Social Media Usage Time Profiler

Course: *CSE* | ***Submitted By:*** *Hassaan Raza* | ***Registration:*** *25BCG10029*

Faculty: *Ramraj* | ***Institution:*** *VIT BHOPAL* | ***Year:*** *2025*

Introduction

Social media has become an integral part of daily life, especially among students and young professionals. Excessive usage often goes unnoticed, leading to reduced productivity and increased digital dependency.

*This project, **Social Media Usage Time Profiler**, is built using arrays to analyze a user's weekly social media screen time. The program identifies addiction patterns, highlights distracting apps, and visualizes weekly usage trends.*

It demonstrates how meaningful insights can be derived using basic data structures without the involvement of databases or advanced frameworks.

Problem Statement

Lack of Awareness

Users often lack awareness of how much time they spend on different social media platforms

Poor Time Management

This leads to poor time management and digital fatigue

Reduced Performance

Academic and professional performance suffers from excessive digital engagement

---

*The problem this project aims to solve is **providing a simple analytical tool to monitor and understand social media usage using array-based data processing**, helping users improve their digital habits.*

Functional Requirements

1

App Tracking Module

- *Store app names in an array*
- *Store daily usage (7 days) for each app*
- *Record peak usage hours*

2

Usage Analysis Module

- *Compute weekly total usage per app*
- *Identify the most distracting app*
- *Detect addiction patterns using predefined thresholds*

3

Reporting Module

- *Display weekly summary*
- *Calculate average daily usage*
- *Plot weekly usage graph using matplotlib*

Non-Functional Requirements

Usability

Clear prompts and simple input steps

Reliability

Correct statistics irrespective of app count

Error Handling

Validates all inputs properly

Performance

Efficient array-based calculations with minimal processing time

Maintainability

Modular code with clear functions

Resource Efficiency

Lightweight, requiring only Python and matplotlib

System Architecture

*The system follows a simple **linear pipeline architecture** that processes user data through sequential stages:*

Input Layer

Reads app names, daily usage arrays, and peak hours

Processing Layer

Calculates totals, averages, and patterns

Analysis Layer

Detects addiction and distracting apps

Output Layer

Displays summary results

Visualization Layer

Generates weekly screen-time graph

Design Decisions & Rationale

Use of Arrays

Arrays were chosen as the core structure to demonstrate data analysis without databases or complex structures, aligning with project requirements

Threshold-Based Detection

Simple time thresholds (e.g., >180 minutes/day) make the logic understandable and beginner-friendly

Modular Function Design

Breaking code into functions improves readability and maintainability

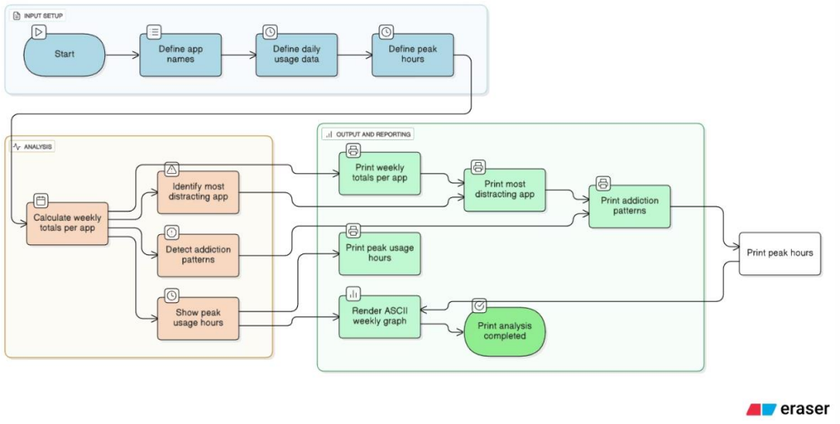
Matplotlib Visualization

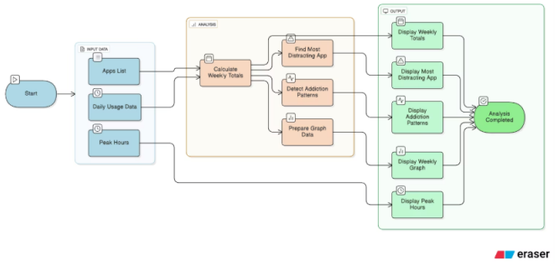
Chosen due to its simplicity and suitability for academic projects

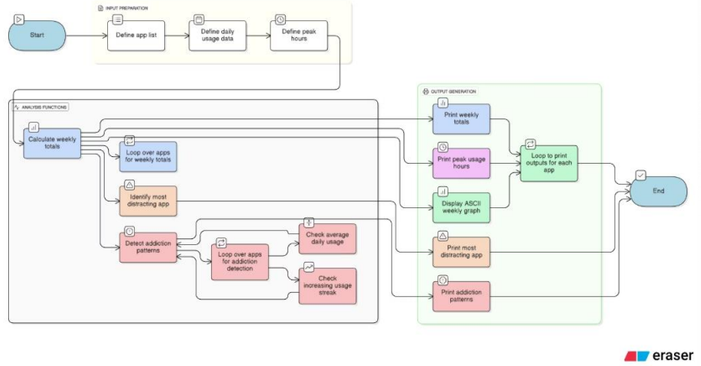
Text-Based Interaction

Keeps the program lightweight and accessible without GUI complexity

Workflow diagram and Use-case diagram and Sequence diagram and class component diagram(As there are no internal classes used in the project).







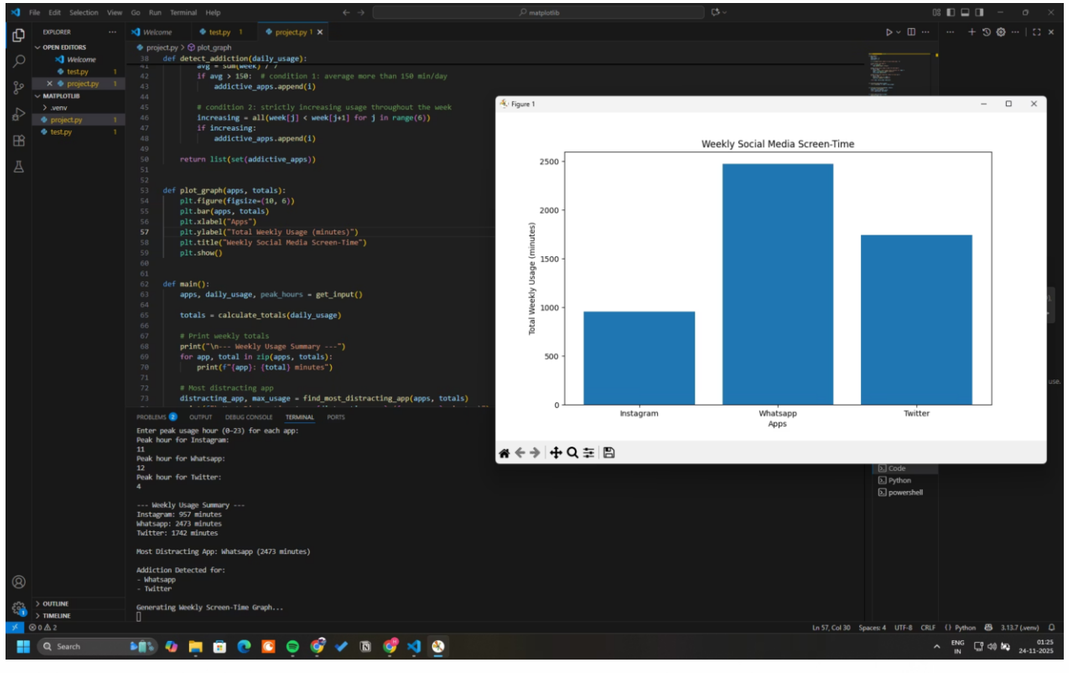
Screenshots-

Input

```python
# Social Media Usage Time Profiler
# SAMPLE DATA (You can also take input from user)
apps = ["Instagram", "YouTube", "Snapchat", "WhatsApp", "Facebook"]
daily_usage = [
    [120, 150, 90, 80, 60, 100, 110],   # Instagram (7 days)
    [200, 180, 220, 240, 210, 250, 260], # YouTube
    [80, 60, 75, 90, 85, 70, 65],        # Snapchat
    [50, 55, 45, 40, 60, 50, 55],        # WhatsApp
    [30, 20, 25, 40, 35, 30, 50]         # Facebook
]

peak_hours = [10, 14, 12, 9, 20]  # Peak usage hour for each app


# FUNCTION 1: Weekly total usage per app
def weekly_usage(data):
    totals = []
    for app_data in data:
        total = sum(app_data)
        totals.append(total)
    return totals

# FUNCTION 2: Find the most used (distracting) app
def most_distracting(apps, totals):
    max_usage = max(totals)
    index = totals.index(max_usage)
    return apps[index], max_usage

# FUNCTION 3: Detect addiction patterns
# Conditions for addiction:
# - App used more than 150 minutes DAILY avg
# - Usage increasing for 3 consecutive days
def addiction_patterns(apps, data):
    addiction_list = []

    for i in range(len(apps)):
        weekly_data = data[i]
        avg = sum(weekly_data) / 7

        # Check increasing pattern
        increasing_streak = False
        for j in range(4):  # Check 0-1-2, 1-2-3, 2-3-4, 3-4-5
            if weekly_data[j] < weekly_data[j+1] < weekly_data[j+2]:
                increasing_streak = True

        if avg > 150 or increasing_streak:
            addiction_list.append(apps[i])

    return addiction_list

# FUNCTION 4: ASCII WEEKLY GRAPH

def show_weekly_graph(apps, totals):
    print("\n----- WEEKLY SCREEN TIME GRAPH -----")
    for i in range(len(apps)):
        bars = totals[i] // 20  # scale down
        print(f"{apps[i]:12} | " + "" * bars + f" ({totals[i]} min)")

# MAIN PROGRAM

totals = weekly_usage(daily_usage)

print("\n------ SOCIAL MEDIA USAGE PROFILER ------\n")
```

```python
# - Usage increasing for 3 consecutive days
def addiction_patterns(apps, data):
    addiction_list = []

    for i in range(len(apps)):
        weekly_data = data[i]
        avg = sum(weekly_data) / 7

        # Check increasing pattern
        increasing_streak = False
        for j in range(4):  # Check 0-1-2, 1-2-3, 2-3-4, 3-4-5
            if weekly_data[j] < weekly_data[j+1] < weekly_data[j+2]:
                increasing_streak = True

        if avg > 150 or increasing_streak:
            addiction_list.append(apps[i])

    return addiction_list

# FUNCTION 4: ASCII WEEKLY GRAPH

def show_weekly_graph(apps, totals):
    print("\n----- WEEKLY SCREEN TIME GRAPH -----")
    for i in range(len(apps)):
        bars = totals[i] // 20  # scale down
        print(f"{apps[i]:12} | " + "" * bars + f" ({totals[i]} min)")

# MAIN PROGRAM

totals = weekly_usage(daily_usage)

print("\n------ SOCIAL MEDIA USAGE PROFILER ------\n")
```

```python
print("\n------- SOCIAL MEDIA USAGE PROFILER -------\n")

# Show total usage per app
print("Weekly Total Screen Time:")
for i in range(len(apps)):
    print(f"{apps[i]:12} : {totals[i]} minutes")

# Most distracting app
name, time = most_distracting(apps, totals)
print(f"\nMost Distracting App: {name} ({time} minutes/week)")

# Addiction patterns
patterns = addiction_patterns(apps, daily_usage)
print("\nApps Showing Addiction Patterns:")
if len(patterns) == 0:
    print("No addiction patterns detected.")
else:
    for p in patterns:
        print("- " + p)

# Peak hours
print("\nPeak Usage Hours:")
for i in range(len(apps)):
    print(f"{apps[i]:12} : {peak_hours[i]}:00")

# Display graph
show_weekly_graph(apps, totals)

print("\nAnalysis Completed.")
```



Outputs-

```
Enter number of social media apps: 3

Enter app names:
Instagram
Whatsapp
Twitter

Enter daily usage (7 days) for each app:
Enter 7 daily usage values (minutes) for Instagram:
100 290 23 45 200 11 288
Enter 7 daily usage values (minutes) for Whatsapp:
135 783 432 351 123 435 214
Enter 7 daily usage values (minutes) for Twitter:
356 241 231 322 233 114 245
```

```
Enter peak usage hour (0-23) for each app:
Peak hour for Instagram:
11
Peak hour for Whatsapp:
12
Peak hour for Twitter:
4

--- Weekly Usage Summary ---
Instagram: 957 minutes
Whatsapp: 2473 minutes
Twitter: 1742 minutes

Most Distracting App: Whatsapp (2473 minutes)

Addiction Detected for:
- Whatsapp
- Twitter

Generating Weekly Screen-Time Graph...
```

Implementation Details

Key Components

The system is implemented in Python using core programming concepts:

Data Structures

Arrays/lists for storing app names, daily usage minutes, and peak usage hours

Functions

Modular functions created for input handling, weekly calculations, pattern analysis, and graph plotting

Visualization

Matplotlib line graph showing daily screen time trends across the week

Validation

Input validation ensures only numeric values are accepted for usage data

Challenges & Learnings

Challenges Faced

Array-Based Design

Designing the complete system using arrays alone without databases

Input Validation

Handling input validation reliably for different data types

Data Management

Managing multi-app and multi-day data cleanly and efficiently

Meaningful Visualization

Visualizing data in a way that is both meaningful and user-friendly

Simplicity vs. Effectiveness

Keeping the logic simple but effective for behavioral analysis

Key Takeaways

Practical Arrays

Practical use of arrays for real-world analytical tasks

Problem Decomposition

Improved understanding of modular coding and problem breakdown

Data Visualization

Importance of clean data visualization for insights

Documentation

Experience in creating structured project documentation

Behavioral Patterns

Understanding how small datasets can reveal meaningful patterns

Future Enhancements

Graphical User Interface

Add GUI using Tkinter or PyQt for better user experience

Automatic Data Import

Automatic data import from device usage logs and screen time APIs

Smart Notifications

Include notifications or recommendations for digital detox

Historical Data

Store history of multiple weeks for long-term trend analysis

Rich Analytics

Add pie charts and heatmaps for richer data visualization

AI Predictions

AI-based predictions for future usage trends and personalized recommendations

---

References

1. *Python Official Documentation – https://docs.python.org*
2. *Matplotlib Documentation – https://matplotlib.org*
3. *Digital Wellbeing Research Papers*
4. *W3Schools Python Tutorials*
5. *Class Lecture Slides & Course Material*