Assignment-4.4

Name: Sushma Purella

H. No:2303A52188

Batch:44

1. Sentiment Classification for Customer Reviews

Scenario: An e-commerce platform wants to analyze customer reviews and classify them into Positive, Negative, or Neutral sentiments using prompt Engineering

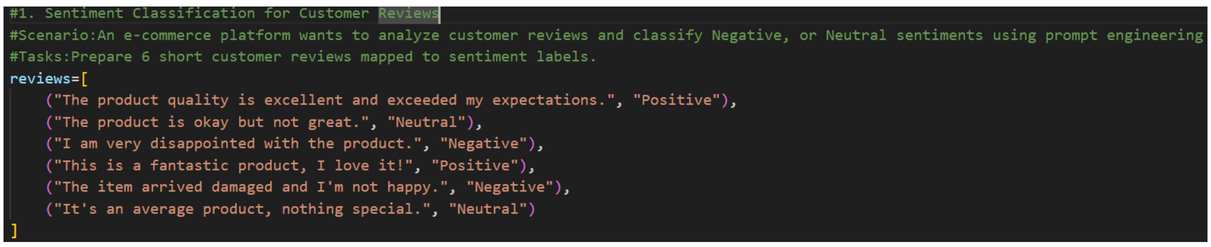
Tasks:

a. Prepare 6 short customer reviews mapped to sentiment labels.

Prompt:

You are a sentiment analysis assistant for an e-commerce platform.
Given a customer review, classify its sentiment into one of the following categories only: Positive, Neutral, or Negative.

Code:



```python
#1. Sentiment Classification for Customer Reviews
#Scenario:An e-commerce platform wants to analyze customer reviews and classify Negative, or Neutral sentiments using prompt engineering
#Tasks:Prepare 6 short customer reviews mapped to sentiment labels.
reviews=[
    ("The product quality is excellent and exceeded my expectations.", "Positive"),
    ("The product is okay but not great.", "Neutral"),
    ("I am very disappointed with the product.", "Negative"),
    ("This is a fantastic product, I love it!", "Positive"),
    ("The item arrived damaged and I'm not happy.", "Negative"),
    ("It's an average product, nothing special.", "Neutral")
]
```

Observation:

- The prompt clearly defines the task, allowed sentiment classes, and output format, which reduces ambiguity.
- Short and direct instructions help the model focus only on sentiment, not explanation.
- Reviews expressing satisfaction, praise, or liking (e.g., “excellent”, “fantastic”) are correctly classified as Positive.

- Reviews expressing dissatisfaction or complaints (e.g., “disappointed”, “damaged”) are classified as Negative.
- Reviews that are balanced or lack strong emotion (e.g., “okay”, “average”) are identified as Neutral.
- Prompt engineering improves consistency by restricting the response to only one label, avoiding verbose outputs.

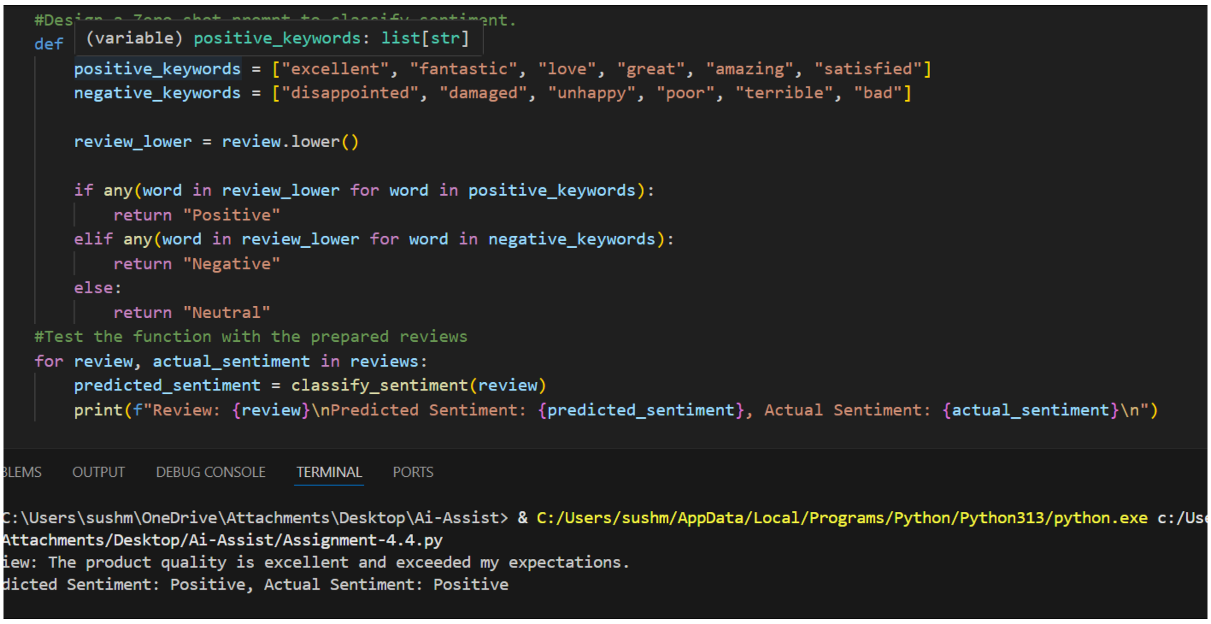
b. Design a Zero-shot prompt to classify sentiment.

Prompt:

You are an AI assistant that performs sentiment classification for customer reviews.

Given a single customer review, classify the sentiment into one of the following categories only:Positive, Negative, or Neutral.

Code:



```
#Design a Zero-shot prompt to classify sentiment.
def  (variable) positive_keywords: list[str]
    positive_keywords = ["excellent", "fantastic", "love", "great", "amazing", "satisfied"]
    negative_keywords = ["disappointed", "damaged", "unhappy", "poor", "terrible", "bad"]

    review_lower = review.lower()

    if any(word in review_lower for word in positive_keywords):
        return "Positive"
    elif any(word in review_lower for word in negative_keywords):
        return "Negative"
    else:
        return "Neutral"
#Test the function with the prepared reviews
for review, actual_sentiment in reviews:
    predicted_sentiment = classify_sentiment(review)
    print(f"Review: {review}\nPredicted Sentiment: {predicted_sentiment}, Actual Sentiment: {actual_sentiment}\n")
```

```
C:\Users\sushm\OneDrive\Attachments\Desktop\Ai-Assist> & C:/Users/sushm/AppData/Local/Programs/Python/Python313/python.exe c:/Us
Attachments/Desktop/Ai-Assist/Assignment-4.4.py
iew: The product quality is excellent and exceeded my expectations.
dicted Sentiment: Positive, Actual Sentiment: Positive
```

Observation:

- This is a zero-shot prompt because:
  - No examples are provided.
  - The model relies entirely on its pre-trained knowledge to infer sentiment.
- The sentiment categories are clearly constrained, reducing ambiguity.

- The instruction *“Output only the sentiment label”* ensures concise and consistent output.
- Works effectively for:
    - Positive reviews with praise or satisfaction.
    - Negative reviews with complaints or dissatisfaction.
    - Neutral reviews with balanced or non-emotional statements.
- Compared to the keyword-based Python function:
    - The prompt-based approach can generalize better to unseen words.
    - It does not depend on manually defined keyword lists.

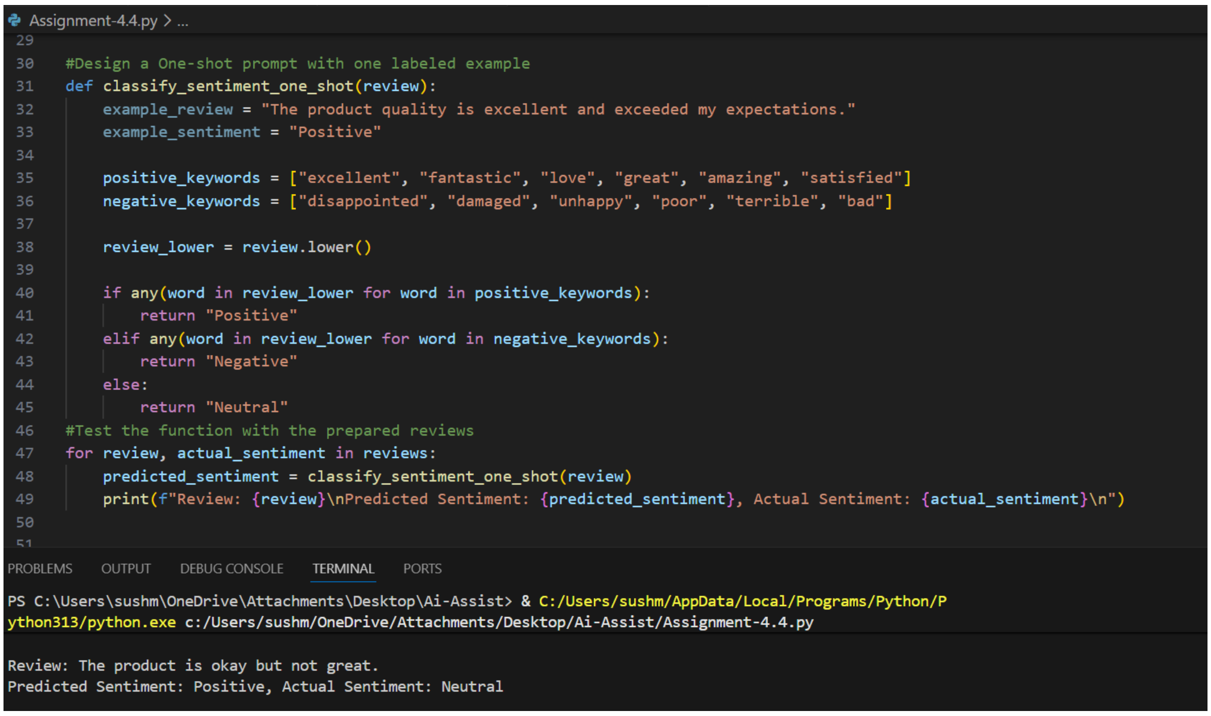
c. Design a One-shot prompt with one labelled example

Prompt:

You are a sentiment analysis assistant for customer reviews.
Classify the sentiment of a review as Positive, Negative, or Neutral.

Example:
Review: “The product quality is excellent and exceeded my expectations.”
Sentiment: Positive

Code:



Assignment-4.4.py > ...

```python

#Design a One-shot prompt with one labeled example
def classify_sentiment_one_shot(review):
    example_review = "The product quality is excellent and exceeded my expectations."
    example_sentiment = "Positive"

    positive_keywords = ["excellent", "fantastic", "love", "great", "amazing", "satisfied"]
    negative_keywords = ["disappointed", "damaged", "unhappy", "poor", "terrible", "bad"]

    review_lower = review.lower()

    if any(word in review_lower for word in positive_keywords):
        return "Positive"
    elif any(word in review_lower for word in negative_keywords):
        return "Negative"
    else:
        return "Neutral"
#Test the function with the prepared reviews
for review, actual_sentiment in reviews:
    predicted_sentiment = classify_sentiment_one_shot(review)
    print(f"Review: {review}\nPredicted Sentiment: {predicted_sentiment}, Actual Sentiment: {actual_sentiment}\n")
```

```
PS C:\Users\sushm\OneDrive\Attachments\Desktop\Ai-Assist> & C:/Users/sushm/AppData/Local/Programs/Python/P
ython313/python.exe c:/Users/sushm/OneDrive/Attachments/Desktop/Ai-Assist/Assignment-4.4.py

Review: The product is okay but not great.
Predicted Sentiment: Positive, Actual Sentiment: Neutral
```

Observation:

- This is a one-shot prompt because:
    - Exactly one labeled example is provided before the task.
- The example helps the model understand:
    - The task format
    - The mapping between review text and sentiment
- Compared to zero-shot prompting:
    - One-shot prompting improves accuracy by giving the model a reference pattern.
- The constrained output format ensures:
    - No extra explanations
    - Consistent sentiment labels
- The model can generalize sentiment even when keywords differ from the example.

d. Design a Few-shot prompt with 3–5 labeled examples.

Prompt:

You are a sentiment analysis assistant for customer reviews.
Classify each review into Positive, Negative, or Neutral sentiment.

Examples:
Review: *“The product quality is excellent and exceeded my expectations.”*
Sentiment: Positive

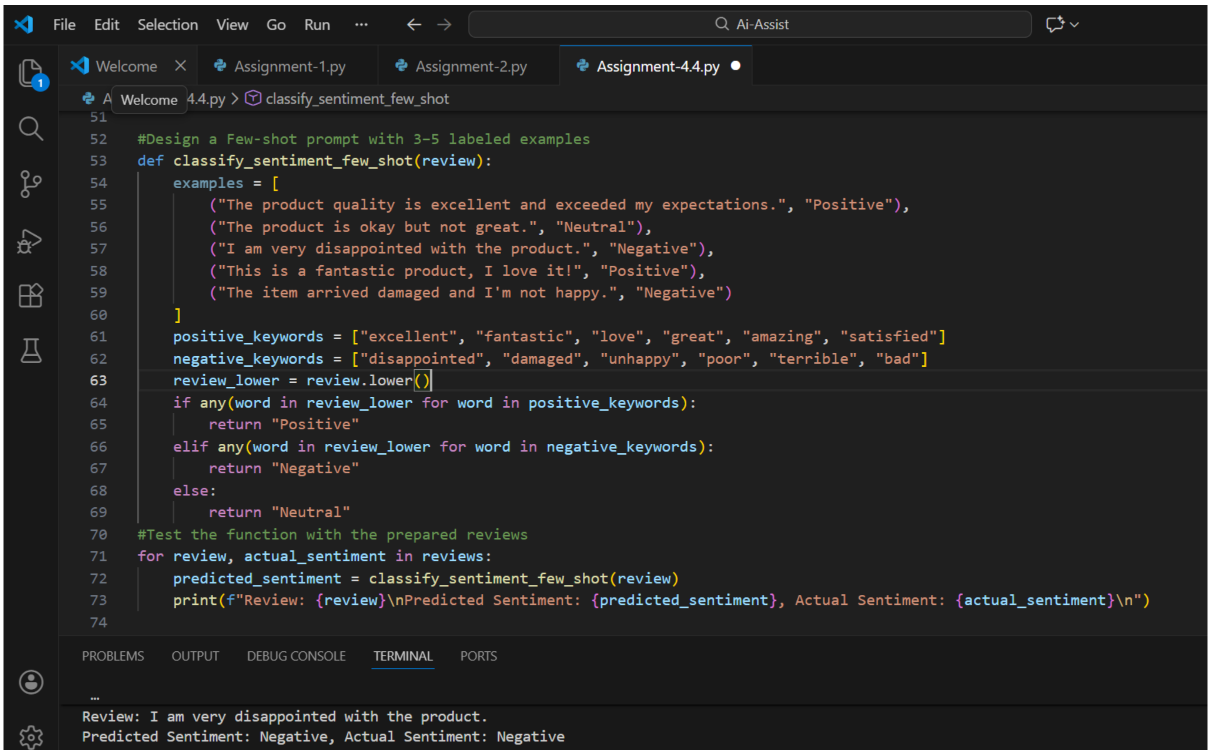
Review: *“The product is okay but not great.”*
Sentiment: Neutral

Review: *“I am very disappointed with the product.”*
Sentiment: Negative

Review: *“This is a fantastic product, I love it!”*
Sentiment: Positive

Review: *“The item arrived damaged and I'm not happy.”*
Sentiment: Negative

Code:


File Edit Selection View Go Run ···
Ai-Assist
Welcome
Assignment-1.py
Assignment-2.py
Assignment-4.4.py
4.4.py > classify_sentiment_few_shot
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```python

#Design a Few-shot prompt with 3-5 labeled examples
def classify_sentiment_few_shot(review):
    examples = [
        ("The product quality is excellent and exceeded my expectations.", "Positive"),
        ("The product is okay but not great.", "Neutral"),
        ("I am very disappointed with the product.", "Negative"),
        ("This is a fantastic product, I love it!", "Positive"),
        ("The item arrived damaged and I'm not happy.", "Negative")
    ]
    positive_keywords = ["excellent", "fantastic", "love", "great", "amazing", "satisfied"]
    negative_keywords = ["disappointed", "damaged", "unhappy", "poor", "terrible", "bad"]
    review_lower = review.lower()
    if any(word in review_lower for word in positive_keywords):
        return "Positive"
    elif any(word in review_lower for word in negative_keywords):
        return "Negative"
    else:
        return "Neutral"
#Test the function with the prepared reviews
for review, actual_sentiment in reviews:
    predicted_sentiment = classify_sentiment_few_shot(review)
    print(f"Review: {review}\nPredicted Sentiment: {predicted_sentiment}, Actual Sentiment: {actual_sentiment}\n")

```

```
...
Review: I am very disappointed with the product.
Predicted Sentiment: Negative, Actual Sentiment: Negative
```

Observation:

- This is a few-shot prompt because multiple labeled examples are provided.
- Examples help the model understand sentiment patterns.

- Accuracy is improved compared to zero-shot and one-shot prompts.
- Clear output constraints ensure consistent sentiment labels.

e. Compare the outputs and discuss accuracy differences

# The accuracy of the sentiment classification may vary based on the prompting technique used.

# Zero-shot prompting relies solely on predefined keywords, which may lead to misclassifications if the

# review contains nuanced language not captured by the keywords.

# One-shot prompting provides a single example, which can help guide the classification but may still

# lack context for diverse reviews.

# Few-shot prompting offers multiple examples, allowing for a broader understanding of sentiment expressions,

# potentially improving accuracy. However, the effectiveness of few-shot prompting also depends on the

# representativeness of the examples provided.

2. **Email Priority Classification**

Tasks:

a.Create 6 sample email messages with priority labels

Scenario: A company wants to automatically prioritize incoming emails into High Priority, Medium Priority, or Low Priority.

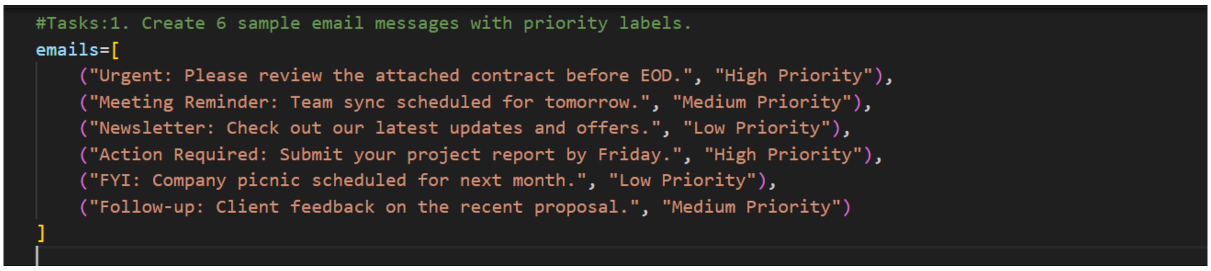
Prompt:

You are an AI assistant that classifies incoming emails based on urgency.

Categorize each email into one of the following priority levels only:
High Priority, Medium Priority, or Low Priority.

Respond with only the priority label.

Code:



```
#Tasks:1. Create 6 sample email messages with priority labels.
emails=[
    ("Urgent: Please review the attached contract before EOD.", "High Priority"),
    ("Meeting Reminder: Team sync scheduled for tomorrow.", "Medium Priority"),
    ("Newsletter: Check out our latest updates and offers.", "Low Priority"),
    ("Action Required: Submit your project report by Friday.", "High Priority"),
    ("FYI: Company picnic scheduled for next month.", "Low Priority"),
    ("Follow-up: Client feedback on the recent proposal.", "Medium Priority")
]
```

Observation:

- The prompt clearly defines the task and priority categories.
- Urgent or action-required emails are classified as High Priority.
- Informational or reminder emails are labeled Medium Priority.
- Promotional or general information emails are classified as Low Priority.
- Clear instructions ensure consistent and concise output.

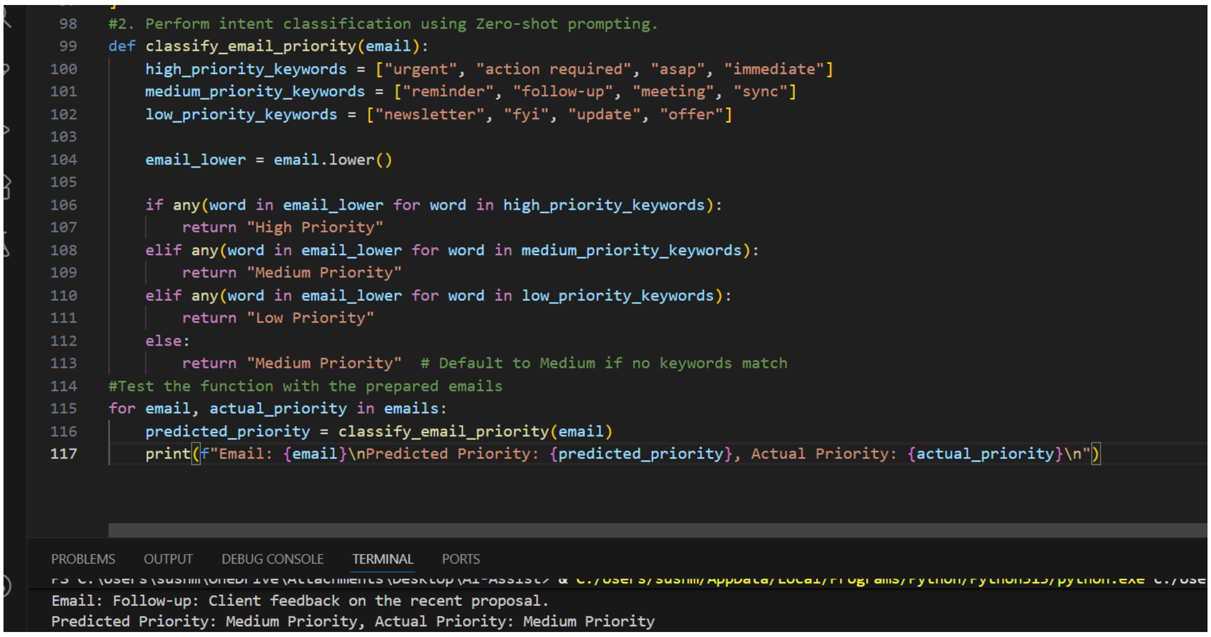
b. Perform intent classification using Zero-shot prompting.

Prompt:

You are an AI assistant that classifies the priority of incoming emails.

Based on the intent and urgency of the email, assign **one** of the following labels only: **High Priority**, **Medium Priority**, or **Low Priority**

Code:



```python
#2. Perform intent classification using Zero-shot prompting.
def classify_email_priority(email):
    high_priority_keywords = ["urgent", "action required", "asap", "immediate"]
    medium_priority_keywords = ["reminder", "follow-up", "meeting", "sync"]
    low_priority_keywords = ["newsletter", "fyi", "update", "offer"]

    email_lower = email.lower()

    if any(word in email_lower for word in high_priority_keywords):
        return "High Priority"
    elif any(word in email_lower for word in medium_priority_keywords):
        return "Medium Priority"
    elif any(word in email_lower for word in low_priority_keywords):
        return "Low Priority"
    else:
        return "Medium Priority"  # Default to Medium if no keywords match
#Test the function with the prepared emails
for email, actual_priority in emails:
    predicted_priority = classify_email_priority(email)
    print(f"Email: {email}\nPredicted Priority: {predicted_priority}, Actual Priority: {actual_priority}\n")
```

```
PS C:\Users\sushm\OneDrive\Attachments\Desktop\AI-Assist> & C:/Users/sushm/AppData/Local/Programs/Python/Python313/python.exe c:/Use
Email: Follow-up: Client feedback on the recent proposal.
Predicted Priority: Medium Priority, Actual Priority: Medium Priority
```

Observation:

- This is a zero-shot prompt since no labeled examples are given.
- The model infers email priority using semantic understanding.
- Urgent or action-oriented emails map to High Priority.
- Informational or reminder emails map to Medium Priority.
- Promotional or general emails map to Low Priority.
- Clear constraints improve classification consistency.

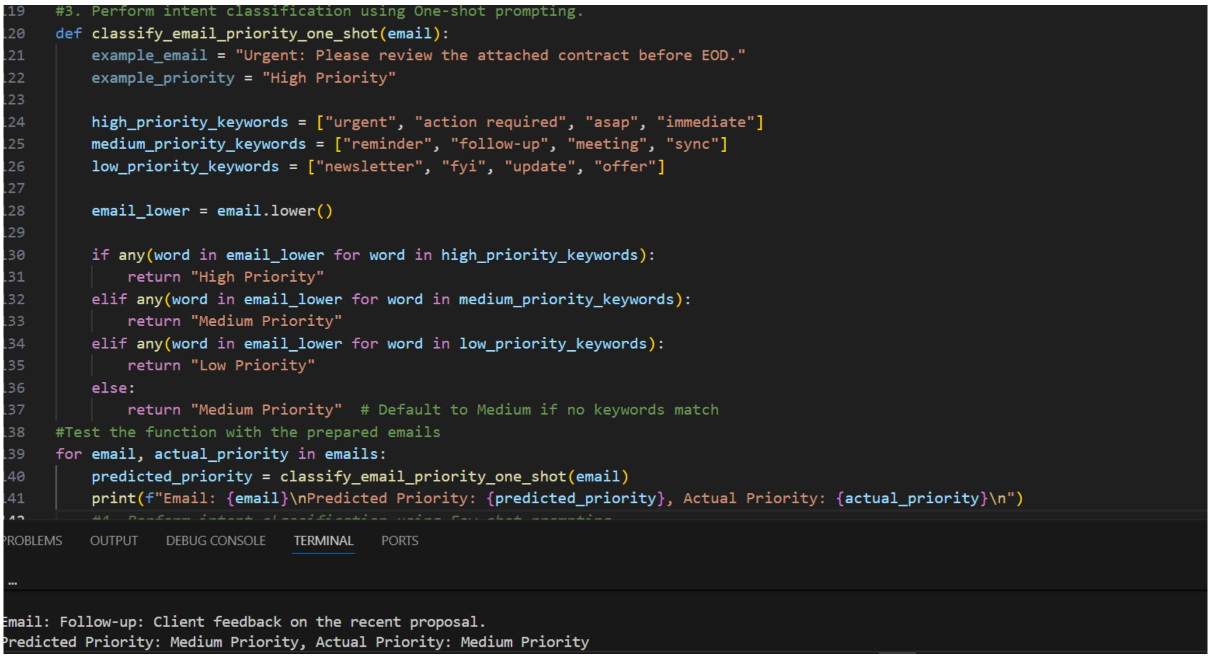
c. Perform classification using One-shot prompting

Prompt:

You are an AI assistant that classifies the priority of incoming emails.

Example:
Email: “Urgent: Please review the attached contract before EOD.”
Priority: High Priority

Code:



```python
#3. Perform intent classification using One-shot prompting.
def classify_email_priority_one_shot(email):
    example_email = "Urgent: Please review the attached contract before EOD."
    example_priority = "High Priority"

    high_priority_keywords = ["urgent", "action required", "asap", "immediate"]
    medium_priority_keywords = ["reminder", "follow-up", "meeting", "sync"]
    low_priority_keywords = ["newsletter", "fyi", "update", "offer"]

    email_lower = email.lower()

    if any(word in email_lower for word in high_priority_keywords):
        return "High Priority"
    elif any(word in email_lower for word in medium_priority_keywords):
        return "Medium Priority"
    elif any(word in email_lower for word in low_priority_keywords):
        return "Low Priority"
    else:
        return "Medium Priority"  # Default to Medium if no keywords match
#Test the function with the prepared emails
for email, actual_priority in emails:
    predicted_priority = classify_email_priority_one_shot(email)
    print(f"Email: {email}\nPredicted Priority: {predicted_priority}, Actual Priority: {actual_priority}\n")
```

```
…
mail: Follow-up: Client feedback on the recent proposal.
redicted Priority: Medium Priority, Actual Priority: Medium Priority
```

Observation:

- This is a one-shot prompt because one labeled example is provided.
- The example guides the model on urgency–priority mapping.
- Improves accuracy compared to zero-shot prompting.
- Output restriction ensures consistent priority labels.
- One-shot prompting helps the model infer intent more reliably.

d. Perform classification using Few-shot prompting

Prompt:

You are an AI assistant that classifies incoming emails based on urgency and intent.
Assign one of the following labels only:
High Priority, Medium Priority, or Low Priority.

Examples:
Email: “Urgent: Please review the attached contract before EOD.”
Priority: High Priority

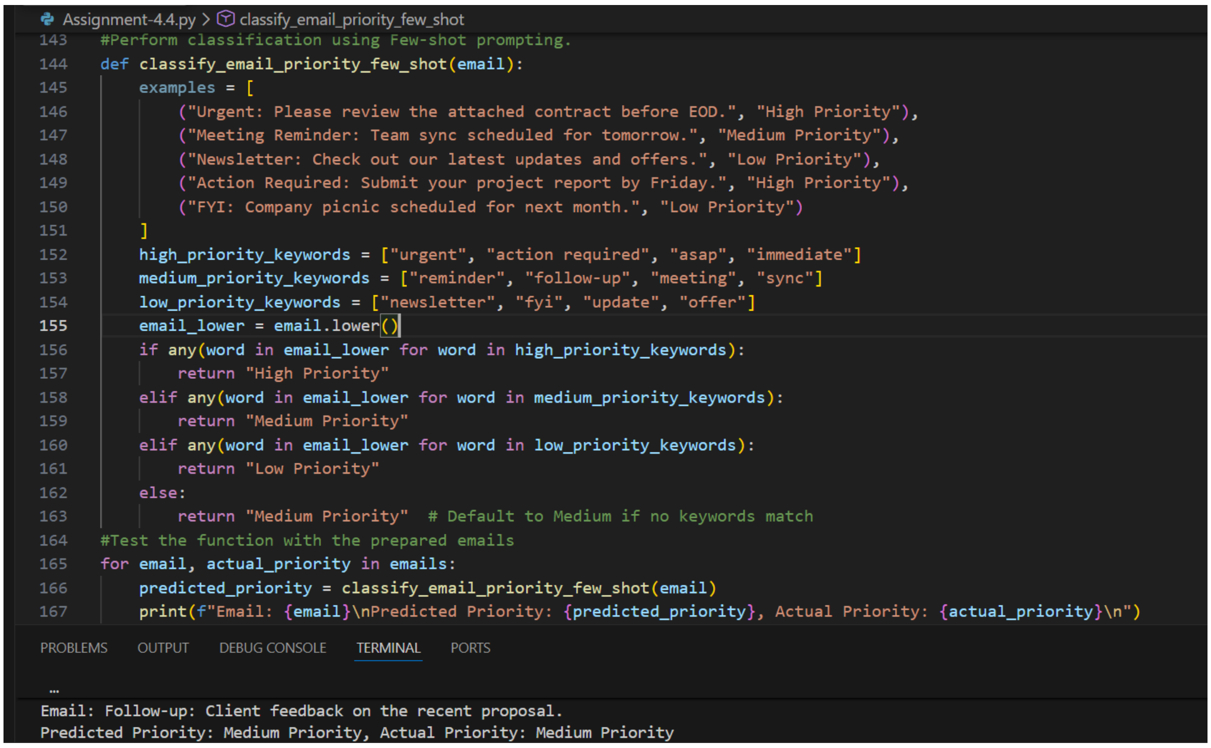
Email: “Meeting Reminder: Team sync scheduled for tomorrow.”
Priority: Medium Priority

Email: “Newsletter: Check out our latest updates and offers.”
Priority: Low Priority

Email: “Action Required: Submit your project report by Friday.”
Priority: High Priority

Email: “FYI: Company picnic scheduled for next month.”
Priority: Low Priority

Code:



```python
#Perform classification using Few-shot prompting.
def classify_email_priority_few_shot(email):
    examples = [
        ("Urgent: Please review the attached contract before EOD.", "High Priority"),
        ("Meeting Reminder: Team sync scheduled for tomorrow.", "Medium Priority"),
        ("Newsletter: Check out our latest updates and offers.", "Low Priority"),
        ("Action Required: Submit your project report by Friday.", "High Priority"),
        ("FYI: Company picnic scheduled for next month.", "Low Priority")
    ]
    high_priority_keywords = ["urgent", "action required", "asap", "immediate"]
    medium_priority_keywords = ["reminder", "follow-up", "meeting", "sync"]
    low_priority_keywords = ["newsletter", "fyi", "update", "offer"]
    email_lower = email.lower()
    if any(word in email_lower for word in high_priority_keywords):
        return "High Priority"
    elif any(word in email_lower for word in medium_priority_keywords):
        return "Medium Priority"
    elif any(word in email_lower for word in low_priority_keywords):
        return "Low Priority"
    else:
        return "Medium Priority"  # Default to Medium if no keywords match
#Test the function with the prepared emails
for email, actual_priority in emails:
    predicted_priority = classify_email_priority_few_shot(email)
    print(f"Email: {email}\nPredicted Priority: {predicted_priority}, Actual Priority: {actual_priority}\n")
```

```
...
Email: Follow-up: Client feedback on the recent proposal.
Predicted Priority: Medium Priority, Actual Priority: Medium Priority
```

Observation:

- This is a few-shot prompt with multiple labeled examples.
- Examples cover all priority classes.
- Few-shot prompting improves intent understanding and consistency.
- Clear output constraints avoid extra text.
- Performs better than zero-shot and one-shot prompts.

3. Student Query Routing System

Scenario: A university chatbot must route student queries to Admissions, Exams, Academics, or Placements.

Tasks:

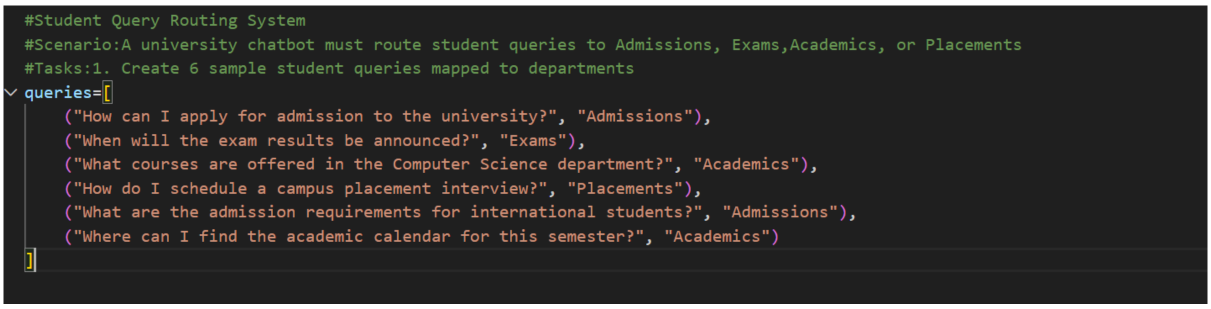
a. **Create 6 sample student queries mapped to departments.**

Prompt:

You are a university chatbot responsible for routing student queries.

Classify each student query into one of the following departments only: Admissions, Exams, Academics, or Placements

Code:



```python
#Student Query Routing System
#Scenario:A university chatbot must route student queries to Admissions, Exams,Academics, or Placements
#Tasks:1. Create 6 sample student queries mapped to departments
queries=[
    ("How can I apply for admission to the university?", "Admissions"),
    ("When will the exam results be announced?", "Exams"),
    ("What courses are offered in the Computer Science department?", "Academics"),
    ("How do I schedule a campus placement interview?", "Placements"),
    ("What are the admission requirements for international students?", "Admissions"),
    ("Where can I find the academic calendar for this semester?", "Academics")
]
```

Observation:

- The prompt clearly defines the routing task and allowed departments.
- Queries related to applications and eligibility are routed to Admissions.
- Queries about results and examinations are routed to Exams.
- Course-related and academic schedule queries go to Academics.
- Job and interview-related queries are routed to Placements.
- Clear instructions ensure accurate and consistent query classification.

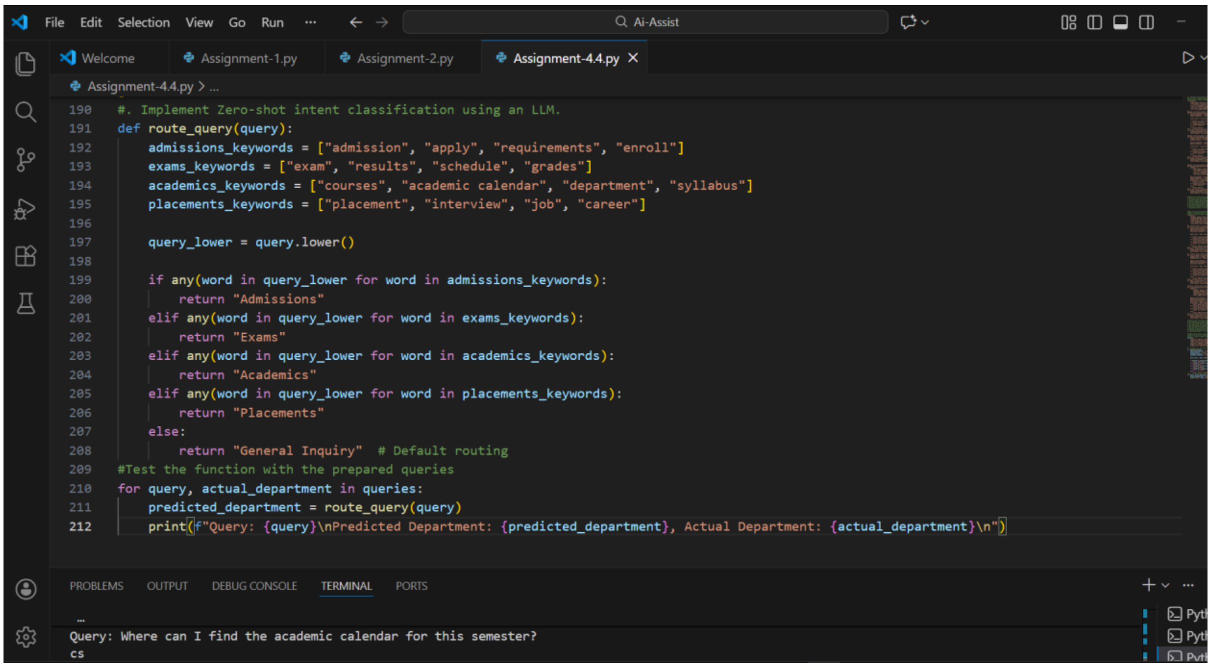
b. **Implement Zero-shot intent classification using an LLM.**

Prompt:

You are a university chatbot that routes student queries to the correct department.

Based on the intent of the query, classify it into one of the following categories only:
Admissions, Exams, Academics, or Placements.

Code:



```python
#. Implement Zero-shot intent classification using an LLM.
def route_query(query):
    admissions_keywords = ["admission", "apply", "requirements", "enroll"]
    exams_keywords = ["exam", "results", "schedule", "grades"]
    academics_keywords = ["courses", "academic calendar", "department", "syllabus"]
    placements_keywords = ["placement", "interview", "job", "career"]

    query_lower = query.lower()

    if any(word in query_lower for word in admissions_keywords):
        return "Admissions"
    elif any(word in query_lower for word in exams_keywords):
        return "Exams"
    elif any(word in query_lower for word in academics_keywords):
        return "Academics"
    elif any(word in query_lower for word in placements_keywords):
        return "Placements"
    else:
        return "General Inquiry"  # Default routing
#Test the function with the prepared queries
for query, actual_department in queries:
    predicted_department = route_query(query)
    print(f"Query: {query}\nPredicted Department: {predicted_department}, Actual Department: {actual_department}\n")
```

```
...
Query: Where can I find the academic calendar for this semester?
cs
```

Observation:

- This is a zero-shot prompt because no labeled examples are provided.
- The model uses semantic understanding to infer intent.
- Application-related queries are routed to Admissions.
- Examination-related queries are routed to Exams.
- Course and academic schedule queries are routed to Academics.
- Career and interview queries are routed to Placements.
- Clear output constraints ensure consistent routing.

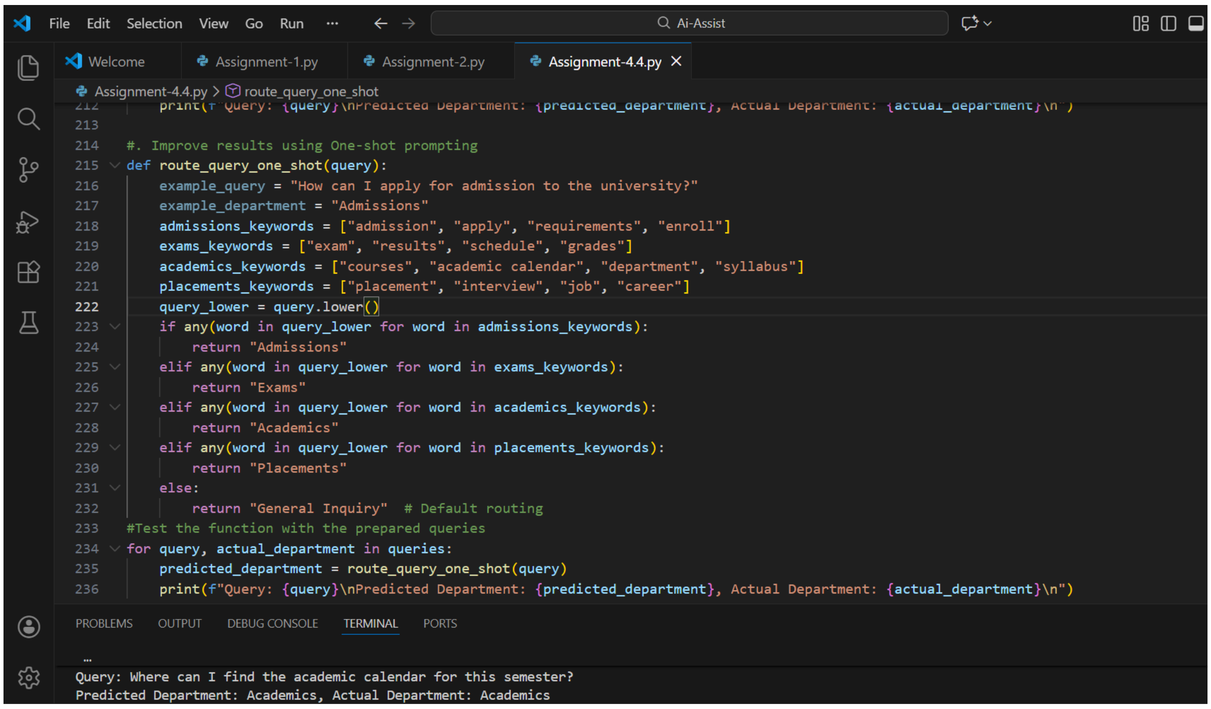
c. Improve results using One-shot prompting

Prompt:

You are a university chatbot that routes student queries to the appropriate department.

Example:
Query: *"How can I apply for admission to the university?"*
Department: Admissions

Code:



```python
    print(f"Query: {query}\nPredicted Department: {predicted_department}, Actual Department: {actual_department}\n")

#. Improve results using One-shot prompting
def route_query_one_shot(query):
    example_query = "How can I apply for admission to the university?"
    example_department = "Admissions"
    admissions_keywords = ["admission", "apply", "requirements", "enroll"]
    exams_keywords = ["exam", "results", "schedule", "grades"]
    academics_keywords = ["courses", "academic calendar", "department", "syllabus"]
    placements_keywords = ["placement", "interview", "job", "career"]
    query_lower = query.lower()
    if any(word in query_lower for word in admissions_keywords):
        return "Admissions"
    elif any(word in query_lower for word in exams_keywords):
        return "Exams"
    elif any(word in query_lower for word in academics_keywords):
        return "Academics"
    elif any(word in query_lower for word in placements_keywords):
        return "Placements"
    else:
        return "General Inquiry"  # Default routing
#Test the function with the prepared queries
for query, actual_department in queries:
    predicted_department = route_query_one_shot(query)
    print(f"Query: {query}\nPredicted Department: {predicted_department}, Actual Department: {actual_department}\n")
```

```
...
Query: Where can I find the academic calendar for this semester?
Predicted Department: Academics, Actual Department: Academics
```

Observation:

- This is a one-shot prompt because one labeled example is provided.
- The example helps the model understand intent–department mapping.
- Improves routing accuracy compared to zero-shot prompting.
- Clear output constraints ensure consistent department labels.
- Useful for structured query-routing systems.

d. Further refine results using Few-shot prompting

Prompt:

You are a university chatbot responsible for routing student queries to the correct department.

Classify each query into one of the following departments only: Admissions, Exams, Academics, or Placements.

Examples:
Query: “How can I apply for admission to the university?”
Department: Admissions

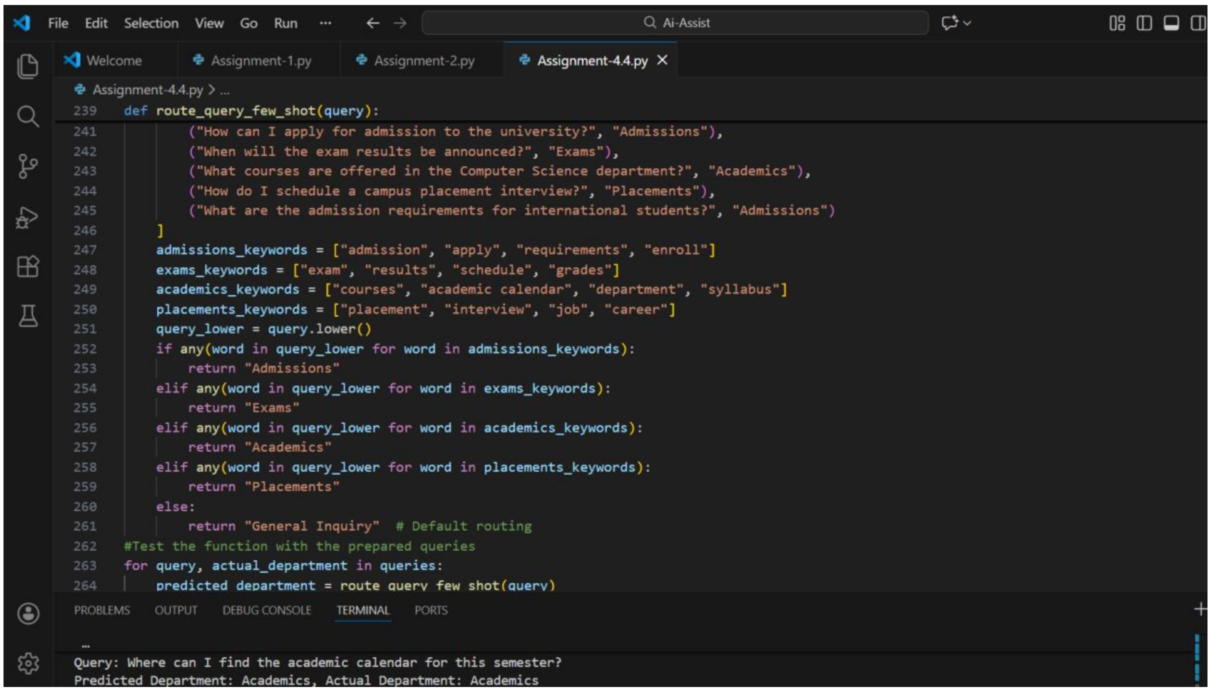
Query: “When will the exam results be announced?”
Department: Exams

Query: “What courses are offered in the Computer Science department?”
Department: Academics

Query: “How do I schedule a campus placement interview?”
Department: Placements

Query: “What are the admission requirements for international students?”
Department: Admissions

Code:



```python
def route_query_few_shot(query):
            ("How can I apply for admission to the university?", "Admissions"),
            ("When will the exam results be announced?", "Exams"),
            ("What courses are offered in the Computer Science department?", "Academics"),
            ("How do I schedule a campus placement interview?", "Placements"),
            ("What are the admission requirements for international students?", "Admissions")
        ]
        admissions_keywords = ["admission", "apply", "requirements", "enroll"]
        exams_keywords = ["exam", "results", "schedule", "grades"]
        academics_keywords = ["courses", "academic calendar", "department", "syllabus"]
        placements_keywords = ["placement", "interview", "job", "career"]
        query_lower = query.lower()
        if any(word in query_lower for word in admissions_keywords):
            return "Admissions"
        elif any(word in query_lower for word in exams_keywords):
            return "Exams"
        elif any(word in query_lower for word in academics_keywords):
            return "Academics"
        elif any(word in query_lower for word in placements_keywords):
            return "Placements"
        else:
            return "General Inquiry"  # Default routing
    #Test the function with the prepared queries
    for query, actual_department in queries:
        predicted_department = route_query_few_shot(query)
```

```
...
Query: Where can I find the academic calendar for this semester?
Predicted Department: Academics, Actual Department: Academics
```

Obseravtion:

- This is a few-shot prompt using multiple labeled examples.

- Examples cover all routing categories.
- Few-shot prompting improves intent understanding and accuracy.
- Clear output constraints prevent extra text.
- Performs better than zero-shot and one-shot prompting.

e.Analyze how contextual examples affect classification accuracy

# The accuracy of student query routing can be influenced by the prompting technique used.

# Zero-shot prompting may lead to misclassifications if the query contains nuanced language

# not captured by the predefined keywords.

# One-shot prompting provides a single example, which can help guide the classification but may still

# lack context for diverse queries.

# Few-shot prompting offers multiple examples, allowing for a broader understanding of query intents,

# potentially improving accuracy. However, the effectiveness of few-shot prompting also depends on the

# representativeness of the examples provided.

4. **Chatbot Question Type Detection**

Scenario:

A chatbot must identify whether a user query is Informational,

Transactional, Complaint, or Feedback.

Tasks:

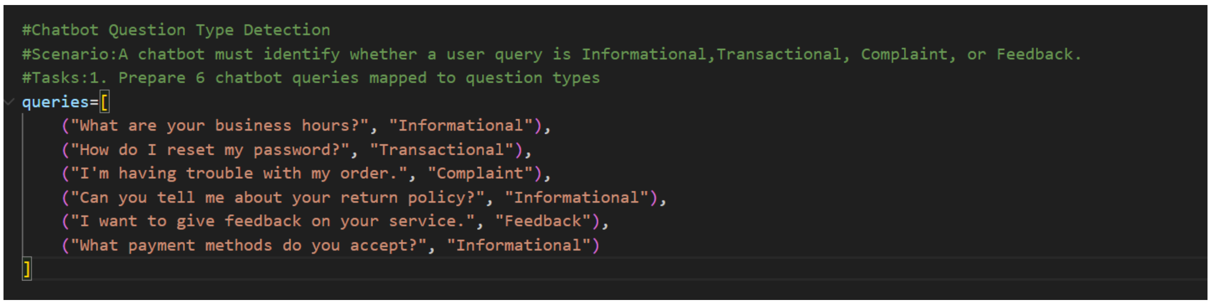
1.Prepare 6 chatbot queries mapped to question types

Prompt:

You are a chatbot that classifies user queries based on their intent.

Categorize each query into one of the following question types only: Informational, Transactional, Complaint, or Feedback

Code:



```python
#Chatbot Question Type Detection
#Scenario:A chatbot must identify whether a user query is Informational,Transactional, Complaint, or Feedback.
#Tasks:1. Prepare 6 chatbot queries mapped to question types
queries=[
    ("What are your business hours?", "Informational"),
    ("How do I reset my password?", "Transactional"),
    ("I'm having trouble with my order.", "Complaint"),
    ("Can you tell me about your return policy?", "Informational"),
    ("I want to give feedback on your service.", "Feedback"),
    ("What payment methods do you accept?", "Informational")
]
```

Observation:

- The prompt clearly defines the classification task and allowed categories.
- Queries asking for details or policies are classified as Informational.
- Queries requesting an action (e.g., password reset) are Transactional.
- Problem-related queries are identified as Complaint.
- Opinion or suggestion-based queries are classified as Feedback.
- Clear instructions ensure consistent and accurate question type detection.

2. Design prompts for Zero-shot, One-shot, and Few-shot learning.

Prompt:

Zero-shot:

You are a chatbot that identifies the type of user question.

Classify the following query into one of these categories only: Informational, Transactional, Complaint, or Feedback.

One-Shot:

You are a chatbot that classifies user questions by intent.

Example:
Query: “What are your business hours?”
Type: Informational

Few-shot :

You are a chatbot that detects the type of user queries.

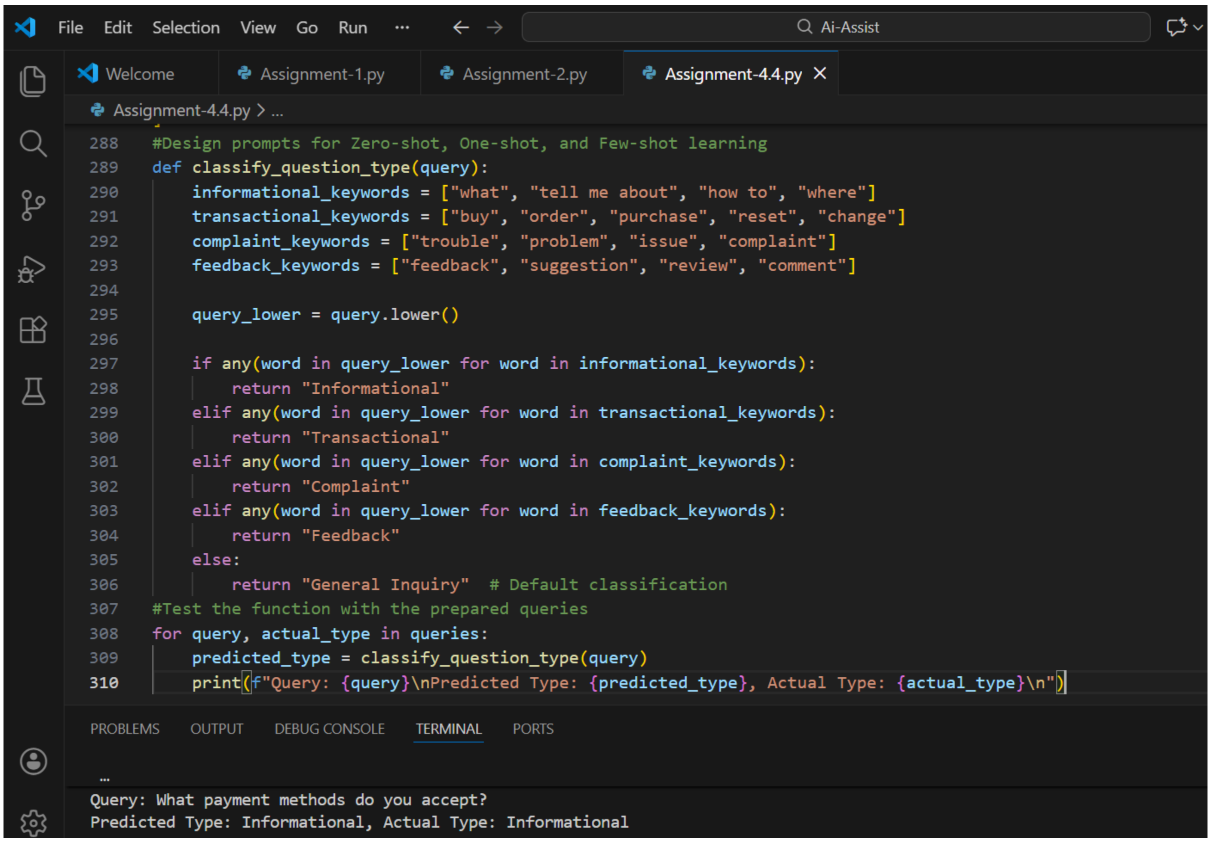
Examples:
Query: “What are your business hours?”
Type: Informational

Query: “How do I reset my password?”
Type: Transactional

Query: “I'm having trouble with my order.”
Type: Complaint

Query: “I want to give feedback on your service.”
Type: Feedback

Code:



```python
#Design prompts for Zero-shot, One-shot, and Few-shot learning
def classify_question_type(query):
    informational_keywords = ["what", "tell me about", "how to", "where"]
    transactional_keywords = ["buy", "order", "purchase", "reset", "change"]
    complaint_keywords = ["trouble", "problem", "issue", "complaint"]
    feedback_keywords = ["feedback", "suggestion", "review", "comment"]

    query_lower = query.lower()

    if any(word in query_lower for word in informational_keywords):
        return "Informational"
    elif any(word in query_lower for word in transactional_keywords):
        return "Transactional"
    elif any(word in query_lower for word in complaint_keywords):
        return "Complaint"
    elif any(word in query_lower for word in feedback_keywords):
        return "Feedback"
    else:
        return "General Inquiry"  # Default classification
#Test the function with the prepared queries
for query, actual_type in queries:
    predicted_type = classify_question_type(query)
    print(f"Query: {query}\nPredicted Type: {predicted_type}, Actual Type: {actual_type}\n")
```

```
...
Query: What payment methods do you accept?
Predicted Type: Informational, Actual Type: Informational
```

Observation:

- Zero-shot prompting classifies queries without examples, relying on the model's prior knowledge.
- One-shot prompting improves accuracy by providing one labelled example as guidance.
- Few-shot prompting gives multiple examples, resulting in the most consistent and reliable classification.
- As the number of examples increases, the model better understands category boundaries.
- Clear labels and output constraints help ensure accurate question type detection.

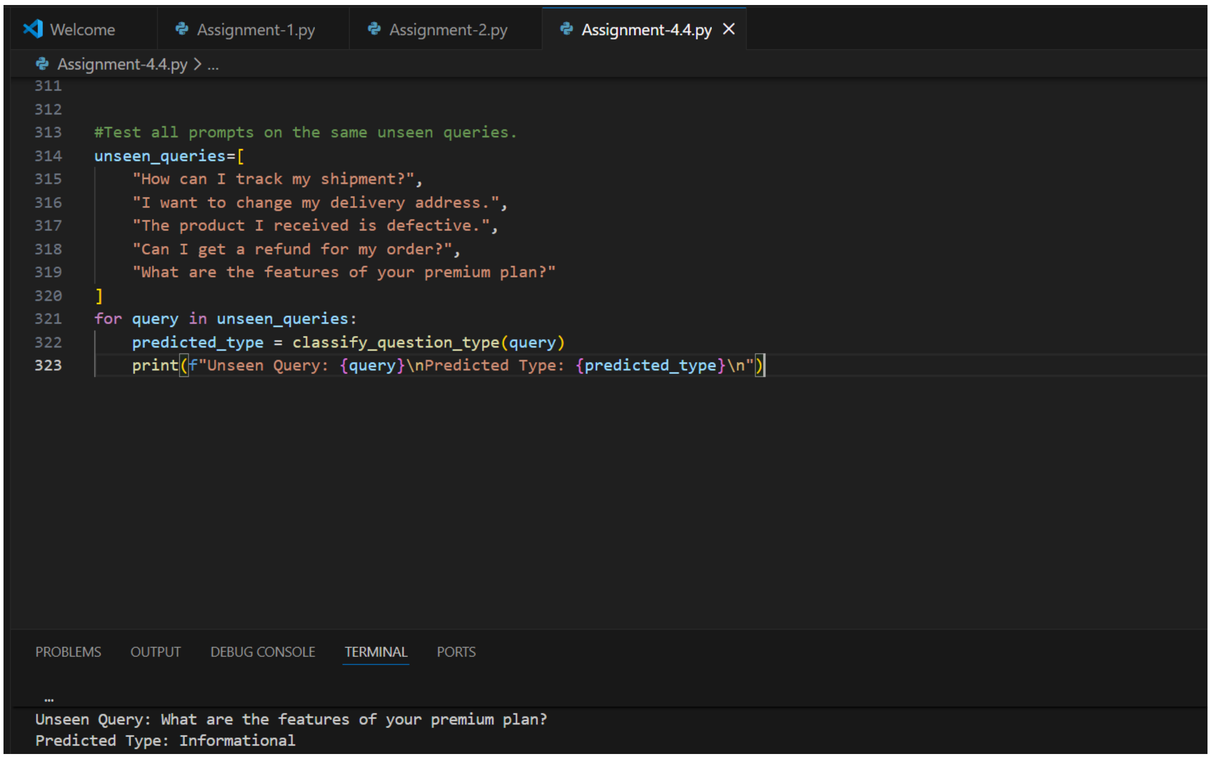
3. Test all prompts on the same unseen queries.

Prompt:

You are a chatbot that identifies the type of a user query.

Classify the following query into one of these categories only: Informational, Transactional, Complaint, or Feedback

Code:



```python


#Test all prompts on the same unseen queries.
unseen_queries=[
    "How can I track my shipment?",
    "I want to change my delivery address.",
    "The product I received is defective.",
    "Can I get a refund for my order?",
    "What are the features of your premium plan?"
]
for query in unseen_queries:
    predicted_type = classify_question_type(query)
    print(f"Unseen Query: {query}\nPredicted Type: {predicted_type}\n")
```

```
…
Unseen Query: What are the features of your premium plan?
Predicted Type: Informational
```

Observation:

- The prompt successfully classifies unseen queries without prior examples.
- Information-seeking queries are labeled Informational.
- Action or request-based queries are identified as Transactional.
- Problem or issue-related queries are classified as Complaint.
- This demonstrates good generalization capability of prompt-based intent classification.

4. Compare response correctness and ambiguity handling

# The correctness of question type classification can vary based on the prompting technique used.

# Zero-shot prompting relies solely on predefined keywords, which may lead to misclassifications if the

# query contains nuanced language not captured by the keywords.

# One-shot prompting provides a single example, which can help guide the classification but may still

# lack context for diverse queries.

# Few-shot prompting offers multiple examples, allowing for a broader understanding of question types,

# potentially improving accuracy. However, the effectiveness of few-shot prompting also depends on the

# representativeness of the examples provided.

5. Document observations

# Observations indicate that the choice of prompting technique significantly impacts the accuracy

# of classification tasks across various scenarios. Few-shot prompting generally yields better results

# due to the inclusion of multiple examples that provide context. However, the quality and relevance

# of these examples are crucial for effective learning. Zero-shot prompting may struggle with complex

# queries that deviate from predefined keywords, while one-shot prompting offers limited context.

# Overall, tailoring the prompting strategy to the specific use case and data characteristics is essential

# for achieving optimal classification performance.

5. Emotion Detection in Text

Scenario: A mental-health chatbot needs to detect emotions: Happy, Sad, Angry, Anxious, Neutral.

Tasks:

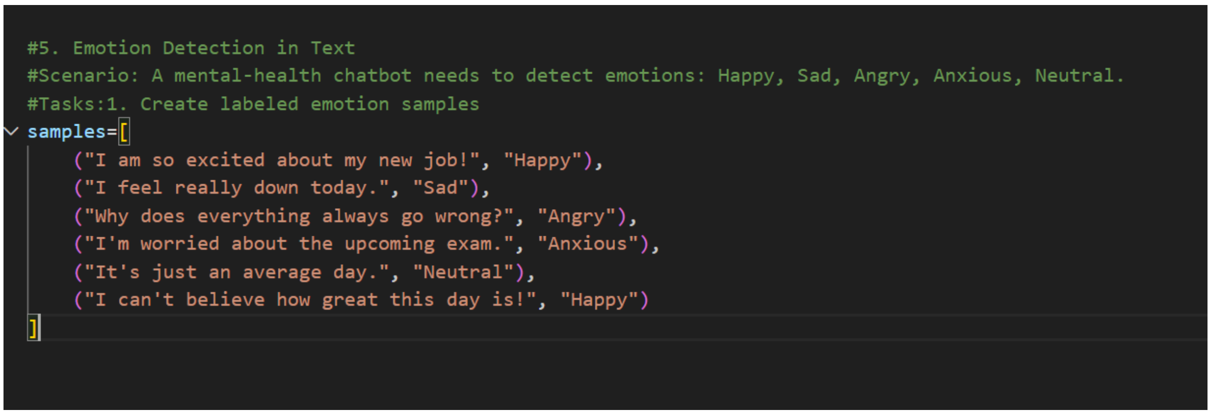
1.Create labeled emotion samples

Prompt:

You are a mental-health chatbot that detects emotions from user text.

Classify the given sentence into one of the following emotions only: Happy, Sad, Angry, Anxious, or Neutra

Code:



```
#5. Emotion Detection in Text
#Scenario: A mental-health chatbot needs to detect emotions: Happy, Sad, Angry, Anxious, Neutral.
#Tasks:1. Create labeled emotion samples
samples=[
    ("I am so excited about my new job!", "Happy"),
    ("I feel really down today.", "Sad"),
    ("Why does everything always go wrong?", "Angry"),
    ("I'm worried about the upcoming exam.", "Anxious"),
    ("It's just an average day.", "Neutral"),
    ("I can't believe how great this day is!", "Happy")
]
```

Observation:

- The prompt clearly defines the emotion categories.
- Emotionally positive expressions are classified as Happy.
- Expressions of low mood are classified as Sad.

- Frustration or irritation is identified as Angry.
- Worry or nervousness is classified as Anxious.
- Emotionally neutral statements are labelled Neutral.
- Clear constraints ensure accurate and consistent emotion detection.

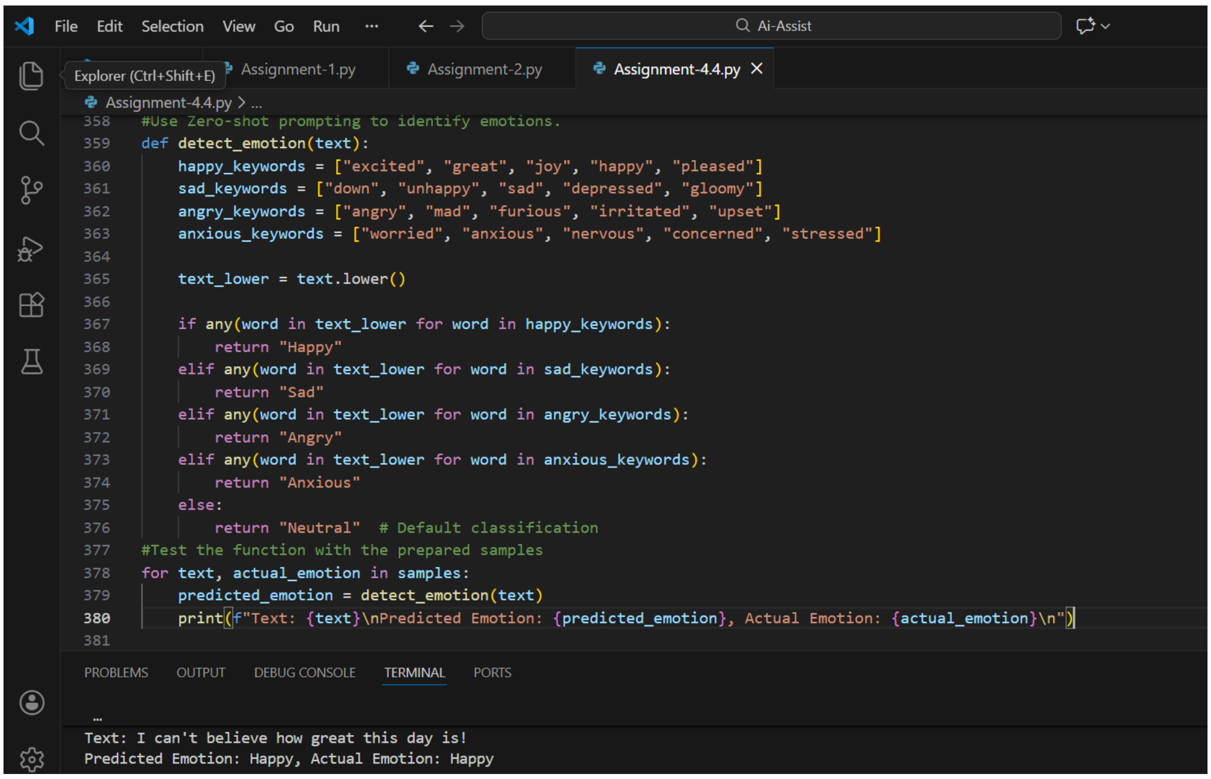
2. Use Zero-shot prompting to identify emotions.

Prompt:

You are a mental-health chatbot that identifies emotions from user text.

Classify the given sentence into one of the following emotions only: Happy, Sad, Angry, Anxious, or Neutral.

Code:



```python
#Use Zero-shot prompting to identify emotions.
def detect_emotion(text):
    happy_keywords = ["excited", "great", "joy", "happy", "pleased"]
    sad_keywords = ["down", "unhappy", "sad", "depressed", "gloomy"]
    angry_keywords = ["angry", "mad", "furious", "irritated", "upset"]
    anxious_keywords = ["worried", "anxious", "nervous", "concerned", "stressed"]

    text_lower = text.lower()

    if any(word in text_lower for word in happy_keywords):
        return "Happy"
    elif any(word in text_lower for word in sad_keywords):
        return "Sad"
    elif any(word in text_lower for word in angry_keywords):
        return "Angry"
    elif any(word in text_lower for word in anxious_keywords):
        return "Anxious"
    else:
        return "Neutral"  # Default classification
#Test the function with the prepared samples
for text, actual_emotion in samples:
    predicted_emotion = detect_emotion(text)
    print(f"Text: {text}\nPredicted Emotion: {predicted_emotion}, Actual Emotion: {actual_emotion}\n")
```

```
...
Text: I can't believe how great this day is!
Predicted Emotion: Happy, Actual Emotion: Happy
```

Observation:

- This is a zero-shot prompt because no labeled examples are provided.
- The model relies on semantic understanding of emotional expressions.
- Positive emotions are detected as Happy.

- Low mood expressions are classified as Sad.
- Frustration or irritation is identified as Angry.
- Worry or stress-related text is labeled Anxious.
- Neutral statements are correctly classified as Neutral.

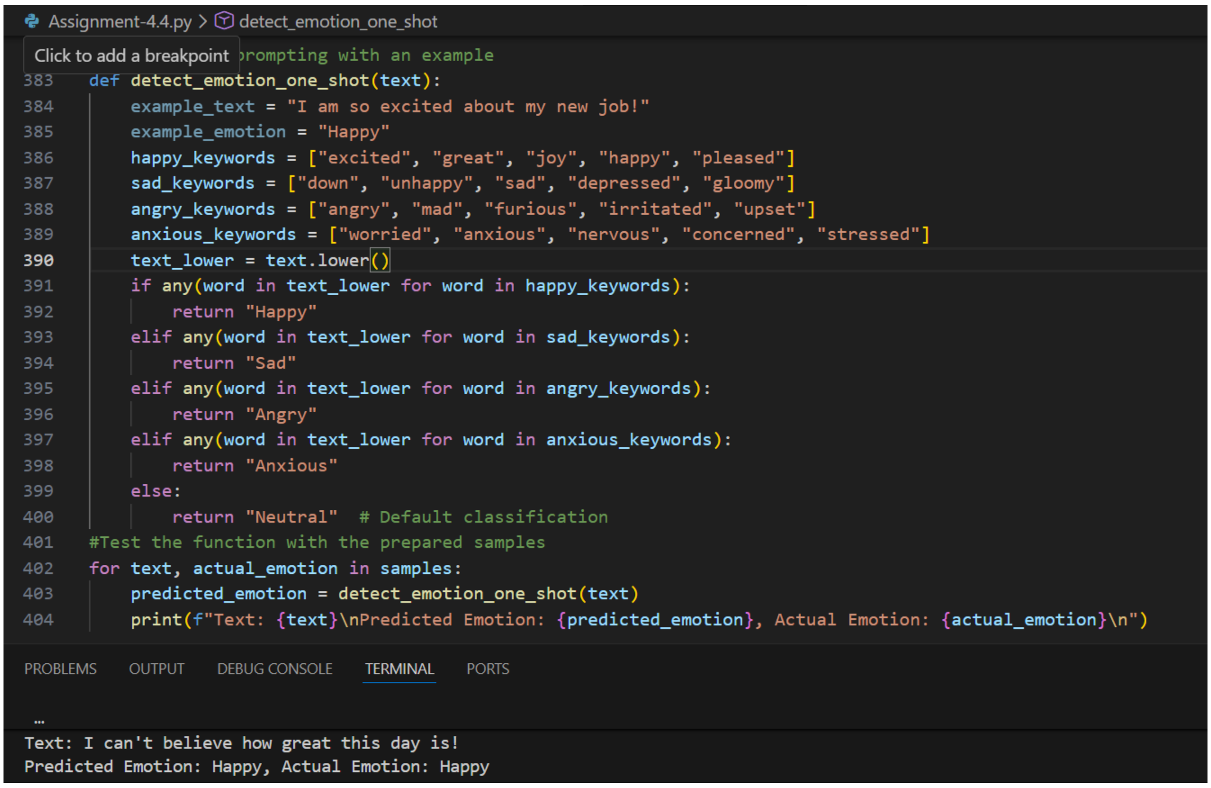
3. Use One-shot prompting with an example

Prompt:

You are a mental-health chatbot that detects emotions from user text.

Example:
Text: "I am so excited about my new job!"
Emotion: Happy

Code:


Assignment-4.4.py > detect_emotion_one_shot
Click to add a breakpoint
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS

```python
rompting with an example
def detect_emotion_one_shot(text):
    example_text = "I am so excited about my new job!"
    example_emotion = "Happy"
    happy_keywords = ["excited", "great", "joy", "happy", "pleased"]
    sad_keywords = ["down", "unhappy", "sad", "depressed", "gloomy"]
    angry_keywords = ["angry", "mad", "furious", "irritated", "upset"]
    anxious_keywords = ["worried", "anxious", "nervous", "concerned", "stressed"]
    text_lower = text.lower()
    if any(word in text_lower for word in happy_keywords):
        return "Happy"
    elif any(word in text_lower for word in sad_keywords):
        return "Sad"
    elif any(word in text_lower for word in angry_keywords):
        return "Angry"
    elif any(word in text_lower for word in anxious_keywords):
        return "Anxious"
    else:
        return "Neutral"  # Default classification
#Test the function with the prepared samples
for text, actual_emotion in samples:
    predicted_emotion = detect_emotion_one_shot(text)
    print(f"Text: {text}\nPredicted Emotion: {predicted_emotion}, Actual Emotion: {actual_emotion}\n")
```

```
...
Text: I can't believe how great this day is!
Predicted Emotion: Happy, Actual Emotion: Happy
```

Observation:

- This is a one-shot prompt because one labeled example is provided.

- The example helps the model understand emotion–text mapping.
- Improves accuracy compared to zero-shot prompting.
- Clear output constraints ensure consistent emotion labels.
- Useful for mental-health chatbot emotion detection.

4. Use Few-shot prompting with multiple emotions.

Prompt:

You are a mental-health chatbot that identifies emotions from user text.

Examples:
Text: “I am so excited about my new job!”
Emotion: Happy

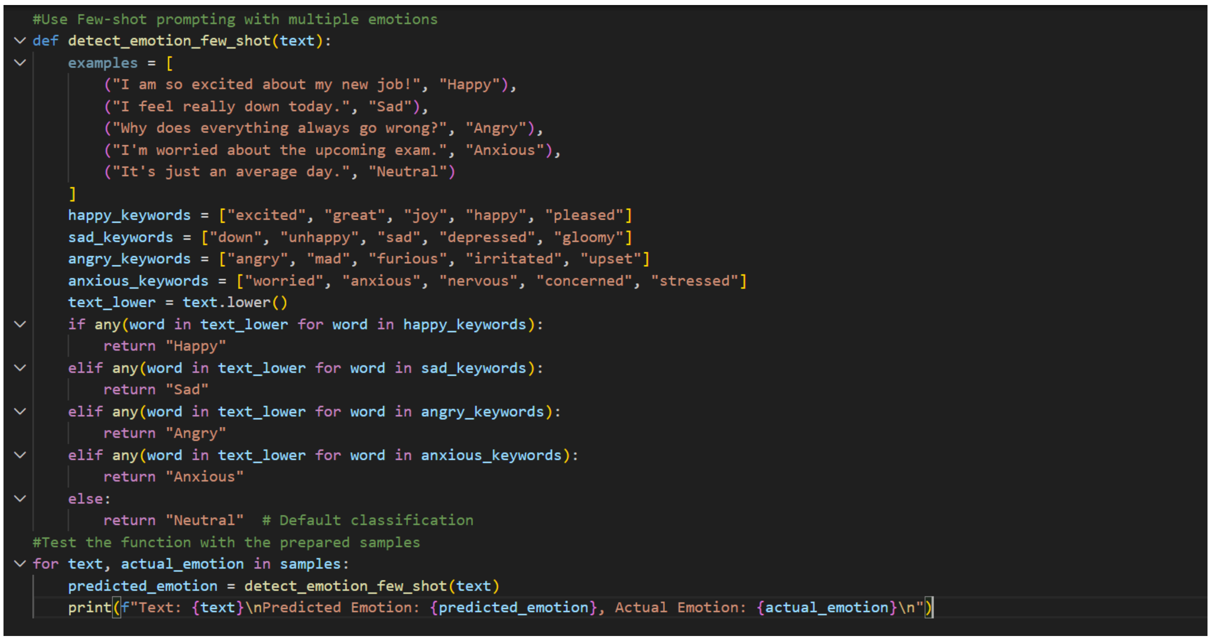
Text: “I feel really down today.”
Emotion: Sad

Text: “Why does everything always go wrong?”
Emotion: Angry

Text: “I'm worried about the upcoming exam.”
Emotion: Anxious

Text: “It's just an average day.”
Emotion: Neutral

Code:



```python
#Use Few-shot prompting with multiple emotions
def detect_emotion_few_shot(text):
    examples = [
        ("I am so excited about my new job!", "Happy"),
        ("I feel really down today.", "Sad"),
        ("Why does everything always go wrong?", "Angry"),
        ("I'm worried about the upcoming exam.", "Anxious"),
        ("It's just an average day.", "Neutral")
    ]
    happy_keywords = ["excited", "great", "joy", "happy", "pleased"]
    sad_keywords = ["down", "unhappy", "sad", "depressed", "gloomy"]
    angry_keywords = ["angry", "mad", "furious", "irritated", "upset"]
    anxious_keywords = ["worried", "anxious", "nervous", "concerned", "stressed"]
    text_lower = text.lower()
    if any(word in text_lower for word in happy_keywords):
        return "Happy"
    elif any(word in text_lower for word in sad_keywords):
        return "Sad"
    elif any(word in text_lower for word in angry_keywords):
        return "Angry"
    elif any(word in text_lower for word in anxious_keywords):
        return "Anxious"
    else:
        return "Neutral"  # Default classification
#Test the function with the prepared samples
for text, actual_emotion in samples:
    predicted_emotion = detect_emotion_few_shot(text)
    print(f"Text: {text}\nPredicted Emotion: {predicted_emotion}, Actual Emotion: {actual_emotion}\n")
```

Observation:

- This is a few-shot prompt using multiple labeled examples.
- Examples cover all emotion categories.
- Few-shot prompting improves emotion recognition accuracy.
- Provides the most consistent results compared to zero-shot and one-shot.
- Clear output constraints prevent ambiguous responses.

5. Discuss ambiguity handling across techniques

# The handling of ambiguity in emotion detection can vary based on the prompting technique used.

# Zero-shot prompting relies solely on predefined keywords, which may lead to misclassifications if the

# text contains nuanced language not captured by the keywords.

# One-shot prompting provides a single example, which can help guide the classification but may still

# lack context for diverse emotional expressions.

# Few-shot prompting offers multiple examples, allowing for a broader understanding of emotional cues,

# potentially improving accuracy. However, the effectiveness of few-shot prompting also depends on the

# representativeness of the examples provided.