

LAB

Develop prompt templates
using variables to generate
output from LLMs

Introduction	3
Software requirements.....	3
Objective.....	3
Lab steps	3
Scenario.....	4
Background information.....	4
Challenge.....	4
Solution	4
Create a GitHub account	5
Overview.....	5
Instructions.....	5
Create a Replicate account	9
Overview.....	9
Instructions.....	9
Sign up for Google Colab	13
Overview.....	13
Instructions.....	13
Load the Jupyter notebook and initialize the model	17
Overview.....	17
Instructions.....	17
Draft a baseline prompt for a recurring task	23
Overview.....	23
Instructions.....	23
Identify variable elements and convert to a template	25
Overview.....	25
Instructions.....	25
Test and finalize the prompt template	27
Overview.....	27
Instructions.....	27
Conclusion.....	31

Introduction

In this lab, you will identify variables and develop prompt templates to generate relevant output from large language models (LLMs).

Software requirements

To complete this lab, you don't have to install any software on your computer. You only require a Google account to access Google Colab and a Replicate account to access and use various artificial intelligence (AI) models.

Objective

After completing this lab, you should be able to:

- Develop prompt templates using variables to generate output from LLMs

Lab steps

This lab requires you to complete the following procedures:

1. Create a GitHub account
2. Create a Replicate account
3. Sign up for Google Colab
4. Load the Jupyter notebook and initialize the model
5. Draft a baseline prompt for a recurring task
6. Identify variable elements and convert to a template
7. Test and finalize the prompt template

Estimated duration to complete

20 minutes

Scenario**Background information**

Codal Technologies is expanding and offering consultation services to some of its major clients. With more stakeholders involved, analysts need to produce accurate weekly project updates for them.

Challenge

However, the variety of formats and prompting styles has led to inconsistencies in quality, tone, and completeness of these outputs.

Solution

To address this, your manager asks you, a senior analyst, to create the firm's first prompt template. This template will streamline the workflow by allowing analysts to update only a few project-specific variables while maintaining a uniform structure. The goal is to deliver high-quality and consistent communication, no matter who prepares the update or which client it is for.

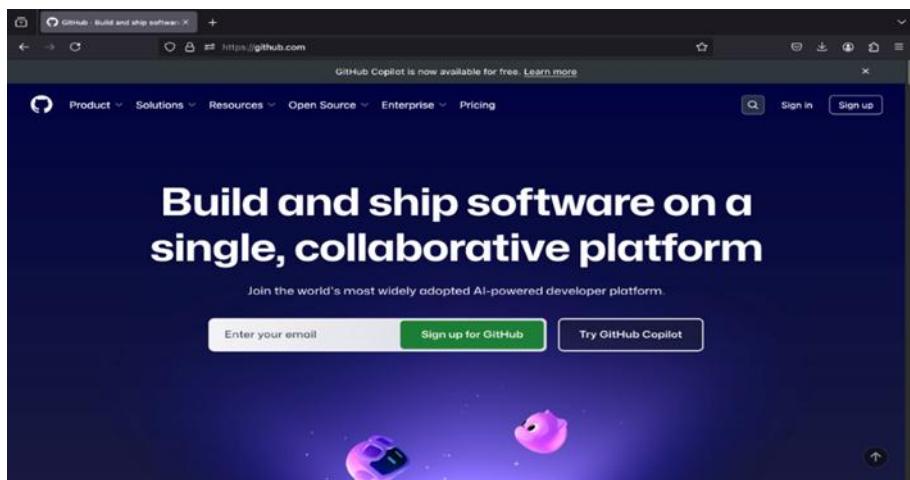
Create a GitHub account

Overview

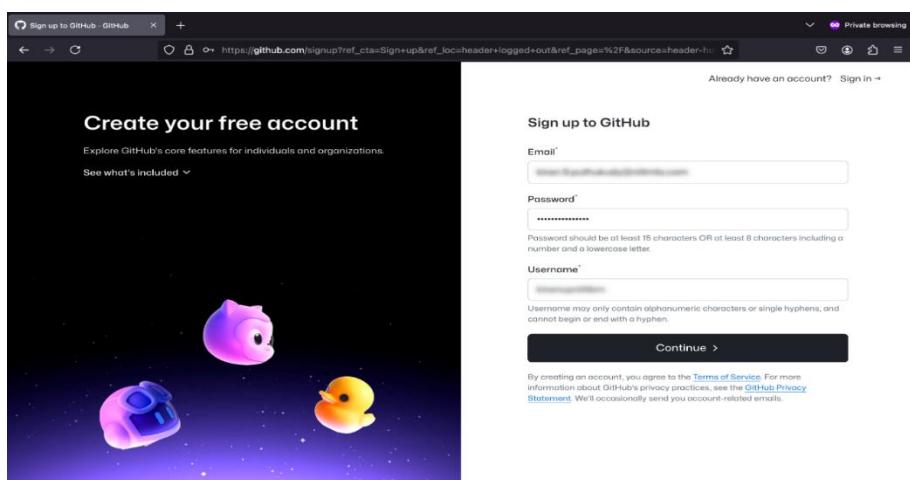
In this procedure, you'll set up a GitHub account. GitHub is a platform that helps developers store, manage, and share code, while also supporting collaboration through tools such as version control, bug tracking, and task management. Setting up a GitHub account provides access to the Replicate platform, which is required to complete the lab efficiently.

Instructions

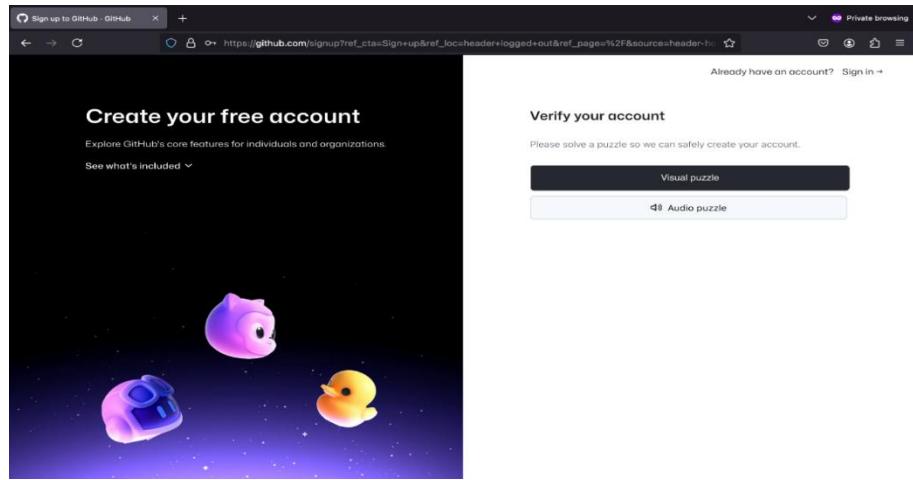
1. To create a GitHub account, go to the [GitHub](#) website.
2. Select the **Sign up for GitHub** button, as shown in the following screenshot.



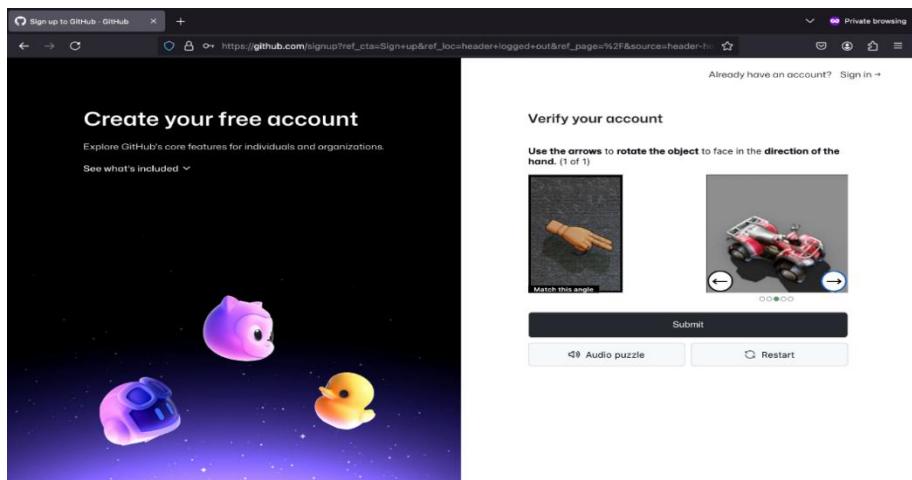
3. The following screenshot shows the "Sign up to GitHub" page. Enter your details in the **Email**, **Password**, and **Username** fields. Then, select **Continue**.



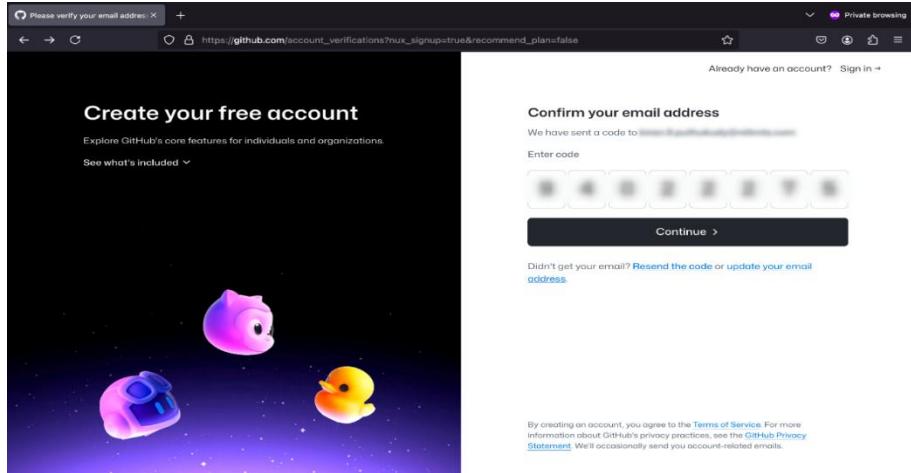
4. To verify your account, select the **Visual puzzle** button.



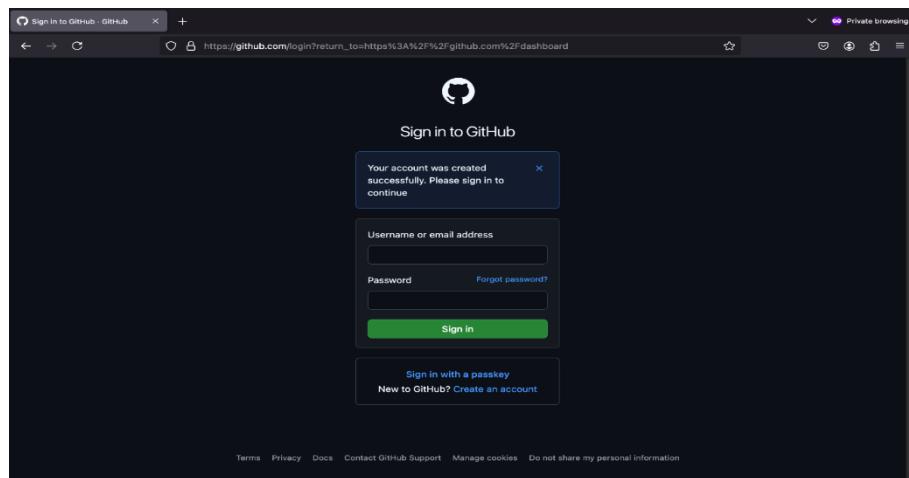
5. Solve the visual puzzle and select **Submit**.



6. To confirm your email, enter the confirmation code sent to your registered email in the **Enter code** field and select **Continue**.

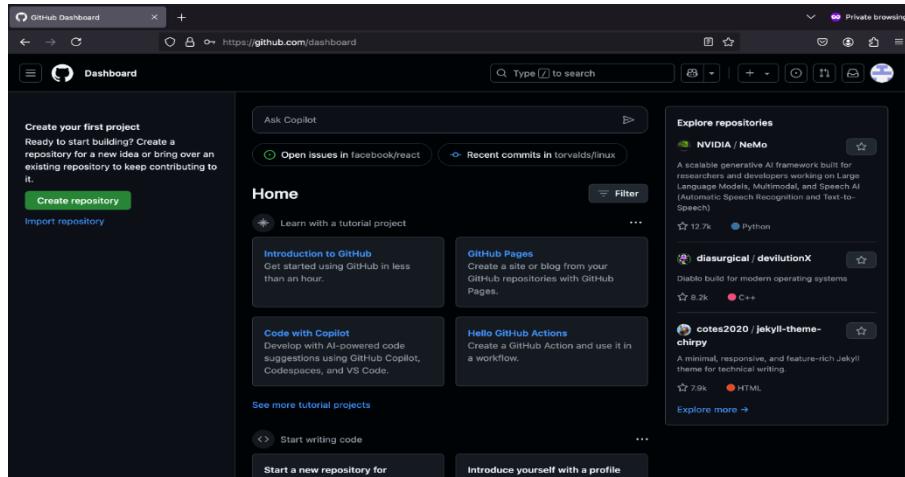


7. After your GitHub account has been successfully created, a confirmation message is displayed as shown in the following screenshot.



8. To sign in to your account, enter your credentials in the **Username or email address** and **Password** fields, and then select **Sign in**.

9. After you sign in, the GitHub dashboard displays as shown in the following screenshot.



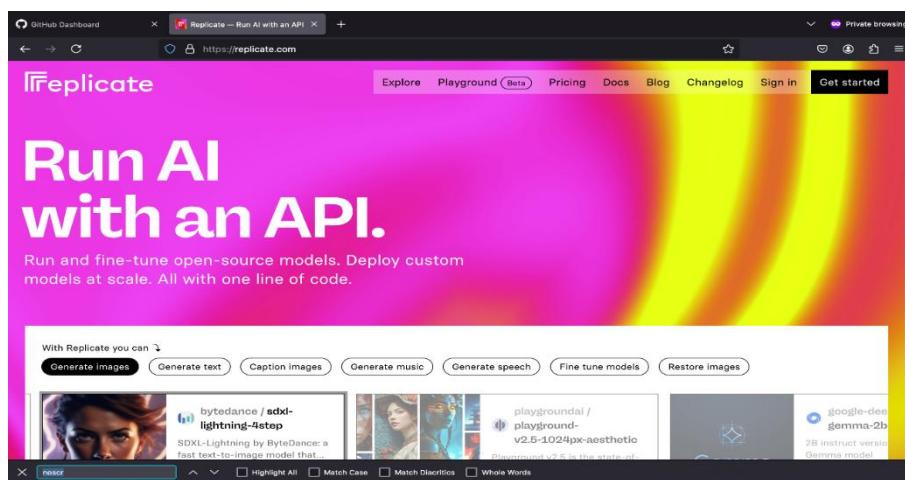
Create a Replicate account

Overview

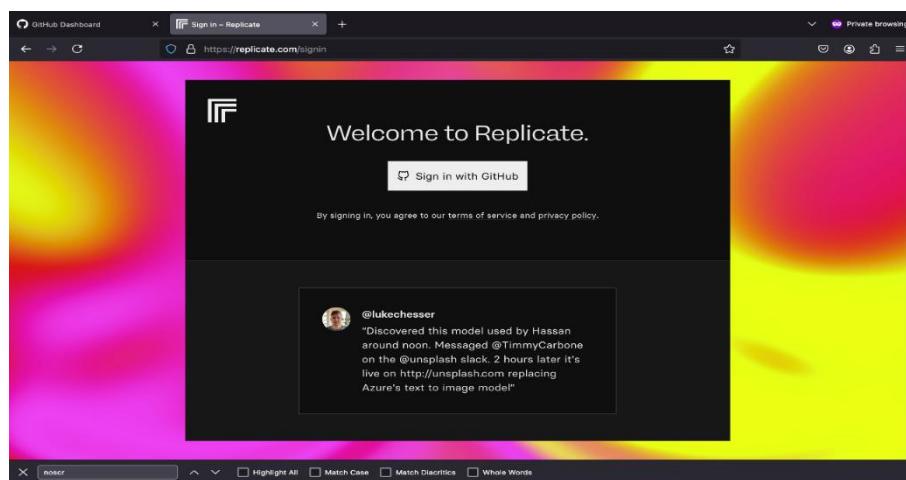
In this procedure, you'll use your GitHub account to register for a Replicate account. Replicate is a cloud-based platform that lets you use AI models such as IBM Granite without requiring advanced hardware. As part of this procedure, you'll create a Replicate token, a secure access key that allows the lab environment to authenticate with Replicate and run models from your account in Google Colab.

Instructions

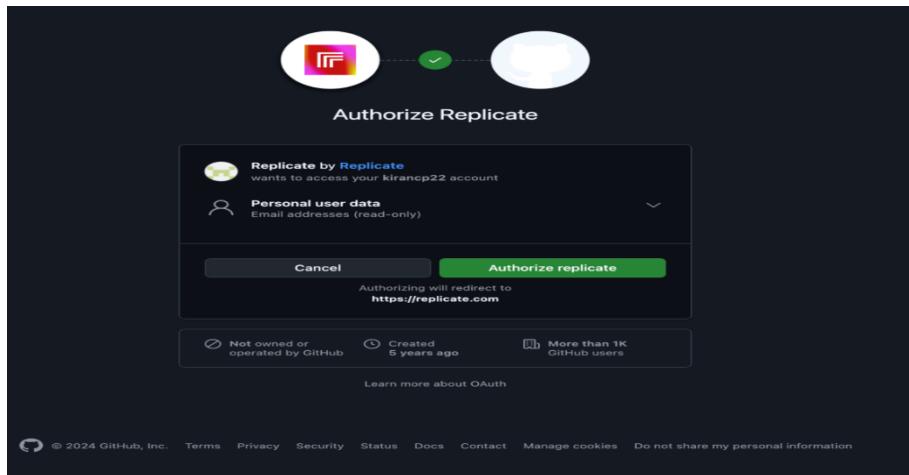
1. Go to the [Replicate](#) website.
2. Select **Get started**, as shown in the following screenshot.



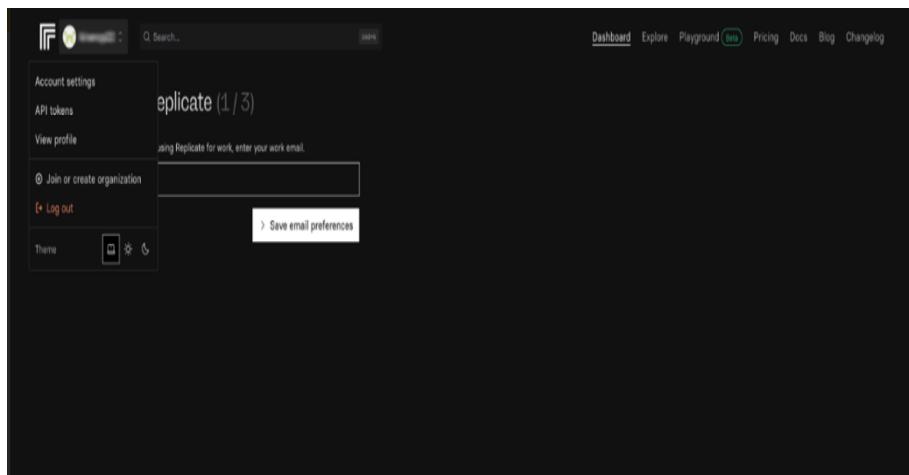
3. Select the **Sign in with GitHub** button, as shown in the following screenshot.



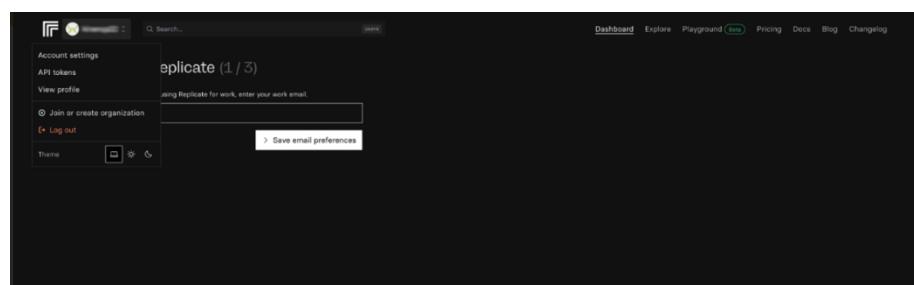
4. Select **Authorize replicate** to continue, as shown in the following screenshot.



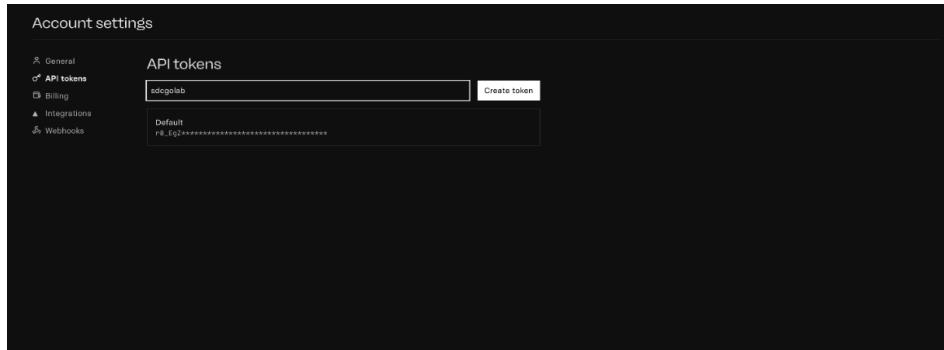
5. After your Replicate account is created, the Replicate dashboard displays as shown in the following screenshot. To create a Replicate token, select the **Account settings** option from the navigation menu.



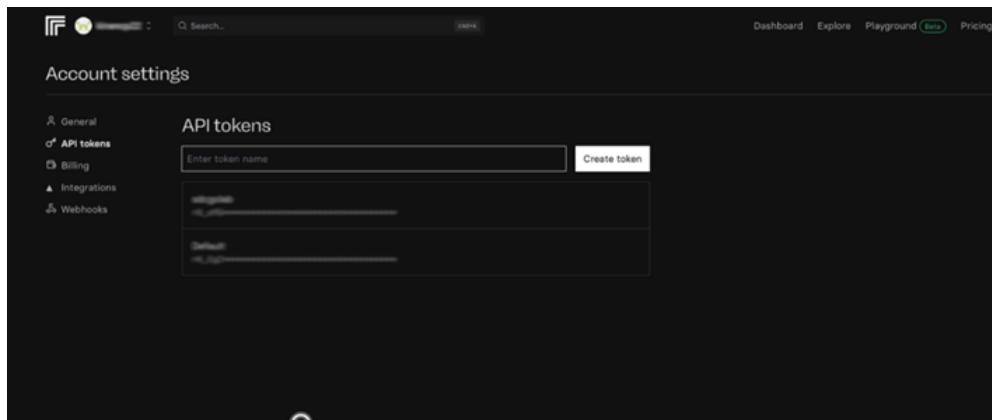
6. The following screenshot shows the Replicate dashboard. Select **API tokens** from the "Account settings" panel.



7. Enter a name for the token in the **API tokens** field and then select **Create token**, as shown in the following screenshot.

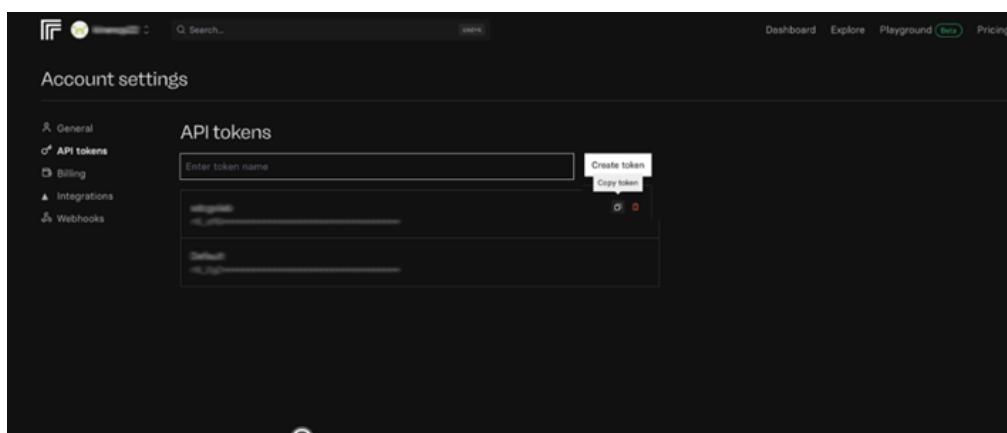


8. After your API token is created, it is displayed on the “Account settings” page as shown in the following screenshot.



9. To copy the Replicate API, select the **Copy token** icon, as shown in the following screenshot.

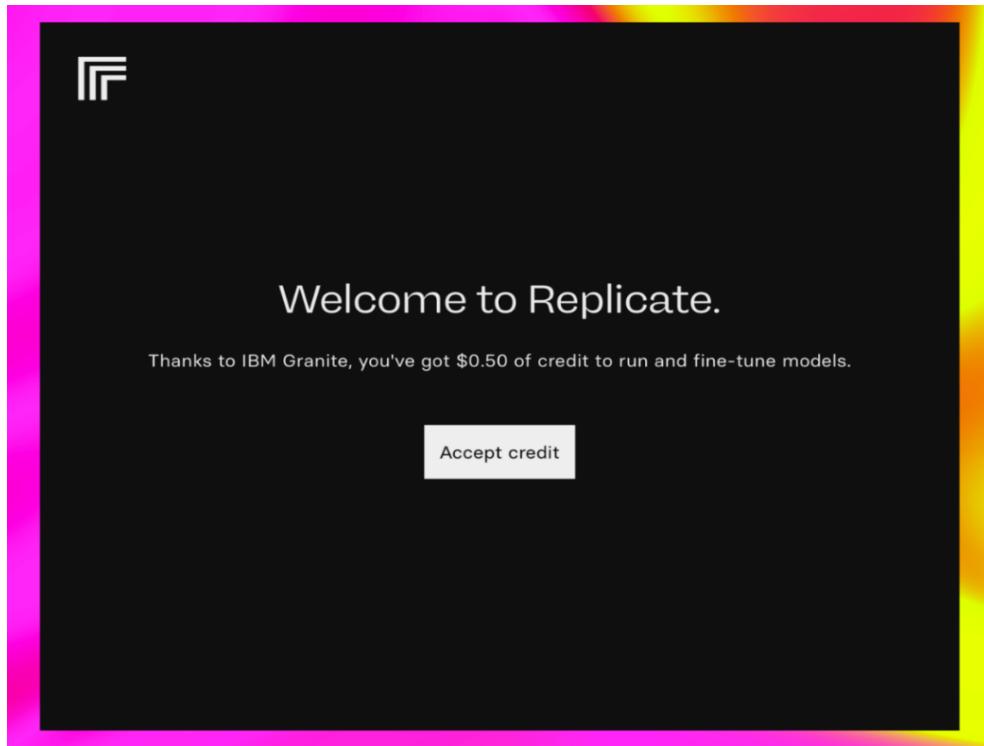
Note: Save the Replicate token because you'll need it to authenticate with the Replicate API when running code in the Google Colab environment later in this lab.



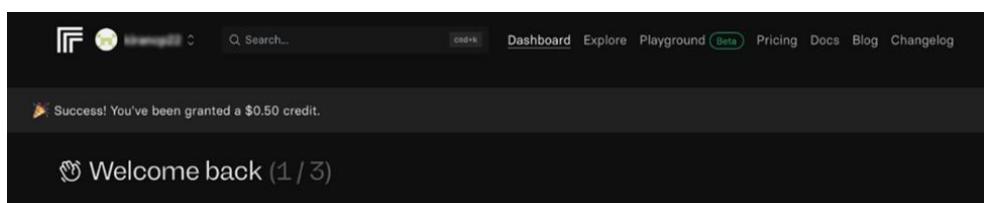
10. To run the lab without interruptions, you'll require some Replicate credits.

Go to the [Replicate invite](#) link to claim your free \$0.50 credit.

11. To claim the amount, select **Accept credit**.



12. A confirmation message is displayed indicating that a \$0.50 credit has been added to your Replicate account as shown in the following screenshot.



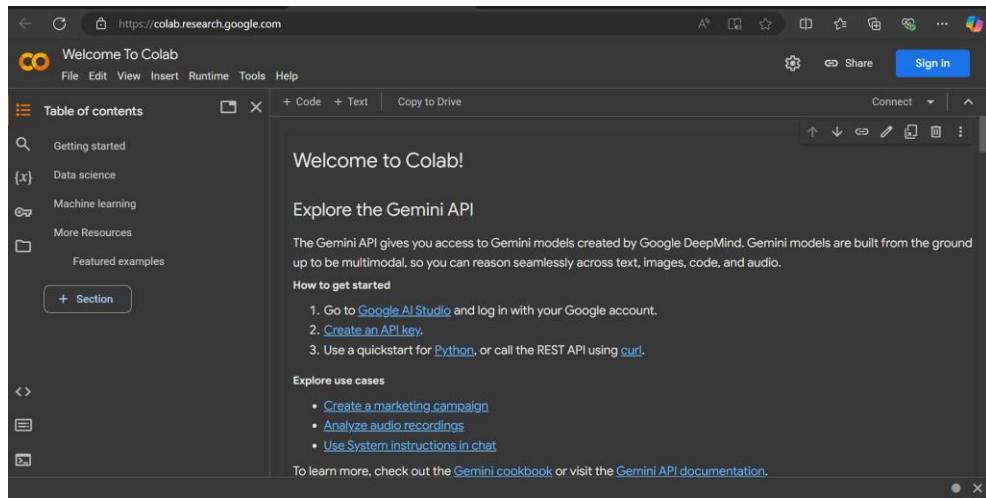
Sign up for Google Colab

Overview

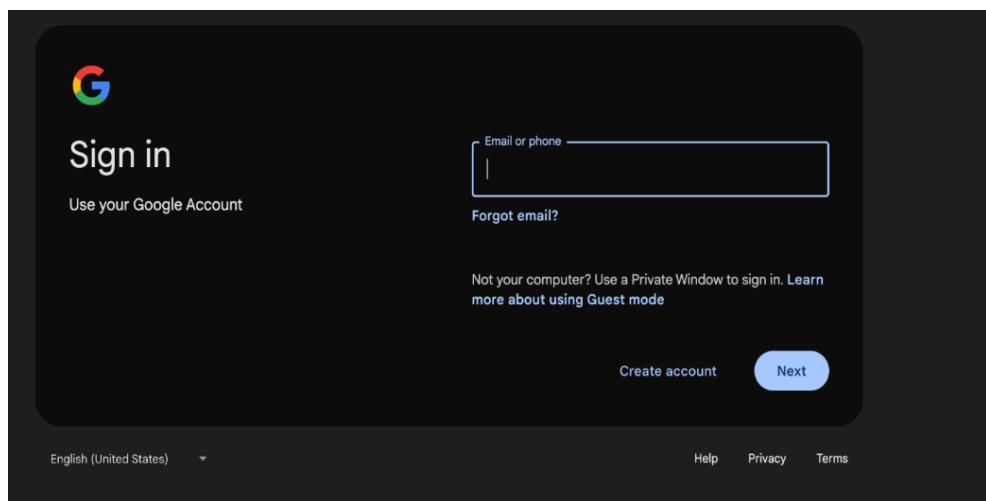
In this procedure, you'll set up a Google Colab account. Google Colab is a free cloud platform that lets you run code in notebooks, which are commonly used for machine learning, data science, and AI tasks. A Google Colab account allows you to install and use the tools needed to complete this lab.

Instructions

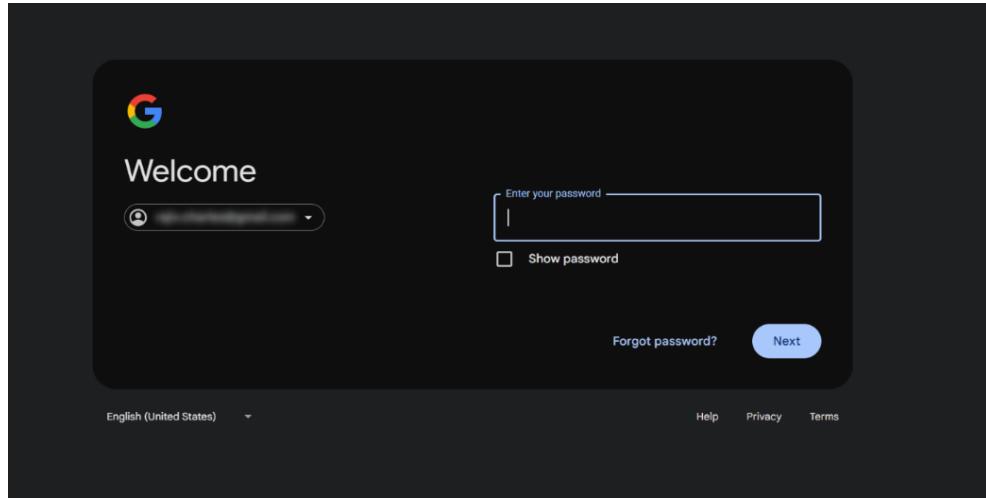
1. To sign up, go to the [Google Colab](#) website.
2. Select **Sign in**, as shown in the following screenshot.



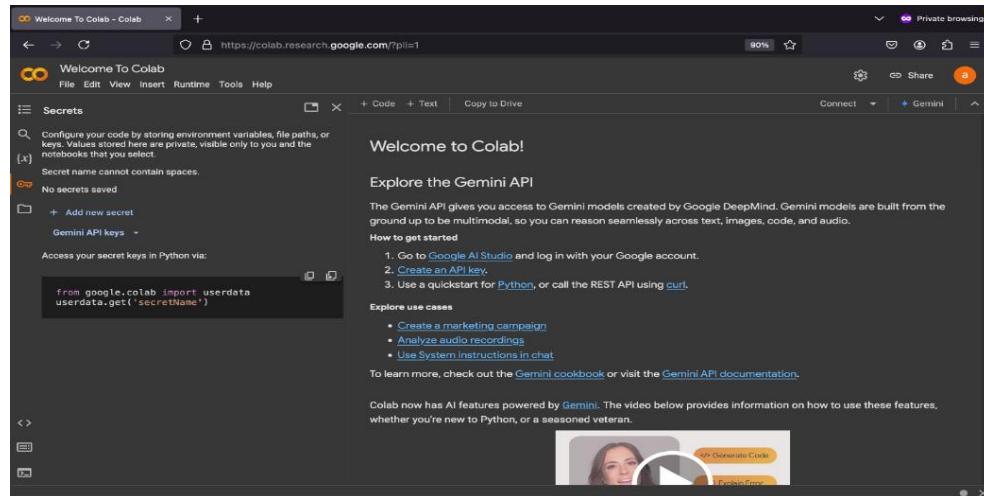
3. Enter your email or phone number in the **Email or phone** field, and then select **Next**.
The following screenshot shows the Google sign-in page.



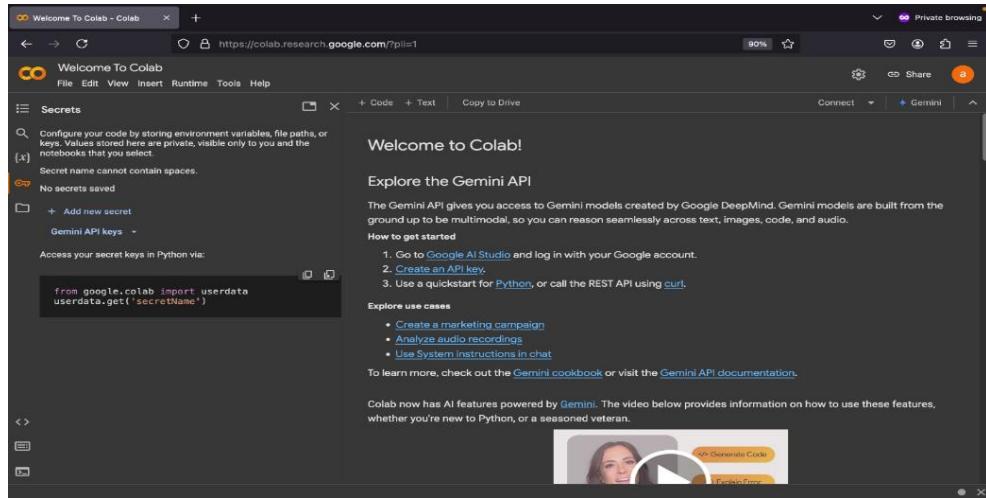
4. Enter your password in the **Enter your password** field, and then select **Next**.



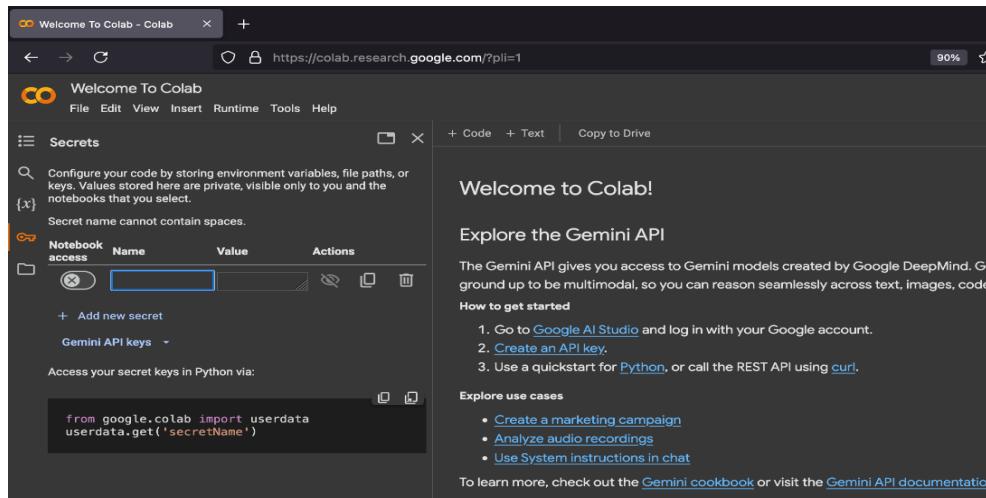
5. To store your Replicate API token, select the key icon from the Secrets tab on the Welcome to Colab page, as shown in the following screenshot.



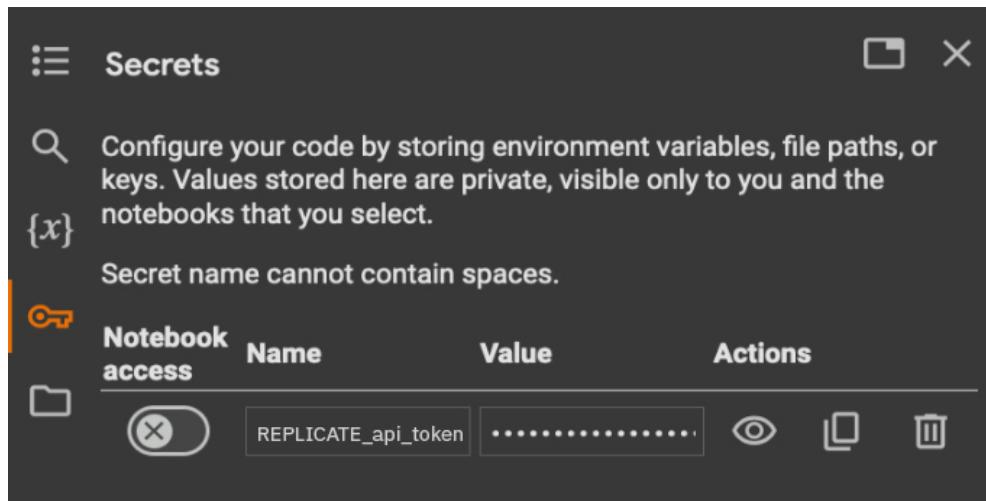
6. Select Add new secret.



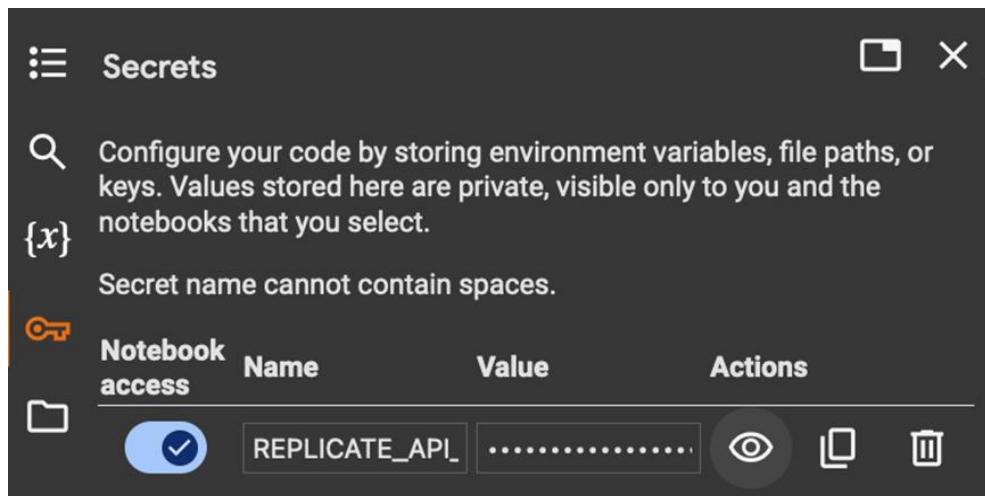
7. Type REPLICATE_api_token in the Name field.



8. Paste your Replicate API token you copied into the **Value** field, as shown in the following screenshot.



9. Set the **Notebook access** switch on, as shown in the following screenshot. Then, select the close icon to exit the configuration.



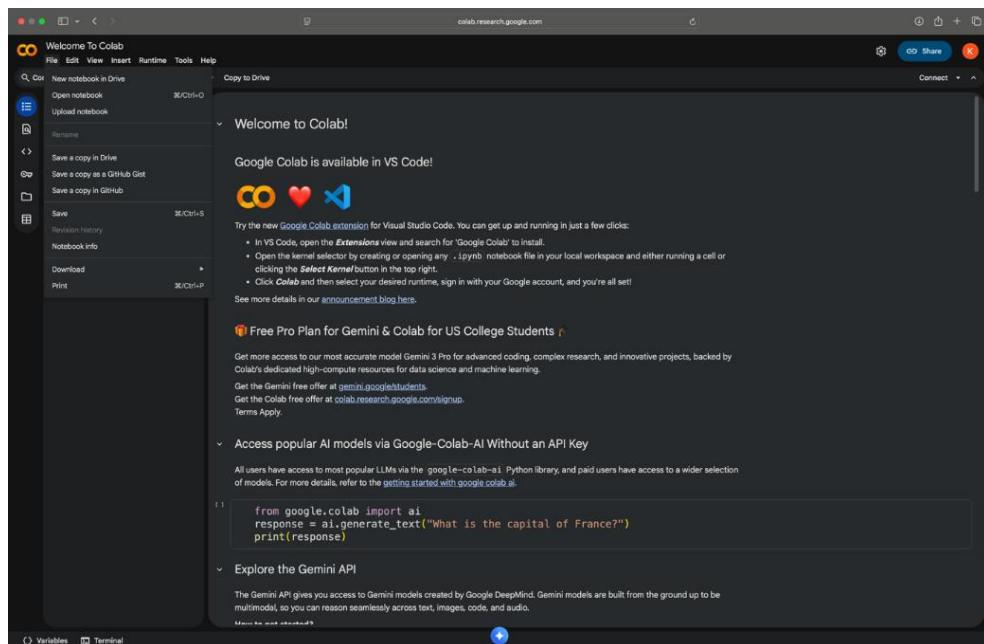
Load the Jupyter notebook and initialize the model

Overview

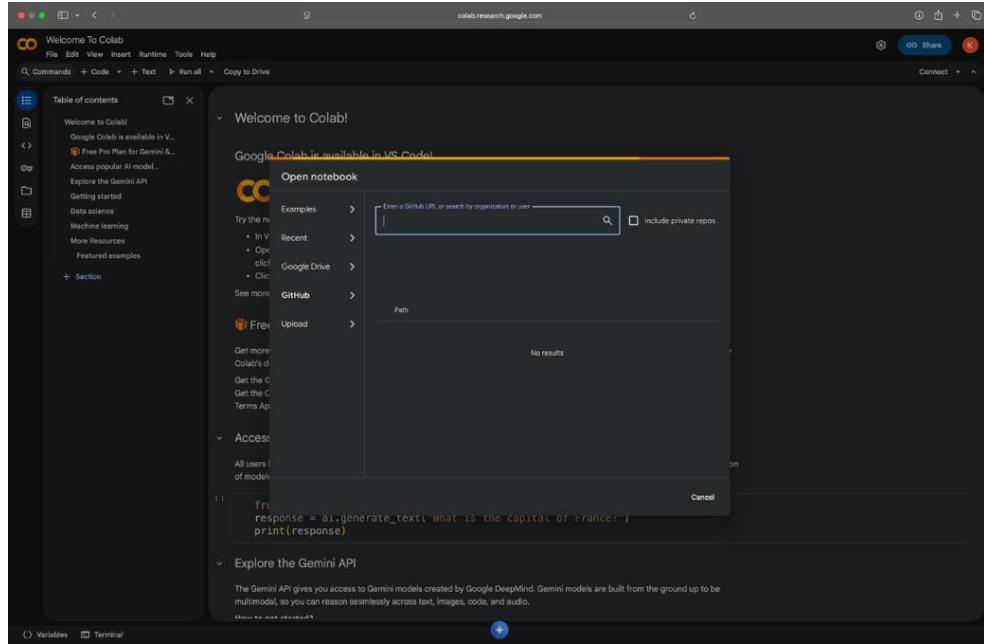
In this procedure, you'll load the Jupyter notebook and initialize the AI model. This sets up your environment before you can interact with the AI model.

Instructions

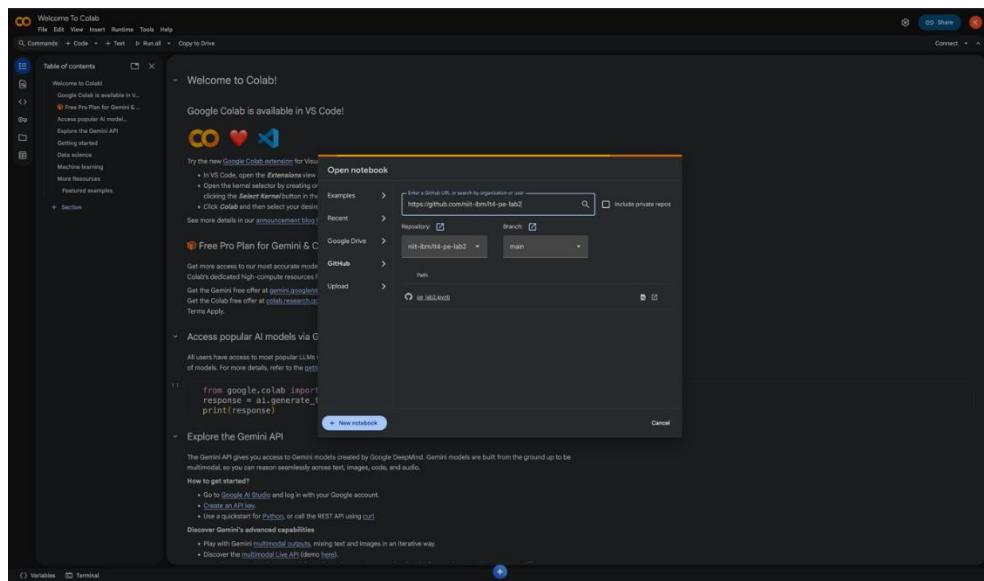
1. In your Google Colab workspace, select **File > Open notebook**.



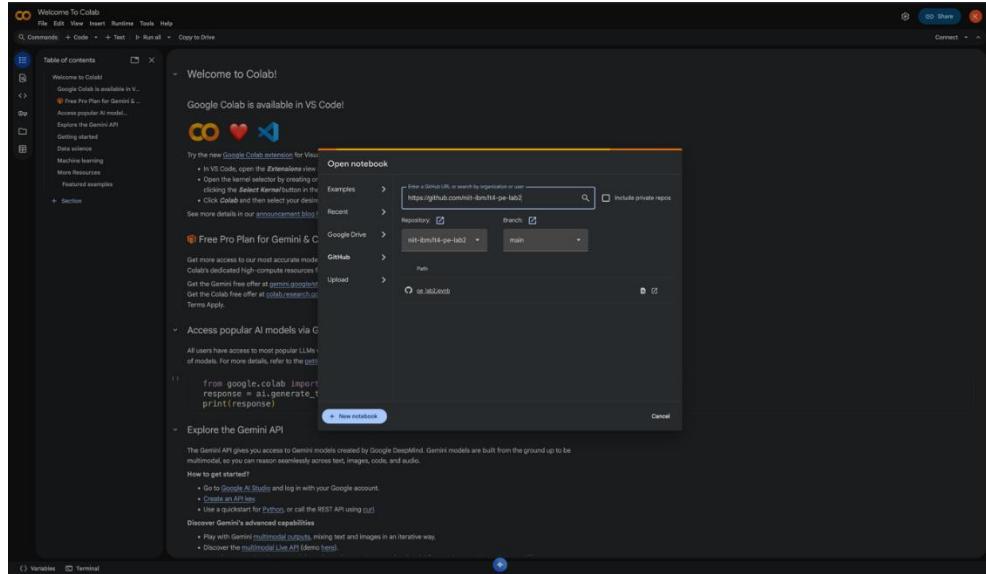
2. From the navigation menu, select **GitHub**.



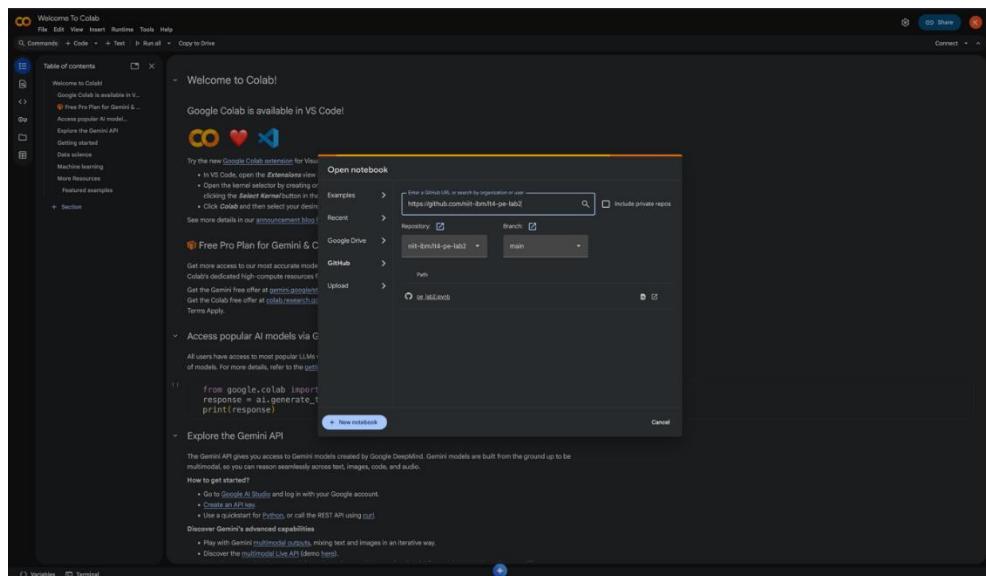
3. In the **Enter a GitHub URL or search by organization or user** field, copy and paste the following URL: <https://github.com/niit-ibm/lt4-pe-lab2> and then, select the **search** icon.



4. In the Branch section, select **main**.



5. Select the **pe_lab2.ipynb** notebook in the Path section to open the notebook in Google Colab.



Result: A notebook titled “**pe-lab2**” that you need to use for this lab activity opens in the Colab Workspace. Note that each row in the notebook is referred to as a **cell**.

6. To run the code in the Jupyter notebook, you’ll need to start a new instance in Google Colab runtime. For this, select the “**Connect**” menu on the Google Colab navigation bar, and then, select the **Connect to a hosted runtime** option.

Result: A green checkmark indicates that you have successfully connected to the hosted runtime.

```

File Edit View Insert Runtime Tools Help
File Commands + Code - + Text Run Cell Kernel
pe_lab2.ipynb ⓘ
Cell 1 of 1
[1]: # Install required libraries
!pip install git+https://github.com/ibm-granite-community/utils \
    --upgrade --quiet
[2]: replicate

[3]: import os
import replicate
from langchain import LangChain
from ibm_granite_community.llms import Replicate
from ibm_granite_community.notebook_utils import get_env_var
import ipywidgit as widgit
from IPython.display import display, Markdown, HTML
from string import Template

# Use the utility function to get from environment or prompt
replicate_api_token = get_env_var("REPLICATE_API_TOKEN")
os.environ["REPLICATE_API_TOKEN"] = replicate_api_token

# Model configuration - matching reference implementation
MAX_TOKENS = 1024
TEMPERATURE = 0.2

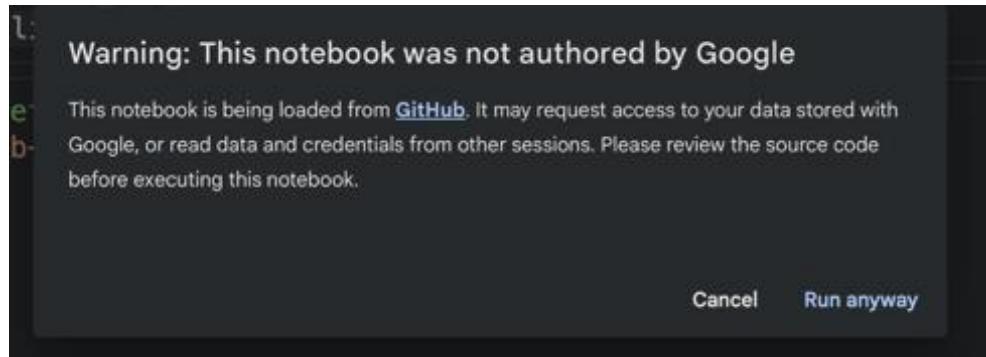
# Initialize the model
llm = Replicate(
    model=MODEL_NAME,
    model_kwargs={},
    max_tokens=MAX_TOKENS,
    temperature=TEMPERATURE
)
print(f"Model initialized: {MODEL_NAME}")
print(f"Max Tokens: {MAX_TOKENS}")
print(f"Temperature: {TEMPERATURE}")

# Task 1: Draft an initial prompt for a specific project
# This is NOT a template yet - it's hard-coded for one specific use case
baseline_prompt = """Write a concise weekly status summary for the DeltaFin client project.
Highlight progress made this week, note any blockers, and outline next steps in a professional tone suitable for a client email."""
print("TASK 1: Baseline Prompt (Single-Use)")
print("w70")
print("\nprompt:")

```

- In the first cell of the notebook, select the **Play** button to install the required libraries from the Granite suite.

- Select **Run anyway** to proceed loading the required libraries.



9. Once installation is complete you are prompted to restart the runtime session. Select **Restart session**.

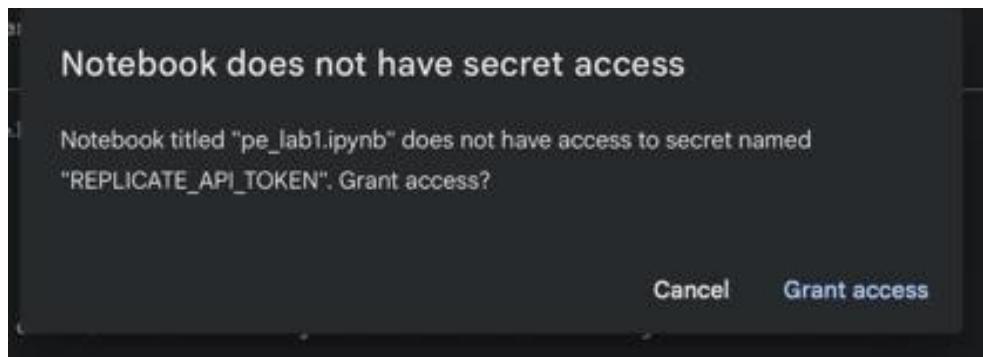
10. Select **Run** to import the libraries.



The screenshot shows a Jupyter Notebook interface with a single cell containing Python code. The code imports various modules and defines a function named 'Replicate'. The notebook has a title 'RESTART SESSION' and includes standard Jupyter navigation buttons.

```
import os
import replicate
from langchain_community.llms import Replicate
from langchain_granite.langchain_notebook_utils import get_env_var
import warnings
warnings.filterwarnings('ignore')
from IPython.display import display, Markdown, HTML
from string import Template
```

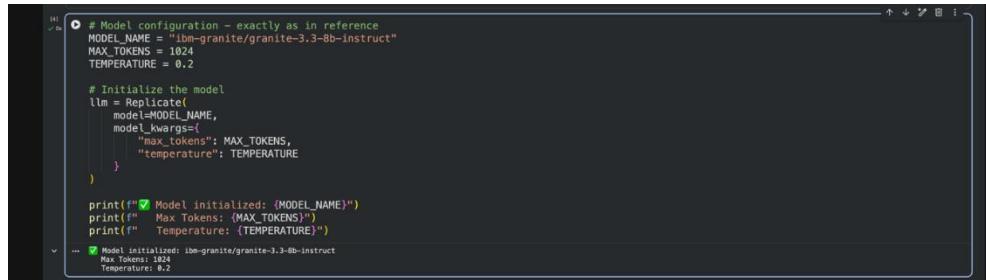
11. From the next cell, select **Grant access** to let the model access the Replicate Token from the secrets.



12. The following message displays at the bottom of the cell: REPLICATE_API_TOKEN loaded from Google Colab secret.

```
# Use the utility function to get from environment or prompt
replicate_api_token = get_env_var("REPLICATE_API_TOKEN")
os.environ["REPLICATE_API_TOKEN"] = replicate_api_token
... REPLICATE_API_TOKEN loaded from Google Colab secret.
```

13. Select the **Play** button to initialize the granite-3.3-8B-instruct model.



The screenshot shows a code editor window with a dark theme. The code is written in Python and defines a model configuration and its initialization. It uses the Replicate library to initialize the model with specific parameters like MAX_TOKENS and TEMPERATURE. The code then prints the initialized model's name, maximum tokens, and temperature. The output shows the model has been initialized with the name 'ibm-granite/granite-3.3-8b-instruct', a maximum of 1024 tokens, and a temperature of 0.2.

```
# Model configuration - exactly as in reference
MODEL_NAME = "ibm-granite/granite-3.3-8b-instruct"
MAX_TOKENS = 1024
TEMPERATURE = 0.2

# Initialize the model
l1m = Replicate(
    model=MODEL_NAME,
    model_kwargs={
        "max_tokens": MAX_TOKENS,
        "temperature": TEMPERATURE
    }
)

print(f"\n✓ Model initialized: {MODEL_NAME}")
print(f"  Max Tokens: {MAX_TOKENS}")
print(f"  Temperature: {TEMPERATURE}")

... -- ✓ Model initialized: ibm-granite/granite-3.3-8b-instruct
      Max Tokens: 1024
      Temperature: 0.2
```

Draft a baseline prompt for a recurring task

Overview

In this procedure, you'll draft an initial, working prompt for a specific use case.

Instructions

1. Begin by writing a baseline prompt for generating weekly status summaries of the projects.

To do this, copy the following code, paste it into a new code cell, and then select the **Play** icon to execute it.

```
# Task 3: Draft an initial prompt for a specific project
# This is NOT a template yet - it's hard-coded for one specific use
case
baseline_prompt = """Write a concise weekly status summary for the
DeltaFin
client project.
Highlight progress made this week, note any blockers, and outline next
steps in a
professional
tone suitable for a client email."""

print("TASK 3: Baseline Prompt (Single-Use)")
print("-" * 70)
print("\nPrompt:")
print(baseline_prompt)
print("\n" + "-" * 70)
print("Generating output...")

# Generate output using the baseline prompt
baseline_output = lm.invoke(baseline_prompt)

print("Generated Output:")
print("-" * 70)
print(baseline_output)
print("-" * 70)
```

2. The following screenshot shows the code pasted into the code cell.

```
pe_lab2.ipynb: ②
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run All
③
# Task 3: Draft an initial prompt for a specific project
# This is NOT a template yet - it's hard-coded for one specific use case
baseline_prompt = """Write a concise weekly status summary for the DeltaIn client project.
Highlight progress made this week, note any blockers, and outline next steps in a professional tone suitable for a client email."""
print("TASK 3: Baseline Prompt (Single-Use)")
print("\n#70")
print("\nprompt")
print(baseline_prompt)
print("\n" + "-"*70)
print("Generating output...\n")

# Generate output using the baseline prompt
baseline_output = llm.invoke(baseline_prompt)

print("Generated Output:")
print("-" * 70)
print(baseline_output)
print("-" * 70)
```

Result: After you enter the baseline prompt, the LLM generates an output as shown in the following screenshot.

```
pe_lab2.ipynb: ②
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run All
③
print("\n#70")
print("\nprompt")
print(baseline_prompt)
print("\n" + "-"*70)
print("Generating output...\n")

# Generate output using the baseline prompt
baseline_output = llm.invoke(baseline_prompt)

print("Generated Output:")
print("-" * 70)
print(baseline_output)
print("-" * 70)

--- TASK 3: Baseline Prompt (Single-Use)
_____
Prompts
Write a concise weekly status summary for the DeltaIn client project.
Highlight progress made this week, note any blockers, and outline next steps in a professional tone suitable for a client email.

Generating output...
Generated Output:
Subject: Weekly Status Update - DeltaIn Project
Dear Client's Name,
I hope this message finds you well. I am writing to provide you with a concise update on the DeltaIn project's progress for this week.

+Progress Update+
1. +Data Migration+ We successfully migrated 80% of the client data into the new system. The remaining 20% is currently undergoing a thorough review to ensure data integrity before the final migration.
2. +System Integration+ The integration of the DeltaIn system with the existing financial infrastructure is progressing smoothly. We have completed the initial testing phase and are now in the final stages of validation.
3. +User Training+ Our training sessions for key personnel have been well-received, with 95% of attendees reporting a good understanding of the new system's functionalities.

+Blockers+
1. +Data Review+ The remaining 20% of data requires additional scrutiny due to inconsistencies identified during the initial migration. This process is expected to be completed by the end of this week.
2. +Integration Testing+ A minor bug was discovered during the final stages of integration testing. Our development team is working diligently to resolve this issue, with an estimated resolution by early next week.

+Next Steps+
1. +Final Data Migration+ Once the data review is complete, we will proceed with the final migration, aiming to have it wrapped up by the end of this week.
2. +Integration Testing+ Post-bug resolution, we will re-run the integration tests and seek a successful validation by mid-next week.
3. +User Training+ We will schedule additional training sessions for the remaining team members, targeting completion by the end of next week.

We appreciate your patience and understanding as we navigate these final stages of the project. Our team remains committed to delivering a seamless transition to the DeltaIn system.

Should you have any questions or require further clarification, please do not hesitate to reach out.

Best regards,
[Your Name]
[Your Position]
[Your Contact Information]
```

Note that the output addresses all required elements and produces a complete output. However, it gives a vague time reference ("this week") and it cannot be reused for other projects without a complete rewrite. This prompt works for one specific situation, but it's not scalable.

Identify variable elements and convert to a template

Overview

In this procedure, you'll identify variables such as client name, time period, progress items, blockers, next steps, tone, and audience and convert your baseline prompt to a template.

Instructions

1. Enter the details of the variable of your prompt template in the LLM.

To do this, copy the following code, paste it into a new code cell, and then select the **Play** icon to execute it.

```
# Task 4: Convert the baseline prompt into a reusable template
# with variables
# Using Python's string formatting with named placeholders
prompt_template = """Write a concise weekly status summary for
the
{client_name} project.
```

Summarize progress made during {time_period}, note {blockers},
and outline
{next_steps}
in a {tone} tone suitable for {audience}.

Additional context:

- progress: {progress}
- format: {output_format}"""

```
print("TASK 4: Reusable Prompt Template")
print("-" * 70)
print("\nTemplate with Variables:")
print("-" * 70)
print(prompt_template)
print("-" * 70)

print("\n Template created successfully!")
print("\nVariables identified:")
variables = ["{client_name}", "{time_period}", "{progress}",
"{blockers}",
" {next_steps}", "{tone}", "{audience}",
"{output_format}"]
```

```
for var in variables:  
    print(f" - {var}")
```

Result: The following screenshot shows the code pasted into the code cell.

```
# Task 4: Convert the baseline prompt into a reusable template with variables

# Using Python's string formatting with named placeholders
prompt_template = "Write a concise weekly status summary for the {client_name} project.

Summarize progress made during {time_period}, note {blockers}, and outline
{next_steps}
in a {tone} tone suitable for {audience}.

Additional context:
- Progress: {progress}
- Format: {output_format}
{more}

print("TASK 4: Reusable Prompt Template")
print("Template with Variables:")
print("-*#")
print(prompt_template)
print("-*#")

print("Template created successfully!")
print("Variables identified:")
variables = ["client_name", "{time_period}", "{progress}", "{blockers}",
            "{next_steps}", "{tone}", "{audience}", "{output_format}"]
for var in variables:
    print("  " + var)
```

The message “Template created successfully!” is displayed. At the end, the identified variables are displayed as follows:

- {client_name}
 - {time_period}
 - {progress}
 - {blockers}
 - {next_steps}
 - {tone}
 - {audience}
 - {output_format}

This prompt template can be used to generate summaries for any project. It produces outputs with a consistent structure and is scalable for use across teams. You only need to supply values for the variables. The structure and instructions remain fixed, producing reliable and consistent output quality.

```
○ a@ubc2pylab: ~ ○
File Edit New Insert Runfile Tools Help
File Commands + Code + Test > Runall +
[ ] prompt_template = """Write a concise weekly status summary for the
  (client_name) project.

  Summarize progress made during {time_period}, note {blockers}, and outline
  {next_steps} in a {tone} tone suitable for {audience}.

  Additional context:
    - Progress: {progress}
    - Format: {output_format}
  """

  print("TASK 4: Reusable Prompt Template")
  print("=*=#0")
  print("Template with Variables:")
  print("=*=#0")
  print(prompt_template)
  print("=*=#0")

  print("✓ Template created successfully!")
  print("Variables identified:")
  variables = ['(client_name)', '{time_period}', '{progress}', '{blockers}', '{next_steps}', '{tone}', '{audience}', '{output_format}']
  for var in variables:
    print("  " + var)

... --> TASK 4: Reusable Prompt Template
=====
Template with Variables
-----
Write a concise weekly status summary for the (client_name) project.

  Summarize progress made during {time_period}, note {blockers}, and outline {next_steps} in a {tone} tone suitable for {audience}.

  Additional context:
    - Progress: {progress}
    - Format: {output_format}

-----
✓ Template created successfully!
Variables identified:
  (client_name)
  {time_period}
  {progress}
  {blockers}
  {next_steps}
  {tone}
  {audience}
  {output_format}
```

Test and finalize the prompt template

Overview

In this procedure, you'll test your prompt template with real inputs to verify if it produces reliable outputs. You'll test the template with two set of inputs to verify:

- Consistent structure across different inputs
- Proper tone adaptation
- Audience-appropriate content
- Complete coverage of requirements

Instructions

1. Begin by testing the template for defining and loading the variables for the DeltaFin Project, a Financial services project. To do this, copy the following code, paste it into a new code cell, and then select the **Play** icon to execute it.

```
# Test Case 1: DeltaFin Project (Financial Services)
# Define variables for Project 1: DeltaFin
project1_vars = {
    "client_name": "DeltaFin",
    "time_period": "Week 4",
    "progress": "Completed API integration testing and resolved 15 critical bugs",
    "blockers": "Testing delays caused by the external vendor's infrastructure issues",
    "next_steps": "Finalize the integration plan and begin UAT preparation",
    "tone": "formal",
    "audience": "senior client stakeholders",
    "output_format": "structured paragraph with clear sections"
}

# Substitute variables into the template
project1_prompt = prompt_template.format(**project1_vars)

print("TASK 5: Testing Template - Project 1 (DeltaFin)")
print("-" * 70)
print("\nVariable Values:")
for key, value in project1_vars.items():
    print(f"{key}: {value}")

print("\n" + "-" * 70)
print("\nGenerated Prompt:")
print("-" * 70)
```

```
print(project1_prompt)
print("-" * 70)

print("\n" + "-" * 70)
print("Generating output... \n")

# Generate output
project1_output = lm.invoke(project1_prompt)

print("Generated Status Summary:")
print("-" * 70)
print(project1_output)
print("-" * 70)
```

The following screenshot shows the code pasted into the code cell.

The screenshot shows a Jupyter Notebook cell titled "Test Case 1: DeltaFin Project (Financial Services)". The code in the cell is as follows:

```
# Define variables for Project 1: DeltaFin
project1_vars = {
    "client_name": "DeltaFin",
    "time_period": "Week 4",
    "progress": "Completed API integration testing and resolved 15 critical
        bugs.",
    "blockers": "Testing delays caused by the external vendor's infrastructure
        issues.",
    "next_steps": "Finalize the integration plan and begin UAT preparation",
    "tone": "Formal",
    "audience": "Senior client stakeholders",
    "output_format": "structured paragraphs with clear sections"
}

# Substitute variables into the template
project1_prompt = prompt_template.format(**project1_vars)

print("TASK 5: Testing Template - Project 1 (DeltaFin)")
print("-" * 70)
print("Available Values:")
for key, value in project1_vars.items():
    print(f" {key}: {value}")

print("\n" + "-" * 70)
print("Generated Prompt:")
print(project1_prompt)

print("\n" + "-" * 70)
print("Generating output... \n")

# Generate output
project1_output = lm.invoke(project1_prompt)

print("Generated Status Summary:")
print("-" * 70)
print(project1_output)
print("-" * 70)
```

Result: After you enter the template and edit it as per DeltaFin's project requirements, the generated result is tailored to this use case.

```

print("\n"+ "="*70)
print("Generating output...\n")
# Generate output
project1_output = llm.invoke(project1_prompt)
print("Generated Status Summary:")
print("-" * 70)
print(project1_output)
print("-" * 70)

-- TASK 3: Testing Template - Project 1 (DeltaFin)
Variable Values:
client_name: DeltaFin
time_period: Week 4
progress: Completed API integration testing and resolved 15 critical bugs
status: Stable, but some minor delays due to external vendor's infrastructure issues
next_steps: Finalize the integration plan and begin VAT preparation
tone: Formal
audience: Senior client stakeholders
output_format: Structured paragraph with clear sections

Generated Prompt:
Write a concise weekly status summary for the DeltaFin project.
Summarize progress made during Week 4, note testing delays caused by the external vendor's infrastructure issues, and outline finalize the integration plan and begin VAT preparation.
In a formal tone suitable for senior client stakeholders.

Additional context:
- Progress: Completed API integration testing and resolved 15 critical bugs
- Format: structured paragraph with clear sections

Generating output...
Generated Status Summary:
Week 4 Status Summary: DeltaFin Project
Progress Overview:
This week marked significant advancements in the DeltaFin project, with the completion of API integration testing and the resolution of 15 critical bugs. These accomplishments have laid a robust foundation for the subsequent phases of the project.
Challenges Encountered:
Notably, testing activities encountered delays due to unforeseen infrastructure issues experienced by our external vendor. These challenges have necessitated a recalibration of our timeline, emphasizing the importance of contingency planning in future project scope.
Upcoming Actions:
In response to these developments, our immediate focus shifts to finalizing the integration plan. This involves a comprehensive review of all integrated components to ensure seamless functionality and compatibility across the system. Concurrently, we will initiate a thorough security audit to ensure the system's resilience against potential threats.
We remain committed to maintaining transparency and are prepared to address any concerns or questions regarding these matters. Your continued support and collaboration are invaluable as we navigate these critical stages of the DeltaFin project.

Respectfully submitted,
[Your Name]
[Your Position]
[Your Contact Information]

```

- Now, try defining and loading the variables for the MediTrack, a project for a Healthcare organization. To do this, copy the following code, paste it into a new code cell, and then select the **Play** icon to execute it.

```

# Test Case 2: MediTrack Project (Healthcare Technology)
# Define variables for Project 2: MediTrack
project2_vars = {
    "client_name": "MediTrack",
    "time_period": "Sprint 2",
    "progress": "Successfully deployed the patient data dashboard and completed security audit",
    "blockers": "pending approvals from the compliance team for HIPAA certification",
    "next_steps": "complete the data-mapping activity and prepare for pilot testing",
    "tone": "neutral, business-professional",
    "audience": "internal project sponsors",
    "output_format": "bullet-point list with clear categories"
}

# Substitute variables into the template
project2_prompt = prompt_template.format(**project2_vars)

```

```
print("TASK 5: Testing Template - Project 2 (MediTrack)")  
print("-" * 70)  
print("\nVariable Values:")  
for key, value in project2_vars.items():  
    print(f"{key}: {value}")  
  
print("\n" + "-" * 70)  
print("\nGenerated Prompt:")  
print("-" * 70)  
print(project2_prompt)  
print("-" * 70)  
  
print("\n" + "-" * 70)  
print("Generating output... \n")  
  
# Generate output  
project2_output = lm.invoke(project2_prompt)  
  
print("Generated Status Summary:")  
print("-" * 70)  
print(project2_output)  
print("-" * 70)  
2. Context: The Reusable Prompt Template Definition  
  
# Task 4: Convert the baseline prompt into a reusable template  
with variables  
# Using Python's string formatting with named placeholders  
prompt_template = """Write a concise weekly status summary for  
the  
{client_name} project.  
  
Summarize progress made during {time_period}, note {blockers},  
and outline  
{next_steps}  
in a {tone} tone suitable for {audience}.  
  
Additional context:  
- progress: {progress}  
- format: {output_format}"""
```

The following screenshot shows the code pasted into the code cell.

```

# Define variables for Project 2: MediTrack
project2_vars = {
    "client_name": "MediTrack",
    "time_period": "Sprint 2",
    "progress": "Successfully deployed the patient data dashboard and completed security audit",
    "blockers": "pending approvals from the compliance team for HIPAA certification",
    "next_steps": "complete the data-mapping activity and prepare for pilot testing",
    "tone": "neutral, business-professional",
    "audience": "internal project sponsors",
    "output_format": "bullet-point list with clear categories"
}

# Substitute variables into the template
project2_prompt = prompt_template.format(**project2_vars)

print("TASK 5: Testing Template - Project 2 (MediTrack)")
print("\nVariable Values:")
for key, value in project2_vars.items():
    print(f" {key}: {value}")

print("\n" + "="*70)
print("Generated Prompt:")
print(project2_prompt)

print("\n" + "="*70)
print("Generating output...\n")

# Generate output
project2_output = llm.invoke(project2_prompt)

print("Generated Status Summary:")
print("-" * 70)
print(project2_output)
print("="*70)

```

Result: After you enter the variables as per MediTrack's project requirements, the following output is generated.

This template produces consistent, high-quality, and well-formatted summaries that adapt to different contexts, tones, and audiences, while maintaining professional standards.

```

# Define variables for Project 2: MediTrack
project2_vars = {
    "client_name": "MediTrack",
    "time_period": "Sprint 2",
    "progress": "Successfully deployed the patient data dashboard and completed security audit",
    "blockers": "pending approvals from the compliance team for HIPAA certification",
    "next_steps": "complete the data-mapping activity and prepare for pilot testing",
    "tone": "neutral, business-professional",
    "audience": "internal project sponsors",
    "output_format": "bullet-point list with clear categories"
}

# Substitute variables into the template
project2_prompt = prompt_template.format(**project2_vars)

print("TASK 5: Testing Template - Project 2 (MediTrack)")

Variable Values:
client_name: MediTrack
time_period: Sprint 2
progress: Successfully deployed the patient data dashboard and completed security audit
blockers: pending approvals from the compliance team for HIPAA certification
next_steps: complete the data-mapping activity and prepare for pilot testing
tone: neutral, business-professional
audience: internal project sponsors
output_format: bullet-point list with clear categories

Generated Prompt:
Write a concise weekly status summary for the MediTrack project.

Summarize progress made during Sprint 2, note pending approvals from the compliance team for HIPAA certification, and outline complete the data-mapping activity and prepare for pilot testing
In a neutral, business-professional tone suitable for internal project sponsors.

Additional Notes:
- Progress: Successfully deployed the patient data dashboard and completed security audit
- Format: bullet-point list with clear categories

Generating output...
Generated Status Summary:
- Sprint 2 Progress:
  - Successfully deployed the patient data dashboard, enhancing user accessibility and functionality.
  - Completed security audit; ensuring non-compliance data protection measures are in place.
- Pending Approvals:
  - Awaiting HIPAA certification approval from the compliance team to ensure full regulatory compliance.
- Upcoming Activities:
  - Completing the data-mapping activity, establishing clear connections between various data sources and the MediTrack system.
  - Preparing for pilot testing, which will involve select stakeholders to validate system performance and gather user feedback before full-scale implementation.

This summary outlines the accomplishments and next steps for the MediTrack project, emphasizing our commitment to both regulatory adherence and system readiness for broader use.

```

Conclusion

Congratulations! You have learned how to develop a prompt template using variables to generate output from LLMs. Your template now produces clear, consistent outputs for any project, and can be used across teams.