BILLSON INDIA: ANALYSIS

About Billson India:

Billson India, established in **2018** and headquartered in **Chandigarh**, is a trusted provider of advanced **Point of Sale (POS) solutions** for the *Food & Beverage*, *Hospitality*, and *Retail* sectors.

With a focus on **reliable products**, **innovative software**, and **essential peripherals**, Billson India helps businesses streamline operations, improve billing efficiency, and manage inventory with ease.

Guided by **service excellence** and **genuine business guidance**, the company proudly serves a loyal customer base of **around 1,500 clients** in its catered regions, building long-term partnerships through **personalized support** and **transparent consultation**.

Business Problem to Solve

Our company, a provider of point-of-sale solutions for hotels, F&B, and retail businesses, faces the ongoing challenge of sustaining robust sales growth in a competitive market. Recent sales data reveals fluctuating revenue trends and uneven performance across products, customer segments, and payment modes. It is therefore critical to identify underlying factors driving sales declines, pinpoint underperforming items and customers, and discover actionable strategies to maximize revenue, reduce churn, and capitalize on market opportunities

Data Description

For this project, we are working with **two interlinked datasets** that capture sales transactions at both the **bill level** and the **item level**. These datasets together provide a holistic view of our business sales performance, enabling both aggregated and granular analysis.

---

1. BILL_WISE_DATA

This dataset contains information at the **bill or invoice level**. Each row represents a single customer bill and summarizes the total value, tax components, payment mode, and other transaction details.

Columns:

- **B_DATE** – Bill date (transaction date)
- **B_NO / B_C** – Bill number and bill code identifiers
- **B_PARTY** – Name of the customer/business purchasing (client)
- **GST_NO** – GST registration number of the customer
- **TAXABLE** – Taxable amount before GST
- **NET_AMT** – Final bill amount after taxes
- **SGST, CGST, IGST, CESS, COMP_GST, TCS** – Applicable tax components in the bill
- **CSH_1_CRD_2** – Indicator for payment mode (cash or credit)
- **SAO** – Special adjustment or other charges
- **Unnamed: 15-17** – Empty or unused columns, likely placeholders or system exports

2. ITEM_WISE_DATA

This dataset contains information at the **item or product level** for each bill. Each row represents a specific product sold in a bill, with quantity, price, and tax details.

Columns:

- **B_DATE** – Bill date (same link as in BILL_WISE_DATA)
- **B_NO / BILL_NO** – Bill identifiers to join with bill-level dataset
- **B_PARTY** – Customer/business name (same as in bill data)
- **S_ITEM** – Name/description of the item sold
- **S_PCS / S_QTY** – Number of pieces and quantity sold
- **UNIT** – Unit of measurement (e.g., pcs, kg, liters)
- **S_RATE** – Selling rate per unit
- **DISOUNT** – Discount applied to the item
- **TAXABLE** – Taxable value for this specific item line
- **GST, S_SGST, S_CGST, IGST, S_CESS** – GST and other applicable taxes for the item
- **S_TYPE** – Sales type/category (if applicable)
- **NET_AMT** – Final amount for the item line after discounts and taxes
- **ONF** – Additional status flag or code (meaning to be defined from business context)

3. CUSTOMER DATA

To enhance our sales analysis and enable location-based insights, we are also using an additional dataset: `customer_data`.

For this stage, we are keeping only the two essential columns:

- `customer` → The name of the customer or business
- `City` → The city where the customer operates

Important Python Liberaries to Import | Reading Files | Basic Data Exploration

```
#IMPORTING LIBERARIES:

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re
import warnings
warnings.filterwarnings('ignore')

from scipy import stats
from scipy.stats import skew, kurtosis
import statsmodels.api as sm
from scipy.stats import norm
from scipy.stats import ttest_ind, shapiro, levene, f_oneway, chi2_contingency
import datetime as dt
from datetime import datetime

import matplotlib as mpl

# Disable scientific notation globally for all plots
mpl.rcParams['axes.formatter.useoffset'] = False
mpl.rcParams['axes.formatter.use_locale'] = False
mpl.rcParams['axes.formatter.limits'] = (-999999, 999999)  # practically disables sci notation

#READING THE FILE:

billdata = pd.read_csv('bill_wise_data_cleaned.csv')
```

```
itemdata = pd.read_csv('item_wise_data_cleaned.csv')
customerdata = pd.read_csv('customer_data_cleaned.csv')
finaldata = pd.read_csv('combined_final_data_cleaned.csv')

#other necessary code to implement before analysis

pd.set_option('display.float_format', '{:,.0f}'.format)
# pd.set_option('display.max_rows', None)
pd.reset_option('display.max_rows')
```

```
billdata.head()
```

	date	bill_no	customer	total_amount	cash_online	payment_mode
0	2020-05-07	1719	SMART BUY DEPARTMENTAL STORE	32,000	2	Online
1	2020-05-11	1720	KHALSA CONFECTIONERS	23,000	2	Online
2	2020-05-14	1721	NAB ENTERPRISES	47,000	2	Online
3	2020-05-18	1722	DOWN TOWN	7,080	2	Online
4	2020-05-19	1723	DOWN TOWN	47,920	2	Online

```
# Convert bill date to datetime format
billdata['date'] = pd.to_datetime(billdata['date'].str.strip(), format='%Y-%m-%d', errors='coerce')
```

```
billdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4504 entries, 0 to 4503
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   date          4504 non-null   datetime64[ns]
 1   bill_no       4504 non-null   int64
 2   customer      4504 non-null   object
 3   total_amount  4504 non-null   float64
 4   cash_online   4504 non-null   int64
 5   payment_mode  4504 non-null   object
dtypes: datetime64[ns](1), float64(1), int64(2), object(2)
memory usage: 211.3+ KB
```

```
itemdata.head()
```

	date	bill_no	item	quantity
0	2020-05-07	1719	NS POS ASPIRE	1
1	2020-05-07	1719	SUBSCRIPTION CHARGES - 1 YEAR	1
2	2020-05-11	1720	ELECTRONIC CASH REGISTER T-90	1
3	2020-05-14	1721	NS POS PRO	1
4	2020-05-14	1721	SUBSCRIPTION CHARGES - 1 YEAR	1

```
print(itemdata.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7446 entries, 0 to 7445
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   date      7446 non-null   object
 1   bill_no   7446 non-null   int64
 2   item      7446 non-null   object
 3   quantity  7446 non-null   int64
dtypes: int64(2), object(2)
memory usage: 232.8+ KB
None
```

```
# Convert bill date to datetime format
itemdata['date'] = pd.to_datetime(itemdata['date'].str.strip(), format='%Y-%m-%d', errors='coerce')
```

```
customerdata.head()
```

	customer	city
0	100 YARDS DEPARTMENTAL STORE	anantnag
1	2R INTERNATIONALS	zirakpur
2	A N COMPONENTS	chandigarh
3	A S THAKUR	sirmour
4	A SQUARE MEDIA	amritsar

```
print(customerdata.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1486 entries, 0 to 1485
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   customer  1486 non-null   object
 1   city      1438 non-null   object
dtypes: object(2)
memory usage: 23.3+ KB
None
```

FEATURE ENGINEERING ON BILLDATA

```
billdata.head()
```

	date	bill_no	customer	total_amount	cash_online	payment_mode
0	2020-05-07	1719	SMART BUY DEPARTMENTAL STORE	32,000	2	Online
1	2020-05-11	1720	KHALSA CONFECTIONERS	23,000	2	Online
2	2020-05-14	1721	NAB ENTERPRISES	47,000	2	Online
3	2020-05-18	1722	DOWN TOWN	7,080	2	Online
4	2020-05-19	1723	DOWN TOWN	47,920	2	Online

```
# **CREATING NEW TIME-RELATED COLUMNS FOR OUR UNDERSTANDING AND TIME SERIES ANALYSIS:
billdata['year'] = billdata['date'].dt.year
billdata['month'] = billdata['date'].dt.month
billdata['day'] = billdata['date'].dt.day
billdata['month_name'] = billdata['date'].dt.month_name()
billdata['day_of_week'] = billdata['date'].dt.dayofweek
billdata['day_name'] = billdata['date'].dt.day_name()
billdata['quarter'] = billdata['date'].dt.quarter
billdata['week_of_year'] = billdata['date'].dt.isocalendar().week
```

```
billdata.head(1)
```

	date	bill_no	customer	total_amount	cash_online	payment_mode	year	month	day	month_name	day_of_week	day_name	quarter	week_of_year
0	2020-05-07	1719	SMART BUY DEPARTMENTAL STORE	32,000	2	Online	2020	5	7	May	3	Thursday	2	19

```
# Calculate bill count per customer
customer_bill_counts = billdata.groupby('customer')['bill_no'].nunique().reset_index()
customer_bill_counts.rename(columns={'bill_no': 'bill_count'}, inplace=True)


def categorize_customer(bill_count): #define a function to categorize customer type based on bill count
    if bill_count > 30:
        return 'Agent/reseller/dealer'
    elif 4 <= bill_count < 30:
        return 'Loyal'
    else:
        return 'Casual'

customer_bill_counts['customer_type'] = customer_bill_counts['bill_count'].apply(categorize_customer)

billdata = pd.merge(billdata, customer_bill_counts[['customer', 'customer_type']], on='customer', how='left')

display(billdata.head())
```

	date	bill_no	customer	total_amount	cash_online	payment_mode	year	month	day	month_name	day_of_week	day_name	quarter	week_of_year	customer_type
0	2020-05-07	1719	SMART BUY DEPARTMENTAL STORE	32,000	2	Online	2020	5	7	May	3	Thursday	2	19	Casual
1	2020-05-11	1720	KHALSA CONFECTIONERS	23,000	2	Online	2020	5	11	May	0	Monday	2	20	Loyal
2	2020-05-14	1721	NAB ENTERPRISES	47,000	2	Online	2020	5	14	May	3	Thursday	2	20	Casual
3	2020-05-18	1722	DOWN TOWN	7,080	2	Online	2020	5	18	May	0	Monday	2	21	Casual
4	2020-05-19	1723	DOWN TOWN	47,920	2	Online	2020	5	19	May	1	Tuesday	2	21	Casual

FEATURE ENGINEERING ON ITEMDATA

```

itemdata["item"] = itemdata["item"].astype(str).str.upper().str.strip()

item_to_category = {
    "ELECTRONIC CASH REGISTER T-90":"ECR", "THERMAL PRINTER MECH. 3 INCH":"THERMAL PRINTER", "ELECTRONIC CASH REGISTER T-20":"ECR", "ELECTRONIC CASH REGISTER ZIP-20":"ECR", "THERMAL PRINTER 3 INCH RPT":"THERMAL PRINTER",
    "ELECTRONIC CASH DRAWER": "CASH DRAWER", "ELECTRONIC CASH REGISTER T-10": "ECR", "RPT-260IV SU CLASSIC THERMAL PRINTER": "THERMAL PRINTER", "HAND HELD TERMINAL HANDY-30": "ECR", "THERMAL PRINTER MECH. 2 INCH": "SPARE
    PART", "PRINTER GEARS SET": "SPARE PART", "GEAR PF20202-10F": "SPARE PART", "GEAR PF20203-02F": "SPARE PART", "GEAR PF20201-01F": "SPARE PART", "GEAR PF20204-01F": "SPARE PART", "NS-SALES APP SUBS. SOFTWARE-": "AMC",
    "MJ-2818B 2D BAR CODE SCANNER": "SCANNER", "ELECTRONIC CASH REGISTER ZIP-50": "ECR", "RPT-260IV UE ADVANCE THERMAL PRINTER": "THERMAL PRINTER", "NS POS ASPIRE": "POS TERMINAL", "NS ELITE POS": "POS TERMINAL", "TSC
    TE-200 BAR CODE PRINTER": "BARCODE PRINTER", "TSC TTP-244 TE BARCODE PRINTER": "BARCODE PRINTER", "CASH DRAWER MODEL CR-410B": "CASH DRAWER", "AMC POS SOFTWARE": "AMC", "THERMAL PAPER ROLL 3\\": "CONSUMABLE",
    "CHARGER HANDY-30": "SPARE PART", "POSIFLEX RUGTEK RP326 PRINTER": "THERMAL PRINTER", "SUBSCRIPTION CHARGES - 1 YEAR": "AMC", "THERMAL PAPER ROLL-2": "CONSUMABLE", "LABEL STICKER ROLL": "CONSUMABLE", "RIBBON WAX
    W333": "CONSUMABLE", "POSIFLEX PS 3616 POS TERMINAL": "POS TERMINAL", "ELECTRONIC CASH REGISTER T-50": "ECR", "POSIFLEX RTP 2 RUGTEK 15.6 ANDROID POS": "POS TERMINAL", "NS POS PRO": "POS TERMINAL", "3.6V VERTA BTY":
    "SPARE PART", "BILL COUNTER / C-6": "CURRENCY COUNTER", "POKETPOS BASIC LICENSE": "SOFTWARE LICENSE", "SENSIBLE-10": "POS TERMINAL", "RPT-8150K I5 TOUCH POS SYSTEM": "POS TERMINAL", "BAR CODE SCANNER HONEYWELL 1D
    5145": "SCANNER", "WIN 10 IOT LTSC": "WINDOWS", "NS-SALES APP SUBS. SOFTWARE-1 YR.": "AMC", "LCD / LED MONITOR": "DISPLAY", "SERVICE CHARGES": "SERVICE", "BATTERY 11.1V LI-ON 2200 MAH": "SPARE PART", "NS-POS PRO 2":
    "POS TERMINAL", "AMC SOFTWARE": "AMC", "THERMAL PRINTER RP-80": "THERMAL PRINTER", "AMC ANDROID APP": "AMC", "BILLING SOFTWARE INSTALLATION CHARGES": "SERVICE", "REPAIR CHARGES": "SERVICE", "KLS 1/2 KEY LOCK": "SPARE
    PART", "7.4 V/2200 MAH BATTERY LI-ION": "SPARE PART", "KEY BOARD-ECR-SET": "SPARE PART", "RSP 815 CORE I3 TOUCH POS SYSTEM": "POS TERMINAL", "RPT 260V BT+5 PRINTER": "THERMAL PRINTER", "24 V DC POWER ADAPTOR FOR
    RPT": "SPARE PART", "UPS": "COMPUTER PART", "POSIFLEX RUGTEL CR 410B CASH DRAWER": "CASH DRAWER", "RPT 815 CORE I5 TOUCH POS SYSTEM": "POS TERMINAL", "ENTERPRISE DEVICE LIC.": "SOFTWARE", "THERMAL PAPER ROLL-":
    "CONSUMABLE", "COMPUTER MOTHERBOARD PLUS": "SPARE PART", "KEY PAD HANDY-30": "SPARE PART", "SUBSCRIPTION": "AMC", "POSIFLEX POS RT-6015 G2": "POS TERMINAL", "SMALL HANDY BILL COUNTER A LCD": "OTHER", "GRAPHY LCD
    H-30": "SPARE PART", "POSIFLEX RUGTEK LS-3000R SCANNER ID-USB": "SCANNER", "BARCODE SCANNER": "SCANNER", "PRINTER HEAD FOR RPT 250/280": "SPARE PART", "ENTERPRISE SERVER LICENSE": "SOFTWARE", "SOFTWARE FIRMWARE-----
    HEX FILE": "SOFTWARE", "CLOUD BACKUP": "SOFTWARE", "RPT-8150K COMPUTER POS SYSTEM I5": "POS TERMINAL", "POSIFLEX-RP15, 15.6 FULL HD SCREEN": "POS TERMINAL", "POSIFLEX TOUCH POS RT5015": "POS TERMINAL", "PAPER TRANS.
    COVER HOLDER": "SPARE PART", "BASE BLANK PLASTIC BODY": "SPARE PART", "POSIFLEX PS 3616 Q POS TERMINAL": "POS TERMINAL", "3V6 60MAH BATTERY": "SPARE PART", "MOUSE": "COMPUTER ACCESSORY", "ZENPOS SOFTWARE APP":
    "SOFTWARE", "POSIFLEX PS 3616 D 128GB POS TERMINAL": "POS TERMINAL", "DC DC MINI POWER SUPPLY": "SPARE PART", "POSIFLEX RUGTEK POS RTP-4": "POS TERMINAL", "ELITE MONITOR POS": "POS TERMINAL", "DASHBOARD LICENSE":
    "SOFTWARE", "CASH DRAWER GS-405B": "CASH DRAWER", "IC HAT 1016": "SPARE PART", "DC-PIN": "SPARE PART", "MJ-2818C-2D WIRELESS SCANNER": "SCANNER", "ELECTRONIC CASH REGISTER": "ECR", "SWITCH": "SPARE PART", "TTP 244 TE
    BAR CODE PRINTER": "BARCODE PRINTER", "KEYBOARD MOUSE KIT WIRED": "COMPUTER ACCESSORY", "ANDROID SOFTWARE APP.": "SOFTWARE", "TSC BARCODE PRINTER": "BARCODE PRINTER", "PALMTECH AMPHIBIA DEVICE HANDY-": "POS
    TERMINAL", "KEY BOARD SET": "COMPUTER ACCESSORY", "PRINTER FLAP T-90": "SPARE PART", "INVENTORY SERVER LICENSE": "SOFTWARE", "MAIN DISPLAY": "SPARE PART", "POWER ADOPTER": "SPARE PART", "TOP BLANK PLASTIC ECR":
    "SPARE PART", "PARK RAHI LOGIN SUBSCRIPTION PER MONTH": "AMC", "DIGITAL MENU WITH ORDERING": "SOFTWARE", "RPT 815 CORE I3 TOUCH POS SYSTEM": "POS TERMINAL", "POSIFLEX RUGTEK RP-80 V1": "THERMAL PRINTER", "MJ-2818D 2D
    EXPERT SCANNER": "SCANNER", "INSTALLATION CHARGES OF SOFTWARE": "SERVICE", "IC74 HCT138 SM": "SPARE PART", "DC DC POWER SUPPLY ASSLY OF ECR": "SPARE PART", "INVENTORY STORE LICENSE": "SOFTWARE", "FREIGHT / COURIER
    CHARGES": "SERVICE", "KEY TOP CHERY": "SPARE PART", "RUGTEK SCANNER CD 3200U": "SCANNER", "ANDROID PC 4GB/16GB": "POS TERMINAL", "PARK RAHI LOGIN SUBSCRIPTION": "AMC", "RUGTEK BP-02 MOBILE PRINTER": "THERMAL
    PRINTER", "ANTIVIRUS": "COMPUTER ACCESSORY", "IC LB 1845": "SPARE PART", "AMC MENSON POS SOFTWARE": "AMC", "KEYBOARD": "COMPUTER ACCESSORY", "PRINTER INTERFACE CARD T-50": "SPARE PART", "BATTERY FOR HANDY": "SPARE
    PART", "MONEY COUNTER -600": "NOTE COUNTING MACHINE", "HARD DISK SSD": "SPARE PART", "PC-CPU-CABINET ZEBRONIX": "COMPUTER ACCESSORY", "POSIFLEX RUGTEK RP-82 PRINTER": "THERMAL PRINTER", "WIN 10 IOT": "WINDOWS",
    "SMART CARDS": "OTHER", "ELECTRONIC CASH REGISTER SPARES": "SPARE PART", "ESSAE POS 815B": "POS TERMINAL", "CABINET BLANK H-30": "SPARE PART", "PRINTER PLASTIC COVER": "SPARE PART", "KEY SWITCH": "SPARE PART", "INTER
    FACE T-90": "SPARE PART", "SNAP BIZ DONGLE": "OTHER", "CPU DC 2ND GEN": "SPARE PART", "RUGTEK 2D SCANNER LS3002": "SCANNER", "HEX FILE REPLACEMENT CHARGES.": "SERVICE", "COMPUTER DUAL CORE M.BOARD": "COMPUTER PART",
    "IC 74 LVX 4066": "SPARE PART", "IMIN M2-202 MOBILE POS DEVICE": "POS TERMINAL", "KEY CAP CHERRY": "SPARE PART", "ON OFF SWITCH": "SPARE PART", "PRINTER FLAP T-90": "SPARE PART", "PRINTER INTERFACE CARD T-90": "SPARE
    PART", "KEY BOARD T-10/T-40": "SPARE PART", "COMPUTER DC 2ND GEN, 240 SSD": "COMPUTER", "KDS LICENCE": "SOFTWARE", "DC-DC T-90": "SPARE PART", "KB MOUSE KIT WIRELESS": "COMPUTER PART", "KEY BOARD-ECR": "SPARE PART",
    "DOUBLE CAP TOP": "SPARE PART", "SB-001 ANDROID SOFTWARE": "SOFTWARE", "BASE BLANK PLASTIC EC": "SPARE PART", "UPS INTEX": "COMPUTER PART", "POSIFLEX POS 6216 I3": "POS TERMINAL", "16 PIN BOX WIRE": "SPARE PART",
    "MJ-2818C 2D WIRELESS BARCODE SCANNER": "SCANNER", "POSIFLEX POS PS6216": "POS TERMINAL", "PRINTER COVER T-90": "SPARE PART", "SMALL HANDY BILL COUNTER A LCD-0": "SPARE PART", "5 PIN CONN.M": "SPARE PART",
    "NS-SOFTWARE APPLICATION": "SOFTWARE", "SAMSUNG 65 LFD QM 65C TV": "TV", "IMIN D3-504 POS DEVICE": "POS TERMINAL", "CCM VMS ESSENTIAL VALUE COUNTER": "NOTE COUNTING MACHINE", "IMIN M2-PRO MOBILE POS DEVICE": "POS
    TERMINAL", "D-LINK CAT 6 CABLE BOX 100 MTR.": "OTHER", "RUGTEK BLUETOOTH SCANNER CD3200BTRC": "SCANNER", "VERTIV UPS ITON CX 600VA": "COMPUTER PART", "POSBANK APEXA EL": "POS TERMINAL", "THERMAL PRINTER TENAX S300N":
    "THERMAL PRINTER", "TOP BLNK PLST-H30": "SPARE PART", "PC MOTHERBOARD H61": "COMPUTER PART", "POSIFLEX TOUCH POS PS 3316E+ 128 GB": "POS TERMINAL", "POSIFLEX TOUCH POS PS 3316+ 64GB": "POS TERMINAL", "WINDOWS 10 IOT
    (EL) LICENCE": "WINDOWS", "PRINTER COVER T-20": "SPARE PART", "STAND FOR SCANNER CD 3200U": "OTHER", "MJ 9320 ALPHA 2D DESK TOP SCANNER": "SCANNER", "BILL POWER SOFTWARE APPLICATION": "SOFTWARE", "PRINTER FLAP":
    "SPARE PART", "CAP 104/105/106/PFSMD": "SPARE PART", "SMS KIT": "SOFTWARE", "NOTE COUNTER LR-6600": "NOTE COUNTING MACHINE", "ELECTRONIC TOUCH CASH REGISTER S-10": "POS TERMINAL", "SAMSUNG BE43C-H, 43": "TV",
    "SAMSUNG LFD 55 BE55D-H": "TV", "PRINTER HEAD": "SPARE PART", "WIRELESS DATA POS TERMINAL": "POS TERMINAL", "PRINTER PLATE PLASTIC": "SPARE PART", "KEY BOARD USB": "COMPUTER PART", "TOP PLASTIC BLANK T-10": "SPARE
    PART", "RAM 4GB DDR3": "COMPUTER PART", "3 THERMAL PRINTER XP-600 BT": "THERMAL PRINTER", "INTEL FAN ORIGINAL": "COMPUTER PART", "RUGTEK SCANNER 3002 WIRED": "SCANNER", "CABINET ENTER": "COMPUTER PART ", "SOFTWARE
    SHIFTING AND INSTALLATION": "SERVICE", "RUGTEK SCANNER LS3002 WIRED": "SCANNER", "TOP-T-90": "SPARE PART", "POSIFLEX LM3210E-B LCD SECOND DISPLAY": "ACCESSORY", "WIFI WF-680": "COMPUTER PART", "DC DC POWER SUPPLY
    ASSLY.": "SPARE PART", "PRINTER PLASTIC STAND": "SPARE PART", "TOUCH POS": "POS TERMINAL", "POSIFLEX PS 3316 E+ 64GB POS TERMINAL": "POS TERMINAL", "POS-615C, 128 SSD ESSAE MAKE": "POS TERMINAL", "WIN 10 IOT ENT 2019
    LTSC ENTRY": "WINDOWS", "LCD TOUCH MONITOR 15.6 INCH": "MONITOR", "USB INTERFACE CARD": "SPARE PART", "MSR 2106U POSIFLEX": "ACCESSORY", "DESK TOP HP 1901": "COMPUTER", "SWITCH 24 PORT D-LINK": "COMPUTER", "POS MAGIC
    MONITOR TOUCH": "POS TERMINAL", "RUGTEK SCANNER CD 3200 BT-E": "SCANNER", "HANDY STOPPER": "SPARE PART", "POSIFLEX 2ND DISPLAY TM-3010": "ACCESSORY", "BATTERY COVERY-30": "SPARE PART", "ROUTER WIFI": "OTHER",
    "PRINTER PLASTIC COVER T-10": "SPARE PART", "TOUCH SCREEN -MINI": "OTHER", "BRO. DR-2365": "OTHER", "TOUCH POS RSP-815": "POS TERMINAL", "PTP-II BLUE TOOTH PRINTER": "THERMAL PRINTER", "CPU FAN": "COMPUTER PART",
    "HARD DISK-8471": "COMPUTER PART", "CPU-PC MOTHERBOARD-INTEX CABINET": "COMPUTER PART", "A M C": "AMC", "D-LINK CAT6 UTP CABLE": "OTHER", "RAM ZEB H 81": "COMPUTER PART", "RAM 4GB": "COMPUTER PART", "MONITOR TRIPOD
    STAND": "OTHER", "SAMSUNG BE 65 TV": "TV", "SAMSUNG 43 LH43QB": "TV", "PRINTER INTERFACE CARD": "SPARE PART", "BIOMATRIC TIME WATCH BIO-20": "OTHER", "CAMERA": "OTHER", "WINDOWS NEO S/W": "SOFTWARE", "THERMAL PRINTER
    RUGTEK RP326": "THERMAL PRINTER", "POS MAGIC": "POS TERMINAL", "5 PIN (M)": "SPARE PART", "HEAD CLEANING PEN-PRINTER": "SPARE PART", "EXIDE BATTERY 7AH": "SPARE PART", "KEY LOCK SET FOR CR-410B": "SPARE PART",
    "SAMSUNG LFD 65 BUSINESS TV-BE65D-H": "TV", "WINDOW NEO SOFTWARE": "SOFTWARE", "CLOUD LINK": "SOFTWARE", "CONTROLLER IC": "SPARE PART", "POSIFLEX XT-4015 W/O OS. , XTJB0873": "POS TERMINAL", "NOTE COUNTING MACHINE":
    "NOTE COUNTING MACHINE", "IC MAX 232 SMD": "SPARE PART", "PLASTIC BEEDING": "SPARE PART", "REPAIR OF BILLING MACHINE": "SERVICE", "FUSE HOLDER": "SPARE PART", "3 PIN BATTERY CONNECTOR": "SPARE PART", "MACHINE REPAIR
    CHARGES": "SERVICE", "F18A-221144C0432WHP08-T": "SPARE PART", "INSTALLATION & TRAINING CHARGES": "SERVICE", "HONEYWELL SCANNER HF 680": "SCANNER", "RUGTEK THERMAL PRINTER MODEL RP 80B-USE": "THERMAL PRINTER",
    "THERMAL PAPER ROLL-2": "CONSUMABLE", "TOKEN DISPLAY": "OTHER", "BASE COVER HHT PLASTIC": "SPARE PART", "SOFTWARE APP.": "SOFTWARE", "BAR CODE SCANNER HONEYWELL 7120": "SCANNER", "ANDROID HANDHELD TERMINAL (TPS-900)
    ": "POS TERMINAL", "POS TT": "OTHER", "SAMSUNG QB43": "TV", "INTEGRATION PERMIT": "SOFTWARE", "I21M01-IMIN M2 PRO MOBILE POS DEVICE": "POS TERMINAL", "MJ 2818D NEO 2D TABLE TOP SCANNER": "SCANNER", "CHARGER ADAPTER
    WITH CABLE": "SPARE PART", "DC CHARGING JACK": "SPARE PART", "BELKIN 4 SOCKET SURGE PROTECTOR": "OTHER", "HEX FILE T-20 PROGRAMMING": "OTHER", "LCD MONITOR": "COMPUTER PART", "RUGTEK BLUETOOTH POS ENABLED 2D/1D
    SCANNER CD3200 BT-R": "SCANNER", "POSIFLEX THERMAL PRINTER PP8803 LAN": "THERMAL PRINTER", "NS POS-ELITE-PLUS": "POS TERMINAL", "POSIFLEX POS XT-6015C": "POS TERMINAL", "POS STAND": "ACCESSORY", "CAP VCR 14D/471 TH.
    H": "SPARE PART", "STACKER MOTOR": "SPARE PART", "RAM RILV0408": "COMPUTER PART", "PC-CPU-CABINET SET": "COMPUTER PART", "CABLE D-LINK": "OTHER", "SMS PACK LINK": "SOFTWARE", "CARTRIDGES ERC39": "OTHER", "POSBANK
    APEXA G POS J1900, 8GB/128 SSD": "POS TERMINAL", "MONITOR PHILIPS LED 15.6\\": "DISPLAY", "POSBANK APEXA G POS- J1900, 4GB/128 SSD": "POS TERMINAL", "SAMSUNG 55 LFD QM 55C": "TV", "SMPS 18": "COMPUTER PART",
    "POSIFLEX RUGTEK 2 BP02 (WITH POUCH)": "THERMAL PRINTER", "XP-C230 3 INCH MOB.LABEL PRINTER": "BARCODE PRINTER", "SAMSUNG 55 LED MONITOR QB": "TV", "XP-DT460B LABEL PRINTER": "BARCODE PRINTER", "DATALOGIC BAR CODE
    SCANNER M3551": "SCANNER", "GPRS KIT": "OTHER", "ODROID N2L": "COMPUTER PART", "INSTALLATION CHARGES FOR TICKETING MACHINE": "SERVICE", "DEL WIRELESS KIT KEYBOARD": "COMPUTER PART", "POSBANK SIDE DISPLAY 9.7\\":
    "ACCESSORY", "12V DC POWER ADAPTOR FOR POS": "SPARE PART", "PRINTER COVER T-90 TOP": "SPARE PART", "POSIFLEX PROG.KEYBOARD KB-6600": "ACCESSORY", "POSIFLEX LM4010B": "ACCESSORY", "RUGTEK SCANNER CD3200 BT-S":
    "SCANNER", "PRINTER PLATE STAND": "SPARE PART", "SAMSUNG LED TV 55\\": "TV", "HARD DISK SSD 120": "COMPUTER PART", "HEX FILE LOADER": "OTHER", "CPU T-90 SMS": "SPARE PART", "WINDOWS NEO SNAPBIZZ SOFTWARE":
    "SOFTWARE", "RITTAL 12U RACK-9790764": "OTHER", "SAMSUNG LS 22A334": "MONITOR", "CPU I5 4TH": "COMPUTER PART", "BUSINESS NOTEBOOK/ALC/240G10": "COMPUTER", "POSBANK SIDE DISPALY 9.7\\": "ACCESSORY", "WIN-11 IOT FOR PS
    QC POS": "WINDOWS", "CPU ZIP-20": "SPARE PART", "HEX FILE LOADING CHARGES": "SERVICE", "MSR 2106U  POSIFLEX": "ACCESSORY", "POSIFLEX PS 3616 D  128GB POS TERMINAL": "POS TERMINAL", "POSIFLEX PS 3616  POS TERMINAL":
    "POS TERMINAL", "POSIFLEX RTP 2 RUGTEK 15.6 ANDROID POS\"\"": "POS TERMINAL", "3 THERMAL PRINTER XP-600 BT\"\"": "THERMAL PRINTER", "SAMSUNG 55 LED MONITOR QB \"\"": "TV", "POSIFLEX RUGTEK 2 BP02 (WITH POUCH)\"\"":
    "THERMAL PRINTER", "SAMSUNG 55 LFD QM 55C\"\"": "TV", "SAMSUNG BE43C-H, 43 \"\"": "TV", "SAMSUNG LFD 55 BE55D-H\"\"": "TV", "SAMSUNG 43 LH43QB\"\"": "TV", "SAMSUNG 65 LFD QM 65C TV\"\"": "TV", "SAMSUNG LFD 65
    BUSINESS TV-BE65D-H\"\"": "TV", "ANDROID POS SOFTWARE": "SOFTWARE",
    "THERMAL PAPER ROLL 3\"":"CONSUMABLE","MONITOR PHILIPS LED 15.6\"":"DISPLAY", "POSBANK SIDE DISPLAY 9.7\"":"ACCESSORY",
    "SAMSUNG LED TV 55\"":"TV","SAMSUNG LED TV 55\"":"TV","POSBANK SIDE DISPALY 9.7\"":"ACCESSORY"
}


itemdata["product_category"] = itemdata["item"].map(item_to_category)


unmapped_items = itemdata.loc[itemdata["product_category"].isna(), "item"].unique()

print("Unmapped items:", unmapped_items)
```

```
Unmapped items: []
```

FEATURE ENGINEERING ON CUSTOMER DATA

```
# convert to string, coercing errors, then apply upper
customerdata['city'] = customerdata['city'].str.upper()
customerdata.head()
```

	customer	city
0	100 YARDS DEPARTMENTAL STORE	ANANTNAG
1	2R INTERNATIONALS	ZIRAKPUR
2	A N COMPONENTS	CHANDIGARH
3	A S THAKUR	SIRMOUR
4	A SQUARE MEDIA	AMRITSAR

```
# clean the city names
def clean_city_name(city):
    if pd.isna(city):
        return None
    city = str(city).strip().upper()
    city = re.sub(r'[^A-Z\s]', '', city)
    city = re.sub(r'\s+', ' ', city)
    return city.strip()

customerdata['city_clean'] = customerdata['city'].apply(clean_city_name)

city_to_state = {
    # Chandigarh, Mohali, Panchkula Tricity
    "CHANDIGARH": "Chandigarh",    "CHANDIHARH": "Chandigarh",    "CHANDGARH": "Chandigarh",    "SAS NAGAR": "Punjab",    "SAS NAGAR MOHALI": "Punjab",    "S.A.S NAGAR MOHALI": "Punjab",
    "S.A. NAGAR MOHALI": "Punjab",    "MOHALI": "Punjab",    "MHALI": "Punjab",    "MANIMAJRA": "Chandigarh",    "PANCHKULA": "Haryana",    "PANVHKULA": "Haryana",    "PANCHKULLA": "Haryana",    "ZIRAKPUR": "Punjab",    "D
    "DERABASSI": "Punjab",    "PINJORE": "Haryana",    "KURALI": "Punjab",    "KHARAR": "Punjab",    "SIRHIND": "Punjab",
    "AMBALA": "Haryana",    "AMBALA CANTT": "Haryana",    "AMBALA CITY": "Haryana",    "KURUKSHETRA": "Haryana",    "ROHTAK": "Haryana",    "SONIPAT": "Haryana",
    "HISAR": "Haryana",    "KAITHAL": "Haryana",    "UCHANA": "Haryana",    "FARIDABAD": "Haryana",    "YAMUNANAGAR": "Haryana",    "NARAINGARH": "Haryana",
    "GHARAUNDA": "Haryana",    "GHARONDA": "Haryana",
    # Punjab
    "LUDHIANA": "Punjab",    "JALANDHAR": "Punjab",    "JALANDHAR CITY": "Punjab",    "JALANDHER": "Punjab",    "AMRITSAR": "Punjab",    "PATIALA": "Punjab",
    "BATHINDA": "Punjab",    "BARNALA": "Punjab",    "FIROZPUR": "Punjab",    "FEROZEPUR": "Punjab",    "HOSHIARPUR": "Punjab",    "KHANNA": "Punjab",    "MALOUT": "Punjab",    "SANGRUR": "Punjab",    "SUNAM": "Punjab",
    # Himachal Pradesh
    "SHIMLA": "Himachal Pradesh",    "KULLU": "Himachal Pradesh",    "MANDI": "Himachal Pradesh",    "KANGRA": "Himachal Pradesh",    "CHAMBA": "Himachal Pradesh",    "SOLAN": "Himachal Pradesh",    "BILASPUR": "Himachal P
    # Delhi/NCR
    "DELHI": "Delhi",    "NEW DELHI": "Delhi",    "NOIDA": "Uttar Pradesh",    "GREATER NOIDA": "Uttar Pradesh",    "GURGAON": "Haryana",    "GURUGRAM": "Haryana",    "BALLIA": "Uttar Pradesh",    "GHAZIABAD": "Uttar Prade
    # Uttarakhand
    "DEHRADUN": "Uttarakhand",    "HARIDWAR": "Uttarakhand",    "CHAMPAWAT": "Uttarakhand",    "RUDRAPUR": "Uttarakhand",    "RUDERAPUR": "Uttarakhand",    "KASHIPUR": "Uttarakhand",
    # Rajasthan
    "JAIPUR": "Rajasthan",    "MANDSAUR": "Madhya Pradesh",    "BEAWAR": "Rajasthan",
    # Maharashtra
    "MUMBAI": "Maharashtra",    "PUNE": "Maharashtra",    "THANE": "Maharashtra",    "BHIWANDI": "Maharashtra",    "PANVEL": "Maharashtra",
    # Karnataka
    "BANGALORE": "Karnataka",    "BENGALURU": "Karnataka",    "BELAGAVI": "Karnataka",    "HUBLI": "Karnataka",
    # Tamil Nadu
    "CHENNAI": "Tamil Nadu",
    # Telangana
    "HYDERABAD": "Telangana",
    # West Bengal
    "KOLKATA": "West Bengal",
    # Odisha
    "BHUBANESHWAR": "Odisha",    "BHUBNESHWAR": "Odisha",
    # Assam
    "GUWAHATI": "Assam",    "TINSUKIA": "Assam",
    # Kerala
    "TRIVANDRUM": "Kerala",    "THIRUVANANTHAPURAM": "Kerala",
    # Gujarat
    "AHMEDABAD": "Gujarat",    "SURAT": "Gujarat",
    # Chhattisgarh
    "RAIPUR": "Chhattisgarh",
    # Ladakh / JK
    "LEH": "Ladakh",
    "JAMMU": "Jammu & Kashmir",    "ANANTNAG": "Jammu & Kashmir",    "BUDGAM": "Jammu & Kashmir",    "KATHUA": "Jammu & Kashmir",    "SRINAGAR": "Jammu & Kashmir",    "SAMBA": "Jammu & Kashmir",
    # UP
    "LUCKNOW": "Uttar Pradesh",    "GORAKHPUR": "Uttar Pradesh",    "VARANASI": "Uttar Pradesh",    "PRAYAGRAJ": "Uttar Pradesh",    "ALIGARH": "Uttar Pradesh",    "MUZAFFARNAGAR": "Uttar Pradesh",    "SAHARANPUR": "Uttar
    # MP
    "BHOPAL": "Madhya Pradesh",
    # Punjab clusters/infrequent
    "BARANALA": "Punjab",    "PATIALA": "Punjab",    "CHUNNI": "Punjab",    "PEHOWA": "Haryana",    "BHAWANIGARH": "Punjab",    "BANUR": "Punjab",    "TARNTARAN": "Punjab",    "TARANTARN": "Punjab",    "TALWARA": "Punjab",
    # Haryana clusters/infrequent
    "JIND": "Haryana",    "KARNAL": "Haryana",    "RAJOUND": "Haryana",    "PAI": "Haryana",    "PUNDRI": "Haryana",    "KALAYAT": "Haryana",    "SHAHABAD": "Haryana",    "GHARAUNDA": "Haryana",    "KURALI": "Punjab",
    # Misc
    "PANIPAT": "Haryana",    "BARNALA": "Punjab",    "BADDI": "Himachal Pradesh",    "KOTKAPURA": "Punjab",    "MASERAN": "Punjab",    "CHAMYARI": "Punjab",    "LALRU": "Punjab",
    "CO APO": "ARMY POST OFFICE",    "DHAKOLI": "Punjab",    "RAJPURA": "Punjab",    "PATRAN": "Punjab",    "DHAKOLI": "Punjab",    "MORADABAD": "Uttar Pradesh",
    "ABOHAR": "Punjab",    "SAS NAGARMOHALI": "Punjab",    "BARWALA": "Punjab",    "FATHEGARH SAHIB": "Punjab",    "TOHANA": "Haryana",    "MUSSOORIE": "Uttarakhand",
    "BAJPUR": "Uttarakhand",    "SA NAGAR MOHALI": "Punjab",    "RUPNAGAR": "Punjab",    "NAINA DEVI": "Himachal Pradesh",    "ROPAR": "Punjab",    "KALKA": "Haryana",    "MANDI GOBINDGARH": "Punjab",    "PANAJI": "Goa",
    "MAHILPUR": "Punjab",    "BANGALURU": "Karnataka",    "BHAGTA": "Punjab",    "BAHADURGARH": "Haryana",    "ZIRKPUR": "Punjab",    "MUKERIAN": "Punjab",    "DHARIWAL": "Punjab",
    "GHUMARWIN": "Himachal Pradesh",    "SIRSA": "Haryana",    "TALWANDI BHAI": "Punjab",    "CHANDIMANDIR CANTT": "Haryana",    "BALACHOUR": "Punjab",
    }

customerdata['state'] = customerdata['city_clean'].map(city_to_state)
unmapped = customerdata[customerdata['state'].isna()]['city_clean'].unique()
print("Unmapped cities:", unmapped)

customerdata.drop(columns=['city'], inplace=True)
customerdata.rename(columns={'city_clean': 'city'}, inplace=True)
customerdata.head()
```

```
Unmapped cities: [None]
```

	customer	city	state
0	100 YARDS DEPARTMENTAL STORE	ANANTNAG	Jammu & Kashmir
1	2R INTERNATIONALS	ZIRAKPUR	Punjab
2	A N COMPONENTS	CHANDIGARH	Chandigarh
3	A S THAKUR	SIRMOUR	Himachal Pradesh
4	A SQUARE MEDIA	AMRITSAR	Punjab

```
print(customerdata['state'].value_counts())
```

```
state
Punjab              679
Chandigarh          284
Haryana             178
Himachal Pradesh    151
Delhi                28
Uttar Pradesh        27
Uttarakhand          18
Maharashtra          16
Jammu & Kashmir      15
Rajasthan             8
Karnataka             7
Gujarat               4
Telangana             4
West Bengal           3
Ladakh                3
Odisha                2
Assam                 2
Tamil Nadu            2
Madhya Pradesh        2
Kerala                2
ARMY POST OFFICE      1
Goa                   1
Chhattisgarh          1
Name: count, dtype: int64
```

```
state_to_region = {
    "Punjab": "North",    "Chandigarh": "North",    "Haryana": "North",    "Himachal Pradesh": "North",    "Delhi": "North",    "Uttarakhand": "North",    "Jammu & Kashmir": "North",    "Ladakh": "North",    "Rajasthan": "
}
```

```
customerdata['region'] = customerdata['state'].map(state_to_region)
customerdata['region'] = customerdata['region'].fillna('Others')

```

```
customerdata
```

	customer	city	state	region
0	100 YARDS DEPARTMENTAL STORE	ANANTNAG	Jammu & Kashmir	North
1	2R INTERNATIONALS	ZIRAKPUR	Punjab	North
2	A N COMPONENTS	CHANDIGARH	Chandigarh	North
3	A S THAKUR	SIRMOUR	Himachal Pradesh	North
4	A SQUARE MEDIA	AMRITSAR	Punjab	North
...	...	...	...	...
1481	VINOD KUMAR	AMBALA	Haryana	North
1482	WAADE PAAJI-70	MOHALI	Punjab	North
1483	WAVE FOOD SOLUTION	AMRITSAR	Punjab	North
1484	YELLOW LIGHT STORE	AMRITSAR	Punjab	North
1485	ZEKTRA INTERNATIONAL PVT LIMITED	NEW DELHI	Delhi	North

1486 rows × 4 columns

```
Start coding or generate with AI.
```

ANALYSIS ON THE UPDATED DATA

```
itemdata
```

	date	bill_no	item	quantity	product_category
0	2020-05-07	1719	NS POS ASPIRE	1	POS TERMINAL
1	2020-05-07	1719	SUBSCRIPTION CHARGES - 1 YEAR	1	AMC
2	2020-05-11	1720	ELECTRONIC CASH REGISTER T-90	1	ECR
3	2020-05-14	1721	NS POS PRO	1	POS TERMINAL
4	2020-05-14	1721	SUBSCRIPTION CHARGES - 1 YEAR	1	AMC
...	...	...	...	...	...
7441	2025-07-28	251	GEAR PF20201-01F	15	SPARE PART
7442	2025-07-28	251	GEAR PF20202-10F	15	SPARE PART
7443	2025-07-28	251	GEAR PF20203-02F	15	SPARE PART
7444	2025-07-28	251	GEAR PF20204-01F	15	SPARE PART
7445	2025-07-28	252	SENSIBLE-10	1	POS TERMINAL

7446 rows × 5 columns

```
# for i in [billdata, itemdata, customerdata]:
#   print(i.columns)
```

```
Start coding or generate with AI.
```

```
print("Bills per payment mode:")
print(billdata['payment_mode'].value_counts())

print("\nTotal revenue by payment mode:")
print(billdata.groupby('payment_mode')['total_amount'].sum().sort_values(ascending=False))

print("\nSales by day of week:")
print(billdata.groupby('day_name')['total_amount'].sum().reindex(
    ['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday']))

```

```
Bills per payment mode:
payment_mode
Online    4356
Cash       148
Name: count, dtype: int64

Total revenue by payment mode:
payment_mode
Online   85,071,868
Cash        974,001
Name: total_amount, dtype: float64

Sales by day of week:
day_name
Monday      15,270,475
Tuesday     13,804,293
Wednesday   15,307,834
Thursday    13,973,490
Friday      12,812,528
Saturday    14,095,957
Sunday         781,292
Name: total_amount, dtype: float64
```

Bills per Payment Mode

- Online payments dominate with **4,356 bills**, compared to only **148 cash payments**.
- This shows a strong customer preference for digital transactions, reflecting wider adoption of online payment methods.

Total Revenue by Payment Mode

- Total sales from online payments are around **₹8.51 crore (₹85,07,18,680)**, vastly surpassing cash sales at just **₹9.74 lakh (₹9,74,001)**.
- Online payment modes contribute over **98.8% of revenue**, underscoring their critical role in business growth and cashless transaction trends.

Sales by Day of the Week

- Highest sales revenue happens on **Monday (₹1.53 crore)** and **Wednesday (₹1.53 crore)**, followed closely by **Saturday (₹1.41 crore)** and **Thursday (₹1.40 crore)**.
- Sales dip slightly on **Friday (₹1.28 crore)** and **Tuesday (₹1.38 crore)**.
- **Sunday shows lowest sales (₹7.81 lakh)**, indicating possibly fewer operational hours or lower customer footfall.

```
print("Total bills:", billdata['bill_no'].nunique())
print("Total sales revenue:", billdata['total_amount'].sum())
print("Average bill amount:", round(billdata['total_amount'].mean(),2))
print("Median bill amount:", billdata['total_amount'].median())

```

```
Total bills: 3498
Total sales revenue: 86045869.08
Average bill amount: 19104.32
Median bill amount: 12744.0
```

Sales Performance Summary

- A total of **3,498 bills** were generated, reflecting a steady transaction volume across the business.
- The overall sales revenue amounted to approximately **₹8.60 crore (₹86,04,58,869)**, indicating robust business scale.
- The **average bill value** stands at around **₹19,100**, showing that on average, customers spend a significant amount per transaction.
- The **median bill value** is **₹12,744**, highlighting that half of the bills are below this amount, suggesting some high-value bills elevate the average.

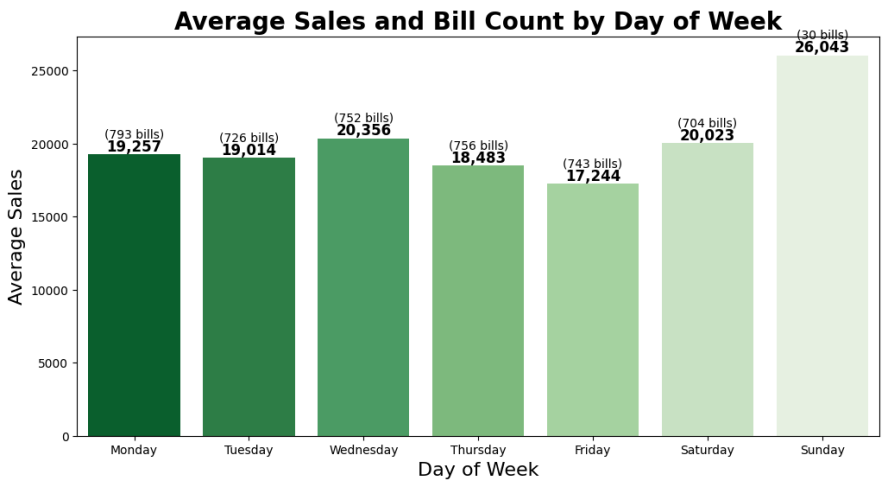
Business Implications

- The difference between the average and median bill values indicates a **right-skewed distribution**, with some transactions significantly higher than typical sales.
- Such insights help in identifying the presence of high-value customers or bulk order transactions influencing overall revenue.
- These KPIs provide a solid foundation for targeting marketing efforts, pricing strategies, and inventory planning designed to increase average transaction size without losing smaller customers.
- Monitoring trends in total bills and sales revenue over time will help assess growth and seasonal fluctuations critical for strategic decisions.

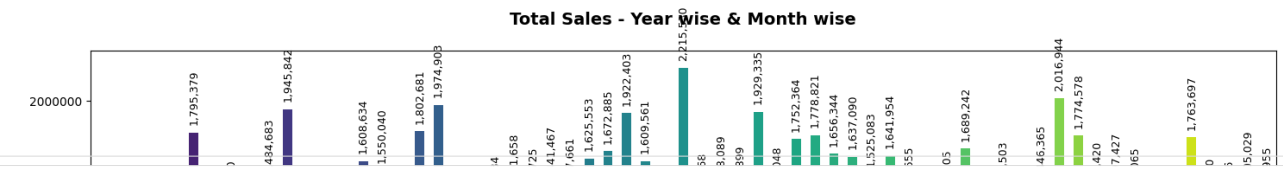
Show code

Show code

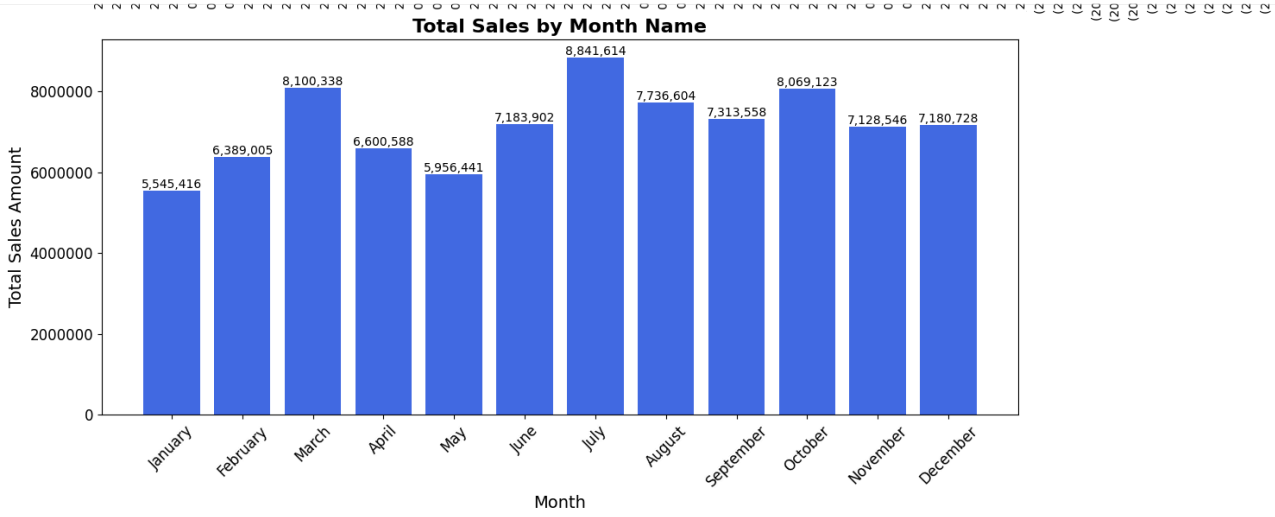
```
bill_counts_by_day = billdata['day_name'].value_counts().reindex(['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday'])

dow_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
dow_sales = billdata.groupby('day_name')['total_amount'].mean().reindex(dow_order)

plt.figure(figsize=(12,6))
ax = sns.barplot(x=dow_sales.index, y=dow_sales.values, palette='Greens_r')

plt.title('Average Sales and Bill Count by Day of Week', fontsize=20, fontweight="bold")
plt.xlabel('Day of Week', fontsize=16)
plt.ylabel('Average Sales', fontsize=16)

for p in ax.patches:
    height = p.get_height()
    ax.annotate(f'{height:,.0f}',
                (p.get_x() + p.get_width() / 2, height),
                ha='center', va='bottom', fontsize=12, fontweight='bold')

for i, day in enumerate(dow_order):
    count = bill_counts_by_day[day]
    plt.text(i, dow_sales[day] + 1000, f'({count} bills)', ha='center', va='bottom', fontsize=10)

plt.show()
```



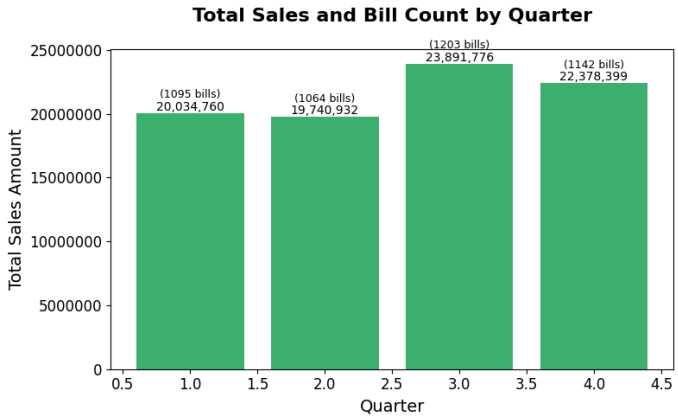

```
billdata['date'] = pd.to_datetime(billdata['date'])

monthly_sales = billdata.groupby(['year', 'month'])['total_amount'].sum()

plt.figure(figsize=(15,6))
ax = monthly_sales.plot(kind='bar', color=sns.color_palette('viridis', len(monthly_sales)))
plt.title('Total Sales - Year wise & Month wise\n', fontsize=14, fontweight='bold')
plt.xlabel("Year, Month", fontsize=12)
plt.ylabel("Total Revenue", fontsize=12)
plt.xticks(rotation=90, fontsize=10)
plt.yticks(fontsize=10)

for bar in ax.patches:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 50000, f'{yval:,.0f}', ha='center', va='bottom', rotation=90, fontsize=9)


plt.tight_layout()
plt.show()
```



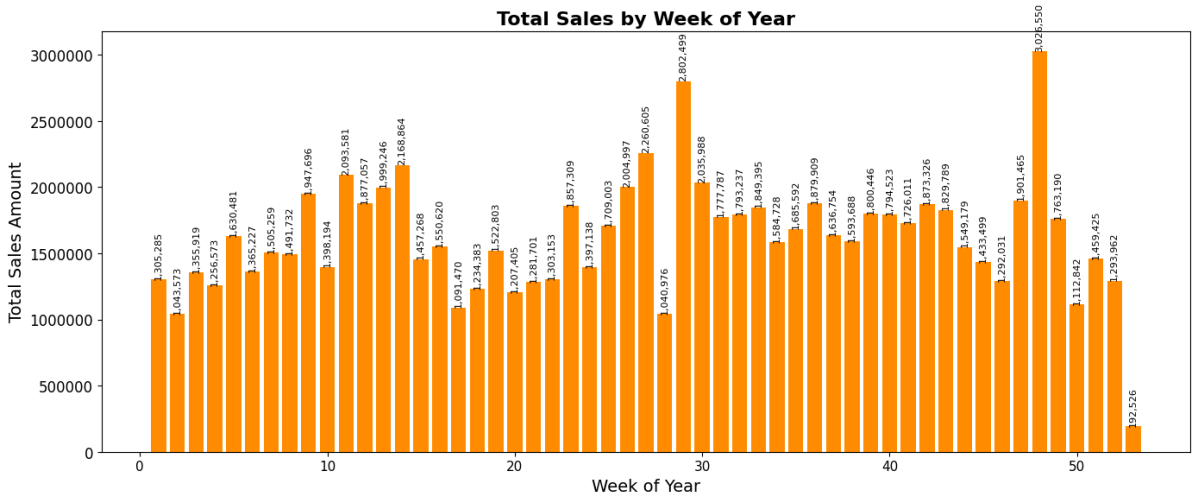

```
plt.figure(figsize=(12,6))
month_order = ['January','February','March','April','May','June','July','August','September','October','November','December']
month_sales = billdata.groupby('month_name')['total_amount'].sum().reindex(month_order)

bars = plt.bar(month_sales.index, month_sales.values, color='royalblue')
plt.title('Total Sales by Month Name', fontweight='bold', fontsize=16)
plt.xlabel('Month', fontsize=14)
plt.ylabel('Total Sales Amount', fontsize=14)
plt.xticks(rotation=45, fontsize=12)
plt.yticks(fontsize=12)

for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, height, f'{int(height):,}', ha='center', va='bottom', fontsize=10)

plt.tight_layout()
plt.show()
```



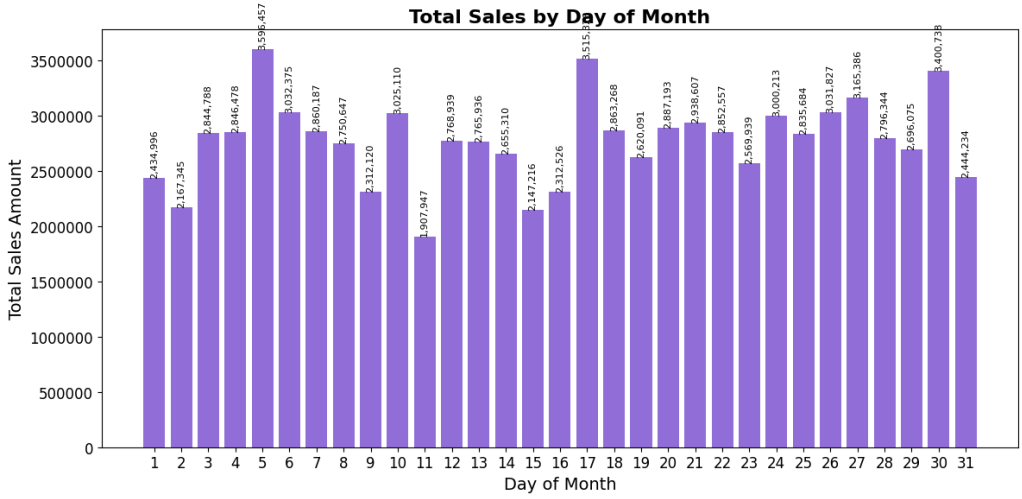

```
plt.figure(figsize=(8,5))
quarter_sales = billdata.groupby('quarter')['total_amount'].sum().sort_index()
quarterly_bill_counts = billdata['quarter'].value_counts().sort_index()

bars = plt.bar(quarter_sales.index, quarter_sales.values, color='mediumseagreen')
plt.title('Total Sales and Bill Count by Quarter\n', fontweight='bold', fontsize=16)
plt.xlabel('Quarter', fontsize=14)
plt.ylabel('Total Sales Amount', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

for i, bar in enumerate(bars):
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, height, f'{int(height):,}', ha='center', va='bottom', fontsize=10)
    count = quarterly_bill_counts.iloc[i]
    plt.text(bar.get_x() + bar.get_width()/2, height + 1000000, f'({count} bills)', ha='center', va='bottom', fontsize=9) # Adjusted vertical position

plt.tight_layout()
plt.show()
```




```
plt.figure(figsize=(14,6))
week_sales = billdata.groupby('week_of_year')['total_amount'].sum()

bars = plt.bar(week_sales.index, week_sales.values, color='darkorange')
plt.title('Total Sales by Week of Year', fontweight='bold', fontsize=16)
plt.xlabel('Week of Year', fontsize=14)
plt.ylabel('Total Sales Amount', fontsize=14)
plt.xticks(fontsize=11)
plt.yticks(fontsize=12)

for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, height, f'{int(height):,}', ha='center', va='bottom', fontsize=8, rotation=90)

plt.tight_layout()
plt.show()

```



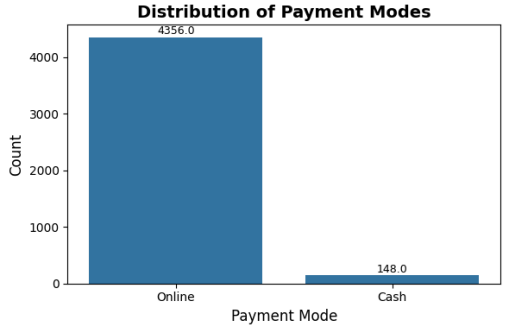

```
plt.figure(figsize=(12,6))
day_sales = billdata.groupby('day')['total_amount'].sum()

bars = plt.bar(day_sales.index, day_sales.values, color='mediumpurple')
plt.title('Total Sales by Day of Month', fontweight='bold', fontsize=16)
plt.xlabel('Day of Month', fontsize=14)
plt.ylabel('Total Sales Amount', fontsize=14)
plt.xticks(range(1,32), fontsize=12)
plt.yticks(fontsize=12)

for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, height, f'{int(height):,}', ha='center', va='bottom', fontsize=8, rotation=90)

plt.tight_layout()
plt.show()

```



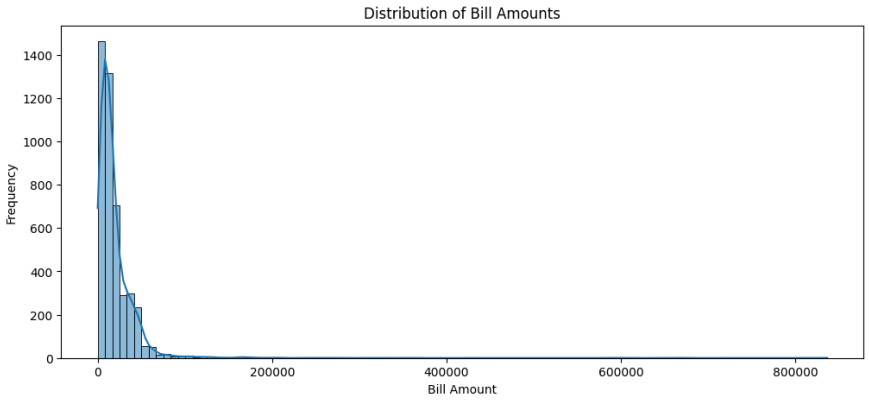

```
Start coding or generate with AI.
```

Double-click (or enter) to edit

```
plt.figure(figsize=(6,4))
ax = sns.countplot(x='payment_mode', data=billdata)
plt.title('Distribution of Payment Modes', fontsize=14, fontweight='bold')
plt.xlabel('Payment Mode', fontsize=12)
plt.ylabel('Count', fontsize=12)

# Add counts above the bars
for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', xytext=(0, 5), textcoords='offset points', fontsize=9)

plt.tight_layout()
plt.show()
```



```
plt.figure(figsize=(12,5))
sns.histplot(billdata['total_amount'], bins=100, kde=True)
plt.title('Distribution of Bill Amounts')
plt.xlabel('Bill Amount')
plt.ylabel('Frequency')
plt.show()

```




```
q_low = billdata['total_amount'].quantile(0.05)
q_high = billdata['total_amount'].quantile(0.99)
outliers = billdata[(billdata['total_amount'] < q_low) | (billdata['total_amount'] > q_high)]
print("Outlier bills (by total_amount):\n", outliers[['bill_no','total_amount']])

```

```
Outlier bills (by total_amount):
       bill_no  total_amount
19       1738       116,168
22       1741         1,180
89       1808         1,169
121      1840         1,770
144      1863         1,000
...       ...           ...
4446      195         1,416
4447      196           236
4450      199       121,000
4464      213             1
4469      218         1,180

[271 rows x 2 columns]
```

```
Q1 = billdata['total_amount'].quantile(0.25)
Q3 = billdata['total_amount'].quantile(0.75)

IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

outliers_iqr = billdata[(billdata['total_amount'] < lower_bound) | (billdata['total_amount'] > upper_bound)]

print("Outliers found using IQR method:")
display(outliers_iqr)

print(f"\nSummary of Outlier Detection using IQR:")
print(f"Q1 (25th percentile): {Q1:,.2f}")
print(f"Q3 (75th percentile): {Q3:,.2f}")
print(f"IQR (Q3 - Q1): {IQR:,.2f}")
print(f"Lower Bound (Q1 - 1.5 * IQR): {lower_bound:,.2f}")
print(f"Upper Bound (Q3 + 1.5 * IQR): {upper_bound:,.2f}")
print(f"Number of outliers detected: {outliers_iqr.shape[0]}")
```

```
Outliers found using IQR method:
```

	date	bill_no	customer	total_amount	cash_online	payment_mode	year	month	day	month_name	day_of_week	day_name	quarter	week_of_year	customer_type
6	2020-05-22	1725	EIDHO UTILITY STORE	77,970	2	Online	2020	5	22	May	4	Friday	2	21	Casual
19	2020-06-06	1738	NEEDS OFFICE SYSTEMS	116,168	2	Online	2020	6	6	June	5	Saturday	2	23	Loyal
85	2020-07-15	1804	SAMRIDHI ENTERPRISES	89,500	2	Online	2020	7	15	July	2	Wednesday	3	29	Casual
88	2020-07-16	1807	NEEDS OFFICE SYSTEMS	54,770	2	Online	2020	7	16	July	3	Thursday	3	29	Loyal
96	2020-07-17	1815	I TECH ENTERPRISES,	49,860	2	Online	2020	7	17	July	4	Friday	3	29	Casual
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4450	2025-07-05	199	R K TRADING COMPANY	121,000	2	Online	2025	7	5	July	5	Saturday	3	27	Casual
4454	2025-07-06	203	SMART STORE	80,000	2	Online	2025	7	6	July	6	Sunday	3	27	Casual
4475	2025-07-16	224	KHUSHDEEP GARG	49,000	2	Online	2025	7	16	July	2	Wednesday	3	29	Casual
4483	2025-07-21	232	R P TRADING COMPANY	55,106	2	Online	2025	7	21	July	0	Monday	3	30	Casual
4493	2025-07-23	242	PARDHAN DHABA	58,000	2	Online	2025	7	23	July	2	Wednesday	3	30	Casual

266 rows × 15 columns

```
Summary of Outlier Detection using IQR:
Q1 (25th percentile): 7,080.00
Q3 (75th percentile): 23,423.00
IQR (Q3 - Q1): 16,343.00
Lower Bound (Q1 - 1.5 * IQR): -17,434.50
Upper Bound (Q3 + 1.5 * IQR): 47,937.50
Number of outliers detected: 266
```

Outlier Detection Summary Using IQR

- The **25th percentile (Q1)** of bill amounts is ₹7,080 while the **75th percentile (Q3)** is ₹23,423, making the **IQR** (Q3 - Q1) ₹16,343.
- Using Tukey's method, the **lower bound** is calculated as -₹17,434.50 (not meaningful here since bills can't be negative), and the **upper bound** is ₹47,937.50.
- A total of **266 outlier bills** were detected, i.e., bills with values exceeding ₹47,937.50.

Business Insights

- These outliers represent **high-value bills well beyond the typical spending range**.
- For this business, it is crucial to **retain these outliers** instead of removing them because order values naturally vary widely—from as low as ₹500 to even ₹10 lakhs or more—depending on the scale and nature of customer businesses.
- Ignoring or removing these high-value outliers would lead to **loss of critical information about bulk or corporate sales** that significantly impact overall revenue.
- Including outliers ensures that your sales analysis reflects true business complexity and supports better decision-making for inventory, pricing, and customer segmentation.

Conclusion

The IQR method is an effective approach to identify outliers, but in this multi-scale business setting, these outliers are **valid and essential**, not errors or anomalous noise. Analytical models and reports should therefore incorporate them to grasp the real sales dynamics and revenue distribution.

```
Start coding or generate with AI.
```

```
print("Top 10 items (sales quantity):")
print(itemdata.groupby('item')['quantity'].sum().sort_values(ascending=False).head(10))

print("\nTop 5 categories (sales quantity):")
print(itemdata.groupby('product_category')['quantity'].sum().sort_values(ascending=False).head(5))

```

```
Top 10 items (sales quantity):
item
THERMAL PAPER ROLL-2            5111
THERMAL PAPER ROLL 3"           3626
PRINTER GEARS SET               3426
GEAR PF20203-02F                2109
GEAR PF20202-10F                2109
GEAR PF20204-01F                2109
GEAR PF20201-01F                2109
THERMAL PRINTER MECH. 3 INCH    1622
THERMAL PRINTER MECH. 2 INCH    1040
THERMAL PAPER ROLL-             1025
Name: quantity, dtype: int64

Top 5 categories (sales quantity):
product_category
SPARE PART         14032
CONSUMABLE         10071
THERMAL PRINTER     2962
ECR                 2683
CASH DRAWER          650
Name: quantity, dtype: int64
```

Top Selling Items by Quantity

- **Thermal Paper Rolls** are the highest in quantity sold, with **over 5,111 units** of the 2-inch variant and **3,626 units** of the 3-inch variant, revealing strong recurring demand likely due to consumable nature and frequent replacement needs.
- Several **gear parts** (e.g., "PRINTER GEARS SET" and various "GEAR PF" models) rank highly, indicating steady demand for spare and replacement components essential for maintenance.
- **Thermal Printer Mechanisms** also appear as significant contributors, reflecting sales of hardware parts that support thermal printers widely used in POS setups.

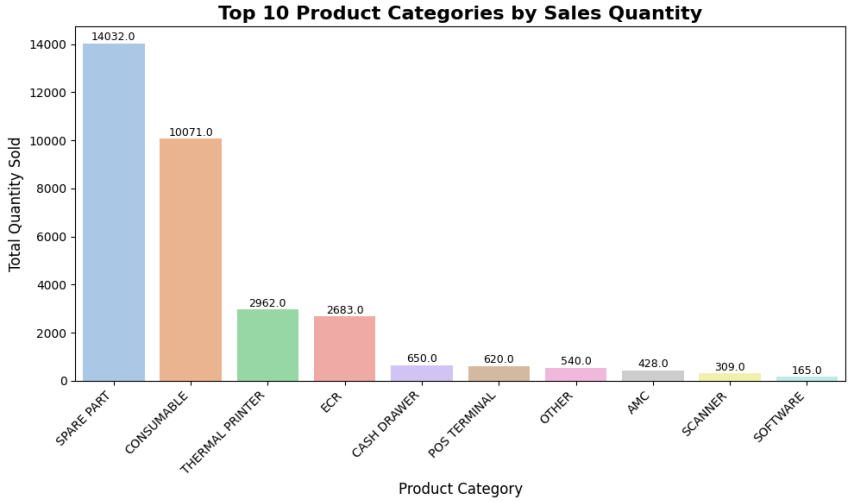
Top Categories by Quantity Sold

- **Spare Parts dominate the sales quantity** at over **14,000 units**, emphasizing the importance of maintenance and repair parts in ongoing customer needs.
- **Consumables (10,071 units)** such as paper rolls align with this trend, highlighting the consumable nature of key retail supplies.
- **Thermal Printers (2,962 units)** and **ECR devices (2,683 units)** show substantial sales, reinforcing the importance of core POS hardware alongside ancillary parts.
- **Cash Drawers (650 units)** contribute modestly, but remain relevant as critical POS components.

Business Insights & Implications

- High quantities of consumables and spare parts indicate a **stable recurring revenue stream** from maintenance and replacements, critical for sustained business operations beyond initial hardware sales.
- The mix of top items and categories reflects a **balanced portfolio** serving both first-time hardware buyers and existing customers needing parts, a healthy sign for customer retention and upselling.
- Ensuring optimal inventory levels for both consumables and spare parts will be vital to avoid stockouts that can disrupt customer operations.
- Marketing and sales strategies could focus on **bundled offers** combining core POS hardware (ECR, thermal printers) with accessories and consumables to improve Average Transaction Value (ATV).

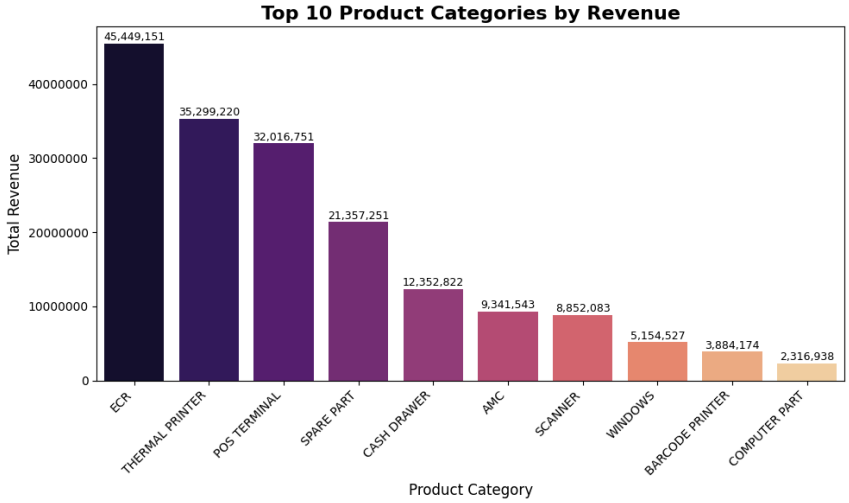
```
plt.figure(figsize=(10, 6))
top_10_categories = itemdata.groupby('product_category')['quantity'].sum().sort_values(ascending=False).head(10)
ax = sns.barplot(x=top_10_categories.index, y=top_10_categories.values, palette='pastel')
plt.title('Top 10 Product Categories by Sales Quantity', fontsize=16, fontweight='bold')
plt.xlabel('Product Category', fontsize=12)
plt.ylabel('Total Quantity Sold', fontsize=12)
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.yticks(fontsize=10)

for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', xytext=(0, 5), textcoords='offset points', fontsize=9)

plt.tight_layout()
plt.show()
```



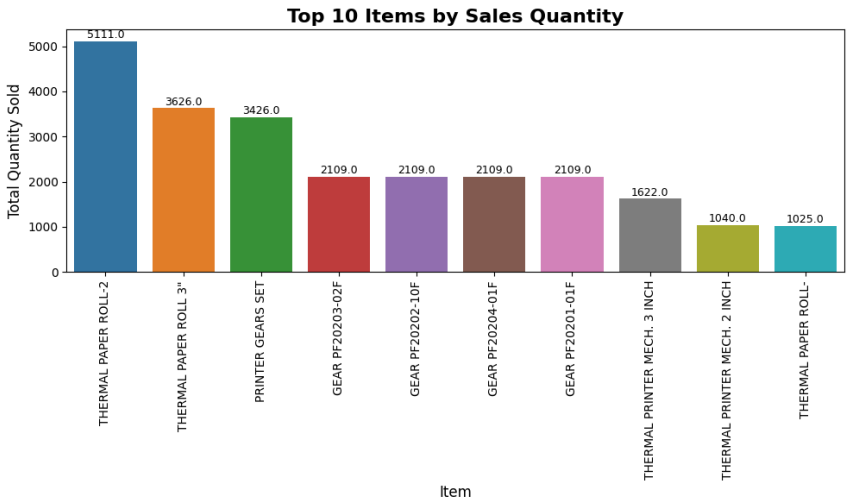

```
merged_data = pd.merge(billdata, itemdata, on=['date', 'bill_no'], how='inner')
display(merged_data.head(1))
```

	date	bill_no	customer	total_amount	cash_online	payment_mode	year	month	day	month_name	day_of_week	day_name	quarter	week_of_year	customer_type	item	quantity	product_category
0	2020-05-07	1719	SMART BUY DEPARTMENTAL STORE	32,000	2	Online	2020	5	7	May	3	Thursday	2	19	Casual	NS POS ASPIRE	1	POS TERMINAL

```
category_revenue = merged_data.groupby('product_category')['total_amount'].sum().sort_values(ascending=False)

top_10_categories_revenue = category_revenue.head(10)

plt.figure(figsize=(10, 6))
ax = sns.barplot(x=top_10_categories_revenue.index, y=top_10_categories_revenue.values, palette='magma')
plt.title('Top 10 Product Categories by Revenue', fontsize=16, fontweight='bold')
plt.xlabel('Product Category', fontsize=12)
plt.ylabel('Total Revenue', fontsize=12)
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.yticks(fontsize=10)

for p in ax.patches:
    ax.annotate(f'{p.get_height():,.0f}', (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', xytext=(0, 5), textcoords='offset points', fontsize=9)

plt.tight_layout()
plt.show()
```

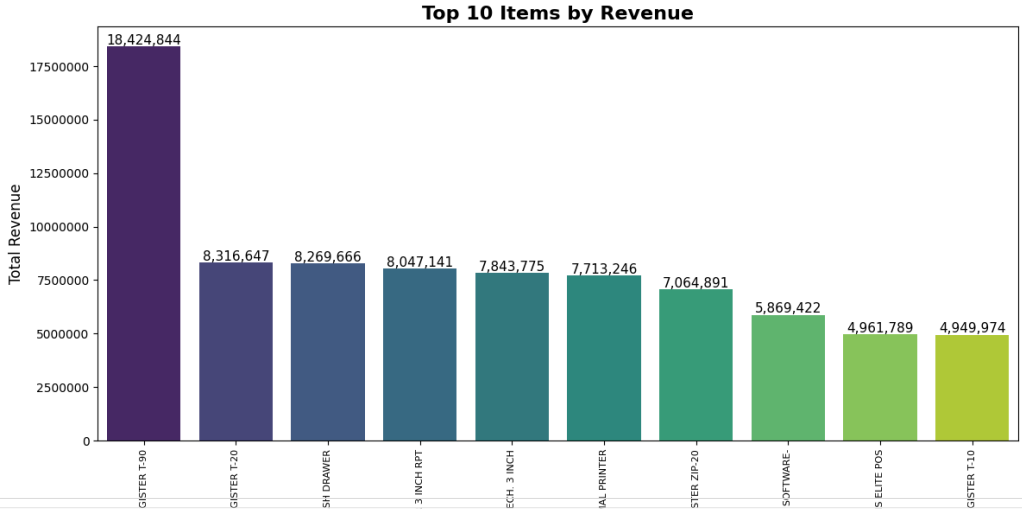



```
plt.figure(figsize=(10, 6))
top_10_items = itemdata.groupby('item')['quantity'].sum().sort_values(ascending=False).head(10)
ax = sns.barplot(x=top_10_items.index, y=top_10_items.values, palette='tab10')
plt.title('Top 10 Items by Sales Quantity', fontsize=16, fontweight='bold')
plt.xlabel('Item', fontsize=12)
plt.ylabel('Total Quantity Sold', fontsize=12)
plt.xticks(rotation=90, fontsize=10)
plt.yticks(fontsize=10)

for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', xytext=(0, 5), textcoords='offset points', fontsize=9)

plt.tight_layout()
plt.show()
```

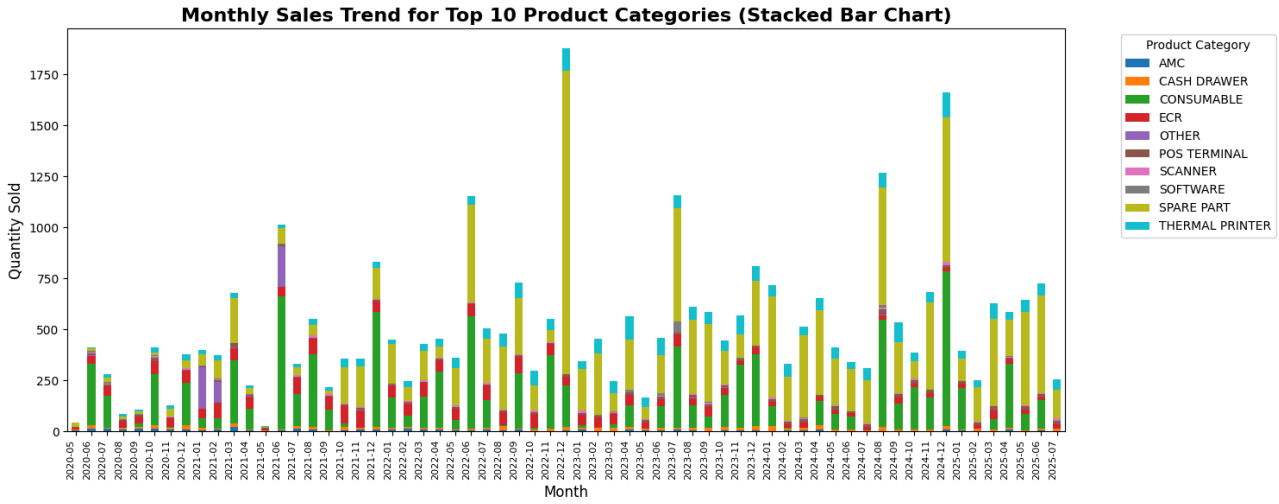



```
Start coding or generate with AI.
```

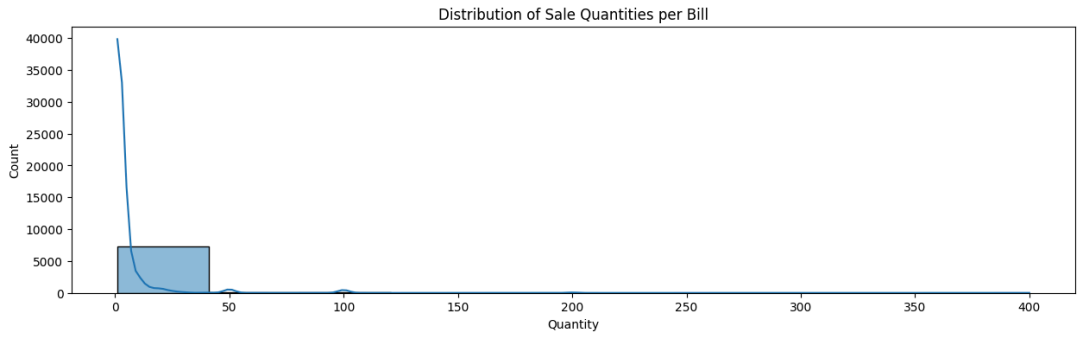
```
item_revenue = merged_data.groupby('item')['total_amount'].sum().sort_values(ascending=False)
top_10_items_revenue = item_revenue.head(10)

plt.figure(figsize=(12, 8))
ax = sns.barplot(x=top_10_items_revenue.index, y=top_10_items_revenue.values, palette='viridis')
plt.title('Top 10 Items by Revenue', fontsize=16, fontweight='bold')
plt.xlabel('Item', fontsize=12)
plt.ylabel('Total Revenue', fontsize=12)
plt.xticks(rotation=90, fontsize=8)
plt.yticks(fontsize=10)

for p in ax.patches:
    ax.annotate(f'{p.get_height():,.0f}', (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', xytext=(0, 5), textcoords='offset points', fontsize=11)

plt.tight_layout()
plt.show()
```




```
itemdata['date'] = pd.to_datetime(itemdata['date'])
itemdata['month'] = itemdata['date'].dt.to_period('M')

top_categories = itemdata.groupby('product_category')['quantity'].sum().sort_values(ascending=False).head(10).index.tolist()

cat_trends_top = itemdata[itemdata['product_category'].isin(top_categories)]

cat_trends_pivot = cat_trends_top.groupby(['month', 'product_category'])['quantity'].sum().unstack().fillna(0)

cat_trends_pivot.plot(kind='bar', stacked=True, figsize=(15, 6))
plt.title('Monthly Sales Trend for Top 10 Product Categories (Stacked Bar Chart)', fontsize=16, fontweight='bold')
plt.xlabel("Month", fontsize=12)
plt.ylabel("Quantity Sold", fontsize=12)
plt.xticks(rotation=90, ha='right', size=8)
plt.legend(title='Product Category', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(axis='y', linestyle='', alpha=0.2) # Reduced grid
plt.tight_layout()
plt.show()
```




```
plt.figure(figsize=(15,4))
sns.histplot(itemdata['quantity'], bins=10, kde=True)
plt.title('Distribution of Sale Quantities per Bill')
plt.xlabel('Quantity')
plt.show()

```




```
top_per_category = (itemdata.groupby(['product_category','item'])['quantity']
                    .sum()
                    .groupby(level=0, group_keys=False)
                    .nlargest(5)
                    .reset_index())

categories = top_per_category['product_category'].unique()
n_categories = len(categories)

n_cols = 4
n_rows = 6

plt.figure(figsize=(n_cols * 5, n_rows * 3))

for i, category in enumerate(categories):
    plt.subplot(n_rows, n_cols, i + 1)
    category_data = top_per_category[top_per_category['product_category'] == category]
    ax = sns.barplot(x='quantity', y='item', data=category_data, palette='viridis')
    plt.title(f'Top 5 Items in {category}', fontsize=12
```

```
    plt.xlabel('Quantity Sold', fontsize=9)
    plt.ylabel('Item', fontsize=9)
    plt.xticks(fontsize=7)
    plt.yticks(fontsize=7)
    plt.grid(axis='x', linestyle='--', alpha=0.6)

    for p in ax.patches:
        width = p.get_width()
        plt.text(width, p.get_y() + p.get_height()/2, f'{int(width):,}',
                 ha='left', va='center', fontsize=7)

plt.tight_layout()
plt.show()
```

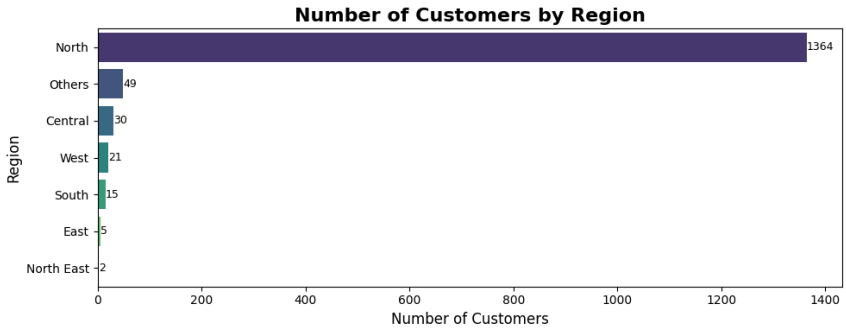

```
1 Start coding or generate with AI.
```

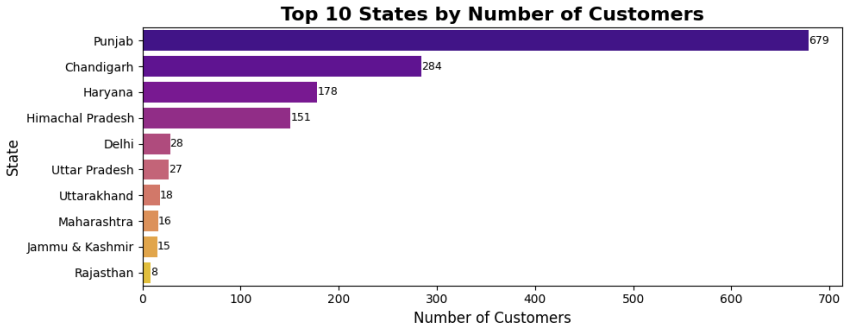
CUSTOMER ANALYSIS

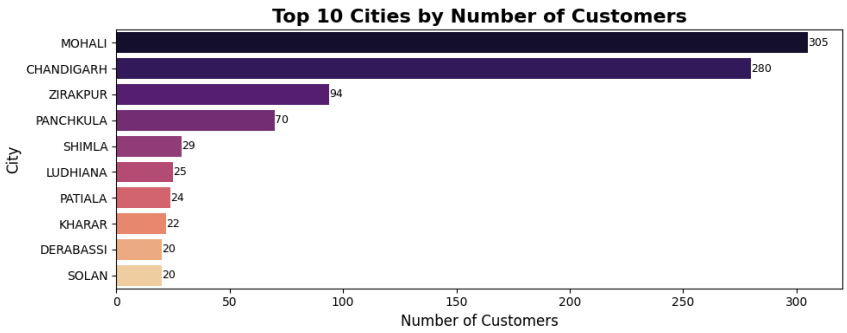
Customer Demographics

```
plt.figure(figsize=(10, 4))
ax1 = sns.countplot(y='region', data=customerdata, order=customerdata['region'].value_counts().index, palette='viridis')
plt.title('Number of Customers by Region', fontsize=16, fontweight='bold')
plt.xlabel('Number of Customers', fontsize=12)
plt.ylabel('Region', fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)

for p in ax1.patches:
    plt.text(p.get_width(), p.get_y() + p.get_height()/2, f'{int(p.get_width())}',
             ha='left', va='center', fontsize=9)

plt.tight_layout()
plt.show()

plt.figure(figsize=(10, 4))
ax2 = sns.countplot(y='state', data=customerdata, order=customerdata['state'].value_counts().head(10).index, palette='plasma')
plt.title('Top 10 States by Number of Customers', fontsize=16, fontweight='bold')
plt.xlabel('Number of Customers', fontsize=12)
plt.ylabel('State', fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)

for p in ax2.patches:
    plt.text(p.get_width(), p.get_y() + p.get_height()/2, f'{int(p.get_width())}',
             ha='left', va='center', fontsize=9)
```

```
plt.tight_layout()
plt.show()

plt.figure(figsize=(10, 4))
ax3 = sns.countplot(y='city', data=customerdata, order=customerdata['city'].value_counts().head(10).index, palette='magma')
plt.title('Top 10 Cities by Number of Customers', fontsize=16, fontweight='bold')
plt.xlabel('Number of Customers', fontsize=12)
plt.ylabel('City', fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)

for p in ax3.patches:
    plt.text(p.get_width(), p.get_y() + p.get_height()/2, f'{int(p.get_width())}',
             ha='left', va='center', fontsize=9)

plt.tight_layout()
plt.show()
```







Top Customers by Revenue

- The top 10 customers generate significant revenue, with **SARGUN ENTERPRISES** leading at approximately **₹60 lakh**, followed closely by **VICTORY COMMUNICATION** (₹55.5 lakh) and **BIOMATIC OFFICE SOLUTIONS** (₹48.3 lakh).
- The revenue contribution from these customers ranges between about **₹10 lakh to ₹60 lakh**, collectively accounting for a substantial portion of total sales.

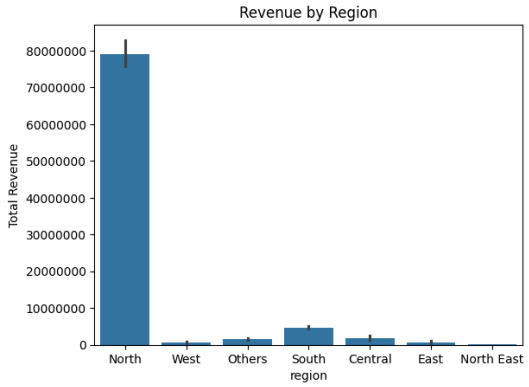
Business Implications

- These top customers are predominantly **dealers, agents, or resellers**, which explains their high purchase volumes and bulk buying behavior.
- This concentration indicates a **B2B-driven revenue model**, where a few key partners or intermediaries drive a large share of overall sales.
- Maintaining strong relationships with these dealers and resellers is crucial, as their bulk orders significantly impact business performance and stock planning.
- Targeted dealer-specific promotions, volume discounts, and customized services can help strengthen loyalty and sustain high sales.
- Monitoring the health and buying patterns of these top customers helps preempt supply chain risks and identify growth or retention opportunities.

```
Start coding or generate with AI.
```

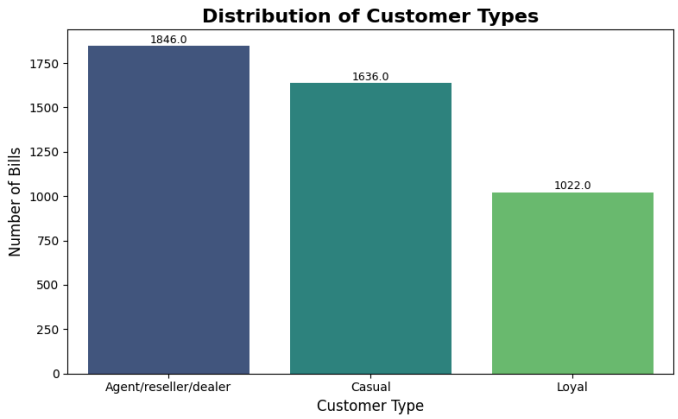
Regional Sales

```
region_sales = billdata.merge(customerdata[['customer','region']], on='customer')
sns.barplot(x='region', y='total_amount', data=region_sales, estimator=sum, order=region_sales['region'].unique())
plt.title('Revenue by Region')
plt.ylabel('Total Revenue')
plt.xticks(size=10)
plt.show()

```



Cohort & Repeat Customer Analysis

```
plt.figure(figsize=(8, 5))
ax = sns.countplot(x='customer_type', data=billdata, order=billdata['customer_type'].value_counts().index, palette='viridis')
plt.title('Distribution of Customer Types', fontsize=16, fontweight='bold')
plt.xlabel('Customer Type', fontsize=12)
plt.ylabel('Number of Bills', fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)

for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', xytext=(0, 5), textcoords='offset points', fontsize=9)

plt.tight_layout()
plt.show()

customer_type_revenue = billdata.groupby('customer_type')['total_amount'].sum().sort_values(ascending=False)

print("\nTotal Revenue by Customer Type:")
print(customer_type_revenue)

customer_type_avg_revenue = billdata.groupby('customer_type')['total_amount'].mean().sort_values(ascending=False)

print("\nAverage Revenue per Bill by Customer Type:")
print(customer_type_avg_revenue)
```




```
Total Revenue by Customer Type:
customer_type
Casual                  37,233,477
Agent/reseller/dealer   32,589,751
Loyal                   16,222,641
Name: total_amount, dtype: float64

Average Revenue per Bill by Customer Type:
customer_type
Casual                  22,759
Agent/reseller/dealer   17,654
Loyal                   15,873
Name: total_amount, dtype: float64
```

Revenue Contribution by Customer Type

- **Casual customers** generate the highest total revenue at approximately **₹3.72 crore (₹37,23,34,770)**, reflecting a large base of occasional buyers with smaller order frequency.
- The **Agent/Reseller/Dealer segment** contributes nearly **₹3.26 crore (₹32,58,97,510)** in total revenue, highlighting their critical role as bulk purchasers driving a significant share of business.
- The **Loyal customers**, while fewer in number, still contribute over **₹1.62 crore (₹16,22,26,410)**, signifying strong recurring revenue from repeat buyers.

Average Revenue Per Bill by Customer Type

- The **Casual segment** has the highest average bill value at around **₹22,759**, indicating that even occasional buyers tend to spend relatively large amounts per transaction.
- **Agent/Reseller/Dealer customers** average about **₹17,654** per bill, reflecting consistent bulk ordering behavior spread across fewer transactions.
- **Loyal customers** have a slightly lower average bill value of approximately **₹15,873**, possibly due to more frequent but smaller purchases.

Business Insights and Recommendations

- The business benefits from maintaining a **diverse customer mix** across casual, reseller, and loyal segments, each driving revenue differently.
- Casual customers offer higher-value one-off transactions but may require targeted marketing to improve retention.
- Agents and resellers form a **strategic sales channel** providing bulk volume; customized incentives or loyalty programs for this group can support sustained sales.
- Loyal customers provide stable, repeat business and are ideal targets for upselling and cross-selling initiatives.
- Monitoring these metrics regularly helps optimize customer engagement strategies and resource allocation to maximize sales growth and profitability.

```
Start coding or generate with AI.
```

```python
# Filter billdata for Agent/reseller/dealer customer type
agent_reseller_dealers = billdata[billdata['customer_type'] == 'Agent/reseller/dealer']

# Group by customer and calculate total revenue
top_agents_revenue = agent_reseller_dealers.groupby('customer')['total_amount'].sum().sort_values(ascending=False)

# Display the top 10
print("Top 10 Agent/Reseller/Dealer Customers by Revenue:")
print(top_agents_revenue.head(10))

top_10_agents_revenue = top_agents_revenue.head(10)
```

```
Top 10 Agent/Reseller/Dealer Customers by Revenue:
customer
SARGUN ENTERPRISES             6,005,414
VICTORY COMMUNICATION          5,550,024
BIOMATIC OFFICE SOLUTIONS      4,829,545
INTOWN AUTOMATION              4,494,893
SRAS INDIA PRIVATE LIMITED     1,602,263
G P ENTERPRISES                1,462,317
B.S.ELECTRONIC INSTRUMENTS     1,177,939
INFOACCORDS TECHNOLOGIES CHD   1,139,290
P.S.ENTERPRISES                1,008,538
CASH A/C                         971,405
Name: total_amount, dtype: float64
```

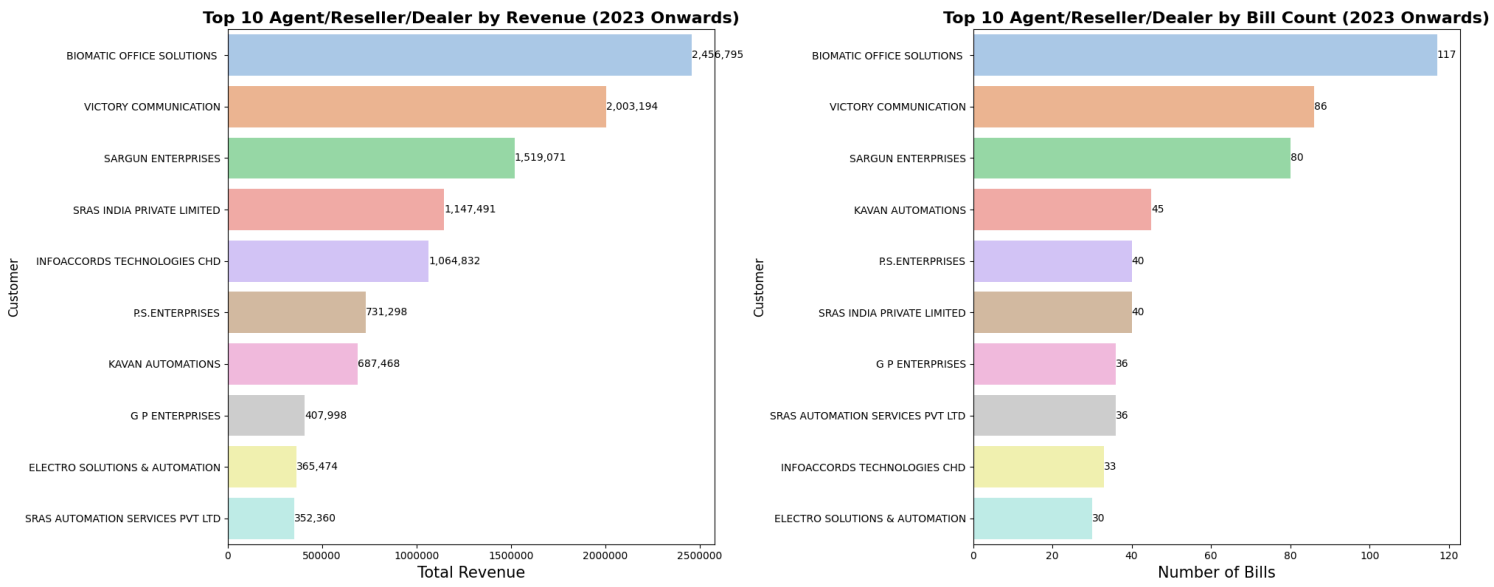
Top Agent/Reseller/Dealer Customers by Revenue

- The top dealers contribute significantly to your sales, with **SARGUN ENTERPRISES** leading at roughly **₹60 lakh**, followed by **VICTORY COMMUNICATION** (~~₹55.5 lakh~~) and ~~**BIOMATIC OFFICE SOLUTIONS** (~~₹48.3 lakh).
- These customers, mostly dealers, agents, or resellers, drive a large share of your business volume through bulk purchases.
- The presence of "CASH A/C" in the top 10 suggests some cash transactions or miscellaneous accounts contributing nearly ₹9.7 lakh, which is also significant.

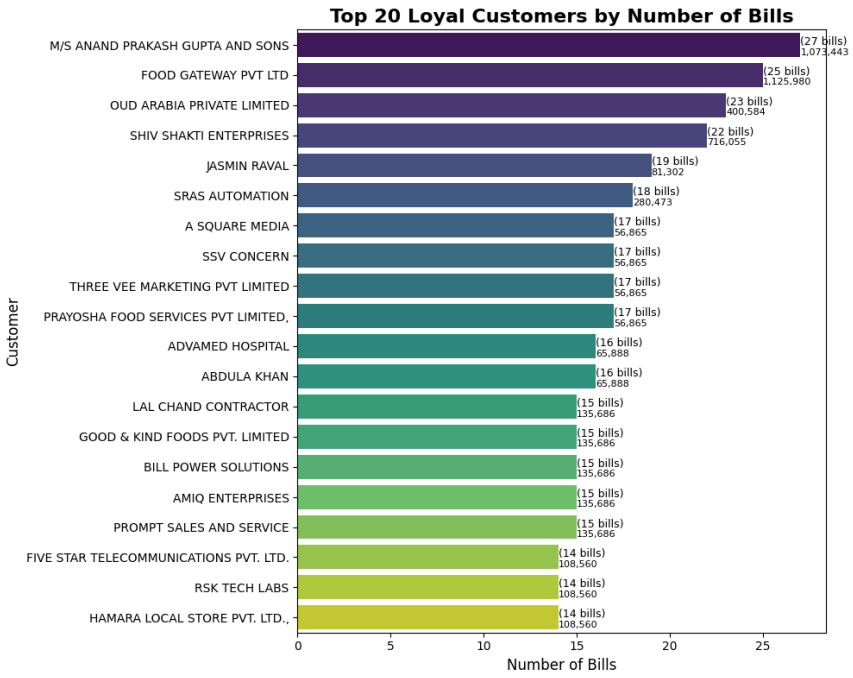
Business Implications

- Dealer/reseller relationships form a **critical channel for revenue growth**, underlining the importance of maintaining strong partnerships.
- Tailored incentives like volume discounts, flexible credit terms, and timely support can help strengthen loyalty and encourage larger orders.
- Monitoring purchase patterns of these top accounts enables proactive inventory management to sustain supply and avoid stockouts.
- Strategic account management focused on these key partners can maximize business impact and reduce risks associated with dependency on few customers.

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

# filter billdata for Agent/reseller/dealer customer type and data from 2023 onwards
agent_reseller_dealers_2023_onwards = billdata[(billdata['customer_type'] == 'Agent/reseller/dealer') & (billdata['date'].dt.year >= 2023)].copy()

agent_reseller_dealers_summary_2023_onwards = agent_reseller_dealers_2023_onwards.groupby('customer').agg(
    total_revenue=('total_amount', 'sum'),
    bill_count=('bill_no', 'nunique')
).reset_index()

agent_reseller_dealers_2023_onwards = billdata[(billdata['customer_type'] == 'Agent/reseller/dealer') & (billdata['date'].dt.year >= 2023)].copy()
top_agents_revenue_2023_onwards = agent_reseller_dealers_2023_onwards.groupby('customer')['total_amount'].sum().sort_values(ascending=False).head(10)

top_10_agents_bill_count_2023_onwards = agent_reseller_dealers_summary_2023_onwards.sort_values(by='bill_count', ascending=False).head(10)

fig, axes = plt.subplots(1, 2, figsize=(20, 8))

sns.barplot(x=top_agents_revenue_2023_onwards.values, y=top_agents_revenue_2023_onwards.index, palette='pastel', ax=axes[0])
axes[0].set_title('Top 10 Agent/Reseller/Dealer by Revenue (2023 Onwards)', fontsize=16, fontweight='bold')
axes[0].set_xlabel('Total Revenue', fontsize=15)
axes[0].set_ylabel('Customer', fontsize=12)
axes[0].tick_params(axis='both', which='major', labelsize=10)

for p in axes[0].patches:
    width = p.get_width()
    axes[0].text(width, p.get_y() + p.get_height()/2, f'{int(width):,}',
             ha='left', va='center', fontsize=10)

sns.barplot(x='bill_count', y='customer', data=top_10_agents_bill_count_2023_onwards, palette='pastel', ax=axes[1])
axes[1].set_title('Top 10 Agent/Reseller/Dealer by Bill Count (2023 Onwards)', fontsize=16, fontweight='bold')
axes[1].set_xlabel('Number of Bills', fontsize=15)
axes[1].set_ylabel('Customer', fontsize=12)
axes[1].tick_params(axis='both', which='major', labelsize=10)

for p in axes[1].patches:
    width = p.get_width()
    axes[1].text(width, p.get_y() + p.get_height()/2, f'{int(width)}',
             ha='left', va='center', fontsize=10)

plt.tight_layout()
plt.show()
```



```
loyal_customers = billdata[billdata['customer_type'] == 'Loyal']

loyal_customer_summary = loyal_customers.groupby('customer').agg(
    total_revenue=('total_amount', 'sum'),
    bill_count=('bill_no', 'nunique')
).reset_index()

top_20_loyal_by_bills = loyal_customer_summary.sort_values(by='bill_count', ascending=False).head(20)

plt.figure(figsize=(10, 8))
ax = sns.barplot(x='bill_count', y='customer', data=top_20_loyal_by_bills, palette='viridis')
plt.title('Top 20 Loyal Customers by Number of Bills', fontsize=16, fontweight='bold')
plt.xlabel('Number of Bills', fontsize=12)
plt.ylabel('Customer', fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)

for p in ax.patches:
    width = p.get_width()
    customer_name = p.get_y() + p.get_height()/2
    customer_data = top_20_loyal_by_bills[top_20_loyal_by_bills['bill_count'] == width].iloc[0]
    customer_revenue = customer_data['total_revenue']
    customer_bill_count = customer_data['bill_count']

    plt.text(width, p.get_y() + p.get_height()/2 - 0.1, f'({int(customer_bill_count)} bills)',
             ha='left', va='center', fontsize=9)

    plt.text(width, p.get_y() + p.get_height()/2 + 0.25, f'{int(customer_revenue):,}',
             ha='left', va='center', fontsize=8)

plt.tight_layout()
plt.show()
```



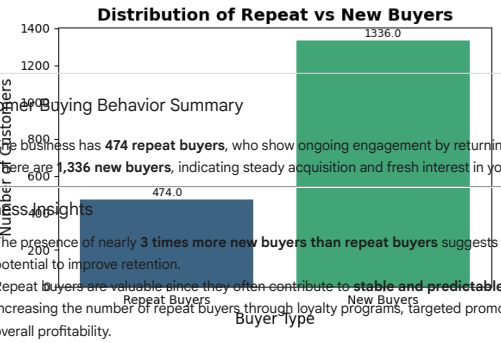

```
Start coding or generate with AI.
```

ADVANCED / DEEP ANALYSIS

Cohort & Repeat Customer Analysis

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

billdata['date'] = pd.to_datetime(billdata['date'])
first_purchase = billdata.groupby('customer')['date'].min().reset_index()
first_purchase.columns = ['customer','first_purchase_date']
bill_with_fp = pd.merge(billdata, first_purchase, on='customer')
bill_with_fp['days_since_first'] = (bill_with_fp['date'] - bill_with_fp['first_purchase_date']).dt.days
repeaters = bill_with_fp[bill_with_fp['days_since_first']>0]['customer'].nunique()
newers = bill_with_fp[bill_with_fp['days_since_first']==0]['customer'].nunique()
print(f"Repeat buyers: {repeaters}, New buyers: {newers}")

labels = ['Repeat Buyers', 'New Buyers']
counts = [repeaters, newers]

plt.figure(figsize=(6, 4))
ax = sns.barplot(x=labels, y=counts, palette='viridis')
plt.title('Distribution of Repeat vs New Buyers', fontsize=14, fontweight='bold')
plt.xlabel('Buyer Type', fontsize=12)
plt.ylabel('Number of Customers', fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)

for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', xytext=(0, 5), textcoords='offset points', fontsize=9)

plt.tight_layout()
plt.show()
```

```
Repeat buyers: 474, New buyers: 1336
```



Customer Buying Behavior Summary

- The business has **474 repeat buyers**, who show ongoing engagement by returning to make additional purchases.
- There are **1,336 new buyers**, indicating steady acquisition and fresh interest in your products.

Business Insights

- The presence of nearly **3 times more new buyers than repeat buyers** suggests **healthy customer acquisition**, but also highlights potential to improve retention.
- Repeat buyers are valuable since they often contribute to **stable and predictable revenue streams**.
- Increasing the number of repeat buyers through loyalty programs, targeted promotions, and excellent customer service can improve overall profitability.
- Tracking this balance between new and repeat customers regularly provides a clear picture of business growth dynamics and customer loyalty.

CORRELATION

```python
item_bill_amount = itemdata.merge(billdata[['bill_no','total_amount']], on='bill_no')
cor = item_bill_amount['quantity'].corr(item_bill_amount['total_amount'])
print("Correlation between item quantity and bill amount:", cor)

```

```
Correlation between item quantity and bill amount: -0.02353434803421006
```

Correlation Between Item Quantity and Bill Amount

- The correlation coefficient is **-0.0235**, which indicates a **very weak negative correlation** between the quantity of items purchased and the bill amount.
- This means that as quantity increases slightly, the bill amount tends to decrease very slightly, but the relationship is almost negligible.
- Such a weak inverse relationship could arise because customers buying large quantities might be purchasing low-priced items (e.g., consumables or spare parts), while some bills with fewer items could be high-value products or bulk-priced hardware.
- This finding aligns with typical retail scenarios where **high-value bills do not necessarily correspond to high quantities**, and vice versa. For example, buying a few expensive POS terminals drives up bill amount, while purchasing many low-cost thermal paper rolls increases quantity but not bill amount proportionally.

Business Takeaway

- The near-zero correlation indicates that **quantity alone is not a reliable predictor of total bill value** in this business.
- Pricing, product mix, and customer type likely play larger roles in driving bill amounts than sheer item count.
- This insight can guide segmentation and pricing strategies by recognizing that different product categories and order types behave differently in volume-to-value relationships.

Basket (Market Basket Analysis) — Top Item Combos

```python
from itertools import combinations
from collections import Counter

filtered_itemdata = itemdata[~itemdata['item'].str.contains('GEAR|MECH', regex=True)].copy()

basket = filtered_itemdata.groupby('bill_no')['item'].apply(list)

basket = basket[basket.apply(len) > 1]

all_combos = Counter([combo for items in basket for combo in combinations(items, 2)])

print("Top 10 Item Combinations (size 2), excluding items containing 'GEAR' or 'MECH':")
for combo, count in all_combos.most_common(10):
    print(f"- {combo[0]} and {combo[1]}: {count}")
```

```
Top 10 Item Combinations (size 2), excluding items containing 'GEAR' or 'MECH':
- ELECTRONIC CASH REGISTER T-90 and ELECTRONIC CASH REGISTER T-20: 53
- ELECTRONIC CASH REGISTER T-90 and ELECTRONIC CASH REGISTER ZIP-20: 51
- RPT-260IV SU CLASSIC THERMAL PRINTER and ELECTRONIC CASH DRAWER: 48
- ELECTRONIC CASH REGISTER T-20 and ELECTRONIC CASH REGISTER T-90: 38
- ELECTRONIC CASH REGISTER T-20 and ELECTRONIC CASH REGISTER ZIP-20: 33
- ELECTRONIC CASH REGISTER T-10 and ELECTRONIC CASH REGISTER T-20: 32
- RPT-260IV SU CLASSIC THERMAL PRINTER and MJ-2818B 2D BAR CODE SCANNER: 32
- ELECTRONIC CASH REGISTER T-10 and ELECTRONIC CASH REGISTER T-90: 31
- ELECTRONIC CASH REGISTER T-90 and ELECTRONIC CASH REGISTER T-10: 31
- ELECTRONIC CASH REGISTER T-20 and ELECTRONIC CASH DRAWER: 27
```

Key Insights: Top Product Pairings (with Dealer Bulk Sales Noted for ECR)

- **Electronic Cash Registers (ECR) Dominate Combos Due to Dealer Bulk Sales:**
  Nearly all top item pairs involve various models of "Electronic Cash Register" (T-90, T-20, ZIP-20, T-10). This dominance is primarily because dealers place **bulk orders** for these registers, significantly driving up their sales compared to regular retail demand.
- **Accessory Attachments Are Common:**
  "RPT-260IV SU Classic Thermal Printer" and "MJ-2818B 2D Barcode Scanner" often appear in combinations with ECRs and cash drawers, showing that these accessories are frequently bundled with main register hardware.
- **Drawer Upsell Opportunity:**
  Frequent pairing of "Electronic Cash Drawer" with register models highlights an opportunity for upselling drawers alongside core POS hardware.
- **Bundling Patterns Reflect Core POS Components:**
  The top product combinations focus on essential POS hardware (registers, printers, drawers, scanners), rather than peripheral or unrelated accessories.

Actionable Takeaways

- **Dealer-Focused Stocking & Promotion for ECR:**
  Since ECR sales spikes are due to bulk dealer purchases, inventory and promotions should target dealer channels for these products.
- **Cross-Sell Accessories in Bundles:**
  Promote bundled offers involving printers, scanners, and cash drawers with ECRs to maximize sales.
- **Interpret Sales Data Considering Dealer Influence:**
  When analyzing ECR sales, factor in the bulk dealer orders to distinguish true retail demand from distribution patterns.

Key Insights: Top Product Pairings (with Dealer Bulk Sales Noted for ECR)

- **Electronic Cash Registers (ECR) Dominate Combos Due to Dealer Bulk Sales:**
  Nearly all top item pairs involve various models of "Electronic Cash Register" (T-90, T-20, ZIP-20, T-10). This dominance is primarily because dealers place **bulk orders** for these registers, significantly driving up their sales compared to regular retail demand.
- **Accessory Attachments Are Common:**
  "RPT-260IV SU Classic Thermal Printer" and "MJ-2818B 2D Barcode Scanner" often appear in combinations with ECRs and cash drawers, showing that these accessories are frequently bundled with main register hardware.
- **Drawer Upsell Opportunity:**
  Frequent pairing of "Electronic Cash Drawer" with register models highlights an opportunity for upselling drawers alongside core POS hardware.
- **Bundling Patterns Reflect Core POS Components:**
  The top product combinations focus on essential POS hardware (registers, printers, drawers, scanners), rather than peripheral or unrelated accessories.

Actionable Takeaways

- **Dealer-Focused Stocking & Promotion for ECR:**
  Since ECR sales spikes are due to bulk dealer purchases, inventory and promotions should target dealer channels for these products.
- **Cross-Sell Accessories in Bundles:**
  Promote bundled offers involving printers, scanners, and cash drawers with ECRs to maximize sales.
- **Interpret Sales Data Considering Dealer Influence:**
  When analyzing ECR sales, factor in the bulk dealer orders to distinguish true retail demand from distribution patterns.

IS THERE A SIGNIFICANT DIFFERENCE IN THE MEAN SALES OF VARIOUS PRODUCT CATEGORIES?

```python
from scipy.stats import f_oneway

item_bill = itemdata.merge(billdata[['bill_no', 'total_amount']], on='bill_no')

min_obs = 30
grouped = item_bill.groupby('product_category')
groups = [group['total_amount'] for name, group in grouped if len(group) >= min_obs]

anova_stat, anova_p = f_oneway(*groups)

alpha = 0.05
confidence_level = 1 - alpha
null_hypothesis = "The mean sales are the same across all product categories."
alternate_hypothesis = "The mean sales are different for at least one product category."

print("Null Hypothesis (H0):", null_hypothesis)
print("Alternate Hypothesis (H1):", alternate_hypothesis)
print("Confidence Level:", confidence_level)
print("Alpha Value:", alpha)
print("-" * 30)
print(f"ANOVA Statistic: {anova_stat:.4f}")
print(f"ANOVA p-value: {anova_p:.4f}")

print("-" * 30)
if anova_p < alpha:
    print(f"Result: Since the p-value ({anova_p:.4f}) is less than the alpha value ({alpha}), we reject the null hypothesis.")
    print("Conclusion: There is a significant difference in mean sales among product categories.")
else:
    print(f"Result: Since the p-value ({anova_p:.4f}) is greater than the alpha value ({alpha}), we fail to reject the null hypothesis.")
    print("Conclusion: There is no significant difference in mean sales among product categories.")
```

```
Null Hypothesis (H0): The mean sales are the same across all product categories.
Alternate Hypothesis (H1): The mean sales are different for at least one product category.
Confidence Level: 0.95
Alpha Value: 0.05
------------------------------
ANOVA Statistic: 43.2729
ANOVA p-value: 0.0000
------------------------------
Result: Since the p-value (0.0000) is less than the alpha value (0.05), we reject the null hypothesis.
Conclusion: There is a significant difference in mean sales among product categories.
```

ANOVA Test Summary and Insights

- The **Null Hypothesis (H0)** states that the mean sales are the same across all product categories.
- The **Alternative Hypothesis (H1)** states that the mean sales differ for at least one product category.
- The test was conducted with a **95% confidence level** and an alpha of **0.05**.
- The computed **ANOVA statistic is 43.27** with a very small **p-value (0.0000)**.
- Since the p-value is less than the alpha (0.05), we **reject the null hypothesis**, confirming that **there is a statistically significant difference in mean sales among product categories**.

Business Implications

- This result tells us that not all product categories contribute equally to sales, which likely reflects market demand, pricing, and product popularity variations.
- Understanding which categories outperform or underperform can help optimize inventory, marketing, and product development strategies.

Next Steps: Post-Hoc Analysis

- Since the ANOVA test indicates significant differences, the next step is to perform a **Tukey's Honest Significant Difference (HSD) test**.
- Tukey's HSD will help pinpoint **exactly which product categories differ significantly** in their mean sales.
- This deeper insight will enable more granular targeting—for example, identifying underperforming categories to improve or successful categories to focus on expanding.

```python
from statsmodels.stats.multicomp import pairwise_tukeyhsd

item_bill = itemdata.merge(billdata[['bill_no', 'total_amount']], on='bill_no')
data_for_tukey = item_bill[['total_amount', 'product_category']].copy()

min_obs_tukey = 30
category_counts = data_for_tukey['product_category'].value_counts()
categories_to_include = category_counts[category_counts >= min_obs_tukey].index
data_for_tukey_filtered = data_for_tukey[data_for_tukey['product_category'].isin(categories_to_include)]

tukey_result = pairwise_tukeyhsd(endog=data_for_tukey_filtered['total_amount'],
                                 groups=data_for_tukey_filtered['product_category'],
                                 alpha=0.05)

print(tukey_result)
```

```
                AMC       POS TERMINAL  19419.3587    0.0  12428.0666  26410.6509   True
                AMC            SCANNER  11482.7232 0.0011   2555.1074  20410.3391   True
                AMC            SERVICE -11893.0254 0.0395 -23541.4804   -244.5705   True
                AMC           SOFTWARE   -9176.721 0.2208  -20030.954    1677.512  False
                AMC   SOFTWARE LICENSE -11494.9216 0.5171 -27530.0622    4540.219  False
                AMC         SPARE PART  -7603.0079 0.0024 -13764.9558  -1441.0599   True
                AMC    THERMAL PRINTER    457.9706    1.0  -5762.5125   6678.4538  False
                AMC            WINDOWS   52241.925    0.0  37017.6969  67466.1532   True
    BARCODE PRINTER        CASH DRAWER   1201.4409    1.0  -8930.1216  11333.0034  False
    BARCODE PRINTER COMPUTER ACCESSORY  -1209.6602    1.0  -22865.995  20446.6747  False
    BARCODE PRINTER      COMPUTER PART  13039.9909 0.4997   -4985.732  31065.7139  False
    BARCODE PRINTER         CONSUMABLE -13316.3452 0.0036 -24369.0051  -2263.6854   True
    BARCODE PRINTER   CURRENCY COUNTER -10121.3977 0.8669 -27942.1673   7699.3718  False
    BARCODE PRINTER                ECR  -6280.1561 0.6011 -15443.6209   2883.3088  False
    BARCODE PRINTER              OTHER  -5865.6014 0.9999 -25676.9365  13945.7337  False
    BARCODE PRINTER       POS TERMINAL  17007.2375    0.0   7215.3908  26799.0841   True
    BARCODE PRINTER            SCANNER    9070.602 0.2986  -2185.7038  20326.9078  False
    BARCODE PRINTER            SERVICE -14305.1467 0.0254 -27821.3812   -788.9121   True
    BARCODE PRINTER           SOFTWARE -11588.8422 0.1346 -24426.9289   1249.2444  False
    BARCODE PRINTER   SOFTWARE LICENSE -13907.0428 0.3169 -31346.3039   3532.2183  False
    BARCODE PRINTER         SPARE PART -10015.1291  0.018 -19233.1385   -797.1197   True
    BARCODE PRINTER    THERMAL PRINTER  -1954.1506    1.0 -11211.3913   7303.0901  False
    BARCODE PRINTER            WINDOWS  49829.8038    0.0   33133.121  66526.4866   True
        CASH DRAWER COMPUTER ACCESSORY  -2411.1011    1.0 -22758.8702   17936.668  False
        CASH DRAWER      COMPUTER PART    11838.55 0.5074  -4591.9569  28269.0568  False
        CASH DRAWER         CONSUMABLE -14517.7861    0.0 -22715.1124  -6320.4599   True
        CASH DRAWER   CURRENCY COUNTER -11322.8386 0.5655 -27528.2295   4882.5523  False
        CASH DRAWER                ECR   -7481.597 0.0002 -12867.1291  -2096.0649   True
        CASH DRAWER              OTHER  -7067.0423 0.9965 -25438.8649  11304.7802  False
        CASH DRAWER       POS TERMINAL  15805.7966    0.0   9409.3031    22202.29   True
        CASH DRAWER            SCANNER   7869.1611 0.1048   -600.7437  16339.0659  False
        CASH DRAWER            SERVICE -15506.5876 0.0003 -26808.0682  -4205.1069   True
        CASH DRAWER           SOFTWARE -12790.2832 0.0029 -23271.2819  -2309.2844   True
        CASH DRAWER   SOFTWARE LICENSE -15108.4838 0.0791  -30893.372    676.4044  False
        CASH DRAWER         SPARE PART   -11216.57    0.0 -16694.3949  -5738.7452   True
        CASH DRAWER    THERMAL PRINTER  -3155.5915 0.8648    -8699.18   2387.9969  False
        CASH DRAWER            WINDOWS  48628.3629    0.0  33667.9456  63588.7801   True
    COMPUTER ACCESSORY      COMPUTER PART  14249.6511 0.8719 -10975.5315  39474.8336  False
    COMPUTER ACCESSORY         CONSUMABLE  -12106.685 0.8433  -32928.415   8715.0449  False
    COMPUTER ACCESSORY   CURRENCY COUNTER  -8911.7375 0.9986 -33990.8718  16167.3967  False
    COMPUTER ACCESSORY                ECR  -5070.4959    1.0 -24953.9623  14812.9705  False
    COMPUTER ACCESSORY              OTHER  -4655.9412    1.0 -31186.5301  21874.6476  False
    COMPUTER ACCESSORY       POS TERMINAL  18216.8977 0.1346  -1963.8706  38397.6659  False
    COMPUTER ACCESSORY            SCANNER  10280.2622 0.9584 -10650.2793  31210.8036  False
    COMPUTER ACCESSORY            SERVICE -13095.4865 0.8288 -35323.2112   9132.2382  False
    COMPUTER ACCESSORY           SOFTWARE -10379.1821 0.9689 -32201.1805  11442.8164  False
    COMPUTER ACCESSORY   SOFTWARE LICENSE -12697.3827 0.9407 -37506.8763   12112.111  False
    COMPUTER ACCESSORY         SPARE PART  -8805.4689 0.9846 -28714.1315  11103.1936  False
    COMPUTER ACCESSORY    THERMAL PRINTER   -744.4904    1.0  -20671.348  19182.3671  False
    COMPUTER ACCESSORY            WINDOWS   51039.464    0.0  26746.2064  75332.7216   True
         COMPUTER PART         CONSUMABLE -26356.3361    0.0   -43370.28  -9342.3923   True
         COMPUTER PART   CURRENCY COUNTER -23161.3886 0.0275 -45181.6707  -1141.1066   True
         COMPUTER PART                ECR  -19320.147  0.003 -35172.0265  -3468.2675   True
```

Key Insights from Tukey HSD Test on Product Groups

- **Significant Differences Detected:**
  - Several product category pairs show statistically significant differences in their mean sales values (column `reject` = **True**).
  - Notably, categories like **"AMC"** vs **"CONSUMABLE"**, **"SCANNER"**, **"POS TERMINAL"**, **"WINDOWS"** and several others exhibit large, significant differences (very low p-adj values such as 0.0017, 0.0011, 0.0, etc.).
- **Top Performing Segments:**
  - **"WINDOWS"** consistently outperforms many other categories, showing a much higher mean sales value compared to "AMC", "BARCODE PRINTER", "CASH DRAWER", and nearly all other product groups (high meandiff, reject True).
  - **"POS TERMINAL"** also shows strong, significantly higher means when compared to "AMC", "CONSUMABLE", "BARCODE PRINTER", "CASH DRAWER", etc.
- **Categories with Lower Means:**
  - Categories like **"CONSUMABLE"**, **"SERVICE"**, **"SPARE PART"**, often have significantly lower mean sales compared to top hardware/software categories.
- **Actionable Patterns:**
  - Many "True" rejects are in hardware-vs-accessory/software-vs-accessory comparisons (e.g., "ECR" vs "SCANNER", "COMPUTER PART" vs "SERVICE").
  - Only certain category pairs (e.g., "AMC" vs "POS TERMINAL") pass the significance threshold in the positive direction (high value and significant).

How to Use This for Business

- **Product Promotion:**
  Focus future promotions on **top categories** (e.g., "WINDOWS", "POS TERMINAL", "SCANNER") as these have clear, statistical evidence of higher mean sales compared to accessories and consumables.
- **Inventory/Marketing Spend:**
  Consider reducing inventory or marketing efforts for **"CONSUMABLE"** and **"SERVICE"** categories due to their lower performance.
- **Pricing Strategy:**
  The test supports data-driven pricing differentiation among top and bottom-performing categories.

Key Insights from Tukey HSD Test on Product Groups

- **Significant Differences Detected:**
  - Several product category pairs show statistically significant differences in their mean sales values (column `reject` = **True**).
  - Notably, categories like **"AMC"** vs **"CONSUMABLE"**, **"SCANNER"**, **"POS TERMINAL"**, **"WINDOWS"** and several others exhibit large, significant differences (very low p-adj values such as 0.0017, 0.0011, 0.0, etc.).
- **Top Performing Segments:**
  - **"WINDOWS"** consistently outperforms many other categories, showing a much higher mean sales value compared to "AMC", "BARCODE PRINTER", "CASH DRAWER", and nearly all other product groups (high meandiff, reject True).
  - **"POS TERMINAL"** also shows strong, significantly higher means when compared to "AMC", "CONSUMABLE", "BARCODE PRINTER", "CASH DRAWER", etc.
- **Categories with Lower Means:**
  - Categories like **"CONSUMABLE"**, **"SERVICE"**, **"SPARE PART"**, often have significantly lower mean sales compared to top hardware/software categories.
- **Actionable Patterns:**
  - Many "True" rejects are in hardware-vs-accessory/software-vs-accessory comparisons (e.g., "ECR" vs "SCANNER", "COMPUTER PART" vs "SERVICE").
  - Only certain category pairs (e.g., "AMC" vs "POS TERMINAL") pass the significance threshold in the positive direction (high value and significant).

How to Use This for Business

- **Product Promotion:**
  Focus future promotions on **top categories** (e.g., "WINDOWS", "POS TERMINAL", "SCANNER") as these have clear, statistical evidence of higher mean sales compared to accessories and consumables.
- **Inventory/Marketing Spend:**
  Consider reducing inventory or marketing efforts for **"CONSUMABLE"** and **"SERVICE"** categories due to their lower performance.

- **Pricing Strategy:**
  The test supports data-driven pricing differentiation among top and bottom-performing categories.

```
Start coding or generate with AI.
```

Detailed Business Insights from Data Analysis

The analysis of the dataset, which includes bill-wise, item-wise, and customer information, reveals critical insights into Billson India's performance.

- **Uneven Product Performance:** The dataset shows a clear disparity in the performance of different product categories. Categories like "WINDOWS," "POS TERMINAL," and "SCANNER" are top-performing with a statistical evidence of higher mean sales compared to other product groups. Conversely, categories such as "CONSUMABLE" and "SERVICE" show lower performance, indicating that while core products are selling well, there is a missed opportunity for recurring revenue from accessories and services.
- **Dominance of Online Payments:** The data unequivocally highlights a strong customer preference for digital transactions. Online payments account for 4,356 bills compared to only 148 cash payments. This digital preference is even more pronounced in terms of revenue, with online payment modes contributing over ₹8.5 crore, which is more than 98% of the total revenue, vastly surpassing cash sales at ₹9.74 lakh.
- **Sales Fluctuations by Day of the Week:** The total sales revenue varies significantly depending on the day of the week. The data shows that the highest sales revenue occurs on Monday and Wednesday, followed closely by Saturday and Thursday. There is a slight dip in sales on Friday and a significant dip on Sunday.
- **Identification of High-Value Transactions:** The outlier detection analysis shows that a small number of transactions are significantly higher than typical sales. With an upper bound of ₹47,937.50, a total of 266 outlier bills were detected, which are bills with values exceeding this amount.
- **Customer Segmentation and its Revenue Impact:** The data classifies customers into three types based on their bill count: "Agent/reseller/dealer," "Loyal," and "Casual". The analysis of total revenue by customer type shows that "Casual" customers contribute the most revenue at approximately ₹37.2 million, followed by "Agent/reseller/dealer" at ₹32.6 million, and "Loyal" customers at ₹16.2 million.
- **Sales Trends by Month:** The dataset also provides insight into monthly sales performance. For instance, an observation from the data shows that sales in **May** were lower with bill counts of 115 and 99 in weeks 19 and 20 respectively, when compared to other weeks and months. This indicates potential sales troughs during certain periods of the year.

RECOMMENDATIONS:

Based on the data-driven insights, here are actionable strategies to address challenges and capitalize on identified opportunities.

- **a. Enhance Focus on High-Value Products:** The data analysis suggests that product promotions should be focused on top-performing categories such as **"WINDOWS," "POS TERMINAL,"** and **"SCANNER"**. Given their statistically higher mean sales, concentrating marketing efforts and inventory on these items is likely to yield the highest returns.
- **b. Develop a Recurring Revenue Strategy for Underperforming Categories:** The data indicates that "CONSUMABLE" and "SERVICE" categories are underperforming. To increase sales in these areas, the company should implement a bundling strategy. For example, a new POS terminal sale could be bundled with a one-year service and consumable package at a discounted rate.
- **c. Capitalize on Online Payment Trends:** The overwhelming preference for online payments is a significant finding. The company should ensure that its POS solutions are fully optimized to handle a diverse range of digital payment options. Marketing materials and sales pitches should emphasize the seamless integration with online payment gateways as a key feature, positioning the company as a modern and reliable provider.
- **d. Implement a Targeted Customer Engagement Strategy:** The data reveals that "Agent/reseller/dealer" and "Loyal" customers have lower average revenue per bill compared to "Casual" customers, despite being significant contributors to overall sales. Billson India should develop a two-pronged strategy:
  1. **For Agents/Dealers:** Implement tailored incentives such as volume discounts and flexible credit terms to strengthen these relationships and encourage larger orders.
  2. **For Loyal Customers:** The company should implement a customer-retention program that offers exclusive discounts or early access to new products. This will help to maintain long-term partnerships and increase their average bill value over time.
- **e. Initiate Promotional Activities in Low-Sales Months:** The data shows a notable dip in sales during certain months, such as May. To counter this, Billson India should launch targeted promotional campaigns during these historically lean periods. For example, offering a special "summer refresh" discount or a bundled product offer during these months could help stabilize revenue throughout the year.
- **f. Targeted Marketing in Geographically Accessible, High-Potential Cities:** Billson India is headquartered in Chandigarh. The customer dataset shows a geographical distribution of clients in cities such as **Zirakpur, Ambala, and Mohali**, which are all near Chandigarh. There is a business opportunity to increase sales in these and other easily approachable cities where sales are currently low. A direct, localized marketing campaign that highlights the company's proximity and offers faster service and support could attract more clients from these regions, turning them into high-sales territories.

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

Double-click (or enter) to edit

```
billdata.to_excel('billdata.xlsx', index=False)
itemdata.to_excel('itemdata.xlsx', index=False)
customerdata.to_excel('customerdata.xlsx', index=False)
finaldata.to_excel('finaldata.xlsx', index=False)
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```