Restaurant Order Analysis

Abstract:

This analysis explores restaurant order data to identify trends, optimize menu offerings, and streamline operations. Using transactional data containing order timestamps, item details, and quantities, the study employs filtering, aggregation, and visualization techniques to extract actionable insights. The findings indicate peak order times during lunch hours, with desserts and beverages emerging as customer favorites. Additionally, seasonal preferences and delivery trends are analyzed to understand evolving customer behaviors. These insights provide actionable recommendations for menu refinement, staffing optimization, and promotional strategies, driving improved customer satisfaction and operational efficiency.

Requirements of owner:

1. **What items driven most and least Revenue?**
2. **Which time period has highs and lowest Revenue?**

Datasets:

Order_details.csv:

- Order_details_id
- Order_id
- Order_date
- Order_time
- Item_id

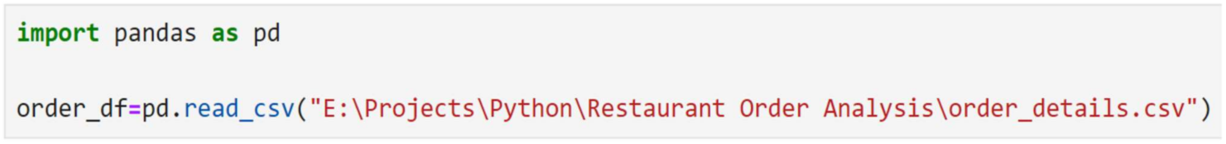
menu_details.csv:

- Menu_item_id
- Item_name
- Category
- Price

Process:

1. **Importing the data:**

1.1 Importing the order_details Dataset:

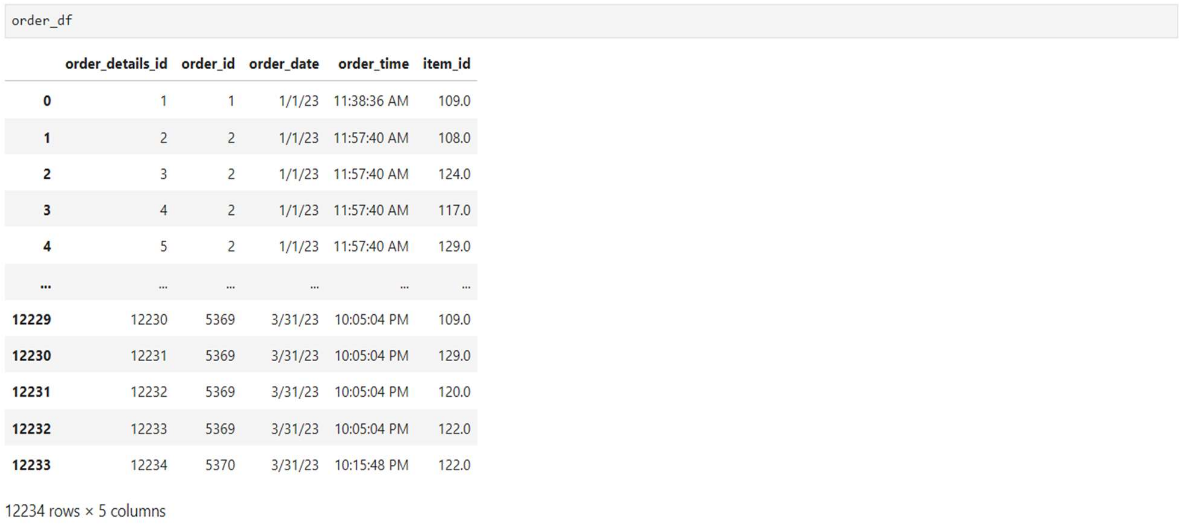


```
import pandas as pd

order_df=pd.read_csv("E:\Projects\Python\Restaurant Order Analysis\order_details.csv")
```

Pandas is a powerful and flexible open-source data manipulation and analysis library for Python. It provides easy-to-use tools for handling structured data (like tables or time-series data) and is widely used for data wrangling, preprocessing, and analysis tasks.

1.2 Information about the DataFrame.



```
order_df
```

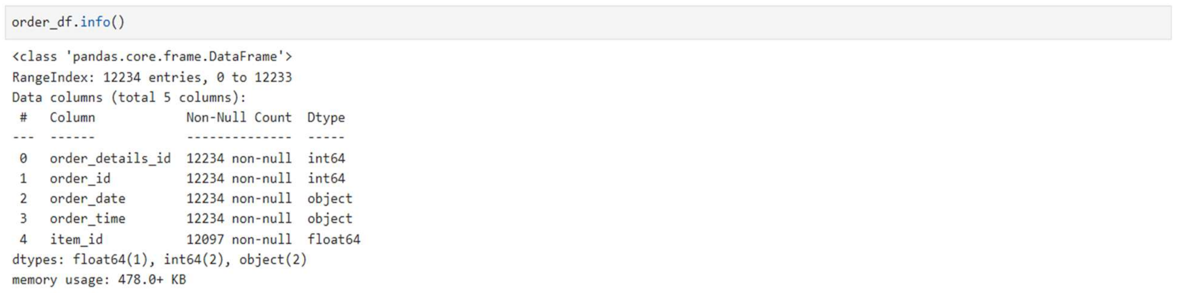
	order_details_id	order_id	order_date	order_time	item_id
0	1	1	1/1/23	11:38:36 AM	109.0
1	2	2	1/1/23	11:57:40 AM	108.0
2	3	2	1/1/23	11:57:40 AM	124.0
3	4	2	1/1/23	11:57:40 AM	117.0
4	5	2	1/1/23	11:57:40 AM	129.0
...	...	...	...	...	...
12229	12230	5369	3/31/23	10:05:04 PM	109.0
12230	12231	5369	3/31/23	10:05:04 PM	129.0
12231	12232	5369	3/31/23	10:05:04 PM	120.0
12232	12233	5369	3/31/23	10:05:04 PM	122.0
12233	12234	5370	3/31/23	10:15:48 PM	122.0

12234 rows × 5 columns

The Dataset consists of First three months of 2023 data form **1/1/2023** to **31/3/2023**

1.3 The information about Dataset are:

The **info()** function in Python's **Pandas** library is used to display a concise summary of a DataFrame
In this dataset there are 12234 Rows for each columns as we have observer there are some **NULL** values in the items_id column so we need to remove the NULL Rows from the DataFrame

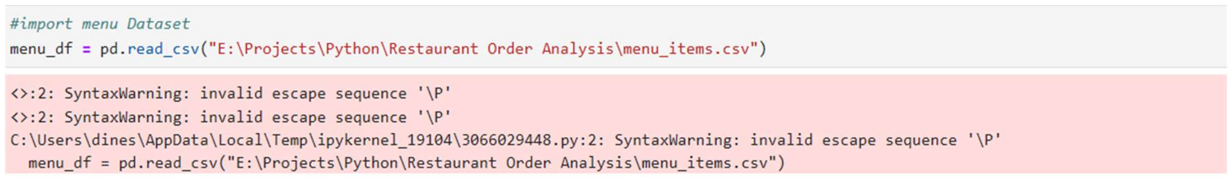


```
order_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12234 entries, 0 to 12233
Data columns (total 5 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   order_details_id  12234 non-null  int64  
 1   order_id          12234 non-null  int64  
 2   order_date        12234 non-null  object 
 3   order_time        12234 non-null  object 
 4   item_id           12097 non-null  float64
dtypes: float64(1), int64(2), object(2)
memory usage: 478.0+ KB
```

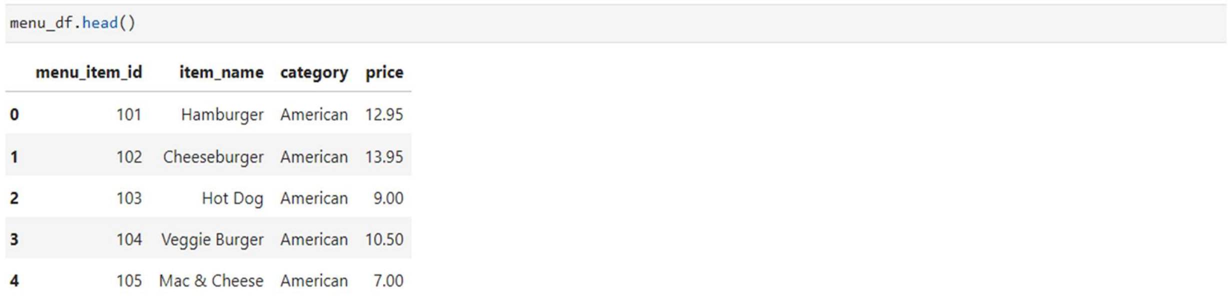
1.4 loading the **menu_details.csv**

Since there are no NULL value we no need Clean up the data.



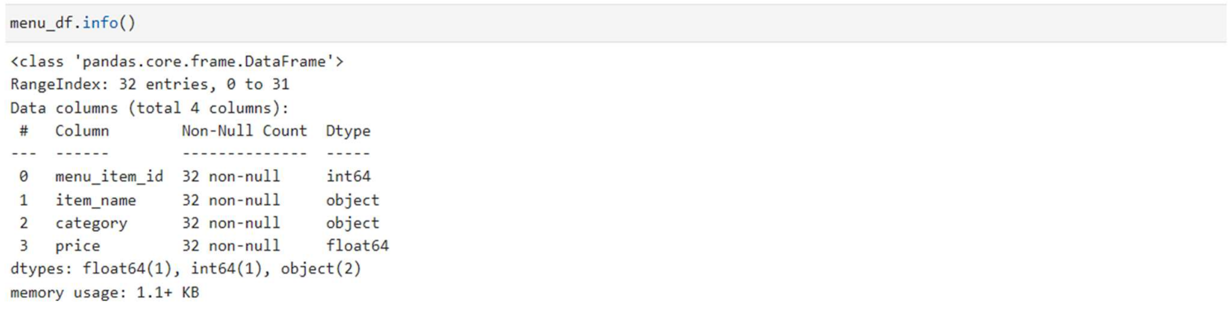
```
#import menu Dataset
menu_df = pd.read_csv("E:\Projects\Python\Restaurant Order Analysis\menu_items.csv")
```

```
<>:2: SyntaxWarning: invalid escape sequence '\P'
<>:2: SyntaxWarning: invalid escape sequence '\P'
C:\Users\dines\AppData\Local\Temp\ipykernel_19104\3066029448.py:2: SyntaxWarning: invalid escape sequence '\P'
  menu_df = pd.read_csv("E:\Projects\Python\Restaurant Order Analysis\menu_items.csv")
```



```
menu_df.head()
```

	menu_item_id	item_name	category	price
0	101	Hamburger	American	12.95
1	102	Cheeseburger	American	13.95
2	103	Hot Dog	American	9.00
3	104	Veggie Burger	American	10.50
4	105	Mac & Cheese	American	7.00



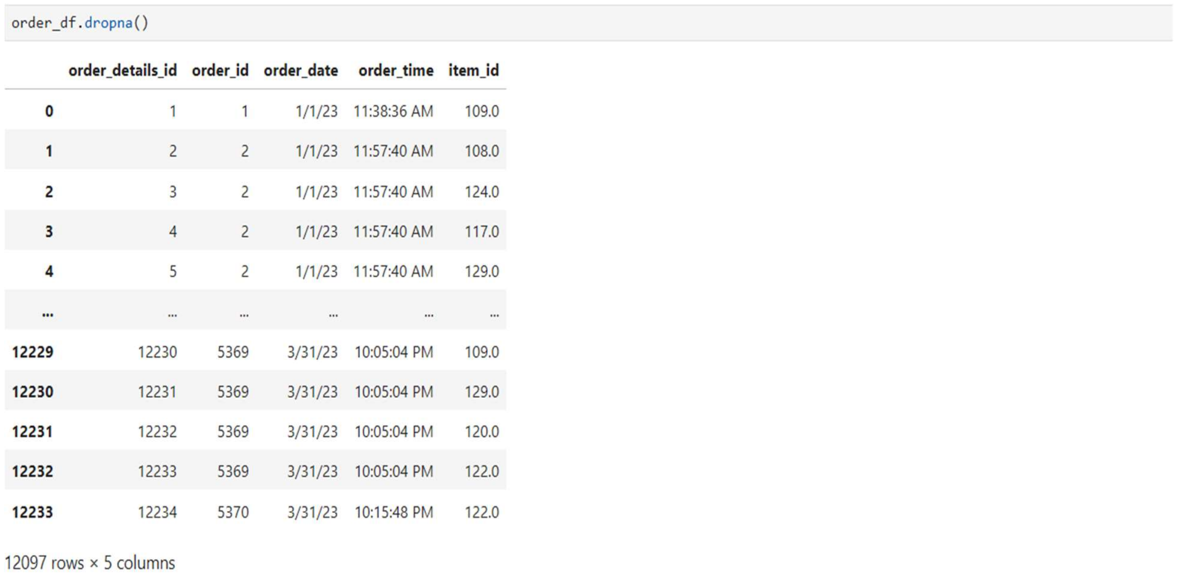
```
menu_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32 entries, 0 to 31
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   menu_item_id  32 non-null     int64  
 1   item_name     32 non-null     object 
 2   category      32 non-null     object 
 3   price         32 non-null     float64
dtypes: float64(1), int64(1), object(2)
memory usage: 1.1+ KB
```

2 Cleaning up Missing values

2.3 Drop the NULL values using **dropna()**:

The dropna() function in Python's **Pandas** library is used to remove missing values (i.e., NaN or None) from a **DataFrame** or **Series**.



```
order_df.dropna()
```

	order_details_id	order_id	order_date	order_time	item_id
0	1	1	1/1/23	11:38:36 AM	109.0
1	2	2	1/1/23	11:57:40 AM	108.0
2	3	2	1/1/23	11:57:40 AM	124.0
3	4	2	1/1/23	11:57:40 AM	117.0
4	5	2	1/1/23	11:57:40 AM	129.0
...	...	...	...	...	...
12229	12230	5369	3/31/23	10:05:04 PM	109.0
12230	12231	5369	3/31/23	10:05:04 PM	129.0
12231	12232	5369	3/31/23	10:05:04 PM	120.0
12232	12233	5369	3/31/23	10:05:04 PM	122.0
12233	12234	5370	3/31/23	10:15:48 PM	122.0

12097 rows × 5 columns

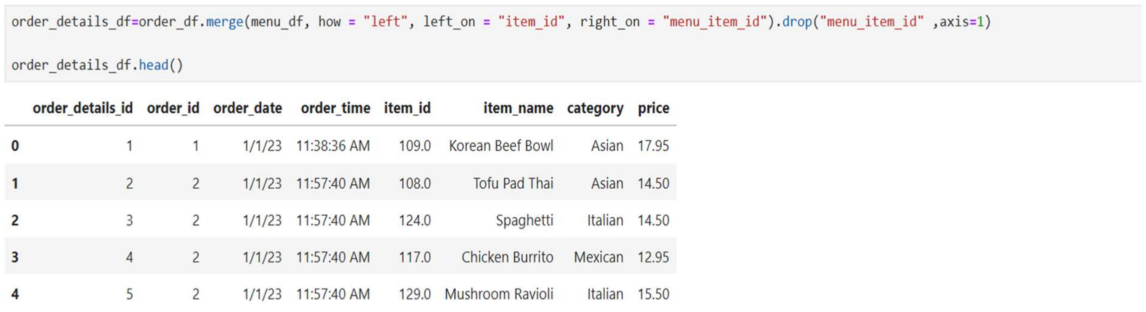
After the dropan() the number or rows in the DataFrame decrease from **12234 to 12097**. The Rows are remove because there are NULL value in the column(items_id).

3 Joining the DataFrame of order_details.csv and menu_deatails.csv:

3.3 Joining the two data set by using merge and join operation.

- **left** & **right**: The two DataFrames to merge.
- **how**: Type of merge:
  - 'inner' (default): Returns rows with matching values in both DataFrames.
  - 'outer': Returns all rows, combining matching ones and filling unmatched rows with NaN.
  - 'left': Returns all rows from the left DataFrame and matches from the right DataFrame.
  - 'right': Returns all rows from the right DataFrame and matches from the left

DataFrame.here we can use **left or right** join to merge the data right_on, left_on will be used to specify the column at which the merge is need to be performed



```
order_details_df=order_df.merge(menu_df, how = "left", left_on = "item_id", right_on = "menu_item_id").drop("menu_item_id" ,axis=1)

order_details_df.head()
```

	order_details_id	order_id	order_date	order_time	item_id	item_name	category	price
0	1	1	1/1/23	11:38:36 AM	109.0	Korean Beef Bowl	Asian	17.95
1	2	2	1/1/23	11:57:40 AM	108.0	Tofu Pad Thai	Asian	14.50
2	3	2	1/1/23	11:57:40 AM	124.0	Spaghetti	Italian	14.50
3	4	2	1/1/23	11:57:40 AM	117.0	Chicken Burrito	Mexican	12.95
4	5	2	1/1/23	11:57:40 AM	129.0	Mushroom Ravioli	Italian	15.50

And then remove the menu_details_id because not to form any duplicate column, the column is dropped by using .**drop().**

4 Adding the Tax and Total Tax Revenue

Adding the sales Tax of **8%** for every item by using the formula

Sales_Tax = price * 0.08

The Total Tax Revenue is find by using the formula :

Total_Tax_Revenue = sales_Tax + price

```
order_items_df["sales_tax"] = (order_items_df.price *.08).round(2)
order_items_df["Total_revenue"] = order_items_df.price + order_items_df.sales_tax

order_items_df.head()
```

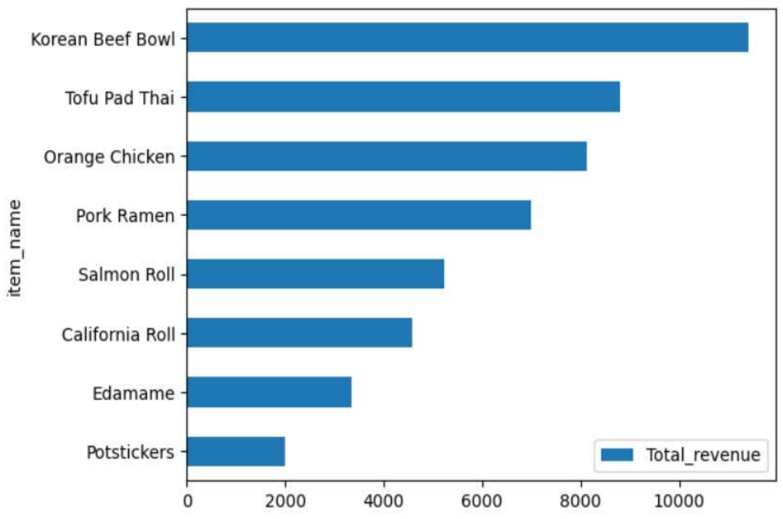
	order_details_id	order_id	order_date	order_time	item_id	item_name	category	price	sales_tax	Total_revenue
0	1	1	1/1/23	11:38:36 AM	109.0	Korean Beef Bowl	Asian	17.95	1.44	19.39
1	2	2	1/1/23	11:57:40 AM	108.0	Tofu Pad Thai	Asian	14.50	1.16	15.66
2	3	2	1/1/23	11:57:40 AM	124.0	Spaghetti	Italian	14.50	1.16	15.66
3	4	2	1/1/23	11:57:40 AM	117.0	Chicken Burrito	Mexican	12.95	1.04	13.99
4	5	2	1/1/23	11:57:40 AM	129.0	Mushroom Ravioli	Italian	15.50	1.24	16.74

5 Visualizing

The visualizing the items in terms of total_renvenue hear we have used

```
(order_items_df
     .query("category == 'Asian'")
    .groupby("item_name")
    .agg({"Total_revenue":"sum"})
    .sort_values("Total_revenue")
     .plot
     .barh()
)
```

```
<Axes: ylabel='item_name'>
```



This bar graph show the most and lowest revenue

- **Query**(): The **query()** function in **Pandas** is a powerful tool used to filter rows of a DataFrame based on a query expression. It allows you to use a simpler and more readable syntax (similar to SQL WHERE clauses) instead of complex conditional filtering with boolean indexing.
- **Groupby():**The groupby() function in Pandas is a powerful tool used to group data based on one or more keys and then perform operations on those groups. It is commonly used for aggregation, transformation, and analysis of data.

- **Agg():**The agg() function in Pandas is a powerful and flexible tool used to perform one or more aggregation operations on grouped data. It allows you to apply multiple functions, either predefined (like sum, mean, etc.) or custom functions, to different columns simultaneously.
- **Sort_values():** The sort_values() function in Pandas is used to sort a DataFrame or Series by one or more columns or values. It is a highly versatile method to arrange data in ascending or descending order based on specified criteria.
- **Plot():** The plot() function in Pandas is used for creating visualizations (such as line plots, bar plots, histograms, etc.) directly from a DataFrame or Series. It is built on top of Matplotlib, and you can use it to easily generate a wide variety of plots.
- **Barh():** The barh() function in Pandas is used to create a horizontal bar plot. This is particularly useful when you want to display data in a horizontal orientation, making it easier to compare categories, especially when the category labels are long or there are many categories.

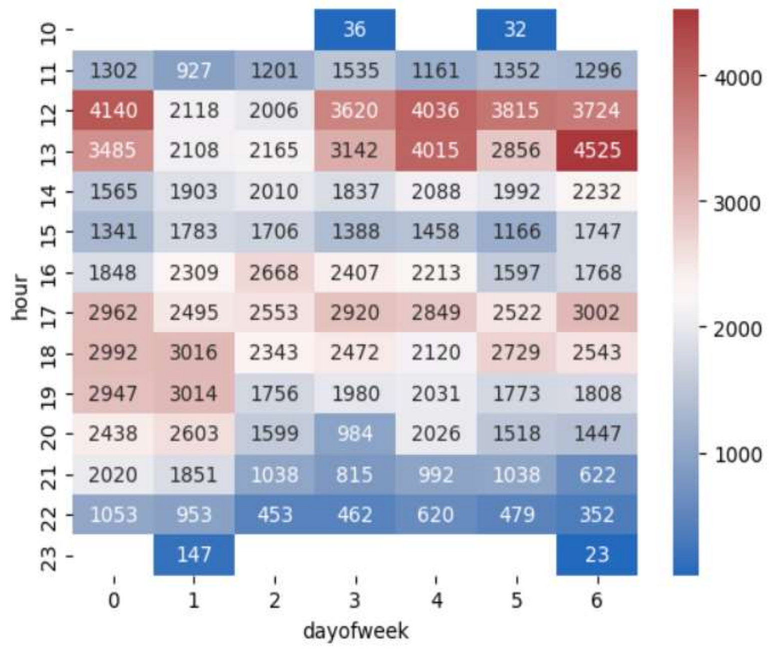
6 Analyzing Busiest Times

We analysis the busiest time by using order_time and order_date

```
import seaborn as sns

sns.heatmap(
    order_items_df.pivot_table(
        index = "hour",
        columns = "dayofweek",
        values = "Total_revenue",
        aggfunc = "sum",
    ).round(),
    annot = True,
    fmt = "g",
    cmap="vlag"
)
```

```
<Axes: xlabel='dayofweek', ylabel='hour'>
```



To perform this report we have import libraries of seaborn with contain a headmap visualization for finding best and worst the best time are represented using red color and the worst time is represented using the blue color.

- **Seaborn** is a Python data visualization library built on top of **Matplotlib**. It provides a high-level interface for creating attractive and informative statistical graphics. It is especially suited for working with **pandas DataFrames** and is widely used for visualizing data distributions, relationships, and patterns.
- the **annot** parameter is commonly used in functions like heatmap() and clustermap() to add annotations (text labels) to the cells of the plot. This is useful when you want to display the actual values or additional information within the plot for better readability or analysis.
- The **fmt** parameter in Seaborn's heatmap() and similar plotting functions (like clustermap()) is used to specify the **format of the annotations** displayed in the cells. It determines how numerical values are presented (e.g., number of decimal places, scientific notation, etc.).
- The **cmap** parameter in **Seaborn** and **Matplotlib** is used to specify the **color map** for visualizations like heatmaps, scatter plots, or other plots where colors represent data values. A color map (or colormap) is a range of colors that can be mapped to specific data values in a plot. Different colormaps can emphasize different aspects of the data and can be particularly useful when visualizing continuous numerical data.

Inference:

The analysis reveals valuable insights into customer preferences and operational efficiency over the past three months. The **Korean Beef Bowl** emerges as the top-performing menu item, driving significant revenue, while **Chicken Tacos** underperformed, generating the least revenue. Peak order times at **12 PM and 1 PM** highlight lunchtime as the busiest period, emphasizing the need for efficient service during these hours. With a total revenue of **$12,097**, these findings provide a clear roadmap to optimize sales by focusing on high-demand items, refining underperforming offerings, and streamlining operations during peak times to enhance customer satisfaction and profitability