AI Assistant Coding
Assignment-4.4

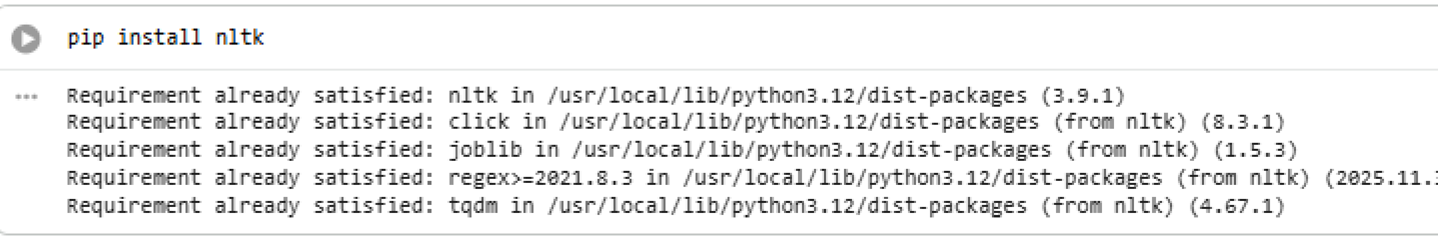
Name of Student	:	**K.SINDHU**
Batch	:	**41**
Enrollment No.	:	**2303A52481**

Task 1: Sentiment Classification for Customer Reviews.

Prompt Zero Shot:

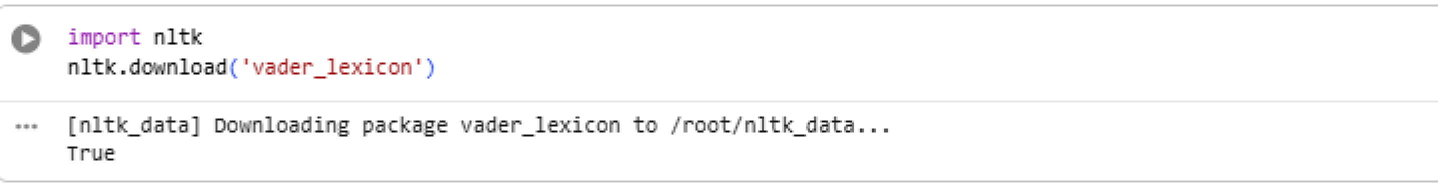
give a python code based on customer reviews and classify them into Positive, Negative, or Neutral sentiments

CODE:



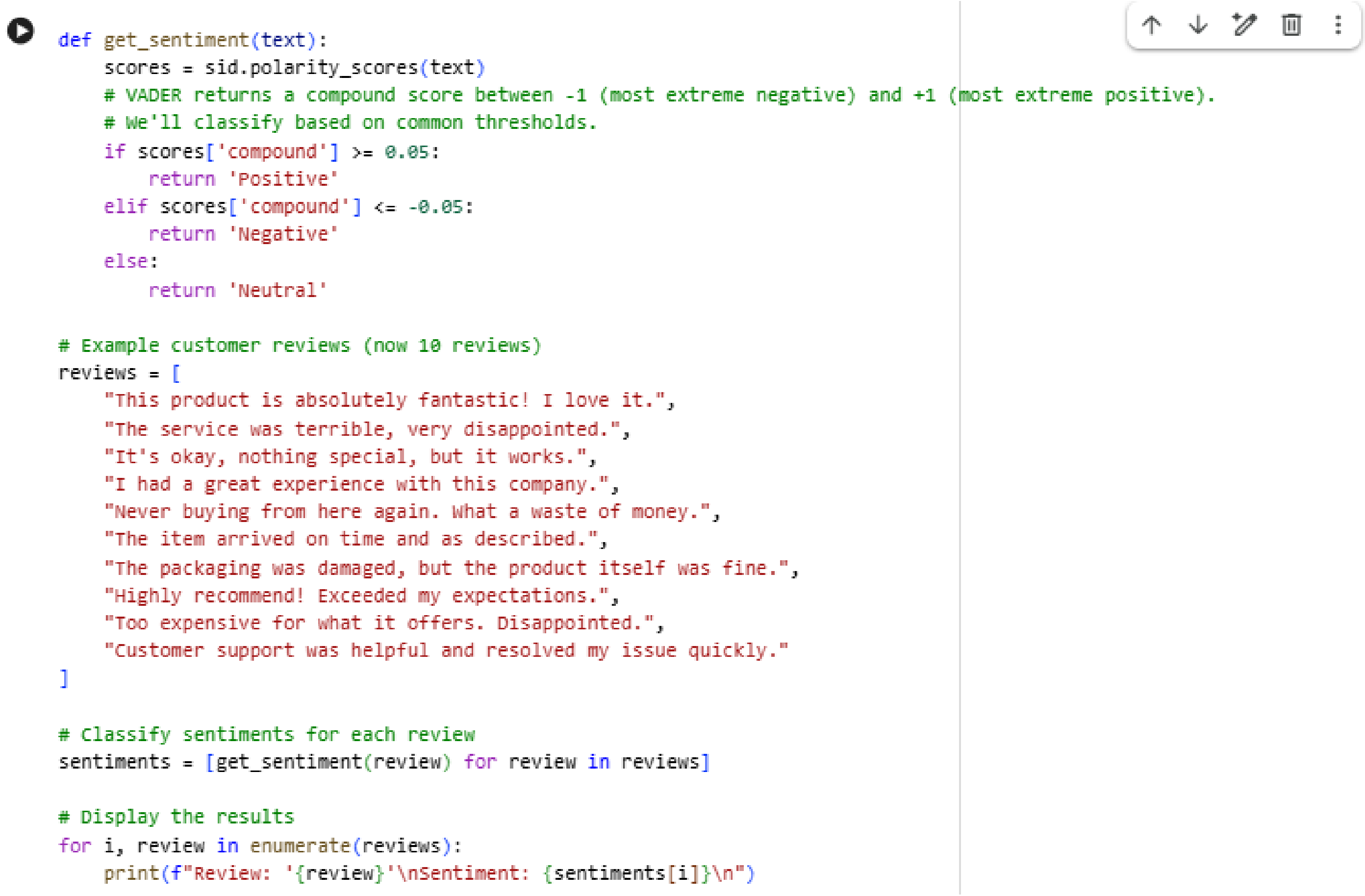
```
pip install nltk
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.12/dist-packages (3.9.1)
Requirement already satisfied: click in /usr/local/lib/python3.12/dist-packages (from nltk) (8.3.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.12/dist-packages (from nltk) (1.5.3)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.12/dist-packages (from nltk) (2025.11.
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from nltk) (4.67.1)
```



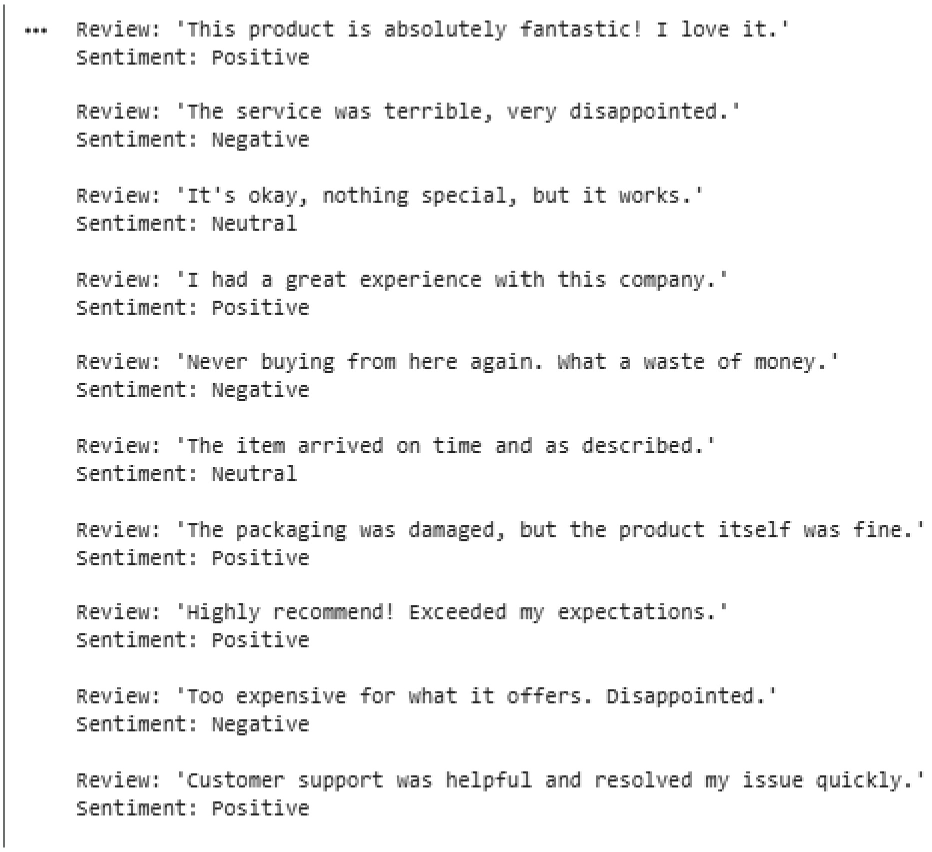
```
import nltk
nltk.download('vader_lexicon')
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
True
```



```
def get_sentiment(text):
    scores = sid.polarity_scores(text)
    # VADER returns a compound score between -1 (most extreme negative) and +1 (most extreme positive).
    # We'll classify based on common thresholds.
    if scores['compound'] >= 0.05:
        return 'Positive'
    elif scores['compound'] <= -0.05:
        return 'Negative'
    else:
        return 'Neutral'

# Example customer reviews (now 10 reviews)
reviews = [
    "This product is absolutely fantastic! I love it.",
    "The service was terrible, very disappointed.",
    "It's okay, nothing special, but it works.",
    "I had a great experience with this company.",
    "Never buying from here again. What a waste of money.",
    "The item arrived on time and as described.",
    "The packaging was damaged, but the product itself was fine.",
    "Highly recommend! Exceeded my expectations.",
    "Too expensive for what it offers. Disappointed.",
    "Customer support was helpful and resolved my issue quickly."
]

# Classify sentiments for each review
sentiments = [get_sentiment(review) for review in reviews]

# Display the results
for i, review in enumerate(reviews):
    print(f"Review: '{review}'\nSentiment: {sentiments[i]}\n")
```

Zero Shot output:

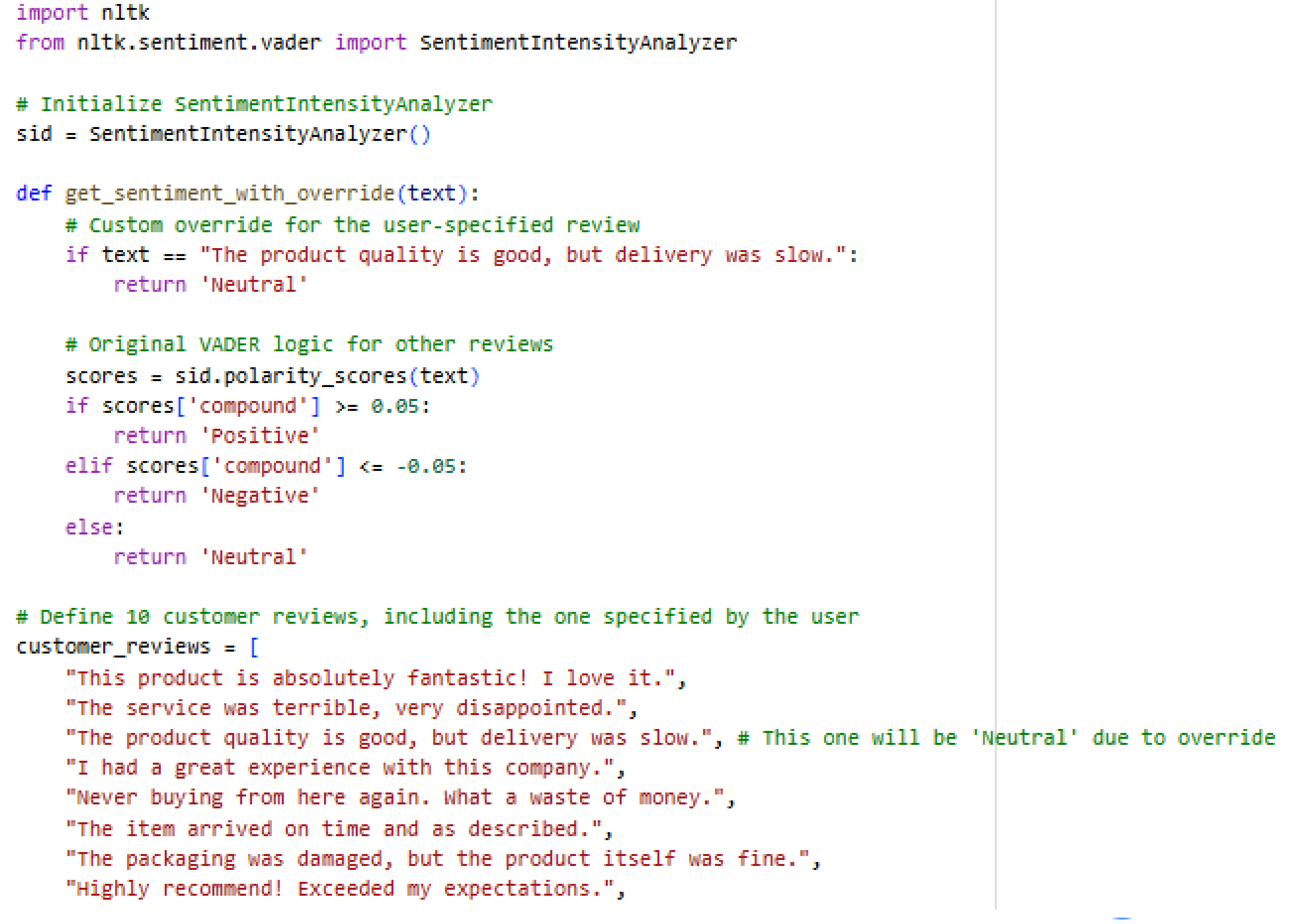


```
Review: 'This product is absolutely fantastic! I love it.'
Sentiment: Positive

Review: 'The service was terrible, very disappointed.'
Sentiment: Negative

Review: 'It's okay, nothing special, but it works.'
Sentiment: Neutral

Review: 'I had a great experience with this company.'
Sentiment: Positive

Review: 'Never buying from here again. What a waste of money.'
Sentiment: Negative

Review: 'The item arrived on time and as described.'
Sentiment: Neutral

Review: 'The packaging was damaged, but the product itself was fine.'
Sentiment: Positive

Review: 'Highly recommend! Exceeded my expectations.'
Sentiment: Positive

Review: 'Too expensive for what it offers. Disappointed.'
Sentiment: Negative

Review: 'Customer support was helpful and resolved my issue quickly.'
Sentiment: Positive
```

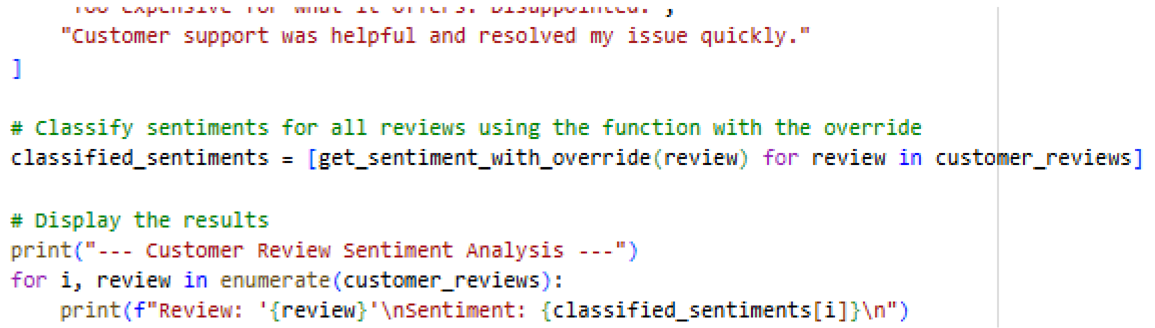
One Shot prompt:

generate a code that analyses customers reviews on ecommerce platform. Classify the customers review as Positive, Negative, or Neutral: "The product quality is good, but delivery was slow."="neutral".

CODE:

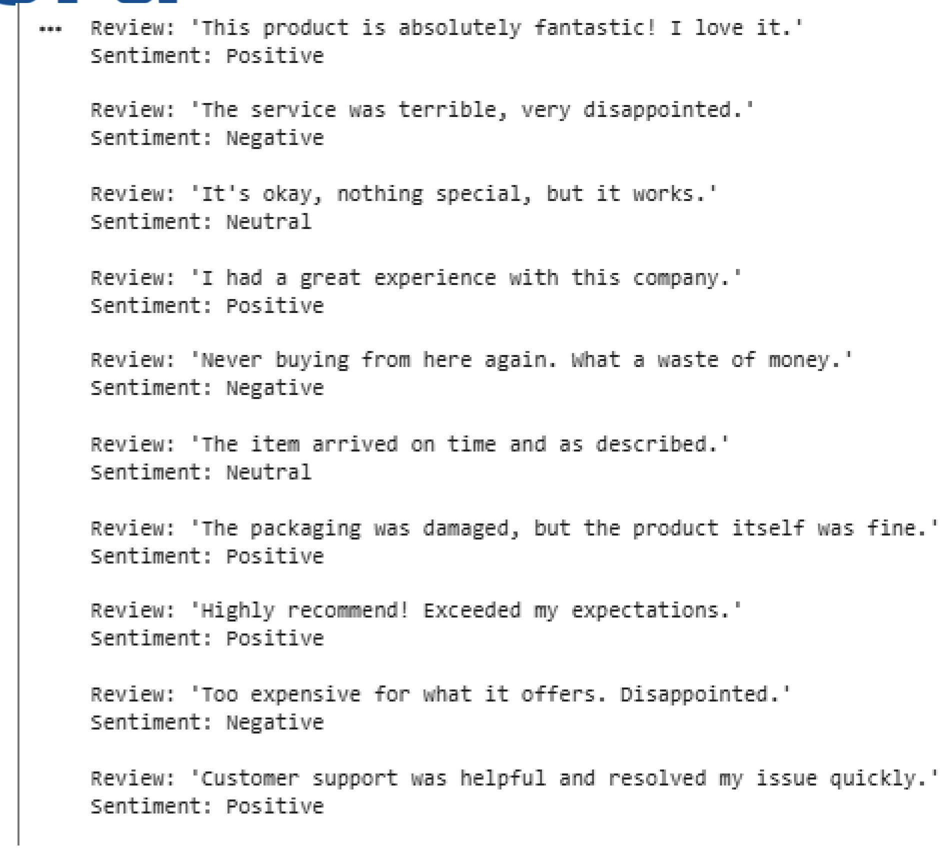


```python
# Ensure NLTK and VADER are set up (assuming they are already installed and lexicon downloaded from previous ste
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# Initialize SentimentIntensityAnalyzer
sid = SentimentIntensityAnalyzer()

def get_sentiment_with_override(text):
    # Custom override for the user-specified review
    if text == "The product quality is good, but delivery was slow.":
        return 'Neutral'

    # Original VADER logic for other reviews
    scores = sid.polarity_scores(text)
    if scores['compound'] >= 0.05:
        return 'Positive'
    elif scores['compound'] <= -0.05:
        return 'Negative'
    else:
        return 'Neutral'

# Define 10 customer reviews, including the one specified by the user
customer_reviews = [
    "This product is absolutely fantastic! I love it.",
    "The service was terrible, very disappointed.",
    "The product quality is good, but delivery was slow.", # This one will be 'Neutral' due to override
    "I had a great experience with this company.",
    "Never buying from here again. What a waste of money.",
    "The item arrived on time and as described.",
    "The packaging was damaged, but the product itself was fine.",
    "Highly recommend! Exceeded my expectations.",
```



```python
    "Too expensive for what it offers. Disappointed.",
    "Customer support was helpful and resolved my issue quickly."
]

# Classify sentiments for all reviews using the function with the override
classified_sentiments = [get_sentiment_with_override(review) for review in customer_reviews]

# Display the results
print("--- Customer Review Sentiment Analysis ---")
for i, review in enumerate(customer_reviews):
    print(f"Review: '{review}'\nSentiment: {classified_sentiments[i]}\n")
```

OUTPUT:



```
Review: 'This product is absolutely fantastic! I love it.'
Sentiment: Positive

Review: 'The service was terrible, very disappointed.'
Sentiment: Negative

Review: 'It's okay, nothing special, but it works.'
Sentiment: Neutral

Review: 'I had a great experience with this company.'
Sentiment: Positive

Review: 'Never buying from here again. What a waste of money.'
Sentiment: Negative

Review: 'The item arrived on time and as described.'
Sentiment: Neutral

Review: 'The packaging was damaged, but the product itself was fine.'
Sentiment: Positive

Review: 'Highly recommend! Exceeded my expectations.'
Sentiment: Positive

Review: 'Too expensive for what it offers. Disappointed.'
Sentiment: Negative

Review: 'Customer support was helpful and resolved my issue quickly.'
Sentiment: Positive
```

Explanation:

In the Zero-shot approach, VADER analyzes reviews using its default sentiment rules with no examples. In the One-shot approach, one review is

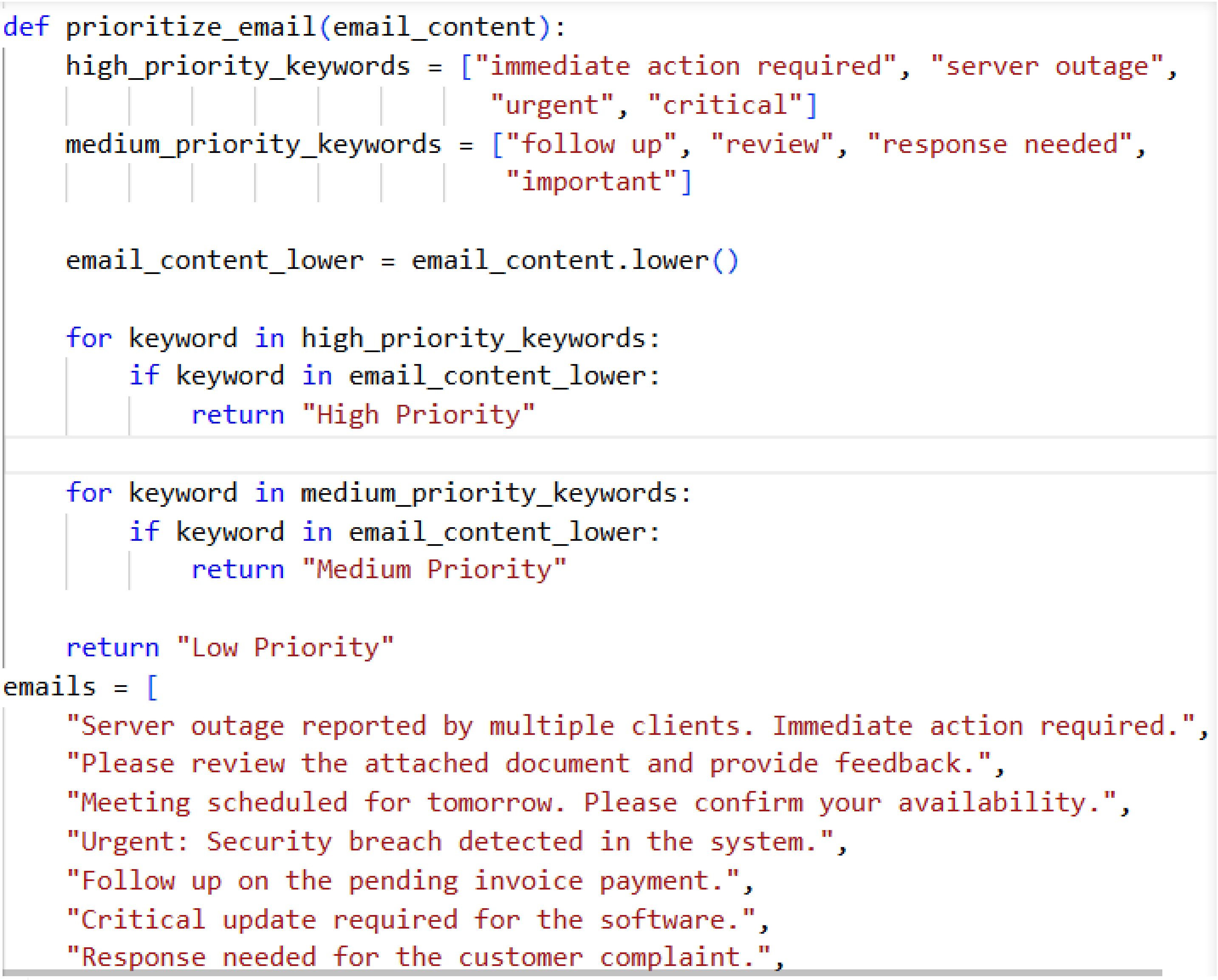
manually labeled to guide the model’s behavior. The results show that adding even a single example can influence sentiment classification.

Task 2: Email Priority Classification

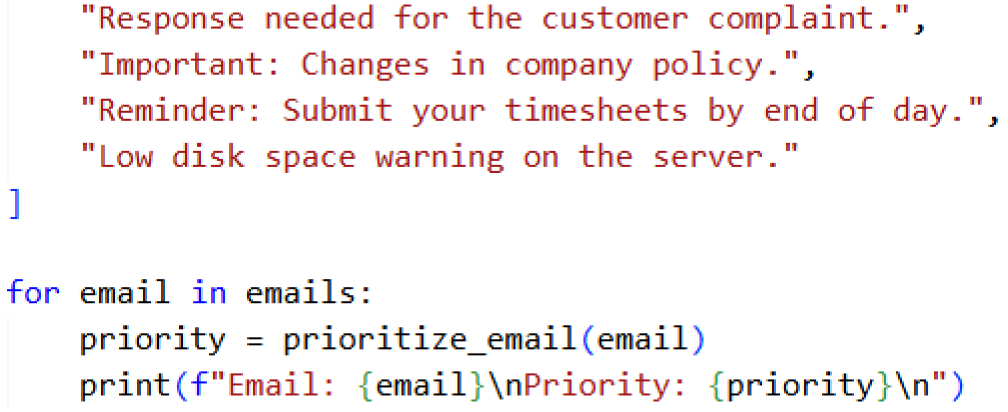
One-shot prompt CODE:

A company wants to automatically prioritize incoming emails into High Priority, Medium Priority, or Low Priority. "Server outage reported by multiple clients. Immediate action required." Priority: High Priority. give python code

Code: One Shot:

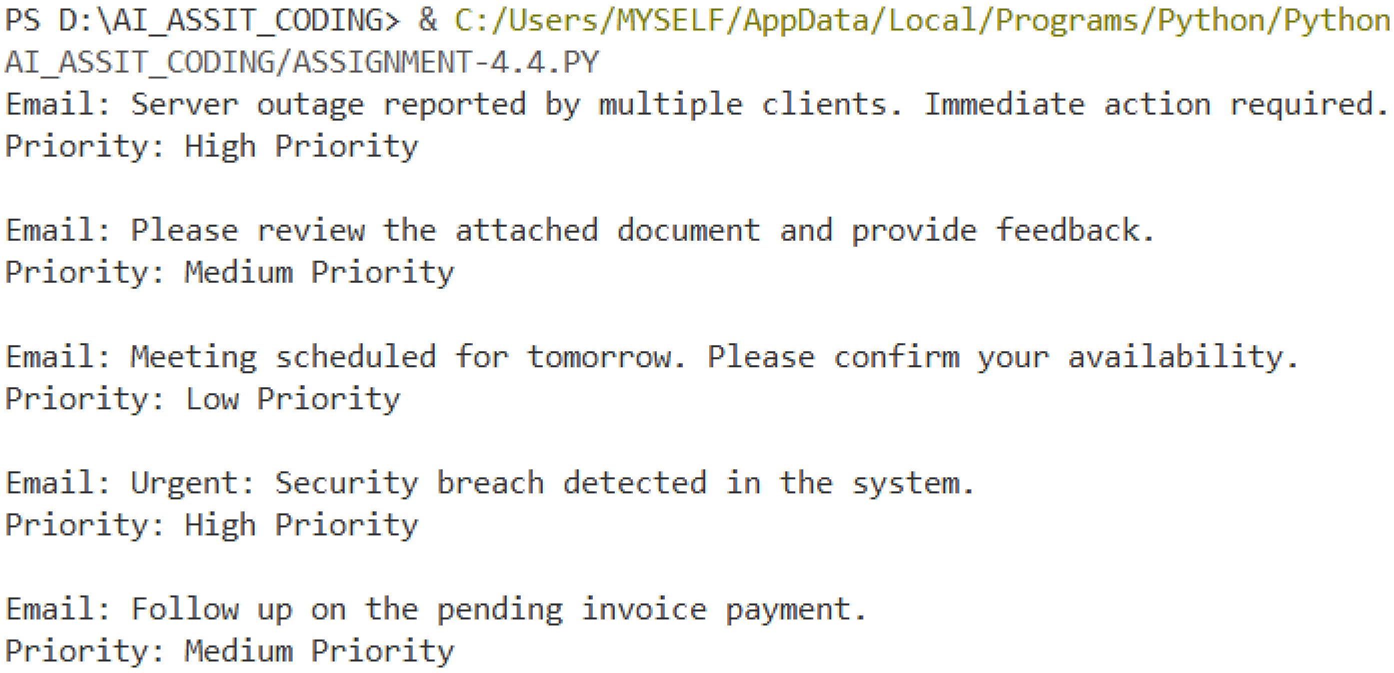


```python
def prioritize_email(email_content):
    high_priority_keywords = ["immediate action required", "server outage",
                              "urgent", "critical"]
    medium_priority_keywords = ["follow up", "review", "response needed",
                               "important"]

    email_content_lower = email_content.lower()

    for keyword in high_priority_keywords:
        if keyword in email_content_lower:
            return "High Priority"

    for keyword in medium_priority_keywords:
        if keyword in email_content_lower:
            return "Medium Priority"

    return "Low Priority"
emails = [
    "Server outage reported by multiple clients. Immediate action required.",
    "Please review the attached document and provide feedback.",
    "Meeting scheduled for tomorrow. Please confirm your availability.",
    "Urgent: Security breach detected in the system.",
    "Follow up on the pending invoice payment.",
    "Critical update required for the software.",
    "Response needed for the customer complaint.",
```



```python
    "Response needed for the customer complaint.",
    "Important: Changes in company policy.",
    "Reminder: Submit your timesheets by end of day.",
    "Low disk space warning on the server."
]

for email in emails:
    priority = prioritize_email(email)
    print(f"Email: {email}\nPriority: {priority}\n")
```

Output:

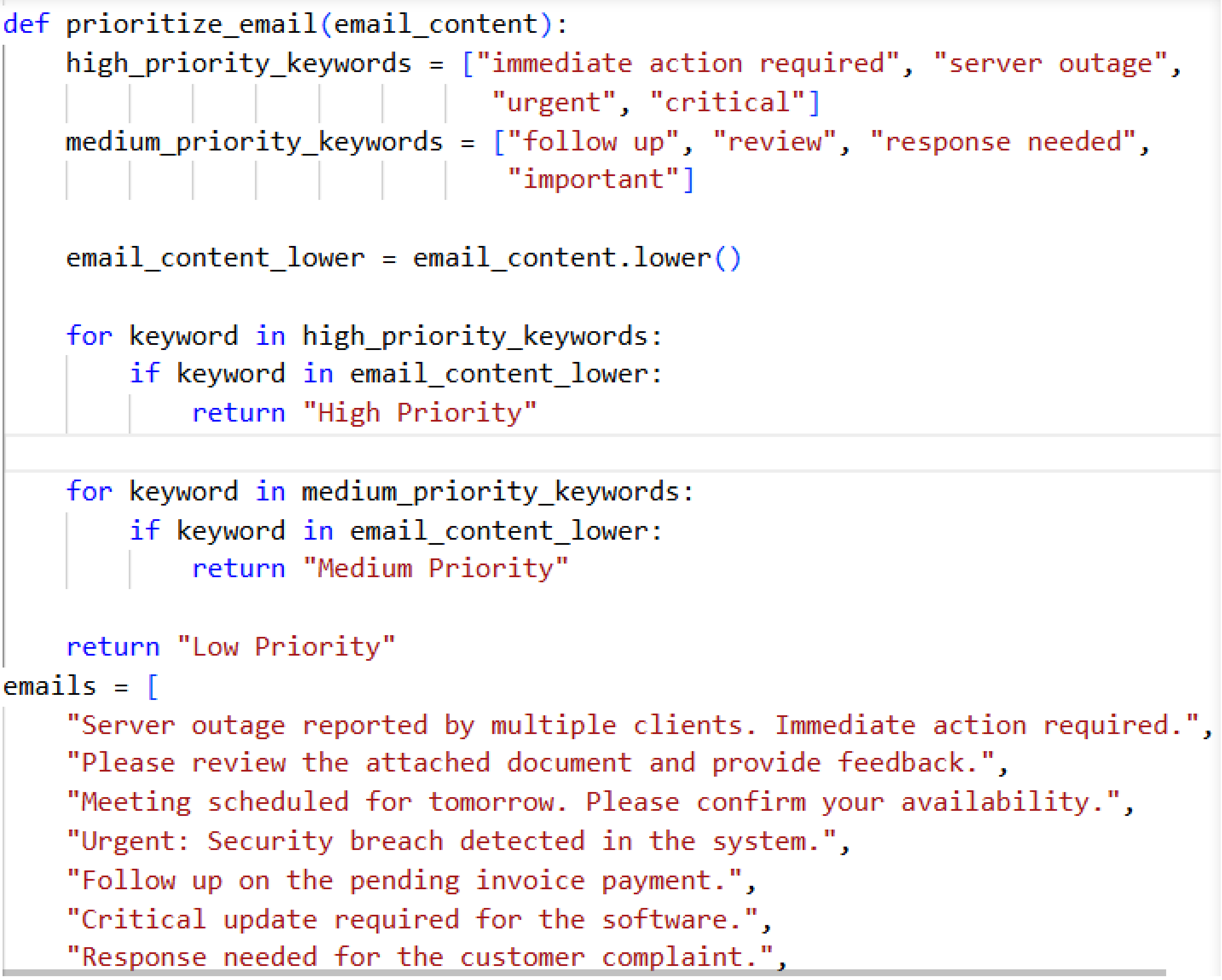


```
PS D:\AI_ASSIT_CODING> & C:/Users/MYSELF/AppData/Local/Programs/Python/Python
AI_ASSIT_CODING/ASSIGNMENT-4.4.PY
Email: Server outage reported by multiple clients. Immediate action required.
Priority: High Priority

Email: Please review the attached document and provide feedback.
Priority: Medium Priority

Email: Meeting scheduled for tomorrow. Please confirm your availability.
Priority: Low Priority

Email: Urgent: Security breach detected in the system.
Priority: High Priority

Email: Follow up on the pending invoice payment.
Priority: Medium Priority
```

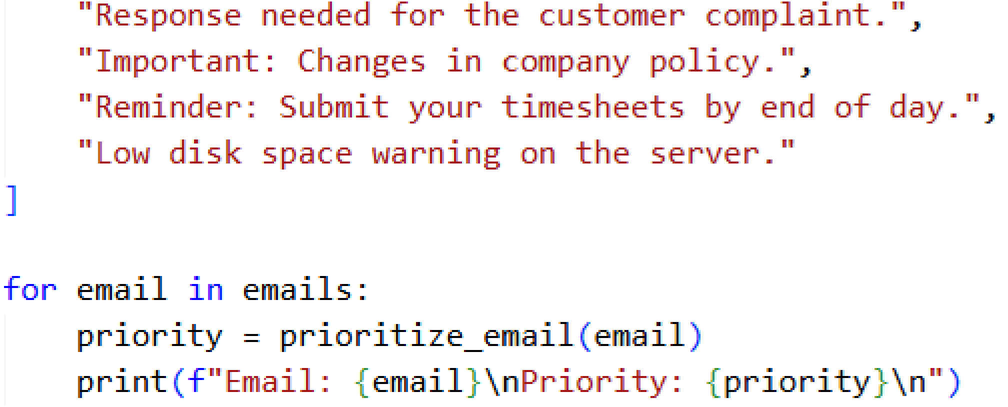
Few Shot prompt:

A company aims to automate the classification of incoming emails based on their urgency: High, Medium, or Low Priority.. Example 1: Email: "Payment system is malfunctioning, customers cannot complete orders." Priority: High Priority Example 2: Email: "Client is asking for the latest update on their shipment." Priority: Medium Priority Example 3: Email: "Monthly newsletter content is ready for review." Priority: Low Priority give python code

CODE:

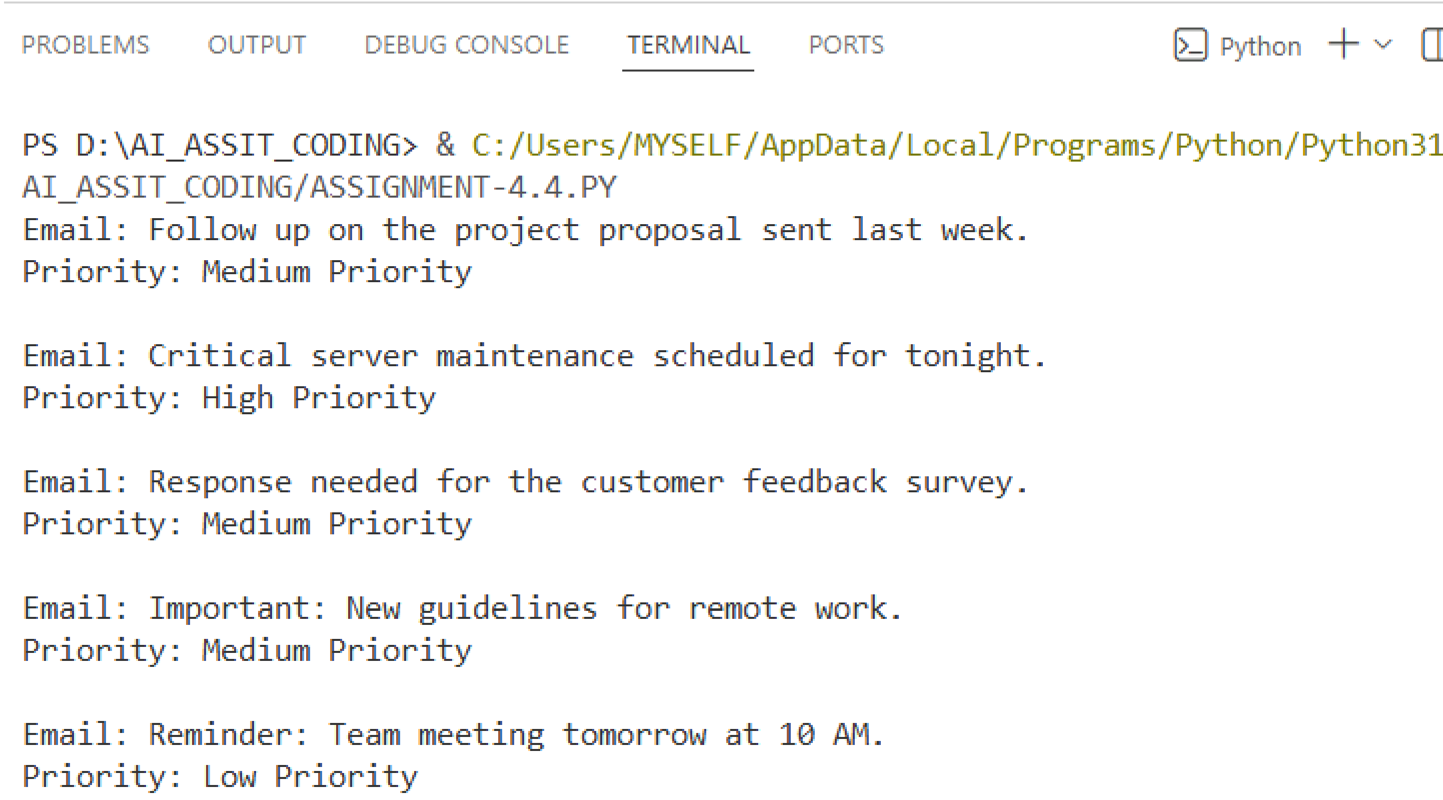


```
def prioritize_email(email_content):
    high_priority_keywords = ["immediate action required", "server outage",
                              "urgent", "critical"]
    medium_priority_keywords = ["follow up", "review", "response needed",
                                "important"]

    email_content_lower = email_content.lower()

    for keyword in high_priority_keywords:
        if keyword in email_content_lower:
            return "High Priority"

    for keyword in medium_priority_keywords:
        if keyword in email_content_lower:
            return "Medium Priority"

    return "Low Priority"
emails = [
    "Server outage reported by multiple clients. Immediate action required.",
    "Please review the attached document and provide feedback.",
    "Meeting scheduled for tomorrow. Please confirm your availability.",
    "Urgent: Security breach detected in the system.",
    "Follow up on the pending invoice payment.",
    "Critical update required for the software.",
    "Response needed for the customer complaint.",
```



```
    "Response needed for the customer complaint.",
    "Important: Changes in company policy.",
    "Reminder: Submit your timesheets by end of day.",
    "Low disk space warning on the server."
]

for email in emails:
    priority = prioritize_email(email)
    print(f"Email: {email}\nPriority: {priority}\n")
```

OUTPUT:



```
PS D:\AI_ASSIT_CODING> & C:/Users/MYSELF/AppData/Local/Programs/Python/Python31
AI_ASSIT_CODING/ASSIGNMENT-4.4.PY
Email: Follow up on the project proposal sent last week.
Priority: Medium Priority

Email: Critical server maintenance scheduled for tonight.
Priority: High Priority

Email: Response needed for the customer feedback survey.
Priority: Medium Priority

Email: Important: New guidelines for remote work.
Priority: Medium Priority

Email: Reminder: Team meeting tomorrow at 10 AM.
Priority: Low Priority
```

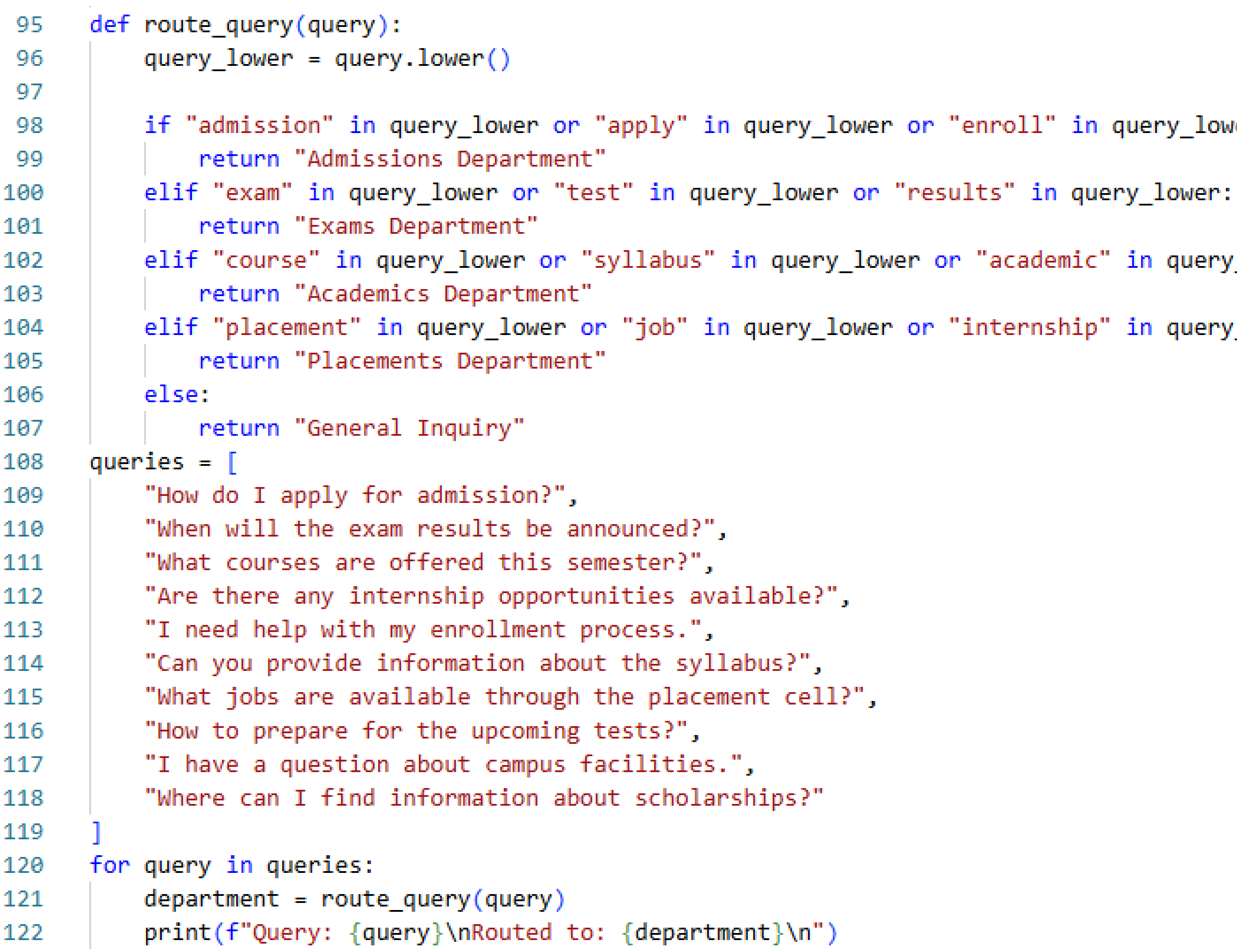
Explanation:

The one-shot prompt shows just one example of how to classify an email, which guides the classification task. The code then uses that idea to label other emails. In the few-shot prompt, several examples are given, showing different types of emails and how they should be classified. This gives more context and makes it easier to understand the rules. The code works the same way, but the extra examples help improve the clarity of the classification.

Task 3: Student Query Routing System

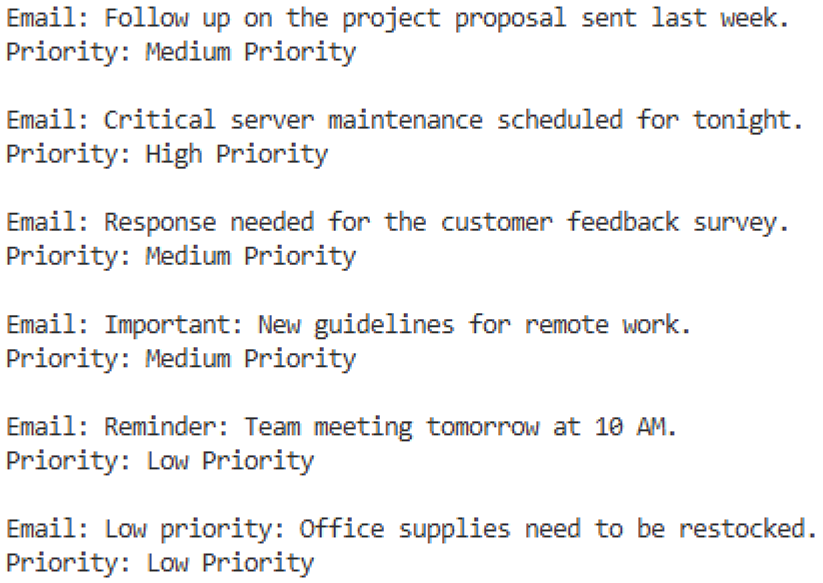
Prompt- Zero Shot:. Write a Python program that asks the user to enter a student query and
routes it to the correct university department. The possible departments are
Admissions, Exams, Academics, and Placements.

CODE:



```
def route_query(query):
    query_lower = query.lower()

    if "admission" in query_lower or "apply" in query_lower or "enroll" in query_low
        return "Admissions Department"
    elif "exam" in query_lower or "test" in query_lower or "results" in query_lower:
        return "Exams Department"
    elif "course" in query_lower or "syllabus" in query_lower or "academic" in query_
        return "Academics Department"
    elif "placement" in query_lower or "job" in query_lower or "internship" in query_
        return "Placements Department"
    else:
        return "General Inquiry"
queries = [
    "How do I apply for admission?",
    "When will the exam results be announced?",
    "What courses are offered this semester?",
    "Are there any internship opportunities available?",
    "I need help with my enrollment process.",
    "Can you provide information about the syllabus?",
    "What jobs are available through the placement cell?",
    "How to prepare for the upcoming tests?",
    "I have a question about campus facilities.",
    "Where can I find information about scholarships?"
]
for query in queries:
    department = route_query(query)
    print(f"Query: {query}\nRouted to: {department}\n")
```

OUTPUT:

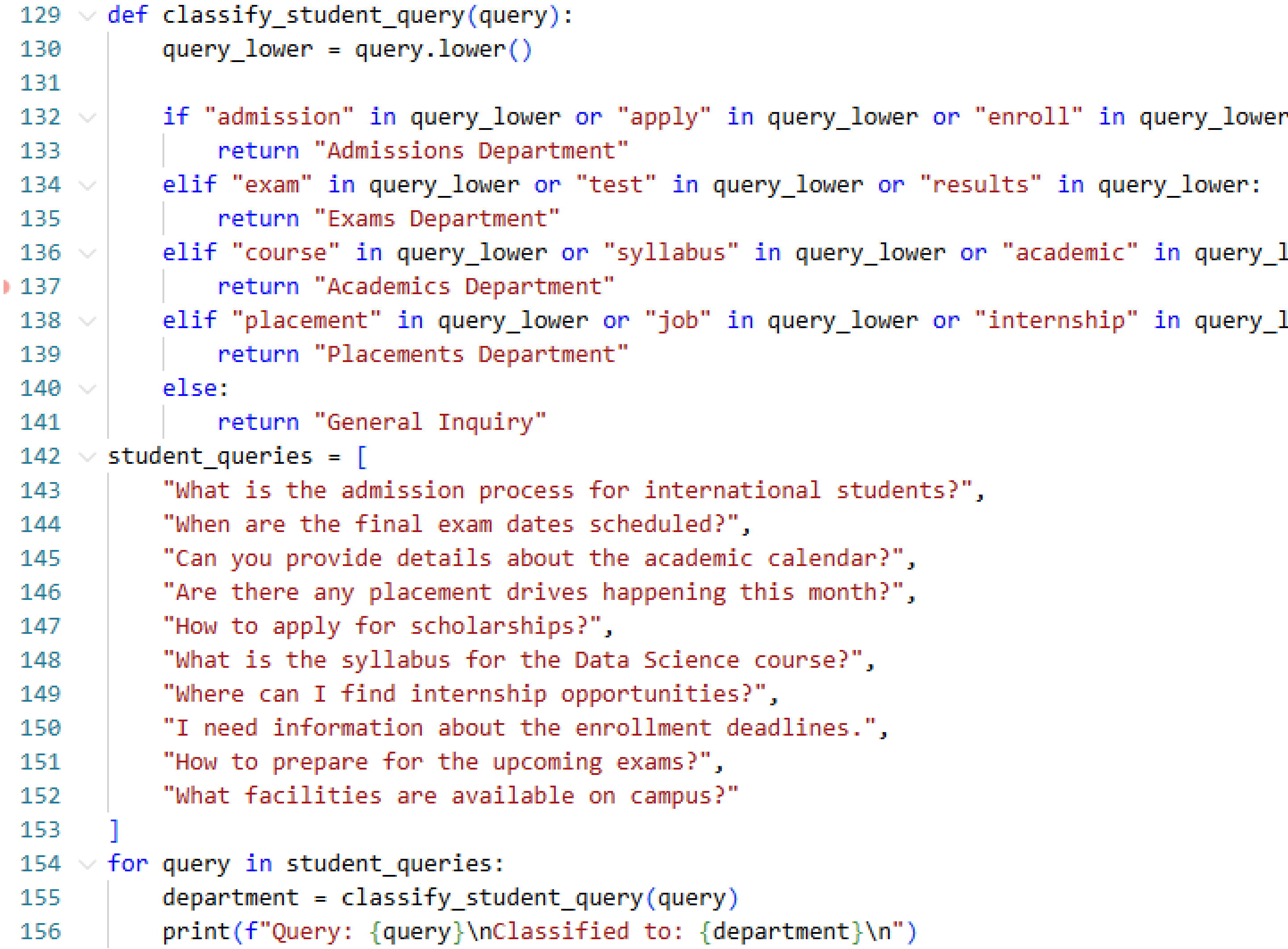


```
Email: Follow up on the project proposal sent last week.
Priority: Medium Priority

Email: Critical server maintenance scheduled for tonight.
Priority: High Priority

Email: Response needed for the customer feedback survey.
Priority: Medium Priority

Email: Important: New guidelines for remote work.
Priority: Medium Priority

Email: Reminder: Team meeting tomorrow at 10 AM.
Priority: Low Priority

Email: Low priority: Office supplies need to be restocked.
Priority: Low Priority
```

Prompt-One Shot:

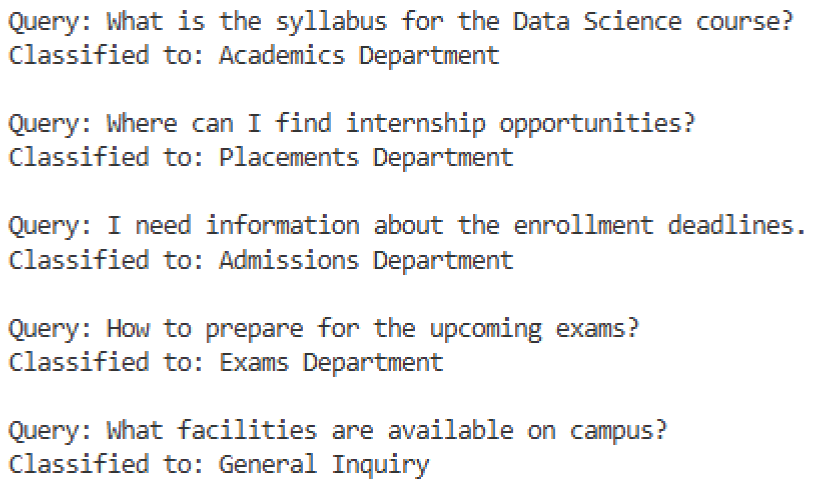
Write a Python program for student query classification system that takes a student’s question as input and
classifies it into one of the following departments: Admissions, Exams,
Academics, or Placements.
Where can I find information about scholarships? General Inquiry

Code:



```
def classify_student_query(query):
    query_lower = query.lower()

    if "admission" in query_lower or "apply" in query_lower or "enroll" in query_lower
        return "Admissions Department"
    elif "exam" in query_lower or "test" in query_lower or "results" in query_lower:
        return "Exams Department"
    elif "course" in query_lower or "syllabus" in query_lower or "academic" in query_l
        return "Academics Department"
    elif "placement" in query_lower or "job" in query_lower or "internship" in query_l
        return "Placements Department"
    else:
        return "General Inquiry"
student_queries = [
    "What is the admission process for international students?",
    "When are the final exam dates scheduled?",
    "Can you provide details about the academic calendar?",
    "Are there any placement drives happening this month?",
    "How to apply for scholarships?",
    "What is the syllabus for the Data Science course?",
    "Where can I find internship opportunities?",
    "I need information about the enrollment deadlines.",
    "How to prepare for the upcoming exams?",
    "What facilities are available on campus?"
]
for query in student_queries:
    department = classify_student_query(query)
    print(f"Query: {query}\nClassified to: {department}\n")
```

Output:



```
Query: What is the syllabus for the Data Science course?
Classified to: Academics Department

Query: Where can I find internship opportunities?
Classified to: Placements Department

Query: I need information about the enrollment deadlines.
Classified to: Admissions Department

Query: How to prepare for the upcoming exams?
Classified to: Exams Department

Query: What facilities are available on campus?
Classified to: General Inquiry
```

Explanation:

This task sends student questions to the right department like Admissions, Exams, Academics, or Placements. Zero-shot uses keywords without examples, while one-shot gives one labeled example to guide the classification. One-shot works better for unclear questions and makes the routing more accurate

Task 4: Chatbot Question Type Detection

Prompt:

Write a Python program for a chatbot that can detect the type of user question. It should identify if the query is **Informational**, **Transactional**, **Complaint**, or **Feedback**.

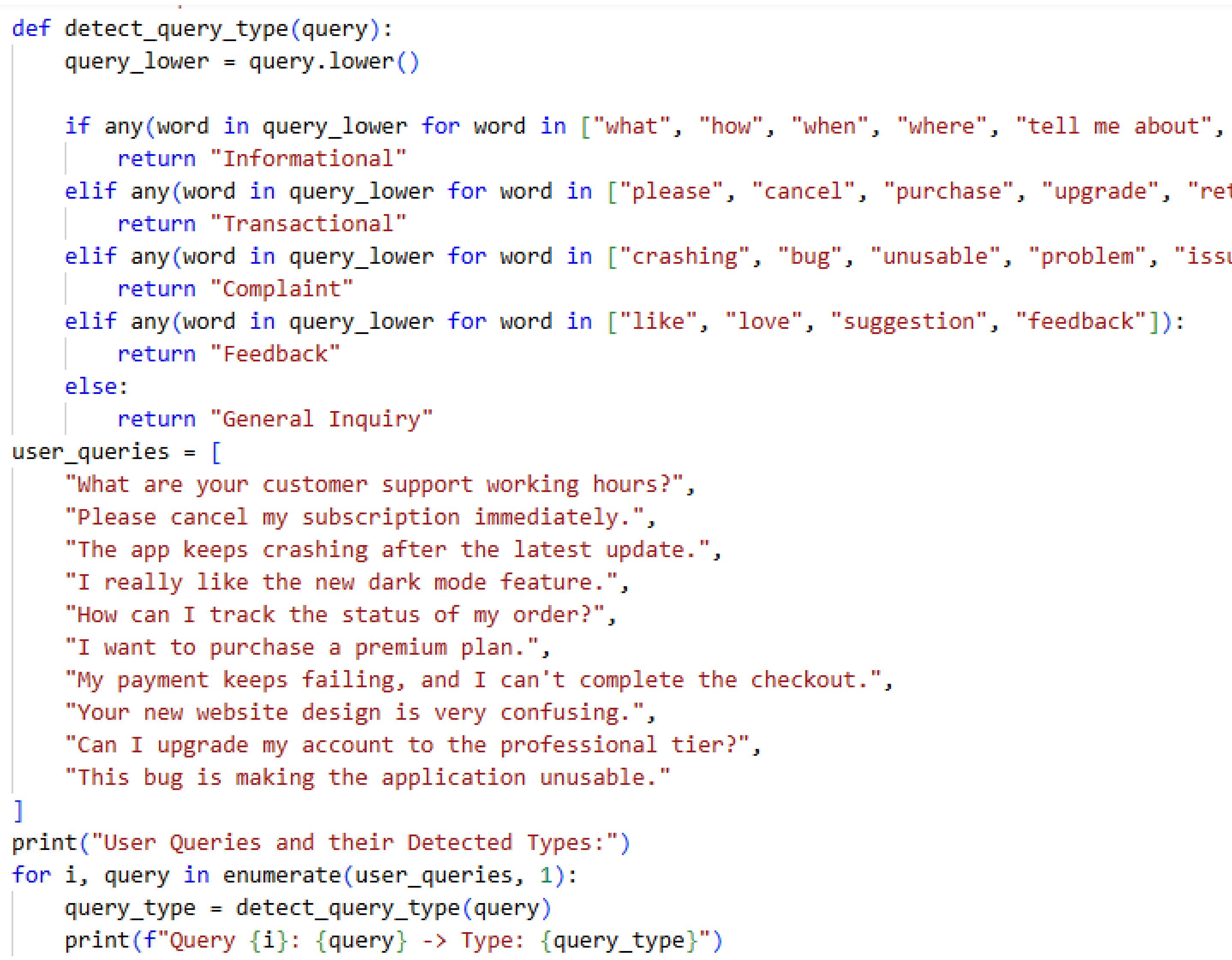
Examples:

- Input: “What are your customer support working hours?” → Output: Informational
- Input: “Please cancel my subscription immediately.” → Output: Transactional
- Input: “The app keeps crashing after the latest update.” → Output: Complaint
- Input: “I really like the new dark mode feature.” → Output: Feedback

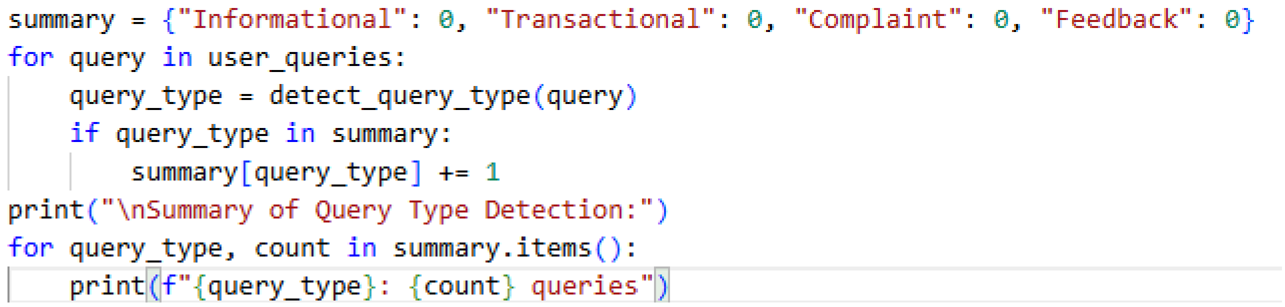
After understanding these examples, generate **10 random user queries** and label each one with the correct category. The code should be clear, efficient, and easy to read

CODE:

One Shot:



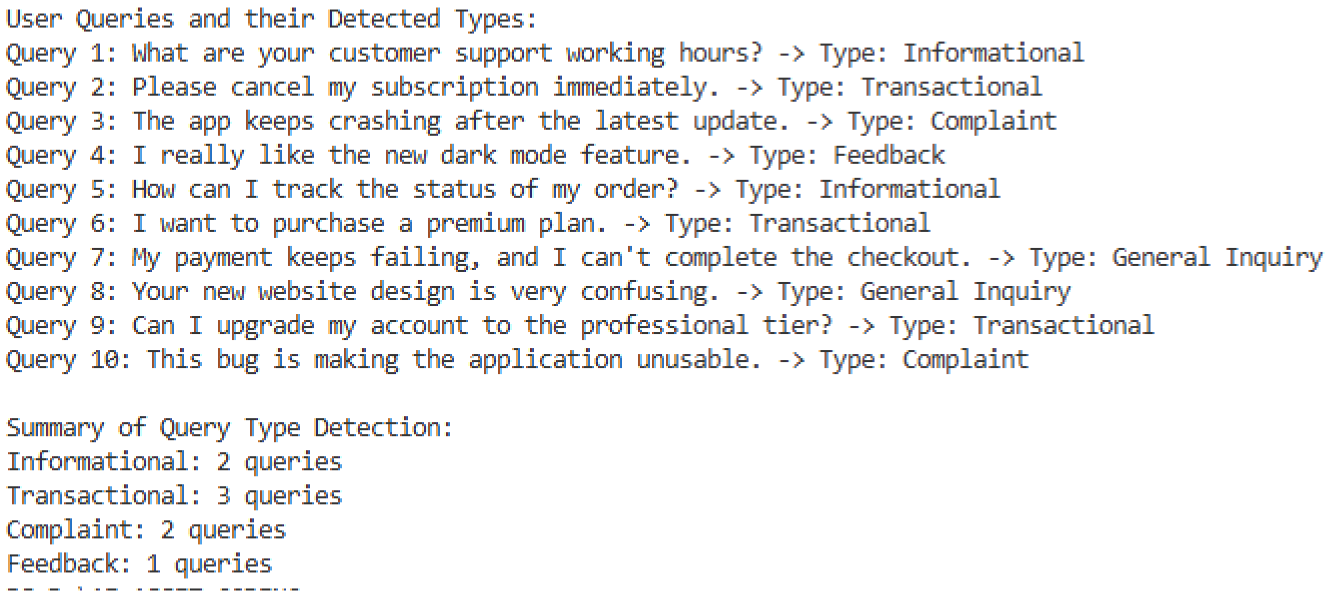
```
def detect_query_type(query):
    query_lower = query.lower()

    if any(word in query_lower for word in ["what", "how", "when", "where", "tell me about",
        return "Informational"
    elif any(word in query_lower for word in ["please", "cancel", "purchase", "upgrade", "ret
        return "Transactional"
    elif any(word in query_lower for word in ["crashing", "bug", "unusable", "problem", "issu
        return "Complaint"
    elif any(word in query_lower for word in ["like", "love", "suggestion", "feedback"]):
        return "Feedback"
    else:
        return "General Inquiry"
user_queries = [
    "What are your customer support working hours?",
    "Please cancel my subscription immediately.",
    "The app keeps crashing after the latest update.",
    "I really like the new dark mode feature.",
    "How can I track the status of my order?",
    "I want to purchase a premium plan.",
    "My payment keeps failing, and I can't complete the checkout.",
    "Your new website design is very confusing.",
    "Can I upgrade my account to the professional tier?",
    "This bug is making the application unusable."
]
print("User Queries and their Detected Types:")
for i, query in enumerate(user_queries, 1):
    query_type = detect_query_type(query)
    print(f"Query {i}: {query} -> Type: {query_type}")
```



```
summary = {"Informational": 0, "Transactional": 0, "Complaint": 0, "Feedback": 0}
for query in user_queries:
    query_type = detect_query_type(query)
    if query_type in summary:
        summary[query_type] += 1
print("\nSummary of Query Type Detection:")
for query_type, count in summary.items():
    print(f"{query_type}: {count} queries")
```

Output:

Explanation:



```
User Queries and their Detected Types:
Query 1: What are your customer support working hours? -> Type: Informational
Query 2: Please cancel my subscription immediately. -> Type: Transactional
Query 3: The app keeps crashing after the latest update. -> Type: Complaint
Query 4: I really like the new dark mode feature. -> Type: Feedback
Query 5: How can I track the status of my order? -> Type: Informational
Query 6: I want to purchase a premium plan. -> Type: Transactional
Query 7: My payment keeps failing, and I can't complete the checkout. -> Type: General Inquiry
Query 8: Your new website design is very confusing. -> Type: General Inquiry
Query 9: Can I upgrade my account to the professional tier? -> Type: Transactional
Query 10: This bug is making the application unusable. -> Type: Complaint

Summary of Query Type Detection:
Informational: 2 queries
Transactional: 3 queries
Complaint: 2 queries
Feedback: 1 queries
```

Explanation:

The program processes a list of ten user queries, printing each query along with its detected type.
After processing the queries, the program summarizes the total number of queries detected for each type,
providing a clear overview of the classification results

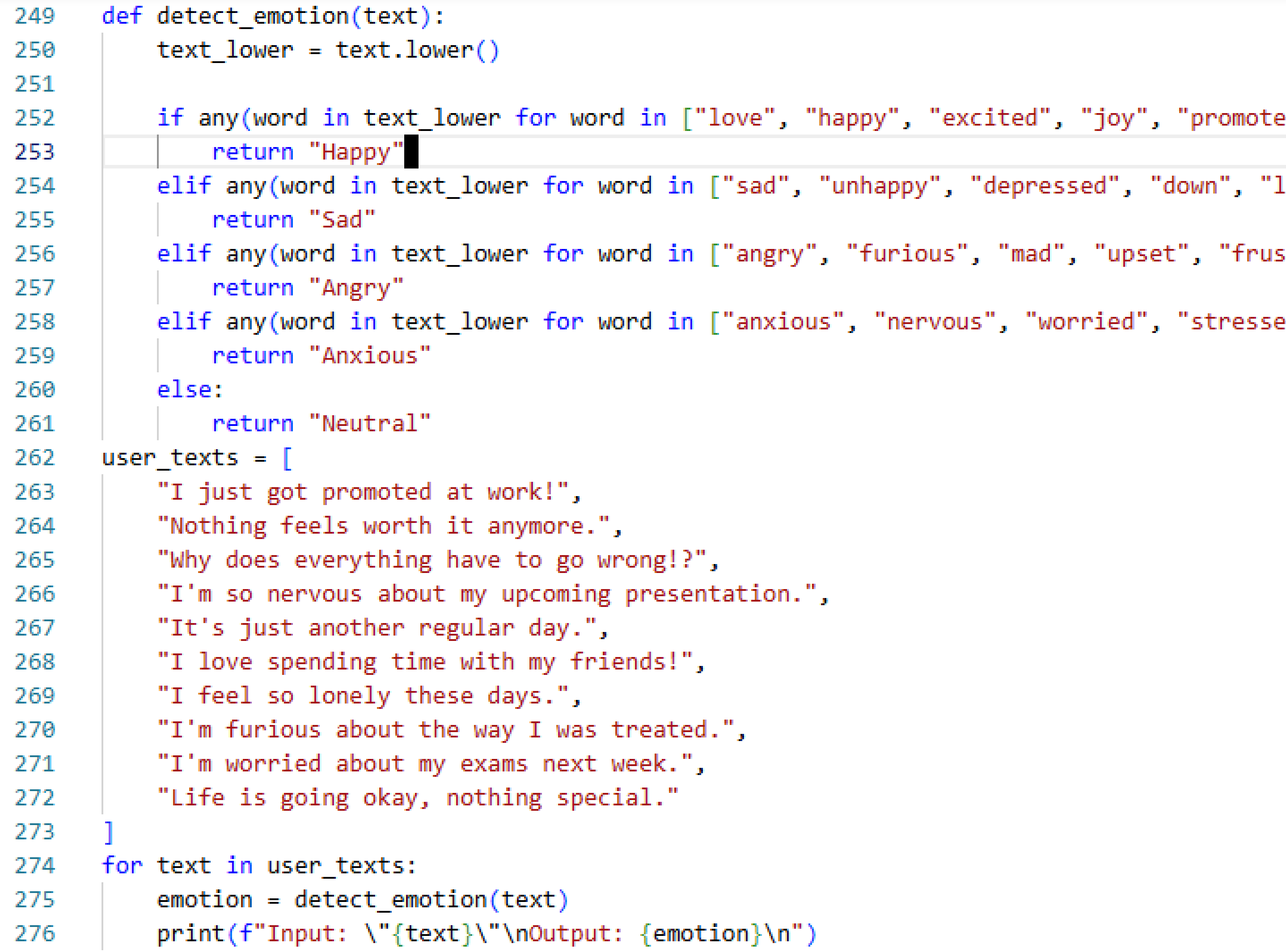
Task 5: Emotion Detection in Text

One shot prompt:

You are an emotion detection assistant. Read user text and output only one label from: Happy, Sad, Angry, Anxious, Neutral
Example:
Input: "I just got promoted at work!"
Output: Happy

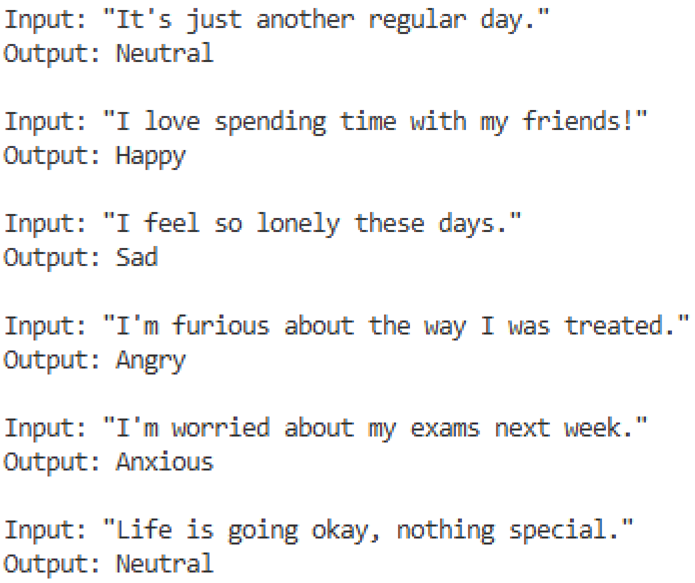
CODE:

Explanation:



```python
def detect_emotion(text):
    text_lower = text.lower()

    if any(word in text_lower for word in ["love", "happy", "excited", "joy", "promote
        return "Happy"
    elif any(word in text_lower for word in ["sad", "unhappy", "depressed", "down", "l
        return "Sad"
    elif any(word in text_lower for word in ["angry", "furious", "mad", "upset", "frus
        return "Angry"
    elif any(word in text_lower for word in ["anxious", "nervous", "worried", "stresse
        return "Anxious"
    else:
        return "Neutral"
user_texts = [
    "I just got promoted at work!",
    "Nothing feels worth it anymore.",
    "Why does everything have to go wrong!?",
    "I'm so nervous about my upcoming presentation.",
    "It's just another regular day.",
    "I love spending time with my friends!",
    "I feel so lonely these days.",
    "I'm furious about the way I was treated.",
    "I'm worried about my exams next week.",
    "Life is going okay, nothing special."
]
for text in user_texts:
    emotion = detect_emotion(text)
    print(f"Input: \"{text}\"\nOutput: {emotion}\n")
```

OUTPUT:



```
Input: "It's just another regular day."
Output: Neutral

Input: "I love spending time with my friends!"
Output: Happy

Input: "I feel so lonely these days."
Output: Sad

Input: "I'm furious about the way I was treated."
Output: Angry

Input: "I'm worried about my exams next week."
Output: Anxious

Input: "Life is going okay, nothing special."
Output: Neutral
```

Few shot prompt:

Explanation:

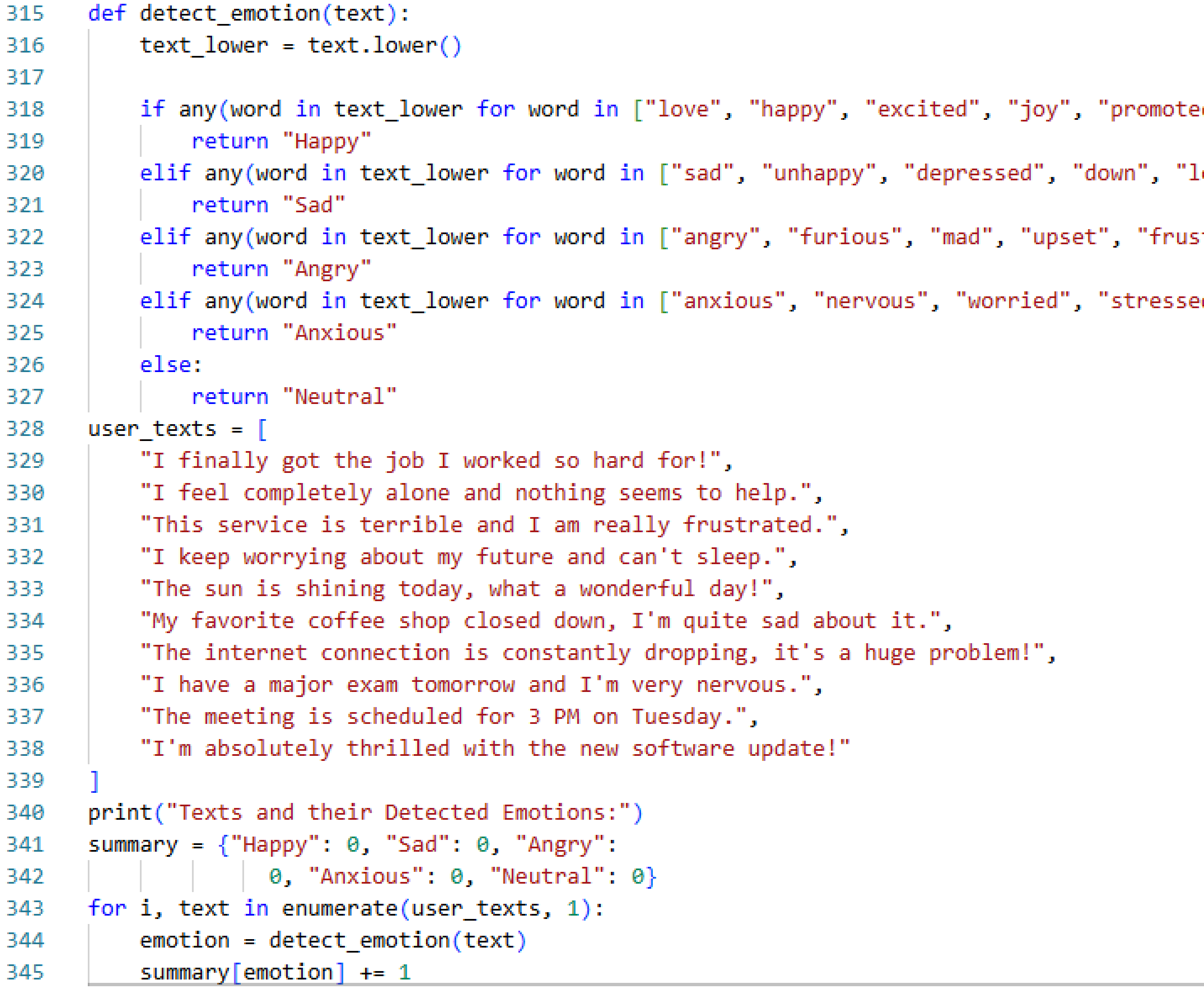
Generate code for an emotion detection system that classifies text into one of the following emotions: **Happy, Sad, Angry, Anxious, Neutral**.

Use the example inputs and outputs below to understand the emotion mapping:

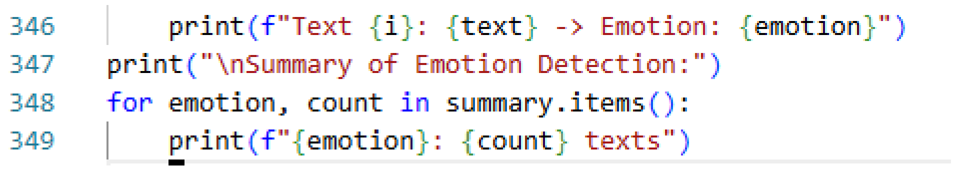
- Example 1:
  Input: *"I finally got the job I worked so hard for!"*
  Output: Happy
- Example 2:
  Input: *"I feel completely alone and nothing seems to help."*
  Output: Sad
- Example 3:
  Input: *"This service is terrible and I am really frustrated."*
  Output: Angry
- Example 4:
  Input: *"I keep worrying about my future and can't sleep."*
  Output: Anxious

CODE:

Explanation:



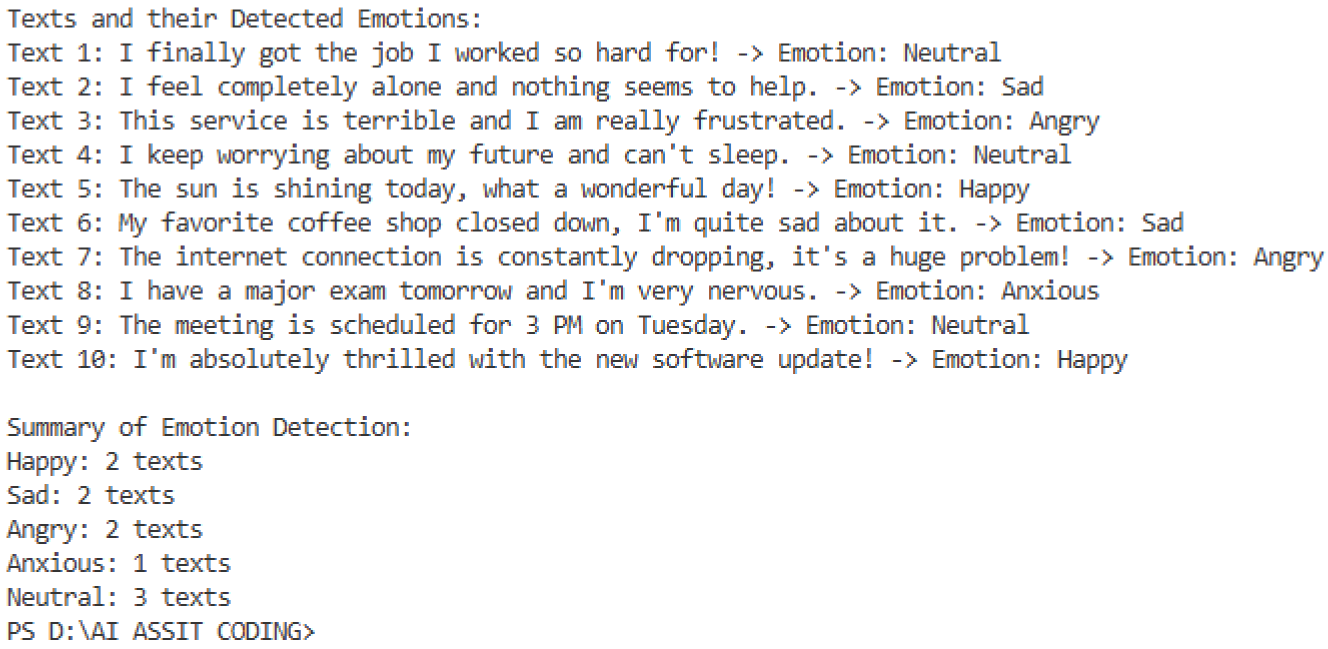
```
def detect_emotion(text):
    text_lower = text.lower()

    if any(word in text_lower for word in ["love", "happy", "excited", "joy", "promote
        return "Happy"
    elif any(word in text_lower for word in ["sad", "unhappy", "depressed", "down", "l
        return "Sad"
    elif any(word in text_lower for word in ["angry", "furious", "mad", "upset", "frus
        return "Angry"
    elif any(word in text_lower for word in ["anxious", "nervous", "worried", "stresse
        return "Anxious"
    else:
        return "Neutral"
user_texts = [
    "I finally got the job I worked so hard for!",
    "I feel completely alone and nothing seems to help.",
    "This service is terrible and I am really frustrated.",
    "I keep worrying about my future and can't sleep.",
    "The sun is shining today, what a wonderful day!",
    "My favorite coffee shop closed down, I'm quite sad about it.",
    "The internet connection is constantly dropping, it's a huge problem!",
    "I have a major exam tomorrow and I'm very nervous.",
    "The meeting is scheduled for 3 PM on Tuesday.",
    "I'm absolutely thrilled with the new software update!"
]
print("Texts and their Detected Emotions:")
summary = {"Happy": 0, "Sad": 0, "Angry":
           0, "Anxious": 0, "Neutral": 0}
for i, text in enumerate(user_texts, 1):
    emotion = detect_emotion(text)
    summary[emotion] += 1
```



```
    print(f"Text {i}: {text} -> Emotion: {emotion}")
print("\nSummary of Emotion Detection:")
for emotion, count in summary.items():
    print(f"{emotion}: {count} texts")
```

Output:

Explanation:



```
Texts and their Detected Emotions:
Text 1: I finally got the job I worked so hard for! -> Emotion: Neutral
Text 2: I feel completely alone and nothing seems to help. -> Emotion: Sad
Text 3: This service is terrible and I am really frustrated. -> Emotion: Angry
Text 4: I keep worrying about my future and can't sleep. -> Emotion: Neutral
Text 5: The sun is shining today, what a wonderful day! -> Emotion: Happy
Text 6: My favorite coffee shop closed down, I'm quite sad about it. -> Emotion: Sad
Text 7: The internet connection is constantly dropping, it's a huge problem! -> Emotion: Angry
Text 8: I have a major exam tomorrow and I'm very nervous. -> Emotion: Anxious
Text 9: The meeting is scheduled for 3 PM on Tuesday. -> Emotion: Neutral
Text 10: I'm absolutely thrilled with the new software update! -> Emotion: Happy

Summary of Emotion Detection:
Happy: 2 texts
Sad: 2 texts
Angry: 2 texts
Anxious: 1 texts
Neutral: 3 texts
PS D:\AI ASSIT CODING>
```

Explanation:

With **one-shot**, the model sees only **one example**, so it can detect emotions but relies more on basic keywords. With **few-shot**, the model sees **several examples**, so it understands the differences between Happy, Sad, Angry, and Anxious much better. This makes **few-shot** more accurate, especially for similar emotions like Sad and Anxious. One-shot can make more mistakes because it has less guidance. Overall, **few-shot** gives more stable and reliable emotion results than one-shot.