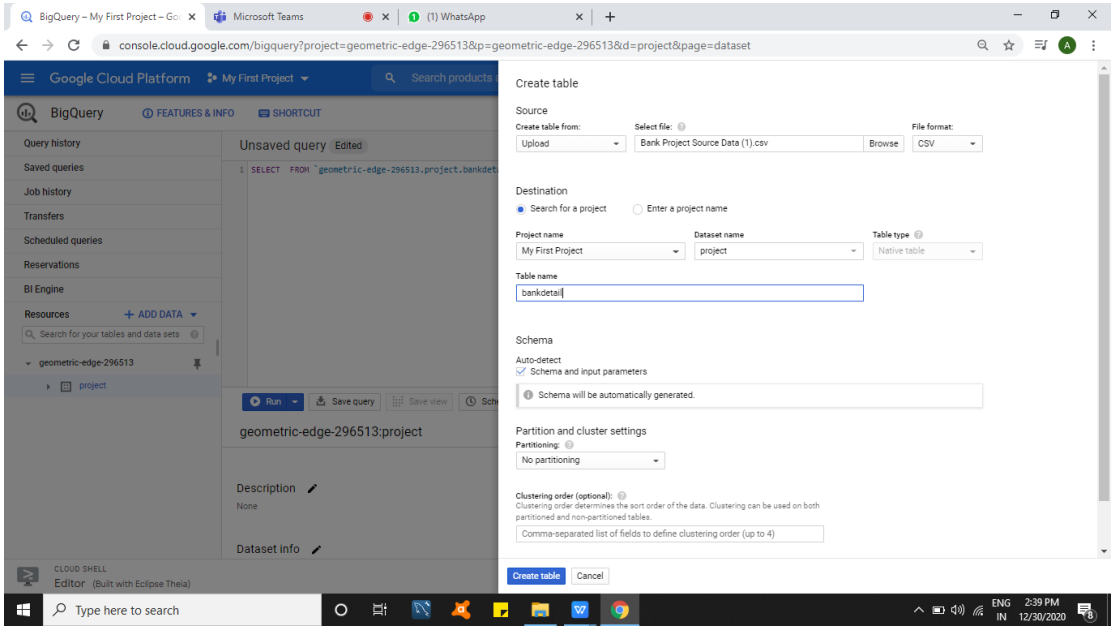
Project Analysis & Reporting System of Bank Account Details

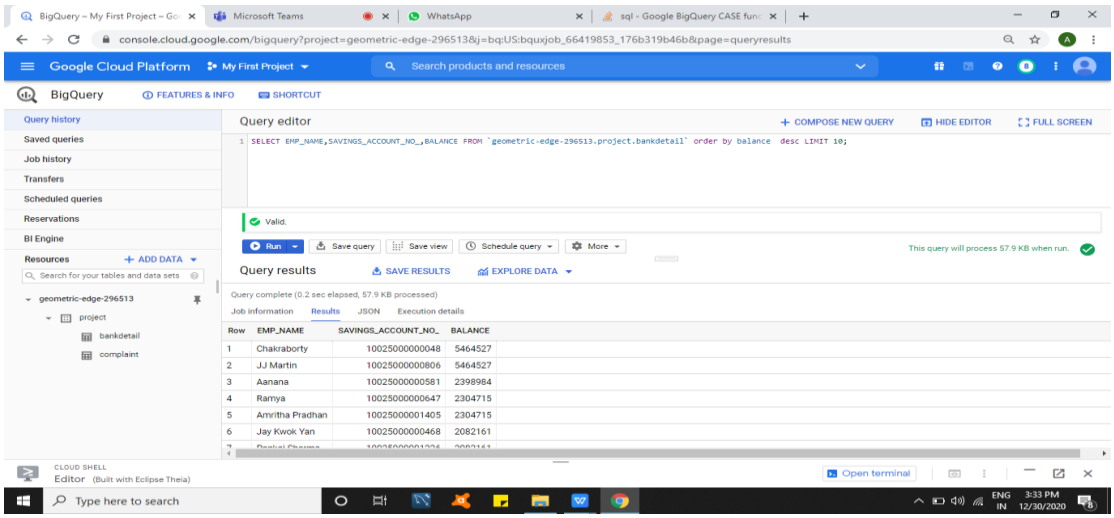
Big query:
1.Analyse the dataset and find the top 10 account and its details based on salary



Code:

```
SELECT EMP_NAME,SAVINGS_ACCOUNT_NO_, SAVINGS_ACCOUNT_NO_,BALANCE FROM
`geometric-edge-296513.project.bankdetail` order by balance    desc LIMIT 10;
```

Output:

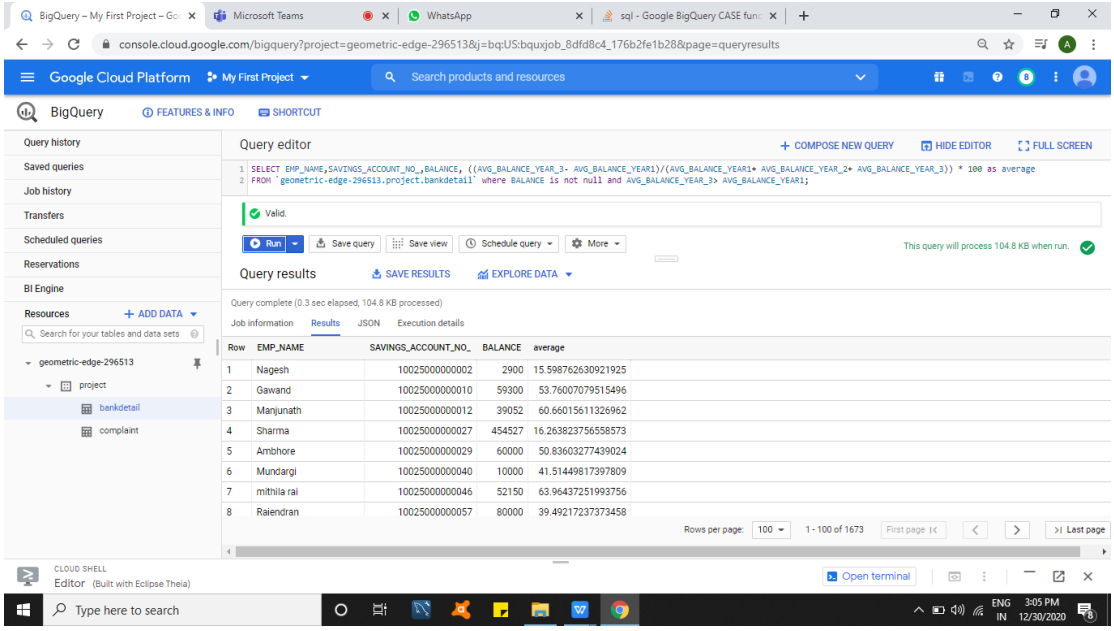


6. Analyse the dataset and find the Customer's % increase in the average bank balance for 2 years.

Code:

```
SELECT EMP_NAME,SAVINGS_ACCOUNT_NO_,BALANCE,
((AVG_BALANCE_YEAR_3- AVG_BALANCE_YEAR1)/(AVG_BALANCE_YEAR1+
AVG_BALANCE_YEAR_2+ AVG_BALANCE_YEAR_3)) * 100 as average
FROM `geometric-edge-296513.project.bankdetail` where BALANCE is not null and
AVG_BALANCE_YEAR_3> AVG_BALANCE_YEAR1;
```

Output:

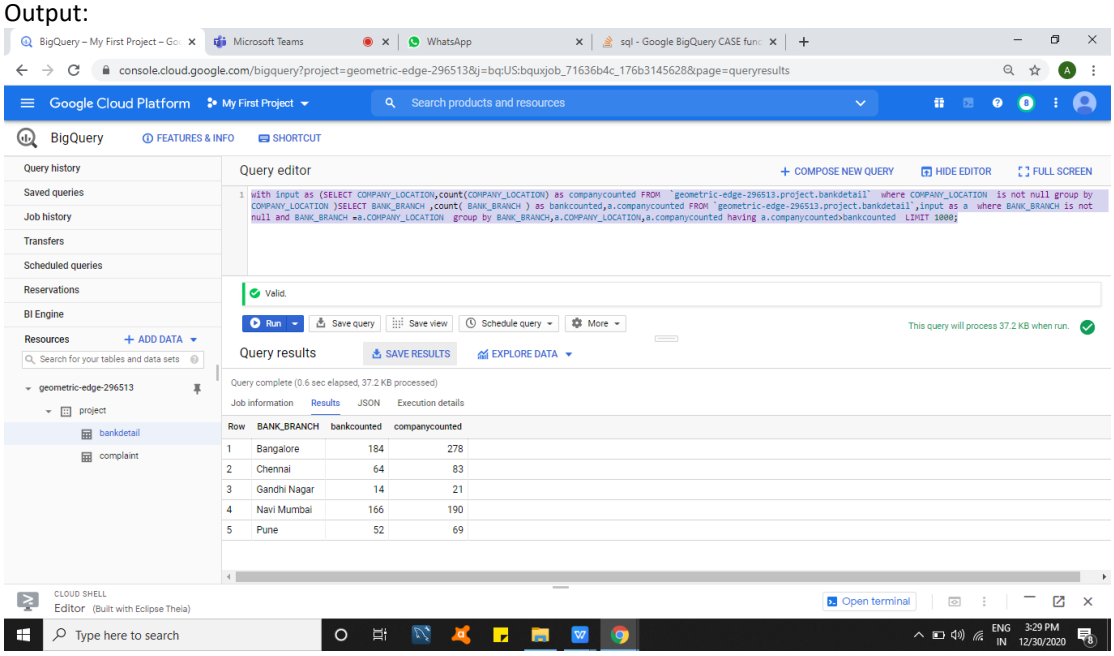


9.Analyse the dataset and find the locations where more banking portals should be opened(number of employees for a particular location and number of bank branches shall be same, if less they need more banking portals)

Code:

```
with input as (SELECT COMPANY_LOCATION,count(COMPANY_LOCATION) as companycounted FROM
`geometric-edge-296513.project.bankdetail`  where COMPANY_LOCATION  is not null group by
COMPANY_LOCATION )SELECT BANK_BRANCH ,count( BANK_BRANCH ) as
bankcounted,a.companycounted FROM `geometric-edge-296513.project.bankdetail`,input as a
where BANK_BRANCH is not null and BANK_BRANCH =a.COMPANY_LOCATION  group by
BANK_BRANCH,a.COMPANY_LOCATION,a.companycounted having a.companycounted>bankcounted
LIMIT 1000;
```

Output:

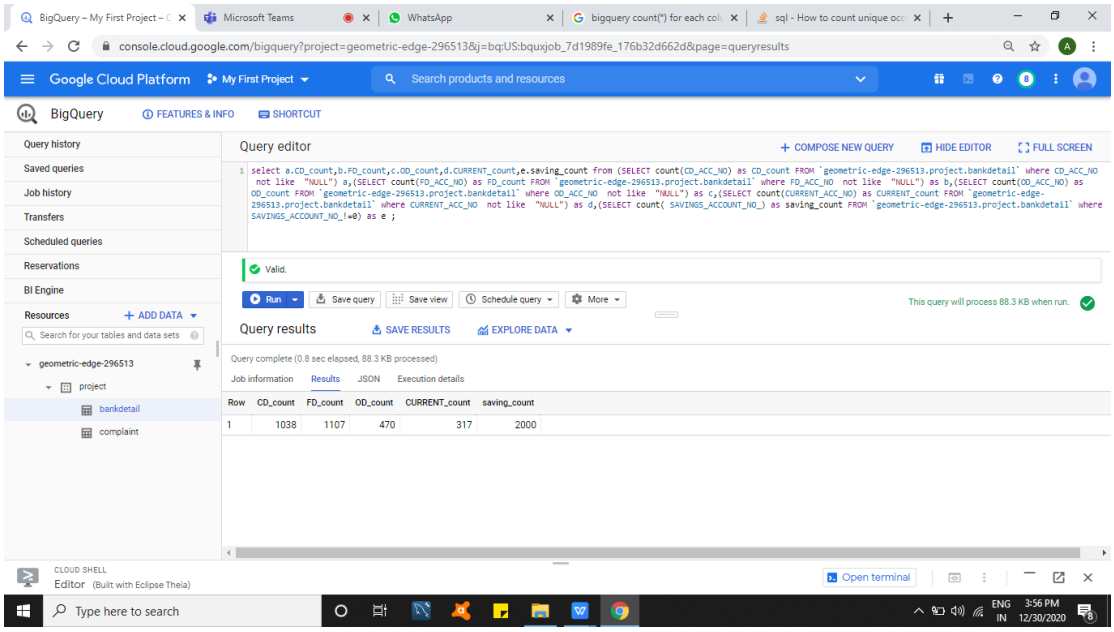


11. Analyse the dataset and count the number of customers belonging to a particular type of account.

Code:

```
select a.CD_count,b.FD_count,c.OD_count,d.CURRENT_count,e.saving_count from
(SELECT count(CD_ACC_NO) as CD_count FROM
`geometric-edge-296513.project.bankdetail` where CD_ACC_NO  not like  "NULL")
a,(SELECT count(FD_ACC_NO) as FD_count FROM
`geometric-edge-296513.project.bankdetail` where FD_ACC_NO  not like  "NULL") as
b,(SELECT count(OD_ACC_NO) as OD_count FROM
`geometric-edge-296513.project.bankdetail` where OD_ACC_NO  not like  "NULL") as
c,(SELECT count(CURRENT_ACC_NO) as CURRENT_count FROM
`geometric-edge-296513.project.bankdetail` where CURRENT_ACC_NO  not like  "NULL")
as d,(SELECT count( SAVINGS_ACCOUNT_NO_) as saving_count FROM
`geometric-edge-296513.project.bankdetail` where SAVINGS_ACCOUNT_NO_!=0) as e ;
```

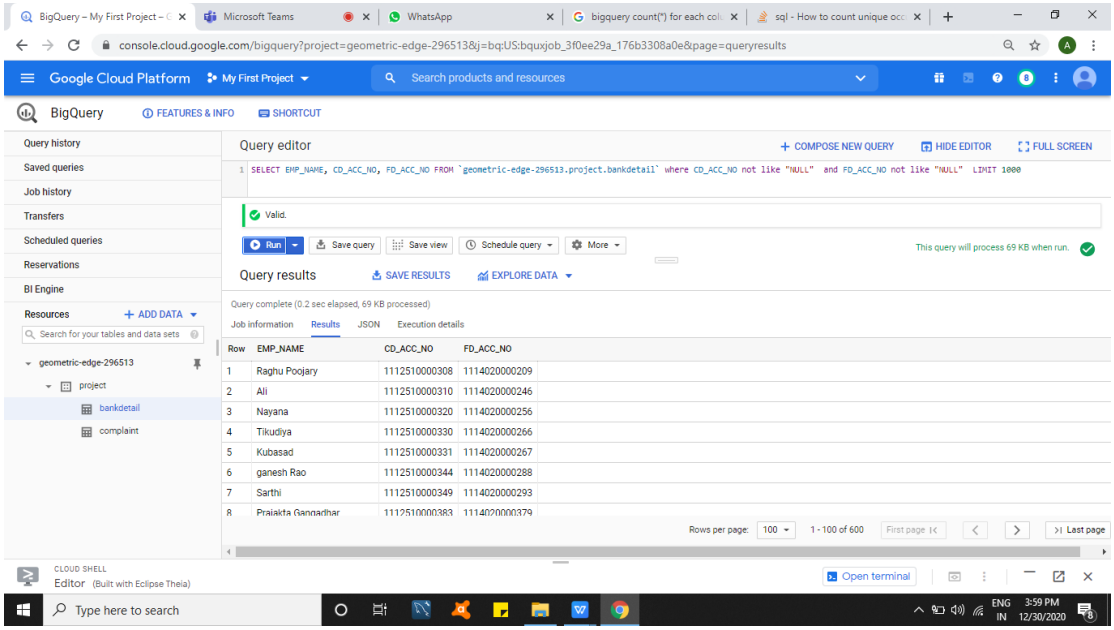
Output:



13. Analyse the dataset and find the employees having both CD ACC and FD ACC.
Code:

```
SELECT EMP_NAME, CD_ACC_NO, FD_ACC_NO FROM
`geometric-edge-296513.project.bankdetail` where CD_ACC_NO not like "NULL"   and
FD_ACC_NO not like "NULL"   LIMIT 1000
```

Output:



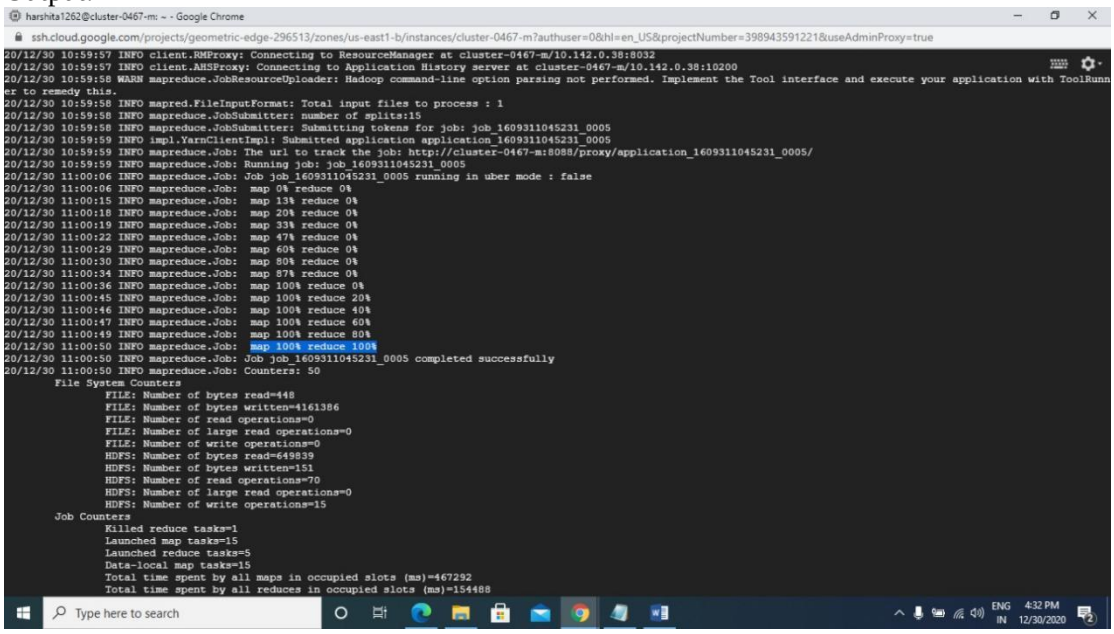
Dataproc:

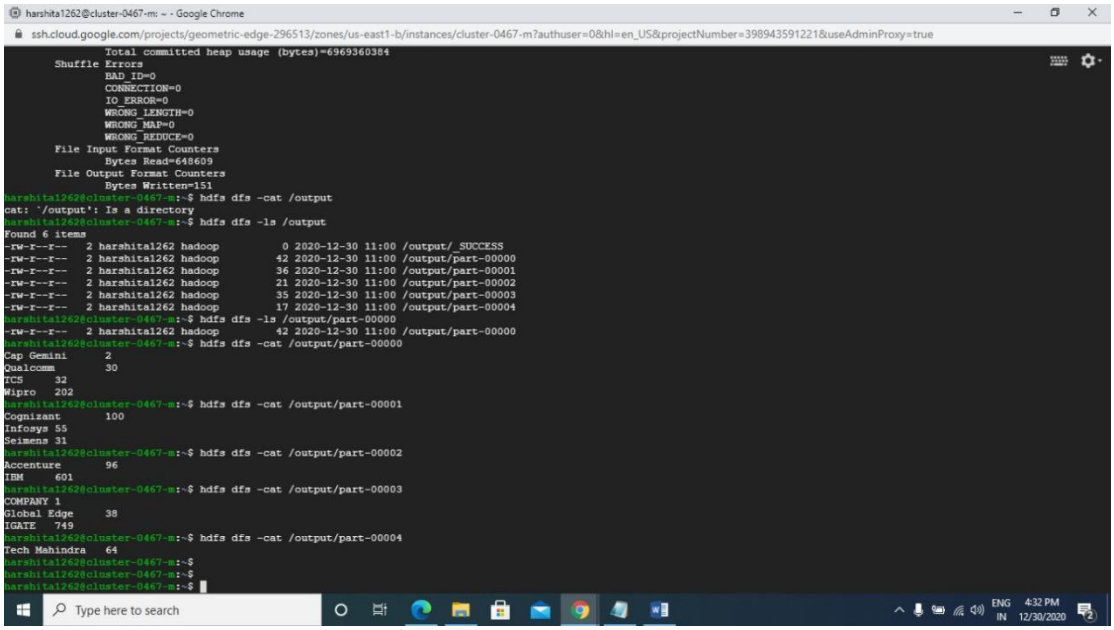
4. Analyse the company and find the count of its employees registering for the bank.
Code:

```
package mypack;

import java.io.IOException;
import java.util.Iterator;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;

public class location {
      public static class Map extends MapReduceBase implements     Mapper<LongWritable ,
Text, Text,     IntWritable>
          {

               public void map(LongWritable key, Text value,   OutputCollector<Text,
IntWritable> output,     Reporter reporter) throws IOException {
                    String line = value.toString();
                    String [] count=line.split(",");

                  output.collect(new Text(count[4]), new IntWritable(1));

               }
          }

public static class Reduce extends MapReduceBase implements Reducer< Text, IntWritable,
Text, IntWritable > {


                    public void reduce( Text key, Iterator <IntWritable> values,
                    OutputCollector<Text, IntWritable> output, Reporter reporter) throws
IOException {
                              int sum_result=0;
```

```
                    while (values.hasNext()) {
                        sum_result+=values.next().get();
                    }
                    output.collect(key, new IntWritable(sum_result));
                }
            }
        public static void main(String args[])throws Exception {
            JobConf conf = new JobConf(location.class);

            conf.setJobName("max_count");
            conf.setOutputKeyClass(Text.class);
            conf.setOutputValueClass(IntWritable.class);
            conf.setMapperClass(Map.class);
            conf.setCombinerClass(Reduce.class);
            conf.setReducerClass(Reduce.class);
            conf.setInputFormat(TextInputFormat.class);
            conf.setOutputFormat(TextOutputFormat.class);

            FileInputFormat.setInputPaths(conf, new Path(args[0]));
            FileOutputFormat.setOutputPath(conf, new Path(args[1]));

            JobClient.runJob(conf);
}
}
```

Output:



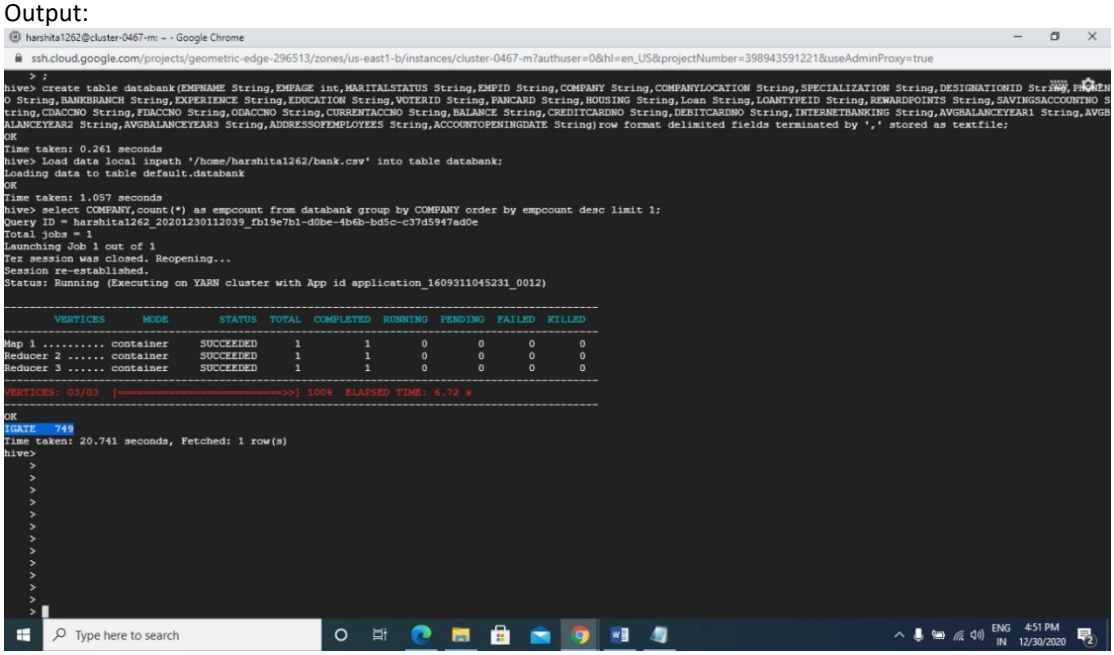


7. Analyse the dataset and find the company having most number of employees registering for the bank.

Code:

```
create table input2(EMPNAME String,EMPAGE String,MARITALSTATUS String,EMPID String,COMPANY String,COMPANYLOCATION String,SPECIALIZATION String,DESIGNATIONID String,PHONENO String,BANKBRANCH String,EXPERIENCE String,EDUCATION String,VOTERID String,PANCARD String,HOUSING String,Loan String,LOANTYPEID String,REWARDPOINTS String,SAVINGSACCOUNTNO String,CDACCNO String,FDACCNO String,ODACCNO String,CURRENTACCNO String,BALANCE String,CREDITCARDNO String,DEBITCARDNO String,INTERNETBANKING String,AVGBALANCEYEAR1 String,AVGBALANCEYEAR2 String,AVGBALANCEYEAR3 String,ADDRESSOFEMPLOYEES String,ACCOUNTOPENINGDATE String)row format delimited fields terminated by ',' stored as textfile;

load data local inpath '/home/ameer17_08/bank.csv' into table input2;

select company,count(*) as counted  from input2 group by company order by counted desc limit 1;
```

Output:

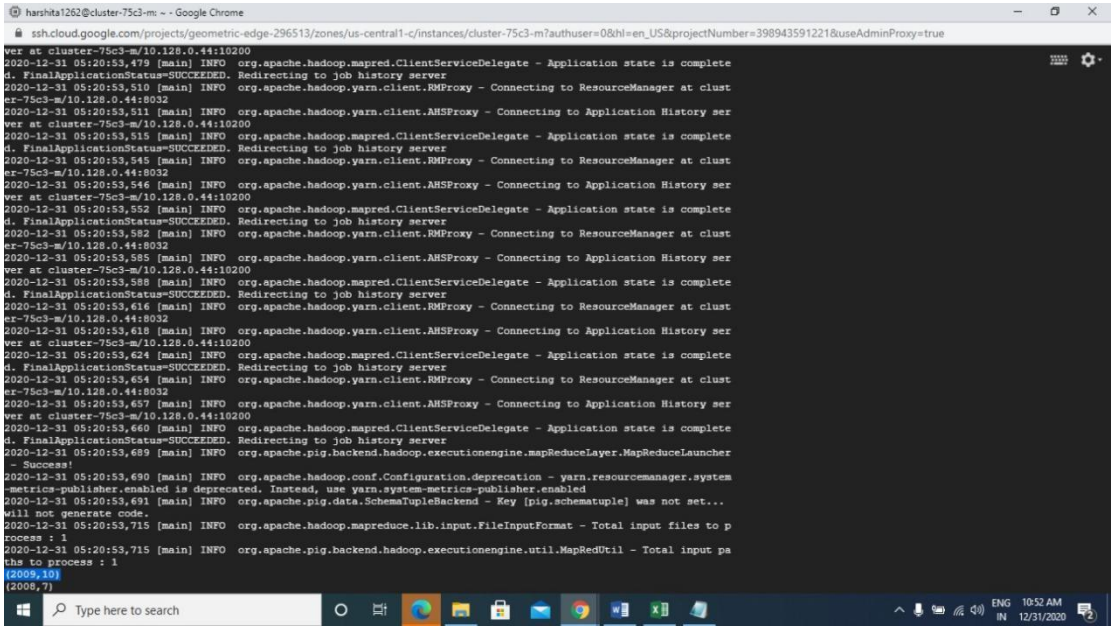


8. Analyse the dataset and find the year in which most number of loan were Sanctioned

Code:

```
file =load '/bankfinal1.csv' using PigStorage(',') as
(EMPNAME,EMPAGE,MARITALSTATUS,EMPID,COMPANY,COMPANYLOCATION,
SPECIALIZATION,DESIGNATIONID,PHONENO,BANKBRANCH,EXPERIENCE,EDU
CATION,VOTERID,PANCARD,HOUSING,Loan,LOANTYPEID,REWARDPOINTS,SAVI
NGSACCOUNTNO,CDACCNO,FDACCNO,ODACCNO,CURRENTACCNO,BALANCE,
CREDITCARDNO,DEBITCARDNO,INTERNETBANKING,AVGBALANCEYEAR1,AV
GBALANCEYEAR2,AVGBALANCEYEAR3,ADDRESSOFEMPLOYEES,ACCOUNTOP
ENINGDATE:chararray);

filter_data = FILTER file BY Loan == TRUE;

substring_data = FOREACH filter_data GENERATE Loan,ACCOUNTOPENINGDATE,
SUBSTRING (ACCOUNTOPENINGDATE, 6,10 ) as year;

group1 =group substring_data by   year;
grouping= foreach group1 generate group,COUNT(substring_data.Loan)as compliant;
final = LIMIT final_result 1;
store final into 'secondprogram.pig';
dump final;
```

Output:



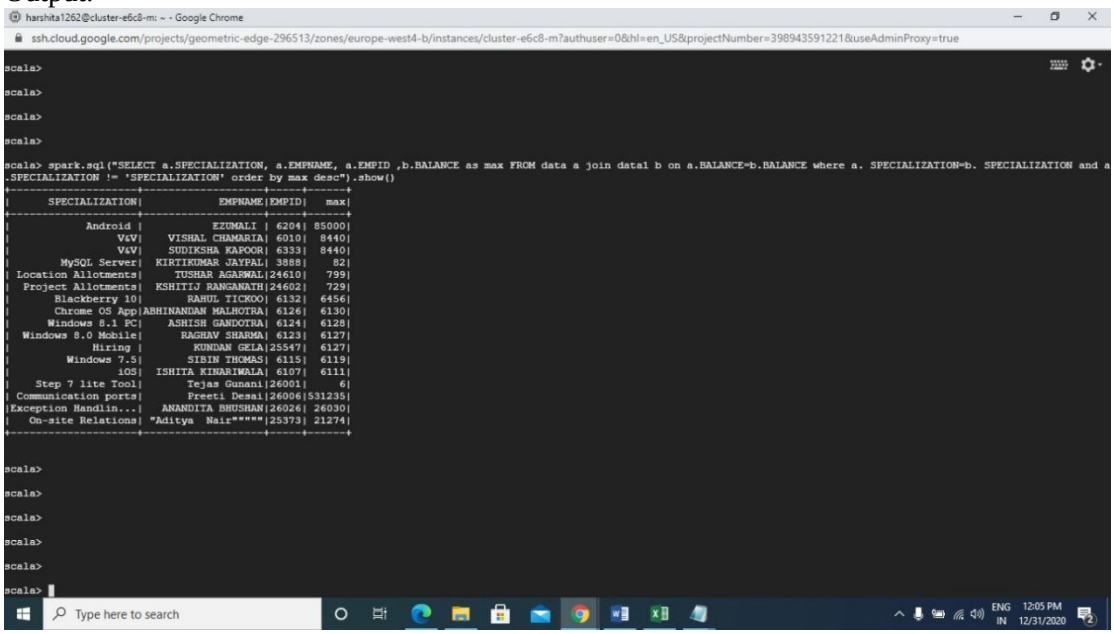
12.Analyse the dataset and find the maximum salary and the details of the employee having that salary for a particular specialization.

Code:

```
spark.sql("create table data(EMPNAME String,EMPAGE int,MARITALSTATUS String,EMPID String,COMPANY String,COMPANYLOCATION String,SPECIALIZATION String,DESIGNATIONID String,PHONENO String,BANKBRANCH String,EXPERIENCE String,EDUCATION String,VOTERID String,PANCARD String,HOUSING String,Loan String,LOANTYPEID String,REWARDPOINTS String,SAVINGSACCOUNTNO String,CDACCNO String,FDACCNO String,ODACCNO String,CURRENTACCNO String,BALANCE String,CREDITCARDNO String,DEBITCARDNO String,INTERNETBANKING String,AVGBALANCEYEAR1 String,AVGBALANCEYEAR2 String,AVGBALANCEYEAR3 String,ADDRESSOFEMPLOYEES String,ACCOUNTOPENINGDATE String) row format delimited fields terminated by ',' stored as textfile")

spark.sql("Load data local inpath '/bankfinal1.csv' into table data")

spark.sql("create table data1(SPECIALIZATION String, BALANCE String) row format delimited fields terminated by ',' stored as textfile")

spark.sql("INSERT INTO data1 SELECT SPECIALIZATION ,MAX(BALANCE) as max FROM data where group by SPECIALIZATION ")

spark.sql("SELECT a.SPECIALIZATION, a.EMPNAME, a.EMPID ,b.max as max FROM data a join data1 b on a.BALANCE=b.max where a. SPECIALIZATION=b.
```

SPECIALIZATION and a. SPECIALIZATION!=’SPECIALIZATION’ order by max desc").show()

Output:

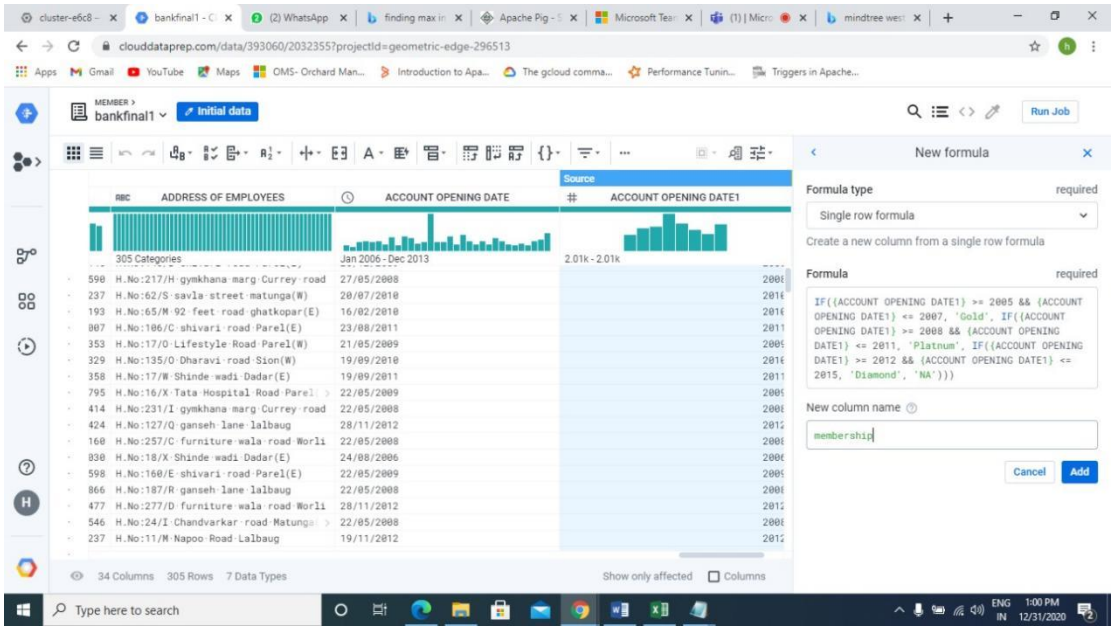


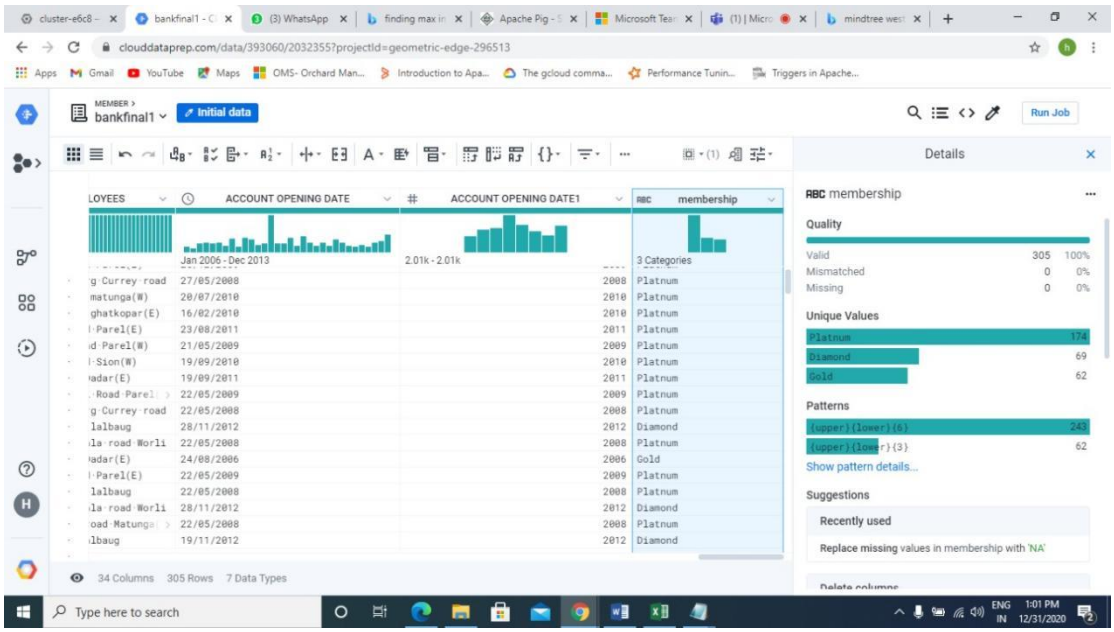
Dataprep:

5.Analyse the dataset and find the Customer's membership(gold, platinum, diamond) based on the account opening date(for year 2005-2007->gold, 2008-2011->platinum, 2012-2015->diamond).
Code:

IF({ACCOUNT OPENING DATE1} >= 2005 && {ACCOUNT OPENING DATE1} <= 2007, 'Gold', IF({ACCOUNT OPENING DATE1} >= 2008 && {ACCOUNT OPENING DATE1} <= 2011, 'Platnum', IF({ACCOUNT OPENING DATE1} >= 2012 && {ACCOUNT OPENING DATE1} <= 2015, 'Diamond', 'NA')))

Output:





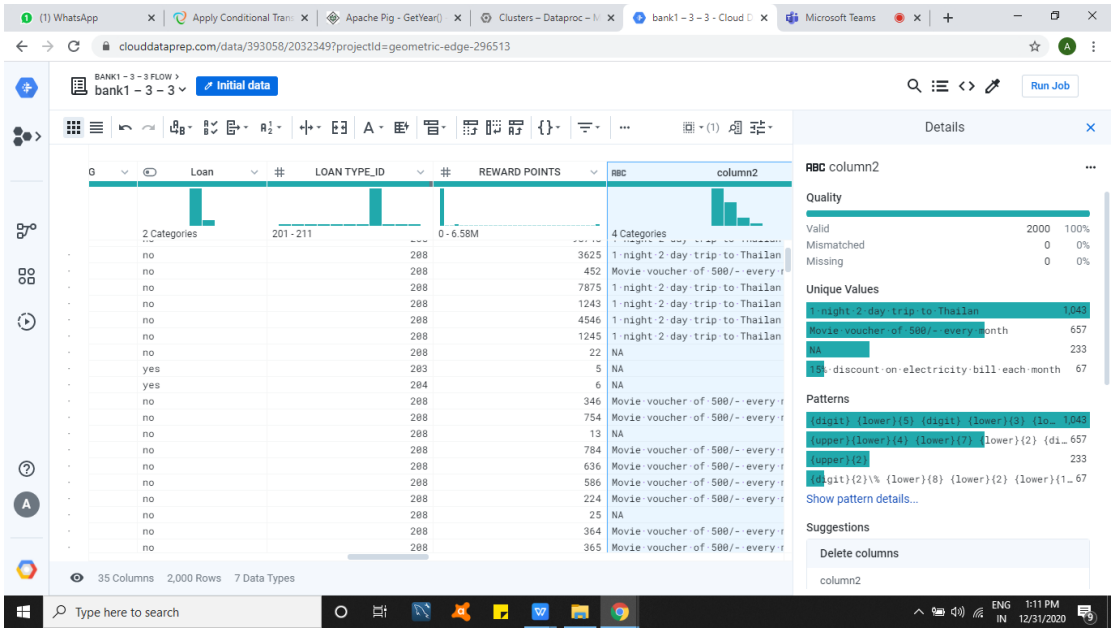
10. Analyse the dataset and provide offers to the customers(if reward points in the range of 100-200k then

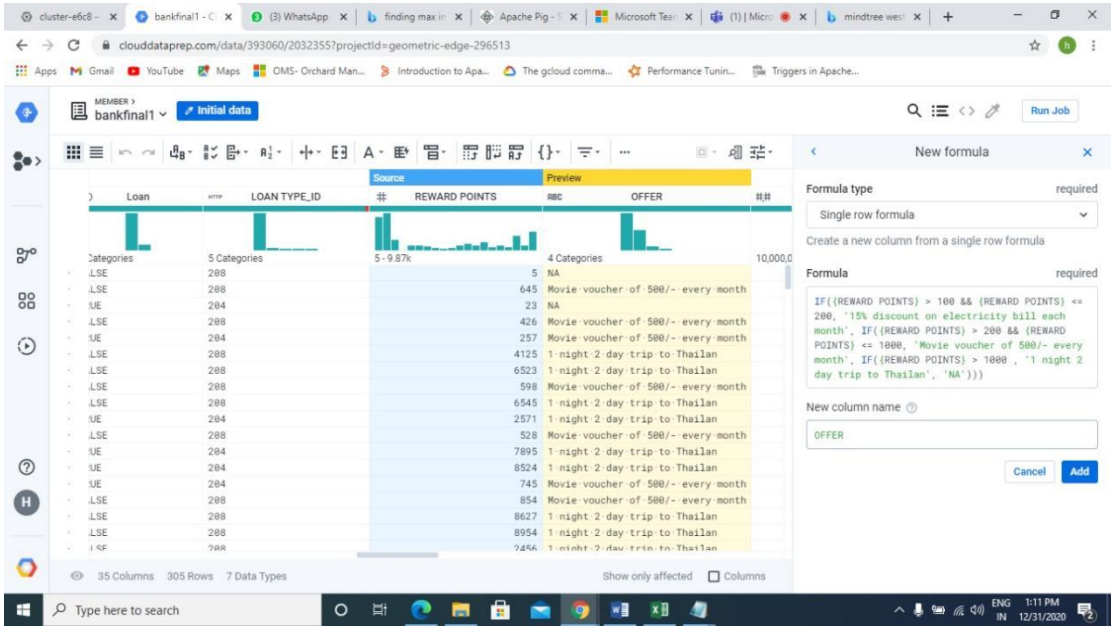
- 15% discount on electricity bill each month – range : 200k-1000k
- Movie voucher of 500/- every month – range : 1000k and above
- 1 night 2 day trip to Thailand

Code:

```
IF({REWARD POINTS} >= 100 && {REWARD POINTS} <= 200000, "15% discount on electricity bill each month", IF({REWARD POINTS} >= 200000 && {REWARD POINTS} <= 1000000, "Movie voucher of 500/- every month", IF({REWARD POINTS} >= 1000000 , "1 night 2 day trip to Thailan", "NA")))
```

Output:





Dataflow:

14. Analyse the dataset and find the most preferred account created by the customers

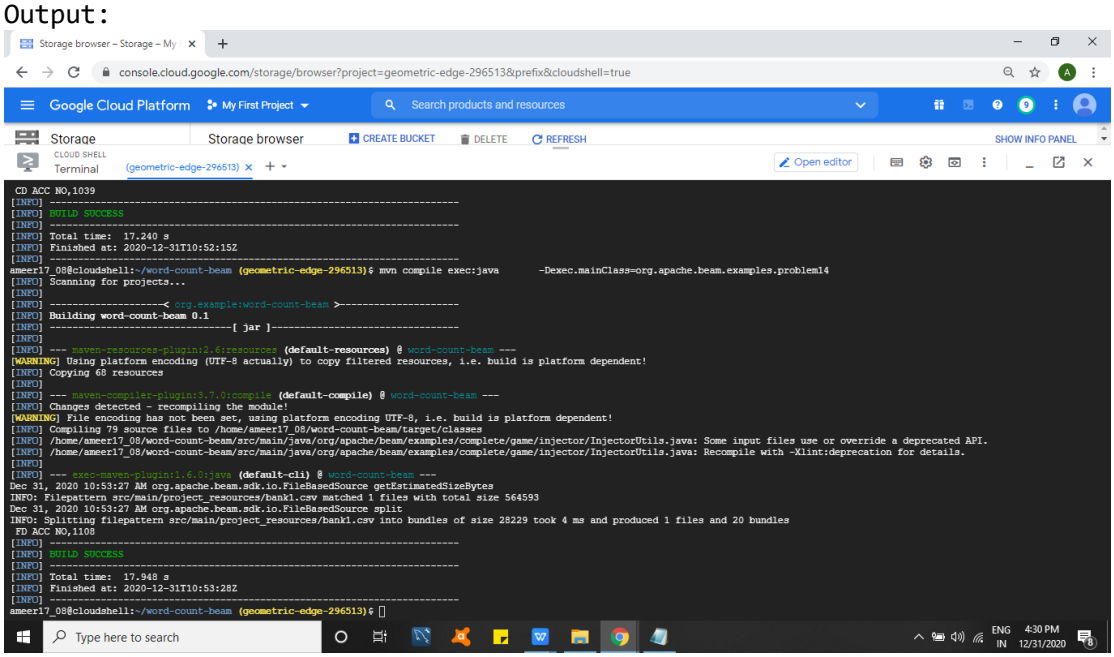
Code:

```
package org.apache.beam.examples;

import org.apache.beam.sdk.Pipeline;
import org.apache.beam.sdk.options.PipelineOptions;
```

```
import org.apache.beam.sdk.options.PipelineOptionsFactory;
import org.apache.beam.sdk.io.TextIO;
import org.apache.beam.sdk.transforms.*;
import org.apache.beam.sdk.values.*;


public class problem14 {
       private static final String CSV_HEADER =
           "Operator,Indoor_Outdoor_Travelling,Network Type,Rating,Call
 Drop Category,Latitude,Longitude,State Name";
   static int count=0;
       public static void main(String[] args) {
        PipelineOptions options = PipelineOptionsFactory.fromArgs(args)
.withValidation().create();
        Pipeline pipeline = Pipeline.create(options);

                 pipeline.apply("ReadAds", TextIO.read().from("src/main
/project_resources/bank1.csv"))
                .apply("FilterHeader", ParDo.of(new FilterHeaderFn(CSV_
HEADER)))
                .apply("MakeAgeKVFn", ParDo.of(new MakeAgeKVFn()))
                .apply(Sum.<String>integersPerKey())
                .apply("MAXKEY",Max.integersPerKey())
                .apply(ParDo.of(new ConvertToStringFn()))
                .apply("PrintToConsole", ParDo.of(new DoFn<String, Void
>() {
                    @ProcessElement
                    public void processElement(ProcessContext c) {
                        System.out.println(c.element());
                    }
                }));

        //value.apply("Writeresult to file", TextIO.write().to("gs://al
binbucket/amita/output"));

        pipeline.run().waitUntilFinish();
    }

    private static class FilterHeaderFn extends DoFn<String, String> {

        private final String header;

        public FilterHeaderFn(String header) {
            this.header = header;
        }

        @ProcessElement
        public void processElement(ProcessContext c) {
            String row = c.element();
```

```
            if (!row.isEmpty() && !row.equals(this.header)) {
                c.output(row);
            }
        }
    }

    private static class MakeAgeKVFn extends DoFn<String, KV<String, In
teger>> {

        @ProcessElement
        public void processElement(ProcessContext c) {
            String[] fields = c.element().split(",");

                String account="";
           if(!fields[19].equals("NULL")){
               account=" CD ACC NO";
            c.output(KV.of(account, 1));
           }
           if(!fields[20].equals("NULL")){
               account=" FD ACC NO";
            c.output(KV.of(account,1));
           }
           if(!fields[20].equals("NULL")){
               account=" OD ACC NO";
            c.output(KV.of(account,1));
           }
        }
    }

    private static class ConvertToStringFn extends DoFn<KV<String,Integ
er>, String> {

        @ProcessElement
        public void processElement(ProcessContext c) {

            if(count==0){
                c.output(c.element().getKey() + "," + c.element().getVa
lue());
                count++;
            }
        }
    }
}
```

Output:



2. Analyse which situation is more favourable to get the loan sanctioned based on marital status and housing

Code:

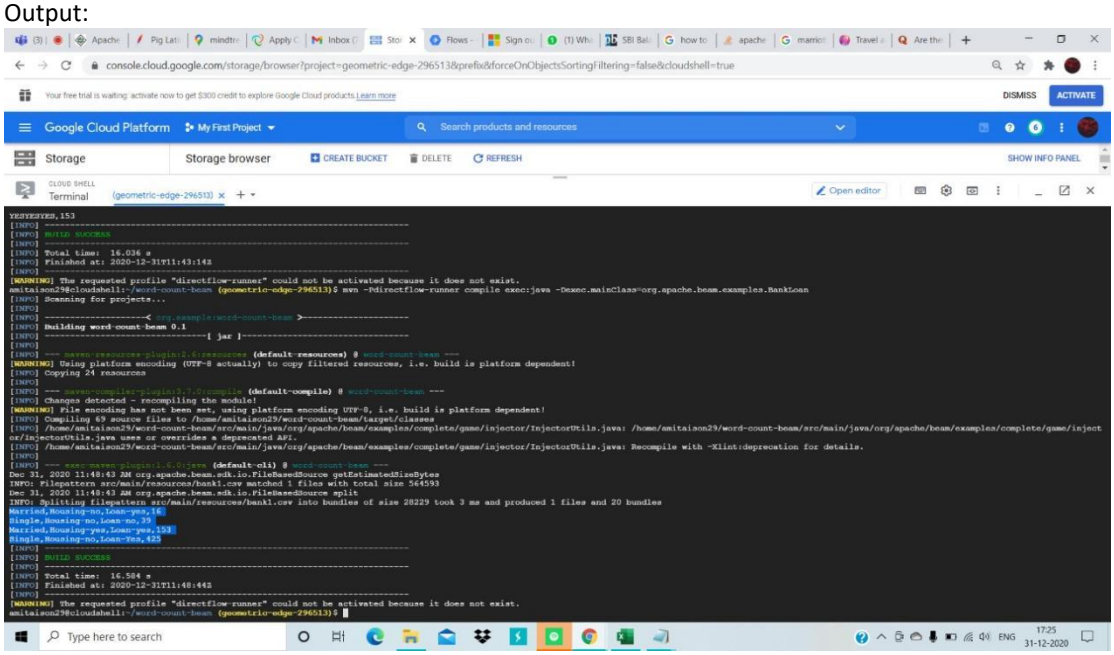
```
package org.apache.beam.examples;

import org.apache.beam.sdk.Pipeline;
import org.apache.beam.sdk.options.PipelineOptions;
import org.apache.beam.sdk.options.PipelineOptionsFactory;
import org.apache.beam.sdk.io.TextIO;
import org.apache.beam.sdk.transforms.*;
import org.apache.beam.sdk.values.*;

import java.util.Calendar;

public class BankLoan {

    public static void main(String[] args) {
        PipelineOptions options = PipelineOptionsFactory.fromArgs(args).withValidation().create();
        Pipeline pipeline = Pipeline.create(options);

        PCollection<String> value = pipeline.apply("ReadAds",
TextIO.read().from("src/main/resources/bank1.csv"))
                //.apply("FilterHeader", ParDo.of(new FilterHeaderFn(CSV_HEADER)))
                .apply("MakeAgeKVFn", ParDo.of(new MakeAgeKVFn()))
                .apply("Sum",Sum.integersPerKey())
```

```
                .apply(ParDo.of(new ConvertToStringFn()));

        value.apply("PrintToConsole", ParDo.of(new DoFn<String, Void>() {
                        @ProcessElement
                        public void processElement(ProcessContext c) {
                                System.out.println(c.element());
                        }
                }));

      // value.apply("Writeresult to file", TextIO.write().to("gs://albinbucket/amita/output"));

        pipeline.run().waitUntilFinish();
}

private static class FilterHeaderFn extends DoFn<String, String> {

        private final String header;

        public FilterHeaderFn(String header) {
                this.header = header;
        }

        @ProcessElement
        public void processElement(ProcessContext c) {
                String row = c.element();

                if (!row.isEmpty() && !row.equals(this.header)) {
                        c.output(row);
                }
        }
}

private static class MakeAgeKVFn extends DoFn<String, KV<String, Integer>> {

        @ProcessElement
        public void processElement(ProcessContext c) {
                String[] fields = c.element().split(",");

                String marriage = fields[2];
                String housing = fields[14];
                String val1="Married,Housing-yes,Loan-yes";
                String val2 = "Married,Housing-no,Loan-yes";
                String val3 ="Married,Housing-yes,Loan-no";
                String val4 ="Married,Housing-no,Loan-no";
                String val5 ="Single,Housing-no,Loan-no";
                String val6="Single,Housing-no,Loan-Yes";
                String val7 = "Single,Housing-no,Loan-Yes";
                String val8 = "Single,Housing-yes,Loan-Yes";
                if(marriage.equals("married") && housing.equals("yes"))
                {
                        if(fields[15].equals("yes"))
                        {
                                c.output(KV.of(val1, 1));
                        }
                }
                else if(marriage.equals("married") && housing.equals("no"))
                {
```

```
                if(fields[15].equals("yes"))
                {
                        c.output(KV.of(val2, 1));
                }
        }
        else if(marriage.equals("married") && housing.equals("yes"))
        {
                if(fields[15].equals("no"))
                {
                        c.output(KV.of(val3, 1));
                }
        }
        else if(marriage.equals("married") && housing.equals("no"))
        {
                if(fields[15].equals("no"))
                {
                        c.output(KV.of(val4, 1));
                }
        }
        else if(marriage.equals("single") && housing.equals("no"))
        {
                if(fields[15].equals("no"))
                {
                        c.output(KV.of(val5, 1));
                }
        }
        else if(marriage.equals("single") && housing.equals("yes"))
        {
                if(fields[15].equals("no"))
                {
                        c.output(KV.of(val6, 1));
                }
        }
        else if(marriage.equals("single") && housing.equals("no"))
        {
                if(fields[15].equals("yes"))
                {
                        c.output(KV.of(val7, 1));
                }
        }
        else if(marriage.equals("single") && housing.equals("yes"))
        {
                if(fields[15].equals("yes"))
                {
                        c.output(KV.of(val8, 1));
                }
        }

    }
}

private static class ConvertToStringFn extends DoFn<KV<String, Integer>, String> {

    @ProcessElement
    public void processElement(ProcessContext c) {
        c.output(c.element().getKey() + "," + c.element().getValue());
    }
```

```
    }
}
```

Output:



3. Analyse based on age and education the bank balance of the employees

Code:

```
package org.apache.beam.examples;

import org.apache.beam.sdk.Pipeline;
import org.apache.beam.sdk.options.PipelineOptions;
import org.apache.beam.sdk.options.PipelineOptionsFactory;
import org.apache.beam.sdk.io.TextIO;
import org.apache.beam.sdk.transforms.*;
import org.apache.beam.sdk.values.*;

import java.util.Calendar;

public class account {

    private static final String CSV_HEADER =
            "car,price,body,mileage,engV,engType,registration,year,model,drive";

    public static void main(String[] args) {
        PipelineOptions options = PipelineOptionsFactory.fromArgs(args).withValidation().create();
        Pipeline pipeline = Pipeline.create(options);

                    pipeline.apply("ReadAds",
TextIO.read().from("/home/harshita1262/word-count-beam/src/main/resources/bankfinal1.csv"))
```

```
                .apply("FilterHeader", ParDo.of(new FilterHeaderFn(CSV_HEADER)))
                .apply("MakeAgeKVFn", ParDo.of(new MakeAgeKVFn()))
                .apply(Sum.integersPerKey())
                .apply(ParDo.of(new ConvertToStringFn()))
                .apply("PrintToConsole", ParDo.of(new DoFn<String, Void>() {
                        @ProcessElement
                        public void processElement(ProcessContext c) {
                                System.out.println(c.element());
                        }
                }));

        //value.apply("Writeresult to file", TextIO.write().to("gs://albinbucket/amita/output"));

        pipeline.run().waitUntilFinish();
}

private static class FilterHeaderFn extends DoFn<String, String> {

        private final String header;

        public FilterHeaderFn(String header) {
                this.header = header;
        }

        @ProcessElement
        public void processElement(ProcessContext c) {
                String row = c.element();

                if (!row.isEmpty() && !row.equals(this.header)) {
                        c.output(row);
                }
        }
}

private static class MakeAgeKVFn extends DoFn<String, KV<String, Integer>> {

        @ProcessElement
        public void processElement(ProcessContext c) {
                String[] fields = c.element().split(",");
                if(!(fields[23].equals("BALANCE"))){
                int sal=Integer.parseInt(fields[23]);

                c.output(KV.of(fields[1]+" , "+fields[11],sal));
                }

        }
}
```

```
    private static class ConvertToStringFn extends DoFn<KV<String, Integer>, String> {

        @ProcessElement
        public void processElement(ProcessContext c) {
            c.output(c.element().getKey() + "," + c.element().getValue());
        }
    }
}
```