SEIS 736 Big Data Architecture
International Census Data Analysis
Project Report

Due: 12/13/2018

Submitted by

Satish Dandayudhapani

Overview:

Researchers and businesses are using tools such as Spark to solve Big Data problems. Spark takes MapReduce to the next level with less expensive shuffles in the data processing. With capabilities like in-memory data storage and near real-time processing, the performance can be several times faster than other big data technologies. The purpose of this paper is to describe my Spark project focused on the International Census Data Analysis. The paper will describe the nature of the analysis, the Spark map-reduce programming algorithms.

Data Source:

The dataset provides estimates of country populations since 1950 and projections through 2050. Specifically, the data set includes midyear population figures broken down by age and gender assignment at birth. Additionally, they provide time-series data for attributes including fertility rates, birth rates, death rates, and migration rates. The U.S. Census Bureau provides estimates and projections for countries and areas that are recognized by the U.S. Department of State that have a population of at least 5,000.

https://www.census.gov/data-tools/demo/idb/informationGateway.php

https://www.kaggle.com/census/international-data

The dataset includes a comprehensive set of indicators, as produced by the U.S. Census Bureau since the 1950's. Through sponsorship from various U.S. government agencies, the dataset is updated on a regular basis to provide information needed for research, program planning, and policy-making decisions, in the U.S. and globally. Data included in the dataset consist of indicators developed from censuses, surveys, administrative records, and special measures of HIV/AIDS-related mortality. Through evaluation and adjustment of data from these sources, measures of population, mortality, fertility, and net migration are estimated for current and past years and then used as the basis for projections to 2050. In this project, the analysis was done using 4 files from the dataset and only considering 1950 to 2018 years.
The dataset is about 1.7 GB of size and primarily use Spark with Scala at the RDD level to explore and analyze the data. The results are plotted using Excel to answer the below questions while exploring, analyzing and visualizing:

- Life expectancy by gender overall
- Life expectancy by country, visualize high and low life expectancy
- Birth rate Vs. Death rate by country
- Male/Female ratio at birth
- Migration between countries
- Population balance (example for United States)

Data Description and Schema:

There are 8 source files in the International Census dataset. I am only using 4 files for this project their File names, description and schema as follows

File Name: age_specific_fertility_rates.csv
Description: Age-specific fertility rates
Schema Column Names: country_code, country_name, year, fertility_rate_15_19, fertility_rate_20_24, fertility_rate_25_29, fertility_rate_30_34, fertility_rate_35_39, fertility_rate_40_44, fertility_rate_45_49, total_fertility_rate, gross_reproduction_rate, sex_ratio_at_birth

File Name: birth_death_growth_rates.csv
Description: Birth and death rates
Schema Column Names: country_code, country_name, year, crude_birth_rate, crude_death_rate, net_migration, rate_natural_increase, growth_rate

File Name: midyear_population_age_sex.csv
Description: Mid-year population by gender
Schema Column Names: country_code, country_name, year,sex, max_age, population_age_0, population_age_1, population_age_2, population_age_3, population_age_4, population_age_5, population_age_6, population_age_7, population_age_8, population_age_9, population_age_10, population_age_11, population_age_12, population_age_13, population_age_14, population_age_15, population_age_16, population_age_17, population_age_18, population_age_19, population_age_20, population_age_21, population_age_22, population_age_23, population_age_24, population_age_25, population_age_26, population_age_27, population_age_28, population_age_29, population_age_30, population_age_31, population_age_32, population_age_33, population_age_34, population_age_35, population_age_36, population_age_37, population_age_38, population_age_39, population_age_40, population_age_41, population_age_42, population_age_43, population_age_44, population_age_45, population_age_46, population_age_47, population_age_48, population_age_49, population_age_50, population_age_51, population_age_52, population_age_53, population_age_54, population_age_55, population_age_56, population_age_57, population_age_58, population_age_59, population_age_60, population_age_61, population_age_62, population_age_63, population_age_64, population_age_65, population_age_66, population_age_67, population_age_68, population_age_69, population_age_70, population_age_71, population_age_72, population_age_73, population_age_74, population_age_75, population_age_76, population_age_77, population_age_78, population_age_79, population_age_80, population_age_81, population_age_82, population_age_83, population_age_84, population_age_85, population_age_86, population_age_87, population_age_88, population_age_89, population_age_90, population_age_91, population_age_92, population_age_93, population_age_94, population_age_95, population_age_96, population_age_97, population_age_98, population_age_99, population_age_100

File Name: mortality_life_expectancy.csv
Description: Life expectancy by country and year

Schema Column Names: country_code, country_name, year, infant_mortality, infant_mortality_male, infant_mortality_female, life_expectancy, life_expectancy_male, life_expectancy_female, mortality_rate_under5, mortality_rate_under5_male, mortality_rate_under5_female, mortality_rate_1to4, mortality_rate_1to4_male, mortality_rate_1to4_female

Data pre-processing:

All the files were preprocessed before transforming the data to desired results. Below are the few steps are done as part of preprocessing:

- Reading the sources files into RDD
- Removing the header from all the files and store the results into an RDD.
- Filter data greater than year 2018 and store it into RDD.

Data Issues:

There were no bad data issues identified in the source files.

Spark Algorithm:

Step 1

The algorithm is expecting the input file location and 3 CSV file names for output. The initial step will load input each file into an RDD after applying the preprocessing steps and resulting in 4 RDD's, one per each source file. This allows the further steps to access the data from these RDD's to transform the data.

Step 2 (Iterative)

The algorithm will parse and aggregate the required fields by looping through the files and store it into a new RDD's

Step 3

As part of the final step the algorithm will join the files, apply filtering were required and repartition to produce the CSV files as output.

Tools:

The project is programmed using Spark and Scala language. The project files are compiled using Scala build tool and the jar file is executed on Hadoop cluster. To take advantage of the parallelism the source files are placed on Hadoop Distributed File System (HDFS). Below are the tools and used for this project.

- Hadoop Cluster
- Spark 1.6
- Scala 2.10.6

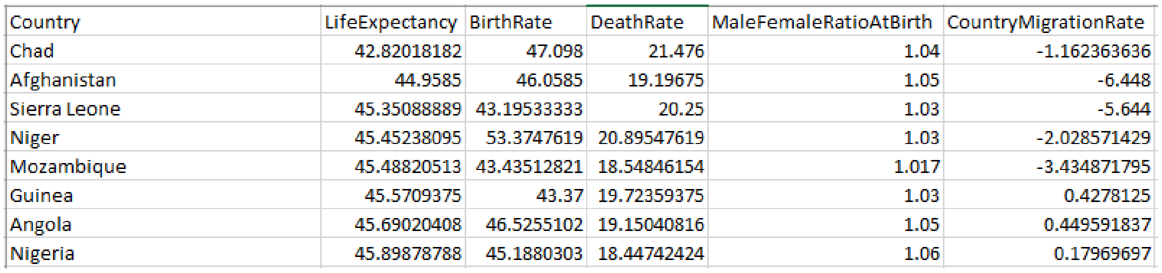
- IntelliJ IDEA
- Scala build tool (SBT) 0.13

Output:

The spark application outputs three CSV files. The name of the files is based on the arguments passed while executing the jar file. Using the output file below are the few questions being answered. The files schema structure and the results from each file as follows:

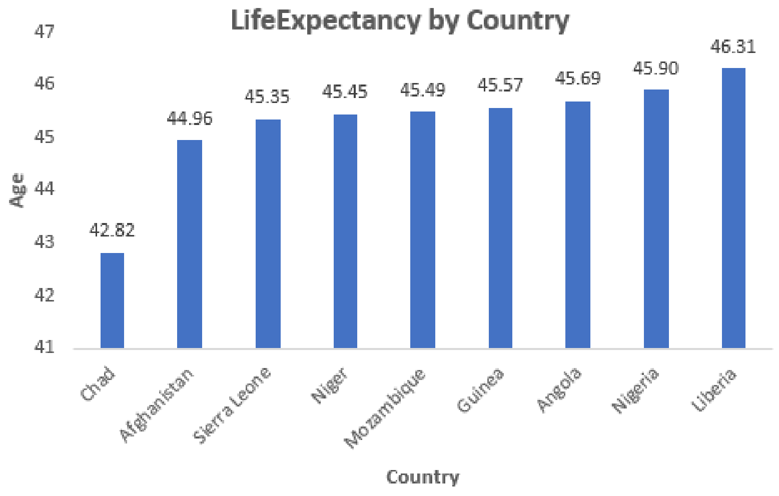
File 1: Life Expectancy by country with Birth, Death rates, Male Vs Female ratio and Migration rate

Columns: Country, LifeExpectancy, BirthRate, DeathRate, MaleFemaleRatioatBirth, CountryMigrationRate

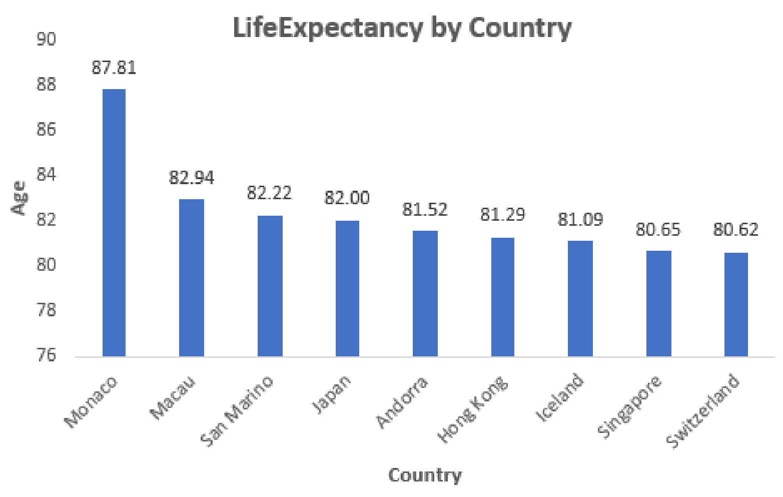


Country	LifeExpectancy	BirthRate	DeathRate	MaleFemaleRatioAtBirth	CountryMigrationRate
Chad	42.82018182	47.098	21.476	1.04	-1.162363636
Afghanistan	44.9585	46.0585	19.19675	1.05	-6.448
Sierra Leone	45.35088889	43.19533333	20.25	1.03	-5.644
Niger	45.45238095	53.3747619	20.89547619	1.03	-2.028571429
Mozambique	45.48820513	43.43512821	18.54846154	1.017	-3.434871795
Guinea	45.5709375	43.37	19.72359375	1.03	0.4278125
Angola	45.69020408	46.5255102	19.15040816	1.05	0.449591837
Nigeria	45.89878788	45.1880303	18.44742424	1.06	0.17969697

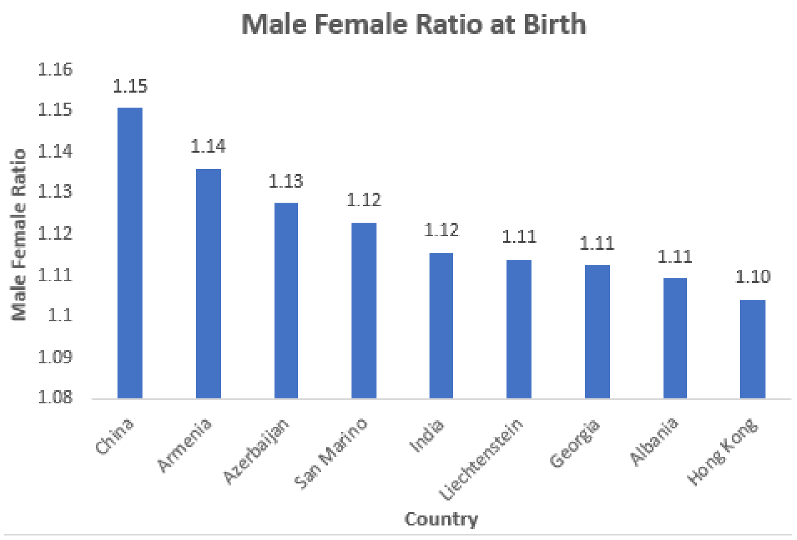
Life expectancy by country, visualize high and low life expectancy –



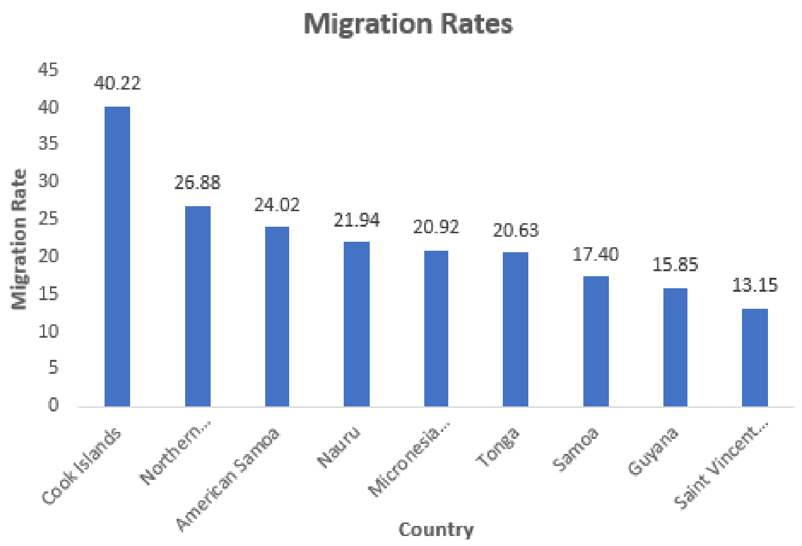
The chart shows 10 countries with low life expectancy



The chart shows 10 countries with high life expectancy



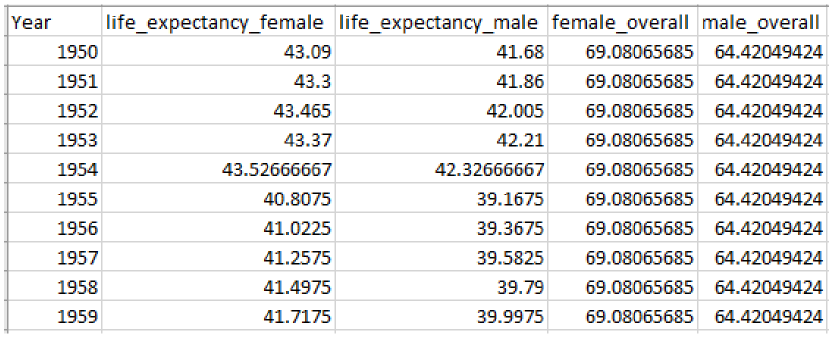
The chart shows 10 countries with high Male Female Ratio at Birth



The chart shows 10 countries with high migration rate to different countries

File 2: Life Expectancy of Male and Female by Year

Columns: year, life_expectancy_female, life_expectancy_male, female_overall, male_overall

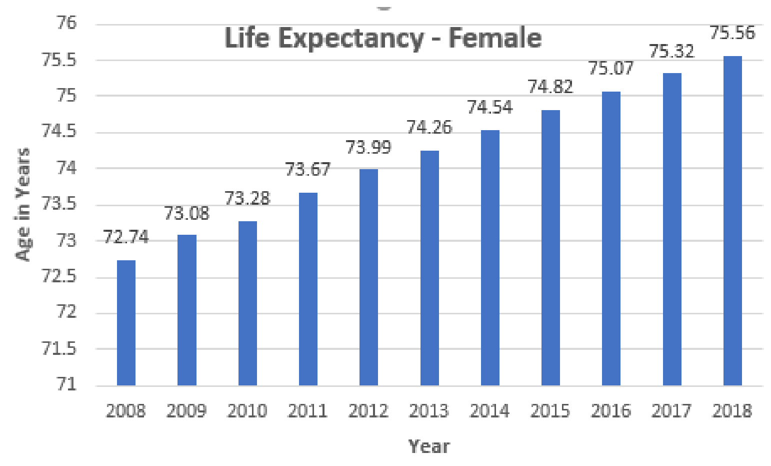


Year	life_expectancy_female	life_expectancy_male	female_overall	male_overall
1950	43.09	41.68	69.08065685	64.42049424
1951	43.3	41.86	69.08065685	64.42049424
1952	43.465	42.005	69.08065685	64.42049424
1953	43.37	42.21	69.08065685	64.42049424
1954	43.52666667	42.32666667	69.08065685	64.42049424
1955	40.8075	39.1675	69.08065685	64.42049424
1956	41.0225	39.3675	69.08065685	64.42049424
1957	41.2575	39.5825	69.08065685	64.42049424
1958	41.4975	39.79	69.08065685	64.42049424
1959	41.7175	39.9975	69.08065685	64.42049424

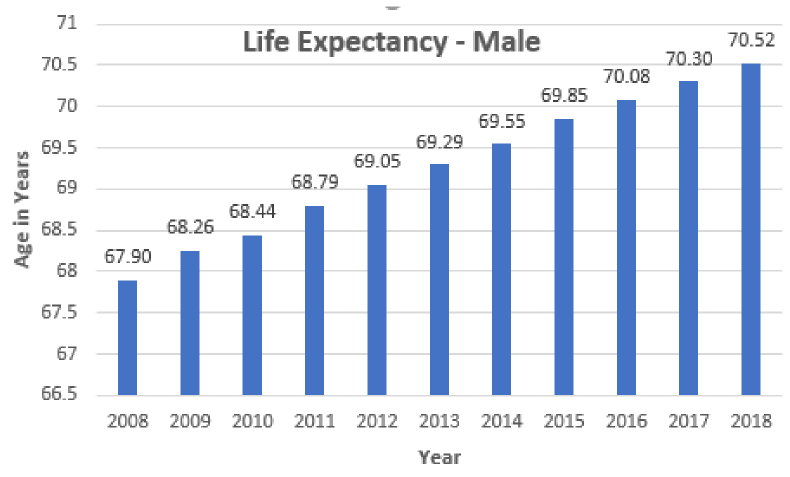
Life expectancy by gender overall –

Female: 69.08 years

Male: 64.42 years



The chart shows the increasing trend (72-75.5 years) of life expectancy (Female0 in the last ten years.

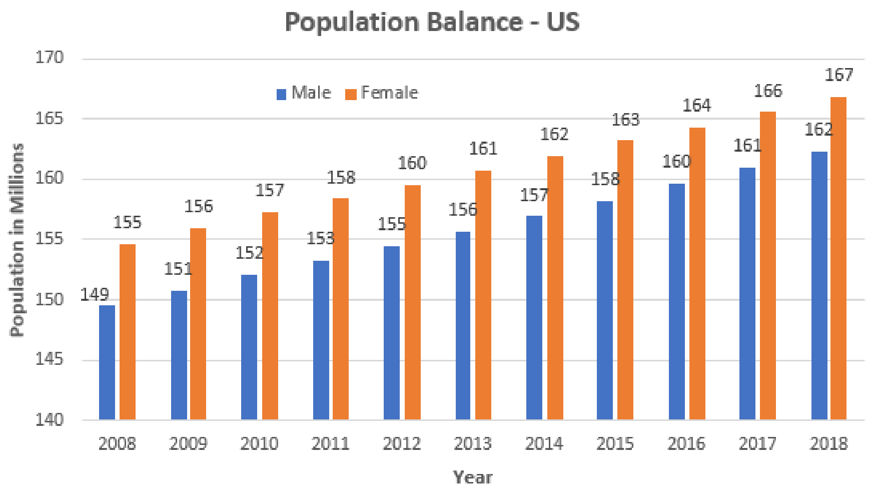


The chart shows the increasing trend (67.9-70.5 years) in life expectancy (Male) for the last ten years.

File 3: Population Balance Male Vs Female in United States

Columns: Country, Year, Male, Female

Country	Year	Male	Female
US	1980	110398730	116825951
US	1981	111502932	117962782
US	1982	112579409	119085049
US	1983	113646996	120144998
US	1984	114670261	121154641
US	1985	115729534	122194261
US	1986	116865159	123267728



The chart shows the Male/Female population balance from 2008 to 2018.

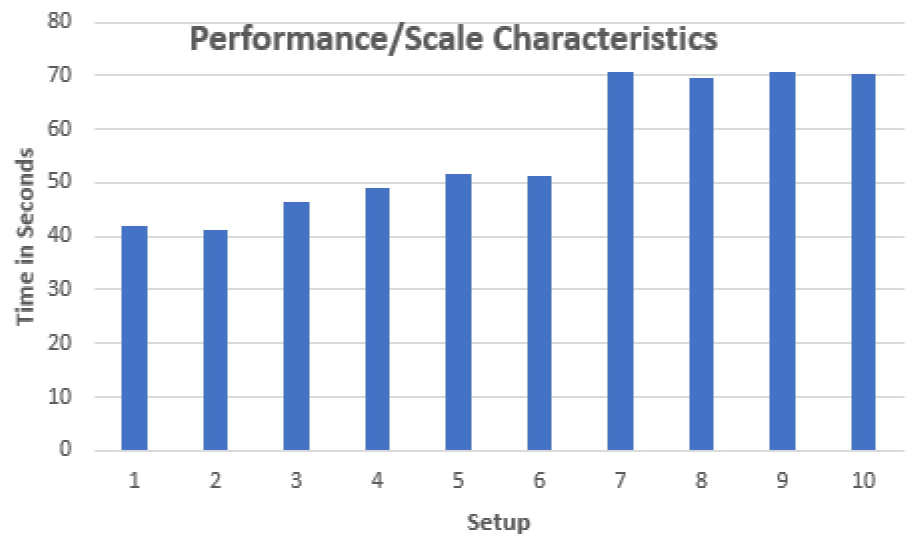
Data Validation:

Data validation was done using the subset of source csv data to calculate the sum, min, max and average in excel and compare it to the spark application output while running it in spark-shell.

Performance/Scale Characteristics:

The performance testing was performed on CensusDataAnalysis spark application using different parameters for the Memory for the executor, Number of executers and Executor cores on gps hadoop cluster. The table shows the different setting used to test the performance/scale characteristics of the application.

Setup	Executor Memory(G)	Number of Executers	Executor Cores	Time Taken (sec)
1	1	2	2	41.9
2	2	2	2	41.2
3	4	4	4	46.4
4	6	6	6	49.2
5	8	8	8	51.8
6	1	10	11	51.3
7	4	15	12	70.8
8	1	15	12	69.7
9	2	15	12	70.6
10	8	15	12	70.2



The chart shows the runtime for each execution using different parameter values. It is observed that the setup number 2(Memory: 2G, Executors: 2, Cores: 2) had performed better with 41.2 seconds to complete the process and setup 7 (Memory: 1G, Executors: 10, Cores: 11) performed poorly taking 70.8 seconds to complete the process.

If this must be done again I would analyzing on a bigger dataset (PIMS) and try to answer more questions while exploring the data. I would also try to make use of all the files from the data set to answer different questions. I would still use Spark platform with Scala but would like to execute it in a container using AWS cloud technologies.

Conclusion:

I learned a number of important lessons while working on this Big Data project. Approximately 60% of my time was spent understanding the data and come up with interesting questions to answer by analyzing the dataset while 40% was spent on coding and implementation (not including this report). There was good documentation on Spark features which I wanted to use in this project. Spark is an efficient and flexible platform for analyzing massive amounts of data. I was able to take earlier lessons from our Big Data Architecture course and extend those lessons using more advanced Spark features. I used those features to analyze the data in more efficient way and produce the CSV files which is easier to consume for visualizing tools. The interesting conclusions from this project in looking at the trends on life expectancy in males, females and overall by country, Migration rates between the countries and Male/Female population balance in Unites States.

I also tried out the project using spark data frames in spark-shell (code included towards the end.)

Containers:

As part of the class and exploration in Big Data technologies I learned about the Containers - Docker technology from the professor’s sabbatical study. I was able to setup Docker environment and download spark image from the Docker repository and create my own image with Spark, Scala and others for my workspace which I wanted to move around different machines to preserve the environment setup and ease not having it done multiple times and synchronized between machines.

Code:

```
import org.apache.spark.{SparkConf, SparkContext }

import org.apache.hadoop.conf.Configuration

import org.apache.hadoop.fs._

object CensusDataAnalysis {

  def main(args: Array[String]) {

    if (args.length < 4) {

      println("Usage: CensusDataAnalysis <inputFolder> <outputFile1> <outputFile2> <outputFile3")

      System.exit(1)

    }

    val sparkConf = new SparkConf().setAppName("Census Data Analysis")

    val sc = new SparkContext(sparkConf)

    /* age fertility rates */

    val age_fert = sc.textFile(args(0)+"/age_specific_fertility_rates.csv")

    val age_fert_hdr = age_fert.first()

    val age_fert_rdd = age_fert.filter(row => row!=age_fert_hdr).map(row => row.split(","))

    /* mortality life expectancy */

    val mortality_life = sc.textFile(args(0)+"/mortality_life_expectancy.csv")

    val mortality_life_hdr = mortality_life.first()

    val mortality_life_rdd = mortality_life.filter(row => row!=mortality_life_hdr).map(row =>
row.split(","))

    /* birth death growth rates */

    val birth_death = sc.textFile(args(0)+"/birth_death_growth_rates.csv")

    val birth_death_hdr = birth_death.first()

    val birth_death_rdd = birth_death.filter(row => row!=birth_death_hdr).map(row => row.split(","))

    /* mid year population */

    val midyear_pop = sc.textFile(args(0)+"/midyear_population_age_sex.csv")

    val midyear_pop_hdr = midyear_pop.first()
    val midyear_pop_rdd = midyear_pop.filter(row => row!=midyear_pop_hdr).map(row => row.split(","))


    /* Life expectancy by gender overall - as of 2018 */

    val life_expectancy_male_overall= mortality_life_rdd.filter(row => row(2)<="2018")

      .map(array => array(7).toDouble)

      .reduce(_+_)/mortality_life_rdd.filter(row => row(2)<="2018").count()

    val life_expectancy_female_overall=mortality_life_rdd.filter(row => row(2)<="2018")

      .map(array => array(8).toDouble)

      .reduce(_+_)/mortality_life_rdd.filter(row => row(2)<="2018").count()

    val life_expectancy_male_overall_by_year=mortality_life_rdd.filter(row => row(2)<="2018")

      .map(array => (array(2),array(7).toDouble)).mapValues(value => (value,1))

      .reduceByKey { case ((sumL, countL), (sumR, countR)) => (sumL + sumR, countL + countR) }

      .mapValues { case(sum, count) => sum / count }.sortByKey(ascending=true)

    val life_expectancy_female_overall_by_year=mortality_life_rdd.filter(row => row(2)<="2018")

      .map(array => (array(2),array(8).toDouble)).mapValues(value => (value,1))

      .reduceByKey { case ((sumL, countL), (sumR, countR)) => (sumL + sumR, countL + countR) }

      .mapValues { case(sum, count) => sum / count }.sortByKey(ascending=true)


    /* Life expectancy by country, visualize high and low life expectancy - as of 2018 */

    val life_expectancy_overall_by_country=mortality_life_rdd.filter(array => array(2)<="2018")

      .map(array => (array(1),array(6).toDouble)).mapValues(value => (value,1))

      .reduceByKey { case ((sumL, countL), (sumR, countR)) => (sumL + sumR, countL + countR) }

      .mapValues { case(sum, count) => sum / count }.sortByKey(ascending=true)


    /* Birth rate Vs. Death rate by country */

    val birth_rate_by_country=birth_death_rdd.filter(array => array(2)<="2018")

      .map(array => (array(1),array(3).toDouble)).mapValues(value => (value,1))

      .reduceByKey { case ((sumL, countL), (sumR, countR)) => (sumL + sumR, countL + countR) }
      .mapValues { case(sum, count) => sum / count }.sortByKey(ascending=true)

    val death_rate_by_country=birth_death_rdd.filter(array => array(2)<="2018")
      .map(array => (array(1),array(4).toDouble)).mapValues(value => (value,1))
      .reduceByKey { case ((sumL, countL), (sumR, countR)) => (sumL + sumR, countL + countR) }
      .mapValues { case(sum, count) => sum / count }.sortByKey(ascending=true)

    val birth_death_rate_by_country =
    birth_rate_by_country.join(death_rate_by_country).sortByKey(ascending=true)

    /* Male/Female ratio at birth */
    val male_female_ratio_at_birth_by_country = age_fert_rdd.filter(array=>array(2)<="2018")
      .map(array => (array(1),array(12).toDouble)).mapValues(value => (value,1))
      .reduceByKey { case ((sumL, countL), (sumR, countR)) => (sumL + sumR, countL + countR) }
      .mapValues { case(sum, count) => sum / count }.sortByKey(ascending=true)

    /* Migration rate by country */
    val migration_rate_by_country = birth_death_rdd.filter(array => array(2)<="2018")
      .map(array => (array(1),array(5).toDouble)).mapValues(value => (value,1))
      .reduceByKey { case ((sumL, countL), (sumR, countR)) => (sumL + sumR, countL + countR) }
      .mapValues { case(sum, count) => sum / count }.sortByKey(ascending=true)

    /* Population balance (United States) */
    val indices = Array.range(5,106)

    val male_us_pop_by_year = midyear_pop_rdd
      .filter(array => array(1)=="United States" && array(3)=="Male" && array(2)<="2018")
      .map(array => (array(2).toInt,indices.map(array).map(_.toInt).sum))
```

```
    val female_us_pop_by_year = midyear_pop_rdd
      .filter(array => array(1)=="United States" && array(3)=="Female" && array(2)<="2018")
      .map(array => (array(2).toInt,indices.map(array).map(_.toInt).sum))

    /* Function to merge to CSV files */
    def merge(srcPath: String, dstPath: String): Unit =  {
      val hadoopConfig = new Configuration()
      val hdfs = FileSystem.get(hadoopConfig)
      FileUtil.copyMerge(hdfs, new Path(srcPath), hdfs, new Path(dstPath), true, hadoopConfig, null)
    }

    /* File1: Census Data Analysis */
    life_expectancy_overall_by_country
    .map{ case(x,y) => (x,(x,y))}
.join(birth_death_rate_by_country).join(male_female_ratio_at_birth_by_country).join(migration_rate_
by_country)
      .values.map{case(x,y)=> x._1._1._1 + "," + x._1._1._2 + "," + x._1._2._1 + "," + x._1._2._2 + "," + x._2
+ "," + y}
      .repartition(1).saveAsTextFile(args(1))

    /* File2: Population Balance */
    male_us_pop_by_year.map{ case(x,y) => (x,(x,y))}.join(female_us_pop_by_year).sortByKey()
      .values.map{case(x,y)=> "US" + "," + x._1 + "," + x._2 + "," + y}.repartition(1).saveAsTextFile(args(2))

    /* File3: Life Expectancy */
    life_expectancy_female_overall_by_year.map{case(x,y)=> (x,(x,y))
}.join(life_expectancy_male_overall_by_year)
```

```
      .values.sortBy(_._1).map{ case(x,y) => x._1 + "," + x._2 + "," + y + "," +
life_expectancy_female_overall + "," + life_expectancy_male_overall }

      /*.union("year" + "," + "life_expec_female" + "," + "life_expec_male" + "," +
"life_expectancy_female_overall" + "," + "life_expectancy_male_overall") */

      .repartition(1).saveAsTextFile(args(3))


    merge(args(1), args(1)+".csv")

    merge(args(2), args(2)+".csv")

    merge(args(3), args(3)+".csv")


    System.exit(0)

  }

}
```

Using Dataframes in spark-shell

```
spark-shell --packages com.databricks:spark-csv_2.10:1.5.0


/* age fertility rates */

   val age_fert = sqlContext.read.format("com.databricks.spark.csv").option("header",
"true").option("inferSchema", "true").load("hdfs:///user/dand9090/census-international-
data/age_specific_fertility_rates.csv")


 /* mortality life expectancy */

    val mortality_life = sqlContext.read.format("com.databricks.spark.csv").option("header",
"true").option("inferSchema", "true").load("hdfs:///user/dand9090/census-international-
data/mortality_life_expectancy.csv")


/* birth death growth rates */
```

```
   val birth_death = sqlContext.read.format("com.databricks.spark.csv").option("header",
"true").option("inferSchema", "true").load("hdfs:///user/dand9090/census-international-
data/birth_death_growth_rates.csv")


 /* mid year population */

   val midyear_pop = sqlContext.read.format("com.databricks.spark.csv").option("header",
"true").option("inferSchema", "true").load("hdfs:///user/dand9090/census-international-
data/midyear_population_age_sex.csv")


   /* Life expectancy by gender overall - as of 2018 */

   val life_expectancy_male_overall=
mortality_life.filter(mortality_life("year")<=2018).select(avg("life_expectancy_male")).show()

   val life_expectancy_female_overall=
mortality_life.filter(mortality_life("year")<=2018).select(avg("life_expectancy_female")).show()

  val life_expectancy_male_overall_by_year=
mortality_life.filter(mortality_life("year")<=2018).groupBy("year").agg(avg("life_expectancy_male")).sho
w()

   val life_expectancy_female_overall_by_year=
mortality_life.filter(mortality_life("year")<=2018).groupBy("year").agg(avg("life_expectancy_female")).s
how()


   /* Life expectancy by country, visualize high and low life expectancy - as of 2018 */

   val life_expectancy_overall_by_country=
mortality_life.filter(mortality_life("year")<=2018).groupBy("country_name").agg(avg("life_expectancy"))
.orderBy("country_name").show()


   /* Birth rate Vs. Death rate by country */

   val birth_rate_by_country=
birth_death.filter(birth_death("year")<=2018).groupBy("country_name").agg(avg("crude_birth_rate")).o
rderBy("country_name").show()
```

```
    val death_rate_by_country=
birth_death.filter(birth_death("year")<=2018).groupBy("country_name").agg(avg("crude_death_rate")).
orderBy("country_name").show()

    val birth_death_rate_by_country =
birth_rate_by_country.join(death_rate_by_country,"country_name").orderBy("country_name").show()


    /* Male/Female ratio at birth */

    val male_female_ratio_at_birth_by_country =
age_fert.filter(age_fert("year")<=2018).groupBy("country_name").agg(avg("sex_ratio_at_birth")).orderB
y("country_name").show()


    /* Migration rate by country */

    val migration_rate_by_country =
birth_death.filter(birth_death("year")<=2018).groupBy("country_name").agg(avg("net_migration")).ord
erBy("country_name").show()
```

Project Source Code:

https://github.com/satishkumard/SEIS736/tree/master/CensusDataAnalysis

Scala File:

https://github.com/satishkumard/SEIS736/blob/master/CensusDataAnalysis/src/main/scala/CensusDataAnalysis.scala

JAR File:

https://github.com/satishkumard/SEIS736/tree/master/CensusDataAnalysis/target/scala-2.11

Output Files:

https://github.com/satishkumard/SEIS736/tree/master/CensusDataAnalysis/output