

LAB ASSIGNMENT 4.4

NAME: M.Geethika

ID NO: 2303A52276

SUBJECT: AI ASST CODING

1.Scenario

The screenshot shows a Google Colab notebook titled "Untitled31.ipynb". In cell [1], Python code is run to print sample customer reviews with their corresponding sentiment labels. The output shows various reviews ranging from positive to negative, with their sentiment categorized as Positive, Negative, or Neutral.

```
# Sample customer reviews with sentiment labels
reviews = [
    ("The product quality is excellent and delivery was fast.", "Positive"),
    ("Very disappointed, the item arrived broken.", "Negative"),
    ("The packaging was okay, nothing special.", "Neutral"),
    ("Amazing service and great value for money!", "Positive"),
    ("Customer support was slow and unhelpful.", "Negative"),
    ("The product works as described.", "Neutral")
]

print("Sample Customer Reviews:\n")
for review, sentiment in reviews:
    print(f"Review: {review}\nSentiment: {sentiment}\n")

... Sample Customer Reviews:

Review: The product quality is excellent and delivery was fast.
Sentiment: Positive

Review: Very disappointed, the item arrived broken.
Sentiment: Negative

Review: The packaging was okay, nothing special.
Sentiment: Neutral

Review: Amazing service and great value for money!
Sentiment: Positive

Review: Customer support was slow and unhelpful.
Sentiment: Negative

Review: The product works as described.
Sentiment: Neutral
```

The screenshot shows a Google Colab notebook titled "Untitled31.ipynb". In cell [1], a function named "show_prompt" is defined to print a prompt and its corresponding model output. In cell [3], a zero-shot classification example is run. It defines a prompt asking to classify a review as Positive, Negative, or Neutral. The review provided is "Very disappointed, the item arrived broken.". The function "show_prompt" is then called with this prompt and the output "Negative".

```
def show_prompt(prompt, output):
    print("PROMPT:\n")
    print(prompt)
    print("\nMODEL OUTPUT:\n")
    print(output)
    print("-" * 60)

zero_shot_prompt = """
Classify the following customer review as:
Positive, Negative, or Neutral.

Review: "Very disappointed, the item arrived broken."
"""

show_prompt(zero_shot_prompt, "Negative")

... PROMPT:

Classify the following customer review as:
Positive, Negative, or Neutral.

Review: "Very disappointed, the item arrived broken."

MODEL OUTPUT:
Negative
```

The screenshot shows a Google Colab notebook titled "Untitled31.ipynb". The code cell contains Python code for sentiment analysis using a one-shot prompt. It includes examples for positive, negative, and neutral reviews, and a prompt for classifying a review. The output cell shows the model's classification of the review as "Negative".

```
one_shot_prompt = """  
Example:  
Review: "The product quality is excellent and delivery was fast."  
Sentiment: Positive  
  
Now classify the following review:  
Review: "Customer support was slow and unhelpful."  
***  
  
show_prompt(one_shot_prompt, "Negative")  
  
PROMPT:  
  
Example:  
Review: "The product quality is excellent and delivery was fast."  
Sentiment: Positive  
  
Now classify the following review:  
Review: "Customer support was slow and unhelpful."  
  
MODEL OUTPUT:  
Negative
```

Below the code cell, there is another code cell with a different prompt, labeled [5]. The output cell for this cell is empty.

The screenshot shows a Google Colab notebook titled "Untitled31.ipynb". The code cell contains Python code for sentiment analysis using a few-shot prompt. It includes examples for negative, neutral, and positive reviews, and a prompt for classifying a review. The output cell shows the model's classification of the review as "Neutral".

```
few_shot_prompt = """  
Example 1:  
Review: "Amazing service and great value for money!"  
Sentiment: Positive  
  
Example 2:  
Review: "Very disappointed, the item arrived broken."  
Sentiment: Negative  
  
Example 3:  
Review: "The packaging was okay, nothing special."  
Sentiment: Neutral  
  
Now classify the following review:  
Review: "The product works as described."  
***  
  
show_prompt(few_shot_prompt, "Neutral")  
  
PROMPT:  
  
Example 1:  
Review: "Amazing service and great value for money!"  
Sentiment: Positive  
  
Example 2:  
Review: "Very disappointed, the item arrived broken."  
Sentiment: Negative  
  
Example 3:  
Review: "The packaging was okay, nothing special."  
Sentiment: Neutral  
  
Now classify the following review:  
Review: "The product works as described."  
  
MODEL OUTPUT:
```

Below the code cell, there is another code cell with a different prompt, labeled [5]. The output cell for this cell is empty.

```
comparison = """
Comparison of Prompting Techniques:

Zero-shot:
- Works well for strong opinions
- Struggles with neutral or subtle reviews

One-shot:
- Better understanding of sentiment boundaries
- More consistent than zero-shot

Few-shot:
- Highest accuracy
- Handles neutral and mixed sentiments well
- Best choice for real-world applications
"""

print(comparison)

Comparison of Prompting Techniques:
Zero-shot:
- Works well for strong opinions
- Struggles with neutral or subtle reviews
One-shot:
- Better understanding of sentiment boundaries
- More consistent than zero-shot
Few-shot:
- Highest accuracy
- Handles neutral and mixed sentiments well
- Best choice for real-world applications
```

FINAL CONCLUSION:

Few-shot prompting provides the best sentiment classification accuracy because multiple examples help the model understand sentiment patterns, tone, and neutrality more effectively than zero-shot or one-shot methods.

2.Scenario

The screenshot shows a Google Colab notebook titled "Untitled31.ipynb". The code defines a scenario with sample emails and their priority levels. The output displays the emails and their corresponding priorities.

```
#scenario 2
# Sample emails with priority labels
emails = [
    ("My account has been charged twice, please resolve immediately.", "High"),
    ("The application server is down and needs urgent attention.", "High"),
    ("Can you update me on the status of my request?", "Medium"),
    ("Please schedule the meeting for next week.", "Medium"),
    ("Thank you for the quick support!", "Low"),
    ("Just checking in to see if there are any updates.", "Low")
]

print("Sample Email Messages:")
for email, priority in emails:
    print(f"Email: {email}")
    print(f"Priority: {priority}\n")
```

... Sample Email Messages:

```
Email: My account has been charged twice, please resolve immediately.
Priority: High

Email: The application server is down and needs urgent attention.
Priority: High

Email: Can you update me on the status of my request?
Priority: Medium

Email: Please schedule the meeting for next week.
Priority: Medium

Email: Thank you for the quick support!
Priority: Low

Email: Just checking in to see if there are any updates.
Priority: Low
```

The screenshot shows a Google Colab notebook titled "Untitled31.ipynb". The code defines a function to show prompts and outputs. It then provides a zero-shot prompt for classifying an email into priorities: High Priority, Medium Priority, or Low Priority. The output shows the prompt and the user's response.

```
def show_prompt(prompt, output):
    print("PROMPT:\n")
    print(prompt)
    print("\nMODEL OUTPUT:\n")
    print(output)
    print("-" * 60)

zero_shot_prompt = """
Classify the following email into one of these priorities:
High Priority, Medium Priority, Low Priority.

Email: "The application server is down and needs urgent attention."
"""

show_prompt(zero_shot_prompt, "High Priority")

PROMPT:

Classify the following email into one of these priorities:
High Priority, Medium Priority, Low Priority.

Email: "The application server is down and needs urgent attention."

MODEL OUTPUT:
High Priority
```

```
[18]
one_shot_prompt = """
...-...-...
```

The screenshot shows a Google Colab notebook titled "Untitled31.ipynb". The code cell [10] contains Python code for handling email priority classification. It defines two functions: `one_shot_prompt` and `few_shot_prompt`. The `one_shot_prompt` function takes an email example and a priority level. The `few_shot_prompt` function takes multiple examples and a priority level. Both functions call `show_prompt` with the prompt and priority. The output shows the prompt and the resulting priority classification ("Medium Priority"). The code cell [11] contains the same logic for `few_shot_prompt`, demonstrating its use with three examples and resulting in "Medium Priority". The status bar at the bottom indicates Python 3 and the current date and time.

```
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
[10]: one_shot_prompt = """
Example:
Email: "My account has been charged twice, please resolve immediately."
Priority: High Priority

Now classify the following email:
Email: "Can you update me on the status of my request?"
"""

show_prompt(one_shot_prompt, "Medium Priority")

PROMPT:

Example:
Email: "My account has been charged twice, please resolve immediately."
Priority: High Priority

Now classify the following email:
Email: "Can you update me on the status of my request?"

MODEL OUTPUT:
Medium Priority

[11]: few_shot_prompt = """
Example 1:
Email: "The application server is down and needs urgent attention."
Priority: High Priority

Example 2:
Email: "Please schedule the meeting for next week."
Priority: Medium Priority

Example 3:
Email: "Thank you for the quick support!"
Priority: Low Priority

Now classify the following email:
Email: "Just checking in to see if there are any updates."
"""

show_prompt(few_shot_prompt, "Low Priority")

PROMPT:

Example 1:
Email: "The application server is down and needs urgent attention."
Priority: High Priority

Example 2:
Email: "Please schedule the meeting for next week."
Priority: Medium Priority

Example 3:
Email: "Thank you for the quick support!"
Priority: Low Priority

Now classify the following email:
Email: "Just checking in to see if there are any updates."

MODEL OUTPUT:
Medium Priority
```

The screenshot shows a Google Colab notebook titled "Untitled31.ipynb". The code cell [11] contains Python code for handling email priority classification. It defines two functions: `one_shot_prompt` and `few_shot_prompt`. The `one_shot_prompt` function takes an email example and a priority level. The `few_shot_prompt` function takes multiple examples and a priority level. Both functions call `show_prompt` with the prompt and priority. The output shows the prompt and the resulting priority classification ("Medium Priority"). The code cell [12] contains the same logic for `few_shot_prompt`, demonstrating its use with three examples and resulting in "Medium Priority". The status bar at the bottom indicates Python 3 and the current date and time.

```
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
[11]: one_shot_prompt = """
Example:
Email: "My account has been charged twice, please resolve immediately."
Priority: High Priority

Now classify the following email:
Email: "Can you update me on the status of my request?"
"""

show_prompt(one_shot_prompt, "Medium Priority")

PROMPT:

Example:
Email: "My account has been charged twice, please resolve immediately."
Priority: High Priority

Now classify the following email:
Email: "Can you update me on the status of my request?"

MODEL OUTPUT:
Medium Priority

[12]: few_shot_prompt = """
Example 1:
Email: "The application server is down and needs urgent attention."
Priority: High Priority

Example 2:
Email: "Please schedule the meeting for next week."
Priority: Medium Priority

Example 3:
Email: "Thank you for the quick support!"
Priority: Low Priority

Now classify the following email:
Email: "Just checking in to see if there are any updates."
"""

show_prompt(few_shot_prompt, "Low Priority")

PROMPT:

Example 1:
Email: "The application server is down and needs urgent attention."
Priority: High Priority

Example 2:
Email: "Please schedule the meeting for next week."
Priority: Medium Priority

Example 3:
Email: "Thank you for the quick support!"
Priority: Low Priority

Now classify the following email:
Email: "Just checking in to see if there are any updates."

MODEL OUTPUT:
Medium Priority
```

```
evaluation = """
Evaluation of Prompting Techniques:

Zero-shot:
- Fast and simple
- Works for clearly urgent emails
- Less reliable for borderline cases

One-shot:
- Provides better guidance
- Reduces ambiguity

Few-shot:
- Highest accuracy
- Learns priority patterns effectively
- Best suited for production systems
"""

print(evaluation)

Evaluation of Prompting Techniques:

Zero-shot:
- Fast and simple
- Works for clearly urgent emails
- Less reliable for borderline cases

One-shot:
- Provides better guidance
- Reduces ambiguity

Few-shot:
- Highest accuracy
```

FINAL CONCLUSION:

Few-shot prompting produces the most reliable results because multiple examples help the model clearly distinguish between high, medium, and low urgency patterns in emails.

3.Scenario

The screenshot shows a Google Colab notebook titled "Untitled31.ipynb". The code cell contains Python code to generate sample student queries with department labels. The output cell shows the generated queries and their corresponding departments.

```
#3 scenario
# Sample student queries with department labels
student_queries = [
    ("What is the last date to apply for the MBA program?", "Admissions"),
    ("How can I download my hall ticket for the semester exam?", "Exams"),
    ("Can you explain the syllabus for Data Structures?", "Academics"),
    ("When will campus placement drives start?", "Placements"),
    ("Is there any entrance exam for B.Tech admissions?", "Admissions"),
    ("How do I apply for internships through college?", "Placements")
]

print("Sample Student Queries:\n")
for query, dept in student_queries:
    print(f"Query: {query}")
    print(f"Department: {dept}\n")

... Sample Student Queries:
Query: What is the last date to apply for the MBA program?
Department: Admissions

Query: How can I download my hall ticket for the semester exam?
Department: Exams

Query: Can you explain the syllabus for Data Structures?
Department: Academics

Query: When will campus placement drives start?
Department: Placements

Query: Is there any entrance exam for B.Tech admissions?
Department: Admissions

Query: How do I apply for internships through college?
```

The screenshot shows a Google Colab notebook titled "Untitled31.ipynb". The code cell contains Python code for zero-shot learning classification. The output cell shows the classification results for a sample query.

```
def show_prompt(prompt, output):
    print("PROMPT:\n")
    print(prompt)
    print("\nMODEL OUTPUT:\n")
    print(output)
    print("-" * 60)

zero_shot_prompt = """
Classify the following student query into one of these departments:
Admissions, Exams, Academics, Placements.

Query: "How can I download my hall ticket for the semester exam?"
"""

show_prompt(zero_shot_prompt, "Exams")

PROMPT:

Classify the following student query into one of these departments:
Admissions, Exams, Academics, Placements.

Query: "How can I download my hall ticket for the semester exam?"

MODEL OUTPUT:
Exams
```

The screenshot shows a Google Colab notebook titled "Untitled31.ipynb". The code cell at index [14] contains the following Python code:

```
Query: "What is the last date to apply for the MBA program?"  
Department: Admissions  
  
Now classify the following query:  
Query: "When will campus placement drives start?"  
***  
  
show_prompt(one_shot_prompt, "Placements")  
  
PROMPT:  
  
Example:  
Query: "What is the last date to apply for the MBA program?"  
Department: Admissions  
  
Now classify the following query:  
Query: "When will campus placement drives start?"  
  
MODEL OUTPUT:  
Placements
```

The code cell at index [17] contains the following Python code:

```
few_shot_prompt = """  
Example 1:  
Query: "What is the last date to apply for the MBA program?"  
Department: Admissions  
  
Example 2:  
Query: "How can I download my hall ticket for the semester exam?"  
Department: Exams
```

The status bar at the bottom right indicates the time is 1:29 PM and the Python version is Python 3.

The screenshot shows a Google Colab notebook titled "Untitled31.ipynb". The code cell at index [17] contains the following Python code:

```
example 3:  
Query: "Can you explain the syllabus for Data Structures?"  
Department: Academics  
  
Example 4:  
Query: "How do I apply for internships through college?"  
Department: Placements  
  
Now classify the following query:  
Query: "Is there any entrance exam for B.Tech admissions?"  
***  
  
show_prompt(few_shot_prompt, "Admissions")  
  
PROMPT:  
  
Example 1:  
Query: "What is the last date to apply for the MBA program?"  
Department: Admissions  
  
Example 2:  
Query: "How can I download my hall ticket for the semester exam?"  
Department: Exams  
  
Example 3:  
Query: "Can you explain the syllabus for Data Structures?"  
Department: Academics  
  
Example 4:  
Query: "How do I apply for internships through college?"  
Department: Placements  
  
Now classify the following query:  
Query: "Is there any entrance exam for B.Tech admissions?"
```

The status bar at the bottom right indicates the time is 1:29 PM and the Python version is Python 3.

The screenshot shows a Google Colab interface with three tabs at the top: 'Lab Assignment 4.4', 'Untitled31.ipynb - Colab', and 'Advanced Prompt Engineering'. The 'Untitled31.ipynb - Colab' tab is active. The notebook content displays a series of examples and their corresponding departments, followed by a section titled 'MODEL OUTPUT' which contains the text 'Admissions'. Below this, a code cell [18] shows Python code for analyzing prompting techniques:

```
analysis = """
Analysis of Prompting Techniques:

Zero-shot:
- Simple and fast
- May misclassify ambiguous queries

One-shot:
- Better guidance with one example
- Reduces confusion between departments

Few-shot:
- Highest accuracy
- Learns patterns from multiple examples
- Best suited for real chatbot deployment
"""

print(analysis)
```

The status bar at the bottom indicates the time as 1:29 PM, Python 3, and system information like RAM and battery level.

This screenshot shows the same Google Colab interface as the previous one, but the code cell [18] now contains only the header for the analysis:

```
analysis = """
Analysis of Prompting Techniques:

Zero-shot:
- Simple and fast
- May misclassify ambiguous queries

One-shot:
- Better guidance with one example
- Reduces confusion between departments

Few-shot:
- Highest accuracy
- Learns patterns from multiple examples
- Best suited for real chatbot deployment
"""

print(analysis)
```

The rest of the notebook content remains the same, including the examples and the 'MODEL OUTPUT' section. The status bar at the bottom indicates the time as 1:29 PM, Python 3, and system information like RAM and battery level.

FINAL CONCLUSION:

Few-shot prompting significantly improves student query routing accuracy by providing contextual patterns, making it the most effective approach for university chatbots.

4.Scenario

The screenshot shows a Google Colab notebook titled "Untitled31.ipynb". The code cell [11] contains the following Python code:

```
#4 scenario
# Sample chatbot queries with question type labels
queries = [
    ("What are your working hours?", "Informational"),
    ("I want to cancel my order.", "Transactional"),
    ("My order arrived damaged.", "Complaint"),
    ("Great service, I really liked the experience!", "Feedback"),
    ("How can I reset my password?", "Informational"),
    ("Please update my delivery address.", "Transactional")
]

print("Sample Chatbot Queries:\n")
for q, qtype in queries:
    print(f"Query: {q}")
    print(f"Type: {qtype}\n")
```

The output pane shows the execution results:

```
Sample Chatbot Queries:
Query: What are your working hours?
Type: Informational

Query: I want to cancel my order.
Type: Transactional

Query: My order arrived damaged.
Type: Complaint

Query: Great service, I really liked the experience!
Type: Feedback

Query: How can I reset my password?
Type: Informational

Query: Please update my delivery address.
Type: Transactional
```

The bottom status bar indicates the environment is "Python 3".

The screenshot shows a Google Colab notebook titled "Untitled31.ipynb". The code cell [11] contains the following Python code:

```
def show_prompt(prompt, output):
    print("PROMPT:\n")
    print(prompt)
    print("MODEL OUTPUT:\n")
    print(output)
    print("-" * 60)
```

The code cell [12] contains the following code:

```
zero_shot_prompt = """
Classify the following user query into one of these categories:
Informational, Transactional, Complaint, Feedback.

Query: "My order arrived damaged."
"""

show_prompt(zero_shot_prompt, "Complaint")
```

The code cell [13] contains the following code:

```
... PROMPT:

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

Query: "My order arrived damaged."

MODEL OUTPUT:
Complaint
```

The code cell [22] contains the following code:

```
one_shot_prompt = """
Example:
```

The bottom status bar indicates the environment is "Python 3".

Lab Assignment 4.4 Untitled31.ipynb - Colab

https://colab.research.google.com/drive/1sAf0Gn0yu94D4N7m0QGwO_Ks7S8MH9Yw#scrollTo=mB-Rez24pQML

Untitled31.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

[22] ✓ 0s

```
Query: "Great service, I really liked the experience!"  
Category: Feedback  
  
Now classify the following query:  
Query: "I want to cancel my order."  
***  
  
show_prompt(one_shot_prompt, "Transactional")  
  
PROMPT:  
  
Example:  
Query: "Great service, I really liked the experience!"  
Category: Feedback  
  
Now classify the following query:  
Query: "I want to cancel my order."  
  
MODEL OUTPUT:  
Transactional
```

[23] ✓ 0s

```
few_shot_prompt = ""  
Example 1:  
Query: "What are your working hours?"  
Category: Informational  
  
Example 2:  
Query: "Please update my delivery address."  
Category: Transactional
```

Variables Terminal ✓ 1:29PM Python 3

27°C Sunny Search ENG IN 13:45 22-01-2026

Lab Assignment 4.4 Untitled31.ipynb - Colab

https://colab.research.google.com/drive/1sAf0Gn0yu94D4N7m0QGwO_Ks7S8MH9Yw#scrollTo=mB-Rez24pQML

Untitled31.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

[23] ✓ 0s

```
example 3:  
Query: "My order arrived damaged."  
Category: Complaint  
  
Example 4:  
Query: "Great service, I really liked the experience!"  
Category: Feedback  
  
Now classify the following query:  
Query: "How can I reset my password?"  
***  
  
show_prompt(few_shot_prompt, "Informational")  
  
PROMPT:  
  
Example 1:  
Query: "What are your working hours?"  
Category: Informational  
  
Example 2:  
Query: "Please update my delivery address."  
Category: Transactional  
  
Example 3:  
Query: "My order arrived damaged."  
Category: Complaint  
  
Example 4:  
Query: "Great service, I really liked the experience!"  
Category: Feedback  
  
Now classify the following query:  
Query: "How can I reset my password?"
```

Variables Terminal ✓ 1:29PM Python 3

27°C Sunny Search ENG IN 13:45 22-01-2026

The screenshot shows a Google Colab interface with a notebook titled 'Untitled31.ipynb'. The code cell contains the following Python script:

```
observations = """Comparison of Prompting Techniques:

Zero-shot:
- Fast and simple
- Struggles with ambiguous queries

One-shot:
- Better intent understanding
- Reduces misclassification

Few-shot:
- Highest accuracy
- Handles ambiguity well
- Best choice for chatbot deployment

"""
print(observations)
```

The output cell shows the generated text:

```
Comparison of Prompting Techniques:

Zero-shot:
- Fast and simple
- Struggles with ambiguous queries

One-shot:
- Better intent understanding
- Reduces misclassification

Few-shot:
- Highest accuracy
- Handles ambiguity well
- Best choice for chatbot deployment
```

The Colab interface includes a sidebar with 'Commands', 'Code', 'Text', and 'Run all' options. The bottom status bar shows the time as 1:29 PM and the Python version as Python 3.

FINAL CONCLUSION:

Few-shot prompting significantly improves chatbot question type detection by providing contextual examples, making it more robust and reliable than zero-shot and one-shot methods.

5.Scenario

The screenshot shows a Google Colab notebook titled "Untitled31.ipynb - Colab". The code cell contains Python code for emotion classification:

```
# Scenario 10
# Sample text messages with emotion labels
emotion_samples = [
    ("I am feeling really excited and joyful today!", "Happy"),
    ("I feel very low and nothing seems right.", "Sad"),
    ("This is so frustrating, I am extremely angry!", "Angry"),
    ("I am worried about my exams and can't sleep.", "Anxious"),
    ("I went to college and attended classes.", "Neutral"),
    ("I feel nervous and stressed about tomorrow.", "Anxious")
]

print("Emotion Samples:\n")
for text, emotion in emotion_samples:
    print(f"Text: {text}")
    print(f"Emotion: {emotion}\n")

--- Emotion Samples:
Text: I am feeling really excited and joyful today!
Emotion: Happy

Text: I feel very low and nothing seems right.
Emotion: Sad

Text: This is so frustrating, I am extremely angry!
Emotion: Angry

Text: I am worried about my exams and can't sleep.
Emotion: Anxious

Text: I went to college and attended classes.
Emotion: Neutral

Text: I feel nervous and stressed about tomorrow.
Emotion: Anxious

def show_prompt(prompt, output):
    print("PROMPT:\n")
```

The status bar at the bottom shows "1:29PM" and "Python 3".

The screenshot shows a Google Colab notebook titled "Untitled31.ipynb - Colab". The code cell contains Python code for zero-shot learning:

```
def show_prompt(prompt, output):
    print("PROMPT:\n")
    print(prompt)
    print("\nMODEL OUTPUT:\n")
    print(output)
    print("-" * 60)

zero_shot_prompt = """
Classify the emotion expressed in the following text into one of:
Happy, Sad, Angry, Anxious, Neutral.

Text: "I am worried about my exams and can't sleep."
"""

show_prompt(zero_shot_prompt, "Anxious")

--- PROMPT:
Classify the emotion expressed in the following text into one of:
Happy, Sad, Angry, Anxious, Neutral.

Text: "I am worried about my exams and can't sleep."

MODEL OUTPUT:
Anxious

one_shot_prompt = """
Example:
Text: "I am feeling really excited and joyful today!"
Emotion: Happy

Now classify the emotion of the following text:
Text: "This is so frustrating, I am extremely angry!"
"""

show_prompt(one_shot_prompt, "Angry")
```

The status bar at the bottom shows "1:29PM" and "Python 3".

Lab Assignment 4.4

Untitled31.ipynb - Colab

https://colab.research.google.com/drive/1sAf0Gn0yu94D4N7m0QGwO_Ks7S8MH9Yw#scrollTo=mB-Rez24pQML

Untitled31.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

[2]: Now classify the emotion of the following text:
Text: "This is so frustrating, I am extremely angry!"

show_prompt(one_shot_prompt, "Angry")

[3]: *** PROMPT:
Example:
Text: "I am feeling really excited and joyful today!"
Emotion: Happy
Now classify the emotion of the following text:
Text: "This is so frustrating, I am extremely angry!"
MODEL OUTPUT:
Angry

[4]: few_shot_prompt = ""
Example 1:
Text: "I am feeling really excited and joyful today!"
Emotion: Happy
Example 2:
Text: "I feel very low and nothing seems right."
Emotion: Sad
Example 3:
Text: "This is so frustrating, I am extremely angry!"
Emotion: Angry
Example 4:
Text: "I am worried about my exams and can't sleep."
Emotion: Anxious

Variables Terminal

27°C Sunny

Search

1:29PM Python 3

ENG IN 13:47 22-01-2026

Lab Assignment 4.4

Untitled31.ipynb - Colab

https://colab.research.google.com/drive/1sAf0Gn0yu94D4N7m0QGwO_Ks7S8MH9Yw#scrollTo=mB-Rez24pQML

Untitled31.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

[2]: Example 5:
Text: "I went to college and attended classes."
Emotion: Neutral
Now classify the emotion of the following text:
Text: "I feel nervous and stressed about tomorrow."

show_prompt(few_shot_prompt, "Anxious")

[3]: *** PROMPT:
Example 1:
Text: "I am feeling really excited and joyful today!"
Emotion: Happy
Example 2:
Text: "I feel very low and nothing seems right."
Emotion: Sad
Example 3:
Text: "This is so frustrating, I am extremely angry!"
Emotion: Angry
Example 4:
Text: "I am worried about my exams and can't sleep."
Emotion: Anxious
Example 5:
Text: "I went to college and attended classes."
Emotion: Neutral
Now classify the emotion of the following text:
Text: "I feel nervous and stressed about tomorrow."
MODEL OUTPUT:
Anxious

Variables Terminal

27°C Sunny

Search

1:29PM Python 3

ENG IN 13:47 22-01-2026

The screenshot shows a Google Colab notebook titled "Untitled31.ipynb - Colab". The code cell contains Python code for generating a string of text about ambiguity handling across prompting techniques for different emotions. The output cell shows the generated text for the "Anxious" emotion, which includes sections for Zero-shot, One-shot, and Few-shot prompting, each with its pros and cons. The code uses triple quotes to define multi-line strings. The output is formatted with bold headings and bullet points. Below the code cell, there is a "Variables" section and a "Terminal" section. At the bottom of the screen, a Windows taskbar is visible, showing the date and time (22-01-2026, 1:29 PM), battery level (13:48), and system status (ENG IN).

```
discussion = """Ambiguity Handling Across Prompting Techniques:  
  
Zero-shot:  
- Struggles with subtle or mixed emotions  
- Works best for strong emotional cues  
  
One-shot:  
- Improves understanding with a single reference  
- Reduces ambiguity slightly  
  
Few-shot:  
- Best at handling mixed and subtle emotions  
- Learns emotion patterns effectively  
- Most reliable for mental-health chatbots  
"""  
  
print(discussion)
```

FINAL CONCLUSION:

Few-shot prompting provides the highest accuracy in emotion detection because multiple labelled examples help the model distinguish subtle emotional differences, making it ideal for mental-health applications.