Cafe Sales Data Analysis

For my portfolio, I worked on a project where I analysed cafe sales data to gain insights into customer behavior, popular products, and revenue patterns. I first focused on cleaning the data using Python, where I handled missing values, removed invalid entries, and ensured data consistency. I then applied visualisation techniques to highlight key trends and insights. Through this process, I gained valuable experience in data cleaning, analysis, and visualising data effectively for decision-making using Python.

Data Import and Initial Inspection

Step 1: Import and Read Data

I began by importing the necessary Python libraries and reading our cafe sales dataset into a Data Frame. This initial step allowed me to inspect the structure and contents of our data.

```
[ ] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
```

```
[ ] from google.colab import drive


    df = pd.read_excel("/content/dirty_cafe_sales.xlsx")
    df.head()
```

	Transaction ID	Item	Quantity	Price Per Unit	Total Spent	Payment Method	Location	Transaction Date
0	TXN_1961373	Coffee	2.0	2	4	Credit Card	Takeaway	2023-09-08 00:00:00
1	TXN_4977031	Cake	4.0	3	12	Cash	In-store	2023-05-16 00:00:00
2	TXN_4271903	Cookie	4.0	1	4	Credit Card	In-store	2023-07-19 00:00:00
3	TXN_7034554	Salad	2.0	5	10	UNKNOWN	UNKNOWN	2023-04-27 00:00:00
4	TXN_3160411	Coffee	2.0	2	4	Digital Wallet	In-store	2023-06-11 00:00:00

Step 2: Create a Copy for Cleaning

To preserve the original data, I created a copy of the dataset. This copy will be used for all subsequent cleaning and analysis operations, ensuring we can always refer back to the original if needed.

```
df_clean = df.copy()
df_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Transaction ID    10000 non-null  object
 1   Item              9375 non-null   object
 2   Quantity          9974 non-null   float64
 3   Price Per Unit    9631 non-null   object
 4   Total Spent       9996 non-null   object
 5   Payment Method    7115 non-null   object
 6   Location          6377 non-null   object
 7   Transaction Date  9699 non-null   object
 8   Unnamed: 8        0 non-null      float64
 9   Unnamed: 9        0 non-null      float64
 10  Unnamed: 10       0 non-null      float64
 11  Unnamed: 11       0 non-null      float64
 12  Unnamed: 12       0 non-null      float64
 13  Unnamed: 13       0 non-null      float64
 14  Unnamed: 14       8 non-null      object
 15  Unnamed: 15       8 non-null      float64
dtypes: float64(8), object(8)
memory usage: 1.2+ MB
```

Data Cleaning Process

After importing the data into Python I began by assigning fixed prices to items using two price dictionaries: one for unique prices and another for non-unique prices. The first dictionary specifies a distinct price for each item, while the second dictionary assigns the same price to items that share a cost, such as Juice and Cake, both priced at 3.

I then merged these two dictionaries into a single prices dictionary, consolidating the item names as keys and their corresponding prices as values.

Next, I focused on converting specific columns to the appropriate numeric data type. Any invalid or non-numeric entries in these columns were replaced with NaN to ensure the data was properly formatted for analysis.

CLEANING THE DATA

```
[ ]  unique_prices = {
         'Cookie': 1.0,
         'Tea': 1.5,
         'Coffee': 2.0,
         'Salad': 5.0}

     non_unique_prices = {
         'Juice': 3.0,
         'Cake': 3.0,
         'Sandwich': 4.0,
         'Smoothie': 4.0}


     prices = unique_prices | non_unique_prices
```

```
[ ]  df_clean[['Total Spent', 'Quantity', 'Price Per Unit']] = df_clean[['Total Spent', 'Quantity', 'Price Per Unit']].apply(pd.to_numeric, errors='coerce')
```

Advanced Data Cleaning Techniques Missing Values

1 Missing Values in "Price Per Unit" (Dictionary)

Next, I focused on filling the missing values in each column. For the "Price Per Unit" column, I wrote a script that checks if the value is missing or marked as NaN. If so, the code looks up the item's price from the previously created prices dictionary, using the item name as the key to fill in the missing price.

2 Missing values in "Price Per Unit" (Calculation)

In the next step, I implemented a similar process, but instead of referencing the dictionary, the price is calculated by dividing the "Total Spent" by the "Quantity" for each row. This ensures that the "Price Per Unit" is populated when possible.

3 Row Removal

Finally, I removed any rows where the "Price Per Unit" remained missing after these operations. This approach guarantees that the "Price Per Unit" column is filled where feasible and eliminates rows with critical missing data from the dataset, ensuring cleaner and more reliable data for analysis.

4 "Total Spent" and "Quantity Columns"

The next two lines calculate any missing values for the total spent and quantity columns by using calculations to ensure that the missing values in both columns are filled based on available data in other columns preserving the consistency of the data set.

```
df_clean['Price Per Unit'] = df_clean.apply(lambda row: prices.get(row['Item']) if pd.isna(row['Price Per Unit']) else row['Price Per Unit'], axis=1)
df_clean['Price Per Unit'] = df_clean.apply(lambda row: row['Total Spent'] / row['Quantity'] if pd.isna(row['Price Per Unit']) else row['Price Per Unit'], axis=1)
df_clean.dropna(subset=['Price Per Unit'], inplace=True)
df_clean.dropna(subset=['Total Spent', 'Quantity'], how='all', inplace=True)
```

```
[ ] df_clean['Total Spent'] = df_clean.apply(lambda row: row['Quantity'] * row['Price Per Unit'] if pd.isna(row['Total Spent']) else row['Total Spent'], axis=1)
```

```
[ ] df_clean['Quantity'] = df_clean.apply(lambda row: row['Total Spent'] * row['Price Per Unit'] if pd.isna(row['Quantity']) else row['Quantity'], axis=1)
```

Advanced Cleaning Techniques

- **Cleaning & Replacing Invalid Data:** The code replaces invalid entries, such as 'UNKNOWN' and 'ERROR', in the Item column with NaN. It then attempts to populate missing **Item** values based on the corresponding Price Per Unit.
- **Date Conversion:** The Transaction Date column is converted to a proper datetime format, and a new column is created to display the month name for each transaction.
- **Row Removal:** Rows with missing Transaction Date values are removed to ensure only complete records are retained.
- **New Column:** A new column is added to show the month corresponding to each transaction.
- **Dataframe Info:** Finally, the code outputs basic information about the DataFrame structure, such as the data types and number of non-null entries.
- This approach effectively cleans the dataset, handles missing or invalid data, and creates new, valuable columns for further analysis.

```
[ ] df_clean['Item'] = df_clean['Item'].replace({'UNKNOWN': np.nan, 'ERROR': np.nan})
    df_clean[df_clean['Item'].isna()]
```

	Transaction ID	Item	Quantity	Price Per Unit	Total Spent	Payment Method	Location	Transaction Date	Unnamed: 8	Unnamed: 9	Unnamed: 10	Unna
6	TXN_4433211	NaN	3.0	3.0	9.0	NaN	Takeaway	2023-10-06	NaN	NaN	NaN	
8	TXN_4717867	NaN	5.0	3.0	15.0	NaN	Takeaway	2023-07-28	NaN	NaN	NaN	
36	TXN_6855453	NaN	4.0	3.0	12.0	NaN	In-store	2023-07-17	NaN	NaN	NaN	
61	TXN_8051289	NaN	1.0	3.0	3.0	NaN	In-store	2023-10-09	NaN	NaN	NaN	
69	TXN_8471743	NaN	5.0	3.0	15.0	Digital Wallet	In-store	2023-04-06	NaN	NaN	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	
9910	TXN_2338617	NaN	2.0	3.0	6.0	Digital Wallet	UNKNOWN	2023-01-12	NaN	NaN	NaN	
9918	TXN_2292088	NaN	1.0	4.0	4.0	Digital Wallet	Takeaway	2023-03-04	NaN	NaN	NaN	
9946	TXN_8807600	NaN	1.0	4.0	4.0	Cash	Takeaway	2023-09-24	NaN	NaN	NaN	
9981	TXN_4583012	NaN	5.0	4.0	20.0	Digital Wallet	NaN	2023-02-27	NaN	NaN	NaN	
9994	TXN_7851634	NaN	4.0	4.0	16.0	NaN	NaN	2023-01-08	NaN	NaN	NaN	

451 rows × 17 columns

```
[ ] df_clean['Item'] = df_clean.apply(lambda row: next((k for k, v in unique_prices.items() if v == row['Price Per Unit']), row['Item']), axis=1)
```

```
[ ] df_clean['Transaction Date'] = pd.to_datetime(df_clean['Transaction Date'], errors='coerce')
```

```
[ ] df_clean.dropna(subset=['Transaction Date'], inplace=True)
```

```
[ ] df_clean['Transaction Month'] = pd.to_datetime(df_clean['Transaction Date']).dt.strftime('%B')
```

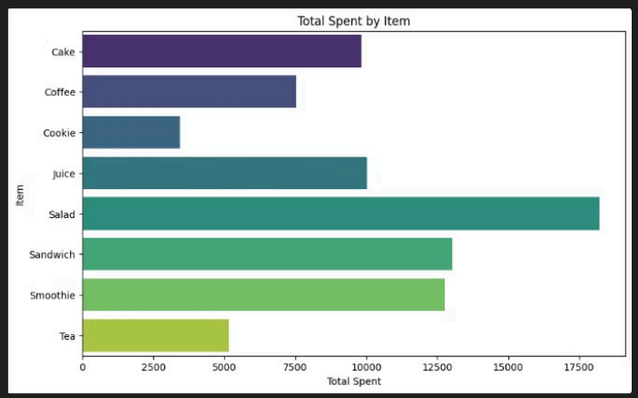
```
df_clean
```

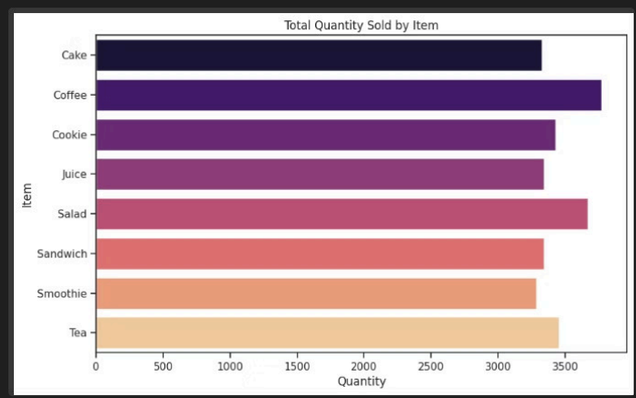
	Transaction ID	Item	Quantity	Price Per Unit	Total Spent	Payment Method	Location	Transaction Date	Unnamed: 8	Unnamed: 9	Unnamed: 10
0	TXN_1961373	Coffee	2.0	2.0	4.0	Credit Card	Takeaway	2023-09-08	NaN	NaN	NaN
1	TXN_4977031	Cake	4.0	3.0	12.0	Cash	In-store	2023-05-16	NaN	NaN	NaN
2	TXN_4271903	Cookie	4.0	1.0	4.0	Credit Card	In-store	2023-07-19	NaN	NaN	NaN
3	TXN_7034554	Salad	2.0	5.0	10.0	UNKNOWN	UNKNOWN	2023-04-27	NaN	NaN	NaN
4	TXN_3160411	Coffee	2.0	2.0	4.0	Digital Wallet	In-store	2023-06-11	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...
9995	TXN_7672686	Coffee	2.0	2.0	4.0	NaN	UNKNOWN	2023-08-30	NaN	NaN	NaN
9996	TXN_9659401	Cookie	3.0	1.0	3.0	Digital Wallet	NaN	2023-06-02	NaN	NaN	NaN
9997	TXN_5255387	Coffee	4.0	2.0	8.0	Digital Wallet	NaN	2023-03-02	NaN	NaN	NaN
9998	TXN_7695629	Cookie	3.0	1.0	3.0	Digital Wallet	NaN	2023-12-02	NaN	NaN	NaN
9999	TXN_6170729	Sandwich	3.0	4.0	12.0	Cash	In-store	2023-11-07	NaN	NaN	NaN

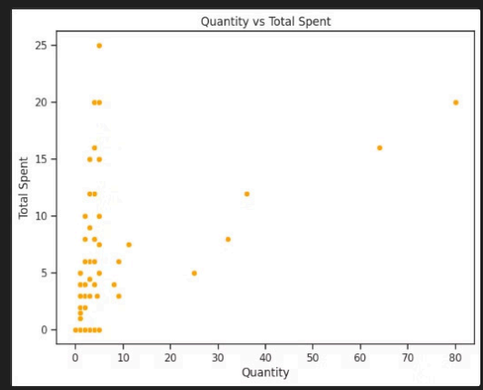
9527 rows × 17 columns

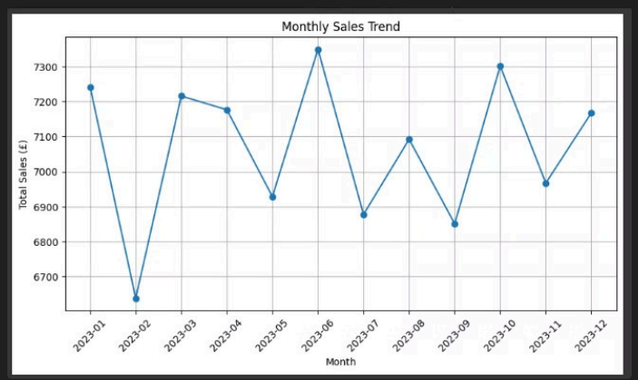
Visuals

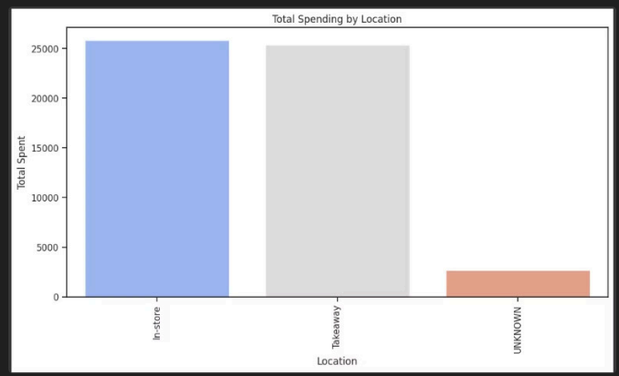
After cleaning the data, I proceeded to create visuals to gain deeper insights and better understand the patterns and trends within the dataset.

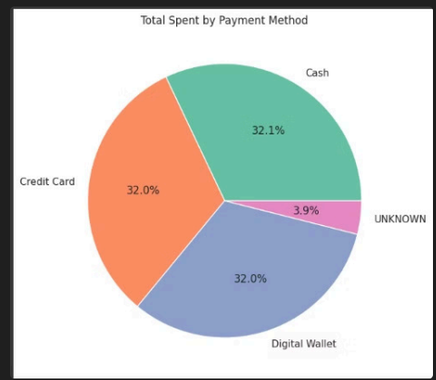






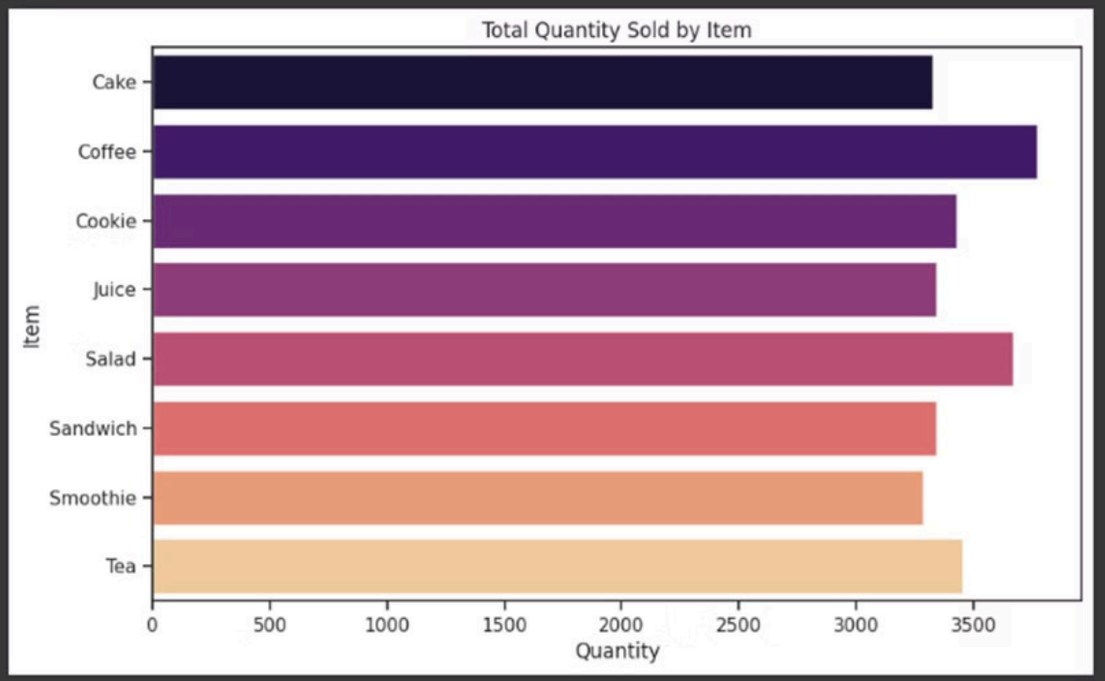






Total Quantity Sold by Item Analysis

```
quantity_sold = df_clean.groupby('Item')['Quantity'].sum().reset_index()

plt.figure(figsize=(10, 6))
sns.barplot(x='Quantity', y='Item', data=quantity_sold, palette='magma')
plt.title('Total Quantity Sold by Item')
plt.xlabel('Quantity')
plt.ylabel('Item')
plt.show()
```

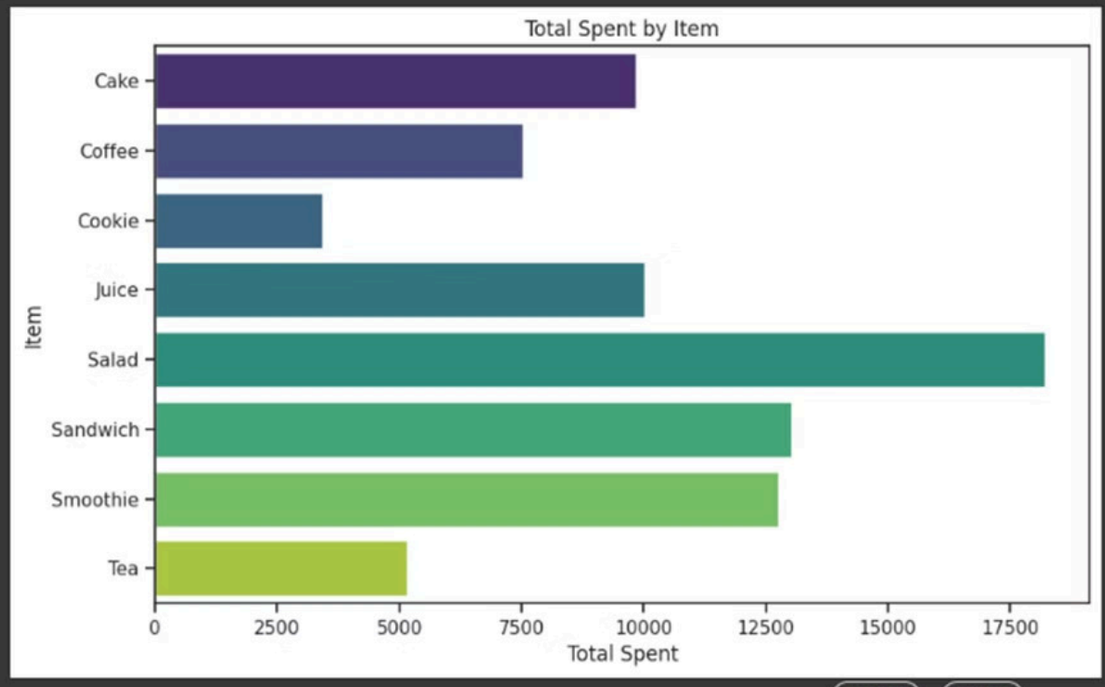


This graph illustrates the total quantity of each item sold, providing insight into the popularity of different products. The sales distribution appears relatively balanced across items, but coffee stands out as the clear favorite, with 3,772 units sold. Salad follows closely behind as the second most popular item, with 3,668 units sold. On the other end of the spectrum, smoothies recorded the lowest sales, with a total of 3,284 units sold. This data helps identify customer preferences and could be useful for inventory management and marketing strategies.

Top Items by Total Spent

VISUALS

```
import matplotlib.pyplot as plt
import seaborn as sns

# Group data by 'Item' and sum 'Total Spent'
item_spending = df_clean.groupby('Item')['Total Spent'].sum().reset_index()

# Plot
plt.figure(figsize=(10, 6))
sns.barplot(x='Total Spent', y='Item', data=item_spending, palette='viridis')
plt.title('Total Spent by Item')
plt.xlabel('Total Spent')
plt.ylabel('Item')
plt.show()
```



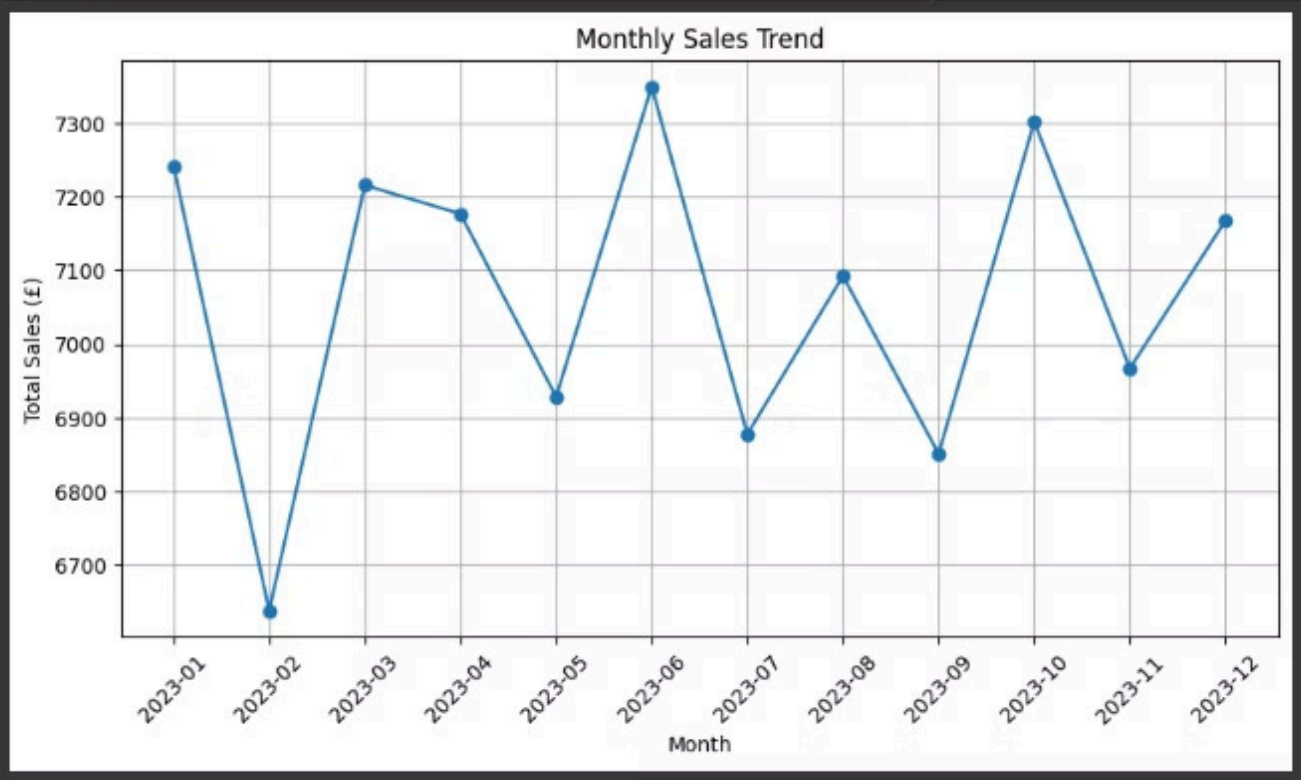
This graph visualises the total revenue generated by each item in the cafe, highlighting which products contribute the most to overall sales. Salads emerge as the top revenue driver, bringing in over $17,500, significantly outperforming other items. Sandwiches take the second spot, generating just over $12,500 in total sales.

At the lower end, cookies bring in the least revenue, with slightly over $3,000 in total sales. This is largely due to their lower price point of $1.00 per unit, compared to salads, which are priced at $5.00 each. This analysis underscores the impact of pricing and volume on revenue, offering insights into product performance and potential pricing or promotional strategies.

Monthly Sales Trends

1. First, I converted the Transaction Date column to datetime format to ensure the data type was correct for analysis.
2. Next, I extracted the month and year while ignoring the day, allowing me to group sales by month rather than specific dates.
3. Then, I grouped the data by month and calculated the total sales for each period to identify revenue trends.
4. To ensure smooth plotting, I converted the Transaction Month column to a string format for proper visualisation.
5. Finally, I formatted the graph for clarity, adding labels, markers, and gridlines to enhance readability and interpretation.

```
df_clean['Transaction Date'] = pd.to_datetime(df_clean['Transaction Date'], errors='coerce')
df_clean['Transaction Month'] = df_clean['Transaction Date'].dt.to_period('M')

monthly_sales = df_clean.groupby('Transaction Month')['Total Spent'].sum().reset_index()


plt.figure(figsize=(10, 5))

monthly_sales['Transaction Month'] = monthly_sales['Transaction Month'].astype(str)

plt.plot(monthly_sales['Transaction Month'], monthly_sales['Total Spent'], marker='o', linestyle='-')

plt.xlabel("Month")
plt.ylabel("Total Sales (£)")
plt.title("Monthly Sales Trend")
plt.xticks(rotation=45)
plt.grid(True)

plt.show()
```



We can see that the best month for sales was June 2023. And the worst was February.

Key Takeaways and Next Steps

Focus on Coffee

With coffee as the top seller, consider expanding the coffee menu and improving the coffee making process.

Menu Expansion

Coffee and salads are the most popular items. To capitalise on this, you could introduce new variations, such as specialty coffee drinks and diverse salad options, to attract more customers and boost sales.

Pricing Changes

Consider pricing changes for some cheaper popular items like cookies.

Seasonal Trends

Ice tea and cold brew sales spike during summer months, indicating a need for seasonal menu adjustments. February is the least popular month. Could introduce a Valentines day specials to boost sales.

By implementing these strategies based on our data analysis, we can enhance customer satisfaction, increase sales, and drive overall business growth. Regular data analysis will help us stay agile and responsive to changing customer preferences and market trends.