

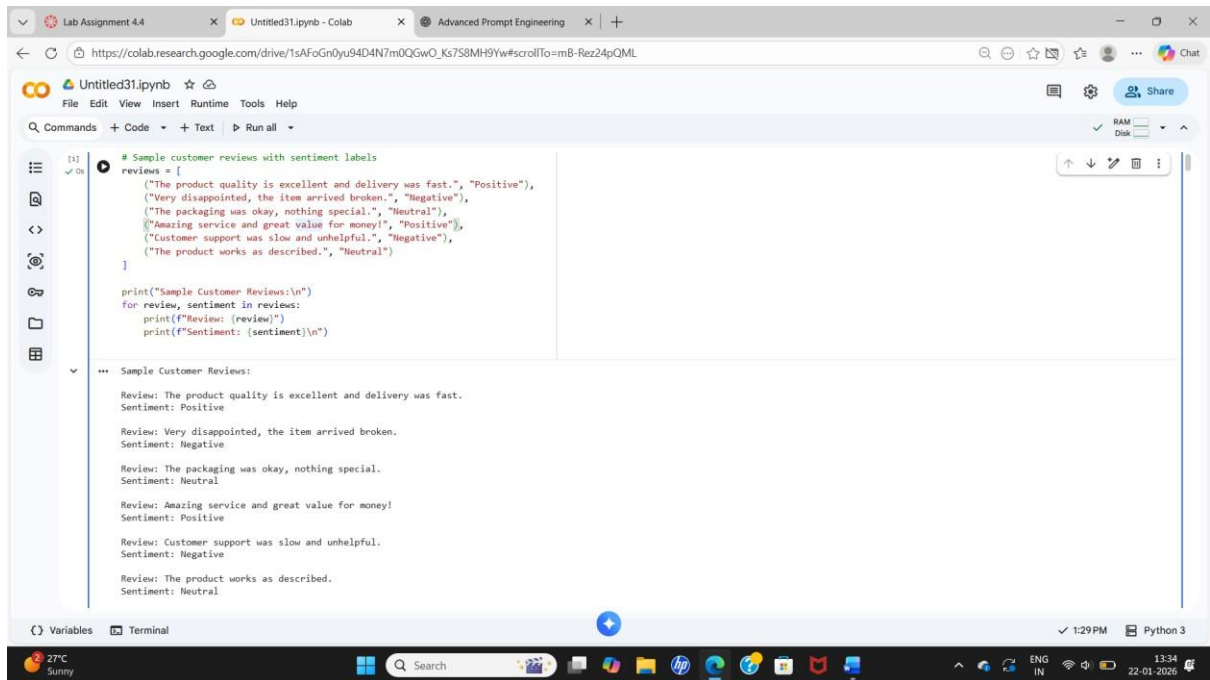
LAB ASSIGNMENT 4.4

NAME: A.Harshita

ID NO: 2303A52211

SUBJECT: AI ASST CODING

1.Scenario



The screenshot shows a Google Colab notebook titled 'Untitled31.ipynb'. The code defines a list of reviews with their corresponding sentiment labels. The output displays each review and its sentiment.

```
[1] ✓ On # 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}")
    print(f"Sentiment: {sentiment}\n")
```

Sample Customer Reviews:

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

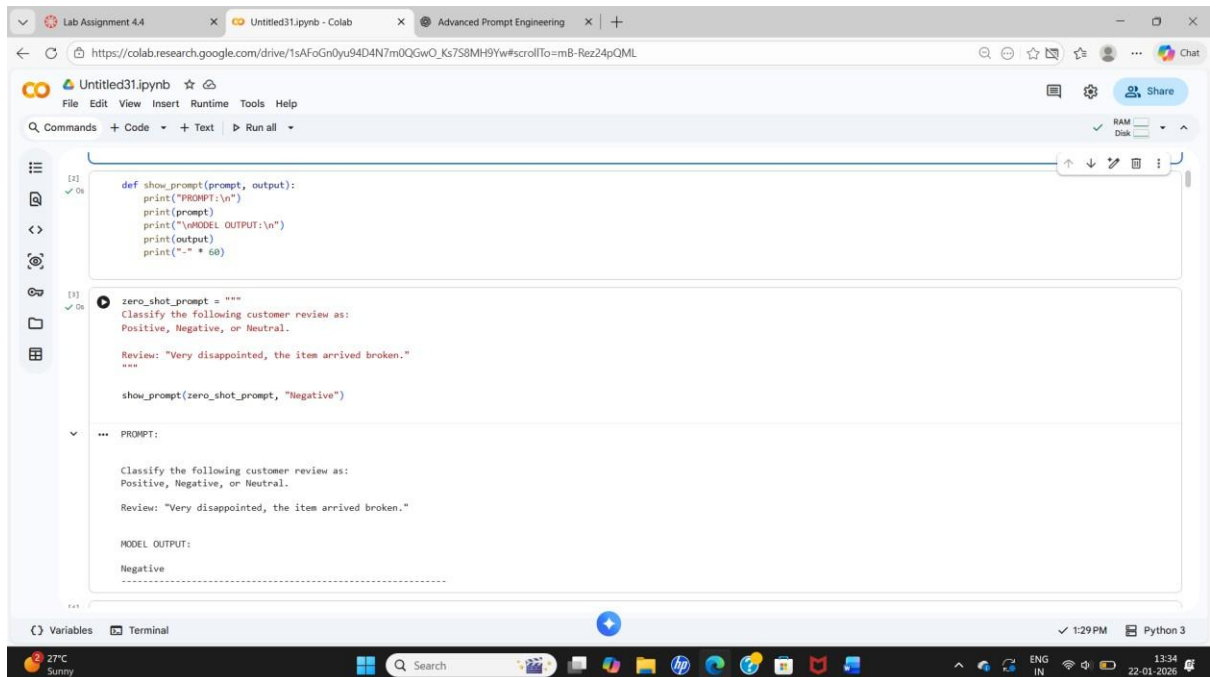
Review: Very disappointed, the item arrived broken.
Sentiment: Negative

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'. The code defines a function to show a prompt and its output. It then uses a zero-shot prompt to classify a customer review as Positive, Negative, or Neutral. The output shows the prompt and the model's output.

```
[2] ✓ On def show_prompt(prompt, output):
print("PROMPT:\n")
print(prompt)
print("\nMODEL OUTPUT:\n")
print(output)
print("-" * 60)

[3] ✓ On 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
-----
```

Lab Assignment 4.4

Untitled31.ipynb - Colab

Advanced Prompt Engineering

+

https://colab.research.google.com/drive/1sAfoGn0yu94D4N7m0QGwO_Ks7S8MH9Yw#scrollTo=mB-Rez24pQML

Colab

Untitled31.ipynb

File Edit View Insert Runtime Tools Help

Commands Code Text Run all

[4] ✓ Os

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
.....

[5] ✓ Os

few_shot_prompt = ""
Example 1:
Review: "Amazing service and great value for money!"
Sentiment: Positive

Example 2:

Variables Terminal

1:29 PM Python 3

27°C Sunny

Search

hp

ENG IN

13:34

22-01-2026

Lab Assignment 4.4

Untitled31.ipynb - Colab

Advanced Prompt Engineering

+

https://colab.research.google.com/drive/1sAfoGn0yu94D4N7m0QGwO_Ks7S8MH9Yw#scrollTo=mB-Rez24pQML

Colab

Untitled31.ipynb

File Edit View Insert Runtime Tools Help

Commands Code Text Run all

[5] ✓ Os

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:

Variables Terminal

1:29 PM Python 3

27°C Sunny

Search

hp

ENG IN

13:34

22-01-2026

```
[4] ✓ On
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

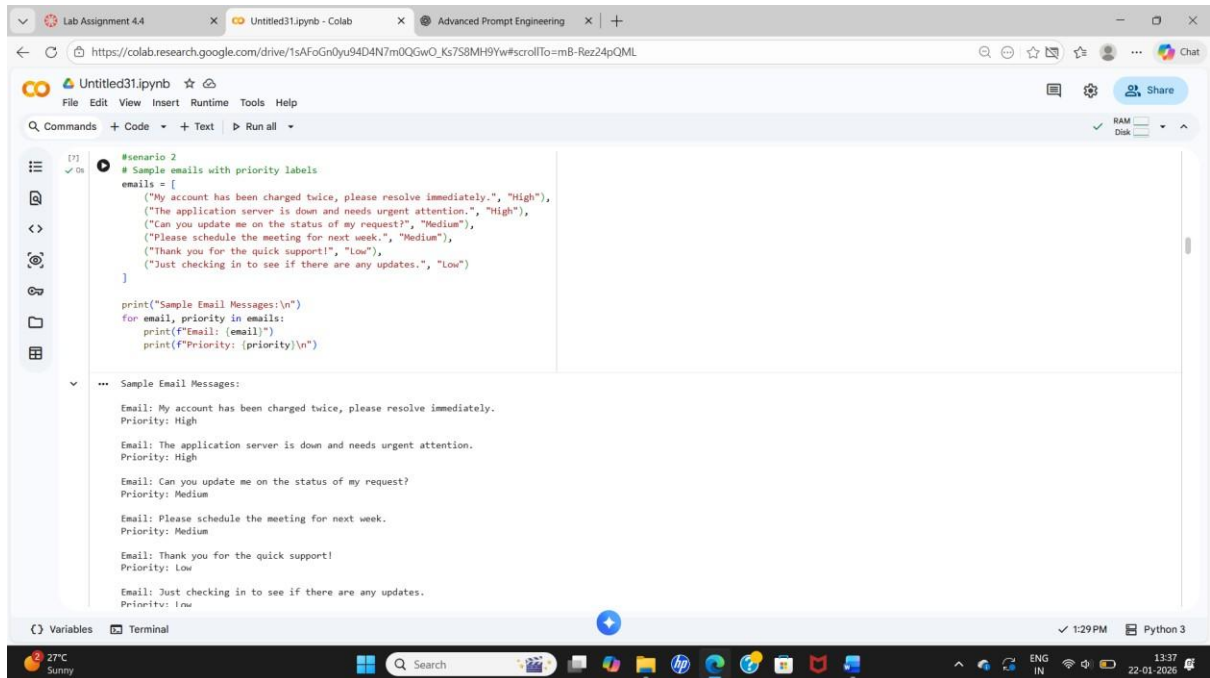
One-shot:
- Better understanding of sentiment boundaries
- More consistent than zero-shot

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 list of sample email messages with their priorities and prints them out. The output shows the email messages and their corresponding priorities.

```
[7] ✓ On #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:\n")
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

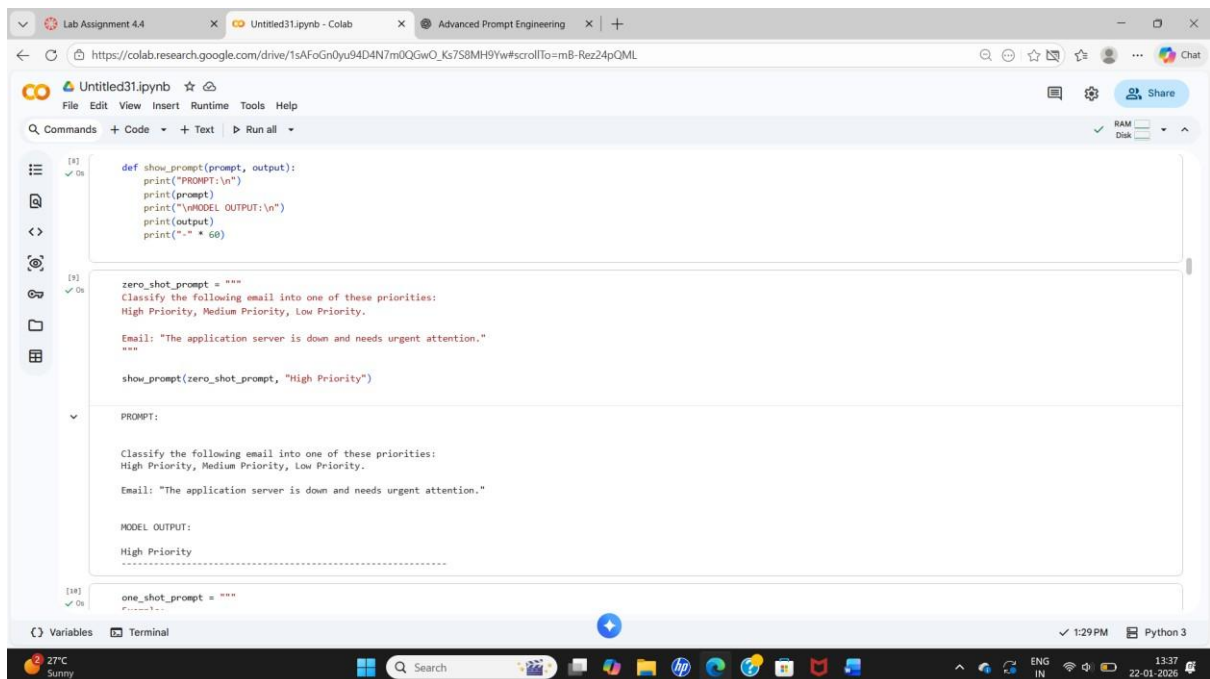
Email: The application server is down and needs urgent attention.
Priority: High

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, and then uses it to demonstrate zero-shot and one-shot prompting for email classification.

```
[8] ✓ On def show_prompt(prompt, output):
    print("PROMPT:\n")
    print(prompt)
    print("\nMODEL OUTPUT:\n")
    print(output)
    print("-" * 60)

[9] ✓ On 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
-----

[10] ✓ On one_shot_prompt = """
```

Lab Assignment 4.4

Untitled31.ipynb - Colab

Advanced Prompt Engineering

+

https://colab.research.google.com/drive/1sAfoGn0yu94D4N7m0QGwQ_Ks7S8MH9Yw#scrollTo=mB-Rez24pQML

Untitled31.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▶ Run all

138

✓ On

```
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

111

✓ On

```
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

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

show_prompt(few_shot_prompt, "Low Priority")
```

Variables

Terminal

27°C Sunny

Search

hp

ENG IN

13:37

22-01-2026

Lab Assignment 4.4

Untitled31.ipynb - Colab

Advanced Prompt Engineering

+

https://colab.research.google.com/drive/1sAfoGn0yu94D4N7m0QGwQ_Ks7S8MH9Yw#scrollTo=mB-Rez24pQML

Untitled31.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▶ Run all

111

✓ On

```
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:

Variables

Terminal

27°C Sunny

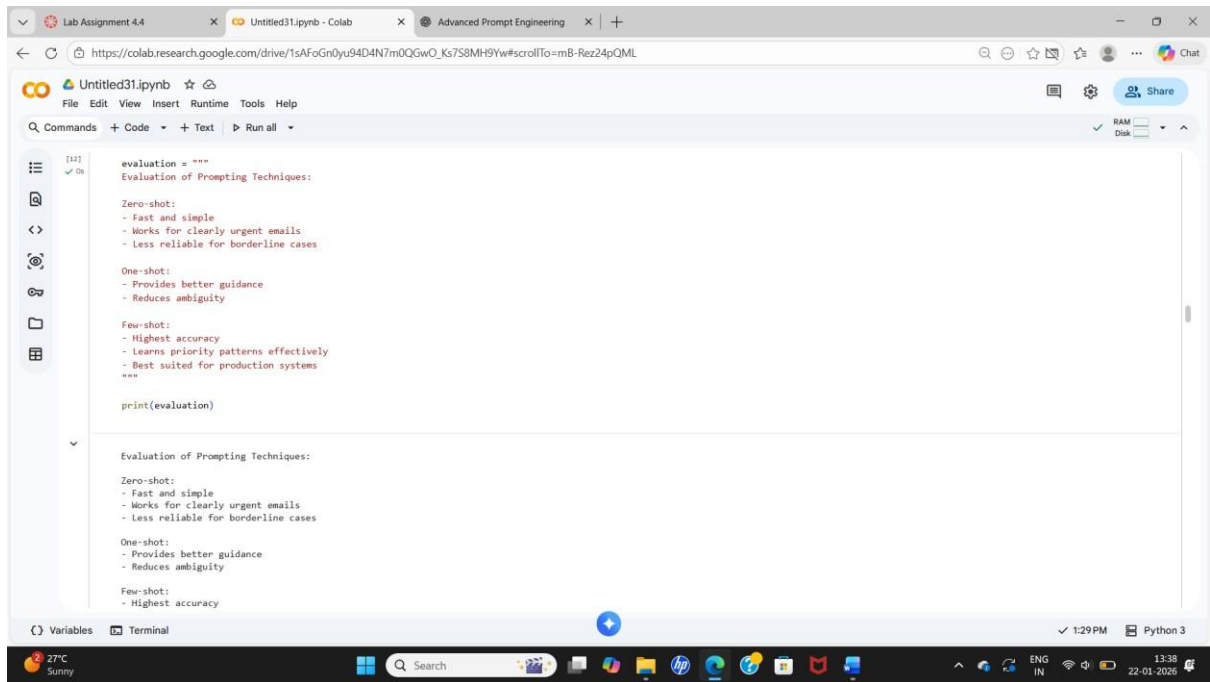
Search

hp

ENG IN

13:37

22-01-2026



```
[12]: 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

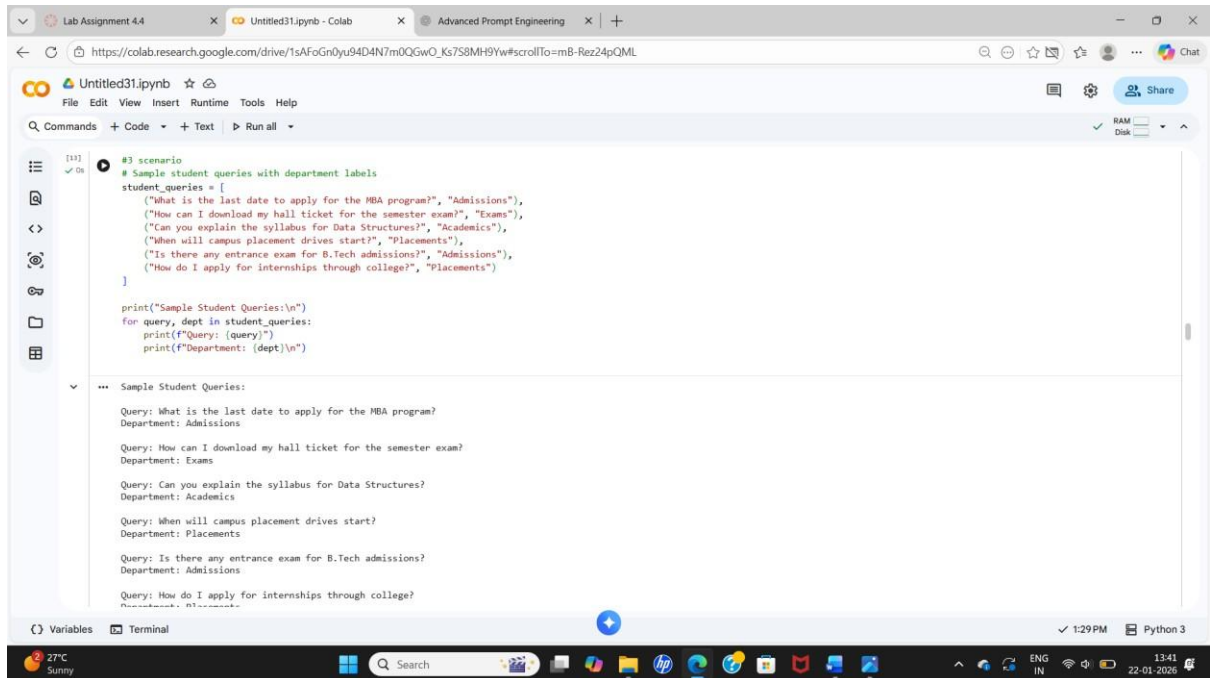
Few-shot:

- Highest accuracy
- Learns priority patterns effectively
- Best suited for production systems

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 defines a list of sample student queries with department labels. The output displays these queries and their assigned departments.

```
[11]: #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

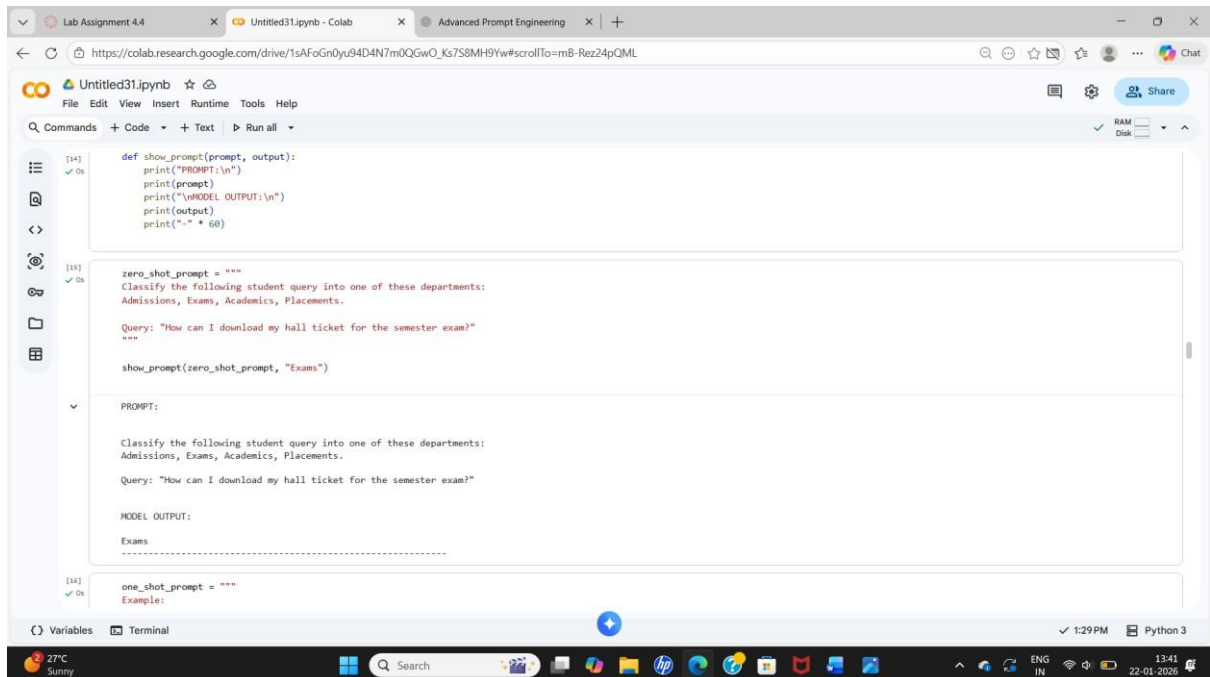
Query: How can I download my hall ticket for the semester exam?
Department: Exams

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?
Department: Placements



The screenshot shows a Google Colab notebook titled 'Untitled31.ipynb'. The code defines a function to show prompts and a zero-shot prompt for classifying student queries into departments. The output shows the prompt and the model's output.

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

[15]: 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
-----

[16]: one_shot_prompt = """
Example:
```

Lab Assignment 4.4 x Untitled31.ipynb - Colab x Advanced Prompt Engineering x +

https://colab.research.google.com/drive/1sAf0Gn0yu94D4N7m0QGwQ_Ks7S8MH9Yw#scrollTo=mB-Rez24pQML

Untitled31.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text + Run all

[14] ✓ On

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
.....

[17] ✓ On

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

Variables Terminal

1:29 PM Python 3

27°C Sunny

Lab Assignment 4.4 x Untitled31.ipynb - Colab x Advanced Prompt Engineering x +

https://colab.research.google.com/drive/1sAf0Gn0yu94D4N7m0QGwQ_Ks7S8MH9Yw#scrollTo=mB-Rez24pQML

Untitled31.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text + Run all

[17] ✓ On

Example 2:
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?"

Variables Terminal

1:29 PM Python 3

27°C Sunny

The screenshot shows a Google Colab notebook titled 'Untitled31.ipynb'. The code defines a dictionary of departments and their associated queries, then classifies a new query based on these patterns. The output shows the classification result as 'Admissions'.

```
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?"

MODEL OUTPUT:
Admissions
.....

[11]: 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
"""
```

This screenshot shows the same Google Colab notebook, but now the code has been executed. The output of the analysis is displayed below the code cell, showing the same text as the 'MODEL OUTPUT' in the first image.

```
.....

[11]: 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)

Analysis of Prompting Techniques:

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

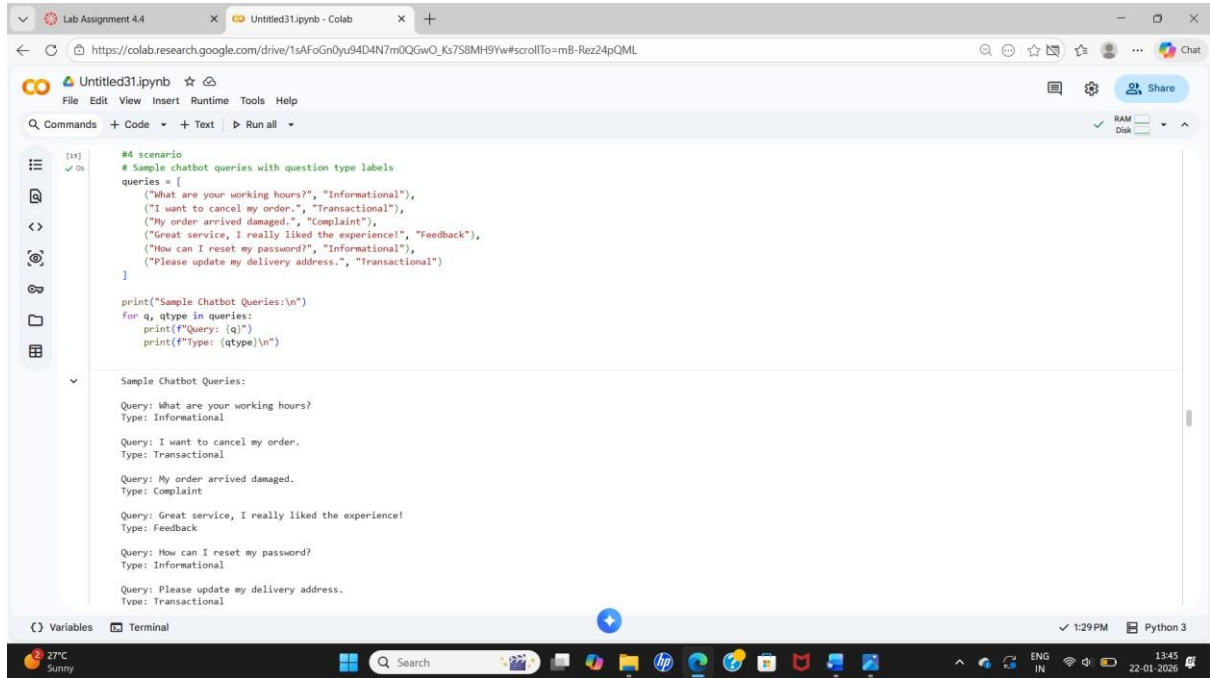
One-shot:
- Better guidance with one example
- Reduces confusion between departments

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

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 defines a list of sample chatbot queries with their corresponding question type labels. The output displays these queries and their types.

```
#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")
```

Sample Chatbot Queries:

Query: What are your working hours?
Type: Informational

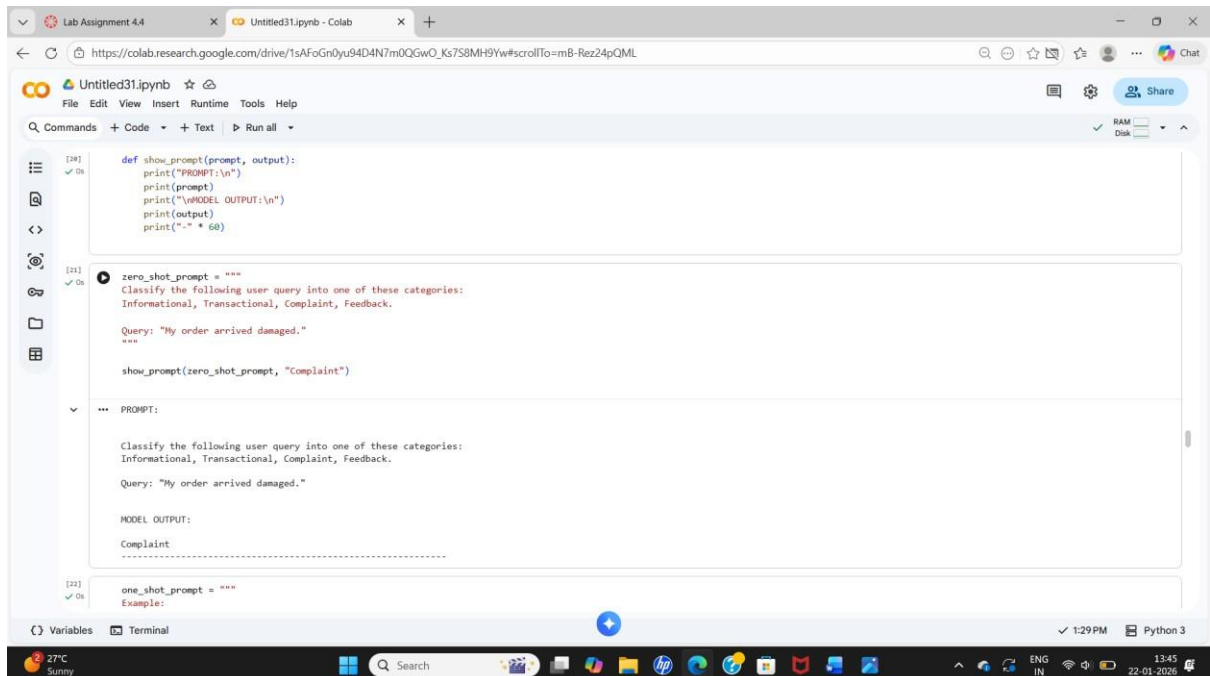
Query: I want to cancel my order.
Type: Transactional

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 screenshot shows a Google Colab notebook titled 'Untitled31.ipynb'. The code defines a function to show a prompt and output, and then uses a zero-shot prompt to classify a user query into one of the categories: Informational, Transactional, Complaint, Feedback.

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

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")

""" PROMPT:

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

Query: "My order arrived damaged."

MODEL OUTPUT:

Complaint
"""

one_shot_prompt = """
Example:
```

Lab Assignment 4.4 x Untitled31.ipynb - Colab

https://colab.research.google.com/drive/1sAfGn0yu94D4N7m0QGwO_Ks7S8MH9Yw#scrollTo=mB-Rez24pQML

Untitled31.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```
[22] ✓ On
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] ✓ On
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

1:29 PM Python 3

27°C Sunny

Lab Assignment 4.4 x Untitled31.ipynb - Colab

https://colab.research.google.com/drive/1sAfGn0yu94D4N7m0QGwO_Ks7S8MH9Yw#scrollTo=mB-Rez24pQML

Untitled31.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```
[23] ✓ On
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

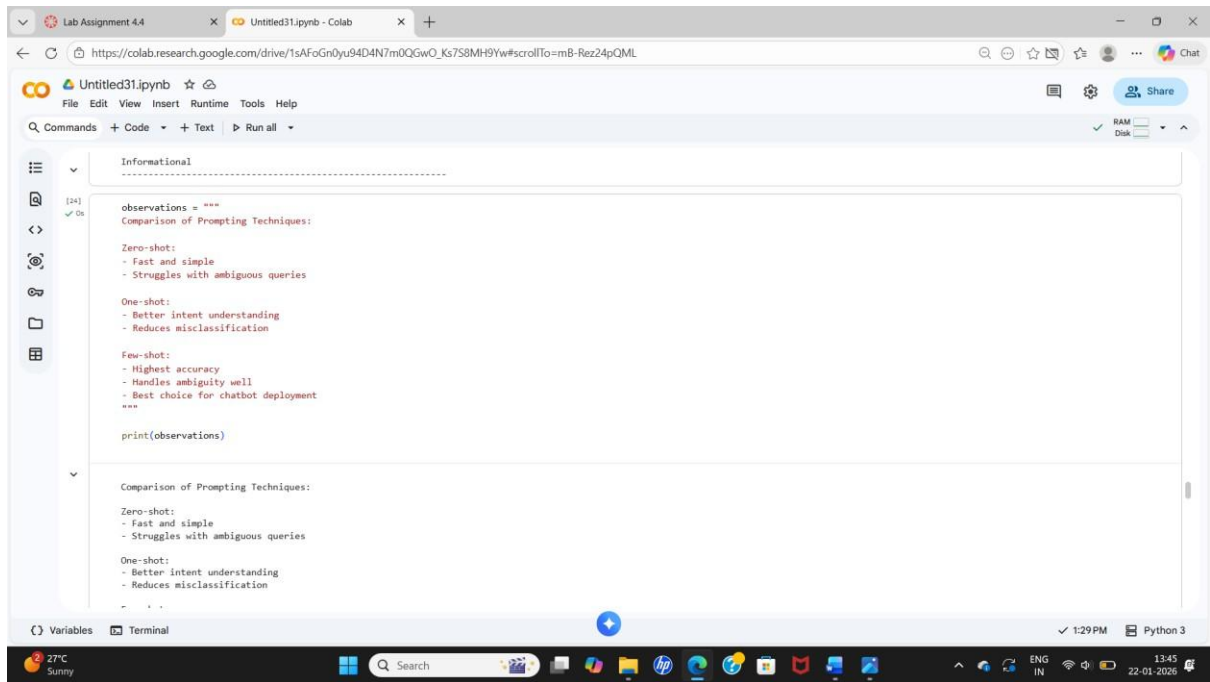
Example 4:
Query: "Great service, I really liked the experience!"
Category: Feedback

Now classify the following query:
Query: "How can I reset my password?"
```

Variables Terminal

1:29 PM Python 3

27°C Sunny



```
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)
```

```
Comparison of Prompting Techniques:

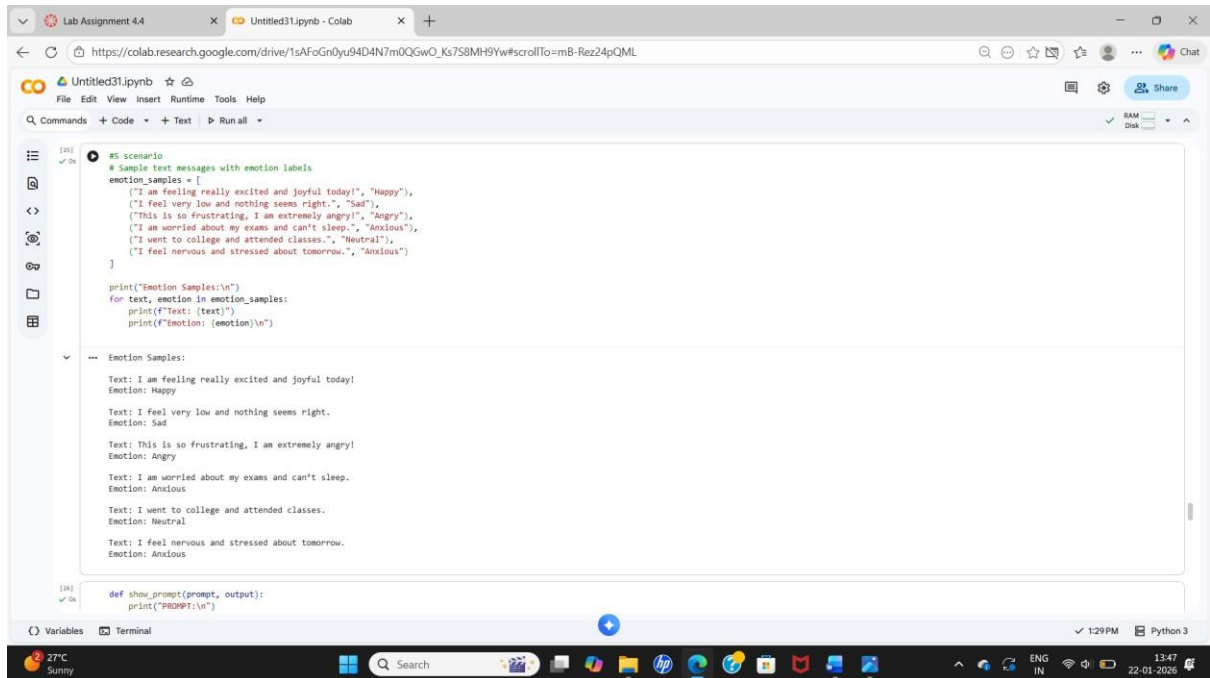
Zero-shot:
- Fast and simple
- Struggles with ambiguous queries

One-shot:
- Better intent understanding
- Reduces misclassification
```

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



```
[1]: #5 scenario
# 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

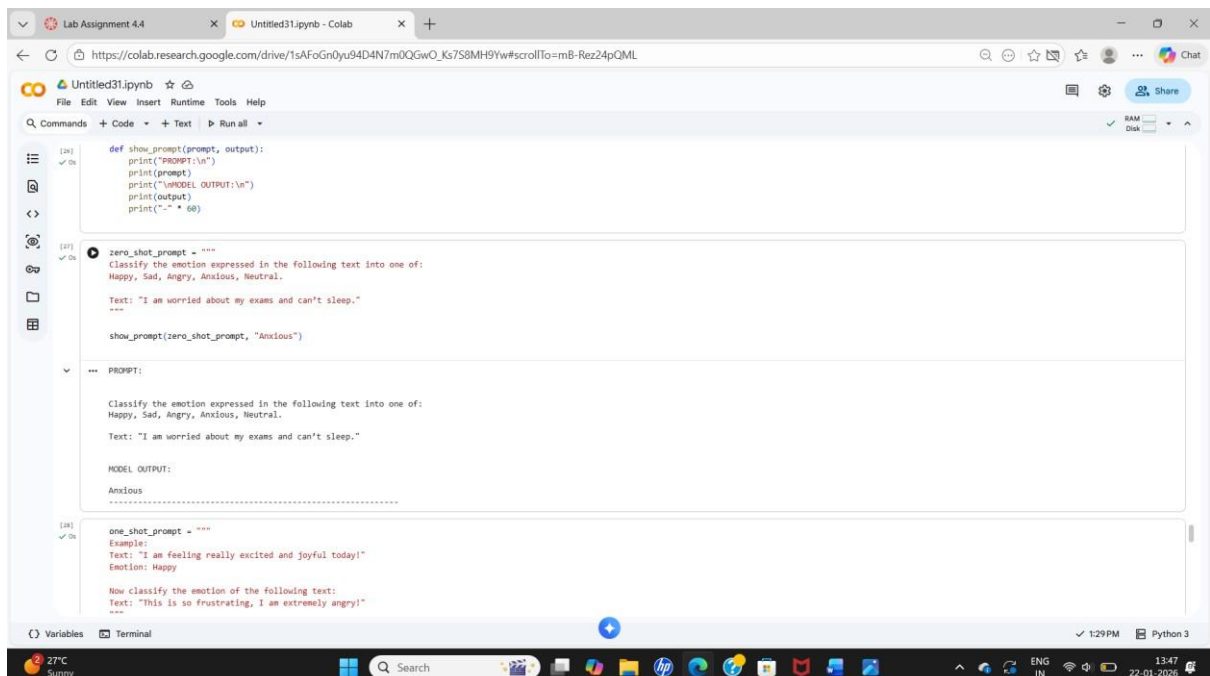
Text: This is so frustrating, I am extremely angry!
Emotion: Angry

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

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



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

```
[4]: 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

```
[5]: 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!"
"""
```

Lab Assignment 4.4 x Untitled31.ipynb - Colab x +

https://colab.research.google.com/drive/1sAfoGn0yu94D4N7m0QGwQ_Ks7S8MH9Yw#scrollTo=mB-Rez24pQML

Untitled31.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text + Run all

RAM Disk

[24] ✓ 0s

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

show_prompt(one_shot_prompt, "Angry")

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

+ Code + Text

[25] ✓ 0s

```
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

1:29 PM Python 3

27°C Sunny

Lab Assignment 4.4 x Untitled31.ipynb - Colab x +

https://colab.research.google.com/drive/1sAfoGn0yu94D4N7m0QGwQ_Ks7S8MH9Yw#scrollTo=mB-Rez24pQML

Untitled31.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text + Run all

RAM Disk

[26] ✓ 0s

```
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")

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

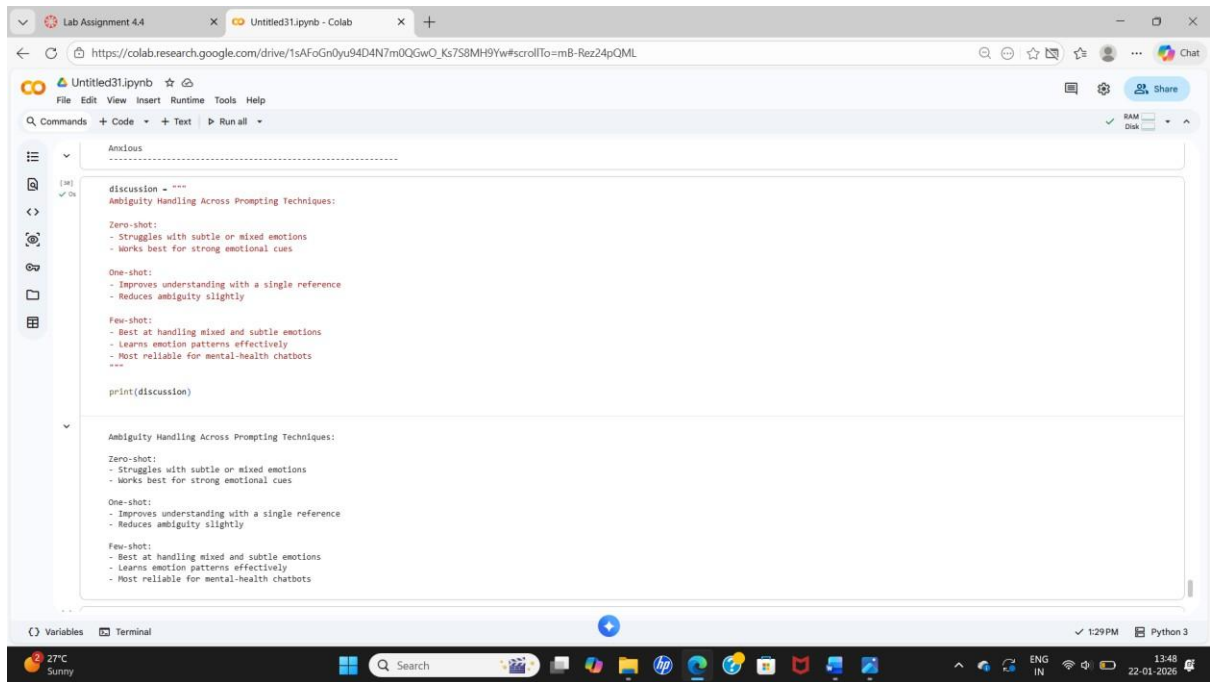
Now classify the emotion of the following text:
Text: "I feel nervous and stressed about tomorrow."

MODEL OUTPUT:
Anxious

Variables Terminal

1:29 PM Python 3

27°C Sunny



The screenshot shows a Google Colab notebook interface. The browser address bar displays the URL: https://colab.research.google.com/drive/1sAfoGn0yu94D4N7m0QGwQ_Ks7S8MH9Yw#scrollTo=mB-Rez24pQML. The notebook is titled 'Untitled31.ipynb'. The code editor contains the following Python code:

```
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)
```

The output of the code is displayed below the code editor, showing the same text as the code:

```
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
```

The bottom of the screen shows a Windows taskbar with the date 22-01-2026 and time 13:48.

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.