

Welcome to Colab!

(New) Try the Gemini API

- [Generate a Gemini API key](#)
- [Talk to Gemini with the Speech-to-Text API](#)
- [Gemini API: Quickstart with Python](#)
- [Gemini API code sample](#)
- [Compare Gemini with ChatGPT](#)
- [More notebooks](#)

If you're already familiar with Colab, check out this video to learn about interactive tables, the executed code history view and the command palette.



Start coding or [generate](#) with AI.

What is Colab?

Colab, or 'Colaboratory', allows you to write and execute Python in your browser, with

- Zero configuration required
- Access to GPUs free of charge
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch [Introduction to Colab](#) to find out more, or just get started below!

✓ Getting started

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

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable and prints the result:

```
seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

↗ 86400

To execute the code in the above cell, select it with a click and then either press the play button to the left of the code, or use the keyboard shortcut 'Command/Ctrl+Enter'. To edit the code, just click the cell and start editing.

Variables that you define in one cell can later be used in other cells:

```
seconds_in_a_week = 7 * seconds_in_a_day
seconds_in_a_week
```

↗ 604800

Colab notebooks allow you to combine **executable code** and **rich text** in a single document, along with **images**, **HTML**, **LaTeX** and more.

When you create your own Colab notebooks, they are stored in your Google Drive account. You can easily share your Colab notebooks with

co-workers or friends, allowing them to comment on your notebooks or even edit them. To find out more, see [Overview of Colab](#). To create a new Colab notebook you can use the File menu above, or use the following link: [Create a new Colab notebook](#).

Colab notebooks are Jupyter notebooks that are hosted by Colab. To find out more about the Jupyter project, see [jupyter.org](#).

✓ Data science

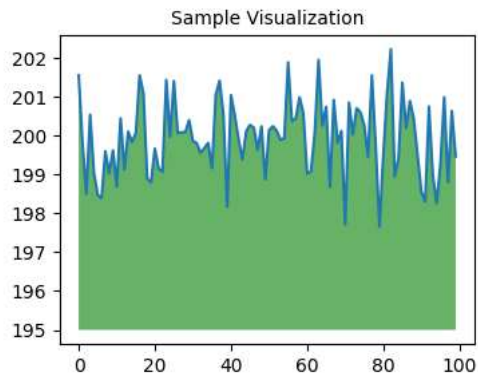
With Colab you can harness the full power of popular Python libraries to analyse and visualise data. The code cell below uses **numpy** to generate some random data, and uses **matplotlib** to visualise it. To edit the code, just click the cell and start editing.

```
import numpy as np
import IPython.display as display
from matplotlib import pyplot as plt
import io
import base64

ys = 200 + np.random.randn(100)
x = [x for x in range(len(ys))]

fig = plt.figure(figsize=(4, 3), facecolor='w')
plt.plot(x, ys, '-')
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)
plt.title("Sample Visualization", fontsize=10)

data = io.BytesIO()
plt.savefig(data)
image = F"data:image/png;base64,{base64.b64encode(data.getvalue()).decode()}"
alt = "Sample Visualization"
display.display(display.Markdown(F"!!![{alt}]({image})"))
plt.close(fig)
```



You can import your own data into Colab notebooks from your Google Drive account, including from spreadsheets, as well as from GitHub and many other sources. To find out more about importing data, and how Colab can be used for data science, see the links below under [Working with data](#).

✓ Machine learning

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

Colab is used extensively in the machine learning community with applications including:

- Getting started with TensorFlow
- Developing and training neural networks
- Experimenting with TPUs
- Disseminating AI research
- Creating tutorials

To see sample Colab notebooks that demonstrate machine learning applications, see the [machine learning examples](#) below.

✓ More resources

Working with notebooks in Colab

- [Overview of Colaboratory](#)
- [Guide to markdown](#)
- [Importing libraries and installing dependencies](#)
- [Saving and loading notebooks in GitHub](#)
- [Interactive forms](#)
- [Interactive widgets](#)

Working with data

- [Loading data: Drive, Sheets and Google Cloud Storage](#)
- [Charts: visualising data](#)
- [Getting started with BigQuery](#)

Machine learning crash course

These are a few of the notebooks from Google's online machine learning course. See the [full course website](#) for more.

- [Intro to Pandas DataFrame](#)
- [Linear regression with tf.keras using synthetic data](#)

Using accelerated hardware

- [TensorFlow with GPUs](#)
- [TensorFlow with TPUs](#)

✓ Featured examples

- [NeMo voice swap](#): Use Nvidia NeMo conversational AI toolkit to swap a voice in an audio fragment with a computer-generated one.
- [Retraining an Image Classifier](#): Build a Keras model on top of a pre-trained image classifier to distinguish flowers.
- [Text Classification](#): Classify IMDB film reviews as either *positive* or *negative*.
- [Style Transfer](#): Use deep learning to transfer style between images.
- [Multilingual Universal Sentence Encoder Q&A](#): Use a machine-learning model to answer questions from the SQuAD dataset.
- [Video Interpolation](#): Predict what happened in a video between the first and the last frame.

🔗 Generate

create a dataframe with 2 columns and 10 rows



Close

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import cifar10

# Load and preprocess dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0 # Normalize images
y_train, y_test = tf.keras.utils.to_categorical(y_train, 10), tf.keras.utils.to_categorical(y_test, 10)

# Define CNN model
def create_model():
    model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(10, activation='softmax')
    ])
    return model
```

```

    })
    return model

model = create_model()
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])


# Train model
history = model.fit(x_train, y_train, epochs=10, batch_size=64, validation_data=(x_test, y_test))

# Plot training history
def plot_results(history):
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Val Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.title('Model Accuracy')

    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.title('Model Loss')
    plt.show()

plot_results(history)

```

 Downloading data from <https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz>
170498071/170498071 ————— **6s** 0us/step
 /usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape` / `input_dim` argument to a layer. These arguments have no effect, and will be removed in a future version of Keras.
 super().__init__(activity_regularizer=activity_regularizer, **kwargs)
 Epoch 1/10
782/782 ————— **134s** 166ms/step - accuracy: 0.2870 - loss: 1.9162 - val_accuracy: 0.4915 - val_loss: 1.4250
 Epoch 2/10
782/782 ————— **129s** 150ms/step - accuracy: 0.5066 - loss: 1.3773 - val_accuracy: 0.6124 - val_loss: 1.0856
 Epoch 3/10
782/782 ————— **141s** 149ms/step - accuracy: 0.5920 - loss: 1.1432 - val_accuracy: 0.6533 - val_loss: 0.9715
 Epoch 4/10
782/782 ————— **118s** 151ms/step - accuracy: 0.6496 - loss: 1.0026 - val_accuracy: 0.6973 - val_loss: 0.8661
 Epoch 5/10
782/782 ————— **137s** 145ms/step - accuracy: 0.6844 - loss: 0.9091 - val_accuracy: 0.7193 - val_loss: 0.8214
 Epoch 6/10
782/782 ————— **145s** 149ms/step - accuracy: 0.7108 - loss: 0.8239 - val_accuracy: 0.7203 - val_loss: 0.8058
 Epoch 7/10
782/782 ————— **142s** 149ms/step - accuracy: 0.7323 - loss: 0.7680 - val_accuracy: 0.7359 - val_loss: 0.7695
 Epoch 8/10
782/782 ————— **144s** 151ms/step - accuracy: 0.7507 - loss: 0.7129 - val_accuracy: 0.7281 - val_loss: 0.7958
 Epoch 9/10
782/782 ————— **140s** 149ms/step - accuracy: 0.7687 - loss: 0.6549 - val_accuracy: 0.7490 - val_loss: 0.7276
 Epoch 10/10
782/782 ————— **114s** 146ms/step - accuracy: 0.7826 - loss: 0.6200 - val_accuracy: 0.7452 - val_loss: 0.7640

