GET STARTED WITH APPLIED MACHINE LEARNING ON GOOGLE CLOUD AND COLAB

LEARN DATA ENGINEERING AND TAKE YOUR FIRST STEP TOWARDS BECOMING A HANDS ON DATA SCIENTIST AND INTELLIGENT AUTOMATION ENGINEER

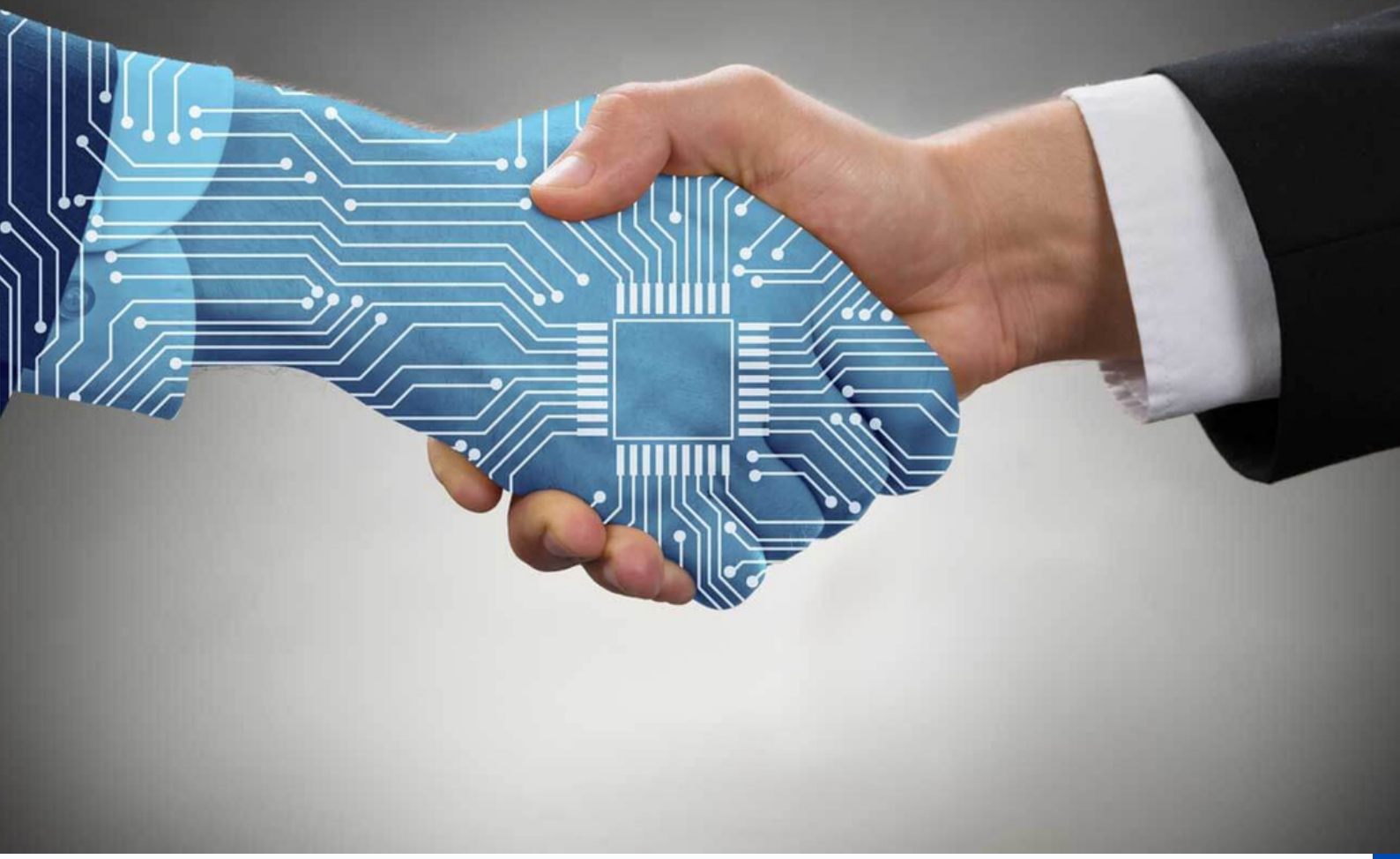
DeepSphere.AI
Enterprise AI and IIoT for Analytics

GET STARTED WITH APPLIED MACHINE LEARNING ON GOOGLE CLOUD AND COLAB

LEARN DATA ENGINEERING AND TAKE YOUR FIRST STEP TOWARDS BECOMING A HANDS ON DATA SCIENTIST AND INTELLIGENT AUTOMATION ENGINEER

DeepSphere.AI
Enterprise AI and IIoT for Analytics

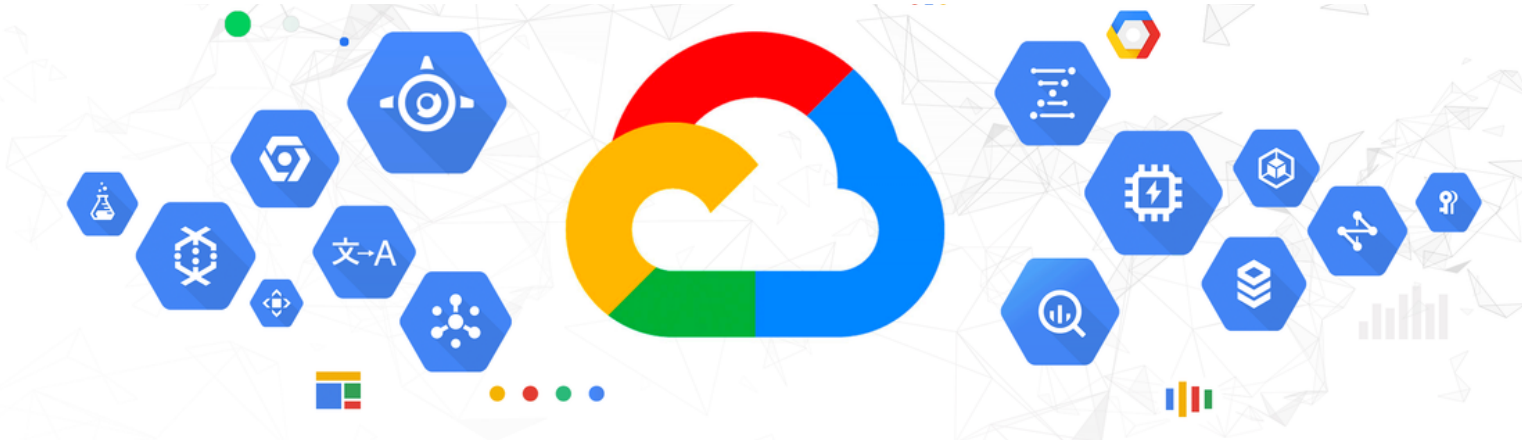
GET STARTED WITH APPLIED MACHINE LEARNING ON GOOGLE CLOUD AND COLAB



LEARN DATA ENGINEERING AND TAKE YOUR FIRST STEP TOWARDS BECOMING A HANDS ON DATA SCIENTIST AND INTELLIGENT AUTOMATION ENGINEER

DeepSphere.AI
Enterprise AI and IIoT for Analytics

DEEPSPHERE.AI AND GOOGLE CLOUD



DeepSphere.AI (DS.AI) is a global leader in providing an advanced higher education platform for schools. We provide an intelligent learning management system (iLMS) to learn applied artificial intelligence, data science, and data engineering at a personalized level. DS.AI's iLMS platform is hosted on Amazon web services (AWS) and the learning resources are developed on Google Cloud Platform (GCP) and SAP Litmos.

In our pursuit to create social readiness and awareness about applied AI, DS.AI continuously o develops learning resources to educate and empower schools, colleges, universities, organizations, and public entities. This guide is part of a series of learning resources, and there will be several such learning modules published to master applied AI on Google Colab. We use several GCP services to develop these learning resources, including storage services, compute services, network services, and other products and services.

Our goal is to go beyond concepts, ideas, visions, and strategies to provide practical problem-solving applied AI skills, knowledge, and expertise to gain hands - on learning experience. To achieve our goals and objectives, we use GCP products and services, including BigQuery, AutoML, AutoML Tables, Dataproc, Dataflow, Data Studio, etc.

Google Cloud colab

DeepSphere.AI
Enterprise AI and IIoT for Analytics

DISCLAIMER

We share this information for learning purposes only, and we've developed this learning material based on our experience, skills, knowledge, and expertise. Our perspective on the tools, technologies, systems, applications, processes, methodologies, and other information used in this learning material may differ from other providers. We advise users to use the learning material at their own risk.

The sample programs in the ensuing learning material were developed based on a few system and data assumptions, and these examples may or may not work for everyone. If there are any issues in following the instructions in the learning material, please feel free to contact our support team, and we will do our best to help you based on the availability of our support services.

The hardware, software, tools, technologies, processes and methodologies, used in the learning material belong to the respective vendors. Users agree to use and implement the learning resources at their own risk, and under any circumstances, DeepSphere.AI is not liable for any of these vendor's products, services, and resources.

DeepSphere.AI
Enterprise AI and IIoT for Analytics

TABLE OF CONTENTS

DeepSphere.AI
Enterprise AI and IIoT for Analytics

TABLE OF CONTENTS

DeepSphere.AI
Enterprise AI and IIoT for Analytics

1.EXECUTIVE SUMMARY

The Google Collab Set –up Lab Guide by DeepSphere.ai is an exclusive, detailed, and learner-friendly module that explains the step-by-step process on how to go about Google Colab. When the data science community was wrestling with the limitations of computing power, owning a GPU and clueless about how to work with large datasets on deep learning and machine learning algorithms, without the 'memory error' throwing up, Google launched Colab and changed it all for them for good!

This online –platform allows anyone to train large datasets and models for free! Allowing you to work with complex models with the ability to share them with co-workers and peers, the Google Colab does not require any set up and can be edited by the team simultaneously just like Google docs.

DeepSphere.AI's Google Collab guide is quick and simple and enables the learner to create, upload, share and store notebooks. Mount the Google Drive, import directories, and run your Python, R, and Scala codes while improving on your coding skills simultaneously. Learn how Colab supports many popular machine learning libraries which can be easily loaded on the notebook. This basic guide takes you through Google Colab's different features and how to use them in projects.

2.LEARNER PROFILE

Learner Pre-requisites

The guide is built to enable students understand the Colab platform which requires them to have a fundamental understanding about a development environment, basic Python coding skills and knowledge of what machine learning and deep learning tools and libraries are all about. This enables them to understand the platform better and perform the different tasks listed in the guide.

Learner Segment

The guide is designed to suit learners from Grade 6 up to professionals. The guide is structured to serve the learning needs of all learners who want to work on the Colab platform. With links to understanding concepts and a serial flow on the features, this is an introduction to the platform that benefits the learner by allowing them to run our sample programs on the Colab notebook to see how the platform works.

3.INTRODUCTION TO GOOGLE COLAB

Colaboratory, or "Colab" for short, is a product from Google Research. It is an incredible online browser-based platform that allows us to train our models on machines for free! It allows us to work with large datasets, build complex models, and even share our work seamlessly with others.
That's the power of Google Colab!

Colab is a Python development environment that runs in the browser using Google Cloud and it is especially well suited for machine learning, data analysis and education.

Most importantly, Google Colab does not require any setup. If you have used Jupyter notebook previously, you would quickly learn to use Google Colab. It is Google's free cloud service for AI developers. With Colab, you can develop Deep Learning Applications on the Graphic Processing Unit (GPU) for free.

3.1 GOOGLE COLAB VS COLAB PRO

Recently, Google released Colab Pro, priced at 9.99$ per month. It has faster GPUs and TPUs, longer running times which means a longer running notebook, and more memory. This simply means a higher RAM and more disk which means more room for your data.

Find below are the comparison between the free version of Colab and Colab Pro.

	Price	GPU	Runtime	Memory
Colab	Free	K80	Up to 12 hours	12GB
Colab Pro	$9.99/m (before tax)	T4 & P100	Up to 24 hours	25GB with high memory VMs

Currently, Colab Pro is available only in US and Canada. However, by downloading "Surfshark VPN Extension" (A lightweight VPN proxy extension to secure your digital life and data) and switching to "US" as the region, we can upgrade to Colab Pro.

4. OVERVIEW OF GOOGLE COLAB FEATURES

The Google Colab can simply overwhelm the programmer with its nimbleness and features. It supplements and empowers the programmer with a variety of functions. Find them listed as below:

- Write and execute code in Python.
- Document your code that supports mathematical equations.
- Create/Upload/Share notebooks.
- Import/Save notebooks from/to Google Drive.
- Import/Publish notebooks from GitHub.
- Import external datasets e.g. from Kaggle.
- Integrate machine learning libraries like PyTorch, TensorFlow, Keras, OpenCV
- Free Cloud service with free GPUs and TPUs.

5. ACCESSING GOOGLE COLAB NOTEBOOK

UNIT TOPICS

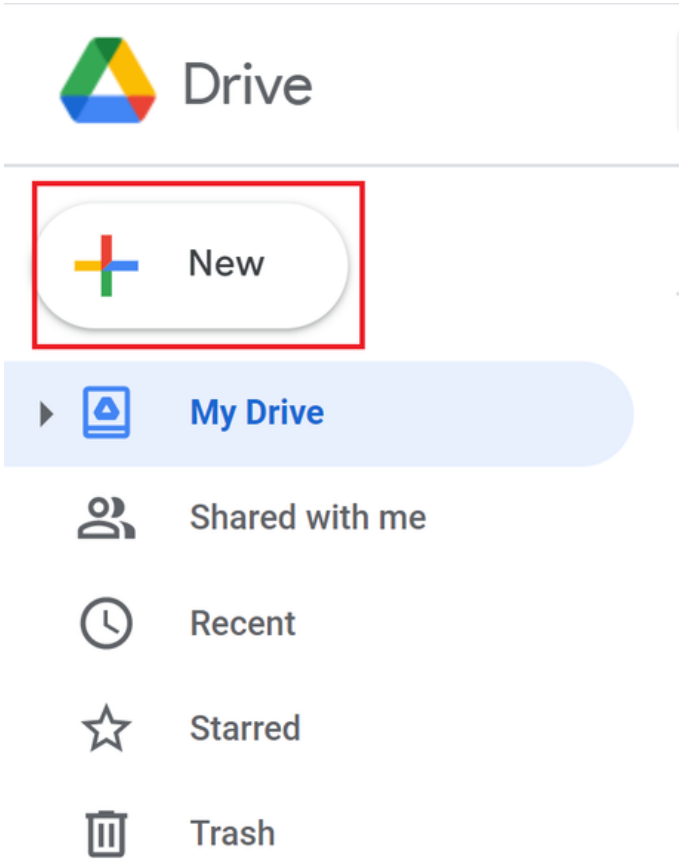
- 5.1 How to access the Google Colab Notebook using Google Drive
- 5.2 How to access a new Colab Notebook using the Collab Interface
- 5.3 How to access an existing Colab Notebook using Google Drive
- 5.4 How to access an existing Colab Notebook using the Colab Interface

DeepSphere.AI
Enterprise AI and IIoT for Analytics

5.1 HOW TO ACCESS THE GOOGLE COLAB NOTEBOOK IN GOOGLE DRIVE

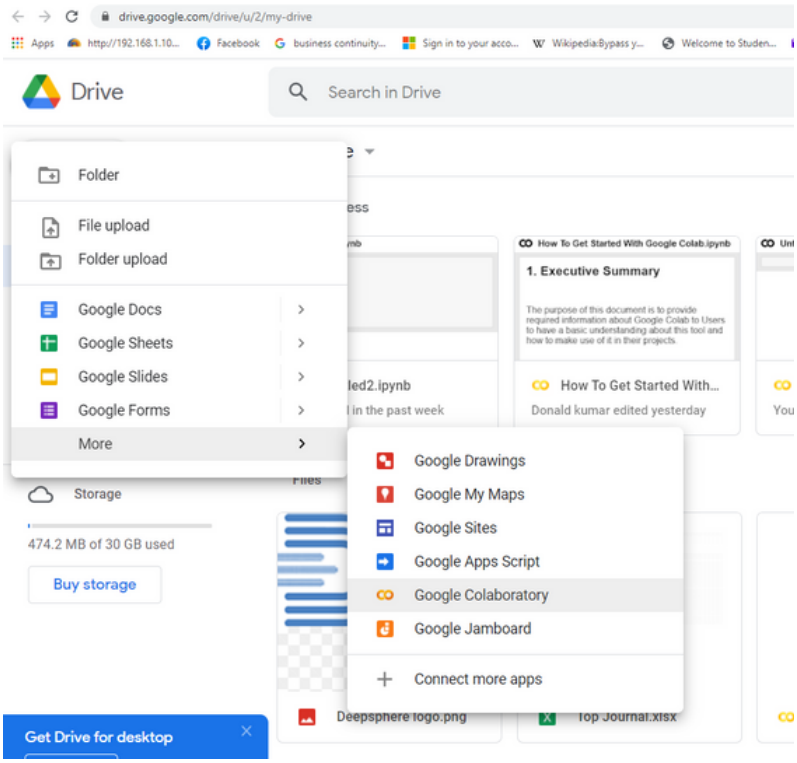
Open Google Drive using the below URL and create a new file as shown below:

https://drive.google.com/drive/my-drive



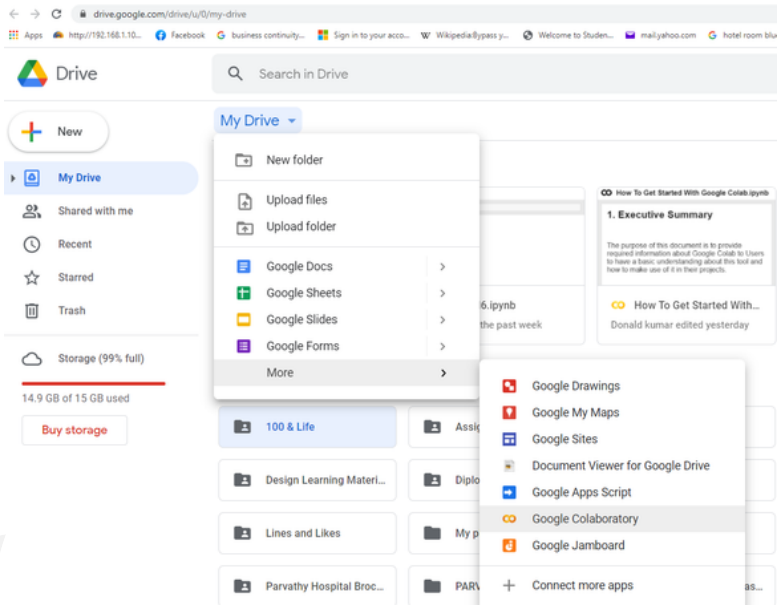
New >More > Google Colaboratory

The below images clearly explain how to get started by getting Google Colab on your Gdrive.

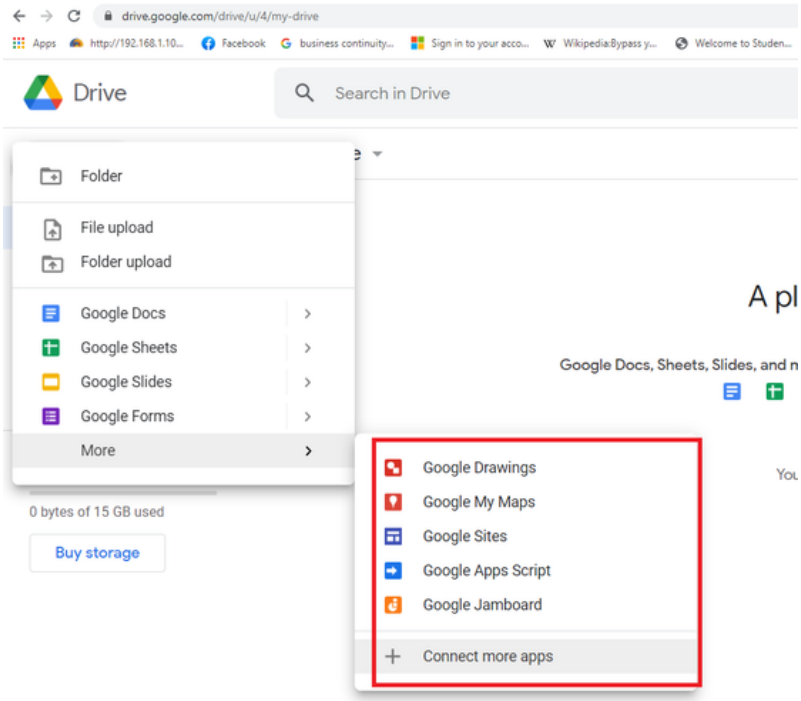


OR

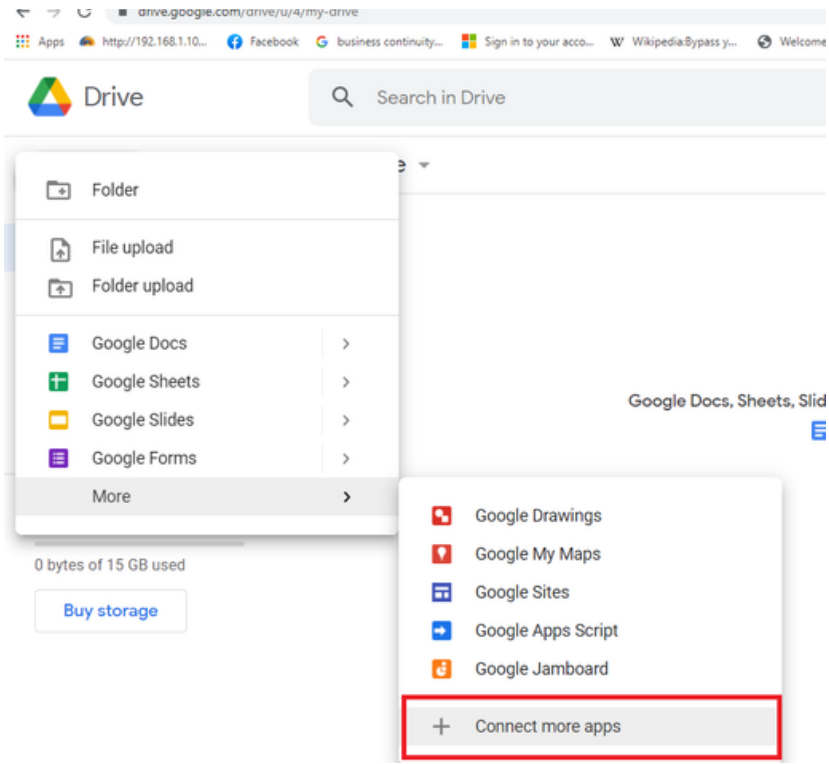
Right-click in a folder and select More > Google Colaboratory from the context menu



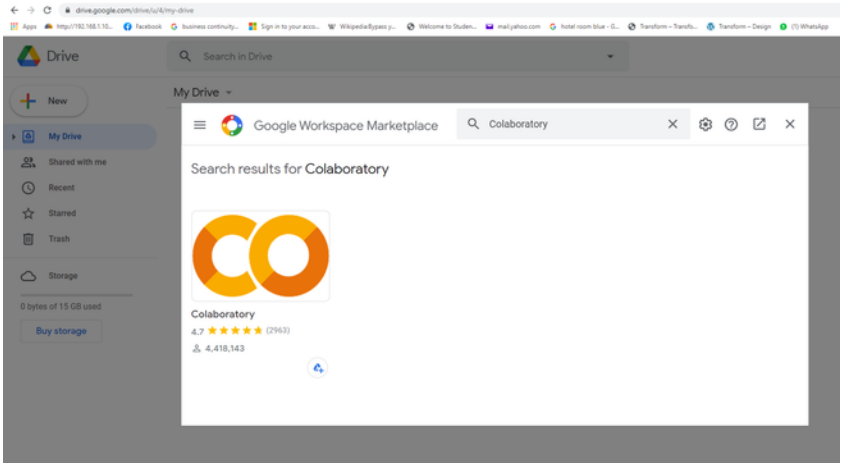
If the 'Google Colaboratory' option is missing from the context menu as shown below, you need to install/connect Google Colaboratory into Google Drive



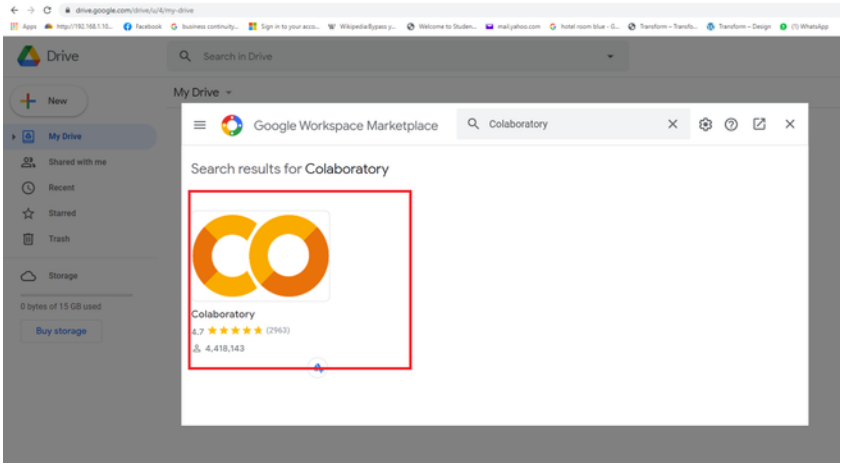
To install/connect Google Colaboratory, please follow the below steps:
New > More > Connect more apps



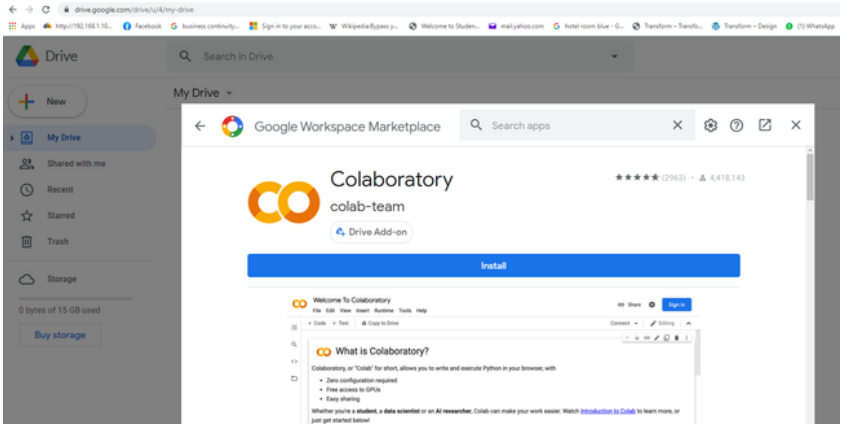
Search for ‘Colaboratory’ in the search field



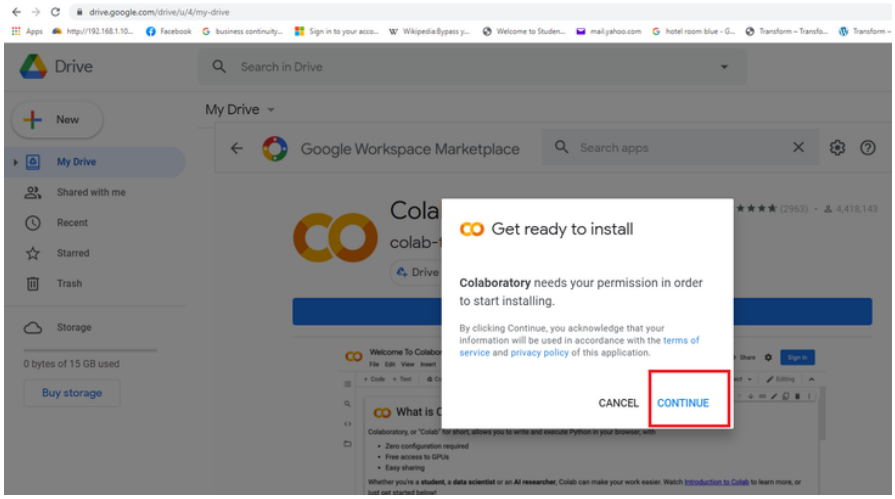
Click on Colaboratory



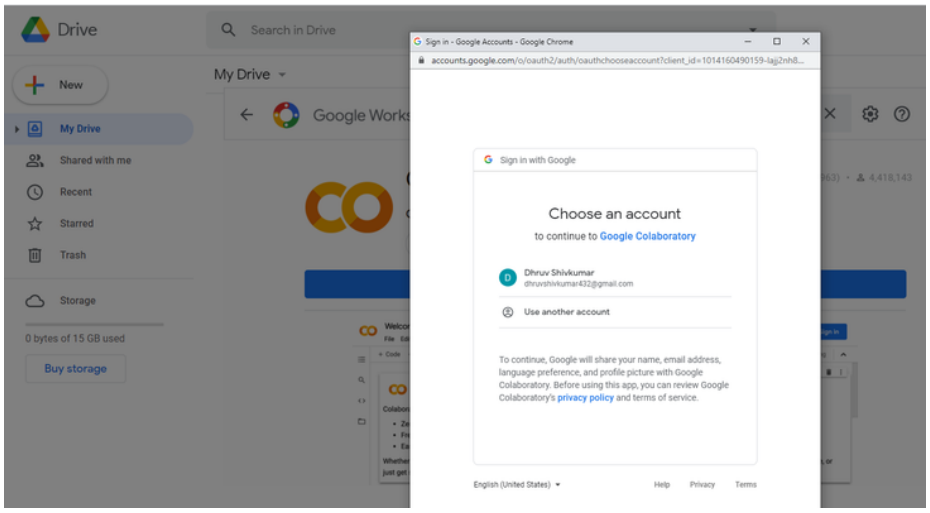
Click on Install



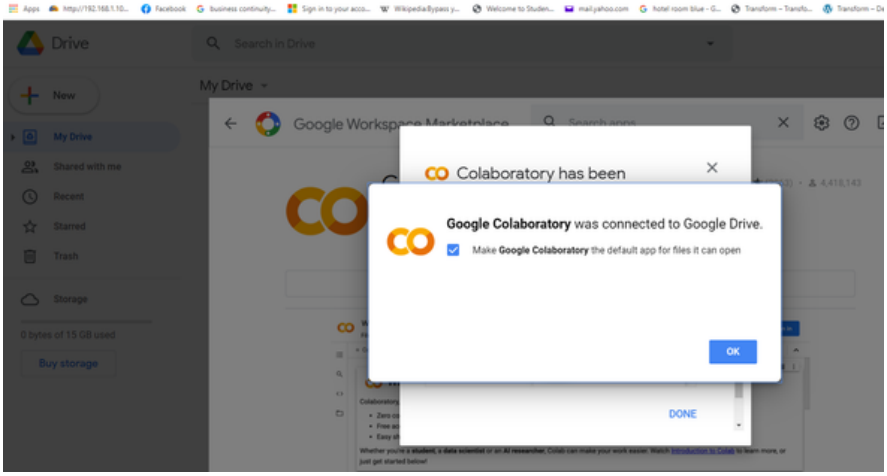
Click on Continue

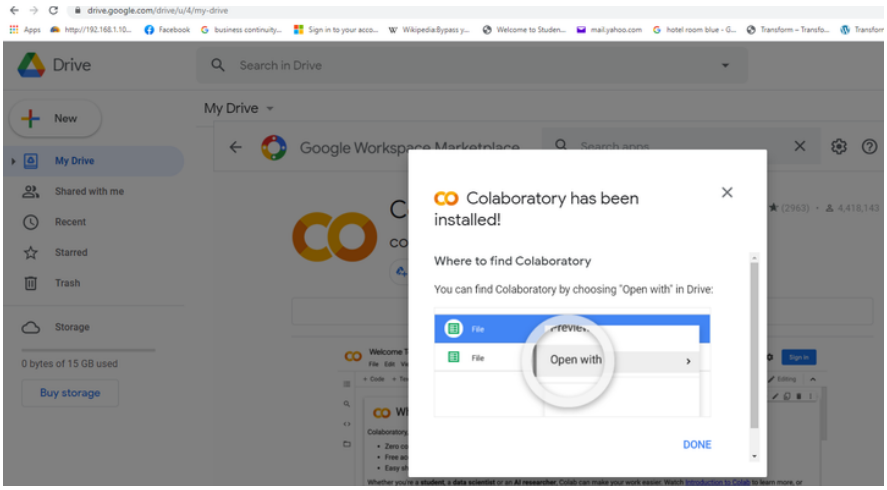


In case you are logged into multiple Google accounts, choose the google account which you want to use. This is done just the same way you switch to use other apps on Google.

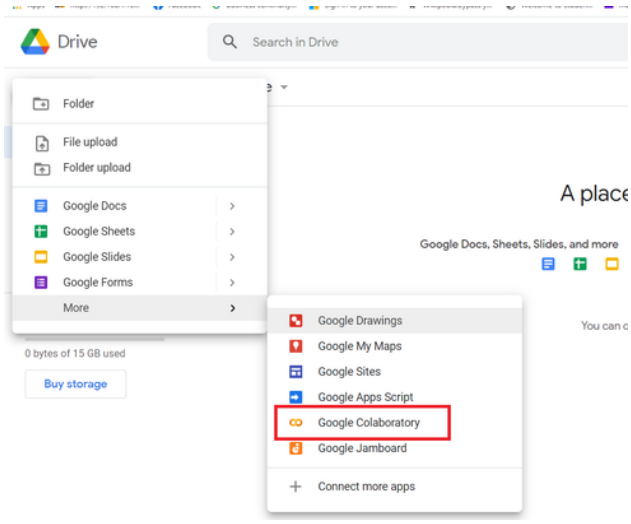


Click OK and then Done





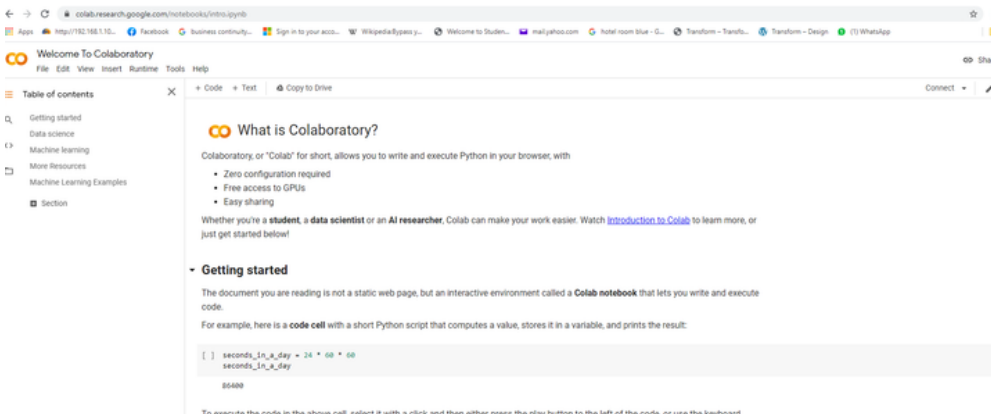
Now, you'll be able to see the Google Colaboratory option as shown below



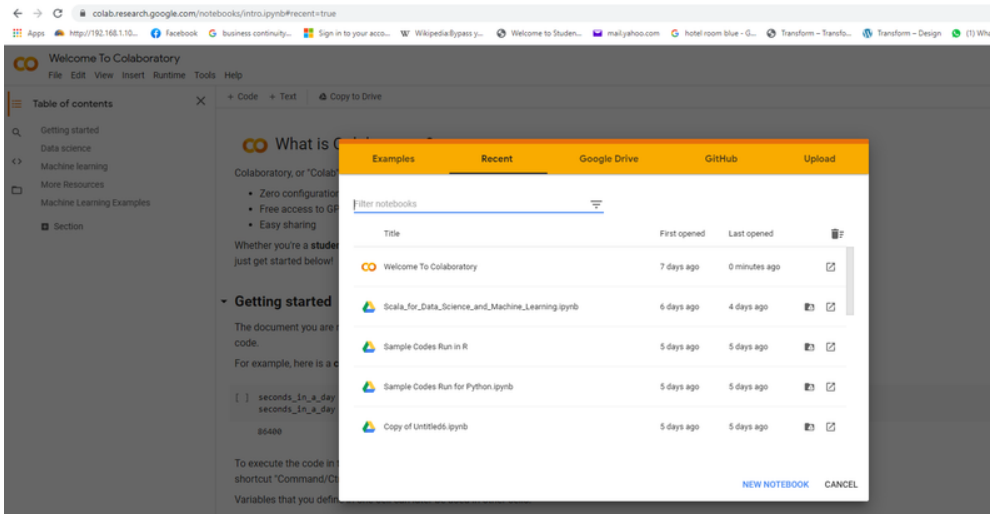
5.2 HOW TO ACCESS COLAB NOTEBOOK USING COLAB INTERFACE

The Collab Interface looks like the below image
Click on the link below to see

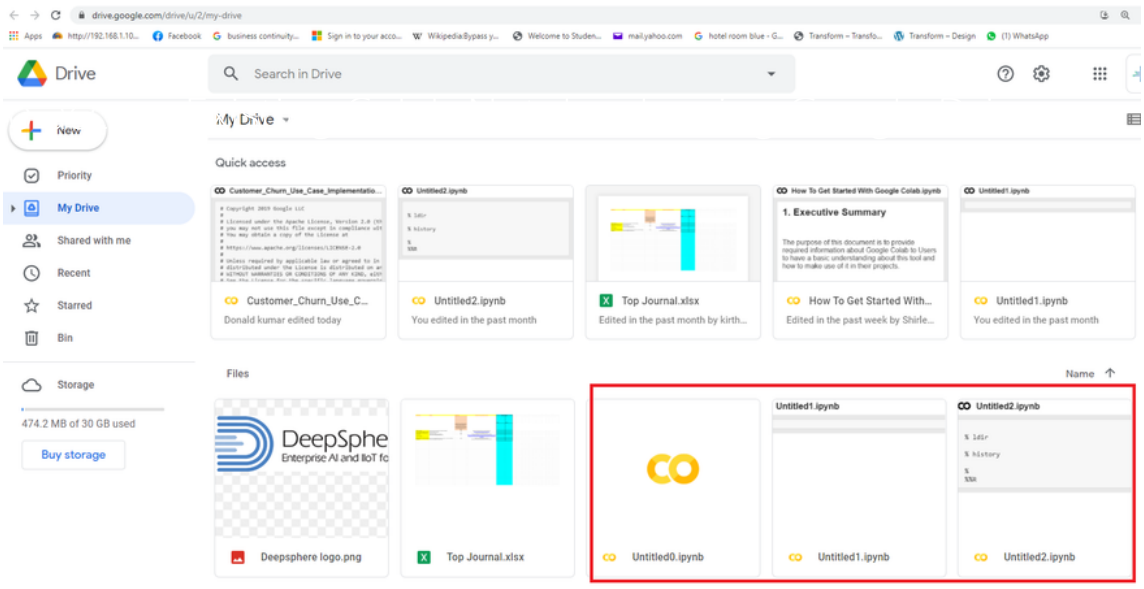
https://colab.research.google.com/



Visit the following Colab site and click on 'New notebook' as shown below.



5.3 HOW TO ACCESS AN EXISTING COLAB NOTEBOOK USING GOOGLE DRIVE



Find Notebooks created in Google Drive to exist in the folder they were created or moved to.

Notebooks created from the Colab interface will be automatically moved to a folder called 'Colab Notebooks' which is automatically added to the 'My Drive' folder of your Google Drive when you start working with Colab.

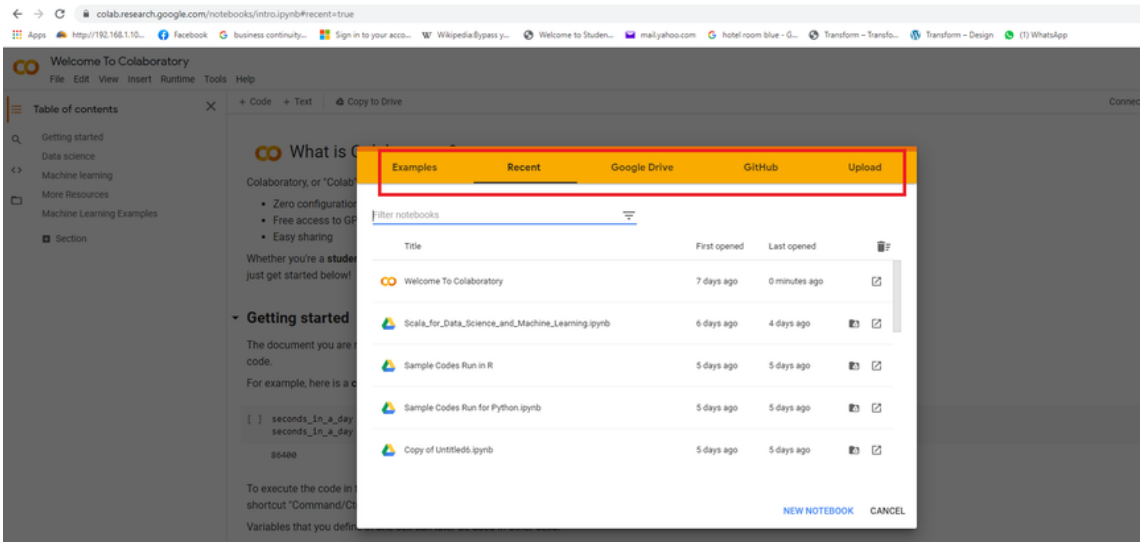
Colab files can be identified by a yellow symbol and '.ipynb' file extension.

Open files by either double-clicking on them or selecting Open with > Colaboratory from the button found at the top of the resulting page or by right-clicking on a file and selecting Open with > Colaboratory from the file's context menu.

5.4 HOW TO ACCESS AN EXISTING COLAB NOTEBOOK USING COLAB INTERFACE

Opening notebooks from the Colab interface allows you to access existing files from Google Drive, GitHub, and local hardware.

Visiting the Colab interface after initial use will result in a file explorer window/ modal appearing. From the tabs at the top of the file explorer, select a source and navigate/ go to the .ipynb file you wish to open.



The file explorer can also be accessed from the Colab interface by selecting File > Open notebook

or by pressing the Ctrl+O keyboard combination.

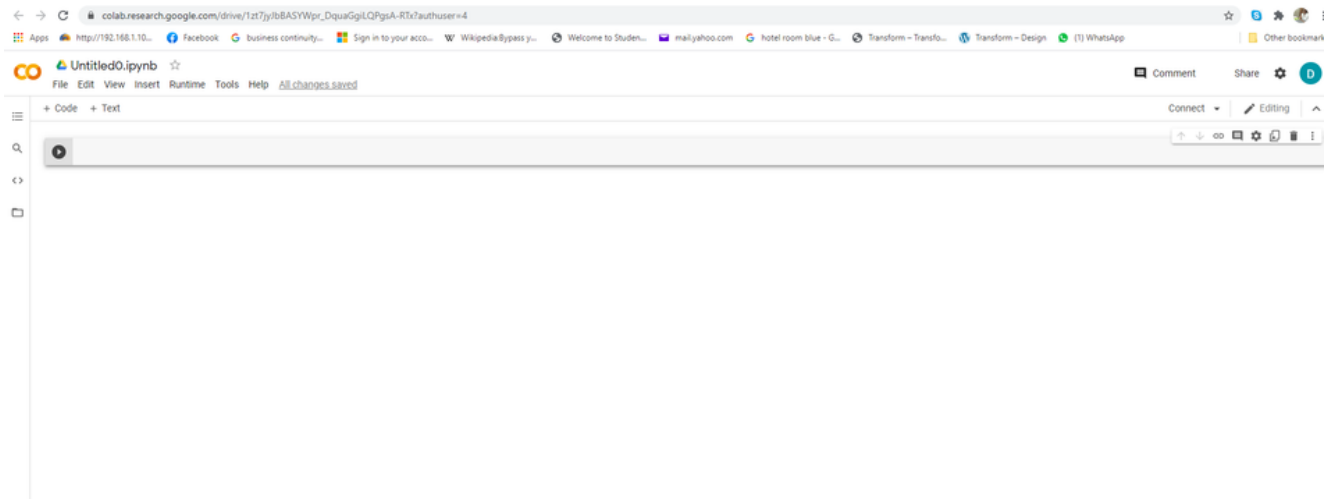
6. HOW TO EXECUTE CODE

What is the Colab Notebook?

The Colab notebook is your playground to **code, execute and share.**

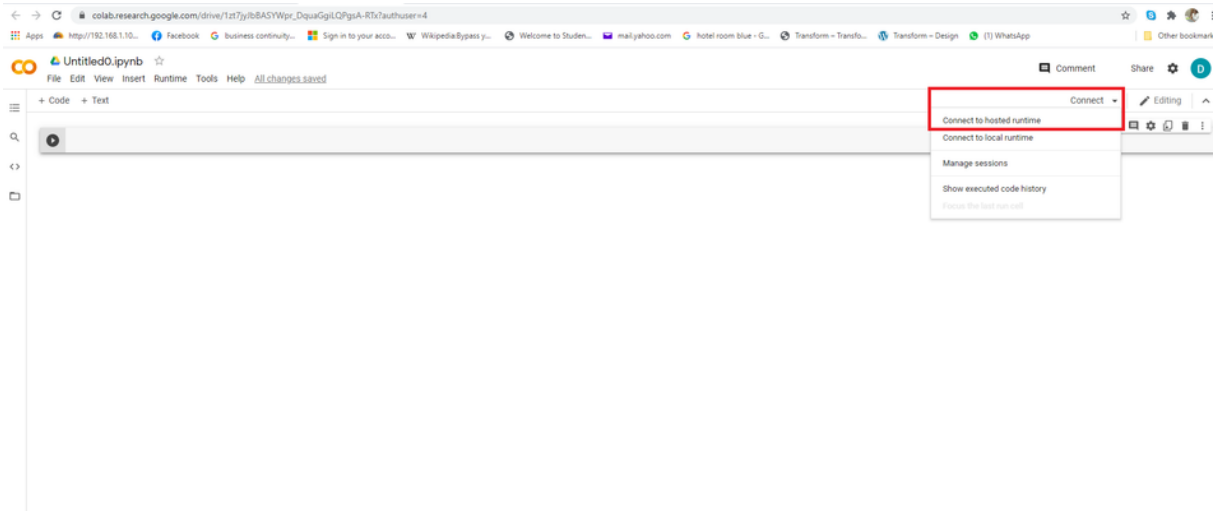
A notebook is a list of cells. Cells contain either explanatory text or executable code and its output. Click a cell to select it.

A blank Google Colab notebook looks like the below image:

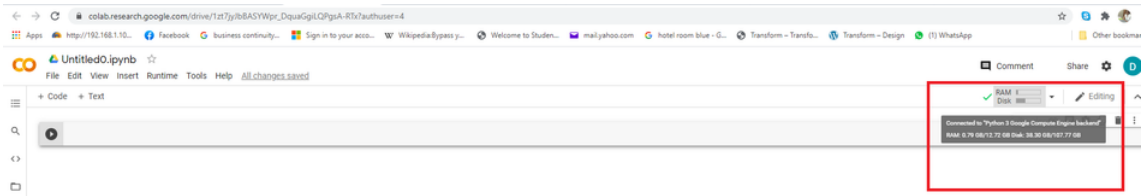


By default, a new notebook is created with a single code cell, as shown in the above figure. If you want to create a new code cell, then click on the Code button. For text cells, you can click on the Text button. You can also insert such cells using the Insert menu.

Before we run the code, we need to ensure that the notebook is connected to a runtime. If the notebook is not connected, click on 'Connect' or click on the dropdown near 'Connect' option and select 'Connect to hosted runtime' as seen in the below screen shot

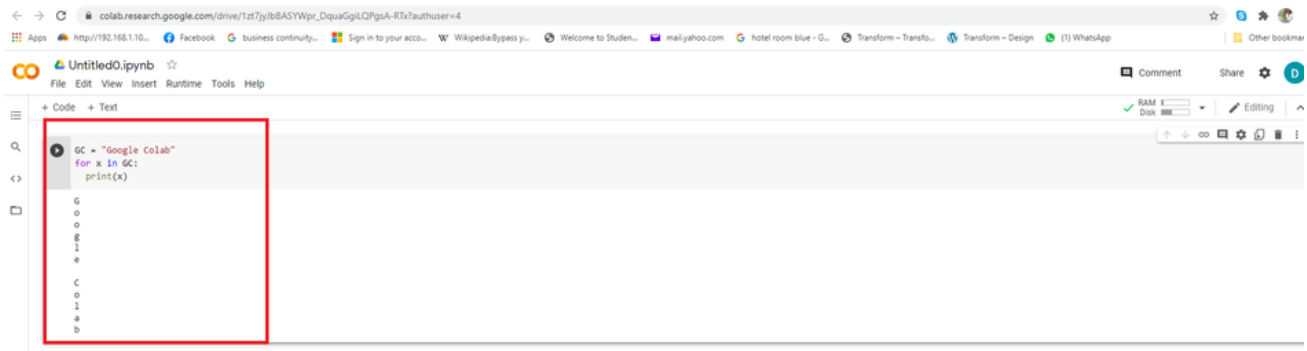


Once the toolbar button indicates CONNECTED as shown below, click on the cell to select it and execute the content.



Add the code in the code cell and then click on the play button on the left side to see results.

Sample screen with code and results as shown below:



The codes in the cell can also be executed in the below mentioned ways:

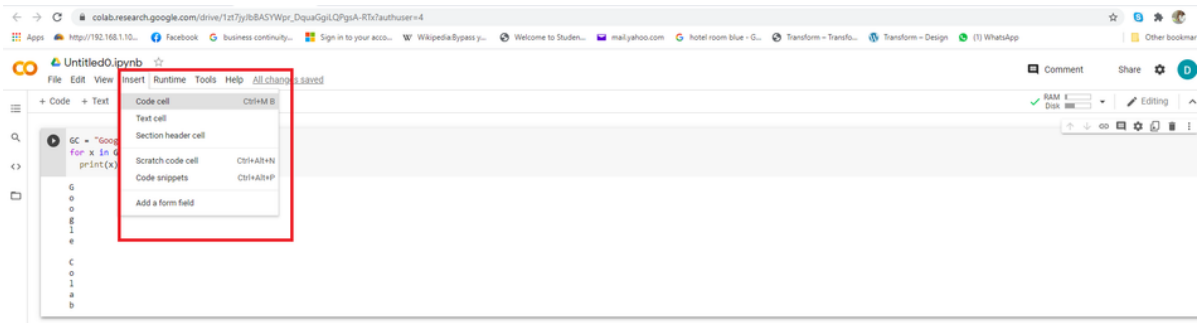
- Click the **Play** icon in the left gutter of the cell;
- Type **Cmd/Ctrl+Enter** to run the cell in place;
- Type **Shift+Enter** to run the cell and move focus to the next cell (adding one if none exists); or
- Type **Alt+Enter** to run the cell and insert a new code cell immediately below it.

·There are additional options for running some or all cells in the Runtime menu.

7. ADDING NEW CODE CELLS

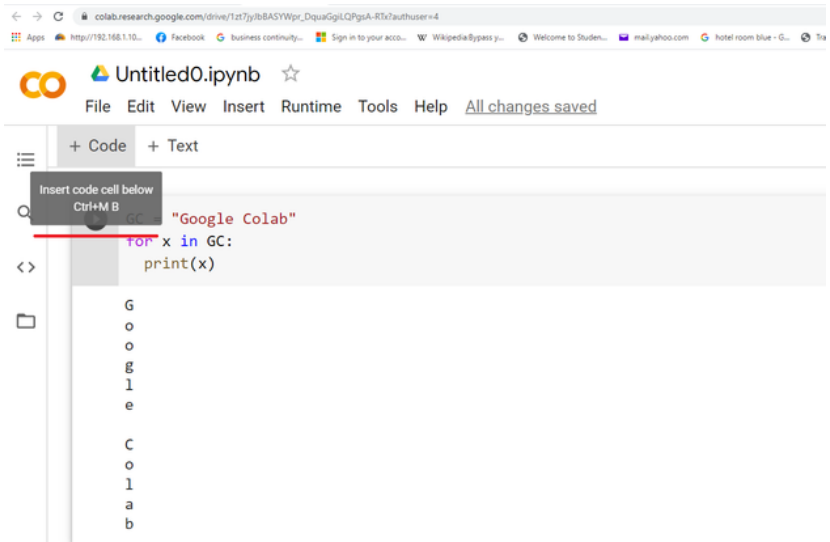
Just like an excel sheet, the Colab notebook can be filled with unlimited code and text cells.

To add more code to your notebook, select the following menu options – Insert / Code Cell



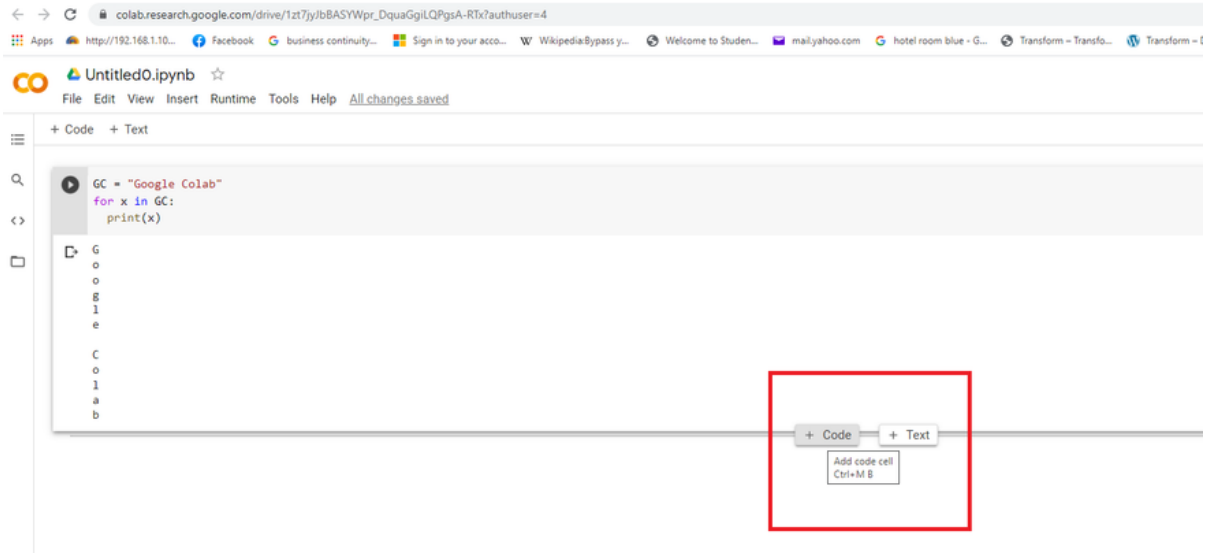
OR

Click on 'Code' as shown below



Alternatively, just hover the mouse down the center of the Code cell. When the CODE and TEXT buttons appear, click on the CODE to add a new cell.

As seen in the screen below:



DeepSphere.AI
Enterprise AI and IIoT for Analytics

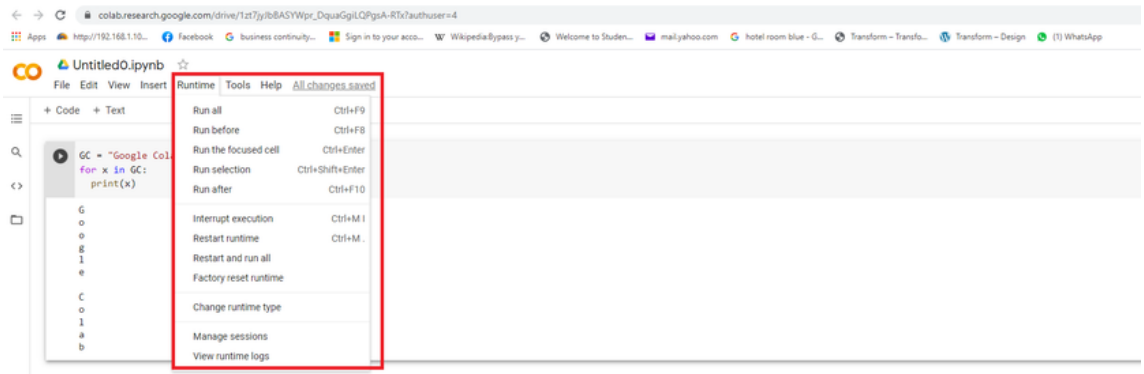
8. EXECUTE ALL FUNCTION

This feature enables you to run and execute your codes on the notebook. You can type or copy-paste your codes in the code cell and run them. See the results displayed below the code cell.

To run all of the codes in your notebook without interruption, run the following menu options:

Runtime / Restart runtime
or
Runtime / Restart and run all
Followed by Runtime / Run all

As seen in the below image with the Runtime menu expanded



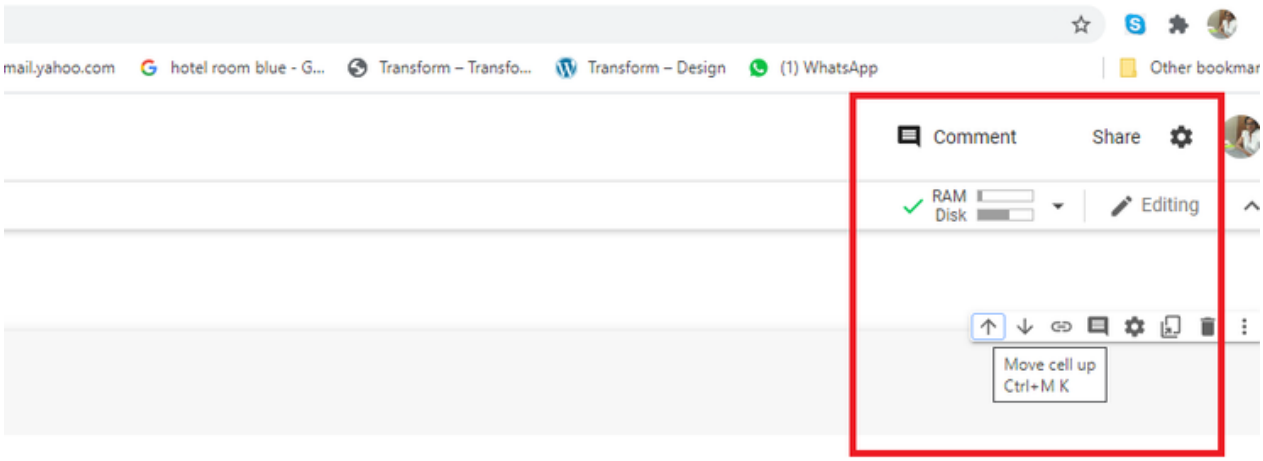
Study the different menu options under Runtime menu to familiarize yourself with the different options available to run the notebook.

9. CHANGING THE ORDER OF CELLS

When your notebook contains a large number of code cells, you may come across situations where you want to change the order of execution of these cells.

To do so, select the cell you want to move and click on UP CELL or DOWN CELL buttons. You can click buttons multiple times to move the cell to more than one position.

Please see reference Image below:

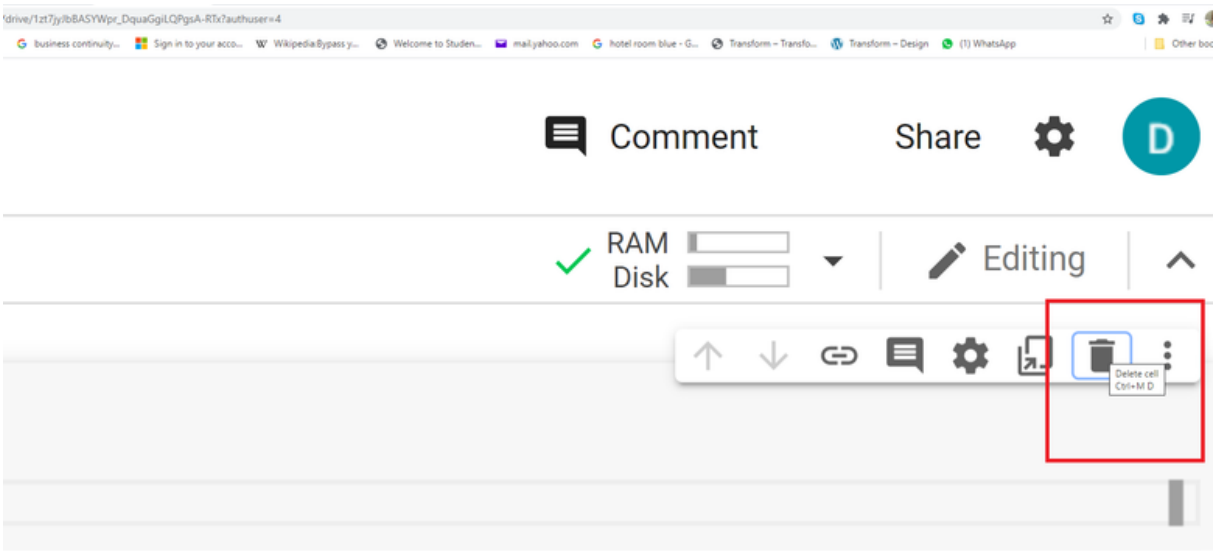


10. DELETING A CELL

During the development of your project, you may have introduced some unwanted cells in your notebook just to try out codes that you may now find redundant.

Easily remove these cells from your project with just one click.

Click the trash icon in the upper right corner of your code cell and the current cell will be deleted



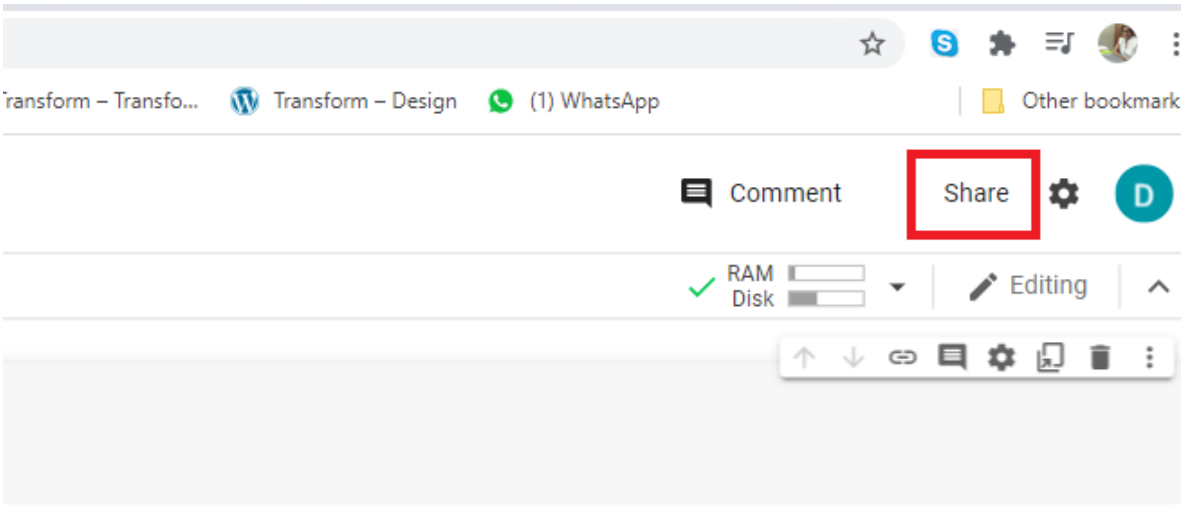
DeepSphere.AI
Enterprise AI and IIoT for Analytics

11. GOOGLE COLAB - SHARING NOTEPAD

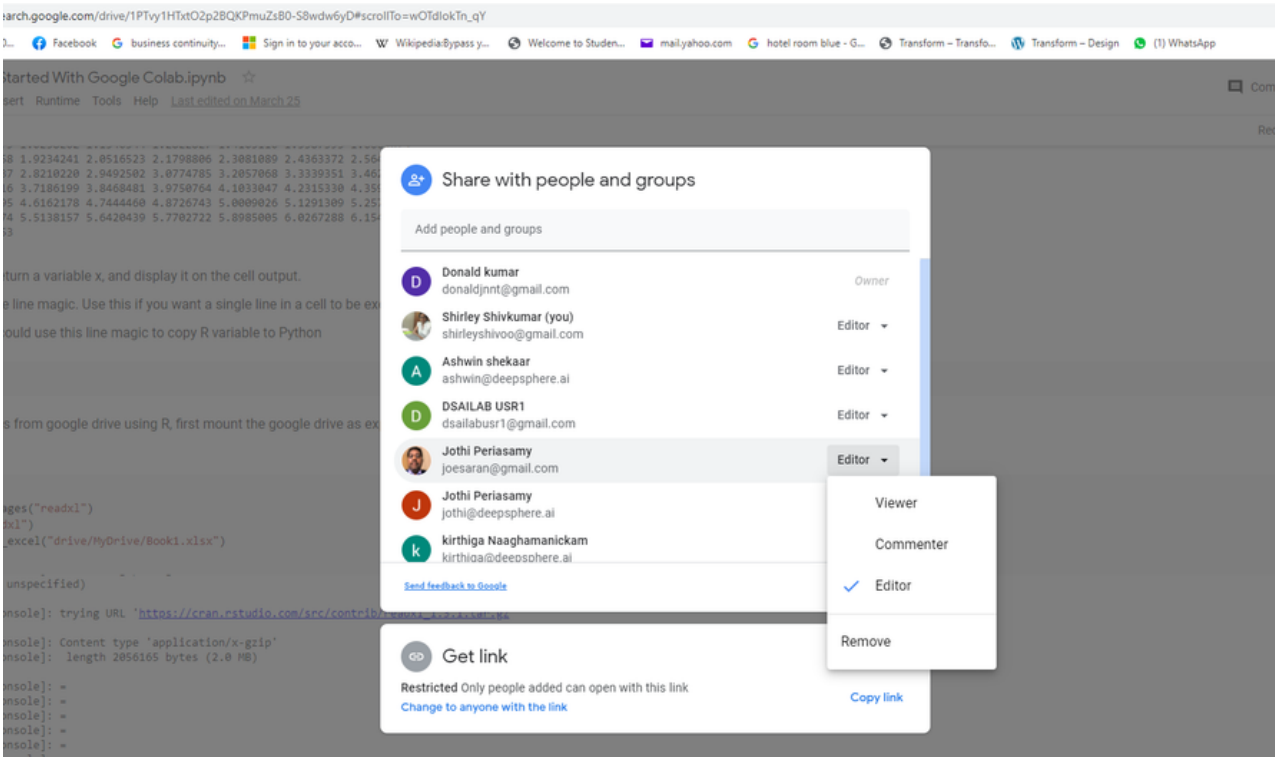
Truly, the Colab is all about collaboration and it allows you to work in a team with ease. Share your notebook with other team members to fasten up the pace of development. To share the notebook you created with other co-developers, the Colab notebook allows you to make a copy of it in Gdrive. to enable easy sharing.

You can also publish the notebook to the general public, by sharing it from your Github repository.

Another way to share your notebook is to click on the SHARE link in the upper right corner of your Colab notebook.



You can enter the email IDs of people you want to share with the current document. You can also can set the type of access by selecting one of the three options. Reference Image below:



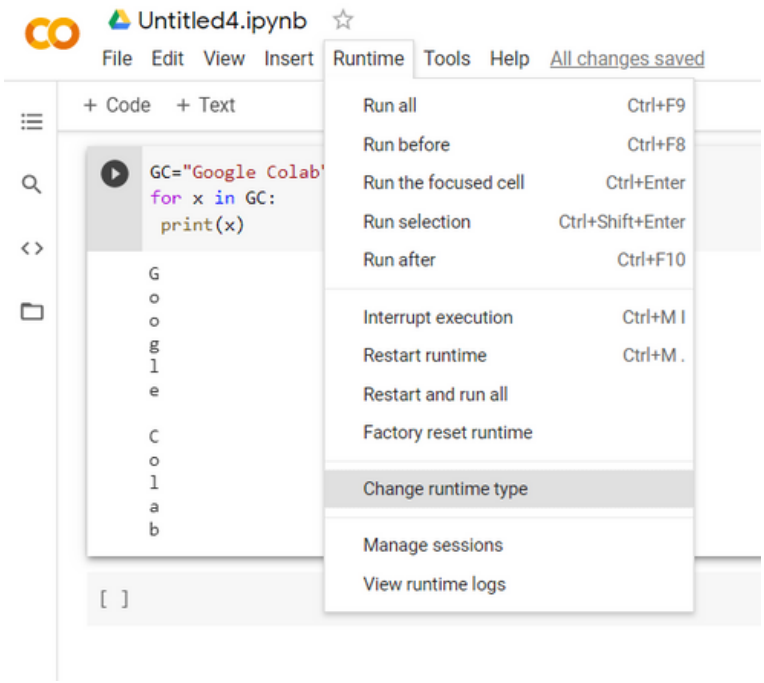
12. ENABLING GPU (GRAPHIC PROCESSING UNIT)

The graphics processing unit (GPU) is a processor that is specially designed to render intensive processing tasks. It is the creative side of the computer helping render graphical user interfaces into visually attractive icons and designs rather than reams of black and white lines.

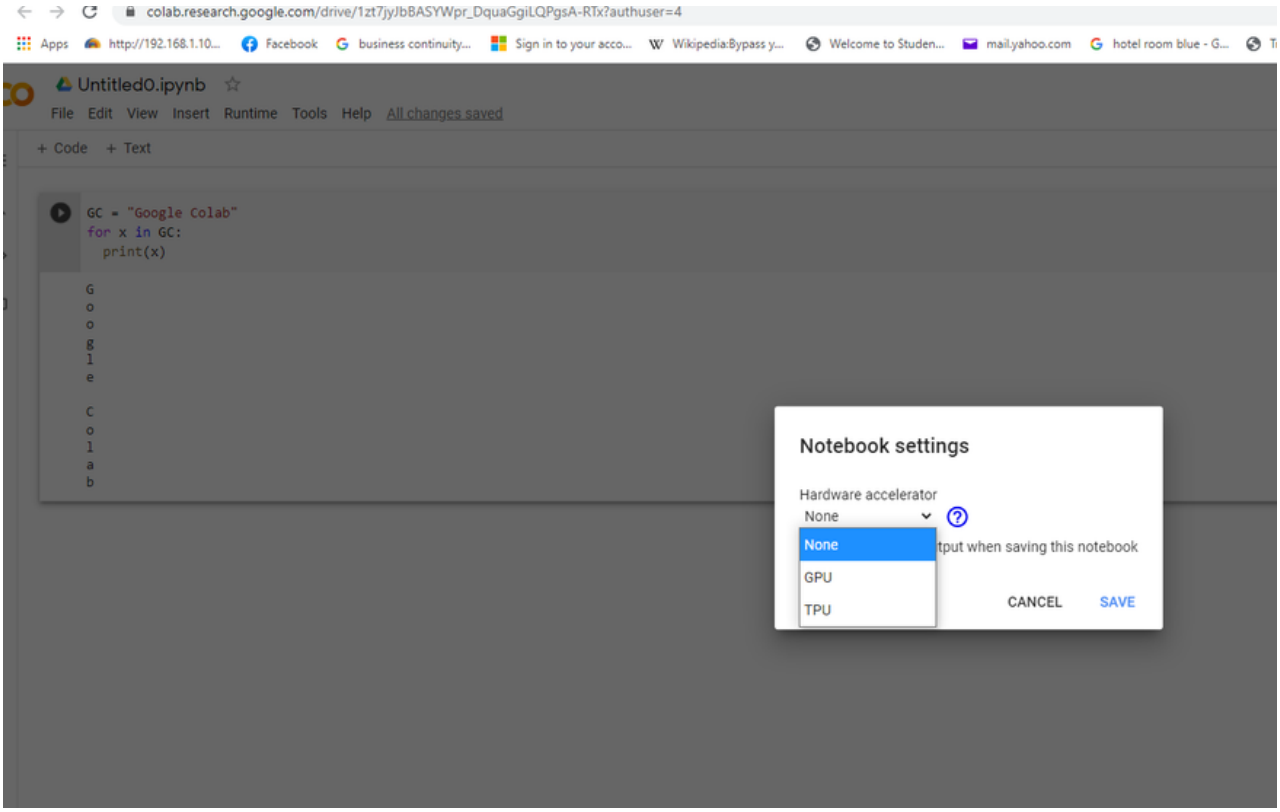
And as we all know the GPU has enormous business applications too, apart from its creative properties and Google Colab supports you with a free GPU!

·To enable GPU in your notebook, select the following menu options:

Runtime / Change runtime type



You will see the following screen as the output



Select the GPU and your notebook will start using the free GPU provided in the cloud while processing.

12. TESTING FOR GPU

·You can easily check if the GPU is enabled by executing the following code :

import tensorflow as tf

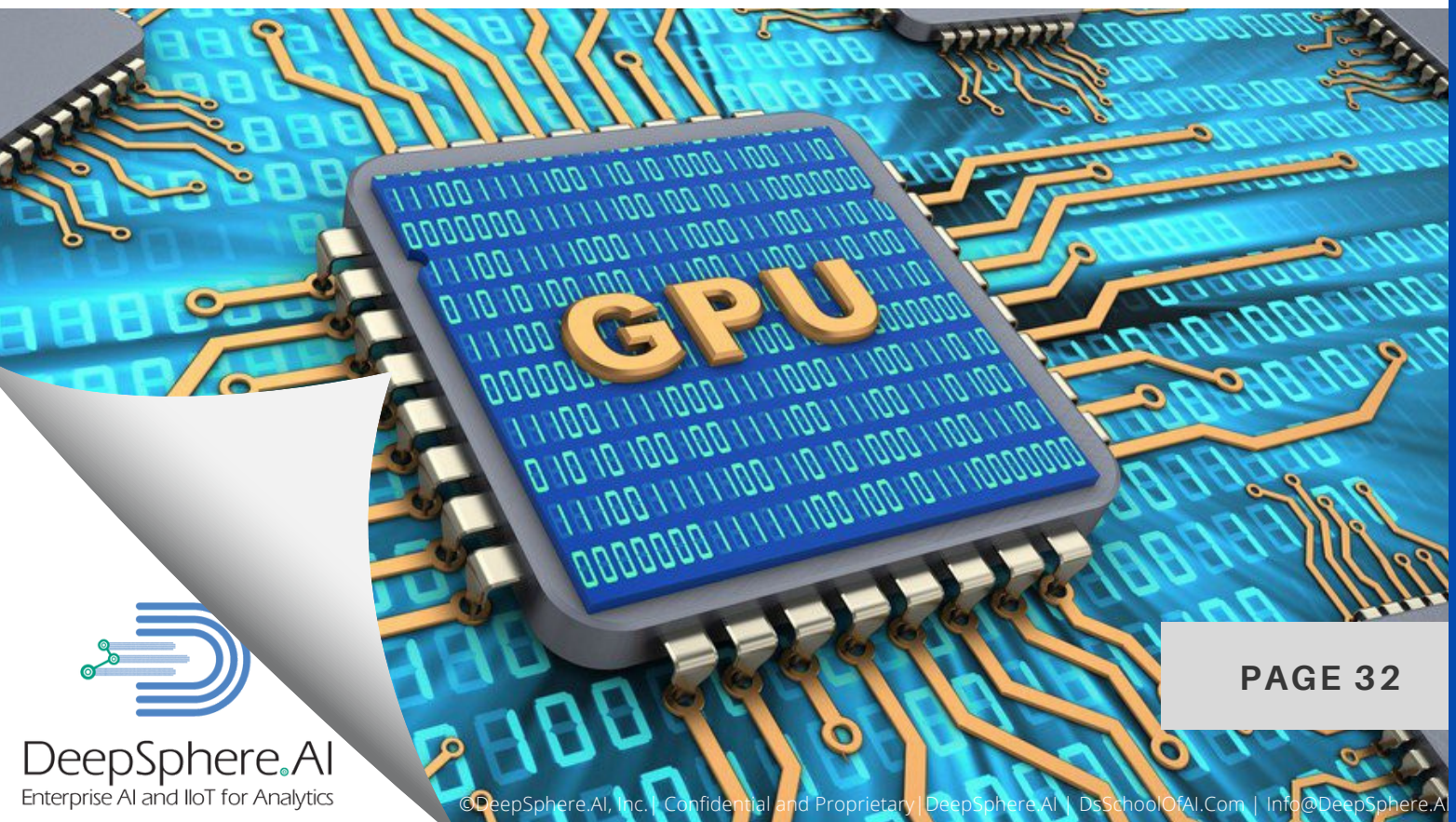
tf.test.gpu_device_name()

·If the GPU is enabled, it will give the following output

'/device:GPU:0'

```
[1]  import tensorflow as tf
     tf.test.gpu_device_name()

     '/device:GPU:0'
```



14. LIST OF DEVICES

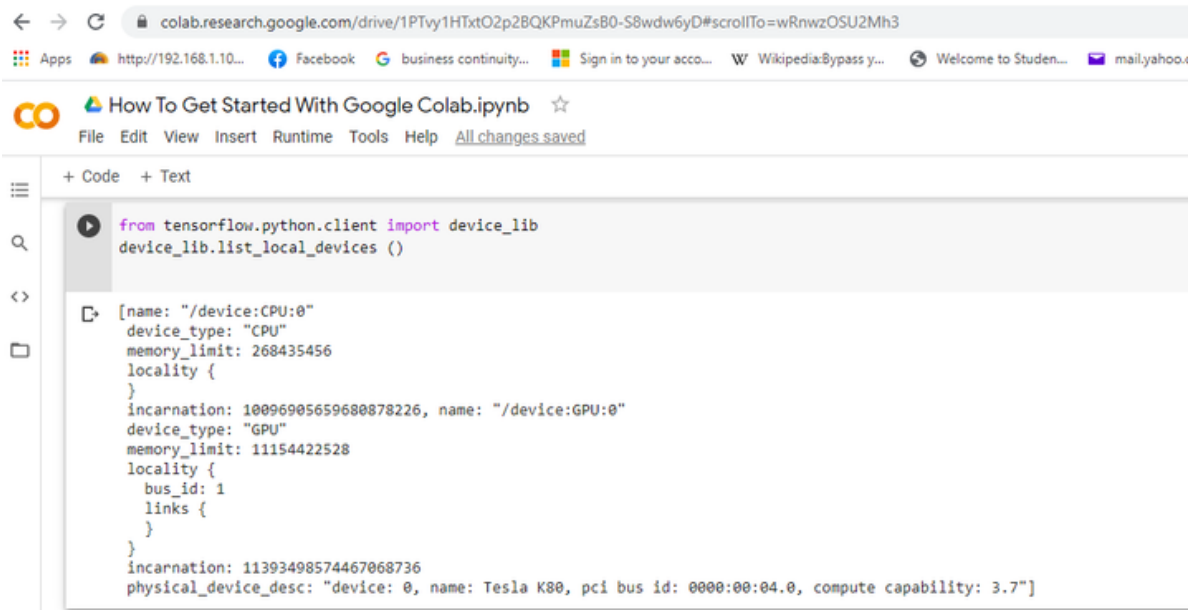
The Google Colab is not just a great collaboration platform but one that keeps the programmer interested, curious and constantly learning about its workings. Now that is the power of transparency!

It brings to the platform the power of collective thinking allowing programmers to talk, debate and share knowledge about its workings. Hence, the platform allows you to see the devices that are being used to process your codes.

If you are curious about the devices used when running your notebook in the cloud, try the following code

```
from tensorflow.python.client import device_lib
device_lib.list_local_devices ()
```

You will see the following output

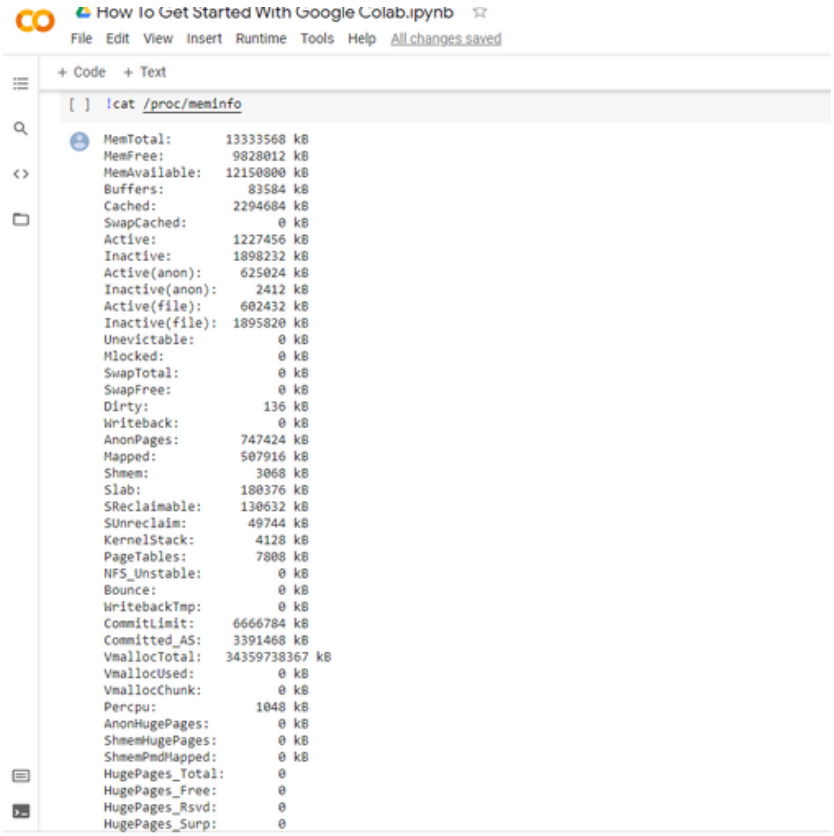


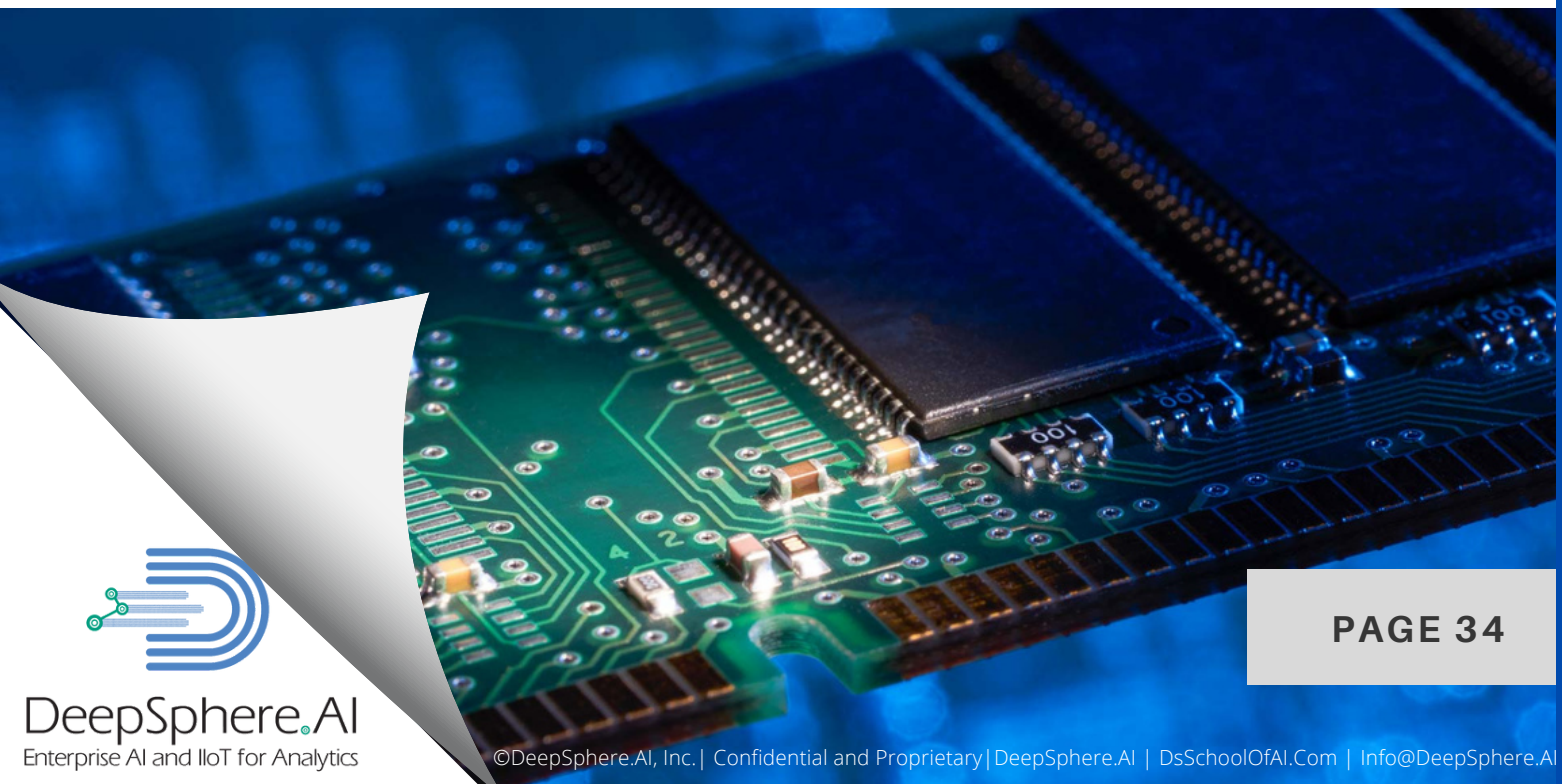
15. CHECK RAM

Just like so many other features that are free on the Google Colab, it also offers the programmer free memory. To see the memory of the resources available for your process, type the following command:

!cat /proc/meminfo

·You will see the following output :





16. GOOGLE COLAB MAGIC

UNIT TOPICS

- 16.1 Line Magic
- 16.2 Cell Magic
- 16.3 Magics List

16. GOOGLE COLAB MAGIC

Magic is a set of system commands that provide an extended mini command language. Magic commands are specific to and provided by the iPython kernel whether magics are available on a kernel developer or a per kernel basis. Magic commands are intended to solve common problems in data analysis using python.

There are two types of Magic

- Line Magic
- Cell Magic

Line Magic as the name suggests,consists of a single command line, while cell magic covers the entire body of the code cell.

In the case of line magic, the command is preceded by a single% character, and in the case of cell magic, it is preceded by two% (%%) characters.

16.1 LINE MAGIC

·Type the following code into your code cell -

% ldir

You will see the contents of your local directory, something like this

drwxr-xr-x 3 root 4096 Jun 20 10:05 drive / drwxr-xr-x 1 root 4096 May 31 16:17 sample_data /

·Try the following command :

% history

This shows the complete history of the commands you have previously executed.

DeepSphere.AI
Enterprise AI and IIoT for Analytics

16.2 CELL MAGIC

Try the following codes to run R codes using Cell Magic in your cell.

NOTE: Mount Google Drive into your Colab Notebook as shown in this topic - 17.2

```
%load_ext rpy2.ipython
```

```
[ ] %%R
    install.packages("readxl")
    library("readxl")
    read_excel("drive/MyDrive/Book1.xlsx")
```

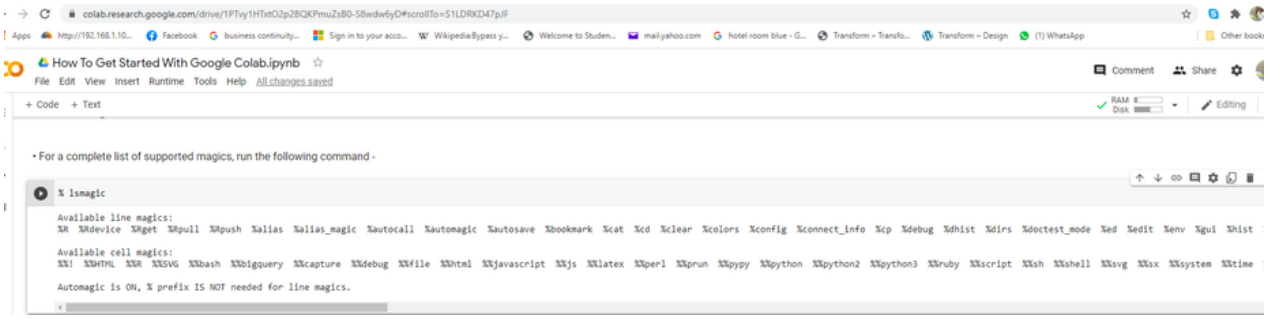
```
[ ] %%R
    data <- read_excel("drive/MyDrive/Book1.xlsx")
    data
```

16.3 MAGICS LIST

For a complete list of supported magics, run the following command -

% lsmagic

·You will see the following output -



17. HOW TO ACCESS EXTERNAL FILES

The Google Colab is dynamic allowing you to access external files by importing them on to Gdrive. This simple and easy way of accessing files from your local system, makes it an incredibly useful platform. Besides, the platform reads almost any type of file ranging from .doc, .xls, .jpg, .png, .pdf, and may others.

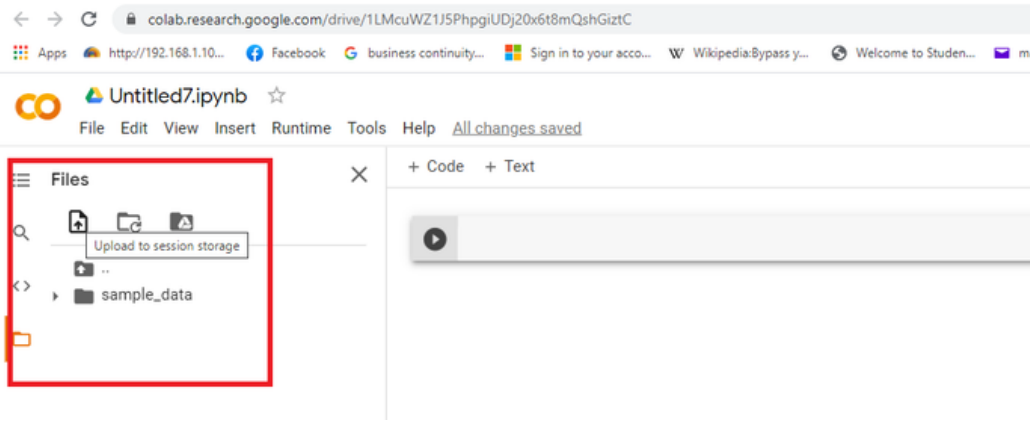
17.1. ACCESS FILES FROM YOUR LOCAL SYSTEM

To import files from your local system, please follow the below steps:

Step 1: Open the File from your local system Click on Files icon:

Step 2: Upload the File to the Session Storage

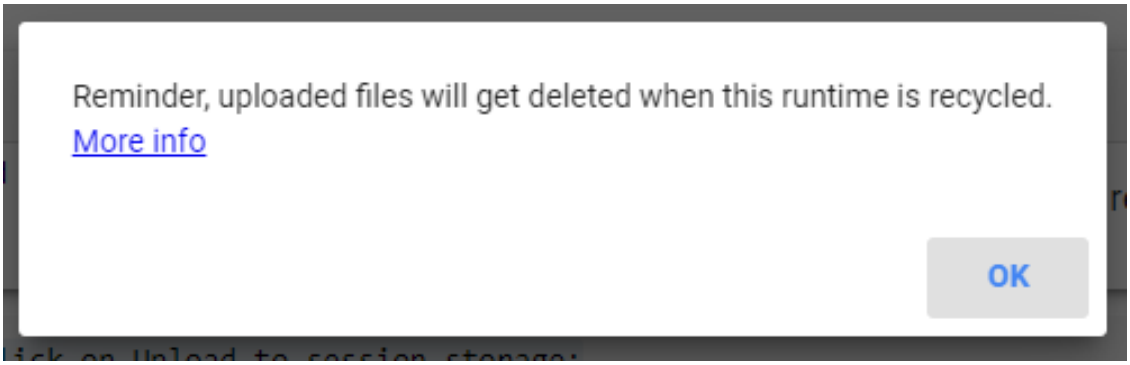
Click on 'Upload to session storage':



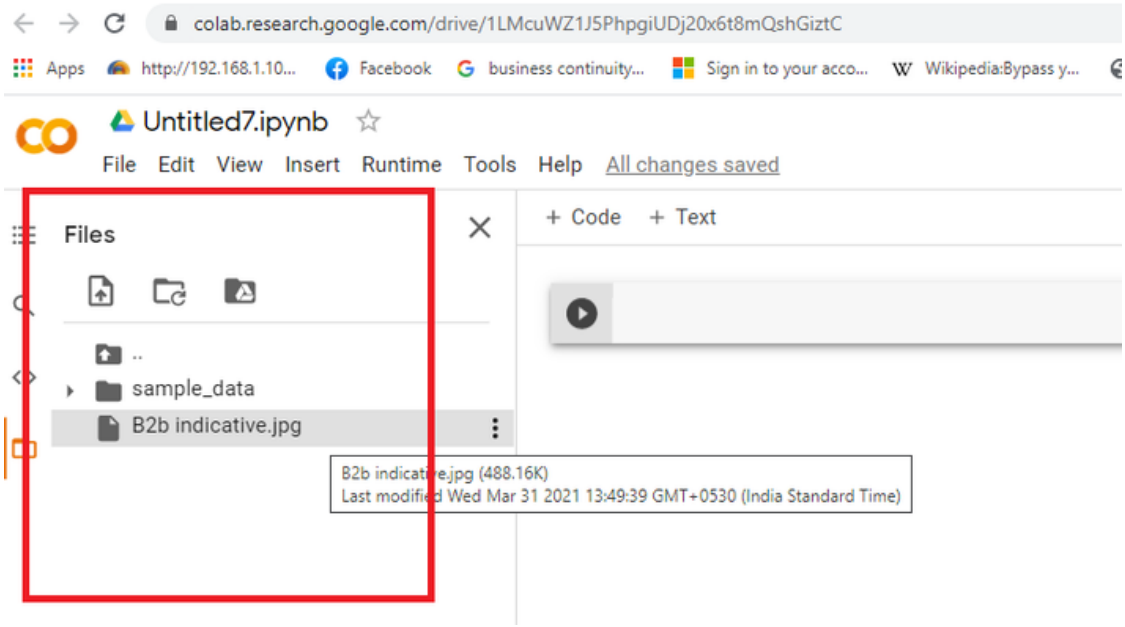
Step 3: Select the File you want to upload

Select the file you would like to upload.

You'll get this pop up to notify that the files saved in the session storage will be deleted after the session is closed as it is just a temporary storage.



As shown in the screenshot below, the imported file has been saved in the session storage



Step 4: Read the Imported File

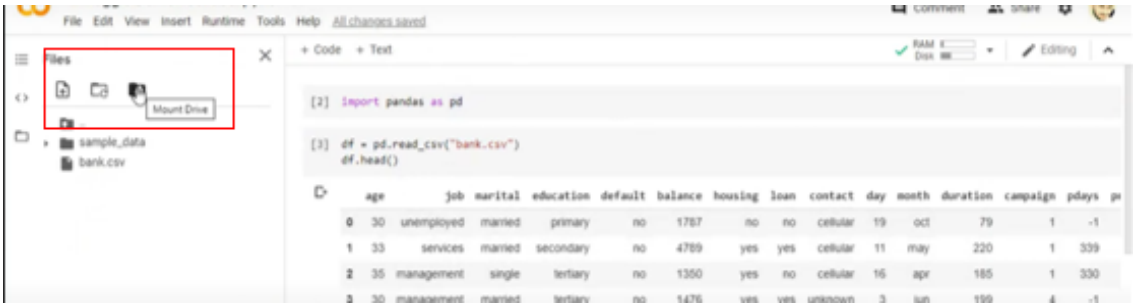
Note: The files saved in the session storage will be deleted after the session is closed as it is just a temporary storage.

The files need to be saved in Google Drive and mounted on to Colab in order to be accessible later. Once this is done all you need to do is open your Google Drive to access the files.

17.2 ACCESS FILES FROM YOUR GOOGLE DRIVE

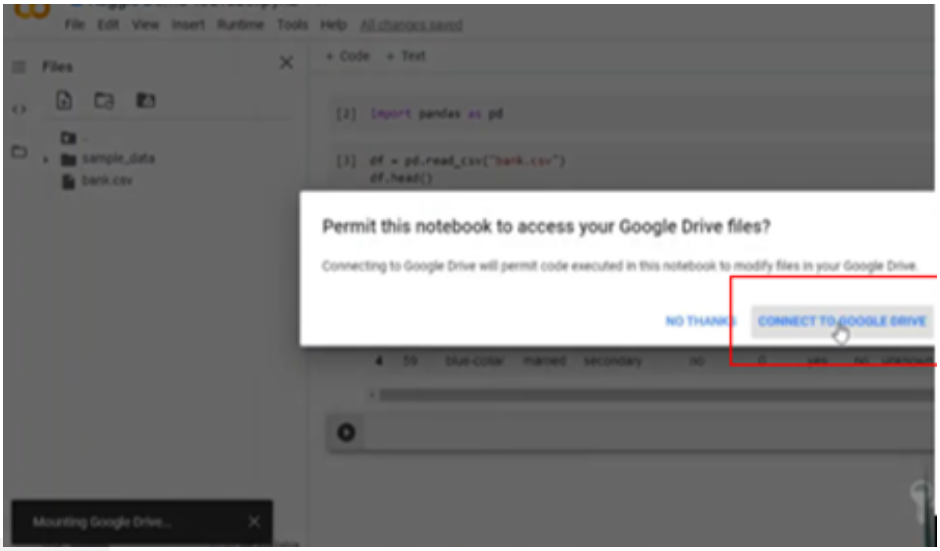
Step 1: Open the File from your Google Drive

Click on Mount Drive icon as shown below

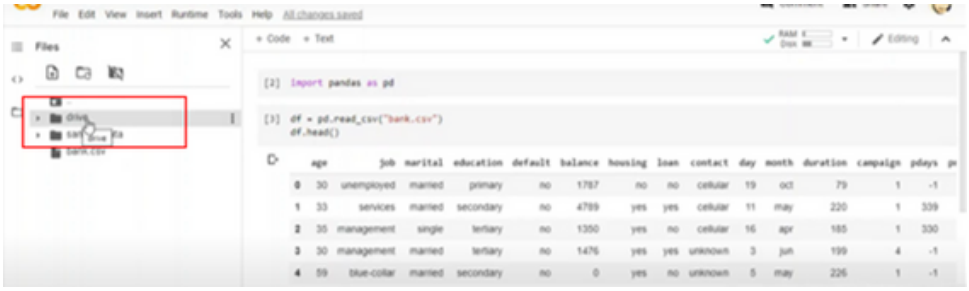


Step 2: Connect to Google Drive

Click on Connect to Google Drive:



As you see below, the drive folder has been created



Step 3: Access the files from Google Drive

To read the files from Google Drive:

18. HOW TO INSTALL MACHINE LEARNING LIBRARIES

As you all know Machine Learning Libraries are the ones that keep the AI magic well working on its wheels. Machine Learning, as the name suggests, is the science of programming a computer by which they are able to learn from different kinds of data. A more general definition given by Arthur Samuel is – "Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed." They are typically used to solve various types of life problems.

In the older days, people used to perform Machine Learning tasks by manually coding all the algorithms and mathematical and statistical formula. This made the process time consuming, tedious and inefficient. But in the modern days, it is become very much easy and efficient compared to the olden days by various python libraries, frameworks, and modules. Today, Python is one of the most popular programming languages for machine learning, and it has replaced many languages in the industry. This is mainly due to its of its vast collection of libraries.

Some important Python libraries used in Machine Learning are as below:

- TensorFlow
- Keras
- PyTorch
- Pandas
- Matplotlib
- Numpy
- Scipy
- Scikit-learn
- Theano

However, there are many more libraries that are used by programmers and data scientists.

Colab supports most of machine learning libraries available in the market. Let us take a quick overview of how to install these libraries in your Colab notebook.

Colab is a virtual machine (VM) you can access directly.

To run commands at the VM's terminal, prefix the line with an exclamation point (!)

To install a library, you can use either of these options −

!pip install
or
!apt-get install

18.1 EXAMPLES

To install Keras, use the following command –

!pip install keras

To install PyTorch, use the following command –

!pip3 install torch torchvision

To install MxNet execute the following commands –

!apt install libnvrtc8.0

!pip install mxnet-cu80

To install OpenCV use the following command –

!apt-get -qq install -y libsm6 libxext6 && pip install -q -U opencv-python

To install XGBoost, use the following command –

!pip install -q xgboost==0.4a30

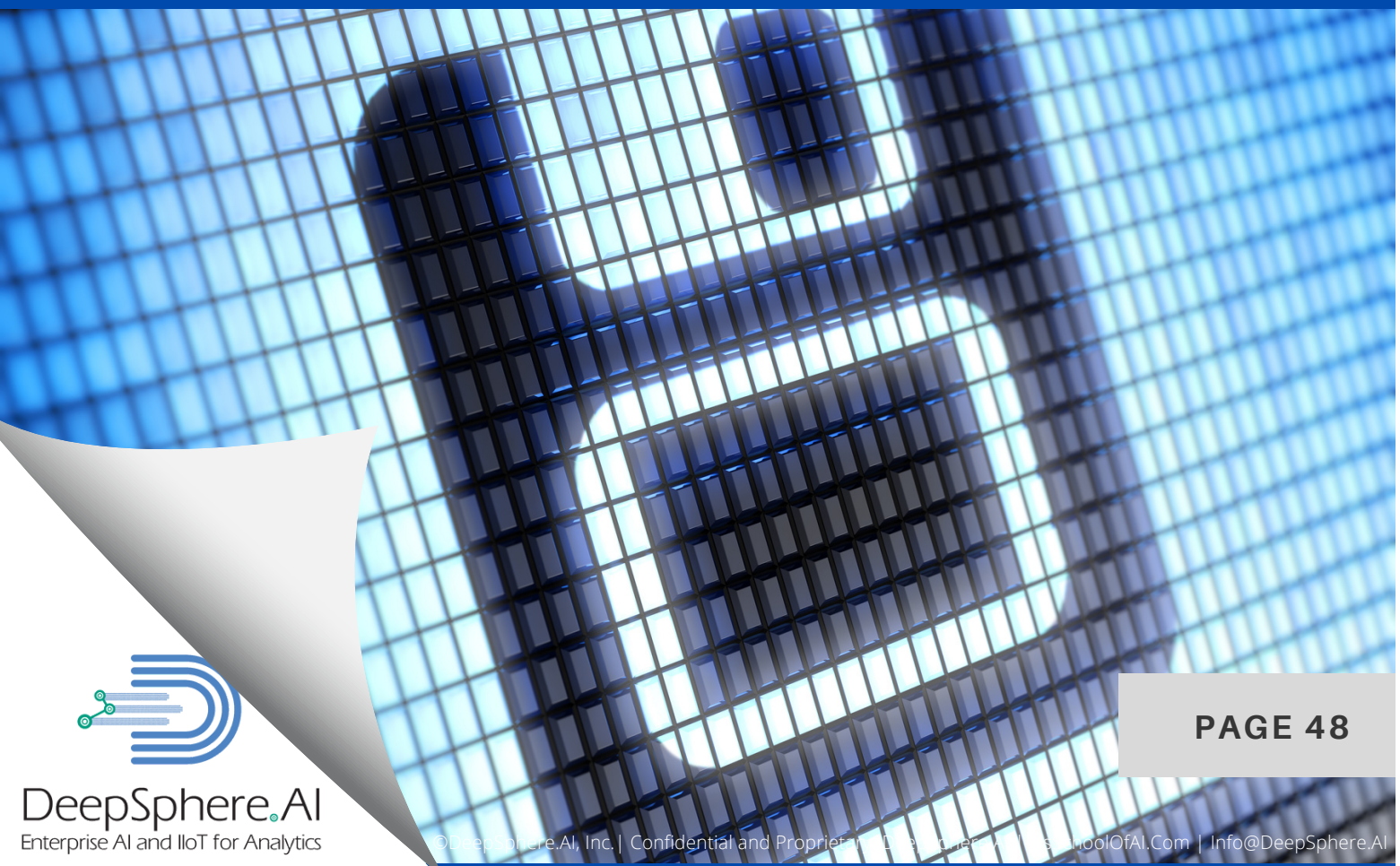
To install GraphViz, use the following command –

!apt-get -qq install -y graphviz && pip install -q pydot

19. HOW TO SAVE THE COLAB NOTEBOOK

UNIT TOPICS

- 19.1 Saving it on Google Drive
- 19.2 Saving it on Git Hub
- 19.3 Cloning Git Repository



19. HOW TO SAVE THE COLAB NOTEBOOK

While the Colab is so much a replica of the Jupyter Notebook, it has some basic differences like reading and writing on the cloud.

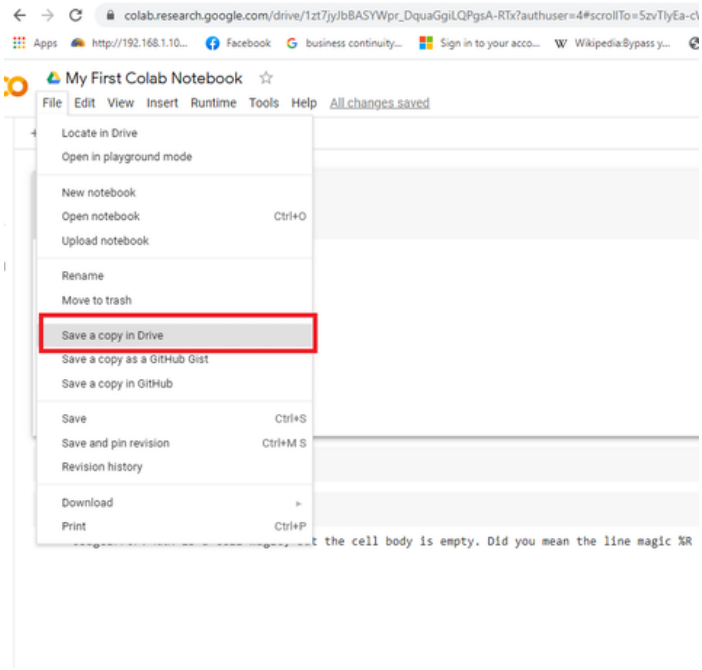
The Notebook forms the very basis of the Google Colaboratory. The Colab in every sense is built on notebooks.

Just like how you can open a new notebook, add pages or sheets an rows in the excel workbook the notebook on Colab allows you to perform similar functions. Let's see how the Colab allows you to save your work to Google Drive or even directly to your GitHub repository.

19.1 SAVING IT ON GOOGLE DRIVE

Colab allows you to save your work to your Google Drive.

To save your notebook, select the following menu options and you will see the below screen –



The action will create a copy of your notebook and save it to your drive. Later on, you may rename the copy to a name of your choice.

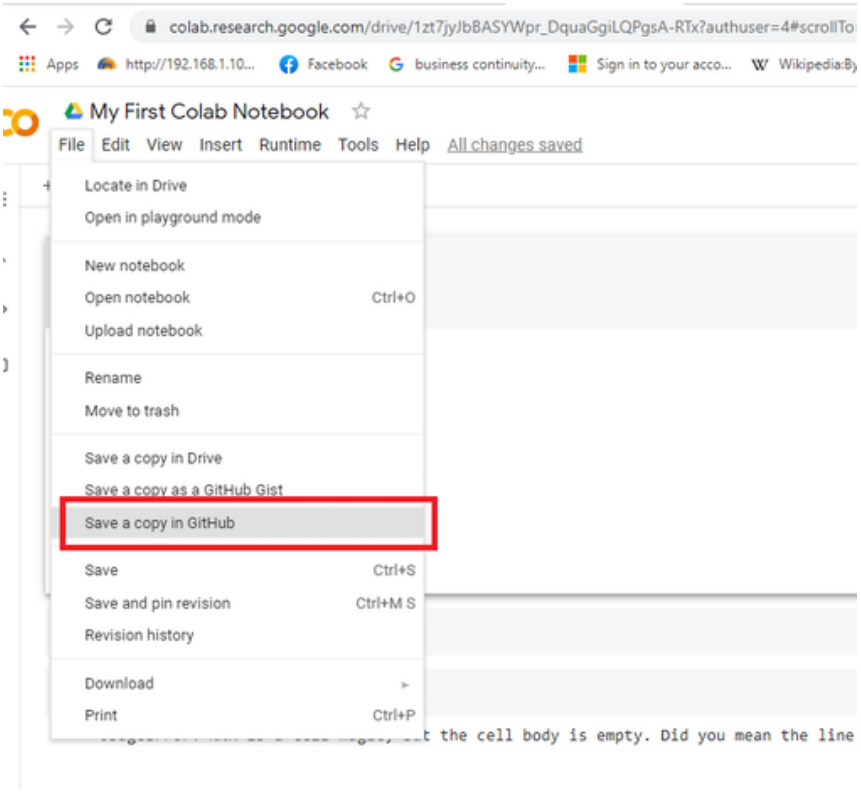
19.2 SAVING IT ON GIT HUB

Colab allows you to save your work to Google Drive or even directly to your GitHub repository.

You may also save your work to your GitHub repository by selecting the following menu options:

File / Save a copy in GitHub

The menu selection is shown in the following screenshot for your quick reference:



You will have to wait until you see the login screen of GitHub. Enter your credentials. If you do not have a repository, create a new one and save your project.

19.3 CLONING GIT REPOSITORY

If a project has already been set up in a central repository, the git clone command is the most common way for users to obtain a development copy. Once a developer has obtained a working copy, all version control operations and collaborations are managed through their local repository.

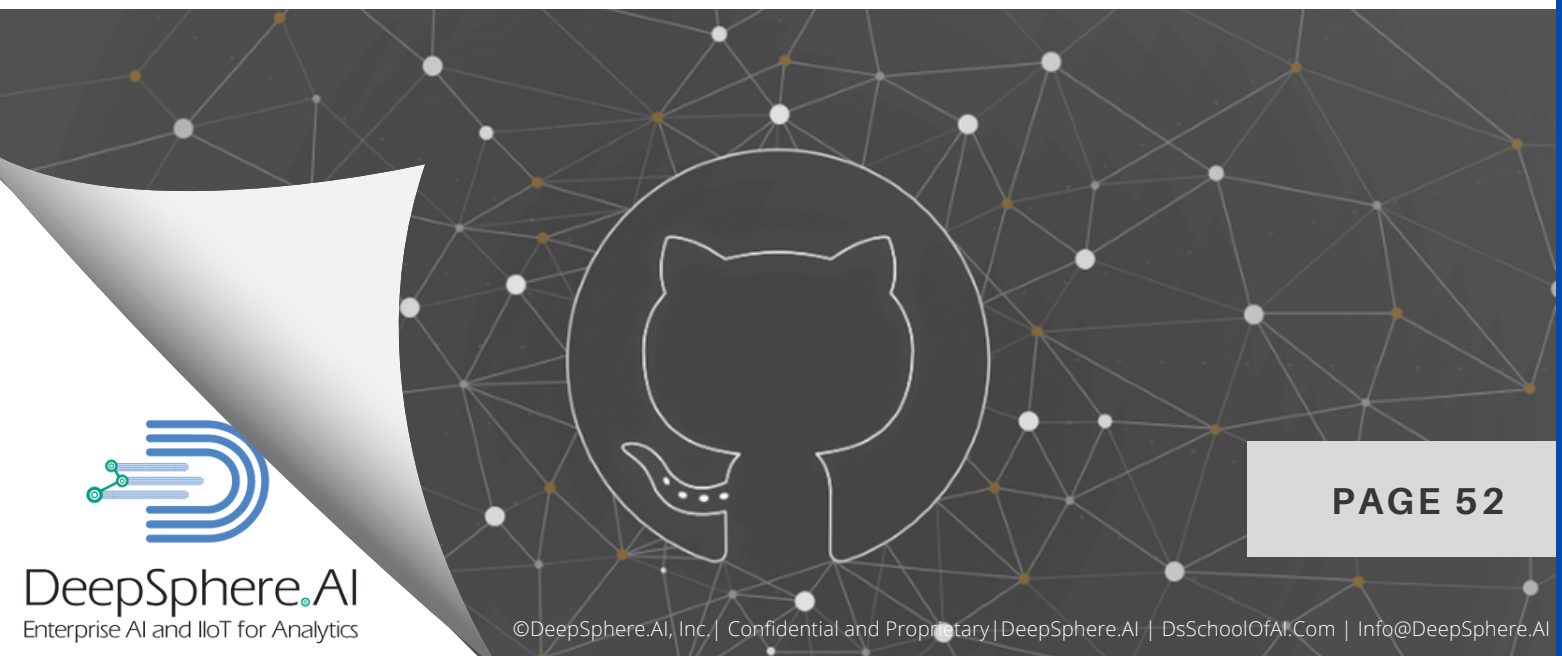
You can clone the entire GitHub repository into Colab using the gitcommand. For example, to clone the Keras tutorial, type the following command in the Code cell –
!git clone https://github.com/wxs/keras-mnist-tutorial.git

After a successful run of the command, you would see the following output –

Cloning into 'keras-mnist-tutorial'...
remote: Enumerating objects: 26, done.

remote: Total 26 (delta 0), reused 0 (delta 0), pack-reused 26
Unpacking objects: 100% (26/26), done.

Once the repo is cloned, locate a Jupyter project (e.g. MINST in keras.ipynb) in it, right-click on the file name and select Open With / Colaboratory menu option to open the project in Colab.



20. LIMITATIONS OF GOOGLE COLAB

- In order to be able to offer computational resources for free, Colab needs to maintain the flexibility to adjust usage limits and hardware availability on the fly.

- Resources available in Colab vary over time to accommodate fluctuations in demand, as well as to accommodate overall growth and other factors.

- This means that overall usage limits as well as idle timeout periods, maximum VM lifetime, GPU types available, and other factors vary over time.
- GPUs and TPUs are sometimes prioritized for users who use Colab interactively rather than for long-running computations, or for users who have recently used less resources in Colab. As a result, users who use Colab for long-running computations, or users who have recently used more resources in Colab, are more likely to run into usage limits and have their access to GPUs and TPUs temporarily restricted.
- Notebooks will also disconnect from VMs when left idle for too long. Maximum VM lifetime and idle timeout behaviour may vary over time, or based on your usage.

21.APPENDIX

21.1. Important Definitions

21.1.1. GCP

- Google Cloud Platform, offered by Google, is a suite of cloud computing services that runs on the same infrastructure that Google uses internally for its end-user products, such as Google Search, Gmail, file storage, and YouTube.

21.1.2. CPU

- Graphics processing unit, a specialized processor originally designed to accelerate graphics rendering. GPUs can process many pieces of data simultaneously, making them useful for machine learning, video editing, and gaming applications.

21.1.3. TPU

- Tensor Processing Unit is an AI accelerator application-specific integrated circuit (ASIC) developed by Google specifically for neural network machine learning, particularly using Google's own TensorFlow software.

21.1.4. Notebook

- A notebook is a list of cells. Cells contain either explanatory text or executable code and its output. Click a cell to select it.

21.1.4. Runtime

- Runtime is the period of time when a program is running. It begins when a program is opened (or executed) and ends when the program is quit or closed. Runtime is a technical term, used most often in software development.

21.1.6. RAM

- RAM stands for Random Access Memory. RAM allows your computer to perform many of its everyday tasks, such as loading applications, browsing the internet, editing a spreadsheet, or experiencing the latest game. Memory also allows you to switch quickly among these tasks, remembering where you are in one task when you switch to another task.

21.1.7. Machine Learning

- Machine learning is a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention.

21.1.9. R

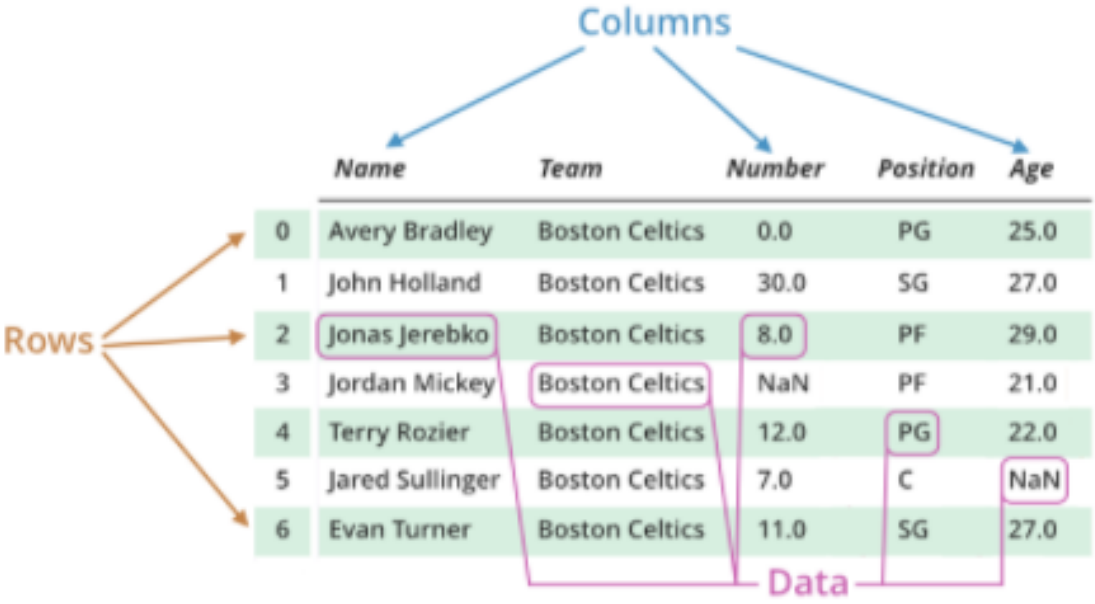
- R is a programming language and free software environment for statistical computing and graphics supported by the R Foundation for Statistical Computing. The R language is widely used among statisticians and data miners for developing statistical software and data analysis

21.1.10. Scala

- Scala is a general-purpose programming language providing support for both object-oriented programming and functional programming. The language has a strong static type system. Designed to be concise, many of Scala's design decisions are aimed to address criticisms of Java.

21.1.11. DataFrame

A DataFrame simply represents a table of data with rows and columns. See the reference inmage below:



	Name	Team	Number	Position	Age
0	Avery Bradley	Boston Celtics	0.0	PG	25.0
1	John Holland	Boston Celtics	30.0	SG	27.0
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN
6	Evan Turner	Boston Celtics	11.0	SG	27.0

22. LANGUAGES SUPPORTED BY GOOGLE COLAB

Although the Colab is primarily used for coding in Python, apparently we can also use it for R and Scala.

22.1. HOW TO USE R IN GOOGLE COLAB

More about R:

R is a language and environment for statistical computing and graphics. R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering,) and graphical techniques, and is highly extensible.

·There are two ways to run R in Colab:

The first way is to use the rpy2 package in the Python runtime. This method allows you to execute R and Python syntax together.

The second way is to actually start the notebook in the R runtime.

22.1.1 How to use R and Python together in Colab

• Open your favourite browser.

• Create a new notebook: https://colab.research.google.com/#create=true

• Run rmagic by executing this command

%load_ext rpy2.ipython

After that, ensure you add %%R in the beginning of each cell , every time you use R. Note that this must be placed at the beginning of the cell.

```
%%R
x <- seq(0, 2*pi, length.out=50)
x
```

• These lines will return a variable x, and display it on the cell output.

• Use %R to execute line magic. Use this if you want a single line in a cell to be executed in R.

• Here is how you could use this line magic to copy R variable to Python

```
x = %R x
```

To access the files from google drive using R, mount the google drive as explained in 17.2 and then we need to use rmagic as shown below.

```
%%R
install.packages("readxl")
library("readxl")
book1 = read_excel("drive/MyDrive/Book1.xlsx")
book1
```

```
# A tibble: 76 x 9
   `FYI. Sample inf… ...2   ...3  ...4   ...5   ...6  ...7   ...8     ...9
   <chr>             <chr>  <chr> <chr>  <chr>  <chr> <chr>  <chr>    <chr>
 1 <NA>              <NA>   <NA>  <NA>   <NA>   <NA>  <NA>   <NA>     <NA>
 2 19.29 to 20.25    <NA>   <NA>  <NA>   <NA>   <NA>  <NA>   <NA>     <NA>
 3 24.26 to 25.47    <NA>   <NA>  <NA>   <NA>   <NA>  <NA>   <NA>     <NA>
 4 <NA>              <NA>   <NA>  <NA>   <NA>   <NA>  <NA>   <NA>     <NA>
 5 <NA>              <NA>   <NA>  <NA>   <NA>   <NA>  <NA>   <NA>     <NA>
 6 Account#           ID    Month Actua… Billed Rate  Rate i… Cola Va… % Incr
 7 1000123456        222681 43466 199.41 160    19.29 Y       153.600… 4.9766
 8 1000123456        580325 43466 209.16 160    19.29 <NA>    <NA>     <NA>
 9 1000123456        513650 43466 178.16 160    19.29 <NA>    <NA>     <NA>
10 1000123456        180395 43466 184.61 160    19.29 <NA>    <NA>     <NA>
# … with 66 more rows
```

22.1.2. How to use R runtime in Colab

To use the notebook directly with R:

Open your favourite browser.

Go to this URL: https://colab.research.google.com/#create=true&language=r,
or this shortened URL https://colab.to/r

After accessing the URL, you will be taken to a new Colab notebook with the default title Unitled.ipynb.

At the first glance, there is no difference between notebooks with Python and R runtimes. However, if we go to the "Runtime" settings, and select "Change runtime type", we get a dialog box confirming that we are already in R runtime.

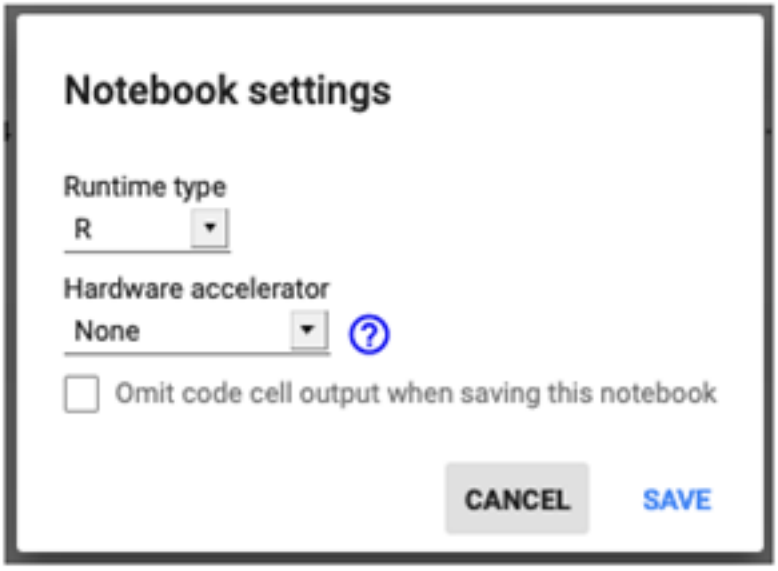


Image by Author, Runtime -> Change runtime type

You have now successfully run R in Colab. You can check the R version by typing R.version.string to print out the R version

```
R.version.string
```

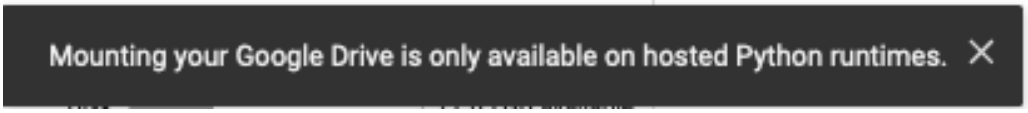
```
'R version 4.0.4 (2021-02-15)'
```

Here, several packages that are useful for data processing and data visualization are already available. You can check it by running print (installed.packages()).

```
print(installed.packages())
```

```
           Package      LibPath                          Version
IRdisplay  "IRdisplay"  "/usr/local/lib/R/site-library"  "1.0"
IRkernel   "IRkernel"   "/usr/local/lib/R/site-library"  "1.1.1"
pbdZMQ     "pbdZMQ"     "/usr/local/lib/R/site-library"  "0.3-5"
repr       "repr"       "/usr/local/lib/R/site-library"  "1.1.3"
uuid       "uuid"       "/usr/local/lib/R/site-library"  "0.1-4"
askpass    "askpass"    "/usr/lib/R/site-library"        "1.1"
assertthat "assertthat" "/usr/lib/R/site-library"        "0.2.1"
backports  "backports"  "/usr/lib/R/site-library"        "1.2.1"
base64enc  "base64enc"  "/usr/lib/R/site-library"        "0.1-3"
BH         "BH"         "/usr/lib/R/site-library"        "1.75.0-0"
blob       "blob"       "/usr/lib/R/site-library"        "1.2.1"
brew       "brew"       "/usr/lib/R/site-library"        "1.0-6"
brio       "brio"       "/usr/lib/R/site-library"        "1.1.1"
```

Unfortunately, we cannot mount Google Drive when we are in the R runtime. The below is the error message.



22.2. HOW TO ACCESS SCALA IN GOOGLE COLAB

More about Scala

Scala is a type-safe JVM (as it is built to run in Java Virtual Machine) language that incorporates both object oriented and functional programming into an extremely concise, logical, and extraordinarily powerful language. Some may be surprised to know that Scala is not quite as new as they thought, having first been introduced in 2003. However, it is particularly within the past few years that Scala has begun to develop a significant following

Scala is not currently a built-in language for Colab, so before we start writing code we need to install Scala to support the environment.

Since the Colab file system is reset after some inactivity, you'll need to re-run this installer whenever you return to a notebook after your log out .

22.2.1. Scala Installer for Colaboratory

The codes mentioned below installs Almond, a Scala kernel for Jupyter, into the Colaboratory and configures it to make the pre-installed Python libraries accessible to Scala code (through ScalaPy).

Since this installer is built on Almond, you get complete Scala support inside your notebooks, including the ability to get your code completed and accessed through Jupyter-specific APIs.

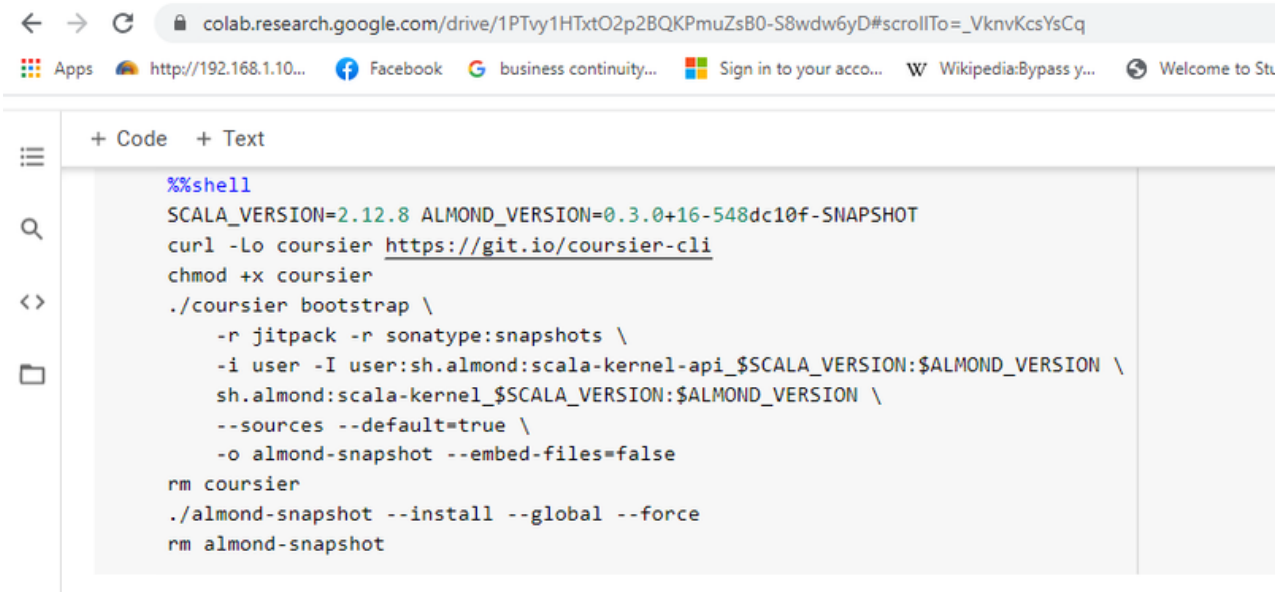
If you are accessing the notebook through GitHub, make sure to not reset your runtime when running this notebook. In order to install the Scala kernel, the installer writes data to the runtime file system, so resetting would clear out or delete the kernel.

As the first step, open the below link and start by running the two codes one after the other mentioned under the link.

https://colab.research.google.com/drive/1Jyg_e0Jnfx-2Jt0eB77sgYV6rHKWbSE5

%%shell

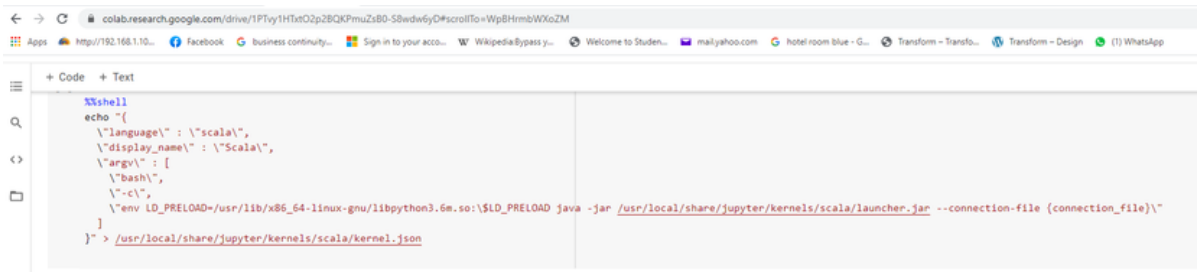
You will arrive at the results mentioned in the below screenshot

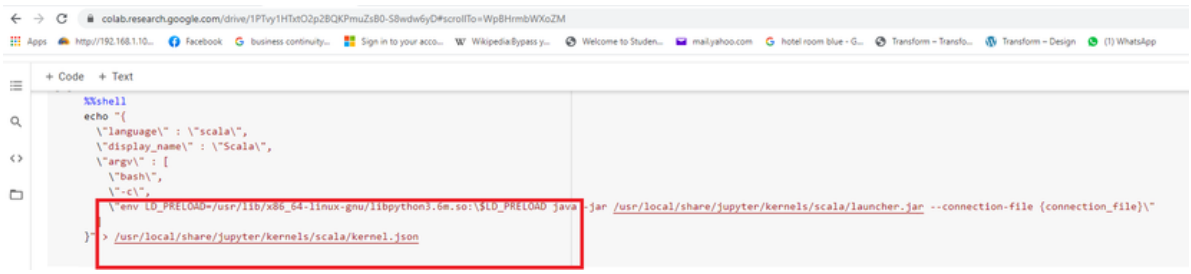


Run the code again in another cell as seen to get the results as mentioned in the screenshot below

%%shell

You will arrive at the results mentioned in the below screenshot





Once you are done reload the page to load the notebook in the Scala kernel that was installed.

SAMPLE CODES:

```
println("Hello, world!")
```

```
Hello, world!
```

```
var b_long = 83456794891
```

```
b_long: Long = 8345679489L
```

```
val capitals= Map("Argentina" -> "Buenos Aires", "Canada" -> "Ottowa",
 "Egypt" -> "Cairo", "Liberia" -> "Monrovia", "Netherland" -> "Amstradam",
  "United Stats"->"Washinghton D.C")
```

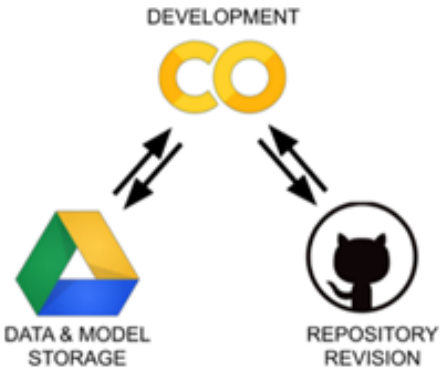
```
capitals: Map[String, String] = Map(
  "Argentina" -> "Buenos Aires",
  "Egypt" -> "Cairo",
  "Canada" -> "Ottowa",
  "Liberia" -> "Monrovia",
  "Netherland" -> "Amstradam",
  "United Stats" -> "Washinghton D.C"
)
```

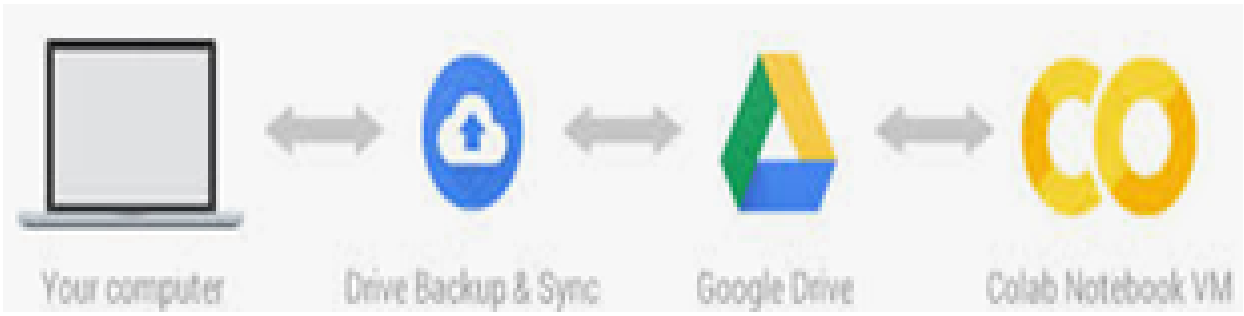
```
capitals.keys
```

```
res10: Iterable[String] = Set(
  "Argentina",
  "Egypt",
  "Canada",
  "Liberia",
  "Netherland",
  "United Stats"
)
```

23. COLAB ARCHITECTURE

The Google Colab is a real game changer basically because it is structured that way. It allows us to quickly start executing scripts without worrying about the underlying architecture. The notebook is a good example of that, as its user just has to insert a new function.





SAMPLE PROGRAMS

The DeepSphere's guide is designed to give the learner practice to implement the codes on the Colab environment and we have added Sample Codes to enable beginners practice on the platform and understand the workings. Since the platform supports the Python, R and Scala, we have added those codes to enable easy implementation and learning. Here we go!

- Sample Programs using Python
- Sample Programs using R
- Sample Programs using Scala

DeepSphere.AI
Enterprise AI and IIoT for Analytics

24. SAMPLE PROGRAMS USING PYTHON

24.1. CREATING A DATA FRAME

```
import pandas as pd

#Creating a dictionary
cars = {'Brand': ['Honda Civic','Toyota Corolla','Ford Focus','Audi A4'],
        'Price': [22000,25000,27000,35000]
        }
#Converting dictionary to dataframe and defining the column names
df = pd.DataFrame(cars, columns = ['Brand', 'Price'])
#printing the dataframe
print(df)
```

	Brand	Price
0	Honda Civic	22000
1	Toyota Corolla	25000
2	Ford Focus	27000
3	Audi A4	35000

24.2. FINDING DATATYPE

```
[4]  #Finding data type
     df.dtypes
```

```
Brand    object
Price     int64
dtype: object
```

24.3. CONVERTING DATA TYPE

```
#Converting data format of Price column to string type
df['Price'] = df['Price'].astype(str)
#Finding data type
df.dtypes
```

```
[7]  df.dtypes

Brand    object
Price    object
dtype: object
```

23.4. CREATING NEW COLUMN

```
#Creating new column by adding 5000 to the price column
df['Price with tax'] = df['Price'] + 5000
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/usr/local/lib/python3.7/dist-packages/pandas/core/ops/array_ops.py in na_arithmetic_op(left, right, op, is_cmp)
    142     try:
--> 143         result = expressions.evaluate(op, left, right)
    144     except TypeError:

                                8 frames
TypeError: can only concatenate str (not "int") to str

During handling of the above exception, another exception occurred:

TypeError                                 Traceback (most recent call last)
/usr/local/lib/python3.7/dist-packages/pandas/core/ops/array_ops.py in masked_arith_op(x, y, op)
    110         if mask.any():
    111             with np.errstate(all="ignore"):
--> 112                 result[mask] = op(xrav[mask], y)
    113
    114     result, _ = maybe_upcast_putmask(result, ~mask, np.nan)

TypeError: can only concatenate str (not "int") to str

SEARCH STACK OVERFLOW
```

We get the above error as we are performing a numeric calculation on a string column type. Therefore we have to change the data type to numeric to create a new column.

See the code now rightly executed in the screenshot below

```
[9] #Converting Price column type to integer
    df['Price'] = df['Price'].astype(int)

    #Creating new column by adding 5000 to the price column
    df['Price with tax'] = df['Price'] + 5000

    print(df)
```

	Brand	Price	Price with tax
0	Honda Civic	22000	27000
1	Toyota Corolla	25000	30000
2	Ford Focus	27000	32000
3	Audi A4	35000	40000

24.5. CREATING 'FOR' LOOP TO PRINT EACH LETTERS OF A WORD

```
GC = "Google Colab"
for x in GC:
print(x)
```

```
  File "<ipython-input-3-6a0d589a0f3a>", line 3
    print(x)
        ^
IndentationError: expected an indented block
```

SEARCH STACK OVERFLOW

We've got an Indentation error, as the first line of the 'for' loop must end with a colon, and the body must be indented. Generally, four whitespaces are used for indentation and are preferred over tabs.

Now, let's execute the code with proper indentation:

```
GC = "Google Colab"
for x in GC:
  print(x)
```

```
G
o
o
g
l
e

C
o
l
a
b
```

24.6. APLYING FILTERS ON DATASET

24.6.1 Creating dataset

```
#First creating dataframe
import pandas as pd
import numpy as np

ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
   'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
   'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
   'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
   'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
df = pd.DataFrame(ipl_data)
df.head()
```

	Team	Rank	Year	Points
0	Riders	1	2014	876
1	Riders	2	2015	789
2	Devils	2	2014	863
3	Devils	3	2015	673
4	Kings	3	2014	741

24.6.2 Applying Filter

```
[10] Filtered_data = df[df['Year'] = 2014]
     Filtered_data
```

```
  File "<ipython-input-10-21c3bce34b97>", line 1
    Filtered_data = df[df['Year'] = 2014]
                                  ^
SyntaxError: invalid syntax
```

SEARCH STACK OVERFLOW

We get this syntax error as we've used assignment operator (=) instead of comparison operator (==).

24.6.3 Using comparison operator for filtering

```
[12] #Running the code again with comparison operator
     Filtered_data = df[df['Year'] == 2014]
     Filtered_data
```

	Team	Rank	Year	Points
0	Riders	1	2014	876
2	Devils	2	2014	863
4	Kings	3	2014	741
9	Royals	4	2014	701

24.7. SORTING DATASET

24.7.1 Sort by ascending order

```
[19] Filtered_data.sort_values(by = "Points")
```

	Team	Rank	Year	Points
9	Royals	4	2014	701
4	Kings	3	2014	741
2	Devils	2	2014	863
0	Riders	1	2014	876

24.7.2 Sort by descending order

```
[18] Filtered_data.sort_values(by = "Points", ascending = False)
```

	Team	Rank	Year	Points
0	Riders	1	2014	876
2	Devils	2	2014	863
4	Kings	3	2014	741
9	Royals	4	2014	701

24.7.3 Sort by multiple columns

```
[8] Filtered_data.sort_values(by = ("Team", "Rank"), ascending = False)

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-8-daf1024b6f83> in <module>()
----> 1 Filtered_data.sort_values(by = ("Team", "Rank"), ascending = False)

                                  1 frames
/usr/local/lib/python3.7/dist-packages/pandas/core/generic.py in _get_label_or_level_values(self, key, axis)
   1561             values = self.axes[axis].get_level_values(key)._values
   1562         else:
-> 1563             raise KeyError(key)
   1564
   1565         # Check for duplicates

KeyError: ('Team', 'Rank')
```

We get this key error as we've used parentheses () to list the column names which we want to sort. In order to avoid that we need to use square brackets [] to list the multiple column names as shown below.

```
[17] Filtered_data.sort_values(by = ["Team", "Rank"], ascending = False)
```

	Team	Rank	Year	Points
9	Royals	4	2014	701
0	Riders	1	2014	876
4	Kings	3	2014	741
2	Devils	2	2014	863

24.8 MERGING DATASETS

Pandas provides a single function called merge, as the entry point for all standard database join operations between Dataframe objects.

```
[20] #Importing pandas library
     import pandas as pd

     #Creating 2 dataframe
     data1 = pd.DataFrame({
        'id':[1,2,3,4,5],
        'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
        'subject_id':['sub1','sub2','sub4','sub6','sub5']})
     data2 = pd.DataFrame({
        'id':[1,2,3,4,5],
        'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
        'subject_id':['sub2','sub4','sub3','sub6','sub5']})

     #Merging 2 datasets using Inner join
     Merged_data = pd.merge(data1, data2, on=subject_id, how='inner')
     Merged_data
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-20-5cc4d356159a> in <module>()
     13
     14 #Merging 2 datasets using Inner join
---> 15 Merged_data = pd.merge(data1, data2, on=subject_id, how='inner')
     16 Merged_data

NameError: name 'subject_id' is not defined
```

We get this name error as the column name (subject_id) is not within quotes in the merge statement.

```
[20] #Importing pandas library
     import pandas as pd

     #Creating 2 dataframe
     data1 = pd.DataFrame({
        'id':[1,2,3,4,5],
        'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
        'subject_id':['sub1','sub2','sub4','sub6','sub5']})
     data2 = pd.DataFrame({
        'id':[1,2,3,4,5],
        'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
        'subject_id':['sub2','sub4','sub3','sub6','sub5']})

     #Merging 2 datasets using Inner join
     Merged_data = pd.merge(data1, data2, on=subject_id, how='inner')
     Merged_data
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-20-5cc4d356159a> in <module>()
     13
     14 #Merging 2 datasets using Inner join
---> 15 Merged_data = pd.merge(data1, data2, on=subject_id, how='inner')
     16 Merged_data

NameError: name 'subject_id' is not defined
```

We get this name error as the column name (subject_id) is not within quotes in the merge statement. See the code rightly executed below:

```
[ ] #Importing pandas library
    import pandas as pd

    #Creating 2 dataframe
    data1 = pd.DataFrame({
       'id':[1,2,3,4,5],
       'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
       'subject_id':['sub1','sub2','sub4','sub6','sub5']})
    data2 = pd.DataFrame({
       'id':[1,2,3,4,5],
       'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
       'subject_id':['sub2','sub4','sub3','sub6','sub5']})

    #Merging 2 datasets using Inner join
    Merged_data = pd.merge(data1, data2, on='subject_id', how='inner')
    Merged_data
```

	id_x	Name_x	subject_id	id_y	Name_y
0	2	Amy	sub2	1	Billy
1	3	Allen	sub4	2	Brian
2	4	Alice	sub6	4	Bryce
3	5	Ayoung	sub5	5	Betty

25. SAMPLE PROGRAMS IN R

Before you start running programs in R first run rmagic by executing this command:

%load_ext rpy2.ipython

25.1 CREATING AND ADDING VECTORS

```
%%R
#Creating vectors
x = c(1,2,3)
y = c(4,5,6)
```

```
%%R
#Adding 2 vectors
x+y
```

```
[1] 5 7 9
```

25.2 CREATING A DATA FRAME

```
%%R
#Creating data frame
data <- data.frame(
   emp_id = c (1:5),
   emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
   salary = c(623.3,515.2,611.0,729.0,843.25),
   stringsAsFactors = FALSE
)
data
```

```
  emp_id emp_name salary
1      1     Rick 623.30
2      2      Dan 515.20
3      3 Michelle 611.00
4      4     Ryan 729.00
5      5     Gary 843.25
```

25.3 ADDING NEW COLUMN

```
[12] %%R
     survey <- data.frame("index" = c(1, 2, 3, 4, 5),
                          "age" = c(24, 25, 42, 56, 22))
     survey$sex <- ("m", "m", "f", "f", "m")
     survey
```

```
---------------------------------------------------------------------------
RParsingError                             Traceback (most recent call last)
<ipython-input-12-13d544ec65ed> in <module>()
----> 1 get_ipython().run_cell_magic('R', '', 'survey <- data.frame("index" = c(1, 2, 3, 4, 5),\n
"age" = c(24, 25, 42, 56, 22))\nsurvey$sex <- ("m", "m", "f", "f", "m")\nsurvey')

                                    7 frames
<decorator-gen-119> in R(self, line, cell, local_ns)

/usr/local/lib/python3.7/dist-packages/rpy2/rinterface_lib/_rinterface_capi.py in _parse(cdata, num, rmemory)
    651     if status[0] != openrlib.rlib.PARSE_OK:
    652         raise RParsingError('Parsing status not OK',
--> 653                             status=PARSING_STATUS(status[0]))
    654     return res

RParsingError: Parsing status not OK - PARSING_STATUS.PARSE_ERROR
```

We get the above error, as we haven't used 'c' while listing the new column values. The 'c' function in R programming stands for 'combine'. The code rightly executed below:

```
[13] %%R
     survey <- data.frame("index" = c(1, 2, 3, 4, 5),
                          "age" = c(24, 25, 42, 56, 22))
     survey$sex <- c("m", "m", "f", "f", "m")
     survey
```

```
  index age sex
1     1  24   m
2     2  25   m
3     3  42   f
4     4  56   f
5     5  22   m
```

25.4 ADDING NEW CALCULATED COLUMN

```
[14] %%R
     survey$newcol <- survey$age + survey$sex
     survey
```

```
R[write to console]: Error in survey$age + survey$sex :
  non-numeric argument to binary operator


Error in survey$age + survey$sex :
  non-numeric argument to binary operator
```

We get this error as we are trying to create a new column by adding a numeric column with non-numeric column. See the code rightly executed below:

```
[15] %%R
     survey$newcol <- survey$age + 10
     survey
```

	index	age	sex	newcol
1	1	24	m	34
2	2	25	m	35
3	3	42	f	52
4	4	56	f	66
5	5	22	m	32

24.5 CREATING 'FOR' LOOP

```
%%R
# Create fruit vector
fruit <- c('Apple', 'Orange', 'Passion fruit', 'Banana')
# Create the for statement
for ( i in fruit){
print i
}
```

```
---------------------------------------------------------------------------
RParsingError                             Traceback (most recent call last)
<ipython-input-19-ef186289cbce> in <module>()
----> 1 get_ipython().run_cell_magic('R', '', "# Create fruit vector\nfruit <- c('Apple', 'Orange', 'Passion fruit',
'Banana')\n# Create the for statement\nfor ( i in fruit){ \nprint i\n}")

                                    7 frames
<decorator-gen-119> in R(self, line, cell, local_ns)

/usr/local/lib/python3.7/dist-packages/rpy2/rinterface_lib/_rinterface_capi.py in _parse(cdata, num, rmemory)
    651     if status[0] != openrlib.rlib.PARSE_OK:
    652         raise RParsingError('Parsing status not OK',
--> 653                             status=PARSING_STATUS(status[0]))
    654     return res

RParsingError: Parsing status not OK - PARSING_STATUS.PARSE_ERROR
```

We get this error as there are no parentheses used for printing 'i' variable.

```
[20] %%R
     # Create fruit vector
     fruit <- c('Apple', 'Orange', 'Passion fruit', 'Banana')
     # Create the for statement
     for ( i in fruit){
     print (i)
     }
```

```
[1] "Apple"
[1] "Orange"
[1] "Passion fruit"
[1] "Banana"
```

25.6 CREATING SUBSET BY FILTERING A DATASET

```
[36] %%R
     print(survey)
     subset(survey, sex <> "m")

---------------------------------------------------------------------------
RParsingError                             Traceback (most recent call last)
<ipython-input-36-b94fbf0661fe> in <module>()
----> 1 get_ipython().run_cell_magic('R', '', 'print(survey)\nsubset(survey, sex <> "m")')

                                  7 frames
<decorator-gen-119> in R(self, line, cell, local_ns)

/usr/local/lib/python3.7/dist-packages/rpy2/rinterface_lib/_rinterface_capi.py in _parse(cdata, num, rmemory)
    651     if status[0] != openrlib.rlib.PARSE_OK:
    652         raise RParsingError('Parsing status not OK',
--> 653                             status=PARSING_STATUS(status[0]))
    654     return res

RParsingError: Parsing status not OK - PARSING_STATUS.PARSE_ERROR
```

We get this error as we've used incorrect 'not equal' symbol <>.
In R, 'not equal' symbol is !=

```
[35] %%R
     print(survey)
     subset(survey, sex != "m")
```

```
  index age sex newcol
1     1  24   m     34
2     2  25   m     35
3     3  42   f     52
4     4  56   f     66
5     5  22   m     32
  index age sex newcol
3     3  42   f     52
4     4  56   f     66
```

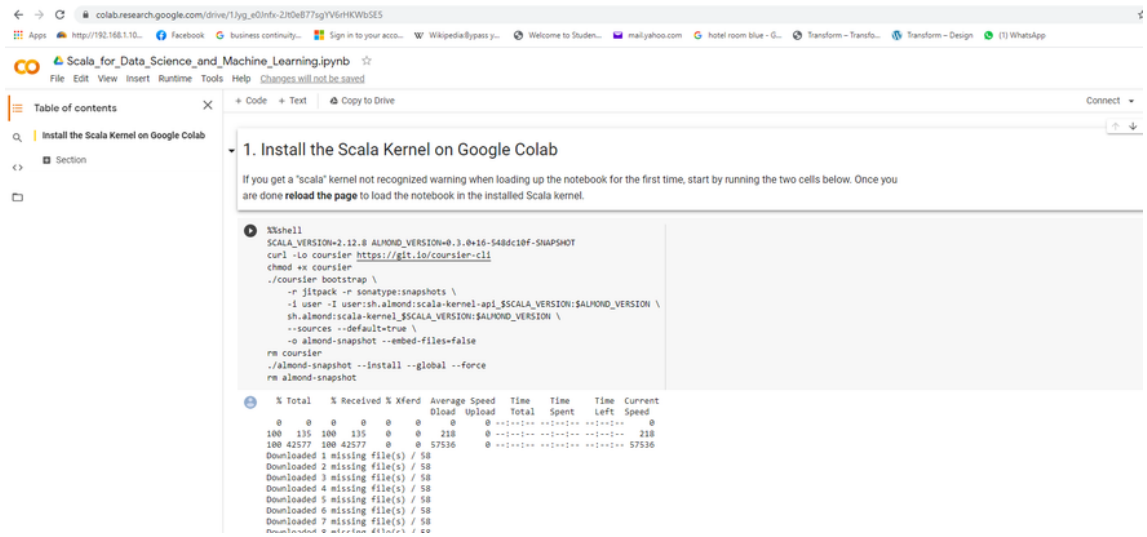
26. SAMPLE PROGRAMS IN SCALA

As mentioned earlier, the platform does not support Scala automatically and hence it is important to open the link below to begin with. Scala is not currently a built-in language for Colab (only Python and Swift are officially supported), so before we start writing code there's a short step needed to install Scala support into your environment. Because the Colab filesystem is reset after some inactivity, you'll need to re-run this installer whenever you return to a notebook after some time.

Run the below codes using this link:

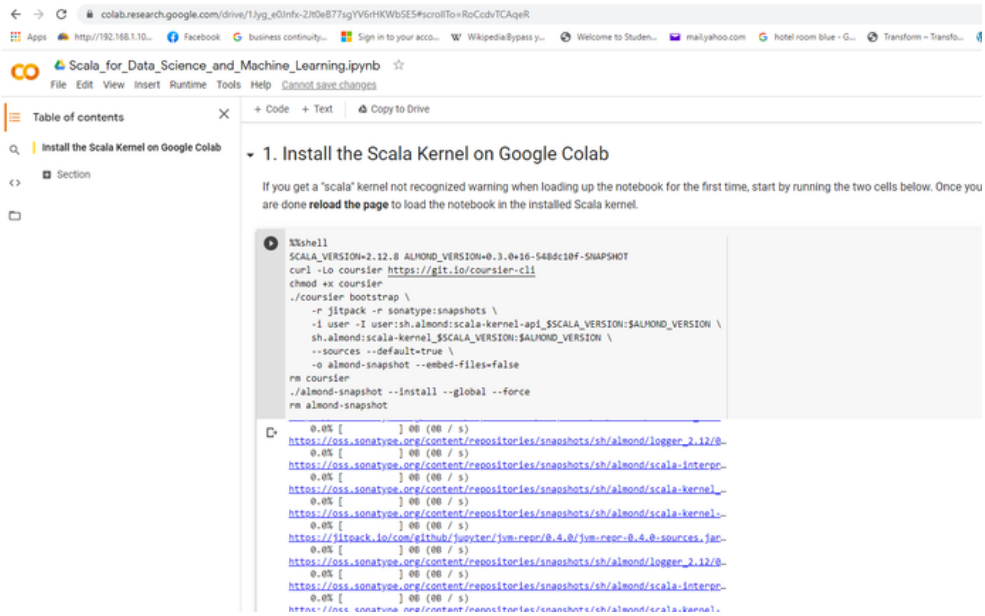
https://colab.research.google.com/drive/1Jyg_e0Jnfx-2Jt0eB77sgYV6rHKWbSE5

26.1 INSTALL THE SCALA KERNEL ON GOOGLE COLAB

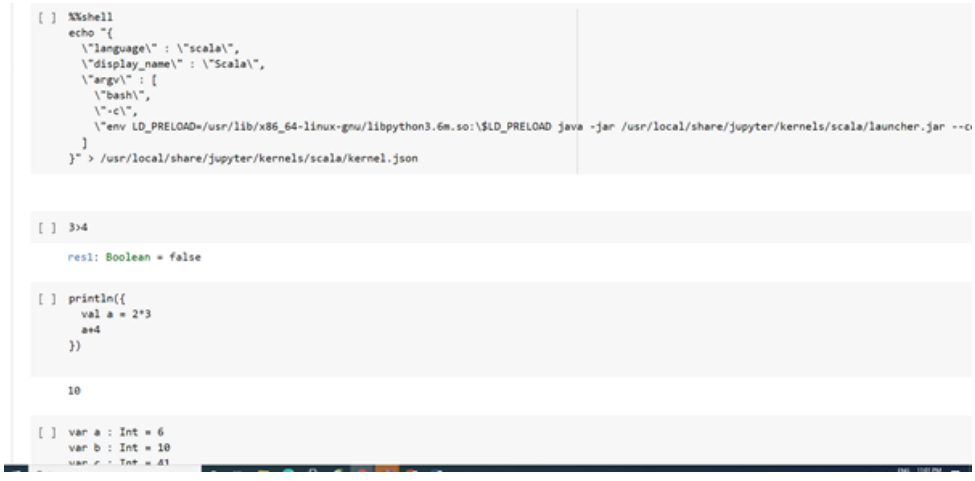


The link opens to the screen above and run the two codes and then reload the page again.

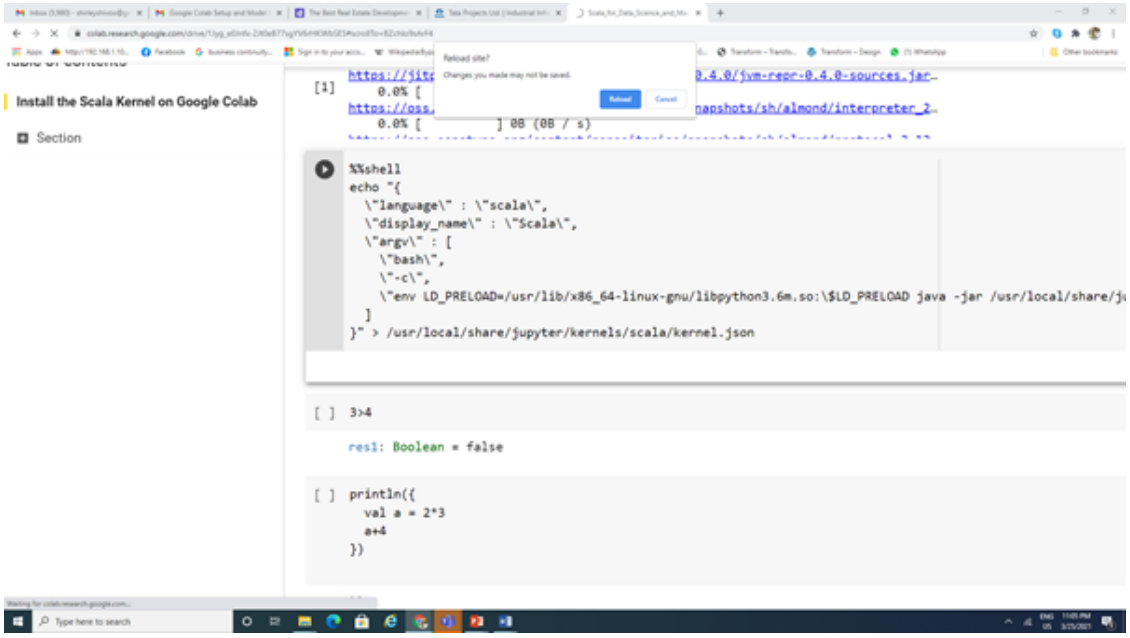
The first code runs for sometime, with links mentioning about the almond kernel getting installed. As seen the screen shot below:



Run the second code block too %%shell as seen in the below screen shot



After you run both codes reload the page



Click reload and then run the codes below

26.2 CREATE A RANGE

```
val myrange = 1 to 10
```

```
myrange: Range.Inclusive = Range.Inclusive(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

26.3 CREATE 'FOR' LOOP

```
for (i <- 1 to 5) println(i)
```

```
1
2
3
4
5
```

26.3 CREATING EXPRESSION

```
println({
  val a = 2*3
  a+4
})
```

```
10
```

26.4 CREATING A FUNCTION

```
def myFunction(a:Int, b:Int): Int = {
  val c = a*b
  return c
}
myFunction(2,4)
```

```
defined function myFunction
res2_1: Int = 8
```

26.5 CREATING SCALA MAP

```
val y = Array("England", "Liberia", "Haiti", "Austrailia", "Sweden" )
y.sorted
```

```
y: Array[String] = Array("England", "Liberia", "Haiti", "Austrailia", "Sweden")
res2_1: Array[String] = Array(
  "Austrailia",
  "England",
  "Haiti",
  "Liberia",
  "Sweden"
)
```

27. REFERENCES

- https://www.tutorialspoint.com/google_colab/google_colab_quick_guide.htm
- https://colab.research.google.com/notebooks/intro.ipyb
- https://medium.com/@shadaj/machine-learning-with-scala-in-google-colaboratory-e6f1661f1c88
- https://medium.com/@senthilnathangautham/colab-gcp-compute-how-to-link-them-together-98747e8d940e
- https://towardsdatascience.com/how-to-use-r-in-google-colab-b6e02d736497

Developed and Designed by

DeepSphere.AI, Inc. 2100 Geng Road, Suite 210 Palo Alto, CA 94303

DeepSphere.AI India Pvt. LTD., #368, First Floor,4th Main Road, Nolambur, Mogappair West, Chennai-600037
DeepSphere.AI | DsSchoolofAI.com

www.deepsphere.ai

DeepSphere.AI
Enterprise AI and IIoT for Analytics