gradio

Prithvi Raj Allada and Abdur Rahman

What is gradio and why would a Data Scientist use it?

Life of a Data Scientist

The chief responsibility of a data scientist is to develop solutions using machine learning models for various business problems.

1. 90% of the time cleaning and understanding data
2. 10% of the time demoing machine learning models

During the lifecycle of a machine learning project, data scientists have to overcome different challenges related to the nature of the problem, and many times most of the models developed never see the light of the day.

Additionally, data scientists often lack web development skills, making it hard to prepare models for production use. This gap hinders stakeholders and managers from understanding the project's progress, as it can take weeks or months to present a model in a user-friendly application form.

Gradio can add value to users by making the intermediate outputs of points 1 and 2 easier to share

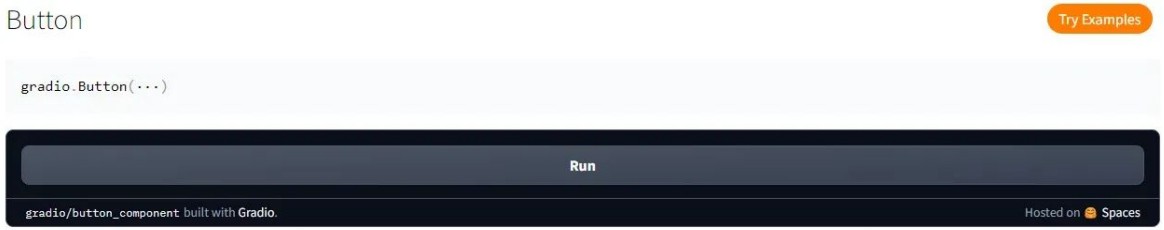
Introduction

Gradio is a Python library that's free and open-source, designed for the swift creation of demos or web applications for machine learning models, APIs, or any Python function. It features easy sharing capabilities, enabling you to distribute a link to your demo or web app in mere seconds through Gradio's sharing options, all without requiring knowledge of JavaScript, CSS, or web hosting.

What can gradio create?

Interfaces(web-based GUI)

Components (ready-to-use components)



Routes (interact with API’s)

Example Usage

```
import gradio as gr
def echo(name, request: gr.Request):
    print("Request headers dictionary:", request.headers)
    print("IP address:", request.client.host)
    return name
io = gr.Interface(echo, "textbox", "textbox").launch()
```

Setup

Gradio installation is fast and easy to setup .You can install Gradio using pip command.

```
pip install gradio
```

Once Installed, import gradio to you python notebook.

```
import gradio as gr
```

Types of Demos

- Interface
- ChatInterface
- TabbedInterface
- Blocks

Types of Methods

- Launch
- Load
- from_pipeline
- Integrate
- queue

load method

This listener is triggered when the Interface initially loads in the browser.

```
gradio.Interface.load(block, ···)
```

- **from_pipeline:** Class method that constructs an Interface from a Hugging Face transformers.Pipeline or diffusers.DiffusionPipeline object. The input and output components are automatically determined from the pipeline.

```
gradio.Interface.from_pipeline(pipeline, ...)
```

- **Integrate**: A catch-all method for integrating with other libraries. This method should be run after launch()

```
gradio.Interface.integrate(...)
```

- **queue**: By enabling the queue you can control when users know their position in the queue, and set a limit on maximum number of events allowed.

```
demo = gr.Interface(image_generator, gr.Textbox(), gr.Image())
demo.queue(max_size=20)
demo.launch()
```

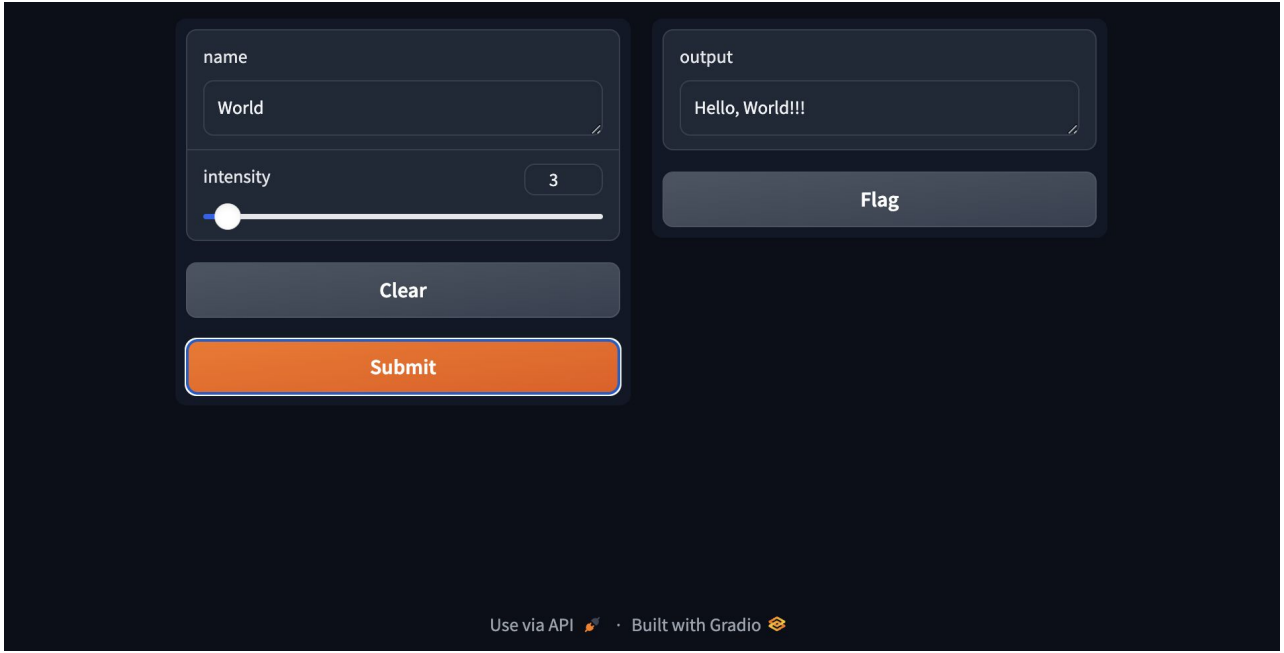
Interface

Designed to create demos for machine learning models which **accepts** one or more inputs and **returns** one or more outputs

- **fn**: the function to wrap a user interface (UI) around
- **inputs**: the Gradio component(s) to use for the input. The number of components should match the number of arguments in your function.
- **outputs**: the Gradio component(s) to use for the output. The number of components should match the number of return values from your function.

The input and output arguments take one or more Gradio components. It includes more than 30 built-in components (such as the gr.Textbox(), gr.Image(), and gr.HTML() components) that are designed for machine learning applications.

First Demo!


name
World
intensity
3
Clear
Submit
output
Hello, World!!!
Flag
Use via API
· Built with Gradio

First Demo!

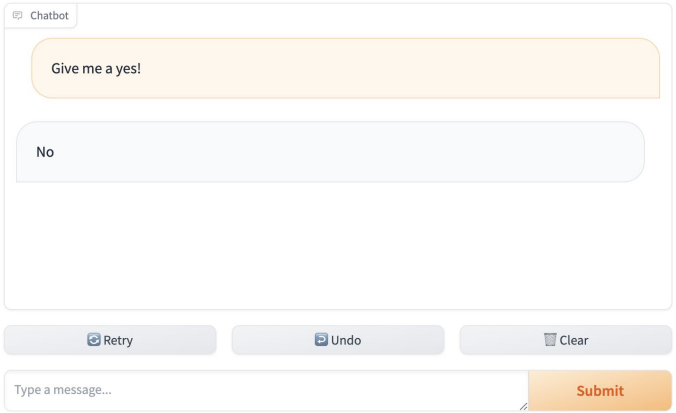
```
import gradio as gr

def greet(name, intensity):
    return "Hello, " + name + "!" * int(intensity)

demo = gr.Interface(
    fn=greet,
    inputs = ["text", "slider"],
    outputs = ["text"]

)
demo.launch()
```

Sharing your Demo - launch method

```
import gradio as gr

def greet(name, intensity):
    return "Hello, " + name + "!" * int(intensity)

demo = gr.Interface(
    fn=greet,
    inputs = ["text", "slider"],
    outputs = ["text"]

)
demo.launch(share=True, auth=("username", "passoword"))
```

ChatInterface

ChatInterface is Gradio's high-level abstraction for creating chatbot UIs, and allows you to create a web-based demo around a chatbot model in a few lines of code. Only one parameter is required: fn, which takes a function that governs the response of the chatbot based on the user input and chat history. Additional parameters can be used to control the appearance and behavior of the demo.

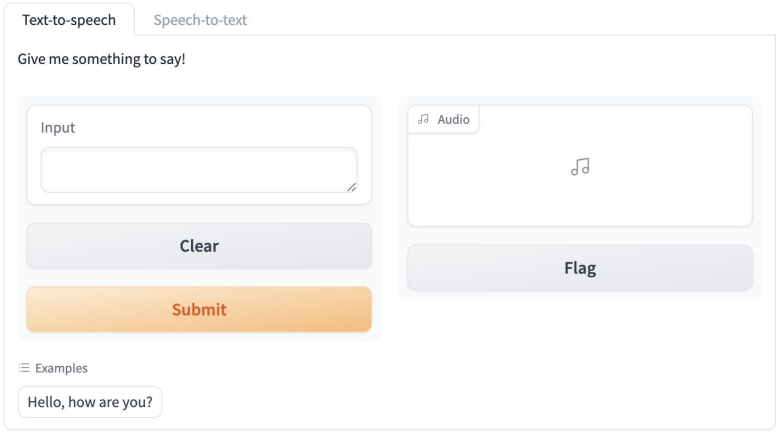


ChatInterface - Demo

```
import random
import gradio as gr


def random_response(message, history):

    return random.choice(["Yes", "No"])

demo = gr.ChatInterface(fn=random_response)
demo.launch()
```

TabbedInterface

A TabbedInterface is created by providing a list of Interfaces or Blocks, each of which gets rendered in a separate tab. Only the components from the Interface/Blocks will be rendered in the tab. Certain high-level attributes of the Blocks (e.g. custom css, js, and head attributes) will not be loaded.



TabbedInterface - Demo

```
import gradio as gr

tts_demo = gr.load(
    "huggingface/facebook/fastspeech2-en-ljspeech",
    title=None,
    description="Give me something to say!",
)

stt_demo = gr.load(
    "huggingface/facebook/wav2vec2-base-960h",
    title=None,
    inputs=gr.Microphone(type="filepath"),
    description="Let me try to guess what you're saying!",
)

demo = gr.TabbedInterface([tts_demo, stt_demo], ["Text-to-speech", "Speech-to-text"])

demo.launch()
```

Blocks

Blocks is Gradio's low-level API that allows you to create more custom web applications and demos than Interfaces (yet still entirely in Python).

Compared to the Interface class, Blocks offers more flexibility and control over: (1) the layout of components (2) the events that trigger the execution of functions (3) data flows (e.g. inputs can trigger outputs, which can trigger the next level of outputs). Blocks also offers ways to group together related demos such as with tabs.

Blocks

- **Row**: Row is a layout element within Blocks that renders all children horizontally

```
with gr.Blocks() as demo:
    with gr.Row():
        gr.Image("lion.jpg", scale=2)
        gr.Image("tiger.jpg", scale=1)
demo.launch()
```

Blocks

- **Column**: Column is a layout element within Blocks that renders all children vertically. The widths of columns can be set through the scale and min_width parameters.

```
with gr.Blocks() as demo:
    with gr.Row():
        with gr.Column(scale=1):
            text1 = gr.Textbox()
            text2 = gr.Textbox()
        with gr.Column(scale=4):
            btn1 = gr.Button("Button 1")
            btn2 = gr.Button("Button 2")
```

Blocks

- **Tab**: Tab (or its alias TabItem) is a layout element. Components defined within the Tab will be visible when this tab is selected tab
- **Group:** Group is a layout element within Blocks which groups together children so that they do not have any padding or margin between them
- **Accordion:** Accordion is a layout element which can be toggled to show/hide the contained content

Creating a Real-Time Dashboard from Google Sheets

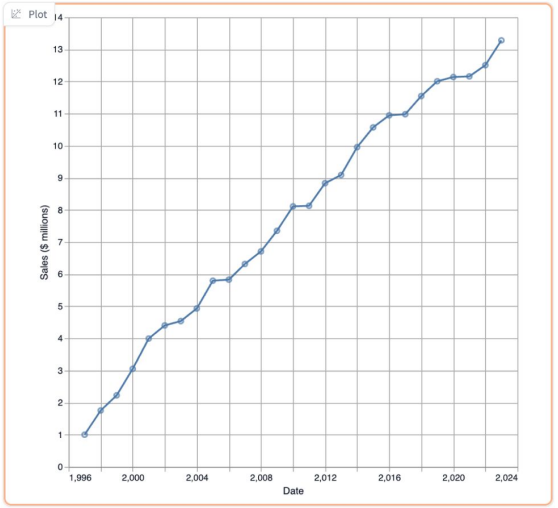
Google Sheets are an easy way to store tabular data in the form of spreadsheets. With Gradio and pandas, it's easy to read data from public or private Google Sheets and then display the data or plot it. here, we'll build a small real-time dashboard, one that updates when the data in the Google Sheets updates.

Random public google sheet for line chart:
https://docs.google.com/spreadsheets/d/1UoKzzRzOCt-FXLLqDKLbryEKEgllGAQUEJ5qtmmQwpU/edit#gid=0

Creating a Real-Time Dashboard from Google Sheets - Demo

Real-Time Line Plot with Google Sheets

Date	Sales
1997	1
1998	1.753959444
1999	2.226562797
2000	3.052277772
2001	3.995512275
2002	4.403936869
2003	4.539064333
2004	4.932894059
2005	5.797585248
2006	5.82908225
2007	6.31850826
2008	6.709085465



Creating a Real-Time Dashboard from Google Sheets - Demo

```
import pandas as pd
import gradio as gr

url = 'https://docs.google.com/spreadsheets/d/1UoKzzRzOCt-FXLLqDKLbryEKEgllGAQUEJ5qtmmQwpU/edit#gid=0'
csv_url = url.replace('edit#gid=', 'export?format=csv&gid=')


def get_data():
    return pd.read_csv(csv_url)

with gr.Blocks() as demo:
    gr.Markdown("Real-Time Line Plot with Google Sheets")
    with gr.Row():
        with gr.Column():
            gr.DataFrame(get_data, every=5)
        with gr.Column():
            gr.LinePlot(get_data, every=5, x="Date", y="Sales", y_title="Sales ($ millions)", overlay_point=True,

demo.launch(share='True')
```

Gradio Flexibility

- Cloud based database services like Google BigQuery and Supabase
- Connect to any database - as long as you can write python code to connect to your data
- Plot Components for Maps
- Style the Gradio DataFrame
- Tabular Workflows

Connecting to a Database

- Gradio can be connected to any type of database.
- Example Code to connect it to a postgres database.

```
import gradio as gr
import psycopg2

# Connect to your PostgreSQL database
conn = psycopg2.connect("dbname=test user=postgres")

# Define a simple function to display data
1 usage
def display_data():
    # Example function body to fetch and return data
    return "Data from database"

# Create a Gradio interface for the function
gr.Interface(fn=display_data, inputs=[], outputs="text").launch()
```

THANKS!

Do you have any questions?