[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

# Importing orders table

[3]:
```
orders = pd.read_csv('Orders.csv')
```

[4]:
```
orders.describe()
```

[4]:

| | OrderID | CustomerID | ProductID | Quantity | TotalAmount | StoreID |
|---|---|---|---|---|---|---|
| **count** | 10000.00000 | 10000.00000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 |
| **mean** | 8000.50000 | 11000.50000 | 1024.915200 | 2.981500 | 501.416700 | 2049.074200 |
| **std** | 2886.89568 | 2886.89568 | 14.610303 | 1.413421 | 431.949384 | 28.908574 |
| **min** | 3001.00000 | 6001.00000 | 1000.000000 | 1.000000 | 14.000000 | 2000.000000 |
| **25%** | 5500.75000 | 8500.75000 | 1012.000000 | 2.000000 | 180.000000 | 2024.000000 |
| **50%** | 8000.50000 | 11000.50000 | 1025.000000 | 3.000000 | 385.000000 | 2049.000000 |
| **75%** | 10500.25000 | 13500.25000 | 1038.000000 | 4.000000 | 700.000000 | 2074.000000 |
| **max** | 13000.00000 | 16000.00000 | 1050.000000 | 5.000000 | 2995.000000 | 2099.000000 |

[5]:
```
orders.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 10 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   OrderID               10000 non-null  int64 
 1   CustomerID            10000 non-null  int64 
 2   OrderDateTime         10000 non-null  object
 3   ProductID             10000 non-null  int64 
 4   Quantity              10000 non-null  int64 
 5   TotalAmount           10000 non-null  int64 
 6   StoreID               10000 non-null  int64 
 7   OrderStatus           10000 non-null  object
 8   ExpectedDeliveryTime  10000 non-null  object
 9   ActualDeliveryTime    10000 non-null  object
dtypes: int64(6), object(4)
memory usage: 781.4+ KB
```

[6]:
```
orders.isnull().sum()
```

[6]:
```
OrderID               0
CustomerID            0
OrderDateTime         0
ProductID             0
Quantity              0
TotalAmount           0
StoreID               0
OrderStatus           0
ExpectedDeliveryTime  0
ActualDeliveryTime    0
dtype: int64
```

[7]:
```
orders.dtypes
```

[7]:
```
OrderID                int64
CustomerID             int64
OrderDateTime         object
ProductID              int64
Quantity               int64
TotalAmount            int64
StoreID                int64
OrderStatus           object
ExpectedDeliveryTime  object
ActualDeliveryTime    object
dtype: object
```

[8]:
```
orders.duplicated()
```

[8]:
```
0       False
1       False
2       False
3       False
4       False
        ...
9995    False
9996    False
9997    False
9998    False
9999    False
Length: 10000, dtype: bool
```

## Changing datatypes of orderdatetime from object to datetime

[9]:
```
orders['OrderDateTime'] = pd.to_datetime(orders['OrderDateTime'], format='%Y-%m-%d %H:%M:%S.%f')
orders
```

[9]:

| | OrderID | CustomerID | OrderDateTime | ProductID | Quantity | TotalAmount | StoreID | OrderStatus | ExpectedDeliveryTime | ActualDeliveryTime |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 3001 | 6001 | 2025-01-22 20:02:03.428463 | 1025 | 2 | 90 | 2051 | Delivered | 2025-01-22 20:12:03.428463 | 2025-01-22 20:27:03.428463 |
| **1** | 3002 | 6002 | 2025-02-18 15:58:03.428463 | 1027 | 1 | 99 | 2084 | Delivered | 2025-02-18 16:08:03.428463 | 2025-02-18 16:10:03.428463 |
| **2** | 3003 | 6003 | 2025-02-04 06:58:03.428463 | 1001 | 3 | 780 | 2008 | Delivered | 2025-02-04 07:08:03.428463 | 2025-02-04 07:10:03.428463 |
| **3** | 3004 | 6004 | 2025-02-08 17:00:03.428463 | 1025 | 4 | 180 | 2008 | Delivered | 2025-02-08 17:10:03.428463 | 2025-02-08 17:25:03.428463 |
| **4** | 3005 | 6005 | 2025-02-06 18:43:03.428463 | 1023 | 3 | 405 | 2071 | Delivered | 2025-02-06 18:53:03.428463 | 2025-02-06 19:08:03.428463 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **9995** | 12996 | 15996 | 2025-02-06 17:44:03.428463 | 1035 | 1 | 90 | 2036 | Delivered | 2025-02-06 17:54:03.428463 | 2025-[illegible] 17:59:03.42[illegible] |
| **9996** | 12997 | 15997 | 2025-02-08 20:34:03.428463 | 1043 | 4 | 560 | 2035 | Delivered | 2025-02-08 20:44:03.428463 | 2025-02-08 20:54:03.428463 |
| **9997** | 12998 | 15998 | 2025-02-08 06:43:03.428463 | 1017 | 4 | 960 | 2085 | Delivered | 2025-02-08 06:53:03.428463 | 2025-02-08 06:55:03.428463 |
| **9998** | 12999 | 15999 | 2025-02-15 05:58:03.428463 | 1039 | 4 | 440 | 2046 | Delivered | 2025-02-15 06:08:03.428463 | 2025-02-15 06:08:03.428463 |
| **9999** | 13000 | 16000 | 2025-01-21 18:20:03.428463 | 1009 | 4 | 140 | 2008 | Delivered | 2025-01-21 18:30:03.428463 | 2025-01-21 18:30:03.428463 |

10000 rows × 10 columns

[10]:
```
orders.dtypes
```

[10]:
```
OrderID                       int64
CustomerID                    int64
OrderDateTime        datetime64[ns]
ProductID                     int64
Quantity                      int64
TotalAmount                   int64
StoreID                       int64
OrderStatus                  object
ExpectedDeliveryTime         object
ActualDeliveryTime           object
dtype: object
```

[11]:
```
orders['ExpectedDeliveryTime'] = pd.to_datetime(orders['ExpectedDeliveryTime'], format='%Y-%m-%d %H:%M:%S.%f')
orders
orders['ActualDeliveryTime'] = pd.to_datetime(orders['ActualDeliveryTime'], format='%Y-%m-%d %H:%M:%S.%f')
orders
```

[11]:

| | OrderID | CustomerID | OrderDateTime | ProductID | Quantity | TotalAmount | StoreID | OrderStatus | ExpectedDeliveryTime | ActualDeliveryTime |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 3001 | 6001 | 2025-01-22 20:02:03.428463 | 1025 | 2 | 90 | 2051 | Delivered | 2025-01-22 20:12:03.428463 | 2025-01-22 20:27:03.428463 |
| **1** | 3002 | 6002 | 2025-02-18 15:58:03.428463 | 1027 | 1 | 99 | 2084 | Delivered | 2025-02-18 16:08:03.428463 | 2025-02-18 16:10:03.428463 |
| **2** | [illegible] | [illegible] | 2025-02-04 [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | 2025-02-04 [illegible] | 2025-02-04 [illegible] |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 3003 | 6003 | 06:58:03.428463 | 1001 | 3 | 780 | 2008 | Delivered | 07:08:03.428463 | 07:10:03.428463 |
| **3** | 3004 | 6004 | 2025-02-08 17:00:03.428463 | 1025 | 4 | 180 | 2008 | Delivered | 2025-02-08 17:10:03.428463 | 2025-02-08 17:25:03.428463 |
| **4** | 3005 | 6005 | 2025-02-06 18:43:03.428463 | 1023 | 3 | 405 | 2071 | Delivered | 2025-02-06 18:53:03.428463 | 2025-02-06 19:08:03.428463 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **9995** | 12996 | 15996 | 2025-02-06 17:44:03.428463 | 1035 | 1 | 90 | 2036 | Delivered | 2025-02-06 17:54:03.428463 | 2025-02-06 17:59:03.428463 |
| **9996** | 12997 | 15997 | 2025-02-08 20:34:03.428463 | 1043 | 4 | 560 | 2035 | Delivered | 2025-02-08 20:44:03.428463 | 2025-02-08 20:54:03.42... |
| **9997** | 12998 | 15998 | 2025-02-08 06:43:03.428463 | 1017 | 4 | 960 | 2085 | Delivered | 2025-02-08 06:53:03.428463 | 2025-02-08 06:55:03.428463 |
| **9998** | 12999 | 15999 | 2025-02-15 05:58:03.428463 | 1039 | 4 | 440 | 2046 | Delivered | 2025-02-15 06:08:03.428463 | 2025-02-15 06:08:03.428463 |
| **9999** | 13000 | 16000 | 2025-01-21 18:20:03.428463 | 1009 | 4 | 140 | 2008 | Delivered | 2025-01-21 18:30:03.428463 | 2025-01-21 18:30:03.428463 |

10000 rows × 10 columns

[12]:
```
orders.dtypes
```

[12]:
```
OrderID                    int64
CustomerID                 int64
OrderDateTime     datetime64[ns]
ProductID                  int64
Quantity                   int64
TotalAmount                int64
StoreID                    int64
OrderStatus               object
ExpectedDeliveryTime  datetime64[ns]
ActualDeliveryTime    datetime64[ns]
dtype: object
```

# Importing dark warehouses table

[13]:
```
dw = pd.read_csv('Dark_Warehouses.csv')
dw
```

[13]:

| | StoreID | StoreName | Location | Pincode | StoreCapacity |
|---|---|---|---|---|---|
| **0** | 2000 | Zepto Hub 1 | Andheri West, Mumbai | 400053 | 1134 |
| **1** | 2001 | Zepto Hub 2 | Borivali, Mumbai | 400092 | 1225 |
| **2** | 2002 | Zepto Hub 3 | Powai, Mumbai | 400076 | 979 |
| **3** | 2003 | Zepto Hub 4 | Koramangala, Bangalore | 560095 | 1217 |
| **4** | 2004 | Zepto Hub 5 | Indiranagar, Bangalore | 560038 | 1126 |
| **...** | ... | ... | ... | ... | ... |
| **95** | 2095 | Zepto Hub 96 | Hinjewadi, Pune | 411057 | 1028 |
| **96** | 2096 | Zepto Hub 97 | Kothrud, Pune | 411038 | 936 |
| **97** | 2097 | Zepto Hub 98 | Viman Nagar, Pune | 411014 | 820 |
| **98** | 2098 | Zepto Hub 99 | Vaishali Nagar, Jaipur | 302021 | 1378 |
| **99** | 2099 | Zepto Hub 100 | Malviya Nagar, Jaipur | 302017 | 1426 |

100 rows × 5 columns

[14]:
```
dw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   StoreID        100 non-null    int64
 1   StoreName      100 non-null    object
 2   Location       100 non-null    object
 3   Pincode        100 non-null    int64
 4   StoreCapacity  100 non-null    int64
dtypes: int64(3), object(2)
memory usage: 4.0+ KB
```

[15]:
```
dw.duplicated()
```

[15]:
```
0     False
1     False
2     False
3     False
4     False
      ...
95    False
96    False
97    False
98    False
99    False
Length: 100, dtype: bool
```

[16]:
```
dw.dtypes
```

[16]:
```
StoreID           int64
StoreName        object
Location         object
Pincode           int64
StoreCapacity     int64
dtype: object
```

[17]:
```
dw.isnull().sum()
```

[17]:
```
StoreID          0
StoreName        0
Location         0
Pincode          0
StoreCapacity    0
dtype: int64
```

[18]:
```
dw.describe()
```

[18]:

| | StoreID | Pincode | StoreCapacity |
|---|---|---|---|
| **count** | 100.000000 | 100.000000 | 100.000000 |
| **mean** | 2049.500000 | 505902.750000 | 1177.290000 |
| **std** | 29.011492 | 121509.042249 | 203.333795 |
| **min** | 2000.000000 | 302017.000000 | 820.000000 |
| **25%** | 2024.750000 | 408283.500000 | 997.500000 |
| **50%** | 2049.500000 | 500057.500000 | 1211.000000 |
| **75%** | 2074.250000 | 600022.750000 | 1370.500000 |
| **max** | 2099.000000 | 700135.000000 | 1493.000000 |

# Importing delivery table

[19]:
```
delivery = pd.read_csv('Delivery.csv')
delivery
```

[19]:

| | DeliveryID | OrderID | DeliveryPartnerName | PickupTime | DeliveryEndTime | DistanceTraveled |
|---|---|---|---|---|---|---|
| **0** | 7001 | 3001 | Arjun Nair | 04:03.4 | 27:03.4 | 1.98 |
| **1** | 7002 | 3002 | Arjun Nair | 00:03.4 | 10:03.4 | 7.93 |
| **2** | 7003 | 3003 | Rohit Sharma | 00:03.4 | 10:03.4 | 4.67 |
| **3** | 7004 | 3004 | Sneha Patil | 02:03.4 | 25:03.4 | 6.93 |
| **4** | 7005 | 3005 | Karan Mehta | 45:03.4 | 08:03.4 | 9.33 |
| **...** | ... | ... | ... | ... | ... | ... |
| **9995** | 16996 | 12996 | Simran Kaur | 46:03.4 | 59:03.4 | 8.07 |
| **9996** | 16997 | 12997 | Rohit Sharma | 36:03.4 | 54:03.4 | 3.61 |
| **9997** | 16998 | 12998 | Karan Mehta | 45:03.4 | 55:03.4 | 5.83 |
| **9998** | 16999 | 12999 | Amit Singh | 00:03.4 | 08:03.4 | 7.94 |
| **9999** | 17000 | 13000 | Karan Mehta | 22:03.4 | 30:03.4 | 8.69 |

10000 rows × 6 columns

[20]:
```
delivery.dtypes
```

[20]:
```
DeliveryID              int64
```

```
OrderID                int64
DeliveryPartnerName    object
PickupTime             object
DeliveryEndTime        object
DistanceTraveled       float64
dtype: object
```

[21]:
```
delivery.describe()
```

[21]:

| | DeliveryID | OrderID | DistanceTraveled |
|---|---|---|---|
| **count** | 10000.00000 | 10000.00000 | 10000.000000 |
| **mean** | 12000.50000 | 8000.50000 | 5.762438 |
| **std** | 2886.89568 | 2886.89568 | 2.457935 |
| **min** | 7001.00000 | 3001.00000 | 1.500000 |
| **25%** | 9500.75000 | 5500.75000 | 3.610000 |
| **50%** | 12000.50000 | 8000.50000 | 5.740000 |
| **75%** | 14500.25000 | 10500.25000 | 7.900000 |
| **max** | 17000.00000 | 13000.00000 | 10.000000 |

[22]:
```
delivery.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 6 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   DeliveryID           10000 non-null  int64
 1   OrderID              10000 non-null  int64
 2   DeliveryPartnerName  10000 non-null  object
 3   PickupTime           10000 non-null  object
 4   DeliveryEndTime      10000 non-null  object
 5   DistanceTraveled     10000 non-null  float64
dtypes: float64(1), int64(2), object(3)
memory usage: 468.9+ KB
```

[23]:
```
delivery.isnull().sum()
```

[23]:
```
DeliveryID           0
OrderID              0
DeliveryPartnerName  0
PickupTime           0
DeliveryEndTime      0
DistanceTraveled     0
dtype: int64
```

[24]:
```
delivery["PickupTime"] = pd.to_datetime(delivery["PickupTime"], format="%M:%S.%f", errors="coerce")
delivery
```

[24]:

| | DeliveryID | OrderID | DeliveryPartnerName | PickupTime | DeliveryEndTime | DistanceTraveled |
|---|---|---|---|---|---|---|
| **0** | 7001 | 3001 | Arjun Nair | 1900-01-01 00:04:03.400 | 27:03.4 | 1.98 |
| **1** | 7002 | 3002 | Arjun Nair | 1900-01-01 00:00:03.400 | 10:03.4 | 7.93 |
| **2** | 7003 | 3003 | Rohit Sharma | 1900-01-01 00:00:03.400 | 10:03.4 | 4.67 |
| **3** | 7004 | 3004 | Sneha Patil | 1900-01-01 00:02:03.400 | 25:03.4 | 6.93 |
| **4** | 7005 | 3005 | Karan Mehta | 1900-01-01 00:45:03.400 | 08:03.4 | 9.33 |
| **...** | ... | ... | ... | ... | ... | ... |
| **9995** | 16996 | 12996 | Simran Kaur | 1900-01-01 00:46:03.400 | 59:03.4 | 8.07 |
| **9996** | 16997 | 12997 | Rohit Sharma | 1900-01-01 00:36:03.400 | 54:03.4 | 3.61 |
| **9997** | 16998 | 12998 | Karan Mehta | 1900-01-01 00:45:03.400 | 55:03.4 | 5.83 |
| **9998** | 16999 | 12999 | Amit Singh | 1900-01-01 00:00:03.400 | 08:03.4 | 7.94 |
| **9999** | 17000 | 13000 | Karan Mehta | 1900-01-01 00:22:03.400 | 30:03.4 | 8.69 |

10000 rows × 6 columns

[25]:
```
delivery["DeliveryEndTime"] = pd.to_datetime(delivery["DeliveryEndTime"], format="%M:%S.%f", errors="coerce")
delivery #date is used only as placeholder because pandas dont have time datatype
```

[25]:

| | DeliveryID | OrderID | DeliveryPartnerName | PickupTime | DeliveryEndTime | DistanceTraveled |
|---|---|---|---|---|---|---|
| **0** | 7001 | 3001 | Arjun Nair | 1900-01-01 00:04:03.400 | 1900-01-01 00:27:03.400 | 1.98 |

| | | | | | | |
|---|---|---|---|---|---|---|
| **1** | 7002 | 3002 | Arjun Nair | 1900-01-01 00:00:03.400 | 1900-01-01 00:10:03.400 | 7.93 |
| **2** | 7003 | 3003 | Rohit Sharma | 1900-01-01 00:00:03.400 | 1900-01-01 00:10:03.400 | 4.67 |
| **3** | 7004 | 3004 | Sneha Patil | 1900-01-01 00:02:03.400 | 1900-01-01 00:25:03.400 | 6.93 |
| **4** | 7005 | 3005 | Karan Mehta | 1900-01-01 00:45:03.400 | 1900-01-01 00:08:03.400 | 9.33 |
| **...** | ... | ... | ... | ... | ... | ... |
| **9995** | 16996 | 12996 | Simran Kaur | 1900-01-01 00:46:03.400 | 1900-01-01 00:59:03.400 | 8.07 |
| **9996** | 16997 | 12997 | Rohit Sharma | 1900-01-01 00:36:03.400 | 1900-01-01 00:54:03.400 | 3.61 |
| **9997** | 16998 | 12998 | Karan Mehta | 1900-01-01 00:45:03.400 | 1900-01-01 00:55:03.400 | 5.83 |
| **9998** | 16999 | 12999 | Amit Singh | 1900-01-01 00:00:03.400 | 1900-01-01 00:08:03.400 | 7.94 |
| **9999** | 17000 | 13000 | Karan Mehta | 1900-01-01 00:22:03.400 | 1900-01-01 00:30:03.400 | 8.69 |

10000 rows × 6 columns

[26]:
```
delivery.dtypes
```

[26]:
```
DeliveryID                  int64
OrderID                     int64
DeliveryPartnerName        object
PickupTime         datetime64[ns]
DeliveryEndTime    datetime64[ns]
DistanceTraveled          float64
dtype: object
```

# Importing inventory table

[27]:
```
inventory = pd.read_csv("Inventory.csv")
inventory
```

[27]:

| | InventoryID | StoreID | ProductID | StockQuantity | ReorderLevel | LastUpdated |
|---|---|---|---|---|---|---|
| **0** | 20001000 | 2000 | 1000 | 379 | 100 | 2025-02-19 08:46:07.994229 |
| **1** | 20011000 | 2001 | 1000 | 886 | 100 | 2025-02-19 08:46:07.994299 |
| **2** | 20021000 | 2002 | 1000 | 276 | 100 | 2025-02-19 08:46:07.994351 |
| **3** | 20031000 | 2003 | 1000 | 675 | 100 | 2025-02-18 08:46:07.994402 |
| **4** | 20041000 | 2004 | 1000 | 156 | 100 | 2025-02-20 08:46:07.994455 |
| **...** | ... | ... | ... | ... | ... | ... |
| **5095** | 20951050 | 2095 | 1050 | 570 | 100 | 2025-02-18 08:46:08.307134 |
| **5096** | 20961050 | 2096 | 1050 | 606 | 100 | 2025-02-17 08:46:08.307193 |
| **5097** | 20971050 | 2097 | 1050 | 685 | 100 | 2025-02-17 08:46:08.307262 |
| **5098** | 20981050 | 2098 | 1050 | 96 | 100 | 2025-02-17 08:46:08.307338 |
| **5099** | 20991050 | 2099 | 1050 | 362 | 100 | 2025-02-18 08:46:08.307423 |

5100 rows × 6 columns

[28]:
```
inventory.dtypes
```

[28]:
```
InventoryID       int64
StoreID           int64
ProductID         int64
StockQuantity     int64
ReorderLevel      int64
LastUpdated      object
dtype: object
```

[29]:
```
inventory["LastUpdated"] = pd.to_datetime(inventory["LastUpdated"], format="%Y-%m-%d %H:%M:%S.%f", errors="coerce")
inventory
```

[29]:

| | InventoryID | StoreID | ProductID | StockQuantity | ReorderLevel | LastUpdated |
|---|---|---|---|---|---|---|
| **0** | 20001000 | 2000 | 1000 | 379 | 100 | 2025-02-19 08:46:07.994229 |
| **1** | 20011000 | 2001 | 1000 | 886 | 100 | 2025-02-19 08:46:07.994299 |
| **2** | 20021000 | 2002 | 1000 | 276 | 100 | 2025-02-19 08:46:07.994351 |
| **3** | 20031000 | 2003 | 1000 | 675 | 100 | 2025-02-18 08:46:07.994402 |
| **4** | 20041000 | 2004 | 1000 | 156 | 100 | 2025-02-20 08:46:07.994455 |

| | | | | | | |
|---|---|---|---|---|---|---|
| **5095** | 20951050 | 2095 | 1050 | 570 | 100 | 2025-02-18 08:46:08.307134 |
| **5096** | 20961050 | 2096 | 1050 | 606 | 100 | 2025-02-17 08:46:08.307193 |
| **5097** | 20971050 | 2097 | 1050 | 685 | 100 | 2025-02-17 08:46:08.307262 |
| **5098** | 20981050 | 2098 | 1050 | 96 | 100 | 2025-02-17 08:46:08.307338 |
| **5099** | 20991050 | 2099 | 1050 | 362 | 100 | 2025-02-18 08:46:08.307423 |

5100 rows × 6 columns

[30]:
```
inventory.dtypes
```

[30]:
```
InventoryID              int64
StoreID                  int64
ProductID                int64
StockQuantity            int64
ReorderLevel             int64
LastUpdated      datetime64[ns]
dtype: object
```

[31]:
```
inventory.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5100 entries, 0 to 5099
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   InventoryID    5100 non-null   int64
 1   StoreID        5100 non-null   int64
 2   ProductID      5100 non-null   int64
 3   StockQuantity  5100 non-null   int64
 4   ReorderLevel   5100 non-null   int64
 5   LastUpdated    5100 non-null   datetime64[ns]
dtypes: datetime64[ns](1), int64(5)
memory usage: 239.2 KB
```

[32]:
```
inventory.isnull().sum()
```

[32]:
```
InventoryID      0
StoreID          0
ProductID        0
StockQuantity    0
ReorderLevel     0
LastUpdated      0
dtype: int64
```

[33]:
```
inventory.describe()
```

[33]:

| | InventoryID | StoreID | ProductID | StockQuantity | ReorderLevel | LastUpdated |
|---|---|---|---|---|---|---|
| **count** | 5.100000e+03 | 5100.0000 | 5100.000000 | 5100.000000 | 5100.0 | 5100 |
| **mean** | 2.049602e+07 | 2049.5000 | 1025.000000 | 526.213137 | 100.0 | 2025-02-18 20:20:09.554430720 |
| **min** | 2.000100e+07 | 2000.0000 | 1000.000000 | 50.000000 | 100.0 | 2025-02-17 08:46:07.994506 |
| **25%** | 2.024851e+07 | 2024.7500 | 1012.000000 | 293.000000 | 100.0 | 2025-02-18 08:46:07.996550400 |
| **50%** | 2.049602e+07 | 2049.5000 | 1025.000000 | 529.000000 | 100.0 | 2025-02-18 08:46:08.291908608 |
| **75%** | 2.074354e+07 | 2074.2500 | 1038.000000 | 763.000000 | 100.0 | 2025-02-19 08:46:08.299325440 |
| **max** | 2.099105e+07 | 2099.0000 | 1050.000000 | 1000.000000 | 100.0 | 2025-02-20 08:46:08.306989 |
| **std** | 2.886890e+05 | 28.8689 | 14.721045 | 272.976504 | 0.0 | NaN |

# Importing products table

[34]:
```
products = pd.read_csv('Products.csv')
products
```

[34]:

| | ProductID | ProductName | Category | Brand | PricePerUnit | StockQuantity |
|---|---|---|---|---|---|---|
| **0** | 1000 | Amul Taaza Milk 1L | Dairy | Amul | 72 | 564 |
| **1** | 1001 | Amul Butter 500g | Dairy | Amul | 260 | 409 |
| **2** | 1002 | Mother Dairy Curd 400g | Dairy | Mother Dairy | 40 | 485 |
| **3** | 1003 | Aashirvaad Atta 5kg | Grocery | Aashirvaad | 255 | 236 |
| **4** | 1004 | Tata Salt 1kg | Grocery | Tata | 28 | 539 |
| **5** | 1005 | India Gate Basmati Rice 1kg | Grocery | India Gate | 180 | 226 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 5 | 1005 | India Gate Basmati Rice 1kg | Grocery | India Gate | 160 | 220 |
| 6 | 1006 | Maggi Noodles 70g | Snacks | Maggi | 14 | 936 |
| 7 | 1007 | Lays Classic Salted 52g | Snacks | Lays | 20 | 951 |
| 8 | 1008 | Britannia Good Day Cashew 600g | Snacks | Britannia | 150 | 262 |
| 9 | 1009 | Farm Fresh Tomatoes 500g | Vegetables | Farm Fresh | 35 | 622 |
| 10 | 1010 | Golden Apples 1kg | Fruits | Farm Fresh | 180 | 352 |
| 11 | 1011 | Bananas (12 pcs) | Fruits | Farm Fresh | 60 | 886 |
| 12 | 1012 | Saffola Gold Oil 1L | Grocery | Saffola | 180 | 307 |
| 13 | 1013 | Kellogg's Corn Flakes 500g | Grocery | Kellogg's | 210 | 519 |
| 14 | 1014 | Dettol Handwash 750ml | Personal Care | Dettol | 145 | 990 |
| 15 | 1015 | Colgate Toothpaste 200g | Personal Care | Colgate | 115 | 286 |
| 16 | 1016 | Nescafe Classic Coffee 100g | Beverages | Nescafe | 150 | 149 |
| 17 | 1017 | Tata Tea Premium 500g | Beverages | Tata | 240 | 999 |
| 18 | 1018 | Surf Excel Matic 1kg | Grocery | Surf Excel | 190 | 367 |
| 19 | 1019 | Dabur Honey 500g | Grocery | Dabur | 199 | 198 |
| 20 | 1020 | Cadbury Dairy Milk Silk 150g | Snacks | Cadbury | 160 | 860 |
| 21 | 1021 | Patanjali Ghee 1L | Grocery | Patanjali | 599 | 583 |
| 22 | 1022 | Fortune Sunflower Oil 1L | Grocery | Fortune | 160 | 402 |
| 23 | 1023 | Haldiram's Bhujia 400g | Snacks | Haldiram's | 135 | 275 |
| 24 | 1024 | Bournvita 500g | Beverages | Cadbury | 225 | 719 |
| 25 | 1025 | Dove Soap 125g | Personal Care | Dove | 45 | 759 |
| 26 | 1026 | Gillette Mach3 Razor | Personal Care | Gillette | 299 | 847 |
| 27 | 1027 | Real Mixed Fruit Juice 1L | Beverages | Real | 99 | 286 |
| 28 | 1028 | Catch Black Pepper 100g | Grocery | Catch | 150 | 325 |
| 29 | 1029 | Horlicks Classic Malt 1kg | Beverages | Horlicks | 385 | 879 |
| 30 | 1030 | Johnson's Baby Powder 400g | Personal Care | Johnson's | 175 | 953 |
| 31 | 1031 | Himalaya Face Wash 100ml | Personal Care | Himalaya | 130 | 822 |
| 32 | 1032 | Whisper Ultra Pads (30 pcs) | Personal Care | Whisper | 320 | 561 |
| 33 | 1033 | Oreo Chocolate Biscuits 300g | Snacks | Oreo | 85 | 898 |
| 34 | 1034 | Lizol Floor Cleaner 1L | Grocery | Lizol | 210 | 427 |
| 35 | 1035 | Sprite 2L | Beverages | Sprite | 90 | 925 |
| 36 | 1036 | Frooti Mango Drink 1L | Beverages | Frooti | 65 | 946 |
| 37 | 1037 | Dabur Chyawanprash 1kg | Health | Dabur | 350 | 612 |
| 38 | 1038 | Coca-Cola 2L | Beverages | Coca-Cola | 90 | 789 |
| 39 | 1039 | Eveready AA Batteries (4 pcs) | Electronics | Eveready | 110 | 100 |
| 40 | 1040 | Sensodyne Toothpaste 100g | Personal Care | Sensodyne | 160 | 566 |
| 41 | 1041 | Nestle Everyday Milk Powder 500g | Dairy | Nestle | 275 | 977 |
| 42 | 1042 | Kissan Mixed Fruit Jam 500g | Grocery | Kissan | 180 | 819 |
| 43 | 1043 | Good Day Butter Cookies 600g | Snacks | Britannia | 140 | 113 |
| 44 | 1044 | Brooke Bond Red Label 500g | Beverages | Brooke Bond | 170 | 447 |
| 45 | 1045 | Parle-G Biscuits 800g | Snacks | Parle | 80 | 146 |
| 46 | 1046 | Sprite 2L | Beverages | Sprite | 90 | 338 |
| 47 | 1047 | Head & Shoulders Shampoo 650ml | Personal Care | Head & Shoulders | 385 | 635 |
| 48 | 1048 | Real Orange Juice 1L | Beverages | Real | 110 | 159 |
| 49 | 1049 | Gillette Shaving Foam 200ml | Personal Care | Gillette | 240 | 237 |
| 50 | 1050 | Dabur Red Toothpaste 200g | Personal Care | Dabur | 99 | 446 |

[35]:
```
products.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51 entries, 0 to 50
```

```
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   ProductID      51 non-null     int64
 1   ProductName    51 non-null     object
 2   Category       51 non-null     object
 3   Brand          51 non-null     object
 4   PricePerUnit   51 non-null     int64
 5   StockQuantity  51 non-null     int64
dtypes: int64(3), object(3)
memory usage: 2.5+ KB
```

[36]:
```
products.dtypes
```

[36]:
```
ProductID         int64
ProductName      object
Category         object
Brand            object
PricePerUnit      int64
StockQuantity     int64
dtype: object
```

[37]:
```
products.isnull().sum()
```

[37]:
```
ProductID        0
ProductName      0
Category         0
Brand            0
PricePerUnit     0
StockQuantity    0
dtype: int64
```

[38]:
```
products.describe()
```

[38]:

| | ProductID | PricePerUnit | StockQuantity |
|---|---|---|---|
| **count** | 51.000000 | 51.000000 | 51.000000 |
| **mean** | 1025.000000 | 167.333333 | 551.647059 |
| **std** | 14.866069 | 109.806314 | 285.374899 |
| **min** | 1000.000000 | 14.000000 | 100.000000 |
| **25%** | 1012.500000 | 90.000000 | 296.500000 |
| **50%** | 1025.000000 | 150.000000 | 539.000000 |
| **75%** | 1037.500000 | 210.000000 | 834.500000 |
| **max** | 1050.000000 | 599.000000 | 999.000000 |

# Importing Returns table

[39]:
```
returns = pd.read_csv('Returns.csv')
returns
```

[39]:

| | ReturnID | OrderID | ProductID | CustomerID | StoreID | ReturnReason | ReturnDate | RefundAmount | ReturnStatus |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 17794 | 12794 | 1011 | 15794 | 2075 | Stale Product | 2025-02-12 08:33:03.428463 | 300 | Processed |
| **1** | 13898 | 8898 | 1030 | 11898 | 2050 | Damaged Item | 2025-01-25 20:43:03.428463 | 875 | Processed |
| **2** | 17271 | 12271 | 1012 | 15271 | 2056 | Damaged Item | 2025-02-08 22:13:03.428463 | 360 | Processed |
| **3** | 12959 | 7959 | 1028 | 10959 | 2050 | Stale Product | 2025-02-11 06:23:03.428463 | 600 | Processed |
| **4** | 13863 | 8863 | 1024 | 11863 | 2016 | Stale Product | 2025-02-22 05:21:03.428463 | 1125 | Processed |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **995** | 16028 | 11028 | 1010 | 14028 | 2004 | Stale Product | 2025-01-28 21:42:03.428463 | 360 | Processed |
| **996** | 8750 | 3750 | 1023 | 6750 | 2003 | Damaged Item | 2025-02-22 01:05:03.428463 | 675 | Processed |
| **997** | 17265 | 12265 | 1035 | 15265 | 2059 | Stale Product | 2025-02-06 18:15:03.428463 | 450 | Processed |
| **998** | 15520 | 10520 | 1044 | 13520 | 2087 | Damaged Item | 2025-02-22 18:00:03.428463 | 850 | Processed |
| **999** | 16852 | 11852 | 1028 | 14852 | 2008 | Damaged Item | 2025-01-31 14:37:03.428463 | 300 | Processed |

1000 rows × 9 columns

[40]:
```
returns.dtypes
```

[40]:
```
ReturnID         int64
OrderID          int64
```

```
ProductID        int64
CustomerID       int64
StoreID          int64
ReturnReason    object
ReturnDate      object
RefundAmount     int64
ReturnStatus    object
dtype: object
```

[41]:
```
returns['ReturnDate'] = pd.to_datetime(returns['ReturnDate'], format = '%Y-%m-%d %H:%M:%S.%f')
returns
```

[41]:

| | ReturnID | OrderID | ProductID | CustomerID | StoreID | ReturnReason | ReturnDate | RefundAmount | ReturnStatus |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 17794 | 12794 | 1011 | 15794 | 2075 | Stale Product | 2025-02-12 08:33:03.428463 | 300 | Processed |
| **1** | 13898 | 8898 | 1030 | 11898 | 2050 | Damaged Item | 2025-01-25 20:43:03.428463 | 875 | Processed |
| **2** | 17271 | 12271 | 1012 | 15271 | 2056 | Damaged Item | 2025-02-08 22:13:03.428463 | 360 | Processed |
| **3** | 12959 | 7959 | 1028 | 10959 | 2050 | Stale Product | 2025-02-11 06:23:03.428463 | 600 | Processed |
| **4** | 13863 | 8863 | 1024 | 11863 | 2016 | Stale Product | 2025-02-22 05:21:03.428463 | 1125 | Processed |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **995** | 16028 | 11028 | 1010 | 14028 | 2004 | Stale Product | 2025-01-28 21:42:03.428463 | 360 | Processed |
| **996** | 8750 | 3750 | 1023 | 6750 | 2003 | Damaged Item | 2025-02-22 01:05:03.428463 | 675 | Processed |
| **997** | 17265 | 12265 | 1035 | 15265 | 2059 | Stale Product | 2025-02-06 18:15:03.428463 | 450 | Processed |
| **998** | 15520 | 10520 | 1044 | 13520 | 2087 | Damaged Item | 2025-02-22 18:00:03.428463 | 850 | Processed |
| **999** | 16852 | 11852 | 1028 | 14852 | 2008 | Damaged Item | 2025-01-31 14:37:03.428463 | 300 | Processed |

1000 rows × 9 columns

[42]:
```
returns.dtypes
```

[42]:
```
ReturnID                int64
OrderID                 int64
ProductID               int64
CustomerID              int64
StoreID                 int64
ReturnReason           object
ReturnDate      datetime64[ns]
RefundAmount            int64
ReturnStatus           object
dtype: object
```

[43]:
```
returns.isnull().sum()
```

[43]:
```
ReturnID        0
OrderID         0
ProductID       0
CustomerID      0
StoreID         0
ReturnReason    0
ReturnDate      0
RefundAmount    0
ReturnStatus    0
dtype: int64
```

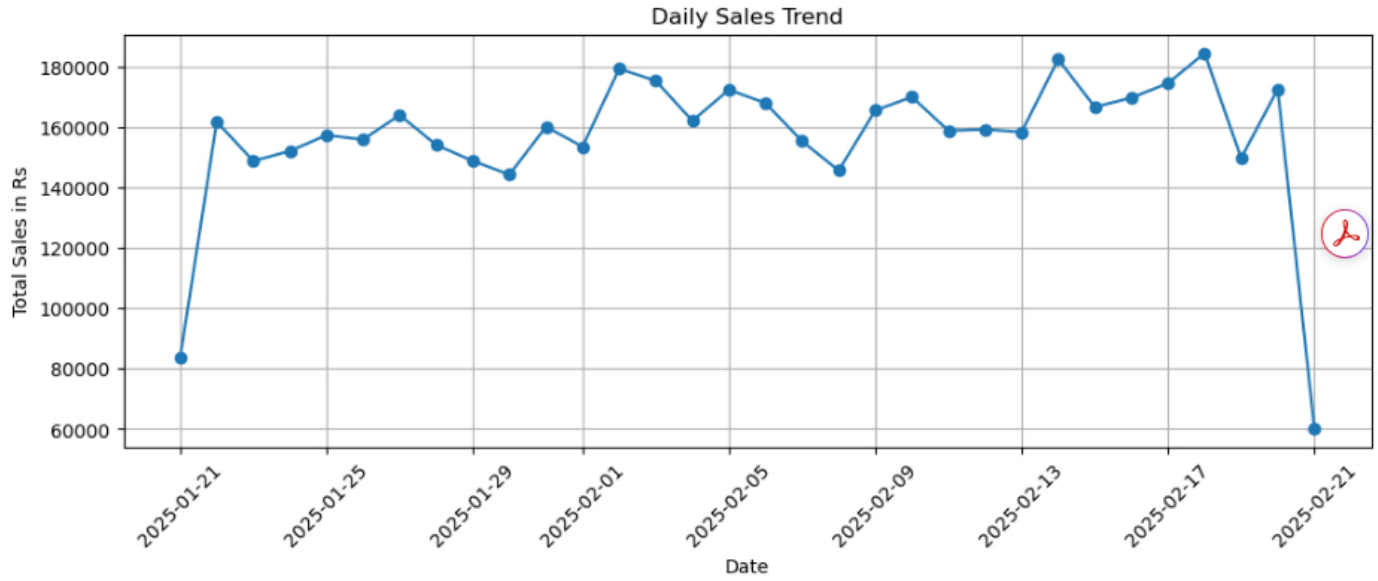
[44]:
```
returns.describe()
```

[44]:

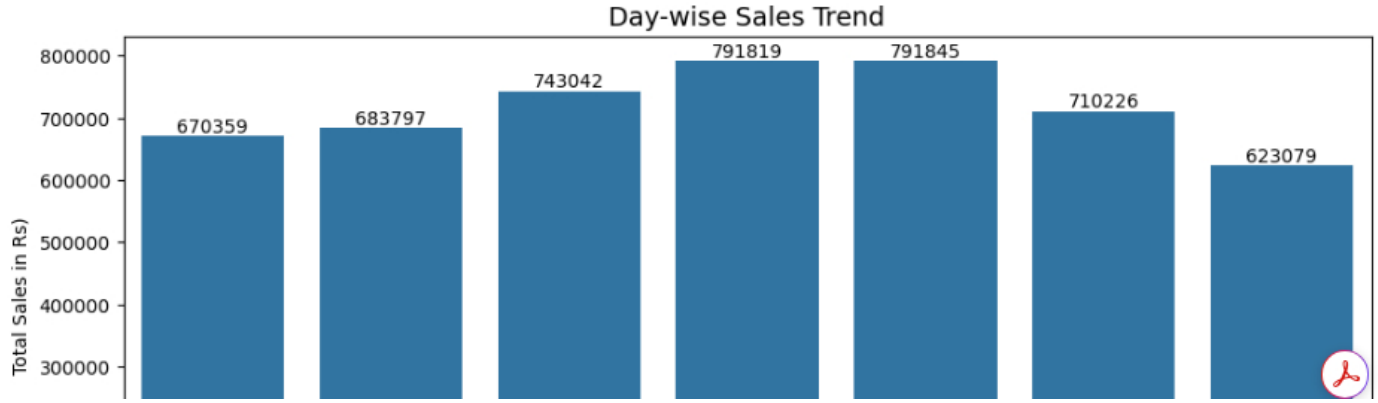
| | ReturnID | OrderID | ProductID | CustomerID | StoreID | ReturnDate | RefundAmount |
|---|---|---|---|---|---|---|---|
| **count** | 1000.000000 | 1000.000000 | 1000.000000 | 1000.000000 | 1000.000000 | 1000 | 1000.000000 |
| **mean** | 13109.208000 | 8109.208000 | 1024.361000 | 11109.208000 | 2047.924000 | 2025-02-06 22:12:09.668462848 | 479.203000 |
| **min** | 8008.000000 | 3008.000000 | 1000.000000 | 6008.000000 | 2000.000000 | 2025-01-22 02:08:03.428463 | 14.000000 |
| **25%** | 10633.500000 | 5633.500000 | 1011.000000 | 8633.500000 | 2021.000000 | 2025-01-29 23:54:03.428463104 | 180.000000 |
| **50%** | 13029.500000 | 8029.500000 | 1024.000000 | 11029.500000 | 2048.000000 | 2025-02-07 03:07:03.428463104 | 360.000000 |
| **75%** | 15723.750000 | 10723.750000 | 1037.000000 | 13723.750000 | 2073.000000 | 2025-02-13 22:10:48.428463104 | 675.000000 |
| **max** | 17987.000000 | 12987.000000 | 1050.000000 | 15987.000000 | 2099.000000 | 2025-02-23 03:04:03.428463 | 2995.000000 |
| **std** | 2901.123785 | 2901.123785 | 14.731113 | 2901.123785 | 29.178273 | NaN | 419.752689 |

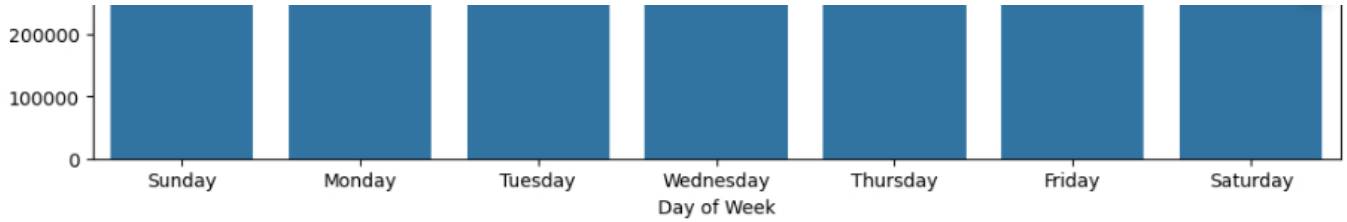
## What is the sales trend over time (day/week/month)?

[45]:
```
# Group by date (ignoring time part)
daily_sales = orders.groupby(orders['OrderDateTime'].dt.date)['TotalAmount'].sum().reset_index()

# Rename columns for clarity
daily_sales.columns = ['Date', 'TotalSales']

#plot line chart
plt.figure(figsize = (12,4))
plt.plot(daily_sales['Date'], daily_sales['TotalSales'], marker = 'o')
plt.title('Daily Sales Trend')
plt.xlabel('Date')
plt.ylabel('Total Sales in Rs')
plt.xticks(rotation = 45)
plt.grid(True)
plt.show()
```



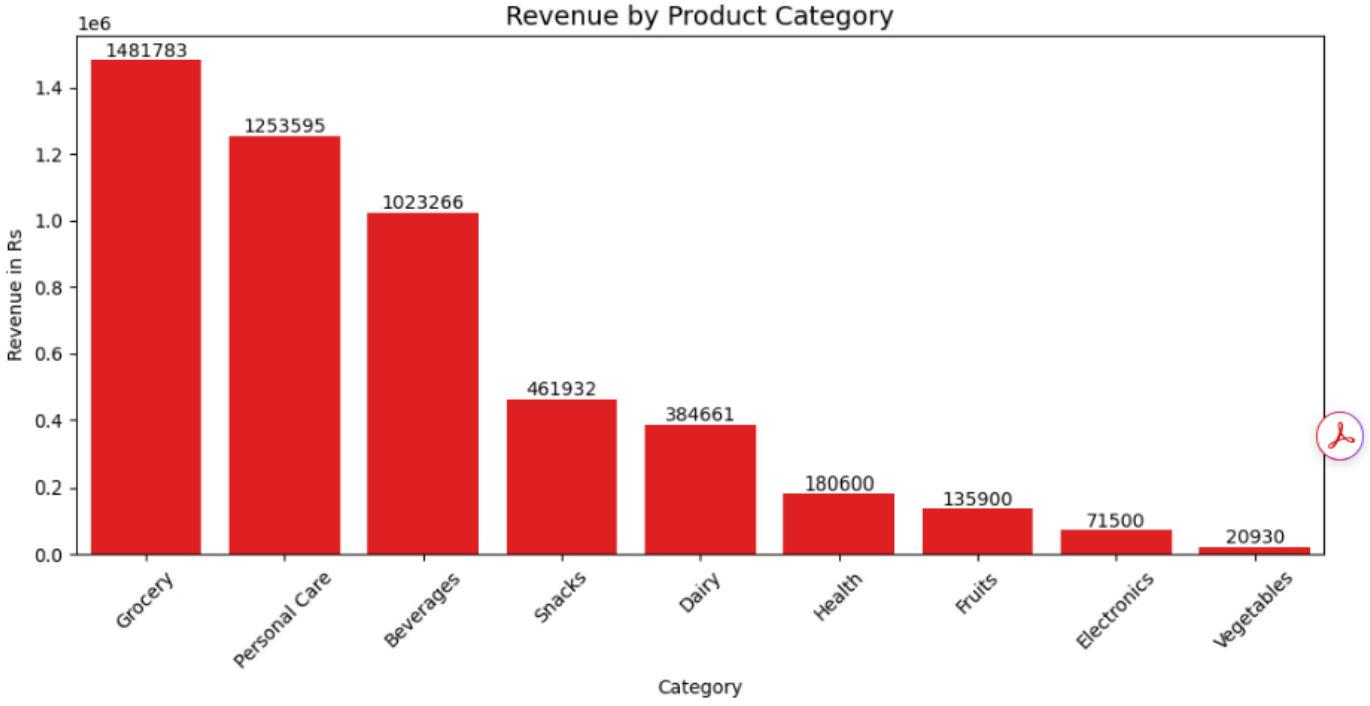

[46]:
```
# Group sales by day of week
daily_sales = orders.groupby(orders['OrderDateTime'].dt.day_name())['TotalAmount'].sum().reset_index()

# Rename columns
daily_sales.columns = ['Day', 'TotalSales']

# Define correct order for days
day_order = ['Sunday','Monday','Tuesday','Wednesday','Thursday','Friday','Saturday']
daily_sales['Day'] = pd.Categorical(daily_sales['Day'], categories=day_order, ordered=True)
daily_sales = daily_sales.sort_values('Day')

# Plot using seaborn
plt.figure(figsize=(12,5))
ax = sns.barplot(x='Day', y='TotalSales', data=daily_sales)

# Add labels on bars
for container in ax.containers:
    ax.bar_label(container, fmt='%d', label_type='edge')

# Chart formatting
plt.title('Day-wise Sales Trend', fontsize=14)
plt.xlabel('Day of Week')
plt.ylabel('Total Sales in Rs)')
plt.show()
```






## Which product categories generate the most revenue?

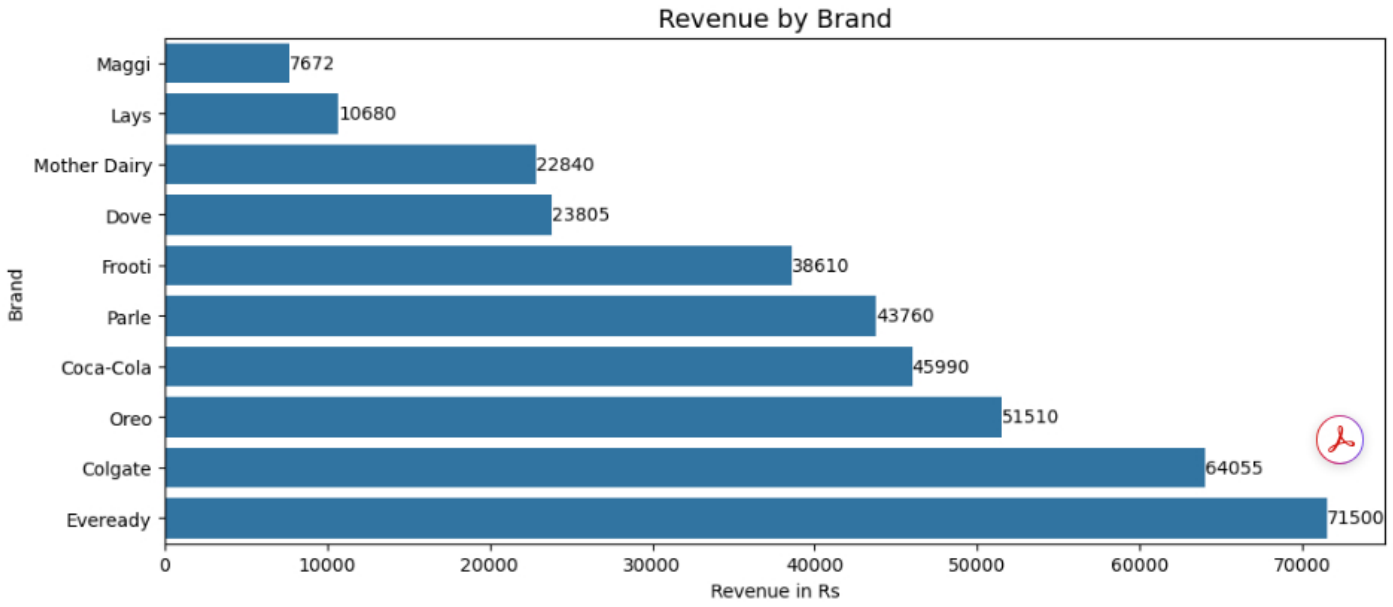
[47]:
```
# Group by category and calculate revenue
category_sales = orders.merge(products, on="ProductID") \
                       .groupby("Category")["TotalAmount"].sum() \
                       .reset_index()

# Sort by revenue for better visuals
category_sales = category_sales.sort_values("TotalAmount", ascending=False)

# Plot
plt.figure(figsize=(12,5))
ax = sns.barplot(x="Category", y="TotalAmount", data=category_sales, color="red")

# Add labels on bars
for container in ax.containers:
    ax.bar_label(container, fmt="%d", fontsize=10)

# Formatting
plt.title("Revenue by Product Category", fontsize=14)
plt.xlabel("Category")
plt.ylabel("Revenue in Rs")
plt.xticks(rotation=45)
plt.show()
```




## Which brands/products are the top sellers?

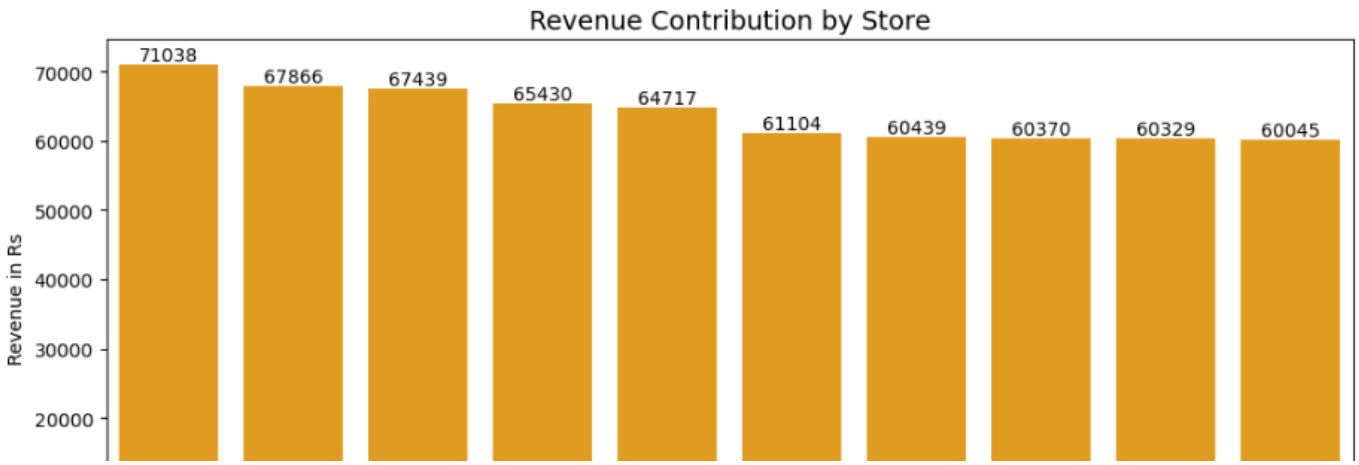
[48]:
```
# Group by category and calculate revenue
brand_sales = products.merge(orders, on= 'ProductID')\
                      .groupby('Brand')['TotalAmount'].sum()\
                      .reset_index()

# Sort by revenue for better visuals
brand_sales = brand_sales.sort_values('TotalAmount').head(10)

#plot
plt.figure(figsize=(12,5))
ax = sns.barplot(x='TotalAmount', y='Brand', data = brand_sales, orient='h')
```

```
# Add labels on bars
for container in ax.containers:
    ax.bar_label(container, fmt="%d")

# Formatting
plt.title("Revenue by Brand", fontsize=14)
plt.xlabel("Revenue in Rs")
plt.ylabel("Brand")
plt.show()
```



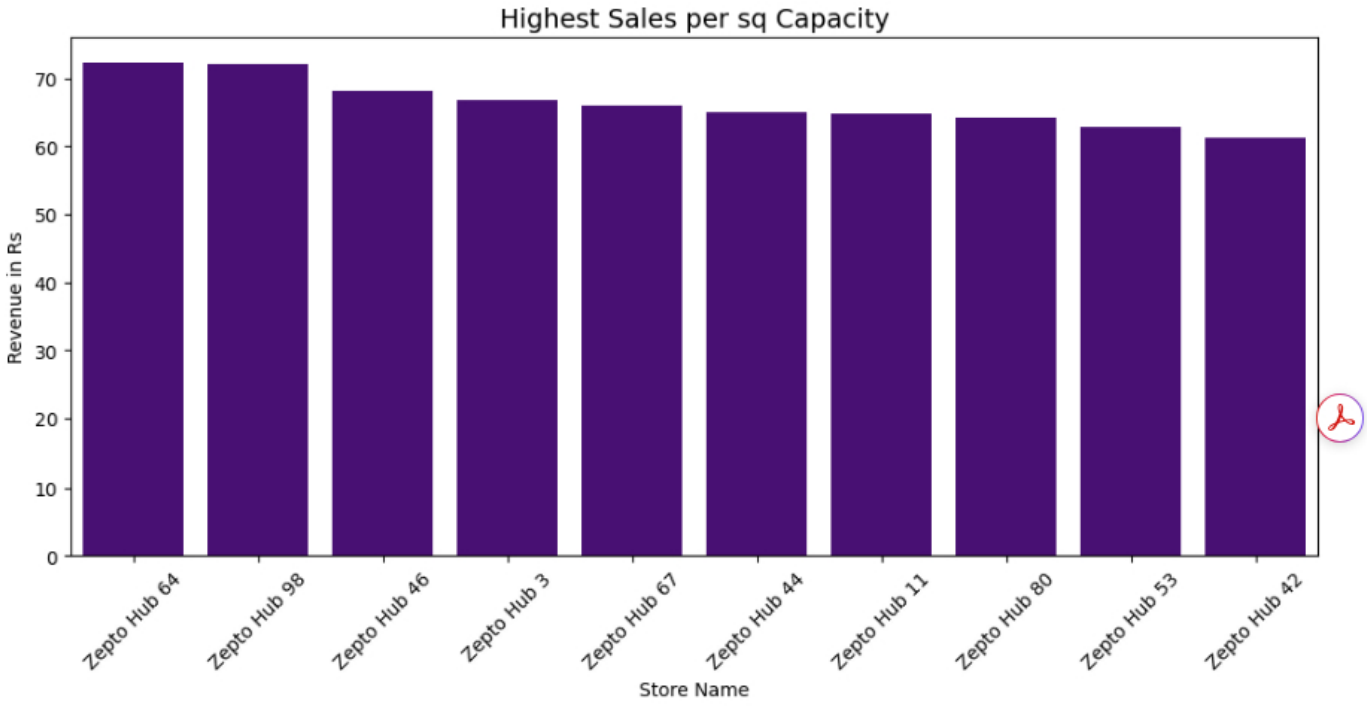

## Revenue contribution by each store/dark warehouse

[49]:
```
# Group by category and calculate revenue
store_sales = dw.merge(orders, on='StoreID')\
                .groupby('StoreName')['TotalAmount'].sum() \
                .reset_index()

# Sort for better visuals
store_sales = store_sales.sort_values('TotalAmount', ascending = False).head(10)

#plot
plt.figure(figsize=(12,5))
ax = sns.barplot(x='StoreName', y='TotalAmount', data = store_sales, color ='orange')

# Add labels on bars
for container in ax.containers:
    ax.bar_label(container, fmt="%d")

# Formatting
plt.title("Revenue Contribution by Store", fontsize=14)
plt.xlabel("Store Name")
plt.ylabel("Revenue in Rs")
plt.xticks(rotation=45)
plt.show()
```




## Which store has the highest sales per square capacity?

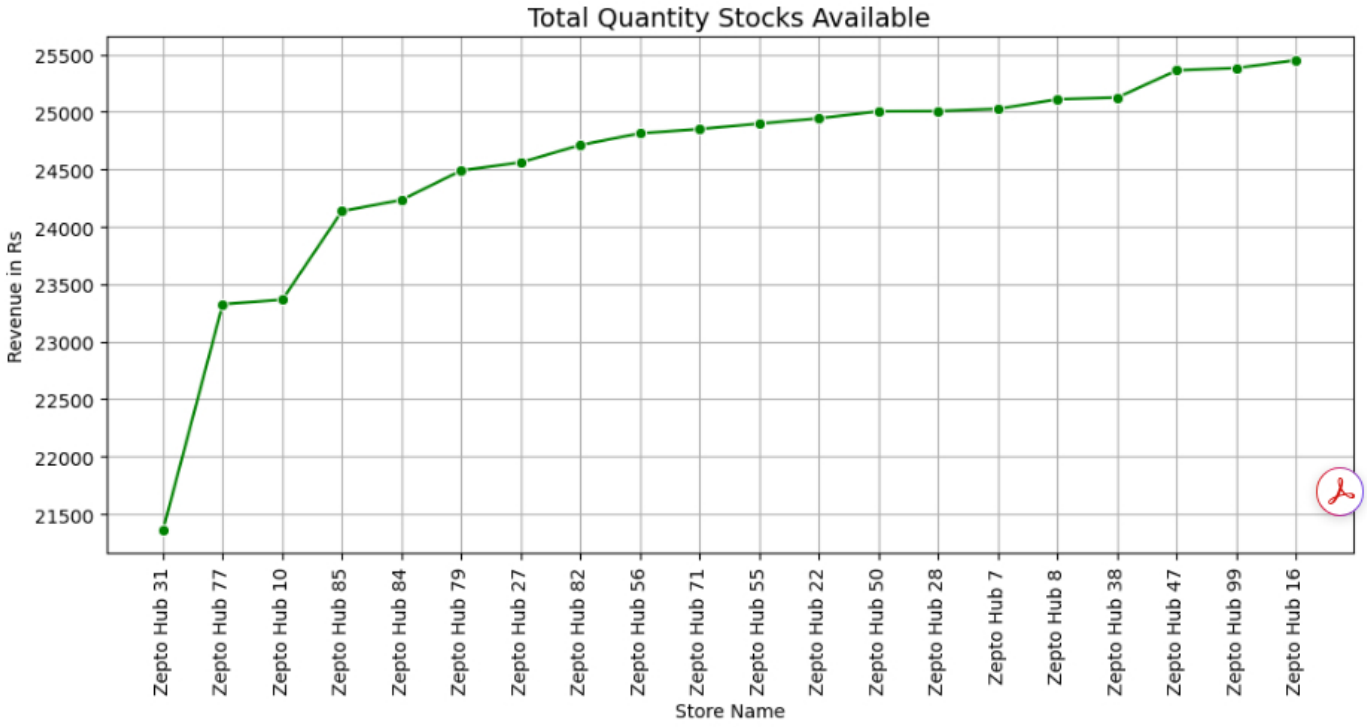
[50]:
```
# Group by category and calculate revenue
store_capacity = dw.merge(orders, on='StoreID')\
                   .groupby(['StoreName', 'StoreCapacity'])['TotalAmount'].sum()\
                   .reset_index()

# Calculate revenue per capacity
store_capacity["RevenuePerCapacity"] = store_capacity["TotalAmount"] / store_capacity["StoreCapacity"]

# Sort by ratio and take top 10
store_capacity = store_capacity.sort_values("RevenuePerCapacity", ascending=False).head(10)

#plot
plt.figure(figsize =(12,5))
ax = sns.barplot(x = 'StoreName', y= 'RevenuePerCapacity', data = store_capacity, color = 'indigo')

# Add labels on bars
for conatiner in ax.containers:
    ax.bar_label(container, fmt="%.2f")

# Formatting
plt.title("Highest Sales per sq Capacity", fontsize=14)
plt.xlabel("Store Name")
plt.ylabel("Revenue in Rs")
plt.xticks(rotation=45)
plt.show()
```




## Total stock quantity available per store

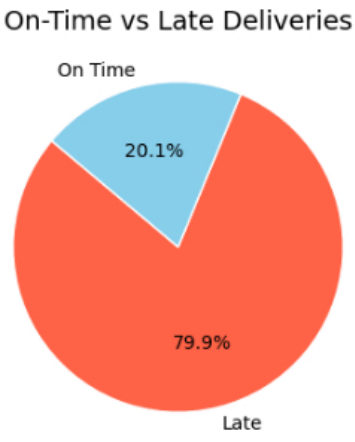
[51]:
```
# Group by category and calculate revenue
store = inventory.merge(dw, on='StoreID')\
                 .groupby('StoreName')['StockQuantity'].sum().reset_index()

#sort
store = store.sort_values('StockQuantity').head(20)
```

```
#plot
plt.figure(figsize =(12,5))
sns.lineplot(data=store, x="StoreName", y="StockQuantity", marker ='o', color = 'green')

# Formatting
plt.title("Total Quantity Stocks Available", fontsize=14)
plt.xlabel("Store Name")
plt.ylabel("Revenue in Rs")
plt.xticks(rotation=90)
plt.grid()
plt.show()
```



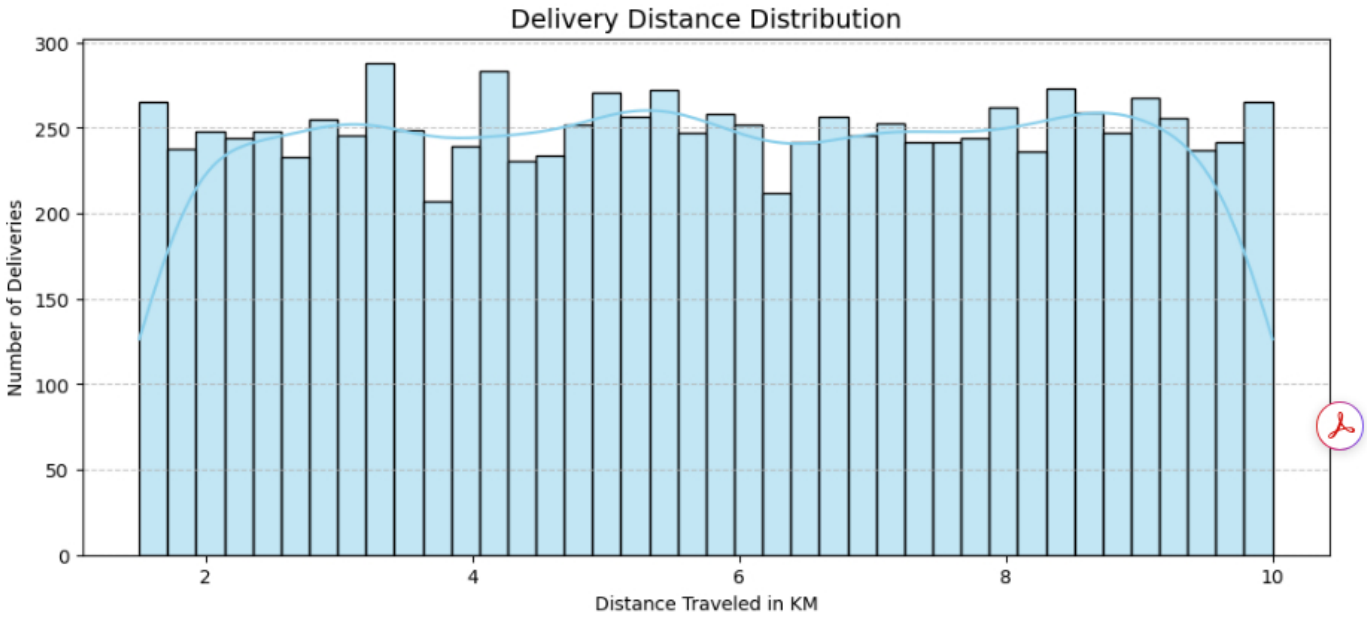

## On-time vs Late Deliveries (overall)

[71]:
```
# Classify deliveries as On-time or Late
orders["DeliveryStatus"] = orders.apply(
    lambda x: "Late" if x["ActualDeliveryTime"] > x["ExpectedDeliveryTime"] else "On Time",
    axis=1
)

# Count status
delivery_status = orders["DeliveryStatus"].value_counts()

# Plot pie chart
plt.figure(figsize=(4,4))
plt.pie(delivery_status, labels=delivery_status.index, autopct="%1.1f%%",
        colors=["tomato", "skyblue"], startangle=140, wedgeprops={'edgecolor':'white'})

plt.title("On-Time vs Late Deliveries", fontsize=14)
plt.show()
```



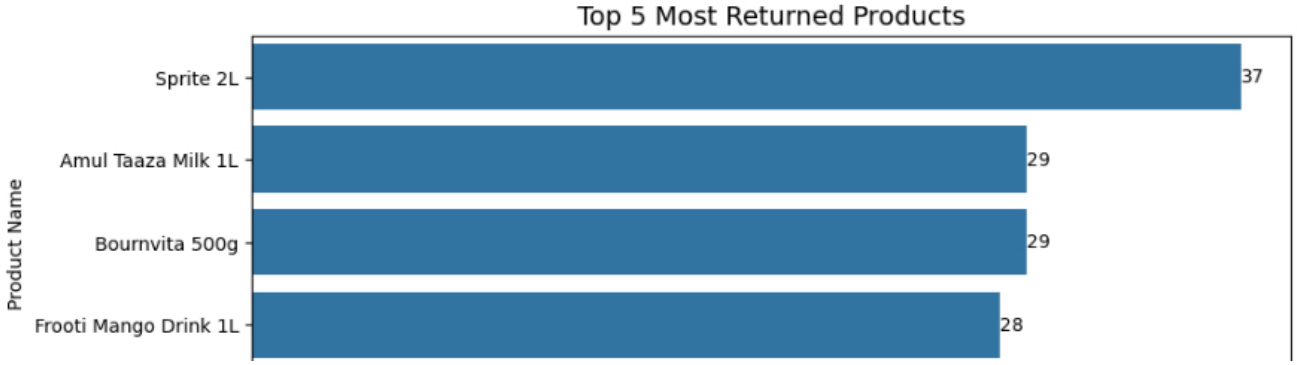
## Delivery distance distribution

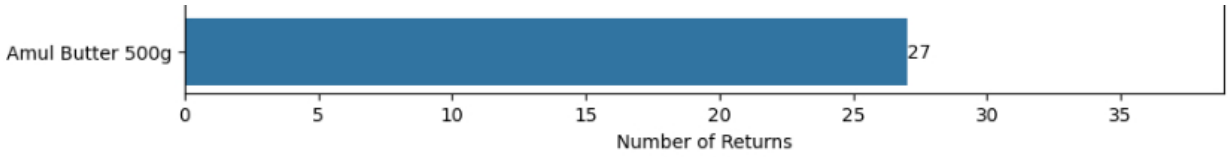
[69]:
```
# Plot histogram
plt.figure(figsize=(12,5))
sns.histplot(delivery["DistanceTraveled"], bins=40, kde=True, color="skyblue")

# Formatting
plt.title("Delivery Distance Distribution", fontsize=14)
plt.xlabel("Distance Traveled in KM")
plt.ylabel("Number of Deliveries")
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()
```




## Top 5 most returned products.

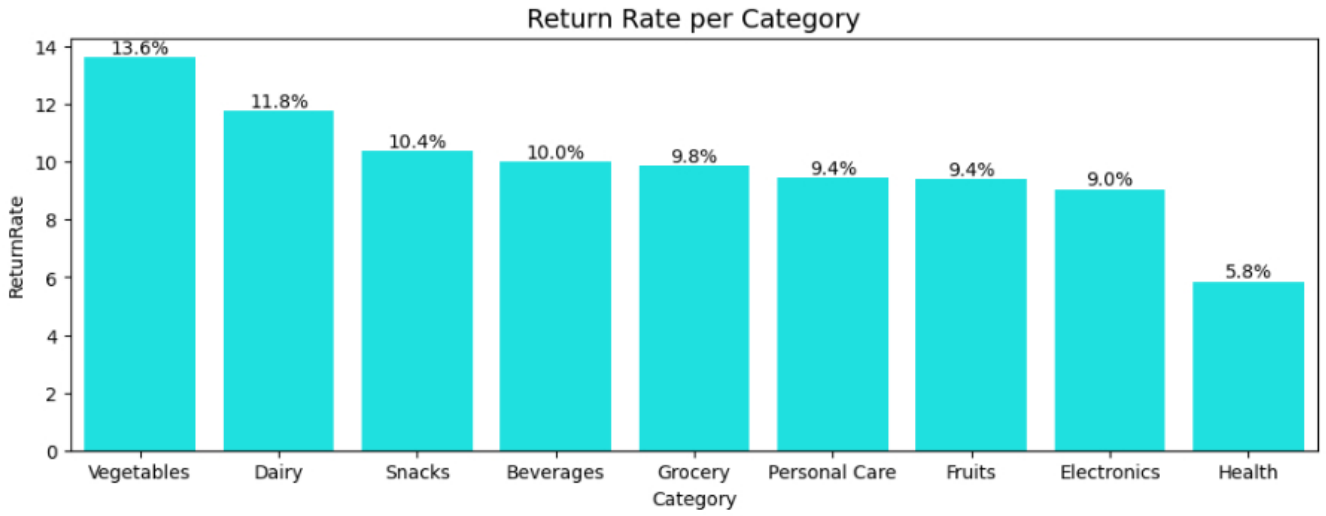
[82]:
```
# Merge products with returns
product_returns = returns.merge(products, on="ProductID")\
                         .groupby('ProductName')['ReturnID'].count().reset_index()

# Sort and take top 5
top_returns = top_returns.sort_values("ReturnID", ascending=False).head(5)

# Plot
plt.figure(figsize=(10,4))
ax = sns.barplot(x="ReturnID", y="ProductName", data=top_returns)

# Add labels on bars
for container in ax.containers:
    ax.bar_label(container, fmt='%d')

# Formatting
plt.title("Top 5 Most Returned Products", fontsize=14)
plt.xlabel("Number of Returns")
plt.ylabel("Product Name")
plt.show()
```






## Return rate per category

[104]:
```
# Merge returns with products & orders with products to get category
returns_products = returns.merge(products, on = 'ProductID')

orders_products = orders.merge(products, on = 'ProductID')

# Total returns & orders per category
returns_per_category = returns_products.groupby('Category')['ReturnID'].count().reset_index(name="TotalReturns")
orders_per_category = orders_products.groupby('Category')['OrderID'].count().reset_index(name="TotalOrders")

# Merge and calculate return rate %
return_rate = returns_per_category.merge(orders_per_category, on="Category")
return_rate["ReturnRate"] = (return_rate["TotalReturns"] / return_rate["TotalOrders"]) * 100

#sort
return_rate = return_rate.sort_values("ReturnRate", ascending = False)

#plot
plt.figure(figsize= (12,4))
ax = sns.barplot(data = return_rate, x ='Category', y='ReturnRate', color='cyan')

#labels
for container in ax.containers:
    ax.bar_label(container, fmt='%.1f%%')

#Formatting
plt.title("Return Rate per Category", fontsize=14)
plt.show()
```




[ ]: