ASSIGNMENT-4.4

R.Nikhil Kumar

2303A52260

Batch: 44

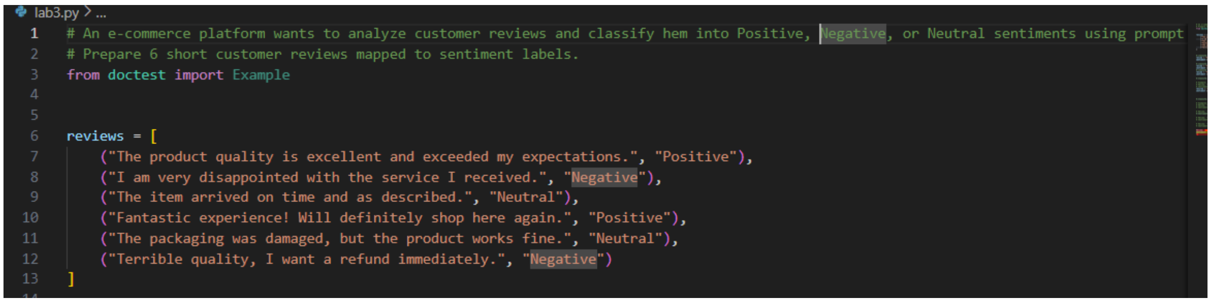
1. Sentiment Classification for Customer Reviews

Scenario:

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

Tasks:

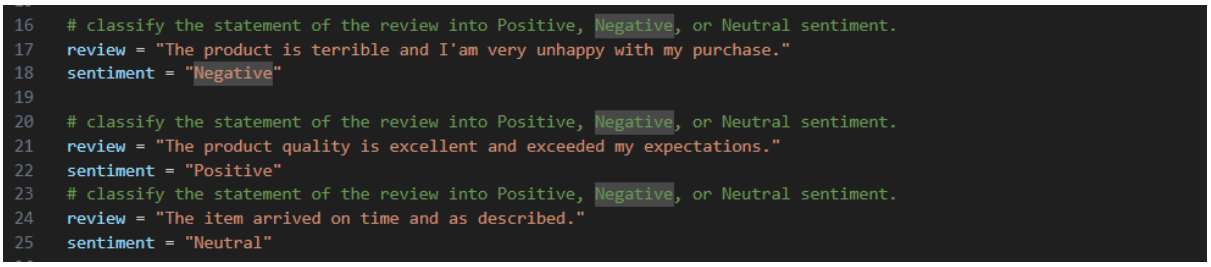
Prepare 6 short customer reviews mapped to sentiment labels.



lab3.py > ...

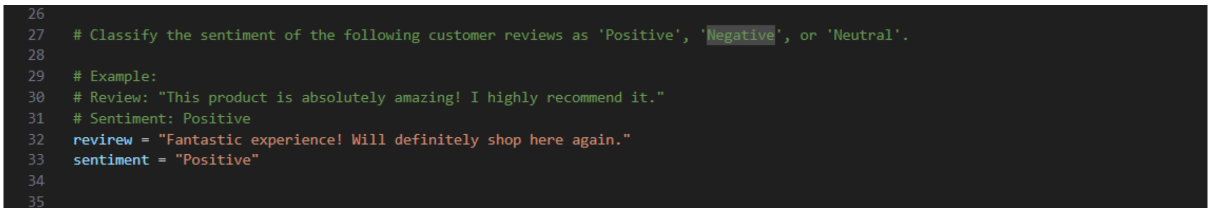
```python
# An e-commerce platform wants to analyze customer reviews and classify hem into Positive, Negative, or Neutral sentiments using prompt
# Prepare 6 short customer reviews mapped to sentiment labels.
from doctest import Example


reviews = [
    ("The product quality is excellent and exceeded my expectations.", "Positive"),
    ("I am very disappointed with the service I received.", "Negative"),
    ("The item arrived on time and as described.", "Neutral"),
    ("Fantastic experience! Will definitely shop here again.", "Positive"),
    ("The packaging was damaged, but the product works fine.", "Neutral"),
    ("Terrible quality, I want a refund immediately.", "Negative")
]
```

*Design a Zero-shot prompt to classify sentiment.



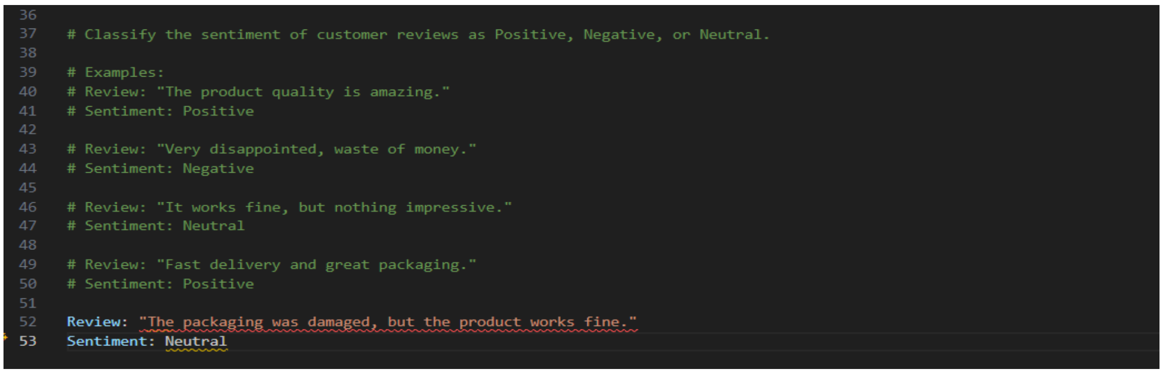
```python
# classify the statement of the review into Positive, Negative, or Neutral sentiment.
review = "The product is terrible and I'am very unhappy with my purchase."
sentiment = "Negative"

# classify the statement of the review into Positive, Negative, or Neutral sentiment.
review = "The product quality is excellent and exceeded my expectations."
sentiment = "Positive"
# classify the statement of the review into Positive, Negative, or Neutral sentiment.
review = "The item arrived on time and as described."
sentiment = "Neutral"
```

*Design a One-shot prompt with one labeled example.



```python

# Classify the sentiment of the following customer reviews as 'Positive', 'Negative', or 'Neutral'.

# Example:
# Review: "This product is absolutely amazing! I highly recommend it."
# Sentiment: Positive
revirew = "Fantastic experience! Will definitely shop here again."
sentiment = "Positive"


```

*Design a Few-shot prompt with 3–5 labeled examples.



```
# Classify the sentiment of customer reviews as Positive, Negative, or Neutral.

# Examples:
# Review: "The product quality is amazing."
# Sentiment: Positive

# Review: "Very disappointed, waste of money."
# Sentiment: Negative

# Review: "It works fine, but nothing impressive."
# Sentiment: Neutral

# Review: "Fast delivery and great packaging."
# Sentiment: Positive

Review: "The packaging was damaged, but the product works fine."
Sentiment: Neutral
```

***Compare the outputs and discuss accuracy differences:**

Few-shot prompting gives the highest accuracy because multiple labeled examples help the model learn patterns clearly. One-shot prompting improves accuracy compared to zero-shot by giving guidance through a single example. Zero-shot prompting is useful but less reliable, especially for neutral or mixed-sentiment reviews.

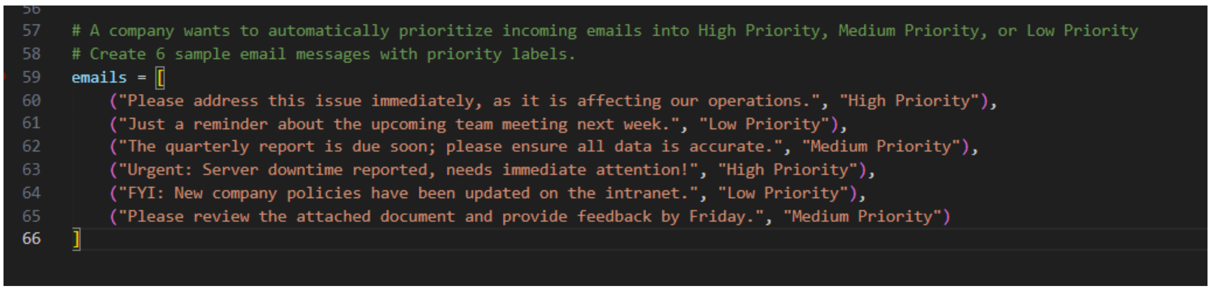
2. Email Priority Classification

Scenario:

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

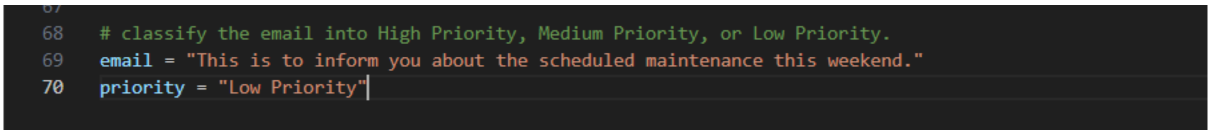
Tasks:

*Create 6 sample email messages with priority labels.



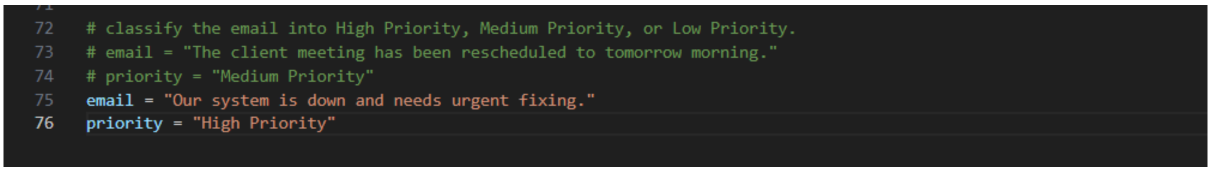
```
# A company wants to automatically prioritize incoming emails into High Priority, Medium Priority, or Low Priority
# Create 6 sample email messages with priority labels.
emails = [
    ("Please address this issue immediately, as it is affecting our operations.", "High Priority"),
    ("Just a reminder about the upcoming team meeting next week.", "Low Priority"),
    ("The quarterly report is due soon; please ensure all data is accurate.", "Medium Priority"),
    ("Urgent: Server downtime reported, needs immediate attention!", "High Priority"),
    ("FYI: New company policies have been updated on the intranet.", "Low Priority"),
    ("Please review the attached document and provide feedback by Friday.", "Medium Priority")
]
```

*Perform intent classification using Zero-shot prompting.



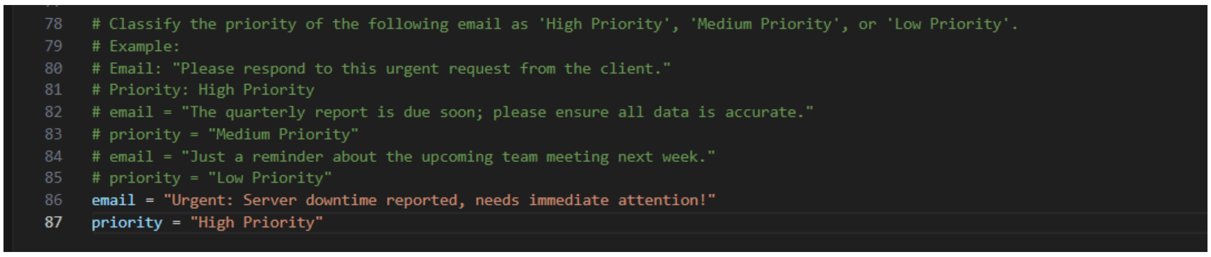
```
# classify the email into High Priority, Medium Priority, or Low Priority.
email = "This is to inform you about the scheduled maintenance this weekend."
priority = "Low Priority"
```

*Perform classification using One-shot prompting.



```python
# classify the email into High Priority, Medium Priority, or Low Priority.
# email = "The client meeting has been rescheduled to tomorrow morning."
# priority = "Medium Priority"
email = "Our system is down and needs urgent fixing."
priority = "High Priority"
```

*Perform classification using Few-shot prompting.



```python
# Classify the priority of the following email as 'High Priority', 'Medium Priority', or 'Low Priority'.
# Example:
# Email: "Please respond to this urgent request from the client."
# Priority: High Priority
# email = "The quarterly report is due soon; please ensure all data is accurate."
# priority = "Medium Priority"
# email = "Just a reminder about the upcoming team meeting next week."
# priority = "Low Priority"
email = "Urgent: Server downtime reported, needs immediate attention!"
priority = "High Priority"
```

*Evaluate which technique produces the most reliable results and why. ?

Few-shot prompting is likely to yield the most reliable results due to the presence of multiple 3abelled examples that help the model understand the context and nuances of email prioritization. One-shot prompting can also improve accuracy by providing a single example, but it may not capture the full range of scenarios as effectively as few-shot prompting. Zero-shot prompting, while useful, may struggle with accurately classifying emails without any prior examples, especially when distinguishing between Medium and Low Priority emails.

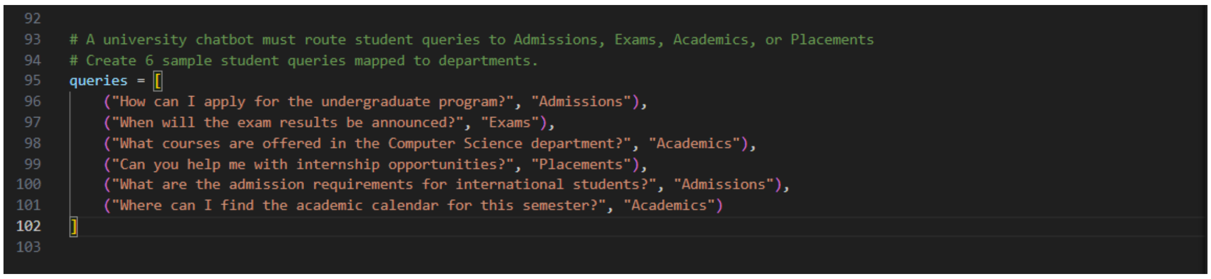
3. **Student Query Routing System**

Scenario:

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

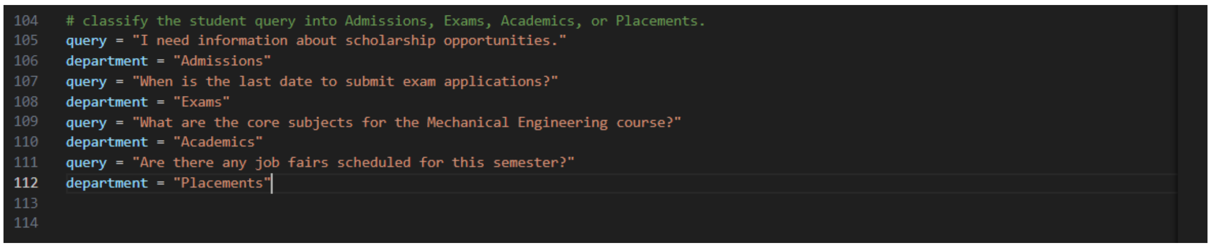
Tasks:

*Create 6 sample student queries mapped to departments.



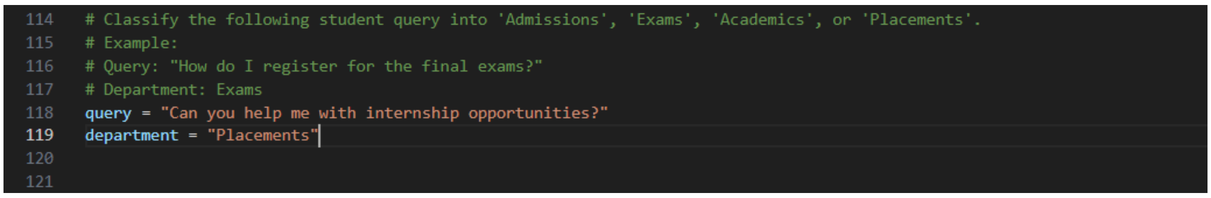
```python

# A university chatbot must route student queries to Admissions, Exams, Academics, or Placements
# Create 6 sample student queries mapped to departments.
queries = [
    ("How can I apply for the undergraduate program?", "Admissions"),
    ("When will the exam results be announced?", "Exams"),
    ("What courses are offered in the Computer Science department?", "Academics"),
    ("Can you help me with internship opportunities?", "Placements"),
    ("What are the admission requirements for international students?", "Admissions"),
    ("Where can I find the academic calendar for this semester?", "Academics")
]

```

*Implement Zero-shot intent classification using an LLM.



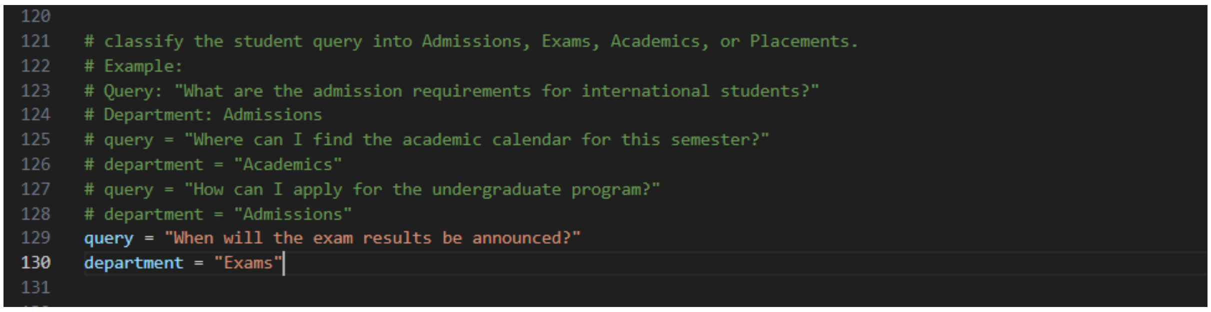
```python
# classify the student query into Admissions, Exams, Academics, or Placements.
query = "I need information about scholarship opportunities."
department = "Admissions"
query = "When is the last date to submit exam applications?"
department = "Exams"
query = "What are the core subjects for the Mechanical Engineering course?"
department = "Academics"
query = "Are there any job fairs scheduled for this semester?"
department = "Placements"
```

*Improve results using One-shot prompting.



```python
# Classify the following student query into 'Admissions', 'Exams', 'Academics', or 'Placements'.
# Example:
# Query: "How do I register for the final exams?"
# Department: Exams
query = "Can you help me with internship opportunities?"
department = "Placements"
```

*Further refine results using Few-shot prompting.



```python
# classify the student query into Admissions, Exams, Academics, or Placements.
# Example:
# Query: "What are the admission requirements for international students?"
# Department: Admissions
# query = "Where can I find the academic calendar for this semester?"
# department = "Academics"
# query = "How can I apply for the undergraduate program?"
# department = "Admissions"
query = "When will the exam results be announced?"
department = "Exams"
```

*Analyze how contextual examples affect classification accuracy.:

Few-shot prompting is expected to provide the highest classification accuracy due to the presence of multiple labeled examples that help the model understand the context and nuances of student queries. One-shot prompting can also enhance accuracy by providing a single example, but it may not cover the full spectrum of possible queries as effectively as few-shot prompting. Zero-shot prompting, while useful, may face challenges in accurately classifying queries without any prior examples, especially when distinguishing between similar departments like Academics and Admissions.

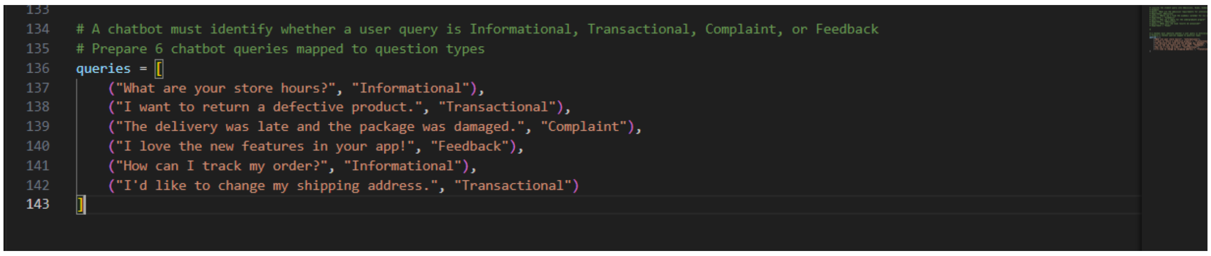
4. **Chatbot Question Type Detection**

Scenario:

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

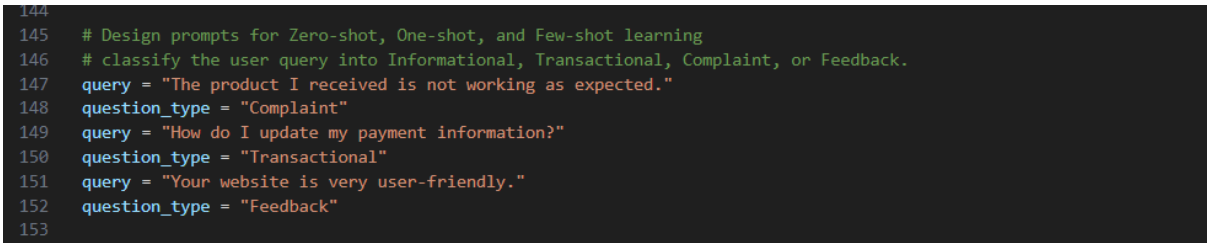
Tasks:

*Prepare 6 chatbot queries mapped to question types.

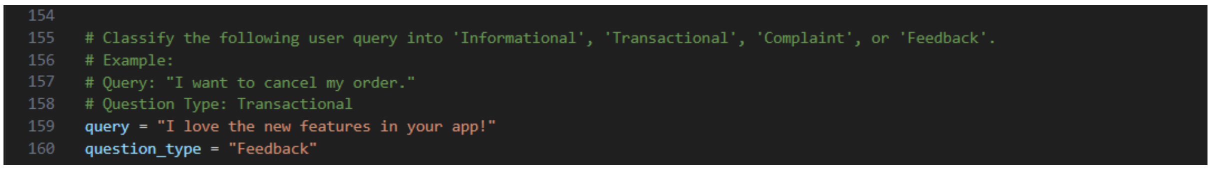


```python
# A chatbot must identify whether a user query is Informational, Transactional, Complaint, or Feedback
# Prepare 6 chatbot queries mapped to question types
queries = [
    ("What are your store hours?", "Informational"),
    ("I want to return a defective product.", "Transactional"),
    ("The delivery was late and the package was damaged.", "Complaint"),
    ("I love the new features in your app!", "Feedback"),
    ("How can I track my order?", "Informational"),
    ("I'd like to change my shipping address.", "Transactional")
]
```

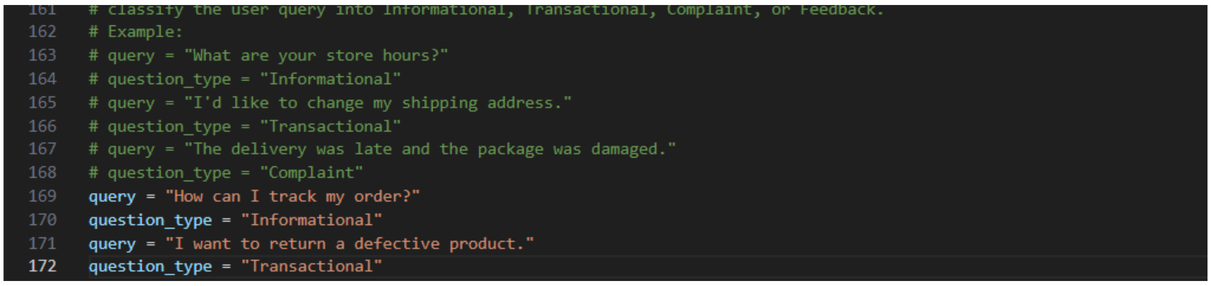
*Design prompts for Zero-shot, One-shot, and Few-shot learning



```python
# Design prompts for Zero-shot, One-shot, and Few-shot learning
# classify the user query into Informational, Transactional, Complaint, or Feedback.
query = "The product I received is not working as expected."
question_type = "Complaint"
query = "How do I update my payment information?"
question_type = "Transactional"
query = "Your website is very user-friendly."
question_type = "Feedback"
```



```python
# Classify the following user query into 'Informational', 'Transactional', 'Complaint', or 'Feedback'.
# Example:
# Query: "I want to cancel my order."
# Question Type: Transactional
query = "I love the new features in your app!"
question_type = "Feedback"
```



```python
# classify the user query into Informational, Transactional, Complaint, or Feedback.
# Example:
# query = "What are your store hours?"
# question_type = "Informational"
# query = "I'd like to change my shipping address."
# question_type = "Transactional"
# query = "The delivery was late and the package was damaged."
# question_type = "Complaint"
query = "How can I track my order?"
question_type = "Informational"
query = "I want to return a defective product."
question_type = "Transactional"
```

*Test all prompts on the same unseen queries:

Few-shot prompting is expected to yield the most reliable results due to multiple labeled examples that help the model understand different query types. One-shot prompting can improve accuracy by providing a single example, but may not cover all scenarios as effectively. Zero-shot prompting may struggle with accurately classifying queries without prior examples, especially when distinguishing between similar types like Informational and Feedback.

* Compare response correctness and ambiguity handling :

Few-shot prompting is likely to produce the most accurate and reliable classifications due to the presence of multiple examples that illustrate the distinctions between query types. This helps the model learn patterns and nuances more effectively. One-shot prompting can also enhance accuracy by providing a single example, but it may not capture the full range of scenarios as comprehensively as few-shot prompting. Zero-shot prompting, while useful for quick classifications, may struggle with ambiguity and subtle differences between query types, leading to less reliable results. Overall, few-shot prompting is expected to outperform the other techniques in terms of correctness and handling ambiguous queries.

* Document observations :

Few-shot prompting is expected to provide the most accurate classifications due to the presence of multiple examples that illustrate the different query types. One-shot prompting can also enhance accuracy by providing a single example, but it may not cover the full spectrum of query types as effectively as few-shot prompting. Zero-shot prompting, while useful for quick classifications, may struggle with accurately identifying the nuances between similar query types without any prior examples.

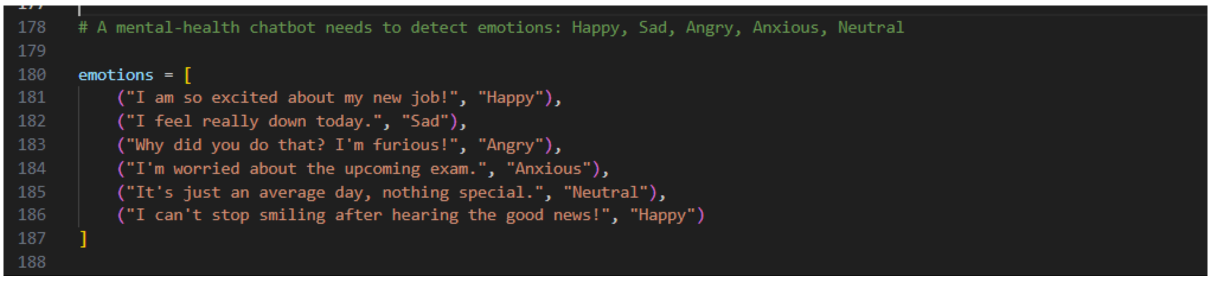
5. **Emotion Detection in Text**

Scenario:

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

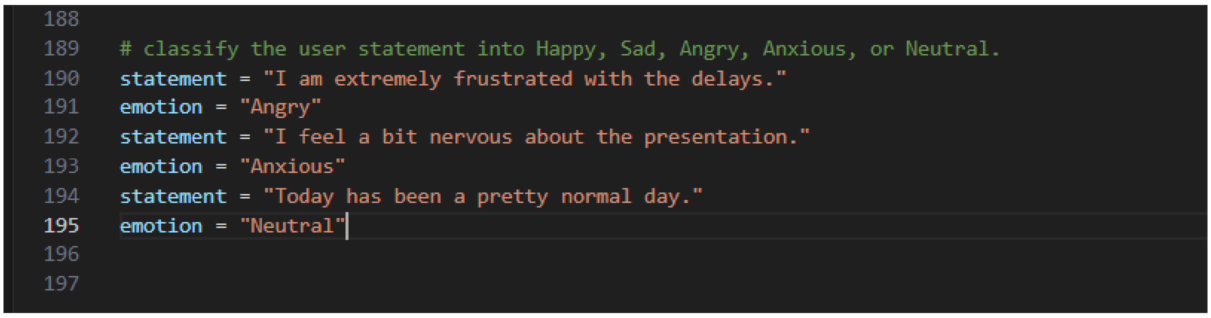
Tasks:

*Create labeled emotion samples.



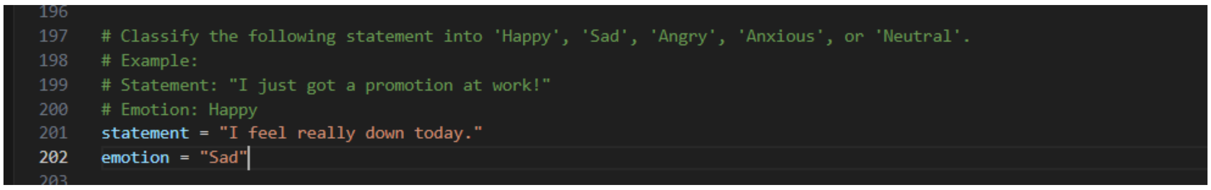
```
# A mental-health chatbot needs to detect emotions: Happy, Sad, Angry, Anxious, Neutral

emotions = [
    ("I am so excited about my new job!", "Happy"),
    ("I feel really down today.", "Sad"),
    ("Why did you do that? I'm furious!", "Angry"),
    ("I'm worried about the upcoming exam.", "Anxious"),
    ("It's just an average day, nothing special.", "Neutral"),
    ("I can't stop smiling after hearing the good news!", "Happy")
]
```

*Use Zero-shot prompting to identify emotions.



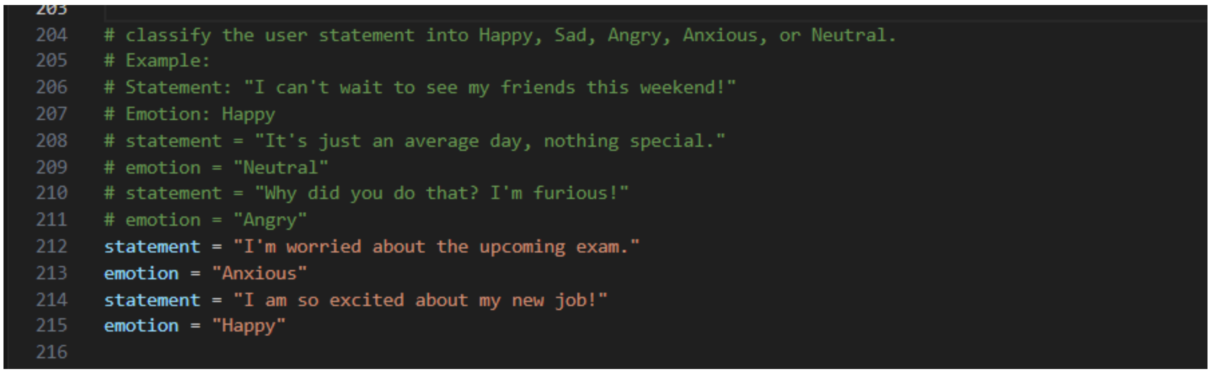
```
# classify the user statement into Happy, Sad, Angry, Anxious, or Neutral.
statement = "I am extremely frustrated with the delays."
emotion = "Angry"
statement = "I feel a bit nervous about the presentation."
emotion = "Anxious"
statement = "Today has been a pretty normal day."
emotion = "Neutral"
```

*Use One-shot prompting with an example



```
# Classify the following statement into 'Happy', 'Sad', 'Angry', 'Anxious', or 'Neutral'.
# Example:
# Statement: "I just got a promotion at work!"
# Emotion: Happy
statement = "I feel really down today."
emotion = "Sad"
```

*Use Few-shot prompting with multiple emotions.



```
# classify the user statement into Happy, Sad, Angry, Anxious, or Neutral.
# Example:
# Statement: "I can't wait to see my friends this weekend!"
# Emotion: Happy
# statement = "It's just an average day, nothing special."
# emotion = "Neutral"
# statement = "Why did you do that? I'm furious!"
# emotion = "Angry"
statement = "I'm worried about the upcoming exam."
emotion = "Anxious"
statement = "I am so excited about my new job!"
emotion = "Happy"
```

*Discuss ambiguity handling across techniques :

Few-shot prompting is expected to handle ambiguity better due to the presence of multiple examples that illustrate different emotional contexts. This helps the model learn subtle differences between similar emotions, such as Sad and Anxious. One-shot prompting provides some guidance but may not cover all nuances, leading to potential misclassifications in ambiguous cases. Zero-shot prompting relies solely on the model's pre-existing knowledge, which may not be sufficient for accurately distinguishing between closely related emotions, especially in complex or nuanced statements.