An Introduction to Google Colab! (2023)

Adrian Dolinay · Follow

15 min read · Aug 8, 2023

Introduction

Hi everyone! This article will go over a comprehensive overview of Google Colaboratory, also known as Google Colab. In order to run Google Colab you need to be signed into a Google account.

For those coding along, you can access the notebook and files on GitHub.

What is Google Colab

Colaboratory, or "Colab" for short, is a product from Google Research. Colab allows users to write and execute arbitrary Python and R code through the browser (this article will focus on Python). Colab is a hosted Jupyter notebook service that requires no setup to use, while providing access free of charge to computing resources including GPUs[1].

How Long Do Colab Runtimes Last?

A runtime is the length of your Google Colab session. Once a runtime shuts down all the data from the Colab page is erased (for example assigned Python variables, data output from code, models and data downloaded into the content folder).

Runtimes will time out if a user is idle. In the free-of-charge version of Colab, notebooks can run for at most 12 hours[1].

Disallowed Actions on Google Colab

Per Google Colab's Terms of Service "If Google reasonably determines that you violated any of the terms and conditions of this Agreement, your rights under this Section 4 will immediately terminate and Google may terminate your access to the Paid Service and/or your Google account without notice and without refund to you."[1]

- file hosting, media serving or other web service offerings not related to interactive compute with Colab
- downloading torrents or engaging in peer-to-peer file-sharing
- remote control such as SSH shells, remote desktops, remote UIs
- connecting to remote proxies
- mining cryptocurrency
- running denial-of-service attacks
- password cracking
- using multiple accounts to work around access or resource usage restrictions
- creating deepfakes

Check the Operating System and Python Version

Now that we have introduced Colab, let us get into the code! The first lines of code we will run are known as “shell commands”. A shell is a command line interface to your computer[2]. For Google Colab, the computer is a virtual machine we access through the Colab session. The way we can run a shell command within Colab is with an exclamation point, also known as a bang: `!`. First we will check the type of operating system Colab is running:

```
!cat /etc/os-release
```

This gives us an output of:

```
PRETTY_NAME="Ubuntu 22.04.2 LTS"
NAME="Ubuntu"
```

```
VERSION_ID="22.04"
VERSION="22.04.2 LTS (Jammy Jellyfish)"
VERSION_CODENAME=jammy
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-poli
UBUNTU_CODENAME=jammy
```

Google Colab is running Jammy Jellyfish, a specific Ubuntu Linux distribution. Next let us check what Python version is running within Colab:

```
!python --version
```

Output: `Python 3.10.6`

As of July 2023, Colab is running Python version 3.10.6. Periodically Colab will update the Python version it runs within its sessions. This can affect scripts as the syntax or package dependencies change over time.

The final shell command we will run displays all of the Python packages installed within the Google Colab session:

```
!pip freeze
```

The output is far too long to display in the article. The packages installed include popular Python modules such as: `pandas`, `matplotlib`, `tensorflow`, `keras`, `scipy` and many more.

Python Code: Mathematical Operations

Next we will get into Python! First we will assign two Python global variables. A global variable means that they can be used in any procedure or subroutine in the program[3]. Specifically the variables declared below are integers (whole numbers). We can run mathematical operations on the variables: addition, subtraction, multiplication, division and modulo.

```
#assigning variables
num1 = 10
num2 = 5

#addition
print(num1 + num2)
#subtraction
print(num1 - num2)
#multiplication
print(num1 * num2)
#division
print(num1 / num2)
#modulus
print(num1 % num2)
```

Output:

```
15
5
50
2.0
0
```

Displaying an Imported Image

We can stylize a Colab notebook by embedding images into it. We will import a module in order to do this, then we will upload the image file into Google Colab. The `files.upload()` function will prompt you to upload a file into the `content` directory (folder) within the Colab session. Once the image is uploaded, the `display.Image` function is run to embed the image into the notebook.

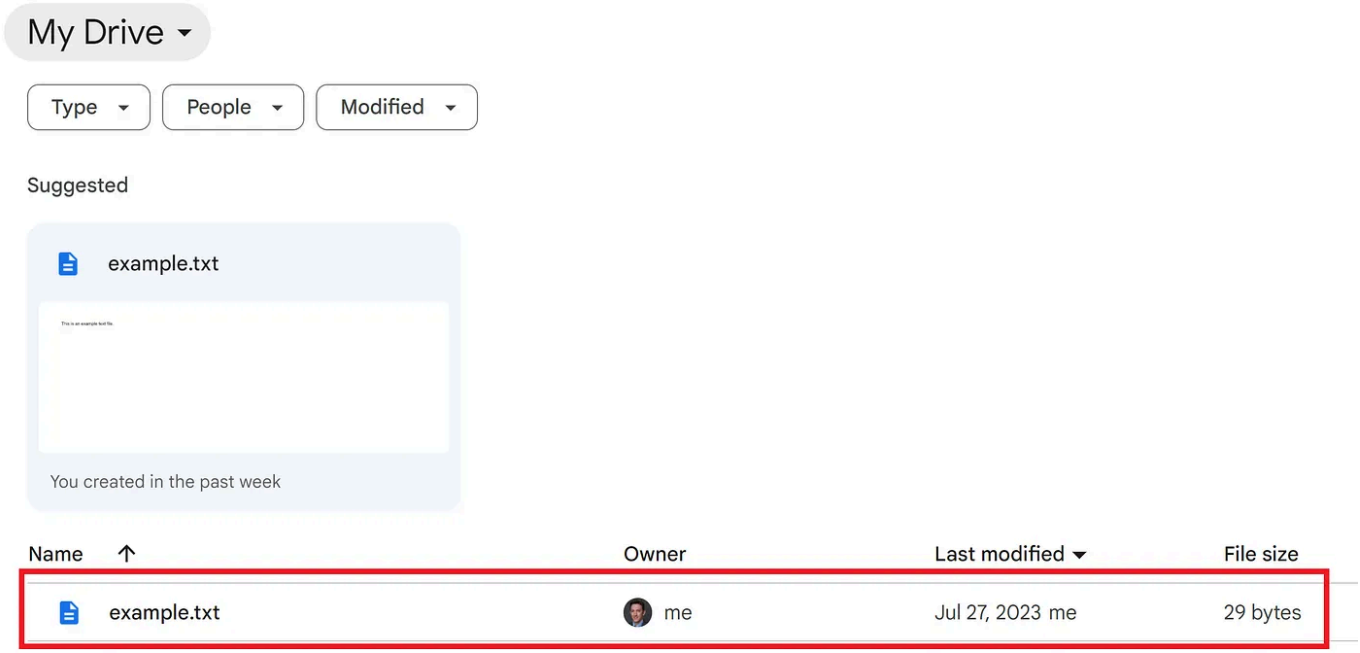
```
from google.colab import files
from IPython import display
files.upload()
```

```
display.Image('/content/sunflower.jpg')
```

We have our field of flowers displayed!

Connecting to your Google Drive

Google Colab can access files saved in a users Google Drive, making it easy to save notebooks to your drive or to read files into your Colab session. Within my Drive I need to access the text file highlighted below and read it into a Python string within my Colab session.

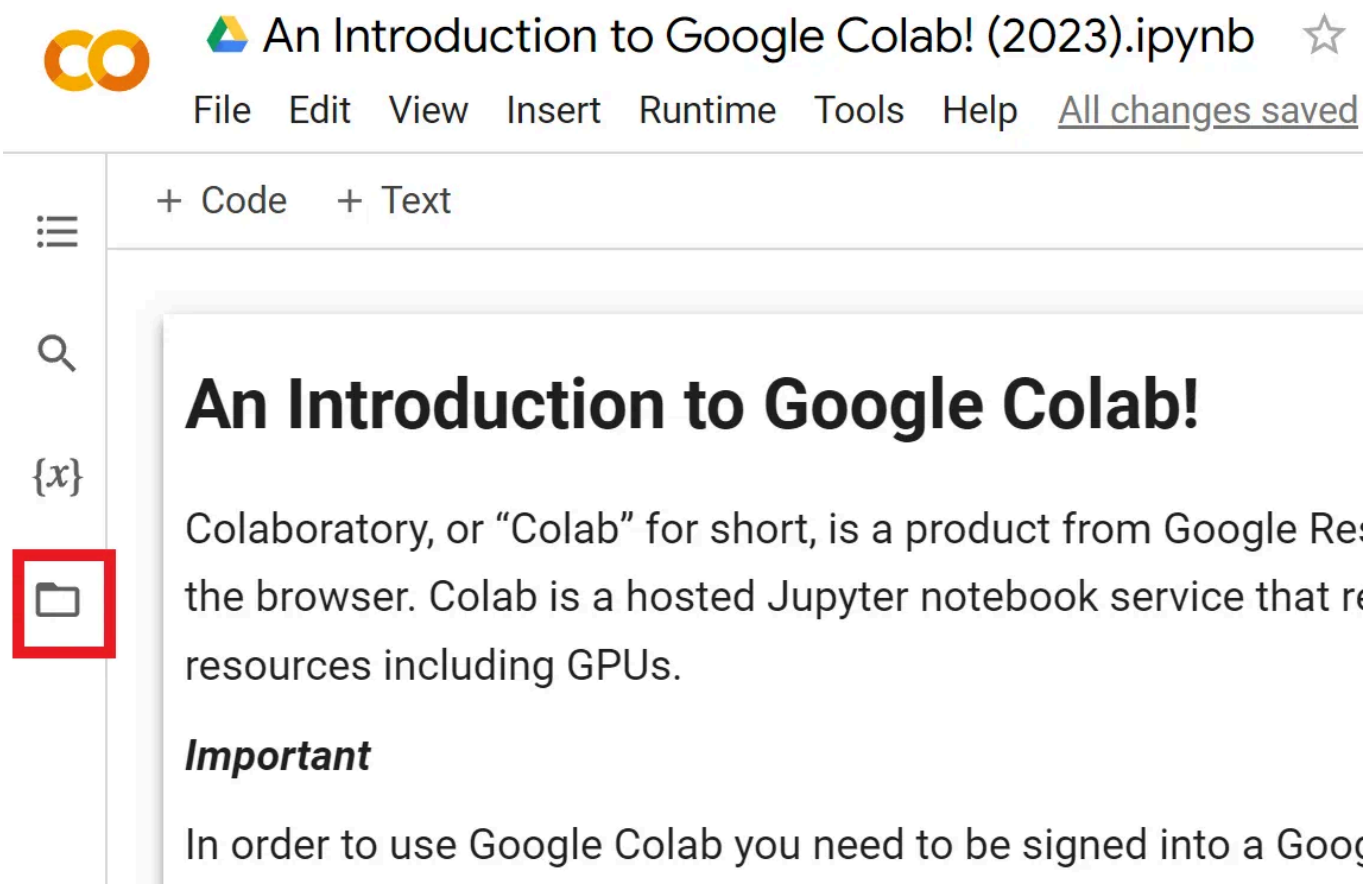


There are two methods to connect to the drive: selecting the “Mount Drive” icon or running a code block to connect to Google Drive.

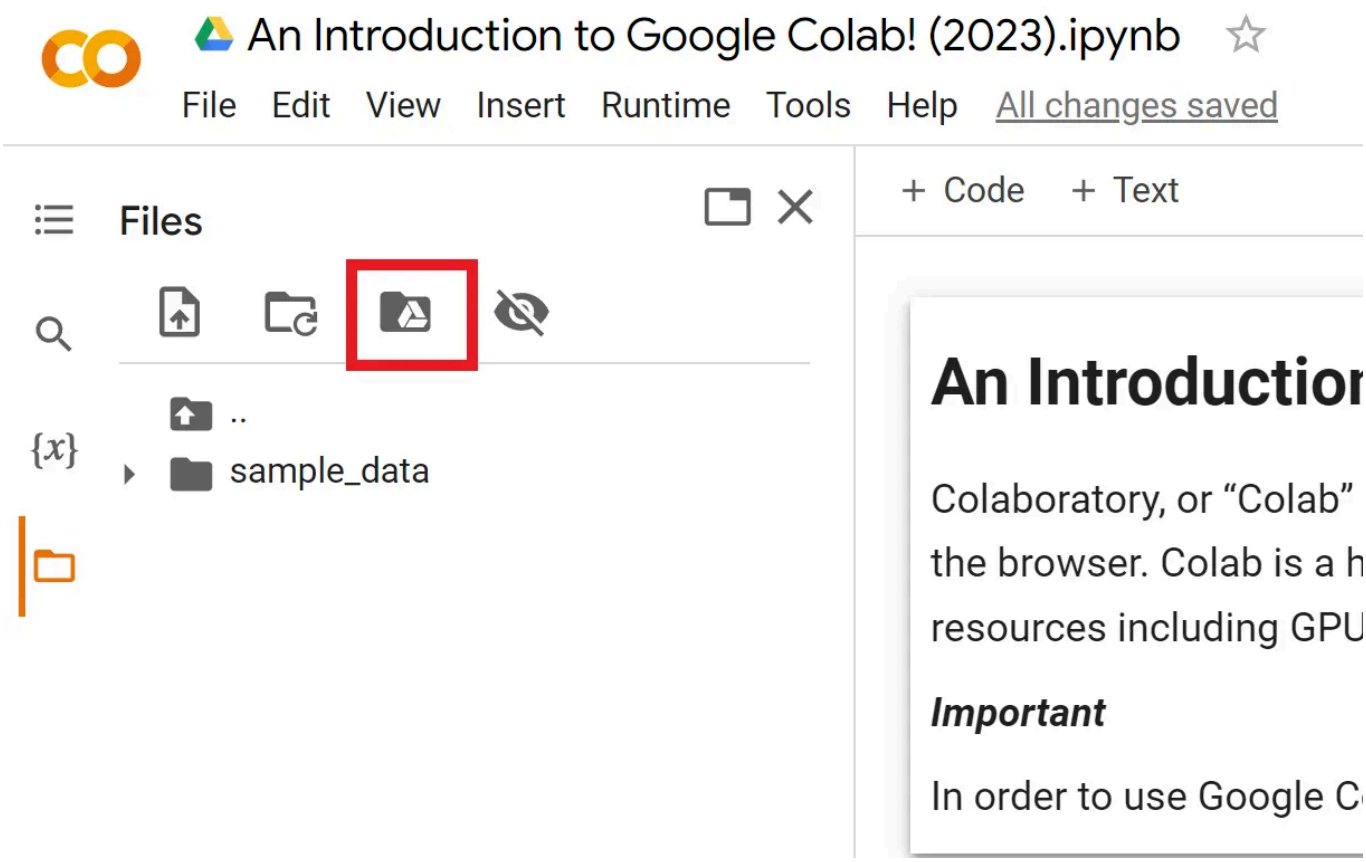
Option 1: Mount Drive Icon

One method for accessing your Google Drive is with the “Mount Drive” icon. **Note that the Google Drive mount button option only works if you are the only editor of the notebook, meaning it cannot be shared across multiple users.**

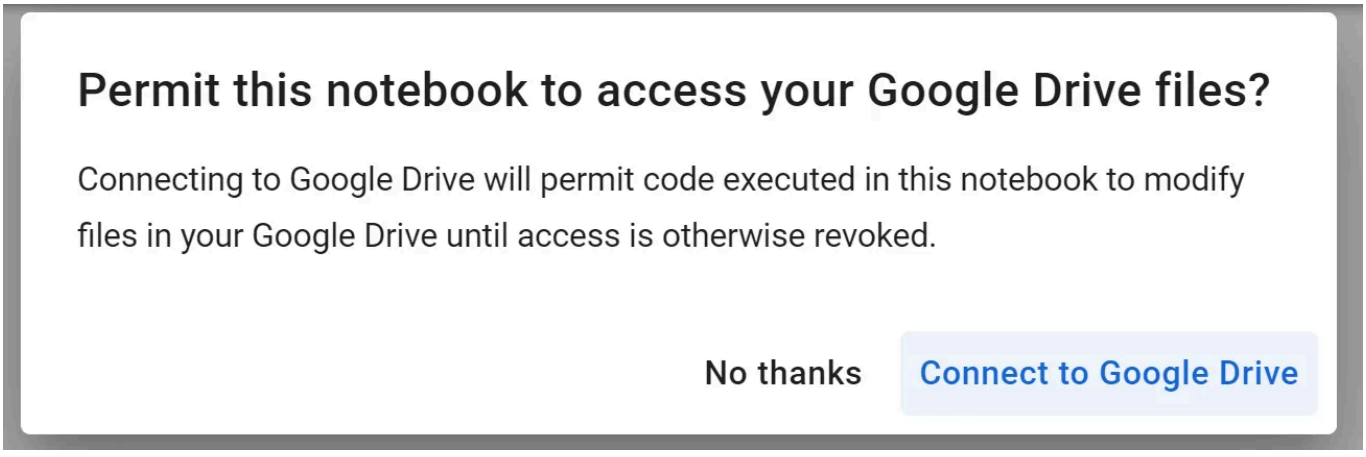
First select the “Files” icon within Colab.



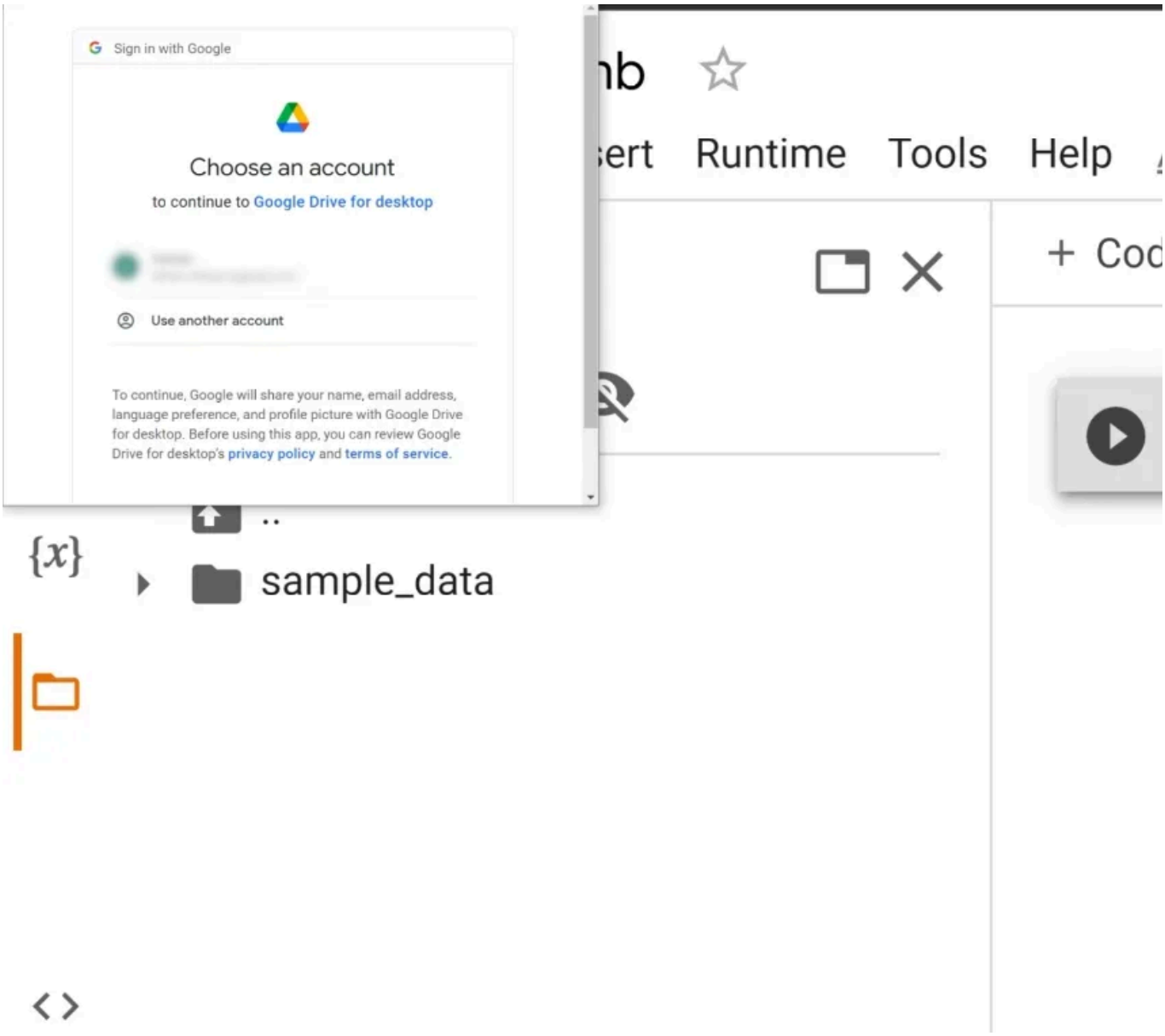
Next select the "Mount Drive" icon. **Note that if the notebook is shared Colab will create a code block for you to mount the drive.**



The banner below asks if you want to give permission for the Colab notebook to access your drive. The prompt states that the notebook will gain read/write permissions for the files in the drive you are connecting, meaning you can delete and edit files.

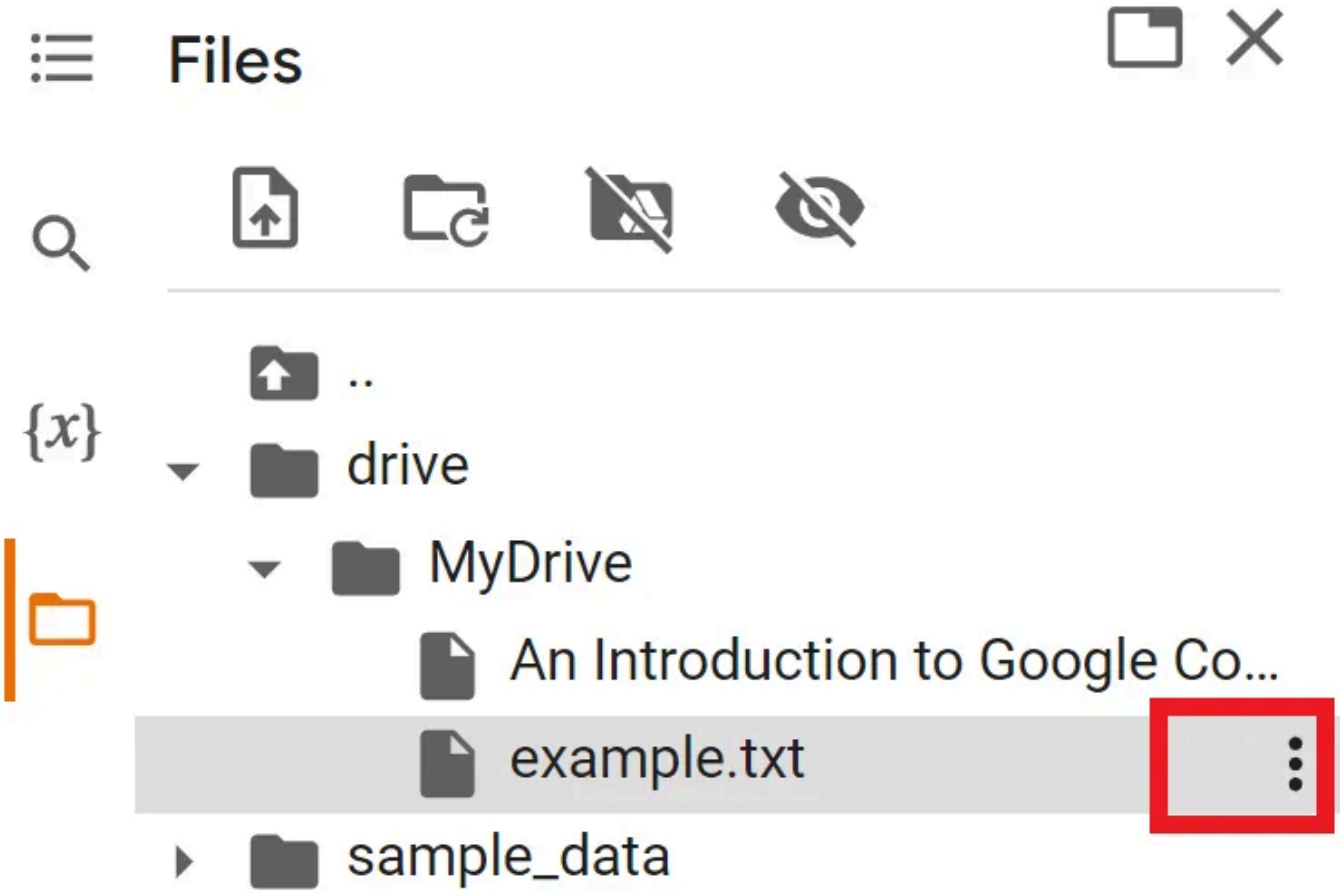


If you connect for the first time you will receive the notice below. The prompt confirms which account you are trying to connect as well as giving you a data sharing notice. Save the notebook after you connect the notebook. Every time you access the notebook and connect it to a Colab session the notebook will automatically mount to your Drive.

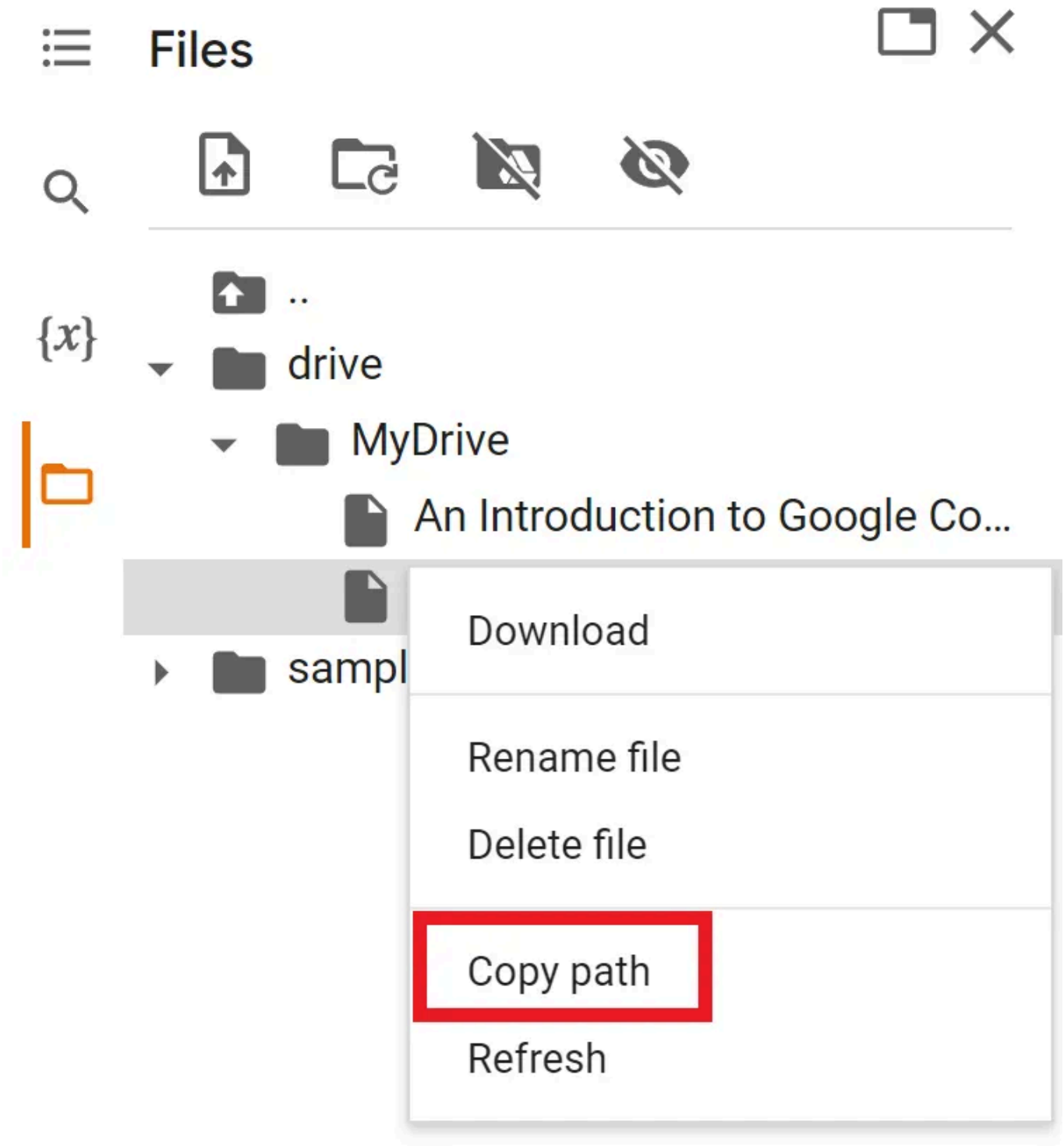


Once you are connected you will see a “drive” folder in your files section. The next folder is “MyDrive” and we can see within that folder we have two

files, the text file and the Colab notebook. Click the vertical 3-dot icon highlighted below.



Click the “Copy path” option highlighted below.



Finally we will read the text into a Python string with the code block below:

```
#opens a text file and saves the contents into a Python string
with open(file='/content/drive/MyDrive/example.txt',mode='r') as text_file:
  text = text_file.read()
```

```
print(text)
```

Output:

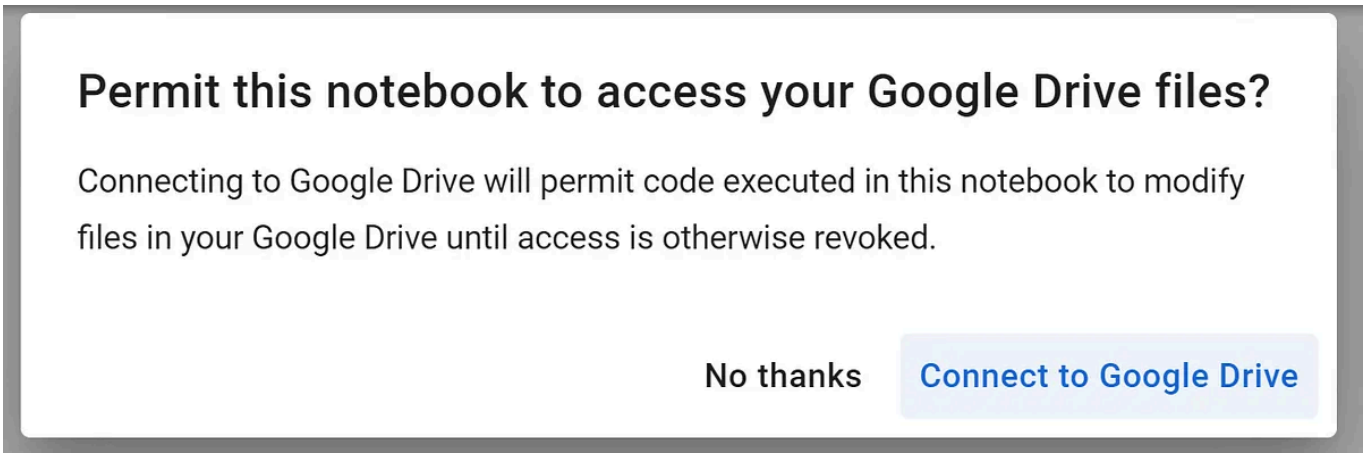
```
This is an example text file.
```

Option 2: Mounting Drive with Python Code

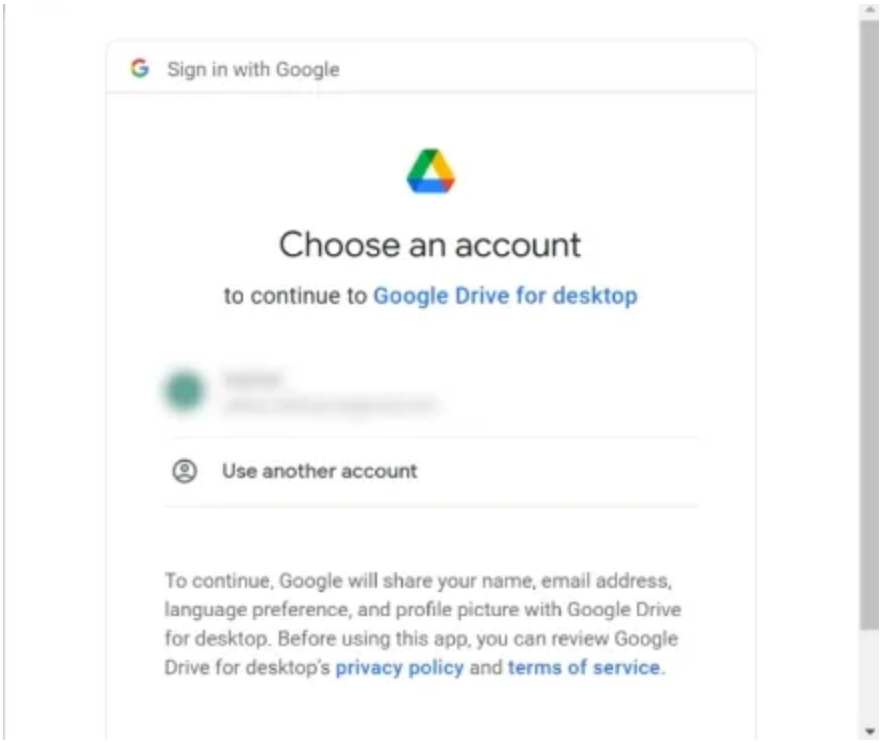
Another option is to run the below code block to mount a drive:

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

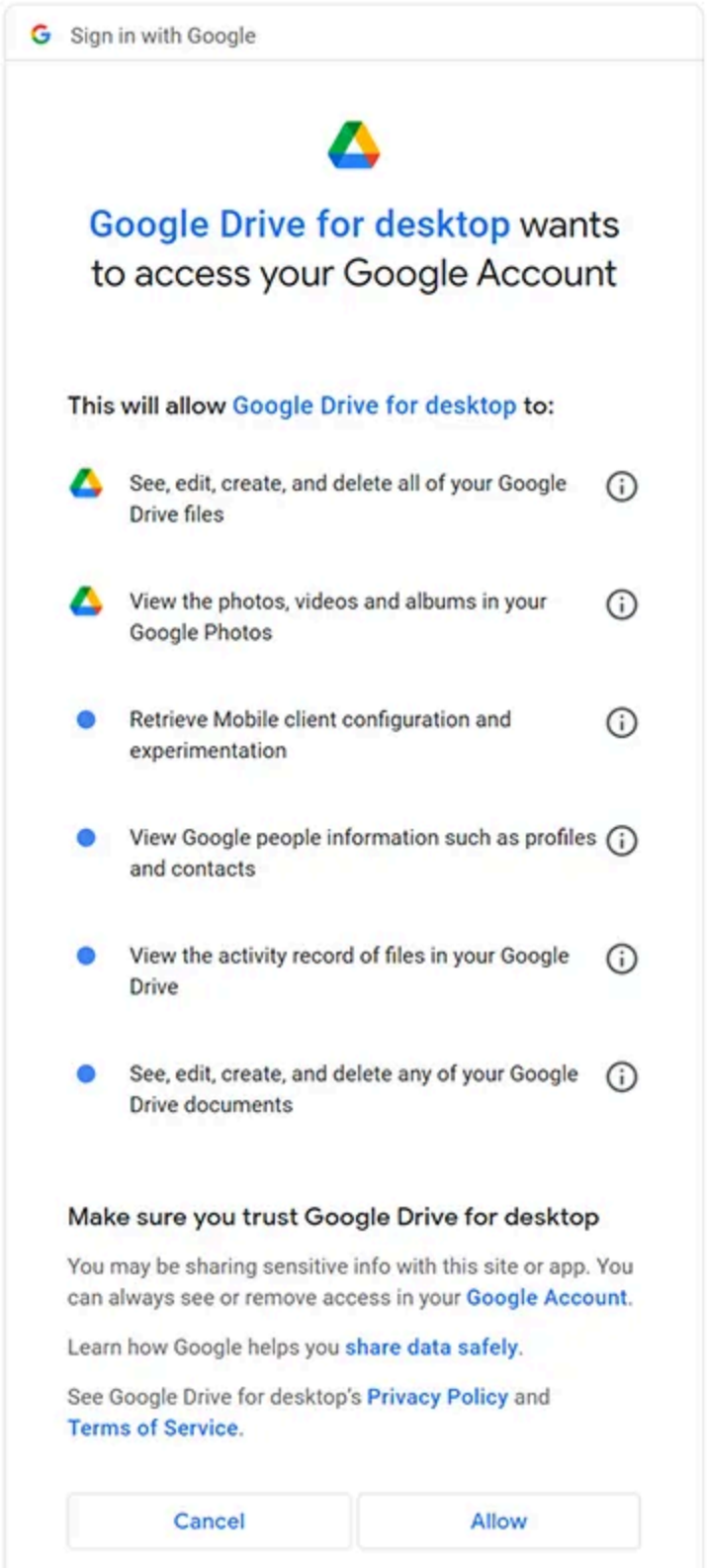
Once the above code block is run you will receive the prompt below:



Google will send another prompt asking which account you want to connect:



Google gives another warning stating that giving the notebook read/write permission will enable users of the notebook to delete files:



Downloading and Reading a CSV/Excel file into a pandas DataFrame

Python is a popular tool for reading data saved in CSV or Excel files. pandas is a go-to module to work with various data set types. First we will import pandas into our session:

```
import pandas as pd
```

Next we will download data on Meteorite landings from NASA's website. We will use `curl` in the command line to download the data into our `content` directory. For curl we need to provide the URL address for our data set then we can name the CSV file:

```
!curl https://data.nasa.gov/api/views/gh4g-9sfh/rows.csv?accessType=DOWNLOAD > m
```

Using pandas we can read the data into a pandas DataFrame. A DataFrame is class that stores data in a tabular format, similar to an Excel spreadsheet. The data in the rows and columns of a DataFrame can be heterogeneous, meaning you can have various data types in the cells: integers, floating point numbers, strings, lists, etc. Let us read in the meteorite CSV sheet into a DataFrame. First we will name our DataFrame (conventionally `df` is used), then we will call the `read_csv` function from pandas to save the data into a DataFrame:

```
df = pd.read_csv('meteors.csv')
```

And we have created our DataFrame! Next we will look at the first five rows of the DataFrame using the `head` function. At the top of the column we have our column names; these identify the type of data contained in the rows below. On the far left side we have the default index which ranges from 0 to the *n* number of rows in the DataFrame. The rows contain the data, we can see we have string values (columns: name, nametype, recclass, fall), integers (column: id), floating point numbers (columns: mass (g), year, reclat, relong) and tuples (column: GeoLocation).

	name	id	nametype	recclass	mass (g)	fall	year	reclat	reclong	GeoLocation
0	Aachen	1	Valid	L5	21.0	Fell	1880.0	50.77500	6.08333	(50.775, 6.08333)
1	Aarhus	2	Valid	H6	720.0	Fell	1951.0	56.18333	10.23333	(56.18333, 10.23333)
2	Abee	6	Valid	EH4	107000.0	Fell	1952.0	54.21667	-113.00000	(54.21667, -113.0)
3	Acapulco	10	Valid	Acapulcoite	1914.0	Fell	1976.0	16.88333	-99.90000	(16.88333, -99.9)
4	Achiras	370	Valid	L6	780.0	Fell	1902.0	-33.16667	-64.95000	(-33.16667, -64.95)

Google Colab includes useful features to turn a DataFrame into an interactive table (wand icon) and gives suggested graphs for your data (graph icon).

Turning the DataFrame into an interactive table results in the output below. The table can be sorted, the number of rows per page can be changed, the data can be copied as a table and the data can be filtered.

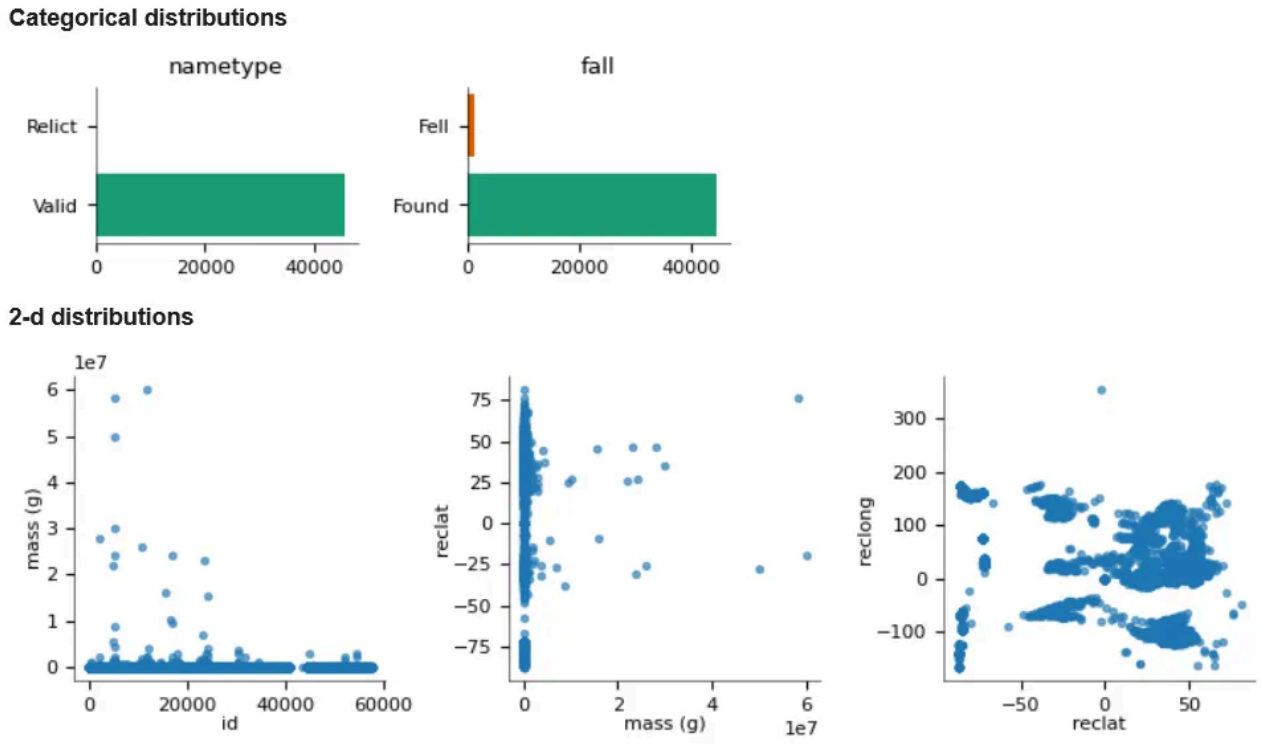
1 to 10 of 20000 entries Filter

index	name	id	nametype	recclass	mass (g)	fall	year	reclat	reclong	GeoLocation
0	Aachen	1	Valid	L5	21.0	Fell	1880.0	50.775	6.08333	(50.775, 6.08333)
1	Aarhus	2	Valid	H6	720.0	Fell	1951.0	56.18333	10.23333	(56.18333, 10.23333)
2	Abee	6	Valid	EH4	107000.0	Fell	1952.0	54.21667	-113.0	(54.21667, -113.0)
3	Acapulco	10	Valid	Acapulcoite	1914.0	Fell	1976.0	16.88333	-99.9	(16.88333, -99.9)
4	Achiras	370	Valid	L6	780.0	Fell	1902.0	-33.16667	-64.95	(-33.16667, -64.95)
5	Adhi Kot	379	Valid	EH4	4239.0	Fell	1919.0	32.1	71.8	(32.1, 71.8)
6	Adzhi-Bogdo (stone)	390	Valid	LL3-6	910.0	Fell	1949.0	44.83333	95.16667	(44.83333, 95.16667)
7	Agen	392	Valid	H5	30000.0	Fell	1814.0	44.21667	0.61667	(44.21667, 0.61667)
8	Aguada	398	Valid	L6	1620.0	Fell	1930.0	-31.6	-65.23333	(-31.6, -65.23333)
9	Aguila Blanca	417	Valid	L	1440.0	Fell	1920.0	-30.86667	-64.55	(-30.86667, -64.55)

Show 10 per page

1 2 10 100 1000 1900 1990 2000

Selecting the “Suggested graphs” option gives the below output. The types of graphs created can include: distributions (including 2d and categorical 2d), scatter plots, bar charts and heat matrices.



Display Interactive Plot

A cool feature within Colab is displaying interactive plots. Plotly allows us to create various types of charts that a user can interact with.

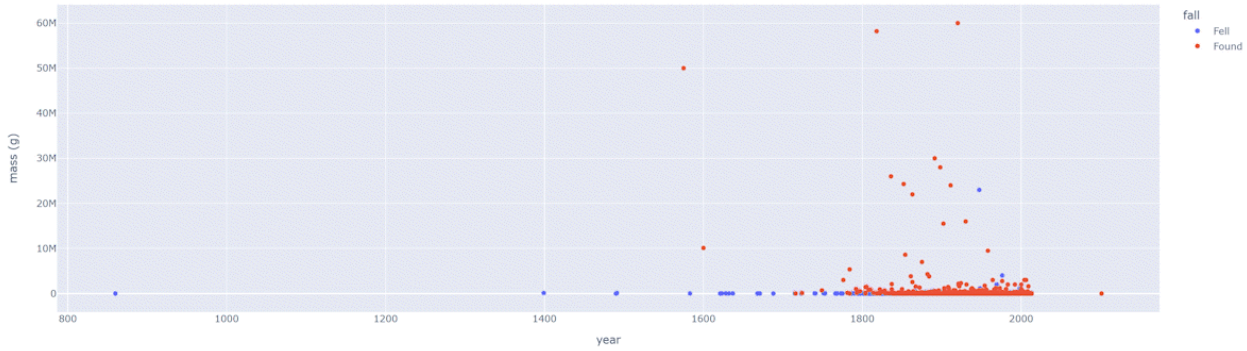
We will display the meteor data we imported into a pandas DataFrame. First we will import Plotly:

```
import plotly.express as px
```

We will graph a scatter plot using the data we imported into a pandas DataFrame. On the y-axis we will plot the estimated mass of the meteor, the x-axis is the year the meteor fell into Earth and we will color the meteor based on wether it was observed falling or found somewhere on the ground.

```
fig = px.scatter(df, x='year', y='mass (g)',color='fall')
fig.show()
```

From the GIF below, we can see how interactive the Plotly graph is. You can hover over individual data points to get more information, zoom into different parts of the graph, download the graph as an image amongst many other features.



Installing Python Packages

Even though Google Colab provides many different Python packages, you may run into a situation where a Python module you want to work with is not installed in your Colab session. In this case we can use `pip` to install and work with a Python package. The Python package we will install is `word_forms`, a package which generates all possible forms of an English word[4]. First we will install the package with `pip`. In some cases after you install a package you will be prompted to restart the Colab session runtime (can be toggled with CTRL + M .).

```
!pip install word_forms
```

Now `word_forms` can be imported. We will import the `get_word_forms` function from the module:

```
from word_forms.word_forms import get_word_forms
```

Let us run the function on the word “happy”:

```
#"r" stands for adverb, "a" for adjective, "n" for noun and "v" for verb
get_word_forms('happy')
```

Output:

```
{'n': {'happiness', 'happinesses'},
 'a': {'happy'},
 'v': set(),
 'r': {'happily'}}
```

Once a Google Colab session shutdowns all installed packages are removed, meaning any non-base packages need to be reinstalled.

Updating a Python Package

Python packages are frequently updated and can feature new tools for users and more optimized code. Within Colab packages can be updated using `pip`. Running the command line code below, we can see the version of pandas that Colab is running:

```
#check out the current module version
!pip show pandas
```

Output:

```
Name: pandas
Version: 1.5.3
Summary: Powerful data structures for data analysis, time series, and statistics
Home-page: https://pandas.pydata.org
Author: The Pandas Development Team
Author-email: pandas-dev@python.org
License: BSD-3-Clause
Location: /usr/local/lib/python3.10/dist-packages
Requires: numpy, python-dateutil, pytz
Required-by: altair, arviz, bokeh, cmdstanpy, cufflinks, datascience, db-dtypes,
```

Currently the pandas version in Colab is 1.5.3, but the newest version as of July 2023 is version 2.0.3. Running pip we update to the newest panda version:

```
#upgrade specified package to the newest available version
!pip install --upgrade pandas
```

Once pandas is updated, we can check that we are running the newest version:

```
pip show pandas
```

Truncated output:

```
Name: pandas
```

```
Version: 2.0.3
Summary: Powerful data structures for data analysis, time series, and statistics
```

Once a Google Colab session shutdowns all updated packages revert to their base Colab versions. The packages can be updated again by running `pip --upgrade` in the new session.

Installing Packages with Ubuntu command-line interface (Linux)

Packages can be installed using the Ubuntu, the operating system Colab runs on. This can be useful for packages that are run in the command line rather than with Python.

We will explain a few of the Linux commands before getting into the code. `sudo` gives administrative privileges and `apt-get update` downloads the information for all packages listed in the sources file[5]. After the package information is update we can install a given package:

```
#updates the package lists
!sudo apt-get update
!sudo apt install boxes
```

Here we install Boxes, a command line program which draws ASCII art boxes[6]. With this fun program let us draw out a dog with a sign that says “I love coding!”

```
!echo 'I love coding!' | boxes -d dog
```

And we have our output of a dog!

```
           __   _,--="=--,_   __
          /  \."    .-.    "./  \
         /  ,/  _   : :   _  \/` \
         \  `| /o\  :_:  /o\ |\__/
          `-'| :="~` _ `~"=: |
             \`     (_)     `/ jgs
     .-"-.   \      |      /   .-"-.
.---{     }--|  /,.-'-.,\  |--{     }---.
 )  (_)_)_)  \_/`~-===-~`\_/  (_(_(_)  (
(  I love coding!                       )
 )                                     (
'-------------------------------------'
```

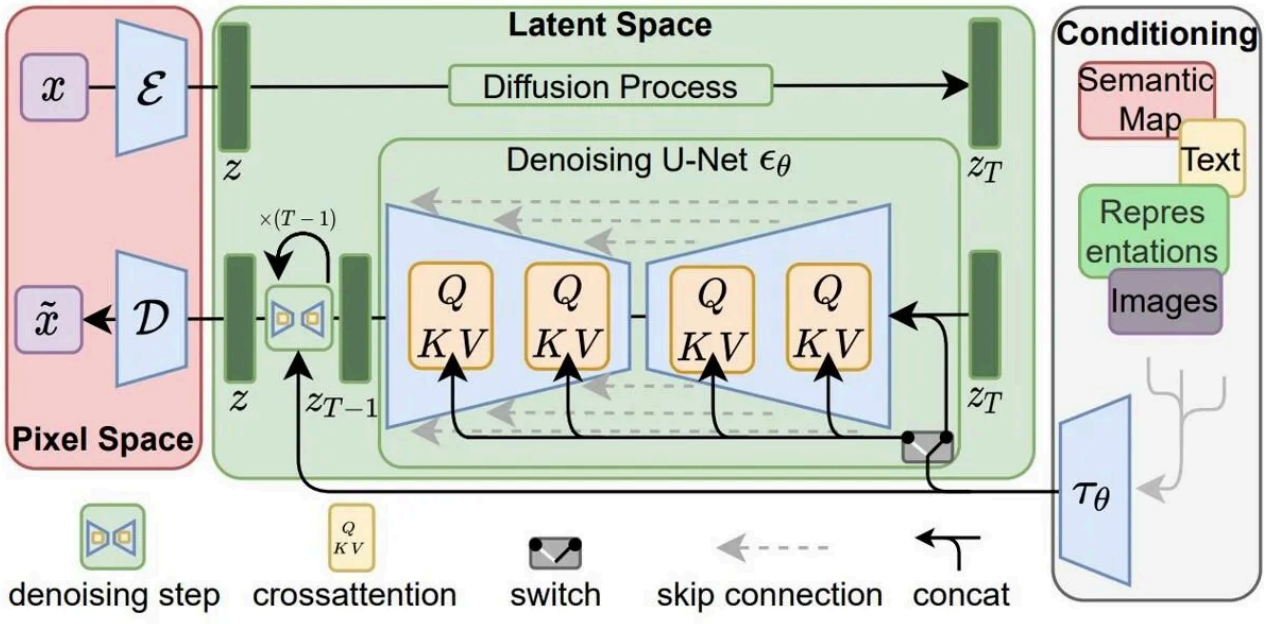
GPU Runtimes

In this portion we will run a machine learning model using a GPU runtime. A graphics processing unit (GPU) is specialized hardware that performs certain computations much faster than a traditional computer's central processing unit (CPU)[7]. Python machine learning modules such as TensorFlow and Torch utilize GPUs.

The types of GPUs that are available in Colab vary over time. This is necessary for Colab to be able to provide access to these resources free of charge[1]. **If you are working with a non-GPU runtime and you switch to a GPU runtime you will lose all your data.**

We will use a GPU runtime to execute a stable diffusion model. Stable diffusion models take text prompts and generate images. The models are trained on text-image pairs. Given the size of the training sets, the images are compressed using a method called an autoencoder (we will not go into

this in much depth, if you want to learn more check out this article). After a model is trained it can output novel images.



Source: Stable Diffusion Presentation by Binxu Wang and John Vastola

We will need to download multiple packages in order to run the SDXL model. We can download the model through `transformers`, a Python package that provides API access to the machine learning site HuggingFace.

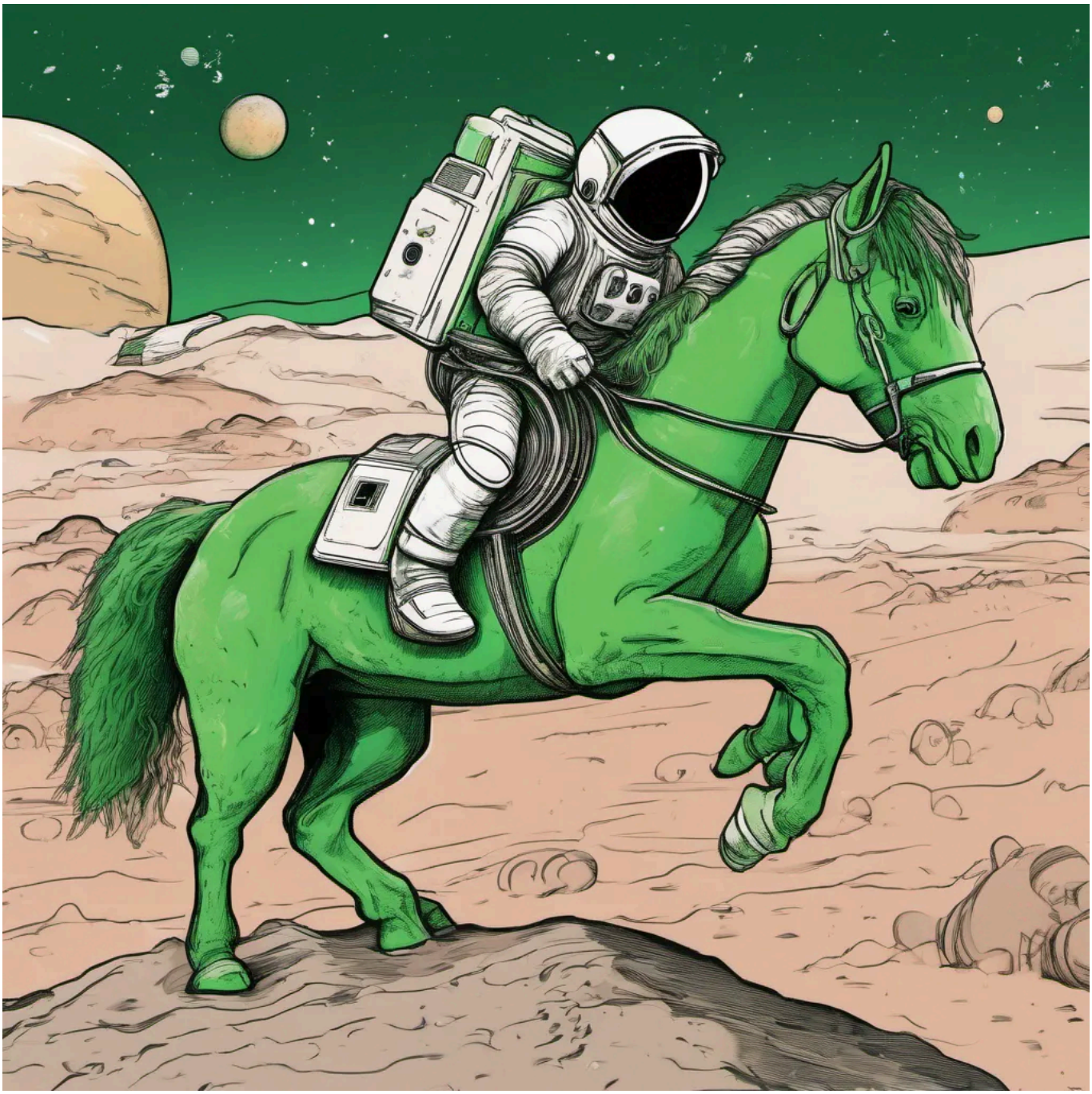
```
#Python packages can be installed with pip, make sure to include an exclamation
!pip install transformers
!pip install diffusers --upgrade
!pip install invisible_watermark transformers accelerate safetensors
```

After downloading all the packages, we import the relevant modules and we can generate an image with a machine learning model in just four lines of

Python code!

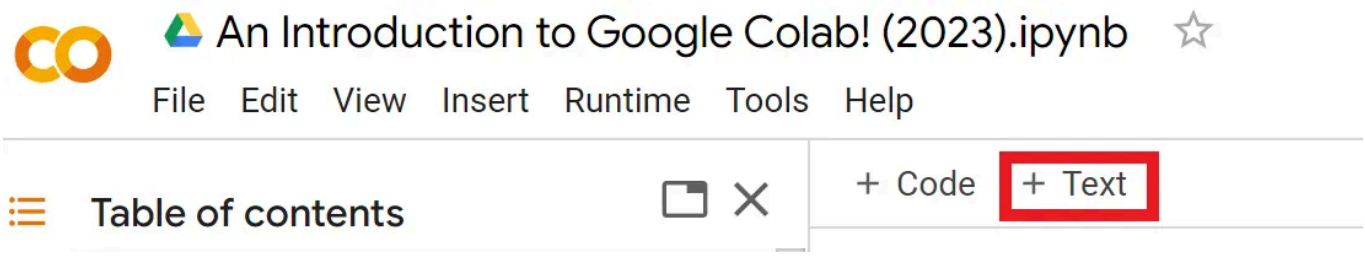
```
from diffusers import DiffusionPipeline
import torch

pipe = DiffusionPipeline.from_pretrained("stabilityai/stable-diffusion-xl-base-1
pipe.to("cuda")

# if using torch < 2.0
# pipe.enable_xformers_memory_efficient_attention()

prompt = "An astronaut riding a green horse"

images = pipe(prompt=prompt).images[0]
```

Output:



Markdown

Markdown is a lightweight markup language that you can use to add formatting elements to plaintext text documents[9]. Markdown is supported within Google Colab. To create a Markdown cell, you can go to the tabs at the top, select "**Insert**" then select "**Text cell**". A Markdown cell can also be created by selecting the "Text" button at the top of the notebook.



Headings

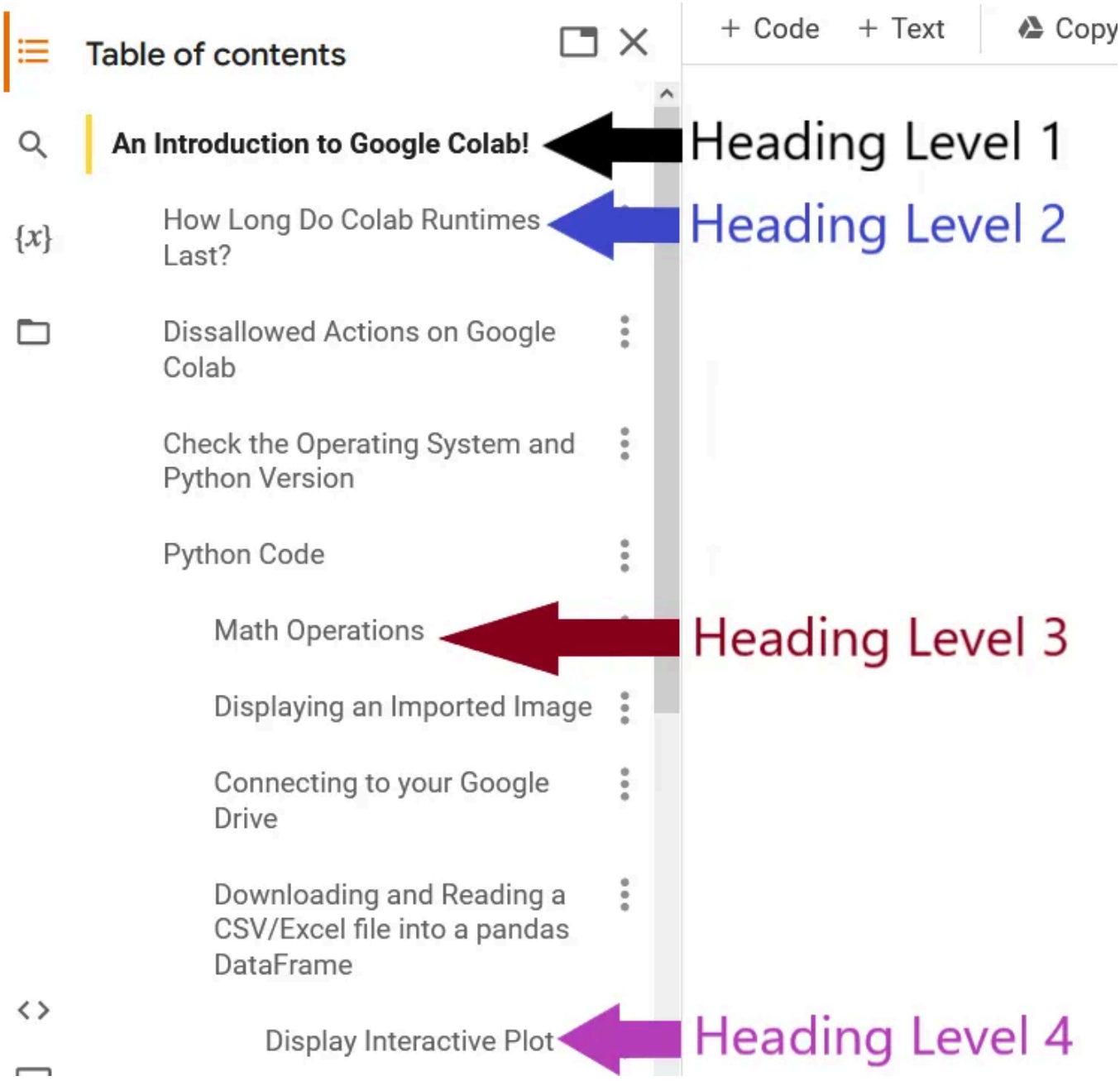
Within the “Table of Contents” in Colab, a notebook can be broken out into different sections. The sections are set by Markdown heading levels set with a hashtag (#).

Markdown	HTML	Rendered Output
`# Heading level 1`	`<h1>Heading level 1</h1>`	Heading level 1
`## Heading level 2`	`<h2>Heading level 2</h2>`	Heading level 2
`### Heading level 3`	`<h3>Heading level 3</h3>`	Heading level 3
`#### Heading level 4`	`<h4>Heading level 4</h4>`	Heading level 4
`##### Heading level 5`	`<h5>Heading level 5</h5>`	Heading level 5
`###### Heading level 6`	`<h6>Heading level 6</h6>`	Heading level 6

Source: Basic Syntax from Markdown.org

We can see from the table above a single hashtag sets the largest heading level. Within Colab, different heading levels will be displayed under the

"Table of Contents". From the image below, we can see that heading level 1 is not indented, while heading levels 2–4 are indented. This sectioning makes it easy for users to navigate through a notebook.



Bolding and Italicizing

Text within Markdown can be formatted with bolding and italicizing. Text is **bolded** with the text in between two asterisks (** **) or with a double

underscore (__ __).

Text is *italicized* with the text in between singe asterisks (* *) or between underscores (_ _).

Text is ***bolded and italicized*** with the text in between three asterisks (*** ***).

Code Blocks

Text can also be stylized as code blocks by putting text between backticks (` `):

```
import random
```

For multi-line code blocks place three backticks at the beginning of the block and three at the end:

```
```

```
import random
random.randint(a=0,b=100)
```

```
```

LaTeX

The final section we will focus on is rendering equations in LaTeX. LaTeX is a typesetting system designed for the production of technical and scientific

documentation[10]. We will go over three different equations that will be rendered: a fraction, integration and linear algebra equation.

LaTeX can be rendered by putting text between dollar signs ($ $). Our first equation we will render is a fraction, where x is a numerator, 2 is the denominator and 10 is the quotient. To create a fraction, we need to use the fraction command written as \frac{}{}. In the first curly brace we place the numerator and in the second the denominator. Outside of the fraction command we put in an equal sign and the quotient. We also want to increase the size of the equation by wrapping it with a large command:

$\Large{\frac{x}{2}=10}$

$$\Large{\frac{x}{2}=10}$$

More complex equations can also be easily rendered. Let us render an integral over an interval [a,b] for the function $3x^2$. To display an integral symbol we will us the following command \int_{}^{}. The curly brace following the underscore is the lower limit while the curly brace following the caret is the upper limit. To square the variable x we also place a caret next to it:

$\int_{a}^{b} 3x^2 \,dx$

$$\int_a^b 3x^2\,dx$$

If we have two vectors where we want to find the dot product, we can call the vec command with the cdot symbol between the two vectors.

$\vec{p} \cdot \vec{q}$

$$\vec{p} \cdot \vec{q}$$

References and Additional Information

The full notebook can be viewed on GitHub.

Connect

Thanks for checking out the article! Feel free to connect with Adrian on YouTube, LinkedIn, X, GitHub and Odysee. Happy coding!

References

1. Google. (2023). *Colaboratory Frequently asked questions*. Retrieved from https://research.google.com/colaboratory/intl/en-GB/faq.html

2. Chan, C.-S. (2022). *Python — an overview*. Retrieved from Iowa State University:

https://cschan.arch.iastate.edu/TU/Lect20_Python_function_calls.pdf

3. Luke, T. (2023). *The Command Shell*. Retrieved from Statistics and Actuarial Science from the University of Iowa: https://homepage.divms.uiowa.edu/~luke/classes/STAT4580-2023/shell.html#:~:text=The%20shell%20is%20a%20command,and%20directories%20(a.k.a%20folders)%3B

4. Chakravorty, D. (2021). *word_forms*. Retrieved from pypi.org: https://pypi.org/project/word-forms/#description

5. Kariuki, D. (2022, June). *Understanding the Linux Apt-Get Update Command*. Retrieved from linuxhint.com: https://linuxhint.com/linux-apt-get-update-command/

6. Jensen, T. (2021). *Boxes*. Retrieved from thomasjensen.com: https://boxes.thomasjensen.com/

7. University of Washing Information Technology. (2023). *GPUs for Machine Learning*. Retrieved from uw.edu: https://itconnect.uw.edu/guides-by-topic/research/research-computing/gpus-for-machine-learning/

8. Wang, B., & Vastola, J. (2022, November 1). *Stable Diffusion*. Retrieved from harvard.edu: https://scholar.harvard.edu/binxuw/classes/machine-learning-scratch/materials/stable-diffusion-scratch

9. MarkdownGuide.org. (2023). *Getting Started: An overview of Markdown, how it works, and what you can do with it.* Retrieved from MarkdownGuide.org: https://www.markdownguide.org/getting-started/

10. LaTeX project. (2023). *An introduction to LaTeX*. Retrieved from latex-project.org/: https://www.latex-project.org/

Python Google Colab Google Colaboratory Markdown Data Science

Written by Adrian Dolinay

9 Followers · 0 Following

Follow

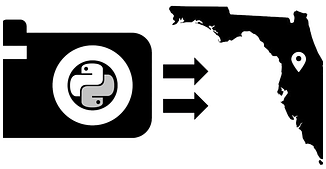
Data Scientist interested in STEM topics from machine learning to cryptography.

More from Adrian Dolinay

Adrian Dolinay

Data Science with Python! Extracting Metadata from Images

Hi everyone! In this post we will learn how to extract metadata from images.



Mar 14, 2023 1

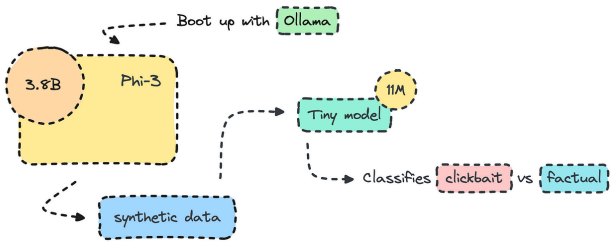
See all from Adrian Dolinay

Recommended from Medium

In CodeX by Rachit

Fine tune BERT for text classification

Jun 20 2



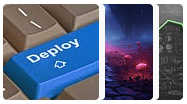
In Towards Data Science by Ida Silfverskiöld

Fine-Tune Smaller Transformer Models: Text Classification

Using Microsoft's Phi-3 to generate synthetic data

May 29 813 7

Lists

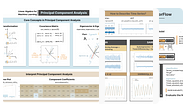


Predictive Modeling w/ Python

20 stories · 1702 saves

Coding & Development

11 stories · 928 saves



Practical Guides to Machine Learning

10 stories · 2073 saves

ChatGPT prompts

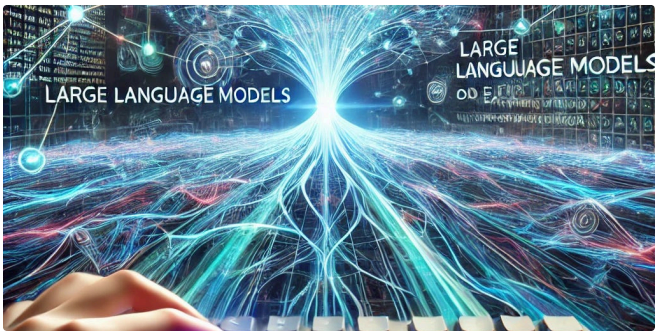
50 stories · 2327 saves

In Stackademic by Abdur Rahman

Python is No More The King of Data Science

5 Reasons Why Python is Losing Its Crown

Oct 23 9K 34



In Generative AI by Shobhit Agarwal

LLM Prompt Secrets For AI Developers: How to Extract...

In today's data-driven world, Large Language Models (LLMs) have become an...

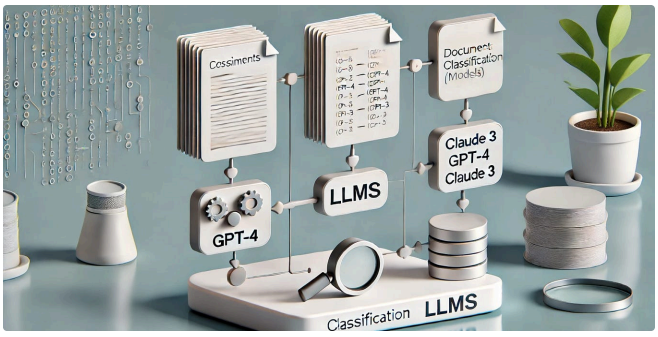
Nov 23 353 1

In Cubed by Michael Wood

The Insanity of Relying on Vector Embeddings: Why RAG Fails

In RAG, the goal is to locate the stored information that has the highest percentage...

Nov 22 1.8K 33



In Level Up Coding by Júlio Almeida

Advanced Document Classification with LLMs

LLMs for document classification. Learn techniques like Type Mapping, Image...

Jun 19 391 2

See more recommendations