**BIG DATA ANALYSIS ON**

**CENSUS DATA**

**USING HADOOP**

**Presented by**

Vikas. M

NIIT Bejai (Mangalore)

**INDEX**

Contents

[1. INTRODUCTION 4](#_Toc484449306)

[**1.1 *What is Big data***? 4](#_Toc484449307)

[1.1.1 Characteristics of Big Data 4](#_Toc484449308)

[1.1.2 Big Data Analytics 5](#_Toc484449309)

[1.2 History of Hadoop 5](#_Toc484449310)

[1.3 Apache Hadoop 6](#_Toc484449311)

[1.3.2 Hadoop Distributed File System(HDFS) 6](#_Toc484449312)

[1.3.3 MapReduce 7](#_Toc484449313)

[1.5 HIVE 8](#_Toc484449314)

[1.6 PIG 8](#_Toc484449315)

[1.7 SQOOP 8](#_Toc484449316)

[1.8 FLUME 9](#_Toc484449317)

[1.9 OOZIE 9](#_Toc484449318)

[1.10 ZOOKEEPER 9](#_Toc484449319)

[2. PROJECT OUTLINE 10](#_Toc484449320)

[3. PROJECT OBJECTIVE 12](#_Toc484449321)

[4. PROJECT IMPLEMENTATION 13](#_Toc484449322)

[4.1 Education Analysis 13](#_Toc484449323)

[4.1.1 Education category-wise gender-wise count 13](#_Toc484449324)

[4.1.2. Education Qualification Count based on Employment 19](#_Toc484449325)

[4.1.3 Calculate Sex Ratio (Male : Female) 20](#_Toc484449326)

[4.2 Financial Analysis 22](#_Toc484449327)

[4.2.1 Total Income of different types of Tax Payers 22](#_Toc484449328)

[4.2.2. Gender wise Total Income Generated 24](#_Toc484449329)

[4.2.3 Total Income of different types of Tax Payers 25](#_Toc484449330)

[4.2.4 Total Tax To Be collected 30](#_Toc484449331)

[4.2.5 Per Capita Income 34](#_Toc484449332)

[4.2.6. Gender wise Per Capita Income 34](#_Toc484449333)

[4.3 Social Analysis 35](#_Toc484449334)

[4.3. 1 Pension Amount to be added after x years 35](#_Toc484449335)

[4.3.2 No. of Orphans for each category based on Parents Present 36](#_Toc484449336)

[4.3.3 No. of Employable Female Citizens who are Widows or Divorced 37](#_Toc484449337)

[4.4 Planning Analysis 38](#_Toc484449338)

[4.3.1 No. of Voters to get added in next X years 38](#_Toc484449339)

[4.4.2 No. of Senior Citizen to get added in next X years 42](#_Toc484449340)

[4.4.3 Citizen vs. Immigrants Ratio for all Employed 47](#_Toc484449341)

# INTRODUCTION

## **1.1 *What is Big data***?

Big data is an all-encompassing term for any collection of data sets so large and complex that it becomes difficult to process them using traditional data processing applications.

A definition of Big Data could be "Big Data is not about size, but it's about granularity". The ability of software systems to identify individuals and personalized data is the ironic implication of "Big". It is the ability to focus on the minutiae of the individual, in real time.

Extremely large data sets that may be analyzed computationally to reveal patterns, trends, and associations, especially relating to human behavior and interactions.

### Characteristics of Big Data

1. **Volume**(scale of data) - the data which is generated in today’s modern era is in very large size i.e., ~ petabytes of data per day which should be handled, stored, processed and used with utmost concern.

2. **Velocity**(analytics of streaming of data) – velocity means the rate of occurrence or the rapidity of motion. In this data world, velocity means the rate at which the data is getting generated or accumulated in the databases.

3. **Variety**(different forms of data) – data does not have any single format or fixed structure or schema, it can be any one of them like images, audio files, videos, log files, xml, excel files, tables etc. The data is categorized into three forms structured, semi-structured, unstructured.

4. **Veracity**(uncertainty of data) – As the data is collected from different sources, it must be trustworthy. Veracity means trustworthiness of the data.

5. **Value**(what we get are researching) – The data or the value which we get after diving deep into the data and performing various operations like storing, analyzing, processing, and generating the output. The value which we get after researching should be useful in real-time systems or for the development of any business organization.



### 1.1.2 Big Data Analytics

It is the process of examining large data sets containing a variety of data types- i.e., big data – to uncover hidden patterns, unknown correlations, market trends, customer preferences and other useful business information.

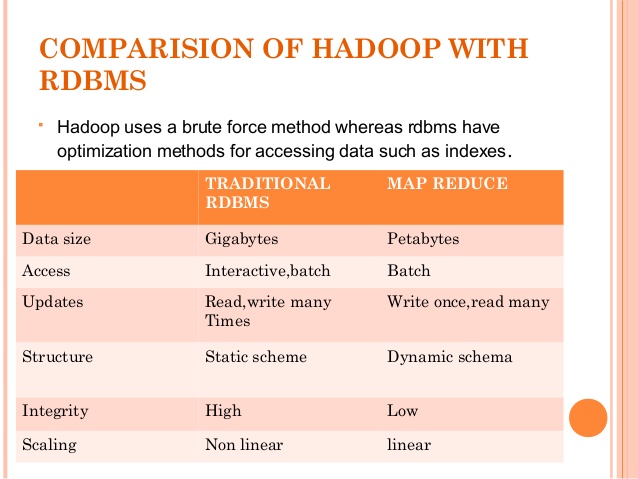
## 1.2 History of Hadoop

We all know about the Google search engine which processes the request of millions of users daily. For executing and displaying the result of queries of those many users daily would be a heavy task. In 1990, problem got raised for storing data as it have to store a huge amount of data to satisfy the needs of millions of users from all over the world. It took 13years to find a solution to that problem as it can’t be solved by using traditional databases. In 2003, GFS(Google File System) and in 2004, MapReduce was introduced by Google for storing and processing huge quantities of data. This solution was a discovery not an implementation method. It is a theoretical paper which describes how to solve that problem and what should be the measures to be taken. The inventor and founder of hadoop is **Doug Cutting**. During the project, everyone was thinking about the logo for the new discovery. One fine day Doug observed his son playing with an elephant toy to which he gave the name as hadoop. At that moment he decided to name their inventory as hadoop.

## 1.3 Apache Hadoop

**Hadoop** is an open-source framework given by Apache Software Foundation for storing huge datasets and processing huge datasets with cluster of commodity hardware. It has two main components namely hadoop distributed filesystem and mapreduce.

***1.3.1 HADOOP VS RDBMS***



### 1.3.2 Hadoop Distributed File System(HDFS)

Hadoop distributed file system is a specially designed file-system for storing large datasets with cluster of commodity hardware and with streaming access pattern. Here cluster is a group of data nodes or network of machines or processors where each of them having their own primary storage(ex-RAM) and secondary storage(ex-harddisk). The hardware used in hadoop file system will be at low cost i.e., we don’t require costly machines for storing and processing our data. Hadoop need not to be worked on highly reliable hardware. It can be worked on simple cheap hardware.

### 1.3.3 MapReduce

Apache MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. MapReduce idea is inspired from similar paradigm used by functional programming languages like LISP. Their programs are easy to run in distributed environment.

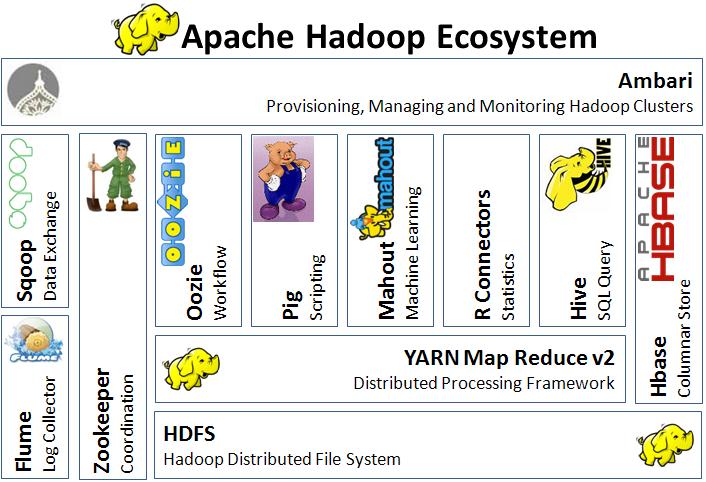
In 2003, Google has introduced the MapReduce concept but later in December 2004, it published a paper explaining architecture of MapReduce.

A MapReduce  job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster.

MapReduce has two core functions: Mapper in which filteration and extraction of data is done in key, value pair and passed to the lowest logical level and Reducer where the mapper output is taken as input in the key, value pair and the aggregate operation is performed. The actual operation and business logic is written in reducer class.

***1.4 Hadoop Ecosystem***



HDFS- framework for storing our files

Mapreduce- framework used for processing the files stored in our HDFS

Flume, Sqoop- tools helpful to import/export the data(data loading tools)

Zookeeper- tool used for synchronizing the various services present in the hadoop framework

Oozie- tool used to keep the track of the workflow

Hive, Pig- frameworks used for processing the stored data

Mahout- framework provides an environment for creating machine learning applications

R Connectors- tool used for predictive analysis

HBase- nosql database

## 1.5 HIVE

Hive is a data warehouse system layer built on Hadoop which allows you to define a structure for your big data and simplifies analysis and queries with an SQL-like scripting language called HiveQL.

Hive uses a database to store metadata, but the data that hive process is stored in hdfs. Simply hive is nothing but which allows to run structured query on unstructured data.

## 1.6 PIG

Pig is an in memory processing tool and also said to be data flow language which supports all data formats. In pig we have grunt shell and by default data type will be byte array. Pig provides a high-level language known as **Pig Latin**.

In Apache Pig, **schema** is optional. We can store data without designing a schema (values are stored as $1, $2 etc.)

## 1.7 SQOOP

Sqoop is a tool designed to transfer data between Hadoop and relational databases or mainframes. You can use Sqoop to import data from a relational database management system (RDBMS) such as MySQL or Oracle or a mainframe into the Hadoop Distributed File System (HDFS), transform the data in Hadoop MapReduce, and then export the data back into an RDBMS.

## 1.8 FLUME

Apache Flume is a distributed, reliable, and available system for efficiently collecting, aggregating and moving large amounts of log data from many different sources to a centralized data store. The use of Apache Flume is not only restricted to log data aggregation. Since data sources are customizable, Flume can be used to transport massive quantities of event data including but not limited to network traffic data, social-media-generated data, email messages and pretty much any data source possible. Apache Flume is a top level project at the Apache Software Foundation.

## 1.9 OOZIE

Oozie is a server based *Workflow Engine* specialized in running workflow jobs with actions that run Hadoop Map/Reduce and Pig jobs. For the purposes of Oozie, a workflow is a collection of actions (i.e. Hadoop Map/Reduce jobs, Pig jobs) arranged in a control dependency DAG (Direct Acyclic Graph) "control dependency" from one action to another means that the second action can't run until the first action has completed. Oozie workflows contain control flow nodes and action nodes.

## 1.10 ZOOKEEPER

ZooKeeper is a distributed co-ordination service to manage large set of hosts. Co-ordinating and managing a service in a distributed environment is a complicated process. ZooKeeper solves this issue with its simple architecture and API. ZooKeeper allows developers to focus on core application logic without worrying about the distributed nature of the application.

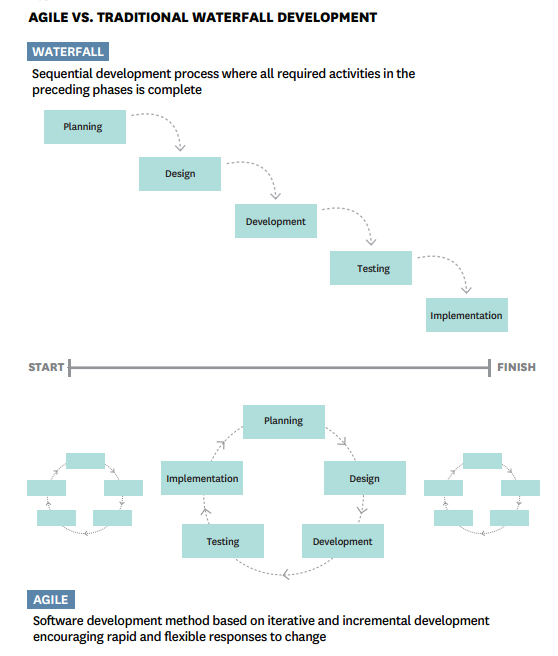
# PROJECT OUTLINE

|  |  |
| --- | --- |
| **Title** | **BigData Analysis in Hadoop on Census Data**. |
| **Inputs** | Census Data |
| **Data Elements** | Age, Education, Marital Status, Gender, TaxFiler Status, Income, Parents, Country of Birth, Citizenship, WeeksWorked |
| **Analysis Relevance** | Education, Social, Finance and Planning |
| **Purpose** | To provide Analyzed Results to Government to help it plan policies to improve social and economic life of citizens of the country. |
| **Methodology** | Agile |

The dataset for this project is the census data. This is a sample data of individuals residing in a country. The dataset consists of 2000 records and is divided into ten columns namely Age, Education, MaritalStatus, Gender, TaxFilerStatus, Income, Parents, CountryOfBirth, Citizenship and WeeksWorked.

This project is implemented using Hadoop. Tools like MapReduce (using Java), Hive and Pig are used to generate the results. HDFS (Hadoop distributed File System) is used for storing the census data. Sqoop and Flume are used for importing/exporting the data into/from the HDFS.

The methodology used in my project is agile software model. Agile is a software development methodology to build a software incrementally using short iterations of 1 to 4 weeks so that the development is aligned with the changing business needs.



# PROJECT OBJECTIVE

The objective is process the data with the help of hadoop framework and demonstrate my capabilities in using Hadoop constituents like Hive, Pig and MapReduce.

# PROJECT IMPLEMENTATION

On analyzing the columns given in the dataset, we can categorize the data into four modules namely:

1. Education analysis

2. Social analysis

3. Financial analysis

4. Planning analysis

Firstly, in education module, all the data related to the education field will fall like the literacy rate etc. In social welfare the issues related to the public and their well-being will be considered. In financial, economic related like income, tax related data and in planning module, plans and measures for the forth-coming issues will be included.

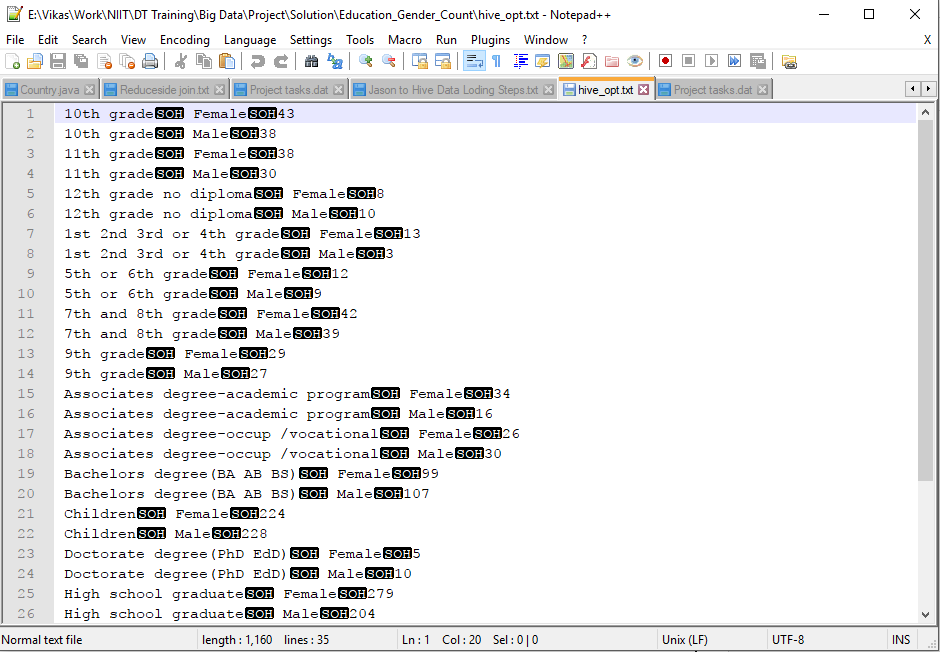
After analyzing this data some of the problem statements can be stated according to the divided modules:

## Education Analysis

### 4.1.1 Education category-wise gender-wise count

|  |
| --- |
| Hive Solution: select education, gender, count(\*) from census group by education, gender; |

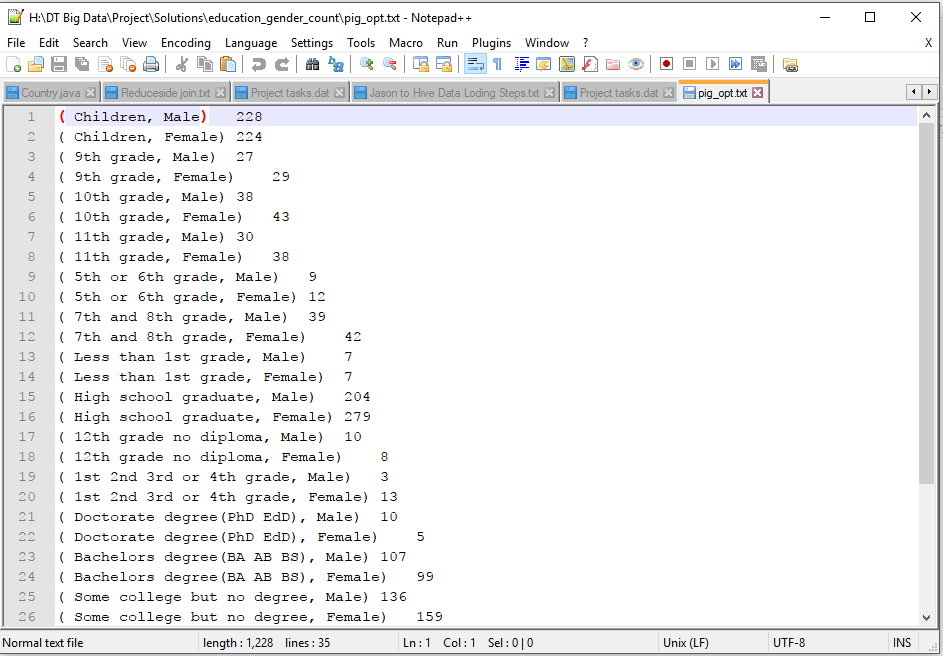
Output:



Pig Solution:

|  |
| --- |
| bag1 = load '/home/hduser/census.txt' using PigStorage(',') as (age:int, education:chararray, maritalstatus:chararray, gender:chararray, taxfilerstatus:chararray, income:float, parents:chararray, countryofbirth:chararray, citizenship:chararray, weeksworked:int);  bag2 = foreach bag1 generate education, gender;  bag3 = group bag2 by (education, gender);  bag4 = foreach bag3 generate group, COUNT(bag2.education);  store bag4 into 'education\_gender\_count' |

Output:



Map Reduce (Java) Solution:

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.Mapper.Context;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class EductionGenderCount

{

public static class MyMapper extends Mapper<LongWritable, Text, Text, IntWritable>

{

public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException

{

String[] arr = value.toString().split(",");

context.write(new Text(arr[1]+":" + arr[3]), new IntWritable(1));

}

}

public static class MyReducer extends Reducer<Text, IntWritable, Text, IntWritable>

{

public void reduce(Text key, Iterable<IntWritable> value, Context context) throws IOException, InterruptedException

{

int sum = 0;

for(IntWritable val:value)

{

sum += val.get();

}

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

}

}

/\*\*

\* @param args

\* @throws IOException

\* @throws InterruptedException

\* @throws ClassNotFoundException

\*/

public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException

{

// TODO Auto-generated method stub

Configuration cfg = new Configuration();

Job job = Job.getInstance(cfg, "EducationGenderCount");

job.setJarByClass(EductionGenderCount.class);

job.setMapperClass(MyMapper.class);

job.setReducerClass(MyReducer.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(job, new Path(args[0]));

FileSystem.get(cfg).delete(new Path(args[1]), true);

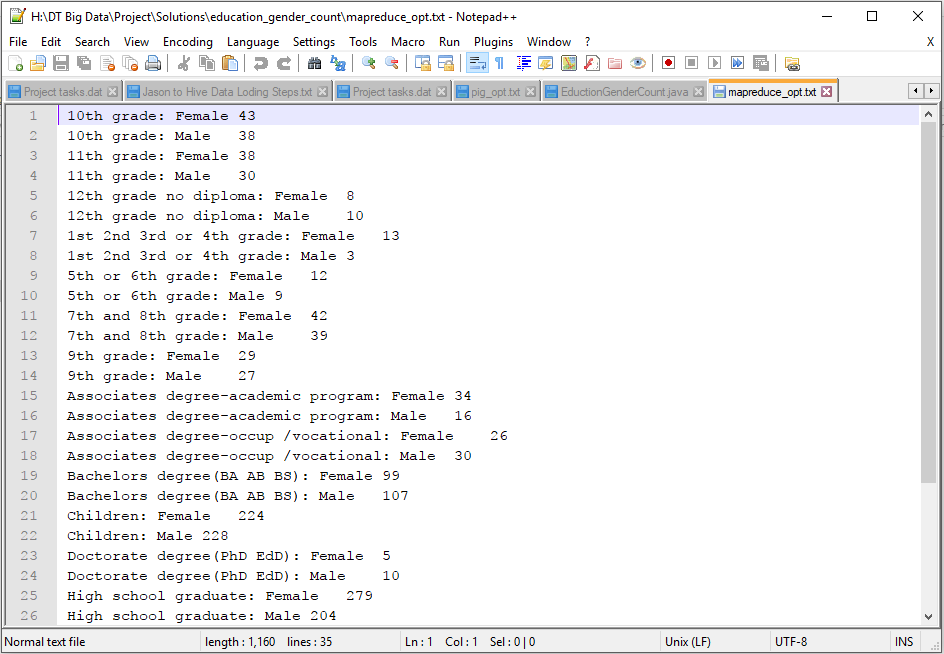
FileOutputFormat.setOutputPath(job, new Path(args[1]));

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

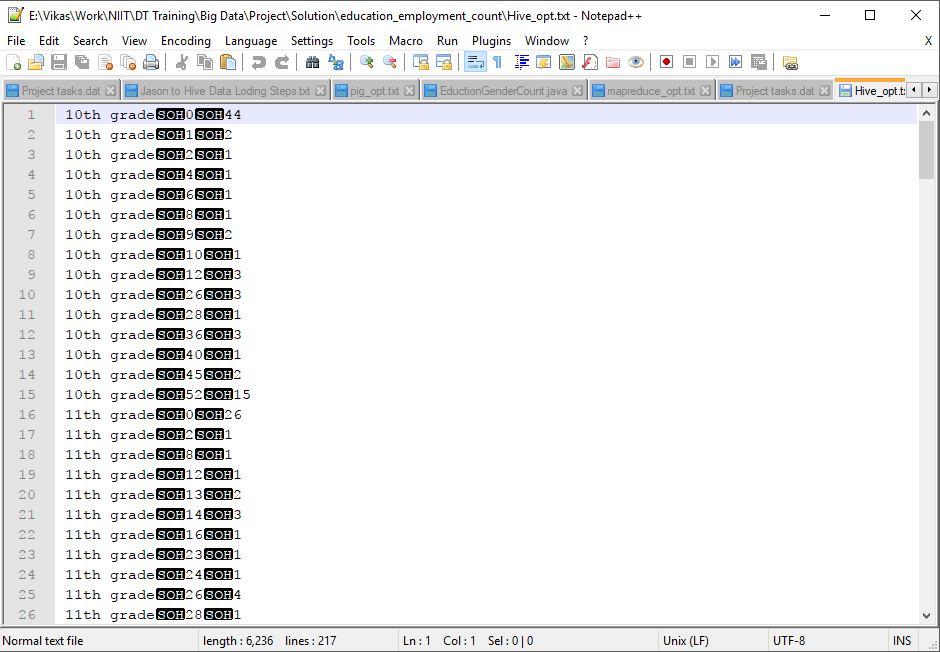
Output:



### 4.1.2. Education Qualification Count based on Employment

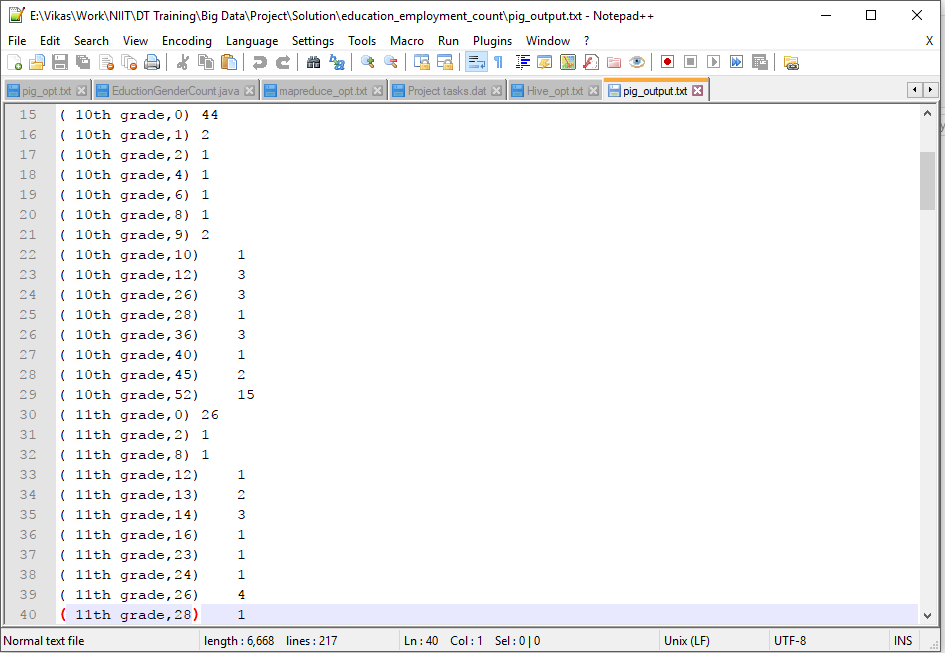
|  |
| --- |
| Hive Solution:  select education, weeksworked, count(\*) from census group by education, weeksworked; |

Output:



|  |
| --- |
| Pig Soluton:  bag1 = load '/home/hduser/census.txt' using PigStorage(',') as (age:int, education:chararray, maritalstatus:chararray, gender:chararray, taxfilerstatus:chararray, income:float, parents:chararray, countryofbirth:chararray, citizenship:chararray, weeksworked:int);  bag2 = foreach bag1 generate education, weeksworked;  bag3 = group bag2 by (education, weeksworked);  bag4 = foreach bag3 generate group, COUNT(bag2.education);  store bag4 into 'education\_employment\_count' |

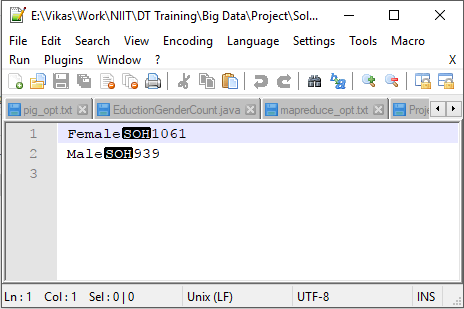
Output:



### 4.1.3 Calculate Sex Ratio (Male : Female)

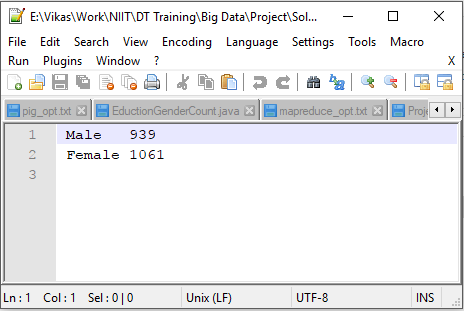
|  |
| --- |
| Hive Solution:  select gender, count(\*) from census group by gender |

Output:



|  |
| --- |
| Pig Solution:  bag2 = foreach bag1 generate gender;  bag3 = group bag2 by gender;  bag4 = foreach bag3 generate group, COUNT(bag2.gender);  store bag4 into 'pig\_output1'; |

Output:

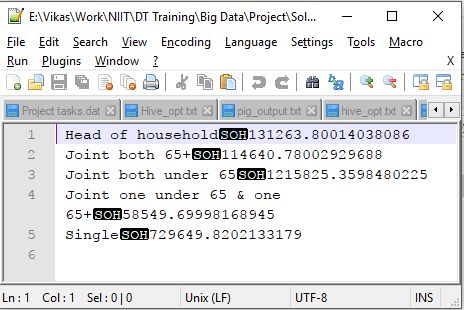


## 4.2 Financial Analysis

### 4.2.1 Total Income of different types of Tax Payers

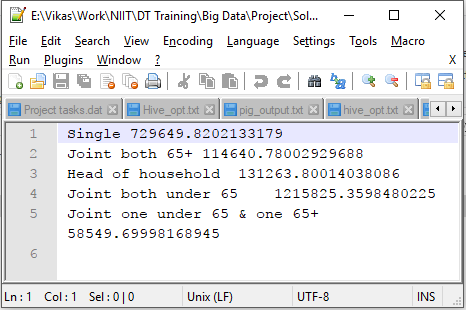
|  |
| --- |
| Hive Solution:  select taxfilerstatus, sum(income) from census where taxfilerstatus not like '%Nonfiler%' group by taxfilerstatus; |

Output:



|  |
| --- |
| Pig Solution:  bag1 = load '/home/hduser/census.txt' using PigStorage(',') as (age:int, education:chararray, maritalstatus:chararray, gender:chararray, taxfilerstatus:chararray, income:float, parents:chararray, countryofbirth:chararray, citizenship:chararray, weeksworked:int);  bag2 = foreach bag1 generate taxfilerstatus, income;  bag3 = filter bag2 by taxfilerstatus != ' Nonfiler';  bag4 = group bag3 by taxfilerstatus;  bag5 = foreach bag4 generate group, SUM(bag3.income);  store bag5 into 'pig\_output1' |

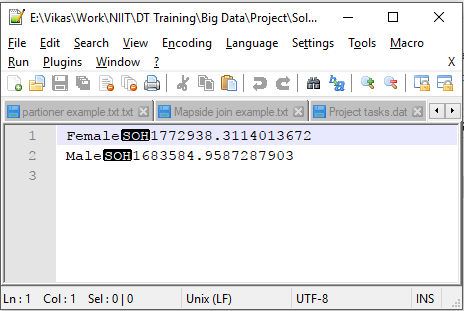
Output:



### 4.2.2. Gender wise Total Income Generated

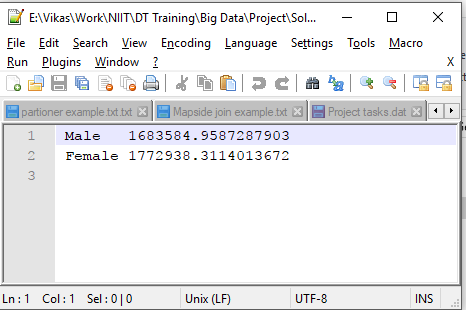
|  |
| --- |
| Hive Solution:  select gender, sum(income) from census group by gender; |

Output:



|  |
| --- |
| Pig Solution:  bag1 = load '/home/hduser/census.txt' using PigStorage(',') as (age:int, education:chararray, maritalstatus:chararray, gender:chararray, taxfilerstatus:chararray, income:float, parents:chararray, countryofbirth:chararray, citizenship:chararray, weeksworked:int);  bag2 = foreach bag1 generate income, gender;  bag3 = group bag2 by gender;  bag4 = foreach bag3 generate group, SUM(bag2.income);  store bag4 into 'pig\_output'; |

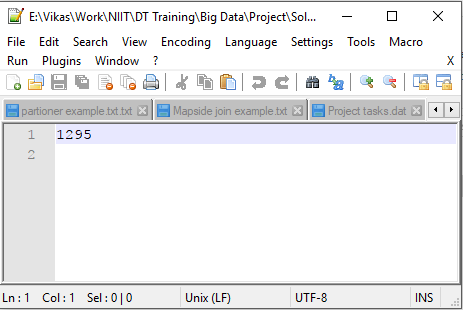
Output:



4.2.3 Total Income of different types of Tax Payers

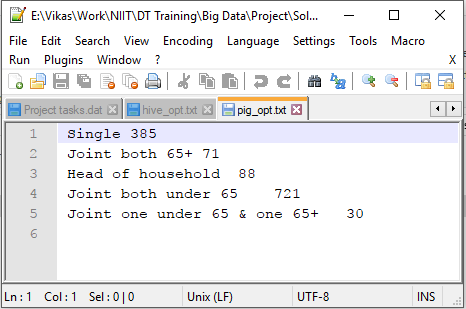
|  |
| --- |
| Hive Solution:  select count(taxfilerstatus) from census where taxfilerstatus not like "%Nonfiler%"; |

Output:



|  |
| --- |
| Pig Solution:  bag1 = load '/home/hduser/census.txt' using PigStorage(',') as (age:int, education:chararray, maritalstatus:chararray, gender:chararray, taxfilerstatus:chararray, income:float, parents:chararray, countryofbirth:chararray, citizenship:chararray, weeksworked:int);  bag2 = foreach bag1 generate taxfilerstatus;  bag3 = filter bag2 by taxfilerstatus != ' Nonfiler';  bag4 = group bag3 by taxfilerstatus;  bag5 = foreach bag4 generate group, COUNT(bag3.taxfilerstatus); |

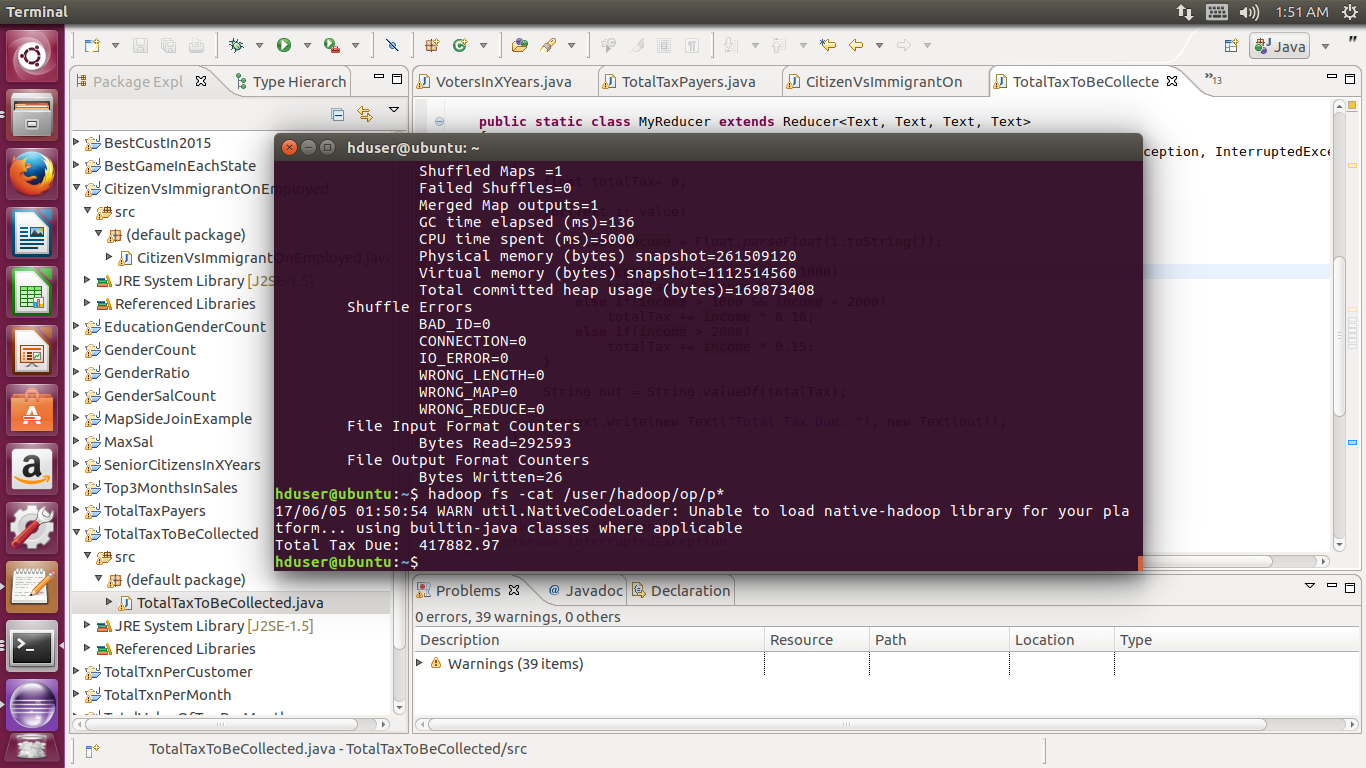
Output:



|  |
| --- |
| Java Code:  import java.io.IOException;  import org.apache.hadoop.conf.Configuration;  import org.apache.hadoop.fs.FileSystem;  import org.apache.hadoop.fs.Path;  import org.apache.hadoop.io.\*;  import org.apache.hadoop.mapreduce.Job;  import org.apache.hadoop.mapreduce.Mapper;  import org.apache.hadoop.mapreduce.Reducer;  import org.apache.hadoop.mapreduce.Mapper.Context;  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  public class TotalTaxPayers  {  public static class MyMapper extends Mapper<LongWritable, Text, Text, Text>  {  public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException  {  String[] arr = value.toString().split(",");  String taxFilerStatus = arr[4];  context.write(new Text(""), new Text(taxFilerStatus));  }  }    public static class MyReducer extends Reducer<Text, Text, Text, IntWritable>  {  public void reduce(Text key, Iterable<Text> value, Context context) throws IOException, InterruptedException  {  int sum= 0;  for(Text i: value)  {  if(!(i.toString().trim().equalsIgnoreCase("nonfiler")))  sum++;  }  context.write(new Text("Total Tax Payers"), new IntWritable(sum));  }  }  /\*\*  \* @param args  \* @throws IOException  \* @throws InterruptedException  \* @throws ClassNotFoundException  \*/  public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException  {  // TODO Auto-generated method stub  Configuration cfg = new Configuration();  Job job = Job.getInstance(cfg, "TotalTaxPayers");  job.setJarByClass(TotalTaxPayers.class);  job.setMapperClass(MyMapper.class);  job.setReducerClass(MyReducer.class);    job.setMapOutputKeyClass(Text.class);  job.setMapOutputValueClass(Text.class);    FileInputFormat.addInputPath(job, new Path(args[0]));  FileSystem.get(cfg).delete(new Path(args[1]), true);  FileOutputFormat.setOutputPath(job, new Path(args[1]));  System.exit(job.waitForCompletion(true) ? 0 : 1);  }  } |

### 4.2.4 Total Tax To Be collected

|  |
| --- |
| Java Solution:  import java.io.IOException;  import org.apache.hadoop.conf.Configuration;  import org.apache.hadoop.fs.FileSystem;  import org.apache.hadoop.fs.Path;  import org.apache.hadoop.io.LongWritable;  import org.apache.hadoop.io.Text;  import org.apache.hadoop.mapreduce.Job;  import org.apache.hadoop.mapreduce.Mapper;  import org.apache.hadoop.mapreduce.Reducer;  import org.apache.hadoop.mapreduce.Mapper.Context;  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  public class TotalTaxToBeCollected  {  public static class MyMapper extends Mapper<LongWritable, Text, Text, Text>  {  public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException  {  String[] arr = value.toString().split(",");    context.write(new Text(""), new Text(arr[5]));  }  }  public static class MyReducer extends Reducer<Text, Text, Text, Text>  {  public void reduce(Text key, Iterable<Text> value, Context context) throws IOException, InterruptedException  {  float totalTax= 0;    for(Text i: value)  {  float income = Float.parseFloat(i.toString());    if(income > 500 && income < 1000)  totalTax += income \* 0.05;  else if(income > 1000 && income < 2000)  totalTax += income \* 0.10;  else if(income > 2000)  totalTax += income \* 0.15;  }    String out = String.valueOf(totalTax);    context.write(new Text("Total Tax Due: "), new Text(out));  }  }    /\*\*  \* @param args  \* @throws IOException  \* @throws InterruptedException  \* @throws ClassNotFoundException  \*/  public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException  {  // TODO Auto-generated method stub  Configuration cfg = new Configuration();  Job job = Job.getInstance(cfg, "TotalTaxToBeCollected");  job.setJarByClass(TotalTaxToBeCollected.class);  job.setMapperClass(MyMapper.class);  job.setReducerClass(MyReducer.class);    job.setMapOutputKeyClass(Text.class);  job.setMapOutputValueClass(Text.class);    FileInputFormat.addInputPath(job, new Path(args[0]));  FileSystem.get(cfg).delete(new Path(args[1]), true);  FileOutputFormat.setOutputPath(job, new Path(args[1]));  System.exit(job.waitForCompletion(true) ? 0 : 1);  }  } |



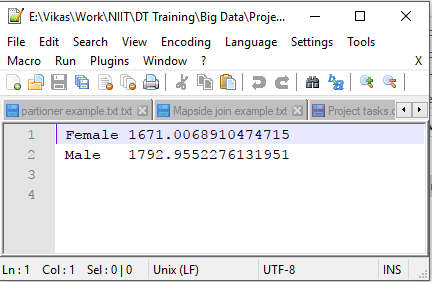
### 4.2.5 Per Capita Income

|  |
| --- |
| Hive Solution:  select sum(income) / count(\*) from census; |

### 4.2.6. Gender wise Per Capita Income

|  |
| --- |
| Hive Solution:  select gender, sum(income) from census group by gender; |

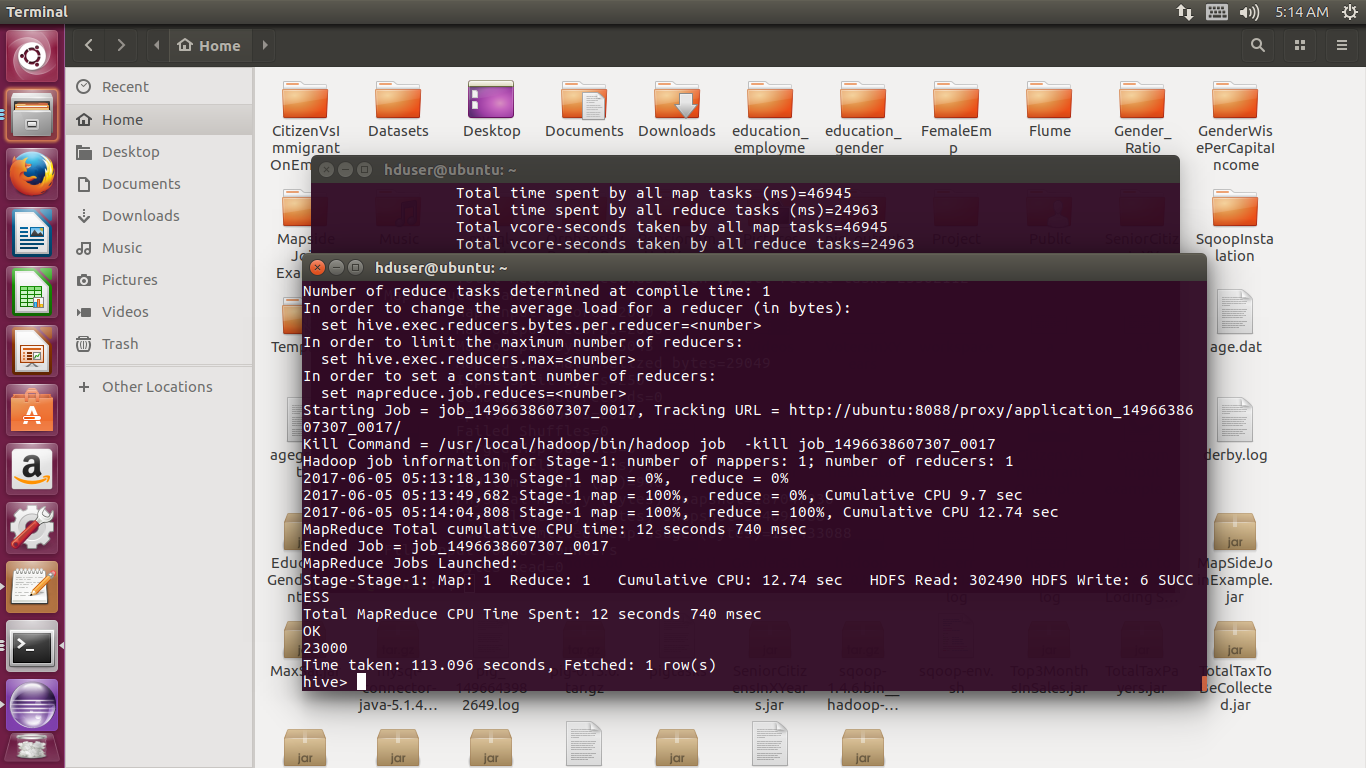
Ouput:



## Social Analysis

### 4.3. 1 Pension Amount to be added after x years

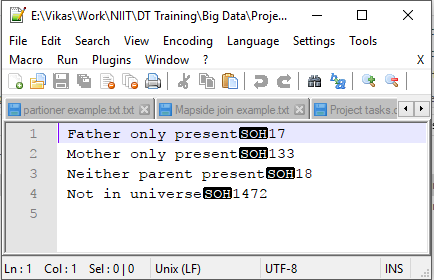
|  |  |
| --- | --- |
| Hive Solution:  Hive Solution: select count(\*)\*500 from census where age+4 > 60 and age < 60; |  |

****

### 4.3.2 No. of Orphans for each category based on Parents Present

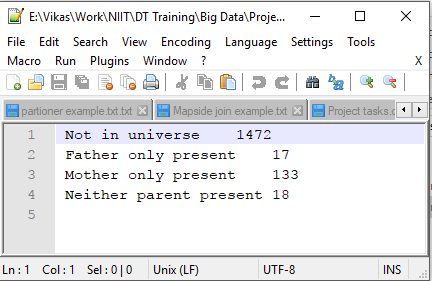
|  |
| --- |
| Hive Solution:  select parents, Count(\*) from census where parents not like '%Both parents present%' group by parents; |

Output:



|  |
| --- |
| Pig Solution:  bag1 = load '/home/hduser/census.txt' using PigStorage(',') as (age:int, education:chararray, maritalstatus:chararray, gender:chararray, taxfilerstatus:chararray, income:float, parents:chararray, countryofbirth:chararray, citizenship:chararray, weeksworked:int);  bag2 = foreach bag1 generate parents;  bag3 = filter bag2 by parents != ' Both parents present';  bag4 = group bag3 by parents;  bag5 = foreach bag4 generate group, COUNT(bag3.parents);  14 store bag5 into 'pig\_output' |

Output:



### 4.3.3 No. of Employable Female Citizens who are Widows or Divorced

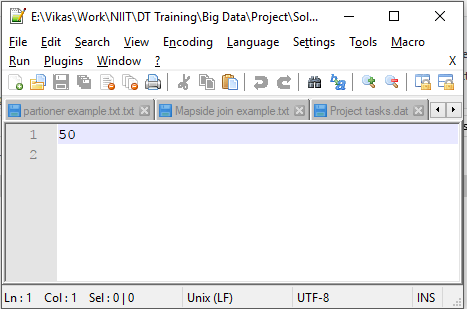
|  |
| --- |
| Hive Solution:  select count(\*) from census where gender like '%Female%' and (maritalstatus like '%Divorced%' or maritalstatus like '%Widowed%') and weeksworked > 0; |

## Planning Analysis

### 4.3.1 No. of Voters to get added in next X years

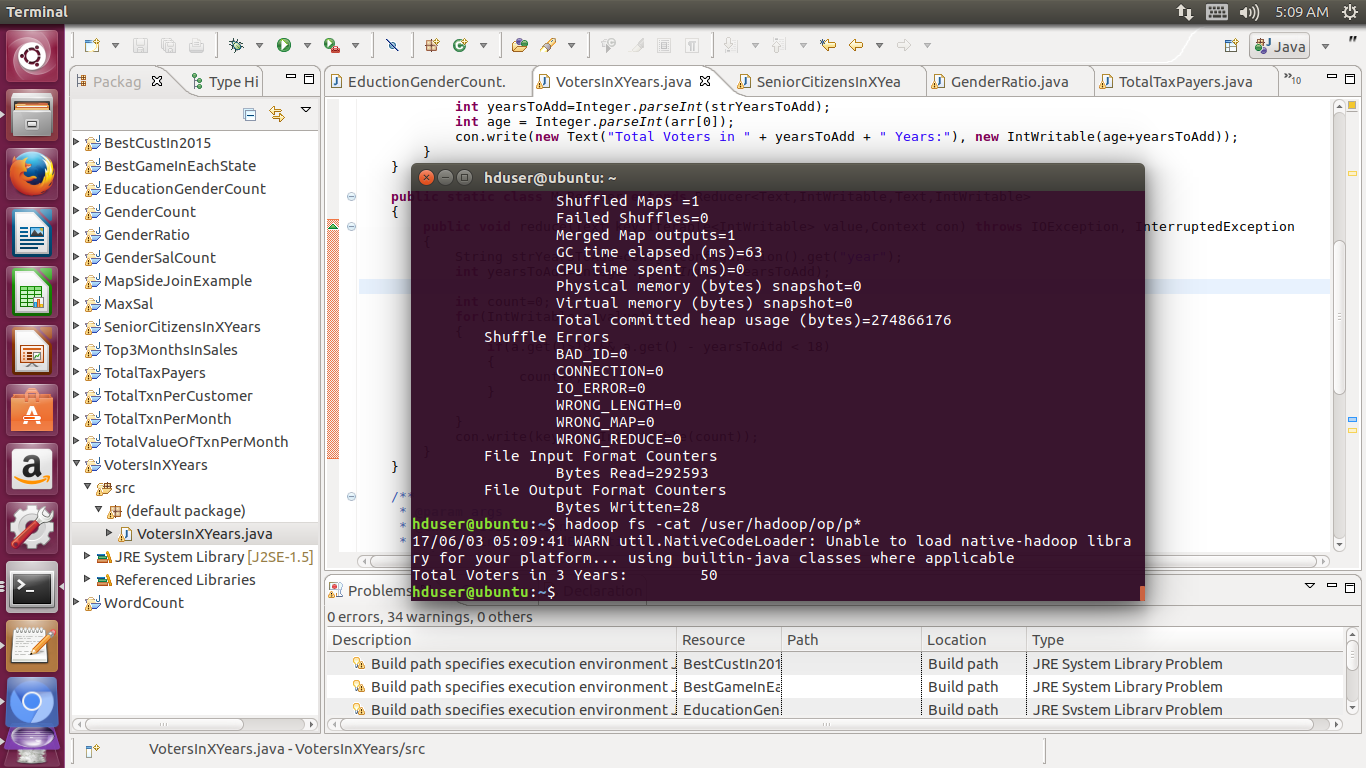
|  |
| --- |
| Hive Solution:  select count(\*) from census where age+3 > 18 and age < 18; |

Output:



|  |
| --- |
| Java Solution:  import java.io.IOException;  import java.util.Scanner;  import org.apache.hadoop.conf.Configuration;  import org.apache.hadoop.fs.FileSystem;  import org.apache.hadoop.fs.Path;  import org.apache.hadoop.io.IntWritable;  import org.apache.hadoop.io.LongWritable;  import org.apache.hadoop.io.Text;  import org.apache.hadoop.mapreduce.Job;  import org.apache.hadoop.mapreduce.Mapper;  import org.apache.hadoop.mapreduce.Reducer;  import org.apache.hadoop.mapreduce.Mapper.Context;  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  public class VotersInXYears  {  public static class MyMapper extends Mapper<LongWritable,Text,Text,IntWritable>  {  public void map(LongWritable key,Text value,Context con) throws IOException, InterruptedException  {  String arr[]=value.toString().split(",");  String strYearsToAdd=con.getConfiguration().get("year");  int yearsToAdd=Integer.parseInt(strYearsToAdd);  int age = Integer.parseInt(arr[0]);  con.write(new Text("Total Voters in " + yearsToAdd + " Years:"), new IntWritable(age+yearsToAdd));  }  }  public static class MyReducer extends Reducer<Text,IntWritable,Text,IntWritable>  {  public void reduce(Text key,Iterable<IntWritable> value,Context con) throws IOException, InterruptedException  {  String strYearsToAdd=con.getConfiguration().get("year");  int yearsToAdd=Integer.parseInt(strYearsToAdd);    int count=0;  for(IntWritable a:value)  {  if(a.get()>18 && a.get() - yearsToAdd < 18)  {  count++;  }    }  con.write(key,new IntWritable(count));  }  }  /\*\*  \* @param args  \* @throws IOException  \* @throws InterruptedException  \* @throws ClassNotFoundException  \*/  public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException  {  // TODO Auto-generated method stub  Configuration cfg=new Configuration();  Scanner s=new Scanner(System.in);  System.out.println("Enter years in which you need to know total senior citizens");  String year=s.next();  cfg.set("year", year);  Job job =Job.getInstance(cfg,"VotersInXYears");  job.setJarByClass(VotersInXYears.class);  job.setMapperClass(MyMapper.class);  job.setMapOutputKeyClass(Text.class);  job.setMapOutputValueClass(IntWritable.class);  job.setReducerClass(MyReducer.class);  job.setOutputKeyClass(Text.class);  job.setOutputValueClass(IntWritable.class);  FileInputFormat.addInputPath(job,new Path(args[0]));  FileSystem.get(cfg).delete(new Path(args[1]),true);  FileOutputFormat.setOutputPath(job,new Path(args[1]));  System.exit(job.waitForCompletion(true) ? 0 : 1);  }  } |

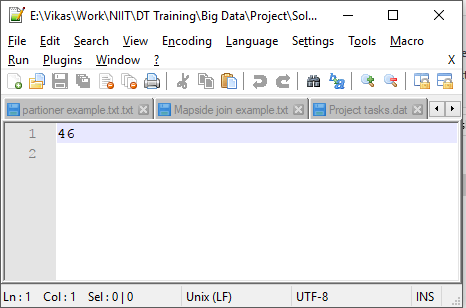
Output:



### 4.4.2 No. of Senior Citizen to get added in next X years

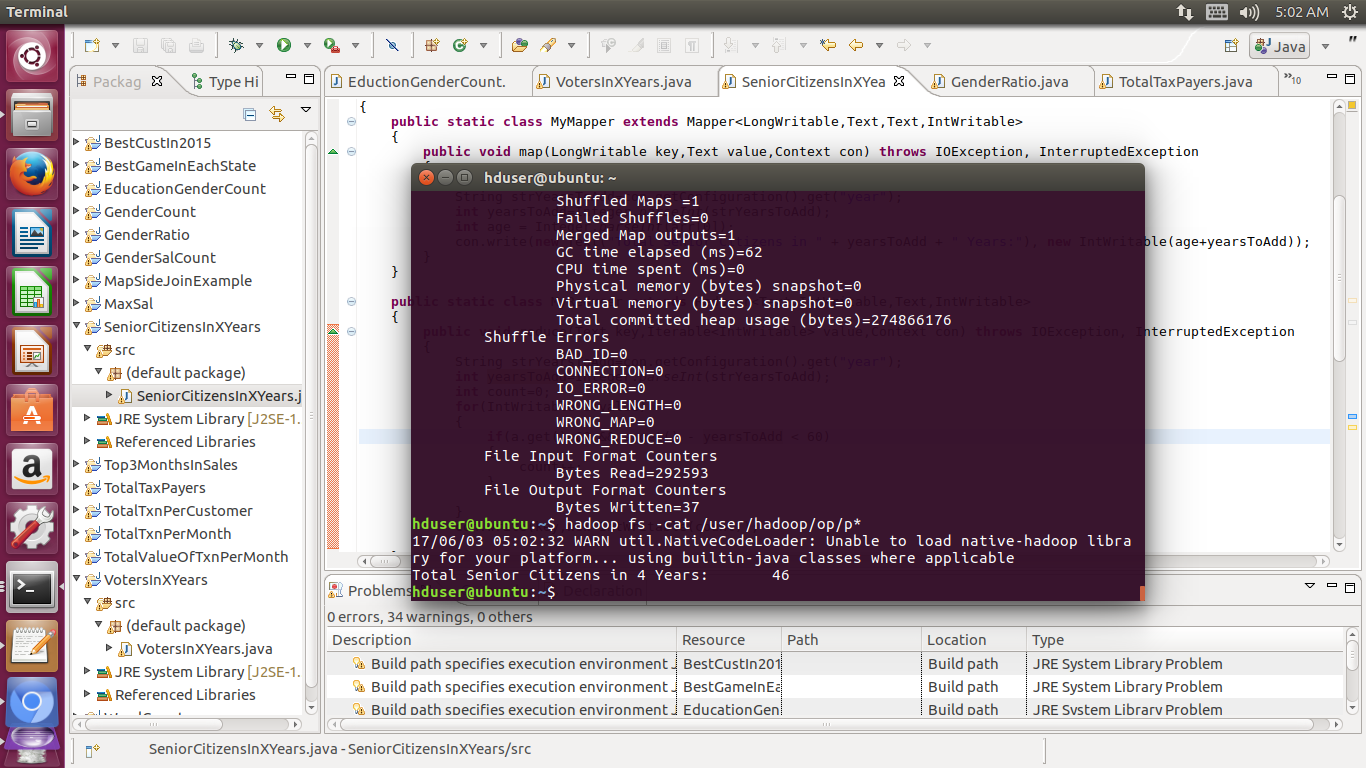
|  |
| --- |
| Hive Solution:  select count(\*) from census where age+4 > 60 and age > 60; |

Output:



|  |
| --- |
| Java Solution:  import java.io.IOException;  import java.util.Scanner;  import org.apache.hadoop.conf.Configuration;  import org.apache.hadoop.fs.FileSystem;  import org.apache.hadoop.fs.Path;  import org.apache.hadoop.io.\*;  import org.apache.hadoop.mapreduce.Job;  import org.apache.hadoop.mapreduce.Mapper;  import org.apache.hadoop.mapreduce.Reducer;  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  public class SeniorCitizensInXYears  {  public static class MyMapper extends Mapper<LongWritable,Text,Text,IntWritable>  {  public void map(LongWritable key,Text value,Context con) throws IOException, InterruptedException  {  String arr[]=value.toString().split(",");  String strYearsToAdd=con.getConfiguration().get("year");  int yearsToAdd=Integer.parseInt(strYearsToAdd);  int age = Integer.parseInt(arr[0]);  con.write(new Text("Total Senior Citizens in " + yearsToAdd + " Years:"), new IntWritable(age+yearsToAdd));  }  }  public static class MyReducer extends Reducer<Text,IntWritable,Text,IntWritable>  {  public void reduce(Text key,Iterable<IntWritable> value,Context con) throws IOException, InterruptedException  {  String strYearsToAdd=con.getConfiguration().get("year");  int yearsToAdd=Integer.parseInt(strYearsToAdd);  int count=0;  for(IntWritable a:value)  {  if(a.get()>60 && a.get() - yearsToAdd < 60)  {  count++;  }    }  con.write(key,new IntWritable(count));  }  }  /\*\*  \* @param args  \* @throws InterruptedException  \* @throws IOException  \* @throws ClassNotFoundException  \*/  public static void main(String[] args) throws ClassNotFoundException, IOException, InterruptedException  {  // TODO Auto-generated method stub  Configuration cfg=new Configuration();  Scanner s=new Scanner(System.in);  System.out.println("Enter years in which you need to know total senior citizens");  String year=s.next();  cfg.set("year", year);  Job job =Job.getInstance(cfg,"SeniorCitizensInXYears");  job.setJarByClass(SeniorCitizensInXYears.class);  job.setMapperClass(MyMapper.class);  job.setMapOutputKeyClass(Text.class);  job.setMapOutputValueClass(IntWritable.class);  job.setReducerClass(MyReducer.class);  job.setOutputKeyClass(Text.class);  job.setOutputValueClass(IntWritable.class);  FileInputFormat.addInputPath(job,new Path(args[0]));  FileSystem.get(cfg).delete(new Path(args[1]),true);  FileOutputFormat.setOutputPath(job,new Path(args[1]));  System.exit(job.waitForCompletion(true) ? 0 : 1);    }  } |

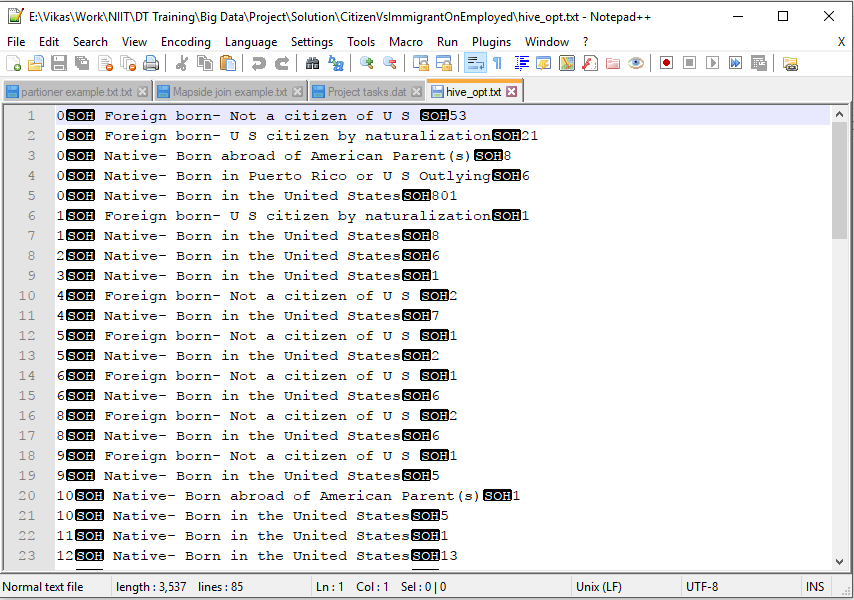
Output:



### 4.4.3 Citizen vs. Immigrants Ratio for all Employed

|  |
| --- |
| Hive Solution:  select weeksworked, citizenship, count(\*) from census group by weeksworked, citizenship; |

Output:



|  |
| --- |
| Pig Solution:  bag1 = load '/home/hduser/census.txt' using PigStorage(',') as (age:int, education:chararray, maritalstatus:chararray, gender:chararray, taxfilerstatus:chararray, income:float, parents:chararray, countryofbirth:chararray, citizenship:chararray, weeksworked:int);  bag2 = foreach bag1 generate citizenship, weeksworked;  bag3 = group bag2 by (citizenship, weeksworked);  bag4 = foreach bag3 generate group, COUNT(bag2.weeksworked);  store bag4 into 'pig\_output' |

|  |
| --- |
| Java Solution:  import java.io.IOException;  import org.apache.hadoop.conf.Configuration;  import org.apache.hadoop.fs.FileSystem;  import org.apache.hadoop.fs.Path;  import org.apache.hadoop.io.IntWritable;  import org.apache.hadoop.io.LongWritable;  import org.apache.hadoop.io.Text;  import org.apache.hadoop.mapreduce.Job;  import org.apache.hadoop.mapreduce.Mapper;  import org.apache.hadoop.mapreduce.Reducer;  import org.apache.hadoop.mapreduce.Mapper.Context;  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  //Citizen vs. Immigrants Ratio for all Employed  public class CitizenVsImmigrantOnEmployed  {  public static class MyMapper extends Mapper<LongWritable, Text, Text, Text>  {  public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException  {  String[] arr = value.toString().split(",");    String weeksWorked = arr[9];    if(weeksWorked != "0")  context.write(new Text(weeksWorked), new Text(arr[8]));  }  }    public static class MyReducer extends Reducer<Text, Text, Text, Text>  {  public void reduce(Text key, Iterable<Text> value, Context context) throws IOException, InterruptedException  {  int ccount= 0, ncount = 0; //citizencount - noncitizencount    for(Text i: value)  {  if(i.toString().trim().contains("Native") || i.toString().trim().contains("naturalization"))  ccount++;  else  ncount++; }  String out = String.valueOf(ccount) + " : " + String.valueOf(ncount);  context.write(new Text(key), new Text(out));  }  }    /\*\*  \* @param args  \* @throws IOException  \* @throws InterruptedException  \* @throws ClassNotFoundException  \*/  public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException  {  // TODO Auto-generated method stub    Configuration cfg = new Configuration();  Job job = Job.getInstance(cfg, "CitizenVsImmigrationOnEmployed");  job.setJarByClass(CitizenVsImmigrantOnEmployed.class);  job.setMapperClass(MyMapper.class);  job.setReducerClass(MyReducer.class);    job.setMapOutputKeyClass(Text.class);  job.setMapOutputValueClass(Text.class);    FileInputFormat.addInputPath(job, new Path(args[0]));  FileSystem.get(cfg).delete(new Path(args[1]), true);  FileOutputFormat.setOutputPath(job, new Path(args[1]));  System.exit(job.waitForCompletion(true) ? 0 : 1);  }  } |

Output:

