Analysis

In [1]:
```
val df_tr = spark.read.option("delimiter","\t").
    option("header","true").
    csv("hdfs://mas.local:8020/dataset/term_rec/ICPSR_36404/DS0001/36404-0001-Data.tsv")
```

```
Starting Spark application
```

ID	YARN Application ID	Kind	State	Spark UI
63	application_1507101807067_0039	spark	idle	Link (http://S1.local:8088/proxy/application_15

```
SparkSession available as 'spark'.
df_tr: org.apache.spark.sql.DataFrame = [ABT_INMATE_ID: string, SEX: string ... 16 more fields]
```

In [2]:
```
df_tr.createOrReplaceTempView("term_rec")
```

In [3]:
```
spark.sql("select count(1) from term_rec where RELTYPE = '3' ").show()
```

```
+--------+
|count(1)|
+--------+
|  186462|
+--------+
```

dropping columns MAND_PRISREL_YEAR, PROJ_PRISREL_YEAR, PARELIG_YEAR,AGERELEASE as they contain a lot of missing values also dropping education as all values are missing

Removing rows reltype=3(indicating others)

In [4]:
```
spark.sql("select ABT_INMATE_ID,SEX,ADMTYPE,OFFGENERAL,ADMITYR,RELEASEYR,SENTLGTH,OFFDETAIL,RACE,AGEADMIT,TIMESRVD,RELTYPE,STATE from term_rec where RELTYPE!='3'").createOrReplaceTempView("term_rec_clV1")
```

In [5]:
```
spark.sql("select count(1) from term_rec_clV1 where RELTYPE = '3' ").
show()
```

```
+--------+
|count(1)|
+--------+
|       0|
+--------+
```

checkig if any admit year or release year are blank

In [6]:
```
spark.sql("select count(1) from term_rec_clV1 where ADMITYR = ' ' ").
show()
```

```
+--------+
|count(1)|
+--------+
|       0|
+--------+
```

In [7]:
```
spark.sql("select count(1) from term_rec_clV1 where RELEASEYR = ' ' "
).show()
```

```
+--------+
|count(1)|
+--------+
|       0|
+--------+
```

In [8]:
```
spark.sql("select count(1) from term_rec_clV1 where ADMITYR = '9999'"
).show()
```

```
+--------+
|count(1)|
+--------+
|     301|
+--------+
```

In [9]:
```
spark.sql("select count(1) from term_rec_clV1 where RELEASEYR = '999
9'").show()
```

```
+--------+
|count(1)|
+--------+
| 1200883|
+--------+
```

In [172]:
```
spark.sql("select count(1) from term_rec_clV1 where OFFDETAIL = '99'"
).show()
```

```
+--------+
|count(1)|
+--------+
|   64777|
+--------+
```

In [6]:
```
spark.sql("select * from term_rec_clV1").show()//add column names to
 data
```

```
+------------------+---+-------+----------+-------+---------+-------
-+---------+----+--------+--------+-------+-----+
|      ABT_INMATE_ID|SEX|ADMTYPE|OFFGENERAL|ADMITYR|RELEASEYR|SENTLGT
H|OFFDETAIL|RACE|AGEADMIT|TIMESRVD|RELTYPE|STATE|
+------------------+---+-------+----------+-------+---------+-------
-+---------+----+--------+--------+-------+-----+
|A022015000000096906|  1|      3|         3|   2008|     2008|
0|       12|   1|       3|       0|       |    2|
|A042015000000118649|  1|      1|         1|   2013|     2014|
0|        6|   1|       1|       0|      1|    4|
|A062015000000167469|  1|      2|         2|   1996|     1996|
2|        7|   1|       2|       0|      1|    6|
|A132015000000550479|  1|      1|         1|   1968|     1972|
3|        4|   1|       1|       2|      1|   13|
|A172015000000995085|  1|      1|         1|   2006|     2009|
3|        4|   2|       2|       2|      1|   17|
|A182015000000019353|  2|      3|         3|   2008|     2009|
2|       12|   1|       2|       0|      1|   18|
|A202015000000031529|  1|      1|         3|   2013|     2014|
1|       12|   1|       3|       1|      2|   20|
|A212015000000115891|  1|      2|         4|   2014|     2014|
2|       13|   9|       1|       0|       |   21|
|A232015000000003075|  1|      1|         1|   2012|     2014|
2|        6|   1|       4|       1|       |   23|
|A242015000000177833|  1|      1|         1|   1999|     9999|
5|        1|   9|       1|       9|       |   24|
|A262015000000320390|  2|      1|         3|   1988|     1997|
5|       12|   9|       2|       3|      1|   26|
|A272015000000050042|  1|      2|         4|   1999|     1999|
1|       13|   2|       3|       0|      1|   27|
|A282015000000019112|  1|      2|         1|   2011|     2012|
4|        5|   2|       2|       0|      1|   28|
|A302015000000002898|  1|      1|         1|   2007|     9999|
6|        1|   3|       1|       9|       |   30|
|A312015000000040102|  1|      1|         3|   2003|     2003|
2|       12|   1|       3|       0|      1|   31|
|A332015000000007173|  1|      2|         4|   2013|     2014|
3|       13|   1|       3|       0|      1|   33|
|A342015000000095444|  2|      1|         1|   2011|     2012|
2|        6|   2|       2|       0|      1|   34|
|A352015000000007367|  1|      2|         2|   2014|     9999|
2|        9|   3|       1|       9|       |   35|
|A362015000000174083|  1|      1|         4|   2012|     2014|
2|       13|   2|       1|       2|      1|   36|
|A372015000000223146|  1|      1|         4|   2012|     9999|
3|       13|   2|       3|       9|       |   37|
+------------------+---+-------+----------+-------+---------+-------
-+---------+----+--------+--------+-------+-----+
only showing top 20 rows
```

selecting id, admit year, release year and arranging them

In [6]:
```
spark.sql("select ABT_INMATE_ID,ADMITYR,RELEASEYR,SEX,SENTLGTH,RACE,A
GEADMIT,TIMESRVD,RELTYPE,STATE,ADMTYPE,OFFGENERAL,OFFDETAIL from term
_rec_clV1 group by ABT_INMATE_ID,ADMITYR,RELEASEYR,SEX,SENTLGTH,RACE,
AGEADMIT,TIMESRVD,RELTYPE,STATE,ADMTYPE,OFFGENERAL,OFFDETAIL order by
 ADMITYR,RELEASEYR ").createOrReplaceTempView("id_yr")
```

In [8]:
```
// statistics of offdetail in term_rec_clV1
spark.sql("select count(OFFDETAIL) as Countof,OFFDETAIL from term_rec
_clV1 group by OFFDETAIL order by cast(OFFDETAIL as int)").show()
```

```
+-------+---------+
|Countof|OFFDETAIL|
+-------+---------+
| 285377|        1|
|  63500|        2|
| 532786|        3|
| 861479|        4|
| 959018|        5|
| 224945|        6|
|1250647|        7|
| 772586|        8|
| 285448|        9|
| 436178|       10|
| 361888|       11|
|2994983|       12|
|1543560|       13|
|  83699|       14|
|  64777|       99|
+-------+---------+
```

In [9]:
```
// showin   records for random 2 inmate ids of idyr
spark.sql("select * from id_yr where ABT_INMATE_ID = 'A36201500000020
5625' OR ABT_INMATE_ID = 'A182015000000208583'").show()
```

```
+------------------+-------+---------+---+--------+----+--------+---
-----+-------+-----+-------+----------+---------+
|      ABT_INMATE_ID|ADMITYR|RELEASEYR|SEX|SENTLGTH|RACE|AGEADMIT|TIM
ESRVD|RELTYPE|STATE|ADMTYPE|OFFGENERAL|OFFDETAIL|
+------------------+-------+---------+---+--------+----+--------+---
-----+-------+-----+-------+----------+---------+
|A182015000000208583|   1999|     2004|  1|       3|   1|       2|
    3|      1|   18|      9|         3|       12|
|A182015000000208583|   2004|     2005|  1|       3|   1|       3|
    0|      1|   18|      2|         3|       12|
|A362015000000205625|   2004|     2007|  1|       2|   2|       1|
    2|      1|   36|      1|         1|        4|
|A362015000000205625|   2009|     2010|  1|       2|   2|       2|
    0|      1|   36|      2|         1|        4|
|A362015000000205625|   2013|     9999|  1|       2|   2|       2|
    9|       |   36|      1|         3|       12|
+------------------+-------+---------+---+--------+----+--------+---
-----+-------+-----+-------+----------+---------+
```

adding indexes and selecting in particular order

In [7]:
```
//ABT_INMATE_ID,ADMITYR,RELEASEYR,SEX,SENTLGTH,RACE,AGEADMIT,TIMESRVD,RELTYPE,STATE,ADMTYPE,OFFGENERAL,OFFDETAIL
spark.sql("select ABT_INMATE_ID,ADMITYR,RELEASEYR,idx1,idx2,SEX,SENTLGTH,RACE,AGEADMIT,TIMESRVD,RELTYPE,STATE,ADMTYPE,OFFGENERAL,OFFDETAIL
 from(select *, row_number() over (partition by ABT_INMATE_ID order by ADMITYR,RELEASEYR ) as idx1,((row_number() over (partition by ABT_INMATE_ID order by ADMITYR,RELEASEYR ))+1)as idx2 from id_yr)").createOrReplaceTempView("id_yr_idx")
```

In [11]:
```
spark.sql("select * from id_yr_idx where ABT_INMATE_ID = 'A362015000000205625' OR ABT_INMATE_ID = 'A182015000000208583'").show()
```

```
+------------------+-------+---------+----+----+---+--------+----+------+--------+-------+-----+-------+----------+---------+
|      ABT_INMATE_ID|ADMITYR|RELEASEYR|idx1|idx2|SEX|SENTLGTH|RACE|AGEADMIT|TIMESRVD|RELTYPE|STATE|ADMTYPE|OFFGENERAL|OFFDETAIL|
+------------------+-------+---------+----+----+---+--------+----+------+--------+-------+-----+-------+----------+---------+
|A362015000000205625|   2004|     2007|   1|   2|  1|       2|   2|     1|       2|      1|   36|      1|         1|        4|
|A362015000000205625|   2009|     2010|   2|   3|  1|       2|   2|     2|       0|      1|   36|      2|         1|        4|
|A362015000000205625|   2013|     9999|   3|   4|  1|       2|   2|     2|       9|       |   36|      1|         3|       12|
|A182015000000208583|   1999|     2004|   1|   2|  1|       3|   1|     2|       3|      1|   18|      9|         3|       12|
|A182015000000208583|   2004|     2005|   2|   3|  1|       3|   1|     3|       0|      1|   18|      2|         3|       12|
+------------------+-------+---------+----+----+---+--------+----+------+--------+-------+-----+-------+----------+---------+
```

In [8]:
```
val idyr_idx_rdd= spark.sql("select * from id_yr_idx").rdd
```

```
idyr_idx_rdd: org.apache.spark.rdd.RDD[org.apache.spark.sql.Row] = MapPartitionsRDD[40] at rdd at <console>:23
```

In [9]:
```
idyr_idx_rdd.take(10)
```

```
res8: Array[org.apache.spark.sql.Row] = Array([A012015000000000158,2003,2007,1,2,1,3,1,2,2,1,1,9,2,7], [A012015000000000158,2013,9999,2,3,1,4,1,3,9, ,1,3,1,5], [A012015000000000593,2006,2007,1,2,1,3,1,1,0,1,1,9,3,12], [A012015000000000593,2007,2008,2,3,1,3,1,2,0,2,1,1,3,12], [A012015000000000593,2012,2012,3,4,1,0,1,2,0,1,1,1,4,13], [A012015000000000593,2014,9999,4,5,1,4,1,2,9, ,1,1,4,13], [A012015000000000886,2006,2007,1,2,1,4,1,3,1,1,1,9,2,7], [A012015000000001131,2007,2009,1,2,1,2,1,1,1,1,1,3,3,12], [A012015000000001260,2007,2008,1,2,1,2,1,1,0,2,1,1,2,8], [A012015000000001560,2007,2009,1,2,1,4,9,3,1,1,1,2,1,4])
```

In [10]:
```
idyr_idx_rdd.map(line=>(line.getString(0),(line.getString(1).toInt,li
ne.getString(2).toInt,line.getInt(3),line.getInt(4),
                                                        line.getS
tring(5),line.getString(6),line.getString(7),
                                                         line.get
String(8),line.getString(9),line.getString(10),
                                                        line.getS
tring(11),line.getString(12),line.getString(13),
                                                        line.getS
tring(14)))).take(4)
```

```
res9: Array[(String, (Int, Int, Int, Int, String, String, String, Str
ing, String, String, String, String, String, String))] = Array((A0120
15000000000158,(2003,2007,1,2,1,3,1,2,2,1,1,9,2,7)), (A01201500000000
0158,(2013,9999,2,3,1,4,1,3,9," ",1,3,1,5)), (A012015000000000593,(20
06,2007,1,2,1,3,1,1,0,1,1,9,3,12)), (A012015000000000593,(2007,2008,
2,3,1,3,1,2,0,2,1,1,3,12)))
```

Grouping by ID with key= Inmate ID and Values= (remaining values)

In [11]:
```
val idyr_idx_grp =idyr_idx_rdd.map(line=>(line.getString(0),(line.get
String(1),line.getString(2),line.getInt(3).toString,line.getInt(4).to
String,
                                                        line.getS
tring(5),line.getString(6),line.getString(7),
                                                         line.get
String(8),line.getString(9),line.getString(10),
                                                        line.getS
tring(11),line.getString(12),line.getString(13),
                                                        line.getS
tring(14)))).
groupByKey
```

```
idyr_idx_grp: org.apache.spark.rdd.RDD[(String, Iterable[(String, Str
ing, String, String, String, String, String, String, String, String,
String, String, String, String)])] = ShuffledRDD[43] at groupByKey at
<console>:30
```

In [12]:
```
idyr_idx_grp.take(4)
```

```
res10: Array[(String, Iterable[(String, String, String, String, Strin
g, String, String, String, String, String, String, String, String, St
ring)])] = Array((A542015000000010283,CompactBuffer((2014,9999,1,2,1,
2,1,1,9, ,54,1,1,3))), (A482015000000305191,CompactBuffer((2012,2012,
1,2,1,0,1,1,0,2,48,1,2,8))), (A182015000000208583,CompactBuffer((199
9,2004,1,2,1,3,1,2,3,1,18,9,3,12), (2004,2005,2,3,1,3,1,3,0,1,18,2,3,
12))), (A362015000000205625,CompactBuffer((2004,2007,1,2,1,2,2,1,2,1,
36,1,1,4), (2009,2010,2,3,1,2,2,2,0,1,36,2,1,4), (2013,9999,3,4,1,2,
2,2,9, ,36,1,3,12))))
```

Values will be displayed as array(array(Int)=> this corresponds to indiviual row entry)

In [13]:
```
idyr_idx_grp.mapValues(it=> it.toArray).take(4)
```

```
res11: Array[(String, Array[(String, String, String, String, String,
String, String, String, String, String, String, String, String, Strin
g)])] = Array((A062015000000785083,Array((2004,2007,1,2,1,2,1,2,2,1,
6,1,2,7))), (A482015000000305191,Array((2012,2012,1,2,1,0,1,1,0,2,48,
1,2,8))), (A542015000000010283,Array((2014,9999,1,2,1,2,1,1,9," ",54,
1,1,3))), (A362015000000205625,Array((2004,2007,1,2,1,2,2,1,2,1,36,1,
1,4), (2009,2010,2,3,1,2,2,2,0,1,36,2,1,4), (2013,9999,3,4,1,2,2,2,
9," ",36,1,3,12))))
```

In [14]:
```
idyr_idx_grp.mapValues(it=> it.toArray).mapValues{case(a)=>a.map{case
 (a)=>Array(a._1,a._2,a._3,a._4,a._5,a._6,a._7,a._8,a._9,a._10,a._11,
a._12,a._13,a._14)}}.take(4)
```

```
res12: Array[(String, Array[Array[String]])] = Array((A06201500000078
5083,Array(Array(2004, 2007, 1, 2, 1, 2, 1, 2, 2, 1, 6, 1, 2, 7))),
(A482015000000305191,Array(Array(2012, 2012, 1, 2, 1, 0, 1, 1, 0, 2,
48, 1, 2, 8))), (A542015000000010283,Array(Array(2014, 9999, 1, 2, 1,
2, 1, 1, 9, " ", 54, 1, 1, 3))), (A362015000000205625,Array(Array(200
4, 2007, 1, 2, 1, 2, 2, 1, 2, 1, 36, 1, 1, 4), Array(2009, 2010, 2,
3, 1, 2, 2, 2, 0, 1, 36, 2, 1, 4), Array(2013, 9999, 3, 4, 1, 2, 2,
2, 9, " ", 36, 1, 3, 12))))
```

In [15]:
```
val idyr_idx_arr=idyr_idx_grp.mapValues(it=> it.toArray).mapValues{ca
se(a)=>a.map{case (a)=>Array(a._1,a._2,a._3,a._4,a._5,a._6,a._7,a._8,
a._9,a._10,a._11,a._12,a._13,a._14)}}
```

```
idyr_idx_arr: org.apache.spark.rdd.RDD[(String, Array[Array[Strin
g]])] = MapPartitionsRDD[48] at mapValues at <console>:27
```

In [16]:
```
idyr_idx_arr.take(5)
```

```
res13: Array[(String, Array[Array[String]])] = Array((A54201500000001
0283,Array(Array(2014, 9999, 1, 2, 1, 2, 1, 1, 9, " ", 54, 1, 1,
3))), (A482015000000305191,Array(Array(2012, 2012, 1, 2, 1, 0, 1, 1,
0, 2, 48, 1, 2, 8))), (A182015000000208583,Array(Array(1999, 2004, 1,
2, 1, 3, 1, 2, 3, 1, 18, 9, 3, 12), Array(2004, 2005, 2, 3, 1, 3, 1,
3, 0, 1, 18, 2, 3, 12))), (A362015000000205625,Array(Array(2004, 200
7, 1, 2, 1, 2, 2, 1, 2, 1, 36, 1, 1, 4), Array(2009, 2010, 2, 3, 1,
2, 2, 2, 0, 1, 36, 2, 1, 4), Array(2013, 9999, 3, 4, 1, 2, 2, 2, 9, "
", 36, 1, 3, 12))), (A022015000000099762,Array(Array(2010, 2010, 1,
2, 1, 0, 4, 1, 0, 2, 2, 9, 4, 13), Array(2010, 2011, 2, 3, 1, 0, 4,
1, 0, 2, 2, 9, 4, 13), Array(2012, 2012, 3, 4, 1, 0, 4, 1, 0, 2, 2,
9, 4, 13))))
```

Filter out Inamte IDs with one entry only

idyr_idx_arr1 contains entries of IDs with only 1 row(term record) . It Has admt_yr,rel_yr,recidivism=0

idyr_idx_arr2 contains entries of IDs with more than 1 row(term record)

In [17]:
```
val idyr_idx_arr1=idyr_idx_arr.filter{case(id,arr)=>arr.size==1}.mapV
alues{case(arr)=>(arr(0)(0),arr(0)(1),0)}
```

```
idyr_idx_arr1: org.apache.spark.rdd.RDD[(String, (String, String, In
t))] = MapPartitionsRDD[50] at mapValues at <console>:29
```

In [18]:
```
idyr_idx_arr1.take(4)
```

```
res14: Array[(String, (String, String, Int))] = Array((A5420150000000
10283,(2014,9999,0)), (A482015000000305191,(2012,2012,0)), (A06201500
0002049552,(1993,1994,0)), (A042015000000027196,(2008,2011,0)))
```

In [19]:
```
val idyr_idx_arr2=idyr_idx_arr.filter{case(id,arr)=>arr.size!=1}
```

```
idyr_idx_arr2: org.apache.spark.rdd.RDD[(String, Array[Array[Strin
g]])] = MapPartitionsRDD[51] at filter at <console>:29
```

In [20]:
```
idyr_idx_arr2.take(4)
```

```
res15: Array[(String, Array[Array[String]])] = Array((A36201500000020
5625,Array(Array(2004, 2007, 1, 2, 1, 2, 2, 1, 2, 1, 36, 1, 1, 4), Ar
ray(2009, 2010, 2, 3, 1, 2, 2, 2, 0, 1, 36, 2, 1, 4), Array(2013, 999
9, 3, 4, 1, 2, 2, 2, 9, " ", 36, 1, 3, 12))), (A022015000000099762,Ar
ray(Array(2010, 2010, 1, 2, 1, 0, 4, 1, 0, 2, 2, 9, 4, 13), Array(201
0, 2011, 2, 3, 1, 0, 4, 1, 0, 2, 2, 9, 4, 13), Array(2012, 2012, 3,
4, 1, 0, 4, 1, 0, 2, 2, 9, 4, 13))), (A182015000000208583,Array(Array
(1999, 2004, 1, 2, 1, 3, 1, 2, 3, 1, 18, 9, 3, 12), Array(2004, 2005,
2, 3, 1, 3, 1, 3, 0, 1, 18, 2, 3, 12))), (A482015000000346105,Array(A
rray(2008, 2010, 1, 2, 1, 2, 3, 1, 1, 2, 48, 1, 3, 12), Array(2012, 2
014, 2, 3, 1, 3, 3, 1, 2, 1, 48, 2, 3, 12))))
```

In [21]:
```
idyr_idx_arr2.mapValues(s =>{var i=0;var j=0;
                             for(i <-0 to s.length-2;j<-1 to s.length-1
;if(s(i)(3)==s(j)(2)))
                              yield (s(i)(1),s(j)(0),s(j)(0).toInt-s(i)
(1).toInt)
                             }).take(3)
```

```
res16: Array[(String, scala.collection.immutable.IndexedSeq[(String,
String, Int)])] = Array((A362015000000205625,Vector((2007,2009,2), (2
010,2013,3))), (A022015000000099762,Vector((2010,2010,0), (2011,2012,
1))), (A182015000000208583,Vector((2004,2004,0))))
```

ABT_INMATE_ID|ADMITYR|RELEASEYR|idx1|idx2|SEX|SENTLGTH|RACE|AGEADMIT|TIMESRVD|RELTYPE

key=ABT_INMATE_ID

values=0ADMITYR,1RELEASEYR,2idx1,3idx2,4SEX,

5SENTLGTH,6RACE,7AGEADMIT,8TIMESRVD,9RELTYPE,10STATE,

11ADMTYPE,12OFFGENERAL,13OFFDETAIL

this term=>next offdetail next admtype

0 to 12=> 13 11

In [22]:
```
idyr_idx_arr2.mapValues(s =>{var i=0;var j=0;
                            for(i <-(0 to s.length-2).toArray;j<-(1 to
 s.length-1).toArray;if(s(i)(3)==s(j)(2)))
                           yield (s(j-1)(0),s(j-1)(1),s(j-1)(4),s(j-1)
(5),
                                  s(j-1)(6),s(j-1)(7),s(j-1)(8),s(j-1
)(9),s(j-1)(10),
                                   s(j-1)(11),s(j-1)(12),s(j-1)(13),
                                 s(j)(13),s(j)(11))
                            }).take(3) //diff=s(j)(0).toInt-s(i)(1).to
Int
```

```
res17: Array[(String, Array[(String, String, String, String, String,
String, String, String, String, String, String, String, String, Strin
g)])] = Array((A182015000000208583,Array((1999,2004,1,3,1,2,3,1,18,9,
3,12,12,2))), (A362015000000205625,Array((2004,2007,1,2,2,1,2,1,36,1,
1,4,4,2), (2009,2010,1,2,2,2,0,1,36,2,1,4,12,1))), (A0220150000000997
62,Array((2010,2010,1,0,4,1,0,2,2,9,4,13,13,9), (2010,2011,1,0,4,1,0,
2,2,9,4,13,13,9))))
```

In [23]:
```
val yr_diff_rdd=idyr_idx_arr2.mapValues(s =>{var i=0;var j=0;
                            for(i <-(0 to s.length-2).toArray;j<-(1 to
 s.length-1).toArray;if(s(i)(3)==s(j)(2)))
                             yield (s(j-1)(0),s(j-1)(1),s(j-1)(4),s(j-
1)(5),
                                   s(j-1)(6),s(j-1)(7),s(j-1)(8),s(j-1
)(9),s(j-1)(10),
                                    s(j-1)(11),s(j-1)(12),s(j-1)(13),
                                  s(j)(13),s(j)(11))
                            })
```

```
yr_diff_rdd: org.apache.spark.rdd.RDD[(String, Array[(String, String,
String, String, String, String, String, String, String, String, Strin
g, String, String, String)])] = MapPartitionsRDD[54] at mapValues at
<console>:31
```

In [24]:
```
yr_diff_rdd.take(5)
```

```
res18: Array[(String, Array[(String, String, String, String, String,
String, String, String, String, String, String, String, String, Strin
g)])] = Array((A362015000000205625,Array((2004,2007,1,2,2,1,2,1,36,1,
1,4,4,2), (2009,2010,1,2,2,2,0,1,36,2,1,4,12,1))), (A0220150000000997
62,Array((2010,2010,1,0,4,1,0,2,2,9,4,13,13,9), (2010,2011,1,0,4,1,0,
2,2,9,4,13,13,9))), (A182015000000208583,Array((1999,2004,1,3,1,2,3,
1,18,9,3,12,12,2))), (A482015000000346105,Array((2008,2010,1,2,3,1,1,
2,48,1,3,12,12,2))), (A482015000000949142,Array((1991,2009,1,4,2,2,4,
1,48,2,2,7,12,1), (2010,2011,1,0,2,4,0," ",48,1,3,12,13,1))))
```

In [25]:
```
//##### key=ABT_INMATE_ID
//values=0ADMITYR,1RELEASEYR,2idx1,3idx2,4SEX,
// 5SENTLGTH,6RACE,7AGEADMIT,8TIMESRVD,9RELTYPE,10STATE,
// 11ADMTYPE,12OFFGENERAL,13OFFDETAIL
val cols = Seq("ABT_INMATE_ID","ADMITYR","RELEASEYR","SEX","SENTLGTH"
,
               "RACE","AGEADMIT","TIMESRVD","RELTYPE","STATE",
               "ADMTYPE","OFFGENERAL","OFFDETAIL","nt_OFFDETAIL","nt_
ADMTYPE","Recidivism")
val yr_recividated=yr_diff_rdd.flatMapValues(x => x).map{case(id,(tt
))=>(id,tt._1,tt._2,tt._3,tt._4,tt._5,tt._6,tt._7,tt._8,tt._9,tt._10,
tt._11,tt._12,tt._13,tt._14,1)}.toDF(cols: _*) //
```

```
yr_recividated: org.apache.spark.sql.DataFrame = [ABT_INMATE_ID: stri
ng, ADMITYR: string ... 14 more fields]
```

In [ ]:
```
yr_recividated.show()
```

In [27]:
```
yr_recividated.createOrReplaceTempView("yr_recividated_view")
```

reemoving missing values

In [28]:
```
spark.sql("""select * from (select * from yr_recividated_view where R
ELEASEYR != '9999' AND ADMTYPE != '9' AND OFFGENERAL != '9'
AND ADMITYR != '9999' AND OFFDETAIL != '99' AND RACE != '9' AND AGEAD
MIT != '9' AND SENTLGTH != '9'
AND RELTYPE != '9' AND TIMESRVD != '9' AND RELTYPE != ' ' AND SENTLGT
H != ' ' AND nt_ADMTYPE!='9'
AND nt_ADMTYPE!=' ' AND nt_OFFDETAIL!='99' AND nt_OFFDETAIL!=' ' )a
 """).createOrReplaceTempView("nm_yr_recividated_view")
```

STATISTICAL ANALYSIS

ANALYSIS Steps to calculate percentage of offence repeated

In [29]:
```
//Total rows:
val total_rows=spark.sql("select count(1) from nm_yr_recividated_vie
w")
```

```
total_rows: org.apache.spark.sql.DataFrame = [count(1): bigint]
```

In [30]:
```
var tot_rows=total_rows.rdd.first().getLong(0)
```

```
tot_rows: Long = 3783411
```

In [31]:
```
// statistics of offdetail in nm_yr_recividated_view
spark.sql("select OFFDETAIL,nt_OFFDETAIL,count(*) as CountOf from nm_
yr_recividated_view group by OFFDETAIL,nt_OFFDETAIL order by cast(Cou
ntOf as int) DESC").show()
```

```
+---------+------------+-------+
|OFFDETAIL|nt_OFFDETAIL|CountOf|
+---------+------------+-------+
|       12|          12| 882528|
|        7|           7| 350014|
|       13|          13| 336615|
|        5|           5| 216388|
|        8|           8| 214278|
|        4|           4| 196026|
|        3|           3|  91756|
|       10|          10|  91402|
|        9|           9|  90143|
|       11|          11|  79317|
|       12|          13|  66931|
|       13|          12|  55249|
|        6|           6|  42971|
|        7|          12|  41114|
|       12|           7|  32121|
|       12|           5|  31382|
|       12|           8|  29303|
|        8|          12|  29050|
|        5|          12|  28674|
|        5|          13|  28079|
+---------+------------+-------+
only showing top 20 rows
```

offdetail 12 12 makes about 23.32 % of the total data and about 33.4% of the 69.8 % repeated offence records

In [421]:
```
%%sql
select OFFDETAIL,nt_OFFDETAIL,count(*) as CountOf from nm_yr_recivida
ted_view group by OFFDETAIL,nt_OFFDETAIL order by cast(CountOf as int
) DESC
```

In [33]:
```
// 1 -offence repeated , 0 - offence not repeated
spark.sql(s"""SELECT Off_Repeated,count(*) as CountOf,((count(*)/$tot_rows)*100) as Percentage from
(SELECT OFFDETAIL,nt_OFFDETAIL,(CASE WHEN OFFDETAIL=nt_OFFDETAIL then
 1 ELSE 0 END)
as Off_Repeated FROM nm_yr_recividated_view) group by Off_Repeated""").show()
```

```
+------------+-------+------------------+
|Off_Repeated|CountOf|        Percentage|
+------------+-------+------------------+
|           1|2642043| 69.83230212102254|
|           0|1141368|30.167697878977464|
+------------+-------+------------------+
```

In [420]:
```
%%sql
SELECT Off_Repeated,count(*) as CountOf from
(SELECT OFFDETAIL,nt_OFFDETAIL,(CASE WHEN OFFDETAIL=nt_OFFDETAIL then
 1 ELSE 0 END)
 as Off_Repeated FROM nm_yr_recividated_view)
group by Off_Repeated
```

Percentange of offence repeat = 2642093/3783412 = 0.69833605222

##

Analysis Steps for Percentage of repeated offence(1) or not(0) w.r.t ADMTYPE(1,2,3)

In [35]:
```
spark.sql("select OFFDETAIL,nt_OFFDETAIL, nt_ADMTYPE, count(*) as Cou
ntOf from nm_yr_recividated_view group by OFFDETAIL,nt_OFFDETAIL,nt_A
DMTYPE order by cast(CountOf as int) DESC").show()
```

```
+---------+------------+----------+-------+
|OFFDETAIL|nt_OFFDETAIL|nt_ADMTYPE|CountOf|
+---------+------------+----------+-------+
|       12|          12|         2| 620526|
|        7|           7|         2| 262126|
|       12|          12|         1| 257045|
|       13|          13|         2| 212127|
|        5|           5|         2| 168548|
|        4|           4|         2| 159715|
|        8|           8|         2| 151170|
|       13|          13|         1| 120880|
|        7|           7|         1|  86060|
|        9|           9|         2|  81857|
|        3|           3|         2|  73090|
|        8|           8|         1|  61791|
|       11|          11|         2|  60425|
|       10|          10|         2|  59478|
|       12|          13|         1|  53268|
|        5|           5|         1|  46698|
|       13|          12|         1|  42952|
|        6|           6|         2|  36676|
|        4|           4|         1|  35430|
|       10|          10|         1|  31173|
+---------+------------+----------+-------+
only showing top 20 rows
```

In [36]:
```
spark.sql(s"""select nt_ADMTYPE,case_cond,count(1) as CountOf,((CAST
(count(1) AS DOUBLE) / CAST($tot_rows AS DOUBLE))*100) as Percentage
from  (Select nt_ADMTYPE,( CASE WHEN OFFDETAIL=nt_OFFDETAIL then 1 EL
SE 0 END) as case_cond from nm_yr_recividated_view)a
 group by case_cond,nt_ADMTYPE order by cast(nt_ADMTYPE as int),cast
(case_cond as int)""").show()
```

```
+----------+---------+-------+------------------+
|nt_ADMTYPE|case_cond|CountOf|        Percentage|
+----------+---------+-------+------------------+
|         1|        0| 861441| 22.768898224380063|
|         1|        1| 699802| 18.496589453273778|
|         2|        0| 272539|  7.203526130256534|
|         2|        1|1925952|  50.90517525058736|
|         3|        0|   7388|0.19527352434086595|
|         3|        1|  16289|0.43053741716139216|
+----------+---------+-------+------------------+
```

ADMTYPE	Label
1	New court commitment (New cc)
2	Parole return/revocation (Parole r)
3	Other admission (including unsentenced, transfer, AWOL/escapee return)

nt_ADMTYPE	case_cond	CountOf	Percentage	Description
1	0	861441	22.768898224380063	New cc recidivated on different offence
1	1	699802	18.496589453273778	New cc recidivated on same offence
2	0	272539	7.203526130256534	Parole r recidivated on different offence
2	1	1925952	50.90517525058736	Parole r recidivated on same offence
3	0	7388	0.19527352434086595	other recidivated on different offence
3	1	16289	0.43053741716139216	other recidivated on same offence

In [422]:
```
%%sql
select CONCAT(nt_ADMTYPE,' ',case_cond) as adm_off_repeated_or_not,co
unt(1) as CountOf
from  (Select nt_ADMTYPE,( CASE WHEN OFFDETAIL=nt_OFFDETAIL then 1 EL
SE 0 END) as case_cond from nm_yr_recividated_view)a
 group by case_cond,nt_ADMTYPE order by cast(nt_ADMTYPE as int),cast(
case_cond as int)
```

Inmates who came back with difrrent offence for next term for new court commit or parol return i.e 10 and 20 , the orange red shaded area

In [42]:
```
spark.sql("select * from nm_yr_recividated_view where OFFDETAIL != nt
_OFFDETAIL and (nt_ADMTYPE == '1' or nt_ADMTYPE == '2') and  CAST(REL
EASEYR AS INT) < 2012 ").write.format("com.databricks.spark.csv").sav
e("hdfs://mas.local:8020/data/PrisDataOrangeRed.csv")
```

In [194]:
```
val cols = Seq("ADMITYR","RELEASEYR","SEX","SENTLGTH",
               "RACE","AGEADMIT","TIMESRVD","RELTYPE","STATE",
               "ADMTYPE","OFFGENERAL","OFFDETAIL","nt_OFFDETAIL","nt_
ADMTYPE")
val orange_red_df=spark.read.
    csv("hdfs://mas.local:8020/data/PrisDataOrangeRed.csv").map(t=>(t
.getString(1).toInt,t.getString(2).toInt,
                                                                t
.getString(3).toInt,t.getString(4).toInt,
                                                                t
.getString(5).toInt,t.getString(6).toInt,
                                                                t
.getString(7).toInt,t.getString(8).toInt,
                                                                t
.getString(9).toInt,t.getString(10).toInt,
                                                                t
.getString(11).toInt,t.getString(12).toInt,
                                                                t
.getString(13).toInt,t.getString(14).toInt)).toDF(cols: _*)
```

```
orange_red_df: org.apache.spark.sql.DataFrame = [ADMITYR: int, RELEAS
EYR: int ... 12 more fields]
```

ABT_INMATE_ID|ADMITYR|RELEASEYR|SEX|SENTLGTH|RACE|AGEADMIT|TIMESRVD|RELTYPE|STATE|A

In [195]:
```
orange_red_df.take(4)
```

```
res153: Array[org.apache.spark.sql.Row] = Array([2003,2004,1,1,2,2,0,
2,37,1,3,12,13,1], [2004,2006,1,3,2,2,1,1,18,1,4,13,5,1], [1994,2008,
1,5,1,1,4,1,48,2,2,7,4,2], [2001,2001,1,0,2,2,0,2,37,1,3,12,13,1])
```

In [375]:
```
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.classification.RandomForestClassifier
import org.apache.spark.ml.classification.RandomForestClassificationM
odel
import org.apache.spark.ml.feature.{StringIndexer, IndexToString, Vec
torIndexer}
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluat
or
import org.apache.spark.mllib.util.MLUtils
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.mllib.evaluation.MulticlassMetrics
```

```
import org.apache.spark.mllib.evaluation.MulticlassMetrics
```

In [377]:
```
// Index labels, adding metadata to the label column.
// Fit on whole dataset to include all labels in index.
val labelIndexer = new StringIndexer().
setInputCol("nt_OFFDETAIL").
setOutputCol("Pred_nt_OFFDETAIL").
fit(orange_red_df)
```

```
labelIndexer: org.apache.spark.ml.feature.StringIndexerModel = strIdx
_af01b695fc61
```

In [376]:
```
//Making a feature column
val assembler = new VectorAssembler().
setInputCols(Array("ADMITYR","RELEASEYR","SEX","SENTLGTH",
                "RACE","AGEADMIT","TIMESRVD","RELTYPE","STATE",
                "ADMTYPE","OFFGENERAL","OFFDETAIL","nt_ADMTYPE")).
setOutputCol("features")
val orange_red_feature_df=assembler.transform(orange_red_df)
```

```
orange_red_feature_df: org.apache.spark.sql.DataFrame = [ADMITYR: in
t, RELEASEYR: int ... 13 more fields]
```

In [378]:
```
// Automatically identify categorical features, and index them.
val featureIndexer = new VectorIndexer().
setInputCol("features").
setOutputCol("indexedFeatures").
fit(orange_red_feature_df)
```

```
featureIndexer: org.apache.spark.ml.feature.VectorIndexerModel = vecI
dx_01d129525528
```

In [379]:
```
// Index labels, adding metadata to the label column.
// Fit on whole dataset to include all labels in index.
val labelIndexer = new StringIndexer().
setInputCol("nt_OFFDETAIL").
setOutputCol("Pred_nt_OFFDETAIL").
fit(orange_red_df)
```

```
labelIndexer: org.apache.spark.ml.feature.StringIndexerModel = strIdx
_1e85b3109815
```

In [394]:
```
// Split the data into training and test sets
val splitSeed = 456
val Array(trainingData, testData) = orange_red_feature_df.randomSplit
(Array(0.6, 0.4), splitSeed)
```

```
trainingData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row]
= [ADMITYR: int, RELEASEYR: int ... 13 more fields]
testData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [A
DMITYR: int, RELEASEYR: int ... 13 more fields]
```

In [395]:
```
// Train a RandomForest model.
val rf = new RandomForestClassifier().
setLabelCol("Pred_nt_OFFDETAIL").
setFeaturesCol("indexedFeatures")
```

```
rf: org.apache.spark.ml.classification.RandomForestClassifier = rfc_4
2a204db795f
```

In [396]:
```
// Convert indexed labels back to original labels.
val labelConverter = new IndexToString().
setInputCol("prediction").
setOutputCol("predictedLabel").
setLabels(labelIndexer.labels)
```

```
labelConverter: org.apache.spark.ml.feature.IndexToString = idxToStr_
ea51bce7cf04
```

In [397]:
```
// Chain indexers and forest in a Pipeline
val pipeline = new Pipeline().
setStages(Array(labelIndexer, featureIndexer, rf, labelConverter))
```

```
pipeline: org.apache.spark.ml.Pipeline = pipeline_22968c12ee15
```

In [398]:
```
// Train model.  This also runs the indexers.
val model = pipeline.fit(trainingData)
```

```
model: org.apache.spark.ml.PipelineModel = pipeline_22968c12ee15
```

In [399]:
```
// Make predictions.
val predictions = model.transform(testData)
```

```
predictions: org.apache.spark.sql.DataFrame = [ADMITYR: int, RELEASEY
R: int ... 19 more fields]
```

In [400]:
```
// Select example rows to display.
predictions.select("predictedLabel", "nt_OFFDETAIL", "features").show
(5)
```

```
+--------------+------------+--------------------+
|predictedLabel|nt_OFFDETAIL|            features|
+--------------+------------+--------------------+
|            12|          13|[1957.0,1971.0,1....|
|            13|           9|[1961.0,1971.0,1....|
|            13|           2|[1962.0,1971.0,1....|
|            13|           4|[1962.0,1972.0,1....|
|            13|           9|[1963.0,1971.0,1....|
+--------------+------------+--------------------+
only showing top 5 rows
```

In [401]:
```
predictions.select("predictedLabel", "nt_OFFDETAIL").rdd.map(x =>
  (x(0).asInstanceOf[String].toDouble, x(1).asInstanceOf[Int].toDoubl
e)).take(20)
```

```
res346: Array[(Double, Double)] = Array((12.0,13.0), (13.0,9.0), (13.
0,2.0), (13.0,4.0), (13.0,9.0), (12.0,13.0), (12.0,5.0), (12.0,7.0),
(12.0,11.0), (13.0,4.0), (12.0,13.0), (13.0,8.0), (12.0,13.0), (12.0,
12.0), (12.0,10.0), (13.0,13.0), (13.0,7.0), (12.0,8.0), (12.0,1.0),
(12.0,12.0))
```

In [402]:
```
val predictionAndLabels=predictions.select("predictedLabel", "nt_OFFD
ETAIL").rdd.map(x =>
  (x(0).asInstanceOf[String].toDouble, x(1).asInstanceOf[Int].toDoubl
e))
```

```
predictionAndLabels: org.apache.spark.rdd.RDD[(Double, Double)] = Map
PartitionsRDD[1041] at map at <console>:126
```

In [403]:
```
val metrics = new MulticlassMetrics(predictionAndLabels)
```

```
metrics: org.apache.spark.mllib.evaluation.MulticlassMetrics = org.ap
ache.spark.mllib.evaluation.MulticlassMetrics@20d7c327
```

In [404]:
```
// Confusion matrix
println("Confusion matrix:")
println(metrics.confusionMatrix)
```

```
0.0  0.0  0.0  0.0  0.0  0.0  24.0   8.0    0.0  0.0  0.0  4757.0   2
084.0   0.0
0.0  0.0  0.0  0.0  0.0  0.0  6.0    4.0    0.0  0.0  0.0  920.0    4
18.0    0.0
0.0  0.0  0.0  0.0  0.0  0.0  43.0   1.0    0.0  0.0  0.0  6903.0   2
500.0   0.0
0.0  0.0  0.0  0.0  0.0  0.0  127.0  57.0   0.0  0.0  0.0  19069.0  7
995.0   0.0
0.0  0.0  0.0  0.0  0.0  0.0  176.0  83.0   0.0  0.0  0.0  30962.0  1
3813.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  36.0   31.0   0.0  0.0  0.0  7616.0   3
265.0   0.0
0.0  0.0  0.0  0.0  0.0  0.0  283.0  171.0  0.0  0.0  0.0  33884.0  1
4740.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  329.0  266.0  0.0  0.0  0.0  31220.0  1
2677.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  86.0   55.0   0.0  0.0  0.0  11866.0  4
567.0   0.0
0.0  0.0  0.0  0.0  0.0  0.0  53.0   99.0   0.0  0.0  0.0  15756.0  7
012.0   0.0
0.0  0.0  0.0  0.0  0.0  0.0  70.0   45.0   0.0  0.0  0.0  15014.0  5
885.0   0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0    0.0    0.0  0.0  0.0  86874.0  4
823.0   0.0
0.0  0.0  0.0  0.0  0.0  0.0  164.0  76.0   0.0  0.0  0.0  42640.0  3
5439.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  22.0   11.0   0.0  0.0  0.0  1759.0   9
44.0    0.0
```

In [405]:
```
// Overall Statistics
val accuracy = metrics.accuracy
println("Summary Statistics")
println(s"Accuracy = $accuracy")
```

```
Accuracy = 0.2872432948041746
```

In [407]:
```
// Weighted stats
println(s"Weighted precision: ${metrics.weightedPrecision}")
```

```
Weighted precision: 0.16947748379263627
```

In [408]:
```
println(s"Weighted recall: ${metrics.weightedRecall}")
```

```
Weighted recall: 0.28724329480417465
```

In [409]:
```
println(s"Weighted F1 score: ${metrics.weightedFMeasure}")
```

```
Weighted F1 score: 0.16214025130915416
```

In [410]:
```
println(s"Weighted false positive rate: ${metrics.weightedFalsePositiveRate}")
```

```
Weighted false positive rate: 0.1846857921321077
```

without pipelining

In [411]:
```
val featureCols=Array("ADMITYR","RELEASEYR","SEX","SENTLGTH",
                "RACE","AGEADMIT","TIMESRVD","RELTYPE","STATE",
                "ADMTYPE","OFFGENERAL","OFFDETAIL","nt_ADMTYPE")
val assembler = new VectorAssembler().setInputCols(featureCols).setOutputCol("features")
val df2 = assembler.transform(orange_red_df)
val labelIndexer = new StringIndexer().setInputCol("nt_OFFDETAIL").setOutputCol("label")
```

```
labelIndexer: org.apache.spark.ml.feature.StringIndexer = strIdx_0a1d7c7b5350
```

In [412]:
```
val df3 = labelIndexer.fit(df2).transform(df2)
val splitSeed = 456
val Array(trainingData, testDataCE) = df3.randomSplit(Array(0.6, 0.4), splitSeed)
val classifier = new RandomForestClassifier().setImpurity("gini").setFeatureSubsetStrategy("auto").setSeed(5043).
setLabelCol("label").setFeaturesCol("features")
val model = classifier.fit(trainingData)
```

```
model: org.apache.spark.ml.classification.RandomForestClassificationModel = RandomForestClassificationModel (uid=rfc_56ec5159d1d4) with 20 trees
```

In [413]:
```
val importanceVector =model.featureImportances
```

```
importanceVector: org.apache.spark.ml.linalg.Vector = (13,[0,1,2,3,4,
5,6,7,8,9,10,11,12],[0.0075134834931116345,0.015191431117774915,0.032
226726753848356,0.0026078609103597937,0.042406527129126716,0.01817656
9151839024,3.085412480219127E-4,0.0018357313008142203,0.0448102749762
7417,0.002523615625953274,0.46350015256569926,0.3636543721913644,0.00
5244713535812347])
```

FEATURE IMPORTANCES

In [415]:
```
importanceVector.toArray.zipWithIndex.
map(_.swap).
sortBy(-_._2).map{case (0,x)=> (x,"0:ADMITYR")
                                            case (1,x) => (x,"1:RELEASEY
R")
                                            case (2,x)=> (x,"2:SEX")
                                            case (3,x) => (x,"3:SENTLGT
H")
                                            case (4,x)=> (x,"4:RACE")
                                            case (5,x) => (x,"5:AGEADMI
T")
                                            case (6,x)=> (x,"6:TIMESRVD"
)
                                            case (7,x) => (x,"7:RELEASEY
R")
                                            case (8,x)=> (x,"8:RELTYPE")
                                            case (9,x) => (x,"9:STATE")
                                            case (10,x)=> (x,"10:ADMTYP
E")
                                            case (11,x) => (x,"11:OFFGEN
ERAL")
                                            case (12,x)=> (x,"12:nt_ADMT
YPE")}.
foreach(x => println(x._1 + " -> " + x._2))
```

```
0.46350015256569926 -> 10:ADMTYPE
0.3636543721913644 -> 11:OFFGENERAL
0.04481027497627417 -> 8:RELTYPE
0.042406527129126716 -> 4:RACE
0.032226726753848356 -> 2:SEX
0.018176569151839024 -> 5:AGEADMIT
0.015191431117774915 -> 1:RELEASEYR
0.0075134834931116345 -> 0:ADMITYR
0.005244713535812347 -> 12:nt_ADMTYPE
0.0026078609103597937 -> 3:SENTLGTH
0.002523615625953274 -> 9:STATE
0.0018357313008142203 -> 7:RELEASEYR
3.085412480219127E-4 -> 6:TIMESRVD
```

Creating Association Rules

In [ ]:
```
ADMITYR|     RELEASEYR|  SEX|  SENTLGTH|  RACE|  AGEADMIT|  TIMESRVD|
  RELTYPE|   STATE|  ADMTYPE|   OFFDETAIL|  nt_ADMTYPE|   nt_OFFDETAI
L|  TimeToReci|
```

In [363]:
```
orange_red_df.show()
```

```
+-------+---------+---+--------+----+--------+--------+-------+-----+
-------+----------+---------+------------+----------+
|ADMITYR|RELEASEYR|SEX|SENTLGTH|RACE|AGEADMIT|TIMESRVD|RELTYPE|STATE|
ADMTYPE|OFFGENERAL|OFFDETAIL|nt_OFFDETAIL|nt_ADMTYPE|
+-------+---------+---+--------+----+--------+--------+-------+-----+
-------+----------+---------+------------+----------+
|   2003|     2004|  1|       1|   2|       2|       0|      2|   37|
      1|         3|       12|          13|         1|
|   2004|     2006|  1|       3|   2|       2|       1|      1|   18|
      1|         4|       13|           5|         1|
|   1994|     2008|  1|       5|   1|       1|       4|      1|   48|
      2|         2|        7|           4|         2|
|   2001|     2001|  1|       0|   2|       2|       0|      2|   37|
      1|         3|       12|          13|         1|
|   1995|     1996|  1|       3|   2|       1|       1|      1|   36|
      1|         2|        7|           4|         2|
|   1998|     2000|  1|       2|   2|       2|       1|      2|   12|
      1|         1|        3|          13|         1|
|   1995|     1997|  1|       3|   2|       1|       2|      1|   17|
      1|         3|       12|           6|         1|
|   2004|     2005|  1|       2|   1|       4|       0|      1|    6|
      2|         1|        4|          13|         2|
|   1995|     1996|  1|       1|   2|       3|       0|      1|   27|
      2|         3|       12|           3|         1|
|   2000|     2002|  1|       2|   1|       4|       1|      1|   17|
      1|         2|        8|           4|         1|
|   2000|     2001|  1|       2|   3|       2|       0|      1|    6|
      2|         1|        4|          13|         1|
|   2000|     2004|  1|       3|   2|       1|       2|      2|   47|
      2|         3|       12|           5|         1|
|   2005|     2005|  1|       0|   4|       2|       0|      2|   44|
      1|         4|       13|          10|         1|
|   2010|     2010|  1|       0|   4|       3|       0|      2|   44|
      1|         2|       10|          13|         1|
|   1998|     2001|  1|       3|   1|       2|       2|      2|   47|
      3|         2|        7|          13|         1|
|   2002|     2003|  1|       0|   2|       1|       0|      2|   37|
      1|         2|       11|           1|         1|
|   2007|     2009|  1|       3|   2|       1|       1|      2|   12|
      1|         2|        9|          11|         1|
|   2003|     2006|  1|       2|   1|       2|       2|      1|   12|
      1|         1|        4|           7|         1|
|   2002|     2002|  1|       2|   2|       3|       0|      2|   45|
      1|         3|       12|           5|         1|
|   2004|     2006|  1|       3|   2|       3|       1|      1|   45|
      1|         1|        5|          12|         1|
+-------+---------+---+--------+----+--------+--------+-------+-----+
-------+----------+---------+------------+----------+
only showing top 20 rows
```

In [339]:
```
val asso_rulesss= orange_red_df
```

```
asso_rulesss: org.apache.spark.sql.DataFrame = [ADMITYR: int, RELEASE
YR: int ... 12 more fields]
```

In [340]:
```
import org.apache.spark.sql.functions._
import org.apache.spark.sql.DataFrame
```

```
import org.apache.spark.sql.DataFrame
```

In [342]:
```
val asso_rules_colnames = asso_rulesss.columns.foldLeft(asso_rulesss)
 {
  (acc: DataFrame, colName: String) =>

    acc.withColumn(colName, concat(lit(colName + " "), col(colName)))
}
```

```
asso_rules_colnames: org.apache.spark.sql.DataFrame = [ADMITYR: strin
g, RELEASEYR: string ... 12 more fields]
```

In [ ]:
```
asso_rules_colnames.show()
```

converting to usable rdd form

In [344]:
```
val asso_rules_colnames_rdd = asso_rules_colnames.map { _.toSeq.map {
_.toString}.toArray }.rdd
```

```
asso_rules_colnames_rdd: org.apache.spark.rdd.RDD[Array[String]] = Ma
pPartitionsRDD[879] at rdd at <console>:99
```

In [345]:
```
asso_rules_colnames_rdd.take(5)
```

```
res294: Array[Array[String]] = Array(Array(ADMITYR 2003, RELEASEYR 20
04, SEX 1, SENTLGTH 1, RACE 2, AGEADMIT 2, TIMESRVD 0, RELTYPE 2, STA
TE 37, ADMTYPE 1, OFFGENERAL 3, OFFDETAIL 12, nt_OFFDETAIL 13, nt_ADM
TYPE 1), Array(ADMITYR 2004, RELEASEYR 2006, SEX 1, SENTLGTH 3, RACE
2, AGEADMIT 2, TIMESRVD 1, RELTYPE 1, STATE 18, ADMTYPE 1, OFFGENERAL
4, OFFDETAIL 13, nt_OFFDETAIL 5, nt_ADMTYPE 1), Array(ADMITYR 1994, R
ELEASEYR 2008, SEX 1, SENTLGTH 5, RACE 1, AGEADMIT 1, TIMESRVD 4, REL
TYPE 1, STATE 48, ADMTYPE 2, OFFGENERAL 2, OFFDETAIL 7, nt_OFFDETAIL
4, nt_ADMTYPE 2), Array(ADMITYR 2001, RELEASEYR 2001, SEX 1, SENTLGTH
0, RACE 2, AGEADMIT 2, TIMESRVD 0, RELTYPE 2, STATE 37, ADMTYPE 1, OF
FGENERAL 3, OFFDETAIL 12, nt_OFFDETAIL 13, nt_ADMTYPE 1), Array(ADMIT
YR 1995, RELEASEYR 1996, SEX 1, SENT...
```

#

Generating Association rules

In [347]:
```
import org.apache.spark.mllib.fpm.FPGrowth
import org.apache.spark.rdd.RDD
```

```
import org.apache.spark.rdd.RDD
```

In [348]:
```
val fpg = new FPGrowth().setMinSupport(0.02).setNumPartitions(10)
val model = fpg.run(asso_rules_colnames_rdd)
```

```
model: org.apache.spark.mllib.fpm.FPGrowthModel[String] = org.apache.
spark.mllib.fpm.FPGrowthModel@2b12f0f0
```

In [349]:
```
val cols_fi = Seq("items", "freq")
val freq_items = model.freqItemsets.map{i=>(i.items.mkString("[", ","
, "]"),i.freq)} //.toDF(cols_fi: _*)
```

```
freq_items: org.apache.spark.rdd.RDD[(String, Long)] = MapPartitionsR
DD[888] at map at <console>:107
```

In [350]:
```
val minConfidence = 0.02

//val cols_ar = Seq("ante", "conseq","confi")
val asso_ruless =  model.generateAssociationRules(minConfidence).map{
 rule =>

    (rule.antecedent.mkString("[", ",", "]")
     ,rule.consequent .mkString("[", ",", "]")
      ,rule.confidence)
} //.toDF(cols_ar: _*)
```

```
asso_ruless: org.apache.spark.rdd.RDD[(String, String, Double)] = Map
PartitionsRDD[896] at map at <console>:111
```

In [351]:
```
asso_ruless.map(rule=>rule._1).take(2)
```

```
res298: Array[String] = Array([SENTLGTH 3,ADMTYPE 1,nt_ADMTYPE 1], [S
ENTLGTH 3,ADMTYPE 1,nt_ADMTYPE 1])
```

In [353]:
```
//freq_items.createOrReplaceTempView("freq_items_view")
//asso_rules.createOrReplaceTempView("asso_ruless_view")
```

In [354]:
```
asso_ruless.take(5)
```

```
res302: Array[(String, String, Double)] = Array(([OFFDETAIL 12,OFFGEN
ERAL 3,STATE 6,nt_ADMTYPE 2,ADMTYPE 2],[RELTYPE 1],0.993257499311989
7), ([OFFGENERAL 3,AGEADMIT 2,TIMESRVD 0,nt_ADMTYPE 1],[OFFDETAIL 1
2],1.0), ([OFFGENERAL 3,AGEADMIT 2,TIMESRVD 0,nt_ADMTYPE 1],[RACE 2],
0.5073417950632367), ([OFFGENERAL 3,AGEADMIT 2,TIMESRVD 0,nt_ADMTYPE
1],[RELTYPE 1],0.5426593361840766), ([OFFGENERAL 3,AGEADMIT 2,TIMESRV
D 0,nt_ADMTYPE 1],[ADMTYPE 1],0.6270503857553338))
```

In [355]:
```
freq_items.take(5)
```

```
res303: Array[(String, Long)] = Array(([ADMITYR 2007],79887), ([ADMIT
YR 2007,RELTYPE 2],32987), ([ADMITYR 2007,RELTYPE 2,ADMTYPE 1],2664
8), ([ADMITYR 2007,RELTYPE 2,ADMTYPE 1,nt_ADMTYPE 1],25723), ([ADMITY
R 2007,RELTYPE 2,ADMTYPE 1,nt_ADMTYPE 1,SEX 1],23134))
```

In [451]:
```
asso_ruless
```

```
res381: org.apache.spark.rdd.RDD[(String, String, Double)] = MapParti
tionsRDD[896] at map at <console>:111
```

In [358]:
```
asso_ruless.filter{f=>
    if(  "\\[nt_ADMTYPE\\s+\\d+\\]".r.findFirstIn(f._2).getOrElse("no
M") !="noM" ||
   "\\[nt_OFFDETAIL\\s+\\d+\\]".r.findFirstIn(f._2).getOrElse("noM")
!="noM"
    )
    {true}
    else
    {false}

}.take(5)
```

```
res306: Array[(String, String, Double)] = Array(([SENTLGTH 0,AGEADMIT
1,OFFGENERAL 2,SEX 1],[nt_ADMTYPE 1],0.9632027371528243), ([TIMESRVD
2,SENTLGTH 3,RACE 2],[nt_ADMTYPE 1],0.7642801133964654), ([OFFDETAIL
11,OFFGENERAL 2,TIMESRVD 0,SEX 1],[nt_ADMTYPE 1],0.7750190884916143),
([AGEADMIT 3,RELTYPE 2,ADMTYPE 1,SEX 1],[nt_ADMTYPE 1],0.978296001810
0133), ([OFFGENERAL 2,RACE 2,RELTYPE 1,TIMESRVD 0,ADMTYPE 1],[nt_ADMT
YPE 1],0.729236272822382))
```

In [468]:
```
val asso_ruless_comb = asso_ruless.filter{f=>
     if( ( "\\[nt_ADMTYPE\\s+\\d+\\]".r.findFirstIn(f._2).getOrElse(
"noM") !="noM" ||
    "\\[nt_OFFDETAIL\\s+\\d+\\]".r.findFirstIn(f._2).getOrElse("noM")
 !="noM" ) &&
        "nt_ADMTYPE\\s+\\d+".r.findFirstIn(f._1).getOrElse("noM") ==
"noM"
    )
    {true}
    else
    {false}

}.map(l=>(l._2,(l._1,l._2,l._3))).leftOuterJoin(freq_items)
```

```
asso_ruless_comb: org.apache.spark.rdd.RDD[(String, ((String, String,
Double), Option[Long]))] = MapPartitionsRDD[1232] at leftOuterJoin at
<console>:130
```

In [467]:
```
"nt_ADMTYPE\\s+\\d".r.findFirstIn("[OFFDETAIL 12,OFFGENERAL 3,nt_ADMT
YPE 1]").getOrElse("noM") =="noM"
```

```
res388: Boolean = false
```

In [469]:
```
asso_ruless_comb.take(3)
```

```
res389: Array[(String, ((String, String, Double), Option[Long]))] = A
rray(([nt_OFFDETAIL 9],(([ADMTYPE 1],[nt_OFFDETAIL 9],0.0331028376090
6922),Some(41064))), ([nt_OFFDETAIL 9],(([RELTYPE 1],[nt_OFFDETAIL
9],0.044007975950193),Some(41064))), ([nt_OFFDETAIL 9],(([TIMESRVD 0,
SEX 1],[nt_OFFDETAIL 9],0.0424882965628723),Some(41064))))
```

In [471]:
```
val total_instances = asso_rules_colnames_rdd.count()
```

```
total_instances: Long = 1068613
```

Calculating Support of (y)

In [472]:
```
val asso_ruless_comb_supp=asso_ruless_comb.map{l=>(l._2._1._1,l._2._1
._2,l._2._1._3,l._2._2.get,l._2._2.get.toDouble/total_instances)}
```

```
asso_ruless_comb_supp: org.apache.spark.rdd.RDD[(String, String, Doub
le, Long, Double)] = MapPartitionsRDD[1233] at map at <console>:125
```

In [473]:
```
asso_ruless_comb_supp.take(10)
```

```
res390: Array[(String, String, Double, Long, Double)] = Array(([ADMTY
PE 1],[nt_OFFDETAIL 9],0.03310283760906922,41064,0.03842738203634056
5), ([RELTYPE 1],[nt_OFFDETAIL 9],0.044007975950193,41064,0.038427382
036340565), ([TIMESRVD 0,SEX 1],[nt_OFFDETAIL 9],0.0424882965628723,4
1064,0.038427382036340565), ([RELTYPE 1,SEX 1],[nt_OFFDETAIL 9],0.044
847067927005296,41064,0.038427382036340565), ([SEX 1],[nt_OFFDETAIL
9],0.039023464227052995,41064,0.038427382036340565), ([TIMESRVD 0],[n
t_OFFDETAIL 9],0.041648822590450746,41064,0.038427382036340565), ([AD
MTYPE 1,SEX 1],[nt_OFFDETAIL 9],0.03353597953252689,41064,0.038427382
036340565), ([SEX 1],[nt_OFFDETAIL 3],0.02361954533365006,23444,0.021
93871869423262), ([ADMTYPE 1],[nt_OFFDETAIL 10],0.057796578181472456,
57073,0.05340848370738518), ([RELTYPE 1]...
```

Calculating Lift and conviction

In [474]:
```
val asso_ruless_comb_supp_lift_convi = asso_ruless_comb_supp.map{l=>
    (l._1,l._2,l._3,l._4,l._5,l._3/l._5,((1-l._5)/(1-l._3))
    )
}
```

```
asso_ruless_comb_supp_lift_convi: org.apache.spark.rdd.RDD[(String, S
tring, Double, Long, Double, Double, Double)] = MapPartitionsRDD[123
4] at map at <console>:127
```

Calculating freq of (x)

In [475]:
```
val asso_ruless_comb_supp_lift_convi_suppx= {asso_ruless_comb_supp_li
ft_convi.
                                             map{l => (l._1,(l._1,l._2,l._
3,l._4,l._5,l._6,l._7))}.
                                             leftOuterJoin(freq_items)}.
map{l=>(l._2._1._1,l._2._1._2,l._2._1._3,l._2._2.get,l._2._2.get.toDo
uble/total_instances,l._2._1._4,l._2._1._5,l._2._1._6,l._2._1._7)}
```

```
asso_ruless_comb_supp_lift_convi_suppx: org.apache.spark.rdd.RDD[(Str
ing, String, Double, Long, Double, Long, Double, Double, Double)] = M
apPartitionsRDD[1239] at map at <console>:132
```

Calculation of Chi square

using formula: (as mentioned in http://www.cs.bc.edu/~alvarez/ChiSquare/chi2tr.pdf (http://www.cs.bc.edu/~alvarez/ChiSquare/chi2tr.pdf))

$$\chi^2 = n\,(lift - 1)^2\,\frac{supp\;conf}{(conf - supp)\,(lift - conf)}$$

where $supp = P(X \bigcap Y) = \frac{conf \times P(X)}{n}$

In [476]:
```
val asso_ruless_comb_supp_lift_convi_suppx_chisqr = asso_ruless_comb_
supp_lift_convi_suppx.map{l=>
    (l._1,l._2,l._3,l._4,l._5,l._6,l._7,l._8,l._9,

    (total_instances*scala.math.pow((l._8 -1),2)*((l._4*l._3/total_in
stances)*l._3)/((l._3 - (l._4*l._3/total_instances))*(l._8 - l._3)))

    )

}
```

```
asso_ruless_comb_supp_lift_convi_suppx_chisqr: org.apache.spark.rdd.R
DD[(String, String, Double, Long, Double, Long, Double, Double, Doubl
e, Double)] = MapPartitionsRDD[1240] at map at <console>:131
```

In [477]:
```
asso_ruless_comb_supp_lift_convi_suppx_chisqr.take(10).foreach { rule
 =>
    println(rule._1 + "=>" + rule._2 + " " + rule._3 + " " + rule._4
+ " " + rule._5  + " " + rule._6 + " " + rule._7 + " " + rule._8 + "
 " + rule._9 + " " + rule._10 )

}
```

```
[STATE 6,ADMTYPE 2,AGEADMIT 2,RELTYPE 1,SEX 1]=>[nt_ADMTYPE 1] 0.3896
5989197780687 67949 0.06358616262388722 810605 0.7585580560970154 0.5
136849959549721 0.39558590485782935 53917.12056319117
[STATE 6,ADMTYPE 2,AGEADMIT 2,RELTYPE 1,SEX 1]=>[nt_ADMTYPE 2] 0.6103
401080221931 67949 0.06358616262388722 258008 0.2414419439029845 2.52
7895932893243 1.9467183349222383 53917.12056319119
[RACE 1,OFFGENERAL 2,SENTLGTH 2,RELTYPE 1]=>[nt_ADMTYPE 1] 0.58793066
12492299 60053 0.056197145271487434 810605 0.7585580560970154 0.77506
34991266071 0.5859255256286242 10114.615944356607
[RACE 1,OFFGENERAL 2,SENTLGTH 2,RELTYPE 1]=>[nt_ADMTYPE 2] 0.41206933
87507702 60053 0.056197145271487434 258008 0.2414419439029845 1.70670
15452640104 1.2902168675558408 10114.615944356627
[STATE 12,AGEADMIT 2,ADMTYPE 1,SEX 1]=>[nt_ADMTYPE 1] 0.9954378126671
491 31783 0.029742292111363047 810605 0.7585580560970154 1.3122763704
98184 52.922408986680196 10036.034334807944
[OFFDETAIL 12,OFFGENERAL 3,AGEADMIT 1,RACE 2,SEX 1]=>[nt_ADMTYPE 1]
0.8132050058925866 35638 0.033349772087743645 810605 0.75855805609701
54 1.0720405634827008 1.292550397598703 601.1381272508005
[AGEADMIT 3,RACE 1,RELTYPE 2,TIMESRVD 0]=>[nt_ADMTYPE 1] 0.9642036673
488994 23941 0.02240380755240672 810605 0.7585580560970154 1.27110068
84693647 6.744879322031921 5654.8375371458205
[STATE 13,OFFGENERAL 1,SEX 1]=>[nt_ADMTYPE 1] 0.9661302478166423 2255
7 0.021108670772300168 810605 0.7585580560970154 1.273640481504661 7.
128541791386934 5421.0530313886775
[OFFGENERAL 3,RACE 2,RELTYPE 1,SEX 1]=>[nt_ADMTYPE 1] 0.6608486946299
411 61247 0.05731448148207068 810605 0.7585580560970154 0.87119066143
75501 0.7119003821599316 3386.7940777095623
[OFFDETAIL 7,OFFGENERAL 2,RACE 2,SEX 1]=>[nt_ADMTYPE 1] 0.77224927301
00764 59148 0.05535025308507383 810605 0.7585580560970154 1.018049003
3729335 1.060114920790864 64.0843243486262
```

In [ ]:
```
asso_ruless_comb_supp_lift_convi_suppx_chisqr.collect().foreach { rul
e =>
    println(rule._1 + "=>" + rule._2 + " " + rule._3 + " " + rule._4
+ " " + rule._5  + " " + rule._6 + " " + rule._7 + " " + rule._8 + "
 " + rule._9 + " " + rule._10 )

}
```

calculating phi sqrt(chi square/ N)

In [479]:
```
asso_ruless_comb_supp_lift_convi_suppx_chisqr.map{l=>
    (l._1,l._2,l._3,l._4,l._5,l._6,l._7,l._8,l._9,l._10,
     scala.math.sqrt(l._10/total_instances)
     )
}.take(10)
```

```
res393: Array[(String, String, Double, Long, Double, Long, Double, Do
uble, Double, Double, Double)] = Array(([STATE 6,ADMTYPE 2,AGEADMIT
2,RELTYPE 1,SEX 1],[nt_ADMTYPE 1],0.38965989197780687,67949,0.0635861
6262388722,810605,0.7585580560970154,0.5136849959549721,0.39558590485
782935,53917.12056319117,0.22462242873427107), ([STATE 6,ADMTYPE 2,AG
EADMIT 2,RELTYPE 1,SEX 1],[nt_ADMTYPE 2],0.6103401080221931,67949,0.0
6358616262388722,258008,0.2414419439029845,2.527895932893243,1.946718
3349222383,53917.12056319119,0.2246224287342711), ([RACE 1,OFFGENERAL
2,SENTLGTH 2,RELTYPE 1],[nt_ADMTYPE 1],0.5879306612492299,60053,0.056
197145271487434,810605,0.7585580560970154,0.7750634991266071,0.585925
5256286242,10114.615944356607,0.09728916409116731), ([RACE 1,OFFGENER
AL 2,SENTLGTH 2,RELTYPE 1],[nt_ADMTYPE...
```

In [480]:
```
val asso_ruless_comb_supp_lift_convi_suppx_chisqr_phi = asso_ruless_c
omb_supp_lift_convi_suppx_chisqr.map{l=>
    (l._1,l._2,l._3,l._4,l._5,l._6,l._7,l._8,l._9,l._10,
     scala.math.sqrt(l._10/total_instances)
     )
}
```

```
asso_ruless_comb_supp_lift_convi_suppx_chisqr_phi: org.apache.spark.r
dd.RDD[(String, String, Double, Long, Double, Long, Double, Double, D
ouble, Double, Double)] = MapPartitionsRDD[1242] at map at <console>:
133
```

Saving to CSV

In [481]:
```
val cols_ar = Seq("antecedent", "consequent","confidence","P(x)","Sup
port(x)","P(y)","Support(y)","lift","conviction","Chi-square","phi")
asso_ruless_comb_supp_lift_convi_suppx_chisqr_phi.toDF(cols_ar: _*).w
rite.format("com.databricks.spark.csv").option("header", "false").sav
e("hdfs://mas.local:8020/data/pris_data_OrangeRedAsso_withChi_Square_
phi.csv")
```