Assignment2

March 5, 2023

1 Overview

Group 136

Name	BITS ID	Contribution
Vinayak Nayak	2021fc04135	100%
Shreysi Kalra	2021fc04586	100%

Setup Details

- Operating System: Ubuntu 20.04 LTS
- Hadoop: Hadoop 3.2.4
- Python: Python 3.8
- Hive: Hive 3.1.3

Submission Details Summary

- Hadoop is installed and configured in `pseudo-distributed` setup since I have only one node/machine. Hence the default replication factor in the `hdfs-site.xml` is set to `1`. In a real-world distributed scenario we would configure it to be 3/5 i.e. odd number > 1 so that we can achieve fault-tolerance and have a majority quorum in case of network-partition or temporary node-failure etc.
- Query 1 is performed using Apache Hive (Which uses Hadoop Map-Reduce as the execution engine beneath it; we haven't configured `tez` or `spark` but use the default `map-reduce` engine for hive)
- Queries 2-7 are performed using Hadoop Map-Reduce with the Hadoop Streaming API for Python.
- All the outputs are stored in the format `/home/query<#>_output/` which are displayed below in chronological order.
- The following videos shows the execution of all our queries with our environment setup and execution engine.
  - Part 1
  - Part 2
- The map-reduce code for each of the queries 2-7 is organized in folders with names `query_{#}` respectively.
- The commands for creation of table, loading of data from csv and querying for unique customers from `Germany` as the country of our choice is organized in the folder `query_1`. > Each query is executed using python data analysis libraries like numpy and also subsequently using

Hadoop map-reduce/ Apache hive. The code for both these methods and the corresponding results are also attached in the document below. Both of these results map which ascertain the veracity of the executed queries.

- Configuration files which were modified for utilization of `hadoop` and `hive` are stored in `configuration_files` respectively
- Changes made to `.bashrc` file on my setup are reflected in `bashrc` file attached in `configuration_files`. Please note I had to remove other lines of code from the `.bashrc` file to preserve my privacy.
- There are some considerations which I had to make made in order to account for some anomalous observations in the dataset. I have mentioned them in the `Data Understanding` module below by doing a quick EDA on the same.

[ ]:
```
# Have a look at the different output files created as a result of map reduce␣
↪output
!hadoop fs -ls /home/
```

```
Found 7 items
drwxr-xr-x   - vinayak supergroup          0 2023-03-04 19:57
/home/query2_output
drwxr-xr-x   - vinayak supergroup          0 2023-03-04 19:58
/home/query3_output
drwxr-xr-x   - vinayak supergroup          0 2023-03-04 19:59
/home/query4_output
drwxr-xr-x   - vinayak supergroup          0 2023-03-04 20:27
/home/query5_output
drwxr-xr-x   - vinayak supergroup          0 2023-03-04 20:00
/home/query6_output
drwxr-xr-x   - vinayak supergroup          0 2023-03-04 20:22
/home/query7_output
drwxr-xr-x   - vinayak supergroup          0 2023-03-04 18:33 /home/vinayak
```

[ ]:
```
# Have a look at the input csv placed in the hadoop file system
!hadoop fs -ls /hdfs_input
```

```
Found 1 items
-rw-r--r--   1 vinayak supergroup   49722977 2023-03-04 17:46
/hdfs_input/assignment2_retail_data_utf8.csv
```

[ ]:
```
# Have a look at the hive data which has gotten created due to
# import of the csv file into hive
!hadoop fs -ls /user/hive/warehouse/test.db
```

```
Found 1 items
drwxr-xr-x   - vinayak supergroup          0 2023-03-05 07:41
/user/hive/warehouse/test.db/retail
```

2 Data Understanding

```
[ ]: import pandas as pd
     df = pd.read_csv(
         "data/Assignment -2  2023 BDS DATA SET online_retail_data.csv",
         encoding="unicode_escape",
     )
     df.head(2)
```

```
[ ]:    Record No. Invoice StockCode                          Description  \
     0           1  489434     85048  15CM CHRISTMAS GLASS BALL 20 LIGHTS
     1           2  489434    79323P                   PINK CHERRY LIGHTS

        Quantity       InvoiceDate  Price  Customer ID         Country
     0        12  12-01-2009 07:45   6.95      13085.0  United Kingdom
     1        12  12-01-2009 07:45   6.75      13085.0  United Kingdom
```

There are non standard i.e. (not `utf-8`) characters also in our dataframe. This will cause our map reduce jobs to fail as they wouldn't be rendered in our python map-reduce characters. We will have to convert these into utf-8 values. For eg, consider the following line.

```
[ ]: !sed -ne 31081p data/"Assignment -2  2023 BDS DATA SET online_retail_data.csv"
```

```
31080,491969,gift_0001_80,Dotcomgiftshop Gift Voucher  80.00,1,12/14/2009
17:57:00,69.56,,United Kingdom
```

There is a pound sign in the description which is not getting rendered properly, hence we will have to first convert them into `utf-8` characters before passing them to **map-reduce** tasks.

```
[ ]: df.dtypes
```

```
[ ]: Record No.       int64
     Invoice         object
     StockCode       object
     Description     object
     Quantity         int64
     InvoiceDate     object
     Price          float64
     Customer ID    float64
     Country         object
     dtype: object
```

We can see that customer ID has been read as a float. This means there is some issue, let's try to see how many nas are present in our data.

```
[ ]: df.isnull().sum()
[ ]: Record No.         0
     Invoice            0
     StockCode          0
     Description     2928
     Quantity           0
     InvoiceDate        0
     Price              0
     Customer ID   107927
     Country            0
     dtype: int64
```

Since there are null values in `Customer ID` column, they are all read as nans and the column has been made into a float dtype.

```
[ ]: df[df.Price < 0].head(3)
```

```
[ ]:         Record No.  Invoice StockCode      Description  Quantity  \
     179403      179404  A506401         B  Adjust bad debt         1
     276274      276275  A516228         B  Adjust bad debt         1
     403472      403473  A528059         B  Adjust bad debt         1

                     InvoiceDate     Price  Customer ID         Country
     179403  04/29/2010 13:36:00 -53594.36          NaN  United Kingdom
     276274  07/19/2010 11:24:00 -44031.79          NaN  United Kingdom
     403472  10/20/2010 12:04:00 -38925.87          NaN  United Kingdom
```

```
[ ]: df[df.Quantity < 0].sample(3)
```

```
[ ]:         Record No.  Invoice StockCode                          Description  \
     17551        17552  C490798    84997D  PINK 3 PIECE MINI DOTS CUTLERY SET
     381112      381113  C526110     22470                HEART OF WICKER LARGE
     442961      442962  C531557     22271                 FELTCRAFT DOLL ROSIE

             Quantity       InvoiceDate  Price  Customer ID         Country
     17551         -6  12-08-2009 11:51   3.75      14277.0          France
     381112        -1  10-08-2010 13:03   2.55      13113.0  United Kingdom
     442961      -144  11-09-2010 10:06   2.55      12454.0           Spain
```

There are negative values in `price` column and `quantity` column which are not actually transactions but some kind of ledger statements from the account or something; we must take this into consideration when using map-reduce on these queries.

Our way of handling

If there is a negative value for quantity or for price, ignore that line item; do not consider it for our analysis.

One more assumption in our analysis is that we're considering the price column as price per unit of the line item mentioned. So our revenue calculations are done after multiplication of quantity *with price and then summing across the line items.*

```
[ ]: stock_codes = [
         "ADJUST", "ADJUST2", "AMAZONFEE", "B", "BANK CHARGES", "C2", "C3", "D",
         "DOT", "GIFT", "M", "POST", "S", "TEST001", "TEST002", "m",
     ]

     df[df.StockCode.isin(stock_codes)].drop_duplicates(subset=["StockCode"])[
         ["StockCode", "Description"]].reset_index(drop=True)
```

```
[ ]:       StockCode                          Description
     0          POST                              POSTAGE
     1             D                             Discount
     2           DOT                       DOTCOM POSTAGE
     3             M                               Manual
     4            C2                             CARRIAGE
     5  BANK CHARGES                         Bank Charges
     6       TEST001                This is a test product.
     7       TEST002                This is a test product.
     8        ADJUST  Adjustment by john on 26/01/2010 16
     9          GIFT                                  NaN
     10            m                               Manual
     11            S                              SAMPLES
     12            B                        Adjust bad debt
     13      ADJUST2  Adjustment by Peter on Jun 25 2010
     14           C3                                  NaN
     15    AMAZONFEE                           AMAZON FEE
```

There are some stock codes which are not actually pertinent to any product but it's related to some accounting/finance details. When writing our queries we will need to skip these test products respectively. As seen above these are clearly not items which are sold but services/miscellaneous things which are dumped in the data.

```
[ ]: # Store the dataframe using `utf-8` encoding for our usecase.
     df.to_csv("data/assignment2_retail_data_utf8.csv", index=False,␣
      ↪encoding="utf-8")
```

Note

Clarification was sought from Professor Sunil Bhutada regarding the file format conversion and permission was granted to consider `utf-8` conversion as mentioned in the `instructor_clarifications` folder of the conversation with the Professor.

```
[ ]: # For local comparisons, make sure to drop these stock code items
     df = df[~df.StockCode.isin(stock_codes)]
```

In the subsequent sections of this document, I have shown the queries and their respective outputs both in Apache ecosystem and in the local ecosystem. The Map-Reduce Job Summary for each of the jobs is attached at the end of the query as requested in the submission requirements respectively.

3 Queries

Query 1 Computation

Total number of unique customers in the "given country".

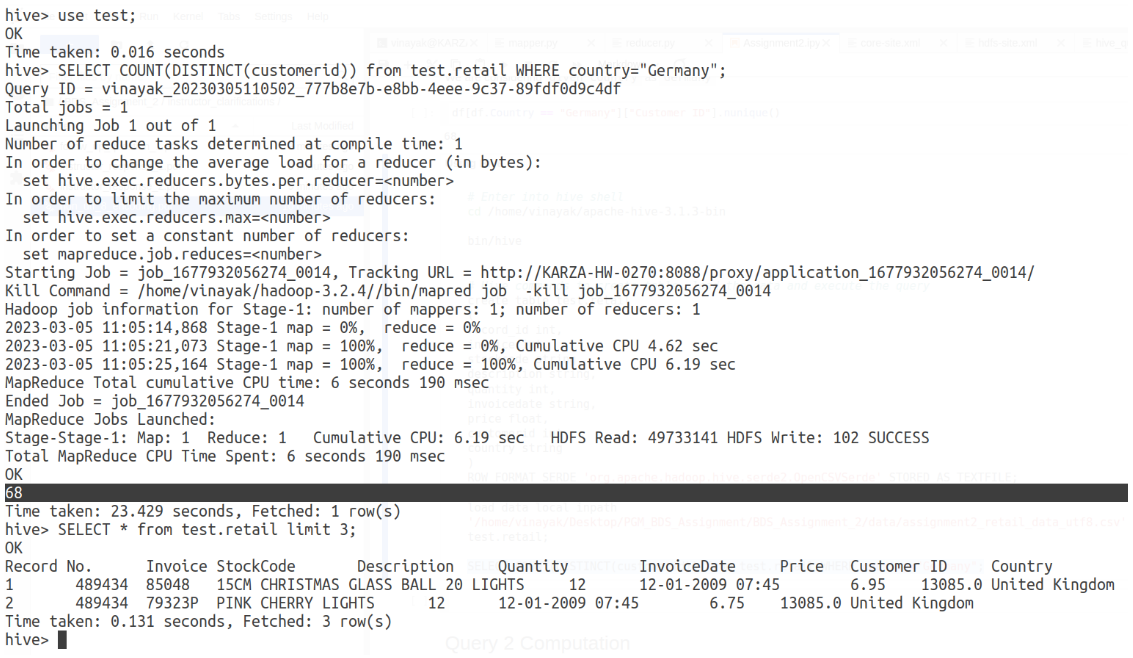
We are considering `given country` as `Germany`

```
[ ]: df[df.Country == "Germany"]["Customer ID"].nunique()
```

```
[ ]: 68
```

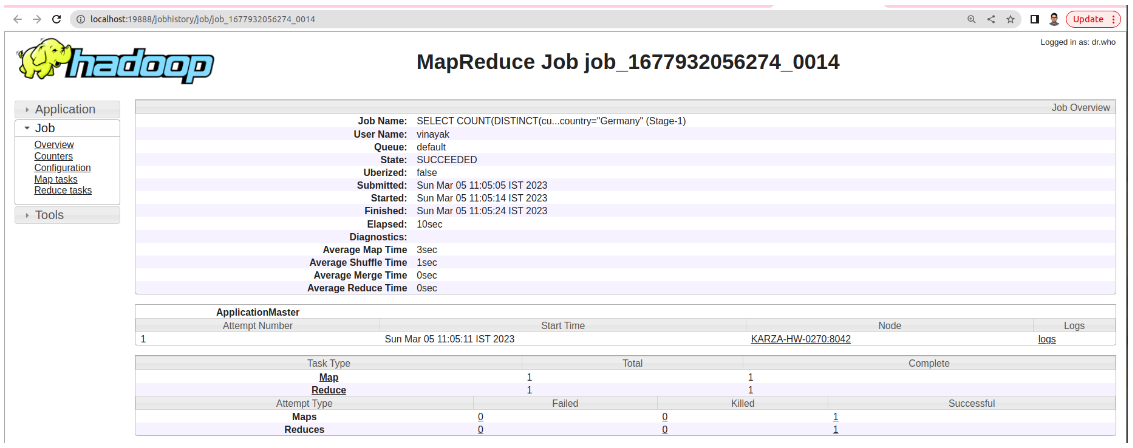
Using Hive

```
# Enter into hive shell
cd /home/vinayak/apache-hive-3.1.3-bin

bin/hive



# Hive commands to create table, load the data and execute the query
create table test.retail
(
record_id int,
invoice int,
stockcode string,
description string,
quantity int,
invoicedate string,
price float,
customerid int,
country string
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde' STORED AS TEXTFILE;

load data local
inpath '/home/vinayak/Desktop/PGM_BDS_Assignment/BDS_Assignment_2/data/assignment2_retail_data
into table test.retail;

SELECT COUNT(DISTINCT(customerid)) from test.retail WHERE country="Germany";
```



```
hive> use test;
OK
Time taken: 0.016 seconds
hive> SELECT COUNT(DISTINCT(customerid)) from test.retail WHERE country="Germany";
Query ID = vinayak_20230305110502_777b8e7b-e8bb-4eee-9c37-89fdf0d9c4df
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1677932056274_0014, Tracking URL = http://KARZA-HW-0270:8088/proxy/application_1677932056274_0014/
Kill Command = /home/vinayak/hadoop-3.2.4//bin/mapred job  -kill job_1677932056274_0014
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2023-03-05 11:05:14,868 Stage-1 map = 0%,  reduce = 0%
2023-03-05 11:05:21,073 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 4.62 sec
2023-03-05 11:05:25,164 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.19 sec
MapReduce Total cumulative CPU time: 6 seconds 190 msec
Ended Job = job_1677932056274_0014
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 6.19 sec   HDFS Read: 49733141 HDFS Write: 102 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 190 msec
OK
68
Time taken: 23.429 seconds, Fetched: 1 row(s)
hive> SELECT * from test.retail limit 3;
OK
Record No.      Invoice StockCode       Description     Quantity        InvoiceDate     Price   Customer ID     Country
1       489434  85048   15CM CHRISTMAS GLASS BALL 20 LIGHTS     12      12-01-2009 07:45        6.95    13085.0 United Kingdom
2       489434  79323P  PINK CHERRY LIGHTS      12      12-01-2009 07:45        6.75    13085.0 United Kingdom
Time taken: 0.131 seconds, Fetched: 3 row(s)
hive> 
```

The summary of the corresponding hive query in map-reduce jobs is as follows.



MapReduce Job job_1677932056274_0014

Job Overview

Job Name:	SELECT COUNT(DISTINCT(cu...country="Germany" (Stage-1)
User Name:	vinayak
Queue:	default
State:	SUCCEEDED
Uberized:	false
Submitted:	Sun Mar 05 11:05:05 IST 2023
Started:	Sun Mar 05 11:05:14 IST 2023
Finished:	Sun Mar 05 11:05:24 IST 2023
Elapsed:	10sec
Diagnostics:	
Average Map Time	3sec
Average Shuffle Time	1sec
Average Merge Time	0sec
Average Reduce Time	0sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Sun Mar 05 11:05:11 IST 2023	KARZA-HW-0270:8042	logs

Task Type	Total	Complete
Map	1	1
Reduce	1	1

Attempt Type	Failed	Killed	Successful
Maps	0	0	1
Reduces	0	0	1

Query 2 Computation

Country from which the maximum revenue was collected from sales in the month of March 2010.

```
[ ]: df["InvoiceDate"] = pd.to_datetime(df["InvoiceDate"])
```

```
[ ]: df["Month_Year"] = (
         df["InvoiceDate"].dt.month.apply(lambda x: str(x))
         + "/"
         + df["InvoiceDate"].dt.year.apply(lambda x: str(x))
     )
```

```
[ ]: revenue_map = {}
     for country, sub_df in df[df["Month_Year"] == "3/2010"].groupby(by="Country"):
         sub_df = sub_df[sub_df.Quantity > 0]
         sub_df = sub_df[sub_df.Price > 0]
         revenue = (sub_df.Quantity * sub_df.Price).sum()
         revenue_map[country] = round(revenue, 2)
```

```
[ ]: print(revenue_map)
```

```
{'Australia': 429.39, 'Austria': 685.13, 'Bahrain': 548.8, 'Belgium': 629.02,
'Bermuda': 1253.14, 'Channel Islands': 1065.12, 'Cyprus': 2879.19, 'Denmark':
7595.18, 'EIRE': 21778.3, 'France': 8027.59, 'Germany': 15347.08, 'Greece':
522.73, 'Italy': 699.22, 'Japan': 110.4, 'Netherlands': 24241.3, 'Poland':
318.86, 'Portugal': 2399.45, 'Spain': 1456.52, 'Sweden': 1357.58, 'Switzerland':
831.62, 'United Arab Emirates': 1201.52, 'United Kingdom': 670999.26}
```

```
[ ]: highest_revenue_country = sorted(
         revenue_map, key=lambda x: revenue_map[x], reverse=True
     )[0]
     print(highest_revenue_country, revenue_map[highest_revenue_country])
```

```
United Kingdom 670999.26
```

Using Hadoop MapReduce

mapper.py

```
#!/home/vinayak/anaconda3/bin/python
"""/home/vinayak/Desktop/PGM_BDS_Assignment/BDS_Assignment_2/query_2/mapper.py"""

import sys
from datetime import datetime

banned_stock_codes = ["ADJUST","ADJUST2","AMAZONFEE","B",
                      "BANK CHARGES","C2","C3","D",
                      "DOT","GIFT","M",
                      "POST","S","TEST001","TEST002","m"]
```

```
banned_stock_codes = set(banned_stock_codes)

# Read the input from standard input
for line in sys.stdin:
    # Remove all the whitespaces
    line = line.strip()

    # Split any given record into corresponding fields
    rec_no, invoice, stock_code, *desc, quantity, invoice_date, price,\
    cust_id, country = line.split(",")

    # We don't want to do any computation for the header of the file,
    # hence skip the first line
    if not country == "Country":

        # Compute the revenue generated
        price = float(price); quantity = int(quantity)
        revenue = price * quantity

        # Parse the date appropriately
        # Some dates are delimited by / and some others are delimited by -
        # Some dates have hour-min-sec in time and some others have hour-min
        if len(invoice_date.split(":")) == 2:
            if "/" in invoice_date:
                dt = datetime.strptime(invoice_date, "%m/%d/%Y %H:%M")
            else:
                dt = datetime.strptime(invoice_date, "%m-%d-%Y %H:%M")
        else:
            if "/" in invoice_date:
                dt = datetime.strptime(invoice_date, "%m/%d/%Y %H:%M:%S")
            else:
                dt = datetime.strptime(invoice_date, "%m-%d-%Y %H:%M:%S")

        # Only print the dates which belong to the month of March 2010
        if (dt.month == 3) and (dt.year == 2010):
            # Check if the revenue, quantity and prices are positive
            if (price > 0) and (quantity > 0) and (revenue > 0):
                # Check if the given stock code is not in the banned stock codes
                if stock_code not in banned_stock_codes:
                    print(f"{country},{revenue}")
```

---

reducer.py

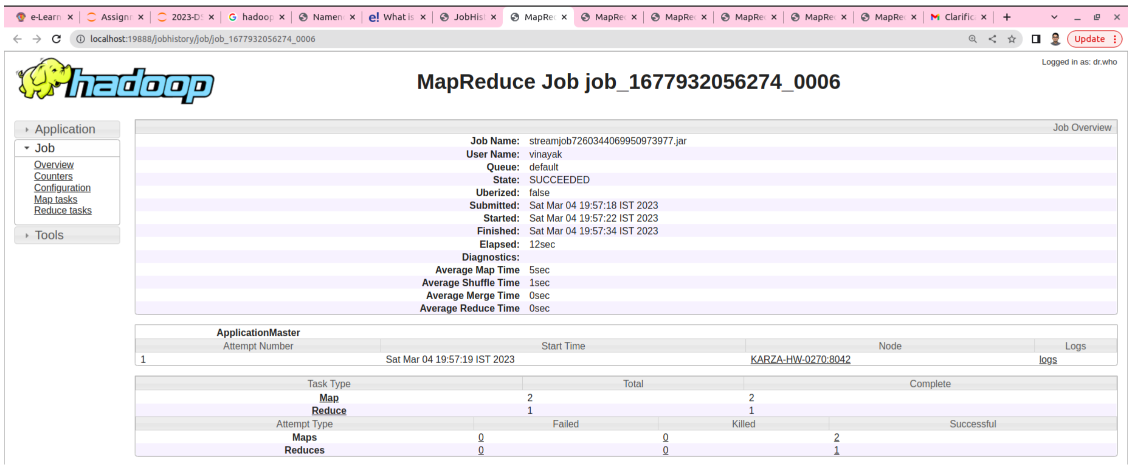
```
#!/home/vinayak/anaconda3/bin/python
"""/home/vinayak/Desktop/PGM_BDS_Assignment/BDS_Assignment_2/query_2/reducer.py"""

import sys
```

```python
from collections import defaultdict
country_revenue_map = defaultdict(lambda: 0)

# Read the input from standard input
for line in sys.stdin:
    # Remove the leading and trailing whitespaces
    line = line.strip()

    # Parse the mapper input
    country, revenue = line.split(',')

    # Sum the revenue into the respective country's account
    country_revenue_map[country] += float(revenue)

# Figure out the country with highest revenue
highest_revenue_country = sorted(country_revenue_map, \
                                 key = lambda x: country_revenue_map[x], \
                                 reverse = True)[0]

# Print the highest revenue country and the highest revenue to console
print(f"For 03/2010; Highest Revenue Country: {highest_revenue_country}\
Highest Revenue: {country_revenue_map[highest_revenue_country]:.3f}")
```

```
hadoop jar /home/vinayak/hadoop-3.2.4/share/hadoop/tools/lib/hadoop-streaming-3.2.4.jar
-file /home/vinayak/Desktop/PGM_BDS_Assignment/BDS_Assignment_2/query_2/mapper.py
-mapper mapper.py
-file /home/vinayak/Desktop/PGM_BDS_Assignment/BDS_Assignment_2/query_2/reducer.py
-reducer reducer.py
-input /hdfs_input/assignment2_retail_data_utf8.csv
-output /home/query2_output/
```

```
[ ]: !hdfs dfs -cat /home/query2_output/part-00000
```

```
For 03/2010; Highest Revenue Country: United Kingdom | Highest Revenue:
670999.261
```



MapReduce Job job_1677932056274_0006

Job Overview

Job Name:	streamjob7260344069950973977.jar
User Name:	vinayak
Queue:	default
State:	SUCCEEDED
Uberized:	false
Submitted:	Sat Mar 04 19:57:18 IST 2023
Started:	Sat Mar 04 19:57:22 IST 2023
Finished:	Sat Mar 04 19:57:34 IST 2023
Elapsed:	12sec
Diagnostics:	
Average Map Time	5sec
Average Shuffle Time	1sec
Average Merge Time	0sec
Average Reduce Time	0sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Sat Mar 04 19:57:19 IST 2023	KARZA-HW-0270:8042	logs

Task Type	Total	Complete
Map	2	2
Reduce	1	1

Attempt Type	Failed	Killed	Successful
Maps	0	0	2
Reduces	0	0	1

Query 3 Computation

Month of 2010 in which maximum number of items were sold.

```
[ ]: num_items = {}
     for my, sub_df in df.groupby(by="Month_Year"):
         sub_df = sub_df[sub_df.Quantity > 0]
         sub_df = sub_df[sub_df.Price > 0]
         sub_df = sub_df[sub_df.InvoiceDate.dt.year == 2010]
         items = sub_df.Quantity.sum()
         num_items[my] = round(items, 2)
```

```
[ ]: highest_selling_month = sorted(num_items, key=lambda x: num_items[x],␣
      ↪reverse=True)[0]
     print(
         f"For 2010; Highest Selling Month: {highest_selling_month} | Items Sold:␣
      ↪{num_items[highest_selling_month]}"
     )
```

```
For 2010; Highest Selling Month: 11/2010 | Items Sold: 727556
```

Using Hadoop MapReduce

mapper.py

```
#!/home/vinayak/anaconda3/bin/python
"""/home/vinayak/Desktop/PGM_BDS_Assignment/BDS_Assignment_2/query_3/mapper.py"""

import sys
from datetime import datetime

banned_stock_codes = ["ADJUST","ADJUST2","AMAZONFEE","B",
                      "BANK CHARGES","C2","C3","D",
                      "DOT","GIFT","M",
                      "POST","S","TEST001","TEST002","m"]
banned_stock_codes = set(banned_stock_codes)

# Read the input from standard input
for line in sys.stdin:
    # Remove all the whitespaces
    line = line.strip()

    # Split any given record into corresponding fields
    rec_no, invoice, stock_code, *desc, quantity, invoice_date, \
    price, cust_id, country = line.split(",")

    # We don't want to do any computation for the header of the file,
    # hence skip the first line
    if not country == "Country":
        # Cast the price and quantity appropriately
        price = float(price); quantity = int(quantity)

        # Parse the date appropriately
        # Some dates are delimited by / and some others are delimited by -
        # Some dates have hour-min-sec in time and some others have hour-min
        if len(invoice_date.split(":")) == 2:
            if "/" in invoice_date:
                dt = datetime.strptime(invoice_date, "%m/%d/%Y %H:%M")
            else:
                dt = datetime.strptime(invoice_date, "%m-%d-%Y %H:%M")
        else:
            if "/" in invoice_date:
                dt = datetime.strptime(invoice_date, "%m/%d/%Y %H:%M:%S")
            else:
                dt = datetime.strptime(invoice_date, "%m-%d-%Y %H:%M:%S")

        # Only print the those transactions which happened in the year 2010
        if dt.year == 2010:
            # Check if the quantity and prices are positive
            if (price > 0) and (quantity > 0):
                # Check if given stock code is not in banned stock codes
                if not stock_code in banned_stock_codes:
                    print(f"{dt.month},{quantity}")
```

---

reducer.py

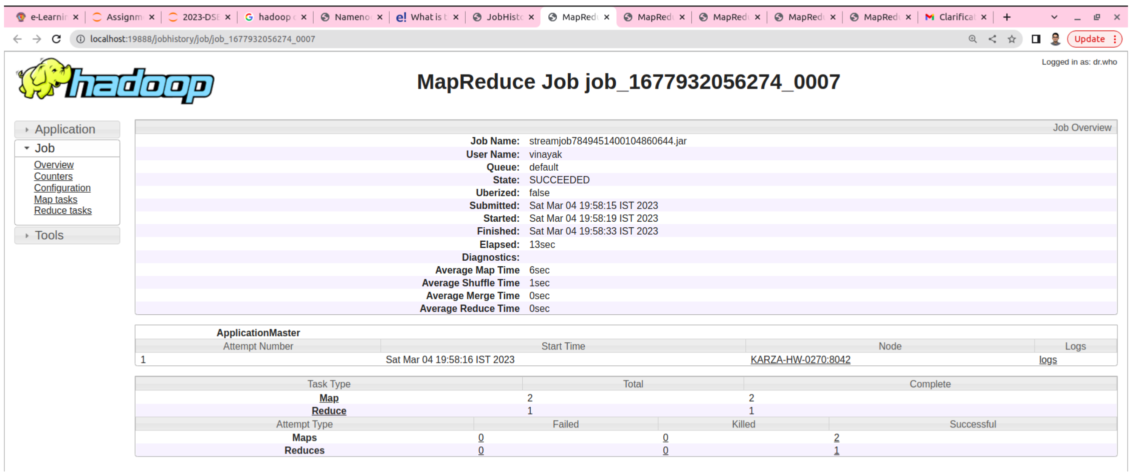
```
#!/home/vinayak/anaconda3/bin/python
"""/home/vinayak/Desktop/PGM_BDS_Assignment/BDS_Assignment_2/query_3/reducer.py"""

import sys
from collections import defaultdict
month_quantity_map = defaultdict(lambda: 0)

# Read the input from standard input
for line in sys.stdin:
    # Remove the leading and trailing whitespaces
    line = line.strip()

    # Parse the mapper input
    month, quantity = line.split(',')

    # Sum the quantity into the respective month's account
    month_quantity_map[month] += int(quantity)

# Figure out the month with highest items sold
highest_selling_month = sorted(month_quantity_map, key = lambda x: month_quantity_map[x], \
                               reverse = True)[0]
```

```python
# Print the highest revenue country and the highest revenue to console
print(f"For 2010; Highest Selling Month: {highest_selling_month:0<2} | \
Items Sold: {month_quantity_map[highest_selling_month]}")
```

```
hadoop jar /home/vinayak/hadoop-3.2.4/share/hadoop/tools/lib/hadoop-streaming-3.2.4.jar
-file /home/vinayak/Desktop/PGM_BDS_Assignment/BDS_Assignment_2/query_3/mapper.py
-mapper mapper.py
-file /home/vinayak/Desktop/PGM_BDS_Assignment/BDS_Assignment_2/query_3/reducer.py
-reducer reducer.py
-input /hdfs_input/assignment2_retail_data_utf8.csv
-output /home/query3_output/
```

```
[ ]: !hdfs dfs -cat /home/query3_output/part-00000
```

```
For 2010; Highest Selling Month: 11 | Items Sold: 727556
```



MapReduce Job job_1677932056274_0007

Job Overview

Job Name:	streamjob7849451400104860644.jar
User Name:	vinayak
Queue:	default
State:	SUCCEEDED
Uberized:	false
Submitted:	Sat Mar 04 19:58:15 IST 2023
Started:	Sat Mar 04 19:58:19 IST 2023
Finished:	Sat Mar 04 19:58:33 IST 2023
Elapsed:	13sec
Diagnostics:	
Average Map Time	6sec
Average Shuffle Time	1sec
Average Merge Time	0sec
Average Reduce Time	0sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Sat Mar 04 19:58:16 IST 2023	KARZA-HW-0270:8042	logs

Task Type	Total	Complete
Map	2	2
Reduce	1	1

Attempt Type	Failed	Killed	Successful
Maps	0	0	2
Reduces	0	0	1

Query 4 Computation

In the month of January 2010, find the country in which maximum number of items were sold

```
[ ]: country_items = {}
     for country, sub_df in df[df["Month_Year"] == "1/2010"].groupby(by="Country"):
         sub_df = sub_df[sub_df.Quantity > 0]
         sub_df = sub_df[sub_df.Price > 0]
         items = sub_df.Quantity.sum()
         country_items[country] = items
```

```
[ ]: highest_selling_country = sorted(
         country_items, key=lambda x: country_items[x], reverse=True
     )[0]
     print(
         f"For 01/2010; Country selling max items: {highest_selling_country} |␣
      ↪Volume Sold: {country_items[highest_selling_country]}"
     )
```

```
For 01/2010; Country selling max items: United Kingdom | Volume Sold: 257473
```

Using Hadoop MapReduce

mapper.py

```
#!/home/vinayak/anaconda3/bin/python
"""/home/vinayak/Desktop/PGM_BDS_Assignment/BDS_Assignment_2/query_4/mapper.py"""

import sys
from datetime import datetime

banned_stock_codes = ["ADJUST","ADJUST2","AMAZONFEE","B",
                      "BANK CHARGES","C2","C3","D",
                      "DOT","GIFT","M",
                      "POST","S","TEST001","TEST002","m"]
banned_stock_codes = set(banned_stock_codes)

# Read the input from standard input
for line in sys.stdin:
    # Remove all the whitespaces
    line = line.strip()

    # Split any given record into corresponding fields
    rec_no, invoice, stock_code, *desc, quantity, invoice_date, \
    price, cust_id, country = line.split(",")

    # We don't want to do any computation for the header of the file,
    # hence skip the first line
    if not country == "Country":
        price = float(price); quantity = int(quantity)

        # Parse the date appropriately
        # Some dates are delimited by / and some others are delimited by -
        # Some dates have hour-min-sec in time and some others have hour-min
        if len(invoice_date.split(":")) == 2:
            if "/" in invoice_date:
                dt = datetime.strptime(invoice_date, "%m/%d/%Y %H:%M")
            else:
                dt = datetime.strptime(invoice_date, "%m-%d-%Y %H:%M")
        else:
            if "/" in invoice_date:
                dt = datetime.strptime(invoice_date, "%m/%d/%Y %H:%M:%S")
            else:
                dt = datetime.strptime(invoice_date, "%m-%d-%Y %H:%M:%S")

        # Only select the dates which belong to the month of January 2010
        if (dt.month == 1) and (dt.year == 2010):
            # Check if the quantity and prices are positive
            if (price > 0) and (quantity > 0):
                # Check if the stock code is not in the banned stock codes
                if stock_code not in banned_stock_codes:
                    print(f"{country},{quantity}")
```

---

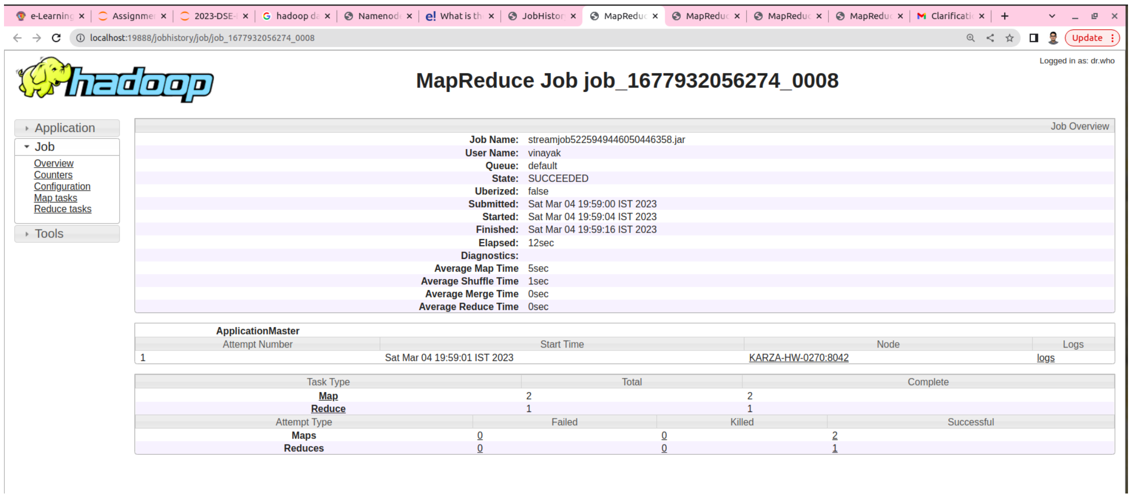
reducer.py

```
#!/home/vinayak/anaconda3/bin/python
"""/home/vinayak/Desktop/PGM_BDS_Assignment/BDS_Assignment_2/query_4/reducer.py"""

import sys
from collections import defaultdict
country_quantity_map = defaultdict(lambda: 0)

# Read the input from standard input
for line in sys.stdin:
    # Remove the leading and trailing whitespaces
    line = line.strip()

    # Parse the mapper input
    country, quantity = line.split(',')

    # Sum the revenue into the respective country's account
    country_quantity_map[country] += int(quantity)

# Figure out the country with highest quantity of items sold
highest_selling_country = sorted(country_quantity_map, key = lambda x: country_quantity_map[x]
                                    reverse = True)[0]

print(f"For 01/2010; Country selling max items: {highest_selling_country}\
| Volume Sold: {country_quantity_map[highest_selling_country]}")
```

```
hadoop jar /home/vinayak/hadoop-3.2.4/share/hadoop/tools/lib/hadoop-streaming-3.2.4.jar
-file /home/vinayak/Desktop/PGM_BDS_Assignment/BDS_Assignment_2/query_4/mapper.py
-mapper mapper.py
-file /home/vinayak/Desktop/PGM_BDS_Assignment/BDS_Assignment_2/query_4/reducer.py
-reducer reducer.py
-input /hdfs_input/assignment2_retail_data_utf8.csv
-output /home/query4_output/
```

[ ]:
```
!hdfs dfs -cat /home/query4_output/part-00000
```

```
For 01/2010; Country selling max items: United Kingdom | Volume Sold: 257473
```



```
localhost:19888/jobhistory/job/job_1677932056274_0008
```

MapReduce Job job_1677932056274_0008

Job Overview

Job Name:	streamjob5225949446050446358.jar
User Name:	vinayak
Queue:	default
State:	SUCCEEDED
Uberized:	false
Submitted:	Sat Mar 04 19:59:00 IST 2023
Started:	Sat Mar 04 19:59:04 IST 2023
Finished:	Sat Mar 04 19:59:16 IST 2023
Elapsed:	12sec
Diagnostics:	
Average Map Time	5sec
Average Shuffle Time	1sec
Average Merge Time	0sec
Average Reduce Time	0sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Sat Mar 04 19:59:01 IST 2023	KARZA-HW-0270:8042	logs

Task Type	Total	Complete
Map	2	2
Reduce	1	1

Attempt Type	Failed	Killed	Successful
Maps	0	0	2
Reduces	0	0	1

Query 5 Computation

The StockCode of the item with the highest number of sales in the "given country" in the year 2010

We are considering `given country` as `Germany`

```
[ ]: stock_code_sales = {}
     for stock_code, sub_df in df[
         (df.InvoiceDate.dt.year == 2010) & (df.Country == "Germany")
     ].groupby(by="StockCode"):
         sub_df = sub_df[sub_df.Quantity > 0]
         sub_df = sub_df[sub_df.Price > 0]
         sales = sub_df.Quantity.sum()
         stock_code_sales[stock_code] = sales
```

```
[ ]: highest_sale_item = sorted(
         stock_code_sales, key=lambda x: stock_code_sales[x], reverse=True
     )[0]
     print(
         f"For Germany; Stock code with highest sale: {highest_sale_item} | Volume␣
      ↪Sold: {stock_code_sales[highest_sale_item]}"
     )
```

```
For Germany; Stock code with highest sale: 22326 | Volume Sold: 1543
```

Using Hadoop MapReduce

mapper.py

```
#!/home/vinayak/anaconda3/bin/python
"""/home/vinayak/Desktop/PGM_BDS_Assignment/BDS_Assignment_2/query_5/mapper.py"""

import sys
from datetime import datetime

banned_stock_codes = ["ADJUST","ADJUST2","AMAZONFEE","B",
                      "BANK CHARGES","C2","C3","D",
                      "DOT","GIFT","M",
                      "POST","S","TEST001","TEST002","m"]
banned_stock_codes = set(banned_stock_codes)

# Read the input from standard input
for line in sys.stdin:
    # Remove all the whitespaces
    line = line.strip()

    # Split any given record into corresponding fields
    rec_no, invoice, stock_code, *desc, quantity, invoice_date, price, \
    cust_id, country = line.split(",")
    # We only want to consider Germany's sales
    if country == "Germany":

        price = float(price); quantity = int(quantity)

        # Parse the date appropriately
        # Some dates are delimited by / and some others are delimited by -
        # Some dates have hour-min-sec in time and some others have hour-min
        if len(invoice_date.split(":")) == 2:
            if "/" in invoice_date:
                dt = datetime.strptime(invoice_date, "%m/%d/%Y %H:%M")
            else:
                dt = datetime.strptime(invoice_date, "%m-%d-%Y %H:%M")
        else:
            if "/" in invoice_date:
                dt = datetime.strptime(invoice_date, "%m/%d/%Y %H:%M:%S")
            else:
                dt = datetime.strptime(invoice_date, "%m-%d-%Y %H:%M:%S")

        # Only select the dates which belong to the month of December 2010
        if (dt.year == 2010):
            # Check if the quantity and prices are positive
            if (price > 0) and (quantity > 0):
                # Check if the stock code is not in the banned stock codes
                if stock_code not in banned_stock_codes:
                    print(f"{stock_code},{quantity}")
```

---

reducer.py

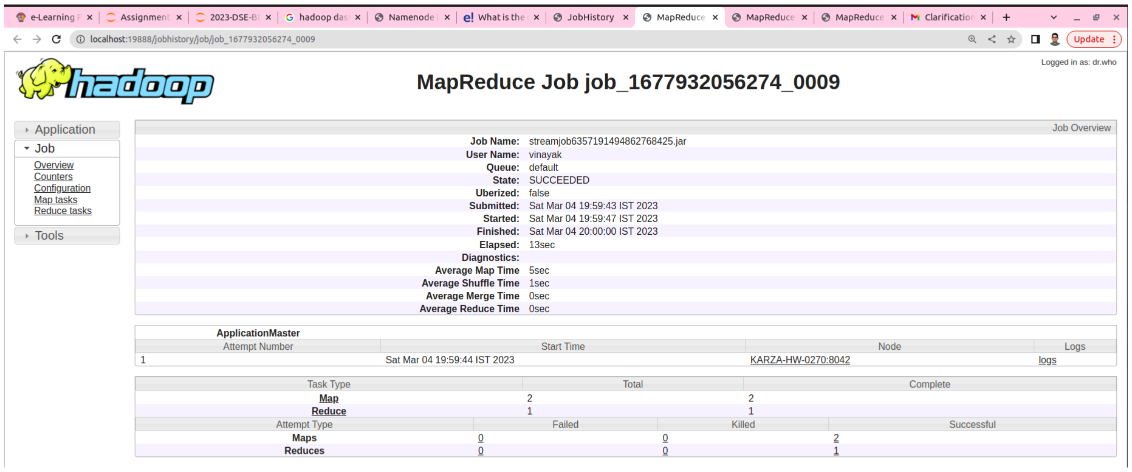
```
#!/home/vinayak/anaconda3/bin/python
"""/home/vinayak/Desktop/PGM_BDS_Assignment/BDS_Assignment_2/query_5/reducer.py"""

import sys
from collections import defaultdict
stock_code_sales = defaultdict(lambda: 0)

# Read the input from standard input
for line in sys.stdin:
    # Remove the leading and trailing whitespaces
    line = line.strip()

    # Parse the mapper input
    stock_code, quantity = line.split(',')

    # Sum the revenue into the respective country's account
    stock_code_sales[stock_code] += int(quantity)
```

```
# Figure out the country with highest quantity of items sold
highest_sale_item = sorted(stock_code_sales, key = lambda x: stock_code_sales[x], \
                           reverse = True)[0]

print(f"For Germany; Stock Code giving highest sale: {highest_sale_item} \
| Volume Sold: {stock_code_sales[highest_sale_item]:.2f}")
```

```
hadoop jar /home/vinayak/hadoop-3.2.4/share/hadoop/tools/lib/hadoop-streaming-3.2.4.jar
-file /home/vinayak/Desktop/PGM_BDS_Assignment/BDS_Assignment_2/query_5/mapper.py
-mapper mapper.py
-file /home/vinayak/Desktop/PGM_BDS_Assignment/BDS_Assignment_2/query_5/reducer.py
-reducer reducer.py
-input /hdfs_input/assignment2_retail_data_utf8.csv
-output /home/query5_output/
```

[ ]:
```
!hdfs dfs -cat /home/query5_output/part-00000
```

```
For Germany; Stock Code giving highest sale: 22326 | Volume Sold: 1543.00
```



MapReduce Job job_1677932056274_0009

Job Overview

Job Name:	streamjob6357191494862768425.jar
User Name:	vinayak
Queue:	default
State:	SUCCEEDED
Uberized:	false
Submitted:	Sat Mar 04 19:59:43 IST 2023
Started:	Sat Mar 04 19:59:47 IST 2023
Finished:	Sat Mar 04 20:00:00 IST 2023
Elapsed:	13sec
Diagnostics:	
Average Map Time	5sec
Average Shuffle Time	1sec
Average Merge Time	0sec
Average Reduce Time	0sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Sat Mar 04 19:59:44 IST 2023	KARZA-HW-0270:8042	logs

Task Type	Total	Complete
Map	2	2
Reduce	1	1

Attempt Type	Failed	Killed	Successful
Maps	0	0	2
Reduces	0	0	1

Query 6 Computation

StockCode of the item for which the maximum revenue was received by sales in the month of December 2010.

```
[ ]: stock_revenue_map = {}
     for stock_code, sub_df in df[df["Month_Year"] == "12/2010"].
      ↪groupby(by="StockCode"):
         sub_df = sub_df[sub_df.Quantity > 0]
         sub_df = sub_df[sub_df.Price > 0]
         revenue = (sub_df.Quantity * sub_df.Price).sum()
         stock_revenue_map[stock_code] = round(revenue, 2)
```

```
[ ]: highest_revenue_item = sorted(
         stock_revenue_map, key=lambda x: stock_revenue_map[x], reverse=True
     )[0]
     print(
         f"For 12/2010; Stock Code giving highest revenue: {highest_revenue_item} |␣
      ↪Revenue Accrued: {stock_revenue_map[highest_revenue_item]}"
     )
```

```
For 12/2010; Stock Code giving highest revenue: 22423 | Revenue Accrued:
13478.98
```

Using Hadoop MapReduce

mapper.py

```
#!/home/vinayak/anaconda3/bin/python
"""/home/vinayak/Desktop/PGM_BDS_Assignment/BDS_Assignment_2/query_6/mapper.py"""

import sys
from datetime import datetime

banned_stock_codes = ["ADJUST","ADJUST2","AMAZONFEE","B",
                      "BANK CHARGES","C2","C3","D",
                      "DOT","GIFT","M",
                      "POST","S","TEST001","TEST002","m"]
banned_stock_codes = set(banned_stock_codes)

# Read the input from standard input
for line in sys.stdin:
    # Remove all the whitespaces
    line = line.strip()

    # Split any given record into corresponding fields
    rec_no, invoice, stock_code, *desc, quantity, invoice_date, price, \
    cust_id, country = line.split(",")

    # We don't want to do any computation for the header of the file,
    # hence skip the first line
    if not country == "Country":

        price = float(price); quantity = int(quantity)
        revenue = price * quantity

        # Parse the date appropriately
        # Some dates are delimited by / and some others are delimited by -
        # Some dates have hour-min-sec in time and some others have hour-min
        if len(invoice_date.split(":")) == 2:
            if "/" in invoice_date:
                dt = datetime.strptime(invoice_date, "%m/%d/%Y %H:%M")
            else:
                dt = datetime.strptime(invoice_date, "%m-%d-%Y %H:%M")
        else:
            if "/" in invoice_date:
                dt = datetime.strptime(invoice_date, "%m/%d/%Y %H:%M:%S")
            else:
                dt = datetime.strptime(invoice_date, "%m-%d-%Y %H:%M:%S")

        # Only select the dates which belong to the month of December 2010
        if (dt.month == 12) and (dt.year == 2010):
            # Check if the quantity and prices are positive
            if (price > 0) and (quantity > 0) and (revenue > 0):
                # Check if the stock code is not in the banned stock codes
                if stock_code not in banned_stock_codes:
                    print(f"{stock_code},{revenue}")
```

---

reducer.py

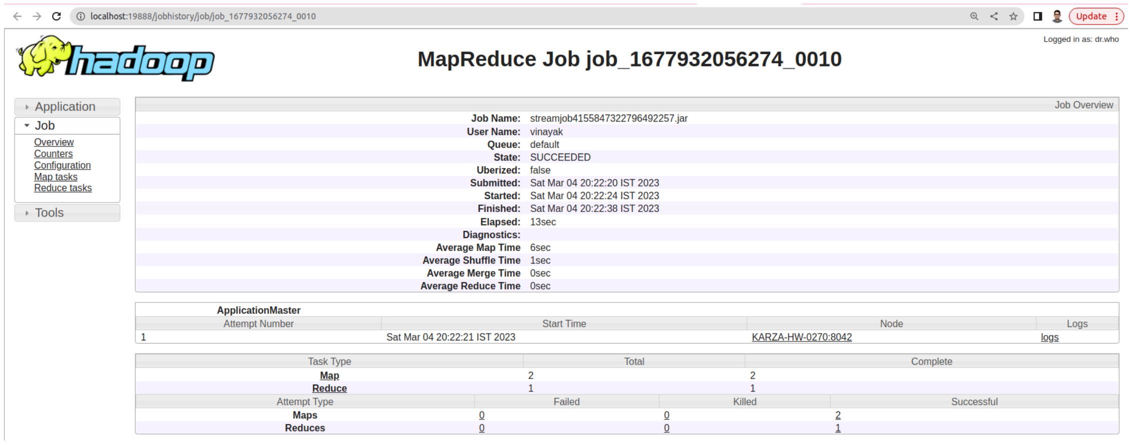
```
#!/home/vinayak/anaconda3/bin/python
"""/home/vinayak/Desktop/PGM_BDS_Assignment/BDS_Assignment_2/query_6/reducer.py"""

import sys
from collections import defaultdict
stock_revenue_map = defaultdict(lambda: 0)

# Read the input from standard input
for line in sys.stdin:
    # Remove the leading and trailing whitespaces
    line = line.strip()

    # Parse the mapper input
    stock_code, revenue = line.split(',')

    # Sum the revenue into the respective country's account
    stock_revenue_map[stock_code] += float(revenue)
```

```
# Figure out the country with highest quantity of items sold
highest_revenue_item = sorted(stock_revenue_map, key = lambda x: stock_revenue_map[x], \
                              reverse = True)[0]

print(f"For 12/2010; Stock Code giving highest revenue: {highest_revenue_item} \
| Revenue Accrued: {stock_revenue_map[highest_revenue_item]:.2f}")
```

```
hadoop jar /home/vinayak/hadoop-3.2.4/share/hadoop/tools/lib/hadoop-streaming-3.2.4.jar
-file /home/vinayak/Desktop/PGM_BDS_Assignment/BDS_Assignment_2/query_6/mapper.py
-mapper mapper.py
-file /home/vinayak/Desktop/PGM_BDS_Assignment/BDS_Assignment_2/query_6/reducer.py
-reducer reducer.py
-input /hdfs_input/assignment2_retail_data_utf8.csv
-output /home/query6_output/
```

```
[ ]: !hdfs dfs -cat /home/query6_output/part-00000
```

```
For 12/2010; Stock Code giving highest revenue: 22423 | Revenue Accrued:
13478.98
```



localhost:19888/jobhistory/job/job_1677932056274_0010

Logged in as: dr.who

MapReduce Job job_1677932056274_0010

Application
Job
Overview
Counters
Configuration
Map tasks
Reduce tasks
Tools

Job Overview

Job Name:	streamjob4155847322796492257.jar
User Name:	vinayak
Queue:	default
State:	SUCCEEDED
Uberized:	false
Submitted:	Sat Mar 04 20:22:20 IST 2023
Started:	Sat Mar 04 20:22:24 IST 2023
Finished:	Sat Mar 04 20:22:38 IST 2023
Elapsed:	13sec
Diagnostics:	
Average Map Time	6sec
Average Shuffle Time	1sec
Average Merge Time	0sec
Average Reduce Time	0sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Sat Mar 04 20:22:21 IST 2023	KARZA-HW-0270:8042	logs

Task Type	Total	Complete
Map	2	2
Reduce	1	1

Attempt Type	Failed	Killed	Successful
Maps	0	0	2
Reduces	0	0	1

Query 7 Computation

The country in which minimum number of sales happened in 2010.

```
[ ]: country_sales = {}
     for country, sub_df in df[df.InvoiceDate.dt.year == 2010].groupby(by="Country"):
         sub_df = sub_df[sub_df.Quantity > 0]
         sub_df = sub_df[sub_df.Price > 0]
         sales = sub_df.Quantity.sum()
         country_sales[country] = round(sales, 2)
```

```
[ ]: lowest_2010_sales_country = sorted(
         country_sales, key=lambda x: country_sales[x], reverse=False
     )[0]
     print(
         f"For 2010, Country with lowest quantity sales: {lowest_2010_sales_country}␣
       ↪| Total items sold: {country_sales[lowest_2010_sales_country]}"
     )
```

```
For 2010, Country with lowest quantity sales: Lebanon | Total items sold: 72
```

Using Hadoop MapReduce

mapper.py

```
#!/home/vinayak/anaconda3/bin/python
"""/home/vinayak/Desktop/PGM_BDS_Assignment/BDS_Assignment_2/query_7/mapper.py"""

import sys
from datetime import datetime

banned_stock_codes = ["ADJUST","ADJUST2","AMAZONFEE","B",
                      "BANK CHARGES","C2","C3","D",
                      "DOT","GIFT","M",
                      "POST","S","TEST001","TEST002","m"]
banned_stock_codes = set(banned_stock_codes)

# Read the input from standard input
for line in sys.stdin:
    # Remove all the whitespaces
    line = line.strip()

    # Split any given record into corresponding fields
    rec_no, invoice, stock_code, *desc, quantity, invoice_date, price,\
    cust_id, country = line.split(",")

    # We don't want to do any computation for the header of the file,
    # hence skip the first line
    if not country == "Country":
        price = float(price); quantity = int(quantity)
        revenue = price * quantity

        # Parse the date appropriately
        # Some dates are delimited by / and some others are delimited by -
        # Some dates have hour-min-sec in time and some others have hour-min
        if len(invoice_date.split(":")) == 2:
            if "/" in invoice_date:
                dt = datetime.strptime(invoice_date, "%m/%d/%Y %H:%M")
            else:
                dt = datetime.strptime(invoice_date, "%m-%d-%Y %H:%M")
        else:
            if "/" in invoice_date:
                dt = datetime.strptime(invoice_date, "%m/%d/%Y %H:%M:%S")
            else:
                dt = datetime.strptime(invoice_date, "%m-%d-%Y %H:%M:%S")

        # Only select the dates which belong to the year 2010
        if dt.year == 2010:
            # Check if the quantity and prices are positive
            if (price > 0) and (quantity > 0):
                # Check if the stock code is not in the banned stock codes
                if stock_code not in banned_stock_codes:
                    print(f"{country},{quantity}")
```

reducer.py

```
#!/home/vinayak/anaconda3/bin/python
"""/home/vinayak/Desktop/PGM_BDS_Assignment/BDS_Assignment_2/query_7/reducer.py"""

import sys
from collections import defaultdict
country_sales_map = defaultdict(lambda: 0)

# Read the input from standard input
for line in sys.stdin:
    # Remove the leading and trailing whitespaces
    line = line.strip()

    # Parse the mapper input
    country, quantity = line.split(',')

    # Sum the revenue into the respective country's account
    country_sales_map[country] += int(quantity)

# Figure out the country with highest quantity of items sold
lowest_sales_item = sorted(country_sales_map, key = lambda x: country_sales_map[x],\
                           reverse = False)[0]
```

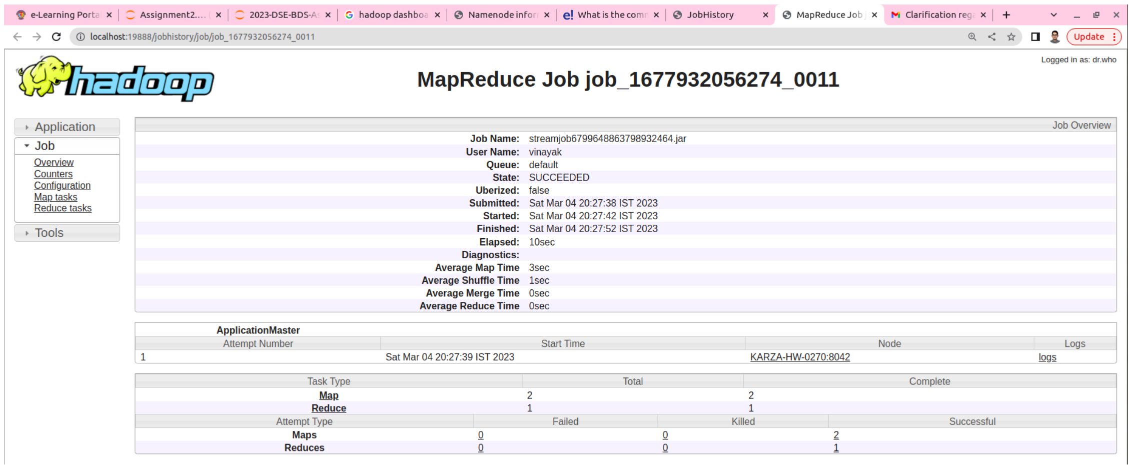
```
print(f"For the year 2010; Country with lowest quantity of sales: {lowest_sales_item} \
| Total items sold: {country_sales_map[lowest_sales_item]:.2f}")
```

Execute the hadoop map-reduce job

```
hadoop jar /home/vinayak/hadoop-3.2.4/share/hadoop/tools/lib/hadoop-streaming-3.2.4.jar
-file /home/vinayak/Desktop/PGM_BDS_Assignment/BDS_Assignment_2/query_7/mapper.py
-mapper mapper.py
-file /home/vinayak/Desktop/PGM_BDS_Assignment/BDS_Assignment_2/query_7/reducer.py
-reducer reducer.py
-input /hdfs_input/assignment2_retail_data_utf8.csv
-output /home/query7_output/
```

```
[ ]: !hdfs dfs -cat /home/query7_output/part-00000
```

```
For the year 2010; Country with lowest quantity of sales: Lebanon | Total items
sold: 72.00
```



localhost:19888/jobhistory/job/job_1677932056274_0011

Logged in as: dr.who

MapReduce Job job_1677932056274_0011

Job Overview

Job Name:	streamjob6799648863798932464.jar
User Name:	vinayak
Queue:	default
State:	SUCCEEDED
Uberized:	false
Submitted:	Sat Mar 04 20:27:38 IST 2023
Started:	Sat Mar 04 20:27:42 IST 2023
Finished:	Sat Mar 04 20:27:52 IST 2023
Elapsed:	10sec
Diagnostics:	
Average Map Time	3sec
Average Shuffle Time	1sec
Average Merge Time	0sec
Average Reduce Time	0sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Sat Mar 04 20:27:39 IST 2023	KARZA-HW-0270:8042	logs

Task Type	Total	Complete
Map	2	2
Reduce	1	1

Attempt Type	Failed	Killed	Successful
Maps	0	0	2
Reduces	0	0	1