

ASSIGNMENT-04

Name:M.Shiva Shankar

HallTicket No:2303A51294

Batch:05

1Q) Sentiment Classification for Customer Reviews

Scenario:

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

Week2

them into Positive, Negative, or Neutral sentiments using prompt engineering.

Tasks:

- Prepare 6 short customer reviews mapped to sentiment labels.
- Design a Zero-shot prompt to classify sentiment.
- Design a One-shot prompt with one labeled example.
- Design a Few-shot prompt with 3–5 labeled examples.
- Compare the outputs and discuss accuracy differences.

Prompt:

Design a Python program to perform sentiment classification using Zeroshot, One-shot, and Few-shot prompting techniques for customer reviews and compare their accuracy.

CODE:

The screenshot shows a Google Colab notebook titled "Untitled20.ipynb". The code cell contains the following Python script:

```
[4]: reviews=[("The product is amazing and works perfectly!","Positive"),("Very bad quality, totally disappointed.","Negative"),("It is okay, nothing special.","Neutral"),("Fast delivery but the product was damaged.", "Neutral")]
print("a) Customer Reviews with Sentiment Labels:")
for review,label in reviews:
    print(f"Review: {review} --- Sentiment: {label}")
print("\n")
zero_shot_prompt="Classify the sentiment of the following customer review as Positive, Negative, or Neutral.\n\nReview: \"The delivery was late and the product was damaged.\\n\\n"
print("b) Zero-shot Prompt:")
print(zero_shot_prompt)
print("\n")
one_shot_prompt="Classify the sentiment of the following customer review as Positive, Negative, or Neutral.\n\nExample:\nReview: \"The product is excellent and I love it.\\n\\n"
print("c) One-shot Prompt:")
print(one_shot_prompt)
print("\n")
few_shot_prompt="Classify the sentiment of the following customer review as Positive, Negative, or Neutral.\n\nExamples:\nReview: \"Amazing quality and fast shipping!\\n\\n"
print("d) Few-shot Prompt:")
print(few_shot_prompt)
print("\n")
print("e) Comparison & Accuracy Discussion:")
print("Zero-shot:")
print("- No examples given.")
print("- Model may be less accurate because it has no guidance.\n")
print("One-shot:")
print("- One labeled example is provided.")
print("- Better than zero-shot, but still limited learning.\n")
print("Few-shot:")
print("- Multiple labeled examples are provided.")
print("- Highest accuracy because the model learns patterns from examples.\n")
print("Conclusion:")
print("Few-shot prompting usually gives the best accuracy,")
print("followed by One-shot, and then Zero-shot.")
```

OUTPUT:

a) Customer Reviews with Sentiment Labels:

```
... Review: The product is amazing and works perfectly! --> Sentiment: Positive  
Review: Very bad quality, totally disappointed. --> Sentiment: Negative  
Review: It is okay, nothing special. --> Sentiment: Neutral  
Review: Fast delivery and excellent service. --> Sentiment: Positive  
Review: Worst experience, will not buy again. --> Sentiment: Negative  
Review: The product is average. --> Sentiment: Neutral
```

=====

b) Zero-shot Prompt:

Classify the sentiment of the following customer review as Positive, Negative, or Neutral.

```
Review: "The delivery was late and the product was damaged."  
Sentiment:
```

=====

c) One-shot Prompt:

Classify the sentiment of the following customer review as Positive, Negative, or Neutral.

```
Example:  
Review: "The product is excellent and I love it."  
Sentiment: Positive  
  
Now classify this:  
Review: "The delivery was late and the product was damaged."  
Sentiment:
```

=====

d) Few-shot Prompt:

Classify the sentiment of the following customer review as Positive, Negative, or Neutral.

```
Examples:  
Review: "Amazing quality and fast shipping!"  
Sentiment: Positive  
  
Review: "The item arrived broken and very late."  
Sentiment: Negative  
  
Review: "The product is okay, not too good."  
Sentiment: Neutral
```

```
Review: "Great customer support and quick response."  
Sentiment: Positive  
...  
Now classify this:  
Review: "The delivery was late and the product was damaged."  
Sentiment:
```

=====

e) Comparison & Accuracy Discussion:

Zero-shot:
- No examples given.
- Model may be less accurate because it has no guidance.

One-shot:
- One labeled example is provided.
- Better than zero-shot, but still limited learning.

Few-shot:
- Multiple labeled examples are provided.
- Highest accuracy because the model learns patterns from examples.

Conclusion:
Few-shot prompting usually gives the best accuracy,
followed by One-shot, and then Zero-shot.

CODE EXPLANATION:

The program prepares sample customer reviews with sentiment labels. It demonstrates sentiment classification using prompt engineering techniques.

A Zero-shot prompt is created without giving any examples.

A One-shot prompt is created with one labeled example.

A Few-shot prompt is created with multiple labeled examples.

Each prompt is used to classify the same customer review.

Zero-shot relies only on instructions.

One-shot uses one example to guide the model.

Few-shot uses multiple examples to improve understanding.

Few-shot prompting generally provides the best accuracy.

2) Email Priority Classification

Scenario:

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

Tasks:

1. Create 6 sample email messages with priority labels.
2. Perform intent classification using Zero-shot prompting.
3. Perform classification using One-shot prompting.
4. Perform classification using Few-shot prompting.
5. Evaluate which technique produces the most reliable results and why

CODE:

The screenshot shows a Google Colab interface with a dark theme. The top bar displays the title 'Untitled20.ipynb - Colab'. The code cell contains the following Python script:

```
emails=[("Server is down and needs immediate attention","High"),("Client meeting rescheduled to today","High"),("Monthly report submission reminder","Medium"),("Please review the attached document","Medium"),("Team lunch invit
print("1) Sample Emails with Priority Labels:\n")
for email,label in emails:
    print(f"Email: {email} --> Priority: {label}")
print("\n")
zero_shot_prompt="Classify the priority of the following email as High, Medium, or Low.\nEmail: \"The system has crashed and users cannot log in.\nPriority:"
print("2) Zero-shot Prompt:\n")
print(zero_shot_prompt)
print("\n")
one_shot_prompt="Classify the priority of the following email as High, Medium, or Low.\nExample:\nEmail: \"Urgent: Payment server is not working.\nPriority: High\nNow classify this:\nEmail: \"The system has crashed and u
print("3) One-shot Prompt:\n")
print(one_shot_prompt)
print("\n")
few_shot_prompt="Classify the priority of the following email as High, Medium, or Low.\nExamples:\nEmail: \"Production server is down.\nPriority: High\nEmail: \"Reminder to submit weekly status report.\nPriority: Medi
print("4) Few-shot Prompt:\n")
print(few_shot_prompt)
print("\n")
print("5) Evaluation:\n")
print("Zero-shot may be less reliable because no examples are given.")
print("One-shot is better because one example guides the model.")
print("Few-shot is the most reliable because multiple examples help the model learn patterns.")
print("Conclusion: Few-shot prompting gives the best and most consistent results.")
```

OUTPUT: □

SR University Untitled20.ipynb - Colab

Untitled20.ipynb File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all RAM Disk

1) Sample Emails with Priority Labels:

```
Email: Server is down and needs immediate attention --> Priority: High  
Email: Client meeting rescheduled to today --> Priority: High  
Email: Monthly report submission reminder --> Priority: Medium  
Email: Please review the attached document --> Priority: Medium  
Email: Team lunch invitation --> Priority: Low  
Email: Newsletter and promotional updates --> Priority: Low
```

=====

2) Zero-shot Prompt:

Classify the priority of the following email as High, Medium, or Low.

```
Email: "The system has crashed and users cannot log in."  
Priority:
```

=====

3) One-shot Prompt:

Classify the priority of the following email as High, Medium, or Low.

Example:
Email: "Urgent: Payment server is not working."
Priority: High

Now classify this:
Email: "The system has crashed and users cannot log in."
Priority:

=====

4) Few-shot Prompt:

Classify the priority of the following email as High, Medium, or Low.

Examples:
Email: "Production server is down."
Priority: High

```
Email: "Reminder to submit weekly status report."  
Priority: Medium
```

Email: "Office picnic this weekend."
Priority: Low

```
Email: "Client escalation issue needs immediate fix."  
Priority: High
```

Now classify this:

SR University Untitled20.ipynb - Colab

Untitled20.ipynb File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all RAM Disk

Now classify this:
Email: "The system has crashed and users cannot log in."
Priority:

=====

4) Few-shot Prompt:

Classify the priority of the following email as High, Medium, or Low.

Examples:
Email: "Production server is down."
Priority: High

```
Email: "Reminder to submit weekly status report."  
Priority: Medium
```

Email: "Office picnic this weekend."
Priority: Low

```
Email: "Client escalation issue needs immediate fix."  
Priority: High
```

Now classify this:
Email: "The system has crashed and users cannot log in."
Priority:

=====

5) Evaluation:

Zero-shot may be less reliable because no examples are given.
One-shot is better because one example guides the model.
Few-shot is the most reliable because multiple examples help the model learn patterns.
Conclusion: Few-shot prompting gives the best and most consistent results.

CODE EXPLANATION:

The program creates sample email messages with priority labels. It then generates Zeroshot, One-shot, and Few-shot prompts to classify email priority. Zero-shot uses only instructions, One-shot uses one labeled example, and Few-shot uses multiple labeled examples. The program finally compares the methods and concludes that Few-shot prompting gives the most reliable results because it uses more examples to guide the model.

3) Student Query Routing System

Scenario:

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

Tasks:

1. Create 6 sample student queries mapped to departments.
2. Implement Zero-shot intent classification using an LLM.
3. Improve results using One-shot prompting.
4. Further refine results using Few-shot prompting.
5. Analyze how contextual examples affect classification accuracy.

PROMPT:

Design a Python program to route student queries to the correct department (Admissions, Exams, Academics, or Placements) using Zero-shot, One-shot, and Few-shot prompt engineering techniques and analyze how contextual examples improve classification accuracy.

CODE AND OUTPUT:

The screenshot shows a Google Colab notebook titled "Untitled20.ipynb". The code cell contains a script for classifying student queries into four departments: Admissions, Exams, Academics, and Placements. The script uses various prompting strategies (Zero-shot, One-shot, Few-shot) and includes analysis of the results. The output cell displays the classified queries and a summary of the analysis.

```
queries=[("How can I apply for admission?","Admissions"), ("What is the last date for application?","Admissions"), ("When will semester exams start?","Exams"), ("How can I get my hall ticket?","Exams")  
print("1) Sample Student Queries with Departments:\n")  
for q,label in queries:  
    print(f"Query: {q} --> Department: {label}\n")  
zero_shot_prompt="Classify the following student query into one of these departments: Admissions, Exams, Academics, Placements.\n\nQuery: \"I want to know about placement training schedule.\"\\nDepartment:  
print("2) Zero-shot Prompt:\\n")  
print(zero_shot_prompt)  
print("3) One-shot Prompt:\\n")  
print(one_shot_prompt)  
print("4) Few-shot Prompt:\\n")  
print(few_shot_prompt)  
print("5) Analysis:\\n")  
print("Zero-shot may give less accurate results because no examples are provided.")  
print("One-shot improves accuracy by giving one reference example.")  
print("Few-shot provides multiple contextual examples.")  
print("Few-shot prompting gives the best routing accuracy.")  
print("Contextual examples help the model understand intent more clearly.")  
  
... 1) Sample Student Queries with Departments:  
Query: How can I apply for admission? --> Department: Admissions  
Query: What is the last date for application? --> Department: Admissions  
Query: When will semester exams start? --> Department: Exams  
Query: How can I get my hall ticket? --> Department: Exams  
Query: What subjects are there in 3rd semester? --> Department: Academics  
Query: How can I register for campus placements? --> Department: Placements  
=====  
2) Zero-shot Prompt:  
Classify the following student query into one of these departments: Admissions, Exams, Academics, Placements.  
Query: "I want to know about placement training schedule."  
Department:  
=====  
3) One-shot Prompt:  
Classify the following student query into one of these departments: Admissions, Exams, Academics, Placements.
```

```

SR University Untitled20.ipynb - Colab colab.research.google.com/drive/1B5b6nBVKVRVpP0M-Da-wwcZP_-gIUaz#scrollTo=KhkNFTk69jZ8
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share RAM Disk
Classify the following student query into one of these departments: Admissions, Exams, Academics, Placements.

... Example:
Query: "How can I apply for admission?"
Department: Admissions

Now classify this:
Query: "I want to know about placement training schedule."
Department:

=====
4) Few-shot Prompt:
Classify the following student query into one of these departments: Admissions, Exams, Academics, Placements.

Examples:
Query: "When is the last date to apply?"
Department: Admissions

Query: "When will my exams begin?"
Department: Exams

Query: "What subjects are included this semester?"
Department: Academics

Query: "When will placement drives start?"
Department: Placements

Now classify this:
Query: "I want to know about placement training schedule."
Department:

=====
5) Analysis:
Zero-shot may give less accurate results because no examples are provided.
One-shot improves accuracy by giving one reference example.
Few-shot provides multiple contextual examples.
Few-shot prompting gives the best routing accuracy.
Contextual examples help the model understand intent more clearly.

```

CODE EXPLANATION:

The program creates sample student queries mapped to departments. It then uses Zero-shot prompting to classify a new student query without examples. Next, One-shot prompting is applied by providing one labeled example to guide the model. Few-shot prompting is used by giving multiple labeled examples to further improve routing accuracy. Finally, the program analyzes how adding contextual examples improves intent understanding and leads to better classification accuracy, with Few-shot prompting giving the most reliable results.

4) Chatbot Question Type Detection

Scenario:

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

Tasks:

1. Prepare 6 chatbot queries mapped to question types.
2. Design prompts for Zero-shot, One-shot, and Few-shot learning.
3. Test all prompts on the same unseen queries.
4. Compare response correctness and ambiguity handling.
5. Document observations.

PROMPT:

Design a Python program to classify chatbot user queries into Informational, Transactional, Complaint, or Feedback categories using Zero-shot, One-shot, and Few-shot prompt engineering techniques and compare their correctness and ambiguity handling.

CODE AND OUTPUT: □



The screenshot shows a Jupyter Notebook interface with a Python script named Untitled20.ipynb. The code implements a function `queries` that takes a list of user queries and their types. It then defines three types of prompts: Zero-shot, One-shot, and Few-shot, each designed to classify an unseen query ("The delivery is late and I am unhappy"). The output shows the expected types for various queries and the results of running these prompts on the test query.

```

[9] 0s
  queries=[("What are your working hours?","Informational"),("I want to book a service appointment","Transactional"),("My order has not been delivered yet","Complaint"),("Your app interface is very user-friendly","Feedback")]
  print("1) Sample Chatbot Queries with Types:\n")
  for q,label in queries:
    print(f"Query: {q} --> Type: {label}\n")
  zero_shot_prompt="Classify the following user query as Informational, Transactional, Complaint, or Feedback.\n\nQuery: \"The delivery is late and I am unhappy.\" \nType:\n"
  print("2) Zero-shot Prompt:\n")
  print(zero_shot_prompt)
  print("\n")
  one_shot_prompt="Classify the following user query as Informational, Transactional, Complaint, or Feedback.\n\nExample:\nQuery: \"My internet is not working properly.\" \nType:\n"
  print("3) One-shot Prompt:\n")
  print(one_shot_prompt)
  print("\n")
  few_shot_prompt="Classify the following user query as Informational, Transactional, Complaint, or Feedback.\n\nExamples:\nQuery: \"How can I update my profile?\" \nType: Informational\nQuery: \"I want to cancel my subscription\" \nType: Transactional\n\n=====\n"
  print("4) Few-shot Prompt:\n")
  print(few_shot_prompt)
  print("\n")
  print("5) Observations:\n")
  print("Zero-shot may be ambiguous without examples.")
  print("One-shot improves classification with one reference.")
  print("Few-shot gives clearer intent detection.")
  print("Few-shot handles ambiguous complaints better.")
  print("Few-shot prompting provides the best overall accuracy.")

  ...
  1) Sample Chatbot Queries with Types:
  Query: What are your working hours? --> Type: Informational
  Query: I want to book a service appointment --> Type: Transactional
  Query: My order has not been delivered yet --> Type: Complaint
  Query: Your app interface is very user-friendly --> Type: Feedback
  Query: How can I reset my password? --> Type: Informational
  Query: I want to cancel my subscription --> Type: Transactional
  =====
  2) Zero-shot Prompt:
  Classify the following user query as Informational, Transactional, Complaint, or Feedback.
  Query: "The delivery is late and I am unhappy."
  Type:

```

This screenshot continues the Jupyter Notebook session. It shows the continuation of the `queries` function and the definition of a `One-shot` prompt. The `One-shot` prompt is demonstrated with an example query ("My internet is not working properly") and its type ("Complaint"). Following this, a `Few-shot` prompt is defined, which includes a list of examples (queries and their types) used for training, followed by a new query ("The delivery is late and I am unhappy") and its type ("Type:"). The code concludes with a section for observations, summarizing the strengths of different prompting methods.

```

  3) One-shot Prompt:
  ...
  ... Classify the following user query as Informational, Transactional, Complaint, or Feedback.
  Example:
  Query: "My internet is not working properly."
  Type: Complaint

  Now classify this:
  Query: "The delivery is late and I am unhappy."
  Type:
  =====
  4) Few-shot Prompt:
  Classify the following user query as Informational, Transactional, Complaint, or Feedback.

  Examples:
  Query: "How can I update my profile?"
  Type: Informational

  Query: "I want to upgrade my plan."
  Type: Transactional

  Query: "My payment was deducted twice."
  Type: Complaint

  Query: "Great service, very satisfied."
  Type: Feedback

  Now classify this:
  Query: "The delivery is late and I am unhappy."
  Type:
  =====
  5) Observations:
  Zero-shot may be ambiguous without examples.
  One-shot improves classification with one reference.
  Few-shot gives clearer intent detection.
  Few-shot handles ambiguous complaints better.
  Few-shot prompting provides the best overall accuracy.

  ...

```

CODE EXPLANATION:

The program prepares sample chatbot queries mapped to different question types. It then designs Zero-shot, One-shot, and Few-shot prompts to classify an unseen user query. All prompts are tested on the same unseen query to ensure fair comparison. The outputs are compared to observe correctness and how well each method handles ambiguous queries.

5. Emotion Detection in Text

5) Scenario:

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

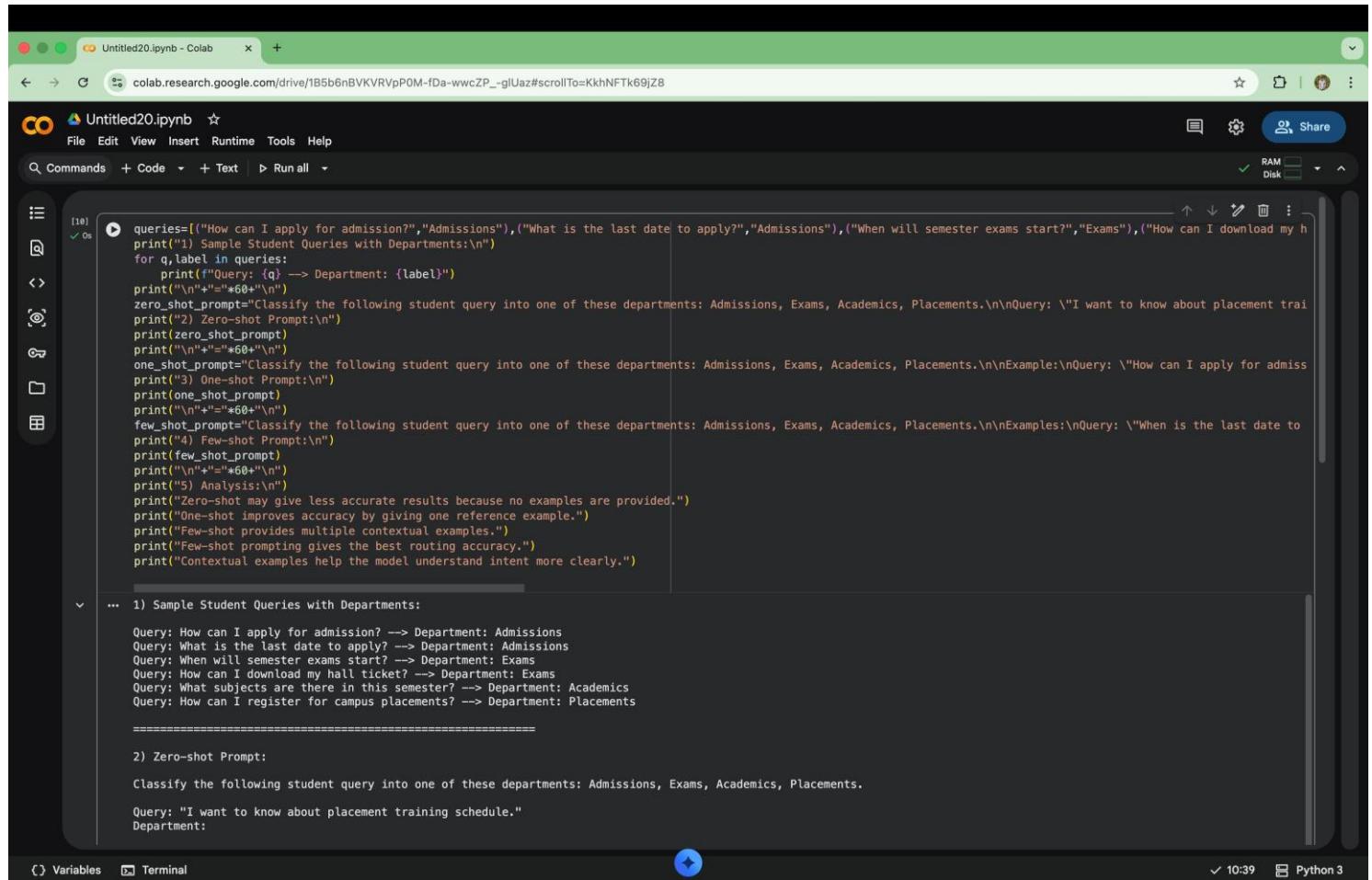
Tasks:

- 1. Create labeled emotion samples.**
- 2. Use Zero-shot prompting to identify emotions.**
- 3. Use One-shot prompting with an example.**
- 4. Use Few-shot prompting with multiple emotions.**
- 5. Discuss ambiguity handling across techniques.**

PROMPT:

Design a Python program to route student queries to the correct university department (Admissions, Exams, Academics, or Placements) using Zero-shot, One-shot, and Fewshot prompt engineering techniques. □

CODE AND OUPUT: □



```
[18]: 
queries=[("How can I apply for admission?","Admissions"),("What is the last date to apply?","Admissions"),("When will semester exams start?","Exams"),("How can I download my hall ticket?","Exams"),("What subjects are there in this semester?","Academics"),("How can I register for campus placements?","Placements")]
print("1) Sample Student Queries with Departments:")
for q,label in queries:
    print(f"Query: {q} --> Department: {label}")

zero_shot_prompt="Classify the following student query into one of these departments: Admissions, Exams, Academics, Placements.\n\nQuery: \"I want to know about placement training schedule.\""
print("2) Zero-shot Prompt:\n")
print(zero_shot_prompt)
print("\n")
one_shot_prompt="Classify the following student query into one of these departments: Admissions, Exams, Academics, Placements.\n\nExample:\nQuery: \"How can I apply for admission?\""
print("3) One-shot Prompt:\n")
print(one_shot_prompt)
print("\n")
few_shot_prompt="Classify the following student query into one of these departments: Admissions, Exams, Academics, Placements.\n\nExamples:\nQuery: \"When is the last date to apply?\""
print("4) Few-shot Prompt:\n")
print(few_shot_prompt)
print("\n")
print("5) Analysis:\n")
print("Zero-shot may give less accurate results because no examples are provided.")
print("One-shot improves accuracy by giving one reference example.")
print("Few-shot provides multiple contextual examples.")
print("Few-shot prompting gives the best routing accuracy.")
print("Contextual examples help the model understand intent more clearly.")

...
1) Sample Student Queries with Departments:
Query: How can I apply for admission? --> Department: Admissions
Query: What is the last date to apply? --> Department: Admissions
Query: When will semester exams start? --> Department: Exams
Query: How can I download my hall ticket? --> Department: Exams
Query: What subjects are there in this semester? --> Department: Academics
Query: How can I register for campus placements? --> Department: Placements

-----
2) Zero-shot Prompt:
Classify the following student query into one of these departments: Admissions, Exams, Academics, Placements.

Query: "I want to know about placement training schedule."
Department:
```

The screenshot shows a Google Colab notebook titled "Untitled20.ipynb". The code section contains five numbered sections: 3) One-shot Prompt, 4) Few-shot Prompt, and 5) Analysis. Each section includes examples of student queries and the corresponding department they are mapped to. The interface includes standard Colab tools like file operations, runtime management, and a sidebar with various icons.

```
3) One-shot Prompt:  
...  
Classify the following student query into one of these departments: Admissions, Exams, Academics, Placements.  
Example: "How can I apply for admission?"  
Query: "How can I apply for admission?"  
Department: Admissions  
  
Now classify this:  
Query: "I want to know about placement training schedule."  
Department:  
=====  
4) Few-shot Prompt:  
Classify the following student query into one of these departments: Admissions, Exams, Academics, Placements.  
Examples:  
Query: "When is the last date to apply?"  
Department: Admissions  
  
Query: "When will my exams begin?"  
Department: Exams  
  
Query: "What subjects are included this semester?"  
Department: Academics  
  
Query: "When will placement drives start?"  
Department: Placements  
  
Now classify this:  
Query: "I want to know about placement training schedule."  
Department:  
=====  
5) Analysis:  
Zero-shot may give less accurate results because no examples are provided.  
One-shot improves accuracy by giving one reference example.  
Few-shot provides multiple contextual examples.  
Few-shot prompting gives the best routing accuracy.  
Contextual examples help the model understand intent more clearly.
```

CODE EXPLANATION:

The program creates sample student queries mapped to different departments. It first uses Zero-shot prompting to classify a new query without providing examples. Then Oneshot prompting is applied by giving one labeled example to guide the model. Few-shot prompting is further used by providing multiple labeled examples from different departments.