```
from flask import Flask, render_template
import pandas as pd
import plotly.express as px
import plotly.io as pio
app = Flask(__name__)
# Load the dataset
def load_data():
    csv_file = 'customer_journey_data.csv'
    return pd.read_csv(csv_file)
@app.route('/')
def index():
    # Load data
    data = load_data()
    # Visualization 1: Income vs Average Spending
    fig1 = px.scatter(
        data,
        x="Income",
        y="Avg_Spending",
        title="Income vs Average Spending",
        labels={"Income": "Income", "Avg_Spending": "Average Spending"}
    )
    chart1 = pio.to_html(fig1, full_html=False)
    # Visualization 2: Purchase Frequency by Location
    location_freq = data.groupby("Location")["Purchase_Frequency"].sum().reset_index()
    fig2 = px.bar(
        location_freq,
        x="Location",
        y="Purchase_Frequency",
        title="Purchase Frequency by Location",
        labels={"Location": "Location", "Purchase_Frequency": "Total Purchase Frequency"}
    )
    chart2 = pio.to_html(fig2, full_html=False)
    # Pass the charts to the template
    return render_template('index.html', chart1=chart1, chart2=chart2)
if __name__ == '__main__':
    app.run(debug=True)
```

```
 * Serving Flask app '__main__'
 * Debug mode: on
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug: * Restarting with stat
```

Gemini API: Prompting Quickstart


Run in Google Colab

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Customer Journey Data</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <header>
        <h1>Customer Journey Data Visualizations</h1>
    </header>
    <main>
        <!-- Visualizations -->
        <section id="visualizations">
            <h2>Visualizations</h2>
            <div id="chart1">
                {{ chart1 | safe }}
            </div>
            <div id="chart2">
```

```
                {{ chart2 | safe }}
            </div>
        </section>
    </main>
    <footer>
        <p>&copy; 2023 Customer Journey Data</p>
    </footer>
</body>
</html>
```

This notebook contains examples of how to write and run your first prompts with the Gemini API.

```
!pip install -U -q "google-generativeai>=0.7.2" # Install the Python SDK
```

```
import google.generativeai as genai
```

Set up your API key

To run the following cell, your API key must be stored it in a Colab Secret named `GOOGLE_API_KEY`. If you don't already have an API key, or you're not sure how to create a Colab Secret, see the Authentication quickstart for an example.

```
from google.colab import userdata
GOOGLE_API_KEY=userdata.get('GOOGLE_API_KEY')
genai.configure(api_key=GOOGLE_API_KEY)
```

Run your first prompt

Use the `generate_content` method to generate responses to your prompts. You can pass text directly to generate_content, and use the `.text` property to get the text content of the response.

```
model = genai.GenerativeModel('gemini-2.0-flash')
response = model.generate_content("Give me python code to sort a list")
print(response.text)
```

```
```python
# Using the built-in sort() method
my_list = [5, 2, 8, 1, 9]
my_list.sort()
print(my_list)  # Output: [1, 2, 5, 8, 9]

# Using the sorted() function (returns a new sorted list)
my_list = [5, 2, 8, 1, 9]
sorted_list = sorted(my_list)
print(sorted_list)  # Output: [1, 2, 5, 8, 9]
print(my_list)  # Output: [5, 2, 8, 1, 9] (original list remains unchanged)

# Sorting in descending order
my_list = [5, 2, 8, 1, 9]
my_list.sort(reverse=True)
print(my_list)  # Output: [9, 8, 5, 2, 1]

# Sorting a list of tuples based on the second element
my_list = [(1, 5), (3, 2), (2, 8)]
my_list.sort(key=lambda x: x[1])
print(my_list)  # Output: [(3, 2), (1, 5), (2, 8)]
```

Explanation:

* **`my_list.sort()`:** This method sorts the list in place (modifies the original list).
* **`sorted(my_list)`:** This function returns a new sorted list without modifying the original list.
* **`reverse=True`:** This argument to the `sort()` method sorts the list in descending order.
* **`key=lambda x: x[1]`:** This argument to the `sort()` method specifies a function that extracts a value from each element to use for

Choose the method that best suits your needs based on whether you want to modify the original list or get a new sorted list.
```

Use images in your prompt

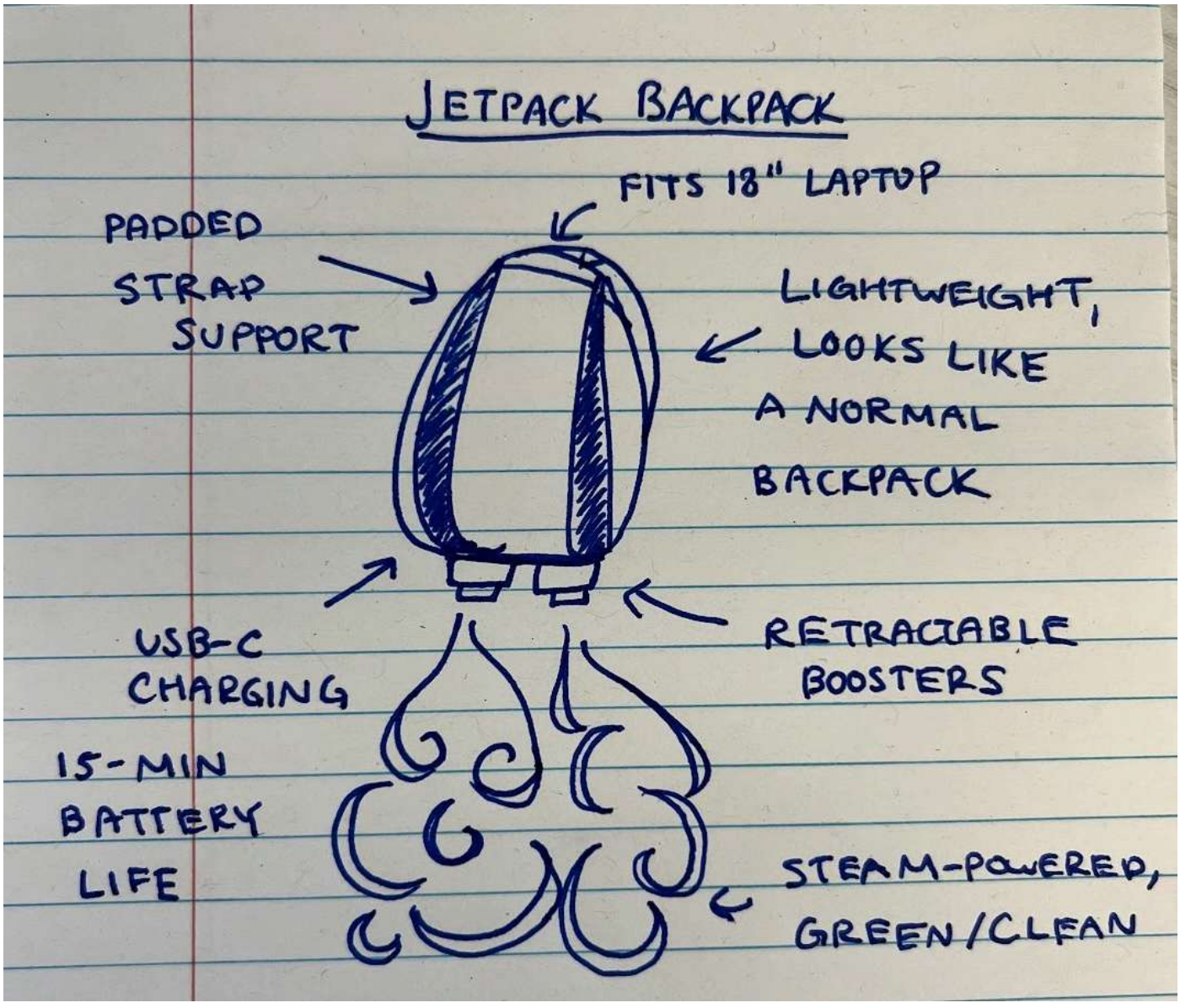
Here you will download an image from a URL and pass that image in our prompt.

First, you download the image and load it with PIL:

```
!curl -o image.jpg "https://storage.googleapis.com/generativeai-downloads/images/jetpack.jpg"
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  349k  100  349k    0     0  7591k      0 --:--:-- --:--:-- --:--:-- 7591k
```

```
import PIL.Image
img = PIL.Image.open('image.jpg')
img
```




```
prompt = """This image contains a sketch of a potential product along with some notes.
Given the product sketch, describe the product as thoroughly as possible based on what you
see in the image, making sure to note all of the product features. Return output in json format:
{description: description, features: [feature1, feature2, feature3, etc]}"""
```

Then you can include the image in our prompt by just passing a list of items to `generate_content`.

```
model = genai.GenerativeModel('gemini-2.0-flash')
response = model.generate_content([prompt, img])
print(response.text)
```

```
```json
{
 "description": "The Jetpack Backpack is a backpack that looks like a normal backpack but has retractable boosters that allow the wearer
 "features": [
  "Looks like a normal backpack",
  "Padded strap support",
  "Fits 18\" laptop",
  "Retractable boosters",
  "USB-C charging",
  "15-minute battery life",
  "Steam-powered",
  "Green/clean"
 ]
}
```
```

Have a chat

The Gemini API enables you to have freeform conversations across multiple turns.

The ChatSession class will store the conversation history for multi-turn interactions.

```
model = genai.GenerativeModel('gemini-2.0-flash')
chat = model.start_chat(history=[])
```

```
response = chat.send_message("In one sentence, explain how a computer works to a young child.")
print(response.text)
```

```
A computer is like a super smart toy that follows instructions from you, using numbers and lights to do amazing things!
```

You can see the chat history:

```
print(chat.history)
```

```
[parts {
  text: "In one sentence, explain how a computer works to a young child."
}
role: "user"
, parts {
  text: "A computer is like a super smart toy that follows instructions from you, using numbers and lights to do amazing things! \n"
}
role: "model"
]
```

You can keep sending messages to continue the conversation:

```
response = chat.send_message("Okay, how about a more detailed explanation to a high schooler?")
print(response.text)
```

Set the temperature

Every prompt you send to the model includes parameters that control how the model generates responses. Use a `genai.GenerationConfig` to set these, or omit it to use the defaults.

Temperature controls the degree of randomness in token selection. Use higher values for more creative responses, and lower values for more deterministic responses.

You can set the `generation_config` when creating the model.

```
model = genai.GenerativeModel(
    'gemini-2.0-flash',
    generation_config=genai.GenerationConfig(
        max_output_tokens=2000,
        temperature=0.9,
    ))
```

Or, set the `generation_config` on an individual call to `generate_content`. Any values set there override values on the model constructor.

Note: Although you can set the `candidate_count` in the generation_config, gemini-2.0-flash models will only return a single candidate at the this time.

```
response = model.generate_content(
    'Give me a numbered list of cat facts.',
    # Limit to 5 facts.
    generation_config = genai.GenerationConfig(stop_sequences=['\n6'])
)

print(response.text)
```

Learn more

There's lots more to learn!

- For more fun prompts, check out Market a Jetpack.
- Check out the safety quickstart next to learn about the Gemini API's configurable safety settings, and what to do if your prompt is blocked.
- For lots more details on using the Python SDK, check out this detailed quickstart.