
```

2. Selecciona el botón **Reproducir** para ejecutar la solicitud inicial.

```
classification_lab1.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
os.environ["REPLICATE_API_TOKEN"] = api_token
# Model setup
model = "ibm-granite/granite-3.0-8b-instruct"
output = Replicate(
    model=model,
    replicate_api_token=api_token,
)
# Define the customer reviews
customer_reviews = [
    "The battery lasts all day, but the phone gets hot during gaming.",
    "The screen is too dim outdoors, but I love the colors indoors.",
    "This phone is fast, but it keeps crashing when I open certain apps."
]
# Refine the prompt to include reviews
reviews_text = "\n".join([f"Review {i+1}: {review}" for i, review
in enumerate(customer_reviews)])
prompt = """
Classify these reviews as positive, negative, or mixed:
{reviews_text}
"""
# Invoke the model with the example prompt
response = output.invoke(prompt)
# Print the response
print("Granite Model Response:\n")
print(response)
```

3. Lee el resultado que genera el modelo. Ten en cuenta que el modelo identifica correctamente todas las reseñas como "mixtas", pero no logra captar detalles específicos de los aspectos prioritarios, como el rendimiento, la batería o la pantalla, de una manera clara y estructurada. La falta de énfasis explícito en los aspectos positivos y negativos hace que la respuesta sea menos práctica o esclarecedora.

Para abordar esto, es necesario refinar la solicitud para mejorar la claridad y la estructura del resultado, garantizando que destaque los aspectos positivos y negativos específicos de cada reseña.

The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** classification_lab1.ipynb
- Toolbar:** File, Edit, View, Insert, Runtime, Tools, Help, All changes saved
- Code Cell:** Contains Python code for invoking a model with different prompts and printing the response.

```
# Invoke the model with the multitask prompt
# response = output.invoke(multitask_prompt)

{x}
# Invoke the model with the example prompt
# response = output.invoke(example_prompt)

# Print the response
# print("Granite Model Response:\n")
# print("Granite Model Refined Response:\n")
```

- Output Cell:** Displays the "Granite Model Response:" section with three items, each describing a review as "Mixed".

```
Granite Model Response:
1. Mixed: The review mentions both a positive aspect (battery lasting all day) and a negative aspect (getting hot during gaming).
2. Mixed: The review mentions both a negative aspect (screen is too dim outdoors) and a positive aspect (loves the colors indoors).
3. Mixed: The review mentions both a positive aspect (phone is fast) and a negative aspect (keeps crashing when opening certain apps).
```

Paso 5: Refinar la solicitud para mejorar el resultado de la clasificación**Descripción**

En este paso, refinará la solicitud paso a paso para mejorar la precisión y la estructura.

Instrucciones

1. **Recuerda:** Especificar los aspectos prioritarios en la solicitud ayuda al modelo a centrarse en aspectos específicos de la reseña, como la duración de la batería, la calidad de la pantalla o el rendimiento. Esta orientación lleva a una clasificación más precisa y proporciona una comprensión más clara de las fortalezas y debilidades mencionadas en cada reseña. Al indicar explícitamente los aspectos prioritarios, el modelo no solo generaliza el sentimiento sino que también resalta los elementos clave que son importantes para el cliente.

Actualiza la solicitud especificando los aspectos prioritarios. Escribe esta solicitud en el campo Instrucción:

```
# Definir una solicitud refinada
refined_prompt = f"""
Clasifica estas reseñas como positivas, negativas o mixtas y
etiqueta las categorías pertinentes (duración de la batería,
calidad de la pantalla o rendimiento):

{reviews_text}
"""

# Invocar el modelo con la solicitud de ejemplo
respuesta = output.invoke(refined_prompt)

# Imprimir la respuesta
print("Granite Model Refined Response:\n")

print(response)
```

```
# # Print the response
# print("Granite Model Response:\n")

{x}

# Define refined prompt
refined_prompt = f"""
Classify these reviews as positive, negative, or mixed, and
tag relevant categories (battery life, screen quality, or performance).

{reviews_text}
"""

# Invoke the model with the refined prompt
response = output.invoke(refined_prompt)

# Print the response
print("Granite Model Refined Response:\n")

print(response)

# Define the prompt to complete the task in 2 steps
# multitask_prompt = f"""
# Complete the task in 2 steps.

# Step 1: Classify these reviews as positive, negative, or mixed.
# Step 2: For each review, identify relevant categories: battery life, screen
# quality, or performance.

# {reviews_text}
# """
# response = output.invoke(multitask_prompt)
# print("Granite Model Response for multitask response:\n")
```

2. Selecciona el botón **Reproducir** para ejecutar la solicitud.

```
# # Print the response
# print("Granite Model Response:\n")

{x}

# Define refined prompt
refined_prompt = f"""
Classify these reviews as positive, negative, or mixed, and
tag relevant categories (battery life, screen quality, or performance).

{reviews_text}
"""

# Invoke the model with the refined prompt
response = output.invoke(refined_prompt)

# Print the response
print("Granite Model Refined Response:\n")

print(response)

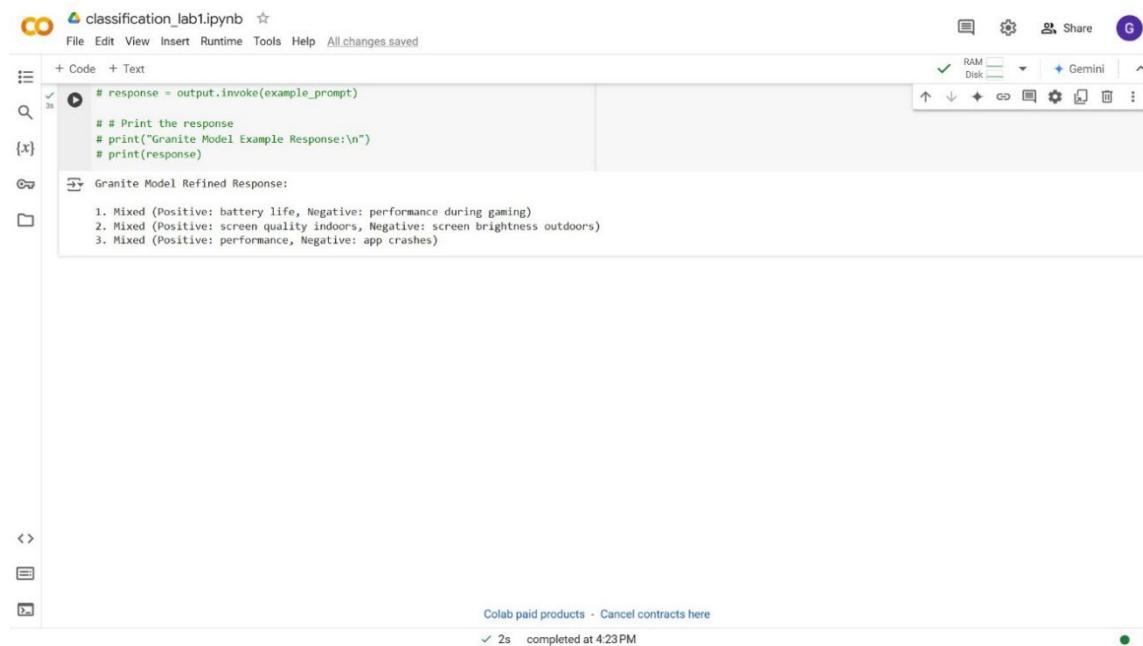
# Define the prompt to complete the task in 2 steps
# multitask_prompt = f"""
# Complete the task in 2 steps.

# Step 1: Classify these reviews as positive, negative, or mixed.
# Step 2: For each review, identify relevant categories: battery life, screen
# quality, or performance.

# {reviews_text}
# """
# response = output.invoke(multitask_prompt)
# print("Granite Model Response for multitask response:\n")
```

3. Lee el resultado que genera el modelo. Ten en cuenta que el resultado muestra una mejora en la claridad, pero el formato es inconsistente. Además, el modelo omite especificar más de un aspecto prioritario en ciertos casos, lo que limita la profundidad del análisis. Por ejemplo, la segunda reseña se centra únicamente en la pantalla, pero no menciona otros posibles aspectos como la usabilidad o el rendimiento.

Para mejorar aún más la respuesta y garantizar un análisis más completo, separemos la tarea en dos pasos.



```
# response = output.invoke(example_prompt)

# Print the response
# print("Granite Model Example Response:\n")
# print(response)

Granite Model Refined Response:

1. Mixed (Positive: battery life, Negative: performance during gaming)
2. Mixed (Positive: screen quality indoors, Negative: screen brightness outdoors)
3. Mixed (Positive: performance, Negative: app crashes)
```

4. **Recuerda:** Dividir la tarea en dos pasos permite que el modelo se centre primero en la clasificación de sentimientos y luego etiquete aspectos prioritarios específicos, como la duración de la batería, la calidad de la pantalla o el rendimiento. Este método garantiza un análisis más estructurado y exhaustivo, proporcionando información clara sobre el sentimiento general y los aspectos clave de la reseña. Al separar estos pasos, el modelo puede centrarse en cada tarea independientemente, mejorando la calidad y la profundidad del resultado.

Actualiza la solicitud dividiendo la tarea en dos pasos: clasificación de sentimientos y etiquetado de los aspectos prioritarios. Escribe esta solicitud en el campo

Instrucción:

```
# Definir la solicitud para completar la tarea en 2 pasos
multitask_prompt = f"""
Completa la tarea en 2 pasos.
```

Paso 1: Clasificar estas reseñas como positivas, negativas o mixtas.

Paso 2: Para cada reseña, identificar las categorías pertinentes: duración de la batería, calidad de la pantalla o rendimiento.

```
{reviews_text}
"""

respuesta = output.invoke(multitask_prompt)
print("Granite Model Response:\n")
print(response)
```

```
# Define the prompt to complete the task in 2 steps
multitask_prompt = f"""
Complete the task in 2 steps.

Step 1: Classify these reviews as Positive, Negative, or Mixed.
Step 2: For each review, identify relevant categories: battery life, screen quality, or performance.

{reviews_text}
"""

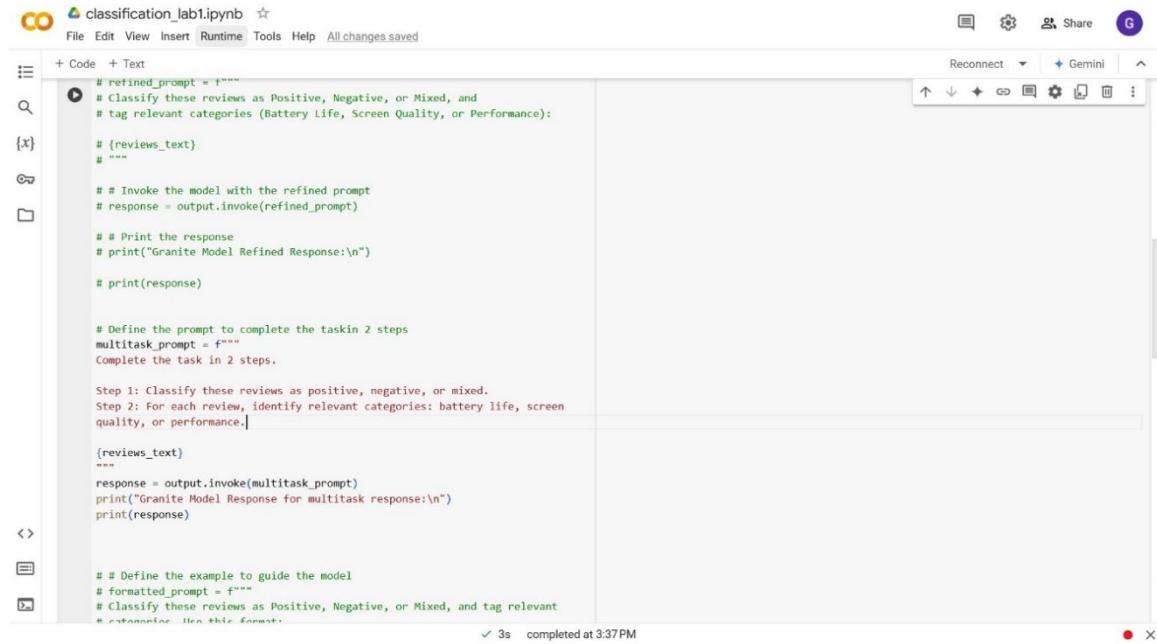
response = output.invoke(multitask_prompt)
print("Granite Model Response for multitask response:\n")
print(response)

# Define the example to guide the model
# formatted_prompt = f"""
# Classify these reviews as Positive, Negative, or Mixed, and tag relevant
# categories. Use this format.
# 
```

Solicitar a un modelo IBM Granite que clasifique y resuma datos

IBM SkillsBuild

5. Selecciona el botón **Reproducir** para ejecutar la solicitud.

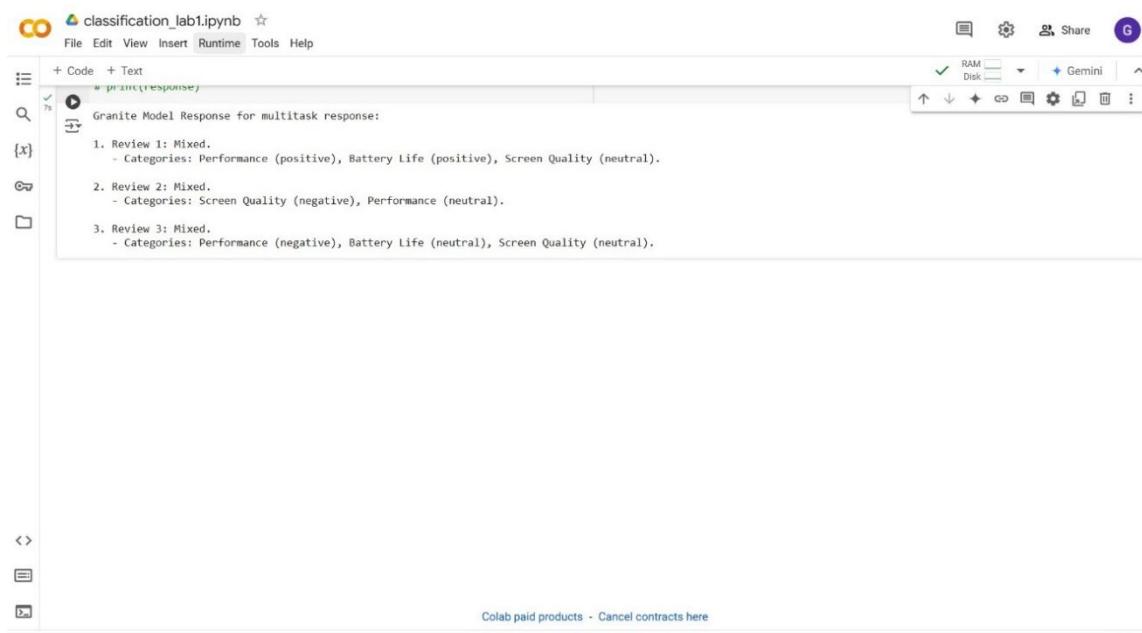


The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** classification_lab1.ipynb
- Toolbar:** File, Edit, View, Insert, Runtime, Tools, Help, All changes saved, Reconnect, Gemini, Share, G
- Code Cell:** Contains Python code for classifying reviews. It includes comments explaining the steps: defining a refined prompt, invoking the model, printing the response, defining a multitask prompt, and printing the multitask response. It also includes a note about defining examples to guide the model.
- Output Cell:** Shows the execution status: ✓ 3s completed at 3:37PM.

6. Lee el resultado que genera el modelo. Ten en cuenta que, si bien dividir la tarea en pasos mejora la precisión de la clasificación de sentimientos y la identificación de categorías, la estructura del resultado sigue siendo inconsistente. Por ejemplo, si bien la clasificación de sentimientos (positivo, negativo o mixto) es clara, la asignación de múltiples categorías en cada reseña muestra variación. Algunas categorías tienen sentimientos mixtos (positivos, negativos, neutrales) y esto podría beneficiarse de una estructura más consistente. El resultado no está estructurado de la mejor manera.

Para refinar aún más el resultado y mejorar la consistencia, guiamos a continuación al modelo para que siga un formato de resultado estructurado.



```
# print(response)

Granite Model Response for multitask response:
1. Review 1: Mixed.
   - Categories: Performance (positive), Battery Life (positive), Screen Quality (neutral).

2. Review 2: Mixed.
   - Categories: Screen Quality (negative), Performance (neutral).

3. Review 3: Mixed.
   - Categories: Performance (negative), Battery Life (neutral), Screen Quality (neutral).
```

7. **Recuerda:** Guiar el modelo para que siga un formato de resultado estructurado garantiza que la respuesta sea clara, consistente y fácil de interpretar. Al especificar el formato requerido, el modelo clasifica el sentimiento de cada reseña e identifica categorías pertinentes de manera estructurada. Esto ayuda a mantener la uniformidad en todos los resultados, lo que hace que sea más fácil analizarlos y compararlos.

Actualiza la solicitud guiando el modelo para que siga un formato de resultado estructurado. Escribe esta solicitud en el campo Instrucción:

```
# Definir el ejemplo para guiar el modelo
formatted_prompt = f"""
Clasifica estas reseñas como positivas, negativas o mixtas y
etiqueta las categorías pertinentes. Utiliza este formato:
```

- Sentimiento: [Sentimiento]

- Categorías: [Categorías].

```
{reviews_text}
"""

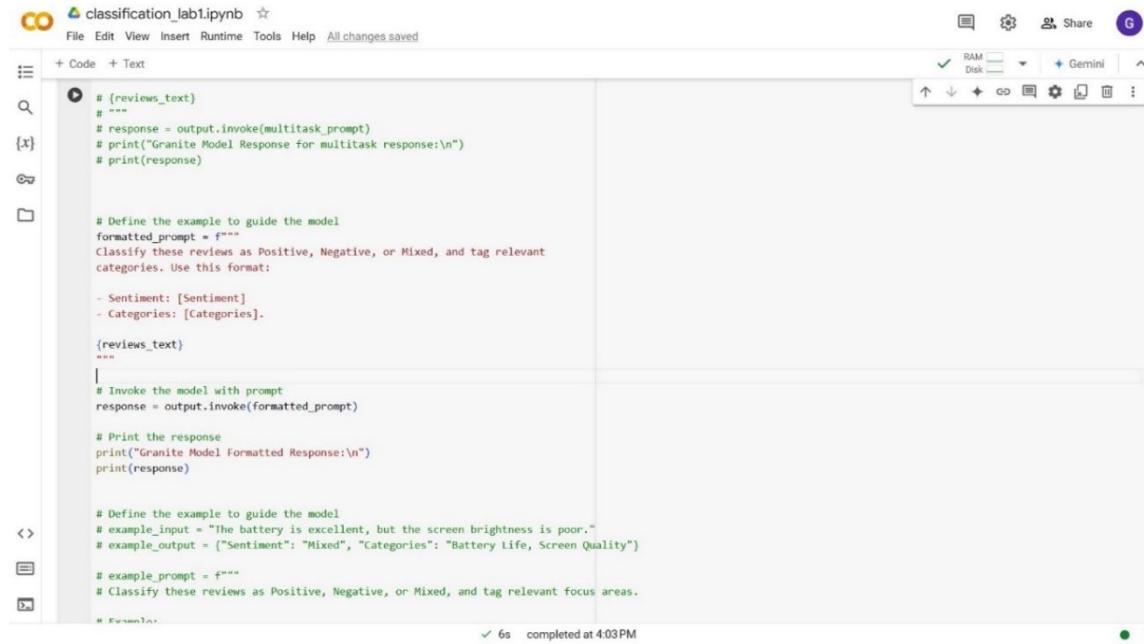
# Invocar el modelo con la solicitud
response = output.invoke(formatted_prompt)

# Imprimir la respuesta
print("Granite Model Formatted Response:\n")
print(response)
```

The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** classification_lab1.ipynb
- Toolbar:** File, Edit, View, Insert, Runtime, Tools, Help, All changes saved.
- Code Cell:** Contains the provided Python code for invoking the model and printing its response.
- Output Cell:** Shows the execution status: ✓ 6s completed at 4:03PM.
- Runtime Tab:** RAM Disk Gemini
- Help Tab:** A purple circular icon with a question mark.

8. Selecciona el botón **Reproducir** para ejecutar la solicitud.



```
# {reviews_text}
# """
# response = output.invoke(multitask_prompt)
# print("Granite Model Response for multitask response:\n")
# print(response)

# Define the example to guide the model
formatted_prompt = f"""
Classify these reviews as Positive, Negative, or Mixed, and tag relevant
categories. Use this format:
- Sentiment: [Sentiment]
- Categories: [Categories].
{reviews_text}
"""

# Invoke the model with prompt
response = output.invoke(formatted_prompt)

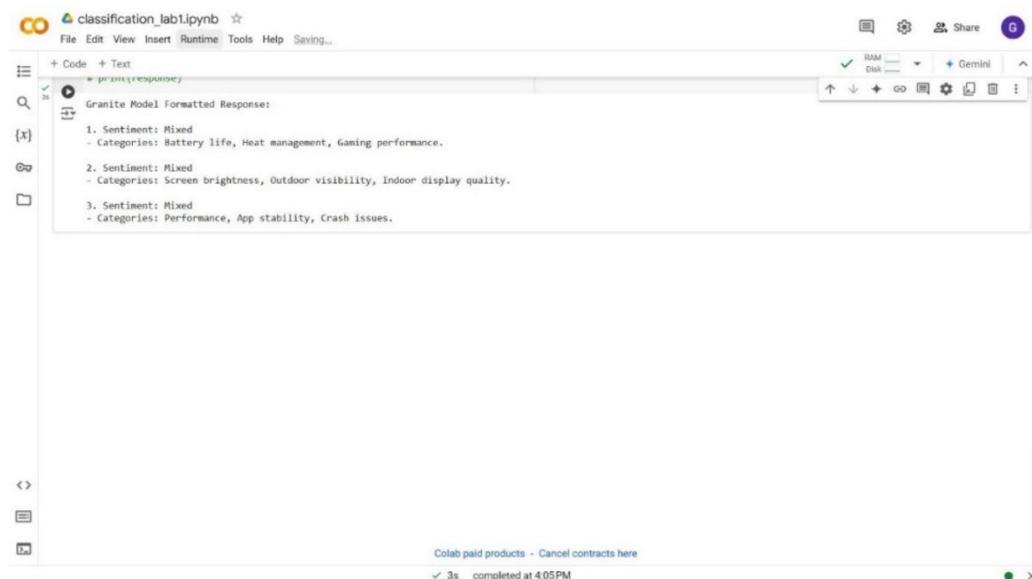
# Print the response
print("Granite Model Formatted Response:\n")
print(response)

# Define the example to guide the model
# example_input = "The battery is excellent, but the screen brightness is poor."
# example_output = {"Sentiment": "Mixed", "Categories": "Battery Life, Screen Quality"}

# example_prompt = f"""
# Classify these reviews as Positive, Negative, or Mixed, and tag relevant focus areas.
# """
```

9. Lee el resultado que genera el modelo. Ten en cuenta que el resultado final es preciso y bien estructurado. El sentimiento de cada reseña se clasifica correctamente como "mixto" y las categorías pertinentes están claramente etiquetadas. El formato es consistente en todas las reseñas: cada una de ellas enumera su sentimiento seguido de las categorías correspondientes de manera clara y organizada. Esta estructura mejora la legibilidad y garantiza que el resultado sea fácil de interpretar. En general, la respuesta es precisa y se ajusta al formato requerido, lo que la hace exacta y bien estructurada.

Has completado la tarea de clasificación. A continuación, resumamos la transcripción de la reunión.



```
classification_lab1.ipynb
File Edit View Insert Runtime Tools Help Saving...
+ Code + Text
Granite Model Formatted Response:
{x}
1. Sentiment: Mixed
- Categories: Battery life, Heat management, Gaming performance.
2. Sentiment: Mixed
- Categories: Screen brightness, Outdoor visibility, Indoor display quality.
3. Sentiment: Mixed
- Categories: Performance, App stability, Crash issues.

Colab paid products - Cancel contracts here
3s completed at 4:05PM
```

Paso 6: Proporcionar una transcripción de la reunión para su resumen y probar la solicitud inicial**Descripción**

En este paso, ingresarás una transcripción de la reunión para su resumen, escribirás una solicitud básica para resumir la reunión y analizarás las deficiencias del resultado.

Instrucciones

1. Ingresá la transcripción de la reunión simulando una reunión semanal. Escribe esta transcripción en el campo Instrucción:

```
customer_meetings = [  
    """
```

La reunión comenzó con una discusión sobre el presupuesto de marketing del tercer trimestre. Se decidió que el 40 % del presupuesto se destinará a anuncios digitales, el 30 % a eventos y el 30 % a campañas en redes sociales. El equipo enfatizó la necesidad de concretar asociaciones con influencers para aumentar la visibilidad de la marca y el marketing por correo electrónico para impulsar la interacción directa. El próximo mes se lanzará un programa piloto para probar nuevos formatos de anuncios y el equipo revisará los resultados a finales del tercer trimestre.

Posteriormente, el equipo discutió las métricas de rendimiento de la campaña. El seguimiento del ROI será una máxima prioridad y se realizarán ajustes en función de los datos de rendimiento.

El equipo de eventos planteó inquietudes sobre la asignación de recursos para las próximas ferias comerciales y se acordó que se reasignarían \$10.000 adicionales para cubrir estos gastos.

Por último, el equipo revisó nuevos conceptos creativos para la próxima campaña y decidió continuar con el Concepto 8, que tuvo mejores resultados entre los focus groups. Se ultimaron los plazos para la entrega de los activos de campaña: todos los entregables deben presentarse a más tardar el 15 de julio.

```
    """
```

```
]
```

```

File Edit View Insert Runtime Tools Help Last edited on December 2
+ Code + Text
[ ] output = Replicate(
    model=model,
    replicate_api_token=api_token,
)
{x}
# Define the customer reviews
customer_meetings = [
    """
    The meeting began with a discussion of the Q3 marketing budget. It
    was decided that 40% of the budget will go to digital ads,
    30% to events, and 30% to social media campaigns. The team emphasized the
    need for influencer partnerships to increase brand visibility
    and email marketing to boost direct engagement. A pilot program to test
    new ad formats will launch next month, with the team reviewing results
    by the end of Q3.

    Later, the team discussed campaign performance metrics. ROI monitoring
    will be a top priority, and adjustments will be made based on performance
    data.
    The events team raised concerns about resource allocation for upcoming
    trade shows, and it was agreed that an additional $10,000 would be
    reallocated to
    cover these costs.

    Lastly, the team reviewed new creative concepts for the upcoming campaign,
    deciding to proceed with Concept B, which tested better among focus groups.
    Deadlines for campaign assets were finalized: all deliverables must be
    submitted by July 15.
    """
]
# Refine the prompt to include reviews
reviews_text = "\n".join([f"Review {i+1}: {review}" for i, review
in enumerate(customer_meetings)])

```

2. Ahora, ingresa la solicitud para generar un resumen de la transcripción de la reunión. Escribe esta solicitud en el campo Instrucción:

```

# Refinar la solicitud para incluir reseñas
reviews_text = "\n".join([f"Review {i+1}: {review}" for i, review
in enumerate(customer_meetings)])

prompt = f"""
Resume esta reunión:

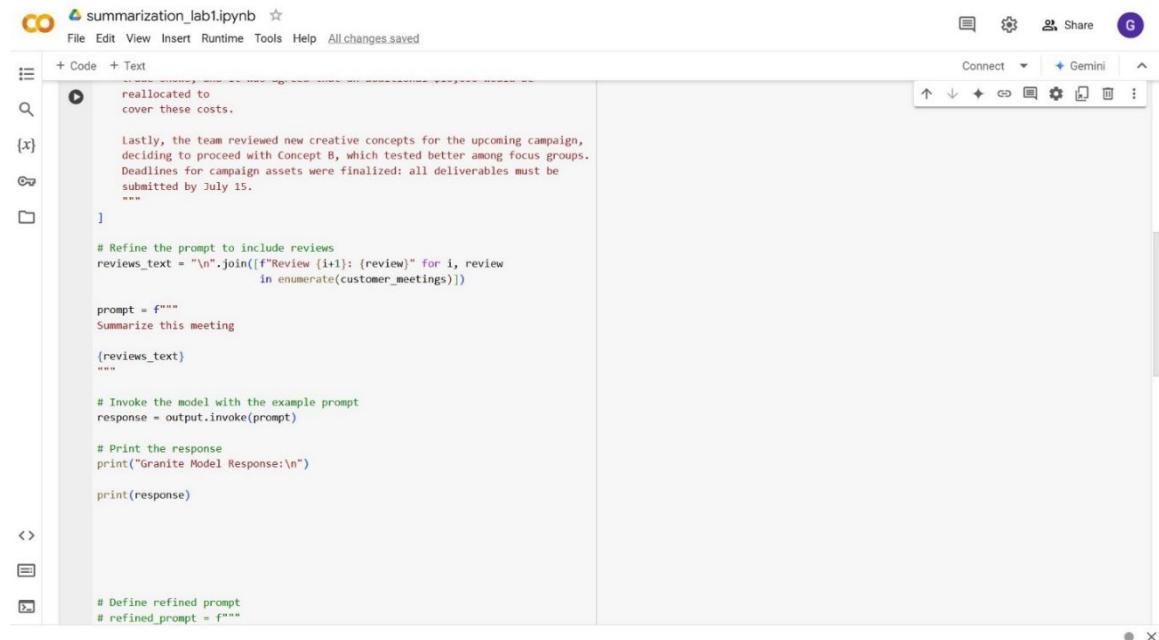
{reviews_text}
"""

# Invocar el modelo con la solicitud de ejemplo
respuesta = output.invoke(prompt)

# Imprimir la respuesta
print("Granite Model Response:\n")

print(response)

```



```
summarization_lab1.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
reallocated to
cover these costs.

Lastly, the team reviewed new creative concepts for the upcoming campaign,
deciding to proceed with Concept B, which tested better among focus groups.
Deadlines for campaign assets were finalized: all deliverables must be
submitted by July 15.
"""

]

# Refine the prompt to include reviews
reviews_text = "\n".join([f"Review {i+1}: {review}" for i, review
in enumerate(customer_meetings)])

prompt = """
Summarize this meeting

(reviews_text)
"""

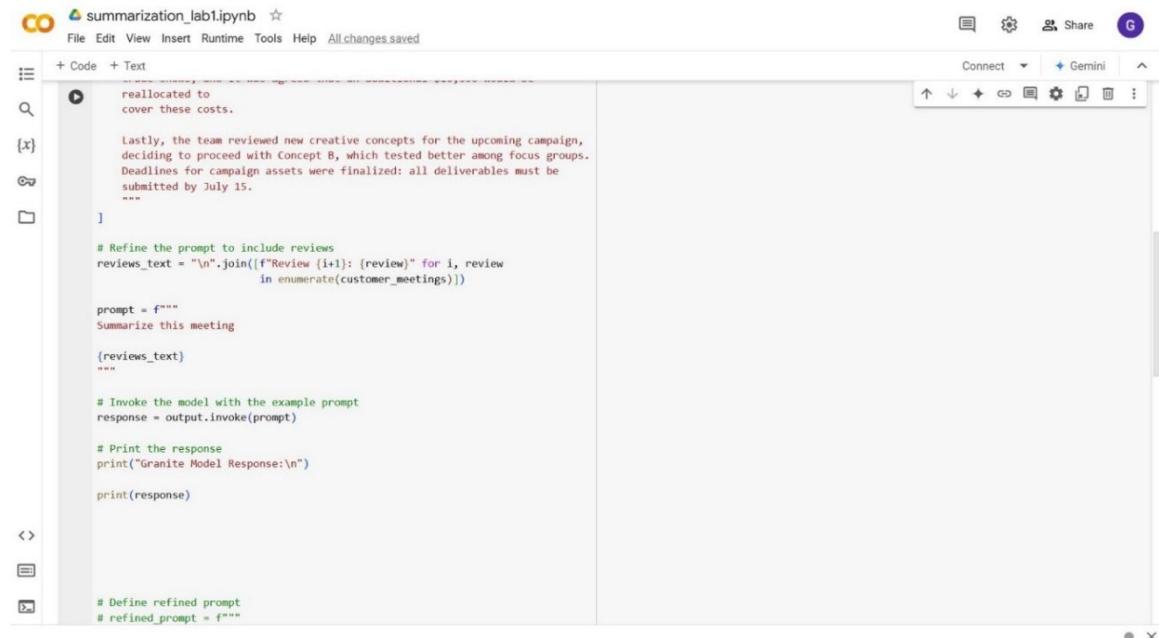
# Invoke the model with the example prompt
response = output.invoke(prompt)

# Print the response
print("Granite Model Response:\n")

print(response)

# Define refined prompt
# refined_prompt = """
```

3. Selecciona el botón **Reproducir** para ejecutar la solicitud.



```
summarization_lab1.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
reallocated to
cover these costs.

Lastly, the team reviewed new creative concepts for the upcoming campaign,
deciding to proceed with Concept B, which tested better among focus groups.
Deadlines for campaign assets were finalized: all deliverables must be
submitted by July 15.
"""

]

# Refine the prompt to include reviews
reviews_text = "\n".join([f"Review {i+1}: {review}" for i, review
in enumerate(customer_meetings)])

prompt = """
Summarize this meeting

(reviews_text)
"""

# Invoke the model with the example prompt
response = output.invoke(prompt)

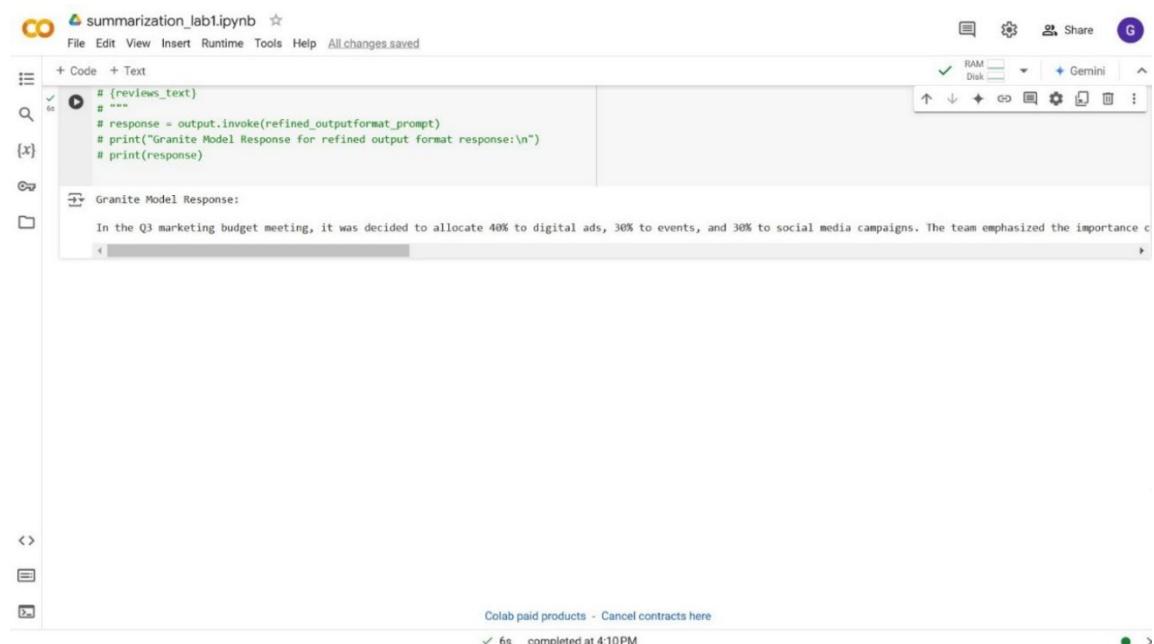
# Print the response
print("Granite Model Response:\n")

print(response)

# Define refined prompt
# refined_prompt = """
```

4. Lee el resultado que genera el modelo. Ten en cuenta que el resumen es demasiado amplio, carece de estructura y omite detalles importantes de la transcripción de la reunión original. No menciona temas importantes como los problemas de los clientes con el rendimiento de la batería y la durabilidad de la pantalla, los planes del equipo de ingeniería para la optimización y las pruebas de estrés, ni los plazos específicos para la finalización y prueba del prototipo. El resultado generado se centra en un tema diferente, lo que lleva a la omisión de puntos cruciales discutidos en la reunión real.

Ahora que tienes el resumen inicial, puedes revisarlo y trabajar para refinar la solicitud y mejorar el resultado.



```
# {reviews_text}
# ...
# response = output.invoke(refined_outputformat_prompt)
# print("Granite Model Response for refined output format response:\n")
# print(response)

Granite Model Response:
In the Q3 marketing budget meeting, it was decided to allocate 40% to digital ads, 30% to events, and 30% to social media campaigns. The team emphasized the importance of...
```

Paso 7: Refinar la solicitud para mejorar el resultado del resumen

Descripción

En este paso, refinarás la solicitud paso a paso para mejorar la calidad del resultado del resumen.

Instrucciones

1. **Recuerda:** Especificar la extensión deseada del resumen ayuda a que el modelo se centre en proporcionar una descripción general concisa pero completa, garantizando que se plasmen los puntos importantes sin detalles excesivos.

Actualiza la solicitud especificando la extensión de resumen deseada. Escribe esta solicitud en el campo **Instrucción:**

```
# Definir una solicitud refinada
refined_prompt = f"""
Summarize this meeting in three sentences:
{reviews_text}
"""

# Invocar el modelo con la solicitud refinada
respuesta = output.invoke(refined_prompt)

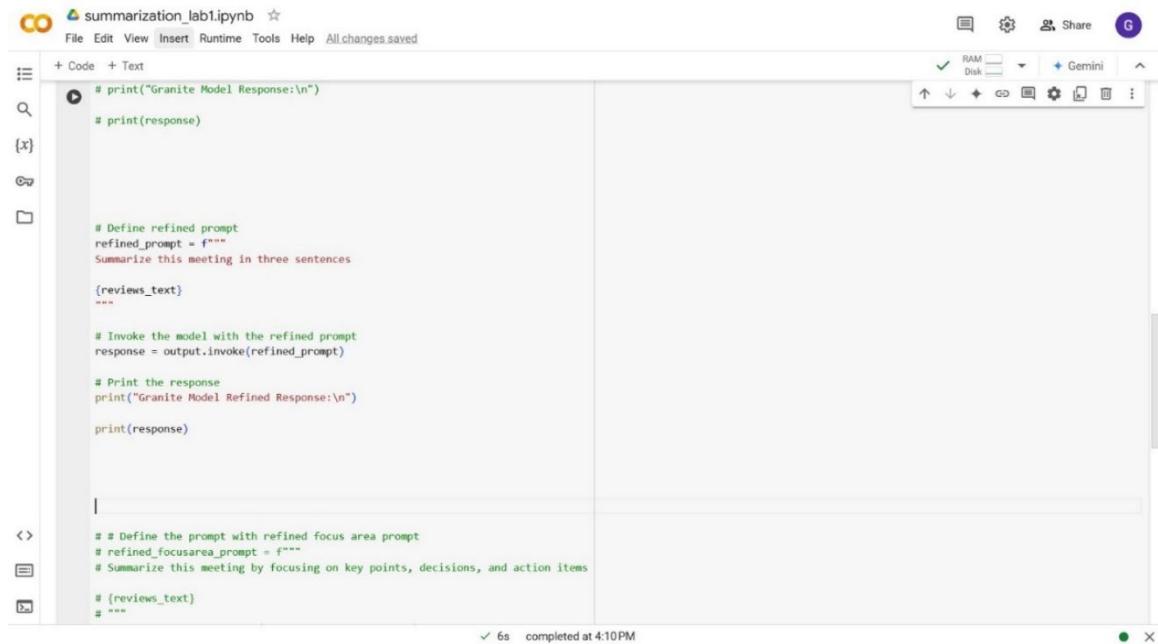
# Imprimir la respuesta
print("Granite Model Refined Response:\n")

print(response)
```

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** The notebook is titled "summarization_lab1.ipynb".
- File Menu:** File, Edit, View, Insert, Runtime, Tools, Help, All changes saved.
- Toolbar:** Share, RAM, Disk, Gemini.
- Code Cell:** Contains the Python code provided above, which defines a refined prompt to summarize a meeting in three sentences.
- Output Cell:** Below the code cell, there is a large empty area where the output will be displayed.
- Bottom Status Bar:** Shows "6s completed at 4:10PM".

2. Selecciona el botón **Reproducir** para ejecutar la solicitud.



The screenshot shows a Jupyter Notebook interface with the file name "summarization_lab1.ipynb". The code cell contains the following Python script:

```
# print("Granite Model Response:\n")
# print(response)

# Define refined prompt
refined_prompt = f"""
Summarize this meeting in three sentences

{reviews_text}
"""

# Invoke the model with the refined prompt
response = output.invoke(refined_prompt)

# Print the response
print("Granite Model Refined Response:\n")
print(response)

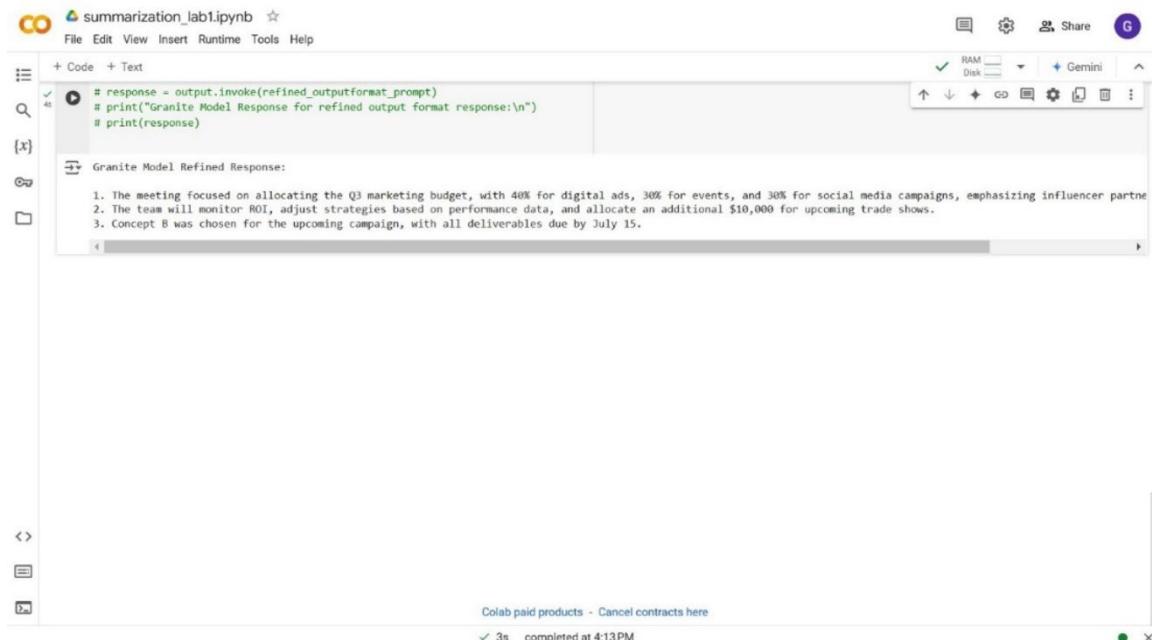
# # Define the prompt with refined focus area prompt
# refined_focusarea_prompt = f"""
# Summarize this meeting by focusing on key points, decisions, and action items

# {reviews_text}
# """
```

The notebook interface includes a toolbar with File, Edit, View, Insert, Runtime, Tools, Help, and a status bar indicating "6s completed at 4:10PM".

3. Lee el resultado que genera el modelo. Observa que el resumen es más conciso y pertinente, pero aun así carece de claridad a la hora de distinguir entre puntos de discusión, decisiones y acciones pendientes. Si bien el resumen aborda brevemente la asignación del presupuesto de marketing, las estrategias y los conceptos de campaña, no separa claramente los diferentes aspectos de la reunión, como lo que se discutió, las decisiones tomadas y las acciones específicas asignadas a los miembros del equipo. Un enfoque más estructurado mejoraría la claridad y el enfoque del resumen.

Puedes mejorar esto definiendo los aspectos prioritarios en el siguiente paso.



```
# response = output.invoke(refined_outputformat_prompt)
# print("Granite Model Response for refined output format response:\n")
# print(response)

{x}
Granite Model Refined Response:
1. The meeting focused on allocating the Q3 marketing budget, with 40% for digital ads, 30% for events, and 30% for social media campaigns, emphasizing influencer partnerships.
2. The team will monitor ROI, adjust strategies based on performance data, and allocate an additional $10,000 for upcoming trade shows.
3. Concept B was chosen for the upcoming campaign, with all deliverables due by July 15.
```

4. **Recuerda:** Definir los aspectos prioritarios, como los puntos clave discutidos, las decisiones tomadas y las acciones pendientes, ayuda al modelo a mantenerse enfocado en los aspectos más críticos de la transcripción de la reunión. Esto garantiza que el resumen capte la esencia de la conversación y la organice de una manera que sea fácil de interpretar y seguir.

Actualiza la solicitud definiendo los aspectos prioritarios: puntos clave discutidos, decisiones y acciones pendientes. Escribe esta solicitud en el campo Instrucción:

```
# Definir la solicitud con una solicitud refinada de aspectos prioritarios
```

```
refined_focusarea_prompt = f"""
```

Resume esta reunión centrándote en los puntos clave, las decisiones tomadas y las acciones pendientes:

```
{reviews_text}
"""

```

```
respuesta = output.invoke(refined_focusarea_prompt)
```

```
print("Granite Model Response for refined focus area response:\n")
print(response)
```

The screenshot shows a Jupyter Notebook interface with a single code cell containing Python code. The code defines two prompts: one for a refined focus area and another for a refined output format. It then uses the `output.invoke` method to generate responses and prints them. A section titled "Granite Model Refined Response" contains the generated summary text.

```
summarization_lab1.ipynb
File Edit View Insert Runtime Tools Help Saving...
+ Code + Text
{x}
# Define the prompt with refined focus area prompt
refined_focusarea_prompt = f"""
Summarize this meeting by focusing on key points, decisions, and action items

{reviews_text}
"""

response = output.invoke(refined_focusarea_prompt)
print("Granite Model Response for refined focus area response:\n")
print(response)

# # Define the prompt with refined output prompt
# refined_outputformat_prompt = f"""
# Summarize this meeting into a structured format using the following
# headings: Key Points Discussed, Decisions Made, and Action Items. Mention
# timelines. Include only two concise bullet points under each heading

# {reviews_text}
# """
# response = output.invoke(refined_outputformat_prompt)
# print("Granite Model Response for refined output format response:\n")
# print(response)

Granite Model Refined Response:
1. The meeting focused on allocating the Q3 marketing budget... with 40% for digital ad... 30% for events... and 30% for social media campaigns... emphasizing influencer partnerships...
```

5. Selecciona el botón Reproducir para ejecutar la solicitud.

This screenshot is identical to the one above, showing the same Jupyter Notebook code cell and its output. The difference is that the "Granite Model Refined Response" section is now empty, indicating that the execution has been completed.

6. Lee el resultado que genera el modelo. Ten en cuenta que el resultado capta los elementos clave de manera más eficaz, pero aun así carece de un formato organizado para facilitar la lectura. Para mejorar esto, puedes actualizar la solicitud definiendo un formato de resultado claro.

The screenshot shows a Jupyter Notebook interface with the file name 'summarization_lab1.ipynb'. The notebook contains a single code cell with the following Python code:

```

# Define the request with a refined result format
refined_outputformat_prompt = f"""
Resume this meeting in a structured format using the following titles: Key points discussed, Decisions made, and Outstanding actions. Mention the timelines.
Include only two brief summaries for each title.

{reviews_text}
"""

response = output.invoke(refined_outputformat_prompt)

```

The output of the code cell is displayed below the code, showing a well-organized meeting summary:

```

Granite Model Response for refined focus area response:
Meeting Summary:
1. Q3 Marketing Budget Allocation:
   - 40% for digital ads
   - 30% for events
   - 30% for social media campaigns

2. Key Strategies:
   - Influencer partnerships to increase brand visibility
   - Email marketing for direct engagement
   - Pilot program for new ad formats to launch next month, with results reviewed by end of Q3

3. Campaign Performance Metrics:
   - ROI monitoring to be a top priority
   - Adjustments to be made based on performance data

4. Resource Allocation:
   - Additional $10,000 reallocated to cover trade show costs

5. Creative Concepts:
   - Concept B chosen for upcoming campaign
   - Deadlines finalized: all deliverables must be submitted by July 15

```

The notebook interface includes a toolbar at the top with File, Edit, View, Insert, Runtime, Tools, Help, and a Share button. The bottom status bar indicates the execution time was 4s and completed at 4:15PM.

7. **Recuerda:** Definir un formato de resultado estructurado con títulos y viñetas concisas garantiza que el resumen esté organizado, lo que facilita la comprensión de la información principal. Al limitar las viñetas a dos por sección, puedes mantener el foco en los aspectos más importantes de la reunión sin abrumar al lector con detalles excesivos.

Actualiza la solicitud definiendo el formato del resultado: utiliza títulos estructurados con un límite de dos viñetas en cada sección. Escribe esta solicitud en el campo Instrucción:

```

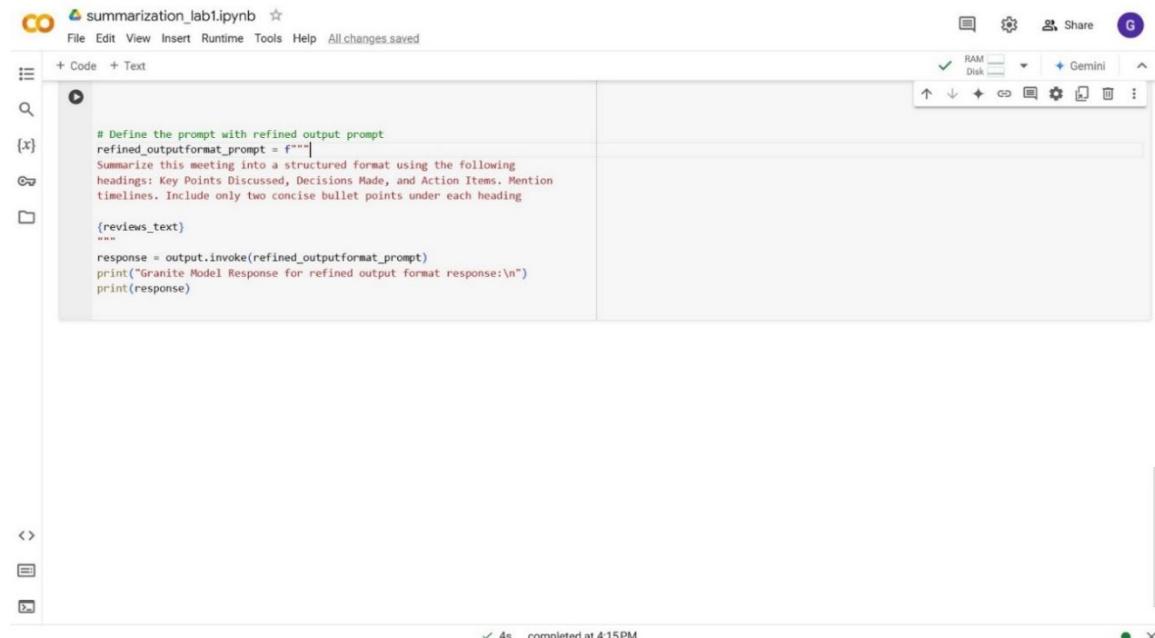
# Define the request with a refined result format
refined_outputformat_prompt = f"""
Resume this meeting in a structured format using the following titles: Key points discussed, Decisions made, and Outstanding actions. Mention the timelines.
Include only two brief summaries for each title.

{reviews_text}
"""

response = output.invoke(refined_outputformat_prompt)

```

```
print("Granite Model Response for refined output format\nresponse:\n")\nprint(response)
```

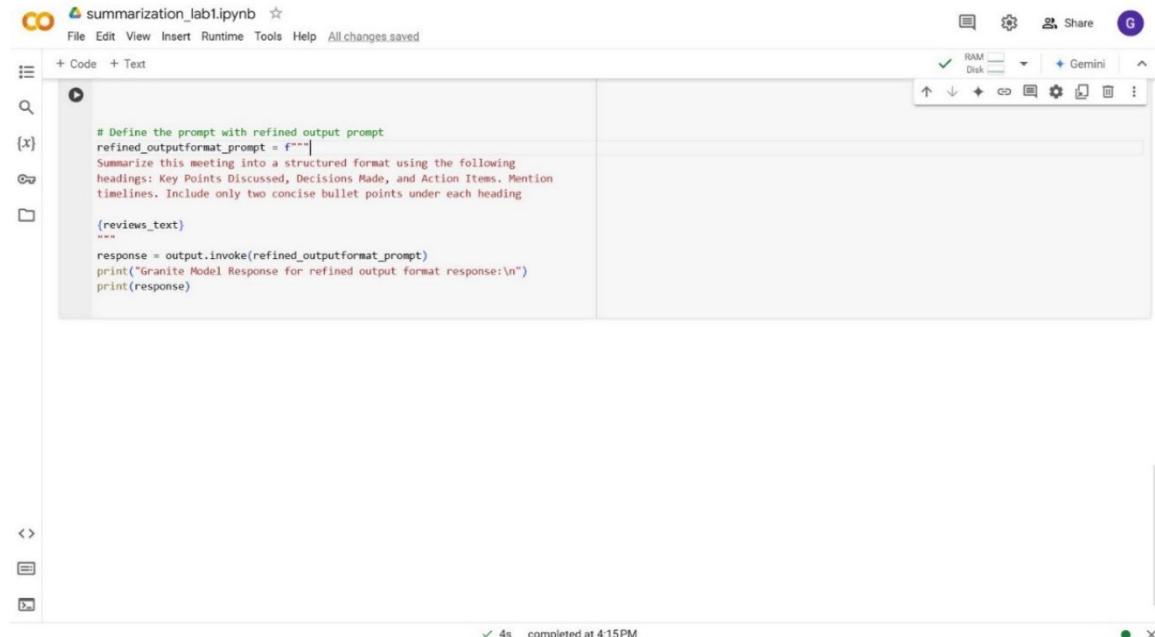


The screenshot shows a Jupyter Notebook cell with the following code:

```
# Define the prompt with refined output prompt\nrefined_outputformat_prompt = f"""|\nSummarize this meeting into a structured format using the following\nheadings: Key Points Discussed, Decisions Made, and Action Items. Mention\ntimelines. Include only two concise bullet points under each heading\n\n{reviews_text}\n***\nresponse = output.invoke(refined_outputformat_prompt)\nprint("Granite Model Response for refined output format response:\n")\nprint(response)
```

The cell has been run successfully, as indicated by the green checkmark and the timestamp "4s completed at 4:15PM".

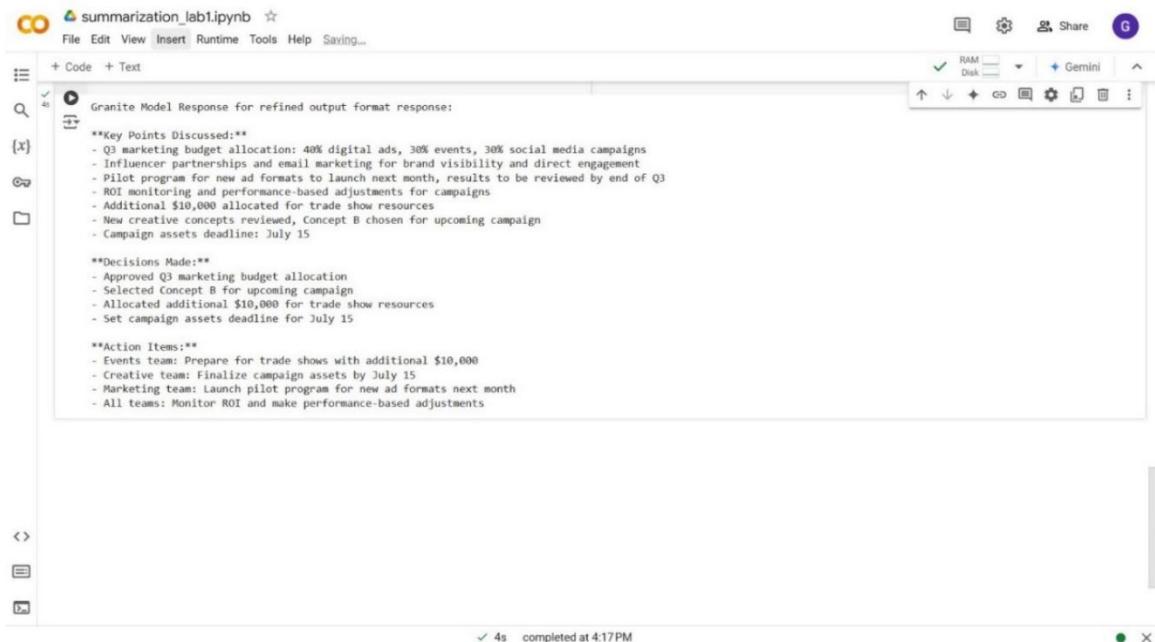
8. Selecciona el botón **Reproducir** para ejecutar la solicitud.



The screenshot shows the same Jupyter Notebook cell as the previous one, but it has not yet been run. The code is identical:

```
# Define the prompt with refined output prompt\nrefined_outputformat_prompt = f"""|\nSummarize this meeting into a structured format using the following\nheadings: Key Points Discussed, Decisions Made, and Action Items. Mention\ntimelines. Include only two concise bullet points under each heading\n\n{reviews_text}\n***\nresponse = output.invoke(refined_outputformat_prompt)\nprint("Granite Model Response for refined output format response:\n")\nprint(response)
```

9. Lee el resultado que genera el modelo. Ten en cuenta que el resultado final está estructurado, es conciso y práctico. Organiza la información clave bajo encabezados claros, lo que les facilita a las partes interesadas la revisión. El uso de viñetas garantiza que el contenido sea fácil de comprender, y la separación de puntos clave, decisiones y acciones pendientes permite una referencia rápida. Este formato mejora la legibilidad y ayuda a las partes interesadas a identificar rápidamente la información importante, garantizando la claridad y una comunicación eficaz.



The screenshot shows a Jupyter Notebook interface with the file name 'summarization_lab1.ipynb'. The notebook contains a single code cell with the following content:

```
Granite Model Response for refined output format response:  
**Key Points Discussed:**  
- Q3 marketing budget allocation: 40% digital ads, 30% events, 30% social media campaigns  
- Influencer partnerships and email marketing for brand visibility and direct engagement  
- Pilot program for new ad formats to launch next month, results to be reviewed by end of Q3  
- ROI monitoring and performance-based adjustments for campaigns  
- Additional $10,000 allocated for trade show resources  
- New creative concepts reviewed, Concept B chosen for upcoming campaign  
- Campaign assets deadline: July 15  
**Decisions Made:**  
- Approved Q3 marketing budget allocation  
- Selected Concept B for upcoming campaign  
- Allocated additional $10,000 for trade show resources  
- Set campaign assets deadline for July 15  
**Action Items:**  
- Events team: Prepare for trade shows with additional $10,000  
- Creative team: Finalize campaign assets by July 15  
- Marketing team: Launch pilot program for new ad formats next month  
- All teams: Monitor ROI and make performance-based adjustments
```

The notebook has a status bar at the bottom indicating '✓ 4s completed at 4:17PM'.

Conclusión

En este laboratorio, tuviste dos responsabilidades principales como gerente de producto: automatizar la clasificación de las reseñas de los clientes y resumir las reuniones semanales. Al refinar las solicitudes paso a paso, garantizaste resultados precisos y estructurados que satisfacen las necesidades de las partes interesadas. Estas habilidades ahorran tiempo, mejoran la precisión y permiten una toma de decisiones más rápida.

A continuación, considera examinar cómo integrar estas soluciones en flujos de trabajo del mundo real, como conectar flujos de datos en vivo o mejorar los sistemas de informes.

© Copyright IBM Corporation 2025.

La información contenida en estos materiales se proporciona únicamente con fines informativos y se proporciona TAL CUAL, sin garantía de ningún tipo, ni expresa ni implícita. IBM no será responsable de ningún daño que surja del uso de estos materiales o que de otro modo esté relacionado con ellos. Nada de lo contenido en estos materiales tiene como propósito, ni tendrá el efecto de, constituir garantías o hacer declaraciones de parte IBM o sus proveedores o licenciantes, ni alterar los términos y condiciones del acuerdo de licencia aplicable que rige el uso del software de IBM. Las referencias que se hagan en estos materiales a productos, programas o servicios de IBM no implican que estarán disponibles en todos los países en los que IBM opera. Esta información se basa en los planes y la estrategia de productos actuales de IBM, que están sujetos a cambios por parte de IBM sin previo aviso. Las fechas de lanzamiento de productos o las capacidades a las que se hace referencia en estos materiales pueden cambiar en cualquier momento a entera discreción de IBM en función de las oportunidades del mercado u otros factores, y no tienen por objeto ser un compromiso con la disponibilidad futura de productos o características de modo alguno.

IBM, el logotipo de IBM e ibm.com son marcas registradas de International Business Machines Corp., registradas en muchas jurisdicciones de todo el mundo. Otros nombres de productos y servicios pueden ser marcas registradas de IBM o de otras empresas. Una lista actual de las marcas registradas de IBM está disponible en la web en "Información de copyright y marcas registradas" en www.ibm.com/legal/copytrade.shtml.



Please Recycle
