Homework 1 Report

---

1. Data Preprocessing and Image Handling

I worked in Google Colab and first installed and
configured langchaingooglegenai and the Gemini model using VERTEX API key.

Steps:

- Downloaded and unzipped receipts.zip from Google Drive to obtain 7 JPG receipts.
- Implemented helper functions mage_to_base64 and get_image_data_url
- Displayed all receipt images in a grid within the notebook for manual inspection of item lines, discounts, subtotals, and payment amounts.

This ensured that each image could be reliably passed to Gemini as an image URL input.

---

2. Structured Extraction with Gemini

2.1 System Prompt Design

I created a **strict system prompt** to force Gemini to output a single JSON object per receipt:

- The required format:

```
{
    "positives": [p1, p2, ...],
    "negatives": [n1, n2, ...],
    "subtotal": s,
    "paid": a
}
```

- positives: all product / charge line prices (no discounts), as floats with two decimals.
- negatives: all discounts / coupons / savings as **negative** floats with two decimals.

- subtotal: printed subtotal on the receipt without roundings.
- paid: final amount paid.

    - Hard constraints in the prompt:

        - Return **only** raw JSON, no explanation, no markdown.
        - Numbers must be JSON numbers (no quotes).
        - If uncertain, still return valid JSON adhering to the schema.

2.2 JSON Extraction and Error Handling

Because LLMs sometimes wrap JSON in text, I implemented:

- extract_json_from_text(text):
    - Strips code fences or extra text.
    - Tries json.loads on the full text first.
    - If that fails, searches for the first { … } block and attempts to parse it.
    - Raises an error if no valid JSON can be extracted.
- extract_receipt_json(image_path):
    - Builds a ChatPromptTemplate with:
        - SystemMessage: the system prompt above.
        - HumanMessage: containing the image URL and an instruction to “Return ONLY the JSON object as specified.”
    - Calls llm.invoke(messages) and then extract_json_from_text() on response.content.
    - Returns a Python dict with positives, negatives, subtotal, paid.

I then iterated over all receipt*.jpg files to build receipt_structs, a list of dicts, one per receipt. For debugging, I printed the extracted fields per image.

---

3. Aggregation Logic for Query 1 and Query 2

3.1 Global Totals Function

I implemented compute_totals_from_structs(receipt_structs) to aggregate information across all receipts:

```
for r in receipt_structs:
        pos = r.get("positives", [])
        neg = r.get("negatives", [])
        subtotal = float(r.get("subtotal", 0.0) or 0.0)
        paid = float(r.get("paid", 0.0) or 0.0)

        total_without += sum(pos)
        total_spent += paid
```

```
return round(total_spent, 2), round(total_without, 2)
```

- Inputs (for r in receipt_structs):

```
{'image': 'receipt3.jpg',
'positives': [19.6, 35.8, 35.8, 15.0, 15.0, 17.9, 16.0, 5.0],
'negatives': [-4.0, -7.8, -7.42, -0.08],
'subtotal': 140.88,
'paid': 140.8},
```

- Outputs (result for each receipt and summary):

```
receipt7.jpg:
  positives sum = 396.00
  negatives sum = -80.40
  subtotal      = 315.64
  paid          = 315.60
----------------------------------------

=== GLOBAL TOTALS ===
Total spent (Query 1 style)          : 1974.30
Total without discount (Query 2 style): 2348.20
```

- total_spent: sum of **paid** values across all receipts (Query 1).

- total_without: sum of all **positives** across receipts, ignoring discounts (Query 2).

3.2 The usage of these computed Results

After the model computed the results, the results stored in **total_spent** and **total_without**.

```
# Run the calculation
total_spent, total_without_discount = compute_totals_from_structs(receipt_structs)
```

The suitable answer will be output according to the query (more details in 4.2).

---

4. Query Classification and Answering

4.1 **Query Type Classification**

I wrote classify_query(user_query) to categorize incoming questions into three categories: TOTAL_SPENT, WITHOUT_DISCOUNT and OUT_OF_DOMAIN

It lowercases the query and checks for keyword sets:

- **Without discount keywords** (checked first):

```
without_discount_keywords = [
        "without discount", "without the discount", "before discount",
        "no discount", "original price", "full price", "before promotion",
        "without promotion", "without savings", "before savings"
]
```

- **Total spent keywords**:

```
total_spent_keywords = [
        "total", "spend", "spent", "pay", "paid", "cost", "how much",
        "amount", "sum", "bill", "expense"
]
```

If no keywords match, it returns OUT_OF_DOMAIN.

4.2 Main Answer Function

answer_query(user_query, image_paths) behaves as follows:

```
# Classify the query
query_type = classify_query(user_query)

# Output the response according to different query type
if query_type == "OUT_OF_DOMAIN":
        return "Sorry, I can only answer questions about total spent or total before discount on these receipts."
elif query_type == "WITHOUT_DISCOUNT":
        return total_without_discount
elif query_type == "TOTAL_SPENT":
        return total_spent
```

Then the return value can be put into test_query() to see whether it is accurate.

---

5. Evaluation and Testing

The provided evaluation code defines testquery(generated_answer, actual_answer), which computes expected_total = sum(groundtruth_costs) and

asserts abs(answer - expected_total) <= 2.

Run the code to evaluate query 1:

```
query_1_costs = [394.7, 316.1, 140.8, 514.0, 102.3, 190.8, 315.6] # do not modify this
query1_answer = answer_query("How much money did I spend in total for these bills?", image_paths)
test_query(query1_answer, query_1_costs)
```

```
sum([394.7, 316.1, 140.8, 514.0, 102.3, 190.8, 315.6])
```

```
1974.3
```

```
query1_answer
```

```
1974.3
```

and for Query 2:

```
query_2_costs = [480.20, 392.20, 160.10, 590.80, 107.70, 221.20, 396.00] # do not modify this
query2_answer = answer_query("How much would I have had to pay without the discount?", image_paths)
test_query(query2_answer, query_2_costs)
```

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

```
2348.2
```

```
query2_answer
```

```
2348.2
```

Also handle the out of the domain query:

```
query3_answer = answer_query("What is the date?", image_paths)
query3_answer
```

```
'Sorry, I can only answer questions about total spent or total before discount on these receipts.'
```

These test results can confirm that the chain design returns the correct numeric answers within the tolerance and handles the out of domain query.