# 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.

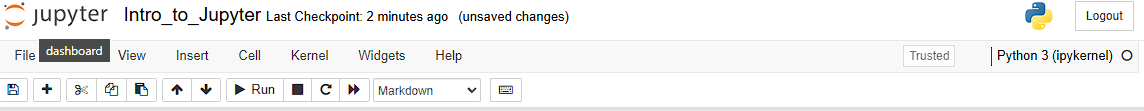
### 1.3. Familiarize yourself with the interface

* **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

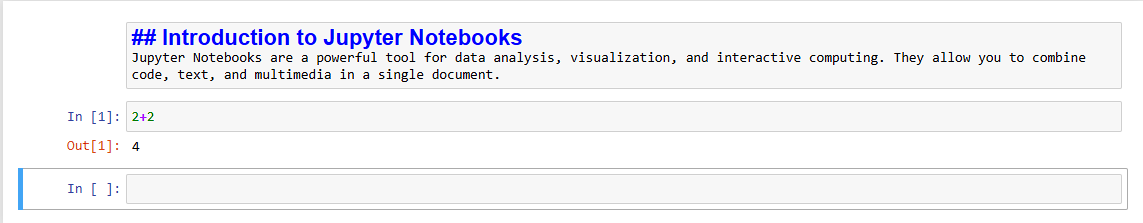
### 2.4. Execute the Code cell

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**



## 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)

1. **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

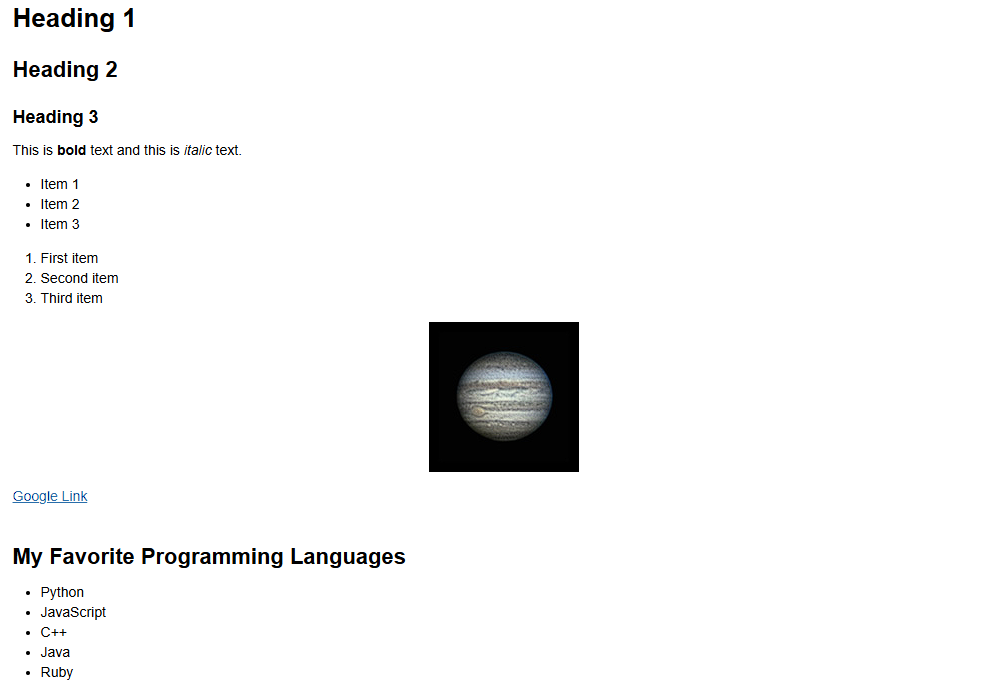
- C++

- Java

- Ruby

**Output: Task 3**





## 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)

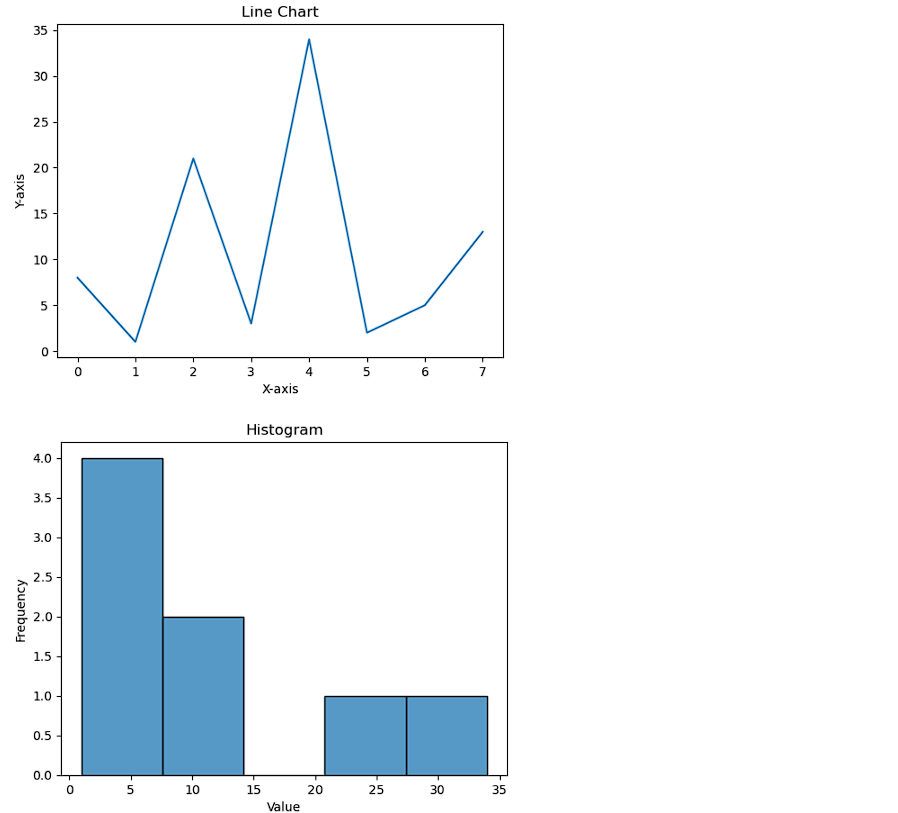
plt.title('Histogram')

plt.xlabel('Value')

plt.ylabel('Frequency')

plt.show()

**Output: Task 4**

## 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

1. **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)

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

**Output: Task 8**



## 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

1. **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.
2. **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](https://colab.research.google.com).
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!")

1. **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**



## 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')

1. **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:

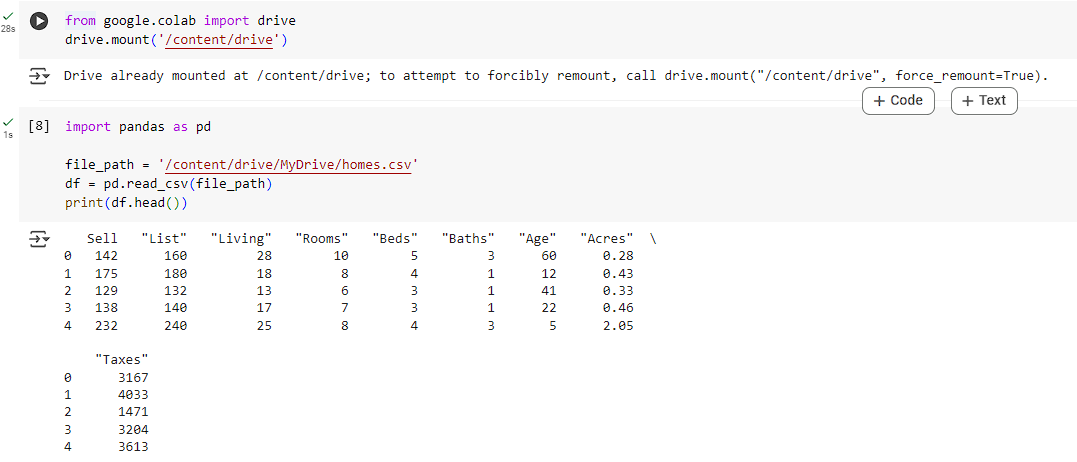
import pandas as pd

file\_path = '/content/drive/My Drive/path\_to\_your\_file.csv'

df = pd.read\_csv(file\_path)

print(df.head())

**Output Task 2**



## 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)

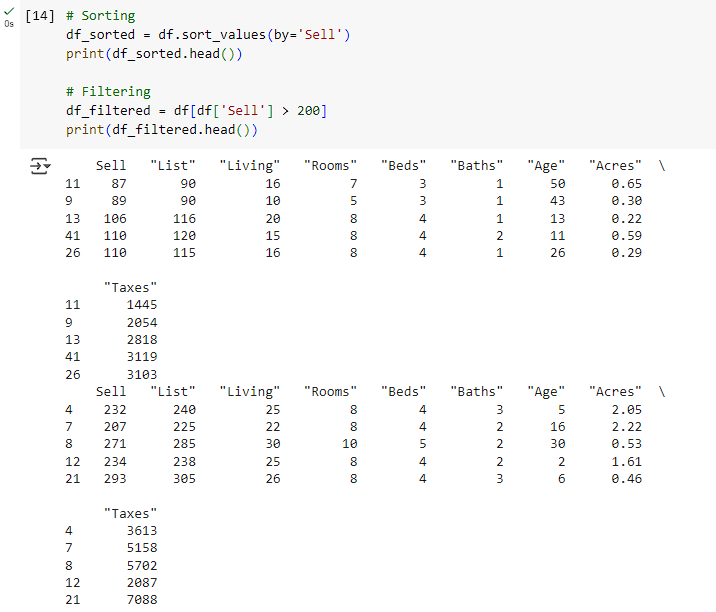
plt.title('Histogram')

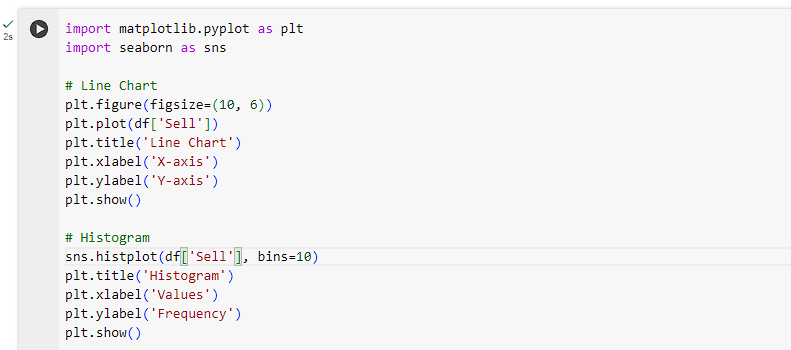
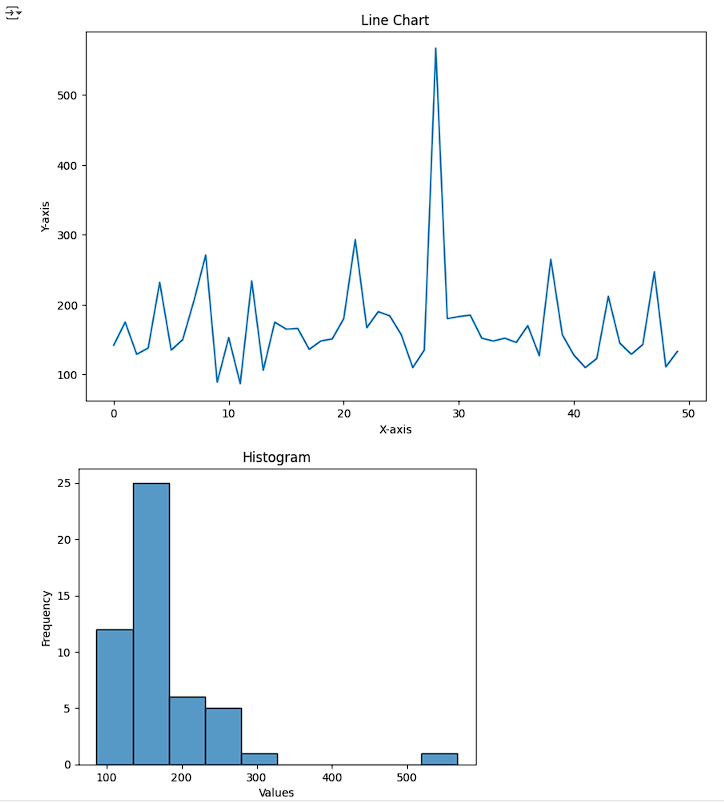
plt.xlabel('Values')

plt.ylabel('Frequency')

plt.show()

**Output Task 3**



## 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

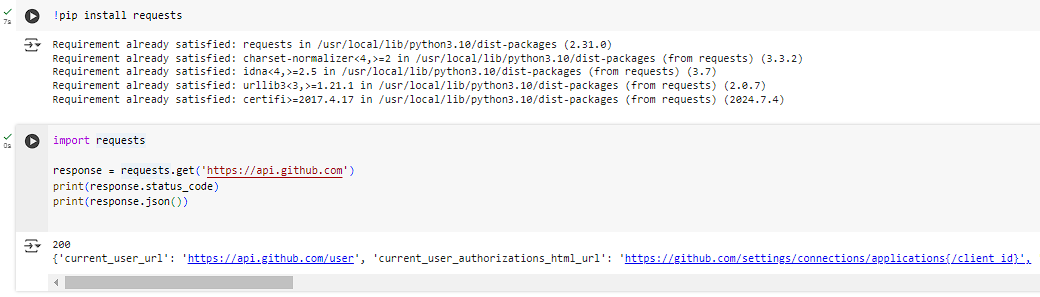
1. **Demonstrate Functionality**: Use the installed library:

import requests

response = requests.get('https://api.github.com')

print(response.status\_code)

print(response.json())



## 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.

# **Lab Record: Introduction to Linear Regression**

## **Introduction**

Linear regression is a fundamental statistical method used to model the relationship between a dependent variable and one or more independent variables. It assumes a linear relationship between the variables, meaning that a change in the independent variable(s) results in a proportional change in the dependent variable. This lab explores linear regression using various techniques, including machine learning and traditional statistical methods, to analyze the relationship between advertising budgets and sales.

## **Dataset and Initial Analysis**

The dataset used in this lab is the "Advertising" dataset, which contains information about advertising budgets for TV, radio, and newspapers, along with the corresponding sales figures. The dataset was loaded from a GitHub repository and initially analyzed using scatter plots to visualize the relationships between the variables.

## **Initial Observations:**

* The scatter plot of TV advertising budget vs. sales suggests a positive linear relationship, indicating that higher TV advertising budgets tend to be associated with higher sales.
* The scatter plot of radio advertising budget vs. sales also shows a positive linear relationship, but it appears to be weaker than the relationship between TV advertising and sales.
* The scatter plot of newspaper advertising budget vs. sales does not exhibit a clear linear relationship, suggesting that newspaper advertising may not be a strong predictor of sales.

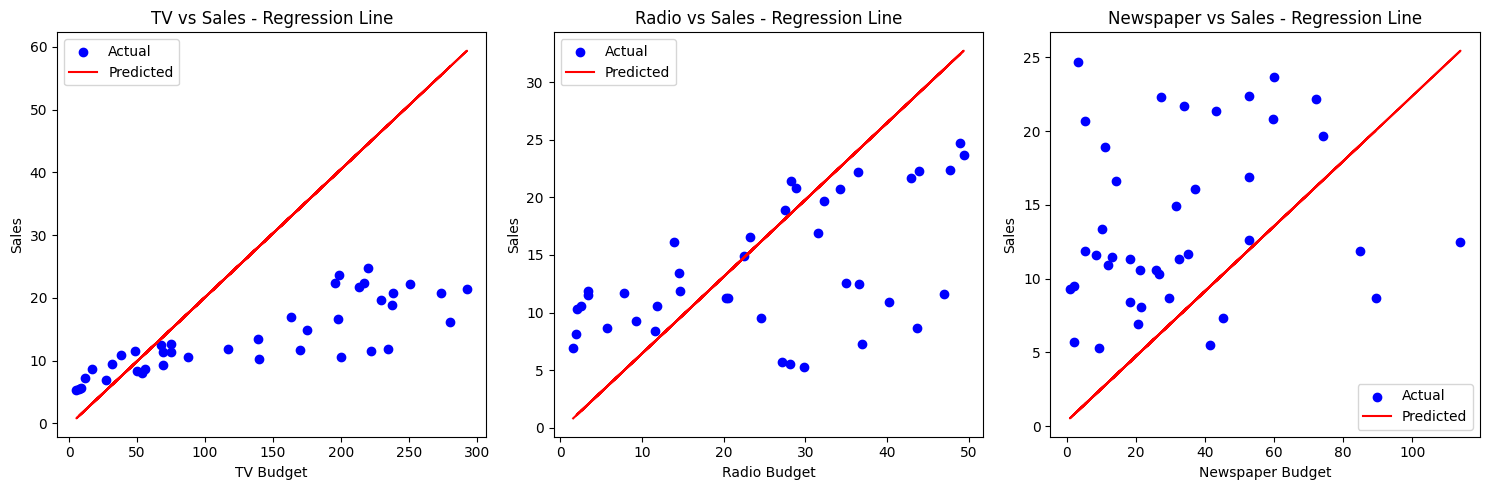
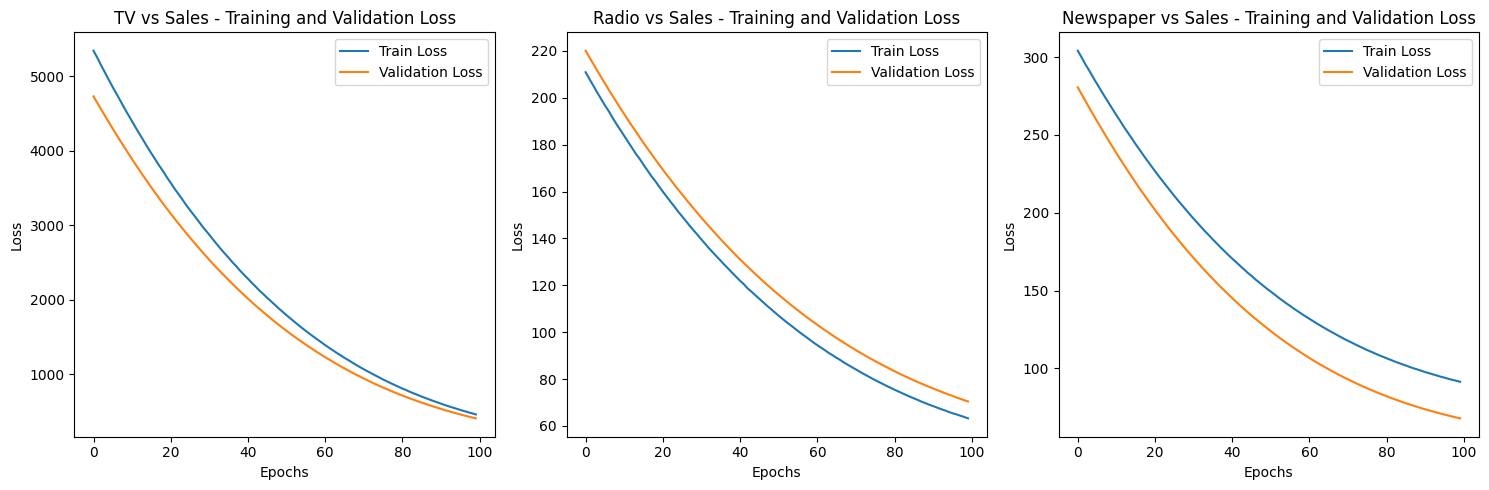
## **Models Description**

### 1. Linear Regression Model using Machine Learning

This model utilizes machine learning libraries, specifically TensorFlow and Keras, to build a linear regression model. The data for each advertising medium (TV, radio, newspaper) was split into training and testing sets. Separate neural networks with a single dense layer were created for each medium. The models were trained using the Adam optimizer and mean squared error loss function.

#### Key Points:

* The models were evaluated based on their test loss, which measures the average squared difference between predicted and actual sales values on the test set.
* Training and validation loss curves were plotted to monitor the learning process and identify potential overfitting.
* Regression lines were plotted to visualize the predicted relationship between advertising budgets and sales for each medium.

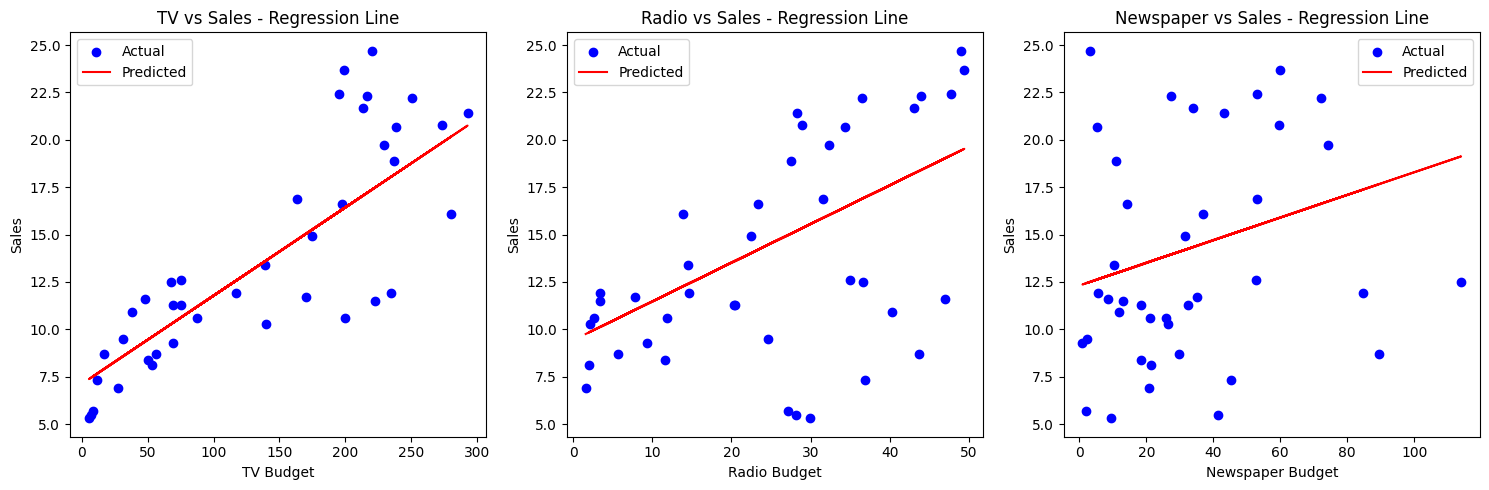


### 2. Linear Regression Model using Scikit-learn (without ML Libraries)

This model employs the LinearRegression class from the scikit-learn library, which provides a traditional statistical approach to linear regression. The data was prepared and split similarly to the machine learning model.

#### Key Points:

* The models were trained using the fit method, which estimates the coefficients of the linear regression equation.
* The models were evaluated using mean squared error (MSE), a common metric for regression problems.
* Regression lines were plotted to visualize the predicted relationship between advertising budgets and sales for each medium.

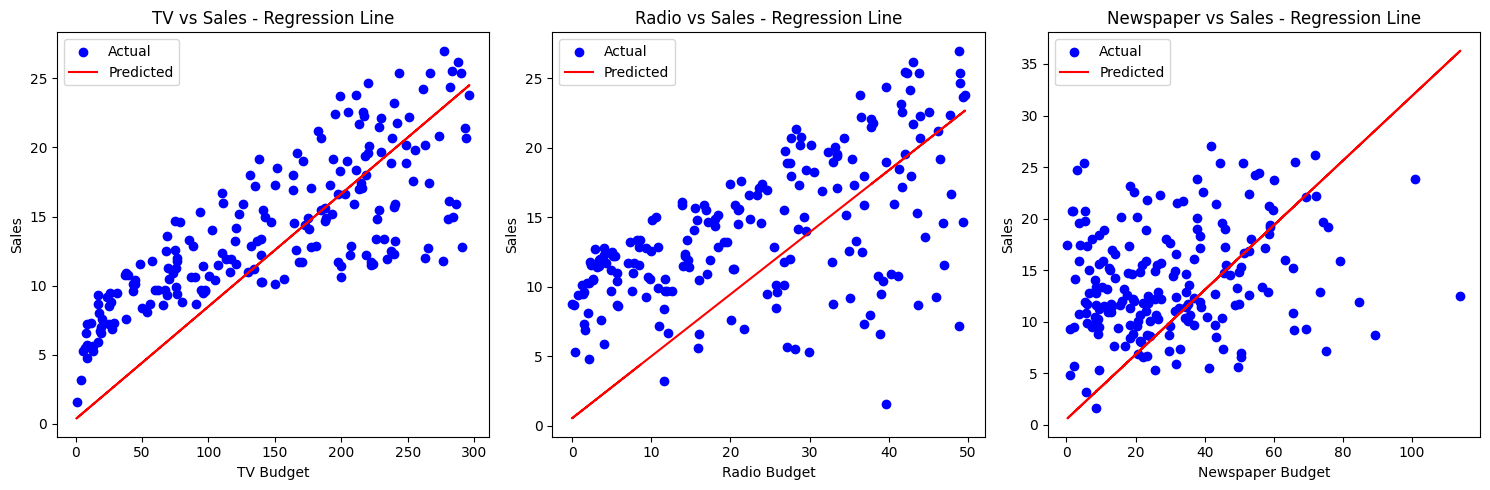


### 3. Linear Regression Model without using Scikit-learn or ML Libraries

This section explores two approaches to linear regression without relying on external libraries:

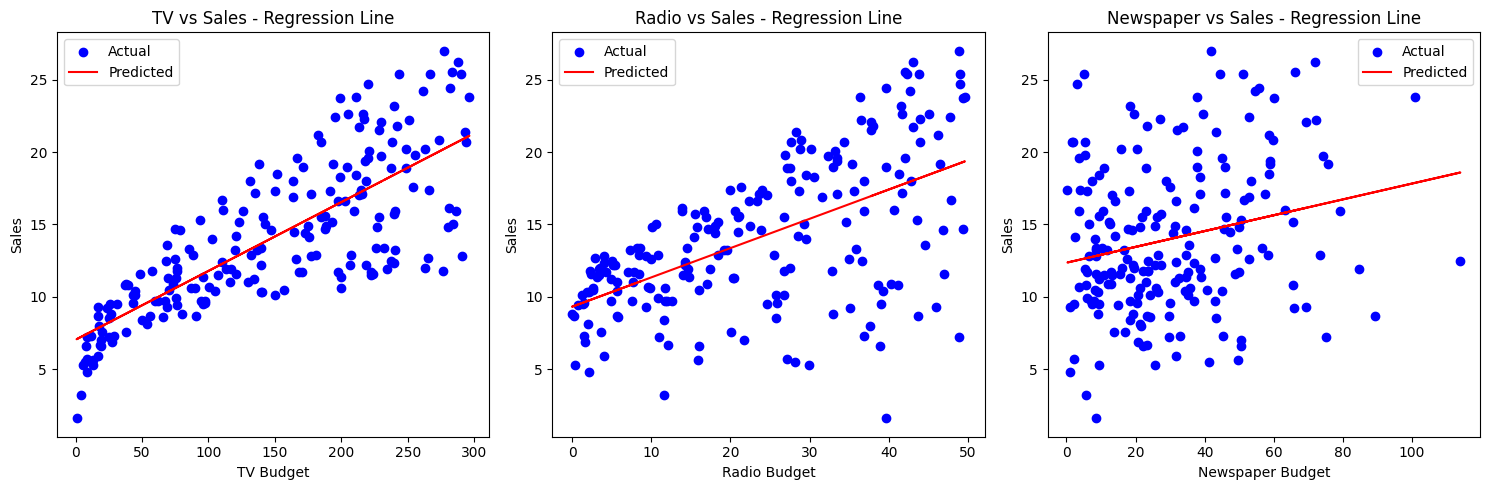
#### A. Using Machine Learning Principles:

* A custom implementation of the Adam optimizer was used to update the weights and bias of the linear regression model iteratively.
* The mean squared error loss function was used to measure the model's performance.
* The model was trained for a specified number of epochs, and the loss history was recorded.
* Regression lines were plotted to visualize the predicted relationship between advertising budgets and sales for each medium.



#### B. Using Coefficient of Regression:

* The slope and intercept of the linear regression equation were calculated manually using the formula for the coefficient of regression.
* The predicted sales values were calculated using the estimated slope and intercept.
* Regression lines were plotted to visualize the predicted relationship between advertising budgets and sales for each medium.



## **Results**

The results of the linear regression analysis are presented in the form of plots, including:

* **Scatter Plots:** These plots show the relationship between each advertising medium and sales, providing an initial visual assessment of the data.
* **Training and Validation Loss Curves:** These plots (for the machine learning models) illustrate the model's learning progress and help identify potential overfitting.
* **Regression Lines:** These plots depict the predicted relationship between advertising budgets and sales for each medium, based on the estimated coefficients of the linear regression models.

## **Observations from the Plots:**

* The machine learning models and the scikit-learn models produced similar regression lines, indicating that both approaches effectively captured the linear relationship between advertising budgets and sales.
* The custom implementation of linear regression using machine learning principles also yielded comparable results, demonstrating the feasibility of building a linear regression model from scratch.
* The regression lines obtained using the coefficient of regression method were consistent with the other models, confirming the validity of this traditional statistical approach.

## **Conclusion**

This lab explored various techniques for performing linear regression analysis, ranging from machine learning models to traditional statistical methods. The results demonstrate that linear regression is a powerful tool for modeling the relationship between variables and predicting outcomes. The choice of method depends on the specific requirements of the analysis, such as the need for scalability, interpretability, or computational efficiency.