

LAB ASSIGNMENT 4.1

NAME : M.Geethika

2303A52276

SUBJECT : AI ASSISTED CODING

The screenshot shows a Jupyter Notebook interface in Google Colab. The code cell contains the following Python script:

```
# Classify customer emails into categories
simple_email = """
Hello, I am charged value for my subscription this month." "Billing"
,"My application creates whenever I open it.", "Technical Support"
,"I would like the new feature, about price.", "Feedback"
,"What are your customer support working hours?", "Others"
,"I am unable to login to my account.", "Technical Support"

print("Simple Email")
for email in simple_email:
    print(email)
    print(classify_email(email))
    print(classify_feedback(email))
    print(classify_others(email))
    print(classify_technical_support(email))

def run_prompt(prompt, response):
    print("INPUT:")
    print(response)

# Run the project
run_prompt("""
Classify the following customer email into one of three categories:
Billing, Technical Support, Feedback, Others.

Hello, I am charged value for my subscription this month." """, "Billing")

not_prompt(run_start_prompt, "Billing")

# Run the project again
run_prompt("""
Classify the following customer email into one of three categories:
Billing, Technical Support, Feedback, Others.

Hello, I am charged value for my subscription this month." """, "Billing")

# Run the project again
run_prompt("""
Hello, I am charged value for my subscription this month." """, "Billing")
```

The screenshot shows a Jupyter Notebook interface in Google Colab. The code cell contains the following Python script:

```
# Run the project
run_start_prompt = """
Hello, I am charged value for my subscription this month." "Billing"
,"My application creates whenever I open it.", "Technical Support"
,"I would like the new feature, about price.", "Feedback"
,"What are your customer support working hours?", "Others"
,"I am unable to login to my account.", "Technical Support"

print("Simple Email")
for email in simple_email:
    print(email)
    print(classify_email(email))
    print(classify_feedback(email))
    print(classify_others(email))
    print(classify_technical_support(email))

def run_prompt(prompt, response):
    print("INPUT:")
    print(response)

# Run the project
run_prompt("""
Classify the following customer email into one of three categories:
Billing, Technical Support, Feedback, Others.

Hello, I am charged value for my subscription this month." """, "Billing")

not_prompt(run_start_prompt, "Billing")

# Run the project again
run_prompt("""
Classify the following customer email into one of three categories:
Billing, Technical Support, Feedback, Others.

Hello, I am charged value for my subscription this month." """, "Billing")

# Run the project again
run_prompt("""
Hello, I am charged value for my subscription this month." """, "Billing")
```

```
Lab Assignment 4 | google.colab - Search | Untitled0.pyptn - Code | Advanced Prompt Engineering | Prantha100A-Coding | + - O X
https://colab.research.google.com/drive/u/Prop71002dAKC_0fTHmwszQhDZmsofTs-u57W7gUW#p
Untitled0.pyptn - P | File Edit View Insert Runtime Tools Help
Q Commands → Code → Run | & Recall |
Example:
Email: "My application crashes frequently."
Category: Technical Support
Run classify the following email into:
Billing, Technical Support, Feedback, or Others.
Email: "I recently tried the new update. Great job!"
```

EMAIL OUTPUT:

```
Feedback
```

Run show_prompt = ""

```
Email: "I just received a bill for my subscription."
Category: Billing
```

Example 1:

```
Email: "My car started after the update."
Category: Technical Support
```

Example 2:

```
Email: "Great service and fast response."
Category: Feedback
```

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

```
run_prompt(show_prompt, "Other")
```

EMAIL OUTPUT:

```
Others
```

Example 3:

```
Email: "I just received a bill for my subscription."
Category: Billing
```

Example 4:

```
Email: "Great service and fast response."
Category: Feedback
```

EMAIL OUTPUT:

```
Feedback
```

```
Lab Assignment 4 | google.colab - Search | Untitled0.pyptn - Code | Advanced Prompt Engineering | Prantha100A-Coding | + - O X
https://colab.research.google.com/drive/u/Prop71002dAKC_0fTHmwszQhDZmsofTs-u57W7gUW#p
Untitled0.pyptn - P | File Edit View Insert Runtime Tools Help
Q Commands → Code → Run | & Recall |
Example 1:
Email: "My app crashed after the update."
Category: Technical Support
Example 2:
Email: "Great service and fast response."
Category: Feedback
Run classify the following email:
Email: "What are your customer support working hours?"
```

EMAIL OUTPUT:

```
Feedback
```

Run comparison = []
"Show-start": "Present the class as simple as possible",
"Show-end": "Improve clarity by showing one example",
"Show-start": "Most accurate and consistent due to multiple examples"
]

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

Description of Prompting Techniques:
Show-start: Present the class as simple as possible
Show-end: Improves clarity by showing one example
Show-start: Most accurate and consistent due to multiple examples

A simple question requires little iteration:
sample_questions = [
 ("What's your account password?", "Account login"),
 ("What is my user registration?", "User creation"),
 ("What's the current weather like?", "Weather forecast"),
 ("What are your account support hours?", "General questions"),
 ("My account is locked.", "Account login"),
 ("What will my order be delivered?", "Delivery status")
]

The screenshot shows a browser window with several tabs open, all related to AI development and prompt engineering. The main content area is a Jupyter Notebook cell containing Python code and JSON data. The code includes functions for generating responses based on user input and handling prompts. The JSON data defines various roles like 'Customer Support' and 'Product Manager' with their respective responsibilities. A terminal window at the bottom shows a command-line interface with the command 'dvc status'.

```
print("Hello! How can I assist you today?")  
for query in queries:  
    print(query)  
    print("Answer: ", answers[query])  
    print("")  
  
# Example function to handle customer support  
def handle_customer_support(query):  
    if "customer support" in query:  
        return "Sorry, I can't assist with customer support.  
        Please contact our support team."  
    else:  
        return "Sorry, I can't assist with customer support.  
        Please contact our support team."  
  
# Example function to handle product inquiries  
def handle_product_inquiries(query):  
    if "product" in query:  
        return "Sorry, I can't assist with product inquiries.  
        Please contact our support team."  
    else:  
        return "Sorry, I can't assist with product inquiries.  
        Please contact our support team."  
  
# Main function to handle prompts  
def handle_prompts(prompt, subject):  
    print("Subject: ", subject)  
    print("Prompt: ", prompt)  
    print("Response: ", responses[subject][prompt])  
    print("")  
  
# Example usage  
handle_prompts("What's the weather like in New York?", "Weather")  
  
# dvc status
```

```
Cell 1: 
class Customer(BusinessEntity):
    def __init__(self, id):
        self.id = id

    def get_order_status(self, order_id):
        query = "SELECT Status FROM Orders WHERE OrderID = %s"
        result = self._execute_query(query, (order_id,))
        return result[0][0]

    def check_product_availability(self, product_id):
        query = "SELECT StockLevel FROM Products WHERE ProductID = %s"
        result = self._execute_query(query, (product_id,))
        return result[0][0] > 0

Cell 2: 
class Supplier(BusinessEntity):
    def __init__(self, id):
        self.id = id

    def get_order_status(self, order_id):
        query = "SELECT Status FROM Orders WHERE OrderID = %s"
        result = self._execute_query(query, (order_id,))
        return result[0][0]

    def check_product_availability(self, product_id):
        query = "SELECT StockLevel FROM Products WHERE ProductID = %s"
        result = self._execute_query(query, (product_id,))
        return result[0][0] > 0

Cell 3: 
class BusinessEntity:
    def __init__(self, id):
        self.id = id

    def _execute_query(self, query, parameters):
        # Implementation of execute_query method
```

```
Lab Assignment 4 | google.colab - Search | Untitled0.pyptn - Code | Advanced Prompt Engineering | Prantha100A-Coding | + - O X
https://colab.research.google.com/drive/u/Prop2002aUIC_0HfbauzedQhDmcwvBz-UW/gfVMF
Untitled0.pyptn - P | File Edit View Insert Runtime Tools Help
Q Commands → Code → Test | RunCell


```

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

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

Example 3:
Query: "Does this laptop support <dependent query>"?
Intent: Product Inquiry

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

New classify the following query:
Query: "When is my order right now?"

not promptForInt, prompt, finalStatus()

Method:

```



```

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

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

Example 3:
Query: "Does this laptop support <dependent query>"?
Intent: Product Inquiry

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

New classify the following query:

```



Variables Terminal


```

```
Lab Assignment 4 | google.colab - Search | Untitled0.pyptn - Code | Advanced Prompt Engineering | Prantha100A-Coding | + - O X
https://colab.research.google.com/drive/u/Prop2002aUIC_0HfbauzedQhDmcwvBz-UW/gfVMF
Untitled0.pyptn - P | File Edit View Insert Runtime Tools Help
Q Commands → Code → Test | RunCell

```

```

New classify the following query:
Query: "When is my order right now?"

Method: prompt
Intent: Status

Evaluation:
evaluator = [
    ("too-weak", "Request for clear query, but my struggle with ambiguity"),
    ("too-short", "Requires intent clarity by showing one example"),
    ("too-weak", "More accurate and consistent due to multiple intent samples")
]

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

Evaluation of Prompting Techniques:
too-weak: Request for clear query, but my struggle with ambiguity
too-short: Requires intent clarity by showing one example
too-weak: More accurate and consistent due to multiple intent samples

# Sample random feedback after consistent intent
sample_feedback = [
    ("The writer has explained very clearly.", "Positive"),
    ("The most accurate and easiest construction.", "Positive"),
    ("The solution was average.", "Neutral"),
    ("Provides next helpful and easy-to-understand.", "Positive"),
    ("Somewhat useful and clear to follow.", "Positive")
]

print("\nSample random feedback:")
for feedback in sample_feedback:
    print(f"\t{feedback[0]}: {feedback[1]}")

Sample random feedback:
```

Variables Terminal

Lab Assignment 4 | google.colab - Search | Untitled0.pypt - Code | Advanced Prompt Engineering | Prantha1024-Coding | +

Untitled0.pypt - P | File Edit View Insert Runtime Tools Help | Commands → Code → Run | In Recall |

Sample Student Feedback:

```
Feedback: The instructor explained concepts very clearly.  
Sentiment: Positive  
  
Feedback: Very poor verbal peer review communication.  
Sentiment: Negative  
  
Feedback: The xyTables were average.  
Sentiment: Neutral  
  
Feedback: Anticipations were helpful and well designed.  
Sentiment: Positive  
  
Feedback: Guidance were boring and hard to follow.  
Sentiment: Negative
```

def run_prompt(prompt, output):
 print("prompt:", prompt)
 print("output:", output)
 print("-----")
 print("-----")

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

Feedback: "Anticipations were helpful and well designed."

For prompt: para_what_prompt, "Positive"

Neutral

Feedback: The xyTables were average.
Sentiment: Neutral

Feedback: Guidance were boring and hard to follow.
Sentiment: Negative

Feedback: "The xyTables were average."
Sentiment: Neutral

Feedback: Guidance were boring and hard to follow.
Sentiment: Negative

Feedback: "The xyTables were average."
Sentiment: Neutral

Feedback: "The xyTables were average."
Sentiment: Positive

Feedback: "The xyTables were average."
Sentiment: Positive

Variables Terminal ✓ 21:09M Python 3 14:23 10-01-2020

Lab Assignment 4 | google.colab - Search | Untitled0.pypt - Code | Advanced Prompt Engineering | Prantha1024-Coding | +

Untitled0.pypt - P | File Edit View Insert Runtime Tools Help | Commands → Code → Run | In Recall |

Feedback: "Anticipations were helpful and well designed."
Sentiment: Positive

para_what_prompt = """
Example:
Feedback: "Guidance were boring and hard to follow."
Sentiment: Negative

Ask classify the following Feedback:
Feedback: "The xyTables are average."

run_prompt(para_what_prompt, "Neutral")

Neutral

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

Ask classify the following Feedback:
Feedback: "The xyTables are average."

Neutral

Feedback:
Feedback: "The xyTables were average."
Sentiment: Positive

Feedback:
Feedback: "The xyTables were average."
Sentiment: Positive

Feedback:
Feedback: "The xyTables were average."
Sentiment: Positive

Variables Terminal ✓ 21:09M Python 3 14:23 10-01-2020


```
def analyze_query(query, level):
    print(query)
    print(level)

analyze_query("I want to learn deep learning algorithms.", "beginner")
analyze_query("I have some experience with data structures.", "intermediate")
analyze_query("I am new to programming.", "advanced")
```

```
def analyze_query(query, level):
    print(query)
    print(level)

analyze_query("I want to learn deep learning algorithms.", "beginner")
analyze_query("I have some experience with data structures.", "intermediate")
analyze_query("I am new to programming.", "advanced")
```

```
Lab Assignment 4.1 - google.colab - Search Untitled0.pyrb - Code Advanced Prompt Engineering Prantha1024-Coding
```

```
File Edit View Insert Runtime Tools Help
```

```
Untitled0.pyrb
```

```
File Edit View Insert Runtime Tools Help
```

```
Q Commands → Code → Run ⌘ R
```

```
How classify the following user prompt:  
Query: "I want to improve my coding logic."
```

```
run_prompt(query, prompt_type="beginner")
```

```
How???
```

```
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 memory usage."  
Level: Advanced
```

```
How classify the following user prompt:  
Query: "I want to improve my coding logic."
```

```
print("Beginner")
```

```
Intermediate
```

```
Discussions ↗
```

```
Improve prompting improves overall communication quality by:
```

1. Clearly defining skill boundaries between levels.
2. Providing reference patterns for lower levels.
3. Reducing ambiguity in queries with unclear requirements.
4. Producing more consistent and accurate classifications.

```
print("Beginner")
```

```
Improve prompting improves overall communication quality by:
```

```
Q Variables Q Terminal
```

```
21:59PM 14:22 10-01-2024
```

```
Lab Assignment 4.1 - google.colab - Search Untitled0.pyrb - Code Advanced Prompt Engineering Prantha1024-Coding
```

```
File Edit View Insert Runtime Tools Help
```

```
Untitled0.pyrb
```

```
File Edit View Insert Runtime Tools Help
```

```
Q Commands → Code → Run ⌘ R
```

```
How classify the following user prompt with categories:  
Query: "I am a beginner developer. I want to learn Python. Can you suggest any resources? I am new to programming."
```

```
simple_query("I am a beginner developer. I want to learn Python. Can you suggest any resources? I am new to programming.", "beginner")
```

```
How???
```

```
Example 1:  
Query: "I am a beginner developer. I want to learn Python. Can you suggest any resources? I am new to programming."  
Category: Beginner
```

```
Example 2:  
Query: "I am an intermediate developer. I want to learn Python. Can you suggest any resources? I have some experience with data structures."  
Category: Intermediate
```

```
Example 3:  
Query: "I am an advanced developer. I want to learn Python. Can you suggest any resources? I have experience with data structures and want to optimize memory usage."  
Category: Advanced
```

```
How classify the following user prompt:  
Query: "I am a beginner developer. I want to learn Python. Can you suggest any resources? I am new to programming."
```

```
print("Beginner")
```

```
for query, category in simple_query:  
    print(f"Query: {query}")  
    print(f"Category: {category}\n")
```

```
Simple Social Media Posts:
```

```
Post: I have this app; it works perfectly!  
Category: Intermediate
```

```
Post: This was my first post.  
Category: Beginner
```

```
Post: Day one and got 1000 likes!! Click here!  
Category: Intermediate
```

```
Post: Worked hard and had 4000 likes!  
Category: Intermediate
```

```
Post: Take a free phone by clicking this link.  
Category: Beginner
```

```
def run_prompt(prompt_type="beginner"):  
    print("Prompt Type: " + prompt_type)  
    print("Prompt: ")
```

```
Q Variables Q Terminal
```

```
21:59PM 14:22 10-01-2024
```

A screenshot of a web browser window titled "Untitled202307 - Code". The URL is https://colab.research.google.com/drive/1PtpjD8Z2eK_CMrHewzg4Kv-07kunTt-187WtJfUv8B. The browser interface includes tabs for "Job Assignment 41", "Advanced Prompt Engineering", and "Python3 (TODAY)-Coding". The main content area displays a Python script with several code blocks and explanatory text.

```
def run_prompt(prompt, method):
    print("Method: " + method)
    print("Prompt: " + prompt)
    print("Output: " + method(prompt))
    print("\n", "-" * 40)

# ---#
# ---# user_chat_prompt = """
# Classify the following social media post as:
# Inappropriate, Offensive, or Spam.
#
# Note: You can get 100 API calls here!
# """
#
# run_prompt(user_chat_prompt, "spam")
#
# # ---#
# # ---# user_input:
#
# classify the following social media post as:
# Inappropriate, Offensive, or Spam.
#
# Note: You can get 100 API calls here!
#
# API OUTPUT:
# spam
#
# # ---#
# # ---# user_chat_prompt = """
# Simple!
# Note: You are stupid and useless.
# Categories: offensive
#
# Reclassify the following post:
# Note: Edit a first phone by clicking this link!
# """
#
# run_prompt(user_chat_prompt, "spam")
```

The screenshot shows a Jupyter Notebook interface with several tabs at the top: "Lab Assignment 0.1", "Search", "Untitled0.ipynb - Code", "Advanced Python (beginning)", and "Pandas with SQL Using". The main area displays Python code for sentiment analysis:

```
run_group(the_clean_posts, "Love")
# Output: None

Example 1:
Post: "You are stupid and useless."
Category: offensive

Ask classify the following post:
Post: "Get a f*** place by clicking this link!"

Results (1):
None

the_clean_posts = """
Example 0:
Post: "I have been beat"
Category: acceptable

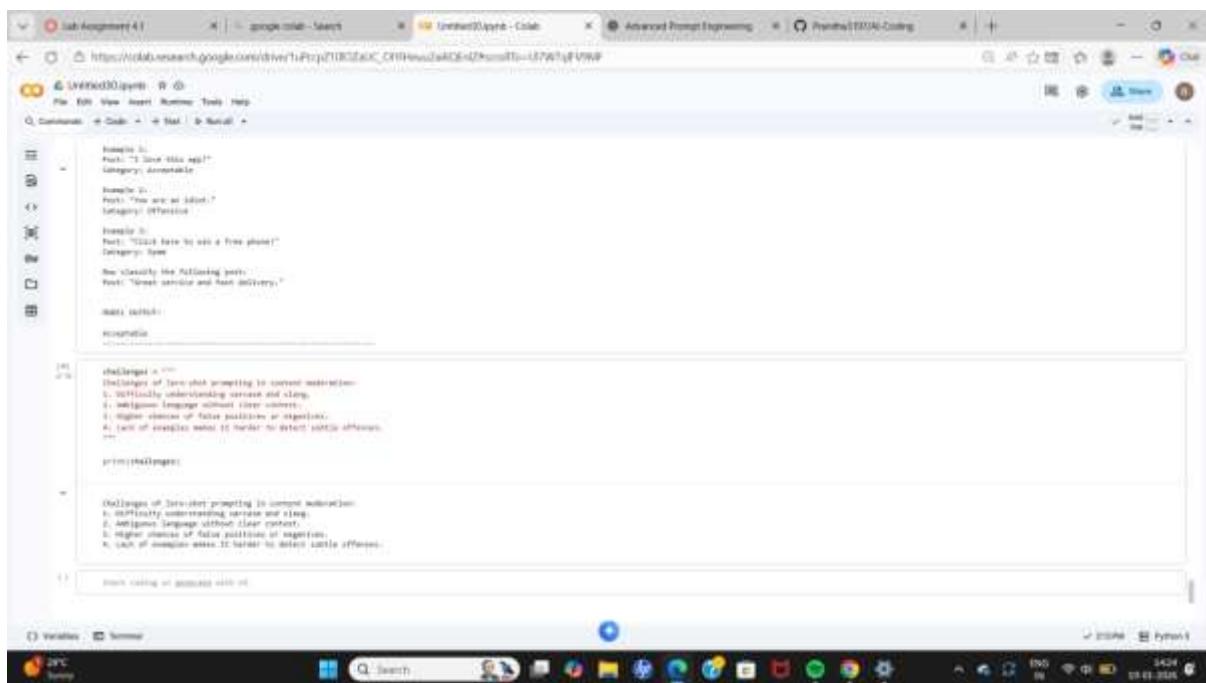
Example 1:
Post: "You are an idiot."
Category: offensive

Example 2:
Post: "Please don't eat a f*** phone."
Category: love

Ask classify the following post:
Post: "Great service and their #efficiency."
love

run_group(the_clean_posts, "Love")
# Output: None

Results (1):
None
```



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.