Machine Learning

Time Series Analysis (Customer Behavior Timing)

What is the optimal timing for advertisements and promotions to maximize customer purchases, based on historical purchase behavior?

1. Overview

This project leverages time series analysis to examine customer purchase behavior throughout the day. By identifying hourly purchase trends from historical data, we aim to determine the most effective times for deploying targeted advertisements and promotional campaigns. The findings will help optimize marketing strategies, align outreach with peak customer activity, and drive revenue growth through smarter engagement timing.

2. Goal

- Determine the hourly purchase trends based on historical transaction data.
- Identify peak hours when customers are most likely to make purchases.
- Recommend the best timeframes to run ads and promotions for maximum impact.
- Provide actionable insights for marketing and sales optimization strategies.

3. Business Challenge

- Uncertainty about the most effective times to reach customers.
- Low engagement or conversion rates from untargeted promotions.
- Missed revenue opportunities due to poor timing of marketing efforts.
- Limited insight into customer behavioral patterns throughout the day.

4. Methodology

- Utilize historical transaction data to group and count purchases by hour.
- Visualize hourly trends using time series plotting to highlight patterns.
- Identify hours with consistently high customer activity.
- Recommend time slots for targeted promotions and ad placements.
- Integrate insights into broader marketing and sales strategies.

Import necessary libraries

In [14]:

```
import pandas as pd
import os
import glob
```

Combine the sales data from all months into a single consolidated CSV file

In [16]:

```
folder_path = r"C:\Monthly_Sales"

# Retrieve all CSV files from the folder using glob
all_files = glob.glob(os.path.join(folder_path, "*.csv"))

# All CSV files combined as one DataFrame
all_data = pd.concat([pd.read_csv(file) for file in all_files], ignore_index=True)

# Merged DataFrame saved into a new CSV
output_file = os.path.join(folder_path, "all_data.csv")
all_data.to_csv(output_file, index=False)

print("All files integrated into:", output_file)
```

```
All files integrated into: C:\Monthly_Sales\all_data.csv
```

Load the updated DataFrame

In [18]:

```
# Skip Blank Rows if present in the dataset

df = pd.read_csv(r'C:\Monthly_Sales\all_data.csv', skip_blank_lines=True)
df.head()
```

Out[18]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address
0	175667	iPhone	1	700.0	04/24/24 19:12	135 Meadow St, Boston, MA 02215
1	175668	AA Batteries (4-pack)	1	5.84	04/20/24 13:45	592 4th St, San Francisco, CA 94016
2	175669	AA Batteries (4-pack)	1	5.84	04/28/24 09:17	632 Park St, Dallas, TX 75001
3	175670	AA Batteries (4-pack)	2	5.84	04/23/24 14:06	131 Pine St, San Francisco, CA 94016
4	175671	Samsung Odyssey Monitor	1	409.99	04/23/24 12:13	836 Forest St, Boston, MA 02215

In [19]:

```
df.shape
```

Out[19]:

```
(10006800, 6)
```

Data Cleaning Process

Thoroughly clean and standardize the data to eliminate errors, ensure consistency, and build a solid foundation for meaningful insights.

Find and remove rows with NaN values

In [22]:

```
df.isna().sum()
```

Out[22]:

```
Order ID            25984
Product Name        25984
Units Purchased     25986
Unit Price          25986
Order Date          25987
Delivery Address    25988
dtype: int64
```

In [23]:

```
# If Nan value is present in Order ID and Unit Purchased, it will be impossible to 
# Therefore, drop Nan values in Order ID and Units Purchased.

df.dropna(subset=['Order ID', 'Units Purchased'], inplace=True)
```

In [24]:

```
# Check if Nan value is present

df.isna().sum()
```

Out[24]:

```
Order ID            0
Product Name        0
Units Purchased     0
Unit Price          0
Order Date          1
Delivery Address    2
dtype: int64
```

In [25]:

```
# Further check if any NaN values or blank rows are present

blank_rows_na = df[df.isnull().any(axis=1)]
blank_rows_na
```

Out[25]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address
2195228	Charging Cable	1	14.95	05/24/24 07:04	852 Hickory St, San Francisco, CA 94016	NaN
3001506	150766	iPhone	1	7	NaN	NaN

Find and remove rows with duplicate values

In [27]:

```
# Find duplicate values
```

```
df.duplicated()
```

Out[27]:

```
0           False
1           False
2           False
3           False
4           False
            ...  
10006795     True
10006796     True
10006797     True
10006798     True
10006799     True
Length: 9980814, dtype: bool
```

In [28]:

```
# Remove duplicated values

df.drop_duplicates(inplace = True)
```

In [29]:

```
# Check again for duplicated values

df.duplicated()
```

Out[29]:

```
0          False
1          False
2          False
3          False
4          False
           ...  
172530     False
2195228    False
3001506    False
6370083    False
6403571    False
Length: 171546, dtype: bool
```

Verify and fix incorrect data types in the dataset

In [31]:

```
# check for data types

df.dtypes
```

Out[31]:

```
Order ID            object
Product Name        object
Units Purchased     object
Unit Price          object
Order Date          object
Delivery Address    object
dtype: object
```

Fix incorrect data types

In [33]:

```
df['Order Date'] = pd.to_datetime(df['Order Date'], format='%m/%d/%y %H:%M', errors
df['Units Purchased'] = pd. to_numeric(df['Units Purchased'], errors='coerce')

df['Unit Price'] = pd. to_numeric(df['Unit Price'], errors='coerce')
```

In [34]:

```
# Verify the presence of NaN values remaining in the columns as a result of using e

df.isna().sum()
```

Out[34]:

```
Order ID            0
Product Name        0
Units Purchased     1
Unit Price          2
Order Date          3
Delivery Address    2
dtype: int64
```

In [35]:

```
df = df.dropna()
```

Change the data type to optimize memory usage (Optional)

In [37]:

```
df['Order ID'] = pd.to_numeric(df['Order ID'], downcast='integer')
df['Product Name'] = df['Product Name'].astype('category')
df['Units Purchased'] = df['Units Purchased']. astype('int8')
df['Unit Price'] = pd.to_numeric(df['Unit Price'], downcast='float')
df['Delivery Address'] = df['Delivery Address'].astype('category')
```

Expand the dataset with supplementary columns

Add month column

In [40]:

```
df['Month'] = df['Order Date'].dt.month
df
```

Out[40]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month
0	175667	iPhone	1	700.00000	2024-04-24 19:12:00	135 Meadow St, Boston, MA 02215	4
1	175668	AA Batteries (4-pack)	1	5.84000	2024-04-20 13:45:00	592 4th St, San Francisco, CA 94016	4
2	175669	AA Batteries (4-pack)	1	5.84000	2024-04-28 09:17:00	632 Park St, Dallas, TX 75001	4
3	175670	AA Batteries (4-pack)	2	5.84000	2024-04-23 14:06:00	131 Pine St, San Francisco, CA 94016	4
4	175671	Samsung Odyssey Monitor	1	409.98999	2024-04-23 12:13:00	836 Forest St, Boston, MA 02215	4
...	...	...	...	...	...	...	...
172528	248378	Google Phone	1	600.00000	2024-09-02 08:53:00	668 Wilson St, Boston, MA 02215	9
172529	248379	Alienware Monitor	1	400.98999	2024-09-04 22:58:00	466 2nd St, Boston, MA 02215	9
172530	248380	AAA Batteries (4-pack)	1	4.99000	2024-09-04 13:09:00	133 Walnut St, Seattle, WA 98101	9
6370083	252436	Apple Airpods Headphones	1	150.00000	2024-10-14 16:44:00	740 Dogwood St, Boston, \rA 02215	10
6403571	233092	USB-C Charging Cable	1	11.95000	2024-08-28 12:39:00	740 Dogwood St, Boston, \rA 02215	8

171543 rows × 7 columns

In [41]:

```
df['Month Name'] = df['Order Date'].dt.strftime('%B')
df
```

Out[41]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	M N
0	175667	iPhone	1	700.00000	2024-04-24 19:12:00	135 Meadow St, Boston, MA 02215	4	
1	175668	AA Batteries (4-pack)	1	5.84000	2024-04-20 13:45:00	592 4th St, San Francisco, CA 94016	4	
2	175669	AA Batteries (4-pack)	1	5.84000	2024-04-28 09:17:00	632 Park St, Dallas, TX 75001	4	
3	175670	AA Batteries (4-pack)	2	5.84000	2024-04-23 14:06:00	131 Pine St, San Francisco, CA 94016	4	
4	175671	Samsung Odyssey Monitor	1	409.98999	2024-04-23 12:13:00	836 Forest St, Boston, MA 02215	4	
...	...	...	...	...	...	...	...	
172528	248378	Google Phone	1	600.00000	2024-09-02 08:53:00	668 Wilson St, Boston, MA 02215	9	Septer
172529	248379	Alienware Monitor	1	400.98999	2024-09-04 22:58:00	466 2nd St, Boston, MA 02215	9	Septer
172530	248380	AAA Batteries (4-pack)	1	4.99000	2024-09-04 13:09:00	133 Walnut St, Seattle, WA 98101	9	Septer
6370083	252436	Apple Airpods Headphones	1	150.00000	2024-10-14 16:44:00	740 Dogwood St, Boston, \rA 02215	10	Oct

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	M N
6403571	233092	USB-C Charging Cable	1	11.95000	2024-08-28 12:39:00	740 Dogwood St, Boston, \rA 02215	8	Au

171543 rows × 8 columns

Add week day column

In [43]:

```
df['Day of Week'] = df['Order Date'].dt.strftime('%a')
df
```

Out[43]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	Mo N
0	175667	iPhone	1	700.00000	2024-04-24 19:12:00	135 Meadow St, Boston, MA 02215	4	
1	175668	AA Batteries (4-pack)	1	5.84000	2024-04-20 13:45:00	592 4th St, San Francisco, CA 94016	4	
2	175669	AA Batteries (4-pack)	1	5.84000	2024-04-28 09:17:00	632 Park St, Dallas, TX 75001	4	
3	175670	AA Batteries (4-pack)	2	5.84000	2024-04-23 14:06:00	131 Pine St, San Francisco, CA 94016	4	
4	175671	Samsung Odyssey Monitor	1	409.98999	2024-04-23 12:13:00	836 Forest St, Boston, MA 02215	4	
...	...	...	...	...	...	...	...	
172528	248378	Google Phone	1	600.00000	2024-09-02 08:53:00	668 Wilson St, Boston, MA 02215	9	Septer
172529	248379	Alienware Monitor	1	400.98999	2024-09-04 22:58:00	466 2nd St, Boston, MA 02215	9	Septer
172530	248380	AAA Batteries (4-pack)	1	4.99000	2024-09-04 13:09:00	133 Walnut St, Seattle, WA 98101	9	Septer
6370083	252436	Apple Airpods Headphones	1	150.00000	2024-10-14 16:44:00	740 Dogwood St, Boston, \rA 02215	10	Oct

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	M N
6403571	233092	USB-C Charging Cable	1	11.95000	2024-08-28 12:39:00	740 Dogwood St, Boston, \rA 02215	8	Au

171543 rows × 9 columns

Add hour column

In [45]:

```
df['Hour'] = df['Order Date'].dt.hour
df
```

Out[45]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	M N
0	175667	iPhone	1	700.00000	2024-04-24 19:12:00	135 Meadow St, Boston, MA 02215	4	
1	175668	AA Batteries (4-pack)	1	5.84000	2024-04-20 13:45:00	592 4th St, San Francisco, CA 94016	4	
2	175669	AA Batteries (4-pack)	1	5.84000	2024-04-28 09:17:00	632 Park St, Dallas, TX 75001	4	
3	175670	AA Batteries (4-pack)	2	5.84000	2024-04-23 14:06:00	131 Pine St, San Francisco, CA 94016	4	
4	175671	Samsung Odyssey Monitor	1	409.98999	2024-04-23 12:13:00	836 Forest St, Boston, MA 02215	4	
...	...	...	...	...	...	...	...	
172528	248378	Google Phone	1	600.00000	2024-09-02 08:53:00	668 Wilson St, Boston, MA 02215	9	Septer
172529	248379	Alienware Monitor	1	400.98999	2024-09-04 22:58:00	466 2nd St, Boston, MA 02215	9	Septer
172530	248380	AAA Batteries (4-pack)	1	4.99000	2024-09-04 13:09:00	133 Walnut St, Seattle, WA 98101	9	Septer
6370083	252436	Apple Airpods Headphones	1	150.00000	2024-10-14 16:44:00	740 Dogwood St, Boston, \rA 02215	10	Oct

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	M N
6403571	233092	USB-C Charging Cable	1	11.95000	2024-08-28 12:39:00	740 Dogwood St, Boston, \rA 02215	8	Au

171543 rows × 10 columns

Add city column

In [47]:

```
def city(address):
    return address.split(",")[1].strip(" ")

def state_abbrev(address):
    return address.split(",")[2].split(" ")[1]

df['City'] = df['Delivery Address'].apply(lambda x: f"{city(x)}  ({state_abbrev(x)}
df.head()
```

Out[47]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	Month Name	Day of Week
0	175667	iPhone	1	700.00000	2024-04-24 19:12:00	135 Meadow St, Boston, MA 02215	4	April	Wed
1	175668	AA Batteries (4-pack)	1	5.84000	2024-04-20 13:45:00	592 4th St, San Francisco, CA 94016	4	April	Sat
2	175669	AA Batteries (4-pack)	1	5.84000	2024-04-28 09:17:00	632 Park St, Dallas, TX 75001	4	April	Sun
3	175670	AA Batteries (4-pack)	2	5.84000	2024-04-23 14:06:00	131 Pine St, San Francisco, CA 94016	4	April	Tue
4	175671	Samsung Odyssey Monitor	1	409.98999	2024-04-23 12:13:00	836 Forest St, Boston, MA 02215	4	April	Tue

Organize Data by Order Date Chronologically and Reindex

In [49]:

```
df = df.sort_values(by = 'Order Date')
df
```

Out[49]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	Month Name
78282	160155	Alienware Monitor	1	400.989990	2024-01-01 05:04:00	765 Ridge St, Portland, OR 97035	1	January
68761	151041	AAA Batteries (4-pack)	1	4.990000	2024-01-01 05:04:00	964 Lakeview St, Atlanta, GA 30301	1	January
64303	146765	AAA Batteries (4-pack)	1	4.990000	2024-01-01 05:20:00	546 10th St, San Francisco, CA 94016	1	January
63092	145617	Amana Washing Machine	1	600.000000	2024-01-01 05:24:00	961 Meadow St, Portland, OR 97035	1	January
74502	156535	iPhone	1	700.000000	2024-01-01 05:45:00	451 Elm St, Los Angeles, CA 90001	1	January
...	...	...	...	...	...	...	...	...
44457	297748	iPhone	1	700.000000	2025-01-01 02:37:00	258 Forest St, Los Angeles, CA 90001	1	January
30663	284606	Bose SoundSport Headphones	1	99.989998	2025-01-01 02:50:00	211 Johnson St, Boston, MA 02215	1	January
49246	302330	AA Batteries (4-pack)	1	5.840000	2025-01-01 03:03:00	665 6th St, San Francisco, CA 94016	1	January
30770	284711	AA Batteries (4-pack)	1	5.840000	2025-01-01 03:19:00	250 8th St, San Francisco, CA 94016	1	January

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	Month Name
50619	303626	USB-C Charging Cable	3	11.950000	2025-01-01 04:43:00	651 Lakeview St, Dallas, TX 75001	1	January

171543 rows × 11 columns

In [50]:
```
df = df.reset_index(drop=True)
df
```

Out[50]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	Mont Nam
0	160155	Alienware Monitor	1	400.989990	2024-01-01 05:04:00	765 Ridge St, Portland, OR 97035	1	Januar
1	151041	AAA Batteries (4-pack)	1	4.990000	2024-01-01 05:04:00	964 Lakeview St, Atlanta, GA 30301	1	Januar
2	146765	AAA Batteries (4-pack)	1	4.990000	2024-01-01 05:20:00	546 10th St, San Francisco, CA 94016	1	Januar
3	145617	Amana Washing Machine	1	600.000000	2024-01-01 05:24:00	961 Meadow St, Portland, OR 97035	1	Januar
4	156535	iPhone	1	700.000000	2024-01-01 05:45:00	451 Elm St, Los Angeles, CA 90001	1	Januar
...	...	...	...	...	...	...	...	.
171538	297748	iPhone	1	700.000000	2025-01-01 02:37:00	258 Forest St, Los Angeles, CA 90001	1	Januar
171539	284606	Bose SoundSport Headphones	1	99.989998	2025-01-01 02:50:00	211 Johnson St, Boston, MA 02215	1	Januar
171540	302330	AA Batteries (4-pack)	1	5.840000	2025-01-01 03:03:00	665 6th St, San Francisco, CA 94016	1	Januar
171541	284711	AA Batteries (4-pack)	1	5.840000	2025-01-01 03:19:00	250 8th St, San Francisco, CA 94016	1	Januar

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	Mont Nam
171542	303626	USB-C Charging Cable	3	11.950000	2025-01-01 04:43:00	651 Lakeview St, Dallas, TX 75001	1	Januar

171543 rows × 11 columns

Add Total Sales column

```
In [52]: df['Total Sales'] = df['Units Purchased'] * df['Unit Price']
         df.head()
```

Out[52]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	Month Name	Day of Week
0	160155	Alienware Monitor	1	400.98999	2024-01-01 05:04:00	765 Ridge St, Portland, OR 97035	1	January	Mon
1	151041	AAA Batteries (4-pack)	1	4.99000	2024-01-01 05:04:00	964 Lakeview St, Atlanta, GA 30301	1	January	Mon
2	146765	AAA Batteries (4-pack)	1	4.99000	2024-01-01 05:20:00	546 10th St, San Francisco, CA 94016	1	January	Mon
3	145617	Amana Washing Machine	1	600.00000	2024-01-01 05:24:00	961 Meadow St, Portland, OR 97035	1	January	Mon
4	156535	iPhone	1	700.00000	2024-01-01 05:45:00	451 Elm St, Los Angeles, CA 90001	1	January	Mon

Format Unit Price and Total Sales to 2 decimal places

```
In [54]: df['Unit Price'] = df['Unit Price'].apply(lambda x: "%.2f" % x)
```

In [55]:

```python
df['Total Sales'] = df['Total Sales'].apply(lambda x: "%.2f" % x)
df.head()
```

Out[55]:

	Order ID	Product Name	Units Purchased	Unit Price	Order Date	Delivery Address	Month	Month Name	Day of Week	Hc
0	160155	Alienware Monitor	1	400.99	2024-01-01 05:04:00	765 Ridge St, Portland, OR 97035	1	January	Mon	
1	151041	AAA Batteries (4-pack)	1	4.99	2024-01-01 05:04:00	964 Lakeview St, Atlanta, GA 30301	1	January	Mon	
2	146765	AAA Batteries (4-pack)	1	4.99	2024-01-01 05:20:00	546 10th St, San Francisco, CA 94016	1	January	Mon	
3	145617	Amana Washing Machine	1	600.00	2024-01-01 05:24:00	961 Meadow St, Portland, OR 97035	1	January	Mon	
4	156535	iPhone	1	700.00	2024-01-01 05:45:00	451 Elm St, Los Angeles, CA 90001	1	January	Mon	

Format Unit Price and Total Sales to numeric

In [57]:

```python
df['Unit Price'] = pd.to_numeric(df['Unit Price'])
df['Total Sales'] = pd.to_numeric(df['Total Sales'])
```

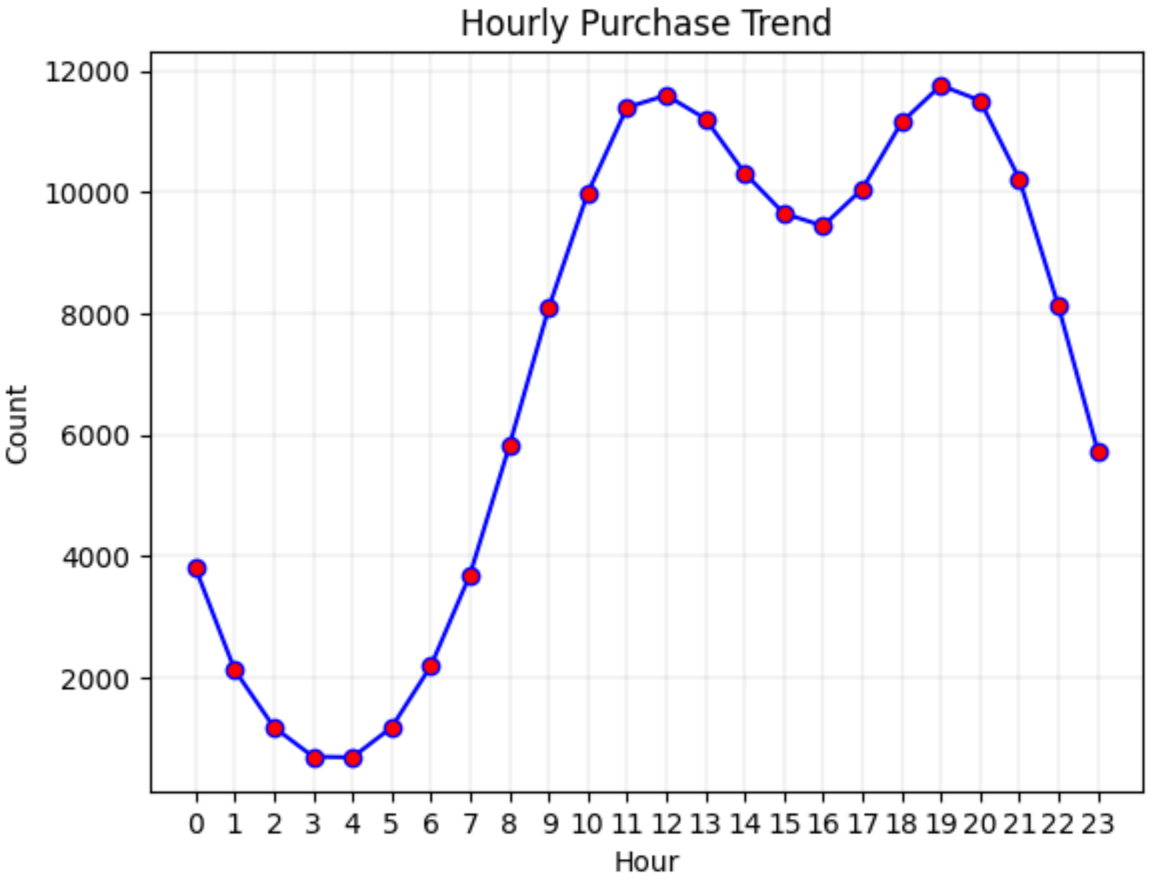
Plot Hourly Purchase Trend

In [67]:

```python
import matplotlib.pyplot as plt

# Group by 'Hour' and counting the number of occurrences
hourly_counts = df.groupby('Hour').size()

# Print out the values
print("Hourly Purchase Counts:")
for hour, count in hourly_counts.items():
    print(f"{hour:02d}:00 - {count} purchases")
```

```
# Plot
plt.plot(hourly_counts, marker='o', markerfacecolor='red', linestyle='-', color='bl
plt.title('Hourly Purchase Trend')
plt.xlabel('Hour')
plt.ylabel('Count')
plt.xticks(range(0, 24))
plt.grid(linewidth=0.2)

plt.show()
```

```
Hourly Purchase Counts:
00:00 - 3802 purchases
01:00 - 2122 purchases
02:00 - 1180 purchases
03:00 - 694 purchases
04:00 - 683 purchases
05:00 - 1177 purchases
06:00 - 2188 purchases
07:00 - 3696 purchases
08:00 - 5813 purchases
09:00 - 8080 purchases
10:00 - 9981 purchases
11:00 - 11393 purchases
12:00 - 11594 purchases
13:00 - 11207 purchases
14:00 - 10312 purchases
15:00 - 9646 purchases
16:00 - 9441 purchases
17:00 - 10050 purchases
18:00 - 11146 purchases
19:00 - 11757 purchases
20:00 - 11507 purchases
21:00 - 10219 purchases
22:00 - 8117 purchases
23:00 - 5738 purchases
```



Key Insights

1. Customer activity follows a clear daily rhythm, starting low in the early morning, building momentum mid-morning, peaking in the afternoon and early evening, and tapering off gradually at night.

2. Consistently high customer activity is observed between 11:00 AM and 9:00 PM, with the most significant spikes from:

- 11:00 AM to 2:00 PM (Peak hours: 11 AM – 11,393; 12 PM – 11,594; 1 PM – 11,207)

- 6:00 PM to 8:00 PM (Evening peak: 6 PM – 11,146; 7 PM – 11,757; 8 PM – 11,507)

3. Low engagement periods are between 2:00 AM and 6:00 AM, where customer activity is minimal.

Strategic Recommendations

To maximize conversions and get the most out of your ad spend:

1. Targeted Promotions Strategy

- Run high-budget, high-impact campaigns between 11:00 AM and 2:00 PM, when

purchase intent is at its peak.
- Complement with reminder or follow-up ads in the evening (6:00 PM – 9:00 PM) when users have more downtime.
- Use lighter awareness-based campaigns in the morning (9:00 AM – 11:00 AM) to set the stage for peak time conversions.

2. Scheduling Tactics

- Segment marketing messages by hour: Informational in the morning, persuasive in the afternoon, urgency-based in the evening.

Recommended Time Slots for Ads & Promotions - Based on Peak Customer Activity

1. Primary Peak Block (Most Impactful):

- 11:00 AM – 2:00 PM — Lunch-time browsing and decision-making.

2. Secondary Peak Block (Evening Push):

- 6:00 PM – 9:00 PM — After-work, relaxed shopping mindset.

3. Morning Ramp-up Slot (Pre-Peak Nudge):

- 9:00 AM – 11:00 AM — Prepping customers mentally before their lunch-time conversion.