LAB

Optimize AI-Generated Code Using IBM Granite Models

IBM **SkillsBuild**

Contents

Introduction

This lab focuses on optimizing AI generated code using few-shot prompt techniques with the IBM Granite Code model on Google Colab to optimize code generation quality. Few-shot prompting allows the model to draw from multiple examples, enabling more accurate and nuanced outputs than the initial zero-shot prompt approach.

This lab builds on Lab 1, which focused on generating code using Granite Code models, by shifting the focus to refining the model output through Few-Shot prompting techniques. The exercise centers on optimizing ipywidgets-based UI components to create a more intuitive interface for an online bookstore, demonstrating how incorporating examples can enhance the quality and precision of the generated code.

In this lab, we use the IBM Granite Code model family hosted on Replicate to generate Python code that integrates Jupyter Notebook ipywidgets, optimizing a user-friendly interface for a bookstore application.

Software requirements
To complete this lab, you must have access to an account with Replicate to make model inference calls. You must also obtain the API token from your Replicate account, which will be securely added as a Google Colab secret. Familiarity with Python is not necessary but can be helpful for understanding the generated code.

Objective
Use IBM Granite models for code generation and programming tasks.

Lab steps
This lab requires you to complete the following steps:

- Step 1: Create a GitHub account
- Step 2: Create a Replicate account
- Step 3: Sign up for Google Colab
- Step 4: Load the Jupyter notebook and initialize the model
- Step 5: Generate code using the IBM Granite model
- Step 6: Validate the generated code

Estimated duration to complete
30 minutes

Case study

Background

ReadersVerse is a local bookstore that plans to build an online presence by creating a website that allows its readers to search for and view its catalog and check availability prior to visiting the physical store. You are the web designer assigned to this project and have created the initial draft of the website using zero-shot prompting technique.

Challenge

The client appreciates the initial output and requests for enhancements to improve the overall visual design and functionality of the website. Use IBM Granite models to optimize the AI generated code and enhance the initial design of the website.

Solution

- You will use few-shot prompting technique to optimize the AI-generated code using IBM Granite to enhance the visual design and functionality of the website.

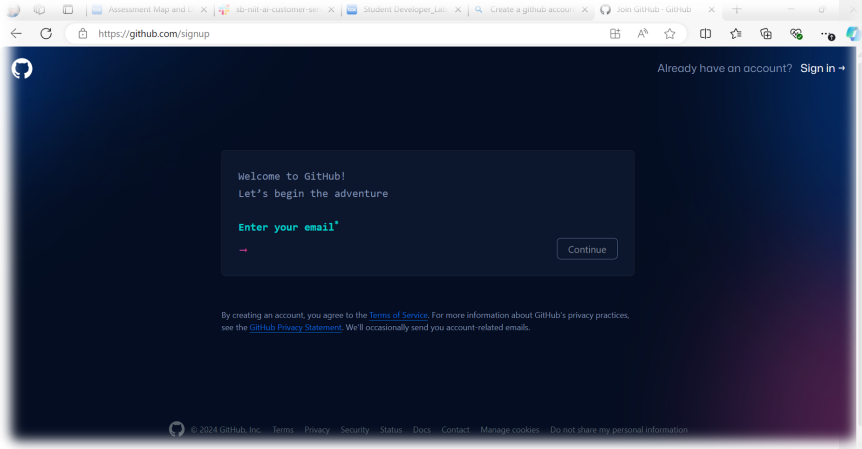
Step 1: Create a GitHub account

Overview

You need to set up the runtime environment for this lab before you can prompt IBM Granite for code generation activities. Create a GitHub account to begin setting up the runtime environment. Skip to Step 2 in this lab if you already have a GitHub account.

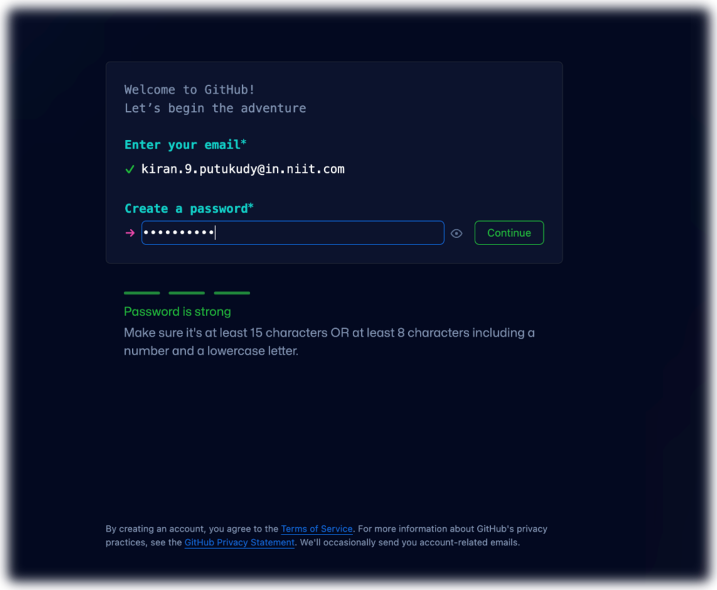
Tasks

1. Go to the GitHub website to create a **GitHub account**.



Developer note: TBD

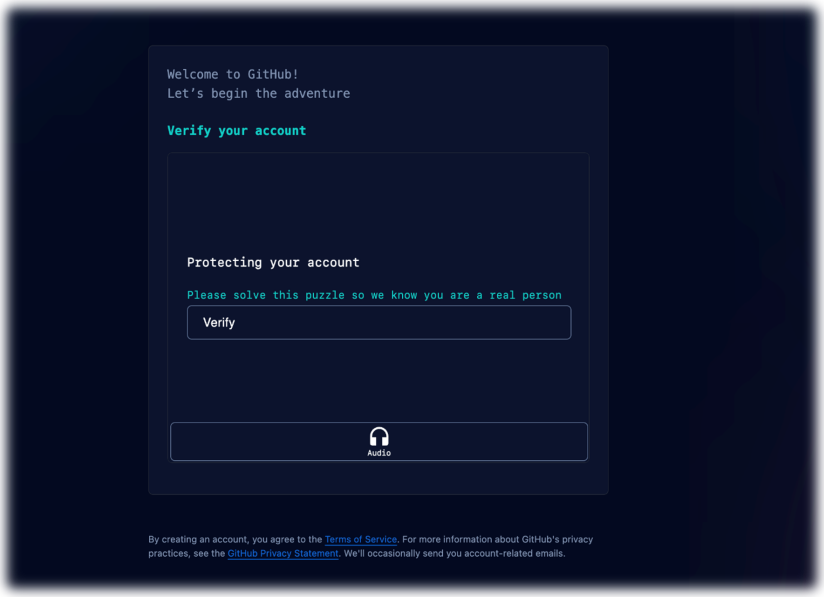
2. Enter your email and password details and select **Continue** to proceed.



3. Specify a username in the **Enter a username** field and select **Continue**.

Note: Remember your GitHub details because you will need these to sign up for Replicate.

4. **Verify your account** details to complete the GitHub account creation process.



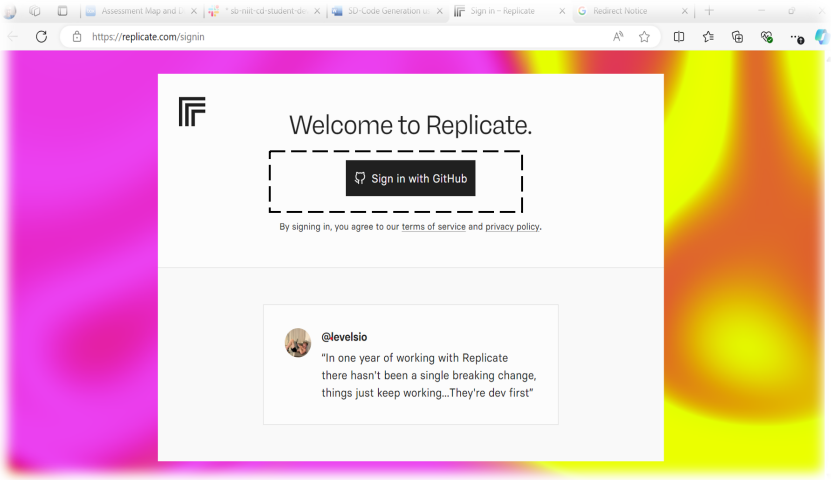
Step 2: Create a Replicate account

Overview

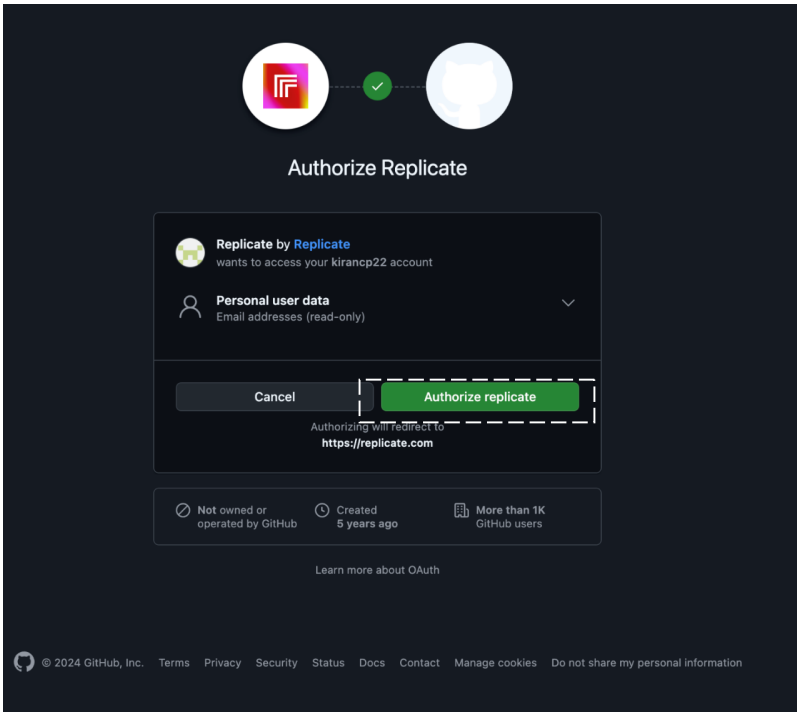
After creating the GitHub account, you will continue to set up the runtime environment by creating a Replicate account using your GitHub login details. This will enable you to create a Replicate token to install and run the various utilities needed to run the lab in the Google Colab environment. Skip to Step 3 in this lab if you already have a Replicate account and API token.

Tasks

1. Go to the **Replicate** website and select **Sign in with GitHub** option.

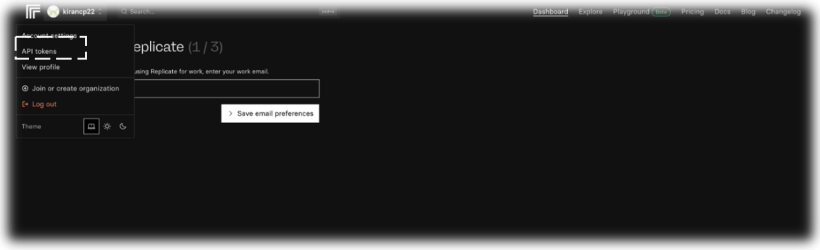


2. Select **Authorize replicate** to continue.

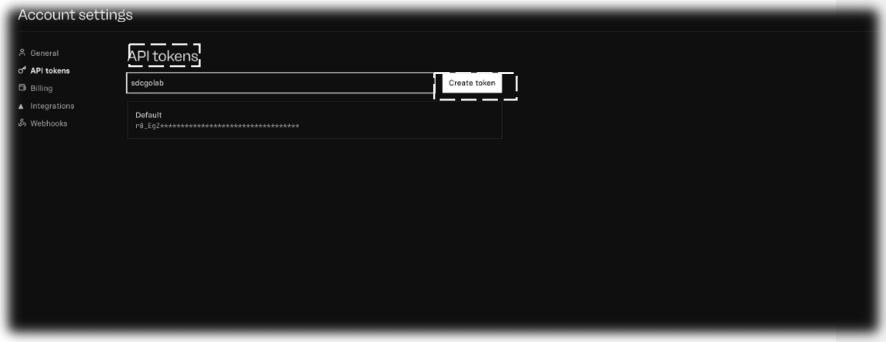


3. To create a **Replicate token**, select the **account settings** icon in the top-left corner of the navigation bar.

4. Select the **API tokens** option on the Account settings menu.



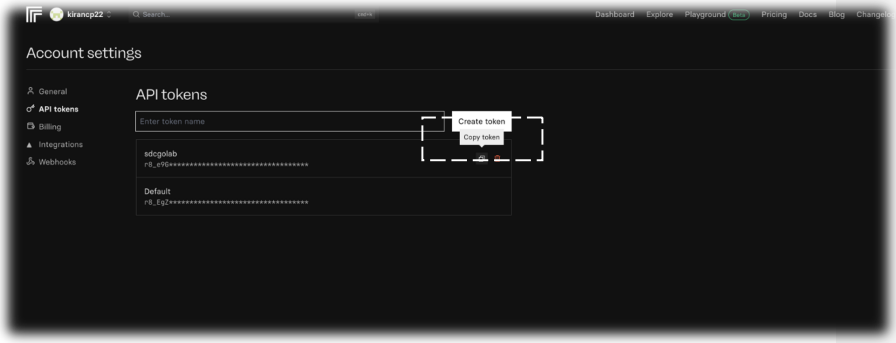
5. Specify a name for the **API token** in the API tokens textbox and select **Create token**.



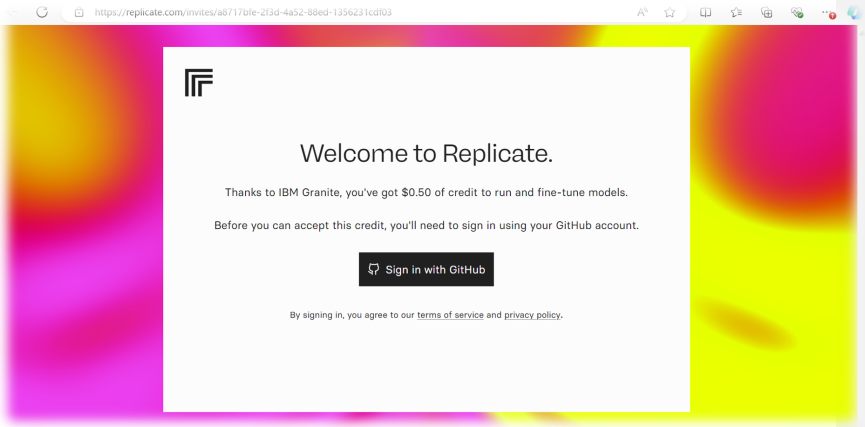
6. Your **API token** is displayed on the screen below the API tokens textbox.

7. Select the **Copy token** icon to copy the Replicate API token.

Note: Do not close this page because you will need to copy the Replicate API token to run the Google Colab instance later in this lab.



8. Open the following link in a new tab or browser window to claim 50 cents in **Replicate credits** to your account, which you will use to run the lab.



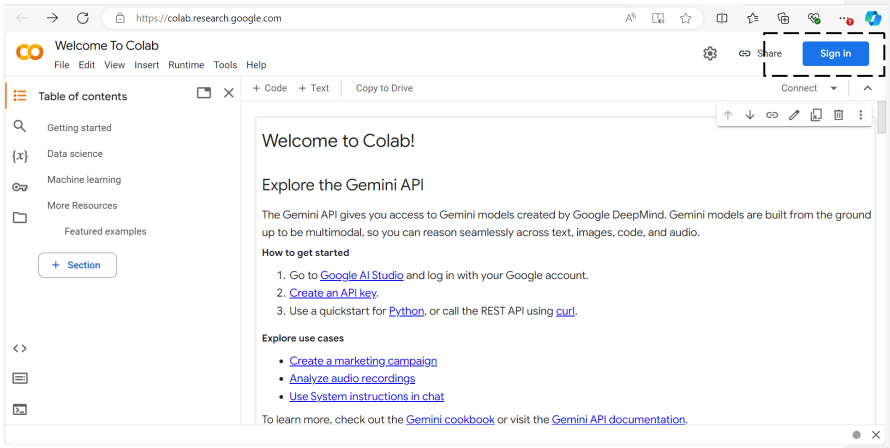
Step 3: Sign up for Google Colab

Overview

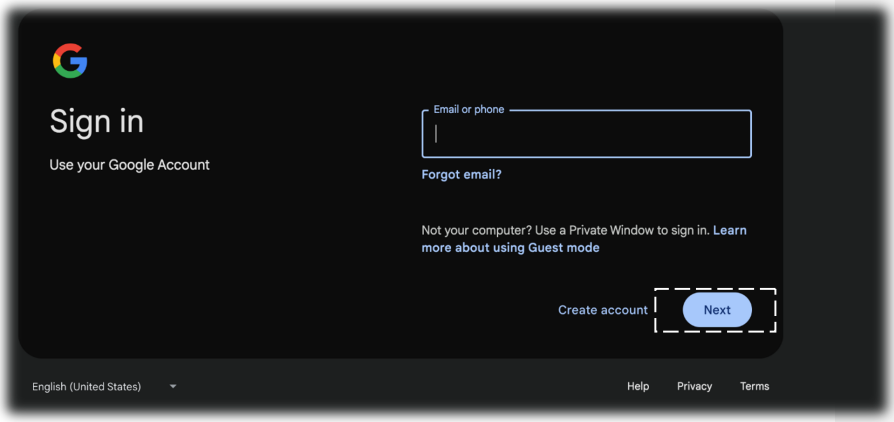
You will continue to set up the runtime environment by signing up for a Google Colab account. This will enable you to install and run the various utilities needed to run the lab in the Google Colab environment. Skip to Step 4 in this lab if you have signed up for Google Colab already.

Tasks

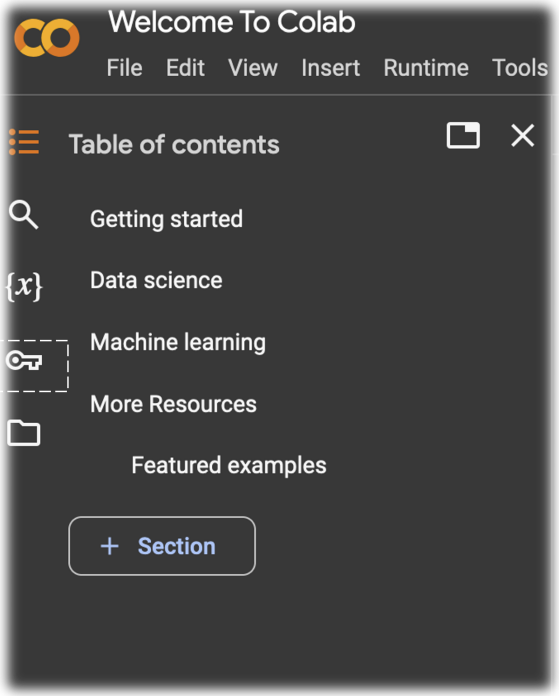
1. Visit the Google Colab website to sign up for a **Google Colab** account and select **Sign in**.



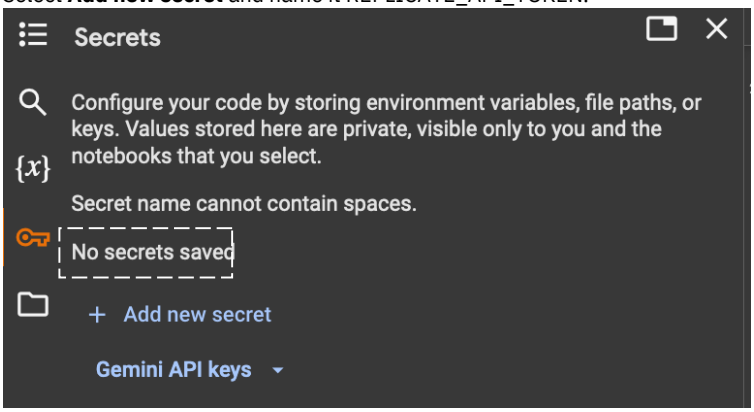
2. Type your email or phone number to create your account and select **Next** to sign in to Google Colab**.**



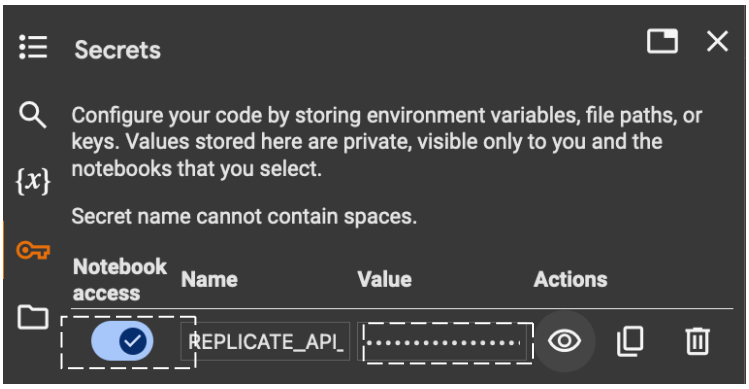
3. Next, you need to store your **Replicate API Token** in Google Colab secret to pass it in the code. In the left-hand sidebar, select the **Key (Secrets) icon**.



4. Select **Add new secret** and name it REPLICATE_API_TOKEN.



5. Paste your Replicate API token into the **Value** field and enable the “Notebook access” (using toggle button). Also, ensure that there are no spaces and Select the **Close** button to exit the configuration.



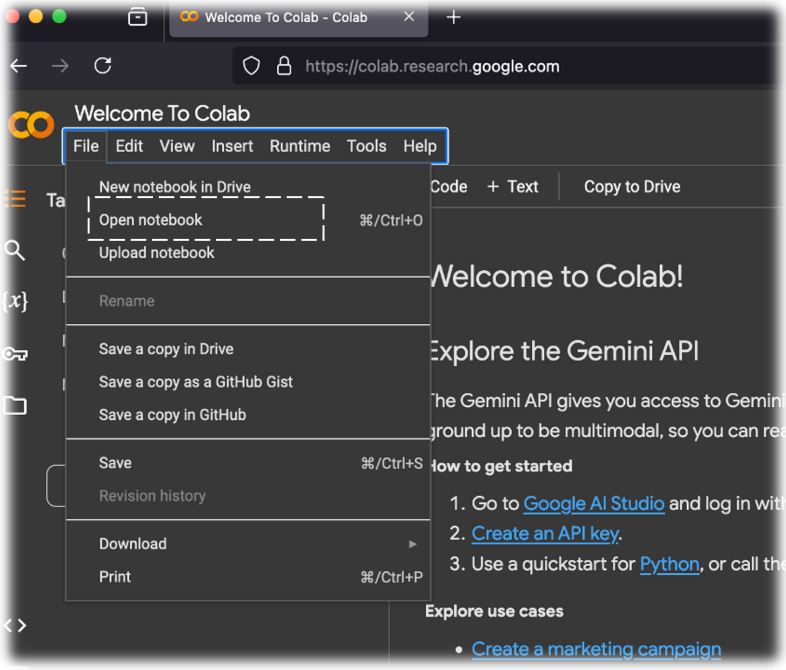
Step 4: Load the Jupyter notebook and initialize the model

Overview

A notebook is a shareable document that combines computer code, plain language descriptions, data and visualizations. In this step, you will load a Jupyter notebook file from GitHub into Google Colab and initialize the IBM Granite model. The notebook contains details of the few-shot prompts to optimize the initial output from Lab 1.

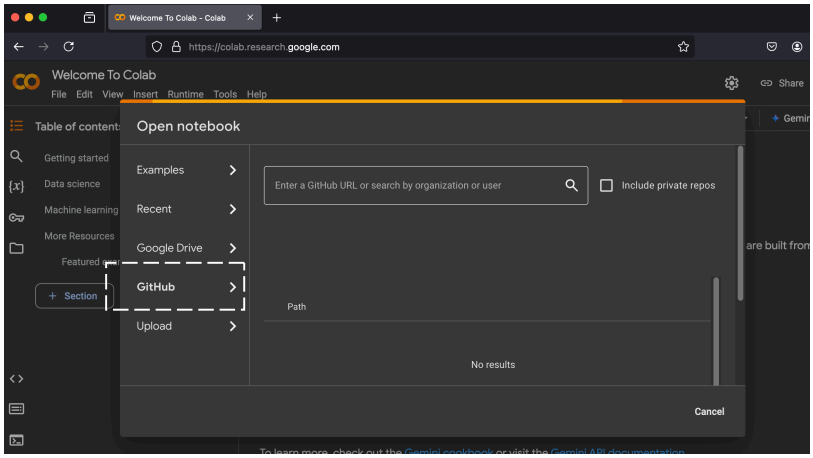
Tasks

1. In your Google Colab workspace, under the **File** menu, select **Open notebook**.

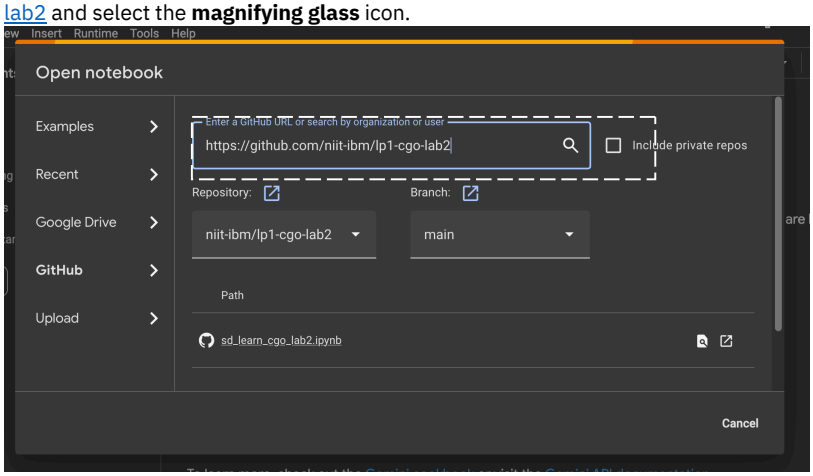


Developer note: Apply effect: drop shadow rectangle to all screenshots.

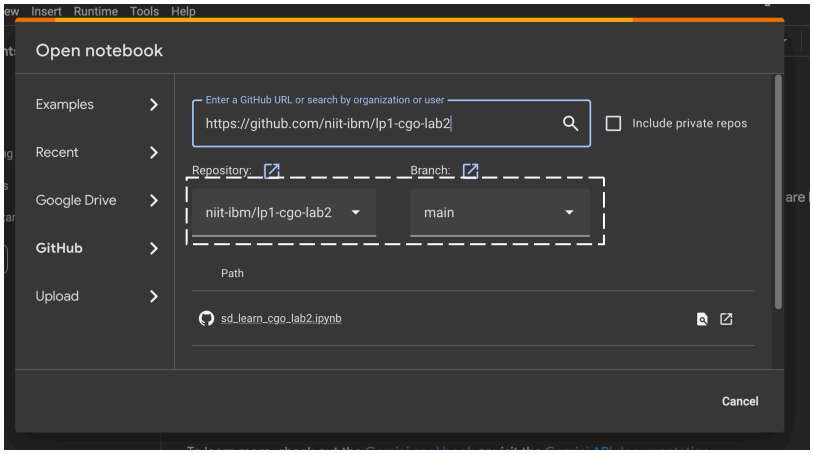
2. Select the **GitHub** tab.



3. In the **GitHub URL field**, type the following URL: https://github.com/niit-ibm/lp1-cgo-lab2 and select the **magnifying glass** icon.

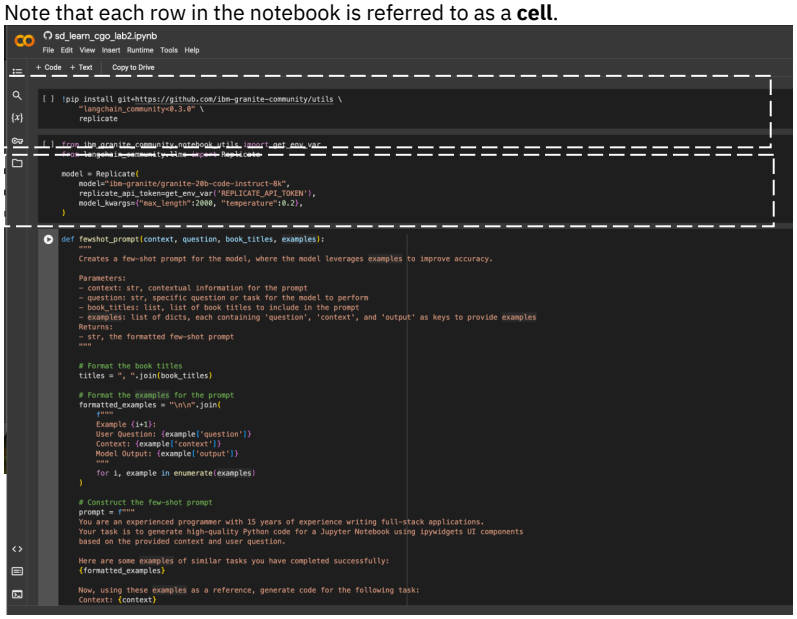


4. Select the **sd_learn_cgo_lab2.ipynb** notebook in the **Path** textbox. Select the **magnifying glass** icon to open the notebook in Colab.

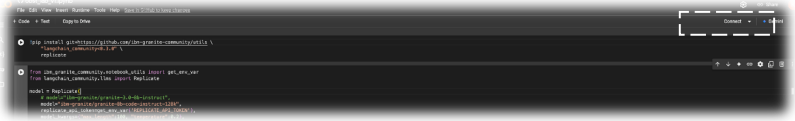


5. You will now have the **sd_learn_cgo_lab2** notebook opened in the Colab workspace.

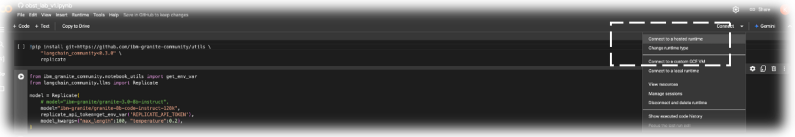
   Note that each row in the notebook is referred to as a **cell**.



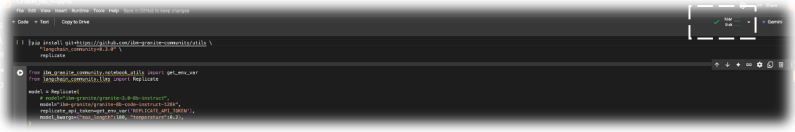
6. You will need to start a new instance in Google Colab runtime before you can run the code in the Jupyter notebook. Select **Connect** on the top-right corner of the Google Colab **navigation bar**.



7. Select **Connect to a hosted runtime** on the Connect menu.



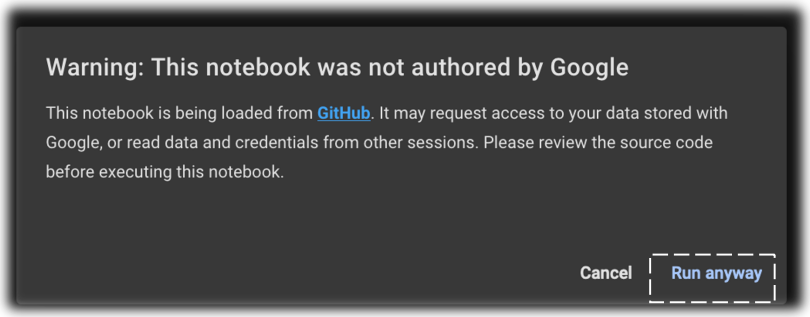
8. **A green checkmark** will indicate that you have successfully connected to the hosted runtime.



9. In the **first cell** of the notebook**,** select the **Run** cell button to install the required libraries from the Granite community.

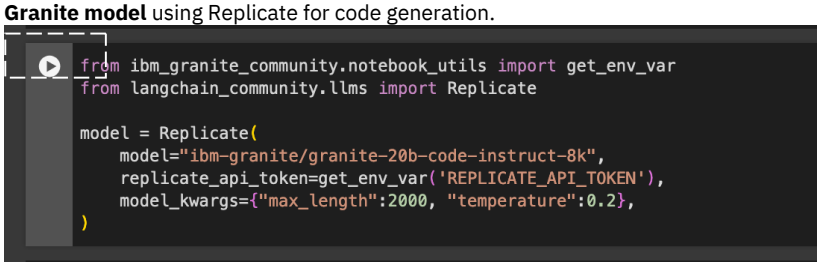
```
!pip install git+https://github.com/ibm-granite-community/utils \
    "langchain_community<0.3.0" \
    replicate
```

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



11. **A green checkmark** will appear beside the **Run** cell button to indicate that the required libraries have been successfully installed.

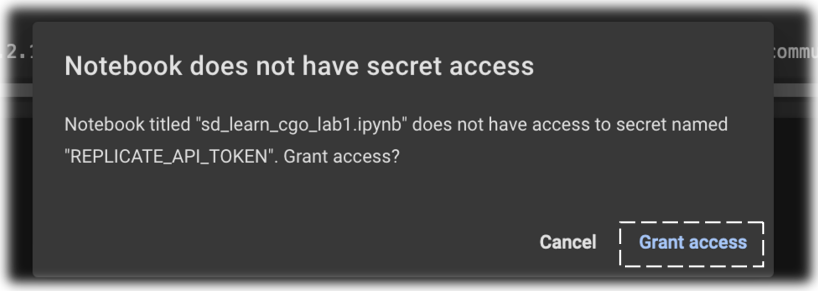
12. In the second cell of the notebook, select the **Run** cell button to initialize the **IBM Granite model** using Replicate for code generation.



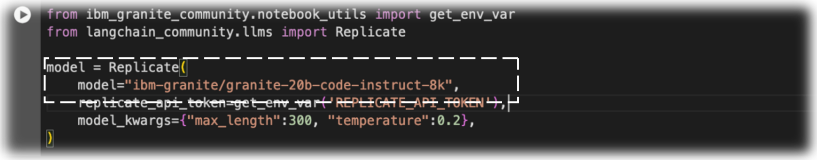
```
from ibm_granite_community.notebook_utils import get_env_var
from langchain_community.llms import Replicate

model = Replicate(
    model="ibm-granite/granite-20b-code-instruct-8k",
    replicate_api_token=get_env_var('REPLICATE_API_TOKEN'),
    model_kwargs={"max_length":2000, "temperature":0.2},
)
```

13. Select **Grant access** to proceed loading the model from Replicate.



14. The following message is displayed at the bottom of the cell “REPLICATE_API_TOKEN loaded from Google Colab secret.” **A green check mark** will appear beside the Run cell button to indicate that the IBM Granite model is initialized using Replicate.



You are now all set to prompt the model to generate code for the given scenario.

Step 5: Generate code using the IBM Granite model

Overview

After setting up the runtime environment and initializing the IBM Granite model using Replicate, you will prompt the model to optimize the code for the online bookstore scenario.

Tasks

1. You will now define a function that creates a prompt for the AI model using examples to improve accuracy. This function will help generate the code to create interactive UI components for the online bookstore.

   The few-shot prompt is structured to guide the AI model by providing context, examples of similar tasks, and specific instructions for the code generation.
   The key difference in the few-shot prompt is the inclusion of **examples** of similar tasks that the model has already completed successfully. These examples consist of past questions, context, and the generated output with specific styling and CSS details. The examples act as a guide for the model, allowing it to:
   - **Learn from past outputs**: The model uses the examples to identify common patterns in the code structure, styling, and formatting that were successful in previous tasks.
   - **Refine its output**: With examples, the model can generate code that is more likely to match the desired structure and styling, based on what worked well in the past.
   - **Ensure consistency**: By referencing these examples, the model ensures that the generated code follows consistent design principles and UI standards, including layout, color schemes, and component arrangement.

Commented [RC1]: Note for Kiran
Do learners need to run Cell 3 of the Notebook here. If yes, then we should add an instruction here for the learners.

```python
def fewshot_prompt(context, question, book_titles, examples):
    """
    Creates a few-shot prompt for the model, where the model leverages examples to improve accuracy.

    Parameters:
    - context: str, contextual information for the prompt
    - question: str, specific question or task for the model to perform
    - book_titles: list, list of book titles to include in the prompt
    - examples: list of dicts, each containing 'question', 'context', and 'output' as keys to provide examples
    Returns:
    - str, the formatted few-shot prompt
    """

    # Format the book titles
    titles = ", ".join(book_titles)

    # Format the examples for the prompt
    formatted_examples = "\n\n".join(
        f"""
        Example {i+1}:
        User Question: {example['question']}
        Context: {example['context']}
        Model Output: {example['output']}
        """
        for i, example in enumerate(examples)
    )

    # Construct the few-shot prompt
    prompt = f"""
    You are an experienced programmer with 15 years of experience writing full-stack applications.
    Your task is to generate high-quality Python code for a Jupyter Notebook using ipywidgets UI components
    based on the provided context and user question.

    Here are some examples of similar tasks you have completed successfully:
    {formatted_examples}

    Now, using these examples as a reference, generate code for the following task:
    Context: {context}
    User Question: {question}
    Include descriptions of elements from the book titles: {titles}
    Ensure that:
    - The code is well-structured and uses appropriate styling, coloring, and formatting for UI elements.
    - Ensure the `value` strings in all `widgets.HTML` components use triple double quotes for multi-line HTML.
    - The output code is optimized and does not exceed 300 tokens.
    - Output only the Python code.
    """
    return prompt
```

2. Define a function to get an answer using the few-shot prompt. This function generates a response from the AI model based on the few-shot prompt.

Commented [RC2]: Note for Kiran
Do learners need to Run Cell 4 of the notebook at this step.

```python
# Function to get answer using few-shot prompting
def get_answer_using_fewshot(context, question, book_titles, examples):
    """
    Generates the response from the model based on a few-shot prompt.

    Parameters:
    - context: str, contextual information for the prompt
    - question: str, specific question for the model to answer
    - book_titles: list, list of book titles to include in the prompt

    Returns:
    - str, the generated result from the model
    """
    prompt = fewshot_prompt(context, question, book_titles, examples)
    result = model.invoke(prompt)

    return result
```

3. Provide examples to guide the AI model. We provide examples of tasks the AI model has completed successfully to guide its responses.

Commented [RC3]: Note for Kiran
Should we ask learners to Run Cell 5 at this step.

```
examples = [
{
    "question": "Add a styled header for my bookstore landing page",
    "context": "Gradient Background and Font Styling for Header",
    "output": """
import ipywidgets as widgets
from IPython.display import display

# Create a styled header
header = widgets.HTML(value="
    <div style='background: linear-gradient(to right, #6a11cb, #2575fc); padding: 20px; border-radius: 8px;'>
        <h1 style='color: white; text-align: center; font-family: Arial, sans-serif;'>"Welcome to Reader's Verse"</h1>
    </div>
")

display(header)
"""
},
{
    "question": "Enhance my bookstore catalog with hover effects",
    "context": "Interactive Book Tiles with Shadows and Hover Animation",
    "output": """
import ipywidgets as widgets
from IPython.display import display

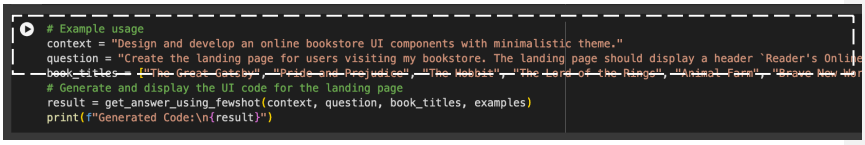
# Create book tiles with hover effects
book_tiles = [
    widgets.HTML(value="
        <div style='background-color: #ffffff; padding: 20px; margin: 15px; border-radius: 8px; transition: transform 0.2s; box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);'
             onmouseover="this.style.transform='scale(1.05)';"
             onmouseout="this.style.transform='scale(1.0)';">
            <h2 style='color: #444;'>The Great Gatsby</h2>
            <p style='color: #555;'><b>Author:</b> F. Scott Fitzgerald</p>
            <p style='color: #555;'><b>Price:</b> $10.00</p>
        </div>
    "),
    widgets.HTML(value="
        <div style='background-color: #ffffff; padding: 20px; margin: 15px; border-radius: 8px; transition: transform 0.2s; box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);'
             onmouseover="this.style.transform='scale(1.05)';"
             onmouseout="this.style.transform='scale(1.0)';">
            <h2 style='color: #444;'>Pride and Prejudice</h2>
            <p style='color: #555;'><b>Author:</b> Jane Austen</p>
            <p style='color: #555;'><b>Price:</b> $15.00</p>
        </div>
    "),
]

container = widgets.VBox(book_tiles)
display(container)
"""
},
```

4. **Generate and display the AI generated code**
Cell 6: Generate and display the UI code for a bookstore landing page

This method formulates a few-shot prompt where the model is guided with example inputs and outputs to simulate the approach of an experienced programmer.

The prompt includes pre-provided examples that demonstrate how to generate Python code for creating an ipywidgets-based UI for an online bookstore. By taking in inputs, guided example, context and question—it builds a comprehensive prompt that ensures the model understands and refines the required UI components for the bookstore interface.



The few-shot response builds on the zero-shot output by incorporating advanced design elements and interactivity. The response significantly refines the zero-shot output by leveraging examples and contextual guidance to produce a more polished and interactive design.

The inclusion of examples and contextual cues in the few-shot prompt enables the Granite model to generate a richer, more polished output.

Step 6: Validate the AI generated code

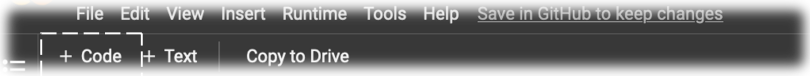
Overview

This task involves validating the Python code generated by the IBM Granite model to ensure it renders the intended UI components for the online bookstore application in a Jupyter Notebook environment.

Tasks

1. **Add a New Cell to the notebook.**
   In the Jupyter Notebook opened in Google Colab, select **+Code** to insert a new code cell.

Note: The new cell will be added as the last row in the notebook.



2. Copy and paste the **Python code** generated by the Granite model into the **new cell**.

```
import ipywidgets as widgets
from IPython.display import display
# Create a header
header = widgets.HTML(value="""
<div style='background-color: #333; color: white; padding: 20px; border-radius: 8px;'>
    <h1 style='text-align: center; font-family: Arial, sans-serif;'>Reader's Online Store</h1>
</div>
""")
# Create a welcome message
welcome = widgets.HTML(value="""
<div style='padding: 20px; border-radius: 8px;'>
    <h2 style='color: white; text-align: center;'>Welcome to Reader's Verse</h2>
</div>
""")
# Create book tiles
book_titles = ["The Great Gatsby", "Pride and Prejudice", "The Hobbit", "The Lord of the Rings", "Animal Farm", "Brave New World"]
book_tiles = []
for i, title in enumerate(book_titles):
    background = "#f5f5f5" if i % 2 == 0 else "#eaeaea"
    book_tiles.append(widgets.HTML(value=f"""
<div style='background-color: {background}; padding: 15px; margin: 10px; border-radius: 8px;'>
    <h2 style='color: #333;'>{title}</h2>
    <p style='color: #555;'>Author: Author Name {i+1}</p>
    <p style='color: #555;'>Price: ${(i+1)*10}.00</p>
</div>
"""))
# Create a container for the book tiles
book_container = widgets.VBox(book_tiles)
# Create a vertical box for the header, welcome message, and book container
main_container = widgets.VBox([header, welcome, book_container])
# Display the main container
display(main_container)
```

3. **Execute the code.**
Select the **Run** cell button on the newly added cell. The output should display the bookstore's landing page UI below the code cell.

The output is significantly refined from the zero-shot output by leveraging examples and contextual guidance to produce a more polished and interactive design.

- Advanced Styling: Includes a dark-themed header, well-padded welcome message, and CSS-like design elements for a professional look.
- Enhanced Components: Features dynamic book tiles with alternating backgrounds, detailed metadata (e.g., author, price), and improved readability.
- Modular Layout: Organized components allow for easier updates and modifications.
- E-Commerce Context: Adds value through thoughtful integration of commerce-related details like pricing and authorship.

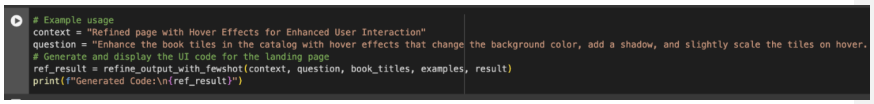
Step 7: Refine the generated output

Overview

Refine the output of the fewshot prompt by applying additional user-requested changes.

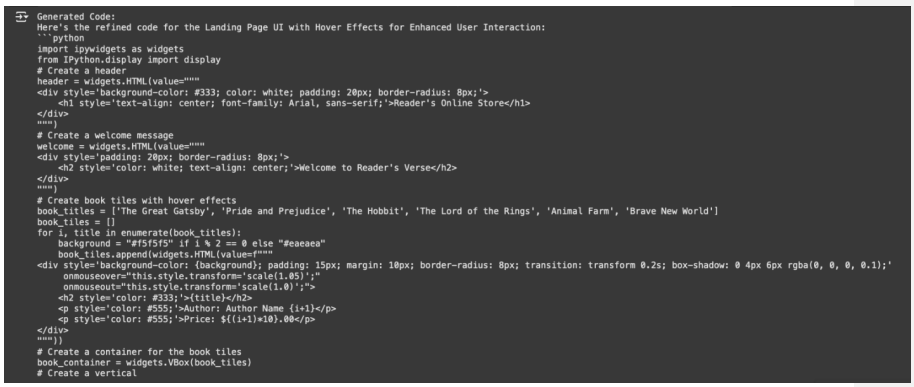
1. This function refines the previously generated output. The prompt here focuses on enhancing existing outputs generated by the few-shot prompt. It builds upon the initial design by incorporating user-requested refinements, leveraging examples of similar tasks for guidance. This approach ensures the code adheres to specific requirements such as styling consistency, improved formatting, and advanced interactivity. Key instructions in the prompt emphasize maintaining a cohesive design while incorporating enhancements like hover effects, dynamic responsiveness, and additional functionality. The prompt exemplifies how iterative improvement, combined with context-driven refinement, can optimize AI-generated outputs for specific user needs.

2. **Generate and display the refined UI code**This function refines the previously generated output of fewshot prompt by applying additional user-requested changes.



The refined output transforms the few-shot-generated code into a more interactive and polished version. Key improvements include the addition of hover effects for book tiles, creating a dynamic user experience. The use of shadow effects, smooth transitions, and responsive designs enhances the visual appeal and interactivity of the landing page. Consistent formatting and modular components ensure a clean, professional layout. These refinements showcase how targeted adjustments based on user feedback can

significantly enhance the usability and aesthetic quality of AI-generated code.

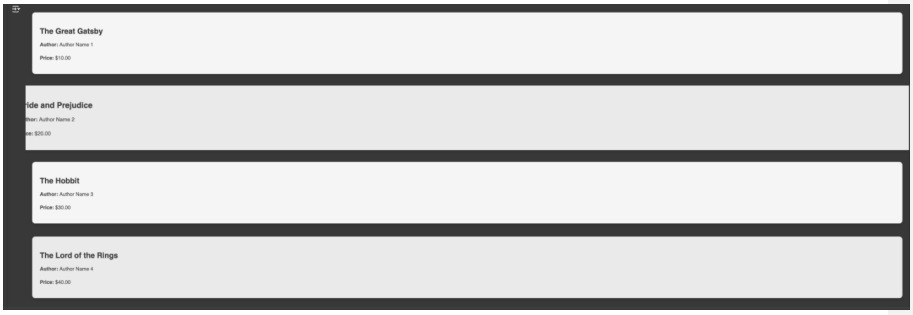


```
Generated Code:
Here's the refined code for the Landing Page UI with Hover Effects for Enhanced User Interaction:
```python
import ipywidgets as widgets
from IPython.display import display
# Create a header
header = widgets.HTML(value="""
<div style='background-color: #333; color: white; padding: 20px; border-radius: 8px;'>
    <h1 style='text-align: center; font-family: Arial, sans-serif;'>Reader's Online Store</h1>
</div>
""")
# Create a welcome message
welcome = widgets.HTML(value="""
<div style='padding: 20px; border-radius: 8px;'>
    <h2 style='color: white; text-align: center;'>Welcome to Reader's Verse</h2>
</div>
""")
# Create book tiles with hover effects
book_titles = ['The Great Gatsby', 'Pride and Prejudice', 'The Hobbit', 'The Lord of the Rings', 'Animal Farm', 'Brave New World']
book_tiles = []
for i, title in enumerate(book_titles):
    background = "#f5f5f5" if i % 2 == 0 else "#eaeaea"
    book_tiles.append(widgets.HTML(value=f"""
<div style='background-color: {background}; padding: 15px; margin: 10px; border-radius: 8px; transition: transform 0.2s; box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);'
     onmouseover="this.style.transform='scale(1.05)';"
     onmouseout="this.style.transform='scale(1.0)';">
    <h2 style='color: #333;'>{title}</h2>
    <p style='color: #555;'>Author: Author Name {i+1}</p>
    <p style='color: #555;'>Price: ${(i+1)*10}.00</p>
</div>
"""))
# Create a container for the book tiles
book_container = widgets.VBox(book_tiles)
# Create a vertical
```

3. Run the cell in book-store.ipynb to display the bookstore's landing page UI with the title, catalog tiles, and search bar.

```
import ipywidgets as widgets
from IPython.display import display
# Create a header
header = widgets.HTML(value="""
<div style='background-color: #333; color: white; padding: 20px; border-radius: 8px;'>
    <h1 style='text-align: center; font-family: Arial, sans-serif;'>Reader's Online Store</h1>
</div>
""")
# Create a welcome message
welcome = widgets.HTML(value="""
<div style='padding: 20px; border-radius: 8px;'>
    <h2 style='color: white; text-align: center;'>Welcome to Reader's Verse</h2>
</div>
""")
# Define book tiles with modern styling and hover effects
book_titles = ['The Great Gatsby', 'Pride and Prejudice', 'The Hobbit', 'The Lord of the Rings', 'Animal Farm', 'Brave New World']
book_tiles = []
for i, title in enumerate(book_titles):
 background = "#f5f5f5" if i % 2 == 0 else "#eaeaea"
 book_tiles.append(widgets.HTML(value=f"""
 <div style="background-color: {background}; padding: 20px; margin: 15px; border-radius: 8px; transition: transform 0.2s; box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);"
 onmouseover="this.style.transform='scale(1.05)';"
 onmouseout="this.style.transform='scale(1.0)';">
 <h2 style="color: #444;">{title}</h2>
 <p style="color: #555;"><b>Author:</b> Author Name {i+1}</p>
 <p style="color: #555;"><b>Price:</b> ${(i+1)*10}.00</p>
 </div>
 """))
container = widgets.VBox(book_tiles)
display(container)
```

The UI should appear below the code cell (Similar to below), allowing for interaction directly in the notebook
```



- **Hover Effects**: Book tiles now respond to user interactions with a smooth scaling effect, enhancing interactivity.
- **Shadowing and Transition**: Shadow effects and transitions provide a modern, professional look.
- **Refined Layout**: Maintains a clean, well-organized design with consistent formatting and spacing.

Conclusion

- You successfully used an IBM Granite model to Optimize code for creating the landing page of a website for ReadersVerse, a hypothetical bookstore. The client is delighted to see how fast you come up with a working prototype of the UI for the welcome page of the website. You successfully used fewshot prompts to generate a python code output from the IBM Granite model and validated the AI generated code to ensure that it correctly renders the expected output.