Simulation 1: Introduction to Jupyter Notebooks through Amazon SageMaker

Simulation overview and objectives

This simulation teaches you about the basic functionality and use cases of Jupyter notebooks.

A *Jupyter notebook* is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. Uses include data cleaning and transformation, numerical simulation, statistical modeling, data visualization, and machine learning.

After completing this simulation, you will know how to do the following:

- Understand the fundamentals of using a notebook
- Write and run basic code and Markdown in a notebook
- Export and share a notebook

Duration

This simulation requires approximately **30 minutes** to complete.

Task 1: Open the notebook instance

In this task, you will learn how to access Jupyter notebook instances within the Amazon SageMaker console. You will also open an existing notebook instance.

A SageMaker notebook instance is a machine learning (ML) compute instance that is running the Jupyter Notebook application. SageMaker manages creating the instance and related resources. Use Jupyter notebooks in your notebook instance to prepare and process data, write code to train ML models, deploy models to SageMaker hosting, and test or validate your ML models.

1. Choose the search box and enter in **Amazon SageMaker**.

2. Choose **Enter**.

3. Choose **Amazon SageMaker** from the menu.

4. In the left navigation pane, expand **Notebook**, and choose **Notebook instances**.

5. This page lists all notebook instances. Running notebooks have a **Status** of *InService*.

6. For the **SageMakerNotebookInstance**, in the **Actions** column, choose **Open Jupyter**.

The notebook instance opens.

Task 2: Create a notebook

In this task, you will create a notebook in the SageMaker notebook instance by using the Python kernel. You will then learn about the basic features and functionality of notebooks, such as different cell types.

7. Choose **New** in the upper-right corner, and then choose **conda_python3**.

A new notebook opens, and the Python 3 kernel starts. The kernel provides the ability to run Python code in this notebook.

Note: A kernel is a set of programming language-specific processes that run independently and interact with the Jupyter application.

Congratulations, you have created your first notebook!

Menus

In the notebook, review the options in the menu bar:

- **File:** Create, open, save, rename, download, or close a notebook.
- **Edit:** Cut, copy, paste, delete, split, merge, or move cells.
- **View:** Toggle the header, toolbar, or line numbers and choose the cell toolbar type.
- **Insert:** Insert a cell above or below the selected cell.
- **Cell:** Run a cell or all cells, and choose the cell type.
- **Kernel:** Interrupt, restart, reconnect, shut down, or change the kernel.

- **Widgets:** Save or clear the widget state.
- **Help:** Access resources for additional information.

Notebook name and file type

8. To change the title of the notebook, choose **untitled** in the upper-left corner.
9. For the notebook name, enter `MySimulationNotebook`
10. Choose **Rename**.

By default, the default file type for a notebook is **.ipynb**. A .ipynb file is a text file that describes the contents of your notebook in a format called JavaScript Object Notation (JSON). Each cell and its contents, including image attachments that have been converted into strings of text, is listed in the file along with some metadata.

Code cells

11. Choose the first notebook cell, and enter the following Python code:

```
print('Hello Jupyter!')
```

Note: A cell is a container to display text or run code.

One of the notebook features is *autocomplete*. For example, if you type `pri` and then press Tab, the text completes to `print`.

12. On the toolbar, choose **Run** to run the code in the cell.

Note: To run the code in the cell in a live Jupyter environment, you can also press Shift + Enter.

After you run the cell, the output displays just below the code. Notice that the cell's label also changed from **In [ ]** to **In [1]**. A new cell also appears below the output.

13. In the new cell, enter the following code:

```
print('This is a new cell.')
```

Next, you will write some code that will not return results immediately. When you are working with a large dataset in a notebook, code will often take some time to run.

14. Choose the new cell to populate the code below, which will show you what the notebook looks like when code is running in the background.

    Note: In a live Jupyter environment you would enter and run the following code in a code cell.

```
import time
time.sleep(10)
```

15. Choose **Run** to run the code.

    While the code is running, notice that the cell label displays **In [*]** with an asterisk in the brackets, as shown in the following image. This indicates that the kernel is busy, and the cell is currently running the task.

16. Choose the new cell to populate the following code, which performs a few basic arithmetic operations.

    Note: In a live Jupyter environment you would enter and run the following code in a code cell.

```
a = 12
b = 2
print(a + b)
print(a**b)
print(a/b)
```

Markdown cells

The primary cell types in a notebook are code and Markdown. A Markdown cell displays text, and you can use the Markdown markup language to format the text.

17. Choose the new cell that was just created.
18. On the toolbar, choose **Code** and choose **Markdown**.

The type for the current cell changes from code to Markdown.

19. Choose the new Markdown cell to see how the following is shown as text.

Note: In a live Jupyter environment you would enter and run the following code in a code cell.

```
# Technical Report

Here is my technical report.
```

20. To run the cell and render the Markdown, choose **Run**.

By using a combination of code cells and Markdown cells, you can create a document that runs Python code and includes text to explain what the code does.

21. Choose the new cell, and change the cell type to Markdown.
22. Choose the new Markdown cell to see the formatted text for how the headers will look.

Note: In a live Jupyter environment you would enter and run the following code in a code cell.

```
# Header 1
## Header 2
### Header 3
#### Header 4
```

The output shows the different heading levels that you can use in Markdown.

23. Choose **Run.**
24. Choose the new cell, and change the cell type to Markdown.
25. Choose the new Markdown cell to see how the item list looks.

Note: In a live Jupyter environment you would enter and run the following code in a code cell.

```
- Item 1
- Item 2
- Item 3
```

With Markdown, you can also insert a code example for your reader to review but not run. For inline code, surround the code with backticks (`).

To insert a code block, use three backticks (```) before and after the code block. A best practice is to specify the code language after the first three backticks.

26. To create a code block, enter and run the following text in a **Markdown** cell:

~~~
```python
s = 'Syntax highlighting for Python'
print(x)x ```
~~~

Shell commands

You can also run shell commands within code cells. In the next few steps, you will run a few basic Linux commands.

27. Choose the new cell to see the following command:

```
ls -l
```

28. Choose **Run**.

The output shows the contents of the folder that the notebook is in.

To write and run multiple lines of bash code, use the `%%bash` prompt before your commands.

29. Choose the new cell to see the following commands:

```
%%bash
pwd
ls -l
date
```

30. Choose **Run.**

The output displays the current working directory, contents of the folder the notebook is in, and current date and time.

Note: For a list of keyboard shortcuts and commands, choose the keyboard icon on the toolbar, as shown in the following image.

Task 3: Export the notebook to HTML

Jupyter has built-in support to export to HTML, PDF, and several other formats. In this task, you will download your notebook as HTML and open it.

Before you download the notebook, you will first clear the outputs and re-run all of the cells. Clearing the outputs helps to ensure that the notebook doesn't contain intermediary output or have a stale state. It also helps to ensure that everything runs as it should if you share the notebook.

31. To clear the output for all cells, choose **Cell** > **All Output** > **Clear**.
32. To restart the kernel and re-run all of the cells, choose **Kernel** > **Restart & Run All**.

Wait for the cells to finish running. Make sure that the brackets in each cell label contain a number and that all cells ran successfully.

33. To export the notebook, choose **File** > **Download as** > **HTML (.html)**.

Download the file to your computer.

34. In a web browser, open and review the HTML file that you just downloaded.
35. Return to the notebook and download it as a **Notebook .ipynb** file.

Now you have two files. The HTML file is convenient to view and share quickly. You can upload or share the notebook (.ipynb) file as needed.

Summary

In this simulation, you created a Jupyter notebook and learned about notebook functionality. You learned that you can use a notebook to learn or teach a programming language, such as Python. You can also use a notebook to share data.

Simulation complete

Congratulations! You have completed the simulation.

36. Choose **End Simulation** at the top of this page, and then select **Yes** to confirm that you want to end the simulation.

A panel indicates that *DELETE has been initiated... You may close this message box now.*

37. To close the panel, choose the **X** icon in the upper-right corner.

Additional resources

For more information about AWS Training and Certification, see https://aws.amazon.com/training/.

Your feedback is welcome and appreciated.

To share any suggestions or corrections, use the AWS Training and Certification Contact Form.