Name: Gavin Pinto
NJIT UCID: gkp2
Email Address: gkp2@njit.edu
Date: 13 October 2024
Professor: Yasser Abduallah
CS 634 101 Data Mining

Midterm Project Report

Implementation and Code Usage

Brute Force Algorithm in Retail Mining

Abstract:
In this project, the Brute Force Algorithm was explored in order to determine its efficiency to mine association rules in transactional datasets. It was implemented using python and data mining formulas, and then compared with industry-standard algorithms like the Apriori and FP-Growth Algorithm. Through the use of data mining techniques, a custom model was created to mine insightful patterns from transactional retail data.

Introduction:
Data mining is the process of discovering patterns in data. These patterns can be useful for all stakeholders of the data involved, providing valuable insights and sometimes even acting as catalysts for important business decisions. This project focuses on mining patterns in transactional data, which in this project represents purchases made with items from various companies.

The way the patterns and rules were generated was the Brute Force Algorithm.

First, it needs to generate the frequent itemsets, which are the groups of items from the dataset that appear most frequently in the data. Here, “frequently” means that the groups had a support greater than or equal to the minimum support, a user-defined parameter.

Second, it takes the frequent itemsets and generates the association rules, which are rules that imply that if a certain group of items are purchased, then it is likely that another group is purchased as well. Here, “likely” means that the rule has a confidence greater than or equal to the minimum confidence, a user-defined parameter.

Support and confidence are described in the core concepts section.

Core Concepts:

Association Rules: These are rules that imply that the purchase of an itemset will involve the purchase of another itemset in the same transaction, above a specified confidence value.

Confidence: This is a measure of how likely an itemset's purchase implies the purchase of another itemset.

Frequent Itemsets: These are itemsets that occur frequently in the transactional database, above a specified support value.

Support: This is a measure of how frequently an itemset occurs in the transactional database.

Project Workflow:

Necessary Imports: In order to run the project, multiple libraries were for common algorithms and data manipulation. These libraries included: mlxtend, pandas, itertools from combinations, association_rules, apriori and fpgrowth from frequent_patterns, and time.

Database Building: The database is built using data from 5 companies: Barnes & Noble, Costco, Gorilla Mind, Rogue Fitness, and Vilros. Popular items from each company's website were selected and manually entered into the files with the suffix "dataset," for example "dataset_barnes_and_noble.csv." Next, a python script, transaction_generator.py, was ran *once* to generate random transactions for each company, and stored in files with the prefix "transaction," for example "transaction_barnes_and_noble.csv." Each transaction file contains 20 transactions.

Data Loading and Preprocessing: The datasets and transactions are extracted from their respective file types from the database folder and stored in lists in order to be used by the algorithm.

Helper Functions for Brute Force Algorithm: To make the code for the actual algorithm simple and readable, helper functions were written to be used inside the main algorithm. This included functions such as getting the support and confidence of a given itemset and rule respectively, generating all possible itemsets from a dataset, and generating all possible rules from a given itemset.

Implementation of Brute Force Algorithm: The function brute_force_frequent_itemsets is written to find all the frequent itemsets that occurred in the transactional database. This is then used for brute_force_association_rules, which is written to take the frequent itemsets, generate rules for each itemset, and return all the rules above the minimum confidence.

User Input for Model Usage: The user is prompted to enter which company they would like to analyze, as well as the minimum support and confidence for the algorithms.

External Algorithms for Comparison: Finally, two algorithms are used to compare against the brute force algorithm: apriori and fp-growth. Although this would output the same results as the brute force algorithm, all three were timed in order to compare performance. The output of each algorithm was also shown, since the order of the rules and itemsets can vary due to different implementations.

Conclusion:

In conclusion, this project demonstrates the application of several data mining techniques and concepts by implementing the brute force algorithm and comparing its performance with the Apriori and FP-Growth algorithms to determine its efficiency and usage for mining association rules and frequent patterns in transactional retail databases.

Screenshots

Below are screenshots of all 5 dataset files (the possible items that appear in the transactions):

--- Barnes and Noble ---

ID	Element
1	The Book of Bill
2	Percy Jackson and the Olympians
3	Harry Potter
4	Warriors
5	The Hunger Games
6	Divergent
7	The Giver
8	Of Mice and Men
9	To Kill a Mockingbird
10	Diary of a Wimpy Kid

--- Costco ---

ID	Element
1	Dave's Killer Bread
2	Season Sardines
3	Hellmann's Mayonnaise
4	Boursin Cheese
5	Perdue Chicken Nuggets
6	Mexican Blend Shredded Cheese
7	Grade A Large Eggs
8	Whole Milk
9	Caramel Tres Leches Cake
10	Farm Raised Atlantic Salmon

--- Gorilla Mind ---

ID	Element
1	Blackberry Lemonade Pre-Workout
2	Birthday Cake Protein Powder
3	Gorilla Dream Sleep and Recovery Formula
4	Creatine
5	Liquid Glycerol
6	Collagen Peptides
7	Melatonin
8	Sauna Tee
9	Helimix Shaker
10	Omega-3 Elixir

--- Rogue Fitness ---

ID	Element
1	Smith Machine
2	45 lb Plate
3	Barbell
4	75 lb Dumbbells
5	Cable Machine
6	Pendulum Squat
7	Adjustable Bench
8	T-Bar Row
9	Seated Hamstring Curl
10	Pec Deck Machine

--- Vilros ---

ID	Element
1	Raspberry Pi 4 Model B
2	Standard to Micro HDMI
3	64GB Samsung Evo Micro SD Card
4	USB-C 5V 3A Power Supply
5	Wireless SNES Style Gamepad
6	Rosin Soldering Flux Paste
7	Heavy-Duty Aluminum Alloy Case
8	7 Inch Touchscreen LCD Display
9	Heatsink
10	Mini Speaker

Below are screenshots of all 5 transactional files:

--- Barnes and Noble ---

ID	Transaction
1	Divergent,Of Mice and Men,The Giver
2	Diary of a Wimpy Kid,Of Mice and Men,The Giver
3	Diary of a Wimpy Kid,Of Mice and Men,The Giver,To Kill a Mockingbird,Warriors
4	Divergent,Percy Jackson and the Olympians,The Giver,To Kill a Mockingbird,Warriors
5	Divergent,Harry Potter,Of Mice and Men,Percy Jackson and the Olympians,The Book of Bill
6	Divergent,Harry Potter,The Giver
7	Divergent,Of Mice and Men,Percy Jackson and the Olympians,The Book of Bill,The Giver
8	Percy Jackson and the Olympians,The Book of Bill,The Hunger Games,To Kill a Mockingbird
9	Diary of a Wimpy Kid,Percy Jackson and the Olympians,The Giver,To Kill a Mockingbird,Warriors
10	Harry Potter,The Book of Bill,The Giver,The Hunger Games,Warriors
11	Divergent,Percy Jackson and the Olympians,The Hunger Games,Warriors
12	Diary of a Wimpy Kid,Harry Potter,Percy Jackson and the Olympians,The Book of Bill,To Kill a Mockingbird
13	Divergent,Percy Jackson and the Olympians,The Hunger Games,To Kill a Mockingbird
14	Diary of a Wimpy Kid,Of Mice and Men,Percy Jackson and the Olympians,The Hunger Games,Warriors
15	The Book of Bill,The Giver,The Hunger Games,Warriors
16	Diary of a Wimpy Kid,Harry Potter,Of Mice and Men,The Book of Bill,To Kill a Mockingbird
17	Divergent,Percy Jackson and the Olympians,The Book of Bill,The Giver
18	Divergent,The Book of Bill,The Hunger Games,To Kill a Mockingbird
19	Diary of a Wimpy Kid,The Hunger Games,Warriors
20	Diary of a Wimpy Kid,The Giver,Warriors

--- Costco ---

ID	Transaction
1	Hellmann's Mayonnaise,Mexican Blend Shredded Cheese,Perdue Chicken Nuggets,Whole Milk
2	Boursin Cheese,Caramel Tres Leches Cake,Dave's Killer Bread
3	Boursin Cheese,Caramel Tres Leches Cake,Hellmann's Mayonnaise,Mexican Blend Shredded Cheese,Whole Milk
4	Caramel Tres Leches Cake,Dave's Killer Bread,Grade A Large Eggs
5	Farm Raised Atlantic Salmon,Hellmann's Mayonnaise,Perdue Chicken Nuggets
6	Caramel Tres Leches Cake,Grade A Large Eggs,Mexican Blend Shredded Cheese,Season Sardines,Whole Milk
7	Boursin Cheese,Hellmann's Mayonnaise,Season Sardines
8	Farm Raised Atlantic Salmon,Grade A Large Eggs,Perdue Chicken Nuggets
9	Dave's Killer Bread,Grade A Large Eggs,Season Sardines,Whole Milk
10	Boursin Cheese,Farm Raised Atlantic Salmon,Hellmann's Mayonnaise,Mexican Blend Shredded Cheese,Season Sardines
11	Grade A Large Eggs,Hellmann's Mayonnaise,Season Sardines
12	Boursin Cheese,Dave's Killer Bread,Mexican Blend Shredded Cheese,Perdue Chicken Nuggets
13	Boursin Cheese,Caramel Tres Leches Cake,Dave's Killer Bread,Mexican Blend Shredded Cheese
14	Dave's Killer Bread,Grade A Large Eggs,Hellmann's Mayonnaise,Perdue Chicken Nuggets,Season Sardines
15	Boursin Cheese,Caramel Tres Leches Cake,Whole Milk
16	Boursin Cheese,Dave's Killer Bread,Grade A Large Eggs,Hellmann's Mayonnaise
17	Boursin Cheese,Farm Raised Atlantic Salmon,Hellmann's Mayonnaise,Season Sardines,Whole Milk
18	Boursin Cheese,Farm Raised Atlantic Salmon,Hellmann's Mayonnaise,Mexican Blend Shredded Cheese
19	Boursin Cheese,Dave's Killer Bread,Mexican Blend Shredded Cheese,Perdue Chicken Nuggets
20	Boursin Cheese,Caramel Tres Leches Cake,Dave's Killer Bread,Hellmann's Mayonnaise,Whole Milk

--- Gorilla Mind ---

ID	Transaction
1	Birthday Cake Protein Powder,Gorilla Dream Sleep and Recovery Formula,Liquid Glycerol,Melatonin,Omega-3 Elixir
2	Blackberry Lemonade Pre-Workout,Creatine,Gorilla Dream Sleep and Recovery Formula,Melatonin,Omega-3 Elixir
3	Collagen Peptides,Creatine,Sauna Tee
4	Collagen Peptides,Creatine,Gorilla Dream Sleep and Recovery Formula,Melatonin
5	Creatine,Gorilla Dream Sleep and Recovery Formula,Liquid Glycerol
6	Blackberry Lemonade Pre-Workout,Melatonin,Omega-3 Elixir,Sauna Tee
7	Blackberry Lemonade Pre-Workout,Creatine,Helimix Shaker
8	Blackberry Lemonade Pre-Workout,Collagen Peptides,Omega-3 Elixir
9	Creatine,Helimix Shaker,Melatonin,Omega-3 Elixir
10	Collagen Peptides,Creatine,Gorilla Dream Sleep and Recovery Formula,Liquid Glycerol,Omega-3 Elixir
11	Blackberry Lemonade Pre-Workout,Collagen Peptides,Creatine,Liquid Glycerol,Sauna Tee
12	Gorilla Dream Sleep and Recovery Formula,Melatonin,Omega-3 Elixir
13	Gorilla Dream Sleep and Recovery Formula,Helimix Shaker,Omega-3 Elixir,Sauna Tee
14	Birthday Cake Protein Powder,Liquid Glycerol,Melatonin,Sauna Tee
15	Birthday Cake Protein Powder,Gorilla Dream Sleep and Recovery Formula,Melatonin
16	Birthday Cake Protein Powder,Gorilla Dream Sleep and Recovery Formula,Melatonin
17	Collagen Peptides,Creatine,Liquid Glycerol,Melatonin,Sauna Tee
18	Birthday Cake Protein Powder,Creatine,Gorilla Dream Sleep and Recovery Formula,Melatonin,Omega-3 Elixir
19	Blackberry Lemonade Pre-Workout,Collagen Peptides,Gorilla Dream Sleep and Recovery Formula,Omega-3 Elixir,Sauna Tee
20	Blackberry Lemonade Pre-Workout,Creatine,Liquid Glycerol,Omega-3 Elixir,Sauna Tee

--- Rogue Fitness ---

ID	Transaction
1	45 lb Plate,Cable Machine,Pec Deck Machine
2	Adjustable Bench,Barbell,Pendulum Squat,Seated Hamstring Curl,T-Bar Row
3	Cable Machine,Seated Hamstring Curl,Smith Machine
4	45 lb Plate,75 lb Dumbbells,Adjustable Bench,Barbell,Seated Hamstring Curl
5	Adjustable Bench,Barbell,Pec Deck Machine,T-Bar Row
6	45 lb Plate,75 lb Dumbbells,Adjustable Bench,Smith Machine,T-Bar Row
7	45 lb Plate,Adjustable Bench,Pec Deck Machine,Seated Hamstring Curl,Smith Machine
8	Pec Deck Machine,Pendulum Squat,Seated Hamstring Curl
9	45 lb Plate,75 lb Dumbbells,Seated Hamstring Curl
10	75 lb Dumbbells,Adjustable Bench,Seated Hamstring Curl,T-Bar Row
11	45 lb Plate,Pendulum Squat,T-Bar Row
12	45 lb Plate,Adjustable Bench,Pendulum Squat
13	45 lb Plate,Cable Machine,T-Bar Row
14	75 lb Dumbbells,Barbell,Seated Hamstring Curl
15	Adjustable Bench,Pendulum Squat,Seated Hamstring Curl,T-Bar Row
16	45 lb Plate,Barbell,Pendulum Squat,Seated Hamstring Curl
17	Adjustable Bench,Barbell,Pec Deck Machine,Pendulum Squat
18	45 lb Plate,Barbell,Seated Hamstring Curl,T-Bar Row
19	45 lb Plate,Adjustable Bench,Barbell,Pendulum Squat
20	Adjustable Bench,Barbell,Cable Machine,Smith Machine

--- Vilros ---

ID	Transaction
1	64GB Samsung Evo Micro SD Card,Mini Speaker,USB-C 5V 3A Power Supply
2	7 Inch Touchscreen LCD Display,Heatsink,Mini Speaker,Wireless SNES Style Gamepad
3	64GB Samsung Evo Micro SD Card,7 Inch Touchscreen LCD Display,Mini Speaker
4	7 Inch Touchscreen LCD Display,Heatsink,Mini Speaker,Wireless SNES Style Gamepad
5	7 Inch Touchscreen LCD Display,Heavy-Duty Aluminum Alloy Case,Raspberry Pi 4 Model B,Standard to Micro HDMI,USB-C 5V 3A Power Supply
6	Heavy-Duty Aluminum Alloy Case,Mini Speaker,Raspberry Pi 4 Model B,Rosin Soldering Flux Paste,Standard to Micro HDMI
7	7 Inch Touchscreen LCD Display,Raspberry Pi 4 Model B,Rosin Soldering Flux Paste,Standard to Micro HDMI,USB-C 5V 3A Power Supply
8	7 Inch Touchscreen LCD Display,Heatsink,Heavy-Duty Aluminum Alloy Case,Wireless SNES Style Gamepad
9	64GB Samsung Evo Micro SD Card,Heatsink,Raspberry Pi 4 Model B,Standard to Micro HDMI
10	Raspberry Pi 4 Model B,Standard to Micro HDMI,USB-C 5V 3A Power Supply
11	Heavy-Duty Aluminum Alloy Case,Mini Speaker,Rosin Soldering Flux Paste,Standard to Micro HDMI
12	7 Inch Touchscreen LCD Display,Heavy-Duty Aluminum Alloy Case,Rosin Soldering Flux Paste,Standard to Micro HDMI,Wireless SNES Style Gamepad
13	64GB Samsung Evo Micro SD Card,Heatsink,Standard to Micro HDMI,Wireless SNES Style Gamepad
14	64GB Samsung Evo Micro SD Card,USB-C 5V 3A Power Supply,Wireless SNES Style Gamepad
15	Heatsink,Raspberry Pi 4 Model B,Rosin Soldering Flux Paste,Standard to Micro HDMI,USB-C 5V 3A Power Supply
16	64GB Samsung Evo Micro SD Card,Heatsink,Rosin Soldering Flux Paste,USB-C 5V 3A Power Supply
17	64GB Samsung Evo Micro SD Card,Heatsink,Rosin Soldering Flux Paste,Wireless SNES Style Gamepad
18	7 Inch Touchscreen LCD Display,Standard to Micro HDMI,USB-C 5V 3A Power Supply
19	64GB Samsung Evo Micro SD Card,Mini Speaker,Raspberry Pi 4 Model B,USB-C 5V 3A Power Supply
20	7 Inch Touchscreen LCD Display,Raspberry Pi 4 Model B,Standard to Micro HDMI,USB-C 5V 3A Power Supply

Below is a screenshot of transaction_generator.py:

```python
import random
import pandas as pd

''' This file generates the transactional database for the datasets. This was ran ONCE to generate the files. '''

# Generates a random transaction of a given length from a given array with no repeats
def generate_random_transaction (array, length):
    # Keep track of usable elements from array
    available_elements = array.copy()
    transaction = []
    # Append to transaction length times
    for i in range(length):
        # Choose random element from available elements and add it to transaction
        random_index = random.randint(0, len(available_elements) - 1)
        transaction.append(available_elements[random_index])
        # Prevent repeats
        available_elements.pop(random_index)
    # Sort final transaction alphabetically
    return sorted(transaction)

# Appends random transactions generated from dataset_file to transaction_file
def append_transactions_csv (num_transactions, dataset_file, transaction_file):
    # Store elements from dataset_file in a list
    df = pd.read_csv(dataset_file)
    elements = df['Element'].tolist()
    transactions = []
    # Append num_transactions transactions
    for i in range(num_transactions):
        # I have decided to make transactions of size 3, 4, or 5
        random_transaction = generate_random_transaction(elements, random.randint(3,5))
        transaction_string = ",".join(random_transaction)
        # Append transaction to transaction_file with i and transaction in ID and Transaction columns
        transactions.append({"ID": i + 1, "Transaction": transaction_string})
    # Convert transactions to df
    transactions_df = pd.DataFrame(transactions)
    # Create file if it doesn't exist and append transactions
    transactions_df.to_csv(transaction_file, mode='a', index=False, header=not pd.io.common.file_exists(transaction_file))

# Generate transaction databases for each company
companies = ["barnes_and_noble", "costco", "gorilla_mind", "rogue_fitness", "vilros"]
for company in companies:
    transaction_filename = "pinto_gavin_midterm_proj/database/transaction_" + company + ".csv"
    dataset_filename = "pinto_gavin_midterm_proj/database/dataset_" + company + ".csv"
    append_transactions_csv(20, dataset_filename, transaction_filename)
```

Below is a screenshot of the necessary imports and file processing helper functions:

```
pip install mlxtend pandas
```

```python
import pandas as pd
from itertools import combinations
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.frequent_patterns import fpgrowth
import time

def get_transactions_as_array (filename):
    df = pd.read_csv(filename)
    transactions = df['Transaction'].tolist()
    return transactions

def get_dataset_as_array (filename):
    df = pd.read_csv(filename)
    elements = df['Element'].tolist()
    return elements
```

Below is a screenshot of the helper functions for the algorithm implementation:

```
# Returns number of occurrences of itemset in transactions
def get_frequency (itemset, transactions):
    frequency = 0
    for transaction in transactions:
        transaction_items = transaction.split(",")
        if set(itemset) <= set(transaction_items):
            frequency += 1
    return frequency

# Returns confidence of rule according to transactions
def get_confidence (rule, transactions):
    # formula: freq(combined) / freq(left)
    combined = rule[0] + rule[1]
    left = rule[0]
    return (1.0 * get_frequency(combined, transactions)) / (1.0 * get_frequency(left, transactions))

# Returns all itemsets of size n from dataset in the form
# list[list[], list[]] with list[0] as the itemsets and list[1] initialized to 0
def get_k_itemsets (dataset, k):
    k_itemsets = []
    frequencies = []
    for i in range(len(dataset) - k + 1):
        element_itemset = [dataset[i]]
        if (k > 1):
            for second in range(i + 1, len(dataset) - k + 2):
                element_itemset = [dataset[i]]
                for j in range(second, second + k - 1):
                    element_itemset.append(dataset[j])
                k_itemsets.append(element_itemset)
                frequencies.append(0)
        else:
            k_itemsets.append(element_itemset)
            frequencies.append(0)
    return [k_itemsets, frequencies]

# Returns all possible rules from itemset in the form [[[],[]], ... ]
def generate_rules(itemset):
    n = len(itemset)
    rules = []
    for r in range(1, n):
        for left in combinations(itemset, r):
            left = list(left)
            right = [item for item in itemset if item not in left]
            rules.append([left, right])
    return rules
```

Below is a screenshot of the Brute Force Algorithm Implementation:

```
# Returns all frequent itemsets from transactions according to min_support
def brute_force_frequent_itemsets (dataset, transactions, min_support):
    # Find all frequent k-itemsets
    frequent_itemsets = []
    frequent_itemsets_frequencies = []
    k = 1
    while True:
        # Generate k-itemsets
        k_itemset_info = get_k_itemsets(dataset, k)
        k_itemsets = k_itemset_info[0]
        k_itemsets_frequencies = k_itemset_info[1]
        frequent_itemsets_found = 0
        # Update frequencies of each itemset in transactions
        for i in range(len(k_itemsets)):
            k_itemsets_frequencies[i] = get_frequency(k_itemsets[i], transactions)
        # Add frequent itemsets
        for i in range(len(k_itemsets)):
            supp = k_itemsets_frequencies[i] / 20
            if supp >= min_support:
                frequent_itemsets.append(k_itemsets[i])
                frequent_itemsets_frequencies.append(k_itemsets_frequencies[i])
                frequent_itemsets_found += 1
        # Terminate algorithm if no frequent k-itemsets found
        if frequent_itemsets_found == 0:
            break
        k += 1
    return [frequent_itemsets, frequent_itemsets_frequencies]

# Returns association rules based on min_confidence
def brute_force_association_rules (dataset, transactions, min_support, min_confidence):
    association_rules = []
    # Get frequent itemsets
    frequent_itemsets = brute_force_frequent_itemsets(dataset, transactions, min_support)[0]
    for itemset in frequent_itemsets:
        if len(itemset) > 1:
            # Get all possible rules of itemset
            rules = generate_rules(itemset)
            # Append all rules above min_confidence to result
            for rule in rules:
                conf = get_confidence(rule, transactions)
                if conf >= min_confidence:
                    association_rules.append([rule, conf])
    return association_rules
```

Below is a screenshot of the user input and its processing:

```
# Prompt the user for their company choice
print("Welcome! Please select the company you'd like to analyze.")
print()
print("1. Barnes & Noble")
print("2. Costco")
print("3. Gorilla Mind")
print("4. Rogue Fitness")
print("5. Vilros")
user_choice = int(input("Please enter the number next to your choice: "))

# Prompt the user for support and confidence
print()
support = float(input("Please enter the minimum support (as a percentage): ")) / 100.0
confidence = float(input("Please enter the minimum confidence (as a percentage): ")) / 100.0
print()

# Process which files to analyze
companies = ["barnes_and_noble", "costco", "gorilla_mind", "rogue_fitness", "vilros"]
company = companies[user_choice - 1]
transaction_file = "database/transaction_" + company + ".csv"
dataset_file = "database/dataset_" + company + ".csv"

dataset = get_dataset_as_array(dataset_file)
transactions = get_transactions_as_array(transaction_file)

# Preprocess data for library implementations
transactions_proper = []
for transaction in transactions:
    transactions_proper.append(transaction.split(","))
df = pd.DataFrame(pd.Series(transactions_proper).apply(lambda x: pd.Series(1, index=x)).fillna(0))
```

Below is a screenshot of the usage of the imported Apriori algorithm and timing:

```
# Start timing for Apriori
start_time = time.time()

# Use the existing Apriori implementation
frequent_itemsets_apriori = apriori(df.astype('bool'), min_support=support, use_colnames=True)
rules_apriori = association_rules(frequent_itemsets_apriori, metric="confidence", min_threshold=confidence)

# End timing for Apriori
end_time = time.time()

# Print Apriori results
print("---APRIORI ALGORITHM---")
print()
print("* Frequent Itemsets (Apriori Algorithm):")
print(frequent_itemsets_apriori)
print("\n* Association Rules (Apriori Algorithm):")
print(rules_apriori[['antecedents', 'consequents', 'confidence']])
elapsed_time = end_time - start_time
print(f"\n* Time taken for Apriori Algorithm: {elapsed_time:.6f} seconds")
print()
```

Below is a screenshot of the usage of the imported FP-Growth algorithm and timing:

```
# Start timing for FP-Growth
start_time = time.time()

# Use the existing package for FP-Growth
frequent_itemsets_fp = fpgrowth(df.astype('bool'), min_support=support, use_colnames=True)
rules_fp = association_rules(frequent_itemsets_fp, metric="confidence", min_threshold=confidence)

# End timing for FP-Growth
end_time = time.time()

# Print FP-Growth results
print("---FP-GROWTH ALGORITHM---")
print()
print("* Frequent Itemsets (FP-Growth):")
print(frequent_itemsets_fp)
print("\n* Association Rules (FP-Growth):")
print(rules_fp[['antecedents', 'consequents', 'confidence']])
elapsed_time = end_time - start_time
print(f"\n* Time taken for FP-Growth: {elapsed_time:.6f} seconds")
print()
```

Below is a screenshot of the usage of the Brute Force algorithm and timing:

```
# Start timing for brute
start_time = time.time()

# Compare the results with the brute force algorithm
frequent_itemsets_brute = brute_force_frequent_itemsets(dataset, transactions, support)[0]
rules_brute = brute_force_association_rules(dataset, transactions, support, confidence)

# End timing for brute
end_time = time.time()

# Print Brute force results
print("---BRUTE FORCE ALGORITHM---")
print()
print("* Frequent Itemsets (Brute Force):")
for item in frequent_itemsets_brute:
    print(item)
print("\n* Association Rules (Brute Force):")
for item in rules_brute:
    rule = item[0]
    conf = item[1]
    print(rule[0], "->", rule[1], f"confidence: {conf:.6f}")
elapsed_time = end_time - start_time
print(f"\n* Time taken for Brute Force Algorithm: {elapsed_time:.6f} seconds")
```

Below are screenshots of the program running in the terminal:

```
Welcome! Please select the company you'd like to analyze.

1. Barnes & Noble
2. Costco
3. Gorilla Mind
4. Rogue Fitness
5. Vilros
Please enter the number next to your choice: 3

Please enter the minimum support (as a percentage): 30
Please enter the minimum confidence (as a percentage): 40
```

Part 1 of output (imported algorithms' output):

```
---APRIORI ALGORITHM---

* Frequent Itemsets (Apriori Algorithm):
   support                                       itemsets
0     0.55        (Gorilla Dream Sleep and Recovery Formula)
1     0.35                               (Liquid Glycerol)
2     0.55                                     (Melatonin)
3     0.55                                (Omega-3 Elixir)
4     0.35               (Blackberry Lemonade Pre-Workout)
5     0.55                                      (Creatine)
6     0.35                             (Collagen Peptides)
7     0.40                                     (Sauna Tee)
8     0.35  (Gorilla Dream Sleep and Recovery Formula, Mel...
9     0.35  (Gorilla Dream Sleep and Recovery Formula, Ome...
10    0.30                     (Omega-3 Elixir, Melatonin)

* Association Rules (Apriori Algorithm):
                                  antecedents                                 consequents  confidence
0  (Gorilla Dream Sleep and Recovery Formula)                                 (Melatonin)    0.636364
1                                 (Melatonin)  (Gorilla Dream Sleep and Recovery Formula)    0.636364
2  (Gorilla Dream Sleep and Recovery Formula)                            (Omega-3 Elixir)    0.636364
3                            (Omega-3 Elixir)  (Gorilla Dream Sleep and Recovery Formula)    0.636364
4                            (Omega-3 Elixir)                                 (Melatonin)    0.545455
5                                 (Melatonin)                            (Omega-3 Elixir)    0.545455

* Time taken for Apriori Algorithm: 0.004043 seconds

---FP-GROWTH ALGORITHM---

* Frequent Itemsets (FP-Growth):
   support                                       itemsets
0     0.55        (Gorilla Dream Sleep and Recovery Formula)
1     0.55                                     (Melatonin)
2     0.55                                (Omega-3 Elixir)
3     0.35                               (Liquid Glycerol)
4     0.55                                      (Creatine)
5     0.35               (Blackberry Lemonade Pre-Workout)
6     0.40                                     (Sauna Tee)
7     0.35                             (Collagen Peptides)
8     0.35  (Gorilla Dream Sleep and Recovery Formula, Mel...
9     0.35  (Gorilla Dream Sleep and Recovery Formula, Ome...
10    0.30                     (Omega-3 Elixir, Melatonin)

* Association Rules (FP-Growth):
                                  antecedents                                 consequents  confidence
0  (Gorilla Dream Sleep and Recovery Formula)                                 (Melatonin)    0.636364
1                                 (Melatonin)  (Gorilla Dream Sleep and Recovery Formula)    0.636364
2  (Gorilla Dream Sleep and Recovery Formula)                            (Omega-3 Elixir)    0.636364
3                            (Omega-3 Elixir)  (Gorilla Dream Sleep and Recovery Formula)    0.636364
4                            (Omega-3 Elixir)                                 (Melatonin)    0.545455
5                                 (Melatonin)                            (Omega-3 Elixir)    0.545455

* Time taken for FP-Growth: 0.006024 seconds
```

Part 2 of output (implemented algorithm output):

```
---BRUTE FORCE ALGORITHM---

* Frequent Itemsets (Brute Force):
['Blackberry Lemonade Pre-Workout']
['Gorilla Dream Sleep and Recovery Formula']
['Creatine']
['Liquid Glycerol']
['Collagen Peptides']
['Melatonin']
['Sauna Tee']
['Omega-3 Elixir']
['Gorilla Dream Sleep and Recovery Formula', 'Melatonin']
['Gorilla Dream Sleep and Recovery Formula', 'Omega-3 Elixir']
['Melatonin', 'Omega-3 Elixir']

* Association Rules (Brute Force):
['Gorilla Dream Sleep and Recovery Formula'] -> ['Melatonin'] confidence: 0.636364
['Melatonin'] -> ['Gorilla Dream Sleep and Recovery Formula'] confidence: 0.636364
['Gorilla Dream Sleep and Recovery Formula'] -> ['Omega-3 Elixir'] confidence: 0.636364
['Omega-3 Elixir'] -> ['Gorilla Dream Sleep and Recovery Formula'] confidence: 0.636364
['Melatonin'] -> ['Omega-3 Elixir'] confidence: 0.545455
['Omega-3 Elixir'] -> ['Melatonin'] confidence: 0.545455

* Time taken for Brute Force Algorithm: 0.004029 seconds
```

Other

The source code and other files as .py files and the database folder are all included in this zip file.

Link to Github Repository: https://github.com/GPintoNJIT/Brute_Force_Rule_Mining