Project Documentation

Project Title

Food Wastage Management System for College Hostels

Problem Statement (in Abstract)

College hostels often face issues of excessive food preparation and uneven distribution, leading to significant food wastage. This project addresses this issue by building a system that collects food consumption and wastage data across hostel blocks and provides actionable insights to minimize food waste using data visualization and smart suggestions.

Index

Description (Detailed)

- The project implements a data-driven solution to monitor and reduce food wastage in college hostels.

- Data is collected for over 2 years, including food prepared, consumed, and wasted along with block-wise servings .

- A modular and menu-driven application is developed using Python and GUI using Tkinter, allowing users to analyze food wastage daily, weekly, or monthly.

- Visualizations such as line and bar graphs are used to show trends and compare waste levels between blocks.

- A smart suggestion engine analyzes patterns and gives personalized suggestions per block and time period (daily/weekly/monthly).

- GUI integrates all modules and supports easy navigation, comparison, and visual interpretation.

- Purpose: Provide actionable insights to reduce over-preparation, encourage awareness, and promote efficient mess operations.

- Outcome: Better food planning, lower waste, improved satisfaction, and data transparency for hostel management and staff.

Data and Output Information

- Input:

  - Food data: Date, Meal, Prepared_kg, Consumed_kg, Wasted_kg

  - Serving data: Date, Block, Portion_Type, Prepared_kg, Consumed_kg, Wasted_kg

- Output:

  - Daily/weekly/monthly waste charts

  - Block-wise comparisons

  - Smart suggestions and messages

Solution Plan

- Gather historical food data and block-wise serving information.

- Preprocess and analyze using Pandas.

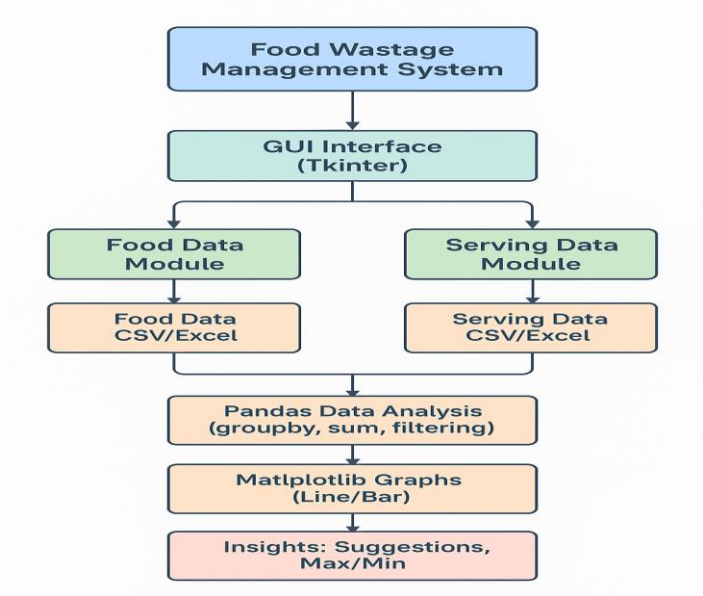
- Design modular Python code for each analysis.

- Visualize trends using Matplotlib.

- Integrate with Tkinter-based GUI.

- Include a smart suggestion .

Design (Diagrams)



- System Architecture: User → GUI → Analysis Modules → Data → Visual Output

- Data Flow Diagram: Data In → Processing (group, filter) → Output Graphs → Suggestions

Implementation

- Modular Scripts: food_data.py and serving_data.py

- Interactive GUI: main_gui.py with buttons for each module

- Visual Analytics: Daily, Weekly, and Monthly graphs

- Suggestions: Textual prompts in console.

project

├─── data/

│ ├─── food_data.py →food_data.csv

│ ├─── serving_data.py →serving_data.csv

│ └─── __init__.py

├─── gui/

│ └─── main_gui.py

│ └─── __init__.py

Code & Explanation

1.Code For food_data.py

```
import pandas as pd
import matplotlib.pyplot as plt

# Reads food data from a CSV file and parses the 'Date' column into datetime format
def read_food_data():
    file_path = "C:\\Users\\admin\\Downloads\\food_data_2years.csv"
    df = pd.read_csv(file_path)
    df['Date'] = pd.to_datetime(df['Date'])  # Convert 'Date' column to datetime
    return df

# Allows the user to input a custom date range and analyzes daily waste
def analyze_daily_waste_user_input():
    df = read_food_data()
    print(f"\nAvailable data: {df['Date'].min().date()} to {df['Date'].max().date()}")
    start = input("Enter start date (YYYY-MM-DD): ")
    end = input("Enter end date (YYYY-MM-DD): ")

    try:
        start_date = pd.to_datetime(start)
        end_date = pd.to_datetime(end)
    except:
        print("Invalid date format. Please enter in YYYY-MM-DD.")
        return
```

```
    # Filter the data to only include dates within the user-specified range
    filtered = df[(df['Date'] >= start_date) & (df['Date'] <= end_date)]
    if filtered.empty:
        print("No data found for this range.")
        return

    # Group the data by date to calculate total waste per day
    daily = filtered.groupby('Date')['Wasted_kg'].sum()
    avg_waste = daily.mean()

    max_date = daily.idxmax()  # Date with highest waste
    min_date = daily.idxmin()  # Date with lowest waste

    # Plot daily waste as a line graph
    plt.figure(figsize=(10, 4))
    plt.plot(daily.index, daily.values, color='orange', linewidth=2, label='Daily Waste')
    plt.scatter([max_date], [daily[max_date]], color='red', label=f"Highest:
{daily[max_date]:.1f} kg")
    plt.scatter([min_date], [daily[min_date]], color='green', label=f"Lowest:
{daily[min_date]:.1f} kg")
    plt.title(f"Daily Food Waste: {start} to {end}")
    plt.xlabel("Date")
    plt.ylabel("Wasted (kg)")
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
```

```
    plt.show()

    # Display smart suggestion based on average daily waste
    print("\n--- Smart Suggestion ---")
    if avg_waste > 30:
        print("High average daily waste detected.")
        print("→ Review portion sizes, attendance variations, and menu planning.")
    else:
        print("Daily waste is within expected range.")
        print("→ Continue current food preparation strategies.")

# Weekly analysis for a selected month
def analyze_weekly_waste_user_input():
    df = read_food_data()
    year = input("Enter year (e.g., 2024): ")
    month = input("Enter month (1–12): ")

    try:
        year = int(year)
        month = int(month)
        start = pd.to_datetime(f"{year}-{month:02d}-01")  # Start of the selected month
        end = start + pd.offsets.MonthEnd(1)              # End of the selected month
    except:
        print("Invalid year or month.")
        return
    # Filter data for the selected month
```

```
month_df = df[(df['Date'] >= start) & (df['Date'] <= end)]
if month_df.empty:
   print("No data for that month.")
   return
# Assign each day to a week number (1st–7th = Week 1, etc.)
month_df['Week'] = ((month_df['Date'].dt.day - 1) // 7) + 1
weekly = month_df.groupby('Week')['Wasted_kg'].sum()
max_week = weekly.idxmax()
min_week = weekly.idxmin()
max_val = weekly.max()
min_val = weekly.min()

# Plot bar chart showing weekly waste
plt.figure(figsize=(7, 4))
bars = plt.bar(weekly.index, weekly.values, color='lightgreen')
bars[max_week - 1].set_color('red')   # Highlight highest waste week
bars[min_week - 1].set_color('blue')  # Highlight lowest waste week
plt.text(max_week, max_val + 1, f"{max_val:.1f} kg (High)", ha='center', color='red')
plt.text(min_week, min_val + 1, f"{min_val:.1f} kg (Low)", ha='center', color='blue')
plt.title(f"Weekly Food Waste – {start.strftime('%B %Y')}")
plt.xlabel("Week Number")
plt.ylabel("Total Waste (kg)")
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```

```
    # Smart suggestion based on weekly waste spread
    print(f"\n--- Smart Suggestion for {start.strftime('%B %Y')} ---")
    print(f"→ Highest waste: Week {max_week} = {max_val:.1f} kg")
    print(f"→ Lowest waste: Week {min_week} = {min_val:.1f} kg")
    if max_val - min_val > 20:
        print("→ Action: Adjust food quantity based on weekday patterns or events.")
    elif max_val > 35:
        print("→ Action: Investigate spikes and revise overcooking trends.")
    else:
        print("→ Waste level is well balanced across weeks.")

# Monthly analysis for a selected year
def analyze_monthly_waste_user_input():
    df = read_food_data()
    year = input("Enter year (e.g., 2024): ")

    try:
        year = int(year)
    except:
        print("Invalid year.")
        return

    # Group data by month of the selected year
    monthly = df[df['Date'].dt.year == year].groupby(df['Date'].dt.month)['Wasted_kg'].sum()
    if monthly.empty:
        print("No data for this year.")
```

```
    return
max_month = monthly.idxmax()
min_month = monthly.idxmin()
max_val = monthly.max()
min_val = monthly.min()
# Plot bar chart of monthly waste
plt.figure(figsize=(10, 5))
bars = plt.bar(monthly.index, monthly.values, color='skyblue')
bars[max_month - 1].set_color('red')   # Highest waste month
bars[min_month - 1].set_color('blue')  # Lowest waste month

plt.text(max_month, max_val + 2, f"{max_val:.1f} kg (High)", ha='center', color='red')
plt.text(min_month, min_val + 2, f"{min_val:.1f} kg (Low)", ha='center', color='blue')
plt.title(f"Monthly Food Waste for {year}")
plt.xlabel("Month")
plt.ylabel("Total Waste (kg)")
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()

# Smart suggestion for monthly waste performance
print(f"\n--- Smart Suggestion for {year} ---")
print(f"→ Highest waste: Month {max_month} = {max_val:.1f} kg")
print(f"→ Lowest waste: Month {min_month} = {min_val:.1f} kg")
if max_val > 1000:
    print("→ Action: Investigate events, exam breaks, or holidays causing spikes.")
```

```
    elif max_val - min_val > 400:
        print("→ Action: Stabilize cooking plans with more accurate forecasting.")
    else:
        print("→ Waste is controlled. Maintain preparation and tracking habits.")

# Command-line menu for selecting food waste analysis type
def show_food_menu():
    while True:
        print("\nFOOD WASTE ANALYSIS MENU")
        print("----------------------------")
        print("1. Daily Waste (give Date Range)")
        print("2. Weekly Waste (Choose Month)")
        print("3. Monthly Waste (Choose Year)")
        print("4. Exit")
        choice = input("Enter your choice (1–4): ")
        if choice == '1':
            analyze_daily_waste_user_input()
        elif choice == '2':
            analyze_weekly_waste_user_input()
        elif choice == '3':
            analyze_monthly_waste_user_input()
        elif choice == '4':
            print("Exiting Food Analysis Menu.")
            break
        else:
            print("Invalid choice. Please try again.")
```

```
# Program entry point
if __name__ == "__main__":
    show_food_menu()
```

2 . Code For serving_data.py

```
import pandas as pd
import matplotlib.pyplot as plt
# Function to read serving data and convert 'Date' column to datetime format
def read_serving_data():
    file_path = "C:\\Users\\admin\\Downloads\\serving_data_2years.csv"
    df = pd.read_csv(file_path)
    df['Date'] = pd.to_datetime(df['Date'])  # Ensure date column is in datetime format
    return df
# Block-wise waste analysis for a custom date range (daily view)
def blockwise_daily_analysis():
    df = read_serving_data()
    print(f"Available data range: {df['Date'].min().date()} to {df['Date'].max().date()}")
    start = input("Enter start date (YYYY-MM-DD): ")
    end = input("Enter end date (YYYY-MM-DD): ")
    try:
        start_date = pd.to_datetime(start)
        end_date = pd.to_datetime(end)
    except:
        print("Invalid date format.")
```

```
        return
    # Filter records within user-specified date range
    df = df[(df['Date'] >= start_date) & (df['Date'] <= end_date)]
    if df.empty:
        print("No data for this range.")
        return
    # Group by Date and Block to get total waste per block per day
    daily_summary = df.groupby(['Date', 'Block'])['Wasted_kg'].sum().unstack()
    # Plot a line graph with each block's waste trend
    daily_summary.plot(kind='line', figsize=(10, 5))
    plt.title("Daily Block-wise Waste")
    plt.xlabel("Date")
    plt.ylabel("Wasted (kg)")
    plt.grid(True)
    plt.tight_layout()
    plt.show()
    # Find highest and lowest waste for the most recent day in the data
    latest_date = df['Date'].max()
    latest_data = df[df['Date'] == latest_date].groupby('Block')['Wasted_kg'].sum()
    max_block = latest_data.idxmax()
    min_block = latest_data.idxmin()
    print(f"On {latest_date.date()}, Highest Waste: {max_block}, Lowest Waste: {min_block}")
    print("Suggestion: Address over-preparation or portioning in high-waste blocks.\n")
# Weekly block-wise waste comparison for a selected month
def blockwise_weekly_analysis():
    df = read_serving_data()
```

```
year = input("Enter year (e.g., 2024): ")
month = input("Enter month (1–12): ")
try:
    year = int(year)
    month = int(month)
    start = pd.to_datetime(f"{year}-{month:02d}-01")
    end = start + pd.offsets.MonthEnd(1)
except:
    print("Invalid input.")
    return
# Filter data within the selected month
df = df[(df['Date'] >= start) & (df['Date'] <= end)]
if df.empty:
    print("No data for this month.")
    return
# Create 'Week' column by dividing dates into 7-day chunks
df['Week'] = ((df['Date'].dt.day - 1) // 7) + 1
# Group by Week and Block to compute weekly waste
weekly_summary = df.groupby(['Week', 'Block'])['Wasted_kg'].sum().unstack()
# Plot a line graph comparing blocks across weeks
plt.figure(figsize=(10, 5))
for block in weekly_summary.columns:
    plt.plot(weekly_summary.index, weekly_summary[block], marker='o', label=block)
plt.title(f"Weekly Waste Comparison – {start.strftime('%B %Y')}")
plt.xlabel("Week")
plt.ylabel("Waste (kg)")
```

```
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
    # Determine which block had highest and lowest total waste in the month
    block_totals = df.groupby('Block')['Wasted_kg'].sum()
    max_block = block_totals.idxmax()
    min_block = block_totals.idxmin()
    print(f"Weekly Waste Totals – Highest: {max_block}, Lowest: {min_block}")
    print("Suggestion: Monitor menu demand and coordinate distribution across blocks.\n")

# Monthly block-wise comparison for an entire year
def blockwise_monthly_analysis():
    df = read_serving_data()
    year = input("Enter year (e.g., 2024): ")
    try:
        year = int(year)
    except:
        print("Invalid year.")
        return
    # Filter data for the selected year
    df = df[df['Date'].dt.year == year]
    if df.empty:
        print("No data for this year.")
        return
    # Create 'Month' column to group data by month
```

```
    df['Month'] = df['Date'].dt.month
    # Group by Month and Block to compute waste per month for each block
    monthly_summary = df.groupby(['Month', 'Block'])['Wasted_kg'].sum().unstack()
    # Plot line graph comparing monthly waste trends across blocks
    plt.figure(figsize=(10, 5))
    for block in monthly_summary.columns:
        plt.plot(monthly_summary.index, monthly_summary[block], marker='o', label=block)
    plt.title(f"Monthly Waste Comparison – {year}")
    plt.xlabel("Month")
    plt.ylabel("Waste (kg)")
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
    # Find blocks with maximum and minimum waste totals for the year
    block_totals = df.groupby('Block')['Wasted_kg'].sum()
    max_block = block_totals.idxmax()
    min_block = block_totals.idxmin()
    print(f"Monthly Waste Totals – Highest: {max_block}, Lowest: {min_block}")
    print("Suggestion: High monthly waste in some blocks. Reassess food planning.\n")
# Main menu function to interactively navigate serving data analysis
def show_serving_menu():
    while True:
        print("\nSERVING DATA ANALYSIS MENU")
        print("--------------------------")
        print("1. Daily Block-wise Waste")
```

```
        print("2. Weekly Block-wise Waste")
        print("3. Monthly Block-wise Waste")
        print("4. Exit")
        choice = input("Enter your choice (1–4): ")
        if choice == '1':
            blockwise_daily_analysis()
        elif choice == '2':
            blockwise_weekly_analysis()
        elif choice == '3':
            blockwise_monthly_analysis()
        elif choice == '4':
            print("Exiting Serving Data Analysis.")
            break
        else:
            print("Invalid input. Please try again.")

# Entry point for standalone execution
if __name__ == "__main__":
    show_serving_menu()
```

3. code for main_gui.py

```
import tkinter as tk
import sys
import os
# Add parent directory to Python path so it can import from ../data
```

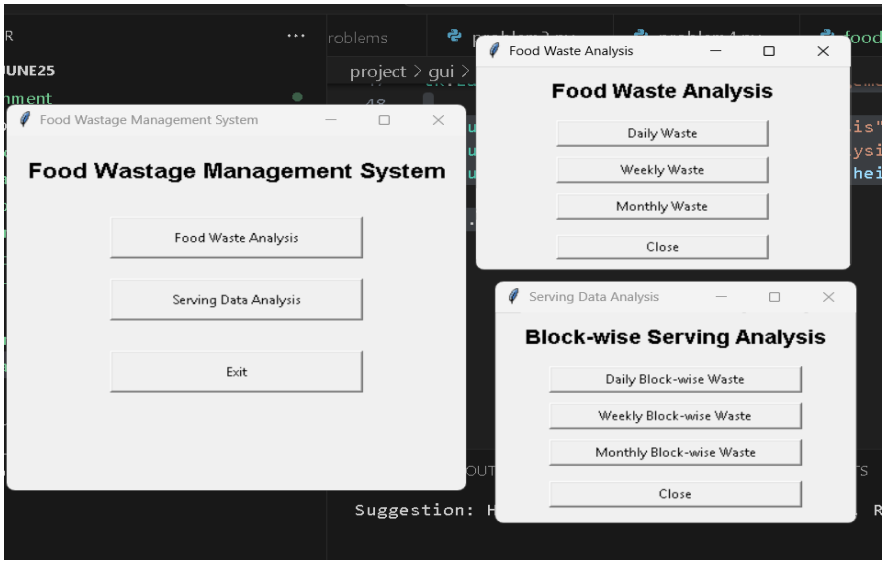
```
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))
# Import analysis functions from food_data.py
from data.food_data import (
    analyze_daily_waste_user_input,
    analyze_weekly_waste_user_input,
    analyze_monthly_waste_user_input
)
# Import blockwise analysis functions from serving_data.py
from data.serving_data import (
    blockwise_daily_analysis,
    blockwise_weekly_analysis,
    blockwise_monthly_analysis
)
# GUI Window for Food Waste Analysis
def show_food_menu_gui():
    window = tk.Toplevel(root)
    window.title("Food Waste Analysis")
    tk.Label(window, text="Food Waste Analysis", font=("Arial", 14, "bold")).pack(pady=10)
    # Buttons to launch daily, weekly, and monthly analysis
    tk.Button(window, text="Daily Waste", width=25,
command=analyze_daily_waste_user_input).pack(pady=5)
    tk.Button(window, text="Weekly Waste", width=25,
command=analyze_weekly_waste_user_input).pack(pady=5)
    tk.Button(window, text="Monthly Waste", width=25,
command=analyze_monthly_waste_user_input).pack(pady=5)
    tk.Button(window, text="Close", width=25, command=window.destroy).pack(pady=10)
# GUI Window for Block-wise Serving Data Analysis
```

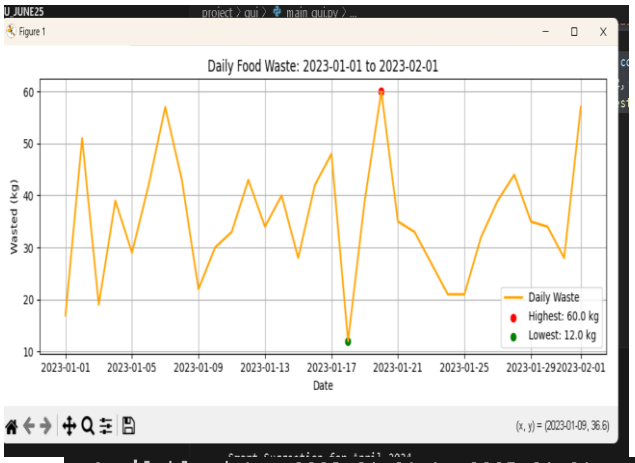
```
def show_serving_menu_gui():
    window = tk.Toplevel(root)
    window.title("Serving Data Analysis")
    tk.Label(window, text="Block-wise Serving Analysis", font=("Arial", 14,
"bold")).pack(pady=10)
    # Buttons to trigger daily, weekly, and monthly block-wise visualizations
    tk.Button(window, text="Daily Block-wise Waste", width=30,
command=blockwise_daily_analysis).pack(pady=5)
    tk.Button(window, text="Weekly Block-wise Waste", width=30,
command=blockwise_weekly_analysis).pack(pady=5)
    tk.Button(window, text="Monthly Block-wise Waste", width=30,
command=blockwise_monthly_analysis).pack(pady=5)
    tk.Button(window, text="Close", width=30, command=window.destroy).pack(pady=10)

# Initialize main GUI window
root = tk.Tk()
root.title("Food Wastage Management System")
root.geometry("400x350")
# Main window title and navigation buttons
tk.Label(root, text="Food Wastage Management System", font=("Arial", 16,
"bold")).pack(pady=20)
tk.Button(root, text="Food Waste Analysis", width=30, height=2,
command=show_food_menu_gui).pack(pady=10)
tk.Button(root, text="Serving Data Analysis", width=30, height=2,
command=show_serving_menu_gui).pack(pady=10)
tk.Button(root, text="Exit", width=30, height=2, command=root.destroy).pack(pady=20)

# Start the GUI event loop
root.mainloop()
```

Output Screenshots

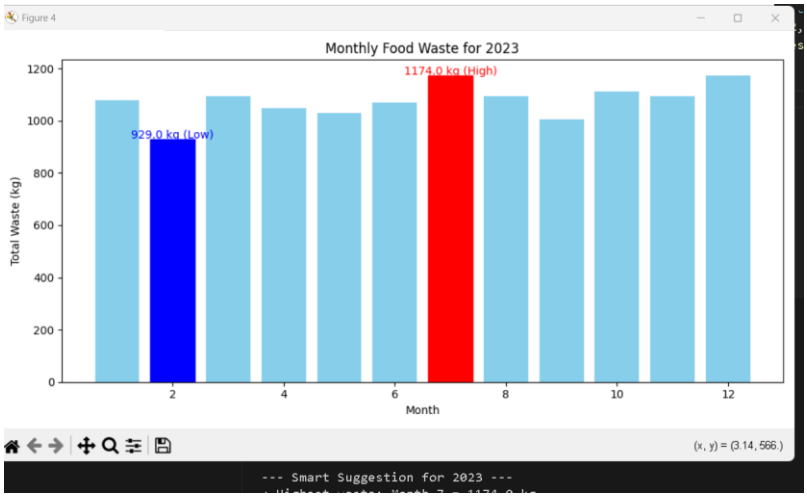




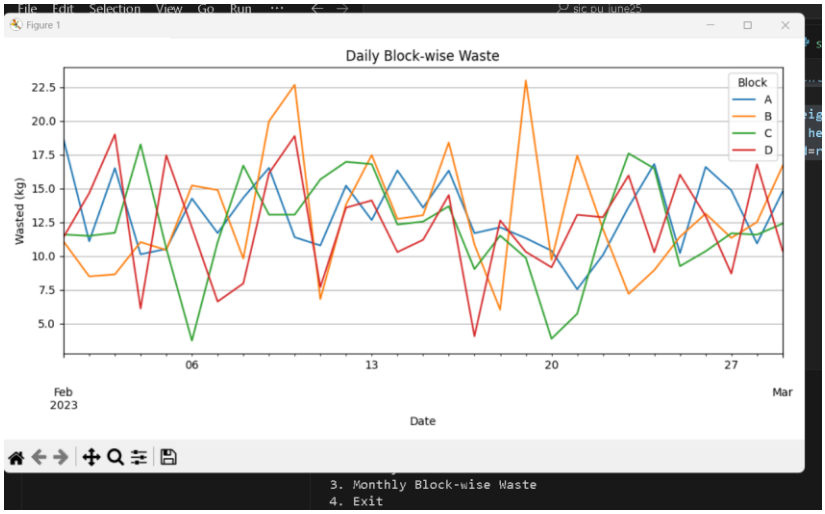

```
Available data: 2023-01-01 to 2025-01-01
Enter start date (YYYY-MM-DD): 2023-01-01
Enter end date (YYYY-MM-DD): 2023-02-01

--- Smart Suggestion ---
High average daily waste detected.
→ Review portion sizes, attendance variations, and menu planning.
```

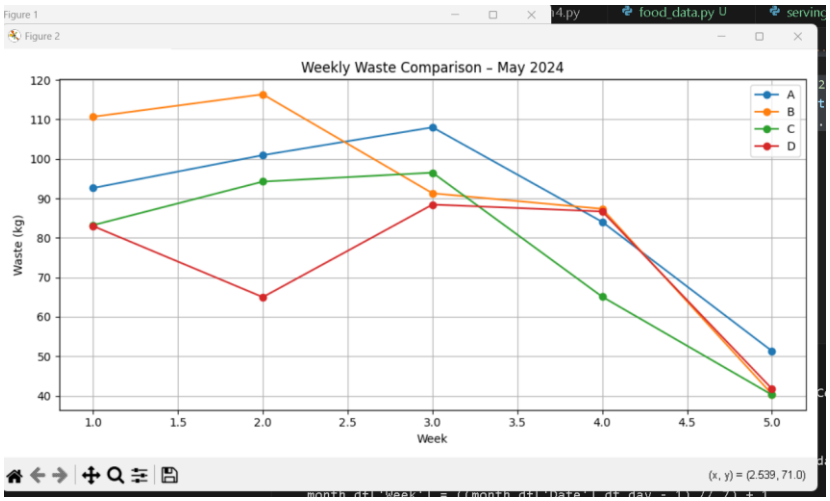

```
--- Smart Suggestion for March 2024 ---
→ Highest waste: Week 4 = 275.0 kg
→ Lowest waste: Week 5 = 139.0 kg
→ Action: Adjust food quantity based on weekday patterns or events.
```



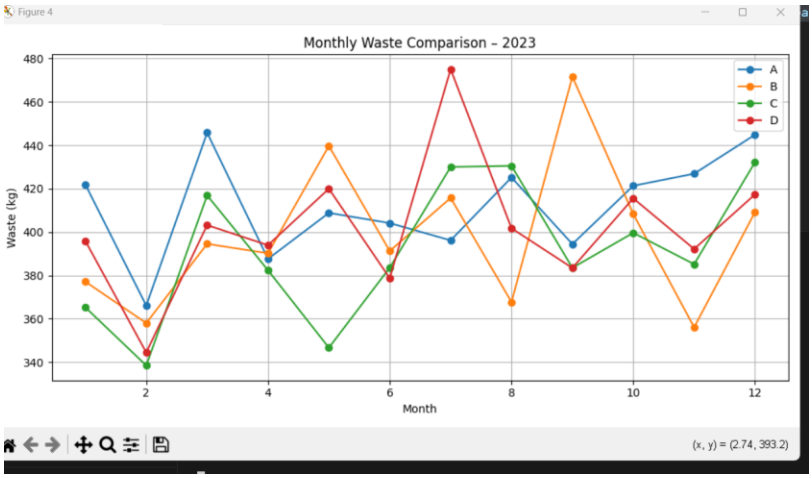

```
--- Smart Suggestion for 2023 ---
→ Highest waste: Month 7 = 1174.0 kg
→ Lowest waste: Month 2 = 929.0 kg
→ Action: Investigate events, exam breaks, or holidays causing spikes.
```




```
Available data range: 2023-01-01 to 2025-01-01
Enter start date (YYYY-MM-DD): 2023-02-01
Enter end date (YYYY-MM-DD): 2023-03-01
On 2023-03-01, Highest Waste: B, Lowest Waste: D
Suggestion: Address over-preparation or portioning in high-waste blocks.
```




```
Enter year (e.g., 2024): 2024
Enter month (1-12): 3
Weekly Waste Totals - Highest: A, Lowest: C
Suggestion: Monitor menu demand and coordinate distribution across blocks.
```




```
Enter year (e.g., 2024): 2023
Monthly Waste Totals - Highest: A, Lowest: C
Suggestion: High monthly waste in some blocks. Reassess food planning.
```

Closure

The Food Wastage Management System successfully integrates real data analytics with visualization and smart recommendations. It supports hostel management in optimizing food preparation and promoting sustainable practices.

Bibliography

- pandas.pydata.org

- matplotlib.org

- python.org (Tkinter)

- Real-world hostel data simulation