**Project - Market Analysis**

**Source Code**

**Step 1:**

**Data cleaning:**

The data given in the cvs file was not readable with correct schema. So, a python script was created which cleans data and copies the data into another cvs file which makes the data readable with correct schema.

**Python Script:**

import csv

list1=[]

with open('/Users/kaushiksekar/Documents/Hadoop Files/Projects/BDHS\_Projects/Project for submission/Project 1/banking\_ma.csv') as csvfile:

#readCSV = csv.reader(csvfile, delimiter=';')

for row in csvfile:

list1.append(row.replace('"',""))

with open('/Users/kaushiksekar/Documents/Hadoop Files/Projects/BDHS\_Projects/Project for submission/Project 1/banking\_ma\_clean.csv','w') as writeFile:

for a in list1:

writeFile.write(a)

**Step 2:**

**Details:**

Import the data and query using spark-sql and data-frame and form RDDs and filter to answer questions asked. StatCounter class was also imported to find mean age,average balance.

Median balance was found using **df.stat().approxQuantile.** This calculates the approximate quantiles of a numerical column of a DataFrame.

The result of this algorithm has the following deterministic bound: If the DataFrame has N elements and if we request the quantile at probability p up to error err, then the algorithm will return a sample x from the DataFrame so that the \*exact\* rank of x is close to (p \* N). More precisely,

floor((p - err) \* N) <= rank(x) <= ceil((p + err) \* N).

**Program:**

package org.sparkcourse.realtimeproject.MarketingAnalysis;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

import org.apache.commons.lang.StringUtils;

import org.apache.spark.SparkConf;

import org.apache.spark.api.java.JavaDoubleRDD;

import org.apache.spark.api.java.JavaPairRDD;

import org.apache.spark.api.java.JavaRDD;

import org.apache.spark.api.java.JavaSparkContext;

import org.apache.spark.api.java.function.DoubleFunction;

import org.apache.spark.api.java.function.Function;

import org.apache.spark.api.java.function.PairFunction;

import org.apache.spark.sql.Dataset;

import org.apache.spark.sql.Row;

import org.apache.spark.sql.SparkSession;

import org.apache.spark.util.StatCounter;

import scala.Tuple2;

import org.apache.spark.sql.Encoders;

public class Analysis {

public static void main(String[] args) {

SparkConf sparkConf = new SparkConf().setMaster("local").setAppName("Marketing Analysis for a Banking company");

JavaSparkContext sc = new JavaSparkContext(sparkConf);

ArrayList<Double> listForCheckFunc=new ArrayList<Double>();

SparkSession sparkSession = SparkSession.builder().appName("Java Spark SQL ").getOrCreate();

Dataset<Row> df = sparkSession.read().format("com.databricks.spark.csv").option("header", "true")

.option("delimiter", ";").option("inferSchema", "true")

.load("/Users/kaushiksekar/Documents/Hadoop Files/Projects/BDHS\_Projects/Project for submission/Project 1/banking\_ma\_clean.csv");

df.createOrReplaceTempView("cases");

Dataset<Row> sqlDF1 = sparkSession.sql("select count(y) from cases where y='yes'");

List<Long> no\_success\_list=sqlDF1.as(Encoders.LONG()).collectAsList();

Dataset<Row> sqlDF2 = sparkSession.sql("select count(y) from cases");

List<Long> total\_no\_list = sqlDF2.as(Encoders.LONG()).collectAsList();

long no\_success=no\_success\_list.get(0);

long total\_no=total\_no\_list.get(0);

long no\_failure=total\_no-no\_success;

float success\_percentage = (no\_success/(float)total\_no)\*100;

float failure\_percentage = (no\_failure/(float)total\_no)\*100;

System.out.println("Total success - "+no\_success+"\nTotal no - "+total\_no+"\nPercentage of success - "+success\_percentage+"%");

System.out.println("Total failure - "+no\_failure+"\nPercentage of failure - "+failure\_percentage+"%");

Dataset<Row> sqlDF3 = sparkSession.sql("select age from cases");

JavaRDD<Row> ageRowRDD = sqlDF3.toJavaRDD();

JavaRDD<Integer> ageRDDTemp=ageRowRDD.map(

new Function<Row, Integer>() {

@Override

public Integer call(Row v1) throws Exception {

return Integer.parseInt(v1.toString().replace("[", "").replace("]", ""));

}

}

);

JavaDoubleRDD ageRDD=ageRDDTemp.mapToDouble(

new DoubleFunction<Integer>() {

@Override

public double call(Integer t) throws Exception {

return ((double)(t));

}

}

);

StatCounter statCounter=ageRDD.stats();

Double mean\_age=statCounter.mean();

Double max\_age=statCounter.max();

Double min\_age=statCounter.min();

sqlDF2=sparkSession.sql("select avg(balance) from cases");

Double avg\_balance = sqlDF2.as(Encoders.DOUBLE()).collectAsList().get(0);

double[] tmp= {0.5};

Double median\_balance = df.stat().approxQuantile("balance", tmp , 0)[0];

System.out.println("Max age - "+max\_age+"\nMin age - "+min\_age+"\nMean age - "+mean\_age);

System.out.println("Median Balance - "+ median\_balance +"\nAverage Balance - "+avg\_balance);

sqlDF2=sparkSession.sql("select age from cases where y='yes'");

sqlDF3=sparkSession.sql("select age from cases");

JavaRDD<Row> ageMattersTemp= sqlDF2.toJavaRDD();

JavaRDD<Row> ageTemp=sqlDF3.toJavaRDD();

JavaRDD<Integer> ageMatters=ageMattersTemp.map(

new Function<Row, Integer>() {

@Override

public Integer call(Row v1) throws Exception {

return Integer.parseInt(v1.toString().replace("[", "").replace("]", ""));

}

}

);

JavaRDD<Integer> age=ageTemp.map(

new Function<Row, Integer>() {

@Override

public Integer call(Row v1) throws Exception {

return Integer.parseInt(v1.toString().replace("[", "").replace("]", ""));

}

}

);

JavaRDD<Integer> earlyAgeGroupMatters=ageMatters.filter(

new Function<Integer, Boolean>() {

@Override

public Boolean call(Integer v1) throws Exception {

return v1>17&&v1<25;

}

}

);

JavaRDD<Integer> earlyAgeGroup=age.filter(

new Function<Integer, Boolean>() {

@Override

public Boolean call(Integer v1) throws Exception {

return v1>17&&v1<25;

}

}

);

System.out.println(earlyAgeGroupMatters.count()+" : "+earlyAgeGroup.count());

listForCheckFunc.add(earlyAgeGroupMatters.count()/(double)earlyAgeGroup.count());

JavaRDD<Integer> middleAgeGroupMatters=ageMatters.filter(

new Function<Integer, Boolean>() {

@Override

public Boolean call(Integer v1) throws Exception {

return v1>24&&v1<55;

}

}

);

JavaRDD<Integer> middleAgeGroup=age.filter(

new Function<Integer, Boolean>() {

@Override

public Boolean call(Integer v1) throws Exception {

return v1>24&&v1<55;

}

}

);

System.out.println(middleAgeGroupMatters.count()+" : "+middleAgeGroup.count());

listForCheckFunc.add(middleAgeGroupMatters.count()/(double)middleAgeGroup.count());

JavaRDD<Integer> seniorWorkersAgeGroupMatters=ageMatters.filter(

new Function<Integer, Boolean>() {

@Override

public Boolean call(Integer v1) throws Exception {

return v1>54&&v1<65;

}

}

);

JavaRDD<Integer> seniorWorkersAgeGroup=age.filter(

new Function<Integer, Boolean>() {

@Override

public Boolean call(Integer v1) throws Exception {

return v1>54&&v1<65;

}

}

);

System.out.println(seniorWorkersAgeGroupMatters.count()+" : "+seniorWorkersAgeGroup.count());

listForCheckFunc.add(seniorWorkersAgeGroupMatters.count()/(double)seniorWorkersAgeGroup.count());

JavaRDD<Integer> retiredAgeGroupMatters=ageMatters.filter(

new Function<Integer, Boolean>() {

@Override

public Boolean call(Integer v1) throws Exception {

return v1>64;

}

}

);

JavaRDD<Integer> retiredAgeGroup=age.filter(

new Function<Integer, Boolean>() {

@Override

public Boolean call(Integer v1) throws Exception {

return v1>64;

}

}

);

System.out.println(retiredAgeGroupMatters.count()+" : "+retiredAgeGroup.count());

listForCheckFunc.add(retiredAgeGroupMatters.count()/(double)retiredAgeGroup.count());

System.out.println("Does age matter in marketing subscription for deposit? : "+checkIfFeatureMattersRatio(listForCheckFunc));

listForCheckFunc=new ArrayList<Double>();

sqlDF2=sparkSession.sql("select marital from cases where y='yes'");

sqlDF3=sparkSession.sql("select marital from cases");

JavaRDD<Row> maritalTemp= sqlDF2.toJavaRDD();

JavaRDD<Row> totalTemp=sqlDF3.toJavaRDD();

JavaRDD<String> maritalSubscribed=maritalTemp.map(

new Function<Row, String>() {

@Override

public String call(Row v1) throws Exception {

return v1.toString().replace("[", "").replace("]", "");

}

}

);

JavaRDD<String> total=totalTemp.map(

new Function<Row, String>() {

@Override

public String call(Row v1) throws Exception {

return v1.toString().replace("[", "").replace("]", "");

}

}

);

JavaRDD<String> singleSubscribed=maritalSubscribed.filter(

new Function<String, Boolean>() {

@Override

public Boolean call(String v1) throws Exception {

return v1.equals("single");

}

}

);

JavaRDD<String> singleTotal=total.filter(

new Function<String, Boolean>() {

@Override

public Boolean call(String v1) throws Exception {

return v1.equals("single");

}

}

);

System.out.println(singleSubscribed.count()+ " : "+singleTotal.count());

listForCheckFunc.add(singleSubscribed.count()/(double)singleTotal.count());

JavaRDD<String> marriedSubscribed=maritalSubscribed.filter(

new Function<String, Boolean>() {

@Override

public Boolean call(String v1) throws Exception {

return v1.equals("married");

}

}

);

JavaRDD<String> marriedTotal=total.filter(

new Function<String, Boolean>() {

@Override

public Boolean call(String v1) throws Exception {

return v1.equals("married");

}

}

);

System.out.println(marriedSubscribed.count()+ " : "+marriedTotal.count());

listForCheckFunc.add(marriedSubscribed.count()/(double)marriedTotal.count());

JavaRDD<String> divorcedSubscribed=maritalSubscribed.filter(

new Function<String, Boolean>() {

@Override

public Boolean call(String v1) throws Exception {

return v1.equals("divorced");

}

}

);

JavaRDD<String> divorcedTotal=total.filter(

new Function<String, Boolean>() {

@Override

public Boolean call(String v1) throws Exception {

return v1.equals("divorced");

}

}

);

System.out.println(divorcedSubscribed.count()+ " : "+divorcedTotal.count());

listForCheckFunc.add(divorcedSubscribed.count()/(double)divorcedTotal.count());

System.out.println("Does marital status matter in marketing subscription for deposit? : "+checkIfFeatureMattersRatio(listForCheckFunc));

sqlDF2=sparkSession.sql("select age,marital from cases where y='yes'");

sqlDF3=sparkSession.sql("select age,marital from cases");

JavaRDD<Row> ageMaritalSubscribedTemp=sqlDF2.toJavaRDD();

JavaRDD<Row> ageMaritalTotalTemp=sqlDF3.toJavaRDD();

JavaPairRDD<Integer, String> ageMaritalSubscribedRDD=ageMaritalSubscribedTemp.map(

new Function<Row, String>() {

@Override

public String call(Row v1) throws Exception {

return v1.toString();

}

}

).mapToPair(

new PairFunction<String, Integer, String>() {

@Override

public Tuple2<Integer, String> call(String t) throws Exception {

return new Tuple2<Integer, String>(Integer.parseInt(t.split(",")[0].replace("[", "").replace("]", "")), t.split(",")[1].replace("[", "").replace("]", ""));

}

}

);

JavaPairRDD<Integer, String> ageMaritalTotalRDD=ageMaritalTotalTemp.map(

new Function<Row, String>() {

@Override

public String call(Row v1) throws Exception {

return v1.toString();

}

}

).mapToPair(

new PairFunction<String, Integer, String>() {

@Override

public Tuple2<Integer, String> call(String t) throws Exception {

return new Tuple2<Integer, String>(Integer.parseInt(t.split(",")[0].replace("[", "").replace("]", "")), t.split(",")[1].replace("[", "").replace("]", ""));

}

}

);

long earlyAgeDivorcedSubscribedCount=ageMaritalSubscribedRDD.filter(

new Function<Tuple2<Integer,String>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, String> v1) throws Exception {

return v1.\_1>17&&v1.\_1<25&&v1.\_2.equals("divorced");

}

}

).count();

long earlyAgeDivorcedTotalCount=ageMaritalTotalRDD.filter(

new Function<Tuple2<Integer,String>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, String> v1) throws Exception {

return v1.\_1>17&&v1.\_1<25&&v1.\_2.equals("divorced");

}

}

).count();

long earlyAgeMarriedSubscribedCount=ageMaritalSubscribedRDD.filter(

new Function<Tuple2<Integer,String>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, String> v1) throws Exception {

return v1.\_1>17&&v1.\_1<25&&v1.\_2.equals("married");

}

}

).count();

long earlyAgeMarriedTotalCount=ageMaritalTotalRDD.filter(

new Function<Tuple2<Integer,String>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, String> v1) throws Exception {

return v1.\_1>17&&v1.\_1<25&&v1.\_2.equals("married");

}

}

).count();

long earlyAgeSingleSubscribedCount=ageMaritalSubscribedRDD.filter(

new Function<Tuple2<Integer,String>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, String> v1) throws Exception {

return v1.\_1>17&&v1.\_1<25&&v1.\_2.equals("single");

}

}

).count();

long earlyAgeSingleTotalCount=ageMaritalTotalRDD.filter(

new Function<Tuple2<Integer,String>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, String> v1) throws Exception {

return v1.\_1>17&&v1.\_1<25&&v1.\_2.equals("single");

}

}

).count();

long middleAgeDivorcedSubscribedCount=ageMaritalSubscribedRDD.filter(new Function<Tuple2<Integer,String>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, String> v1) throws Exception {

return v1.\_1>24&&v1.\_1<55&&v1.\_2.equals("divorced");

}

}).count();

long middleAgeDivorcedTotalCount=ageMaritalTotalRDD.filter(

new Function<Tuple2<Integer,String>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, String> v1) throws Exception {

return v1.\_1>24&&v1.\_1<55&&v1.\_2.equals("divorced");

}

}

).count();

long middleAgeMarriedSubscribedCount=ageMaritalSubscribedRDD.filter(

new Function<Tuple2<Integer,String>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, String> v1) throws Exception {

return v1.\_1>24&&v1.\_1<55&&v1.\_2.equals("married");

}

}

).count();

long middleAgeMarriedTotalCount=ageMaritalTotalRDD.filter(

new Function<Tuple2<Integer,String>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, String> v1) throws Exception {

return v1.\_1>24&&v1.\_1<55&&v1.\_2.equals("married");

}

}

).count();

long middleAgeSingleSubscribedCount=ageMaritalSubscribedRDD.filter(

new Function<Tuple2<Integer,String>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, String> v1) throws Exception {

return v1.\_1>24&&v1.\_1<55&&v1.\_2.equals("single");

}

}

).count();

long middleAgeSingleTotalCount=ageMaritalTotalRDD.filter(

new Function<Tuple2<Integer,String>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, String> v1) throws Exception {

return v1.\_1>24&&v1.\_1<55&&v1.\_2.equals("single");

}

}

).count();

long seniorAgeDivorcedSubscribedCount=ageMaritalSubscribedRDD.filter(

new Function<Tuple2<Integer,String>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, String> v1) throws Exception {

return v1.\_1>54&&v1.\_1<65&&v1.\_2.equals("divorced");

}

}

).count();

long seniorAgeDivorcedTotalCount=ageMaritalTotalRDD.filter(

new Function<Tuple2<Integer,String>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, String> v1) throws Exception {

return v1.\_1>54&&v1.\_1<65&&v1.\_2.equals("divorced");

}

}

).count();

long seniorAgeMarriedSubscribedCount=ageMaritalSubscribedRDD.filter(

new Function<Tuple2<Integer,String>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, String> v1) throws Exception {

return v1.\_1>54&&v1.\_1<65&&v1.\_2.equals("married");

}

}

).count();

long seniorAgeMarriedTotalCount=ageMaritalTotalRDD.filter(

new Function<Tuple2<Integer,String>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, String> v1) throws Exception {

return v1.\_1>54&&v1.\_1<65&&v1.\_2.equals("married");

}

}

).count();

long seniorAgeSingleSubscribedCount=ageMaritalSubscribedRDD.filter(

new Function<Tuple2<Integer,String>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, String> v1) throws Exception {

return v1.\_1>54&&v1.\_1<65&&v1.\_2.equals("single");

}

}

).count();

long seniorAgeSingleTotalCount=ageMaritalTotalRDD.filter(

new Function<Tuple2<Integer,String>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, String> v1) throws Exception {

return v1.\_1>54&&v1.\_1<65&&v1.\_2.equals("single");

}

}

).count();

long retiredAgeDivorcedSubscribedCount=ageMaritalSubscribedRDD.filter(

new Function<Tuple2<Integer,String>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, String> v1) throws Exception {

return v1.\_1>64&&v1.\_2.equals("divorced");

}

}

).count();

long retiredAgeDivorcedTotalCount=ageMaritalTotalRDD.filter(

new Function<Tuple2<Integer,String>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, String> v1) throws Exception {

return v1.\_1>64&&v1.\_2.equals("divorced");

}

}

).count();

long retiredAgeMarriedSubscribedCount=ageMaritalSubscribedRDD.filter(

new Function<Tuple2<Integer,String>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, String> v1) throws Exception {

return v1.\_1>64&&v1.\_2.equals("married");

}

}

).count();

long retiredAgeMarriedTotalCount=ageMaritalTotalRDD.filter(

new Function<Tuple2<Integer,String>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, String> v1) throws Exception {

return v1.\_1>64&&v1.\_2.equals("married");

}

}

).count();

long retiredAgeSingleSubscribedCount=ageMaritalSubscribedRDD.filter(

new Function<Tuple2<Integer,String>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, String> v1) throws Exception {

return v1.\_1>64&&v1.\_2.equals("single");

}

}

).count();

long retiredAgeSingleTotalCount=ageMaritalTotalRDD.filter(

new Function<Tuple2<Integer,String>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, String> v1) throws Exception {

return v1.\_1>64&&v1.\_2.equals("single");

}

}

).count();

listForCheckFunc=new ArrayList<>();

listForCheckFunc.add(earlyAgeDivorcedSubscribedCount/(double)earlyAgeDivorcedTotalCount);

listForCheckFunc.add(earlyAgeMarriedSubscribedCount/(double)earlyAgeMarriedTotalCount);

listForCheckFunc.add(earlyAgeSingleSubscribedCount/(double)earlyAgeSingleTotalCount);

listForCheckFunc.add(middleAgeDivorcedSubscribedCount/(double)middleAgeDivorcedTotalCount);

listForCheckFunc.add(middleAgeMarriedSubscribedCount/(double)middleAgeMarriedTotalCount);

listForCheckFunc.add(middleAgeSingleSubscribedCount/(double)middleAgeSingleTotalCount);

listForCheckFunc.add(seniorAgeDivorcedSubscribedCount/(double)seniorAgeDivorcedTotalCount);

listForCheckFunc.add(seniorAgeMarriedSubscribedCount/(double)seniorAgeMarriedTotalCount);

listForCheckFunc.add(seniorAgeSingleSubscribedCount/(double)seniorAgeSingleTotalCount);

listForCheckFunc.add(retiredAgeDivorcedSubscribedCount/(double)retiredAgeDivorcedTotalCount);

listForCheckFunc.add(retiredAgeMarriedSubscribedCount/(double)retiredAgeMarriedTotalCount);

listForCheckFunc.add(retiredAgeSingleSubscribedCount/(double)retiredAgeSingleTotalCount);

System.out.println("Do marital status and age matter together in marketing subscription for deposit? : "+checkIfFeatureMattersRatio(listForCheckFunc));

sqlDF3=sparkSession.sql("select age,campaign from cases where y='yes'");

JavaRDD<Row> ageCampaignSubscribed= sqlDF3.toJavaRDD();

JavaPairRDD<Integer,Integer> ageCampaignRDD=ageCampaignSubscribed.map(

new Function<Row, String>() {

@Override

public String call(Row v1) throws Exception {

return v1.toString();

}

}

).mapToPair(

new PairFunction<String, Integer, Integer>() {

@Override

public Tuple2<Integer, Integer> call(String t) throws Exception {

return new Tuple2<Integer,Integer>(Integer.parseInt(t.split(",")[0].replace("[", "").replace("]", "")), Integer.parseInt(t.split(",")[1].replace("[", "").replace("]", "")));

}

}

);

long earlyAgeCampaignSubscribedCount=ageCampaignRDD.filter(

new Function<Tuple2<Integer,Integer>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, Integer> v1) throws Exception {

return (v1.\_1>17&&v1.\_1<25&&v1.\_2==1);

}

}

).count();

long middleAgeCampaignSubscribedCount=ageCampaignRDD.filter(

new Function<Tuple2<Integer,Integer>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, Integer> v1) throws Exception {

return (v1.\_1>24&&v1.\_1<55&&v1.\_2==1);

}

}

).count();

long seniorAgeCampaignSubscribedCount=ageCampaignRDD.filter(

new Function<Tuple2<Integer,Integer>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, Integer> v1) throws Exception {

return (v1.\_1>54&&v1.\_1<65&&v1.\_2==1);

}

}

).count();

long retiredAgeCampaignSubscribedCount=ageCampaignRDD.filter(

new Function<Tuple2<Integer,Integer>, Boolean>() {

@Override

public Boolean call(Tuple2<Integer, Integer> v1) throws Exception {

return (v1.\_1>64&&v1.\_2==1);

}

}

).count();

String result=checkIfFeatureMattersCount(earlyAgeCampaignSubscribedCount,middleAgeCampaignSubscribedCount,seniorAgeCampaignSubscribedCount, retiredAgeCampaignSubscribedCount);

System.out.println("Right age effect on campaign is "+result);

sc.close();

}

public static String checkIfFeatureMattersRatio(ArrayList<Double> a) {

double difference=Collections.max(a)-Collections.min(a);

if(difference>0.05)

return "Yes";

return "No";

}

public static String checkIfFeatureMattersCount(long a,long b,long c,long d) {

long max=Math.max(a, Math.max(b, Math.max(c, d)));

if(max==a)

return "Early Age Campaign : 18-24";

else if(max==b)

return "Middle Age Campaign : 25-54";

else if(max==c)

return "Senior Age Campaign : 55-64";

else

return "Retired Age Campaign : >65";

}

}