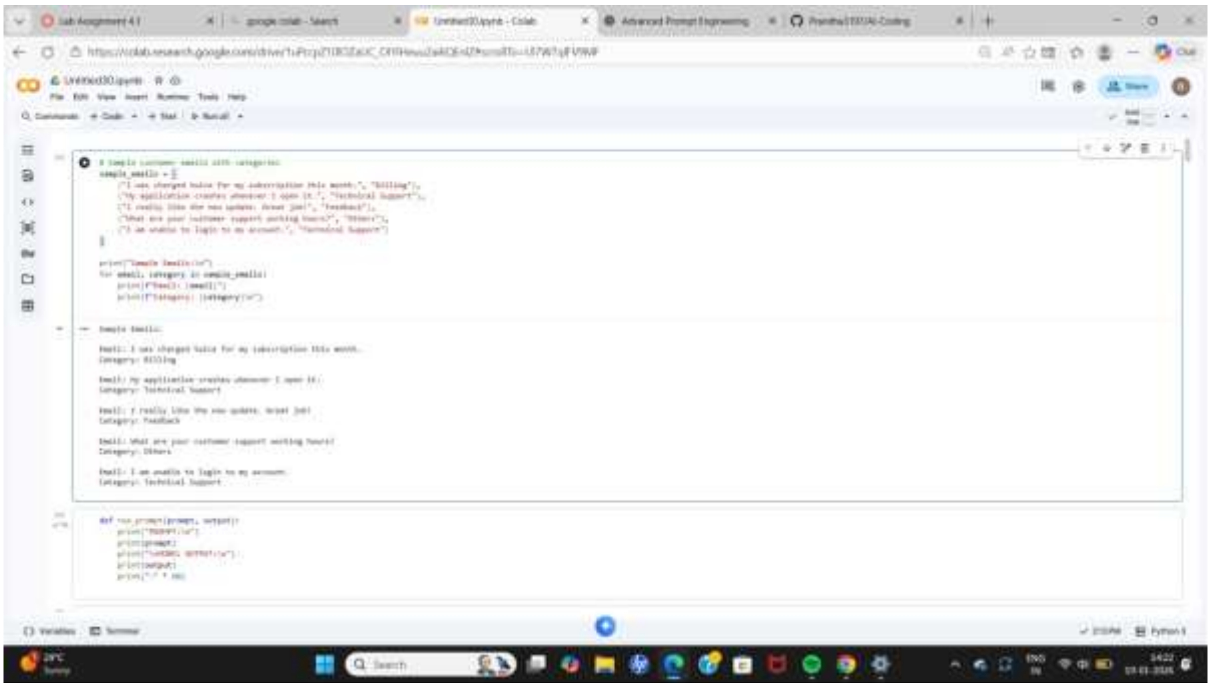
LAB ASSIGNMENT 4.1

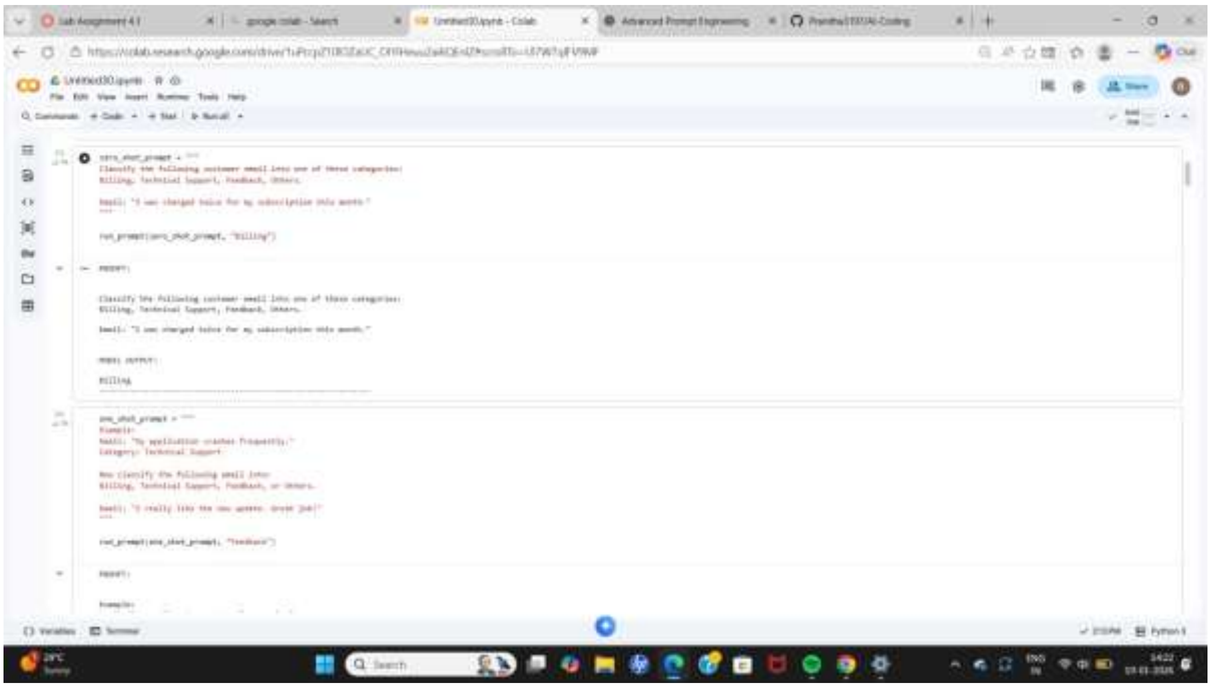
NAME : M.Geethika 2303A52276 SUBJECT : AI ASSISTED CODING



```
# Sample customer emails with categories
sample_emails = [
    ("I was charged twice for my subscription this month.", "Billing"),
    ("My application crashes whenever I open it.", "Technical Support"),
    ("I really like the new update. Great job!", "Feedback"),
    ("What are your customer support working hours?", "Others"),
    ("I am unable to login to my account.", "Technical Support")
]

print("Sample Emails:\n")
for email, category in sample_emails:
    print(f"Email: {email}")
    print(f"Category: {category}\n")
```

```
Sample Emails:

Email: I was charged twice for my subscription this month.
Category: Billing

Email: My application crashes whenever I open it.
Category: Technical Support

Email: I really like the new update. Great job!
Category: Feedback

Email: What are your customer support working hours?
Category: Others

Email: I am unable to login to my account.
Category: Technical Support
```

```
def run_prompt(prompt, output):
    print("PROMPT:\n")
    print(prompt)
    print("\nMODEL OUTPUT:\n")
    print(output)
    print("-" * 80)
```

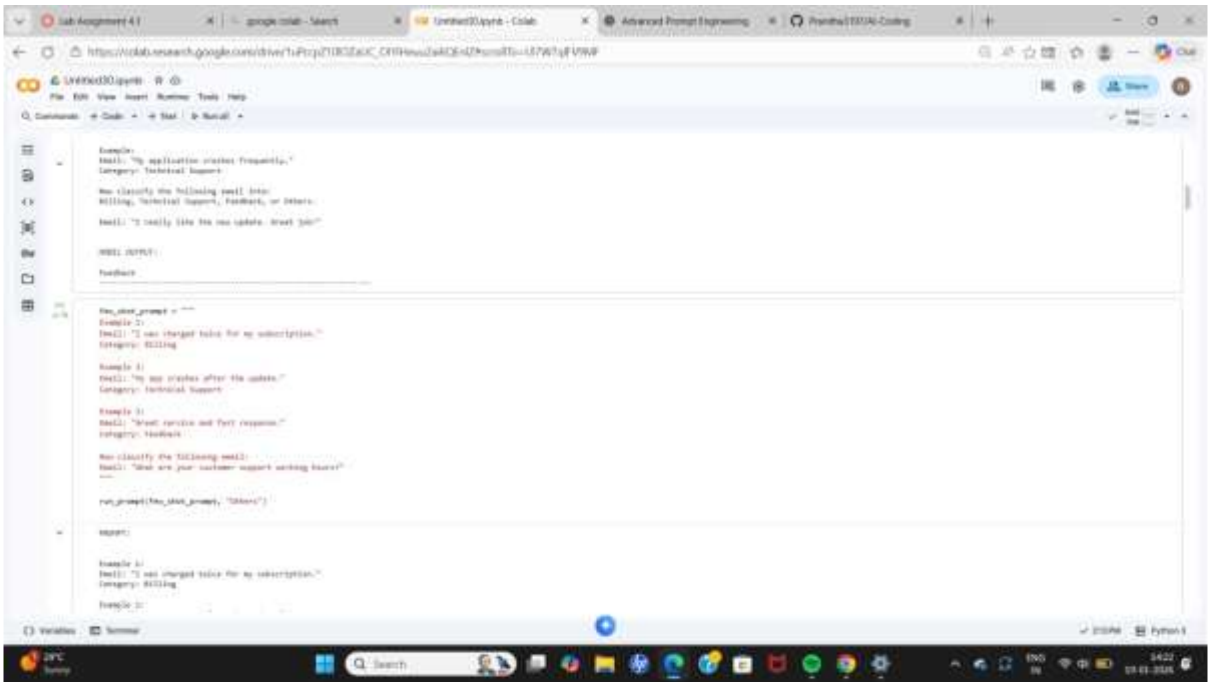


```
zero_shot_prompt = """
Classify the following customer email into one of these categories:
Billing, Technical Support, Feedback, Others.

Email: "I was charged twice for my subscription this month."
"""

run_prompt(zero_shot_prompt, "Billing")
```

```
PROMPT:


Classify the following customer email into one of these categories:
Billing, Technical Support, Feedback, Others.

Email: "I was charged twice for my subscription this month."


MODEL OUTPUT:

Billing
--------------------------------------------------------------------------------
```

```
one_shot_prompt = """
Example:
Email: "My application crashes frequently."
Category: Technical Support

Now classify the following email into:
Billing, Technical Support, Feedback, or Others.

Email: "I really like the new update. Great job!"
"""

run_prompt(one_shot_prompt, "Feedback")
```

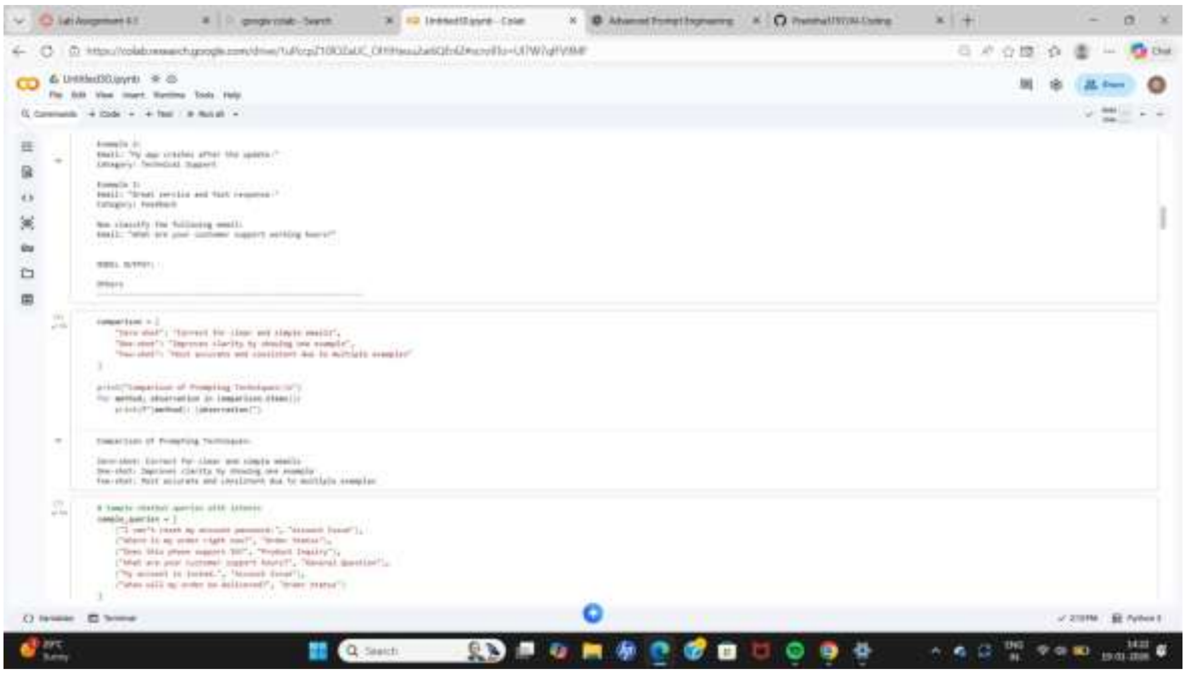
```
PROMPT:


Example:
```



```
Example:
Email: "My application crashes frequently."
Category: Technical Support

Now classify the following email into:
Billing, Technical Support, Feedback, or Others.

Email: "I really like the new update. Great job!"


MODEL OUTPUT:

Feedback
```

```
few_shot_prompt = """
Example 1:
Email: "I was charged twice for my subscription."
Category: Billing

Example 2:
Email: "My app crashes after the update."
Category: Technical Support

Example 3:
Email: "Great service and fast response."
Category: Feedback

Now classify the following email:
Email: "What are your customer support working hours?"
"""

run_prompt(few_shot_prompt, "Others")
```

```
REPORT:


Example 1:
Email: "I was charged twice for my subscription."
Category: Billing

Example 2:
```

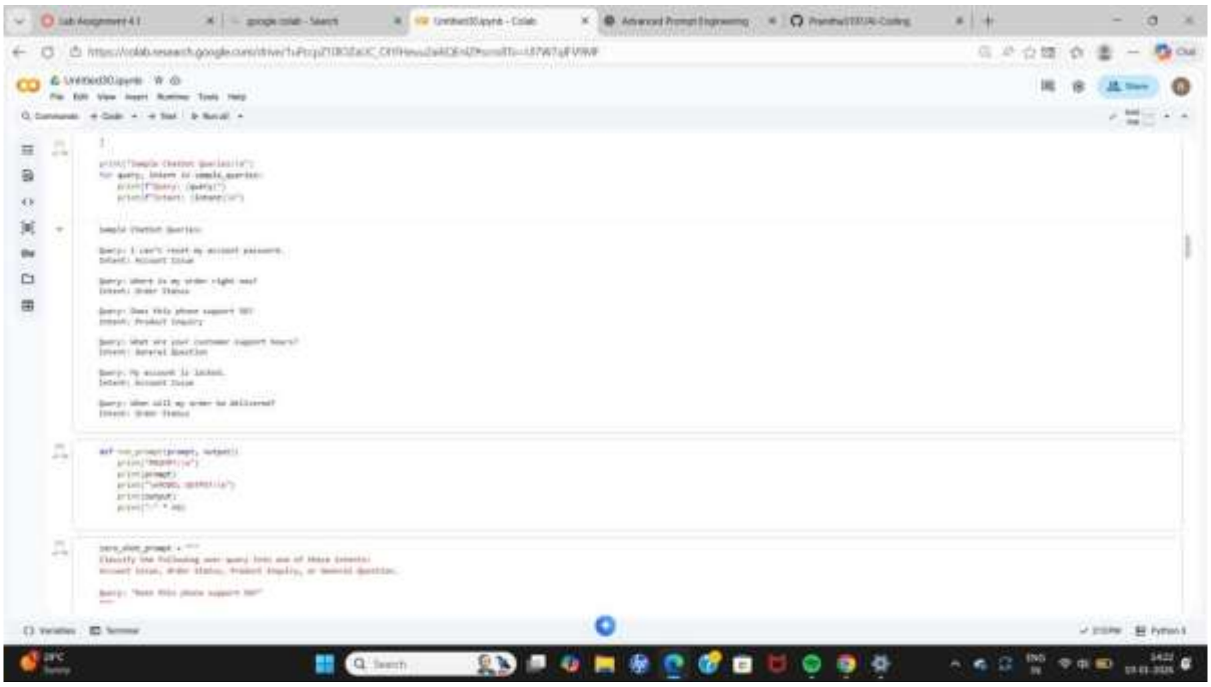


```
Example 2:
Email: "My app crashes after the update."
Category: Technical Support

Example 3:
Email: "Great service and fast response."
Category: Feedback

Now classify the following email:
Email: "What are your customer support working hours?"


MODEL OUTPUT:

Others
```

```
comparison = {
    "Zero-shot": "Correct for clear and simple emails",
    "One-shot": "Improves clarity by showing one example",
    "Few-shot": "Most accurate and consistent due to multiple examples"
}

print("Comparison of Prompting Techniques:\n")
for method, observation in comparison.items():
    print(f"{method}: {observation}")
```

```
Comparison of Prompting Techniques:

Zero-shot: Correct for clear and simple emails
One-shot: Improves clarity by showing one example
Few-shot: Most accurate and consistent due to multiple examples
```

```
# Sample chatbot queries with intents
sample_queries = [
    ("I can't reset my account password.", "Account Issue"),
    ("Where is my order right now?", "Order Status"),
    ("Does this phone support 5G?", "Product Inquiry"),
    ("What are your customer support hours?", "General Question"),
    ("My account is locked.", "Account Issue"),
    ("When will my order be delivered?", "Order Status")
]
```

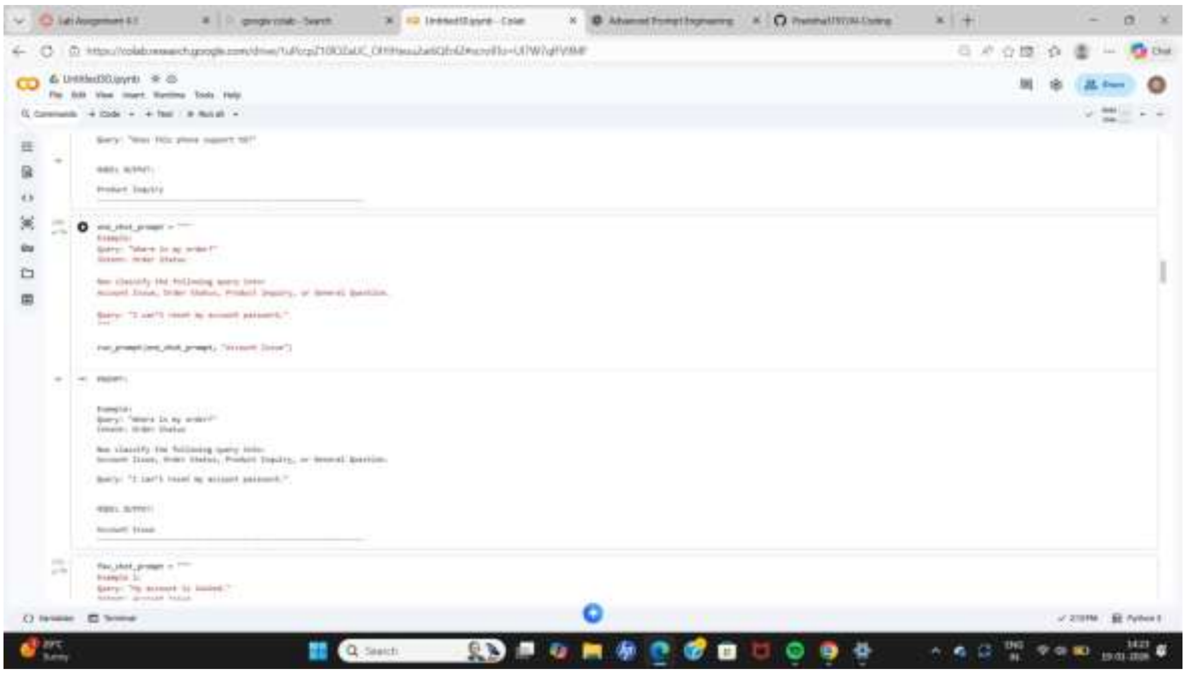


```
]

print("Sample Chatbot Queries:\n")
for query, intent in sample_queries:
    print(f"Query: {query}")
    print(f"Intent: {intent}\n")
```

```
Sample Chatbot Queries:

Query: I can't reset my account password.
Intent: Account Issue

Query: Where is my order right now?
Intent: Order Status

Query: Does this phone support 5G?
Intent: Product Inquiry

Query: What are your customer support hours?
Intent: General Question

Query: My account is locked.
Intent: Account Issue

Query: When will my order be delivered?
Intent: Order Status
```

```
def run_prompt(prompt, output):
    print("PROMPT:\n")
    print(prompt)
    print("\nMODEL OUTPUT:\n")
    print(output)
    print("-" * 40)
```

```
zero_shot_prompt = """
Classify the following user query into one of these intents:
Account Issue, Order Status, Product Inquiry, or General Question.

Query: "Does this phone support 5G?"
"""
```

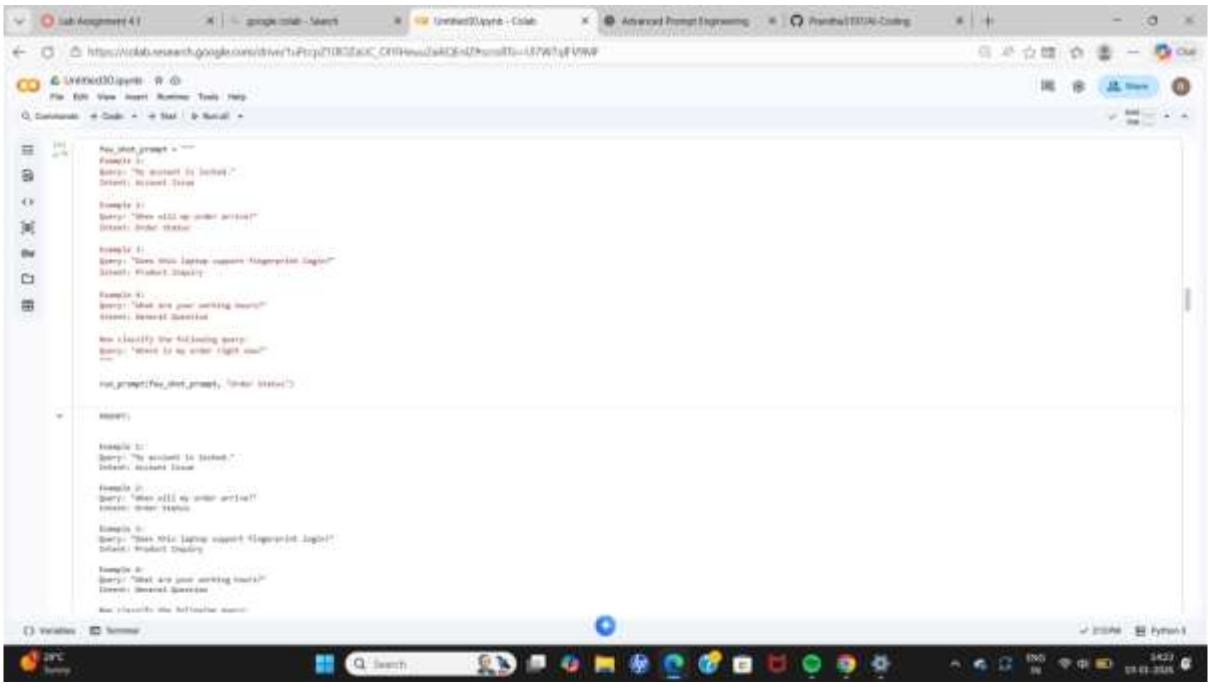


```
Query: "Does this phone support 5G?"

MODEL OUTPUT:

Product Inquiry
----------------------------------------
```

```
one_shot_prompt = """
Example:
Query: "Where is my order?"
Intent: Order Status

Now classify the following query into:
Account Issue, Order Status, Product Inquiry, or General Question.

Query: "I can't reset my account password."
"""

run_prompt(one_shot_prompt, "Account Issue")
```

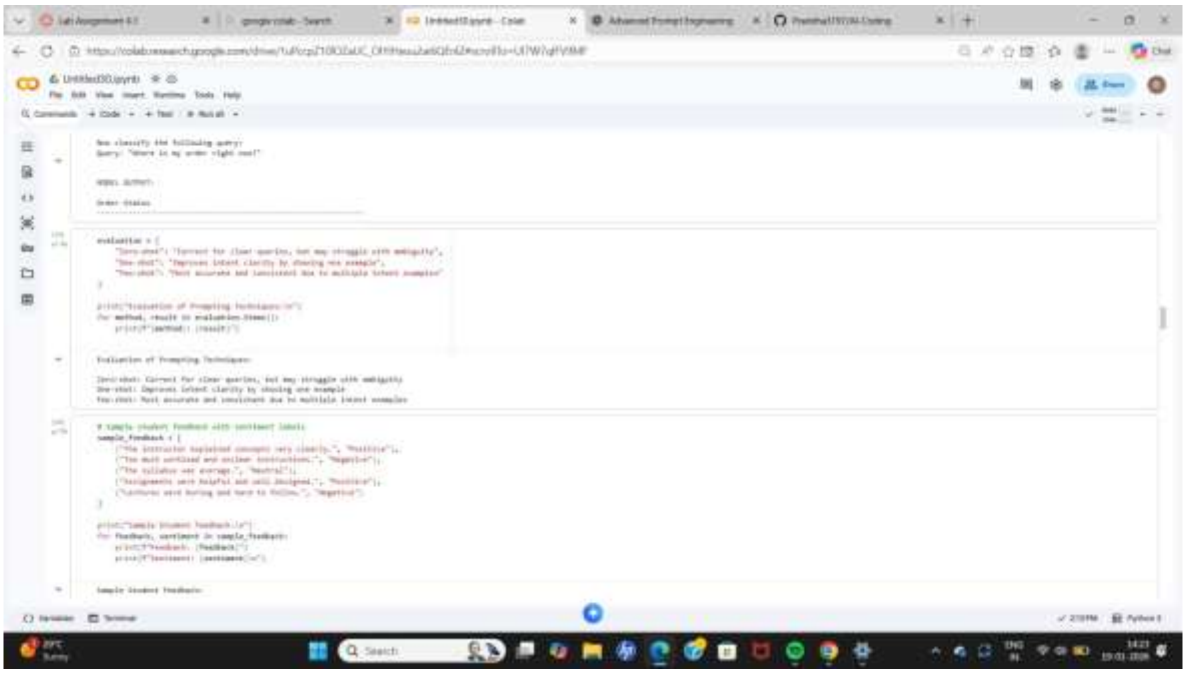
```
PROMPT:


Example:
Query: "Where is my order?"
Intent: Order Status

Now classify the following query into:
Account Issue, Order Status, Product Inquiry, or General Question.

Query: "I can't reset my account password."


MODEL OUTPUT:

Account Issue
----------------------------------------
```

```
few_shot_prompt = """
Example 1:
Query: "My account is locked."
Intent: Account Issue
```



```
few_shot_prompt = """
Example 1:
Query: "My account is locked."
Intent: Account Issue

Example 2:
Query: "When will my order arrive?"
Intent: Order Status

Example 3:
Query: "Does this laptop support fingerprint login?"
Intent: Product Inquiry

Example 4:
Query: "What are your working hours?"
Intent: General Question

Now classify the following query:
Query: "Where is my order right now?"
"""

run_prompt(few_shot_prompt, "Order Status")
```

```
PROMPT:

Example 1:
Query: "My account is locked."
Intent: Account Issue

Example 2:
Query: "When will my order arrive?"
Intent: Order Status

Example 3:
Query: "Does this laptop support fingerprint login?"
Intent: Product Inquiry

Example 4:
Query: "What are your working hours?"
Intent: General Question

Now classify the following query:
```



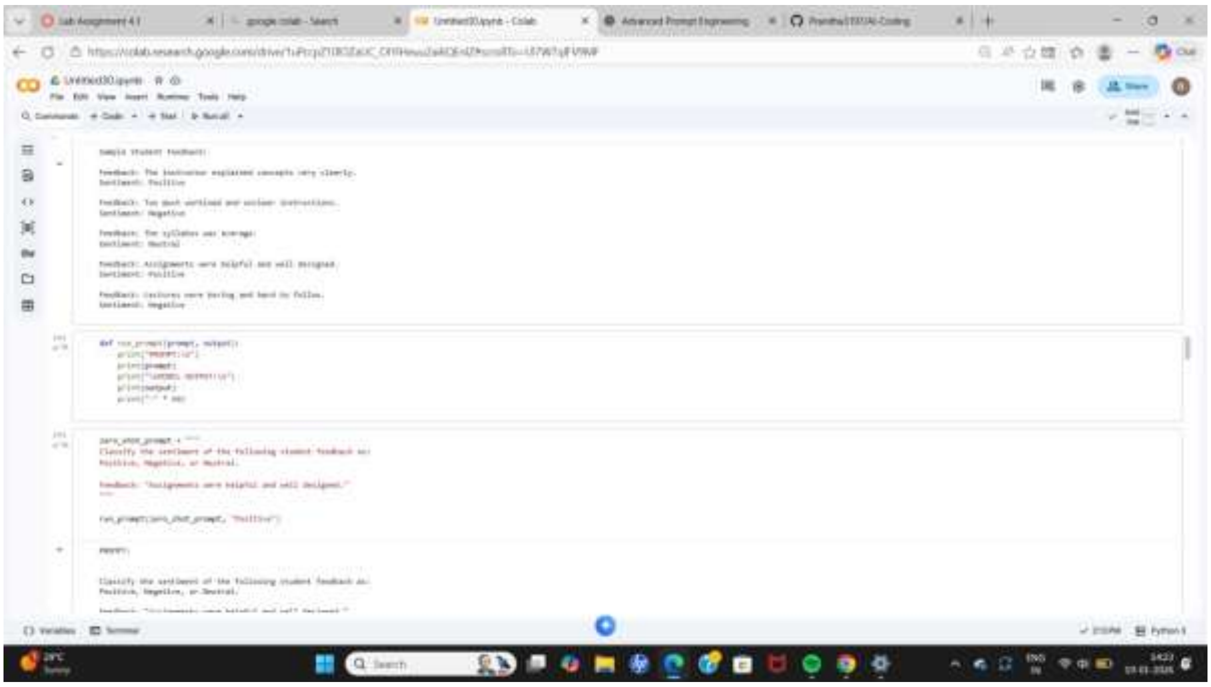
```
Now classify the following query:
Query: "Where is my order right now?"

MODEL OUTPUT:

Order Status
```

```
evaluation = {
    "Zero-shot": "Correct for clear queries, but may struggle with ambiguity",
    "One-shot": "Improves intent clarity by showing one example",
    "Few-shot": "Most accurate and consistent due to multiple intent examples"
}

print("Evaluation of Prompting Techniques:\n")
for method, result in evaluation.items():
    print(f"{method}: {result}")
```

```
Evaluation of Prompting Techniques:

Zero-shot: Correct for clear queries, but may struggle with ambiguity
One-shot: Improves intent clarity by showing one example
Few-shot: Most accurate and consistent due to multiple intent examples
```

```
# Sample student feedback with sentiment labels
sample_feedback = [
    ("The instructor explained concepts very clearly.", "Positive"),
    ("Too much workload and unclear instructions.", "Negative"),
    ("The syllabus was average.", "Neutral"),
    ("Assignments were helpful and well designed.", "Positive"),
    ("Lectures were boring and hard to follow.", "Negative")
]

print("Sample Student Feedback:\n")
for feedback, sentiment in sample_feedback:
    print(f"Feedback: {feedback}")
    print(f"Sentiment: {sentiment}\n")
```

```
Sample Student Feedback:
```

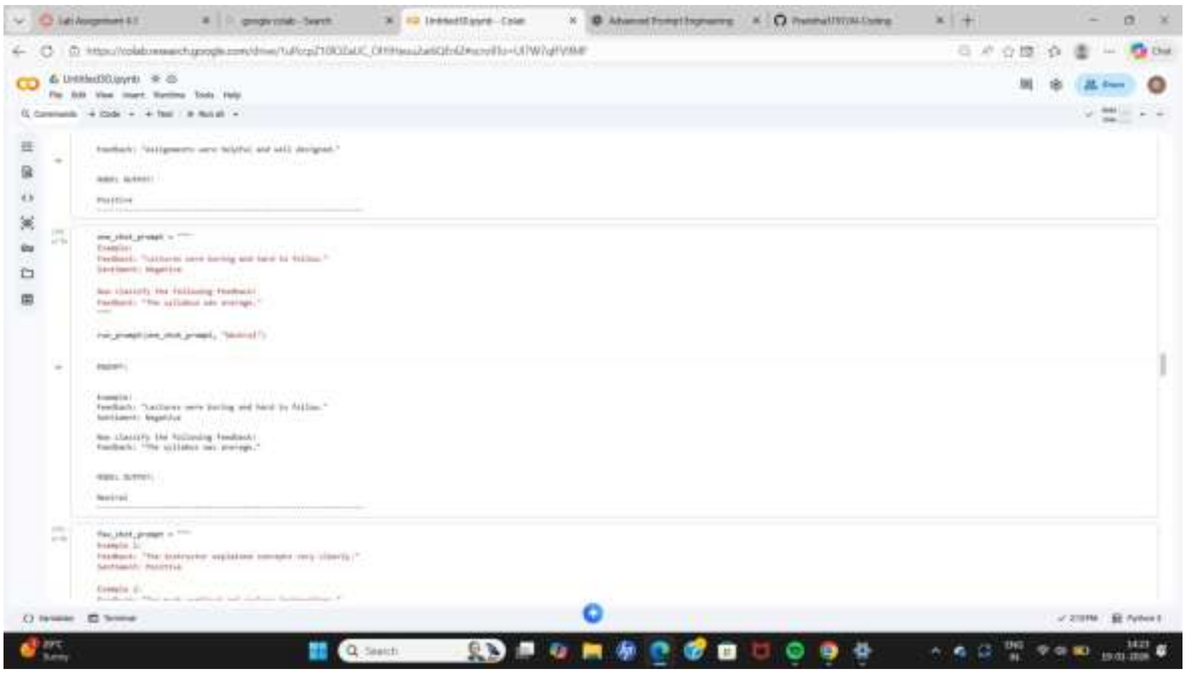


```
Sample Student Feedback:

Feedback: The instructor explained concepts very clearly.
Sentiment: Positive

Feedback: Too much workload and unclear instructions.
Sentiment: Negative

Feedback: The syllabus was average.
Sentiment: Neutral

Feedback: Assignments were helpful and well designed.
Sentiment: Positive

Feedback: Lectures were boring and hard to follow.
Sentiment: Negative
```

```
def run_prompt(prompt, output):
    print("PROMPT:\n")
    print(prompt)
    print("\nMODEL OUTPUT:\n")
    print(output)
    print("-" * 80)
```

```
zero_shot_prompt = """
Classify the sentiment of the following student feedback as:
Positive, Negative, or Neutral.

Feedback: "Assignments were helpful and well designed."
"""

run_prompt(zero_shot_prompt, "Positive")
```

```
PROMPT:

Classify the sentiment of the following student feedback as:
Positive, Negative, or Neutral.

Feedback: "Assignments were helpful and well designed."
```

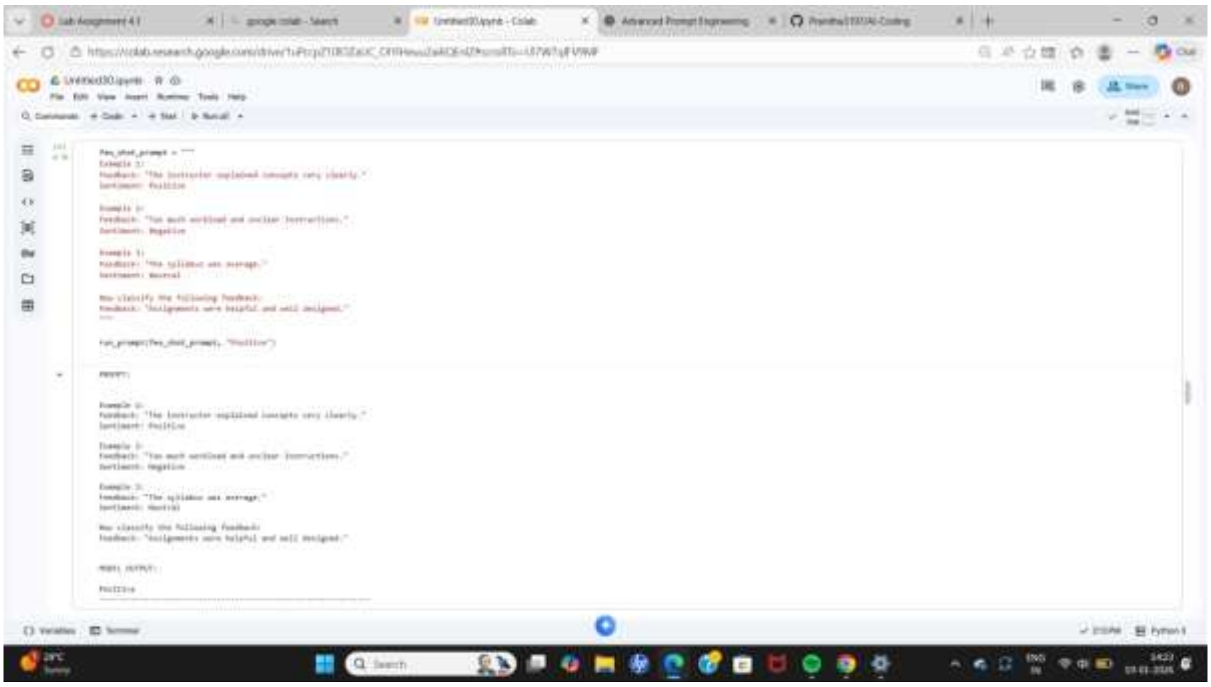


```
Feedback: "Assignments were helpful and well designed."

MODEL OUTPUT:

Positive
```

```
one_shot_prompt = """
Example:
Feedback: "Lectures were boring and hard to follow."
Sentiment: Negative

Now classify the following feedback:
Feedback: "The syllabus was average."
"""

run_prompt(one_shot_prompt, "Neutral")
```

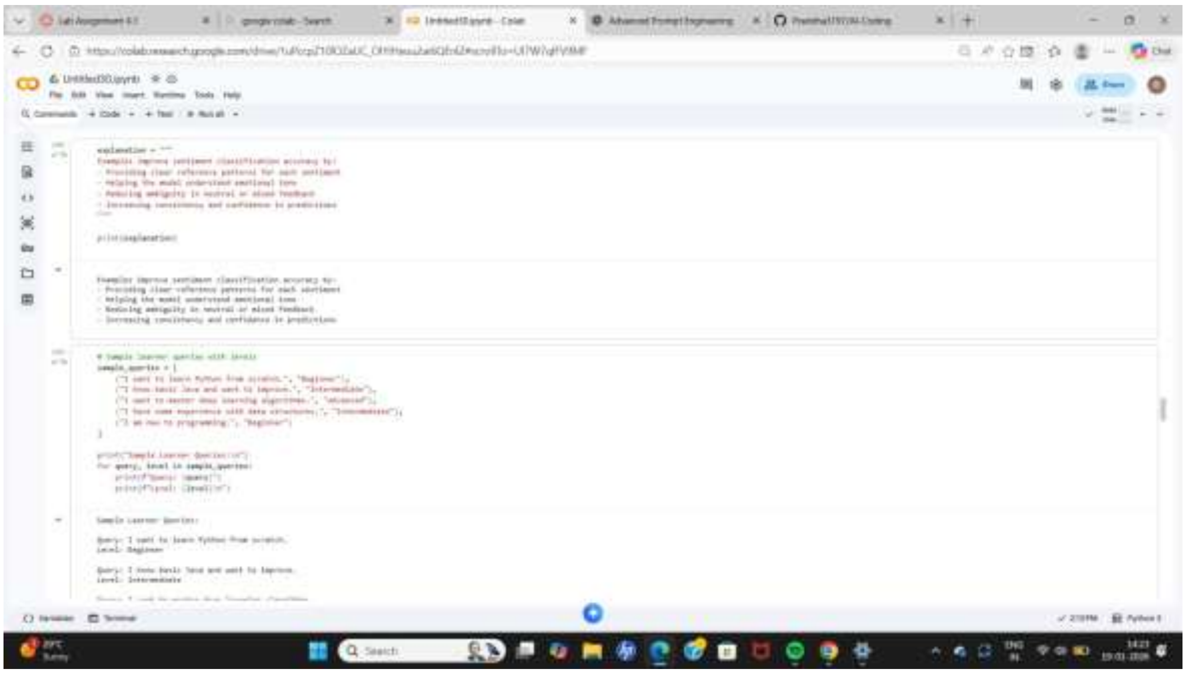
```
PROMPT:

Example:
Feedback: "Lectures were boring and hard to follow."
Sentiment: Negative

Now classify the following feedback:
Feedback: "The syllabus was average."

MODEL OUTPUT:

Neutral
```

```
few_shot_prompt = """
Example 1:
Feedback: "The instructor explained concepts very clearly."
Sentiment: Positive

Example 2:
```



```
few_shot_prompt = """
Example 1:
Feedback: "The instructor explained concepts very clearly."
Sentiment: Positive

Example 2:
Feedback: "Too much workload and unclear instructions."
Sentiment: Negative

Example 3:
Feedback: "The syllabus was average."
Sentiment: Neutral

Now classify the following feedback:
Feedback: "Assignments were helpful and well designed."
"""

run_prompt(few_shot_prompt, "Positive")
```

```
PROMPT:


Example 1:
Feedback: "The instructor explained concepts very clearly."
Sentiment: Positive

Example 2:
Feedback: "Too much workload and unclear instructions."
Sentiment: Negative

Example 3:
Feedback: "The syllabus was average."
Sentiment: Neutral

Now classify the following feedback:
Feedback: "Assignments were helpful and well designed."


MODEL OUTPUT:

Positive
--------------------------------------------------
```

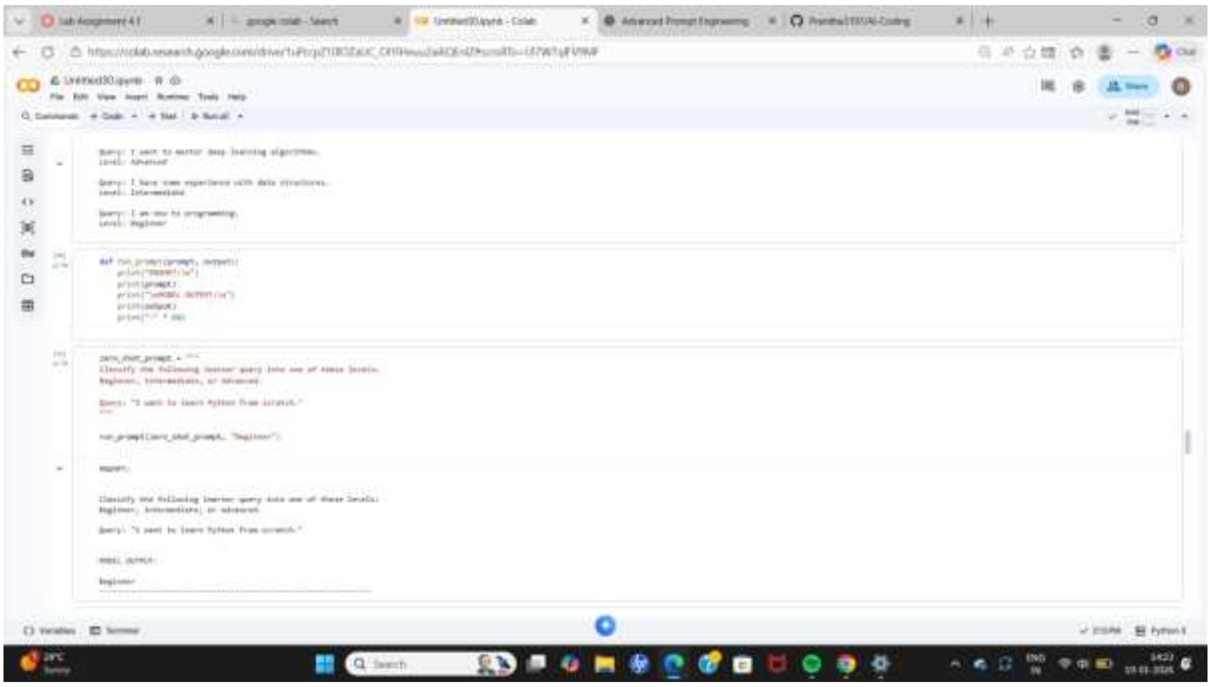


```
explanation = """
Examples improve sentiment classification accuracy by:
- Providing clear reference patterns for each sentiment
- Helping the model understand emotional tone
- Reducing ambiguity in neutral or mixed feedback
- Increasing consistency and confidence in predictions
"""

print(explanation)
```

```
Examples improve sentiment classification accuracy by:
- Providing clear reference patterns for each sentiment
- Helping the model understand emotional tone
- Reducing ambiguity in neutral or mixed feedback
- Increasing consistency and confidence in predictions
```

```
# Sample learner queries with levels
sample_queries = [
    ("I want to learn Python from scratch.", "Beginner"),
    ("I know basic Java and want to improve.", "Intermediate"),
    ("I want to master deep learning algorithms.", "Advanced"),
    ("I have some experience with data structures.", "Intermediate"),
    ("I am new to programming.", "Beginner")
]

print("Sample Learner Queries:\n")
for query, level in sample_queries:
    print(f"Query: {query}")
    print(f"Level: {level}\n")
```

```
Sample Learner Queries:

Query: I want to learn Python from scratch.
Level: Beginner

Query: I know basic Java and want to improve.
Level: Intermediate
```

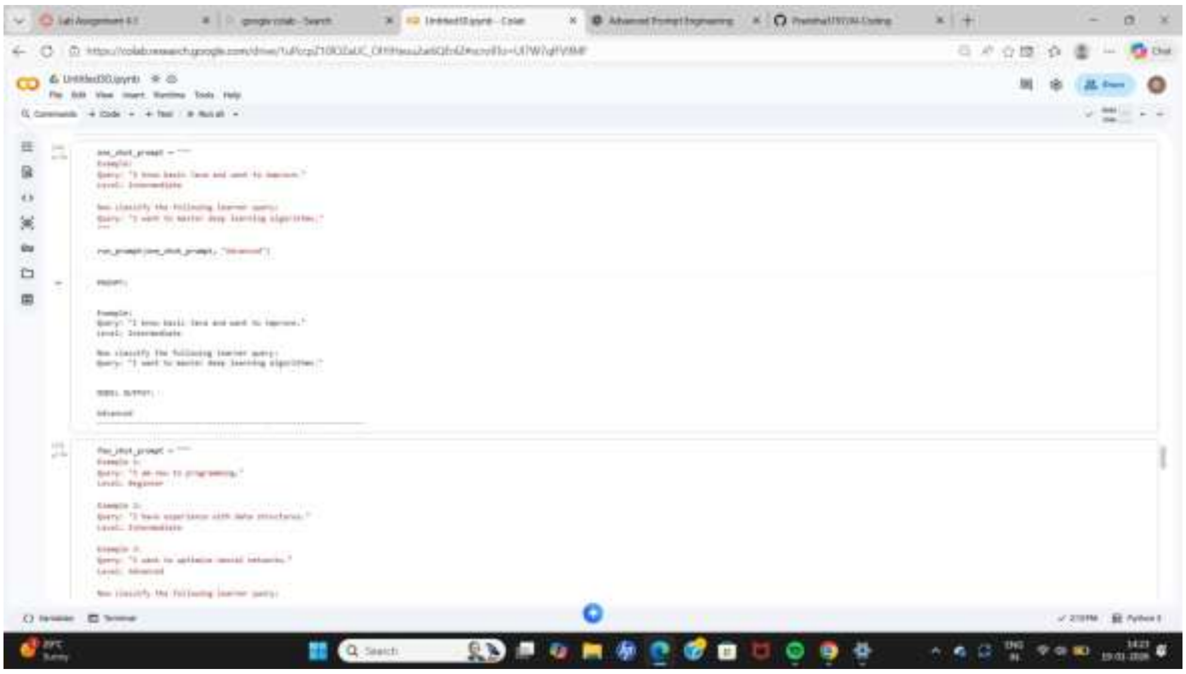


```
Query: I want to master deep learning algorithms.
Level: Advanced

Query: I have some experience with data structures.
Level: Intermediate

Query: I am new to programming.
Level: Beginner
```

```
def run_prompt(prompt, output):
    print("PROMPT:\n")
    print(prompt)
    print("\nMODEL OUTPUT:\n")
    print(output)
    print("-" * 80)
```

```
zero_shot_prompt = """
Classify the following learner query into one of these levels:
Beginner, Intermediate, or Advanced

Query: "I want to learn Python from scratch."
"""

run_prompt(zero_shot_prompt, "Beginner")
```

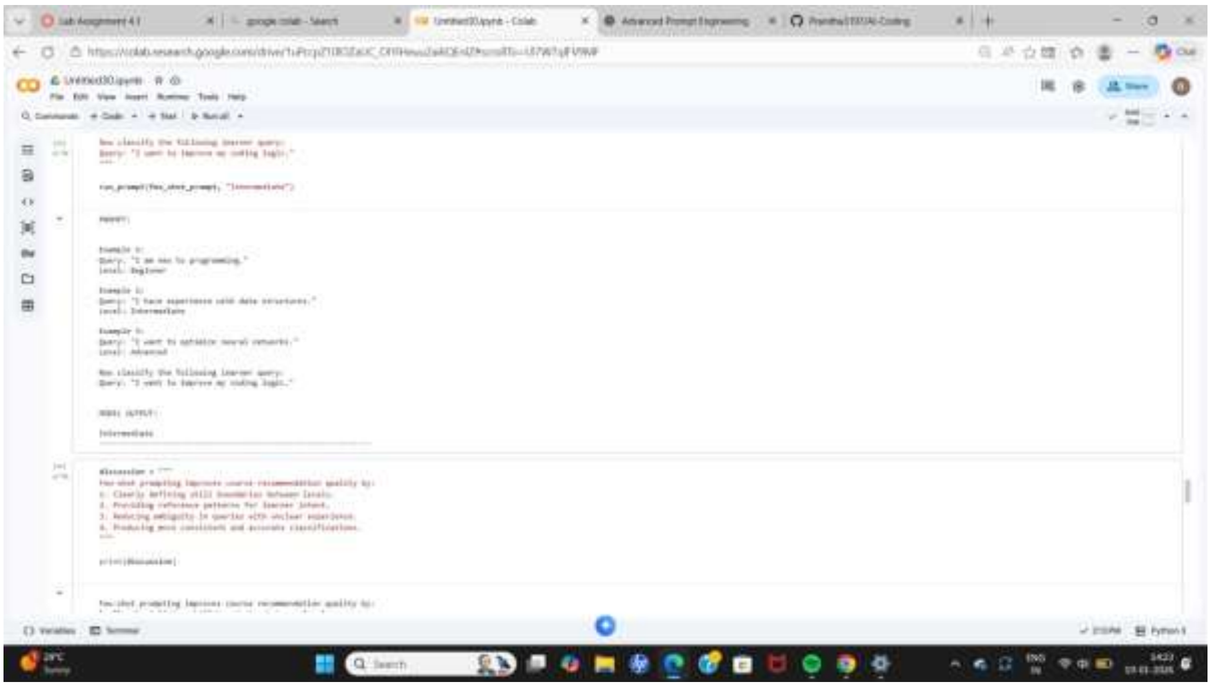
```
PROMPT:

Classify the following learner query into one of these levels:
Beginner, Intermediate, or Advanced

Query: "I want to learn Python from scratch."


MODEL OUTPUT:

Beginner
--------------------------------------------------------------------------------
```



```
one_shot_prompt = """
Example:
Query: "I know basic Java and want to improve."
Level: Intermediate

Now classify the following learner query:
Query: "I want to master deep learning algorithms."
"""

run_prompt(one_shot_prompt, "Advanced")
```

```
PROMPT:


Example:
Query: "I know basic Java and want to improve."
Level: Intermediate

Now classify the following learner query:
Query: "I want to master deep learning algorithms."


MODEL OUTPUT:

Advanced
--------------------------------------------------------------------------------
```

```
few_shot_prompt = """
Example 1:
Query: "I am new to programming."
Level: Beginner

Example 2:
Query: "I have experience with data structures."
Level: Intermediate

Example 3:
Query: "I want to optimize neural networks."
Level: Advanced

Now classify the following learner query:
```

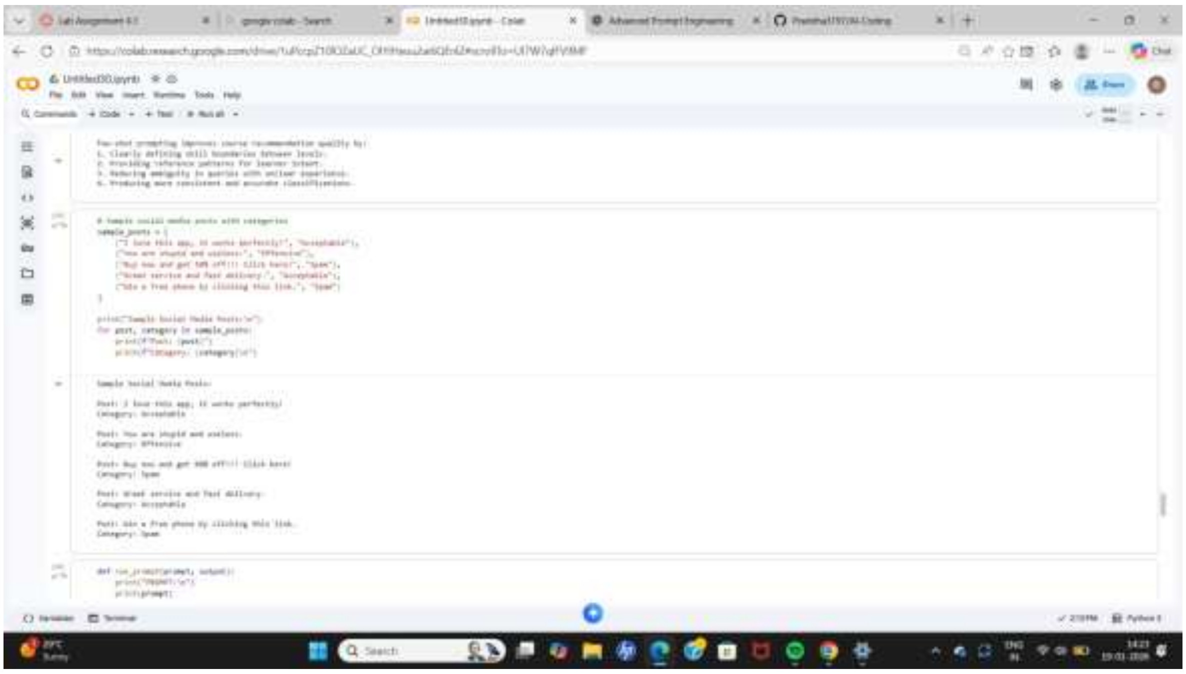


```
Now classify the following learner query:
Query: "I want to improve my coding logic."
"""

run_prompt(few_shot_prompt, "Intermediate")
```

```
PROMPT:

Example 1:
Query: "I am new to programming."
Level: Beginner

Example 2:
Query: "I have experience with data structures."
Level: Intermediate

Example 3:
Query: "I want to optimize neural networks."
Level: Advanced

Now classify the following learner query:
Query: "I want to improve my coding logic."


MODEL OUTPUT:

Intermediate
```

```
discussion = """
Few-shot prompting improves course recommendation quality by:
1. Clearly defining skill boundaries between levels.
2. Providing reference patterns for learner intent.
3. Reducing ambiguity in queries with unclear experience.
4. Producing more consistent and accurate classifications.
"""

print(discussion)
```

```
Few-shot prompting improves course recommendation quality by:
```

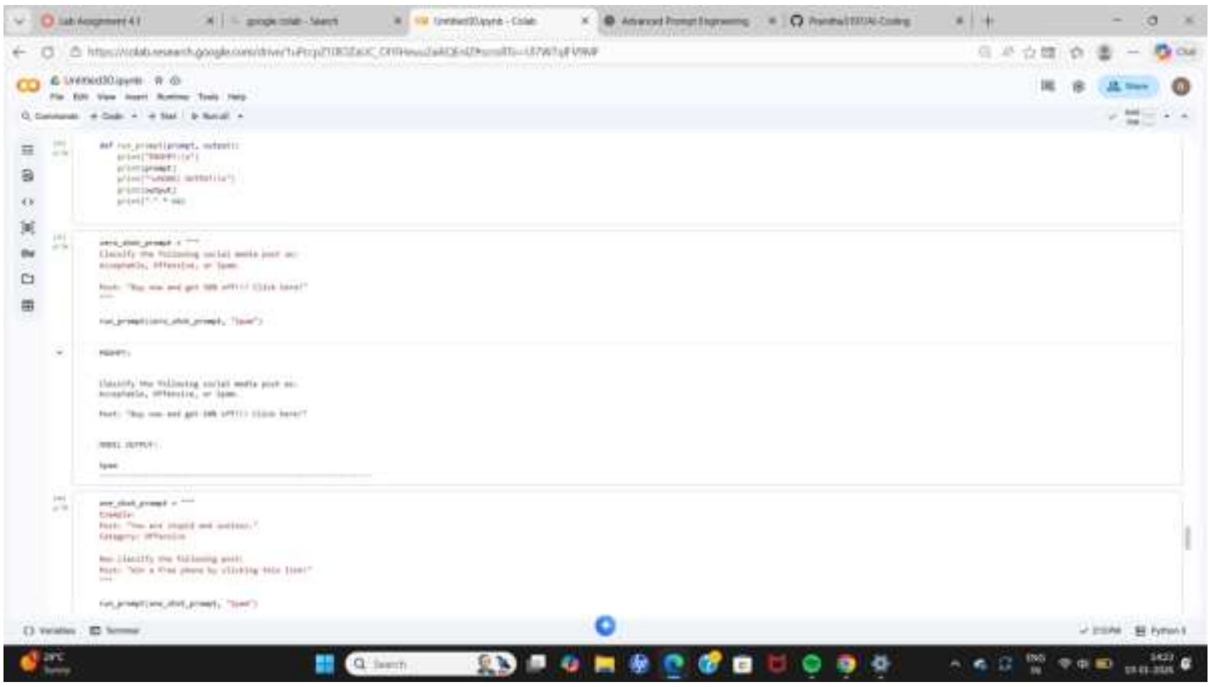


```
Few-shot prompting improves course recommendation quality by:
1. Clearly defining skill boundaries between levels.
2. Providing reference patterns for learner intent.
3. Reducing ambiguity in queries with unclear experience.
4. Producing more consistent and accurate classifications.
```

```
# Sample social media posts with categories
sample_posts = [
    ("I love this app, it works perfectly!", "Acceptable"),
    ("You are stupid and useless.", "Offensive"),
    ("Buy now and get 50% off!!! Click here!", "Spam"),
    ("Great service and fast delivery.", "Acceptable"),
    ("Win a free phone by clicking this link.", "Spam")
]

print("Sample Social Media Posts:\n")
for post, category in sample_posts:
    print(f"Post: {post}")
    print(f"Category: {category}\n")
```

```
Sample Social Media Posts:

Post: I love this app, it works perfectly!
Category: Acceptable

Post: You are stupid and useless.
Category: Offensive

Post: Buy now and get 50% off!!! Click here!
Category: Spam

Post: Great service and fast delivery.
Category: Acceptable

Post: Win a free phone by clicking this link.
Category: Spam
```

```
def run_prompt(prompt, output):
    print("PROMPT:\n")
    print(prompt)
```

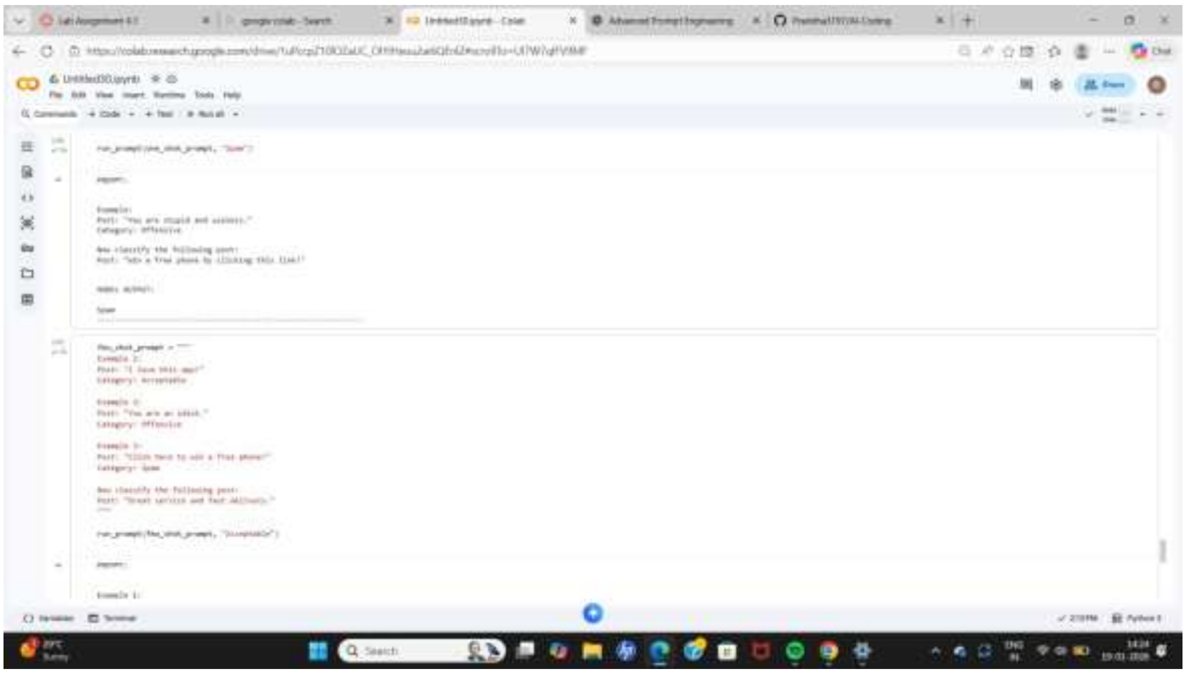


```
def run_prompt(prompt, output):
    print("PROMPT:\n")
    print(prompt)
    print("\nMODEL OUTPUT:\n")
    print(output)
    print("-" * 60)
```

```
zero_shot_prompt = """
Classify the following social media post as:
Acceptable, Offensive, or Spam.

Post: "Buy now and get 50% off!!! Click here!"
"""

run_prompt(zero_shot_prompt, "Spam")
```

```
PROMPT:


Classify the following social media post as:
Acceptable, Offensive, or Spam.

Post: "Buy now and get 50% off!!! Click here!"


MODEL OUTPUT:

Spam
------------------------------------------------------------
```

```
one_shot_prompt = """
Example:
Post: "You are stupid and useless."
Category: Offensive

Now classify the following post:
Post: "Win a free phone by clicking this link!"
"""

run_prompt(one_shot_prompt, "Spam")
```

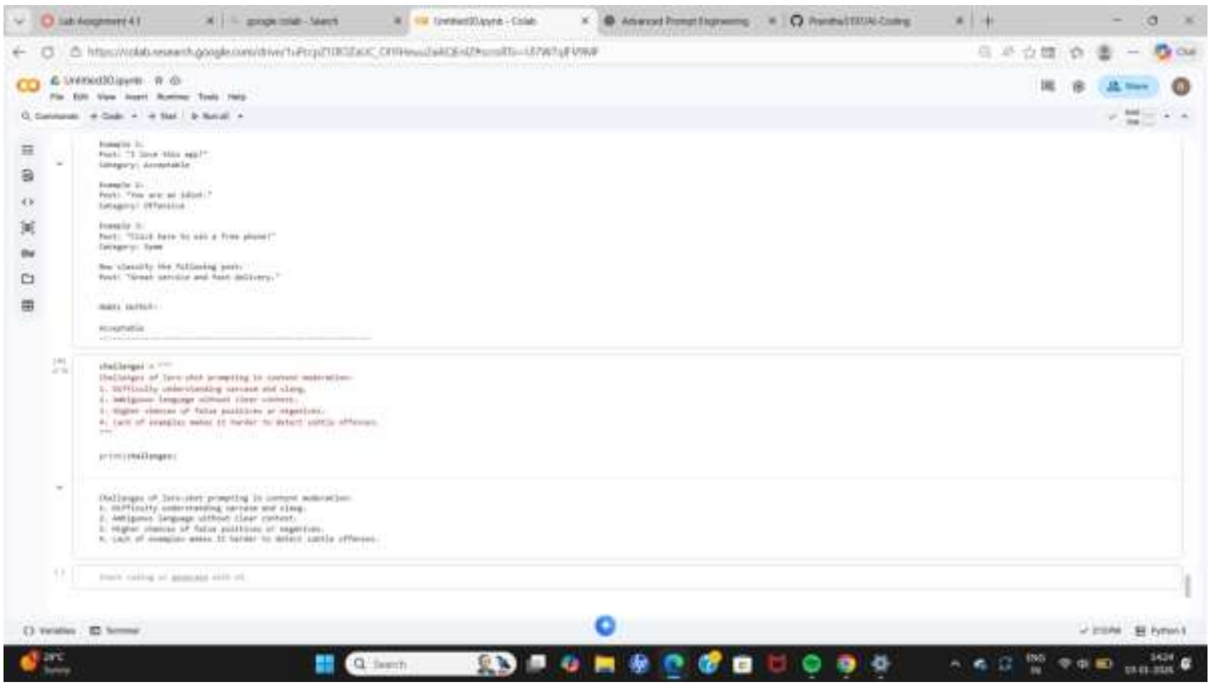


```
run_prompt(one_shot_prompt, "Spam")
```

```
PROMPT:


Example:
Post: "You are stupid and useless."
Category: Offensive

Now classify the following post:
Post: "Win a free phone by clicking this link!"


MODEL OUTPUT:

Spam
------------------------------------------------------------
```

```
few_shot_prompt = """
Example 1:
Post: "I love this app!"
Category: Acceptable

Example 2:
Post: "You are an idiot."
Category: Offensive

Example 3:
Post: "Click here to win a free phone!"
Category: Spam

Now classify the following post:
Post: "Great service and fast delivery."
"""

run_prompt(few_shot_prompt, "Acceptable")
```

```
PROMPT:


Example 1:
```



```
Example 1:
Post: "I love this app!"
Category: Acceptable

Example 2:
Post: "You are an idiot!"
Category: Offensive

Example 3:
Post: "Click here to win a free phone!"
Category: Spam

Now classify the following post:
Post: "Great service and fast delivery."


Model Output:

Acceptable
```

```
challenges = """
Challenges of Zero-shot prompting in content moderation:
1. Difficulty understanding sarcasm and slang.
2. Ambiguous language without clear context.
3. Higher chances of false positives or negatives.
4. Lack of examples makes it harder to detect subtle offenses.
"""

print(challenges)
```

```
Challenges of Zero-shot prompting in content moderation:
1. Difficulty understanding sarcasm and slang.
2. Ambiguous language without clear context.
3. Higher chances of false positives or negatives.
4. Lack of examples makes it harder to detect subtle offenses.
```

Final Observation for Problem Statement 1

Zero-shot prompting works well for straightforward emails.

One-shot prompting improves understanding by providing context.

Few-shot prompting gives the best performance by clearly defining category boundaries and reducing ambiguity.

Final Observation for Problem Statement 2

Zero-shot prompting works for simple and explicit queries.

One-shot prompting improves understanding with minimal context.

Few-shot prompting provides the best performance by clearly defining intent boundaries and reducing ambiguity.

Final Observation for Problem Statement 3

Zero-shot prompting works for clearly emotional feedback.

One-shot prompting improves understanding with minimal guidance.

Few-shot prompting gives the best accuracy by clearly defining positive, negative, and neutral sentiment patterns

Final Observation for Problem Statement 4

Zero-shot prompting works for very clear beginner or advanced queries.

One-shot prompting improves classification with minimal guidance.

Few-shot prompting provides the best results by clearly distinguishing

between beginner, intermediate, and advanced learning needs.

Final Observation for Problem Statement 5

Zero-shot prompting works for clearly spam or offensive posts.

However, it struggles with ambiguity and sarcasm.

One-shot improves clarity, while Few-shot prompting gives the

most accurate and reliable moderation results.