BANKING ANALYSIS

Team Members:-

Mashir Nizami
Anam Tamboli
Piyush Bodhani
Vrishali More
Partheev Boora

==========

Spark Streaming:

1.Load data and create Spark data frame

Ans:

Step1)Import packages:

```
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types.{IntegerType, StringType, StructField, StructType}
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._
```

```
scala> import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.SparkSession

scala> import org.apache.spark.sql.types.{IntegerType, StringType, StructField, StructType}
import org.apache.spark.sql.types.{IntegerType, StringType, StructField, StructType}

scala> import org.apache.spark.sql.types._
import org.apache.spark.sql.types._

scala> import org.apache.spark.sql.functions._
import org.apache.spark.sql.functions._
```

Step2) Create Schema:

```
val schema = StructType(
    List(
      StructField("age", IntegerType, true),
      StructField("job", StringType, true),
      StructField("marital", StringType, true),
      StructField("education", StringType, true),
      StructField("default", StringType, true),
      StructField("balance", IntegerType, true),
      StructField("housing", StringType, true),
```

```
    StructField("loan", StringType, true),
    StructField("contact", StringType, true),
    StructField("day", IntegerType, true),
    StructField("month", StringType, true),
    StructField("duration", IntegerType, true),
    StructField("campaign", IntegerType, true),
    StructField("pdays", IntegerType, true),
    StructField("previous", IntegerType, true),
    StructField("poutcome", StringType, true),
    StructField("y", StringType, true)
))
```

```
scala> val schema = StructType(List(StructField("age", IntegerType, true),StructField("job", StringType, true),StructField("marital", String
Type, true),StructField("education", StringType, true),StructField("default", StringType, true),StructField("balance", IntegerType, true),St
ructField("housing", StringType, true),StructField("loan", StringType, true),StructField("contact", StringType, true),StructField("day", Int
egerType, true),StructField("month", StringType, true),StructField("duration", IntegerType, true),StructField("campaign", IntegerType, true)
,StructField("pdays", IntegerType, true),StructField("previous", IntegerType, true),StructField("poutcome", StringType, true),StructField("y
", StringType, true)))
schema: org.apache.spark.sql.types.StructType = StructType(StructField(age,IntegerType,true), StructField(job,StringType,true), StructField(
marital,StringType,true), StructField(education,StringType,true), StructField(default,StringType,true), StructField(balance,IntegerType,true
), StructField(housing,StringType,true), StructField(loan,StringType,true), StructField(contact,StringType,true), StructField(day,IntegerTyp
e,true), StructField(month,StringType,true), StructField(duration,IntegerType,true), StructField(campaign,IntegerType,true), StructField(pda
ys,IntegerType,true), StructField(previous,IntegerType,true), StructField(poutcome,StringType,true), StructField(y,StringType,true))
```

Step3) create a dataframe: and display schema

val df =
spark.readStream.schema(schema).option("delimeter",",").csv("/user/maria_dev/finalProject/*")
df.printSchema()

```
scala> val df = spark.readStream.schema(schema).option("delimeter",",").csv("/user/maria_dev/finalProject/*")
df: org.apache.spark.sql.DataFrame = [age: int, job: string ... 15 more fields]

scala> df.printSchema()
root
 |-- age: integer (nullable = true)
 |-- job: string (nullable = true)
 |-- marital: string (nullable = true)
 |-- education: string (nullable = true)
 |-- default: string (nullable = true)
 |-- balance: integer (nullable = true)
 |-- housing: string (nullable = true)
 |-- loan: string (nullable = true)
 |-- contact: string (nullable = true)
 |-- day: integer (nullable = true)
 |-- month: string (nullable = true)
 |-- duration: integer (nullable = true)
 |-- campaign: integer (nullable = true)
 |-- pdays: integer (nullable = true)
 |-- previous: integer (nullable = true)
 |-- poutcome: string (nullable = true)
 |-- y: string (nullable = true)
```

2. A. Give marketing success rate. (No. of people subscribed / total no. of entries)

B. Give marketing failure rate

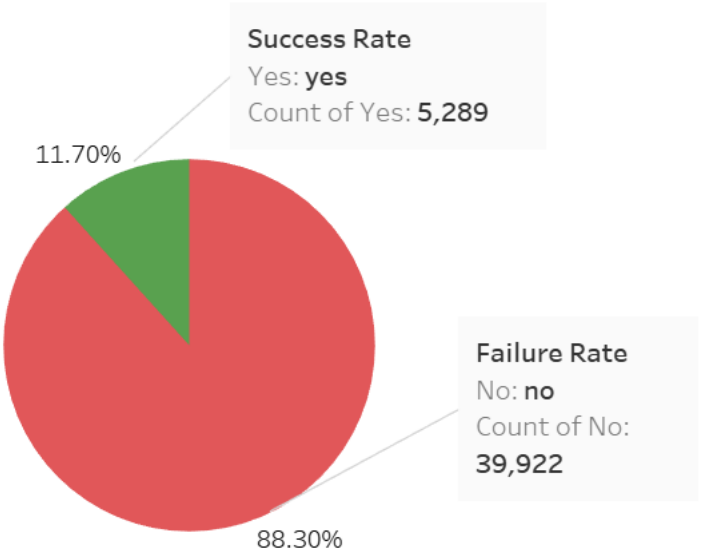
Ans:

val success =spark.sql("Select count(case y when 'yes' then 1 end)/count(*)*100 as success_rate from bankdata")
success.writeStream.outputMode("update").option("truncate",false).format("console").start().awaitTermination(10)

```
scala> val success=spark.sql("Select count(case y when 'yes' then 1 end)/count(*)*100 as success_rate from bankdata")
success: org.apache.spark.sql.DataFrame = [success_rate: double]

scala> success.writeStream.format("console").outputMode("update").start();
res2: org.apache.spark.sql.streaming.StreamingQuery = org.apache.spark.sql.execution.streaming.StreamingQueryWrapper@77b6d94c

scala> -------------------------------------------
Batch: 0
-------------------------------------------
+------------------+
|      success_rate|
+------------------+
|11.657482918227904|
+------------------+
```

val failure=spark.sql("Select count(case y when 'no' then 1 end)/count(*)*100 as failure_rate from bankdata")
failure.writeStream.outputMode("update").option("truncate",false).format("console").start().awaitTermination(10)

```
scala> val failure =spark.sql("Select count(case y when 'no' then 1 end)/count(*)*100 as failure_rate from bankdata")
failure: org.apache.spark.sql.DataFrame = [failure_rate: double]

scala> failure.writeStream.format("console").outputMode("update").start();
res3: org.apache.spark.sql.streaming.StreamingQuery = org.apache.spark.sql.execution.streaming.StreamingQueryWrapper@28446b06

scala> -------------------------------------------
Batch: 0
-------------------------------------------
+-----------------+
|     failure_rate|
+-----------------+
|87.99206524134891|
+-----------------+
```

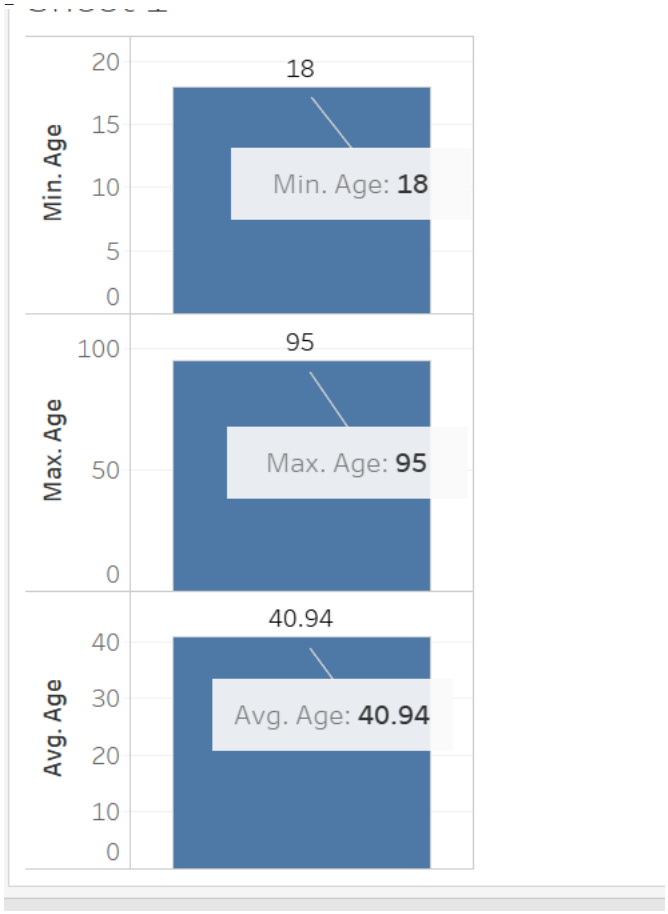


3. Maximum, Mean, and Minimum age of average targeted customer

Ans:

val query="""select max(age),min(age),avg(age) from bankdata""" val targetedCustomer = spark.sql(query)
targetedCustomer.writeStream.outputMode("update").option("truncate",false).format("console").start().awaitTermination(10)

```
scala> val query="""select max(age),min(age),avg(age) from bankdata"""
query: String = select max(age),min(age),avg(age) from bankdata

scala> val targetedCustomer = spark.sql(query)
targetedCustomer: org.apache.spark.sql.DataFrame = [max(age): int, min(age): int ... 1 more field]

scala>

scala> targetedCustomer.writeStream.outputMode("update").option("truncate",false).format("console").start().awaitTermination(10)
res1: Boolean = false

-------------------------------------------
Batch: 0
-------------------------------------------
+--------+--------+-----------------+
|max(age)|min(age)|avg(age)         |
+--------+--------+-----------------+
|95      |18      |40.93621021432837|
+--------+--------+-----------------+
```

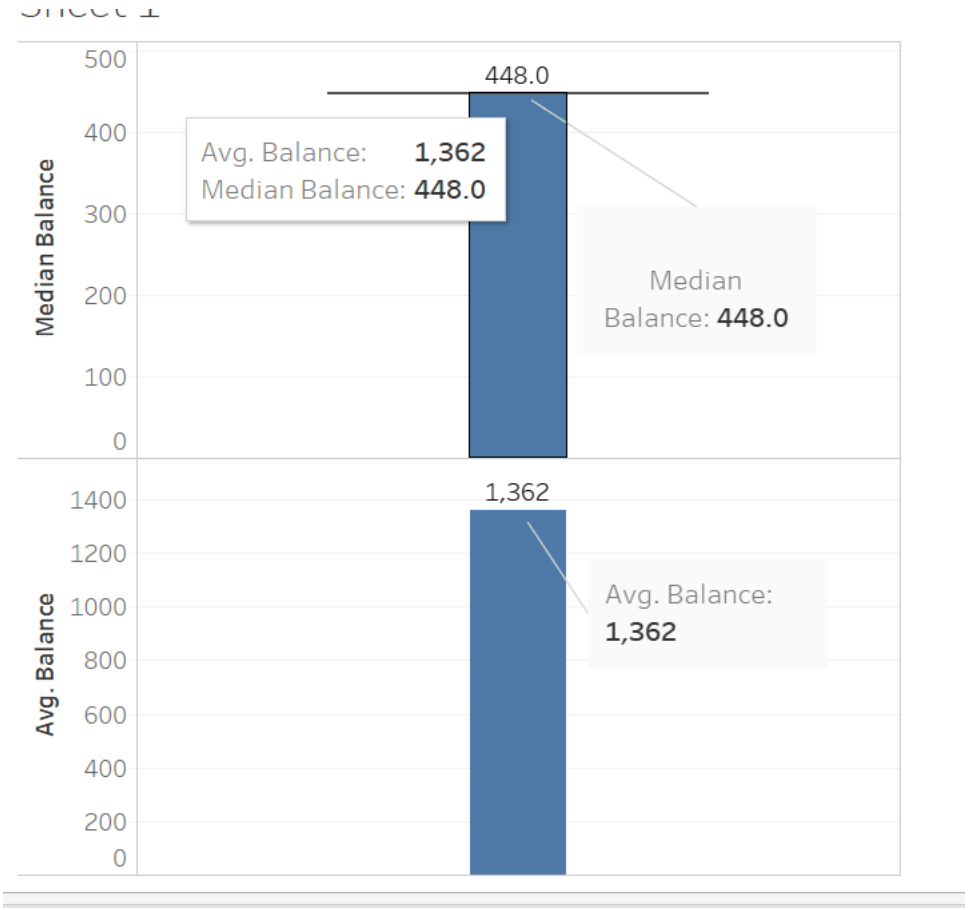


4. Check quality of customers by checking average balance, median balance of customers

Ans:

val query="""select percentile_approx(balance, 0.5),avg(balance) from bankdata"""
val customer= spark.sql(query)
customer.writeStream.outputMode("update").option("truncate",false).format("console").start().await Termination(10)

```
scala> val query="""select percentile_approx(balance, 0.5),avg(balance) from bankdata"""
query: String = select percentile_approx(balance, 0.5),avg(balance) from bankdata

scala> val customer = spark.sql(query)
customer: org.apache.spark.sql.DataFrame = [percentile_approx(balance, CAST(0.5 AS DOUBLE), 10000): int, avg(balance): double]

scala> customer.writeStream.outputMode("update").option("truncate", false).format("console").start().awaitTermination(10)
res11: Boolean = false

scala> -------------------------------------------
Batch: 0
-------------------------------------------
+-----------------------------------------------------+------------------+
|percentile_approx(balance, CAST(0.5 AS DOUBLE), 10000)|avg(balance)      |
+-----------------------------------------------------+------------------+
|448                                                  |1362.2720576850766|
+-----------------------------------------------------+------------------+
```



5.Check if age matters in marketing subscription for deposit

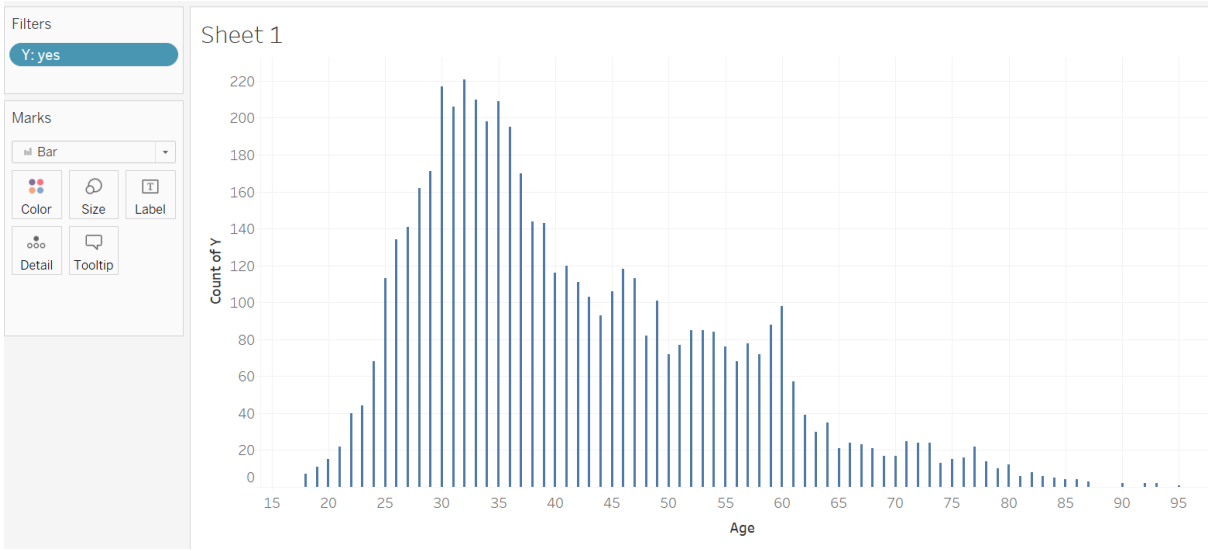
Ans:

val query ="""select age,y as subscription_status from bankdata group by age,y """

val ageMatters = spark.sql(query)

ageMatters.writeStream.outputMode("update").option("truncate", false).format("console").start().awaitTermination(10)

```
scala> val query ="""select age,y as subscription_status from bankdata group by age,y """
query: String = "select age,y as subscription_status from bankdata group by age,y "

scala> val ageMatters = spark.sql(query)
ageMatters: org.apache.spark.sql.DataFrame = [age: int, subscription_status: string]

scala> ageMatters.writeStream.outputMode("update").option("truncate", false).format("console").start().awaitTermination(10)
res25: Boolean = false

-------------------------------------------
Batch: 0
-------------------------------------------
+---+-------------------+
|age|subscription_status|
+---+-------------------+
|20 |no                 |
|78 |no                 |
|56 |yes                |
|28 |yes                |
|29 |yes                |
|71 |no                 |
|86 |yes                |
|57 |no                 |
|79 |yes                |
|22 |yes                |
|42 |no                 |
|31 |yes                |
|59 |yes                |
|87 |yes                |
|25 |no                 |
```

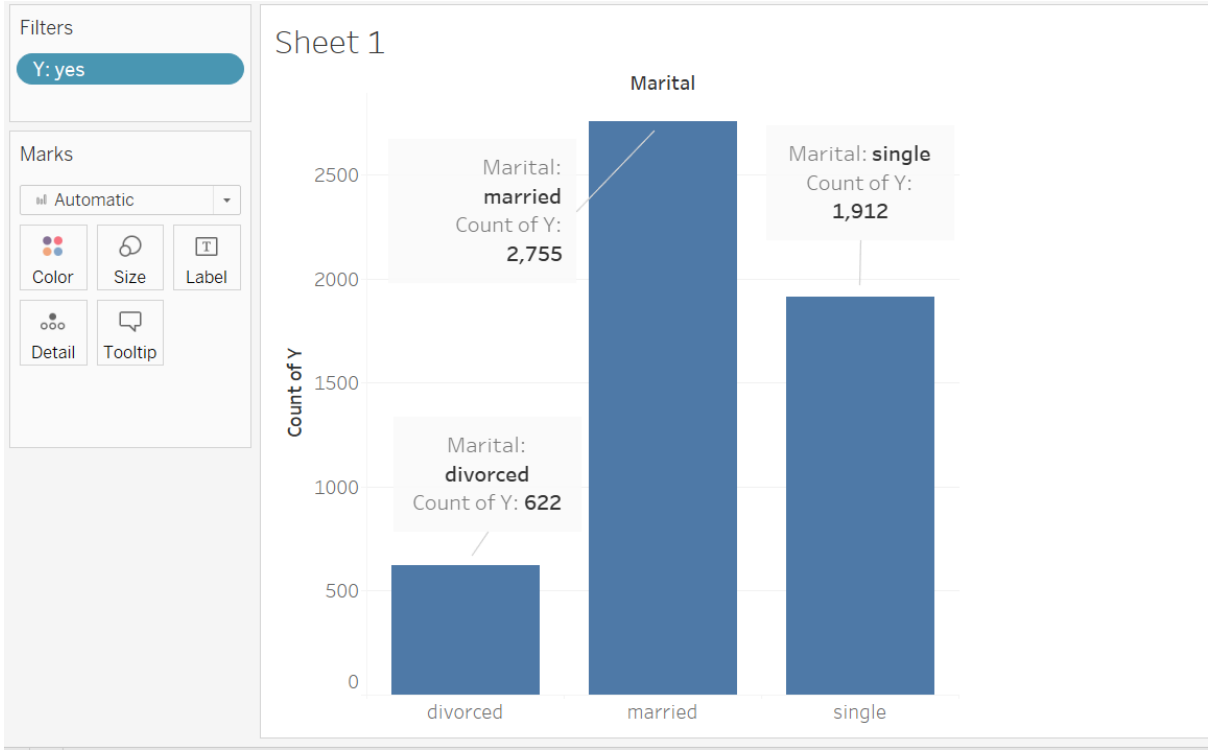


6) Check if marital status mattered for subscription to deposit

Ans:

val query ="""select marital,y as subscription_status from bankdata group by marital,y"""
val status = spark.sql(query)
status.writeStream.outputMode("update").option("truncate",
false).format("console").start().awaitTermination(10)

```
scala> val query ="""select marital,y as subscription_status from bankdata group by marital,y"""
query: String = select marital,y as subscription_status from bankdata group by marital,y

scala> val status = spark.sql(query)
status: org.apache.spark.sql.DataFrame = [marital: string, subscription_status: string]

scala> status.writeStream.outputMode("update").option("truncate", false).format("console").start().awaitTermination(10)
res26: Boolean = false

-------------------------------------------
Batch: 0
-------------------------------------------
+--------+-------------------+
|marital |subscription_status|
+--------+-------------------+
|divorced|yes                |
|single  |no                 |
|single  |yes                |
|divorced|no                 |
|married |yes                |
|married |no                 |
+--------+-------------------+
```



7. Check if age and marital status together mattered for subscription to deposit scheme

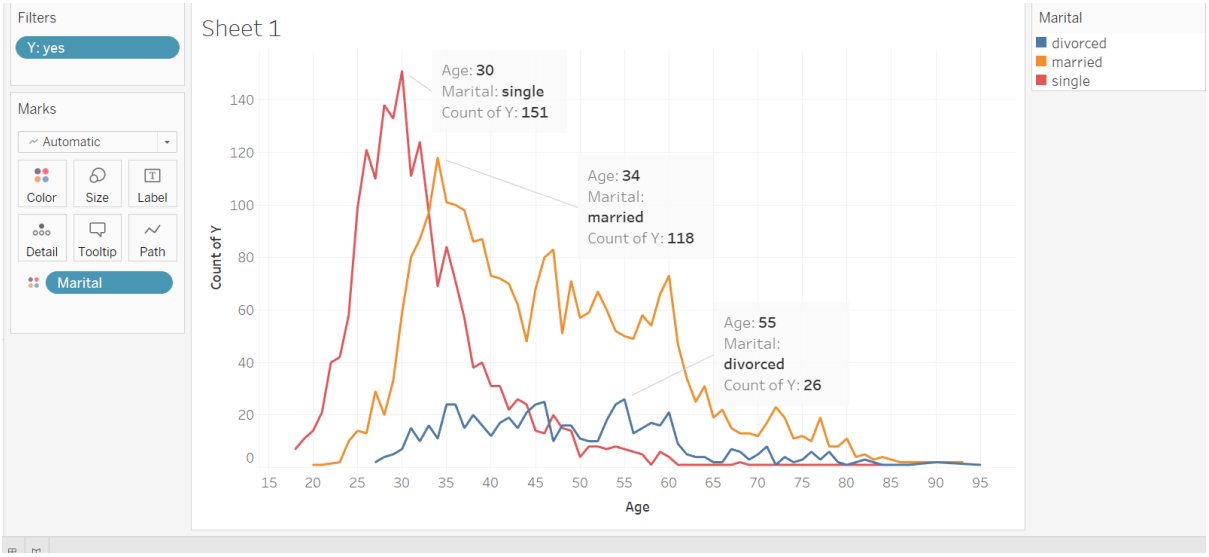
Ans:

val query ="""select age,marital,y as subscription_status from bankdata group by marital,y,age"""

val together= spark.sql(query)

together.writeStream.outputMode("update").option("truncate",false).format("console").start().await Termination(10)

```
scala> val query ="""select age,marital,y as subscription_status from bankdata group by marital,y,age"""
query: String = select age,marital,y as subscription_status from bankdata group by marital,y,age

scala> val together= spark.sql(query)
together: org.apache.spark.sql.DataFrame = [age: int, marital: string ... 1 more field]

scala>

scala> together.writeStream.outputMode("update").option("truncate",false).format("console").start().awaitTermination(10)
res27: Boolean = false

-------------------------------------------
Batch: 0
-------------------------------------------
+---+--------+------------------+
|age|marital |subscription_status|
+---+--------+------------------+
|37 |single  |yes               |
|92 |married |yes               |
|24 |married |no                |
|59 |divorced|no                |
|49 |married |no                |
|29 |divorced|yes               |
|59 |married |yes               |
|46 |single  |yes               |
|87 |divorced|yes               |
|78 |married |no                |
|23 |married |no                |
|32 |divorced|no                |
|44 |married |yes               |
|35 |divorced|no                |
```



WAYS TO DO ANALYSIS USING SPARK:

1) Using java Streaming:

Steps:

1)Create maven project in sprint tools suits.

```
package com.example.bank;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.SparkSession;
import org.apache.spark.sql.streaming.OutputMode;
import org.apache.spark.sql.streaming.StreamingQuery;
import org.apache.spark.sql.types.DataTypes;
import org.apache.spark.sql.types.StructType;


public class BankApplication {

	public static void main(String[] args) throws Exception {

	SparkSession spark = SparkSession.builder().appName("spark streaming").config("spark.master",
"local").config("spark.sql.warehouse.dir", "file:///apps/").getOrCreate();


	spark.sparkContext().setLogLevel("ERROR");

	StructType schema = new StructType().add("age", DataTypes.IntegerType)
				.add("job", DataTypes.StringType)
				.add("marital", DataTypes.StringType)
				.add("education", DataTypes.StringType)
				.add("default", DataTypes.StringType)
				.add("balance", DataTypes.IntegerType)
				.add("housing", DataTypes.StringType)
				.add("loan", DataTypes.StringType)
				.add("contact", DataTypes.StringType)
				.add("day", DataTypes.IntegerType)
				.add("month", DataTypes.StringType)
				.add("duration", DataTypes.IntegerType)
				.add("campaign", DataTypes.IntegerType)
				.add("pdays", DataTypes.IntegerType)
				.add("previous", DataTypes.IntegerType)
				.add("poutcome", DataTypes.StringType)
				.add("y", DataTypes.StringType);

	Dataset<Row> rawData =
spark.readStream().option("header","false").option("delimiter",";").format("csv").schema(schema).csv("/us
er/FinalProject/*");

	rawData.createOrReplaceTempView("bankdata");
```

```
        Dataset<Row> success = spark.sql("select max(age),min(age),avg(age) from bankdata");

        StreamingQuery query =
success.writeStream().outputMode(OutputMode.Update()).format("console").start();

        query.awaitTermination();

                }
}
```

2)Using ambari import jar file in hdp:

➔spark-submit --class "com.example.bank.BankApplication" --master local ./data.jar

```
22/12/02 09:20:44 INFO BlockManagerMasterEndpoint: Registering block manager sandbox-hdp.hortonworks.com:36347 with 366.3 MB RAM, BlockManag
erId(driver, sandbox-hdp.hortonworks.com, 36347, None)
22/12/02 09:20:44 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, sandbox-hdp.hortonworks.com, 36347, None)
22/12/02 09:20:44 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, sandbox-hdp.hortonworks.com, 36347, None)
22/12/02 09:20:44 INFO ContextHandler: Started o.s.j.s.ServletContextHandler@29314cc9{/metrics/json,null,AVAILABLE,@Spark}
22/12/02 09:20:47 INFO EventLoggingListener: Logging events to hdfs:/spark2-history/local-1669972844240
-------------------------------------------
Batch: 0
-------------------------------------------
+--------+--------+-----------------+
|max(age)|min(age)|         avg(age)|
+--------+--------+-----------------+
|      95|      18|40.93621021432837|
+--------+--------+-----------------+
```

2) Using Scala script file:

Steps:

1) Create a script file:
   vi project.scala

```
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types.{IntegerType, StringType, StructField, StructType}
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._

object finalProject{
  def main(args: Array[String]) {

val sqlContext = new org.apache.spark.sql.SQLContext(sc);
   import sqlContext.implicits._

val schema = StructType(
     List(
      StructField("age", IntegerType, true),
      StructField("job", StringType, true),
      StructField("marital", StringType, true),
      StructField("education", StringType, true),
      StructField("default", StringType, true),
      StructField("balance", IntegerType, true),
      StructField("housing", StringType, true),
      StructField("loan", StringType, true),
      StructField("contact", StringType, true),
      StructField("day", IntegerType, true),
      StructField("month", StringType, true),
      StructField("duration", IntegerType, true),
      StructField("campaign", IntegerType, true),
      StructField("pdays", IntegerType, true),
      StructField("previous", IntegerType, true),
      StructField("poutcome", StringType, true),
      StructField("y", StringType, true) ))

val df =
spark.readStream.schema(schema).option("delimiter",";").csv("/user/maria_dev/FinalProject/
*")
   df.printSchema()

df.createOrReplaceTempView("bankdata")

print("Give marketing success rate:")
```

```
val success =spark.sql("Select count(case y when 'yes' then 1 end)/count(*)*100 as
success_rate from bankdata")
success.writeStream.outputMode("update").option("truncate",false).format("console").start()
.awaitTermination(10)

print("Give marketing failuer rate:")
val fail=spark.sql("Select count(case y when 'no' then 1 end)/count(*)*100 as failure_rate
from bankdata")
fail.writeStream.outputMode("update").option("truncate",false).format("console").start().aw
aitTermination(10)

print("Giving the max min and avg age :")
val age="""select max(age),min(age),avg(age) from bankdata"""
val bage = spark.sql(age)
bage.writeStream.outputMode("update").option("truncate",false).format("console").start().a
waitTermination(10)


print("quality of customers by checking average balance, median balance of customers :")
val avg="""select percentile_approx(balance, 0.5),avg(balance) from bankdata"""
val customer= spark.sql(avg)
customer.writeStream.outputMode("update").option("truncate",false).format("console").star
t().awaitTermination(10)

print("age matters in marketing subscription for deposit :")
val q5 ="""select age,y as subscription_status from bankdata group by age,y """
val ageMatters = spark.sql(q5)
ageMatters.writeStream.outputMode("update").option("truncate",
false).format("console").start().awaitTermination(10)

print("marital status mattered for subscription to deposit:")
val q6 ="""select marital,y as subscription_status from bankdata group by marital,y"""
val status = spark.sql(q6)
status.writeStream.outputMode("update").option("truncate",
false).format("console").start().awaitTermination(10)

print(" age and marital status together mattered for subscription to  deposit scheme:")
val q7 ="""select age,marital,y as subscription_status from bankdata group by marital,y,age"""
val together= spark.sql(q7)
together.writeStream.outputMode("update").option("truncate",false).format("console").start
().awaitTermination(10)

}
}
```

2) Commands to execute:

```
spark-shell -i project.scala
finalProject.main(Array("file:///user/maria_dev/finalProject/Bank.txt"))
```