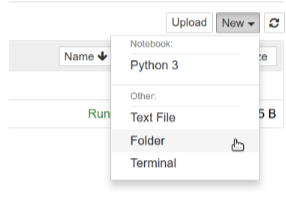
**Tapalla ,John Harven M.  
BSCS 3C**

**Jupyter Notebook**

**Adding Folders**In the upper right-hand corner of the Jupyter Notebook home screen, click on the "New" drop-down button and select "Folder". A new folder called "Untitled Folder" will appear in the list of files on the Jupyter Notebook home screen.

**Adding Text Files:** You can use the open() function to create and write to text files. This function takes the file path and access mode ('w' for writing) as arguments.

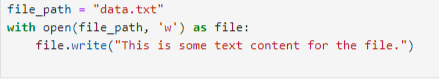
Example:

# Create a new text file named "data.txt"

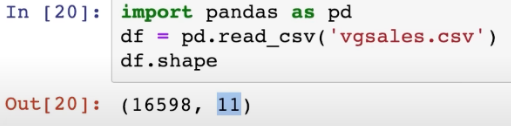
file\_path = "data.txt"

with open(file\_path, 'w') as file:

file.write("This is some text content for the file.")



**CSV file for data analysis and visualization**

CSV (Comma-Separated Values) files are a popular format for storing tabular data in a way that's easily readable by both humans and computers. They are ideal for data analysis and 

visualization in Jupyter Notebooks because of their simplicity and widespread compatibility.

***To Write and Call Dictionary Methods***

Creation of New Dictionary: You can create a dictionary using curly braces {} and specifying key-value pairs separated by colons. For example:

my\_dict = {'name': 'Alice', 'age': 30, 'city': 'New York'}

Accessing Items in the Dictionary**:** Use the key within square brackets [] to access the corresponding value.

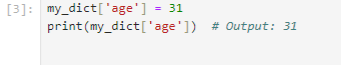
name = my\_dict['name']

print(name) # Output: Alice  


Change Values in the Dictionary:Assign a new value to the key within square brackets.

my\_dict['age'] = 31

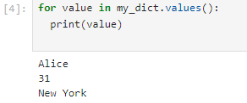
print(my\_dict['age']) # Output: 31



**Loop Through Dictionary Values:** Use a for loop to iterate over the values in the dictionary.

for value in my\_dict.values():

print(value)

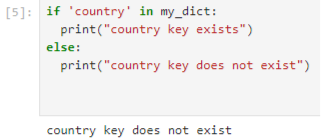
Check if Key Exists in the Dictionary**:** Use the in operator to check if a key exists.

if 'country' in my\_dict:

print("country key exists")

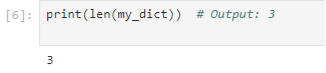
else:

print("country key does not exist")

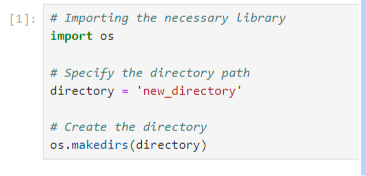


Checking for Dictionary Length**:** Use the len() function to get the number of key-value pairs.

print(len(my\_dict)) # Output: 3

**To Create a directory using Jupyter notebook**

Use the built-in Python functions for file operations. You can execute shell commands directly from Jupyter Notebook cells by prefixing the command with an exclamation mark!.

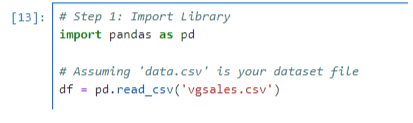
**To Import Libraries**

import pandas as pd: This line imports the Pandas library and gives it the alias pd, which is a common convention. This alias makes it easier to refer to Pandas functions and objects in your code by using pd as a prefix.



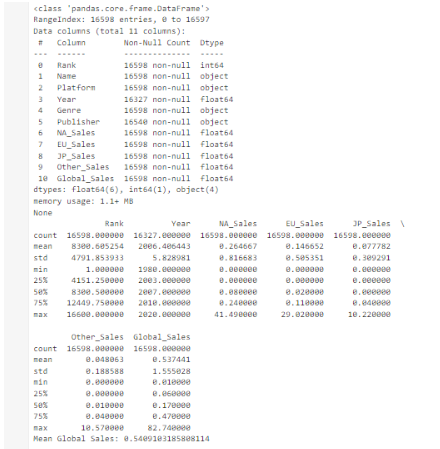
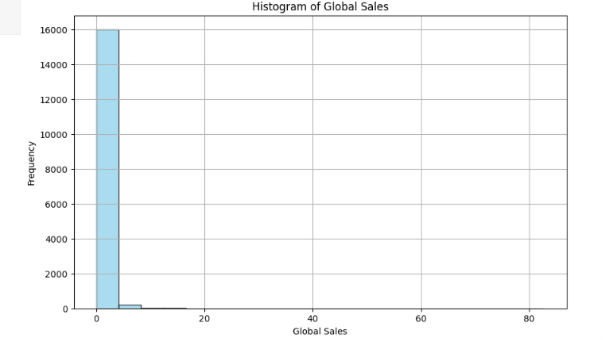
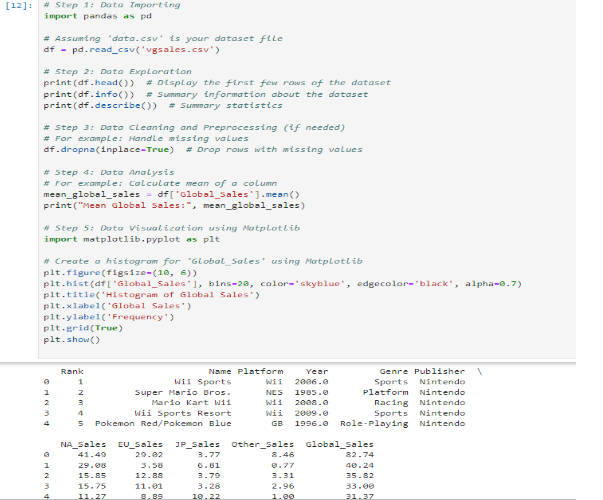
**To use CSV file**

To use a CSV file in Jupyter Notebook, you'll first need to make sure that the CSV file is uploaded or located in the same directory as your Jupyter notebook. Once you've ensured that the CSV file is accessible, you can read it into a Pandas DataFrame using the pd.read\_csv() function.



**Analysis and Visualization**

You can perform data analysis and visualization using various Python libraries such as Pandas, NumPy, Matplotlib, Seaborn, Plotly, and more.

**Importing libraries:** Python has a rich ecosystem of libraries for various tasks. In a Jupyter Notebook cell, you can use the import statement to import libraries like pandas for data analysis, numpy for numerical computing, or matplotlib for creating visualizations.

Example:

import pandas as pd

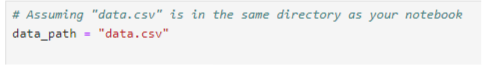


**Finding data:** Jupyter Notebook doesn't directly search for data, but you can use Python code within the notebook to specify the location of your data file (e.g., on your computer or cloud storage). For instance, you might use the os library to navigate directories or specify a URL to download data from the web.

Example:

# Assuming "data.csv" is in the same directory as your notebook

data\_path = "data.csv"



**Importing data:** Once you've identified your data source, you can use libraries like pandas to read the data. pandas offers functions like pd.read\_csv() to read data from CSV files, pd.read\_excel() for Excel files, and others depending on the data format.

data = pd.read\_csv(data\_path)



**Data attributes:** After importing the data, you can explore its attributes using the data object. You can check the number of rows and columns using data.shape, get column names using data.columns, or see a glimpse of the data using methods like data.head() (shows the first few rows). These attributes and methods help you understand the structure and content of your data.

Examples:

print(df.shape) # Output: (number of rows, number of columns)

print(df.columns) # List of column names

print(df.head()) # Show the first few rows

