// 1. Load data and create Spark data frame

```
scala> val lines =
sc.textFile("/user/nmabenteksystems/Project/dataset_bank_

full.csv")

lines: org.apache.spark.rdd.RDD[String] =
/user/nmabenteksystems/Project/dat

aset_bank_full.csv MapPartitionsRDD[1] at textFile at <console>:24

scala> val bank = lines.map(x => x.split(";"))

bank: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[2] at
map at <console>:25

scala> val bfields = bank.mapPartitionsWithIndex { (idx, iter) => if (idx
== 0) iter.drop(1) else iter }

bfields: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[3]
at mapPartitionsWithIndex at <console>:25
```

//Define Class for the schema

```
scala> case class Bank(age:Int, job:String, marital:String,
education:String, dft:String, balance:Int, housing:String, loan:String,
contact:String, day:I

nt, month: String, duration:Int, campaign:Int, pdays:Int, previous:Int,
poutcome:String, y:String)

defined class Bank

scala> val bankrdd = bfields.map( x =>
Bank(x(0).replaceAll("\"","").toInt,
x(1).replaceAll("\"","),x(2).replaceAll("\"",""),
x(3).replaceAll("\"",""),

x(4).replaceAll("\"",""), x(5).toInt, x(6).replaceAll("\"",""),
x(7).replaceAll("\"",""), x(8).replaceAll("\"",""), x(9).toInt,
x(10).replaceAll("\"","")

, x(11).toInt, x(12).toInt, x(13).toInt, x(14).toInt,
x(15).replaceAll("\"",""), x(16).replaceAll("\"","") ) )

bankrdd: org.apache.spark.rdd.RDD[Bank] = MapPartitionsRDD[4] at map at
<console>:27
```

```
scala> val df = bankrdd.toDF()

22/05/27 06:22:51 WARN lineage.LineageWriter: Lineage directory
/var/log/spark/lineage doesn't exist or is not writable. Lineage for this
application will be disabled.

df: org.apache.spark.sql.DataFrame = [age: int, job: string ... 15 more
fields]


scala> df.registerTempTable("bank")

warning: there was one deprecation warning; re-run with -deprecation for
details
```

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

```
scala> val sqlContext = spark.sqlContext

sqlContext: org.apache.spark.sql.SQLContext =
org.apache.spark.sql.SQLContext@38308731


scala> spark.conf.set("spark.sql.crossJoin.enabled", "true")


scala> val success = sqlContext.sql("select (a.subscribed/b.total)*100 as
success_percent from (select count(*) as subscribed from bank where
y='yes') a,(select count(*) as total from bank) b").show()

+------------------+

|   success_percent|

+------------------+

|11.698480458295547|

+------------------+


success: Unit = ()
```

// 2a Give marketing failure rate

```
scala> val failure = sqlContext.sql("select
(a.not_subscribed/b.total)*100 as failure_percent from (select count(*)
as not_subscribed from bank where y='no') a,(select count(*) as total from
bank) b").show()
```

```
+-----------------+

|  failure_percent|

+-----------------+

|88.30151954170445|

+-----------------+


failure: Unit = ()
```

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

```
scala> df.select(max($"age")).show()
```

```
+--------+

|max(age)|

+--------+

|      95|

+--------+
```

```
scala> df.select(min($"age")).show()
```

```
+--------+

|min(age)|

+--------+

|      18|
```

```
+--------+
```

```
scala> df.select(avg($"age")).show()
+-----------------+
|         avg(age)|
+-----------------+
|40.93621021432837|
+-----------------+
```

```
scala> df.select("age").summary().show()
+-------+-----------------+
|summary|              age|
+-------+-----------------+
|  count|            45211|
|   mean|40.93621021432837|
| stddev|10.61876204097542|
|    min|               18|
|    25%|               33|
|    50%|               39|
|    75%|               48|
|    max|               95|
+-------+-----------------+
```

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

```
scala> df.select(avg($"balance")).show()
+------------------+
|      avg(balance)|
+------------------+
|1362.2720576850766|
+------------------+
```

```
scala> val median = sqlContext.sql("SELECT percentile_approx(balance,
0.5) FROM bank").show()
+-----------------------------------------------------+
|percentile_approx(balance, CAST(0.5 AS DOUBLE), 10000)|
+-----------------------------------------------------+
|                                                  448|
+-----------------------------------------------------+

median: Unit = ()
```

// 5. Check if age matters in marketing subscription for deposit

```
scala> val age = sqlContext.sql("select age, count(*) as number from bank
where y='yes' group by age order by number desc").show()
+---+------+
|age|number|
+---+------+
| 32|   221|
| 30|   217|
| 33|   210|
```

```
| 35|   209|
| 31|   206|
| 34|   198|
| 36|   195|
| 29|   171|
| 37|   170|
| 28|   162|
| 38|   144|
| 39|   143|
| 27|   141|
| 26|   134|
| 41|   120|
| 46|   118|
| 40|   116|
| 47|   113|
| 25|   113|
| 42|   111|
+---+------+
only showing top 20 rows


age: Unit = ()
```

// 6. Check if marital status mattered for subscription to deposit.

```
scala> val marital = sqlContext.sql("select marital, count(*) as number
from bank where y='yes' group by marital order by number desc").show()
+--------+------+
| marital|number|
```

```
+--------+------+

| married|  2755|

|  single|  1912|

|divorced|   622|

+--------+------+


marital: Unit = ()
```

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

```
scala> df.groupBy($"y".alias("Did the customer
subscribed")).agg(count($"marital").alias("marital count")).show()

+---------------------------+-------------+

|Did the customer subscribed|marital count|

+---------------------------+-------------+

|                         no|        39922|

|                        yes|         5289|

+---------------------------+-------------+


scala> val age_marital = sqlContext.sql("select age, marital, count(*) as
number from bank where y='yes' group by age,marital order by number
desc").show()

+---+-------+------+

|age|marital|number|

+---+-------+------+

| 30| single|   151|

| 28| single|   138|

| 29| single|   133|
```

```
| 32| single|   124|
| 26| single|   121|
| 34|married|   118|
| 31| single|   111|
| 27| single|   110|
| 35|married|   101|
| 36|married|   100|
| 25| single|    99|
| 37|married|    98|
| 33|married|    97|
| 33| single|    97|
| 39|married|    87|
| 32|married|    87|
| 38|married|    86|
| 35| single|    84|
| 47|married|    83|
| 31|married|    80|
+---+-------+------+
only showing top 20 rows


age_marital: Unit = ()
```

// 8. Do feature engineering for column age and find right age effect on campaign

```
scala> import scala.reflect.runtime.universe
import scala.reflect.runtime.universe
```

```
scala> import org.apache.spark.SparkConf
import org.apache.spark.SparkConf

scala> import org.apache.spark.SparkContext
import org.apache.spark.SparkContext

scala> import org.apache.spark.sql.DataFrame
import org.apache.spark.sql.DataFrame

scala> import org.apache.spark.sql.SQLContext
import org.apache.spark.sql.SQLContext

scala> import org.apache.spark.sql.functions.mean
import org.apache.spark.sql.functions.mean

scala> val ageRDD = sqlContext.udf.register("ageRDD",(age:Int) => {
     |  if (age < 20)
     |  "Teen"
     |  else if (age > 20 && age <= 32)
     |  "Young"
     |  else if (age > 33 && age <= 55)
     |  "Middle Aged"
     |  else
     |  "Old"
     |  })
ageRDD: org.apache.spark.sql.expressions.UserDefinedFunction =
UserDefinedFunction(<function1>,StringType,Some(List(IntegerType)))
```

//Replacing old "age" column with new "age" column

```
scala> val banknewDF = df.withColumn("age",ageRDD(df("age")))

banknewDF: org.apache.spark.sql.DataFrame = [age: string, job: string ...
15 more fields]
```

//Running a query to see the age group which subscribed the most. We see it's 'Middle-Aged'

```
scala> banknewDF.registerTempTable("bank_new")

warning: there was one deprecation warning; re-run with -deprecation for
details
```

//Pipeline

```
scala> val age_target = sqlContext.sql("select age, count(*) as number from
bank_new where y='yes' group by age order by number desc ").show()

+-----------+------+

|        age|number|

+-----------+------+

|Middle Aged|  2601|

|      Young|  1539|

|        Old|  1131|

|       Teen|    18|

+-----------+------+


age_target: Unit = ()


scala> import org.apache.spark.ml.feature.StringIndexer

import org.apache.spark.ml.feature.StringIndexer
```

```
scala> val ageInd = new
StringIndexer().setInputCol("age").setOutputCol("ageIndex")

ageInd: org.apache.spark.ml.feature.StringIndexer = strIdx_ec9f5b32e837


scala> var strIndModel = ageInd.fit(banknewDF)

strIndModel: org.apache.spark.ml.feature.StringIndexerModel =
strIdx_ec9f5b32e837


scala>
strIndModel.transform(banknewDF).select("age","ageIndex").show(5)

+-----------+--------+

|        age|ageIndex|

+-----------+--------+

|        Old|     2.0|

|Middle Aged|     0.0|

|        Old|     2.0|

|Middle Aged|     0.0|

|        Old|     2.0|

+-----------+--------+

only showing top 5 rows


scala>
```