2303A51504

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:

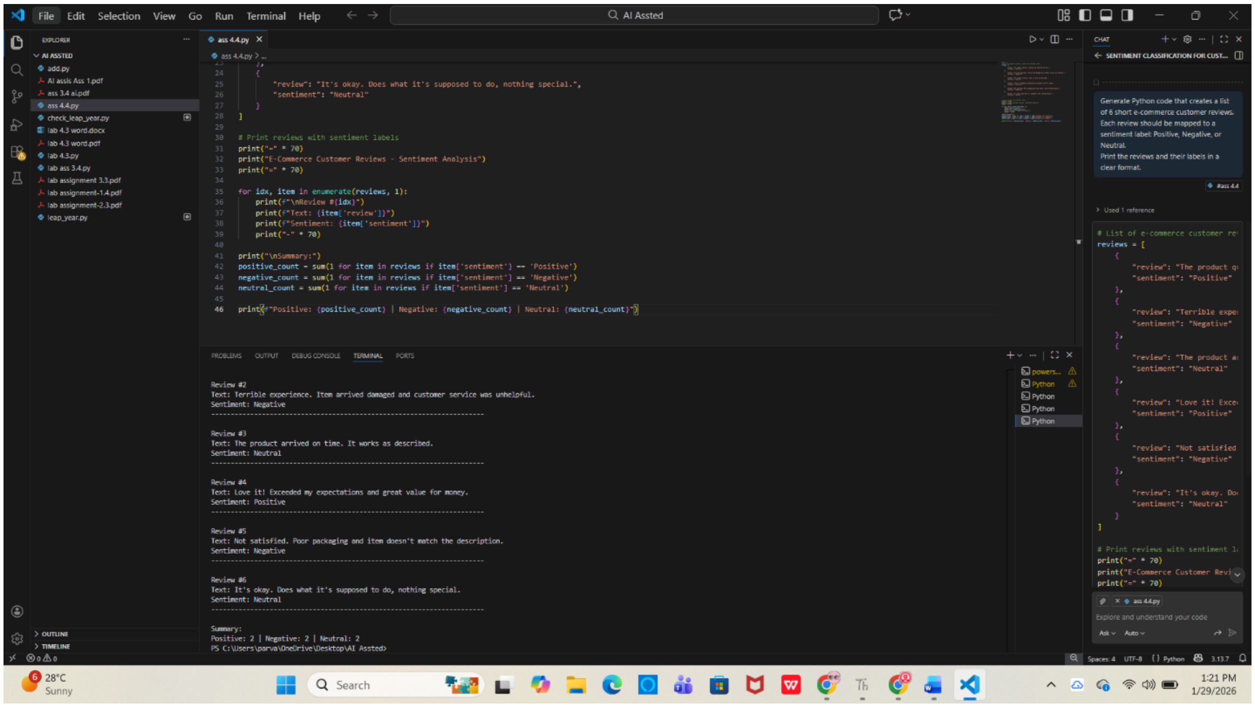
a) Prepare 6 short customer reviews mapped to sentiment labels.

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.

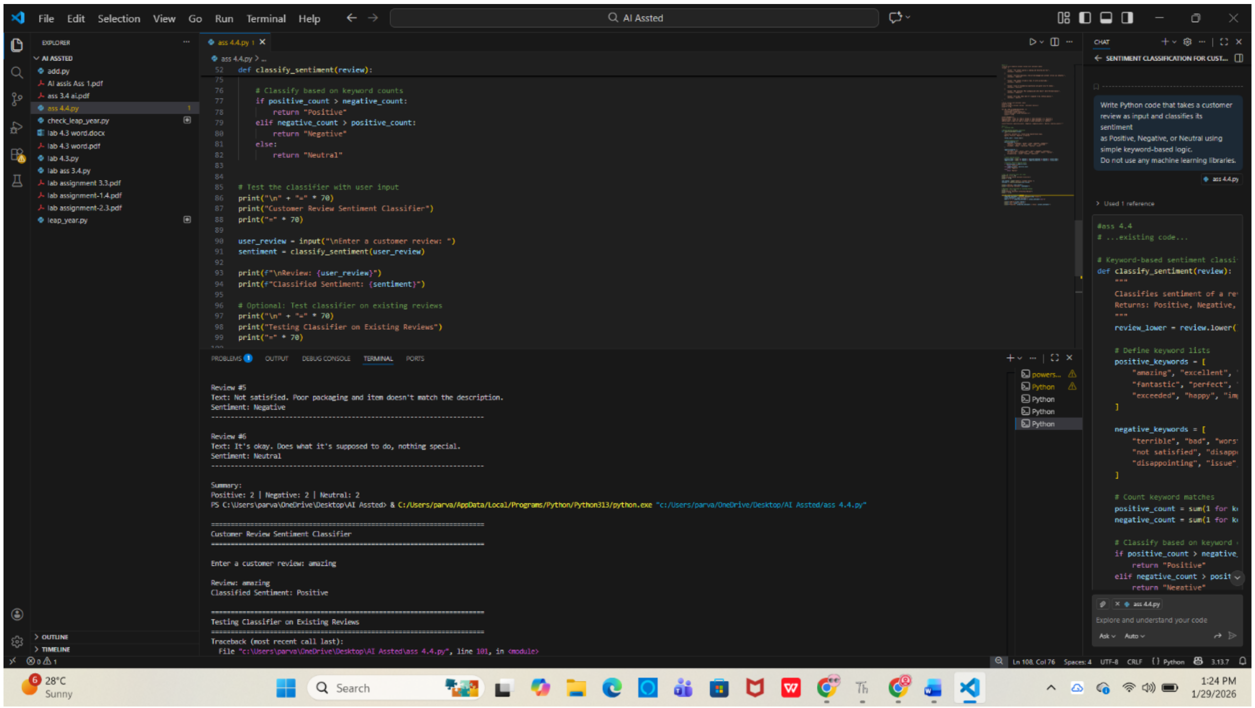


```python
    },
    {
        "review": "It's okay. Does what it's supposed to do, nothing special.",
        "sentiment": "Neutral"
    }
]

# Print reviews with sentiment labels
print("=" * 70)
print("E-Commerce Customer Reviews - Sentiment Analysis")
print("=" * 70)

for idx, item in enumerate(reviews, 1):
    print(f"\nReview #{idx}")
    print(f"Text: {item['review']}")
    print(f"Sentiment: {item['sentiment']}")
    print("-" * 70)

print("\nSummary:")
positive_count = sum(1 for item in reviews if item['sentiment'] == 'Positive')
negative_count = sum(1 for item in reviews if item['sentiment'] == 'Negative')
neutral_count = sum(1 for item in reviews if item['sentiment'] == 'Neutral')

print(f"Positive: {positive_count} | Negative: {negative_count} | Neutral: {neutral_count}")
```

```
Review #2
Text: Terrible experience. Item arrived damaged and customer service was unhelpful.
Sentiment: Negative
----------------------------------------------------------------------

Review #3
Text: The product arrived on time. It works as described.
Sentiment: Neutral
----------------------------------------------------------------------

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
PS C:\Users\parva\OneDrive\Desktop\AI Assted>
```

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 reviews and their labels in a clear format.

```python
# List of e-commerce customer re
reviews = [
    {
        "review": "The product q
        "sentiment": "Positive"
    },
    {
        "review": "Terrible expe
        "sentiment": "Negative"
    },
    {
        "review": "The product a
        "sentiment": "Neutral"
    },
    {
        "review": "Love it! Exce
        "sentiment": "Positive"
    },
    {
        "review": "Not satisfied
        "sentiment": "Negative"
    },
    {
        "review": "It's okay. Do
        "sentiment": "Neutral"
    }
]

# Print reviews with sentiment l
print("=" * 70)
print("E-Commerce Customer Revi
print("=" * 70)
```

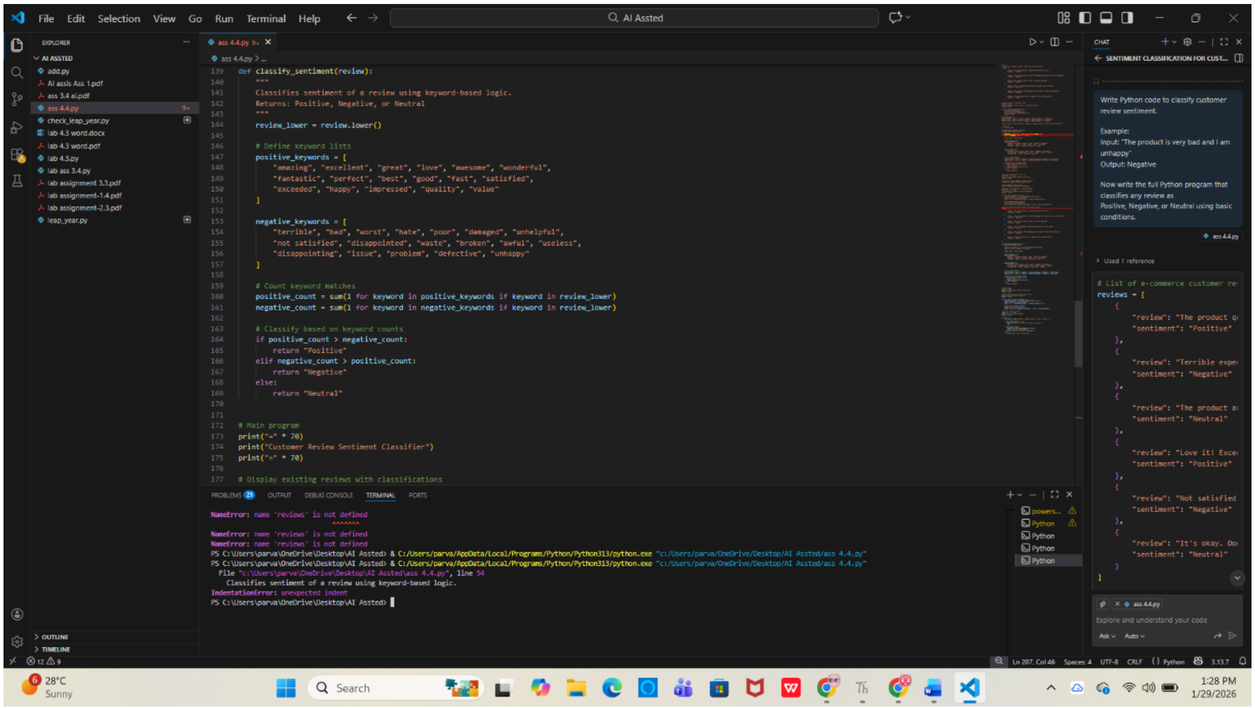


```python
def classify_sentiment(review):

        # Classify based on keyword counts
        if positive_count > negative_count:
            return "Positive"
        elif negative_count > positive_count:
            return "Negative"
        else:
            return "Neutral"


# Test the classifier with user input
print("\n" + "=" * 70)
print("Customer Review Sentiment Classifier")
print("=" * 70)

user_review = input("\nEnter a customer review: ")
sentiment = classify_sentiment(user_review)

print(f"\nReview: {user_review}")
print(f"Classified Sentiment: {sentiment}")

# Optional: Test classifier on existing reviews
print("\n" + "=" * 70)
print("Testing Classifier on Existing Reviews")
print("=" * 70)
```

```
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
PS C:\Users\parva\OneDrive\Desktop\AI Assted> & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assted/ass 4.4.py"

======================================================================
Customer Review Sentiment Classifier
======================================================================

Enter a customer review: amazing

Review: amazing
Classified Sentiment: Positive

======================================================================
Testing Classifier on Existing Reviews
======================================================================
Traceback (most recent call last):
  File "c:\Users\parva\OneDrive\Desktop\AI Assted\ass 4.4.py", line 101, in <module>
```

Write Python code that takes a customer review as input and classifies its sentiment
as Positive, Negative, or Neutral using simple keyword-based logic.
Do not use any machine learning libraries.

```python
#ass 4.4
# ...existing code...

# Keyword-based sentiment classi
def classify_sentiment(review):
    """
    Classifies sentiment of a re
    Returns: Positive, Negative,
    """
    review_lower = review.lower(

    # Define keyword lists
    positive_keywords = [
        "amazing", "excellent",
        "fantastic", "perfect",
        "exceeded", "happy", "im
    ]

    negative_keywords = [
        "terrible", "bad", "wors
        "not satisfied", "disapp
        "disappointing", "issue"
    ]

    # Count keyword matches
    positive_count = sum(1 for k
    negative_count = sum(1 for k

    # Classify based on keyword
    if positive_count > negative_
        return "Positive"
    elif negative_count > posit
        return "Negative"
```



```python
def classify_sentiment(review):
    """
    Classifies sentiment of a review using keyword-based logic.
    Returns: Positive, Negative, or Neutral
    """
    review_lower = review.lower()

    # Define keyword lists
    positive_keywords = [
        "amazing", "excellent", "great", "love", "awesome", "wonderful",
        "fantastic", "perfect", "best", "good", "fast", "satisfied",
        "exceeded", "happy", "impressed", "quality", "value"
    ]

    negative_keywords = [
        "terrible", "bad", "worst", "hate", "poor", "damaged", "unhelpful",
        "not satisfied", "disappointed", "waste", "broken", "awful", "useless",
        "disappointing", "issue", "problem", "defective", "unhappy"
    ]

    # Count keyword matches
    positive_count = sum(1 for keyword in positive_keywords if keyword in review_lower)
    negative_count = sum(1 for keyword in negative_keywords if keyword in review_lower)

    # Classify based on keyword counts
    if positive_count > negative_count:
        return "Positive"
    elif negative_count > positive_count:
        return "Negative"
    else:
        return "Neutral"


# Main program
print("=" * 70)
print("Customer Review Sentiment Classifier")
print("=" * 70)

# Display existing reviews with classifications
```

```
NameError: name 'reviews' is not defined
                              ^^^^^^^
NameError: name 'reviews' is not defined
NameError: name 'reviews' is not defined
PS C:\Users\parva\OneDrive\Desktop\AI Assted> & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assted/ass 4.4.py"
PS C:\Users\parva\OneDrive\Desktop\AI Assted> & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assted/ass 4.4.py"
  File "c:\Users\parva\OneDrive\Desktop\AI Assted\ass 4.4.py", line 54
    Classifies sentiment of a review using keyword-based logic.
IndentationError: unexpected indent
PS C:\Users\parva\OneDrive\Desktop\AI Assted> 
```

Write Python code to classify customer review sentiment.

Example:
Input: "The product is very bad and I am unhappy"
Output: Negative

Now write the full Python program that classifies any review as
Positive, Negative, or Neutral using basic conditions.

ass 4.4.py

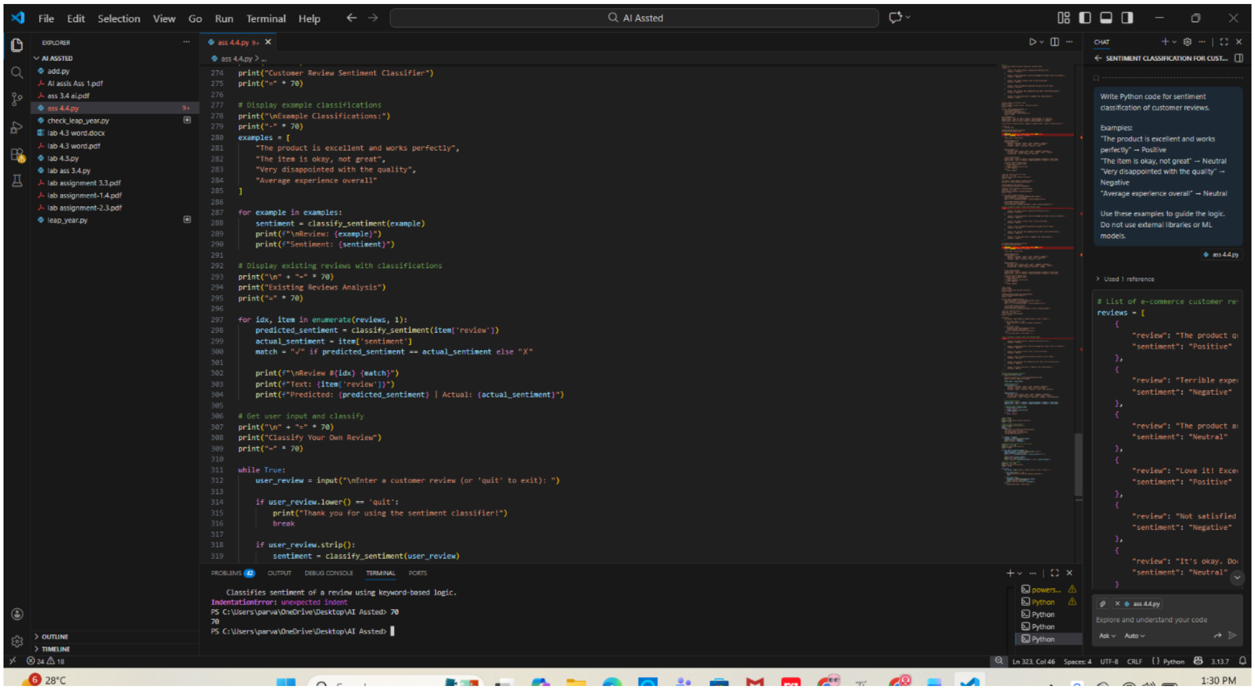
Used 1 reference

```python
# List of e-commerce customer re
reviews = [
    {
        "review": "The product q
        "sentiment": "Positive"
    },
    {
        "review": "Terrible expe
        "sentiment": "Negative"
    },
    {
        "review": "The product a
        "sentiment": "Neutral"
    },
    {
        "review": "Love it! Exce
        "sentiment": "Positive"
    },
    {
        "review": "Not satisfied
        "sentiment": "Negative"
    },
    {
        "review": "It's okay. Do
        "sentiment": "Neutral"
    }
]
```

ass 4.4.py

Explore and understand your code

Ask Auto



```python
print("Customer Review Sentiment Classifier")
print("=" * 70)

# Display example classifications
print("\nExample Classifications:")
print("-" * 70)
examples = [
    "The product is excellent and works perfectly",
    "The item is okay, not great",
    "Very disappointed with the quality",
    "Average experience overall"
]

for example in examples:
    sentiment = classify_sentiment(example)
    print(f"\nReview: {example}")
    print(f"Sentiment: {sentiment}")

# Display existing reviews with classifications
print("\n" + "=" * 70)
print("Existing Reviews Analysis")
print("=" * 70)

for idx, item in enumerate(reviews, 1):
    predicted_sentiment = classify_sentiment(item['review'])
    actual_sentiment = item['sentiment']
    match = "✓" if predicted_sentiment == actual_sentiment else "X"

    print(f"\nReview #{idx} {match}")
    print(f"Text: {item['review']}")
    print(f"Predicted: {predicted_sentiment} | Actual: {actual_sentiment}")

# Get user input and classify
print("\n" + "=" * 70)
print("Classify Your Own Review")
print("-" * 70)

while True:
    user_review = input("\nEnter a customer review (or 'quit' to exit): ")

    if user_review.lower() == 'quit':
        print("Thank you for using the sentiment classifier!")
        break

    if user_review.strip():
        sentiment = classify_sentiment(user_review)
```

```
    Classifies sentiment of a review using keyword-based logic.
IndentationError: unexpected indent
PS C:\Users\parva\OneDrive\Desktop\AI Assted> 70
70
PS C:\Users\parva\OneDrive\Desktop\AI Assted> 
```

Write Python code for sentiment classification of customer reviews.

Examples:
"The product is excellent and works perfectly" → Positive
"The item is okay, not great" → Neutral
"Very disappointed with the quality" → Negative
"Average experience overall" → Neutral

Use these examples to guide the logic.
Do not use external libraries or ML models.

ass 4.4.py

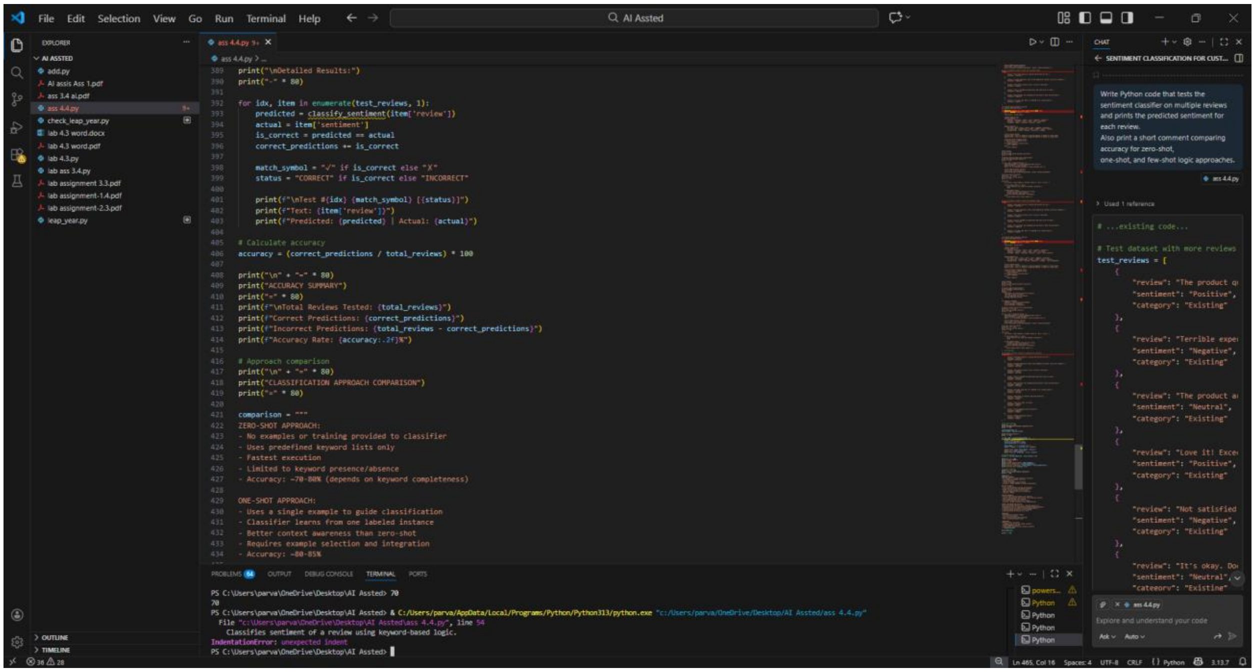
Used 1 reference

```python
# List of e-commerce customer re
reviews = [
    {
        "review": "The product q
        "sentiment": "Positive"
    },
    {
        "review": "Terrible expe
        "sentiment": "Negative"
    },
    {
        "review": "The product a
        "sentiment": "Neutral"
    },
    {
        "review": "Love it! Exce
        "sentiment": "Positive"
    },
    {
        "review": "Not satisfied
        "sentiment": "Negative"
    },
    {
        "review": "It's okay. Do
        "sentiment": "Neutral"
    }
```

ass 4.4.py

Explore and understand your code

Ask Auto



```python
print("\nDetailed Results:")
print("-" * 80)

for idx, item in enumerate(test_reviews, 1):
    predicted = classify_sentiment(item['review'])
    actual = item['sentiment']
    is_correct = predicted == actual
    correct_predictions += is_correct

    match_symbol = "√" if is_correct else "X"
    status = "CORRECT" if is_correct else "INCORRECT"

    print(f"\nTest #{idx} {match_symbol} [{status}]")
    print(f"Text: {item['review']}")
    print(f"Predicted: {predicted} | Actual: {actual}")

# Calculate accuracy
accuracy = (correct_predictions / total_reviews) * 100

print("\n" + "=" * 80)
print("ACCURACY SUMMARY")
print("=" * 80)
print(f"\nTotal Reviews Tested: {total_reviews}")
print(f"Correct Predictions: {correct_predictions}")
print(f"Incorrect Predictions: {total_reviews - correct_predictions}")
print(f"Accuracy Rate: {accuracy:.2f}%")

# Approach comparison
print("\n" + "=" * 80)
print("CLASSIFICATION APPROACH COMPARISON")
print("=" * 80)

comparison = """
ZERO-SHOT APPROACH:
- No examples or training provided to classifier
- Uses predefined keyword lists only
- Fastest execution
- Limited to keyword presence/absence
- Accuracy: ~70-80% (depends on keyword completeness)

ONE-SHOT APPROACH:
- Uses a single example to guide classification
- Classifier learns from one labeled instance
- Better context awareness than zero-shot
- Requires example selection and integration
- Accuracy: ~80-85%
```

```
PS C:\Users\parva\OneDrive\Desktop\AI Assted> 70
70
PS C:\Users\parva\OneDrive\Desktop\AI Assted> & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assted/ass 4.4.py"
  File "c:\Users\parva\OneDrive\Desktop\AI Assted\ass 4.4.py", line 54
    Classifies sentiment of a review using keyword-based logic.
IndentationError: unexpected indent
PS C:\Users\parva\OneDrive\Desktop\AI Assted> 
```

Write Python code that tests the sentiment classifier on multiple reviews and prints the predicted sentiment for each review.
Also print a short comment comparing accuracy for zero-shot, one-shot, and few-shot logic approaches.

ass 4.4.py

Used 1 reference

```python
# ...existing code...

# Test dataset with more reviews
test_reviews = [
    {
        "review": "The product qu
        "sentiment": "Positive",
        "category": "Existing"
    },
    {
        "review": "Terrible expe
        "sentiment": "Negative",
        "category": "Existing"
    },
    {
        "review": "The product a
        "sentiment": "Neutral",
        "category": "Existing"
    },
    {
        "review": "Love it! Exce
        "sentiment": "Positive",
        "category": "Existing"
    },
    {
        "review": "Not satisfied
        "sentiment": "Negative",
        "category": "Existing"
    },
    {
        "review": "It's okay. Do
        "sentiment": "Neutral",
        "category": "Existing"
```

2. Email Priority Classification

Scenario:

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

Tasks:

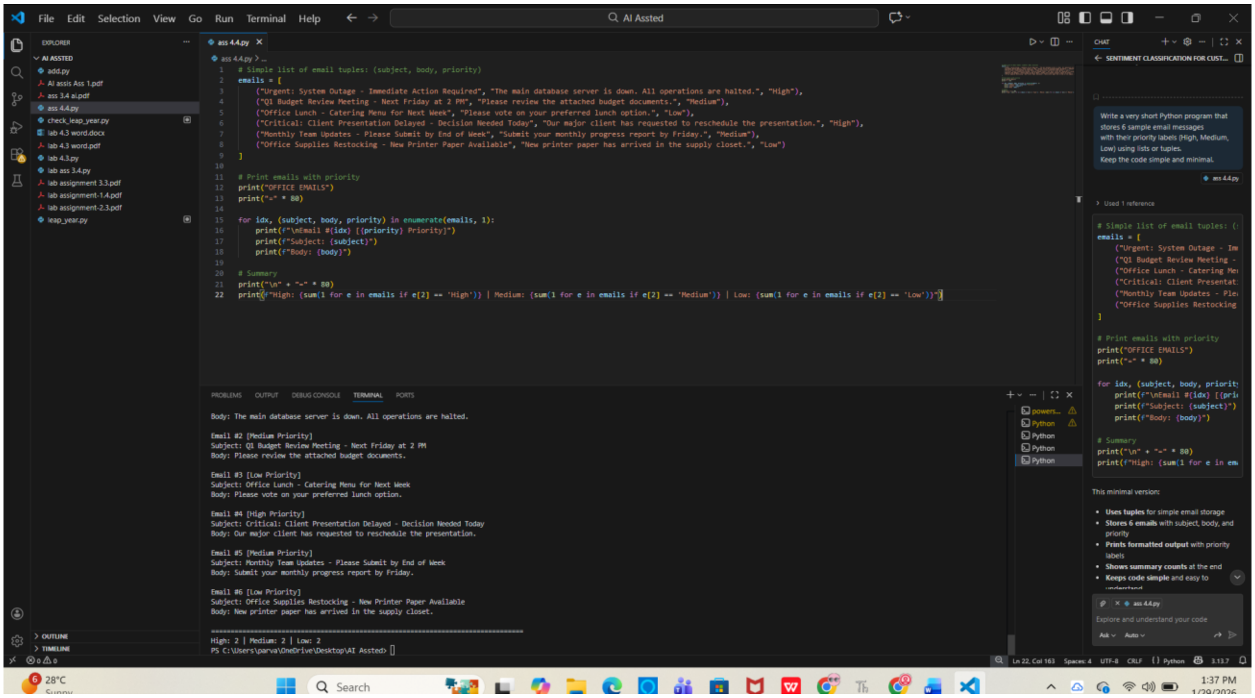
1. Create 6 sample email messages with priority labels.

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.



```python
# Simple list of email tuples: (subject, body, priority)
emails = [
    ("Urgent: System Outage - Immediate 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"\nEmail #{idx} [{priority} Priority]")
    print(f"Subject: {subject}")
    print(f"Body: {body}")

# Summary
print("\n" + "=" * 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')}")
```

```
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
PS C:\Users\parva\OneDrive\Desktop\AI Assted> 
```

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.

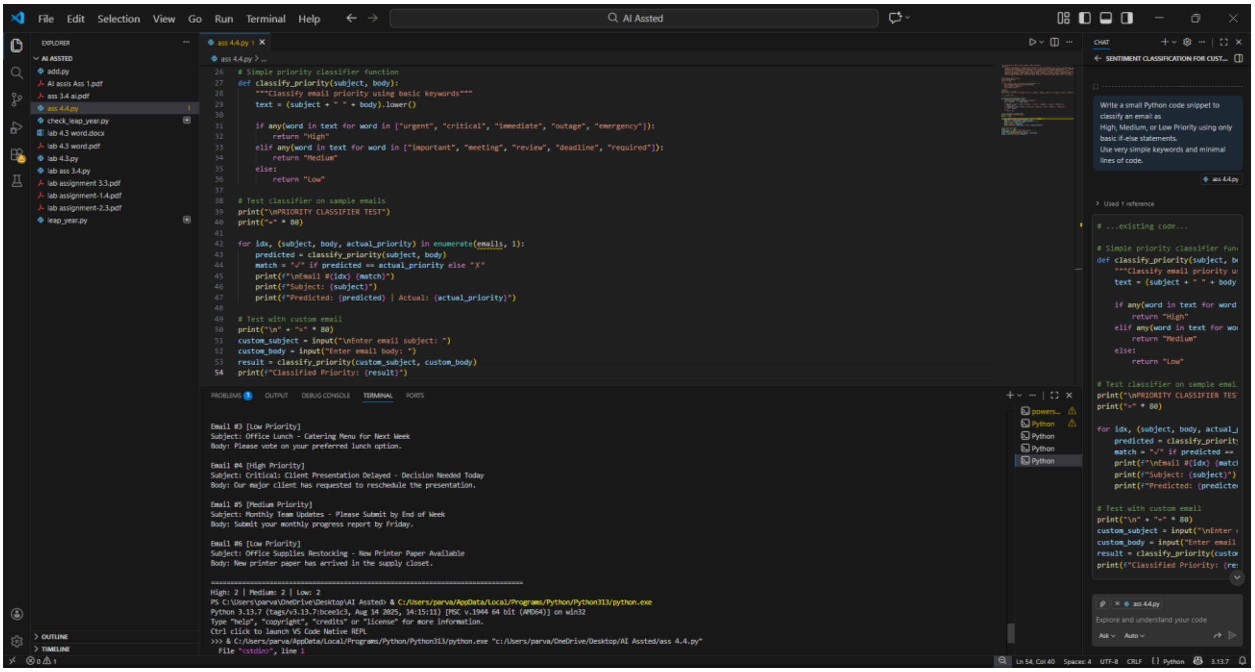
Used 1 reference

```python
# Simple list of email tuples: (:
emails = [
    ("Urgent: System Outage - Im
    ("Q1 Budget Review Meeting -
    ("Office Lunch - Catering Me
    ("Critical: Client Presentat
    ("Monthly Team Updates - Ple
    ("Office Supplies Restocking
]

# Print emails with priority
print("OFFICE EMAILS")
print("=" * 80)

for idx, (subject, body, priority
    print(f"\nEmail #{idx} [{pri
    print(f"Subject: {subject}")
    print(f"Body: {body}")

# Summary
print("\n" + "=" * 80)
print(f"High: {sum(1 for e in em
```

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

Explore and understand your code



```python
# Simple priority classifier function
def classify_priority(subject, body):
    """Classify email priority using basic keywords"""
    text = (subject + " " + body).lower()

    if any(word in text for word in ["urgent", "critical", "immediate", "outage", "emergency"]):
        return "High"
    elif any(word in text for word in ["important", "meeting", "review", "deadline", "required"]):
        return "Medium"
    else:
        return "Low"

# Test classifier on sample emails
print("\nPRIORITY CLASSIFIER TEST")
print("=" * 80)

for idx, (subject, body, actual_priority) in enumerate(emails, 1):
    predicted = classify_priority(subject, body)
    match = "✓" if predicted == actual_priority else "X"
    print(f"\nEmail #{idx} {match}")
    print(f"Subject: {subject}")
    print(f"Predicted: {predicted} | Actual: {actual_priority}")

# Test with custom email
print("\n" + "=" * 80)
custom_subject = input("\nEnter email subject: ")
custom_body = input("Enter email body: ")
result = classify_priority(custom_subject, custom_body)
print(f"Classified Priority: {result}")
```

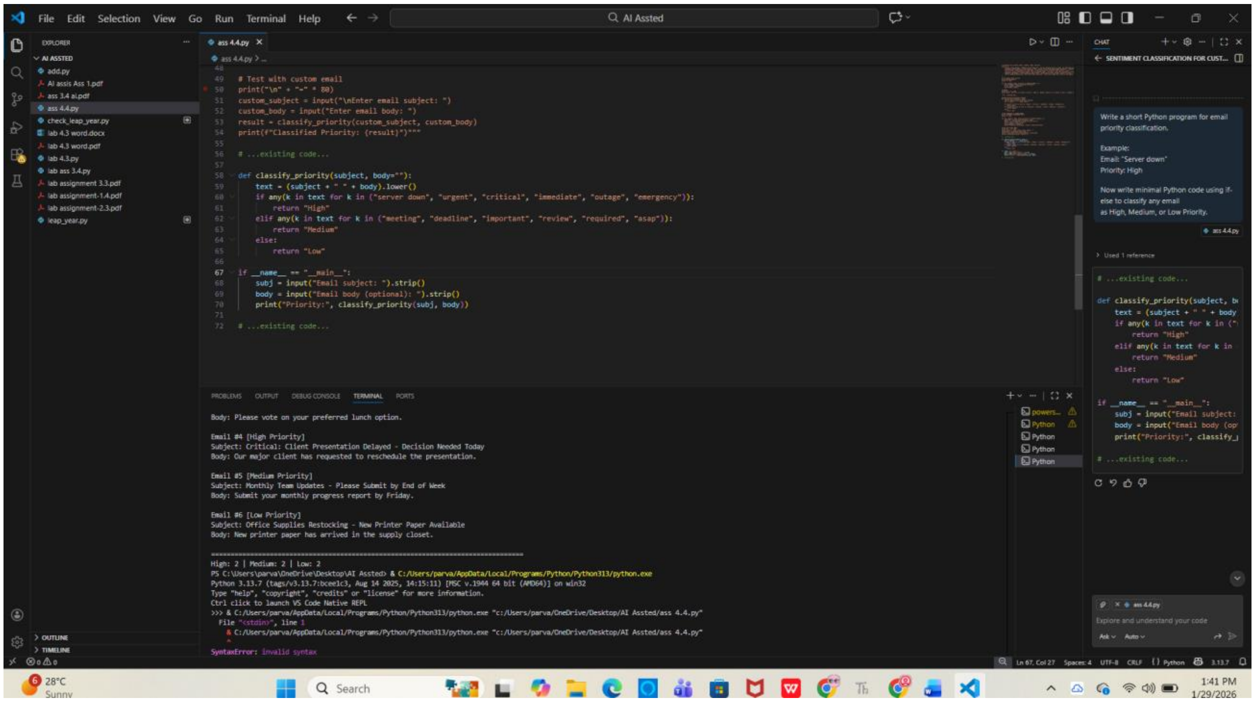
```
Email #3 [Low Priority]
Subject: Office Lunch - Catering Menu for Next Week
Body: Please vote on your preferred lunch option.

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
PS C:\Users\parva\OneDrive\Desktop\AI Assted> & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
Ctrl click to launch VS Code Native REPL
>>> & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assted/ass 4.4.py"
  File "<stdin>", line 1
```

Write a small Python code snippet to classify an email as
High, Medium, or Low Priority using only basic if-else statements.
Use very simple keywords and minimal lines of code.

Used 1 reference

```python
# ...existing code...

# Simple priority classifier fun
def classify_priority(subject, b
    """Classify email priority u
    text = (subject + " " + body

    if any(word in text for word
        return "High"
    elif any(word in text for wo
        return "Medium"
    else:
        return "Low"

# Test classifier on sample emai
print("\nPRIORITY CLASSIFIER TES
print("=" * 80)

for idx, (subject, body, actual_
    predicted = classify_priorit
    match = "✓" if predicted ==
    print(f"\nEmail #{idx} {matc
    print(f"Subject: {subject}")
    print(f"Predicted: {predicte

# Test with custom email
print("\n" + "=" * 80)
custom_subject = input("\nEnter
custom_body = input("Enter email
result = classify_priority(custo
print(f"Classified Priority: {re
```

Explore and understand your code



```python
# Test with custom email
print("\n" + "=" * 80)
custom_subject = input("\nEnter email subject: ")
custom_body = input("Enter email body: ")
result = classify_priority(custom_subject, custom_body)
print(f"Classified Priority: {result}")"""

# ...existing code...

def classify_priority(subject, body=""):
    text = (subject + " " + body).lower()
    if any(k in text for k in ("server down", "urgent", "critical", "immediate", "outage", "emergency")):
        return "High"
    elif any(k in text for k in ("meeting", "deadline", "important", "review", "required", "asap")):
        return "Medium"
    else:
        return "Low"

if __name__ == "__main__":
    subj = input("Email subject: ").strip()
    body = input("Email body (optional): ").strip()
    print("Priority:", classify_priority(subj, body))

# ...existing code...
```

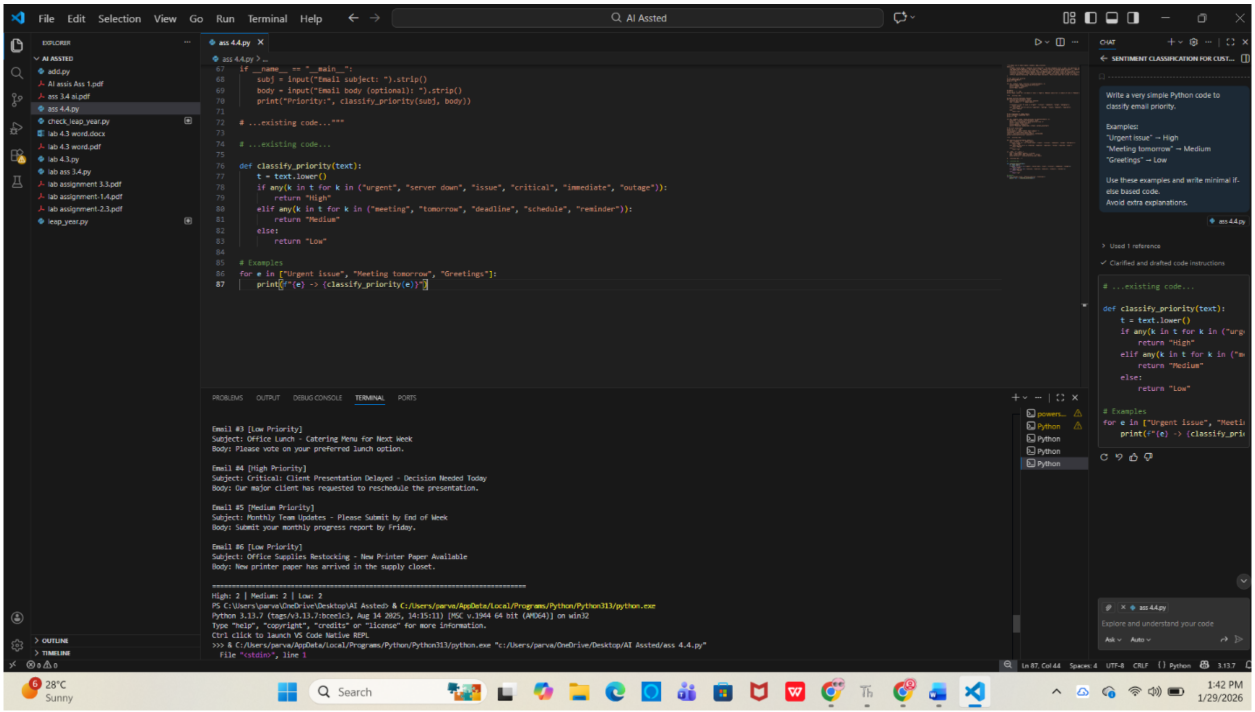
```
Body: Please vote on your preferred lunch option.

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
PS C:\Users\parva\OneDrive\Desktop\AI Assted> & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
Ctrl click to launch VS Code Native REPL
>>> & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assted/ass 4.4.py"
  File "<stdin>", line 1
    & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assted/ass 4.4.py"
    ^
SyntaxError: invalid syntax
```

Write a short Python program for email priority classification.

Example:
Email: "Server down"
Priority: High

Now write minimal Python code using if-else to classify any email
as High, Medium, or Low Priority.

```python
# ...existing code...

def classify_priority(subject, b
    text = (subject + " " + body
    if any(k in text for k in ("
        return "High"
    elif any(k in text for k in
        return "Medium"
    else:
        return "Low"

if __name__ == "__main__":
    subj = input("Email subject:
    body = input("Email body (op
    print("Priority:", classify_

# ...existing code...
```



```python
if __name__ == "__main__":
    subj = input("Email subject: ").strip()
    body = input("Email body (optional): ").strip()
    print("Priority:", classify_priority(subj, body))

# ...existing code..."""

# ...existing code...

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)}")
```

```
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
PS C:\Users\parva\OneDrive\Desktop\AI Assted> & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
Ctrl click to launch VS Code Native REPL
>>> & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assted/ass 4.4.py"
  File "<stdin>", line 1
```

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.

```python
# ...existing code...

def classify_priority(text):
    t = text.lower()
    if any(k in t for k in ("urg
        return "High"
    elif any(k in t for k in ("m
        return "Medium"
    else:
        return "Low"

# Examples
for e in ["Urgent issue", "Meeti
    print(f"{e} -> {classify_pri
```

3. Student Query Routing System

Scenario:

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

Tasks:

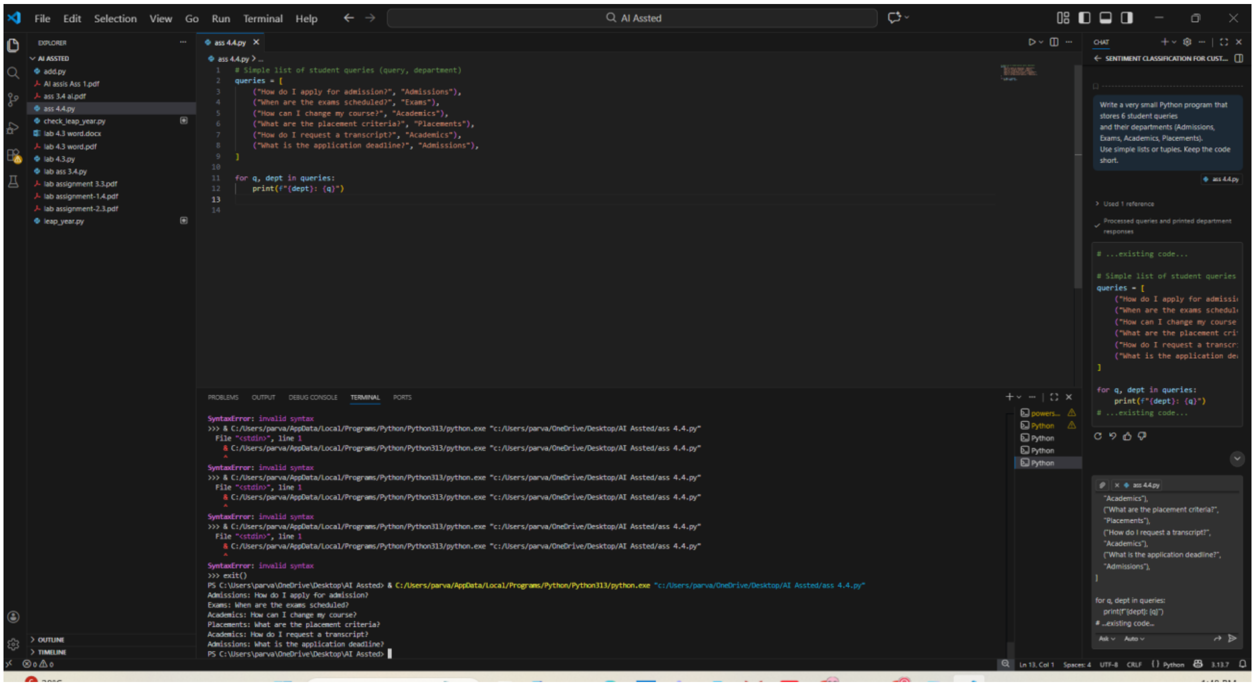
1. Create 6 sample student queries mapped to departments.

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.



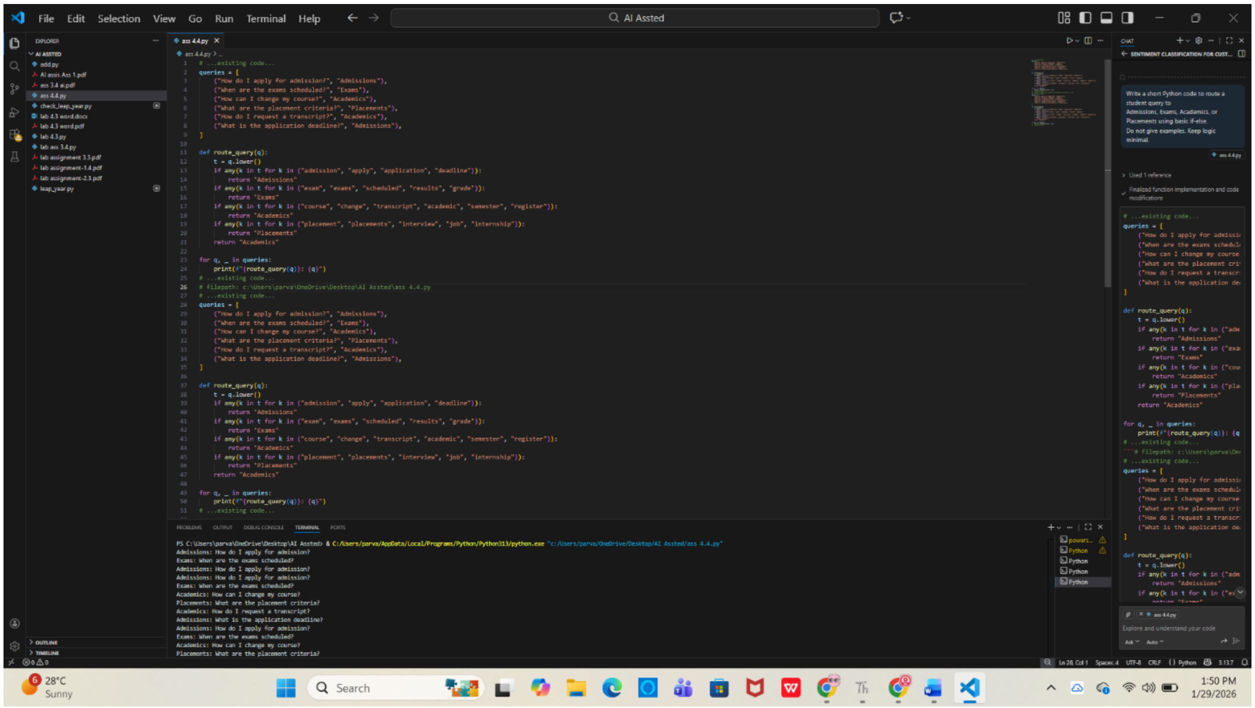
```python
# Simple list of student queries (query, department)
queries = [
    ("How do I apply for admission?", "Admissions"),
    ("When are the exams scheduled?", "Exams"),
    ("How can I change my course?", "Academics"),
    ("What are the placement criteria?", "Placements"),
    ("How do I request a transcript?", "Academics"),
    ("What is the application deadline?", "Admissions"),
]

for q, dept in queries:
    print(f"{dept}: {q}")
```

```
SyntaxError: invalid syntax
>>> & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assted/ass 4.4.py"
  File "<stdin>", line 1
    & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assted/ass 4.4.py"
    ^
SyntaxError: invalid syntax
>>> & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assted/ass 4.4.py"
  File "<stdin>", line 1
    & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assted/ass 4.4.py"
    ^
SyntaxError: invalid syntax
>>> & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assted/ass 4.4.py"
  File "<stdin>", line 1
    & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assted/ass 4.4.py"
    ^
SyntaxError: invalid syntax
>>> exit()
PS C:\Users\parva\OneDrive\Desktop\AI Assted> & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assted/ass 4.4.py"
Admissions: How do I apply for admission?
Exams: When are the exams scheduled?
Academics: How can I change my course?
Placements: What are the placement criteria?
Academics: How do I request a transcript?
Admissions: What is the application deadline?
PS C:\Users\parva\OneDrive\Desktop\AI Assted>
```

Write a very small Python program that stores 6 student queries
and their departments (Admissions, Exams, Academics, Placements).
Use simple lists or tuples. Keep the code short.

```python
# ...existing code...

# Simple list of student queries
queries = [
    ("How do I apply for admissi
    ("When are the exams schedul
    ("How can I change my course
    ("What are the placement cri
    ("How do I request a transcr
    ("What is the application de
]

for q, dept in queries:
    print(f"{dept}: {q}")
# ...existing code...
```

```
"Academics"),
("What are the placement criteria?",
"Placements"),
("How do I request a transcript?",
"Academics"),
("What is the application deadline?",
"Admissions"),
]

for q, dept in queries:
  print(f"{dept}: {q}")
# ...existing code...
```



```python
# ...existing code...
queries = [
    ("How do I apply for admission?", "Admissions"),
    ("When are the exams scheduled?", "Exams"),
    ("How can I change my course?", "Academics"),
    ("What are the placement criteria?", "Placements"),
    ("How do I request a transcript?", "Academics"),
    ("What is the application deadline?", "Admissions"),
]

def route_query(q):
    t = q.lower()
    if any(k in t for k in ("admission", "apply", "application", "deadline")):
        return "Admissions"
    if any(k in t for k in ("exam", "exams", "scheduled", "results", "grade")):
        return "Exams"
    if any(k in t for k in ("course", "change", "transcript", "academic", "semester", "register")):
        return "Academics"
    if any(k in t for k in ("placement", "placements", "interview", "job", "internship")):
        return "Placements"
    return "Academics"

for q, _ in queries:
    print(f"{route_query(q)}: {q}")
# ...existing code...
# filepath: c:\Users\parva\OneDrive\Desktop\AI Assted\ass 4.4.py
# ...existing code...
queries = [
    ("How do I apply for admission?", "Admissions"),
    ("When are the exams scheduled?", "Exams"),
    ("How can I change my course?", "Academics"),
    ("What are the placement criteria?", "Placements"),
    ("How do I request a transcript?", "Academics"),
    ("What is the application deadline?", "Admissions"),
]

def route_query(q):
    t = q.lower()
    if any(k in t for k in ("admission", "apply", "application", "deadline")):
        return "Admissions"
    if any(k in t for k in ("exam", "exams", "scheduled", "results", "grade")):
        return "Exams"
    if any(k in t for k in ("course", "change", "transcript", "academic", "semester", "register")):
        return "Academics"
    if any(k in t for k in ("placement", "placements", "interview", "job", "internship")):
        return "Placements"
    return "Academics"

for q, _ in queries:
    print(f"{route_query(q)}: {q}")
# ...existing code...
```

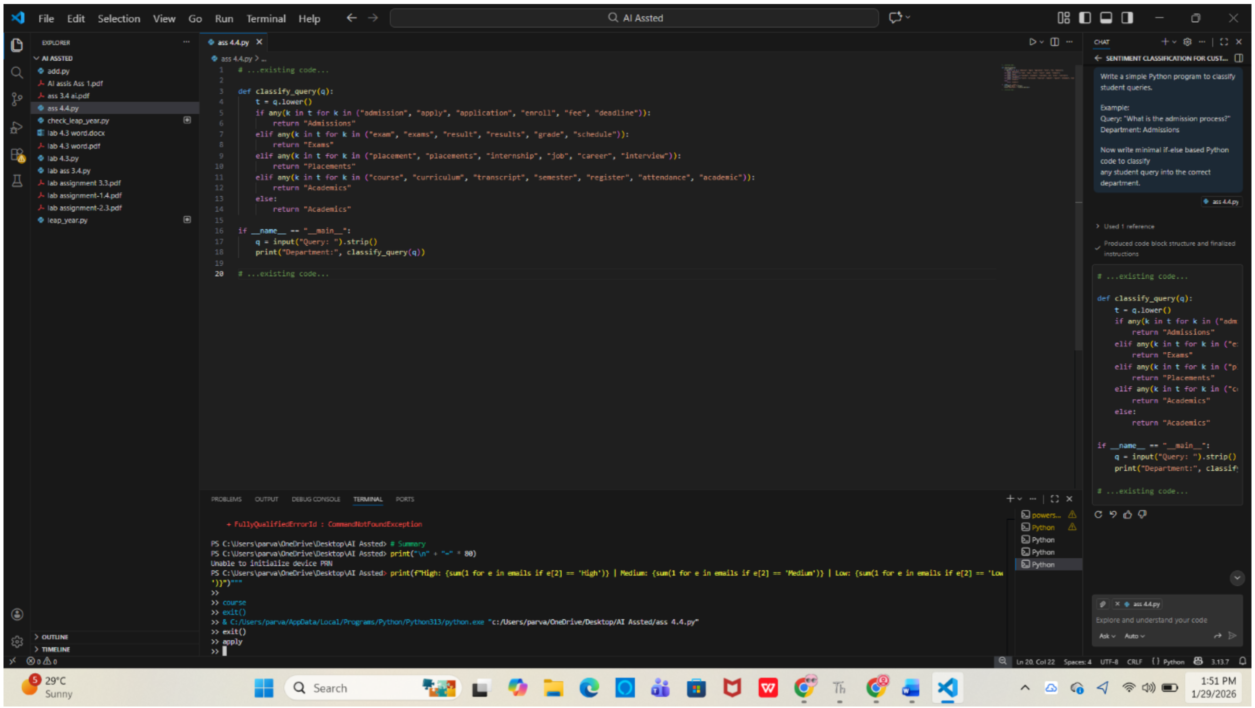
```
PS C:\Users\parva\OneDrive\Desktop\AI Assted> & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assted/ass 4.4.py"
Admissions: How do I apply for admission?
Exams: When are the exams scheduled?
Admissions: How do I apply for admission?
Admissions: How do I apply for admission?
Exams: When are the exams scheduled?
Academics: How can I change my course?
Placements: What are the placement criteria?
Academics: How do I request a transcript?
Admissions: What is the application deadline?
Admissions: How do I apply for admission?
Exams: When are the exams scheduled?
Academics: How can I change my course?
Placements: What are the placement criteria?
```

Write a short Python code to route a student query to
Admissions, Exams, Academics, or Placements using basic if-else.
Do not give examples. Keep logic minimal.

ass 4.4.py

Used 1 reference

✓ Finalized function implementation and code modifications



```python
# ...existing code...

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 "Placements"
    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))

# ...existing code...
```

```
    + FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\parva\OneDrive\Desktop\AI Assted> # Summary
PS C:\Users\parva\OneDrive\Desktop\AI Assted> print("\n" + "=" * 80)
Unable to initialize device PRN
PS C:\Users\parva\OneDrive\Desktop\AI Assted> 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
')}")"""
>>
>> course
>> exit()
>> & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assted/ass 4.4.py"
>> exit()
>> apply
>>
```

Write a simple Python program to classify student queries.

Example:
Query: "What is the admission process?"
Department: Admissions

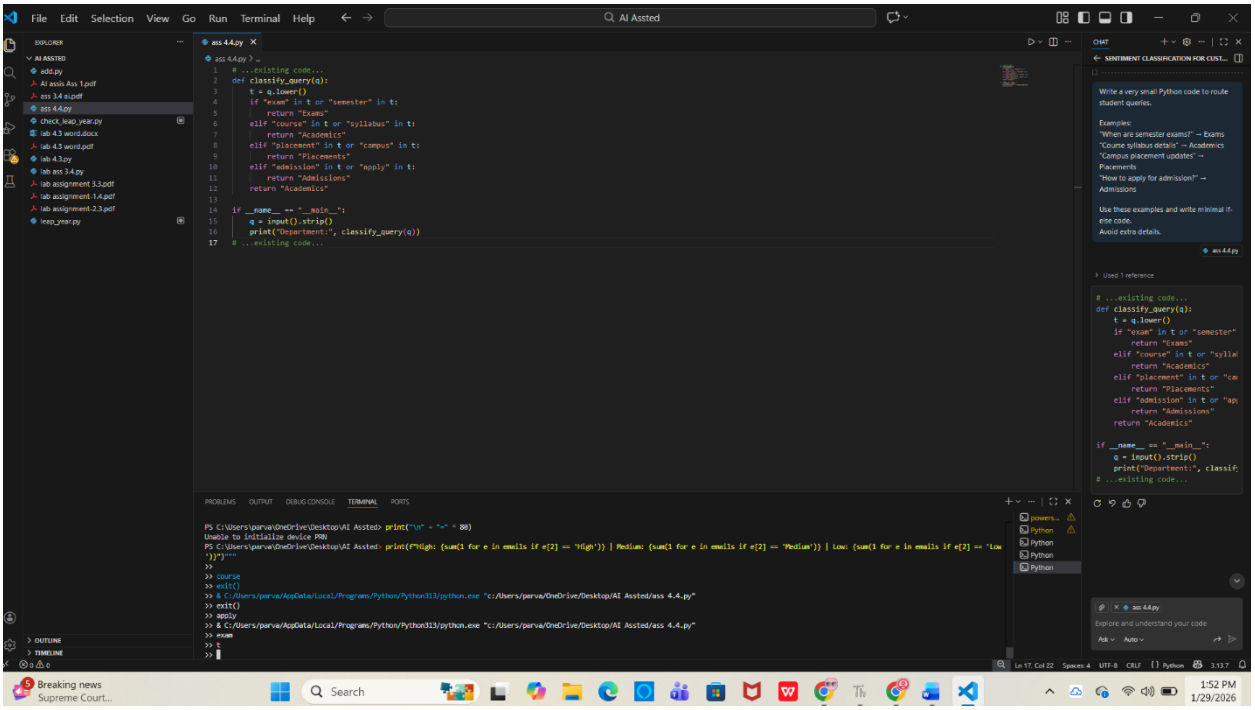
Now write minimal if-else based Python code to classify
any student query into the correct department.

ass 4.4.py

Used 1 reference

✓ Produced code block structure and finalized instructions

```python
# ...existing code...

def classify_query(q):
    t = q.lower()
    if any(k in t for k in ("adm
        return "Admissions"
    elif any(k in t for k in ("e
        return "Exams"
    elif any(k in t for k in ("p
        return "Placements"
    elif any(k in t for k in ("c
        return "Academics"
    else:
        return "Academics"

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

# ...existing code...
```



```python
# ...existing code...
def classify_query(q):
    t = q.lower()
    if "exam" in t or "semester" in t:
        return "Exams"
    elif "course" in t or "syllabus" in t:
        return "Academics"
    elif "placement" in t or "campus" in t:
        return "Placements"
    elif "admission" in t or "apply" in t:
        return "Admissions"
    return "Academics"

if __name__ == "__main__":
    q = input().strip()
    print("Department:", classify_query(q))
# ...existing code...
```

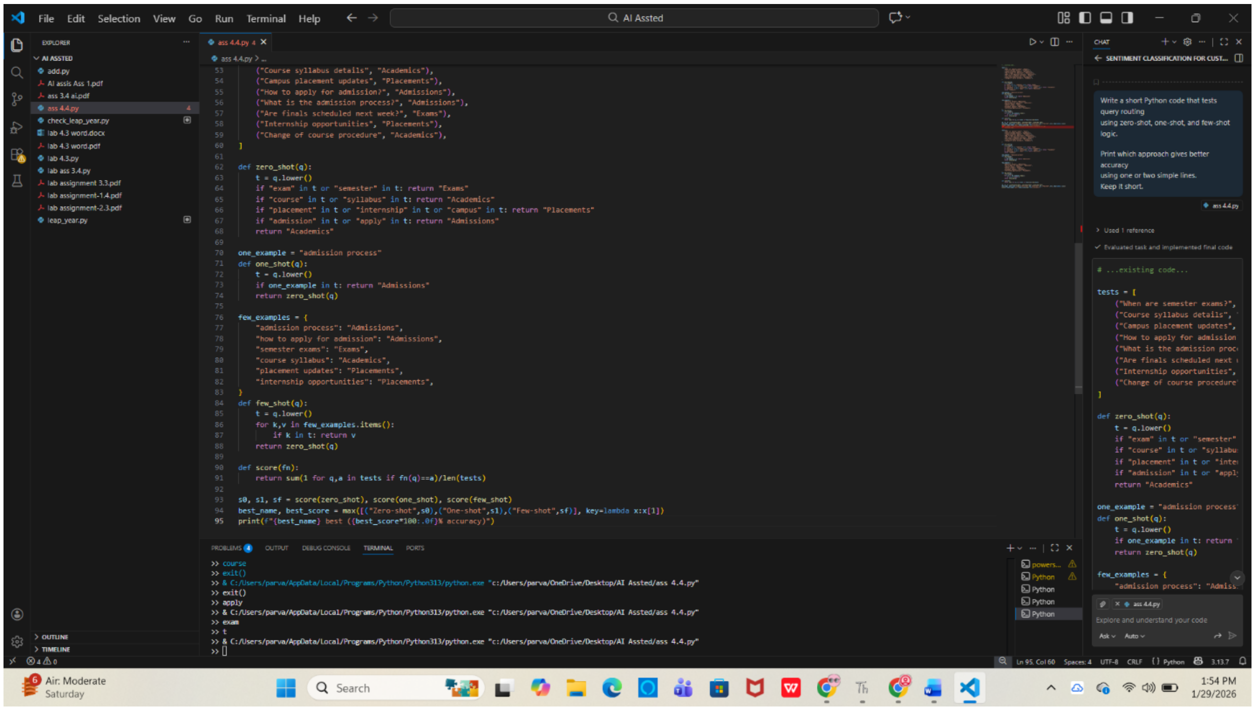
```
PS C:\Users\parva\OneDrive\Desktop\AI Assted> print("\n" + "=" * 80)
Unable to initialize device PRN
PS C:\Users\parva\OneDrive\Desktop\AI Assted> print(f"High: {sum(1 for e in emails if e[2] == 'High')} | Medium: {sum(1 for e in emails if e[2] == 'Medium')} | Low: {sum(1 for e in emails if e[2] == 'Low
')}")"""
>>
>> course
>> exit()
>> & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assted/ass 4.4.py"
>> exit()
>> apply
>> & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assted/ass 4.4.py"
>> exam
>> t
>> 
```

Write a very small Python code to route student queries.

Examples:
"When are semester exams?" → Exams
"Course syllabus details" → Academics
"Campus placement updates" → Placements
"How to apply for admission?" → Admissions

Use these examples and write minimal if-else code.
Avoid extra details.

```python
# ...existing code...
def classify_query(q):
    t = q.lower()
    if "exam" in t or "semester"
        return "Exams"
    elif "course" in t or "syllal
        return "Academics"
    elif "placement" in t or "cai
        return "Placements"
    elif "admission" in t or "ap
        return "Admissions"
    return "Academics"

if __name__ == "__main__":
    q = input().strip()
    print("Department:", classif
# ...existing code...
```



```python
    ("Course syllabus details", "Academics"),
    ("Campus placement updates", "Placements"),
    ("How to apply for admission?", "Admissions"),
    ("What is the admission process?", "Admissions"),
    ("Are finals scheduled next week?", "Exams"),
    ("Internship opportunities", "Placements"),
    ("Change of course procedure", "Academics"),
]

def zero_shot(q):
    t = q.lower()
    if "exam" in t or "semester" in t: return "Exams"
    if "course" in t or "syllabus" in t: return "Academics"
    if "placement" in t or "internship" in t or "campus" in t: return "Placements"
    if "admission" in t or "apply" in t: return "Admissions"
    return "Academics"

one_example = "admission process"
def one_shot(q):
    t = q.lower()
    if one_example in t: return "Admissions"
    return zero_shot(q)

few_examples = {
    "admission process": "Admissions",
    "how to apply for admission": "Admissions",
    "semester exams": "Exams",
    "course syllabus": "Academics",
    "placement updates": "Placements",
    "internship opportunities": "Placements",
}
def few_shot(q):
    t = q.lower()
    for k,v in few_examples.items():
        if k in t: return v
    return zero_shot(q)

def score(fn):
    return sum(1 for q,a in tests if fn(q)==a)/len(tests)

s0, s1, sf = score(zero_shot), score(one_shot), score(few_shot)
best_name, best_score = max([("Zero-shot",s0),("One-shot",s1),("Few-shot",sf)], key=lambda x:x[1])
print(f"{best_name} best ({best_score*100:.0f}% accuracy)")
```

```
>> course
>> exit()
>> & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assted/ass 4.4.py"
>> exit()
>> apply
>> & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assted/ass 4.4.py"
>> exam
>> t
>> & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assted/ass 4.4.py"
>> 
```

Write a short Python code that tests query routing
using zero-shot, one-shot, and few-shot logic.

Print which approach gives better accuracy
using one or two simple lines.
Keep it short.

```python
# ...existing code...

tests = [
    ("When are semester exams?",
    ("Course syllabus details",
    ("Campus placement updates",
    ("How to apply for admission
    ("What is the admission proc
    ("Are finals scheduled next
    ("Internship opportunities",
    ("Change of course procedure
]

def zero_shot(q):
    t = q.lower()
    if "exam" in t or "semester"
    if "course" in t or "syllabu
    if "placement" in t or "inte
    if "admission" in t or "appl
    return "Academics"

one_example = "admission process
def one_shot(q):
    t = q.lower()
    if one_example in t: return
    return zero_shot(q)

few_examples = {
    "admission process": "Admiss
```

4. Chatbot Question Type Detection

Scenario:

A chatbot must identify whether a user query is Informational,

Transactional, Complaint, or Feedback.

Tasks:

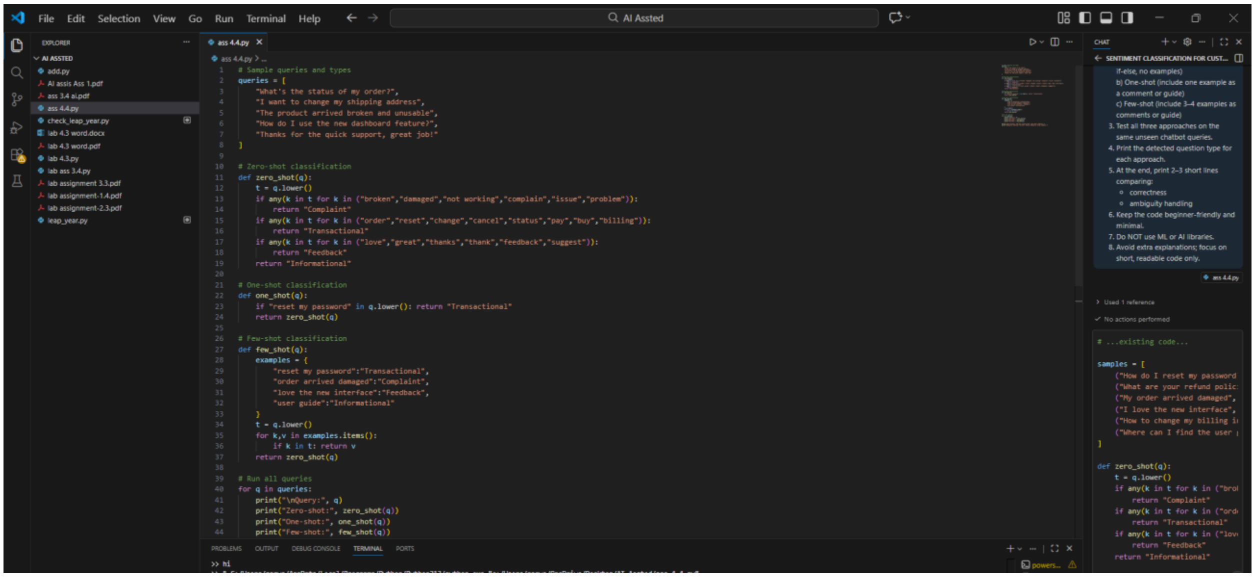
1. Prepare 6 chatbot queries mapped to question types.

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.



```python
# Sample 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(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"

# 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("\nQuery:", q)
    print("Zero-shot:", zero_shot(q))
    print("One-shot:", one_shot(q))
    print("Few-shot:", few_shot(q))
```

```
>> hi
```

if-else, no examples)
b) One-shot (include 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 unseen chatbot queries.
4. Print the detected question type for each approach.
5. At the end, print 2–3 short lines comparing:
   - correctness
   - ambiguity handling
6. Keep the code beginner-friendly and minimal.
7. Do NOT use ML or AI libraries.
8. Avoid extra explanations; focus on short, readable code only.

```python
# ...existing code...

samples = [
    ("How do I reset my password
    ("What are your refund polic
    ("My order arrived damaged",
    ("I love the new interface",
    ("How to change my billing i
    ("Where can I find the user
]

def zero_shot(q):
    t = q.lower()
    if any(k in t for k in ("brol
        return "Complaint"
    if any(k in t for k in ("ord
        return "Transactional"
    if any(k in t for k in ("lov
        return "Feedback"
    return "Informational"
```

5. Emotion Detection in Text

Scenario:

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

Anxious, Neutral.

Tasks:

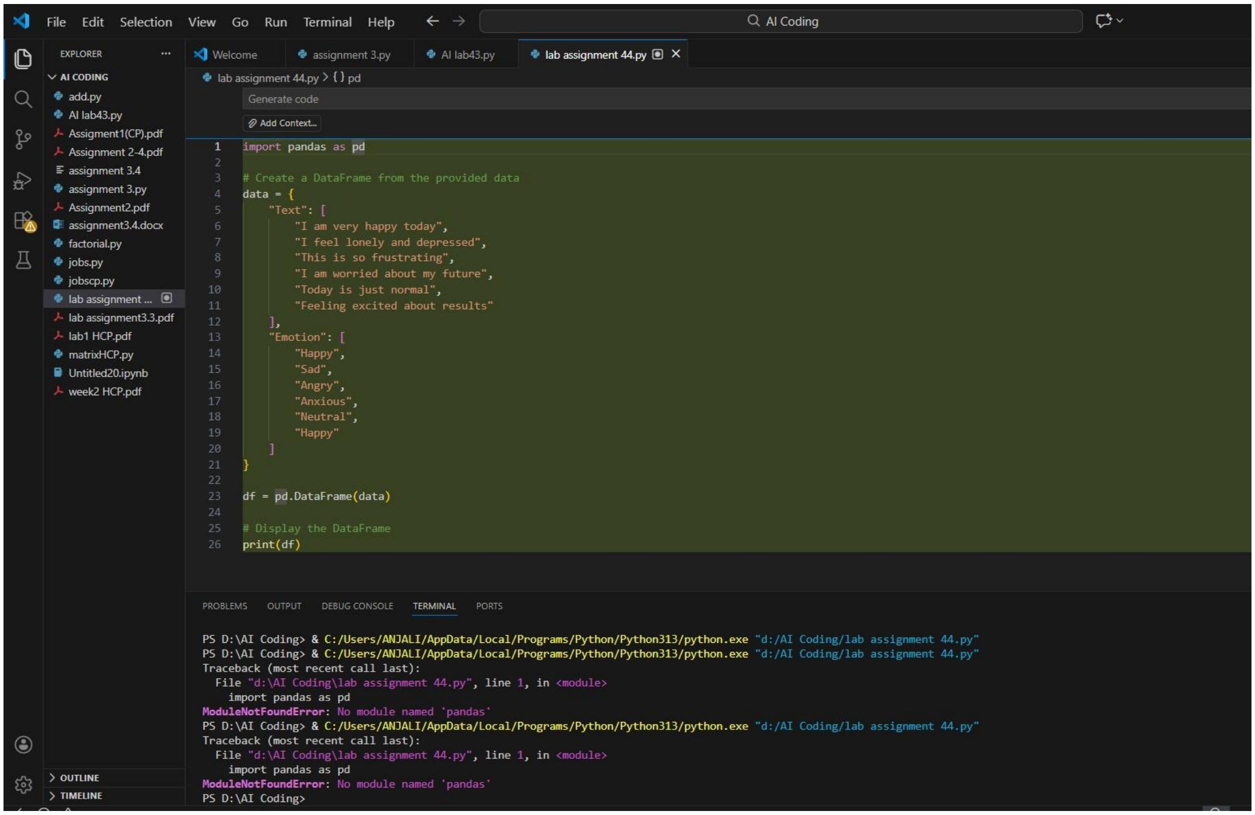
1. Create labeled emotion samples.

2. Use Zero-shot prompting to identify emotions.

3. Use One-shot prompting with an example.

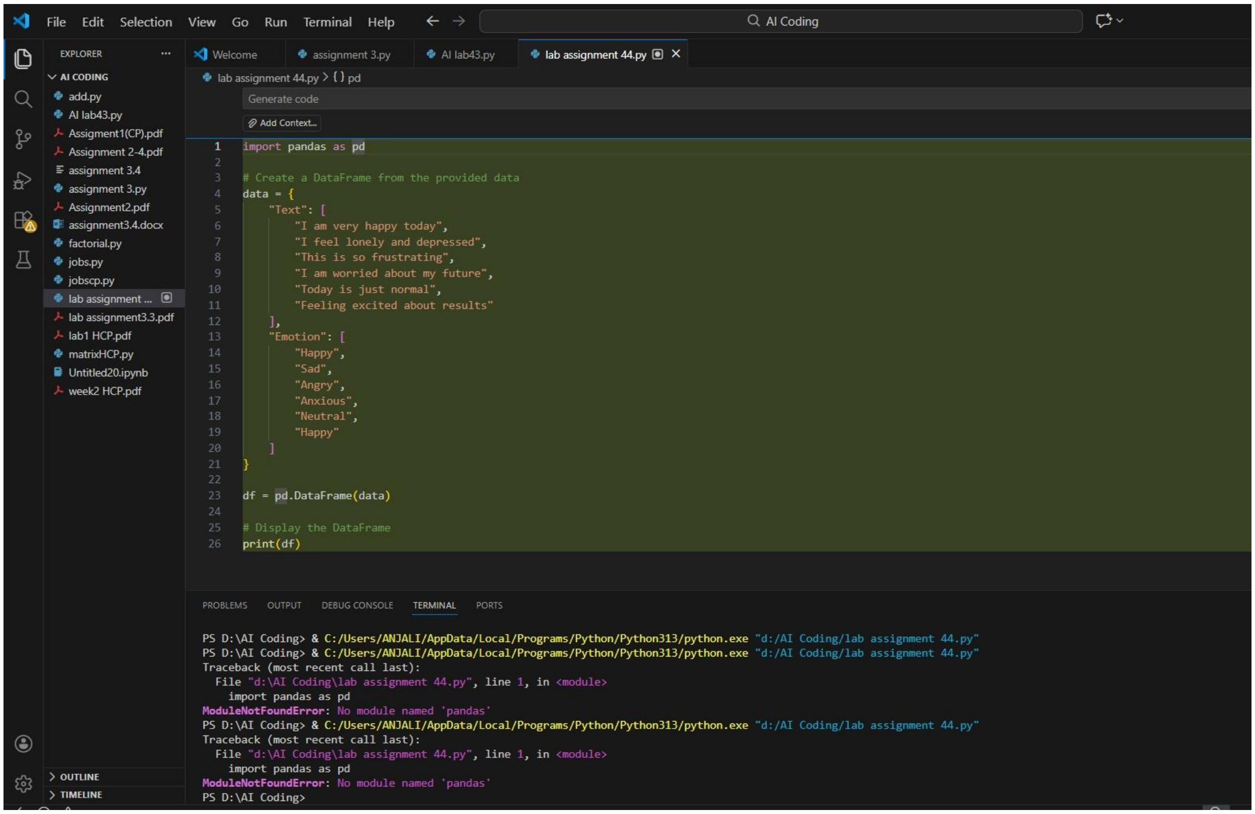
4. Use Few-shot prompting with multiple emotions.

5. Discuss ambiguity handling across techniques.



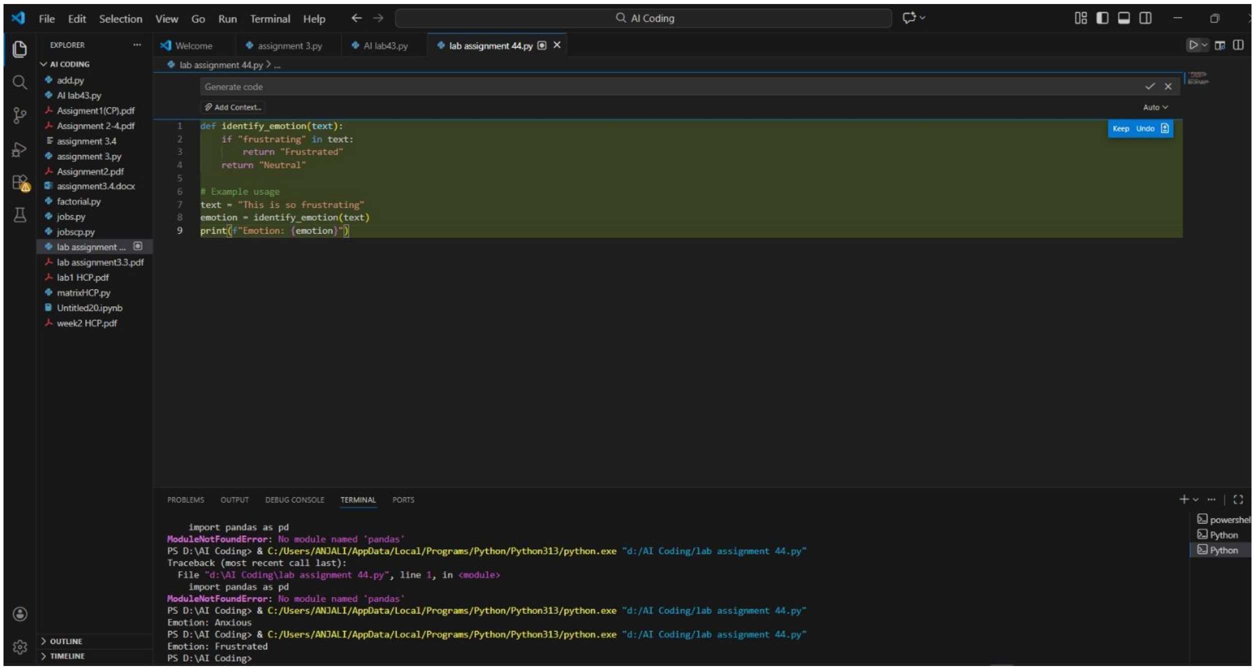
```python
import pandas as pd

# Create a DataFrame from the provided data
data = {
    "Text": [
        "I am very happy today",
        "I feel lonely and depressed",
        "This is so frustrating",
        "I am worried about my future",
        "Today is just normal",
        "Feeling excited about results"
    ],
    "Emotion": [
        "Happy",
        "Sad",
        "Angry",
        "Anxious",
        "Neutral",
        "Happy"
    ]
}

df = pd.DataFrame(data)

# Display the DataFrame
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>
```



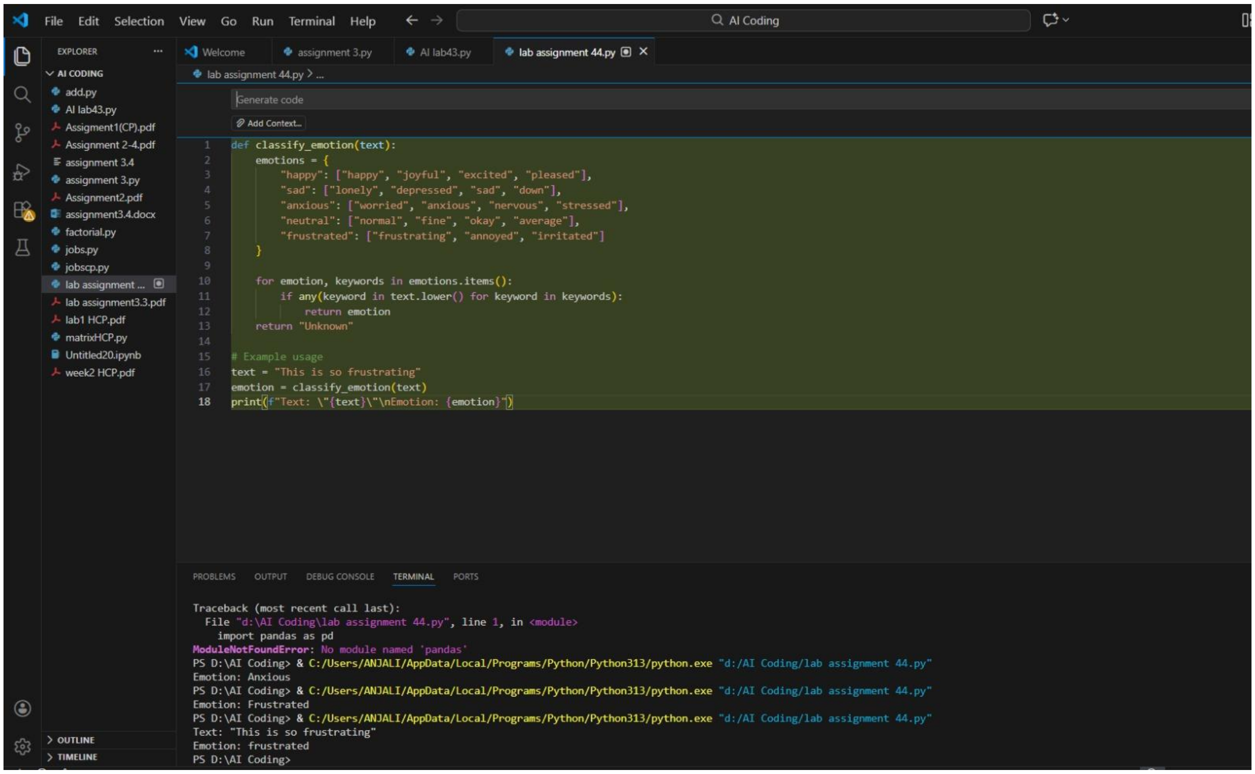
```python
import pandas as pd

# Create a DataFrame from the provided data
data = {
    "Text": [
        "I am very happy today",
        "I feel lonely and depressed",
        "This is so frustrating",
        "I am worried about my future",
        "Today is just normal",
        "Feeling excited about results"
    ],
    "Emotion": [
        "Happy",
        "Sad",
        "Angry",
        "Anxious",
        "Neutral",
        "Happy"
    ]
}

df = pd.DataFrame(data)

# Display the DataFrame
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>
```



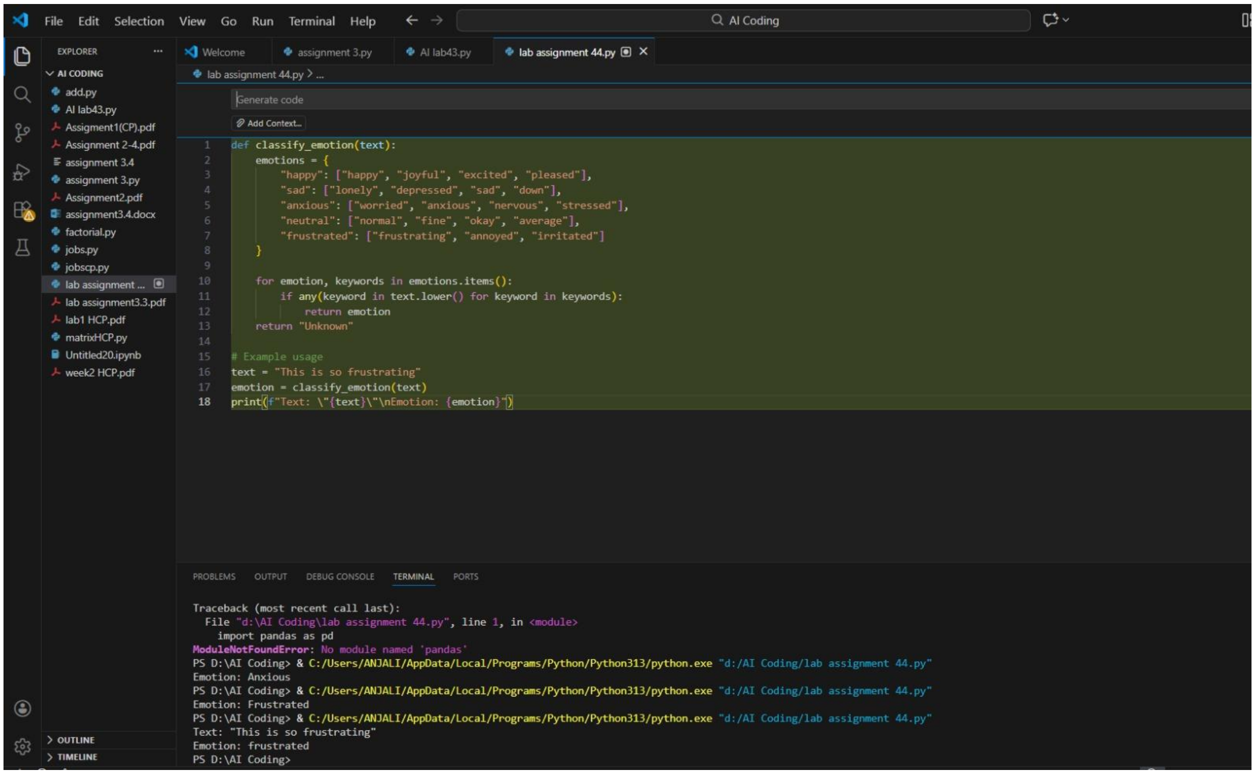
```python
def identify_emotion(text):
    if "frustrating" in text:
        return "Frustrated"
    return "Neutral"

# Example usage
text = "This is so frustrating"
emotion = identify_emotion(text)
print(f"Emotion: {emotion}")
```

```
    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>
```



```python
def classify_emotion(text):
    emotions = {
        "happy": ["happy", "joyful", "excited", "pleased"],
        "sad": ["lonely", "depressed", "sad", "down"],
        "anxious": ["worried", "anxious", "nervous", "stressed"],
        "neutral": ["normal", "fine", "okay", "average"],
        "frustrated": ["frustrating", "annoyed", "irritated"]
    }

    for emotion, keywords in emotions.items():
        if any(keyword in text.lower() for keyword in keywords):
            return emotion
    return "Unknown"

# Example usage
text = "This is so frustrating"
emotion = classify_emotion(text)
print(f"Text: \"{text}\"\nEmotion: {emotion}")
```

```
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>
```



```python
def classify_emotion(text):
    emotions = {
        "happy": ["happy", "joyful", "excited", "pleased"],
        "sad": ["lonely", "depressed", "sad", "down"],
        "anxious": ["worried", "anxious", "nervous", "stressed"],
        "neutral": ["normal", "fine", "okay", "average"],
        "frustrated": ["frustrating", "annoyed", "irritated"]
    }

    for emotion, keywords in emotions.items():
        if any(keyword in text.lower() for keyword in keywords):
            return emotion
    return "Unknown"

# Example usage
text = "This is so frustrating"
emotion = classify_emotion(text)
print(f"Text: \"{text}\"\nEmotion: {emotion}")
```

```
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>
```