Loading the datasets

In [ ]:
```
import pandas as pd
order_products_df = pd.read_csv("C:\\Users\\richa\\COMP 541 Data Mining\\Proje

orders_df = pd.read_csv("C:\\Users\\richa\\COMP 541 Data Mining\\Project\\orde

products_df = pd.read_csv("C:\\Users\\richa\\COMP 541 Data Mining\\Project\\pr

departments_df = pd.read_csv("C:\\Users\\richa\\COMP 541 Data Mining\\Project\

aisles_df = pd.read_csv("C:\\Users\\richa\\COMP 541 Data Mining\\Project\\aisl

'''
order_products_df = pd.read_csv("C:\\Users\\Nexxa\\Desktop\\Fall 24\\Comp 541\

orders_df = pd.read_csv("C:\\Users\\Nexxa\\Desktop\\Fall 24\\Comp 541\\instaca

products_df = pd.read_csv("C:\\Users\\Nexxa\\Desktop\\Fall 24\\Comp 541\\insta

departments_df = pd.read_csv("C:\\Users\\Nexxa\\Desktop\\Fall 24\\Comp 541\\in

aisles_df = pd.read_csv("C:\\Users\\Nexxa\\Desktop\\Fall 24\\Comp 541\\instaca
'''
```

In [2]:
```
order_products_df.head()
```

Out[2]:

	order_id	product_id	add_to_cart_order	reordered
0	1	49302	1	1
1	1	11109	2	1
2	1	10246	3	0
3	1	49683	4	0
4	1	43633	5	1

In [3]:
```
order_products_df.isna().sum()
```

Out[3]:
```
order_id             0
product_id           0
add_to_cart_order    0
reordered            0
dtype: int64
```

In [4]:
```
products_df.head()
```

Out[4]:

	product_id	product_name	aisle_id	department_id
0	1	Chocolate Sandwich Cookies	61	19
1	2	All-Seasons Salt	104	13
2	3	Robust Golden Unsweetened Oolong Tea	94	7
3	4	Smart Ones Classic Favorites Mini Rigatoni Wit...	38	1
4	5	Green Chile Anytime Sauce	5	13

In [5]:
```
products_df.isna().sum()
```

Out[5]:
```
product_id       0
product_name     0
aisle_id         0
department_id    0
dtype: int64
```

In [96]:
```
departments_df.head()
```

Out[96]:

	department_id	department
0	1	frozen
1	2	other
2	3	bakery
3	4	produce
4	5	alcohol

In [7]:
```
departments_df.isna().sum()
```

Out[7]:
```
department_id    0
department       0
dtype: int64
```

In [8]:
```
aisles_df.head()
```

Out[8]:

	aisle_id	aisle
0	1	prepared soups salads
1	2	specialty cheeses
2	3	energy granola bars
3	4	instant foods
4	5	marinades meat preparation

In [9]:
```
aisles_df.isna().sum()
```

Out[9]:
```
aisle_id    0
aisle       0
dtype: int64
```

In [10]:
```
orders_df.head()
```

Out[10]:

	order_id	user_id	eval_set	order_number	order_dow	order_hour_of_day	days_since_prior_o
0	2539329	1	prior	1	2	8	
1	2398795	1	prior	2	3	7	
2	473747	1	prior	3	3	12	
3	2254736	1	prior	4	4	7	
4	431534	1	prior	5	4	15	

In [11]:
```
orders_df.isna().sum()
```

Out[11]:
```
order_id                       0
user_id                        0
eval_set                       0
order_number                   0
order_dow                      0
order_hour_of_day              0
days_since_prior_order    206209
dtype: int64
```

Calculate Reorder Rate By Each Department

In [12]:
```
merged_df = pd.merge(order_products_df, products_df, on='product_id')
merged_df = pd.merge(merged_df, departments_df, on='department_id')
```

In [13]:
```
merged_df.head()
```

Out[13]:

	order_id	product_id	add_to_cart_order	reordered	product_name	aisle_id	department_id	de
0	1	49302	1	1	Bulgarian Yogurt	120	16	(
1	1	11109	2	1	Organic 4% Milk Fat Whole Milk Cottage Cheese	108	16	(
2	1	10246	3	0	Organic Celery Hearts	83	4	
3	1	49683	4	0	Cucumber Kirby	83	4	
4	1	43633	5	1	Lightly Smoked Sardines in Olive Oil	95	15	

In [16]:
```
merged_df.shape
```

Out[16]: (1384617, 8)

In [17]:
```
merged_df.head()
```

Out[17]:

	order_id	product_id	add_to_cart_order	reordered	product_name	aisle_id	department_id	de
0	1	49302	1	1	Bulgarian Yogurt	120	16	(
1	1	11109	2	1	Organic 4% Milk Fat Whole Milk Cottage Cheese	108	16	(
2	1	10246	3	0	Organic Celery Hearts	83	4	
3	1	49683	4	0	Cucumber Kirby	83	4	
4	1	43633	5	1	Lightly Smoked Sardines in Olive Oil	95	15	

In [18]:
```python
merged_df["department"].value_counts()
```

Out[18]:
```
department
produce            409087
dairy eggs         217051
snacks             118862
beverages          114046
frozen             100426
pantry              81242
bakery              48394
canned goods        46799
deli                44291
dry goods pasta     38713
household           35986
meat seafood        30307
breakfast           29500
personal care       21570
babies              14941
international       11902
missing              8251
alcohol              5598
pets                 4497
other                1795
bulk                 1359
Name: count, dtype: int64
```

In [19]:
```python
department_reorder_rate = (
    merged_df.groupby('department')['reordered']
    .mean()
    .sort_values(ascending=False)
    .reset_index()
)
```

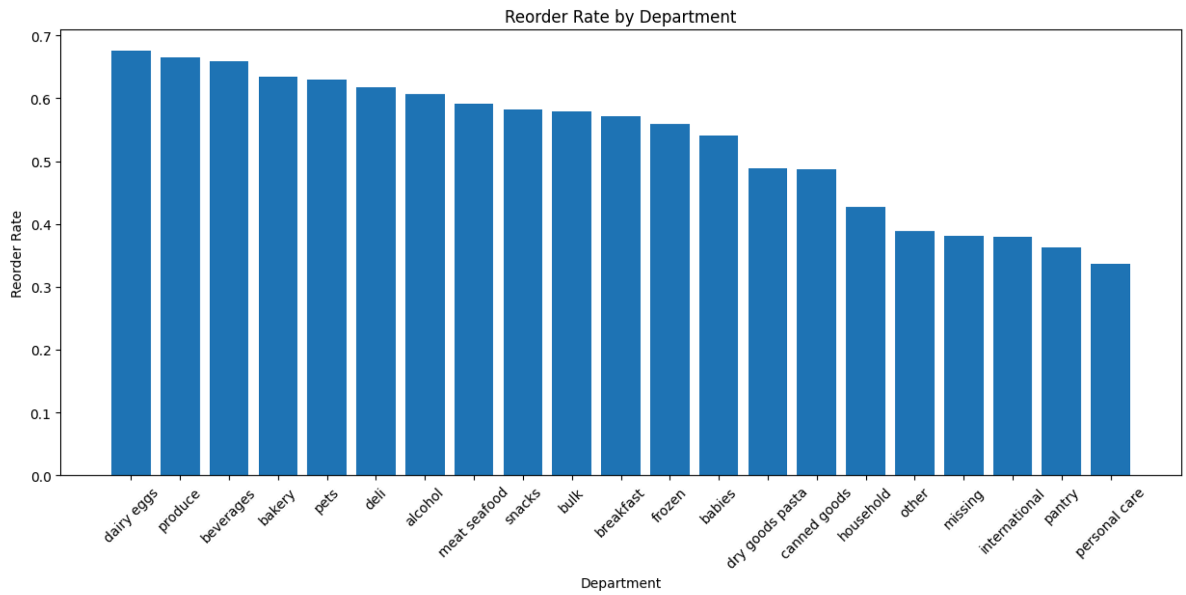
In [95]:
```
department_reorder_rate
```

Out[95]:

	department	reordered
0	dairy eggs	0.674966
1	produce	0.664617
2	beverages	0.658155
3	bakery	0.634211
4	pets	0.630198
5	deli	0.617891
6	alcohol	0.606824
7	meat seafood	0.590854
8	snacks	0.581363
9	bulk	0.578366
10	breakfast	0.571661
11	frozen	0.559297
12	babies	0.541062
13	dry goods pasta	0.487821
14	canned goods	0.486805
15	household	0.427166
16	other	0.388301
17	missing	0.381530
18	international	0.379936
19	pantry	0.363088
20	personal care	0.337089

In [21]:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(15, 6))
plt.bar(department_reorder_rate['department'], department_reorder_rate['reorde
plt.xlabel('Department')
plt.ylabel('Reorder Rate')
plt.title('Reorder Rate by Department')
plt.xticks(rotation=45)
#plt.savefig(r"C:\Users\richa\COMP 541 Data Mining\Reorder Rate by Department.
plt.show()
```



Calculate Item Sold By Each Department

In [22]:

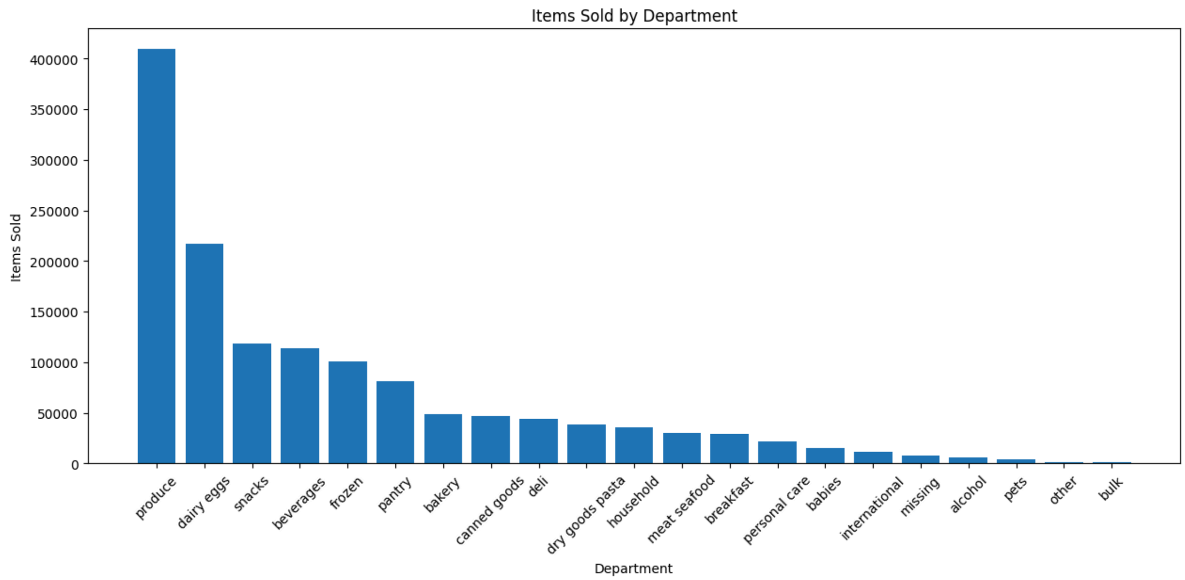
```
items_sold_by_department = (
    merged_df.groupby('department')['product_id']
    .count()
    .reset_index()
    .rename(columns={'product_id': 'items_sold'})
    .sort_values(by='items_sold', ascending=False)
)
```

In [99]:
```
items_sold_by_department
```

Out[99]:

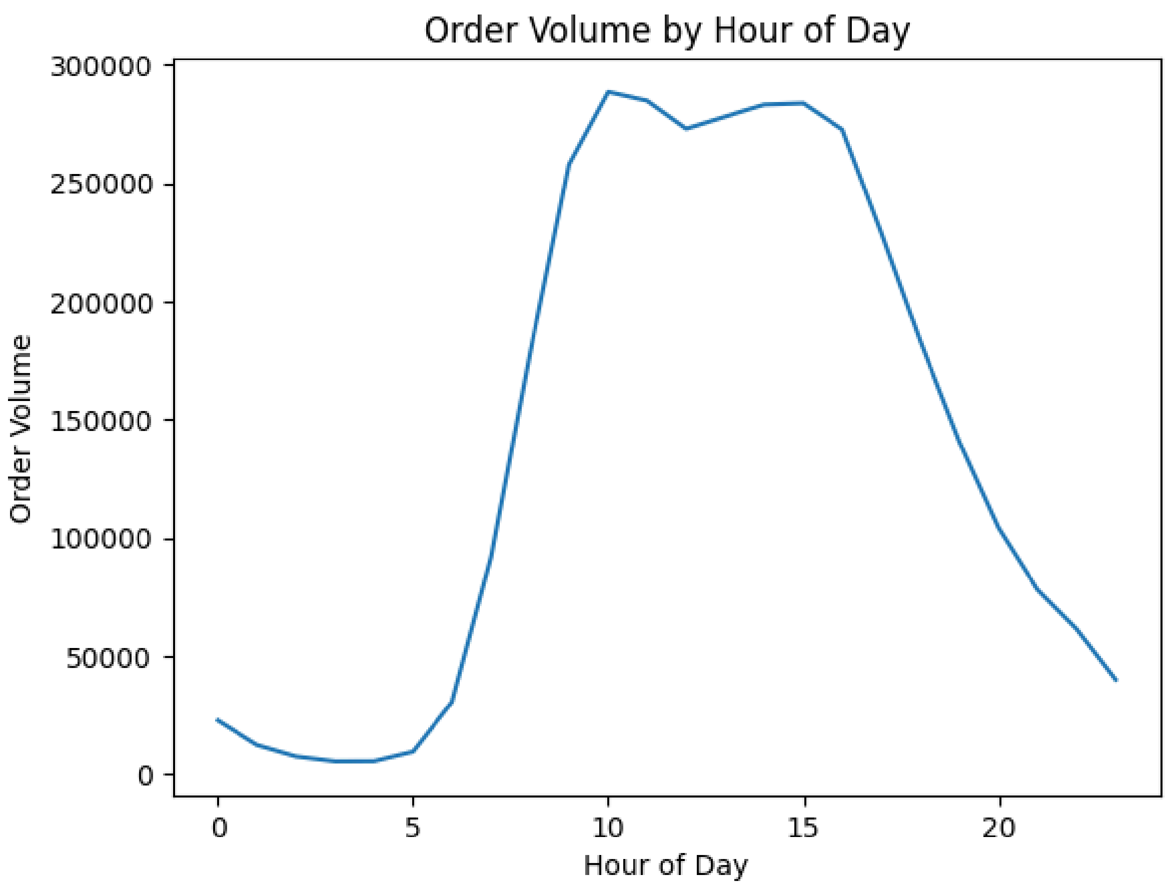
	department	items_sold
19	produce	409087
7	dairy eggs	217051
20	snacks	118862
3	beverages	114046
10	frozen	100426
16	pantry	81242
2	bakery	48394
6	canned goods	46799
8	deli	44291
9	dry goods pasta	38713
11	household	35986
13	meat seafood	30307
4	breakfast	29500
17	personal care	21570
1	babies	14941
12	international	11902
14	missing	8251
0	alcohol	5598
18	pets	4497
15	other	1795
5	bulk	1359

In [24]:
```
plt.figure(figsize=(15, 6))
plt.bar(items_sold_by_department['department'], items_sold_by_department['item
plt.xlabel('Department')
plt.ylabel('Items Sold')
plt.title('Items Sold by Department')
plt.xticks(rotation=45)
#plt.savefig(r"C:\Users\richa\COMP 541 Data Mining\Item Sold by Department.png
plt.show()
```



Calculate Order Volume By Hour of Day

In [25]:
```
hourly_trends = orders_df.groupby('order_hour_of_day')['order_id'].count().res
hourly_trends.columns = ['hour', 'order_count']

plt.plot(hourly_trends['hour'], hourly_trends['order_count'])
plt.title('Order Volume by Hour of Day')
plt.xlabel('Hour of Day')
plt.ylabel('Order Volume')
#plt.savefig(r"C:\Users\richa\COMP 541 Data Mining\Order Volume by Hour of Day
plt.show()
```



Calculate Top 10 Most Popular Products

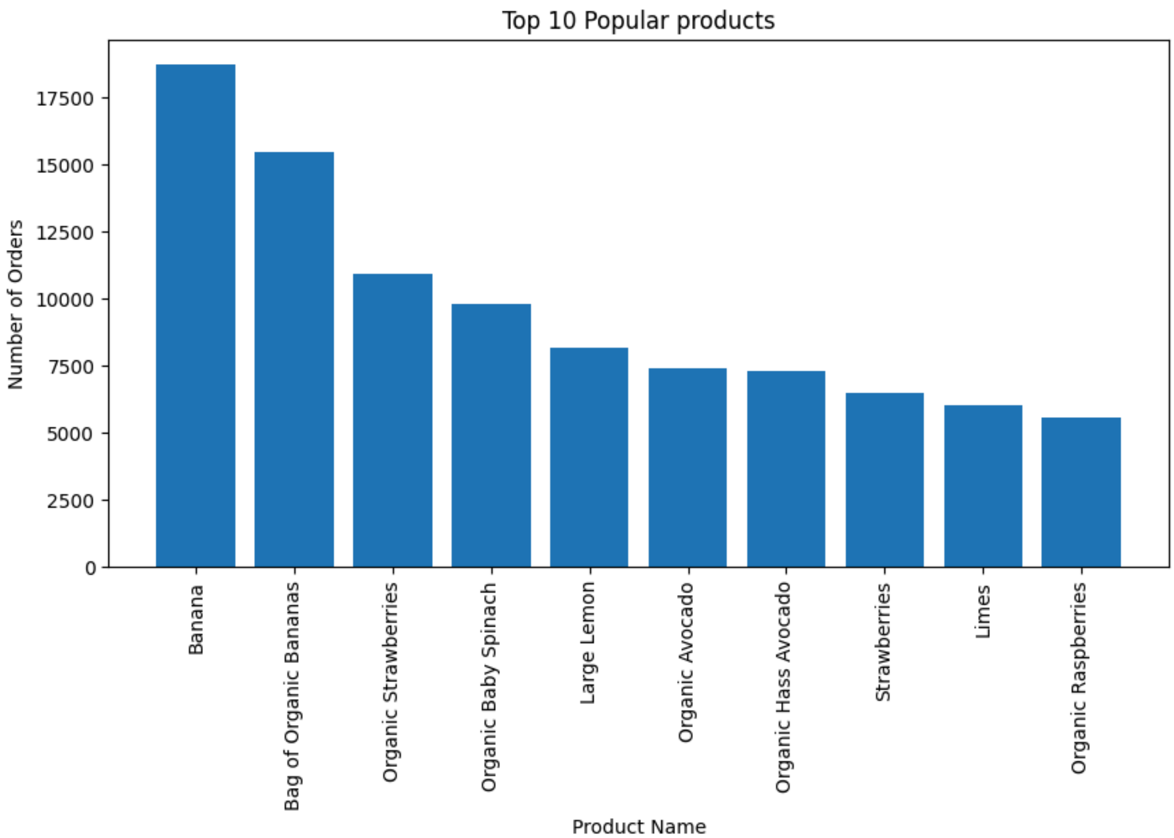
In [26]:

```
# Calculating frequency of each product and getting only the top 10
top_products = order_products_df['product_id'].value_counts().head(10)

# Merging product frequency with product names
top_products = top_products.reset_index().merge(products_df[['product_id', 'pr

# Changing column name from count to frequency
top_products.rename(columns={'count': 'frequency'}, inplace=True)

# Displaying the top 10 products and frequency of each
top_products
```

Out[26]:

	product_name	frequency
0	Banana	18726
1	Bag of Organic Bananas	15480
2	Organic Strawberries	10894
3	Organic Baby Spinach	9784
4	Large Lemon	8135
5	Organic Avocado	7409
6	Organic Hass Avocado	7293
7	Strawberries	6494
8	Limes	6033
9	Organic Raspberries	5546

In [27]:
```
# Using Bar Chart to visualize top 10 data
plt.figure(figsize=(10,5))
plt.bar(top_products['product_name'], top_products['frequency'])
plt.xlabel('Product Name')
plt.ylabel('Number of Orders')
plt.title('Top 10 Popular products')
plt.xticks(rotation = 90)
plt.show()
```



Calculate Distribution of the number of products per transaction

In [28]:
```
# Calculating the number of products per order
products_per_order = order_products_df.groupby('order_id').size().reset_index(

print(products_per_order['product_count'].describe())
products_per_order
```

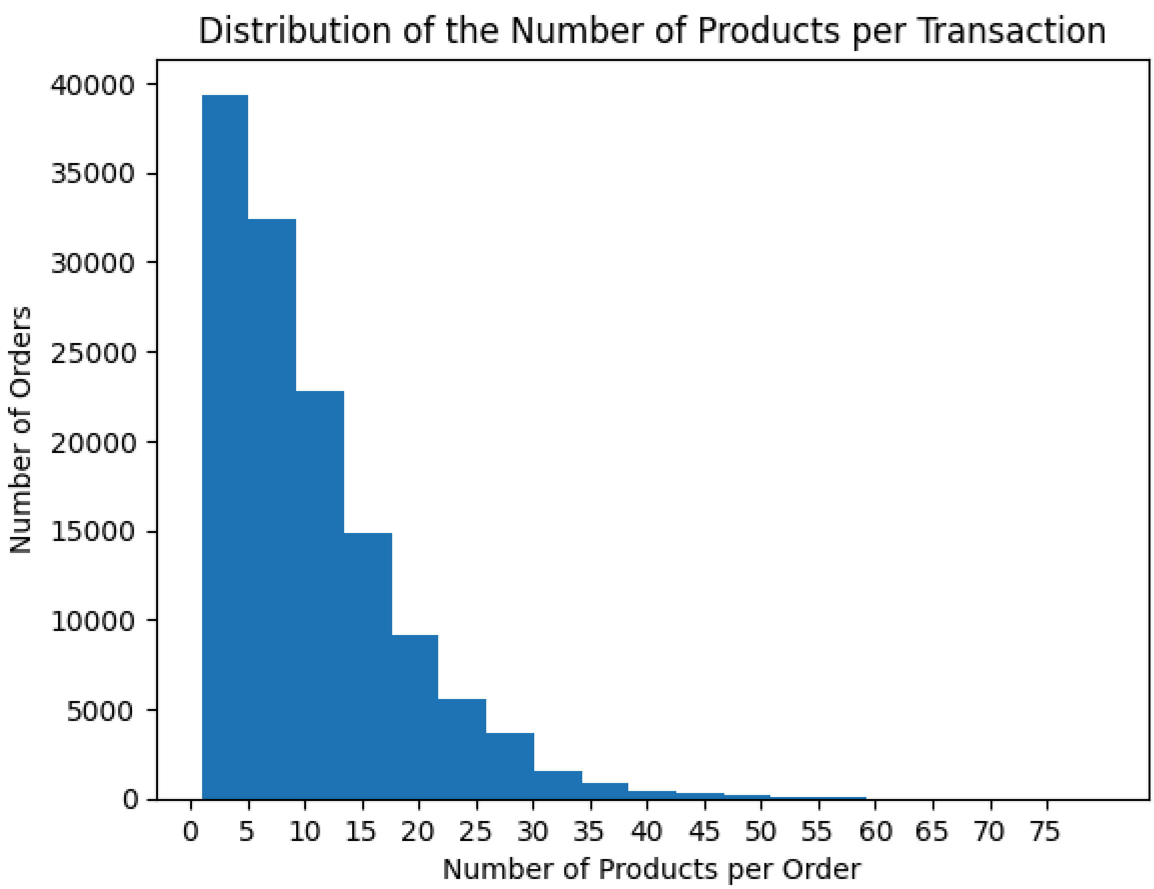
```
count    131209.000000
mean         10.552759
std           7.932847
min           1.000000
25%           5.000000
50%           9.000000
75%          14.000000
max          80.000000
Name: product_count, dtype: float64
```

Out[28]:

	order_id	product_count
0	1	8
1	36	8
2	38	9
3	96	7
4	98	49
...	...	...
131204	3421049	6
131205	3421056	5
131206	3421058	8
131207	3421063	4
131208	3421070	3

131209 rows × 2 columns

In [29]:
```python
# Using Histogram to visualize the distribution of the number of products per
import numpy as np
number_of_bins = int(np.ceil(np.log2(len(products_per_order)) + 1))

plt.hist(products_per_order['product_count'], bins = number_of_bins)
plt.xlabel('Number of Products per Order')
plt.ylabel('Number of Orders')
plt.xticks(np.arange(0, max(products_per_order['product_count']), 5))
plt.title('Distribution of the Number of Products per Transaction')
plt.show()
```



Part 3 Mine Strong Associations Across Different Departments and Across Aisles in Departments

In [30]:

```
# Merge products with aisles and departments to include aisle and department d
products_full_df = pd.merge(products_df, aisles_df, on='aisle_id', how='left')
products_full_df = pd.merge(products_full_df, departments_df, on='department_i

# Merge the products_full_df with order_products_df
order_products_merged_df = pd.merge(order_products_df, products_full_df, on='p


order_products_merged_df = order_products_merged_df[['order_id', 'product_name


order_products_merged_df.head()
```

Out[30]:

	order_id	product_name	aisle	department	department_id	aisle_id
0	1	Bulgarian Yogurt	yogurt	dairy eggs	16	120
1	1	Organic 4% Milk Fat Whole Milk Cottage Cheese	other creams cheeses	dairy eggs	16	108
2	1	Organic Celery Hearts	fresh vegetables	produce	4	83
3	1	Cucumber Kirby	fresh vegetables	produce	4	83
4	1	Lightly Smoked Sardines in Olive Oil	canned meat seafood	canned goods	15	95

In [31]:
```python
from mlxtend.preprocessing import TransactionEncoder

# Group products by order_id
grouped_products = order_products_merged_df.groupby('order_id')['product_name'

# Use TransactionEncoder to create a basket (sparse matrix)
te = TransactionEncoder()
te_ary = te.fit(grouped_products).transform(grouped_products)
basket_products = pd.DataFrame(te_ary, columns=te.columns_)
basket_products.head()
```

Out[31]:

	#2 Coffee Filters	#2 Cone White Coffee Filters	#2 Mechanical Pencils	#4 Natural Brown Coffee Filters	& Go! Hazelnut Spread + Pretzel Sticks	+Energy Black Cherry Vegetable & Fruit Juice	0 Calorie Acai Raspberry Water Beverage	0 Calorie Fuji Apple Pear Water Beverage	0 Calorie Strawberry Dragonfruit Water Beverage
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False

5 rows × 39123 columns

In [32]:
```python
# Group aisles by order_id
grouped_aisles = order_products_merged_df.groupby('order_id')['aisle'].apply(l

te_ary = te.fit(grouped_aisles).transform(grouped_aisles)
basket_aisles = pd.DataFrame(te_ary, columns=te.columns_)
basket_aisles.head()
```

Out[32]:

	air fresheners candles	asian foods	baby accessories	baby bath body care	baby food formula	bakery desserts	baking ingredients	baking supplies decor	beauty	bee coole
0	False	False	False	False	False	False	False	False	False	Fal
1	False	False	False	False	False	False	False	False	False	Fal
2	False	False	False	False	False	False	False	False	False	Fal
3	False	False	False	False	False	False	False	False	False	Fal
4	False	False	False	False	False	False	True	False	False	Fal

5 rows × 134 columns

In [33]:
```
# Group departments by order_id
grouped_departments = order_products_merged_df.groupby('order_id')['department

te_ary = te.fit(grouped_departments).transform(grouped_departments)
basket_departments = pd.DataFrame(te_ary, columns=te.columns_)
basket_departments.head()
```

Out[33]:

	alcohol	babies	bakery	beverages	breakfast	bulk	canned goods	dairy eggs	deli	dry goods pasta	...	housel
0	False	False	False	False	False	False	True	True	False	False	...	F
1	False	False	False	True	False	False	False	True	True	False	...	F
2	False	False	False	False	False	False	False	False	False	False	...	F
3	False	False	False	False	False	False	False	False	True	False	...	F
4	False	False	True	True	False	False	True	True	True	False	...	

5 rows × 21 columns

uses the FP-Growth algorithm to find frequent itemsets in the product-level transactional data

In [34]:
```
from mlxtend.frequent_patterns import fpgrowth

# Frequent itemsets for products
frequent_itemsets_products = fpgrowth(basket_products, min_support=0.005, use_

frequent_itemsets_products.head()
```

Out[34]:

	support	itemsets
0	0.117980	(Bag of Organic Bananas)
1	0.055583	(Organic Hass Avocado)
2	0.018391	(Cucumber Kirby)
3	0.015190	(Organic Whole String Cheese)
4	0.008094	(Organic Celery Hearts)

In [35]:
```
# Frequent itemsets for aisles
frequent_itemsets_aisles = fpgrowth(basket_aisles, min_support=0.005, use_coln
frequent_itemsets_aisles.head()
```

Out[35]:

	support	itemsets
0	0.550099	(fresh fruits)
1	0.450975	(fresh vegetables)
2	0.253405	(yogurt)
3	0.237781	(packaged cheese)
4	0.088096	(other creams cheeses)

In [36]:
```
# Frequent itemsets for departments
frequent_itemsets_departments = fpgrowth(basket_departments, min_support=0.005
frequent_itemsets_departments.head()
```

Out[36]:

	support	itemsets
0	0.738722	(produce)
1	0.666113	(dairy eggs)
2	0.224192	(canned goods)
3	0.468581	(beverages)
4	0.246027	(deli)

In [37]:
```
from mlxtend.frequent_patterns import association_rules

# Association rules for products
rules_products = association_rules(frequent_itemsets_products, metric="confide

# Filter for cross-department rules
department_map = products_full_df.set_index('product_name')['department']
rules_products['antecedent_departments'] = rules_products['antecedents'].apply
rules_products['consequent_departments'] = rules_products['consequents'].apply
cross_department_rules_products = rules_products[
    rules_products['antecedent_departments'] != rules_products['consequent_dep
]
cross_department_rules_products[['antecedents', 'consequents', 'support', 'con
```

Out[37]:

	antecedents	consequents	support	confidence	lift
2	(Bag of Organic Bananas, Organic Strawberries)	(Organic Hass Avocado)	0.005411	0.230969	4.155391
3	(Bag of Organic Bananas, Organic Hass Avocado)	(Organic Strawberries)	0.005411	0.293388	3.533615
4	(Organic Strawberries, Organic Hass Avocado)	(Bag of Organic Bananas)	0.005411	0.461339	3.910321
38	(Blueberries)	(Banana)	0.005457	0.308222	2.159645
42	(Organic Whole Milk)	(Bag of Organic Bananas)	0.008521	0.227791	1.930767
43	(Organic Whole Milk)	(Banana)	0.007957	0.212714	1.490440

In [38]:
```
# Association rules for aisles
rules_aisles = association_rules(frequent_itemsets_aisles, metric="confidence"

# Inspect the most interesting rules
rules_aisles[['antecedents', 'consequents', 'support', 'confidence', 'lift']]
```

Out[38]:

	antecedents	consequents	support	confidence	lift
0	(fresh vegetables)	(fresh fruits)	0.327333	0.725833	1.319458
1	(fresh fruits)	(fresh vegetables)	0.327333	0.595043	1.319458
2	(yogurt)	(fresh fruits)	0.185102	0.730458	1.327865
3	(fresh fruits)	(yogurt)	0.185102	0.336488	1.327865
4	(yogurt)	(fresh vegetables)	0.146682	0.578844	1.283540
...	...	...	...	...	...
53914	(seafood counter)	(fresh fruits)	0.006288	0.770308	1.400307
53915	(seafood counter)	(packaged vegetables fruits)	0.005053	0.619048	1.616185
53916	(seafood counter, fresh vegetables)	(fresh fruits)	0.005358	0.829009	1.507017
53917	(seafood counter, fresh fruits)	(fresh vegetables)	0.005358	0.852121	1.889508
53918	(seafood counter)	(fresh vegetables, fresh fruits)	0.005358	0.656396	2.005286

53919 rows × 5 columns

In [39]:
```python
# Association rules for departments
rules_departments = association_rules(frequent_itemsets_departments, metric="c

# Inspect the most interesting rules
rules_departments[['antecedents', 'consequents', 'support', 'confidence', 'lif
```

Out[39]:

	antecedents	consequents	support	confidence	lift
0	(dairy eggs)	(produce)	0.543835	0.816430	1.105192
1	(produce)	(dairy eggs)	0.543835	0.736183	1.105192
2	(canned goods)	(produce)	0.195863	0.873640	1.182637
3	(produce)	(canned goods)	0.195863	0.265138	1.182637
4	(dairy eggs)	(canned goods)	0.179157	0.268959	1.199681
...	...	...	...	...	...
180097	(pets, pantry, produce)	(dairy eggs)	0.005983	0.858862	1.289364
180098	(dairy eggs, pets)	(pantry, produce)	0.005983	0.411857	1.382966
180099	(pets, pantry)	(dairy eggs, produce)	0.005983	0.673820	1.239016
180100	(pets, produce)	(dairy eggs, pantry)	0.005983	0.442503	1.578757
180101	(pets)	(dairy eggs, pantry, produce)	0.005983	0.287651	1.186607

180102 rows × 5 columns

In [40]:

```
# Filter product-level rules where antecedents and consequents belong to diffe
aisle_map = products_full_df.set_index('product_name')['aisle']
rules_products['antecedent_aisles'] = rules_products['antecedents'].apply(lamb
rules_products['consequent_aisles'] = rules_products['consequents'].apply(lamb
cross_aisle_rules_products = rules_products[
    rules_products['antecedent_aisles'] != rules_products['consequent_aisles']
]
cross_aisle_rules_products[['antecedents', 'consequents', 'support', 'confiden
```

Out[40]:

	antecedents	consequents	support	confidence	lift
2	(Bag of Organic Bananas, Organic Strawberries)	(Organic Hass Avocado)	0.005411	0.230969	4.155391
3	(Bag of Organic Bananas, Organic Hass Avocado)	(Organic Strawberries)	0.005411	0.293388	3.533615
4	(Organic Strawberries, Organic Hass Avocado)	(Bag of Organic Bananas)	0.005411	0.461339	3.910321
5	(Cucumber Kirby)	(Banana)	0.005663	0.307915	2.157496
6	(Asparagus)	(Banana)	0.006044	0.205016	1.436499
7	(Organic Raspberries)	(Bag of Organic Bananas)	0.013566	0.320952	2.720400
8	(Organic Raspberries)	(Organic Strawberries)	0.012728	0.301118	3.626710
9	(Organic Blueberries)	(Organic Strawberries)	0.009672	0.255538	3.077735
10	(Organic Blueberries)	(Bag of Organic Bananas)	0.008666	0.228957	1.940646
11	(Organic Cucumber)	(Organic Strawberries)	0.007865	0.223716	2.694465
12	(Organic Cucumber)	(Bag of Organic Bananas)	0.009664	0.274875	2.329853
13	(Organic Cucumber)	(Organic Baby Spinach)	0.007111	0.202254	2.712348
14	(Organic Grape Tomatoes)	(Banana)	0.006623	0.227308	1.592700
16	(Organic Grape Tomatoes)	(Bag of Organic Bananas)	0.006250	0.214491	1.818035
17	(Organic Zucchini)	(Bag of Organic Bananas)	0.007934	0.226847	1.922761
18	(Organic Zucchini)	(Banana)	0.007157	0.204620	1.433726
19	(Organic Zucchini)	(Organic Baby Spinach)	0.007240	0.207017	2.776213
20	(Organic Yellow Onion)	(Bag of Organic Bananas)	0.007621	0.233100	1.975765
21	(Organic Garlic)	(Bag of Organic Bananas)	0.006951	0.219336	1.859101
28	(Organic Baby Carrots)	(Bag of Organic Bananas)	0.006288	0.229358	1.944044
29	(Organic Baby Carrots)	(Banana)	0.005632	0.205449	1.439536
30	(Organic Cilantro)	(Limes)	0.007675	0.285593	6.211228
31	(Organic Cilantro)	(Bag of Organic Bananas)	0.005442	0.202496	1.716361
32	(Broccoli Crown)	(Banana)	0.007050	0.315484	2.210530
33	(Organic Baby Spinach)	(Banana)	0.015243	0.204415	1.432294
34	(Organic Baby Spinach)	(Bag of Organic Bananas)	0.017042	0.228536	1.937082
38	(Blueberries)	(Banana)	0.005457	0.308222	2.159645

	antecedents	consequents	support	confidence	lift
42	(Organic Whole Milk)	(Bag of Organic Bananas)	0.008521	0.227791	1.930767
43	(Organic Whole Milk)	(Banana)	0.007957	0.212714	1.490440
44	(Red Peppers)	(Banana)	0.006387	0.288468	2.021234
46	(Seedless Red Grapes)	(Banana)	0.008856	0.286277	2.005883
47	(Organic Red Onion)	(Bag of Organic Bananas)	0.006135	0.210843	1.787116
49	(Raspberries)	(Strawberries)	0.005015	0.200671	4.054486
54	(Yellow Onions)	(Banana)	0.008163	0.284689	1.994754

In [41]:

```python
from mlxtend.frequent_patterns import fpgrowth

# Generate frequent itemsets using FP-Growth
frequent_itemsets_fp = fpgrowth(basket_products, min_support=0.005, use_colnam

frequent_itemsets_fp.head()
```

Out[41]:

	support	itemsets
0	0.117980	(Bag of Organic Bananas)
1	0.055583	(Organic Hass Avocado)
2	0.018391	(Cucumber Kirby)
3	0.015190	(Organic Whole String Cheese)
4	0.008094	(Organic Celery Hearts)

In [42]:

```python
from mlxtend.frequent_patterns import association_rules

# Generate association rules
rules_fp = association_rules(frequent_itemsets_fp, metric="confidence", min_th


rules_fp.head()
```

Out[42]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage
0	(Organic Hass Avocado)	(Bag of Organic Bananas)	0.055583	0.117980	0.018444	0.331825	2.812560	0.011886
1	(Organic Hass Avocado)	(Organic Strawberries)	0.055583	0.083028	0.011729	0.211024	2.541609	0.007114
2	(Bag of Organic Bananas, Organic Strawberries)	(Organic Hass Avocado)	0.023428	0.055583	0.005411	0.230969	4.155391	0.004109
3	(Bag of Organic Bananas, Organic Hass Avocado)	(Organic Strawberries)	0.018444	0.083028	0.005411	0.293388	3.533615	0.003880
4	(Organic Strawberries, Organic Hass Avocado)	(Bag of Organic Bananas)	0.011729	0.117980	0.005411	0.461339	3.910321	0.004027

In [43]:
```
# Filter for rules with lift > 1.5 and confidence > 0.3
high_lift_rules = rules_fp[(rules_fp['lift'] > 1.5) & (rules_fp['confidence']

# Display only relevant columns
high_lift_rules[['antecedents', 'consequents', 'support', 'confidence', 'lift'
```

Out[43]:

	antecedents	consequents	support	confidence	lift
0	(Organic Hass Avocado)	(Bag of Organic Bananas)	0.018444	0.331825	2.812560
4	(Organic Strawberries, Organic Hass Avocado)	(Bag of Organic Bananas)	0.005411	0.461339	3.910321
5	(Cucumber Kirby)	(Banana)	0.005663	0.307915	2.157496
7	(Organic Raspberries)	(Bag of Organic Bananas)	0.013566	0.320952	2.720400
8	(Organic Raspberries)	(Organic Strawberries)	0.012728	0.301118	3.626710
24	(Organic Lemon)	(Bag of Organic Bananas)	0.008132	0.304422	2.580293
32	(Broccoli Crown)	(Banana)	0.007050	0.315484	2.210530
37	(Honeycrisp Apple)	(Banana)	0.009382	0.346663	2.428991
38	(Blueberries)	(Banana)	0.005457	0.308222	2.159645
40	(Organic Large Extra Fancy Fuji Apple)	(Bag of Organic Bananas)	0.007416	0.336562	2.852709
41	(Apple Honeycrisp Organic)	(Bag of Organic Bananas)	0.005236	0.305062	2.585717
48	(Organic Fuji Apple)	(Banana)	0.009222	0.371508	2.603072
52	(Organic Navel Orange)	(Bag of Organic Bananas)	0.005526	0.366162	3.103598

Cross-Department Insights

Highlights relationships between products in different departments, useful for marketing strategies and merchandising.

High-Lift Rules: Focuses on rules with strong associations (high lift) to prioritize actionable insights.

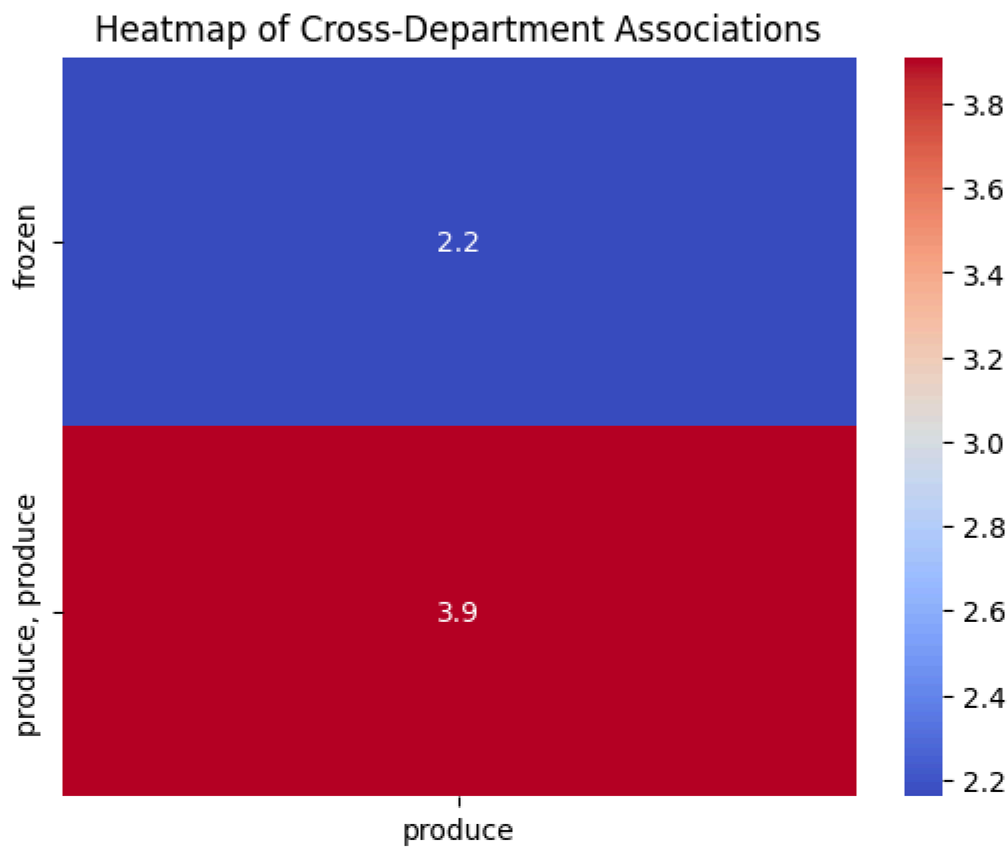
In [44]:
```
# Map products to departments
department_map = products_full_df.set_index('product_name')['department']

# Add department information to rules
high_lift_rules.loc[:, 'antecedent_departments'] = high_lift_rules['antecedent
high_lift_rules.loc[:, 'consequent_departments'] = high_lift_rules['consequent

# Filter for rules where antecedents and consequents come from different depar
cross_department_rules = high_lift_rules[
    high_lift_rules['antecedent_departments'] != high_lift_rules['consequent_d
]

cross_department_rules[['antecedents', 'consequents', 'support', 'confidence',
```

```
C:\Users\Nexxa\AppData\Local\Temp\ipykernel_33028\52806777.py:5: SettingWithC
opyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  high_lift_rules.loc[:, 'antecedent_departments'] = high_lift_rules['anteced
ents'].apply(lambda x: [department_map[item] for item in x])
C:\Users\Nexxa\AppData\Local\Temp\ipykernel_33028\52806777.py:6: SettingWithC
opyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  high_lift_rules.loc[:, 'consequent_departments'] = high_lift_rules['consequ
ents'].apply(lambda x: [department_map[item] for item in x])
```

Out[44]:

	antecedents	consequents	support	confidence	lift	antecedent_departments	consequ
4	(Organic Strawberries, Organic Hass Avocado)	(Bag of Organic Bananas)	0.005411	0.461339	3.910321	[produce, produce]	
38	(Blueberries)	(Banana)	0.005457	0.308222	2.159645	[frozen]	

In [91]:
```
import seaborn as sns

# Flatten the lists into strings
cross_department_rules['antecedent_departments_str'] = cross_department_rules[
cross_department_rules['consequent_departments_str'] = cross_department_rules[

# Create a pivot table for the heatmap
heatmap_data = cross_department_rules.pivot_table(index='antecedent_department

sns.heatmap(heatmap_data, annot=True, cmap="coolwarm")
plt.xlabel("")
plt.ylabel("")
plt.title("Heatmap of Cross-Department Associations")
plt.show()
```



In [46]:

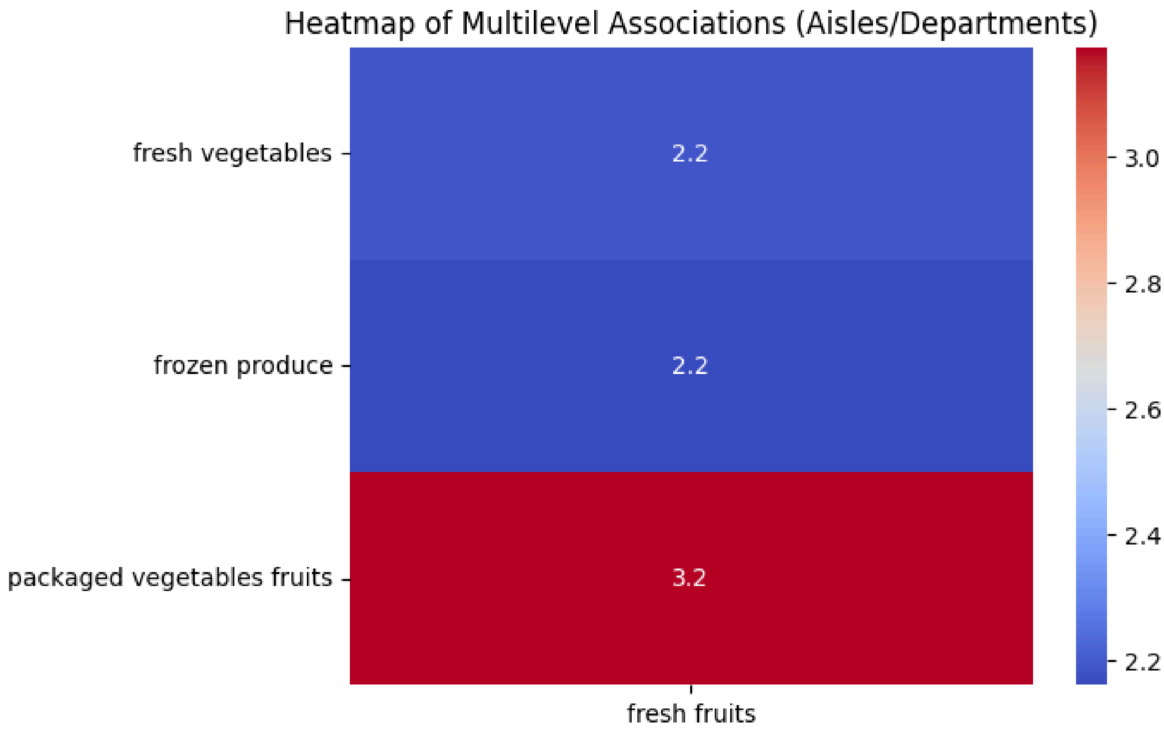
```python
# Map products to aisles
aisle_map = products_full_df.set_index('product_name')['aisle']

# Add aisle information to antecedents and consequents
high_lift_rules = high_lift_rules.copy()  # Avoid modifying the original DataF
high_lift_rules['antecedent_aisles'] = high_lift_rules['antecedents'].apply(
    lambda x: {aisle_map[item] for item in x})
high_lift_rules['consequent_aisles'] = high_lift_rules['consequents'].apply(
    lambda x: {aisle_map[item] for item in x})

# Filter for cross-aisle rules
cross_aisle_rules = high_lift_rules[
    high_lift_rules['antecedent_aisles'] != high_lift_rules['consequent_aisles
]

# Sort by support and lift to find the most diverse and impactful rules
diverse_rules = cross_aisle_rules.sort_values(by=['lift', 'support'], ascendin

# Select top interesting and diverse rules
output_columns = ['antecedents', 'consequents', 'support', 'confidence', 'lift
                  'antecedent_aisles', 'consequent_aisles']
top_diverse_rules = diverse_rules[output_columns].head(10)


top_diverse_rules
```

Out[46]:

	antecedents	consequents	support	confidence	lift	antecedent_aisles	consequent_ai:
8	(Organic Raspberries)	(Organic Strawberries)	0.012728	0.301118	3.626710	{packaged vegetables fruits}	{fresh fr
7	(Organic Raspberries)	(Bag of Organic Bananas)	0.013566	0.320952	2.720400	{packaged vegetables fruits}	{fresh fr
32	(Broccoli Crown)	(Banana)	0.007050	0.315484	2.210530	{fresh vegetables}	{fresh fr
38	(Blueberries)	(Banana)	0.005457	0.308222	2.159645	{frozen produce}	{fresh fr
5	(Cucumber Kirby)	(Banana)	0.005663	0.307915	2.157496	{fresh vegetables}	{fresh fr

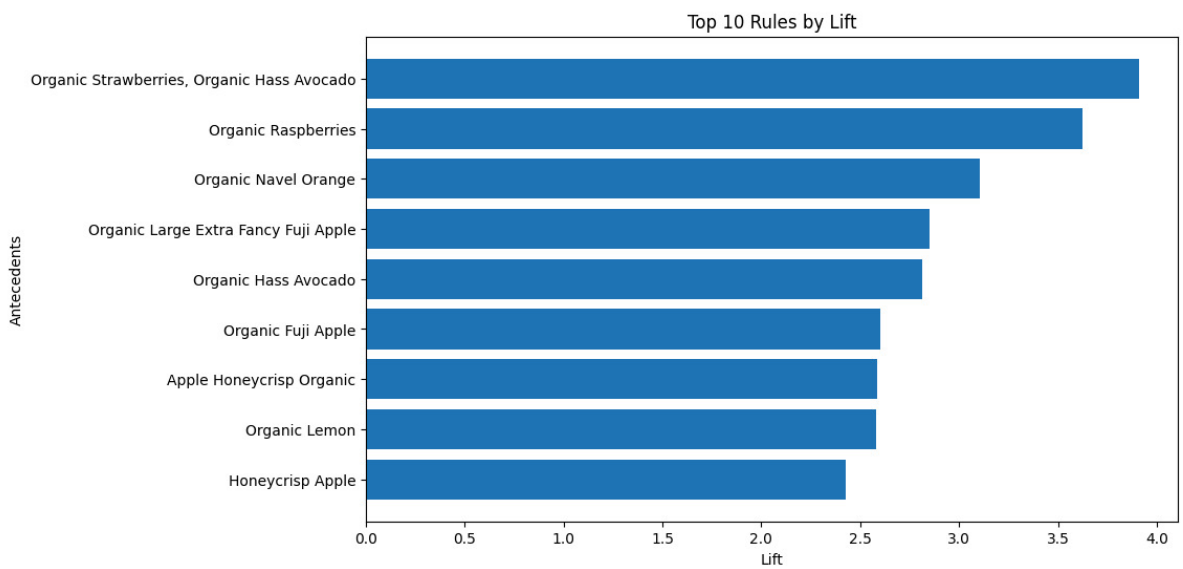
In [92]:
```python
import seaborn as sns

# Flatten the lists into strings
cross_aisle_rules['antecedent_aisles_str'] = cross_aisle_rules['antecedent_ais
cross_aisle_rules['consequent_aisles_str'] = cross_aisle_rules['consequent_ais

# Create a pivot table for the heatmap
heatmap_data = cross_aisle_rules.pivot_table(index='antecedent_aisles_str', co

sns.heatmap(heatmap_data, annot=True, cmap="coolwarm")
plt.xlabel("")
plt.ylabel("")
plt.title("Heatmap of Multilevel Associations (Aisles/Departments)")
plt.show()
```

```
C:\Users\Nexxa\AppData\Local\Temp\ipykernel_33028\1760451142.py:4: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  cross_aisle_rules['antecedent_aisles_str'] = cross_aisle_rules['antecedent_
aisles'].apply(lambda x: ', '.join(x))
C:\Users\Nexxa\AppData\Local\Temp\ipykernel_33028\1760451142.py:5: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  cross_aisle_rules['consequent_aisles_str'] = cross_aisle_rules['consequent_
aisles'].apply(lambda x: ', '.join(x))
```



Heatmap of Multilevel Associations (Aisles/Departments)
fresh vegetables
frozen produce
packaged vegetables fruits
fresh fruits
2.2
2.2
3.2
3.0
2.8
2.6
2.4
2.2

In [48]:
```
import matplotlib.pyplot as plt

# Sort rules by lift and select the top 10
top_lift_rules = high_lift_rules.sort_values(by='lift', ascending=False).head(


plt.figure(figsize=(10, 6))
plt.barh(
    [', '.join([str(i) for i in rule]) for rule in top_lift_rules['antecedents
    top_lift_rules['lift']
)
plt.xlabel('Lift')
plt.ylabel('Antecedents')
plt.title('Top 10 Rules by Lift')
plt.gca().invert_yaxis()
#plt.savefig(r"C:\Users\richa\COMP 541 Data Mining\Project\top 10 rules by lif
plt.show()
```



Extra Credits

Predict Order Size Using Multiple Regression

In [49]:
```
order_count = order_products_df.groupby('order_id')['product_id'].count().rese
order_count.rename(columns={'product_id': 'order_size'}, inplace=True)
```

In [50]:
```
order_count
```

Out[50]:

	order_id	order_size
0	1	8
1	36	8
2	38	9
3	96	7
4	98	49
...	...	...
131204	3421049	6
131205	3421056	5
131206	3421058	8
131207	3421063	4
131208	3421070	3

131209 rows × 2 columns

In [51]:
```
orders_with_size_df = pd.merge(orders_df, order_count, on='order_id')
```

In [52]:
```
orders_with_size_df.head()
```

Out[52]:

	order_id	user_id	eval_set	order_number	order_dow	order_hour_of_day	days_since_prior_o
0	1187899	1	train	11	4	8	
1	1492625	2	train	15	1	11	
2	2196797	5	train	5	0	11	
3	525192	7	train	21	2	11	
4	880375	8	train	4	1	14	

Feature Engineering

In [53]:
```
order_size_user_avg = orders_with_size_df.groupby('user_id')['order_size'].mea
order_size_user_avg.rename(columns={'order_size': 'user_avg_order_size'}, inpl
```

In [54]:
```
order_size_user_avg
```

Out[54]:

	user_id	user_avg_order_size
0	1	11.0
1	2	31.0
2	5	9.0
3	7	9.0
4	8	18.0
...	...	...
131204	206199	22.0
131205	206200	19.0
131206	206203	13.0
131207	206205	19.0
131208	206209	8.0

131209 rows × 2 columns

In [55]:
```
orders_with_size_df = pd.merge(orders_with_size_df, order_size_user_avg, on='u

orders_with_size_df.head()
```

Out[55]:

	order_id	user_id	eval_set	order_number	order_dow	order_hour_of_day	days_since_prior_o
0	1187899	1	train	11	4	8	
1	1492625	2	train	15	1	11	
2	2196797	5	train	5	0	11	
3	525192	7	train	21	2	11	
4	880375	8	train	4	1	14	

In [56]:
```
import statsmodels.api as sm
import statsmodels.formula.api as smf

orders_with_size_dummy_df = pd.get_dummies(orders_with_size_df, columns=['orde

orders_with_size_dummy_df.head()
```

Out[56]:

	order_id	user_id	eval_set	order_number	days_since_prior_order	order_size	user_avg_order
0	1187899	1	train	11	14.0	11	
1	1492625	2	train	15	30.0	31	
2	2196797	5	train	5	6.0	9	
3	525192	7	train	21	6.0	9	
4	880375	8	train	4	10.0	18	

5 rows × 36 columns

In [57]:
```
print(orders_with_size_dummy_df.columns)
```

```
Index(['order_id', 'user_id', 'eval_set', 'order_number',
       'days_since_prior_order', 'order_size', 'user_avg_order_size',
       'order_dow_1', 'order_dow_2', 'order_dow_3', 'order_dow_4',
       'order_dow_5', 'order_dow_6', 'order_hour_of_day_1',
       'order_hour_of_day_2', 'order_hour_of_day_3', 'order_hour_of_day_4',
       'order_hour_of_day_5', 'order_hour_of_day_6', 'order_hour_of_day_7',
       'order_hour_of_day_8', 'order_hour_of_day_9', 'order_hour_of_day_10',
       'order_hour_of_day_11', 'order_hour_of_day_12', 'order_hour_of_day_1
3',
       'order_hour_of_day_14', 'order_hour_of_day_15', 'order_hour_of_day_1
6',
       'order_hour_of_day_17', 'order_hour_of_day_18', 'order_hour_of_day_1
9',
       'order_hour_of_day_20', 'order_hour_of_day_21', 'order_hour_of_day_2
2',
       'order_hour_of_day_23'],
      dtype='object')
```

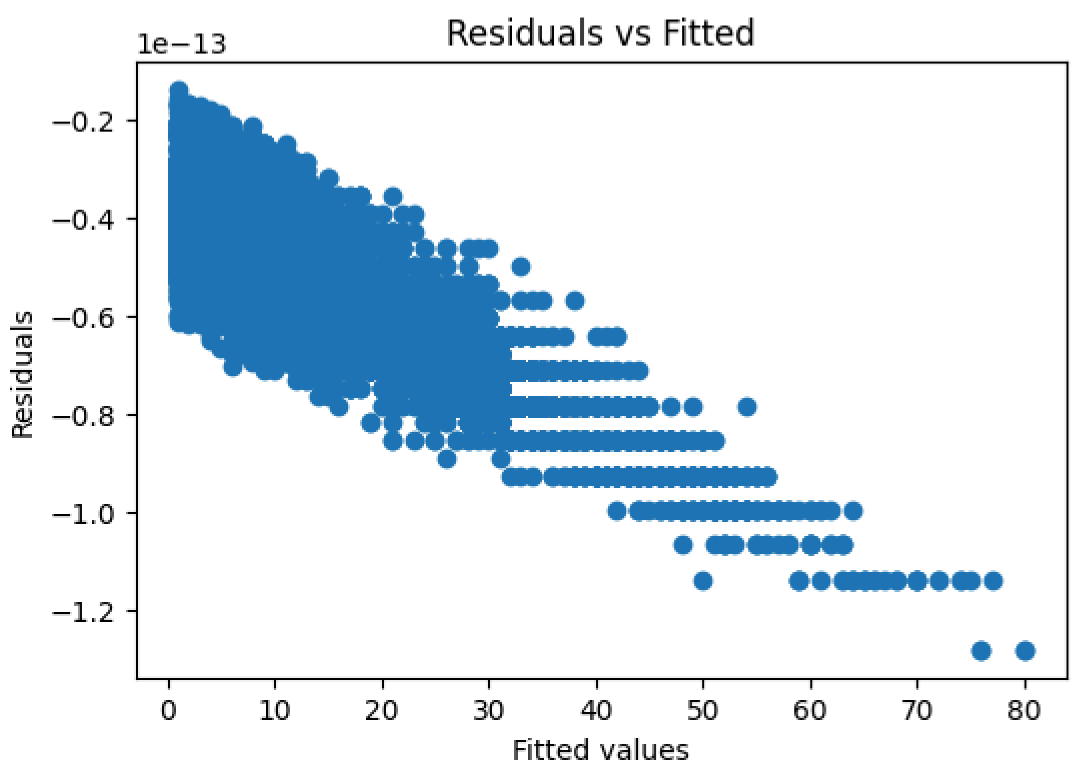
In [58]:
```
formula = 'order_size ~ order_number + days_since_prior_order + user_avg_order
          'order_dow_1 + order_dow_2 + order_dow_3 + order_dow_4 + order_dow_5
          'order_hour_of_day_1 + order_hour_of_day_2 + order_hour_of_day_3 + o
          'order_hour_of_day_5 + order_hour_of_day_6 + order_hour_of_day_7 + o
          'order_hour_of_day_9 + order_hour_of_day_10 + order_hour_of_day_11 +
          'order_hour_of_day_13 + order_hour_of_day_14 + order_hour_of_day_15
          'order_hour_of_day_17 + order_hour_of_day_18 + order_hour_of_day_19
          'order_hour_of_day_21 + order_hour_of_day_22 + order_hour_of_day_23'

model = smf.ols(formula=formula, data=orders_with_size_dummy_df).fit()

print(model.summary())
```

```
                            OLS Regression Results
=============================================================================
=
Dep. Variable:             order_size   R-squared:                       1.00
0
Model:                            OLS   Adj. R-squared:                  1.00
0
Method:                 Least Squares   F-statistic:                 9.984e+3
1
Date:                Fri, 29 Nov 2024   Prob (F-statistic):               0.0
0
Time:                        16:23:25   Log-Likelihood:            3.8302e+0
6
No. Observations:              131209   AIC:                       -7.660e+0
6
Df Residuals:                  131176   BIC:                       -7.660e+0
6
Df Model:                          32
Covariance Type:            nonrobust
=============================================================================
==================
                                  coef    std err          t      P>|t|
[0.025      0.975]
-----------------------------------------------------------------------------
------------------
Intercept                    3.136e-14   1.81e-15     17.364      0.000
2.78e-14    3.49e-14
order_dow_1[T.True]         -3.034e-15   4.76e-16     -6.373      0.000   -
3.97e-15    -2.1e-15
order_dow_2[T.True]          5.922e-15   5.06e-16     11.702      0.000
4.93e-15    6.91e-15
order_dow_3[T.True]          8.476e-15    5.1e-16     16.607      0.000
7.48e-15    9.48e-15
order_dow_4[T.True]          2.595e-15   5.08e-16      5.110      0.000
1.6e-15    3.59e-15
order_dow_5[T.True]         -4.108e-15   4.94e-16     -8.319      0.000   -
5.08e-15   -3.14e-15
order_dow_6[T.True]          2.398e-15   4.81e-16      4.985      0.000
1.46e-15    3.34e-15
order_hour_of_day_1[T.True]    8.92e-15   2.85e-15      3.128      0.002
3.33e-15    1.45e-14
order_hour_of_day_2[T.True]   6.132e-15   3.39e-15      1.808      0.071   -
5.17e-16    1.28e-14
order_hour_of_day_3[T.True]   1.728e-14   3.82e-15      4.519      0.000
9.79e-15    2.48e-14
order_hour_of_day_4[T.True]  -5.503e-16   3.86e-15     -0.143      0.887   -
8.11e-15    7.01e-15
order_hour_of_day_5[T.True]  -1.489e-14   3.08e-15     -4.839      0.000   -
2.09e-14   -8.86e-15
order_hour_of_day_6[T.True]  -9.428e-15   2.31e-15     -4.079      0.000    -
1.4e-14    -4.9e-15
order_hour_of_day_7[T.True]   2.537e-16   1.95e-15      0.130      0.896   -
3.57e-15    4.07e-15
order_hour_of_day_8[T.True]   1.126e-15   1.86e-15      0.607      0.544   -
2.51e-15    4.76e-15
order_hour_of_day_9[T.True]   4.348e-15   1.82e-15      2.386      0.017
7.77e-16    7.92e-15
```

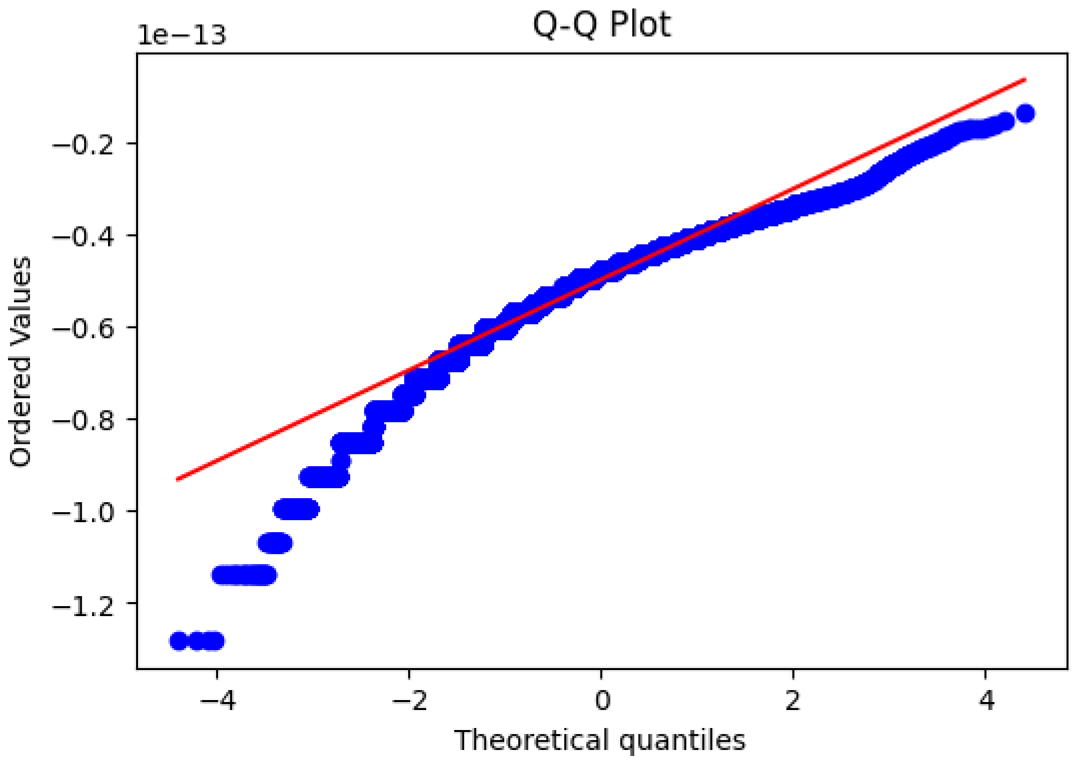
```
order_hour_of_day_10[T.True]  5.068e-15   1.81e-15      2.799      0.005
1.52e-15    8.62e-15
order_hour_of_day_11[T.True]  2.727e-16   1.81e-15      0.151      0.880   -
3.28e-15    3.82e-15
order_hour_of_day_12[T.True] -1.052e-15   1.81e-15     -0.581      0.561    -
4.6e-15     2.5e-15
order_hour_of_day_13[T.True]  2.381e-15   1.81e-15      1.316      0.188   -
1.17e-15    5.93e-15
order_hour_of_day_14[T.True]   4.43e-15   1.81e-15      2.451      0.014
8.88e-16    7.97e-15
order_hour_of_day_15[T.True]  4.515e-15   1.81e-15      2.498      0.012
9.73e-16    8.06e-15
order_hour_of_day_16[T.True]  2.892e-15   1.81e-15      1.597      0.110   -
6.58e-16    6.44e-15
order_hour_of_day_17[T.True]  2.388e-15   1.82e-15      1.313      0.189   -
1.18e-15    5.95e-15
order_hour_of_day_18[T.True]  2.634e-15   1.84e-15      1.434      0.152   -
9.67e-16    6.24e-15
order_hour_of_day_19[T.True]  1.996e-15   1.87e-15      1.069      0.285   -
1.66e-15    5.66e-15
order_hour_of_day_20[T.True]  2.534e-15   1.92e-15      1.320      0.187   -
1.23e-15     6.3e-15
order_hour_of_day_21[T.True]  3.701e-15   1.97e-15      1.878      0.060   -
1.61e-16    7.56e-15
order_hour_of_day_22[T.True]   3.45e-15   2.02e-15      1.706      0.088   -
5.15e-16    7.42e-15
order_hour_of_day_23[T.True]  1.823e-16   2.17e-15      0.084      0.933   -
4.08e-15    4.44e-15
order_number                  9.994e-17   9.18e-18     10.886      0.000
8.19e-17    1.18e-16
days_since_prior_order        7.107e-17   1.43e-17      4.967      0.000
4.3e-17    9.91e-17
user_avg_order_size              1.0000   1.78e-17   5.62e+16      0.000
1.000       1.000
=============================================================================
=
Omnibus:                    20819.790   Durbin-Watson:                   0.07
8
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            42051.92
1
Skew:                          -0.970   Prob(JB):                         0.0
0
Kurtosis:                       4.983   Cond. No.                     1.76e+0
3
=============================================================================
=

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correc
tly specified.
[2] The condition number is large, 1.76e+03. This might indicate that there a
re
strong multicollinearity or other numerical problems.
```

In [59]:
```
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
import statsmodels.formula.api as smf
import pandas as pd
from scipy.stats import probplot

# model = smf.ols(formula='order_size ~ order_number+order_dow+order_hour_of_d
model = smf.ols(formula=formula, data=orders_with_size_dummy_df).fit()

fitted_values = model.fittedvalues  # Fitted values
residuals = model.resid             # Residuals
standardized_residuals = (residuals - np.mean(residuals)) / np.std(residuals)
```

In [60]:
```
# Residuals vs Fitted Plot
plt.figure(figsize=(6, 4))
plt.scatter(fitted_values, residuals)
#plt.axhline(0, color='red', linestyle='--')
plt.title('Residuals vs Fitted')
plt.xlabel('Fitted values')
plt.ylabel('Residuals')
#plt.savefig(r"C:\Users\richa\COMP 541 Data Mining\Residuals fitted plot.png")
plt.show()
```



In [61]:
```
plt.figure(figsize=(6, 4))
probplot(residuals, dist="norm", plot=plt)
plt.title('Q-Q Plot')
#plt.savefig(r"C:\Users\richa\COMP 541 Data Mining\Project\QQ plot.png")
plt.show()
```



In [62]:
```
from sklearn.linear_model import LinearRegression

X = orders_with_size_df.drop(columns=["order_size", "eval_set"])
y = orders_with_size_df.loc[:, "order_size"]
```

In [63]:
```
model = LinearRegression().fit(X, y)
```

In [64]:
```
predictions = model.predict(X)
```

In [65]:
```
from sklearn.metrics import r2_score

r2 = r2_score(y, predictions)
print("R²:", r2)
```

```
R²: 1.0
```

In [66]:
```
from sklearn.metrics import mean_squared_error
import numpy as np

rmse = np.sqrt(mean_squared_error(y, predictions))
print("RMSE:", rmse)
```

```
RMSE: 2.8832904537953726e-14
```

Note: Potential overfitting is concern, but we need more data to investigate on this. Overall, we think this is fairly a good result with this RMSE score.

Also, the data might not be linearly in nature, as the diagostic plots show some concern of violating linear regression assumption such as Heteroscedasticity

Forecast product demand over time for inventory optimization.

In [67]:
```
product_demand = orders_df.groupby('order_dow')['order_id'].count().reset_inde
```

In [68]:
```
product_demand
```

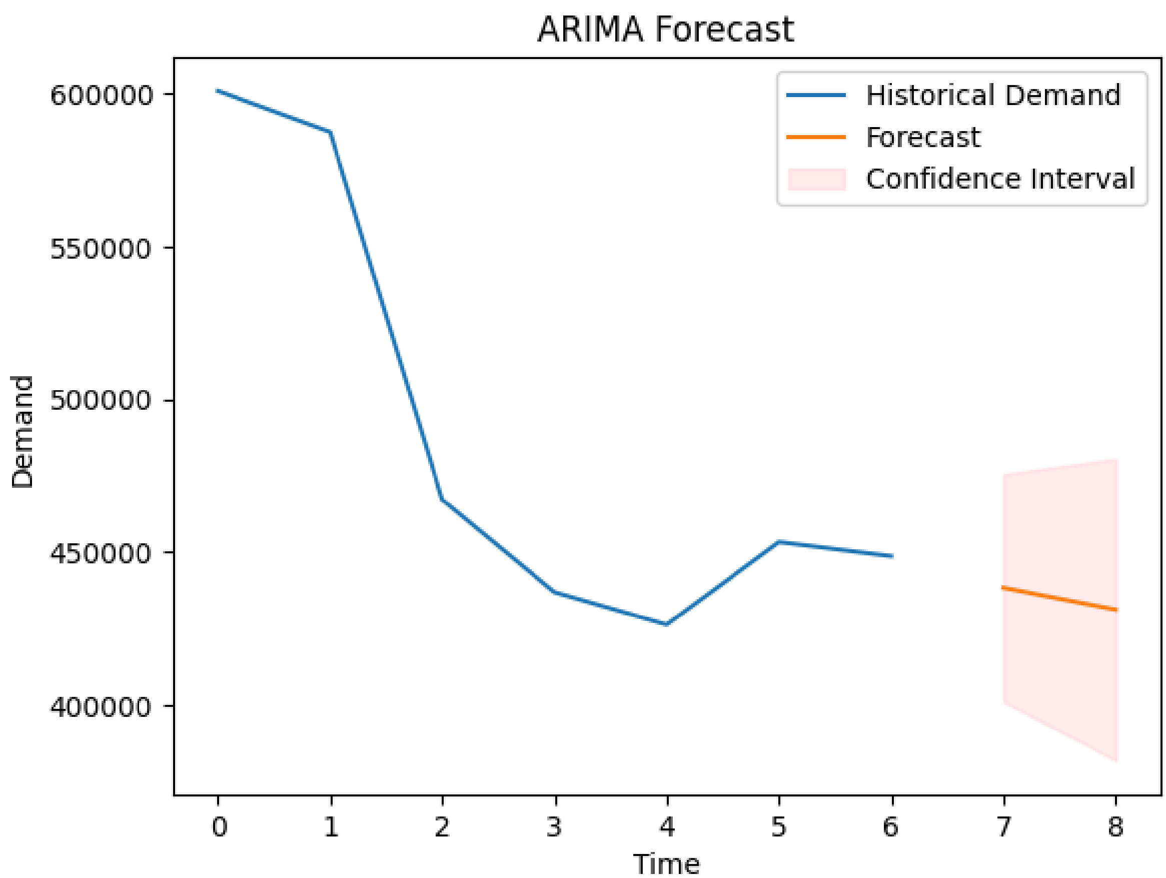
Out[68]:

	order_dow	order_id
0	0	600905
1	1	587478
2	2	467260
3	3	436972
4	4	426339
5	5	453368
6	6	448761

In [69]:
```
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_absolute_error

# Aggregate data
product_demand = orders_df.groupby('order_dow')['order_id'].count().reset_inde

test_size = 2  # Use the last 4 data points as test set
train_data = product_demand[:-test_size]
test_data = product_demand[-test_size:]

# Fit ARIMA model
model = ARIMA(product_demand['order_id'], order=(1, 1, 1))
model_fit = model.fit()

forecast = model_fit.get_forecast(steps=test_size)
forecast_values = forecast.predicted_mean
forecast_conf_int = forecast.conf_int()

actual = test_data['order_id'].values  # Actual values from the test set
mae = mean_absolute_error(actual, forecast_values)
rmse = np.sqrt(mean_squared_error(actual, forecast_values))

# Results
print("Forecasted Values:")
print(forecast_values)

print("\nActual Values:")
print(actual)

print("\nConfidence Intervals:")
print(forecast_conf_int)

print(f"\nEvaluation Metrics: MAE = {mae}, RMSE = {rmse}")
```

```
Forecasted Values:
7    438412.83337
8    431275.73847
Name: predicted_mean, dtype: float64

Actual Values:
[453368 448761]

Confidence Intervals:
   lower order_id  upper order_id
7   401216.858481   475608.808258
8   381937.349880   480614.127060

Evaluation Metrics: MAE = 16220.21408004794, RMSE = 16269.471099370132
```

In [70]:
```
import matplotlib.pyplot as plt

plt.plot(product_demand['order_id'], label="Historical Demand")
plt.plot(range(len(product_demand), len(product_demand) + len(forecast_values)
plt.fill_between(
    range(len(product_demand), len(product_demand) + len(forecast_values)),
    forecast_conf_int.iloc[:, 0],
    forecast_conf_int.iloc[:, 1],
    color='pink',
    alpha=0.3,
    label="Confidence Interval"
)
plt.legend()
plt.xlabel("Time")
plt.ylabel("Demand")
plt.title("ARIMA Forecast")
#plt.savefig(r"ARIMA Forecast")
plt.show()
```



In [71]:
```python
import numpy as np

# Actual and Forecasted values
actual = np.array([453368, 448761]) #Because we use the last two rows to test
forecasted = np.array([438412.83, 431275.74])

# Calculate MAPE
mape = np.mean(np.abs((actual - forecasted) / actual)) * 100
print(f"MAPE: {mape:.2f}%")
```

```
MAPE: 3.60%
```

Note: We don't have enough data to improve our score, but as far as we can do, the graph is off about 3.60%. We also want to do the hyperparameter tuning and grid search to optimize the model, but we don't have enough data to train on.

In summary:

This is a fair step to get a preliminary result