KANTI SWEETS DATA ANALYSIS

PROBLEM STATEMENT

Kanti Sweets, a renowned sweets and snacks retail chain, is growing rapidly with multiple branches. However, this expansion brings operational challenges such as managing inventory effectively, minimizing wastage, and understanding sales trends to optimize profits. Although data is collected, it is not utilized fully to drive decisions. This project provides a professional, data-driven approach to analyze historical sales and stock data to empower business decisions and improve resource management using Python and visualization tools.

INDEX

DESCRIPTION

This project is centered around the analysis of sales data collected from Kanti Sweets. The analysis focuses on identifying best-selling products, underperforming items, and overall revenue patterns. It helps store managers and head office teams to:

- Determine product demand and customer preferences.
- Recognize sales performance trends across different days.
- Identify least-sold items that may lead to wastage.

The outcome is a system that can assist in making well-informed decisions, improve profitability, and minimize spoilage of perishable items by providing clear, visual insights into sales trends

DATA WITH RELEVANT INFORMATION & OUTPUTS

Input Data:

- **File: raw_data.csv (located in data/ folder, then cleaned to clean_data.csv)**
- **Attributes: datetime (removed), day of week, location (removed), total bill, and various sweet item quantities (e.g., angbutter, croissant).**

Outputs Generated:

- **Graphs:**
  - **Bar graphs showing total revenue and total quantity sold per day.**
  - **Horizontal bar charts for top 5 selling and least 5 selling sweet items.**
  - **Pie chart showing sweet sales distribution on a user-selected weekday.**
  - **Heatmap visualizing sweet sales across user-selected weekdays.**
- **Textual Analysis:**
  - **Insights highlighting top/least selling items and highest revenue/quantity days.**

Objective:

- **Identify daily sales patterns and product popularity.**
- **Understand revenue and quantity trends across weekdays.**
- **Highlight underperforming products to minimize potential wastage.**

Expected Benefits:

- **Improved inventory management and production planning.**
- **Reduced operational costs due to optimized stock.**
- **Enhanced decision-making for promotions and product offerings.**

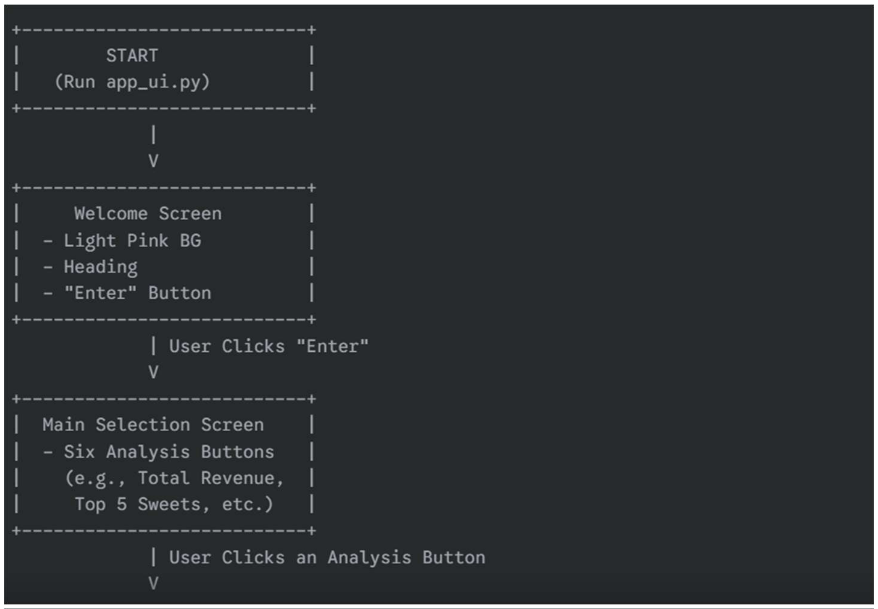
PLAN

To provide a clear and modular solution, the project plan included:

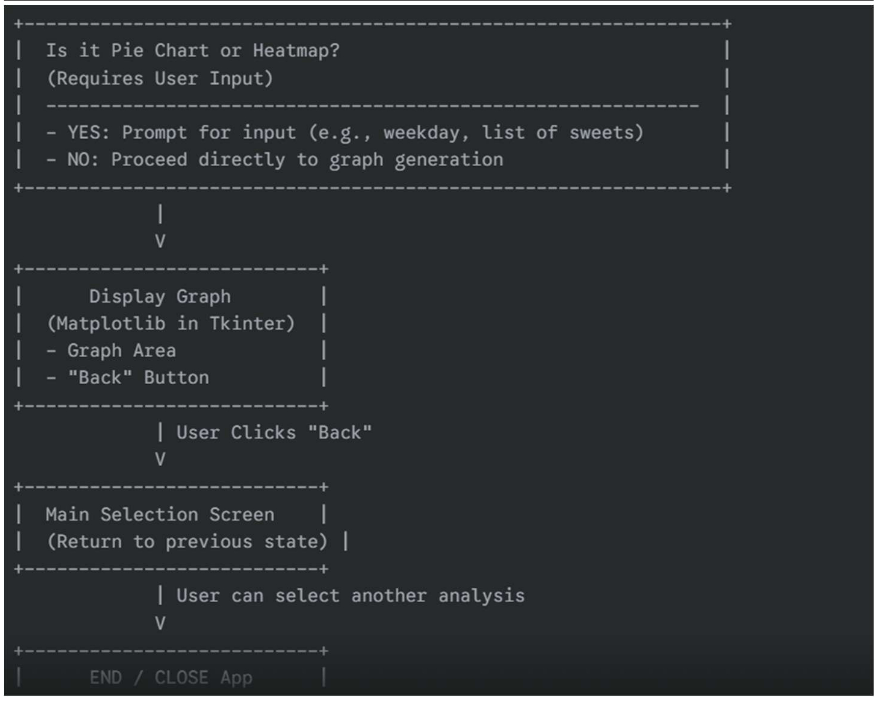
- **Data Collection & Cleaning:** Importing raw_data.csv using pandas, then cleaning it by dropping irrelevant columns (datetime, place), handling missing values (replacing with 0), and converting data types to integers.
- **Data Analysis & Visualization:** Developing individual Python scripts using pandas and matplotlib/seaborn to extract specific insights and generate various charts (bar, horizontal bar, pie, heatmap).
- **User Interface:** Creating an interactive GUI using Tkinter to allow non-technical users to easily select and view different analyses.
- **Modular Design:** Segmenting the logic into separate Python files for data cleaning, each type of analysis, and the main GUI, ensuring clarity and maintainability.

DESIGN

- User Flow Diagram



```
+----------------------------+
|        START               |
|   (Run app_ui.py)          |
+----------------------------+
            |
            V
+----------------------------+
|     Welcome Screen         |
|  - Light Pink BG           |
|  - Heading                 |
|  - "Enter" Button          |
+----------------------------+
            | User Clicks "Enter"
            V
+----------------------------+
|  Main Selection Screen     |
|  - Six Analysis Buttons    |
|    (e.g., Total Revenue,   |
|     Top 5 Sweets, etc.)    |
+----------------------------+
            | User Clicks an Analysis Button
            V
```



```
+-------------------------------------------------------------------+
|  Is it Pie Chart or Heatmap?                                      |
|  (Requires User Input)                                            |
|  --------------------------------------------------------------  |
|  - YES: Prompt for input (e.g., weekday, list of sweets)          |
|  - NO: Proceed directly to graph generation                       |
+-------------------------------------------------------------------+
             |
             V
+----------------------------+
|      Display Graph         |
|  (Matplotlib in Tkinter)   |
|  - Graph Area              |
|  - "Back" Button           |
+----------------------------+
             | User Clicks "Back"
             V
+----------------------------+
|  Main Selection Screen     |
|  (Return to previous state) |
+----------------------------+
             | User can select another analysis
             V
+----------------------------+
|      END / CLOSE App       |
```

IMPLEMENTATION

The project's code is modularized for clarity and maintainability, with distinct files handling specific functionalities:

- **clean_data.py**: This script is responsible for the initial data preparation. It loads the raw sales data, removes irrelevant columns (datetime, place), handles missing values by replacing them with 0, and converts all numerical quantity and bill columns to integers. A key aspect was ensuring robust error handling for file loading and data type conversions.
- **Analysis Modules (in analyzed_data/ folder)**:
  - bar_chart_total_money_vs_day.py: Calculates and visualizes total revenue per weekday.
  - bar_chart_total_sale_quantitiy_vs_day.py: Computes and displays total quantity of sweets sold per weekday.
  - horizontal_bar_chart_top5.py: Identifies and plots the top 5 highest-selling sweet items.
  - bar_chart_least5.py: Highlights and visualizes the 5 least-selling sweet items, aiding in identifying underperforming products.
  - pie_chart_of_selected_weekday_sale_of_sweets.py: Generates a pie chart showing sweet sales distribution for a user-selected weekday.
  - heat_map_selected_weekdays_and_sweets.py: Creates a heatmap to visualize sweet sales across multiple user-selected weekdays, revealing patterns. Each of these modules loads the cleaned data, performs specific aggregations using pandas, and generates a Matplotlib Figure object. Challenges included dynamically identifying sweet columns and ensuring consistent plot sizing for GUI integration.
- **The GUI (app_ui.py)**: This script integrates all analytical features into a single Tkinter-based dashboard. It manages screen transitions (welcome screen to selection, selection to graph view), handles button clicks, and embeds the Matplotlib Figure objects from the analysis modules into the Tkinter window. User interaction for specific graphs (pie chart, heatmap) is managed via simpledialog pop-ups. Robust error handling is included for invalid user inputs or issues during graph generation. All modules share a consistent data flow, primarily communicating via pandas DataFrames and Matplotlib figures.

CODE & EXPLANATION

#cleaning data

```
# Drop 'datetime' and 'place'

df = df.drop(columns=['datetime', 'place'])


# Fill NaN with 0 and convert to integers

df.fillna(0, inplace=True)

df.iloc[:, 2:] = df.iloc[:, 2:].astype(int)


# Save cleaned CSV

df.to_csv("data/clean_data.csv", index=False)
```

Explanation for clean_data.py

Purpose: Prepares raw sales data for analysis.

Explanation: This script loads the raw sales data, removes irrelevant columns like datetime and place, handles all missing values by replacing them with 0, and converts numerical fields (like sweet quantities) to integers. This ensures the data is clean and consistent for all subsequent analyses

.

# bar chart(total money vs day)

```
# Revenue calculation using PRICE_DICT

df['revenue'] = df.apply(lambda row: sum(row[item] * PRICE_DICT.get(item, 0) for item in PRICE_DICT), axis=1)


# Group by weekday

grouped = df.groupby('day of week')['revenue'].sum().reindex(['Mon', 'Tue', 'Wed', 'Thur', 'Fri', 'Sat', 'Sun'])
```

```
        # Plotting
        sns.barplot(x=grouped.index, y=grouped.values, palette='pink')
return plt.gcf(), "Saturday generated the highest revenue, indicating strong weekend sales."
```

Explanation for bar_chart_total_money_vs_day.py

Purpose: Visualizes daily revenue trends.

Explanation: This script calculates the total revenue for each transaction using predefined sweet prices, then aggregates this revenue by day of week. It generates a bar chart showing the total money collected on each day, providing an insight into the highest revenue day.

#bar chart (total sale quantitiy vs day)

```
        # Total quantity = sum of all sweet columns
        df['total_quantity'] = df.iloc[:, 2:].sum(axis=1)

        # Grouped by weekday
grouped = df.groupby('day of week')['total_quantity'].sum().reindex(['Mon', 'Tue', 'Wed',
'Thur', 'Fri', 'Sat', 'Sun'])

        # Plotting
        sns.barplot(x=grouped.index, y=grouped.values, palette='Blues')
        return plt.gcf(), "Sales quantity peaks on Saturday, suggesting more foot traffic during
weekends."
```

Explanation for bar_chart_total_sale_quantitiy_vs_day.py

Purpose: Visualizes total sales volume per day.

Explanation: This script calculates the total quantity of all sweet items sold in each transaction, aggregates these quantities by day of week, and then generates a bar chart. It shows which days have the highest overall sales volume, offering a different perspective from revenue.

#horizontal bar chart (top5)

```
# Sum quantities per sweet

totals = df.iloc[:, 2:].sum().sort_values(ascending=False).head(5)


# Horizontal bar chart

totals.plot(kind='barh', color='green')

plt.title("Top 5 Best Selling Sweets")

return plt.gcf(), f"Top sellers include: {', '.join(totals.index[:3])}."
```

Explanation for horizontal_bar_chart_top5.py

Purpose: Identifies and visualizes the most popular sweet items.

Explanation: This script sums the total units sold for each sweet item across all transactions. It then sorts them to identify the top 5 best-selling sweets and presents this information in a horizontal bar chart, along with an insight about the top item.

#bar chart(least 5)

```
# Bottom 5 selling sweets

totals = df.iloc[:, 2:].sum().sort_values().head(5)


# Bar chart

sns.barplot(x=totals.values, y=totals.index, palette='Oranges')

plt.title("Least Sold (Wasted) Sweets")

return plt.gcf(), f"Least sold items are: {', '.join(totals.index)} — consider reevaluating inventory."
```

Explanation for bar_chart_least5.py

1. • **Purpose:** Identifies and visualizes underperforming sweet items.
2. • **Explanation:** Similar to the top 5 script, this one calculates total units sold per sweet. However, it focuses on the 5 least-selling items, presenting them in a horizontal bar chart. It also provides an insight suggesting potential action for the lowest-selling item.
   ○

#pie chart(selected weekday sale of sweets)

```
df_weekday = df[df['day of week'] == selected_day]
```

```
sweet_sums = df_weekday.iloc[:, 2:].sum()


plt.pie(sweet_sums, labels=sweet_sums.index, ...)
return plt.gcf()  # No insight needed
```

Explanation for pie_chart_of_selected_weekday_sale_of_sweets.py

Purpose: Shows the proportional sales distribution of sweets for a specific weekday.

Explanation: This script takes a user-selected weekday, filters the sales data for that day, and calculates the total quantity sold for each sweet on that particular day. It then generates a pie chart illustrating the percentage contribution of each sweet to the day's total sales.

# heat map(selected weekdays and sweets)

```
df_filtered = df[df['day of week'].isin(selected_days)]
pivot = df_filtered.groupby('day of week')[df.columns[2:]].sum()


sns.heatmap(pivot, cmap='YlGnBu')
return plt.gcf()  # No insight needed
```

Explanation for heat_map_selected_weekdays_and_sweets.py

Purpose: Visualizes sales patterns of sweets across multiple selected weekdays.

Explanation: This script allows the user to select multiple weekdays. It aggregates the sales quantities of all sweets for these chosen days and generates a heatmap. The heatmap visually represents sales volumes, helping to identify trends and performance consistency for different sweets across different days of the week.

#GUI(Graphic user interface)

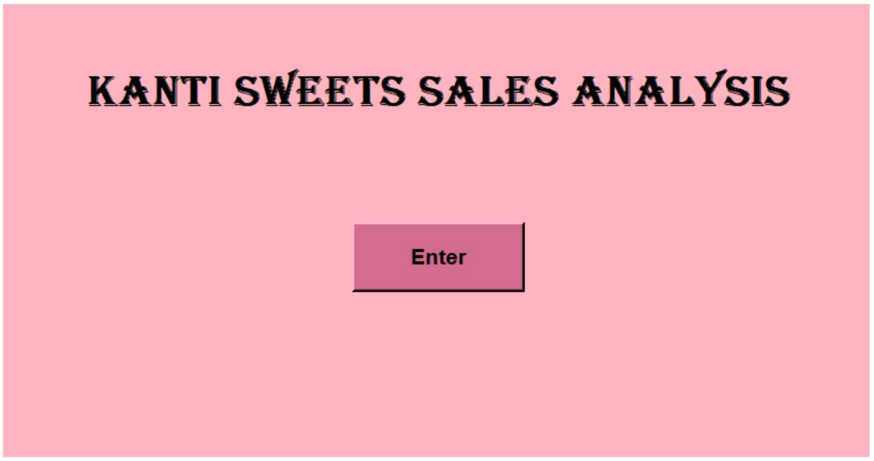
```
# Modified display function to accept insight:
def display_graph(self, fig, insight=""):
    ...
    if insight:
        tk.Label(self.root, text=insight, ...).pack()


# Each graph function returns fig, insight:
def show_total_money(self):
    fig, insight = plot_total_money_vs_day()
    self.display_graph(fig, insight)
```

Purpose: Provides the interactive graphical user interface (GUI) for the entire application.

Explanation: This is the main application script built with Tkinter. It features a welcome screen, a main menu with buttons for each analysis, and dynamically displays the Matplotlib charts generated by the other scripts. It handles user input via dialog boxes for interactive graphs and manages screen transitions, making the complex analysis accessible and user-friendly.

OUTPUT SCREENSHOTS

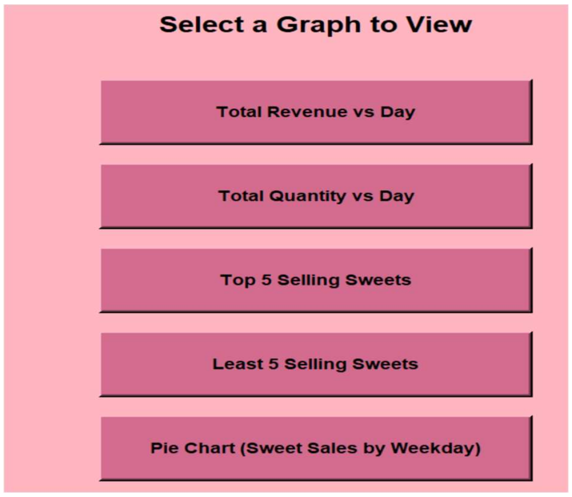
Welcome Screen



Content: The initial application window with a light pink background, the bold "Kanti Sweets Sales Analysis" heading (using Algerian or Arial Black font), and the "Enter" button (styled in dark pink with black text).

Description: This screenshot demonstrates the application's inviting welcome interface, showcasing the chosen color scheme and font styling

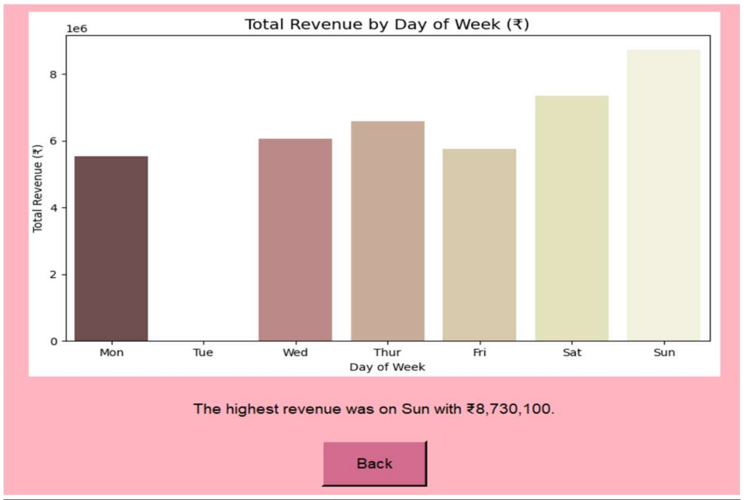
Main Selection Screen



Content: The screen displayed after clicking "Enter" from the welcome screen. It should show the "Select a Graph to View" title and the grid of six large, dark pink buttons, each clearly labeled with an analysis type (e.g., "Total Revenue vs Day", "Top 5 Selling Sweets", etc.).

Description: This image illustrates the intuitive navigation hub, providing users with clear choices for various sales insights.

Bar Chart - Total Revenue vs. Day

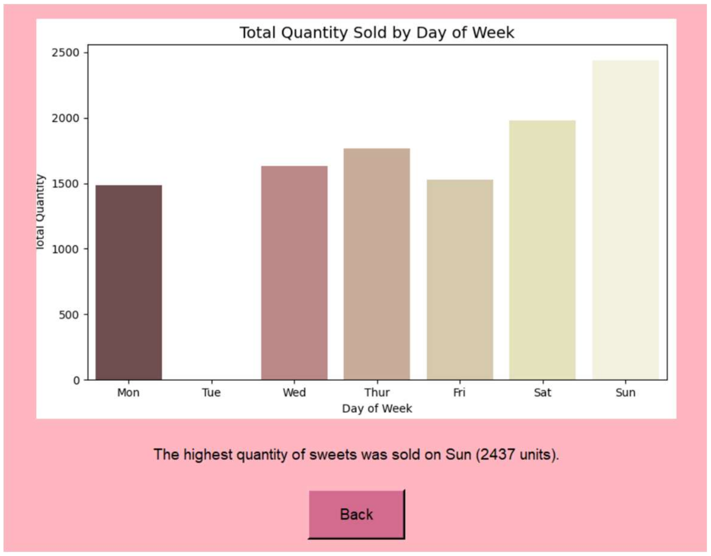


The highest revenue was on Sun with ₹8,730,100.

Content: The Matplotlib bar chart showing total revenue on the Y-axis and days of the week on the X-axis. The chart should be embedded within the Tkinter window, with the "Back to Main Menu" button visible below it. Also, include the insight text (e.g., "The highest revenue was on [Day] with ₹[Value].") below the graph.

Description: This screenshot highlights the visualization of daily revenue trends, allowing for quick identification of high-performing sales days, along with a key textual insight

Bar Chart - Total Quantity vs. Day

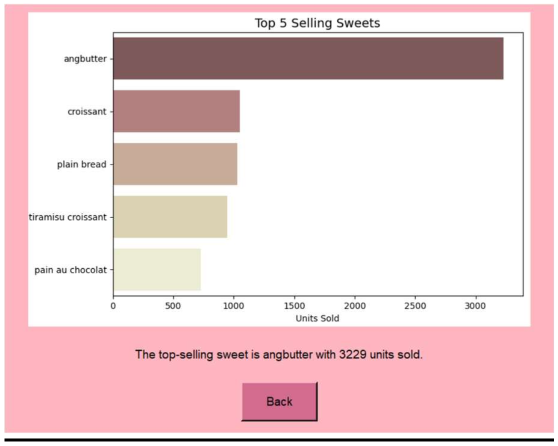


The highest quantity of sweets was sold on Sun (2437 units).

Content: The Matplotlib bar chart showing total sales quantity (all items combined) on the Y-axis and days of the week on the X-axis. The chart should be embedded in the Tkinter window, with the "Back" button visible below it. Also, include the insight text (e.g., "The highest quantity of sweets was sold on [Day] ([Value] units).") below the graph.

Description: This illustrates the total volume of sales per day, which might differ from revenue trends and offers another perspective on daily activity, along with a key textual insight.

Horizontal Bar Chart - Top 5 Selling Sweets

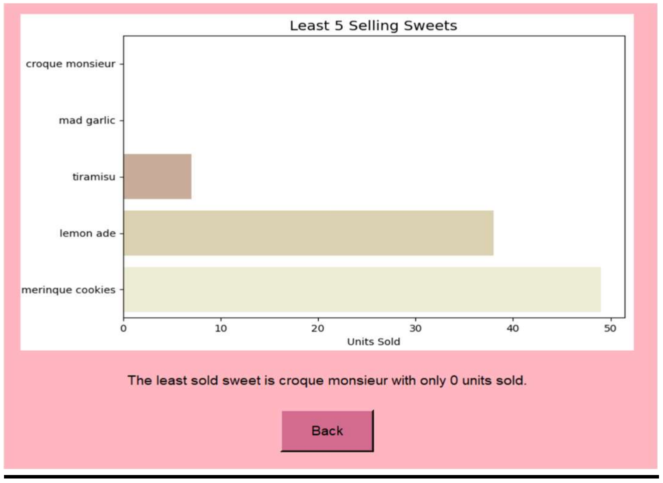


The top-selling sweet is angbutter with 3229 units sold.

Content: The Matplotlib horizontal bar chart displaying the names of the top 5 sweet items on the Y-axis and their total quantities sold on the X-axis. The chart should be embedded in the Tkinter window, with the "Back" button visible below it.

Description: This visualizes the most popular sweet items, providing direct insight for inventory management and marketing focus, along with a key textual insight.

Bar Chart - Least 5 Selling Sweets

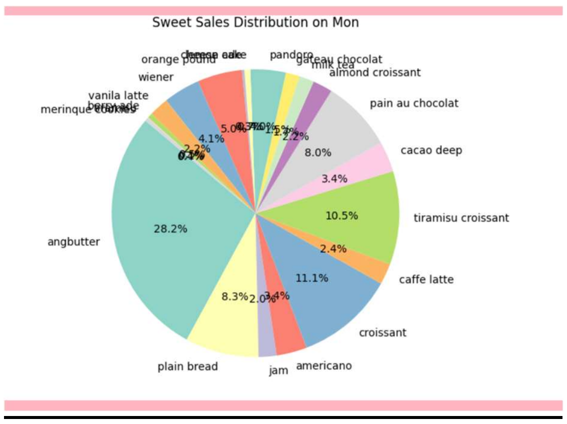


The least sold sweet is croque monsieur with only 0 units sold.

Content: The Matplotlib horizontal bar chart displaying the names of the least 5 selling sweet items on the Y-axis and their total quantities sold on the X-axis. The chart should be embedded in the Tkinter window, with the "Back" button visible below it

Description: This screenshot identifies underperforming products, crucial for making decisions about stock reduction or promotional strategies, along with a key textual insight suggesting action.

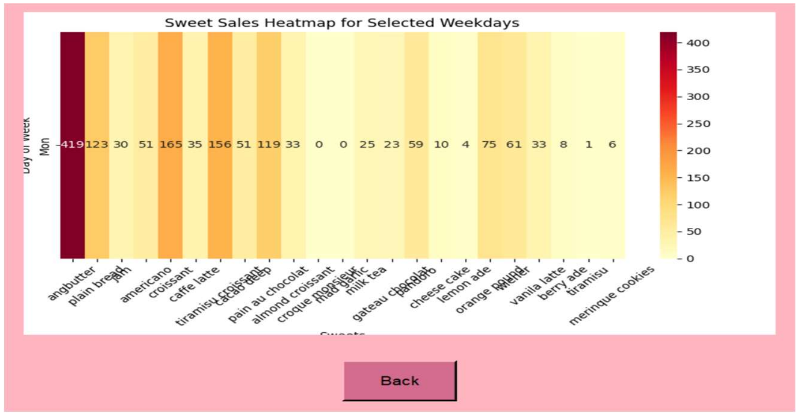
Pie Chart - Selected Weekday Sales



Content: The resulting Matplotlib pie chart for the selected weekday's sweet sales distribution, embedded in the Tkinter main window with the "Back" button visible below i

Description: This demonstrates the interactive capability, showing how users can drill down into sales data for a specific day, providing proportional insights into daily sweet popularity.

Heatmap - Selected Weekdays and Sweets



Content: The resulting Matplotlib heatmap, visualizing sales quantities of various sweets across the selected weekdays, embedded in the Tkinter main window with the "Back" button visible below it.

Description: This highlights the advanced visualization, enabling the discovery of patterns in sweet sales across multiple days, crucial for cross-day comparison and strategic planning.

CLOSURE

The Kanti Sweets Sales Analysis project is a robust, user-friendly application designed to transform raw sales data into actionable business intelligence. It effectively cleans transactional data, visualizes key sales trends through a series of insightful charts, and provides an intuitive graphical interface for easy interaction. From identifying peak sales days and top-performing products to pinpointing underutilized inventory, this tool empowers Kanti Sweets management to make data-driven decisions that enhance operational efficiency, optimize marketing strategies, and ultimately drive growth. Its modular architecture ensures future expandability, making it a valuable asset for ongoing sales analysis.

BIBLIOGRAPHY

Python Programming Language: https://www.python.org/

Pandas Documentation: https://pandas.pydata.org/docs/

Matplotlib Documentation: https://matplotlib.org/stable/contents.html

Seaborn Documentation: https://seaborn.pydata.org/

Tkinter Documentation (Python official docs): https://docs.python.org/3/library/tkinter.html

Matplotlib with Tkinter Integration (FigureCanvasTkAgg): Typically found within Matplotlib's backend documentation or examples.