Retail Analytics Conjoint Analysis

Spring 2025, University of West Georgia

By

Enita Omuvwie

Table of contents

Introduction

Purpose of Analysis:

- To perform Conjoint Analysis on the Grocery dataset
- To understand customer preferences and identify the key products that influence sales

Data Overview

- The dataset contains a total of 25 variables with 65535 entries

Key Columns:

- **Store_Id, Counter_No, Bill_No**: Identifiers for transactions.
- **Date and Time**: Date and time_bill for transaction timestamps.
- **Product Details**: Item_code, Item_Descp, Category, Sub_Category.
- **Sales Metrics**: Qty (Quantity), SP (Selling Price), Net_sales.
- **Discounts and Promotions**: Disc, Disc_2, Promo_Stat.
- **Financial Metrics**: CP (Cost Price), GM (Gross Margin).
- **Additional Cols**: WEEKDAY, Item_Flag, Free_bee, Division, Counter_Type.

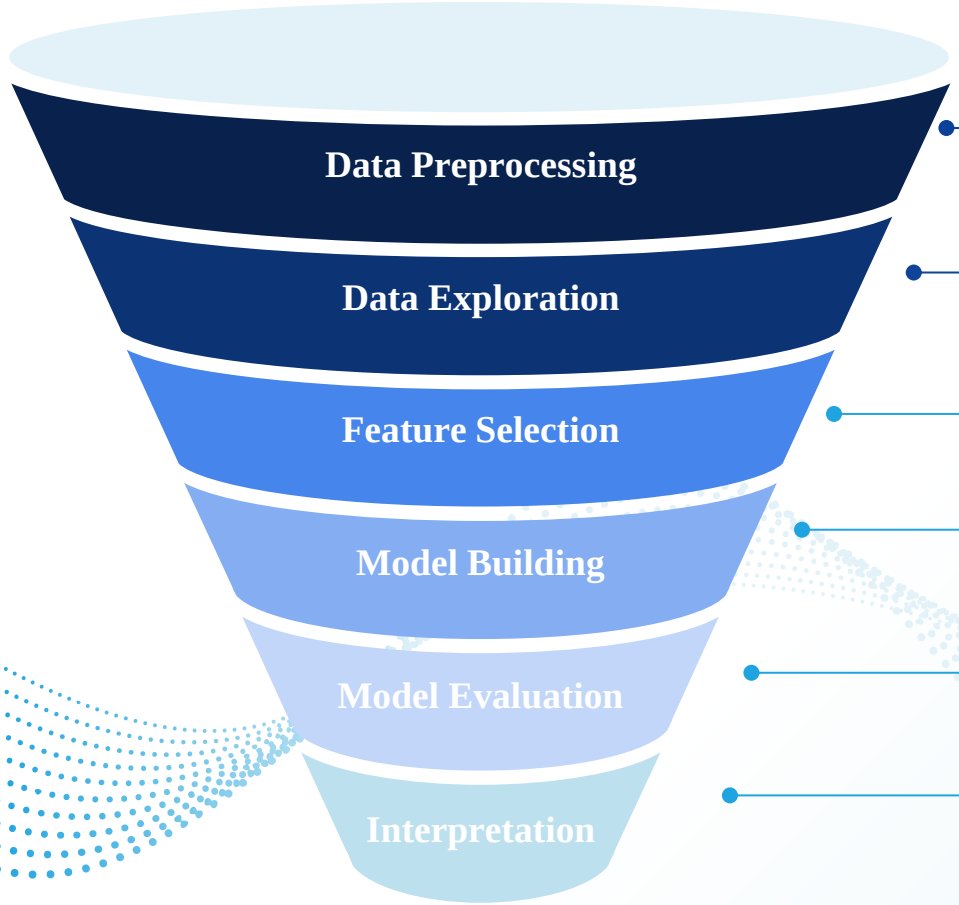
Objective

- **Understand Customer Preferences:** Identify key product attributes that influence sales
- **Analyze Sales Patterns:** Explore how different variables affect sales performance
- **Identify Relationships:** Determine correlations between product features and sales

Scope of Analysis

- **Data Exploration:** Uncovering patterns and trends
- **Feature Engineering:** Create derived features to enhance insights
- **Conjoint Analysis:** Use regression models to estimate importance of product attributes
- **Visualization:** Visualize to enhance insights to findings
- **Actionable Insights:** Provide recommendations based on analysis for informed business strategies

Methodology



- Import cleaned dataset
- Check for missing values and handle missing values

- Correlation analysis using hestmaps to identify relationships
- Check for relationship between variables and sales

- Select the best features that drive sales based on exploration

- Perform Linear regression with the best independent variables
- Visualize model results

- Assess the model accuracy using R-squared, Adjusted R-squared

- Interpret coefficients to understand the impact of indepenednt variables on sales
- Make data-driven recommendations

Code and Output : Data Preprocessing

Reading in the data, check the data overview

[1]:
```
# Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
```

[2]:
```
# Load the dataset
bill_df = pd.read_csv("bill_all.csv", encoding = 'latin1')

# Display basic information
print("Dataset Overview:")
print(bill_df.info())
print(bill_df.describe())
```

```
Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 65535 entries, 0 to 65534
Data columns (total 25 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Store_Id      65535 non-null  int64
 1   Counter_No    65535 non-null  int64
 2   Bill_No       65535 non-null  int64
 3   no            65535 non-null  int64
 4   time_bill     65535 non-null  object
 5   Date          65535 non-null  object
 6   WEEKDAY       65535 non-null  int64
 7   Item_code     65535 non-null  object
 8   Item_Descp    65535 non-null  object
 9   Qty           65535 non-null  float64
 10  SP            65535 non-null  float64
 11  Tax           65535 non-null  float64
 12  Promo_Stat    6837 non-null   object
 13  Disc          65535 non-null  float64
 14  Net_sales     65535 non-null  float64
 15  Disc_2        65535 non-null  float64
 16  Item_Flag     65535 non-null  object
 17  Free_bee      11915 non-null  object
 18  Division      65535 non-null  object
 19  CP            65515 non-null  float64
 20  GM            65515 non-null  float64
 21  Category      65535 non-null  object
 22  Sub_Category  65535 non-null  object
 23  Counter_Type  65535 non-null  object
 24  Bill_num      65535 non-null  float64
dtypes: float64(9), int64(5), object(11)
memory usage: 12.5+ MB
```

Code and Output : Data Preprocessing

Checking the data summary statistics of the variables in the data frame

	Store_Id	Counter_No	Bill_No	no	WEEKDAY
count	65535.0	65535.000000	6.553500e+04	65535.000000	65535.000000
mean	6.0	5.589288	3.055893e+12	8687.701778	4.054902
std	0.0	2.824479	2.824479e+10	4705.091579	2.627528
min	6.0	1.000000	3.010000e+12	1307.000000	1.000000
25%	6.0	3.000000	3.030000e+12	4742.000000	1.000000
50%	6.0	5.000000	3.050000e+12	9989.000000	4.000000
75%	6.0	8.000000	3.080000e+12	10940.000000	7.000000
max	6.0	10.000000	3.100000e+12	18991.000000	7.000000

	Qty	SP	Tax	Disc	Net_sales
count	65535.000000	65535.000000	65535.000000	65535.000000	65535.000000
mean	1.307589	65.680285	3.518230	0.344109	73.064017
std	3.530801	184.702927	16.675287	12.230128	210.875282
min	0.004000	0.000000	-3.560000	-4.700000	-32.000000
25%	1.000000	14.900000	0.000000	0.000000	13.900000
50%	1.000000	26.900000	0.960000	0.000000	28.750000
75%	1.000000	60.000000	3.440000	0.000000	65.000000
max	600.000000	22490.000000	2498.890000	2000.000000	22490.000000

	Disc_2	CP	GM	Bill_num
count	65535.000000	65515.000000	65515.000000	6.553500e+04
mean	0.344324	49.732306	16.134519	3.055893e+12
std	12.230100	151.023768	162.758591	2.824479e+10
min	0.000000	0.000000	-38511.000000	3.010000e+12
25%	0.000000	10.000000	2.000000	3.030000e+12
50%	0.000000	20.000000	6.000000	3.050000e+12
75%	0.000000	47.000000	14.000000	3.080000e+12
max	2000.000000	21451.000000	3360.000000	3.100000e+12

Code and Output : Data Preprocessing

Changing the date variables, create new features and handle missing values

[4]:
```
# Convert the date format
bill_df['Date'] = pd.to_datetime(bill_df['Date'], format='%d-%b-%y')
bill_df['Hour'] = pd.to_datetime(bill_df['time_bill'], format= '%H:%M:%S').dt.hour
bill_df['Day_of_Week'] = bill_df['Date'].dt.day_name()
bill_df['Month'] = bill_df['Date'].dt.month
```

[5]:
```
# Creating new features: Revenue, Total Discount
bill_df['Revenue'] = bill_df['Qty'] * bill_df['CP']
bill_df['Total_Discount'] = bill_df['Disc'] + bill_df['Disc_2']
```

[6]:
```
# Drop columns with high missing values
bill_df.drop(columns=['Promo_Stat', 'Free_bee'], inplace=True)

# Selecting only non-NaN observations for 'CP' and 'GM'
bill_df = bill_df[~bill_df.CP.isnull()]
bill_df = bill_df[~bill_df.GM.isnull()]

# Verify if missing values are handled
print("Missing Values After Handling:")
print(bill_df.isnull().sum())
```

```
Missing Values After Handling:
Store_Id          0
Counter_No        0
Bill_No           0
no                0
time_bill         0
Date              0
WEEKDAY           0
Item_code         0
Item_Descp        0
Qty               0
SP                0
Tax               0
Disc              0
Net_sales         0
Disc_2            0
Item_Flag         0
Division          0
CP                0
GM                0
Category          0
Sub_Category      0
Counter_Type      0
Bill_num          0
Hour              0
```

Code and Output : Data Preprocessing

Checking the Descriptive statistics of some numerical variables and changing the casing of Category

[77]:
```
# Checking the Descriptive Statistics for some variables
print(bill_df[['Qty', 'CP', 'GM', 'Revenue']].describe())
```

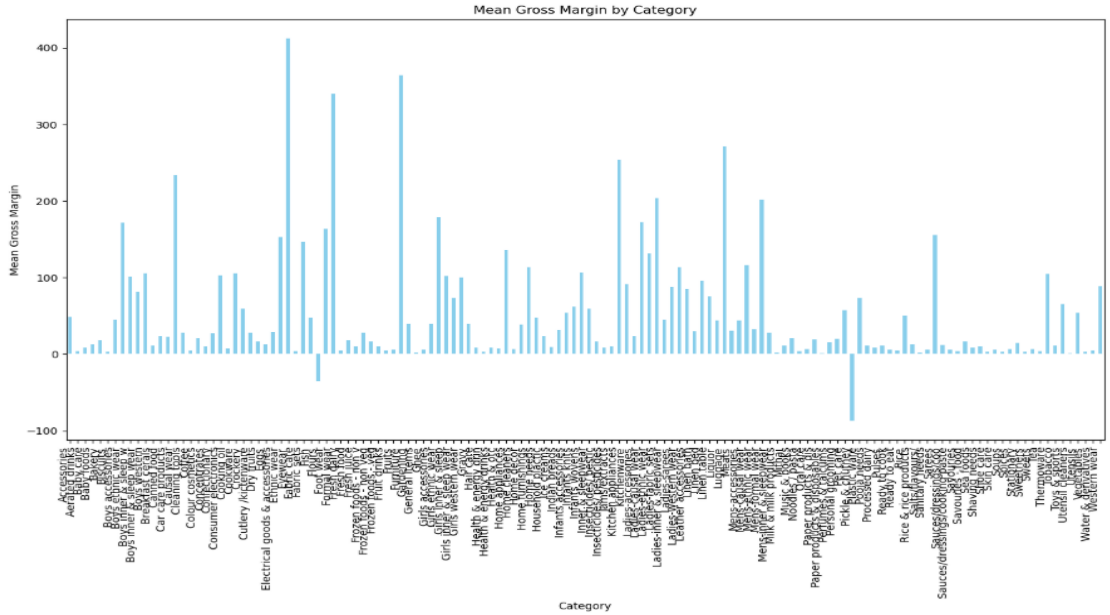
```
                Qty            CP            GM       Revenue
count  65515.000000  65515.000000  65515.000000  65515.000000
mean       1.307635     49.732306     16.134519     56.940827
std        3.531317    151.023768    162.758591    249.499797
min        0.004000      0.000000 -38511.000000      0.000000
25%        1.000000     10.000000      2.000000      9.000000
50%        1.000000     20.000000      6.000000     20.000000
75%        1.000000     47.000000     14.000000     51.000000
max      600.000000  21451.000000   3360.000000  38511.000000
```

[18]:
```
# Convert values in 'Category' to sentence case
bill_df['Category'] = bill_df['Category'].str.lower().str.capitalize()
```

Code and Output : Data Exploration

Mean Gross Margin Analysis by Category

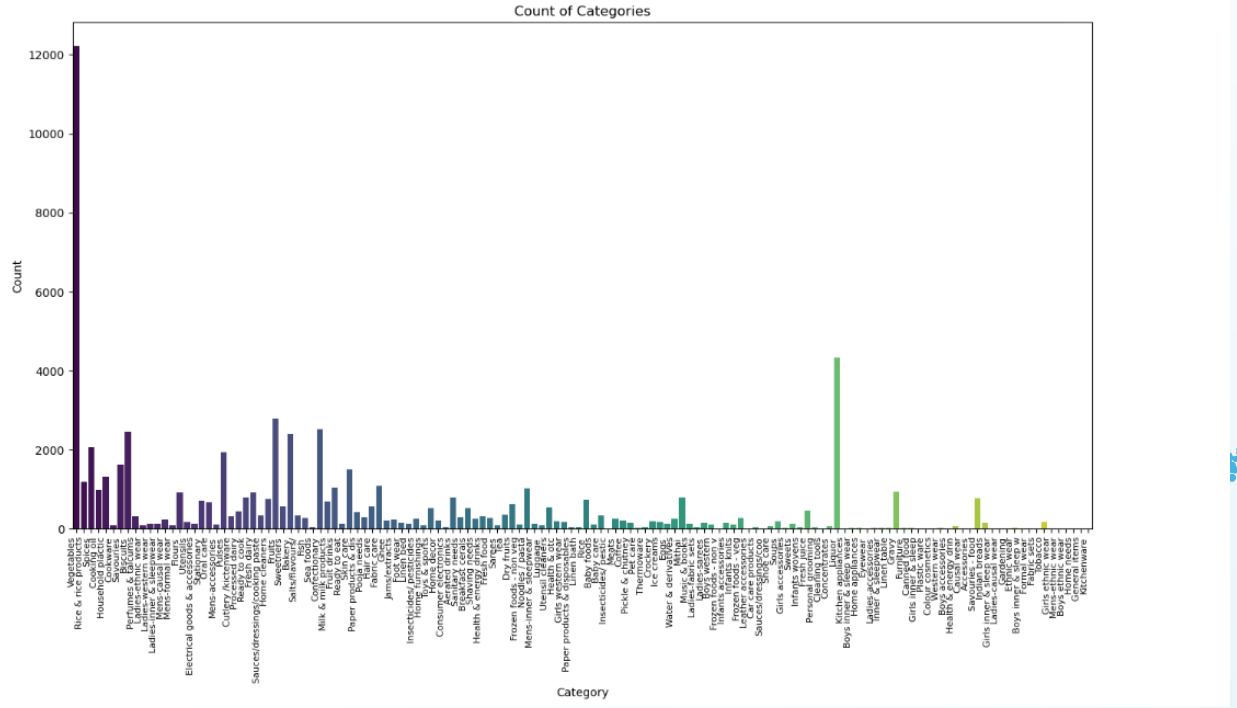
78]:
```
# Group Analysis by Category
# Grouping data
category_group = bill_df.groupby('Category')[['Qty', 'CP', 'GM', 'Revenue']].mean()

# Create a single figure before plotting
fig, ax = plt.subplots(figsize=(16, 8))

# Plot the bar chart on the same figure
category_group['GM'].plot(kind='bar', color='skyblue', ax=ax)

# Add title and labels
ax.set_title('Mean Gross Margin by Category')
ax.set_xlabel('Category')
ax.set_ylabel('Mean Gross Margin')

# Adjust x-axis labels
plt.xticks(rotation=90, ha='right')

# Show the plot
plt.show()
```



Code and Output : Data Exploration

Checking **Category Counts** to know the trends of product category sold(**Veggies is the most sold product**)

```
[27]: # Categories Counts
fig, ax = plt.subplots(figsize=(16, 8))

sns.countplot(data=bill_df, x='Category', palette='viridis')
plt.title('Count of Categories')
plt.xlabel('Category')
plt.ylabel('Count')
plt.xticks(rotation=90, ha='right', fontsize=8)
plt.show()
```

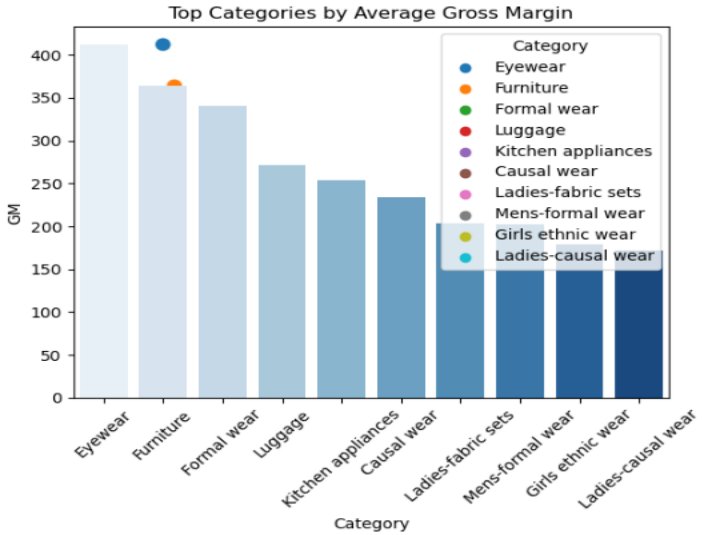


Code and Output : Data Exploration

Distribution of Top Product Categories vs. Average Gross Margin

```
# Aggregate data by category
category_summary = bill_df.groupby('Category').agg({'Qty': 'mean', 'GM': 'mean'}).reset_index()
print(category_summary.head())

top_categories = category_summary.nlargest(10, 'GM')
sns.scatterplot(data=top_categories, x='Qty', y='GM', hue='Category', palette='tab10', s=100)

sns.barplot(data=top_categories, x='Category', y='GM', palette='Blues')
plt.xticks(rotation=45)
plt.title('Top Categories by Average Gross Margin')
plt.show()
```

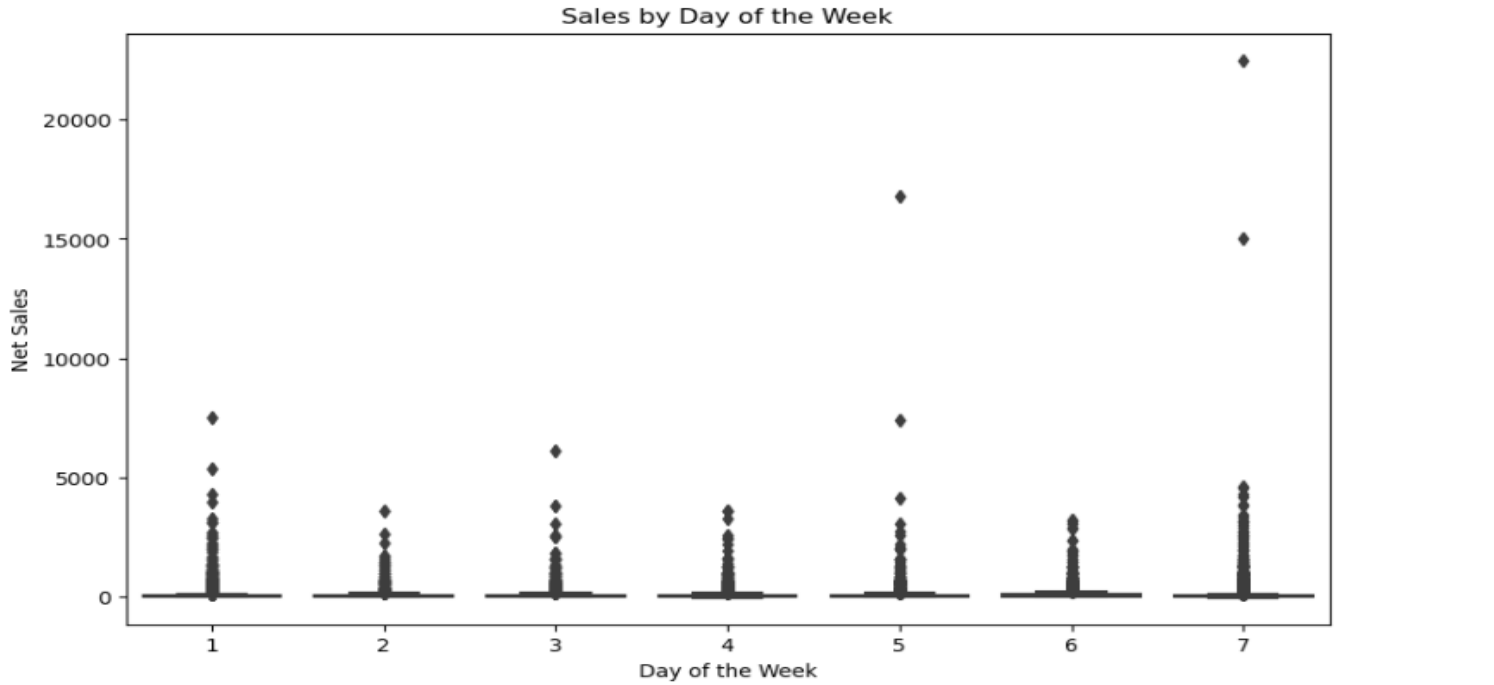
```
        Category       Qty         GM
0     Accessories  1.000000  48.454545
1  Aerated drinks  1.181472   4.345178
2       Baby care  1.055233   8.281977
3      Baby foods  1.000000  12.495238
4          Bakery  1.603354  18.499371
```



Code and Output : Data Exploration

Sales show significant outliers across all days but day 7 has the highest variability

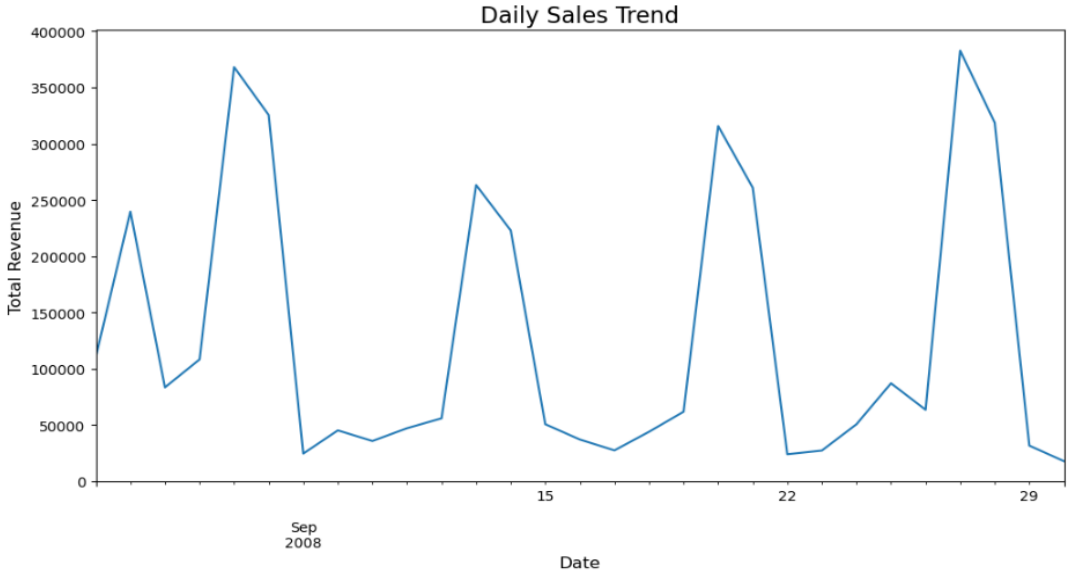
[39]:
```
# Sales by Day of the Week
plt.figure(figsize=(10, 6))
sns.boxplot(x='WEEKDAY', y='Net_sales', data=bill_df)
plt.title('Sales by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Net Sales')
plt.show()
```



Code and Output : Data Exploration

This shows daily sales trend with periodic spikes hence a fluctuation in the revenue

```
bill_df['Date'] = pd.to_datetime(bill_df['Date'])
daily_sales = bill_df.groupby('Date')['Revenue'].sum()

plt.figure(figsize=(12, 6))
daily_sales.plot()

plt.title('Daily Sales Trend', fontsize=16)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Total Revenue', fontsize=12)

plt.grid(False)

plt.show()
```

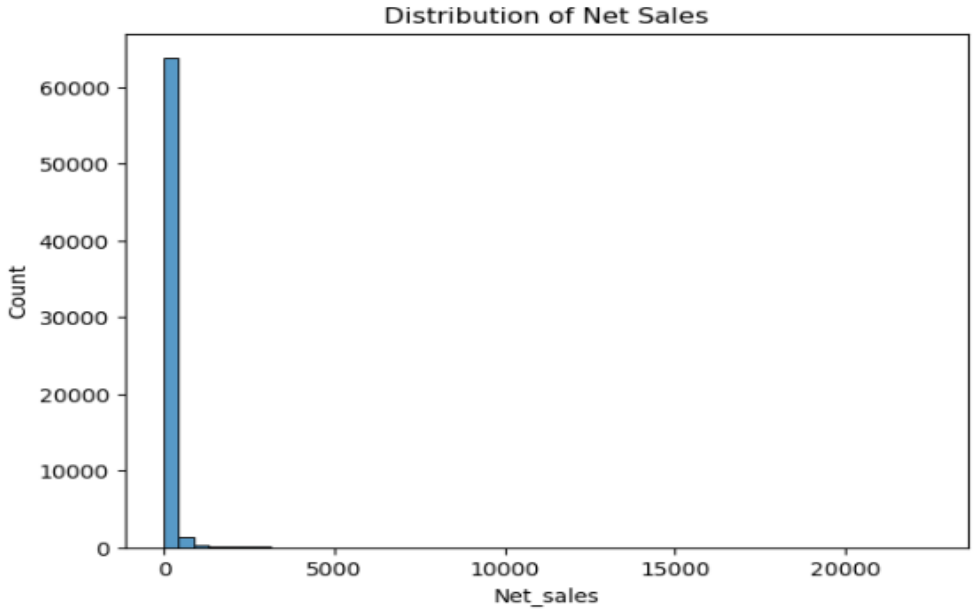


Code and Output : Data Exploration

Distribution of net sales is highly skewed with most values concentrated around zero.

```
# Sales distribution
plt.figure(figsize=(15, 5))
plt.subplot(1, 2, 1)
sns.histplot(data=bill_df, x='Net_sales', bins=50)
plt.title('Distribution of Net Sales')
```

```
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-packages/seaborn/_oldcore.py:111
  with pd.option_context('mode.use_inf_as_na', True):
Text(0.5, 1.0, 'Distribution of Net Sales')
```

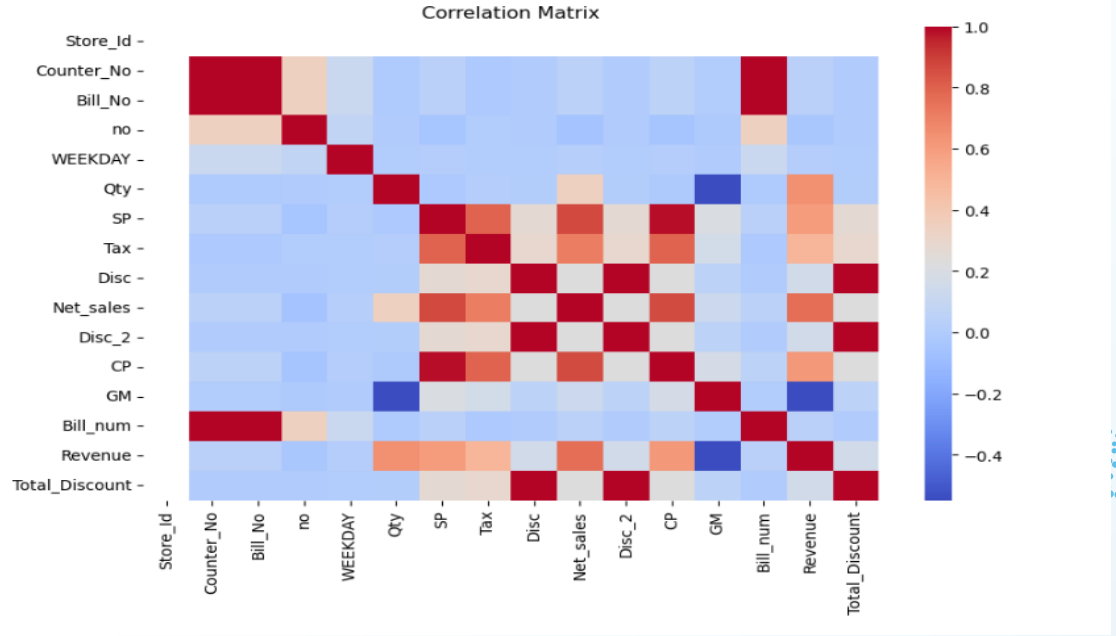


Code and Output : Data Exploration

Correlation matrix shows a varied relationship among variables with some strong positive and negative correlations

```
# Correlation heatmap
# First, select only numeric columns for correlation
numeric_df = bill_df.select_dtypes(include=['int64', 'float64'])

plt.figure(figsize=(10, 6))
sns.heatmap(numeric_df.corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Matrix")
plt.show()
```

```
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-packages/seaborn/matrix.py:260: FutureWarning: Format strings passed to
  annotation = ("{:" + self.fmt + "}").format(val)
```

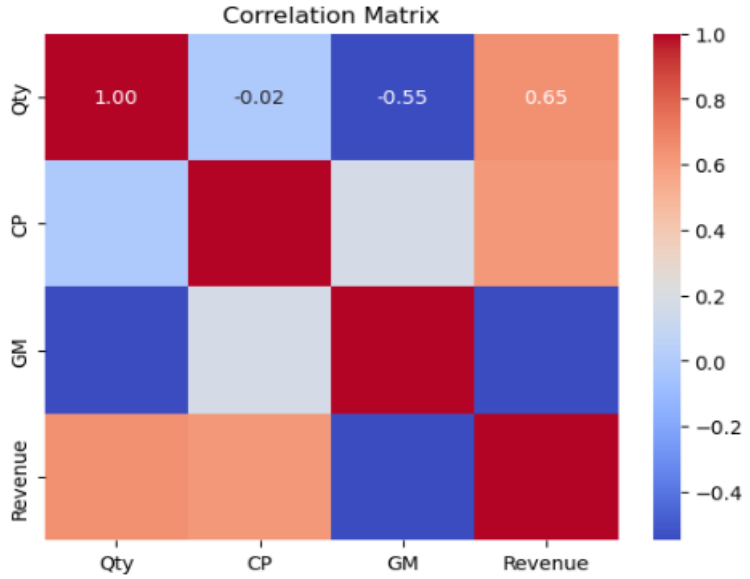


Code and Output : Data Exploration

Correlation matrix shows the strong positive correlation between quantity and revenue

```
correlation_matrix = bill_df[['Qty', 'CP', 'GM', 'Revenue']].corr()
print(correlation_matrix)
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```

```
                Qty        CP        GM   Revenue
Qty        1.000000 -0.015171 -0.547950  0.648077
CP        -0.015171  1.000000  0.172128  0.615893
GM        -0.547950  0.172128  1.000000 -0.545096
Revenue    0.648077  0.615893 -0.545096  1.000000
```



Code and Output : Data Modeling

Data split and regression parameters for the model

```
# Model split
X = bill_df[['WEEKDAY','Disc_2', 'CP', 'GM']

# Define dependent variable (y)
y = bill_df['Net_sales']

# Verify shapes
print("\nShape verification:")
print("X shape:", X.shape)
print("y shape:", y.shape)
print("Columns in X:", list(X.columns))
```

```
Shape verification:
X shape: (65515, 4)
y shape: (65515,)
Columns in X: ['WEEKDAY', 'Disc_2', 'CP', 'GM']
```

```
# Add constant term for regression
X = sm.add_constant(X)

# Fit OLS regression model
model = sm.OLS(y, X).fit()

# Print results
print(model.summary())

# Extract part-worth utilities (importance of attributes)
coefficients = model.params
print("\nEstimated Part-Worths:\n", coefficients)
```

Code and Output : Data Modeling

OLS Regression results indicating that 74.4% of the variance in net sales was explained (**R-squared = 0.7444)**

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              Net_sales   R-squared:                       0.744
Model:                            OLS   Adj. R-squared:                  0.744
Method:                 Least Squares   F-statistic:                 4.756e+04
Date:                Thu, 10 Apr 2025   Prob (F-statistic):               0.00
Time:                        22:16:14   Log-Likelihood:            -3.9894e+05
No. Observations:               65515   AIC:                         7.979e+05
Df Residuals:                   65510   BIC:                         7.979e+05
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         12.2492      0.777     15.756      0.000      10.725      13.773
WEEKDAY        0.3706      0.159      2.335      0.020       0.060       0.682
Disc_2         0.5838      0.035     16.679      0.000       0.515       0.652
CP             1.1976      0.003    416.709      0.000       1.192       1.203
GM            -0.0293      0.003    -11.273      0.000      -0.034      -0.024
==============================================================================
Omnibus:                   227772.695   Durbin-Watson:                   1.432
Prob(Omnibus):                  0.000   Jarque-Bera (JB):     226021286665.330
Skew:                          66.150   Prob(JB):                         0.00
Kurtosis:                    9101.379   Cond. No.                         333.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Estimated Part-Worths:
 const      12.249228
WEEKDAY      0.370644
Disc_2       0.583819
CP           1.197649
GM          -0.029324
dtype: float64
```

Code and Output : Data Model Interpretation

Key Discovery

- **Weekday**: Positive coeff. **(0.3706)** which means **sales increase** on certain weekdays with significant p-value (**0.020**)
- **Disc_2**: Positive coeff. (**0.5838**) indicating this discount type drives **higher sales**, it is also highly significant (**p<0.001**)
- **CP(Cost Price)**: **Strong positive** impact on **sales** (coeff. = **1.1976**), highly significant (**p < 0.001**)
- **GM(Gross Margin)**: Negative coeff. (**-0.0293**) indicating the higher the gross margin, the **lower** the **sales**, highly significant(**p<0.001**)

Interpretation:

This model suggests that **specific days of the week**, **discount strategies**, **cost pricing** significantly influence the **net sales** while higher **gross margins** reduce sales.

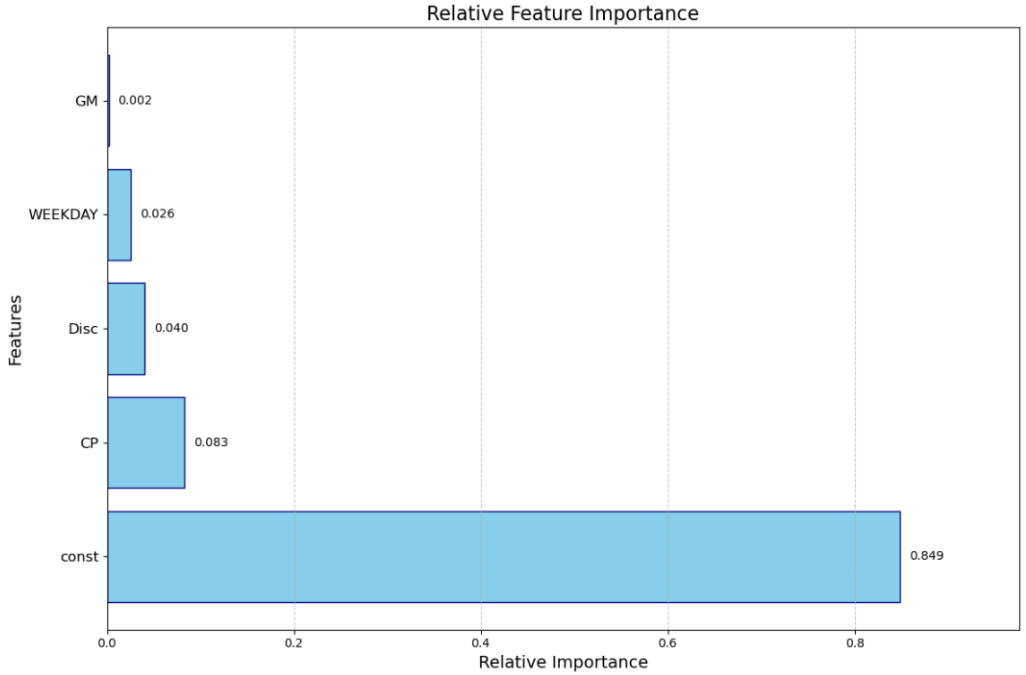
Code and Output : Data Interpretation

Calculation of feature importance based on model coefficients

```python
# Calculate feature importance based on model coefficients
def calculate_feature_importance(model):
    # Get model coefficients
    coefficients = model.params

    # Group coefficients by feature (handling both single coefficients and dummy variables)
    feature_coeffs = {}
    for key, coeff in coefficients.items():
        # Split on underscore to separate feature name from level
        parts = key.split('_')
        feature = parts[0]

        if feature not in feature_coeffs:
            feature_coeffs[feature] = []
        feature_coeffs[feature].append(coeff)

    # Calculate importance based on coefficient ranges or absolute values
    importance = {}
    for feature, coeffs in feature_coeffs.items():
        if len(coeffs) > 1:
            # If multiple coefficients exist for a feature (e.g., dummy variables)
            # Use range (max - min) as importance
            importance[feature] = max(coeffs) - min(coeffs)
        else:
            # If only one coefficient, use absolute value
            importance[feature] = abs(coeffs[0])

    # Calculate relative importance
    total_importance = sum(importance.values())
    relative_importance = {k: v/total_importance for k, v in importance.items()}

    return relative_importance

# Calculate feature importance
importance = calculate_feature_importance(model)

# Create DataFrame for plotting
importance_df = pd.DataFrame(
    list(importance.items()),
    columns=['Feature', 'Importance']
).sort_values(by='Importance', ascending=False)

# Create the plot
fig, ax = plt.subplots(figsize=(12, 8))
bars = ax.barh(
    np.arange(len(importance_df)),
    importance_df['Importance'],
    color='skyblue',
    edgecolor='navy'
)

# Add labels and formatting
ax.set_title('Relative Feature Importance', fontsize=16)
ax.set_xlabel('Relative Importance', fontsize=14)
ax.set_ylabel('Features', fontsize=14)
ax.set_yticks(np.arange(len(importance_df)))
ax.set_yticklabels(importance_df['Feature'], fontsize=12)
ax.grid(axis='x', linestyle='--', alpha=0.7)

# Add value labels
for i, v in enumerate(importance_df['Importance']):
    ax.text(v + 0.01, i, f'{v:.3f}', va='center')

# Set x-axis limit to make room for labels
ax.set_xlim(0, max(importance_df['Importance']) * 1.15)

plt.tight_layout()
plt.show()

# Print importance values
print("\nFeature Importance Ranking:")
for i, (feature, importance) in enumerate(zip(importance_df['Feature'], importance_df['Importance'])):
    print(f"{i+1}. {feature}: {importance:.3f} ({importance*100:.1f}%)")
```

Code and Output : Data Interpretation

This is the Relative Feature Importance Ranking for the model the constant having highest relevant importance. **Cost Price** had an **8.3%** significant impact on the model**. Discount and weekdays** had moderate impact, and **Gross margin** has minimal impact.




```
Feature Importance Ranking:
1. const: 0.849 (84.9%)
2. CP: 0.083 (8.3%)
3. Disc: 0.040 (4.0%)
4. WEEKDAY: 0.026 (2.6%)
5. GM: 0.002 (0.2%)
```

Conclusion

- **Key Influencers: Cost Price**, **Discounts, Weekdays**, and **Gross Margin** influenced customer decisions
- **Attribute Importance:** Determine the relative importance of each attribute in decision making
- **Part-Worth Utilities:** Assessing value customers place on different levels of the attributes
- **Market Simulation:** This helped in predicting how the changes in attributes affect customer preferences and market share
- **Recommendations:** These insights can be used to optimize marketing strategies, product pricing and offerings.

Thank you

Any questions?

You can find me at

- **eomuvwi1@my.westga.edu**