Siddhant Sadangi | 6 min | 5th September, 2023

Machine Learning Tools

Table of contents

Google Colaboratory is a free Jupyter notebook environment that runs on Google's cloud servers, letting the user leverage backend hardware like GPUs and TPUs. This lets you do everything you can in a Jupyter notebook hosted in your local machine, without requiring the installations and setup for hosting a notebook in your local machine.

Colab comes with (almost) all the setup you need to start coding, but what it doesn't have out of the box is your datasets! How do you access your data from within Colab?

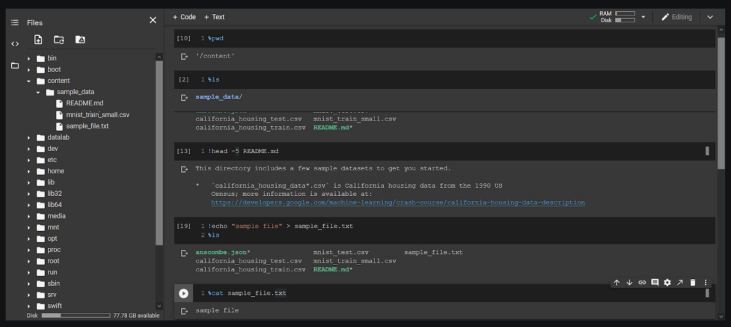
In this article we will talk about:

- How to load data to Colab from a multitude of data sources
- How to write back to those data sources from within Colab
- Limitations of Google Colab while working with external files

Directory and file operations in Google Colab

Since Colab lets you do everything which you can in a locally hosted Jupyter notebook, you can also use shell commands like `ls, dir, pwd, cd, cat, echo`, et cetera using line-magic (%) or bash (!).

To browse the directory structure, you can use the file-explorer pane on the left.



How to upload files to and download files from Google Colab

Load individual files directly from GitHub

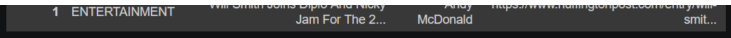
In case you just have to work with a few files rather than the entire repository, you can load them directly from GitHub without needing to clone the repository to Colab.

To do this:

1. click on the file in the repository,
2. click on **View Raw**,
3. copy the URL of the raw file,
4. use this URL as the location of your file.

```
[4]  1 df3 = pd.read_json("News_Category_Dataset_v2.json", lines=True)
     2 df3.head()
```

	category	headline	authors	link
0	CRIME	There Were 2 Mass Shootings In Texas Last Week...	Melissa Jeltsen	https://www.huffingtonpost.com/entry/texas-ama...



GitHub without needing to clone the repository to Colab.

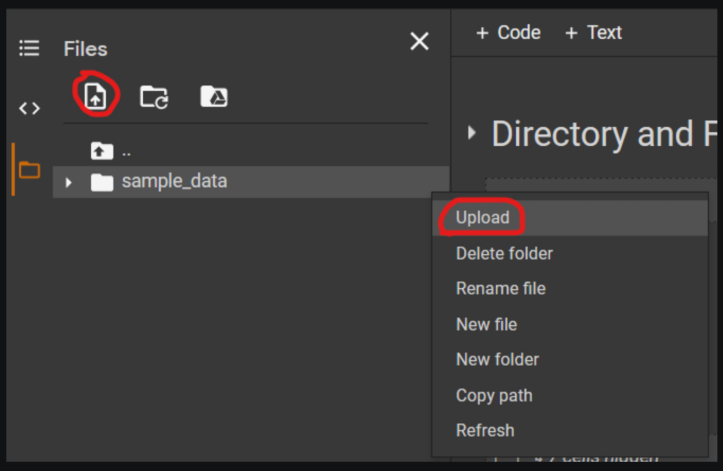
To do this:

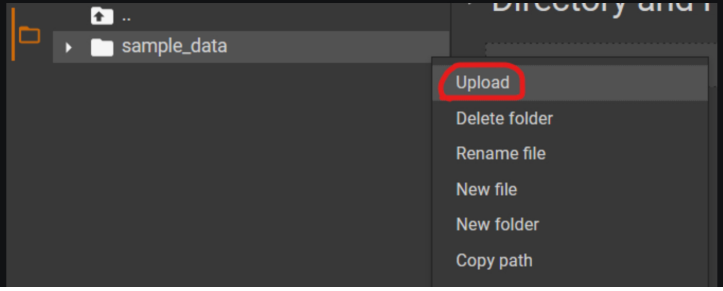
1. click on the file in the repository,
2. click on **View Raw**,
3. copy the URL of the raw file,
4. use this URL as the location of your file.

```
[4]  1 df3 = pd.read_json("News_Category_Dataset_v2.json", lines=True)
     2 df3.head()
```

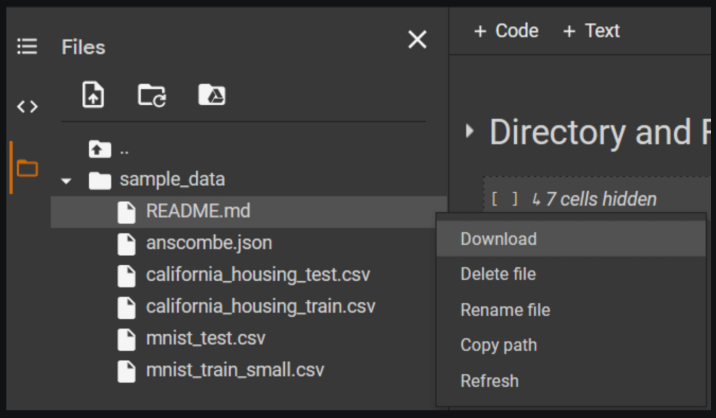
1. Click on the three dots visible when you hover above the directory

2. Select the “upload” option.

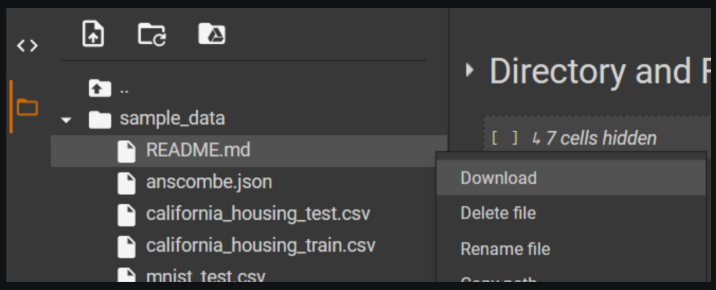




Click on the three dots which are visible while hovering above the filename, and select the “download” option.



Accessing local file system using Python code




```
1 uploaded = files.upload()
```

```
Browse... 2 files selected.
News_Category_Dataset_v2.json(application/json) - 738 bytes, last modified: n/a - 100% done
sample file.txt(text/plain) - 11 bytes, last modified: n/a - 100% done
Saving News_Category_Dataset_v2.json to News_Category_Dataset_v2.json
Saving sample file.txt to sample file.txt
```

The `uploaded` object is a dictionary having the filename and content as it’s key-value pairs:

```
1 uploaded
```

```
{'News_Category_Dataset_v2.json': b'{"category": "CRIME", "headline": "There Were
 'sample file.txt': b'sample text'}
```

Once the upload is complete, you can either read it as any other file from colab:

```
df4 = pd.read_json("News_Category_Dataset_v2.json", lines=True)
```

Or read it directly from the `uploaded` dict using the `io` library:

```
sample file.txt(text/plain) - 11 bytes, last modified: n/a - 100% done
Saving News_Category_Dataset_v2.json to News_Category_Dataset_v2.json
Saving sample file.txt to sample file.txt
```

The `uploaded` object is a dictionary having the filename and content as it's key-value pairs:

```
1 uploaded
```

```
{'News_Category_Dataset_v2.json': b'{"category": "CRIME", "headline": "There Were
 'sample file.txt': b'sample text'}
```

Accessing Google Drive from Google Colab

You can use the `drive` module from `google.colab` to mount your entire Google Drive to Colab by:

1. Executing the below code which will provide you with an authentication link

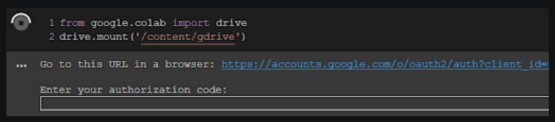
```
from google.colab import drive
drive.mount('/content/gdrive')
```

2. Open the link

3. Choose the Google account whose Drive you want to mount

4. Allow Google Drive Stream access to your Google Account

5. Copy the code displayed, paste it in the text box as shown below, and press Enter




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

2. Open the link

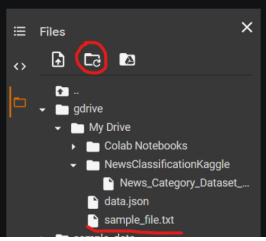
3. Choose the Google account whose Drive you want to mount

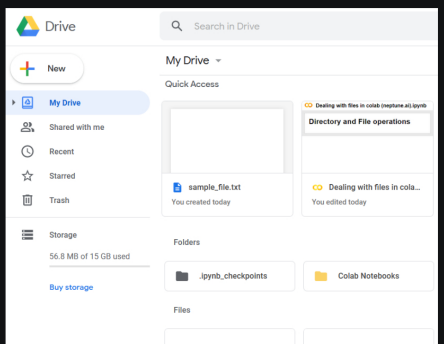
4. Allow Google Drive Stream access to your Google Account

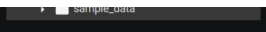
5. Copy the code displayed, paste it in the text box as shown below, and press Enter

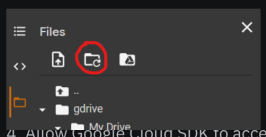

```
!touch "/content/gdrive/My Drive/sample_file.txt"
```

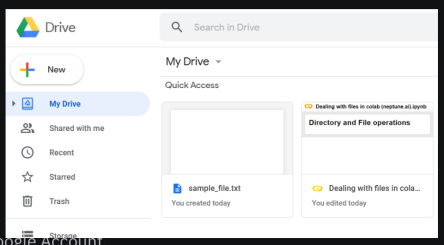
This will create a file in your Google Drive, and will be visible in the file-explorer pane once you refresh it:











4. Allow Google Cloud SDK to access your Google Account,

5. Finally copy the code displayed and paste it in the text box shown, and hit Enter.

```
1 from google.colab import auth
2 auth.authenticate_user()
```

```
...  Go to the following link in your browser:

        https://accounts.google.com/o/oauth2/auth?client_id=

     Enter verification code:
```

To interact with Google Sheets, you need to import the preinstalled `gspread` library. And to authorize `gspread` access to your Google account, you need the `GoogleCredentials` method from the preinstalled `oauth2client.client` library:

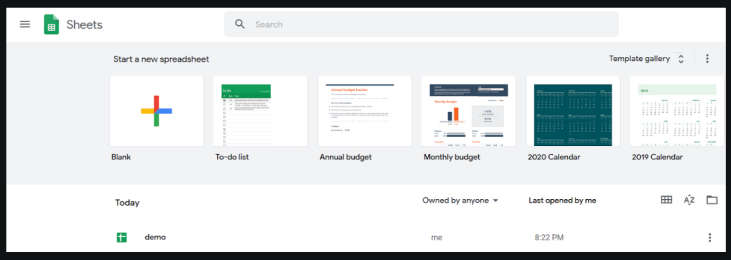
```
import gspread
from oauth2client.client import GoogleCredentials

gc = gspread.authorize(GoogleCredentials.get_application_default())
```


```
        https://accounts.google.com/o/oauth2/auth?client_id=

     Enter verification code:
```

To interact with Google Sheets, you need to import the preinstalled `gspread` library. And to authorize `gspread` access to your Google account, you need the `GoogleCredentials` method from the preinstalled `oauth2client.client` library:


```
import gspread
from oauth2client.client import GoogleCredentials

wb = gc.create('demo')
```

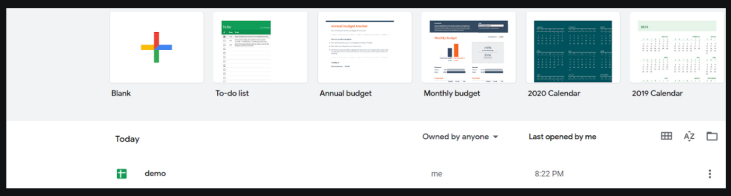
2. Once the workbook is created, you can view it in sheets.google.com.



3. To write values to the workbook, first open a worksheet:

```
ws = gc.open('demo').sheet1
```

4. Then select the cell(s) you want to write to:

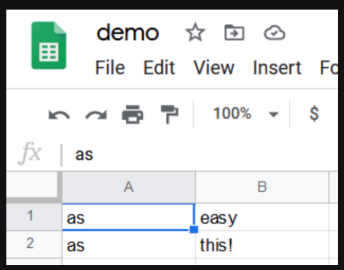


3. To write values to the workbook, first open a worksheet:

```
ws = gc.open('demo').sheet1
```

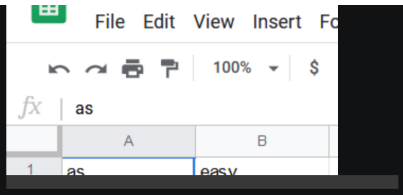
```
{'spreadsheetId': '10nlyR8oClsBYhfyMOeR8Hgc3E6CVnm5C2hoPvlR9IL0',
 'updatedCells': 4,
 'updatedColumns': 2,
 'updatedRange': 'Sheet1!A1:B2',
 'updatedRows': 2}
```

7. The changes will now be reflected in your Google Sheet.



Downloading data from a Google Sheet

7. The changes will now be reflected in your Google Sheet.



Accessing Google Cloud Storage (GCS) from Google Colab

You need to have a Google Cloud Project (GCP) to use GCS. You can create and access your GCS buckets in Colab via the preinstalled `gsutil` command-line utility.

1. First specify your project ID:

```
project_id = '<project_ID>'
```

2. To access GCS, you've to authenticate your Google account:

```
from google.colab import auth
auth.authenticate_user()
```

You need to have a Google Cloud Project (GCP) to use GCS. You can create and access your GCS buckets in Colab via the preinstalled `gsutil` command-line utility.

1. First specify your project ID:

```
project_id = '<project_ID>'
```

```
!gcloud config set project {project_id}
```

8. You can make a bucket using the make bucket (`mb`) command. GCP buckets must have a universally unique name, so use the preinstalled `uuid` library to generate a Universally Unique ID:

```
import uuid
bucket_name = f'sample-bucket-{uuid.uuid1()}'
!gsutil mb gs://{bucket_name}
```

9. Once the bucket is created, you can upload a file from your colab environment to it:

```
!gsutil cp /tmp/to_upload.txt gs://{bucket_name}/
```

10. Once the upload has finished, the file will be visible in the GCS browser for your project: https://console.cloud.google.com/storage/browser?project=**<project_id>**

```
!gsutil cp gs://{bucket_name}/{filename} {download_location}
```

```
!gsutil mb gs://{bucket_name}
```

9. Once the bucket is created, you can upload a file from your colab environment to it:

```
!gsutil cp /tmp/to_upload.txt gs://{bucket_name}/
```

10. Once the upload has finished, the file will be visible in the GCS browser for your project:
https://console.cloud.google.com/storage/browser?project=**<project_id>**

```
!gsutil cp gs://{bucket_name}/{filename} {download_location}
```

```
AWS Secret Access Key [None]: <your_secret_access_key>
Default region name [None]:
Default output format [None]:
```

1. Enter your `access_key` and `secret_access_key` in the text boxes, and press enter.

Then you can download any file from S3:

```
!aws s3 cp s3://{bucket_name} ./{download_location} --recursive --exclude "*" --include
{filepath_on_s3}
```

`filepath_on_s3` can point to a single file, or match multiple files using a pattern.

You will be notified once the download is complete, and the downloaded file(s) will be available in the location you specified to be used as you wish.

To upload a file, just reverse the source and destination arguments:

```
!aws s3 cp ./{upload_from} s3://{bucket_name} --recursive --exclude "*" --include
{file_to_upload}
```


```
!aws s3 cp s3://{bucket_name} ./{download_location} --recursive --exclude "*" --include
{filepath_on_s3}
```

`filepath_on_s3` can point to a single file, or match multiple files using a pattern.

You will be notified once the download is complete, and the downloaded file(s) will be available in the location you specified to be used as you wish.

To upload a file, just reverse the source and destination arguments:


```
!aws s3 cp ./{upload_from} s3://{bucket_name} --recursive --exclude "*" --include
{file_to_upload}
!mkdir ~/.kaggle #create the .kaggle folder in your root directory
!echo '<PASTE_CONTENTS_OF_KAGGLE_API_JSON>' > ~/.kaggle/kaggle.json #write kaggle API
credentials to kaggle.json
!chmod 600 ~/.kaggle/kaggle.json  # set permissions
!pip install kaggle #install the kaggle library
```

4. Once the kaggle.json file has been created in Colab, and the Kaggle library has been installed, you can search for a dataset using

for a dataset using

```
!kaggle datasets list -s {KEYWORD}
```

5. And then download the dataset using

```
!kaggle datasets download -d {DATASET NAME} -p /content/kaggle/
```

The dataset will be downloaded and will be available in the path specified (`/content/kaggle/` in this case).


```
!kaggle datasets list -s {KEYWORD}
```

5. And then download the dataset using

```
!kaggle datasets download -d {DATASET NAME} -p /content/kaggle/
```

The dataset will be downloaded and will be available in the path specified (`/content/kaggle/` in this case).

Accessing MySQL databases from Google Colab

```
query = f"SELECT * FROM {DATABASE}.{TABLE}"

import pandas as pd
df = pd.read_sql_query(query, engine)
```

Limitations of Google Colab while working with Files

One important caveat to remember while using Colab is that the files you upload to it won't be available forever. Colab is a temporary environment with an idle timeout of 90 minutes and an absolute timeout of 12 hours. This means that the runtime will disconnect if it has remained idle for 90 minutes, or if it has been in use for 12 hours. On disconnection, you lose all your variables, states, installed packages, and files and will be connected to an entirely new and clean environment on reconnecting.

Also, Colab has a disk space limitation of 108 GB, of which only 77 GB is available to the user. While this should be enough for most tasks, keep this in mind while working with larger datasets like image or video data.

Conclusion

Limitations of Google Colab while working with Files

One important caveat to remember while using Colab is that the files you upload to it won't be available forever. Colab is a temporary environment with an idle timeout of 90 minutes and an absolute timeout of 12 hours. This means that the runtime will disconnect if it has remained idle for 90 minutes, or if it has been in use for 12 hours. On disconnection, you lose all your variables, states, installed packages, and files and will be connected to an entirely new and clean environment on reconnecting.

Also, Colab has a disk space limitation of 108 GB, of which only 77 GB is available to the user. While this should be enough for most tasks, keep this in mind while working with larger datasets like image or video data.

Conclusion

Yes No

More about How to Deal With Files in Google Colab: Everything You Need to Know

Check out our **product resources** and **related articles** below:

Related article
How to Version and Organize ML Experiments That You Run in Google Colab
Read more →

Related article
ML Experiment Tracking: What It Is, Why It Matters, and How to Implement It
Read more →

Related article
How to Use Google Colab for Deep Learning – Complete Tutorial
Read more →

Product resource
How to track ML Model Training: Colab + neptune.ai Integration
Read more →

Explore more content topics:

Related article
How to Version and Organize ML Experiments That You Run in Google Colab
Read more →

Related article
ML Experiment Tracking: What It Is, Why It Matters, and How to Implement It
Read more →

Related article
How to Use Google Colab for Deep Learning – Complete Tutorial
Read more →

Product resource
How to track ML Model Training: Colab + neptune.ai Integration
Read more →

Explore more content topics:

Computer Vision | General | Machine Learning Tools | ML Model Development | MLOps | Natural Language Processing | Reinforcement Learning | Tabular Data | Time Series

Try Neptune for free | Check out the Docs

Take an interactive product tour →

month.

Your e-mail

Get Newsletter

Pricing

Deployment options

Roadmap

Service status

SOLUTIONS

ML Platform Engineer

Data Scientist

ML Engineer

ML Team Lead

Enterprise

Neptune Integrations

Neptune vs MLflow

Neptune vs TensorBoard

Other Comparisons

ML Experiment Tracking Tools

ML Metadata Stores

ML Model Registries

Podcast

MLOps Newsletter

Careers

Security portal and SOC 2

The Best MLOps Tools • MLOps at a Reasonable Scale • ML Metadata Store • MLOps: What, Why, and How • Experiment Tracking in Machine Learning