Lab Record 1: Introduction to Jupyter Notebook

Task 1: Setting Up Jupyter Notebook

1.1. Launch Jupyter Notebook on your computer

1. **Install Anaconda**: The easiest way to install Jupyter Notebook is by downloading the Anaconda distribution, which includes Jupyter as well as many other data science tools. You can download Anaconda from the official Anaconda website.
2. **Open Anaconda Navigator**: After installation, open Anaconda Navigator from your computer.
3. **Launch Jupyter Notebook**: In Anaconda Navigator, you will see a list of applications. Find Jupyter Notebook and click "Launch." This will open Jupyter Notebook in your default web browser.
4. **Use Anaconda prompt**: Type 'jupyter notebook' in anaconda terminal after activation of required environment and press enter.

1.2. Create a new notebook

1. **New Notebook**: In the Jupyter Notebook interface, click the "New" button on the right side of the screen.
2. **Select Python 3**: From the dropdown menu, select "Python 3." This will create a new notebook.

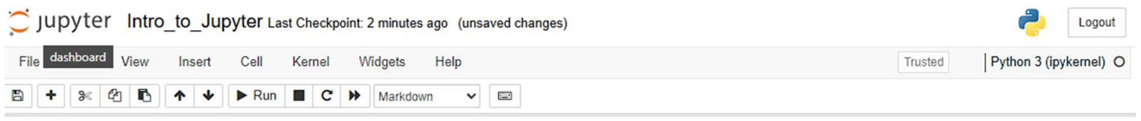
1.3. Familiarize yourself with the interface

- **Cells**: Jupyter notebooks are made up of cells. There are two main types of cells: Code cells (for writing and executing code) and Markdown cells (for writing text).
- **Toolbar**: The toolbar provides buttons for common actions like saving the notebook, adding new cells, cutting, copying, pasting cells, and running the current cell.
- **Menu options**: The menu bar includes options for file operations, editing cells, viewing and running code, inserting new cells, and managing the notebook's kernel.

1.4. Rename your notebook

1. **Rename**: Click on the notebook title (usually "Untitled") at the top of the screen.
2. **Enter New Name**: A dialog will appear. Enter "Intro_to_Jupyter.ipynb" as the new name and click "Rename."

Output: Task 1



Task 2: Understanding and Using Cells

2.1. Understand the two main types of cells: Code and Markdown

- **Code Cells**: Used for writing and executing code.
- **Markdown Cells**: Used for writing text formatted with Markdown (headings, lists, links, etc.).

2.2. Create a new Markdown cell

1. **New Markdown Cell**: Click the "Insert" menu and select "Insert Cell Below." Change the cell type to Markdown by selecting "Markdown" from the dropdown in the toolbar.
2. **Write a brief introduction**: Write the following text in the new Markdown cell:

```
## Introduction to Jupyter Notebooks

Jupyter Notebooks are a powerful tool for data analysis, visualization, and
interactive computing. They allow you to combine code, text, and multimedia in a
single document.
```

2.3. Create a new Code cell and write a simple Python expression

1. **New Code Cell**: Insert a new cell below the Markdown cell and ensure the cell type is set to "Code."
2. **Python Expression**: Write the following code:

```
2 + 2
```

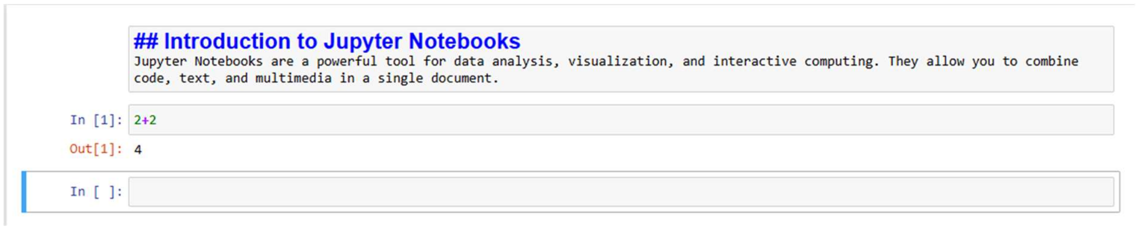
2.4. Execute the Code cell

1. **Run Cell**: Press Shift + Enter to execute the code. You will see the output (4) directly below the cell.

2.5. Experiment with adding, deleting, and moving cells

- **Add Cells**: Use the "Insert" menu or the toolbar buttons to add new cells.
- **Delete Cells**: Select a cell and press the "scissors" icon in the toolbar to cut (delete) it.
- **Move Cells**: Use the up and down arrow buttons in the toolbar to move selected cells.

Output: Task 2



```
## Introduction to Jupyter Notebooks
Jupyter Notebooks are a powerful tool for data analysis, visualization, and interactive computing. They allow you to combine
code, text, and multimedia in a single document.

In [1]: 2+2
Out[1]: 4

In [ ]:
```

Task 3: Markdown Syntax for Text Formatting

3.1. Learn basic Markdown syntax

- **Headings**: Use # for headings. More # symbols indicate smaller headings.
- **Bold**: Use **bold** or __bold__.
- **Italic**: Use *italic* or _italic_.
- **Lists**: Use - or * for unordered lists, and numbers for ordered lists.

3.2. Practice creating Markdown cells with formatted text

1. **New Markdown Cell**: Insert a new Markdown cell and practice the following:

```
# Heading 1

## Heading 2

### Heading 3

This is **bold** text and this is *italic* text.

- Item 1
- Item 2
- Item 3

1. First item
2. Second item
3. Third item
```

3.3. Insert images and hyperlinks

1. **Image**: Insert an image using the following syntax:

```
![Alt text](image_url)
```

2. **Hyperlink**: Insert a hyperlink using:

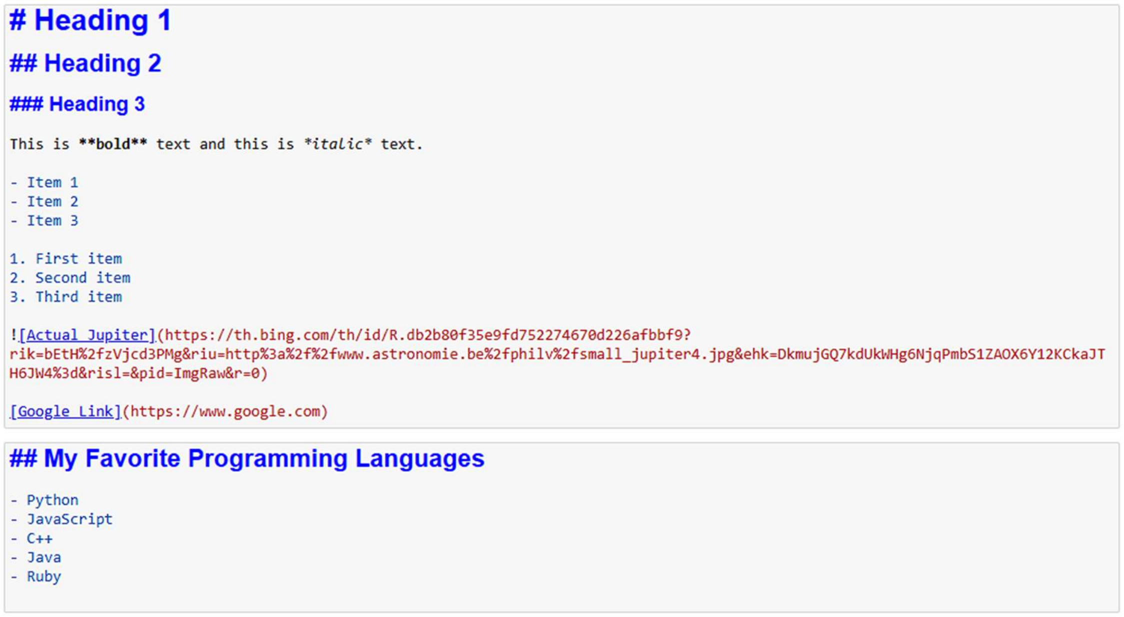
```
[Link text](URL)
```

3.4. Create a Markdown cell with a list of your favourite programming languages

1. **New Markdown Cell**: Insert a new Markdown cell and write:

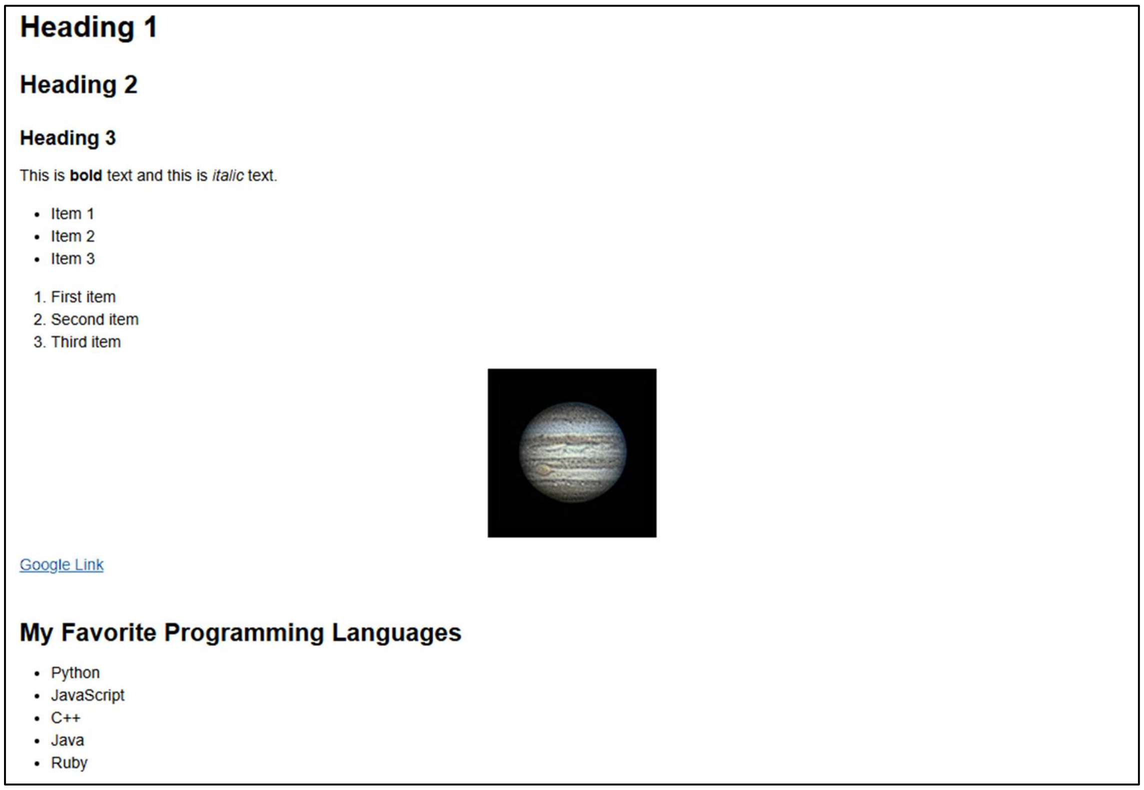
```
## My Favorite Programming Languages

- Python
- JavaScript
- C++
- Java
- Ruby
```

Output: Task 3



```
# Heading 1
## Heading 2
### Heading 3

This is **bold** text and this is *italic* text.

- Item 1
- Item 2
- Item 3

1. First item
2. Second item
3. Third item

![Actual Jupiter](https://th.bing.com/th/id/R.db2b80f35e9fd752274670d226afbbf9?
rik=bEtH%2fzVjcd3PMg&riu=http%3a%2f%2fwww.astronomie.be%2fphilv%2fsmall_jupiter4.jpg&ehk=DkmujGQ7kdUkWHg6NjqPmbS1ZAOX6Y12KCkaJT
H6JW4%3d&risl=&pid=ImgRaw&r=0)

[Google Link](https://www.google.com)
```

```
## My Favorite Programming Languages

- Python
- JavaScript
- C++
- Java
- Ruby
```



Heading 1

Heading 2

Heading 3

This is **bold** text and this is *italic* text.

- Item 1
- Item 2
- Item 3

1. First item
2. Second item
3. Third item

Google Link

My Favorite Programming Languages

- Python
- JavaScript
- C++
- Java
- Ruby

Task 4: Writing and Testing Python Code

4.1. Write a Python function that calculates the factorial of a given number

1. **New Code Cell**: Insert a new code cell and write:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

4.2. Test your function with different input values

1. **Test Function**: In the same or a new code cell, test the function:

```
print(factorial(5))  # Output: 120
print(factorial(7))  # Output: 5040
```

4.3. Import a Python library (e.g., NumPy)

1. **Import Library**: Insert a new code cell and write:

```
import numpy as np

data = [1, 2, 3, 4, 5]
mean_value = np.mean(data)
print(f"Mean: {mean_value}")
```

4.4. Visualize data using Matplotlib or Seaborn

1. **Install and Import Library**: Insert a new code cell and write:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Sample data
data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Line chart
plt.plot(data)
plt.title('Line Chart')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()

# Histogram
sns.histplot(data, bins=5)
plt.title('Histogram')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()
```

Output: Task 4

```
In [3]: def factorial(n):
            if n == 0:
                return 1
            else:
                return n * factorial(n-1)
```

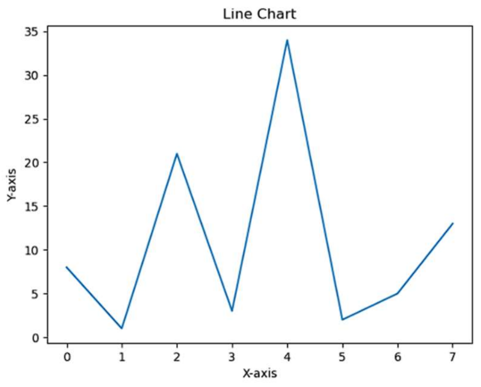
```
In [4]: print(factorial(5))  # Output: 120
        print(factorial(7))  # Output: 5040
```

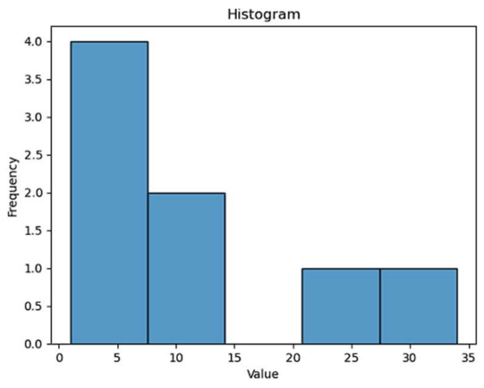
```
120
5040
```

```
In [5]: import numpy as np

        data = [1, 2, 3, 4, 5]
        mean_value = np.mean(data)
        print(f"Mean: {mean_value}")
```

```
Mean: 3.0
```

```
In [7]: import matplotlib.pyplot as plt
        import seaborn as sns

        # Sample data
        data = [8, 1, 21, 3, 34, 2, 5, 13]

        # Line chart
        plt.plot(data)
        plt.title('Line Chart')
        plt.xlabel('X-axis')
        plt.ylabel('Y-axis')
        plt.show()

        # Histogram
        sns.histplot(data, bins=5)
        plt.title('Histogram')
        plt.xlabel('Value')
        plt.ylabel('Frequency')
        plt.show()
```





Task 5: Understanding and Managing Kernels

5.1. Understand what a kernel is

- A kernel is the computational engine that executes the code in your Jupyter Notebook. Each notebook is associated with a kernel.

5.2. Switch kernels

1. **Switch Kernel**: Go to Kernel > Change kernel and select a different kernel if available (e.g., Python 2, Julia).

5.3. Run a simple command in a different kernel

1. **New Code Cell**: Insert a new code cell and write a simple command in the selected kernel:

```
print("Running in a different kernel")
```

5.4. Reset the kernel and clear the outputs

1. **Reset Kernel**: Go to Kernel > Restart & Clear Output. This will reset the kernel and clear all outputs in the notebook.

Task 6: Exporting Notebooks

6.1. Explore different file formats for exporting your notebook

- **File Formats**: Jupyter Notebooks can be exported in various formats, including PDF, HTML, and slideshows.

6.2. Export your notebook

1. **Export Notebook**: Go to File > Download as and select the desired format (e.g., .ipynb, .pdf, .html).
2. **Save**: The exported file will be saved to your default downloads folder. You can then move it to your desired location.

Task 7: Viewing Notebooks Online

7.1. Learn about nbviewer

- **nbviewer**: An online tool to view Jupyter Notebooks without running them. Visit nbviewer to view any notebook by providing its URL.

7.2. Upload your notebook to a public GitHub repository

1. **Create GitHub Repository**: Create a new repository on GitHub.
2. **Upload Notebook**: Upload your Jupyter Notebook (.ipynb file) to the repository.

7.3. Use nbviewer to view your notebook online

1. **Get Notebook URL**: Copy the URL of your uploaded notebook on GitHub.
2. **View in nbviewer**: Paste the URL in nbviewer to view your notebook online.

Task 8: Interactive Widgets in Jupyter

8.1. Learn about interactive widgets in Jupyter

- **Interactive Widgets**: Widgets in Jupyter Notebooks allow for interactive elements like sliders, buttons, and dropdowns that can be used to control the input to your code and dynamically update outputs.

8.2. Create a simple interactive widget

1. **Install ipywidgets**: If not already installed, you can install the ipywidgets library:

```
!pip install ipywidgets
```

2. **Import and Create a Slider**: Insert a new code cell and write:

```
import ipywidgets as widgets
from IPython.display import display

# Create a slider widget
slider = widgets.IntSlider(value=5, min=0, max=10, step=1,
description='Number:')
display(slider)
```

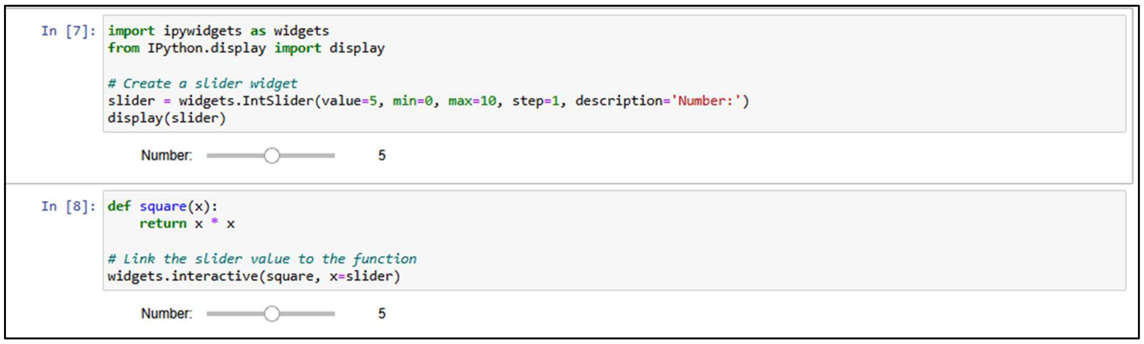
8.3. Link the widget to a Python function

1. **Define a Function and Link to Widget**: Insert a new code cell and write:

```
def square(x):
    return x * x

# Link the slider value to the function
widgets.interactive(square, x=slider)
```

2. **Dynamic Output**: The output will update dynamically as you move the slider.

Output: Task 8



```
In [7]: import ipywidgets as widgets
        from IPython.display import display

        # Create a slider widget
        slider = widgets.IntSlider(value=5, min=0, max=10, step=1, description='Number:')
        display(slider)
```

```
In [8]: def square(x):
            return x * x

        # Link the slider value to the function
        widgets.interactive(square, x=slider)
```

Task 9: Jupyter Notebook Extensions

9.1. Learn about Jupyter Notebook extensions

- **Jupyter Notebook Extensions**: Extensions add extra functionality to Jupyter Notebooks, such as code folding, spell checking, and more.

9.2. Install and Demonstrate an Extension

1. **Install jupyter_contrib_nbextensions**: This is a popular collection of extensions for Jupyter Notebooks:

```
!pip install jupyter_contrib_nbextensions
!jupyter contrib nbextension install –user
```

2. **Enable an Extension**: After installation, you can enable extensions through the Jupyter Notebook interface:

    - Go to the Jupyter home page.
    - Click on the "Nbextensions" tab.
    - Enable desired extensions by checking the corresponding boxes.

3. **Demonstrate an Extension**: For example, enable the "Table of Contents (2)" extension:

    - This will add a table of contents to your notebook, which updates automatically based on the headings in your Markdown cells.

Lab Record 2: Introduction to Colab

Task 1: Using Google Colab

1.1. Access Google Colab and create a new notebook

1. **Visit Colab**: Go to Google Colab.
2. **Create New Notebook**: Click on "New Notebook" to create a new notebook.

1.2. Familiarize yourself with the Colab interface

- **Menu and Toolbar**: Similar to Jupyter Notebook, Colab has menus for file operations, editing cells, and runtime management.
- **Cells**: You can insert Code and Text cells using the "+ Code" and "+ Text" buttons.

1.3. Write and execute a simple Python command

1. **Insert Code Cell**: Click "+ Code" and write:

```
print("Hello, Colab!")
```

2. **Run Cell**: Press Shift + Enter to execute the code.

1.4. Create and manipulate Python lists and dictionaries

1. **Lists and Dictionaries**: Write and execute the following code:

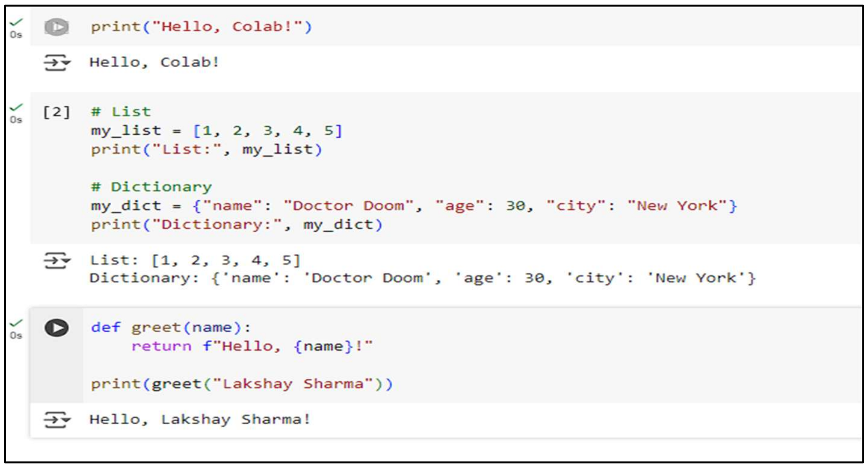
```
# List
my_list = [1, 2, 3, 4, 5]
print("List:", my_list)

# Dictionary
my_dict = {"name": "Doctor Doom", "age": 30, "city": "New York"}
print("Dictionary:", my_dict)
```

1.5. Define and call a simple Python function

1. **Function**: Write and execute the following code:

```
def greet(name):
    return f"Hello, {name}!"

print(greet("Lakshay Sharma"))
```

Output Task 1



```
print("Hello, Colab!")
```

```
Hello, Colab!
```

```
[2] # List
my_list = [1, 2, 3, 4, 5]
print("List:", my_list)

# Dictionary
my_dict = {"name": "Doctor Doom", "age": 30, "city": "New York"}
print("Dictionary:", my_dict)
```

```
List: [1, 2, 3, 4, 5]
Dictionary: {'name': 'Doctor Doom', 'age': 30, 'city': 'New York'}
```

```
def greet(name):
    return f"Hello, {name}!"

print(greet("Lakshay Sharma"))
```

```
Hello, Lakshay Sharma!
```

Task 2: Mounting Google Drive in Colab

2.1. Mount Google Drive

1. **Mount Drive**: Insert a new code cell and write:

```
from google.colab import drive
drive.mount('/content/drive')
```

2. **Authenticate**: Follow the prompts to authenticate and mount your Google Drive.

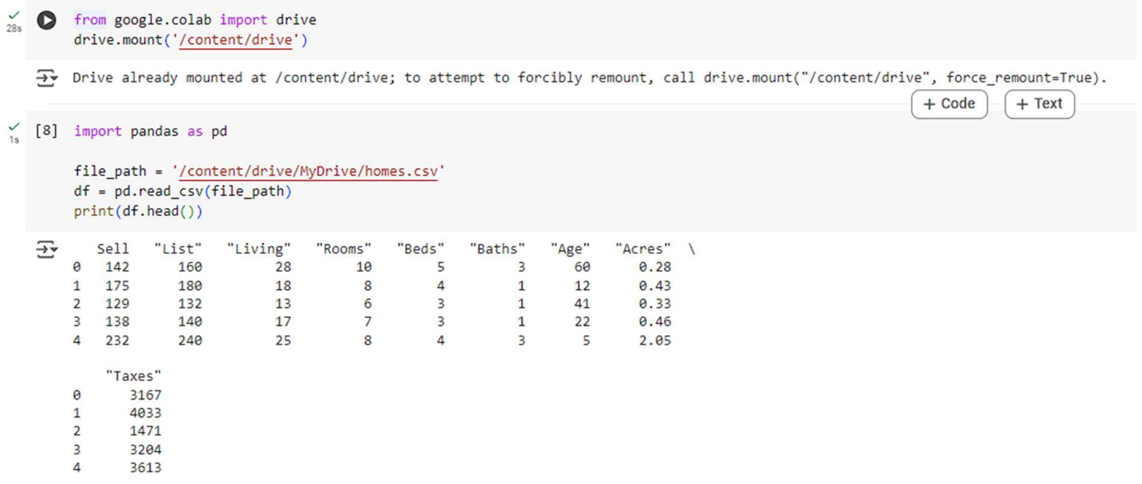
2.2. Access a file from Google Drive

1. **Navigate Directory**: Use the file explorer on the left side of Colab to navigate through your Drive.

2. **Read a CSV File**: Write and execute the following code to read a CSV file into a pandas DataFrame:

```
import pandas as pd

file_path = '/content/drive/My Drive/path_to_your_file.csv'
df = pd.read_csv(file_path)
print(df.head())
```

Output Task 2



```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[8] import pandas as pd

    file_path = '/content/drive/MyDrive/homes.csv'
    df = pd.read_csv(file_path)
    print(df.head())
```

```
   Sell  "List"  "Living"  "Rooms"  "Beds"  "Baths"  "Age"  "Acres"  \
0   142     160        28       10       5        3     60     0.28
1   175     180        18        8       4        1     12     0.43
2   129     132        13        6       3        1     41     0.33
3   138     140        17        7       3        1     22     0.46
4   232     240        25        8       4        3      5     2.05

    "Taxes"
0      3167
1      4033
2      1471
3      3204
4      3613
```

Task 3: Basic Data Analysis and Visualization

3.1. Perform basic data analysis operations

1. **Sorting and Filtering**: Write and execute the following code:

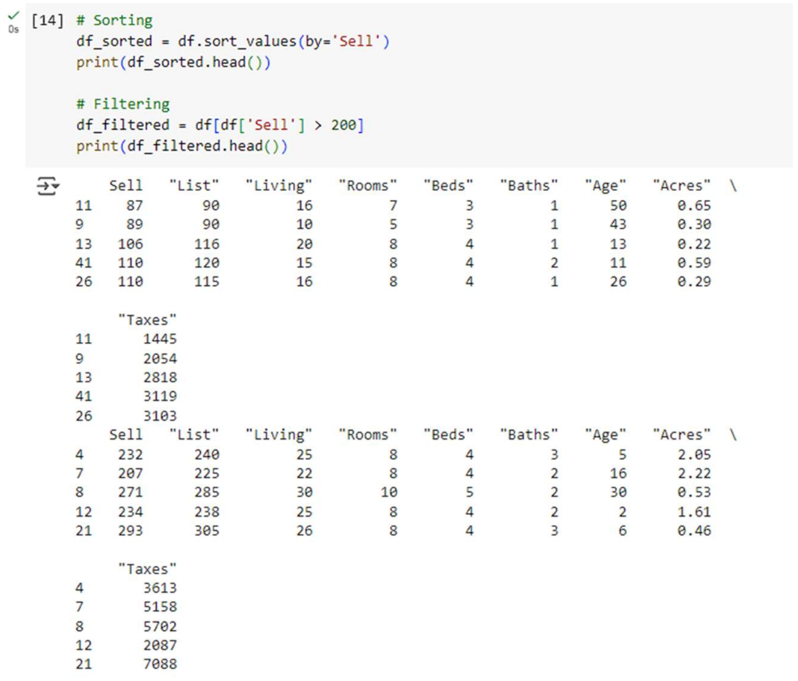
```
# Sorting
df_sorted = df.sort_values(by='column_name')
print(df_sorted.head())

# Filtering
df_filtered = df[df['column_name'] > some_value]
print(df_filtered.head())
```

3.2. Create a simple visualization

1. **Install and Import Libraries**: Write and execute the following code:

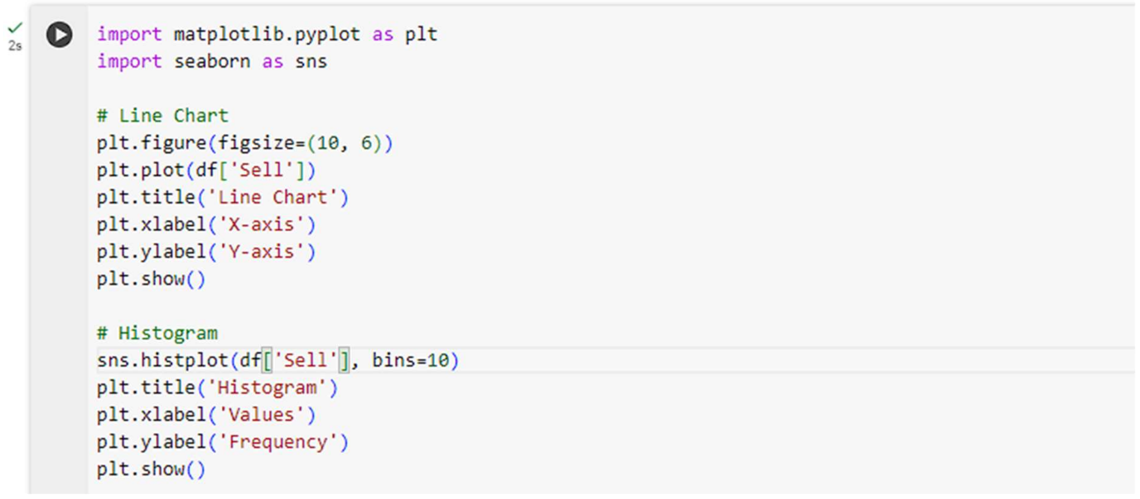
```
import matplotlib.pyplot as plt
import seaborn as sns

# Line Chart
plt.figure(figsize=(10, 6))
plt.plot(df['column_name'])
plt.title('Line Chart')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()

# Histogram
sns.histplot(df['column_name'], bins=10)
plt.title('Histogram')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.show()
```

Output Task 3

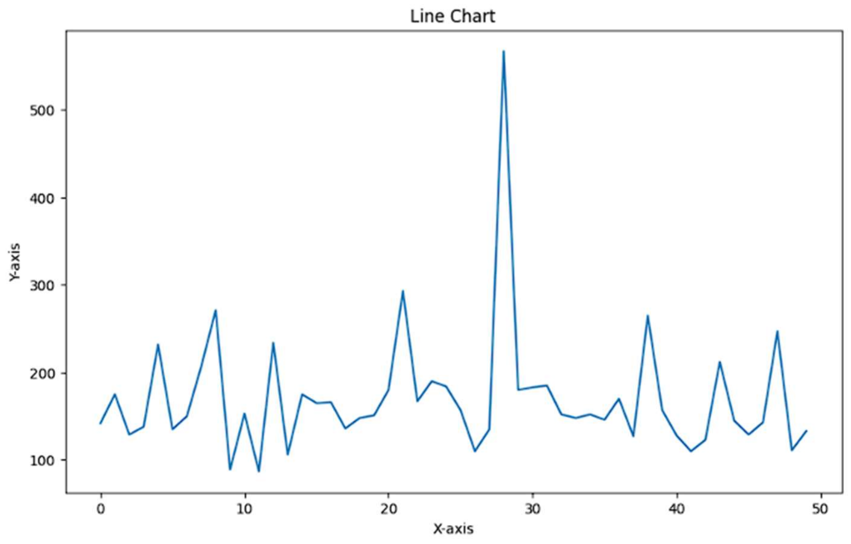


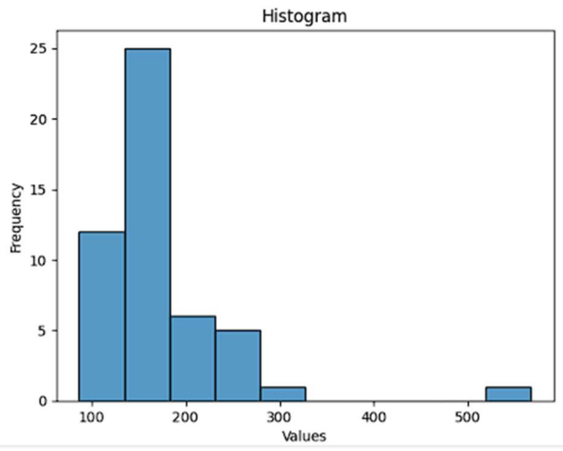
```
[14] # Sorting
     df_sorted = df.sort_values(by='Sell')
     print(df_sorted.head())

     # Filtering
     df_filtered = df[df['Sell'] > 200]
     print(df_filtered.head())
```

```
     Sell   "List"  "Living"  "Rooms"  "Beds"  "Baths"  "Age"  "Acres"  \
11     87       90        16        7       3        1     50     0.65
9      89       90        10        5       3        1     43     0.30
13    106      116        20        8       4        1     13     0.22
41    110      120        15        8       4        2     11     0.59
26    110      115        16        8       4        1     26     0.29

     "Taxes"
11      1445
9       2054
13      2818
41      3119
26      3103
     Sell   "List"  "Living"  "Rooms"  "Beds"  "Baths"  "Age"  "Acres"  \
4     232      240        25        8       4        3      5     2.05
7     207      225        22        8       4        2     16     2.22
8     271      285        30       10       5        2     30     0.53
12    234      238        25        8       4        2      2     1.61
21    293      305        26        8       4        3      6     0.46

     "Taxes"
4       3613
7       5158
8       5702
12      2087
21      7088
```



```
import matplotlib.pyplot as plt
import seaborn as sns

# Line Chart
plt.figure(figsize=(10, 6))
plt.plot(df['Sell'])
plt.title('Line Chart')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()

# Histogram
sns.histplot(df['Sell'], bins=10)
plt.title('Histogram')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.show()
```





Task 4: Exploring Libraries and Installing New Ones

4.1. Explore pre-installed libraries

- **List Libraries**: You can view the list of pre-installed libraries in Colab by writing:

```
!pip list
```

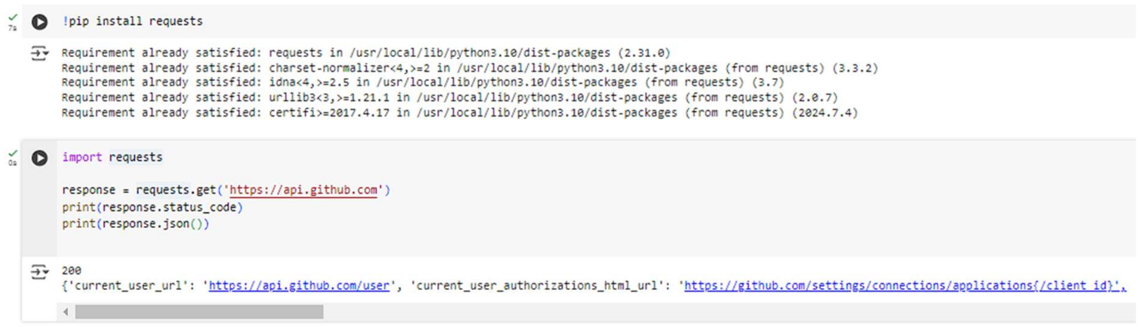
4.2. Install a new library

1. **Install Library**: For example, install the requests library:

```
!pip install requests
```

2. **Demonstrate Functionality**: Use the installed library:

```
import requests

response = requests.get('https://api.github.com')
print(response.status_code)
print(response.json())
```



```
!pip install requests
```

```
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (2.31.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests) (2024.7.4)
```

```
import requests

response = requests.get('https://api.github.com')
print(response.status_code)
print(response.json())
```

```
200
{'current_user_url': 'https://api.github.com/user', 'current_user_authorizations_html_url': 'https://github.com/settings/connections/applications{/client_id}',
```

Task 5: Collaboration and Sharing

5.1. Share your Colab notebook

1. **Share Notebook**: Click the "Share" button in the top right corner of Colab.

2. **Get Link**: You can get a shareable link to your notebook and adjust the sharing settings (e.g., view only, edit permissions).

5.2. Use Comments and Version History

1. **Comments**: Click the "Comments" icon to add comments to specific cells for collaboration.

2. **Version History**: Access version history by going to File > Revision history to see changes made to the notebook over time.

Task 6: Saving and Exporting Notebooks

6.1. Save your notebook to Google Drive

1. **Save Notebook**: Your Colab notebook is automatically saved to your Google Drive under "Colab Notebooks."

6.2. Export your notebook

1. **Export Formats**: Go to File > Download .ipynb or Download .pdf to export your notebook in the desired format.
2. **Save to Drive**: The exported file will be saved to your default downloads folder, and you can move it to your Google Drive.