VIETNAM GENERAL CONFEDERATION OF LABOUR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

MINING MASSIVE DATA SETS

Midterm Essay

Instructor: **MSc. NGUYEN THANH AN**

Students: **LÊ QUỐC HUY – 520H0536**

TĂNG ĐẠI – 520H0523

NGUYỄN ĐÌNH VIỆT HOÀNG – 522H0120

NGUYỄN THIÊN HUY – 521H0072

TRƯƠNG HUỲNH ĐĂNG KHOA – 521H0503

HO CHI MINH CITY, 2024

VIETNAM GENERAL CONFEDERATION OF LABOUR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

MINING MASSIVE DATA SETS

Midterm Essay

Instructor: **MSc. NGUYEN THANH AN**
Students: **LÊ QUỐC HUY – 520H0536**
TĂNG ĐẠI – 520H0523
NGUYỄN ĐÌNH VIỆT HOÀNG – 522H0120
NGUYỄN THIÊN HUY – 521H0072
TRƯƠNG HUỲNH ĐĂNG KHOA – 521H0503

HO CHI MINH CITY, 2024

THANK YOU

We would like to sincerely thank MSc. Nguyen Thanh An for teaching us the necessary knowledge and how to present the report in the most complete way

SUMMARY

This report consists of 3 parts: the theory, the group member list and the approaches to solve each task

TABLE OF CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

ABBREVIATIONS

RDD	Resilient Distributed Datasets
API	Application Programming Interface
PCY	Park-Chen-Yu

LIST OF FIGURES

CHAPTER 1 – Big Data Processing

1.1 Resilient Distributed Datasets (RDDs)

- Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark. It is an immutable distributed collection of objects that can be operated on in parallel.
- There are two ways to create RDDs:

+ Creating from an existing dataset in the language being used such as Java, Python, Scala.

+ Fetching from datasets stored in external storage systems like HDFS, Hbase, or relational databases.

- Types of RDDs:

+ RDDs represent a fixed, partitioned collection of records that can be processed in parallel.

+ The records in RDDs can be Java, Scala, or Python objects depending on the choice of the programmer. Unlike DataFrames, each record of a DataFrame must be a structured row containing pre-defined fields.

+ RDDs used to be the primary API in Spark 1.x series and can still be used in version 2.x but are no longer used frequently.

+ RDD API can be used in Python, Scala, or Java: Scala and Java: Comparable performance in most parts. (The biggest cost is when dealing with raw objects) Python: Incurs some performance loss, mainly for serialization between Python processes and the JVM

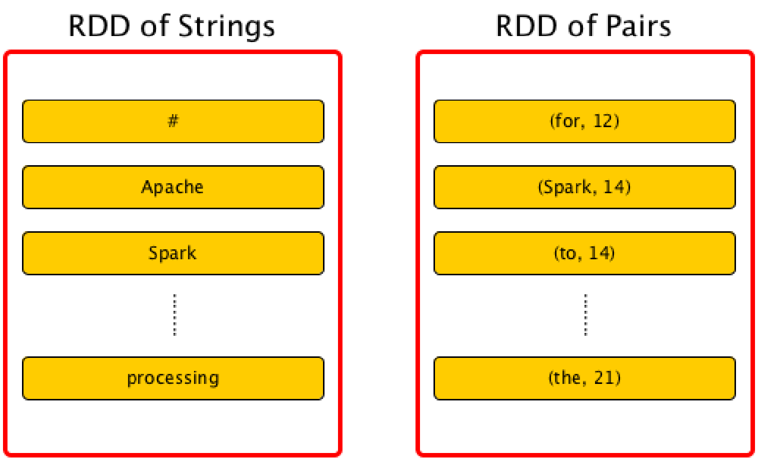


Figure 1.1 Type of RDDs

1.2 DataFrame

- A Dataset is a distributed collection of data. A Dataset can be constructed from JVM objects and then manipulated using functional transformations (map, flatMap, filter, etc.)

- A DataFrame is an immutable, distributed collection of data that is organized into rows, where each one consists a set of columns and each column has a name and an associated type

The characteristics include:

- Immutable: DataFrame's immutability implies that the data within a DataFrame remains unchanged once created. If modifications are required, a new DataFrame needs to be created from the original one, utilizing the DataFrame API.
- Rows: These represent individual data records. A DataFrame comprises a distributed collection of rows.
- Set of columns with names and associated types: This indicates that the data within a DataFrame is structured, consisting of named columns with associated data types.

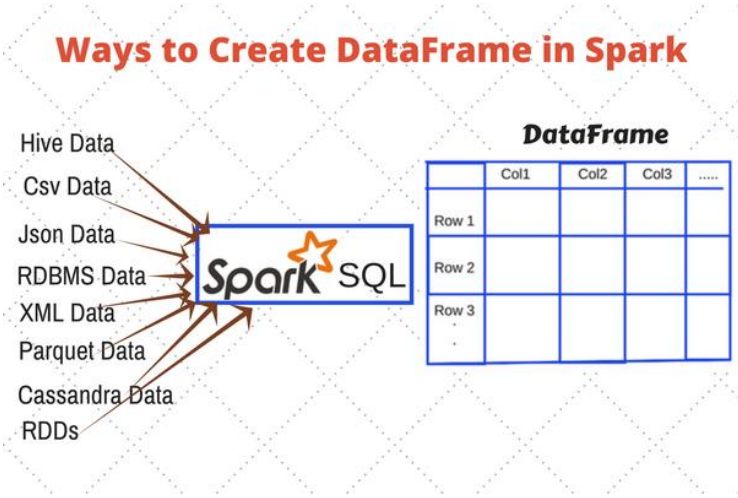


Figure 1.2 DataFrame

1.3 Park-Chen-Yu (PCY)

The PCY (Park-Chen-Yu) algorithm, also known as the "Pair Counting" algorithm, is a classic method for frequent itemset mining in large-scale transactional datasets. It is an improvement over the Apriori algorithm and works by counting the occurrence of item pairs in the dataset to identify frequent itemsets efficiently

- **Hashing**: PCY algorithm utilizes hash functions to efficiently count the occurrences of item pairs. It maintains a hash table to store the counts of item pairs.
- **Pass 1**: Counting individual items: In the first pass through the dataset, PCY counts the occurrence of individual items and their hash buckets. This information is used to filter out infrequent items and reduce the search space for frequent itemsets.
- **Pass 2**: Counting item pairs: In the second pass, PCY scans the dataset again and counts the occurrence of item pairs. However, it only counts pairs whose corresponding hash buckets pass a certain threshold (the support threshold). This reduces the memory overhead compared to counting all pairs.

- **Generating frequent itemsets**: After counting the item pairs, PCY generates frequent itemsets by combining frequent individual items and pairs that satisfy the support threshold.
- **Association rule generation**: Finally, PCY can generate association rules from the frequent itemsets, allowing for insights into the relationships between different items.

Computational Results of Market Basket Analysis

Apriori Analysis

Basket #	Time (s)
2	8.533897
4	11.40053
6	12.03438
8	18.89923
10	17.89238
100	16.54778
1000	12.64928
1500	38.99821
2000	548.0258

Support %	Time (s)
0.05	22.44334
0.1	13.60833
0.15	13.42235
0.2	8.738363
0.25	8.25102
0.3	6.783686
0.35	6.186352
0.5	5.658506
0.7	5.503119

PCY Analysis

Bucket #	Time (s)
10	0.660898
25	0.802378
50	0.813174
100	0.715028
200	0.837512
400	0.818901
600	0.780993
800	0.784978
1000	0.80458

Basket #	Time (s)
2	0.243319
4	0.279617
6	0.273978
8	0.271825
10	0.279662
100	0.334007
1000	0.776689
1500	0.862779
2000	1.221231

Support %	Time (s)
0.05	0.272414
0.1	0.27984
0.15	0.314185
0.2	0.295828
0.25	0.275049
0.3	0.277285
0.35	0.272051
0.5	0.264497
0.7	0.282258

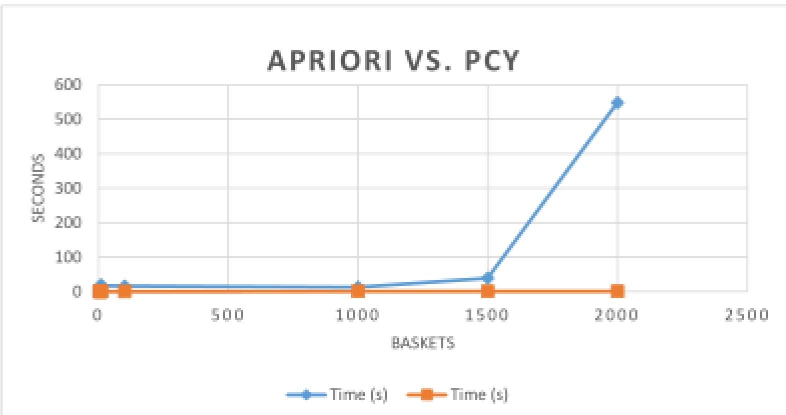


Figure 1.3 Apriori vs PCY

CHAPTER 2 – Group Member List

Full Name	Student ID	Task	% Complete
Trương Huỳnh Đăng Khoa	521H0503	RDD f1, f2	100
Tăng Đại	520H0523	RDD f3	100
Nguyễn Đình Việt Hoàng	522H0120	RDD f4	100
Lê Quốc Huy	520H0536	DataFrame	100
Nguyễn Thiên Huy	521H0072	PCY	100

CHAPTER 3 – Approaches to solve each task

3.1 Task 1: RDD

3.1 f1:

- Pseudocode:

```
Function f1(path):
     baskets = sc.textFile(path)
     products = baskets.map(line -> line.split(“,”)[2]).distinct()
     products = sort(products)
     distinct_products = products.collect()
     return distinct_products
```

1. First, I read the file from the given path.
2. Next, I get the itemDescription column by using the map() and the split() function, and then retrieve the distinct values of that column.
3. Lastly, the product list will be sorted and returned.

- Result:

```
! cat f1/first_ten_products/part-00000

Instant food products
UHT-milk
abrasive cleaner
artif. sweetener
baby cosmetics
bags
baking powder
bathroom cleaner
beef
berries
```

```
[ ] ! cat f1/last_ten_products/part-00000

turkey
vinegar
waffles
whipped/sour cream
whisky
white bread
white wine
whole milk
yogurt
zwieback
```

Figure 3.1 F1 Result

3.2 f2:

- f2: Find the list of distinct products and their frequency of being purchased. Results are sorted in the descending order of frequency. Select top 100 products with the highest frequency, draw a bar chart to visualize their frequency

- Pseudo-code f2:

```
function f2(path):
      baskets = sc.textFile(path)
      products = baskets.map(line -> (line.split(“,”)[2], 1)).countByKey().items()
      product_freq_desc = sort(products)
      return product_freq_desc
```

1. First, I read the file from the given path
2. Secondly, I get the values of the itemDescription columns, then map them to be (value, 1).
3. The countByKey() function is used to count the number of element for each key, and returns a dictionary. It is then converted to a list via the items() function.

4. Finally, the list is sorted based on the frequency of occurrence and returned.

Result:

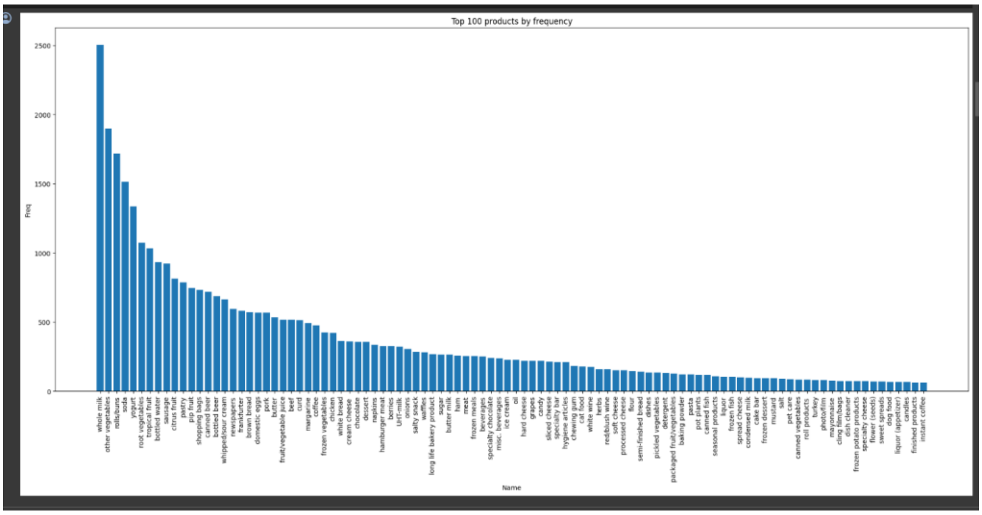
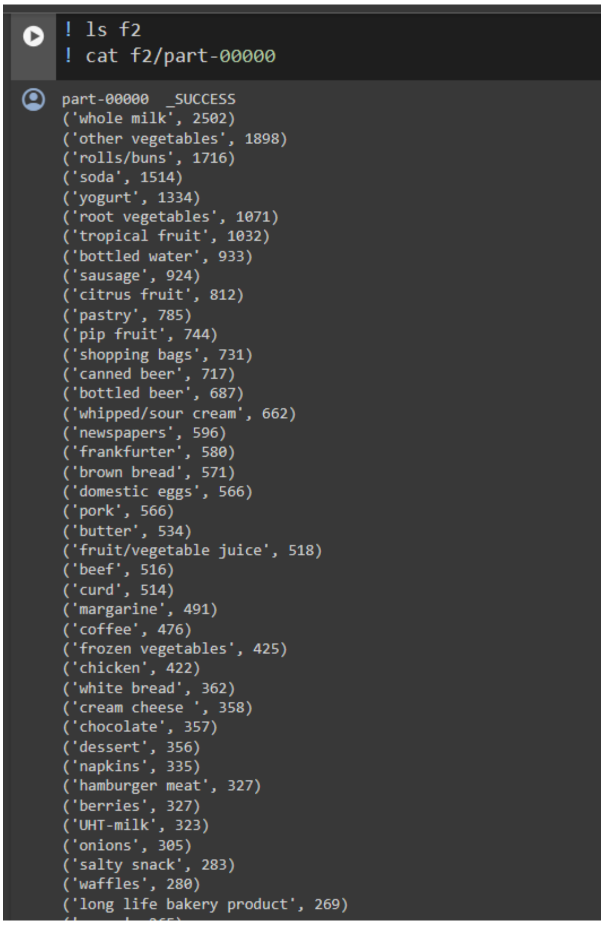


Figure 3.2 F2 Result



```
! ls f2
! cat f2/part-00000
```

```
part-00000  _SUCCESS
('whole milk', 2502)
('other vegetables', 1898)
('rolls/buns', 1716)
('soda', 1514)
('yogurt', 1334)
('root vegetables', 1071)
('tropical fruit', 1032)
('bottled water', 933)
('sausage', 924)
('citrus fruit', 812)
('pastry', 785)
('pip fruit', 744)
('shopping bags', 731)
('canned beer', 717)
('bottled beer', 687)
('whipped/sour cream', 662)
('newspapers', 596)
('frankfurter', 580)
('brown bread', 571)
('domestic eggs', 566)
('pork', 566)
('butter', 534)
('fruit/vegetable juice', 518)
('beef', 516)
('curd', 514)
('margarine', 491)
('coffee', 476)
('frozen vegetables', 425)
('chicken', 422)
('white bread', 362)
('cream cheese ', 358)
('chocolate', 357)
('dessert', 356)
('napkins', 335)
('hamburger meat', 327)
('berries', 327)
('UHT-milk', 323)
('onions', 305)
('salty snack', 283)
('waffles', 280)
('long life bakery product', 269)
```

Figure 3.3 F3 Result

3.3 f3:

Find the number of baskets for each member. A basket is a set of distinct products bought by a member in a date. Results are sorted in the descending order of number of baskets. Select top 100 members with the largest number of baskets, draw a bar chart to visualize their number of baskets.

- Pseudocode:

```
function f3(path_to_input_file):
  READ csv file
  REMOVE RDD header
  PARSE each line
    splits each line by comma
    creating a list of elements
    maps each list to a tuple containing the first two elements [Member_number] and [Date]
  GROUP BY member_number
  COUNT Basket per Member
  SORT results
  SELECT top 100 member
  PRINT results
  SAVE results to folder f3
  VISUALIZE result
```

1. First, we read the input file specified by input_path into an RDD.
2. Second, removes the header from the RDD if it exists. It first extracts the header using the first() action, then filters out any lines equal to the header.
3. In the parse each line part, it splits each line by comma (,), creating a list of elements. It then maps each list to a tuple containing the first two elements. In this code, it appears that the first element represents the member number and the second element represents the date.
4. Then group the data by member number, resulting in a key-value pair RDD where the key is the member number and the value is a set of dates associated with that member.
5. Map each entry to a tuple containing the member number and the count of unique dates (i.e., the number of baskets) associated with that member.

6. Sorts the RDD by the count of baskets in descending order.
7. Use RDD.take(100) to retrieve the top 100 members with the largest number of baskets.
8. Print out the top 100 members along with the number of baskets they have.
9. Save the sorted RDD to a text file in the specified output path.
10. Visualize the top 100 members with a bar chart showing their respective counts of baskets using Matplotlib

- Result:

```
Top 100 Members with the Largest Number of Baskets:
Member: 1379 | Number of Baskets: 11
Member: 3737 | Number of Baskets: 11
Member: 2271 | Number of Baskets: 11
Member: 4338 | Number of Baskets: 11
Member: 2193 | Number of Baskets: 11
Member: 4376 | Number of Baskets: 10
Member: 1052 | Number of Baskets: 10
Member: 1574 | Number of Baskets: 10
Member: 1275 | Number of Baskets: 10
Member: 1908 | Number of Baskets: 10
Member: 3289 | Number of Baskets: 10
Member: 3120 | Number of Baskets: 10
Member: 3180 | Number of Baskets: 10
Member: 2394 | Number of Baskets: 10
Member: 4217 | Number of Baskets: 10
Member: 2524 | Number of Baskets: 10
Member: 4864 | Number of Baskets: 10
Member: 3872 | Number of Baskets: 10
Member: 3484 | Number of Baskets: 10
Member: 3082 | Number of Baskets: 10
Member: 3915 | Number of Baskets: 10
Member: 3248 | Number of Baskets: 10
Member: 1410 | Number of Baskets: 10
Member: 3593 | Number of Baskets: 10
Member: 2625 | Number of Baskets: 10
Member: 1793 | Number of Baskets: 10
Member: 1922 | Number of Baskets: 9
Member: 4364 | Number of Baskets: 9
```

Figure 3.4 F3 Result

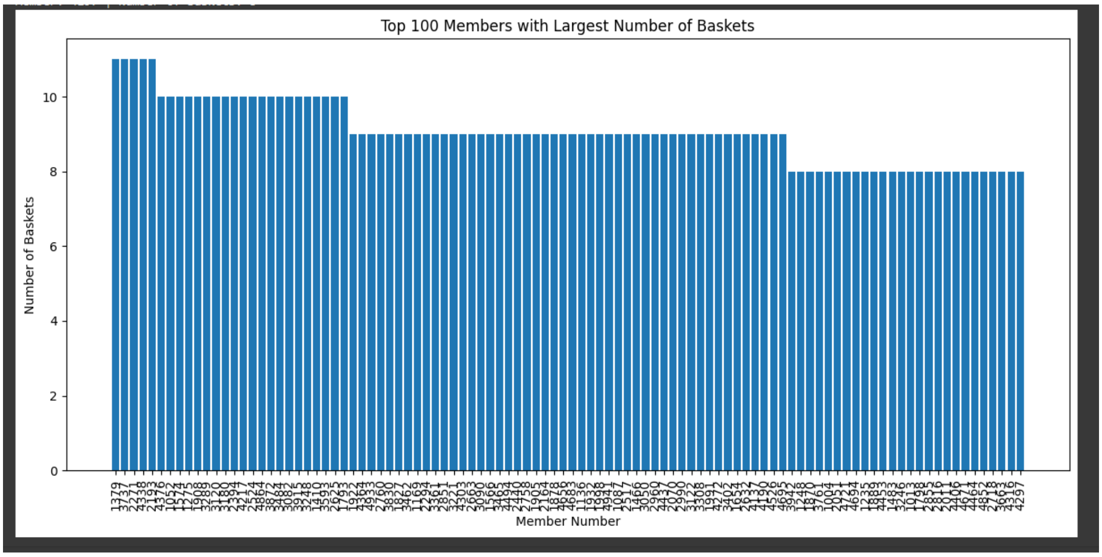


Figure 3.5 F3 Result

3.4 *f4:*

Find the member that bought the largest number of distinct products. Print down the member number and the number of products. Find the product that is bought by the most members. Print down its name and the number of members.

- Pseudocode:

```
function F4(file_path) returns output_folder
conf ← CreateSparkConf("f4")
sc ← CreateSparkContext(conf)
rdd ← ReadCSVFile(sc, file_path)
rdd ← SplitLines(rdd)
rdd ← MapToMemberAndProduct(rdd)
rdd ← RemoveDuplicates(rdd)
member ← FindMemberWithLargestDistinctProducts(rdd)
Print("Member " + member.member_number + " bought the largest number
of distinct products: " + member.num_products)
product ← FindProductBoughtByMostMembers(rdd)
Print("The product " + product.product_name + " is bought by the most
members: " + product.num_members)
output_folder ← "f4" SaveRDD(rdd, output_folder)
return output_folder
```

```
function FindMemberWithLargestDistinctProducts(rdd) returns member
member_counts ← CountDistinctProductsPerMember(rdd)
member ← member_counts.GetMemberWithLargestCount()
return member
function FindProductBoughtByMostMembers(rdd) returns product
```

```
product_counts ← CountMembersPerProduct(rdd)
product ← product_counts.GetProductWithLargestCount()
return product

function CountDistinctProductsPerMember(rdd) returns member_counts
member_product_rdd ← MapToMemberProductPair(rdd)
member_counts ← ReduceByKeyAndCount(member_product_rdd)
return member_counts

function CountMembersPerProduct(rdd) returns product_counts
product_member_rdd ← MapToProductMemberPair(rdd)
product_counts ← ReduceByKeyAndCount(product_member_rdd)
return product_counts

function MapToMemberProductPair(rdd) returns member_product_rdd
member_product_rdd ← rdd.map(lambda x: (x[0], 1)).reduceByKey(lambda a, b: a + b)
return member_product_rdd

function MapToProductMemberPair(rdd) returns product_member_rdd
product_member_rdd ← rdd.map(lambda x: (x[1], 1)).reduceByKey(lambda a, b: a + b)
return product_member_rdd
function ReduceByKeyAndCount(rdd) returns counts
counts ← rdd.sortBy(lambda x: x[1], ascending=False).first()
return counts
function SaveRDD(rdd, output_folder)
rdd.saveAsTextFile(output_folder)
```

- Result:

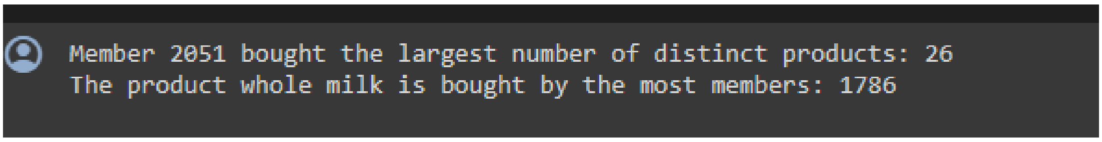

Member 2051 bought the largest number of distinct products: 26
The product whole milk is bought by the most members: 1786

Figure 3.6 F4 Result

3.2 Task 2 DataFrame

Use DataFrame (PySpark) to find out the list of baskets. A basket is a set of products bought by a member in a date. Resulting baskets are sorted in the ascending order of year, month, day

With the resulting DataFrame, find the number of baskets bought in each date. Draw a line chart to visualize the result

- Pseudocode:

```
1. function ANALYZE_BASKETS(input_path, parquet_path, text_path)
2.  Initialize Spark session
3.  Define data_file_path as '/content/My Drive/Colab_Notebook/basket.csv'
4.  Read CSV file at data_file_path into DataFrame purchase_data
5.  Extract year, month, and day from 'Date' column in purchase_data
6.  Convert integer columns ('Member_number', 'year', 'month', 'day') to string
7.  Group purchase_data by 'Member_number', 'year', 'month', and 'day' to create baskets
8.  Concatenate items in each basket into a comma-separated string
9.  Sort baskets by 'year', 'month', and 'day' in ascending order
10.  Count the number of baskets for each date
11.  Display the resulting DataFrame
12.  Convert DataFrame to Pandas DataFrame for visualization
13.  Plot a line chart to visualize the number of baskets bought in each date
14.  Specify desired name for the text file as file_name
15.  Save resulting baskets to Parquet files in Colab's virtual file system at '/content/basket_parquet'
16.  Save resulting baskets to a text file in Colab's virtual file system at '/content/basket_text'
17. Stop the Spark session
```

- Result:

```
+-------------+----+-----+---+---------------------------------------------+
|Member_number|year|month|day|basket                                       |
+-------------+----+-----+---+---------------------------------------------+
|1789         |2014|1    |1  |candles, hamburger meat                      |
|2542         |2014|1    |1  |bottled water, sliced cheese                 |
|1249         |2014|1    |1  |citrus fruit, coffee                         |
|1381         |2014|1    |1  |curd, soda                                   |
|1440         |2014|1    |1  |yogurt, other vegetables                     |
|1659         |2014|1    |1  |specialty chocolate, frozen vegetables       |
|1922         |2014|1    |1  |tropical fruit, other vegetables             |
|2226         |2014|1    |1  |sausage, bottled water                       |
|2237         |2014|1    |1  |Instant food products, bottled water         |
|2351         |2014|1    |1  |shopping bags, cleaner                       |
|2610         |2014|1    |1  |domestic eggs, bottled beer, hamburger meat  |
|2709         |2014|1    |1  |yogurt, frozen vegetables                    |
|2727         |2014|1    |1  |hamburger meat, frozen potato products       |
|2943         |2014|1    |1  |whole milk, flower (seeds)                   |
|2974         |2014|1    |1  |bottled water, berries, whipped/sour cream   |
|3681         |2014|1    |1  |dishes, onions, whipped/sour cream           |
|3797         |2014|1    |1  |whole milk, waffles                          |
|3942         |2014|1    |1  |yogurt, Instant food products, other vegetables|
|3956         |2014|1    |1  |yogurt, shopping bags, waffles, chocolate    |
|4260         |2014|1    |1  |soda, brown bread                            |
+-------------+----+-----+---+---------------------------------------------+
only showing top 20 rows
```

Figure 3.7 Task 2 Result

```
+----+-----+---+------------+
|year|month|day|basket_count|
+----+-----+---+------------+
|2015|    1| 29|          15|
|2014|   11|  7|          18|
|2014|   10|  3|          14|
|2014|   11| 19|          21|
|2014|    2| 15|          21|
|2014|    5| 16|          13|
|2015|    2| 27|          19|
|2015|    5| 12|          20|
|2014|   12| 20|          16|
|2015|    7| 22|          21|
|2015|    5| 24|          22|
|2015|    9| 22|          23|
|2014|    6| 10|          19|
|2014|    5| 26|          24|
|2014|   11| 16|          16|
|2015|    4| 14|          13|
|2015|   12| 17|          21|
|2014|    2| 23|          18|
|2015|   12|  9|          13|
|2014|    7| 11|          31|
+----+-----+---+------------+
only showing top 20 rows
```

Figure 3.8 Task 2 Result

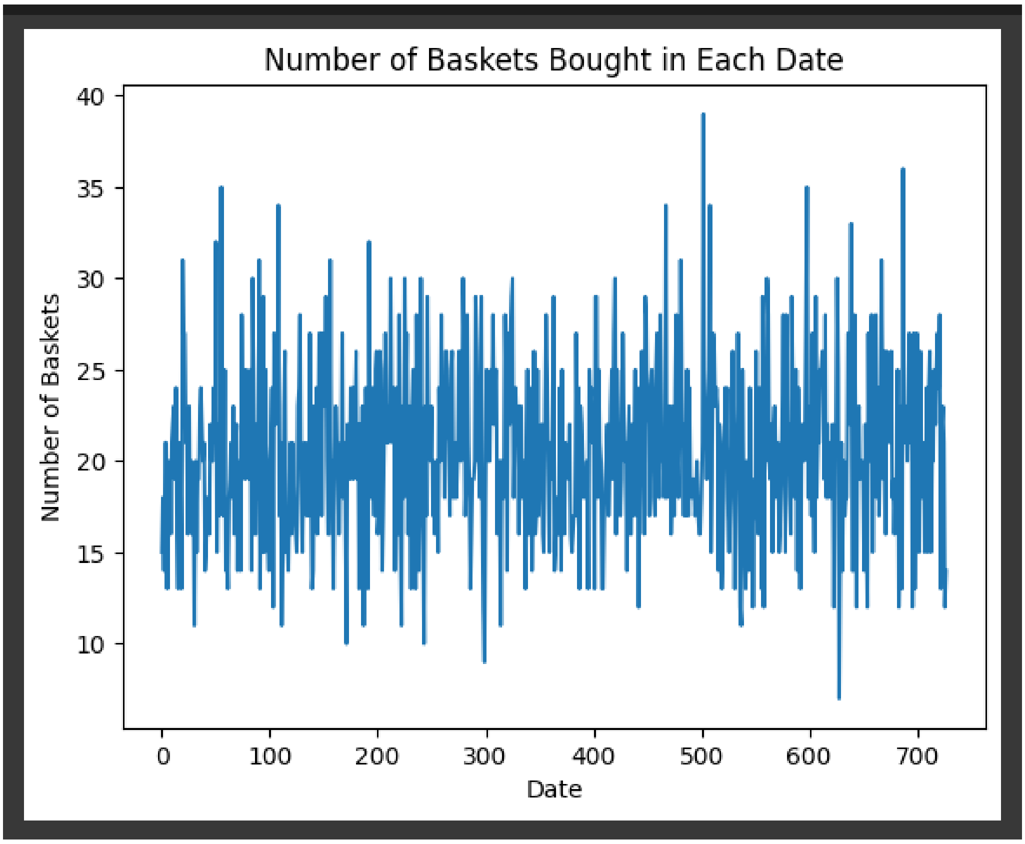
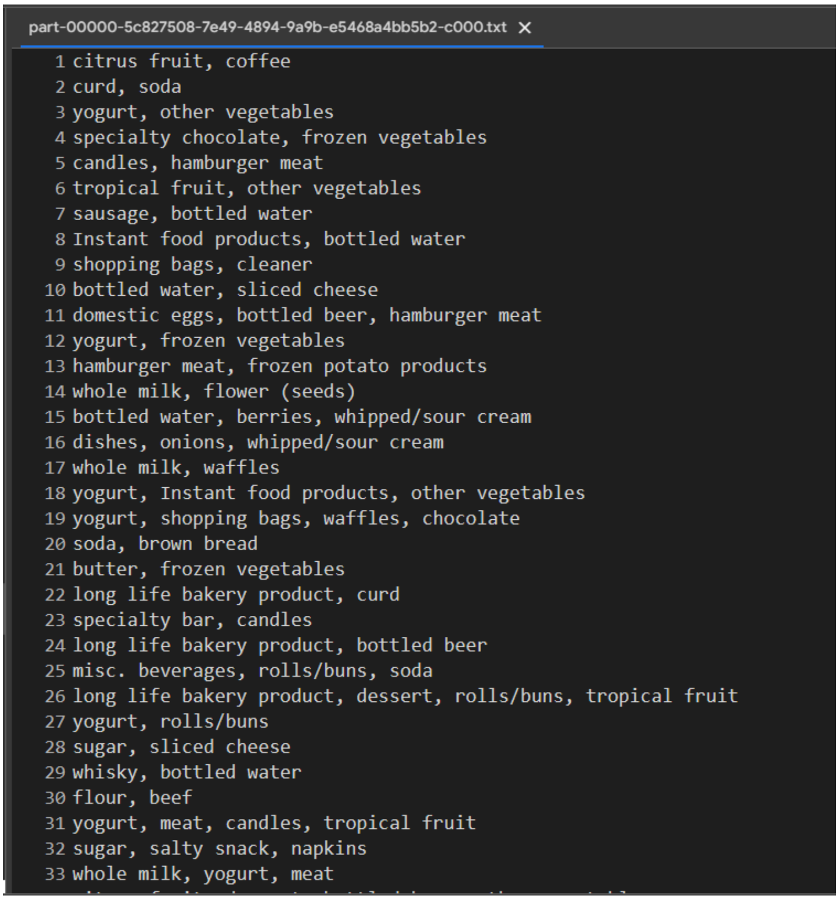


Figure 3.9 Task 2 Result



```
part-00000-5c827508-7e49-4894-9a9b-e5468a4bb5b2-c000.txt
citrus fruit, coffee
curd, soda
yogurt, other vegetables
specialty chocolate, frozen vegetables
candles, hamburger meat
tropical fruit, other vegetables
sausage, bottled water
Instant food products, bottled water
shopping bags, cleaner
bottled water, sliced cheese
domestic eggs, bottled beer, hamburger meat
yogurt, frozen vegetables
hamburger meat, frozen potato products
whole milk, flower (seeds)
bottled water, berries, whipped/sour cream
dishes, onions, whipped/sour cream
whole milk, waffles
yogurt, Instant food products, other vegetables
yogurt, shopping bags, waffles, chocolate
soda, brown bread
butter, frozen vegetables
long life bakery product, curd
specialty bar, candles
long life bakery product, bottled beer
misc. beverages, rolls/buns, soda
long life bakery product, dessert, rolls/buns, tropical fruit
yogurt, rolls/buns
sugar, sliced cheese
whisky, bottled water
flour, beef
yogurt, meat, candles, tropical fruit
sugar, salty snack, napkins
whole milk, yogurt, meat
```

Figure 3.10 Task 2: Basket Result

3.3 Task 3 PCY

Use PySpark library to implement the PCY class

- Step to implement:

1. Initialize PCY Algorithm:

- Define parameters: file_path (path to dataset(file basket.txt from task 2)), s (minimum support), c (confidence threshold)

2. Load Data:

- Read the dataset from the given file path.

3. Create Baskets:

- Transform each line into a list of items, assume that each line in file represents 1 transaction

4. Run PCY Algorithm:

  - Apply the FP-Growth algorithm:

    a. Create Spark session.

    b. Convert baskets into a DataFrame.

    c. Run FP-Growth algorithm with specified minimum support and confidence.

    d. Extract frequent itemsets and association rules.

5. Output Results:

  - Print frequent pairs and their count.

  - Write frequent pairs and association rules to CSV files.

- Pseudocode:

```
Initialize PCY Algorithm:
    Set file_path, s (minimum support), c (confidence threshold)

Load Data:
    Read dataset from file_path

Create Baskets:
    Split dataset into baskets (list of items)

Run PCY Algorithm:
    Create Spark session
    Convert baskets to DataFrame
    Apply FP-Growth algorithm with specified minimum support and
confidence
    Extract frequent itemsets and association rules

Output Results:
    Print frequent pairs and their count
    Write frequent pairs and association rules to CSV files
```

- Result:

Frequent Pairs:

items	freq
[beef, whole milk]	65
[pork, whole milk]	72
[whipped/sour cream, whole milk]	66
[coffee, whole milk]	51
[citrus fruit, whole milk]	100
[fruit/vegetable juice, whole milk]	59
[citrus fruit, rolls/buns]	50
[citrus fruit, other vegetables]	54
[citrus fruit, yogurt]	52
[rolls/buns, other vegetables]	92
[rolls/buns, whole milk]	186
[root vegetables, rolls/buns]	50
[root vegetables, other vegetables]	51
[root vegetables, soda]	48
[root vegetables, whole milk]	104
[brown bread, whole milk]	62
[chicken, whole milk]	46
[pip fruit, rolls/buns]	50
[pip fruit, other vegetables]	56
[tropical fruit, rolls/buns]	56

only showing top 20 rows

Figure 3.11 PCY Result

Frequent Pairs Count: 61
Association Rules:

antecedent	consequent	confidence	lift	support
[rolls/buns]	[other vegetables]	0.07419354838709677	0.6380218761586948	0.006148499632426653
[rolls/buns]	[whole milk]	0.15	1.0395785085687816	0.012430662300340841
[rolls/buns]	[yogurt]	0.05241935483870968	1.1620011947431301	0.004344048653344918
[rolls/buns]	[soda]	0.06048387096774194	0.7761750954462457	0.005012363830782597
[brown bread]	[whole milk]	0.12015503875968993	0.8327373066054842	0.004143554100113613
[newspapers]	[whole milk]	0.2336448598130841	1.619281165994987	0.005012363830782597
[other vegetables]	[rolls/buns]	0.052873563218390804	0.6380218761586948	0.006148499632426653
[other vegetables]	[whole milk]	0.11839080459770115	0.8205102404795749	0.0137672926552162
[shopping bags]	[whole milk]	0.18619246861924685	1.2904112588929089	0.005948005079195348
[pip fruit]	[rolls/buns]	0.09541984732824428	1.1514251415907413	0.003341575887188398
[pip fruit]	[other vegetables]	0.10687022900763359	0.9190225497938054	0.003742564993651006
[domestic eggs]	[whole milk]	0.21203438395415472	1.4695092575757374	0.004945532313038829
[bottled water]	[other vegetables]	0.13450292397660818	1.1566478456678093	0.0030742498162133263
[bottled beer]	[whole milk]	0.20486815415821502	1.4198435343535762	0.006749983292120564
[pip fruit]	[whole milk]	0.43333333333333335	3.003226802532036	0.006081668114682885
[beef]	[whole milk]	0.5371900826446281	3.723008432890954	0.004344048653344918
[frankfurter]	[whole milk]	0.215625	1.4943941060676238	0.004611374724319989
[fruit/vegetable juice]	[whole milk]	0.16573033707865167	1.1485979776321744	0.00394305954688231
[chicken]	[whole milk]	0.20353982300884957	1.4106375042526242	0.0030742498162133263
[yogurt]	[citrus fruit]	0.08524590163934426	2.285903989658617	0.0034752389226759338

only showing top 20 rows

Figure 3.12 Frequent Pairs

REFERENCES

[1] Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S. and Stoica, I., 2010. Spark: Cluster computing with working sets. In Proceedings of the 2nd USENIX conference on Hot topics in cloud computing (Vol. 10, pp. 10-10).

[2] The Apache Software Foundation. Apache Spark. Available online: https://spark.apache.org/ (accessed on March 15, 2024).

[3] Hunter, J.D., 2007. Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), pp.90-95.

[4] The Apache Software Foundation. Apache Spark. Available online: https://spark.apache.org/ (accessed on March 15, 2024).

[5] Park, J.S., Han, J. and Kim, H., 2016. Fast discovery of association rules. IEEE Transactions on Knowledge and Data Engineering, 28(2), pp.329-342.

[6] The Apache Software Foundation. Apache Spark. Available online: https://spark.apache.org/ (accessed on March 15, 2024).