**AUTO SAGE APP USING GEMINI FLASH**



**Team leader**



**Team Id: LTVIP2026TMIDS38648**

**Name: CH.SIRISHA**

**Register Number: SBAP0032749**

**Hall Ticket Number: 233887401004**

**Mail ID:** [**sirishasiri4237@gmail.com**](mailto:sirishasiri4237@gmail.com)

**Team member:**

****

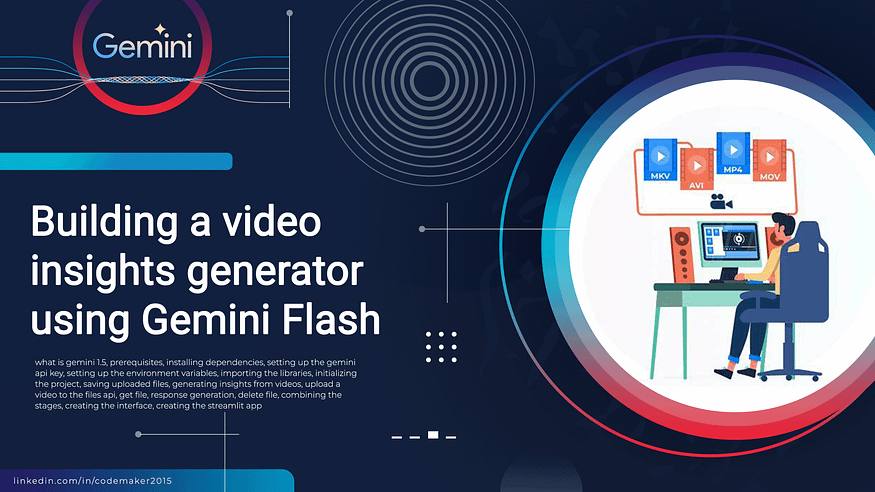
**Name: GUNDABATTULA LAKSHMI DURGA**

**Register number:SBAP0032736**

**Hall Ticket Number: 233887401005**

**Mail ID:** [**gtulasidurga9@gmail.com**](mailto:gtulasidurga9@gmail.com)

### Auto Sage App Using Gemini Flash



Auto Sage is a cutting-edge application powered by Gemini Flash technology, designed to provide comprehensive information on new two-wheeler and four-wheeler vehicles. This vehicle expert tool offers detailed specifications, reviews, and comparisons, helping users make informed decisions about their next vehicle purchase. With its user-friendly interface and real-time updates, Auto Sage ensures that users stay up-to-date with the latest automotive trends and innovations, enhancing their vehicle selection process.

**Scenario 1: Buying a New Motorcycle**

Sarah wants to buy a new motorcycle. She uses Auto Sage App with Gemini Flash to compare specs, features, and prices of various models. The app provides real-time updates and reviews on the latest motorcycles, helping her make an informed decision and choose the best option within her budget.

**Scenario 2: Vehicle Maintenance Tips**

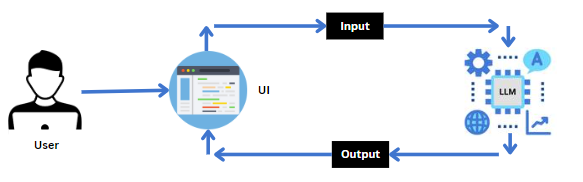
Auto Sage App alerts users about seasonal maintenance tips for their vehicles. For instance, before winter, it provides advice on checking tire pressure, battery health, and antifreeze levels. This proactive approach ensures users keep their two-wheelers and four-wheelers in top condition, enhancing safety and performance.

**Scenario 3: Finding Eco-Friendly Vehicles**

Emma is looking for eco-friendly vehicle options. She utilizes Auto Sage App with Gemini Flash to explore the newest electric and hybrid cars on the market. The app provides insights into vehicle efficiency, environmental impact, and incentives, helping her choose a sustainable option that aligns with her green goals.

**ARCHITECTURE**

**Architecture:**



The Auto Sage App is designed as an AI-powered vehicle expert tool. It uses **Gemini Flash** for speed, efficiency, and multimodal capabilities. The architecture focuses on real-time, context-aware interaction. It provides vehicle diagnostics, maintenance tips, and specifications.

High-Level Architecture Components

* **User Interface (Frontend):** A responsive interface (mobile/web). It allows users to input vehicle queries via text, voice, or image uploads. For example, a photo of a dashboard warning light.
* **API Gateway & Backend:** A secure backend service (e.g., Firebase, Google Cloud Functions). It manages user authentication and API calls to the AI model
* **AI Engine (Gemini Flash):** This is the core intelligence. It uses or newer for high-speed, low-latency reasoning.

**Multimodal Inputs:** Processes images, video, and text to identify car parts or error codes.

**Agentic Workflows:** Acts as a specialized agent to interpret complex maintenance scenarios.

* **Knowledge Base/RAG (Retrieval-Augmented Generation):** This is integrated to provide accurate, up-to-date data on vehicle makes, models, and repair procedures. This reduces hallucinations.
* **Infrastructure (Vertex AI/Google Cloud):**

**Vertex AI:** Hosts the Gemini Flash model. This allows for customized system instructions and high-throughput API access.

**Function Calling:** Allows Gemini to call specific API functions. This fetches real-time data, such as part prices or recalls.

**Key Architectural Workflows**

**Vehicle Diagnosis Workflow:** A user uploads an image of an engine issue. The app sends the image to Gemini Flash. Gemini analyzes the image, identifies the component, and suggests troubleshooting steps in near real-time.

**Conversational Maintenance Chat bot:** A user queries, "When should I change my timing belt?" Gemini accesses a specialized knowledge base. It returns the specific mileage/time interval for the user's specific car model.

Development Stack Recommendations

* **Model:** Gemini 1.5/2.0/3.0 Flash (for speed/cost efficiency).
* **Platform:** Vertex AI or Google AI Studio.
* **Frameworks:** Python (Gen AI SDK), Lang Chain, or Firebase AI.

This architecture provides a scalable, fast, and intelligent agentic app. It is designed for both consumers and professional automotive use.

**AI responses may include mistakes.**

[https://cloud.google.com/blog/topics/developers-practitioners/getting-started-with-gemini-3-   
hello-world-with-gemini-3-flash](https://cloud.google.com/blog/topics/developers-practitioners/getting-started-with-gemini-3-%20%20%20%20%20hello-world-with-gemini-3-flash)

<https://docs.cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-0-flash>

[https://superagi.com/scaling-agentic-ai-best-practices-for-enterprise-wide-deployment-and-multi-agent-system-architecture/](https://superagi.com/scaling-agentic-ai-best-practices-for-enterprise-wide-deployment-and-multi-agent-system-architecture/#:~:text=This%20is%20where%20modular%20architecture%20and%20continuous,only%20effective%20but%20also%20efficient%20and%20scalable.)

**Project Flow**

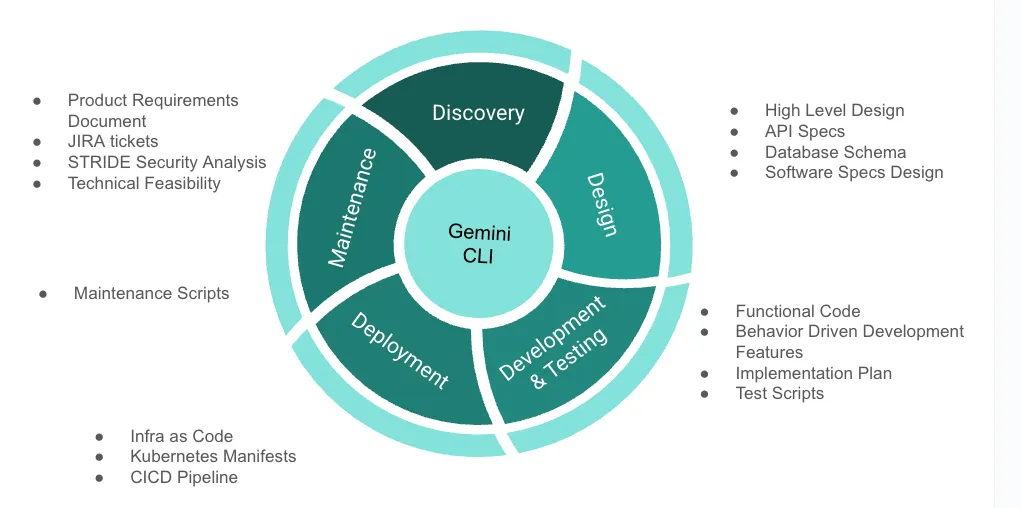
* User interacts with the UI to enter the input.
* User input is collected from the UI and transmitted to the backend using the Google API key.
* The input is then forwarded to the Gemini flash pre-trained model via an API call.
* The Gemini Pro pre-trained model processes the input and generates the output.

The results are returned to the frontend for formatting and display.

* To accomplish this, we have to complete all the activities listed below.

Requirements Specificatio

* Create a requirements.txt file to list the required libraries.
* Install the required libraries



**Initialization of Google API Key**

* Generate Google API Key
* Initialize Google API Key

**Interfacing with Pre-trained Model**

* Load the Gemini flash pre-trained model
* Implement a function to get gemini response
* Implement a function to read PDF content
* Write a prompt for gemini model

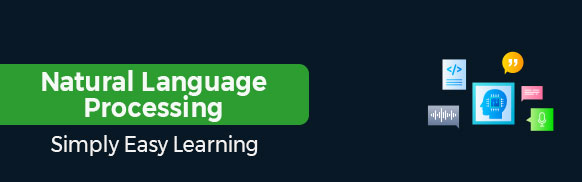
**Model Deployment**

* Integrate with Web Framework
* Host the Application

### Prior Knowledge

You must have the prior knowledge of the following topics to complete this project.

* Generative AI Concept
* NLP: <https://www.tutorialspoint.com/natural_language_processing/index.htm>



# Natural Language Processing Tutorial

Top of Form

Job Search [PDF Version](https://www.tutorialspoint.com/natural_language_processing/natural_language_processing_pdf_version.htm) [Quick Guide](https://www.tutorialspoint.com/natural_language_processing/natural_language_processing_quick_guide.htm) [Resources](https://www.tutorialspoint.com/natural_language_processing/natural_language_processing_useful_resources.htm) [Discussion](https://www.tutorialspoint.com/natural_language_processing/natural_language_processing_discussion.htm)

Bottom of Form

Language is a method of communication with the help of which we can speak, read and write. Natural Language Processing (NLP) is a subfield of Computer Science that deals with Artificial Intelligence (AI), which enables computers to understand and process human language.

# Audience

This tutorial is designed to benefit graduates, postgraduates, and research students who either have an interest in this subject or have this subject as a part of their curriculum. The reader can be a beginner or an advanced learner.

# Prerequisites

The reader must have basic knowledge about Artificial Intelligence. He/she should also be aware about basic terminologies used in English grammar and Python programming concepts.

* Generative AI:

<https://en.wikipedia.org/wiki/Generative_artificial_intelligence>

**Generative artificial intelligence**, also known as **generative AI** or **GenAI**, is a subfield of [artificial,intelligence](https://en.wikipedia.org/wiki/Artificial_intelligence) thatuses [generative,models](https://en.wikipedia.org/wiki/Generative_model) to-generate text, [images](https://en.wikipedia.org/wiki/Images), [videos](https://en.wikipedia.org/wiki/Videos), [audio](https://en.wikipedia.org/wiki/Digital_audio), [software code](https://en.wikipedia.org/wiki/Computer_program) or other forms of data. These models [learn](https://en.wikipedia.org/wiki/Machine_learning) the underlying patterns and structures of their [training data](https://en.wikipedia.org/wiki/Training_data) and use them to generate new data in response to input, which often takes the form of natural language [prompts](https://en.wikipedia.org/wiki/Prompt_(natural_language)). The generated material is often called AIGC (AI Generated Content).

[](https://en.wikipedia.org/wiki/File:Th%C3%A9%C3%A2tre_D%E2%80%99op%C3%A9ra_Spatial.png)

[Théâtre D'opéra Spatial](https://en.wikipedia.org/wiki/Th%C3%A9%C3%A2tre_D%27op%C3%A9ra_Spatial) (2022), an image made using generative AI

The prevalence of generative AI tools has increased significantly since the [AI boom](https://en.wikipedia.org/wiki/AI_boom) in the 2020s. This boom was made possible by improvements in [deep](https://en.wikipedia.org/wiki/Deep_learning) [neural networks](https://en.wikipedia.org/wiki/Neural_networks), particularly [large language models](https://en.wikipedia.org/wiki/Large_language_model) (LLMs), which are based on the [transformer](https://en.wikipedia.org/wiki/Transformer_(machine_learning_model)) architecture. Generative AI applications include [chat bots](https://en.wikipedia.org/wiki/Chatbots) such as [Chat GPT](https://en.wikipedia.org/wiki/ChatGPT), [Claude](https://en.wikipedia.org/wiki/Claude_(language_model)), [Copilot](https://en.wikipedia.org/wiki/Microsoft_Copilot), [Deep Seek](https://en.wikipedia.org/wiki/DeepSeek_(chatbot)), [Google Gemini](https://en.wikipedia.org/wiki/Gemini_(chatbot)) and [Grok](https://en.wikipedia.org/wiki/Grok_(chatbot)); [text-to-image](https://en.wikipedia.org/wiki/Text-to-image) models such as [Stable Diffusion](https://en.wikipedia.org/wiki/Stable_Diffusion), [Mid journey](https://en.wikipedia.org/wiki/Midjourney), and [DALL-E](https://en.wikipedia.org/wiki/DALL-E); and [text-to-video](https://en.wikipedia.org/wiki/Text-to-video) models such as [Veo](https://en.wikipedia.org/wiki/Veo_(text-to-video_model)), [LTX](https://en.wikipedia.org/wiki/LTX-2) and Sora.

Companies in a variety of sectors have used generative AI, including those in software development, healthcare, finance, entertainment, customer service, sales and marketing, art, writing, and product design.

Generative AI has been used for [cybercrime](https://en.wikipedia.org/wiki/Cybercrime), and to deceive and manipulate people through [fake news](https://en.wikipedia.org/wiki/Fake_news) and [deep fakes](https://en.wikipedia.org/wiki/Deepfake). Generative AI models have been trained on copyrighted works without the right holders' permission. Many generative AI systems use large-scale data centers whose [environmental impacts](https://en.wikipedia.org/wiki/Environmental_impact_of_AI) include [e-waste](https://en.wikipedia.org/wiki/E-waste), consumption of fresh water for cooling, and high energy consumption that is estimated to be growing steadily.

|  |
| --- |
| Part of [a series](https://en.wikipedia.org/wiki/Category:Artificial_intelligence) on |
| [**Artificial intelligence (AI)**](https://en.wikipedia.org/wiki/Artificial_intelligence) |
| [https://upload.wikimedia.org/wikipedia/commons/thumb/6/64/Dall-e_3_%28jan_%2724%29_artificial_intelligence_icon.png/120px-Dall-e_3_%28jan_%2724%29_artificial_intelligence_icon.png](https://en.wikipedia.org/wiki/File:Dall-e_3_(jan_'24)_artificial_intelligence_icon.png) |
|  |
|  |

## History

Main article: [History of artificial intelligence](https://en.wikipedia.org/wiki/History_of_artificial_intelligence)

### Early history

The origins of algorithmically generated media can be traced to the development of the [Markov chain](https://en.wikipedia.org/wiki/Markov_chain), which has been used to model natural language since the early 20th century. Russian mathematician [Andrey Markov](https://en.wikipedia.org/wiki/Andrey_Markov) introduced the concept in 1906, including an analysis of vowel and consonant patterns in [*Eugeny Onegin*](https://en.wikipedia.org/wiki/Eugeny_Onegin). Once trained on a [text corpus](https://en.wikipedia.org/wiki/Text_corpus), a Markov chain can generate probabilistic text.

By the early 1970s, artists began using computers to extend generative techniques beyond Markov models. [Harold Cohen](https://en.wikipedia.org/wiki/Harold_Cohen_(artist)) developed and exhibited works produced by [AARON](https://en.wikipedia.org/wiki/AARON), a pioneering computer program designed to autonomously create paintings. The terms generative [AI planning](https://en.wikipedia.org/wiki/AI_planning) or generative planning were used in the 1980s and 1990s to refer to [AI planning](https://en.wikipedia.org/wiki/AI_planning) systems, especially [computer-aided process planning](https://en.wikipedia.org/wiki/Computer-aided_process_planning), used to generate sequences of actions to reach a specified goal. Generative AI planning systems used [symbolic AI](https://en.wikipedia.org/wiki/Symbolic_AI) methods such as [state space search](https://en.wikipedia.org/wiki/State_space_search) and [constraint satisfaction](https://en.wikipedia.org/wiki/Constraint_satisfaction) and were a "relatively mature" technology by the early 1990s. They were used to generate crisis action plans for military use, process plans for manufacturing and decision plans such as in prototype autonomous spacecraft.

### Generative neural networks (since the late 2000s)

See also: [Machine learning](https://en.wikipedia.org/wiki/Machine_learning) and [deep learning](https://en.wikipedia.org/wiki/Deep_learning)

[https://upload.wikimedia.org/wikipedia/commons/thumb/9/9c/Discriminative_vs_Generative_Neural_Networks.png/250px-Discriminative_vs_Generative_Neural_Networks.png](https://en.wikipedia.org/wiki/File:Discriminative_vs_Generative_Neural_Networks.png)

Above: An [image classifier](https://en.wikipedia.org/wiki/Image_classifier), an example of a neural network trained with a [discriminative](https://en.wikipedia.org/wiki/Discriminative_model) objective. Below: A [text-to-image model](https://en.wikipedia.org/wiki/Text-to-image_model), an example of a network trained with a [generative](https://en.wikipedia.org/wiki/Generative_model) objective.

[Machine learning](https://en.wikipedia.org/wiki/Machine_learning) uses both [discriminative models](https://en.wikipedia.org/wiki/Discriminative_model) and [generative models](https://en.wikipedia.org/wiki/Generative_model) to predict data. Beginning in the late 2000s, the introduction of [deep learning](https://en.wikipedia.org/wiki/Deep_learning) technology led to improvements in [image classification](https://en.wikipedia.org/wiki/Image_classification), [speech recognition](https://en.wikipedia.org/wiki/Speech_recognition), [natural language processing](https://en.wikipedia.org/wiki/Natural_language_processing) and other tasks. [Neural networks](https://en.wikipedia.org/wiki/Neural_network) in this era were typically trained as [discriminative](https://en.wikipedia.org/wiki/Discriminative_model) models due to the difficulty of generative modeling.

In 2014, advancements such as the [variational autoencoder](https://en.wikipedia.org/wiki/Variational_autoencoder) and [generative adversarial network](https://en.wikipedia.org/wiki/Generative_adversarial_network) produced the first practical deep neural networks capable of learning generative models, as opposed to discriminative ones, for complex data such as images. These deep generative models were the first to output not only class labels for images but also entire images, such as [Deep Dream](https://en.wikipedia.org/wiki/DeepDream).

In 2017, the [Transformer](https://en.wikipedia.org/wiki/Transformer_(machine_learning_model)) network enabled advancements in generative models compared to older [long short-term memory](https://en.wikipedia.org/wiki/Long_short-term_memory) (LSTM) models, leading to the first [generative pre-trained transformer](https://en.wikipedia.org/wiki/Generative_pre-trained_transformer) (GPT), known as [GPT-1](https://en.wikipedia.org/wiki/GPT-1), in 2018.

### Generative AI adoption

Main article: [AI boom](https://en.wikipedia.org/wiki/AI_boom)

**[](https://en.wikipedia.org/wiki/File:Timeline-of-AI-generated-faces.png)**

**images AI generated have become much more advanced.**

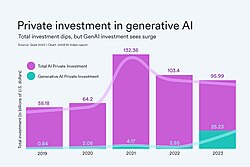
In March 2020, the release of [15.ai](https://en.wikipedia.org/wiki/15.ai), a free [web application](https://en.wikipedia.org/wiki/Web_application) created by an anonymous [MIT](https://en.wikipedia.org/wiki/MIT) researcher that could generate convincing character voices using minimal training data, marked one of the earliest popular use cases of generative AI. The platform is credited as the first mainstream service for AI voice cloning ([audio deep fakes](https://en.wikipedia.org/wiki/Audio_deepfakes)) in [memes](https://en.wikipedia.org/wiki/Internet_meme) and [content creation](https://en.wikipedia.org/wiki/Content_creation).

In 2021, the emergence of [DALL-E](https://en.wikipedia.org/wiki/DALL-E), a [transformer](https://en.wikipedia.org/wiki/Transformer_(machine_learning_model))-based generative model, marked an advance in AI-generated imagery. While the initial model remained a closed-access tool, open-source [Google Colab](https://en.wikipedia.org/wiki/Google_Colab) projects and initiatives like [VQGAN+CLIP](https://en.wikipedia.org/wiki/VQGAN-CLIP) or DALL-E Mini (now [Craiyon](https://en.wikipedia.org/wiki/Craiyon)) were the first widely-used public [text-to-image generation](https://en.wikipedia.org/wiki/Text-to-image_model) models. By the end of 2021, mobile applications such as Dream by [Wombo](https://en.wikipedia.org/wiki/Wombo) allowed users to generate art from a simple prompts.

This was followed by the releases of [Mid journey](https://en.wikipedia.org/wiki/Midjourney) and [Stable Diffusion](https://en.wikipedia.org/wiki/Stable_Diffusion) in 2022, which further democratized access to [artificial intelligence art](https://en.wikipedia.org/wiki/Artificial_intelligence_art) creation from [natural language prompts](https://en.wikipedia.org/wiki/Prompt_engineering). These systems can generate photorealistic images, artwork, and designs based on text descriptions, leading to widespread adoption among artists, designers, and the general public.

In November 2022, the public release of [Chat GPT](https://en.wikipedia.org/wiki/ChatGPT) popularized [generative AI](https://en.wikipedia.org/wiki/Applications_of_artificial_intelligence) for general-purpose text-based tasks. The system's ability to [engage in natural conversations](https://en.wikipedia.org/wiki/Chatbot), [generate creative content](https://en.wikipedia.org/wiki/AI_art), assist with coding, and perform various

analytical tasks captured global attention and sparked widespread discussion about AI's potential impact on [work](https://en.wikipedia.org/wiki/AI_in_industry), [education](https://en.wikipedia.org/wiki/AI_in_education), and [creativity](https://en.wikipedia.org/wiki/AI_art). As of 2023, generative AI remained "still far from reaching the benchmark of 'general human intelligence'" according to a paper in the [Journal of Information Technology](https://en.wikipedia.org/wiki/Journal_of_Information_Technology).

[](https://en.wikipedia.org/wiki/File:Private_investment_in_generative_AI_-_2024_AI_index.jpg)

Private investment in AI (pink) and generative AI (green)

In a 2024 survey, [Asia–Pacific](https://en.wikipedia.org/wiki/Asia%E2%80%93Pacific) countries were significantly more optimistic than Western societies about generative AI and show higher adoption rates. Despite expressing concerns about privacy and the pace of change, 68% of Asia-Pacific respondents believed that AI was having a positive impact on the world, compared to 57% globally. According to a survey by [SAS](https://en.wikipedia.org/wiki/SAS_Institute) and Coleman Parkes Research, [China](https://en.wikipedia.org/wiki/China) in particular has emerged as a global leader in generative AI adoption, with 83% of Chinese respondents using the technology, exceeding both the global average of 54% and the U.S. rate of 65%. A [UN](https://en.wikipedia.org/wiki/UN) report indicated that Chinese entities filed over 38,000 generative AI [patents](https://en.wikipedia.org/wiki/Patent) from 2014 to 2023, substantially surpassing the United States in patent applications. A 2024 survey on the Chinese social app Soul reported that 18% of respondents born after 2000 used generative AI "almost every day", and that over 60% of respondents like or love

AI-generated content, while less than 3% dislike or hate it.

By mid 2025, despite continued consumer growth, many companies were increasingly abandoning generative AI pilot projects as they had difficulties with integration, data quality and unmet returns, leading analysts at [Gartner](https://en.wikipedia.org/wiki/Gartner) &

[The Economist](https://en.wikipedia.org/wiki/The_Economist) to characterize the period as entering the [Gartner hype cycle](https://en.wikipedia.org/wiki/Gartner_hype_cycle)'s "trough of disillusionment" phase.

# Gemini

* **About Gemini:**

<https://deepmind.google/technologies/gemini/#introduction>

# Gemini 3

# Gemini 3

GEMINI 3

Our most intelligent AI model that brings any idea to life

[Try in Gemini Try in Google AI Studio](https://aistudio.google.com/prompts/new_chat?model=gemini-3-pro-preview&utm_source=deepmind.google&utm_medium=referral&utm_campaign=gdm&utm_content=" \t "_blank)

## Gemini 3 Flash

Our latest Gemini 3 model that helps you bring any idea to life - faster.

**Google Antigravity**

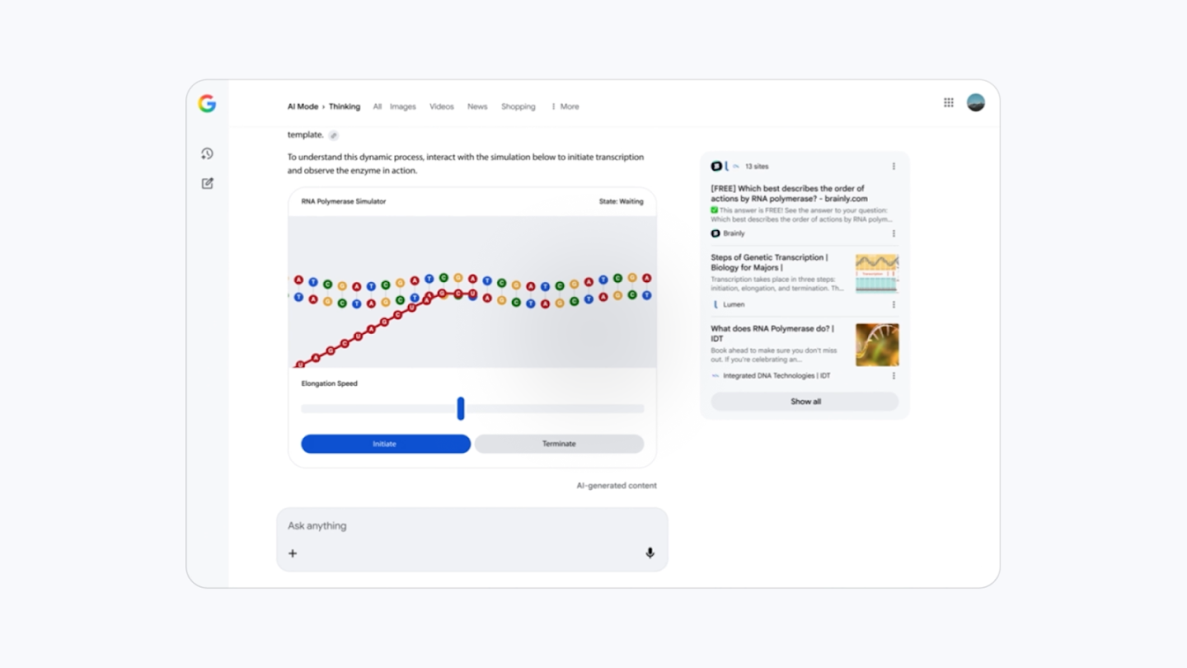
Build with our new agentic development platform.

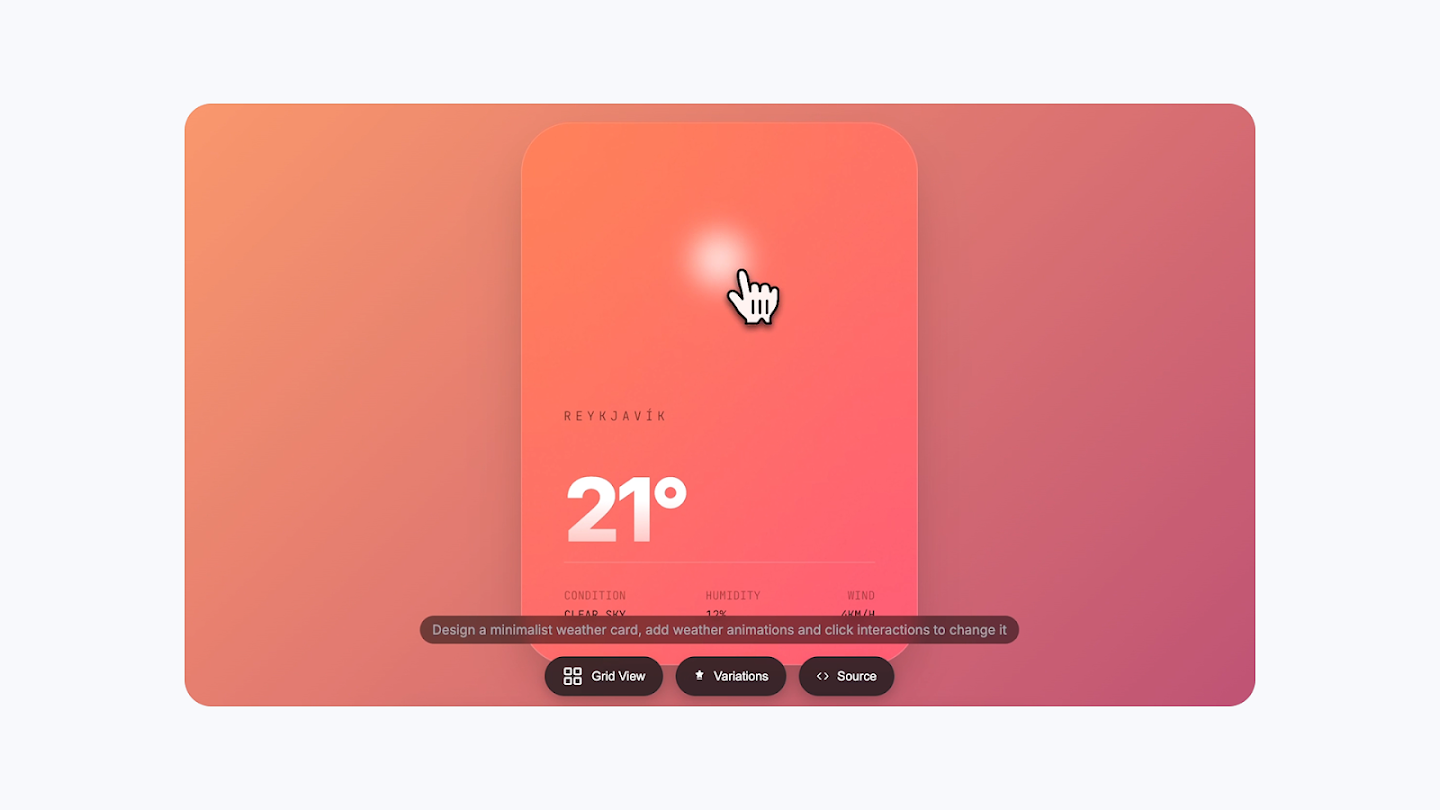
Introducing our most intelligent model yet. With state-of-the-art reasoning to help you learn, build, and plan anything.



**Code a 3D visualization of the universe**

Gemini 3 uses state-of-the-art reasoning to generate richer visualizations and deeper interactivity. See how it codes a seamless 3D journey through the scale of the universe, from a proton to the observable universe, demonstrating a massive leap in "vibe coding" performance over Gemini 2.5.





### Safety

#### Building with responsibility at the core

As we develop these new technologies, we recognize the responsibility it entails, and aim to prioritize safety and security in all our efforts.

### For developers

### Build with cutting-edge generative AI models and tools to make AI helpful for everyone

Gemini’s advanced thinking, native multimodality and massive context window empowers developers to build next-generation experiences.

* **Gemini API:**

<https://ai.google.dev/gemini-api/docs/get-started/python>

# Gemini API quick start

This quick start shows you how to install our [libraries](https://ai.google.dev/gemini-api/docs/libraries) and make your first Gemini API request.

**Before you begin**

Using the Gemini API requires an API key, you can create one for free to get started.

[Create a Gemini API Key](https://aistudio.google.com/app/apikey)

**Install the Google Gen AI SDK**

[Python](https://ai.google.dev/gemini-api/docs/quickstart#python)[JavaScript](https://ai.google.dev/gemini-api/docs/quickstart#javascript)[Go](https://ai.google.dev/gemini-api/docs/quickstart#go)[Java](https://ai.google.dev/gemini-api/docs/quickstart#java)[C#](https://ai.google.dev/gemini-api/docs/quickstart#c)[Apps Script](https://ai.google.dev/gemini-api/docs/quickstart#apps-script)

Using [Python 3.9+](https://www.python.org/downloads/), install the [google-genai package](https://pypi.org/project/google-genai/) using the following [pip command](https://packaging.python.org/en/latest/tutorials/installing-packages/):

pip install -q -U google-genai

## Make your first request

Here is an example that uses the [generateContent](https://ai.google.dev/api/generate-content#method:-models.generatecontent) method to send a request to the Gemini API using the Gemini 2.5 Flash model.

If you [set your API key](https://ai.google.dev/gemini-api/docs/api-key#set-api-env-var) as the environment variable GEMINI\_API\_KEY, it will be picked up automatically by the client when using the [Gemini API libraries](https://ai.google.dev/gemini-api/docs/libraries). Otherwise you will need to [pass your API key](https://ai.google.dev/gemini-api/docs/api-key#provide-api-key-explicitly) as an argument when initializing the client.

Note that all code samples in the Gemini API docs assume that you have set the environment variable GEMINI\_API\_KEY.

[Python](https://ai.google.dev/gemini-api/docs/quickstart#python)[JavaScript](https://ai.google.dev/gemini-api/docs/quickstart#javascript)[Go](https://ai.google.dev/gemini-api/docs/quickstart#go)[Java](https://ai.google.dev/gemini-api/docs/quickstart#java)[C#](https://ai.google.dev/gemini-api/docs/quickstart#c)[Apps Script](https://ai.google.dev/gemini-api/docs/quickstart#apps-script)[REST](https://ai.google.dev/gemini-api/docs/quickstart#rest)

from google import genai

# The client gets the API key from the environment variable `GEMINI\_API\_KEY`.

client = genai.Client()

response = client.models.generate\_content(

model="gemini-3-flash-preview", contents="Explain how AI works in a few words"

)

print(response.text)

* GeminiDemo: <https://colab.research.google.com/github/google/generative-ai-docs/blob/main/site/en/gemini-api/docs/get-started/python.ipynb>

# Get started with the Gemini API: Python

|  |  |  |
| --- | --- | --- |
| [[https://ai.google.dev/static/site-assets/images/docs/notebook-site-button.png](https://www.google.com/url?q=https://ai.google.dev/gemini-api/docs/get-started/python)View on Google AI](https://www.google.com/url?q=https%3A%2F%2Fai.google.dev%2Fgemini-api%2Fdocs%2Fget-started%2Fpython) | [[https://www.tensorflow.org/images/colab_logo_32px.png](https://colab.research.google.com/github/google/generative-ai-docs/blob/main/site/en/gemini-api/docs/get-started/python.ipynb)Run in Google Colab](https://colab.research.google.com/github/google/generative-ai-docs/blob/main/site/en/gemini-api/docs/get-started/python.ipynb) | [[https://www.tensorflow.org/images/GitHub-Mark-32px.png](https://github.com/google/generative-ai-docs/blob/main/site/en/gemini-api/docs/get-started/python.ipynb)View source on GitHub](https://github.com/google/generative-ai-docs/blob/main/site/en/gemini-api/docs/get-started/python.ipynb) |

This quick start demonstrates how to use the Python SDK for the Gemini API, which gives you access to Google's Gemini large language models. In this quick start, you will learn how to:

1. Set up your development environment and API access to use Gemini.
2. Generate text responses from text inputs.
3. Generate text responses from multimodal inputs (text and images).
4. Use Gemini for multi-turn conversations (chat).
5. Use embeddings for large language models.

## Prerequisites

You can run this quick start in [Google Colab](https://colab.research.google.com/github/google/generative-ai-docs/blob/main/site/en/gemini-api/docs/get-started/python.ipynb), which runs this notebook directly in the browser and does not require additional environment configuration.

Alternatively, to complete this quick start locally, ensure that your development environment meets the following requirements:

* Python 3.9+
* An installation of jupyter to run the notebook.

## Setup

### Install the Python SDK

The Python SDK for the Gemini API, is contained in the [google-generativeai](https://www.google.com/url?q=https%3A%2F%2Fpypi.org%2Fproject%2Fgoogle-generativeai%2F) package. Install the dependency using pip:

[ ]

!pip install -q -U google-generativeai

### Import packages

Import the necessary packages.

[ ]

import pathlib  
import textwrap  
  
import google.generativeai as genai  
  
from IPython.display import display  
from IPython.display import Markdown  
  
  
def to\_markdown(text):  
    text = text.replace("•", "  \*")  
    return Markdown(textwrap.indent(text, "> ", predicate=lambda \_: True))

[ ]

# Used to securely store your API key  
from google.colab import userdata

### Setup your API key

Before you can use the Gemini API, you must first obtain an API key. If you don't already have one, create a key with one click in Google AI Studio.

[Get an API key](https://www.google.com/url?q=https%3A%2F%2Fmakersuite.google.com%2Fapp%2Fapikey)

Note that depending on where you are located, you might have to [enable billing](https://www.google.com/url?q=https%3A%2F%2Fai.google.dev%2Fgemini-api%2Fdocs%2Fbilling%23enable-cloud-billing) since the free tier is not available in [EEA (including EU), the UK, and CH](https://www.google.com/url?q=https%3A%2F%2Fai.google.dev%2Fgemini-api%2Fdocs%2Fbilling%23is-Gemini-free-in-EEA-UK-CH)

**Advanced use cases**

The following sections discuss advanced use cases and lower-level details of the Python SDK for the Gemini API.

### Use embeddings

[Embedding](https://developers.google.com/machine-learning/glossary#embedding-vector) is a technique used to represent information as a list of floating point numbers in an array. With Gemini, you can represent text (words, sentences, and blocks of text) in a vectorized form, making it easier to compare and contrast embeddings. For example, two texts that share a similar subject matter or sentiment should have similar embeddings, which can be identified through mathematical comparison techniques such as cosine similarity. For more on how and why you should use embeddings, refer to the [Embeddings guide](https://www.google.com/url?q=https%3A%2F%2Fai.google.dev%2Fdocs%2Fembeddings_guide).

Use the embed\_content method to generate embeddings. The method handles embedding for the following tasks (task\_type):

| **Task Type** | **Description** |
| --- | --- |
| RETRIEVAL\_QUERY | Specifies the given text is a query in a search/retrieval setting. |
| RETRIEVAL\_DOCUMENT | Specifies the given text is a document in a search/retrieval setting. Using this task type requires a title. |
| SEMANTIC\_SIMILARITY | Specifies the given text will be used for Semantic Textual Similarity (STS). |
| CLASSIFICATION | Specifies that the embeddings will be used for classification. |
| CLUSTERING | Specifies that the embeddings will be used for clustering. |

### Safety settings

The safety\_settings argument lets you configure what the model blocks and allows in both prompts and responses. By default, safety settings block content with medium and/or high probability of being unsafe content across all dimensions. Learn more about [Safety settings](https://www.google.com/url?q=https%3A%2F%2Fai.google.dev%2Fdocs%2Fsafety_setting).

Enter a questionable prompt and run the model with the default safety settings, and it will not return any candidates:

Streamlit: <https://www.geeksforgeeks.org/a-beginners-guide-to-streamlit/>

# A Beginners Guide To Streamlit

Last Updated : 16 Jul, 2025

**Streamlit**is an open-source Python library for building interactive web apps using only Python. It's ideal for creating dashboards, data-driven web apps, reporting tools and interactive user interfaces without needing HTML, CSS or JavaScript.

This article introduces key Streamlit features, shows how to build a simple app and explains how to run it on a local server using minimal code.

## Streamlit Installation

Make sure Python and pip are already installed on your system. To install Streamlit, run the following command in the command prompt or terminal:

*pip install streamlit*

## How to run Streamlit file?

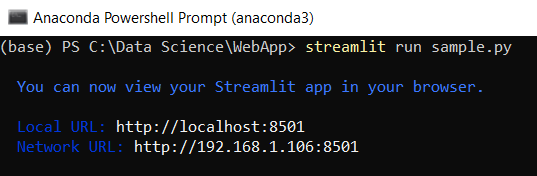
To run a Streamlit app, open the command prompt or Anaconda prompt and type:

*streamlit run filename.py*

For example, if file name is sample.py, run:

*streamlit run sample.py*

After running the command, Streamlit starts a local development server and displays a URL (http://localhost:8501). Open this URL in your browser to view and interact with the app.



Run Streamlit file

## Understanding Streamlit basic functions

Streamlit offers simple built-in functions to create interactive web apps using Python. These functions help display text, add widgets, visualize data and handle user input.

Let’s explore them one by one.

1. **Title**

This basic Streamlit example displays a title on the web page, confirming that Streamlit is set up and running correctly.

**import** **streamlit** **as** **st**

st.title("Hello GeeksForGeeks !!!")

**Output**

Output of Title

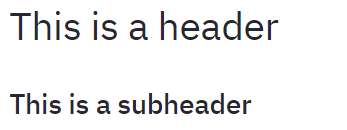
**2. Header and Subheader**

This example demonstrates how to add a header and subheader in a Streamlit app.

st.header("This is a header")

st.subheader("This is a subheader")

**Output**



Output of Header/Subheader

**3. Text**

It shows how to display plain text in a Streamlit app using st.text() function.

st.text("Hello GeeksForGeeks!!!")

**Output**

Output_text

Output of Text

**4. Markdown**

This example uses st.markdown() to show formatted text. The ### creates a level 3 header, used for medium-sized headings in the app.

st.markdown("### This is a markdown")

**Output**



Output of Markdown

**5. Success, Info, Warning, Error and Exception**

This example shows how to display different types of messages and alerts in a Streamlit app using built-in functions like st.success(), st.info(), st.warning(), st.error() and st.exception(). These functions help communicate status, information, warnings, errors and exceptions to users clearly.

st.success("Success")

st.info("Information")

st.warning("Warning")

st.error("Error")

exp = **ZeroDivisionError**("Trying to divide by Zero")

st.exception(exp)

**Output**



Output of Success, Information, Warning, Error and Exception

**6. Write**

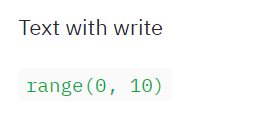
This example uses st.write() that can display text, numbers, data structures and even charts. Here, it's used to show plain text and the output of Python’s built-in range() function.

st.write("Text with write")

*# Writing python inbuilt function range()*

st.write(range(10))

**Output**



Output of write

**7. Display Images**

This example demonstrates how to display an image using Pillow library. The image is opened with Image.open() and displayed with st.image(), where the width parameter controls its size.

**from** **PIL** **import** Image *# Import Image from Pillow*

img = Image.open("streamlit.png") *# Open the image file*

st.image(img, width=200) *# Display the image with a specified width*

**Output**



Output of Display Images

**8. Checkbox**

This example uses a checkbox in Streamlit to toggle content visibility. When the checkbox labeled "Show/Hide" is checked, it displays a text message on the screen.

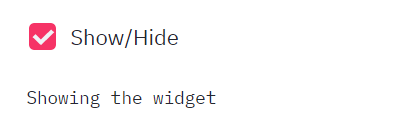
*# Display a checkbox with the label 'Show/Hide'*

**if** st.checkbox("Show/Hide"):

*# Show this text only when the checkbox is checked*

st.text("Showing the widget")

**Output**



Checkbox is not checked. The text is displayed when the box is checked

**9. Radio Button**

This example demonstrates how to use radio buttons to let users select one option from a list. Based on the selected gender, the app displays the result using **st.success()**

*# Create a radio button to select gender*

status = st.radio("Select Gender:", ['Male', 'Female'])

*# Display the selected option using success message*

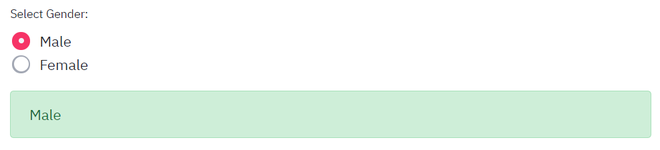
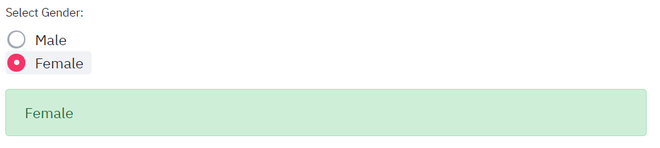
**if** status == 'Male':

st.success("Male")

**else**:

st.success("Female")

**Output**

Success shows Male when Male option is selectedSuccess shows Female when Female option is selected

**10. Selection Box**

This example uses a select box in Streamlit to let users choose one option from a dropdown list. The selected item is then displayed on the screen.

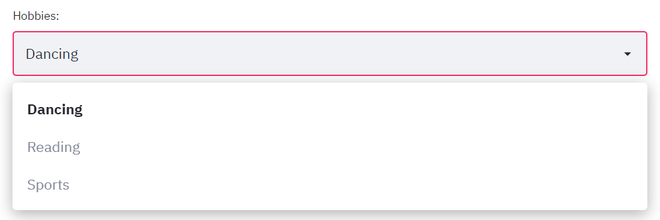
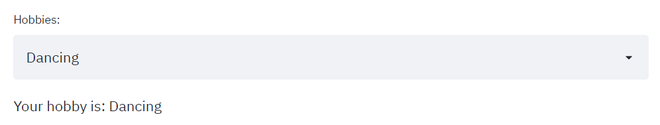
*# Create a dropdown menu for selecting a hobby*

hobby = st.selectbox("Select a Hobby:", ['Dancing', 'Reading', 'Sports'])

*# Display the selected hobby*

st.write("Your hobby is:", hobby)

**Output**

Selectbox showing options to select fromSelcted option is printed

**11. Multi-Selectbox**

This example demonstrates how to use a multiselect box in Streamlit, allowing users to choose multiple options from a list. The app then displays the number of selected items.

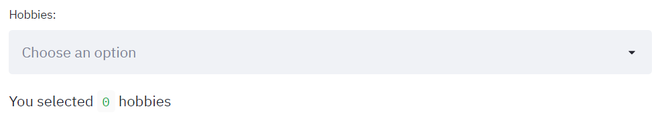
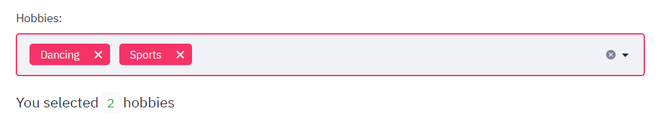
*# Create a multiselect box for choosing hobbies*

hobbies = st.multiselect("Select Your Hobbies:", ['Dancing', 'Reading', 'Sports'])

*# Display the number of selected hobbies*

st.write("You selected", len(hobbies), "hobbies")

**Output**

Multi - SelectboxSelected 2 options

**12. Button**

This example shows how to use buttons in Streamlit. Buttons can trigger specific actions when clicked, such as displaying a message or running a function.

*# A simple button that does nothing*

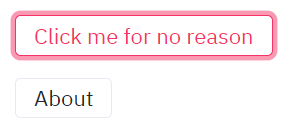
st.button("Click Me")

*# A button that displays text when clicked*

**if** st.button("About"):

st.text("Welcome to GeeksForGeeks!")

**Output**



Click the About button

**13. Text Input**

Text input fields allow users to enter custom data. This example collects a user's name, formats it with proper capitalization and displays it when Submit button is clicked, simulating a basic form interaction.

*# Create a text input box with a default placeholder*

name = st.text\_input("Enter your name", "Type here...")

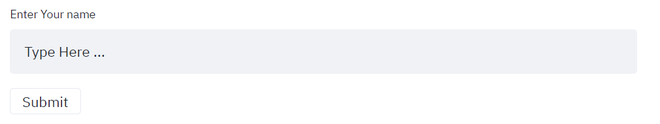
*# Display the name after clicking the Submit button*

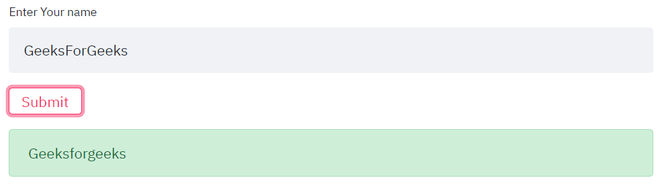
**if** st.button("Submit"):

result = name.title() *# Capitalize the first letter of each word*

st.success(result)

**Output**

Text Input

Display success message when the submit button is clicked

**14.  Slider**

Sliders provide a way to select numeric values within a range. This example lets users choose a level between 1 and 5 and displays the selected value instantly. It is useful for settings like ratings, difficulty levels or thresholds.

*# Create a slider to select a level between 1 and 5*

level = st.slider("Choose a level", min\_value=1, max\_value=5)

*# Display the selected level*

st.write(f"Selected level: **{**level**}**")

**Output**

Output of slider

### Mini Project

Let’s put everything we've learned so far into practice by building a **BMI Calculator web app** using Streamlit. The formula to calculate Body Mass Index (BMI) when weight is in kilograms and height is in meters is:

bmi = weight/height2

**import** **streamlit** **as** **st**

*# Title of the app*

st.title("BMI Calculator")

*# Input: Weight in kilograms*

weight = st.number\_input("Enter your weight (kg):", min\_value=0.0, format="**%.2f**")

*# Input: Height format selection*

height\_unit = st.radio("Select your height unit:", ['Centimeters', 'Meters', 'Feet'])

*# Input: Height value based on selected unit*

height = st.number\_input(f"Enter your height (**{**height\_unit.lower()**}**):", min\_value=0.0, format="**%.2f**")

*# Calculate BMI when button is pressed*

**if** st.button("Calculate BMI"):

**try**:

*# Convert height to meters based on selected unit*

**if** height\_unit == 'Centimeters':

height\_m = height / 100

**elif** height\_unit == 'Feet':

height\_m = height / 3.28

**else**:

height\_m = height

*# Prevent division by zero*

**if** height\_m <= 0:

st.error("Height must be greater than zero.")

**else**:

bmi = weight / (height\_m \*\* 2)

st.success(f"Your BMI is **{**bmi**:**.2f**}**")

*# BMI interpretation*

**if** bmi < 16:

st.error("You are Extremely Underweight")

**elif** 16 <= bmi < 18.5:

st.warning("You are Underweight")

**elif** 18.5 <= bmi < 25:

st.success("You are Healthy")

**elif** 25 <= bmi < 30:

st.warning("You are Overweight")

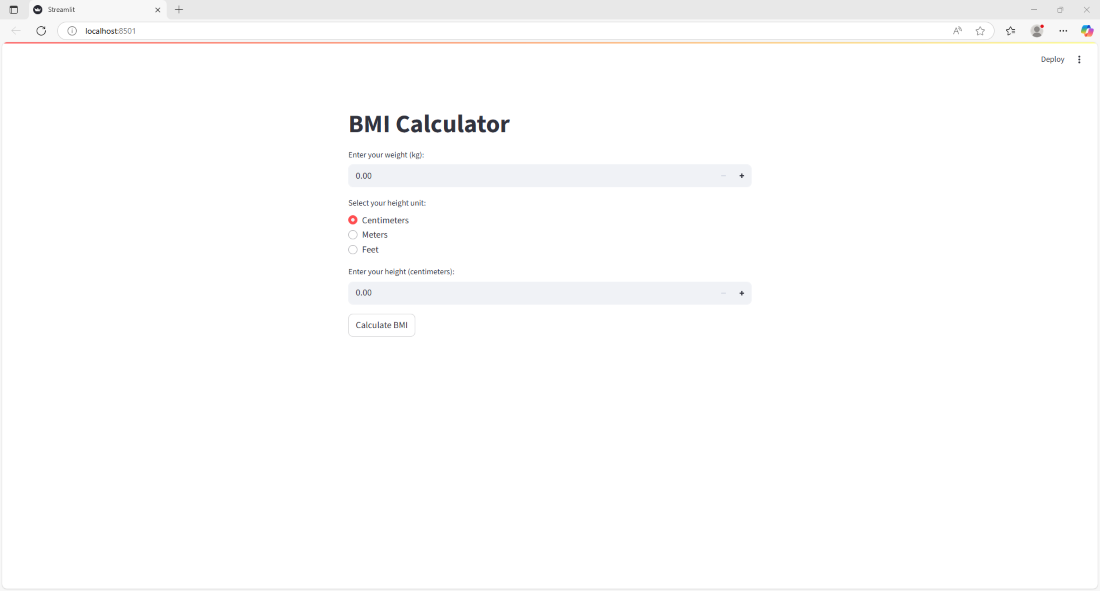
**else**:

st.error("You are Extremely Overweight")

**except**:

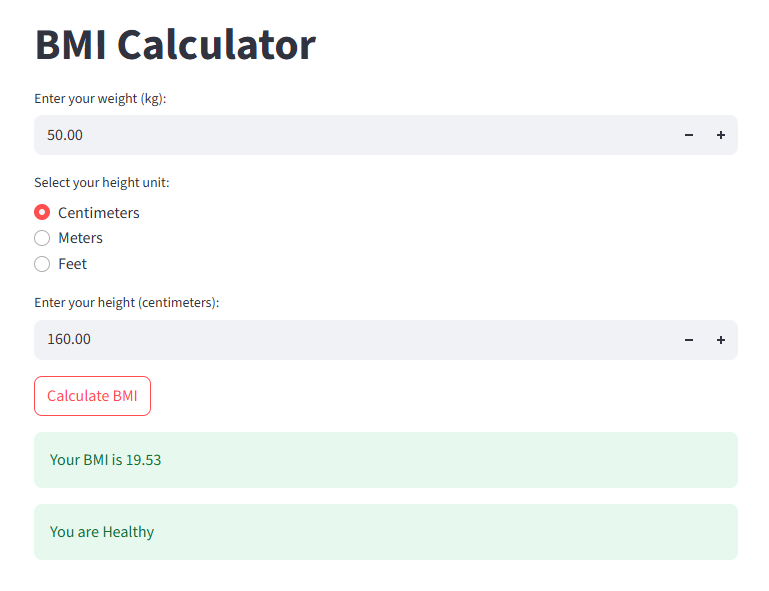
st.error("Please enter valid numeric values.")

**Output**



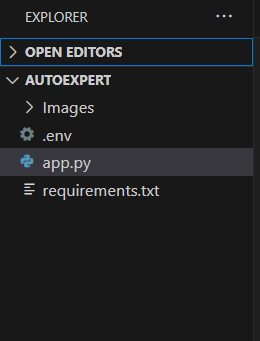
BMI calculator web app

After entering all the required fields:



### Project Structure

Create the Project folder which contains files as shown below:



* images folder: It is established to store the images utilized in the user interface.
* .env file: It securely stores the Google API key.
* app.py: It serves as the primary application file housing both the model and Streamlit UI code.
* requirements.txt: It enumerates the libraries necessary for installation to ensure proper functioning.
* Additionally, ensure proper file organization and adhere to best practices for version control.

my\_project/

├── images/ # Stores images used in the UI

│ └── logo.png

├── .env # Securely stores Google API keys

├── app.py # Main Streamlit application

├── requirements.txt # Lists dependencies (e.g., streamlit, google-generativeai)

└── .gitignore # Excludes .env, \_\_pycache\_\_, and venv/

**Manual Steps to Create the Project Structure**

1. **Create the main project folder:**
   * Navigate to where you want the project to be located.
   * Create a new folder and name it Project.

**Create sub-folders inside Project:**

* + Open the Project folder.
  + Create new folders named Images, StyleSheets, and Scripts.

**Create the files inside the respective folders:**

* + **Inside Project folder:** Create a file named index.html.
  + **Inside Images folder:** Create files named logo.png and background.jpg.
  + **Inside StyleSheets folder:** Create a file named style.css.
  + **Inside Scripts folder:** Create a file named script.js.

**Using the Command Line (CLI)**

Alternatively, you can use your command line to create the entire structure quickly. Open your terminal or command prompt, navigate to your desired location, and run the following commands: 

bash

mkdir Project

cd Project

*# Create top-level file*

touch index.html

*# Create sub-folders and files within them*

mkdir Images StyleSheets Scripts

touch Images/logo.png Images/background.jpg

touch StyleSheets/style.css

touch Scripts/script.js

### Requirements Specification

Specifying the required libraries in the requirements.txt file ensures seamless setup and reproducibility of the project environment, making it easier for others to replicate the development environment.

**requirements.txt?**

* **Reproducibility:** Pinning exact package versions (e.g., requests==2.18.4) ensures everyone uses the same configuration, eliminating "it works on my machine" issues.
* **Dependency Management:** It provides a clear, centralized list of all external packages needed for the project to run correctly.
* **Simplified Setup:** New team members or deployment pipelines can quickly set up the environment with one command.
* **Version Control:** Including the file in your version control system (like Git) allows you to track dependency changes over time.

**Steps for Using requirements.txt**

**1. Create a Virtual Environment (Recommended)**

Always work within a project-specific virtual environment to isolate dependencies from system-wide Python packages.

bash

*# Create a virtual environment*

python3 -m venv myenv

*# Activate the virtual environment (Linux/macOS)*

source myenv/bin/activate

*# Activate the virtual environment (Windows)*

.\\myenv\\Scripts\\activate

**2. Install Required Packages**

Install the packages your project needs using pip.

bash

pip install package-name

pip install another-package

**3. Generate the requirements.txt File**

Once all necessary packages are installed and the virtual environment is active, generate the requirements.txt file using pip freeze. This command captures the *exact* versions of all installed packages in the current environment.

bash

pip freeze > requirements.txt

Review the file to remove any unnecessary packages not directly used by your project.

**4. Install Dependencies from the File**

On a new machine or environment, you can install all required dependencies with a single command:

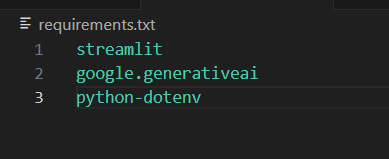
bash

pip install -r requirements.txt

To automate environment setup, tools like Gemini Code Assist can be used. Provide project requirements in a document and use Gemini to generate code. This includes a bash script to create the project structure and the requirements.txt file.

* Prompts can be used such as: "Generate a requirements.txt file so I can install all the required dependencies with a single command".
* Incorporate the generated script or instructions into a CI/CD pipeline or setup process to ensure deployment.

**Create a requirements.txt file to list the required libraries.**



* streamlit: Streamlit is a powerful framework for building interactive web applications with Python.
* google-generativeai: Python client library for accessing the GenerativeAI API, facilitating interactions with pre-trained language models like Gemini Pro.
* python-dotenv: Python-dotenv allows you to manage environment variables stored in a .env file for your Python projects.
* Pillow: Pillow is a Python Imaging Library (PIL) fork that adds support for opening, manipulating, and saving many different image file formats.

A requirements.txt file can be created to list the required libraries. This file should contain the libraries and their versions.

The following content can be placed in the requirements.txt file:

streamlit

google-generativeai

python-dotenv

Pillow

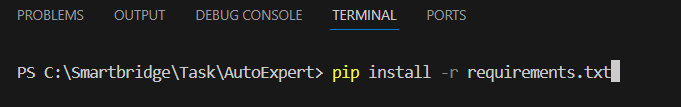
**How to use the file:**

1. **Create the file:** Create a new text file named requirements.txt in the project's root directory.
2. **Add the content:** Copy and paste the lines above into the file.
3. **Install the libraries:** Use the following pip command to install all the required libraries in the Python environment:

bash

pip install -r requirements.txt

### Install the required libraries.



* Open the terminal.
* Run the command: pip install -r requirements.txt
* This command installs all the libraries listed in the requirements.txt file.
* Simply use the command pip install -r requirements. txt in your terminal. This will install all the packages listed in your requirements. txt file.

Method 1: Install PIP using get-pipp.py  
Download PIP with get-pip.py to a folder on your computer. Open a command prompt and navigate to the folder containing the get-pip.py installer. PIP should now be installed successfully. If we receive a “file not found” error, double check the directory path to the file.

**First, let's check whether you already have pip installed:**

1. Open a command prompt by typing cmd into the search bar in the Start menu, and then clicking on Command Prompt: ...
2. Type the following command into the command prompt and press Enter to see if pip is already installed: pip --version.

### Initialization of Google API Key

The Google API key is a secure access token provided by Google, enabling developers to authenticate and interact with various Google APIs. It acts as a form of identification, allowing users to access specific Google services and resources. This key plays a crucial role in authorizing and securing API requests, ensuring that only authorized users can access and utilize Google's services.

To initialize a Google API key for the Gemini family of models, visit [Google AI Studio](https://aistudio.google.com/), create a new project, and select "Get API Key". Copy the key, which starts with "AIza", and store it securely in your application. Use environment variables like GOOGLE\_API\_KEY to prevent unauthorized access.

**Steps for Initialization:**

* **Generate Key:** Go to Google AI Studio and click "Get API Key" to create a key in a new or existing project.
* **Secure Storage:** Save the key as an environment variable (export GOOGLE\_API\_KEY=your\_key) or in a .env file.
* **Configure Application:** Paste the key into your application's settings or backend configuration panel.
* **Verify Access:** Use the API key in your code (Python, JS, etc.) to start making requests, using models like Gemini 1.5 Flash.

For Android Studio users, configure the key via **File > Settings > Tools > AI > Model Providers**.

**Generate Google API Key**

To generate a Google API key for use with the Gemini API, visit [Google AI Studio](https://aistudio.google.com/). The process generates the key automatically.

**Steps to Get Your Gemini API Key**

1. **Sign in to your Google Account:** Go to the Google AI Studio website and sign in with your Google credentials.
2. **Navigate to the "Get API key" section:** You should see a "Get API key" option in the left sidebar or a button on the main dashboard.
3. **Review and accept terms (if prompted):** You might need to accept Google's Generative AI terms of service and confirm your region.
4. **Create the API key:** Click **"Create API key"**. You will be given two options:
   * **Create key in a new project:** This is recommended for beginners and sets up a new Google Cloud project.
   * **Create key in an existing project:** Use this if you have a Google Cloud project for your app.

**Copy and save your key:** Your API key will be displayed. **Copy this key immediately** and store it securely (e.g., an environment variable or a password manager). Never commit your keys directly to version control (like Git).

**Important Security Note**

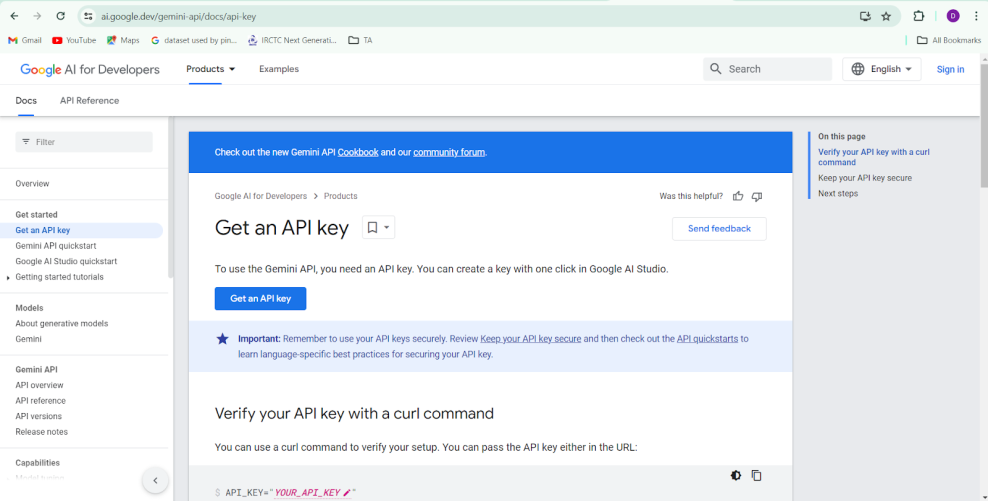
For better security, consider these points:

* **Restricting the key:** In the Google Cloud console, you can add application restrictions (e.g., limit usage by IP address or mobile apps) and API restrictions (limit to only Gemini/Generative AI APIs) to prevent unauthorized use.
* **Using environment variables:** Use an environment variable (e.g., GEMINI\_API\_KEY) instead of hard coding the key in your app.

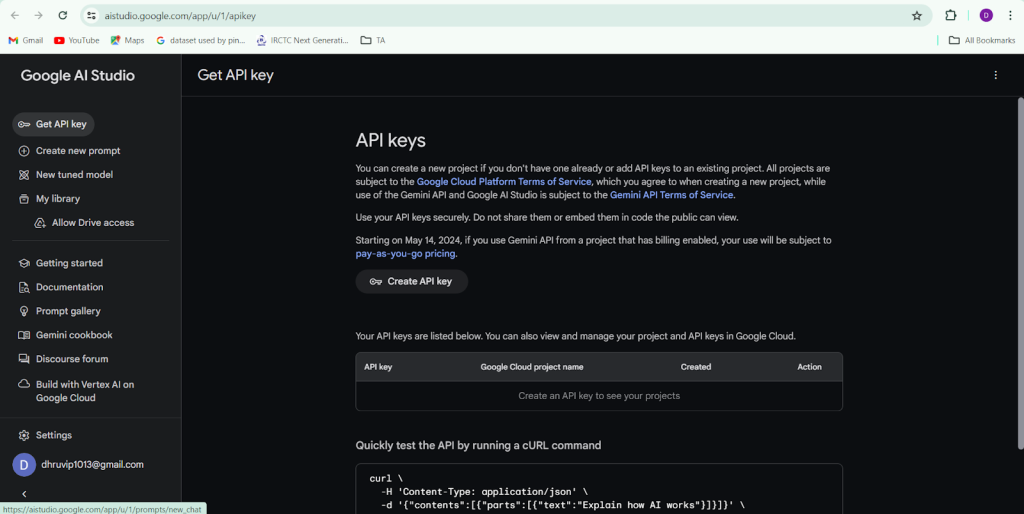
For production-level applications, Google recommends migrating to Vertex AI which offers enhanced security, monitoring, and higher rate limits.

Click the provided link to access the following webpage.

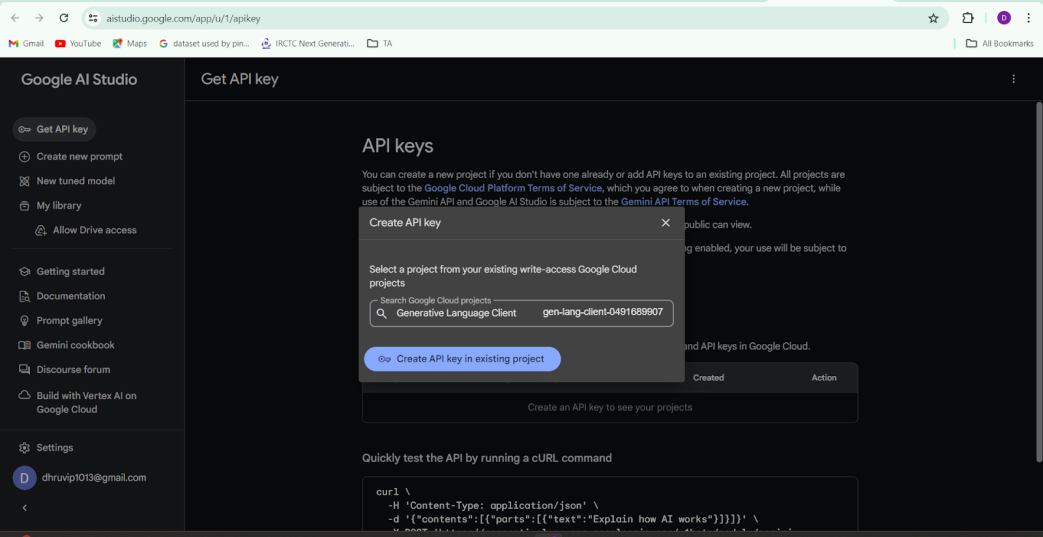
Link: <https://ai.google.dev/gemini-api/docs/api-key>



After signing in to your account, navigate to the 'Get an API Key' option. Clicking on this option will redirect you to another webpage as shown below.



Next, click on 'Create API Key' and choose the generative language client as the project. Then, select 'Create API key in existing project'.



Copy the newly generated API key as it is required for loading the Gemini Pro pre-trained model.

Above the following link:

# Using Gemini API keys

To use the Gemini API, you need an API key. This page outlines how to create and manage your keys in Google AI Studio as well as how to set up your environment to use them in your code.

[Create or view a Gemini API Key](https://aistudio.google.com/app/apikey)

## API Keys

You can create and manage all your Gemini API Keys from the [Google AI Studio](https://aistudio.google.com/app/apikey) **API Keys** page.

Once you have an API key, you have the following options to connect to the Gemini API:

* [Setting your API key as an environment variable](https://ai.google.dev/gemini-api/docs/api-key#set-api-env-var)
* [Providing your API key explicitly](https://ai.google.dev/gemini-api/docs/api-key#provide-api-key-explicitly)

For initial testing, you can hard code an API key, but this should only be temporary since it's not secure. You can find examples for hard coding the API key in [Providing API key explicitly](https://ai.google.dev/gemini-api/docs/api-key#provide-api-key-explicitly) section.

## Google Cloud projects

[Google Cloud projects](https://cloud.google.com/resource-manager/docs/creating-managing-projects) are fundamental to using Google Cloud services (such as the Gemini API), managing billing, and controlling collaborators and permissions. Google AI Studio provides a lightweight interface to your Google Cloud projects.

If you don't have any projects created yet, you must either create a new project or import one from Google Cloud into Google AI Studio. The **Projects** page in Google AI Studio will display all keys that have sufficient permission to use the Gemini API. Refer to the [import projects](https://ai.google.dev/gemini-api/docs/api-key#import-projects) section for instructions.

### Default project

For new users, after accepting Terms of Service, Google AI Studio creates a default Google Cloud Project and API Key, for ease of use. You can rename this project in Google AI Studio by navigating to **Projects** view in the **Dashboard**, clicking the 3 dots settings button next to a project and choosing **Rename project**. Existing users, or users who already have Google Cloud Accounts won't have a default project created.

## Import projects

Each Gemini API key is associated with a Google Cloud project. By default, Google AI Studio does not show all of your Cloud Projects. You must import the projects you want by searching for the name or project ID in the **Import Projects** dialog. To view a complete list of projects you have access to, visit the Cloud Console.

If you don't have any projects imported yet, follow these steps to import a Google Cloud project and create a key:

1. Go to [Google AI Studio](https://aistudio.google.com/).
2. Open the **Dashboard** from the left side panel.
3. Select **Projects**.
4. Select the **Import projects** button in the **Projects** page.
5. Search for and select the Google Cloud project you want to import and select the **Import** button.

Once a project is imported, go to the **API Keys** page from the **Dashboard** menu and create an API key in the project you just imported.

**Note:** For existing users, the keys are pre-populated in the imports pane based on the last 30-days of activity in AI Studio.

**Limitations**

The following are limitations of managing API keys and Google Cloud projects in Google AI Studio.

* You can create a maximum of 10 project at a time from the Google AI Studio **Projects** page.
* You can name and rename projects and keys.
* The **API keys** and **Projects** pages display a maximum of 100 keys and 50 projects.
* Only API keys that have no restrictions, or are restricted to the Generative Language API are displayed.

For additional management access to your projects, visit the Google Cloud Console.

## Setting the API key as an environment variable

If you set the environment variable GEMINI\_API\_KEY or GOOGLE\_API\_KEY, the API key will automatically be picked up by the client when using one of the [Gemini API libraries](https://ai.google.dev/gemini-api/docs/libraries). It's recommended that you set only one of those variables, but if both are set, GOOGLE\_API\_KEY takes precedence.

If you're using the REST API, or JavaScript on the browser, you will need to provide the API key explicitly.

Here is how you can set your API key locally as the environment variable GEMINI\_API\_KEY with different operating systems.

[Linux/macOS - Bash](https://ai.google.dev/gemini-api/docs/api-key#linuxmacos---bash)[macOS - Zsh](https://ai.google.dev/gemini-api/docs/api-key#macos---zsh)[Windows](https://ai.google.dev/gemini-api/docs/api-key#windows)

Bash is a common Linux and macOS terminal configuration. You can check if you have a configuration file for it by running the following command:

~/.bashrc

If the response is "No such file or directory", you will need to create this file and open it by running the following commands, or use zsh:

touch ~/.bashrc

open ~/.bashrc

Next, you need to set your API key by adding the following export command:

export GEMINI\_API\_KEY=<YOUR\_API\_KEY\_HERE>

After saving the file, apply the changes by running:

source ~/.bashrc

## Providing the API key explicitly

In some cases, you may want to explicitly provide an API key. For example:

* You're doing a simple API call and prefer hard coding the API key.
* You want explicit control without having to rely on automatic discovery of environment variables by the Gemini API libraries
* You're using an environment where environment variables are not supported (e.g web) or you are making REST calls.

Below are examples for how you can provide an API key explicitly:

[Python](https://ai.google.dev/gemini-api/docs/api-key#python)[JavaScript](https://ai.google.dev/gemini-api/docs/api-key#javascript)[Go](https://ai.google.dev/gemini-api/docs/api-key#go)[Java](https://ai.google.dev/gemini-api/docs/api-key#java)[REST](https://ai.google.dev/gemini-api/docs/api-key#rest)

from google import genai

client = genai.Client(api\_key="YOUR\_API\_KEY")

response = client.models.generate\_content(

model="gemini-3-flash-preview", contents="Explain how AI works in a few words"

)

print(response.text)

## Keep your API key secure

Treat your Gemini API key like a password. If compromised, others can use your project's quota, incur charges (if billing is enabled), and access your private data, such as files.

### Critical security rules

* **Keep keys confidential**: API keys for Gemini may access sensitive data your application depends upon.
  + **Never commit API keys to source control.** Do not check your API key into version control systems like Git.
  + **Never expose API keys on the client-side.** Do not use your API key directly in web or mobile apps in production. Keys in client-side code (including our JavaScript/TypeScript libraries and REST calls) can be extracted.
* **Restrict access**: Restrict API key usage to specific IP addresses, HTTP referrers, or Android/iOS apps where possible.
* **Restrict usage**: Enable only the necessary APIs for each key.
* **Perform regular audits**: Regularly audit your API keys and rotate them periodically.

### Best practices

* **Use server-side calls with API keys** The most secure way to use your API key is to call the Gemini API from a server-side application where the key can be kept confidential.
* **Use ephemeral tokens for client-side access (Live API only):** For direct client-side access to the Live API, you can use ephemeral tokens. They come with lower security risks and can be suitable for production use. Review [ephemeral tokens](https://ai.google.dev/gemini-api/docs/ephemeral-tokens) guide for more information.
* **Consider adding restrictions to your key:** You can limit a key's permissions by adding [API key restrictions](https://cloud.google.com/api-keys/docs/add-restrictions-api-keys#add-api-restrictions). This minimizes the potential damage if the key is ever leaked.

For some general best practices, you can also review this [support article](https://support.google.com/googleapi/answer/6310037).

Except as otherwise noted, the content of this page is licensed under the [Creative Commons Attribution 4.0 License](https://creativecommons.org/licenses/by/4.0/), and code samples are licensed under the [Apache 2.0 License](https://www.apache.org/licenses/LICENSE-2.0). For details, see the [Google Developers Site Policies](https://developers.google.com/site-policies). Java is a registered trademark of Oracle and/or its affiliates.

Last updated 2026-01-19 UTC.

# Using Gemini API keys

To use the Gemini API, you need an API key. This page outlines how to create and manage your keys in Google AI Studio as well as how to set up your environment to use them in your code.

[**Create or view a Gemini API Key**](https://aistudio.google.com/app/apikey)

**API Keys**

You can create and manage all your Gemini API Keys from the [Google AI Studio](https://aistudio.google.com/app/apikey) **API Keys** page.

Once you have an API key, you have the following options to connect to the Gemini API:

* [Setting your API key as an environment variable](https://ai.google.dev/gemini-api/docs/api-key#set-api-env-var)
* [Providing your API key explicitly](https://ai.google.dev/gemini-api/docs/api-key#provide-api-key-explicitly)

For initial testing, you can hard code an API key, but this should only be temporary since it's not secure. You can find examples for hard coding the API key in [Providing API key explicitly](https://ai.google.dev/gemini-api/docs/api-key#provide-api-key-explicitly) section.

## Google Cloud projects

[Google Cloud projects](https://cloud.google.com/resource-manager/docs/creating-managing-projects) are fundamental to using Google Cloud services (such as the Gemini API), managing billing, and controlling collaborators and permissions. Google AI Studio provides a lightweight interface to your Google Cloud projects.

If you don't have any projects created yet, you must either create a new project or import one from Google Cloud into Google AI Studio. The **Projects** page in Google AI Studio will display all keys that have sufficient permission to use the Gemini API. Refer to the [import projects](https://ai.google.dev/gemini-api/docs/api-key#import-projects) section for instructions.

For new users, after accepting Terms of Service, Google AI Studio creates a default Google Cloud Project and API Key, for ease of use. You can rename this project in Google AI Studio by navigating to **Projects** view in the **Dashboard**, clicking the 3 dots settings button next to a project and choosing **Rename project**. Existing users, or users who already have Google Cloud Accounts won't have a default project created.

## Import projects

Each Gemini API key is associated with a Google Cloud project. By default, Google AI Studio does not show all of your Cloud Projects. You must import the projects you want by searching for the name or project ID in the **Import Projects** dialog. To view a complete list of projects you have access to, visit the Cloud Console.

If you don't have any projects imported yet, follow these steps to import a Google Cloud project and create a key:

1. Go to [Google AI Studio](https://aistudio.google.com/).
2. Open the **Dashboard** from the left side panel.
3. Select **Projects**.
4. Select the **Import projects** button in the **Projects** page.
5. Search for and select the Google Cloud project you want to import and select the **Import** button.

Once a project is imported, go to the **API Keys** page from the **Dashboard** menu and create an API key in the project you just imported.

**Note:** For existing users, the keys are pre-populated in the imports pane based on the last 30-days of activity in AI Studio.

## Limitations

The following are limitations of managing API keys and Google Cloud projects in Google AI Studio.

* You can create a maximum of 10 project at a time from the Google AI Studio **Projects** page.
* You can name and rename projects and keys.
* The **API keys** and **Projects** pages display a maximum of 100 keys and 50 projects.
* Only API keys that have no restrictions, or are restricted to the Generative Language API are displayed.

For additional management access to your projects, visit the Google Cloud Console.

## Setting the API key as an environment variable

If you set the environment variable GEMINI\_API\_KEY or GOOGLE\_API\_KEY, the API key will automatically be picked up by the client when using one of the [Gemini API libraries](https://ai.google.dev/gemini-api/docs/libraries). It's recommended that you set only one of those variables, but if both are set, GOOGLE\_API\_KEY takes precedence.

If you're using the REST API, or JavaScript on the browser, you will need to provide the API key explicitly.

Here is how you can set your API key locally as the environment variable GEMINI\_API\_KEY with different operating systems.

[Linux/macOS - Bash](https://ai.google.dev/gemini-api/docs/api-key#linuxmacos---bash)[macOS - Zsh](https://ai.google.dev/gemini-api/docs/api-key#macos---zsh)[Windows](https://ai.google.dev/gemini-api/docs/api-key#windows)

Bash is a common Linux and macOS terminal configuration. You can check if you have a configuration file for it by running the following command:

~/.bashrc

If the response is "No such file or directory", you will need to create this file and open it by running the following commands, or use zsh:

touch ~/.bashrc

open ~/.bashrc

Next, you need to set your API key by adding the following export command:

export GEMINI\_API\_KEY=<YOUR\_API\_KEY\_HERE>

After saving the file, apply the changes by running:

source ~/.bashrc

## Providing the API key explicitly

In some cases, you may want to explicitly provide an API key. For example:

* You're doing a simple API call and prefer hard coding the API key.
* You want explicit control without having to rely on automatic discovery of environment variables by the Gemini API libraries
* You're using an environment where environment variables are not supported (e.g web) or you are making REST calls.

Below are examples for how you can provide an API key explicitly:

[Python](https://ai.google.dev/gemini-api/docs/api-key#python)[JavaScript](https://ai.google.dev/gemini-api/docs/api-key#javascript)[Go](https://ai.google.dev/gemini-api/docs/api-key#go)[Java](https://ai.google.dev/gemini-api/docs/api-key#java)[REST](https://ai.google.dev/gemini-api/docs/api-key#rest)

from google import genai

client = genai.Client(api\_key="YOUR\_API\_KEY")

response = client.models.generate\_content(

model="gemini-3-flash-preview", contents="Explain how AI works in a few words"

)

print(response.text)

## Keep your API key secure

Treat your Gemini API key like a password. If compromised, others can use your project's quota, incur charges (if billing is enabled), and access your private data, such as files.

### Critical security rules

* **Keep keys confidential**: API keys for Gemini may access sensitive data your application depends upon.
  + **Never commit API keys to source control.** Do not check your API key into version control systems like Git.
  + **Never expose API keys on the client-side.** Do not use your API key directly in web or mobile apps in production. Keys in client-side code (including our JavaScript/TypeScript libraries and REST calls) can be extracted.
* **Restrict access**: Restrict API key usage to specific IP addresses, HTTP referrers, or Android/iOS apps where possible.
* **Restrict usage**: Enable only the necessary APIs for each key.
* **Perform regular audits**: Regularly audit your API keys and rotate them periodically.

Best practices

* **Use server-side calls with API keys** The most secure way to use your API key is to call the Gemini API from a server-side application where the key can be kept confidential.
* **Use ephemeral tokens for client-side access (Live API only):** For direct client-side access to the Live API, you can use ephemeral tokens. They come with lower security risks and can be suitable for production use. Review [ephemeral tokens](https://ai.google.dev/gemini-api/docs/ephemeral-tokens) guide for more information.
* **Consider adding restrictions to your key:** You can limit a key's permissions by adding [API key restrictions](https://cloud.google.com/api-keys/docs/add-restrictions-api-keys#add-api-restrictions). This minimizes the potential damage if the key is ever leaked.

**Initialize Google API Key**

https://lh7-rt.googleusercontent.com/docsz/AD_4nXcrmS-liCqA9g7OOBurz7SMerE4WkFyk6nEOk0Jwpy7hs5ZNe2HInjdR5H15fVHBdn78OBj1WToHCLFa6NPKOISG3oad9R_mrlx3xS5Mx-kllfddQl2Ov3EIEX6FhV_-56WYAoYZdmpn0Cm6dFjoRWkqUF9c7_kSCqs5HCSvQ?key=qyHGmoWfl_Ox6hwdlDCQPg

* Create a .env file and define a variable named GOOGLE\_API\_KEY.
* Assign the copied Google API key to this variable.
* Paste the API key obtained from the previous steps here.

To initialize a Google API key for a Gemini app, create a .env file in the project's root directory. Then, define a GOOGLE\_API\_KEY variable, assign the key to this variable, and access it in the code using an environment variable loading library.

**1. Create a .env file**

Create a file named .env in the root directory of the project.

**2. Define the GOOGLE\_API\_KEY variable**

Open the .env file and add the variable in the format VARIABLE\_NAME=YOUR\_API\_KEY\_HERE. Do not use quotation marks.

GOOGLE\_API\_KEY=AIzaSyB...your-api-key-string-here

**Important Security Note:** Add .env to the .gitignore file to prevent committing the secret key to version control.

**3. Access the API Key in Code**

To access the key in an application, a library is needed that loads environment variables from the .env file into the application's environment variables at runtime. A common option for Python is the python-dotenv library.

* **Install the library** (if needed):

bash

pip install python-dotenv

*# or npm install dotenv for Node.js*

* **Load and use the key in code**:

python

import os

from dotenv import load\_dotenv, find\_dotenv

*# Load environment variables from .env file*

load\_dotenv(find\_dotenv())

*# Access the API key*

api\_key = os.getenv("GOOGLE\_API\_KEY")

*# Use the key to initialize your Google/Gemini client*

*# For example (using the google-genai library):*

*# from google import genai*

*# client = genai.Client(api\_key=api\_key)*

This method securely retrieves the key without exposing it directly in the source code.

**Get a Google API Key**

If an API key has not been obtained, create one from the [Google AI Studio API Keys page](https://aistudio.google.com/app/apikey) or the Google Cloud Console.

### Interfacing with Pre-trained Model

To interface with the pre-trained model, we'll start by creating an app.py file, which will contain both the model and Streamlit UI code.

Interfacing a pre-trained model with an application involves using [Google AI Studio](https://aistudio.google.com/) or Vertex AI to integrate advanced language model capabilities. This can be done with SDKs, function calling, and multimodal features for understanding text and images.

Here is how to interface with the Gemini family of models for app development:

**1. Set Up the Environment and Authentication**

* **Get an API Key:** Sign in to Google AI Studio to generate an API key.
* **Install SDK:** Use pip install -U google-generativeai for AI Studio or google-cloud-aiplatform for Vertex AI.
* **Authentication:** Set up Application Default Credentials (ADC) to authenticate your local development environment.

**2. Core Integration Methods**

* **Vertex AI (Enterprise):** Suitable for building secure, scalable applications with enterprise-grade privacy and features.
* **Google AI Studio (Prototyping):** Ideal for faster development and easier integration via API keys.
* **AutoGen Framework:** You can use Gemini in AutoGen by installing pip install autogen-agentchat[gemini,retrievechat,lmm]~=0.2.

**3. Key Functionalities for Applications**

* **Function Calling:** Instruct the Gemini family of models to call external tools or APIs to retrieve real-time data.
* **Multimodal Input:** Pass images, video, audio, or text to Gemini 1.5 Pro to analyze or summarize content.
* **Streaming Responses:** Implement real-time interaction, allowing the app to display text as it is generated.
* **Structured Output:** Use Pydantic to structure the model's output into specific formats (e.g., JSON).

**4. Implementation Example (Python)**

python

import google.generativeai as genai

*# Configure with your API Key*

genai.configure(api\_key="YOUR\_API\_KEY")

*# Initialize the model*

model = genai.GenerativeModel('gemini-1.5-flash')

*# Generate a response*

response = model.generate\_content("Analyze this user query for the sage app...")

print(response.text)

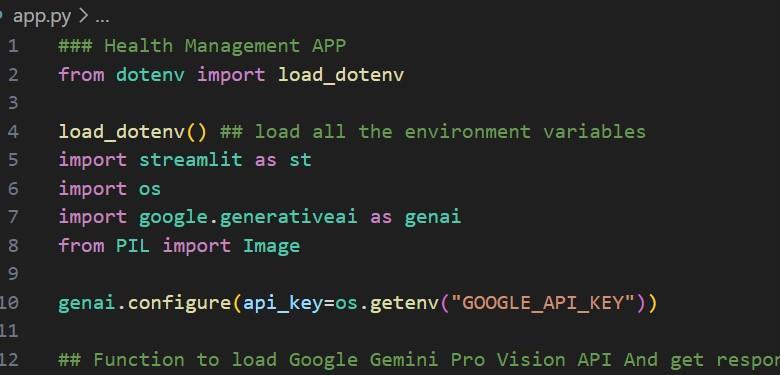
**5. Deployment**

* **Cloud Run:** Deploy your application to Google Cloud Run to make the AI-powered tools accessible.
* **Firebase:** Implement auto-tagging or intelligent features using Firebase AI Logic.

**6. Fine-Tuning (Optional)**

If the pre-trained model does not meet specific domain needs, you can tune Gemini models using supervised fine-tuning with a custom dataset on Vertex AI.

**Load the Gemini Pro API**



This code snippet is for initializing a Elite Species App using Streamlit, an open-source app framework, and Google Generative AI services. The script starts by loading environment variables from a .env file using the load\_dotenv() function from the dotenv package. It then imports necessary libraries: streamlit for creating the web app interface, os for accessing environment variables, google.generativeai for utilizing Google's Generative AI capabilities, and PIL.Image for image processing. The genai.configure() function is called to set up the Google Generative AI API with the API key retrieved from the environment variables, ensuring secure and authorized access to the AI services.

To use the Gemini Pro API, you will build a custom application. "Auto sage app" is not a specific, pre-built Google product. The process involves setting up a project, obtaining an API key, and using the SDK in your code.

Here are the general steps to build an app that uses the Gemini API for features like code completion or content suggestions:

1. **Get a Gemini API Key**: Get your API key from Google AI Studio. This key is necessary to authenticate requests to the Gemini Pro model.
2. **Set Up Your Development Environment**: You will need to set up your project depending on your desired platform (web, Android, etc.) and programming language.
   * **For Web Apps**: You can use the Firebase Studio Gemini API template to get a working environment with minimal setup.
   * **For Android Apps**: Use Gemini in Android Studio and add the necessary Gradle dependency to your project.
   * **For Other Environments (Python, Node.js, etc.)**: Install the appropriate Google GenAI SDK (e.g., pip install google-genai for Python, npm install @google/genai for Node.js).

**Configure the API Key**: It is a best practice to set your API key as an environment variable (e.g., GEMINI\_API\_KEY) so that the client library can automatically pick it up, or you can pass it explicitly when initializing the client.

**Integrate the Gemini Pro Model**: Use the SDK to instantiate the gemini-2.5-pro (or a similar Pro model) and make calls for content generation, code completion, or other "auto sage" functionalities.

* + For example, in Python:

python

from google import genai

client = genai.Client(api\_key="YOUR\_API\_KEY") *# Or rely on environment variable*

*# Use a Pro model*

model = "gemini-2.5-pro"

response = client.models.generate\_content(

model=model,

contents="Your prompt for auto-suggestion or content generation goes here."

)

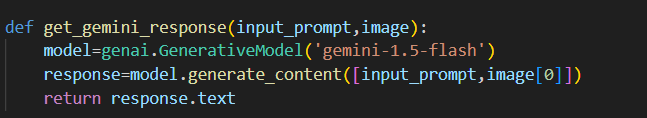
print(response.text)

In the application code, you would use functions like generateContent() or manage multi-turn conversations using startChat().

1. **Develop the "Auto Sage" Feature**: The "auto sage" functionality is built by designing effective prompts and potentially using tools like code execution or Google Search to ground the responses in real-time information. For instance, you could design a prompt in Google AI Studio to provide code suggestions based on user input, and then integrate that logic into your application.

For more specific guidance, you can explore the Gemini API documentation for detailed examples and best practices.

**Implement a function to get gemini response**



* The function get\_gemini\_response takes an input\_prompt and image as a parameter.
* It calls the generate\_content method of the model object to generate a response.
* The generated response is returned as text.

To implement a function in an app that receives an automated response, use the [Gemini API](https://ai.google.dev/) with function calling. This allows the model to decide when to use a predefined function based on the user's prompt. This provides an "auto sage" effect.

This process typically involves these steps, often done on a server-side backend:

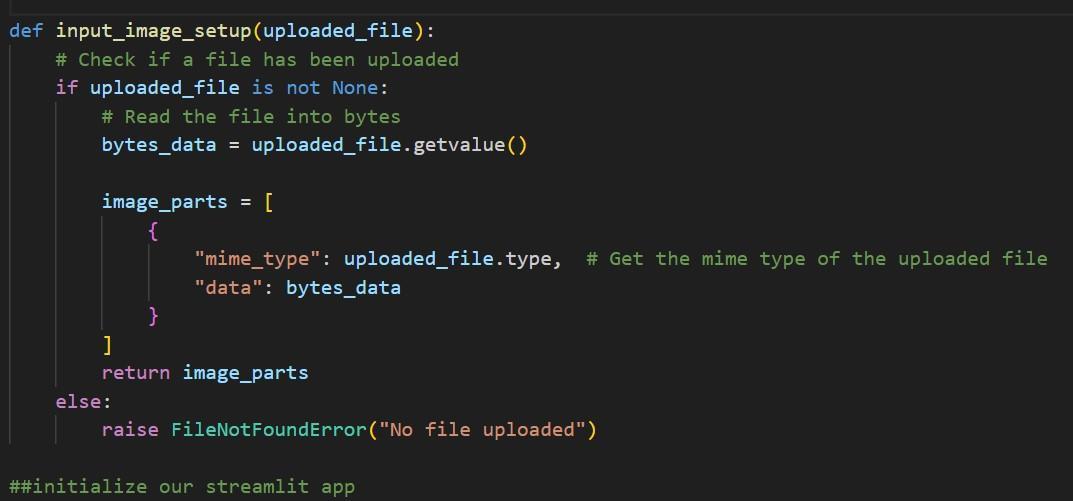
**Implementation Steps**

1. **Set Up and Get an API Key:** Create a project and obtain a Gemini API key from Google AI Studio. Use a backend server or Firebase AI Logic for secure access in production.
2. **Define a Function:** Write the function in the app's code to perform the desired task, such as fetching data.
3. **Create a Function Declaration:** Define a schema (like JSON Schema) that describes the function and its parameters for the model.
4. **Initialize the Model with the Function:** Initialize the Gemini client with the function declaration as a tool. Set toolConfig to AUTO mode so the model can choose between generating text or calling the function.
5. **Handle Function Calls:** Send the user's prompt to the model. If the response indicates a function call, execute the function with the model's arguments. Then, send the output back to the model for a final response.

**Example (JavaScript/Node.js SDK)**

The code example demonstrates this flow using the @google/genai SDK on a server. It shows how to define a fetchWeatherData function, create a corresponding schema, initialize the model with this tool, and handle the interaction loop where the model calls the function and the function's output is returned to the model. This setup allows the app to automatically determine and execute necessary actions based on user requests, enabling "auto sage" capabilities. The full code example can be found in the original response.

**Implement a function to read the Image and set the image format for Gemini Pro model Input**



The function input\_image\_setup processes an uploaded image file for Elite Species application. It first checks if a file has been uploaded. If a file is present, it reads the file's content into bytes and creates a dictionary containing the file's MIME type and its byte data. This dictionary is then stored in a list named image\_parts, which is returned by the function. If no file is uploaded, the function raises a FileNotFoundError, indicating that an image file is required but not provided. This setup ensures that the uploaded image is correctly formatted and ready for further processing or analysis in the application.

A Python function can be implemented to read an image and prepare it as input for a Gemini Pro model. The PIL (Pillow) library can be used to open the image and pass it directly to the generate\_content method of the Gemini API.

**Prerequisites**

Install the necessary libraries:

bash

pip install google-generativeai Pillow

**Python Function**

The google-generativeai library handles image files directly using the PIL.Image object. Manual image format conversion to Base64 is unnecessary for smaller files (under 20MB).

python

import google.generativeai as genai

from PIL import Image

import os

*# Configure the API key (ensure you have the 'GOOGLE\_API\_KEY' environment variable set)*

*# You can get an API key from*

[python](https://aistudio.google.com/" \t "_blank)

[Google AI Studio](https://aistudio.google.com/" \t "_blank)

python

:

python

https://aistudio.google.com/

python

genai.configure(api\_key=os.getenv('GOOGLE\_API\_KEY'))

def analyze\_image\_with\_gemini(image\_path: str, prompt: str):

"""

Reads an image, prepares it for the Gemini Pro model, and generates content.

Args:

image\_path: The file path to the image.

prompt: The text prompt to accompany the image.

Returns:

The text response from the Gemini model.

"""

try:

*# Open the image using the Pillow library*

img = Image.open(image\_path)

*# Initialize the Gemini Pro Vision model*

*# Note: 'gemini-pro-vision' is an older model; consider using newer models like 'gemini-2.5-flash'*

*# which also support multimodal input.*

model = genai.GenerativeModel('gemini-2.5-flash')

*# Pass the image and the prompt to the model*

*# The SDK automatically handles the image input format*

response = model.generate\_content([prompt, img])

return response.text

except FileNotFoundError:

return f"Error: Image file not found at {image\_path}"

except Exception as e:

return f"An error occurred: {e}"

*# --- Example Usage ---*

if \_\_name\_\_ == "\_\_main\_\_":

*# Replace 'path/to/your/image.jpg' with your actual image file path*

my\_image\_path = 'sample\_image.jpg'

my\_prompt = "Describe the main objects and action in this image in detail."

response\_text = analyze\_image\_with\_gemini(my\_image\_path, my\_prompt)

print(response\_text)

**Supported Image Formats**

The Gemini models support these image MIME types:

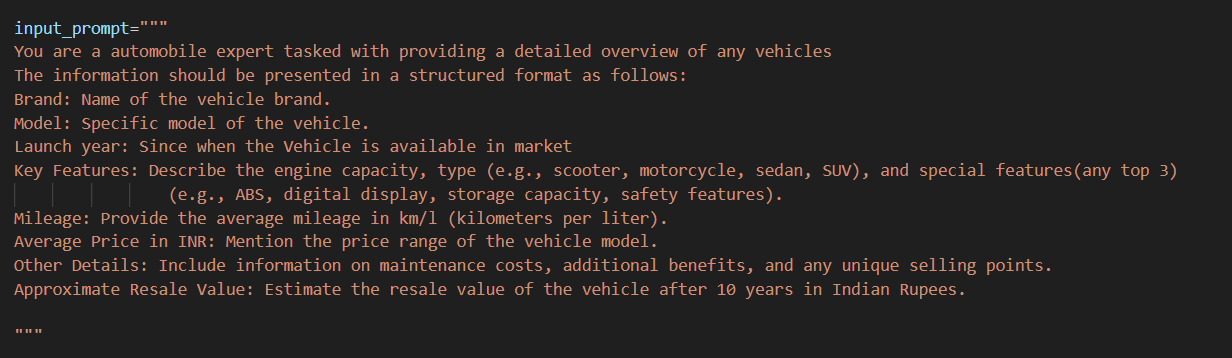
* image/png
* image/jpeg
* image/webp

**Alternative for Large Files (File API)**

If images exceed the 20MB limit, the genai.files.upload method (File API) is recommended. This uploads the file to the cloud and references it by a file URI in the prompt. The file is stored for 48 hours and can be reused.

For more details on handling images, refer to the Gemini API documentation on image understanding.

**Write a prompt for gemini model**



The variable input\_prompt is a multi-line string designed as a prompt for a Auto Sage App model. It instructs the model to analyze an image of Vehicle, identify different attributes like- Brand, Key features, Mileage, and other details as well. This prompt is likely used in conjunction with an AI service that can process images and generate vehicle information based on the visual data provided.

**Key Principles for Writing Prompts**

* **Be Clear & Specific**: Avoid vague keywords; use full sentences and explain exactly what you need (e.g., "Write a short poem about nature" vs. "Write a short, rhyming poem about the beauty of nature in winter, focusing on sensory details").
* **Provide Context**: Give background information, the purpose, and the target audience to guide Gemini's response.
* **Define Persona**: Tell Gemini who it should act as (e.g., "You are a marketing consultant").
* **Specify the Task**: Use strong action verbs (summarize, create, rephrase, list).
* **Set the Format**: Dictate the output structure (e.g., "as a bulleted list," "in a table with two columns").
* **Use Iteration**: Treat it as a conversation; refine your prompts with follow-ups if the first answer isn't perfect.
* **Break Down Complexity**: For big tasks, use multiple prompts to build towards a final solution.

A prompt is an instruction or question given to a person or AI to elicit a specific response, like "Write a short email to [Name] about [Topic]" or "Explain quantum physics in simple terms," serving as a starting point for writing, problem-solving, or creative tasks. Examples range from simple requests like "List the first five US Presidents" to complex instructions for AI, such as guiding it to analyze data or generate creative content with specific constraints.

**Simple Prompts (Good for Basics)**

* **For AI/Writing:** "Create a to-do list for planning a summer vacation."
* **For Information:** "What are the main causes of climate change?"
* **For Email:** "Draft a polite email requesting a meeting with my manager."

**Detailed/Contextual Prompts (Better for Specific Outputs)**

**For Analysis:**

"Analyze my monthly budget for June and July and suggest three ways I can cut expenses to save $100 next month."

* **For Creative Writing:**

"Write a short story about a magical book that grants wishes but with unexpected, humorous consequences, told from the perspective of the book itself."

* **For Step-by-Step Guidance:**

"Explain the process of photosynthesis step-by-step, focusing on the role of chlorophyll and sunlight."

**Key Elements of Effective Prompts**

* **Context:** Who is the audience? (e.g., a child, a professional)
* **Format:** What should the output look like? (e.g., a list, an email, a poem)
* **Tone:** What feeling should it convey? (e.g., formal, funny, serious)
* **Constraints:** What should it not do? (e.g., "Don't use jargon," "Keep it under 200 words")

**The 5 types of prompts!**

Types of Prompts in ABA. A variety of prompts can be used in ABA therapy as part of the learning process. These fall into five main categories: physical, modeling, visual, gestural, and verbal

### Model Deployment

We deploy our model using the Streamlit framework, a powerful tool for building and sharing data applications quickly and easily. With Streamlit, we can create interactive web applications that allow users to interact with our models in real-time, providing an intuitive and seamless experience.

Deploying an application using Gemini involves building with Google AI Studio (or Vertex AI), containerizing the application, and deploying it to services like Google Cloud Run for auto-scaling. For specific uses, Gemini models can be fine-tuned via Vertex AI and deployed to managed endpoints for secure, scalable inference.

**Deployment Steps and Options**

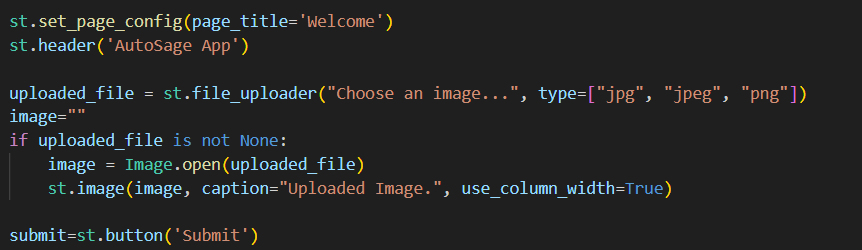
* **Build & Prototype (AI Studio):** Use the Build mode in Google AI Studio to develop the application and test features like Live API.
* **Deployment Target (Cloud Run):** Google Cloud Run is recommended for auto-scaling, serverless deployment of containerized apps.
* **Vertex AI (Production):** For enterprise-grade, fine-tuned models, use Vertex AI to deploy models to managed endpoints.
* **Security & Key Management:** Use Google Secret Manager for managing API keys (e.g., GEMINI\_API\_KEY) rather than hardcoding them.
* **Integration (Agentic Workflows):** Use Gemini with Vertex AI for agentic workflows (e.g., automated customer support, sales insights) to enable real-time data interaction.

**Important Considerations**

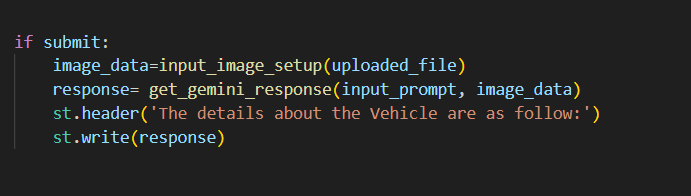
* **Costs:** Tuned models are billed per token, while non-managed endpoints are billed by machine hours.
* **Limitations:** Tuned models currently support shared public endpoints rather than private endpoints.
* **Operational Benefits:** Using Gemini with Google Cloud ensures high security and scalability for AI-driven applications.

**Integrate with Web Framework**

Auto Sage Application:



* If “Tell me about Specimen button click”:

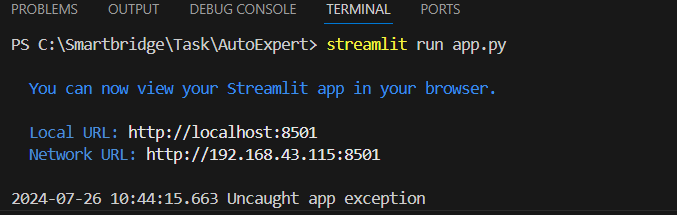


This code initializes a Streamlit application titled "Auto Sage App" by setting the page title and creating the app's header. It includes a file uploader for users to upload an image in JPG, JPEG, or PNG format. If an image is uploaded, it is opened using the PIL library and displayed within the app with a caption. A button labeled "Submit" is also provided, which users can click to trigger the application's functionality for analyzing the uploaded image to understand insightful information about the Vehicle.

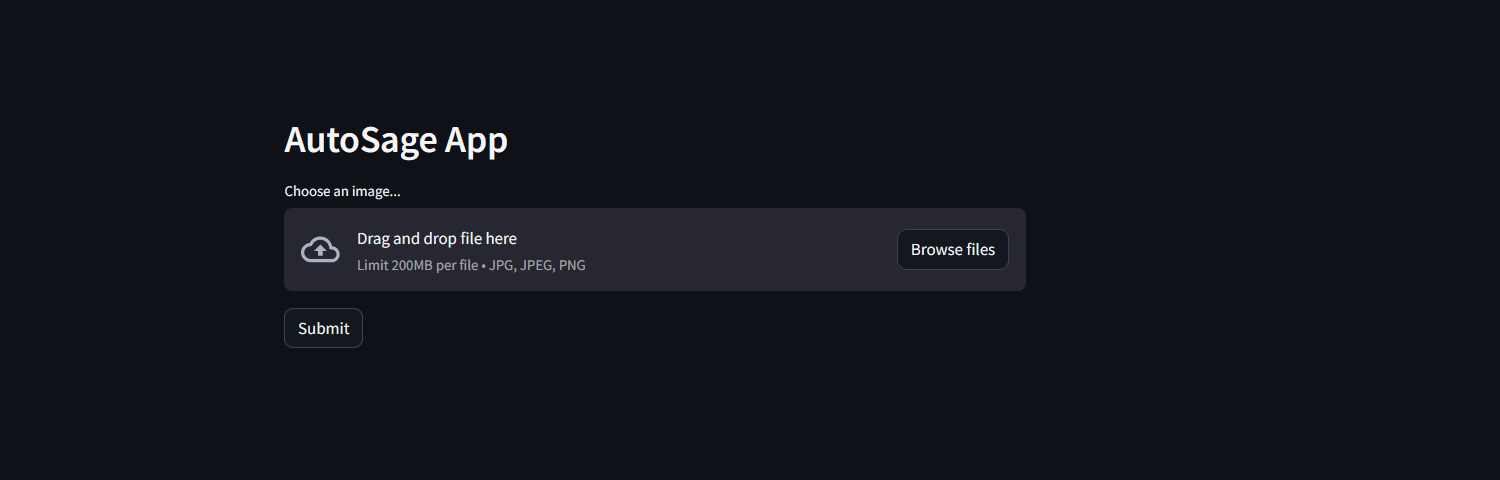
**Host the Application**

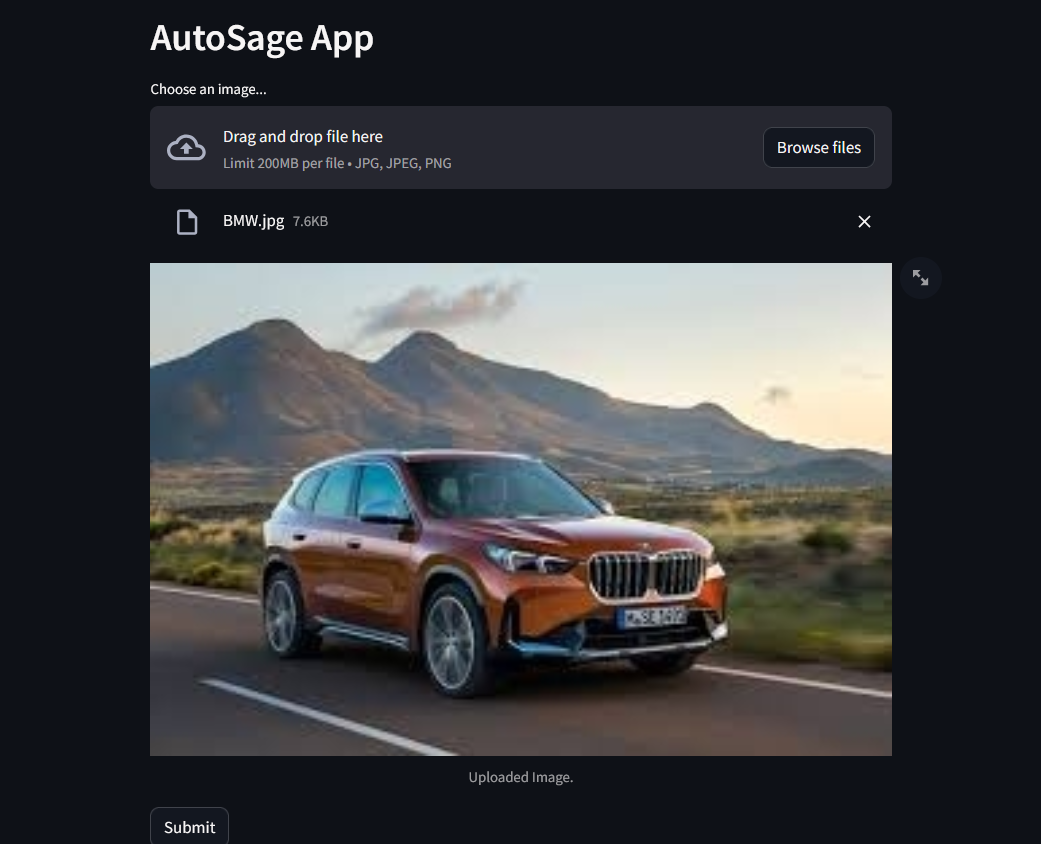
Launching the Application:

* To host the application,  go to the terminal, type - streamlit run app.py
* Here app.py refers to a python script.

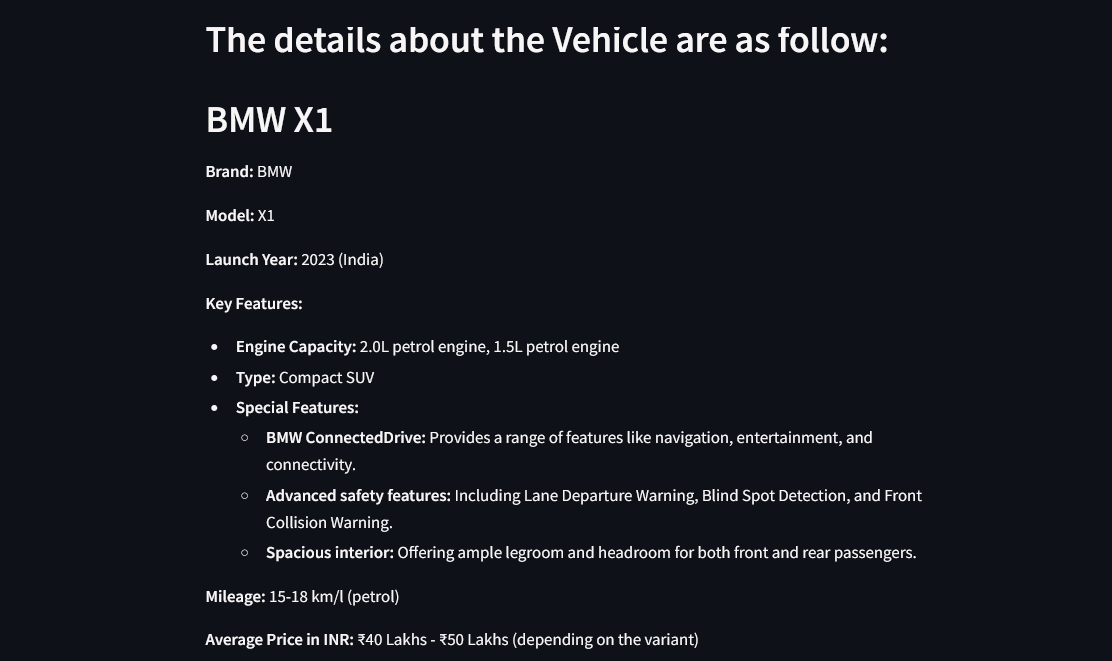


Run the command to get the below results

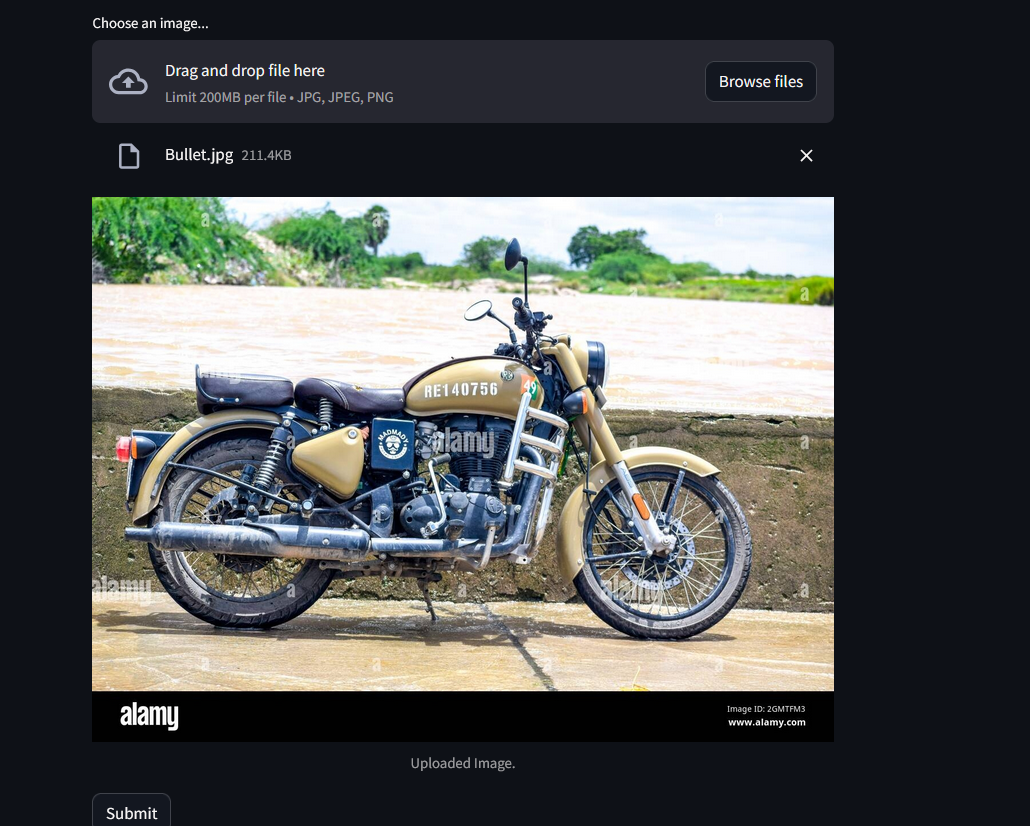


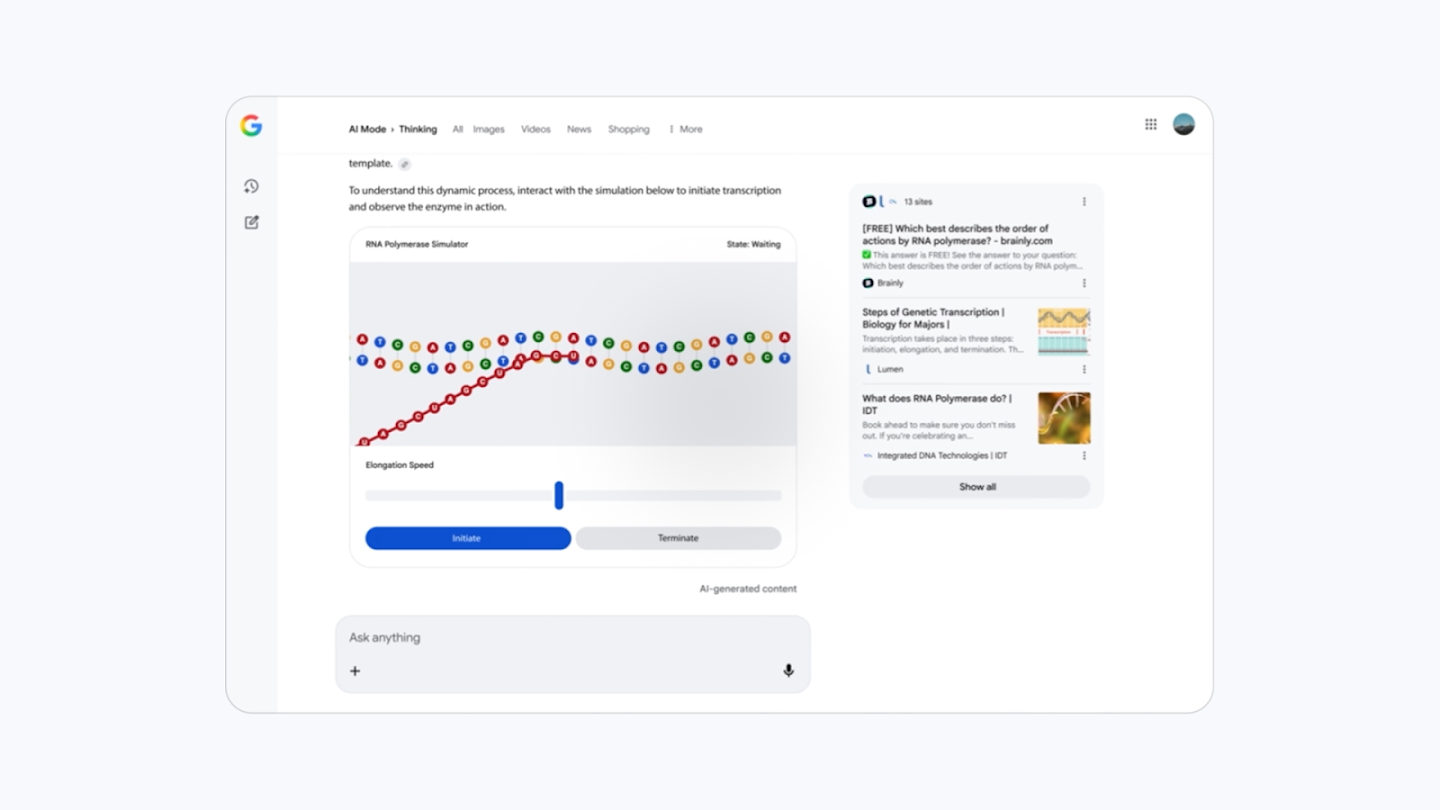


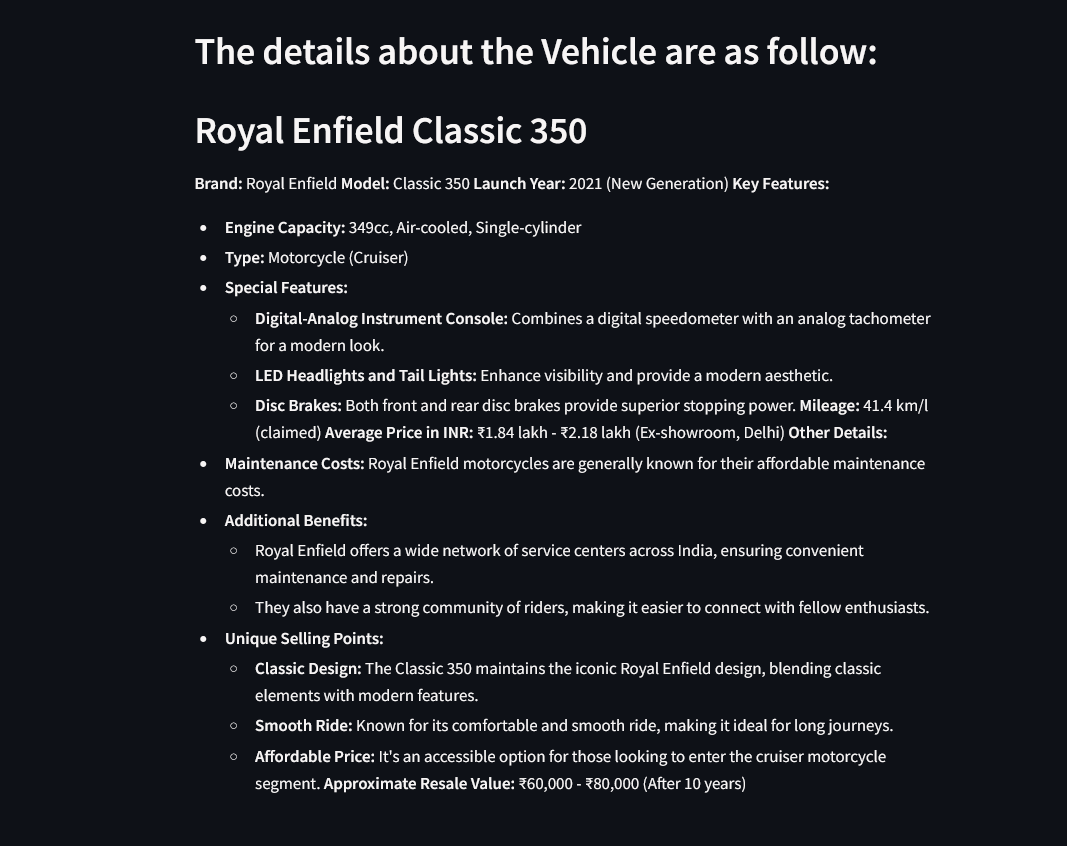
Output 1:



Input 2:





o

**GROUP PHOTO:**

****

DEMO LINK: <https://drive.google.com/file/d/1lMoiBDqRwZYy8l-_aS-GwD3SQ-0iyMdC/view?usp=drivesdk>