Name :- Raghav Tigadi
UCID :- rt486
Email :- rt486@njit.edu
Date :-
Professor : - Yasser Abduallah
CS634 <>Data Mining

Midterm Project Report

Implementation and Code Usage

Apriori Algorithm Implementation in Retail Data Mining:

i. Abstract :-

In this research, I use the Apriori Algorithm, a fundamental data mining approach, to identify correlations in retail transactions. I evaluated the algorithm's effectiveness and efficiency using several data mining techniques and approaches. Through design moreover, creating customised data mining tools, I develop a customised model to extract relevant insights from transaction data.

ii. Introduction :-

Data mining represents a potent methodological approach aimed at unveiling latent patterns and correlations embedded within expansive datasets. Our endeavour is centred around the Apriori Algorithm, a seminal technique renowned for its prowess in mining association rules, particularly within the domain of retail. We shall expound upon the fundamental tenets and principles of data mining that underpin our study.

Crafting and presenting association rules constitute the crux of the Apriori Algorithm. Its essence lies in establishing correlations. To achieve this, a meticulous examination of the transaction list was imperative to identify the most prevalent items. Subsequently, contingent upon the user's specified support parameter, the support for each item had to be computed. Upon deriving the support values for each item, those failing to meet the predefined support threshold could be discarded. The Apriori algorithm stands as a quintessential data mining tool, employing a brute force methodology to unearth frequent itemsets and devise association rules. Its modus operandi involves iteratively augmenting the scale of itemsets while sieving out those failing to surpass a designated support threshold.

In this particular implementation, I applied the Apriori algorithm to a bespoke dataset associated with a retail establishment, enabling us to identify prevalent itemsets and infer association rules. Critical stages in this procedural sequence encompassed.

Subsequent Procedures :

- Step 1: Setting up dictionaries to hold candidate and frequent itemsets.
- Step 2: Importing the dataset and itemsets from CSV files.
- Step 3: Preparing the dataset to ensure item order and uniqueness are maintained.
- Sep 4: Gathering user input for minimum support and confidence thresholds.
- Step 5: Sequentially producing candidate itemsets and refining frequent itemsets through the Apriori algorithm, which employs a methodical approach by exhaustively exploring all conceivable item combinations.

iii. Foundational Concepts and Principles:

Discovery of Common Itemsets:
The essence of the Apriori Algorithm lies in uncovering common itemsets, denoting groups of items recurrently appearing together in transactions. These sets offer valuable insights into customer purchasing patterns and preferences.

Support and Confidence:
In the realm of data mining, pivotal metrics include support and confidence. Support quantifies the frequency of occurrence for an item or itemset, while confidence evaluates the probability of items being bought in tandem. These metrics serve as guiding principles for our analytical endeavours.

Association Rules :
Through the identification of robust association rules, I ascertain which items are frequently bought in conjunction. These rules play a crucial role in enhancing sales strategies, including personalised recommendations.

iV. Project Workflow :

Our project adheres to a methodical workflow encompassing multiple stages and the utilisation of the Apriori Algorithm.

Data Loading and Preprocessing :
Commencing with the acquisition of transactional data from a retail store dataset, each transaction embodies a roster of items procured by a customer. Prior to analysis, meticulous preprocessing of the dataset ensues, involving the curation of distinct items and their arrangement according to a predetermined sequence for enhanced data fidelity.

Establishing Minimum Support and Confidence Thresholds :
User input holds paramount importance in the realm of data mining. We solicit the user's specifications regarding minimum support and confidence thresholds, essential for sieving out less salient patterns.

Iterating Through Candidate Itemsets :
The iterative process of applying the Apriori Algorithm entails the generation of candidate itemsets progressively expanding in size. We commence with individual items (itemset size K = 1), advancing to K = 2, K = 3, and beyond. This iterative procedure employs a methodical "brute force" approach to exhaustively generate all feasible combinations of itemsets.

Support Count Computation :
In the determination of each candidate itemset, we ascertain its support through the tallying of transactions encompassing the said itemset. Those itemsets aligning with the predefined minimum support threshold are preserved, while those failing to meet the criterion are disregarded.

Confidence Computation:
We assess the confidence of association rules, signifying the potency of correlations among items. This process demands a thorough juxtaposition of support values pertaining to individual items and itemsets.

Association Rule Formation:
We extract association rules that fulfil both the minimum support and minimum confidence criteria. These rules unveil invaluable insights into the frequent co-purchasing of items.

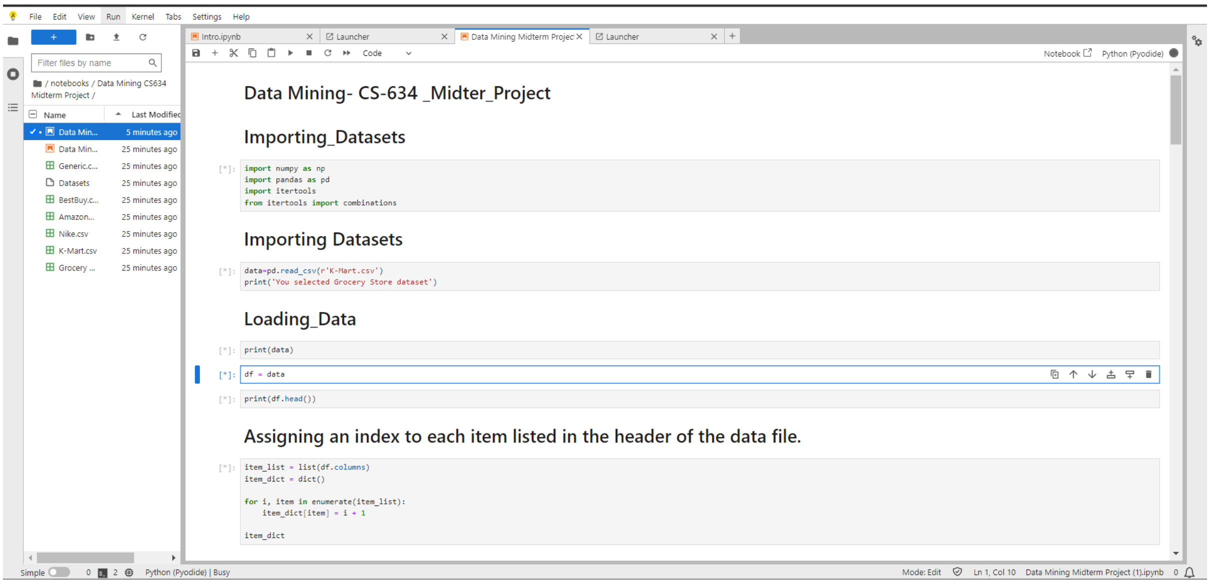
<u>V. Results and Evaluation :</u>

The project's efficacy and efficiency are appraised through performance metrics encompassing support, confidence, and the resultant association rules. Additionally, we conduct a comparative analysis between our bespoke Apriori Algorithm implementation and the Apriori library to gauge its dependability.

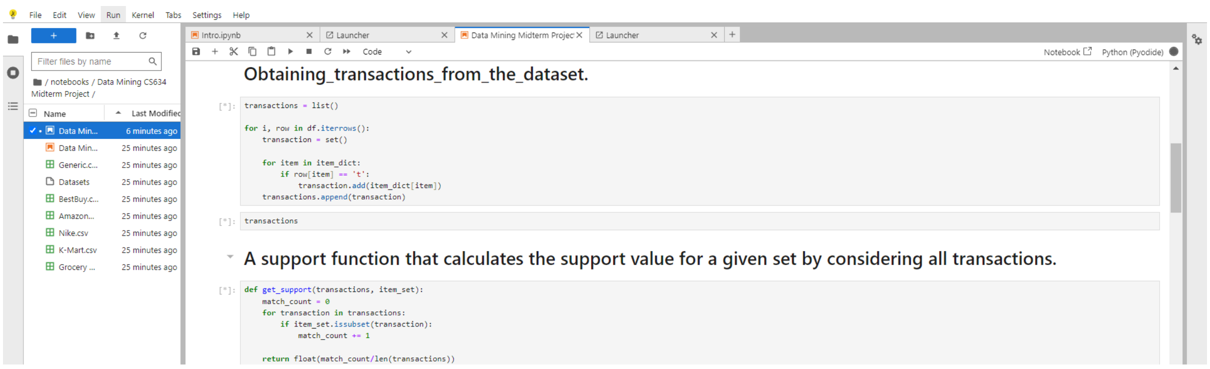
<u>Vi. Conclusion</u> :

In summary, our project showcases the practical utilisation of data mining ideologies, principles, and techniques. We effectively deployed the Apriori Algorithm to derive significant association rules from retail transactional data. The iterative process, employing a methodical "brute force" strategy, alongside the bespoke algorithmic design and strict adherence to user-defined parameters, underscores the efficacy of data mining in unveiling pivotal patterns conducive to informed decision-making within the retail sector.

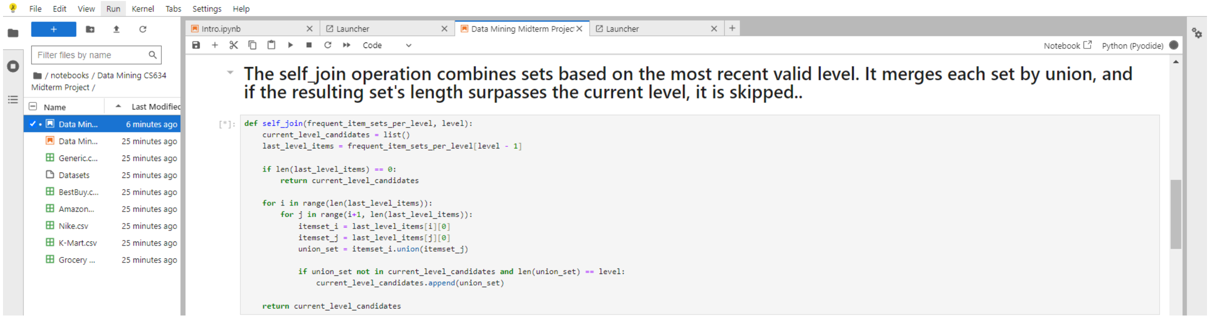
<u>Vii. Screenshots (Code Part) :</u>



(a)



(b)


The self_join operation combines sets based on the most recent valid level. It merges each set by union, and if the resulting set's length surpasses the current level, it is skipped..
def self_join(frequent_item_sets_per_level, level):
    current_level_candidates = list()
    last_level_items = frequent_item_sets_per_level[level - 1]

    if len(last_level_items) == 0:
        return current_level_candidates

    for i in range(len(last_level_items)):
        for j in range(i+1, len(last_level_items)):
            itemset_i = last_level_items[i][0]
            itemset_j = last_level_items[j][0]
            union_set = itemset_i.union(itemset_j)

            if union_set not in current_level_candidates and len(union_set) == level:
                current_level_candidates.append(union_set)

    return current_level_candidates

(c)


def get_single_drop_subsets(item_set):
    single_drop_subsets = list()
    for item in item_set:
        temp = item_set.copy()
        temp.remove(item)
        single_drop_subsets.append(temp)

    return single_drop_subsets

def is_valid_set(item_set, prev_level_sets):
    single_drop_subsets = get_single_drop_subsets(item_set)

    for single_drop_set in single_drop_subsets:
        if single_drop_set not in prev_level_sets:
            return False
    return True

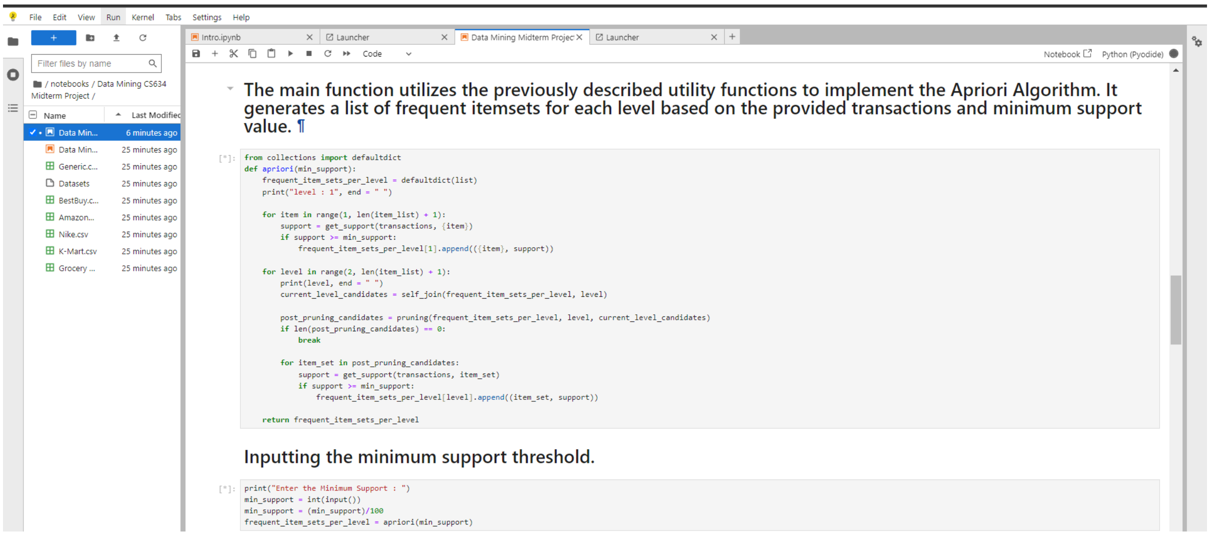
def pruning(frequent_item_sets_per_level, level, candidate_set):
    post_pruning_set = list()
    if len(candidate_set) == 0:
        return post_pruning_set

    prev_level_sets = list()
    for item_set, _ in frequent_item_sets_per_level[level - 1]:
        prev_level_sets.append(item_set)

    for item_set in candidate_set:
        if is_valid_set(item_set, prev_level_sets):
            post_pruning_set.append(item_set)

    return post_pruning_set

(d)


The main function utilizes the previously described utility functions to implement the Apriori Algorithm. It generates a list of frequent itemsets for each level based on the provided transactions and minimum support value. ¶
from collections import defaultdict
def apriori(min_support):
    frequent_item_sets_per_level = defaultdict(list)
    print("level : 1", end = " ")

    for item in range(1, len(item_list) + 1):
        support = get_support(transactions, {item})
        if support >= min_support:
            frequent_item_sets_per_level[1].append(({item}, support))

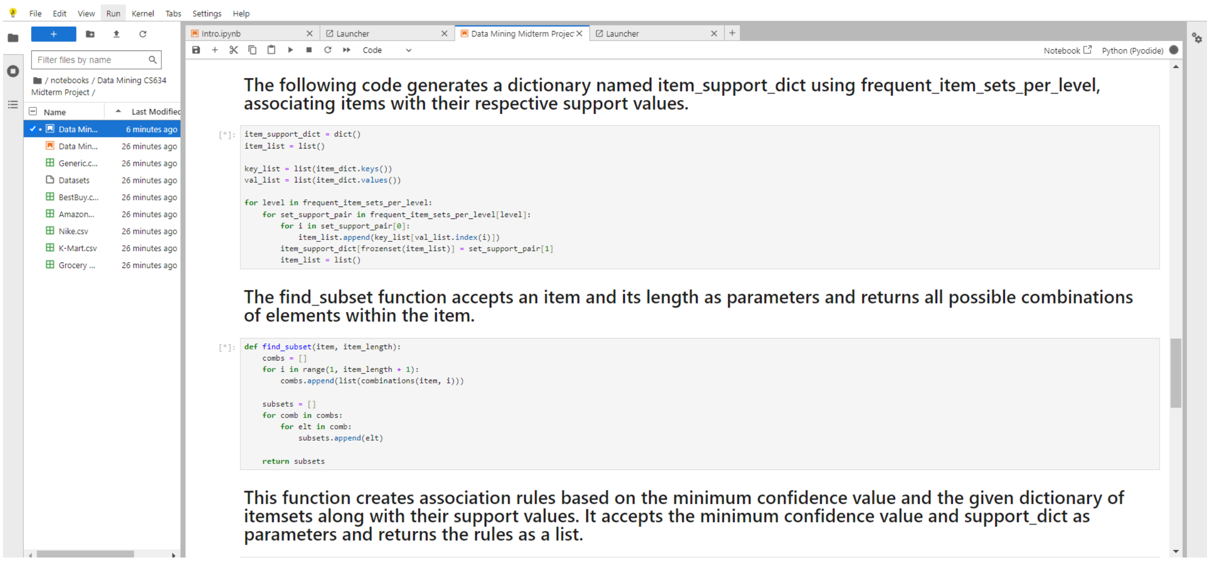
    for level in range(2, len(item_list) + 1):
        print(level, end = " ")
        current_level_candidates = self_join(frequent_item_sets_per_level, level)

        post_pruning_candidates = pruning(frequent_item_sets_per_level, level, current_level_candidates)
        if len(post_pruning_candidates) == 0:
            break

        for item_set in post_pruning_candidates:
            support = get_support(transactions, item_set)
            if support >= min_support:
                frequent_item_sets_per_level[level].append((item_set, support))

    return frequent_item_sets_per_level
Inputting the minimum support threshold.
print("Enter the Minimum Support : ")
min_support = int(input())
min_support = (min_support)/100
frequent_item_sets_per_level = apriori(min_support)

(e)


The following code generates a dictionary named item_support_dict using frequent_item_sets_per_level, associating items with their respective support values.
item_support_dict = dict()
item_list = list()

key_list = list(item_dict.keys())
val_list = list(item_dict.values())

for level in frequent_item_sets_per_level:
    for set_support_pair in frequent_item_sets_per_level[level]:
        for i in set_support_pair[0]:
            item_list.append(key_list[val_list.index(i)])
        item_support_dict[frozenset(item_list)] = set_support_pair[1]
        item_list = list()
The find_subset function accepts an item and its length as parameters and returns all possible combinations of elements within the item.
def find_subset(item, item_length):
    combs = []
    for i in range(1, item_length + 1):
        combs.append(list(combinations(item, i)))

    subsets = []
    for comb in combs:
        for elt in comb:
            subsets.append(elt)

    return subsets
This function creates association rules based on the minimum confidence value and the given dictionary of itemsets along with their support values. It accepts the minimum confidence value and support_dict as parameters and returns the rules as a list.

(f)

This function creates association rules based on the minimum confidence value and the given dictionary of itemsets along with their support values. It accepts the minimum confidence value and support_dict as parameters and returns the rules as a list.

```python
def association_rules(min_confidence, support_dict):
    rules = list()
    for item, support in support_dict.items():
        item_length = len(item)

        if item_length > 1:
            subsets = find_subset(item, item_length)

            for A in subsets:
                B = item.difference(A)

                if B:
                    A = frozenset(A)

                    AB = A | B

                    confidence = support_dict[AB] / support_dict[A]
                    if confidence >= min_confidence:
                        rules.append((A, B, confidence))

    return rules
```

```python
print("Enter the Minimum Confidence : ")
confidence = int(input())
min_confidence = (confidence)/100
association_rules = association_rules(min_confidence, support_dict = item_support_dict)
```

(g)

Screenshots (End Result Out put Part) :

In [76]:
```python
print("Number of rules: ", len(association_rules))

for rule in association_rules:
    print('{0} -> {1} <confidence: {2}>'.format(set(rule[0]), set(rule[1]), rule[2]))
```

```
Number of rules:  32
{'Bedspreads'} -> {'Quilts'} <confidence: 0.46153846153846l5>
{'Quilts'} -> {'Bedspreads'} <confidence: 0.6666666666666666>
{'Decorative Pillows'} -> {'Quilts'} <confidence: 0.5384615384615384>
{'Quilts'} -> {'Decorative Pillows'} <confidence: 0.7777777777777778>
{'Quilts'} -> {'Shams'} <confidence: 0.7777777777777778>
{'Shams'} -> {'Quilts'} <confidence: 0.7777777777777778>
{'Bedspreads'} -> {'Decorative Pillows'} <confidence: 0.6923076923076923>
{'Decorative Pillows'} -> {'Bedspreads'} <confidence: 0.6923076923076923>
{'Bedspreads'} -> {'Bedding Collections'} <confidence: 0.5384615384615384>
{'Bedding Collections'} -> {'Bedspreads'} <confidence: 0.6363636363636364>
{'Bedspreads'} -> {' Kids Bedding'} <confidence: 0.4615384615384615>
{' Kids Bedding'} -> {'Bedspreads'} <confidence: 0.6666666666666666>
{'Bedspreads'} -> {'Embroidered Bedspread'} <confidence: 0.6153846153846153>
{'Embroidered Bedspread'} -> {'Bedspreads'} <confidence: 0.8888888888888888>
{'Bed Skirts'} -> {'Decorative Pillows'} <confidence: 0.75>
{'Decorative Pillows'} -> {'Bed Skirts'} <confidence: 0.4615384615384615>
{'Decorative Pillows'} -> {'Shams'} <confidence: 0.4615384615384615>
{'Shams'} -> {'Decorative Pillows'} <confidence: 0.6666666666666666>
{'Decorative Pillows'} -> {'Bedding Collections'} <confidence: 0.6153846153846153>
{'Bedding Collections'} -> {'Decorative Pillows'} <confidence: 0.7272727272727272>
{'Decorative Pillows'} -> {' Kids Bedding'} <confidence: 0.6153846153846153>
{' Kids Bedding'} -> {'Decorative Pillows'} <confidence: 0.8888888888888888>
{'Bed Skirts'} -> {'Bedding Collections'} <confidence: 0.875>
{'Bedding Collections'} -> {'Bed Skirts'} <confidence: 0.6363636363636364>
{'Bedding Collections'} -> {' Kids Bedding'} <confidence: 0.5454545454545454>
{' Kids Bedding'} -> {'Bedding Collections'} <confidence: 0.6666666666666666>
{'Decorative Pillows'} -> {'Quilts', 'Shams'} <confidence: 0.4615384615384615>
{'Quilts'} -> {'Decorative Pillows', 'Shams'} <confidence: 0.6666666666666666>
{'Shams'} -> {'Decorative Pillows', 'Quilts'} <confidence: 0.6666666666666666>
{'Decorative Pillows', 'Quilts'} -> {'Shams'} <confidence: 0.8571428571428571>
{'Decorative Pillows', 'Shams'} -> {'Quilts'} <confidence: 1.0>
{'Quilts', 'Shams'} -> {'Decorative Pillows'} <confidence: 0.8571428571428571>
```

Data Base - K- Mart

In [19]:
```python
print("Number of rules: ", len(association_rules))

for rule in association_rules:
    print('{0} -> {1} <confidence: {2}>'.format(set(rule[0]), set(rule[1]), rule[2]))
```

```
Number of rules:  4038
{'Java: The Complete Reference'} -> {'A Beginner's Guide'} <confidence: 0.2857142857142857>
{'A Beginner's Guide'} -> {'Java: The Complete Reference'} <confidence: 0.25>
{'Java For Dummies'} -> {'A Beginner's Guide'} <confidence: 0.8571428571428571>
{'A Beginner's Guide'} -> {'Java For Dummies'} <confidence: 0.75>
{'Android Programming: The Big Nerd Ranch'} -> {'A Beginner's Guide'} <confidence: 0.4444444444444444>
{'A Beginner's Guide'} -> {'Android Programming: The Big Nerd Ranch'} <confidence: 0.5>
{'A Beginner's Guide'} -> {'Head First Java 2nd Edition'} <confidence: 0.375>
{'Head First Java 2nd Edition'} -> {'A Beginner's Guide'} <confidence: 0.5>
{'Beginning Programming with Java'} -> {'A Beginner's Guide'} <confidence: 0.375>
{'A Beginner's Guide'} -> {'Beginning Programming with Java'} <confidence: 0.375>
{'A Beginner's Guide'} -> {'Java 8 Pocket Guide'} <confidence: 0.625>
{'Java 8 Pocket Guide'} -> {'A Beginner's Guide'} <confidence: 0.5555555555555556>
{'C++ Programming in Easy Steps'} -> {'A Beginner's Guide'} <confidence: 0.4>
{'A Beginner's Guide'} -> {'C++ Programming in Easy Steps'} <confidence: 0.25>
{'A Beginner's Guide'} -> {'Effective Java (2nd Edition)'} <confidence: 0.625>
{'Effective Java (2nd Edition)'} -> {'A Beginner's Guide'} <confidence: 0.8333333333333334>
{'HTML and CSS: Design and Build Websites'} -> {'A Beginner's Guide'} <confidence: 0.3333333333333333>
{'A Beginner's Guide'} -> {'HTML and CSS: Design and Build Websites'} <confidence: 0.25>
```

Data Base - Amazon

```
In [36]: print("Number of rules: ", len(association_rules))

         for rule in association_rules:
             print('{0} -> {1} <confidence: {2}>'.format(set(rule[0]), set(rule[1]), rule[2]))
```

```
Number of rules:  4874
{' LapTop'} -> {'Digital Camera '} <confidence: 0.5>
{'Digital Camera '} -> {' LapTop'} <confidence: 0.6>
{'Desk Top'} -> {'Digital Camera '} <confidence: 0.5555555555555556>
{'Digital Camera '} -> {'Desk Top'} <confidence: 1.0>
{'Digital Camera '} -> {'Printer'} <confidence: 0.6>
{'Printer'} -> {'Digital Camera '} <confidence: 0.6>
{'Digital Camera '} -> {'Flash Drive'} <confidence: 0.4>
{'Flash Drive'} -> {'Digital Camera '} <confidence: 0.3333333333333333>
{'Microsoft Office'} -> {'Digital Camera '} <confidence: 0.2>
{'Digital Camera '} -> {'Microsoft Office'} <confidence: 0.2>
{'Digital Camera '} -> {'Speakers'} <confidence: 0.6>
{'Speakers'} -> {'Digital Camera '} <confidence: 0.375>
{'Digital Camera '} -> {'LapTop Case'} <confidence: 0.4>
{'LapTop Case'} -> {'Digital Camera '} <confidence: 0.25>
{'Digital Camera '} -> {'Anti-Virus'} <confidence: 0.6>
{'Anti-Virus'} -> {'Digital Camera '} <confidence: 0.42857142857142855>
{'Digital Camera '} -> {'External HardDrive'} <confidence: 0.4>
{'External HardDrive'} -> {'Digital Camera '} <confidence: 0.4>
```

```
In [ ]:
```

Data Base - Best Buy

```
In [57]: print("Number of rules: ", len(association_rules))

         for rule in association_rules:
             print('{0} -> {1} <confidence: {2}>'.format(set(rule[0]), set(rule[1]), rule[2]))
```

```
Number of rules:  136
{'A'} -> {'B'} <confidence: 0.625>
{'B'} -> {'A'} <confidence: 0.5555555555555556>
{'A'} -> {'C'} <confidence: 0.125>
{'C'} -> {'A'} <confidence: 0.2>
{'A'} -> {'D'} <confidence: 0.5>
{'D'} -> {'A'} <confidence: 0.5>
{'A'} -> {'E'} <confidence: 0.25>
{'E'} -> {'A'} <confidence: 0.3333333333333333>
{'A'} -> {'F'} <confidence: 0.375>
{'F'} -> {'A'} <confidence: 0.5>
{'C'} -> {'B'} <confidence: 0.4>
{'B'} -> {'C'} <confidence: 0.2222222222222222>
{'B'} -> {'D'} <confidence: 0.2222222222222222>
{'D'} -> {'B'} <confidence: 0.25>
{'E'} -> {'B'} <confidence: 0.5>
{'B'} -> {'E'} <confidence: 0.3333333333333333>
{'F'} -> {'B'} <confidence: 0.3333333333333333>
{'B'} -> {'F'} <confidence: 0.2222222222222222>
```

```
In [ ]:
```

Data Base - Generic

```
In [75]: print("Number of rules: ", len(association_rules))

         for rule in association_rules:
             print('{0} -> {1} <confidence: {2}>'.format(set(rule[0]), set(rule[1]), rule[2]))
```

```
Number of rules:  336
{'bread'} -> {'milk'} <confidence: 0.3076923076923077>
{'milk'} -> {'bread'} <confidence: 0.8>
{'biscuit'} -> {'milk'} <confidence: 0.28571428571428575>
{'milk'} -> {'biscuit'} <confidence: 0.4>
{'cornflakes'} -> {'milk'} <confidence: 0.33333333333333337>
{'milk'} -> {'cornflakes'} <confidence: 0.4>
{'jam'} -> {'milk'} <confidence: 0.5>
{'milk'} -> {'jam'} <confidence: 0.2>
{'maggi'} -> {'milk'} <confidence: 0.2>
{'milk'} -> {'maggi'} <confidence: 0.2>
{'tea'} -> {'milk'} <confidence: 0.14285714285714288>
{'milk'} -> {'tea'} <confidence: 0.2>
{'coffee'} -> {'milk'} <confidence: 0.125>
{'milk'} -> {'coffee'} <confidence: 0.2>
{'biscuit'} -> {'bread'} <confidence: 0.5714285714285715>
{'bread'} -> {'biscuit'} <confidence: 0.3076923076923077>
{'cornflakes'} -> {'bread'} <confidence: 0.16666666666666669>
{'bread'} -> {'cornflakes'} <confidence: 0.07692307692307693>
```

```
In [ ]:
```

Data Base - Grocery

```
In [93]: print("Number of rules: ", len(association_rules))

         for rule in association_rules:
             print('{0} -> {1} <confidence: {2}>'.format(set(rule[0]), set(rule[1]), rule[2]))
```

```
Number of rules:  1712
{'Running Shoe'} -> {'Soccer Shoe'} <confidence: 0.3>
{'Soccer Shoe'} -> {'Running Shoe'} <confidence: 0.3333333333333333>
{'Running Shoe'} -> {'Socks'} <confidence: 0.1>
{'Socks'} -> {'Running Shoe'} <confidence: 0.2>
{'Running Shoe'} -> {'Swimming Shirt'} <confidence: 0.4>
{'Swimming Shirt'} -> {'Running Shoe'} <confidence: 0.8>
{'Running Shoe'} -> {'Dry Fit V-Nick'} <confidence: 0.4>
{'Dry Fit V-Nick'} -> {'Running Shoe'} <confidence: 0.5714285714285714>
{'Rash Guard'} -> {'Running Shoe'} <confidence: 0.6>
{'Running Shoe'} -> {'Rash Guard'} <confidence: 0.3>
{'Running Shoe'} -> {'Sweatshirts'} <confidence: 0.4>
{'Sweatshirts'} -> {'Running Shoe'} <confidence: 0.5>
{'Running Shoe'} -> {'Hoodies'} <confidence: 0.4>
{'Hoodies'} -> {'Running Shoe'} <confidence: 0.5714285714285714>
{'Running Shoe'} -> {'Tech Pants'} <confidence: 0.3>
{'Tech Pants'} -> {'Running Shoe'} <confidence: 0.5>
{'Running Shoe'} -> {'Modern Pants'} <confidence: 0.7>
{'Modern Pants'} -> {'Running Shoe'} <confidence: 0.875>
```

```
In [ ]:
```

Data Base - Nike

jupyter Data Mining Midterm Project (1) Last Checkpoint: 2 days ago

File Edit View Run Kernel Settings Help Not Trusted

Code JupyterLab Python 3 (ipykernel)

```
{'Decorative Pillows'} -> {'Quilts', 'Shams'} <confidence: 0.46153846153846115>
{'Quilts', 'Shams'} -> {'Decorative Pillows'} <confidence: 0.8571428571428571>
{'Decorative Pillows', 'Shams'} -> {'Quilts'} <confidence: 1.0>
{'Quilts', 'Decorative Pillows'} -> {'Shams'} <confidence: 0.8571428571428571>
```

Using the_mlxtend Librarry and implementing the the apriori algorithm

```
[25]: from mlxtend.frequent_patterns import apriori
      from mlxtend.frequent_patterns import association_rules

      # Convert the 'Items' column to a one-hot encoded format
      df_encoded = df.replace('t', 1)

      # Apply Apriori algorithm
      frequent_itemsets = apriori(df_encoded.fillna(0), min_support=min_support, use_colnames=True)

      # Generate association rules
      rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=min_confidence)

      # Display the frequent itemsets and association rules
      print("Frequent Itemsets:")
      print(frequent_itemsets)
```

```
Frequent Itemsets:
    support                  itemsets
0   0.473684                 (Quilts)
1   0.684211             (Bedspreads)
2   0.684211     (Decorative Pillows)
3   0.421053             (Bed Skirts)
4   0.473684                  (Shams)
```

implementing the apriori algorithm

File Edit View Run Kernel Settings Help Not Trusted

Markdown JupyterLab Python 3 (ipykernel)

```
Frequent Itemsets:
    support                                    itemsets
0   0.473684                                   (Quilts)
1   0.684211                               (Bedspreads)
2   0.684211                       (Decorative Pillows)
3   0.421053                               (Bed Skirts)
4   0.473684                                    (Shams)
5   0.578947                      (Bedding Collections)
6   0.473684                             ( Kids Bedding)
7   0.473684                     (Embroidered Bedspread)
8   0.315789                                   (Towels)
9   0.315789                       (Bedspreads, Quilts)
10  0.368421               (Quilts, Decorative Pillows)
11  0.368421                            (Quilts, Shams)
12  0.473684           (Bedspreads, Decorative Pillows)
13  0.368421          (Bedspreads, Bedding Collections)
14  0.315789                 ( Kids Bedding, Bedspreads)
15  0.421053        (Bedspreads, Embroidered Bedspread)
16  0.315789           (Bed Skirts, Decorative Pillows)
17  0.315789                (Shams, Decorative Pillows)
18  0.421053  (Decorative Pillows, Bedding Collections)
19  0.421053         ( Kids Bedding, Decorative Pillows)
20  0.368421          (Bed Skirts, Bedding Collections)
21  0.315789        ( Kids Bedding, Bedding Collections)
22  0.315789        (Shams, Quilts, Decorative Pillows)
```

Test Results (i)

```
Jupyter Data Mining Midterm Project (1) Last Checkpoint: 2 days ago
File Edit View Run Kernel Settings Help                         Not Trusted
Markdown                                JupyterLab  Python 3 (ipykernel)
22  0.315789        (Shams, Quilts, Decorative Pillows)

Association Rules:
                    antecedents                 consequents
0                 (Bedspreads)                    (Quilts)
1                     (Quilts)                (Bedspreads)
2                     (Quilts)        (Decorative Pillows)
3         (Decorative Pillows)                    (Quilts)
4                     (Quilts)                     (Shams)
5                      (Shams)                    (Quilts)
6                 (Bedspreads)        (Decorative Pillows)
7         (Decorative Pillows)                (Bedspreads)
8                 (Bedspreads)       (Bedding Collections)
9        (Bedding Collections)                (Bedspreads)
10               ( Kids Bedding)              (Bedspreads)
11                (Bedspreads)             ( Kids Bedding)
12                (Bedspreads)       (Embroidered Bedspread)
13     (Embroidered Bedspread)                (Bedspreads)
14                (Bed Skirts)        (Decorative Pillows)
15        (Decorative Pillows)                (Bed Skirts)
16                     (Shams)        (Decorative Pillows)
17        (Decorative Pillows)                     (Shams)
18        (Decorative Pillows)       (Bedding Collections)
19       (Bedding Collections)        (Decorative Pillows)
20             ( Kids Bedding)        (Decorative Pillows)
21        (Decorative Pillows)             ( Kids Bedding)
22                (Bed Skirts)       (Bedding Collections)
23       (Bedding Collections)                (Bed Skirts)
24             ( Kids Bedding)       (Bedding Collections)
25       (Bedding Collections)             ( Kids Bedding)
26             (Quilts, Shams)        (Decorative Pillows)
27  (Decorative Pillows, Shams)                   (Quilts)
28 (Quilts, Decorative Pillows)                    (Shams)
```

Test Results(ii)

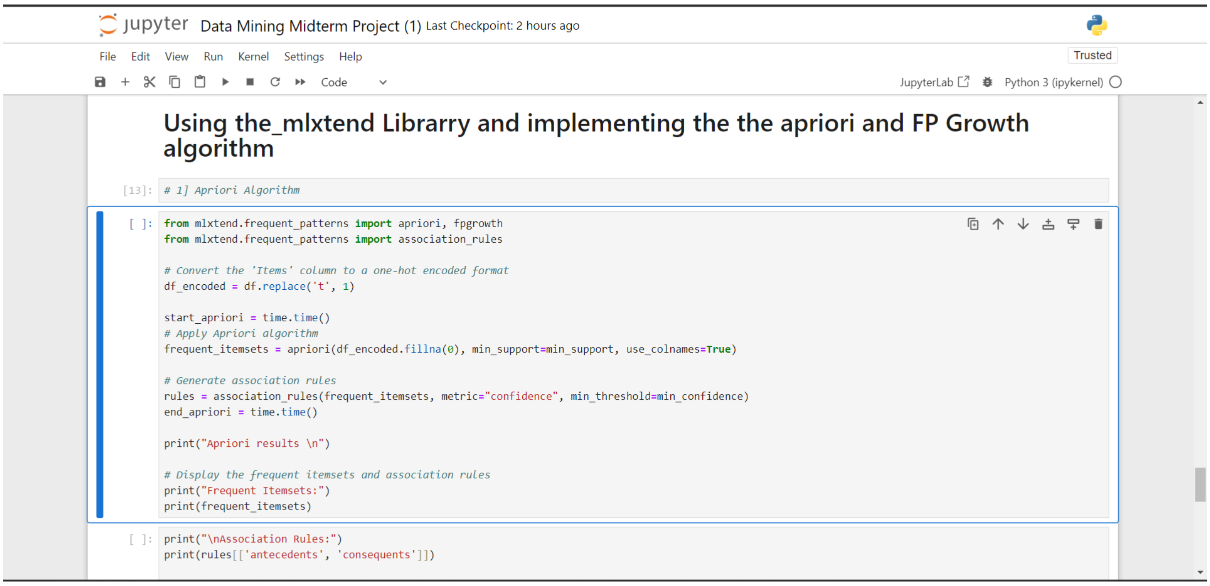
```
Jupyter Data Mining Midterm Project (1) Last Checkpoint: 2 days ago
File Edit View Run Kernel Settings Help                             Trusted
Code                                    JupyterLab  Python 3 (ipykernel)
[20]: print("\nAssociation Rules:")
      print(rules[['antecedents', 'consequents']])

Association Rules:
                    antecedents                 consequents
0                     (Quilts)                (Bedspreads)
1                 (Bedspreads)                    (Quilts)
2         (Decorative Pillows)                    (Quilts)
3                     (Quilts)        (Decorative Pillows)
4                      (Shams)                    (Quilts)
5                     (Quilts)                     (Shams)
6         (Decorative Pillows)                (Bedspreads)
7                 (Bedspreads)        (Decorative Pillows)
8        (Bedding Collections)                (Bedspreads)
9                 (Bedspreads)       (Bedding Collections)
10             ( Kids Bedding)                (Bedspreads)
11                (Bedspreads)             ( Kids Bedding)
12     (Embroidered Bedspread)                (Bedspreads)
13                (Bedspreads)     (Embroidered Bedspread)
14        (Decorative Pillows)                (Bed Skirts)
15                (Bed Skirts)        (Decorative Pillows)
16        (Decorative Pillows)                     (Shams)
17                     (Shams)        (Decorative Pillows)
18        (Decorative Pillows)       (Bedding Collections)
19       (Bedding Collections)        (Decorative Pillows)
20        (Decorative Pillows)             ( Kids Bedding)
21             ( Kids Bedding)        (Decorative Pillows)
22       (Bedding Collections)                (Bed Skirts)
23                (Bed Skirts)       (Bedding Collections)
24             ( Kids Bedding)       (Bedding Collections)
25       (Bedding Collections)             ( Kids Bedding)
```

Test Results(iii)

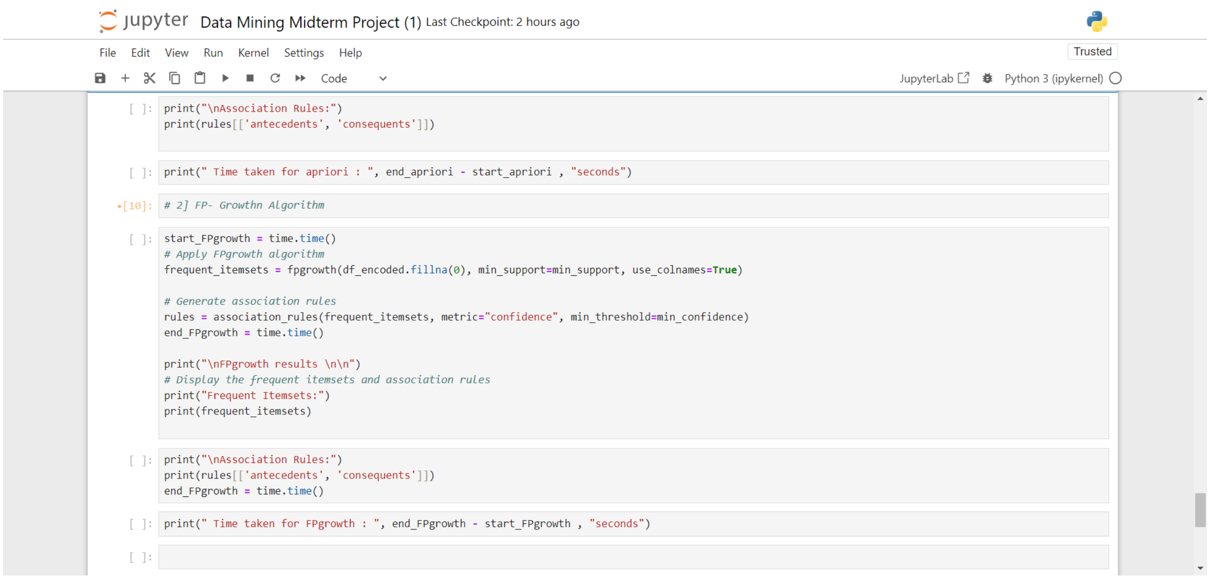
```
Jupyter Data Mining Midterm Project (1) Last Checkpoint: 2 days ago
File Edit View Run Kernel Settings Help                             Trusted
Code                                    JupyterLab  Python 3 (ipykernel)
1                 (Bedspreads)                    (Quilts)
2         (Decorative Pillows)                    (Quilts)
3                     (Quilts)        (Decorative Pillows)
4                      (Shams)                    (Quilts)
5                     (Quilts)                     (Shams)
6         (Decorative Pillows)                (Bedspreads)
7                 (Bedspreads)        (Decorative Pillows)
8        (Bedding Collections)                (Bedspreads)
9                 (Bedspreads)       (Bedding Collections)
10             ( Kids Bedding)                (Bedspreads)
11                (Bedspreads)             ( Kids Bedding)
12     (Embroidered Bedspread)                (Bedspreads)
13                (Bedspreads)     (Embroidered Bedspread)
14        (Decorative Pillows)                (Bed Skirts)
15                (Bed Skirts)        (Decorative Pillows)
16        (Decorative Pillows)                     (Shams)
17                     (Shams)        (Decorative Pillows)
18        (Decorative Pillows)       (Bedding Collections)
19       (Bedding Collections)        (Decorative Pillows)
20        (Decorative Pillows)             ( Kids Bedding)
21             ( Kids Bedding)        (Decorative Pillows)
22       (Bedding Collections)                (Bed Skirts)
23                (Bed Skirts)       (Bedding Collections)
24             ( Kids Bedding)       (Bedding Collections)
25       (Bedding Collections)             ( Kids Bedding)
26  (Decorative Pillows, Shams)                   (Quilts)
27 (Decorative Pillows, Quilts)                    (Shams)
28             (Quilts, Shams)        (Decorative Pillows)
29        (Decorative Pillows)             (Quilts, Shams)
30                     (Shams) (Decorative Pillows, Quilts)
31                    (Quilts)  (Decorative Pillows, Shams)
```

Test Results (iv)

<u>End Result outputs of Appriori and FP growth Algorithm:</u>

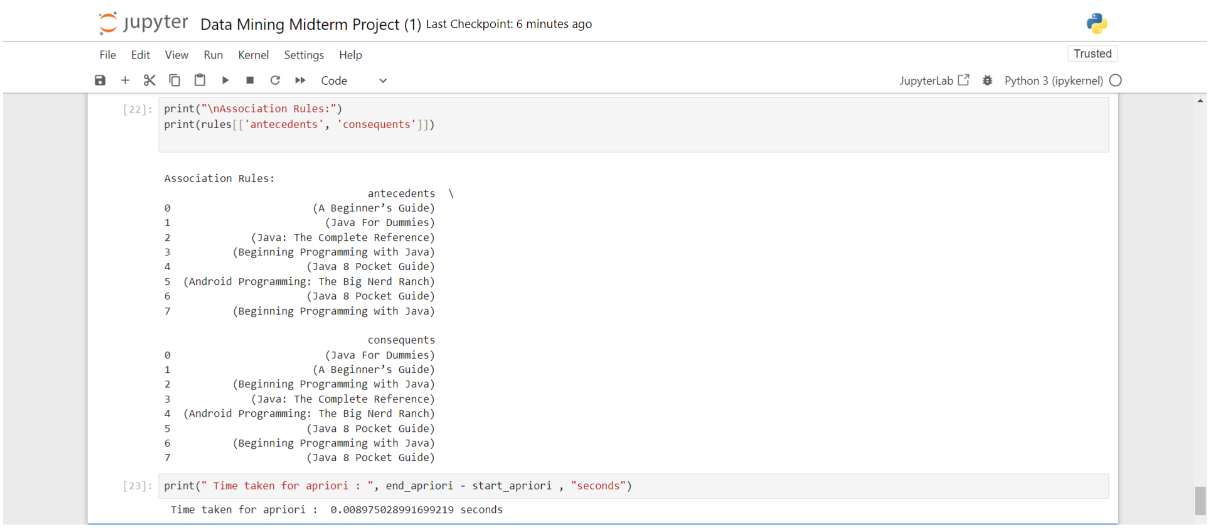


(i. Apriori Algorithm)



(ii. FP-Growth Algorithm)

Results On Each Data Set (Apriori and FP-Growth Algorithm)



(i. Amazon)

jupyter Data Mining Midterm Project (1) Last Checkpoint: 13 minutes ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

Using the_mlxtend Librarry and implementing the the apriori and FP Growth algorithm

```
[45]: # 1] Apriori Algorithm
```

```
[46]: from mlxtend.frequent_patterns import apriori, fpgrowth
from mlxtend.frequent_patterns import association_rules

# Convert the 'Items' column to a one-hot encoded format
df_encoded = df.replace('t', 1)

start_apriori = time.time()
# Apply Apriori algorithm
frequent_itemsets = apriori(df_encoded.fillna(0), min_support=min_support, use_colnames=True)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=min_confidence)
end_apriori = time.time()

print("Apriori results \n")

# Display the frequent itemsets and association rules
print("Frequent Itemsets:")
print(frequent_itemsets)
```

ii. Best Buy

jupyter Data Mining Midterm Project (1) Last Checkpoint: 14 minutes ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

```
Apriori results

Frequent Itemsets:
    support                         itemsets
0    0.3125                (Digital Camera )
1    0.3750                        ( LapTop)
2    0.5625                       (Desk Top)
3    0.3125                        (Printer)
4    0.3750                    (Flash Drive)
5    0.3125               (Microsoft Office)
6    0.5000                       (Speakers)
7    0.5000                    (LapTop Case)
8    0.4375                      (Anti-Virus)
9    0.3125              (External HardDrive)
10   0.3125        (Desk Top, Digital Camera )
11   0.3125               (Desk Top, Printer)
12   0.3125              (Desk Top, Speakers)
13   0.3125           (Speakers, LapTop Case)
14   0.3125         (Anti-Virus, LapTop Case)
15   0.3125  (Anti-Virus, External HardDrive)
```

```
[47]: print("\nAssociation Rules:")
print(rules[['antecedents', 'consequents']])
```

(Best Buy - Apriori Algorithm)

jupyter Data Mining Midterm Project (1) Last Checkpoint: 18 minutes ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

```
Association Rules:
             antecedents           consequents
0             (Desk Top)     (Digital Camera )
1      (Digital Camera )            (Desk Top)
2             (Desk Top)             (Printer)
3              (Printer)            (Desk Top)
4             (Desk Top)            (Speakers)
5             (Speakers)            (Desk Top)
6             (Speakers)         (LapTop Case)
7          (LapTop Case)            (Speakers)
8           (Anti-Virus)         (LapTop Case)
9          (LapTop Case)          (Anti-Virus)
10          (Anti-Virus)  (External HardDrive)
11  (External HardDrive)          (Anti-Virus)
```

```
[48]: print(" Time taken for apriori : ", end_apriori - start_apriori , "seconds")

 Time taken for apriori :  0.010971307754516602 seconds
```

(Best Buy - FP-Growth Algorithm)

jupyter Data Mining Midterm Project (1) Last Checkpoint: 25 minutes ago

File Edit View Run Kernel Settings Help Trusted

JupyterLab Python 3 (ipykernel)

```
# Generate association rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=min_confidence)
end_apriori = time.time()

print("Apriori results \n")

# Display the frequent itemsets and association rules
print("Frequent Itemsets:")
print(frequent_itemsets)
```

```
Apriori results

Frequent Itemsets:
    support itemsets
0  0.421053      (A)
1  0.473684      (B)
2  0.421053      (D)
3  0.315789      (E)
4  0.315789      (F)
```

[70]:
```
print("\nAssociation Rules:")
print(rules[['antecedents', 'consequents']])
```

```
Association Rules:
Empty DataFrame
Columns: [antecedents, consequents]
Index: []
```

[71]:
```
print(" Time taken for apriori : ", end_apriori - start_apriori , "seconds")
```

```
 Time taken for apriori :  0.006438016891479492 seconds
```

iii. Generic Apriori Algorithm

jupyter Data Mining Midterm Project (1) Last Checkpoint: 25 minutes ago

File Edit View Run Kernel Settings Help Trusted

JupyterLab Python 3 (ipykernel)

[72]:
```
# 2] FP- Growthn Algorithm
```

[73]:
```
start_FPgrowth = time.time()
# Apply FPgrowth algorithm
frequent_itemsets = fpgrowth(df_encoded.fillna(0), min_support=min_support, use_colnames=True)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=min_confidence)
end_FPgrowth = time.time()

print("\nFPgrowth results \n\n")
# Display the frequent itemsets and association rules
print("Frequent Itemsets:")
print(frequent_itemsets)
```

```
FPgrowth results


Frequent Itemsets:
    support itemsets
0  0.421053      (D)
1  0.421053      (A)
2  0.315789      (F)
3  0.473684      (B)
4  0.315789      (E)
```

[74]:
```
print("\nAssociation Rules:")
print(rules[['antecedents', 'consequents']])
end_FPgrowth = time.time()
```

(Generic FP Growth)

jupyter Data Mining Midterm Project (1) Last Checkpoint: 30 minutes ago

File Edit View Run Kernel Settings Help Trusted

JupyterLab Python 3 (ipykernel)

```
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=min_confidence)
end_apriori = time.time()

print("Apriori results \n")

# Display the frequent itemsets and association rules
print("Frequent Itemsets:")
print(frequent_itemsets)
```

```
Apriori results

Frequent Itemsets:
   support      itemsets
0     0.65       (bread)
1     0.35     (biscuit)
2     0.30  (cornflakes)
3     0.35         (tea)
4     0.40      (coffee)
5     0.30       (sugar)
```

[95]:
```
print("\nAssociation Rules:")
print(rules[['antecedents', 'consequents']])
```

```
Association Rules:
Empty DataFrame
Columns: [antecedents, consequents]
Index: []
```

[96]:
```
print(" Time taken for apriori : ", end_apriori - start_apriori , "seconds")
```

```
 Time taken for apriori :  0.007322549819946289 seconds
```

iv Grocery Apriori Algorithm

Jupyter Data Mining Midterm Project (1) Last Checkpoint: 30 minutes ago

File Edit View Run Kernel Settings Help — Trusted

JupyterLab — Python 3 (ipykernel)

```python
print("\nFPgrowth results \n\n")
# Display the frequent itemsets and association rules
print("Frequent Itemsets:")
print(frequent_itemsets)
```

```
FPgrowth results


Frequent Itemsets:
   support      itemsets
0     0.65       (bread)
1     0.35     (biscuit)
2     0.30  (cornflakes)
3     0.35         (tea)
4     0.40      (coffee)
5     0.30       (sugar)
```

[99]:
```python
print("\nAssociation Rules:")
print(rules[['antecedents', 'consequents']])
end_FPgrowth = time.time()
```

```
Association Rules:
Empty DataFrame
Columns: [antecedents, consequents]
Index: []
```

[100]:
```python
print(" Time taken for FPgrowth : ", end_FPgrowth - start_FPgrowth , "seconds")
```

```
 Time taken for FPgrowth :  0.7708413600921631 seconds
```

(FP-Growth Algorithm)

```
FPgrowth results


Frequent Itemsets:
     support                            itemsets
0   0.684211                 (Decorative Pillows)
1   0.578947                (Bedding Collections)
2   0.473684                       ( Kids Bedding)
3   0.473684                            (Quilts)
4   0.421053                        (Bed Skirts)
5   0.315789                            (Towels)
6   0.684211                        (Bedspreads)
7   0.473684              (Embroidered Bedspread)
8   0.473684                             (Shams)
9   0.421053  (Decorative Pillows, Bedding Collections)
10  0.368421        (Bedding Collections, Bedspreads)
11  0.421053       ( Kids Bedding, Decorative Pillows)
12  0.315789      ( Kids Bedding, Bedding Collections)
13  0.315789               ( Kids Bedding, Bedspreads)
14  0.368421             (Quilts, Decorative Pillows)
15  0.315789                     (Quilts, Bedspreads)
16  0.368421                          (Quilts, Shams)
17  0.315789      (Quilts, Decorative Pillows, Shams)
18  0.368421        (Bed Skirts, Bedding Collections)
19  0.315789         (Bed Skirts, Decorative Pillows)
20  0.473684         (Decorative Pillows, Bedspreads)
21  0.421053      (Bedspreads, Embroidered Bedspread)
22  0.315789              (Decorative Pillows, Shams)
```

V K-mart Apriori Algorithm

```
Association Rules:
                  antecedents                 consequents
0        (Decorative Pillows)       (Bedding Collections)
1       (Bedding Collections)        (Decorative Pillows)
2       (Bedding Collections)                (Bedspreads)
3                (Bedspreads)       (Bedding Collections)
4              ( Kids Bedding)       (Decorative Pillows)
5        (Decorative Pillows)              ( Kids Bedding)
6              ( Kids Bedding)      (Bedding Collections)
7       (Bedding Collections)              ( Kids Bedding)
8              ( Kids Bedding)               (Bedspreads)
9                (Bedspreads)              ( Kids Bedding)
10                   (Quilts)        (Decorative Pillows)
11       (Decorative Pillows)                    (Quilts)
12                   (Quilts)                (Bedspreads)
13               (Bedspreads)                    (Quilts)
14                   (Quilts)                     (Shams)
15                    (Shams)                    (Quilts)
16 (Quilts, Decorative Pillows)                   (Shams)
17            (Quilts, Shams)        (Decorative Pillows)
18  (Decorative Pillows, Shams)                  (Quilts)
19                   (Quilts) (Decorative Pillows, Shams)
20       (Decorative Pillows)             (Quilts, Shams)
21                    (Shams) (Quilts, Decorative Pillows)
22               (Bed Skirts)       (Bedding Collections)
23      (Bedding Collections)                (Bed Skirts)
24               (Bed Skirts)        (Decorative Pillows)
25       (Decorative Pillows)                (Bed Skirts)
26       (Decorative Pillows)                (Bedspreads)
27               (Bedspreads)        (Decorative Pillows)
28               (Bedspreads)     (Embroidered Bedspread)
29    (Embroidered Bedspread)                (Bedspreads)
30       (Decorative Pillows)                     (Shams)
```

```
8             ( Kids Bedding)                  (Bedspreads)
9                (Bedspreads)               ( Kids Bedding)
10                  (Quilts)          (Decorative Pillows)
11       (Decorative Pillows)                      (Quilts)
12                  (Quilts)                  (Bedspreads)
13               (Bedspreads)                      (Quilts)
14                  (Quilts)                       (Shams)
15                   (Shams)                      (Quilts)
16  (Quilts, Decorative Pillows)                   (Shams)
17            (Quilts, Shams)          (Decorative Pillows)
18   (Decorative Pillows, Shams)                   (Quilts)
19                  (Quilts)   (Decorative Pillows, Shams)
20       (Decorative Pillows)               (Quilts, Shams)
21                   (Shams)  (Quilts, Decorative Pillows)
22              (Bed Skirts)          (Bedding Collections)
23      (Bedding Collections)                 (Bed Skirts)
24              (Bed Skirts)          (Decorative Pillows)
25       (Decorative Pillows)                 (Bed Skirts)
26       (Decorative Pillows)                 (Bedspreads)
27               (Bedspreads)         (Decorative Pillows)
28               (Bedspreads)       (Embroidered Bedspread)
29    (Embroidered Bedspread)                 (Bedspreads)
30       (Decorative Pillows)                      (Shams)
31                   (Shams)          (Decorative Pillows)
```

```
[25]: print(" Time taken for FPgrowth : ", end_FPgrowth - start_FPgrowth , "seconds")
```

```
 Time taken for FPgrowth :  1.7616987228393555 seconds
```

K-Mart FP - Growth

```
Apriori results

Frequent Itemsets:
    support                    itemsets
0  0.526316              (Running Shoe)
1  0.473684               (Soccer Shoe)
2  0.368421             (Dry Fit V-Nick)
3  0.421053               (Sweatshirts)
4  0.368421                   (Hoodies)
5  0.315789                (Tech Pants)
6  0.421053              (Modern Pants)
7  0.368421  (Running Shoe, Modern Pants)
```

```
[45]: print("\nAssociation Rules:")
      print(rules[['antecedents', 'consequents']])
```

```
Association Rules:
      antecedents     consequents
0  (Running Shoe)  (Modern Pants)
1  (Modern Pants)  (Running Shoe)
```

```
[46]: print(" Time taken for apriori : ", end_apriori - start_apriori , "seconds")
```

```
 Time taken for apriori :  0.007979869842529297 seconds
```

```
[47]: # 2] FP- Growthn Algorithm
```

```
[48]: start_FPgrowth = time.time()
      # Apply FPgrowth algorithm
      frequent_itemsets = fpgrowth(df_encoded.fillna(0), min_support=min_support, use_colnames=True)
```

Vi Nike Apriori Algorithm

```
FPgrowth results


Frequent Itemsets:
    support                    itemsets
0  0.526316              (Running Shoe)
1  0.421053              (Modern Pants)
2  0.421053               (Sweatshirts)
3  0.368421             (Dry Fit V-Nick)
4  0.473684               (Soccer Shoe)
5  0.315789                (Tech Pants)
6  0.368421                   (Hoodies)
7  0.368421  (Running Shoe, Modern Pants)
```

```
[49]: print("\nAssociation Rules:")
      print(rules[['antecedents', 'consequents']])
      end_FPgrowth = time.time()
```

```
Association Rules:
      antecedents     consequents
0  (Running Shoe)  (Modern Pants)
1  (Modern Pants)  (Running Shoe)
```

```
[50]: print(" Time taken for FPgrowth : ", end_FPgrowth - start_FPgrowth , "seconds")
```

```
 Time taken for FPgrowth :  1.8089354038238525 seconds
```

Nike - FP-Growth

Git Repository :
https://github.com/Raghavnjit22/Data-Mining-CS-634-Midterm-Project