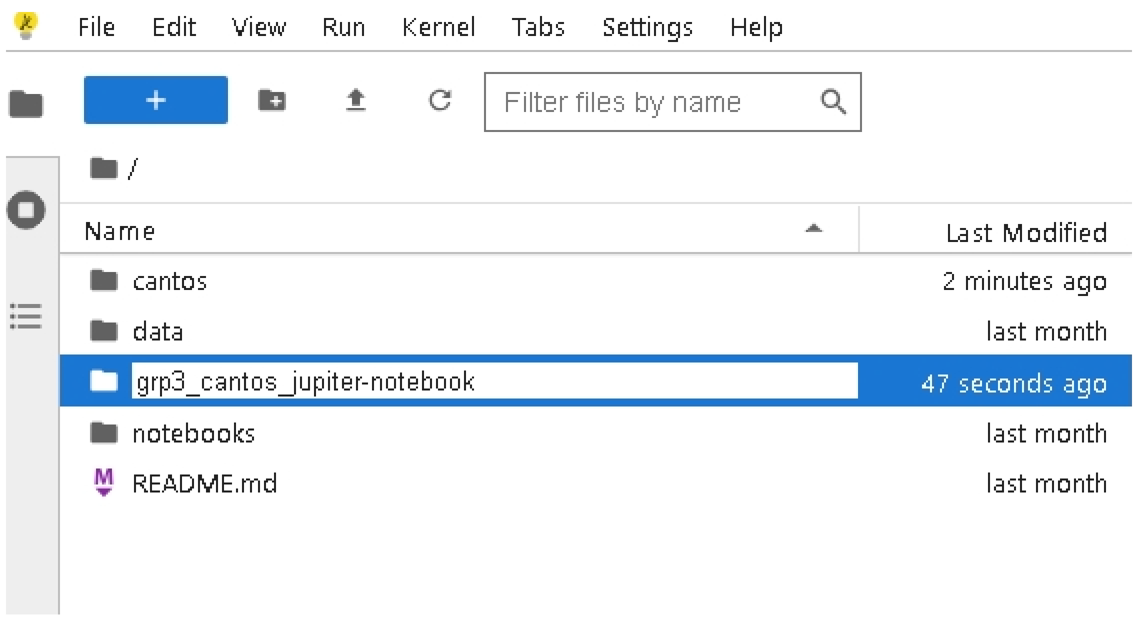
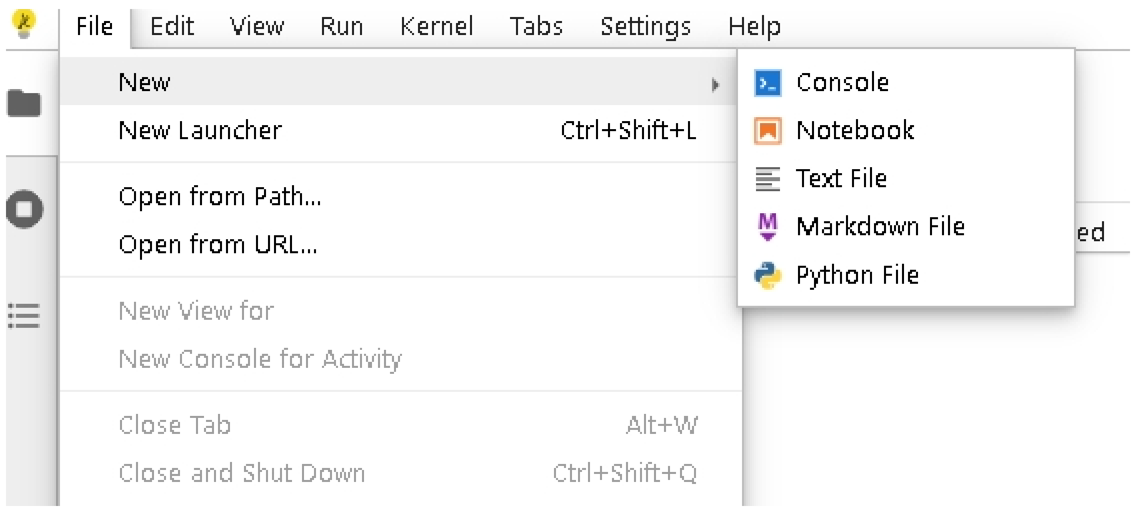
Jupyter Notebook

Adding Folders

In the upper right-hand corner of the Jupyter Notebook Lite home screen, click on the “folder” with a “plus sign” in the middle. A new folder called "Untitled Folder" will appear in the list of files on the Jupyter Notebook Lite home screen.

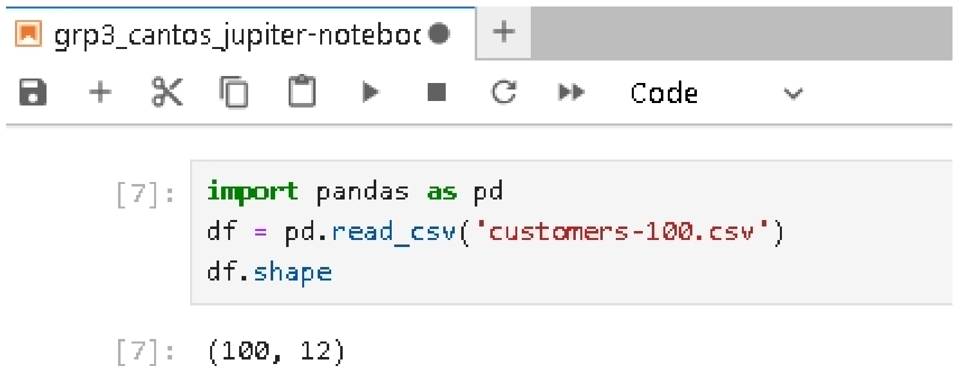


Adding Text Files: Use the “New” button in the Jupyter Notebook interface and select “Text File” to create a new text file.



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.

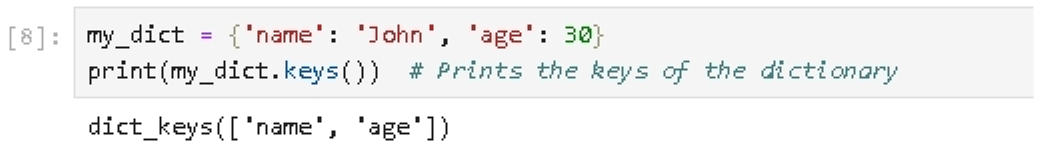


```
[7]: import pandas as pd
     df = pd.read_csv('customers-100.csv')
     df.shape

[7]: (100, 12)
```

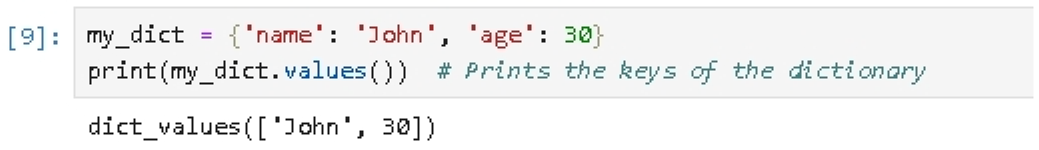
To Write and Call Dictionary Methods

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'}



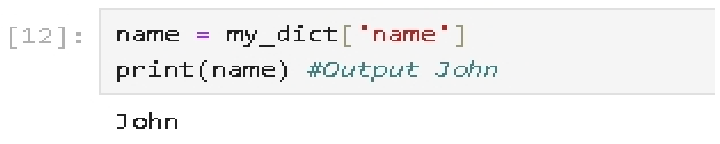
```
[8]: my_dict = {'name': 'John', 'age': 30}
     print(my_dict.keys())  # Prints the keys of the dictionary

     dict_keys(['name', 'age'])
```



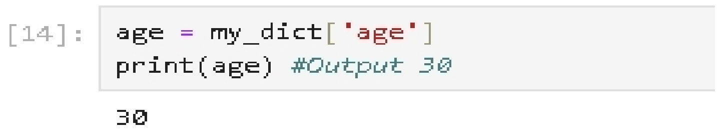
```
[9]: my_dict = {'name': 'John', 'age': 30}
     print(my_dict.values())  # Prints the keys of the dictionary

     dict_values(['John', 30])
```

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

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



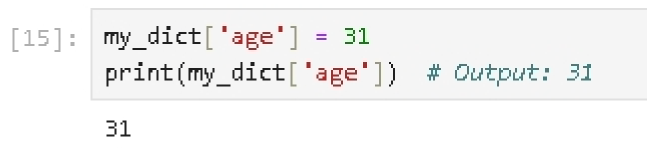
```
[12]: name = my_dict['name']
      print(name) #Output John

      John
```



```
[14]: age = my_dict['age']
      print(age) #Output 30

      30
```

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

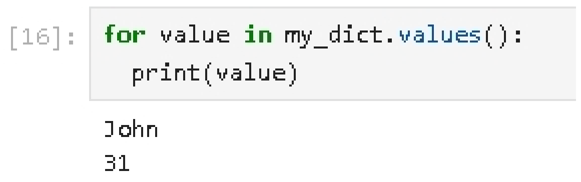
```
my_dict['age'] = 31
print(my_dict['age'])  # Output: 31
```



```
[15]: my_dict['age'] = 31
      print(my_dict['age'])  # Output: 31

      31
```

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

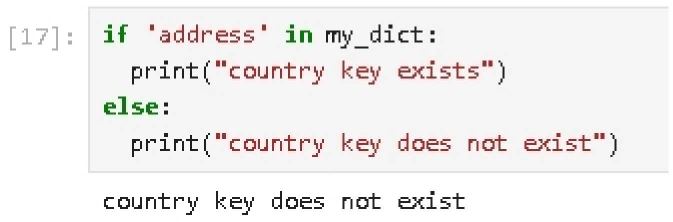
```
for value in my_dict.values():
  print(value)
```



```
[16]: for value in my_dict.values():
        print(value)

      John
      31
```

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

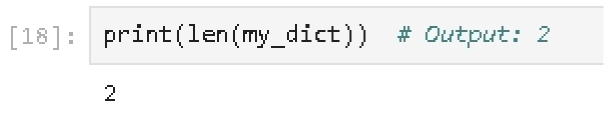
```
if 'address' in my_dict:
  print("country key exists")
else:
  print("country key does not exist")
```



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

      country key does not exist
```

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

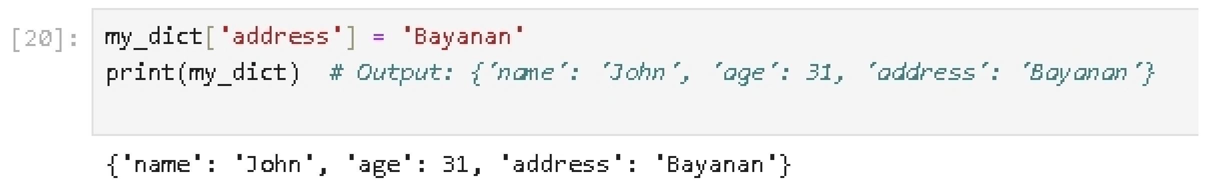
```
print(len(my_dict))  # Output: 2
```



```
[18]: 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.

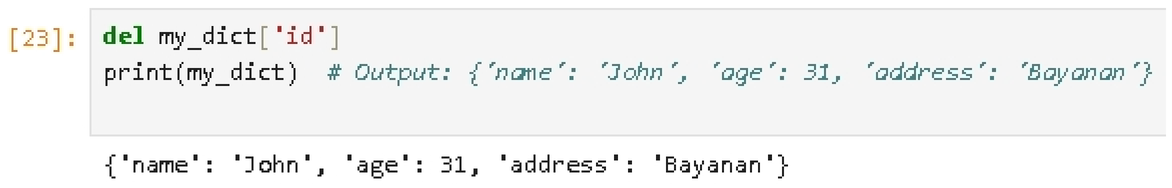
```
my_dict['address'] = 'Bayanan'
print(my_dict)  # Output: {'name': 'John', 'age': 31, 'address': 'Bayanan'}
```



```
[20]: my_dict['address'] = 'Bayanan'
      print(my_dict)  # Output: {'name': 'John', 'age': 31, 'address': 'Bayanan'}

{'name': 'John', 'age': 31, 'address': 'Bayanan'}
```

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

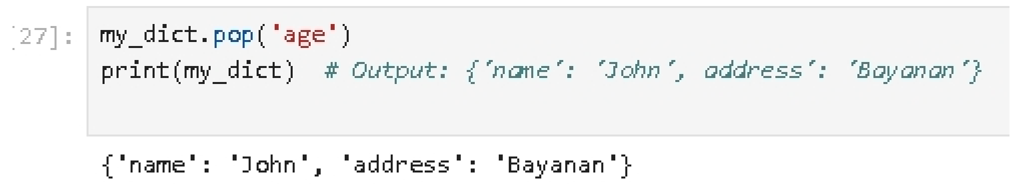
del my_dict['id']
print(my_dict) # Output: {'name': 'John', 'age': 31, 'address': 'Bayanan'}



```
[23]: del my_dict['id']
      print(my_dict)  # Output: {'name': 'John', 'age': 31, 'address': 'Bayanan'}

{'name': 'John', 'age': 31, 'address': 'Bayanan'}
```

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

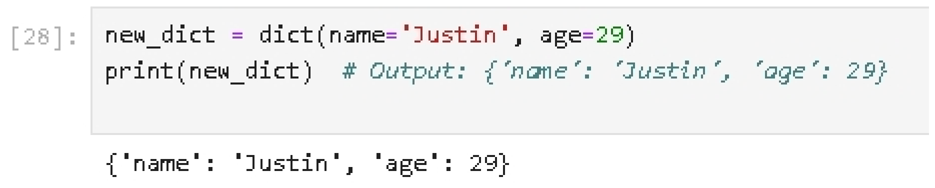
my_dict.pop('age')
print(my_dict) # Output: {'name': 'John', address': 'Bayanan'}



```
[27]: my_dict.pop('age')
      print(my_dict)  # Output: {'name': 'John', address': 'Bayanan'}

{'name': 'John', 'address': 'Bayanan'}
```

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

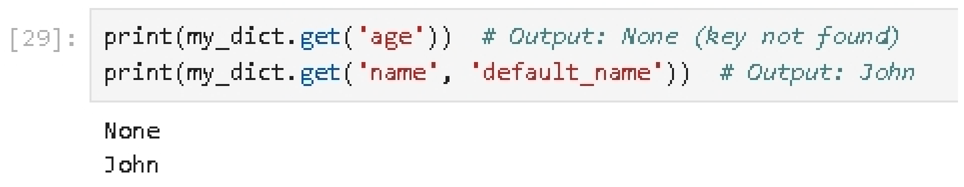
new_dict = dict(name='Justin', age=29)
print(new_dict) # Output: {'name': 'Justin', 'age': 29}



```
[28]: new_dict = dict(name='Justin', age=29)
      print(new_dict)  # Output: {'name': 'Justin', 'age': 29}

{'name': 'Justin', 'age': 29}
```

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.

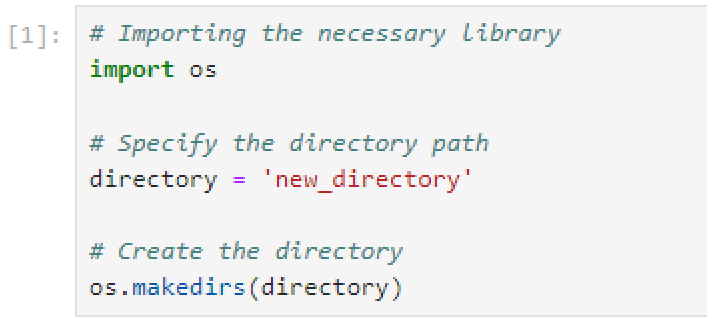
print(my_dict.get('age')) # Output: None (key not found) print(my_dict.get('name', 'default_name')) # Output: John



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

      None
      John
```

To Create a directory using Jupyter notebook

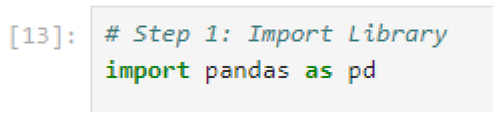
Use the built-in Python functions for file operations. You can execute shell commands directly from Jupyter Notebook cells by prefixing the command with an exclamation mark!.



```
[1]: # 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.

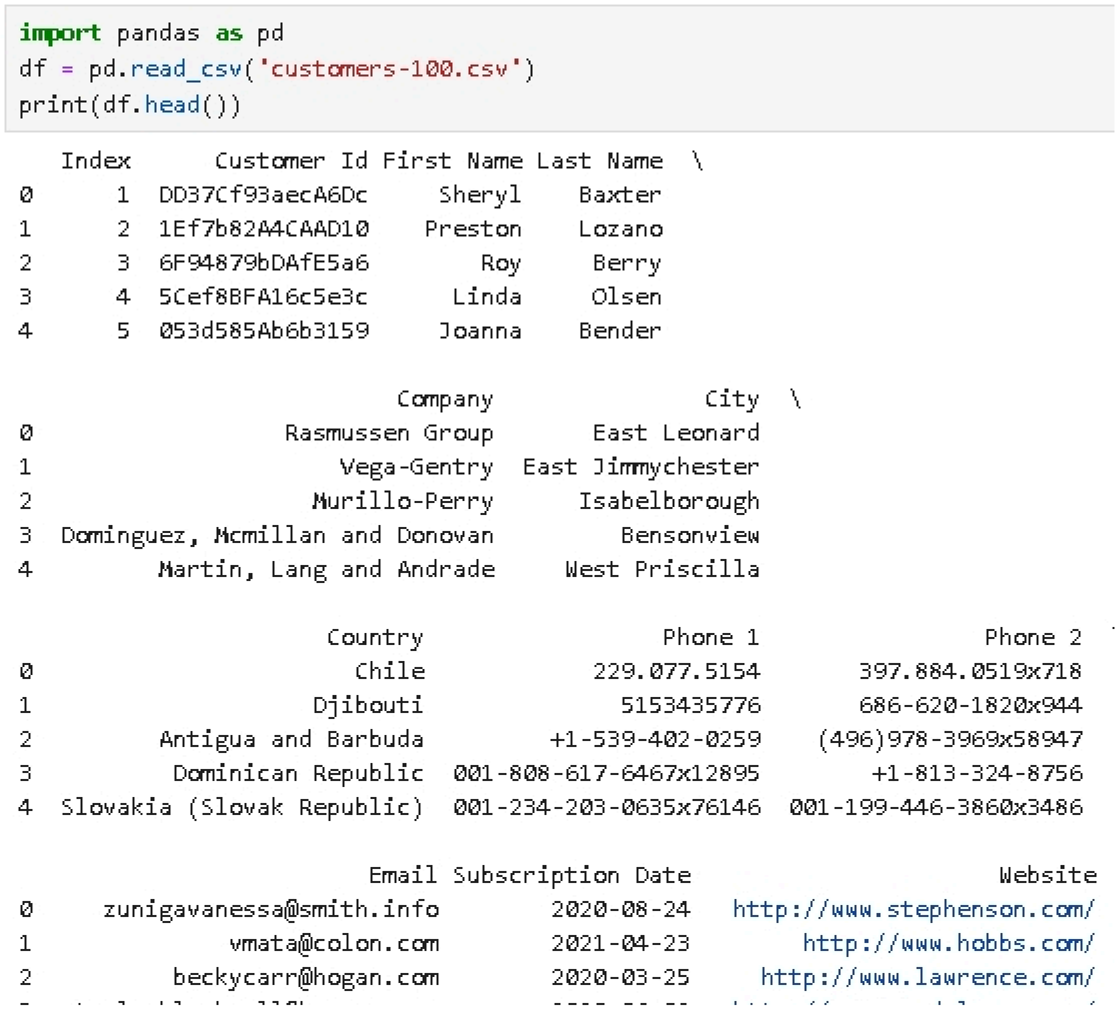


```
[13]: # Step 1: Import Library
      import pandas as pd
```

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. You can view the first few rows of the DataFrame using the head() method to ensure it’s loaded correctly.



```python
import pandas as pd
df = pd.read_csv('customers-100.csv')
print(df.head())
```

```
   Index      Customer Id First Name Last Name  \
0      1  DD37Cf93aecA6Dc     Sheryl    Baxter
1      2  1Ef7b82A4CAAD10    Preston    Lozano
2      3  6F94879bDAfE5a6        Roy     Berry
3      4  5Cef8BFA16c5e3c      Linda     Olsen
4      5  053d585Ab6b3159     Joanna    Bender

                          Company               City  \
0                 Rasmussen Group       East Leonard
1                     Vega-Gentry  East Jimmychester
2                   Murillo-Perry      Isabelborough
3  Dominguez, Mcmillan and Donovan         Bensonview
4         Martin, Lang and Andrade     West Priscilla

                     Country                Phone 1               Phone 2
0                      Chile           229.077.5154      397.884.0519x718
1                   Djibouti             5153435776      686-620-1820x944
2        Antigua and Barbuda        +1-539-402-0259   (496)978-3969x58947
3         Dominican Republic  001-808-617-6467x12895       +1-813-324-8756
4  Slovakia (Slovak Republic) 001-234-203-0635x76146  001-199-446-3860x3486

                       Email Subscription Date                     Website
0    zunigavanessa@smith.info        2020-08-24  http://www.stephenson.com/
1             vmata@colon.com        2021-04-23       http://www.hobbs.com/
2           beckycarr@hogan.com      2020-03-25    http://www.lawrence.com/
```

Analysis and Visualization

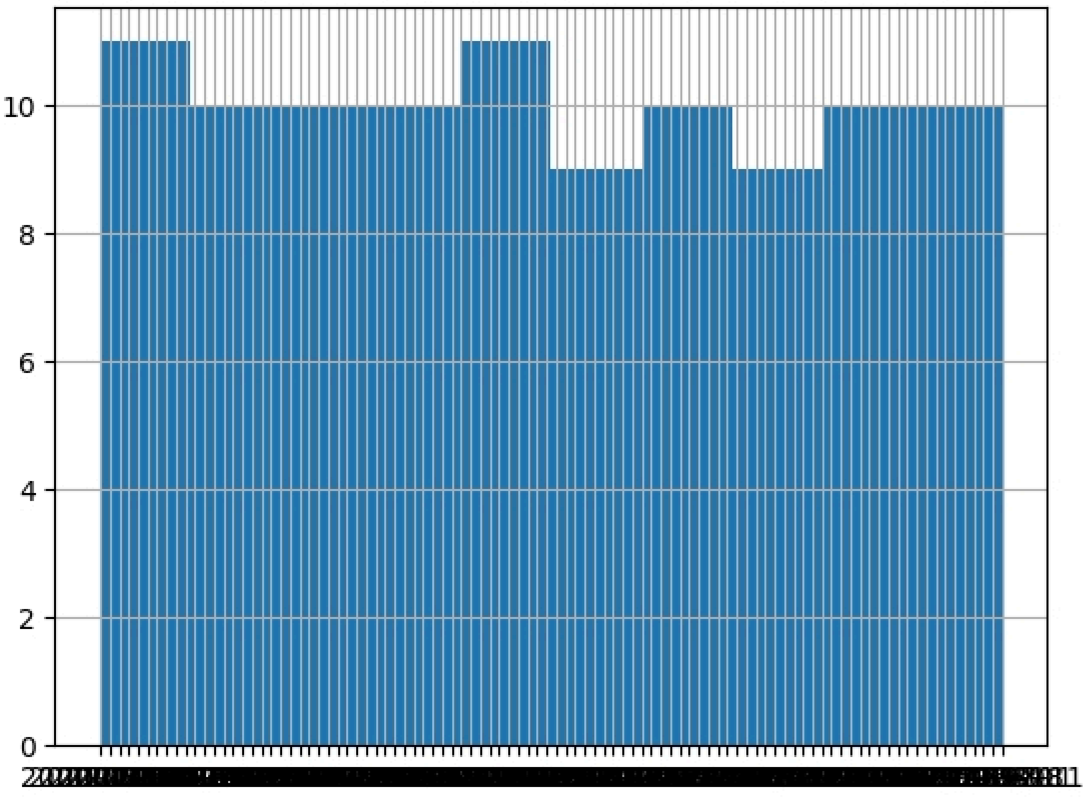
You can perform data analysis and visualization using various Python libraries such as Pandas, NumPy, Matplotlib, Seaborn, Plotly, and more.

[32]:

```python
import pandas as pd
import matplotlib.pyplot as plt

# Load the data into a DataFrame
df = pd.read_csv('customers-100.csv')

# Clean the data if necessary (e.g., handling missing values)
df = df.dropna()

# Explore the data
print(df.describe())
print(df.info())

# Visualize the data
df['Subscription Date'].hist()
plt.show()
```

Output:

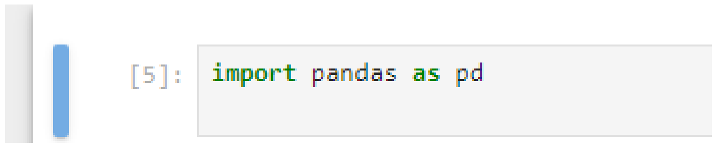
```
            Index
count  100.000000
mean    50.500000
std     29.011492
min      1.000000
25%     25.750000
50%     50.500000
75%     75.250000
max    100.000000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   Index              100 non-null    int64 
 1   Customer Id        100 non-null    object
 2   First Name         100 non-null    object
 3   Last Name          100 non-null    object
 4   Company            100 non-null    object
 5   City               100 non-null    object
 6   Country            100 non-null    object
 7   Phone 1            100 non-null    object
 8   Phone 2            100 non-null    object
 9   Email              100 non-null    object
 10  Subscription Date  100 non-null    object
 11  Website            100 non-null    object
dtypes: int64(1), object(11)
memory usage: 5.1+ KB
None
```



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

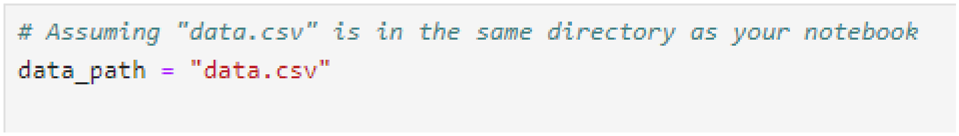


```
[5]: import pandas as pd
```

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"
```



```
# 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.

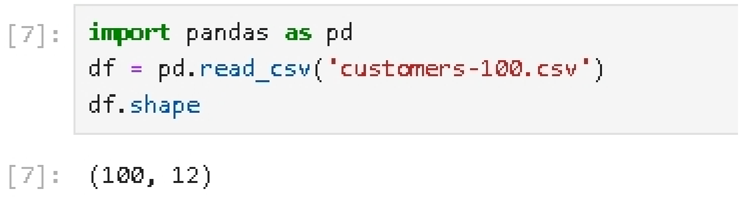
```
data = pd.read_csv(data_path)
```

```
[7]: 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
```



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

[7]: (100, 12)
```