

Assignment 2

Introduction

BAM-1043-01 Assignment

C#: c0932089_Assignment2

Name: Prabesh Rai

Date: 1 March, 2024

Question 01,

MapReduce

Objective: Create and construct a MapReduce job to efficiently handle huge data sets using the MapReduce paradigm.

Task Description

1. Problem Statement: Define the problem statement the MapReduce task seeks to solve. Determine the data sets to be processed and the tasks to be completed using MapReduce.

2. Input Data: Please specify the format and source of the input data. Describe the data's structure and any pretreatment processes necessary before MapReduce processing.

3. Map phase:

Explain the function of the Map step in the MapReduce process. Define the key-value pairs generated by the Mapper. Outline the logic and actions that will be done in the mapper function.

4. Shuffle and sort:

- Describe how to shuffle and sort intermediate key-value pairs between the Mapper and Reducer stages.
- Explain how data is partitioned and delivered to reducers.

5. Reduce phase:

- Define the key-value (K-V) pairs sent to the Reducer.
- Specify the logic and operations that will be executed in the Reducer function.
- Define the output format of the Reducer phase.

6. Output Data: Describe the format and destination for the MapReduce job's output data.

7. Testing and Evaluation: Discuss the technique for validating the MapReduce job's accuracy and efficiency. Establish measures for measuring work performance.

8. Documentation and Reporting: Specify the documentation needs, such as code comments, job setup information, and troubleshooting tips. Specify the format of the final project report.

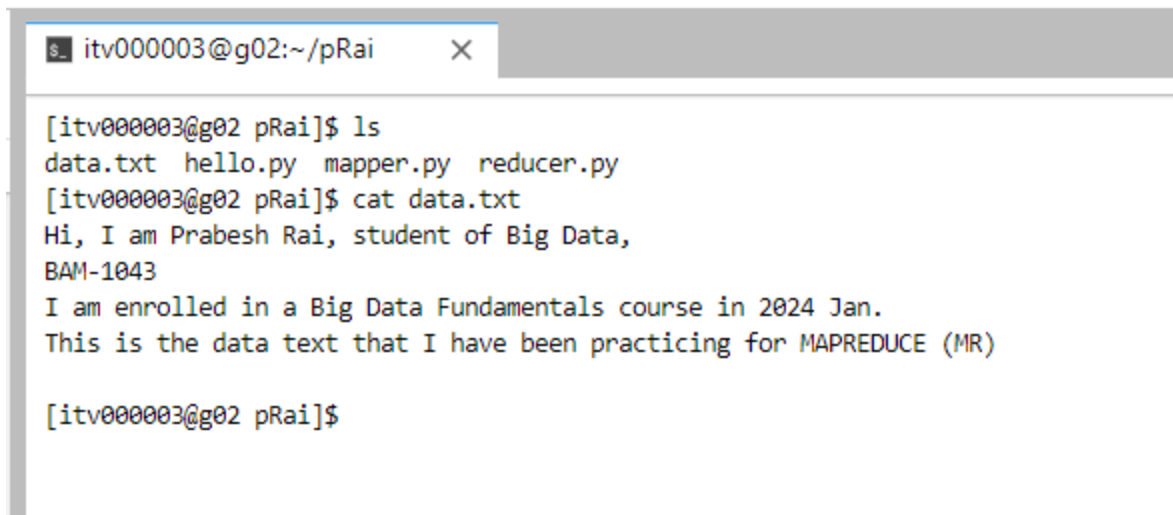
1. Problem Statement: The problem statement for this MapReduce task is to analyze a large text dataset. I've generated a data.txt file within the itversity lab pRai folder (personal folder) to determine the data sets for processing via MapReduce tasks. Screenshots of the created file are presented below:

```

itv000003@g02:~/pRai X
Hi, I am Prabesh Rai, student of Big Data,
BAM-1043
I am enrolled in a Big Data Fundamentals course in 2024
Jan.
This is the data text that I have been practicing for MAA
PRODUCE (MR)

```

Fig 1: Creating a dataset for a task inside the data.txt file

A terminal window with a title bar showing the user 'itv000003' at host 'g02' in directory '~ / pRai'. The terminal displays the output of 'ls' and 'cat data.txt' commands. The 'ls' command lists 'data.txt', 'hello.py', 'mapper.py', and 'reducer.py'. The 'cat data.txt' command displays the content of the file, which includes a greeting, a student ID, and course information.

```
itv000003@g02:~/pRai X
[itv000003@g02 pRai]$ ls
data.txt hello.py mapper.py reducer.py
[itv000003@g02 pRai]$ cat data.txt
Hi, I am Prabesh Rai, student of Big Data,
BAM-1043
I am enrolled in a Big Data Fundamentals course in 2024 Jan.
This is the data text that I have been practicing for MAPREDUCE (MR)

[itv000003@g02 pRai]$
```

Fig 2: displaying the content of the data.txt file

2. Input Data: The input data for this MapReduce task is stored in a plain text file named data.txt. The file was generated within the itversity lab pRai folder (personal folder) as described in step 1.

Format: The data is unstructured plain text.

Source: The source of the data is the data.txt file created within the itversity lab pRai folder.

Data Structure: Since the data is unstructured, there is no fixed structure or format. Each line in the data.txt file represents a different type of information.

Pretreatment Processes: For the pretreatment processes of the data I have striped and split the data. I have created a mapper.py file where the functions for strip and split are programmed. Screenshots of stripping and splitting using Python programming are presented below in Fig 2.

- a. **Stripping:** Stripping involves removing unnecessary characters, whitespace, or special symbols from the raw text data. The main purpose of the stripping process help clean the data by eliminating noise and irrelevant information that could interfere with further analysis. In Fig 3, inside the scope of the for-loop, line = line.strip() function is for stripping.
- b. **Splitting:** Splitting divides the text into smaller units or tokens, such as words or phrases, based on predefined delimiters or patterns. The main purpose of splitting is to break down the text into manageable chunks, splitting facilitates subsequent analysis tasks such as counting word frequencies or identifying patterns. In Fig 3, inside the scope of the food-loop, line = line.split() function is for splitting.

We will now write a mapper.py program as follows;
Screenshots

Fig 3: (program) logic for pretreatment process file

Fig 4: displaying the content of mapper.py file

3. Map phase: The Map step in the MapReduce process, processes each input record independently and generates intermediate key-value pairs. Each input record is processed independently by the mapper function.

Key-value pair: The output of the mapper function comprises key-value pairs, as depicted in Fig 5. In Fig 5, the command `python mapper.py < data.txt` executes the Python script "mapper.py" and feeds the data from "data.txt" into the script for subsequent processing. Within "mapper.py," the program for the map phase is encapsulated, defining the logic and actions performed on each input record to generate the intermediate key-value pairs necessary for the MapReduce process. Each line in Fig 5, represents a key-value pair generated by the mapper. The keys are individual words extracted from the input data such as "Hi," "I," "am," "Prabesh," etc. The value associated with each key is the number 1, indicating the occurrence of that word in the input data.

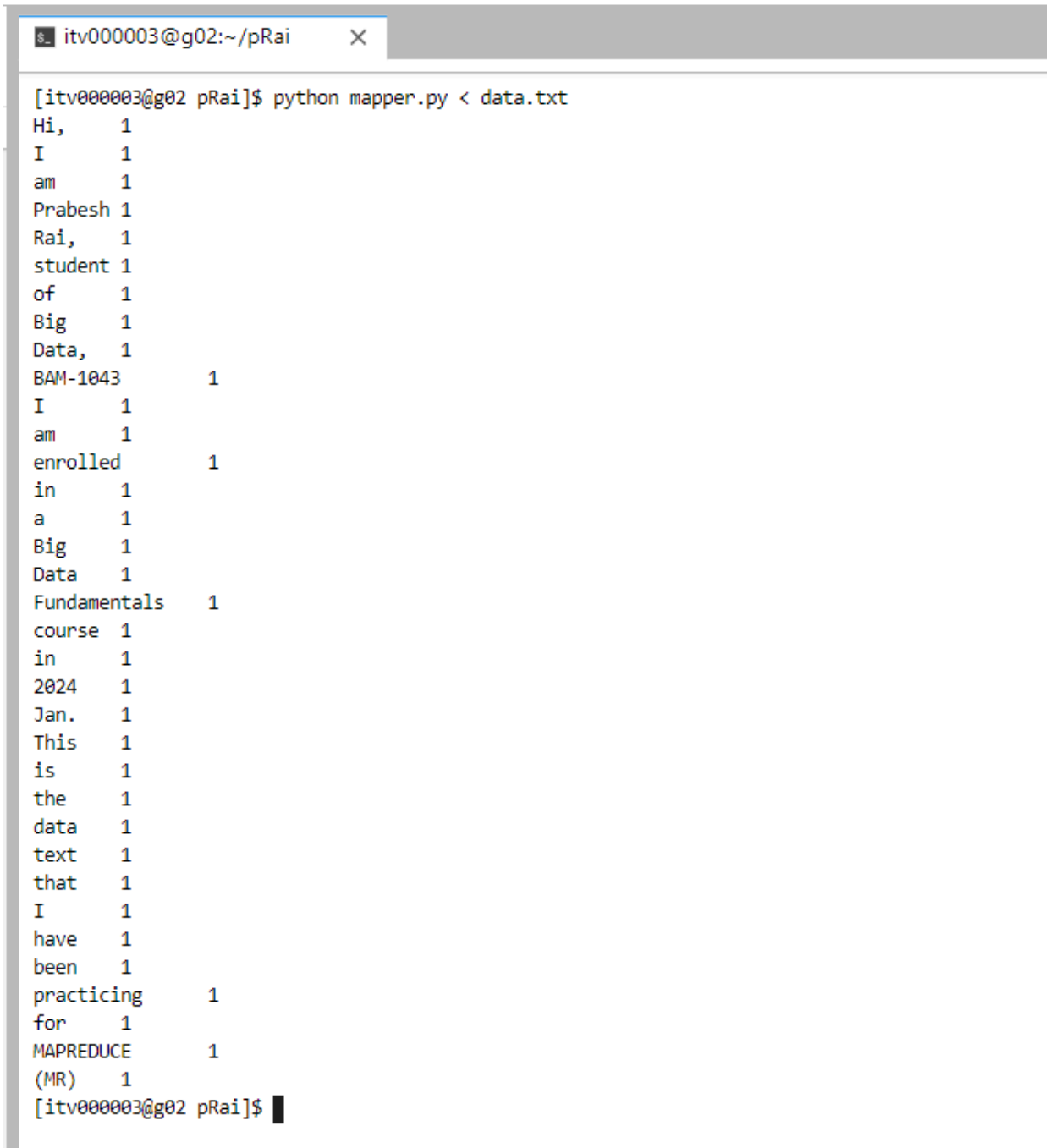
Logic and action: The mapper function logic includes tokenization, filtering, and counting as programmed in Fig 3, and the actions include emitting key-value pairs, key generation, value assignment, and output to intermediate storage as shown in Fig 5.

Screenshots for Map phase are presented below (step 3):

Step 3.

Perform a Mapper task by passing the file to the mapper.py;

Screenshots



```
itv000003@g02:~/pRai X
[itv000003@g02 pRai]$ python mapper.py < data.txt
Hi,      1
I        1
am       1
Prabesh  1
Rai,     1
student  1
of       1
Big      1
Data,    1
BAM-1043      1
I        1
am       1
enrolled      1
in       1
a        1
Big      1
Data     1
Fundamentals  1
course   1
in       1
2024    1
Jan.    1
This    1
is      1
the     1
data    1
text    1
that    1
I       1
have    1
been    1
practicing  1
for     1
MAPREDUCE      1
(MR)    1
[itv000003@g02 pRai]$
```

Fig 5: Map phase

4. Shuffle and sort: In the shuffle and sort phase of MapReduce, intermediate key-value pairs generated by the mapper undergo shuffling and sorting before being transmitted to reducers. This process includes partitioning the data based on keys and ensuring that all values associated with a specific key are grouped, optimizing the efficiency of processing by reducers.

After the data is shuffled and organized, it is dispatched to the corresponding reducers for additional processing. Each reducer is provided with a partition of key-value pairs, enabling it to execute aggregation and summarization tasks autonomously. For example, the outputs from passing data.txt data into mapper.py (in Fig 5) are combined and grouped as follows:

Group 1.

I 1

I 1

I 1

Group 2.

Big 1

Big 1

Group 3.

Fundamentals 1

And so on.

5. Reduce phase: The reduce phase, the second step in the MapReduce process, aims to aggregate and process the intermediate key-value pairs produced by the mappers. These intermediate pairs are received by reducers, which carry out predefined operations on them.


Key-value (K-V) pairs sent to the Reducer: Each reducer is supplied with a collection of key-value pairs organized by key. Here, these keys denote significant attributes, such as words, while the corresponding values encompass counts.

Logic and operations: The reducer function operates on grouped key-value pairs, typically performing common operations such as stripping, splitting, filtering, transformation, and output preparation. The logic and operations functions to execute in the reducer phase are programmed inside reducer.py file as presented in Fig 6 (step 4).

Step 4.

reducer.py file created

Screenshots



```
itv000003@g02:~/pRai X
"""reducer.py"""

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()

    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)

    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue

    # this IF-switch only works because Hadoop sorts mapp
    output
    # by key (here: word) before it is passed to the redd
    ucer
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # write result to STDOUT
            print '%s\t%s' % (current_word, current_counn
t)
            current_count = count
            current_word = word

# do not forget to output the last word if needed!
if current_word == word:
    print '%s\t%s' % (current_word, current_count)
39,5 Bot
```

Fig 6: (program) logic for reduce phase


```
itv000003@g02:~/pRai X
[itv000003@g02 pRai]$ cat reducer.py
"""reducer.py"""

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()

    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)

    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # write result to STDOUT
            print '%s\t%s' % (current_word, current_count)
            current_count = count
            current_word = word

# do not forget to output the last word if needed!
if current_word == word:
    print '%s\t%s' % (current_word, current_count)
[itv000003@g02 pRai]$
```

Fig 7: displaying the content of the reducer.py file

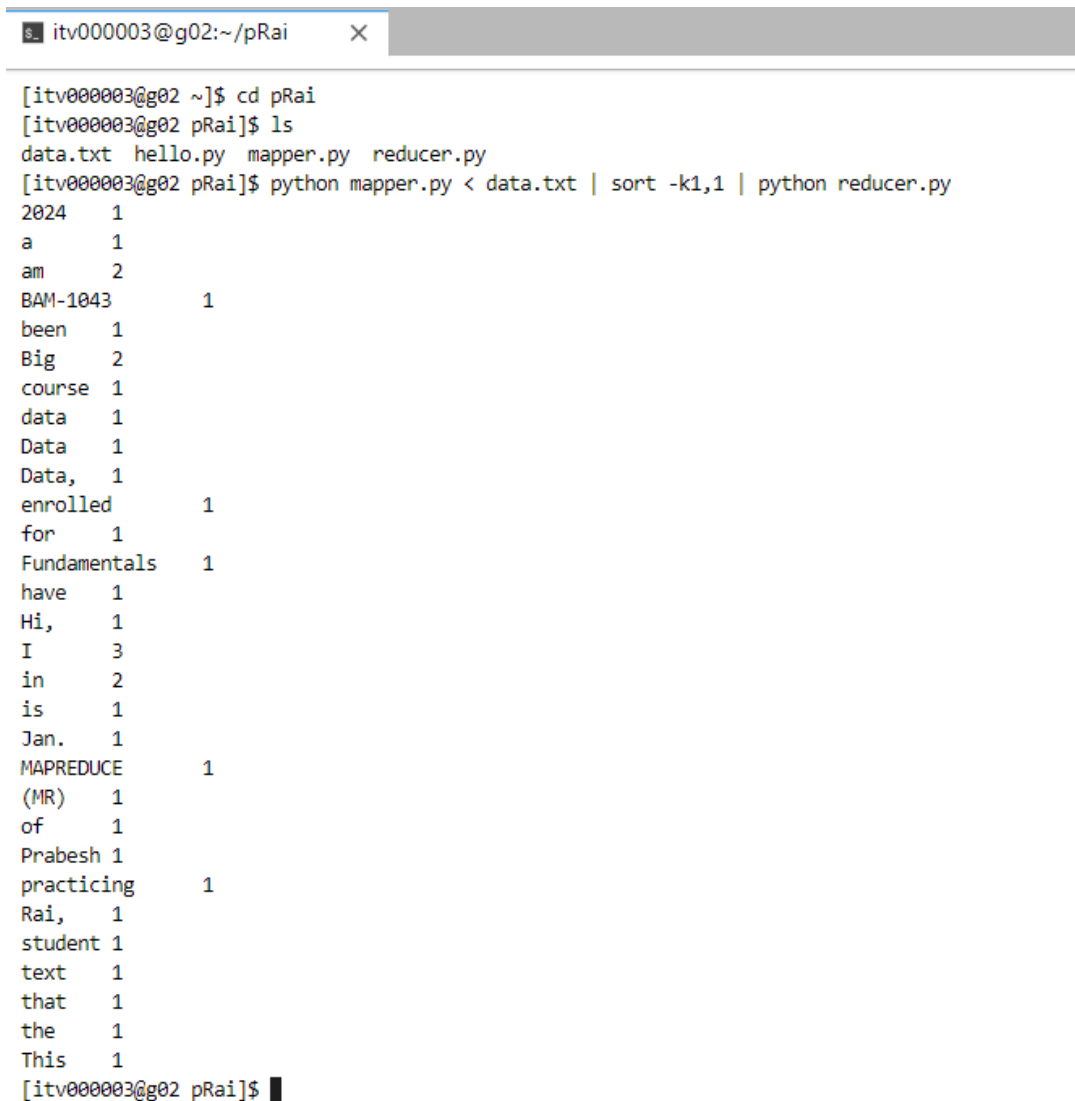
Output Format of the Reducer Phase: The final output consists of a list of key-value pairs, presenting total counts and results customized to the problem being addressed.

Output Data: The output data of a MapReduce job is structured as key-value pairs, where each key signifies a significant attribute such as a word and its corresponding value encapsulates pertinent information, such as counts. The output key-value data is stored in a distributed file system, such as the Hadoop Distributed File System (HDFS), ensuring fault tolerance, scalability, and efficient data distribution across a cluster. Output files are partitioned and distributed across multiple nodes in the cluster, facilitating accessibility for users to analyze or process the data further. The output is presented in Fig 8 (step 5).

Step 5.

Perform a Reducer task by passing the file to the `reduce.py`, which takes two arguments;

Screenshots



```
itv000003@g02:~/pRai X
[itv000003@g02 ~]$ cd pRai
[itv000003@g02 pRai]$ ls
data.txt  hello.py  mapper.py  reducer.py
[itv000003@g02 pRai]$ python mapper.py < data.txt | sort -k1,1 | python reducer.py
2024      1
a          1
am         2
BAM-1043   1
been       1
Big        2
course     1
data       1
Data       1
Data,      1
enrolled   1
for        1
Fundamentals 1
have       1
Hi,        1
I          3
in         2
is         1
Jan.       1
MAPREDUCE  1
(MR)       1
of         1
Prabesh    1
practicing 1
Rai,       1
student    1
text       1
that       1
the        1
This       1
[itv000003@g02 pRai]$
```

Fig 8: Output data from a MapReduce

7. Testing and Evaluation: To ensure the precision and efficiency of a MapReduce operation involves conducting unit tests on specific components of the MapReduce task, such as mapper and reducer functions, to confirm their output matches expectations across various input scenarios for validation and performance evaluation. Also, integration testing can be carried out to validate the entire MapReduce workflow, encompassing data input, processing, and output phases.

To measure work performance, one can establish several fundamental metrics:

- a. **Execution Time:** Measure the duration required for the MapReduce job to finalize its execution, encompassing the period from data input commencement to the production of the final output.
- b. **Resource Utilization:** Track the usage of cluster resources, such as CPU, memory, and network bandwidth, throughout job execution to verify effective resource allocation.
- c. **Scalability Assessment:** Analyze the scalability of the MapReduce task by assessing its performance with escalating data volume and cluster dimensions, evaluating its ability to manage larger datasets and accommodate additional processing nodes.
- d. **Data Consistency:** Verify that the output data generated by the MapReduce job maintains consistency and accuracy across various executions and under diverse conditions.
- e. **Fault Tolerance Evaluation:** Test the MapReduce framework's resilience to failures by simulating errors and observing seamless recovery, ensuring uninterrupted execution and data integrity.

8. Documentation and Reporting

Code Comments: Comments are embedded within MapReduce code (inside mapper.py & reducer.py files) to elucidate the purpose of each function, class, and section, and to clarify any intricate logic or non-obvious decisions made during development.

Job Setup Information: Comprehensive documentation on configuring the MapReduce job, including instructions on setting up input data sources, are presented as different steps with screenshots.

Troubleshooting Tips: Troubleshooting guidelines for common issues encountered during MapReduce job execution are presented which includes addressing error messages and their potential causes, as well as suggesting diagnostic steps and resolutions to resolve encountered problems.

The format for the final project report includes:

- a. **Introduction:** Description of the project's context and goals.
- b. **Methodology:** Details of the MapReduce architecture and components used and description of the mapper and reducer functions.

- c. Implementation: Explanation of how the MapReduce job was set up and Include configuration details.
- d. Results and Analysis: The final output and insights.
- e. Challenges and Lessons Learned: Highlight of difficulties faced during development.
- f. Conclusion: Recap of the project's achievements and discussion for future improvements.
- g. Appendices: Include code snippets, sample input/output, and additional details.

Question 02,

Create appropriate visualizations for the KPIs (please refer to the D2L page of the course).

Databricks:

```
c0932089 Assignemnt 2, Notebook 2024-03-01 01:43:35 Python ⌵ ☆
File Edit View Run Help Last edit was 5 hours ago New cell UI: OFF ⌵

Cmd 1
1 /FileStore/tables/menu-1.csv
2 /FileStore/tables/sales-3.csv

Cmd 2
1 from pyspark.sql.types import StructType, StructField, IntegerType, StringType, DateType

Command took 0.03 seconds -- by raiprabesh789@gmail.com at 3/1/2024, 2:37:14 AM on My Cluster
```

Schema structure:

a. Data Schema for Menu

```
Cmd 3
1 # Menu Data Schema
2
3 schema = StructType([
4     StructField("Id", IntegerType(), True),
5     StructField("Item Name", StringType(), True),
6     StructField("Price in $", StringType(), True)
7 ])
8 menu_df = spark.read.format("csv").option("inferSchema", "true").schema(schema).load("/FileStore/tables/menu-1.csv")
9 display(menu_df)
10 #menu_df.show()
```

▶ (1) Spark Jobs

▶ menu_df: pyspark.sql.dataframe.DataFrame = [Id: integer, Item Name: string ... 1 more field]

Table ⌵ +

	Id	Item Name	Price in \$
1	1	PIZZA	100
2	2	Chowmin	150
3	3	sandwich	120
4	4	Dosa	110
5	5	Biryani	80
6	6	Pasta	180

6 rows | 0.72 seconds runtime

Command took 0.72 seconds -- by raiprabesh789@gmail.com at 3/1/2024, 3:08:12 AM on My Cluster

b. Data Schema for Sales

Cmd 4

```
1 # Sales Data Schema
2 schema= StructType([
3     StructField("Id", IntegerType(), True),
4     StructField("Customer Name", StringType(), True),
5     StructField("Order Date", DateType(), True),
6     StructField("Location", StringType(), True),
7     StructField("Source Order", StringType(), True),
8 ])
9 sales_df = spark.read.format("csv").option("inferSchema", "true").schema(schema).load("/FileStore/tables/sales-3.csv")
10 display(sales_df)
11 #sales_df.show()
12
```

► (1) Spark Jobs

► sales_df: pyspark.sql.dataframe.DataFrame = [Id: integer, Customer Name: string ... 3 more fields]

Table ▾ +

	Id	Customer Name	Order Date	Location	Source Order
1	1	A	2023-01-01	Canada	Uber Eats
2	2	A	2022-01-01	Canada	Uber Eats
3	2	A	2023-01-07	Canada	Uber Eats
4	3	A	2023-01-10	Canada	Restaurant
5	3	A	2022-01-11	Canada	Uber Eats
6	3	A	2023-01-11	Canada	Restaurant
7	2	B	2022-02-01	Canada	Uber Eats

118 rows | 0.66 seconds runtime

Command took 0.66 seconds -- by raiprabesh789@gmail.com at 3/1/2024, 3:16:40 AM on My Cluster

Data analysis using PySpark and visualizations in Databricks

Joining the Sales and Menu data (Preparation for data visualization)

Cmd 5

Join Sales Data and Menu Data

Cmd 6

```
1 joined_data = sales_df.join(menu_df, sales_df.Id == menu_df.Id, "left")
2 cleaned_data = joined_data.na.drop(subset=["Customer Name", "Order Date", "location", "Source Order", "Item Name", "Price in $"])
3 display(cleaned_data)
```

► (2) Spark Jobs

► joined_data: pyspark.sql.dataframe.DataFrame = [Id: integer, Customer Name: string ... 6 more fields]

► cleaned_data: pyspark.sql.dataframe.DataFrame = [Id: integer, Customer Name: string ... 6 more fields]

Table ▾ +

	Id	Customer Name	Order Date	Location	Source Order	Id	Item Name	Price in \$
1	1	A	2023-01-01	Canada	Uber Eats	1	PIZZA	100
2	2	A	2022-01-01	Canada	Uber Eats	2	Chowmin	150
3	2	A	2023-01-07	Canada	Uber Eats	2	Chowmin	150
4	3	A	2023-01-10	Canada	Restaurant	3	sandwich	120
5	3	A	2022-01-11	Canada	Uber Eats	3	sandwich	120
6	3	A	2023-01-11	Canada	Restaurant	3	sandwich	120
7	2	B	2022-02-01	Canada	Uber Eats	2	Chowmin	150

117 rows | 0.99 seconds runtime

Command took 0.99 seconds -- by raiprabesh789@gmail.com at 3/1/2024, 4:02:54 AM on My Cluster

1. Total Amount spent by each food category.

Cmd 7

Total Amount spent by each food category

Cmd 8

```
1 from pyspark.sql.functions import sum, upper
2 # Group by the capitalized "Item Name" and calculate the total sales amount for each category
3 category_totals = (cleaned_data
4                     .withColumn("Item Name", upper(cleaned_data["Item Name"]))
5                     .groupBy("Item Name")
6                     .agg(sum("Price in $").alias("Total Amount")))
7 # To remove null value
8 display(category_totals)
```

▶ (3) Spark Jobs

▶ category_totals: pyspark.sql.dataframe.DataFrame = [Item Name: string, Total Amount: double]

Table

Visualization 1

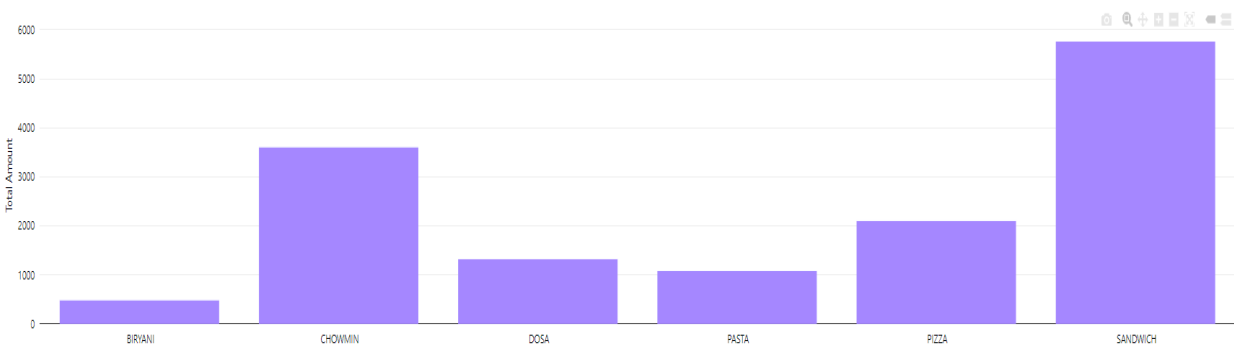
+

	Item Name	Total Amount
1	PASTA	1080
2	PIZZA	2100
3	DOSA	1320
4	SANDWICH	5760
5	BIRYANI	480
6	CHOWMIN	3600

6 rows | 0.98 seconds runtime

Visualization

Table Visualization 1



2. Total Amount spent by each customer.

Cmd 9

Total Amount spent by each customer

Cmd 10

```
1 customer_totals = cleaned_data.groupBy("Customer Name").agg(sum("Price in $").alias("Total Amount"))
2 display(customer_totals)
```

▶ (3) Spark Jobs

▶ customer_totals: pyspark.sql.dataframe.DataFrame = [Customer Name: string, Total Amount: double]

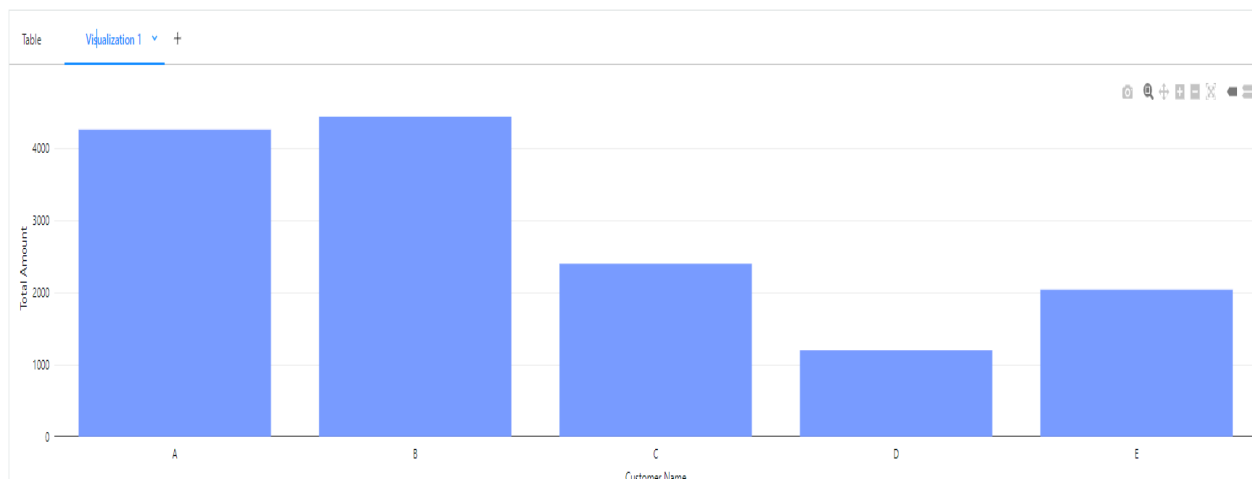
Table Visualization 1 +

	Customer Name	Total Amount
1	E	2040
2	B	4440
3	D	1200
4	C	2400
5	A	4260

↓ 5 rows | 1.74 seconds runtime

Command took 1.74 seconds -- by raiprabesh789@gmail.com at 3/1/2024, 3:17:07 AM on My Cluster

Visualization



3. Total Amount of sales for each month.

Cmd 11

Total Amount of sales for each month

Cmd 12

```
1 import pyspark.sql.functions as F
2
3 # Define a DataFrame containing the mapping of month numbers to month names
4 month_mapping = spark.createDataFrame([(1, "Jan"), (2, "Feb"), (3, "Mar"), (4, "Apr"), (5, "May"), (6, "Jun"), (7, "Jul"), (8, "Aug"), (9, "Sep"), (10, "Oct"), (11, "Nov"), (12, "Dec")], ["month", "Each Month"])
5
6 # First, extract the month from the order_date column
7 # Assuming order_date is a timestamp column
8 cleaned_data = cleaned_data.withColumn("Order Month", F.month("Order Date"))
9
10 # Then, join with the month_mapping DataFrame to replace month numbers with month names
11 cleaned_data = cleaned_data.join(month_mapping, cleaned_data["Order Month"] == month_mapping["month"], "left").drop("month")
12
13 # Group by the Months and calculate the total sales amount for each month
14 monthly_sales = cleaned_data.groupBy(cleaned_data["Each Month"]).agg(F.sum("Price in $").alias("Total Sales"))
15
16 # Sort the results by month
17 monthly_sales = monthly_sales.sort("Each Month")
18
19 # Finally, display or collect the results
20 display(monthly_sales)
```

▶ (19) Spark Jobs

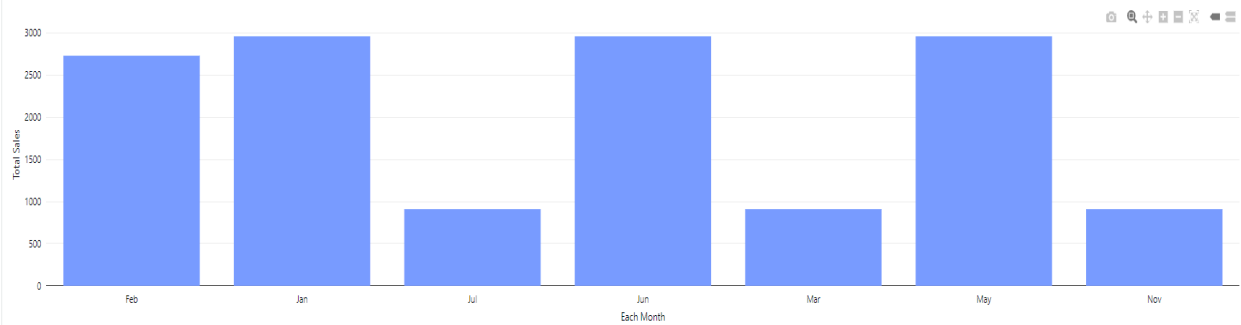
- ▶ month_mapping: pyspark.sql.dataframe.DataFrame = [month: long, Each Month: string]
- ▶ cleaned_data: pyspark.sql.dataframe.DataFrame = [id: integer, Customer Name: string ... 23 more fields]
- ▶ monthly_sales: pyspark.sql.dataframe.DataFrame = [Each Month: string, Total Sales: double]

Table Visualization 1 +

	Each Month	Total Sales
1	Feb	2730
2	Jan	2960
3	Jul	910
4	Jun	2960
5	Mar	910
6	May	2960
7	Nov	910

Visualization

Table Visualization 1 +



4. What are the yearly and quarterly sales?

a. For Yearly sales

Cmd 13

What are the yearly and quarterly sales?

Cmd 14

For yearly sales:

Cmd 15

```
1 # Calculate the year from the Order Date column
2 cleaned_data = cleaned_data.withColumn("Order Year", F.year("Order Date"))
3
4 # Group by the Order Year and calculate the total sales amount for each year
5 yearly_sales = cleaned_data.groupBy("Order Year").agg(F.sum("Price in $").alias("Total Sales"))
6
7 # Sort the results by Order Year
8 yearly_sales = yearly_sales.sort("Order Year")
9
10 # Display or collect the results
11 display(yearly_sales)
```

► (9) Spark Jobs

► cleaned_data: pyspark.sql.dataframe.DataFrame = [Id: integer, Customer Name: string ... 12 more fields]

► yearly_sales: pyspark.sql.dataframe.DataFrame = [Order Year: integer, Total Sales: double]

Table ▾

Visualization 1

+

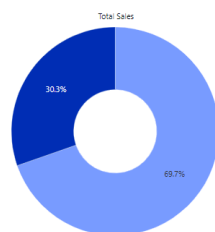
	Order Year ▲	Total Sales ▲
1	2022	4350
2	2023	9990

⬇ 2 rows | 4.19 seconds runtime

Command took 4.19 seconds -- by raiprabesh789@gmail.com at 3/1/2024, 3:27:21 AM on My Cluster

Visualization for yearly sales

Table Visualization 1 ▾ +



■ 2023
■ 2022

b. For quarterly sales

Cmd 16

For quarterly sales:

Cmd 17

```
1 # Calculate the quarter from the Order Date column
2 cleaned_data = cleaned_data.withColumn("Order quarter", F.quarter("Order Date"))
3
4 # Group by the Order Year and Order quarter and calculate the total sales amount for each quarter
5 quarterly_sales = cleaned_data.groupBy("Order Year", "Order quarter").agg(F.sum("price in $").alias("Total Sales"))
6
7 # Sort the results by Order Year and Order quarter
8 quarterly_sales = quarterly_sales.sort("Order Year", "Order quarter")
9
10 # Add "Quarter" before each count of quarter
11 quarterly_sales = quarterly_sales.withColumn("Order quarter", F.concat(F.lit("Quarter "), quarterly_sales["Order quarter"]))
12
13 # Display or collect the results
14 display(quarterly_sales)
```

► (9) Spark Jobs

► cleaned_data: pyspark.sql.dataframe.DataFrame = [Id: integer, Customer Name: string ... 13 more fields]

► quarterly_sales: pyspark.sql.dataframe.DataFrame = [Order Year: integer, Order quarter: string ... 1 more field]

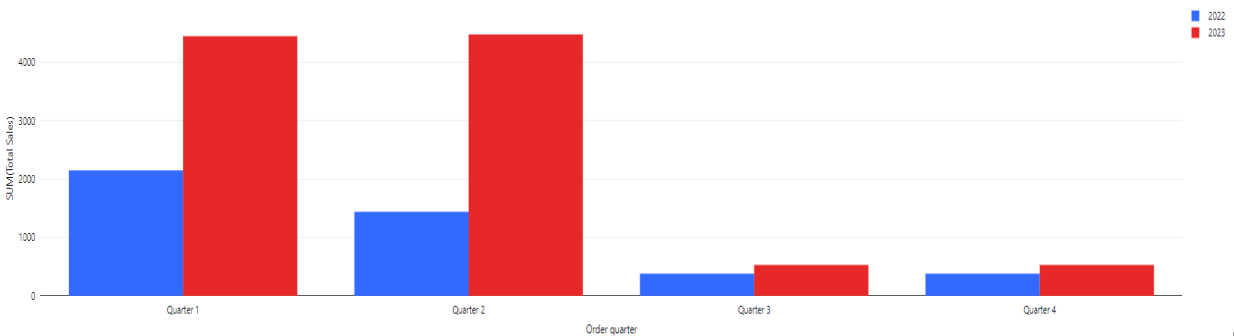
Table Visualization 1 +

	Order Year	Order quarter	Total Sales
1	2022	Quarter 1	2150
2	2022	Quarter 2	1440
3	2022	Quarter 3	380
4	2022	Quarter 4	380
5	2023	Quarter 1	4450
6	2023	Quarter 2	4480
7	2023	Quarter 3	530
8	2023	Quarter 4	530

8 rows | 3.10 seconds runtime

Visualization for quarterly sales

Table Visualization 1 +



5. What are the top-ordered food items?

Cmd 18

Top Ordered Food Items

Cmd 19

```
1 from pyspark.sql.functions import count
2
3 # Make all item names capital, group by the item name, and calculate the order count for each item
4 top_ordered_items = (cleaned_data
5                       .withColumn("Item Name", upper(cleaned_data["Item Name"]))
6                       .groupBy("Item Name")
7                       .agg(count("*").alias("Order Count"))
8                       .orderBy("Order Count", ascending=False))
9 display(top_ordered_items)
```

► (9) Spark Jobs

► top_ordered_items: pyspark.sql.dataframe.DataFrame = [Item Name: string, Order Count: long]

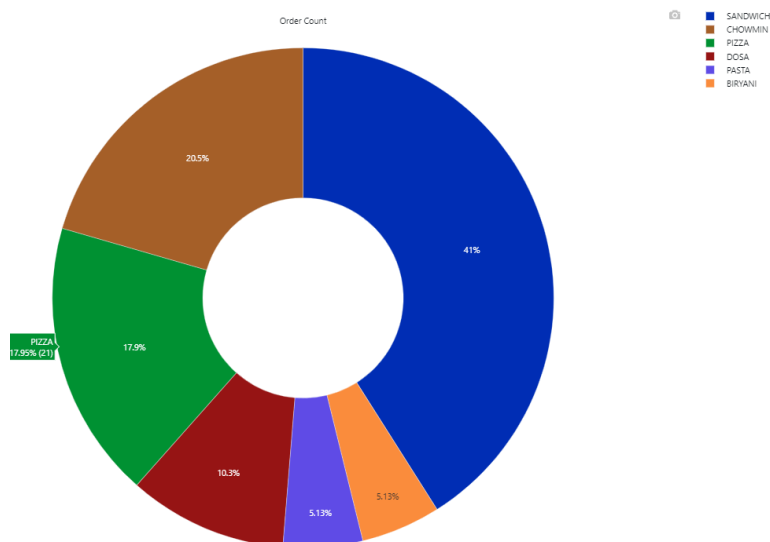
Table Visualization 1 +

	Item Name	Order Count
1	SANDWICH	48
2	CHOWMIN	24
3	PIZZA	21
4	DOSA	12
5	PASTA	6
6	BIRYANI	6

↓ 6 rows | 2.57 seconds runtime

Command took 2.57 seconds -- by raiprabesh789@gmail.com at 3/1/2024, 3:48:50 AM on My Cluster

Visualization



6. What are the total sales by each country?


Cmd 20


What are the total sales by each country?

Cmd 21

```
1 from pyspark.sql.functions import split, sum
2
3 # Extract country information from the location column
4 cleaned_data = cleaned_data.withColumn("country", split(cleaned_data["location"], ",").getItem(0))
5
6 # Group by the country and calculate the total sales amount for each country
7 total_sales_by_country = cleaned_data.groupBy("country").agg(sum("Price in $").alias("Total Sales"))
8
9 # Display the results
10 display(total_sales_by_country)
```

▶ (3) Spark Jobs

▶  cleaned_data: pyspark.sql.dataframe.DataFrame = [Id: integer, Customer Name: string ... 7 more fields]

▼  total_sales_by_country: pyspark.sql.dataframe.DataFrame

country: string
Total Sales: double

Table ▾

Visualization 1

+

Visualization

