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.

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

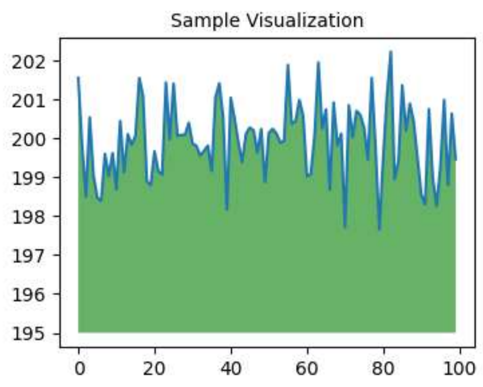
With Colab you can harness the full power of popular Python libraries to analyse and visualise data. The code cell below uses **numpy** to generate some random data, and uses **matplotlib** to visualise it. To edit the code, just click the cell and start editing.

```
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.

```
import nltk
nltk.download('all')
```

```
[nltk_data] Downloading collection 'all'
[nltk_data]    |
[nltk_data]    | Downloading package abc to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/abc.zip.
[nltk_data]    | Downloading package alpino to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/alpino.zip.
[nltk_data]    | Downloading package averaged_perceptron_tagger to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data]    | Downloading package averaged_perceptron_tagger_ru to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping
[nltk_data]    |       taggers/averaged_perceptron_tagger_ru.zip.
[nltk_data]    | Downloading package basque_grammars to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping grammars/basque_grammars.zip.
[nltk_data]    | Downloading package bcp47 to /root/nltk_data...
[nltk_data]    | Downloading package biocreative_ppi to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/biocreative_ppi.zip.
[nltk_data]    | Downloading package bllip_wsj_no_aux to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping models/bllip_wsj_no_aux.zip.
[nltk_data]    | Downloading package book_grammars to
[nltk_data]    |     /root/nltk_data...
```

```
[nltk_data]    |   Unzipping grammars/book_grammars.zip.
[nltk_data]    | Downloading package brown to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/brown.zip.
[nltk_data]    | Downloading package brown_tei to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/brown_tei.zip.
[nltk_data]    | Downloading package cess_cat to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/cess_cat.zip.
[nltk_data]    | Downloading package cess_esp to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/cess_esp.zip.
[nltk_data]    | Downloading package chat80 to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/chat80.zip.
[nltk_data]    | Downloading package city_database to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/city_database.zip.
[nltk_data]    | Downloading package cmudict to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/cmudict.zip.
[nltk_data]    | Downloading package comparative_sentences to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/comparative_sentences.zip.
[nltk_data]    | Downloading package comtrans to /root/nltk_data...
[nltk_data]    | Downloading package conll2000 to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/conll2000.zip.
[nltk_data]    | Downloading package conll2002 to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/conll2002.zip.
[nltk_data]    | Downloading package conll2007 to /root/nltk_data...
[nltk_data]    | Downloading package crubadan to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/crubadan.zip.
[nltk_data]    | Downloading package dependency_treebank to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/dependency_treebank.zip.
[nltk_data]    | Downloading package dolch to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/dolch.zip.
```

```python
# importing regular expression
import re

# words
w = []

# reading text file
with open('/content/final.txt', 'r', encoding="utf8") as f:
    file_name_data = f.read()
    file_name_data = file_name_data.lower()
    w = re.findall('\w+', file_name_data)

# vocabulary
main_set = set(w)
```

```python
# Functions to count the frequency
# of the words in the whole text file


def counting_words(words):
    word_count = {}
    for word in words:
        if word in word_count:
            word_count[word] += 1
        else:
            word_count[word] = 1
    return word_count


# Calculating the probability of each word
def prob_cal(word_count_dict):
    probs = {}
    m = sum(word_count_dict.values())
    for key in word_count_dict.keys():
        probs[key] = word_count_dict[key] / m
    return probs
```

```
pip install pattern
```

```
  Collecting zc.lockfile (from cherrypy->pattern)
    Downloading zc.lockfile-3.0.post1-py3-none-any.whl (9.8 kB)
  Collecting jaraco.collections (from cherrypy->pattern)
    Downloading jaraco.collections-5.0.1-py3-none-any.whl (10 kB)
  Collecting sgmllib3k (from feedparser->pattern)
    Downloading sgmllib3k-1.0.0.tar.gz (5.8 kB)
    Preparing metadata (setup.py) ... done
  Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk->pattern) (8.1.7)
  Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk->pattern) (1.4.0)
  Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk->pattern) (2023.12.25)
  Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk->pattern) (4.66.2)
  Requirement already satisfied: charset-normalizer>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from pdfminer.six->pattern) (3.3
  Requirement already satisfied: cryptography>=36.0.0 in /usr/local/lib/python3.10/dist-packages (from pdfminer.six->pattern) (42.0.5)
  Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from python-docx->pattern) (4.11.0)
  Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->pattern) (3.7)
  Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->pattern) (2.0.7)
  Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->pattern) (2024.2.2)
  Collecting jaraco.functools (from cheroot>=8.2.1->cherrypy->pattern)
    Downloading jaraco.functools-4.0.1-py3-none-any.whl (9.8 kB)
  Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.10/dist-packages (from cryptography>=36.0.0->pdfminer.six->patter
  Collecting tempora>=1.8 (from portend>=2.1.1->cherrypy->pattern)
    Downloading tempora-5.5.1-py3-none-any.whl (13 kB)
  Collecting jaraco.text (from jaraco.collections->cherrypy->pattern)
    Downloading jaraco.text-3.12.0-py3-none-any.whl (11 kB)
  Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from zc.lockfile->cherrypy->pattern) (67.7.2)
  Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-packages (from cffi>=1.12->cryptography>=36.0.0->pdfminer
  Requirement already satisfied: pytz in /usr/local/lib/python3.10/dist-packages (from tempora>=1.8->portend>=2.1.1->cherrypy->pattern)
  Collecting jaraco.context>=4.1 (from jaraco.text->jaraco.collections->cherrypy->pattern)
    Downloading jaraco.context-5.3.0-py3-none-any.whl (6.5 kB)
  Collecting autocommand (from jaraco.text->jaraco.collections->cherrypy->pattern)
    Downloading autocommand-2.2.2-py3-none-any.whl (19 kB)
  Requirement already satisfied: inflect in /usr/local/lib/python3.10/dist-packages (from jaraco.text->jaraco.collections->cherrypy->pa
  Collecting backports.tarfile (from jaraco.context>=4.1->jaraco.text->jaraco.collections->cherrypy->pattern)
    Downloading backports.tarfile-1.1.0-py3-none-any.whl (29 kB)
  Requirement already satisfied: pydantic>=1.9.1 in /usr/local/lib/python3.10/dist-packages (from inflect->jaraco.text->jaraco.collecti
  Requirement already satisfied: annotated-types>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from pydantic>=1.9.1->inflect->jara
  Requirement already satisfied: pydantic-core==2.18.1 in /usr/local/lib/python3.10/dist-packages (from pydantic>=1.9.1->inflect->jarac
  Building wheels for collected packages: pattern, mysqlclient, sgmllib3k
    Building wheel for pattern (setup.py) ... done
    Created wheel for pattern: filename=Pattern-3.6-py3-none-any.whl size=22332702 sha256=0e3114a41b66f1bccb3adb24901996c28047fb3ec1bec
    Stored in directory: /root/.cache/pip/wheels/d1/8f/40/fe23abd593ef60be5bfaf3e02154d3484df42aa947bbf4d499
    Building wheel for mysqlclient (pyproject.toml) ... done
    Created wheel for mysqlclient: filename=mysqlclient-2.2.4-cp310-cp310-linux_x86_64.whl size=124730 sha256=9cc5b81137ac7dcfbb4863bc5
    Stored in directory: /root/.cache/pip/wheels/ac/96/ac/2a4d8cb58a4d95de1dffc3f8b0ea42e0e5b63ab97640edbda3
    Building wheel for sgmllib3k (setup.py) ... done
    Created wheel for sgmllib3k: filename=sgmllib3k-1.0.0-py3-none-any.whl size=6049 sha256=6ed6d8c2f4d0600e1f6916cb6660d4911e3050bf562
    Stored in directory: /root/.cache/pip/wheels/f0/69/93/a47e9d621be168e9e33c7ce60524393c0b92ae83cf6c6e89c5
  Successfully built pattern mysqlclient sgmllib3k
  Installing collected packages: sgmllib3k, backports.csv, zc.lockfile, python-docx, mysqlclient, jaraco.functools, feedparser, backpor
  Successfully installed autocommand-2.2.2 backports.csv-1.0.7 backports.tarfile-1.1.0 cheroot-10.0.0 cherrypy-18.9.0 feedparser-6.0.11
```

```python
# LemmWord: extracting and adding
# root word i.e.Lemma using pattern module
import pattern
from pattern.en import lemma, lexeme
from nltk.stem import WordNetLemmatizer


def LemmWord(word):
    return list(lexeme(wd) for wd in word.split())[0]


# Deleting letters from the words
def DeleteLetter(word):
    delete_list = []
    split_list = []

    # considering letters 0 to i then i to -1
    # Leaving the ith letter
    for i in range(len(word)):
        split_list.append((word[0:i], word[i:]))

    for a, b in split_list:
        delete_list.append(a + b[1:])
    return delete_list
```

```
# Switching two letters in a word
def Switch_(word):
    split_list = []
    switch_l = []

    #creating pair of the words(and breaking them)
    for i in range(len(word)):
        split_list.append((word[0:i], word[i:]))

    #Printint the first word (i.e. a)
    #then replacing the first and second character of b
    switch_l = [a + b[1] + b[0] + b[2:] for a, b in split_list if len(b) >= 2]
    return switch_l


def Replace_(word):
    split_l = []
    replace_list = []

    # Replacing the letter one-by-one from the list of alphs
    for i in range(len(word)):
        split_l.append((word[0:i], word[i:]))
    alphs = 'abcdefghijklmnopqrstuvwxyz'
    replace_list = [a + l + (b[1:] if len(b) > 1 else '')
                    for a, b in split_l if b for l in alphs]
    return replace_list


def insert_(word):
    split_l = []
    insert_list = []

    # Making pairs of the split words
    for i in range(len(word) + 1):
        split_l.append((word[0:i], word[i:]))

    # Storing new words in a list
    # But one new character at each location
    alphs = 'abcdefghijklmnopqrstuvwxyz'
    insert_list = [a + l + b for a, b in split_l for l in alphs]
    return insert_list


# Collecting all the words
# in a set(so that no word will repeat)
def colab_1(word, allow_switches=True):
    colab_1 = set()
    colab_1.update(DeleteLetter(word))
    if allow_switches:
        colab_1.update(Switch_(word))
    colab_1.update(Replace_(word))
    colab_1.update(insert_(word))
    return colab_1

# collecting words using by allowing switches
def colab_2(word, allow_switches=True):
    colab_2 = set()
    edit_one = colab_1(word, allow_switches=allow_switches)
    for w in edit_one:
        if w:
            edit_two = colab_1(w, allow_switches=allow_switches)
            colab_2.update(edit_two)
    return colab_2


# Only storing those values which are in the vocab
def get_corrections(word, probs, vocab, n=2):
    suggested_word = []
    best_suggestion = []
    suggested_word = list(
        (word in vocab and word) or colab_1(word).intersection(vocab)
        or colab_2(word).intersection(
            vocab))

    # finding out the words with high frequencies
    best_suggestion = [[s, probs[s]] for s in list(reversed(suggested_word))]
    return best_suggestion
```

```
# Input
my_word = input("Enter any word:")

# Counting word function
word_count = counting_words(main_set)

# Calculating probability
probs = prob_cal(word_count)

# only storing correct words
tmp_corrections = get_corrections(my_word, probs, main_set, 2)
for i, word_prob in enumerate(tmp_corrections):
    if(i < 3):
        print(word_prob[0])
    else:
        break
```

```
 Enter any word:
```