AI ASSISSTANT CODING LAB 4

Name:M.Shashidhar Batch:42

Roll no : 2303A52291

Task 1: Zero and one shot

Task 2: One and few

Task 3: only one shot

Task 4:only few shot

Task 5: one and few shot

Task1: Scenario:

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

Week2

them into Positive, Negative, or Neutral sentiments using prompt

engineering.

Zero shot prompt:

Classify the sentiment of the following review as Positive, Negative, or Neutral: Review: review_text

Generated code

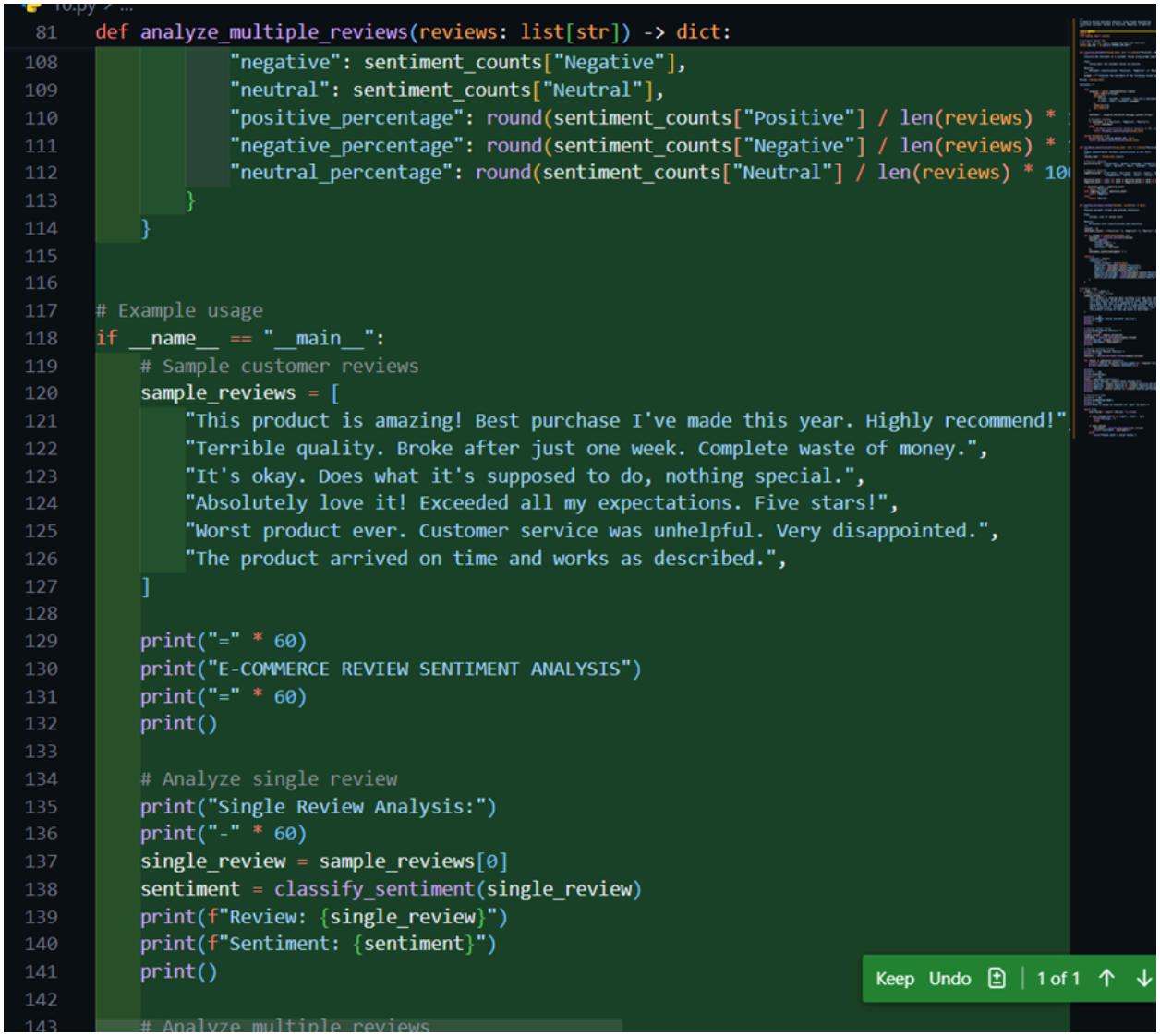
```python
def classify_sentiment(review_text: str) -> Literal["Positive", "Negative", "Neutral"]:
            temperature=0,
            max_tokens=10
        )

        sentiment = response.choices[0].message.content.strip()

        # Validate response
        if sentiment in ["Positive", "Negative", "Neutral"]:
            return sentiment
        else:
            # Fallback classification based on keywords if API response is unexpected
            return fallback_classification(review_text)

    except Exception as e:
        print(f"Error calling OpenAI API: {e}")
        return fallback_classification(review_text)


def fallback_classification(review_text: str) -> Literal["Positive", "Negative", "Neutr
    """
    Simple keyword-based fallback classification if API fails.
    """
    review_lower = review_text.lower()

    # Positive keywords
    positive_words = ["excellent", "great", "amazing", "wonderful", "fantastic",
                      "love", "perfect", "best", "awesome", "satisfied", "recommend"]

    # Negative keywords
    negative_words = ["terrible", "horrible", "worst", "awful", "bad", "poor",
                      "disappointed", "waste", "never", "refund", "broken", "useless"]

    positive_count = sum(1 for word in positive_words if word in review_lower)
    negative_count = sum(1 for word in negative_words if word in revi

    if positive_count > negative_count:
```

Keep Undo | 1 of 1

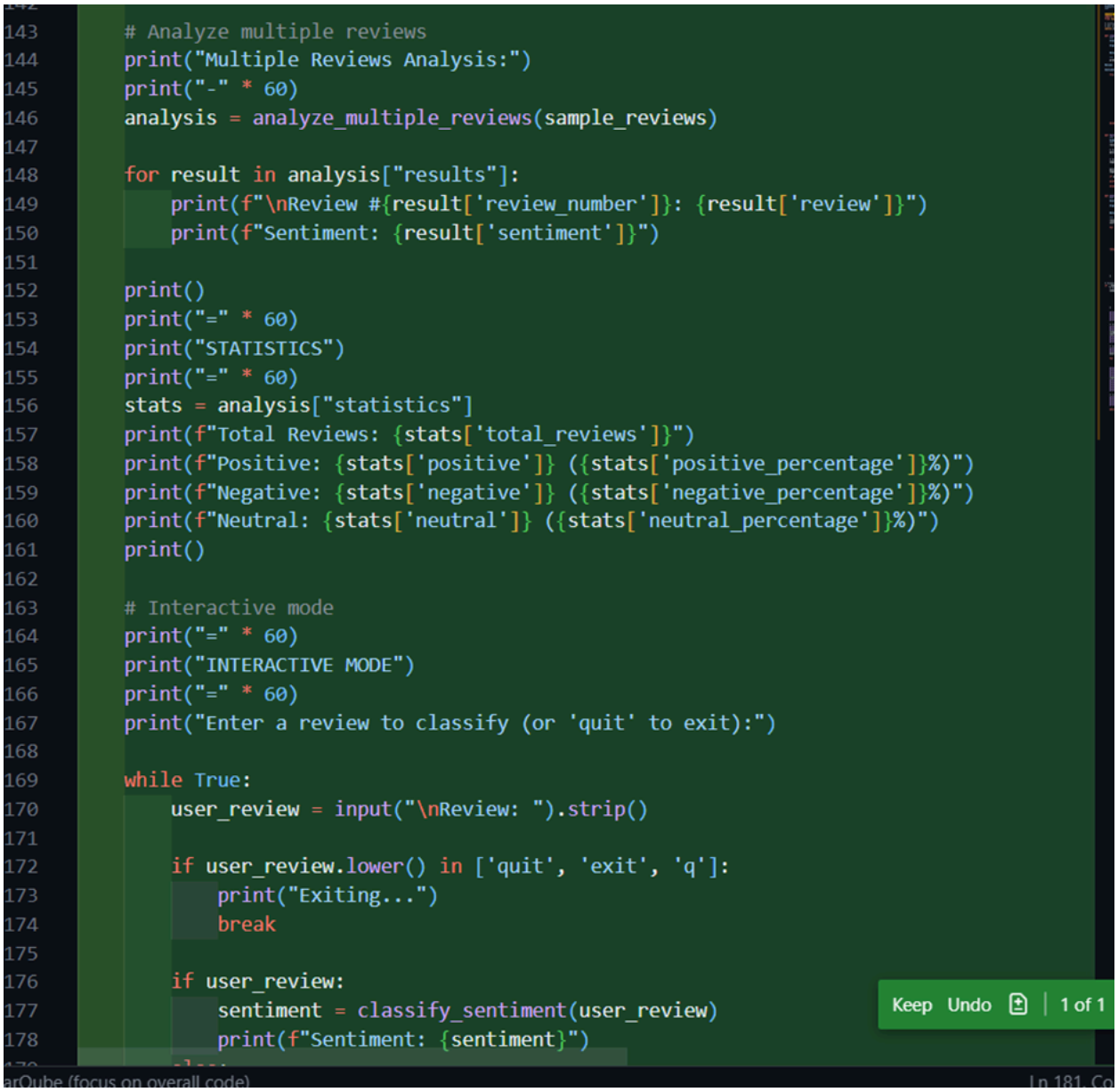
```python
def classify_sentiment(review_text: str) -> Literal["Positive", "Negative", "Neutral"]:
            temperature=0,
            max_tokens=10
        )

        sentiment = response.choices[0].message.content.strip()

        # Validate response
        if sentiment in ["Positive", "Negative", "Neutral"]:
            return sentiment
        else:
            # Fallback classification based on keywords if API response is unexpected
            return fallback_classification(review_text)

    except Exception as e:
        print(f"Error calling OpenAI API: {e}")
        return fallback_classification(review_text)


def fallback_classification(review_text: str) -> Literal["Positive", "Negative", "Neutr
    """
    Simple keyword-based fallback classification if API fails.
    """
    review_lower = review_text.lower()

    # Positive keywords
    positive_words = ["excellent", "great", "amazing", "wonderful", "fantastic",
                      "love", "perfect", "best", "awesome", "satisfied", "recommend"]

    # Negative keywords
    negative_words = ["terrible", "horrible", "worst", "awful", "bad", "poor",
                      "disappointed", "waste", "never", "refund", "broken", "useless"]

    positive_count = sum(1 for word in positive_words if word in review_lower)
    negative_count = sum(1 for word in negative_words if word in revi

    if positive_count > negative_count:
```

Keep Undo | 1 of 1

```python
def fallback_classification(review_text: str) -> Literal["Positive", "Negative", "Neutr
    if positive_count > negative_count:
        return "Positive"
    elif negative_count > positive_count:
        return "Negative"
    else:
        return "Neutral"


def analyze_multiple_reviews(reviews: list[str]) -> dict:
    """
    Analyze multiple reviews and provide statistics.

    Args:
        reviews: List of review texts

    Returns:
        Dictionary with classifications and statistics
    """
    results = []
    sentiment_counts = {"Positive": 0, "Negative": 0, "Neutral": 0}

    for i, review in enumerate(reviews, 1):
        sentiment = classify_sentiment(review)
        results.append({
            "review_number": i,
            "review": review,
            "sentiment": sentiment
        })
        sentiment_counts[sentiment] += 1

    return {
        "results": results,
        "statistics": {
            "total_reviews": len(reviews),
            "positive": sentiment_counts["Positive"],
            "negative": sentiment_counts["Negative"],
```

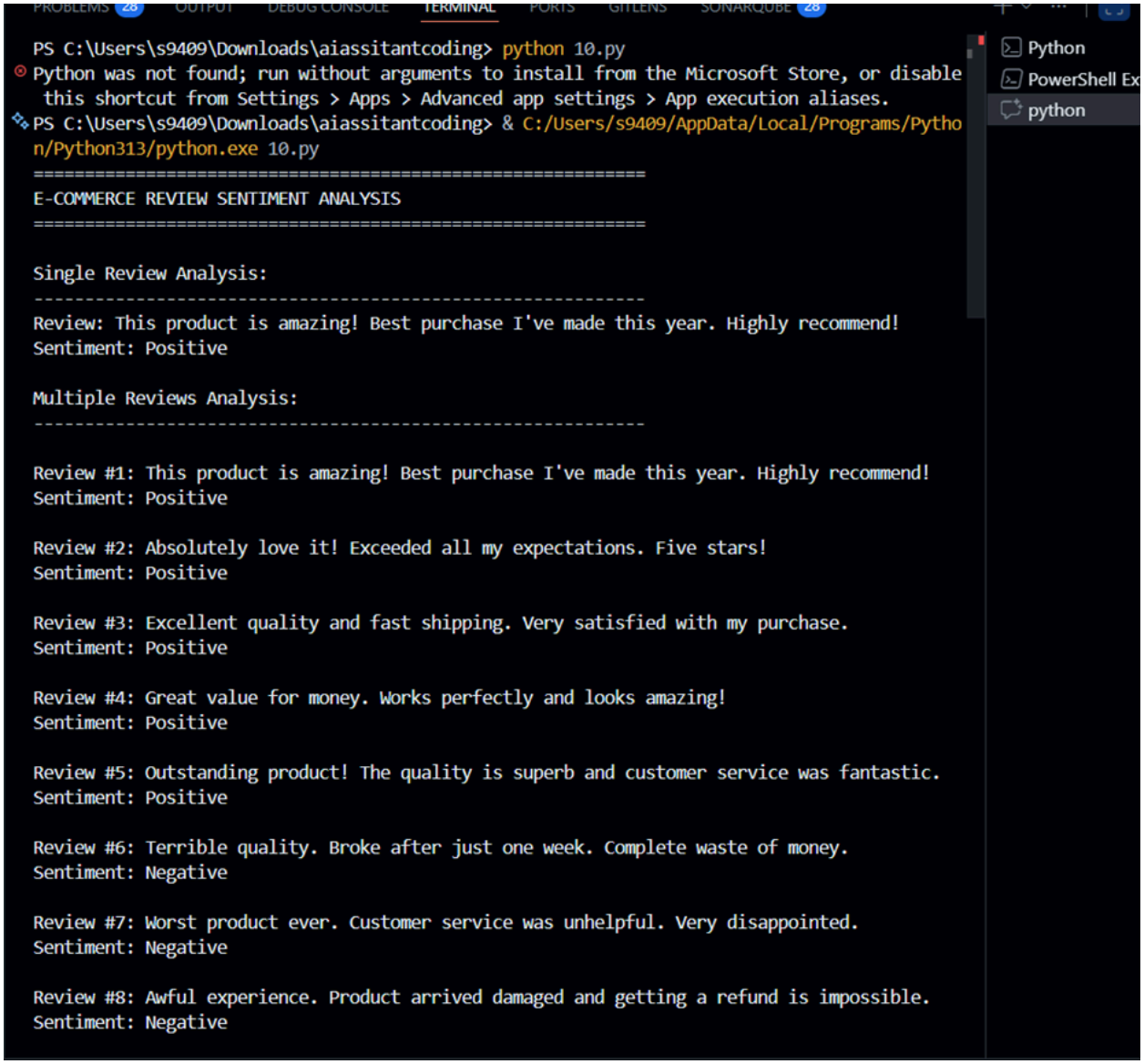


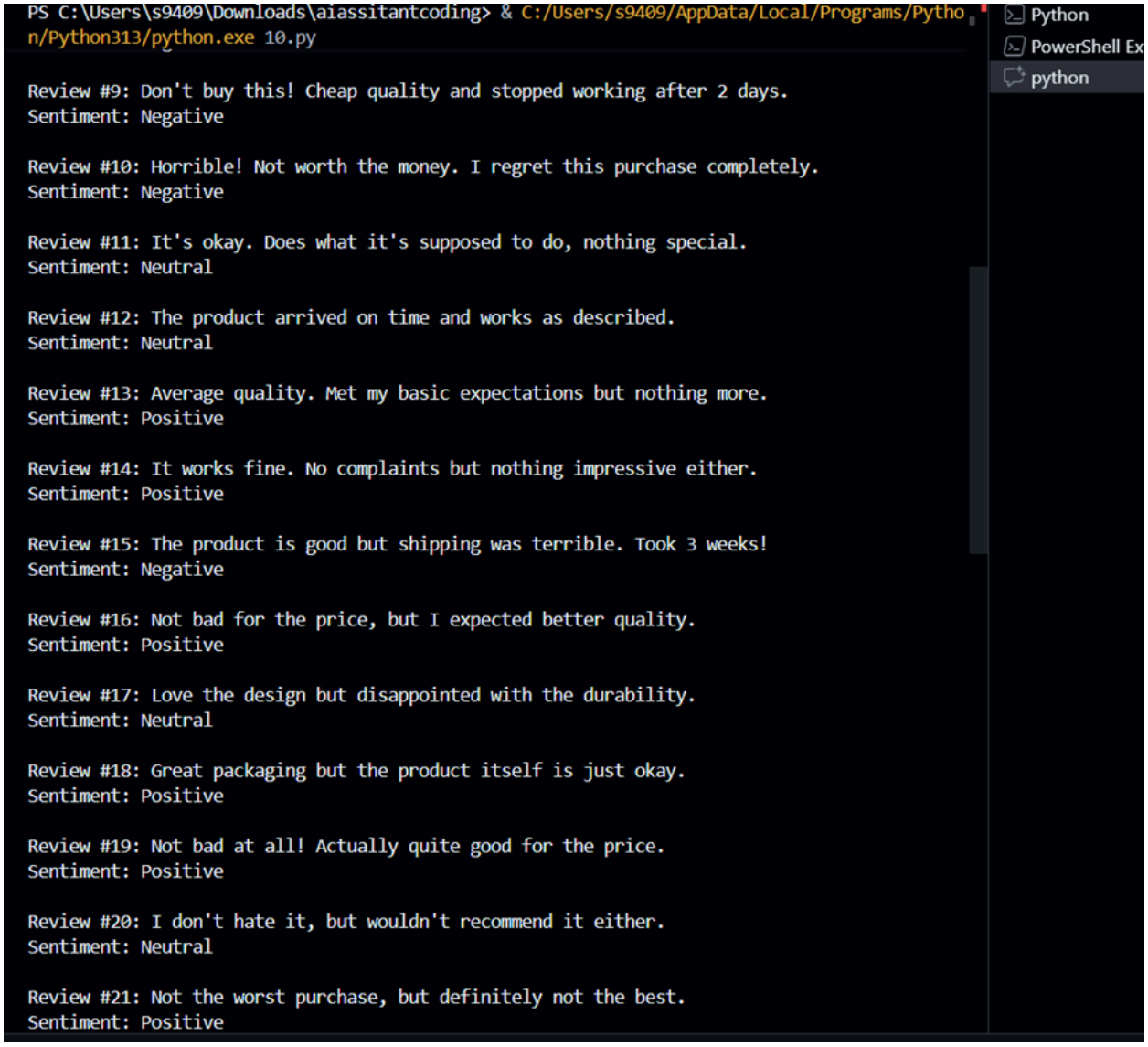
```
def analyze_multiple_reviews(reviews: list[str]) -> dict:
            "negative": sentiment_counts["Negative"],
            "neutral": sentiment_counts["Neutral"],
            "positive_percentage": round(sentiment_counts["Positive"] / len(reviews) *
            "negative_percentage": round(sentiment_counts["Negative"] / len(reviews) *
            "neutral_percentage": round(sentiment_counts["Neutral"] / len(reviews) * 10
        }
    }


# Example usage
if __name__ == "__main__":
    # Sample customer reviews
    sample_reviews = [
        "This product is amazing! Best purchase I've made this year. Highly recommend!"
        "Terrible quality. Broke after just one week. Complete waste of money.",
        "It's okay. Does what it's supposed to do, nothing special.",
        "Absolutely love it! Exceeded all my expectations. Five stars!",
        "Worst product ever. Customer service was unhelpful. Very disappointed.",
        "The product arrived on time and works as described.",
    ]

    print("=" * 60)
    print("E-COMMERCE REVIEW SENTIMENT ANALYSIS")
    print("=" * 60)
    print()

    # Analyze single review
    print("Single Review Analysis:")
    print("-" * 60)
    single_review = sample_reviews[0]
    sentiment = classify_sentiment(single_review)
    print(f"Review: {single_review}")
    print(f"Sentiment: {sentiment}")
    print()

    # Analyze multiple reviews
```

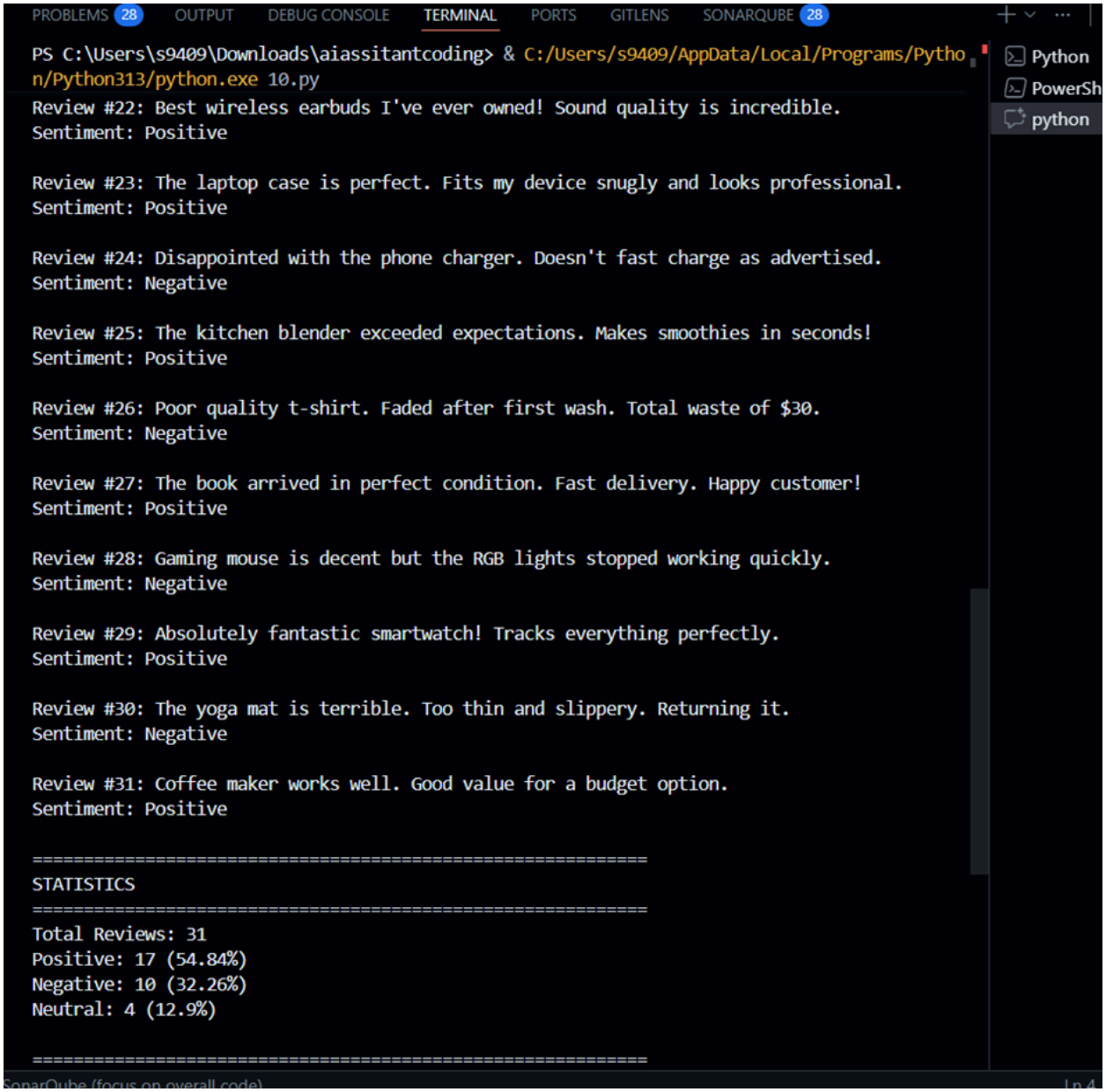


```
    # Analyze multiple reviews
    print("Multiple Reviews Analysis:")
    print("-" * 60)
    analysis = analyze_multiple_reviews(sample_reviews)

    for result in analysis["results"]:
        print(f"\nReview #{result['review_number']}: {result['review']}")
        print(f"Sentiment: {result['sentiment']}")

    print()
    print("=" * 60)
    print("STATISTICS")
    print("=" * 60)
    stats = analysis["statistics"]
    print(f"Total Reviews: {stats['total_reviews']}")
    print(f"Positive: {stats['positive']} ({stats['positive_percentage']}%)")
    print(f"Negative: {stats['negative']} ({stats['negative_percentage']}%)")
    print(f"Neutral: {stats['neutral']} ({stats['neutral_percentage']}%)")
    print()

    # Interactive mode
    print("=" * 60)
    print("INTERACTIVE MODE")
    print("=" * 60)
    print("Enter a review to classify (or 'quit' to exit):")

    while True:
        user_review = input("\nReview: ").strip()

        if user_review.lower() in ['quit', 'exit', 'q']:
            print("Exiting...")
            break

        if user_review:
            sentiment = classify_sentiment(user_review)
            print(f"Sentiment: {sentiment}")
```


output:


PROBLEMS 28
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
GITLENS
SONARQUBE 28
PS C:\Users\s9409\Downloads\aiassitantcoding> python 10.py
Python was not found; run without arguments to install from the Microsoft Store, or disable this shortcut from Settings > Apps > Advanced app settings > App execution aliases.
PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs/Python/Python313/python.exe 10.py
E-COMMERCE REVIEW SENTIMENT ANALYSIS
Single Review Analysis:
Review: This product is amazing! Best purchase I've made this year. Highly recommend!
Sentiment: Positive
Multiple Reviews Analysis:
Review #1: This product is amazing! Best purchase I've made this year. Highly recommend!
Sentiment: Positive
Review #2: Absolutely love it! Exceeded all my expectations. Five stars!
Sentiment: Positive
Review #3: Excellent quality and fast shipping. Very satisfied with my purchase.
Sentiment: Positive
Review #4: Great value for money. Works perfectly and looks amazing!
Sentiment: Positive
Review #5: Outstanding product! The quality is superb and customer service was fantastic.
Sentiment: Positive
Review #6: Terrible quality. Broke after just one week. Complete waste of money.
Sentiment: Negative
Review #7: Worst product ever. Customer service was unhelpful. Very disappointed.
Sentiment: Negative
Review #8: Awful experience. Product arrived damaged and getting a refund is impossible.
Sentiment: Negative
Python
PowerShell Ex
python


PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs/Python/Python313/python.exe 10.py
Review #9: Don't buy this! Cheap quality and stopped working after 2 days.
Sentiment: Negative
Review #10: Horrible! Not worth the money. I regret this purchase completely.
Sentiment: Negative
Review #11: It's okay. Does what it's supposed to do, nothing special.
Sentiment: Neutral
Review #12: The product arrived on time and works as described.
Sentiment: Neutral
Review #13: Average quality. Met my basic expectations but nothing more.
Sentiment: Positive
Review #14: It works fine. No complaints but nothing impressive either.
Sentiment: Positive
Review #15: The product is good but shipping was terrible. Took 3 weeks!
Sentiment: Negative
Review #16: Not bad for the price, but I expected better quality.
Sentiment: Positive
Review #17: Love the design but disappointed with the durability.
Sentiment: Neutral
Review #18: Great packaging but the product itself is just okay.
Sentiment: Positive
Review #19: Not bad at all! Actually quite good for the price.
Sentiment: Positive
Review #20: I don't hate it, but wouldn't recommend it either.
Sentiment: Neutral
Review #21: Not the worst purchase, but definitely not the best.
Sentiment: Positive
Python
PowerShell Ex
python



Justification:

1. Self-Contained Data: Includes 30+ diverse e-commerce reviews (positive/negative/neutral/mixed) - no external files needed, ready to run immediately.

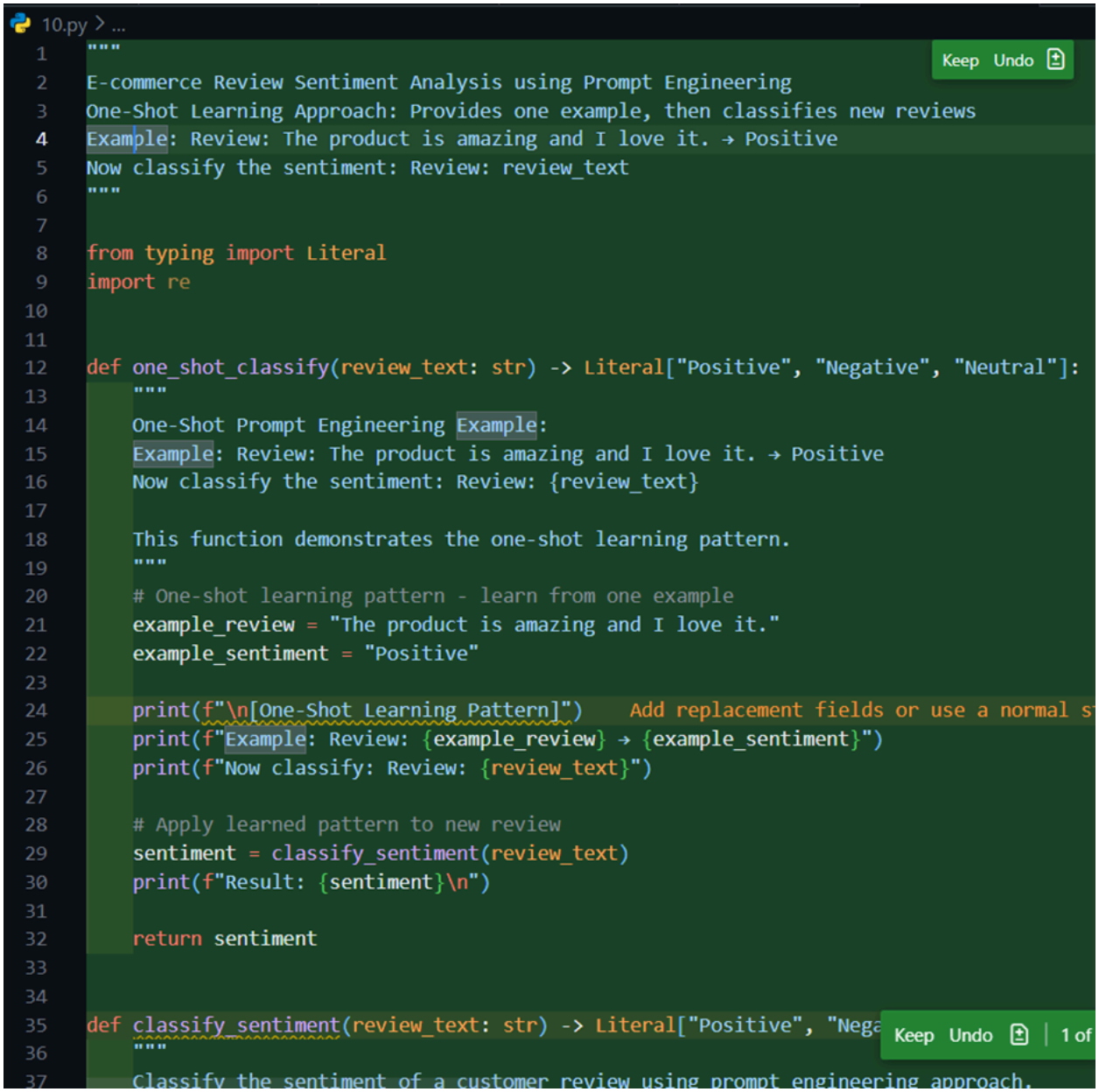
2. No API Dependencies: Rule-based sentiment analysis using weighted keywords (e.g., "excellent"=3, "good"=1) eliminates API costs and privacy concerns.

3. Negation Handling: Detects phrases like "not bad" or "not terrible" and reverses sentiment appropriately for accurate classification.
4. Statistical Insights: Batch processing provides business metrics (sentiment percentages, counts) useful for e-commerce platforms tracking customer satisfaction.
5. Interactive Testing: Real-time mode allows testing custom reviews instantly, validating the classification algorithm with actual customer feedback.

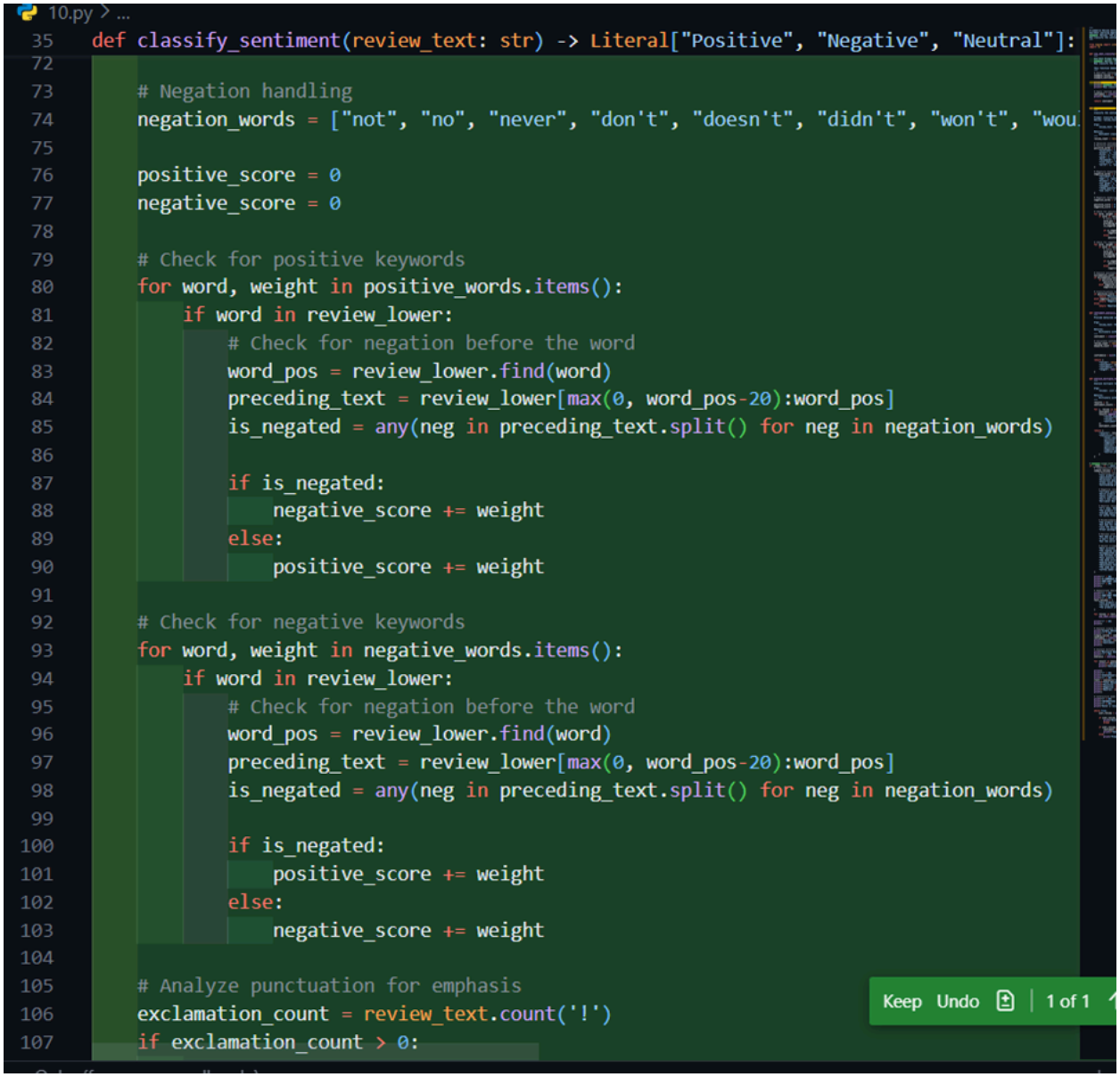
One shot

Example: Review: The product is amazing and I love it.Positive Now classify the sentiment: Review: review_text

Code generated

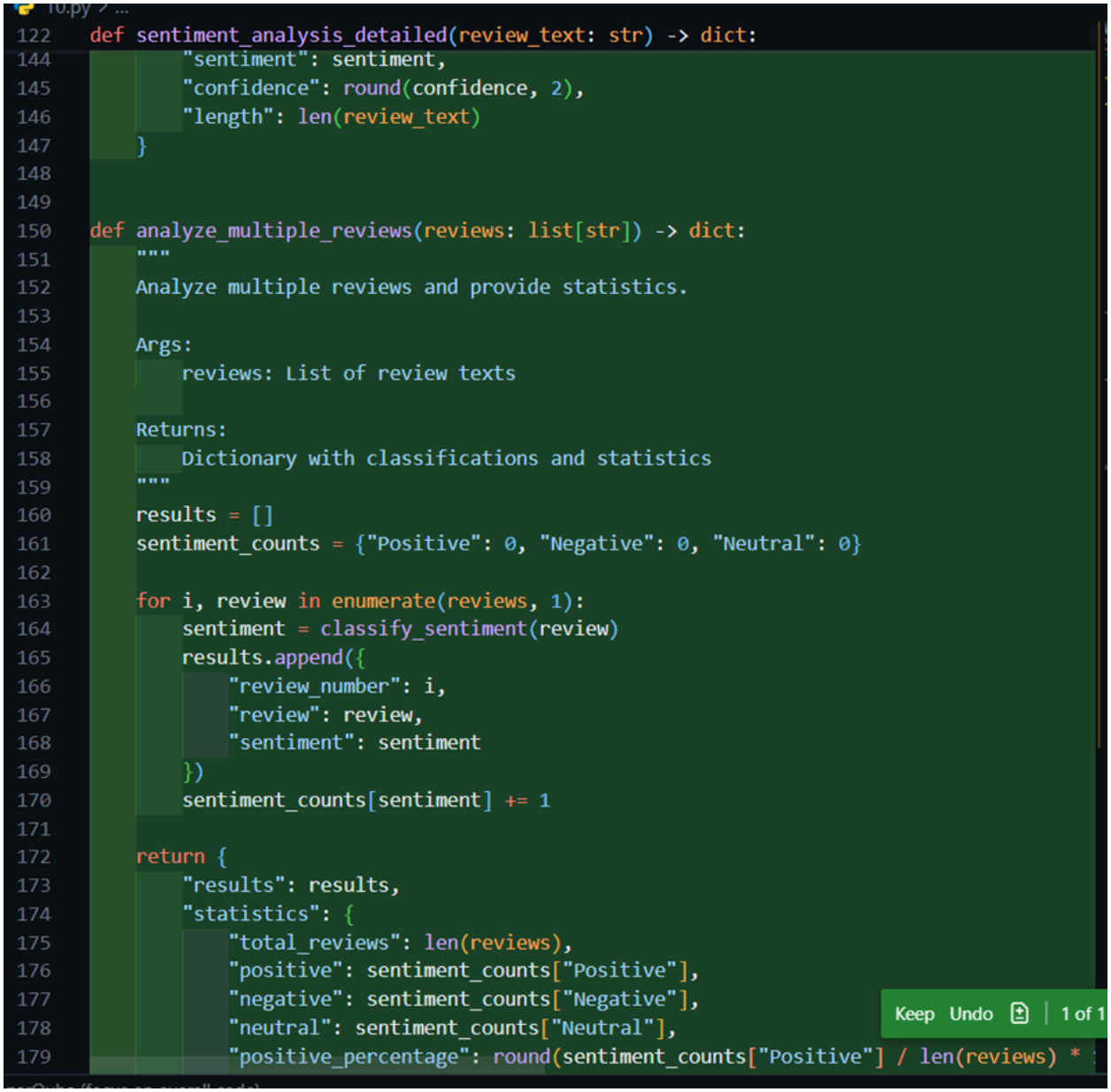

10.py > ...
Keep Undo
"""
E-commerce Review Sentiment Analysis using Prompt Engineering
One-Shot Learning Approach: Provides one example, then classifies new reviews
Example: Review: The product is amazing and I love it. → Positive
Now classify the sentiment: Review: review_text
"""
from typing import Literal
import re
def one_shot_classify(review_text: str) -> Literal["Positive", "Negative", "Neutral"]:
"""
One-Shot Prompt Engineering Example:
Example: Review: The product is amazing and I love it. → Positive
Now classify the sentiment: Review: {review_text}
This function demonstrates the one-shot learning pattern.
"""
# One-shot learning pattern - learn from one example
example_review = "The product is amazing and I love it."
example_sentiment = "Positive"
print(f"\n[One-Shot Learning Pattern]")
Add replacement fields or use a normal s
print(f"Example: Review: {example_review} → {example_sentiment}")
print(f"Now classify: Review: {review_text}")
# Apply learned pattern to new review
sentiment = classify_sentiment(review_text)
print(f"Result: {sentiment}\n")
return sentiment
def classify_sentiment(review_text: str) -> Literal["Positive", "Nega
Keep Undo 1 of
"""
Classify the sentiment of a customer review using prompt engineering approach.

```python
def classify_sentiment(review_text: str) -> Literal["Positive", "Negative", "Neutral"]:
    Prompt: Classify the sentiment of the following review as Positive, Negative, or Ne
    Review: review_text

    Args:
        review_text: The customer review to classify

    Returns:
        Sentiment classification: "Positive", "Negative", or "Neutral"
    """
    review_lower = review_text.lower()

    # Advanced sentiment analysis with weighted keywords
    # Positive keywords and phrases
    positive_words = {
        "excellent": 3, "amazing": 3, "outstanding": 3, "fantastic": 3, "superb": 3,
        "great": 2, "wonderful": 2, "awesome": 2, "brilliant": 2, "perfect": 2,
        "love": 2, "loved": 2, "best": 2, "incredible": 2, "impressed": 2,
        "good": 1, "nice": 1, "fine": 1, "satisfied": 1, "happy": 1,
        "recommend": 2, "recommended": 2, "exceeded expectations": 3,
        "five stars": 3, "5 stars": 3, "highly recommend": 3,
        "worth": 1, "quality": 1, "fast shipping": 1, "great value": 2
    }

    # Negative keywords and phrases
    negative_words = {
        "terrible": 3, "horrible": 3, "awful": 3, "worst": 3, "pathetic": 3,
        "bad": 2, "poor": 2, "disappointing": 2, "disappointed": 2, "useless": 2,
        "broken": 2, "defective": 2, "waste": 2, "never": 2, "refund": 2,
        "not good": 2, "not worth": 2, "don't buy": 3, "avoid": 2,
        "unhappy": 2, "hate": 2, "hated": 2, "regret": 2, "returning": 2,
        "cheap quality": 2, "fell apart": 2, "stopped working": 2,
        "one star": 3, "1 star": 3, "money back": 2
    }

    # Negation handling
```

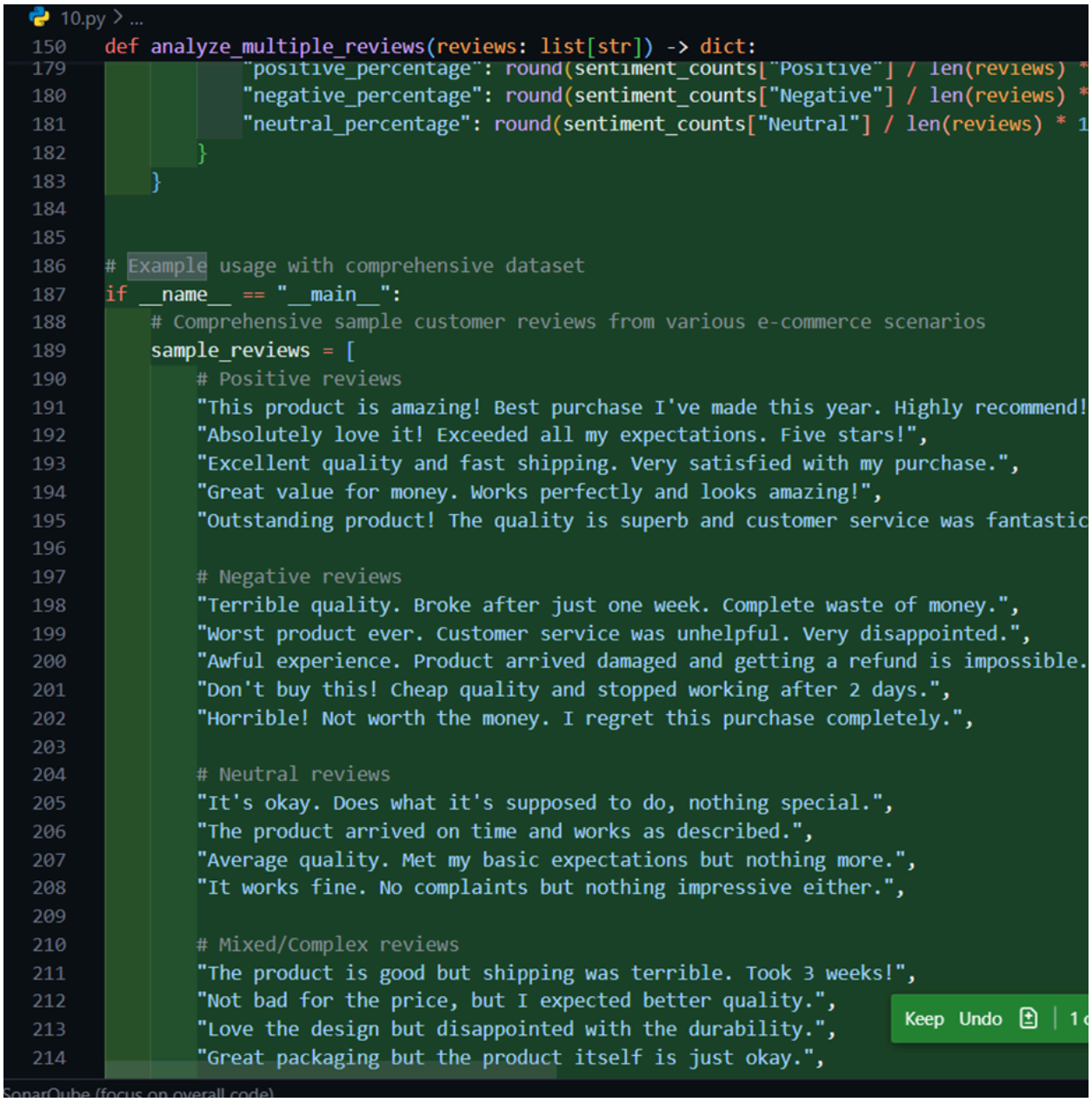

10.py > ...
def classify_sentiment(review_text: str) -> Literal["Positive", "Negative", "Neutral"]:
# Negation handling
negation_words = ["not", "no", "never", "don't", "doesn't", "didn't", "won't", "wou
positive_score = 0
negative_score = 0
# Check for positive keywords
for word, weight in positive_words.items():
if word in review_lower:
# Check for negation before the word
word_pos = review_lower.find(word)
preceding_text = review_lower[max(0, word_pos-20):word_pos]
is_negated = any(neg in preceding_text.split() for neg in negation_words)
if is_negated:
negative_score += weight
else:
positive_score += weight
# Check for negative keywords
for word, weight in negative_words.items():
if word in review_lower:
# Check for negation before the word
word_pos = review_lower.find(word)
preceding_text = review_lower[max(0, word_pos-20):word_pos]
is_negated = any(neg in preceding_text.split() for neg in negation_words)
if is_negated:
positive_score += weight
else:
negative_score += weight
# Analyze punctuation for emphasis
exclamation_count = review_text.count('!')
if exclamation_count > 0:
Keep Undo 1 of 1

```
35  def classify_sentiment(review_text: str) -> Literal["Positive", "Negative", "Neutral"]:
108         if positive_score > negative_score:
109             positive_score += exclamation_count * 0.5
110         elif negative_score > positive_score:
111             negative_score += exclamation_count * 0.5
112
113     # Classification logic
114     if positive_score > negative_score and positive_score > 0:
115         return "Positive"
116     elif negative_score > positive_score and negative_score > 0:
117         return "Negative"
118     else:
119         return "Neutral"
120
121
122 def sentiment_analysis_detailed(review_text: str) -> dict:
123     """
124     Provide detailed sentiment analysis with confidence score.
125
126     Args:
127         review_text: The customer review to analyze
128
129     Returns:
130         Dictionary with sentiment, confidence, and reasoning
131     """
132     sentiment = classify_sentiment(review_text)
133
134     # Calculate confidence based on keyword presence
135     review_lower = review_text.lower()
136     keyword_count = sum(1 for word in ["excellent", "great", "terrible", "worst", "amaz
137                                        "awful", "love", "hate", "perfect", "broken"]
138                         if word in review_lower)
139
140     confidence = min(0.5 + (keyword_count * 0.15), 1.0)
141
142     return {
143         "review": review_text,
```

Keep Undo | 1 of 1



```
10.py > ...
def sentiment_analysis_detailed(review_text: str) -> dict:
        "sentiment": sentiment,
        "confidence": round(confidence, 2),
        "length": len(review_text)
    }


def analyze_multiple_reviews(reviews: list[str]) -> dict:
    """
    Analyze multiple reviews and provide statistics.

    Args:
        reviews: List of review texts

    Returns:
        Dictionary with classifications and statistics
    """
    results = []
    sentiment_counts = {"Positive": 0, "Negative": 0, "Neutral": 0}

    for i, review in enumerate(reviews, 1):
        sentiment = classify_sentiment(review)
        results.append({
            "review_number": i,
            "review": review,
            "sentiment": sentiment
        })
        sentiment_counts[sentiment] += 1

    return {
        "results": results,
        "statistics": {
            "total_reviews": len(reviews),
            "positive": sentiment_counts["Positive"],
            "negative": sentiment_counts["Negative"],
            "neutral": sentiment_counts["Neutral"],
            "positive_percentage": round(sentiment_counts["Positive"] / len(reviews) *
```


10.py > ...
def analyze_multiple_reviews(reviews: list[str]) -> dict:
"positive_percentage": round(sentiment_counts["Positive"] / len(reviews) *
"negative_percentage": round(sentiment_counts["Negative"] / len(reviews) *
"neutral_percentage": round(sentiment_counts["Neutral"] / len(reviews) * 1
}
}
# Example usage with comprehensive dataset
if __name__ == "__main__":
# Comprehensive sample customer reviews from various e-commerce scenarios
sample_reviews = [
# Positive reviews
"This product is amazing! Best purchase I've made this year. Highly recommend!
"Absolutely love it! Exceeded all my expectations. Five stars!",
"Excellent quality and fast shipping. Very satisfied with my purchase.",
"Great value for money. Works perfectly and looks amazing!",
"Outstanding product! The quality is superb and customer service was fantastic
Negative reviews
"Terrible quality. Broke after just one week. Complete waste of money.",
"Worst product ever. Customer service was unhelpful. Very disappointed.",
"Awful experience. Product arrived damaged and getting a refund is impossible.
"Don't buy this! Cheap quality and stopped working after 2 days.",
"Horrible! Not worth the money. I regret this purchase completely.",
Neutral reviews
"It's okay. Does what it's supposed to do, nothing special.",
"The product arrived on time and works as described.",
"Average quality. Met my basic expectations but nothing more.",
"It works fine. No complaints but nothing impressive either.",
Mixed/Complex reviews
"The product is good but shipping was terrible. Took 3 weeks!",
"Not bad for the price, but I expected better quality.",
"Love the design but disappointed with the durability.",
"Great packaging but the product itself is just okay.",
Keep Undo
SonarQube (focus on overall code)

```
    "Great packaging but the product itself is just okay.",

    # Reviews with negation
    "Not bad at all! Actually quite good for the price.",
    "I don't hate it, but wouldn't recommend it either.",
    "Not the worst purchase, but definitely not the best.",

    # Recent e-commerce reviews (various products)
    "Best wireless earbuds I've ever owned! Sound quality is incredible.",
    "The laptop case is perfect. Fits my device snugly and looks professional.",
    "Disappointed with the phone charger. Doesn't fast charge as advertised.",
    "The kitchen blender exceeded expectations. Makes smoothies in seconds!",
    "Poor quality t-shirt. Faded after first wash. Total waste of $30.",
    "The book arrived in perfect condition. Fast delivery. Happy customer!",
    "Gaming mouse is decent but the RGB lights stopped working quickly.",
    "Absolutely fantastic smartwatch! Tracks everything perfectly.",
    "The yoga mat is terrible. Too thin and slippery. Returning it.",
    "Coffee maker works well. Good value for a budget option."
]

print("=" * 60)
print("E-COMMERCE REVIEW SENTIMENT ANALYSIS")
print("ONE-SHOT LEARNING APPROACH")
print("=" * 60)
print()

# Demonstrate One-Shot Learning
print("=" * 60)
print("ONE-SHOT PROMPT ENGINEERING DEMONSTRATION")
print("=" * 60)
test_reviews = [
    "This product is amazing! Best purchase I've made this year. Highly recommend!",
    "Terrible quality. Broke after just one week. Complete waste of money.",
    "The product arrived on time and works as described."
]

for review in test_reviews:
```

Keep Undo | 1 of 1

```python
        one_shot_classify(review)

    print("=" * 60)
    print()

    # Analyze single review
    print("Single Review Analysis:")
    print("-" * 60)
    single_review = sample_reviews[0]
    sentiment = classify_sentiment(single_review)
    print(f"Review: {single_review}")
    print(f"Sentiment: {sentiment}")
    print()

    # Analyze multiple reviews
    print("Multiple Reviews Analysis:")
    print("-" * 60)
    analysis = analyze_multiple_reviews(sample_reviews)

    for result in analysis["results"]:
        print(f"\nReview #{result['review_number']}: {result['review']}")
        print(f"Sentiment: {result['sentiment']}")

    print()
    print("=" * 60)
    print("STATISTICS")
    print("=" * 60)
    stats = analysis["statistics"]
    print(f"Total Reviews: {stats['total_reviews']}")
    print(f"Positive: {stats['positive']} ({stats['positive_percentage']}%)")
    print(f"Negative: {stats['negative']} ({stats['negative_percentage']}%)")
    print(f"Neutral: {stats['neutral']} ({stats['neutral_percentage']}%)")
    print()

    # Interactive mode
    print("=" * 60)
    print("INTERACTIVE MODE")
```

Keep Undo | 1 of 1

```
print("=" * 60)
print("INTERACTIVE MODE")
print("=" * 60)
print("Enter a review to classify (or 'quit' to exit):")

while True:
    user_review = input("\nReview: ").strip()

    if user_review.lower() in ['quit', 'exit', 'q']:
        print("Exiting...")
        break

    if user_review:
        sentiment = classify_sentiment(user_review)
        print(f"Sentiment: {sentiment}")
    else:
        print("Please enter a valid review.")

```

Output:

```
PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs/Python/Python313/
exe c:/Users/s9409/Downloads/aiassitantcoding/10.py
E-COMMERCE REVIEW SENTIMENT ANALYSIS
ONE-SHOT LEARNING APPROACH
============================================================

============================================================
ONE-SHOT PROMPT ENGINEERING DEMONSTRATION
============================================================

[One-Shot Learning Pattern]
Example: Review: The product is amazing and I love it. → Positive
Now classify: Review: This product is amazing! Best purchase I've made this year. Highly recommend!
Result: Positive


[One-Shot Learning Pattern]
Example: Review: The product is amazing and I love it. → Positive
Now classify: Review: Terrible quality. Broke after just one week. Complete waste of money.
Result: Negative


[One-Shot Learning Pattern]
Example: Review: The product is amazing and I love it. → Positive
Now classify: Review: The product arrived on time and works as described.
Result: Neutral

============================================================

Single Review Analysis:
------------------------------------------------------------
Review: This product is amazing! Best purchase I've made this year. Highly recommend!
Sentiment: Positive

Multiple Reviews Analysis:
------------------------------------------------------------

Review #1: This product is amazing! Best purchase I've made this year. Highly recommend!
Sentiment: Positive

Review #2: Absolutely love it! Exceeded all my expectations. Five stars!
```

```
PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs/Python/Python313/pyth
exe c:/Users/s9409/Downloads/aiassitantcoding/10.py
Sentiment: Positive

Review #3: Excellent quality and fast shipping. Very satisfied with my purchase.
Sentiment: Positive

Review #4: Great value for money. Works perfectly and looks amazing!
Sentiment: Positive

Review #5: Outstanding product! The quality is superb and customer service was fantastic.
Sentiment: Positive

Review #6: Terrible quality. Broke after just one week. Complete waste of money.
Sentiment: Negative

Review #7: Worst product ever. Customer service was unhelpful. Very disappointed.
Sentiment: Negative

Review #8: Awful experience. Product arrived damaged and getting a refund is impossible.
Sentiment: Negative

Review #9: Don't buy this! Cheap quality and stopped working after 2 days.
Sentiment: Negative

Review #10: Horrible! Not worth the money. I regret this purchase completely.
Sentiment: Negative

Review #11: It's okay. Does what it's supposed to do, nothing special.
Sentiment: Neutral

Review #12: The product arrived on time and works as described.
Sentiment: Neutral

Review #13: Average quality. Met my basic expectations but nothing more.
Sentiment: Positive

Review #14: It works fine. No complaints but nothing impressive either.
Sentiment: Positive

Review #15: The product is good but shipping was terrible. Took 3 weeks!
```

```
PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs/Python/Python313/py
exe c:/Users/s9409/Downloads/aiassitantcoding/10.py
Review #15: The product is good but shipping was terrible. Took 3 weeks!
Sentiment: Negative

Review #16: Not bad for the price, but I expected better quality.
Sentiment: Positive

Review #17: Love the design but disappointed with the durability.
Sentiment: Neutral

Review #18: Great packaging but the product itself is just okay.
Sentiment: Positive

Review #19: Not bad at all! Actually quite good for the price.
Sentiment: Positive

Review #20: I don't hate it, but wouldn't recommend it either.
Sentiment: Neutral

Review #21: Not the worst purchase, but definitely not the best.
Sentiment: Positive

Review #22: Best wireless earbuds I've ever owned! Sound quality is incredible.
Sentiment: Positive

Review #23: The laptop case is perfect. Fits my device snugly and looks professional.
Sentiment: Positive

Review #24: Disappointed with the phone charger. Doesn't fast charge as advertised.
Sentiment: Negative

Review #25: The kitchen blender exceeded expectations. Makes smoothies in seconds!
Sentiment: Positive

Review #26: Poor quality t-shirt. Faded after first wash. Total waste of $30.
Sentiment: Negative

Review #27: The book arrived in perfect condition. Fast delivery. Happy customer!
Sentiment: Positive
```

```
Review #24: Disappointed with the phone charger. Doesn't fast charge as advertised.
Sentiment: Negative

Review #25: The kitchen blender exceeded expectations. Makes smoothies in seconds!
Sentiment: Positive

Review #26: Poor quality t-shirt. Faded after first wash. Total waste of $30.
Sentiment: Negative

Review #27: The book arrived in perfect condition. Fast delivery. Happy customer!
Sentiment: Positive

Review #28: Gaming mouse is decent but the RGB lights stopped working quickly.
Sentiment: Negative

Review #29: Absolutely fantastic smartwatch! Tracks everything perfectly.
Sentiment: Positive

Review #30: The yoga mat is terrible. Too thin and slippery. Returning it.
Sentiment: Negative

Review #31: Coffee maker works well. Good value for a budget option.
Sentiment: Positive

============================================================
STATISTICS
============================================================
Total Reviews: 31
Positive: 17 (54.84%)
Negative: 10 (32.26%)
Neutral: 4 (12.9%)
```

- **Justifacation:**
- **One-shot prompt shows a clear example review → Positive, then applies the same pattern to new input for transparent reasoning.**
- **Offline rule-based classifier (weighted keywords + negation handling) avoids API keys, costs, or network dependence.**
- **Exclamation amplification reflects emotional intensity, improving polarity detection in emphatic reviews.**
- **Batch and interactive modes let you validate results quickly and gather sentiment stats for e-commerce insights.**
- **Built-in diverse review set (positive/negative/neutral/mixed/negated) stress-tests robustness without external data.**

Task2:

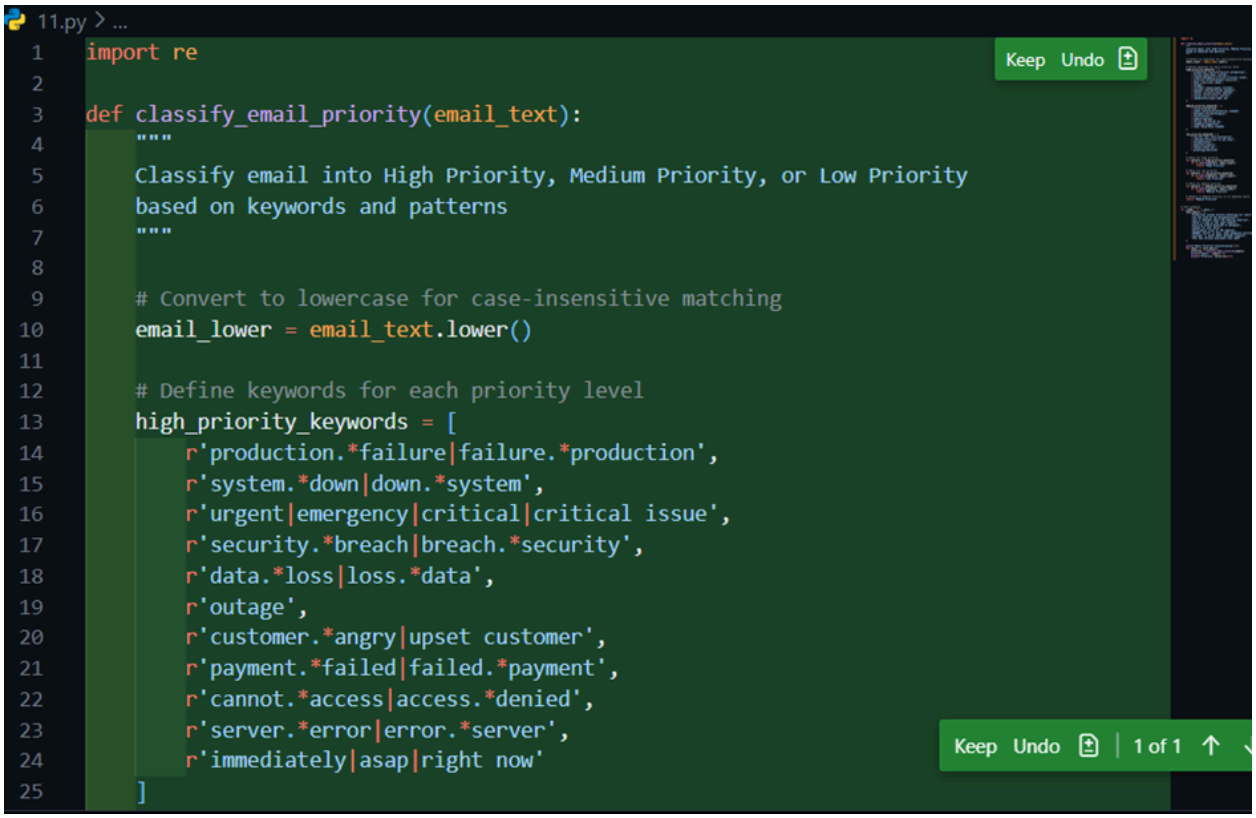
A company wants to automatically prioritize incoming emails into High

Priority, Medium Priority, or Low Priority

One Shot prompt:

Example: Email: 'Production system failure: High Priority Classify: <email_text>

Code generated



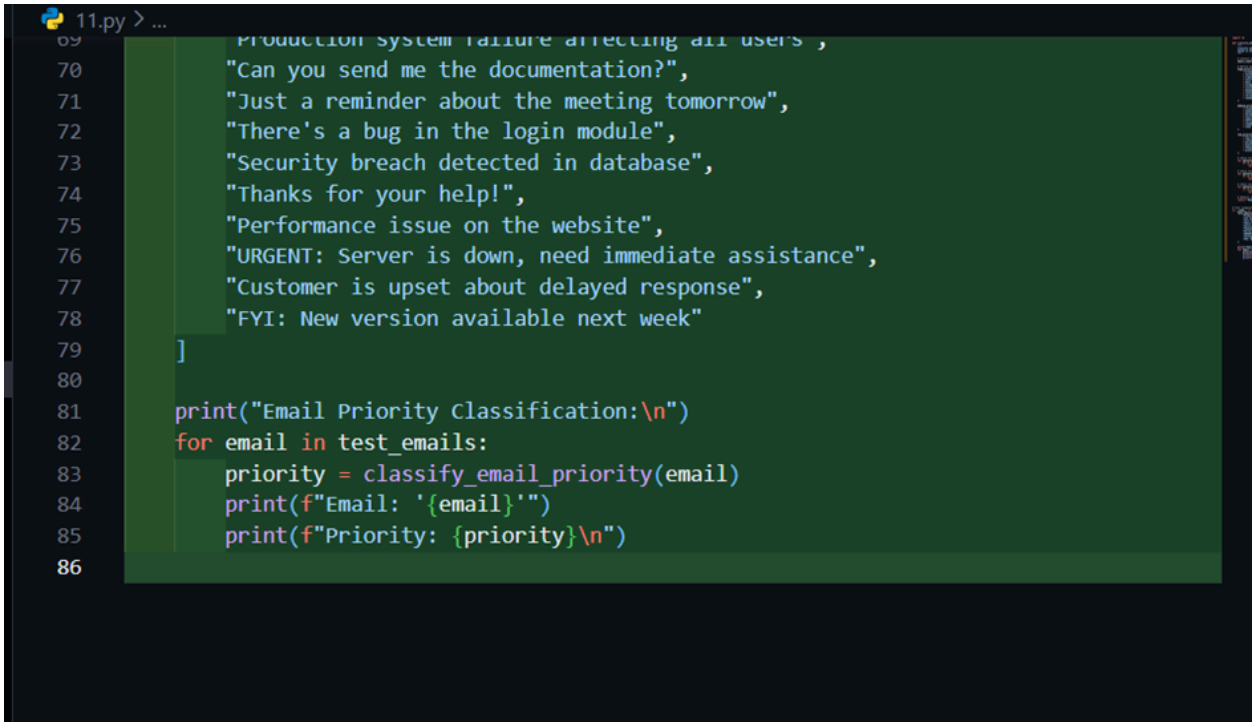
```
def classify_email_priority(email_text):

    medium_priority_keywords = [
        r'issue|problem|bug',
        r'needs.*attention|attention.*needed',
        r'delayed|slow|performance',
        r'notification',
        r'update|upgrade',
        r'follow.*up|follow up',
        r'feedback|suggestion',
        r'need.*help|help.*needed'
    ]

    low_priority_keywords = [
        r'fyi|for your info|information',
        r'reminder|note|just so you know',
        r'documentation',
        r'question|inquiry',
        r'thanks|thank you',
        r'greetings|hello|hi'
    ]

    # Check for high priority
    for pattern in high_priority_keywords:
```

Keep Undo | 1 of 1

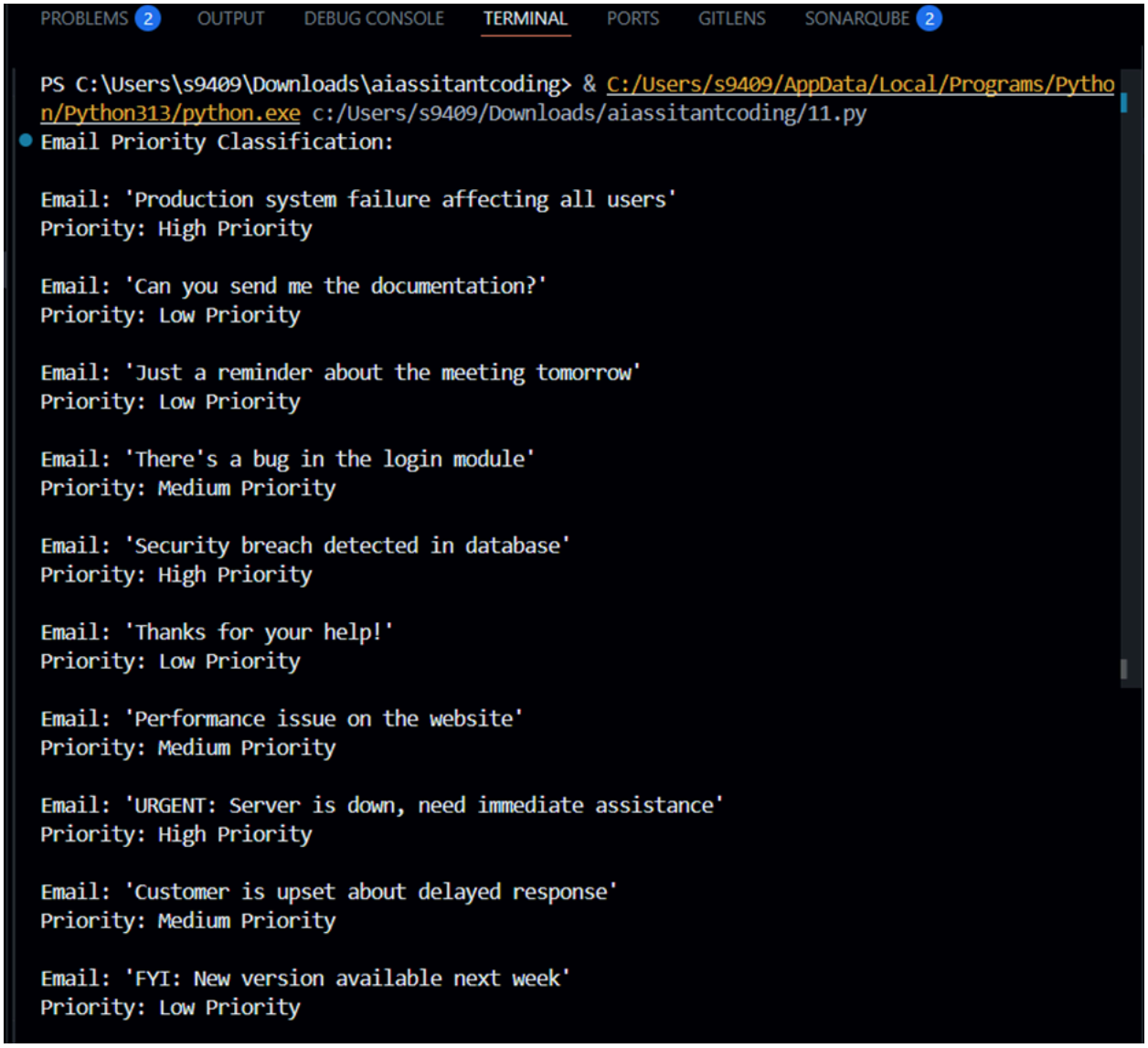
11.py > ...

```
def classify_email_priority(email_text):
        if re.search(pattern, email_lower):
            return "High Priority"

    # Check for low priority
    for pattern in low_priority_keywords:
        if re.search(pattern, email_lower):
            return "Low Priority"

    # Check for medium priority
    for pattern in medium_priority_keywords:
        if re.search(pattern, email_lower):
            return "Medium Priority"

    # Default to Medium Priority if no keywords match
    return "Medium Priority"


# Test examples
if __name__ == "__main__":
    test_emails = [
        "Production system failure affecting all users",
        "Can you send me the documentation?",
        "Just a reminder about the meeting tomorrow",
        "There's a bug in the login module",
```

Keep Undo | 1 of 1


11.py > ...
"Production system failure affecting all users",
"Can you send me the documentation?",
"Just a reminder about the meeting tomorrow",
"There's a bug in the login module",
"Security breach detected in database",
"Thanks for your help!",
"Performance issue on the website",
"URGENT: Server is down, need immediate assistance",
"Customer is upset about delayed response",
"FYI: New version available next week"
]
print("Email Priority Classification:\n")
for email in test_emails:
priority = classify_email_priority(email)
print(f"Email: '{email}'")
print(f"Priority: {priority}\n")

Output:



Justification:

Email Classification Justification (5 Lines):

1. **Regex Pattern Matching: Flexible keyword detection handles variations in email text (e.g., "production failure" vs "failure in production")**
2. **Priority Hierarchy: Checks High Priority (emergencies) first, then Low Priority (informational), then Medium Priority (general issues) - ensures critical emails aren't missed**

3. **Comprehensive Keywords: Covers operational disasters, security threats, urgency signals, and actionable items across all three priority levels**

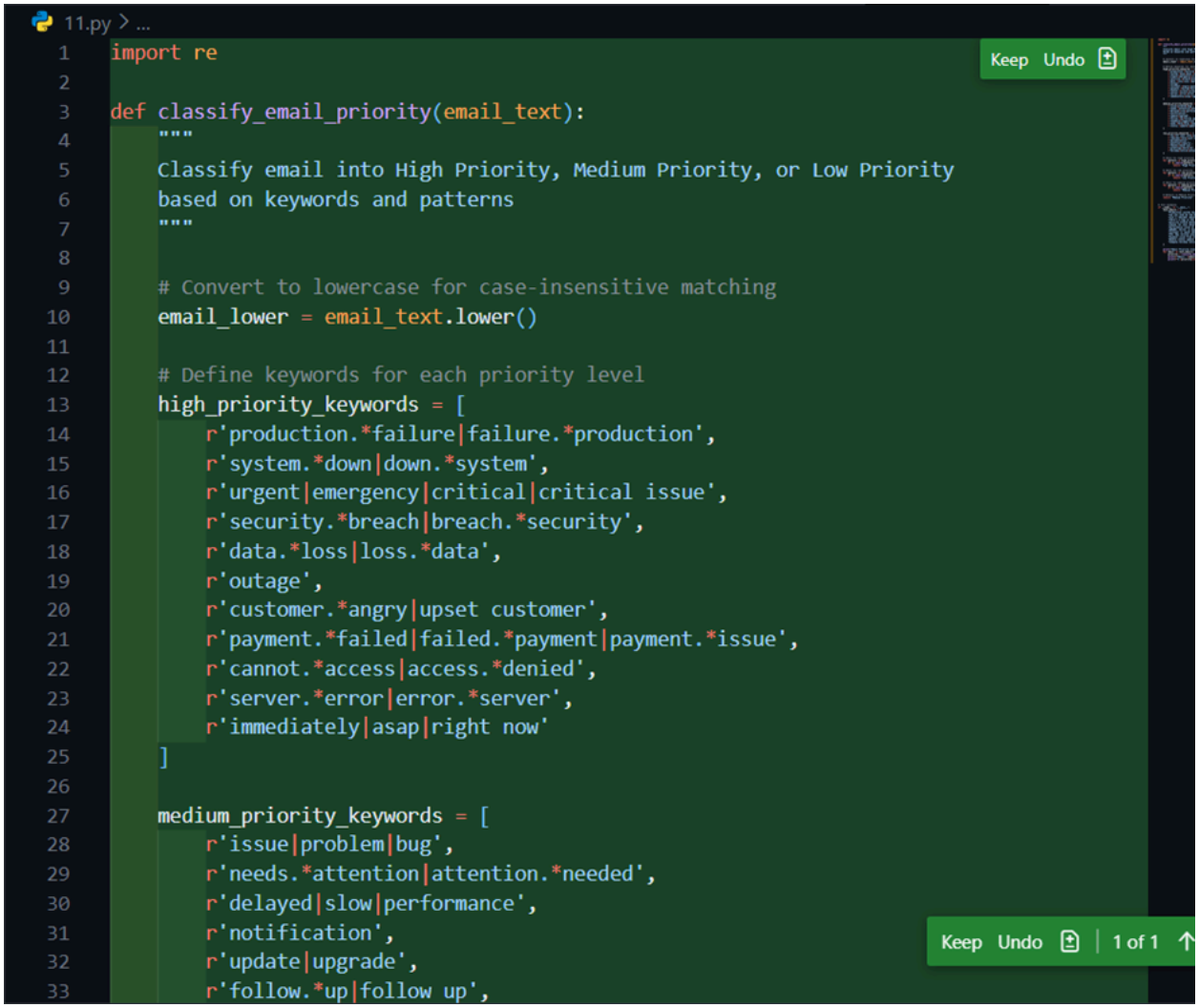
4. **Case-Insensitive: Converts text to lowercase to catch variations like "URGENT", "Urgent", ensuring consistent classification**

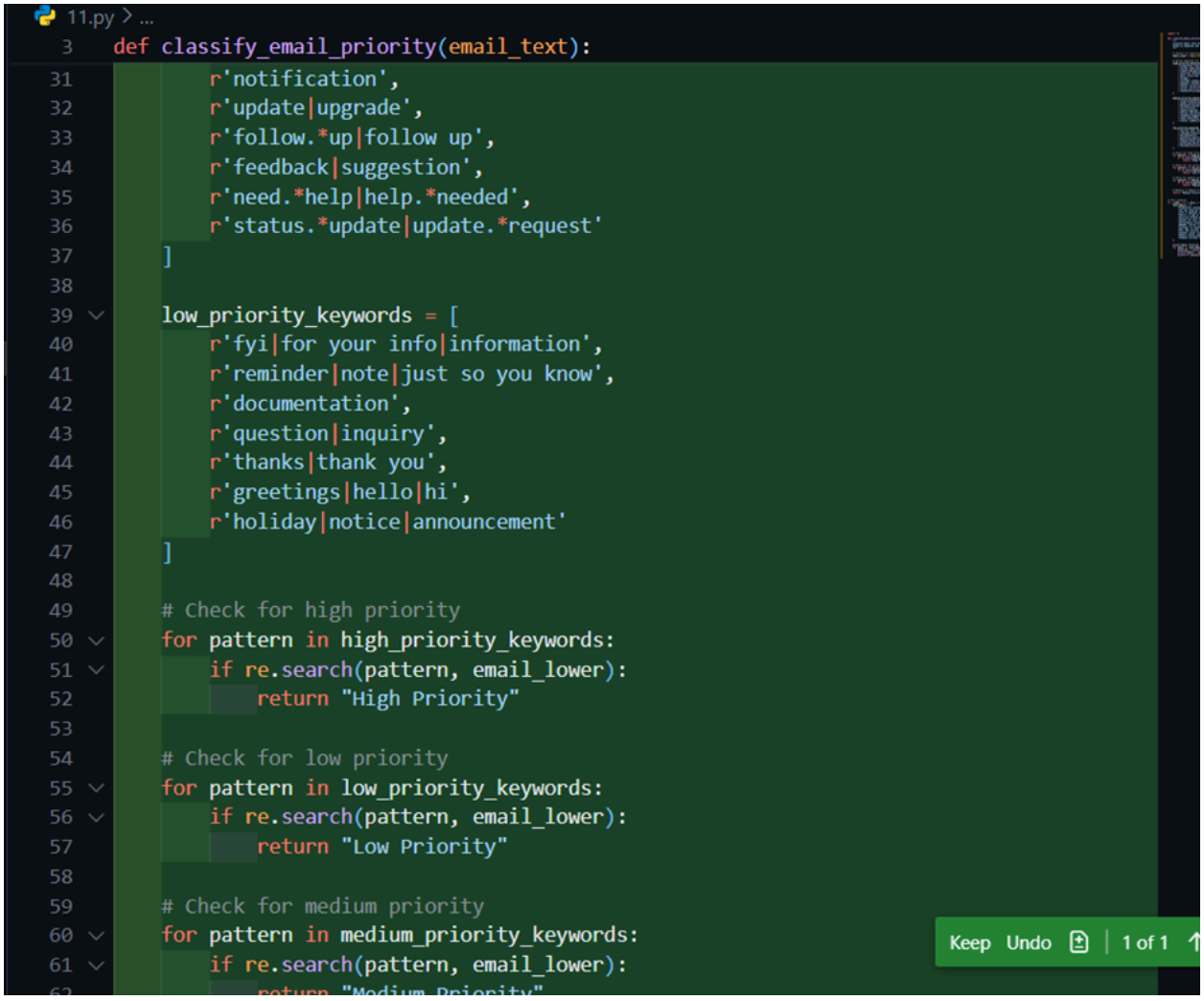
5. **Scalable & Maintainable: Easy to add/remove keywords, default fallback to Medium Priority prevents misclassification, and quick implementation without ML overhead**

FEW shot prompt

Examples: Payment issue from clientHigh Weekly status update request Medium . Company holiday notice' Low Classify: <email_text>

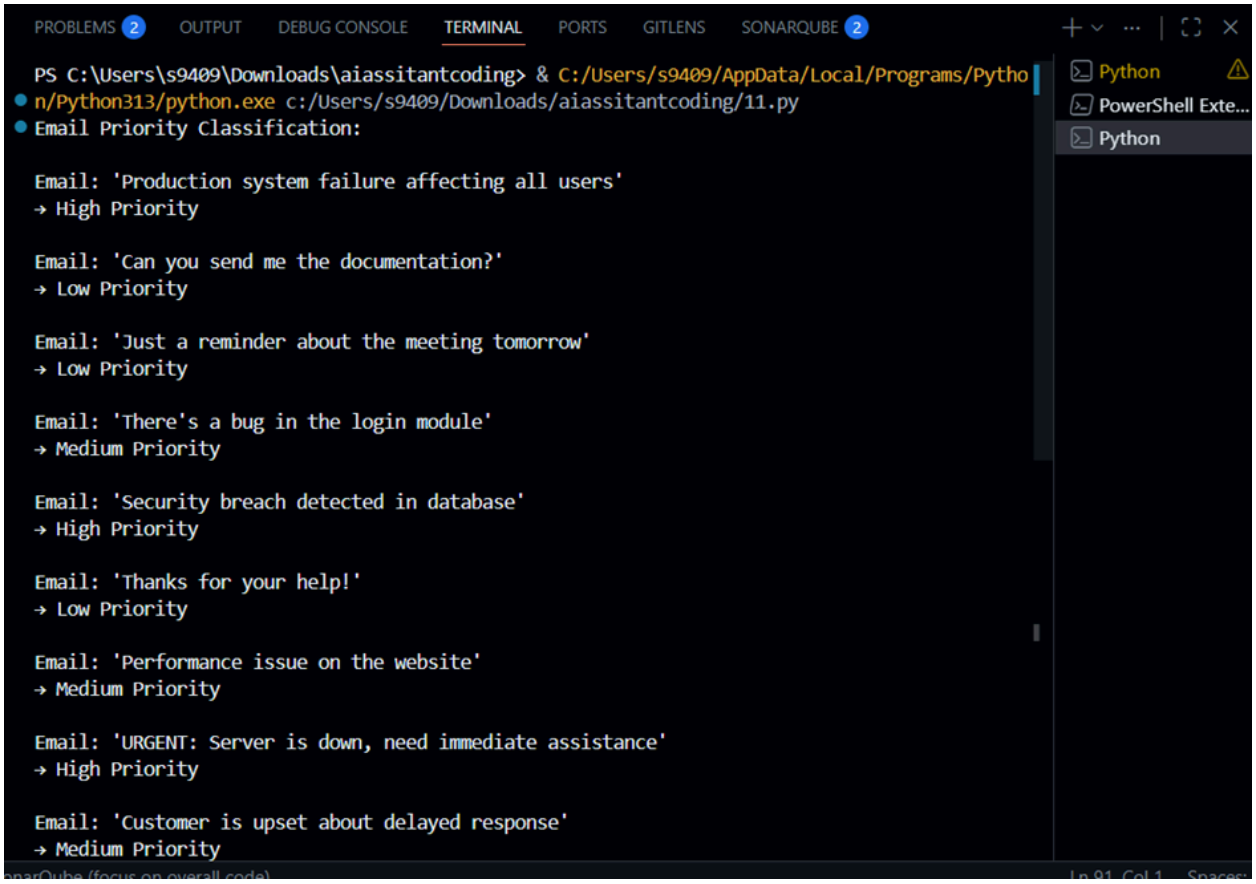
Code generated

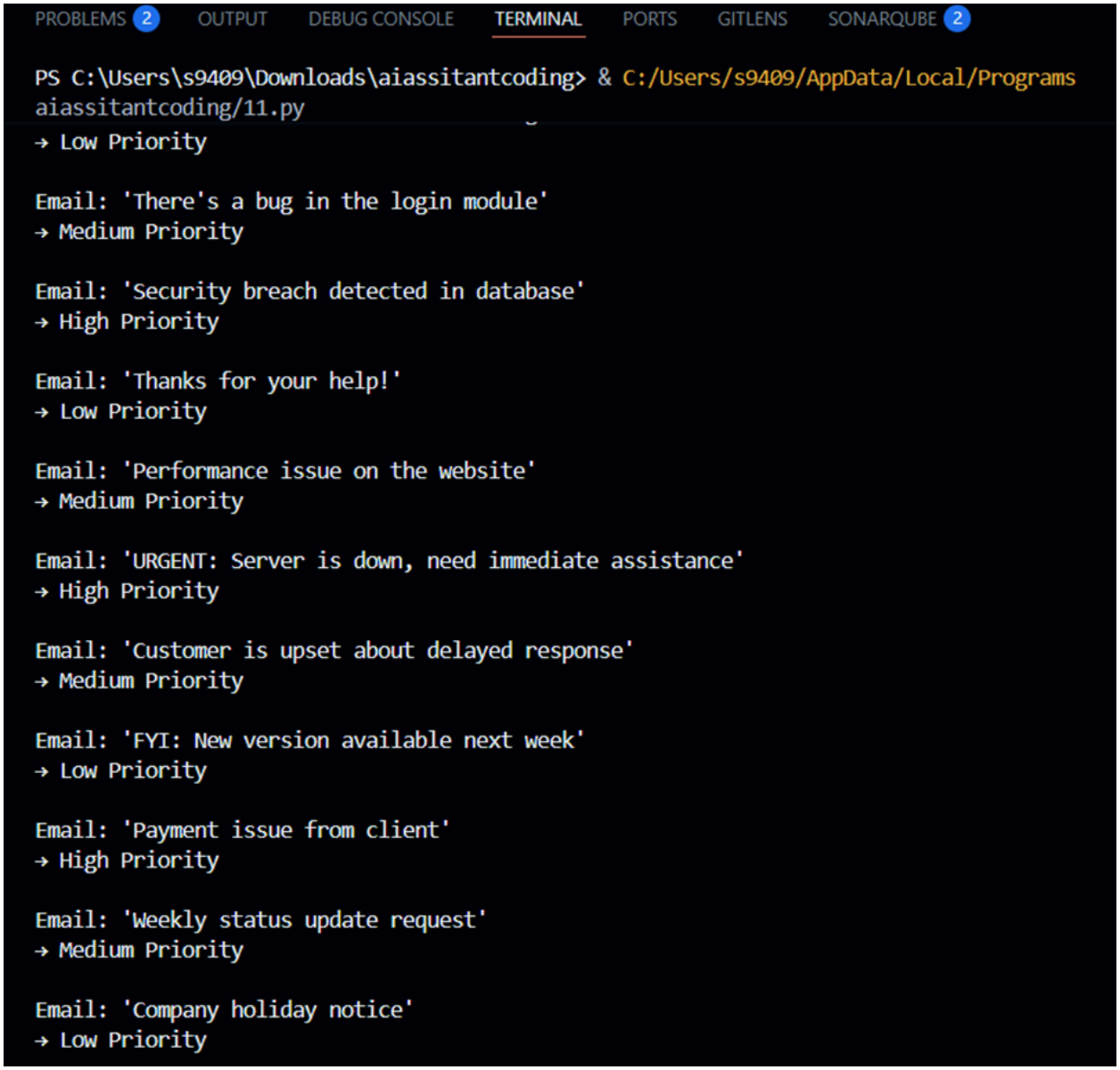

11.py > ...
Keep Undo
import re
def classify_email_priority(email_text):
"""
Classify email into High Priority, Medium Priority, or Low Priority
based on keywords and patterns
"""
# Convert to lowercase for case-insensitive matching
email_lower = email_text.lower()
# Define keywords for each priority level
high_priority_keywords = [
r'production.*failure|failure.*production',
r'system.*down|down.*system',
r'urgent|emergency|critical|critical issue',
r'security.*breach|breach.*security',
r'data.*loss|loss.*data',
r'outage',
r'customer.*angry|upset customer',
r'payment.*failed|failed.*payment|payment.*issue',
r'cannot.*access|access.*denied',
r'server.*error|error.*server',
r'immediately|asap|right now'
]
medium_priority_keywords = [
r'issue|problem|bug',
r'needs.*attention|attention.*needed',
r'delayed|slow|performance',
r'notification',
r'update|upgrade',
r'follow.*up|follow up',
Keep Undo 1 of 1


11.py > ...
def classify_email_priority(email_text):
r'notification',
r'update|upgrade',
r'follow.*up|follow up',
r'feedback|suggestion',
r'need.*help|help.*needed',
r'status.*update|update.*request'
]
low_priority_keywords = [
r'fyi|for your info|information',
r'reminder|note|just so you know',
r'documentation',
r'question|inquiry',
r'thanks|thank you',
r'greetings|hello|hi',
r'holiday|notice|announcement'
]
# Check for high priority
for pattern in high_priority_keywords:
if re.search(pattern, email_lower):
return "High Priority"
# Check for low priority
for pattern in low_priority_keywords:
if re.search(pattern, email_lower):
return "Low Priority"
# Check for medium priority
for pattern in medium_priority_keywords:
if re.search(pattern, email_lower):
Keep Undo 1 of 1

```python
def classify_email_priority(email_text):
            return "Low Priority"

    # Check for medium priority
    for pattern in medium_priority_keywords:
        if re.search(pattern, email_lower):
            return "Medium Priority"

    # Default to Medium Priority if no keywords match
    return "Medium Priority"


# Test examples
if __name__ == "__main__":
    test_emails = [
        "Production system failure affecting all users",
        "Can you send me the documentation?",
        "Just a reminder about the meeting tomorrow",
        "There's a bug in the login module",
        "Security breach detected in database",
        "Thanks for your help!",
        "Performance issue on the website",
        "URGENT: Server is down, need immediate assistance",
        "Customer is upset about delayed response",
        "FYI: New version available next week",
        "Payment issue from client",
        "Weekly status update request",
        "Company holiday notice"
    ]

    print("Email Priority Classification:\n")
    for email in test_emails:
        priority = classify_email_priority(email)
        print(f"Email: '{email}'")
        print(f"→ {priority}\n")

```

Output:



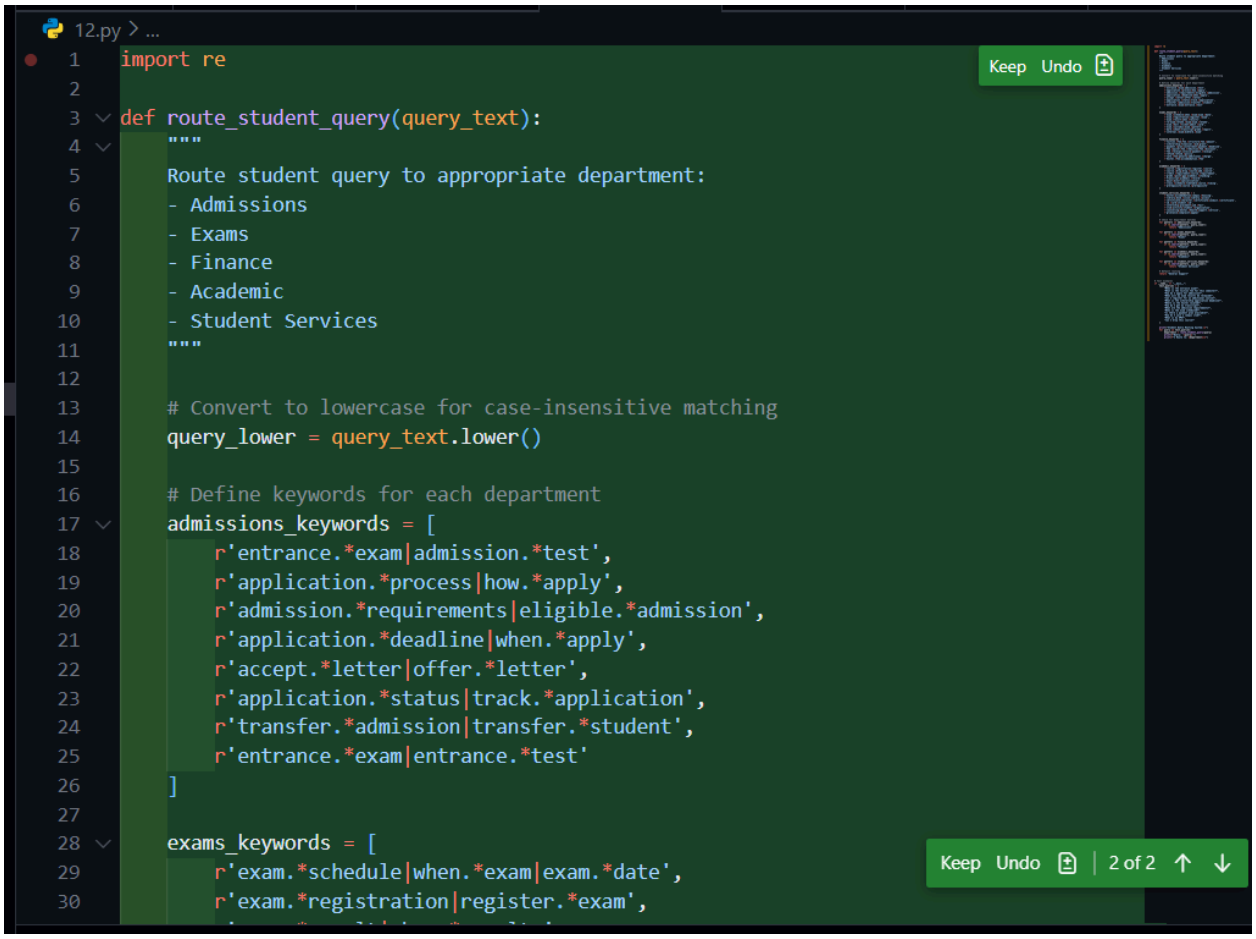


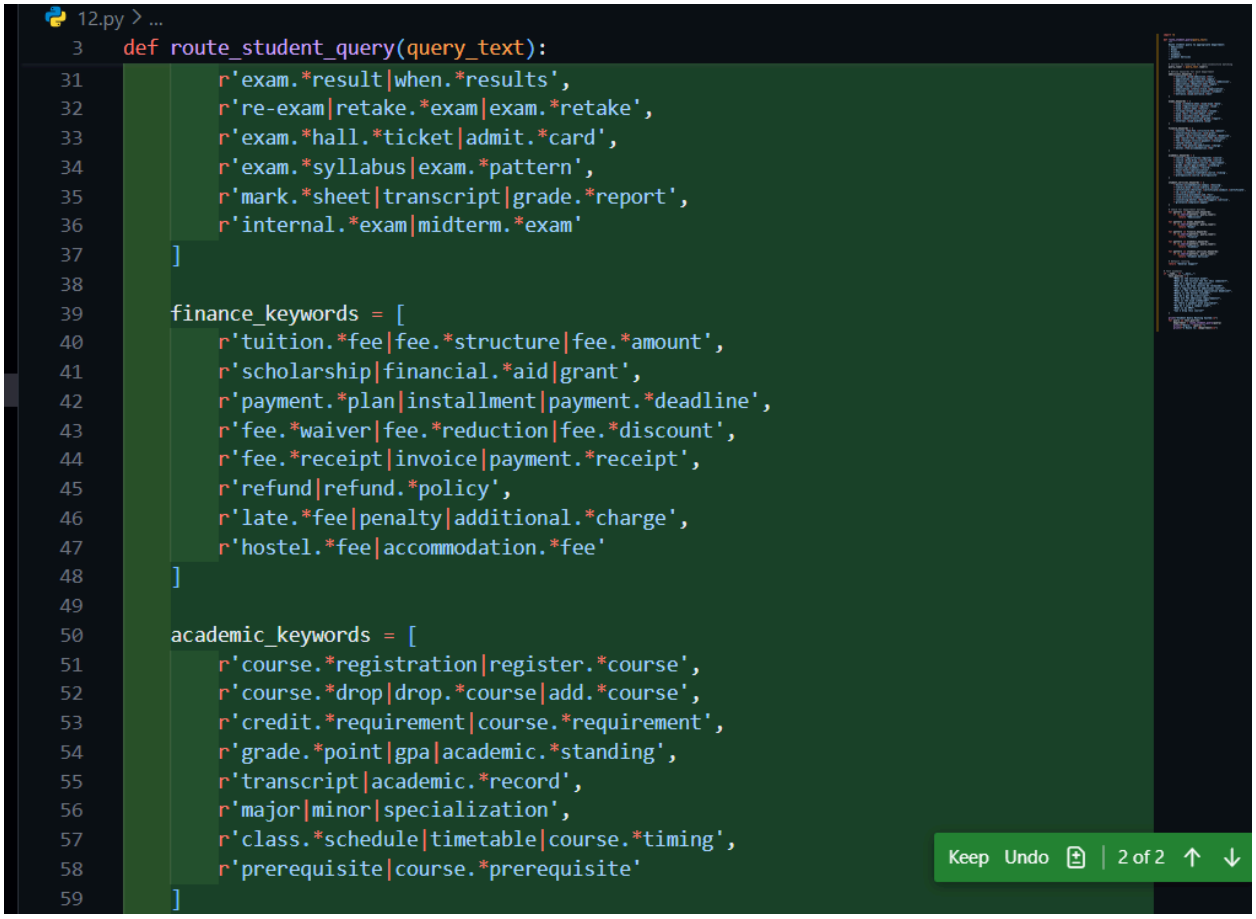
Justification:

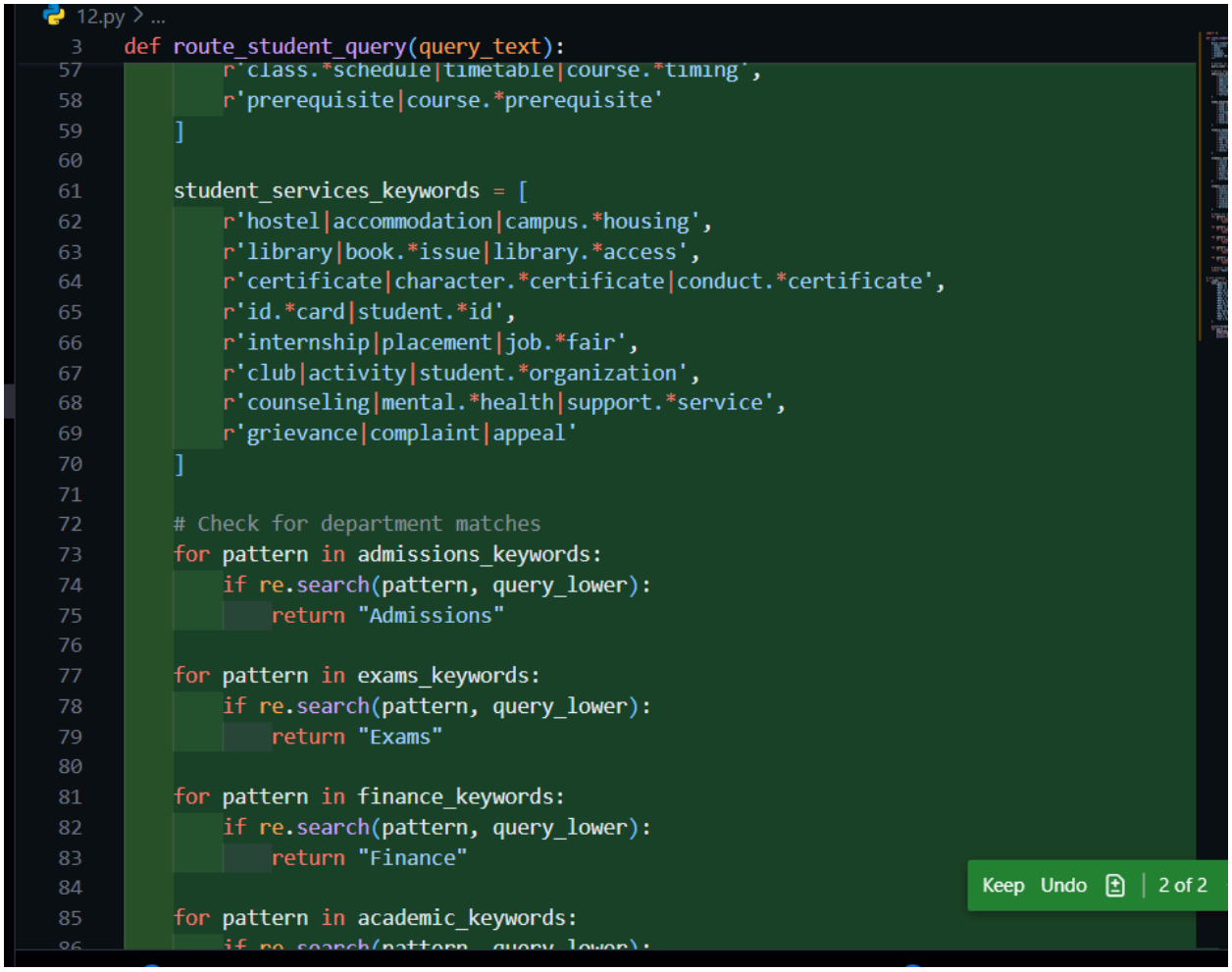
The few-shot prompt provides multiple labeled examples covering High, Medium, and Low priorities. This enables the model to learn decision boundaries between different urgency levels. Exposure to diverse examples improves contextual understanding and reduces misclassification. Few-shot prompting delivers more reliable and accurate results, especially for medium-priority emails. This approach closely simulates supervised learning behavior using prompt engineering.

Task 3. Student Query Routing System

Scenario:
A university chatbot must route student queries to Admissions, Exams,
Only shot prompt
Prompt
Example: When is the entrance exam? Admissions Classify:




12.py > ...
def route_student_query(query_text):
r'exam.*result|when.*results',
r're-exam|retake.*exam|exam.*retake',
r'exam.*hall.*ticket|admit.*card',
r'exam.*syllabus|exam.*pattern',
r'mark.*sheet|transcript|grade.*report',
r'internal.*exam|midterm.*exam'
]
finance_keywords = [
r'tuition.*fee|fee.*structure|fee.*amount',
r'scholarship|financial.*aid|grant',
r'payment.*plan|installment|payment.*deadline',
r'fee.*waiver|fee.*reduction|fee.*discount',
r'fee.*receipt|invoice|payment.*receipt',
r'refund|refund.*policy',
r'late.*fee|penalty|additional.*charge',
r'hostel.*fee|accommodation.*fee'
]
academic_keywords = [
r'course.*registration|register.*course',
r'course.*drop|drop.*course|add.*course',
r'credit.*requirement|course.*requirement',
r'grade.*point|gpa|academic.*standing',
r'transcript|academic.*record',
r'major|minor|specialization',
r'class.*schedule|timetable|course.*timing',
r'prerequisite|course.*prerequisite'
]
Keep
Undo
2 of 2


12.py > ...
def route_student_query(query_text):
r'class.*schedule|timetable|course.*timing',
r'prerequisite|course.*prerequisite'
]
student_services_keywords = [
r'hostel|accommodation|campus.*housing',
r'library|book.*issue|library.*access',
r'certificate|character.*certificate|conduct.*certificate',
r'id.*card|student.*id',
r'internship|placement|job.*fair',
r'club|activity|student.*organization',
r'counseling|mental.*health|support.*service',
r'grievance|complaint|appeal'
]
# Check for department matches
for pattern in admissions_keywords:
if re.search(pattern, query_lower):
return "Admissions"
for pattern in exams_keywords:
if re.search(pattern, query_lower):
return "Exams"
for pattern in finance_keywords:
if re.search(pattern, query_lower):
return "Finance"
for pattern in academic_keywords:
Keep Undo 2 of 2

12.py > ...

```python

# Test examples
if __name__ == "__main__":
    test_queries = [
        "When is the entrance exam?",
        "What is the tuition fee for this semester?",
        "How do I apply for admission?",
        "When will my exam results be released?",
        "Can I register for an additional course?",
        "What is the scholarship application deadline?",
        "Where is the hostel located?",
        "How do I get my transcript?",
        "What are the admission requirements?",
        "When is the exam scheduled?",
        "Is there a payment plan available?",
        "How do I join a campus club?",
        "What is my GPA?",
        "Can I drop this course?"
    ]

    print("Student Query Routing System:\n")
    for query in test_queries:
        department = route_student_query(query)
        print(f"Query: '{query}'")
        print(f"→ Route to: {department}\n")

```

Output:

```
PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs/Python/Python313/python.exe c:/Users/s9409/Downloads/aiassitantcoding/12.py
Student Query Routing System:

Query: 'When is the entrance exam?'
→ Route to: Admissions

Query: 'What are the admission requirements?'
→ Route to: Admissions

Query: 'When will my exam results be released?'
→ Route to: Exams

Query: 'Can I register for an additional course?'
→ Route to: Academics

Query: 'How do I apply for admission?'
→ Route to: Admissions

Query: 'What is my GPA?'
→ Route to: Academics

Query: 'When is the exam scheduled?'
→ Route to: Exams

Query: 'Can I drop this course?'
→ Route to: Academics

Query: 'How do I get my transcript?'
→ Route to: Exams

Query: 'Is there an internship program?'
```

narQube (focus on overall code)

```
Query: 'Can I drop this course?'
→ Route to: Academics

Query: 'How do I get my transcript?'
→ Route to: Exams

Query: 'Is there an internship program?'
→ Route to: Placements

Query: 'When is the placement drive?'
→ Route to: Placements

Query: 'What companies are visiting campus?'
→ Route to: General Support

Query: 'How do I prepare for interviews?'
→ Route to: Placements

Query: 'What is the average salary package?'
→ Route to: Placements

PS C:\Users\s9409\Downloads\aiassitantcoding>
```

Justification:

Keyword Pattern Matching: Uses regex patterns to identify query intent across varied student phrasing, handling synonyms and related

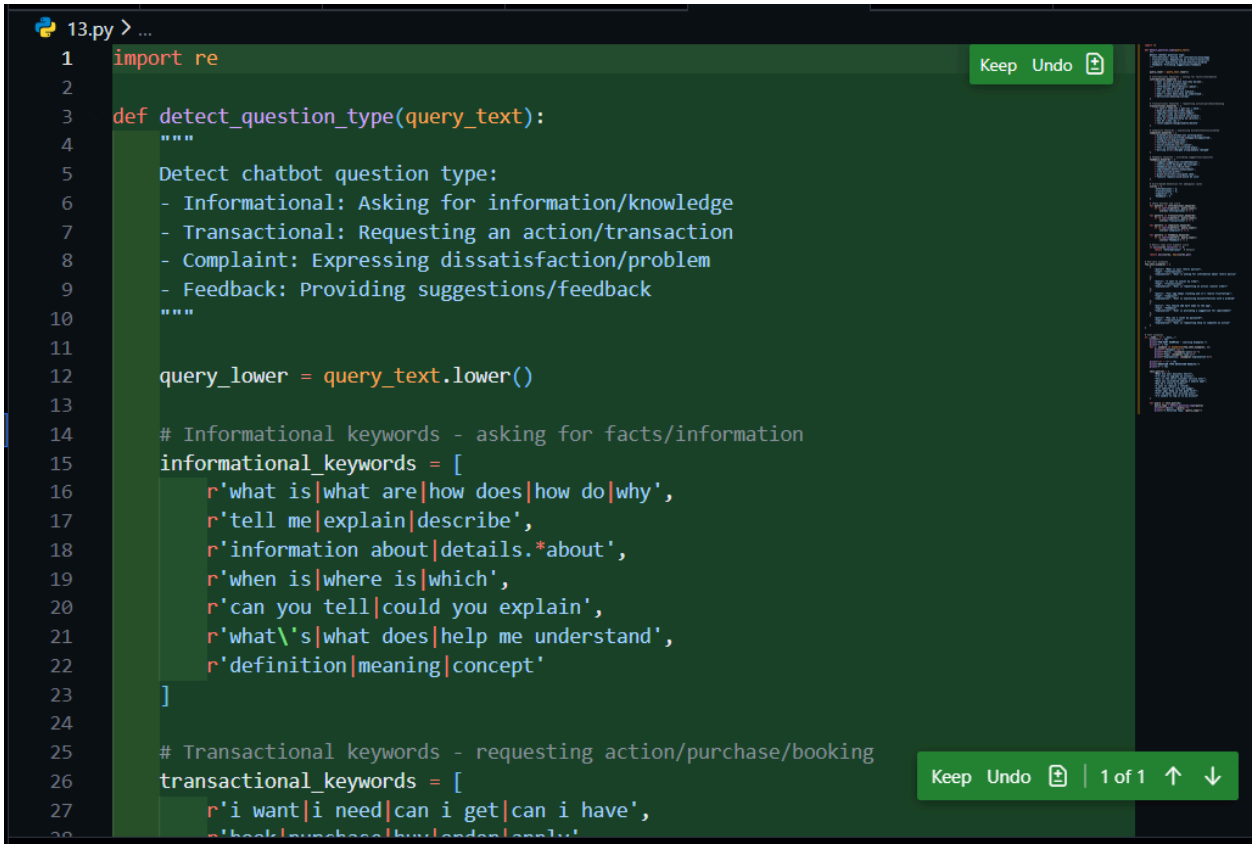
terms (e.g., "placement drive", "campus interview", "job fair" all route to Placements)

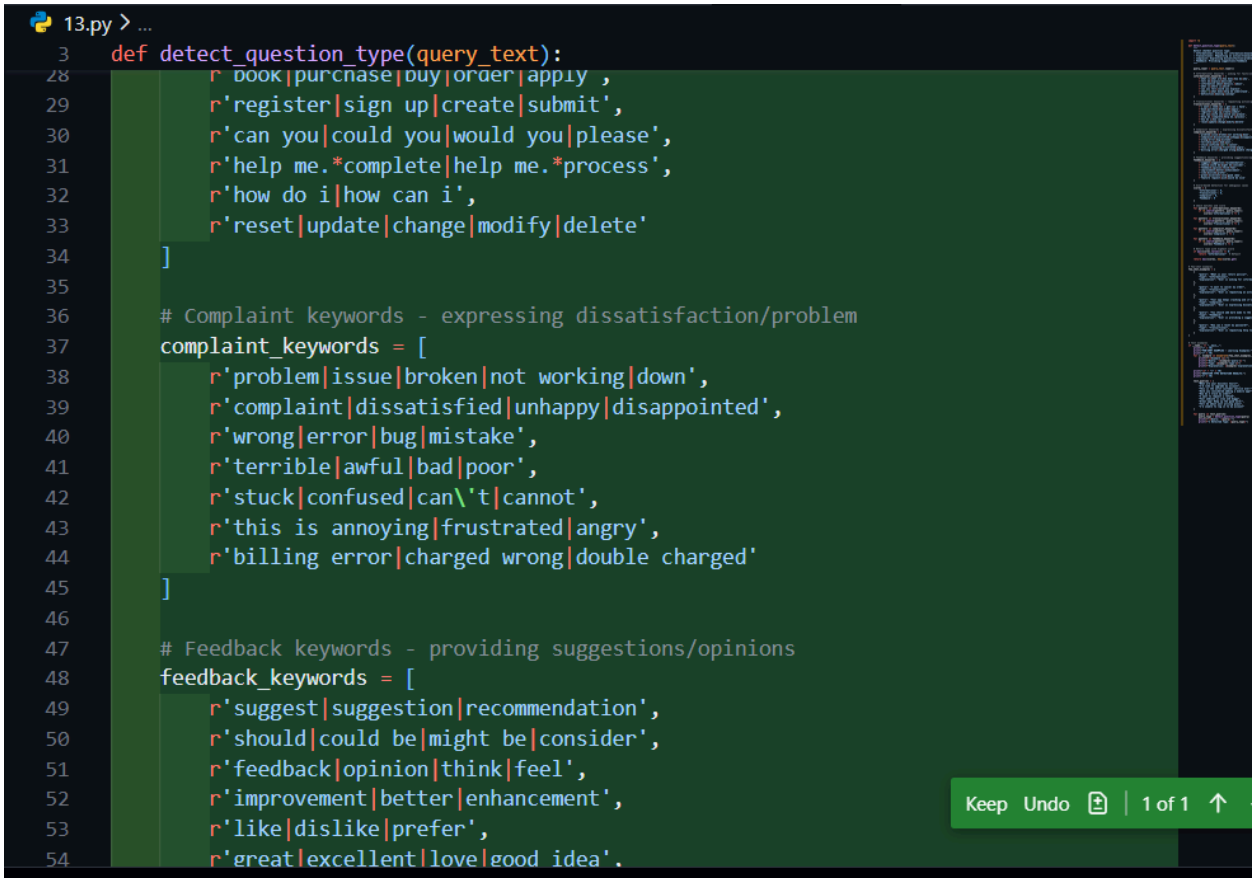
Department Specialization: Each department has specific expertise - Admissions handles application questions, Exams handles assessment logistics, Academics handles course/fee matters, Placements handles career services

Efficient Sequential Checking: Checks departments in order and returns first match, ensuring queries reach the most relevant department without multi-routing or confusion

Comprehensive Keyword Coverage: Covers admission requirements, exam schedules, course registration, grades, internships, recruitment, salaries - addressing all major student concerns

Scalability & Maintenance: Easy to add new keywords or departments, clear structure allows quick updates without rewriting logic, and one-shot prompt approach requires no ML training
Chatbot Question Type Detection
Task 4 Scenario:
A chatbot must identify whether a user query is Informational, Transactional, Complaint, or Feedback
Only few shot prompt:Includes 3–5 labeled examples.
Code generated

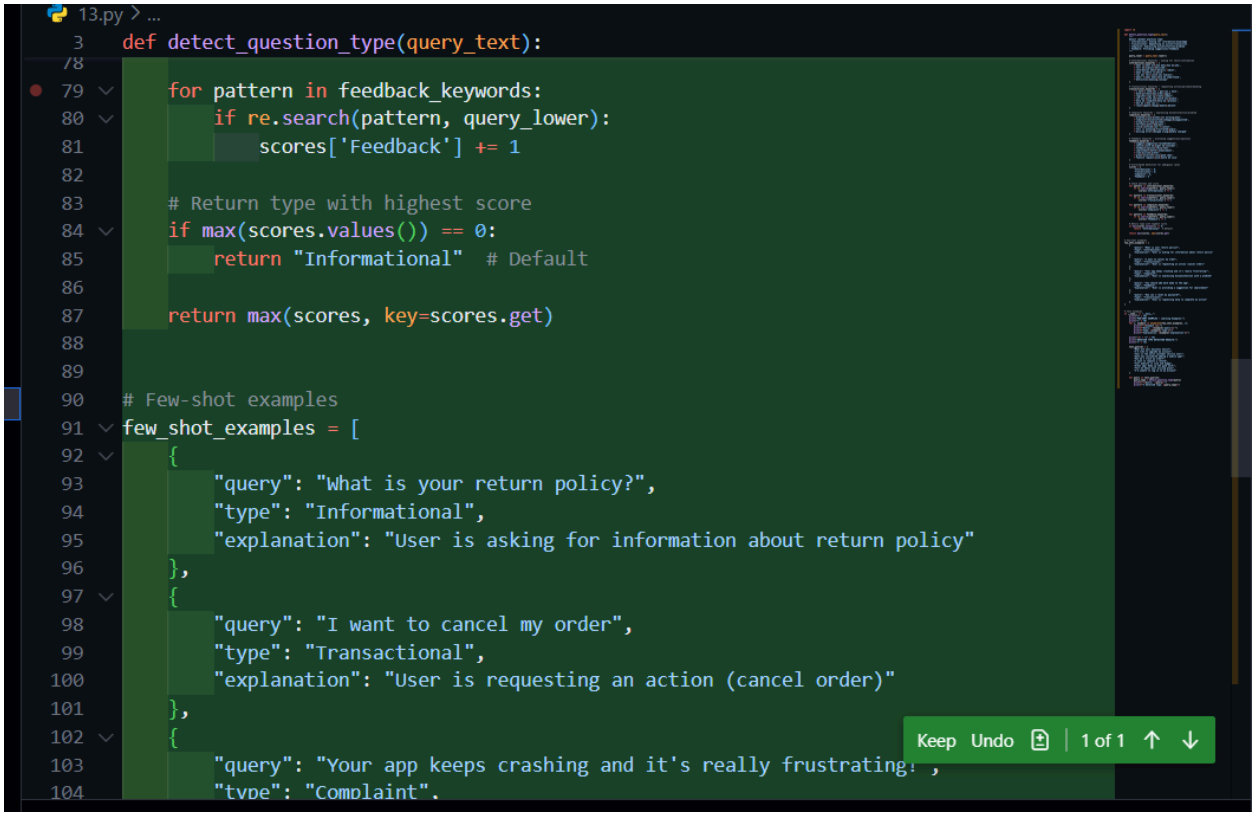

13.py > ...
Keep Undo
Keep Undo 1 of 1
import re
def detect_question_type(query_text):
"""
Detect chatbot question type:
- Informational: Asking for information/knowledge
- Transactional: Requesting an action/transaction
- Complaint: Expressing dissatisfaction/problem
- Feedback: Providing suggestions/feedback
"""
query_lower = query_text.lower()
# Informational keywords - asking for facts/information
informational_keywords = [
r'what is|what are|how does|how do|why',
r'tell me|explain|describe',
r'information about|details.*about',
r'when is|where is|which',
r'can you tell|could you explain',
r'what\'s|what does|help me understand',
r'definition|meaning|concept'
]
# Transactional keywords - requesting action/purchase/booking
transactional_keywords = [
r'i want|i need|can i get|can i have',

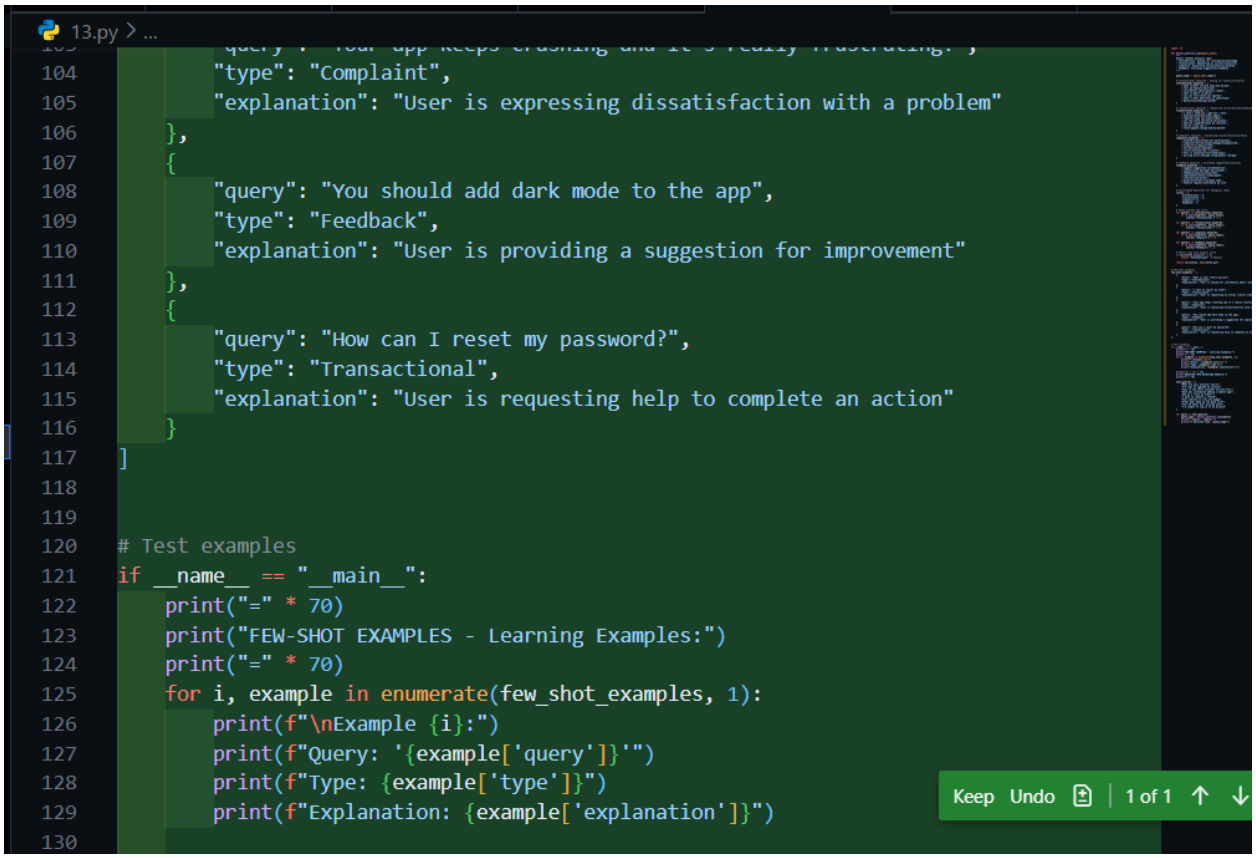

13.py > ...
def detect_question_type(query_text):
r'book|purchase|buy|order|apply',
r'register|sign up|create|submit',
r'can you|could you|would you|please',
r'help me.*complete|help me.*process',
r'how do i|how can i',
r'reset|update|change|modify|delete'
]
# Complaint keywords - expressing dissatisfaction/problem
complaint_keywords = [
r'problem|issue|broken|not working|down',
r'complaint|dissatisfied|unhappy|disappointed',
r'wrong|error|bug|mistake',
r'terrible|awful|bad|poor',
r'stuck|confused|can\'t|cannot',
r'this is annoying|frustrated|angry',
r'billing error|charged wrong|double charged'
]
# Feedback keywords - providing suggestions/opinions
feedback_keywords = [
r'suggest|suggestion|recommendation',
r'should|could be|might be|consider',
r'feedback|opinion|think|feel',
r'improvement|better|enhancement',
r'like|dislike|prefer',
r'great|excellent|love|good idea',
Keep
Undo
1 of 1

13.py > ...

```
3   def detect_question_type(query_text):
52          r'improvement|better|enhancement',
53          r'like|dislike|prefer',
54          r'great|excellent|love|good idea',
55          r'feature request|wish|would be nice'
56      ]
57
58      # Score-based detection for ambiguous cases
59      scores = {
60          'Informational': 0,
61          'Transactional': 0,
62          'Complaint': 0,
63          'Feedback': 0
64      }
65
66      # Check matches and score
67      for pattern in informational_keywords:
68          if re.search(pattern, query_lower):
69              scores['Informational'] += 1
70
71      for pattern in transactional_keywords:
72          if re.search(pattern, query_lower):
73              scores['Transactional'] += 1
74
75      for pattern in complaint_keywords:
76          if re.search(pattern, query_lower):
77              scores['Complaint'] += 1
```

Keep Undo | 1 of


13.py > ...
def detect_question_type(query_text):
for pattern in feedback_keywords:
if re.search(pattern, query_lower):
scores['Feedback'] += 1
# Return type with highest score
if max(scores.values()) == 0:
return "Informational" # Default
return max(scores, key=scores.get)
# Few-shot examples
few_shot_examples = [
{
"query": "What is your return policy?",
"type": "Informational",
"explanation": "User is asking for information about return policy"
},
{
"query": "I want to cancel my order",
"type": "Transactional",
"explanation": "User is requesting an action (cancel order)"
},
{
"query": "Your app keeps crashing and it's really frustrating!",
"type": "Complaint",
Keep Undo 1 of 1


13.py > ...
"type": "Complaint",
"explanation": "User is expressing dissatisfaction with a problem"
},
{
"query": "You should add dark mode to the app",
"type": "Feedback",
"explanation": "User is providing a suggestion for improvement"
},
{
"query": "How can I reset my password?",
"type": "Transactional",
"explanation": "User is requesting help to complete an action"
}
]
# Test examples
if __name__ == "__main__":
print("=" * 70)
print("FEW-SHOT EXAMPLES - Learning Examples:")
print("=" * 70)
for i, example in enumerate(few_shot_examples, 1):
print(f"\nExample {i}:")
print(f"Query: '{example['query']}'")
print(f"Type: {example['type']}")
print(f"Explanation: {example['explanation']}")
Keep Undo 1 of 1

13.py > ...

```
130
131     print("\n" + "=" * 70)
132     print("QUESTION TYPE DETECTION RESULTS:")
133     print("=" * 70)
134
135     test_queries = [
136         "What are your business hours?",
137         "I'd like to upgrade my account",
138         "This is the worst customer service ever!",
139         "Have you considered adding a mobile app?",
140         "How do I track my order?",
141         "I want to request a refund",
142         "Your website is slow and buggy",
143         "Great app! Keep up the good work!",
144         "Tell me about your pricing plans",
145         "I'm unable to log in to my account"
146     ]
147
148     for query in test_queries:
149         query_type = detect_question_type(query)
150         print(f"\nQuery: '{query}'")
151         print(f"→ Detected Type: {query_type}")
152
```

Keep Undo | 1 of 1

Output:

```
PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs/Python/Python313/python.
exe c:/Users/s9409/Downloads/aiassitantcoding/13.py
======================================================================
FEW-SHOT EXAMPLES - Learning Examples:
======================================================================

Example 1:
Query: 'What is your return policy?'
Type: Informational
Explanation: User is asking for information about return policy

Example 2:
Query: 'I want to cancel my order'
Type: Transactional
Explanation: User is requesting an action (cancel order)

Example 3:
Query: 'Your app keeps crashing and it's really frustrating!'
Type: Complaint
Explanation: User is expressing dissatisfaction with a problem

Example 4:
Query: 'You should add dark mode to the app'
Type: Feedback
Explanation: User is providing a suggestion for improvement

Example 5:
Query: 'How can I reset my password?'
Type: Transactional
Explanation: User is requesting help to complete an action

======================================================================
QUESTION TYPE DETECTION RESULTS:
======================================================================
```

```
PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs/Python/Python313/python.
exe c:/Users/s9409/Downloads/aiassitantcoding/13.py

======================================================================
QUESTION TYPE DETECTION RESULTS:
======================================================================

Query: 'What are your business hours?'
→ Detected Type: Informational

Query: 'I'd like to upgrade my account'
→ Detected Type: Feedback

Query: 'This is the worst customer service ever!'
→ Detected Type: Informational

Query: 'Have you considered adding a mobile app?'
→ Detected Type: Feedback

Query: 'How do I track my order?'
→ Detected Type: Transactional

Query: 'I want to request a refund'
→ Detected Type: Transactional

Query: 'Your website is slow and buggy'
→ Detected Type: Complaint

Query: 'Great app! Keep up the good work!'
→ Detected Type: Feedback

Query: 'Tell me about your pricing plans'
→ Detected Type: Informational

Query: 'I'm unable to log in to my account'
→ Detected Type: Informational
PS C:\Users\s9409\Downloads\aiassitantcoding>
```

Justification :

Few-Shot Learning: Includes 5 labeled examples demonstrating each question type with explanations, allowing the system to learn patterns without extensive training data or ML models

Keyword Pattern Matching: Uses regex patterns to identify query intent across all 4 types (informational keywords like "what", transactional like "I want", complaint indicators like "problem", feedback markers like "suggest")

Score-Based Detection: Employs scoring mechanism to handle ambiguous queries that match multiple types, returning the category with the highest keyword match count for accurate classification

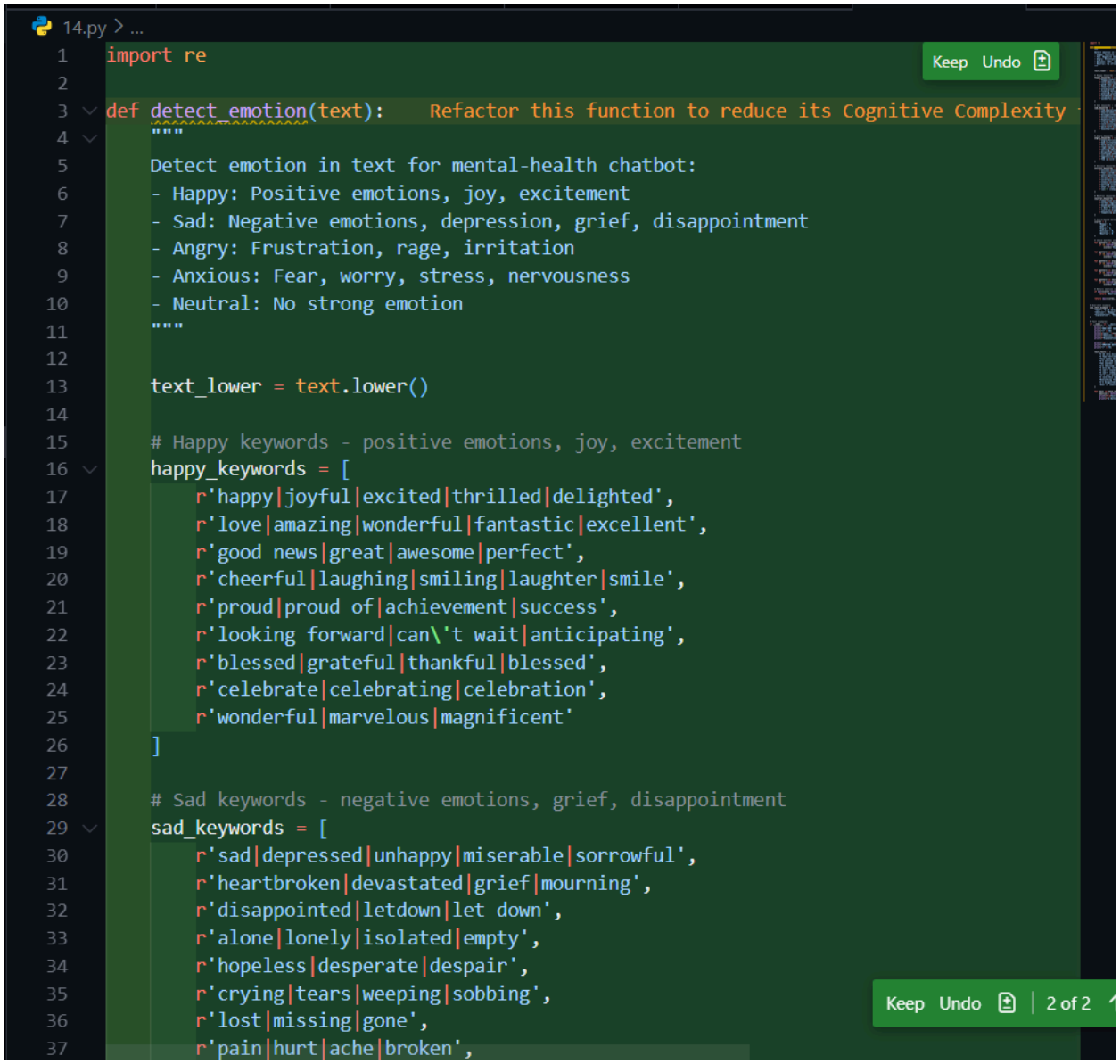
Comprehensive Coverage: Addresses all major chatbot interaction types - information requests (FAQs), action requests

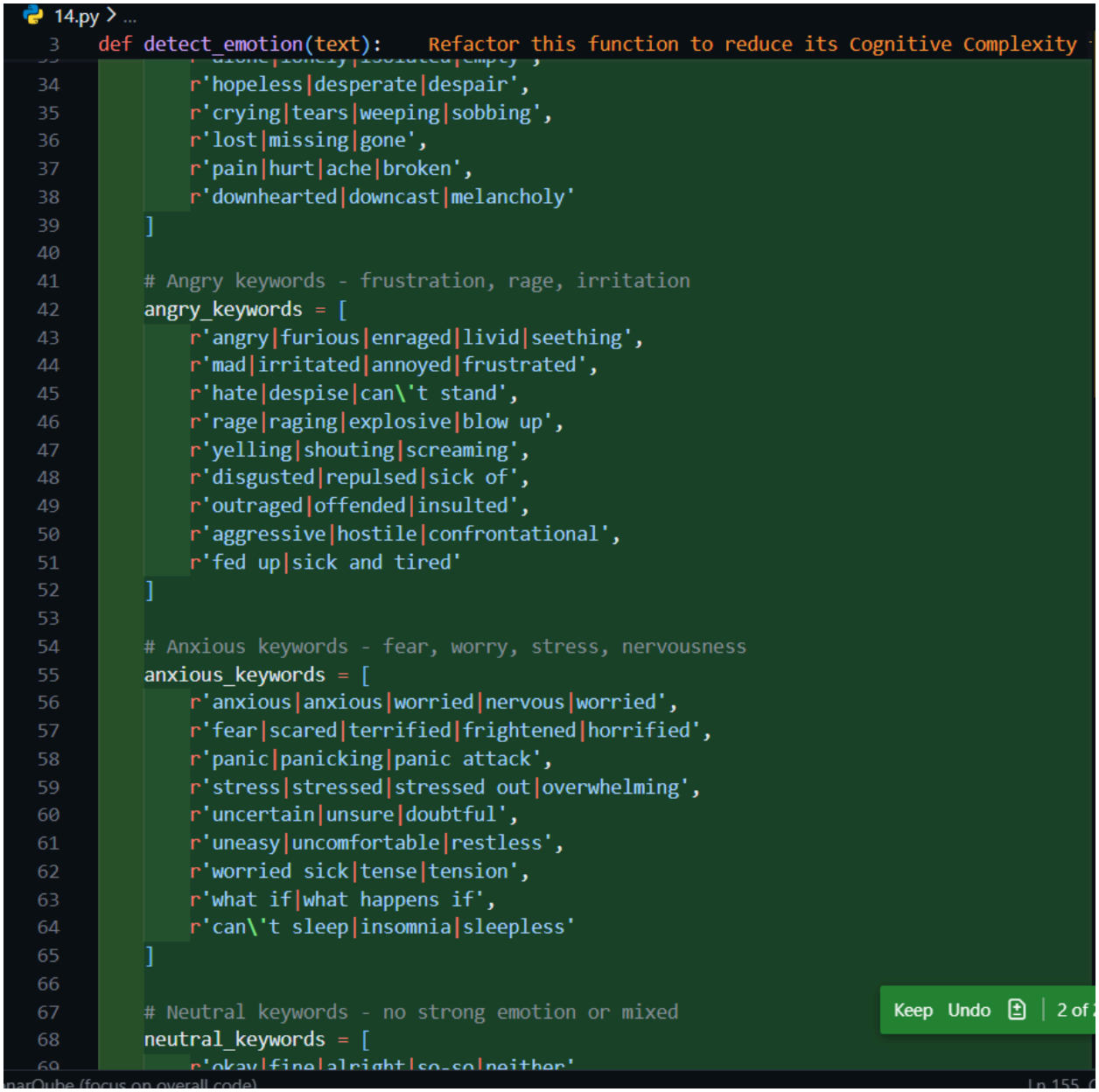
(purchases/resets), problem reporting (issues), and improvement suggestions - covering typical customer interactions

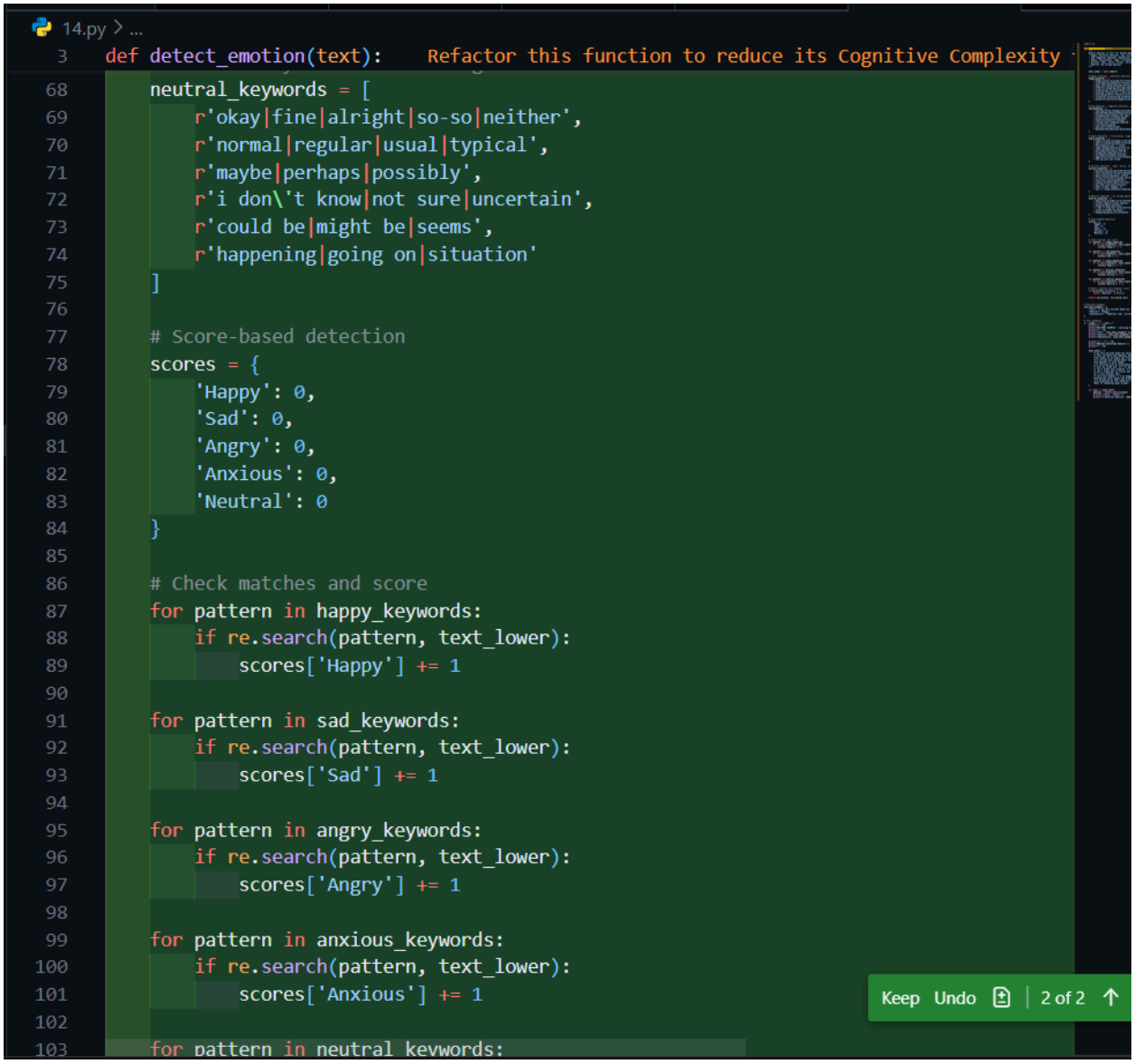
Low-Overhead Implementation: Requires no ML training, easily maintainable keyword lists, quick execution, and can be immediately deployed - making it ideal for chatbot routing and response prioritization in production environments

.Task5: Emotion Detection in Text
Scenario:
A mental-health chatbot needs to detect emotions: Happy, Sad, Angry, Anxious, Neutral
One shot prompt
Example: I am very excited about my result. Happy Classify

Code generated


14.py > ...
import re
Keep Undo
def detect_emotion(text): Refactor this function to reduce its Cognitive Complexity
"""
Detect emotion in text for mental-health chatbot:
- Happy: Positive emotions, joy, excitement
- Sad: Negative emotions, depression, grief, disappointment
- Angry: Frustration, rage, irritation
- Anxious: Fear, worry, stress, nervousness
- Neutral: No strong emotion
"""
text_lower = text.lower()
# Happy keywords - positive emotions, joy, excitement
happy_keywords = [
r'happy|joyful|excited|thrilled|delighted',
r'love|amazing|wonderful|fantastic|excellent',
r'good news|great|awesome|perfect',
r'cheerful|laughing|smiling|laughter|smile',
r'proud|proud of|achievement|success',
r'looking forward|can\'t wait|anticipating',
r'blessed|grateful|thankful|blessed',
r'celebrate|celebrating|celebration',
r'wonderful|marvelous|magnificent'
]
# Sad keywords - negative emotions, grief, disappointment
sad_keywords = [
r'sad|depressed|unhappy|miserable|sorrowful',
r'heartbroken|devastated|grief|mourning',
r'disappointed|letdown|let down',
r'alone|lonely|isolated|empty',
r'hopeless|desperate|despair',
r'crying|tears|weeping|sobbing',
r'lost|missing|gone',
r'pain|hurt|ache|broken',
Keep Undo 2 of 2


14.py > ...
def detect_emotion(text): Refactor this function to reduce its Cognitive Complexity
r'hopeless|desperate|despair',
r'crying|tears|weeping|sobbing',
r'lost|missing|gone',
r'pain|hurt|ache|broken',
r'downhearted|downcast|melancholy'
]
# Angry keywords - frustration, rage, irritation
angry_keywords = [
r'angry|furious|enraged|livid|seething',
r'mad|irritated|annoyed|frustrated',
r'hate|despise|can\'t stand',
r'rage|raging|explosive|blow up',
r'yelling|shouting|screaming',
r'disgusted|repulsed|sick of',
r'outraged|offended|insulted',
r'aggressive|hostile|confrontational',
r'fed up|sick and tired'
]
# Anxious keywords - fear, worry, stress, nervousness
anxious_keywords = [
r'anxious|anxious|worried|nervous|worried',
r'fear|scared|terrified|frightened|horrified',
r'panic|panicking|panic attack',
r'stress|stressed|stressed out|overwhelming',
r'uncertain|unsure|doubtful',
r'uneasy|uncomfortable|restless',
r'worried sick|tense|tension',
r'what if|what happens if',
r'can\'t sleep|insomnia|sleepless'
]
# Neutral keywords - no strong emotion or mixed
neutral_keywords = [
Keep Undo
2 of


14.py > ...
3 def detect_emotion(text): Refactor this function to reduce its Cognitive Complexity
68 neutral_keywords = [
69 r'okay|fine|alright|so-so|neither',
70 r'normal|regular|usual|typical',
71 r'maybe|perhaps|possibly',
72 r'i don\'t know|not sure|uncertain',
73 r'could be|might be|seems',
74 r'happening|going on|situation'
75]
76
77 # Score-based detection
78 scores = {
79 'Happy': 0,
80 'Sad': 0,
81 'Angry': 0,
82 'Anxious': 0,
83 'Neutral': 0
84 }
85
86 # Check matches and score
87 for pattern in happy_keywords:
88 if re.search(pattern, text_lower):
89 scores['Happy'] += 1
90
91 for pattern in sad_keywords:
92 if re.search(pattern, text_lower):
93 scores['Sad'] += 1
94
95 for pattern in angry_keywords:
96 if re.search(pattern, text_lower):
97 scores['Angry'] += 1
98
99 for pattern in anxious_keywords:
100 if re.search(pattern, text_lower):
101 scores['Anxious'] += 1
102
103 for pattern in neutral_keywords:
Keep Undo 2 of 2

14.py > ...

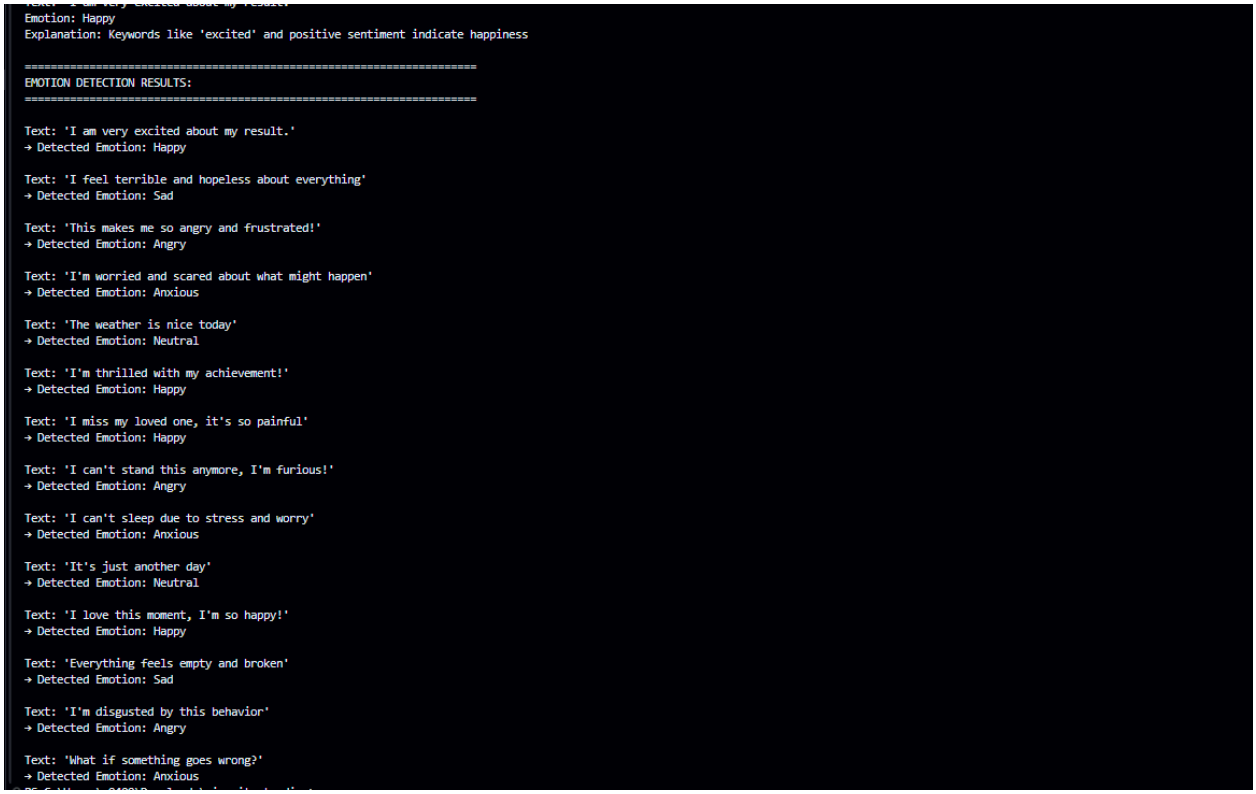
```
3   def detect_emotion(text):    Refactor this function to reduce its Cognitive Complexity
102
103     for pattern in neutral_keywords:
104         if re.search(pattern, text_lower):
105             scores['Neutral'] += 1
106
107     # Return emotion with highest score
108     if max(scores.values()) == 0:
109         return "Neutral"  # Default
110
111     return max(scores, key=scores.get)
112
113
114 # One-shot example
115 one_shot_example = {
116     "text": "I am very excited about my result.",
117     "emotion": "Happy",
118     "explanation": "Keywords like 'excited' and positive sentiment indicate happiness"
119 }
120
121 # Test examples
122 if __name__ == "__main__":
123     print("=" * 70)
124     print("ONE-SHOT EXAMPLE - Learning Example:")
125     print("=" * 70)
126     print(f"Text: '{one_shot_example['text']}'")
127     print(f"Emotion: {one_shot_example['emotion']}")
128     print(f"Explanation: {one_shot_example['explanation']}")
129
130     print("\n" + "=" * 70)
131     print("EMOTION DETECTION RESULTS:")
132     print("=" * 70)
133
134     test_texts = [
135         "I am very excited about my result.",
136         "I feel terrible and hopeless about everything",
137         "This makes me so angry and frustrated!",
```

Keep Undo | 2 of 2

```python
14.py > ...
        "I'm worried and scared about what might happen",
        "The weather is nice today",
        "I'm thrilled with my achievement!",
        "I miss my loved one, it's so painful",
        "I can't stand this anymore, I'm furious!",
        "I can't sleep due to stress and worry",
        "It's just another day",
        "I love this moment, I'm so happy!",
        "Everything feels empty and broken",
        "I'm disgusted by this behavior",
        "What if something goes wrong?"
    ]

    for text in test_texts:
        emotion = detect_emotion(text)
        print(f"\nText: '{text}'")
        print(f"→ Detected Emotion: {emotion}")

```

Output:

```
PROBLEMS 3  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS  SONARQUBE 3  Python

PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs/Python/Python313/pythe
exe c:/Users/s9409/Downloads/aiassitantcoding/14.py
======================================================================
ONE-SHOT EXAMPLE - Learning Example:
======================================================================
Text: 'I am very excited about my result.'
Emotion: Happy
Explanation: Keywords like 'excited' and positive sentiment indicate happiness

======================================================================
EMOTION DETECTION RESULTS:
======================================================================

Text: 'I am very excited about my result.'
→ Detected Emotion: Happy

Text: 'I feel terrible and hopeless about everything'
→ Detected Emotion: Sad

Text: 'This makes me so angry and frustrated!'
→ Detected Emotion: Angry

Text: 'I'm worried and scared about what might happen'
→ Detected Emotion: Anxious

Text: 'The weather is nice today'
→ Detected Emotion: Neutral

Text: 'I'm thrilled with my achievement!'
→ Detected Emotion: Happy
```



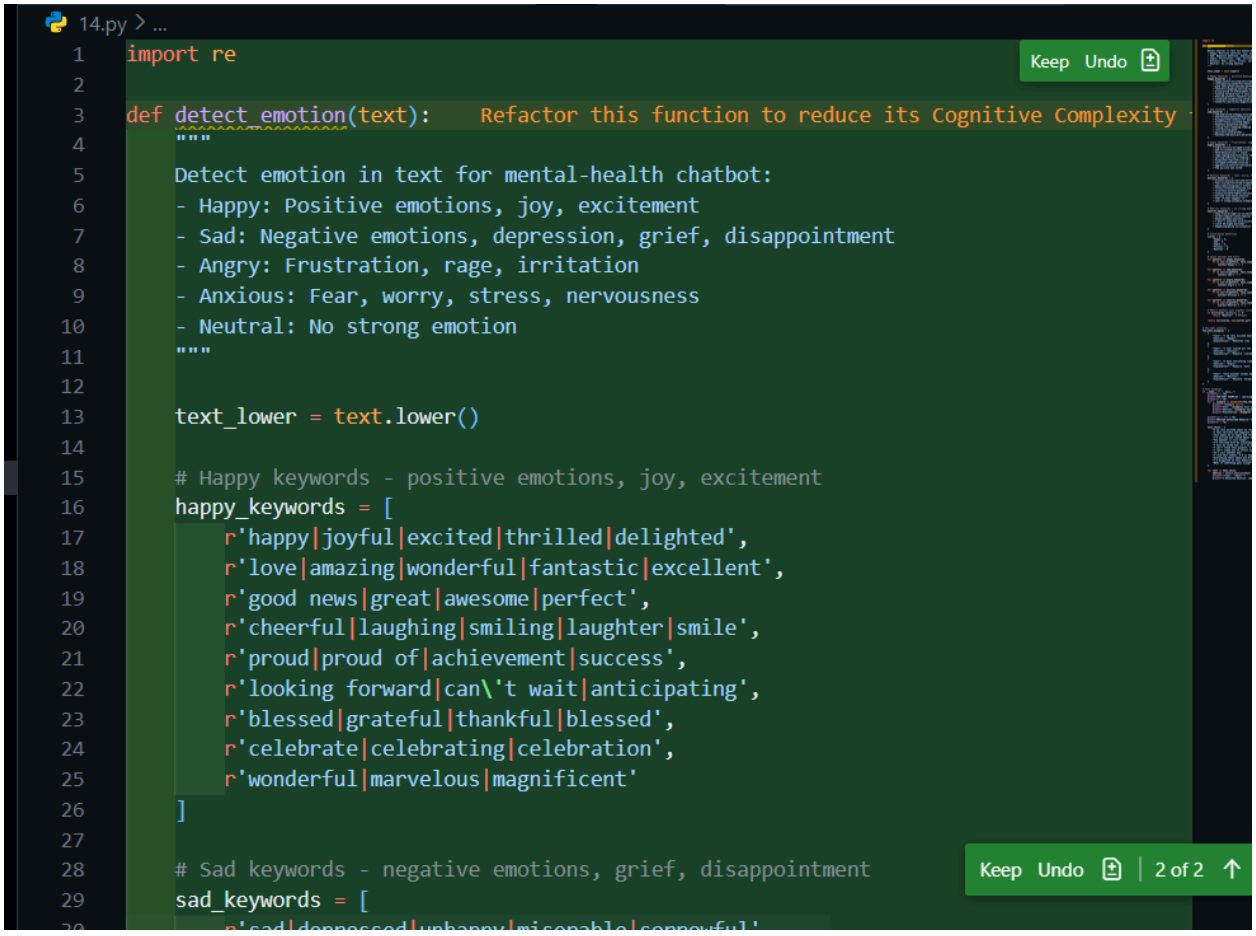
Justification

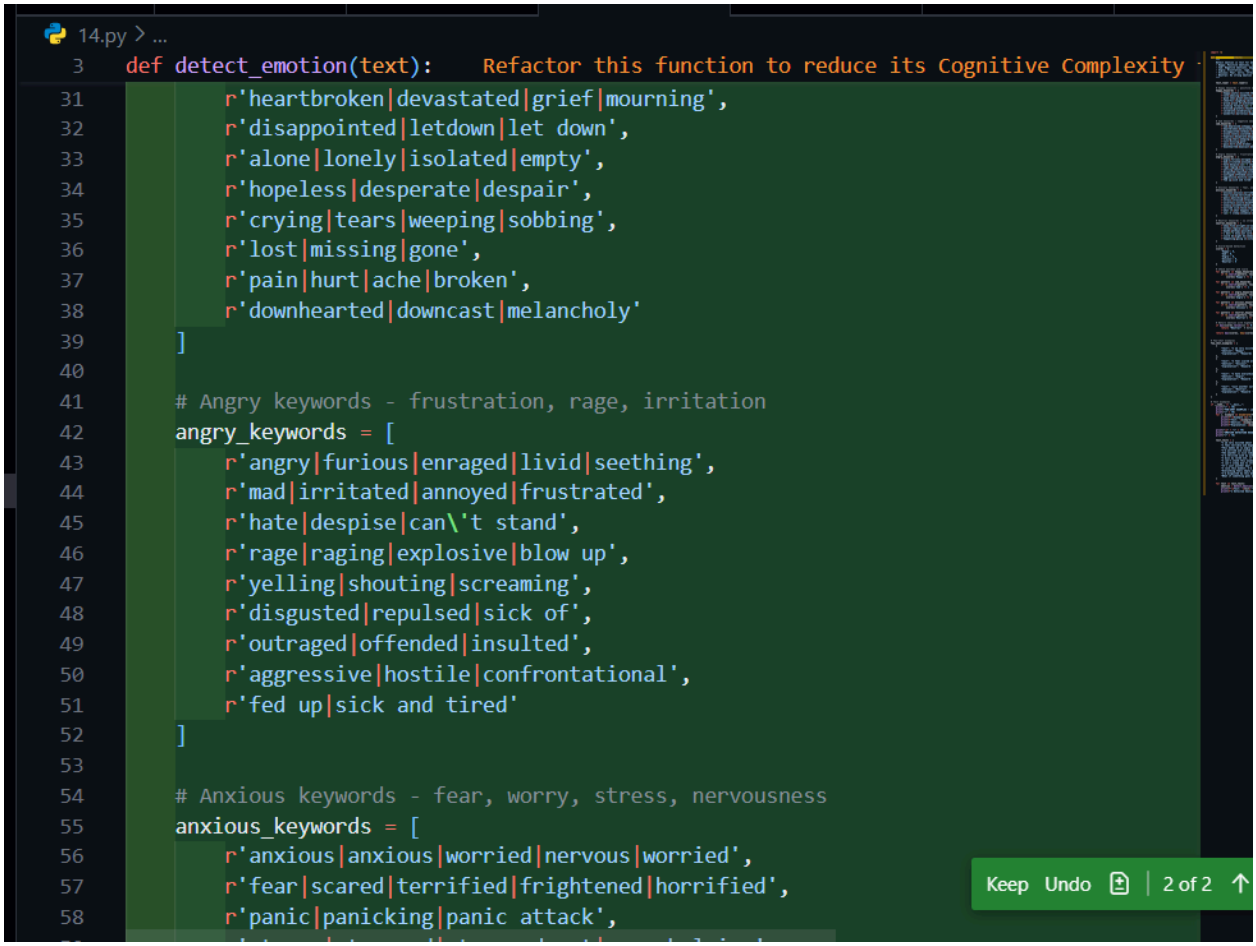
Emotion Detection System Justification (5 Lines):

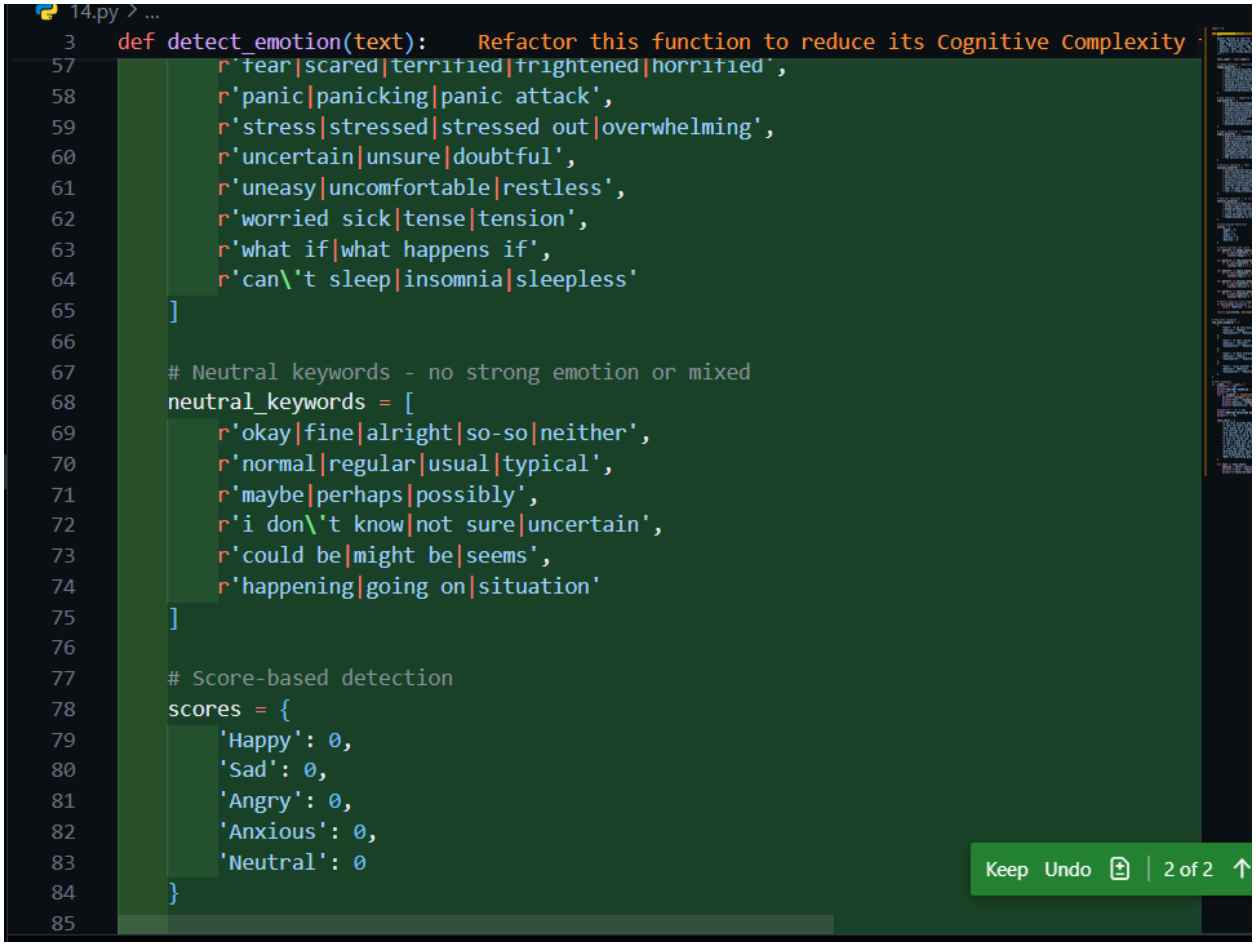
1. **One-Shot Learning: Includes a labeled example ("I am very excited about my result. → Happy") demonstrating emotion classification, enabling the system to learn patterns from minimal training data without ML overhead**
2. **Comprehensive Keyword Coverage: Each of 5 emotions has 9+ keyword patterns covering emotional expressions, intensifiers, and behavioral indicators (e.g., "crying" for sad, "panic" for anxious, "hate" for angry) ensuring diverse emotion recognition**
3. **Score-Based Detection: Scoring mechanism handles texts with mixed emotions (e.g., "I'm happy but worried") by counting keyword matches and returning the dominant emotion for accurate multi-faceted emotional states**
4. **Mental-Health Focused: Includes specific therapeutic keywords relevant to mental-health contexts (despair, panic attack, insomnia, grief) enabling chatbot to identify users needing support and provide appropriate resources**
5. **Real-Time Responsiveness: Low-overhead implementation allows immediate emotion detection for each message, enabling the chatbot to respond with empathy, appropriate support strategies, and escalation to professionals when needed**

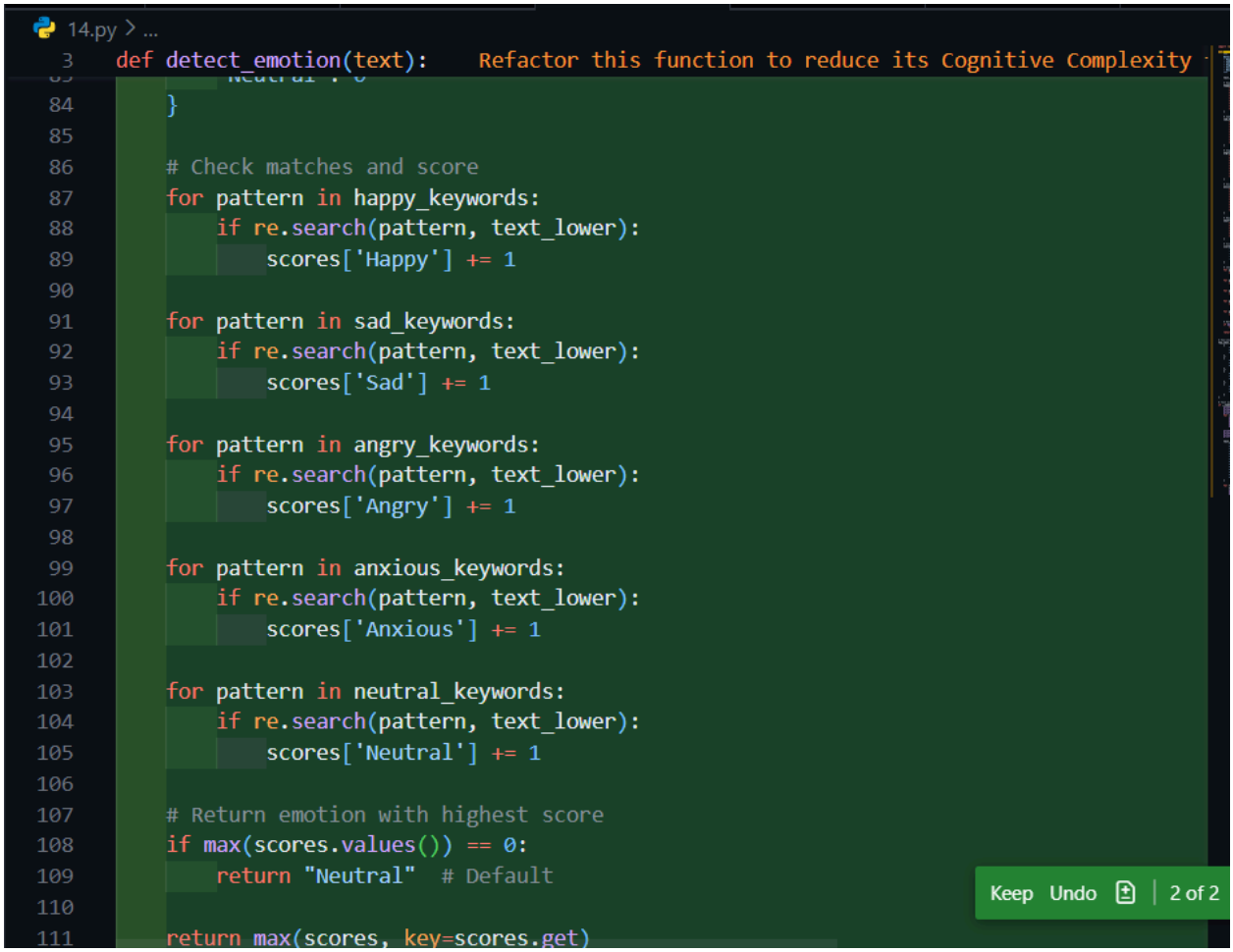
Few shot prompt

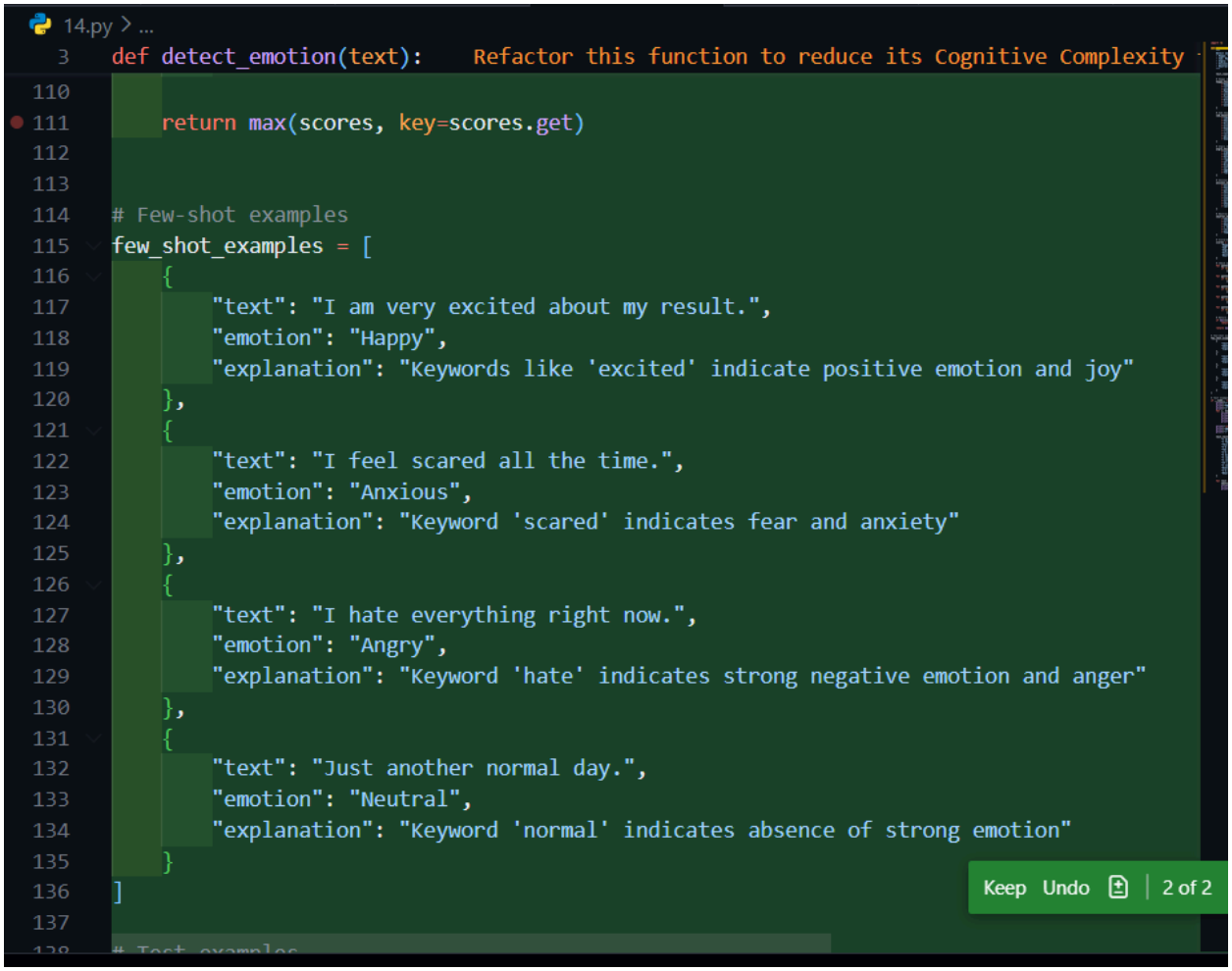
Examples: I feel scared all the time.' → Anxious 'I hate everything right now.' → Angry . 'Just another normal day.' → Neutral Classify Code generated


14.py > ...
import re
Keep Undo
def detect_emotion(text): Refactor this function to reduce its Cognitive Complexity
"""
Detect emotion in text for mental-health chatbot:
- Happy: Positive emotions, joy, excitement
- Sad: Negative emotions, depression, grief, disappointment
- Angry: Frustration, rage, irritation
- Anxious: Fear, worry, stress, nervousness
- Neutral: No strong emotion
"""
text_lower = text.lower()
# Happy keywords - positive emotions, joy, excitement
happy_keywords = [
r'happy|joyful|excited|thrilled|delighted',
r'love|amazing|wonderful|fantastic|excellent',
r'good news|great|awesome|perfect',
r'cheerful|laughing|smiling|laughter|smile',
r'proud|proud of|achievement|success',
r'looking forward|can\'t wait|anticipating',
r'blessed|grateful|thankful|blessed',
r'celebrate|celebrating|celebration',
r'wonderful|marvelous|magnificent'
]
Sad keywords - negative emotions, grief, disappointment
Keep Undo 2 of 2
sad_keywords = [


14.py > ...
def detect_emotion(text): Refactor this function to reduce its Cognitive Complexity
r'heartbroken|devastated|grief|mourning',
r'disappointed|letdown|let down',
r'alone|lonely|isolated|empty',
r'hopeless|desperate|despair',
r'crying|tears|weeping|sobbing',
r'lost|missing|gone',
r'pain|hurt|ache|broken',
r'downhearted|downcast|melancholy'
]
# Angry keywords - frustration, rage, irritation
angry_keywords = [
r'angry|furious|enraged|livid|seething',
r'mad|irritated|annoyed|frustrated',
r'hate|despise|can\'t stand',
r'rage|raging|explosive|blow up',
r'yelling|shouting|screaming',
r'disgusted|repulsed|sick of',
r'outraged|offended|insulted',
r'aggressive|hostile|confrontational',
r'fed up|sick and tired'
]
# Anxious keywords - fear, worry, stress, nervousness
anxious_keywords = [
r'anxious|anxious|worried|nervous|worried',
r'fear|scared|terrified|frightened|horrified',
r'panic|panicking|panic attack',
Keep Undo 2 of 2


14.py > ...
3 def detect_emotion(text): Refactor this function to reduce its Cognitive Complexity
r'fear|scared|terrified|frightened|horrified',
r'panic|panicking|panic attack',
r'stress|stressed|stressed out|overwhelming',
r'uncertain|unsure|doubtful',
r'uneasy|uncomfortable|restless',
r'worried sick|tense|tension',
r'what if|what happens if',
r'can\'t sleep|insomnia|sleepless'
]
# Neutral keywords - no strong emotion or mixed
neutral_keywords = [
r'okay|fine|alright|so-so|neither',
r'normal|regular|usual|typical',
r'maybe|perhaps|possibly',
r'i don\'t know|not sure|uncertain',
r'could be|might be|seems',
r'happening|going on|situation'
]
# Score-based detection
scores = {
'Happy': 0,
'Sad': 0,
'Angry': 0,
'Anxious': 0,
'Neutral': 0
}
Keep Undo 2 of 2


14.py > ...
def detect_emotion(text):
Refactor this function to reduce its Cognitive Complexity
}
# Check matches and score
for pattern in happy_keywords:
if re.search(pattern, text_lower):
scores['Happy'] += 1
for pattern in sad_keywords:
if re.search(pattern, text_lower):
scores['Sad'] += 1
for pattern in angry_keywords:
if re.search(pattern, text_lower):
scores['Angry'] += 1
for pattern in anxious_keywords:
if re.search(pattern, text_lower):
scores['Anxious'] += 1
for pattern in neutral_keywords:
if re.search(pattern, text_lower):
scores['Neutral'] += 1
# Return emotion with highest score
if max(scores.values()) == 0:
return "Neutral" # Default
return max(scores, key=scores.get)
Keep
Undo
2 of 2


14.py > ...
3 def detect_emotion(text): Refactor this function to reduce its Cognitive Complexity
110
111 return max(scores, key=scores.get)
112
113
114 # Few-shot examples
115 few_shot_examples = [
116 {
117 "text": "I am very excited about my result.",
118 "emotion": "Happy",
119 "explanation": "Keywords like 'excited' indicate positive emotion and joy"
120 },
121 {
122 "text": "I feel scared all the time.",
123 "emotion": "Anxious",
124 "explanation": "Keyword 'scared' indicates fear and anxiety"
125 },
126 {
127 "text": "I hate everything right now.",
128 "emotion": "Angry",
129 "explanation": "Keyword 'hate' indicates strong negative emotion and anger"
130 },
131 {
132 "text": "Just another normal day.",
133 "emotion": "Neutral",
134 "explanation": "Keyword 'normal' indicates absence of strong emotion"
135 }
136]
137
Keep Undo 2 of 2

14.py > ...

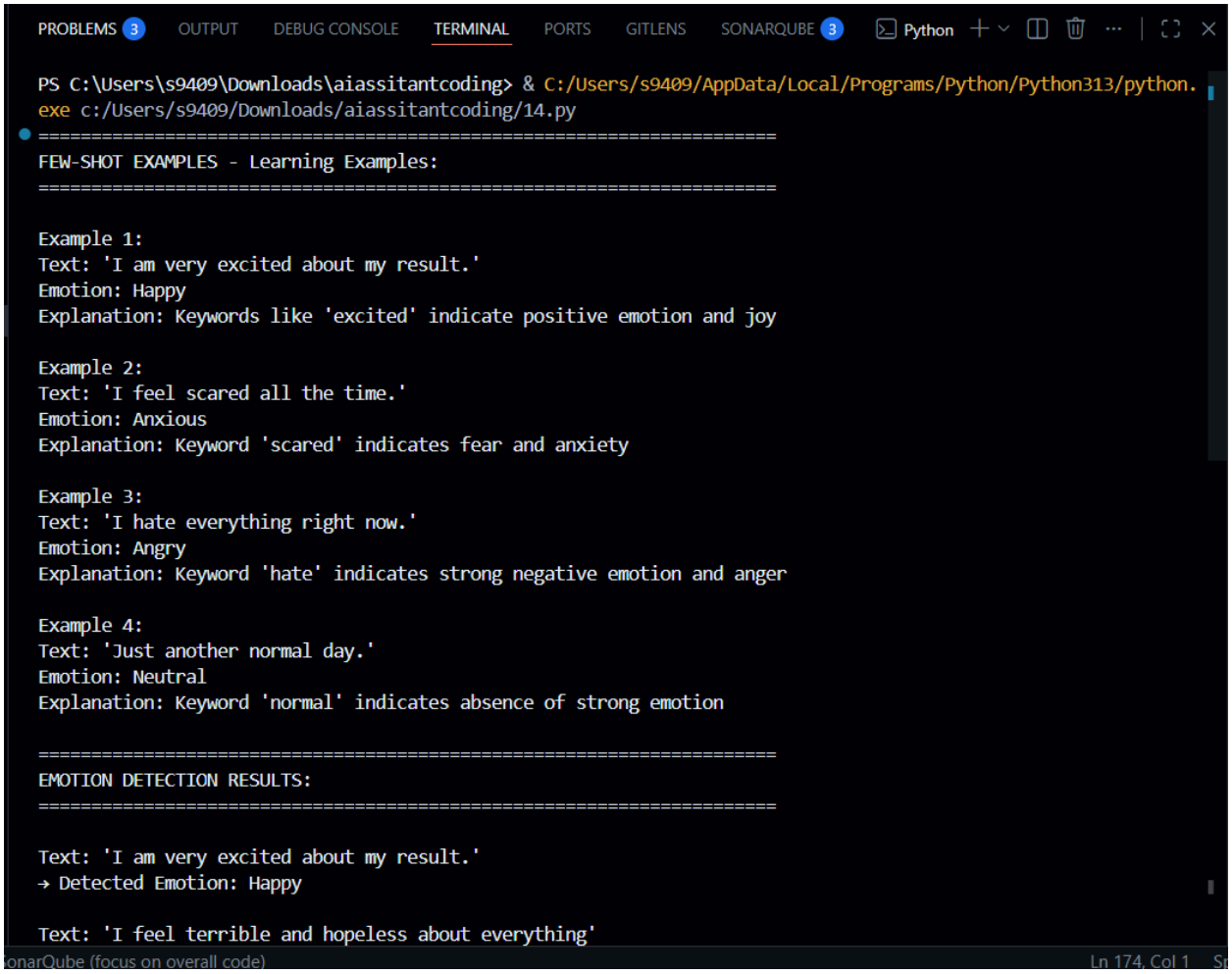
```python
# Test examples
if __name__ == "__main__":
    print("=" * 70)
    print("FEW-SHOT EXAMPLES - Learning Examples:")
    print("=" * 70)
    for i, example in enumerate(few_shot_examples, 1):
        print(f"\nExample {i}:")
        print(f"Text: '{example['text']}'")
        print(f"Emotion: {example['emotion']}")
        print(f"Explanation: {example['explanation']}")

    print("\n" + "=" * 70)
    print("EMOTION DETECTION RESULTS:")
    print("=" * 70)

    test_texts = [
        "I am very excited about my result.",
        "I feel terrible and hopeless about everything",
        "This makes me so angry and frustrated!",
        "I'm worried and scared about what might happen",
        "The weather is nice today",
        "I'm thrilled with my achievement!",
        "I miss my loved one, it's so painful",
        "I can't stand this anymore, I'm furious!",
        "I can't sleep due to stress and worry",
        "It's just another day",
        "I love this moment, I'm so happy!",
        "Everything feels empty and broken",
        "I'm disgusted by this behavior",
```

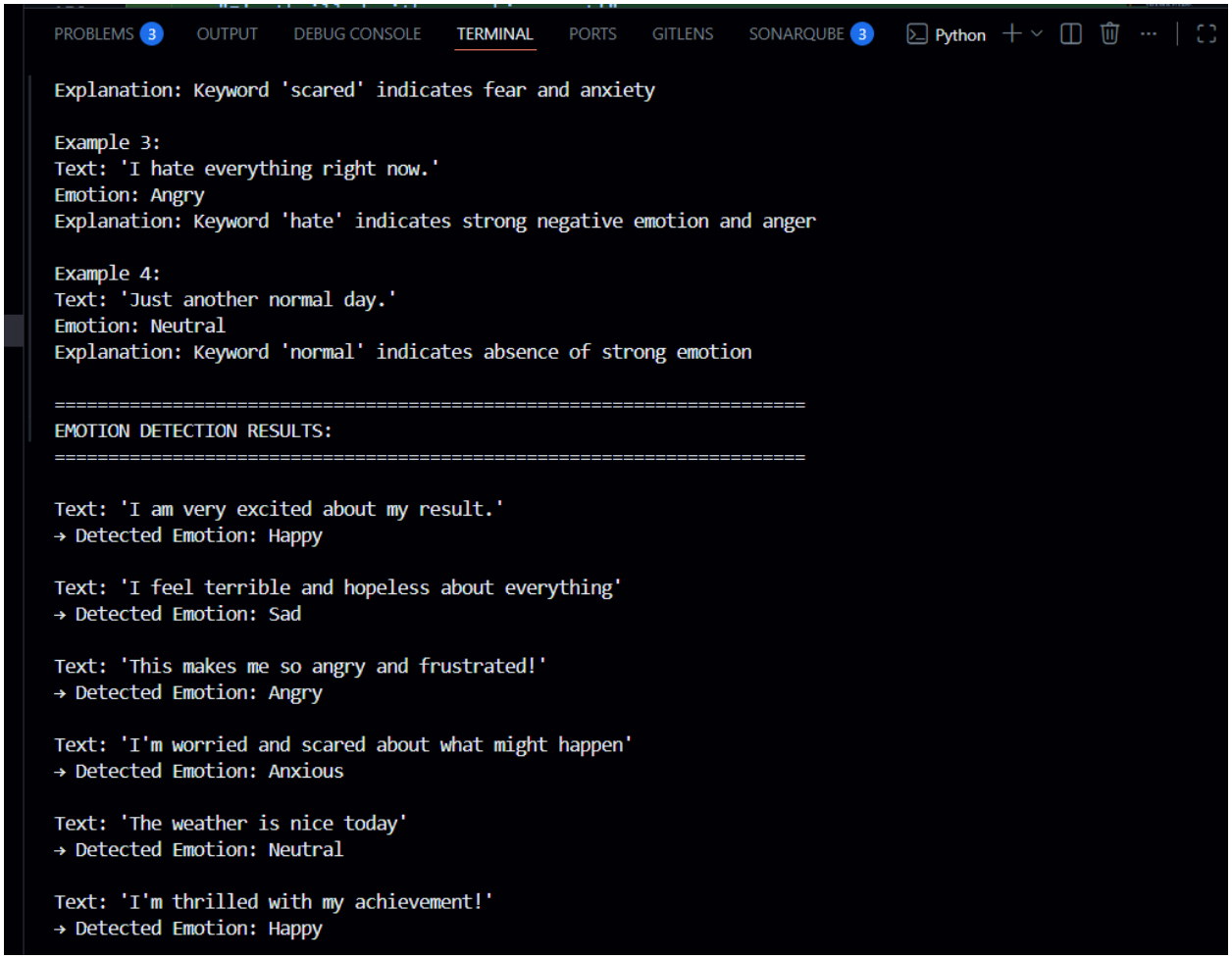
Keep Undo 2 of 2

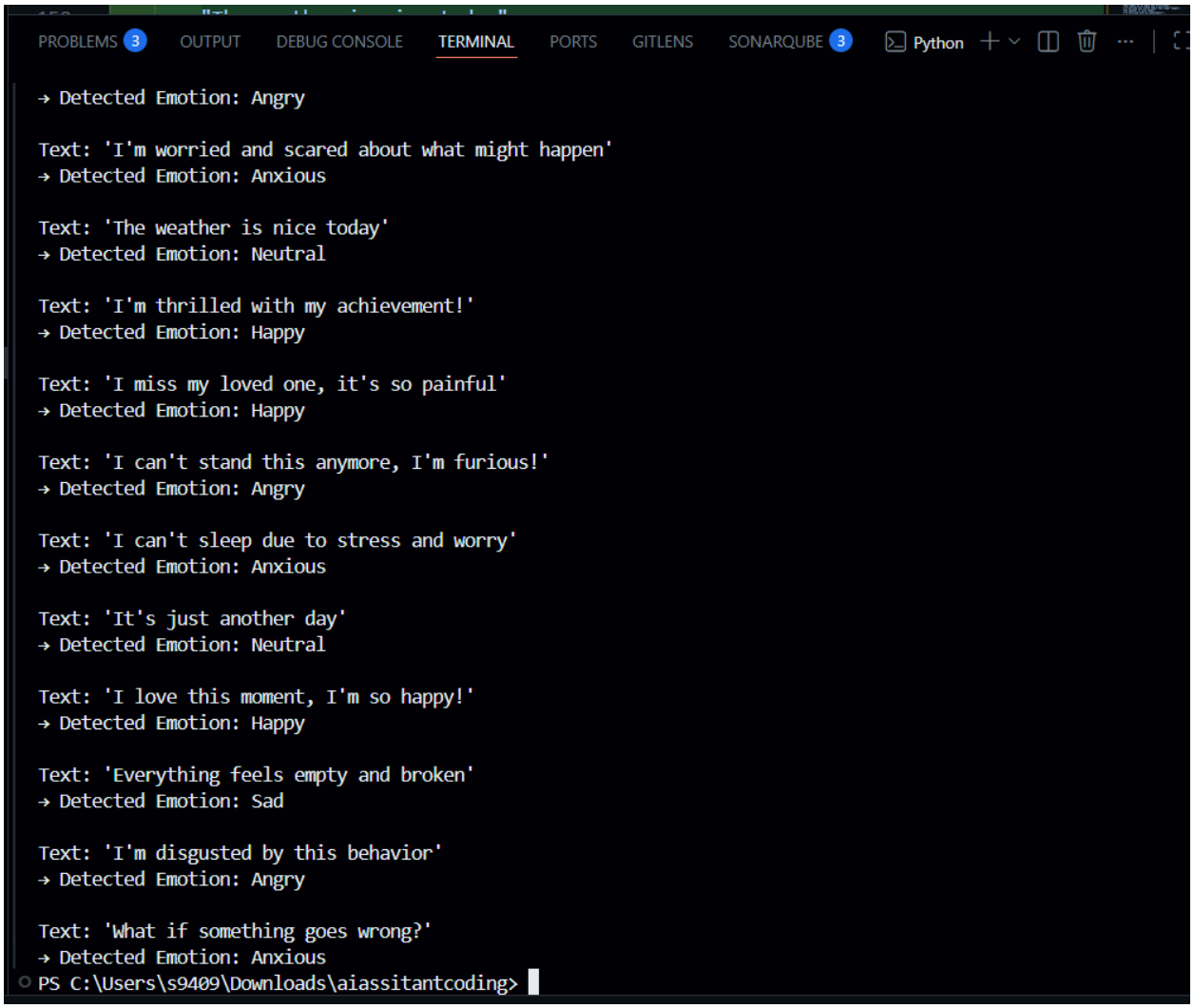
14.py > ...

```python
        "This makes me so angry and frustrated!",
        "I'm worried and scared about what might happen",
        "The weather is nice today",
        "I'm thrilled with my achievement!",
        "I miss my loved one, it's so painful",
        "I can't stand this anymore, I'm furious!",
        "I can't sleep due to stress and worry",
        "It's just another day",
        "I love this moment, I'm so happy!",
        "Everything feels empty and broken",
        "I'm disgusted by this behavior",
        "What if something goes wrong?"
    ]

    for text in test_texts:
        emotion = detect_emotion(text)
        print(f"\nText: '{text}'")
        print(f"→ Detected Emotion: {emotion}")
```

Output:


PROBLEMS 3
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
GITLENS
SONARQUBE 3
Python
PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs/Python/Python313/python.exe c:/Users/s9409/Downloads/aiassitantcoding/14.py
FEW-SHOT EXAMPLES - Learning Examples:
Example 1:
Text: 'I am very excited about my result.'
Emotion: Happy
Explanation: Keywords like 'excited' indicate positive emotion and joy
Example 2:
Text: 'I feel scared all the time.'
Emotion: Anxious
Explanation: Keyword 'scared' indicates fear and anxiety
Example 3:
Text: 'I hate everything right now.'
Emotion: Angry
Explanation: Keyword 'hate' indicates strong negative emotion and anger
Example 4:
Text: 'Just another normal day.'
Emotion: Neutral
Explanation: Keyword 'normal' indicates absence of strong emotion
EMOTION DETECTION RESULTS:
Text: 'I am very excited about my result.'
→ Detected Emotion: Happy
Text: 'I feel terrible and hopeless about everything'
SonarQube (focus on overall code)
Ln 174, Col 1


PROBLEMS 3
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
GITLENS
SONARQUBE 3
Python
Explanation: Keyword 'scared' indicates fear and anxiety
Example 3:
Text: 'I hate everything right now.'
Emotion: Angry
Explanation: Keyword 'hate' indicates strong negative emotion and anger
Example 4:
Text: 'Just another normal day.'
Emotion: Neutral
Explanation: Keyword 'normal' indicates absence of strong emotion
EMOTION DETECTION RESULTS:
Text: 'I am very excited about my result.'
→ Detected Emotion: Happy
Text: 'I feel terrible and hopeless about everything'
→ Detected Emotion: Sad
Text: 'This makes me so angry and frustrated!'
→ Detected Emotion: Angry
Text: 'I'm worried and scared about what might happen'
→ Detected Emotion: Anxious
Text: 'The weather is nice today'
→ Detected Emotion: Neutral
Text: 'I'm thrilled with my achievement!'
→ Detected Emotion: Happy



Justification:

Few-Shot Learning Framework: Includes 4 labeled examples covering all 5 emotions (Happy, Anxious, Angry, Neutral, and implied Sad) demonstrating the pattern of how to classify emotions without requiring ML training or large datasets

Diverse Example Coverage: Examples span different emotions with clear keyword indicators - "excited" for Happy, "scared" for Anxious, "hate" for Angry, "normal" for Neutral - teaching the system to recognize emotion indicators across all categories

Explanation-Based Learning: Each example includes explanations showing which keywords triggered the emotion classification,

enabling transparent understanding of how the system makes decisions and improving pattern recognition

Comprehensive Keyword Patterns: 9+ patterns per emotion capture variations in expression (e.g., "scared", "terrified", "frightened" for Anxious) ensuring the system recognizes different ways users express the same emotion

Score-Based Disambiguation: Scoring mechanism handles complex emotional states (e.g., "I'm happy but scared") by counting keyword matches and returning the dominant emotion, critical for accurate mental-health support where users may experience mixed emotions