

Assignment 4.4

1. Sentiment Classification for Customer Reviews

Scenario:

An e-commerce platform wants to analyze customer reviews and classify

Week2

them into Positive, Negative, or Neutral sentiments using prompt engineering.

Tasks:

- Prepare 6 short customer reviews mapped to sentiment labels.
- Design a Zero-shot prompt to classify sentiment.
- Design a One-shot prompt with one labeled example.
- Design a Few-shot prompt with 3–5 labeled examples.
- Compare the outputs and discuss accuracy differences.

The screenshot shows a code editor interface with several tabs and panes. The main pane displays a Python script named `ass 4.4.py`. The code reads reviews from a file and prints them along with their sentiment labels. A sidebar provides a summary of the reviews. The bottom pane shows the terminal output of the script running on a local file.

```
File Edit Selection View Go Run Terminal Help <- > AI Assisted
EXPLORER ass 4.4.py
AI ASSISTED
ass 4.4.py
ass 4.4.pdf
ass 3.4.pdf
lab 4.3.pdf
lab 4.3.py
lab 4.3.pdf
lab assignment 3.3.pdf
lab assignment 1.4.pdf
lab assignment 2.3.pdf
lab.pdf
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Review #1
Text: Terrible experience. Item arrived damaged and customer service was unhelpful.
Sentiment: Negative
-----
Review #2
Text: Product arrived on time. It works as described.
Sentiment: Neutral
-----
Review #3
Text: Love it! Exceeded my expectations and great value for money.
Sentiment: Positive
-----
Review #4
Text: Love it! Exceeded my expectations and great value for money.
Sentiment: Positive
-----
Review #5
Text: Not satisfied. Poor packaging and item doesn't match the description.
Sentiment: Negative
-----
Review #6
Text: It's okay. Does what it's supposed to do, nothing special.
Sentiment: Neutral
-----
Summary:
Positive: 2 | Negative: 2 | Neutral: 2
P: C:\Users\varia\OneDrive\Desktop\AI Assisted
Space 4 UTM-B Python 3.13.7 1/29/2026
28°C Sunny
```

Output pane:

```
AI Assisted + v - i x
SENTIMENT CLASSIFICATION FOR CUST... [Python]
Generate Python code that creates a list of 6 short e-commerce customer reviews. Each review should be mapped to a sentiment label Positive, Negative, or Neutral. Print the review and their labels in a clear format.
Basic Ad
Used 1 reference
# List of e-commerce customer reviews
reviews = [
    {
        "review": "The product is great!",
        "sentiment": "Positive"
    },
    {
        "review": "Terrible experience.",
        "sentiment": "Negative"
    },
    {
        "review": "The product arrived quickly.",
        "sentiment": "Neutral"
    },
    {
        "review": "Love it! Excellent value for the price."
        "sentiment": "Positive"
    },
    {
        "review": "Not satisfied with the delivery time."
        "sentiment": "Negative"
    },
    {
        "review": "It's okay. Does what it's supposed to do, nothing special."
        "sentiment": "Neutral"
    }
]

# Print reviews with sentiment labels
print(len(reviews))
print("Commerce Customer Reviews")
print(len(reviews))
```

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with the following details:

- File Explorer:** Shows files in the 'AI Assisted' folder, including 'ass-4.4.py', 'ass-4.4.pdf', 'ass-3.4.pdf', 'check_leap_array', 'lao-4.3.word.docx', 'lao-4.3.pdf', 'lao-3.4.pdf', 'lao assignment 3.3.pdf', 'lao assignment 1-2.pdf', 'lao assignment 2-3.pdf', and 'lisp_star.py'.
- Code Editor:** Displays the content of 'ass-4.4.py'. The code defines a function 'classify_sentiment' that takes a review as input and returns 'Positive', 'Negative', or 'Neutral' based on keyword counts. It also includes a test loop where it prompts for user input, classifies the review, and prints the result.
- Terminal:** Shows the command line output of running the script, indicating the classification of reviews as Positive, Negative, or Neutral.
- Output:** Shows the results of the 'Review #5' test.
- Search:** Shows the search term 'AI Assisted'.
- Chat Panel:** Titled 'SENTIMENT CLASSIFICATION FOR CUSTOMER REVIEWS', it contains a message from AI Assisted asking for a Python code snippet to classify customer reviews using simple keyword logic. It also includes a note to avoid using machine learning libraries.
- Code Snippet:** A snippet titled 'Used 1 reference' is shown, which is a copy of the 'ass-4.4.py' code.
- Code Hover:** A tooltip for the word 'review' in the code editor shows its definition as 'review.lower()'.
- Code Completion:** A tooltip for the word 'positive' shows completion suggestions like 'positive', 'positive_1', 'positive_2', etc.
- Code Coverage:** A tooltip for the word 'negative' shows coverage statistics: 'positive_count = 0' for Positive, 'negative_count = 0' for Negative, and 'neutral_count = 0' for Neutral.
- Code Help:** A tooltip for the word 'sentiment' shows its documentation: 'Classifies sentiment of a review. Returns: Positive, Negative, ...'. It also shows the definition of 'review.lower()'. A note at the bottom says 'Explore and understand your code'.

A screenshot of Microsoft Visual Studio Code (VS Code) interface. The title bar includes 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', 'Terminal', 'Help' and a search bar 'AI Assisted'. The left sidebar shows 'EXPLORER' with files like 'ai-assist.ipynb', 'check_leap_year.ipynb', 'lap-4.3.ipynb', 'lap-4.3.py', 'lap-4.3.word.docx', 'lap-4.3.year.ipynb', 'lap-4.3.year.pdf', 'lap-assignment-1.ipynb', 'lap-assignment-1.3.ipynb', 'lap-assignment-1.3.pdf', 'lap-assignment-1.3.year.ipynb', and 'lap-assignment-1.3.year.pdf'. The main editor area displays Python code for sentiment analysis, with syntax highlighting and code completion suggestions. A status bar at the bottom shows '28°C Sunny' and system icons.

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with the following details:

- File Explorer:** Shows files like `AI Assisted`, `add.py`, `AI ass Ass 1.pdf`, `ass 3 4.pdf`, `ass 4.pdf`, `chatgpt.logz.json`, `code 4.4.pdf`, `do 4.3.pdf`, `do 4.4.pdf`, `do assignment 1-4.pdf`, `do assignment 3.3.pdf`, `do assignment 3-4.pdf`, `do assignment 3-5.pdf`, and `do leap year.py`.
- Code Editor:** Displays Python code for a "Customer Review Sentiment Classifier". The code includes imports, definitions for `sentiment` and `analyze_reviews` functions, and examples of sentiment classification for various reviews. It also includes a loop for user input and classification.
- Terminal:** Shows the command `python add.py` being run, with output indicating the script is executing.
- ChatGPT AI Assistant:** A sidebar titled "SENTIMENT CLASSIFICATION FOR CUST..." containing a conversation with ChatGPT about sentiment analysis. It includes examples of positive, negative, and neutral reviews, and a note about using three examples to guide the logic.
- Bottom Status Bar:** Shows file paths like `C:\Users\parva\OneDrive\Desktop\AI Assisted\78` and `C:\Users\parva\OneDrive\Desktop\AI Assisted\1`, along with system information such as CPU, RAM, and disk usage.

2. Email Priority Classification

Scenario:

A company wants to automatically prioritize incoming emails into High, Medium, or Low priority levels based on specific criteria.

Priority, Medium Priority, or Low Priority.

Tasks:

1. Create 6 sample email messages with priority labels.
2. Perform intent classification using Zero-shot prompting.
3. Perform classification using One-shot prompting.
4. Perform classification using Few-shot prompting.
5. Evaluate which technique produces the most reliable results and why.

```

File Edit Selection View Go Run Terminal Help < > AI Assisted
EXPLORER AI ASSISTED ass 4.4.py ass 3.4.pdf check_leap_year.py ab-4.5 word.docx ab-4.5 word.pdf lab-4.5 py lab-4.5.pdf lab assignment 3.3.pdf lab assignment 1.4.pdf lab assignment 2.3.pdf leap_year.py
CHAT SENTIMENT CLASSIFICATION FOR CUSTOMERS
Write a very short Python program that stores 6 sample email messages with their priority labels (High, Medium, Low) using lists or tuples.
Keep the code simple and minimal.
+ ass 4.4.py
Used in Reference
# Simple list of email tuples:
emails = [
    ("Urgent: System Glitch - Timeline Action Required", "The main database server is down. All operations are halted.", "High"),
    ("Q1 Budget Review Meeting - Next Friday at 2 PM", "Please review the attached budget documents.", "Medium"),
    ("Office Lunch - Catering Menu For Next Week", "Please vote on your preferred lunch option.", "Low"),
    ("Critical: Client Presentation Delayed - Decision Needed Today", "Our major client has requested to reschedule the presentation.", "High"),
    ("Monthly Team Updates - Please Submit by End of Week", "Submit your monthly progress report by Friday.", "Medium"),
    ("Office Supplies Restocking - New Printer Paper Available", "New printer paper has arrived in the supply closet.", "Low")
]
# Print emails with priority
print("OFFICE EMAILS")
print("-" * 80)
for idx, (subject, body, priority) in enumerate(emails, 1):
    print(f"Email {idx} ({priority})")
    print(f"Subject: {subject}")
    print(f"Body: {body}")
    print()
# Summary
print("-" * 80)
print(f"High: {sum(1 for e in emails if e[2] == 'High')} | Medium: {sum(1 for e in emails if e[2] == 'Medium')} | Low: {sum(1 for e in emails if e[2] == 'Low')}")
+ ass 4.4.py
problems output debug console terminal ports
Body: The main database server is down. All operations are halted.
Email #2 [Medium Priority]
Subject: Q1 Budget Review Meeting - Next Friday at 2 PM
Body: Please review the attached budget documents.

Email #3 [Low Priority]
Subject: Office Lunch - Catering Menu For Next Week
Body: Please vote on your preferred lunch option.

Email #4 [High Priority]
Subject: Critical: Client Presentation Delayed - Decision Needed Today
Body: Our major client has requested to reschedule the presentation.

Email #5 [Medium Priority]
Subject: Monthly Team Updates - Please Submit by End of Week
Body: Submit your monthly progress report by Friday.

Email #6 [Low Priority]
Subject: Office Supplies Restocking - New Printer Paper Available
Body: New printer paper has arrived in the supply closet.

-----
High: 2 | Medium: 2 | Low: 2
TS C:\Users\qasim\OneDrive\Desktop\AI Assistant []
This minimal version:
• Uses tuples for simple email storage
• Stores 6 emails with subject, body, and priority
• Prints formatted output with priority labels
• Shows summary counts at the end
• Keeps code simple and easy to understand
Ask Auto
28°C
1:17 PM
1/20/2026

```

The screenshot shows a Microsoft Visual Studio Code (VS Code) window with the following details:

- File Explorer:** Shows files in the 'AI Assisted' folder, including 'ass_4.4.py', 'check_leap_year.py', 'lab_4.3_word.pdf', 'lab_4.3.py', 'lab_assignment_3.3.pdf', 'lab_assignment_1.4.pdf', 'lab_assignment_2.3.pdf', and 'lab_leap_year.py'.
- Code Editor:** Displays the Python script 'ass_4.4.py'. The code defines a function 'classify_priority(subject, body)' that checks for specific keywords and returns a priority level ('High', 'Medium', or 'Low'). It also includes a test function 'test_classifier' and a custom classifier function 'classify_priority_custom(subject, body)'.
- Terminal:** Shows the output of running the script. It prints priority levels for several sample emails and then prompts for custom input. The output includes:
 - Email #3 (Low Priority): Subject: Office Lunch - Catering Menu For Next Week Body: Please vote on your preferred lunch option.
 - Email #4 (High Priority): Subject: Critical: Client Presentation Delayed - Decision Needed Today Body: Our major client has requested to reschedule the presentation.
 - Email #5 (Medium Priority): Subject: Monthly Team Updates - Please Submit by End of Week Body: Submit monthly progress report by Friday.
 - Email #6 (Low Priority): Subject: Office Supplies Restock - New Printer Paper Available Body: New printer paper has arrived in the supply closet.
- Output Panel:** Shows standard output and error messages from the terminal run.
- Right Sidebar:** Includes a 'CHAT' section with a message about sending an email, and a 'SENTIMENT CLASSIFICATION FOR CUST...' section with a snippet of code for sentiment analysis.

The screenshot shows the Microsoft Visual Studio Code interface. On the left is the Explorer sidebar with files like 'add.py', 'AI assisted', 'check_lotto_word.py', 'lab 4.3 word.docx', 'lab 4.3 word.pdf', 'lab 4.3 py', 'lab 4.3 py', 'lab assignment 3.3.pdf', 'lab assignment 1.4.pdf', 'lab assignment 2.3.pdf', and 'leap_year.py'. The main editor area contains Python code for classifying email priority based on subject and body text. The code includes examples for 'High', 'Medium', and 'Low' priority levels. The terminal at the bottom shows command-line output for running the script. A ChatGPT AI Assistant panel is open on the right, providing instructions for writing minimal if-else based code to classify email priority.

```

// AI ASSISTED
// SENTIMENT CLASSIFICATION FOR EMAILS
// Write a very simple Python code to classify email priority
// Examples:
// "Urgent issue" - High
// "Meeting tomorrow" - Medium
// "Greetings" - Low
// Use these examples and write minimal if-else based code.
// Avoid extra explanations.

def classify_priority(text):
    t = text.lower()
    if any(k in t for k in ("urgent", "server down", "issue", "critical", "immediate", "outage")):
        return "High"
    elif any(k in t for k in ("meeting", "tomorrow", "deadline", "schedule", "reminder")):
        return "Medium"
    else:
        return "Low"

# Examples
for e in ["Urgent issue", "Meeting tomorrow", "Greetings"]:
    print(f"({e}) - {classify_priority(e)}")

```

3. Student Query Routing System

Scenario:

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

Tasks:

1. Create 6 sample student queries mapped to departments.
2. Implement Zero-shot intent classification using an LLM.
3. Improve results using One-shot prompting.
4. Further refine results using Few-shot prompting.
5. Analyze how contextual examples affect classification accuracy.

The screenshot shows the Visual Studio Code interface with the AI Assistant extension active. The Explorer sidebar on the left lists various files, including 'add.py' and 'ass 4.4.py'. The main editor window displays Python code for sentiment classification. The AI Assistant panel on the right provides examples and instructions for writing a simple Python program to classify student queries. The terminal at the bottom shows command-line output related to the code execution.

```
ass 4.4.py
...
def classify_query(q):
    t = q.lower()
    if any(k in t for k in ("admission", "apply", "application", "enroll", "fee", "deadline")):
        return "Admissions"
    elif any(k in t for k in ("exam", "exams", "result", "results", "grade", "schedule")):
        return "Exams"
    elif any(k in t for k in ("placement", "placements", "internship", "job", "career", "interview")):
        return "Placement"
    elif any(k in t for k in ("course", "curriculum", "transcript", "semester", "register", "attendance", "academic")):
        return "Academics"
    else:
        return "Academics"

if __name__ == "__main__":
    q = input("Query: ").strip()
    print("Department:", classify_query(q))

```

This screenshot is nearly identical to the first one, showing the same VS Code interface with the AI Assistant extension. The code in the editor and the AI Assistant panel remain the same, providing examples for sentiment classification based on student queries.

```
ass 4.4.py
...
def classify_query(q):
    t = q.lower()
    if any(k in t for k in ("admission", "apply", "application", "enroll", "fee", "deadline")):
        return "Admissions"
    elif any(k in t for k in ("exam", "exams", "result", "results", "grade", "schedule")):
        return "Exams"
    elif any(k in t for k in ("placement", "placements", "internship", "job", "career", "interview")):
        return "Placement"
    elif any(k in t for k in ("course", "curriculum", "transcript", "semester", "register", "attendance", "academic")):
        return "Academics"
    else:
        return "Academics"

if __name__ == "__main__":
    q = input("Query: ").strip()
    print("Department:", classify_query(q))

```

The screenshot shows a Microsoft Visual Studio Code (VS Code) window with the following details:

- File Explorer:** Shows files like `AI Assisted`, `AI ass4 Ass 4.pdf`, `check_leap_year.py`, `lab 4.3 word.docx`, `lab 4.3 py`, `lab 4.3 assignment 3.3.pdf`, `lab 4.3 assignment 2.3.pdf`, and `lisp_star.py`.
- Code Editor:** Displays Python code for sentiment classification. The code includes functions for `zero_shot`, `one_shot`, `few_shot`, and `score`. It uses NLTK's `wordnet` and `stopwords` modules.
- Terminal:** Shows command-line history for running the script with different arguments (`ass4 Ass 4.pdf`, `AI ass4 Ass 4.pdf`, etc.).
- Output:** Shows standard output from the terminal runs.
- Search:** A search bar at the top with the placeholder "Search".
- Activity Bar:** Includes icons for Output, Debug Console, Terminal, and Ports.
- Chat Panel:** On the right, titled "SENTIMENT CLASSIFICATION FOR CUSTOMER", it displays a conversation with AI Assistant. The user asks for a Python code snippet for sentiment analysis using NLTK, and the AI provides a short one-liner:

```
Write a short Python code that tests routing using one or two simple lines.
```
- Status Bar:** Shows file paths, line numbers (L 85 Col 60), spaces used (Spaces 4), and other system information.

4. Chatbot Question Type Detection

Scenario:

A chatbot must identify whether a user query is Informational,

Transactional, Complaint, or Feedback.

Tasks:

1. Prepare 6 chatbot queries mapped to question types.
 2. Design prompts for Zero-shot, One-shot, and Few-shot learning
 3. Test all prompts on the same unseen queries.
 4. Compare response correctness and ambiguity handling.
 5. Document observations.

```

File Edit Selection View Go Run Terminal Help < > Q AI Assisted
EXPLORER AI ASSISTED
  add.py
  ass 4.4.pdf
  AI ass Ass 1.pdf
  ass 4.4.pdf
  ass 4.4.pdf
  check_leap_year.py
  lab 4.3 word.docx
  lab 4.3 word.pdf
  lab 4.3 py
  lab 4.3 py
  lab assignment 3.3.pdf
  lab assignment 1.4.pdf
  lab assignment 2.3.pdf
  leap_year.py
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
>>> nl

```

```

ass 4.4.py
# samples queries and types
queries = [
    "What's the status of my order",
    "I want to change my shipping address",
    "The product arrived broken and unusable",
    "How do I use the new dashboard feature?",
    "Thanks for the quick support, great job"
]

# Zero-shot classification
def zero_shot():
    t = q.lower()
    if any(k in t for k in ("broken", "damaged", "not working", "complain", "issue", "problem")):
        return "Complaint"
    if any(k in t for k in ("order", "reset", "change", "cancel", "status", "pay", "buy", "billing")):
        return "Transactional"
    if any(k in t for k in ("love", "great", "thanks", "thank", "feedback", "suggest")):
        return "Feedback"
    return "Informational"

# One-shot classification
def one_shot(q):
    if "reset my password" in q.lower(): return "Transactional"
    return zero_shot(q)

# Few-shot classification
def few_shot(q):
    examples = [
        "reset my password": "Transactional",
        "order arrived damaged": "Complaint",
        "love the new interface": "Feedback",
        "user guide": "Informational"
    ]
    t = q.lower()
    for k, v in examples.items():
        if k in t: return v
    return zero_shot(q)

# Run all queries
for q in queries:
    print("Query:", q)
    print("Zero-shot:", zero_shot(q))
    print("One-shot:", one_shot(q))
    print("Few-shot:", few_shot(q))

```

AI Assisted Chat

SENTIMENT CLASSIFICATION FOR CUSTOMERS

If else, no examples
a) One-shot - provide one example as a comment or guide.
c) Few-shot (include 3-4 examples as comments or guide)
3. Test all three approaches on the same set of test input queries.
4. Print the detected question type for each approach.
5. At the end, print 2-3 short lines comparing:
a) correctness
b) ambiguity handling
6. Keep the code beginner-friendly and readable.
7. DO NOT use ML or AI libraries.
& Avoid extra explanations focus on short, reusable code only.

Used 1 reference
No actions performed
...existing code...
samples = [
 ("How do I reset my password", "Feedback"),
 ("What are your refund policies", "Informational"),
 ("My order arrived damaged", "Complaint"),
 ("I love the new interface", "Feedback"),
 ("How to change my billing info", "Informational"),
 ("Where can I find the user manual", "Informational")
]

```

def zero_shot(q):
    t = q.lower()
    if any(k in t for k in ("broken", "damaged", "not working", "complain", "issue", "problem")):
        return "Complaint"
    if any(k in t for k in ("order", "reset", "change", "cancel", "status", "pay", "buy", "billing")):
        return "Transactional"
    if any(k in t for k in ("love", "great", "thanks", "thank", "feedback", "suggest")):
        return "Feedback"
    return "Informational"

```

5. Emotion Detection in Text

Scenario:

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

Anxious, Neutral.

Tasks:

1. Create labeled emotion samples.
2. Use Zero-shot prompting to identify emotions.
3. Use One-shot prompting with an example.
4. Use Few-shot prompting with multiple emotions.
5. Discuss ambiguity handling across techniques.

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

- File Explorer:** Shows files in the "AI CODING" folder, including "add.py", "Al lab43.py", "Assignment1(CP).pdf", "Assignment 3.4", "assignment 3.py", "Assignment2.pdf", "assignment3.4.docx", "factorial.py", "jobs.py", "jobscp.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 imports pandas and creates a DataFrame from a dictionary of text and emotion pairs.
- Terminal:** Shows the command line output of running the script with Python 3.13, which results in a ModuleNotFoundError for 'pandas'.

```
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>
```

The screenshot shows a second instance of VS Code with the same setup as the first one, displaying the "lab assignment 44.py" script and its execution output in the terminal.

```
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>
```

File Edit Selection View Go Run Terminal Help

AI Coding

EXPLORER AI CODING

- Assignment1(CP).pdf
- Assignment 2-4.pdf
- assignment 3.py
- Assignment 3.pdf
- Assignment2.pdf
- factorial.py
- jobs.py
- jobsCP.py
- lab assignment ...
- lab assignment3.3.pdf
- lab1 HCP.pdf
- matrixHCP.py
- Untitled20.py
- week2 HCP.pdf

Generate code

Add Context...

```
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}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

ModuleNotFoundError: No module named 'pandas'

PS D:\AI Coding & C:/Users/ANDALI/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

ModuleNotFoundError: No module named 'pandas'

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

Emotion: Anxious

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

Emotion: Frustrated

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

File Edit Selection View Go Run Terminal Help

AI Coding

EXPLORER AI CODING

- Assignment1(CP).pdf
- Assignment 2-4.pdf
- assignment 3.py
- Assignment 3.pdf
- Assignment2.pdf
- factorial.py
- jobs.py
- jobsCP.py
- lab assignment ...
- lab assignment3.3.pdf
- lab1 HCP.pdf
- matrixHCP.py
- Untitled20.py
- week2 HCP.pdf

Generate code

Add Context...

```
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}")
```

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/ANDALI/AppData/Local/Programs/Python/Python313/python.exe "d:/AI Coding/lab assignment 44.py"

Emotion: Anxious

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

Emotion: Frustrated

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

The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which lists various files and folders under the 'AI CODING' category, including 'add.py', 'Assignment1(CP).pdf', 'Assignment 2-4.pdf', 'assignment 3.4', 'Assignment2.pdf', 'assignment3.4.docx', 'factorial.py', 'jobs.py', 'jobs.py', 'lab assignment ...', 'lab assignment3.3.pdf', 'lab1 HCP.pdf', 'matnHCP.py', 'Untitled20.py', and 'week2 HCP.pdf'. The main workspace displays a Python script named 'lab assignment 44.py' with the following 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}")
```

Below the code editor is a terminal window showing the following session:

```
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>
```