

## **ASSIGNMENT – 4.5**

**NAME:AKSHITHA**

**HT NO:2303A51360**

**BATCH NO:29**

### **ADVANCED PROMPT ENGINEERING: ZERO-SHOT, ONE-SHOT & FEW-SHOT**

#### **TASK-1:**

##### **ZERO-SHOT**

A. Preparing Sample data:

```
test_emails = [  
    "My payment failed but money was deducted.",  
    "The app is not opening on my phone.",  
    "Great customer service, very satisfied.",  
    "What is your customer care number?",  
    "Invoice amount seems incorrect."  
]
```

Expected Labels (for evaluation):

```
true_labels = [  
    "Billing",  
    "Technical Support", "Feedback",  
    "Others",  
    "Billing"  
]
```

#### **PROMPT:**

Classify the following email into one of the categories:

Billing, Technical Support, Feedback, Others.

Email: "<email\_text>"

Return only the category name.

## **CODE:**

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The top bar includes 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', 'Terminal', 'Help' menus, and a search bar labeled 'ASSISTANT CODING'. The left sidebar has sections for 'EXPLORER', 'ASSISTANT CODINGR', and a 'CHAT' section with a message bubble icon. The main area displays a Python script named '4.5.PY' with the following code:

```
# Sample data
test_emails = [
    "My payment failed but money was deducted.",
    "The app is not opening on my phone.",
    "great customer service, very satisfied.",
    "What is your customer care number?",
    "Invoice amount seems incorrect."
]

# Expected Labels (for evaluation)
true_labels = [
    "Billing",
    "Technical Support",
    "Feedback",
    "Others",
    "Billing"
]

# Simple rule-based classifier
def classify_email(email):
    email = email.lower()

    if any(word in email for word in ["payment", "invoice", "amount", "deducted", "bill"]):
        return "Billing"

    if "app" in email or "phone" in email:
        return "Technical Support"

    if "customer" in email or "service" in email:
        return "Feedback"

    if "care" in email:
        return "Others"

    return "Billing"
```

Below the code editor, tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', 'PORTS', and 'GITLENS' are visible. The 'TERMINAL' tab is active, showing PowerShell output:

```
PowerShell detected that you might be using a screen reader and has disabled PSReadline for compatibility purposes. If you want to re-enable it, run 'Import-Module PSReadline'.
```

```
PS C:\Users\nampa\OneDrive\Desktop\ASSISTANT CODINGR> & C:/Users/nampa/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/nampa/Desktop/ASSISTANT CODINGR/4.5.py"
Email: My payment failed but money was deducted.
Predicted Category: Billing

Email: The app is not opening on my phone.
Predicted Category: Technical Support

Email: Great customer service, very satisfied.
Predicted Category: Feedback
```

The bottom status bar shows 'Ln 16, Col 2' and 'Python 3.13 (64-bit)'.

## OBSERVATION:

Classifies emails using only instructions, without examples.

Works if keywords are clear, may misclassify ambiguous emails.

Quick and simple, but less accurate for complex cases.

## ONE SHOT:

## PROMPT: Example:

Email: "I was charged twice for my last payment."

## Category: Billing

Now classify the following email:

Email: “<email text>”

## **CODE:**

```

37 # Define keyword lists for each category
38 billing_keywords = ["payment", "invoice", "billing", "charged", "deducted", "amount"]
39 tech_support_keywords = ["app", "not opening", "not working", "error", "issue", "bug", "crash"]
40 feedback_keywords = ["great", "satisfied", "thank you", "awesome", "love", "impressed"]
41
42 # Classification function
43 def classify_email(email_text):
44     email_lower = email_text.lower()
45     if any(keyword in email_lower for keyword in billing_keywords):
46         return "Billing"
47     elif any(keyword in email_lower for keyword in tech_support_keywords):
48         return "Technical Support"
49     elif any(keyword in email_lower for keyword in feedback_keywords):
50         return "Feedback"
51     else:
52         return "Others"
53
54 # Sample email data
55 test_emails = [
56     "My payment failed but money was deducted.",
57     "The app is not opening on my phone.",
58     "Great customer service, very satisfied."
59 ]

```

PS C:\Users\nampa\OneDrive\Desktop\ASSISTANT CODING> & c:\Users\nampa\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/nampa/OneDrive/Desktop/ASSISTANT CODING/4.5.py"

- Email: My payment failed but money was deducted.  
Predicted Category: Billing
- Email: The app is not opening on my phone.  
Predicted Category: Technical Support
- Email: Great customer service, very satisfied.  
Predicted Category: Feedback

Email: What is your customer care number?  
Predicted Category: Others

## OBSERVATION:

Provides one example to guide the AI's reasoning.

Improves accuracy over zero-shot and handles slightly ambiguous emails better.

Still limited; accuracy depends on how representative the example is.

One example helps the AI understand the expected format and category mapping.

Classification accuracy improves compared to zero-shot, especially for similar issues.

Performance depends heavily on how relevant the single example is to the new email.

Few shot:

PROMPT: Email: "I was charged twice for my last payment." → Billing

Email: "The app crashes on login." → Technical Support

Email: "I love the new update." → Feedback

Email: "What are your office hours?" → Others

```

File Edit Selection View Go Run Terminal Help ← → Q ASSISTANT CODING
EXPLORER ... 3.1.PY 4.2.PY 4.5.PY
ASSISTANT CODING
3.1.PY
4.2.PY
4.5.PY
4.5.PY > ...
76     def classify_email(email_text):
77         if any(keyword in email.lower() for keyword in tech_support_keywords):
78             return "Technical Support"
79         elif any(keyword in email.lower() for keyword in feedback_keywords):
80             return "Feedback"
81         else:
82             return "Others"
83
84
85
86
87     # Sample email data
88     test_emails = [
89         "My payment failed but money was deducted.",
90         "The app is not opening on my phone.",
91         "Great customer service, very satisfied.",
92         "What is your customer care number?",
93         "Invoice amount seems incorrect."
94     ]
95
96     # Run classification
97     for email in test_emails:
98         category = classify_email(email)
99         print(f"Email: '{email}'\nCategory: {category}\n{'-'*40}")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

Email: "The app is not opening on my phone."  
Category: Technical Support  
-----  
Email: "Great customer service, very satisfied."  
Category: Feedback  
-----  
Email: "What is your customer care number?"  
Category: Others  
-----  
Email: "Invoice amount seems incorrect."  
Category: Billing

PS C:\Users\nampa\OneDrive\Desktop\ASSISTANT CODING> | In 99, Col 65 Spaces: 4 UTF-8 CRLF { } Python 🐍 Go Live 🚀 Prettier ⚡

## OBSERVATION:

Provides multiple examples to show patterns to the AI. Highest accuracy;

AI can generalize better for unseen emails.

Slightly longer prompts but most reliable for real-world use

## TASK-2:

# Sample travel queries (short & simple)

travel\_queries = [

"Book a flight from Delhi to Mumbai.",

"Cancel my hotel reservation in Paris.",

"What is the baggage allowance?",

"I need a hotel in London for 2 nights.",

"Cancel my flight ticket to New York."

]

# True labels for evaluation

true\_labels = [

"Flight Booking",

"Cancellation",

"General Travel Info",

"Hotel Booking",

"Cancellation"

]

## ZERO-SHOT:

PROMPT:Classify the the following travel query into one of the categories:

Flight Booking, Hotel Booking, Cancellation, General Travel Info.

Query: "<travel\_query>"

## CODE:

The screenshot shows a VS Code interface with the following details:

- File Explorer:** Shows files 3.1.PY, 4.2.PY, and 4.5.PY. 4.5.PY is the active file.
- Code Editor:** Displays the following Python code:

```
def classify_query(query):
    if "cancel" in query:
        return "Cancellation"
    return "Hotel Booking"

    elif any(word in query for word in ["baggage", "allowance", "luggage", "info"]):
        return "General Travel Info"

    elif "cancel" in query:
        return "Cancellation"

    else:
        return "General Travel Info"

# Testing the classifier
for query in travel_queries:
    print("Query:", query)
    print("Predicted Category:", classify_query(query))
    print()
```
- Terminal:** Shows test results for four queries:

```
Query: Cancel my hotel reservation in Paris.
Predicted Category: Cancellation

Query: What is the baggage allowance?
Predicted Category: General Travel Info

Query: I need a hotel in London for 2 nights.
Predicted Category: Hotel Booking

Query: Cancel my flight ticket to New York.
Predicted Category: Cancellation
```
- Right Panel:** A "Build with Agent" panel is open, with a message: "AI responses may be inaccurate. Generate Agent Instructions to onboard AI onto your codebase." It also shows a "Describe what to build next" input field and "Agent" settings.

## OBSERVATION:

Classifies queries using only instructions, without examples.

Works for obvious keywords like “flight” or “cancel”, may misclassify tricky queries.

Fast and simple, but accuracy is lower for ambiguous cases.

one-shot:

## PROMPT:Example:

Query: "Cancel my flight ticket."

Category: Cancellation

Now classify the following query:

Query: "<travel\_query>"

## CODE:

The screenshot shows a Visual Studio Code (VS Code) interface with an AI extension. The left sidebar has an 'EXPLORER' view showing files 3.1.PY, 4.2.PY, and 4.5.PY. The main editor area contains Python code for a travel query classifier. A floating 'CHAT' window on the right says 'Build with Agent' with the message 'AI responses may be inaccurate. Generate Agent Instructions to onboard AI onto your codebase.' Below the editor are tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, and GITLENS. The TERMINAL tab is selected, showing a powershell prompt. At the bottom, status bar details include 'Line 116, Col 2', 'Spaces: 4', 'UTF-8', 'CRLF', 'Python 3.13 (64-bit)', and 'Prettier'.

```
File Edit Selection View Go Run Terminal Help < > ASSISTANT CODINGR CHAT ... + v ... x
EXPLORER ... 3.1.PY 4.2.PY 4.5.PY
ASSISTANT CODINGR
travel_queries = [
    "Book a flight from Delhi to Mumbai.",
    "Cancel my hotel reservation in Paris.",
    "What is the baggage allowance?",
    "I need a hotel in London for 2 nights.",
    "Cancel my flight ticket to New York."
]
# True labels for evaluation
true_labels = [
    "Flight Booking",
    "Cancellation",
    "General Travel Info",
    "Hotel Booking",
    "Cancellation"
]
# Zero-shot classifier (rule-based simulation)
def classify_query(query):
    query = query.lower()
    if any(word in query for word in ["flight", "ticket", "air"]):
        if "cancel" in query:
            return "Cancellation"
Example:
Query: "Cancel my flight ticket."
Category: Cancellation
Now classify the following query:
Query: "Cancel my flight ticket to New York."
Return only the category name
-----
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS
powershell + v ... x
PS C:\Users\nampa\OneDrive\Desktop\ASSISTANT CODING>
Line 116, Col 2 Spaces: 4 UTF-8 CRLF Python 3.13 (64-bit) Prettier
```

## OBSERVATION:

Provides one example to guide AI's reasoning.

Improves accuracy and handles slightly ambiguous queries better.

Accuracy depends on how representative the example is.

## FEW SHOT:

## PROMPT:

Examples:

Query: "Book a flight to Mumbai."

## Category: Flight Booking

Query: "Cancel my hotel reservation."

## Category: Cancellation

Query: "I need a hotel in London."

## Category: Hotel Booking

Query: "What is the baggage allowance?"

## Category: General Travel Info

Now classify the following query:

Query: "<travel\_query>"

## CODE:

The screenshot shows a VS Code interface with the following details:

- File Explorer:** Shows files 3.1.PY, 4.2.PY, 4.5.PY, and 4.5.PY (selected).
- Code Editor:** Displays Python code for travel classification. The code defines a function `few_shot_travel_classification` that takes a list of queries. It prints each query, checks for specific keywords like "cancel", "flight", or "hotel", and returns a corresponding output ("Cancellation", "Flight Booking", "Hotel Booking") or "General Travel Info". The code then prints the AI output and a separator line. Finally, it defines a list `travel_queries` containing five travel-related prompts.
- Terminal:** Shows AI responses for the first two queries:

```
==== FEW-SHOT PROMPTING ====  
Query: "Book a flight from Delhi to Mumbai."  
AI Output: Flight Booking  
-----  
Query: "Cancel my hotel reservation in Paris."  
AI Output: Cancellation  
-----  
Query: "What is the baggage allowance?"  
AI Output: General Travel Info  
-----  
Query: "I need a hotel in London for 2 nights."  
AI Output: Hotel Booking
```
- Right Panel:** Features a "Build with Agent" section with a message: "AI responses may be inaccurate. Generate Agent Instructions to onboard AI onto your database." A "CHAT" tab is also visible.
- Bottom Status Bar:** Shows file path (main.py), line count (Ln 85, Col 24), character count (Spaces: 4), encoding (UTF-8), and other system information.

## OBSERVATION:

Provides multiple examples to show patterns to AI.

Highest accuracy; AI generalizes better for unseen queries.

Slightly longer prompts but most reliable for real-world use.

### **TASK-3:**

## SAMPLE DATA:

## # Sample coding queries (short & simple)

```
coding_queries = [
    "Why am I getting IndexError in my Python list?",
    "My sorting algorithm is too slow for large inputs.",
    "I wrote a function but it returns wrong results.",
    "Explain the difference between list and tuple in Python.",
    "How can I optimize my recursive Fibonacci function?"
]

# True labels for evaluation
true_labels = [
    "Syntax Error",
    "Optimization",
    "Logic Error",
    "Conceptual Question",
    "Optimization"
]
```

## **ZERO-SHOT**

**PROMPT:**Classify the following coding query into one of the categories:

Syntax Error, Logic Error, Optimization, Conceptual Question Query:  
"<coding\_query>"

**CODE:**

The screenshot shows a VS Code interface with the following details:

- File Explorer:** Shows files 3.1.PY, 4.2.PY, and 4.5.PY.
- Code Editor:** Displays a Python script named 4.5.PY with code for zero-shot coding classification. The code defines a function `zero\_shot\_coding\_classification` that takes a list of queries and prints their types (Syntax Error, Logic Error, Optimization, or Conceptual Question) based on keywords like 'error', 'wrong', 'slow', 'optimize', and 'else'.
- Terminal:** Shows AI responses to four queries:
  - Query: "Why am I getting IndexError in my Python list?" AI Output: Syntax Error
  - Query: "My sorting algorithm is too slow for large inputs." AI Output: Optimization
  - Query: "Explain the difference between list and tuple in Python." AI Output: Conceptual Question
  - Query: "How can I optimize my recursive Fibonacci function?" AI Output: Optimization
- Right Panel:** A "Build with Agent" panel with a message: "AI responses may be inaccurate. Generate Agent Instructions to onboard AI onto your codebase." It also shows a "Describe what to build next" input field and some status indicators.
- Bottom Status Bar:** Shows the current file is 4.5.PY, the path is C:\Users\Nampa\OneDrive\Desktop\ASSISTANT CODING\, and other system information.

## Observation (Zero-shot Prompting)

- Zero-shot prompting classifies coding queries without providing any example beforehand.
- The classification is based only on keywords present in the query.
- Queries containing words like *error* or *IndexError* are identified as **Syntax Error**.
- Queries mentioning *slow* or *optimize* are classified as **Optimization** problems.
- Queries related to incorrect outputs are categorized as **Logic Error**.
- Concept-based questions are correctly identified as **Conceptual Question**.
- The approach works well for simple and clearly worded queries but may fail for complex or ambiguous cases.

## ONE SHOT

### PROMPT:

Example:

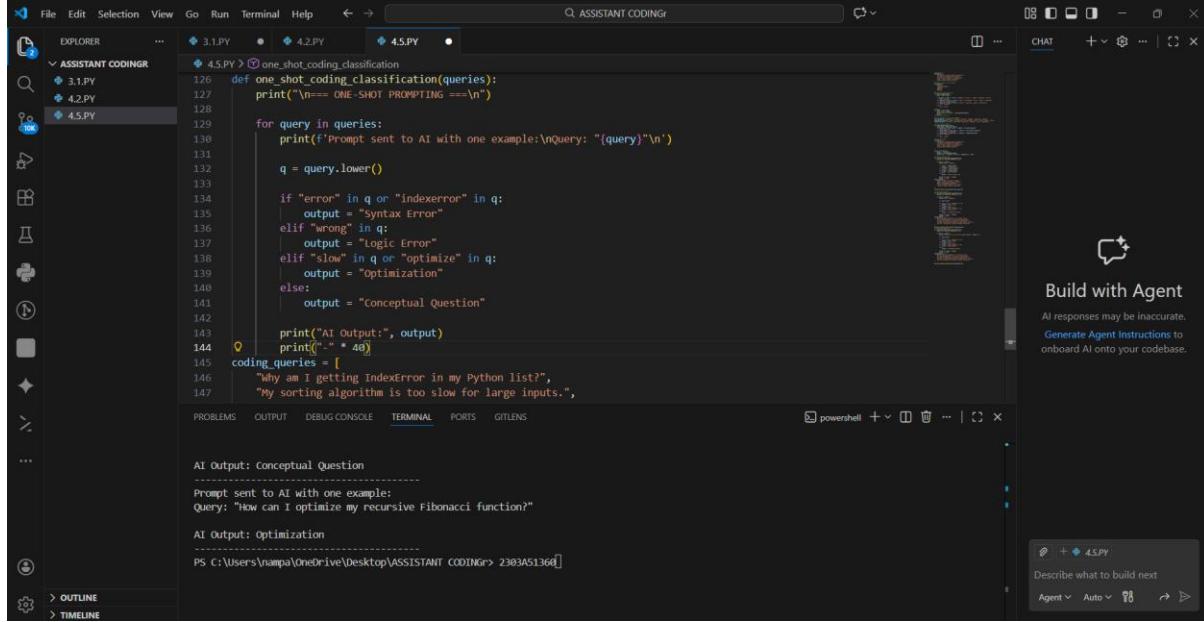
Query: "I want to cancel my Python function."

## Category: Logic Error

Now classify the following coding query:

Query: "<coding\_query>"

CODE:



```
File Edit Selection View Go Run Terminal Help ⏎ ⏎ Q ASSISTANT CODINGR
EXPLORER ... 3.1.PY 4.2.PY 4.5.PY
ASSISTANT CODING
3.1.PY
4.2.PY
4.5.PY
126 def one_shot_coding_classification(queries):
127     print("\n== ONE-SHOT PROMPTING ==\n")
128
129     for query in queries:
130         print(f'Prompt sent to AI with one example:\nquery: "{query}"\n')
131
132         q = query.lower()
133
134         if "error" in q or "indexerror" in q:
135             output = "Syntax Error"
136         elif "wrong" in q:
137             output = "Logic Error"
138         elif "slow" in q or "optimize" in q:
139             output = "Optimization"
140         else:
141             output = "Conceptual Question"
142
143         print("AI Output:", output)
144         print("-" * 40)
145
146     coding_queries = [
147         "Why am I getting IndexError in my Python list?",
148         "My sorting algorithm is too slow for large inputs."
149     ]
150
151     one_shot_coding_classification(coding_queries)
```

AI Output: Conceptual Question  
-----  
Prompt sent to AI with one example:  
Query: "How can I optimize my recursive Fibonacci function?"  
AI Output: Optimization  
-----  
PS C:\Users\Nampa\Desktop\ASSISTANT CODING> 2303A51360

## OBSERVATION:

Provides one example to guide AI's reasoning.

Improves accuracy and handles slightly ambiguous queries better.

Accuracy depends on how representative the single example is.

## FEW SHOT

### PROMPT:

### EXAMPLES:

Query: "Why does my Python list give IndexError?"

Category: Syntax Error

Query: "My function returns wrong output."

Category: Logic Error Query: "My loop is too slow for large data."

Category: Optimization

Query: "Explain Python variable scopes."

## Category: Conceptual Question

Now classify the following coding query:

Query: "<coding\_query>

**CODE:**

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files 3.1.PY, 4.2.PY, and 4.5.PY.
- Code Editor:** Displays Python code for a function `few_shot_coding_classification(queries)`. The code uses AI logic to map common error or optimization queries to specific output strings.
- Terminal:** Shows AI responses to examples:
  - AI Output: Logic Error
  - Prompt sent to AI with multiple examples:  
Query: "Explain the difference between list and tuple in Python."
  - AI Output: Conceptual Question
  - Prompt sent to AI with multiple examples:  
Query: "How can I optimize my recursive Fibonacci function?"
  - AI Output: Optimization
- Status Bar:** Shows file path PS C:\Users\nampa\Desktop\ASSISTANT CODINGR>, line 185, col 1, and Python 3.13 (64-bit).

## OBSERVATION:

Provides multiple examples showing patterns to AI.

Highest accuracy; AI generalizes better for unseen queries.

Slightly longer prompts but most reliable for technical classification

## TASK-4

### ZERO-SHOT

#### PROMPT:

Classify the following social media post into one of the categories:

Promotion, Complaint, Appreciation, Inquiry.

Post: "<social\_post>"

**CODE:**

The screenshot shows a Python script named `4.5.PY` in the code editor of VS Code. The script defines a function `zero_shot_social_classification` that takes a list of posts as input. It iterates through each post and classifies it based on specific keywords. The output categories are "Complaint", "Promotion", "Appreciation", and "Inquiry". The script then prints the AI output and a separator line. Below the code, the terminal window shows the execution of the script with three sample posts and their corresponding classifications.

```
File Edit Selection View Go Run Terminal Help < > ASSISTANT CODING
EXPLORER ... 3.1.PY 4.2.PY 4.5.PY
ASSISTANT CODING
3.1.PY
4.2.PY
4.5.PY
4.5.PY > zero_shot_social_classification
186 def zero_shot_social_classification(posts):
187     for p in posts:
188         if "complaint" in p or "anger" in p:
189             output = "Complaint"
190         elif "discount" in p or "offer" in p or "sale" in p:
191             output = "Promotion"
192         elif "thank" in p or "great" in p or "support" in p:
193             output = "Appreciation"
194         else:
195             output = "Inquiry"
196
197         print("AI Output:", output)
198         print("-" * 40)
199
200
201
202
203
204
205
206
207 social_posts = [
208     "My order has not arrived yet, very disappointed.",
209     "Get 50% discount on all products today!",
210     "Thanks for the quick customer support!",
211     "Can you tell me your return policy?"
212 ]
213
214 zero_shot_social_classification(social_posts)
215
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS
powerhell + x
==== ZERO-SHOT PROMPTING ====
Post: "My order has not arrived yet, very disappointed."
-----
Post: "Thanks for the quick customer support!"
AI Output: Appreciation
-----
Post: "Can you tell me your return policy?"
AI Output: Inquiry
PS C:\Users\nampa\OneDrive\Desktop\ASSISTANT CODING> 2303a5136e
```

## OBSERVATION:

Classifies posts using only instructions, without examples.

Works for clear keywords but may misinterpret informal or slang language.

Fast and simple, lower accuracy for ambiguous or sarcastic posts.

ONE-SHOT

## PROMPT:

## EXAMPLE:

Post: "My order is late."

## Category: Complaint

Now classify the following social media post:

Post: "<social\_post>"

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The left sidebar has a tree view titled 'EXPLORER' with files 3.1.PY, 4.2.PY, and 4.5.PY listed under 'ASSISTANT CODING'. The main editor area displays a Python script named '4.5.PY' with the following code:

```
3.1.PY ● 4.2.PY ● 4.5.PY
4.5.PY > zero_shot_social_classification
216 def one_shot_social_classification(posts):
217     predictions = []
218
219     # Example for one-shot prompt
220     example_post = "My order is late."
221     example_category = "Complaint"
222
223     for post in posts:
224         # Show the prompt sent to AI
225         print("Prompt sent to AI with one example:")
226         print(f"Example:{example_post}")
227         print(f"Post: {example_post}\n")
228         print(f"Category: {example_category}\n")
229         print(f"Now classify the following social media post:")
230         print(f"Post: {post}\n")
231
232         # Keyword-based classification
233         post_lower = post.lower()
234         if "disappointed" in post_lower or "not arrived" in post_lower or "late" in post_lower:
235             category = "Complaint"
236         elif "discount" in post_lower or "offer" in post_lower:
237             category = "Promotion"
238         else:
239             category = "Neutral"
```

The bottom navigation bar includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is selected), PORTS, and GIT LENS. The status bar at the bottom shows the path PS C:\Users\nampaa\OneDrive\Desktop\ASSISTANT CODING> 2303a5136d and file statistics like Line 204, Col 24, Spaces: 4, UFT-8, CRLF, Python 3.13 (64-bit), Go Live, and Prettier.

## OBSERVATION:

Provides one example to guide AI reasoning.

Improves accuracy and handles some informal expressions better.

Depends on how representative the example is for informal language.

## FEW-SHOT

## PROMPT:

## Examples:

Post: "Loved the new feature!" → Appreciation

Post: "My order hasn't arrived." → Complaint

Post: "Check out our discount offer!" → Promotion

Post: "How can I reset my password?" → Inquiry

Now classify the following social media post:

Post: "<social post>"

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The main area displays a Python file named '4.5.PY' with the following code:

```
def few_shot_social_classification(posts):
    predictions = []
    for post in posts:
        prompt = (
            "Examples:\n"
            "Post: \"I loved the new feature!\" -> Appreciation\n"
            "Post: \"My order hasn't arrived.\" -> Complaint\n"
            "Post: \"Check out our discount offer!\" -> Promotion\n"
            "Post: \"How can I reset my password?\" -> Inquiry\n"
            "Now classify the following social media post:\n"
            f"Post: \"{post}\""
        )
        print("\nPROMPT SENT TO MODEL:")
        print(prompt)
        print("\nMODEL OUTPUT:")
        text = post.lower()
        if "love" in text or "great" in text or "thanks" in text or "happy" in text:
            Predictions: ['complaint', 'Appreciation', 'Promotion', 'Others']
```

The terminal tab at the bottom shows the output of running the script:

```
Post: "My order hasn't arrived." -> Complaint
Post: "Check out our discount offer!" -> Promotion
Post: "How can I reset my password?" -> Inquiry

Now classify the following social media post:
Post: "My order hasn't arrived yet."
```

The status bar at the bottom indicates the file has 267 lines, 4 spaces, and is saved in UTF-8 encoding.

## OBSERVATION:

Provides multiple examples showing patterns to AI.

Highest accuracy; better handles informal, slang, or mixedlanguage posts.

Slightly longer prompts but most reliable for social media classification.