**Introduction to Jupyter Notebooks (AWS SageMaker)**

**Introduction**

Jupyter Notebooks are the standard tool for data scientists to interact with and manipulate data. This hands-on lab covers the basic structure of a notebook, how to execute code, and how to make changes. You'll also get to create a simple machine learning model and use it to make inferences. This lab uses AWS SageMaker Notebooks and provides you with the foundational knowledge required to use this service for more advanced topics.

You are provided some sample heights and weights of individual penguins. We're going to use the data (stored in a csv file) to train a data model and create inferences so that you can make guesses about what a penguin weighs if you know its height.

The files used in this lab can be found [here on GitHub](https://github.com/linuxacademy/content-aws-mls-c01).

**Navigate to Jupyter Notebook**

Log in to AWS with the credentials provided on the hands-on lab page.

1. In the *Search for services* box, search for **SageMaker**. It should be the first item that pops up in the list.
2. Once you're in the SageMaker dashboard, select **Notebook** > **Notebook instances** from the left-hand menu. You should see 1 notebook available. If you do not see any notebooks, double-check to make sure that you're in the *N. Virginia* region (using the dropdown in the upper right of the dashboard).
3. Click **Open Jupyter** on the right side to open up the notebook you'll be working with.

**Browse Jupyter Notebooks**

When you first open a notebook, notice that it looks similar to a file explorer window. You can select items in the window, and they'll open up in new tabs. For example, if you click an image, that image will open up in a new browser tab. The actual Jupyter notebook file has a .ipynb file extension.

On the right-hand side of the screen, there are a couple of buttons: **New** and **Upload**.

**Create New Folders and Files**

In the notebook's directory (where you can see all of the files associated with this notebook), make a new directory:

1. Click **New** in the upper right.
2. At the bottom of the menu, select **Folder**.
3. Note that an *Untitled Folder* is now displayed in the directory.
4. To rename it, select the checkbox next to it, and then click **Rename**, above the list of files and folders.
5. Type my\_new\_folder, and click **Rename**.
6. Click the folder's name to open it. To get back out, either click the double-dot link (the **..**), or click somewhere in the blue path above (currently, a picture of a folder, followed by / my\_new\_folder).

**The Notebook File**

Out in the main directory, select the notebook (the ipynb file), which will then open it in a new tab. If you navigate back to the other tab you were in, notice that the notebook icon is now green. That means it's now running. If you select the **Running** tab at the top, you can see a list of all running notebooks.

**The Notebook Structure**

**Note:** This notebook is already built for you and includes several blocks or cells of different types. As you continue through the lab, you will work with the code that is provided in each cell. At the end, you will be provided with instructions on how to build your own notebook. Feel free to use this file to explore and edit the existing code.

Notebook files contain 3 different types of cells: *Code*, *Markdown*, or *Raw NBConvert*. In this lab, you will work with the Code and Markdown types. Raw NBConvert cells are used for converting the output to something else (e.g., HTML or PDF via LaTeX).

**Use Markdown to Add Richly Formatted Text to a Notebook**

If the cell is Markdown, you can type Markdown-formatted text into it. [See this site](https://daringfireball.net/projects/markdown/syntax) for more information about Markdown syntax). Try adding a new Markdown cell to the file:

1. With the **1) Markdown** cell highlighted, open the **Insert** menu.
2. Select **Insert Cell Below**. By default, you'll get a new code cell.
3. At the top of the screen, click the dropdown that currently says **Code**, and select **Markdown**.
4. Use the following syntax to add an image to the cell: ![Pinehead](pinehead.jpg).
5. Click **Run**. You'll see the image with an alt tag of *Pinehead*. In this case, the output displays the HTML rendering of what you typed with Markdown.

**Code Cells**

If it's a Code type of cell, then you can type some executable code in the cell. Review the cell in the **2) Command-Line Operations** section of the notebook. It looks like this:

!whoami

!which python

The exclamation points at the beginning of each line tell the cell to run the commands — just like in the command line. Just like before, with the cell highlighted, click **Run**.

Below the code cell, you'll see the results:

ec2-user

/home/ec2-user/anaconda3/envs/python3/bin/python

**Use a Code Cell to Evaluate the Output of Python Code**

Add a new cell below the previous one, and use the following syntax to find your Python version:

!python --version

The output will show you which version of Python is being used. This notebook has been created with a Python3 kernel, so you can run actual Python code in it, without the preceding !python tags.

For example, scroll down, and you can see your notebook contains the following syntax:

words = ['awesome', 'amazing', 'great']

for w in words:

print('This Linux Academy lab is %s!' % w)

With the cell highlighted, click **Run**, and you should see the following output:

This Linux Academy lab is awesome!

This Linux Academy lab is amazing!

This Linux Academy lab is great!

**Python Lists**

One of the primary reasons to use Jupyter Notebooks is data manipulation. Next, you'll get to work with a few different methods to do that.

Your notebook contains the following list syntax:

myList = [0, 1, 2, 3, 4, 5]

myList

Here you are setting a variable called myList. If you run the code cell, the output will be [0, 1, 2, 3, 4, 5].

In the next cell, you are adding 'blue' (a string) to the end of your list of integers.

myList.append('blue')

myList

The above syntax produces [0, 1, 2, 3, 4, 5, 'blue'] as the output.

If you run myList[3], Python will return the third data point in the list (3 in this case). Running myList[3:] will return everything from the third point onward: [3, 4, 5, 'blue'].

Finally, run the following syntax from the last cell in this section: len(myList). You should see the length of your list in the output (7).

**NumPy**

NumPy is a package designed to perform scientific computing with Python. Run the cell to import it. You can use this package to complete calculations such as printing out the value of *pi*.

import numpy as np

**Note:** Here you are calling numpy as np, so instead of having to run numpy, you'll just have to run np.

Next, run the following syntax in the cell below: np.pi. This will print out the value of the number pi.

Now, you can use it for performing some actual calculations too. You can find the circumference of a circle using the 2*pi*r equation, providing a radius of **10**. Run the following syntax from the code cell:

radius = 10

circumference = 2 \* np.pi \* radius

circumference

After running the code, you will get the value of the circumference (**62.831...**).

**NumPy Arrays**

You can also manipulate data using NumPy arrays. You can pass in what's essentially a list of lists, and then perform other calculations with that data.

data = np.array([['','Col1','Col2'],

['Row1',1,2],

['Row2',3,4],

['Row3',5,6]])

print(data)

If you run this cell, you'll get back pretty much what you put in. data now holds this array.

Taking this a step further, with something like print(data[1:,1:]), you are essentially selecting exactly what you want to get back. This syntax will output the data below the first row (the column headers, essentially) and to the right of the first column:

[['1' '2']

['3' '4']

['5' '6']]

**pandas**

Now you will work with the open-source library pandas. Like you did with NumPy, import pandas as pd:

import pandas as pd

Once you've run that, you can run the next code cell:

df = pd.DataFrame(data=data[1:,1:],

index=data[1:,0],

columns=data[0,1:])

df

What you've done is created a pandas DataFrame. It's another way to store data. This particular DataFrame takes all of the data from your NumPy array, then gives its rows' titles the same names that the array had. It also names the rows with the same names as those corresponding rows in the array. Click **Run**, and a richly formatted HTML table is displayed, complete with row highlighting.

**Loading Data From an External Source**

Navigate back to the **Home** tab. From the list, open the csv file. You will see a set of data points, 2 per line, separated by commas. This is the list of penguin heights and weights. Now, you will put them to use!

Back in the notebook, note that you have the following syntax:

penguin\_data = pd.read\_csv("penguin-data.csv")

penguin\_data.shape

Run this first cell to import the csv file. The output will also give you an overview of the file because of the penguin\_data.shape statement. After running this cell, note the output is 20,2, which means it has 20 rows and 2 columns.

The next cell contains the following syntax:

penguin\_data.head()

Now, when you run this second cell, you'll get the first 5 rows of the data, in a formatted HTML table. You can put numbers between the parentheses (e.g., penguin\_data.head(2)) to pull a different number of rows.

**Matplotlib**

Matplotlib is another Python library that works together with pandas and pandas DataFrames. Run the following cell to import the library as plt:

import matplotlib.pyplot as plt

%matplotlib inline

penguin\_data.plot(kind='scatter',x='Height',y='Weight',color='red')

The result is a plotted graph of the data, instead of a table. Observe any patterns such as the linear relationship between the height and weight of the penguins in the dataset.

**Use scikit-learn to Build a Simple Machine Learning Model**

scikit-learn is a native Python library used to create machine learning models. In this scenario, you're going to create a model that creates a *line of best fit*, for the data points in the graph. Run the following cell:

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

# Set up the Linear Regression model

model = LinearRegression()

# Train the model with our data

model.fit(penguin\_data[['Height']], penguin\_data['Weight'])

The following actions occur when this cell is run:

* The code calls in the appropriate libraries (the first 2 lines).
* It sets up a Linear Regression model.
* The data is run through and fit to the model, based on the model.fit line.

Note that it may take a few seconds for the output to display. The output may look like a failure (because the Out [21]: text is red) — but it is in fact totally fine! This output is simply a linear regression model, with some of the attributes included in the output.

Now you can create a new graph, using the model. This graph will include the line of best fit, drawn over the data points. Run the following cell:

# Plot our original training data

axes = plt.axes()

axes.scatter(x=penguin\_data['Height'], y=penguin\_data['Weight'])

# Determine the best fit line

slope = model.coef\_[0]

intercept = model.intercept\_

# Plot our model line

x = np.linspace(10,20)

y = slope\*x+intercept

axes.plot(x, y, 'r')

# Add some labels to the graph

axes.set\_xlabel('Height')

axes.set\_ylabel('Weight')

plt.show()

You should see a graph of data points with the line of best fit.

Using that line, you can make a prediction. You can guess what a penguin would weigh based on its height — or the other way around!

The next code cell performs this same action. This syntax takes a height of **14** and then provides an estimate of what that penguin will weigh:

height = 14

# Reshape the hight into an array

new\_height = np.reshape([height],(1, -1))

# Pass the new height to the model so that a predicted weight can be infered

weight = model.predict(new\_height)[0]

# Print the information back to the user

print ( "If you see a penguin thats %.2f tall, you can expect it to be %.2f in weight." % (height, weight))

The output is displayed in the form of a sentence indicating that a **14** tall penguin (note you never specified a unit of measure — this could be inches, cm, etc.) will weigh **18.84**. If you look at the newest graph, with the line, you'll see that this result matches the graph. Try another calculation, changing **14** to another number. Your result should match what the line of best fit says.

**Build Your Own Notebook**

In this hands-on lab, you built a machine learning model that tells you approximately what a penguin should weigh, based on its height. Spend the remaining time in the lab creating your own notebook.

1. Back in the main Jupyter tab (where you can see the list of files associated with the notebook), click **New**.
2. From the dropdown list, select **conda\_python3**. This creates a brand new notebook, using the Python3 kernel.
3. Click on the word **Untitled** at the top of the screen, and give the notebook a title.
4. Click **Rename** to update the title.
5. Note that the first item in the notebook is a code cell. Change this to be a Markdown type cell, just so you can let people know what the notebook is about. From here, feel free to use the tools and methods discussed earlier to do whatever you want in the notebook.
6. As the time you have left on the lab comes to an end, click **File** > **Download as** > **Notebook (.ipynb)**. You can download the notebook to your local machine, and keep working in your own Jupyer Notebook server or again on this server.

**Conclusion**

Congratulations — you just learned how to work with Jupyter Notebooks and create your own notebook!

You'll need to have a Jupyter Notebook server if you're going to play with the file locally – but don't be afraid to run the lab again and just upload the file to it. If you want to keep working with the same file, just download the newest version of your notebook file, then spin up the lab again and repeat the process, picking up where you left off. Good luck!