Installing packages

```
!pip install langchain_google_genai
```

```
Requirement already satisfied: langchain_google_genai in /usr/local/lib/python3.12/dist-packages (4.2.0)
Requirement already satisfied: filetype<2.0.0,>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from langchain_google_genai) (1.2.0)
Requirement already satisfied: google-genai<2.0.0,>=1.56.0 in /usr/local/lib/python3.12/dist-packages (from langchain_google_genai) (1.6
Requirement already satisfied: langchain-core<2.0.0,>=1.2.5 in /usr/local/lib/python3.12/dist-packages (from langchain_google_genai) (1.
Requirement already satisfied: pydantic<3.0.0,>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from langchain_google_genai) (2.12.3)
Requirement already satisfied: anyio<5.0.0,>=4.8.0 in /usr/local/lib/python3.12/dist-packages (from google-genai<2.0.0,>=1.56.0->langcha
Requirement already satisfied: google-auth<3.0.0,>=2.47.0 in /usr/local/lib/python3.12/dist-packages (from google-auth[requests]<3.0.0,>
Requirement already satisfied: httpx<1.0.0,>=0.28.1 in /usr/local/lib/python3.12/dist-packages (from google-genai<2.0.0,>=1.56.0->langch
Requirement already satisfied: requests<3.0.0,>=2.28.1 in /usr/local/lib/python3.12/dist-packages (from google-genai<2.0.0,>=1.56.0->lan
Requirement already satisfied: tenacity<9.2.0,>=8.2.3 in /usr/local/lib/python3.12/dist-packages (from google-genai<2.0.0,>=1.56.0->lang
Requirement already satisfied: websockets<15.1.0,>=13.0.0 in /usr/local/lib/python3.12/dist-packages (from google-genai<2.0.0,>=1.56.0->
Requirement already satisfied: typing-extensions<5.0.0,>=4.11.0 in /usr/local/lib/python3.12/dist-packages (from google-genai<2.0.0,>=1.
Requirement already satisfied: distro<2,>=1.7.0 in /usr/local/lib/python3.12/dist-packages (from google-genai<2.0.0,>=1.56.0->langchain_
Requirement already satisfied: sniffio in /usr/local/lib/python3.12/dist-packages (from google-genai<2.0.0,>=1.56.0->langchain_google_ge
Requirement already satisfied: jsonpatch<2.0.0,>=1.33.0 in /usr/local/lib/python3.12/dist-packages (from langchain-core<2.0.0,>=1.2.5->l
Requirement already satisfied: langsmith<1.0.0,>=0.3.45 in /usr/local/lib/python3.12/dist-packages (from langchain-core<2.0.0,>=1.2.5->l
Requirement already satisfied: packaging<26.0.0,>=23.2.0 in /usr/local/lib/python3.12/dist-packages (from langchain-core<2.0.0,>=1.2.5->
Requirement already satisfied: pyyaml<7.0.0,>=5.3.0 in /usr/local/lib/python3.12/dist-packages (from langchain-core<2.0.0,>=1.2.5->langc
Requirement already satisfied: uuid-utils<1.0,>=0.12.0 in /usr/local/lib/python3.12/dist-packages (from langchain-core<2.0.0,>=1.2.5->la
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.12/dist-packages (from pydantic<3.0.0,>=2.0.0->langchain
Requirement already satisfied: pydantic-core==2.41.4 in /usr/local/lib/python3.12/dist-packages (from pydantic<3.0.0,>=2.0.0->langchain_
Requirement already satisfied: typing-inspection>=0.4.2 in /usr/local/lib/python3.12/dist-packages (from pydantic<3.0.0,>=2.0.0->langcha
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.12/dist-packages (from anyio<5.0.0,>=4.8.0->google-genai<2.0.0,>=1.56
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.12/dist-packages (from google-auth<3.0.0,>=2.47.0->google
Requirement already satisfied: cryptography>=38.0.3 in /usr/local/lib/python3.12/dist-packages (from google-auth<3.0.0,>=2.47.0->google-
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.12/dist-packages (from google-auth<3.0.0,>=2.47.0->google-auth[re
Requirement already satisfied: certifi in /usr/local/lib/python3.12/dist-packages (from httpx<1.0.0,>=0.28.1->google-genai<2.0.0,>=1.56.
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.12/dist-packages (from httpx<1.0.0,>=0.28.1->google-genai<2.0.0,>
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.12/dist-packages (from httpcore==1.*->httpx<1.0.0,>=0.28.1->google-ge
Requirement already satisfied: jsonpointer>=1.9 in /usr/local/lib/python3.12/dist-packages (from jsonpatch<2.0.0,>=1.33.0->langchain-cor
Requirement already satisfied: orjson>=3.9.14 in /usr/local/lib/python3.12/dist-packages (from langsmith<1.0.0,>=0.3.45->langchain-core<
Requirement already satisfied: requests-toolbelt>=1.0.0 in /usr/local/lib/python3.12/dist-packages (from langsmith<1.0.0,>=0.3.45->langc
Requirement already satisfied: zstandard>=0.23.0 in /usr/local/lib/python3.12/dist-packages (from langsmith<1.0.0,>=0.3.45->langchain-co
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests<3.0.0,>=2.28.1->google
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests<3.0.0,>=2.28.1->google-genai
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.12/dist-packages (from cryptography>=38.0.3->google-auth<3.0.0,>=2.4
Requirement already satisfied: pyasn1<0.7.0,>=0.6.1 in /usr/local/lib/python3.12/dist-packages (from pyasn1-modules>=0.2.1->google-auth<
Requirement already satisfied: pycparser in /usr/local/lib/python3.12/dist-packages (from cffi>=1.12->cryptography>=38.0.3->google-auth<
```

Setup your API key

To run the following cell, your API key must be stored it in a Colab Secret named VERTEX_API_KEY.

1. Look for the key icon on the left panel of your colab.
2. Under Name, create VERTEX_API_KEY.
3. Copy your key to Value.

```
from google.colab import userdata
GEMINI_VERTEX_API_KEY = userdata.get('VERTEX_API_KEY')
```

Downloading receipts.zip

The codes below download and unzip receipts.zip from Google Drive. receipts.zip contains all images from the Fusion folder on BlackBoard.

```
import gdown
file_id = "1oe2FZd3ZT07nrDqjCafNvxic108oF8JF"
download_url = f"https://drive.google.com/uc?id={file_id}"
gdown.download(download_url, "receipts.zip", quiet=False)
```

```
Downloading...
From: https://drive.google.com/uc?id=1oe2FZd3ZT07nrDqjCafNvxic108oF8JF
To: /content/receipts.zip
100%|██████████| 1.61M/1.61M [00:00<00:00, 148MB/s]
'receipts.zip'
```

```
!unzip receipts.zip
```

```
Archive: receipts.zip
replace receipt1.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename:
```

1. Helper functions

We need two functions

- image_to_base64 convert your jpg image into Base64 encoded string (basically a sequence of 64 characters to make your image easily transfered via API)
- get_image_data_url takes your jpg image, converting them into base64 string and construct a suitable input for GEMINI api call.

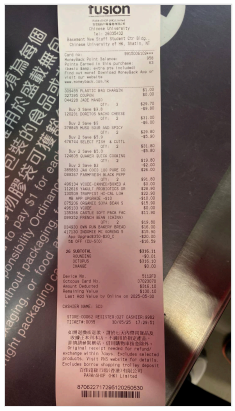
```
import base64
import mimetypes

# Helper function to read and encode image
def image_to_base64(img_path):
        with open(img_path, "rb") as img_file:
                return base64.b64encode(img_file.read()).decode('utf-8')

# Helper function to encode local file to Base64 Data URL
def get_image_data_url(image_path):
        # Guess the mime type (e.g., image/png, image/jpeg) based on file extension
        mime_type, _ = mimetypes.guess_type(image_path)
        if mime_type is None:
                mime_type = "image/png" # Default fallback

        encoded_string = image_to_base64(image_path)

        # Construct the Data URL
        return f"data:{mime_type};base64,{encoded_string}"
```

```
from langchain_google_genai import ChatGoogleGenerativeAI
llm = ChatGoogleGenerativeAI(
        model="gemini-2.5-flash",
        api_key=userdata.get('VERTEX_API_KEY'), # Ensure this key is set in Colab secrets
        temperature=0,
        vertexai=True
)
```

Display jpg images. Alternatively, open the folder icon on the left pannel to see the images.

```
from IPython.display import HTML, display
import glob, os

image_paths = glob.glob("*.jpg")
image_paths.sort()
html_content = '<div style="display: flex; flex-wrap: wrap; gap: 20px;">'

for path in image_paths:
        b64 = image_to_base64(path)
        filename = os.path.basename(path) # Clean up path to show just the name

        # Create a vertical column for each image + text
        html_content += f'''
        <div style="display: flex; flex-direction: column; align-items: center;">
                <img src="{b64}" style="height: 300px; border: 1px solid #ddd; margin-bottom: 5px;"/>
                <span style="font-family: monospace; font-size: 14px;">{filename}</span>
        </div>
        '''

html_content += '</div>'

display(HTML(html_content))
```


receipt1.jpg

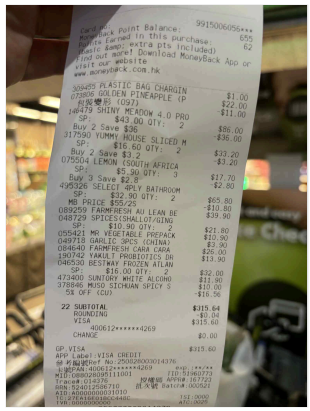

receipt2.jpg


receipt3.jpg


receipt4.jpg


receipt5.jpg


receipt6.jpg


receipt7.jpg

2. Image input to Gemini

Different from text, image needs to be converted into base64 encoded string and then formated into url before inputting to the language model. This is convenient for image-type input to be transfered through the API.

This project builds a multimodal LLM pipeline to extract totals from multiple receipt images and answer two queries: **Q1** (total paid after discounts) and **Q2** (total before discounts).

Each receipt is processed individually. The image is converted to a data URL, sent to the LLM with a strict output format (`FINAL=...`, `ORIGINAL=...`), and the numbers are extracted by regex and summed into `query1_answer` and `query2_answer` (single floats).

If `ORIGINAL` is missing, the code asks the LLM for `DISCOUNT` and estimates the pre-discount total as `FINAL + DISCOUNT`. Results are stored and printed in a summary table.

A separate classifier chain outputs only `Q1`,`Q2`,or `REJECT`, so irrelevant user queries are rejected.

```
from google.colab import auth
auth.authenticate_user()

import os, re
from langchain_core.prompts import ChatPromptTemplate

# ---------- Process receipts one by one to reduce errors ----------
query1_answer = 0.0
query2_answer = 0.0

results = []   # store per-receipt outputs for summary

for i, image_path in enumerate(image_paths):
    image_url = get_image_data_url(image_path)

    # Create a per-receipt prompt
    per_receipt_prompt = ChatPromptTemplate.from_messages([
        ("system",
         "You are a cashier analyzing ONE receipt. Extract these exact numbers:\n"
         "1. Find the 'Total', 'Amount Payable', or 'Grand Total' AFTER all discounts.\n"
         "2. Find the total BEFORE any discounts. If not shown, add up all item prices.\n"
         "3. If you see discounts (negative numbers or 'Discount', 'Save'), note them.\n"
         "Return ONLY in this format: FINAL=XX.XX, ORIGINAL=YY.YY"
        ),
        ("human", [
```

```
                        {"type": "text", "text": "What is the final total and original total on this receipt?"},
                        {"type": "image_url", "image_url": {"url": image_url}}
                ])
        ])

        per_receipt_chain = per_receipt_prompt | llm
        response = per_receipt_chain.invoke({})
        text = str(response.content)

        # Extract numbers using regex (tolerant)
        final_match = re.search(r'FINAL\s*[:=]\s*[$€£¥]?\s*([\d]+(?:\.[\d]+)?)', text, re.IGNORECASE)
        original_match = re.search(r'ORIGINAL\s*[:=]\s*[$€£¥]?\s*([\d]+(?:\.[\d]+)?)', text, re.IGNORECASE)

        final_total = None
        original_total = None
        original_est = None
        discount = None

        if final_match:
                final_total = float(final_match.group(1))
                query1_answer += final_total

        if original_match:
                original_total = float(original_match.group(1))
                query2_answer += original_total
        else:
                # If ORIGINAL is not found, ask the LLM to compute total discount and estimate ORIGINAL
                if final_total is not None:
                        discount_prompt = ChatPromptTemplate.from_messages([
                                ("system",
                                 "Find the total discount amount on this receipt.\n"
                                 "Return ONLY in this format: DISCOUNT=XX.XX"),
                                ("human", [
                                        {"type": "text", "text": "What is the total discount amount on this receipt?"},
                                        {"type": "image_url", "image_url": {"url": image_url}}
                                ])
                        ])
                        discount_response = (discount_prompt | llm).invoke({})
                        disc_text = str(discount_response.content)

                        disc_match = re.search(r'DISCOUNT\s*[:=]\s*[$€£¥]?\s*([\d]+(?:\.[\d]+)?)', disc_text, re.IGNORECASE)
                        if not disc_match:
                                disc_match = re.search(r'([\d]+(?:\.[\d]+)?)', disc_text)    # fallback

                        if disc_match:
                                discount = float(disc_match.group(1))
                                original_est = final_total + discount
                                query2_answer += original_est

        results.append({
                "No": i + 1,
                "Receipt": os.path.basename(image_path),
                "FINAL": final_total,
                "ORIGINAL": original_total,
                "ORIG_EST": original_est,
                "RAW": text
        })

# ========================
# Summary output (not split)
# ========================
print("\n" + "="*80)
print("Summary (all receipts):")
print(f"{'No.':<4}{'Receipt':<25}{'FINAL':>12}{'ORIGINAL':>14}{'ORIG_EST':>12}")
for r in results:
        f = "" if r["FINAL"] is None else f"{r['FINAL']:.2f}"
        o = "" if r["ORIGINAL"] is None else f"{r['ORIGINAL']:.2f}"
        e = "" if r["ORIG_EST"] is None else f"{r['ORIG_EST']:.2f}"
        print(f"{r['No']:<4}{r['Receipt']:<25}{f:>12}{o:>14}{e:>12}")

print("-"*80)
print(f"Query 1 total spent (after discount): {query1_answer:.2f}")
print(f"Query 2 total spent (before discount): {query2_answer:.2f}")
print("="*80)

# ========================
# Add-on: Reject irrelevant queries
# ========================
def answer_user_query(user_query: str):
        cls_prompt = ChatPromptTemplate.from_messages([
                ("system",
                 "Classify the user query. Output ONLY one token: Q1, Q2, or REJECT.\n"
                 "Q1 = total spent in total AFTER discounts.\n"
```

```
                "Q2 = total that would have been paid WITHOUT discounts (before discounts).\n"
                "Anything else = REJECT."),
            ("human", "{q}")
        ])

        tag = (cls_prompt | llm).invoke({"q": user_query}).content.strip().upper()

        if "Q1" in tag:
            return query1_answer
        elif "Q2" in tag:
            return query2_answer
        else:
            return "REJECTED"   # if needed: "NONE"
```

```
================================================================================
Summary (all receipts):
No. Receipt                        FINAL      ORIGINAL    ORIG_EST
1   receipt1.jpg                  394.70        480.40
2   receipt2.jpg                  316.10        392.20
3   receipt3.jpg                  140.80        160.10
4   receipt4.jpg                  514.00        590.80
5   receipt5.jpg                  102.30        107.70
6   receipt6.jpg                  190.80        221.20
7   receipt7.jpg                  315.60        396.00
--------------------------------------------------------------------------------
Query 1 total spent (after discount): 1974.30
Query 2 total spent (before discount): 2348.40
================================================================================
```

3. Evaluation Code

- Make sure your LLM return a single float as the answer, stored in `query1_answer` and `query2_answer`
- Run the following code blocks: (1) If the blocks does not return any error, then your chain design is correct. Otherwise, please check your chain design.
- Do not modify `query_1_costs` and `query_2_costs`

```
def test_query(answer, ground_truth_costs):
    # Convert string to float if necessary
    if isinstance(answer, str):
        answer = float(answer)

    # Calculate the ground truth sum once for clarity
    expected_total = sum(ground_truth_costs)

    # Check if the answer is within +/- $2 of the expected total
    assert abs(answer - expected_total) <= 2
```

Run the following code block to evaluate query 1:

> How much money did I spend in total for these bills?

```
query_1_costs = [394.7, 316.1, 140.8, 514.0, 102.3, 190.8, 315.6] # do not modify this
test_query(query1_answer, query_1_costs)
```

Run the following code block to evaluate query 2:

> How much would I have had to pay without the discount?

```
query_2_costs = [480.20, 392.20, 160.10, 590.80, 107.70, 221.20, 396.00] # do not modify this
test_query(query2_answer, query_2_costs)
```

```
sum([480.20, 392.20, 160.10, 590.80, 107.70, 221.20, 396.00])
```

```
2348.2
```