

2303A51491

Batch-25

Assignment-4.4

## 1. Sentiment Classification for Customer Reviews

Scenario:

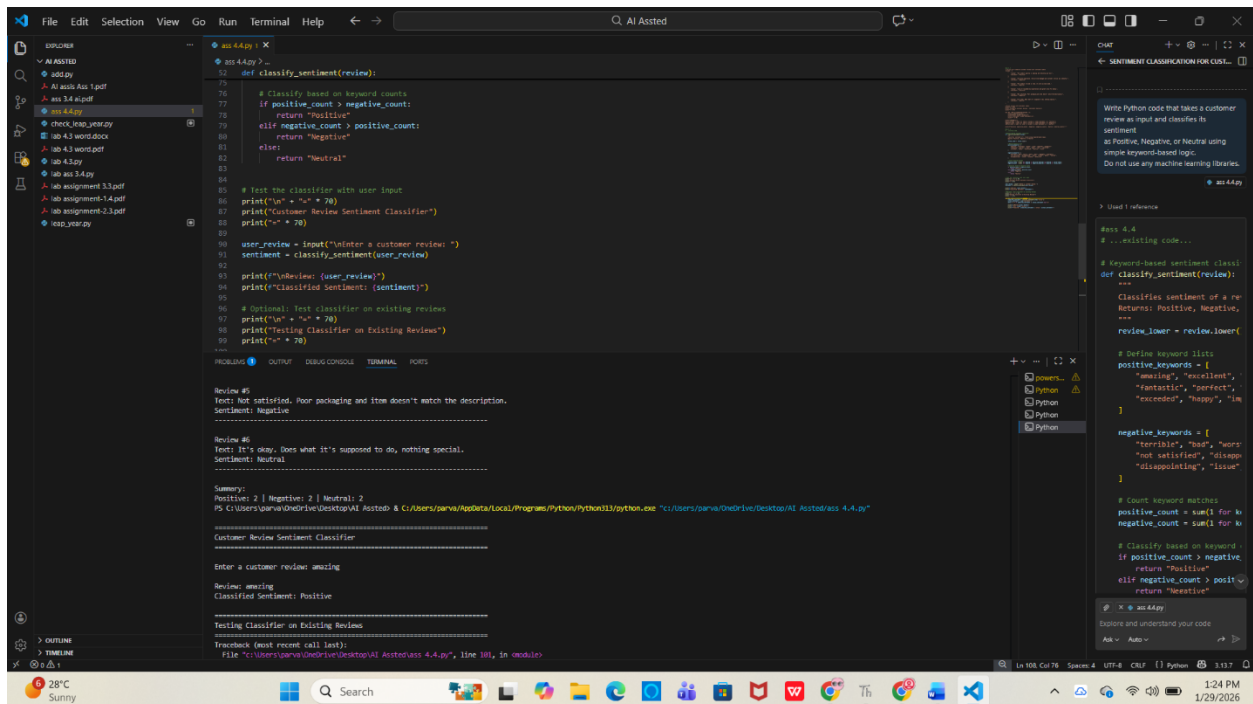
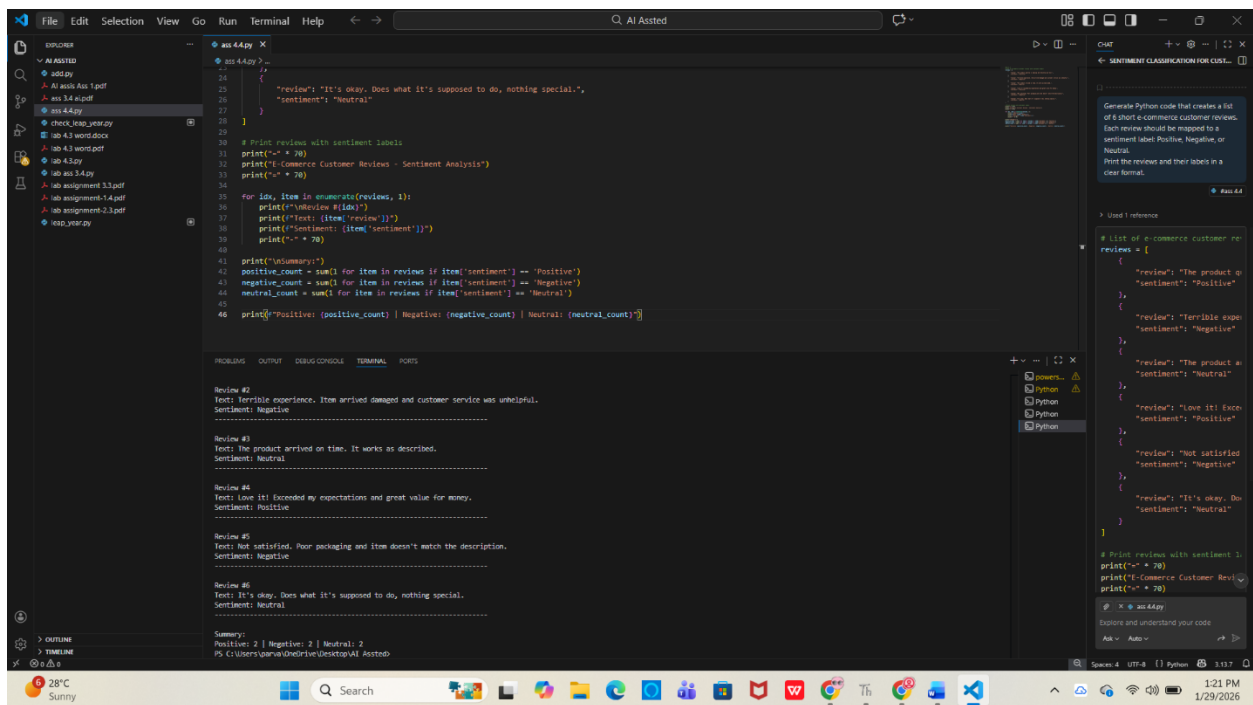
An e-commerce platform wants to analyze customer reviews and classify

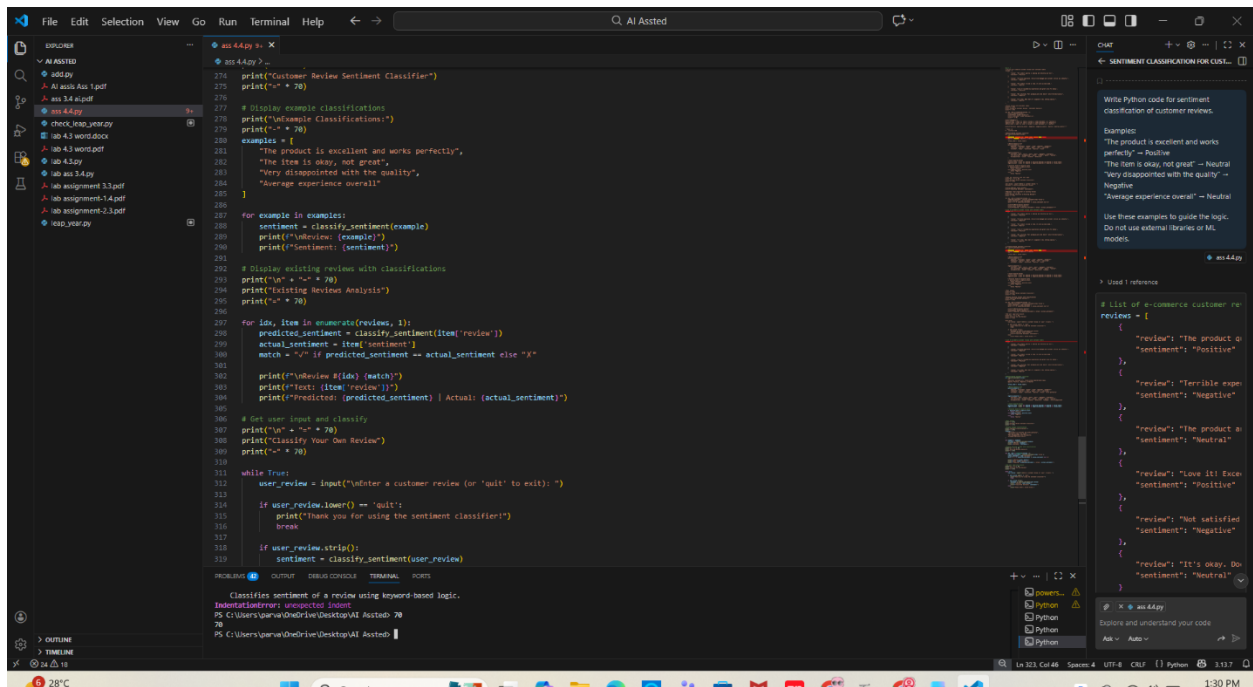
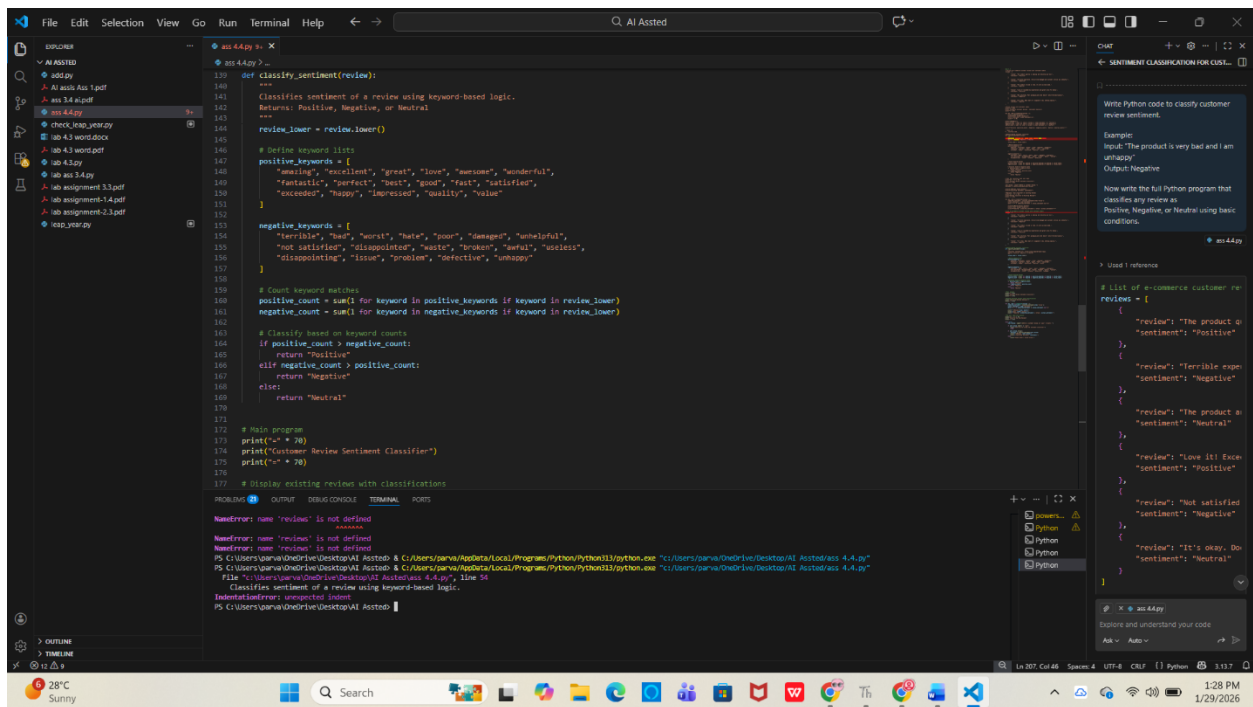
Week2

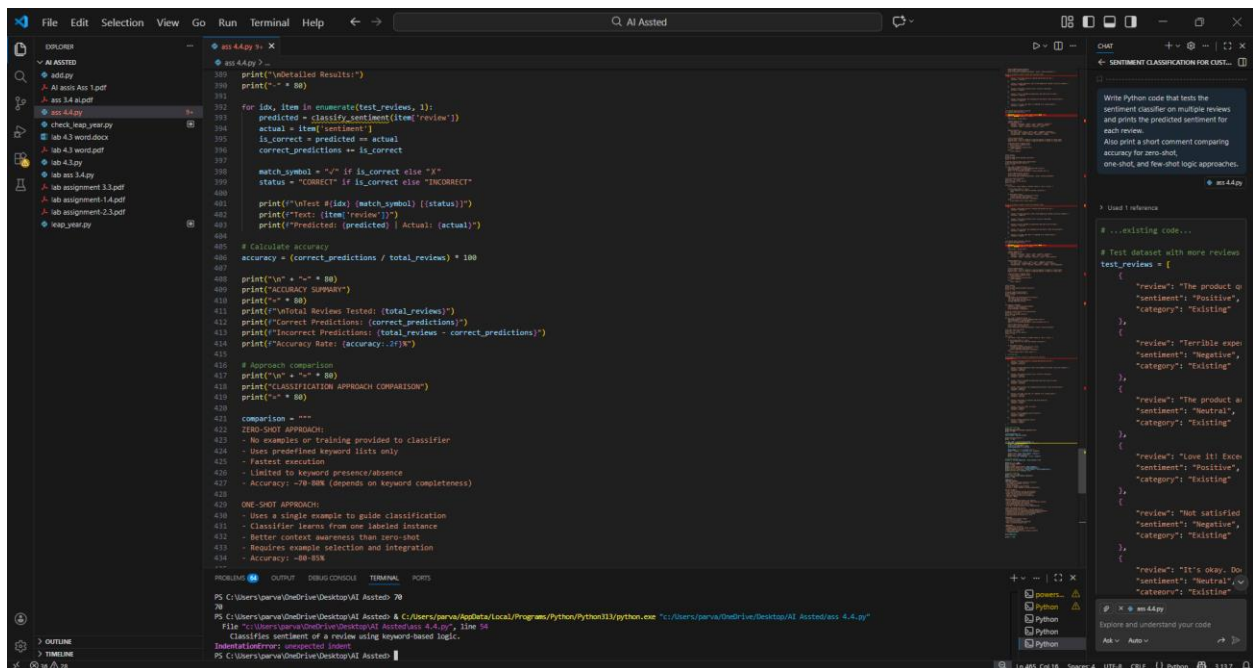
them into Positive, Negative, or Neutral sentiments using prompt engineering.

Tasks:

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







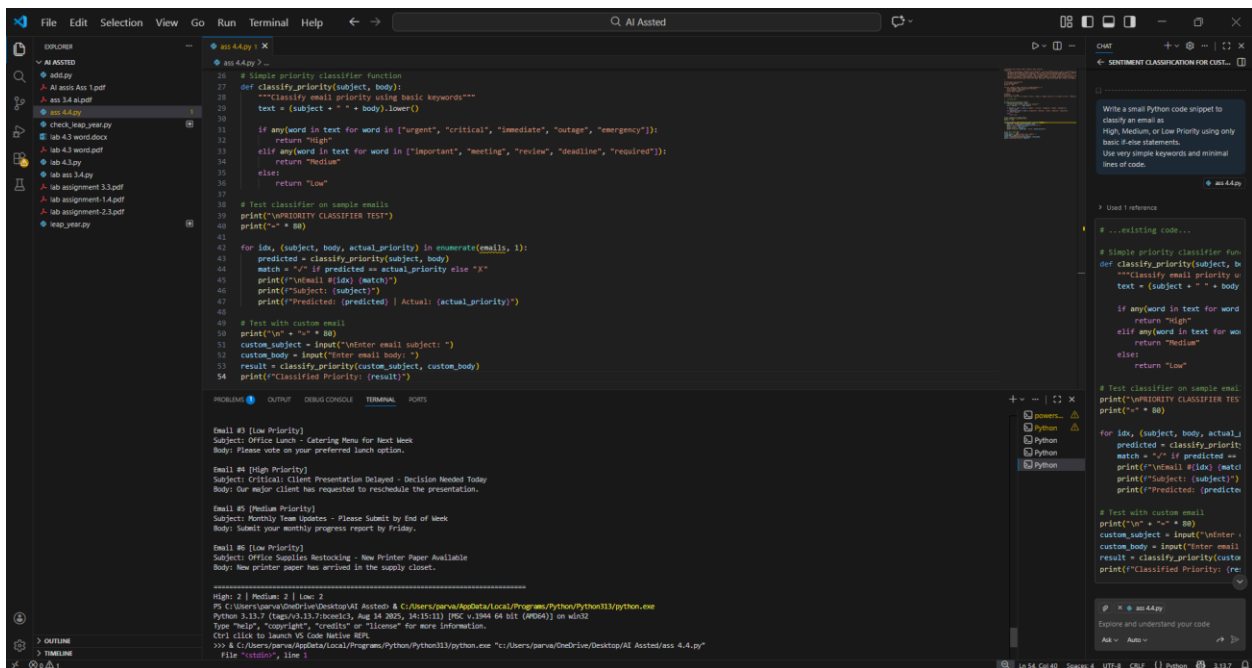
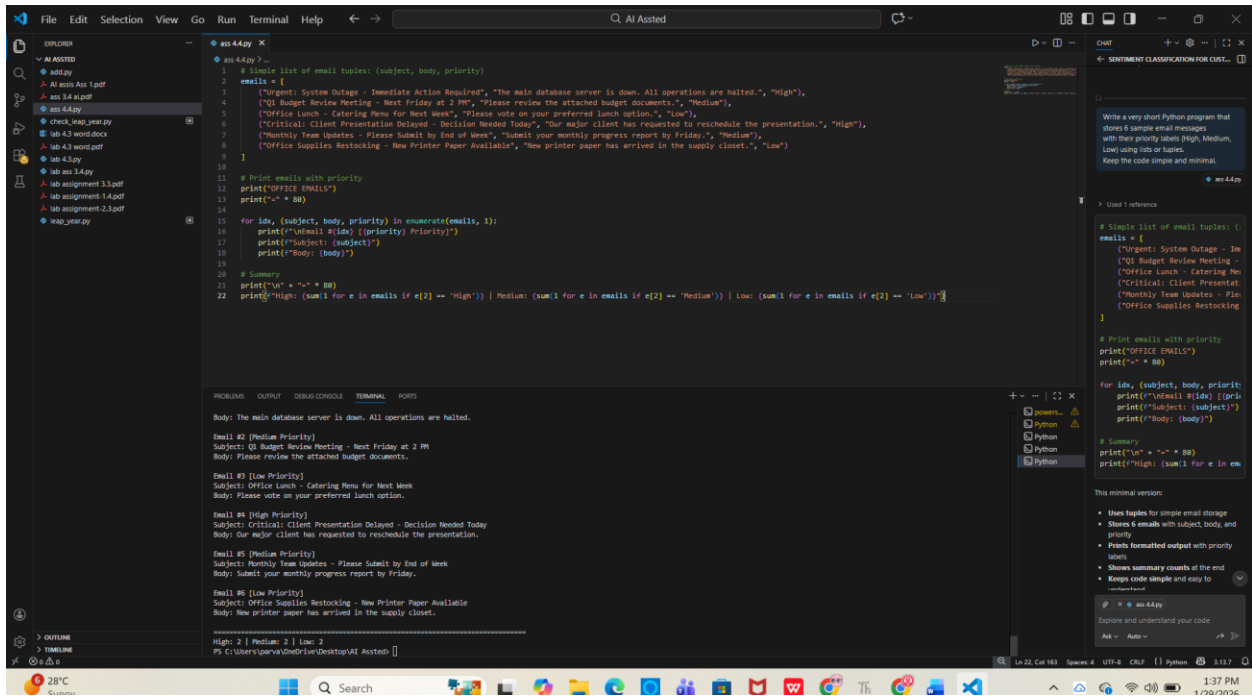
## 2. Email Priority Classification

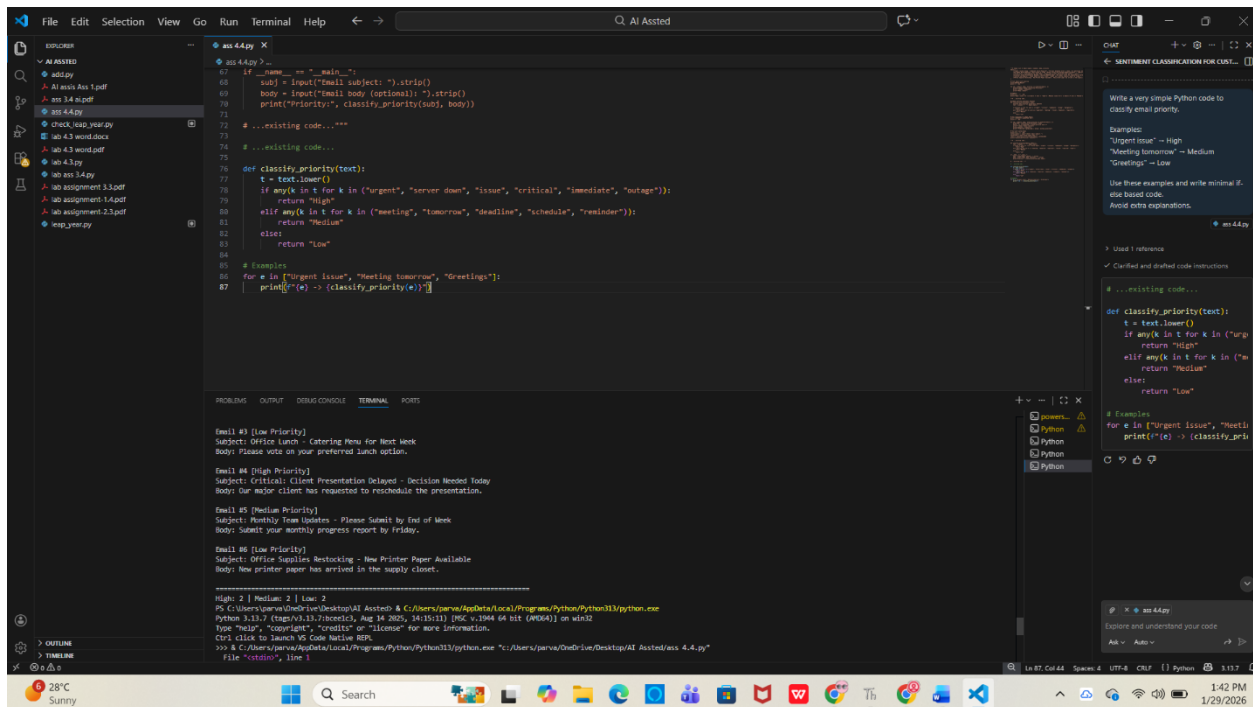
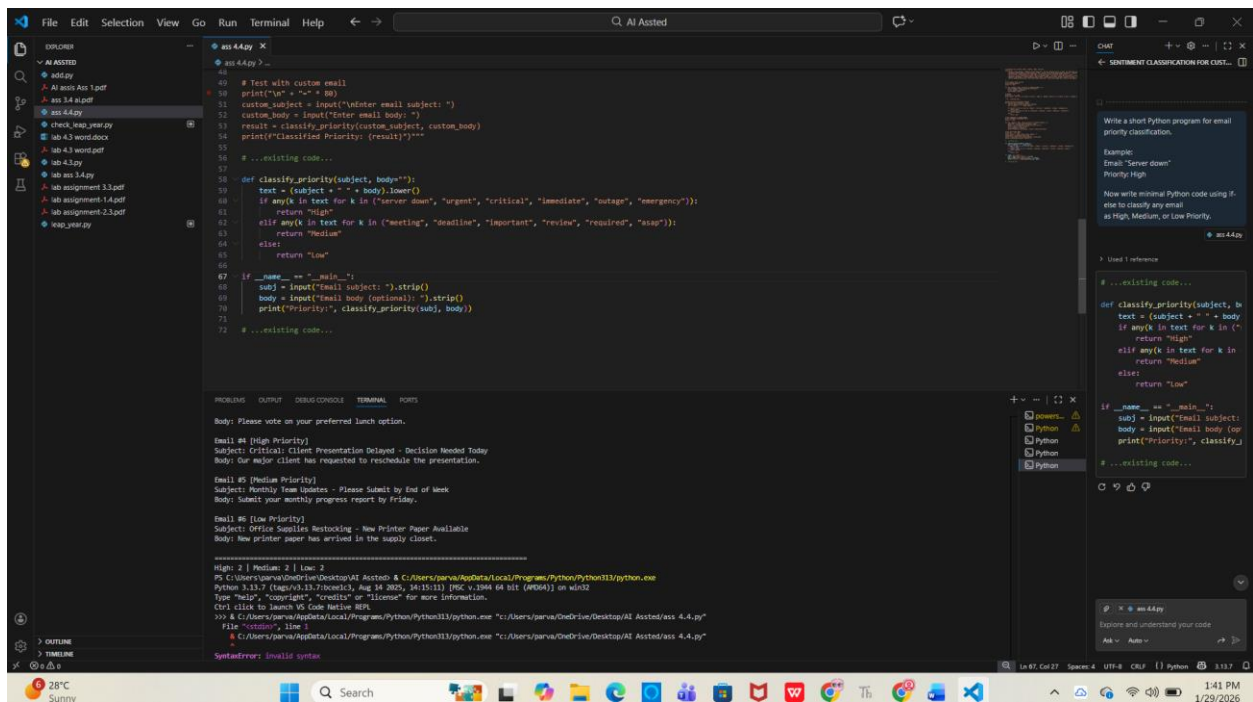
Scenario:

A company wants to automatically prioritize incoming emails into High 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.



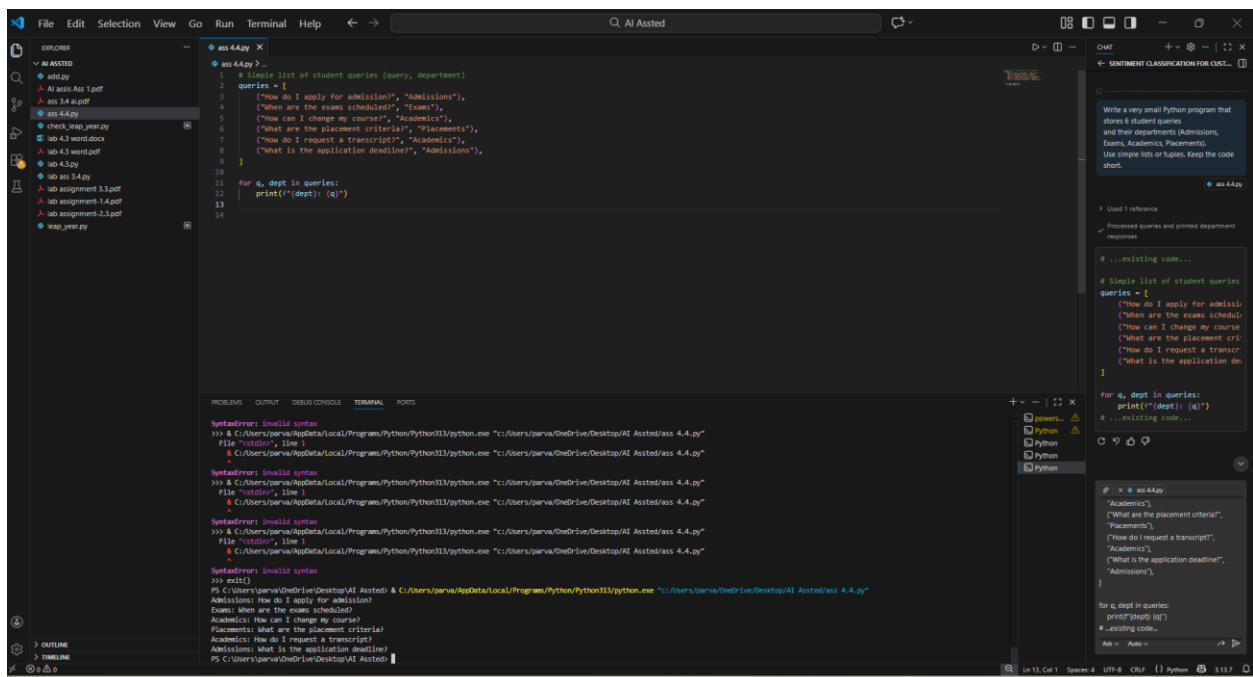


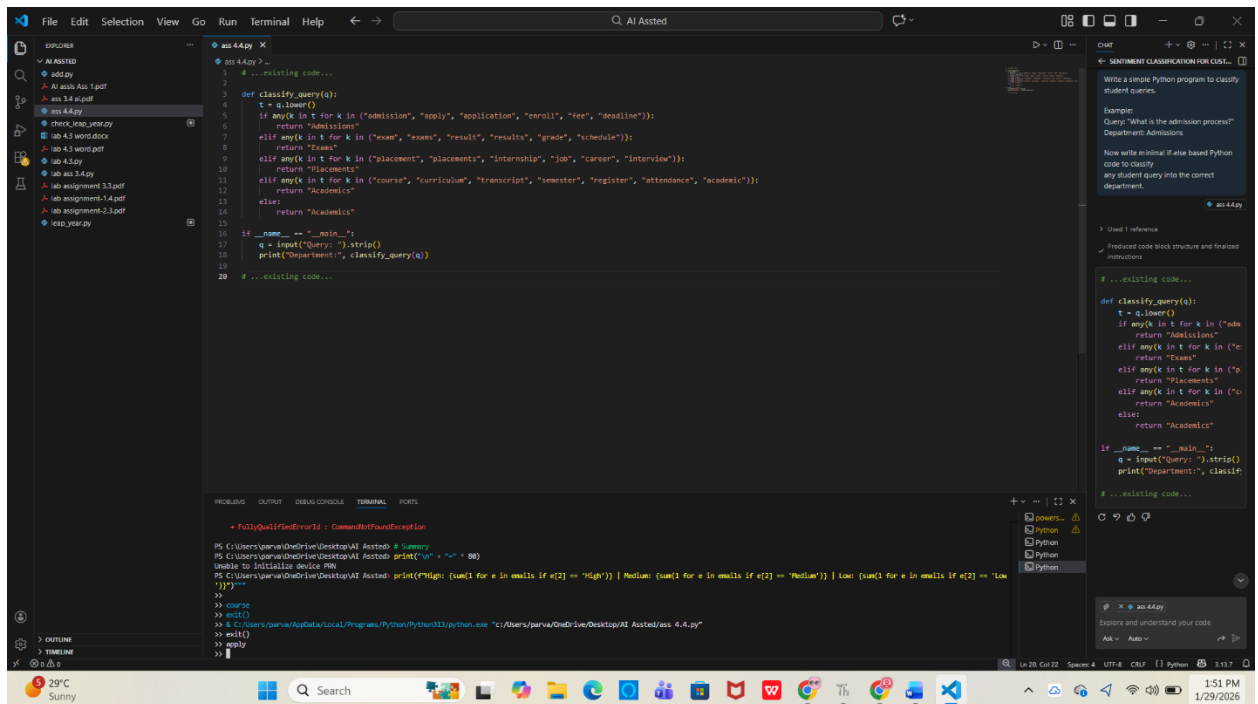
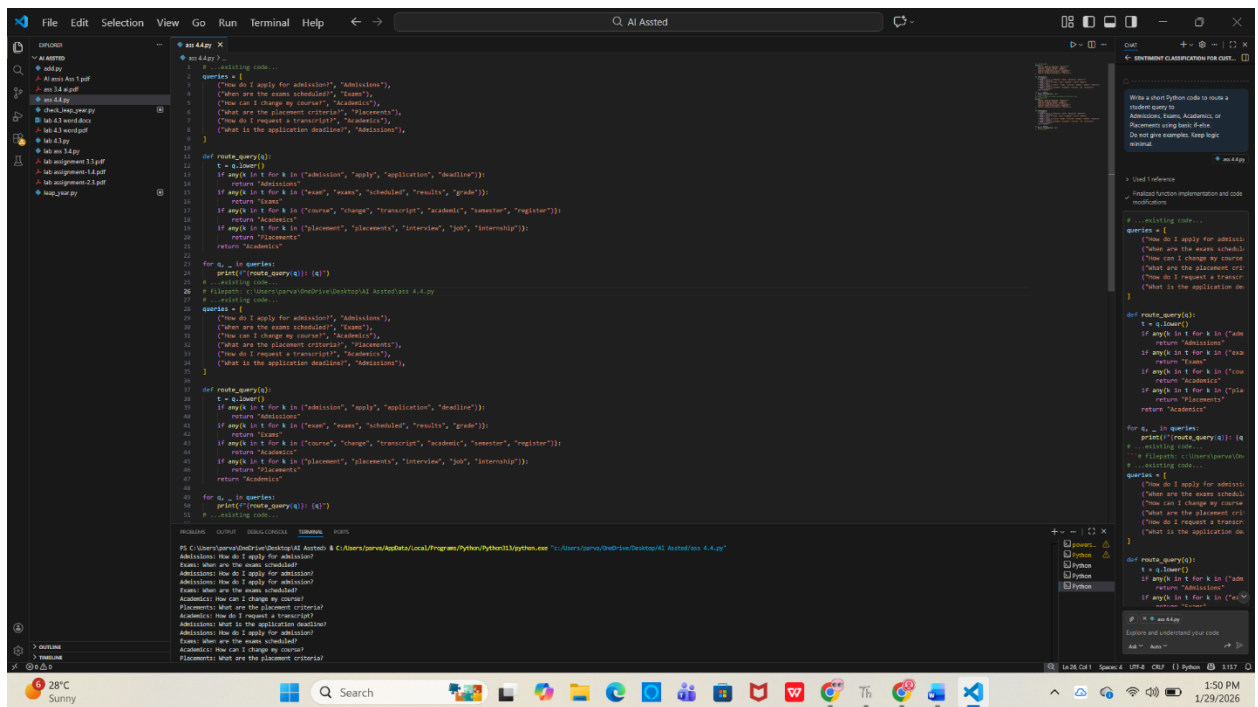
### 3. Student Query Routing System

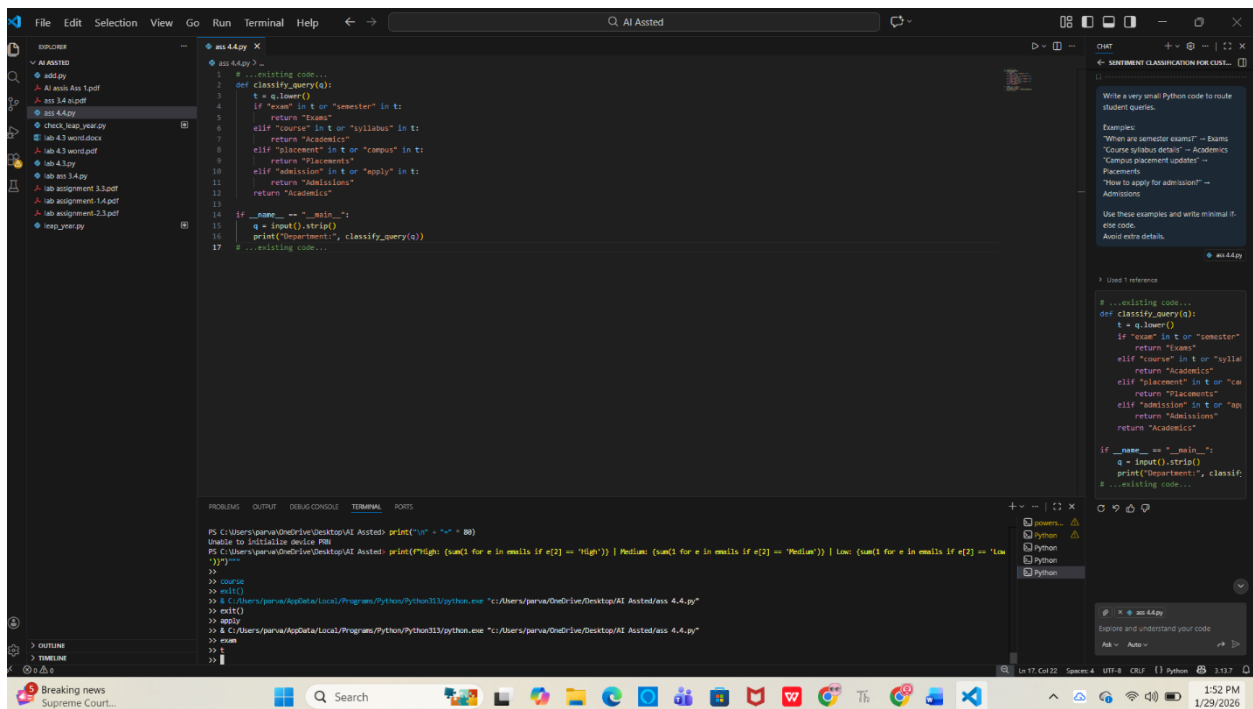
Scenario:

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

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.

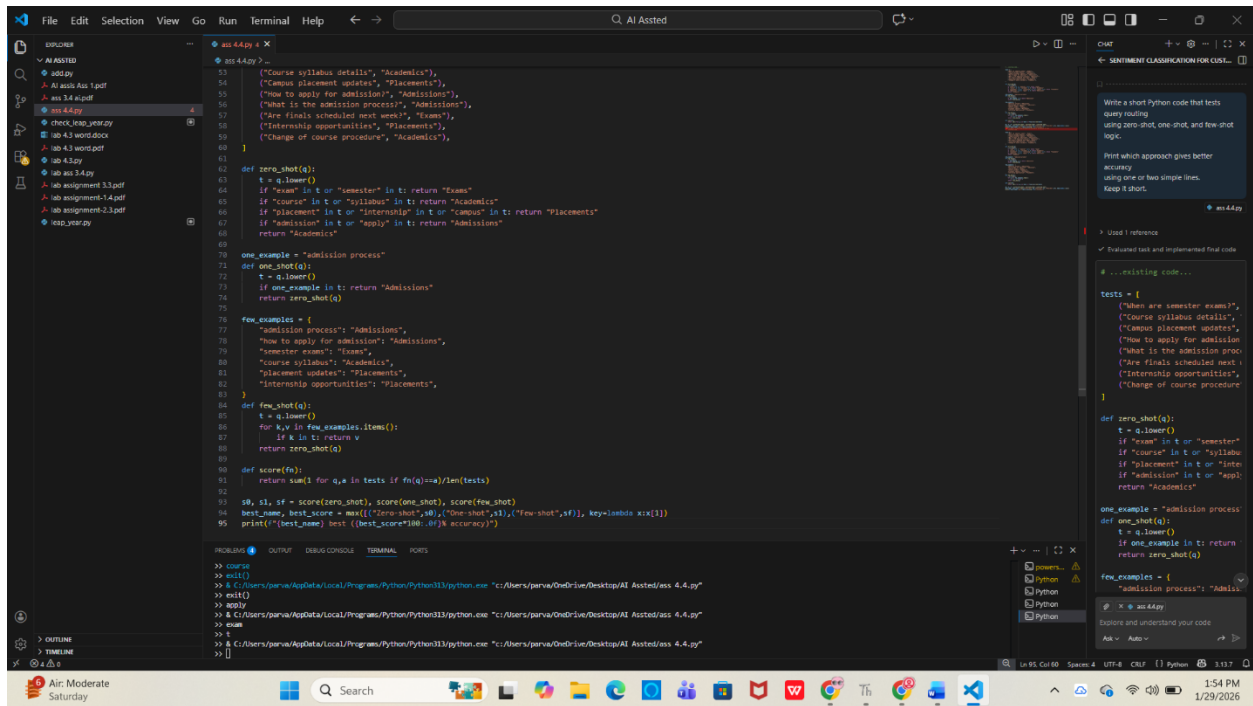






```
1 # ...existing code...
2 def classify_query(q):
3     t = q.lower()
4     if "exam" in t or "semester" in t:
5         return "Exams"
6     elif "course" in t or "syllabus" in t:
7         return "Academics"
8     elif "placement" in t or "campus" in t:
9         return "Placements"
10    elif "admission" in t or "apply" in t:
11        return "Admissions"
12    return "Academics"
13
14 if __name__ == "__main__":
15     q = input().strip()
16     print("Department:", classify_query(q))
17 # ...existing code...
```

```
PS C:\Users\parva\OneDrive\Desktop\AI Assted> python -i ass 4.4.py
Unable to initialize device FIM
PS C:\Users\parva\OneDrive\Desktop\AI Assted> print(FHigh( sum(1 for e in emails if e[2] == "High") | Median( sum(1 for e in emails if e[2] == "Medium") | Low( sum(1 for e in emails if e[2] == "Low") ) ) ) )
>>> course
>>> exit()
>>> C:\Users\parva\AppData\Local\Programs\Python\Python311\python.exe "C:\Users\parva\OneDrive\Desktop\AI Assted\ass 4.4.py"
>>> exit()
>>> apply
>>> C:\Users\parva\AppData\Local\Programs\Python\Python311\python.exe "C:\Users\parva\OneDrive\Desktop\AI Assted\ass 4.4.py"
>>> exam
>>>
```



```
53 ("Course syllabus details", "Academics"),
54 ("Campus placement updates", "Placements"),
55 ("How to apply for admission", "Admissions"),
56 ("What is the admission process", "Admissions"),
57 ("Are finals scheduled next week", "Exams"),
58 ("Internship opportunities", "Placements"),
59 ("Change of course procedure", "Academics"),
60 ]
61
62 def zero_shot(q):
63     t = q.lower()
64     if "exam" in t or "semester" in t: return "Exams"
65     if "course" in t or "syllabus" in t: return "Academics"
66     if "placement" in t or "internship" in t or "campus" in t: return "Placements"
67     if "admission" in t or "apply" in t: return "Admissions"
68     return "Academics"
69
70 one_example = "admission process"
71 def one_shot(q):
72     t = q.lower()
73     if one_example in t: return "Admissions"
74     return zero_shot(q)
75
76 few_examples = [
77     "admission process", "Admissions",
78     "how to apply for admission", "Admissions",
79     "semester exam", "Exams",
80     "course syllabus", "Academics",
81     "placement updates", "Placements",
82     "internship opportunities", "Placements",
83 ]
84
85 def few_shot(q):
86     t = q.lower()
87     for i,v in few_examples.items():
88         if i in t: return v
89     return zero_shot(q)
90
91 def score(fa):
92     return sum(1 for q,a in tests if fa(q)==a)/len(tests)
93
94 s0, s1, sF = score(zero_shot), score(one_shot), score(few_shot)
95 best_name, best_score = max([(zero_shot,s0), (one_shot,s1), (few_shot,sF)], key=lambda x:x[1])
96 print("({best_name}) best ({best_score}={100 - 87%N accuracy})")
```

```
>>> course
>>> exit()
>>> C:\Users\parva\AppData\Local\Programs\Python\Python311\python.exe "C:\Users\parva\OneDrive\Desktop\AI Assted\ass 4.4.py"
>>> exit()
>>> apply
>>> C:\Users\parva\AppData\Local\Programs\Python\Python311\python.exe "C:\Users\parva\OneDrive\Desktop\AI Assted\ass 4.4.py"
>>> exam
>>> C:\Users\parva\AppData\Local\Programs\Python\Python311\python.exe "C:\Users\parva\OneDrive\Desktop\AI Assted\ass 4.4.py"
>>>
```

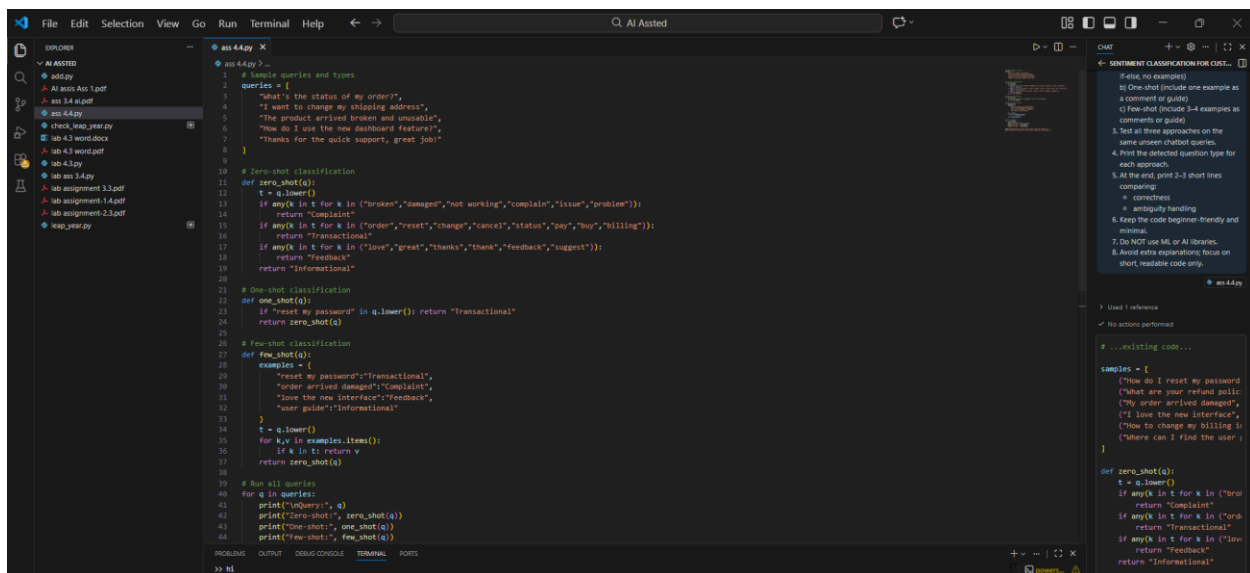
## 4. Chatbot Question Type Detection

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.



The screenshot shows a VS Code editor with a Python file named `ass 4.4.py`. The script defines a list of queries and three classification functions: `zero_shot_classification`, `one_shot_classification`, and `few_shot_classification`. The `few_shot_classification` function uses a list of examples to guide the model. The script also includes a `Run` function that iterates over the queries and prints the results of each classification. On the right side of the editor, there is a chat window titled "SENTIMENT CLASSIFICATION FOR CUSTOMER SERVICE" showing a list of queries and their corresponding sentiment classifications. The chat window also displays a list of instructions for the model, such as "1. Test all three approaches on the same unseen chatbot queries." and "2. Do NOT use ML or AI libraries."

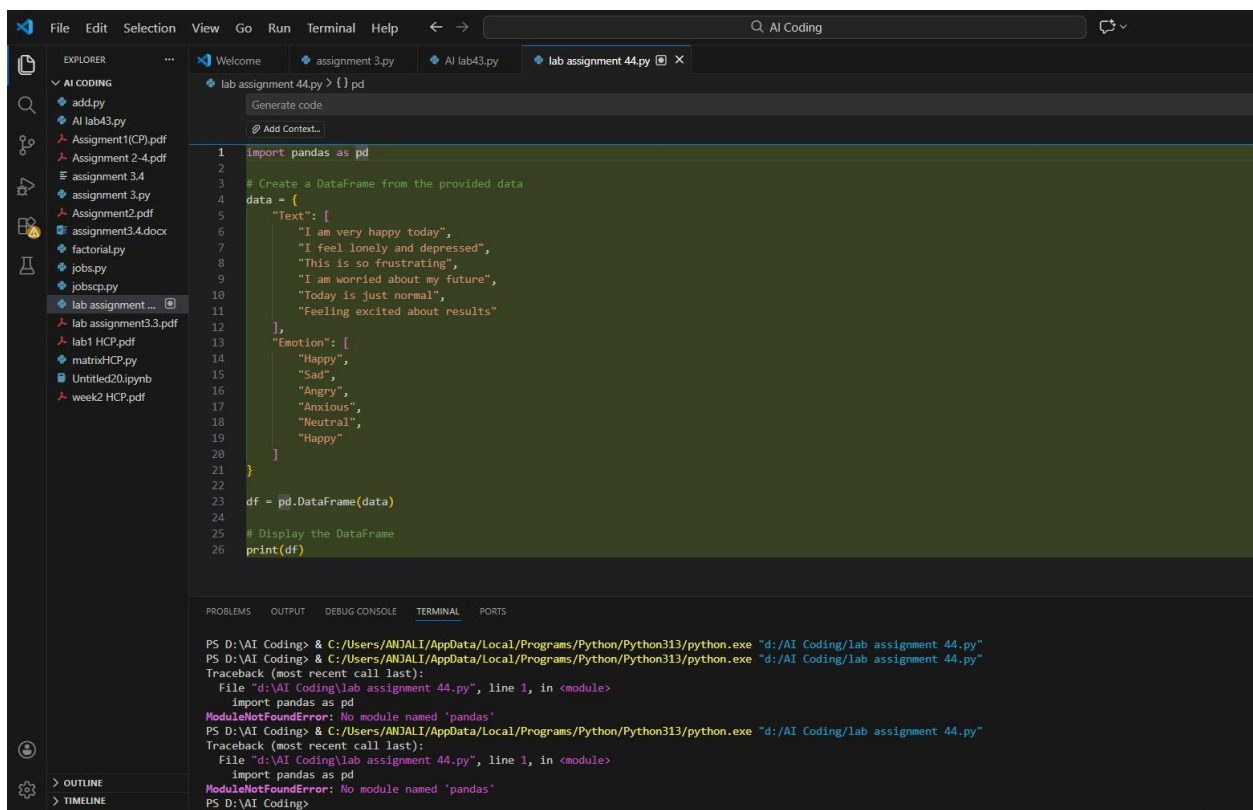
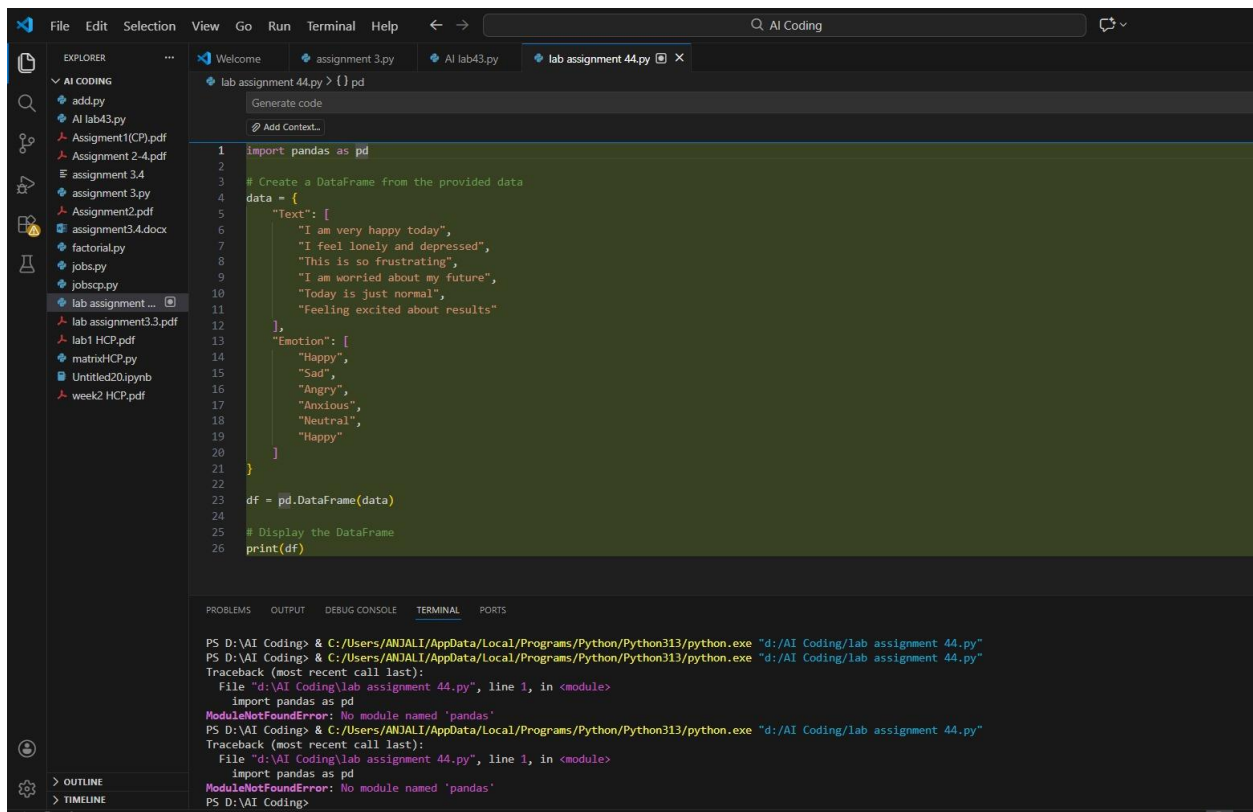
## 5. Emotion Detection in Text

### 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 the VS Code interface with a file explorer on the left containing various assignment files. The main editor displays a Python script named 'lab assignment 44.py'. The script defines a function 'identify\_emotion' that checks for the word 'frustrating' in a text string and returns 'frustrated' or 'neutral'. It includes an example usage where the function is called on the string 'This is so frustrating', resulting in the output 'Emotion: (frustrated)'.

```
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}")
```

The terminal at the bottom shows the command prompt output, including the command to run the script and the resulting output: 'Emotion: (frustrated)'.

The screenshot shows the VS Code interface with a file explorer on the left containing various assignment files. The main editor displays a Python script named 'lab assignment 44.py'. The script defines a function 'classify\_emotion' that uses a dictionary to map keywords to emotions. It includes an example usage where the function is called on the string 'This is so frustrating', resulting in the output 'Text: "This is so frustrating" Emotion: frustrated'.

```
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}\" Emotion: {emotion}")
```

The terminal at the bottom shows the command prompt output, including the command to run the script and the resulting output: 'Text: "This is so frustrating" Emotion: frustrated'.

