```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from itertools import combinations
from collections import Counter
import warnings
warnings.filterwarnings('ignore')

# I hereby declare that the work presented in the submitted report and the accompanying code is entirely my own
# No portion of this submission has been copied, reproduced, or directly generated/refined using the responses
# or outputs of any AI tools (including, but not limited to, ChatGPT, Copilot, Gemini, DeepSeek, or other
# automated systems). Any external sources, datasets, or tools that have been used are properly cited and
# referenced. I understand that any breach of this declaration may result in submission cancellation or
# signif
```

```
# load the data
df = pd.read_csv('/content/groceries.csv', header=None)
print("Data loaded successfully")
print(f"Shape: {df.shape}")
df.head()
```

```
Data loaded successfully
Shape: (4472, 4)
```

	0	1	2	3
0	citrus fruit	semi-finished bread	margarine	ready soups
1	tropical fruit	yogurt	coffee	NaN
2	whole milk	NaN	NaN	NaN
3	pip fruit	yogurt	cream cheese	meat spreads
4	other vegetables	whole milk	condensed milk	long life bakery product

Next steps: Generate code with df | New interactive sheet

```
# schema: each row represents one transaction, items are comma-separated strings
# need to convert this to transaction_id and list of items
print("\nData Dictionary:")
print("- Original format: Single column with comma-separated product names")
print("- Each row = 1 transaction")
print("- Need to transform into: transaction_id, items (list), basket_size")
```

```
Data Dictionary:
- Original format: Single column with comma-separated product names
- Each row = 1 transaction
- Need to transform into: transaction_id, items (list), basket_size
```

```
# basic stats before cleaning
total_transactions = len(df)
print(f"\nTotal transactions: {total_transactions}")

# parse items from each row
all_items = []
for idx, row in df.iterrows():
    items = [item.strip() for item in str(row[0]).split(',')]
    all_items.extend(items)

unique_products = len(set(all_items))
print(f"Unique products: {unique_products}")
```

```
Total transactions: 4472
Unique products: 142
```

```
# basic stats before cleaning
total_transactions = len(df)
print(f"\nTotal transactions: {total_transactions}")
```

```
# parse items from each row
all_items = []
for idx, row in df.iterrows():
    items = [item.strip() for item in str(row[0]).split(',')]
    all_items.extend(items)

unique_products = len(set(all_items))
print(f"Unique products: {unique_products}")
```

```
Total transactions: 4472
Unique products: 142
```

```
# get basket sizes
basket_sizes = []
for idx, row in df.iterrows():
    items = [item.strip() for item in str(row[0]).split(',')]
    basket_sizes.append(len(items))

print(f"\nBasket size stats:")
print(f"Min: {min(basket_sizes)}")
print(f"Median: {np.median(basket_sizes)}")
print(f"95th percentile: {np.percentile(basket_sizes, 95)}")
```

```
Basket size stats:
Min: 1
Median: 1.0
95th percentile: 1.0
```

```
# count item frequencies
item_counts = Counter(all_items)
top20 = item_counts.most_common(20)

print("\nTop 20 products by frequency:")
for item, count in top20:
    print(f"{item}: {count}")
```

```
Top 20 products by frequency:
whole milk: 481
rolls/buns: 258
other vegetables: 253
soda: 241
tropical fruit: 174
citrus fruit: 165
yogurt: 160
frankfurter: 144
root vegetables: 129
bottled water: 121
pip fruit: 119
pastry: 108
beef: 107
chicken: 86
coffee: 71
beverages: 70
ice cream: 68
whipped/sour cream: 64
berries: 56
newspapers: 56
```

```
transactions = []

for idx, row in df.iterrows():
    items_clean = []
    # Iterate through all columns in the row to get items
    for col in row.index:
        item = str(row[col])
        cleaned = item.strip().lower()
        if cleaned and cleaned != 'nan':  # remove empty items and 'nan' strings
            items_clean.append(cleaned)

    # only keep baskets with at least 2 items
    if len(items_clean) >= 2:
        transactions.append({
            'transaction_id': idx,
            'items': items_clean,
```

```
            'basket_size': len(items_clean)
        })

print(f"Transactions after cleaning: {len(transactions)}")
print(f"Removed {total_transactions - len(transactions)} transactions with < 2 items")
```

```
Transactions after cleaning: 2820
Removed 1652 transactions with < 2 items
```

```
# create dataframe - keep items as they are for now
trans_df = pd.DataFrame(transactions)
print(f"Created dataframe with {len(trans_df)} transactions")
trans_df.head()
```

```
Created dataframe with 2820 transactions
```

	transaction_id	items	basket_size
0	0	[citrus fruit, semi-finished bread, margarine, ready soups]	4
1	1	[tropical fruit, yogurt, coffee]	3
2	3	[pip fruit, yogurt, cream cheese, meat spreads]	4
3	4	[other vegetables, whole milk, condensed milk, long life bakery product]	4
4	7	[whole milk, cereals]	2

Next steps: Generate code with `trans_df` New interactive sheet

```
# save clean transactions - convert list to string for storage
trans_save = trans_df.copy()
trans_save['items'] = trans_save['items'].apply(lambda x: ','.join(x))
trans_save.to_parquet('transactions_clean.parquet', index=False)
print("Saved transactions_clean.parquet")
```

```
Saved transactions_clean.parquet
```

```
# get all unique products from cleaned data - using the in-memory df
all_products = set()
for items_list in trans_df['items']:
    all_products.update(items_list)

all_products = sorted(list(all_products))
print(f"Total unique products after cleaning: {len(all_products)}")
```

```
Total unique products after cleaning: 146
```

```
# assign random prices to products
# using fixed seed for reproducibility
rng = np.random.default_rng(42)

product_prices = {}
for product in all_products:
    price = rng.uniform(0.50, 15.00)
    product_prices[product] = round(price, 2)

# save to csv
price_df = pd.DataFrame(list(product_prices.items()), columns=['product', 'price'])
price_df.to_csv('product_prices.csv', index=False)
print("Saved product_prices.csv")
print(f"\nSample prices:")
print(price_df.head(10))
```

```
Saved product_prices.csv

Sample prices:
            product  price
0  abrasive cleaner  11.72
1  artif. sweetener   6.86
2    baby cosmetics  12.95
3     baking powder  10.61
4  bathroom cleaner   1.87
5              beef  14.65
6           berries  11.54
7         beverages  11.90
8     bottled water   2.36
```

```
9       brown bread   7.03
```

```
# calculate basket totals
basket_totals = []

for items in trans_df['items']:
    total = sum(product_prices[item] for item in items)
    basket_totals.append(round(total, 2))

trans_df['basket_total'] = basket_totals
print("Added basket totals")
print(f"\nBasket total stats:")
print(f"Min: ${min(basket_totals):.2f}")
print(f"Mean: ${np.mean(basket_totals):.2f}")
print(f"Max: ${max(basket_totals):.2f}")
```

```
Added basket totals

Basket total stats:
Min: $2.18
Mean: $19.73
Max: $51.49
```

```
# save transactions with prices
trans_df.to_csv('transactions_priced.csv', index=False)
print("Saved transactions_priced.csv")
trans_df.head()
```

```
Saved transactions_priced.csv
```

	transaction_id	items	basket_size	basket_total
0	0	[citrus fruit, semi-finished bread, margarine, ready soups]	4	36.74
1	1	[tropical fruit, yogurt, coffee]	3	24.51
2	3	[pip fruit, yogurt, cream cheese, meat spreads]	4	25.14
3	4	[other vegetables, whole milk, condensed milk, long life bakery product]	4	23.25
4	7	[whole milk, cereals]	2	18.45

Next steps: Generate code with `trans_df` New interactive sheet

```
# count pairs - items that appear together
min_count = 20  # configurable threshold
k = 10  # top k results

pair_counter = Counter()

for items in trans_df['items']:
    if len(items) >= 2:
        pairs = combinations(sorted(items), 2)
        pair_counter.update(pairs)

print(f"Total unique pairs: {len(pair_counter)}")
```

```
Total unique pairs: 2562
```

```
# filter pairs by min support
pairs_filtered = {pair: count for pair, count in pair_counter.items() if count >= min_count}
print(f"Pairs with support >= {min_count}: {len(pairs_filtered)}")
```

```
Pairs with support >= 20: 51
```

```
# get top k pairs
topk_pairs = sorted(pairs_filtered.items(), key=lambda x: (-x[1], x[0]))[:k]

print(f"\nTop {k} pairs by frequency:")
total_baskets = len(trans_df)
for pair, count in topk_pairs:
    support_frac = count / total_baskets
    print(f"{pair[0]} + {pair[1]}: count={count}, support={support_frac:.4f}")
```

```
Top 10 pairs by frequency:
rolls/buns + soda: count=84, support=0.0298
rolls/buns + whole milk: count=80, support=0.0284
other vegetables + whole milk: count=75, support=0.0266
bottled water + soda: count=49, support=0.0174
frankfurter + rolls/buns: count=48, support=0.0170
pastry + whole milk: count=48, support=0.0170
pastry + soda: count=47, support=0.0167
soda + whole milk: count=47, support=0.0167
whole milk + yogurt: count=46, support=0.0163
root vegetables + whole milk: count=45, support=0.0160
```

```
# count triples
triple_counter = Counter()

for items in trans_df['items']:
    if len(items) >= 3:
        triples = combinations(sorted(items), 3)
        triple_counter.update(triples)

print(f"Total unique triples: {len(triple_counter)}")
```

```
Total unique triples: 3037
```

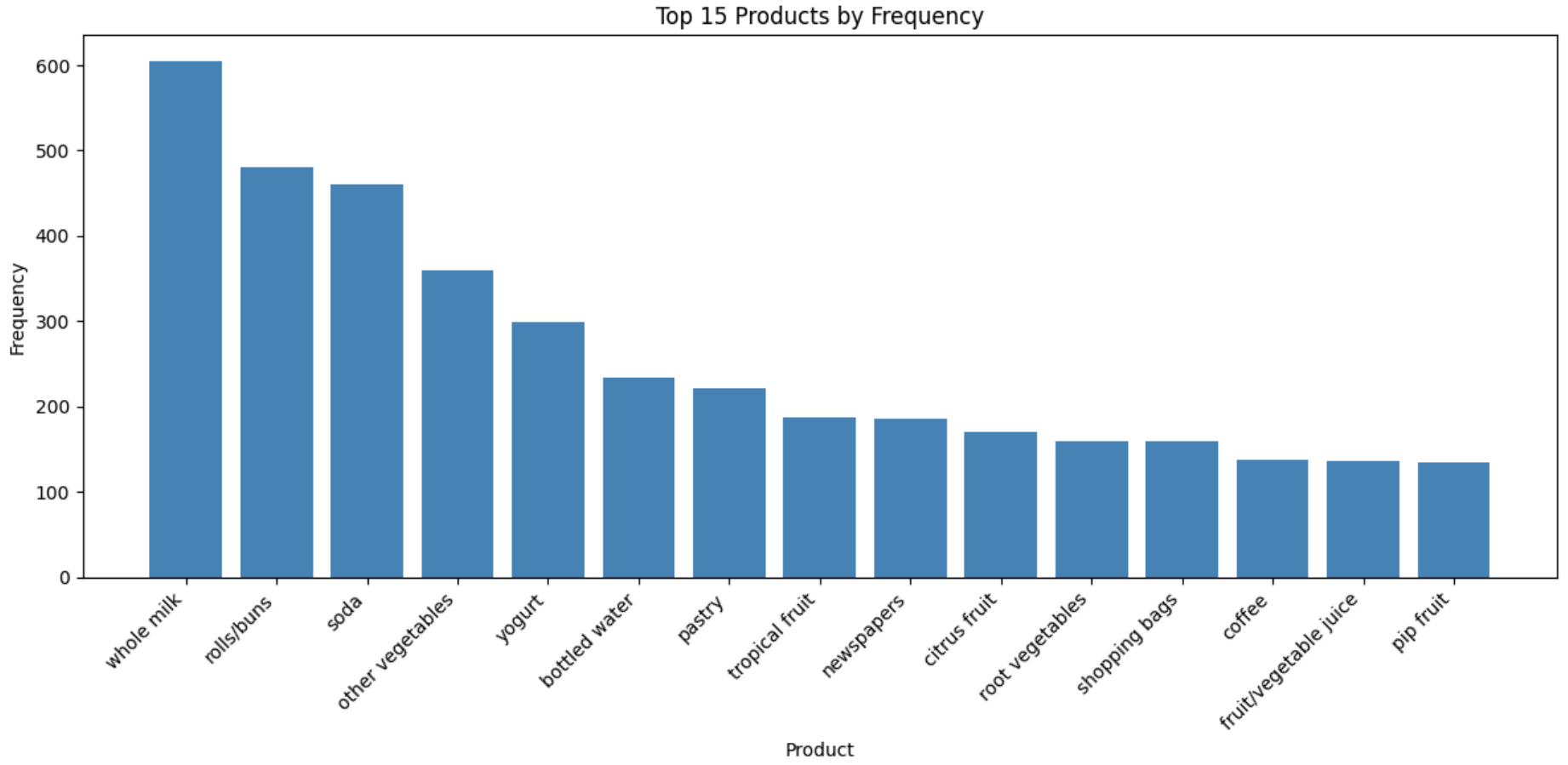
```
# filter triples by min support
triples_filtered = {triple: count for triple, count in triple_counter.items() if count >= min_count}
print(f"Triples with support >= {min_count}: {len(triples_filtered)}")
```

```
Triples with support >= 20: 0
```

```
# get top k triples
topk_triples = sorted(triples_filtered.items(), key=lambda x: (-x[1], x[0]))[:k]

print(f"\nTop {k} triples by frequency:")
for triple, count in topk_triples:
    support_frac = count / total_baskets
    print(f"{triple[0]} + {triple[1]} + {triple[2]}: count={count}, support={support_frac:.4f}")
```

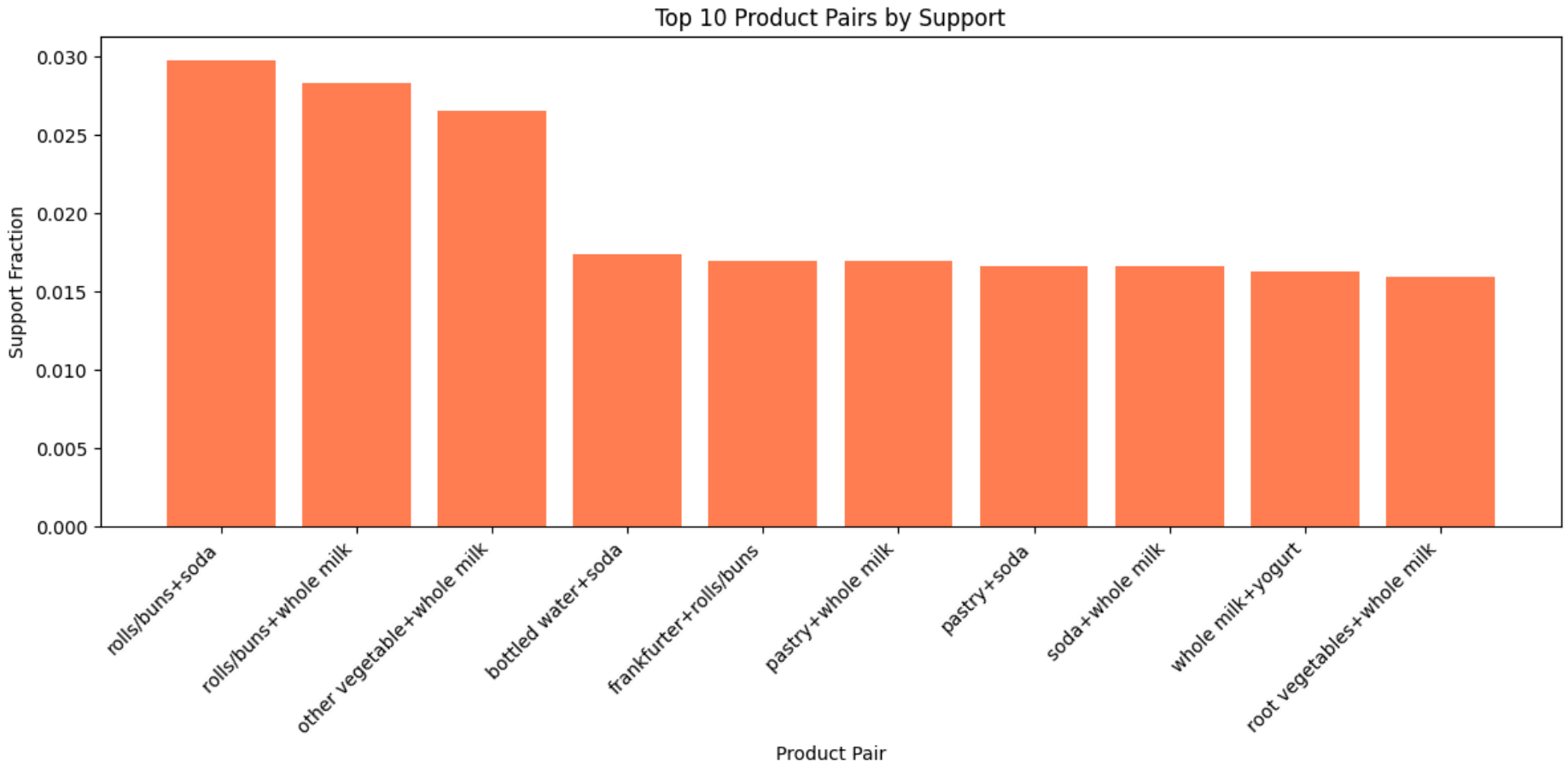
```
Top 10 triples by frequency:
```

```
# bar chart - top 15 individual items
item_counter = Counter()
for items in trans_df['items']:
    item_counter.update(items)

top15_items = item_counter.most_common(15)
items_names = [item[0] for item in top15_items]
items_counts = [item[1] for item in top15_items]

plt.figure(figsize=(12, 6))
plt.bar(range(len(items_names)), items_counts, color='steelblue')
plt.xticks(range(len(items_names)), items_names, rotation=45, ha='right')
plt.xlabel('Product')
plt.ylabel('Frequency')
plt.title('Top 15 Products by Frequency')
plt.tight_layout()
plt.show()
```



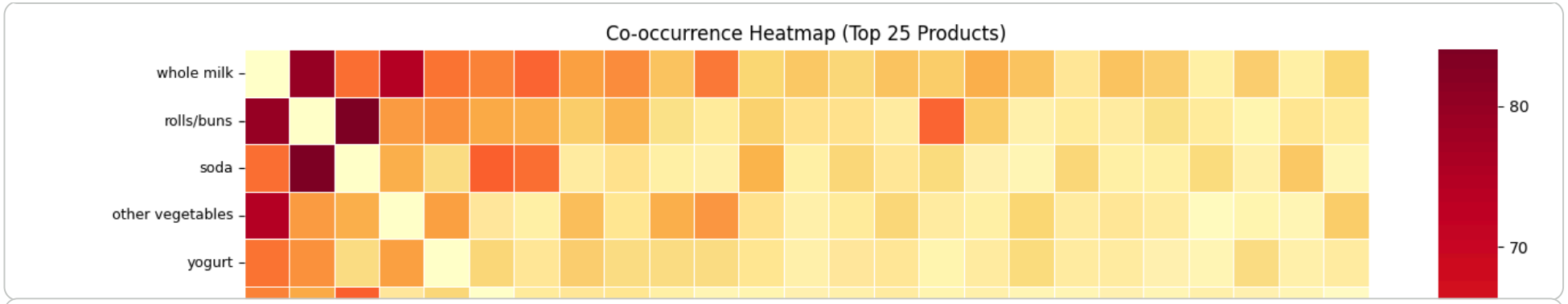

```
# bar chart - top k pairs by support fraction
pair_names = [f"{p[0][0][:15]}+{p[0][1][:15]}" for p in topk_pairs]
pair_supports = [p[1]/total_baskets for p in topk_pairs]

plt.figure(figsize=(12, 6))
plt.bar(range(len(pair_names)), pair_supports, color='coral')
plt.xticks(range(len(pair_names)), pair_names, rotation=45, ha='right')
plt.xlabel('Product Pair')
plt.ylabel('Support Fraction')
plt.title(f'Top {k} Product Pairs by Support')
plt.tight_layout()
```

```
plt.show()
```

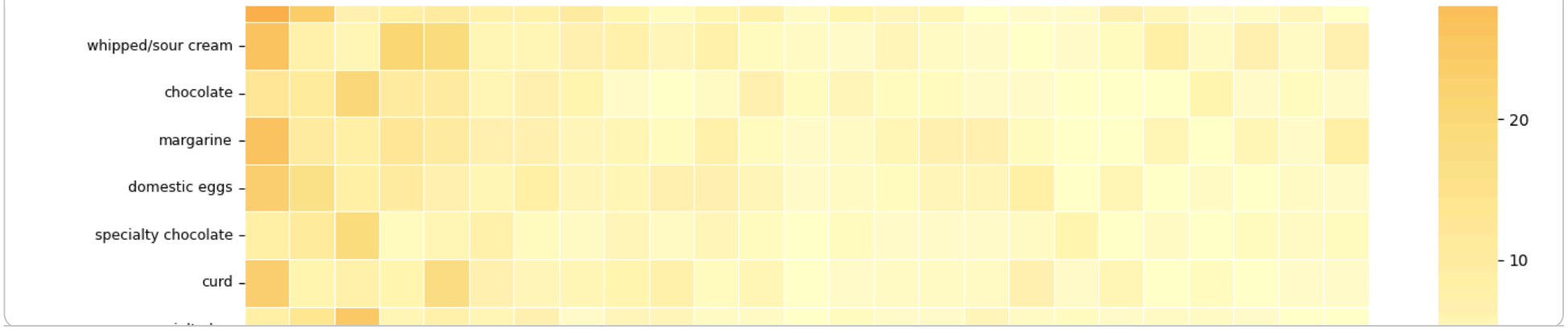


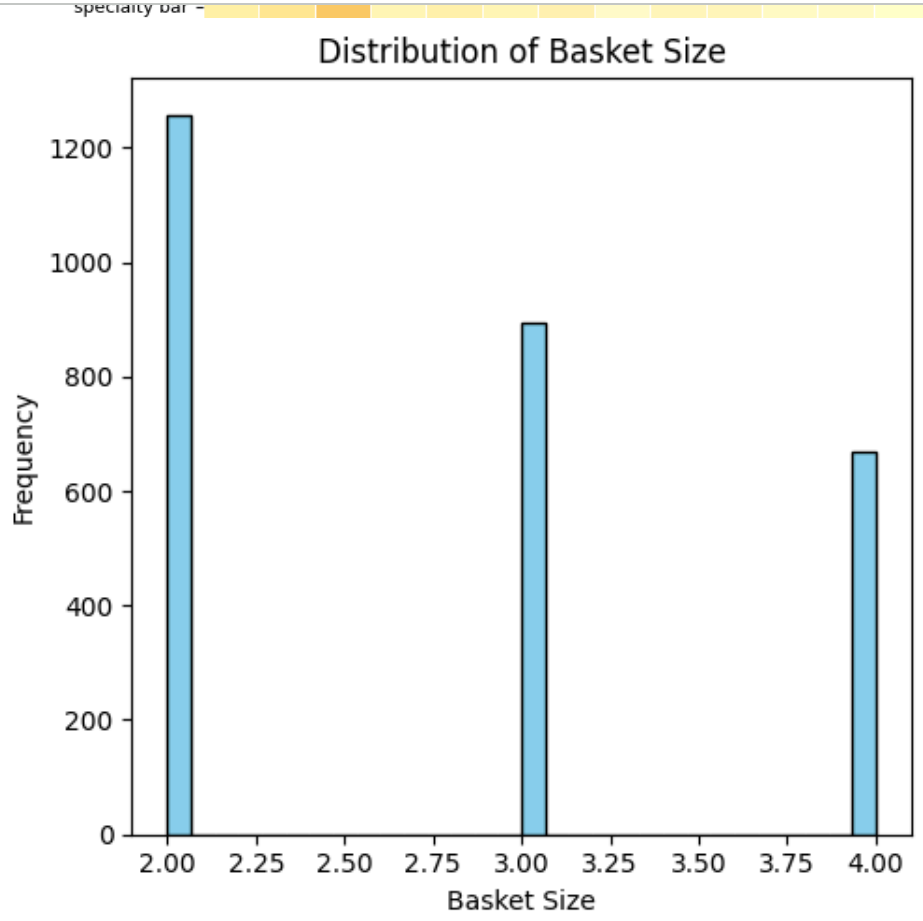

```
# heatmap - co-occurrence matrix for top 25 items
top25_items = [item[0] for item in item_counter.most_common(25)]

# build co-occurrence matrix properly
cooc_matrix = np.zeros((25, 25))

for items in trans_df['items']:
    # check which top25 items are in this basket
```

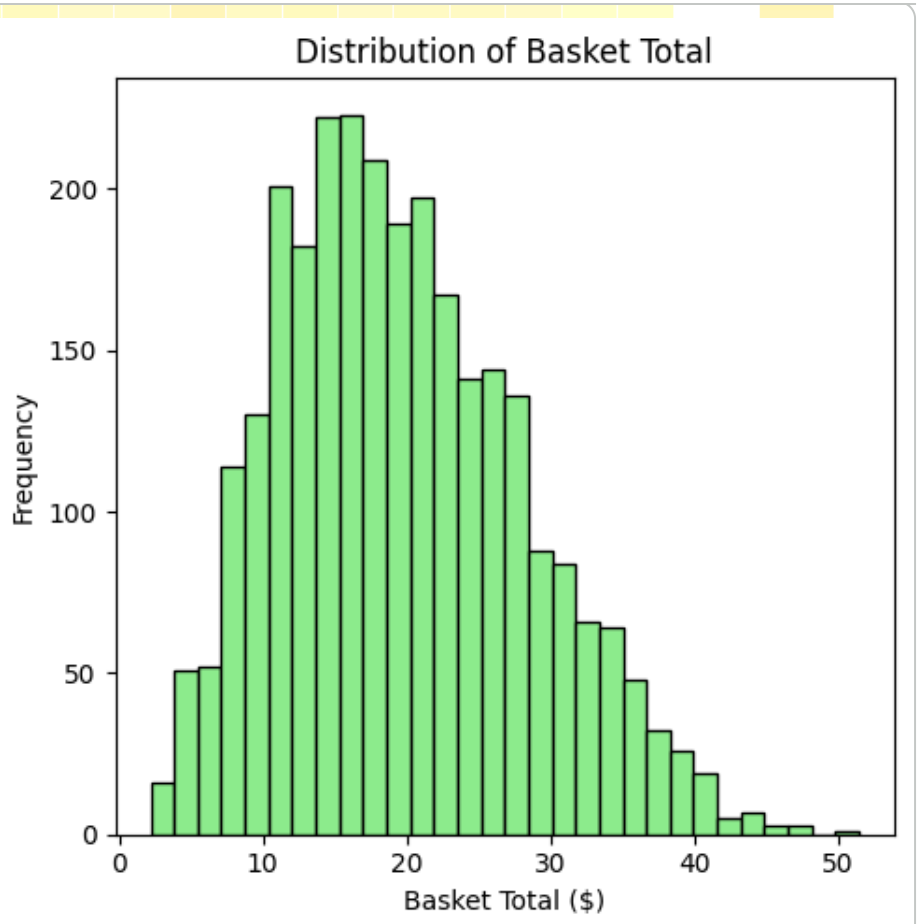
```
        for i in range(25):
            for j in range(25):
                if i != j:  # dont count item with itself
                    if top25_items[i] in items and top25_items[j] in items:
                        cooc_matrix[i][j] += 1

    # The matrix is already symmetric and contains correct counts from the nested loops above,
    # so no further division is needed.

    plt.figure(figsize=(14, 12))
    sns.heatmap(cooc_matrix, xticklabels=top25_items, yticklabels=top25_items,
                cmap='YlOrRd', linewidths=0.5, cbar_kws={'label': 'Co-occurrence Count'},
                fmt='.0f', annot=False)
    plt.xticks(rotation=45, ha='right', fontsize=9)
    plt.yticks(rotation=0, fontsize=9)
    plt.title('Co-occurrence Heatmap (Top 25 Products)')
    plt.tight_layout()
    plt.show()
```




```
# distribution plot - basket size
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.hist(trans_df['basket_size'], bins=30, color='skyblue', edgecolor='black')
plt.xlabel('Basket Size')
plt.ylabel('Frequency')
plt.title('Distribution of Basket Size')

# distribution plot - basket total
plt.subplot(1, 2, 2)
plt.hist(trans_df['basket_total'], bins=30, color='lightgreen', edgecolor='black')
plt.xlabel('Basket Total ($)')
plt.ylabel('Frequency')
plt.title('Distribution of Basket Total')

plt.tight_layout()
plt.show()
```








```
# summary statistics
print("="*60)
print("MARKET BASKET ANALYSIS - SUMMARY REPORT")
print("="*60)
print(f"\nDataset Overview:")
print(f"  Total transactions: {len(trans_df)}")
print(f"  Unique products: {len(all_products)}")
print(f"  Average basket size: {trans_df['basket_size'].mean():.2f}")
print(f"  Average basket total: ${trans_df['basket_total'].mean():.2f}")
```

```
print(f"\nCo-occurrence Analysis:")
print(f"  Minimum support threshold: {min_count}")
print(f"  Pairs meeting threshold: {len(pairs_filtered)}")
print(f"  Triples meeting threshold: {len(triples_filtered)}")

print(f"\nTop 3 Most Frequent Items:")
for i, (item, count) in enumerate(top15_items[:3], 1):
    print(f"  {i}. {item}: {count} occurrences")

print(f"\nTop 3 Product Pairs:")
for i, (pair, count) in enumerate(topk_pairs[:3], 1):
    print(f"  {i}. {pair[0]} + {pair[1]}: {count} co-occurrences")
```

```
============================================================
MARKET BASKET ANALYSIS - SUMMARY REPORT
```