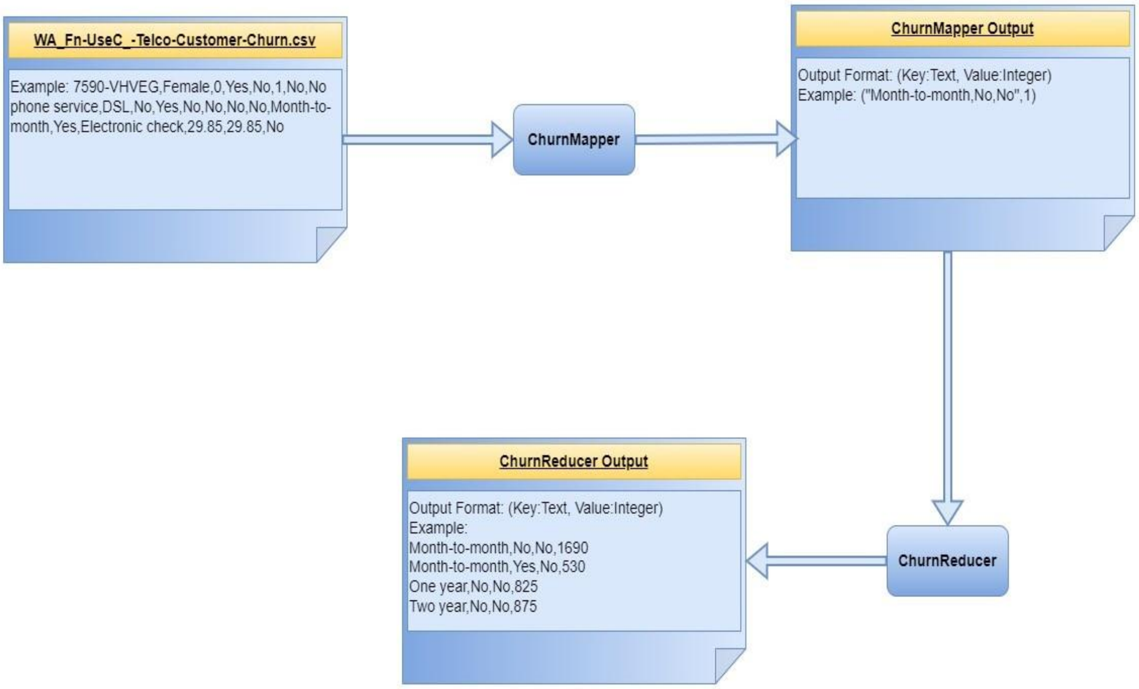
Data Flow:

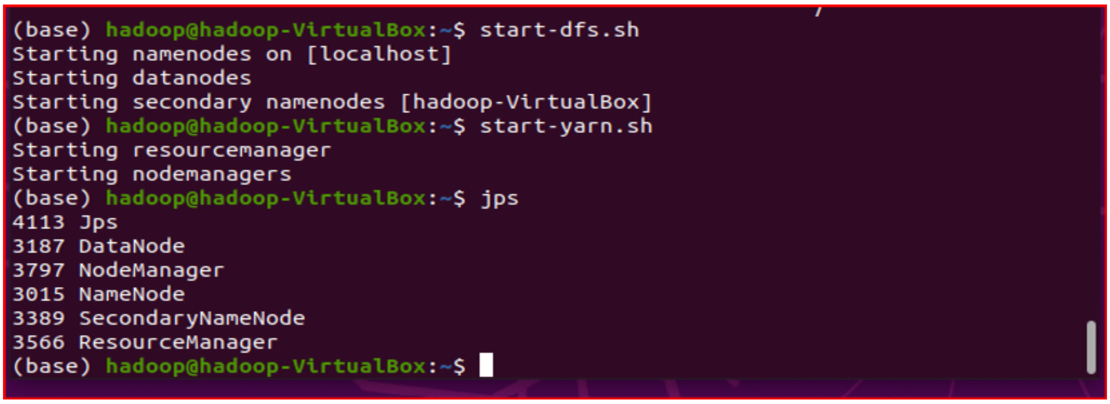

WA_Fn-UseC_-Telco-Customer-Churn.csv
Example: 7590-VHVEG,Female,0,Yes,No,1,No,No phone service,DSL,No,Yes,No,No,No,No,Month-to-month,Yes,Electronic check,29.85,29.85,No
ChurnMapper
ChurnMapper Output
Output Format: (Key:Text, Value:Integer)
Example: ("Month-to-month,No,No",1)
ChurnReducer
ChurnReducer Output
Output Format: (Key:Text, Value:Integer)
Example:
Month-to-month,No,No,1690
Month-to-month,Yes,No,530
One year,No,No,825
Two year,No,No,875

2.1 Implementation Steps:

1. As per the MapReduce programming model, create the below three classes and generate a .jar file for this requirement.
    a) ChurnMapper
    b) ChurnReducer
    c) ChurnAnalysis

- The complete implementation and the code are in the separate file '**Mapreduce_ChurnAnalysis**' and the file has been submitted along with this coursework.

2. Firstly, start the HDFS file system using commands 'start-dfs.sh' and 'start-yarn.sh' and confirm if all the services are running or not using a command called 'jps'

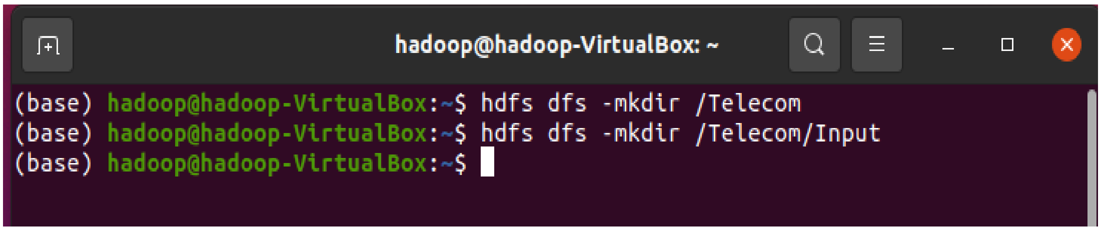


```
(base) hadoop@hadoop-VirtualBox:~$ start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [hadoop-VirtualBox]
(base) hadoop@hadoop-VirtualBox:~$ start-yarn.sh
Starting resourcemanager
Starting nodemanagers
(base) hadoop@hadoop-VirtualBox:~$ jps
4113 Jps
3187 DataNode
3797 NodeManager
3015 NameNode
3389 SecondaryNameNode
3566 ResourceManager
(base) hadoop@hadoop-VirtualBox:~$
```

Figure 1

- As per figure 1, all the services are up and running.

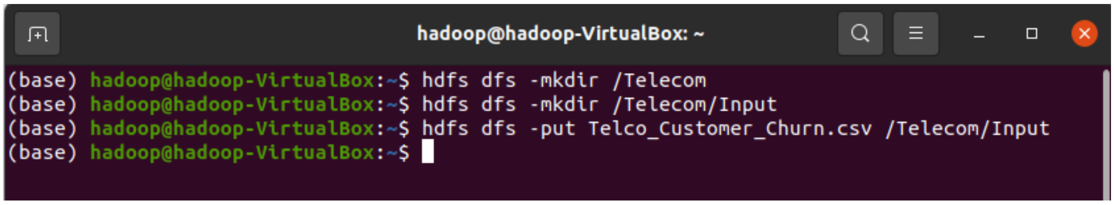
3. Create the directory in the HDFS file system to provide our input to the HDFS and execute the MapReduce job. In this case, I have created the 'Telecom' directory and then created the 'Input' directory inside the 'Telecom' directory as shown below.



```
(base) hadoop@hadoop-VirtualBox:~$ hdfs dfs -mkdir /Telecom
(base) hadoop@hadoop-VirtualBox:~$ hdfs dfs -mkdir /Telecom/Input
(base) hadoop@hadoop-VirtualBox:~$ 
```

Figure 2

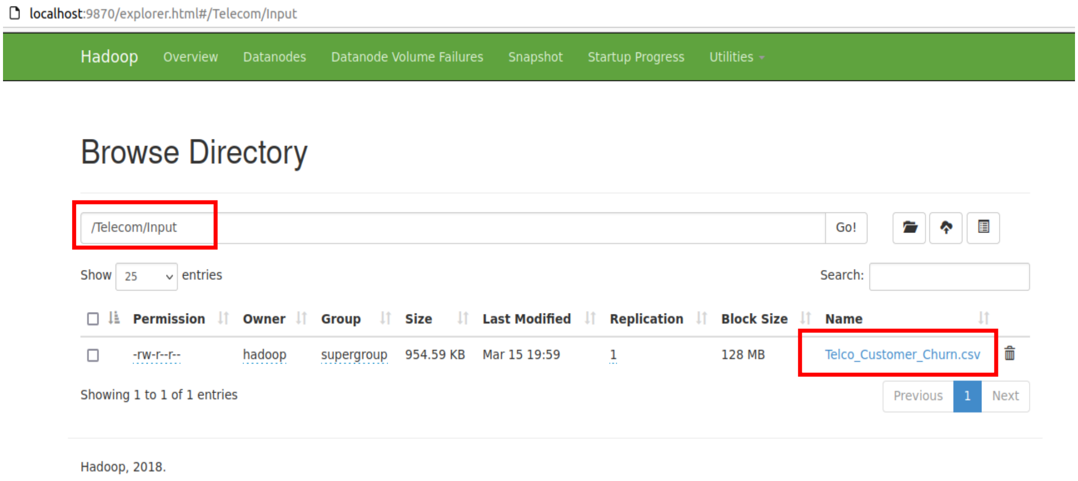
4. Copy and place the input file 'WA_Fn-UseC_-Telco- Customer-Churn.csv' in the 'Input' folder using the 'put' command as shown below.
Note: Here, I have renamed the original downloaded file from 'WA_Fn-UseC_-Telco-Customer-Churn' to 'Telco_Customer_Churn'.



```
(base) hadoop@hadoop-VirtualBox:~$ hdfs dfs -mkdir /Telecom
(base) hadoop@hadoop-VirtualBox:~$ hdfs dfs -mkdir /Telecom/Input
(base) hadoop@hadoop-VirtualBox:~$ hdfs dfs -put Telco_Customer_Churn.csv /Telecom/Input
(base) hadoop@hadoop-VirtualBox:~$ 
```

Figure 3

5. Now, check whether the input file 'Telco_Customer_Churn.csv' is placed inside the /Telecom/Input directory or not. To do that, login to the interface, by default, it is available at http://localhost:9870. Next, go to Utilities and browse the file system **'/Telecom/Input/'.** Notice that the file 'Telco_Customer_Churn' is inserted in **/Telecom/Input/** directory.



Browse Directory

/Telecom/Input

Show 25 entries

Search:

	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
□	-rw-r--r--	hadoop	supergroup	954.59 KB	Mar 15 19:59	1	128 MB	Telco_Customer_Churn.csv

Showing 1 to 1 of 1 entries

Hadoop, 2018.

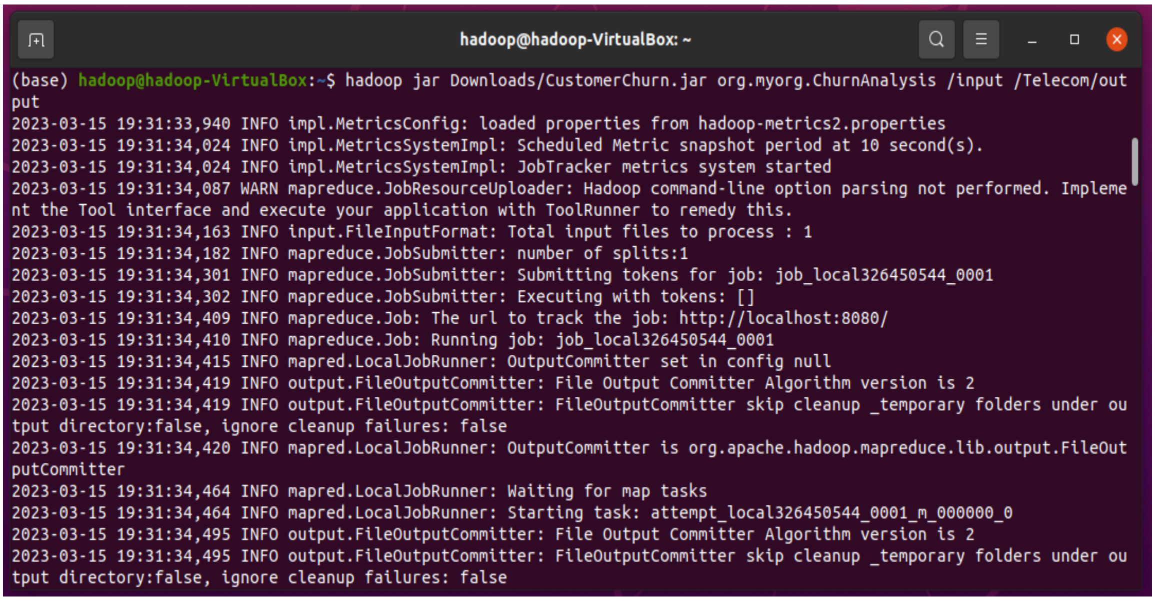
Figure 4

6. Now, In the Eclipse IDE, create the project and the required Java classes and export the .jar file. (Steps to create the java classes and exporting the .jar file mentioned in section 2.3 JAVA Classes Implementation and 2.4 Export the Jar file respectively)

7. Finally, run the exported jar file using the below command.

hadoop jar <jar file name> <driver class name> <input path> <output path>

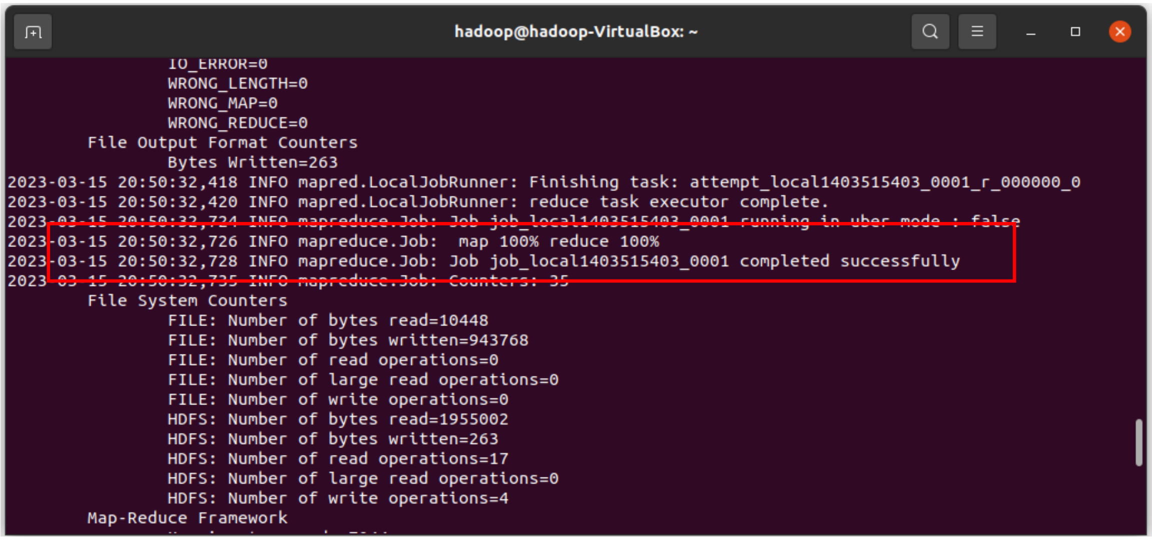
Ex:<hadoop jar Downloads/CustomerChurn.jar org.myorg.ChurnAnalysis /Telecom/Input /Telecom/Output>



```
(base) hadoop@hadoop-VirtualBox:~$ hadoop jar Downloads/CustomerChurn.jar org.myorg.ChurnAnalysis /input /Telecom/out
put
2023-03-15 19:31:33,940 INFO impl.MetricsConfig: loaded properties from hadoop-metrics2.properties
2023-03-15 19:31:34,024 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot period at 10 second(s).
2023-03-15 19:31:34,024 INFO impl.MetricsSystemImpl: JobTracker metrics system started
2023-03-15 19:31:34,087 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Impleme
nt the Tool interface and execute your application with ToolRunner to remedy this.
2023-03-15 19:31:34,163 INFO input.FileInputFormat: Total input files to process : 1
2023-03-15 19:31:34,182 INFO mapreduce.JobSubmitter: number of splits:1
2023-03-15 19:31:34,301 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local326450544_0001
2023-03-15 19:31:34,302 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-03-15 19:31:34,409 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
2023-03-15 19:31:34,410 INFO mapreduce.Job: Running job: job_local326450544_0001
2023-03-15 19:31:34,415 INFO mapred.LocalJobRunner: OutputCommitter set in config null
2023-03-15 19:31:34,419 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2023-03-15 19:31:34,419 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under ou
tput directory:false, ignore cleanup failures: false
2023-03-15 19:31:34,420 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapreduce.lib.output.FileOut
putCommitter
2023-03-15 19:31:34,464 INFO mapred.LocalJobRunner: Waiting for map tasks
2023-03-15 19:31:34,464 INFO mapred.LocalJobRunner: Starting task: attempt_local326450544_0001_m_000000_0
2023-03-15 19:31:34,495 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2023-03-15 19:31:34,495 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under ou
tput directory:false, ignore cleanup failures: false
```

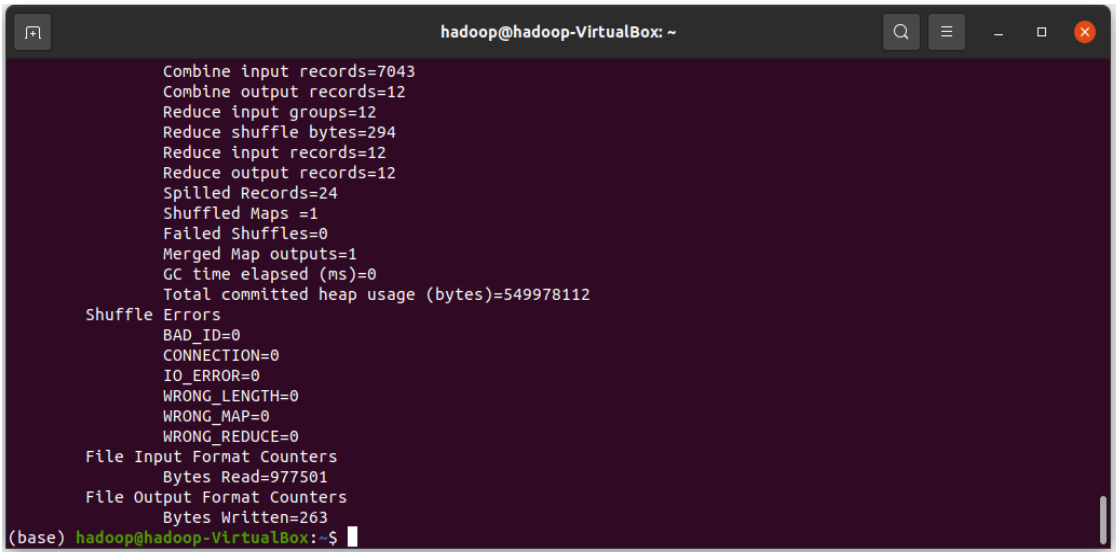
Figure 5

8. When the job runs successfully, no error messages will be displayed after the completion of MapReduce as shown below.



```
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Output Format Counters
                Bytes Written=263
2023-03-15 20:50:32,418 INFO mapred.LocalJobRunner: Finishing task: attempt_local1403515403_0001_r_000000_0
2023-03-15 20:50:32,420 INFO mapred.LocalJobRunner: reduce task executor complete.
2023-03-15 20:50:32,724 INFO mapreduce.Job: Job job_local1403515403_0001 running in uber mode : false
2023-03-15 20:50:32,726 INFO mapreduce.Job:  map 100% reduce 100%
2023-03-15 20:50:32,728 INFO mapreduce.Job: Job job_local1403515403_0001 completed successfully
2023-03-15 20:50:32,735 INFO mapreduce.Job: Counters: 35
        File System Counters
                FILE: Number of bytes read=10448
                FILE: Number of bytes written=943768
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=1955002
                HDFS: Number of bytes written=263
                HDFS: Number of read operations=17
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=4
        Map-Reduce Framework
```

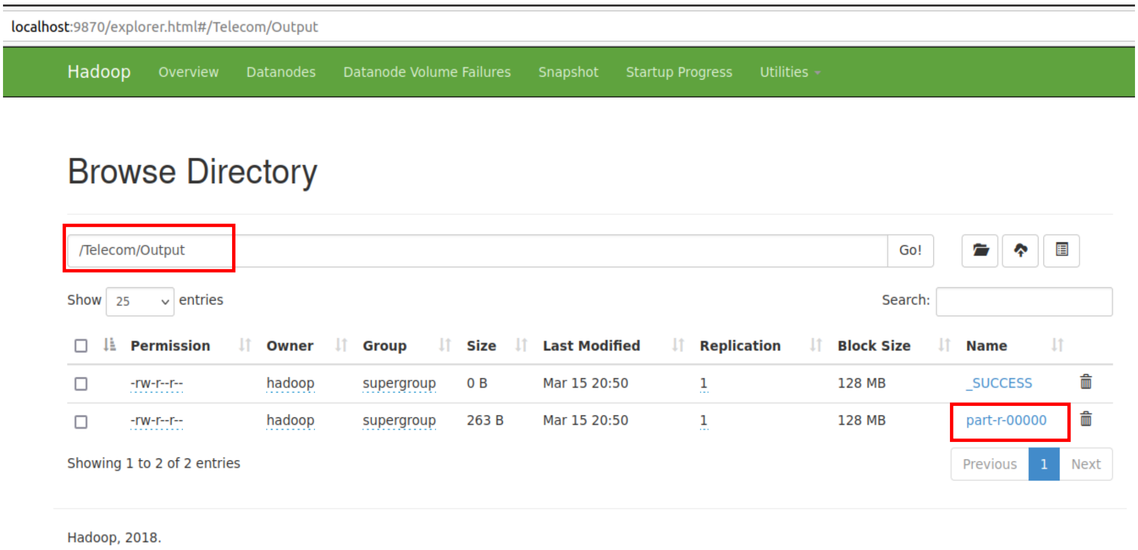
Figure 6



```
                Combine input records=7043
                Combine output records=12
                Reduce input groups=12
                Reduce shuffle bytes=294
                Reduce input records=12
                Reduce output records=12
                Spilled Records=24
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=0
                Total committed heap usage (bytes)=549978112
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=977501
        File Output Format Counters
                Bytes Written=263
(base) hadoop@hadoop-VirtualBox:~$
```

Figure 7

9. Check the output of the job which is created in the following location. /Telecom/Output/part-r-00000



Browse Directory

/Telecom/Output

Show 25 entries   Search:

☐	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
☐	-rw-r--r--	hadoop	supergroup	0 B	Mar 15 20:50	1	128 MB	_SUCCESS
☐	-rw-r--r--	hadoop	supergroup	263 B	Mar 15 20:50	1	128 MB	part-r-00000

Showing 1 to 2 of 2 entries   Previous 1 Next

Hadoop, 2018.

Figure 8

2.2 JAVA Classes Implementation

1. Create the project as shown in figure 9. Also, four additional jars were used in the development of this project.

a. hadoop-common.jar – available at
</usr/local/hadoop/share/hadoop/common>

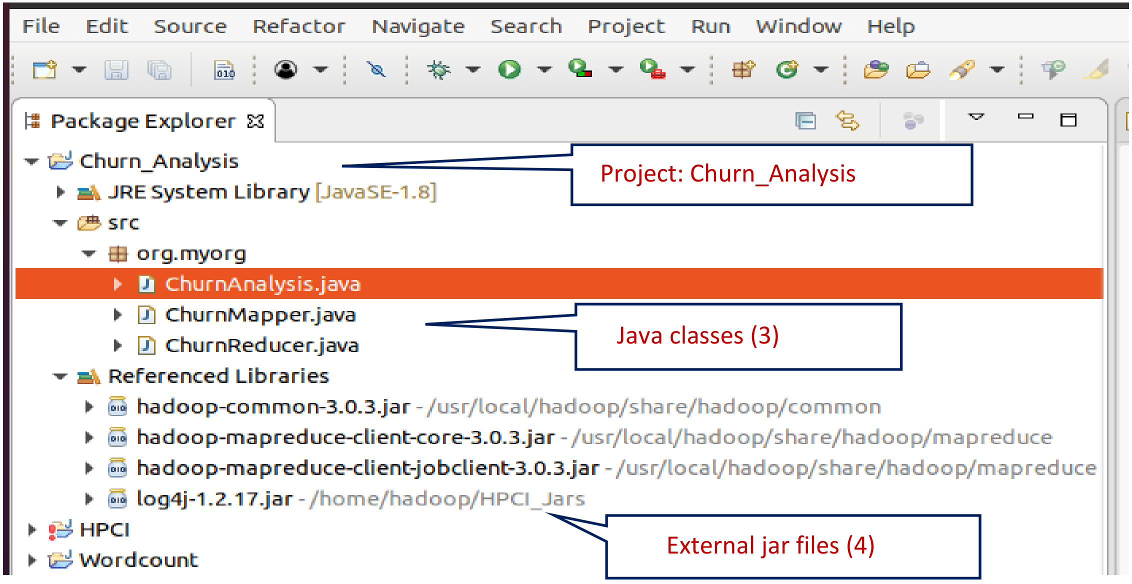
b. hadoop-mapreduct-client-core.jar – available at
</usr/local/hadoop/share/hadoop/mapreduce>

c. hadoop-mapreduct-client-jobclient.jar – available at
</usr/local/hadoop/share/hadoop/mapreduce>

d. log4j.jar – available at

```
</home/hadoop/HPCI_Jars >
```

2. To obtain the log4j jar file needed for the project, you can download it from the official Apache log4j website at https://logging.apache.org/log4j/1.2/download.html. A separate file named "Mapreduce_ChurnAnalysis" has been provided with clear instructions for downloading the jar file. Log4j is a Java logging framework developed by the Apache Software Foundation and is used to track and manage program logs.



```
Churn_Analysis
  JRE System Library [JavaSE-1.8]
  src
    org.myorg
      ChurnAnalysis.java
      ChurnMapper.java
      ChurnReducer.java
  Referenced Libraries
    hadoop-common-3.0.3.jar - /usr/local/hadoop/share/hadoop/common
    hadoop-mapreduce-client-core-3.0.3.jar - /usr/local/hadoop/share/hadoop/mapreduce
    hadoop-mapreduce-client-jobclient-3.0.3.jar - /usr/local/hadoop/share/hadoop/mapreduce
    log4j-1.2.17.jar - /home/hadoop/HPCI_Jars
HPCI
Wordcount
```

Project: Churn_Analysis

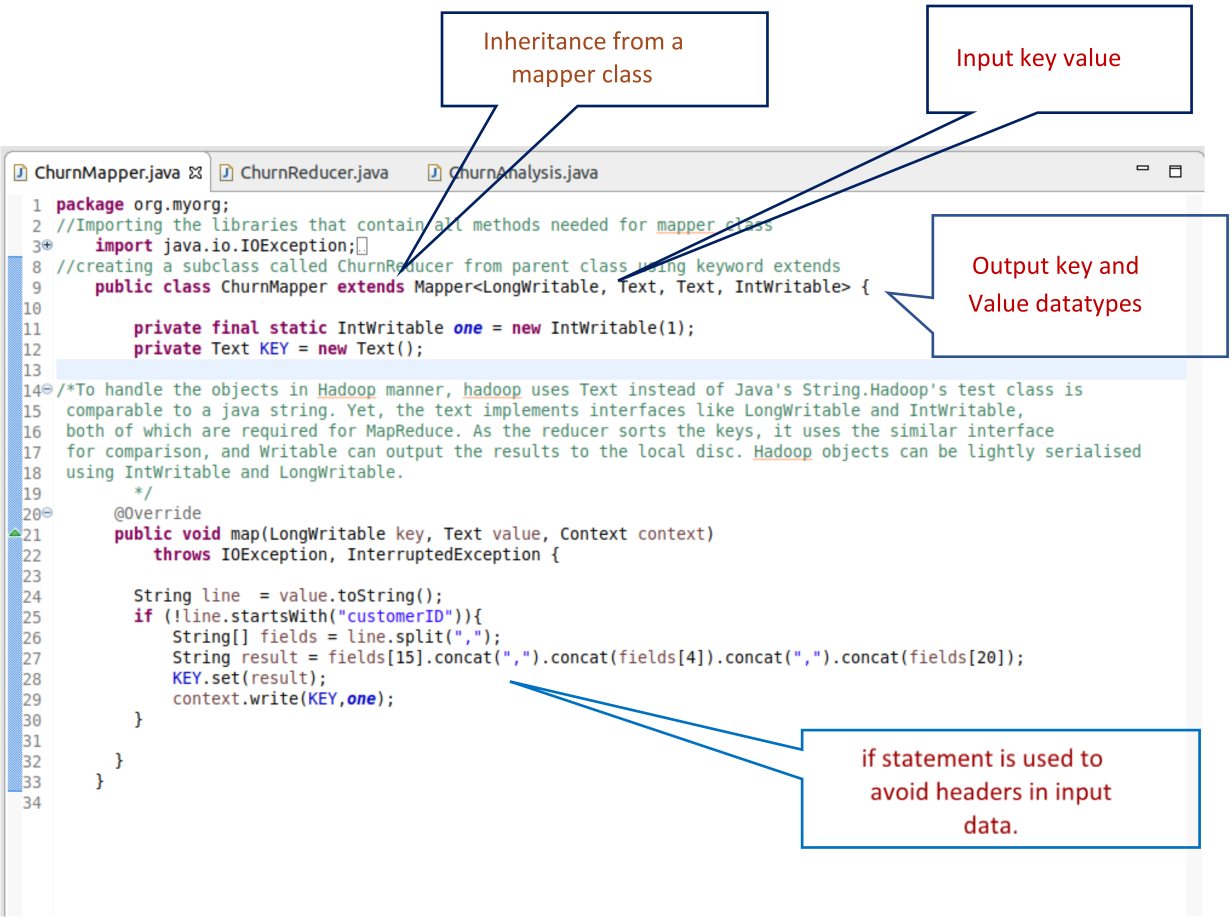
Java classes (3)

External jar files (4)

Figure 9

ChurnMapper (Mapper Class):

The Mapper class will take a line from the CSV file one by one as input. For each line, first, it will convert the line into a string and store it in a line variable. If the line is the header of data, then, it will avoid that line else it will split the line by comma to separate every field of a row. We need only contracts, dependents, and churn fields for this research question. So, these fields are accessed by using their respective indexes. The result of the mapper class consists of a key and a value. Mapper class will return contract, dependents and churn as the key in the form of Text and 1 as the value in the form of IntWritable. So, for each customer, we will get its contract, dependents, and churn as keys and 1 as values by using the mapper class.



Inheritance from a mapper class

Input key value

Output key and Value datatypes

if statement is used to avoid headers in input data.

```
package org.myorg;
//Importing the libraries that contain all methods needed for mapper class
    import java.io.IOException;
//creating a subclass called ChurnReducer from parent class using keyword extends
    public class ChurnMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

        private final static IntWritable one = new IntWritable(1);
        private Text KEY = new Text();

/*To handle the objects in Hadoop manner, hadoop uses Text instead of Java's String.Hadoop's test class is
 comparable to a java string. Yet, the text implements interfaces like LongWritable and IntWritable,
 both of which are required for MapReduce. As the reducer sorts the keys, it uses the similar interface
 for comparison, and Writable can output the results to the local disc. Hadoop objects can be lightly serialised
 using IntWritable and LongWritable.
        */
      @Override
      public void map(LongWritable key, Text value, Context context)
          throws IOException, InterruptedException {

        String line  = value.toString();
        if (!line.startsWith("customerID")){
            String[] fields = line.split(",");
            String result = fields[15].concat(",").concat(fields[4]).concat(",").concat(fields[20]);
            KEY.set(result);
            context.write(KEY,one);
        }

      }
    }
```

Figure 10

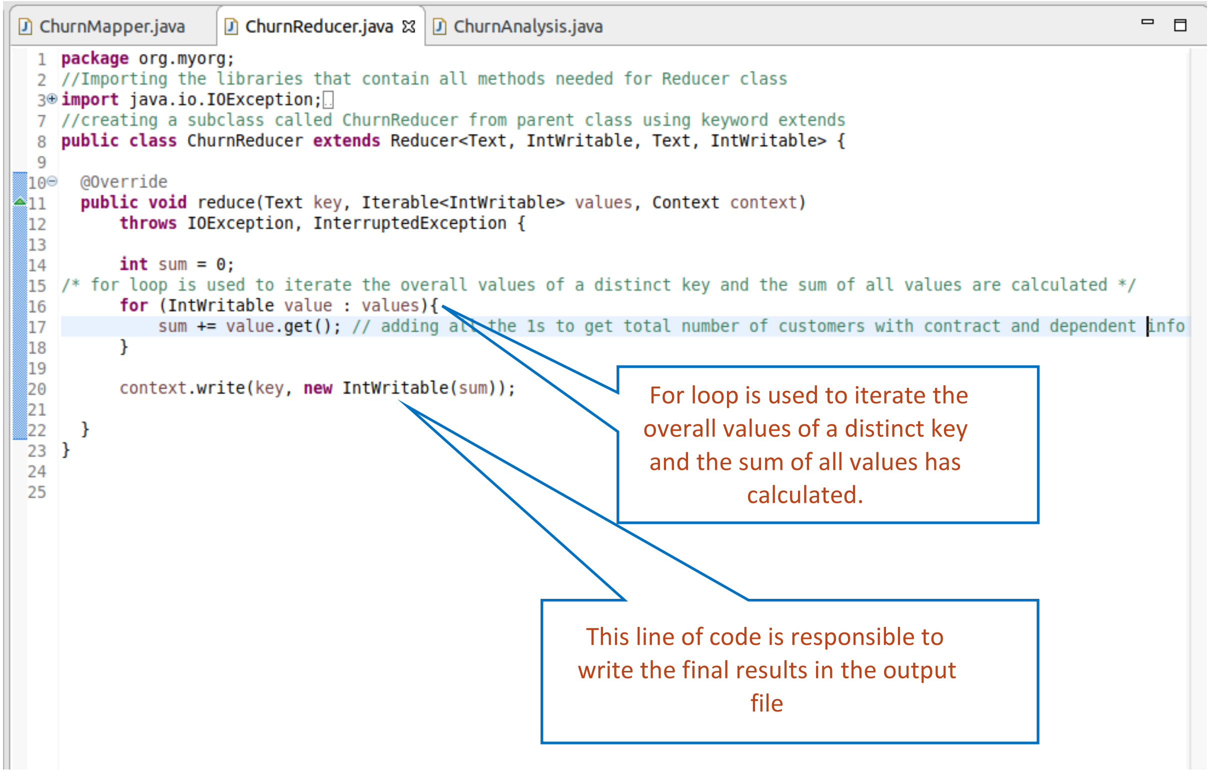
ChurnReducer (Reducer Class):

The Reducer class will take the results of the mapper class as input. It will group the values of the mapper class result by key and will aggregate those values. In this reducer class, the sum is applied as an aggregation function. For each key, it will take the 'sum' of all of its values. Reducer class will return contract, dependents, and churn as key in form of Text and 'sum of all values of that key' as the value in the form of IntWritable. So, the output would be in the following form:

Contract, dependent, churn, Sumofvalues

Example: Month-to-Month, No, Yes, 1396

In this example, 'Month-to-month' is a contract, 'No' is dependents, and 'Yes' is churn. Thus, 1396 is the total number of customers who have a Month-to-month contract, have 'No' dependents, and shave churned the subscription (which means the customer has unsubscribed/left the contract).



```java
package org.myorg;
//Importing the libraries that contain all methods needed for Reducer class
import java.io.IOException;
//creating a subclass called ChurnReducer from parent class using keyword extends
public class ChurnReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

  @Override
  public void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {

      int sum = 0;
/* for loop is used to iterate the overall values of a distinct key and the sum of all values are calculated */
      for (IntWritable value : values){
          sum += value.get(); // adding all the 1s to get total number of customers with contract and dependent info
      }

      context.write(key, new IntWritable(sum));

  }
}
```

For loop is used to iterate the overall values of a distinct key and the sum of all values has calculated.

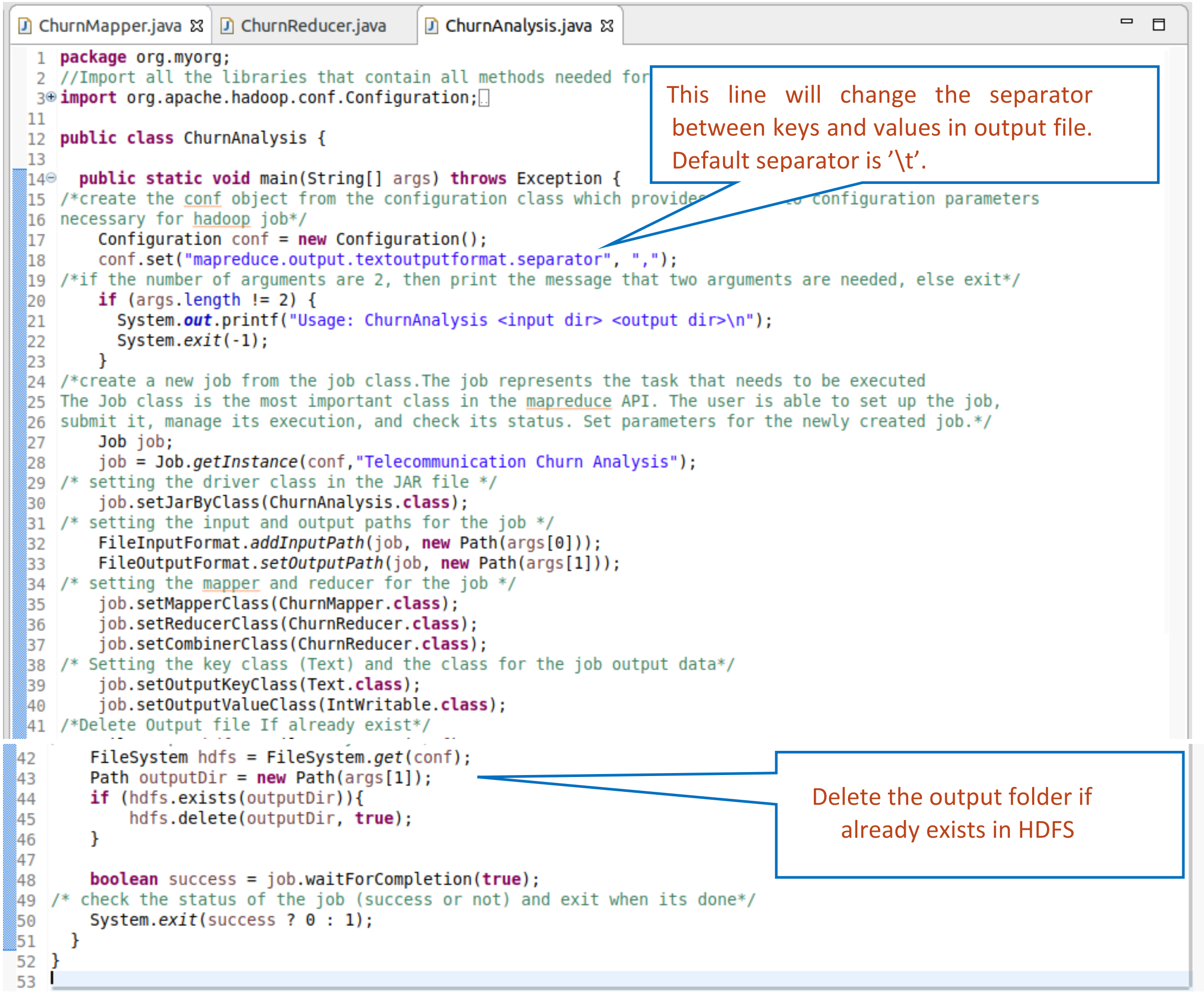
This line of code is responsible to write the final results in the output file

Figure 11

ChurnAnalysis (Driver Class):

This is the driver class of the program. It is implemented to run the whole project. Make sure both the input and output paths are passed in the command. It will create the job instance and is responsible for passing inputs to the mapper and reducer classes. It will delete the output folder from hdfs if that folder already exists. It returns a success message after the job gets completed successfully.

- **Job Name:** name of the job, job
- **Executable (Jar) Class:** the main executable class. Here it's, ChurnAnalysis.
- **Mapper Class:** class that overrides the "map" function. Here it's, ChurnMapper.
- **Reducer Class:** class that overrides the "reduce" function. Here it's, ChurnReducer.
- **Output Key:** type of output key. It's, Text.
- **Output Value:** type of output value. It's IntWritable.
- **File Input Path:** /Telecom/Input
- **File Output Path:** /Telecom/Output



```java
package org.myorg;
//Import all the libraries that contain all methods needed for
import org.apache.hadoop.conf.Configuration;

public class ChurnAnalysis {

  public static void main(String[] args) throws Exception {
/*create the conf object from the configuration class which provide        configuration parameters
necessary for hadoop job*/
    Configuration conf = new Configuration();
    conf.set("mapreduce.output.textoutputformat.separator", ",");
/*if the number of arguments are 2, then print the message that two arguments are needed, else exit*/
    if (args.length != 2) {
      System.out.printf("Usage: ChurnAnalysis <input dir> <output dir>\n");
      System.exit(-1);
    }
/*create a new job from the job class.The job represents the task that needs to be executed
The Job class is the most important class in the mapreduce API. The user is able to set up the job,
submit it, manage its execution, and check its status. Set parameters for the newly created job.*/
    Job job;
    job = Job.getInstance(conf,"Telecommunication Churn Analysis");
/* setting the driver class in the JAR file */
    job.setJarByClass(ChurnAnalysis.class);
/* setting the input and output paths for the job */
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
/* setting the mapper and reducer for the job */
    job.setMapperClass(ChurnMapper.class);
    job.setReducerClass(ChurnReducer.class);
    job.setCombinerClass(ChurnReducer.class);
/* Setting the key class (Text) and the class for the job output data*/
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
/*Delete Output file If already exist*/
    FileSystem hdfs = FileSystem.get(conf);
    Path outputDir = new Path(args[1]);
    if (hdfs.exists(outputDir)){
        hdfs.delete(outputDir, true);
    }

    boolean success = job.waitForCompletion(true);
/* check the status of the job (success or not) and exit when its done*/
    System.exit(success ? 0 : 1);
  }
}
```

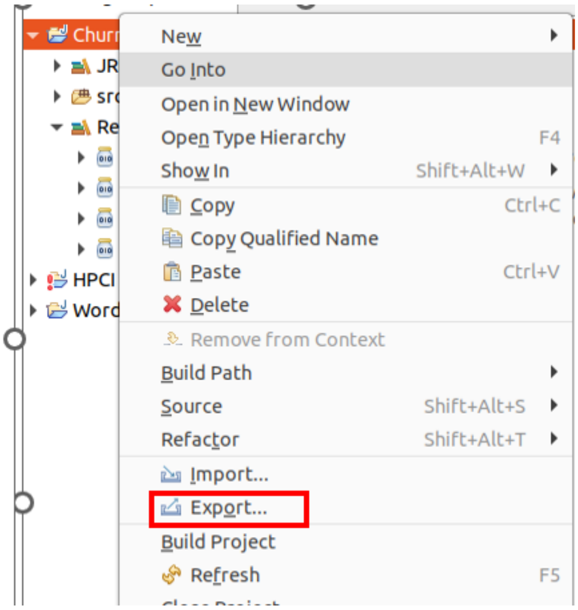
This line will change the separator between keys and values in output file. Default separator is '\t'.

Delete the output folder if already exists in HDFS

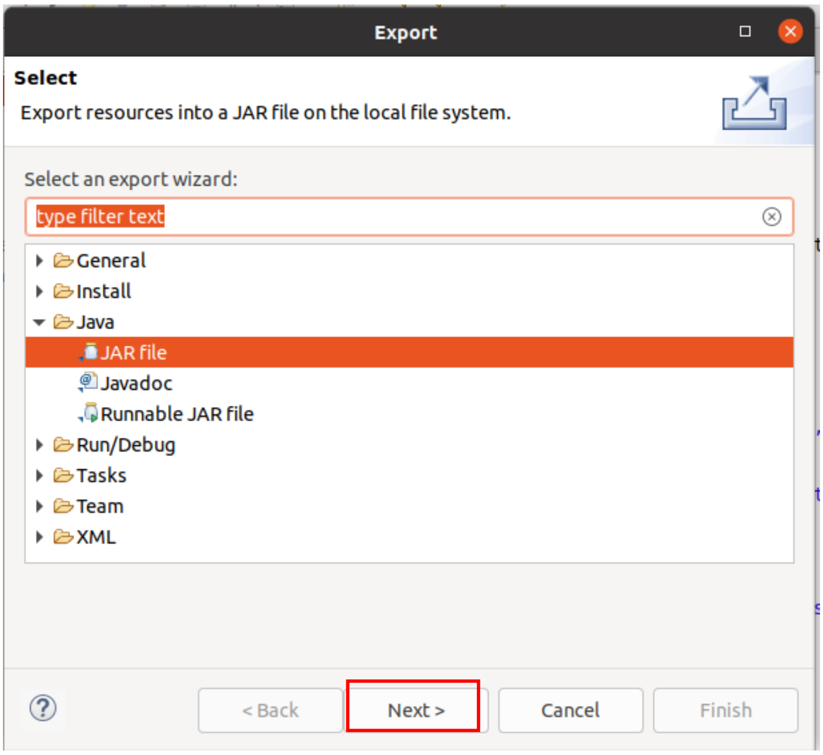
Figure 12

2.3 Export the Jar file

1. Right-click on the project (Churn_Analysis) and click on Export.



2. Click on the Next button



Export

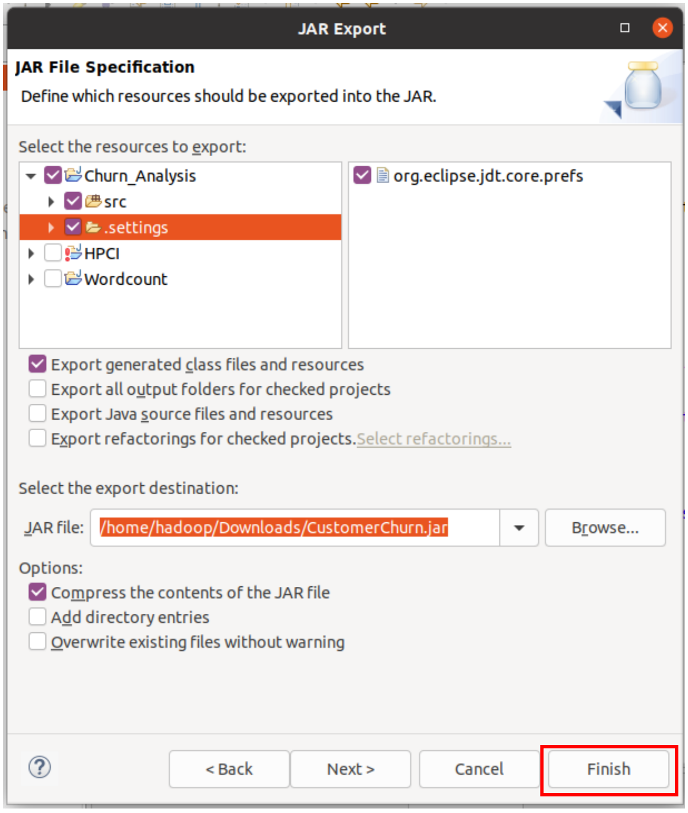
Select

Export resources into a JAR file on the local file system.

Select an export wizard:

type filter text

3. Select the resources to export (Churn_Analysis) and select the export destination where the jar file needs to be saved in your system. Hit the next button.



JAR Export

JAR File Specification

Define which resources should be exported into the JAR.

Select the resources to export:

- ☑ Churn_Analysis
  - ☑ src
  - ☑ .settings
- □ HPCI
- □ Wordcount

☑ org.eclipse.jdt.core.prefs

☑ Export generated class files and resources
□ Export all output folders for checked projects
□ Export Java source files and resources
□ Export refactorings for checked projects. Select refactorings...

Select the export destination:

JAR file: /home/hadoop/Downloads/CustomerChurn.jar

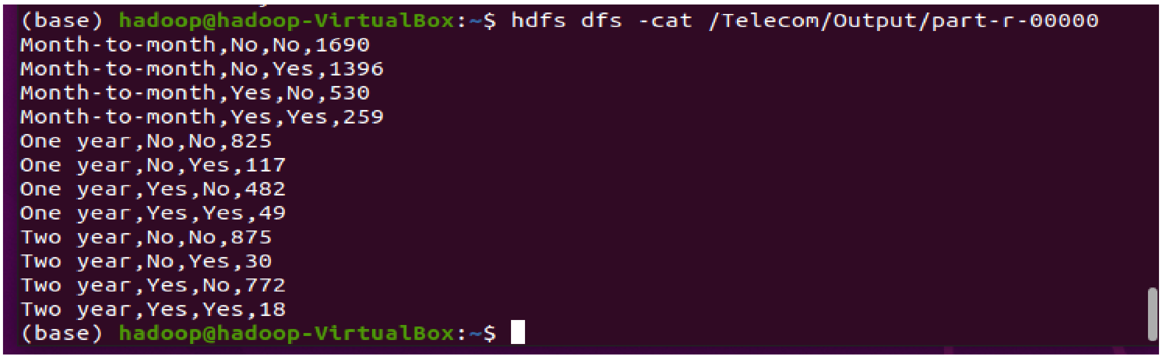
Options:

☑ Compress the contents of the JAR file
□ Add directory entries
□ Overwrite existing files without warning

4. Select the main class and click on finish. It will export the jar in the given destination folder.

3. **Results & Evaluation**

1. The obtained output file ‘part-r-00000’ is in the format shown below.

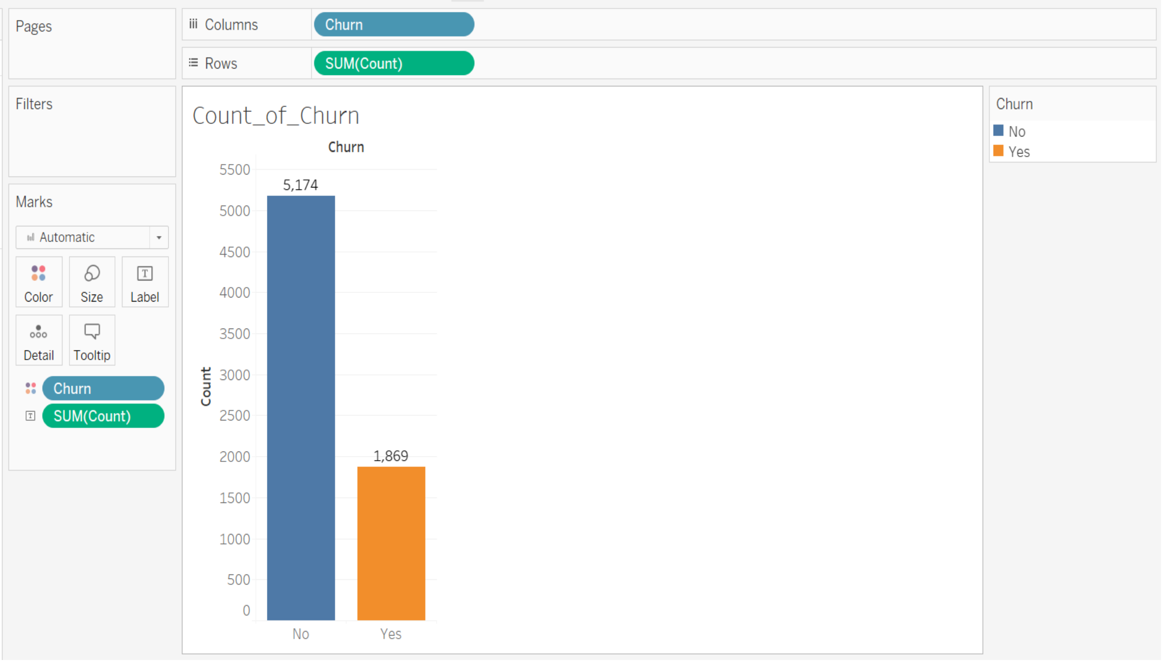


```
(base) hadoop@hadoop-VirtualBox:~$ hdfs dfs -cat /Telecom/Output/part-r-00000
Month-to-month,No,No,1690
Month-to-month,No,Yes,1396
Month-to-month,Yes,No,530
Month-to-month,Yes,Yes,259
One year,No,No,825
One year,No,Yes,117
One year,Yes,No,482
One year,Yes,Yes,49
Two year,No,No,875
Two year,No,Yes,30
Two year,Yes,No,772
Two year,Yes,Yes,18
(base) hadoop@hadoop-VirtualBox:~$ 
```

2. The output file has been converted to CSV and is imported into the tableau tool to visualize and identify the customer churned vary from contract period and people having dependents.

4. **Observations:**

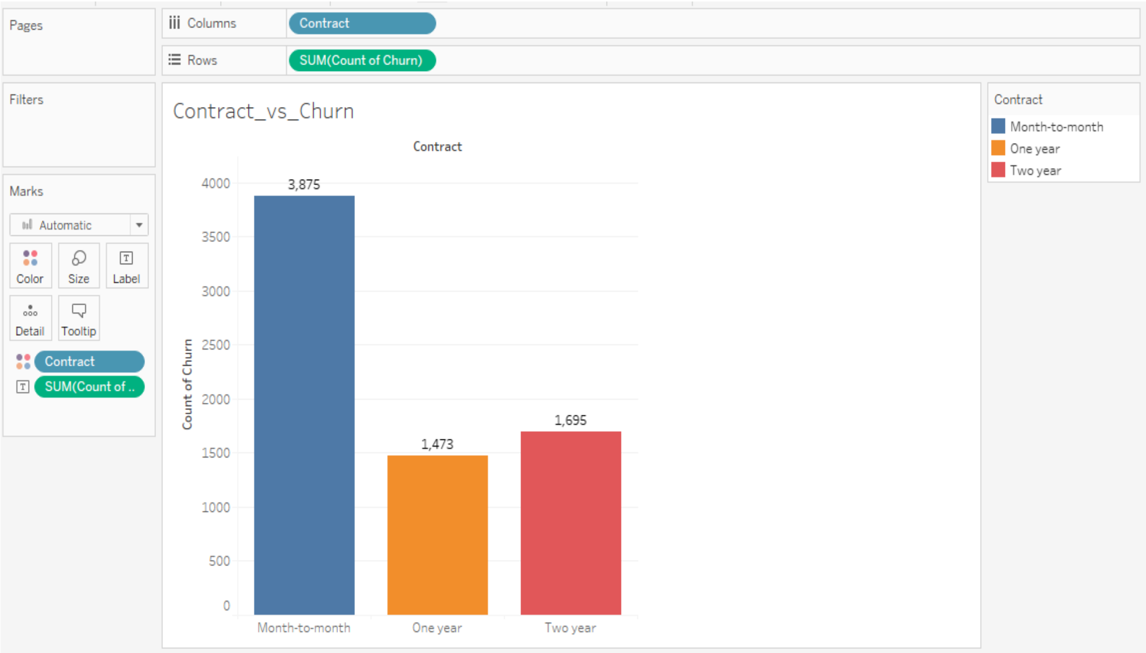
a. It is observed that the company has lost around 25% of subscribers as shown in the plot below.



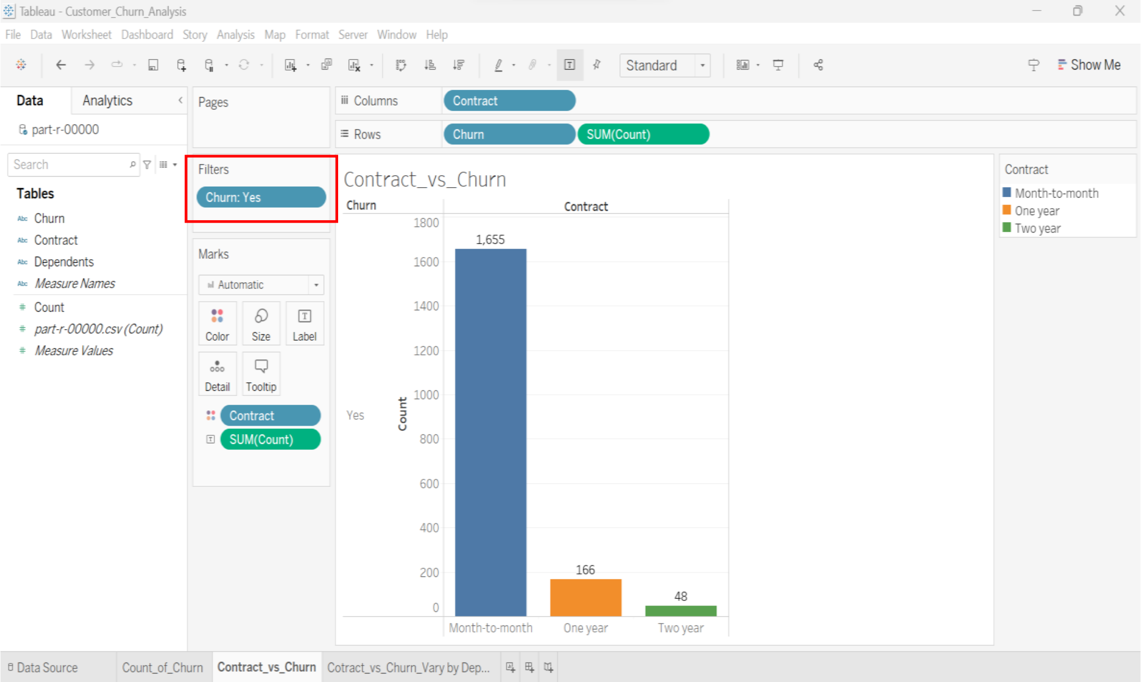
Columns: Churn

Rows: SUM(Count)

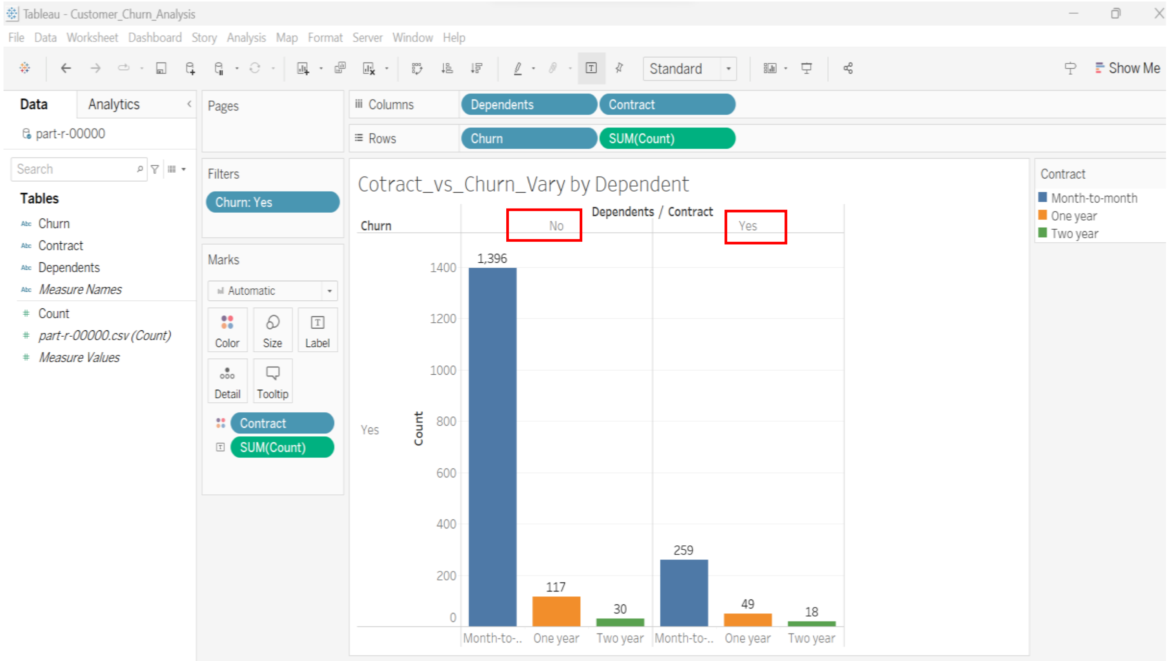
b. It is observed that most of the subscribers like to choose month-to-month contracts as shown in the plot below.



c. It is observed that customers having month-to-month contract periods are more likely to churn compared to customers having one-year or two-year contract periods.



d. It is observed that the customers who do not have dependents and have subscribed for less contract period (month-to-month) are more likely to churn compared to the others as shown in the plot below.



Conclusion

With these observations, our research question is answered. It can be concluded that customers who subscribe for a month-to-month subscription, having no dependents, tend to cancel their subscription more frequently than those who opt for longer contract periods and have dependents. This difference is significant enough to affect the overall conclusion. Additionally, telecom companies can develop multiple strategies to improve customer retention and reduce churn. For example, companies can focus on offering long-term contract periods to customers, especially those without dependents. It not only develops their own business but also improves overall customer satisfaction.

Appendix

As part of the coursework, The Mapreduce_ChurnAnalysis Word document has been attached containing all supporting documents as an appendix to the coursework. It includes all the contents, such as the java classes code, the original dataset used for this project, the renamed dataset, the Log4j file, the output file, and a visualization created in Tableau (.twb file) to evaluate the results. This comprehensive list of contents in a single word document is intended to support and enhance the project's overall quality, allowing for a thorough analysis and evaluation of the project's findings.

References

hadoop.apache.org. (n.d.). Apache Hadoop 3.3.1 – Hadoop: Setting up a Single Node Cluster. [online] Available at: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html.

hadoop.apache.org. (n.d.). MapReduce Tutorial. [online] Available at: https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html.

Apache.org. (2015). Apache log4j 1.2 - Download Apache log4j 1.2. [online] Available at: https://logging.apache.org/log4j/1.2/download.html [Accessed 17 Mar. 2023].

help.tableau.com. (n.d.). Aggregate Functions in Tableau. [online] Available at:

https://help.tableau.com/current/pro/desktop/enus/calculations_calculatedfields_aggregate_create.htm [Accessed on 09 Mar, 2023].

logging.apache.org. (n.d.). LogManager (Apache Log4j API). [online] Available at: https://logging.apache.org/log4j/2.x/log4j-api/apidocs/org/apache/logging/log4j/LogManager.html [Accessed on 08 Mar, 2023].

Apache.org. (2015). Apache log4j 1.2 - Download Apache log4j 1.2. [online] Available at: https://logging.apache.org/log4j/1.2/download.html.

www.tutorialspoint.com. (n.d.). Eclipse - Create Jar Files. [online] Available at: https://www.tutorialspoint.com/eclipse/eclipse_create_jar_files.htm# [Accessed 17 Apr. 2023].

GeeksforGeeks. (2019). HDFS Commands. [online] Available at:
https://www.geeksforgeeks.org/hdfs-commands/ [Accessed on 08 Mar, 2023].