Course: AI Assisted Coding

LAB EXAM Total – 30M

Hall.no:2403A52106

Name:N.Jyoshna Sri

Set 1

1. Use **zero-shot prompting** in ChatGPT to generate a Python function that validates email addresses.

Run and test the function in Google Colab.

Prompt:

Open ChatGPT → Give a clear zero-shot prompt like *"Write a Python function to validate email*

Format"

Code:

```
import re


def validate_email(email):
    """
    Validate email using regex.
    Returns True if valid, False otherwise.
    """
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    return re.match(pattern, email) is not None


# ----------------------
```

```
# Test the function
# ----------------------

test_emails = [
    "test@example.com",
    "user.name@domain.co.in",
    "abc@.com",
    "hello@gmail",
    "valid_email123@gmail.com",
    "invalid-email"
]

print("Email Validation Results:\n")
for email in test_emails:
    print(f"{email:35} → {validate_email(email)}")
```

output:

Email Validation Results:

test@example.com → True

user.name@domain.co.in → True

abc@.com → False

hello@gmail → False

valid_email123@gmail.com → True

invalid-email → False

Explanation: **1. The validate_email function:**

This function takes an email address as input and uses a regular expression (pattern) to check if it matches a common email format.

- ^[a-zA-Z0-9._%+-]+: Matches the part before the @ symbol, allowing letters, numbers, and common special characters.
- @[a-zA-Z0-9.-]+: Matches the @ symbol followed by the domain name, allowing letters, numbers, and hyphens.
- \.[a-zA-Z]{2,}$: Matches the dot (.) followed by the top-level domain (like .com, .org), which must be at least two letters long.

It returns True if the email matches the pattern (is valid), and False otherwise.

2. The test_emails list:

This list contains several email addresses that are used to test the validate_email function. It includes examples of both valid and invalid email formats.

3. The Output (Email Validation Results):

The code then iterates through each email in the test_emails list, calls validate_email on it, and prints the email along with the True/False result:

- test@example.com → True: This is a standard, valid email format.
- user.name@domain.co.in → True: This also follows a valid format, including a subdomain and a two-part top-level domain.

- abc@.com → False: Invalid because the domain part (.com) immediately follows the @ without a proper domain name (e.g., abc@example.com).
- hello@gmail → False: Invalid because it lacks a top-level domain (like .com, .org).
- valid_email123@gmail.com → True: Another perfectly valid email format.
- invalid-email → False: Invalid because it's missing both the @ symbol and a domain part.

2. Use **GitHub Copilot** to suggest improvements to your function's readability and refactor it based

on the suggestions.

Prompt: Paste your function → Type a comment like **“# improve readability”**.

Code:

"""Run a quick demo comparing the messy and refactored functions.

This script creates a small temporary CSV, runs both implementations

and prints their output so you can compare results.

"""

```
import csv
import os
from pathlib import Path
from example_before import messy_process_csv
```

```
from example_refactored import process_csv


def make_sample_csv(path: str) -> None:
    rows = [
        ['id', 'value'],
        ['a', '1'],
        ['b', '2.0'],
        ['c', '3'],
        ['d', 'not_a_number'],
        ['e', '4.5'],
    ]
    with open(path, 'w', newline='') as fh:
        writer = csv.writer(fh)
        writer.writerows(rows)


def main():
    tmp = Path(__file__).parent / 'sample.csv'
    make_sample_csv(str(tmp))
    print('--- messy_process_csv output ---')
    print(messy_process_csv(str(tmp)))
    print('\n--- refactored process_csv output ---')
    print(process_csv(str(tmp)))
    try:
        os.remove(tmp)
```

```
    except Exception:
        pass


if __name__ == '__main__':
    main()
```

output:

--- refactored process_csv output ---

{'count': 4, 'sum': 10.5, 'avg': 2.625, 'min': 1.0, 'max': 4.5, 'median': 2.5}

PS C:\Users\HP\OneDrive\Desktop\lab test 5>

Explanation:

- **Files added:** example_before.py, example_refactored.py, run_examples.py, README.md.
- **What changed:** added # improve readability to the messy file and created a refactored version with small helpers.
- **Key improvements: Separation of concerns:** split reading vs. calculation. **Safer I/O:** uses with open(...). **Clear errors:** only catches ValueError when parsing. **Types/docs:** type hints and docstrings.
- **Expected result:** both implementations produce the same stats for valid rows (bad rows skipped).