

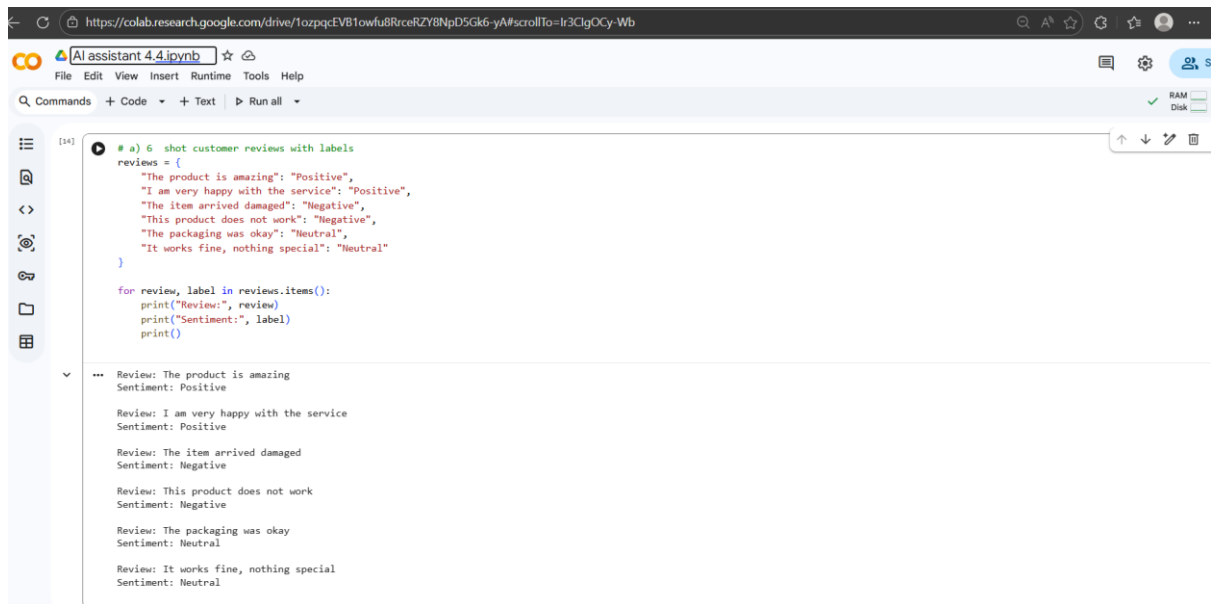
Assignment-4.4

Name: B. Sanjana

H.T.NO: 2303A52306

Scenario1: E-commerce

a) 6 short customer reviews mapped to sentiment labels.



```
[14]: # 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()
```

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



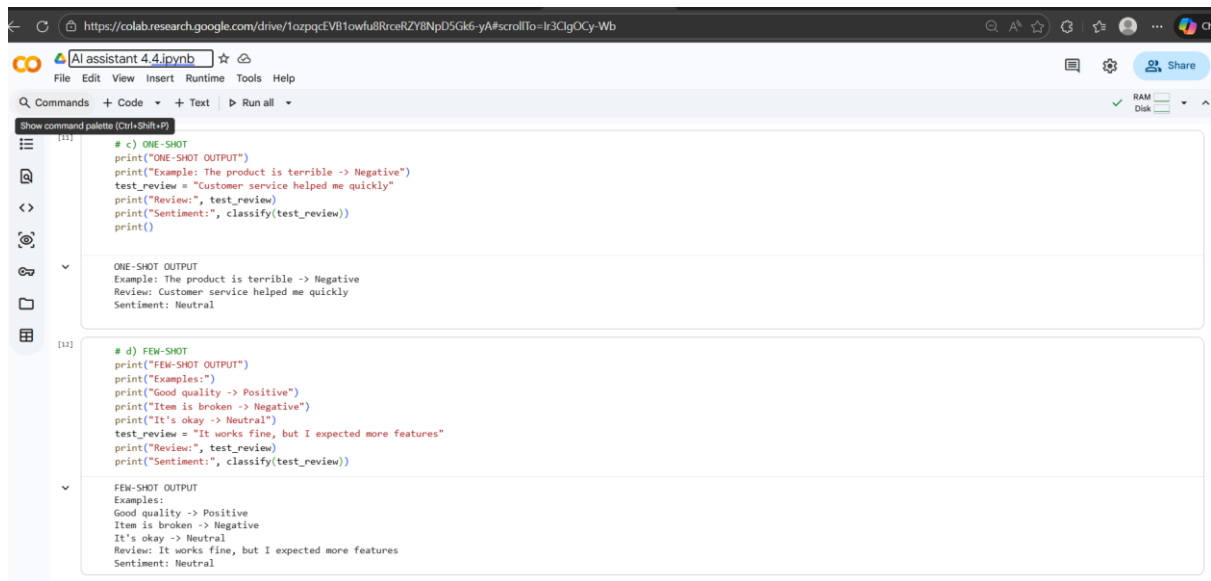
```
[18]: # b) ZERO-SHOT
def classify(review):
    zero_shot_prompt = f"classify the sentiment of the following customer review into Positive, Negative, or Neutral.
    Review: '{review}'
    Sentiment: ''"

    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()
```

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". It contains two code cells. The first cell, labeled [11], demonstrates a "One-Shot" classification task. It defines a function that takes a review string and a list of example reviews with their sentiment labels. The function uses a prompt to instruct the model to classify the new review based on the provided examples. The output shows the model correctly classifying the sentiment as "Neutral".

```
[11] # 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()
```

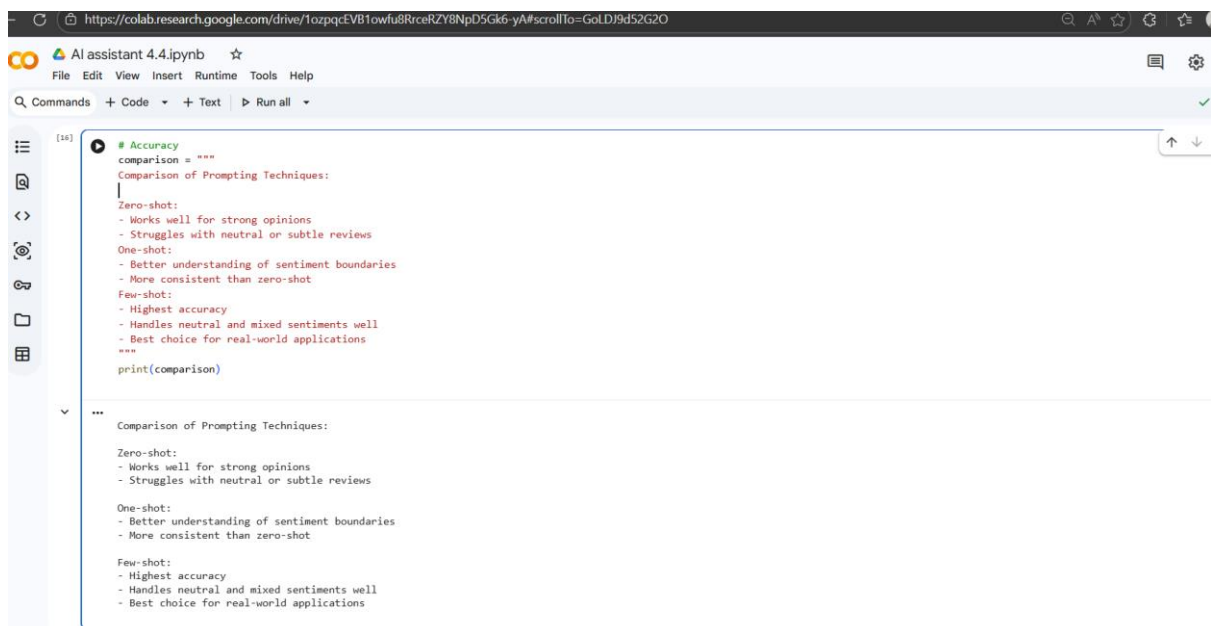
ONE-SHOT OUTPUT
Example: The product is terrible -> Negative
Review: Customer service helped me quickly
Sentiment: Neutral

The second cell, labeled [12], demonstrates a "Few-Shot" classification task. It follows a similar pattern but provides more than one example in the prompt. The output shows the model classifying the sentiment as "Neutral".

```
[12] # 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))
```

FEW-SHOT OUTPUT
Examples:
Good quality -> Positive
Item is broken -> Negative
It's okay -> Neutral
Review: It works fine, but I expected more features
Sentiment: Neutral

Comparison



The screenshot shows a Google Colab notebook titled "AI assistant 4.4.ipynb". It contains a single code cell, labeled [16], which defines a function to compare three prompting techniques: Zero-shot, One-shot, and Few-shot. The function generates a comparison table based on the provided criteria. The output shows the model's comparison of the three techniques.

```
[16] # 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

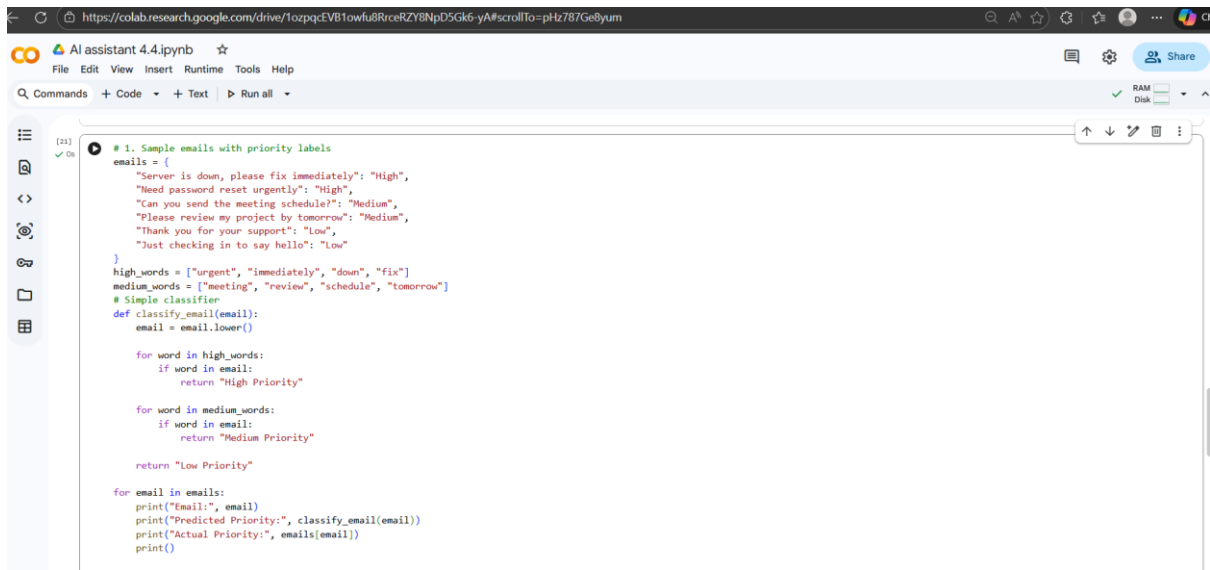
One-shot:

- 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



```
# 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

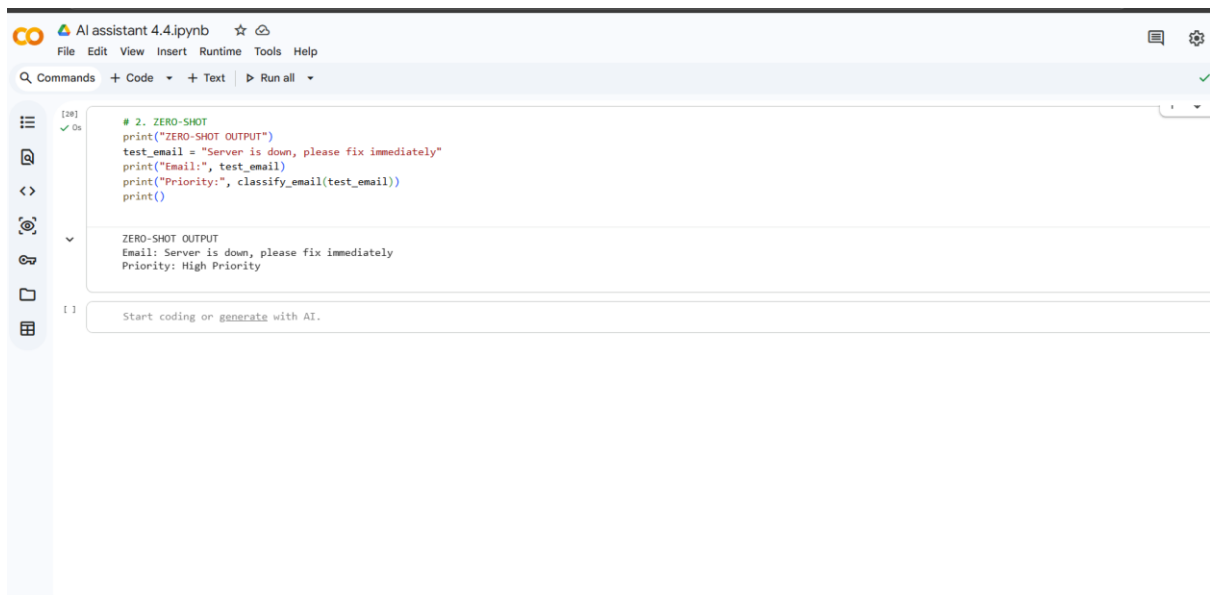
Email: Can you send the meeting schedule?
Predicted Priority: Medium Priority
Actual Priority: Medium

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

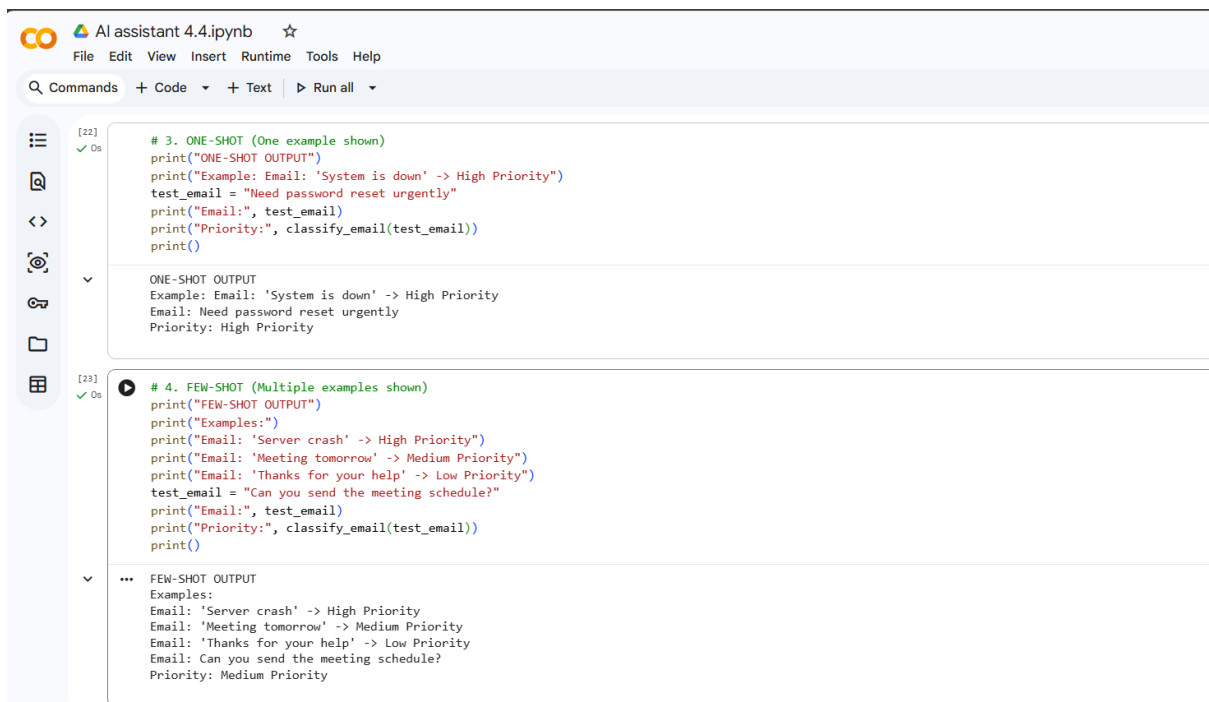


```
# 2. ZERO-SHOT
print("ZERO-SHOT OUTPUT")
test_email = "Server is down, please fix immediately"
print("Email:", test_email)
print("Priority:", classify_email(test_email))
print()
```

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 Google Colab notebook titled "AI assistant 4.4.ipynb". It contains two code cells. The first cell, labeled [22], demonstrates a ONE-SHOT example where a single email is classified. The second cell, labeled [23], demonstrates a FEW-SHOT example where multiple examples are provided for classification. Both cells show the code and the resulting output.

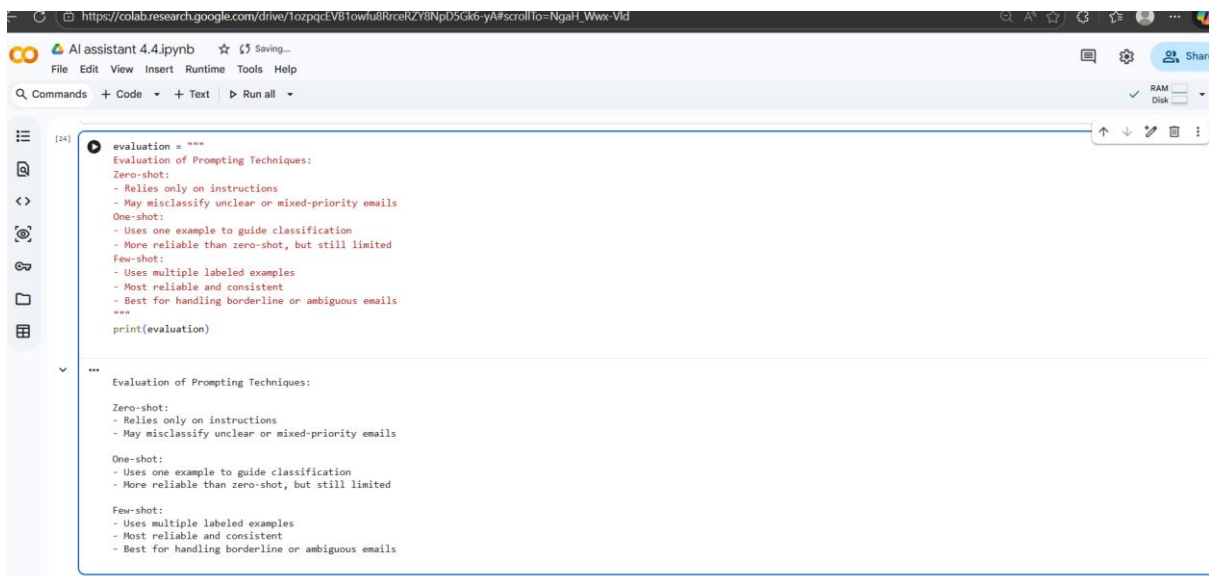
```
[22] ✓ Os
# 3. ONE-SHOT (One example shown)
print("ONE-SHOT OUTPUT")
print("Example: Email: 'System is down' -> High Priority")
test_email = "Need password reset urgently"
print("Email:", test_email)
print("Priority:", classify_email(test_email))
print()

ONE-SHOT OUTPUT
Example: Email: 'System is down' -> High Priority
Email: Need password reset urgently
Priority: High Priority

[23] ✓ Os
# 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()

FEW-SHOT OUTPUT
Examples:
Email: 'Server crash' -> High Priority
Email: 'Meeting tomorrow' -> Medium Priority
Email: 'Thanks for your help' -> Low Priority
Email: Can you send the meeting schedule?
Priority: Medium Priority
```

Comparison

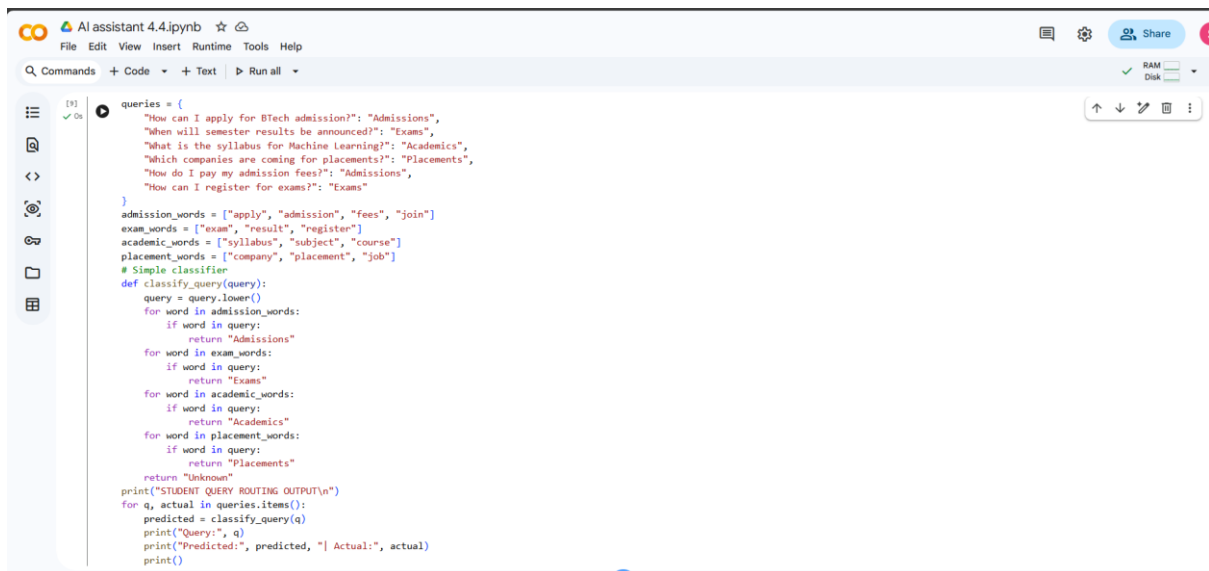


The screenshot shows a Google Colab notebook titled "AI assistant 4.4.ipynb". It contains a code cell labeled [24] that evaluates different prompting techniques. The code defines a string variable 'evaluation' containing a comparison of Zero-shot, One-shot, and Few-shot techniques. The output of the code is displayed below the cell.

```
[24]
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



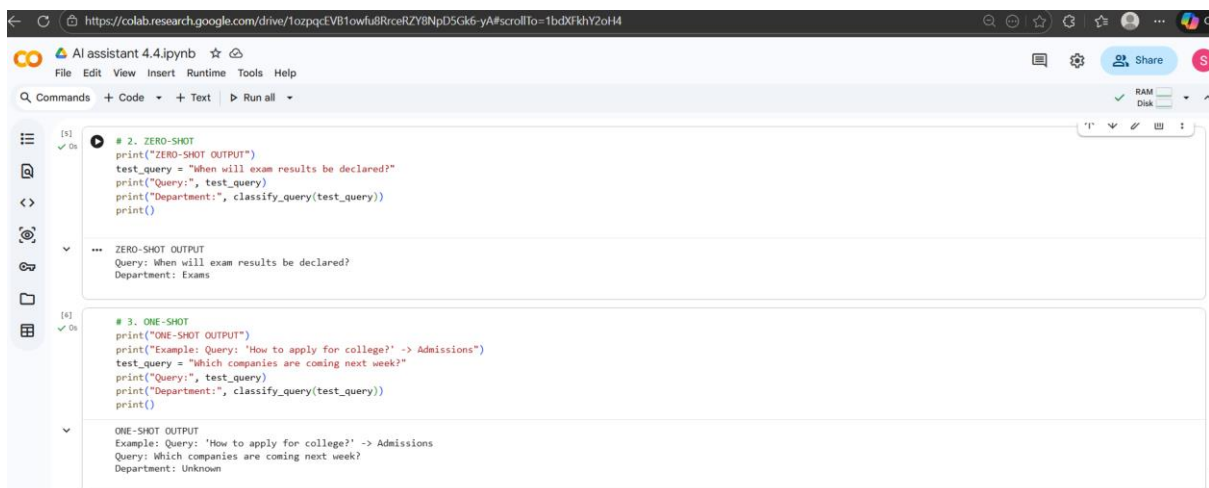
```
[1]: ✓ On
queries = {
    "How can I apply for BTech admission?": "Admissions",
    "When will semester results be announced?": "Exams",
    "What is the syllabus for Machine Learning?": "Academics",
    "Which companies are coming for placements?": "Placements",
    "How do I pay my admission fees?": "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



```
[5]: ✓ On
# 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

[6]: ✓ On
# 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



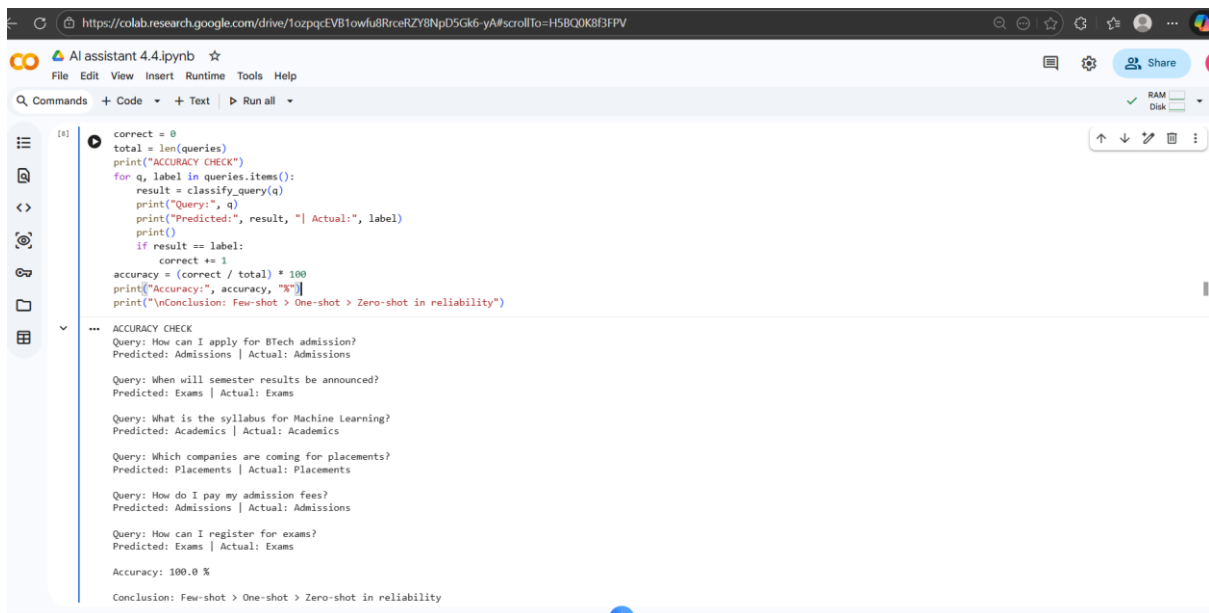
The image shows a Jupyter Notebook interface for 'AI assistant 4.4.ipynb'. The notebook has two cells. The first cell, labeled '[7]', contains Python code for a FEW-SHOT experiment. It prints 'FEW-SHOT OUTPUT', lists examples of queries and their predicted departments, and then uses a `classify_query` function to predict the department for a test query. The output shows the predicted department as 'Admissions'. The second cell, labeled '[8]', contains Python code for an Accuracy Check. It initializes `correct = 0` and `total = len(queries)`, then iterates over a list of queries, comparing the predicted department with the actual department. The code prints the accuracy percentage and a conclusion: 'Conclusion: Few-shot > One-shot > Zero-shot in reliability'.

```
[7]
# 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

[8]
# 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 image shows a Jupyter Notebook interface for 'AI assistant 4.4.ipynb'. The notebook has two cells. The first cell, labeled '[8]', contains Python code for an Accuracy Check. It initializes `correct = 0` and `total = len(queries)`, then iterates over a list of queries, comparing the predicted department with the actual department. The code prints the accuracy percentage and a conclusion: 'Conclusion: Few-shot > One-shot > Zero-shot in reliability'. The second cell, labeled '[9]', contains the output of the accuracy check. It shows a table of queries, predicted departments, and actual departments. The predicted departments are: Admissions, Exams, Academics, Placements, Admissions, Exams, and Exams. The actual departments are: Admissions, Exams, Academics, Placements, Admissions, Exams, and Exams. The accuracy is 100.0 %.

```
[8]
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")

[9]
...
ACCURACY CHECK
Query: How can I apply for B.Tech 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

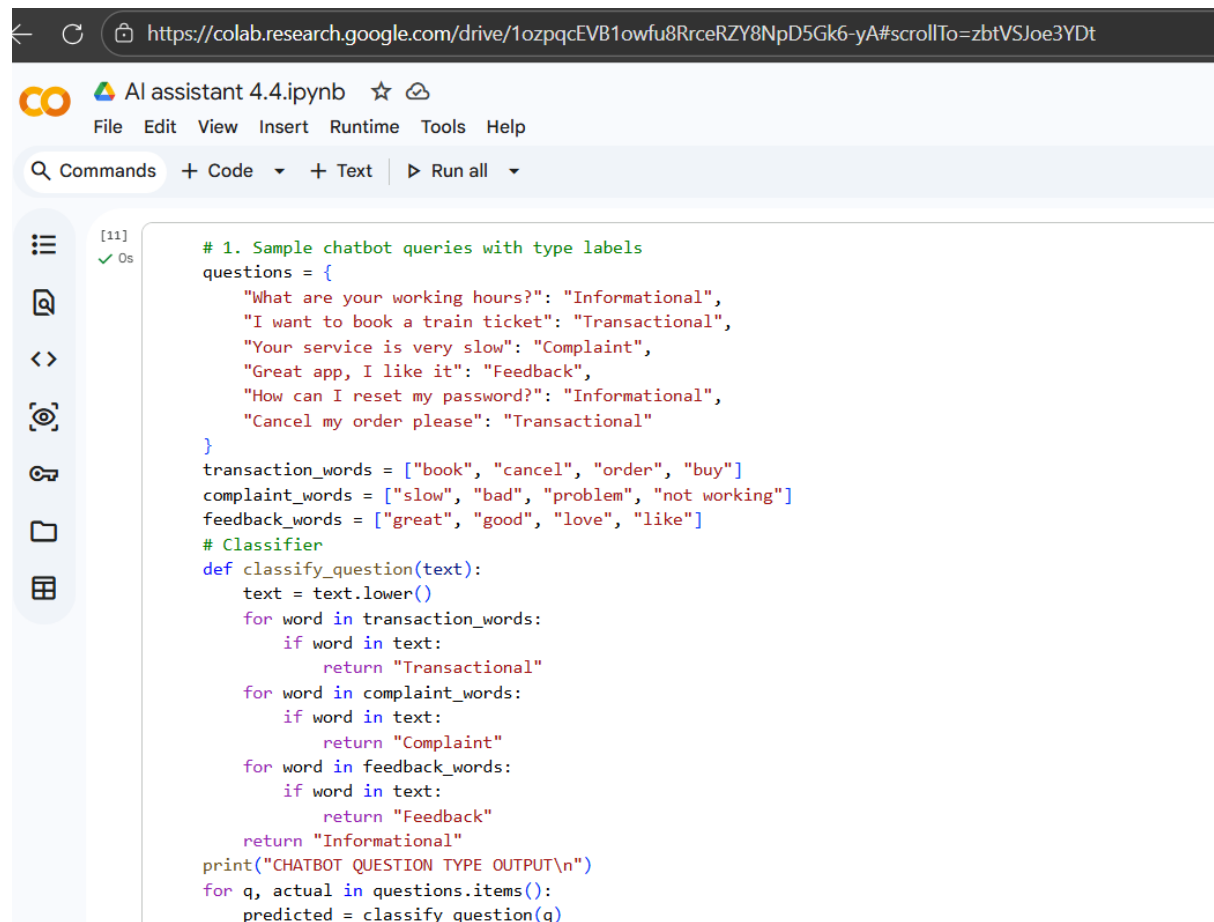
Query: How do I pay my admission fees?
Predicted: Admissions | Actual: Admissions

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

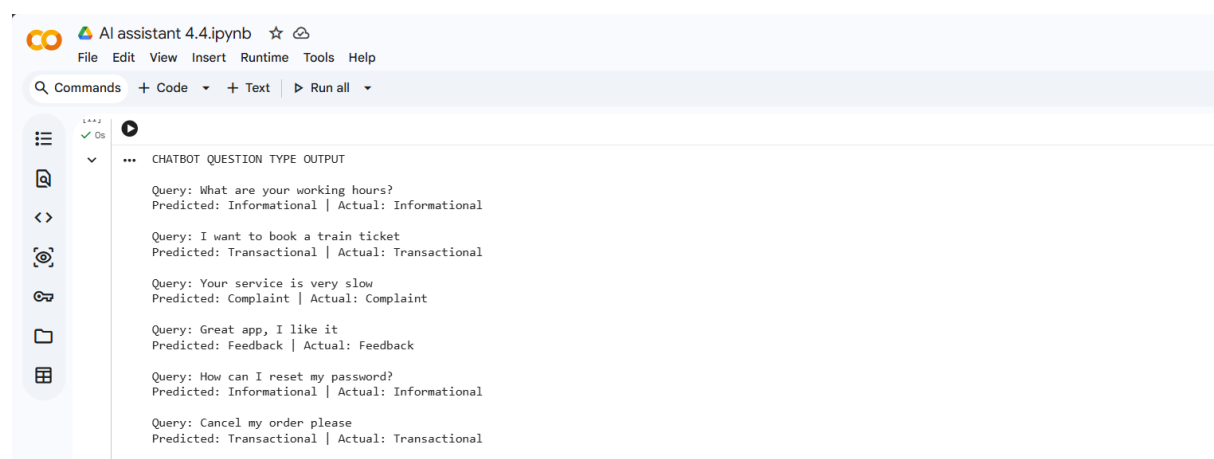
Accuracy: 100.0 %

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

Scenario4: Chatbot Question Type Detection



```
[11] ✓ Os
# 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)
```



```
✓ Os
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

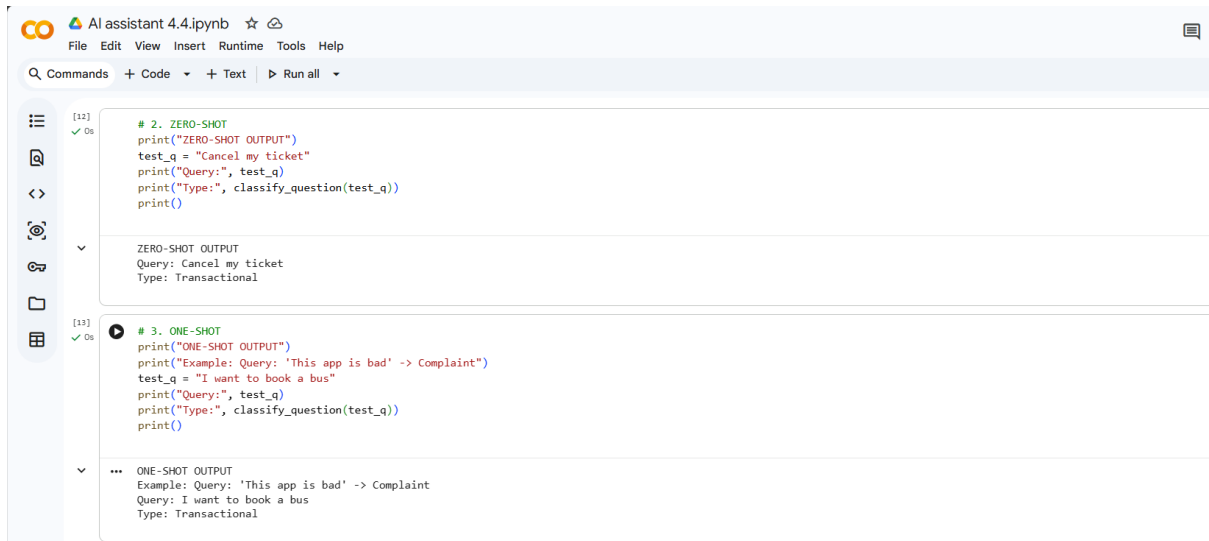
Query: Your service is very slow
Predicted: Complaint | Actual: Complaint

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



The screenshot shows a Jupyter Notebook titled "AI assistant 4.4.ipynb". It contains two code cells. The first cell, labeled "[12] ✓ Os", defines a function `classify_question` and tests it with the query "Cancel my ticket", resulting in a "Transactional" classification. The second cell, labeled "[13] ✓ Os", tests the same function with the query "This app is bad", resulting in a "Complaint" classification. The output for the second cell shows the example query and the resulting classification.

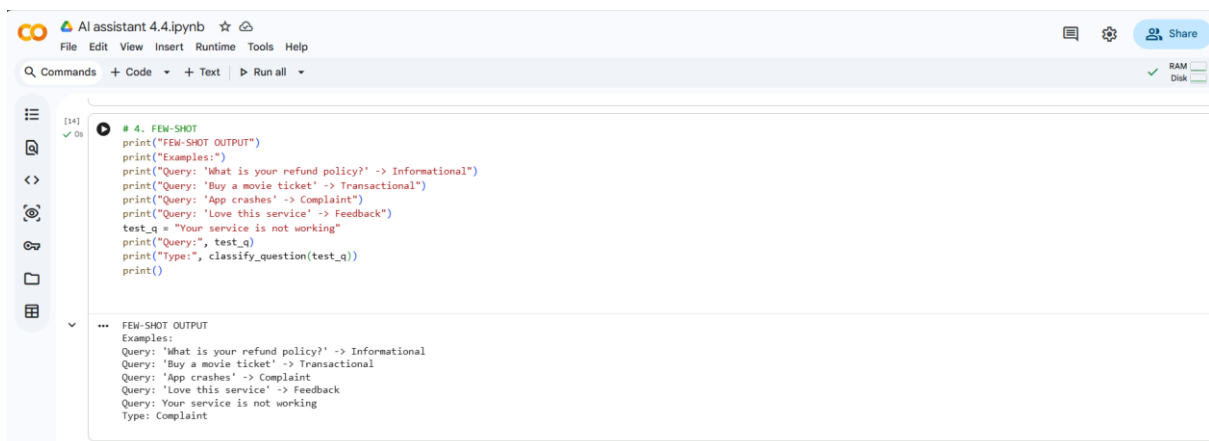
```
# 2. ZERO-SHOT
print("ZERO-SHOT OUTPUT")
test_q = "Cancel my ticket"
print("Query:", test_q)
print("Type:", classify_question(test_q))
print()

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

# 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
```

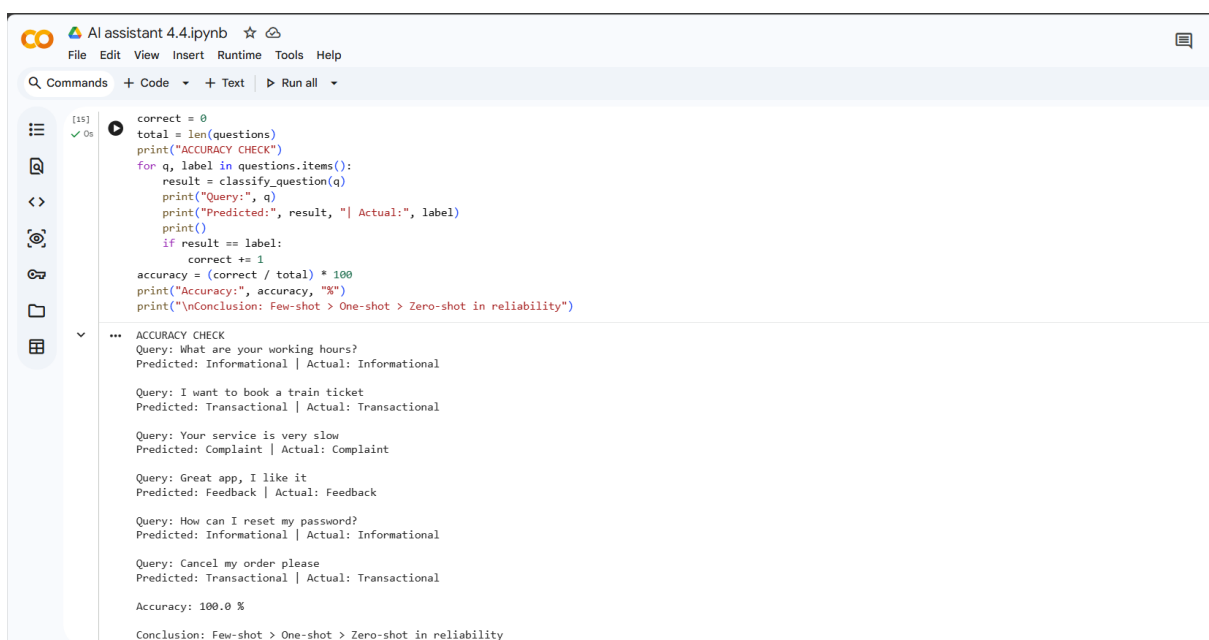
Few shot



The screenshot shows a Jupyter Notebook titled "AI assistant 4.4.ipynb". It contains a code cell labeled "[14] ✓ Os" that defines a function `classify_question` and tests it with a list of example queries. The output shows the example queries and the resulting classifications: "What is your refund policy?" is "Informational", "Buy a movie ticket" is "Transactional", "App crashes" is "Complaint", "Love this service" is "Feedback", and "Your service is not working" is "Complaint".

```
# 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
```



The screenshot shows a Jupyter Notebook titled "AI assistant 4.4.ipynb". It contains a code cell labeled "[15] ✓ Os" that defines a function `classify_question` and tests it with a list of example queries. The output shows the example queries and the resulting classifications: "What are your working hours?" is "Informational", "I want to book a train ticket" is "Transactional", "Your service is very slow" is "Complaint", "Great app, I like it" is "Feedback", "How can I reset my password?" is "Informational", and "Cancel my order please" is "Transactional". The output also shows the accuracy of the model, which is 100.0%.

```
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

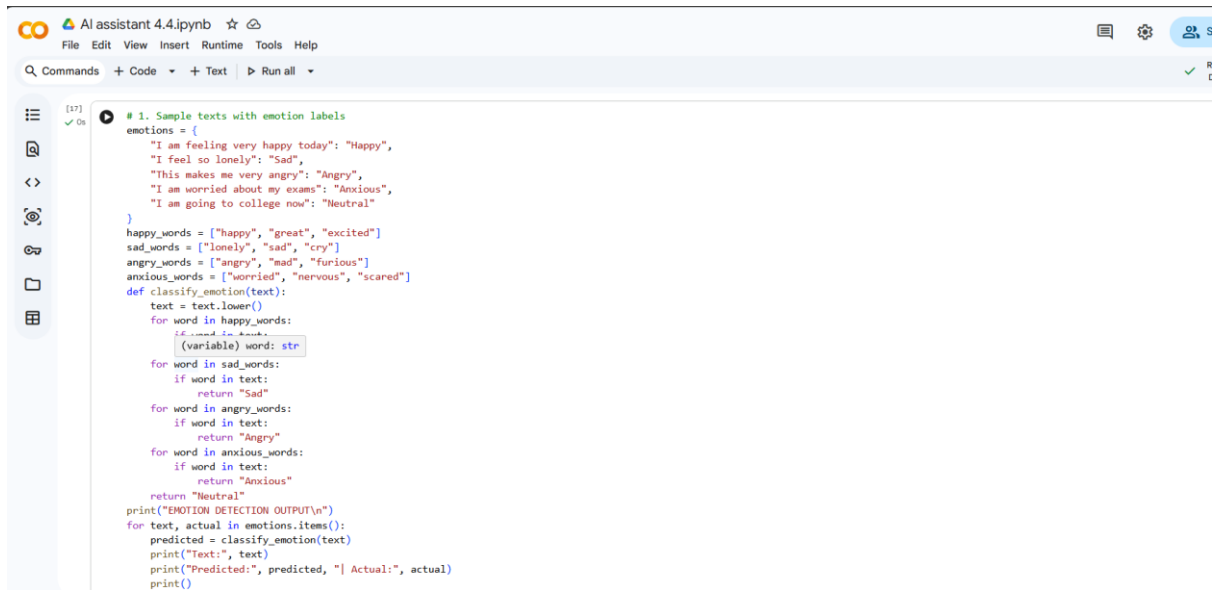
Query: How can I reset my password?
Predicted: Informational | Actual: Informational

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

Accuracy: 100.0 %

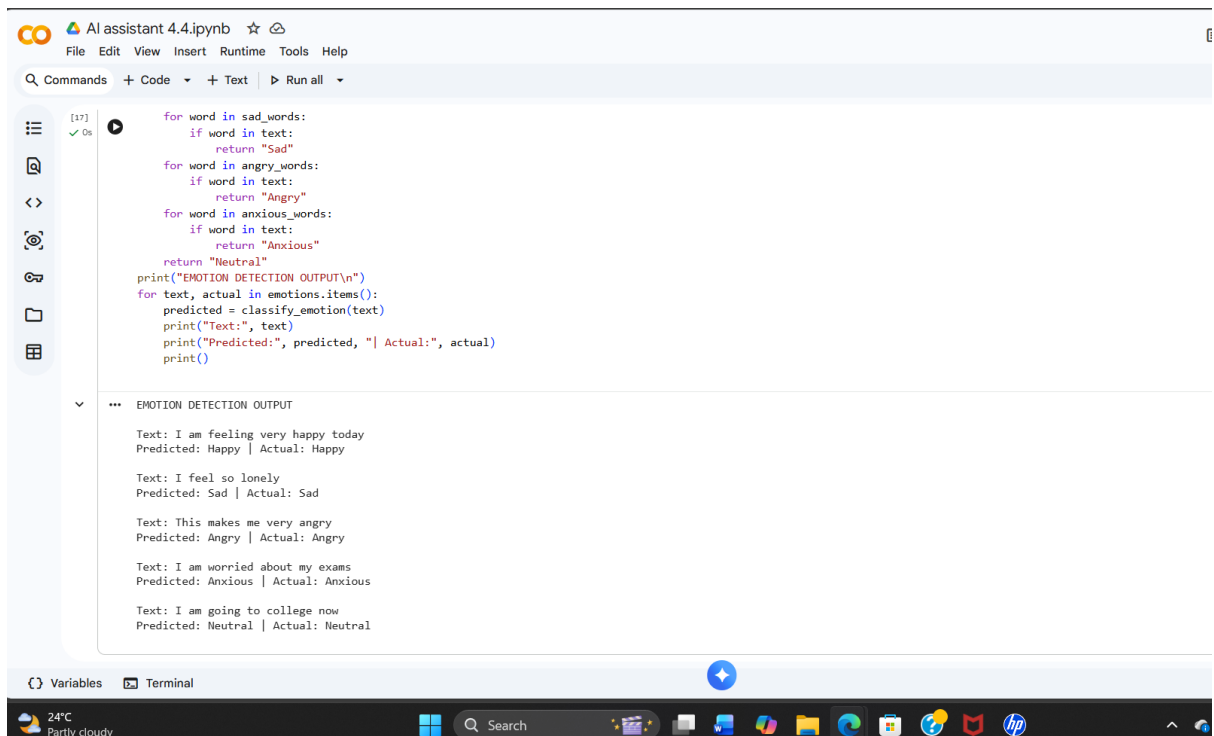
Conclusion: Few-shot > One-shot > Zero-shot in reliability
```


Scenario5: Emotion Detection in Text



The screenshot shows a Jupyter Notebook titled "AI assistant 4.4.ipynb". The code defines a dictionary of emotions and lists of words associated with each emotion. A function `classify_emotion(text)` is defined to classify the emotion of a given text. The code then iterates over the emotions and prints the predicted emotion for each sample text.

```
[17] ✓ Os
# 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 same Jupyter Notebook interface, but now displaying the output of the code. The output is a list of sample texts and their predicted emotions, along with the actual emotions from the dictionary.

```
[17] ✓ Os
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()

--- EMOTION DETECTION OUTPUT

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

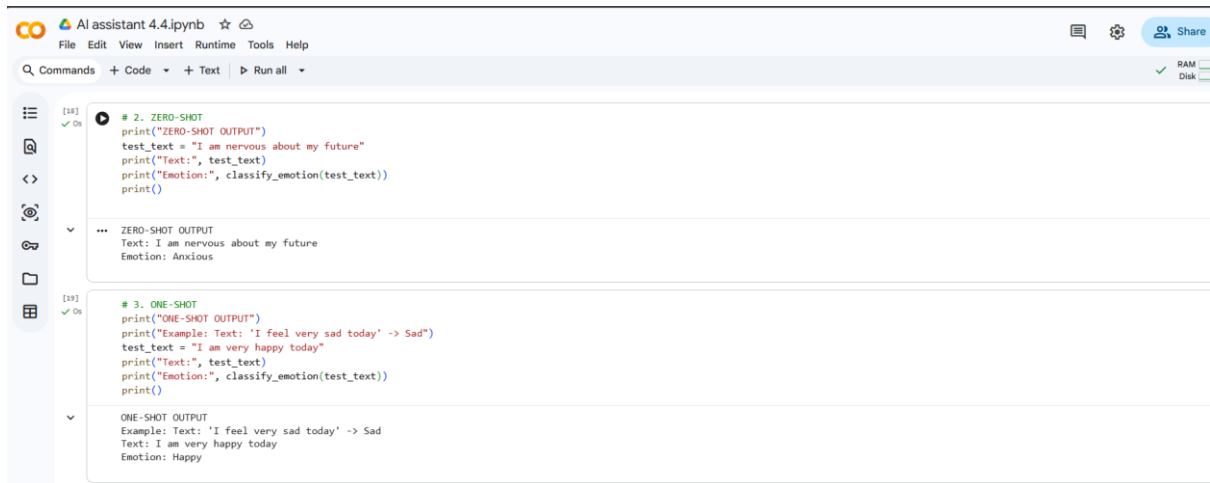
Text: I feel so lonely
Predicted: Sad | Actual: Sad

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
```

Zero&one shot



The screenshot shows a Jupyter Notebook interface for 'AI assistant 4.4.ipynb'. It contains two code cells. The first cell, labeled '[18]', is titled '# 2. ZERO-SHOT' and contains Python code to print 'ZERO-SHOT OUTPUT', a test text 'I am nervous about my future', and the emotion classification result 'Anxious'. The second cell, labeled '[19]', is titled '# 3. ONE-SHOT' and contains Python code to print 'ONE-SHOT OUTPUT', an example text 'I feel very sad today' with a label 'Sad', a test text 'I am very happy today', and the emotion classification result 'Happy'.

```
[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



The screenshot shows a Jupyter Notebook interface for 'AI assistant 4.4.ipynb'. It contains a code cell labeled '[20]' titled '# 4. FEW-SHOT'. The code prints 'FEW-SHOT OUTPUT', followed by several examples of text and their corresponding emotion labels (Happy, Sad, Angry, Anxious). A test text 'I am scared about my results' is then used to demonstrate the model's prediction, which is 'Anxious'.

```
[20] # 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
```



The screenshot shows a Jupyter Notebook interface for 'AI assistant 4.4.ipynb'. It contains a code cell labeled '[21]' titled '# 5. Accuracy Check'. The code iterates through a list of text samples, compares the predicted emotion with the actual label, and calculates the overall accuracy. The results show 100.0% accuracy across five samples. The conclusion states: 'Conclusion: Few-shot > One-shot > Zero-shot in reliability'.

```
[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
```