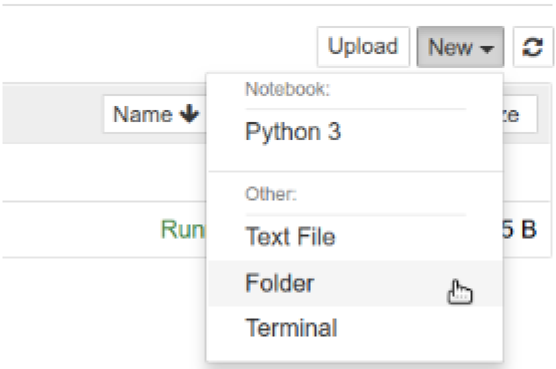
Malaluan, Nervin C. BSCS - 3C Group 3

Using Jupyter Notebook:

## Jupyter Notebook

### Adding Folders

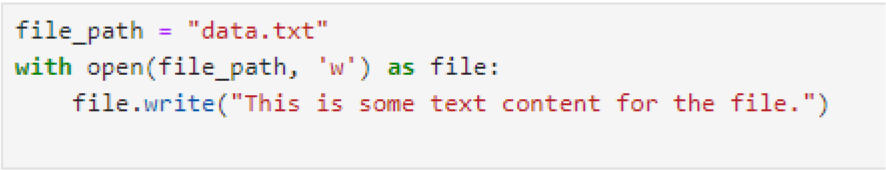
In the upper right-hand corner of the Jupyter Notebook home screen, click on the "New" drop-down button and select "Folder". A new folder called "Untitled Folder" will appear in the list of files on the Jupyter Notebook home screen.



**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:

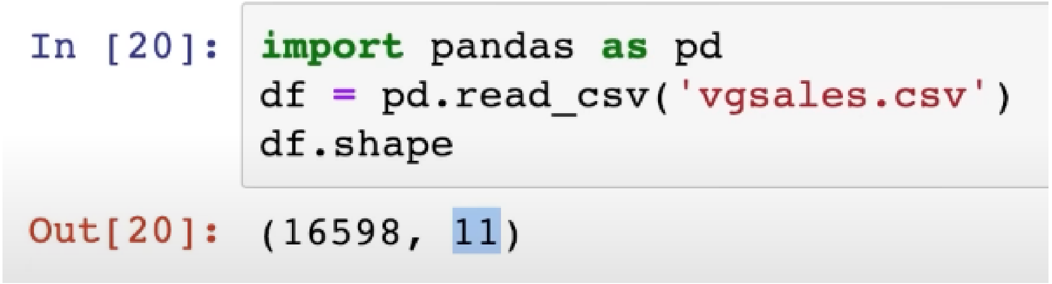
# Create a new text file named "data.txt"
file_path = "data.txt"
with open(file_path, 'w') as file:
  file.write("This is some text content for the file.")



```
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.

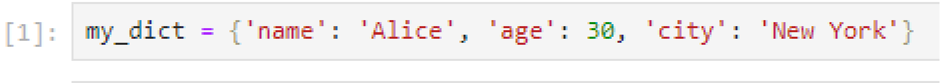


```
In [20]: import pandas as pd
         df = pd.read_csv('vgsales.csv')
         df.shape

Out[20]: (16598, 11)
```

### *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:

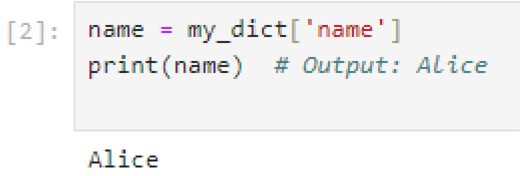
my_dict = {'name': 'Alice', 'age': 30, 'city': 'New York'}



```
[1]: 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.

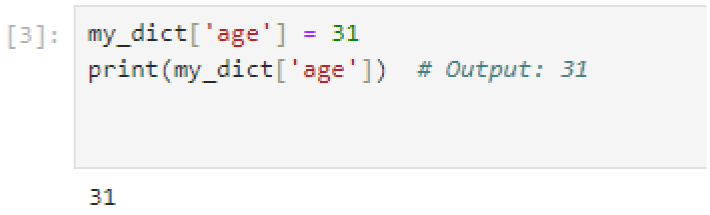
name = my_dict['name']
print(name)  # Output: Alice



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

     Alice
```

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

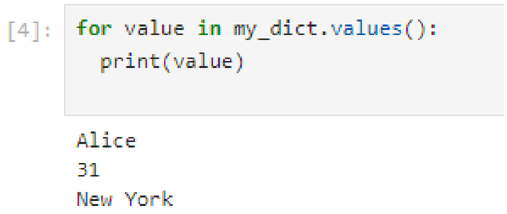
my_dict['age'] = 31
print(my_dict['age'])  # Output: 31



```
[3]: 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.

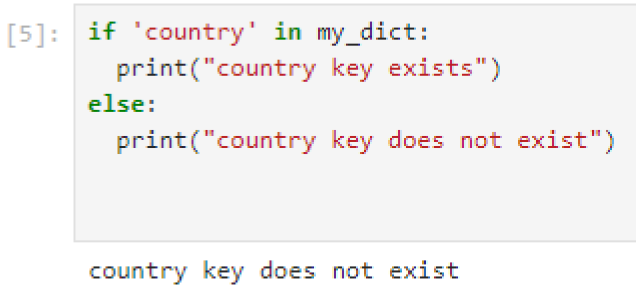
```
for value in my_dict.values():
  print(value)
```



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

Alice
31
New York
```

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

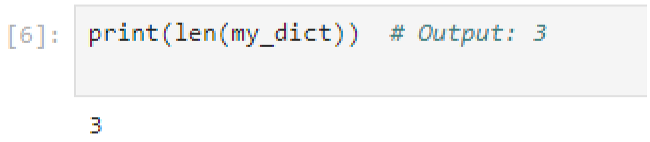
```
if 'country' in my_dict:
  print("country key exists")
else:
  print("country key does not exist")
```



```
[5]: if 'country' 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.

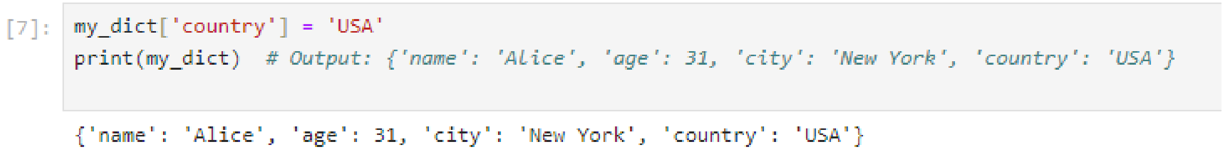
```
print(len(my_dict))  # Output: 3
```



```
[6]: print(len(my_dict))  # Output: 3

3
```

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

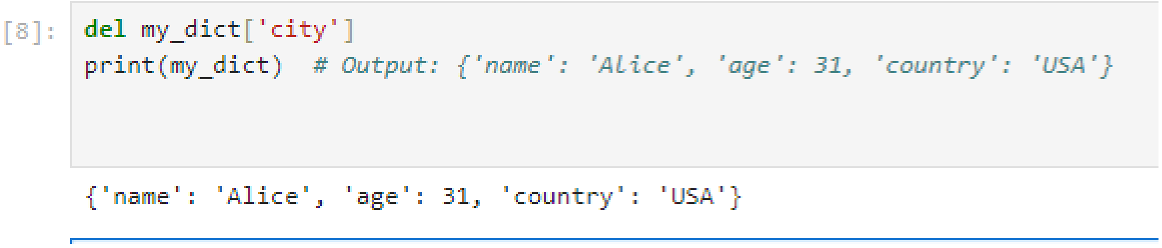
```
my_dict['country'] = 'USA'
print(my_dict)  # Output: {'name': 'Alice', 'age': 31, 'city': 'New York', 'country': 'USA'}
```



```
[7]: my_dict['country'] = 'USA'
     print(my_dict)  # Output: {'name': 'Alice', 'age': 31, 'city': 'New York', 'country': 'USA'}

     {'name': 'Alice', 'age': 31, 'city': 'New York', 'country': 'USA'}
```

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

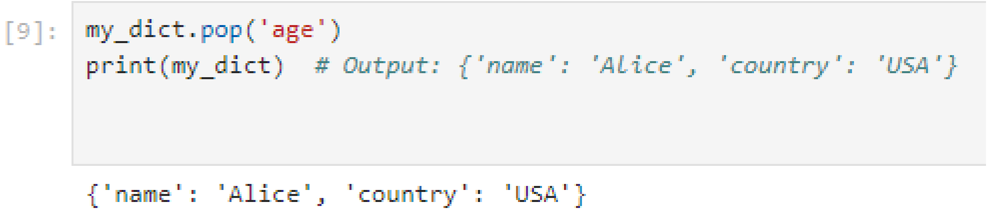
del my_dict['city']
print(my_dict)  # Output: {'name': 'Alice', 'age': 31, 'country': 'USA'}



```
[8]: del my_dict['city']
     print(my_dict)  # Output: {'name': 'Alice', 'age': 31, 'country': 'USA'}

     {'name': 'Alice', 'age': 31, 'country': 'USA'}
```

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

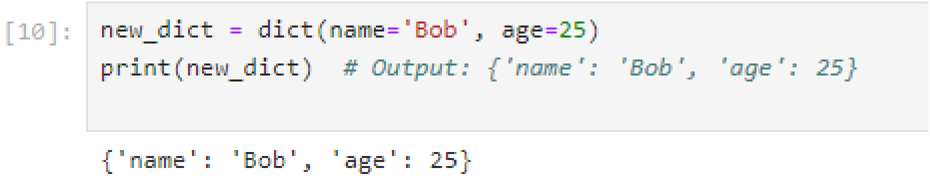
my_dict.pop('age')
print(my_dict)  # Output: {'name': 'Alice', 'country': 'USA'}



```
[9]: my_dict.pop('age')
     print(my_dict)  # Output: {'name': 'Alice', 'country': 'USA'}

     {'name': 'Alice', 'country': 'USA'}
```

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

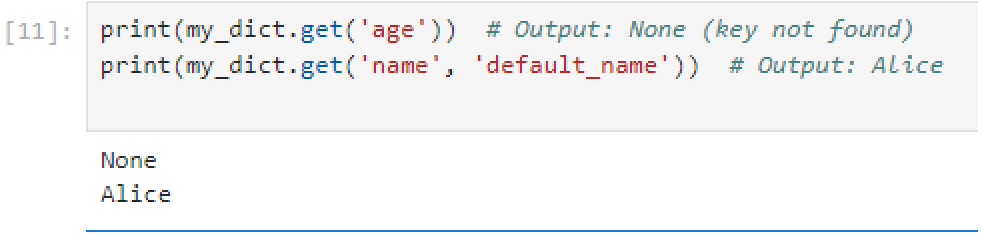
new_dict = dict(name='Bob', age=25)
print(new_dict)  # Output: {'name': 'Bob', 'age': 25}



```
[10]: new_dict = dict(name='Bob', age=25)
      print(new_dict)  # Output: {'name': 'Bob', 'age': 25}

      {'name': 'Bob', 'age': 25}
```

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.

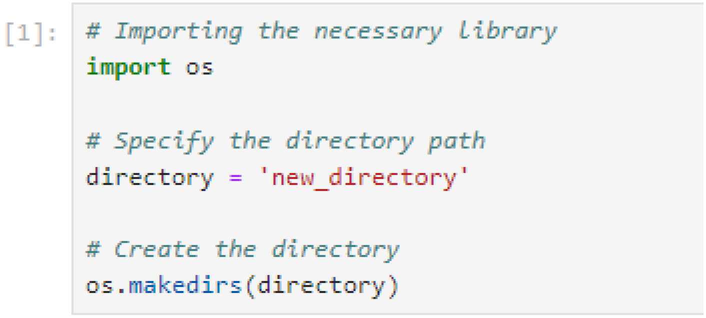
print(my_dict.get('age'))  # Output: None (key not found)
print(my_dict.get('name', 'default_name'))  # Output: Alice



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

      None
      Alice
```

### To Create a directory using Jupyter notebook

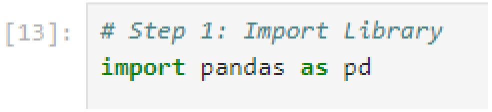
Use the built-in Python functions for file operations. You can execute shell commands directly from Jupyter Notebook cells by prefixing the command with an exclamation mark!.



```
[1]: # Importing the necessary library
     import os

     # Specify the directory path
     directory = 'new_directory'

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

### To Import Libraries

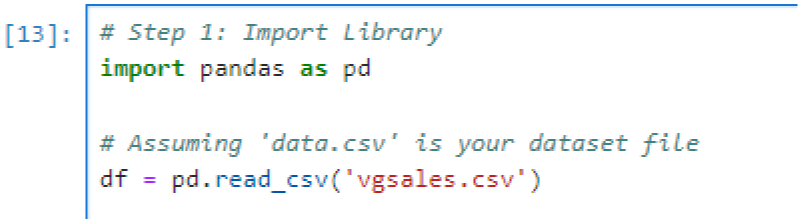
import pandas as pd: This line imports the Pandas library and gives it the alias pd, which is a common convention. This alias makes it easier to refer to Pandas functions and objects in your code by using pd as a prefix.



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

## To use CSV file

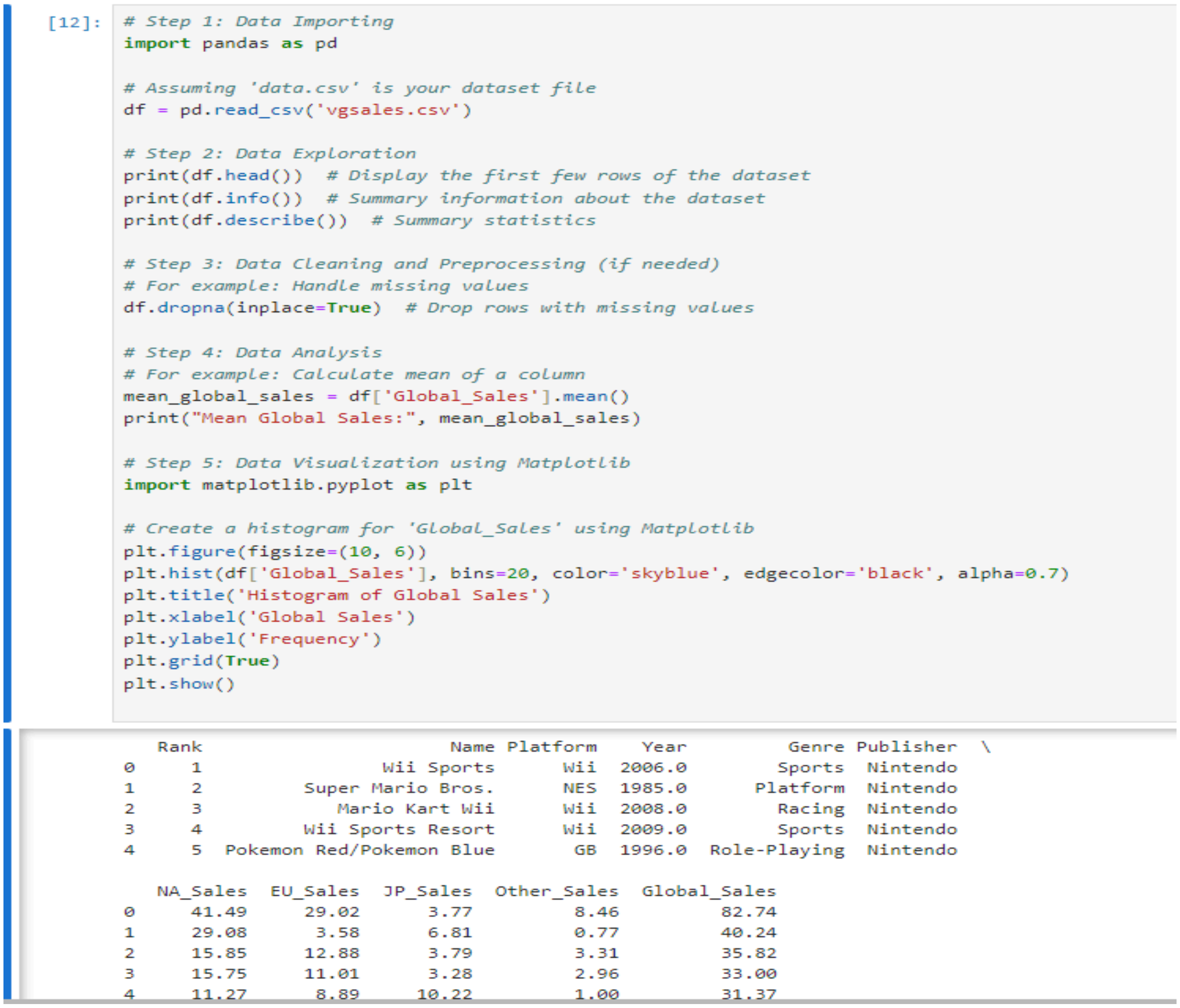
To use a CSV file in Jupyter Notebook, you'll first need to make sure that the CSV file is uploaded or located in the same directory as your Jupyter notebook. Once you've ensured that the CSV file is accessible, you can read it into a Pandas DataFrame using the pd.read_csv() function.



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

      # Assuming 'data.csv' is your dataset file
      df = pd.read_csv('vgsales.csv')
```

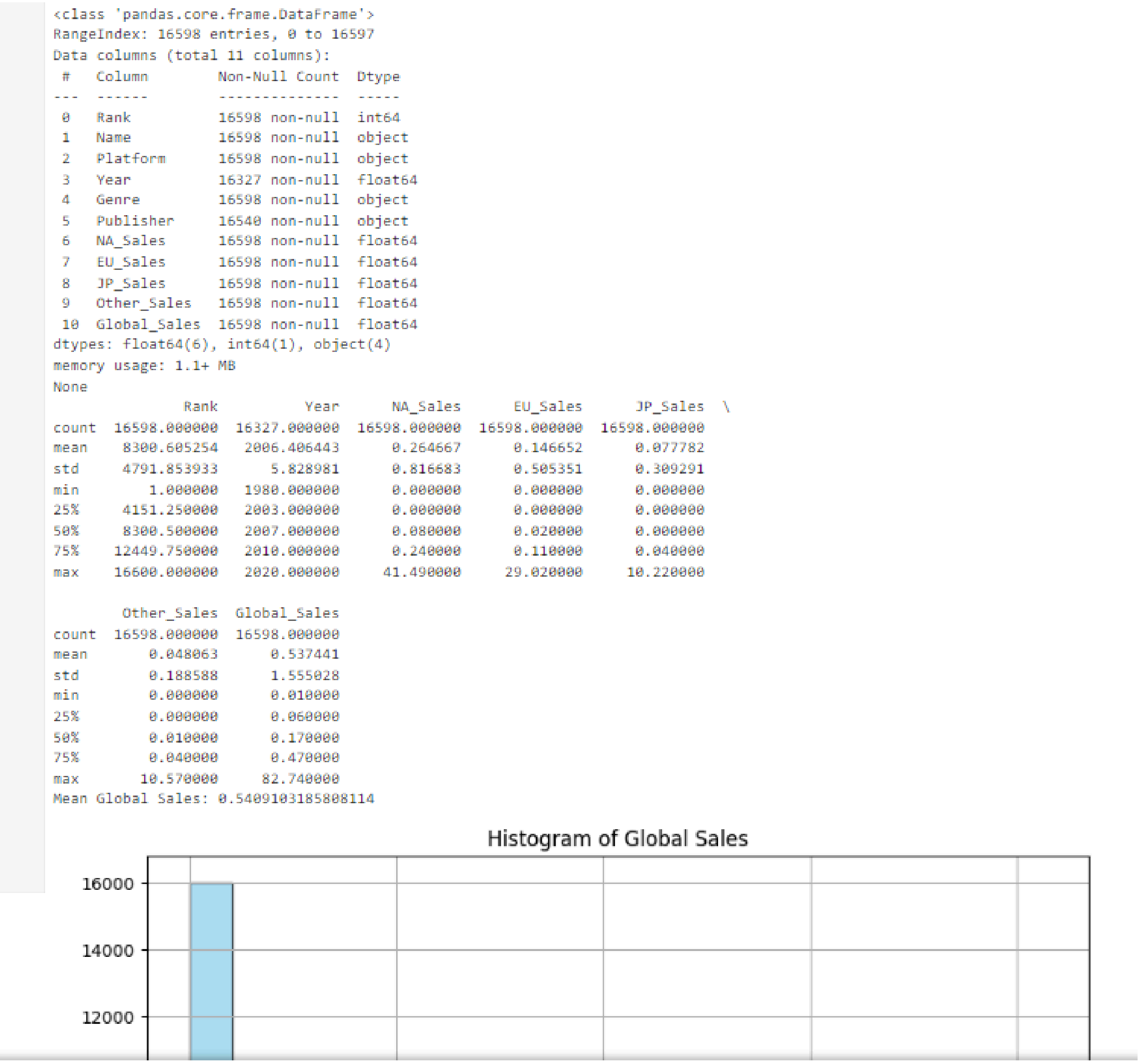
## Analysis and Visualization

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

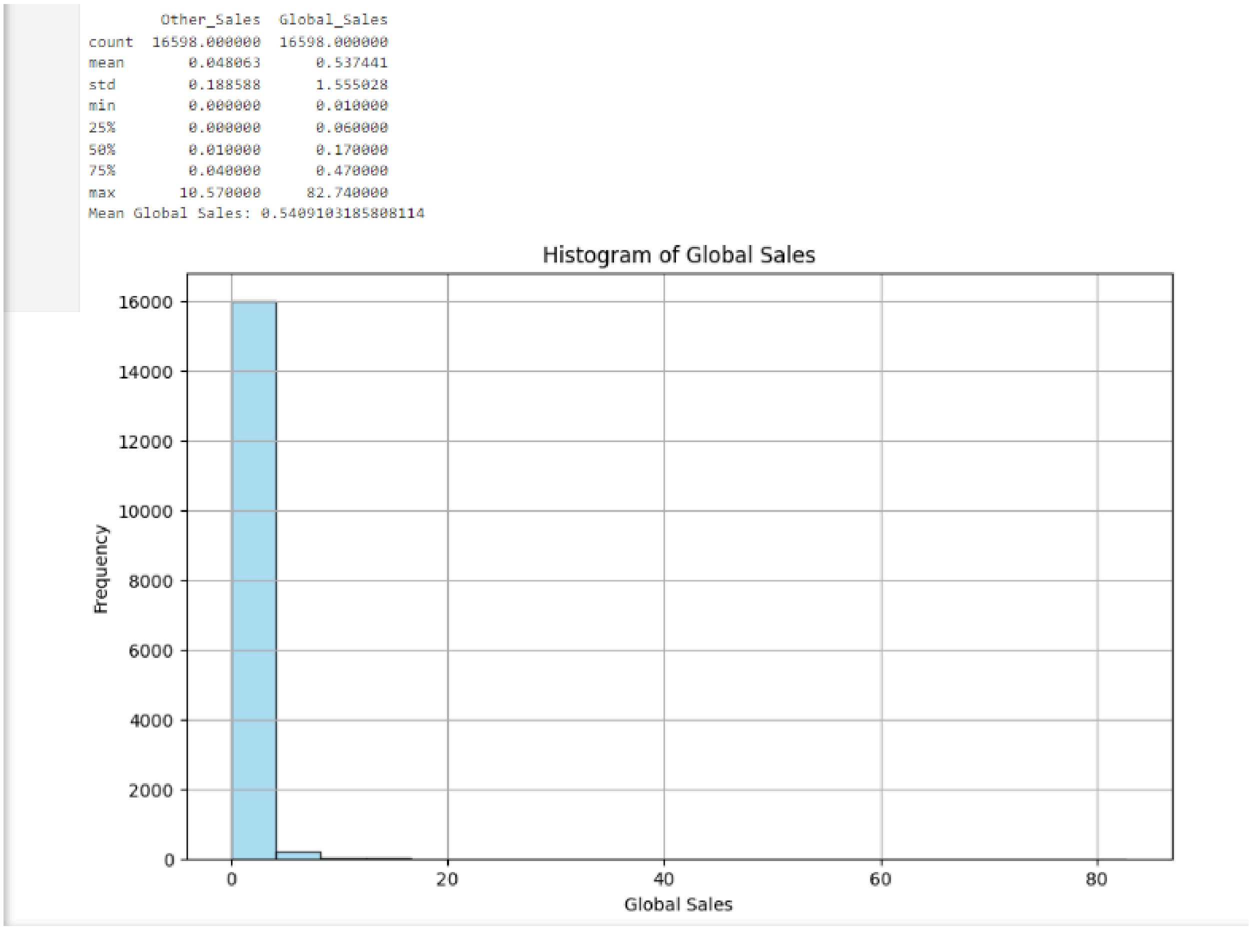


```
[12]: # Step 1: Data Importing
      import pandas as pd

      # Assuming 'data.csv' is your dataset file
      df = pd.read_csv('vgsales.csv')

      # Step 2: Data Exploration
      print(df.head())  # Display the first few rows of the dataset
      print(df.info())  # Summary information about the dataset
      print(df.describe())  # Summary statistics

      # Step 3: Data Cleaning and Preprocessing (if needed)
      # For example: Handle missing values
      df.dropna(inplace=True)  # Drop rows with missing values

      # Step 4: Data Analysis
      # For example: Calculate mean of a column
      mean_global_sales = df['Global_Sales'].mean()
      print("Mean Global Sales:", mean_global_sales)

      # Step 5: Data Visualization using Matplotlib
      import matplotlib.pyplot as plt

      # Create a histogram for 'Global_Sales' using Matplotlib
      plt.figure(figsize=(10, 6))
      plt.hist(df['Global_Sales'], bins=20, color='skyblue', edgecolor='black', alpha=0.7)
      plt.title('Histogram of Global Sales')
      plt.xlabel('Global Sales')
      plt.ylabel('Frequency')
      plt.grid(True)
      plt.show()
```

```
   Rank                      Name Platform    Year         Genre Publisher  \
0     1                Wii Sports      Wii  2006.0        Sports  Nintendo
1     2         Super Mario Bros.      NES  1985.0      Platform  Nintendo
2     3            Mario Kart Wii      Wii  2008.0        Racing  Nintendo
3     4         Wii Sports Resort      Wii  2009.0        Sports  Nintendo
4     5  Pokemon Red/Pokemon Blue       GB  1996.0  Role-Playing  Nintendo

   NA_Sales  EU_Sales  JP_Sales  Other_Sales  Global_Sales
0     41.49     29.02      3.77         8.46         82.74
1     29.08      3.58      6.81         0.77         40.24
2     15.85     12.88      3.79         3.31         35.82
3     15.75     11.01      3.28         2.96         33.00
4     11.27      8.89     10.22         1.00         31.37
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16598 entries, 0 to 16597
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Rank          16598 non-null  int64  
 1   Name          16598 non-null  object 
 2   Platform      16598 non-null  object 
 3   Year          16327 non-null  float64
 4   Genre         16598 non-null  object 
 5   Publisher     16540 non-null  object 
 6   NA_Sales      16598 non-null  float64
 7   EU_Sales      16598 non-null  float64
 8   JP_Sales      16598 non-null  float64
 9   Other_Sales   16598 non-null  float64
 10  Global_Sales  16598 non-null  float64
dtypes: float64(6), int64(1), object(4)
memory usage: 1.1+ MB
None
               Rank          Year      NA_Sales      EU_Sales      JP_Sales  \
count  16598.000000  16327.000000  16598.000000  16598.000000  16598.000000   
mean    8300.605254   2006.406443      0.264667      0.146652      0.077782   
std     4791.853933      5.828981      0.816683      0.505351      0.309291   
min        1.000000   1980.000000      0.000000      0.000000      0.000000   
25%     4151.250000   2003.000000      0.000000      0.000000      0.000000   
50%     8300.500000   2007.000000      0.080000      0.020000      0.000000   
75%    12449.750000   2010.000000      0.240000      0.110000      0.040000   
max    16600.000000   2020.000000     41.490000     29.020000     10.220000   

        Other_Sales  Global_Sales  
count  16598.000000  16598.000000  
mean       0.048063      0.537441  
std        0.188588      1.555028  
min        0.000000      0.010000  
25%        0.000000      0.060000  
50%        0.010000      0.170000  
75%        0.040000      0.470000  
max       10.570000     82.740000  
Mean Global Sales: 0.5409103185808114
```

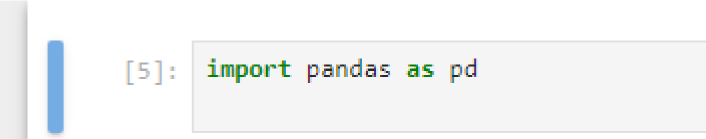


```
        Other_Sales  Global_Sales
count  16598.000000  16598.000000
mean       0.048063      0.537441
std        0.188588      1.555028
min        0.000000      0.010000
25%        0.000000      0.060000
50%        0.010000      0.170000
75%        0.040000      0.470000
max       10.570000     82.740000
Mean Global Sales: 0.5409103185808114
```

**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

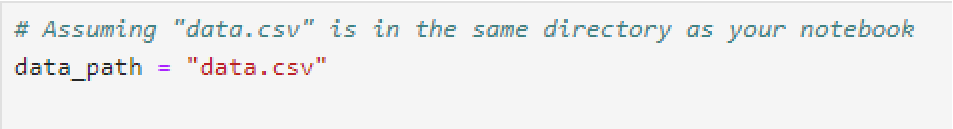


```
[5]: import pandas as pd
```

**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:

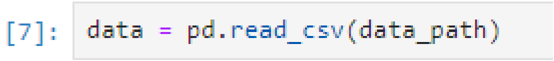
# Assuming "data.csv" is in the same directory as your notebook
data_path = "data.csv"



```
# 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)

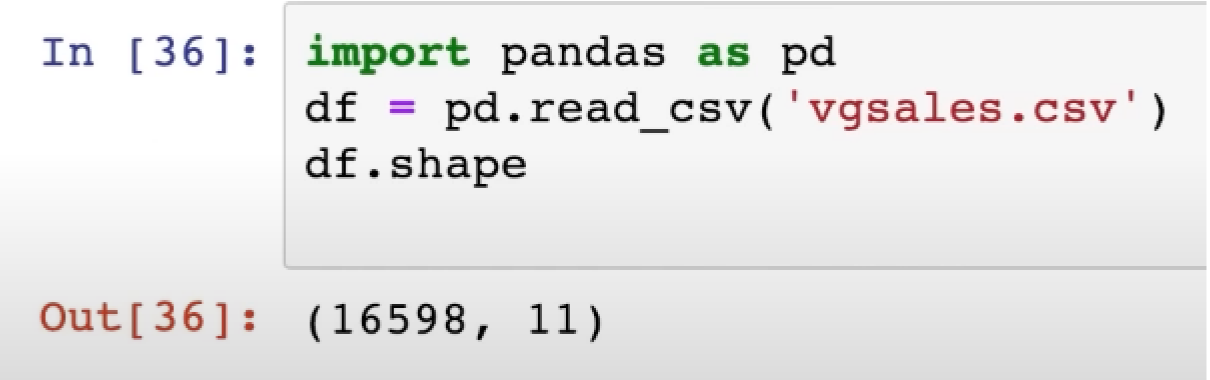


```
[7]:  data = pd.read_csv(data_path)
```

**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



```
In [36]:  import pandas as pd
          df = pd.read_csv('vgsales.csv')
          df.shape

Out[36]:  (16598, 11)
```