Uni

BERLIN SC

Assignment Title:-Scalable Big Data Solutions with Hadoop Architecture and Map Reduce Abstraction.

Assignment Type:- Report.

Programme title: Big Data Analytics.

Name: Jinesh Kumar Kanakmal Jain.

Year: 2026

CONTENTS

Table of Contents.

Index of Figures

Statement of compliance with academic ethics and the avoidance of plagiarism

I honestly declare that this dissertation is entirely my own work and none of its part has been copied from printed or electronic sources, translated from foreign sources and reproduced from essays of other researchers or students. Wherever I have been based on ideas or other people texts I clearly declare it through the good use of references following academic ethics.

(In the case that is proved that part of the essay does not constitute an original work, but a copy of an already published essay or from another source, the student will be expelled permanently from the program).

Name and Surname (Capital letters):

Jinesh Kumar Kanakmal Jain

Date: 07/02/2026

INTRODUCTION

This report explores scalable retail analytics with the Hadoop ecosystem and MapReduce abstraction in response to the business requirement of determining the most selling products by month and aiding in demand driven decision-making. The project is deployed on a single-node, pseudo-distributed Hadoop cluster, which is suited to illustrate the main concepts of distributed storage and processing and is manageable within a manageable academic context.

The scope includes end-to-end handling of a large transactional dataset:

(i). Defining the business problem and motivating the use of Big Data technologies

(ii). Configuring key Hadoop components (NameNode, DataNode, Secondary NameNode) and ingesting data into HDFS with evidence of block-level distribution

(iii). Performing distributed aggregation using Hadoop Streaming on YARN with Mapper/Reducer logic being documented and verified results, and

(iv). Creating a machine learning pipeline to classify demand, featuring engineering, metrics of evaluation, and business-oriented visualisation.

The report is organised by Tasks 1–4, followed by a concluding discussion of findings, limitations, and future improvements.

Task 1:- Problem Definition and Business Context.

1. **Business problem statement:-**

Retail organisations need information about the performance of products promptly and in real-time to maximise inventory, pricing and promotional choices. The business objective that is addressed by this project is to determine the top selling products by the month and forecast the demand within a month with large scale transactional data. The analytical task could be described as demand intelligence workflow: the first stage will be aggregation of transactions to measure demand of products in a month (e.g. total units sold and total spend), and subsequently predict or classify demand using the historical patterns. The key stakeholders are the operations teams, where they need stable demand indicators in making store-level fulfilment and logistics plans. Inventory and replenishment managers who are required to reduce stockouts and overstocks by matchng orders with the expected demand; and marketing teams, who can utilise top-seller and demand indicators to craft specific promotions, product bundling, and seasonal campaigns. The desired result is the scalable pipeline that can generate consistent monthly demand summaries and predictive signals that can be used by managers in making managerial decisions

2. **Why Big Data is needed**

Retail datasets at transaction level are highly voluminous and updated regularly, which in many cases can be hundreds of megabytes up to several gigabytes even in short periods of time. This large scale encourages the application of Big Data technologies that assist in distributed storage and parallel computerization. HDFS is a fault-tolerant storage mechanism of Hadoop, which divides big files into fixed-size blocks (e.g. 128 MB) and distributes them over nodes, and MapReduce/YARN is a scalable processing framework that allows the processing of huge data by distributing tasks and scheduling the resources across nodes. Conversely, a single-machine solution is limited by the memory space, slower throughput on the transfer of multi-hundred-million-row files, and less tolerant to failure (e.g. termination of a process invalidates long-running calculations). Moreover, there is a lot of shuffling and aggregation involved in processing operations like grouping based on time period or product code which may easily bottle neck on a single environment when the data surpasses the available RAM or disk I/O capabilities. The reason why the workflow implemented using Hadoop is, then, not just justified by the size of the datasets, but by the desire to show scalable architectures that are typical of enterprise analytics: distributed storage, parallel processing and reproducible batch pipelines that can be scaled as the size of the data grows.

3. Dataset selection and justification

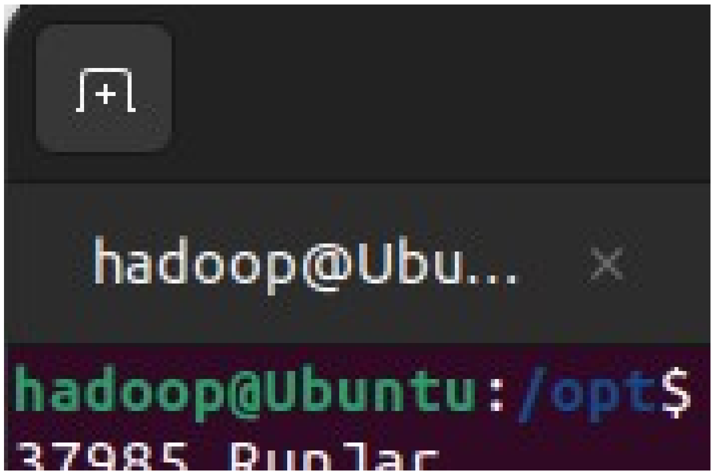
A subset of the dunnhumby The Complete Journey retail data (Part 1) is utilised in the project and it includes the basket level records of the transaction, adequate to estimate demand and analyse the product performance. A subset of the entire dataset was used because the complete dataset is too large but the need to create a number of HDFS blocks was also necessary. In particular, two transaction files, transactions_200618.csv (size of approximately 366 MB), and transaction files, transactions_200619.csv (size of approximately 354 MB) were transferred safely to the Hadoop node and were joined together to form one large file to be ingested and processed uniformly. The resulting consolidated data is about 700 + MB, which is sufficient to meet the minimum threshold of 3-4 blocks of HDFS at 128 MB block and therefore can enable a meaningful show of distributed processing. The transaction schema incorporates variables that were necessary to operationalise the business problem: SHOP_DATE (to derive months and aggregate of time), PROD_CODE (product identifier), QUANTITY (volume of demands) and SPEND (sale proxy). This data sample choice hence does not interfere with the analytical purposes but is computationally feasible in a pseudo-distributed research school setup.

Task 2:- Environment Setup and Data Ingestion

1. Cluster design and environment

The project environment was implemented as a single-node, pseudo-distributed Hadoop cluster running on an Ubuntu virtual machine. This setup works well to showcase the Hadoop ecosystem in end to end mode i.e. HDFS storage, YARN resource management and MapReduce execution but it is feasible even with limited compute resources. In pseudo-distributed mode Hadoop daemons execute independent processes within the same host, however, the framework effectively acts as a distributed system to the user,such as block management, job submission and container scheduling. The method meets the assignment criteria to set up core Hadoop components and load data into HDFS and demonstrates the block-level storage (Ma, Zhao, and Zhao, 2023).

Hadoop was set up under a specific Linux user account, with Java being setup under JAVA_HOME and Hadoop binaries being set under HADOOP_HOME and PATH. With the same user, passwordless SSH was also activated (through ssh-keygen and authorized-keys in the home directory) since Hadoop management scripts use SSH to coordinate the launching of daemons even when a single node is being deployed. Configurative base files were specified in core-site.xml (e.g. fs.defaultFS), hdfs-site.xml (location of storage directories and replication), mapred-site.xml (MapReduce on YARN), and yarn-site.xml (NodeManager shuffle service), and then an initial hdfs namenode -format was run to initialise the metadata store. Services were initiated with start-dfs.sh and start-yarn.sh and environment was checked with process checks (jps) and service level checks through Name Node and Resource Manager web interfaces attached to localhost to monitor and capture evidences. In general, the environment is focused on functional completeness, auditability, and reproducibility which are paramount to applied big data engineering.



```
hadoop@Ubuntu:/opt$
37985 RunJar
```

Figure 1: jps output showing active Hadoop daemons (HDFS and YARN).

2. Hadoop components configured (HDFS + YARN)

HDFS had been configured with the three necessitated daemons, NameNode, DataNode and SecondaryNameNode. NameNode is the metadata of the filesystem, containing directory hierarchy, file permissions, and file to block mapping, but not user data. Data blocks are actually stored on the DataNode and this node receives read and write requests and periodically reports the status information to the NameNode. The SecondaryNameNode aids the NameNode in doing periodical metadata checkpointing so as to enhance recovery and restart behaviour.

YARN was facilitated to facilitate distributed computation. The ResourceManager is the scheduler and resource controller in the entire cluster and the NodeManager controls the execution of containers on the node. This isolation enables the architectural separation of storage (HDFS) and compute (YARN). To determine a valid baseline in the future processing tasks, all the necessary daemons were checked as active at the same time using jps (Li and Hei, 2025).

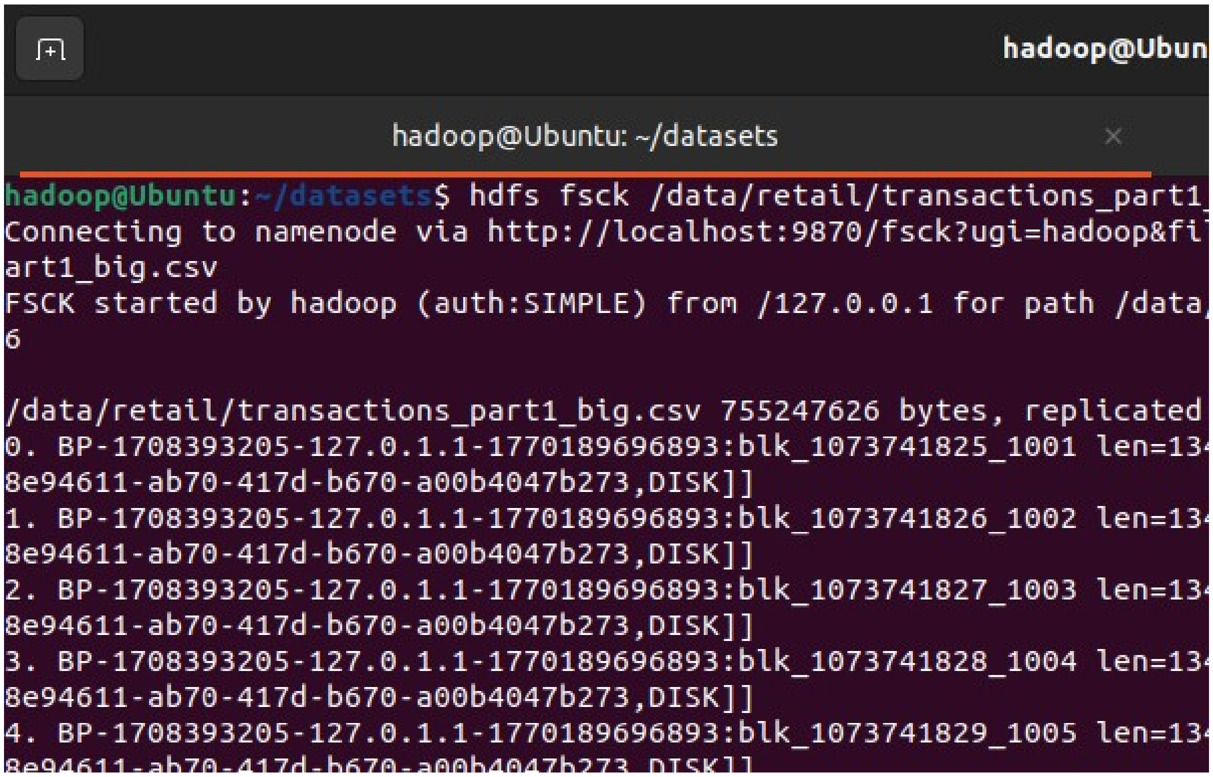
3. FSImage, EditLog, and checkpointing

The NameNode is a persistent metadata structure that holds the metadata in two structures, namely FSImage and EditLog. The FSImage contains a full snapshot of filesystem namespace at a point in time whereas the EditLog contains a record of the changes that occur after the snapshot. This design is capable of rapid metadata updates but may have significantly long recovery times in case of a large EditLog (Qi and Liu, 2024).

Checkpointing helps to counter this risk. The FSImage and EditLog are regularly combined to a new, consolidated FSImage by the SecondaryNameNode and makes recovery overheads and maintenance more manageable. Notably, the SecondaryNameNode is not a standby NameNode. It is there to control the metadata expansion and enhance trustworthiness. This is a distinction that is brought out clearly to show the correct knowledge of HDFS internals.

4. Data ingestion steps and HDFS layout

The data ingestion was done via HDFS command-line interface with well-defined directory structure: /data/retail/ containing raw data that could not be changed and /out/ containing derived data. To begin with, the necessary directories were made with the help of hdfs dfs -mkdir -p /data/retail and hdfs dfs -mkdir -p /out. The two big monthly transaction files of the subset of selected data were copied with the help of secure copy (scp) between the host machine and the Ubuntu VM, and locally combined into a single large file (transactions_part1_big.csv) to simplify further processing. The consolidated file was added to HDFS with the command hdfs dfs -put transactions_part1_big.csv /data/retail/ so that the file could be stored with a distributed format allowing further access by subsequent MapReduce tasks.



```
hadoop@Ubuntu:~/datasets$ hdfs fsck /data/retail/transactions_part1_
Connecting to namenode via http://localhost:9870/fsck?ugi=hadoop&fi
art1_big.csv
FSCK started by hadoop (auth:SIMPLE) from /127.0.0.1 for path /data
6

/data/retail/transactions_part1_big.csv 755247626 bytes, replicated
0. BP-1708393205-127.0.1.1-1770189696893:blk_1073741825_1001 len=13
8e94611-ab70-417d-b670-a00b4047b273,DISK]]
1. BP-1708393205-127.0.1.1-1770189696893:blk_1073741826_1002 len=13
8e94611-ab70-417d-b670-a00b4047b273,DISK]]
2. BP-1708393205-127.0.1.1-1770189696893:blk_1073741827_1003 len=13
8e94611-ab70-417d-b670-a00b4047b273,DISK]]
3. BP-1708393205-127.0.1.1-1770189696893:blk_1073741828_1004 len=13
8e94611-ab70-417d-b670-a00b4047b273,DISK]]
4. BP-1708393205-127.0.1.1-1770189696893:blk_1073741829_1005 len=13
8e94611-ab70-417d-b670-a00b4047b273,DISK]]
```

Figure 2: hdfs fsck output confirming healthy HDFS status and block distribution for the uploaded dataset

The combined transaction dataset stored in HDFS (transactions_part1_big.csv) had a size of 755,247,626 bytes. The default options of an HDFS block size (128 MB) and replication (1) meant that the file was split into six blocks (five complete blocks and one partial block). The file was checked with hdfs fsck /data/retail/transactions_part1_big.csv -files -blocks -locations and it stated that the file was replicated and the filesystem had the status of HEALTHY. Even though replication was intrinsically restricted in this environment, the block report still offers the evidence of the basic HDFS mechanism which provides the scalable storage and parallel processing. Detailed command outputs and screen shots should be put in an appendix, with the remainder of the report summarising the important measures of size (file size), replication, and block count) and their consequences to future distributed computation.

Task 3– Distributed Processing (MapReduce abstraction) & Comparison

1. Processing objective and output definition

Task 3 operationalises the business problem as it carries out a distributed aggregation that generates monthly product demand aggregations based on the retail transactions data stored in HDFS. This is aimed at summarising demand on a granularity that is actionable directly on inventory planning and promotion analysis. The categories of the output records are (month, product) and include the sum of the amount sold and the sum of money that a customer has spent on that (month, product). As a matter of operations, operationally, month is calculated as SHOP_DATE truncated to YYYYMM (e.g. 200606) and product-id is calculated as PROD-CODE and measures are calculated as SUM(QUANTITY) and SUM(SPEND). The resulting output schema is:

- month (YYYYMM)
- product_id (PROD_CODE)
- total_quantity (sum of QUANTITY)
- total_spend (sum of SPEND)

This output is persisted to HDFS under /out/monthly_product_totals_yarn/ and becomes the “clean” analytical dataset used later for modelling and visualisation in Task 4.

2. MapReduce Streaming design

The distributed computation was implemented using Apache Hadoop Streaming, which allows Python mapper/reducer scripts to be executed as MapReduce tasks on YARN. The design follows the standard MapReduce pattern:

Mapper responsibilities.

The mapper parses each CSV line, extracts SHOP_DATE, PROD_CODE, QUANTITY, and SPEND, and derives month = SHOP_DATE[:6]. It then emits a composite key (month, product_id) and a value (quantity, spend). The header row is also filtered by the mapper and in this case the counters indicate that there is one less map output record compared with the map input records.

Shuffle/sort and grouping.

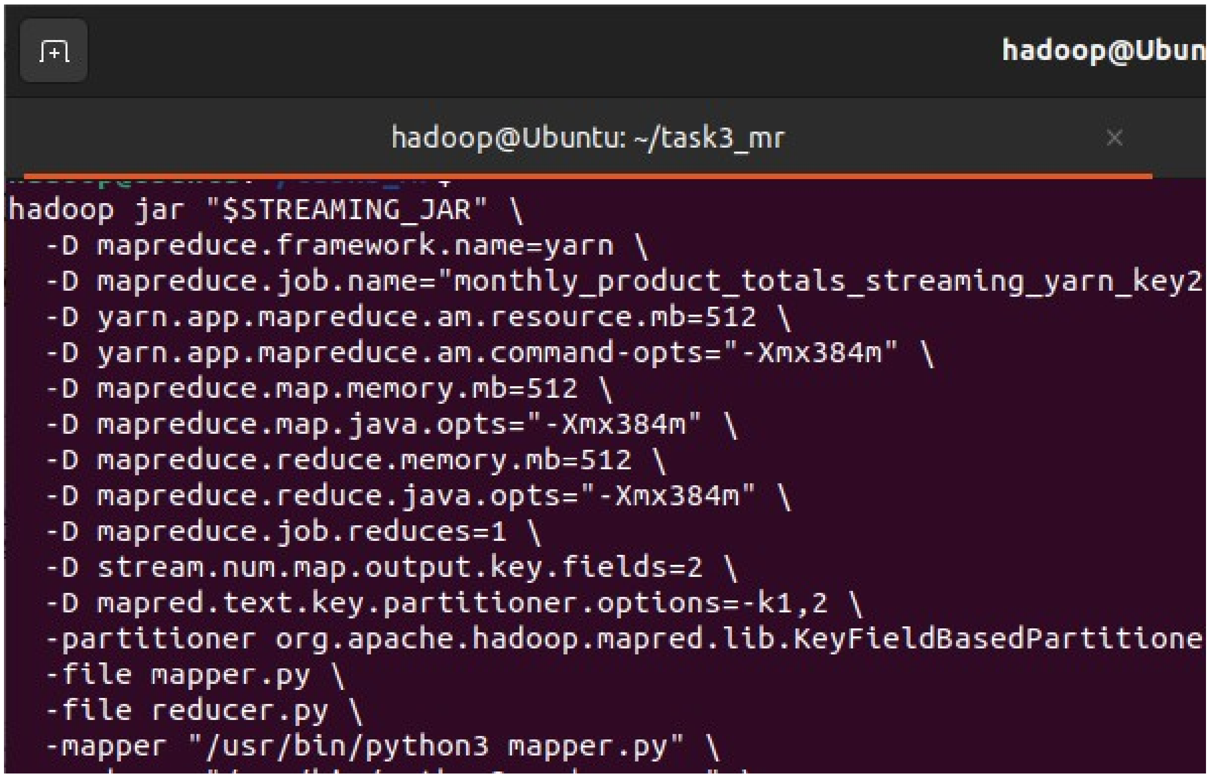
The MapReduce system classifies the output of the mapper and collects all values that have the same key. To have proper partitioning/grouping of the first and second fields of the composite key, job configuration is set to stream.num.map.output.key.fields=2 and partitioner is KeyFieldBasedPartitioner that is set with -k1,2. This imposes the desired semantics that all records of the same (month, product), are collected and subsequently reduced (Adamov, 2023).

Reducer responsibilities.

The reducer is fed with grouped records (by (month, productid)) and a simple associative aggregation: it adds quantities and spend of all of the values in the group to produce one consolidated output record per key. Ideally, the phase adopts the distributed equivalent to a SQL GROUP BY month, product id with two aggregations.

3. Submitting to YARN and monitoring evidence

The job was submitted to YARN using the Hadoop Streaming JAR. The exact submission command used is shown below**:**



hadoop@Ubun

hadoop@Ubuntu: ~/task3_mr

```
hadoop jar "$STREAMING_JAR" \
  -D mapreduce.framework.name=yarn \
  -D mapreduce.job.name="monthly_product_totals_streaming_yarn_key2
  -D yarn.app.mapreduce.am.resource.mb=512 \
  -D yarn.app.mapreduce.am.command-opts="-Xmx384m" \
  -D mapreduce.map.memory.mb=512 \
  -D mapreduce.map.java.opts="-Xmx384m" \
  -D mapreduce.reduce.memory.mb=512 \
  -D mapreduce.reduce.java.opts="-Xmx384m" \
  -D mapreduce.job.reduces=1 \
  -D stream.num.map.output.key.fields=2 \
  -D mapred.text.key.partitioner.options=-k1,2 \
  -partitioner org.apache.hadoop.mapred.lib.KeyFieldBasedPartitione
  -file mapper.py \
  -file reducer.py \
  -mapper "/usr/bin/python3 mapper.py" \
```

Figure 3: Submission and execution of a Hadoop Streaming MapReduce job on YARN using Python mapper and reducer scripts

Successful submission is evidenced by the assigned YARN application ID **application_1770196691222_0003**, the tracking URL shown in the client output (...:8088/proxy/application_1770196691222_0003/), and the ResourceManager UI indicating **FINISHED** with **SUCCEEDED** status. Job execution characteristics further confirm correct distributed processing: the system created **6 input splits**, aligning with the 128MB HDFS block configuration and the earlier FSCK evidence of multiple blocks.

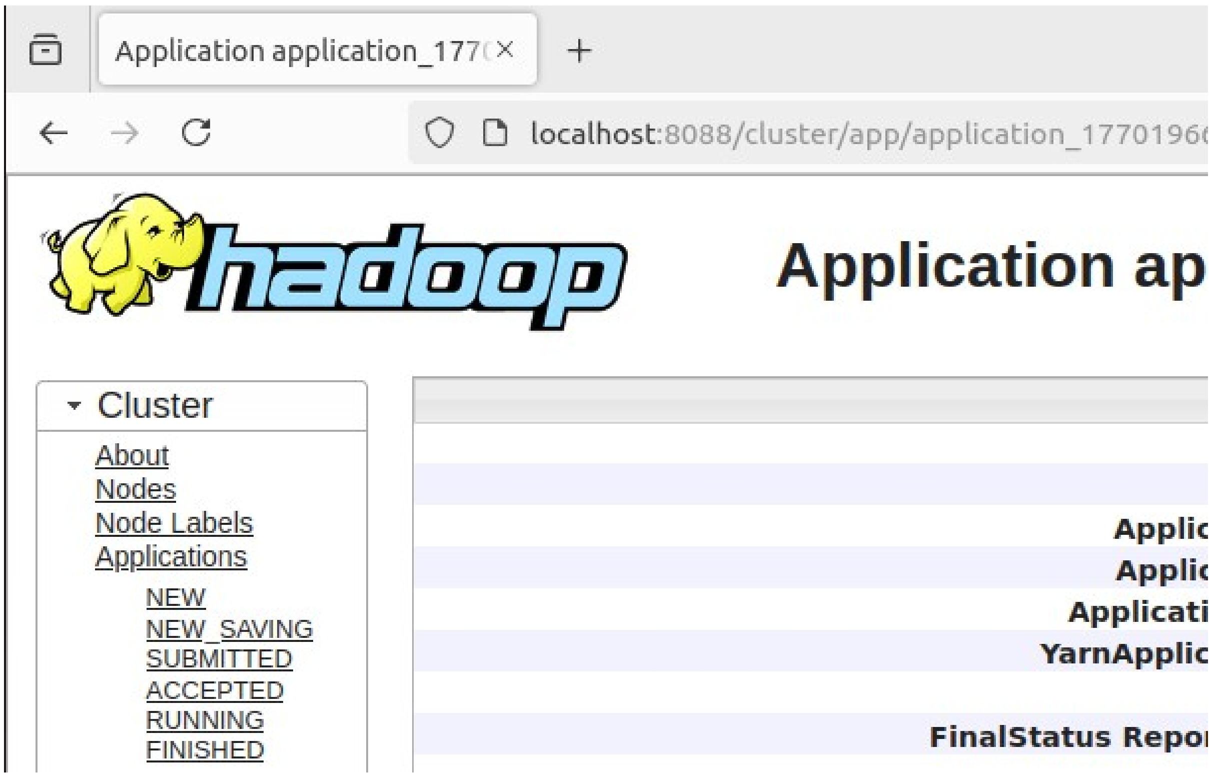
Key counters provide concise performance and scale evidence:

Map input records = 5,296,063,

Map output records = 5,296,062 (header removed),

Reduce input groups = 8,132, and

Reduce output records = 8,132, indicating one output row per unique (month, product) combination.


Application application_177(
localhost:8088/cluster/app/application_1770196

Application ap

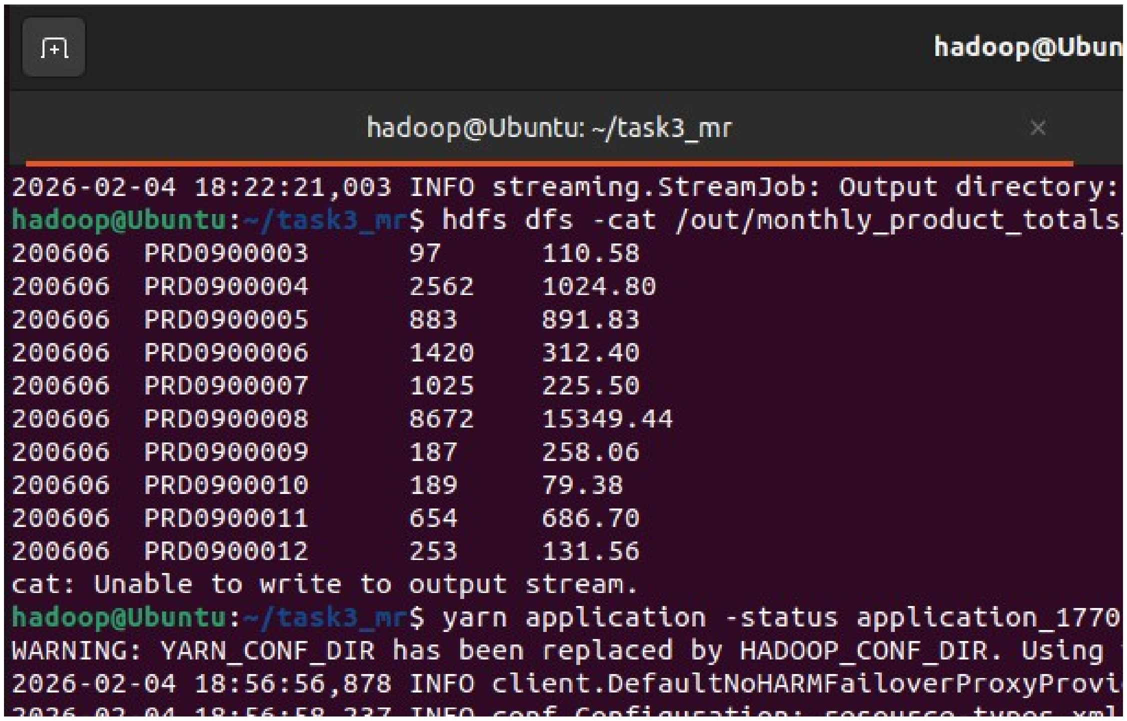
Cluster
About
Nodes
Node Labels
Applications
NEW
NEW_SAVING
SUBMITTED
ACCEPTED
RUNNING
FINISHED

Applic
Applic
Applicati
YarnApplic
FinalStatus Repo

Figure 4: YARN ResourceManager UI showing successful completion of the MapReduce application

4. Results and brief interpretation

The HDFS output was validated by previewing the reducer results:



```
hadoop@Ubun
hadoop@Ubuntu: ~/task3_mr
2026-02-04 18:22:21,003 INFO streaming.StreamJob: Output directory:
hadoop@Ubuntu:~/task3_mr$ hdfs dfs -cat /out/monthly_product_totals
200606  PRD0900003      97      110.58
200606  PRD0900004      2562    1024.80
200606  PRD0900005      883     891.83
200606  PRD0900006      1420    312.40
200606  PRD0900007      1025    225.50
200606  PRD0900008      8672    15349.44
200606  PRD0900009      187     258.06
200606  PRD0900010      189     79.38
200606  PRD0900011      654     686.70
200606  PRD0900012      253     131.56
cat: Unable to write to output stream.
hadoop@Ubuntu:~/task3_mr$ yarn application -status application_1770
WARNING: YARN_CONF_DIR has been replaced by HADOOP_CONF_DIR. Using
2026-02-04 18:56:56,878 INFO client.DefaultNoHARMFailoverProxyProvi
```

Figure 5: Viewing MapReduce output in HDFS and confirming successful job completion via YARN application status

From the output, it has been seen that each row indicates that, for month **200606**, product **PRD0900004** sold **2,562 units** with a total spend of **1,024.80** (currency units as recorded). This representation supports both "top-selling product" identification (by ranking total_quantity or total_spend) and downstream ML feature engineering (e.g., lagged monthly demand). It does not show the corruption of data and failure of jobs.

5. Comparison (MapReduce vs Hive)

Although the implementation here uses MapReduce Streaming, the same computation can be expressed in Apache Hive as a declarative aggregation, conceptually:

SELECT substr(SHOP_DATE,1,6) AS month,

PROD_CODE AS product_id,

SUM(QUANTITY) AS total_quantity,

SUM(SPEND) AS total_spend

FROM transactions

GROUP BY substr(SHOP_DATE,1,6), PROD_CODE;

In terms of development, MapReduce has additional boilerplate (custom parsing, explicit key/value emission, reducer logic), but has as much flexibility as desired with irregular data cleansing and custom transformations. Hive usually saves on development time due to the fact that logic is written in SQL, is more maintainable in analytics team, and has optimisation capabilities (e.g. query planning, predicate pushdown based on storage formats). By contrast, MapReduce offers more fine-grained control over partitioning, memory, and execution parameters, which may be beneficial in performance tuning and scale-debugging at the price of a more complex implementation (El Yazidi et al., 2021).

Task 4:- Machine Learning + Visualization + Business Insights

1. ML objective and label definition

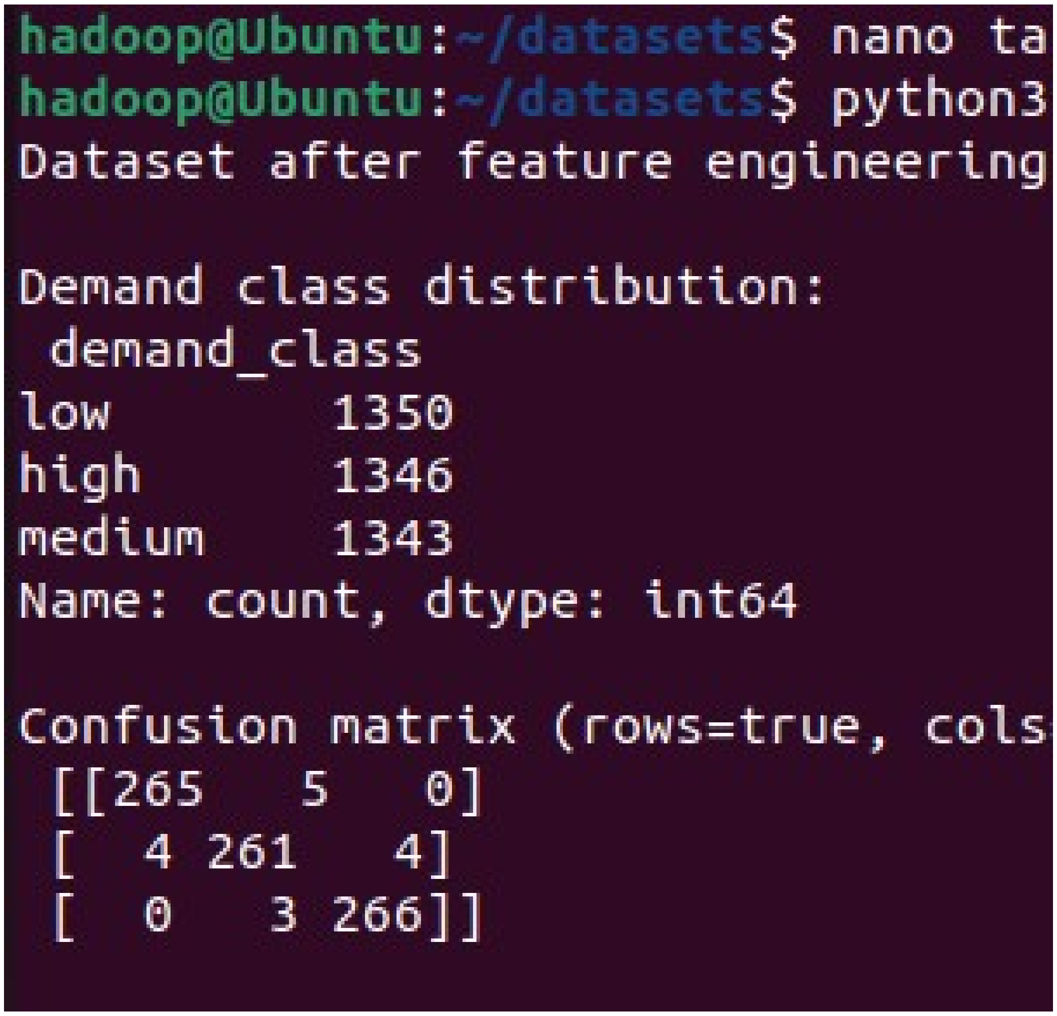
Task 4 was an operationalisation of a supervised learning problem which is in line with the retail goal of understanding demand and planning. The goal was to categorize the monthly product demand into three discrete categories: low, medium, and high with the use of the aggregated monthly outputs created on Task 3 (month, product code, total quantity, and total spend). The quantile-based discretisation was used to define the class label based on the response variable total_quantity using tertiles (three equal-frequency bins). This decision option is suitable to decision situations where the stakeholders need interpretable bands of demand (e.g., high-demand items should be replenished first), but not specific continuous predictions. It also makes it less sensitive to extreme values and makes model training and evaluation balanced (Moraffah et al., 2020).

2. Feature engineering and preprocessing

The feature engineering was intended to represent the temporal persistence and product level baseline behaviour without being computationally complex and explainable. To build the lagged predictors that indicate short-run momentum (qty_lag1 (quantity in previous month) and spend_lag1 (spend in previous month)) the records were first sorted by (prod_code, month). Second, the product-specific baseline demand was modeled with the help of qty_mean_prod and spend_mean_prod that were calculated as the average per-product values within the available months. This was supplemented by a numeric index of the month (e.g. 200606) to enable the model to capture crudely seasonal or time position effect in the small sample. The rows in which there was no lag value available (i.e., the first month to be observed in each product) were discarded, and a feature-engineered dataset with (4039, 10) records was obtained. No further scaling was needed since the chosen algorithm (Random Forest) is monotonic feature scaling invariant and has high heterogeneous feature magnitude tolerance. It was checked that the balance between classes was almost the same (low=1350, medium=1343, high=1346) and it allows maintaining a stable training without re-weighting (Zhu et al., 2022).

3. Model training and evaluation

A Random Forest classifier was chosen due to the following reasons: (i) it models non-linear interactions between lag effects and demand level, (ii) it can interact between features of quantity and spend without the need to specification, and (iii) it does provide feature-importance values that facilitate results to be interpreted by business users. Eighty percent train-two-fifths test split with stratification was used to train the model to maintain the proportions of the classes and a random seed of 80 to ensure repeatability. The selected model was configured with 200 trees and in general, it was adequate to stabilise ensemble predictions without making training time too high.



```
hadoop@Ubuntu:~/datasets$ nano ta
hadoop@Ubuntu:~/datasets$ python3
Dataset after feature engineering

Demand class distribution:
 demand_class
low       1350
high      1346
medium    1343
Name: count, dtype: int64

Confusion matrix (rows=true, cols
 [[265   5   0]
 [  4 261   4]
 [  0   3 266]]
```

Figure 6: Model Training and Evaluation

Performance was measured by conventional multi-class classification measures. The performance was found to be quite good: the overall accuracy ≈ 0.98 on the held-out test set (n=808). The confusion matrix indicated lower levels of misclassification, mostly between closest classes (e.g. medium vs. high or low), which is reasonable since the boundaries of the classes are quantiles. Precision, recall, and F1-scores were all high (high: precision=0.99, recall=0.99, low: precision=0.99, recall=0.98, medium: precision=0.97, recall=0.97). Taken together, these findings all suggest that the engineered predictors would capture significant signal in terms of demand class. It must also be reported, though, that random splitting may provide optimistic estimates of time-indexed retail data; a time-based split (train on past months, test on future months), would be even more valid given more months of data (Zhou et al., 2021).

4. Visualisations and business insights

Three business-facing visualisations were generated and saved as PNG outputs: **(i) demand class distribution**, **(ii) top-10 products by total quantity sold**, and **(iii) feature importance**. The demand distribution plot confirmed that the class design supports balanced learning and avoids dominance by a single category.

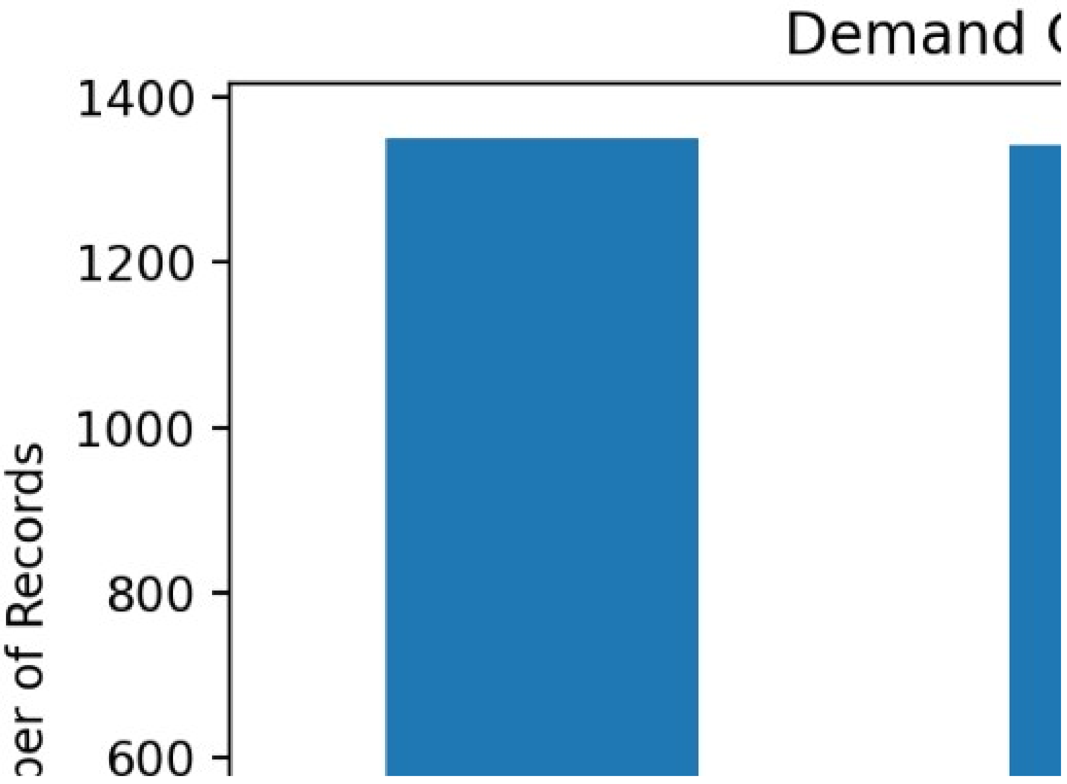


Figure 7: Demand Class Distribution

The top-10 bar chart offers real-time operation importance as it identifies the products that cause the highest proportions of unit movement in the chosen sample. Such products are obvious targets of closer inventory controls, increased service level goals, and anticipatory restocking.

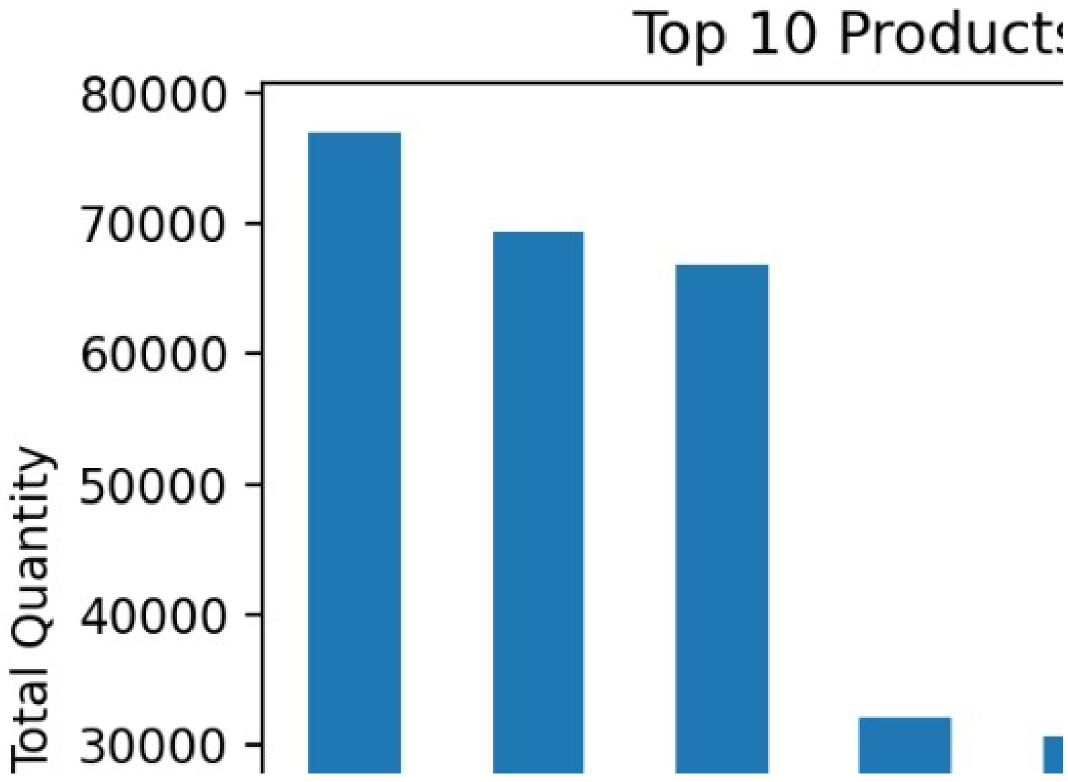


Figure 8: Top 10 Products by Total Quantity Sold

Lastly, the feature-importance plot revealed that the variables of the most predictive performance were qty_mean_prod and qty_lag1, meaning that long-run product popularity and short-run momentum of demand were the most influential factors in demand classification. In this setup, spend-based characteristics would not be as informative and it may be that quantity trends would be more informative in the subset available to classify the demand bands using expenditure.

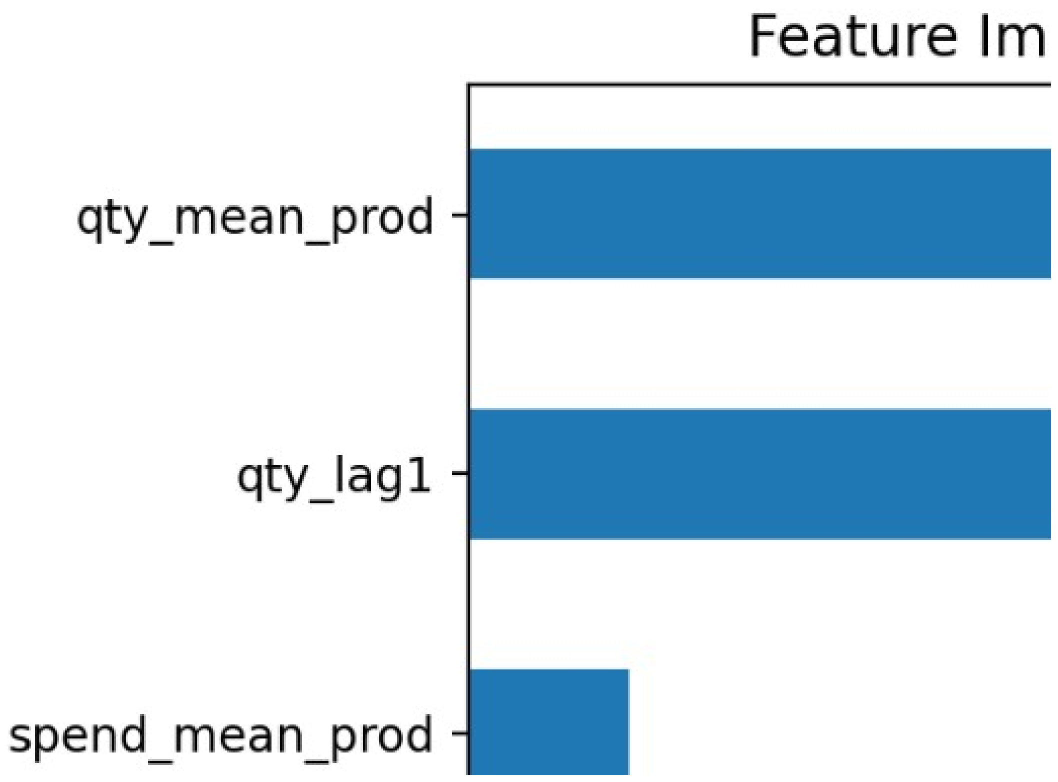


Figure 9: Feature Importance

In terms of business, the model facilitates segmented stock planning (e.g., reorder priority to high-demand items), promotion targeting (targeting campaigns at items that are expected to jump high-demand to high-demand), and exception reporting (notifying items with high-demand that are starting to move down in lag features).

Conclusion and Future Work

The project was able to showcase an end to end workflow of big data analytics on a pseudo-distributed Hadoop environment. Task 1 was a retail issue that aimed at understanding the most sold products on a monthly basis and making decisions based on demand. Task 2 deployed the platform and ingestion pipeline and stored a combined set of transactions in HDFS with distributed verification of the storage by block-level evidence (FSCK) and management of healthy namespace metadata. Task 3 operationalised the operationalised distributed processing with the help of Hadoop streaming on YARN to calculate monthly product totals (quantity and spend) in order to generate reproducible outputs to be used in downstream analytics. Task 4 used a supervised learning method, which involved the creation of lag and product-level features and a Random Forest classifier to forecast demand classes with a high level of evaluation and generating business-facing visualisations.

Each has its limitations such as a single-node deployment, a replication factor of 1 and a subset range of months. Further work should perform the corresponding Hive query to make empirical comparison of future work, extend to other months, perform additional modelling to forecasting based on regression equation, and scale-up to a more realistic multi-node cluster.

Bibliography.

dunnhumby Source Files. Dataset portal for retail analytics and time-series datasets. Available at: https://www.dunnhumby.com/source-files/ (accessed on 04-02-2026).

Ma, C., Zhao, M. and Zhao, Y., 2023. An overview of Hadoop applications in transportation big data. *Journal of traffic and transportation engineering (English edition)*, *10*(5), pp.900-917. https://doi.org/10.1016/j.jtte.2023.05.003

Li, Y. and Hei, X., 2025. Performance optimization of computing task scheduling based on the Hadoop big data platform. *Neural Computing and Applications*, *37*(13), pp.8181-8192. https://doi.org/10.1007/s00521-022-08114-3

Qi, T. and Liu, H., 2024, September. Research on the Design of a Sales Forecasting System Based on Hadoop Big Data Analysis. In *Proceedings of the 2024 2nd International Conference on Internet of Things and Cloud Computing Technology* (pp. 193-198). https://doi.org/10.1145/3702879.3702913

Zhang, D., Dai, Z.Y., Sun, X.P., Wu, X.T., Li, H., Tang, L. and He, J.H., 2024. A distributed data processing scheme based on Hadoop for synchrotron radiation experiments. *Synchrotron Radiation*, *31*(3), pp.635-645. https://doi.org/10.1107/S1600577524002637

Adamov, A., 2023. Large-scale data modelling in hive and distributed query processing using mapreduce and tez. *arXiv preprint arXiv:2301.12454*. https://doi.org/10.48550/arXiv.2301.12454

El Yazidi, A., Azizi, M.S., Benlachmi, Y. and Hasnaoui, M.L., 2021. Apache Hadoop-MapReduce on YARN framework latency. *Procedia Computer Science*, *184*, pp.803-808. https://doi.org/10.1016/j.procs.2021.03.100

Zhou, J., Gandomi, A.H., Chen, F. and Holzinger, A., 2021. Evaluating the quality of machine learning explanations: A survey on methods and metrics. *Electronics*, *10*(5), p.593. https://doi.org/10.3390/electronics10050593

Zhu, M., Wang, J., Yang, X., Zhang, Y., Zhang, L., Ren, H., Wu, B. and Ye, L., 2022. A review of the application of machine learning in water quality evaluation. *Eco-Environment & Health*, *1*(2), pp.107-116. https://doi.org/10.1016/j.eehl.2022.06.001

Moraffah, R., Karami, M., Guo, R., Raglin, A. and Liu, H., 2020. Causal interpretability for machine learning-problems, methods and evaluation. *ACM SIGKDD Explorations Newsletter*, *22*(1), pp.18-33. https://doi.org/10.1145/3400051.3400058

Appendix

mapper.py

```
#!/usr/bin/env python3

import sys, csv

reader = csv.reader(sys.stdin)

for row in reader:

    if not row:

        continue

    # Skip header row

    if row[0] == "SHOP_WEEK":

        continue

    try:

        shop_date = row[1].strip()          # SHOP_DATE e.g. 20060627

        month = shop_date[:6]               # YYYYMM

        qty = int(row[4])                   # QUANTITY

        spend = float(row[5])               # SPEND

        prod = row[6].strip()               # PROD_CODE

    except Exception:

        continue  # skip malformed lines
    # key = month + product; value = qty and spend
    print(f"{month}\t{prod}\t{qty}\t{spend}")
```

reducer.py

```
#!/usr/bin/env python3
import sys
import signal
signal.signal(signal.SIGPIPE, signal.SIG_DFL)
current_key = None
qty_sum = 0
spend_sum = 0.0
for line in sys.stdin:
    parts = line.strip().split("\t")
    if len(parts) != 4:
        continue

    month, prod, qty_s, spend_s = parts
    key = (month, prod)
    try:
        qty = int(qty_s)
        spend = float(spend_s)
    except Exception:
        continue
    if current_key is None:
        current_key = key
    if key != current_key:
        # output aggregated result
        m, p = current_key
        print(f"{m}\t{p}\t{qty_sum}\t{spend_sum:.2f}")
        current_key = key
        qty_sum = 0
        spend_sum = 0.0
    qty_sum += qty
    spend_sum += spend


# last key
if current_key is not None:
   m, p = current_key
    print(f"{m}\t{p}\t{qty_sum}\t{spend_sum:.2f}")
```