S. B. JAIN INSTITUTE OF TECHNOLOGY,

MANAGEMENT & RESEARCH, NAGPUR.

Practical No. 01 [Pre-Lab]

Aim: Understanding Google Colab and various libraries of Python.

Name of Student : Shrutika Pradeep Bagdi

Roll No : CS22130

Semester/Year : VIIth Sem / IVth Year

Academic Session : 2025-2026 [ODD]

Date of Performance : _____

Date of Submission : _____

AIM: Understanding Google Colab and various libraries of Python.

THEORY:

GOOGLE COLABORATORY :

Colab is a hosted Jupyter Notebook service that requires no setup to use and provides free access to computing resources, including GPUs and TPUs. Colab is especially well suited to machine learning, data science, and education.

Google Colab, short for Google Colaboratory, is a free, cloud-based platform provided by Google that allows users to write and execute Python code collaboratively. Here are some key aspects and theories related to Google Colab:

Cloud Computing and Accessibility:

- Google Colab leverages cloud computing resources, allowing users to run their code on powerful machines without the need for high-end local hardware.
- This cloud-based approach makes it accessible to a broader audience, as users can access the platform from anywhere with an internet connection.

Jupyter Notebooks:

- Google Colab is built on the Jupyter Notebook environment, which is widely used in data science and machine learning.
- Jupyter Notebooks allow users to create and share documents containing live code, equations, visualizations, and narrative text.

Integration with Google Drive:

- Colab integrates seamlessly with Google Drive, enabling users to save their work directly to their Google Drive accounts.
- This integration facilitates easy sharing and collaboration among users, making it a collaborative and user-friendly platform.

Free Access to GPU and TPU:

- Google Colab provides free access to Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs), which are critical for accelerating computations in tasks such as deep learning.
- This access to specialized hardware is particularly valuable for researchers and developers working on computationally intensive tasks.

Version Control and History:

- Colab maintains version control of notebooks, allowing users to track changes, revert to previous versions, and collaborate more effectively.
- The version history feature enhances reproducibility and collaboration in data science projects.

Libraries and Dependencies:

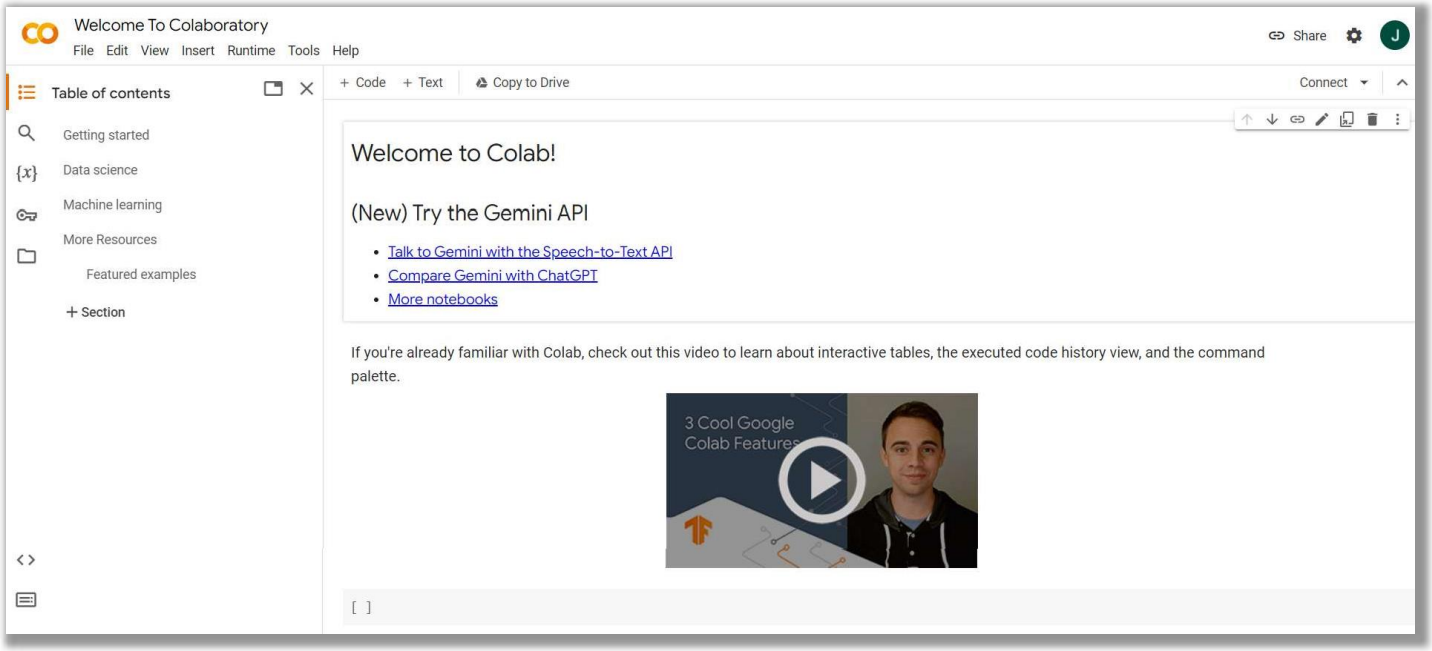
- Colab comes pre-installed with many popular Python libraries used in data science and machine learning, such as NumPy, Pandas, Matplotlib, TensorFlow, and PyTorch.
- Users can install additional libraries as needed, making it a versatile environment for various programming tasks.

Educational Use:

- Google Colab is often used in educational settings to teach programming, data science, and machine learning.
- Its ease of use, coupled with free access to computing resources, makes it an attractive choice for educators and students.

Community and Documentation:

- Colab has a vibrant user community, and Google provides extensive documentation and tutorials to help users get started and troubleshoot issues.
- This community support enhances the learning experience and helps users make the most of the platform.



Welcome to Colab!

Getting started

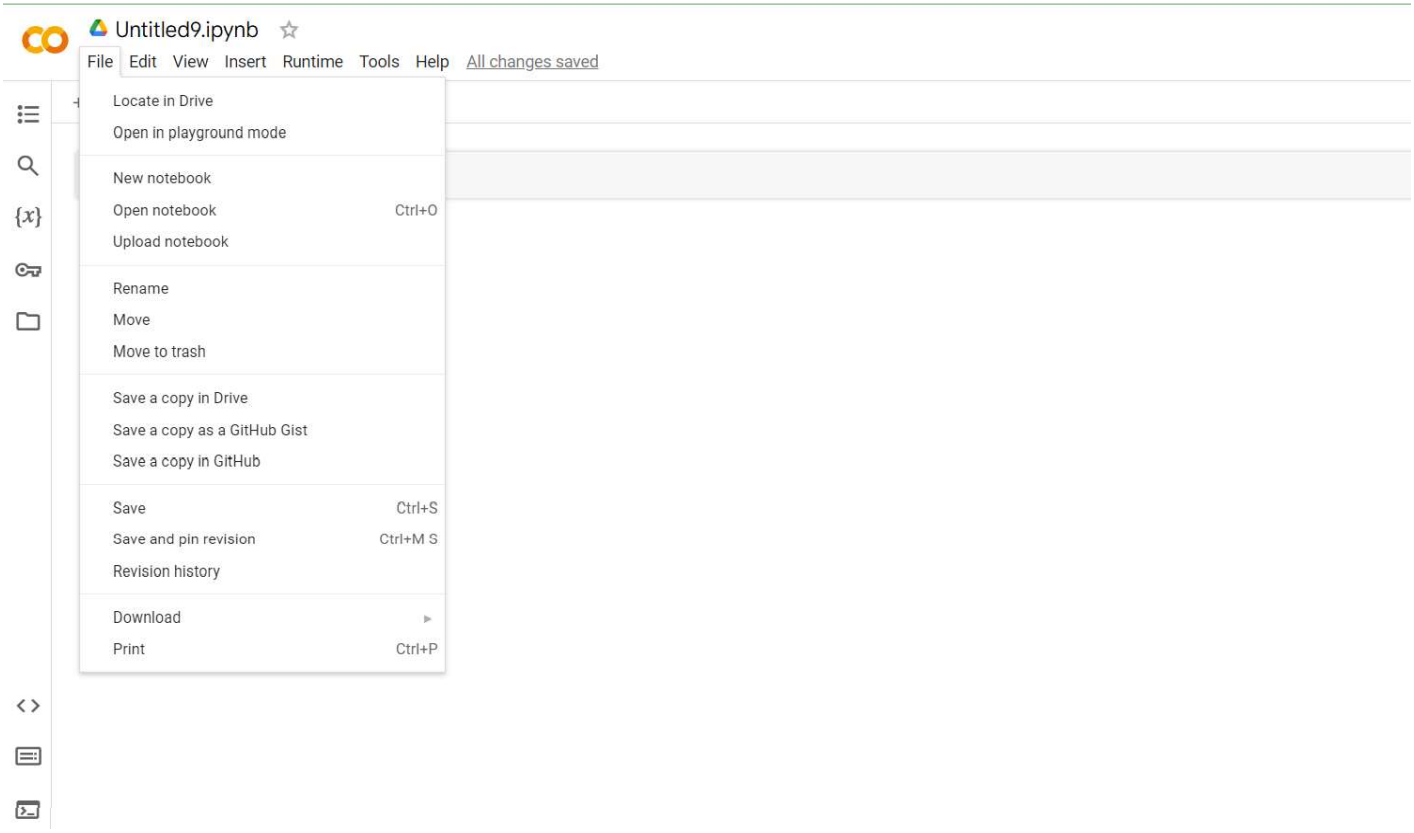
The document you are reading is not a static web page, but an interactive environment called a Colab notebook that lets you write and execute code.

Data science

With Colab you can harness the full power of popular Python libraries to analyze and visualize data. The code cell below uses numpy to generate some random data, and uses matplotlib to visualize it. To edit the code, just click the cell and start editing.

Machine learning

With Colab you can import an image dataset, train an image classifier on it, and evaluate the model, all in just a few lines of code. Colab notebooks execute code on Google's cloud servers, meaning you can leverage the power of Google hardware, including GPUs and TPUs, regardless of the power of your machine.



1. New Notebook: Create a new, blank Jupyter notebook in Google Colab.

2. Open Notebook: Open an existing Jupyter notebook from Google Drive or upload from a local machine.

3. Upload Notebook: Upload a Jupyter notebook file (.ipynb) from a local machine to Google Colab.

4. Save a Copy in Drive: Save a duplicate copy of the current notebook to your Google Drive account.

5. Save: Save the changes made to the current notebook to the existing file on Google Drive.

6. Save and Pin Revision: Save the current version of the notebook and pin it as a revision for version control.

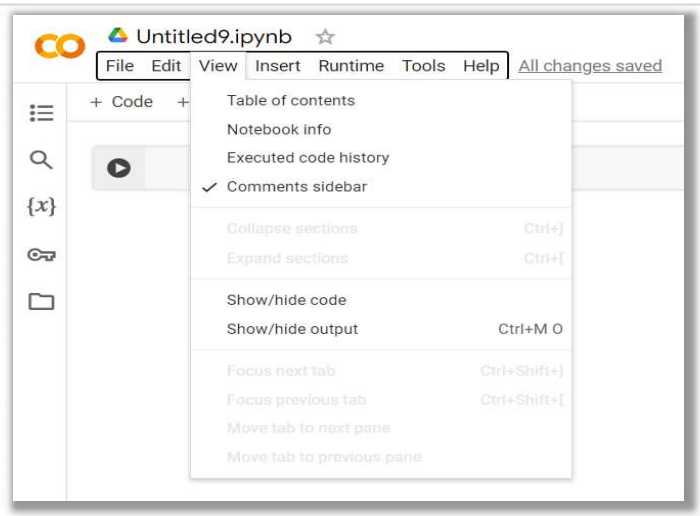
7. Revision History : View and restore previous versions of the notebook from the revision history.

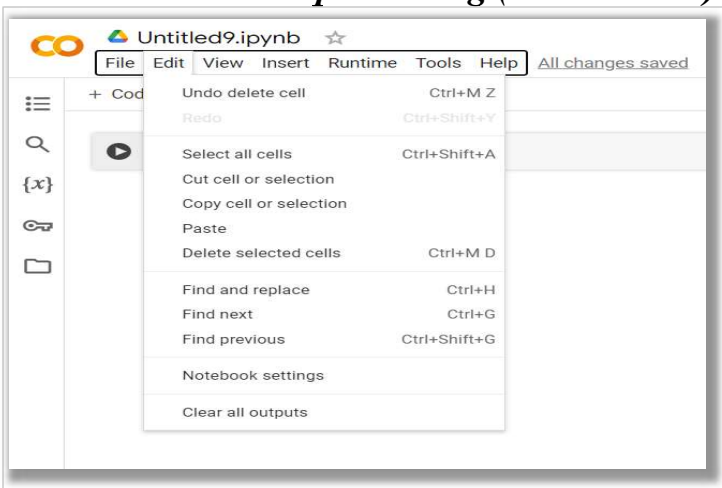
8. Download: Download the current notebook in various formats, such as .ipynb, .py, or .html.

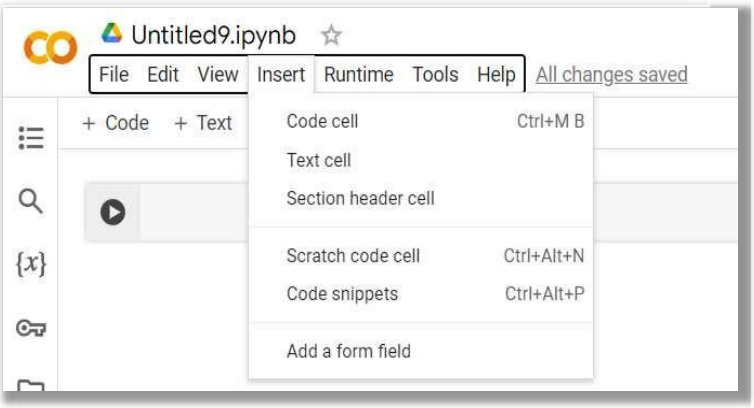
9. Convert to Form: Convert the current notebook into an interactive Google Form.

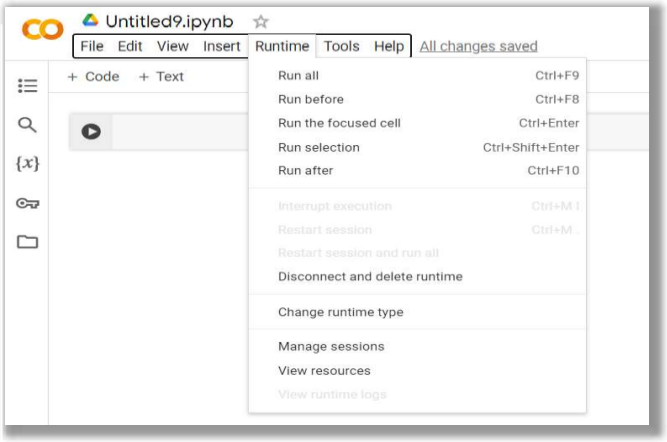
10. Convert to PDF: Export the current notebook as a PDF file.

11. Print: Print the current notebook or save it as a PDF for physical or digital documentation.









Edit:

- **Cut, Copy, Paste**: Basic text manipulation functions for efficient notebook editing.
- **Undo, Redo:** Reverse or repeat previous actions, enhancing the editing experience.
- **Find and Replace:** Locate specific text and replace it with another, aiding in code exploration and modification.

View:

- **Toolbar:** Toggle the visibility of the toolbar for a cleaner interface or additional screen space.
- **Header/Footer:** Customize the visibility of the header and footer to optimize the display of the notebook.
- **Cell Toolbar:** Control the visibility of the cell toolbar, offering options for cell manipulation.

Insert:

- **Insert Code Cell:** Add a new code cell to the notebook for coding purposes.
- **Insert Text Cell**: Include a text cell to provide explanations, documentation within the notebook.
- **Insert Image**: Embed images directly into the notebook for enhanced data visualization or documentation.

Runtime:

- **Change Runtime Type:** Modify the runtime environment, enabling users to switch between CPU and GPU resources.
- **Run All:** Execute all cells in the notebook in sequential order, automating code execution.
- **Run Before/Run After:** Execute cells that come before or after the currently selected cell, streamlining code execution.

CODE:

Pytorch:

1)
```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
```

2)
```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(784, 128)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(128, 10)
def forward(self, x):
    x = self.relu(self.fc1(x))
    return self.fc2(x)
```

3)
```
transform = transforms.Compose([transforms.ToTensor(), transforms.Lambda(lambda x: x.view(-1))])
train_data = datasets.MNIST(root='./data', train=True, download=True, transform=transform)
train_loader = torch.utils.data.DataLoader(train_data, batch_size=64, shuffle=True)
```

4)
```
model = Net()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

5)
```
# Training loop
for epoch in range(5):
    for images, labels in train_loader:
        outputs = model(images)
        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
print(f"Epoch {epoch+1}, Loss: {loss.item():.4f}")
```

TensorFlow:

1)
```
import tensorflow as tf
from tensorflow.keras import layers, models
```

2)
```
# Define a simple feedforward neural network
model = models.Sequential([
    layers.Dense(128, activation='relu', input_shape=(784,)),
    layers.Dropout(0.2),
```

```
    layers.Dense(10, activation='softmax')
```

3)
```
# Compile the model
model.compile(optimizer='adam',
      loss='sparse_categorical_crossentropy',
          metrics=['accuracy'])
```

4)
```
# Load and preprocess data (e.g., MNIST)
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train = x_train.reshape(-1, 784).astype("float32") / 255
x_test = x_test.reshape(-1, 784).astype("float32") / 255
```

5)
```
# Train the model
model.fit(x_train, y_train, epochs=5, batch_size=32)
```

6)
```
# Evaluate
model.evaluate(x_test, y_test)
```

Keres:

1)
```
from tensorflow import keras
from tensorflow.keras import layers
```

2)
```
# Define a Sequential model
model = keras.Sequential([
   layers.Dense(128, activation='relu', input_shape=(784,)),
   layers.Dropout(0.3),
   layers.Dense(64, activation='relu'),
   layers.Dense(10, activation='softmax')
])
```

3)
```
# Compile the model
model.compile(optimizer='adam',
          loss='sparse_categorical_crossentropy',
          metrics=['accuracy'])
```

4)
```
# Load and preprocess MNIST data
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
x_train = x_train.reshape(-1, 784).astype("float32") / 255
x_test = x_test.reshape(-1, 784).astype("float32") / 255
```

5)
```
# Train the model
model.fit(x_train, y_train, epochs=5, batch_size=32)
```

6)
```
# Evaluate
model.evaluate(x_test, y_test)
```

OUTPUT:

Pytorch:

```
Epoch 1, Loss: 0.2017
Epoch 2, Loss: 0.0619
Epoch 3, Loss: 0.0108
Epoch 4, Loss: 0.0733
Epoch 5, Loss: 0.0840
```

TensorFlow:

```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8569 - loss: 0.4898
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9551 - loss: 0.1518
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.9683 - loss: 0.1043
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9741 - loss: 0.0849
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9766 - loss: 0.0740
<keras.src.callbacks.history.History at 0x7b9fc89a39d0>
```

```
[ ]
# Evaluate
model.evaluate(x_test, y_test)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9742 - loss: 0.0827
[0.07191614806652069, 0.9779000282287598]
```

Keres:

```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - accuracy: 0.8473 - loss: 0.5060
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9488 - loss: 0.1700
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9583 - loss: 0.1335
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9658 - loss: 0.1088
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9696 - loss: 0.0997
<keras.src.callbacks.history.History at 0x7b9fc8d76750>
```

```
[ ]
# Evaluate
model.evaluate(x_test, y_test)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9701 - loss: 0.0989
[0.08198625594377518, 0.973800003528595]
```

DISCUSSION AND VIVA VOCE:

1. What is NumPy?
2. What is Pandas?
3. What is Slicing?
4. What is Indexing?

CONCLUSION: In conclusion, Google Colab is a powerful and versatile cloud-based platform that facilitates collaborative Python programming. It provides free access to computing resources, including GPUs and TPUs, making it especially valuable for data science and machine learning tasks. The integration with Google Drive simplifies file management, version control, and collaboration. The combination of Google Colab and Python libraries empowers users to seamlessly work on data science projects, leveraging the collaborative features of Colab along with the rich functionality of libraries to explore, analyze, and visualize data.

REFERENCES:

- https://www.w3schools.com/python/numpy/default.asp
- https://www.w3schools.com/python/pandas/default.asp
- https://www.codecademy.com/article/introduction-to-numpy-and-pandas
- https://www.geeksforgeeks.org/difference-between-pandas-vs-numpy/
- https://colab.research.google.com/drive/1cKwzlBxACar5xP6RK8u7i-q963WepRXW?