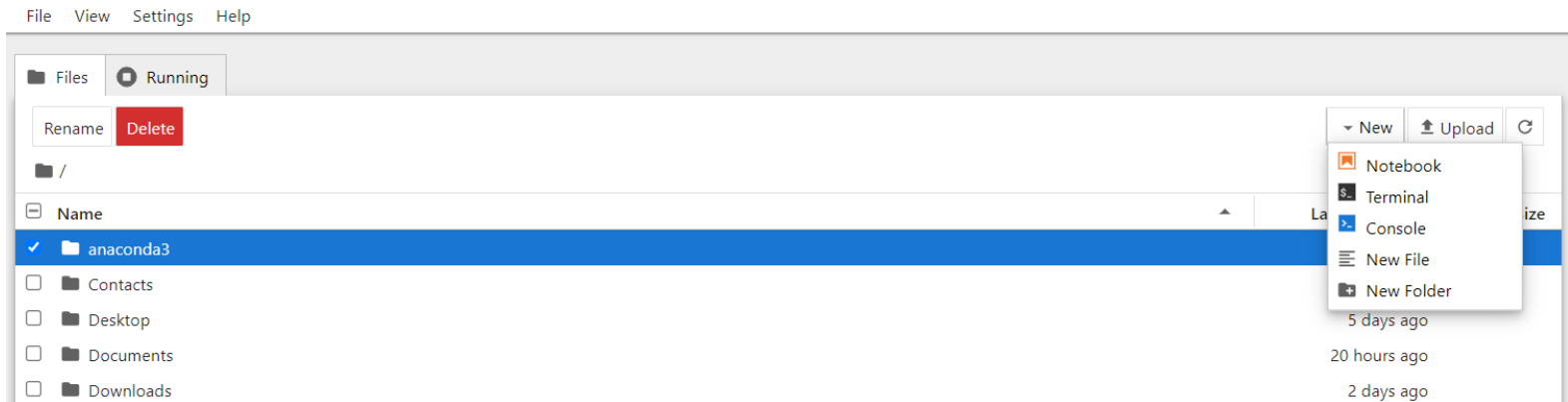


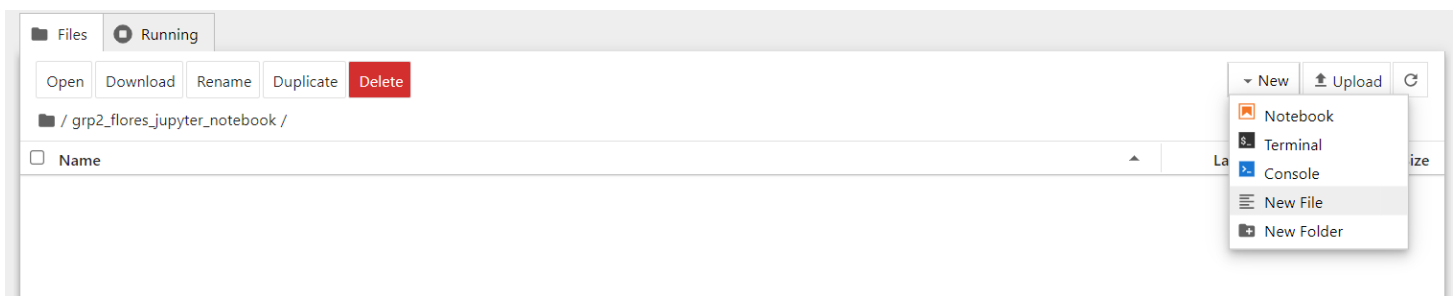
JUPYTER NOTEBOOK

ADDING FOLDERS



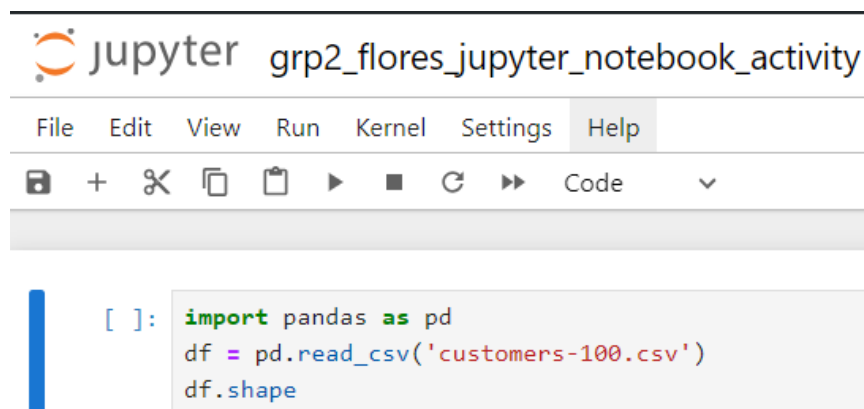
- To create or add a folder, click the “new” on the upper right corner then “New Folder” to create a folder.

ADDING TEXT FILE



- To add a new text file, click again the “new” button then click the “New File” to create a text file. The created text file was named “Untitled.txt” Make sure to rename your file.

CSV File for Data Analysis and Visualization



- Comma-Separated Values, or CSV, files are a common format for keeping tabular data organized in a that both machines and humans can read with ease. They are perfect for analyzing data and visualization in Jupyter Notebooks due to its widely compatible nature and ease of use.

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.

```
•[1]: my_dict = {'name': 'Christian', 'age': 21}
      print(my_dict.keys()) # prints the keys of the dictionary
      dict_keys(['name', 'age'])

[2]: my_dict = {'name': 'Christian', 'age': 21}
      print(my_dict.values())
      dict_values(['Christian', 21])
```

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

```
name = my_dict['name']
print(name) # Output: name
```

```
[3]: name = my_dict['name']
      print(name) #Output Christian
      Christian

[4]: age = my_dict['age']
      print(age) #Output 21
      21
```

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

```
[5]: my_dict['age'] = 24
      print(my_dict['age']) #Output 24
      24
```

LOOP THROUGH DICTIONARY VALUES

- : Use a for loop to iterate over the values in the dictionary

```
[6]: for value in my_dict.values():
      print(value)
      Christian
      24
```

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

```
[7]: if 'address' in my_dict:
      print ("country key exists")
      else:
          print ("country key does not exists")

country key does not exists
```

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

```
[8]: print (len(my_dict)) #Output 2

2
```

- Adding Items in the Dictionary: You can add new key-value pairs using the assignment operator with the key in square brackets.

```
•[9]: my_dict['address'] = 'Alabang'
      print (my_dict) # Output: {'name': 'Christian', 'age': 24, 'address': 'Alabang'}

{'name': 'Christian', 'age': 24, 'address': 'Alabang'}
```

- Removing Items in the Dictionary: Use the del keyword with the key in square brackets to remove a key-value pair.

```
[18]: del my_dict['address']
      print(my_dict) #Output {'name': 'Christian', 'age': 22,}

{'name': 'Christian', 'age': 22}
```

- Remove an Item Using del Statement: Alternatively, use the pop() method to remove a key-value pair and return the value.

```
[19]: my_dict.pop('age')
      print(my_dict) # Output: {'name': 'Christian'}

{'name': 'Christian'}
```

- The dict() Constructor: You can also create dictionaries using the dict() constructor and passing key-value pairs as arguments.

```
[20]: new_dict = dict(name='Dion', age = 18)
      print(new_dict) # Output: {'name': 'Dion', 'age': 18}

{'name': 'Dion', 'age': 18}
```

- Dictionary Methods: Dictionaries have built-in methods for various operations. For example, .get(key, default) returns the value for the key or a default value if the key doesn't exist.

```
[21]: print(my_dict.get('age')) #Output None (key not found)
      print(my_dict.get('name', 'default_name')) #Output Christian

None
Christian
```

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!

```
[ ]: #Importing the necessary Library
import os

#Specify the Directory Path
directory = 'new_directory'

#Create the Directory
os.makedirs (directory)
```

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.

```
[ ]: # Step 1: Import Library
import pandas as pd
```

TO USE THE 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. You can view the first few rows of the DataFrame using the `head()` method to ensure it's loaded correctly.

```
[ ]: import pandas as pd

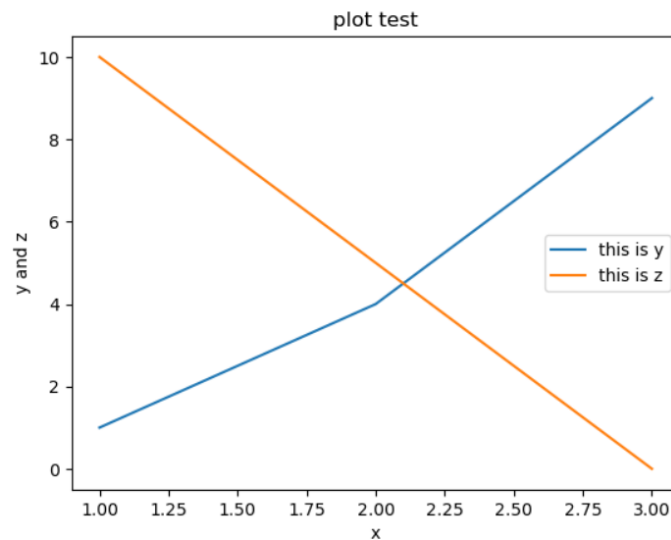
[ ]: from matplotlib import pyplot as plt

[ ]: x = [1, 2, 3]
     y = [1, 4, 9]
     z = [10, 5, 0]
     plt.plot (x, y)
     plt.plot (x, z)
     plt.title ("plot test")
     plt.xlabel ("x")
     plt.ylabel ("y and z")
     plt.legend (["this is y", "this is z"])
```

ANALYSIS AND VISUALIZATION

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

[26]: <matplotlib.legend.Legend at 0x17e1e338710>



IMPORT 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.

```
[ ]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

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.

```
[ ]: # Assuming "data.csv" is in the same directory as your notebook
data_path = "data.csv"
```

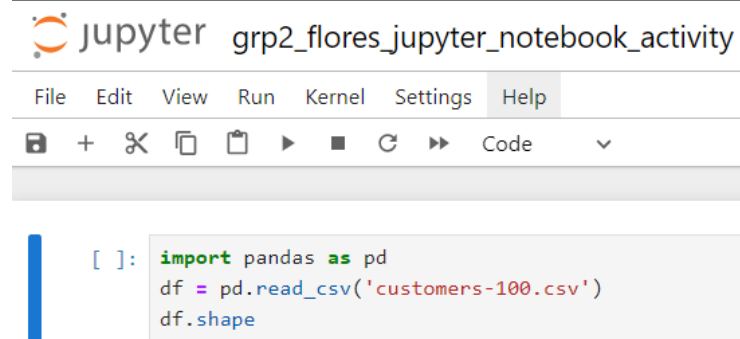
IMPORTING DATA

- 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



The screenshot shows a Jupyter Notebook window titled "jupyter grp2_flores_jupyter_notebook_activity". The interface includes a menu bar with "File", "Edit", "View", "Run", "Kernel", "Settings", and "Help". Below the menu is a toolbar with icons for saving, adding, deleting, copying, pasting, running, and other actions. The main area displays a code cell with the following Python code:

```
[ ]: import pandas as pd
      df = pd.read_csv('customers-100.csv')
      df.shape
```