DSC 650-T302 BIG DATA

DIPIKA SHARMA

Table of Contents

Week 11 & 12 Final Project
Black Friday Sales

Introduction:

Black Friday is an important day for customer and retailers. Customers prepare list of expensive item electronics, furniture and others and plan to buy them during Black Friday at low prices whereas retailers look forward to this day to increase their sales, clear inventory and to add new customers.

Objective:

For term project I choose Black Friday sales data to understand the customer behavior. This analysis will help the retailers to understand their customer and can come up with list of optimize inventory and improve marketing strategies to increase their sales and attract more customers.
To achieve the goal of the project my plan is to utilize big data analytics on Black Friday Sales data to gather important information about the customer and can help business to develop strategies to meet customer need , provide better services and increase revenue for the retailers.

Data Source:

I found the dataset from Kaggle. This data is from retail company “ABC Private limited”. This dataset is showing the black Friday sales by customer against various products of different categories. This dataset contains the demographic information of the customer like age, gender, marital status, and current city type. For privacy purpose we have used the masked data for product category. Data can be easily found at below location:

https://www.kaggle.com/datasets/prepinstaprime/black-friday-sales-data

The dataset contained below columns:

User_ID	: Customer unique Identification number.
Product_ID	: Product unique identification number.
Gender	: Gender of the customer.
Age	: Customer Age in bins
Occupation	: Customer Occupation (Masked)
City_Category	: Customer Category of the City (A,B,C)

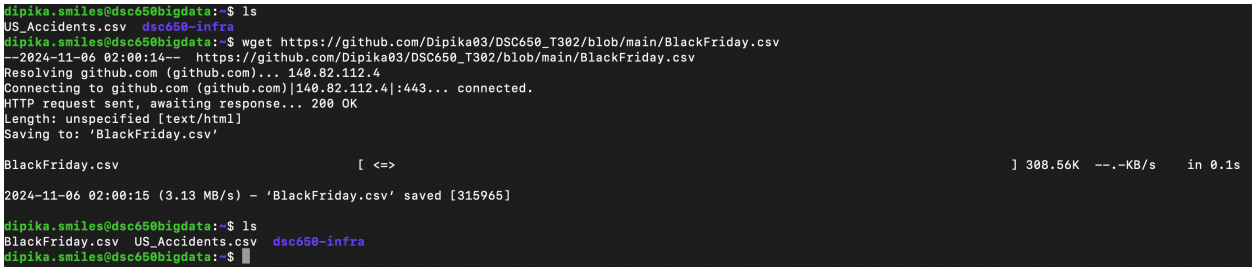
Stay_In_Current_City_Years : Number of years stay in current city
Marital_Status : Customer Marital Status
Product_Category_1 : Product Category (Masked)
Product_Category_2 : Product may belong to another category also (Masked)
Product_Category_3 : Product may belong to another category also (Masked)
Purchase : Total Purchase Amount (Target Variable) by Customer

Data Ingestion:

Data consumption is the first step where we need to import the selected dataset into HDFS. HDFS stands for Hadoop Distributed System, HDFS can store large amount of structured and unstructured data. To upload the CSV data file, we will start the google cloud virtual machine and connect it with the HDFS instance.

In order to do that we first download the csv data file from the Kaggle website and save it to the GitHub repository. And then using the below command upload the csv file into our virtual machine.

- Getting the csv file from GitHub directory to virtual machine directory.
  wget https://github.com/Dipika03/DSC650_T302/blob/main/BlackFriday.csv



HDFS for Data Storage:

Once I have uploaded the csv file to virtual machine, next step is to start the docker container and accessing the master container. After that move the data file to HDFs data directory.

- Start your Docker containers:

  docker-compose up -d

- Access the master container:

  docker-compose exec master bash

- Load the BFriday.csv into HDFS

  hdfs dfs -put /data/BFriday.csv /data/ BFriday.csv

```
bash-5.0#  hdfs dfs -put /data/BFriday.csv /data/BFriday.csv
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/program/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/Stat
icLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/program/tez/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/program/hive/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4j/impl/StaticLoggerBinder.clas
s]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
2024-11-09 00:13:26,370 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin
-java classes where applicable
put: `/data/BFriday.csv': File exists
bash-5.0#
```

- Check if file is loaded

  hdfs dfs -ls /

```
bash-5.0#  hdfs dfs -ls /
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/program/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.cla
SLF4J: Found binding in [jar:file:/usr/program/tez/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/program/hive/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
2024-11-08 00:34:14,840 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes wher
Found 8 items
-rw-r--r--   1 root supergroup       7112 2024-11-08 00:24 /BFriday.csv
drwxr-xr-x   - root supergroup          0 2024-11-07 01:17 /hbase
drwxr-xr-x   - root supergroup          0 2024-11-06 02:38 /log
drwxr-xr-x   - root supergroup          0 2024-11-06 02:39 /spark-jars
drwxr-xr-x   - root supergroup          0 2024-11-06 02:40 /tez
drwxr-xr-x   - root supergroup          0 2024-11-07 01:18 /tmp
drwxrwx---   - root supergroup          0 2024-11-06 02:38 /user
drwxr-xr-x   - root supergroup          0 2024-11-07 01:34 /usr
```

Hive for Data warehousing:

a) Table Creation

Once the data upload to HDFS, the next step is to move the data to hive table. This can be achieved by starting the hive session by using below command:

- Start a Hive session
  Hive

- In the Hive CLI, create a table:

  CREATE TABLE BFridaySales(
  `User_ID` INT,
  `Product_ID` STRING,
  `Gender` STRING,
  `Age` DOUBLE,
  `Occupation` INT,

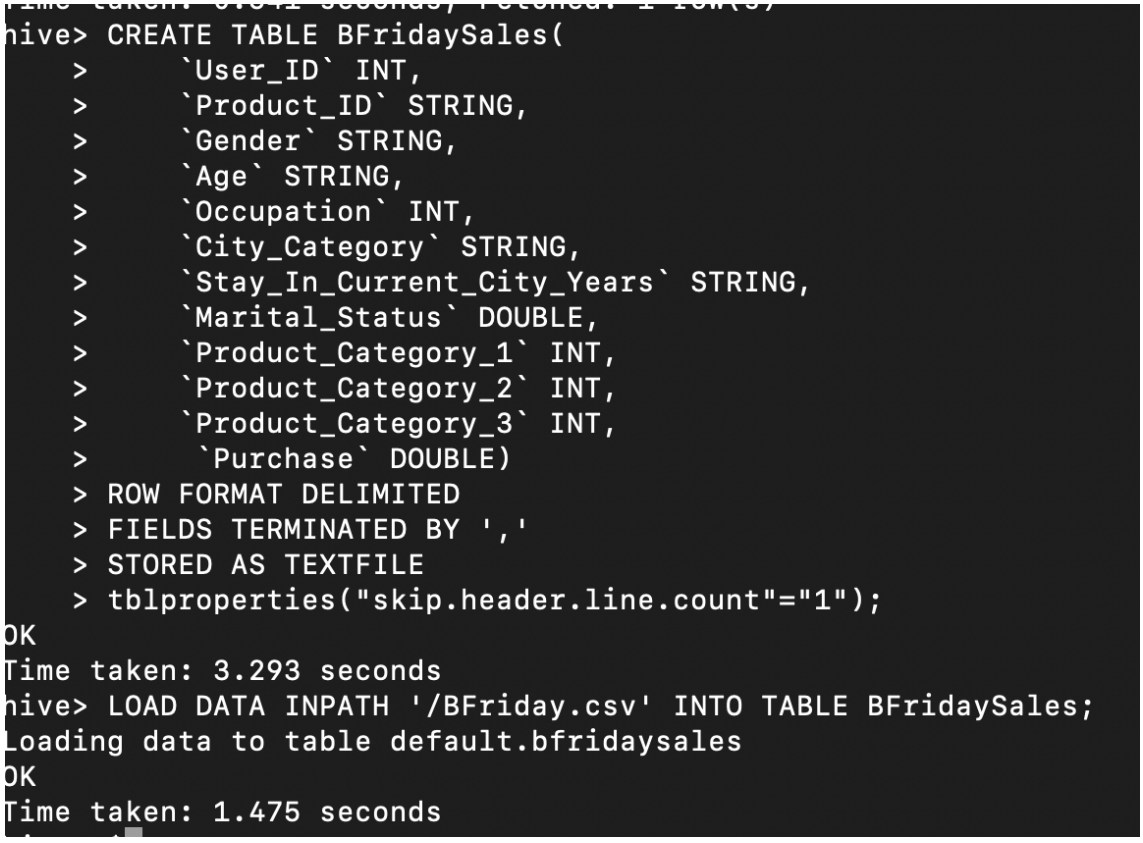
```
    `City_Category` STRING,

    `Stay_In_Current_City_Years` STRING,

    `Marital_Status` DOUBLE,

    `Product_Category_1` INT,

    `Product_Category_2` INT,
    `Product_Category_3` INT,
    `Purchase` DOUBLE)
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
  STORED AS TEXTFILE
tblproperties("skip.header.line.count"="1");
```



```
hive> CREATE TABLE BFridaySales(
    >     `User_ID` INT,
    >     `Product_ID` STRING,
    >     `Gender` STRING,
    >     `Age` STRING,
    >     `Occupation` INT,
    >     `City_Category` STRING,
    >     `Stay_In_Current_City_Years` STRING,
    >     `Marital_Status` DOUBLE,
    >     `Product_Category_1` INT,
    >     `Product_Category_2` INT,
    >     `Product_Category_3` INT,
    >      `Purchase` DOUBLE)
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE
    > tblproperties("skip.header.line.count"="1");
OK
Time taken: 3.293 seconds
hive> LOAD DATA INPATH '/BFriday.csv' INTO TABLE BFridaySales;
Loading data to table default.bfridaysales
OK
Time taken: 1.475 seconds
```

b) Data Loading

Once the table created, next step is to load the data into the table

- LOAD DATA INPATH '/BFriday.csv' INTO TABLE BFridaySales;

Let's run a query to see how many rows we have in our dataset using aggregate count function.

- SELECT count(*) FROM BFridaySales;

```
hive> Select count(*) from BFridaySales;
2024-11-08 00:40:04,551 INFO  [8fc5ec6a-d989-4a24-82a3-1619fed97f94 main] reducesink.VectorReduceSinkEmptyKeyOperator: VectorReduceSinkEmptyK
 org.apache.hadoop.hive.ql.plan.VectorReduceSinkInfo@2f498f21
Query ID = root_20241108004003_f0b5c906-bc05-4dc2-83f1-b3b580d905a3
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1731026053949_0003)

----------------------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 8.04 s
----------------------------------------------------------------------------------------------
OK
149
Time taken: 9.747 seconds, Fetched: 1 row(s)
hive>
```

As we can see above, query result show we have 149 rows in our Black Friday dataset.

Running describe function to know about the dataset columns

- describe BFridaySales;

```
hive> Describe BFridaySales;
OK
user_id                 int
product_id              string
gender                  string
age                     string
occupation              int
city_category           string
stay_in_current_city_years      string
marital_status          double
product_category_1      int
product_category_2      int
product_category_3      int
purchase                double
Time taken: 0.154 seconds, Fetched: 12 row(s)
hive>
```

The above result is showing the different columns we have in our dataset also it is showing what data type each column stores.

Let's begin the analysis to understand how much total purchase was done by customers on Black Friday day.

- Select sum(purchase) as TotalPurchase from BFridaySales;

```
hive> Select sum(purchase) as TotalPurchase from BFridaySales;
2024-11-08 00:43:43,070 INFO  [8fc5ec6a-d989-4a24-82a3-1619fed97f94 main] reducesink.VectorReduceSinkEmptyKeyOperator: VectorReduceSinkEmptyKeyOperator const
ructor vectorReduceSinkInfo org.apache.hadoop.hive.ql.plan.VectorReduceSinkInfo@5c4516ec
Query ID = root_20241108004342_22cdc6e9-263c-44df-a8db-9dca93cab56b
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1731026053949_0003)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........  container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ......  container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 5.23 s
--------------------------------------------------------------------------------
OK
1453324.0
Time taken: 6.655 seconds, Fetched: 1 row(s)
hive>
```

The result above shows the total purchase of $1,45,3324 was done by customer.

To further analyze the data, it's important to understand the customer. Let's see the total purchase by gender.

- Select gender, sum(purchase) as TotalPurchase from BFridaySales group by gender;

```
hive> Select gender, sum(purchase) as TotalPurchase from BFridaySales group by gender
    > ;
Query ID = root_20241108004548_1368fd47-3f0c-420e-9fe8-e222e1bfeabb
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1731026053949_0003)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........  container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ......  container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 7.56 s
--------------------------------------------------------------------------------
OK
F       506041.0
M       947283.0
Time taken: 8.752 seconds, Fetched: 2 row(s)
hive>
```

The above result shows that $947,283 was spent by males and $506,041 was spent by females. It is interesting to know that male customer spent more compared to females during black Friday.

Now let's learn about the customer more by knowing how much purchased was done on Black Friday by city category.

- Select city_Category, sum(purchase) as TotalPurchase from BFridaySales group by city_category;

```
hive> Select city_category, sum(purchase) as TotalPurchase from BFridaySales group by city_category;
Query ID = root_20241108004638_cfb93a6e-a22a-4ec5-affa-d97ac89775bc
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1731026053949_0003)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 8.37 s
--------------------------------------------------------------------------------
OK
A       434101.0
B       638735.0
C       380488.0
Time taken: 9.351 seconds, Fetched: 3 row(s)
hive>
```

The results show city B spent maximum amount of $638,735 compared to another city A which spent $434,101 and city C which spent $380,488 on black Friday.

For further analysis let's learn customer from which occupation spent most in black Friday.

- Select occupation, sum(purchase) as TotalPurchase from BFridaySales group by occupation order by TotalPurchase DESC limit 5;

```
hive> Select occupation, sum(purchase) as TotalPurchase from BFridaySales group by occupation order by TotalPurchase DESC limit 5;
2024-11-08 00:48:28,752 INFO  [8fc5ec6a-d989-4a24-82a3-1619fed97f94 main] reducesink.VectorReduceSinkObjectHashOperator: VectorReduceSinkObjectHashOperator c
onstructor vectorReduceSinkInfo org.apache.hadoop.hive.ql.plan.VectorReduceSinkInfo@2396408a
Query ID = root_20241108004828_f5f87688-50cf-4cee-86fe-c11ab25597e0
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1731026053949_0003)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
Reducer 3 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 7.13 s
--------------------------------------------------------------------------------
OK
1       328481.0
7       225946.0
3       162046.0
0       160031.0
10      134616.0
Time taken: 8.4 seconds, Fetched: 5 row(s)
hive>
```

Result shows the top 5 occupation of the customer who spent most. Occupation 1 is on the top list as they spent maximum of $328,481 on black Friday compared to other occupation.

Spark:

a) Data Processing:

To understand more about the customer let's use spark for further analysis. Spark runs multiple algorithm or rule-based checks in processing layers to see if data form any pattern and send it downstream for further analysis. The recognizing of the patterns will help us to understand customer behavior to know if there is any product which are in demand, or we can learn about customer interest and make changes to the inventory or plan strategies for those customers.
The csv file data is already in hdfs so next step is to enter the spark shell

- spark-shell

```
bash-5.0# spark-shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/program/spark/jars/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/program/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
0    [main] WARN  org.apache.hadoop.util.NativeCodeLoader  - Unable to load native-hadoop library for your platform... using builtin-java classes
cable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://localhost:4040
Spark context available as 'sc' (master = yarn, app id = application_1731026053949_0004).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.0.0
      /_/

Using Scala version 2.12.10 (OpenJDK 64-Bit Server VM, Java 1.8.0_275)
Type in expressions to have them evaluated.
Type :help for more information.

scala>

scala>
```

Let's run the following SparkSQL commands created data frame df using Black Friday csv data file.

- **val** df = spark.read.format("csv").option("header", "true").load("/data/BFriday.csv")
  df.createOrReplaceTempView("df")

  spark.sql("SHOW TABLES").show()

```
>>> df.createOrReplaceTempView('df')
>>> spark.sql('SHOW TABLES').show()
444122 [Thread-4] WARN  org.apache.hadoop.hive.conf.HiveConf  - HiveConf of name hive.strict.managed.tables does not exist
444122 [Thread-4] WARN  org.apache.hadoop.hive.conf.HiveConf  - HiveConf of name hive.create.as.insert.only does not exist
444224 [Thread-4] WARN  org.apache.spark.sql.hive.client.HiveClientImpl  - Detected HiveConf hive.execution.engine is 'tez' and will be re
et to 'mr' to disable useless hive logic
+--------+------------+-----------+
|database|   tableName|isTemporary|
+--------+------------+-----------+
| default|     bfriday|      false|
| default|bfridaysales|      false|
| default| blackfriday|      false|
|        |          df|       true|
+--------+------------+-----------+

>>>
```

As we can see above df is created. Now let's see data stored in df.

- df.show()

```
>>> df = spark.read.format('csv').option('header', 'true').load('/data/BFriday.csv')
>>> df.show()
+-------+----------+------+-----+----------+-------------+--------------------------+--------------+------------------+------------------+------------------+--------+
|User_ID|Product_ID|Gender|  Age|Occupation|City_Category|Stay_In_Current_City_Years|Marital_Status|Product_Category_1|Product_Category_2|Product_Category_3|Purchase|
+-------+----------+------+-----+----------+-------------+--------------------------+--------------+------------------+------------------+------------------+--------+
|1000001| P00069042|     F| 0-17|        10|            A|                         2|             0|                 3|              null|              null|    8370|
|1000001| P00248942|     F| 0-17|        10|            A|                         2|             0|                 1|                 6|                14|   15200|
|1000001| P00087842|     F| 0-17|        10|            A|                         2|             0|                12|              null|              null|    1422|
|1000001| P00085442|     F| 0-17|        10|            A|                         2|             0|                12|                14|              null|    1057|
|1000002| P00285442|     M|  55+|        16|            C|                        4+|             0|                 8|              null|              null|    7969|
|1000003| P00193542|     M|26-35|        15|            A|                         3|             0|                 1|                 2|              null|   15227|
|1000004| P00184942|     M|46-50|         7|            B|                         2|             1|                 1|                 8|                17|   19215|
|1000004| P00346142|     M|46-50|         7|            B|                         2|             1|                 1|                15|              null|   15854|
|1000004|  P0097242|     M|46-50|         7|            B|                         2|             1|                 1|                16|              null|   15686|
|1000005| P00274942|     M|26-35|        20|            A|                         1|             1|                 8|              null|              null|    7871|
|1000005| P00251242|     M|26-35|        20|            A|                         1|             1|                 5|                11|              null|    5254|
|1000005| P00014542|     M|26-35|        20|            A|                         1|             1|                 8|              null|              null|    3957|
|1000005| P00031342|     M|26-35|        20|            A|                         1|             1|                 8|              null|              null|    6073|
|1000005| P00145042|     M|26-35|        20|            A|                         1|             1|                 1|                 2|                 5|   15665|
|1000006| P00231342|     F|51-55|         9|            A|                         1|             0|                 5|                 8|                14|    5378|
|1000006| P00190242|     F|51-55|         9|            A|                         1|             0|                 4|                 5|              null|    2079|
|1000006|  P0096642|     F|51-55|         9|            A|                         1|             0|                 2|                 3|                 4|   13055|
|1000006| P00058442|     F|51-55|         9|            A|                         1|             0|                 5|                14|              null|    8851|
|1000007| P00036842|     M|36-45|         1|            B|                         1|             1|                 1|                14|                16|   11788|
|1000008| P00249542|     M|26-35|        12|            C|                        4+|             1|                 1|                 5|                15|   19614|
+-------+----------+------+-----+----------+-------------+--------------------------+--------------+------------------+------------------+------------------+--------+
only showing top 20 rows
```

The above command showing the top 20 rows stored in df.

We can also see data by simply running the select query:

- Spark.sql('Select * From df').show()

```
>>> spark.sql('Select * From df').show()
+-------+----------+------+-----+----------+-------------+--------------------------+--------------+------------------+------------------+------------------+--------+
|User_ID|Product_ID|Gender|  Age|Occupation|City_Category|Stay_In_Current_City_Years|Marital_Status|Product_Category_1|Product_Category_2|Product_Category_3|Purchase|
+-------+----------+------+-----+----------+-------------+--------------------------+--------------+------------------+------------------+------------------+--------+
|1000001| P00069042|     F| 0-17|        10|            A|                         2|             0|                 3|              null|              null|    8370|
|1000001| P00248942|     F| 0-17|        10|            A|                         2|             0|                 1|                 6|                14|   15200|
|1000001| P00087842|     F| 0-17|        10|            A|                         2|             0|                12|              null|              null|    1422|
|1000001| P00085442|     F| 0-17|        10|            A|                         2|             0|                12|                14|              null|    1057|
|1000002| P00285442|     M|  55+|        16|            C|                        4+|             0|                 8|              null|              null|    7969|
|1000003| P00193542|     M|26-35|        15|            A|                         3|             0|                 1|                 2|              null|   15227|
|1000004| P00184942|     M|46-50|         7|            B|                         2|             1|                 1|                 8|                17|   19215|
|1000004| P00346142|     M|46-50|         7|            B|                         2|             1|                 1|                15|              null|   15854|
|1000004|  P0097242|     M|46-50|         7|            B|                         2|             1|                 1|                16|              null|   15686|
|1000005| P00274942|     M|26-35|        20|            A|                         1|             1|                 8|              null|              null|    7871|
|1000005| P00251242|     M|26-35|        20|            A|                         1|             1|                 5|                11|              null|    5254|
|1000005| P00014542|     M|26-35|        20|            A|                         1|             1|                 8|              null|              null|    3957|
|1000005| P00031342|     M|26-35|        20|            A|                         1|             1|                 8|              null|              null|    6073|
|1000005| P00145042|     M|26-35|        20|            A|                         1|             1|                 1|                 2|                 5|   15665|
|1000006| P00231342|     F|51-55|         9|            A|                         1|             0|                 5|                 8|                14|    5378|
|1000006| P00190242|     F|51-55|         9|            A|                         1|             0|                 4|                 5|              null|    2079|
|1000006|  P0096642|     F|51-55|         9|            A|                         1|             0|                 2|                 3|                 4|   13055|
|1000006| P00058442|     F|51-55|         9|            A|                         1|             0|                 5|                14|              null|    8851|
|1000007| P00036842|     M|36-45|         1|            B|                         1|             1|                 1|                14|                16|   11788|
|1000008| P00249542|     M|26-35|        12|            C|                        4+|             1|                 1|                 5|                15|   19614|
+-------+----------+------+-----+----------+-------------+--------------------------+--------------+------------------+------------------+------------------+--------+
only showing top 20 rows
```

This command also shows the top 20 rows of the dataset stored in df.

b) Data Querying

Now let's start analyzing the data to learn about the customer. Next thing we can do is to see top 3 customer who purchased most product and what age group they belong to.

- Spark.sql('Select User_ID, age, count(Product_ID) as NumberOfProduct from df group by user_id, age order by NumberOfProduct desc limit 3').show()

```
>>> spark.sql("Select User_ID, Age, count(Product_ID) as NumberOfProduct from df group by user_id, age order by NumberofProduct desc limit 3").show()
+-------+-----+--------------+
|User_ID|  Age|NumberOfProduct|
+-------+-----+--------------+
|1000010|36-45|            18|
|1000018|18-25|            15|
|1000022|18-25|            14|
+-------+-----+--------------+

>>>
```

The above query result shows that customer 100010 purchased most of the product that is 18 and the customer is in age group of 36-45. The next customer who purchased 15 product is in age group 18-25. Also, the third customer who purchased 14 products are also in 10-25 age group.
The Retailers can plan accordingly to make sure they have inventory based on age group. Using this we can see which age group purchased more product and plan strategies to improve performance.

We know we have 149 rows in our dataset, it would be interesting to see how many female and male users we have.
Spark.sql(“Select Gender, count(User_ID) as NumberOfUsers from df group by Gender”).show()

```
>>> spark.sql("Select Gender, count(User_ID) as NumberOfUsers from df group by Gender").show()
+------+-------------+
|Gender|NumberOfUsers|
+------+-------------+
|     F|           51|
|     M|           98|
+------+-------------+
```

The above query result shares the interesting fact that we have 98 male users and 51 female users. We can increase products for male users also can offer some schemes to increase sales.

I think what we do for living also plays an important role in developing interest to certain products. If I am the painter, I will always be looking for some colors and canvas. In order to attract the right audiences, it is important to understand what occupation they are in.

Spark.sql(“Select Gender, occupation, count(User_Id) as NumberOfUsers from df group by Gender, occupation”).show()

```
>>> spark.sql("Select Gender, occupation,  count(User_ID) as NumberOfUsers from df group by Gender, occupation").show()
+------+----------+-------------+
|Gender|occupation|NumberOfUsers|
+------+----------+-------------+
|     M|        20|            5|
|     M|        10|           12|
|     M|         1|            7|
|     M|        17|            4|
|     M|         4|            3|
|     M|         7|           24|
|     F|         7|            3|
|     M|        16|            2|
|     M|        12|            8|
|     F|         1|           23|
|     F|         3|           15|
|     M|        11|            5|
|     F|         9|            4|
|     F|         0|            2|
|     M|         0|           13|
|     F|        10|            4|
|     M|        15|           15|
+------+----------+-------------+
```

As we can see we have 24 male customers who all are in occupation 7, also we can see that there are 3 female customer who are in same occupation 7.

I also ran another query to see which occupation has most users.

Spark.sql(“Select occupation, count(User_Id) as NumberOfUsers from df group by occupation”).show()

```
>>> spark.sql("Select occupation,  count(User_ID) as NumberOfUsers from df group by occupation").show()
+----------+-------------+
|occupation|NumberOfUsers|
+----------+-------------+
|         7|           27|
|        15|           15|
|        11|            5|
|         3|           15|
|        16|            2|
|         0|           15|
|        17|            4|
|         9|            4|
|         1|           30|
|        20|            5|
|        10|           16|
|         4|            3|
|        12|            8|
+----------+-------------+
```

Occupation 1 has most of 30 users, 27 customers are in occupation 7. And occupation 10, 3 and 15 also has more customers.

c) Spark sql query using pyspark

Start the HBase Thrift server to allow Python to connect via happybase. This command will be run in the master docker container & runs the Thrift server in the background.

- hbase thrift start

Open a Pyspark session and use the following code to write data to HBase

- import happybase
  from pyspark.sql import SparkSession

  # Initialize SparkSession
  spark = SparkSession.builder \
  .appName('WriteToHBaseWithHappybase') \
  .getOrCreate()

Let's see if we can see data

- df = spark.sql("Select * From BFridaySales")
- df.show()

```
>>> from pyspark.sql import SparkSession
>>> spark = SparkSession.builder     .appName("PySpark Hive Example")     .enableHiveSupport()     .getOrCreate()
>>> df = spark.sql("SELECT * FROM BFridaySales")
518903 [Thread-4] WARN  org.apache.hadoop.hive.conf.HiveConf  - HiveConf of name hive.strict.managed.tables does not exist
518903 [Thread-4] WARN  org.apache.hadoop.hive.conf.HiveConf  - HiveConf of name hive.create.as.insert.only does not exist
518970 [Thread-4] WARN  org.apache.spark.sql.hive.client.HiveClientImpl  - Detected HiveConf hive.execution.engine is 'tez' and will be reset to 'mr' to disable useless hive logic
df.show()>>> df.show()
+-------+----------+------+-----+----------+-------------+--------------------------+--------------+-----------------+-----------------+-----------------+--------+
|user_id|product_id|gender|  age|occupation|city_category|stay_in_current_city_years|marital_status|product_category_1|product_category_2|product_category_3|purchase|
+-------+----------+------+-----+----------+-------------+--------------------------+--------------+-----------------+-----------------+-----------------+--------+
|   null|Product_ID|Gender|  Age|      null|City_Category|      Stay_In_Current_C...|          null|             null|             null|             null|    null|
|1000001| P00069042|     F| 0-17|        10|            A|                         2|           0.0|                3|             null|             null|  8370.0|
|1000001| P00248942|     F| 0-17|        10|            A|                         2|           0.0|                1|                6|               14| 15200.0|
|1000001| P00087842|     F| 0-17|        10|            A|                         2|           0.0|               12|             null|             null|  1422.0|
|1000001| P00085442|     F| 0-17|        10|            A|                         2|           0.0|               12|               14|             null|  1057.0|
|1000002| P00285442|     M|  55+|        16|            C|                        4+|           0.0|                8|             null|             null|  7969.0|
|1000003| P00193542|     M|26-35|        15|            A|                         3|           0.0|                1|                2|             null| 15227.0|
|1000004| P00184942|     M|46-50|         7|            B|                         2|           1.0|                1|                8|               17| 19215.0|
|1000004| P00346142|     M|46-50|         7|            B|                         2|           1.0|                1|               15|             null| 15854.0|
|1000004|  P0097242|     M|46-50|         7|            B|                         2|           1.0|                1|               16|             null| 15686.0|
|1000005| P00274942|     M|26-35|        20|            A|                         1|           1.0|                8|             null|             null|  7871.0|
|1000005| P00251242|     M|26-35|        20|            A|                         1|           1.0|                5|               11|             null|  5254.0|
|1000005| P00014542|     M|26-35|        20|            A|                         1|           1.0|                8|             null|             null|  3957.0|
|1000005| P00031342|     M|26-35|        20|            A|                         1|           1.0|                8|             null|             null|  6073.0|
|1000005| P00145042|     M|26-35|        20|            A|                         1|           1.0|                1|                2|                5| 15665.0|
|1000006| P00231342|     F|51-55|         9|            A|                         1|           0.0|                5|                8|               14|  5378.0|
|1000006| P00190242|     F|51-55|         9|            A|                         1|           0.0|                4|                5|             null|  2079.0|
|1000006|  P0096642|     F|51-55|         9|            A|                         1|           0.0|                2|                3|                4| 13055.0|
|1000006| P00058442|     F|51-55|         9|            A|                         1|           0.0|                5|               14|             null|  8851.0|
|1000007| P00036842|     M|36-45|         1|            B|                         1|           1.0|                1|               14|               16| 11788.0|
+-------+----------+------+-----+----------+-------------+--------------------------+--------------+-----------------+-----------------+-----------------+--------+
only showing top 20 rows
```

The result is showing top 20 rows of the table BFridaySales.

Now it's time to learn which users spent most on Black Friday day. Let's see the top 5 customer who have spent most.

```
>>> df = spark.sql("Select user_id, sum(purchase) as TotalPurchase from BFridaySales group by user_id order by TotalPurchase DESC limit 5")
>>> df.show()
+-------+-------------+
|user_id|TotalPurchase|
+-------+-------------+
|1000010|     201995.0|
|1000018|     162046.0|
|1000023|     133726.0|
|1000022|     116254.0|
|1000019|     108567.0|
+-------+-------------+

>>>
```

The result shows that the customer with user id 100010 has spent most of $201,995 on Black Friday day. Also, there are other customer 100018, 100023, 100022, and 100019 who have spent significant amount of money during black Friday. Maybe we can give them some discounts or some scheme to make them happy so that they continue being our loyal customers.

It would be beneficial to know which age group have spent most on black Friday.

- df = spark.sql("Select age, sum(purchase) as TotalPurchase From BFridaySales group by age order by TotalPut=rchase DESC limit 5")
- df.show()

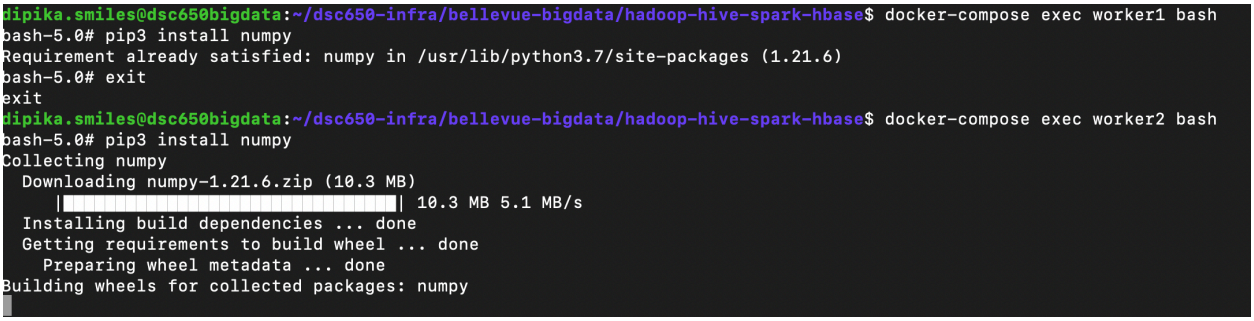
```
>>> df = spark.sql("Select age, sum(purchase) as TotalPurchase from BFridaySales group by age order by TotalPurchase DESC limit 5")
>>> df.show()
+-----+-------------+
|  age|TotalPurchase|
+-----+-------------+
|26-35|     451836.0|
|36-45|     373814.0|
|18-25|     331460.0|
| 0-17|     134616.0|
|46-50|      84613.0|
+-----+-------------+
```

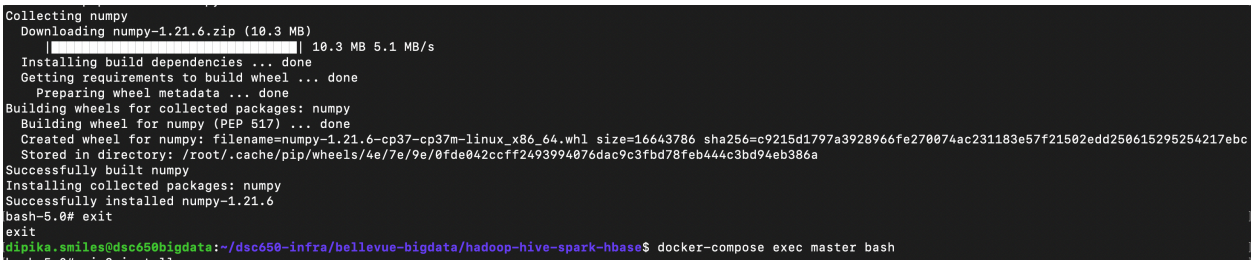
The age group 26-35 have spent most of $451,836. the other age group 36-45 and 18-25 are also close to top group. We can make changes to inventory accordingly. To make sure we have everything to attract those age group customers.

d) Machine Learning with Spark MLIB and PySpark:

Let's log in to Worker Nodes and Install the Library.

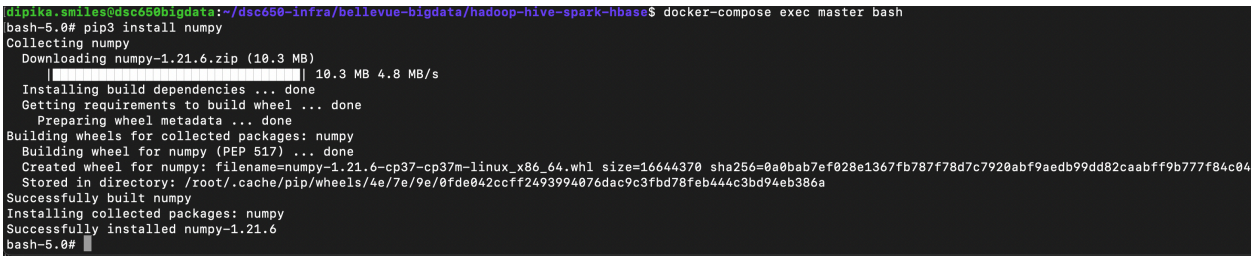
- For Worker Node 1:
  docker-compose exec worker1 bash
- Now we are installing the library using pip3 and exit the worker node 1
  pip3 install numpy
  Exit
- Repeat the steps for Worker Node 2
  docker-compose exec worker2 bash
  pip3 install numpy
  Exit





Now we Log in to the Master Node and Install the Library

- Log in to the Master Node:
  docker-compose exec master bash
- Install the Library using pip3:
  pip3 install numpy



Let get into the pyspark

- pyspark

Let's run some analysis

- Spark.sql("Select user_id, age, occupation From BFridaySales")

```
>>> BFriday_df = spark.sql("SELECT user_id, age, occupation FROM BFridaySales")
>>> BFriday_df = BFriday_df.na.drop()
>>> BFriday_df.show()
+-------+-----+----------+
|user_id|  age|occupation|
+-------+-----+----------+
|1000001| 0-17|        10|
|1000001| 0-17|        10|
|1000001| 0-17|        10|
|1000001| 0-17|        10|
|1000002|  55+|        16|
|1000003|26-35|        15|
|1000004|46-50|         7|
|1000004|46-50|         7|
|1000004|46-50|         7|
|1000005|26-35|        20|
|1000005|26-35|        20|
|1000005|26-35|        20|
|1000005|26-35|        20|
|1000005|26-35|        20|
|1000006|51-55|         9|
|1000006|51-55|         9|
|1000006|51-55|         9|
|1000006|51-55|         9|
|1000007|36-45|         1|
|1000008|26-35|        12|
+-------+-----+----------+
only showing top 20 rows
```

This is showing top 20 rows of the BFridaySales table.

Running query to see what age group customer purchasing what product and spending how much money on that.

```
>>> BFriday_df = spark.sql("SELECT user_id, product_id, age, occupation, purchase FROM BFridaySales")
>>> BFriday_df = BFriday_df.na.drop()
>>> BFriday_df.show()
+-------+----------+-----+----------+--------+
|user_id|product_id|  age|occupation|purchase|
+-------+----------+-----+----------+--------+
|1000001| P00069042| 0-17|        10|  8370.0|
|1000001| P00248942| 0-17|        10| 15200.0|
|1000001| P00087842| 0-17|        10|  1422.0|
|1000001| P00085442| 0-17|        10|  1057.0|
|1000002| P00285442|  55+|        16|  7969.0|
|1000003| P00193542|26-35|        15| 15227.0|
|1000004| P00184942|46-50|         7| 19215.0|
|1000004| P00346142|46-50|         7| 15854.0|
|1000004|  P0097242|46-50|         7| 15686.0|
|1000005| P00274942|26-35|        20|  7871.0|
|1000005| P00251242|26-35|        20|  5254.0|
|1000005| P00014542|26-35|        20|  3957.0|
|1000005| P00031342|26-35|        20|  6073.0|
|1000005| P00145042|26-35|        20| 15665.0|
|1000006| P00231342|51-55|         9|  5378.0|
|1000006| P00190242|51-55|         9|  2079.0|
|1000006|  P0096642|51-55|         9| 13055.0|
|1000006| P00058442|51-55|         9|  8851.0|
|1000007| P00036842|36-45|         1| 11788.0|
|1000008| P00249542|26-35|        12| 19614.0|
+-------+----------+-----+----------+--------+
only showing top 20 rows
```

From above result we can gather information like customer with user id 1000001 purchased 4 different products. We can run analysis to create a link between customer and product to see how frequently customer is buying same product in what time span and create plan to improve sales.

Running another analysis to see how many products one customer is buying and how much total purchased the customer is making on black Friday.

- Spark.sql(“Select user_id, occupation, purchase from BFridaySales”)

```
>>> BFriday_df = spark.sql("SELECT user_id, occupation, purchase FROM BFridaySales")
>>> BFriday_df = BFriday_df.na.drop()
>>> BFriday_df.show()
+-------+----------+--------+
|user_id|occupation|purchase|
+-------+----------+--------+
|1000001|        10|  8370.0|
|1000001|        10| 15200.0|
|1000001|        10|  1422.0|
|1000001|        10|  1057.0|
|1000002|        16|  7969.0|
|1000003|        15| 15227.0|
|1000004|         7| 19215.0|
|1000004|         7| 15854.0|
|1000004|         7| 15686.0|
|1000005|        20|  7871.0|
|1000005|        20|  5254.0|
|1000005|        20|  3957.0|
|1000005|        20|  6073.0|
|1000005|        20| 15665.0|
|1000006|         9|  5378.0|
|1000006|         9|  2079.0|
|1000006|         9| 13055.0|
|1000006|         9|  8851.0|
|1000007|         1| 11788.0|
|1000008|        12| 19614.0|
+-------+----------+--------+
only showing top 20 rows
```

From above query result we can see customer with user od 100001 is in occupation 10 and have made multiple purchase on black Friday.

e) PySpark – Mlib Linear Regression

Using the PySpark to predict the purchase using MLlib Linear Regression

- from pyspark.sql import SparkSession
  from pyspark.ml.feature import VectorAssembler
  from pyspark.ml.regression import LinearRegression

```
>>> from pyspark.sql import SparkSession
>>> from pyspark.ml.feature import VectorAssembler
>>> from pyspark.ml.regression import LinearRegression
```

Prepare the data for MLlib by assembling features into a vector

- assembler = VectorAssembler(
    inputCols=["test1", "test2", "test3", "test4"],
    outputCol="features",
    handleInvalid="skip" # Skip rows with null values
)
  assembled_df = assembler.transform(BFriday_df).select("features", "purchase")

Split the data into training and testing sets

- train_data, test_data = assembled_df.randomSplit([0.7, 0.3])

Initialize and train a Linear Regression model

- lr = LinearRegression(labelCol="purchase")
  lr_model = lr.fit(train_data)

```
>>> assembler = VectorAssembler(
...     inputCols=["user_id", "occupation"],
...     outputCol="features",
...     handleInvalid="skip"  # Skip rows with null values
... )
>>> assembled_df = assembler.transform(BFriday_df).select("features", "purchase")
>>> train_data, test_data = assembled_df.randomSplit([0.7, 0.3])
>>> lr = LinearRegression(labelCol="purchase")
>>> lr_model = lr.fit(train_data)
```

Evaluate the model on the test data

- test_results = lr_model.evaluate(test_data)

Print the model performance metrics

-
  print(f"RMSE: {test_results.rootMeanSquaredError}")
  print(f"R^2: {test_results.r2}")

```
>>> test_results = lr_model.evaluate(test_data)
>>> print(f"RMSE: {test_results.rootMeanSquaredError}")
RMSE: 5462.308779262024
>>> print(f"R^2: {test_results.r2}")
R^2: 0.005225768027638633
```

RMSE - The average error between the predictions and actuals in this Black Friday dataset is 5462 which is lower than the average purchase of 9754 which means linear regression model fit the dataset, but we can also look for better model for better prediction.
If we look at the R2 value which is lower than 1 but not very close to 1 which indicate that the linear regression line does not fir the data. It might be because we are not using a complete dataset and with current dataset, we are not able to create a relation between independent and dependent variable.

Stop the Spark session

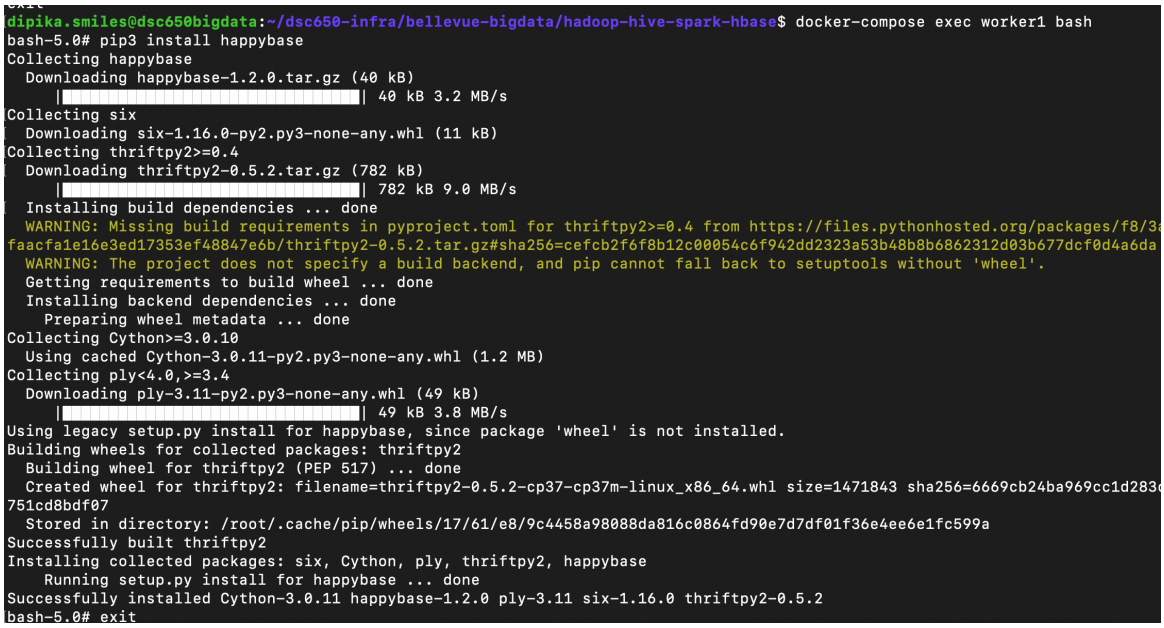
- spark.stop()

HBase with Pyspark:

a) Initialization – Table creation and data loading

- Log in to Worker Node 1 and install the library using pip3
  docker-compose exec worker1 bash
  pip3 install happybase

- Exit from the worker node 1 now
  exit



- Repeat the steps again for worker node 2

  docker-compose exec worker2 bash
  pip3 install happybase
  exit

```
dipika.smiles@dsc650bigdata:~/dsc650-infra/bellevue-bigdata/hadoop-hive-spark-hbase$ docker-compose exec worker2 bash
bash-5.0# pip3 install happybase
Collecting happybase
  Downloading happybase-1.2.0.tar.gz (40 kB)
     |████████████████████████████████| 40 kB 2.8 MB/s
Collecting six
  Downloading six-1.16.0-py2.py3-none-any.whl (11 kB)
Collecting thriftpy2>=0.4
  Downloading thriftpy2-0.5.2.tar.gz (782 kB)
     |████████████████████████████████| 782 kB 6.9 MB/s
  Installing build dependencies ... done
  WARNING: Missing build requirements in pyproject.toml for thriftpy2>=0.4 from https://files.pythonhosted.org/packages/f8/3a/d983b26df17583a3cc865a9e1737bb8
faacfa1e16e3ed17353ef48847e6b/thriftpy2-0.5.2.tar.gz#sha256=cefcb2f6f8b12c00054c6f942dd2323a53b48b8b6862312d03b677dcf0d4a6da (from happybase).
  WARNING: The project does not specify a build backend, and pip cannot fall back to setuptools without 'wheel'.
  Getting requirements to build wheel ... done
  Installing backend dependencies ... done
    Preparing wheel metadata ... done
Collecting Cython>=3.0.10
  Using cached Cython-3.0.11-py2.py3-none-any.whl (1.2 MB)
Collecting ply<4.0,>=3.4
  Downloading ply-3.11-py2.py3-none-any.whl (49 kB)
     |████████████████████████████████| 49 kB 5.6 MB/s
Using legacy setup.py install for happybase, since package 'wheel' is not installed.
Building wheels for collected packages: thriftpy2
  Building wheel for thriftpy2 (PEP 517) ... done
  Created wheel for thriftpy2: filename=thriftpy2-0.5.2-cp37-cp37m-linux_x86_64.whl size=1471837 sha256=ebab24269085e9440675e31bb251663e939055ac5af8ecf7fd30a
600aac4d5f9
  Stored in directory: /root/.cache/pip/wheels/17/61/e8/9c4458a98088da816c0864fd90e7d7df01f36e4ee6e1fc599a
Successfully built thriftpy2
Installing collected packages: six, Cython, ply, thriftpy2, happybase
    Running setup.py install for happybase ... done
Successfully installed Cython-3.0.11 happybase-1.2.0 ply-3.11 six-1.16.0 thriftpy2-0.5.2
bash-5.0#
```

- Let's log into the Master node now and install the library

  docker-compose exec master bash
  pip3 install happybase

```
dipika.smiles@dsc650bigdata:~/dsc650-infra/bellevue-bigdata/hadoop-hive-spark-hbase$ docker-compose exec master bash
bash-5.0# pip3 install happybase
Collecting happybase
  Downloading happybase-1.2.0.tar.gz (40 kB)
     |████████████████████████████████| 40 kB 2.9 MB/s
Collecting six
  Downloading six-1.16.0-py2.py3-none-any.whl (11 kB)
Collecting thriftpy2>=0.4
  Downloading thriftpy2-0.5.2.tar.gz (782 kB)
     |████████████████████████████████| 782 kB 8.1 MB/s
  Installing build dependencies ... done
  WARNING: Missing build requirements in pyproject.toml for thriftpy2>=0.4 from https://files.pythonhosted.org/packages/f8/3a/d983b26df17583a3cc865a9e1737bb8
faacfa1e16e3ed17353ef48847e6b/thriftpy2-0.5.2.tar.gz#sha256=cefcb2f6f8b12c00054c6f942dd2323a53b48b8b6862312d03b677dcf0d4a6da (from happybase).
  WARNING: The project does not specify a build backend, and pip cannot fall back to setuptools without 'wheel'.
  Getting requirements to build wheel ... done
  Installing backend dependencies ... done
    Preparing wheel metadata ... done
Collecting Cython>=3.0.10
  Using cached Cython-3.0.11-py2.py3-none-any.whl (1.2 MB)
Collecting ply<4.0,>=3.4
  Downloading ply-3.11-py2.py3-none-any.whl (49 kB)
     |████████████████████████████████| 49 kB 5.4 MB/s
Using legacy setup.py install for happybase, since package 'wheel' is not installed.
Building wheels for collected packages: thriftpy2
  Building wheel for thriftpy2 (PEP 517) ... done
  Created wheel for thriftpy2: filename=thriftpy2-0.5.2-cp37-cp37m-linux_x86_64.whl size=1471849 sha256=aa6d879493a00d6e8f059ff25eba7b165866a042e704ad80df735
c2812d8784e
  Stored in directory: /root/.cache/pip/wheels/17/61/e8/9c4458a98088da816c0864fd90e7d7df01f36e4ee6e1fc599a
Successfully built thriftpy2
Installing collected packages: six, Cython, ply, thriftpy2, happybase
    Running setup.py install for happybase ... done
Successfully installed Cython-3.0.11 happybase-1.2.0 ply-3.11 six-1.16.0 thriftpy2-0.5.2
bash-5.0#
```

- Open the HBase shel
  hbase shell

```
bash-5.0# hbase shell
2024-11-09 02:08:41,453 WARN  [main] util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applic
able
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
For Reference, please visit: http://hbase.apache.org/2.0/book.html#shell
Version 2.3.6, r7414579f2620fca6b75146c29ab2726fc4643ac9, Wed Jul 28 22:24:42 UTC 2021
Took 0.0016 seconds
hbase(main):001:0>
```

- Now create the table in HBase with name BFriday with column family cf.

  create 'BFriday', 'cf'
  Exit the hbase shell
  exit

```
hbase(main):001:0> create 'BFriday', 'cf'
Created table BFriday
Took 6.5193 seconds
=> Hbase::Table - BFriday
hbase(main):002:0>
```

Starting the HBase Thrift server, this allows the Python to connect via happybase. This command will be tun in the master container.

- hbase thrift start &
- Open the Pyspark session

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.0.0
      /_/

Using Python version 3.7.10 (default, Mar  2 2021 09:06:08)
SparkSession available as 'spark'.
>>> import happybase
>>> from pyspark.sql import SparkSession
>>> spark = SparkSession.builder \
...     .appName('WriteToHBaseWithHappybase') \
...     .getOrCreate()
>>>
```

Let’s import happybase and sql library from sparksession

- import happybase
  from pyspark.sql import SparkSession

```
>>> import happybase
>>> from pyspark.sql import SparkSession
>>> spark = SparkSession.builder \
...     .appName('WriteToHBaseWithHappybase') \
...     .getOrCreate()
```

Next step is to initialize sparksession

- spark = SparkSession.builder \
  .appName('WriteToHBaseWithHappybase') \
  .getOrCreate()

- data = [('row1', 'cf:UserID’, ‘1’),
  ('row2', 'cf: UserID’, '2'),
  ('row3', 'cf: UserID’, '3'),
  ('row4', 'cf: UserID’, '4')]

```
>>> data = [('row1', 'cf:UserID', '1'),
...         ('row2', 'cf:UserID', '2'),
...     ('row3', 'cf:UserID', '3'),
...     ('row4', 'cf:UserID', '4')]
```

Defining the function to insert data into HBase each partition

- def write_to_hbase_partition(partition):
  connection = happybase.Connection('master')
  connection.open()
  table = connection.table('BFriday)
  for row in partition:
  row_key, column, value = row
  table.put(row_key, {column: value})
  connection.close()

```
>>> def write_to_hbase_partition(partition):
...     connection = happybase.Connection('master')
...     connection.open()
...     table = connection.table('BFriday') # Update table name
...     for row in partition:
...         row_key, column, value = row
...         table.put(row_key, {column: value})
...     connection.close()
```

Next step is to parallelize data and apply the function with foreachPartition

- rdd = spark.sparkContext.parallelize(data)
  rdd.foreachPartition(write_to_hbase_partition)

  spark.stop()

```
>>> rdd = spark.sparkContext.parallelize(data)
>>> rdd.foreachPartition(write_to_hbase_partition)
```

b) Verify Data in HBase

- Let's verify the data, for doing that go to HBase and run the can command.
  hbase shell

  scan 'BFriday'

```
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
For Reference, please visit: http://hbase.apache.org/2.0/book.html#shell
Version 2.3.6, r7414579f2620fca6b75146c29ab2726fc4643ac9, Wed Jul 28 22:24:42 UTC 2021
Took 0.0015 seconds
hbase(main):001:0> scan 'BFriday'
ROW                                   COLUMN+CELL
 row1                                 column=cf:UserID, timestamp=2024-11-09T02:22:33.375, value=1
 row2                                 column=cf:UserID, timestamp=2024-11-09T02:22:33.562, value=2
 row3                                 column=cf:UserID, timestamp=2024-11-09T02:22:33.374, value=3
 row4                                 column=cf:UserID, timestamp=2024-11-09T02:22:33.565, value=4
4 row(s)
Took 3.0540 seconds
hbase(main):002:0>
```

As we can see the BFriday table is created, and we can see the data we loaded in earlier step.

We can also check the table created or not by using the list command.

- list

```
hbase(main):001:0> list
TABLE
BFriday
1 row(s)
Took 1.8342 seconds
=> ["BFriday"]
hbase(main):002:0>
```

As we can see the table BFriday is created.

Let's add more data to the table so that it would be easy to analyze data using different metrics.

- put 'BFriday, '5', 'cf:UserId', '5'
  put 'BFriday, '6', 'cf:UserId', '6'
  put 'BFriday, '7, 'cf:UserId', '7'

```
hbase(main):002:0> put 'BFriday', '5', 'cf:Userid', '5'
Took 0.5561 seconds
hbase(main):003:0> put 'BFriday', '6', 'cf:Userid', '6'
Took 0.0085 seconds
hbase(main):004:0>
hbase(main):005:0* put 'BFriday', '7', 'cf:Userid', '7'
Took 0.0085 seconds
hbase(main):006:0>
hbase(main):007:0* put 'BFriday', '8', 'cf:Userid', '8'
Took 0.0150 seconds
hbase(main):008:0>
hbase(main):009:0* put 'BFriday', '9', 'cf:Userid', '9'
Took 0.0091 seconds
hbase(main):010:0>
hbase(main):011:0* put 'BFriday', '10', 'cf:Userid', '10'
```

Using scan command to see if the data loaded correctly or not
Scan ‘BFriday’

```
hbase(main):014:0> scan 'BFriday'
ROW                                   COLUMN+CELL
 10                                   column=cf:Userid, timestamp=2024-11-09T02:37:54.934, value=10
 5                                    column=cf:Userid, timestamp=2024-11-09T02:37:54.621, value=5
 6                                    column=cf:Userid, timestamp=2024-11-09T02:37:54.662, value=6
 7                                    column=cf:Userid, timestamp=2024-11-09T02:37:54.735, value=7
 8                                    column=cf:Userid, timestamp=2024-11-09T02:37:54.807, value=8
 9                                    column=cf:Userid, timestamp=2024-11-09T02:37:54.876, value=9
 row1                                 column=cf:UserID, timestamp=2024-11-09T02:22:33.375, value=1
 row2                                 column=cf:UserID, timestamp=2024-11-09T02:22:33.562, value=2
 row3                                 column=cf:UserID, timestamp=2024-11-09T02:22:33.374, value=3
 row4                                 column=cf:UserID, timestamp=2024-11-09T02:22:33.565, value=4
```

Query the data from the ‘BFriday’ table to retrieve the details of the customer with UserID 1.

- get ‘BFriday, 'row1'

```
Took 0.1182 seconds
hbase(main):015:0> get 'BFriday', 'row1'
COLUMN                                CELL
 cf:UserID                            timestamp=2024-11-09T02:22:33.375, value=1
1 row(s)
Took 0.0150 seconds
hbase(main):016:0>
```

Adding purchase column data to the table BFriday.

- put ‘BFriday’, row1, ‘cf:purchase', '100'
  put ‘BFriday’, row2, ‘cf:purchase', '200'

  put ‘BFriday’, row3, ‘cf:purchase', '150'

  put ‘BFriday’, row4, ‘cf:purchase', '300'

  put ‘BFriday’, 5, ‘cf:purchase', '500'

```
Took 0.0150 seconds
hbase(main):016:0> put 'BFriday', 'row1', 'cf:purchase', '100'
Took 0.0354 seconds
hbase(main):017:0> put 'BFriday', 'row2', 'cf:purchase', '200'
Took 0.0117 seconds
hbase(main):018:0>
hbase(main):019:0* put 'BFriday', 'row3', 'cf:purchase', '150'
Took 0.0224 seconds
hbase(main):020:0>
hbase(main):021:0* put 'BFriday', 'row4', 'cf:purchase', '300'
Took 0.0146 seconds
hbase(main):022:0>
hbase(main):023:0* put 'BFriday', '5', 'cf:purchase', '500'
Took 0.0731 seconds
hbase(main):024:0> put 'BFriday', '6', 'cf:purchase', '600'
Took 0.0091 seconds
hbase(main):025:0>
hbase(main):026:0* put 'BFriday', '7', 'cf:purchase', '700'
Took 0.0147 seconds
hbase(main):027:0>
hbase(main):028:0* put 'BFriday', '8', 'cf:purchase', '180'
Took 0.0151 seconds
hbase(main):029:0>
hbase(main):030:0* put 'BFriday', '9', 'cf:purchase', '90'
Took 0.0134 seconds
hbase(main):031:0>
hbase(main):032:0* put 'BFriday', '10', 'cf:purchase', '100'
Took 0.0098 seconds
```

```
Took 0.0098 seconds
hbase(main):033:0> scan 'BFriday'
ROW                                   COLUMN+CELL
 10                                   column=cf:Userid, timestamp=2024-11-09T02:37:54.934, value=10
 10                                   column=cf:purchase, timestamp=2024-11-09T02:46:29.643, value=100
 5                                    column=cf:Userid, timestamp=2024-11-09T02:37:54.621, value=5
 5                                    column=cf:purchase, timestamp=2024-11-09T02:46:29.201, value=500
 6                                    column=cf:Userid, timestamp=2024-11-09T02:37:54.662, value=6
 6                                    column=cf:purchase, timestamp=2024-11-09T02:46:29.354, value=600
 7                                    column=cf:Userid, timestamp=2024-11-09T02:37:54.735, value=7
 7                                    column=cf:purchase, timestamp=2024-11-09T02:46:29.457, value=700
 8                                    column=cf:Userid, timestamp=2024-11-09T02:37:54.807, value=8
 8                                    column=cf:purchase, timestamp=2024-11-09T02:46:29.521, value=180
 9                                    column=cf:Userid, timestamp=2024-11-09T02:37:54.876, value=9
 9                                    column=cf:purchase, timestamp=2024-11-09T02:46:29.572, value=90
 row1                                 column=cf:UserID, timestamp=2024-11-09T02:22:33.375, value=1
 row1                                 column=cf:purchase, timestamp=2024-11-09T02:46:28.610, value=100
 row2                                 column=cf:UserID, timestamp=2024-11-09T02:22:33.562, value=2
 row2                                 column=cf:purchase, timestamp=2024-11-09T02:46:28.709, value=200
 row3                                 column=cf:UserID, timestamp=2024-11-09T02:22:33.374, value=3
 row3                                 column=cf:purchase, timestamp=2024-11-09T02:46:28.834, value=150
 row4                                 column=cf:UserID, timestamp=2024-11-09T02:22:33.565, value=4
 row4                                 column=cf:purchase, timestamp=2024-11-09T02:46:28.994, value=300
10 row(s)
Took 0.0479 seconds
```

Adding name column data to the table BFriday.

- put ‘BFriday’, row1, ‘cf:name’, ‘John’
  put ‘BFriday’, row2, ‘cf: name’, 'Doe'

  put ‘BFriday’, row3, ‘cf: name’, 'Joe'

  put ‘BFriday’, row4, ‘cf: name’, 'Ryan'

  put ‘BFriday’, 5, ‘cf: name’, ‘Mitch'

```
hbase(main):034:0> put 'BFriday', 'row1', 'cf:name', 'John'
Took 0.0138 seconds
hbase(main):035:0> put 'BFriday', 'row2', 'cf:name', 'Doe'
Took 0.0068 seconds
hbase(main):036:0>
hbase(main):037:0* put 'BFriday', 'row3', 'cf:name', 'Joe'
Took 0.0097 seconds
hbase(main):038:0>
hbase(main):039:0* put 'BFriday', 'row4', 'cf:name', 'Ryan'
Took 0.0072 seconds
hbase(main):040:0>
hbase(main):041:0* put 'BFriday', '5', 'cf:name', 'Mitch'
Took 0.0069 seconds
hbase(main):042:0> put 'BFriday', '6', 'cf:name', 'Miller'
Took 0.0114 seconds
hbase(main):043:0>
hbase(main):044:0* put 'BFriday', '7', 'cf:name', 'Taylor'
Took 0.0080 seconds
hbase(main):045:0>
hbase(main):046:0* put 'BFriday', '8', 'cf:name', 'Chris'
Took 0.0068 seconds
hbase(main):047:0>
hbase(main):048:0* put 'BFriday', '9', 'cf:name', 'Tom'
Took 0.0072 seconds
hbase(main):049:0>
hbase(main):050:0* put 'BFriday', '10', 'cf:name', 'Ron'
Took 0.0063 seconds
```

```
ROW                                  COLUMN+CELL
 10                                  column=cf:Userid, timestamp=2024-11-09T02:37:54.934, value=10
 10                                  column=cf:name, timestamp=2024-11-09T02:48:49.686, value=Ron
 10                                  column=cf:purchase, timestamp=2024-11-09T02:46:29.643, value=100
 5                                   column=cf:Userid, timestamp=2024-11-09T02:37:54.621, value=5
 5                                   column=cf:name, timestamp=2024-11-09T02:48:49.539, value=Mitch
 5                                   column=cf:purchase, timestamp=2024-11-09T02:46:29.201, value=500
 6                                   column=cf:Userid, timestamp=2024-11-09T02:37:54.662, value=6
 6                                   column=cf:name, timestamp=2024-11-09T02:48:49.560, value=Miller
 6                                   column=cf:purchase, timestamp=2024-11-09T02:46:29.354, value=600
 7                                   column=cf:Userid, timestamp=2024-11-09T02:37:54.735, value=7
 7                                   column=cf:name, timestamp=2024-11-09T02:48:49.596, value=Taylor
 7                                   column=cf:purchase, timestamp=2024-11-09T02:46:29.457, value=700
 8                                   column=cf:Userid, timestamp=2024-11-09T02:37:54.807, value=8
 8                                   column=cf:name, timestamp=2024-11-09T02:48:49.628, value=Chris
 8                                   column=cf:purchase, timestamp=2024-11-09T02:46:29.521, value=180
 9                                   column=cf:Userid, timestamp=2024-11-09T02:37:54.876, value=9
 9                                   column=cf:name, timestamp=2024-11-09T02:48:49.656, value=Tom
 9                                   column=cf:purchase, timestamp=2024-11-09T02:46:29.572, value=90
 row1                                column=cf:UserID, timestamp=2024-11-09T02:22:33.375, value=1
 row1                                column=cf:name, timestamp=2024-11-09T02:48:49.417, value=John
 row1                                column=cf:purchase, timestamp=2024-11-09T02:46:28.610, value=100
 row2                                column=cf:UserID, timestamp=2024-11-09T02:22:33.562, value=2
 row2                                column=cf:name, timestamp=2024-11-09T02:48:49.443, value=Doe
 row2                                column=cf:purchase, timestamp=2024-11-09T02:46:28.709, value=200
 row3                                column=cf:UserID, timestamp=2024-11-09T02:22:33.374, value=3
 row3                                column=cf:name, timestamp=2024-11-09T02:48:49.477, value=Joe
 row3                                column=cf:purchase, timestamp=2024-11-09T02:46:28.834, value=150
 row4                                column=cf:UserID, timestamp=2024-11-09T02:22:33.565, value=4
 row4                                column=cf:name, timestamp=2024-11-09T02:48:49.510, value=Ryan
 row4                                column=cf:purchase, timestamp=2024-11-09T02:46:28.994, value=300
10 row(s)
```

Let's perform some data manipulation to get more information about the customer.
I am adding the middle name column to the customer whose row id from 5 to 10.

- (5..10).each do |i|
  middle_name = "MidName#{i}"
  put 'BFriday', "#{i}, 'cf:middlename', middle_name
  end

```
hbase(main):007:0> (5..10).each do |i|
hbase(main):008:1* Display all 606 possibilities? (y or n)
hbase(main):008:1*     middle_name = "MidName#{i}"
hbase(main):009:1>     put 'BFriday', "#{i}", 'cf:middlename', middle_name
hbase(main):010:1> end
Took 0.5233 seconds
Took 0.0133 seconds
Took 0.0118 seconds
Took 0.0079 seconds
Took 0.0198 seconds
Took 0.0120 seconds
=> 5..10
hbase(main):011:0>
```

Let's see the data to understand how the change work.

```
hbase(main):011:0> scan 'BFriday'
ROW                                   COLUMN+CELL
 10                                   column=cf:Userid, timestamp=2024-11-09T02:37:54.934, value=10
 10                                   column=cf:middlename, timestamp=2024-11-09T17:40:02.333, value=MidName10
 10                                   column=cf:name, timestamp=2024-11-09T02:48:49.686, value=Ron
 10                                   column=cf:purchase, timestamp=2024-11-09T02:46:29.643, value=100
 5                                    column=cf:Userid, timestamp=2024-11-09T02:37:54.621, value=5
 5                                    column=cf:middlename, timestamp=2024-11-09T17:40:02.193, value=MidName5
 5                                    column=cf:name, timestamp=2024-11-09T02:48:49.539, value=Mitch
 5                                    column=cf:purchase, timestamp=2024-11-09T02:46:29.201, value=500
 6                                    column=cf:Userid, timestamp=2024-11-09T02:37:54.662, value=6
 6                                    column=cf:middlename, timestamp=2024-11-09T17:40:02.269, value=MidName6
 6                                    column=cf:name, timestamp=2024-11-09T02:48:49.560, value=Miller
 6                                    column=cf:purchase, timestamp=2024-11-09T02:46:29.354, value=600
 7                                    column=cf:Userid, timestamp=2024-11-09T02:37:54.735, value=7
 7                                    column=cf:middlename, timestamp=2024-11-09T17:40:02.283, value=MidName7
 7                                    column=cf:name, timestamp=2024-11-09T02:48:49.596, value=Taylor
 7                                    column=cf:purchase, timestamp=2024-11-09T02:46:29.457, value=700
 8                                    column=cf:Userid, timestamp=2024-11-09T02:37:54.807, value=8
 8                                    column=cf:middlename, timestamp=2024-11-09T17:40:02.296, value=MidName8
 8                                    column=cf:name, timestamp=2024-11-09T02:48:49.628, value=Chris
 8                                    column=cf:purchase, timestamp=2024-11-09T02:46:29.521, value=180
 9                                    column=cf:Userid, timestamp=2024-11-09T02:37:54.876, value=9
 9                                    column=cf:middlename, timestamp=2024-11-09T17:40:02.316, value=MidName9
 9                                    column=cf:name, timestamp=2024-11-09T02:48:49.656, value=Tom
 9                                    column=cf:purchase, timestamp=2024-11-09T02:46:29.572, value=90
```

As we can see for row id 5 to 10 the middle name got added. This is the sample data, but I think we can use this manipulation to add some relevant information which can add value to the data, and we can get some valuable insight by analyzing the dataset.

The next manipulation is modifying the middle name for customer with row id 5 to 10

- (5..10).each do |i|
  Put ‘BFriday’, “#{i}”, ‘cf:middlename’, “MidName#{i}_Modified”
  end

```
Took 0.1637 seconds
hbase(main):012:0> (5..10).each do |i|
hbase(main):013:1*     put 'BFriday', "#{i}", 'cf:middlename', "MidName#{i}_Modified"
hbase(main):014:1> end
Took 0.0144 seconds
Took 0.0073 seconds
Took 0.0115 seconds
Took 0.0066 seconds
Took 0.0063 seconds
Took 0.0063 seconds
=> 5..10
hbase(main):015:0>
```

Looking the data if we can see changes to our data.

```
hbase(main):015:0> scan 'BFriday'
ROW                                   COLUMN+CELL
 10                                   column=cf:Userid, timestamp=2024-11-09T02:37:54.934, value=10
 10                                   column=cf:middlename, timestamp=2024-11-09T17:46:34.713, value=MidName10_Modified
 10                                   column=cf:name, timestamp=2024-11-09T02:48:49.686, value=Ron
 10                                   column=cf:purchase, timestamp=2024-11-09T02:46:29.643, value=100
 5                                    column=cf:Userid, timestamp=2024-11-09T02:37:54.621, value=5
 5                                    column=cf:middlename, timestamp=2024-11-09T17:46:34.664, value=MidName5_Modified
 5                                    column=cf:name, timestamp=2024-11-09T02:48:49.539, value=Mitch
 5                                    column=cf:purchase, timestamp=2024-11-09T02:46:29.201, value=500
 6                                    column=cf:Userid, timestamp=2024-11-09T02:37:54.662, value=6
 6                                    column=cf:middlename, timestamp=2024-11-09T17:46:34.676, value=MidName6_Modified
 6                                    column=cf:name, timestamp=2024-11-09T02:48:49.560, value=Miller
 6                                    column=cf:purchase, timestamp=2024-11-09T02:46:29.354, value=600
 7                                    column=cf:Userid, timestamp=2024-11-09T02:37:54.735, value=7
 7                                    column=cf:middlename, timestamp=2024-11-09T17:46:34.686, value=MidName7_Modified
 7                                    column=cf:name, timestamp=2024-11-09T02:48:49.596, value=Taylor
 7                                    column=cf:purchase, timestamp=2024-11-09T02:46:29.457, value=700
 8                                    column=cf:Userid, timestamp=2024-11-09T02:37:54.807, value=8
 8                                    column=cf:middlename, timestamp=2024-11-09T17:46:34.697, value=MidName8_Modified
 8                                    column=cf:name, timestamp=2024-11-09T02:48:49.628, value=Chris
 8                                    column=cf:purchase, timestamp=2024-11-09T02:46:29.521, value=180
 9                                    column=cf:Userid, timestamp=2024-11-09T02:37:54.876, value=9
 9                                    column=cf:middlename, timestamp=2024-11-09T17:46:34.705, value=MidName9_Modified
 9                                    column=cf:name, timestamp=2024-11-09T02:48:49.656, value=Tom
 9                                    column=cf:purchase, timestamp=2024-11-09T02:46:29.572, value=90
```

As we can see from above the middlename of the customer with row is 5 to 10 have been modified.

As part of next manipulation lets perform delete function. Using delete I am deleting customer with row id 1 to 4.

- (1..4).each do |i|
  deleteall ‘BFriday’, “row#{i}”
  end

```
Took 0.0522 seconds
hbase(main):016:0> (1..4).each do |i|
hbase(main):017:1*   deleteall 'BFriday', "row#{i}"
hbase(main):018:1> end
Took 0.0281 seconds
Took 0.0152 seconds
Took 0.0091 seconds
Took 0.0080 seconds
=> 1..4
hbase(main):019:0>
```

Let’s look at the data result after performing the deleteall

```
hbase(main):019:0> scan 'BFriday'
ROW                                             COLUMN+CELL
 10                                             column=cf:Userid, timestamp=2024-11-09T02:37:54.934, value=10
 10                                             column=cf:middlename, timestamp=2024-11-09T17:46:34.713, value=MidName10
 10                                             column=cf:name, timestamp=2024-11-09T02:48:49.686, value=Ron
 10                                             column=cf:purchase, timestamp=2024-11-09T02:46:29.643, value=100
 5                                              column=cf:Userid, timestamp=2024-11-09T02:37:54.621, value=5
 5                                              column=cf:middlename, timestamp=2024-11-09T17:46:34.664, value=MidName5_
 5                                              column=cf:name, timestamp=2024-11-09T02:48:49.539, value=Mitch
 5                                              column=cf:purchase, timestamp=2024-11-09T02:46:29.201, value=500
 6                                              column=cf:Userid, timestamp=2024-11-09T02:37:54.662, value=6
 6                                              column=cf:middlename, timestamp=2024-11-09T17:46:34.676, value=MidName6_
 6                                              column=cf:name, timestamp=2024-11-09T02:48:49.560, value=Miller
 6                                              column=cf:purchase, timestamp=2024-11-09T02:46:29.354, value=600
 7                                              column=cf:Userid, timestamp=2024-11-09T02:37:54.735, value=7
 7                                              column=cf:middlename, timestamp=2024-11-09T17:46:34.686, value=MidName7_
 7                                              column=cf:name, timestamp=2024-11-09T02:48:49.596, value=Taylor
 7                                              column=cf:purchase, timestamp=2024-11-09T02:46:29.457, value=700
 8                                              column=cf:Userid, timestamp=2024-11-09T02:37:54.807, value=8
 8                                              column=cf:middlename, timestamp=2024-11-09T17:46:34.697, value=MidName8_
 8                                              column=cf:name, timestamp=2024-11-09T02:48:49.628, value=Chris
 8                                              column=cf:purchase, timestamp=2024-11-09T02:46:29.521, value=180
 9                                              column=cf:Userid, timestamp=2024-11-09T02:37:54.876, value=9
 9                                              column=cf:middlename, timestamp=2024-11-09T17:46:34.705, value=MidName9_
 9                                              column=cf:name, timestamp=2024-11-09T02:48:49.656, value=Tom
 9                                              column=cf:purchase, timestamp=2024-11-09T02:46:29.572, value=90
6 row(s)
```

As we can see the customer with row id 1 to 4 have been deleted.
Using the delete function we can delete any irrelevant information which adding no value but noise to the dataset.

```
hbase(main):015:0> scan 'BFriday'
ROW                                 COLUMN+CELL
 10                                 column=cf:Userid, timestamp=2024-11-09T02:37:54.934, value=10
 10                                 column=cf:middlename, timestamp=2024-11-09T17:46:34.713, value=MidName10_Modified
 10                                 column=cf:name, timestamp=2024-11-09T02:48:49.686, value=Ron
 10                                 column=cf:purchase, timestamp=2024-11-09T02:46:29.643, value=100
 5                                  column=cf:Userid, timestamp=2024-11-09T02:37:54.621, value=5
 5                                  column=cf:middlename, timestamp=2024-11-09T17:46:34.664, value=MidName5_Modified
 5                                  column=cf:name, timestamp=2024-11-09T02:48:49.539, value=Mitch
 5                                  column=cf:purchase, timestamp=2024-11-09T02:46:29.201, value=500
 6                                  column=cf:Userid, timestamp=2024-11-09T02:37:54.662, value=6
 6                                  column=cf:middlename, timestamp=2024-11-09T17:46:34.676, value=MidName6_Modified
 6                                  column=cf:name, timestamp=2024-11-09T02:48:49.560, value=Miller
 6                                  column=cf:purchase, timestamp=2024-11-09T02:46:29.354, value=600
 7                                  column=cf:Userid, timestamp=2024-11-09T02:37:54.735, value=7
 7                                  column=cf:middlename, timestamp=2024-11-09T17:46:34.686, value=MidName7_Modified
 7                                  column=cf:name, timestamp=2024-11-09T02:48:49.596, value=Taylor
 7                                  column=cf:purchase, timestamp=2024-11-09T02:46:29.457, value=700
 8                                  column=cf:Userid, timestamp=2024-11-09T02:37:54.807, value=8
 8                                  column=cf:middlename, timestamp=2024-11-09T17:46:34.697, value=MidName8_Modified
 8                                  column=cf:name, timestamp=2024-11-09T02:48:49.628, value=Chris
 8                                  column=cf:purchase, timestamp=2024-11-09T02:46:29.521, value=180
 9                                  column=cf:Userid, timestamp=2024-11-09T02:37:54.876, value=9
 9                                  column=cf:middlename, timestamp=2024-11-09T17:46:34.705, value=MidName9_Modified
 9                                  column=cf:name, timestamp=2024-11-09T02:48:49.656, value=Tom
 9                                  column=cf:purchase, timestamp=2024-11-09T02:46:29.572, value=90
 row1                               column=cf:UserID, timestamp=2024-11-09T02:22:33.375, value=1
 row1                               column=cf:name, timestamp=2024-11-09T02:48:49.417, value=John
 row1                               column=cf:purchase, timestamp=2024-11-09T02:46:28.610, value=100
 row2                               column=cf:UserID, timestamp=2024-11-09T02:22:33.562, value=2
 row2                               column=cf:name, timestamp=2024-11-09T02:48:49.443, value=Doe
 row2                               column=cf:purchase, timestamp=2024-11-09T02:46:28.709, value=200
 row3                               column=cf:UserID, timestamp=2024-11-09T02:22:33.374, value=3
 row3                               column=cf:name, timestamp=2024-11-09T02:48:49.477, value=Joe
 row3                               column=cf:purchase, timestamp=2024-11-09T02:46:28.834, value=150
 row4                               column=cf:UserID, timestamp=2024-11-09T02:22:33.565, value=4
 row4                               column=cf:name, timestamp=2024-11-09T02:48:49.510, value=Ryan
 row4                               column=cf:purchase, timestamp=2024-11-09T02:46:28.994, value=300
```

Let’s try to get data using the random row id.

I am getting user with row id 5, 8, and 10 to see what information we have for those customers.

- [5, 8, 10].each do |userid|
  Get 'BFriday', "#{userid}"
  end

```
Took 0.0426 seconds
hbase(main):020:0> [5, 8, 10].each do |userid|
hbase(main):021:1*  get 'BFriday', "#{userid}"
hbase(main):022:1> end
COLUMN                                          CELL
 cf:Userid                                      timestamp=2024-11-09T02:37:54.621, value=5
 cf:middlename                                  timestamp=2024-11-09T17:46:34.664, value=MidName5_Modified
 cf:name                                        timestamp=2024-11-09T02:48:49.539, value=Mitch
 cf:purchase                                    timestamp=2024-11-09T02:46:29.201, value=500
1 row(s)
Took 0.0965 seconds
COLUMN                                          CELL
 cf:Userid                                      timestamp=2024-11-09T02:37:54.807, value=8
 cf:middlename                                  timestamp=2024-11-09T17:46:34.697, value=MidName8_Modified
 cf:name                                        timestamp=2024-11-09T02:48:49.628, value=Chris
 cf:purchase                                    timestamp=2024-11-09T02:46:29.521, value=180
1 row(s)
Took 0.0118 seconds
COLUMN                                          CELL
 cf:Userid                                      timestamp=2024-11-09T02:37:54.934, value=10
 cf:middlename                                  timestamp=2024-11-09T17:46:34.713, value=MidName10_Modified
 cf:name                                        timestamp=2024-11-09T02:48:49.686, value=Ron
 cf:purchase                                    timestamp=2024-11-09T02:46:29.643, value=100
1 row(s)
Took 0.0109 seconds
=> [5, 8, 10]
hbase(main):023:0>
```

Conclusion:

The overall documentation shows that each component plays an important role in providing highly scalable and flexible system which can easily handles the growing need of retailers in handling the increase volume of data or managing the dataset efficiently.

We have successfully ingested data into HDFS from GitHub, as next step we able to define table schema in Hive and were able to load data into the table. Once the data loaded into the hive table, we were able to analyze the data using HiveQL.

In order to perform the advanced data manipulation and transformation we used PySpark. Spark runs multiple algorithm or rule-based checks in processing layers to see if data form any pattern and send it downstream for further analysis. The recognizing of the patterns will help us to understand customer behavior to know if there is any product which are in demand, or we can learn about customer interest and make changes to the inventory or plan strategies to those customers.

Using the components, we were able to get lots of useful information about the customer. We analyze different set of queries result to get the valuable insight of the black Friday dataset.
We get to know that the total purchase of $1,45,3324 was done by customer on Black Friday. Out of which $947,283 was spent by males and $506,041 was spent by females. It is interesting to know that male customer spent more compared to females during black Friday. We also understand that city B spent maximum of amount $638,735 compared to another city A and C. we also get to know that that customer with user id 100010 purchased most of the product that is 18 and the customer is in age group of 36-45. On further investigation we found that we have 98 male users and 51 female users. We also did some analysis on customer occupation and found that Occupation 1 has most of 30 users. The above analysis also indicates that the customer with user id 100010 has spent most of $201,995 and the age group 26-35 have made purchase of $451,836 on Black Friday.

All the analysis performed gives us very important information about customer and using which retailer will be able to understand the behavior or need of the customer. Using which retailers can improve marketing strategies, offer scheme to loyal customer, also can make changes to inventory. All the changes will help retailers to improve sales and build trust with customer and will attract more customer.

For this project I used dataset with masked column data also the dataset is very small. But for future analysis we can include complete dataset with real time data processing to make sure we have all the pieces to build machine learning system which is up to date and predict trends or demand changes.