DATA PRE-PROCESSING: SALES DATA

- GROUP 5
  - 2602569722 - AURELLIA GITA ELYSIA
  - 2602571286 - GERRALD NATHANIEL SYARIF
  - 2602569685 - MARWAH INAS RABBANI

December 2024

IDENTIFICATION PROBLEMS

WE WILL SOLVE THE PROBLEMS

01 Missing Values

```
1 df.isnull().sum()
```

	0
Order_ID	0
Product_Name	0
Category	0
Price	0
Quantity	194
Total_Price	194
Customer_Name	60
Date	0
Discount	687
Region	40

The dataset include missing values for several tables;

- Quantity
- Total_Price
- Customer_Name
- Discount
- Region

02 Outliers

	Order_ID	Product_Name	Category	Price	Quantity	Total_Price	Customer_Name	Date	Discount	Region
4	2004	Your Classic	Furniture	-949.11	2.0	1898.22	Michelle Smith	6/19/2023	NaN	North
5	2005	Western Lite	Sports	80.92	1.0	80.92	Kyle Moody PhD	9/7/2023	7%	South
6	2006	With Max	Electronics	455.06	4.0	1820.24	Robin Wood	3/6/2023	24%	East
7	2007	Let Max	Sports	31.56	2.0	63.12	Jasmine Mayer	8/10/2024	9%	East
8	2008	Among Max	Electronics	50.91	4.0	203.64	NaN	1/27/2023	NaN	South
9	2009	Challenge Lite	Apparel	33.24	3.0	99.72	Thomas Schneider	1/21/2023	NaN	West
10	2010	Chair Max	Electronics	67.25	NaN	NaN	Katherine Howe	7/24/2023	NaN	South
11	2011	Far Plus	Home Decor	27.48	NaN	NaN	Maria Soto	2/27/2024	4%	South
12	2012	Check Max	Home Decor	83.39	4.0	333.56	7037	8/24/2024	NaN	West

Some outliers include:

- Negative value in the Price column (-33.24)
- Numerical for Customer_Name (7037)

IDENTIFICATION PROBLEMS

WE WILL SOLVE THE PROBLEMS

03 Invalid Date

Because invalid date still have the price values that are important to the analysis, we leave them as invalid

04 Duplicate Data

```
[13]  1 # Check if there are any duplicate rows in the entire DataFrame
      2 duplicates = df[df.duplicated()]
      3
      4 # Display the duplicate rows
      5 print(duplicates)
```

```
Empty DataFrame
Columns: [Order_ID, Product_Name, Category, Price, Quantity, Total_Price, Customer_Name, Date, Discount, Region, Month, Year]
Index: []
```

Based on the code above, there are no duplicates within the dataset

IDENTIFICATION PROBLEMS

WE WILL SOLVE THE PROBLEMS

03 Invalid Date

Because invalid date still have the price values that are important to the analysis, we leave them as invalid

04 Duplicate Data

```
[13]  1 # Check if there are any duplicate rows in the entire DataFrame
      2 duplicates = df[df.duplicated()]
      3
      4 # Display the duplicate rows
      5 print(duplicates)
```

```
Empty DataFrame
Columns: [Order_ID, Product_Name, Category, Price, Quantity, Total_Price, Customer_Name, Date, Discount, Region, Month, Year]
Index: []
```

Based on the code above, there are no duplicates within the dataset

SOLUTION

Missing Values

Region

First method, fill missing region with product that sold in the same region

```
def fill_region(row, region_map):
    if pd.isna(row['Region']):
        product_name = row['Product_Name']
        if product_name in region_map:
            return region_map[product_name]
    return row['Region']
```

```
region_map = (df.dropna(subset=['Region'])
              .groupby('Product_Name')['Region']
              .agg(lambda x: x.mode()[0] if not x.mode().empty else np.nan)
              .to_dict())

df['Region'] = df.apply(lambda row: fill_region(row, region_map), axis=1)

print(f"Missing values in 'Region': {df['Region'].isna().sum()}")
```

Second, fill the rest of missing region with mode of each category

```
df['Region'] = df.groupby('Category')['Region'].transform(lambda x: x.fillna(x.mode()[0] if not x.mode().empty else 'Unknown'))

print(f"Missing values in 'Region': {df['Region'].isna().sum()}")
```

SOLUTION

Missing Values

Customer Name

Because there's no customer that did multiple purchase on the same products, we fill missing customer name with “Customer_number”

ex : Customer 1, Customer 2

```
# Initialize Counter
customer_counter = 1

# Replace missing or numeric values with 'Customer {n}'
for idx in range(len(df)):
    value = df.at[idx, 'Customer_Name']

    # Check missing and numeric value
    if pd.isnull(value) or value.isdigit():
        df.at[idx, 'Customer_Name'] = f'Customer {customer_counter}'  # Replace with 'Customer {n}'
        customer_counter += 1  # Increment the counter for the next missing or numeric value

# Verify the changes
print(df['Customer_Name'].head(20))  # Preview the column
print(f"Missing values in 'Customer_Name': {df['Customer_Name'].isna().sum()}")
```

SOLUTION

Missing Values

Quantity

```
# Extract 'Month' and 'Year' from the 'Date' column
df['Month'] = df['Date'].dt.month
df['Year'] = df['Date'].dt.year

# Convert the 'Month' and 'Year' to integer to remove decimals
df['Year'] = df['Year'].astype('Int64')
df['Month'] = df['Month'].astype('Int64')

# Group by 'Category', 'Year', and 'Month' and calculate the average quantity
average_quantity_calculation = df.groupby(['Category', 'Year', 'Month'])['Quantity'].mean().round().reset_index()

# Rename the column for clarity
average_quantity_calculation = average_quantity_calculation.rename(columns={'Quantity': 'Average_Quantity'})

# Display the result
print(average_quantity_calculation.head())
```

First approach, fill missing quantity based on average purchase each month and year of each category

Second approach, fill the rest of missing quantity with mode of the category

```
# For each category, find the mode (most frequent value) of the 'Quantity' column
category_mode = df.groupby('Category')['Quantity'].apply(lambda x: x.mode()[0])

# Fill missing 'Quantity' based on category mode
def fill_missing_quantity(row):
    if pd.isna(row['Quantity']):
        return category_mode[row['Category']]
    return row['Quantity']
df['Quantity'] = df.apply(fill_missing_quantity, axis=1)

# Verify missing values
print(df[df['Quantity'].isna()])
```

SOLUTION

Missing Values

Total Price

Multiplying Quantity with price

```
df['Total_Price'] = df['Total_Price'].fillna(df['Price'] * df['Quantity'])
print(df[df['Total_Price'].isnull()])
```

For total price that we found mismatch with quantity * price,
we replace manually using our calculation

```
df = df.drop(columns=['Calculated_Total_Price'])
df.loc[mismatch_rows.index, 'Total_Price'] = df.loc[mismatch_rows.index, 'Price'] * df.loc[mismatch_rows.index, 'Quantity']

# Verify the changes
df.loc[mismatch_rows.index]
```

SOLUTION

Missing Values

Discount

First approach, fill discount with other products with the same product name and month that have a discount

```
# Group by 'Date' and 'Discount' to count how many products have each discount on a given day
discounts_by_date = df.groupby(['Date', 'Discount']).size().reset_index(name='Product_Count')

# Filter to keep only the rows where there are more than 1 product with the same discount on the same day
valid_discounts = discounts_by_date[discounts_by_date['Product_Count'] > 1]

# Map each 'Date' to the most frequent discount for that date, but only for dates with more than 1 product
date_to_discount = valid_discounts.set_index('Date')['Discount'].to_dict()

# Fill missing 'Discount' values based on the most frequent discount for the same date (only if more than 1 product has the same discount)
def fill_discount_based_on_date(row):
    if pd.isna(row['Discount']):
        # Look for the most frequent discount for the same date, but only if there are multiple products with the same discount
        discount = date_to_discount.get(row['Date'], np.nan)
        if pd.notna(discount):
            return discount
    return row['Discount']

# Apply the function to fill missing 'Discount' values
df['Discount'] = df.apply(fill_discount_based_on_date, axis=1)
```

SOLUTION

Missing Values

Discount

Second approach, fill missing value with the same discount on the same day

```python
# Group by 'Date' and 'Discount' to count how many products have each discount on a given day
discounts_by_date = df.groupby(['Date', 'Discount']).size().reset_index(name='Product_Count')

# Filter to keep only the rows where there are more than 1 product with the same discount on the same day
valid_discounts = discounts_by_date[discounts_by_date['Product_Count'] > 1]

# Map each 'Date' to the most frequent discount for that date, but only for dates with more than 1 product
date_to_discount = valid_discounts.set_index('Date')['Discount'].to_dict()

# Fill missing 'Discount' values based on the most frequent discount for the same date (only if more than 1 product has the same discount)
def fill_discount_based_on_date(row):
    if pd.isna(row['Discount']):
        # Look for the most frequent discount for the same date, but only if there are multiple products with the same discount
        discount = date_to_discount.get(row['Date'], np.nan)
        if pd.notna(discount):
            return discount
    return row['Discount']

# Apply the function to fill missing 'Discount' values
df['Discount'] = df.apply(fill_discount_based_on_date, axis=1)

# Check the result
# print(df)
print(df[df['Discount'].isnull()])
```

SOLUTION

Missing Values

Discount

Last approach, if there are no matching pattern found, we will assume that the missing values indicate no discount, and set those values to 0%

```
# Fill the remaining missing values with 0%, indicating that there are no discount in the transaction
df['Discount'] = df['Discount'].fillna(0)


# Convert 'Discount' column to percentage
def convert_to_percentage(value):
    if isinstance(value, str) and '%' in value:
        return value  # Already a percentage, return as is
    elif isinstance(value, (int, float)):
        return f"{value * 100}%"  # Convert decimal to percentage
    return value  # Return the value as is if it doesn't match either condition

# Apply the function to the column
df['Discount'] = df['Discount'].apply(convert_to_percentage)
```

SOLUTION

Outliers

Negative Value on Price column

First approach, find products with the same name, if the price is positive, then replace the negative price with the price of the same product name

```
# Initialize counters
valid_price_count = 0
no_valid_price_count = 0

# Step 2: Loop through rows with negative 'Price'
for index, row in negative_price.iterrows():
    # Check if there are other products with the same 'Product_Name'
    matching_products = df[df['Product_Name'] == row['Product_Name']]

    # Filter matching products for valid prices
    valid_prices = matching_products[matching_products['Price'] > 0]['Price']

    # Check if there are any valid prices available
    if not valid_prices.empty:
        # Get the first valid price
        valid_price = valid_prices.iloc[0]

        # Fill the negative price with the valid price
        df.at[index, 'Price'] = valid_price

        # Increment the valid price counter
        valid_price_count += 1
    else:
        # Increment the no valid price counter
        no_valid_price_count += 1

# Output the counts
print(f"Number of products with valid prices filled: {valid_price_count}")
print(f"Number of products with no valid price found: {no_valid_price_count}")
```

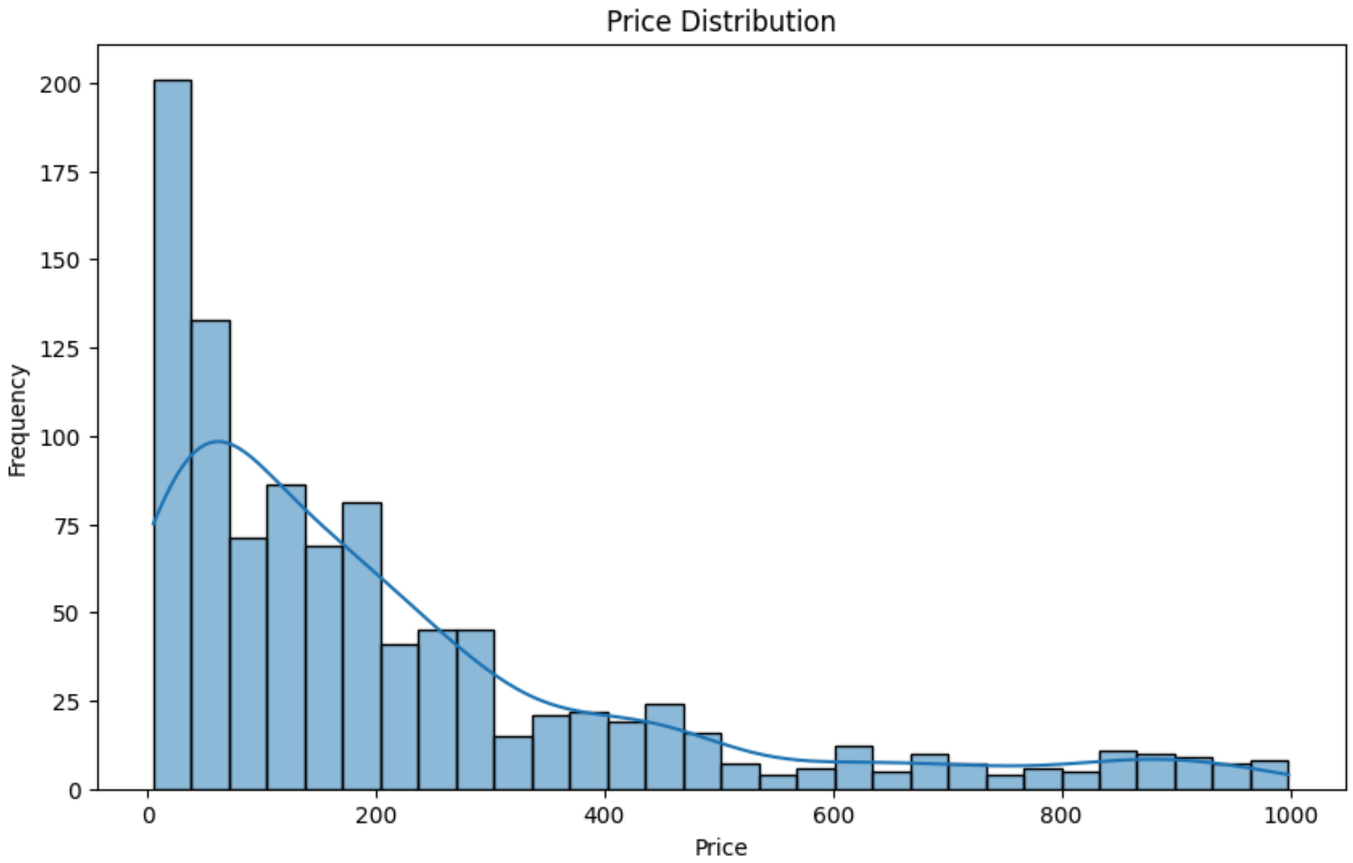
```
df['Price'] = df['Price'].abs()
```

Second approach, fill the rest negative price by convert negative to positive by using .abs() command

GRAPH

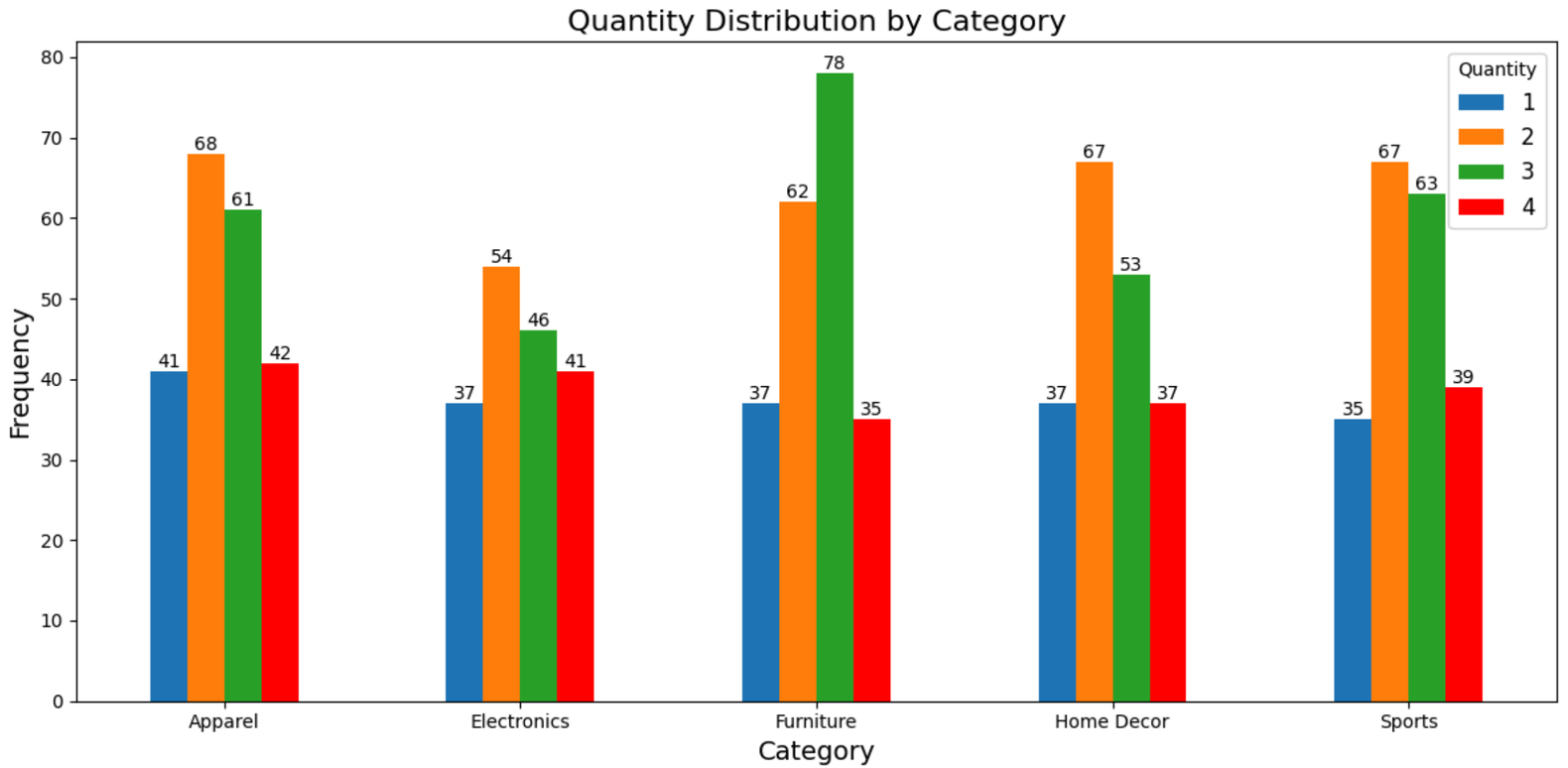
The histogram bars indicate that the majority of sales fall within the lower price ranges, particularly below 200. This is further supported by the overlaid density curve, which highlights the right-skewed nature of the distribution. As prices increase, the frequency of sales diminishes significantly, with relatively few transactions occurring in the higher price ranges (above 500).

This distribution suggests that the store's revenue is largely driven by smaller transactions, indicating that customers tend to purchase low-cost items or have smaller basket sizes. High-value sales are infrequent, potentially highlighting an opportunity to increase revenue through strategies like upselling, bundling products, or targeting high-value customers with premium offerings.



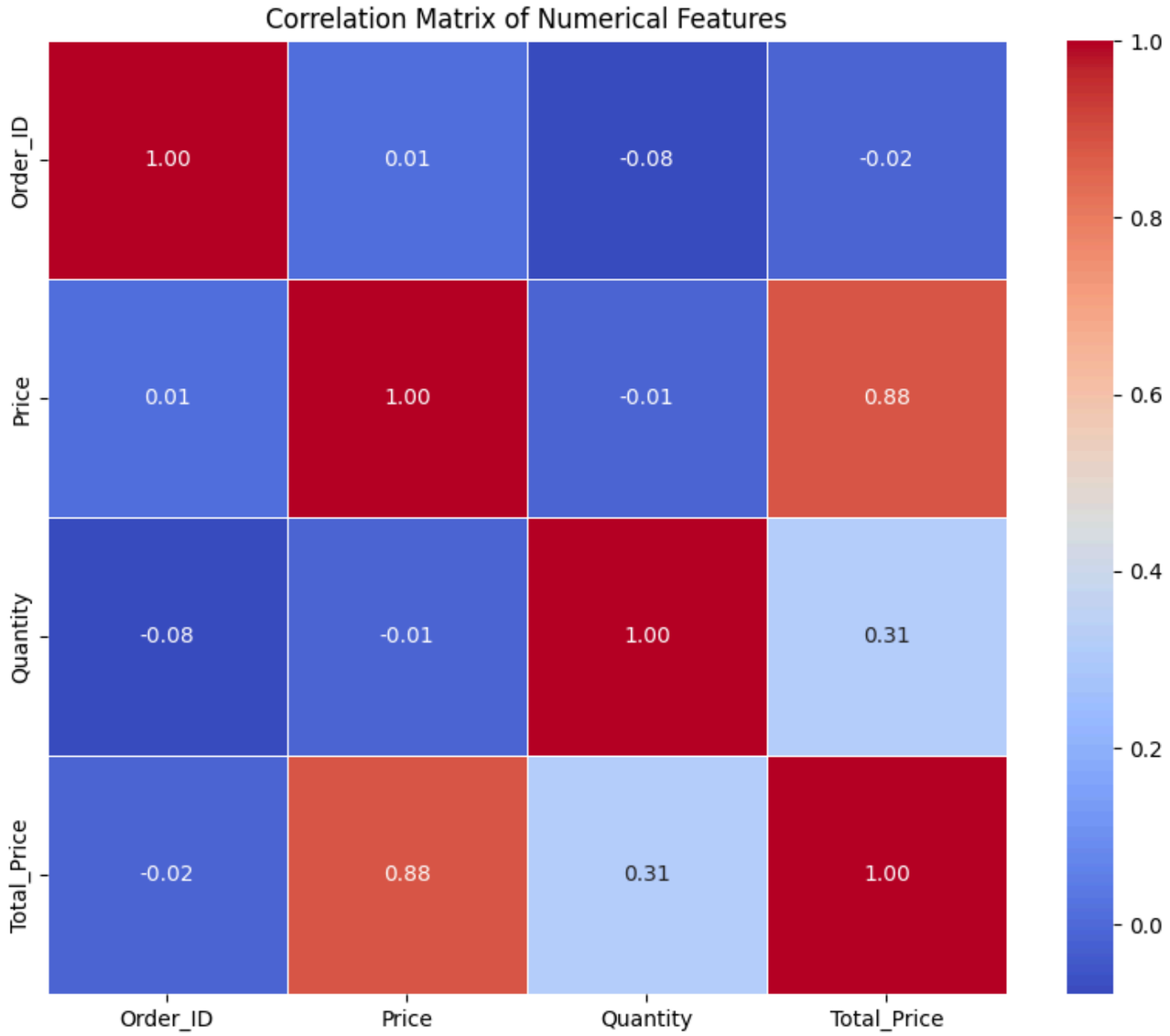
Additionally, understanding the distribution can guide inventory management and promotional strategies to cater to the predominant customer spending patterns. For example, focusing on popular lower-cost items could optimize stock turnover, while marketing higher-value products may help balance the sales distribution.

GRAPH



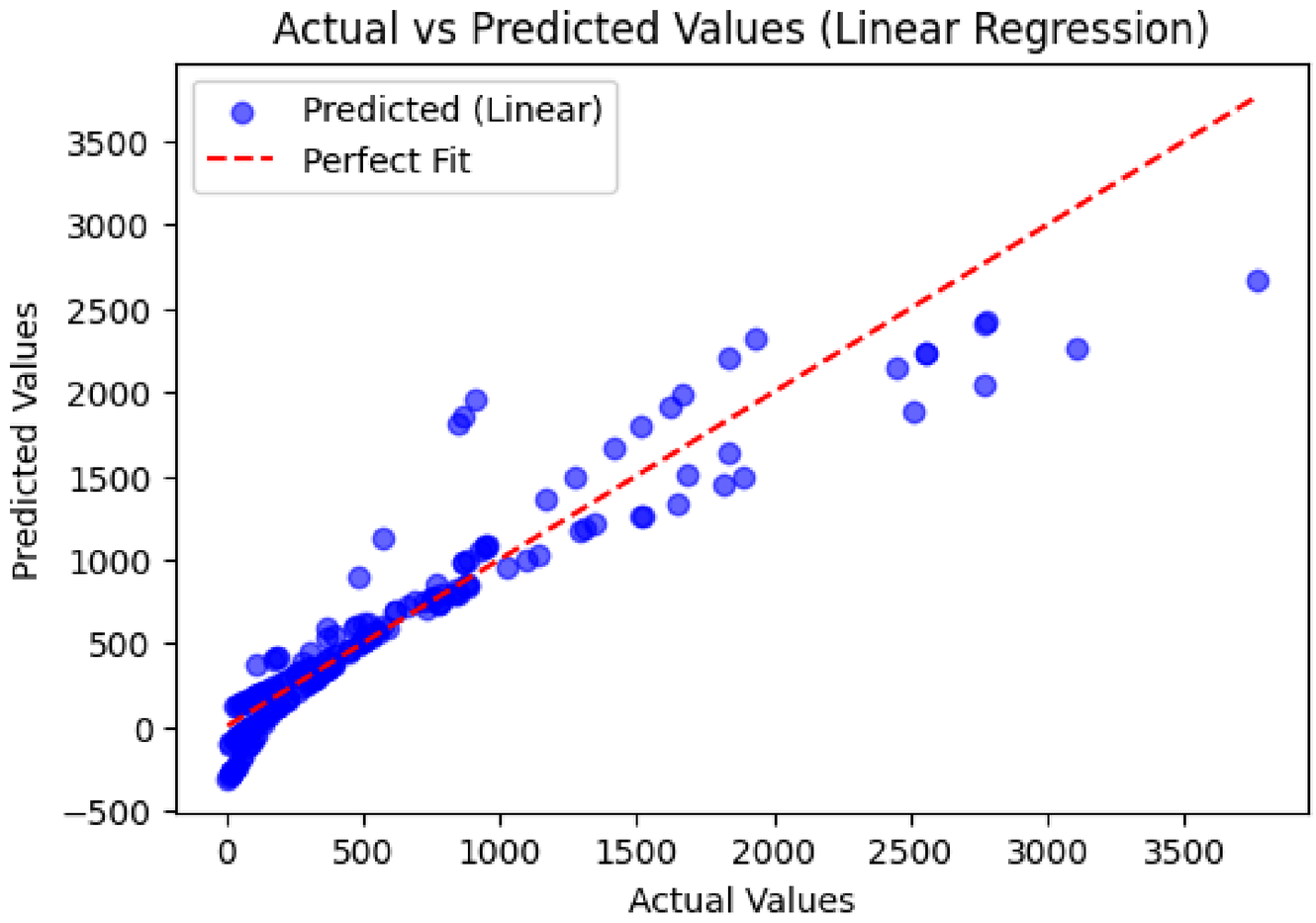
This chart illustrates the frequency of purchase quantities (1 to 4) across five product categories: Apparel, Electronics, Furniture, Home Decor, and Sports. For most categories, quantities of 2 and 3 are the most frequently purchased, with quantity 2 dominating in Apparel, Electronics, Home Decor, and Sports, and quantity 3 being particularly prominent in Furniture. Single-item purchases (quantity 1) and larger purchases (quantity 4) are less common across all categories, with relatively consistent frequencies. Notably, Furniture exhibits a distinct purchasing pattern where quantity 3 is significantly more frequent than other quantities, indicating a strong preference for mid-sized bundles in this category.

HEATMAP



- The matrix suggests that **Price** (0.88) and **Quantity** (0.31) are the two most relevant features for predicting **Total_Price**. Including these in the machine learning model can provide meaningful patterns for accurate predictions.
- Features with negligible correlation, like **Order_ID**, are unlikely to contribute to the prediction and can be excluded to reduce noise and improve model efficiency.

SCATTER PLOT

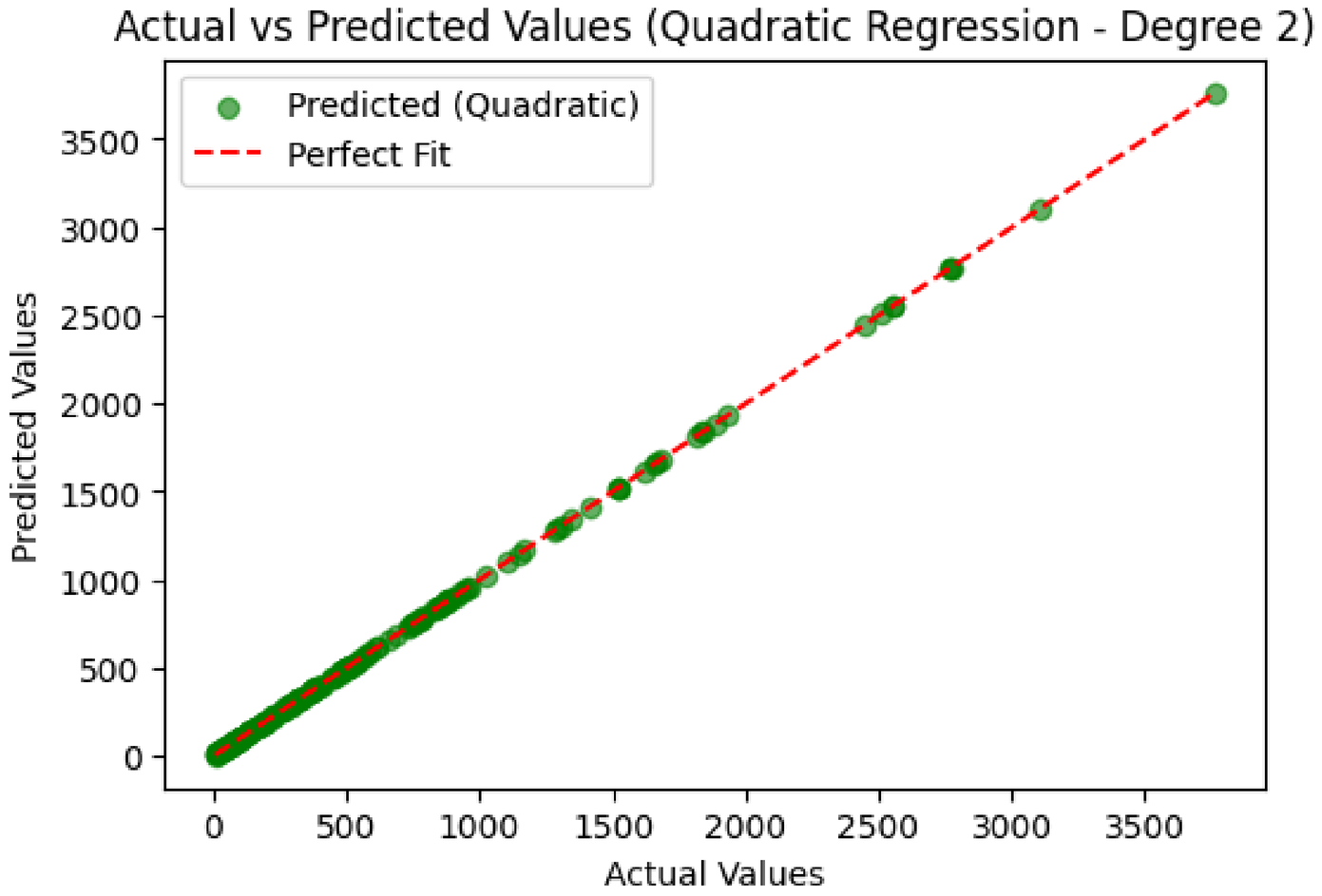


Linear Regression Metrics:

- **Mean Squared Error (MSE):** 52359.421313589475
- **Root Mean Squared Error (RMSE):** 228.82181127154263
- **Mean Absolute Error (MAE):** 142.06257048165526

Model ini menunjukkan kesalahan prediksi yang cukup besar, yang berarti hubungan antara fitur dan target tidak cukup linear.

SCATTER PLOT

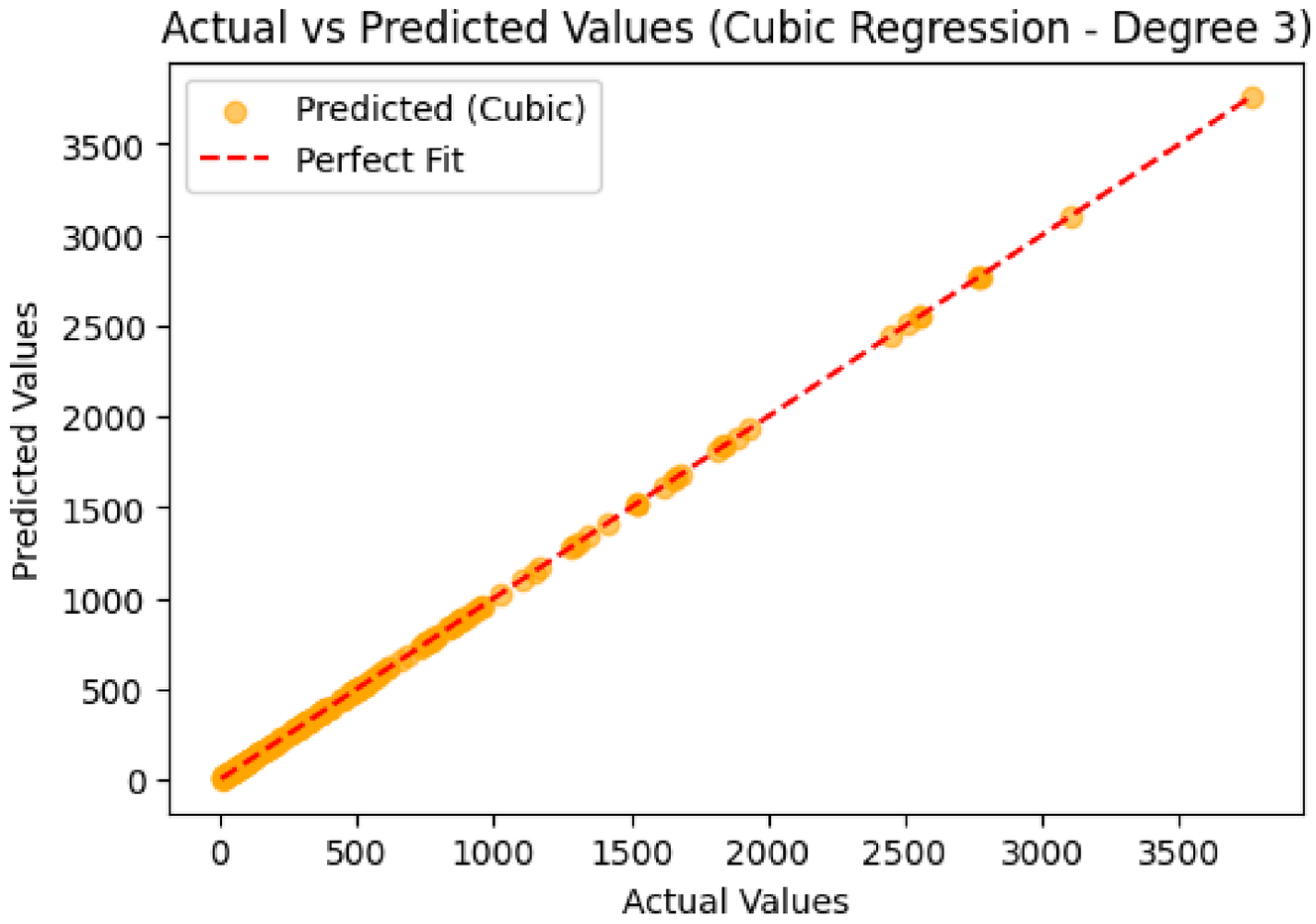


Quadratic Regression (Degree=2) Metrics:

- **Mean Squared Error (MSE):** 8.481969714733828e-26
- **Root Mean Squared Error (RMSE):** 2.912382137483649e-13
- **Mean Absolute Error (MAE):** 1.9135804052439197e-13

Model ini hampir sempurna dengan kesalahan yang sangat kecil, menunjukkan kemampuan baik dalam menangkap hubungan non-linear.

SCATTER PLOT



Cubic Regression (Degree=3) Metrics:

- **Mean Squared Error (MSE):** 2.3768049330450704e-25
- **Root Mean Squared Error (RMSE):** 4.875248642936143e-13
- **Mean Absolute Error (MAE):** 3.935252124165345e-13

Seperti quadratic regression, hasilnya hampir sempurna, tetapi model ini lebih kompleks dan berisiko overfitting.

GROUP 5

THANK YOU

FOR YOUR NICE ATTENTION

December 2024