TITLE: SUPERSTORE DATA ANALYSIS

AUTHOR: FALAK ZOYA KHAN

Introduction: The Superstore dataset is used to analyze sales, profits, and other key business metrics to derive insights for improving business strategies.

Objective: To explore the Superstore dataset, perform data analysis, and identify trends to optimize sales and profitability.

Data Exploration: The dataset is explored to understand its structure, identify missing values, and analyze relationships between total_sales, price_per_unit, and quantity.

Data Preprocessing: Necessary steps include cleaning the data, handling missing values, encoding categorical variable.

Data Visualization: visualizing the data using bar charts, pie chart, boxplot and histogram, to uncover patterns, distribution and relationships in the dataset.

Model Building: Implementing Linear Regression and Random Forest Regressor models to analyze patterns and make predictions based on the dataset.

Model Evaluation: Models are evaluated using accuracy, RMSE (Root Mean Square Error), and r2 metrics to assess prediction performance.

Results: The best-performing model is selected, and conclusions are drawn about the accuracy and efficiency of different models.

Tools Used: Python libraries like pandas, matplotlib, sckit-learn, and seaborn are used for data manipulation, visualization, and model building.

In [ ]:
```
import pandas as pd
import numpy as np
import random

# Set random seed for reproducibility
random.seed(42)
np.random.seed(42)

# Number of rows for the dataset
num_rows = 1_000_000

# Generate raw data for the dataset
countries = ['USA', 'India', 'Brazil', 'China', 'Germany', 'Australia', 'South Afri
regions = ['North', 'South', 'East', 'West', 'Central']
store_types = ['Supermarket', 'Grocery Store', 'Wholesale', 'Hypermarket', None]
categories = ['Fruits', 'Vegetables', 'Dairy', 'Meat', 'Snacks', 'Beverages', 'Hous
days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday
payment_methods = ['Cash', 'Credit Card', 'Mobile Payment', 'Debit Card', None]
bool_options = [True, False, None]

data = {
    'Transaction_ID': [f"T{random.randint(1000000, 9999999)}" for _ in range(num_ro
    'Country': [random.choice(countries) for _ in range(num_rows)],
    'Region': [random.choice(regions) for _ in range(num_rows)],
    'Store_Type': [random.choice(store_types) for _ in range(num_rows)],
    'Category': [random.choice(categories) for _ in range(num_rows)],
    'Item_ID': [f"I{random.randint(10000, 99999)}" for _ in range(num_rows)],
    'Quantity': [random.randint(1, 100) if random.random() > 0.1 else None for _ in
    'Price_per_Unit': [round(random.uniform(1.0, 500.0), 2) if random.random() > 0.
    'Total_Sales': [None for _ in range(num_rows)],  # Placeholder for raw data
    'Discount_Applied': [random.choice(bool_options) for _ in range(num_rows)],
    'Day': [random.choice(days) for _ in range(num_rows)],
    'Time_of_Purchase': [f"{random.randint(0, 23)}:{random.randint(0, 59)}" if rand
    'Payment_Method': [random.choice(payment_methods) for _ in range(num_rows)],
    'Customer_ID': [f"C{random.randint(100000, 999999)}" if random.random() > 0.2 e
    'Loyalty_Card_Used': [random.choice(bool_options) for _ in range(num_rows)],
    'Employee_ID': [f"E{random.randint(1000, 9999)}" if random.random() > 0.1 else
    'Feedback_Score': [random.randint(1, 5) if random.random() > 0.3 else None for
    'Return': [random.choice(bool_options) for _ in range(num_rows)],
    'Weather': [random.choice(['Sunny', 'Rainy', 'Snowy', 'Cloudy', None]) for _ in
    'Holiday': [random.choice(bool_options) for _ in range(num_rows)]
}

# Create a DataFrame
```

```
df = pd.DataFrame(data)

# Introduce leading/trailing spaces to some columns
df['Country'] = df['Country'].apply(lambda x: f"  {x}  " if random.random() > 0.7 e
df['Store_Type'] = df['Store_Type'].apply(lambda x: f"{x}  " if random.random() > 0
df['Category'] = df['Category'].apply(lambda x: f"  {x}" if random.random() > 0.5 e

# Add some duplicate rows
df = pd.concat([df, df.sample(frac=0.05)], ignore_index=True)

# Save to CSV
df.to_csv("raw_superstore_data.csv", index=False)
print("Dataset saved as raw_superstore_data.csv")
```

```
Dataset saved as raw_superstore_data.csv
```

step 1: LOAD THE DATASET

In [ ]:
```
df=pd.read_csv("raw_superstore_data.csv")
df.head(100000)
```

Out[ ]:

	Transaction_ID	Country	Region	Store_Type	Category	Item_ID	Quantity	Price
0	T2867825	UK	South	Supermarket	Fruits	I36836	43.0	
1	T1419610	USA	South	Wholesale	Beverages	I85457	76.0	
2	T5614226	USA	Central	None	Beverages	I28818	53.0	
3	T5108603	Canada	East	NaN	Vegetables	I98489	29.0	
4	T4744854	Brazil	West	Grocery Store	Dairy	I22306	13.0	
...	...	...	...	...	...	...	...	...
99995	T9548345	Japan	East	Supermarket	Beverages	I42779	82.0	
99996	T2667085	Canada	North	Hypermarket	Beverages	I24807	NaN	
99997	T7505908	Brazil	Central	Grocery Store	None	I35241	40.0	
99998	T8011527	Brazil	North	Hypermarket	Snacks	I94587	63.0	
99999	T2369196	Germany	South	Hypermarket	Dairy	I46380	58.0	

100000 rows × 20 columns

STEP 2: UNDERSTAND THE DATA

In [ ]:
```
# check the numbers of rows and column
print("numbers of rows: ",df.shape[0])  # no. of rows
print("numbers of columns: ",df.shape[1]) # no. of columns

# display column names
print("column names: ",df.columns)
```

```
numbers of rows:  1050000
numbers of columns:  20
column names:  Index(['Transaction_ID', 'Country', 'Region', 'Store_Type', 'Categor
y',
       'Item_ID', 'Quantity', 'Price_per_Unit', 'Total_Sales',
       'Discount_Applied', 'Day', 'Time_of_Purchase', 'Payment_Method',
       'Customer_ID', 'Loyalty_Card_Used', 'Employee_ID', 'Feedback_Score',
       'Return', 'Weather', 'Holiday'],
      dtype='object')
```

IDENTIFY MISSING VALUES

In [ ]:
```
missing_transaction_ID=df["Transaction_ID"].isnull().sum()
missing_Country=df["Country"].isnull().sum()
missing_region=df["Region"].isnull().sum()
missing_Store_Type=df["Store_Type"].isnull().sum()
missing_Category=df["Category"].isnull().sum()
missing_Item_ID=df["Item_ID"].isnull().sum()
missing_Quantity=df["Quantity"].isnull().sum()
missing_Price_per_unit=df["Price_per_Unit"].isnull().sum()
missing_Total_sales=df["Total_Sales"].isnull().sum()
missing_Discount_Applied=df["Discount_Applied"].isnull().sum()
missing_Day=df["Day"].isnull().sum()
missing_Time_of_purchase=df["Time_of_Purchase"].isnull().sum()
missing_Payment_Method=df["Payment_Method"].isnull().sum()
missing_Customer_ID=df["Customer_ID"].isnull().sum()
missing_Loyalty_Card_Used=df["Loyalty_Card_Used"].isnull().sum()
missing_Employee_ID=df["Employee_ID"].isnull().sum()
missing_Feedback_Score=df["Feedback_Score"].isnull().sum()
missing_Return=df["Return"].isnull().sum()
missing_Weather=df["Weather"].isnull().sum()
missing_Holiday=df["Holiday"].isnull().sum()

print("Missing values in Transaction_ID:", missing_transaction_ID)
print("Missing values in Country:", missing_Country)
print("Missing values in region:", missing_region)
print("Missing values in Store_Type:", missing_Store_Type )
print("Missing values in Category:", missing_Category)
print("Missing values in Item_ID:", missing_Item_ID)
print("Missing values in Quantity:", missing_Quantity)
print("Missing values in Price_per_unit:", missing_Price_per_unit)
print("Missing values in Total_sales:", missing_Total_sales)
print("Missing values in Discount_Applied:", missing_Discount_Applied)
print("Missing values in Day:", missing_Day)
print("Missing values in Time_of_purchase:", missing_Time_of_purchase)
print("Missing values in Payment_Method:", missing_Payment_Method)
print("Missing values in Customer_ID:", missing_Customer_ID)
```

```
print("Missing values in Loyalty_Card_Used:", missing_Loyalty_Card_Used)
print("Missing values in Employee_ID:", missing_Employee_ID)
print("Missing values in Feedback_Score:", missing_Feedback_Score)
print("Missing values in Return:", missing_Return)
print("Missing values in Weather:", missing_Weather)
print("Missing values in Holiday:", missing_Holiday)
```

```
Missing values in Transaction_ID: 0
Missing values in Country: 0
Missing values in region: 0
Missing values in Store_Type: 104726
Missing values in Category: 52815
Missing values in Item_ID: 0
Missing values in Quantity: 104894
Missing values in Price_per_unit: 52296
Missing values in Total_sales: 1050000
Missing values in Discount_Applied: 350216
Missing values in Day: 0
Missing values in Time_of_purchase: 105228
Missing values in Payment_Method: 209698
Missing values in Customer_ID: 209819
Missing values in Loyalty_Card_Used: 349698
Missing values in Employee_ID: 104961
Missing values in Feedback_Score: 315275
Missing values in Return: 350336
Missing values in Weather: 210194
Missing values in Holiday: 350303
```

In [ ]:
```
# check data type
data_type=df.dtypes
print("data_type of data : ",data_type)
```

```
data_type of data :  Transaction_ID        object
Country               object
Region                object
Store_Type            object
Category              object
Item_ID               object
Quantity             float64
Price_per_Unit       float64
Total_Sales          float64
Discount_Applied      object
Day                   object
Time_of_Purchase      object
Payment_Method        object
Customer_ID           object
Loyalty_Card_Used     object
Employee_ID           object
Feedback_Score       float64
Return                object
Weather               object
Holiday               object
dtype: object
```

DROP DUPLICATES

In [ ]:
```
S# rows before remove duplicate
before_rows= df.shape[0]

# remove duplicates
df=df.drop_duplicates()

# rows_after removing duplicates

after_rows = df.shape[0]

print("rows before removing duplicate = ",before_rows)
print("rows after removing duplicates = ",after_rows)
```

```
rows before removing duplicate =  1050000
rows after removing duplicates =  1000000
```

STEP 3: TRIM WHITE SPACES IN CELLS

In [ ]:
```
# TRIM WHITE SPACES IN CELLS
df["Country"] = df["Country"].str.strip()
df["Region"] = df["Region"].str.strip()
df["Store_Type"] = df["Store_Type"].str.strip()
df["Category"] = df["Category"].str.strip()
df["Item_ID"] = df["Item_ID"].str.strip()
df["Discount_Applied"] = df["Discount_Applied"].astype(str).str.strip()
df["Day"] = df["Day"].str.strip()
df["Time_of_Purchase"] = df["Time_of_Purchase"].str.strip()
df["Payment_Method"] = df["Payment_Method"].str.strip()
df["Customer_ID"] = df["Customer_ID"].str.strip()
df["Loyalty_Card_Used"] = df["Loyalty_Card_Used"].astype(str).str.strip()
df["Employee_ID"] = df["Employee_ID"].str.strip()
df["Return"] = df["Return"].astype(str).str.strip()
df["Weather"] = df["Weather"].str.strip()
df["Holiday"] = df["Holiday"].astype(str).str.strip()
print("white space remove in the column")
```

```
white space remove in the column
```

STEP4: FILL MISSING REGION BASED ON COUNTRY

In [ ]:
```
# REGION MAPPING
region_mapping = {
    'USA':'North',
    'INDIA':'CENTRAL',
    'BRAZIL':'SOUTH',
    'China':'East',
```

```
    'Germany':'West',
    'Australia':'East',
    'South Africa':'South',
    'Canada':'north',
    'UK':'West',
    'Japan':'East',
}
 # fill missing region based on country
df['Region']=df.apply(
             lambda row : region_mapping[row['Country']] if pd.isnull(row['Region']
)
print(df['Region'])
```

```
0           South
1           South
2         Central
3            East
4            West
           ...
999995      South
999996    Central
999997       West
999998       West
999999       West
Name: Region, Length: 1000000, dtype: object
```

STEP 5: FIXING THE MISSING VALUES

In [ ]:
```
# impute catagoprical columns with 'unknown'
categorical_cols = ['Store_Type','Category','Payment_Method','Weather','Employee_ID
for col in categorical_cols:
  df[col] = df[col].fillna("Unknown") # Replacing missing values with unknown
df.head()
```

Out[ ]:

	Transaction_ID	Country	Region	Store_Type	Category	Item_ID	Quantity	Price_per_
0	T2867825	UK	South	Supermarket	Fruits	I36836	43.0	34
1	T1419610	USA	South	Wholesale	Beverages	I85457	76.0	41
2	T5614226	USA	Central	None	Beverages	I28818	53.0	15
3	T5108603	Canada	East	Unknown	Vegetables	I98489	29.0	37
4	T4744854	Brazil	West	Grocery Store	Dairy	I22306	13.0	46

In [ ]:
```
# Logical imputation for Boolean like columns
df['Discount_Applied'] = df['Discount_Applied'].fillna(False) # assume no discount
df['Loyalty_Card_Used'] = df['Loyalty_Card_Used'].fillna(False) # assume no loyalty
df['Return'] = df['Return'].fillna(False) # assume no return is missing
df['Holiday'] = df['Holiday'].fillna(False) # assume no holiday is missing
df.head()
```

Out[ ]:

	Transaction_ID	Country	Region	Store_Type	Category	Item_ID	Quantity	Price_per_
0	T2867825	UK	South	Supermarket	Fruits	I36836	43.0	34
1	T1419610	USA	South	Wholesale	Beverages	I85457	76.0	41
2	T5614226	USA	Central	None	Beverages	I28818	53.0	15
3	T5108603	Canada	East	Unknown	Vegetables	I98489	29.0	37
4	T4744854	Brazil	West	Grocery Store	Dairy	I22306	13.0	46

STEP 6: FILL MISSING TOTAL SALES

In [ ]:
```
# impute numerical columns with median
numerical_cols = ["Quantity","Price_per_Unit","Feedback_Score"]
for col in numerical_cols:
   df[col] = df[col].fillna(df[col].median()) # using median or mode to avoid outli
df.head()
```

Out[ ]:

	Transaction_ID	Country	Region	Store_Type	Category	Item_ID	Quantity	Price_per_
0	T2867825	UK	South	Supermarket	Fruits	I36836	43.0	34
1	T1419610	USA	South	Wholesale	Beverages	I85457	76.0	41
2	T5614226	USA	Central	None	Beverages	I28818	53.0	15
3	T5108603	Canada	East	Unknown	Vegetables	I98489	29.0	37
4	T4744854	Brazil	West	Grocery Store	Dairy	I22306	13.0	46

In [ ]:
```
# FILL TOTAL SALES BASED ON QUANTITY AND PRICE PER UNIT
df["Total_Sales"] = df["Quantity"] * df["Price_per_Unit"]
df["Total_Sales"]
```

Out[ ]:

	Total_Sales
0	14818.23
1	31474.64
2	8340.08
3	10786.26
4	6033.17
...	...
999995	17511.66
999996	23077.00
999997	10044.44
999998	26861.43
999999	10245.08

1000000 rows × 1 columns

dtype: float64

STEP 7: FILL MISSING STORE TYPE BASED ON REGION

In [ ]:
```
# FIND THE MOST COMMON STORE TYPE IN EACH REGION
common_store_type_per_region = df.groupby('Region')['Store_Type'].agg(lambda x : x.
common_store_type_per_region
```

Out[ ]:

Region	Store_Type
Central	Hypermarket
East	Grocery Store
North	Supermarket
South	Supermarket
West	Hypermarket

dtype: object

STEP 8: VERIFY THE FINAL DATASET

In [ ]:
```
# CHECK FOR REMAINING MISSING VALUES
missing_values_after = df.isnull().sum()
print(missing_values_after)
```

```
Transaction_ID            0
Country                   0
Region                    0
Store_Type                0
Category                  0
Item_ID                   0
Quantity                  0
Price_per_Unit            0
Total_Sales               0
Discount_Applied          0
Day                       0
Time_of_Purchase     100275
Payment_Method            0
Customer_ID               0
Loyalty_Card_Used         0
Employee_ID               0
Feedback_Score            0
Return                    0
Weather                   0
Holiday                   0
dtype: int64
```

STEP 9: SAVE THE CLEANED DATASET

In [ ]:
```
df.to_csv("cleaned_superstore_data.csv",index=False)
print("cleaned dataset saved as cleaned_superstore_data.csv")
```

```
cleaned dataset saved as cleaned_superstore_data.csv
```

(EDA) EXPLORATORY DATA ANALYSIS

In [ ]:
```
# CHECK THE DATA TYPE AND MISSING VALUES
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1000000 entries, 0 to 999999
Data columns (total 20 columns):
 #   Column             Non-Null Count    Dtype  
---  ------             --------------    -----  
 0   Transaction_ID     1000000 non-null  object 
 1   Country            1000000 non-null  object 
 2   Region             1000000 non-null  object 
 3   Store_Type         1000000 non-null  object 
 4   Category           1000000 non-null  object 
 5   Item_ID            1000000 non-null  object 
 6   Quantity           1000000 non-null  float64
 7   Price_per_Unit     1000000 non-null  float64
 8   Total_Sales        1000000 non-null  float64
 9   Discount_Applied   1000000 non-null  object 
 10  Day                1000000 non-null  object 
 11  Time_of_Purchase   899725 non-null   object 
 12  Payment_Method     1000000 non-null  object 
 13  Customer_ID        1000000 non-null  object 
 14  Loyalty_Card_Used  1000000 non-null  object 
 15  Employee_ID        1000000 non-null  object 
 16  Feedback_Score     1000000 non-null  float64
 17  Return             1000000 non-null  object 
 18  Weather            1000000 non-null  object 
 19  Holiday            1000000 non-null  object 
dtypes: float64(4), object(16)
memory usage: 160.2+ MB
None
```

In [ ]:
```
# display summary statistic for numeric column
df.describe()
```

Out[ ]:

	Quantity	Price_per_Unit	Total_Sales	Feedback_Score
count	1000000.000000	1000000.000000	1000000.000000	1000000.000000
mean	50.447168	250.272948	12620.490618	3.000176
std	27.391196	140.429403	10568.397514	1.183455
min	1.000000	1.000000	1.010000	1.000000
25%	28.000000	132.110000	3829.080000	2.000000
50%	50.000000	249.880000	10067.625000	3.000000
75%	73.000000	368.650000	19083.602500	4.000000
max	100.000000	500.000000	49992.000000	5.000000

In [ ]:
```
# calculate percentage of missing values for each column:
missing_percentage = df.isnull().sum() / len(df) * 100
print("percentage of missing values: ")
print(missing_percentage)
```

```
percentage of missing values:
Transaction_ID        0.0000
Country               0.0000
Region                0.0000
Store_Type            0.0000
Category              0.0000
Item_ID               0.0000
Quantity              0.0000
Price_per_Unit        0.0000
Total_Sales           0.0000
Discount_Applied      0.0000
Day                   0.0000
Time_of_Purchase     10.0275
Payment_Method        0.0000
Customer_ID           0.0000
Loyalty_Card_Used     0.0000
Employee_ID           0.0000
Feedback_Score        0.0000
Return                0.0000
Weather               0.0000
Holiday               0.0000
dtype: float64
```

In [ ]:
```
# CALCULATE MEAN , MEADIAN AND MODE FOR QUANTITY
print("mode of Quantity : ",df["Quantity"].mode()[0])
print("median of Quantity : ",df["Quantity"].median())
print("mean of Quantity : ",df["Quantity"].mean())

# CALCULATE MEAN , MEADIAN AND MODE FOR Price_per_Unit
print("mode of Price_per_Unit : ",df["Price_per_Unit"].mode()[0])
print("median of Price_per_Unit : ",df["Price_per_Unit"].median())
print("mean of Price_per_Unit : ",df["Price_per_Unit"].mean())
```

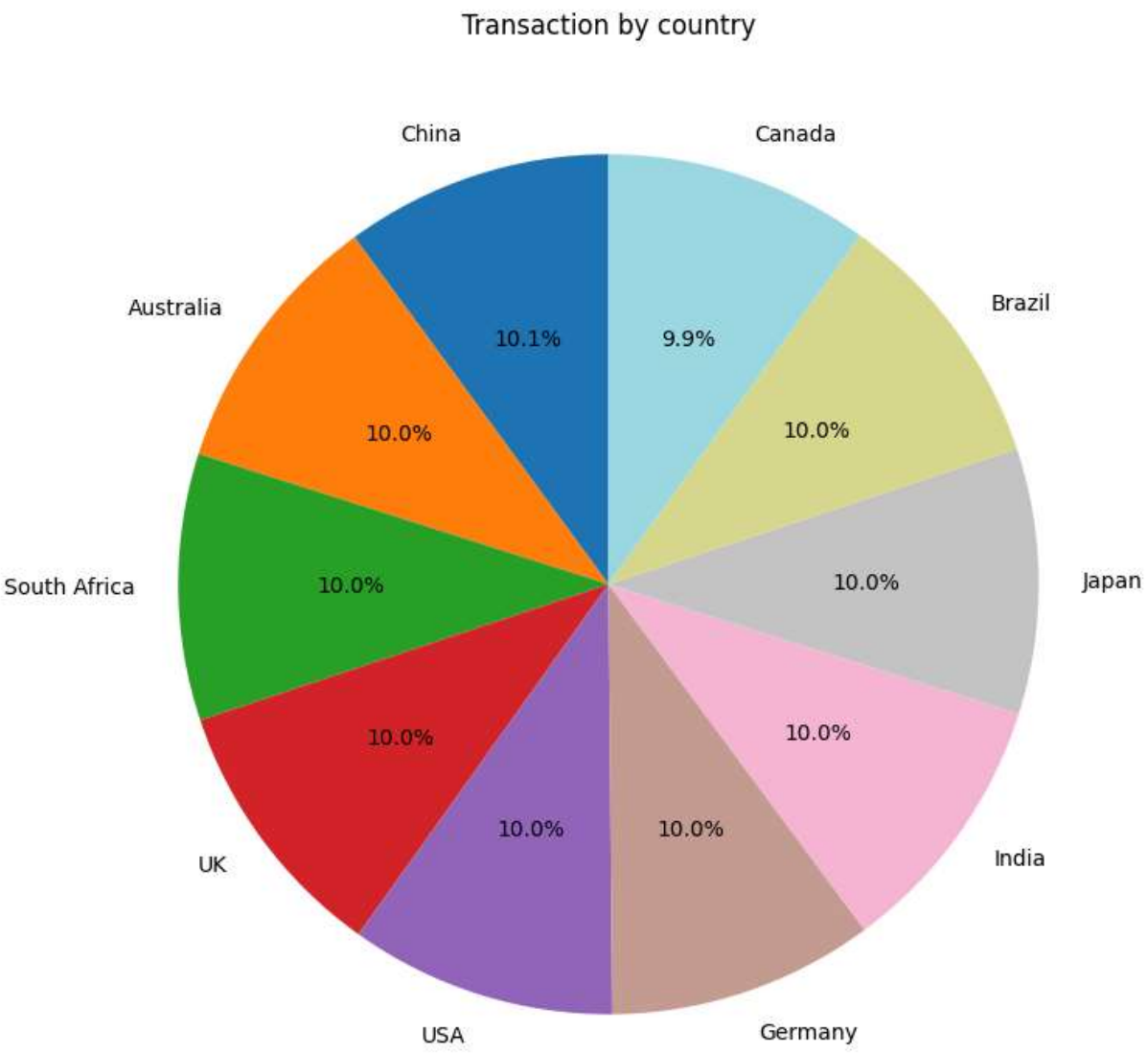
```
mode of Quantity :  50.0
median of Quantity :  50.0
mean of Quantity :  50.447168
mode of Price_per_Unit :  249.88
median of Price_per_Unit :  249.88
mean of Price_per_Unit :  250.27294781999998
```

CATEGORICAL VARIABLE ANALYSIS

In [ ]:
```
# DISTRIBUTION OF COUNTRY
# COUNT TRANSACTION BY COUNTRY
country_counts = df["Country"].value_counts()
print("Transaction by country")
print(country_counts)
```

```
Transaction by country
Country
China           100536
Australia       100379
South Africa    100297
UK              100116
USA             100111
Germany          99973
India            99859
Japan            99684
Brazil           99585
Canada           99460
Name: count, dtype: int64
```

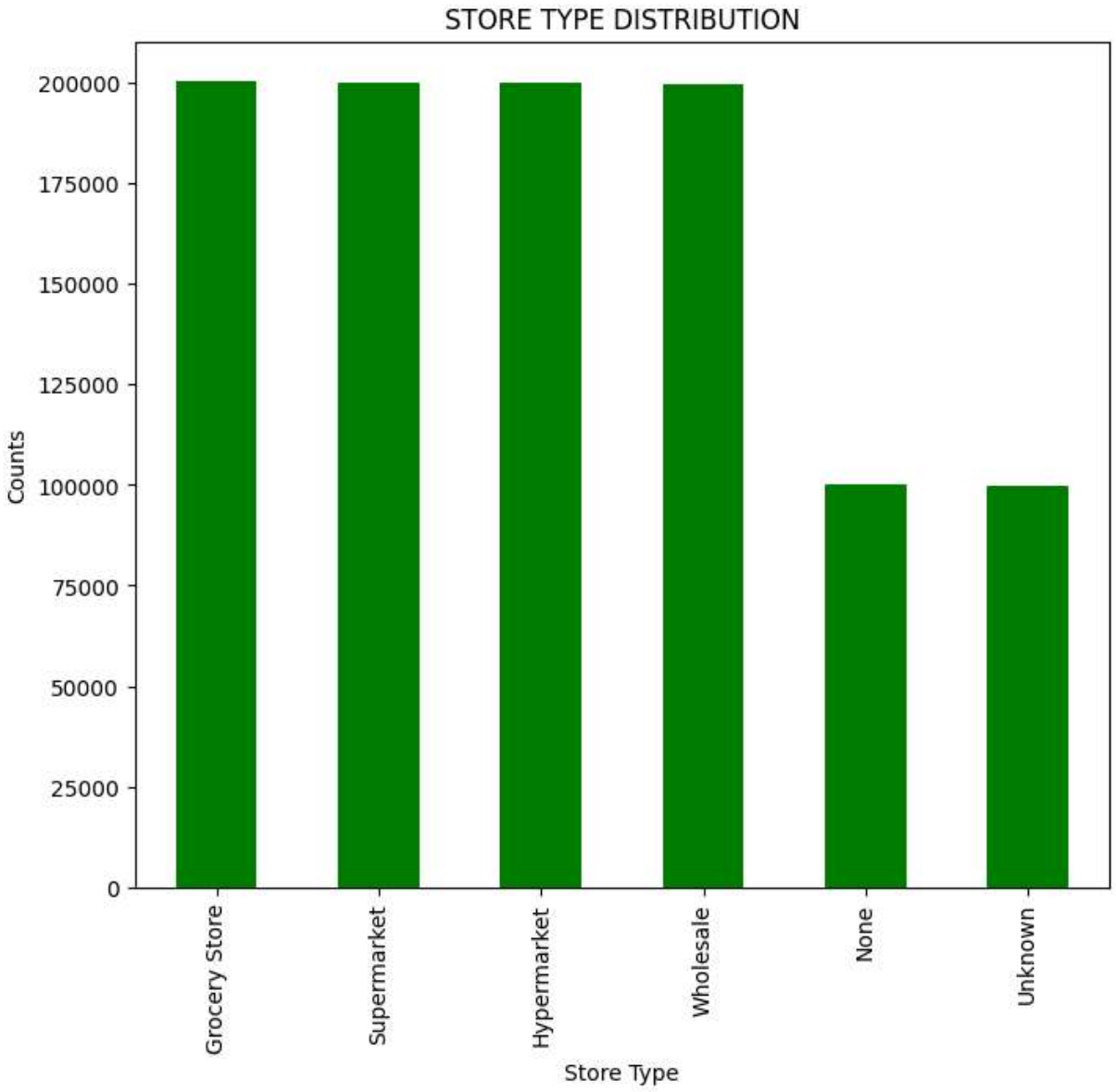
In [ ]:
```python
# PLOT DISTRIBUTION OF TRANSACTIONS BY COUNTRY
import matplotlib.pyplot as plt
plt.figure(figsize=(9,9))
country_counts.plot(kind="pie", autopct = '%1.1f%%' , startangle = 90, legend=False
plt.title("Transaction by country")
plt.ylabel('')
plt.show()
```



In [ ]:
```
# COUNT STORE TYPES
store_type_counts=df["Store_Type"].value_counts()
print("Store Type Distribution")
print(store_type_counts)
```

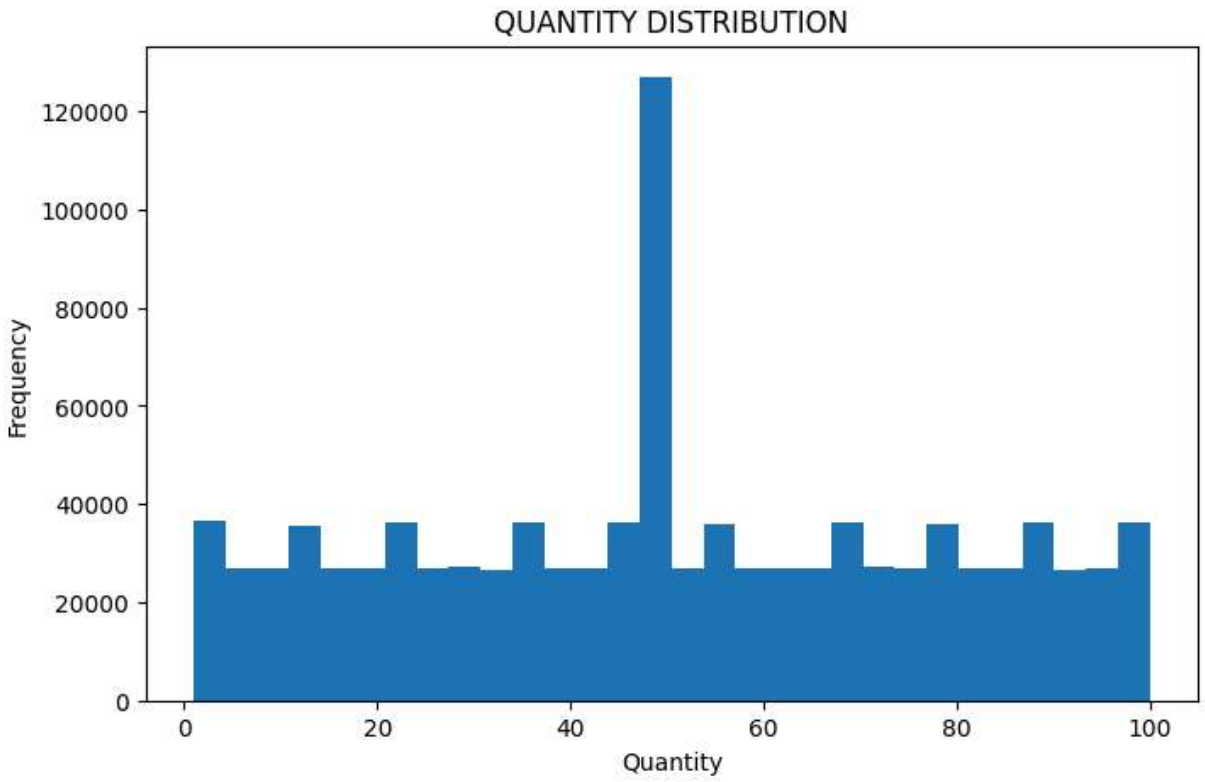
```
Store Type Distribution
Store_Type
Grocery Store    200265
Supermarket      200034
Hypermarket      199869
Wholesale        199744
None             100306
Unknown           99782
Name: count, dtype: int64
```

In [ ]:
```
# PLOT STORE TYPE DISTRIBUTION
store_type_counts.plot(kind='bar', title="STORE TYPE DISTRIBUTION", figsize=(8,7),
plt.xlabel("Store Type")
plt.ylabel("Counts")
plt.show()
```



NUMERICL VARIABLE ANALYSIS

In [ ]:
```
df['Quantity'].plot(kind='hist', bins=30, title="QUANTITY DISTRIBUTION", figsize=(8
plt.xlabel("Quantity")
plt.show()
```



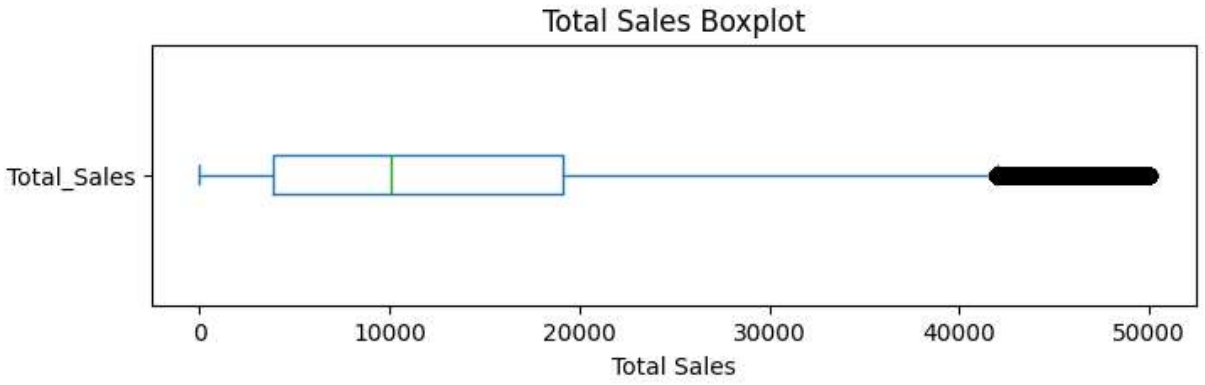
TOTAL SALES ANALYSIS

In [ ]:
```
# CALCULATE MEAN , MEDIAN FOR TOTAL SALES
print("Mean of Total Sales: ",df["Total_Sales"].mean())
print("Median of Total Sales : ",df["Total_Sales"].median())
```

```
Mean of Total Sales:  12620.49061751
Median of Total Sales :  10067.625
```

In [ ]:
```
# plot BOXPLOT OF TOTAL SALES
import matplotlib.pyplot as plt

# Boxplot of Total Sales
df['Total_Sales'].plot(kind="box", title='Total Sales Boxplot', vert=False, figsize
plt.xlabel("Total Sales") # Set the x-axis label
plt.show()
```

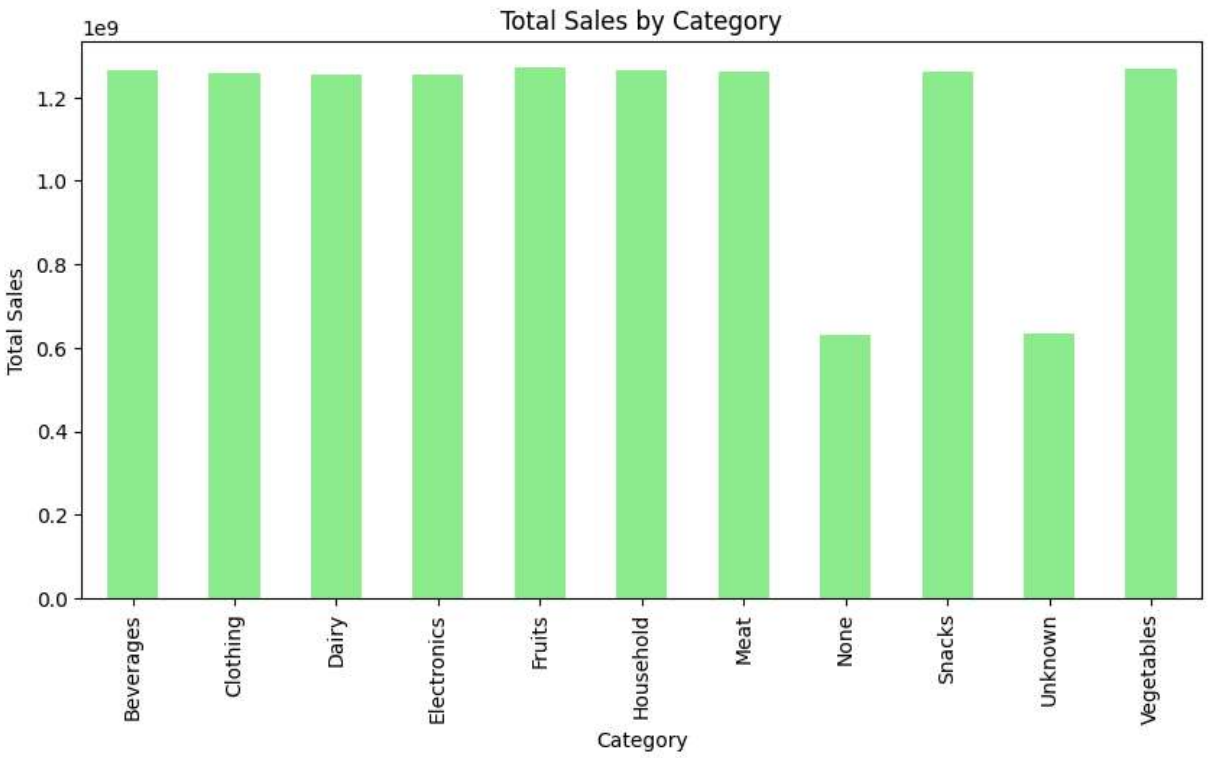


RELATIONSHIPS BETWEEN VARIABLES

In [ ]:
```
# CALCULATE TOTAL SALES BT CATEGORY
sales_by_category = df.groupby("Category")["Total_Sales"].sum()
print("Total Sales By Category")
print(sales_by_category)
```

```
Total Sales By Category
Category
Beverages      1.264905e+09
Clothing       1.257052e+09
Dairy          1.253018e+09
Electronics    1.254130e+09
Fruits         1.270640e+09
Household      1.264944e+09
Meat           1.262668e+09
None           6.310069e+08
Snacks         1.260826e+09
Unknown        6.331559e+08
Vegetables     1.268145e+09
Name: Total_Sales, dtype: float64
```

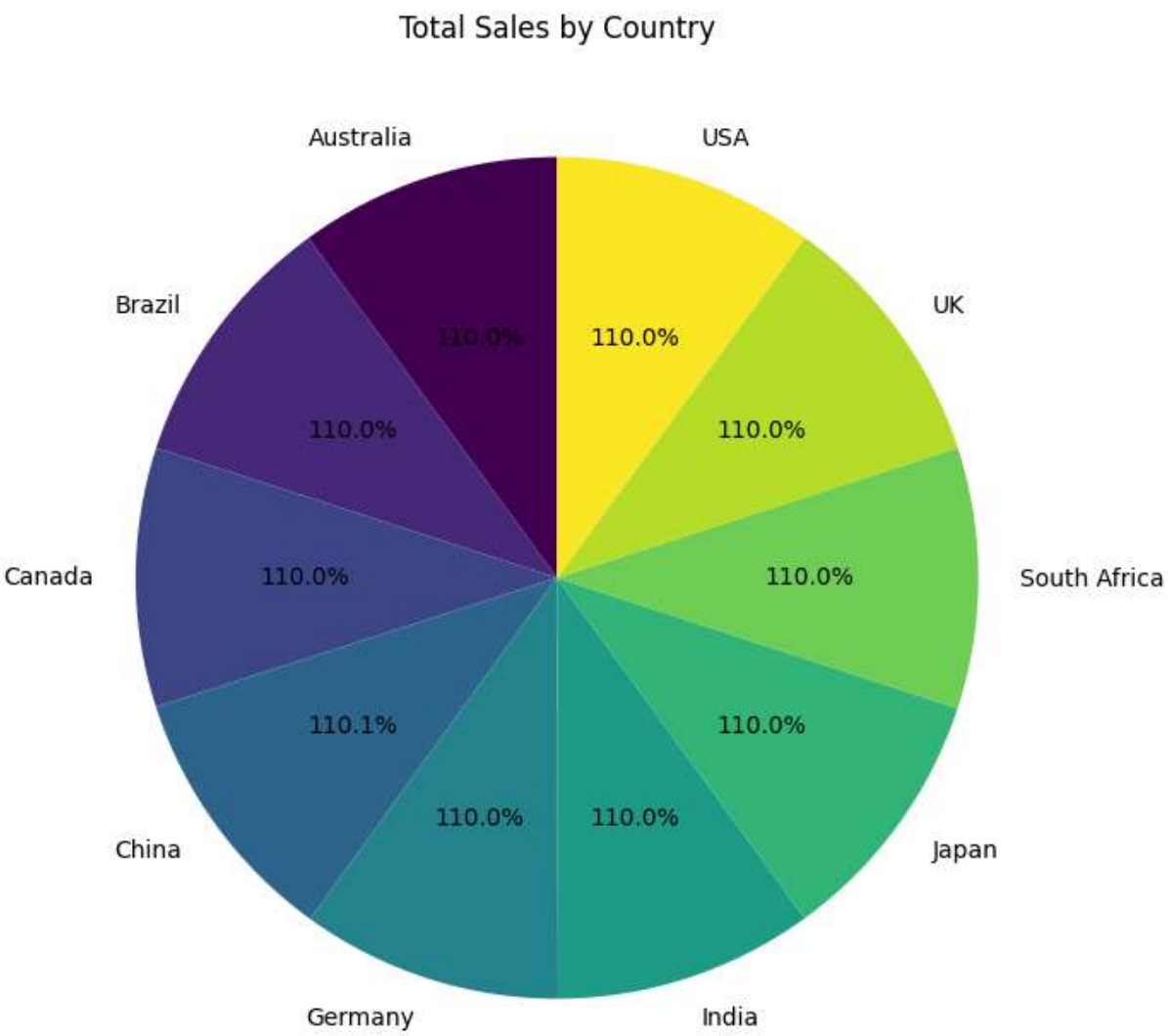
In [ ]:
```
# plot total sales by category
import matplotlib.pyplot as plt
sales_by_category.plot(kind='bar', title='Total Sales by Category', figsize=(10,5),
plt.xlabel("Category")
plt.ylabel("Total Sales")
plt.show()
```



In [ ]:
```
# CALCULATE TOTAL SALES BY COUNTRY
sales_by_country=df.groupby("Country")["Total_Sales"].sum()
print("Total Sales By Country")
print(sales_by_country)
```

```
Total Sales By Country
Country
Australia       1.267520e+09
Brazil          1.257053e+09
Canada          1.257347e+09
China           1.271371e+09
Germany         1.262303e+09
India           1.257777e+09
Japan           1.258318e+09
South Africa    1.264204e+09
UK              1.258527e+09
USA             1.266071e+09
Name: Total_Sales, dtype: float64
```

In [ ]:
```
# PLOT TOTAL SALES BY COUNTRY
ax=sales_by_country.plot(kind="pie", startangle=90, legend=False, autopct="1%.1f%%"
plt.title("Total Sales by Country")
ax.set_ylabel('')
plt.show()
```

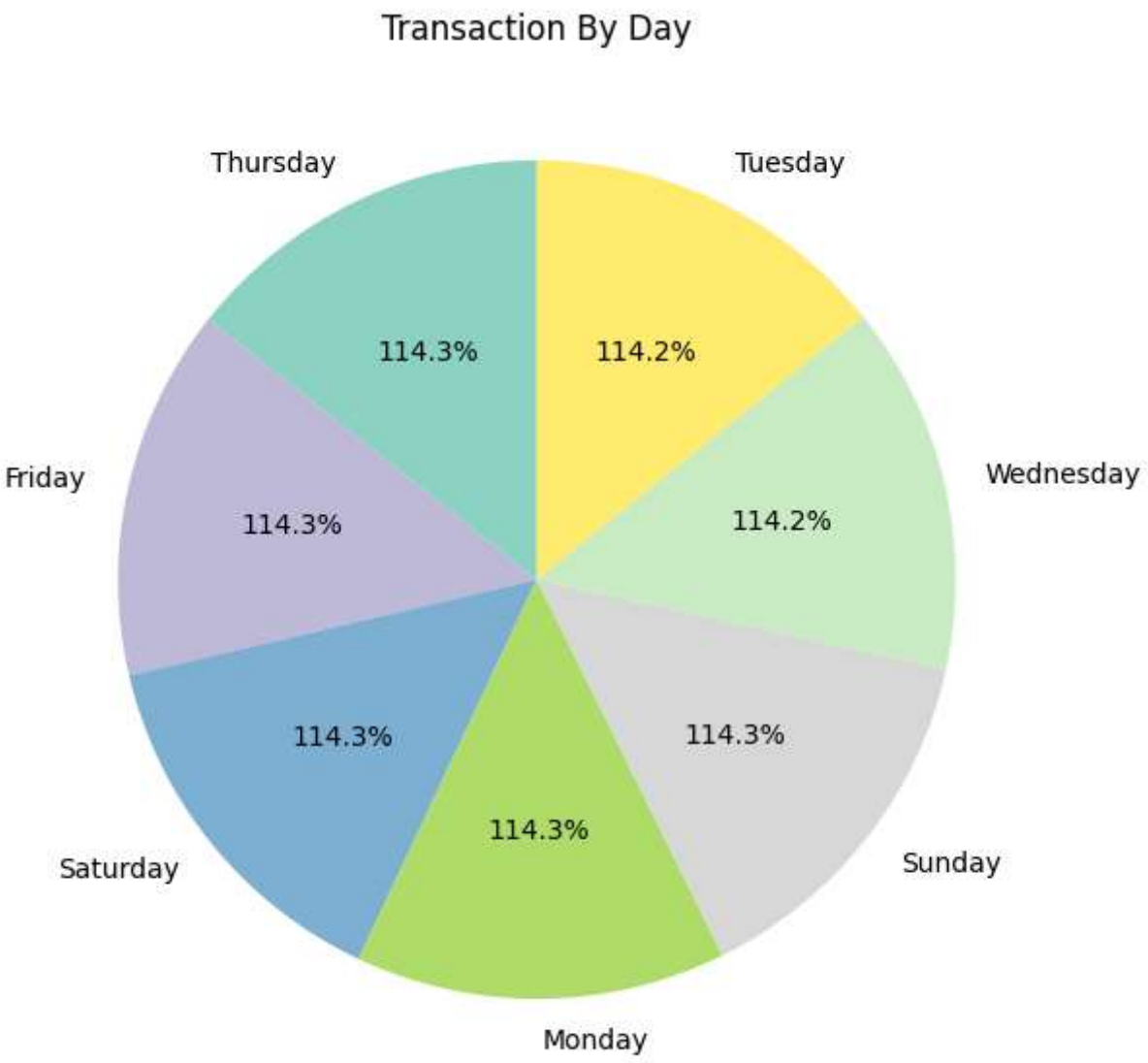


In [ ]:
```
# TRANSACTION BY DAY
# COUNT TRANSACTION BY DAY
transaction_by_day=df['Day'].value_counts()
print("Transaction by Day: ")
print(transaction_by_day)
```

```
Transaction by Day:
Day
Thursday     143415
Friday       143320
Saturday     143312
Monday       143082
Sunday       142525
Wednesday    142226
Tuesday      142120
Name: count, dtype: int64
```

In [ ]:
```
# PLOT TRANSACTION BY DAY
transaction_by_day.plot(kind='pie', autopct= '1%.1f%%', legend=False, startangle=90
plt.title("Transaction By Day")
```

```
plt.ylabel('')
plt.show()
```



In [ ]:
```
# CORRELATION ANALYSIS
# CALCULATE CORRELATION MATRX
correlation_matrix=df[['Quantity', 'Total_Sales', 'Price_per_Unit']].corr()
print("correlation matrix")
print(correlation_matrix)
```

```
correlation matrix
                Quantity  Total_Sales  Price_per_Unit
Quantity        1.000000     0.647023       -0.001318
Total_Sales     0.647023     1.000000        0.669130
Price_per_Unit -0.001318     0.669130        1.000000
```

In [ ]:
```
# KEY INSIGHTS
print("Most Transaction By Country: ",country_counts.idxmax())
print("Highest Total Sales by Category: ",sales_by_category.idxmax())
print("Country with highest total sales: ",sales_by_country.idxmax())
print("Day with most transaction: ",transaction_by_day.idxmax())
```

```
Most Transaction By Country:  China
Highest Total Sales by Category:  Fruits
Country with highest total sales:  China
Day with most transaction:  Thursday
```

SALES PERFOMANCE ANALYSIS

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1000000 entries, 0 to 999999
Data columns (total 20 columns):
 #   Column             Non-Null Count    Dtype  
---  ------             --------------    -----  
 0   Transaction_ID     1000000 non-null  object 
 1   Country            1000000 non-null  object 
 2   Region             1000000 non-null  object 
 3   Store_Type         1000000 non-null  object 
 4   Category           1000000 non-null  object 
 5   Item_ID            1000000 non-null  object 
 6   Quantity           1000000 non-null  float64
 7   Price_per_Unit     1000000 non-null  float64
 8   Total_Sales        1000000 non-null  float64
 9   Discount_Applied   1000000 non-null  object 
 10  Day                1000000 non-null  object 
 11  Time_of_Purchase   899725 non-null   object 
 12  Payment_Method     1000000 non-null  object 
 13  Customer_ID        1000000 non-null  object 
 14  Loyalty_Card_Used  1000000 non-null  object 
 15  Employee_ID        1000000 non-null  object 
 16  Feedback_Score     1000000 non-null  float64
 17  Return             1000000 non-null  object 
 18  Weather            1000000 non-null  object 
 19  Holiday            1000000 non-null  object 
dtypes: float64(4), object(16)
memory usage: 160.2+ MB
```

In [ ]:
```
df.head()
```

Out[ ]:

	Transaction_ID	Country	Region	Store_Type	Category	Item_ID	Quantity	Price_per_
0	T2867825	UK	South	Supermarket	Fruits	I36836	43.0	34
1	T1419610	USA	South	Wholesale	Beverages	I85457	76.0	41
2	T5614226	USA	Central	None	Beverages	I28818	53.0	15
3	T5108603	Canada	East	Unknown	Vegetables	I98489	29.0	37
4	T4744854	Brazil	West	Grocery Store	Dairy	I22306	13.0	46

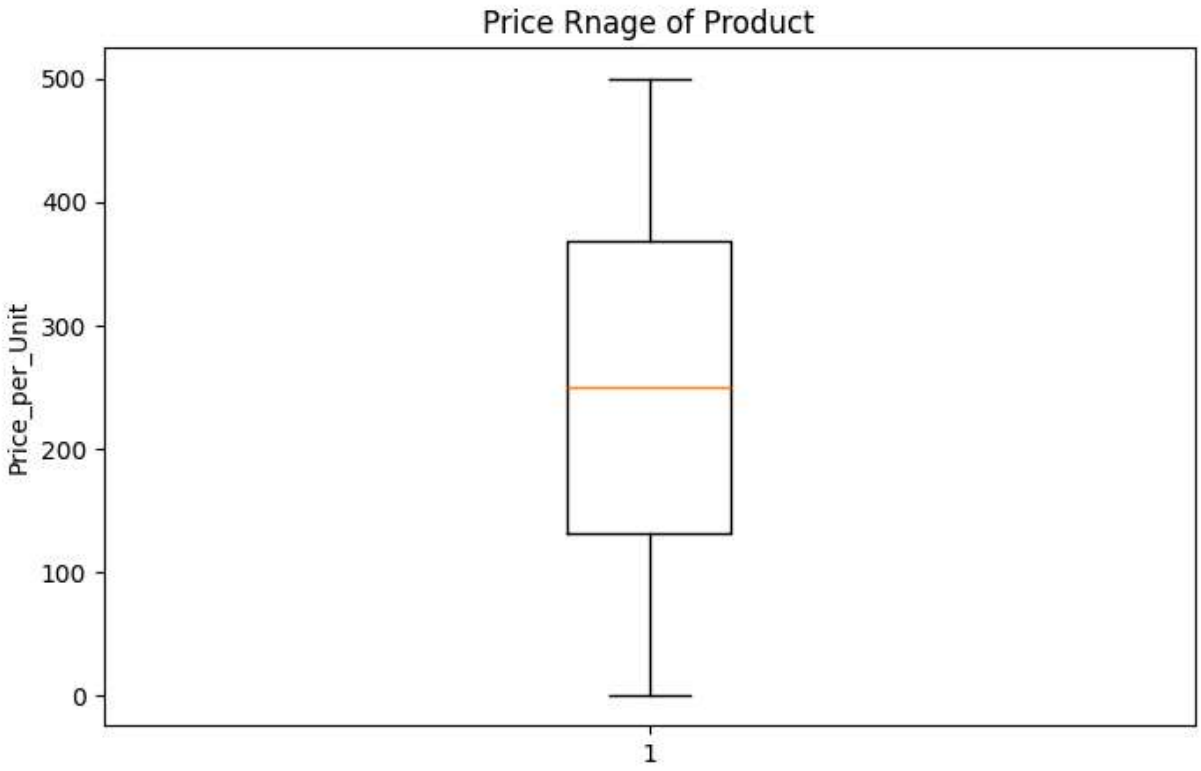
PRICE RANGES OF PRODUCT

In [ ]:
```
# CALCULATE MINIMUM , MAXIMUM AND AVERAGE PRICE PER UNIT
min_price=df["Price_per_Unit"].min()
max_price=df["Price_per_Unit"].max()
mean_price=df["Price_per_Unit"].mean()

print("Minimum Price: ",min_price)
print("Maximum price: ",max_price)
print("Mean Price: ",mean_price)
```

```
Minimum Price:  1.0
Maximum price:  500.0
Mean Price:  250.27294781999998
```

In [ ]:
```
# PLOT PRICE PER RANGE
plt.figure(figsize=(8,5))
plt.boxplot(df["Price_per_Unit"].dropna())
plt.title("Price Rnage of Product")
plt.ylabel("Price_per_Unit")
plt.show()
```



In [ ]:
```
df.head()
```

Out[ ]:

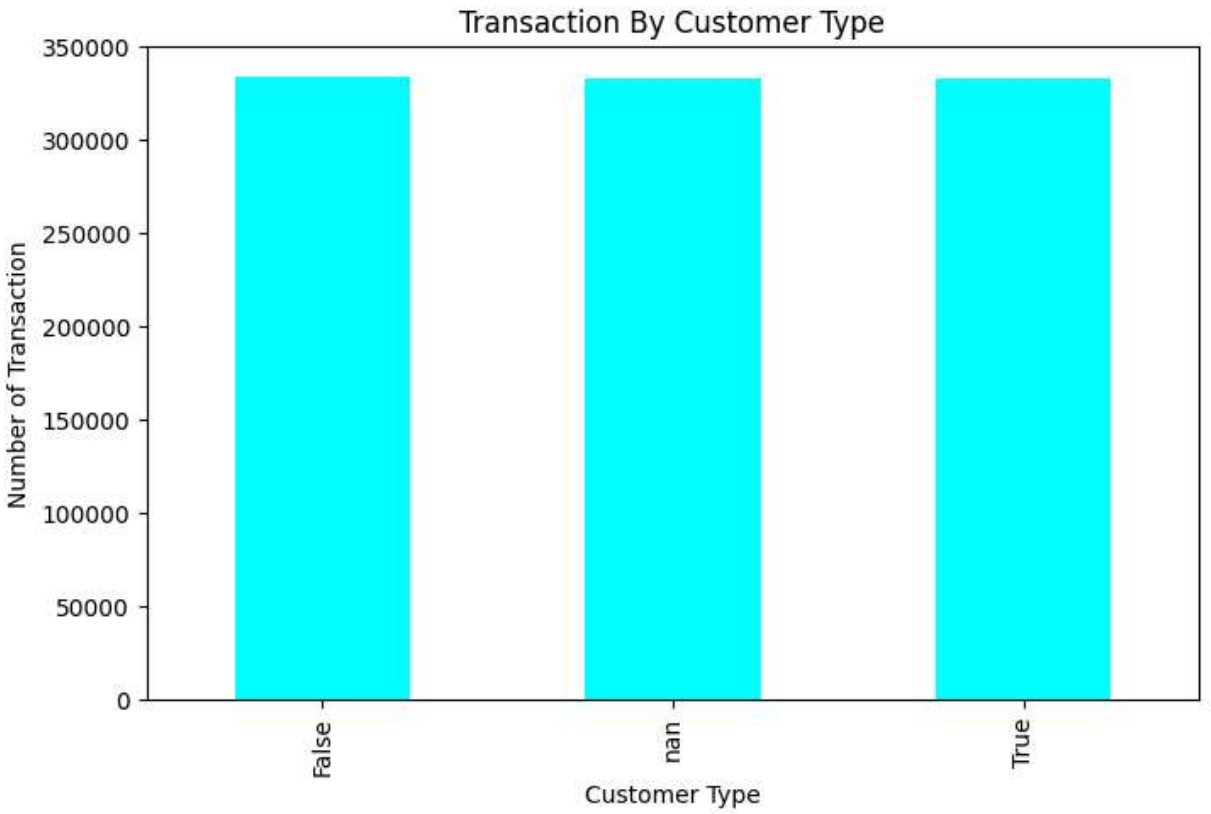
	Transaction_ID	Country	Region	Store_Type	Category	Item_ID	Quantity	Price_per_
0	T2867825	UK	South	Supermarket	Fruits	I36836	43.0	34
1	T1419610	USA	South	Wholesale	Beverages	I85457	76.0	41
2	T5614226	USA	Central	None	Beverages	I28818	53.0	15
3	T5108603	Canada	East	Unknown	Vegetables	I98489	29.0	37
4	T4744854	Brazil	West	Grocery Store	Dairy	I22306	13.0	46

In [ ]:
```
 # NUMBER OF TRANSACTION PER CUSTOMER TYPE
# COUNT TRANSACTION BY CUSTOMER TYPE
transaction_by_customer = df["Loyalty_Card_Used"].value_counts()
print("Number of transaction by customer type: ")
print(transaction_by_customer)
```

```
Number of transaction by customer type:
Loyalty_Card_Used
False    333824
nan      333153
True     333023
Name: count, dtype: int64
```

In [ ]:
```
# PLOT TRANSACTION BY CUSTOMER TYPE
from importlib import reload
import matplotlib.pyplot as plt
plt = reload(plt)

transaction_by_customer.plot(kind='bar', title="Transaction By Customer Type", colo
plt.xlabel("Customer Type")
plt.ylabel("Number of Transaction")
plt.show()
```



In [ ]:
```
df.head()
```

Out[ ]:

	Transaction_ID	Country	Region	Store_Type	Category	Item_ID	Quantity	Price_per_
0	T2867825	UK	South	Supermarket	Fruits	I36836	43.0	34
1	T1419610	USA	South	Wholesale	Beverages	I85457	76.0	41
2	T5614226	USA	Central	None	Beverages	I28818	53.0	15
3	T5108603	Canada	East	Unknown	Vegetables	I98489	29.0	37
4	T4744854	Brazil	West	Grocery Store	Dairy	I22306	13.0	46

PREPARING DATA FOR MODELING

HANDLE CATEGORICAL VARIABLE

In [ ]:
```
from sklearn.preprocessing import LabelEncoder

# STEP 1: CONVERT CATEGORICAL COLUMNS WITH NUMERIC CODE
df["Country"] = df["Country"].astype("category").cat.codes
df["Region"] = df["Region"].astype("category").cat.codes
df["Store_Type"] = df["Store_Type"].astype("category").cat.codes
df["Category"] = df["Category"].astype("category").cat.codes
df["Day"] = df["Day"].astype("category").cat.codes
```

```
df["Payment_Method"] = df["Payment_Method"].astype("category").cat.codes
df["Weather"] = df["Weather"].astype("category").cat.codes
```

In [ ]:
```
# LABEL ENCODING FOR BINARY COLUMNS
df_encoded = df.copy()
label_encoder = LabelEncoder()
binary_columns = ["Discount_Applied", "Loyalty_Card_Used", "Return", "Holiday"]

for col in binary_columns:
  df_encoded[col] = label_encoder.fit_transform(df_encoded[col].astype(str))
df_encoded
```

Out[ ]:

	Transaction_ID	Country	Region	Store_Type	Category	Item_ID	Quantity	Price_
0	T2867825	8	3	3	4	I36836	43.0	
1	T1419610	9	3	5	0	I85457	76.0	
2	T5614226	9	0	2	0	I28818	53.0	
3	T5108603	2	1	4	10	I98489	29.0	
4	T4744854	1	4	0	2	I22306	13.0	
...	...	...	...	...	...	...	...	
999995	T3306632	8	3	1	6	I41722	54.0	
999996	T5231756	4	0	4	8	I89670	50.0	
999997	T3801124	3	4	1	9	I61582	29.0	
999998	T6473675	9	4	1	3	I67279	71.0	
999999	T4259391	7	4	5	3	I71131	41.0	

1000000 rows × 20 columns

In [ ]:
```
# STEP 2:  DEFINE PREDICTORS AND TARGET VARIABLE
X = df_encoded[["Quantity","Total_Sales","Discount_Applied"]] # independent variabl
y = df_encoded["Price_per_Unit"] # dependent variable
```

In [ ]:
```
# STEP 3: SCALING FEATURES
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

PREDICTIVE MODELING

In [ ]:
```
from sklearn.model_selection import train_test_split
# split the data
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2, random_st
```

In [ ]:
```
# PRINT RESULT FOR VERIFICATION:
print("X_train shape: ",X_train.shape)
print("X test shape: ",X_test.shape)
print("y_train shape: ",y_train.shape)x
print("y_test shape: ",y_test.shape)
```

```
X_train shape:  (800000, 3)
X test shape:  (200000, 3)
 y train shape:  (800000,)
y_test shape:  (200000,)
```

TRAIN A LINEAR REGRESSION MODEL

In [ ]:
```
from sklearn.linear_model import LinearRegression

# TRAIN THE MODEL
model=LinearRegression()
model.fit(X_train, y_train)

# DISPLAY MODEL COEFFICIENTS
print("ModeL Coefficients: ",model.coef_)
print("Model_Intercept: ",model.intercept_)
```

```
ModeL Coefficients:  [-3.83291199  0.01531533 -0.09993237]
Model_Intercept:  250.50349030535344
```

EVALUATE THE MODEL

In [ ]:
```
from sklearn.metrics import mean_squared_error, r2_score

# Predict on test data
y_pred = model.predict(X_test)

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R-squared:", r2)

range_value = df["Price_per_Unit"].max() - df["Price_per_Unit"].min()
print("range of dependent variable",range_value)
```

```
Mean Squared Error: 4494.57893639602
R-squared: 0.7722739018455487
range of dependent variable 499.0
```

In [ ]:
```
y_pred
```

Out[ ]:
```
array([527.27597047, 298.02077037, 419.26524611, ..., 552.60457202,
       30.16804625, 270.4698619 ])
```

In [ ]:
```
y_test
```

Out[ ]:

	Price_per_Unit
987231	438.58
79954	329.82
567130	470.91
500891	249.37
55399	309.32
...	...
90245	210.52
639296	98.25
311939	477.07
324459	44.93
390499	268.06

200000 rows × 1 columns

dtype: float64

FEATURE IMPORTANCE

In [ ]:
```
# Analyze which features have the most impact on Price.
# Display feature importance (coefficients)
features = ["Quantity","Total_Sales","Discount_Applied"]
importance = model.coef_

print("Feature Importance:")
for feature, coef in zip(features, importance):
    print(f"{feature}: {coef}")
```

```
Feature Importance:
Quantity: -3.832911991471183
Total_Sales: 0.015315334994994581
Discount_Applied: -0.09993237393530624
```

SAVE PREDICTIONS

In [ ]:
```
a# Save the predictions for future use or comparison.
# Save predictions to a new DataFrame
predictions = X_test.copy()
predictions["Actual Price"] = y_test
```

```
predictions["Predicted Price"] = y_pred

# Save to a CSV file
predictions.to_csv("superstore_predictions.csv", index=False)

print("Predictions saved to 'superstore_predictions.csv'")
```

```
Predictions saved to 'superstore_predictions.csv'
```

ADVANCED PREDICTIVE MODELING WITH RANDOM FOREST

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor

# Train a Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

print("Random Forest model trained successfully.")
```

```
Random Forest model trained successfully.
```

EVALUATE RANDOM FOREST PERFOMANCE

In [ ]:
```
# Compare performance with the earlier linear regression model.
# Predict on test data
rf_predictions = rf_model.predict(X_test)

# Calculate evaluation metrics
from sklearn.metrics import mean_squared_error, r2_score

rf_mse = mean_squared_error(y_test, rf_predictions)
rf_r2 = r2_score(y_test, rf_predictions)

print("Random Forest Mean Squared Error:", rf_mse)
print("Random Forest R-squared:", rf_r2)
import math
math.sqrt(rf_mse)
```

```
Random Forest Mean Squared Error: 0.0015920292572499833
Random Forest R-squared: 0.9999999193369132
```

Out[ ]:
```
0.039900241318192343
```

FEATURE IMPORTANCE IN RANDOM FOREST

In [ ]:
```
# Display feature importance
feature_importance = rf_model.feature_importances_

print("Feature Importance (Random Forest):")
for feature, importance in zip(features, feature_importance):
    print(f"{feature}: {importance}")
```

```
Feature Importance (Random Forest):
Quantity: 0.33460618636854783
Total_Sales: 0.6653937486284534
Discount_Applied: 6.500299889335449e-08
```

SAVED ENHANCED DATASET

In [ ]:
```
df["Predict Price (RF)"] = rf_model.predict(X)

# SAVE TO A NEW CSV
df.to_csv("enhanced_superstore_dataset.csv",index=False)
print("Enhanced Dataset Saved as 'enhanced superstore dataset.csv")
```

```
Enhanced Dataset Saved as 'enhanced superstore dataset.csv
```

In [ ]:

In [ ]: