HW1

This code is an intelligent extraction and automatic analysis tool for supermarket receipt financial data based on the Gemini large model. It runs primarily in the Google Colab environment and is specifically designed for the statistics of amounts on multiple receipts, discount restoration, and result verification. The core tasks can be summarized as the following six points:

1. Receipt image loading and visualization: Automatically search for receipt images in the current directory in jpg/jpeg/png formats, remove duplicates, sort them, and display them visually (with file names and fixed-size layout). Provide a clear prompt when no images are found.

```
def display_images():
    image_paths = []
    for ext in ("*.jpg", "*.jpeg", "*.png", "*.JPG"):
        image_paths.extend(glob.glob(ext))
    image_paths = sorted(list(set(image_paths)))

    if not image_paths:
        print("Warning: No images found to display. Please upload receipt images first!")
        return

    html_content = '<div style="display: flex; flex-wrap: wrap; gap: 20px; padding: 10px;">'
    for path in image_paths:
        b64 = image_to_base64(path)
        filename = os.path.basename(path)
        html_content += f'''
        <div style="display: flex; flex-direction: column; align-items: center;">
            <img src="{b64}" style="height: 250px; border: 1px solid #ddd; border-radius: 4px; margin-bottom: 5px;"/>
            <span style="font-family: monospace; font-size: 12px; color: #333;">[filename]</span>
        </div>
        '''
    html_content += '</div>'
    display(HTML(html_content))

print("--- Loaded Receipt Images ---")
display_images()
```

2. Image format conversion: Encode the receipt image into Base64 format, generate a Data URL that conforms to the specification, and adapt to the image input requirements of large models.

```
def image_to_base64(img_path):
    with open(img_path, "rb") as img_file:
        return base64.b64encode(img_file.read()).decode('utf-8')

def get_image_data_url(image_path):
    mime_type, _ = mimetypes.guess_type(image_path)
    if mime_type is None:
        mime_type = "image/png"
    encoded_string = image_to_base64(image_path)
    return f"data:{mime_type};base64,{encoded_string}"
```

3. Model configuration and invocation: Initialize the Google Gemini-3-Flash model (via Vertex AI interface), combine system instructions (professional accounting

rules) and user queries, send a composite request of "text + multiple images" to the model, and obtain the model's response;

```
llm = ChatGoogleGenerativeAI(
        model="gemini-3-flash-preview",
        api_key=userdata.get('VERTEX_API_KEY'),
        temperature=0,
        vertexai=True
)
```

4. Multi-scenario financial data extraction and calculation: According to preset strict rules, extract the core amounts from receipts. It supports two types of core calculations (total amount actually paid for all receipts, original total amount before discount), and irrelevant queries will directly return the fixed identifier "IRRELEVANT".

```
system_instruction = """
You are an expert accountant and receipt scanner. Your job is to extract financial data from images of supermarket bills.

You will receive several images of receipts and a user query.
Follow these rules strictly:

RULE 1 (Total Spend):
If the user asks "How much money did I spend in total?" (or similar),
locate the final 'Total' or 'Balance Due' on EACH bill and sum them up

RULE 2 (Calculate Original Total Price / Gross Amount):
If the user asks "How much was the total before discounts?" or similar, calculate the sum of the original prices by reversing the discounts.
Calculate the ORIGINAL TOTAL (before discounts) using this strict rule:
ORIGINAL TOTAL = Total Paid (实付金额: 收据中「小計」后的数字) + SUM of ALL discount amounts (所有带减号的消费类折扣金额)

CRITICAL RULES (必须遵守):
1. ONLY include: 「小計」后的实付金额 + 消费项旁的带减号折扣（如-12.40、-30.00）
2. NEVER include: 「餘額」/「余额」（账户余额），无论余额是正还是负（比如-500.00的余额完全忽略）
3. List ALL discounts first (避免漏算)，再求和（只加折扣的绝对值）
4. Must output a number > Total Paid (原价>实付)
5. ONLY output the final number (no text, e.g., "486.10")

RULE 3 (Irrelevant):
If the user asks anything not related to the costs (e.g., "Is this healthy?", "What is the store name?"),
reply exactly: "IRRELEVANT".

OUTPUT FORMAT:
- Return ONLY a single floating-point number representing the final calculated sum.
- Do NOT include currency symbols (like $ or HKD).
- Do NOT write sentences.
- Example correct output: 154.50
"""

messages = [
        SystemMessage(content=system_instruction),
        HumanMessage(content=message_content)
]
response = llm.invoke(messages)
```

5. Model response cleaning and parsing: Uniformly clean the multiple formats of responses returned by the model (dictionaries / lists / strings), extract numbers using regular expressions and convert them to floating-point types, ensuring that the output is purely numerical results;

```
def get_model_response(user_query, image_pattern="*.jpg"):

    image_paths = []
    for ext in ("*.jpg", "*.jpeg", "*.png", "*.JPG"):
        image_paths.extend(glob.glob(ext))
    image_paths = sorted(list(set(image_paths)))

    if not image_paths:
        return "No images found."


    message_content = [{"type": "text", "text": user_query}]
    for img_path in image_paths:
        image_url = get_image_data_url(img_path)
        message_content.append({
            "type": "image_url",
            "image_url": {"url": image_url}
        })
```

```
raw_content = response.content
content = ""

if isinstance(raw_content, dict):

    for key in raw_content.keys():
        if key.strip().upper() == "TEXT":
            content = str(raw_content[key])
            break

    if not content:
        content = str(raw_content)

elif isinstance(raw_content, list):
    content = ''.join(map(str, raw_content))

else:
    content = str(raw_content)


content_clean = content.strip().upper()

if "IRRELEVANT" in content_clean:
    return "IRRELEVANT"

try:

    match = re.search(r"[-+]?\d+\.?\d*", content.strip())
    if match:
        return float(match.group())
    return content.strip()
except Exception as e:
    return content.strip()
```

6. Batch Query and Automated Result Evaluation: Pre-set 12 types of typical user queries (including core amount queries, irrelevant queries, multi-intent queries,

etc.) and execute them in batches. At the same time, provide a test function to compare the model results with the actual labeled data, determine whether the error is within the acceptable range (≤2), and output clear pass/fail results.

```
q1_text = "How much money did I spend in total for these bills?"
q2_text = "How much would I have had to pay without the discount?"
q3_text = "who are you?"
q4_text = "What is the total amount I paid across all these receipts?"
q5_text = "Calculate the total original price before any discounts were applied."
q6_text = "Calculate the total original price before discounts, do not include any account balance numbers."
q7_text = "How much did I spend in total, and what color is the shop logo?"
q8_text = "计算所有小票的小计总和，不要包含任何余额"
q9_text = "What is the name of the supermarket on these receipts?"
q10_text = "What is the transaction date on each receipt?"
q11_text = "List all the items I bought from these receipts."
q12_text = "Did I pay with credit card or cash for these bills?"

queries = [
        ("Query 1", q1_text),
        ("Query 2", q2_text),
        ("Query 3", q3_text),
        ("Query 4", q4_text),
        ("Query 5", q5_text),
        ("Query 6", q6_text),
        ("Query 7", q7_text),
        ("Query 8", q8_text),
        ("Query 9", q9_text),
        ("Query 10", q10_text),
        ("Query 11", q11_text),
        ("Query 12", q12_text),
]

for name, text in queries:
        print(f"\nProcessing {name}: {text}...")
        answer = get_model_response(text)
        print(f"Model {name}: {answer}")

def test_query(answer, ground_truth_costs):

        if answer == "IRRELEVANT":
                print("Skip Evaluation: Answer is IRRELEVANT (irrelevant user query)")
                return

        if answer is None or answer == "No images found.":
                print(f"Error: Invalid answer - [{answer}]. Please check if receipt images are uploaded correctly!")
                return

        try:
                val = float(answer)
        except (ValueError, TypeError):
                print(f"Error: Could not convert model answer to float -> Answer: [{answer}]")
                return

        expected_total = sum(ground_truth_costs)
        print(f"Expected Result: {expected_total:.2f}, Model Result: {val:.2f}")

        if abs(val - expected_total) <= 2:
                print("✅ Test Passed! (Error within acceptable range)")
        else:
                print(f"❌ Test Failed! (Absolute Difference: {abs(val - expected_total):.2f})")
```

Additional note: The code has also been adapted for Chinese receipts (such as recognizing Chinese labels like "Subtotal", "Balance", "Discount with minus sign", etc.) and has imposed strict formatting constraints on the model output (only returning floating-point numbers, without text or currency symbols), ensuring the accuracy and usability of financial data.