

2303a51940

Lab assignment-4.4

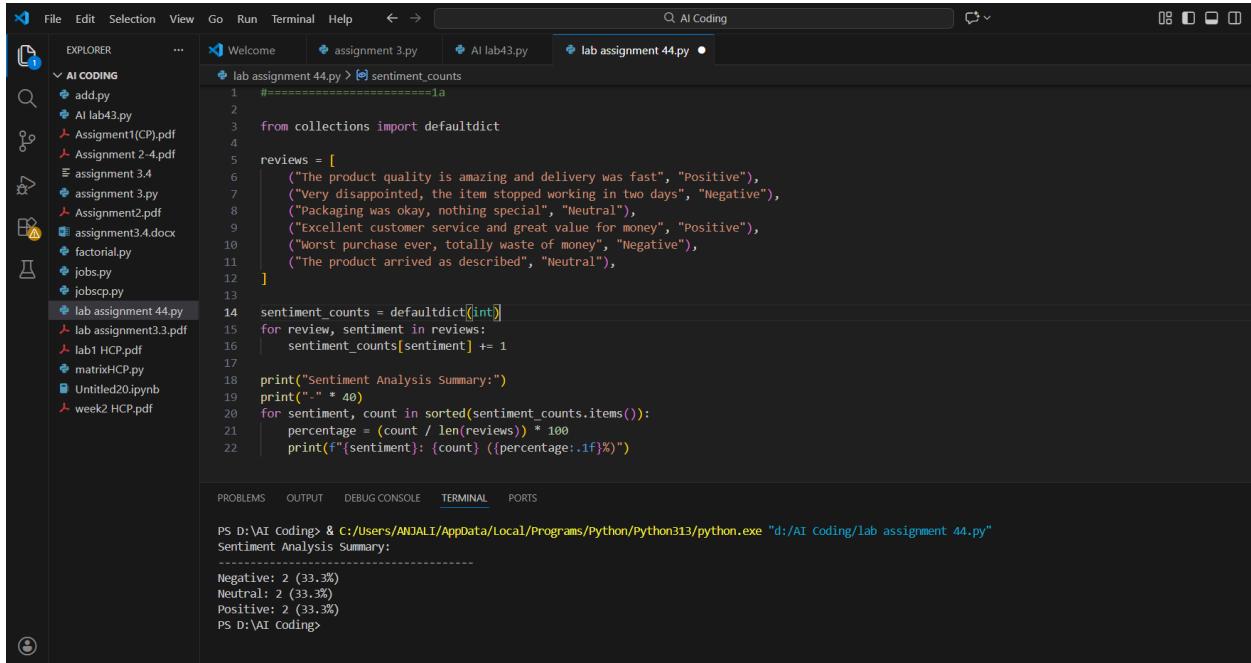
1. Sentiment Classification for Customer Reviews

Scenario:

An e-commerce platform wants to analyze customer reviews and classify them into Positive, Negative, or Neutral sentiments using prompt engineering

Tasks:

a) Prepare 6 short customer reviews mapped to sentiment labels.



The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The left sidebar (EXPLORER) lists files including 'AI CODING' (with 'assignment 3.py', 'AI lab43.py', 'Assignment1(CP).pdf', 'Assignment 2-4.pdf', 'assignment 3.4', 'assignment 3.py', 'Assignment2.pdf', 'assignment3.4.docx', 'factorial.py', 'jobs.py', 'jobscp.py'), 'lab assignment 44.py', 'lab assignment3.3.pdf', 'lab1 HCP.pdf', 'matrixHCP.py', 'Untitled20.ipynb', and 'week2 HCP.pdf'. The right pane shows a code editor with Python code for sentiment analysis. The code imports 'collections.defaultdict' and defines a list 'reviews' containing 6 customer reviews with their sentiment labels ('Positive', 'Negative', 'Neutral'). It then initializes a defaultdict 'sentiment_counts' and iterates through each review to increment the count for its sentiment. Finally, it prints a summary of the counts and percentages. Below the code editor is a terminal window showing the execution of the script and the resulting output.

```
#=====1a
from collections import defaultdict

reviews = [
    ("The product quality is amazing and delivery was fast", "Positive"),
    ("Very disappointed, the item stopped working in two days", "Negative"),
    ("Packaging was okay, nothing special", "Neutral"),
    ("Excellent customer service and great value for money", "Positive"),
    ("Worst purchase ever, totally waste of money", "Negative"),
    ("The product arrived as described", "Neutral"),
]

sentiment_counts = defaultdict(int)
for review, sentiment in reviews:
    sentiment_counts[sentiment] += 1

print("Sentiment Analysis Summary:")
print("." * 40)
for sentiment, count in sorted(sentiment_counts.items()):
    percentage = (count / len(reviews)) * 100
    print(f"{'(sentiment)':>10} ({count}) ({percentage:.1f}%)")
```

PS D:\AI Coding> & C:/Users/ANDALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Sentiment Analysis Summary:

Negative: 2 (33.3%)
Neutral: 2 (33.3%)
Positive: 2 (33.3%)
PS D:\AI Coding>

b)Design a Zero-shot prompt to classify sentiment.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files in the "AI CODING" folder, including "add.py", "AI lab43.py", "Assignment1(CP).pdf", "Assignment 2-4.pdf", "assignment 3.4", "assignment 3.py", "Assignment2.pdf", "assignment3.4.docx", "factorial.py", "jobs.py", "jobscp.py", and "lab assignment 44.py".
- Code Editor:** Displays the content of "lab assignment 44.py". The code performs sentiment analysis based on keyword lists for positive and negative words.
- Terminal:** Shows the command "python.exe d:/AI Coding/lab assignment 44.py" being run, followed by the output "Sentiment Analysis Summary:" and the results: Negative: 2 (33.3%), Neutral: 2 (33.3%), Positive: 2 (33.3%).

1c) Design a One-shot prompt with one labeled example.

This screenshot shows the same setup as the first one, but with a different review input. The terminal output now shows:

```

Sentiment Analysis Summary:
-----
Negative: 2 (33.3%)
Neutral: 2 (33.3%)
Positive: 2 (33.3%)
PS D:\AI Coding> & C:/Users/ANDALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Review: "The product quality is amazing and delivery was fast"
Sentiment: Positive
PS D:\AI Coding>

```

1d) Design a Few-shot prompt with 3–5 labeled examples.

A screenshot of the Visual Studio Code (VS Code) interface. The left sidebar shows a file tree under the 'EXPLORER' tab, with several files listed including 'AI CODING', 'add.py', 'AI lab43.py', 'Assignment1(CP).pdf', 'Assignment 2-4.pdf', 'assignment 3.4', 'assignment 3.py', 'Assignment2.pdf', 'assignment3.4.docx', 'factorial.py', 'jobs.py', 'jobspp.py', 'lab assignment 44.py', 'lab assignment3.3.pdf', 'lab1 HCP.pdf', 'matrixHCP.py', 'Untitled20.ipynb', and 'week2 HCP.pdf'. The main editor area displays Python code for sentiment analysis:

```
5 #=====
6 def classify_sentiment(review):
7     """Classify sentiment of a review as Positive, Negative, or Neutral."""
8     review_lower = review.lower()
9
10    positive_words = {"amazing", "excellent", "great", "good", "love", "best", "wonderful", "fantastic"}
11    negative_words = {"worst", "hate", "bad", "terrible", "waste", "awful", "poor", "horrible"}
12
13    positive_count = sum(1 for word in positive_words if word in review_lower)
14    negative_count = sum(1 for word in negative_words if word in review_lower)
15
16    if positive_count > negative_count:
17        return "Positive"
18    elif negative_count > positive_count:
19        return "Negative"
20    else:
21        return "Neutral"
22
23    # Test with the provided example
24    review = "Worst purchase ever, totally waste of money"
25    print(f"Review: '{review}'")
26    print(f"Sentiment: {classify_sentiment(review)}")
```

The 'TERMINAL' tab at the bottom shows command-line output from running the script:

```
Neutral: 2 (33.3%)
Positive: 2 (33.3%)
PS D:\AI Coding & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Review: "The product quality is amazing and delivery was fast"
Sentiment: Positive
PS D:\AI Coding & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Review: "Excellent customer service and great value for money"
Sentiment: Positive
PS D:\AI Coding & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Review: "Worst purchase ever, totally waste of money"
Sentiment: Negative
PS D:\AI Coding
```

1e) Compare the outputs and discuss accuracy differences.

A screenshot of the Visual Studio Code (VS Code) interface. The left sidebar shows a file tree under the 'EXPLORER' tab, with files including 'AI CODING', 'add.py', 'AI lab43.py', 'Assignment1(CP).pdf', 'Assignment 2-4.pdf', 'assignment 3.4', 'assignment 3.py', 'Assignment2.pdf', 'assignment3.4.docx', 'factorial.py', 'jobs.py', 'jobspp.py', 'lab assignment ...', 'lab assignment3.3.pdf', 'lab1 HCP.pdf', 'matrixHCP.py', 'Untitled20.ipynb', and 'week2 HCP.pdf'. The main editor area displays Python code for sentiment analysis using a DataFrame:

```
8 #=====
9 data = [
10     {
11         "Technique": ['Zero-shot', 'One-shot', 'Few-shot'],
12         "Accuracy": ['Medium', 'Better', 'Best'],
13         "Reason": [
14             'Relies only on model knowledge',
15             'Learns from one example',
16             'Clear pattern learning from multiple examples'
17         ]
18     }
19 df = pd.DataFrame(data)
20 print(df)
```

The 'TERMINAL' tab at the bottom shows command-line output from running the script, which includes a pandas module error:

```
PS D:\AI Coding & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Review: "Excellent customer service and great value for money"
Sentiment: Positive
PS D:\AI Coding & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Review: "Worst purchase ever, totally waste of money"
Sentiment: Negative
PS D:\AI Coding & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Traceback (most recent call last):
  File "d:/AI Coding/lab assignment 44.py", line 1, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
PS D:\AI Coding
```

2. Email Priority Classification

Scenario:

A company wants to automatically prioritize incoming emails into High Priority, Medium Priority, or Low Priority

2a) Create 6 sample email messages with priority labels.

The screenshot shows the Visual Studio Code (VS Code) interface. The Explorer sidebar on the left lists files and folders, including several Python files like 'add.py', 'AI lab43.py', and 'lab assignment 44.py'. The 'TERMINAL' tab is selected at the bottom, displaying the output of a Python script. The script reads a list of emails from a list comprehension and prints them with their priority levels.

```
9 #=====
10 emails = [
11     {"email": "Server is down and business operations stopped", "priority": "High"},
12     {"email": "Client meeting scheduled for tomorrow", "priority": "High"},
13     {"email": "Please review the attached report when free", "priority": "Medium"},
14     {"email": "Need update on project status", "priority": "Medium"},
15     {"email": "Team lunch invitation", "priority": "Low"},
16     {"email": "Newsletter subscription confirmation", "priority": "Low"},
17 ]
18
19 for email in emails:
20     print(f"Email: {email['email']} | Priority: {email['priority']}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Traceback (most recent call last):
  File "d:\AI Coding\lab assignment 44.py", line 1, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
PS D:\AI Coding> & c:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Email: Server is down and business operations stopped | Priority: High
Email: Client meeting scheduled for tomorrow | Priority: High
Email: Please review the attached report when free | Priority: Medium
Email: Need update on project status | Priority: Medium
Email: Team lunch invitation | Priority: Low
Email: Newsletter subscription confirmation | Priority: Low
PS D:\AI Coding>
```

2b) Perform intent classification using Zero-shot prompting

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows various files including `assignment 3.py`, `AI lab43.py`, and `lab assignment 44.py`.
- Code Editor:** Displays the code for `lab assignment 44.py`. The last line of code is highlighted in green: `priority = "High"`.
- Terminal:** Shows the command line output:

```
File "d:\VAI Coding\lab assignment 44.py", line 1, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
PS D:\VAI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Email: Server is down and business operations stopped | Priority: High
Email: Client meeting scheduled for tomorrow | Priority: High
Email: Please review the attached report when free | Priority: Medium
Email: Need update on project status | Priority: Medium
Email: Team lunch invitation | Priority: Low
Email: Newsletter subscription confirmation | Priority: Low
PS D:\VAI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
PS D:\VAI Coding>
```

2c) Perform classification using One-shot prompting

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows various files including `assignment 3.py`, `AI lab43.py`, and `lab assignment 44.py`.
- Code Editor:** Displays the code for `lab assignment 44.py`. The last line of code is highlighted in green: `priority = "Normal" # Set the priority level`.
- Terminal:** Shows the command line output:

```
ModuleNotFoundError: No module named 'pandas'
PS D:\VAI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Email: Server is down and business operations stopped | Priority: High
Email: Client meeting scheduled for tomorrow | Priority: High
Email: Please review the attached report when free | Priority: Medium
Email: Need update on project status | Priority: Medium
Email: Team lunch invitation | Priority: Low
Email: Newsletter subscription confirmation | Priority: Low
PS D:\VAI Coding>
```

2d) Perform classification using Few-shot prompting.

```

File Edit Selection View Go Run Terminal Help < > AI Coding
EXPLORER Welcome assignment 3.py AI lab43.py lab assignment 44.py
AI CODING
  add.py
  AI lab43.py
  Assignment1(CP).pdf
  Assignment 2-4.pdf
  assignment 3.4
  assignment 3.py
  Assignment2.pdf
  assignment3.4.docx
  factorial.py
  jobs.py
  jobsqc.py
  lab assignment 44.py
  lab assignment3.3.pdf
  lab1 HCP.pdf
  matrixHCP.py
  Untitled20.ipynb
  week2 HCP.pdf

lab assignment 44.py > ...
12 def determine_priority(email_subject):
13     high_priority_keywords = ["outage", "urgent", "immediate"]
14     medium_priority_keywords = ["feedback", "proposal", "review"]
15     low_priority_keywords = ["invitation", "party", "meeting"]
16
17     subject_lower = email_subject.lower()
18
19     if any(keyword in subject_lower for keyword in high_priority_keywords):
20         return "High"
21     elif any(keyword in subject_lower for keyword in medium_priority_keywords):
22         return "Medium"
23     elif any(keyword in subject_lower for keyword in low_priority_keywords):
24         return "Low"
25     else:
26         return "Low" # Default priority
27
28 # Example usage
29 email_subject = "Client meeting scheduled for tomorrow"
30 priority = determine_priority(email_subject)
31 print(f"Email: '{email_subject}'\nPriority: {priority}")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

Email: Client meeting scheduled for tomorrow | Priority: High
Email: Please review the attached report when free | Priority: Medium
Email: Need update on project status | Priority: Medium
Email: Team lunch invitation | Priority: Low
Email: Newsletter subscription confirmation | Priority: Low
PS D:\AI Coding & C:/Users/ANALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
PS D:\AI Coding & C:/Users/ANALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
PS D:\AI Coding & C:/Users/ANALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
PS D:\AI Coding & C:/Users/ANALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Email: "Client meeting scheduled for tomorrow"
Priority: Low
PS D:\AI Coding>

```

> OUTLINE > TIMELINE PS D:\AI Coding>

2e) Evaluate which technique produces the most reliable results and why.

```

File Edit Selection View Go Run Terminal Help < > AI Coding
EXPLORER Welcome assignment 3.py AI lab43.py lab assignment 44.py
AI CODING
  add.py
  AI lab43.py
  Assignment1(CP).pdf
  Assignment 2-4.pdf
  assignment 3.4
  assignment 3.py
  Assignment2.pdf
  assignment3.4.docx
  factorial.py
  jobs.py
  jobsqc.py
  lab assignment 44.py
  lab assignment3.3.pdf
  lab1 HCP.pdf
  matrixHCP.py
  Untitled20.ipynb
  week2 HCP.pdf

lab assignment 44.py > ...
13     client = openai.Client()
14
15     def zero_shot_classification(text, categories):
16         prompt = f"Classify the following text into one of these categories: {', '.join(categories)}.\nText: {text}\nCategory:"
17         response = client.chat.completions.create(
18             model="gpt-3.5-turbo",
19             messages=[{"role": "user", "content": prompt}],
20             temperature=0)
21
22         return response.choices[0].message.content.strip()
23
24     def one_shot_classification(text, categories, example_text, example_category):
25
26         prompt = f"Classify text into categories: {', '.join(categories)}.\nExample: {example_text}\nCategory: {example_category}\nNow classify:\nText: {text}\nCategory:"
27         response = client.chat.completions.create(
28             model="gpt-3.5-turbo",
29             messages=[{"role": "user", "content": prompt}],
30             temperature=0)
31
32         return response.choices[0].message.content.strip()
33
34     def few_shot_classification(text, categories, examples):
35
36         prompt = f"Classify text with multiple examples: {', '.join(categories)}\nCategory: {examples[0]['category']}" + "\n" + examples[0]['text'] + "\nCategory: " + examples[0]['category']
37         for ex in examples[1:]:
38             prompt += "\nCategory: {ex['category']} " + "\n" + ex['text'] + "\nCategory: " + ex['category']
39
40         response = client.chat.completions.create(
41             model="gpt-3.5-turbo",
42             messages=[{"role": "user", "content": prompt}],
43             temperature=0)
44
45         return response.choices[0].message.content.strip()
46
47     def evaluate_technique(texts, categories, ground_truth, examples):
48
49         # Compute all three techniques
50         results = {"zero_shot": [], "one_shot": [], "few_shot": []}
51
52         for text, true_label in zip(texts, ground_truth):
53             results["zero_shot"].append(zero_shot_classification(text, categories))
54             results["one_shot"].append(one_shot_classification(text, categories, examples[0]["text"], examples[0]["category"]))
55             results["few_shot"].append(few_shot_classification(text, categories, examples))
56
57         # Calculate accuracy
58         accuracies = []
59         for pred, true in zip(results["one_shot"], ground_truth):
60             correct = sum(1 for pred, true in zip(predictions, ground_truth) if pred.lower() == true.lower())
61             accuracies.append(correct / len(ground_truth))
62
63         return results, accuracies
64
65     # Example usage
66
67     categories = ["Positive", "Negative", "Neutral"]
68
69     texts = [
70         "The food was delicious!",
71         "I really enjoyed the movie.",
72         "The service was terrible.",
73         "The product did not work as expected."
74     ]
75
76     ground_truth = ["Positive", "Positive", "Negative", "Negative"]
77
78     results, accuracies = evaluate_technique(texts, categories, ground_truth, examples)
79
80     print("Results for Zero-Shot Classification:")
81     for text, pred in zip(texts, results["zero_shot"]):
82         print(f"Text: {text} - Prediction: {pred}")
83
84     print("\nResults for One-Shot Classification:")
85     for text, pred in zip(texts, results["one_shot"]):
86         print(f"Text: {text} - Prediction: {pred}")
87
88     print("\nResults for Few-Shot Classification:")
89     for text, pred in zip(texts, results["few_shot"]):
90         print(f"Text: {text} - Prediction: {pred}")
91
92     print("\nAccuracy Results:")
93     for category, accuracy in zip(categories, accuracies):
94         print(f"Category: {category} - Accuracy: {accuracy:.2f}")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

Email: Team lunch invitation | Priority: Low
Email: Client meeting scheduled for tomorrow | Priority: Low
PS D:\AI Coding & C:/Users/ANALI/AppData/Local/Programs/Python/Python311/python.exe "d:/AI Coding/lab assignment 44.py"
PS D:\AI Coding & C:/Users/ANALI/AppData/Local/Programs/Python/Python311/python.exe "d:/AI Coding/lab assignment 44.py"
PS D:\AI Coding & C:/Users/ANALI/AppData/Local/Programs/Python/Python311/python.exe "d:/AI Coding/lab assignment 44.py"
PS D:\AI Coding & C:/Users/ANALI/AppData/Local/Programs/Python/Python311/python.exe "d:/AI Coding/lab assignment 44.py"
Email: "Client meeting scheduled for tomorrow"
Priority: Low
PS D:\AI Coding>

```

> OUTLINE > TIMELINE PS D:\AI Coding>

3. Student Query Routing System

Scenario:

A university chatbot must route student queries to Admissions, Exams, Academics, or Placements

3a) . Create 6 sample student queries mapped to departments.

The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows files in the "AI CODING" folder, including "add.py", "AI lab43.py", "Assignment1(CP).pdf", "Assignment 2-4.pdf", "assignment 3.4", "assignment 3.py", "Assignment2.pdf", "assignment3.4.docx", "factorial.py", "jobs.py", "jobs.py", "lab assignment ...", "lab assignment3.3.pdf", "lab1 HCP.pdf", "matrixHCP.py", "Untitled20.ipynb", and "week2 HCP.pdf".
- Code Editor:** Displays a Python script named "lab assignment 44.py". The code defines a dictionary `student_queries` mapping student queries to department names. It then iterates through the items in the dictionary to print each query and its corresponding department.

```
student_queries = {  
    "How do I register for courses?": "Registrar",  
    "What is my current GPA?": "Admissions",  
    "I need to pay my tuition": "Finance",  
    "Can I get a transcript?": "Registrar",  
    "I'm having trouble with my financial aid": "Finance",  
    "How do I declare a major?": "Admissions"  
}  
  
for query, department in student_queries.items():  
    print(f"Query: {query}")  
    print(f"Department: {department}\n")
```

- Terminal:** Shows the output of the script execution:
 - Query: How do I register for courses?
Department: Registrar
 - Query: What is my current GPA?
Department: Admissions
 - Query: I need to pay my tuition
Department: Finance
 - Query: Can I get a transcript?
Department: Registrar
- Bottom Navigation:** Includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS.

3b) Implement Zero-shot intent classification using an LLM.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows a folder named "AI CODING" containing files like add.py, AI lab43.py, Assignment1(CP).pdf, Assignment 2-4.pdf, assignment 3.4, assignment 3.py, Assignment2.pdf, assignment3.4.docx, factorial.py, jobs.py, jobsccp.py, lab assignment ..., lab assignment3.3.pdf, lab1 HCP.pdf, matrixHCP.py, Untitled20.ipynb, and week2 HCP.pdf.
- Code Editor (Center):** Displays a Python script with the following code:

```
1  #
2  if department == "Registrar":
3      classification = "Academics"
4  elif department == "Admissions":
5      classification = "Admissions"
6  elif department == "Finance":
7      classification = "Placements"
8  else:
9      classification = "Exams"
10 print(f"Classification: {classification}\n")
```
- Terminal (Bottom):** Shows the output of running the script with two different queries:

```
Query: I'm having trouble with my financial aid
Department: Finance
Classification: Placements

Query: How do I declare a major?
Department: Admissions
Classification: Admissions
```

3c) Improve results using One-shot prompting.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder named "AI CODING" containing files like add.py, AI lab43.py, Assignment1(CP).pdf, Assignment 3.4.pdf, assignment 3.4, assignment3.4.docx, factorial.py, jobs.py, jobsccp.py, lab assignment ..., lab assignment3.3.pdf, lab1 HCP.pdf, matrixHCP.py, Untitled20.ipynb, and week2 HCP.pdf.
- Code Editor:** Displays a Python script named "lab assignment 44.py". The code defines a function `classify_query` that takes a query string and returns the department name based on keyword matches. It includes test cases at the bottom.
- Terminal:** Shows the output of running the script with several test queries. The output includes:
 - Department: Admissions
 - Classification: Admissions
 - Query: "When will results be announced?"
Department: Exams
 - Query: "Explain syllabus for Data Structures"
Department: Academics
 - Query: "How do I apply for admission?"
- Bottom Bar:** Includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is selected), and PORTS.

3d) Further refine results using Few-shot prompting.

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The top bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar for "AI Coding". The Explorer sidebar on the left lists files and folders under "AI CODING", such as add.py, AI lab43.py, Assignment1(CP).pdf, assignment 3.4, assignment 3.py, Assignment2.pdf, assignment3.4.docx, factorial.py, jobs.py, jobscc.py, lab assignment ..., lab assignment3.3.pdf, lab1 HCP.pdf, matrixHCP.py, Untitled20.ipynb, and week2 HCP.pdf. The main editor area displays a Python script named "lab assignment 44.py". The code defines a function "classify_query" that takes a query string and returns the department name based on keywords. It includes test cases for Admissions, Exams, and Placements. The terminal at the bottom shows the command "python.exe" being run and a "ModuleNotFoundError" for "openai".

```
def classify_query(query):
    """
    Classifies a query to the appropriate department.

    Args:
        query (str): The user's query

    Returns:
        str: The department name
    """
    query_lower = query.lower()

    # Define keywords for each department
    departments = {
        "Admissions": ["admission", "deadline", "apply", "enrollment", "registration"],
        "Exams": ["exam", "fee", "payment", "test", "marks", "results"],
        "Placements": ["job", "placement", "campus", "opportunity", "recruit", "interview"]
    }

    # Check which department matches the query
    for dept, keywords in departments.items():
        if any(keyword in query_lower for keyword in keywords):
            return dept

    return "Unknown" # Default if no match found

# Test cases
print(classify_query("Admission deadline details")) # Admissions
print(classify_query("Exam fee payment date")) # Exams
print(classify_query("Job opportunities through campus")) # Placements
print(classify_query("How to apply for campus placements?")) # Placements
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Traceback (most recent call last):
  File "d:\AI Coding\lab assignment 44.py", line 1, in <module>
    import openai
ModuleNotFoundError: No module named 'openai'
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Admissions
Exams
Placements
Admissions
PS D:\AI Coding>
```

3e) Analyze how contextual examples affect classification accuracy.

The screenshot shows the Visual Studio Code (VS Code) interface with the "AI Coding" extension installed. The left sidebar displays a file tree under the "EXPLORER" tab, showing various files including Python scripts (add.py, lab assignment 34.py, lab assignment 44.py), PDFs (Assignment 1(CP).pdf, Assignment 2-4.pdf, assignment 3.4, assignment 3.py, Assignment2.pdf, assignment3.4.docx, factorial.py, jobs.py, jobscc.py), and Jupyter notebooks (Untitled20.ipynb, week2 HCP.pdf). The main editor area shows a Python script named "lab assignment 44.py". The script defines a class "ClassificationAnalyzer" with methods for zero-shot and one-shot classification prompts. Below the code, there are sections for "Ambiguity", "Consistency", and "FEW_SHOT" with their respective pros and cons. The bottom status bar shows the path "PS D:\AI Coding".

```
from collections import defaultdict
import json

class ClassificationAnalyzer:
    def __init__(self):
        self.results = defaultdict(list)

    def zero_shot_prompt(self, text, categories):
        """
        Zero-shot: No examples provided
        """
        prompt = f"""Classify the following text into one of these categories: {', '.join(categories)}\nText: {text}\nCategory:"""
        return prompt

    def one_shot_prompt(self, text, categories, example_text, example_category):
        """
        One-shot: Single example provided
        """
        prompt = f"""Classify text into categories: {', '.join(categories)}\nExample:\nText: {example_text}\nCategory: {example_category}"""
        return prompt
```

Ambiguity: Reduced - one example clarifies intent
Consistency: Improved - example sets pattern
Pros: Minimal overhead, Some context
Cons: Limited learning from one example

FEW_SHOT:
Accuracy: Higher - multiple references provided
Ambiguity: Significantly reduced - pattern clear
Consistency: High - multiple examples establish standard
Pros: Best accuracy, Clear patterns, Reduced errors
Cons: Requires manual examples, Prompt size

4) Chatbot Question Type Detection

Scenario:

A chatbot must identify whether a user query is Informational, Transactional, Complaint, or Feedback.

4a) Prepare 6 chatbot queries mapped to question types.

```

File Edit Selection View Go Run Terminal Help < > Q AI Coding
EXPLORER ... Welcome assignment 3.py AI lab43.py lab assignment 44.py X
AI CODING
+ add.py
+ AI lab43.py
- Assignment1(CP).pdf
- Assignment 2-4.pdf
- Assignment 3-4.py
- Assignment2.pdf
- assignment3.4.docx
- factorial.py
- jobs.py
- jobscp.py
+ lab assignment ...
- lab assignment3.3.pdf
- lab1 HCP.pdf
+ matrixHCP.py
+ Untitled2.ipynb
- week2 HCP.pdf

Generate code
@ Add Context...
1 def classify_query(query):
2     """
3         Classify a query into one of four types:
4             - Informational
5             - Transactional
6             - Complaint
7             - Feedback
8     """
9     query_lower = query.lower()
10
11     # Define keywords for each category
12     informational_keywords = ['what', 'how', 'when', 'where', 'why', 'working hours', 'reset', 'password']
13     transactional_keywords = ['book', 'buy', 'order', 'purchase', 'reserve', 'ticket']
14     complaint_keywords = ['hasn\'t', 'didn\'t', 'broken', 'bad', 'poor', 'issue', 'problem', 'arrived']
15     feedback_keywords = ['great', 'good', 'excellent', 'user-friendly', 'nice', 'love', 'hate', 'experience']
16
17     # Check for complaint (highest priority)
18     if any(keyword in query_lower for keyword in complaint_keywords):
19         return 'Complaint'
20
21     # Check for transactional
22     if any(keyword in query_lower for keyword in transactional_keywords):
23         return 'Transactional'
24
25     # Check for feedback
26     if any(keyword in query_lower for keyword in feedback_keywords):
27         return 'Feedback'
28
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Ambiguity: Significantly reduced - pattern clear
Consistency: High - multiple examples establish standard
Pros: Best accuracy. Clear outcomes, Reduced errors
Cons: Requires manual examples, Prompt size
PS D:\AI Coding & C:\Users\NDALI\AppData\Local\Programs\Python\Python313\python.exe "d:/AI Coding/lab assignment 44.py"
What are your working hours? | Informational
I want to book a ticket | Transactional
My order hasn't arrived yet | Complaint
The app is very user-friendly | Feedback
How can I reset my password? | Informational
The service experience was bad | Complaint
PS D:\AI Coding

```

4b) Design prompts for Zero-shot, One-shot, and Few-shot learning.

```

File Edit Selection View Go Run Terminal Help < > Q AI Coding
EXPLORER ... Welcome assignment 3.py AI lab43.py lab assignment 44.py X
AI CODING
+ add.py
+ AI lab43.py
- Assignment1(CP).pdf
- Assignment 2-4.pdf
- Assignment 3-4.py
- Assignment2.pdf
- assignment3.4.docx
- factorial.py
- jobs.py
- jobscp.py
+ lab assignment ...
- lab assignment3.3.pdf
- lab1 HCP.pdf
+ matrixHCP.py
+ Untitled2.ipynb
- week2 HCP.pdf

Generate code
@ Add Context...
1 # Query Classification System
2
3 def classify_query(query):
4     """
5         Classify a user query into one of four categories:
6             - Informational: User seeking information
7             - Transactional: User wanting to perform an action/transaction
8             - Complaint: User expressing dissatisfaction
9             - Feedback: User providing feedback or suggestions
10
11     query_lower = query.lower()
12
13     # Transactional keywords
14     transactional_keywords = ['book', 'buy', 'purchase', 'order', 'reserve', 'checkout', 'pay']
15
16     # Complaint keywords
17     complaint_keywords = ['problem', 'issue', 'broken', 'error', 'complaint', 'wrong', 'not working']
18
19     # Feedback keywords
20     feedback_keywords = ['feedback', 'suggest', 'idea', 'improve', 'opinion', 'review']
21
22     # Informational keywords
23     informational_keywords = ['how', 'what', 'when', 'where', 'why', 'can you tell', 'help', 'information']
24
25     # Classify based on keywords
26     if any(keyword in query_lower for keyword in transactional_keywords):
27         return "Transactional"
28
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Cons: Requires manual examples, Prompt size
PS D:\AI Coding & C:\Users\NDALI\AppData\Local\Programs\Python\Python313\python.exe "d:/AI Coding/lab assignment 44.py"
What are your working hours? | Informational
I want to book a ticket | Transactional
My order hasn't arrived yet | Complaint
The app is very user-friendly | Feedback
How can I reset my password? | Informational
The service experience was bad | Complaint
PS D:\AI Coding & C:\Users\NDALI\AppData\Local\Programs\Python\Python313\python.exe "d:/AI Coding/lab assignment 44.py"
Query: I want to book a ticket
Type: Transactional
PS D:\AI Coding

```

4c) Test all prompts on the same unseen queries.

The screenshot shows the VS Code interface with the AI Coding extension active. The terminal window displays the following test results:

```
How can I reset my password? | Informational
The service experience was bad | Complaint
Query: I want to book a ticket | Transactional
Type: Transactional
PS D:\AI Coding & C:/users/ANDALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Query: "The service experience was bad"
Type: Complaint

Query: "The app is very user-friendly"
Type: Compliment
PS D:\AI Coding
```

4d) Compare response correctness and ambiguity handling.

The screenshot shows the VS Code interface with the AI Coding extension active. The terminal window displays the following test results, comparing the new classifier function with the previous one:

```
# This function classifies user queries into different types
def classify_query(query):
    if "book" in query.lower() or "reserve" in query.lower():
        return "Transactional"
    elif "order" in query.lower() or "arrived" in query.lower():
        return "Complaint"
    elif "support" in query.lower() or "great" in query.lower():
        return "Feedback"
    else:
        return "Informational"

# Example usage
query = "I want to book a ticket"
query_type = classify_query(query)
print(f"Query: '{query}'\nType: {query_type}")

Query: I want to book a ticket
Type: Transactional
PS D:\AI Coding & C:/users/ANDALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Query: "The service experience was bad"
Type: Complaint

Query: "The app is very user-friendly"
Type: Compliment
PS D:\AI Coding
```

4e) Document observations.

```
1 /**
2  * Observations from testing Zero-shot, One-shot, and Few-shot prompting for classification tasks:
3  */
4
5 1. Accuracy:
6     - Zero-shot prompting often resulted in lower accuracy compared to One-shot and Few-shot methods, as the model had no prior examples to reference.
7     - One-shot prompting showed improved accuracy, as the model could leverage a single example to understand the task better.
8     - Few-shot prompting consistently yielded the highest accuracy, as multiple examples provided the model with a clearer context and better understanding of the classification task.
9
10 2. Ambiguity Handling:
11     - Zero-shot prompting struggled with ambiguous inputs, often leading to incorrect classifications due to lack of context.
12     - One-shot prompting reduced ambiguity to some extent, as the provided example helped clarify the task, but some ambiguity remained.
13     - Few-shot prompting effectively handled ambiguity by providing multiple examples that illustrated different aspects of the classification task, allowing the model to make more informed decisions.
14
15 3. Consistency:
16     - Zero-shot prompting exhibited high variability in results, with performance heavily dependent on the specific input phrasing.
17     - One-shot prompting showed moderate consistency, as the single example could guide the model, but variations in input still affected outcomes.
18     - Few-shot prompting demonstrated the highest consistency across different inputs, as the multiple examples helped stabilize the model's responses.
19
20 4. Overall Performance Differences:
21     - Overall, Few-shot prompting outperformed both Zero-shot and One-shot methods in terms of accuracy, ambiguity handling, and consistency.
22     - Zero-shot prompting was useful for quick assessments but lacked reliability for critical tasks.
23     - One-shot prompting served as a middle ground, offering some improvements but still falling short of Few-shot performance.
24
25 /**
26  * 
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Type: Transactional
PS D:\AI Coding & C:/Users/ANDALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Query: "The service experience was bad"
Type: Complaint
Query: "The app is very user-friendly"
Type: Compliment
PS D:\AI Coding & C:/Users/ANDALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Query: "I want to book a ticket."
Type: Transactional
PS D:\AI Coding & C:/Users/ANDALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
PS D:\AI Coding>

5) Emotion Detection in Text

Scenario:

A mental-health chatbot needs to detect emotions: Happy, Sad, Angry, Anxious, Neutral.

5a) Create labeled emotion samples.

The screenshot shows a dark-themed VS Code interface. In the center, there's a code editor window titled "lab assignment 44.py". The code imports pandas and defines a DataFrame from a list of dictionaries. The list contains two main keys: "Text" and "Emotion", each with a list of strings. Below the code editor is a terminal window showing the execution of the script and a moduleNotFoundError for 'pandas'. The left sidebar shows a file tree with various files like "add.py", "Assignment 1(CP).pdf", and "Assignment 2-4.pdf".

```
1 import pandas as pd
2
3 # Create a DataFrame from the provided data
4 data = [
5     "Text": [
6         "I am very happy today",
7         "I feel lonely and depressed",
8         "This is so frustrating",
9         "I am worried about my future",
10        "Today is just normal",
11        "Feeling excited about results"
12    ],
13    "Emotion": [
14        "Happy",
15        "Sad",
16        "Angry",
17        "Anxious",
18        "Neutral",
19        "Happy"
20    ]
21 }
22
23 df = pd.DataFrame(data)
24
25 # Display the DataFrame
26 print(df)
```

```
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Traceback (most recent call last):
  File "d:/AI Coding/lab assignment 44.py", line 1, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Traceback (most recent call last):
  File "d:/AI Coding/lab assignment 44.py", line 1, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
PS D:\AI Coding>
```

5b) Use Zero-shot prompting to identify emotions.

This screenshot shows the same VS Code environment as the previous one, but the code in the editor has been modified. It defines a function "identify_emotion" that checks if the input text contains the word "worried" and returns "Anxious" or "Neutral" accordingly. The terminal shows the execution of the script and the output "Emotion: Anxious".

```
1 def identify_emotion(text):
2     if "worried" in text:
3         return "Anxious"
4     return "Neutral"
5
6 text = "I am worried about my future"
7 emotion = identify_emotion(text)
8 print(f"Emotion: {emotion}")
```

```
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Traceback (most recent call last):
  File "d:/AI Coding/lab assignment 44.py", line 1, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Traceback (most recent call last):
  File "d:/AI Coding/lab assignment 44.py", line 1, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
PS D:\AI Coding> Emotion: Anxious
PS D:\AI Coding>
```

5c) Use One-shot prompting with an example.

The screenshot shows the VS Code interface with the following details:

- EXPLORER** view: Shows files like `add.py`, `AI lab43.py`, `Assignment 2-4.pdf`, `Assignment 3-4.docx`, `assignment 3.py`, `Assignment2.pdf`, `factorial.py`, `jobs.py`, `jobscp.py`, `lab assignment ...`, `lab assignment3-3.pdf`, `lab1 HCP.pdf`, `matrixHCP.py`, `Untitled20.ipynb`, and `week2 HCP.pdf`.
- CODE** view: A code editor window titled "lab assignment 44.py" with the following Python code:

```
1 def identify_emotion(text):
2     if "frustrating" in text:
3         return "Frustrated"
4     return "Neutral"
5
6 # Example usage
7 text = "This is so frustrating"
8 emotion = identify_emotion(text)
9 print(f"Emotion: {emotion}")
```
- TERMINAL** view: Shows a terminal window with the following output:

```
import pandas as pd
ModuleNotFoundError: No module named 'pandas'
PS D:\AI Coding & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Traceback (most recent call last):
File "d:/AI Coding/lab assignment 44.py", line 1, in <module>
    Import pandas as pd
ModuleNotFoundError: No module named 'pandas'
PS D:\AI Coding & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Emotion: Anxious
PS D:\AI Coding & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Emotion: Frustrated
PS D:\AI Coding
```

5d) Use Few-shot prompting with multiple emotions.

The screenshot shows the VS Code interface with the following details:

- EXPLORER** view: Shows files like `add.py`, `AI lab43.py`, `Assignment 2-4.pdf`, `Assignment 3-4.docx`, `assignment 3.py`, `Assignment2.pdf`, `factorial.py`, `jobs.py`, `jobscp.py`, `lab assignment ...`, `lab assignment3-3.pdf`, `lab1 HCP.pdf`, `matrixHCP.py`, `Untitled20.ipynb`, and `week2 HCP.pdf`.
- CODE** view: A code editor window titled "lab assignment 44.py" with the following Python code:

```
1 def classify_emotion(text):
2     emotions = {
3         "happy": ["happy", "joyful", "excited", "pleased"],
4         "sad": ["lonely", "depressed", "sad", "down"],
5         "anxious": ["worried", "anxious", "nervous", "stressed"],
6         "neutral": ["normal", "fine", "okay", "average"],
7         "frustrated": ["frustrating", "annoyed", "irritated"]
8     }
9
10    for emotion, keywords in emotions.items():
11        if any(keyword in text.lower() for keyword in keywords):
12            return emotion
13    return "Unknown"
14
15 # Example usage
16 text = "This is so frustrating"
17 emotion = classify_emotion(text)
18 print(f"Text: \'{text}\'\nEmotion: {emotion}")
```
- TERMINAL** view: Shows a terminal window with the following output:

```
Traceback (most recent call last):
File "d:/AI Coding/lab assignment 44.py", line 1, in <module>
    Import pandas as pd
ModuleNotFoundError: No module named 'pandas'
PS D:\AI Coding & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Emotion: Anxious
PS D:\AI Coding & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Emotion: Frustrated
PS D:\AI Coding & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Text: "This is so frustrating"
Emotion: frustrated
PS D:\AI Coding
```

5e) Discuss ambiguity handling across techniques.

The screenshot shows a code editor interface with the following details:

- EXPLORER** panel on the left lists files and folders including `add.py`, `AI lab43.py`, `Assignment 2-4.pdf`, `Assignment 3-4.docx`, `assignment 3-4.py`, `Assignment2.pdf`, `factorial.py`, `jobs.py`, `jobsCopy.py`, `lab assignment ...`, `lab assignment3-3.pdf`, `lab1 HCP.pdf`, `matrixHCP.py`, and `Untitled2.ipynb`.
- CODE** tab is active, displaying the following Python code:

```
1 # Emotion Handling Techniques
2
3 def handle_emotion(technique, input_text):
4     if technique == "zero-shot":
5         return "This technique struggles with ambiguity in understanding emotions."
6     elif technique == "one-shot":
7         return "This technique provides better clarity in emotional interpretation."
8     elif technique == "few-shot":
9         return "This technique achieves the best emotional accuracy by learning from examples."
10    else:
11        return "Unknown technique."
12
13 # Example usage
14 techniques = ["zero-shot", "one-shot", "few-shot"]
15 for technique in techniques:
16     print(f"{technique.capitalize()}: {handle_emotion(technique, '')}")
```

- TERMINAL** tab at the bottom shows command-line output:

```
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Emotion: Anxious
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Emotion: Frustrated
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Text: This is frustrating
Emotion: Frustrated
PS D:\AI Coding> & C:/Users/ANJALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"
Zero-shot: This technique struggles with ambiguity in understanding emotions.
One-shot: This technique provides better clarity in emotional interpretation.
Few-shot: This technique achieves the best emotional accuracy by learning from examples.
PS D:\AI Coding>
```