

Assignment-4.4

Name:M.Sanjana

H.T.NO:2303A52292

Scenario1: E-commerce

a) 6 short customer reviews mapped to sentiment labels.

The screenshot shows a Google Colab notebook titled "AI assistant 4.4.ipynb". In cell [14], the following Python code is run:

```
# a) 6 short customer reviews with labels
reviews = {
    "The product is amazing": "Positive",
    "I am very happy with the service": "Positive",
    "The item arrived damaged": "Negative",
    "This product does not work": "Negative",
    "The packaging was okay": "Neutral",
    "It works fine, nothing special": "Neutral"
}

for review, label in reviews.items():
    print("Review:", review)
    print("Sentiment:", label)
    print()
```

The output shows the reviews and their corresponding sentiment labels:

```
Review: The product is amazing
Sentiment: Positive
Review: I am very happy with the service
Sentiment: Positive
Review: The item arrived damaged
Sentiment: Negative
Review: This product does not work
Sentiment: Negative
Review: The packaging was okay
Sentiment: Neutral
Review: It works fine, nothing special
Sentiment: Neutral
```

b) Zero shot

The screenshot shows a Google Colab notebook titled "AI assistant 4.4.ipynb". In cell [10], the following Python code is run:

```
# b) ZERO-SHOT
def classify(review):
    zero_shot_prompt = """Classify the sentiment of the following customer review into Positive, Negative, or Neutral.
Review: {review}
Sentiment:"""

    if "amazing" in review.lower() or "fast" in review.lower() or "love" in review.lower() or "fantastic" in review.lower():
        return "Positive"
    elif "terrible" in review.lower() or "disappointing" in review.lower() or "broke" in review.lower() or "late" in review.lower():
        return "Negative"
    else:
        return "Neutral"

print("ZERO-SHOT OUTPUT")
test_review = "The product quality is amazing and delivery was fast"
print("Review:", test_review)
print("Sentiment:", classify(test_review))
print()
```

The output shows the zero-shot classification of the test review:

```
ZERO-SHOT OUTPUT
Review: The product quality is amazing and delivery was fast
Sentiment: Positive
```

One-shot & Few shot

The screenshot shows a Google Colab notebook titled "AI assistant 4.4.ipynb". The code cell [11] contains two examples: one for "ONE-SHOT" and one for "FEW-SHOT".

```
# c) ONE-SHOT
print("ONE-SHOT OUTPUT")
print("Example: The product is terrible -> Negative")
test_review = "Customer service helped me quickly"
print("Review:", test_review)
print("Sentiment:", classify(test_review))
print()

# d) FEW-SHOT
print("FEW-SHOT OUTPUT")
print("Examples:")
print("Good quality -> Positive")
print("Item is broken -> Negative")
print("It's okay -> Neutral")
test_review = "It works fine, but I expected more features"
print("Review:", test_review)
print("Sentiment:", classify(test_review))
```

The output shows the results of the sentiment classification for the provided reviews.

Comparision

The screenshot shows a Google Colab notebook titled "AI assistant 4.4.ipynb". The code cell [16] contains a comparison of Prompting Techniques.

```
# Accuracy
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
```

Scenario2: Email Priority Classification

https://colab.research.google.com/drive/1ozpqcEVB1owfu8RceRZY8NpD5Gk6-yA#scrollTo=pHz78Ge8ym

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```
[21] # 1. Sample emails with priority labels
emails = {
    "Server is down, please fix immediately": "High",
    "Need password reset urgently": "High",
    "Can you send the meeting schedule?": "Medium",
    "Please review my project by tomorrow": "Medium",
    "Thank you for your support": "Low",
    "Just checking in to say hello": "Low"
}
high_words = ["urgent", "immediately", "down", "fix"]
medium_words = ["meeting", "review", "schedule", "tomorrow"]
# Simple classifier
def classify_email(email):
    email = email.lower()

    for word in high_words:
        if word in email:
            return "High Priority"

    for word in medium_words:
        if word in email:
            return "Medium Priority"

    return "Low Priority"

for email in emails:
    print("Email:", email)
    print("Predicted Priority:", classify_email(email))
    print("Actual Priority:", emails[email])
    print()
```

... Email: Server is down, please fix immediately
Predicted Priority: High Priority
Actual Priority: High

Email: Need password reset urgently
Predicted Priority: High Priority
Actual Priority: High

Email: Can you send the meeting schedule?
Predicted Priority: Medium Priority
Actual Priority: Medium

Email: Please review my project by tomorrow
Predicted Priority: Medium Priority
Actual Priority: Medium

Email: Thank you for your support
Predicted Priority: Low Priority
Actual Priority: Low

Email: Just checking in to say hello
Predicted Priority: Low Priority
Actual Priority: Low

Zero shot

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```
[28] # 2. ZERO-SHOT
print("ZERO-SHOT OUTPUT")
test_email = "Server is down, please fix immediately"
print("Email:", test_email)
print("Priority:", classify_email(test_email))
print()
```

ZERO-SHOT OUTPUT
Email: Server is down, please fix immediately
Priority: High Priority

Start coding or generate with AI.

One shot&few shot

The screenshot shows a Jupyter Notebook interface with two code cells. The first cell, titled '# 3. ONE-SHOT (One example shown)', contains Python code to print an email example and its priority. The output shows a single example: 'Email: 'System is down' -> High Priority'. The second cell, titled '# 4. FEW-SHOT (Multiple examples shown)', contains Python code to print multiple email examples and their priorities. The output shows four examples: 'Server crash' -> High Priority, 'Meeting tomorrow' -> Medium Priority, 'Thanks for your help' -> Low Priority, and 'Can you send the meeting schedule?' -> Medium Priority.

```
# 3. ONE-SHOT (One example shown)
print("ONE-SHOT OUTPUT")
print("Example: Email: 'System is down' -> High Priority")
test_email = "Need password reset urgently"
print("Email:", test_email)
print("Priority:", classify_email(test_email))
print()

# 4. FEW-SHOT (Multiple examples shown)
print("FEW-SHOT OUTPUT")
print("Examples:")
print("Email: 'Server crash' -> High Priority")
print("Email: 'Meeting tomorrow' -> Medium Priority")
print("Email: 'Thanks for your help' -> Low Priority")
test_email = "Can you send the meeting schedule?"
print("Email:", test_email)
print("Priority:", classify_email(test_email))
print()
```

Comparision

The screenshot shows a Jupyter Notebook interface with two code cells. The first cell, titled 'evaluation = """', contains Python code defining three prompting techniques: Zero-shot, One-shot, and Few-shot. The output shows the definitions for each. The second cell, also titled 'Evaluation of Prompting Techniques:', contains Python code to print the evaluation string. The output shows the same three prompting techniques again.

```
evaluation = """
Evaluation of Prompting Techniques:
Zero-shot:
- Relies only on instructions
- May misclassify unclear or mixed-priority emails
One-shot:
- Uses one example to guide classification
- More reliable than zero-shot, but still limited
Few-shot:
- Uses multiple labeled examples
- Most reliable and consistent
- Best for handling borderline or ambiguous emails
"""

print(evaluation)

"""
Evaluation of Prompting Techniques:
Zero-shot:
- Relies only on instructions
- May misclassify unclear or mixed-priority emails
One-shot:
- Uses one example to guide classification
- More reliable than zero-shot, but still limited
Few-shot:
- Uses multiple labeled examples
- Most reliable and consistent
- Best for handling borderline or ambiguous emails
```

Scenario3: Student Query Routing System

```
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
queries = [
    "How can I apply for BTech admission?", "Admissions",
    "When will semester results be announced?", "Exams",
    "What is the syllabus for Machine Learning?", "Academics",
    "Which companies are coming for placements?", "Placements",
    "How do I pay my admission fees?", "Admissions",
    "How can I register for exams?", "Exams"
]

admission_words = ["apply", "admission", "fees", "join"]
exam_words = ["exam", "result", "register"]
academic_words = ["syllabus", "subject", "course"]
placement_words = ["company", "placement", "job"]

# Simple classifier
def classify_query(query):
    query = query.lower()
    for word in admission_words:
        if word in query:
            return "Admissions"
    for word in exam_words:
        if word in query:
            return "Exams"
    for word in academic_words:
        if word in query:
            return "Academics"
    for word in placement_words:
        if word in query:
            return "Placements"
    return "Unknown"

print("STUDENT QUERY ROUTING OUTPUT\n")
for q, actual in queries.items():
    predicted = classify_query(q)
    print("Query:", q)
    print("Predicted:", predicted, "| Actual:", actual)
    print()
```

Zero shot&one shot

```
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
# 2. ZERO-SHOT
print("ZERO-SHOT OUTPUT")
test_query = "When will exam results be declared?"
print("Query:", test_query)
print("Department:", classify_query(test_query))
print()

... ZERO-SHOT OUTPUT
Query: When will exam results be declared?
Department: Exams

# 3. ONE-SHOT
print("ONE-SHOT OUTPUT")
print("Example: Query: 'How to apply for college?' -> Admissions")
test_query = "Which companies are coming next week?"
print("Query:", test_query)
print("Department:", classify_query(test_query))
print()

ONE-SHOT OUTPUT
Example: Query: 'How to apply for college?' -> Admissions
Query: Which companies are coming next week?
Department: Unknown
```

Few shot

The screenshot shows a Jupyter Notebook cell with two parts. The first part contains Python code for a few-shot learning example:

```
# 4. FEW-SHOT
print("FEW-SHOT OUTPUT")
print("Examples:")
print("Query: 'Exam registration' -> Exams")
print("Query: 'Syllabus for AI' -> Academics")
print("Query: 'Placement drive info' -> Placements")
test_query = "How do I pay my admission fees?"
print("Query:", test_query)
print("Department:", classify_query(test_query))
print()

... FEW-SHOT OUTPUT
Examples:
Query: 'Exam registration' -> Exams
Query: 'Syllabus for AI' -> Academics
Query: 'Placement drive info' -> Placements
Query: How do I pay my admission fees?
Department: Admissions
```

The second part contains code for an accuracy check:

```
# 5. Accuracy Check
correct = 0
total = len(queries)

print("ACCURACY CHECK")
for q, label in queries.items():
    result = classify_query(q)
    print("Query:", q)
    print("Predicted:", result, "| Actual:", label)
    print()
    if result == label:
        correct += 1
accuracy = (correct / total) * 100
print("Accuracy:", accuracy, "%")
print("\nConclusion: Few-shot > One-shot > Zero-shot in reliability")
```

The screenshot shows a Google Colab cell with two parts. The first part contains Python code for a few-shot learning example:

```
correct = 0
total = len(queries)
print("ACCURACY CHECK")
for q, label in queries.items():
    result = classify_query(q)
    print("Query:", q)
    print("Predicted:", result, "| Actual:", label)
    print()
    if result == label:
        correct += 1
accuracy = (correct / total) * 100
print("Accuracy:", accuracy, "%")
print("\nConclusion: Few-shot > One-shot > Zero-shot in reliability")
```

The second part contains code for an accuracy check:

```
... ACCURACY CHECK
Query: How can I apply for BTech admission?
Predicted: Admissions | Actual: Admissions

Query: When will semester results be announced?
Predicted: Exams | Actual: Exams

Query: What is the syllabus for Machine Learning?
Predicted: Academics | Actual: Academics

Query: Which companies are coming for placements?
Predicted: Placements | Actual: Placements

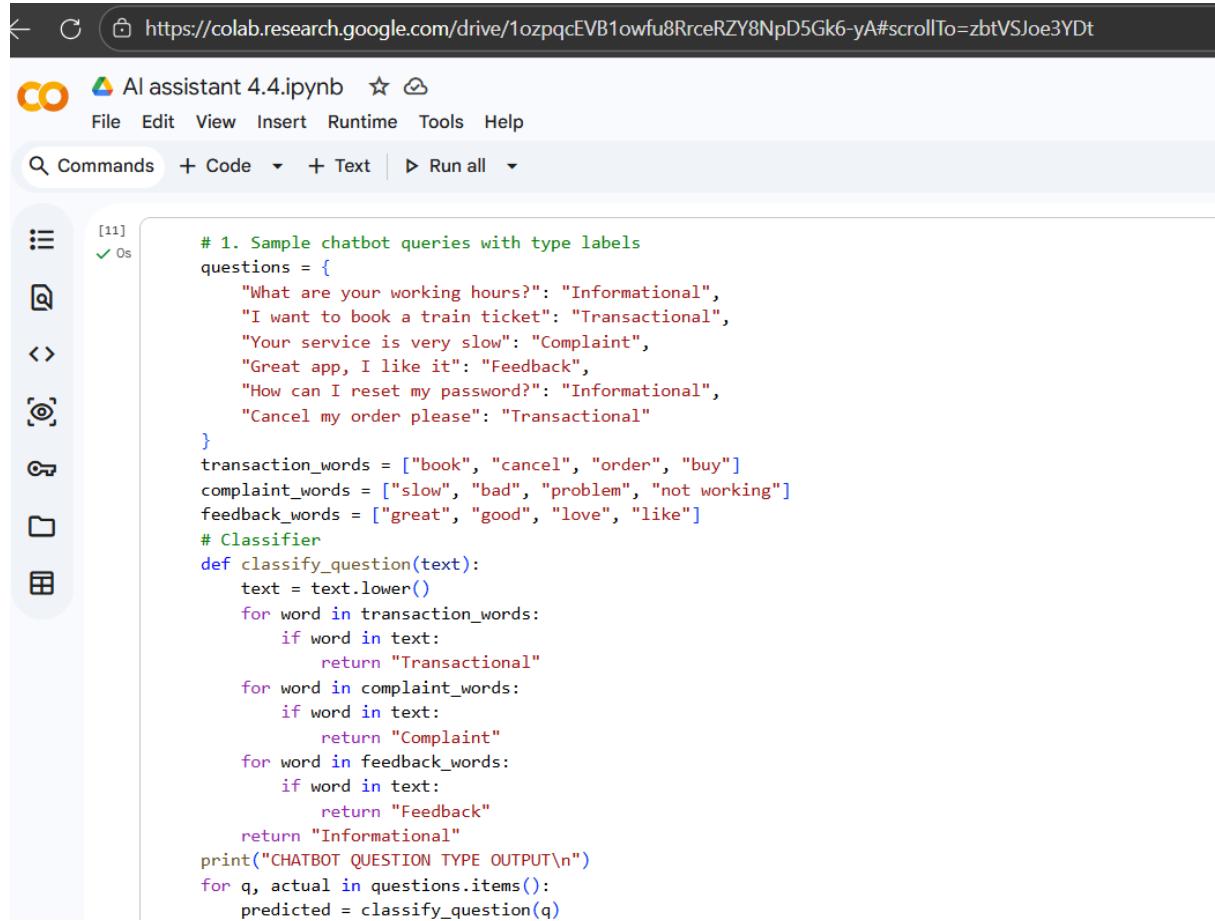
Query: How do I pay my admission fees?
Predicted: Admissions | Actual: Admissions

Query: How can I register for exams?
Predicted: Exams | Actual: Exams

Accuracy: 100.0 %

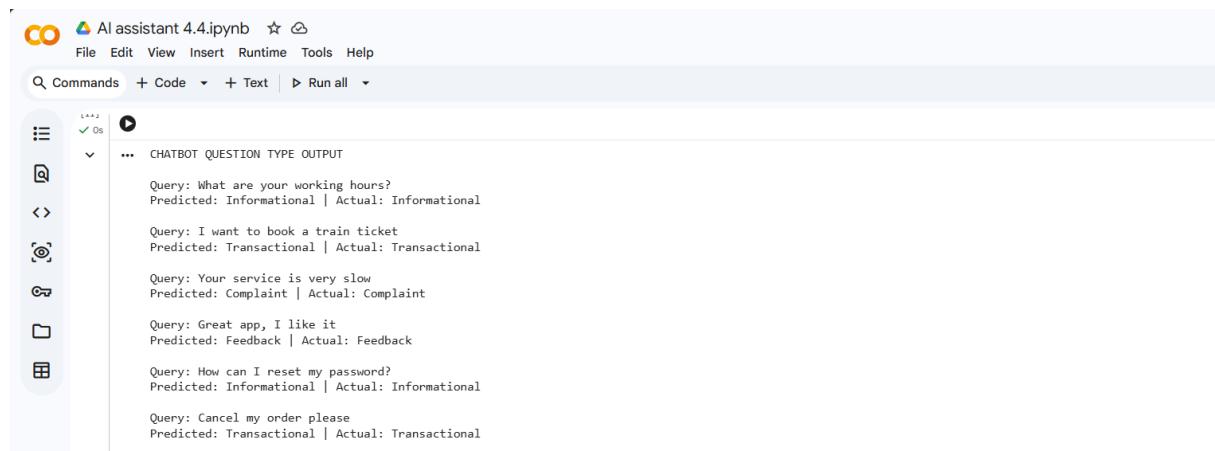
Conclusion: Few-shot > One-shot > Zero-shot in reliability
```

Scenario4: Chatbot Question Type Detection



The screenshot shows a Google Colab notebook titled "AI assistant 4.4.ipynb". The code cell [11] contains the following Python script:

```
# 1. Sample chatbot queries with type labels
questions = {
    "What are your working hours?": "Informational",
    "I want to book a train ticket": "Transactional",
    "Your service is very slow": "Complaint",
    "Great app, I like it": "Feedback",
    "How can I reset my password?": "Informational",
    "Cancel my order please": "Transactional"
}
transaction_words = ["book", "cancel", "order", "buy"]
complaint_words = ["slow", "bad", "problem", "not working"]
feedback_words = ["great", "good", "love", "like"]
# Classifier
def classify_question(text):
    text = text.lower()
    for word in transaction_words:
        if word in text:
            return "Transactional"
    for word in complaint_words:
        if word in text:
            return "Complaint"
    for word in feedback_words:
        if word in text:
            return "Feedback"
    return "Informational"
print("CHATBOT QUESTION TYPE OUTPUT\n")
for q, actual in questions.items():
    predicted = classify_question(q)
```



The screenshot shows the execution results of the code cell from the previous screenshot. The output is:

```
... CHATBOT QUESTION TYPE OUTPUT
Query: What are your working hours?
Predicted: Informational | Actual: Informational

Query: I want to book a train ticket
Predicted: Transactional | Actual: Transactional

Query: Your service is very slow
Predicted: Complaint | Actual: Complaint

Query: Great app, I like it
Predicted: Feedback | Actual: Feedback

Query: How can I reset my password?
Predicted: Informational | Actual: Informational

Query: Cancel my order please
Predicted: Transactional | Actual: Transactional
```

zero shot&one shot

A screenshot of a Jupyter Notebook interface titled "AI assistant 4.4.ipynb". The code cell [12] contains Python code for zero-shot learning:

```
# 2. ZERO-SHOT
print("ZERO-SHOT OUTPUT")
test_q = "Cancel my ticket"
print("Query:", test_q)
print("Type:", classify_question(test_q))
print()

ZERO-SHOT OUTPUT
Query: Cancel my ticket
Type: Transactional
```

The code cell [13] contains Python code for one-shot learning:

```
# 3. ONE-SHOT
print("ONE-SHOT OUTPUT")
print("Example: Query: 'This app is bad' -> Complaint")
test_q = "I want to book a bus"
print("Query:", test_q)
print("Type:", classify_question(test_q))
print()

... ONE-SHOT OUTPUT
Example: Query: 'This app is bad' -> Complaint
Query: I want to book a bus
Type: Transactional
```

Few shot

A screenshot of a Jupyter Notebook interface titled "AI assistant 4.4.ipynb". The code cell [14] contains Python code for few-shot learning:

```
# 4. FEW-SHOT
print("FEW-SHOT OUTPUT")
print("Examples:")
print("Query: 'What is your refund policy?' -> Informational")
print("Query: 'Buy a movie ticket' -> Transactional")
print("Query: 'App crashes' -> Complaint")
print("Query: 'Love this service' -> Feedback")
test_q = "Your service is not working"
print("Query:", test_q)
print("Type:", classify_question(test_q))
print()

... FEW-SHOT OUTPUT
Examples:
Query: 'What is your refund policy?' -> Informational
Query: 'Buy a movie ticket' -> Transactional
Query: 'App crashes' -> Complaint
Query: 'Love this service' -> Feedback
Query: 'Your service is not working'
Type: Complaint
```

A screenshot of a Jupyter Notebook interface titled "AI assistant 4.4.ipynb". The code cell [15] contains Python code for an accuracy check and reliability comparison:

```
correct = 0
total = len(questions)
print("ACCURACY CHECK")
for q, label in questions.items():
    result = classify_question(q)
    print("Query:", q)
    print("Predicted:", result, "| Actual:", label)
    print()
    if result == label:
        correct += 1
accuracy = (correct / total) * 100
print("Accuracy:", accuracy, "%")
print("\nConclusion: Few-shot > One-shot > Zero-shot in reliability")

... ACCURACY CHECK
Query: What are your working hours?
Predicted: Informational | Actual: Informational

Query: I want to book a train ticket
Predicted: Transactional | Actual: Transactional

Query: Your service is very slow
Predicted: Complaint | Actual: Complaint

Query: Great app, I like it
Predicted: Feedback | Actual: Feedback

Query: How can I reset my password?
Predicted: Informational | Actual: Informational

Query: Cancel my order please
Predicted: Transactional | Actual: Transactional

Accuracy: 100.0 %

Conclusion: Few-shot > One-shot > Zero-shot in reliability
```

Scenario5: Emotion Detection in Text

The screenshot shows a Jupyter Notebook interface with the following code:

```
# 1. Sample texts with emotion labels
emotions = {
    "I am feeling very happy today": "Happy",
    "I feel so lonely": "Sad",
    "This makes me very angry": "Angry",
    "I am worried about my exams": "Anxious",
    "I am going to college now": "Neutral"
}

happy_words = ["happy", "great", "excited"]
sad_words = ["lonely", "sad", "cry"]
angry_words = ["angry", "mad", "furious"]
anxious_words = ["worried", "nervous", "scared"]

def classify_emotion(text):
    text = text.lower()
    for word in happy_words:
        if word in text:
            return "Happy"
    for word in sad_words:
        if word in text:
            return "Sad"
    for word in angry_words:
        if word in text:
            return "Angry"
    for word in anxious_words:
        if word in text:
            return "Anxious"
    return "Neutral"

print("EMOTION DETECTION OUTPUT\n")
for text, actual in emotions.items():
    predicted = classify_emotion(text)
    print("Text:", text)
    print("Predicted:", predicted, "| Actual:", actual)
    print()
```

The screenshot shows the execution results of the code from the previous screenshot. The notebook cell output displays the predicted emotions for the sample texts:

```
... EMOTION DETECTION OUTPUT
Text: I am feeling very happy today
Predicted: Happy | Actual: Happy

Text: I feel so lonely
Predicted: Sad | Actual: Sad

Text: This makes me very angry
Predicted: Angry | Actual: Angry

Text: I am worried about my exams
Predicted: Anxious | Actual: Anxious

Text: I am going to college now
Predicted: Neutral | Actual: Neutral
```

At the bottom of the screen, the Windows taskbar is visible with various icons for applications like File Explorer, Edge, and Mail.

Zero&one shot

A screenshot of a Jupyter Notebook interface titled "AI assistant 4.4.ipynb". The notebook contains three code cells:

- [18] **# 2. ZERO-SHOT**

```
print("ZERO-SHOT OUTPUT")
test_text = "I am nervous about my future"
print("Text:", test_text)
print("Emotion:", classify_emotion(test_text))
print()
```

... ZERO-SHOT OUTPUT
Text: I am nervous about my future
Emotion: Anxious
- [19] **# 3. ONE-SHOT**

```
print("ONE-SHOT OUTPUT")
print("Example: Text: 'I feel very sad today' -> Sad")
test_text = "I am very happy today"
print("Text:", test_text)
print("Emotion:", classify_emotion(test_text))
print()
```

... ONE-SHOT OUTPUT
Example: Text: 'I feel very sad today' -> Sad
Text: I am very happy today
Emotion: Happy

Few shot

A screenshot of a Jupyter Notebook interface titled "AI assistant 4.4.ipynb". The notebook contains two code cells:

- [28] **# 4. FEW-SHOT**

```
print("FEW-SHOT OUTPUT")
print("Examples:")
print("Text: 'I am excited today' -> Happy")
print("Text: 'I feel lonely' -> Sad")
print("Text: 'This is so annoying' -> Angry")
print("Text: 'I am scared about exams' -> Anxious")
test_text = "I am scared about my results"
print("Text:", test_text)
print("Emotion:", classify_emotion(test_text))
print()
```

... FEW-SHOT OUTPUT
Examples:
Text: 'I am excited today' -> Happy
Text: 'I feel lonely' -> Sad
Text: 'This is so annoying' -> Angry
Text: 'I am scared about exams' -> Anxious
Text: I am scared about my results
Emotion: Anxious

A screenshot of a Jupyter Notebook interface titled "AI assistant 4.4.ipynb". The notebook contains two code cells:

- [21] **# 5. Accuracy Check**

```
correct = 0
total = len(emotions)
print("ACCURACY CHECK")
for text, label in emotions.items():
    result = classify_emotion(text)
    print("Text:", text)
    print("Predicted:", result, "| Actual:", label)
    print()
    if result == label:
        correct += 1
accuracy = (correct / total) * 100
print("Accuracy:", accuracy, "%")
print("\nConclusion: Few-shot > One-shot > Zero-shot in reliability")
```

... ACCURACY CHECK
Text: I am feeling very happy today
Predicted: Happy | Actual: Happy

Text: I feel so lonely
Predicted: Sad | Actual: Sad

Text: This makes me very angry
Predicted: Angry | Actual: Angry

Text: I am worried about my exams
Predicted: Anxious | Actual: Anxious

Text: I am going to college now
Predicted: Neutral | Actual: Neutral

Accuracy: 100.0 %

Conclusion: Few-shot > One-shot > Zero-shot in reliability