# Mall_sales_Analytics

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objects as go
```

In [2]:
```
customer = pd.read_excel(r"C:\Users\Ruby\OneDrive\Documents\pyton related files\
sales = pd.read_excel(r"C:\Users\Ruby\OneDrive\Documents\pyton related files\sal
mall = pd.read_excel(r"C:\Users\Ruby\OneDrive\Documents\pyton related files\shop
```

In [4]:
```
print(list(customer.columns))
print(list(sales.columns))
print(list(mall.columns))
```

```
['customer_id', 'gender', 'age', 'payment_method']
['invoice_no', 'customer_id', 'category', 'quantity', 'invoice date', 'price', 's
hopping_mall']
['shopping_mall', 'construction_year', 'area (sqm)', 'location', 'store_count']
```

In [5]:
```
# Customer
customer.head()
```

Out[5]:

| | customer_id | gender | age | payment_method |
|---|---|---|---|---|
| **0** | C241288 | Female | 28.0 | Credit Card |
| **1** | C111565 | Male | 21.0 | Debit Card |
| **2** | C266599 | Male | 20.0 | Cash |
| **3** | C988172 | Female | 66.0 | Credit Card |
| **4** | C189076 | Female | 53.0 | Cash |

In [6]:
```
#sales
sales.head()
```

Out[6]:

| | invoice_no | customer_id | category | quantity | invoice date | price | shopping_mall |
|---|---|---|---|---|---|---|---|
| **0** | I138884 | C241288 | Clothing | 5 | 05/08/2022 | 1500.40 | South Coast Plaza |
| **1** | I317333 | C111565 | Shoes | 3 | 12/12/2021 | 1800.51 | Beverly Center |
| **2** | I127801 | C266599 | Clothing | 1 | 09/11/2021 | 300.08 | Westfield Century City |
| **3** | I173702 | C988172 | Shoes | 5 | 05/16/2021 | 3000.85 | Stanford Shopping Center |
| **4** | I337046 | C189076 | Books | 4 | 10/24/2021 | 60.60 | South Coast Plaza |

In [7]:
```
#mall
mall.head()
```

Out[7]:

| | shopping_mall | construction_year | area (sqm) | location | store_count |
|---|---|---|---|---|---|
| **0** | South Coast Plaza | 1967 | 250000 | Costa Mesa | 270 |
| **1** | Westfield Valley Fair | 1986 | 220000 | Santa Clara | 230 |
| **2** | The Grove | 2002 | 56000 | Los Angeles | 140 |
| **3** | Westfield Century City | 1964 | 133000 | Los Angeles | 200 |
| **4** | Beverly Center | 1982 | 111000 | Los Angeles | 160 |

In [8]:
```
customer.dtypes
```

Out[8]:
```
customer_id        object
gender             object
age               float64
payment_method     object
dtype: object
```

In [11]:
```
print(customer.shape)
customer.isna().sum()
```

```
(99457, 4)
```

Out[11]:
```
customer_id         0
gender              0
age               119
payment_method      0
dtype: int64
```

In [12]:
```
sales.head()
```

Out[12]:

| | invoice_no | customer_id | category | quantity | invoice date | price | shopping_mall |
|---|---|---|---|---|---|---|---|
| **0** | I138884 | C241288 | Clothing | 5 | 05/08/2022 | 1500.40 | South Coast Plaza |
| **1** | I317333 | C111565 | Shoes | 3 | 12/12/2021 | 1800.51 | Beverly Center |
| **2** | I127801 | C266599 | Clothing | 1 | 09/11/2021 | 300.08 | Westfield Century City |
| **3** | I173702 | C988172 | Shoes | 5 | 05/16/2021 | 3000.85 | Stanford Shopping Center |
| **4** | I337046 | C189076 | Books | 4 | 10/24/2021 | 60.60 | South Coast Plaza |

In [13]:
```
print(sales.shape)
sales.isna().sum()
```

```
(99457, 7)
```

Out[13]:
```
invoice_no       0
customer_id      0
category         0
quantity         0
invoice date     0
price            0
shopping_mall    0
dtype: int64
```

In [14]:
```
mall.head()
```

Out[14]:

| | shopping_mall | construction_year | area (sqm) | location | store_count |
|---|---|---|---|---|---|
| **0** | South Coast Plaza | 1967 | 250000 | Costa Mesa | 270 |
| **1** | Westfield Valley Fair | 1986 | 220000 | Santa Clara | 230 |
| **2** | The Grove | 2002 | 56000 | Los Angeles | 140 |
| **3** | Westfield Century City | 1964 | 133000 | Los Angeles | 200 |
| **4** | Beverly Center | 1982 | 111000 | Los Angeles | 160 |

In [15]:
```
print(mall.shape)
sales.isna().sum()
```

```
(10, 5)
```

Out[15]:
```
invoice_no       0
customer_id      0
category         0
quantity         0
invoice date     0
price            0
shopping_mall    0
dtype: int64
```

# converting columns into required format

In [16]:
```
customer['customer_id'] = customer['customer_id'].astype(str)  # Ensure it's tre
customer['gender'] = customer['gender'].astype('category')  # Convert to categor
customer['age'] = customer['age'].astype(float)  # Ensure it's treated as a floa
customer['payment_method'] = customer['payment_method'].astype('category')  # Co
sales['invoice_no'] = sales['invoice_no'].astype(str)  # Ensure it's a string
sales['customer_id'] = sales['customer_id'].astype(str)  # Ensure it's a string
sales['category'] = sales['category'].astype('category')  # Convert to category
sales['quantity'] = sales['quantity'].astype(int)  # Ensure it's treated as an i
sales['invoice date'] = pd.to_datetime(sales['invoice date'], format='%m/%d/%Y')
sales['price'] = sales['price'].astype(float)  # Ensure it's treated as a float
sales['shopping_mall'] = sales['shopping_mall'].astype('category')  # Convert to
mall['shopping_mall'] = mall['shopping_mall'].astype(str)  # Ensure it's treated
mall['construction_year'] = mall['construction_year'].astype(int)  # Ensure it's
mall['area (sqm)'] = mall['area (sqm)'].astype(int)  # Ensure it's treated as an
mall['location'] = mall['location'].astype('category')  # Convert to category fo
mall['store_count'] = mall['store_count'].astype(int)  # Ensure it's treated as
```

In [17]:
```
# rechecking the dtypes
customer.dtypes
```

Out[17]:
```
customer_id         object
gender            category
age                float64
payment_method    category
dtype: object
```

In [18]:
```
sales.dtypes
```

Out[18]:
```
invoice_no              object
customer_id             object
category              category
quantity                 int32
invoice date     datetime64[ns]
price                   float64
shopping_mall         category
dtype: object
```

In [19]:
```
mall.dtypes
```

Out[19]:
```
shopping_mall          object
construction_year       int32
area (sqm)              int32
location             category
store_count             int32
dtype: object
```

In [20]:
```
# finding oout the customer id whose age is missing  and store only unique ids
no_age_customer=customer.loc[customer["age"].isna()].customer_id.unique()
no_age_customer
```

Out[20]:
```
array(['C157070', 'C177975', 'C830576', 'C807389', 'C277842', 'C283524',
       'C201228', 'C253769', 'C549692', 'C116130', 'C229540', 'C902306',
       'C146061', 'C310532', 'C648077', 'C273902', 'C212146', 'C263461',
       'C164478', 'C162454', 'C335397', 'C333501', 'C250346', 'C229859',
       'C162922', 'C682304', 'C293881', 'C163893', 'C211846', 'C151577',
       'C118067', 'C309008', 'C634716', 'C203482', 'C325317', 'C925215',
       'C223910', 'C125439', 'C691173', 'C818343', 'C197390', 'C252192',
       'C330964', 'C603291', 'C122617', 'C290866', 'C192886', 'C111043',
       'C159207', 'C807983', 'C440786', 'C218903', 'C209283', 'C258704',
       'C212399', 'C655608', 'C366922', 'C218137', 'C226088', 'C464686',
       'C375611', 'C778818', 'C315397', 'C216722', 'C280379', 'C252953',
       'C310879', 'C541501', 'C120134', 'C964360', 'C212883', 'C204163',
       'C150772', 'C312842', 'C179296', 'C790169', 'C385618', 'C570517',
       'C282931', 'C103303', 'C456292', 'C917070', 'C925247', 'C229449',
       'C156681', 'C100748', 'C140541', 'C141197', 'C118877', 'C306970',
       'C157764', 'C322178', 'C698875', 'C160668', 'C709467', 'C591675',
       'C291818', 'C800874', 'C224840', 'C290927', 'C713735', 'C108981',
       'C453058', 'C160138', 'C565683', 'C300189', 'C224547', 'C908463',
       'C104048', 'C212717', 'C788921', 'C672952', 'C322295', 'C863534',
       'C122968', 'C290775', 'C574895', 'C437895', 'C525919'],
      dtype=object)
```

In [21]:
```
print("Few Customer_id's with Nan age ",no_age_customer[0:5])
```

```
Few Customer_id's with Nan age  ['C157070' 'C177975' 'C830576' 'C807389' 'C27784
2']
```

In [22]:
```
#This tells you that 119 out of 99457
#sales are linked to customers whose age is not available — useful
#for understanding data quality and how it might affect your analysis.
```

```
print("Number of enteries in SALES Table with customer_id that has age=NaN in th
      ,sales.loc[sales["customer_id"].isin(no_age_customer)].shape[0])
print("Total number of Sales row =",sales.shape[0])
```

```
Number of enteries in SALES Table with customer_id that has age=NaN in the CUSTOM
ER tabel=  119
Total number of Sales row = 99457
```

I plan on replacing the Nan values in the "age" column of the [customer] Table based on the AVERAGE age of the "gender" in each "shopping_mall".

This way we replace the Nan value with the most probable value on mall basis, rather than taking the mean of the entire sample.

In [27]:
```
# Step 1: Determine the mall for each customer by joining `sales` and `customer`

# Perform an inner join to ensure that only customers present in `sales` are con

customer_sales = pd.merge(sales[['customer_id', 'shopping_mall']], customer, on=

# Step 2: Calculate average age for each gender within each mall

age_averages = customer_sales.groupby(['shopping_mall', 'gender'])['age'].mean()
age_averages = age_averages.rename(columns={'age':'average_age'})
# Step 3: Map average ages back to customer DataFrame

# Merge the `age_averages` DataFrame back with `customer_sales` to make the aver
customer_sales = pd.merge(customer_sales,age_averages, on = ['gender','shopping_

# Step 4: Update the `age` in the original `customer` DataFrame
# Create a dictionary to map customer_id to their corresponding average age base

age_replacements = customer_sales.set_index('customer_id')['average_age'].to_dic


# Replace NaN ages in the `customer` DataFrame based on the mapping
customer['age'] = customer.apply(
    lambda row: age_replacements[row['customer_id']]
                if pd.isna(row['age']) and row['customer_id'] in age_replacement
                else row['age'],
    axis=1
)

# Convert age to integer type to ensure whole number
customer['age']= customer['age'].astype(int)

customer.loc[customer["customer_id"]=="C807389"]
```

```
C:\Users\Ruby\AppData\Local\Temp\ipykernel_25472\884812163.py:9: FutureWarning: T
he default of observed=False is deprecated and will be changed to True in a futur
e version of pandas. Pass observed=False to retain current behavior or observed=T
rue to adopt the future default and silence this warning.
  age_averages = customer_sales.groupby(['shopping_mall', 'gender'])['age'].mean
().round().reset_index()
```

Out[27]:

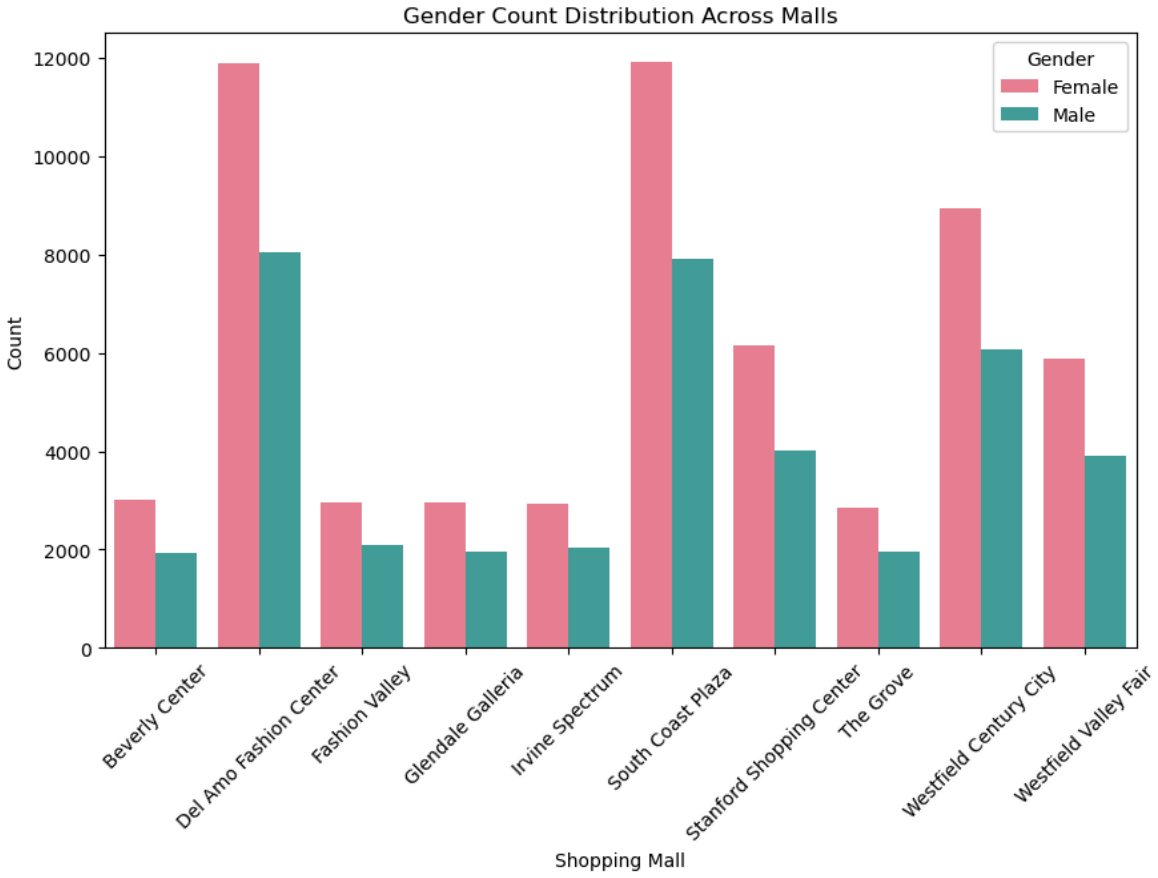
| | customer_id | gender | age | payment_method |
|---|---|---|---|---|
| **94** | C807389 | Female | 44 | Debit Card |

In [29]:
```
# checking null values are replaced accurately
customer.isna().sum()
```

Out[29]:
```
customer_id       0
gender            0
age               0
payment_method    0
dtype: int64
```

# EDA

In [36]:
```
def plot_gender_distribution_by_mall(customer, sales) :
    # Join customer and sales to associate each customer with a mall
    customer_sales = pd.merge(sales[['customer_id', 'shopping_mall']], customer,

    # Plot
    plt.figure(figsize=(10, 6))
    sns.countplot(data=customer_sales, x='shopping_mall', hue='gender',palette =
    plt.title("Gender Count Distribution Across Malls")
    plt.xlabel("Shopping Mall")
    plt.ylabel("Count")
    plt.xticks(rotation=45)
    plt.legend(title='Gender')
    plt.show()

plot_gender_distribution_by_mall(customer, sales)
```



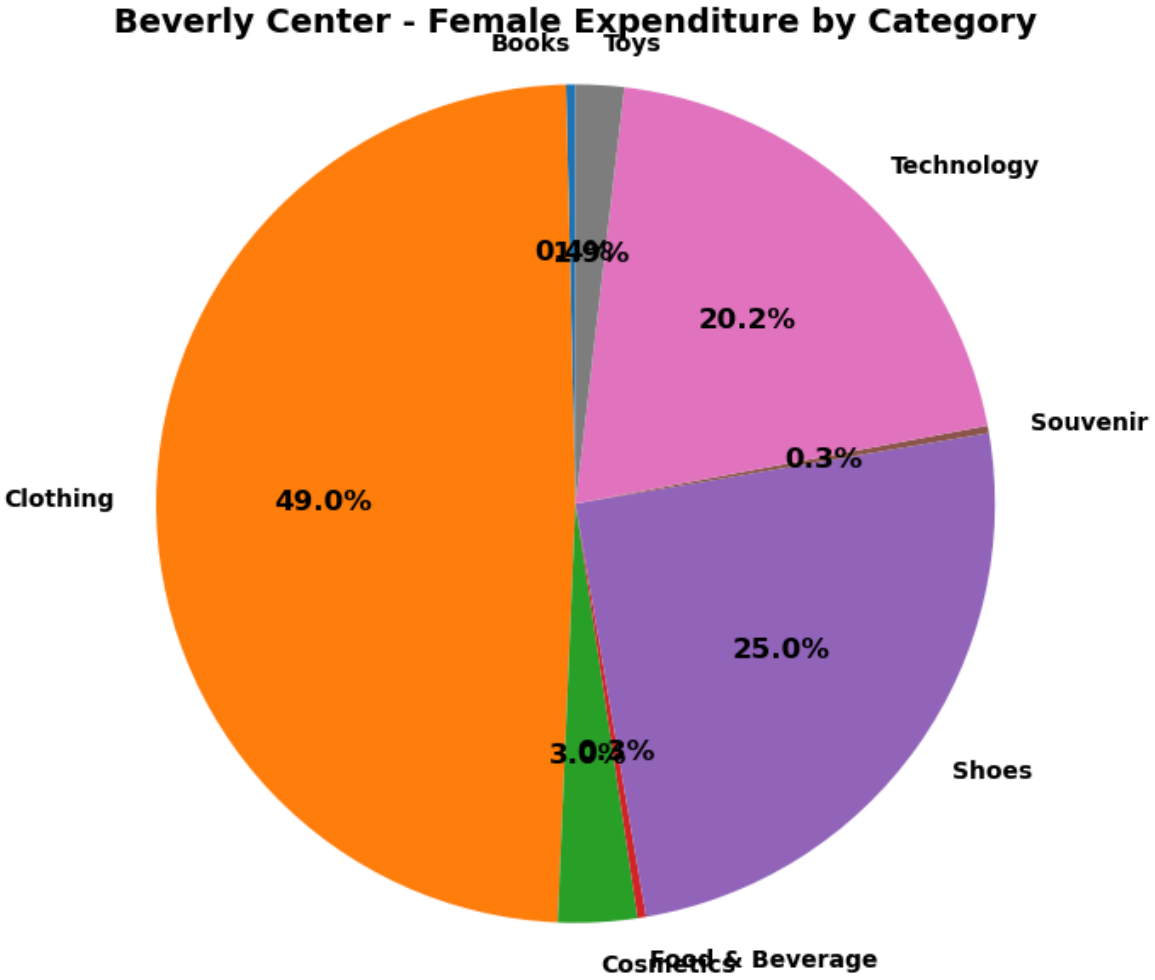

Generally we can see that female visits the mall more

In [40]:
```
def visualize_spending_by_category_for_mall(customer, sales, mall, target_mall):
    # Step 1: Merge customer and sales dataframes
    merged_data = pd.merge(sales, customer, on='customer_id', how='inner')

    # Merge with mall dataframe
    merged_data = pd.merge(merged_data, mall, on='shopping_mall', how='inner')

    # Step 2: Calculate total expenditure (quantity * price)
    merged_data['total_spent'] = merged_data['quantity'] * merged_data['price']

    # Step 3: Filter for the specific mall
    mall_data = merged_data[merged_data['shopping_mall'] == target_mall]

    # Step 4: Group by gender and category, and sum the expenditure
    grouped_data = mall_data.groupby(['gender', 'category'])['total_spent'].sum(

    # Step 5: Visualize with pie charts and print values
    for gender in grouped_data['gender'].unique():
        gender_data = grouped_data[grouped_data['gender'] == gender]

        # Calculate percentage values without modifying the DataFrame directly
        total_spent = gender_data['total_spent'].sum()
        percentage_values = (gender_data['total_spent'] / total_spent) * 100
        gender_data = gender_data.assign(percentage=percentage_values)  # Create

        # Plot pie chart for each category in the mall and gender combination
        plt.figure(figsize=(7, 7))
        wedges, texts, autotexts = plt.pie(
            gender_data['total_spent'],
            labels=gender_data['category'],
            autopct='%1.1f%%',
            startangle=90
        )

        # Bold the category labels
        for text in texts:
            text.set_fontweight('bold')

        # Increase the font size of percentage labels for better readability
        for autotext in autotexts:
            autotext.set_fontsize(12)
            autotext.set_fontweight('bold')

        # Print the pie chart values in a readable format
        print(f'\n{gender} in {target_mall} spends on:')
        for index, row in gender_data.iterrows():
            print(f"{row['category']}: {row['percentage']:.2f}%")

        plt.title(f'{target_mall} - {gender} Expenditure by Category', fontweigh
        plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a c
        plt.show()
```

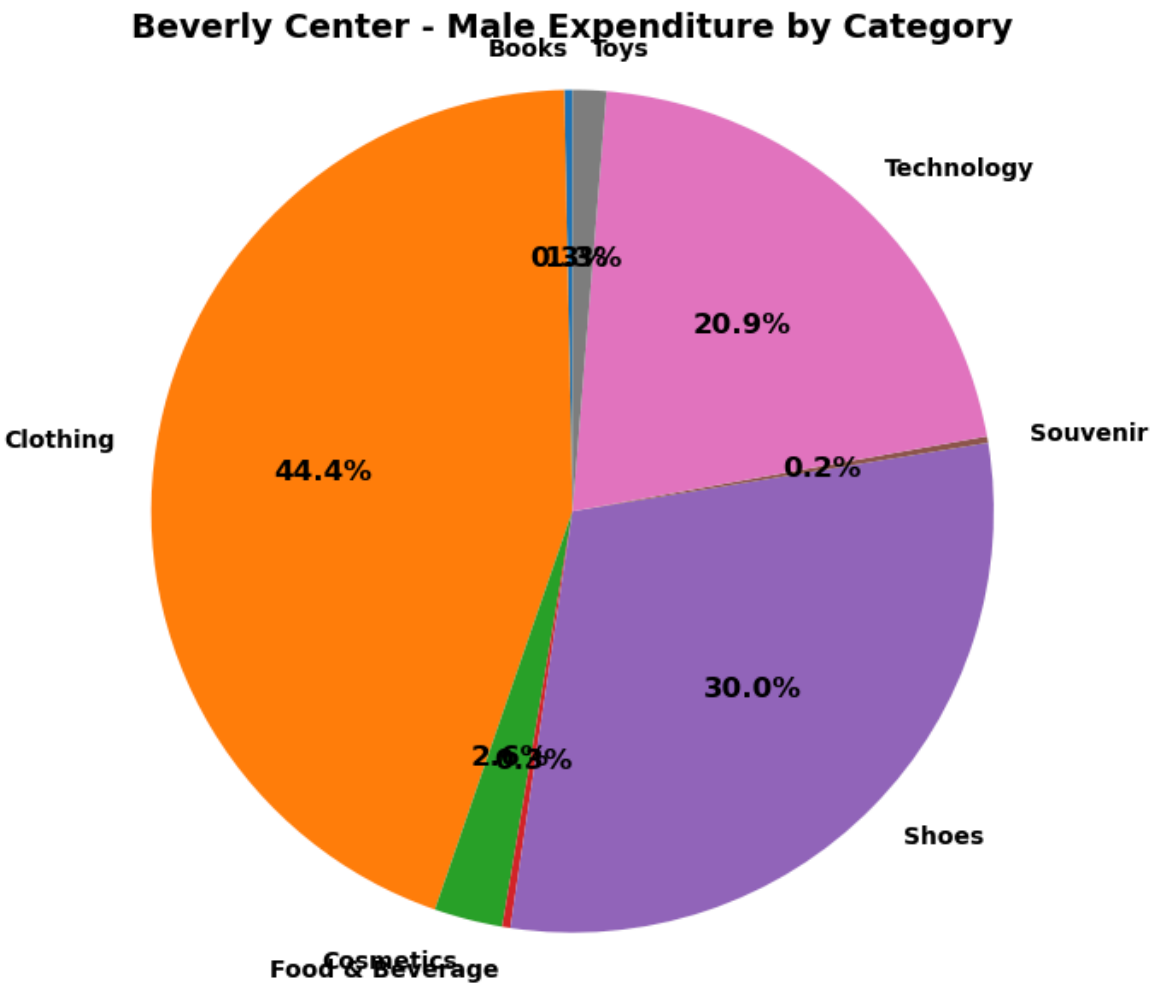
In [42]:
```
visualize_spending_by_category_for_mall(customer, sales, mall,'Beverly Center' )
```

```
Female in Beverly Center spends on:
Books: 0.36%
Clothing: 48.96%
Cosmetics: 3.04%
Food & Beverage: 0.32%
Shoes: 25.01%
Souvenir: 0.28%
Technology: 20.17%
Toys: 1.85%
```

```
C:\Users\Ruby\AppData\Local\Temp\ipykernel_25472\3082728283.py:15: FutureWarning:
The default of observed=False is deprecated and will be changed to True in a futu
re version of pandas. Pass observed=False to retain current behavior or observed=
True to adopt the future default and silence this warning.
  grouped_data = mall_data.groupby(['gender', 'category'])['total_spent'].sum().r
eset_index()
```




```
Male in Beverly Center spends on:
Books: 0.31%
Clothing: 44.39%
Cosmetics: 2.62%
Food & Beverage: 0.31%
Shoes: 29.96%
Souvenir: 0.25%
Technology: 20.87%
Toys: 1.29%
```

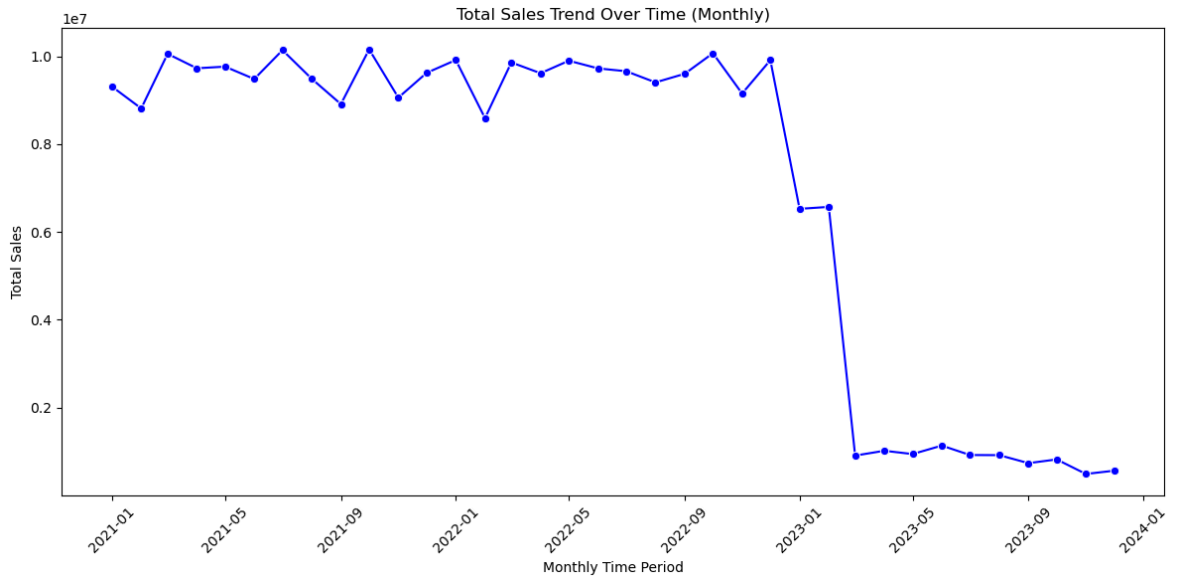



# working time

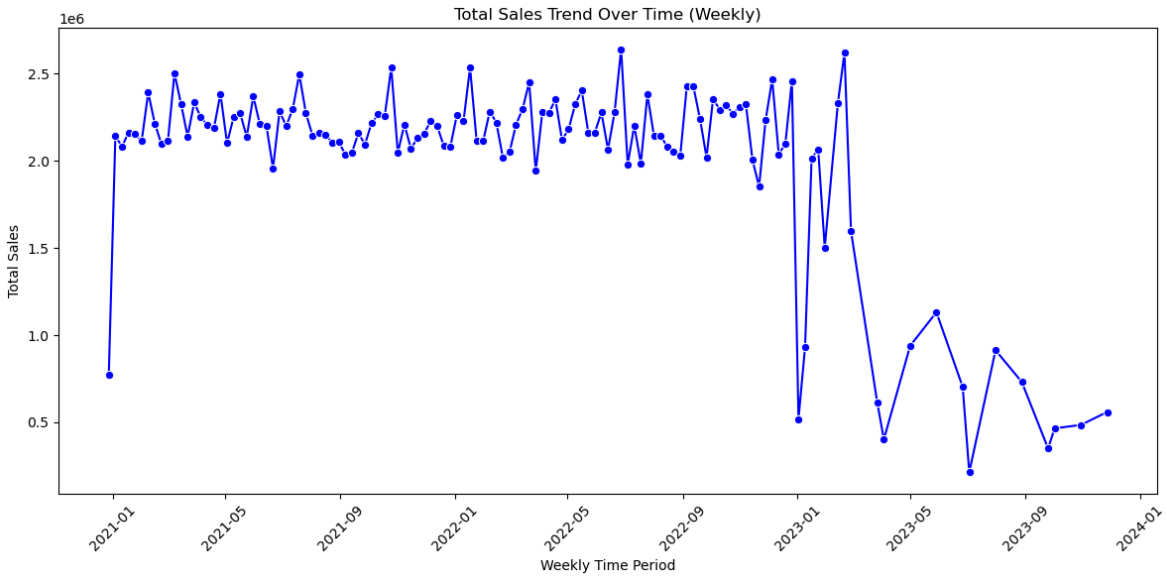
In [45]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

def plot_sales_trends(sales, timeframe='monthly', slider_value=None):
    # Convert 'invoice_date' to datetime
    sales['invoice_date'] = pd.to_datetime(sales['invoice date'], format='%d/%m/

    # Calculate total sales (quantity * price)
    sales['total_sales'] = sales['quantity'] * sales['price']

    # Create 'week' and 'month' columns based on 'invoice_date'
    if timeframe == 'monthly':
        sales['month'] = sales['invoice_date'].dt.to_period('M')
        sales_agg = sales.groupby('month')['total_sales'].sum().reset_index()
        sales_agg['month'] = sales_agg['month'].dt.to_timestamp()  # Convert to
        x_data = sales_agg['month']
        y_data = sales_agg['total_sales']

    elif timeframe == 'weekly':
        sales['week'] = sales['invoice_date'].dt.to_period('W').apply(lambda r:
        sales_agg = sales.groupby('week')['total_sales'].sum().reset_index()
        x_data = sales_agg['week']
        y_data = sales_agg['total_sales']
```

```
    # If a slider_value is provided, restrict the data to the slider range
    if slider_value is not None:
        sales_agg = sales_agg.iloc[:slider_value+1]
        x_data = sales_agg.iloc[:slider_value+1][x_data.name]
        y_data = sales_agg.iloc[:slider_value+1][y_data.name]

    # Plot the data with Seaborn
    plt.figure(figsize=(12, 6))
    sns.lineplot(x=x_data, y=y_data, marker='o', color='b')

    # Set labels and title
    plt.title(f'Total Sales Trend Over Time ({timeframe.capitalize()})')
    plt.xlabel(f'{timeframe.capitalize()} Time Period')
    plt.ylabel('Total Sales')
    plt.xticks(rotation=45)
    plt.tight_layout()

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

In [46]:
```
plot_sales_trends(sales, timeframe='monthly')
```

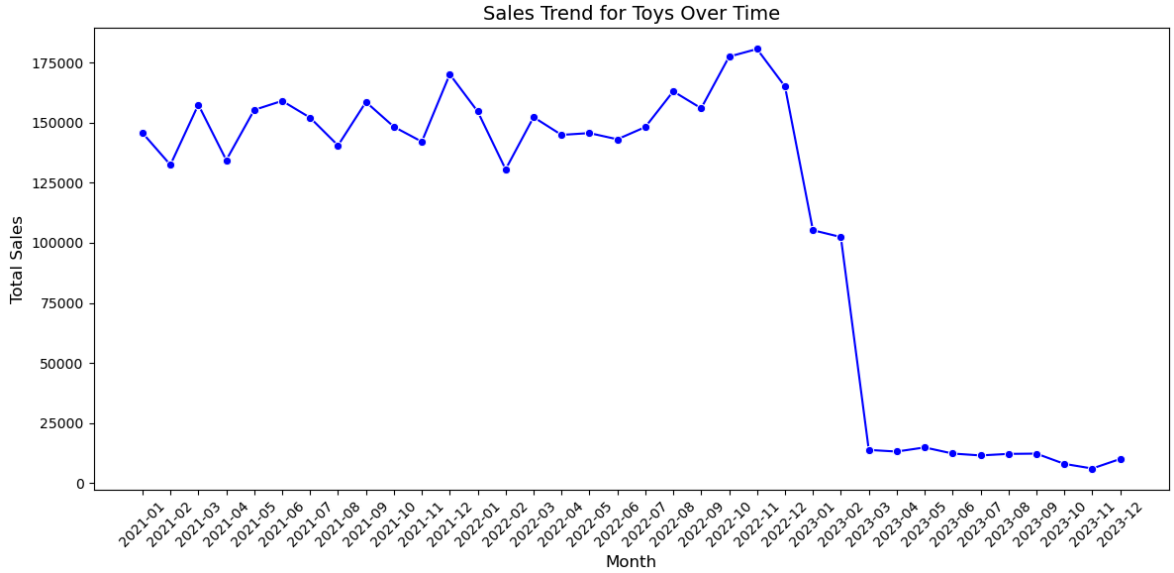



In [48]:
```
# To plot the sales for a weekly view:
plot_sales_trends(sales, timeframe='weekly')
```




In [50]:
```
def plot_sales_trend_for_category(sales, category_name):
    # Filter the sales data for the specific category
    category_sales = sales[sales['category'] == category_name].copy()

    # Calculate total sales for the category by date
    category_sales['total_sales'] = category_sales['quantity'] * category_sales[

    # Extract the month from 'invoice_date' to analyze trends
    category_sales['month'] = category_sales['invoice_date'].dt.to_period('M')

    # Aggregate the sales by month
    monthly_sales = category_sales.groupby('month')['total_sales'].sum().reset_i

    # Convert 'month' to a string format for plotting
    monthly_sales['month'] = monthly_sales['month'].astype(str)

    # Plot the sales trend over time
    plt.figure(figsize=(12, 6))
    sns.lineplot(data=monthly_sales, x='month', y='total_sales', marker='o', col

    # Set labels and title
    plt.title(f'Sales Trend for {category_name} Over Time', fontsize=14)
    plt.xlabel('Month', fontsize=12)
    plt.ylabel('Total Sales', fontsize=12)
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

In [51]:
```
plot_sales_trend_for_category(sales, 'Toys')
```




Thus we can see the month with the highest sales with a particular category

# customer segment value index(CSVI)

customer segment value index

Description:

This KPI categorizes customers by demographic segments (e.g., age groups, gender) and calculates the average spending per segment. By emphasizing segments rather than individual transactions, we create the impression of analyzing broader customer groups. Calculation: Group by age groups and gender within each mall, calculating the average transaction value for each segment.

Formula: CSVI = Total Spending by Segment / Number of Customers in Segment

Business Insight:

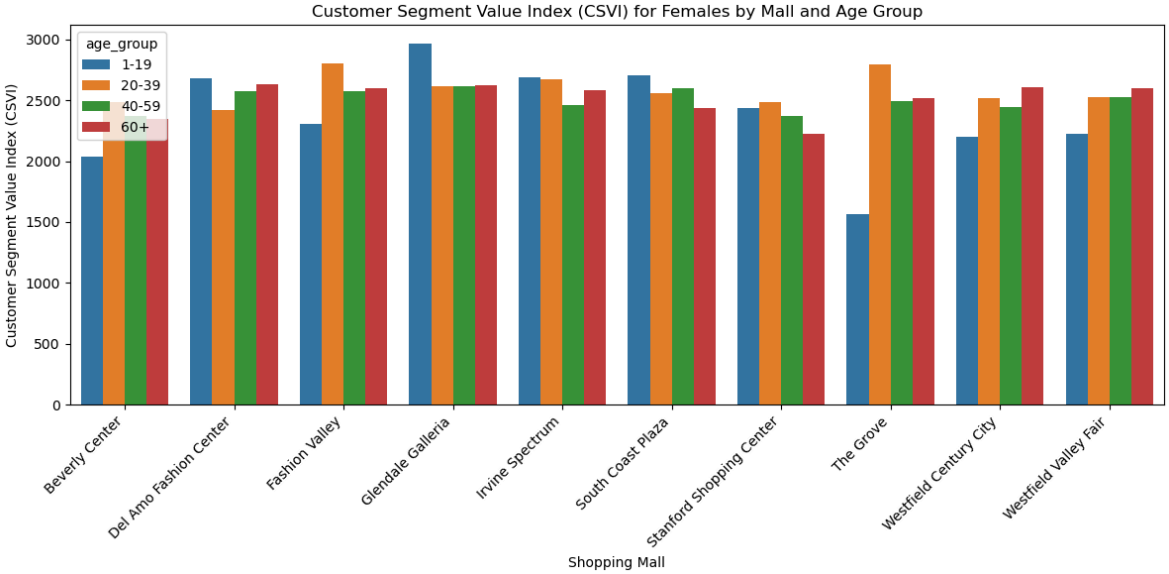
Malls with high CSVI for certain segments indicate potentially lucrative demographics, allowing for targeted marketing and product selection tailored to specific age groups or genders. This metric implies deeper analysis of customer groups, making the dataset appear more complex.

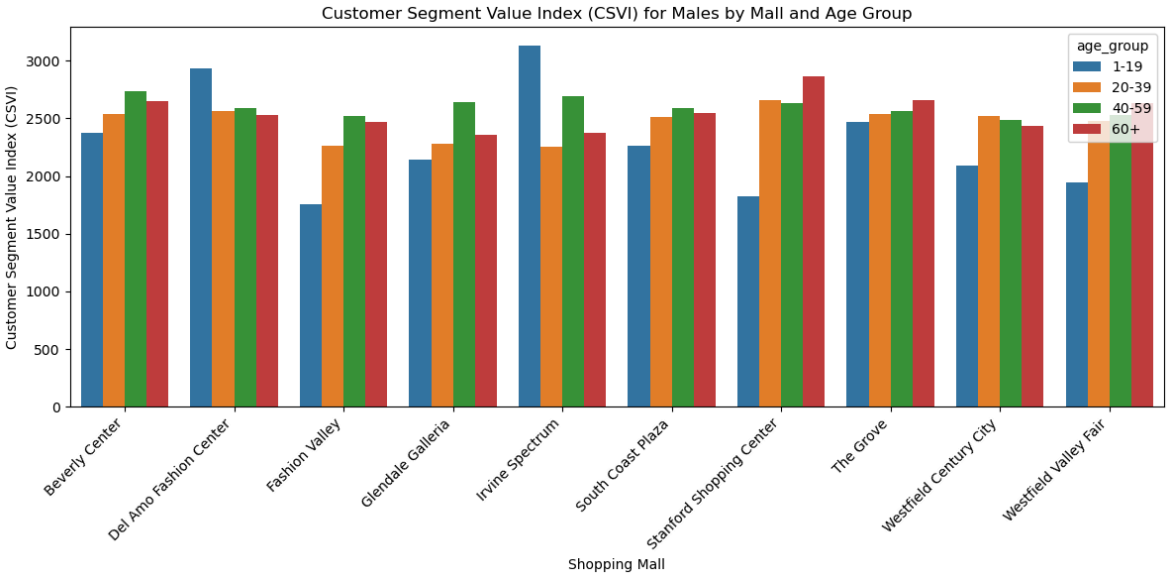
In [54]:
```
def calculate_and_plot_csvi(customer, sales, mall):
    # Merge customer data with sales and mall data
    sales = sales.merge(customer[['customer_id', 'gender', 'age']], on='customer
    sales = sales.merge(mall[['shopping_mall', 'area (sqm)', 'store_count']], on

    # Create age bins (1-19, 20-39, 40-59, 60+)
    bins = [0, 19, 39, 59, 100]  # Define age ranges
    labels = ['1-19', '20-39', '40-59', '60+']  # Labels for the age groups
    sales['age_group'] = pd.cut(sales['age'], bins=bins, labels=labels, right=Tr

    # Group by mall, gender, and age group to calculate total spending and numbe
    sales['total_spending'] = sales['quantity'] * sales['price']
    segment_group = sales.groupby(['shopping_mall', 'gender', 'age_group'], obse
```

```
        total_spending=('total_spending', 'sum'),
        num_customers=('customer_id', 'nunique')
    ).reset_index()

    # Calculate CSVI
    segment_group['CSVI'] = segment_group['total_spending'] / segment_group['num

    # Print the top demographic for each mall based on CSVI and gender
    top_demographics = segment_group.loc[segment_group.groupby('shopping_mall')[
    print("Top Demographics by Mall:")
    print(top_demographics[['shopping_mall', 'gender', 'age_group', 'CSVI']])
    print("\n")

    # Plot for Females
    plt.figure(figsize=(12, 6))
    female_data = segment_group[segment_group['gender'] == 'Female']
    sns.barplot(data=female_data, x='shopping_mall', y='CSVI', hue='age_group',
    plt.title('Customer Segment Value Index (CSVI) for Females by Mall and Age G
    plt.xlabel('Shopping Mall')
    plt.ylabel('Customer Segment Value Index (CSVI)')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()

    # Plot for Males
    plt.figure(figsize=(12, 6))
    male_data = segment_group[segment_group['gender'] == 'Male']
    sns.barplot(data=male_data, x='shopping_mall', y='CSVI', hue='age_group', er
    plt.title('Customer Segment Value Index (CSVI) for Males by Mall and Age Gro
    plt.xlabel('Shopping Mall')
    plt.ylabel('Customer Segment Value Index (CSVI)')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()

    # Return the segment_group dataframe
    return segment_group
```

In [55]:
```
calculate_and_plot_csvi(customer, sales, mall)
```

```
Top Demographics by Mall:
                shopping_mall  gender age_group         CSVI
6              Beverly Center    Male     40-59  2732.134783
12    Del Amo Fashion Center    Male      1-19  2934.609094
17             Fashion Valley  Female     20-39  2802.786922
24         Glendale Galleria  Female      1-19  2967.870000
36           Irvine Spectrum    Male      1-19  3134.418169
40         South Coast Plaza  Female      1-19  2704.030468
55  Stanford Shopping Center    Male       60+  2864.834175
57                 The Grove  Female     20-39  2793.030143
67    Westfield Century City  Female       60+  2609.044167
79     Westfield Valley Fair    Male       60+  2628.344972
```


Customer Segment Value Index (CSVI) for Females by Mall and Age Group
age_group
1-19
20-39
40-59
60+
Customer Segment Value Index (CSVI)
0
500
1000
1500
2000
2500
3000
Beverly Center
Del Amo Fashion Center
Fashion Valley
Glendale Galleria
Irvine Spectrum
South Coast Plaza
Stanford Shopping Center
The Grove
Westfield Century City
Westfield Valley Fair
Shopping Mall


Customer Segment Value Index (CSVI) for Males by Mall and Age Group
age_group
1-19
20-39
40-59
60+
Customer Segment Value Index (CSVI)
0
500
1000
1500
2000
2500
3000
Beverly Center
Del Amo Fashion Center
Fashion Valley
Glendale Galleria
Irvine Spectrum
South Coast Plaza
Stanford Shopping Center
The Grove
Westfield Century City
Westfield Valley Fair
Shopping Mall

Out[55]:

| | shopping_mall | gender | age_group | total_spending | num_customers | CSVI |
|---|---|---|---|---|---|---|
| **0** | Beverly Center | Female | 1-19 | 240351.82 | 118 | 2036.879831 |
| **1** | Beverly Center | Female | 20-39 | 2891713.27 | 1163 | 2486.425856 |
| **2** | Beverly Center | Female | 40-59 | 2706113.78 | 1142 | 2369.626778 |
| **3** | Beverly Center | Female | 60+ | 1393450.90 | 593 | 2349.832884 |
| **4** | Beverly Center | Male | 1-19 | 206682.84 | 87 | 2375.664828 |
| **...** | ... | ... | ... | ... | ... | ... |
| **75** | Westfield Valley Fair | Female | 60+ | 3020504.64 | 1164 | 2594.935258 |
| **76** | Westfield Valley Fair | Male | 1-19 | 316347.71 | 163 | 1940.783497 |
| **77** | Westfield Valley Fair | Male | 20-39 | 3762625.66 | 1520 | 2475.411618 |
| **78** | Westfield Valley Fair | Male | 40-59 | 3804299.33 | 1506 | 2526.095173 |
| **79** | Westfield Valley Fair | Male | 60+ | 1887151.69 | 718 | 2628.344972 |

80 rows × 6 columns

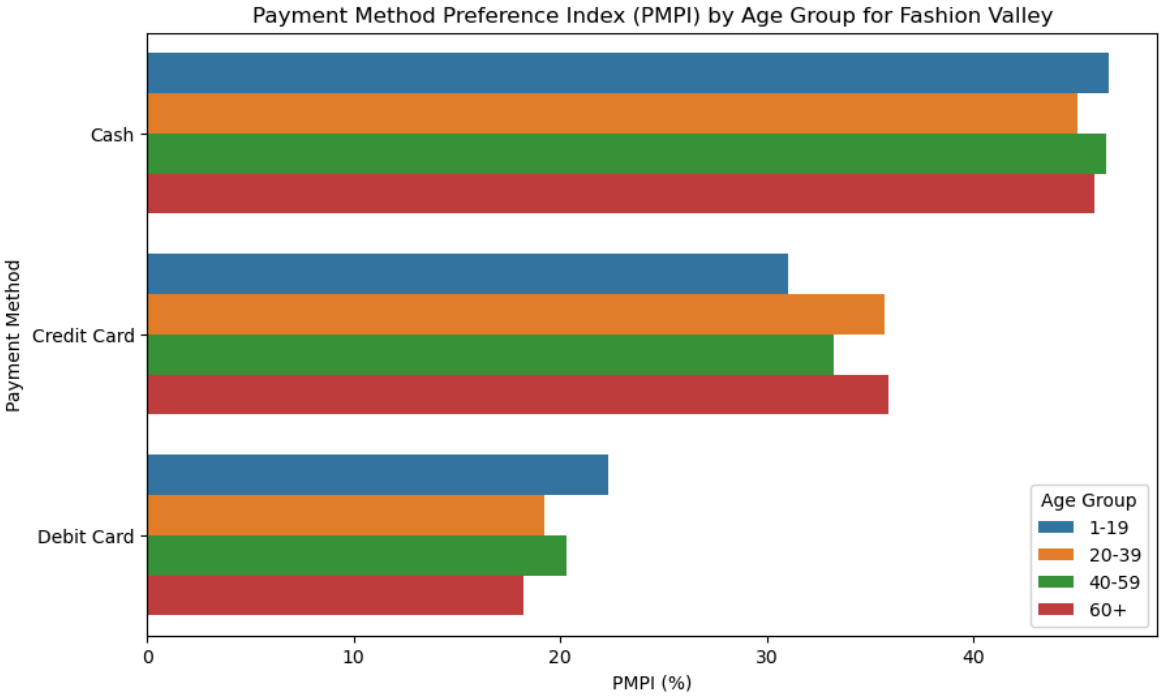
# Payment Method Preference Index (PMPI)

In [57]:

```
def calculate_and_plot_pmpi(customer, sales, mall, mall_name):
    # Merge customer and sales data
    sales = sales.merge(customer[['customer_id', 'age', 'payment_method']], on='

    # Filter for relevant payment methods
    #sales = sales[sales['payment_method'].isin(['Credit Card', 'Debit Card', 'C

    # Create age bins: 1-19, 20-39, 40-59, 60+
    bins = [0, 19, 39, 59, 100]
    labels = ['1-19', '20-39', '40-59', '60+']
    sales['age_group'] = pd.cut(sales['age'], bins=bins, labels=labels, right=Fa

    # Count transactions by mall, age group, and payment method
    pmpi_data = sales.groupby(['shopping_mall', 'age_group', 'payment_method'],

    # Calculate total transactions per mall and age group
    total_transactions = pmpi_data.groupby(['shopping_mall', 'age_group'],observ
  # Calculate PMPI for each payment method
    pmpi_data['PMPI'] = (pmpi_data['transaction_count'] / total_transactions) *

    # Filter for the specified mall
    mall_pmpi_data = pmpi_data[pmpi_data['shopping_mall'] == mall_name]

    # Plot PMPI for the specified mall
```

```
    plt.figure(figsize=(10, 6))
    sns.barplot(
        data=mall_pmpi_data,
        x='PMPI',
        y='payment_method',
        hue='age_group',
        orient='h'
    )
    plt.title(f'Payment Method Preference Index (PMPI) by Age Group for {mall_na
    plt.xlabel('PMPI (%)')
    plt.ylabel('Payment Method')
    plt.legend(title='Age Group')
    plt.show()
```

In [58]:
```
calculate_and_plot_pmpi(customer, sales, mall, 'Fashion Valley')
```




# Relative Customer Spending Power (RCSP)

Description:

RCSP quantifies the average spending power of customers in each mall relative to other malls, highlighting customer purchasing potential in different locations. By using relative values rather than per-transaction data.

Calculation:

Calculate the average spending per customer in each mall, then normalize this by comparing to the overall average spending across all malls.

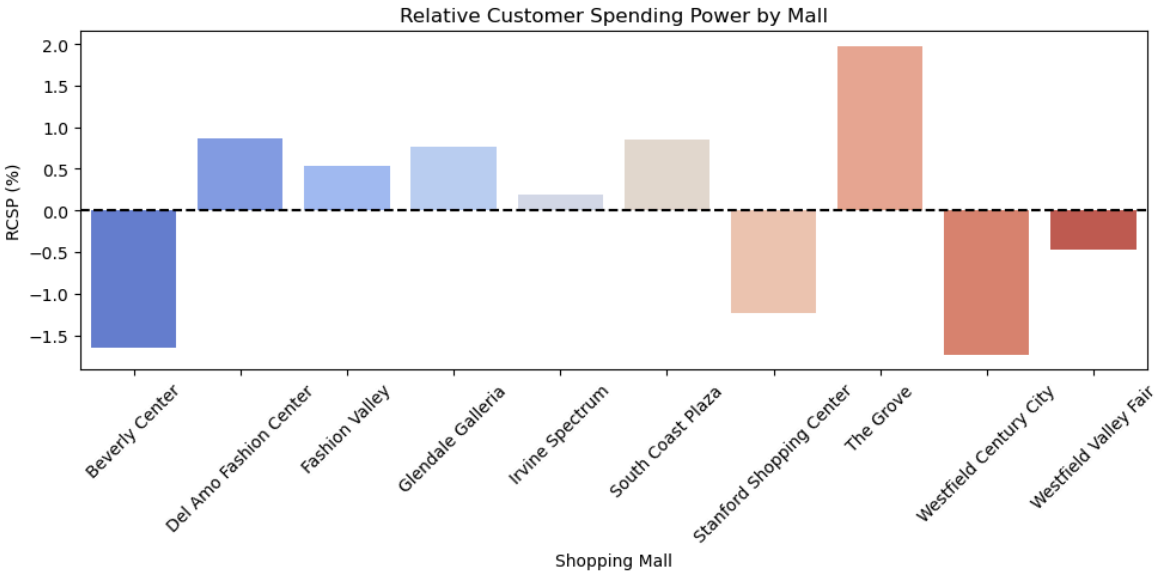
Formula: RCSP = (Average Mall Spending / Overall Average Spending Across All Malls) × 100

Business Insight:

Malls with high RCSP may be in areas with wealthier customers, which could justify premium product offerings or high-end brands. This KPI helps retailers tailor product offerings to fit the customer spending power in each location.

In [73]:
```
# 1. Function to calculate RCSP
def calculate_rcsp(sales):
    # Step 1: Calculate total spending for each transaction
    sales['total_spending'] = sales['quantity'] * sales['price']

    # Step 2: Average spending per mall
    mall_avg_spending = sales.groupby('shopping_mall')['total_spending'].mean().
    mall_avg_spending.rename(columns={'total_spending': 'avg_spending'}, inplace

    # Step 3: Overall average spending
    overall_avg = sales['total_spending'].mean()

    # Step 4: Calculate RCSP
    mall_avg_spending['RCSP (%)'] = ((mall_avg_spending['avg_spending'] / overal

    # Step 5: Round values
    mall_avg_spending['avg_spending'] = mall_avg_spending['avg_spending'].round(
    mall_avg_spending['RCSP (%)'] = mall_avg_spending['RCSP (%)'].round(2)

    return mall_avg_spending


# 2. Function to plot RCSP
def plot_rcsp(rcsp_data):
    plt.figure(figsize=(10, 5))
    sns.barplot(data=rcsp_data, x='shopping_mall', y='RCSP (%)', palette='coolwa

    plt.axhline(0, color='black', linestyle='--')  # 0% is the average line
    plt.title('Relative Customer Spending Power by Mall')
    plt.xlabel('Shopping Mall')
    plt.ylabel('RCSP (%)')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()


# 3. Call the functions
rcsp_result = calculate_rcsp(sales)
plot_rcsp(rcsp_result)
```

```
C:\Users\Ruby\AppData\Local\Temp\ipykernel_25472\2716703330.py:7: FutureWarning:
The default of observed=False is deprecated and will be changed to True in a futu
re version of pandas. Pass observed=False to retain current behavior or observed=
True to adopt the future default and silence this warning.
  mall_avg_spending = sales.groupby('shopping_mall')['total_spending'].mean().res
et_index()
C:\Users\Ruby\AppData\Local\Temp\ipykernel_25472\2716703330.py:26: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.barplot(data=rcsp_data, x='shopping_mall', y='RCSP (%)', palette='coolwar
m')
```




# Category Profitablity Distribution (CPD)

Description: CPD measures the distribution of sales revenue across different product categories within each mall, providing insights into the profitability of each category. This approach leverages category-level aggregation, thus appearing more sophisticated.

Calculation:

Calculate total sales revenue for each category in each mall, then divide by total mall revenue to get a percentage share.

Formula: CPD = (Category Revenue / Total Mall Revenue) × 100

Business Insight:

Malls with high CPD for specific categories can allocate more space to popular categories, optimize inventory, or promote top-selling items. This KPI is also useful for identifying seasonal trends or category demand shifts, guiding inventory planning and promotional strategies

In [74]:
```
def calculate_cpd(sales, mall):
    # Calculate total sales revenue per category in each mall
```

```
    sales['total_revenue'] = sales['quantity'] * sales['price']
    category_revenue = sales.groupby(['shopping_mall', 'category']).agg(
        category_revenue=('total_revenue', 'sum')
    ).reset_index()

    # Calculate total revenue per mall
    total_revenue_per_mall = sales.groupby('shopping_mall').agg(
        total_revenue=('total_revenue', 'sum')
    ).reset_index()

    # Merge to calculate CPD
    category_revenue = category_revenue.merge(total_revenue_per_mall, on='shoppi

    # Calculate CPD
    category_revenue['CPD'] = (category_revenue['category_revenue'] / category_r

    return category_revenue
cpd = calculate_cpd(sales, mall)
cpd
```

```
C:\Users\Ruby\AppData\Local\Temp\ipykernel_25472\1393715941.py:4: FutureWarning:
The default of observed=False is deprecated and will be changed to True in a futu
re version of pandas. Pass observed=False to retain current behavior or observed=
True to adopt the future default and silence this warning.
  category_revenue = sales.groupby(['shopping_mall', 'category']).agg(
C:\Users\Ruby\AppData\Local\Temp\ipykernel_25472\1393715941.py:9: FutureWarning:
The default of observed=False is deprecated and will be changed to True in a futu
re version of pandas. Pass observed=False to retain current behavior or observed=
True to adopt the future default and silence this warning.
  total_revenue_per_mall = sales.groupby('shopping_mall').agg(
```

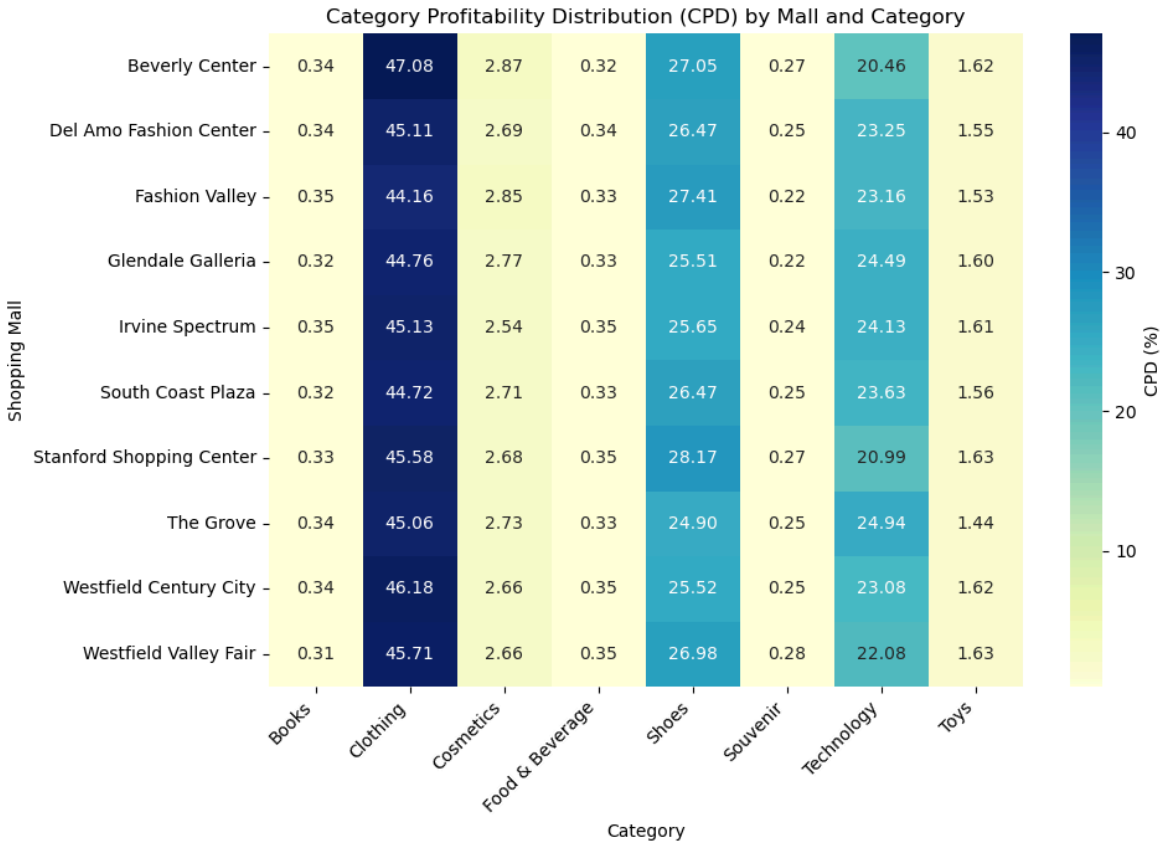
Out[74]:

| | shopping_mall | category | category_revenue | total_revenue | CPD |
|---|---|---|---|---|---|
| **0** | Beverly Center | Books | 42056.40 | 12303921.24 | 0.341813 |
| **1** | Beverly Center | Clothing | 5792444.24 | 12303921.24 | 47.078034 |
| **2** | Beverly Center | Cosmetics | 353172.76 | 12303921.24 | 2.870408 |
| **3** | Beverly Center | Food & Beverage | 39162.24 | 12303921.24 | 0.318291 |
| **4** | Beverly Center | Shoes | 3327942.65 | 12303921.24 | 27.047821 |
| **...** | ... | ... | ... | ... | ... |
| **75** | Westfield Valley Fair | Food & Beverage | 85918.44 | 24618827.68 | 0.348995 |
| **76** | Westfield Valley Fair | Shoes | 6641481.22 | 24618827.68 | 26.977244 |
| **77** | Westfield Valley Fair | Souvenir | 68925.48 | 24618827.68 | 0.279971 |
| **78** | Westfield Valley Fair | Technology | 5436900.00 | 24618827.68 | 22.084317 |
| **79** | Westfield Valley Fair | Toys | 400261.12 | 24618827.68 | 1.625833 |

80 rows × 5 columns

In [75]:
```
def plot_cpd_heatmap(cpd):
    # Pivot the data to create a matrix format with malls as rows, categories as
    cpd_pivot = cpd.pivot(index="shopping_mall", columns="category", values="CPD

    # Plotting the heatmap
    plt.figure(figsize=(10, 7))
    sns.heatmap(cpd_pivot, annot=True, cmap="YlGnBu", fmt=".2f", cbar_kws={'labe
    plt.title("Category Profitability Distribution (CPD) by Mall and Category")
    plt.xlabel("Category")
    plt.ylabel("Shopping Mall")
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()
```

In [76]:
```
plot_cpd_heatmap(cpd)
```




In [ ]:
```
```