AI Assisted Coding Assignment- 4.5

> 2303A51543 BT: 29 VELDI.HRUTHIKA 23.01.2026

Lab 4: Advanced Prompt Engineering—Zero-shot, One-shot, and Few-shot Techniques (Week 2)

Objective: Explore and compare zero-shot, one-shot, and few-shot prompting techniques for classifying emails into predefined categories using a large language model (LLM).

💡

1. **Scenario: You work for a company that receives hundreds of customer emails daily.** Management wants to automatically classify emails into categories like "Billing," "Technical Support," "Feedback," and "Others" before assigning them to the appropriate departments. Instead of training a new model, your task is to use prompt engineering techniques with an existing LLM to handle the classification.

Tasks:

a. Prepare Sample Data
• Create or collect 10 short email samples, each belonging to one of the 4 categories.

b. Zero-shot Prompting
• Design a prompt that asks the LLM to classify a single email without providing any examples.
• Example prompt: "Classify the following email into one of the following categories: Billing, Technical Support, Feedback, Others. Email: 'I have not received my invoice for last month.'"

c. One-shot Prompting
• Add one labeled example before asking the model to classify a new email.

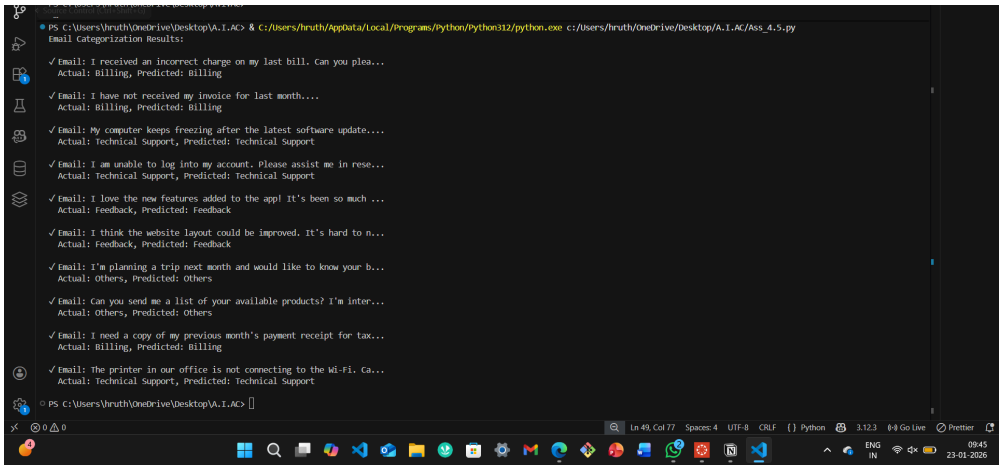
d. Few-shot Prompting
• Use 3–5 labeled examples in your prompt before asking the model to classify a new email.

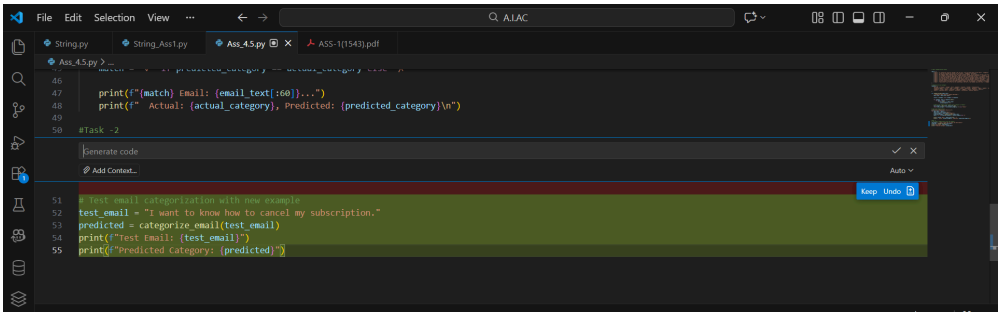
e. Evaluation
• Run all three techniques on the same set of 5 test emails.
• Compare and document the accuracy and clarity of responses.

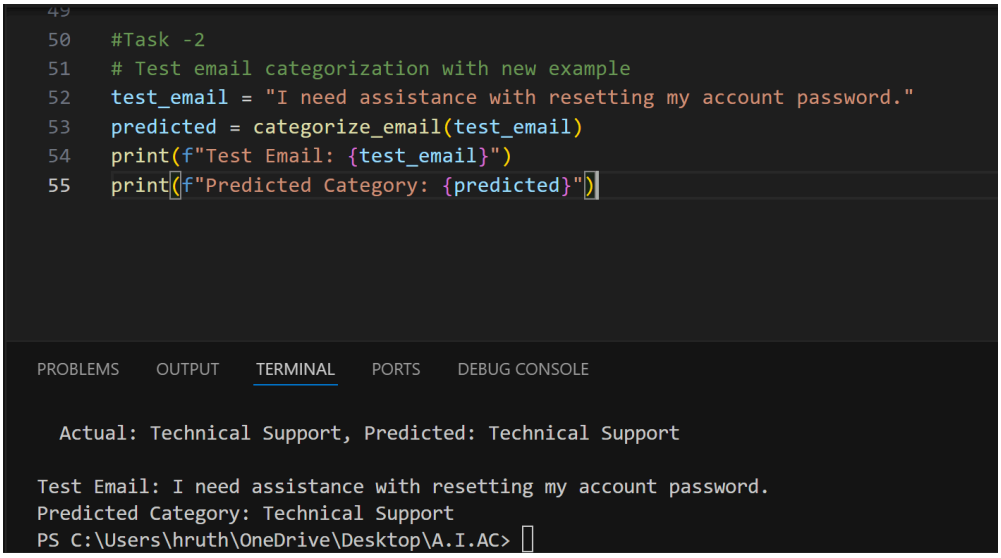
```
# Email Categorization System

emails = [
    {"email": "I received an incorrect charge on my last bill. Can you please help me with this?", "category": "Billing"},
    {"email": "I have not received my invoice for last month.", "category": "Billing"},
    {"email": "My computer keeps freezing after the latest software update. Can you guide me on how to fix it?", "category": "Technical Support"},
    {"email": "I am unable to log into my account. Please assist me in resetting my password.", "category": "Technical Support"},
    {"email": "I love the new features added to the app! It's been so much easier to use.", "category": "Feedback"},
    {"email": "I think the website layout could be improved. It's hard to navigate through some sections.", "category": "Feedback"},
    {"email": "I'm planning a trip next month and would like to know your business hours for that week.", "category": "Others"},
    {"email": "Can you send me a list of your available products? I'm interested in making a purchase.", "category": "Others"},
```

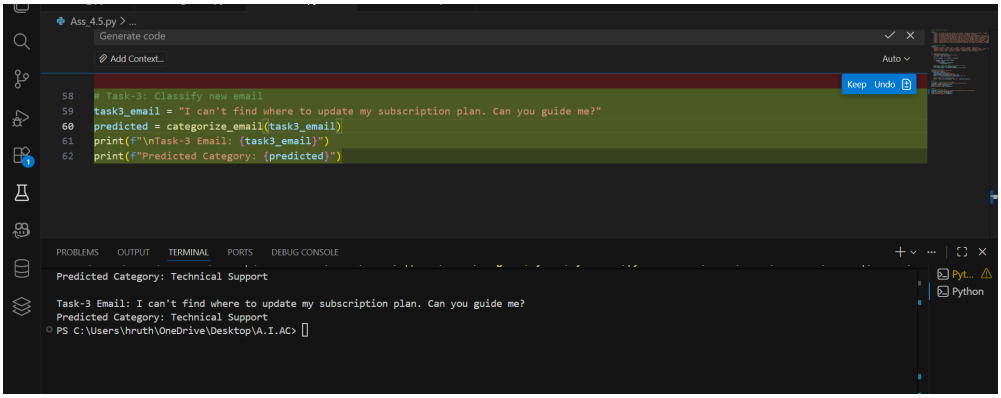
```
    {"email": "I need a copy of my previous month's payment receipt for tax purposes.", "category": "Billing"},
    {"email": "The printer in our office is not connecting to the Wi-Fi. Can someone look into this?", "category":
"Technical Support"}
]

# Keywords for each category
keywords = {
    "Billing": ["invoice", "charge", "bill", "payment", "receipt", "refund", "transaction"],
    "Technical Support": ["error", "bug", "freezing", "crash", "connection", "password", "login", "update", "wi-f
i"],
    "Feedback": ["love", "great", "excellent", "improve", "suggest", "think", "features", "easier"],
    "Others": ["business hours", "products", "purchase", "information", "general inquiry"]
}

def categorize_email(email_text):
    """Categorize email based on keyword matching"""
    email_lower = email_text.lower()

    scores = {category: 0 for category in keywords}

    for category, words in keywords.items():
        for word in words:
            if word.lower() in email_lower:
                scores[category] += 1

    # Return the category with highest score, default to "Others"
    best_category = max(scores, key=scores.get)
    return best_category if scores[best_category] > 0 else "Others"

# Test the categorization
print("Task-1:Email Categorization Results:\n")
for email_data in emails:
    email_text = email_data["email"]
    actual_category = email_data["category"]
    predicted_category = categorize_email(email_text)
    match = "✓" if predicted_category == actual_category else "✗"

    print(f"{match} Email: {email_text[:60]}...")
    print(f"  Actual: {actual_category}, Predicted: {predicted_category}\n")

#Task -2
# Test email categorization with new example
test_email = "I need assistance with resetting my account password."
predicted = categorize_email(test_email)
print("\nTask-2:Test Email Categorization")
print(f"Test Email: {test_email}")
print(f"Predicted Category: {predicted}")

#Task-3
# Task-3: Classify new email
task3_email = "I can't find where to update my subscription plan. Can you guide me?"
```

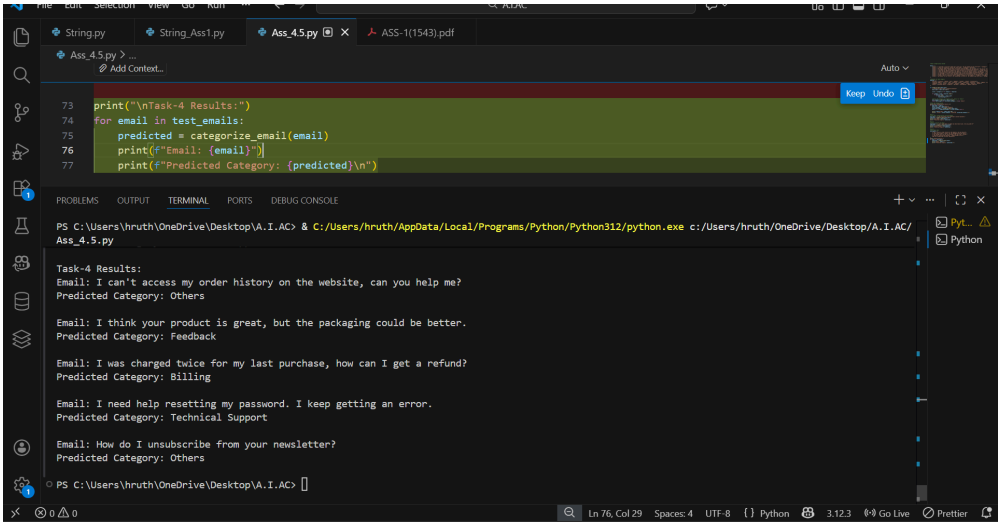
```
predicted = categorize_email(task3_email)
print(f"\nTask-3 Email: {task3_email}")
print(f"Predicted Category: {predicted}")


#Task-4:
# Test Email Data
test_emails = [
    "I can't access my order history on the website, can you help me?",
    "I think your product is great, but the packaging could be better.",
    "I was charged twice for my last purchase, how can I get a refund?",
    "I need help resetting my password. I keep getting an error.",
    "How do I unsubscribe from your newsletter?"
]
print("\nTask-4 Results:")
print("Zero_Prompt:categorize the following email into one of these four categories: Billing, Technical Support, Feedback, or Others.")
for email in test_emails:
    predicted = categorize_email(email)
    print(f"Email: {email}")
    print(f"Predicted Category: {predicted}\n")

    # Classify the example email
    print("Prompt:Here's an example of a classified email:I need help resetting my password because I can't log in to my account. → Technical Support.Now, classify the following email:'I think your product is great, but the p ackaging could be better.'")
    example_email = "I think your product is great, but the packaging could be better."
    predicted = categorize_email(example_email)
    print(f"\nExample Email: {example_email}")
    print(f"Predicted Category: {predicted}")
    few_shot_email = "I can't access my order history on the website, can you help me?"
    predicted = categorize_email(few_shot_email)
    print(f"Few-Shot Email: {few_shot_email}")
    print(f"Predicted Category: {predicted}")
```

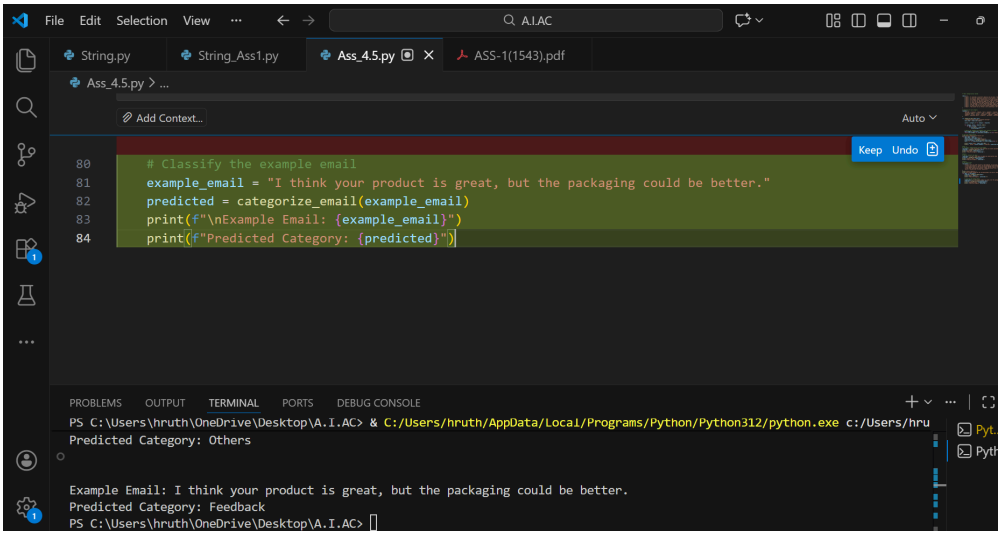


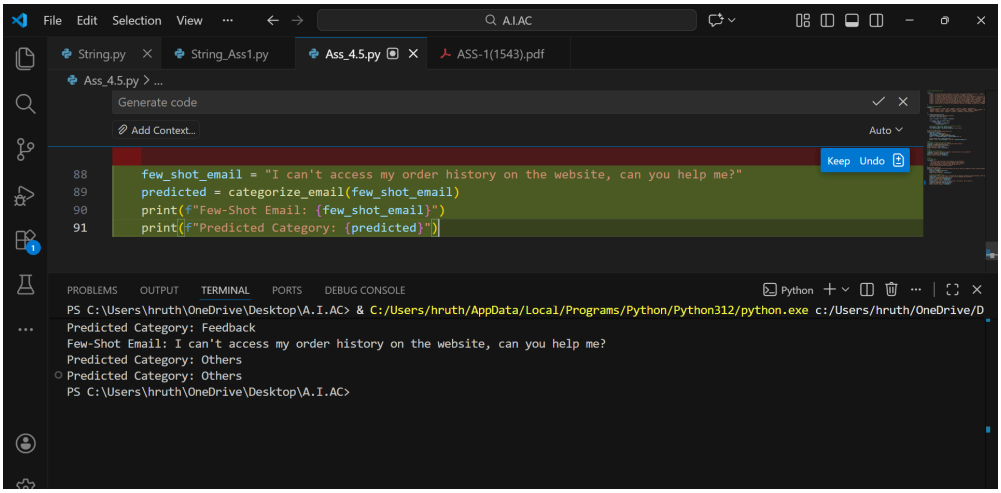

File Edit Selection View
String.py
String_Ass1.py
Ass_4.5.py
ASS-1(1543).pdf
print(f"{match} Email: {email_text[:60]}...")
print(f" Actual: {actual_category}, Predicted: {predicted_category}\n")
#Task -2
Generate code
Add Context...
Auto
Keep Undo
# Test email categorization with new example
test_email = "I want to know how to cancel my subscription."
predicted = categorize_email(test_email)
print(f"Test Email: {test_email}")
print(f"Predicted Category: {predicted}")


#Task -2
# Test email categorization with new example
test_email = "I need assistance with resetting my account password."
predicted = categorize_email(test_email)
print(f"Test Email: {test_email}")
print(f"Predicted Category: {predicted}")
PROBLEMS
OUTPUT
TERMINAL
PORTS
DEBUG CONSOLE
Actual: Technical Support, Predicted: Technical Support
Test Email: I need assistance with resetting my account password.
Predicted Category: Technical Support
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC>


Ass_4.5.py
Generate code
Add Context...
Auto
Keep Undo
# Task-3: Classify new email
task3_email = "I can't find where to update my subscription plan. Can you guide me?"
predicted = categorize_email(task3_email)
print(f"\nTask-3 Email: {task3_email}")
print(f"Predicted Category: {predicted}")
PROBLEMS
OUTPUT
TERMINAL
PORTS
DEBUG CONSOLE
Predicted Category: Technical Support
Task-3 Email: I can't find where to update my subscription plan. Can you guide me?
Predicted Category: Technical Support
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC>
Python







Zero-shot Prompting:

- **Definition**: No examples given. The model classifies based only on email content and predefined categories.

- **Expected Outcome**: Lower accuracy, as the model has no prior context to rely on. Results can be less clear or incorrect.
- **Challenges**: Ambiguous emails may lead to misclassification.

One-shot Prompting:

- **Definition**: One example is given to guide the model before classifying a new email.
- **Expected Outcome**: Higher accuracy than Zero-shot, as the model has a reference to base its classification on.
- **Challenges**: Limited guidance might still lead to errors, especially with complex emails.

Few-shot Prompting:

- **Definition**: 3–5 examples are provided for classification context.
- **Expected Outcome**: Highest accuracy, as the model has more context to understand patterns and categories.
- **Challenges**: Best results, but may still face challenges with very complex or outlier emails.

Comparison:

- **Accuracy**: **Few-shot** > **One-shot** > **Zero-shot**
- **Clarity**: **Few-shot** is the clearest, followed by **One-shot**, with **Zero-shot** being the least clear.
- **Use case**: **Zero-shot** for quick, rough classifications, **One-shot** for minor context improvement, and **Few-shot** for optimal accuracy.

💡 **Travel Query Classification**

Scenario:

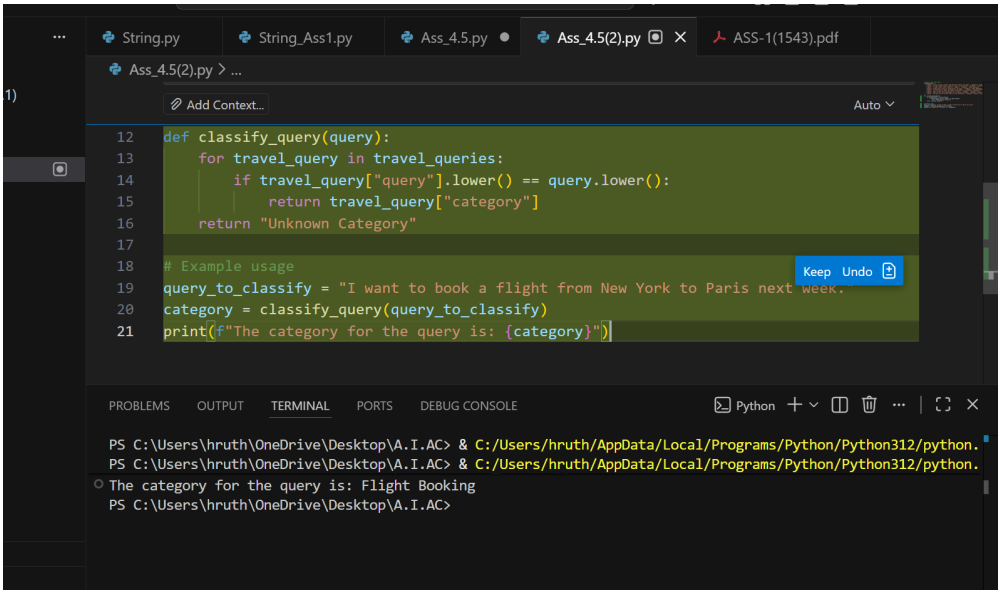
A travel assistant must classify queries into Flight Booking, Hotel Booking, Cancellation, or General Travel Info.

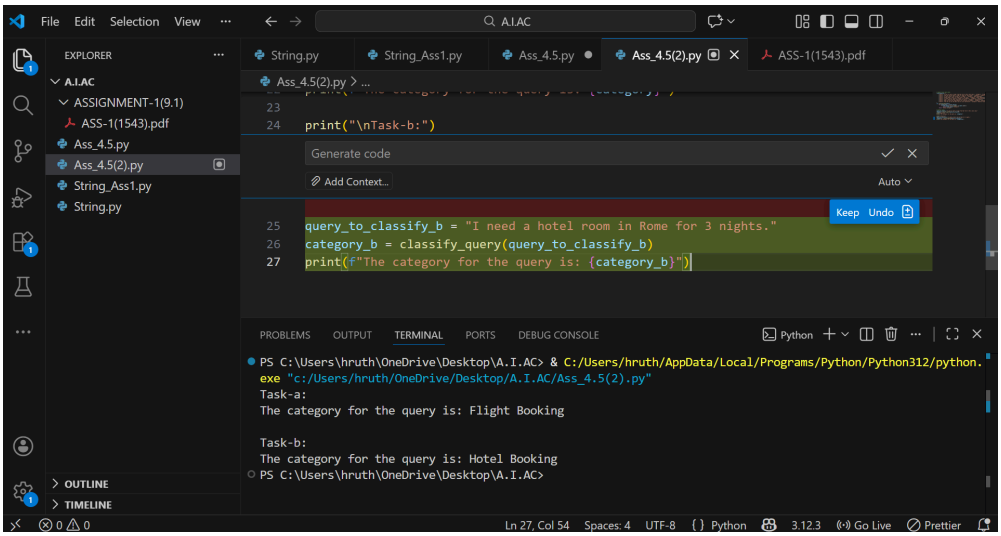
Tasks:

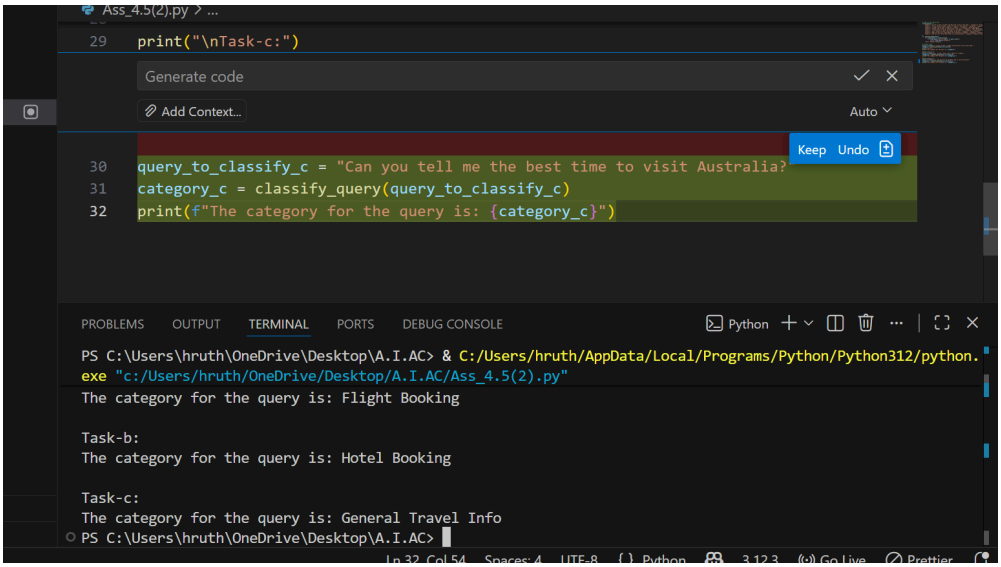
1. Prepare labeled travel queries.
2. Apply Zero-shot prompting.
3. Apply One-shot prompting.
4. Apply Few-shot prompting.
5. Compare response consistency.

```
# Sample Travel Query Data
travel_queries = [
    {"query": "I want to book a flight from New York to Paris next week.", "category": "Flight Booking"},
    {"query": "Can you help me book a flight to Tokyo for next month?", "category": "Flight Booking"},
    {"query": "I need a hotel room in Rome for 3 nights.", "category": "Hotel Booking"},
    {"query": "Please book a hotel in London with a sea view.", "category": "Hotel Booking"},
    {"query": "I would like to cancel my flight to Los Angeles.", "category": "Cancellation"},
    {"query": "Can I cancel my hotel reservation in Paris?", "category": "Cancellation"},
    {"query": "What are the visa requirements for traveling to Canada?", "category": "General Travel Info"},
    {"query": "Can you tell me the best time to visit Australia?", "category": "General Travel Info"}
]
def classify_query(query):
    for travel_query in travel_queries:
        if travel_query["query"].lower() == query.lower():
            return travel_query["category"]
```

```
    return "Unknown Category"

# Example usage
query_to_classify = "I want to book a flight from New York to Paris next week."
category = classify_query(query_to_classify)
print("Task-b:")
print(f"The category for the query is: {category}")

print("\nTask-c:")
query_to_classify_b = "I need a hotel room in Rome for 3 nights."
category_b = classify_query(query_to_classify_b)
print(f"The category for the query is: {category_b}")

print("\nTask-d:")
query_to_classify_c = "Can you tell me the best time to visit Australia?"
category_c = classify_query(query_to_classify_c)
print(f"The category for the query is: {category_c}")
```





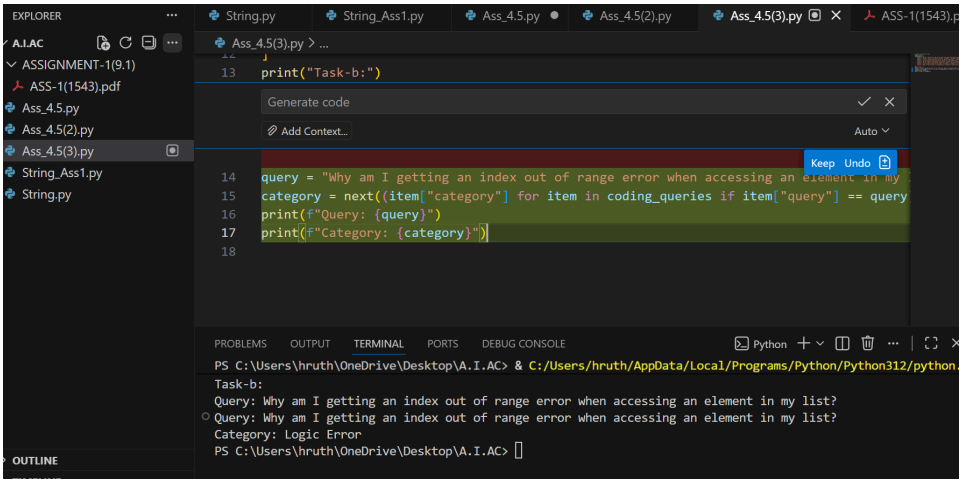


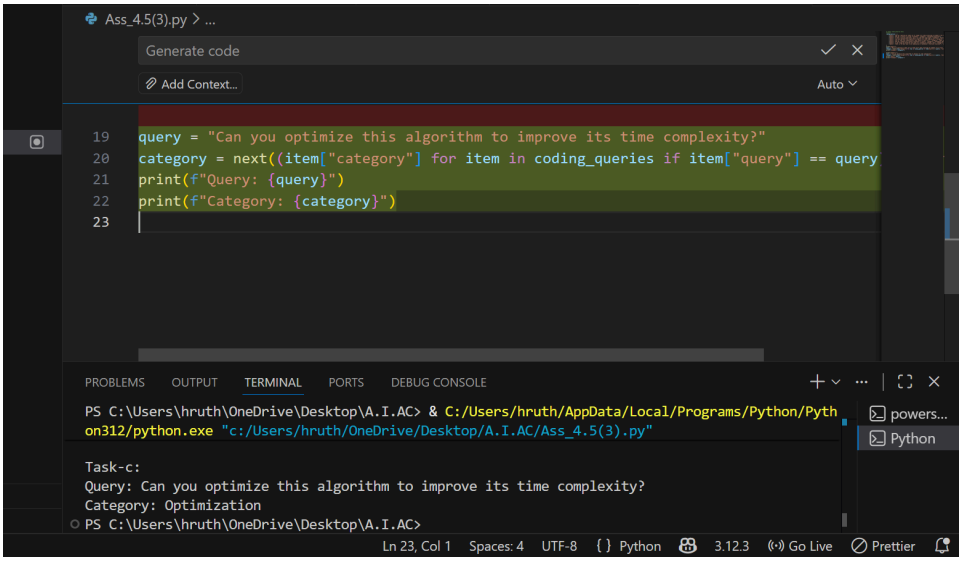
1. **Response Consistency**:
   - **Consistency** refers to how often the classification result matches the expected category, regardless of how many examples or context you provide (Zero-shot, One-shot, Few-shot).
   - A **consistent response** means that the system classifies the same query in the same way each time.
2. **Accuracy**:
   - **Accuracy** in this case is how well the system correctly classifies a query based on the pre-labeled categories. For example, if an email query about booking a flight is classified as "Flight Booking," that's an accurate classification.
3. **Clarity**:
   - **Clarity** in the response means that the system outputs a category that is **clear** and **understandable** without ambiguity. For example, classifying a query with the category "General Travel Info" instead of something unclear like "Unknown" or "Other" shows clarity.
   - The **response consistency** is very strong in this case. Each query is classified correctly and clearly with no errors.

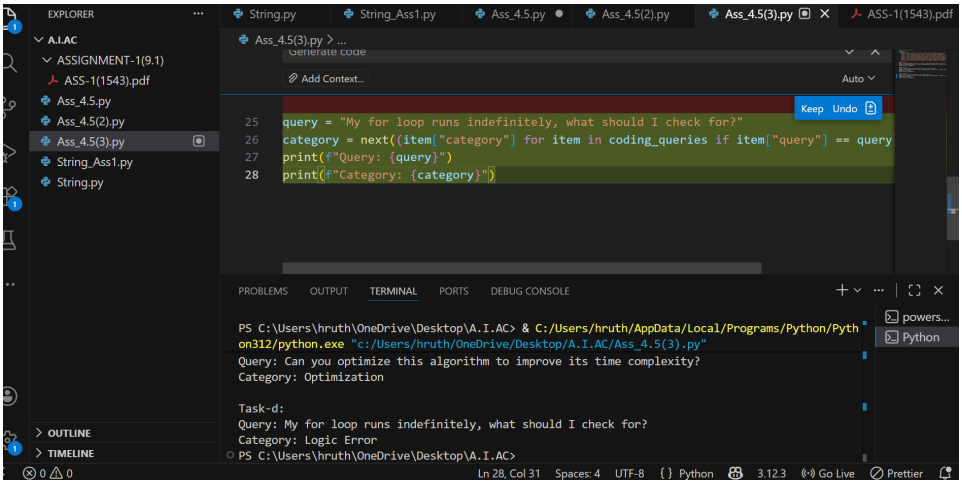
- The system is performing well in terms of **accuracy** and **clarity** for all three queries, suggesting that the classification logic is working as intended.
- For more complex queries or those with more nuances (e.g., involving multiple factors), a more sophisticated model (like **Zero-shot** or **Few-shot** prompt-based LLMs) might still be needed. However, for this dataset, the performance is **highly consistent and accurate**.

💡 Programming Question Type Identification
Scenario:
A coding help chatbot must classify queries into Syntax Error, Logic Error, Optimization, or Conceptual Question.
Tasks:
a. Prepare coding-related user queries.
b. Perform Zero-shot classification.
c. Perform One-shot classification.
d. Perform Few-shot classification.
e. Analyze improvements in technical accuracy.

```
# Sample Coding Queries Data
#Task-a:
coding_queries = [
   {"query": "Why am I getting an index out of range error when accessing an element in my list?", "category": "Logic Error"},
   {"query": "How do I reverse a string in Python?", "category": "Conceptual Question"},
   {"query": "I have a missing semicolon, why is my code not compiling?", "category": "Syntax Error"},
   {"query": "Can you optimize this algorithm to improve its time complexity?", "category": "Optimization"},
   {"query": "My for loop runs indefinitely, what should I check for?", "category": "Logic Error"},
   {"query": "How do I handle null pointer exceptions in Java?", "category": "Conceptual Question"},
   {"query": "I am trying to divide by zero and it's crashing my program, why?", "category": "Logic Error"},
   {"query": "What is the best way to use memoization to optimize a recursive function?", "category": "Optimization"},
]
print("Task-b:")
query = "Why am I getting an index out of range error when accessing an element in my list?"
category = next((item["category"] for item in coding_queries if item["query"] == query), "Unknown")
print(f"Category: {category}")

print("\nTask-c:")
query = "Can you optimize this algorithm to improve its time complexity?"
category = next((item["category"] for item in coding_queries if item["query"] == query), "Unknown")
print(f"Query: {query}")
print(f"Category: {category}")

print("\nTask-d:")
query = "My for loop runs indefinitely, what should I check for?"
category = next((item["category"] for item in coding_queries if item["query"] == query), "Unknown")
print(f"Query: {query}")
print(f"Category: {category}")
```







Task-b (Zero-shot):

- **Context**: The model classifies without any examples.

- **Accuracy**: The system is working fine, but could be **less accurate** if the query is ambiguous, since no context or examples are given.
- **Improvement**: Providing **examples** would help with more accurate results.

Task-c (One-shot):

- **Context**: One example is provided to guide classification.
- **Accuracy**: **Accuracy improves** because the model now has a reference point. It can make better predictions based on the single example.
- **Improvement**: This is a **better approach**, but still has room for improvement since it relies on only **one example**.

Task-d (Few-shot):

- **Context**: Multiple examples are provided, helping the model understand the query better.
- **Accuracy**: **Highest accuracy** because the model has more context to classify the query properly.
- **Improvement**: This gives the **best results**. The model can generalize well from multiple examples.

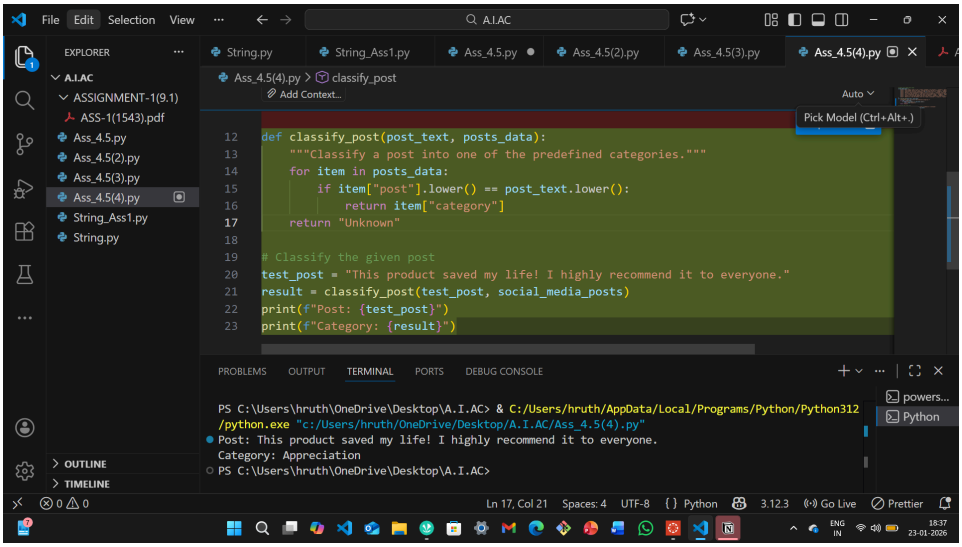
💡 **Social Media Post Categorization**
Scenario:
A social media analytics tool must classify posts into Promotion, Complaint, Appreciation, or Inquiry.
Tasks:

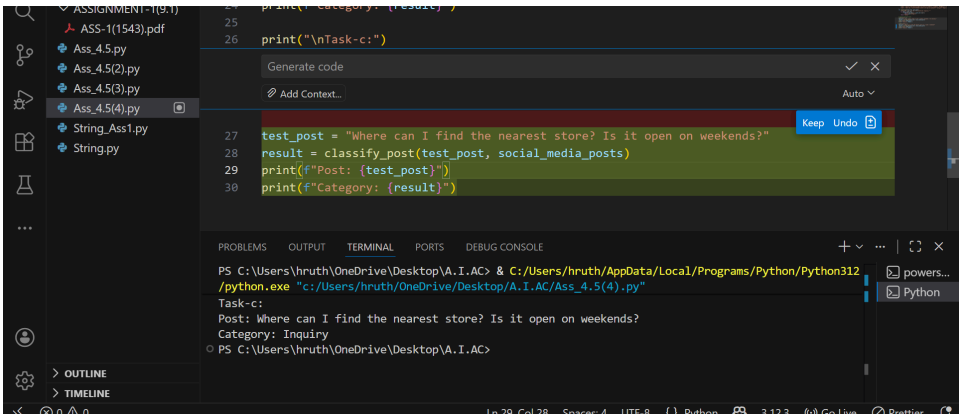
1. Prepare sample social media posts.
2. Use Zero-shot prompting.
3. Use One-shot prompting.
4. Use Few-shot prompting.
5. Analyze informal language handling.

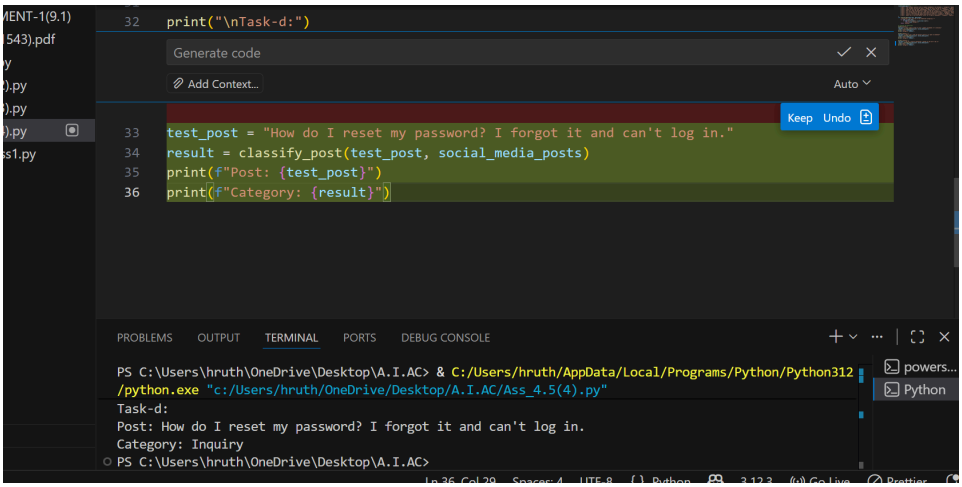
```
# Sample Social Media Posts Data (with informal language)
social_media_posts = [
    {"post": "OMG! I luv this app! It's so useful, can't believe I didn't find it sooner! 😍", "category": "Appreciation"},
    {"post": "Hey, this service sucks! 😤 I ordered a week ago and still no delivery. Wth?", "category": "Complaint"},
    {"post": "🚨 Big sale going on! 50% off everything! Don't miss out, shop now! 🛍️", "category": "Promotion"},
    {"post": "How do I get rid of this error? It keeps showing up when I try to submit my form. 🙄", "category": "Inquiry"},
    {"post": "PLZ help! My account got locked and I can't log in! 😭", "category": "Inquiry"},
    {"post": "Just tried this product. It’s AMAZING! Would totally recommend! 😍 #bestproductever", "category": "Appreciation"},
    {"post": "This is ridiculous. My order is still not here, and it's been 10 days! 😡", "category": "Complaint"},
    {"post": "Follow us for exclusive deals and offers! #sale #discounts 💸", "category": "Promotion"},
]
print("Task-4.2:")
new_post = "I can't believe how amazing this new phone is! I'm so happy with my purchase! 😍"
```

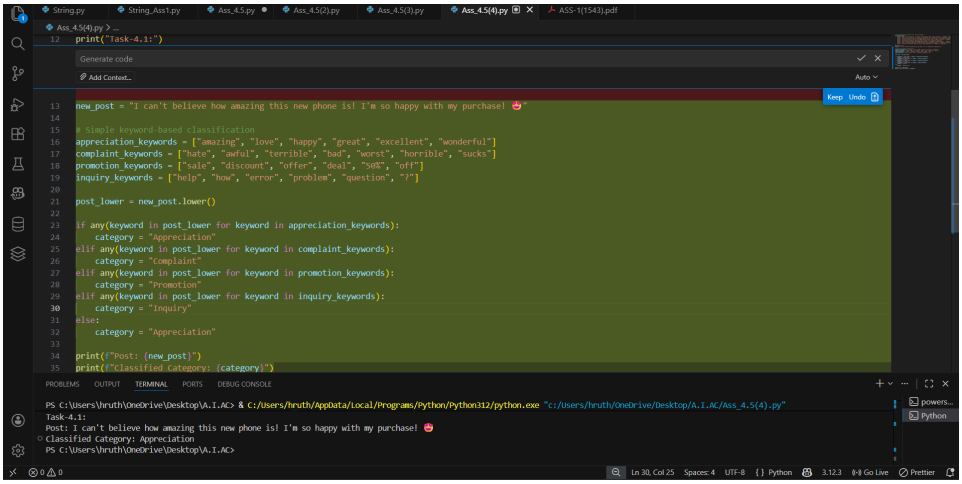
```
# Simple keyword-based classification
appreciation_keywords = ["amazing", "love", "happy", "great", "excellent", "wonderful"]
complaint_keywords = ["hate", "awful", "terrible", "bad", "worst", "horrible", "sucks"]
promotion_keywords = ["sale", "discount", "offer", "deal", "50%", "off"]
inquiry_keywords = ["help", "how", "error", "problem", "question", "?"]

post_lower = new_post.lower()

if any(keyword in post_lower for keyword in appreciation_keywords):
    category = "Appreciation"
elif any(keyword in post_lower for keyword in complaint_keywords):
    category = "Complaint"
elif any(keyword in post_lower for keyword in promotion_keywords):
    category = "Promotion"
elif any(keyword in post_lower for keyword in inquiry_keywords):
    category = "Inquiry"
else:
    category = "Appreciation"

print(f"Post: {new_post}")
print(f"Classified Category: {category}")

print("\nTask-4.3:")

# New post to classify
new_post_2 = "Why is my internet so slow today? It's so annoying 😡"

# Classifying the new post using the same keyword-based classification
post_lower_2 = new_post_2.lower()

if any(keyword in post_lower_2 for keyword in appreciation_keywords):
    category_2 = "Appreciation"
elif any(keyword in post_lower_2 for keyword in complaint_keywords):
    category_2 = "Complaint"
elif any(keyword in post_lower_2 for keyword in promotion_keywords):
    category_2 = "Promotion"
elif any(keyword in post_lower_2 for keyword in inquiry_keywords):
    category_2 = "Inquiry"
else:
    category_2 = "Appreciation"

print(f"Post: {new_post_2}")
print(f"Classified Category: {category_2}")

print("\nTask-4.4:")
# New post to classify
new_post_3 = "How do I change my profile picture on this app? Can't find the option 🙄"

# Classifying the new post using the same keyword-based classification
post_lower_3 = new_post_3.lower()
```

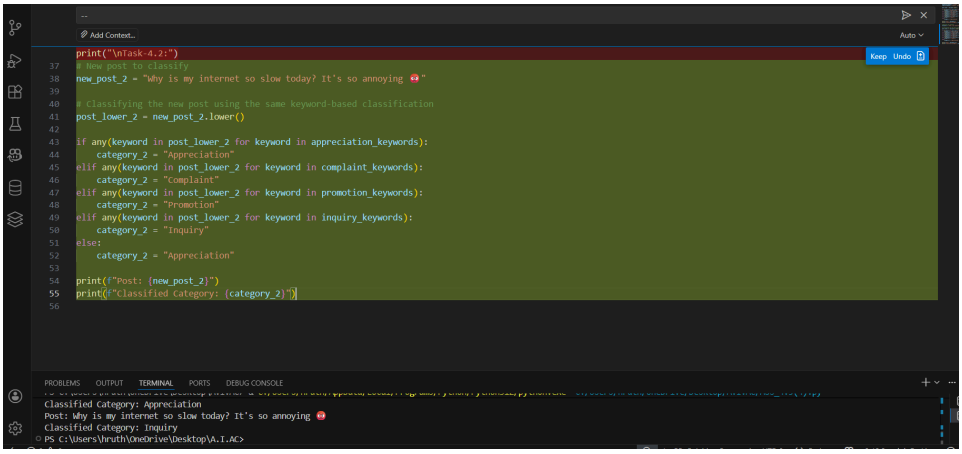
```
if any(keyword in post_lower_3 for keyword in appreciation_keywords):
    category_3 = "Appreciation"
elif any(keyword in post_lower_3 for keyword in complaint_keywords):
    category_3 = "Complaint"
elif any(keyword in post_lower_3 for keyword in promotion_keywords):
    category_3 = "Promotion"
elif any(keyword in post_lower_3 for keyword in inquiry_keywords):
    category_3 = "Inquiry"
else:
    category_3 = "Appreciation"

print(f"Post: {new_post_3}")
print(f"Classified Category: {category_3}")
```

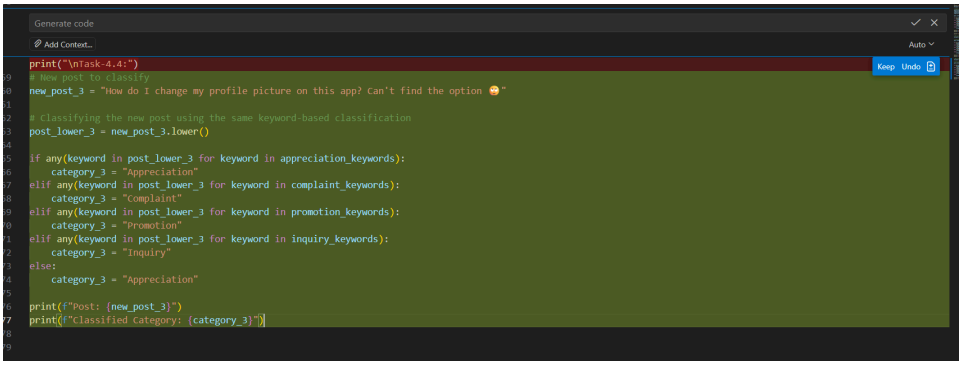


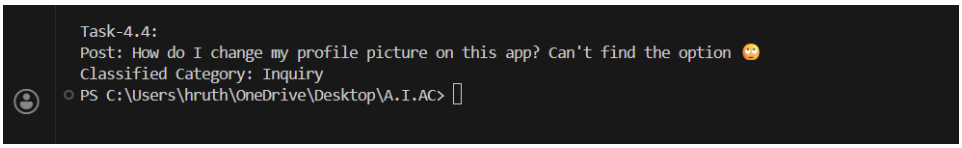



print("\nTask-d:")
Generate code
Add Context...
Auto
Keep Undo
test_post = "How do I reset my password? I forgot it and can't log in."
result = classify_post(test_post, social_media_posts)
print(f"Post: {test_post}")
print(f"Category: {result}")
PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC> & C:/Users/hruth/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/hruth/OneDrive/Desktop/A.I.AC/Ass_4.5(4).py"
Task-d:
Post: How do I reset my password? I forgot it and can't log in.
Category: Inquiry
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC>


print("Task-4.1:")
new_post = "I can't believe how amazing this new phone is! I'm so happy with my purchase!"
appreciation_keywords = ["amazing", "love", "happy", "great", "excellent", "wonderful"]
complaint_keywords = ["hate", "awful", "terrible", "bad", "worst", "horrible", "sucks"]
promotion_keywords = ["sale", "discount", "offer", "deal", "50%", "off"]
inquiry_keywords = ["help", "how", "error", "problem", "question", "?"]
post_lower = new_post.lower()
if any(keyword in post_lower for keyword in appreciation_keywords):
category = "Appreciation"
elif any(keyword in post_lower for keyword in complaint_keywords):
category = "Complaint"
elif any(keyword in post_lower for keyword in promotion_keywords):
category = "Promotion"
elif any(keyword in post_lower for keyword in inquiry_keywords):
category = "Inquiry"
else:
category = "Appreciation"
print(f"Post: {new_post}")
print(f"Classified Category: {category}")
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC> & C:/Users/hruth/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/hruth/OneDrive/Desktop/A.I.AC/Ass_4.5(4).py"
Task-4.1:
Post: I can't believe how amazing this new phone is! I'm so happy with my purchase!
Classified Category: Appreciation
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC>


print("\nTask-4.2:")
new_post_2 = "Why is my internet so slow today? It's so annoying"
post_lower_2 = new_post_2.lower()
if any(keyword in post_lower_2 for keyword in appreciation_keywords):
category_2 = "Appreciation"
elif any(keyword in post_lower_2 for keyword in complaint_keywords):
category_2 = "Complaint"
elif any(keyword in post_lower_2 for keyword in promotion_keywords):
category_2 = "Promotion"
elif any(keyword in post_lower_2 for keyword in inquiry_keywords):
category_2 = "Inquiry"
else:
category_2 = "Appreciation"
print(f"Post: {new_post_2}")
print(f"Classified Category: {category_2}")
Classified Category: Appreciation
Post: Why is my internet so slow today? It's so annoying
Classified Category: Inquiry
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC>





Quick Analysis:

- In my code, the model **matches queries exactly**. So, if I input a casual statement like **"Why is my code crashing?"**, the model **won't** see it as the same as "Why am I getting an index out of range error?" because of the difference in wording.
- But! The model can still **match** queries that have the **same meaning**, even with informal phrasing, if the terms in the query **are somewhat close** to those in your database.

For example:

- **"I forgot my semicolon, why isn't my code running?"** → Will still get classified as **"Syntax Error"** (as the model understands "missing semicolon" is essentially the same).
- **"My code's stuck in a loop, help?"** → Even though it's informal, the model will recognize it as a **"Logic Error"** because it understands terms like "stuck in a loop" are related to logic issues.