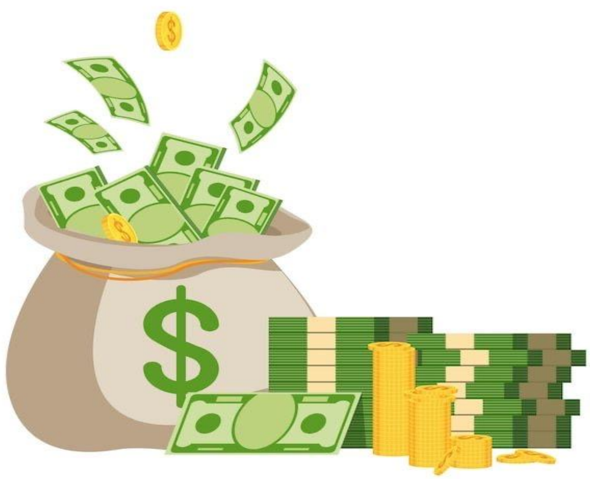
KING SAUD UNIVERSITY
COLLEGE OF COMPUTER AND INFORMATION SCIENCES INFORMATION TECHNOLOGY DEPARTMENT

IT462

Big Data



Group members:
<Dana alsaeedi, 441201237>
<Atheer alzaid, 441201404>

Supervised by

L.Nada alharbi

Table of content

1.introduction

About our dataset :

This dataset contains insights into a collection of credit card transactions made in India, offering a comprehensive look at the spending habits of Indians across the nation. From the Gender and Card type used to carry out each transaction, to which city saw the highest amount of spending and even what kind of expenses were made, this dataset paints an overall picture about how money is being spent in India today.

2.Data preprocessing

First we installed pyspark on our google colab using the following command:

```
[ ] !pip install pyspark py4j

    Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
    Collecting pyspark
      Downloading pyspark-3.3.1.tar.gz (281.4 MB)
         ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 281.4/281.4 MB 5.2 MB/s eta 0:00:00
      Preparing metadata (setup.py) ... done
    Collecting py4j
      Downloading py4j-0.10.9.7-py2.py3-none-any.whl (200 kB)
         ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 200.5/200.5 KB 21.9 MB/s eta 0:00:00
      Downloading py4j-0.10.9.5-py2.py3-none-any.whl (199 kB)
         ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 199.7/199.7 KB 17.7 MB/s eta 0:00:00
    Building wheels for collected packages: pyspark
      Building wheel for pyspark (setup.py) ... done
      Created wheel for pyspark: filename=pyspark-3.3.1-py2.py3-none-any.whl size=281845512 sha256=33ee66323639f2e8b5a4a98009f492d514ed985dc6f90369197b44
      Stored in directory: /root/.cache/pip/wheels/43/dc/11/ec201cd671da62fa9c5cc77078235e40722170ceba231d7598
    Successfully built pyspark
    Installing collected packages: py4j, pyspark
    Successfully installed py4j-0.10.9.5 pyspark-3.3.1
```

Then we started a sparkSession and imported our dataset using spark.read.csv()

```
[25] from pyspark.sql import SparkSession
```

```
[98] spark = SparkSession.builder.getOrCreate()
```

```
[99] spark.conf.set("spark.sql.legacy.timeParserPolicy","LEGACY")
```

```
[100] DF = spark.read.csv("/content/drive/MyDrive/data/Creditcard.csv", inferSchema=True, header = True)
```

We used method show() to make sure our dataset was imported and to check on the first 20 rows

```
DF.show()
```

```
+-----+--------------------+---------+---------+--------+------+------+
|index|                City|     Date|Card Type|Exp Type|Gender|Amount|
+-----+--------------------+---------+---------+--------+------+------+
|    0|        Delhi, India|29-Oct-14|     Gold|   Bills|     F| 82475|
|    1|Greater Mumbai, I...|22-Aug-14| Platinum|   Bills|     F| 32555|
|    2|     Bengaluru, India|27-Aug-14|   Silver|   Bills|     F|101738|
|    3|Greater Mumbai, I...|12-Apr-14|Signature|   Bills|     F|123424|
|    4|     Bengaluru, India| 5-May-15|     Gold|   Bills|     F|171574|
|    5|        Delhi, India| 8-Sep-14|   Silver|   Bills|     F|100036|
|    6|        Delhi, India|24-Feb-15|     Gold|   Bills|     F|143250|
|    7|Greater Mumbai, I...|26-Jun-14| Platinum|   Bills|     F|150980|
|    8|        Delhi, India|28-Mar-14|   Silver|   Bills|     F|192247|
|    9|        Delhi, India| 1-Sep-14| Platinum|   Bills|     F| 67932|
|   10|        Delhi, India|22-Jun-14| Platinum|   Bills|     F|280061|
|   11|Greater Mumbai, I...| 7-Dec-13|Signature|   Bills|     F|278036|
|   12|Greater Mumbai, I...| 7-Aug-14|     Gold|   Bills|     F| 19226|
|   13|        Delhi, India|27-Apr-14|Signature|   Bills|     F|254359|
|   14|Greater Mumbai, I...|15-Aug-14|Signature|   Bills|     F|302834|
|   15|Greater Mumbai, I...|28-Nov-14| Platinum|   Bills|     F|647116|
|   16|Greater Mumbai, I...|14-Jun-14|Signature|   Bills|     F|421878|
|   17|Greater Mumbai, I...|30-Mar-15|     Gold|   Bills|     F|986379|
|   18|Greater Mumbai, I...|15-Mar-14| Platinum|   Bills|     F|213047|
|   19|Greater Mumbai, I...| 9-Nov-13| Platinum|   Bills|     F|735566|
```

We renamed columns (Card Type) and (Exp Type) using withColumnRenamed() to remove the space so it would be suitable more in programming

```
[30] DF = DF.withColumnRenamed("Card Type","CardType")
```

```
[31] DF = DF.withColumnRenamed("Exp Type","ExpType")
```

```
DF.show()
```

```
+-----+-------------------+---------+---------+-------+------+------+
|index|               City|     Date| CardType|ExpType|Gender|Amount|
+-----+-------------------+---------+---------+-------+------+------+
|    0|       Delhi, India|29-Oct-14|     Gold|  Bills|     F| 82475|
|    1|Greater Mumbai, I...|22-Aug-14| Platinum|  Bills|     F| 32555|
|    2|    Bengaluru, India|27-Aug-14|   Silver|  Bills|     F|101738|
|    3|Greater Mumbai, I...|12-Apr-14|Signature|  Bills|     F|123424|
|    4|    Bengaluru, India| 5-May-15|     Gold|  Bills|     F|171574|
|    5|       Delhi, India| 8-Sep-14|   Silver|  Bills|     F|100036|
|    6|       Delhi, India|24-Feb-15|     Gold|  Bills|     F|143250|
|    7|Greater Mumbai, I...|26-Jun-14| Platinum|  Bills|     F|150980|
|    8|       Delhi, India|28-Mar-14|   Silver|  Bills|     F|192247|
|    9|       Delhi, India| 1-Sep-14| Platinum|  Bills|     F| 67932|
|   10|       Delhi, India|22-Jun-14| Platinum|  Bills|     F|280061|
|   11|Greater Mumbai, I...| 7-Dec-13|Signature|  Bills|     F|278036|
|   12|Greater Mumbai, I...| 7-Aug-14|     Gold|  Bills|     F| 19226|
|   13|       Delhi, India|27-Apr-14|Signature|  Bills|     F|254359|
```

We checked if there are any null values in our dataset by using isNull() for each column, as we can see there weren't any null values.
Null values can cause poor accuracy and performance.

```
DF.filter(DF.CardType.isNull()).show()
```

```
+-----+----+----+--------+-------+------+------+
|index|City|Date|CardType|ExpType|Gender|Amount|
+-----+----+----+--------+-------+------+------+
+-----+----+----+--------+-------+------+------+
```

```
[ ]
```

```
[36] DF.filter(DF.ExpType.isNull()).show()
```

```
+-----+----+----+--------+-------+------+------+
|index|City|Date|CardType|ExpType|Gender|Amount|
+-----+----+----+--------+-------+------+------+
+-----+----+----+--------+-------+------+------+
```

```
[37] DF.filter(DF.Gender.isNull()).show()
```

```
+-----+----+----+--------+-------+------+------+
|index|City|Date|CardType|ExpType|Gender|Amount|
+-----+----+----+--------+-------+------+------+
+-----+----+----+--------+-------+------+------+
```

```
[ ]
```

```
DF.filter(DF.Amount.isNull()).show()
```

```
+-----+----+----+--------+-------+------+------+
|index|City|Date|CardType|ExpType|Gender|Amount|
+-----+----+----+--------+-------+------+------+
+-----+----+----+--------+-------+------+------+
```

Printing schema (we noticed here that Date is of type string) which is something that we don't want

```
[39] DF.printSchema()

root
 |-- index: integer (nullable = true)
 |-- City: string (nullable = true)
 |-- Date: string (nullable = true)
 |-- CardType: string (nullable = true)
 |-- ExpType: string (nullable = true)
 |-- Gender: string (nullable = true)
 |-- Amount: integer (nullable = true)
```

We changed the months names in column (Date) into numbers using regexp_replace(), so we could have a correct formula and change the type of column (Date) to date

```
[43] DF = DF.withColumn("Date" , F.regexp_replace("Date" , "Jan" , "01"))
```

```
[44] DF = DF.withColumn("Date" , F.regexp_replace("Date" , "Feb" , "02"))
```

```
[45] DF = DF.withColumn("Date" , F.regexp_replace("Date" , "Mar" , "03"))
```

```
[46] DF = DF.withColumn("Date" , F.regexp_replace("Date" , "Apr" , "04"))
```

```
[47] DF = DF.withColumn("Date" , F.regexp_replace("Date" , "May" , "05"))
```

```
[48] DF = DF.withColumn("Date" , F.regexp_replace("Date" , "Jun" , "06"))
```

```
[49] DF = DF.withColumn("Date" , F.regexp_replace("Date" , "Jul" , "07"))
```

```
[50] DF = DF.withColumn("Date" , F.regexp_replace("Date" , "Aug" , "08"))
```

```
[51] DF = DF.withColumn("Date" , F.regexp_replace("Date" , "Sep" , "09"))
```

```
[50] DF = DF.withColumn("Date" , F.regexp_replace("Date" , "Aug" , "08"))
```

```
[51] DF = DF.withColumn("Date" , F.regexp_replace("Date" , "Sep" , "09"))
```

```
[52] DF = DF.withColumn("Date" , F.regexp_replace("Date" , "Oct" , "10"))
```

```
[53] DF = DF.withColumn("Date" , F.regexp_replace("Date" , "Nov" , "11"))
```

```
[54] DF = DF.withColumn("Date" , F.regexp_replace("Date" , "Dec" , "12"))
```

DataFrame after changing

```
[55] DF.show()
```

```
+-----+--------------------+--------+---------+-------+------+------+
|index|                City|    Date| CardType|ExpType|Gender|Amount|
+-----+--------------------+--------+---------+-------+------+------+
|    0|        Delhi, India|29-10-14|     Gold|  Bills|     F| 82475|
|    1|Greater Mumbai, I...|22-08-14| Platinum|  Bills|     F| 32555|
|    2|    Bengaluru, India|27-08-14|   Silver|  Bills|     F|101738|
|    3|Greater Mumbai, I...|12-04-14|Signature|  Bills|     F|123424|
|    4|    Bengaluru, India| 5-05-15|     Gold|  Bills|     F|171574|
|    5|        Delhi, India| 8-09-14|   Silver|  Bills|     F|100036|
|    6|        Delhi, India|24-02-15|     Gold|  Bills|     F|143250|
|    7|Greater Mumbai, I...|26-06-14| Platinum|  Bills|     F|150980|
|    8|        Delhi, India|28-03-14|   Silver|  Bills|     F|192247|
|    9|        Delhi, India| 1-09-14| Platinum|  Bills|     F| 67932|
|   10|        Delhi, India|22-06-14| Platinum|  Bills|     F|280061|
|   11|Greater Mumbai, I...| 7-12-13|Signature|  Bills|     F|278036|
|   12|Greater Mumbai, I...| 7-08-14|     Gold|  Bills|     F| 19226|
|   13|        Delhi, India|27-04-14|Signature|  Bills|     F|254359|
|   14|Greater Mumbai, I...|15-08-14|Signature|  Bills|     F|302834|
|   15|Greater Mumbai, I...|28-11-14| Platinum|  Bills|     F|647116|
|   16|Greater Mumbai, I...|14-06-14|Signature|  Bills|     F|421878|
|   17|Greater Mumbai, I...|30-03-15|     Gold|  Bills|     F|986379|
|   18|Greater Mumbai, I...|15-03-14| Platinum|  Bills|     F|213047|
|   19|Greater Mumbai, I...| 9-11-13| Platinum|  Bills|     F|735566|
```

After that we were able to change the type of (Date) into date using withColumn() and to_date()

```
[ ] DF = DF.withColumn("Date",to_date("Date","dd-MM-yy"))
```

```
DF.printSchema()
```

```
root
 |-- index: integer (nullable = true)
 |-- City: string (nullable = true)
 |-- Date: date (nullable = true)
 |-- CardType: string (nullable = true)
 |-- ExpType: string (nullable = true)
 |-- Gender: string (nullable = true)
 |-- Amount: integer (nullable = true)
```

Then we wanted to create a new column (spending) to categorize spending amount into two categories (high) and (low)

First we used mean() to get the mean of column (Amount)

```
[100] data.mean()
```

```
<ipython-input-100-abc01cf6c622>:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') :
  data.mean()
index      13025.500000
Amount    156411.537425
dtype: float64
```

Then we created new column (spending) that has two values (high) and (low)

```
[105] DF = DF.withColumn(
          'spending',
          F.when((F.col("Amount") >= '156422'), 'high')\
          .when((F.col("Amount") < '156411') , 'low')\
          .otherwise(0)
      )
```

We extracted the year only from column (date) since the date may not be helpful in our analysis as much as the year

```
[ ] DF = DF.withColumn('year', col('Date').substr(1,4))
```

This is how the final data frame looks like:

City	CardType	ExpType	Gender	Amount	year	spending
Delhi, India	Gold	Bills	F	82475	2014	0
Greater Mumbai, I...	Platinum	Bills	F	32555	2014	0
Bengaluru, India	Silver	Bills	F	101738	2014	0
Greater Mumbai, I...	Signature	Bills	F	123424	2014	0
Bengaluru, India	Gold	Bills	F	171574	2015	1
Delhi, India	Silver	Bills	F	100036	2014	0
Delhi, India	Gold	Bills	F	143250	2015	0
Greater Mumbai, I...	Platinum	Bills	F	150980	2014	0
Delhi, India	Silver	Bills	F	192247	2014	1
Delhi, India	Platinum	Bills	F	67932	2014	0
Delhi, India	Platinum	Bills	F	280061	2014	1
Greater Mumbai, I...	Signature	Bills	F	278036	2013	1
Greater Mumbai, I...	Gold	Bills	F	19226	2014	0
Delhi, India	Signature	Bills	F	254359	2014	1
Greater Mumbai, I...	Signature	Bills	F	302834	2014	1
Greater Mumbai, I...	Platinum	Bills	F	647116	2014	1
Greater Mumbai, I...	Signature	Bills	F	421878	2014	1
Greater Mumbai, I...	Gold	Bills	F	986379	2015	1
Greater Mumbai, I...	Platinum	Bills	F	213047	2014	1
Greater Mumbai, I...	Platinum	Bills	F	735566	2013	1

3.RDD operations

To implement RDD operations on our dataset, we created a case class that matches the data in our file. Then we read the file and we created an RDD named credit to perform our operations on it.

```
scala> case class card(index: Int, City: String, Date: String, CardType: String, ExpType: String, Gender: String, Amount: Int)
defined class card

scala> val textFile = sc.textFile("D:/BigData/CreditcardN.csv")
textFile: org.apache.spark.rdd.RDD[String] = D:/BigData/CreditcardN.csv MapPartitionsRDD[1] at textFile at <console>:24

scala> val credit = textFile.map{ row =>
     | val fields = row.split(",").map(_.trim)
     | card(fields(0).toInt, fields(1), fields(2), fields(3), fields(4), fields(5), fields(6).toInt)}
credit: org.apache.spark.rdd.RDD[card] = MapPartitionsRDD[2] at map at <console>:27
```

First we used the action takeOrdered() to see the biggest transaction that was made in India which was 998077
It turned out to be from greater mumbai and by a female, which lead to the conclusion that spendings are high in greater mumbai.

```
scala> credit.takeOrdered(1)(Ordering[Int].reverse.on(x=>x.Amount))
res4: Array[card] = Array(card(80,Greater Mumbai,14-Oct-14,Platinum,Bills,F,998077))
```

Then we used transformation filter() to include only the transactions with card type : gold
And we used action count() to see how many transactions were made, which are 6367
Which means that few people in India use card type gold.

```
scala> val y = credit.filter(x=>x.CardType=="Gold")
y: org.apache.spark.rdd.RDD[card] = MapPartitionsRDD[6] at filter at <console>:25

scala> y.count()
res9: Long = 6367
```

we used transformation filter() to include only transactions that has amount greater than 800000
And we used action collect() to take a look on the transactions
We can see that majority of the transactions were made in greater mumbai , delhi , bengaluru and ahmedabad. Which include that living is significantly expensive in these cities.

```
scala> val w = credit.filter(_.Amount > 800000)
w: org.apache.spark.rdd.RDD[card] = MapPartitionsRDD[11] at filter at <console>:25

scala> w.collect
res17: Array[card] = Array(card(17,Greater Mumbai,30-Mar-15,Gold,Bills,F,986379), card(21,Delhi,1-Jul-14,Signature,Bills,F,809623), card(28,Bengaluru,18-Jan-15,Platinum,Bil
ls,F,987935), card(33,Ahmedabad,8-Nov-14,Gold,Bills,F,864090), card(35,Ahmedabad,24-Mar-15,Platinum,Bills,F,954660), card(42,Bengaluru,10-Nov-14,Platinum,Bills,F,804938), c
ard(43,Delhi,30-Jan-15,Silver,Bills,F,888341), card(46,Ahmedabad,10-Dec-13,Gold,Bills,F,892016), card(58,Delhi,8-Oct-13,Platinum,Bills,F,900101), card(68,Greater Mumbai,22-
Mar-14,Gold,Bills,F,991685), card(70,Greater Mumbai,8-May-14,Gold,Bills,F,829742), card(73,Greater Mumbai,14-Jun-14,Platinum,Bills,F,835872), card(80,Greater Mumbai,14-Oct-
14,Platinum,Bills,F,998077), card(81,Ahmedabad,2-Feb-14,Silver,Bills,F,934205), card(82,Bengaluru,1-Apr-15,Pl...
```

We used action takesample() and we took sample of 50 transactions to have a better perspective on our dataset
We can see that female transactions were mostly on groceries , and male transactions are mostly on fuel and bills. Also female and male spending amounts appear to be similar.

```
scala> credit.takeSample(true,50)
res18: Array[card] = Array(card(22494,Jaipur,28-Jan-14,Platinum,Grocery,F,163080), card(15231,Karaikal,15-Dec-14,Platinum,Entertainment,M,154629), card(7842,Ahmedabad,22-De
c-14,Silver,Fuel,F,293435), card(24396,Lucknow,10-Mar-14,Signature,Fuel,M,282212), card(6334,Bengaluru,12-Jun-14,Signature,Entertainment,F,130892), card(11666,Greater Mumba
i,4-Jun-14,Silver,Bills,M,267655), card(14470,Jatani,28-Jan-14,Signature,Grocery,F,162425), card(9580,Greater Mumbai,30-Jul-14,Gold,Food,M,110722), card(7043,Greater Mumbai
,7-Dec-13,Gold,Bills,M,171932), card(4119,Bengaluru,6-Dec-14,Gold,Fuel,M,34697), card(6417,Greater Mumbai,18-Aug-14,Platinum,Fuel,F,221674), card(5175,Delhi,18-Dec-13,Silve
r,Grocery,F,146001), card(20257,Kanpur,31-Oct-13,Silver,Grocery,F,111158), card(16024,Rajkot,29-Apr-14,Platin...
```

We used map and reduceByKey to map each city with how many transactions it made.
We can see here that some cities made so many transactions like Kanpur made 764 transactions and pune made 747 transactions, while other cities made only few transactions like Paradip and Uran Islampur who made only 2 transactions each.
We also used the action foreach to print the result.

```
scala> val rdd:RDD[(String,Int)]=credit.map(m=>(m.City,1))
rdd: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[5] at map at <console>:26

scala> val rdd2 = rdd.reduceByKey(_+_)
rdd2: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[6] at reduceByKey at <console>:26

scala> rdd2.foreach(println)
(Paradip,2)
(Uran Islampur,2)
(Vinukonda,4)
(Suryapet,4)
(Thrissur,6)
(Goalpara,6)
(Madhugiri,5)
(Sirkali,5)
(Madikeri,6)
(Palacole,10)
(Fazilka,1)
(Rajpipla,3)
(Lakshmeshwar,6)
(Vikramasingapuram,5)
(Mapusa,4)
(Deesa,3)
(Pilani,9)
(Suratgarh,3)
(Kanpur,764)
(Baramula,11)
(Hoshiarpur,3)
(Gurgaon,12)
(Tiruchendur,9)
(Wai,5)
(Ajmer,4)
(Sitamarhi,9)
(Raisinghnagar,3)
(Solan,4)
(Sujangarh,6)
(Cherthala,4)
(Baleshwar Town,7)
(Dhuri,6)
(Theni Allinagaram,5)
```

We used a transformation filter to watch the relationships between cities that had the greatest transactions and having the gold card. We noticed that all these cities have a similar number of gold cards, Delhi has the most and Ahmedabad has the least.

```
scala> val g =credit.filter(x=>x.City=="Delhi").filter(x=>x.CardType =="Gold")
g: org.apache.spark.rdd.RDD[card] = MapPartitionsRDD[18] at filter at <console>:25

scala> g.count
res17: Long = 863

scala> val g =credit.filter(x=>x.City=="Bengaluru").filter(x=>x.CardType =="Gold")
g: org.apache.spark.rdd.RDD[card] = MapPartitionsRDD[20] at filter at <console>:25

scala> g.count
res18: Long = 857

scala> val g =credit.filter(x=>x.City=="Ahmedabad").filter(x=>x.CardType =="Gold")
g: org.apache.spark.rdd.RDD[card] = MapPartitionsRDD[22] at filter at <console>:25

scala> g.count
res19: Long = 809

scala> val g =credit.filter(x=>x.City=="Greater Mumbai").filter(x=>x.CardType =="Gold")
g: org.apache.spark.rdd.RDD[card] = MapPartitionsRDD[24] at filter at <console>:25

scala> g.count
res20: Long = 848
```

We transformed the dataset to lowercase to ease the use of it.

```
scala> val L =textFile.map(_.toLowerCase)
L: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[35] at map at <console>:25

scala> L.collect
res48: Array[String] = Array(0,delhi ,29-oct-14,gold,bills,f,82475, 1,greater mumbai ,22-aug-14,platinum,bills,f,32555, 2,bengaluru ,27-aug-14,silver,bills,f,101738, 3,grea
ter mumbai ,12-apr-14,signature,bills,f,123424, 4,bengaluru ,5-may-15,gold,bills,f,171574, 5,delhi ,8-sep-14,silver,bills,f,100036, 6,delhi ,24-feb-15,gold,bills,f,143250,
7,greater mumbai ,26-jun-14,platinum,bills,f,150980, 8,delhi ,28-mar-14,silver,bills,f,192247, 9,delhi ,1-sep-14,platinum,bills,f,67932, 10,delhi ,22-jun-14,platinum,bills,
f,280061, 11,greater mumbai ,7-dec-13,signature,bills,f,278036, 12,greater mumbai ,7-aug-14,gold,bills,f,19226, 13,delhi ,27-apr-14,signature,bills,f,254359, 14,greater mum
bai ,15-aug-14,signature,bills,f,302834, 15,greater mumbai ,28-nov-14,platinum,bills,f,647116, 16,greater mum...
```

4.SQL operations

To implement sql operations in our dataset we imported the following libraries, created a case class then read the data file and we transformed it to dataframe using toDF(). We also created a view for the dataframe.

```
scala> import spark.sqlContext
import spark.sqlContext

scala> import sqlContext.implicits._
import sqlContext.implicits._

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

scala> case class card(index: Int, City: String, Date: String, CardType: String, ExpType: String, Gender: String, Amount: Int)
defined class card
```

```
scala> val cardDF =
     | spark.sparkContext.textFile("D:/BigData/CreditcardN.csv").map(_.split(",")).map(attributes=>card(attributes(0).trim.toInt,attributes(1),attributes(2),attributes(3),a
ttributes(4),attributes(5),attributes(6).trim.toInt)).toDF()
cardDF: org.apache.spark.sql.DataFrame = [index: int, City: string ... 5 more fields]

scala> cardDF.show
+-----+--------------+---------+---------+-------+------+------+
|index|          City|     Date| CardType|ExpType|Gender|Amount|
+-----+--------------+---------+---------+-------+------+------+
|    0|        Delhi |29-Oct-14|     Gold|  Bills|     F| 82475|
|    1|Greater Mumbai |22-Aug-14| Platinum|  Bills|     F| 32555|
|    2|     Bengaluru |27-Aug-14|   Silver|  Bills|     F|101738|
|    3|Greater Mumbai |12-Apr-14|Signature|  Bills|     F|123424|
|    4|     Bengaluru | 5-May-15|     Gold|  Bills|     F|171574|
|    5|        Delhi | 8-Sep-14|   Silver|  Bills|     F|100036|
|    6|        Delhi |24-Feb-15|     Gold|  Bills|     F|143250|
|    7|Greater Mumbai |26-Jun-14| Platinum|  Bills|     F|150980|
|    8|        Delhi |28-Mar-14|   Silver|  Bills|     F|192247|
|    9|        Delhi | 1-Sep-14| Platinum|  Bills|     F| 67932|
|   10|        Delhi |22-Jun-14| Platinum|  Bills|     F|280061|
|   11|Greater Mumbai | 7-Dec-13|Signature|  Bills|     F|278036|
|   12|Greater Mumbai | 7-Aug-14|     Gold|  Bills|     F| 19226|
|   13|        Delhi |27-Apr-14|Signature|  Bills|     F|254359|
|   14|Greater Mumbai |15-Aug-14|Signature|  Bills|     F|302834|
|   15|Greater Mumbai |28-Nov-14| Platinum|  Bills|     F|647116|
|   16|Greater Mumbai |14-Jun-14|Signature|  Bills|     F|421878|
|   17|Greater Mumbai |30-Mar-15|     Gold|  Bills|     F|986379|
|   18|Greater Mumbai |15-Mar-14| Platinum|  Bills|     F|213047|
|   19|Greater Mumbai | 9-Nov-13| Platinum|  Bills|     F|735566|
+-----+--------------+---------+---------+-------+------+------+
only showing top 20 rows


scala> cardDF.createOrReplaceTempView("card")
```

For the first operation we calculated the average spending amount for both females and males as shown:
We can see that average spending for females was around 161206. Which is higher than the average spending for males. We can conclude that in general females spend significantly more money than males in india.

```
scala> cardDF.groupBy(cardDF.col("Gender")).agg(avg("Amount")).show
+------+------------------+
|Gender|       avg(Amount)|
+------+------------------+
|     F| 161206.9466374269|
|     M|151109.14508567733|
+------+------------------+
```

For the second operation we viewed the dataframe when the spending amount was less than 156422. Which is the mean for Amount that we calculated in the preprocessing phase. From what we can see most of them came from Delhi, greater mumbai and bengaluru. Which lead us to the conclusion that spending habits in these cities are not too high.

```
scala> cardDF.where($"Amount"< 156422).show
+-----+---------------+---------+---------+-------+------+------+
|index|           City|     Date| CardType|ExpType|Gender|Amount|
+-----+---------------+---------+---------+-------+------+------+
|    0|         Delhi |29-Oct-14|     Gold|  Bills|     F| 82475|
|    1|Greater Mumbai |22-Aug-14| Platinum|  Bills|     F| 32555|
|    2|     Bengaluru |27-Aug-14|   Silver|  Bills|     F|101738|
|    3|Greater Mumbai |12-Apr-14|Signature|  Bills|     F|123424|
|    5|         Delhi | 8-Sep-14|   Silver|  Bills|     F|100036|
|    6|         Delhi |24-Feb-15|     Gold|  Bills|     F|143250|
|    7|Greater Mumbai |26-Jun-14| Platinum|  Bills|     F|150980|
|    9|         Delhi | 1-Sep-14| Platinum|  Bills|     F| 67932|
|   12|Greater Mumbai | 7-Aug-14|     Gold|  Bills|     F| 19226|
|  353|Greater Mumbai |27-Aug-14|   Silver|  Bills|     F| 41002|
|  354|     Bengaluru | 3-Jan-14|   Silver|  Bills|     F| 34743|
|  357|     Ahmedabad |14-May-14|Signature|  Bills|     F|118112|
|  358|Greater Mumbai |12-Dec-13| Platinum|  Bills|     F| 61572|
|  360|Greater Mumbai |21-Feb-14|Signature|  Bills|     F| 43854|
|  361|         Delhi |27-Apr-14|     Gold|  Bills|     F|  8798|
|  363|     Bengaluru |10-Apr-14|   Silver|  Bills|     F|128164|
|  367|     Ahmedabad |13-May-15|     Gold|  Bills|     F| 13162|
|  368|     Bengaluru | 5-Feb-15| Platinum|  Bills|     F|  3427|
|  371|     Ahmedabad |19-Oct-14| Platinum|  Bills|     F|123417|
|  372|         Delhi | 3-Dec-13|   Silver|  Bills|     F| 81088|
+-----+---------------+---------+---------+-------+------+------+
only showing top 20 rows
```

And for the third operation we viewed the cities where the transaction card type was ‘Gold’
We can see that the cities were delhi, bengaluru, greater mumbai and ahmedabad
Which lead us to believe that people in these cities prefer the card type ‘Gold’.

```
scala> val cityDF = spark.sql("SELECT City , CardType FROM card WHERE CardType= 'Gold'")
cityDF: org.apache.spark.sql.DataFrame = [City: string, CardType: string]

scala> cityDF.map(x =>" "+x(0)+" "+x(1)).show
+-------------------+
|              value|
+-------------------+
|         Delhi  Gold|
|     Bengaluru  Gold|
|         Delhi  Gold|
| Greater Mumbai  ...|
| Greater Mumbai  ...|
|     Ahmedabad  Gold|
|     Ahmedabad  Gold|
|         Delhi  Gold|
| Greater Mumbai  ...|
|     Ahmedabad  Gold|
|         Delhi  Gold|
|     Bengaluru  Gold|
|     Ahmedabad  Gold|
| Greater Mumbai  ...|
|     Bengaluru  Gold|
| Greater Mumbai  ...|
| Greater Mumbai  ...|
| Greater Mumbai  ...|
| Greater Mumbai  ...|
|     Bengaluru  Gold|
+-------------------+
only showing top 20 rows
```

For the fourth operation we viewed the minimum amount of spending in each city so that we can know what city has the lowest minimum, from this we can see that Dhamtari has the lowest minimum which indicates the weakness of its economy.

```
scala> cardDF.groupBy(cardDF.col("City")).agg(min("Amount")).show
+-----------+-----------+
|       City|min(Amount)|
+-----------+-----------+
| Jehanabad |       8564|
| Bharatpur |      19746|
|  Ranaghat |      19343|
|    Batala |      16377|
|    Modasa |      25879|
|Wanaparthy |      54813|
| Kasaragod |      13463|
|    Guntur |      61543|
| Modinagar |      62599|
|  Vaijapur |      48429|
|    Tamluk |       6726|
|   Sandila |       3832|
|  Dhamtari |       1416|
|Mokokchung |       6269|
| Pathankot |      16257|
|    Kollam |      80808|
|      Rewa |      53947|
|Pratapgarh |       6204|
|   Aligarh |      64855|
|  Sibsagar |      15795|
+-----------+-----------+
only showing top 20 rows
```

For the last operation we tried to find correlation between the date and the amount so that we can see how the date affects the amount of money people spend. We chose 6000 since it's a low amount we wanted to see when people spend less.

```
scala> val cDF = spark.sql("SELECT Date,Amount FROM card WHERE Amount<6000")
cDF: org.apache.spark.sql.DataFrame = [Date: string, Amount: int]
```

```
scala> cDF.map(x =>" "+x(0)+" "+x(1)).show
+---------------+
|          value|
+---------------+
|  5-Feb-15 3427|
| 27-Apr-15 2138|
| 18-Oct-13 2397|
|  7-Feb-15 2686|
| 10-May-14 5397|
| 23-Jan-14 1400|
| 12-Apr-15 4377|
| 31-Oct-13 2586|
|  8-May-14 2741|
| 10-Jan-14 3421|
| 29-Apr-15 3621|
|  3-Feb-14 5855|
| 20-Mar-15 1709|
| 19-Jan-14 4590|
| 26-Feb-15 5706|
|  1-Oct-14 5545|
| 29-Jun-14 5100|
|  5-May-15 1448|
| 26-May-14 3445|
| 14-Apr-14 4607|
+---------------+
only showing top 20 rows
```

5.Machine Learning Operations

We decided to choose decision tree algorithm since it's one of the most powerful tools and it can effectively deal with large, complicated non-linear datasets without imposing a complicated parametric structure, it can also handle high-dimensional data really well and it has a good accuracy in general

First we imported the required libraries.

```
[55] from pyspark.ml import Pipeline
     from pyspark.ml.classification import DecisionTreeClassifier
     from pyspark.ml.feature import StringIndexer, VectorIndexer
     from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

```
[56] from pyspark.ml.linalg import Vectors
```

```
[57] from pyspark.ml.feature import VectorAssembler
```

```
[58] from pyspark.ml.feature import StringIndexer
```

We used StringIndexer for encoding categorical string columns of DataFrame into numerical values. For example if you live in Delhi that would be represented as 3. For gender males are represented as 1 and females are represented as 0. And so on

```
[59] indexerC = StringIndexer(inputCol="City", outputCol="CityIndex")
```

```
[60] DF = indexerC.fit(DF).transform(DF)
```

```
[61] indexerD = StringIndexer(inputCol="CardType", outputCol="CardIndex")
```

```
[62] DF = indexerD.fit(DF).transform(DF)
```

```
[63] indexerEx = StringIndexer(inputCol="ExpType", outputCol="ExpIndex")
```

```
[64] DF = indexerEx.fit(DF).transform(DF)
```

```
[65] indexerG = StringIndexer(inputCol="Gender", outputCol="genderIndex")
```

```
[66] DF = indexerG.fit(DF).transform(DF)
```

The results of the above operations .

```
DF.show()
```

```
+--------------------+---------+-------+------+------+----+--------+---------+---------+--------+-----------+
|                City| CardType|ExpType|Gender|Amount|year|spending|CityIndex|CardIndex|ExpIndex|genderIndex|
+--------------------+---------+-------+------+------+----+--------+---------+---------+--------+-----------+
|        Delhi, India|     Gold|  Bills|     F| 82475|2014|       0|      3.0|      3.0|     2.0|        0.0|
|Greater Mumbai, I...| Platinum|  Bills|     F| 32555|2014|       0|      1.0|      2.0|     2.0|        0.0|
|    Bengaluru, India|   Silver|  Bills|     F|101738|2014|       0|      0.0|      0.0|     2.0|        0.0|
|Greater Mumbai, I...|Signature|  Bills|     F|123424|2014|       0|      1.0|      1.0|     2.0|        0.0|
|    Bengaluru, India|     Gold|  Bills|     F|171574|2015|       1|      0.0|      3.0|     2.0|        0.0|
|        Delhi, India|   Silver|  Bills|     F|100036|2014|       0|      3.0|      0.0|     2.0|        0.0|
|        Delhi, India|     Gold|  Bills|     F|143250|2015|       0|      3.0|      3.0|     2.0|        0.0|
|Greater Mumbai, I...| Platinum|  Bills|     F|150980|2014|       0|      1.0|      2.0|     2.0|        0.0|
|        Delhi, India|   Silver|  Bills|     F|192247|2014|       1|      3.0|      0.0|     2.0|        0.0|
|        Delhi, India| Platinum|  Bills|     F| 67932|2014|       0|      3.0|      2.0|     2.0|        0.0|
|        Delhi, India| Platinum|  Bills|     F|280061|2014|       1|      3.0|      2.0|     2.0|        0.0|
|Greater Mumbai, I...|Signature|  Bills|     F|278036|2013|       1|      1.0|      1.0|     2.0|        0.0|
|Greater Mumbai, I...|     Gold|  Bills|     F| 19226|2014|       0|      1.0|      3.0|     2.0|        0.0|
|        Delhi, India|Signature|  Bills|     F|254359|2014|       1|      3.0|      1.0|     2.0|        0.0|
|Greater Mumbai, I...|Signature|  Bills|     F|302834|2014|       1|      1.0|      1.0|     2.0|        0.0|
|Greater Mumbai, I...| Platinum|  Bills|     F|647116|2014|       1|      1.0|      2.0|     2.0|        0.0|
|Greater Mumbai, I...|Signature|  Bills|     F|421878|2014|       1|      1.0|      1.0|     2.0|        0.0|
|Greater Mumbai, I...|     Gold|  Bills|     F|986379|2015|       1|      1.0|      3.0|     2.0|        0.0|
|Greater Mumbai, I...| Platinum|  Bills|     F|213047|2014|       1|      1.0|      2.0|     2.0|        0.0|
|Greater Mumbai, I...| Platinum|  Bills|     F|735566|2013|       1|      1.0|      2.0|     2.0|        0.0|
+--------------------+---------+-------+------+------+----+--------+---------+---------+--------+-----------+
only showing top 20 rows
```

We used vectoreAssembler to combine CardIndex, genderIndex , year and ExpIndex into one column (features) to train the machine learning model.

```
[68] assembler = VectorAssembler(
         inputCols=["CardIndex", "genderIndex", "year", "ExpIndex"],
         outputCol="features")
```

Here we assigned the label to be spending

```
[69] label_indexer = StringIndexer().setInputCol("spending").setOutputCol("label")
```

We have split the data, 80% for training and 20% for testing

```
[149] (trainingData, testData) = DF.randomSplit([0.8, 0.2])
```

We defined the stages of Pipeline which are label_indexer, assembler and dt and then we defined two variables, (model) to train the model and predictions to test the model .

```
[151] pipeline = Pipeline(stages=[label_indexer, assembler, dt])
```

```
[152] model = pipeline.fit(trainingData)
```

```
[153] predictions = model.transform(testData)
```

Now the results of our model.

```
predictions.select("prediction", "label", "features").show(20)
```

```
+----------+-----+-------------------+
|prediction|label|           features|
+----------+-----+-------------------+
|       0.0|  1.0|[3.0,1.0,2014.0,2.0]|
|       1.0|  1.0|[0.0,0.0,2013.0,4.0]|
|       0.0|  1.0|[2.0,0.0,2014.0,2.0]|
|       1.0|  1.0|[2.0,0.0,2015.0,2.0]|
|       1.0|  0.0|[0.0,0.0,2015.0,2.0]|
|       0.0|  1.0|[1.0,1.0,2014.0,2.0]|
|       1.0|  0.0|    (4,[2],[2015.0])|
|       1.0|  0.0|[1.0,0.0,2015.0,3.0]|
|       0.0|  0.0|[1.0,1.0,2014.0,0.0]|
|       0.0|  1.0|[0.0,1.0,2014.0,4.0]|
|       1.0|  0.0|[3.0,0.0,2015.0,2.0]|
|       0.0|  0.0|[3.0,0.0,2014.0,2.0]|
|       1.0|  1.0|[3.0,0.0,2015.0,2.0]|
|       1.0|  1.0|[3.0,0.0,2015.0,2.0]|
|       0.0|  1.0|[3.0,0.0,2013.0,2.0]|
|       0.0|  1.0|[3.0,0.0,2014.0,2.0]|
|       0.0|  1.0|[3.0,0.0,2014.0,2.0]|
|       0.0|  1.0|[3.0,0.0,2014.0,2.0]|
|       1.0|  1.0|[3.0,0.0,2015.0,2.0]|
|       0.0|  1.0|[3.0,0.0,2014.0,2.0]|
+----------+-----+-------------------+
```

To find the accuracy we used the multiclassClassificationEvaluator function with three columns label, prediction and accuracy. We can see that the error rate was 0.479404

```
[155] evaluator = MulticlassClassificationEvaluator(
          labelCol="label", predictionCol="prediction", metricName="accuracy")
```

```
[156] accuracy = evaluator.evaluate(predictions)
```

```
[264] print("Test Error = %g " % (1.0 - accuracy))

      Test Error = 0.479404
```

```
[158] treeModel = model.stages[2]
```

```
[159] print(treeModel)

      DecisionTreeClassificationModel: uid=DecisionTreeClassifier_3d46e6f408c6, depth=5, numNodes=41, numClasses=2, numFeatures=4
```

The confusion Matrix shows that we have
True positive =1957,
False positive =717,
False negative =1762,
And True negative =735

```
[161] y_pred=predictions.select("prediction").collect()
```

```
[162] y_orig=predictions.select("label").collect()
```

```
[271] cm = confusion_matrix(y_orig, y_pred)
      print("Confusion Matrix:")
      print(cm)

Confusion Matrix:
[[1957  717]
 [1762  735]]
```

Analysis :

The data set	We couldn't use the column city since it has 900 values and max bins is 32 so it couldn't fit all values
Training and testing	We initially divided the data into training and testing groups using the ratios of 0.7 and 0.3, however this provided the worst results (testing error was 0.5). We then altered the ratio to 0.8 for training and 0.2 for testing, which resulted in slightly better results (testing error was 0.47)
Testing error	We recognize that the accuracy is poor and that is due to two factors. First, not all columns were able to be features. Second, some of our features were generic so they weren't really useful.