**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ТАРАСА ШЕВЧЕНКА**

Факультет комп’ютерних наук та кібернетики

Кафедра теорії та технології програмування

Звіт

до лабораторної роботи №1 на тему

«Market Basket Analysis»

Виконав студент 4-го курсу

Групи ТТП-42

Ходаков Максим Олегович

**Київ – 2025**

**Анотація**

В цій лабораторній роботі було реалізовано:

* **алгоритм Apriori** для пошуку частих наборів товарів (Frequent Itemsets) та генерації **правил асоціацій** (Market Basket Analysis);
* **візуалізацію** знайдених найчастіших наборів та правил у вигляді діаграм і мережевих графів.

Для реалізації використовувалась мова програмування **Python**, яка має розвинуту екосистему бібліотек для обробки даних, візуалізації та машинного навчання.

Було використано такі бібліотеки:

* **pandas** – універсальна бібліотека для роботи з табличними даними, що надає зручні структури (DataFrame) та операції для маніпуляцій із даними;
* **mlxtend** – містить реалізацію алгоритму **Apriori** і методів для побудови правил асоціацій (association\_rules);
* **seaborn** і **matplotlib** – для побудови двовимірних діаграм та графіків різного типу;
* **networkx** – для побудови й візуалізації мережевих структур (спрямованих графів правил).

Таким чином, у межах лабораторної роботи було показано, як виявляти приховані закономірності у даних про покупки, перетворювати їх у зручний формат для аналізу, а також будувати й інтерпретувати результати у вигляді графічних візуалізацій.

**Вступ**

**Мета**

Виявити та дослідити асоціації й кореляції між різними товарами, які клієнти найчастіше купують разом у супермаркетах чи на онлайн-платформах.

**Ключова концепція**

* **Видобуток асоціативних правил** (Association Rule Mining)
* **Алгоритм Apriori** для пошуку частих наборів товарів

**Інструменти та технології**

* **Python**
  + **pandas**: для маніпулювання даними
  + **mlxtend**: для реалізації алгоритму Apriori та генерування правил асоціацій
  + **matplotlib** або **seaborn**: для візуалізації результатів
* **Набір даних**: стандартний набір Groceries, який містить перелік транзакцій із переліком придбаних товарів

**Розподіл завдань**

1. **Попередня обробка даних**
   * **Завантаження даних**: Зчитати вихідний CSV (або інший формат) та підготувати його для аналізу.
   * **Очищення даних**: Усунути або обробити відсутні значення.
   * **Перетворення даних**: Виконати перетворення у формат, придатний для Apriori (наприклад, однократне кодування / one-hot encoding).
2. **Попередній аналіз даних (EDA)**
   * **Визначення найпоширеніших товарів** і (за потреби) найпоширеніших наборів товарів.
   * **Візуалізація** частоти появи топ-елементів або топ-наборів елементів за допомогою гістограм тощо.
3. **Реалізація алгоритму Apriori**
   * **Налаштування параметрів**: Встановити min\_support, min\_confidence та (за потреби) lift.
   * **Генерація частих наборів елементів**: Знайти Frequent Itemsets за допомогою apriori.
   * **Генерація правил**: Створити правила асоціацій (association rules) з цих частих наборів.
4. **Аналіз результатів**
   * **Інтерпретація**: Пояснити, що означають отримані правила (наприклад, {хліб, масло} -> {молоко}).
   * **Фільтрація правил**: Вибрати лише найзначущі правила за допомогою таких метрик, як confidence та lift.
5. **Візуалізація**
   * **Побудова діаграм**: відобразити підтримку (support) і частоту появи топ-наборів елементів.
   * **Мережеві графіки** (за допомогою networkx або інших інструментів) для демонстрації правил асоціацій.

**Опис алгоритму Apriori**

Алгоритм Apriori є популярним методом для виявлення частих наборів елементів у великих базах даних. Він використовує принцип "Apriori Property", який стверджує, що якщо певний набір елементів є частим, то всі його піднабори також є частими.

Послідовність дій в алгоритмі:

* Визначаємо мінімальне значення підтримки (support).
* Знаходимо множину елементів, які задовольняють мінімальну підтримку.
* Формуємо множини з двох елементів на основі попередньої множини та відкидаємо ті, що не задовольняють мінімального значення підтримки.
* Повторюємо попередній крок, додаючи за кожною ітерацією по одному елементу в множину, до тих пір, поки не буде отримана множина, що не задовольняє мінімального значення підтримки.

Протягом процесу, алгоритм використовує такі метрики:

Support (Підтримка). Знаходить рівень частоти входження елементу. Формула:

*Support (X) = (Transactions containing (X)) / (Total Transactions)*

Confidence (Упевненість). Міра того, наскільки часто умова правила вірна. Формула:

*Confidence (X -> Y) = (Transactions containing both (X and Y)) / (Transactions containing X)*

 Lift (Піднесення). Показує рівень частоти входження (Y) при умові, що товар (X) куплено.

Формула:

*Lift (X -> Y) = Confidence (X -> Y)/ Support (Y)*

**Опис роботи та код програми**

## 1. Попередня обробка даних

1. **Завантаження даних**
   * У коді використовується:

df = pd.read\_csv('Groceries\_dataset.csv') # файл із даними

df.dropna(how='all', inplace=True) # видалення цілком порожніх рядків

* + Ці рядки відповідають за:
    - **Читання CSV-файлу** зі стовпцями Member\_number, Date, itemDescription.
    - **Видалення** абсолютно порожніх рядків, якщо вони існують у датасеті.
  + Переконайтеся, що **вказано правильний шлях** до файлу (або ім’я файлу).

1. **Очищення даних**
   * Якщо б були інші пропущені дані (NaN) чи дублікати, можна застосовувати:

df.drop\_duplicates(inplace=True) # за потреби видалити дублікати

* + В коді, крім dropna(how='all', ...), інших операцій на відсутні дані немає, що припускає мінімальні проблеми з цим датасетом.

1. **Перетворення даних**
   * Головна ідея – **згрупувати** товари, які належать до однієї транзакції.
   * У нашому випадку транзакція визначається (Member\_number, Date).

grouped\_df = df.groupby(["Member\_number", "Date"])["itemDescription"].apply(list).reset\_index()

transactions = grouped\_df["itemDescription"].tolist()

* + **One-Hot кодування** потрібне алгоритму Apriori. Застосовується через:

te = TransactionEncoder()

te\_array = te.fit(transactions).transform(transactions)

encoded\_df = pd.DataFrame(te\_array, columns=te.columns\_)

* + Після цього ми маємо **таблицю** encoded\_df, де кожен рядок – це транзакція, а кожен стовпець – конкретний товар. Значення True/False вказують, чи належить товар до цієї транзакції.

## 2. Попередній аналіз даних (EDA)

1. **Визначення найпоширеніших товарів**
   * У коді виконується:

item\_counts = encoded\_df.sum().sort\_values(ascending=False)

print(item\_counts.head(10))

* + encoded\_df.sum() по кожному стовпцю підсумовує кількість True (що конвертується в 1), тому ми отримуємо **загальну частоту** появи кожного товару.
  + sort\_values(ascending=False) сортує від найбільшого до найменшого.

1. **Візуалізація частоти топ-товарів**
   * Для **топ-10** товарів будується горизонтальний бар-плот:

top\_items = item\_counts.head(10)

sns.barplot(x=top\_items.values, y=top\_items.index, orient='h', color='blue')

plt.title('Топ-10 товарів за частотою')

plt.xlabel('Частота')

plt.ylabel('Товар')

plt.show()

* + Тут ми бачимо, які товари найчастіше зустрічаються в транзакціях (наприклад, whole milk, other vegetables тощо).

A blue bar graph with white text

Description automatically generated

Рисунок 1. Графік топ товарів за частотою.

A green bar graph with numbers

Description automatically generated

Рисунок 2. Графік топ частих наборів за підтримкою.

## 3. Реалізація алгоритму Apriori

1. **Налаштування параметрів**
   * У прикладі вказано:

min\_support\_threshold = 0.001 # або інше значення

min\_confidence\_threshold = 0.1

* + min\_support\_threshold вказує на відсоток (або пропорцію) транзакцій, у яких **мінімально** повинен траплятися набір товарів, щоб вважатися “частим” (Frequent Itemset).
  + min\_confidence\_threshold — мінімальна впевненість (confidence), щоб правило вважалося достатньо сильним.

1. **Генерація частих наборів елементів**
   * Використовується функція:

frequent\_itemsets = apriori(encoded\_df, min\_support=min\_support\_threshold, use\_colnames=True)

* + Якщо use\_colnames=True, то в frequent\_itemsets колонки itemsets містять **назви товарів** замість індексів.
  + frequent\_itemsets зазвичай має стовпці: itemsets, support (і може мати інші — залежно від версії mlxtend).

1. **Генерація правил (association rules)**
   * У коді:

rules = association\_rules(

frequent\_itemsets,

num\_itemsets,

metric="confidence",

min\_threshold=min\_confidence\_threshold

)

* + metric="confidence" вказує, що ми відфільтровуємо за критерієм впевненості.
  + num\_itemsets = len(encoded\_df) потрібен у **старіших** версіях mlxtend. Якщо у вас **сучасніша** версія, цей аргумент не потрібен. У автора нажаль стара.

A screenshot of a computer

Description automatically generated

Рисунок 3. Усі згенеровані правила.

A screenshot of a computer screen

Description automatically generated

Рисунок 4. Топ-10 правил за lift >=1.

## 4. Аналіз результатів

1. **Інтерпретація**
   * Після генерації rules дивимося на такі стовпці, як:
     + **antecedents**: передумови (товари зліва від стрілки)
     + **consequents**: наслідки (товари справа)
     + **support**, **confidence**, **lift** — ключові метрики.
   * Наприклад, якщо є правило {bread} -> {butter} зі support=0.03, confidence=0.6, lift=1.2, це означає:
     + У 3% транзакцій є bread і butter разом.
     + Якщо вже куплено bread, ймовірність, що куплять butter, дорівнює 60%.
     + lift=1.2 (>1) свідчить, що це відбувається на 20% частіше, ніж очікували б при випадкових покупках.
2. **Фільтрація правил**
   * Зазвичай ми залишаємо тільки ті правила, де lift >= 1, тобто ймовірність спільної покупки вища за випадкову.
   * У коді:

filtered\_rules = rules[rules["lift"] >= 1].copy()

* + Також можна додатково фільтрувати за confidence >= 0.5 чи іншими критеріями.

## 5. Візуалізація

1. **Бар-плот частих наборів**
   * Після знаходження frequent\_itemsets робимо:

top\_10\_frequent\_itemsets = frequent\_itemsets.nlargest(10, "support").copy()

top\_10\_frequent\_itemsets["itemsets\_str"] = top\_10\_frequent\_itemsets["itemsets"].apply(lambda x: ", ".join(x))

sns.barplot(x="support", y="itemsets\_str", data=top\_10\_frequent\_itemsets, orient='h', color='green')

* + Так бачимо, які **набори** товарів (не просто одиночні товари) найчастіше зустрічаються.

1. **Мережевий граф (network graph) правил**
   * Якщо у нас є відібрані правила, будуємо nx.DiGraph(), додаємо вузли та стрілочки (edges).
   * Для кожного правила:

for \_, row in top\_rules.iterrows():

antecedents = list(row["antecedents"])

consequents = list(row["consequents"])

# додаємо зв’язки від antecedents до consequents з мітками (confidence, lift)

* + За допомогою networkx малюється спрямована дуга: **antecedents -> consequents**.

1. **Інтерпретація мережі**
   * Кожен вузол — товар.
   * Стрілка (ребро) від A до B символізує правило “A -> B”, де на ребрі може бути написано conf: 0.6, lift: 1.2.

A diagram of food items

Description automatically generated Рисунок 5. Візуалізований граф з правилами.

**Висновок**

У процесі виконання роботи було **успішно реалізовано** повний цикл аналізу ринкового кошика (Market Basket Analysis) з використанням алгоритму Apriori, що включало:

1. **Формулювання завдання та мети**:
   * Виявити асоціації й кореляції між товарами, які часто купують разом, та інтерпретувати знайдені закономірності для прийняття бізнес-рішень.
   * Забезпечити **зручну візуалізацію** результатів, аби легко зрозуміти, які товари клієнти найчастіше комбінують.
2. **Попередню обробку даних**:
   * Зчитування й очищення датасету (з відсутніми значеннями, дублікатами тощо).
   * Перетворення у формат, придатний для застосування алгоритму Apriori (лист транзакцій, one-hot кодування).
3. **Попередній аналіз даних** (EDA):
   * Визначення найбільш поширених товарів (згідно з кількістю появ у транзакціях).
   * Візуалізація топ-10 товарів за частотою покупок, що підтвердила **домінування** певних товарних категорій.
4. **Реалізацію алгоритму Apriori**:
   * Пошук **частих наборів** товарів за обраним рівнем підтримки (support).
   * Генерацію правил асоціацій (association rules) за метрикою впевненості (confidence) та оцінку цих правил за показником підйому (lift).
5. **Інтерпретацію та фільтрацію результатів**:
   * Виявлення й відсіювання нерелевантних правил з низькими показниками lift та confidence.
   * Оцінка тих правил, які мають бізнес-цінність (наприклад, “{хліб} -> {молоко}”).
6. **Візуалізацію**:
   * Створення бар-плотів та **мережевих графів** (network graphs) для наочного відображення як товарів, що часто купуються разом, так і зв’язків між ними.

Таким чином, **мети роботи було досягнуто**:

* Показано, як **застосувати алгоритм Apriori** для пошуку частих наборів та побудови правил асоціацій у ринкових даних.
* Продемонстровано **повний інструментарій** (від завантаження та чистки даних до комплексної візуалізації), який полегшує прийняття рішень щодо асортименту, мерчандайзингу й маркетингових акцій.

Отримані результати можуть бути використані для:

* **Оптимізації розташування товарів** у торговій залі.
* **Маркетингових кампаній** (рекомендації комбінацій товарів, дисконтні пропозиції).
* **Цінової політики**, коли товари, що часто купують разом, можна пакетувати чи пропонувати зі знижками.

Зрештою, виконана робота не лише **продемонструвала** можливості й гнучкість алгоритму Apriori, а й відкрила перспективу подальшого вдосконалення аналізу, зокрема шляхом:

* Застосування **більш складних алгоритмів** (FP-Growth тощо).
* Інтеграції різних метрик (наприклад, conviction, leverage) і методів візуалізації.

Отже, комплексний підхід до аналізу ринкового кошика **дозволяє краще зрозуміти** поведінку клієнтів та приймати **обґрунтовані рішення** в контексті товарного менеджменту й маркетингу.

**ДОДАТОК**

import pandas as *pd*import seaborn as *sns*import matplotlib.pyplot as *plt*import networkx as *nx*from mlxtend.preprocessing import TransactionEncoder  
from mlxtend.frequent\_patterns import apriori, association\_rules  
  
*# ----------------------------------------------------------  
# 1. Завантаження та попередня обробка даних  
# ----------------------------------------------------------  
df* = pd.read\_csv('Groceries\_dataset.csv')  
  
*# Видаляємо порожні рядки (якщо є)  
df*.dropna(how='all', inplace=True)  
  
print("Перші рядки DataFrame:")  
print(*df*.head())  
print("\nІнформація про DataFrame:")  
print(*df*.info())  
  
*# ----------------------------------------------------------  
# 2. Формування транзакцій  
# ----------------------------------------------------------  
# Групування за (Member\_number, Date), щоб зібрати всі itemDescription у списки  
grouped\_df* = *df*.groupby(["Member\_number", "Date"])["itemDescription"].apply(list).reset\_index()  
print(f"\nКількість транзакцій (унікальних (Member\_number, Date)): {len(*grouped\_df*)}")  
  
*transactions* = *grouped\_df*["itemDescription"].tolist()  
  
*# ----------------------------------------------------------  
# 3. One-Hot кодування за допомогою TransactionEncoder  
# ----------------------------------------------------------  
te* = TransactionEncoder()  
*te\_array* = *te*.fit(*transactions*).transform(*transactions*)  
*encoded\_df* = pd.DataFrame(*te\_array*, columns=*te*.columns\_)  
  
print("\nПриклад закодованих даних (перші 5 рядків):")  
print(*encoded\_df*.head())  
  
*# ----------------------------------------------------------  
# 4. Exploratory Data Analysis (EDA)  
# ----------------------------------------------------------  
item\_counts* = *encoded\_df*.sum().sort\_values(ascending=False)  
print("\nТоп-10 товарів за частотою зустрічань:")  
print(*item\_counts*.head(10))  
  
*# Візуалізація топ-10 товарів  
top\_n* = 10  
*top\_items* = *item\_counts*.head(*top\_n*)  
  
plt.figure(figsize=(8, 5))  
*# Використовуємо один колір ('blue'), щоб уникнути Deprecated Warning у Seaborn*sns.barplot(x=*top\_items*.values, y=*top\_items*.index, orient='h', color='blue')  
plt.title(f'Топ-{*top\_n*} товарів за частотою')  
plt.xlabel('Частота')  
plt.ylabel('Товар')  
plt.tight\_layout()  
plt.show()  
  
*# ----------------------------------------------------------  
# 5. Пошук частих наборів (Apriori)  
# ----------------------------------------------------------  
min\_support\_threshold* = 0.001  
  
*frequent\_itemsets* = apriori(*encoded\_df*, min\_support=*min\_support\_threshold*, use\_colnames=True)  
print("\nПриклади частих наборів (Frequent Itemsets):")  
print(*frequent\_itemsets*.head())  
  
*# ----------------------------------------------------------  
# 6. Генерація правил асоціацій з нижчою confidence  
# ----------------------------------------------------------  
min\_confidence\_threshold* = 0.1  
  
*num\_itemsets* = len(*encoded\_df*)  
  
*rules* = association\_rules(  
 *frequent\_itemsets*,  
 *num\_itemsets*,  
 metric="confidence",  
 min\_threshold=*min\_confidence\_threshold*)  
  
print("\nУсі згенеровані правила:")  
print(*rules*)  
  
*# ----------------------------------------------------------  
# 7. Аналіз та фільтрація правил  
# ----------------------------------------------------------*if *rules*.empty:  
 print("\n--- Жодне правило не знайдено за поточними параметрами ---")  
else:  
 *# Сортуємо за lift та confidence  
 rules*.sort\_values(["lift", "confidence"], ascending=False, inplace=True)  
 print("\nПриклади правил (sorted by lift & confidence):")  
 print(*rules*.head(10))  
  
 *# Додатково фільтруємо за lift >= 1  
 filtered\_rules* = *rules*[*rules*["lift"] >= 1].copy()  
  
 *# Примусово робимо lift/confidence числовими, якщо потрібно  
 filtered\_rules*["lift"] = pd.to\_numeric(*filtered\_rules*["lift"], errors="coerce")  
 *filtered\_rules*["confidence"] = pd.to\_numeric(*filtered\_rules*["confidence"], errors="coerce")  
  
 if *filtered\_rules*.empty:  
 print("\nПісля фільтрації lift >= 1, жодних правил не лишилось.")  
 *top\_rules* = pd.DataFrame()  
 else:  
 *# Обираємо 10 найкращих правил за lift  
 top\_rules* = *filtered\_rules*.nlargest(10, "lift")  
 print("\nТоп-10 правил за lift >= 1:")  
 print(*top\_rules*)  
  
 *# ----------------------------------------------------------  
 # 8. ВІЗУАЛІЗАЦІЯ РЕЗУЛЬТАТІВ  
 # ----------------------------------------------------------  
 # (A) Бар-плот топ-10 частих наборів  
 top\_10\_frequent\_itemsets* = *frequent\_itemsets*.nlargest(10, "support").copy()  
 *top\_10\_frequent\_itemsets*["itemsets\_str"] = *top\_10\_frequent\_itemsets*["itemsets"].apply(lambda *x*: ", ".join(*x*))  
  
 plt.figure(figsize=(8, 5))  
 sns.barplot(x="support", y="itemsets\_str", data=*top\_10\_frequent\_itemsets*, orient='h', color='green')  
 plt.title(f"Топ-10 частих наборів (min\_support={*min\_support\_threshold*})")  
 plt.xlabel("Підтримка")  
 plt.ylabel("Набір товарів")  
 plt.tight\_layout()  
 plt.show()  
  
 *# (B) Побудова графу правил (Top 10 за lift), якщо вони існують* if not *top\_rules*.empty:  
 *G* = nx.DiGraph()  
 for *\_*, *row* in *top\_rules*.iterrows():  
 *antecedents* = list(*row*["antecedents"])  
 *consequents* = list(*row*["consequents"])  
 for *ant* in *antecedents*:  
 for *con* in *consequents*:  
 *G*.add\_node(*ant*, color='skyblue')  
 *G*.add\_node(*con*, color='lightgreen')  
 *label\_str* = f"conf: {*row*['confidence']:.2f}, lift: {*row*['lift']:.2f}"  
 *G*.add\_edge(*ant*, *con*, weight=*row*["lift"], label=*label\_str*)  
  
 plt.figure(figsize=(10, 6))  
 *pos* = nx.spring\_layout(*G*, k=2, seed=42)  
 *colors* = [*G*.nodes[*node*]['color'] for *node* in *G*.nodes()]  
  
 nx.draw\_networkx\_nodes(*G*, *pos*, node\_color=*colors*, node\_size=2000)  
 nx.draw\_networkx\_labels(*G*, *pos*, font\_size=10)  
 nx.draw\_networkx\_edges(*G*, *pos*, arrowstyle='->', arrowsize=20)  
  
 *edge\_labels* = nx.get\_edge\_attributes(*G*, 'label')  
 nx.draw\_networkx\_edge\_labels(*G*, *pos*, edge\_labels=*edge\_labels*, font\_size=8)  
  
 plt.title("Граф правил (Top 10 за Lift ≥ 1)")  
 plt.axis("off")  
 plt.show()  
 else:  
 print("\nНемає правил із lift >= 1 для візуалізації.")  
  
print("\n--- Аналіз ринкового кошика завершено! ---")

**Джерела**

1. **Agrawal R., Srikant R.** Fast Algorithms for Mining Association Rules. Proceedings of the 20th International Conference on Very Large Data Bases (VLDB), 1994, pp. 487–499.
2. **Han J., Kamber M.** Data Mining: Concepts and Techniques. 3rd Edition, Morgan Kaufmann, 2011.
3. **Raschka S.** MLxtend: Providing Extensions and Helper Modules to Python’s Data Analysis and Machine Learning Libraries. GitHub: <https://github.com/rasbt/mlxtend>
4. **pandas Documentation**. https://pandas.pydata.org/docs/
5. **Matplotlib Documentation**. https://matplotlib.org/stable/index.html
6. **Seaborn Documentation**. https://seaborn.pydata.org/
7. **NetworkX Documentation**. https://networkx.org/documentation/stable/
8. **Kaggle Datasets – Groceries Dataset**. https://www.kaggle.com/datasets/heeraldedhia/groceries-dataset