🎾 Tennis Central Sales Analysis 📊

🏬 Business Context

Tennis Central, a premier sports store, is expanding its operations by opening a new store at another club. To ensure a **profitable and efficient launch**, a **comprehensive sales analysis** is crucial.

📌 Project Objective

This project focuses on:
✅ **Analyzing sales trends** to identify high-demand products 📈
✅ **Understanding customer behavior** to optimize stocking strategies 🛍️
✅ **Evaluating product performance** for data-driven decision-making 🎯

The insights derived will **support strategic business decisions** to enhance operational efficiency and profitability at the new location. 🚀

Step 1: Importing Libraries and Loading the Dataset

Overview:

We will begin by importing the necessary libraries and loading the dataset into a pandas DataFrame.

In [10]:
```
# Step 1: Importing Libraries and Load the Dataset
import pandas as pd
import matplotlib.pyplot as plt

# Defining file path
file_path = r"D:TLTC Sales Summary Q4 2024.csv"

# Loading the dataset
data = pd.read_csv(file_path)

# Displaying the first few rows of the dataset
print(data.head())
```

```
  Sale Completed Date Customer Company Customer Type  \
0          01-10-2024           R1609A          TLTC
1          01-10-2024            Y1537          TLTC
2          01-10-2024            W1884          TLTC
3          01-10-2024            S0953          TLTC
4          01-10-2024            T2768          TLTC

                Item Description Sale Completed Time  Sale Lin
e Quantity Sold  \
0  Wilson U.S. Open Regular Duty    01-10-2024 04:53
1
1  Wilson U.S. Open Regular Duty    01-10-2024 04:57
1
2  Wilson U.S. Open Regular Duty    01-10-2024 05:00
1
3  Wilson U.S. Open Regular Duty    01-10-2024 05:30
1
4    Wilson U.S. Open Extra Duty    01-10-2024 06:06
1

  Sale Line  Total
0              7.2
1              7.2
2              7.2
3              7.2
4              7.2
```

Step 2: Dropping Unnecessary Columns

Overview:

In this step, dropping the columns **'Sale Completed Date'** and **'Customer Type'** as they are not required for analysis. Removing these unnecessary columns helps in decluttering the dataset and focusing only on relevant information.

In [11]:

```
# Step 2: Dropping Unnecessary Columns
# Dropping 'Sale Completed Date' and 'Customer Type'
data.drop(columns=["Sale Completed Date", "Customer Type"], inpl

# Displaying the first few rows of the updated dataset
print(data.head())
```

```
  Customer Company               Item Description Sale Complet
ed Time  \
0           R1609A  Wilson U.S. Open Regular Duty    01-10-202
4 04:53
1            Y1537  Wilson U.S. Open Regular Duty    01-10-202
4 04:57
2            W1884  Wilson U.S. Open Regular Duty    01-10-202
4 05:00
3            S0953  Wilson U.S. Open Regular Duty    01-10-202
4 05:30
4            T2768    Wilson U.S. Open Extra Duty    01-10-202
4 06:06

   Sale Line Quantity Sold Sale Line  Total
0                        1              7.2
1                        1              7.2
2                        1              7.2
3                        1              7.2
4                        1              7.2
```

Step 3: Renaming Columns

Overview:

In this step, renaming the following columns to make them more descriptive and consistent:

1. 'Customer Company' -> 'Customer_ID'

2. 'Item Description' -> 'Item_Description'

3. 'Sale Line Quantity Sold' -> 'Quantity_Sold'

4. 'Sale Line Total' -> 'Total_Sales'

Renaming columns ensures clarity and maintains consistency in the dataset.

In [12]:
```
# Step 3: Renaming Columns
data.rename(columns={
    "Customer Company": "Customer_ID",
    "Item Description": "Item_Description",
    "Sale Line Quantity Sold": "Quantity_Sold",
    "Sale Line  Total ": "Total_Sales"
}, inplace=True)

# Displaing the first few rows of the updated dataset
print(data.head())
```

```
  Customer_ID               Item_Description Sale Completed Ti
me  \
0      R1609A  Wilson U.S. Open Regular Duty    01-10-2024 04:
53
1       Y1537  Wilson U.S. Open Regular Duty    01-10-2024 04:
57
2       W1884  Wilson U.S. Open Regular Duty    01-10-2024 05:
00
3       S0953  Wilson U.S. Open Regular Duty    01-10-2024 05:
30
4       T2768    Wilson U.S. Open Extra Duty    01-10-2024 06:
06

   Quantity_Sold  Total_Sales
0              1          7.2
1              1          7.2
2              1          7.2
3              1          7.2
4              1          7.2
```

Step 4: Splitting the 'Sale Completed Time' Column

Overview:

In this step, splitting the 'Sale Completed Time' column into two separate columns:

1. 'Date_Of_Sale' - Contains only the date in DD/MM/YYYY format.

2. 'Time_Of_Sale' - Contains only the time in HH:MM format.

This will make it easier to analyze sales based on date and time separately.

```
In [13]:
# Step 4: Splitting the 'Sale Completed Time' Column
# Splitting 'Sale Completed Time' into 'Date_Of_Sale' and 'Time_
data[['Date_Of_Sale', 'Time_Of_Sale']] = data['Sale Completed Ti

# Dropping the original 'Sale Completed Time' column
data.drop(columns=['Sale Completed Time'], inplace=True)

# Ensuring 'Date_Of_Sale' is in the format DD/MM/YYYY
data['Date_Of_Sale'] = pd.to_datetime(data['Date_Of_Sale'], errc

# Displaying the first few rows of the updated dataset
print(data.head())
```

```
  Customer_ID               Item_Description  Quantity_Sold To
tal_Sales  \
0      R1609A  Wilson U.S. Open Regular Duty              1
7.2
1       Y1537  Wilson U.S. Open Regular Duty              1
7.2
2       W1884  Wilson U.S. Open Regular Duty              1
7.2
3       S0953  Wilson U.S. Open Regular Duty              1
7.2
4       T2768    Wilson U.S. Open Extra Duty              1
7.2

  Date_Of_Sale Time_Of_Sale
0   10/01/2024        04:53
1   10/01/2024        04:57
2   10/01/2024        05:00
3   10/01/2024        05:30
4   10/01/2024        06:06

C:\Users\rushi\AppData\Local\Temp\ipykernel_33540\2021287005.p
y:9: UserWarning: Parsing dates in DD/MM/YYYY format when dayf
irst=False (the default) was specified. This may lead to incon
sistently parsed dates! Specify a format to ensure consistent
parsing.
  data['Date_Of_Sale'] = pd.to_datetime(data['Date_Of_Sale'],
errors='coerce').dt.strftime('%d/%m/%Y')
```

Step 5: Checking Data Types

Overview:

Before checking for missing values, inspecting the data types of each column to ensure they are appropriate for analysis. If needed, we will correct the data types to avoid errors during subsequent operations.

In [14]:

```
# Step 5: Checking Data Types
# Displaying the data types of each column
print(data.dtypes)
```

```
Customer_ID         object
Item_Description    object
Quantity_Sold        int64
Total_Sales         object
Date_Of_Sale        object
Time_Of_Sale        object
dtype: object
```

Step 6: Correcting Data Types

Overview:

Correcting the data types of the following columns to ensure consistency and facilitate analysis:

1. **'Customer_ID':** No changes needed as it remains an object.
2. **'Item_Description':** No changes needed as it remains an object.
3. **'Quantity_Sold':** Already in an int type, so no changes are required.
4. **'Total_Sales':** Converting to numeric type for quantitative analysis.
5. **'Date_Of_Sale':** Converting to datetime format for date-based analysis.
6. **'Time_Of_Sale':** No changes needed, as treating it as an object for time-related grouping.

In [15]:

```
# Step 6: Correcting Data Types

# Convert 'Total_Sales' to numeric
data['Total_Sales'] = pd.to_numeric(data['Total_Sales'], errors=

# Convert 'Date_Of_Sale' to datetime with specified format to av
data['Date_Of_Sale'] = pd.to_datetime(data['Date_Of_Sale'], form

# Display the updated data types
print(data.dtypes)
```

```
Customer_ID                 object
Item_Description            object
Quantity_Sold                int64
Total_Sales                float64
Date_Of_Sale        datetime64[ns]
Time_Of_Sale                object
dtype: object
```

Step 7: Checking for Null Values

Overview:

In this step, checking the number of null values in each column of the dataset. This helps identify any missing data that needs to be addressed before proceeding with further analysis

In [16]:
```
# Step 7: Checking for Null Values
# Counting the number of null values in each column
null_counts = data.isnull().sum()

# Displaying the null counts
print(null_counts)
```

```
Customer_ID         11
Item_Description    20
Quantity_Sold        0
Total_Sales          1
Date_Of_Sale         0
Time_Of_Sale         0
dtype: int64
```

Step 8: Checking for Outliers and Irregular Values

Overview

Before proceeding with data cleaning, we need to identify irregular values and outliers in key columns.
This step helps us understand which issues exist before handling them.

Checking

1. Customer_ID

- Checking if **all IDs start with an alphabet** (valid IDs should not be entirely numeric).
- Identifying any **invalid customer IDs** that might indicate system-generated or erroneous entries.

2. Quantity_Sold

- Checking for **transactions with a quantity of 0**, as they may indicate incomplete or invalid sales.

3. Total_Sales

- Identifying transactions where **Total_Sales is 0**, as these might be irrelevant for analysis.

```
In [17]:  # Step 8: Checking for Outliers and Irregular Values

          # Checking Customer_ID
          print("\nChecking 'Customer_ID' Column:")
          invalid_customer_ids = data[~data['Customer_ID'].str.match(r'^[A
          print(f"Invalid Customer_IDs (not starting with a letter): {inva

          # Checking Quantity_Sold
          print("\nChecking 'Quantity_Sold' Column:")
          zero_quantity = data[data['Quantity_Sold'] == 0]
          print(f"Number of rows where Quantity_Sold is 0: {len(zero_quant

          # Checking Total_Sales
          print("\nChecking 'Total_Sales' Column:")
          zero_sales = data[data['Total_Sales'] == 0]
          print(f"Number of rows where Total_Sales is 0: {len(zero_sales)}
```

```
Checking 'Customer_ID' Column:
Invalid Customer_IDs (not starting with a letter): ['3258' nan
'3273' '3213' '3300' '3244' '3240' '3285']

Checking 'Quantity_Sold' Column:
Number of rows where Quantity_Sold is 0: 1

Checking 'Total_Sales' Column:
Number of rows where Total_Sales is 0: 23
```

Step 9: Removing Nulls, Outliers, and Cleaning Columns

Overview:

To ensure the dataset is clean and structured, this step focuses on:

1. **Handling `Customer_ID`** : Removing nulls, outliers, and invalid values.
2. **Handling `Item_Description`** : Removing nulls.
3. **Handling `Quantity_Sold`** : Removing invalid transactions where quantity is 0.
4. **Cleaning and Handling `Total_Sales`** :
    - Removing commas and converting it to numeric.
    - Removing transactions where `Total_Sales` is 0.

Actions Taken:

1. Handling `Customer_ID`

- Dropping rows where `Customer_ID` is **null**.
- Removing **outlier** where `Customer_ID` is **"Toronto Lawn Coach"**.

- Removing `Customer_IDs` that **do not start with an alphabet** (likely system-generated or errors).

2. Handling `Item_Description`

- Dropping rows where `Item_Description` is **null** to ensure all transactions have a valid product description.

3. Handling `Quantity_Sold`

- Removing rows where `Quantity_Sold` is **0**, as these are incomplete transactions.

4. Cleaning and Handling `Total_Sales`

- Removing commas (if any) and converting `Total_Sales` to numeric.
- Removing rows where `Total_Sales` is **0**, as these transactions have no revenue.

In [18]:

```
# Step 9: Removing Nulls, Outliers, and Cleaning Columns

# Handling 'Customer_ID'
data = data[data['Customer_ID'].notnull()]  # Removing rows with
data = data[data['Customer_ID'] != "Toronto Lawn Coach"]  # Remo
data = data[data['Customer_ID'].str.match(r"^[A-Za-z]", na=False

# Handling 'Item_Description'
data = data[data['Item_Description'].notnull()]  # Removing rows

# Handling 'Quantity_Sold'
data = data[data['Quantity_Sold'] != 0]  # Removing rows where '

# Cleaning and Handling 'Total_Sales'
data['Total_Sales'] = data['Total_Sales'].astype(str).str.replac
data = data[data['Total_Sales'] != 0]  # Removing rows where 'To

# Display the first few rows to confirm changes
print("\nFirst Few Rows After Cleaning:")
print(data.head())
```

```
First Few Rows After Cleaning:
  Customer_ID                Item_Description  Quantity_Sold  T
otal_Sales  \
0      R1609A  Wilson U.S. Open Regular Duty              1
7.2
1       Y1537  Wilson U.S. Open Regular Duty              1
7.2
2       W1884  Wilson U.S. Open Regular Duty              1
7.2
3       S0953  Wilson U.S. Open Regular Duty              1
7.2
4       T2768    Wilson U.S. Open Extra Duty              1
7.2

  Date_Of_Sale Time_Of_Sale
0   2024-01-10        04:53
1   2024-01-10        04:57
2   2024-01-10        05:00
3   2024-01-10        05:30
4   2024-01-10        06:06
```

In [21]:

```
# Path to save the file
output_path = r"D:\BIA\SEM 4\Capstone Course - (BIA 5450 - 0LA)

# Save the DataFrame to Excel
data.to_excel(output_path, index=False)

print(f"File successfully saved to:\n{output_path}")
```

```
File successfully saved to:
D:\BIA\SEM 4\Capstone Course - (BIA 5450 - 0LA) H114 - Samer
H. Al-Obaidi\Project Work\CLEAN\Cleaned_Tennis_Data.xlsx
```

```
In [ ]:
```

```
In [ ]:
```