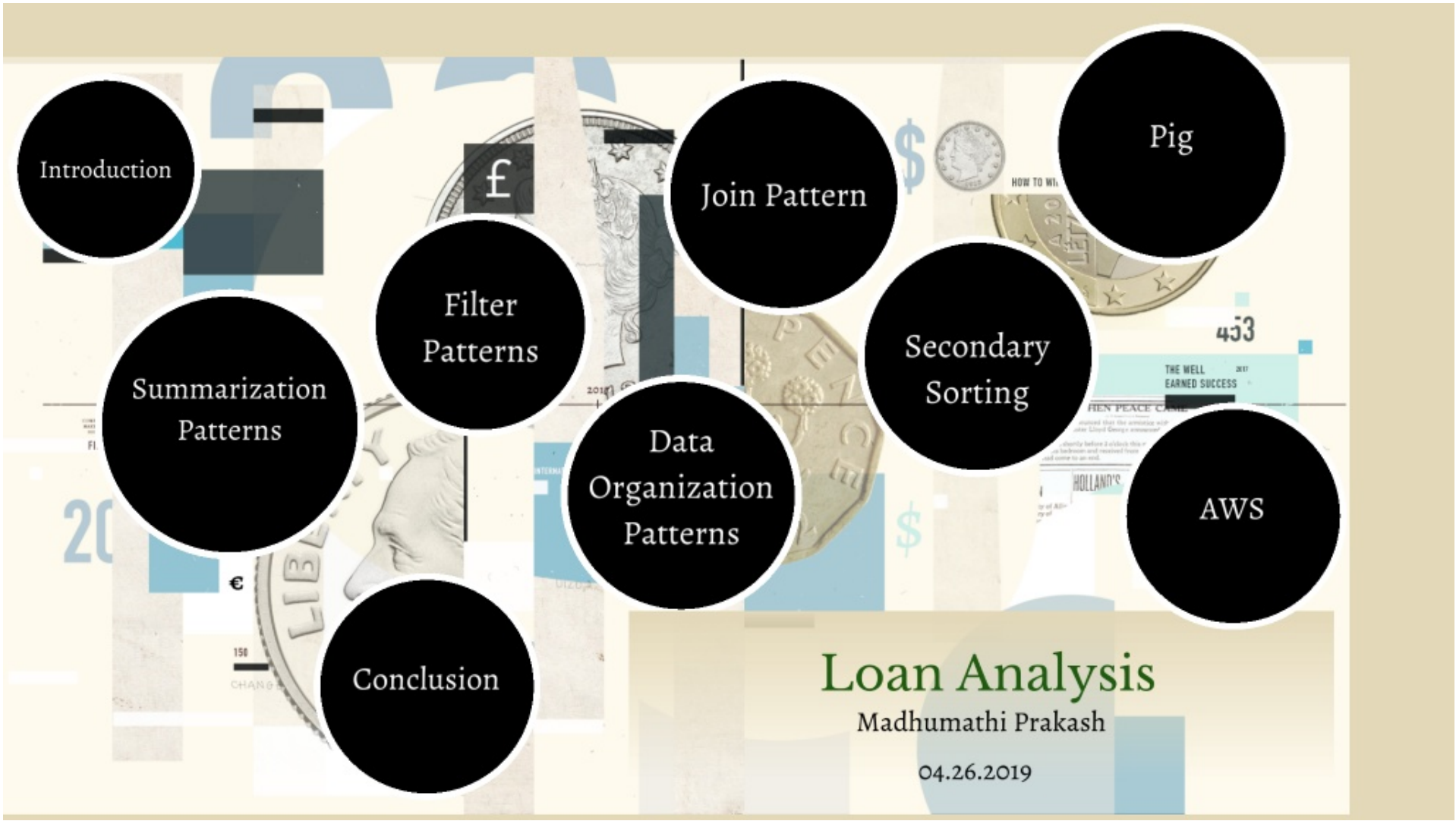
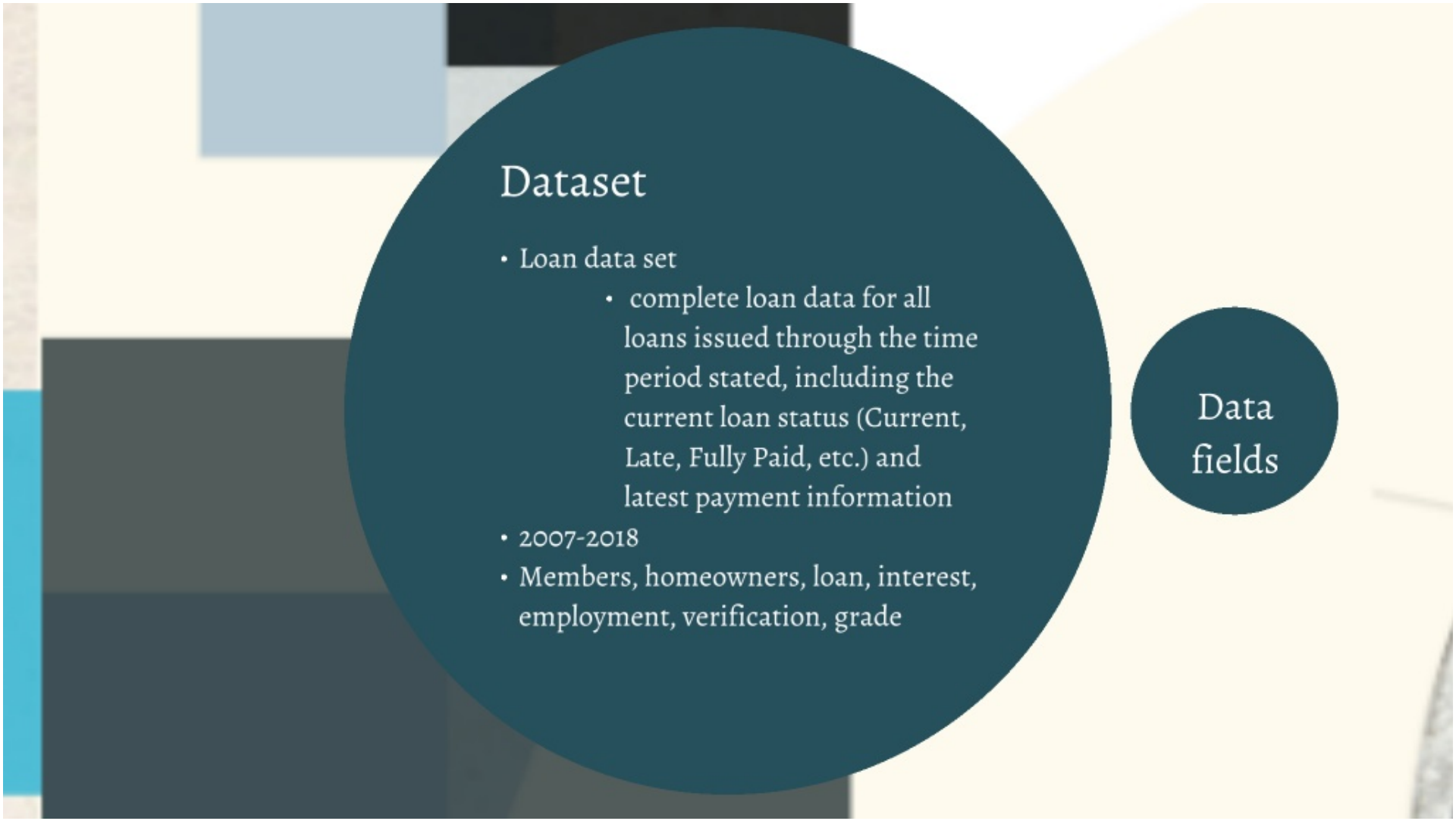

Introduction
Filter Patterns
Join Pattern
Pig
Summarization Patterns
Secondary Sorting
Data Organization Patterns
AWS
Conclusion
Loan Analysis
Madhumathi Prakash
04.26.2019

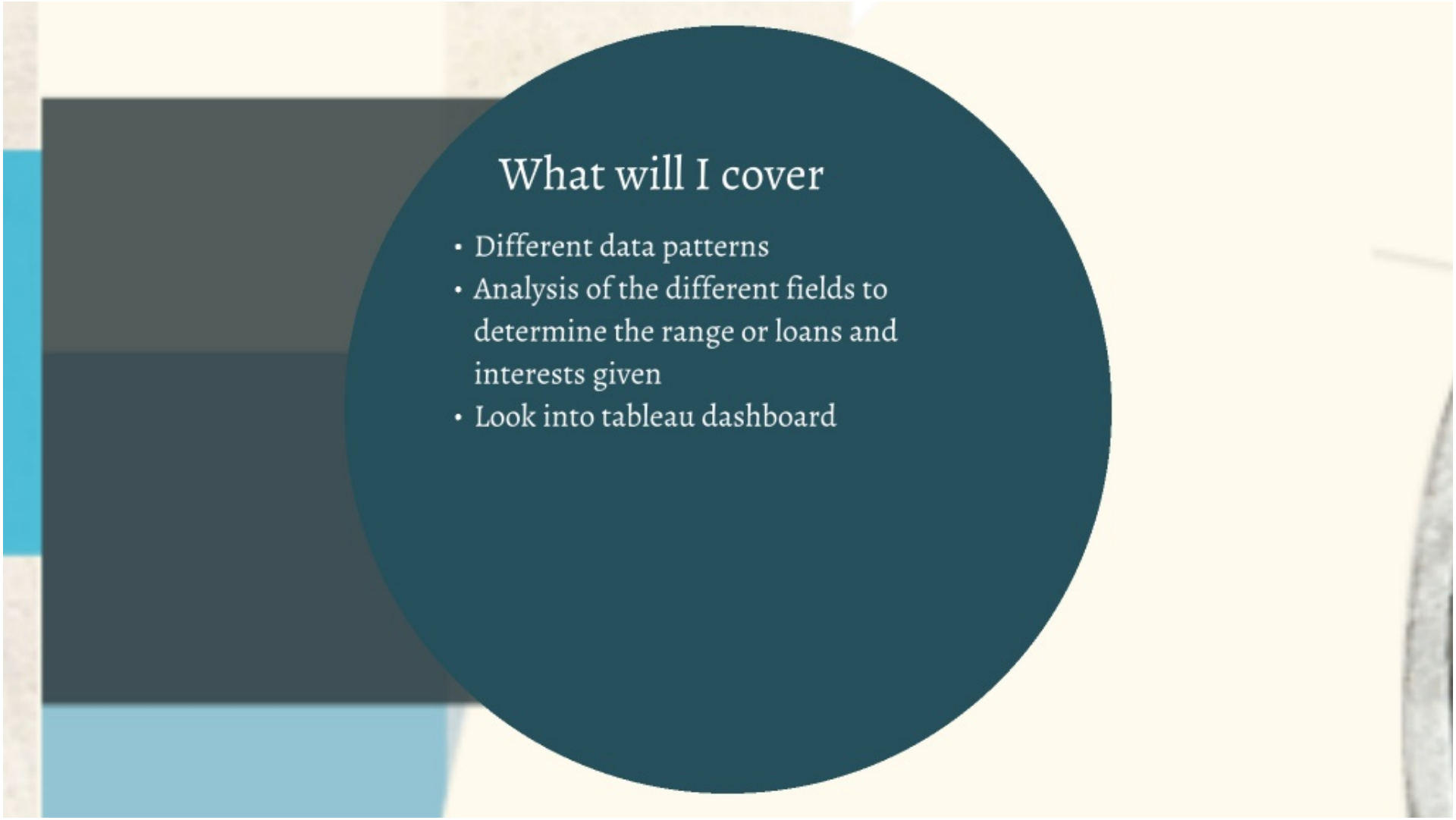

Introduction
Dataset link: https://www.lendingclub.com/info/download-data.action
Lending Club
Dataset
Overview
Putmerge


About
LendingClub
• Help millions of people take control of their debt, grow their small businesses, and invest for the future
• Invest or Borrow
• Personal, business, education loans


Dataset
• Loan data set
• complete loan data for all loans issued through the time period stated, including the current loan status (Current, Late, Fully Paid, etc.) and latest payment information
• 2007-2018
• Members, homeowners, loan, interest, employment, verification, grade
Data fields

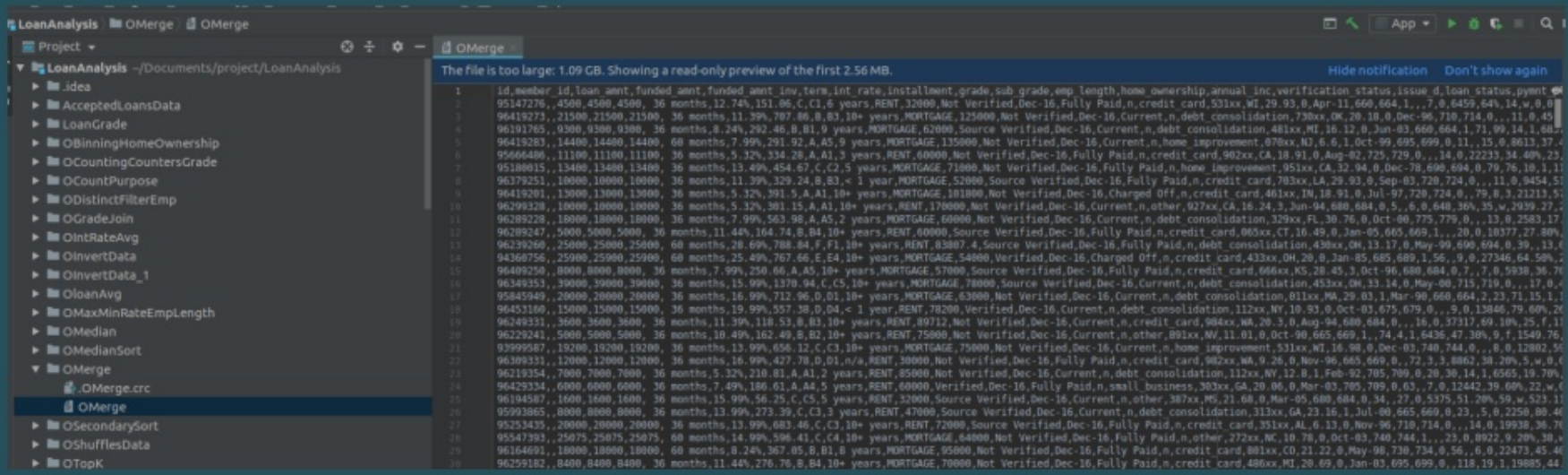
Data Fields

Data Field	Description
addrState	The state provided by the borrower in the loan application
annualInc	The self-reported annual income provided by the borrower during registration.
application_type	Indicates whether the loan is an individual application or a joint application with two co-borrowers
delinq2Yrs	The number of 30+ days past-due incidences of delinquency in the borrower's credit file for the past 2 years
delinqAmnt	The past-due amount owed for the accounts on which the borrower is now delinquent.
empLength	Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years.
expDefaultRate	The expected default rate of the loan.
ficoRangeHigh	The upper boundary range the borrower's FICO at loan origination belongs to.
ficoRangeLow	The lower boundary range the borrower's FICO at loan origination belongs to.
grade	LC assigned loan grade
homeOwnership	The home ownership status provided by the borrower during registration. Our values are: RENT, OWN, MORTGAGE, OTHER.
id	A unique LC assigned ID for the loan listing.
installment	The monthly payment owed by the borrower if the loan originates.
intRate	Interest Rate on the loan
loanAmnt	The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.
purpose	A category provided by the borrower for the loan request.
subGrade	LC assigned loan subgrade
term	The number of payments on the loan. Values are in months and can be either 36 or 60.
totalAcc	The total number of credit lines currently in the borrower's credit file
verified_status_joint	Indicates if the co-borrowers' joint income was verified by LC, not verified, or if the income source was verified
zip_code	The first 3 numbers of the zip code provided by the borrower in the loan application.

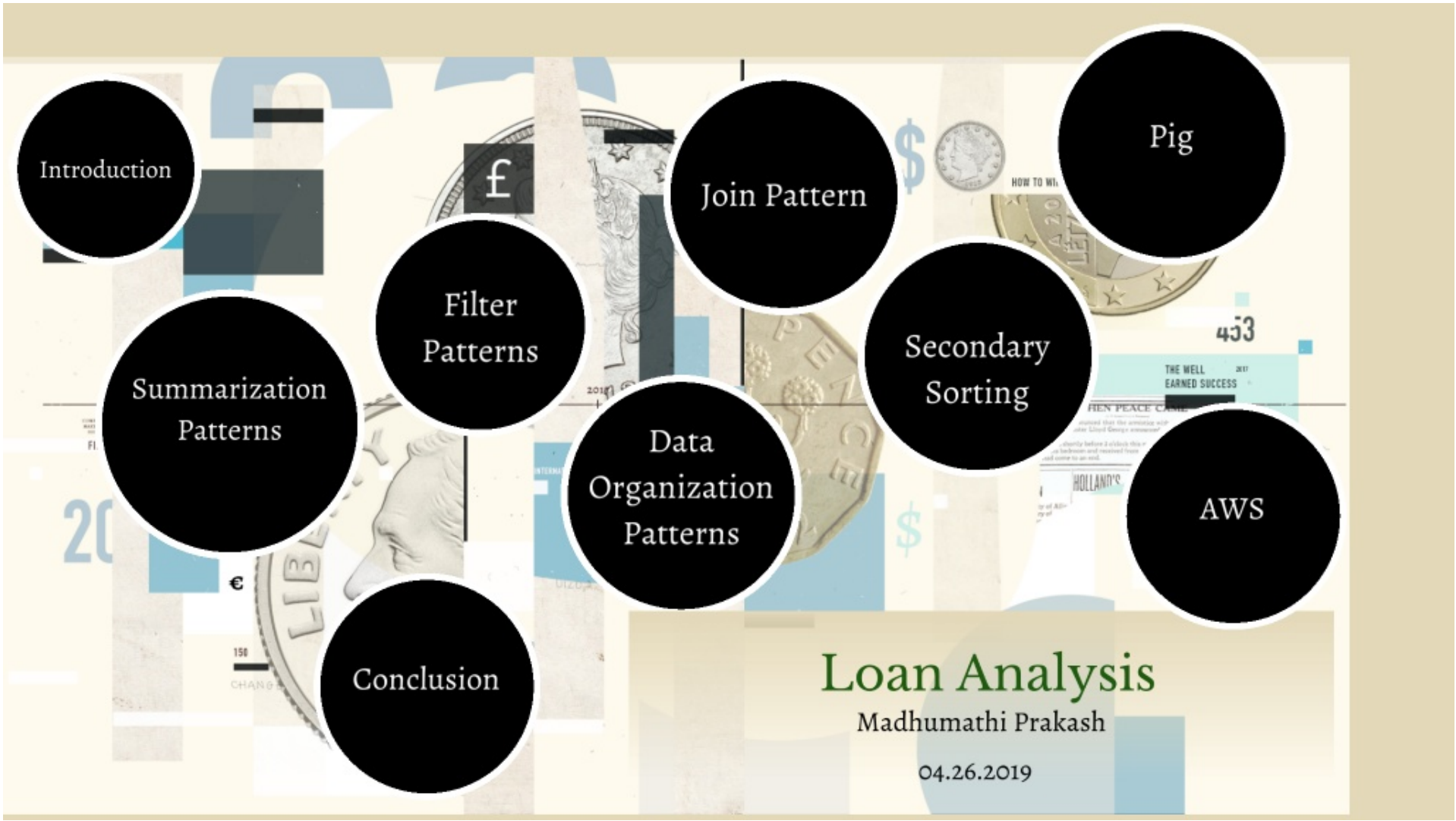

What will I cover
• Different data patterns
• Analysis of the different fields to determine the range or loans and interests given
• Look into tableau dashboard

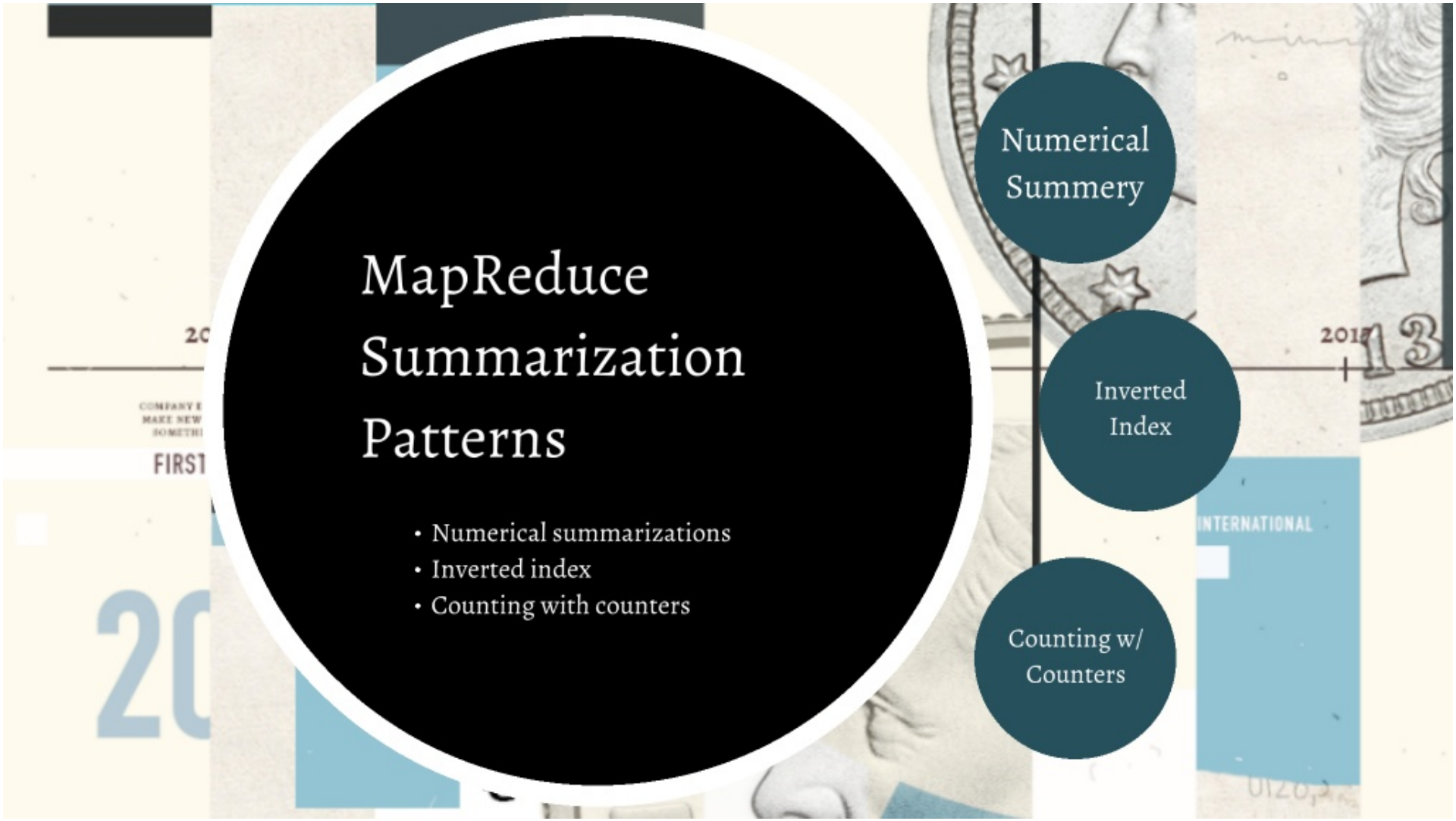
Putmerge

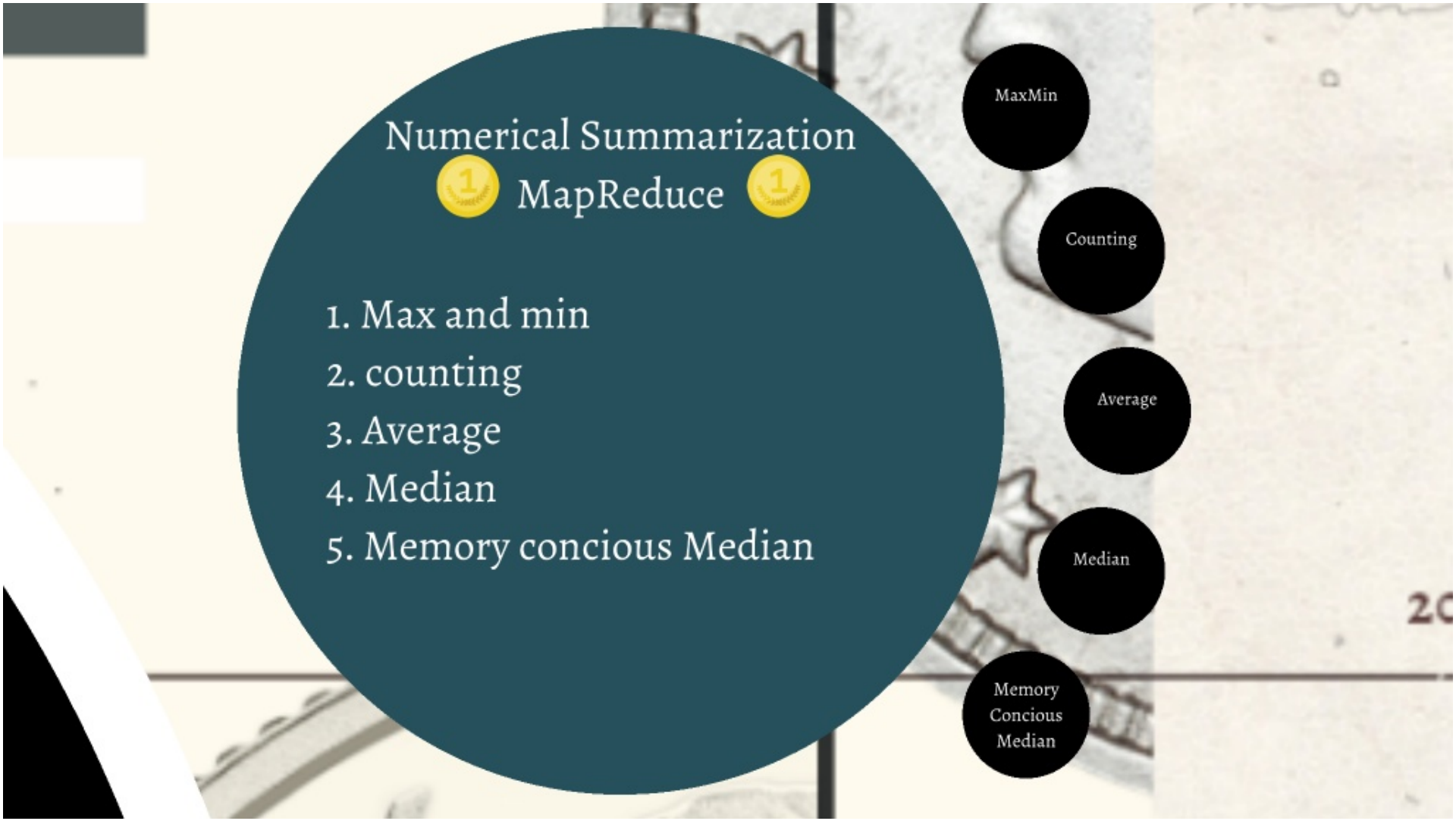
Combined all CSVs dealing with loan data - utilize just the driver



Take data csv as input and put in HDFS as 1 files that is output

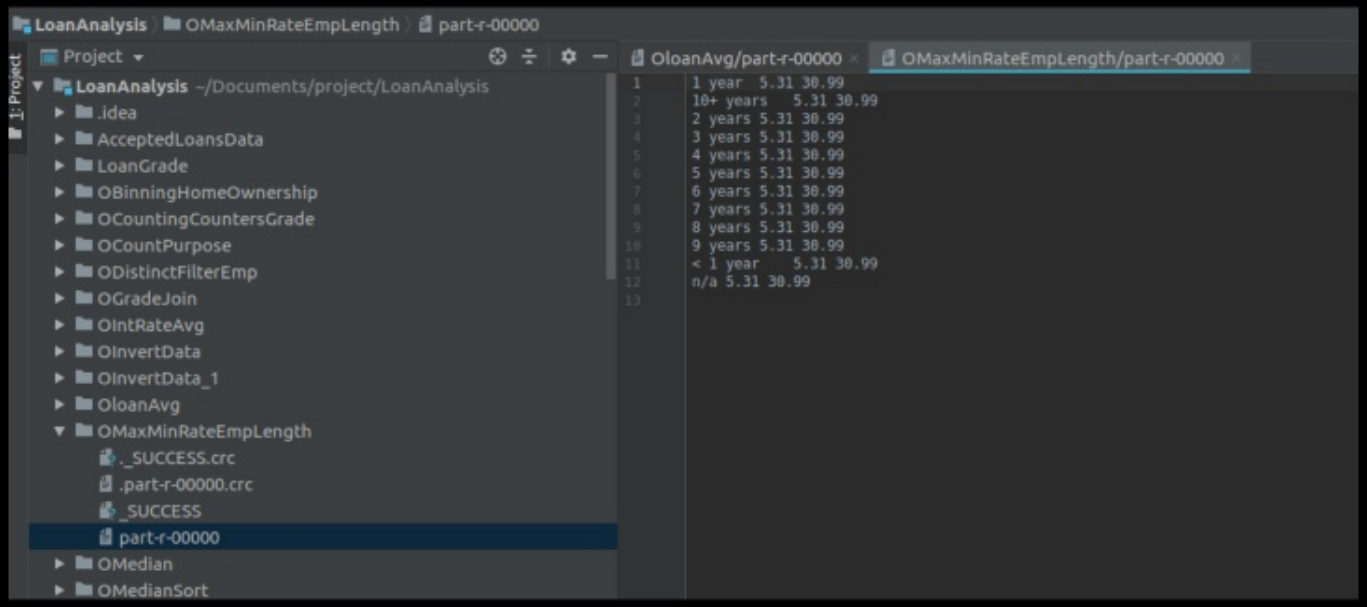

Introduction
Filter Patterns
Join Pattern
Pig
Summarization Patterns
Secondary Sorting
Data Organization Patterns
AWS
Conclusion
Loan Analysis
Madhumathi Prakash
04.26.2019


MapReduce
Summarization
Patterns
• Numerical summarizations
• Inverted index
• Counting with counters
Numerical
Summery
Inverted
Index
Counting w/
Counters


Numerical Summarization
MapReduce
1. Max and min
2. counting
3. Average
4. Median
5. Memory concious Median
MaxMin
Counting
Average
Median
Memory Concious Median

Maximum and Minimum

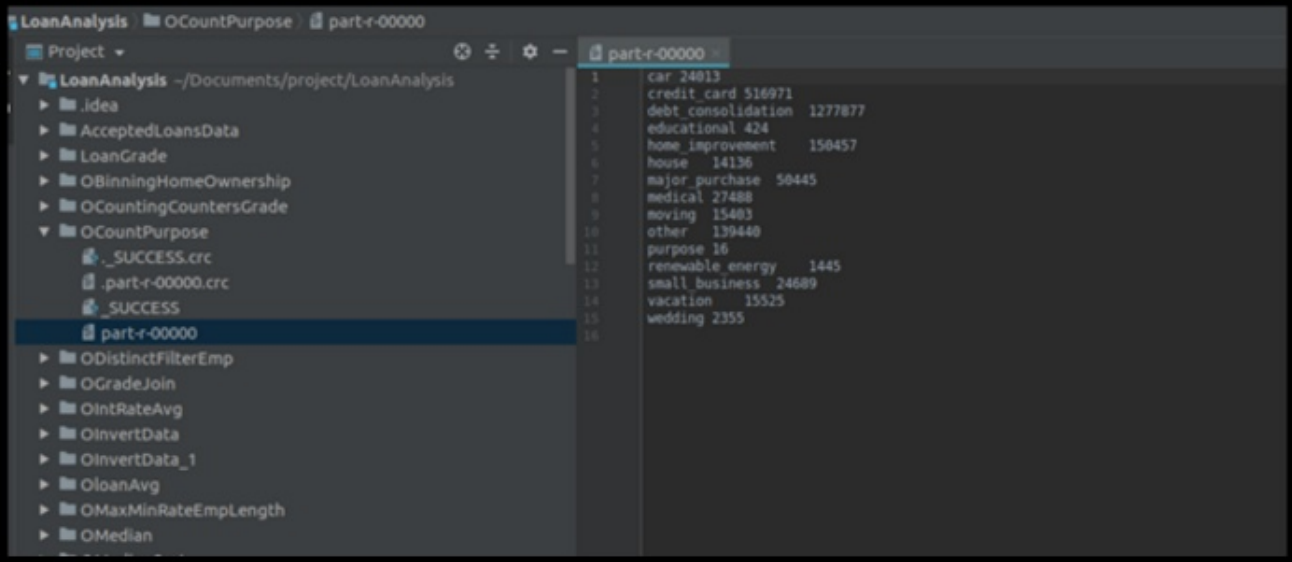
Found Minimum and Maximum interest Rate based on the Length of Employment in years



Utilized NLine Mapper: Key as employment and value passing interest rate

Reducer: Key as employment and value concat the calculated max and min

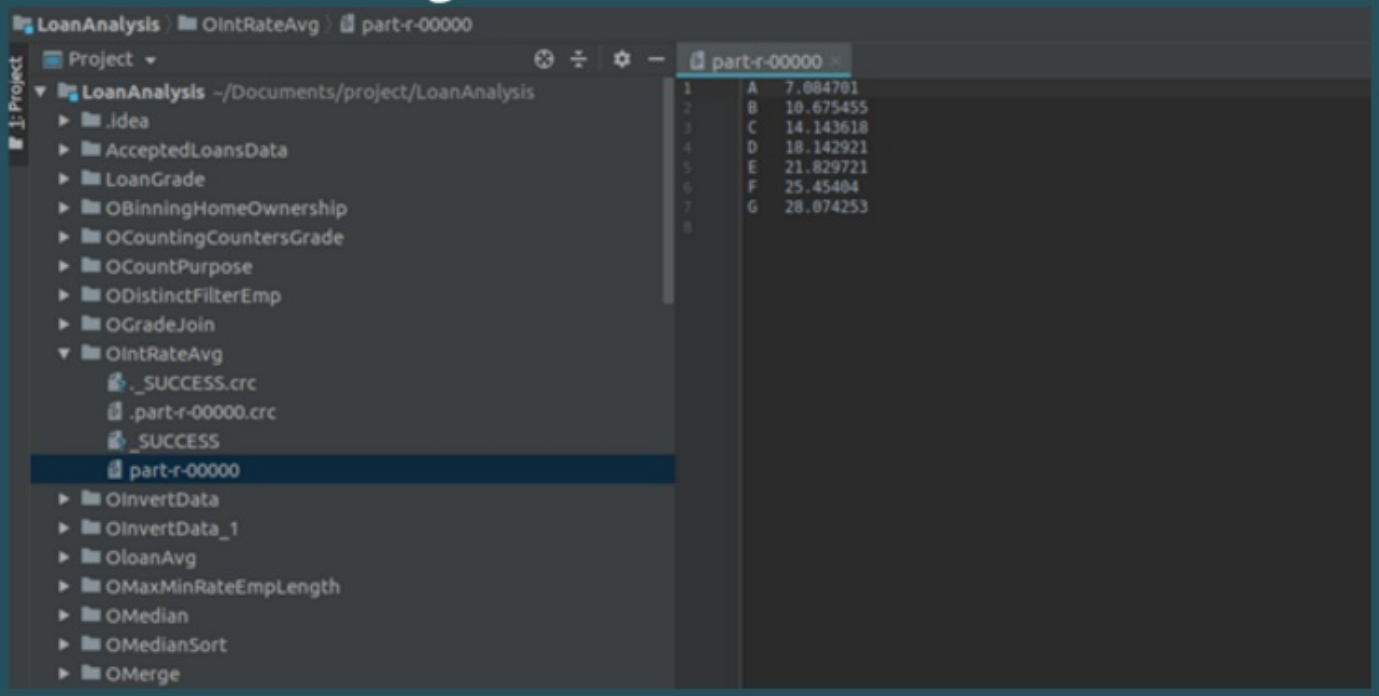
Determined number of data based on the Purpose



Key: purpose Value: sum of count


Average
interest Rate
Loan

Calculated Average for Interest rate based on grade

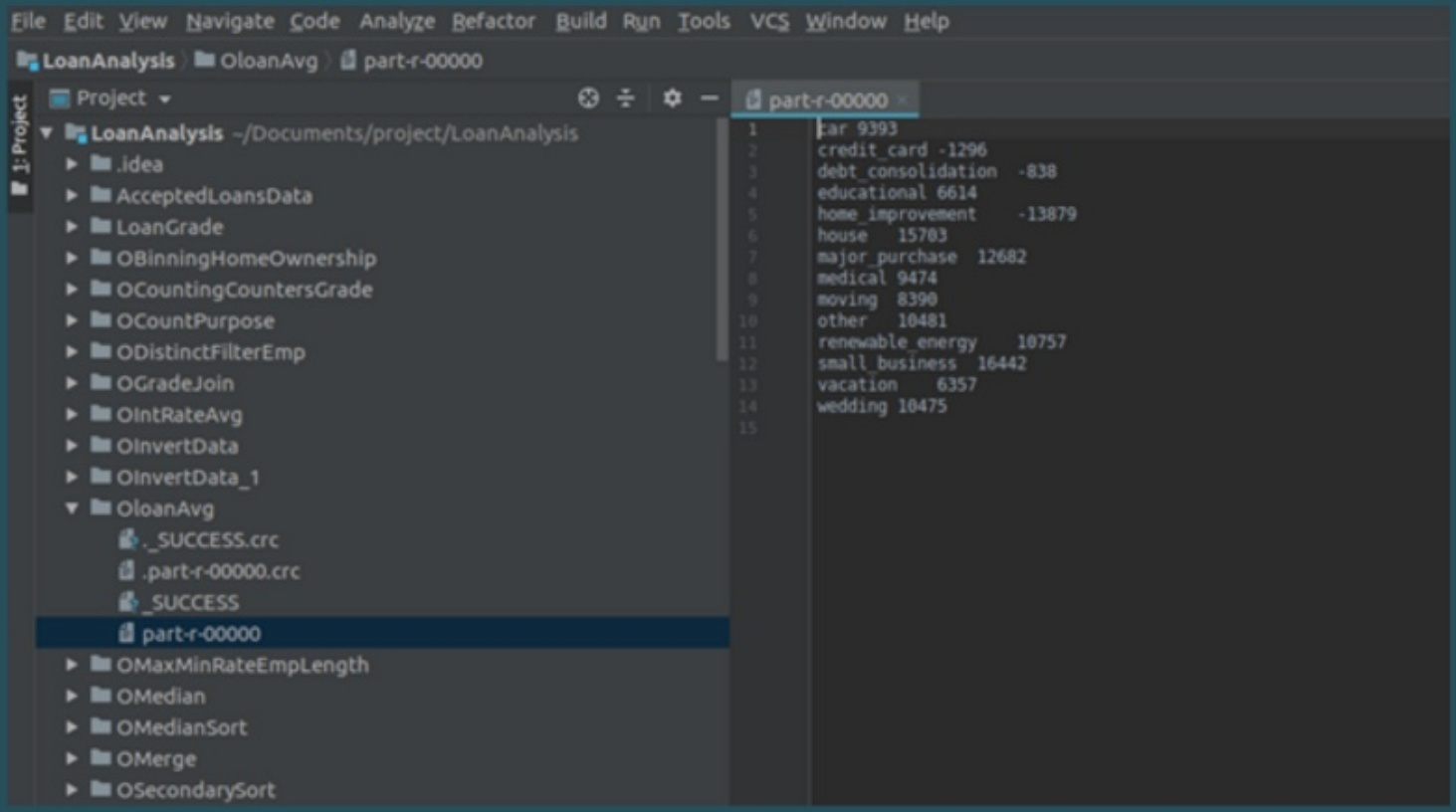


In map we create a pair of the sum and count of the average

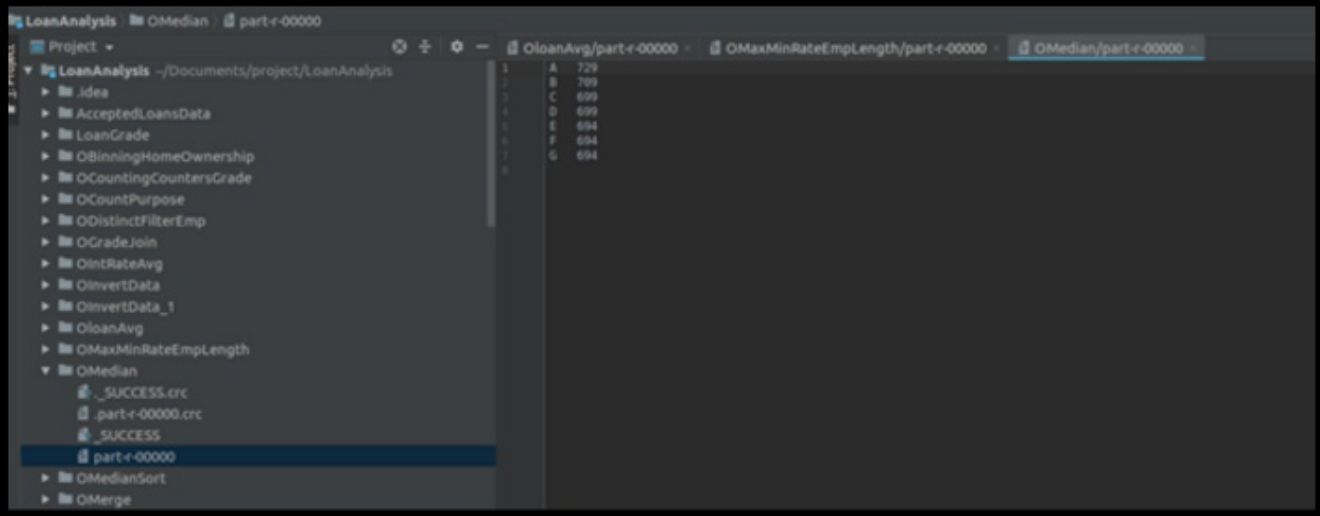
Cleanup Context.write the key being grade and value being pair of sum and count

Allows the use of combiner that reducces the work of reducer

Calculated Average of loan based on the Purpose

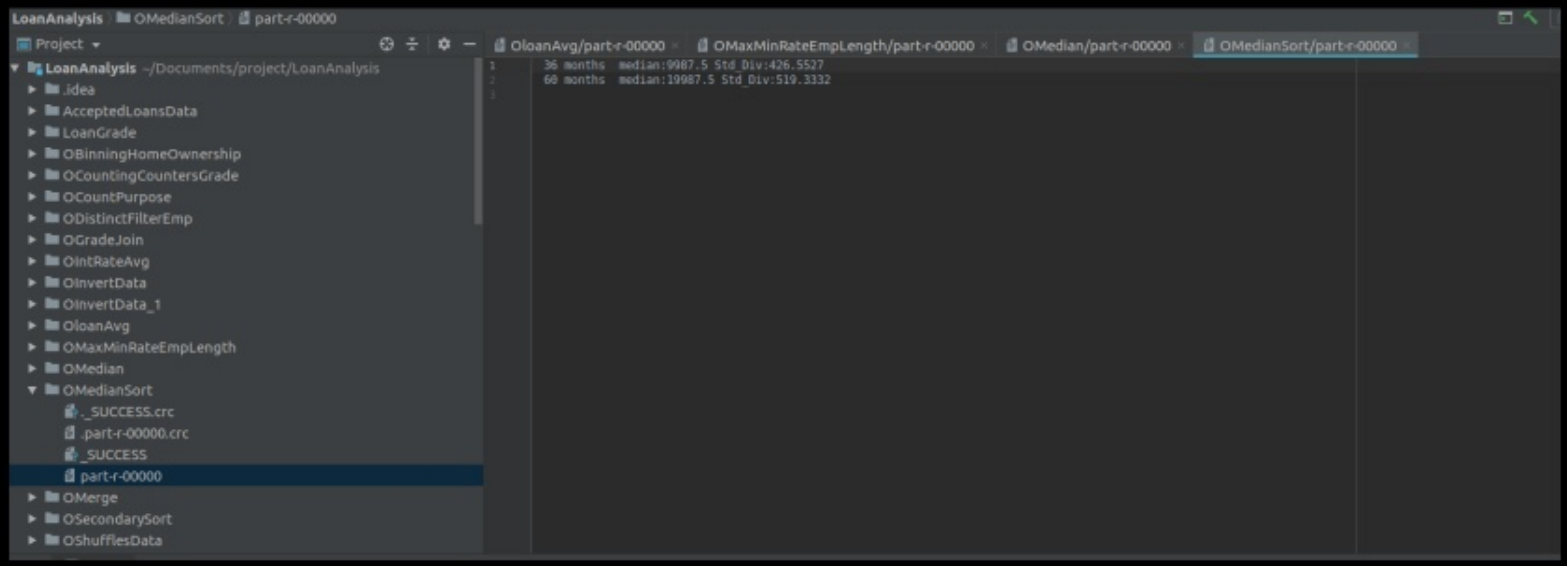


Calculated Median of Fico score based on the Grade.



- Reducer adding everything to arraylist and finding the middle fico score for each grade
- The loan grade is the result of a formula that takes into account not only credit score, but also a combination of several indicators of credit risk from the credit report and loan application

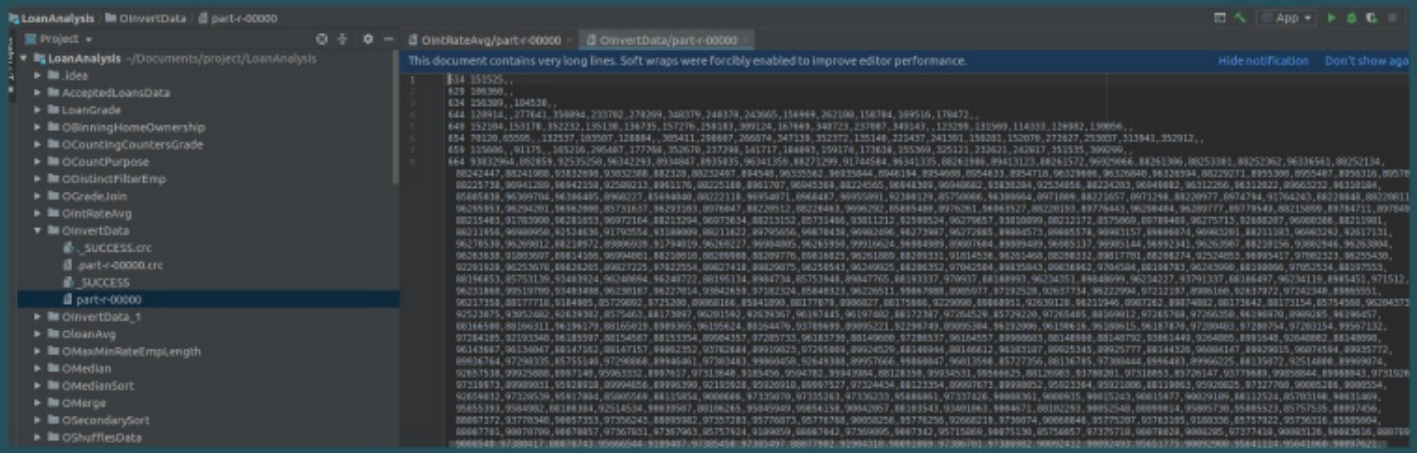
Memory Conscious Median of Loan Amount based on the Verification status



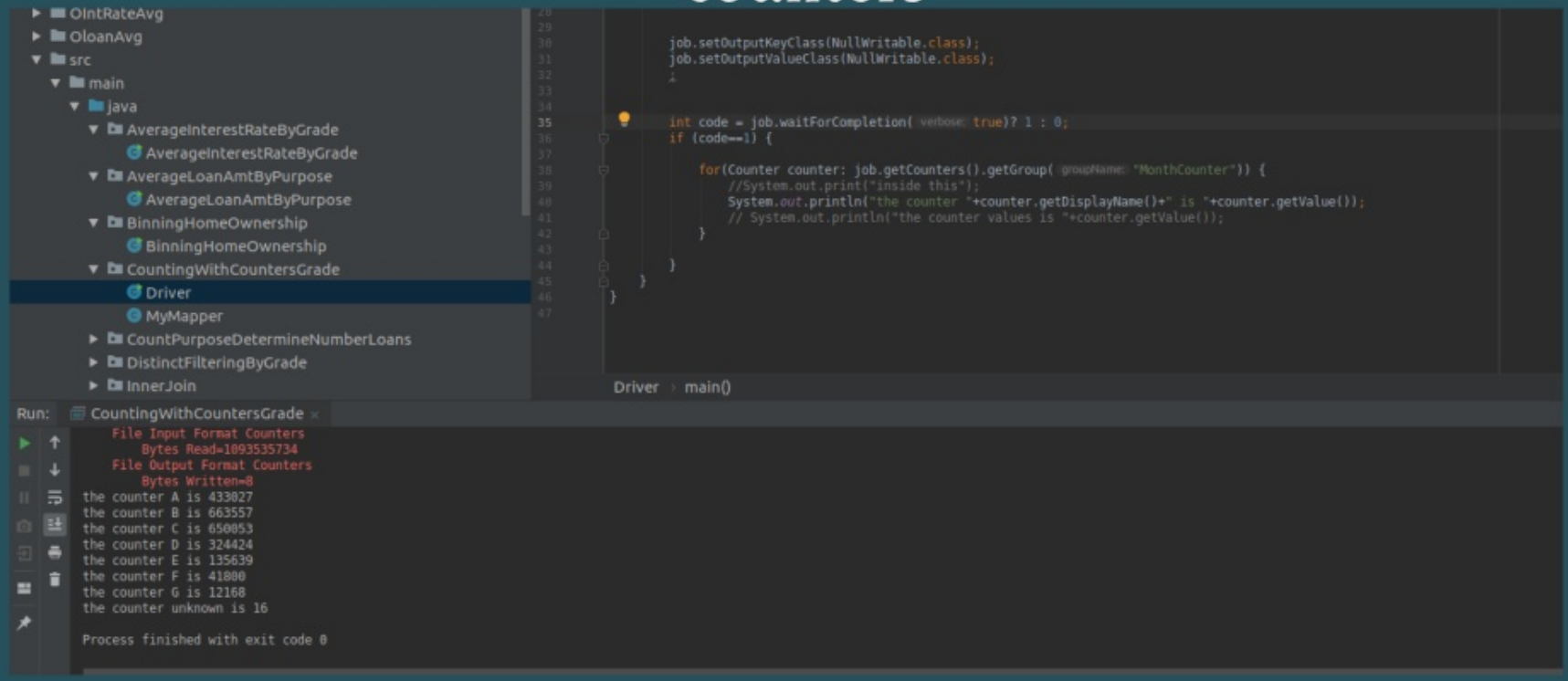
Utilizing Combiner and displaying both the median for 36 month and 60 month verification status

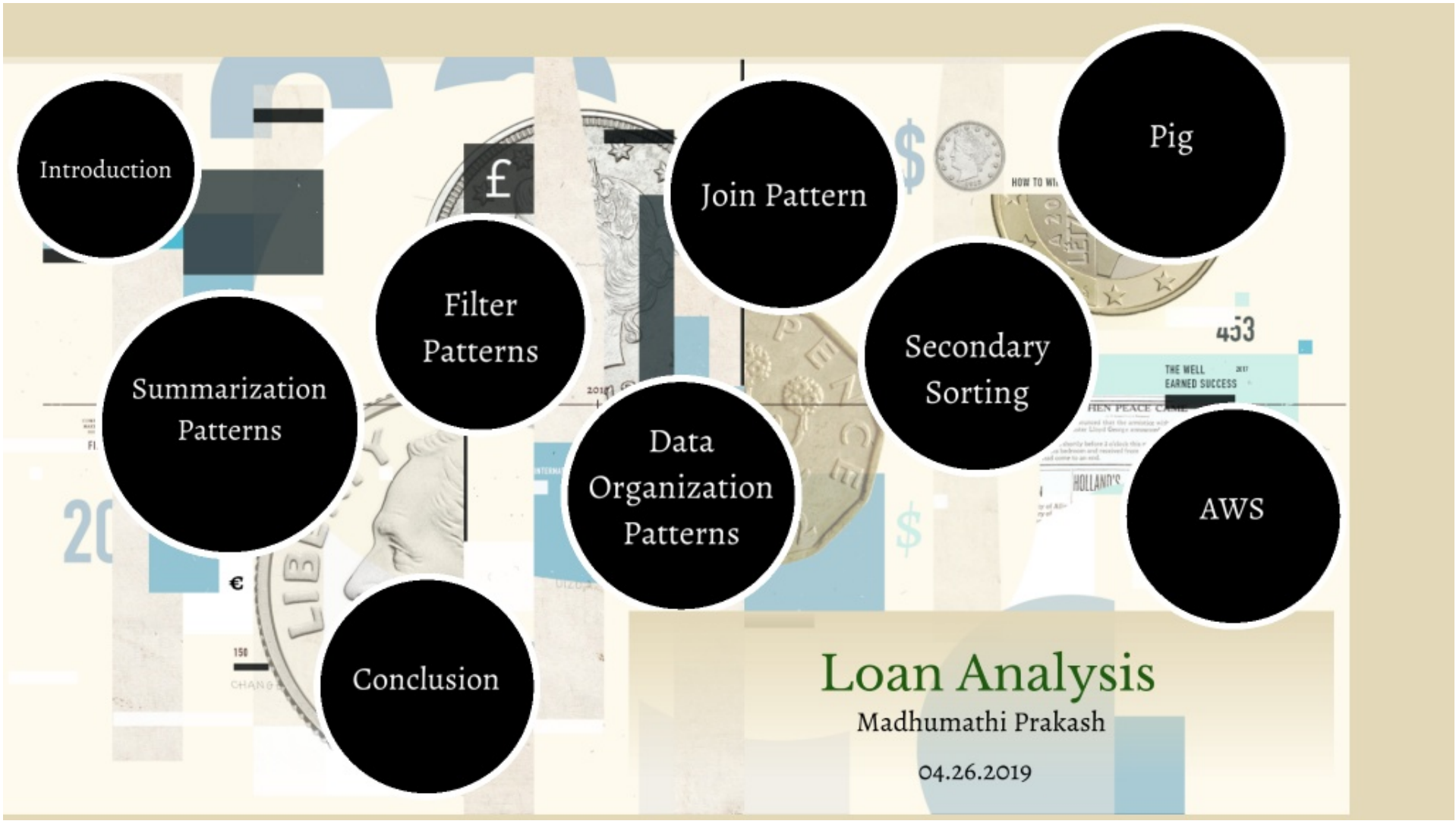
Inverted Index Pattern

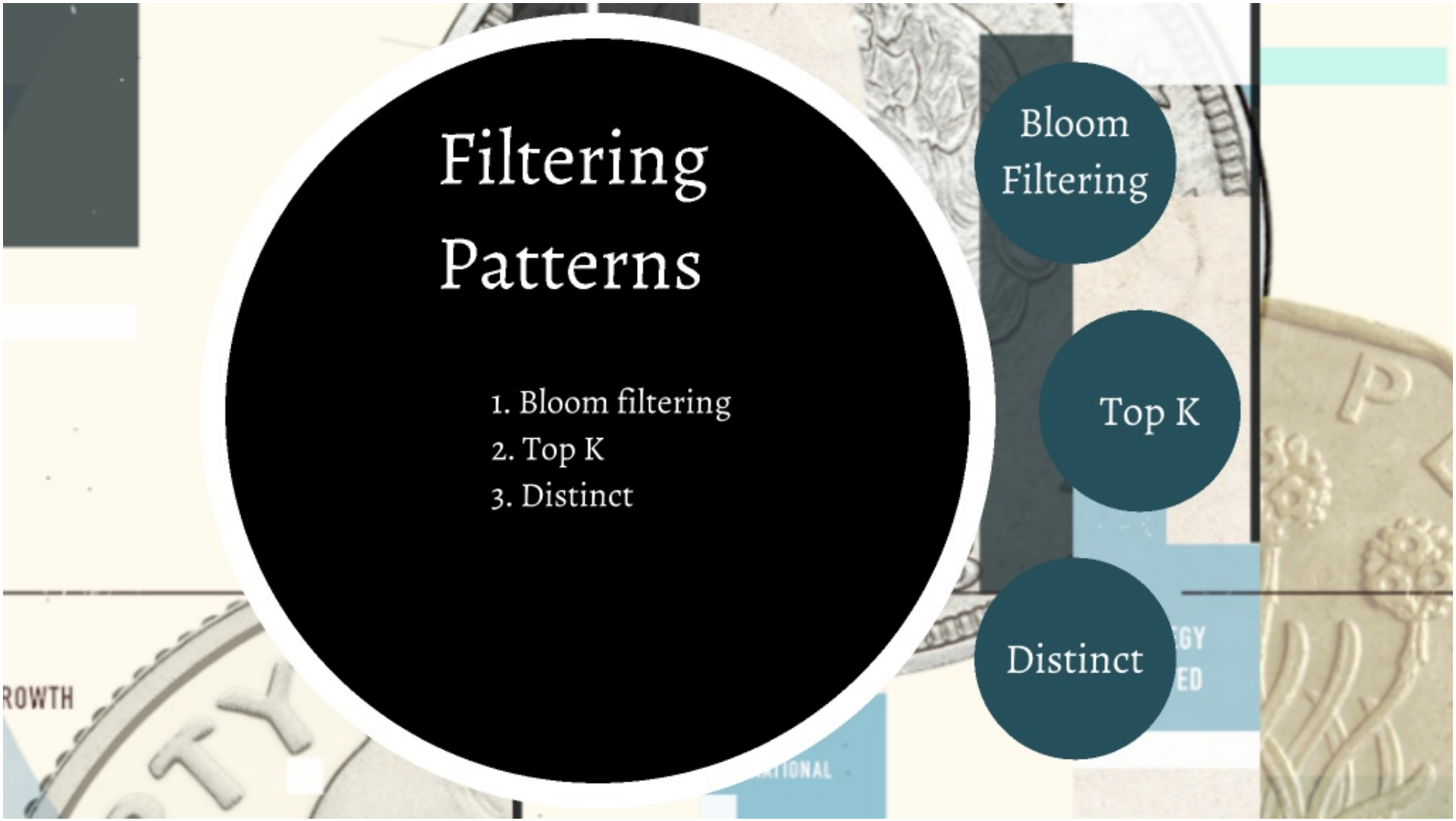
Used chaining for filtered Data and found the list loan ids based on the Fico score as key



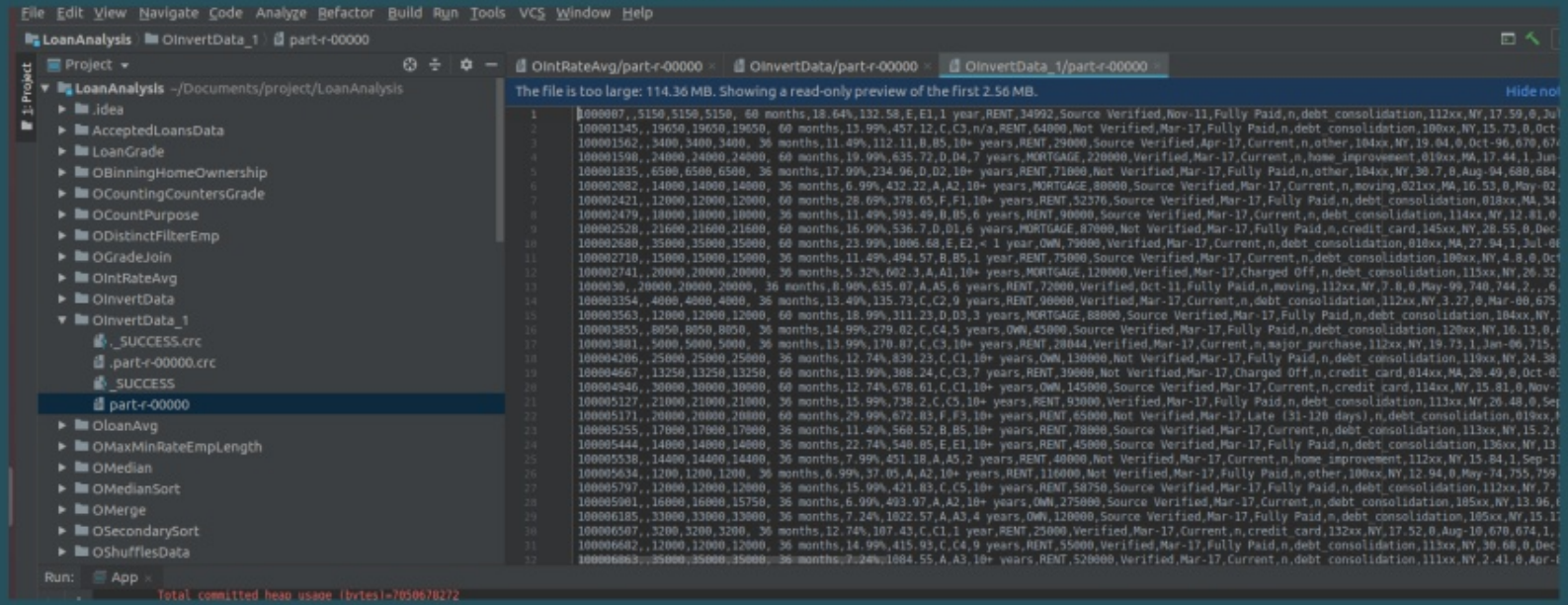
Counting the number of data based on grades using counters




Introduction
Filter Patterns
Join Pattern
Pig
Summarization Patterns
Secondary Sorting
Data Organization Patterns
AWS
Conclusion
Loan Analysis
Madhumathi Prakash
04.26.2019

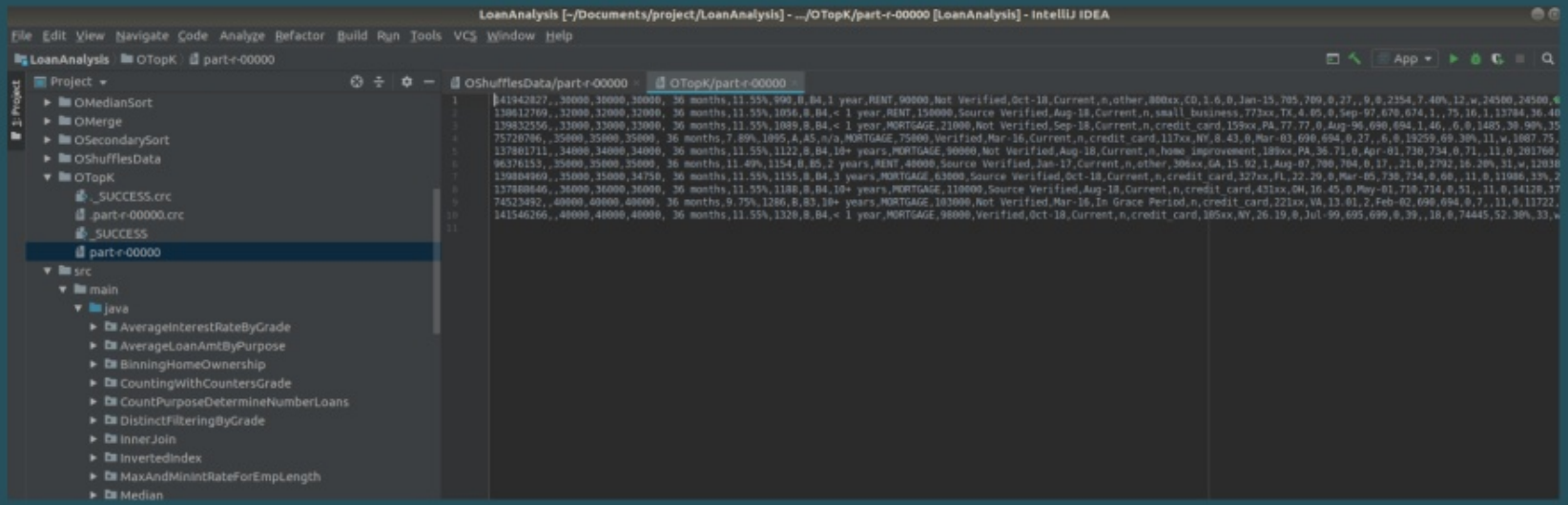

Filtering Patterns
1. Bloom filtering
2. Top K
3. Distinct
Bloom Filtering
Top K
Distinct

Used Bloom Filter to filter the data and select only the data related to MA and NY



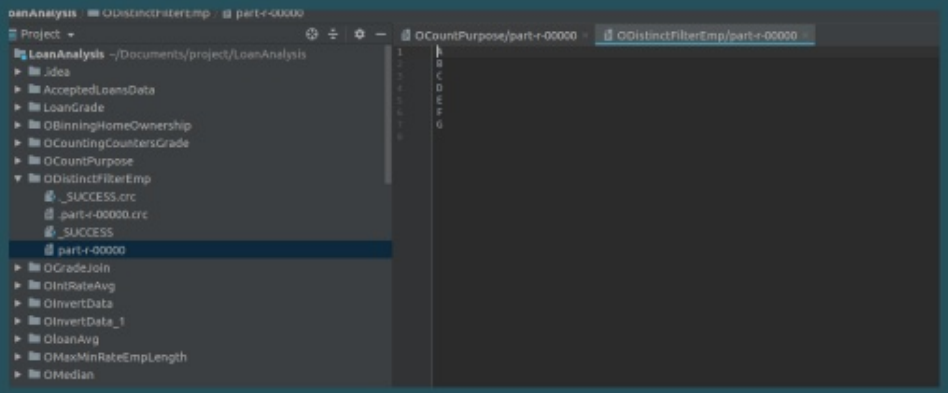
- Put the 2 items in the this.origin = BloomFilter in the setup
- In map checking if data coming in is matching the BloomFilter

Identified the Top 10 members based on the Loan Amount



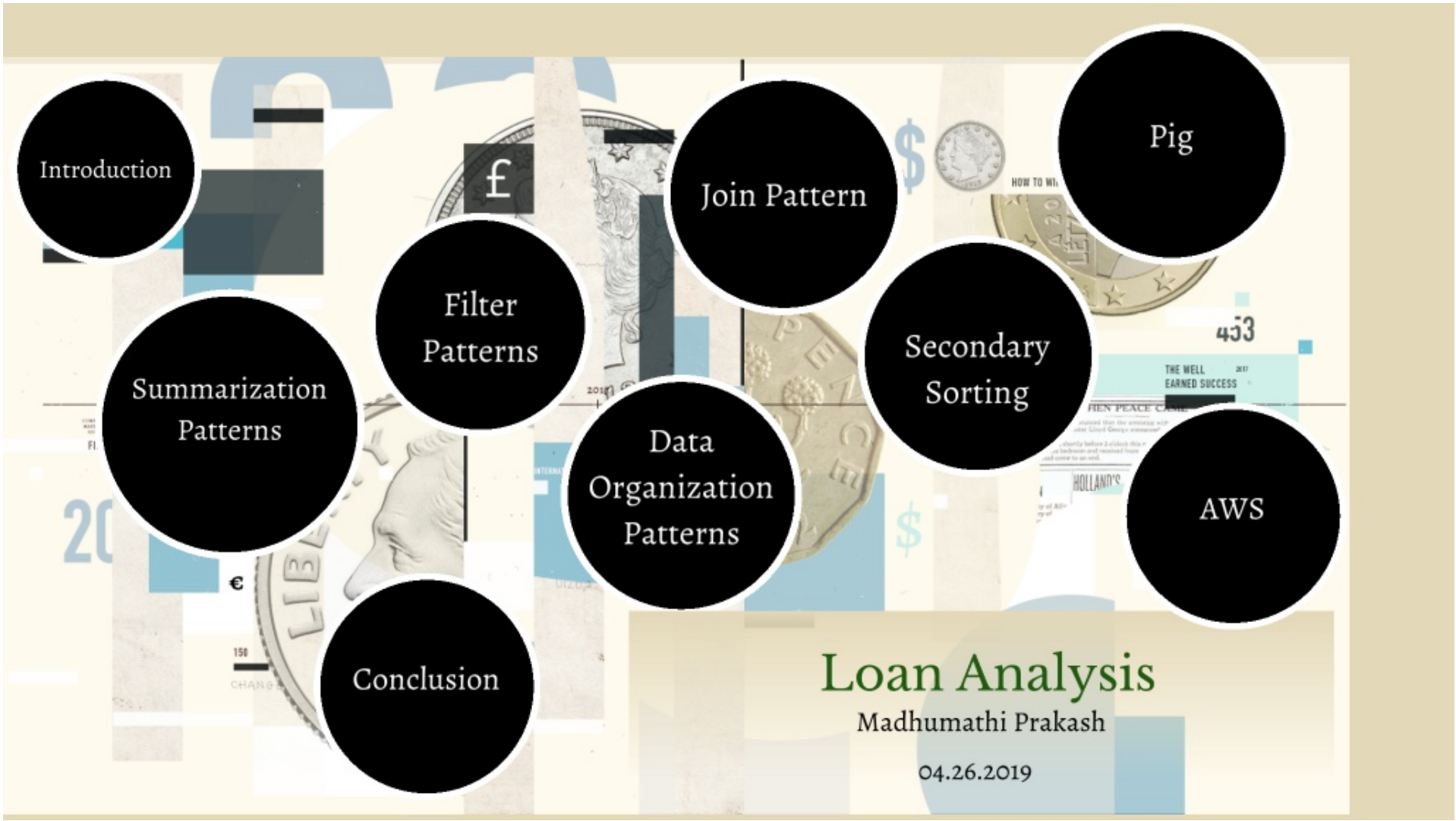
To get the top 10 checking that tMap.size() > K in the map and doing the context.write in the cleanup of the mapper

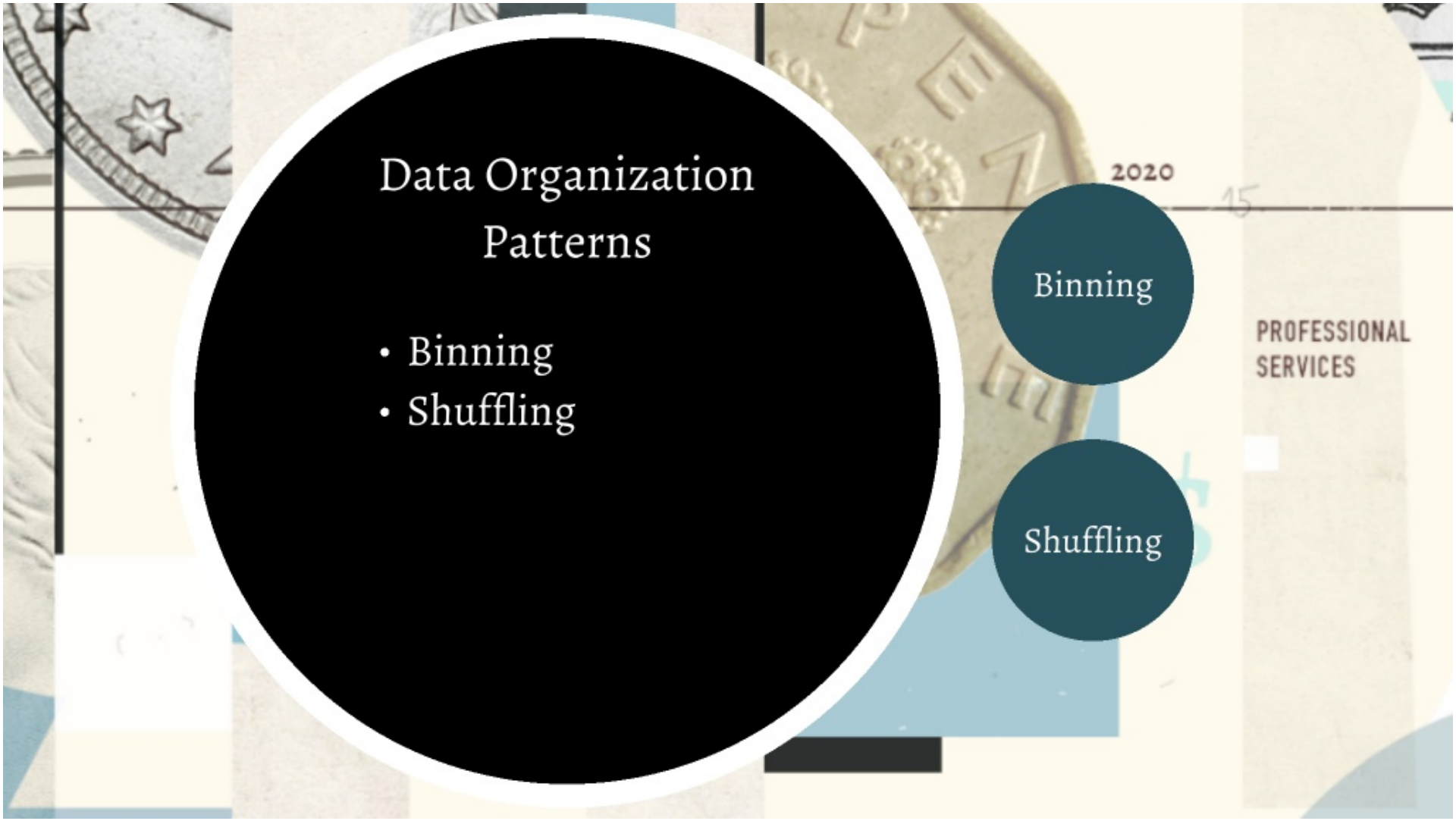
Distinct grades



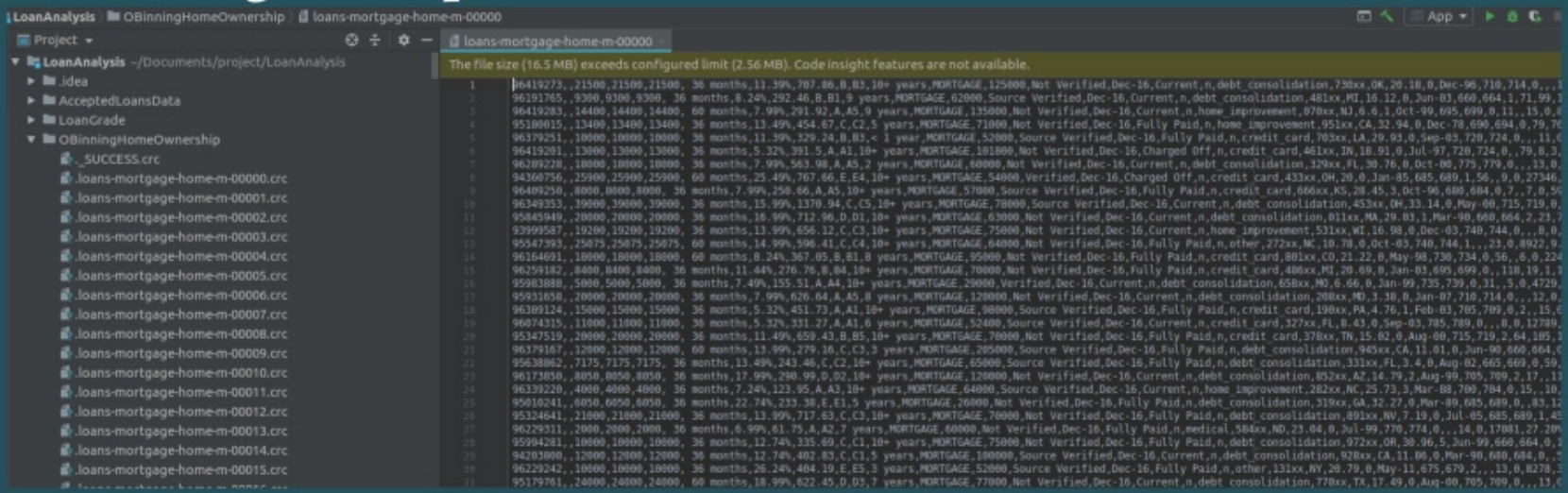
The key will have the grade passed and the value will be NullWritable to keep it unique

```java
@Override
protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text, NullWritable>.Context context)
        throws IOException, InterruptedException {
    String line = value.toString();

    String[] tokens = line.split( regex: ",");
    if(tokens.length > 15 && !line.contains("grade")) {
        context.write(new Text(tokens[8]), NullWritable.get()); //10
    }

}
```


Introduction
Filter Patterns
Join Pattern
Pig
Summarization Patterns
Secondary Sorting
Data Organization Patterns
AWS
Conclusion
Loan Analysis
Madhumathi Prakash
04.26.2019


Data Organization Patterns
• Binning
• Shuffling
2020
Binning
PROFESSIONAL SERVICES
Shuffling

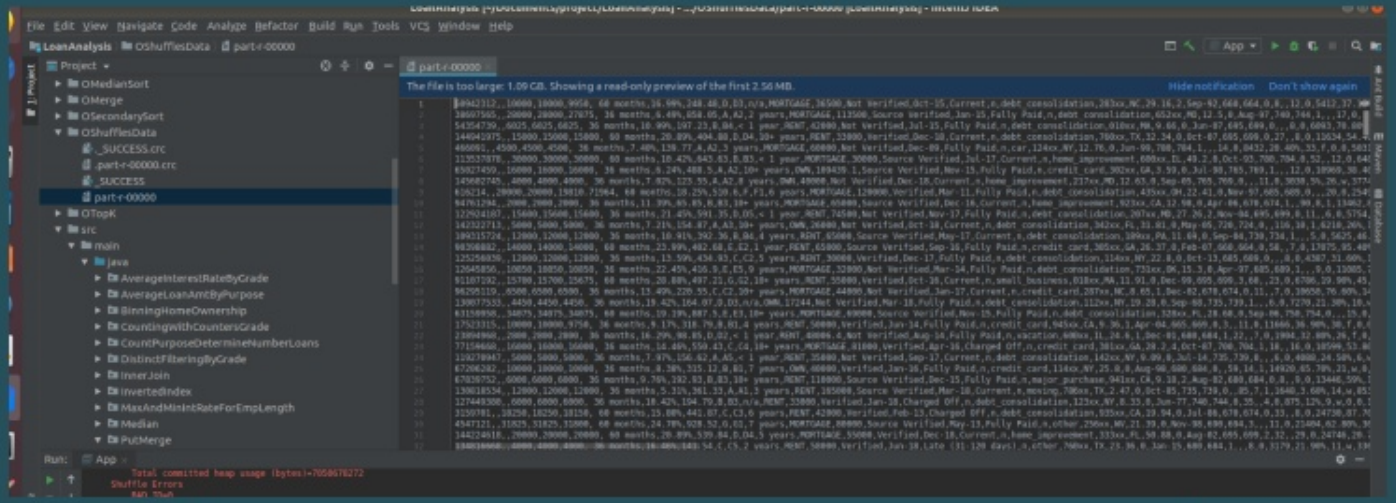
Binning to separate the file based on Home Ownership



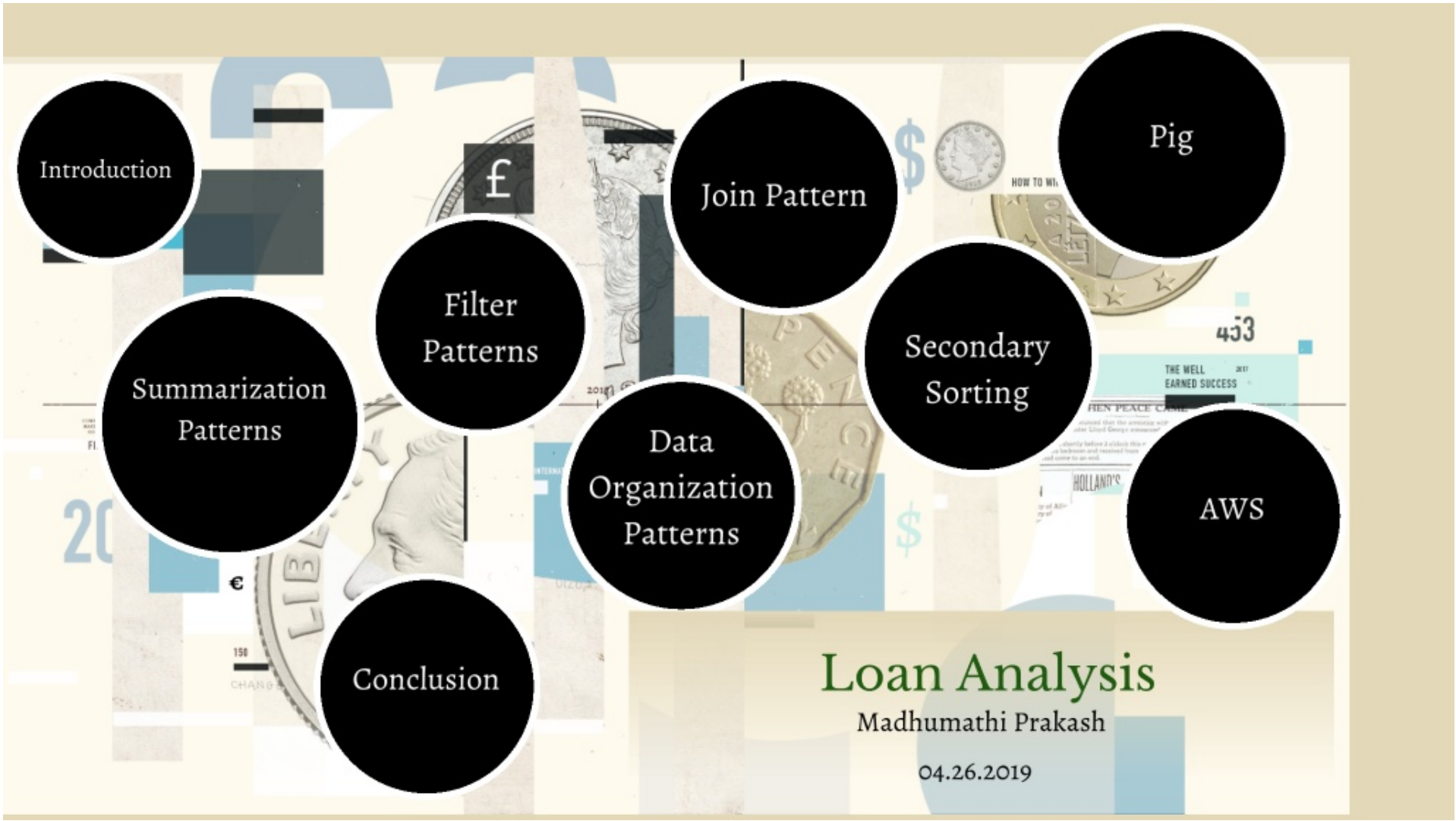
Positive is the use of only a Mapper and no need of partitioner and reducer

```
@Override
protected void setup(Context context) throws IOException, InterruptedException
{
    mos = new MultipleOutputs(context);
}

@Override
public void map(Object key, Text value, Context context)
        throws IOException, InterruptedException
{

    String line = value.toString();
    String row[] = value.toString().split( regex: ",");

    if(row.length > 15 && !line.contains("home_ownership") ){
        String cancellationCode = row[11];

        // System.out.println(cancellationCode);
        if(cancellationCode.contains("MORTGAGE"))
            mos.write( namedOutput: "bins", value, NullWritable.get(), baseOutputPath: "loans-mortgage-home");
        if(cancellationCode.contains("OWN"))
            mos.write( namedOutput: "bins", value, NullWritable.get(), baseOutputPath: "loans-own-home");
        if(cancellationCode.contains("RENT"))
            mos.write( namedOutput: "bins", value, NullWritable.get(), baseOutputPath: "loans-rent-home");

    }
```

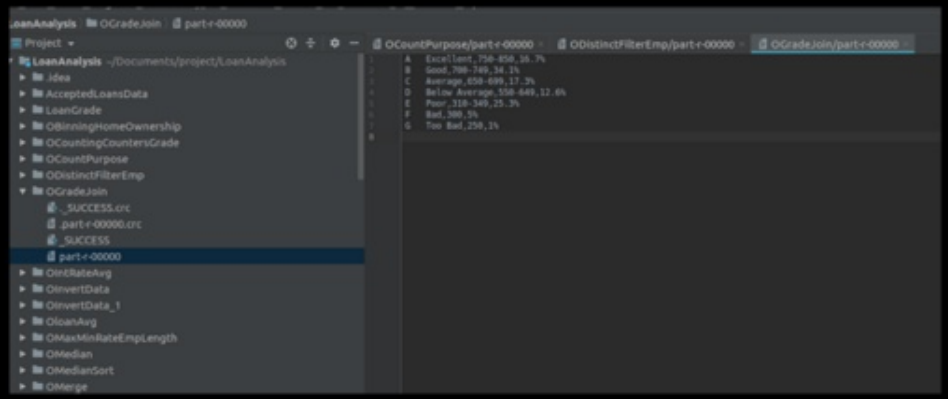
Shuffled the Data in order to randomize the data




```java
public class ShuffleMapper extends Mapper<LongWritable, Text,IntWritable, Text> {
    private static final IntWritable one = new IntWritable( value: 1);
    private Random random= new Random();
    @Override
    protected void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        String line = value.toString();

        String[] tokens = line.split( regex: ",");
        if(tokens.length > 15) {
            context.write(new IntWritable(random.nextInt()), value);
        }

    }
```


Introduction
Filter Patterns
Join Pattern
Pig
Summarization Patterns
Secondary Sorting
Data Organization Patterns
AWS
Conclusion
Loan Analysis
Madhumathi Prakash
04.26.2019

Used Joins to join the distinct grade to its data in different csv



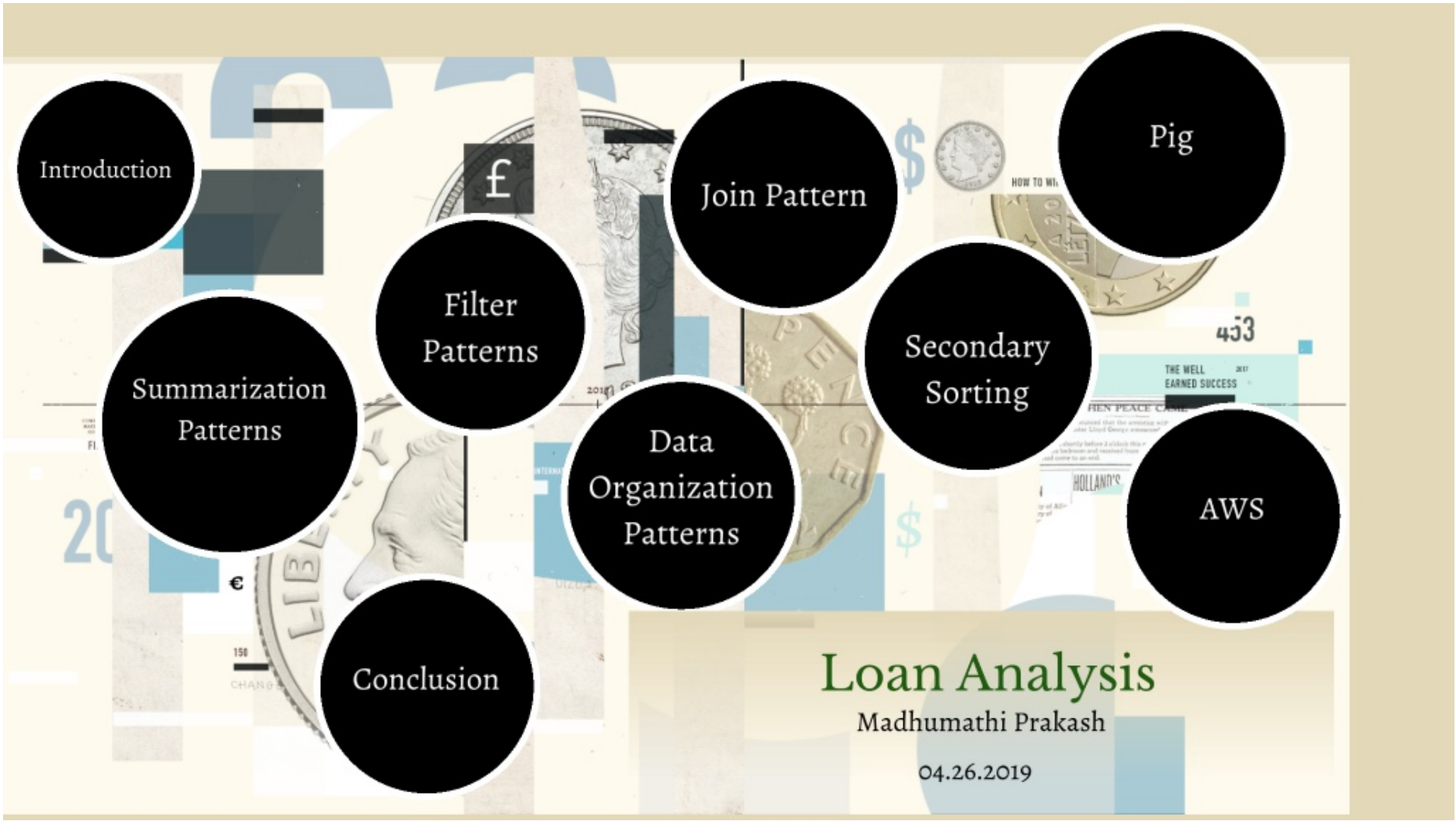
The other csv contained data on what the grade means and the credit score as well as the percent of people who usually have the grade

Driver: have 2 mapper classes

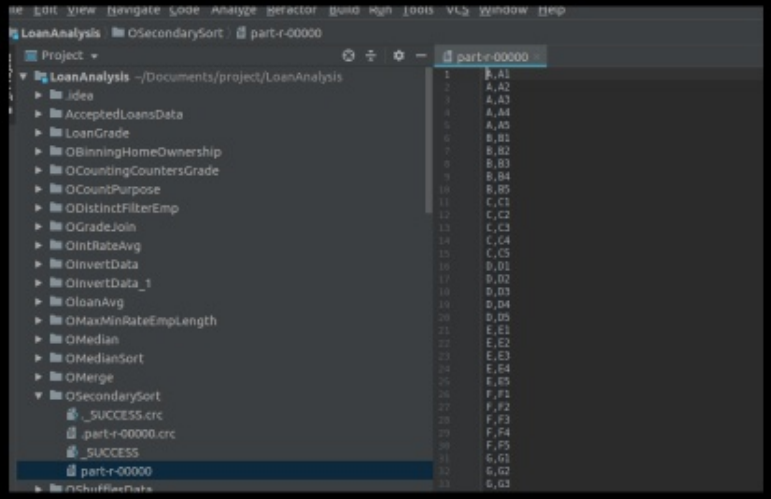
```
MultipleInputs.addInputPath(job, new Path(args[0]),TextInputFormat.class, DistinctMapper.class);
MultipleInputs.addInputPath(job,new Path(args[1]), TextInputFormat.class, CSVMapper.class);
```

```
public void executeJoin(Context context) throws IOException, InterruptedException {

    if(joinType.equalsIgnoreCase( anotherString: "inner")){
        if(!r.isEmpty() && !m.isEmpty()){
            for(Text ra: r){
                for(Text mo: m){
                    context.write(ra,mo);
                }
            }
        }
    }
```

Reducer has the inner join


Introduction
Filter Patterns
Join Pattern
Pig
Summarization Patterns
Secondary Sorting
Data Organization Patterns
AWS
Conclusion
Loan Analysis
Madhumathi Prakash
04.26.2019

Sorted the Grade and Subgrade using Secondary Sorting Technique




```
job.setGroupingComparatorClass(GroupComparator.class);
job.setSortComparatorClass(SecodarySortComparator.class);
job.setPartitionerClass(KeyPartition.class);
```

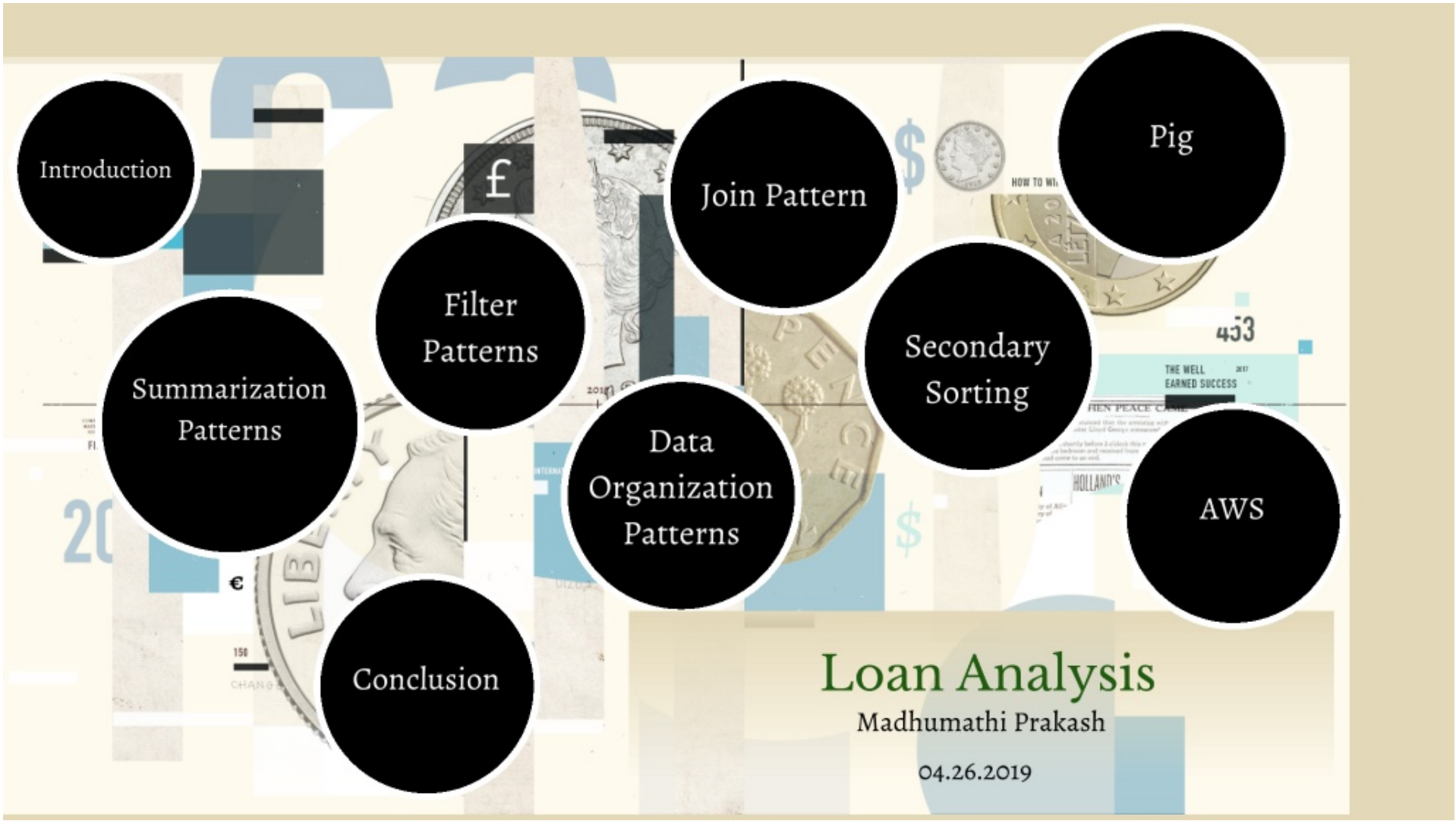
```
job.setOutputKeyClass(CompositeKeyWritable.class);
job.setOutputValueClass(NullWritable.class);
```

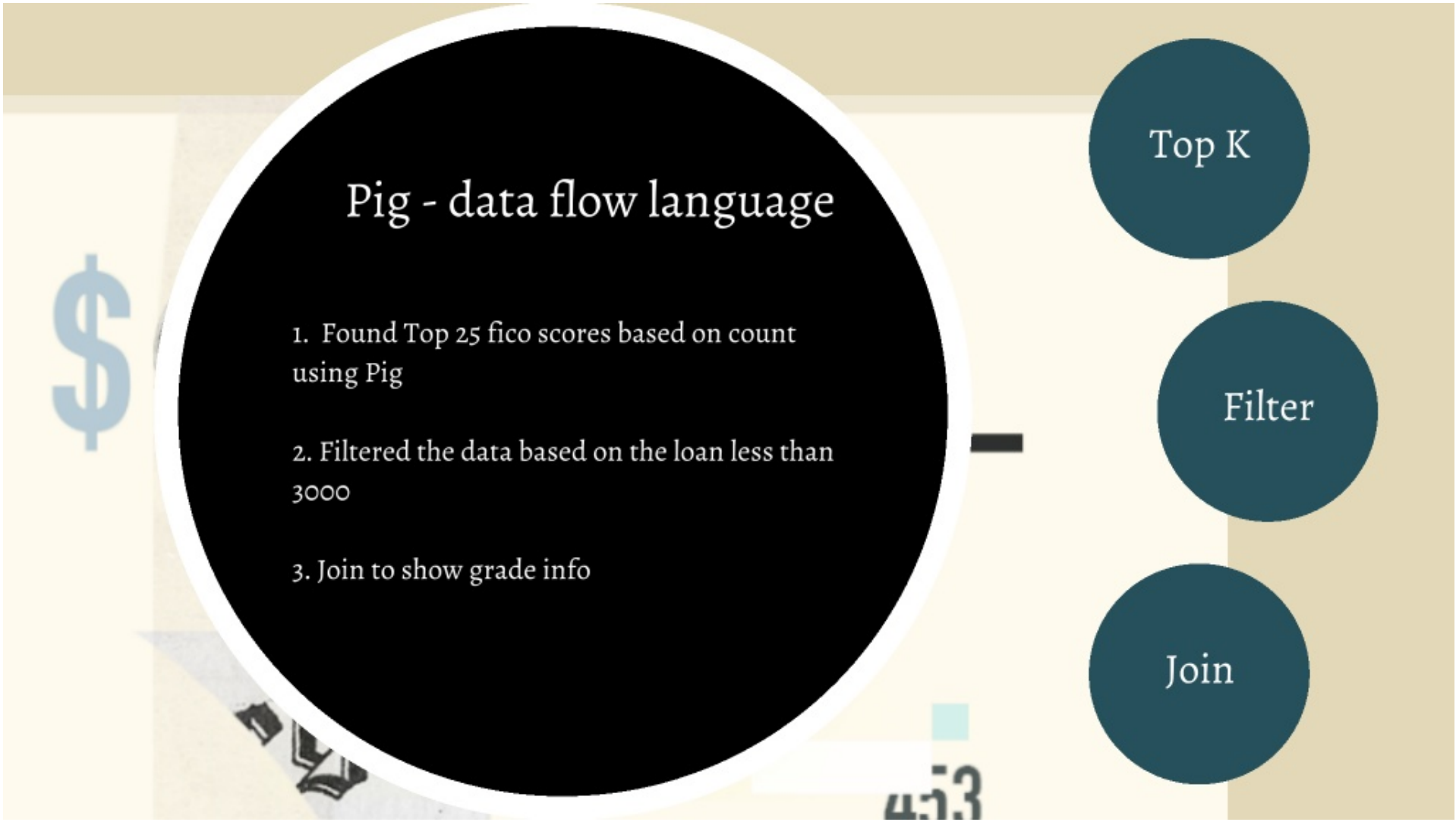
Driver: group comparator, sec sort comparator, composite key partitionor
Sec sort has compositKey class that implements WritableComparable

Group comparator and the sec sort comparator extend WritableComparable

```
protected GroupComparator() { super(CompositeKeyWritable.class, createInstances: true); }

@Override
public int compare(Object a, Object b) {

    CompositeKeyWritable ckw1 = (CompositeKeyWritable)a;
    CompositeKeyWritable ckw2 = (CompositeKeyWritable)b;

    return ckw1.getGrade().compareTo(ckw2.getGrade());
}
```

```
@Override
public int getPartition(CompositeKeyWritable key, NullWritable value, int numPartitions) {

    return key.getGrade().hashCode()%numPartitions;

}
```


Introduction
Filter Patterns
Join Pattern
Pig
Summarization Patterns
Secondary Sorting
Data Organization Patterns
AWS
Conclusion
Loan Analysis
Madhumathi Prakash
04.26.2019


Pig - data flow language
1. Found Top 25 fico scores based on count using Pig
2. Filtered the data based on the loan less than 3000
3. Join to show grade info
Top K
Filter
Join

Top 25 fico scores based on count using Pig

Open ▾ topk.pig ~/Documents/big_data/pig-0.17.0/bin Save

```
users = LOAD 'LoanStats3a_securev1.csv' using PigStorage(',');
grouped = GROUP users BY $24;
summed = FOREACH grouped GENERATE group, COUNT(users) AS cntd;
sorted = ORDER summed BY cntd DESC;
top25 = LIMIT sorted 25;
STORE top25 into 'topk_ficoscore_pig.csv';
```

Open ▾ topk.pi

```
689     2310
704     2267
684     2228
699     2202
694     2196
679     1994
709     1970
724     1949
719     1891
729     1891
674     1854
669     1821
714     1771
734     1670
739     1653
664     1584
744     1490
749     1319
754     1294
759     1100
764     984
774     799
769     792
779     654
784     573
```

Filtered the data based on the loan less than 3000

SmallLoan.pig
~/Documents/big_data/pig-0.17.0/bin

```
users = LOAD 'LoanStats3a_securev1.csv' using PigStorage(',');
s1 = FILTER users BY $2 < 3000;
s2 = FOREACH s1 GENERATE $0,$1,$2,$3,$4,$5,$6,$7,$8,$9,$10,$11,$12,$13,$14,'T';
STORE s2 into 'SmallLoanAmount';
```

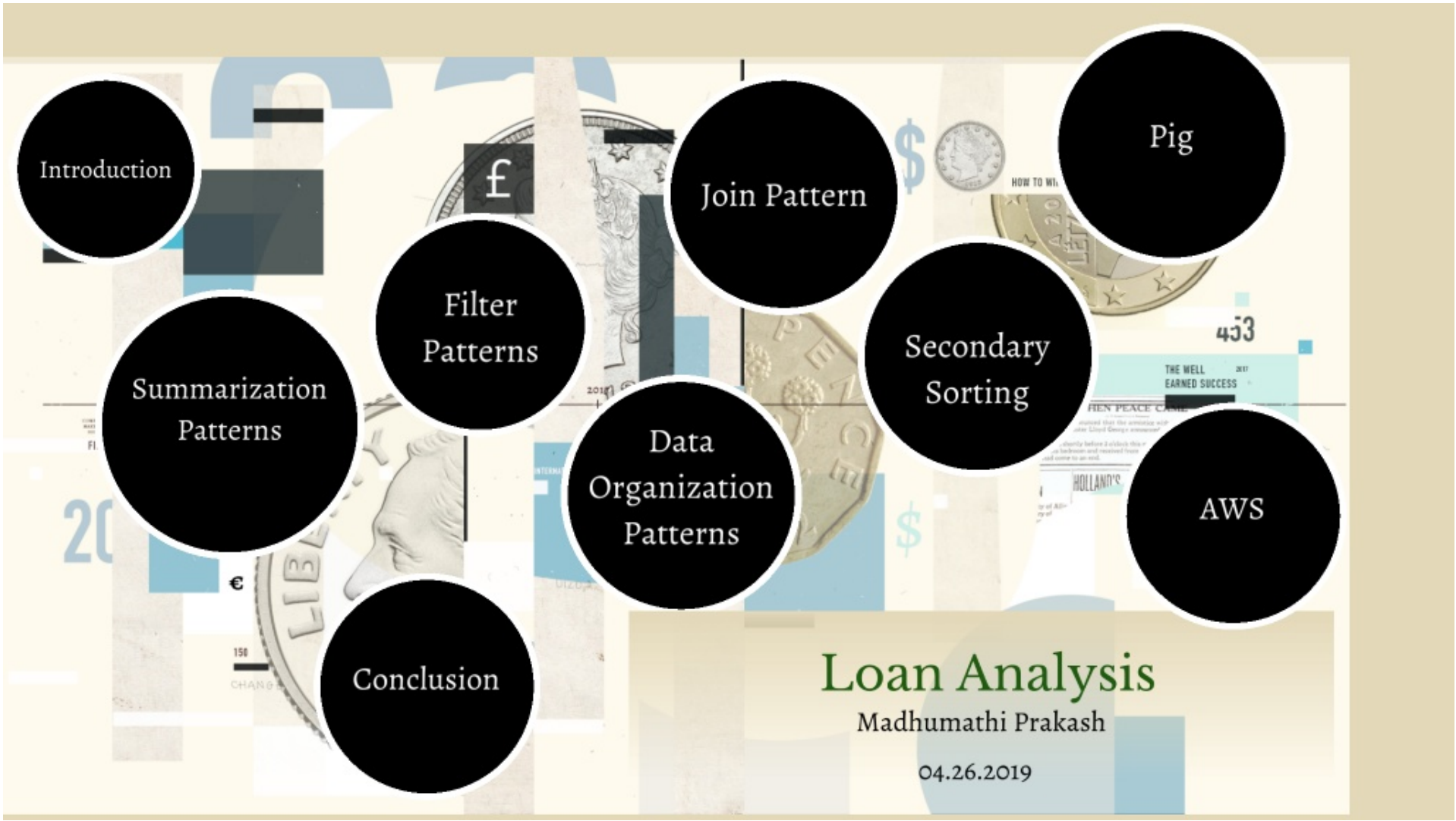
part-m-00000
~/Documents/big_data/pig-0.17.0/bin/SmallLoanAmount

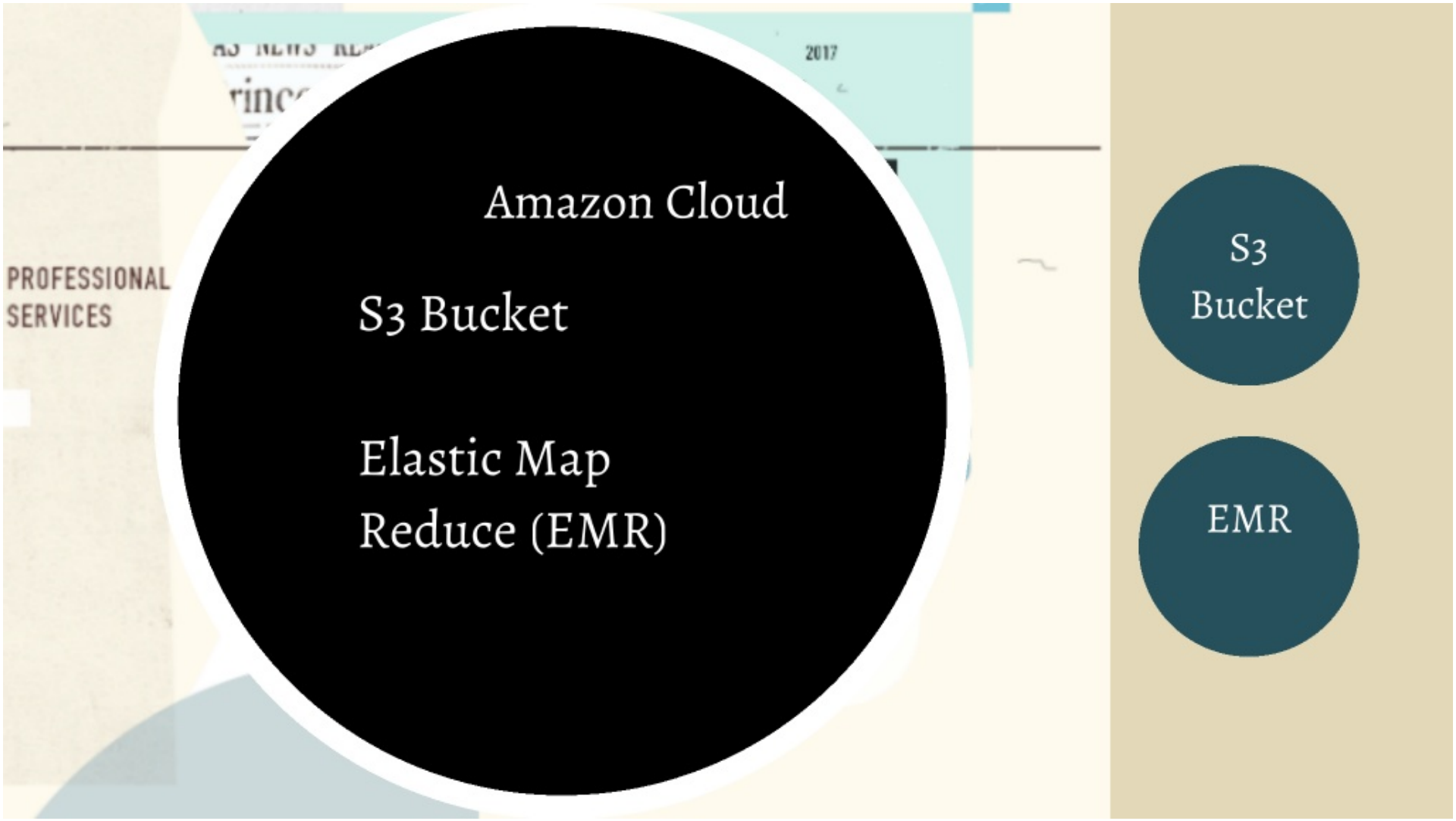
```
1077430         2500    2500    2500     60 months      15.27%  59.83   C       C4      < 1
year    RENT    30000   Source Verified Dec-11  T
1077175         2400    2400    2400     36 months      15.96%  84.33   C       C5      10+
years   RENT    12252   Not Verified    Dec-11  T
1069759         1000    1000    1000     36 months      16.29%  35.31   D       D1      < 1
year    RENT    28000   Not Verified    Dec-11  T
1068292         2100    2100    2100     36 months      12.42%  70.18   B       B4      10+
years   MORTGAGE        30000   Not Verified    Dec-11  T
1068180         2500    2500    2500     36 months      14.27%  85.78   C       C2      7 years
RENT    49500   Source Verified Dec-11  T
1067644         2500    2500    2500     36 months      7.51%   77.78   A       A3      n/a
OWN     12000   Source Verified Dec-11  T
1067601         2500    2500    2500     36 months      13.49%  84.83   C       C1      4 years
RENT    27200   Not Verified    Dec-11  T
1066836         2000    2000    2000     36 months      15.27%  69.6    C       C4      2 years
RENT    41000   Verified        Dec-11  T
1066617         1500    1500    1500     36 months      17.27%  53.69   D       D3      10+
years   MORTGAGE        144000  Verified        Dec-11  T
1066598         2500    2500    2500     36 months      11.71%  82.69   B       B3      7 years
MORTGAGE        29000   Verified        Dec-11  T
1065382         1400    1400    1400     36 months      7.90%   43.81   A       A4      1 year
RENT    28800   Not Verified    Dec-11  T
1066081         1800    1800    1800     36 months      6.62%   55.27   A       A2      10+
years   RENT    18000   Not Verified    Dec-11  T
1064453         2000    2000    2000     36 months      7.90%   62.59   A       A4      4 years
RENT    39600   Not Verified    Dec-11  T
1064251         1500    1500    1500     36 months      18.64%  54.72   E       E1      < 1
year    RENT    17000   Source Verified Dec-11  T
1064366         1000    1000    1000     36 months      14.65%  34.5    C       C3      n/a
RENT    18408   Not Verified    Dec-11  T
1064006         2625    2625    2625     36 months      11.71%  86.83   B       B3      10+
years   MORTGAGE        52800   Not Verified    Dec-11  T
1063448         2200    2200    2200     36 months      15.27%  76.56   C       C4      10+
years   OWN     48000   Not Verified    Dec-11  T
1062897         1700    1700    1700     36 months      7.90%   53.2    A       A4      n/a
MORTGAGE        37800   Not Verified    Dec-11  T
1062510         2000    2000    2000     36 months      9.91%   64.45   B       B1      1 year
```

Join to display grade with more info

```
grade = LOAD 'grade' using PigStorage(',') as (grd);
detail = LOAD 'gradedetails.csv' using PigStorage(',') as (id,desc,name,creditscore);
joinop = JOIN grade BY grd, detail BY id;
out = FOREACH joinop GENERATE $0,$2,$3,$4;
STORE summed out 'gradeoutput';
```

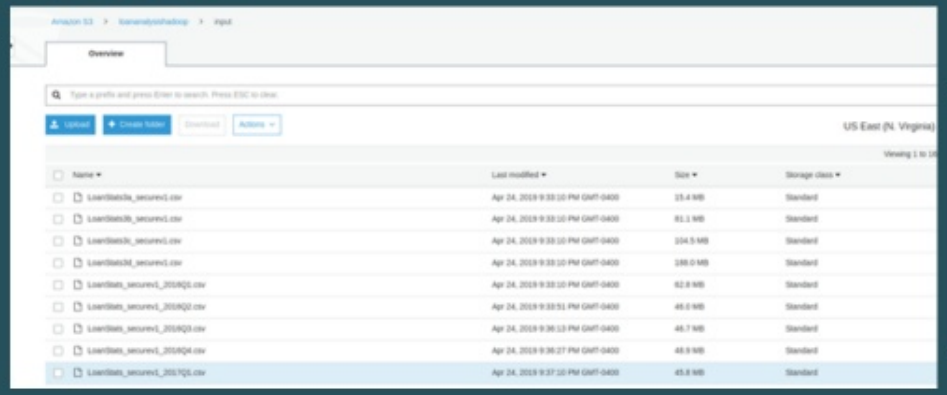
```
A       Excellent,750-850,16.7%
B       Good,700-749,34.1%
C       Average,650-699,17.3%
D       Below Average,550-649,12.6%
E       Poor,310-349,25.3%
F       Bad,300,5%
G       Too Bad,250,1%
```


Introduction
Filter Patterns
Join Pattern
Pig
Summarization Patterns
Secondary Sorting
Data Organization Patterns
AWS
Conclusion
Loan Analysis
Madhumathi Prakash
04.26.2019

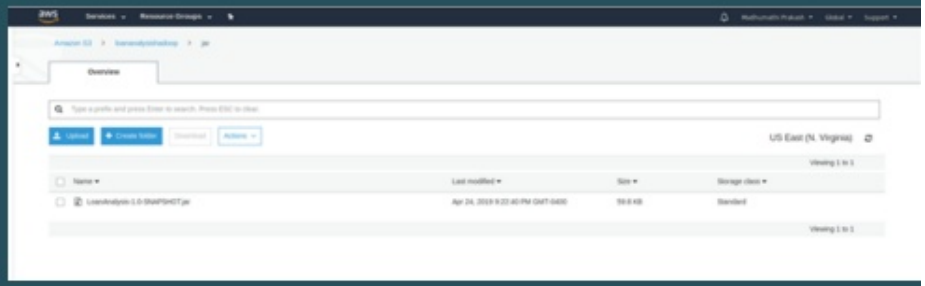

Amazon Cloud
S3 Bucket
Elastic Map Reduce (EMR)
S3 Bucket
EMR

S3 Bucket

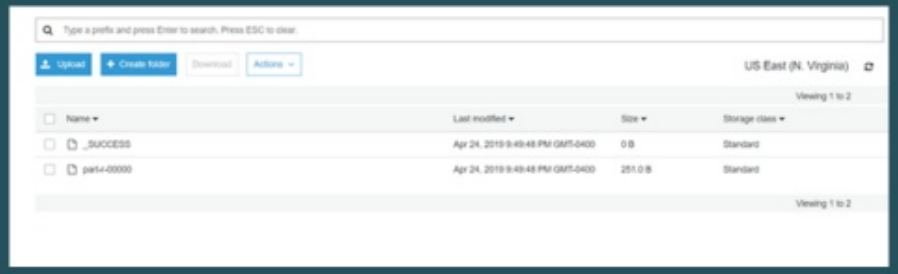
input folder



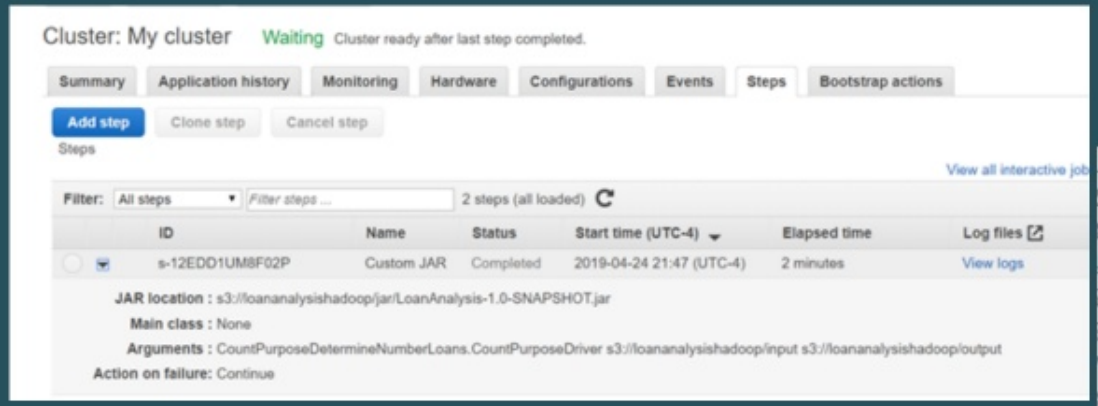
JAR File



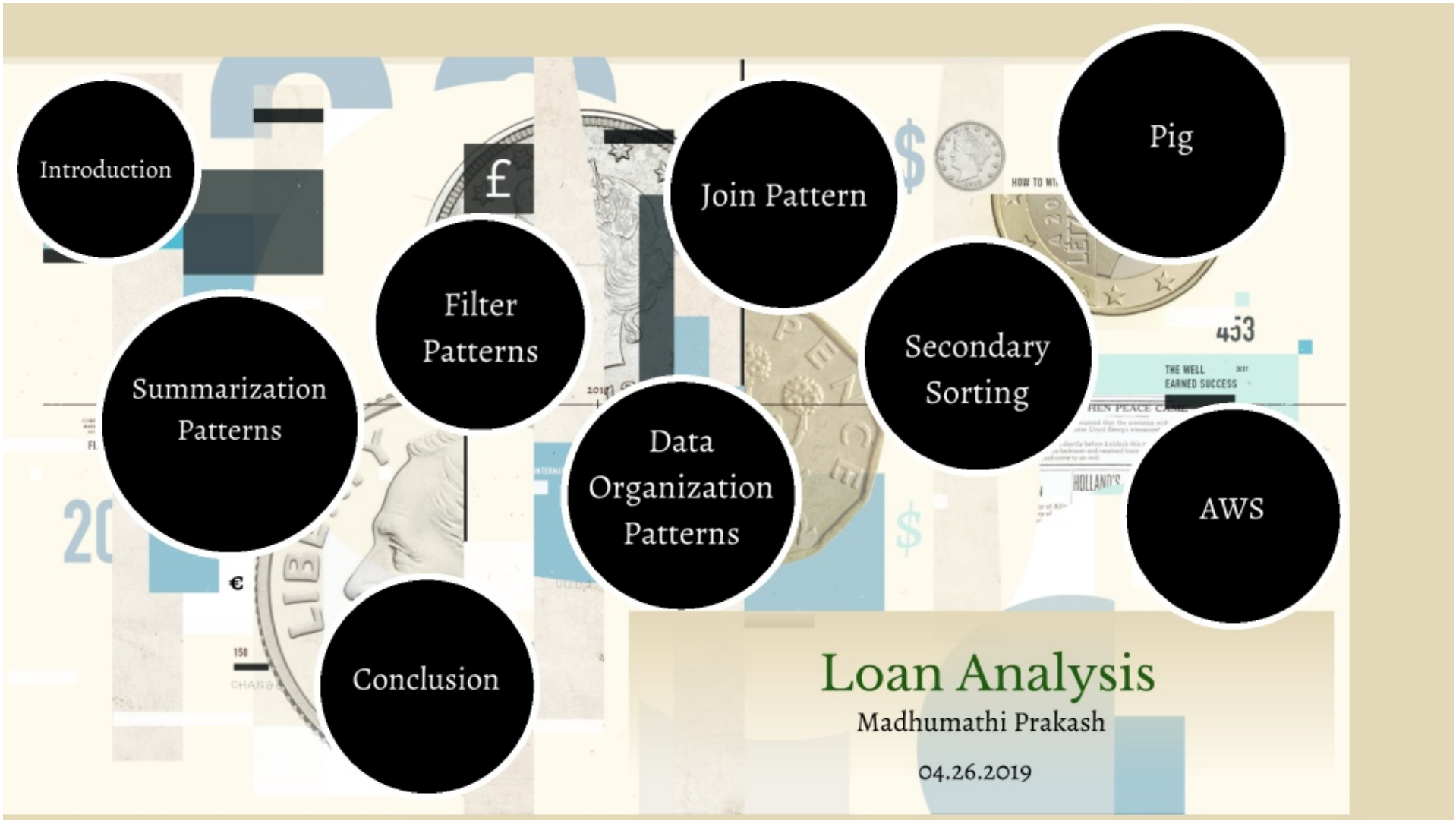
Output Folder

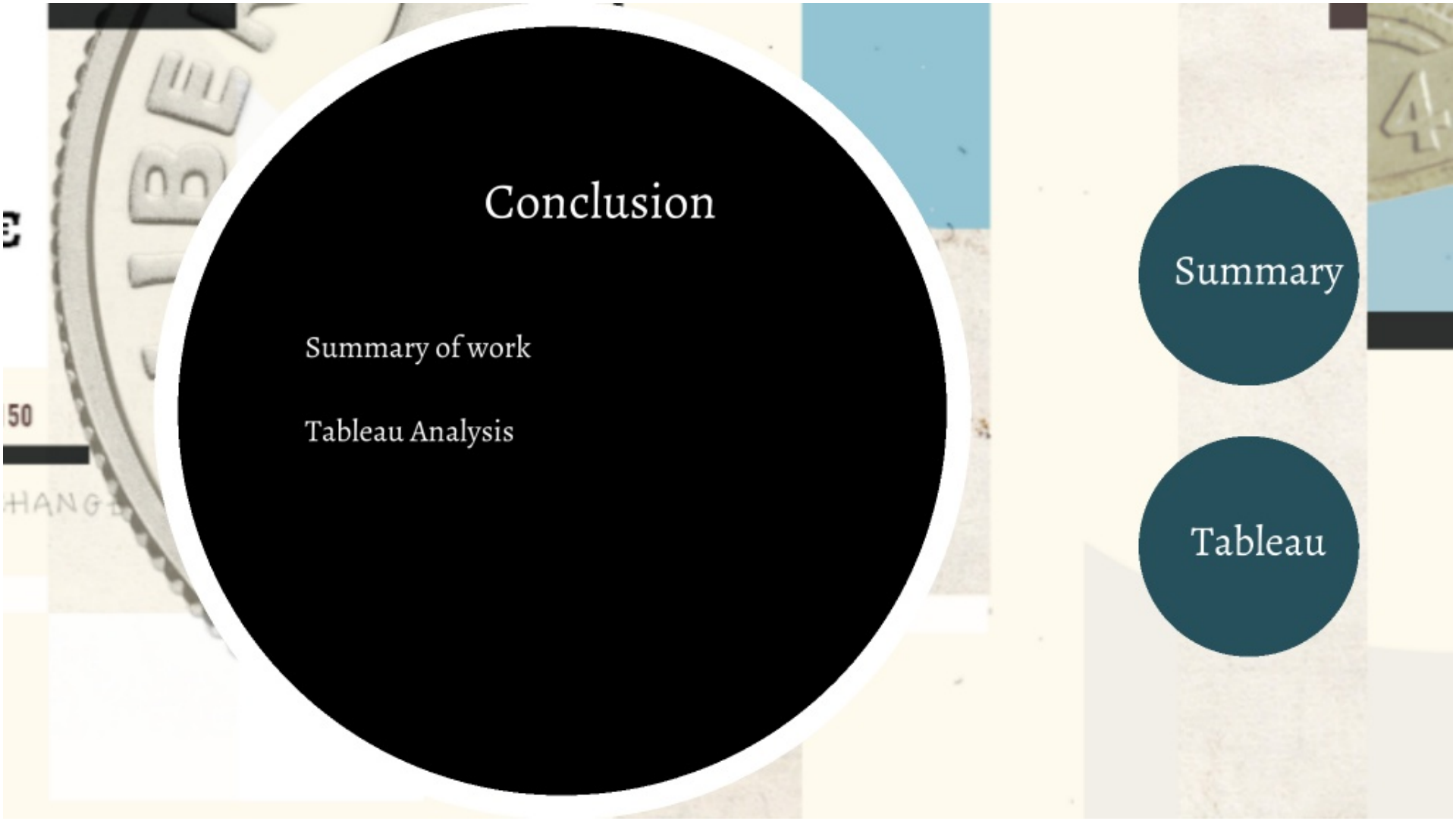


EMR



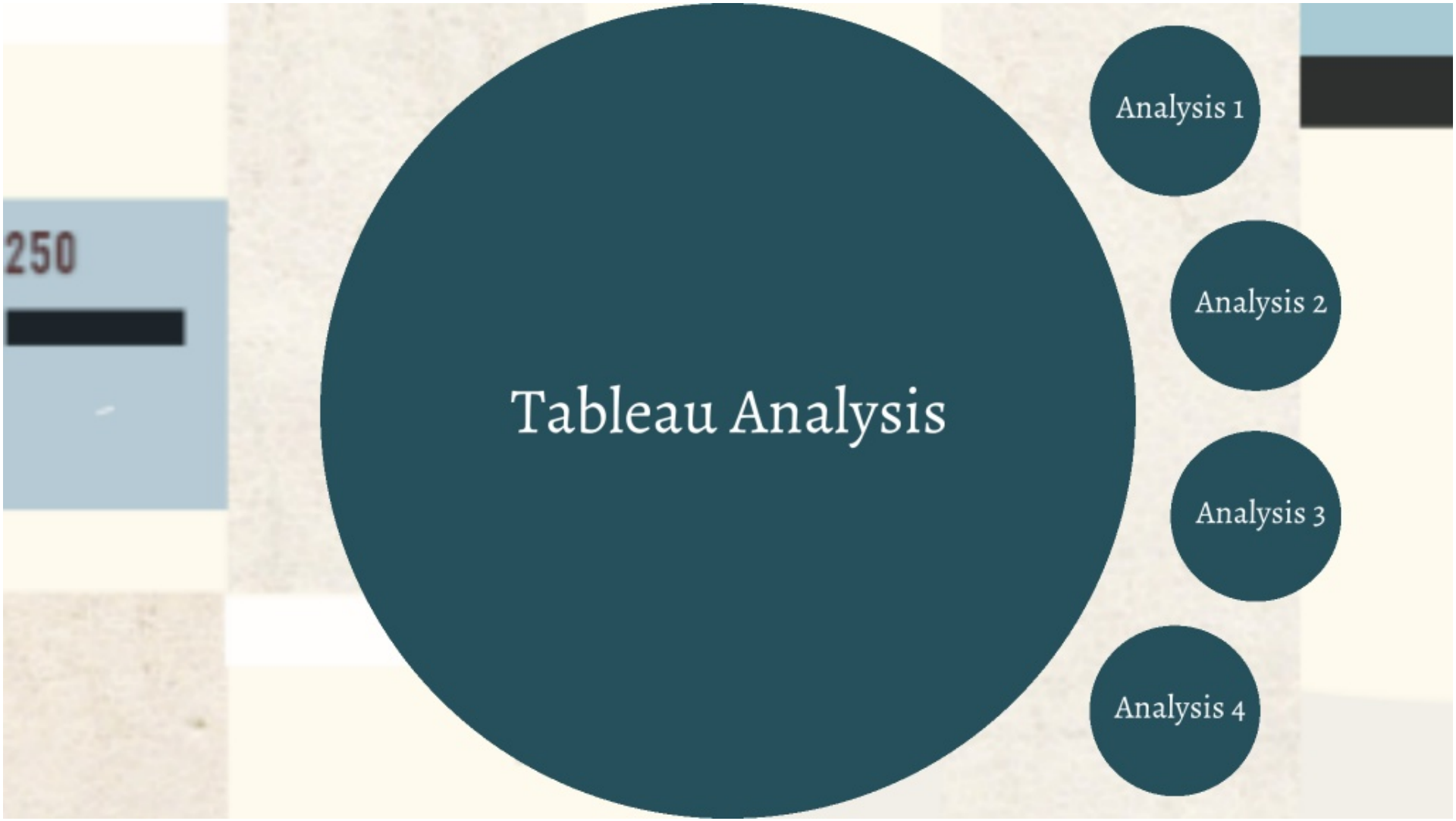

```
car 24013
credit_card 516971
debt_consolidation  1277877
educational 424
home_improvement    150457
house   14136
major_purchase  50445
medical 27488
moving  15403
other   139440
purpose 16
renewable_energy    1445
small_business  24689
vacation    15525
wedding 2355
```


Introduction
Filter Patterns
Join Pattern
Pig
Summarization Patterns
Secondary Sorting
Data Organization Patterns
AWS
Conclusion
Loan Analysis
Madhumathi Prakash
04.26.2019

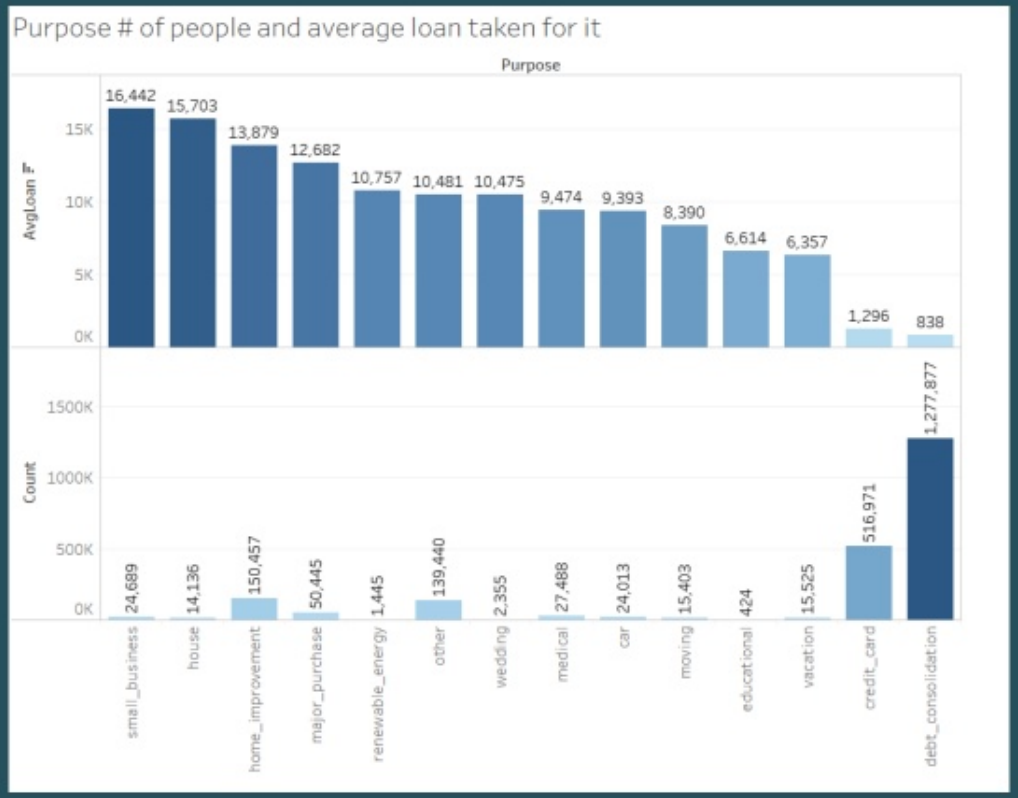

Conclusion
Summary of work
Tableau Analysis
Summary
Tableau

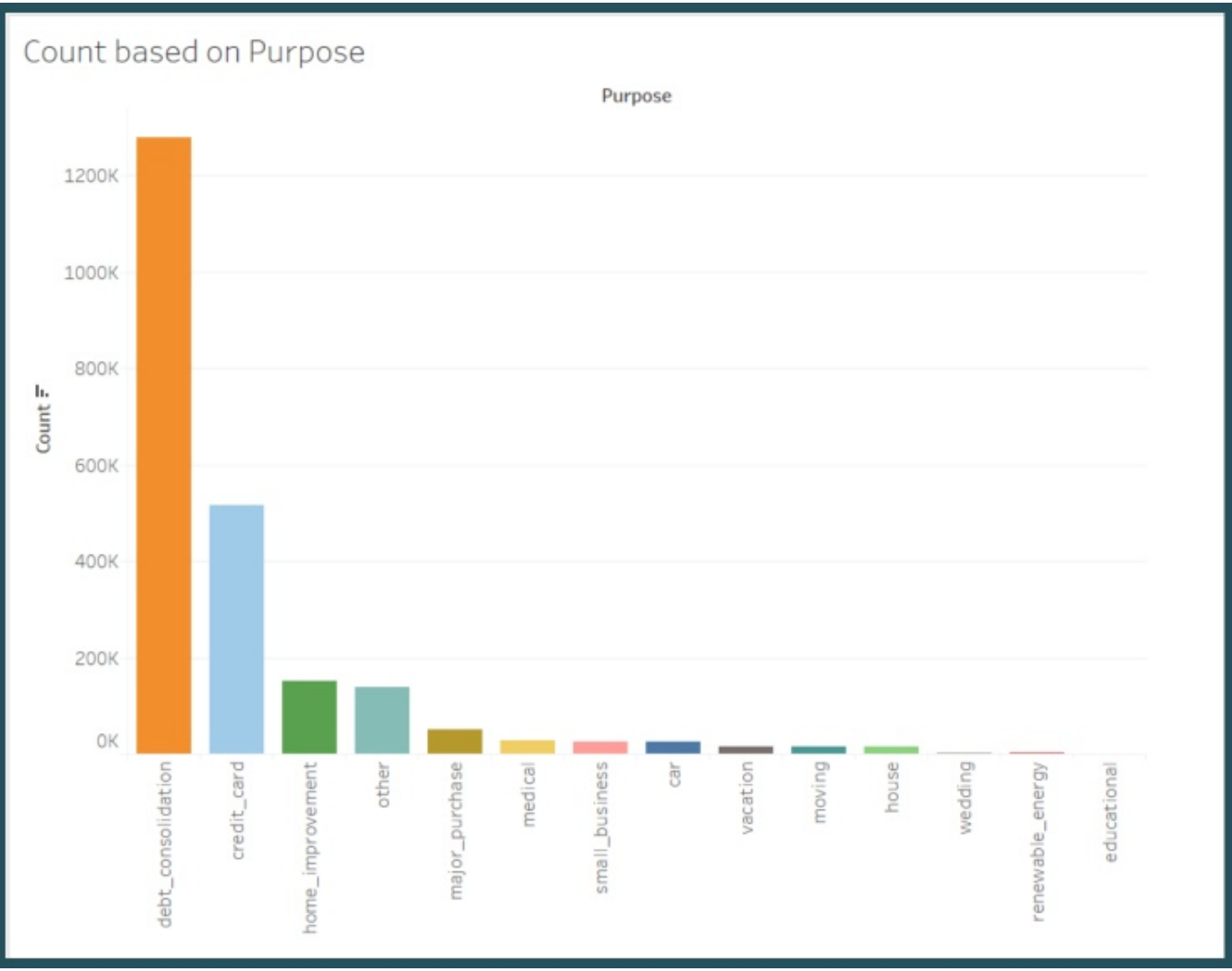
Summary Of Work Done

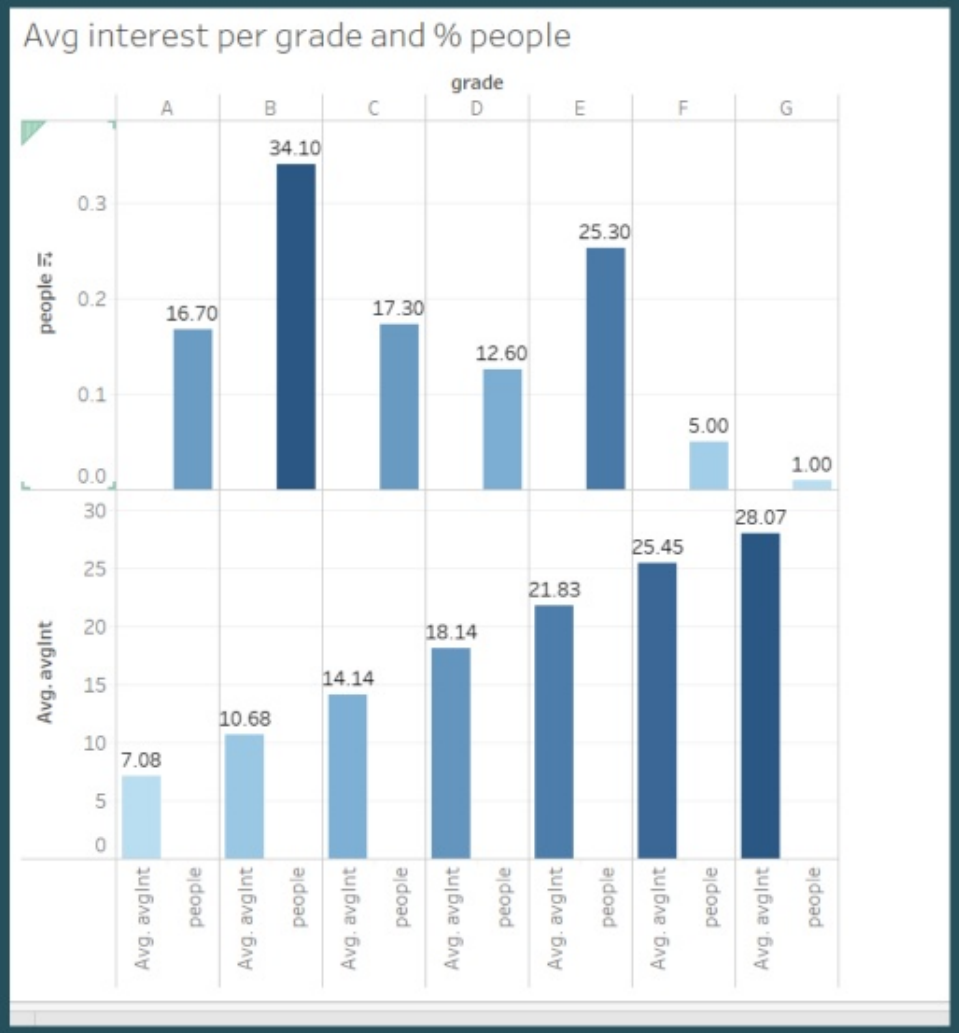
Overview	Work done
Summarization patterns	Mand and min
	Nline
	Counting
	Average
	Median
	Mem concious Median
	Custome Combiner
	Inverted index
	Counting with counters
Filter Patterns	
	Bloom filtering
	Top K
	Distinct
Data org Patterns	
	Binning
	Shuffling
Joins	inner joins
Secondary Sort	WritableComparable
Chaining	
Pig	
	Top K
	Filtering
	Join
AWS Cloud	

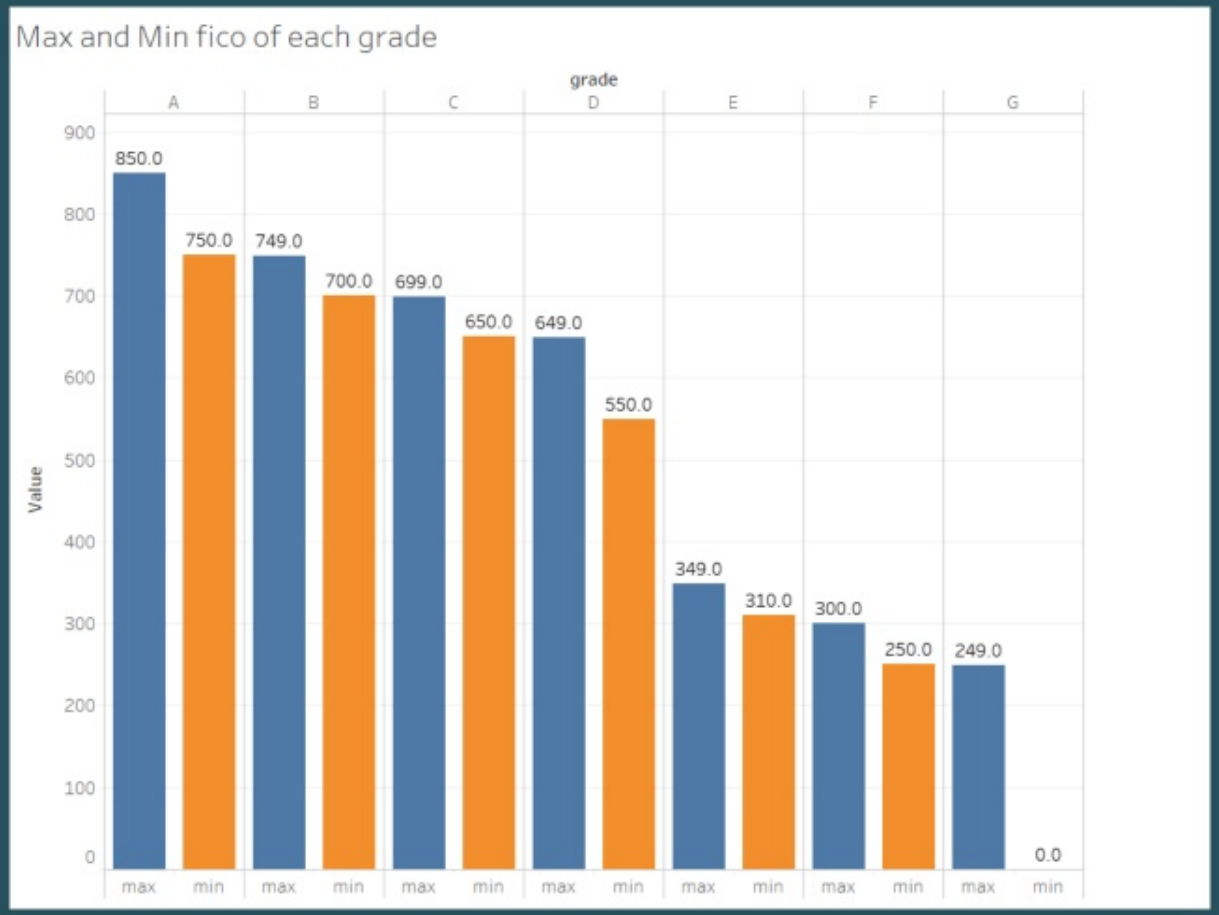

250
Tableau Analysis
Analysis 1
Analysis 2
Analysis 3
Analysis 4

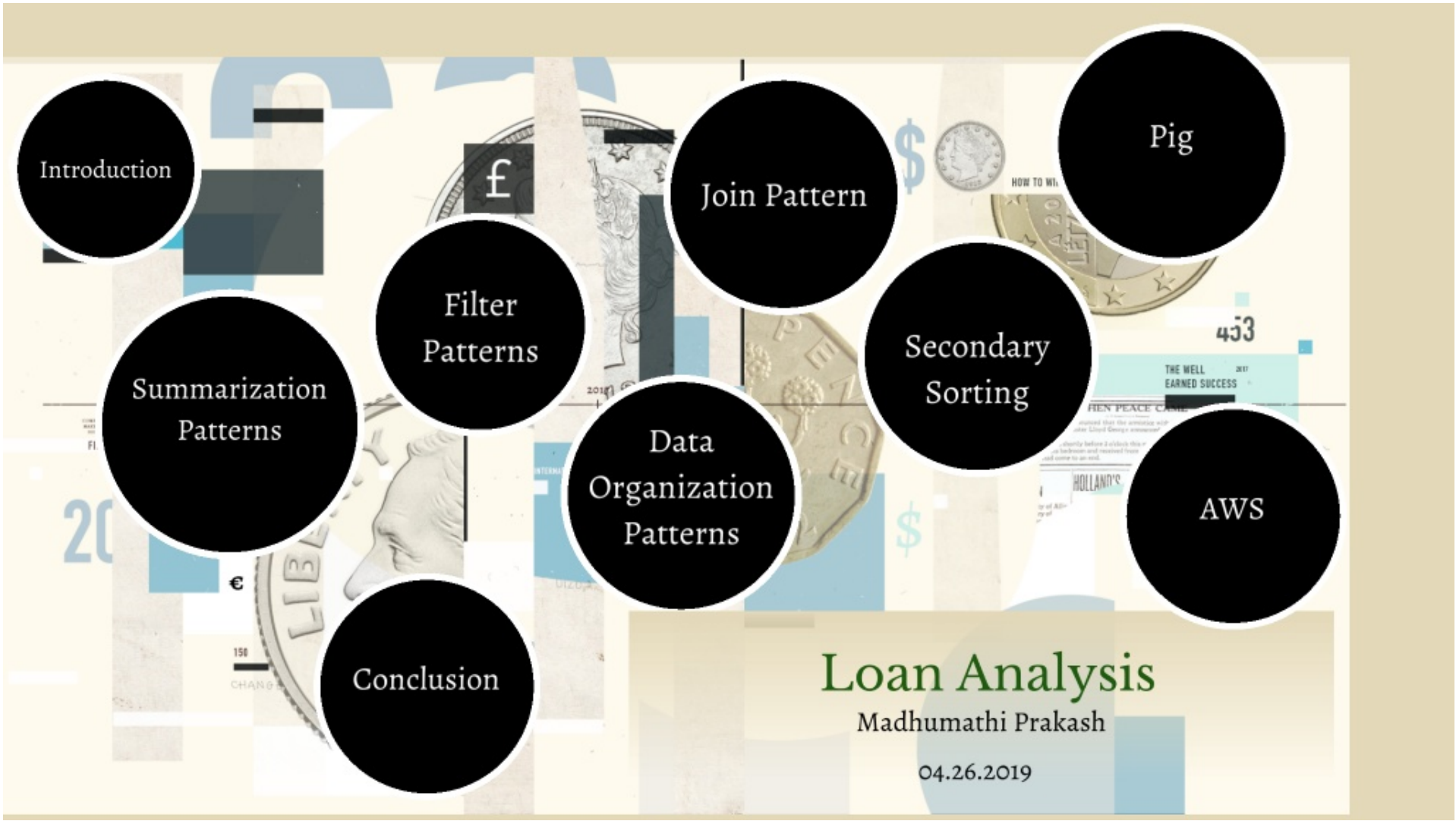
Purpose # of people and average loan taken for it




Count based on Purpose
Purpose
Count
1200K
1000K
800K
600K
400K
200K
0K
debt_consolidation
credit_card
home_improvement
other
major_purchase
medical
small_business
car
vacation
moving
house
wedding
renewable_energy
educational


Avg interest per grade and % people
grade
A
B
C
D
E
F
G
people
0.3
0.2
0.1
0.0
16.70
34.10
17.30
12.60
25.30
5.00
1.00
Avg. avgInt
30
25
20
15
10
5
0
7.08
10.68
14.14
18.14
21.83
25.45
28.07
Avg. avgInt
people


Max and Min fico of each grade
grade
A
B
C
D
E
F
G
Value
900
800
700
600
500
400
300
200
100
0
850.0
750.0
749.0
700.0
699.0
650.0
649.0
550.0
349.0
310.0
300.0
250.0
249.0
0.0
max
min


Introduction
Filter Patterns
Join Pattern
Pig
Summarization Patterns
Secondary Sorting
Data Organization Patterns
AWS
Conclusion
Loan Analysis
Madhumathi Prakash
04.26.2019

Appendix

AverageInterestRateByGrade

```
package AverageInterestRateByGrade;


import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import java.util.*;


import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;


public class AverageInterestRateByGrade {
   public static class MapLoan extends Mapper<LongWritable, Text, Text, LoanAveragingPair> {
      private Text outText = new Text();
      private Map<String,LoanAveragingPair> pairMap = new HashMap<String,LoanAveragingPair>();
      private LoanAveragingPair pair = new LoanAveragingPair();


      public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
         String line = value.toString();
         String[] columnValues = line.split(",");
```

```
        if (columnValues.length > 6) {
            if (!columnValues[0].contains("id")) {
                outText.set(columnValues[8]);
                Float LoanTime = Float.parseFloat(columnValues[6].replace("%", ""));
                LoanAveragingPair pair = pairMap.get(columnValues[8]);
                if (pair == null) {
                    pair = new LoanAveragingPair();
                    pairMap.put(columnValues[8], pair);

                }
                Float Loan = pair.getLoan().get() + LoanTime;
                int count = pair.getCount().get() + 1;
                pair.set(Loan, count);
            }
        }
    }

    @Override
    protected void cleanup(Context context) throws IOException, InterruptedException {
        Set<String> keys = pairMap.keySet();
        Text keyText = new Text();
        for (String key : keys) {
            keyText.set(key);
            context.write(keyText,pairMap.get(key));
        }
    }
}

public static class Reduce extends Reducer<Text, LoanAveragingPair, Text, FloatWritable> {
```

```
    public void reduce(Text key, Iterable<LoanAveragingPair> values
        , Context context)
        throws IOException, InterruptedException {
      float sum = 0;
      int count = 0;
      for (LoanAveragingPair val : values) {
        sum += val.getLoan().get();
        count += val.getCount().get();
      }
      context.write(key, new FloatWritable(sum/count));
    }
  }

  public static class LoanAveragingPair implements Writable, WritableComparable<LoanAveragingPair>
{
    private FloatWritable Loan;
    private IntWritable count;

    public LoanAveragingPair() {
      set(new FloatWritable(0), new IntWritable(0));
    }

    public LoanAveragingPair(Float Loan, int count) {
      set(new FloatWritable(Loan), new IntWritable(count));
    }

    public void write(DataOutput out) throws IOException {
      Loan.write(out);
```

```
    count.write(out);
}


public void readFields(DataInput in) throws IOException {
    Loan.readFields(in);
    count.readFields(in);
}


public int compareTo(LoanAveragingPair other) {
    int compareVal = this.Loan.compareTo(other.getLoan());
    if (compareVal != 0) {
        return compareVal;
    }
    return this.count.compareTo(other.getCount());
}
public void set(Float Loan, int count){
    this.Loan.set(Loan);
    this.count.set(count);
}

public void set(FloatWritable Loan, IntWritable count) {
    this.Loan = Loan;
    this.count = count;
}


public static LoanAveragingPair read(DataInput in) throws IOException {
```

```
    LoanAveragingPair averagingPair = new LoanAveragingPair();
    averagingPair.readFields(in);
    return averagingPair;
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    LoanAveragingPair that = (LoanAveragingPair) o;

    if (!count.equals(that.count)) return false;
    if (!Loan.equals(that.Loan)) return false;

    return true;
}

@Override
public int hashCode() {
    int result = Loan.hashCode();
    result = 163 * result + count.hashCode();
    return result;
}

@Override
public String toString() {
    return "LoanAveragingPair{" +
            "Loan=" + Loan +
            ", count=" + count +
```

```
            '}';
    }

    public FloatWritable getLoan() {
        return Loan;
    }

    public IntWritable getCount() {
        return count;
    }
}

public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

    Job job = new Job(conf, "int rate count");
    job.setJarByClass(AverageInterestRateByGrade.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(LoanAveragingPair.class);

    job.setMapperClass(MapLoan.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
```

```
        job.waitForCompletion(true);
    }
}
```

AverageLoanAmtByPurpose

```
package AverageLoanAmtByPurpose;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;


public class AverageLoanAmtByPurpose {
    public static class MapLoan extends Mapper<LongWritable, Text, Text, LoanAveragingPair> {
        private Text outText = new Text();
        private Map<String,LoanAveragingPair> pairMap = new HashMap<String,LoanAveragingPair>();
        private LoanAveragingPair pair = new LoanAveragingPair();
```

```
    public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
        String line = value.toString();
        String[] columnValues = line.split(",");
        // System.out.println(columnValues[0] + " " + columnValues[2]);
        if (columnValues.length > 2) {
            if (!columnValues[2].contains("loan")) {
                outText.set(columnValues[17]);
                int LoanTime = Integer.parseInt(columnValues[2]);
                LoanAveragingPair pair = pairMap.get(columnValues[17]);
                if (pair == null) {
                    pair = new LoanAveragingPair();
                    pairMap.put(columnValues[17], pair);

                }
                int Loan = pair.getLoan().get() + LoanTime;
                int count = pair.getCount().get() + 1;
                pair.set(Loan, count);
            }
    }
    }

    @Override
    protected void cleanup(Context context) throws IOException, InterruptedException {
        Set<String> keys = pairMap.keySet();
        Text keyText = new Text();
        for (String key : keys) {
            keyText.set(key);
            context.write(keyText,pairMap.get(key));
```

```
        }
      }
    }

    public static class Reduce extends Reducer<Text, LoanAveragingPair, Text, IntWritable> {
      public void reduce(Text key, Iterable<LoanAveragingPair> values
          , Context context)
          throws IOException, InterruptedException {
        int sum = 0;
        int count = 0;
        for (LoanAveragingPair val : values) {
          sum += val.getLoan().get();
          count += val.getCount().get();
        }
        context.write(key, new IntWritable(sum/count));
      }
    }

    public static class LoanAveragingPair implements Writable, WritableComparable<LoanAveragingPair>
{
      private IntWritable Loan;
      private IntWritable count;

      public LoanAveragingPair() {
        set(new IntWritable(0), new IntWritable(0));
      }

      public LoanAveragingPair(int Loan, int count) {
        set(new IntWritable(Loan), new IntWritable(count));
```

```
}

public void write(DataOutput out) throws IOException {
   Loan.write(out);
   count.write(out);
}

public void readFields(DataInput in) throws IOException {
   Loan.readFields(in);
   count.readFields(in);
}

public int compareTo(LoanAveragingPair other) {
   int compareVal = this.Loan.compareTo(other.getLoan());
   if (compareVal != 0) {
      return compareVal;
   }
   return this.count.compareTo(other.getCount());
}
public void set(int Loan, int count){
   this.Loan.set(Loan);
   this.count.set(count);
}

public void set(IntWritable Loan, IntWritable count) {
   this.Loan = Loan;
```

```
    this.count = count;
}


public static LoanAveragingPair read(DataInput in) throws IOException {
    LoanAveragingPair averagingPair = new LoanAveragingPair();
    averagingPair.readFields(in);
    return averagingPair;
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    LoanAveragingPair that = (LoanAveragingPair) o;

    if (!count.equals(that.count)) return false;
    if (!Loan.equals(that.Loan)) return false;

    return true;
}

@Override
public int hashCode() {
    int result = Loan.hashCode();
    result = 163 * result + count.hashCode();
    return result;
}
```

```
    @Override
    public String toString() {
        return "LoanAveragingPair{" +
                "Loan=" + Loan +
                ", count=" + count +
                '}';
    }

    public IntWritable getLoan() {
        return Loan;
    }

    public IntWritable getCount() {
        return count;
    }
}

public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

    Job job = new Job(conf, "loan count");
    job.setJarByClass(AverageLoanAmtByPurpose.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(LoanAveragingPair.class);

    job.setMapperClass(MapLoan.class);
    job.setReducerClass(Reduce.class);
```

```
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.waitForCompletion(true);
    }
}
```

BinningHomeOwnership

```
package BinningHomeOwnership;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.MultipleOutputs;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import java.io.IOException;

public class BinningHomeOwnership {
    public static class TMapper extends Mapper<Object, Text, Text, NullWritable>
```

```
{

   private MultipleOutputs<Text, NullWritable> mos=null;

   @Override
   protected void setup(Context context) throws IOException, InterruptedException
   {
      mos = new MultipleOutputs(context);
   }

   @Override
   public void map(Object key, Text value, Context context)
         throws IOException, InterruptedException
   {

      String line = value.toString();
      String row[] = value.toString().split(",");

      if(row.length > 15 && !line.contains("home_ownership") ){
         String cancellationCode = row[11];

         // System.out.println(cancellationCode);
         if(cancellationCode.contains("MORTGAGE"))
            mos.write("bins", value, NullWritable.get(),"loans-mortgage-home");
         if(cancellationCode.contains("OWN"))
            mos.write("bins", value, NullWritable.get(),"loans-own-home");
         if(cancellationCode.contains("RENT"))
            mos.write("bins", value, NullWritable.get(),"loans-rent-home");
```

```
        }


    }


    @Override
    protected void cleanup(Context context) throws IOException, InterruptedException
    {
        mos.close();
    }


}
```

```
  public static void main(String[] args) throws IOException, InterruptedException,
ClassNotFoundException
  {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "Cancellation Binning ");
    job.setJarByClass(BinningHomeOwnership.class);
    job.setMapperClass(TMapper.class);



    MultipleOutputs.addNamedOutput(job, "bins", TextOutputFormat.class, Text.class,
IntWritable.class);
```

```
      MultipleOutputs.setCountersEnabled(job, true);
      job.setNumReduceTasks(0);

      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(NullWritable.class);
      FileInputFormat.addInputPath(job, new Path(args[0]));
      FileOutputFormat.setOutputPath(job, new Path(args[1]));
      System.exit(job.waitForCompletion(true) ? 0 : 1);
   }

}
```

CountingWithCountersGrade

```
package CountingWithCountersGrade;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Counter;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

public class Driver {
   public static void main( String[] args ) throws IOException, ClassNotFoundException,
InterruptedException
   {
      Job job = Job.getInstance();
```

```
job.setJarByClass(Driver.class);



FileInputFormat.addInputPath(job, new Path(args[0]));
Path outDir = new Path(args[1]);
FileOutputFormat.setOutputPath(job, outDir);

job.setMapperClass(MyMapper.class);




job.setOutputKeyClass(NullWritable.class);
job.setOutputValueClass(NullWritable.class);
;



int code = job.waitForCompletion(true)? 1 : 0;
if (code==1) {

   for(Counter counter: job.getCounters().getGroup("MonthCounter")) {
      //System.out.print("inside this");
      System.out.println("the counter "+counter.getDisplayName()+" is "+counter.getValue());
      // System.out.println("the counter values is "+counter.getValue());
   }

}
```

```
   }
}

package CountingWithCountersGrade;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.util.Arrays;
import java.util.HashSet;

public class MyMapper extends Mapper<LongWritable, Text, NullWritable,NullWritable> {
   @Override
   protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {

      String grades[]= {"A","B","C","D","E","F", "G"};
      HashSet<String> gradesHashSet= new HashSet<String>(Arrays.asList(grades));

      String input = value.toString();
      String splitString[] = input.split(",");
      if(splitString.length > 8) {
         String grade = splitString[8];
         //  System.out.print(month);

         if (gradesHashSet.contains(grade) && splitString[8] != "grade") {
```

```
            // System.out.print(month);
            context.getCounter("MonthCounter", grade).increment(1);
          } else {
            context.getCounter("MonthCounter", "unknown").increment(1);

          }
        }


    }
}
```

CountPurposeDetermineNumberLoans

```
package CountPurposeDetermineNumberLoans;

import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import java.io.IOException;


public class CountPurposeDriver {
```

```
   public static void main(String[] args) throws IOException, ClassNotFoundException,
InterruptedException {
      Job job = Job.getInstance();
      job.setJarByClass(CountPurposeDriver.class);

      job.setMapperClass(CountPurposeMapper.class);
      job.setCombinerClass(CountPurposeReducer.class);
      job.setReducerClass(CountPurposeReducer.class);

      job.setInputFormatClass(TextInputFormat.class);
      job.setOutputFormatClass(TextOutputFormat.class);

      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(IntWritable.class);

      job.setNumReduceTasks(1);

      FileInputFormat.addInputPath(job, new Path(args[0]));
      FileOutputFormat.setOutputPath(job, new Path(args[1]));

      FileSystem fs = FileSystem.get(job.getConfiguration());
/*
Path outDir = new Path(args[1]);
if(fs.exists(outDir)) {
fs.delete(outDir, true);
}
Submit the job, then poll for progress until the job is complete
*/
```

```
        job.waitForCompletion(true);
    }
}

package CountPurposeDetermineNumberLoans;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class CountPurposeMapper extends Mapper<LongWritable, Text,Text, IntWritable> {
private static final IntWritable one = new IntWritable(1);

@Override
protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text,
IntWritable>.Context context)
        throws IOException, InterruptedException {
        String line = value.toString();

        String[] tokens = line.split(",");
        if(tokens.length > 15) {
                //  for(String s : tokens) {
                //  System.out.println(tokens[17]);
                context.write(new Text(tokens[17]), one);
        }
        //}
```

```
    }

@Override
protected void cleanup(Mapper<LongWritable, Text, Text, IntWritable>.Context context)
    throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    super.cleanup(context);
    }

@Override
public void run(Mapper<LongWritable, Text, Text, IntWritable>.Context context)
    throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    super.run(context);
    }

@Override
protected void setup(Mapper<LongWritable, Text, Text, IntWritable>.Context context)
    throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    super.setup(context);
    }


}

package CountPurposeDetermineNumberLoans;
```

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class CountPurposeReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

   @Override
   protected void cleanup(Reducer<Text, IntWritable, Text, IntWritable>.Context context)
         throws IOException, InterruptedException {
      // TODO Auto-generated method stub
      super.cleanup(context);
   }

   @Override
   protected void reduce(Text key, Iterable<IntWritable> values,
                  Reducer<Text, IntWritable, Text, IntWritable>.Context context) throws IOException,
InterruptedException {

      int sum = 0;

      for (IntWritable value : values) {
         sum += value.get();
      }

      context.write(key, new IntWritable(sum));
   }
```

```
    @Override
    public void run(Reducer<Text, IntWritable, Text, IntWritable>.Context arg0)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        super.run(arg0);
    }

    @Override
    protected void setup(Reducer<Text, IntWritable, Text, IntWritable>.Context context)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        super.setup(context);
    }

}
```

DistinctFilteringByGrade

```
package DistinctFilteringByGrade;

import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
```

```
import java.io.IOException;


public class DistinctDriver {

   public static void main(String[] args) throws IOException, ClassNotFoundException,
InterruptedException {

      Job job = Job.getInstance();

      job.setJarByClass(DistinctDriver.class);


      job.setMapperClass(DistinctMapper.class);

      job.setCombinerClass(DistinctReducer.class);

      job.setReducerClass(DistinctReducer.class);


      job.setInputFormatClass(TextInputFormat.class);

      job.setOutputFormatClass(TextOutputFormat.class);


      job.setOutputKeyClass(Text.class);

      job.setOutputValueClass(NullWritable.class);


      job.setNumReduceTasks(1);


      FileInputFormat.addInputPath(job, new Path(args[0]));

      FileOutputFormat.setOutputPath(job, new Path(args[1]));


      FileSystem fs = FileSystem.get(job.getConfiguration());



      job.waitForCompletion(true);

   }

}
```

```
package DistinctFilteringByGrade;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class DistinctMapper extends Mapper<LongWritable, Text,Text, NullWritable> {
   private static final IntWritable one = new IntWritable(1);

   @Override
   protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text,
NullWritable>.Context context)
         throws IOException, InterruptedException {
      String line = value.toString();

      String[] tokens = line.split(",");
      if(tokens.length > 15 && !line.contains("grade")) {
         context.write(new Text(tokens[8]), NullWritable.get()); //10
      }

   }

   @Override
   protected void cleanup(Mapper<LongWritable, Text, Text, NullWritable>.Context context)
```

```
        throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    super.cleanup(context);
  }

  @Override
  public void run(Mapper<LongWritable, Text, Text, NullWritable>.Context context)
        throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    super.run(context);
  }

  @Override
  protected void setup(Mapper<LongWritable, Text, Text, NullWritable>.Context context)
        throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    super.setup(context);
  }


}

package DistinctFilteringByGrade;

import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;
```

```
public class DistinctReducer extends Reducer<Text, NullWritable, Text, NullWritable> {

   @Override
   protected void cleanup(Reducer<Text, NullWritable, Text, NullWritable>.Context context)
         throws IOException, InterruptedException {
      // TODO Auto-generated method stub
      super.cleanup(context);
   }

   @Override
   protected void reduce(Text key, Iterable<NullWritable> values,
                  Reducer<Text, NullWritable, Text, NullWritable>.Context context) throws IOException,
InterruptedException {

      context.write(key, NullWritable.get());
   }

   @Override
   public void run(Reducer<Text, NullWritable, Text, NullWritable>.Context arg0)
         throws IOException, InterruptedException {
      // TODO Auto-generated method stub
      super.run(arg0);
   }

   @Override
   protected void setup(Reducer<Text, NullWritable, Text, NullWritable>.Context context)
         throws IOException, InterruptedException {
      // TODO Auto-generated method stub
```

```
        super.setup(context);
    }

}
```

InnerJoin

```
package InnerJoin;

import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.MultipleInputs;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class JoinPattern {

    public static void main(String[] args){

        try {
            Job job = Job.getInstance();
            job.setJarByClass(JoinPattern.class);

            job.getConfiguration().set("type",args[3]);

            MultipleInputs.addInputPath(job, new Path(args[0]),TextInputFormat.class,
DistinctMapper.class);
```

```
        MultipleInputs.addInputPath(job,new Path(args[1]), TextInputFormat.class, CSVMapper.class);

        //Reducer
        job.setReducerClass(JoinReducer.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        //Number of Reducers
        job.setNumReduceTasks(1);

        //Specify Key value
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        //Specify Output Path
        Path output = new Path(args[2]);
        FileOutputFormat.setOutputPath(job, output);

        FileSystem fs = FileSystem.get(job.getConfiguration());
        if(fs.exists(output)) {
           fs.delete(output, true);
        }

        System.exit(job.waitForCompletion(true) ? 0 : 1);

      }catch(Exception ex){
        ex.printStackTrace();
   }
   }
}
```

```
package InnerJoin;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class CSVMapper extends Mapper<LongWritable, Text, Text, Text> {

   @Override
   protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
      String[] fields = value.toString().split(",");
      Text t1 = new Text(fields[0]);
      String val = "M"+fields[1]+","+fields[2]+","+fields[3];
      Text t2 = new Text(val);

      context.write(t1,t2);
   }
}

package InnerJoin;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
```

```
import java.io.IOException;


public class DistinctMapper extends Mapper<LongWritable, Text, Text, Text> {


   @Override

   protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {


      String[] fields = value.toString().split(" ");

      Text t1 = new Text(fields[0].replace("\"","").trim());

      String val = "R"+fields[0].replace("\"","").trim();

      Text t2 = new Text(val);

      context.write(t1,t2);

   }

}


package InnerJoin;


import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;


import java.io.IOException;


public class DistinctMapper extends Mapper<LongWritable, Text, Text, Text> {


   @Override

   protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
```

```
        String[] fields = value.toString().split(" ");
        Text t1 = new Text(fields[0].replace("\"","").trim());
        String val = "R"+fields[0].replace("\"","").trim();
        Text t2 = new Text(val);
        context.write(t1,t2);
    }
}
```

InvertedIndex

```
package InvertedIndex;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import java.io.IOException;

public class Driver {
    public static void main( String[] args ) throws IOException, ClassNotFoundException,
InterruptedException
    {
        Job job = Job.getInstance();
```

```
 job.setJarByClass(Driver.class);

 FileInputFormat.addInputPath(job, new Path(args[0]));
 Path outDir = new Path(args[1]+"_1");
 FileOutputFormat.setOutputPath(job, outDir);

 job.setMapperClass(FilterMapper.class);
//  job.setReducerClass(MyReducer.class);
 // job.setCombinerClass(MyReducer.class);

 job.setOutputFormatClass(TextOutputFormat.class);
 job.setMapOutputValueClass(NullWritable.class);
 job.setMapOutputKeyClass(Text.class);
 job.setInputFormatClass(TextInputFormat.class);

//  job.setNumReduceTasks(1);


 Boolean a=job.waitForCompletion(true);

 if(a==true)
 {
    Job job2 = Job.getInstance();

    job2.setJarByClass(Driver.class);

    // job2.setGroupingComparatorClass(GroupComparator.class);
    // job2.setSortComparatorClass(SecodarySortComparator.class);
    //job2.setPartitionerClass(KeyPartition.class);
```

```
        FileInputFormat.addInputPath(job2, new Path(args[1]+"_1"));
        Path outDir1 = new Path(args[1]);
        FileOutputFormat.setOutputPath(job2, outDir1);

        job2.setMapperClass(MyMapper.class);
        job2.setReducerClass(MyReducer.class);
        job2.setCombinerClass(MyReducer.class);

        job2.setOutputFormatClass(TextOutputFormat.class);
        job2.setMapOutputValueClass(Text.class);
        job2.setMapOutputKeyClass(Text.class);
        job2.setInputFormatClass(TextInputFormat.class);

        job2.setNumReduceTasks(1);

        // job2.setOutputKeyClass(CompositeKeyWritable.class);
        // job2.setOutputValueClass(NullWritable.class);


        job2.waitForCompletion(true);
      }
   }
}

package InvertedIndex;

import com.google.common.base.Charsets;
import com.google.common.hash.BloomFilter;
```

```
import com.google.common.hash.Funnel;
import com.google.common.hash.Sink;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.util.ArrayList;

public class BloomMapper extends Mapper<LongWritable, Text, Text, NullWritable> {

   private BloomFilter<String> origin;

   Funnel<String> p = new Funnel<String>() {

      public void funnel(String from, Sink into) {

         into.putString(from, Charsets.UTF_8);

      }
   };

   @Override
    protected void setup(Context context) throws IOException, InterruptedException {

      this.origin = BloomFilter.create(p, 300000, 0.1);

      String p1 = "MA";
```

```
        String p2 = "NY";
       // Person p2 = new Person("Jamie", "Scott");

        ArrayList<String> originList = new ArrayList<String>();
        originList.add(p1);
        originList.add(p2);

        for (String ps : originList) {
            origin.put(ps);
        }

    }

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
        try {
            String input = value.toString();

            String splitString[] = input.split(",");
            String origins = splitString[19];
            //System.out.println(origins);
            //String values[] = value.toString().split(",");
            //Person p = new Person(values[1], values[2]);

            if (origin.mightContain(origins) && !splitString[0].contains("id")) {
                context.write(value,NullWritable.get());
            }
```

```
      } catch(Exception e) {
         System.out.println(e);
      }
   }

   @Override
   protected void cleanup(Context context) throws IOException, InterruptedException
   {
      super.cleanup(context);
   }
}

package InvertedIndex;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class MyMapper extends Mapper<LongWritable, Text, Text, Text> {
   @Override
   protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
      try {
         String input = value.toString();

         String splitString[] = input.split(",");
         String origin = splitString[24];
```

```
            context.write(new Text(origin), new Text(splitString[0]));
        } catch(Exception e) {
            System.out.println(e);
        }
    }
}
```

```
package InvertedIndex;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class MyReducer extends Reducer<Text, Text, Text, Text> {
    @Override
    protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
InterruptedException {

        String concatString = "";

        for(Text s:values) {
            if(!concatString.contains(s.toString())) {
                concatString = s.toString() + "," + concatString;
            }

        }
        context.write(key,new Text(concatString));
```

```
   }
}
```

MaxAndMinIntRateForEmpLength

```
package MaxAndMinIntRateForEmpLength;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.NLineInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

public class MaxMinRateDriver {
   public static void main( String[] args ) throws IOException, ClassNotFoundException,
InterruptedException
   {
      Job job = Job.getInstance();

      job.setJarByClass(MaxMinRateDriver.class);

      FileInputFormat.addInputPath(job, new Path(args[0]));
      Path outDir = new Path(args[1]);
      FileOutputFormat.setOutputPath(job, outDir);
```

```
        job.setMapperClass(MaxMinRateMapper.class);
        job.setReducerClass(MaxMinRateReducer.class);
        job.setCombinerClass(MaxMinRateReducer.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);
        job.setInputFormatClass(NLineInputFormat.class);
        NLineInputFormat.setNumLinesPerSplit(job, 10000);
        job.setNumReduceTasks(1);



        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);


        job.waitForCompletion(true);
    }
}

package MaxAndMinIntRateForEmpLength;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class MaxMinRateMapper extends Mapper<LongWritable,Text,Text,Text> {
```

```
    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
        Text symbol;
        String line = value.toString();
        String[] fields = line.split(",");
        if (fields.length > 5) {
            if (!line.contains("int_rate")) {
                symbol = new Text(fields[10]);
                context.write(symbol, new Text(fields[6]+" "+fields[6]));
            }

        }
    }
}

package MaxAndMinIntRateForEmpLength;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class MaxMinRateReducer extends Reducer<Text,Text,Text,Text> {
    @Override
    protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
InterruptedException {
        Float currentMaxRateValue;
        Float currentMinRateValue;
```

```
        Float maxRateValue = (float) 0;
        Float minRateValue = (float) Integer.MAX_VALUE;

        for (Text value : values) {
String a[]=value.toString().split(" ");
           currentMaxRateValue = Float.parseFloat(a[1].trim().replace("%", ""));
           currentMinRateValue = Float.parseFloat(a[0].trim().replace("%", ""));

           if (currentMaxRateValue > maxRateValue) {
              maxRateValue = currentMaxRateValue;
           }
           if (currentMinRateValue < minRateValue) {
              minRateValue = currentMinRateValue;
           }
        }
      //  String result = "Min Rate: " + String.valueOf(minRateValue) + " Max Rate: "
      //         + String.valueOf(maxRateValue);
        context.write(key, new Text(minRateValue+" "+maxRateValue));
    }
}
```

Median

```
 package Median;

import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
```

```
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import java.io.IOException;

public class Driver {
   public static void main( String[] args ) throws IOException, ClassNotFoundException,
InterruptedException
   {
      Job job = Job.getInstance();

      job.setJarByClass(Driver.class);

      FileInputFormat.addInputPath(job, new Path(args[0]));
      Path outDir = new Path(args[1]);
      FileOutputFormat.setOutputPath(job, outDir);

      job.setMapperClass(FilterMapper.class);
      job.setReducerClass(FilterReducer.class);

      job.setInputFormatClass(TextInputFormat.class);
      job.setOutputFormatClass(TextOutputFormat.class);

      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(Text.class);

      job.setNumReduceTasks(1);
```

```
      FileInputFormat.addInputPath(job, new Path(args[0]));
      FileOutputFormat.setOutputPath(job, new Path(args[1]));

      FileSystem fs = FileSystem.get(job.getConfiguration());

      job.waitForCompletion(true);
   }
}

package Median;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class FilterMapper extends Mapper<LongWritable, Text, Text, Text> {
   @Override
   protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
      try {
         String split[]=value.toString().split(",");
        // IntWritable one = new IntWritable(1);
         String  symbol = split[0];
         if(!symbol.contains("member_id") && split.length>20 && !split[5].equalsIgnoreCase("0") &&
!split[20].equalsIgnoreCase("") && !split[20].equalsIgnoreCase("")) {
            context.write(new Text(split[8]),new Text(split[24]));
```

```
        }
      } catch(Exception ex) {
        System.out.println("Exception in mapper"+ex);
      }
    }
}

package Median;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;

public class FilterReducer extends Reducer<Text, Text, Text,Text> {
   ArrayList<Integer> array= new ArrayList<Integer>();
   @Override
   protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
InterruptedException {
   Integer median=0;
      //int sum = 0;
      //int count = 0;

//Integer temp= (int) Float.parseFloat(key.toString());
      try {
         for (Text value:values) {
```

```
            Integer temp = (int) Float.parseFloat(value.toString());
            if (temp != 0){
               array.add(temp);
            }
            //context.write(new Text(key), NullWritable.get());
         }
         Collections.sort(array);
         System.out.println(array.size()+":"+array);
         int len=array.size();
         if(array.size()%2==0){
            median= (array.get((int) len/2 - 1)+array.get((int) len/2)) /2;
         }
         else{
            median= array.get(len/2);
         }

         context.write(key,new Text(median+""));

      }
      catch (Exception e){
         System.out.println(key+e.getMessage());
      }
   }
}
```

PutMerge

```
package PutMerge;

import org.apache.hadoop.conf.Configuration;
```

```
import org.apache.hadoop.fs.*;import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;


public class MergeFiles {

   public static void main(String[] args) throws IOException, ClassNotFoundException,
InterruptedException {//Hadoop File System instance

      Configuration conf = new Configuration();

      FileSystem hdfs = FileSystem.get(conf);       //local File System instance

      FileSystem local = FileSystem.getLocal(conf);       //Path to the directory in local file System

      Path inputDir = new Path(args[0]);

      Path hdfsFile = new Path(args[1]);       try {

         //List of Files Names from FileStatus

         FileStatus[] inputFiles = local.listStatus(inputDir);

         //Writing to hdfs using OutputStream

         FSDataOutputStream out = hdfs.create(hdfsFile);          //Reading the input files using
FSDataInputStream

         for (int i = 0; i < inputFiles.length; i++) {

            FSDataInputStream in = local.open(inputFiles[i].getPath());

            // BufferedReader br = new BufferedReader(new InputStreamReader(in, UTF8));

            //int lineCount = 0;

            byte buffer[] = new byte[256];

            int bytesRead = 0;              while ((bytesRead = in.read(buffer)) > 0 ) {
out.write(buffer, 0, bytesRead);

            }

            in.close();

         }

         out.close();       } catch (IOException e) {

         e.printStackTrace();

      }
```

```
   }
}
```

Secondary Sort

```
package SecSort;

import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

public class Driver
{
   public static void main( String[] args ) throws IOException, ClassNotFoundException,
InterruptedException
   {
      Job job = Job.getInstance();

      job.setJarByClass(Driver.class);

      job.setGroupingComparatorClass(GroupComparator.class);
      job.setSortComparatorClass(SecodarySortComparator.class);
      job.setPartitionerClass(KeyPartition.class);

      FileInputFormat.addInputPath(job, new Path(args[0]));
```

```
        Path outDir = new Path(args[1]);
        FileOutputFormat.setOutputPath(job, outDir);

        job.setMapperClass(MyMapper.class);
        job.setReducerClass(MyReducer.class);

        job.setNumReduceTasks(1);

        job.setOutputKeyClass(CompositeKeyWritable.class);
        job.setOutputValueClass(NullWritable.class);

        FileSystem fs = FileSystem.get(job.getConfiguration());
        if(fs.exists(outDir)) {
            fs.delete(outDir, true);
        }

job.waitForCompletion(true);

//      }
   }
}

package SecSort;

import org.apache.hadoop.io.WritableComparable;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
```

```
public class CompositeKeyWritable implements WritableComparable<CompositeKeyWritable> {

	private String grade;
	private String subgrade;

	public CompositeKeyWritable() {
		super();
	}

	public CompositeKeyWritable(String grade, String id) {
		super();
		this.grade = grade;
		this.subgrade = id;
	}

	public void write(DataOutput out) throws IOException {

		out.writeUTF(grade);
		out.writeUTF(subgrade);
	}

	public void readFields(DataInput in) throws IOException {

		grade = in.readUTF();
		subgrade = in.readUTF();
	}

	public int compareTo(CompositeKeyWritable o) {
```

```
		int result = this.grade.compareTo(o.grade);
		if (result == 0) {
			return this.subgrade.compareTo(o.subgrade);
		}
		return result;
}

public String getGrade() {
		return grade;
}

public void setGrade(String grade) {
		this.grade = grade;
}

public String getSubgrade() {
		return subgrade;
}

public void setSubgrade(String subgrade) {
		this.subgrade = subgrade;
}

@Override
public String toString() {

		return grade + "," + subgrade;
}
```

```
}

package SecSort;

import org.apache.hadoop.io.WritableComparator;

public class GroupComparator extends WritableComparator {

	protected GroupComparator() {
		super(CompositeKeyWritable.class, true);
	}

	@Override
	public int compare(Object a, Object b) {

		CompositeKeyWritable ckw1 = (CompositeKeyWritable)a;
		CompositeKeyWritable ckw2 = (CompositeKeyWritable)b;

		return ckw1.getGrade().compareTo(ckw2.getGrade());
	}

}

package SecSort;

import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Partitioner;

public class KeyPartition extends Partitioner<CompositeKeyWritable, NullWritable> {
```

```
	@Override
	public int getPartition(CompositeKeyWritable key, NullWritable value, int numPartitions) {

		return key.getGrade().hashCode()%numPartitions;

	}

}

package SecSort;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class MyMapper extends Mapper<LongWritable, Text, CompositeKeyWritable, NullWritable> {

	@Override
	protected void map(LongWritable key, Text value,
	    Context context)
			throws IOException, InterruptedException {
		String line = value.toString();

		String[] tokens = line.split(",");
		String grade = null;
```

```
                String subbgrade = null;
        if(!value.toString().contains("member_id") && tokens.length>14)  {
        try {

                grade = tokens[8];
                subbgrade = tokens[9];

        } catch (Exception e) {

        }

        if (grade != null && subbgrade != null) {

                CompositeKeyWritable outKey = new CompositeKeyWritable(grade, subbgrade);

                context.write(outKey, NullWritable.get());

        }
        }
        }
}

package SecSort;

import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;
```

```
public class MyReducer extends Reducer<CompositeKeyWritable, NullWritable, CompositeKeyWritable,
NullWritable> {

	@Override
	protected void reduce(CompositeKeyWritable key, Iterable<NullWritable> values,
		Context context)
			throws IOException, InterruptedException {

	  //for (NullWritable v : values) {
			context.write(key, NullWritable.get());
	  //}

	}

}

package SecSort;

import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;

public class SecodarySortComparator extends WritableComparator {

	protected SecodarySortComparator() {
		super(CompositeKeyWritable.class, true);
	}

	@Override
```

```
	public int compare(WritableComparable a, WritableComparable b) {

		CompositeKeyWritable ckw1 = (CompositeKeyWritable) a;
		CompositeKeyWritable ckw2 = (CompositeKeyWritable) b;

		int result = ckw1.getGrade().compareTo(ckw2.getGrade());

		if (result == 0) {
			return ckw1.getSubgrade().compareTo(ckw2.getSubgrade());
		}

		return result;
	}



}
```

ShuffleMemberId

```
package ShuffleMemberId;

import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
```

```
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;


import java.io.IOException;


public class ShuffleDriver {

   public static void main(String[] args) throws IOException, ClassNotFoundException,
InterruptedException {

      Job job = Job.getInstance();

      job.setJarByClass(ShuffleDriver.class);


      job.setMapperClass(ShuffleMapper.class);

      job.setReducerClass(ShuffleReducer.class);


      job.setInputFormatClass(TextInputFormat.class);

      job.setOutputFormatClass(TextOutputFormat.class);


      job.setOutputKeyClass(IntWritable.class);

      job.setOutputValueClass(Text.class);


      job.setNumReduceTasks(1);


      FileInputFormat.addInputPath(job, new Path(args[0]));

      FileOutputFormat.setOutputPath(job, new Path(args[1]));


      FileSystem fs = FileSystem.get(job.getConfiguration());



      job.waitForCompletion(true);
```

```
    }
}

package ShuffleMemberId;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.util.Random;

public class ShuffleMapper extends Mapper<LongWritable, Text,IntWritable, Text> {
    private static final IntWritable one = new IntWritable(1);
    private Random random= new Random();
    @Override
    protected void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        String line = value.toString();

        String[] tokens = line.split(",");
        if(tokens.length > 15) {
            context.write(new IntWritable(random.nextInt()), value);
        }

    }
```

```
    @Override
    protected void cleanup(Context context)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        super.cleanup(context);
    }

    @Override
    public void run(Context context)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        super.run(context);
    }

    @Override
    protected void setup(Context context)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        super.setup(context);
    }


}

package ShuffleMemberId;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
```

```
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class ShuffleReducer extends Reducer<IntWritable, Text,Text, NullWritable> {

   @Override
   protected void cleanup(Context context)
         throws IOException, InterruptedException {
      // TODO Auto-generated method stub
      super.cleanup(context);
   }

   @Override
   protected void reduce(IntWritable key, Iterable<Text> values,
                  Context context) throws IOException, InterruptedException {
      for (Text val: values) {
        // context.write(new IntWritable(),val);
         context.write(val,NullWritable.get());
      }
   }

   @Override
   public void run(Context arg0)
         throws IOException, InterruptedException {
      // TODO Auto-generated method stub
      super.run(arg0);
   }
```

```
    @Override
    protected void setup(Context context)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        super.setup(context);
    }

}
```

Top10MembersLoanAmt

```
package Top10MembersLoanAmt;

import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import java.io.IOException;



/**
 * Hello world!
 *
 */
```

```
public class App
{
   public static void main( String[] args ) throws IOException, ClassNotFoundException,
InterruptedException
   {
      Job job = Job.getInstance();
      job.setJarByClass(App.class);

      job.setInputFormatClass(TextInputFormat.class);
      job.setOutputFormatClass(TextOutputFormat.class);

      Path input = new Path(args[0]);
      Path output = new Path(args[1]);

      job.setOutputKeyClass(NullWritable.class);
      job.setOutputValueClass(Text.class);

      FileInputFormat.addInputPath(job, input);
      FileOutputFormat.setOutputPath(job, output);

      job.setNumReduceTasks(1);

      job.setMapperClass(TopKMapper.class);
      job.setReducerClass(TopKReducer.class);

      FileSystem fs = FileSystem.get(job.getConfiguration());
      if(fs.exists(output)) {
          fs.delete(output, true);
      }
```

```
        job.waitForCompletion(true);
    }
}
```

```
package Top10MembersLoanAmt;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.util.TreeMap;

public class TopKMapper extends Mapper<LongWritable, Text, NullWritable, Text> {

        private static final int K = 10;
        private TreeMap<Integer, Text> tMap;

        @Override
        protected void setup(Mapper<LongWritable, Text, NullWritable, Text>.Context context)
                        throws IOException, InterruptedException {
                this.tMap = new TreeMap<Integer, Text>();
        }

        @Override
        protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, NullWritable,
Text>.Context context)
```

```
                throws IOException, InterruptedException {

        String[] tokens = value.toString().split(",");
    if(!value.toString().contains("id") && tokens.length>8) {
    try {

        String score = tokens[7].replace("\"", "").trim();
        tMap.put(Integer.parseInt(score), new Text(value));

    } catch (Exception ex) {

    }

    if (tMap.size() > K) {
        tMap.remove(tMap.firstKey());
    }
}
    }

    @Override
    protected void cleanup(Mapper<LongWritable, Text, NullWritable, Text>.Context context)
                throws IOException, InterruptedException {

        for (Text t : tMap.values()) {
            context.write(NullWritable.get(), t);
        }

    }
```

```
}

package Top10MembersLoanAmt;

import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;
import java.util.TreeMap;

public class TopKReducer extends Reducer<NullWritable, Text, NullWritable, Text> {

	private static final int K = 10;
	private TreeMap<Integer, Text> tMap;

	@Override
	protected void setup(Reducer<NullWritable, Text, NullWritable, Text>.Context context)
			throws IOException, InterruptedException {

		this.tMap = new TreeMap<Integer, Text>();

	}

	@Override
	protected void reduce(NullWritable arg0, Iterable<Text> values,
		Reducer<NullWritable, Text, NullWritable, Text>.Context arg2) throws IOException,
InterruptedException {
```

```
            for (Text t : values) {
                String[] tokens = t.toString().split(",");
                tMap.put(Integer.parseInt(tokens[7]), new Text(t));

                if (tMap.size() > K) {
                    tMap.remove(tMap.firstKey());
                }
            }
        }

        @Override
        protected void cleanup(Reducer<NullWritable, Text, NullWritable, Text>.Context context)
                throws IOException, InterruptedException {

            for(Text t : tMap.values()) {
                context.write(NullWritable.get(), t);
            }
        }

}
```

VerificationStatusMedianStd

```
package VerificationStatusMedianStd;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.SortedMapWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
```

```
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

public class Driver {
   public static void main( String[] args ) throws IOException, ClassNotFoundException,
InterruptedException
   {
      Job job = Job.getInstance();

      job.setJarByClass(Driver.class);

      FileInputFormat.addInputPath(job, new Path(args[0]));
      Path outDir = new Path(args[1]);
      FileOutputFormat.setOutputPath(job, outDir);

      job.setMapperClass(MyMapper.class);
      job.setReducerClass(MyReducer.class);
      job.setCombinerClass(myCombiner.class);

      //job.setOutputFormatClass(TextOutputFormat.class);
      job.setMapOutputValueClass(SortedMapWritable.class);
      job.setMapOutputKeyClass(Text.class);
      job.setInputFormatClass(TextInputFormat.class);
      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(MedianStdDevTuple.class);
```

```
        job.setNumReduceTasks(1);

        job.waitForCompletion(true);
    }
}
```

```
package VerificationStatusMedianStd;

import org.apache.hadoop.io.Writable;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

public class MedianStdDevTuple implements Writable {
    private  float median;
    private float stdDev;

    public MedianStdDevTuple(float median, float stdDev) {
        this.median = median;
        this.stdDev = stdDev;
    }

    public float getMedian() {
        return median;
    }

    public void setMedian(float median) {
        this.median = median;
```

```
}

public float getStdDev() {
   return stdDev;
}

public void setStdDev(float stdDev) {
   this.stdDev = stdDev;
}

public void write(DataOutput out) throws IOException {
   out.writeFloat(this.median);
   out.writeFloat(this.stdDev);
}

@Override
public String toString() {

   return "median:"+ this.median +" Std_Div:"+this.stdDev;
}

public void readFields(DataInput in) throws IOException {
   this.median=in.readFloat();
   this.stdDev=in.readFloat();
}


public MedianStdDevTuple() {
}
```

```
}

package VerificationStatusMedianStd;

import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;
import java.util.Map;

public class myCombiner extends Reducer<Text, SortedMapWritable, Text, SortedMapWritable> {
   @Override
   protected void reduce(Text key, Iterable<SortedMapWritable> values, Context context) throws
IOException, InterruptedException {
      // int sum=0;
       SortedMapWritable map = new SortedMapWritable();
       for(SortedMapWritable v:values) {
          for(Object entry : v.entrySet() ) {
             if(entry instanceof Map.Entry) {
                Map.Entry entry1 = (Map.Entry) entry;
                IntWritable count = (IntWritable) map.get(entry1.getKey());
                FloatWritable mapKey = ((FloatWritable) entry1.getKey());
                Integer mapValue = ((IntWritable) entry1.getValue()).get();
                if (count != null) {
                   map.put(mapKey, new IntWritable(count.get() + ((IntWritable) entry1.getValue()).get()));
                } else {
                   map.put(mapKey, new IntWritable(((IntWritable) entry1.getValue()).get()));
                }
             }
```

```
        }
        v.clear();
      }
      context.write(key,map);

   }
}

package VerificationStatusMedianStd;

import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class MyMapper extends Mapper<LongWritable, Text, Text, SortedMapWritable> {
   @Override
   protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
      try {
         String split[]=value.toString().split(",");
         IntWritable one = new IntWritable(1);
         String  symbol = split[13];
         if(!split[0].contains("id") && split.length>6 ) {
            Float avg = Float.parseFloat(split[2]);
            SortedMapWritable map = new SortedMapWritable();
            map.put(new FloatWritable(avg),one);
            context.write(new Text(symbol.replace("\"","")),map);
         }
```

```
      } catch(Exception ex) {
         System.out.println("Exception in mapper"+ex);
      }
   }
}

package VerificationStatusMedianStd;

import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;
import java.util.Map;
import java.util.TreeMap;

public class MyReducer extends Reducer<Text, SortedMapWritable, Text,MedianStdDevTuple> {
   @Override
   protected void reduce(Text key, Iterable<SortedMapWritable> values, Context context) throws
IOException, InterruptedException {

      //int sum = 0;
      //int count = 0;
      MedianStdDevTuple result = new MedianStdDevTuple();
      TreeMap<Float, Integer> finalMap= new TreeMap<Float, Integer>();

      try{
         Float sum = 0f;
         Integer totalCount = 0;
```

```
for(SortedMapWritable v:values) {
   for(Object entry: v.entrySet() ) {
      if(entry instanceof Map.Entry) {
         Map.Entry entry1 = (Map.Entry) entry;
         Float mapKey = ((FloatWritable) entry1.getKey()).get();
         Integer mapValue = ((IntWritable) entry1.getValue()).get();
         Integer count = finalMap.get(mapKey);

         sum = sum + mapValue * mapKey;
         totalCount = totalCount + mapValue;



         if (count != null) {
            finalMap.put(mapKey, mapValue + count);
         } else {
            finalMap.put(mapKey, mapValue);
         }
      }

   }
   v.clear();
}
Integer middleIndex = totalCount/2;
int prev=0;
int current = 0;
Float prevKey = 0f;
Float median=0f;
```

```
for(Map.Entry<Float,Integer> entry: finalMap.entrySet()) {
   current = entry.getValue();
   if(middleIndex > prev && middleIndex<=(current+prev)) {
      if(totalCount%2==0) {
         median = (entry.getKey()+prevKey)/2;

      } else {
         median = entry.getKey();
      }
     break;
   }
   prev= current+prev;
   prevKey = entry.getKey();

}
result.setMedian(median);

Float mean = sum/totalCount;

Float sumOfSquares=0.f;
for(Map.Entry<Float,Integer> entry:finalMap.entrySet()) {
   sumOfSquares = sumOfSquares + (entry.getKey()-mean)*(entry.getKey()-mean);
}
Float stdDev = (float) Math.sqrt(sumOfSquares/(totalCount-1));

result.setStdDev(stdDev);

context.write(key,result);
```

```
        } catch(Exception e) {
            System.out.println("exception in reducer "+ e);
        }
    }
}
```

Pig:

Filtering:

```
users = LOAD 'LoanStats3a_securev1.csv' using PigStorage(',');
s1 = FILTER users BY $2 < 3000;
s2 = FOREACH s1 GENERATE $0,$1,$2,$3,$4,$5,$6,$7,$8,$9,$10,$11,$12,$13,$14,'T';
STORE s2 into 'SmallLoanAmount';
```

Joins:

```
grade = LOAD 'grade' using PigStorage(',') as (grd);
detail = LOAD 'gradedetails.csv' using PigStorage(',') as (id,desc,name,creditscore);
joinop = JOIN grade BY grd, detail BY id;
out = FOREACH joinop GENERATE $0,$2,$3,$4;
STORE summed out 'gradeoutput';
```

Top K:

```
users = LOAD 'LoanStats3a_securev1.csv' using PigStorage(',');
grouped = GROUP users BY $24;
summed = FOREACH grouped GENERATE group, COUNT(users) AS cntd;
sorted = ORDER summed BY cntd DESC;
top25 = LIMIT sorted 25;
STORE top25 into 'topk_ficoscore_pig.csv';
```