Individual Assignment 2

AI1030 - Python Programming

Rui Gao

November 12, 2025

Abstract

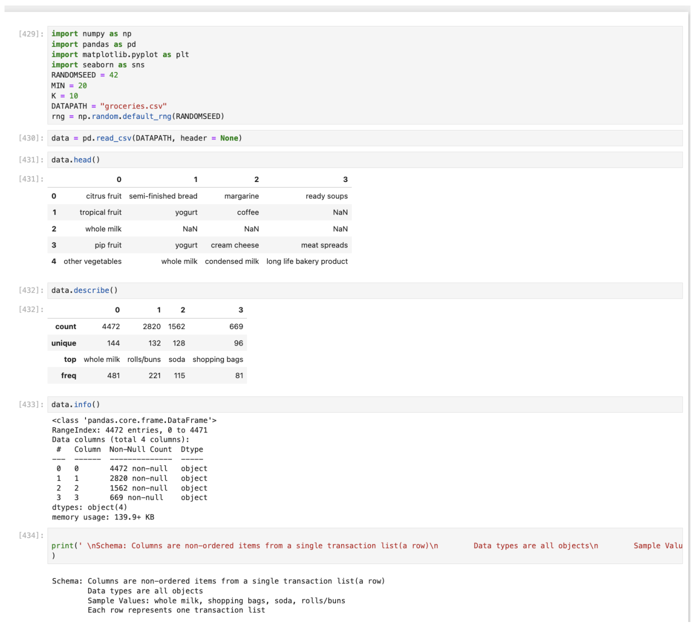
This report presents an end-to-end analysis of a grocery basket using real-world data. This project required us to utilize numerous Python libraries and familiarize ourselves with the Python environment. Although this simple project focuses only on basic data processing and exploration, it lays a huge foundation for further studies, as dealing with data is always the most challenging part in most machine learning projects. As a future AI leader, I have recognized the importance of becoming familiar with these libraries to facilitate future studies and projects. With these in mind, I completed this assignment effectively and learned many new skills. My code and results are released at https://github.com/raysuton/python_individual_assignment_2

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 significant mark deduction.

1 Task Description

The objective of this project is to conduct a comprehensive grocery basket analysis, infer a schema, and gain an understanding of the data. Personally speaking, although I am not familiar with OOP or general programming projects, I have had a decent amount of previous experience on machine learning projects, which has inevitably practiced my skills of data cleaning, processing, and visualization. As a result, this project was significantly easier for me compared to the previous ones that required a more rigorous understanding of project-based programming, or in other words, OOP.

This report will be divided into roughly five parts, abiding by the project requirements:



```
[429]: import numpy as np
       import pandas as pd
       import matplotlib.pyplot as plt
       import seaborn as sns
       RANDOMSEED = 42
       MIN = 20
       K = 10
       DATAPATH = "groceries.csv"
       rng = np.random.default_rng(RANDOMSEED)
```

```
[430]: data = pd.read_csv(DATAPATH, header = None)
```

```
[431]: data.head()
```

[431]:

	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

```
[432]: data.describe()
```

[432]:

	0	1	2	3
count	4472	2820	1562	669
unique	144	132	128	96
top	whole milk	rolls/buns	soda	shopping bags
freq	481	221	115	81

```
[433]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4472 entries, 0 to 4471
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   0       4472 non-null   object
 1   1       2820 non-null   object
 2   2       1562 non-null   object
 3   3       669 non-null    object
dtypes: object(4)
memory usage: 139.9+ KB
```

```
[434]: 
print(' \nSchema: Columns are non-ordered items from a single transaction list(a row)\n        Data types are all objects\n        Sample Valu
)

Schema: Columns are non-ordered items from a single transaction list(a row)
        Data types are all objects
        Sample Values: whole milk, shopping bags, soda, rolls/buns
        Each row represents one transaction list
```

Figure 1: Loading Data and Basic Info

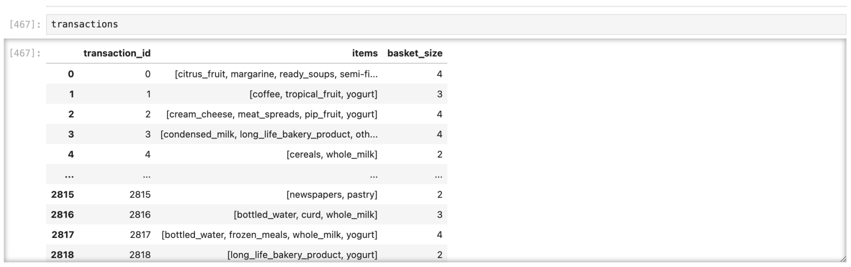
1. `Design & Data Model`: Loading, understanding the data, inferring a schema, and briefly exploring this data to extract important information or patterns.

2. `Methodology`: Cleaning useless values, standardizing item names, and arranging data into ways of highest readability.

3. `Findings`: Presenting key results from the exploratory data analysis and co-occurrence statistics.

4. `Key Figures & Data Visualization`: Visualizing relevant data and co-occurrence statistics based on calculations to provide a detailed and intuitive display of the dataset.

5. `Limitation & Next Steps`: Discussing data limitations, threshold sensitivity, and potential extensions such as deeper confidence/lift analysis and the refinement of revenue scenario simulations.



```
[467]: transactions
```

```
[467]:       transaction_id                                              items  basket_size
         0                0     [citrus_fruit, margarine, ready_soups, semi-fi...            4
         1                1                   [coffee, tropical_fruit, yogurt]            3
         2                2     [cream_cheese, meat_spreads, pip_fruit, yogurt]            4
         3                3   [condensed_milk, long_life_bakery_product, oth...            4
         4                4                             [cereals, whole_milk]            2
       ...              ...                                                ...          ...
      2815             2815                             [newspapers, pastry]            2
      2816             2816                  [bottled_water, curd, whole_milk]            3
      2817             2817   [bottled_water, frozen_meals, whole_milk, yogurt]            4
      2818             2818              [long_life_bakery_product, yogurt]            2
```

Figure 2: Canonical Transaction Schema

2 Design & Data Model

2.1 Raw schema

As Figure 1 shows, the first step of the project involves loading the data set and extracting basic information using .head(), .info(), and .describe(), as illustrated in the previous figure. I concluded that within this dataset, the columns represent non-ordered items from a single transaction list (a row), and the data types are all objects. Some sample values include whole milk, shopping bags, soda, and rolls/buns. Additionally, each row represents a single transaction list.

2.2 Canonical schema

After basic data processing, including name standardization, column concatenation, counting item numbers, and labeling, I arrived at this canonical transaction schema, which is stored within the transaction dataframe. Figure 2 is a picture of the *transactions* dataframe.

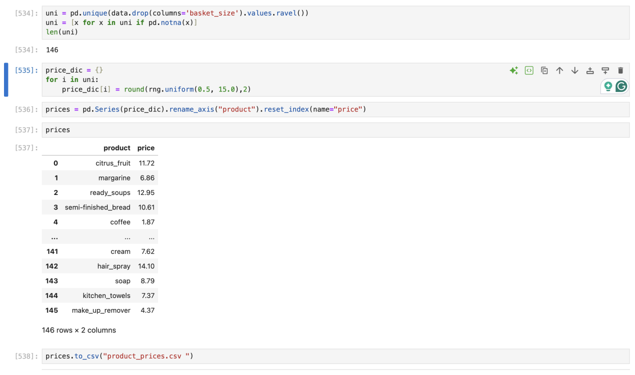
Column	**Type**	**Meaning**
transaction_id	int	Unique identifier for a single transaction (basket).
basket_size	int	Number of items contained in the corresponding basket.
Items	List	List of items in the current basket.

Table 1: Canonical Transaction Schema

2.3 Auxiliary structures

2.3.1 Price Map

From Figure 3, after extracting unique items, I assigned a distinct price to each item randomly using the random seed 42 and stored each value within a price map for further use. The price map could be accessed and viewed in the product_prices.csv



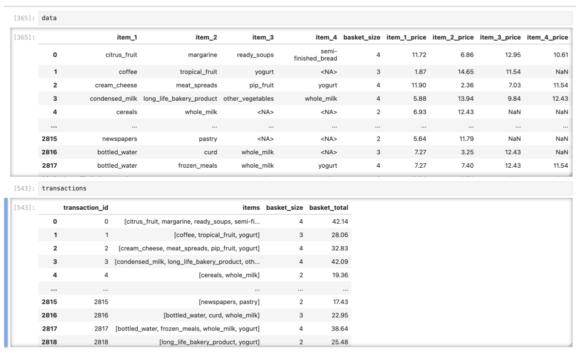
```
[534]: uni = pd.unique(data.drop(columns='basket_size').values.ravel())
       uni = [x for x in uni if pd.notna(x)]
       len(uni)

[534]: 146

[535]: price_dic = {}
       for i in uni:
           price_dic[i] = round(rng.uniform(0.5, 15.0),2)

[536]: prices = pd.Series(price_dic).rename_axis("product").reset_index(name="price")

[537]: prices

[537]:
```

	product	price
0	citrus_fruit	11.72
1	margarine	6.86
2	ready_soups	12.95
3	semi-finished_bread	10.61
4	coffee	1.87
...	...	...
141	cream	7.62
142	hair_spray	14.10
143	soap	8.79
144	kitchen_towels	7.37
145	make_up_remover	4.37

146 rows × 2 columns

```
[538]: prices.to_csv("product_prices.csv ")
```

Figure 3: Price Map



```
[365]: data

[365]:
```

	item_1	item_2	item_3	item_4	basket_size	item_1_price	item_2_price	item_3_price	item_4_price
0	citrus_fruit	margarine	ready_soups	semi-finished_bread	4	11.72	6.86	12.95	10.61
1	coffee	tropical_fruit	yogurt	<NA>	3	1.87	14.65	11.54	NaN
2	cream_cheese	meat_spreads	pip_fruit	yogurt	4	11.90	2.36	7.03	11.54
3	condensed_milk	long_life_bakery_product	other_vegetables	whole_milk	4	5.88	13.94	9.84	12.43
4	cereals	whole_milk	<NA>	<NA>	2	6.93	12.43	NaN	NaN
...	...	...	...	...	...	...	...	...	...
2815	newspapers	pastry	<NA>	<NA>	2	5.64	11.79	NaN	NaN
2816	bottled_water	curd	whole_milk	<NA>	3	7.27	3.25	12.43	NaN
2817	bottled_water	frozen_meals	whole_milk	yogurt	4	7.27	7.40	12.43	11.54

```
[543]: transactions

[543]:
```

	transaction_id	items	basket_size	basket_total
0	0	[citrus_fruit, margarine, ready_soups, semi-fi...	4	42.14
1	1	[coffee, tropical_fruit, yogurt]	3	28.06
2	2	[cream_cheese, meat_spreads, pip_fruit, yogurt]	4	32.83
3	3	[condensed_milk, long_life_bakery_product, oth...	4	42.09
4	4	[cereals, whole_milk]	2	19.36
...	...	...	...	...
2815	2815	[newspapers, pastry]	2	17.43
2816	2816	[bottled_water, curd, whole_milk]	3	22.95
2817	2817	[bottled_water, frozen_meals, whole_milk, yogurt]	4	38.64
2818	2818	[long_life_bakery_product, yogurt]	2	25.48

Figure 4: Data Dump

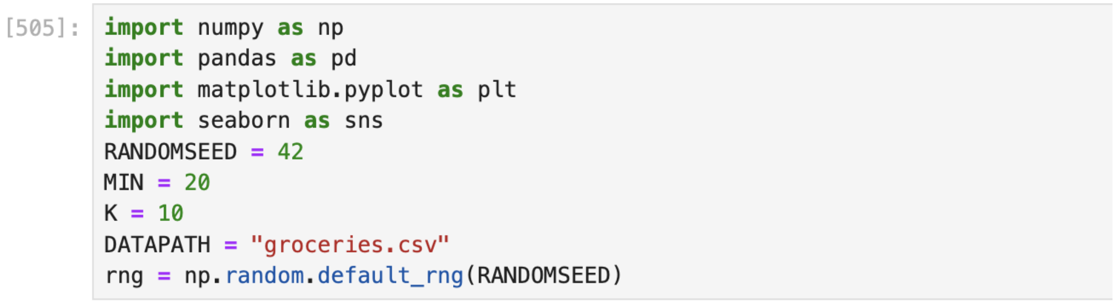
2.3.2 Data Dump

As Figure 4 illustrates, I created two distinct data frames to store the data. The first one is *transactions*, which is the dataframe that only contains the required information stated in the assignment brief. On the other hand, the *data* dataframe stores every modification and action I have made and will make to the dataframe, so I can easily access every value. Think about it this way, *transactions* is facing the public, a front-end result, whereas *data* is facing only me, a private, back-end data log.

3 Methodology

3.1 Cleaning Rules

This is how raw baskets are processed and transformed into the canonical schema. The following text would be an explanation of Figure 5.



```
[505]: import numpy as np
       import pandas as pd
       import matplotlib.pyplot as plt
       import seaborn as sns
       RANDOMSEED = 42
       MIN = 20
       K = 10
       DATAPATH = "groceries.csv"
       rng = np.random.default_rng(RANDOMSEED)
```

Figure 5: Parameter Control

- **Standardization**: I converted all item names to lowercase, stripped spaces, and replaced all spaces with underscores _. I also ensured that empty values are not represented in the final schema. In addition, I normalized the item columns, concatenating the columns['item_1'] to ['item_4'] into one single item column containing lists of all baskets
- **Invalid Removal**: I dropped rows with invalid baskets (null baskets that contain no values and duplicated baskets), removed invalid placeholders like ¡NA¿ and NaN, and removed transactions containing non-string tokens or obviously miscoded fields.
- **Basket Size Filters**: I filtered out baskets with fewer than 2 basket sizes to make the following process of calculating pairs and triples easier and valid

3.2 Parameter Choices

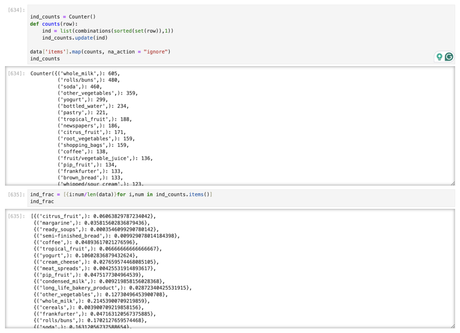
The main parameter choice lies in three main parts: MIN, K, and RANDOMSEED, which can be seen in the following table and Figure 5

Parameter	**Symbol**	**Value**	**Purpose**
Selection threshold	`K`	10	number of pairs selected
Minimum filter	`MIN`	20	smallest subset for inspection
Random seed	`RANDOMSEED`	42	ensures randomness control

Table 2: Key Parameter Choices

3.3 Performance Decisions

Performance Decisions are practical choices that I made to keep analysis efficient and stable.



```
[634]:
ind_counts = Counter()
def counts(row):
    ind = list(combinations(sorted(set(row)),1))
    ind_counts.update(ind)

data['items'].map(counts, na_action = "ignore")
ind_counts
```

```
[634]: Counter({('whole_milk',): 605,
         ('rolls/buns',): 480,
         ('soda',): 460,
         ('other_vegetables',): 359,
         ('yogurt',): 299,
         ('bottled_water',): 234,
         ('pastry',): 221,
         ('tropical_fruit',): 188,
         ('newspapers',): 186,
         ('citrus_fruit',): 171,
         ('root_vegetables',): 159,
         ('shopping_bags',): 159,
         ('coffee',): 138,
         ('fruit/vegetable_juice',): 136,
         ('pip_fruit',): 134,
         ('frankfurter',): 133,
         ('brown_bread',): 133,
         ('whipped/sour_cream',): 123,
```

```
[635]: ind_frac = [{i:num/len(data)}for i,num in ind_counts.items()]
ind_frac
```

```
[635]: [{('citrus_fruit',): 0.06063829787234042},
 {('margarine',): 0.035815602836879436},
 {('ready_soups',): 0.0003546099290780142},
 {('semi-finished_bread',): 0.009929078014184398},
 {('coffee',): 0.04893617021276596},
 {('tropical_fruit',): 0.06666666666666667},
 {('yogurt',): 0.10602836879432624},
 {('cream_cheese',): 0.027659574468085105},
 {('meat_spreads',): 0.00425531914893617},
 {('pip_fruit',): 0.0475177304964539},
 {('condensed_milk',): 0.009219858156028368},
 {('long_life_bakery_product',): 0.02872340425531915},
 {('other_vegetables',): 0.12730496453900708},
 {('whole_milk',): 0.21453900709219859},
 {('cereals',): 0.003900709219858156},
 {('frankfurter',): 0.047163120567375885},
 {('rolls/buns',): 0.1702127659574468},
 {('soda',): 0.16312056737588654},
```

Figure 6: Individual Counts & Individual Fractions

- **Pre-aggregation**: I retained only the top 25 frequent items to reduce the matrix size from thousands of baskets.
- **Vectorization**: used NumPy/Pandas operations (sum(axis=1), div) instead of nested loops because the built-in vectorization is significantly faster
- **Symmetry handling**: I sorted and stored pairs (A,B) with A ¡ B to achieve efficient memory use.
- **Random control**:I fixed RANDOMSEED for reproducible random pricing and perturbation results.
- **Filtering**: I removed rare-item pairs before lift computation to prevent extreme values and overfitting.

4 Findings

4.1 Exploratory Data Analysis (EDA)

Exploratory data analysis can provide an overview of frequency and composition within the transaction dataset, highlighting associations and consumer preferences. Based on the ind_frac structure (a dictionary in which I stored the fraction of appearance of items), the most frequent single items were `whole_milk` (21.45%), `rolls/buns` (17.02%), `soda` (16.31%), `other_vegetables` (12.73%), and `yogurt` (10.60%).

These findings are presented in Figure 6, as all of these are daily necessities that people will purchase on a regular basis. Additionally, these items comprise a significant proportion of transactions and serve as the foundation for subsequent co-occurrence and association structures.

Note: Basket sizes ranged mainly between two and four items, making pairwise and triple-level association analysis both meaningful and computationally achievable.

4.2 Pairwise Co-occurrence and Association Metrics

I've extracted `pair_counts` and `frac_pair` using the Counter library to ensure the program's speed and efficiency. These two Counters captured the strength of two-item relationships. The most common pairs are shown below:

- `rolls/buns -- soda`: 84 baskets, support ≈ 0.0298
- `rolls/buns -- whole_milk`: 80 baskets, support ≈ 0.0284
- `other_vegetables -- whole_milk`: 75 baskets, support ≈ 0.0266
- `bottled_water -- soda`: 49 baskets, support ≈ 0.0174
- `frankfurter -- rolls/buns`: 48 baskets, support ≈ 0.0170
- `pastry -- whole_milk`: 48 baskets, support ≈ 0.0170

4.3 Confidence and Lift Analysis

Association-rule metrics were computed using:

$$\text{Confidence}(A \to B) = \frac{\text{Support}(A, B)}{\text{Support}(A)}, \quad \text{Lift}(A \to B) = \frac{\text{Confidence}(A \to B)}{\text{Support}(B)}.$$

The average confidence for frequent pairs ranged between 0.15 and 0.20, e.g.,

$$\text{Confidence}(\text{rolls/buns} \to \text{soda}) \approx 0.175, \quad \text{Confidence}(\text{soda} \to \text{rolls/buns}) \approx 0.183.$$

These moderate but stable values indicate consistent purchase behavior among popular products. Lift values were more variable; rare items with very low support (e.g., `toilet_cleaner -- jam`) exhibited artificially high lift scores due to small denominators. After applying thresholds of min_support $\geq 1\%$ and min_pair_count ≥ 5, the lift results stabilized and highlighted genuine associations such as `whole_milk -- yogurt` and `rolls/buns -- soda`.

4.4 Triple Co-occurrence Patterns

The `triples_K` structure captured three-item combinations observed most frequently. The top triples, sorted by count, are as follows:

- `other_vegetables -- root_vegetables -- whole_milk`: 8 baskets
- `other_vegetables -- soda -- whole_milk`: 7 baskets

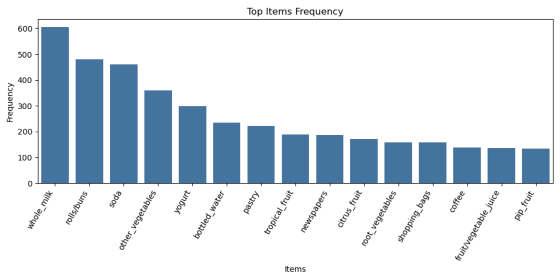


Figure 7: Top Items Frequency

- `beef -- root_vegetables -- whole_milk`: 6 baskets
- `citrus_fruit -- pip_fruit -- tropical_fruit`: 6 baskets
- `frankfurter -- rolls/buns -- soda`: 6 baskets

Although triple supports were typically below 1%, these patterns reinforce the dominance of a few staple items that repeatedly appear across all association levels. Notably, `whole_milk`, `rolls/buns`, and `other_vegetables` appear in the majority of frequent triples, further confirming their central role in the dataset's structure.

4.5 Summary of Analytical Results

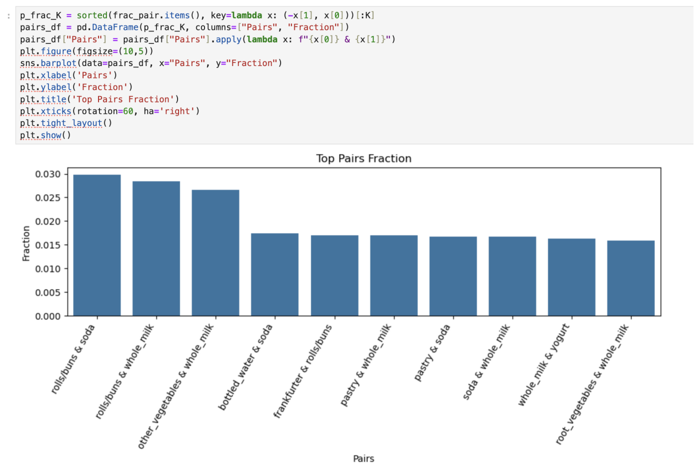
Overall, the findings derived from `ind_frac`, `frac_pair`, and the association-rule metrics can be summarized as follows:

1. A small subset of high-frequency staples (`whole_milk`, `rolls/buns`, `soda`, `other_vegetables`, `yogurt`) drives the majority of pair and triple co-occurrences.
2. Confidence values for dominant pairs are stable and meaningful, whereas lift requires filtering due to inflation from rare items.
3. Triple combinations extend the same structure seen in pairs, confirming a highly consistent, staple-driven market basket pattern.
4. These association metrics collectively indicate structured, non-random purchasing behavior centered around core grocery categories.

5 Key Figures & Data Visualization

5.1 Top Items

Figure 7 provides a rough illustration of the top 15 items with the highest frequency in the transaction basket. I believe that this result is entirely plausible, as all of these items



```
p_frac_K = sorted(frac_pair.items(), key=lambda x: (-x[1], x[0]))[:K]
pairs_df = pd.DataFrame(p_frac_K, columns=["Pairs", "Fraction"])
pairs_df["Pairs"] = pairs_df["Pairs"].apply(lambda x: f"{x[0]} & {x[1]}")
plt.figure(figsize=(10,5))
sns.barplot(data=pairs_df, x="Pairs", y="Fraction")
plt.xlabel('Pairs')
plt.ylabel('Fraction')
plt.title('Top Pairs Fraction')
plt.xticks(rotation=60, ha='right')
plt.tight_layout()
plt.show()
```

Figure 8: Top Pairs by Fraction

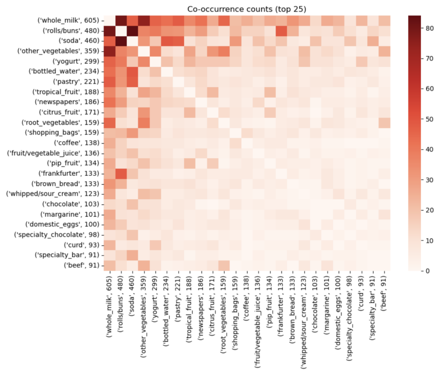


Figure 9: Co-Occurrence Heat Map

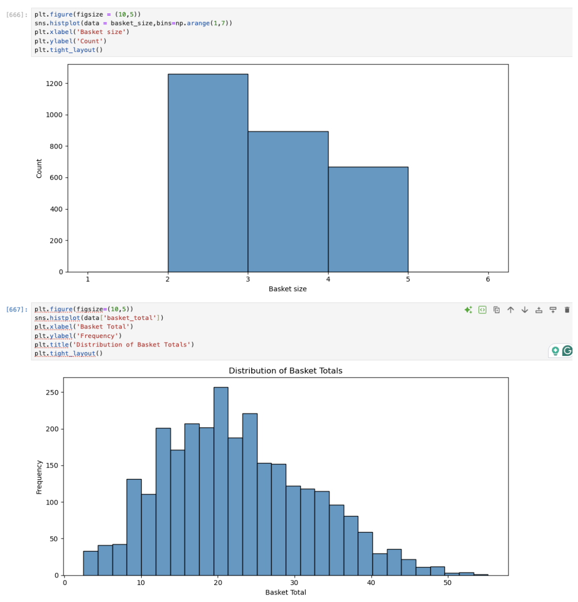
are used and consumed on a daily basis.

5.2 Top K Pairs by Support Fraction

Figure 8 illustrates the top pairs by support fraction. As demonstrated in the previous sections, these pairs represented the core determinants of the most common values in the dataset, including whole milk, rolls/buns, and vegetables, among others.

5.3 Co-Occurrence Heat Map

From Figure 9, I can infer that the darkest areas cluster around the key items we pointed out earlier: whole milk, rolls/buns, soda, other vegetables, and yogurt. These items not



```
[666]: plt.figure(figsize = (10,5))
       sns.histplot(data = basket_size,bins=np.arange(1,7))
       plt.xlabel('Basket size')
       plt.ylabel('Count')
       plt.tight_layout()
```

```
[667]: plt.figure(figsize=(10,5))
       sns.histplot(data['basket_total'])
       plt.xlabel('Basket Total')
       plt.ylabel('Frequency')
       plt.title('Distribution of Basket Totals')
       plt.tight_layout()
```

Figure 10: Basket Bar Charts

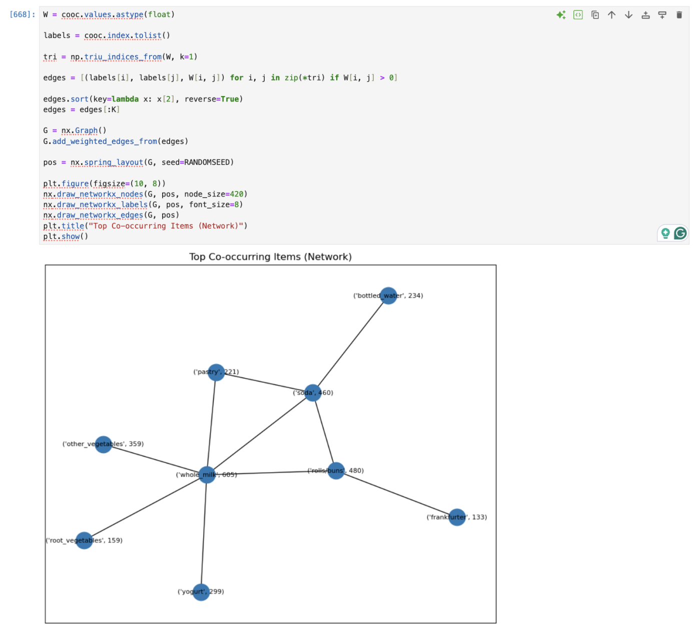
only dominated the pair and triple counts, but also the most transactions and the most appeared together. Whole milk stands out as the central product, co-occurring with nearly all other high-frequency items, hitting the count of 620. The rest of the matrix is lighter, showing that other items are rarely bought together.

5.4 Basket Bar Charts

Figure 10 clearly displayed that the most transactions contain relatively few items, and basket total follows a moderately a moderately right-skewed pattern. The first plot shows that the basket size 2 is the most common, which is plausible since pairs are dominating baskets. This indicates that customers typically make small, focused purchases rather than large stock-up trips. The second plot, which displays the distribution of basket totals, peaks around the mid-range and then gradually tapers off, suggesting that while most customers spend a moderate amount, a smaller number of transactions account for much higher totals

5.5 Top Pairs Network Graph

This step is the visualization of the optional network graphs. I extracted the upper triangular of the co-occurence matrix to avoid duplicates, keeping only pairs with positive counts. Then, through sorting these pairs by their co-occurrence strength, I selected the top K pairs for simplicity and clarity. Additionally, when I draw the graph, I applied a



```
[668]: W = cooc.values.astype(float)

       labels = cooc.index.tolist()

       tri = np.triu_indices_from(W, k=1)

       edges = [(labels[i], labels[j], W[i, j]) for i, j in zip(*tri) if W[i, j] > 0]

       edges.sort(key=lambda x: x[2], reverse=True)
       edges = edges[:K]

       G = nx.Graph()
       G.add_weighted_edges_from(edges)

       pos = nx.spring_layout(G, seed=RANDOMSEED)

       plt.figure(figsize=(10, 8))
       nx.draw_networkx_nodes(G, pos, node_size=420)
       nx.draw_networkx_labels(G, pos, font_size=8)
       nx.draw_networkx_edges(G, pos)
       plt.title("Top Co-occurring Items (Network)")
       plt.show()
```

Figure 11: Network Graph

spring layout with a fixed random seed for consistent positioning and plotted the graph. Figure 9 is the completed network graph

6 Limitations & Next Steps

Even though this project worked pretty well overall, there are still some limits in the data and the way I did the analysis. The dataset doesn't have any time or customer information, so I can't really see how shopping habits change over time or between different people. Also, the prices were random values that I made up, which means the revenue results are only for demo and not really accurate. Some items are also not grouped into categories, so it's hard to see bigger trends like "dairy" or "bakery." Last but not least, Most baskets are pretty small (2–4 items), adding bigger item groups like 4 or 5 items together would be very helpful

The results also depend a lot on the thresholds I used, like the min support or top K pairs. If I use too high values, I lose small but maybe interesting patterns. If I use too low values, there's a lot of noise and random item pairs show up with big lift numbers.

For next steps, I could try to:

- check the confidence and lift values more carefully with different settings;
- run some bootstrap tests to see if the top pairs are stable or just random;

- analyze data over time or by groups of customers (if I had that info);
- and maybe use real price data for better revenue simulation.

In short, this project gave a good start for understanding grocery data, but there's still a lot to improve if I had more detailed or real data next time.