



Northeastern University

H1B Petitions Data Analysis

Final Project

INFO7275 Advanced Database Mgt Sys (Fall 2017)

Ujjval Hemant Thakkar

NUID: 001232511

13/12/2017

Contents

Problem Statement.....	3
Dataset.....	4
Analysis Description.....	5
1. Number of Full-time and Part-time petitions (Normal MR Job).....	5
2. Counting number of petitions based on Location (Counting with counters Summarization).....	5
3. Predictive yearly analysis of successful certification of the petitions (Numerical Summarization, Mean, Standard Deviation, MR Chaining).....	6
4. Top 10 employers to file the most petitions (Top N Filtering Pattern, Partitioner).....	7
5. Most Data Engineer jobs for each year for each Location (Top N Filtering Pattern, Partitioner) ..	8
6. Categorizing petitions filed for a "CEO" for each year (Binning Data Organization Pattern)	9
7. Successful Petitions based on the Wages (Inverted Index Pattern)	10
8. Most Data Scientist positions by Industry (HIVE).....	12
9. Top 10 Most petitions filed by Employers by Year (HIVE).....	14
10. Counting number of petitions based on Location (HIVE)	14
11. Categorizing petitions filed for a "CEO" for each year (HIVE).....	16
12. Counting of Number of Full time and Part time petitions (HIVE)	16
13. Top 10 Locations with most DATA ENGINEER petitions.....	16
14. Top Data Engineer Locations for each year (PIG)	17
15. Top10 Industry by most Data Scientist positions (PIG)	19
Visualization and Charts.....	20
Appendix	25
1. Code: Number of Full time and Part time petitions (Normal MR Job).....	25
2. Code: Number of petitions based on Location (Counting with counters Summarization)	27
3. Code: Predictive yearly analysis of successful certification of the petitions (Numerical Summarization, Standard Deviation, MR Chaining)	29
4. Code: Top 10 employers to file the most petitions (Top N Filtering Pattern, Partitioner)	32
5. Code: Most Data Engineer jobs for each year for each Location (Top N Filtering Pattern, Partitioner)	34
6. Code: Categorizing petitions filed for a "CEO" for each year (Binning Data Organization Pattern)	36
7. Successful Petitions based on the Wages (Inverted Index Pattern)	38

Problem Statement

The H-1B is an employment-based, non-immigrant visa category for temporary foreign workers in the United States. For a foreign national to apply for H1-B visa, an US employer must offer a job and petition for H-1B visa with the US immigration department. This is the most common visa status applied for and held by international students once they complete college/ higher education (Masters, PhD) and work in a full-time position.

We will be performing the following analysis on this H1B visa petition applications between the years 2011-2016 using **Hadoop MapReduce Jobs**:

- Number of Full time and Part time petitions (**Normal MR Job**)
- Number of petitions based on Location (**Counting with counters Summarization**)
- Predictive yearly analysis of successful certification of the petitions (**Numerical Summarization, Standard Deviation, MR Chaining**)
- Top 10 employers to file the most petitions (**Top N Filtering Pattern, Partitioner**)
- Most Data Engineer jobs for each year for each Location (**Top N Filtering Pattern, Partitioner**)
- Categorizing petitions filed for a "CEO" for each year (**Binning Data Organization Pattern**)
- Successful Petitions based on the Wages (**Inverted Index Pattern**)

The following analysis is performed using **HIVE**:

- Most Data Scientist positions by Industry (HIVE)
- Top 10 Most petitions filed by Employers by Year (HIVE)
- Counting number of petitions based on Location (HIVE)
- Categorizing petitions filed for a "CEO" for each year (HIVE)
- Counting of Number of Full time and Part time petitions (HIVE)

The following analysis is performed using **PIG**:

- Top Data Engineer Locations for each year (PIG)
- Top10 Industry by most Data Scientist positions (PIG)

Dataset

Link: <https://www.kaggle.com/nsharan/h-1b-visa/data>

The columns in the dataset include:

The **CASE_STATUS** field denotes the status of the application after LCA processing. Certified applications are filed with USCIS for H-1B approval. **CASE_STATUS: CERTIFIED** does not mean the applicant got his/her H-1B visa approved, it just means that he/she is eligible to file an H-1B.

- i) **CASE_STATUS**: Status associated with the last significant event or decision. Valid values include "Certified," "Certified-Withdrawn," "Denied," and "Withdrawn".
- ii) **EMPLOYER_NAME**: Name of employer submitting labor condition application.
- iii) **SOC_NAME**: Occupational name associated with the SOC_CODE. SOC_CODE is the occupational code associated with the job being requested for temporary labor condition, as classified by the Standard Occupational Classification (SOC) System.
- iv) **JOB_TITLE**: Title of the job
- v) **FULL_TIME_POSITION**: Y = Full Time Position; N = Part Time Position
- vi) **PREVAILING_WAGE**: Prevailing Wage for the job being requested for temporary labor condition. The wage is listed at annual scale in USD. The prevailing wage for a job position is defined as the average wage paid to similarly employed workers in the requested occupation in the area of intended employment. The prevailing wage is based on the employer's minimum requirements for the position.
- vii) **YEAR**: Year in which the H-1B visa petition was filed
- viii) **WORKSITE**: City and State information of the foreign worker's intended area of employment
- ix) **lon**: longitude of the Worksite
- x) **lat**: latitude of the Worksite

Raw dataset description:

https://www.foreignlaborcert.doleta.gov/docs/Performance_Data/Disclosure/FY15-FY16/H-1B_FY16_Record_Layout.pdf

H-1B process: <http://www.immi-usa.com/h1b-application-process-step-by-step-guide/>

Analysis Description

Command used to copy the dataset from Local to HDFS:

```
hadoop fs -copyFromLocal /home/ujjvalthakkar/Desktop/InputFiles/H1BData/h1b_kaggle.csv  
/user/ujjvalthakkar/ProjectFiles/DataSet
```

HDFS Dataset Location: /user/ujjvalthakkar/ProjectFiles/DataSet/h1b_kaggle.csv

1. Number of Full-time and Part-time petitions (Normal MR Job)

We are analyzing the number of petitions that are filed for Part-time and the Full-time positions.

1. In the **H1BSuccessFailureMapper**, we read every petition from the dataset and filter the petitions based on the value of the FULL_TIME_POSITION column. If the value is 'Y' it is a Full Time Position, if the value is 'N' it is a Part Time Position.
2. We set the text key to be FULL TIME or PART TIME and IntWritable value as the count 1.
3. In the **H1BSuccessFailureReducer**, we read all the values for individual keys and count a total sum, which gives us the number of Part-time and Full-time positions

Command to Execute Jar:

```
hadoop jar /home/ujjvalthakkar/eclipse-  
workspace/ADBMS_PROJECT/target/mapreduce_wordcount-0.0.1-SNAPSHOT.jar  
com.neu.h1BAnalysis.H1BFullTimeCount  
/user/ujjvalthakkar/ProjectFiles/DataSet/h1b_kaggle.csv  
/home/ujjvalthakkar/Desktop/ProjectOutputFiles
```

2. Counting number of petitions based on Location (Counting with counters Summarization)

We are analyzing the number of petitions based on locations. This would help us understand where in the United States is the need for most number of H1B employees. Since we are using the counting with counters summarization pattern, there is no reducer and this is a map only job.

1. In the **CountPetitionsPerLocationMapper**, we maintain an array of states **statesArray** and two context counter groups **STATE_COUNTER_GROUP** and **UNKNOWN_COUNTER**.
2. After extracting the location of the petition from the WORKSITE column, we fetch the respective state's counter from the context and increment it.
3. In the driver, upon completion of the mapper, we can fetch the counter and print the number of applications from each state.

```

public static class CountPetitionsPerLocationMapper extends Mapper<LongWritable, Text, NullWritable, NullWritable> {
    public static final String STATE_COUNTER_GROUP = "State";
    public static final String UNKNOWN_COUNTER = "Unknown";

    String[] statesArray = new String[] { "CALIFORNIA", "ALABAMA", "ARKANSAS", "ARIZONA", "ALASKA", "COLORADO",
        "CONNECTICUT", "DELAWARE", "FLORIDA", "GEORGIA", "HAWAII", "IDAHO", "ILLINOIS", "INDIANA", "IOWA",
        "KANSAS", "KENTUCKY", "LOUISIANA", "MAINE", "MARYLAND", "MASSACHUSETTS", "MICHIGAN", "MINNESOTA",
        "MISSISSIPPI", "MISSOURI", "MONTANA", "NEBRASKA", "NEVADA", "NEW HAMPSHIRE", "NEW JERSEY", "NEW MEXICO",
        "NEW YORK", "NORTH CAROLINA", "NORTH DAKOTA", "OHIO", "OKLAHOMA", "OREGON", "PENNSYLVANIA",
        "RHODE ISLAND", "SOUTH CAROLINA", "SOUTH DAKOTA", "TENNESSEE", "TEXAS", "UTAH", "VERMONT", "VIRGINIA",
        "WASHINGTON", "WEST VIRGINIA", "WISCONSIN", "WYOMING" };
    HashSet<String> states = new HashSet<String>(Arrays.asList(statesArray));

    @Override
    public void map(LongWritable key, Text line, Context context) throws IOException, InterruptedException {
        String[] values = line.toString().split("(?=[^\\s]*\\s*[^\\s]*\\s*)");
        if (values == null) {
            return;
        }

        String state = values[8].substring(values[8].indexOf(',') + 2, values[8].length() - 1);

        if (state != null && !state.isEmpty() && !state.equalsIgnoreCase("WORKSITE")) {
            boolean unknown = true;

```

Command to Execute Jar:

```

hadoop jar /home/ujjvalthakkar/eclipse-workspace/ADBMS_PROJECT/target/mapreduce_wordcount-0.0.1-SNAPSHOT.jar com.neu.h1BAnalysis.H1BPetitionCountPerLocation
/user/ujjvalthakkar/ProjectFiles/DataSet/h1b_kaggle.csv
/home/ujjvalthakkar/Desktop/ProjectOutputFiles

```

3. Predictive yearly analysis of successful certification of the petitions (Numerical Summarization, Mean, Standard Deviation, MR Chaining)

We analyze the dataset to find out the mean, standard deviation of the number of successful petitions for every year and try to predict the possible number of successful petitions for the upcoming year. Once, the standard deviation is found, we can then add/subtract that value to the last year's success count and predict the possible number of successful petitions.

We chain the MR jobs here. We find out the number of successful predictions for each year using the first job and then use these values to find the mean, standard deviation of the successful counts to predict the further success in the second job.

1. In the **H1BPetitionSucessMapper**, we filter the petitions based on the **CASE_STATUS** to be **CERTIFIED** or **CERTIFIED-WITHDRAWN** and then extract the year of that petition as the key and the value is set as the count 1.
2. In the **H1BPetitionSucessReducer**, we find the sum of the successful petitions for each year and write the output via the context writer.
3. In the second job, we use the **H1BPetitionSucessPredictiveMapper** to pass the input values to the reducer
4. In the **H1BPetitionSucessPredictiveReducer**, we find the mean, standard deviation and then predict the upcoming success as described above.

Command to Execute Jar:

```
hadoop jar /home/ujjvalthakkar/eclipse-  
workspace/ADBMS_PROJECT/target/mapreduce_wordcount-0.0.1-SNAPSHOT.jar  
com.neu.h1BAnalysis.H1BSuccessfulPetitionPrediction  
/user/ujjvalthakkar/ProjectFiles/DataSet/h1b_kaggle.csv  
/home/ujjvalthakkar/Desktop/ProjectOutputFiles
```

4. Top 10 employers to file the most petitions (Top N Filtering Pattern, Partitioner)

We analyze the dataset using the Top N Filtering pattern to find out the top 10 employers to file the most petitions during the period of 2011-2016 and this helps us to know about the employers who hire most foreign nationals.

1. In the **H1BAppInsPerEmployeePerYearMapper**, we filter the employee name and the year of the application and pass it to the reducer along with the count 1.

```
public static class H1BAppInsPerEmployeePerYearReducer extends Reducer<Text, LongWritable, NullWritable, Text> {  
    private TreeMap<LongWritable, Text> topEmployers = new TreeMap<LongWritable, Text>();  
    long sum = 0;  
  
    public void reduce(Text key, Iterable<LongWritable> values, Context context)  
        throws IOException, InterruptedException {  
        sum = 0;  
        for (LongWritable val : values) {  
            sum += val.get();  
        }  
        topEmployers.put(new LongWritable(sum), new Text(key + "," + sum));  
        if (topEmployers.size() > 10)  
            topEmployers.remove(topEmployers.firstKey());  
    }  
  
    protected void cleanup(Context context) throws IOException, InterruptedException {  
        for (Text t : topEmployers.descendingMap().values())  
            context.write(t, NullWritable.get());  
    }  
}
```

2. We
use the

CustomPartitioner, to partition the petitions according to the year.

3. In the **H1BAppInsPerEmployeePerYearReducer**, we find the sum of the number of petitions for the current employer and use an in-memory list TreeMap to store the sum, the name of the employer. We keep a check on the size of the in-memory list, if it happens to be greater than 10, we remove the first key.
4. On cleanup, we then write the output of the in-memory list via the context writer.
5. Output are 7 part files containing the top 10 employers for each year, one for each year from 2011-2016.

Command to Execute Jar:

```
hadoop jar /home/ujjvalthakkar/eclipse-workspace/ADBMS_PROJECT/target/mapreduce_wordcount-  
0.0.1-SNAPSHOT.jar com.neu.h1BAnalysis.H1BTop10AppInsPerEmployeePerYear  
/user/ujjvalthakkar/ProjectFiles/DataSet/h1b_kaggle.csv  
/home/ujjvalthakkar/Desktop/ProjectOutputFiles
```

5. Most Data Engineer jobs for each year for each Location (Top N Filtering Pattern, Partitioner)

We analyze the dataset using the Top N Filtering pattern to find out the top 10 locations filing petitions for a Data Engineer every year. This analysis would help the Data Engineers to target these locations to find a job sponsoring the H1B visa for them.

1. In the ***H1BAppInsPerEmployeePerYearMapper***, we filter the location and Year of the data engineer petitions and pass it to the reducer along with the count 1.
2. We use the ***CustomPartitioner***, to partition the petitions according to the year.
3. In the ***H1BAppInsPerEmployeePerYearReducer***, we find the sum of the number of petitions and use an in-memory list TreeMap to store the sum, the location. We keep a check on the size of the in-memory list, if it happens to be greater than 10, we remove the first key.
4. On cleanup, we then write the output of the in-memory list via the context writer.
5. Output are 7 part files containing the top 10 locations for each year containing the maximum number of Data engineer petitions, one for each year from 2011-2016.

Command to Execute Jar:

```
hadoop jar /home/ujjvalthakkar/eclipse-workspace/ADBMS_PROJECT/target/mapreduce_wordcount-0.0.1-SNAPSHOT.jar com.neu.h1BAnalysis.H1BTop10DataEngineerPerYear  
/user/ujjvalthakkar/ProjectFiles/DataSet/h1b_kaggle.csv  
/home/ujjvalthakkar/Desktop/ProjectOutputFiles
```


6. Categorizing petitions filed for a "CEO" for each year (Binning Data Organization Pattern)

We analyze the dataset using the Binning Organization pattern to categorize the petitions filed for a CEO of the company. This analysis helps us to find out the growing entrepreneur sector and the growing changes in the existing organizations. Since we use this pattern, this is a map only job and does not contain a reducer.

1. In the ***H1BCEOPetitionsPerYearMapper***, we filter out the petitions filed for a CEO using the JOB_TITLE column and then organize the data into multiple bins depending on the year of the petition.
2. We use MultipleOutputs for this organization.

Command to Execute Jar:

```
hadoop jar /home/ujjvalthakkar/eclipse-  
workspace/ADBMS_PROJECT/target/mapreduce_wordcount-0.0.1-SNAPSHOT.jar  
com.neu.h1BAnalysis.H1BCEOPetitionsPerYear
```

```
job.setMapOutputKeyClass(Text.class);  
job.setMapOutputValueClass(IntWritable.class);  
  
MultipleOutputs.addNamedOutput(job, "bins", TextOutputFormat.class, Text.class, NullWritable.class);  
MultipleOutputs.setCountersEnabled(job, true);  
  
FileInputFormat.addInputPath(job, input);  
FileOutputFormat.setOutputPath(job, outputDir);  
  
// Delete output if exists  
FileSystem hdfs = FileSystem.get(conf);  
if (hdfs.exists(outputDir))  
    hdfs.delete(outputDir, true);  
  
int code = job.waitForCompletion(true) ? 0 : 1;  
  
System.exit(code);  
  
}  
  
public static class H1BCEOPetitionsPerYearMapper extends Mapper<Object, Text, Text, NullWritable> {  
    private MultipleOutputs<Text, NullWritable> mos = null;  
  
    protected void setup(Context context) {  
        mos = new MultipleOutputs<Text, NullWritable>(context);  
    }  
  
    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {  
        String[] values = value.toString().split("(?=[^\\\"]*\\\"[^\"]*\\\"|\".*\")");  
        if (values == null) {  
            return;  
        }  
  
        if (!values[7].equalsIgnoreCase("YEAR")) {  
            if (values[4].substring(1, values[4].length()-1).equalsIgnoreCase("CEO") || values[4].substring(1, values[4].length()-1).equalsIgnoreCase("CHIEF EXECUTIVE OFFICER") || v  
                if (values[7].equalsIgnoreCase("2011")) {  
                    mos.write("bins", value, NullWritable.get(), "Year-2011");  
                } else if (values[7].equalsIgnoreCase("2012")) {  
                    mos.write("bins", value, NullWritable.get(), "Year-2012");  
                } else if (values[7].equalsIgnoreCase("2013")) {  
                    mos.write("bins", value, NullWritable.get(), "Year-2013");  
                } else if (values[7].equalsIgnoreCase("2014")) {  
                    mos.write("bins", value, NullWritable.get(), "Year-2014");  
                }  
            }  
        }  
    }  
}
```

```
/user/ujjvalthakkar/ProjectFiles/DataSet/h1b_kaggle.csv  
/home/ujjvalthakkar/Desktop/ProjectOutputFiles
```

7. Successful Petitions based on the Wages (Inverted Index Pattern)

We analyze the dataset using the Inverted Index Summarization pattern to find out the number of successful and unsuccessful petitions filed for different wage ranges of employee and can be useful for understanding how likely would the visa petition be certified for a particular wage range.

1. In the **H1BWageRangeSummarizationMapper**, we filter the successful and unsuccessful petitions and compare their wage ranges and summarize them into different wage groups.
2. In the **H1BWageRangeSummarizationReducer**, we calculate the count for each wage group and write the output using the context writer.

Command to Execute Jar:

```
hadoop jar /home/ujjvalthakkar/eclipse-workspace/ADBMS_PROJECT/target/mapreduce_wordcount-0.0.1-SNAPSHOT.jar com.neu.h1BAnalysis.H1BWageRangeSummarization  
/user/ujjvalthakkar/ProjectFiles/DataSet/h1b_kaggle.csv /home/ujjvalthakkar/Desktop/ProjectOutputFiles
```

```
    if (case_status.equalsIgnoreCase("\\"CERTIFIED-WITHDRAWN\\"")) {  
        double employeeWage = Double.parseDouble(values[6]);  
        // jobtitle_outValue.set(job_title);  
        if (employeeWage >= 0 && employeeWage <= 30000) {  
            wageRange.set("0-30000 Success Range");  
        } else if (employeeWage > 30000 && employeeWage <= 60000) {  
            wageRange.set("30000-60000 Success Range");  
        } else if (employeeWage > 60000 && employeeWage <= 70000) {  
            wageRange.set("60000-70000 Success Range");  
        } else if (employeeWage > 70000 && employeeWage <= 80000) {  
            wageRange.set("70000-80000 Success Range");  
        } else if (employeeWage > 80000 && employeeWage <= 90000) {  
            wageRange.set("80000-90000 Success Range");  
        } else if (employeeWage > 90000 && employeeWage <= 100000) {  
            wageRange.set("90000-100000 Success Range");  
        } else if (employeeWage > 100000) {  
            wageRange.set("100000- Above Success Range");  
        }  
    } else {  
        if (!values[6].equals("NA")) {  
            double employeeWage = Double.parseDouble(values[6]);  
            // jobtitle_outValue.set(job_title);  
            if (employeeWage >= 0 && employeeWage <= 30000) {  
                wageRange.set("0-30000 Failure Range");  
            } else if (employeeWage > 30000 && employeeWage <= 60000) {  
                wageRange.set("30000-60000 Failure Range");  
            } else if (employeeWage > 60000 && employeeWage <= 70000) {  
                wageRange.set("60000-70000 Failure Range");  
            } else if (employeeWage > 70000 && employeeWage <= 80000) {  
                wageRange.set("70000-80000 Failure Range");  
            } else if (employeeWage > 80000 && employeeWage <= 90000) {  
                wageRange.set("80000-90000 Failure Range");  
            } else if (employeeWage > 90000 && employeeWage <= 100000) {  
                wageRange.set("90000-100000 Failure Range");  
            } else if (employeeWage > 100000) {  
                wageRange.set("100000- Above Failure Range");  
            }  
        }  
    }  
    context.write(wageRange, ONE);  
}  
}  
}  
  
public static class H1BWageRangeSummarizationReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
```

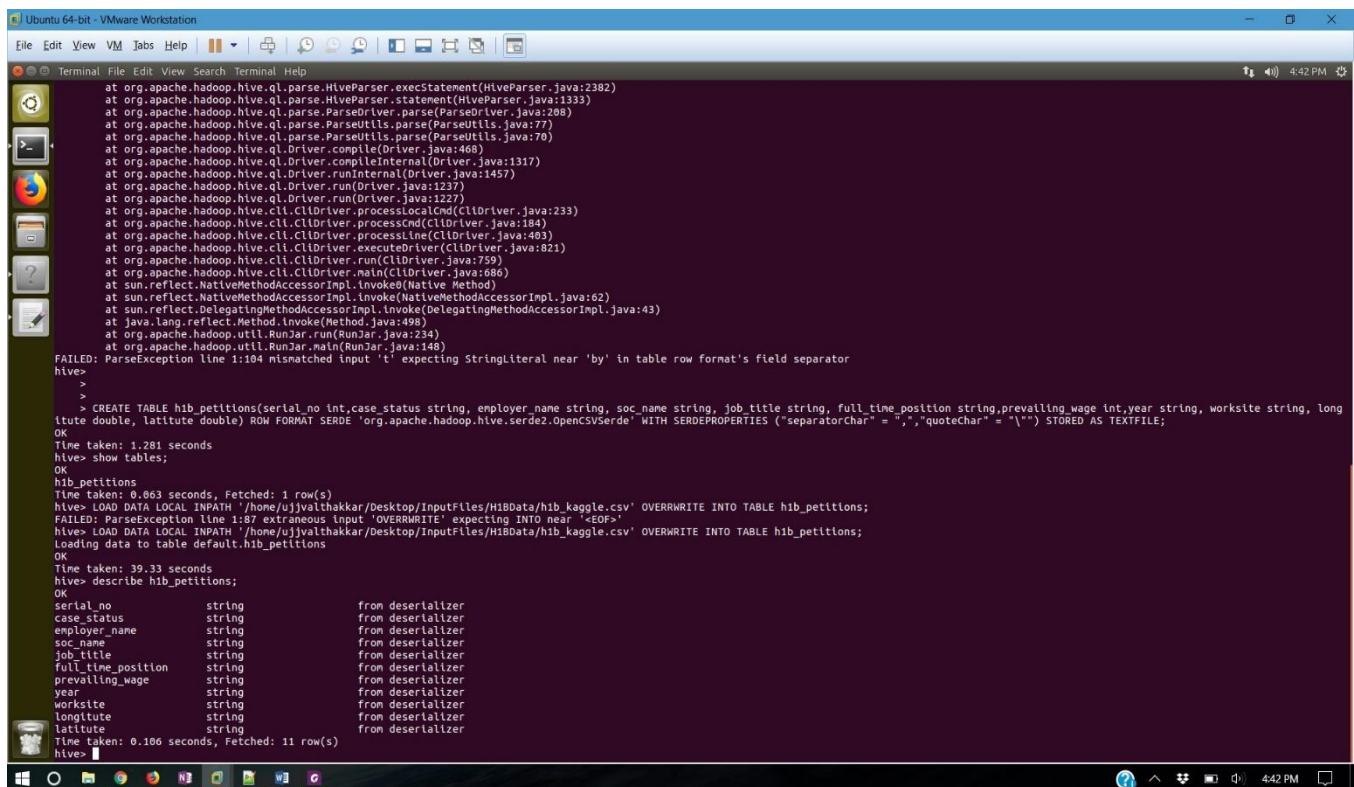
HIVE ANALYSIS

Table Creation

CREATE TABLE h1b_petitions(serial_no int,case_status string, employer_name string, soc_name string, job_title string, full_time_position string,prevailing_wage int,year string, worksite string, longitude double, latitude double) ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde' WITH SERDEPROPERTIES ("separatorChar" = ",", "quoteChar" = "\"") STORED AS TEXTFILE;

Loading the data into the table

LOAD DATA LOCAL INPATH '/home/ujjvalthakkar/Desktop/InputFiles/H1BData/h1b_kaggle.csv' OVERWRITE INTO TABLE h1b_petitions;



```
at org.apache.hadoop.hive ql.parse.HiveParser.execStatement(HiveParser.java:2382)
at org.apache.hadoop.hive ql.parse.HiveParser.statement(HiveParser.java:1333)
at org.apache.hadoop.hive ql.parse.ParseDriver.parse(ParseDriver.java:208)
at org.apache.hadoop.hive ql.parse.ParseUtils.parse(ParseUtils.java:77)
at org.apache.hadoop.hive ql.parse.ParseUtils.parse(ParseUtils.java:70)
at org.apache.hadoop.hive ql.Driver.compile(Driver.java:468)
at org.apache.hadoop.hive ql.Driver.compileInternal(Driver.java:1317)
at org.apache.hadoop.hive ql.Driver.runInternal(Driver.java:1457)
at org.apache.hadoop.hive ql.Driver.run(Driver.java:1227)
at org.apache.hadoop.hive ql.Driver.run(Driver.java:1227)
at org.apache.hadoop.hive.cll.CliDriver.processLocalCmd(CliDriver.java:233)
at org.apache.hadoop.hive.cll.CliDriver.processCmd(CliDriver.java:184)
at org.apache.hadoop.hive.cll.CliDriver.processLine(CliDriver.java:403)
at org.apache.hadoop.hive.cll.CliDriver.executeDriver(CliDriver.java:821)
at org.apache.hadoop.hive.cll.CliDriver.run(CliDriver.java:759)
at org.apache.hadoop.hive.cll.CliDriver.main(CliDriver.java:686)
at sun.reflect.NativeMethodAccessorImpl.invoke(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.lang.reflect.Method.invoke(Method.java:498)
at org.apache.hadoop.util.RunJar.run(RunJar.java:234)
at org.apache.hadoop.util.RunJar.main(RunJar.java:148)
FAILED: ParseException line 1:104 mismatched input 't' expecting StringLiteral near 'by' in table row format's field separator
hive>
>
> CREATE TABLE h1b_petitions(serial_no int,case_status string, employer_name string, soc_name string, job_title string, full_time_position string,prevailing_wage int,year string, worksite string, longitude double, latitude double) ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde' WITH SERDEPROPERTIES ("separatorChar" = ",", "quoteChar" = "\"") STORED AS TEXTFILE;
OK
Time taken: 1.281 seconds
hive> show tables;
OK
h1b_petitions
Time taken: 0.063 seconds, Fetched: 1 row(s)
hive> LOAD DATA LOCAL INPATH '/home/ujjvalthakkar/Desktop/InputFiles/H1BData/h1b_kaggle.csv' OVERWRITE INTO TABLE h1b_petitions;
FAILED: ParseException line 1:87 extraneous input 'OVERWRITE' expecting INTO near '<EOF>'
hive> LOAD DATA LOCAL INPATH '/home/ujjvalthakkar/Desktop/InputFiles/H1BData/h1b_kaggle.csv' OVERWRITE INTO TABLE h1b_petitions;
Loading data to table default.h1b_petitions
OK
Time taken: 39.33 seconds
hive> describe h1b_petitions;
OK
serial_no      string      from deserializer
case_status    string      from deserializer
employer_name  string      from deserializer
soc_name       string      from deserializer
job_title      string      from deserializer
full_time_position string    from deserializer
prevailing_wage string      from deserializer
year           string      from deserializer
worksite       string      from deserializer
longitude       string      from deserializer
latitude       string      from deserializer
Time taken: 0.106 seconds, Fetched: 11 row(s)
hive>
```

8. Most Data Scientist positions by Industry (HIVE)

Hive Query:

```
SELECT SOC_NAME, COUNT(JOB_TITLE) AS POSITIONCOUNT FROM H1B_FINAL WHERE JOB_TITLE == 'DATA SCIENTIST' GROUP BY SOC_NAME ORDER BY POSITIONCOUNT DESC;
```

Output:

SOC_NAME	POSITIONCOUNT
STATISTICIANS	415
COMPUTER AND INFORMATION RESEARCH SCIENTISTS	333
OPERATIONS RESEARCH ANALYSTS	253
Computer and Information Research Scientists	134
COMPUTER OCCUPATIONS, ALL OTHER	121
MATHEMATICIANS	108
Statisticians	104
SOFTWARE DEVELOPERS, APPLICATIONS	79
COMPUTER SYSTEMS ANALYSTS	66
Operations Research Analysts	64
Software Developers, Applications	38
SOFTWARE DEVELOPERS, SYSTEMS SOFTWARE	37
Computer Occupations, All Other	23
Computer Systems Analysts	21
Software Developers, Systems Software	12
FINANCIAL SPECIALISTS, ALL OTHER	11
Mathematicians	10
Database Administrators	8
MATHEMATICAL TECHNICIANS	7
MARKET RESEARCH ANALYSTS AND MARKETING SPECIALISTS	7
DATABASE ADMINISTRATORS	6
BIOLOGICAL SCIENTISTS, ALL OTHER	6
COMPUTER PROGRAMMERS	5
ECONOMISTS	5
COMPUTER NETWORK ARCHITECTS	3
COMPUTER AND INFORMATION SYSTEMS MANAGERS	3
INDUSTRIAL-ORGANIZATIONAL PSYCHOLOGISTS	3
ENGINEERS, ALL OTHER	2
STATISTICAL ASSISTANTS	2
SALES ENGINEERS	2
Computer Programmers	2
Economists	2
ELECTRICAL ENGINEERS	2
Management Analysts	2
SURVEY RESEARCHERS	2
Biological Scientists, All Other	2
Survey Researchers	1

SOFTWARE DEVELOPERS, APPLICATIONS, R&D 1
 Materials Scientists 1
 Market Research Analysts and Marketing Specialists 1
 MATERIALS SCIENTISTS 1
 Engineers, All Other 1
 Computer Software Engineers, Applications 1
 COMPUTER SYSTEMS ANALYST 1
 COMPUTER & INFORMATION RESEARCH SCIENTISTS 1
 CLINICAL DATA MANAGERS 1
 CIVIL ENGINEERS 1
 ACTUARIES 1
 MEDICAL SCIENTISTS, EXCEPT EPIDEMIOLOGISTS 1
 MATERIALS ENGINEERS 1
 MANAGEMENT ANALYSTS 1
 Life Scientists, All Other 1
 Financial Analysts 1
 Electrical Engineers 1
 Credit Analysts 1
 Computer and Information Scientists, Research 1
 COMPUTER OCCUPATIONS, ALL OTHER*1
 COMPUTER & INFORMATION RESEARCH SCIENTIST 1
 BUSINESS INTELLIGENCE ANALYSTS 1

```

> select soc_name,count(job_title) as PositionCount from H1B_petitions where job_title == 'DATA SCIENTIST' and soc_name!='NA' group by soc_name order by PositionCount desc;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = ujyvalthakkar_20171210165713_f9a3e39c-3d33-4bcd-b847-c65b5b76c1ec
Total Jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 2
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1512940697648_0003, Tracking URL = http://ubuntu:8088/proxy/application_1512940697648_0003/
Kill Command = /usr/local/bin/hadoop-2.8.2/bin/hadoop job -kill job_1512940697648_0003
Hadoop job information for Stage-1: number of mappers: 2; number of reducers: 2
2017-12-10 16:57:19,896 Stage-1 map = 0%, reduce = 0%, Cumulative CPU 34.19 sec
2017-12-10 16:57:44,614 Stage-1 map = 37%, reduce = 0%, Cumulative CPU 41.92 sec
2017-12-10 16:57:50,769 Stage-1 map = 55%, reduce = 0%, Cumulative CPU 43.21 sec
2017-12-10 16:57:59,454 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 46.98 sec
2017-12-10 16:58:09,640 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 52.46 sec
MapReduce Total cumulative CPU time: 52 seconds 460 msec
Ended Job = job_1512940697648_0003
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1512940697648_0004, Tracking URL = http://ubuntu:8088/proxy/application_1512940697648_0004/
Kill Command = /usr/local/bin/hadoop-2.8.2/bin/hadoop job -kill job_1512940697648_0004
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2017-12-10 16:58:22,497 Stage-2 map = 0%, reduce = 0%, Cumulative CPU 1.33 sec
2017-12-10 16:58:34,100 Stage-2 map = 100%, reduce = 100%, Cumulative CPU 3.81 sec
  
```

9. Top 10 Most petitions filed by Employers by Year (HIVE)

Hive Query:

```
SELECT EMPLOYER_NAME, YEAR, COUNT(EMPLOYER_NAME) AS PETITIONCOUNT FROM  
H1B_PETITIONS GROUP BY EMPLOYER_NAME, YEAR ORDER BY PETITIONCOUNT DESC LIMIT 10;
```

Output:

EMPLOYER_NAME	YEAR	PETITIONCOUNT
INFOSYS LIMITED	2015	33245
INFOSYS LIMITED	2013	32223
INFOSYS LIMITED	2016	25352
INFOSYS LIMITED	2014	23759
CAPGEMINI AMERICA INC	2016	16725
TATA CONSULTANCY SERVICES LIMITED	2015	16553
INFOSYS LIMITED	2012	15818
TATA CONSULTANCY SERVICES LIMITED	2014	14098
TATA CONSULTANCY SERVICES LIMITED	2016	13134
WIPRO LIMITED	2015	12201

10. Counting number of petitions based on Location (HIVE)

Hive Query:

```
SELECT count(SUBSTR(WORKSITE,(INSTR(WORKSITE,"")+2),length(WORKSITE))),  
SUBSTR(WORKSITE,(INSTR(WORKSITE,"")+2),length(WORKSITE)) FROM H1B_PETITIONS GROUP BY  
(SUBSTR(WORKSITE,(INSTR(WORKSITE,"")+2),length(WORKSITE)));
```

Output:

42054	ARIZONA
14762	ARKANSAS
559941	CALIFORNIA
103168	GEORGIA
30516	INDIANA
16743	IOWA
12702	KENTUCKY
3918	MAINE
56257	MARYLAND
116466	MASSACHUSETTS
82964	MICHIGAN
48686	MINNESOTA
986	MONTANA
214076	NEW JERSEY
79672	NORTH CAROLINA
10597	OKLAHOMA

23790 OREGON
109960 PENNSYLVANIA
1452 PUERTO RICO
12932 SOUTH CAROLINA
27673 TENNESSEE
13107 UTAH
102801 WASHINGTON
3079 WEST VIRGINIA
896 WYOMING
10253 ALABAMA
1396 ALASKA
32038 COLORADO
50253 CONNECTICUT
18138 DELAWARE
22708 DISTRICT OF COLUMBIA
105773 FLORIDA
3752 HAWAII
4191 IDAHO
160814 ILLINOIS
13782 KANSAS
11669 LOUISIANA
4270 MISSISSIPPI
34813 MISSOURI
3604 NA
9256 NEBRASKA
7448 NEVADA
9929 NEW HAMPSHIRE
5541 NEW MEXICO
291411 NEW YORK
2927 NORTH DAKOTA
76575 OHIO
1 ORKSITE
11715 RHODE ISLAND
1902 SOUTH DAKOTA
294454 TEXAS
1941 VERMONT
90028 VIRGINIA
32672 WISCONSIN

11. Categorizing petitions filed for a "CEO" for each year (HIVE)

Hive Query:

```
SELECT Year,Count(year) from H1B_PETITIONS WHERE job_title="CEO" OR job_title="CHIEF  
EXECUTIVE OFFICER" or job_title="CHIEF EXECUTIVE OFFICER" group by year;
```

Output:

```
2011  167  
2013  191  
2015  247  
2012  275  
2014  212  
2016  209
```

12. Counting of Number of Full time and Part time petitions (HIVE)

Hive Query:

```
SELECT COUNT(*) FROM H1B_PETITIONS WHERE FULL_TIME_POSITION="Y" OR  
FULL_TIME_POSITION="Y";
```

Output:

```
2576104
```

13. Top 10 Locations with most DATA ENGINEER petitions

Hive Query:

```
SELECT SPLIT(WORKSITE,[','])[1] AS STATE, COUNT(SPLIT(WORKSITE,[','])[1]) AS JOB_COUNT FROM  
H1B_PETITIONS WHERE JOB_TITLE LIKE='DATA ENGINEER' GROUP BY SPLIT(WORKSITE,[','])[1] ORDER  
BY JOB_COUNT DESC LIMIT 10;
```

Output:

```
CALIFORNIA  323  
NEW YORK    88  
WASHINGTON 24  
MASSACHUSETTS 24  
ILLINOIS    20  
MICHIGAN    16  
TEXAS       15  
VIRGINIA    11  
NEW JERSEY  9  
FLORIDA     8
```


PIG ANALYSIS

Loading the data

```
REGISTER /home/ujjvalthakkar/Jars/piggybank-0.11.0.jar;
```

```
DEFINE CSV_Storage org.apache.pig.piggybank.storage.CSVExcelStorage();
```

```
h1b_petitions = load '/home/ujjvalthakkar/Desktop/InputFiles/H1BData/h1b_kaggle.csv' using  
CSV_Storage() as (serial_no: int,case_status: chararray, employer_name: chararray, soc_name:  
chararray, job_title: chararray, fulltime_position: chararray,prevailing_wage: int,year: chararray,  
worksite: chararray, longitude: double, latitude: double);
```

14. Top Data Engineer Locations for each year (PIG)

Script:

```
filteredGroup = foreach h1b_petitions generate job_title,worksite,year;  
dataEngineerPetitions = FILTER filteredGroup BY job_title == 'DATA ENGINEER';  
dataEngineerPetitions_AllYears= FILTER dataEngineerPetitions BY (year == '2011') OR (year == '2012')  
OR (year == '2013') OR (year == '2014')OR (year == '2015')OR (year == '2016');  
dataEngineerPetitions_AllYearsGrouped = group dataEngineerPetitions_AllYears by  
(job_title,worksite,year);  
dataEngineerPetitionsWithJobTitleCount_AllYears = foreach dataEngineerPetitions_AllYearsGrouped  
generate group,COUNT(dataEngineerPetitions_AllYears.job_title);  
countArrangedDescByWorksite_AllYears = order dataEngineerPetitionsWithJobTitleCount_AllYears  
by $1 DESC;
```

```
year2016 = FILTER countArrangedDescByWorksite_AllYears BY $0.year=='2016';  
year2015 = FILTER countArrangedDescByWorksite_AllYears BY $0.year=='2015';  
year2014 = FILTER countArrangedDescByWorksite_AllYears BY $0.year=='2014';  
year2013 = FILTER countArrangedDescByWorksite_AllYears BY $0.year=='2013';  
year2012 = FILTER countArrangedDescByWorksite_AllYears BY $0.year=='2012';  
year2011 = FILTER countArrangedDescByWorksite_AllYears BY $0.year=='2011';
```

```
year2016 = LIMIT year2016 1;  
year2015 = LIMIT year2015 1;  
year2014 = LIMIT year2014 1;  
year2013 = LIMIT year2013 1;  
year2012 = LIMIT year2012 1;
```

```

year2011 = LIMIT year2011 1;
h1b_topDataEngineerCountLocationEachYear = UNION
year2016,year2015,year2014,year2013,year2012,year2011;
dump h1b_topDataEngineerCountLocationEachYear;

```

Output:

```

((DATA ENGINEER,MENLO PARK, CALIFORNIA,2014),13)
((DATA ENGINEER,SAN FRANCISCO, CALIFORNIA,2015),33)
((DATA ENGINEER,SAN FRANCISCO, CALIFORNIA,2012),7)
((DATA ENGINEER,MENLO PARK, CALIFORNIA,2016),35)
((DATA ENGINEER,SAN FRANCISCO, CALIFORNIA,2011),3)
((DATA ENGINEER,MENLO PARK, CALIFORNIA,2013),10)

```

```

$ ujjvalthakkar@ubuntu:~$ pig -x local
17/12/11 20:12:57 INFO pig.ExecTypeProvider: Trying ExecType: LOCAL
17/12/11 20:12:57 INFO pig.ExecTypeProvider: Picked LOCAL as the ExecType
2017-12-11 20:12:57,411 [main] INFO org.apache.pig.Main - Apache Pig version 0.16.0 (r1746538) compiled Jun 01 2016, 23:10:49
2017-12-11 20:12:57,411 [main] INFO org.apache.pig.Main - Logging error messages to: /home/ujjvalthakkar/pig/151394117710.log
2017-12-11 20:12:57,426 [main] INFO org.apache.pig.impl.util.Utils - Default bootstrap file /home/ujjvalthakkar/pigbootstrap not found
2017-12-11 20:12:57,565 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2017-12-11 20:12:57,567 [main] INFO org.apache.pig.backend.hadoop.executionengine.MExecutionEngine - Connecting to Hadoop file system at: file:///
2017-12-11 20:12:57,764 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2017-12-11 20:12:57,780 [main] WARN org.apache.pig.PigServer - PIG Script ID for the session: PIG-default-22287a1a-31c0-4ae1-be0b-d0e367b9d39a
grunt>
grunt>
grunt> REGISTER /home/ujjvalthakkar/2ars/piggybank-0.11.0.jar;
2017-12-11 20:13:06,703 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2017-12-11 20:13:06,703 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
grunt>
grunt> DEFINE CSV_Storage org.apache.pig.piggybank.storage.CSVExcelStorage();
grunt>
grunt> h1b_petitions = load '/home/ujjvalthakkar/Desktop/INPUTFILES/H1BData/h1b_kaggle.csv' using CSV_Storage() as (serial_no: int,case_status: chararray, employer_name: chararray, soc_name: chararray, jo
b_title: chararray, fulltime_position: chararray,prevailing_wage: int,year: chararray, worksite: chararray, longitude: double, latitude: double);
2017-12-11 20:13:26,797 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2017-12-11 20:13:26,798 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
grunt>
grunt> filteredGroup = foreach h1b_petitions generate job_title,worksite,year;
dataEngineerPetitions_AllYears= FILTER dataEngineerPetitions_AllYears BY $0,year== '2011' OR (year == '2012') OR (year == '2013') OR (year == '2014')OR (year == '2015')OR (year == '2016');
dataEngineerPetitions_AllYearsGrouped = group dataEngineerPetitions_AllYears by (job_title,worksite,year);
dataEngineerPetitionsWithJobTitleCount_AllYears = foreach dataEngineerPetitions_AllYearsGrouped generate group,COUNT(dataEngineerPetitions_AllYears.job_title);
countArrangedDescByWorksite_AllYears = order dataEngineerPetitionsWithJobTitleCount_AllYears by $1 DESC;
year2016 = FILTER countArrangedDescByWorksite_AllYears BY $0,year=='2016';
year2015 = FILTER countArrangedDescByWorksite_AllYears BY $0,year=='2015';
year2014 = FILTER countArrangedDescByWorksite_AllYears BY $0,year=='2014';
year2013 = FILTER countArrangedDescByWorksite_AllYears BY $0,year=='2013';
year2012 = FILTER countArrangedDescByWorksite_AllYears BY $0,year=='2012';
year2011 = FILTER countArrangedDescByWorksite_AllYears BY $0,year=='2011';
year2016 = LIMIT year2016 1;
year2015 = LIMIT year2015 1;
year2014 = LIMIT year2014 1;
year2013 = LIMIT year2013 1;
year2012 = LIMIT year2012 1;
year2011 = LIMIT year2011 1;
h1b_topDataEngineerCountLocationEachYear = UNION year2016,year2015,year2014,year2013,year2012,year2011;
grunt>
Job local1839736857.0010 -> Job local958274268.0011,
Job local482336044.0005 -> Job local958274268.0011,
Job local1147774473.0009 -> Job local958274268.0011,
Job local721768482.0008 -> Job local958274268.0011,
Job local811489337.0007 -> Job local958274268.0011,
Job local1783553037.0006 -> Job local958274268.0011,
Job local958274268.0011
2017-12-11 20:20:14,278 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,279 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,279 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,284 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,285 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,285 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,289 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,290 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,290 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,297 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,298 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,299 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,303 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,305 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,305 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,310 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,310 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,316 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,316 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,316 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,320 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,321 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,321 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,323 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,323 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,324 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,324 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,326 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,326 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,328 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,328 [main] INFO org.apache.hadoop.metrics.jvm.JvmMetrics - Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
2017-12-11 20:20:14,332 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreduce.LiteMapReduceLauncher - Success!
2017-12-11 20:20:14,332 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2017-12-11 20:20:14,333 [main] INFO org.apache.pig.data.SchemaUpBackend - SchemaUpBackend has already been initialized
2017-12-11 20:20:14,357 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 6
2017-12-11 20:20:14,357 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapReduceUtil - Total input paths to process : 6
((DATA ENGINEER,MENLO PARK, CALIFORNIA,2014),13)
((DATA ENGINEER,SAN FRANCISCO, CALIFORNIA,2015),33)
((DATA ENGINEER,SAN FRANCISCO, CALIFORNIA,2012),7)
((DATA ENGINEER,MENLO PARK, CALIFORNIA,2016),35)
((DATA ENGINEER,SAN FRANCISCO, CALIFORNIA,2011),3)
((DATA ENGINEER,MENLO PARK, CALIFORNIA,2013),10)
grunt>

```

15. Top10 Industry by most Data Scientist positions (PIG)

Script:

```
filteredDataScientist = filter h1b_petitions by job_title == 'DATA SCIENTIST';
filteredDataScientist1 = foreach filteredDataScientist generate soc_name,job_title;
groupedByIndustry = group filteredDataScientist1 by soc_name;
groupedByIndustryWithCount = foreach groupedByIndustry generate
group,COUNT(filteredDataScientist1.job_title) as totalCount;
orderedGroupedByIndustryWithCount = order groupedByIndustryWithCount by totalCount desc;
top10OrderedGroupedByIndustryWithCount = LIMIT orderedGroupedByIndustryWithCount 10;
dump top10OrderedGroupedByIndustryWithCount;
```

Output:

```
(STATISTICIANS,415)
(COMPUTER AND INFORMATION RESEARCH SCIENTISTS,333)
(OPERATIONS RESEARCH ANALYSTS,253)
(Computer and Information Research Scientists,134)
(COMPUTER OCCUPATIONS, ALL OTHER,121)
(MATHEMATICIANS,108)
(Statisticians,104)
(SOFTWARE DEVELOPERS, APPLICATIONS,79)
(COMPUTER SYSTEMS ANALYSTS,66)
(Operations Research Analysts,64)
```

Visualization and Charts

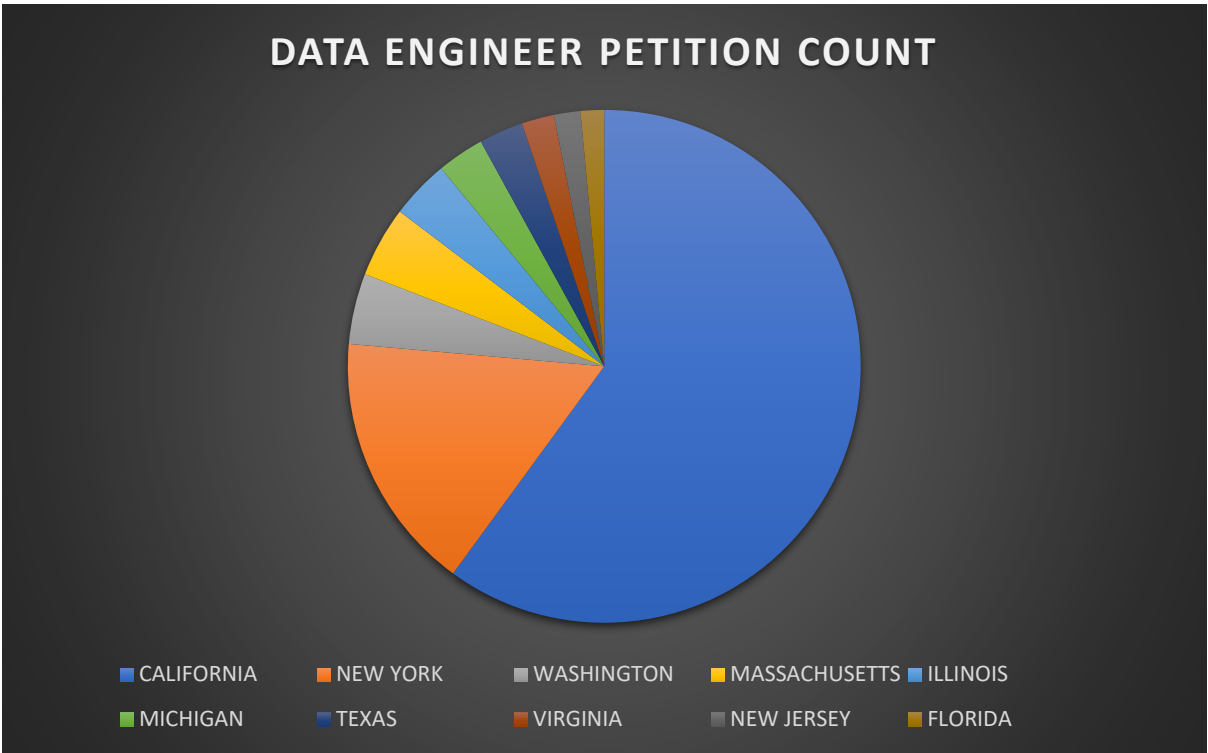


Figure: Top 10 Locations with most DATA ENGINEER petitions

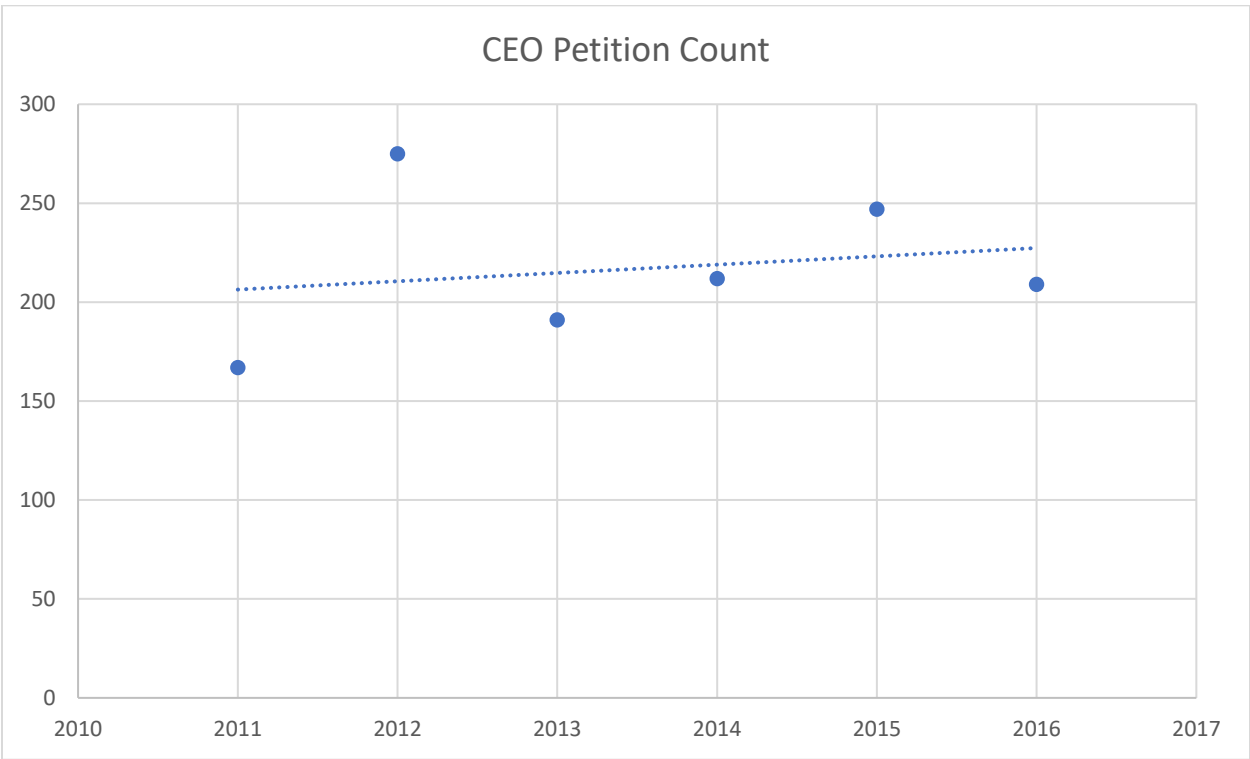


Figure: Categorizing petitions filed for a "CEO" for each year

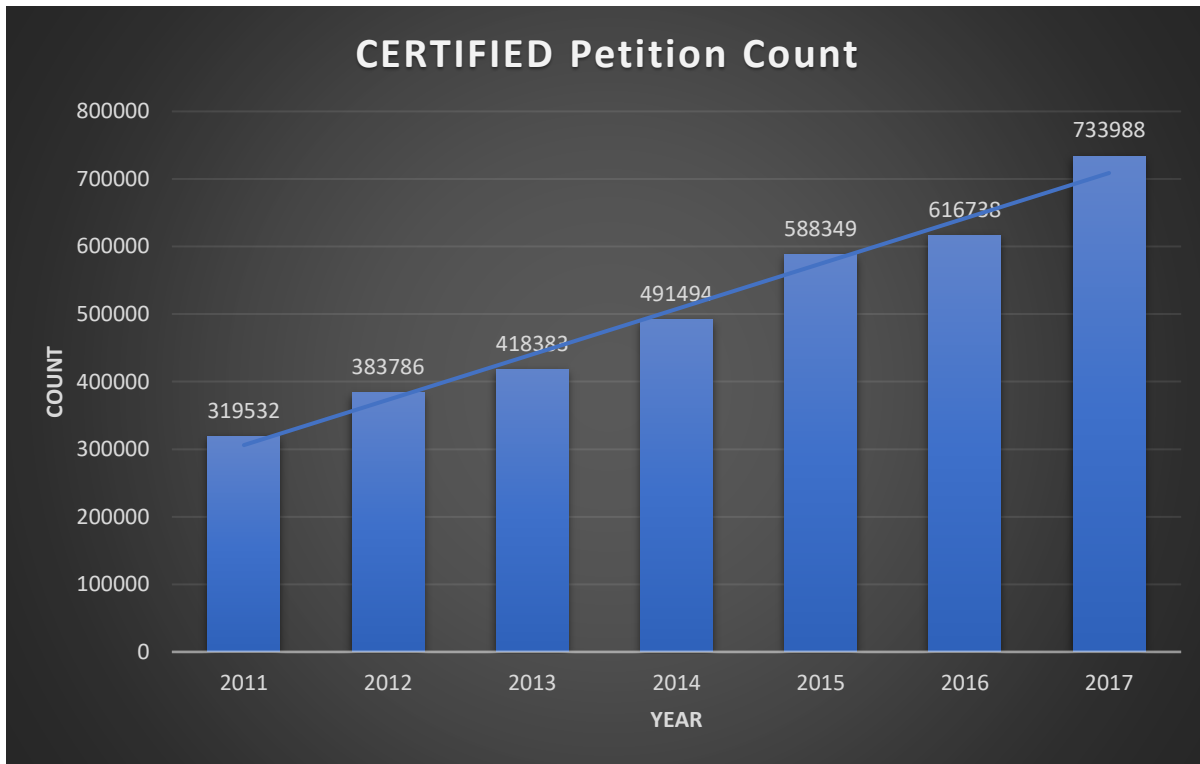


Figure: Predictive yearly analysis of successful certification of the petitions

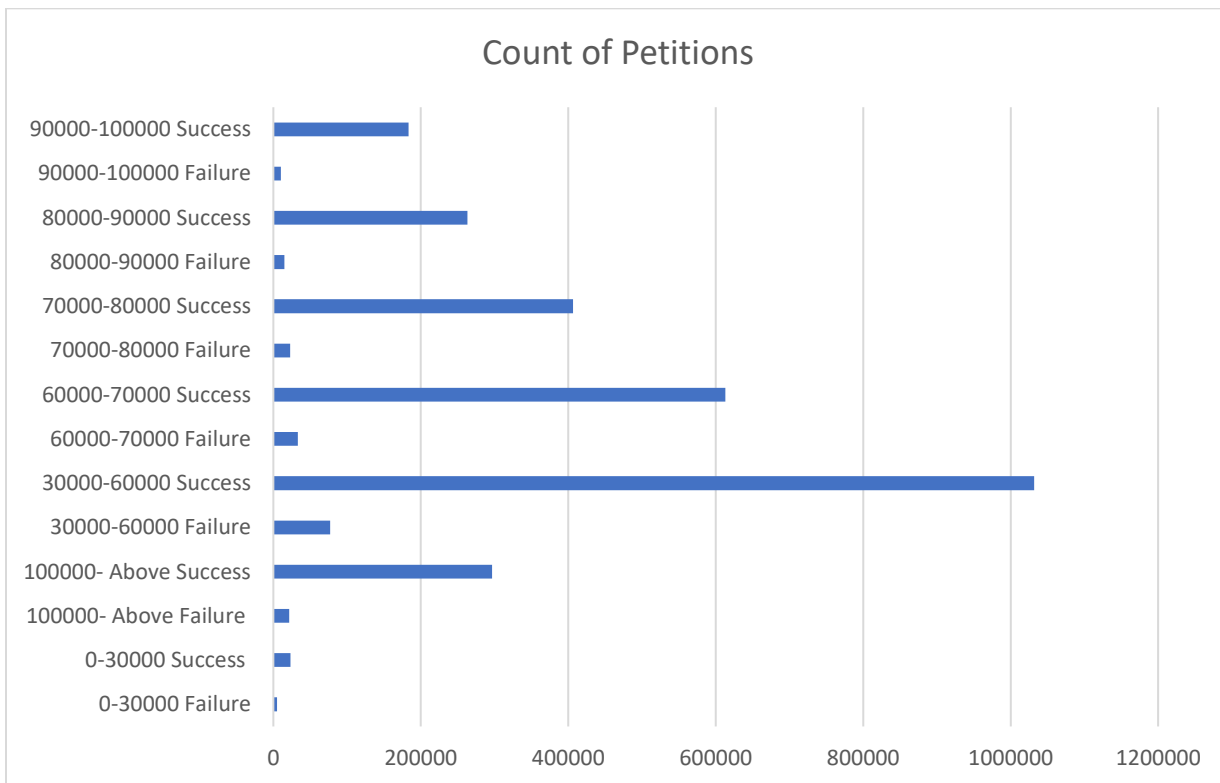


Figure: Successful and Unsuccessful petitions based on the Wage Ranges

I also used the <http://app.rawgraphs.io/> to visualize some of the outputs.

Sample HTML Code for generating the SVG:

```
<!DOCTYPE html>
<html>
<head>
<title>Top10 Industry by most Data Scientist positions </title>
</head>
<body>
<svg width="1200" height="1200" xmlns="http://www.w3.org/2000/svg">
  <g transform="translate(10,10)">
    <g>
      <circle class="node node--root" transform="translate(590,590)" r="590" style="fill-
opacity: 0; stroke: rgb(221, 221, 221); stroke-opacity: 1;"></circle>
      <circle class="node node--leaf"
transform="translate(558.5075487516589,525.8620301548337)" r="199.2135082291827"
style="fill: rgb(191, 105, 105); fill-opacity: 1; stroke: rgb(221, 221, 221); stroke-opacity:
0;"></circle>
      <circle class="node node--leaf"
transform="translate(877.8062200190669,525.8620301548337)" r="120.08516303822536"
style="fill: rgb(191, 156, 105); fill-opacity: 1; stroke: rgb(221, 221, 221); stroke-opacity:
0;"></circle>
      <circle class="node node--leaf"
transform="translate(779.6998382420991,341.57504414686366)" r="88.68877057722851"
style="fill: rgb(174, 191, 105); fill-opacity: 1; stroke: rgb(221, 221, 221); stroke-opacity:
0;"></circle>
      <circle class="node node--leaf"
transform="translate(613.5963985101138,204.9713912860051)" r="126.37147834322454"
style="fill: rgb(122, 191, 105); fill-opacity: 1; stroke: rgb(221, 221, 221); stroke-opacity:
0;"></circle>
      <circle class="node node--leaf"
transform="translate(382.1229406783518,267.70006532706446)" r="113.45105651858101"
style="fill: rgb(105, 191, 139); fill-opacity: 1; stroke: rgb(221, 221, 221); stroke-opacity:
0;"></circle>
      <circle class="node node--leaf"
transform="translate(188.52064995961575,479.6931531363379)" r="173.64285886530925"
style="fill: rgb(105, 191, 191); fill-opacity: 1; stroke: rgb(221, 221, 221); stroke-opacity:
0;"></circle>
      <circle class="node node--leaf"
transform="translate(330.11835277287696,698.9175997964467)" r="87.33466402780026"
style="fill: rgb(105, 139, 191); fill-opacity: 1; stroke: rgb(221, 221, 221); stroke-opacity:
0;"></circle>
```



```

    <circle class="node node--leaf"
transform="translate(476.7963774372216,810.6146493292333)" r="97.03093415625335"
style="fill: rgb(122, 105, 191); fill-opacity: 1; stroke: rgb(221, 221, 221); stroke-opacity:
0;"></circle>
    <circle class="node node--leaf"
transform="translate(788.8123482608656,879.0072909223375)" r="222.3927961348818"
style="fill: rgb(174, 105, 191); fill-opacity: 1; stroke: rgb(221, 221, 221); stroke-opacity:
0;"></circle>
    <circle class="node node--leaf"
transform="translate(1053.7919198829877,676.1355970119864)" r="111.33028902270583"
style="fill: rgb(191, 105, 156); fill-opacity: 1; stroke: rgb(221, 221, 221); stroke-opacity:
0;"></circle>
  </g>
  <g>
    <text text-anchor="middle" transform="translate(590,590)" style="font-size: 11px;
font-family: Arial, Helvetica;"></text>
    <text text-anchor="middle"
transform="translate(558.5075487516589,525.8620301548337)" style="font-size: 11px;
font-family: Arial, Helvetica;">COMPUTER AND INFORMATION RESEARCH SCIENTISTS</text>
    <text text-anchor="middle"
transform="translate(877.8062200190669,525.8620301548337)" style="font-size: 11px;
font-family: Arial, Helvetica;">COMPUTER OCCUPATIONS</text>
    <text text-anchor="middle"
transform="translate(779.6998382420991,341.57504414686366)" style="font-size: 11px;
font-family: Arial, Helvetica;">COMPUTER SYSTEMS ANALYSTS</text>
    <text text-anchor="middle"
transform="translate(613.5963985101138,204.9713912860051)" style="font-size: 11px;
font-family: Arial, Helvetica;">Computer and Information Research Scientists</text>
    <text text-anchor="middle"
transform="translate(382.1229406783518,267.70006532706446)" style="font-size: 11px;
font-family: Arial, Helvetica;">MATHEMATICIANS</text>
    <text text-anchor="middle"
transform="translate(188.52064995961575,479.6931531363379)" style="font-size: 11px;
font-family: Arial, Helvetica;">OPERATIONS RESEARCH ANALYSTS</text>
    <text text-anchor="middle"
transform="translate(330.11835277287696,698.9175997964467)" style="font-size: 11px;
font-family: Arial, Helvetica;">Operations Research Analysts</text>
    <text text-anchor="middle"
transform="translate(476.7963774372216,810.6146493292333)" style="font-size: 11px;
font-family: Arial, Helvetica;">SOFTWARE DEVELOPERS</text>
    <text text-anchor="middle"
transform="translate(788.8123482608656,879.0072909223375)" style="font-size: 11px;
font-family: Arial, Helvetica;">STATISTICIANS</text>

```

```

<text text-anchor="middle"
transform="translate(1053.7919198829877,676.1355970119864)" style="font-size: 11px;
font-family: Arial, Helvetica;">Statisticians</text>
</g>
</g>
</svg>
</body>
</html>

```

OUTPUT:

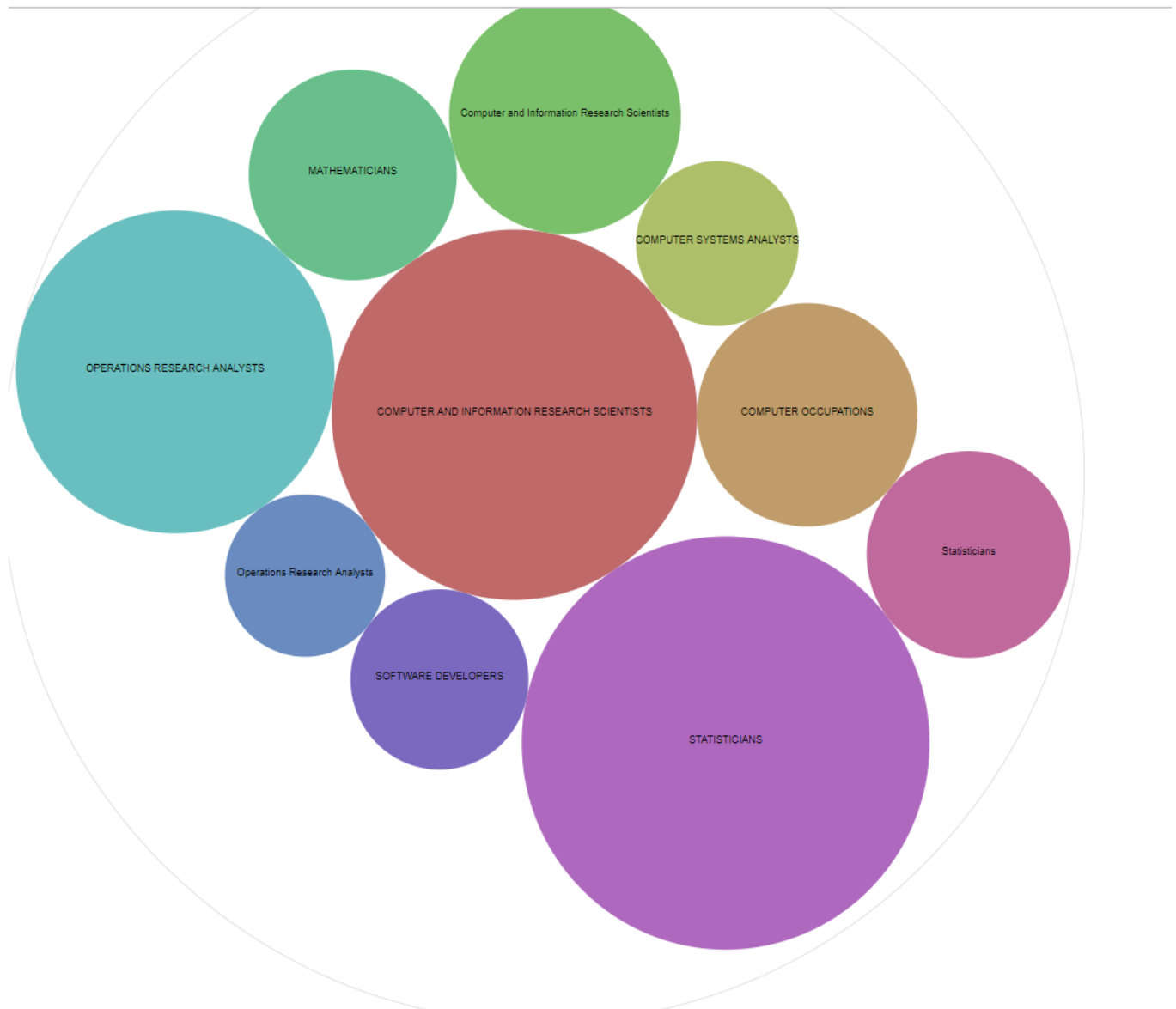


Figure: Top10 Industry by most Data Scientist positions

Appendix

1. Code: Number of Full time and Part time petitions (Normal MR Job)

```
public class H1BFullTimeCount {

    public static void main(String[] args) throws IOException, ClassNotFoundException,
    InterruptedException {
        // TODO Auto-generated method stub

        // Normal Counting of Number of Full time and Part time Petitions

        Configuration conf = new Configuration();

        if (args.length != 2) {
            System.err.println("Usage: H1BDataAnalysis <input> <out>");
            System.exit(2);
        }

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

        Job job = new Job(conf, "H1B Data Analysis");
        job.setJarByClass(H1BSucessFailureMapper.class);
        job.setMapperClass(H1BSucessFailureMapper.class);
        job.setReducerClass(H1BSucessFailureReducer.class);
        job.setNumReduceTasks(1);

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

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

        // Delete output if exists
        FileSystem hdfs = FileSystem.get(conf);
        if (hdfs.exists(outputDir))
            hdfs.delete(outputDir, true);

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

        System.exit(code);
    }

    public static class H1BSucessFailureMapper extends Mapper<Object, Text, Text, IntWritable> {

        private Text outkey = new Text();
        private IntWritable outvalue = new IntWritable();

        public void map(Object key, Text value, Context context) throws IOException,
        InterruptedException {
            //String[] values = value.toString().split(",");
            String[] values = value.toString().split("(?=[^\\"]*\"[^\"]*\"|\"[^\"]*\"|\"[^\"]*$)");
        }
    }
}
```

```

        if (values == null) {
            return;
        }

        if (!values[1].equals("\\CASE_STATUS\\")) {

            if (values[5].equalsIgnoreCase("\\Y\\"))
                outkey.set("FULL TIME");
            else if (values[5].equalsIgnoreCase("\\N\\"))
                outkey.set("PART TIME");
            outvalue.set(1);
            context.write(outkey, outvalue);
        }
    }
}

public static class H1BSucessFailureReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

    private IntWritable outvalue = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {
        int sum = 0;
        // For each input value
        for (IntWritable t : values) {
            sum += t.get();
        }
        outvalue.set(sum);
        context.write(key, outvalue);
    }
}
}

```

2. Code: Number of petitions based on Location (Counting with counters Summarization)

```
public class H1BPetitionCountPerLocation {

    // Counting number of petitions based on Location(Counting with counters)

    public static void main(String[] args) throws IOException, ClassNotFoundException,
    InterruptedException {
        // TODO Auto-generated method stub
        Configuration conf = new Configuration();

        if (args.length != 2) {
            System.err.println("Usage: H1BDataAnalysis <input> <out>");
            System.exit(2);
        }

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

        Job job = new Job(conf, "H1B Data Analysis");
        job.setJarByClass(H1BPetitionCountPerLocation.class);
        job.setMapperClass(CountPetitionsPerLocationMapper.class);
        job.setNumReduceTasks(0);

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

        // Delete output if exists
        FileSystem hdfs = FileSystem.get(conf);
        if (hdfs.exists(outputDir))
            hdfs.delete(outputDir, true);

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

        if (code == 0) {
            System.out.println("OUTPUT VALUES:\nSTATE\tNUMBER OF PETITIONS");
            for (Counter counter :
job.getCounters().getGroup(CountPetitionsPerLocationMapper.STATE_COUNTER_GROUP)) {
                System.out.println(counter.getDisplayName() + "\t" + counter.getValue());
            }
        }

        FileSystem.get(conf).delete(outputDir, true);
    }

    public static class CountPetitionsPerLocationMapper extends Mapper<LongWritable, Text,
    NullWritable, NullWritable> {

        public static final String STATE_COUNTER_GROUP = "State";
        public static final String UNKNOWN_COUNTER = "Unknown";

        String[] statesArray = new String[] { "CALIFORNIA", "ALABAMA", "ARKANSAS", "ARIZONA",
"ALASKA", "COLORADO",
"CONNECTICUT", "DELAWARE", "FLORIDA", "GEORGIA", "HAWAII", "IDAHO",
"ILLINOIS", "INDIANA", "IOWA",
```

```

        "KANSAS", "KENTUCKY", "LOUISIANA", "MAINE", "MARYLAND",
"MASSACHUSETTS", "MICHIGAN", "MINNESOTA",
        "MISSISSIPPI", "MISSOURI", "MONTANA", "NEBRASKA", "NEVADA", "NEW
HAMPSHIRE", "NEW JERSEY", "NEW MEXICO",
        "NEW YORK", "NORTH CAROLINA", "NORTH DAKOTA", "OHIO",
"OKLAHOMA", "OREGON", "PENNSYLVANIA",
        "RHODE ISLAND", "SOUTH CAROLINA", "SOUTH DAKOTA", "TENNESSEE",
"TEXAS", "UTAH", "VERMONT", "VIRGINIA",
        "WASHINGTON", "WEST VIRGINIA", "WISCONSIN", "WYOMING" };
HashSet<String> states = new HashSet<String>(Arrays.asList(statesArray));

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

    String[] values = line.toString().split(",(?=[^\"]*\"[^\"]*\"|^\".*$)");
    if (values == null) {
        return;
    }

    String state = values[8].substring(values[8].indexOf(',') + 2, values[8].length() - 1);

    if (state != null && !state.isEmpty() && !state.equalsIgnoreCase("WORKSITE")) {
        boolean unknown = true;
        if (states.contains(state)) {
            context.getCounter(STATE_COUNTER_GROUP, state).increment(1);
            unknown = false;
        }
        if (unknown) {
            context.getCounter(UNKNOWN_COUNTER, state).increment(1);
        }
    }
}
}
}
}

```

3. Code: Predictive yearly analysis of successful certification of the petitions (Numerical Summarization, Standard Deviation, MR Chaining)

```
public class H1BSucessfullPetitionPrediction {

    // Predictive yearly analysis of successful certication of the petitions(Summarization,Standard
    Deviation)

    public static void main(String[] args) throws IOException, ClassNotFoundException,
    InterruptedException {
        // TODO Auto-generated method stub
        Configuration conf = new Configuration();

        if (args.length != 2) {
            System.err.println("Usage: H1BDataAnalysis <input> <out>");
            System.exit(2);
        }

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

        Job job = new Job(conf, "H1B Data Analysis");

        job.setJarByClass(H1BSucessfullPetitionPrediction.class);
        job.setMapperClass(H1BPetitionSucessMapper.class);
        job.setReducerClass(H1BPetitionSucessReducer.class);
        job.setNumReduceTasks(1);

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

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

        // Delete output if exists
        FileSystem hdfs = FileSystem.get(conf);
        if (hdfs.exists(outputDir))
            hdfs.delete(outputDir, true);

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

        if (code == 0) {
            Job job1 = new Job(conf, "H1B Data Analysis");

            job1.setJarByClass(H1BSucessfullPetitionPrediction.class);
            job1.setMapperClass(H1BPetitionSucessPredictiveMapper.class);
            job1.setReducerClass(H1BPetitionSucessPredictiveReducer.class);
            job1.setNumReduceTasks(1);

            job1.setMapOutputKeyClass(NullWritable.class);
            job1.setMapOutputValueClass(Text.class);
            job1.setOutputKeyClass(Text.class);
            job1.setOutputValueClass(IntWritable.class);
        }
    }
}
```

```

        Path finalOutput = new Path("FinalOutput");
        FileInputFormat.addInputPath(job1, outputDir);
        FileOutputFormat.setOutputPath(job1, finalOutput);
        // Delete output if exists
        if (hdfs.exists(finalOutput))
            hdfs.delete(finalOutput, true);
        code = job1.waitForCompletion(true) ? 0 : 1;
    }
    System.exit(code);
}

public static class H1BPetitionSucessMapper extends Mapper<Object, Text, Text, IntWritable> {

    private Text outkey = new Text();
    private IntWritable outvalue = new IntWritable();

    public void map(Object key, Text value, Context context) throws IOException,
    InterruptedException {
        // String[] values = value.toString().split(",");
        String[] values = value.toString().split(",(?=[^\\"]*"|\"\\\"*\")*\"\\\"*$)");

        if (values == null) {
            return;
        }

        if (!values[1].equals("CASE_STATUS")) {
            String case_status = values[1].trim();
            String year = values[7].trim();
            if ((case_status.equalsIgnoreCase("\\CERTIFIED\\")
                || case_status.equalsIgnoreCase("\\CERTIFIED-
WITHDRAWN\\"))) {

                outkey.set(year);
                outvalue.set(1);
                context.write(outkey, outvalue);
            }
        }
    }
}

public static class H1BPetitionSucessPredictiveMapper extends Mapper<Object, Text, NullWritable,
Text> {

    private Text outkey = new Text();

    public void map(Object key, Text value, Context context) throws IOException,
    InterruptedException {
        // String[] values = value.toString().split(",");
        String[] values = value.toString().split("\\t");

        if (values == null) {
            return;
        } else {
            outkey.set(values[0] + "\\t" + values[1]);
            context.write(NullWritable.get(), outkey);
        }
    }
}

```

```

    }

    public static class H1BPetitionSucessReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

        private IntWritable outvalue = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values, Context context)
            throws IOException, InterruptedException {

            int sum = 0;
            // For each input value
            for (IntWritable t : values) {
                sum += t.get();
            }
            outvalue.set(sum);
            context.write(key, outvalue);
        }
    }

    public static class H1BPetitionSucessPredictiveReducer extends Reducer<NullWritable, Text, Text,
    IntWritable> {

        private IntWritable outvalue = new IntWritable();
        private TreeMap<String, Integer> successRange = new TreeMap<String, Integer>();
        private Text result = new Text();

        public void reduce(NullWritable key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {

            int sum = 0;
            int count = 0;
            successRange.clear();
            // For each input value
            for (Text t : values) {

                String[] value = t.toString().split("\\t");
                successRange.put(value[0], Integer.parseInt(value[1]));
                count++;
                sum += Integer.parseInt(value[1]);
            }

            float mean = sum / count;
            float sumOfSquares = 0.0f;
            for (Entry<String, Integer> entry : successRange.entrySet()) {
                sumOfSquares += (entry.getValue() - mean) * (entry.getValue() - mean);
            }
            float stdDev = (float) Math.sqrt(sumOfSquares / (count - 1));
            Entry<String, Integer> lastEntry = successRange.lastEntry();
            String nextYear = String.valueOf(Integer.parseInt(lastEntry.getKey()) + 1);
            float predictedSucessPetitionValue = lastEntry.getValue() + stdDev;
            successRange.put(nextYear, (int) predictedSucessPetitionValue);
            for (Entry<String, Integer> entry : successRange.entrySet()) {
                context.write(new Text(entry.getKey()), new IntWritable(entry.getValue()));
            }
        }
    }
}

```

4. Code: Top 10 employers to file the most petitions (Top N Filtering Pattern, Partitioner)

```
public class H1BTop10ApplsPerEmployeePerYear {

    public static void main(String[] args) throws IOException, ClassNotFoundException,
    InterruptedException {
        // TODO Auto-generated method stub
        // Top 10 employers to file the most petitions(TOP N FILTERING PATTERN)
        Configuration conf = new Configuration();

        if (args.length != 2) {
            System.err.println("Usage: H1BDataAnalysis <input> <output>");
            System.exit(2);
        }

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

        Job job = new Job(conf, "H1BDataAnalysis Top 10 Employers");
        job.setJarByClass(H1BAppInsPerEmployeePerYearMapper.class);

        job.setMapperClass(H1BAppInsPerEmployeePerYearMapper.class);
        job.setPartitionerClass(CustomPartioner.class);
        job.setReducerClass(H1BAppInsPerEmployeePerYearReducer.class);
        job.setNumReduceTasks(7);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(LongWritable.class);
        job.setOutputKeyClass(NullWritable.class);
        job.setOutputValueClass(Text.class);

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

        // Delete output if exists
        FileSystem hdfs = FileSystem.get(conf);
        if (hdfs.exists(outputDir))
            hdfs.delete(outputDir, true);

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

        System.exit(code);
    }

    public static class CustomPartioner extends Partitioner<Text, LongWritable> {

        @Override
        public int getPartition(Text key, LongWritable value, int numReduceTasks) {
            String[] str = key.toString().split("\\t");
            if (str[1].equals("2011"))
                return 0;
            if (str[1].equals("2012"))
                return 1;
            if (str[1].equals("2013"))
```



```

        return 2;
    if (str[1].equals("2014"))
        return 3;
    if (str[1].equals("2015"))
        return 4;
    if (str[1].equals("2016"))
        return 5;
    else
        return 6;
    }
}

public static class H1BAppInsPerEmployeePerYearMapper extends Mapper<LongWritable, Text, Text,
LongWritable> {
    LongWritable one = new LongWritable(1);

    public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
        if (key.get() > 0)
        {
            String[] values = value.toString().split("(?=[^\\"]*" "[^"]*" )*[^"]*$");
            if (!values[1].equals("NA") && values[1] != null && !values[2].equals("NA")
&& values[2] != null
&& !values[7].equals("NA") && values[7] != null) {
                Text answer = new Text(values[2].replaceAll("\\\"", "\"") + "\\t" +
values[7]);
                context.write(answer, one);
            }
        }
    }
}

public static class H1BAppInsPerEmployeePerYearReducer extends Reducer<Text, LongWritable,
NullWritable, Text> {
    private TreeMap<LongWritable, Text> topEmployers = new TreeMap<LongWritable, Text>();
    long sum = 0;

    public void reduce(Text key, Iterable<LongWritable> values, Context context)
throws IOException, InterruptedException {
        sum = 0;
        for (LongWritable val : values) {
            sum += val.get();
        }
        topEmployers.put(new LongWritable(sum), new Text(key + "," + sum));
        if (topEmployers.size() > 10)
            topEmployers.remove(topEmployers.firstKey());
    }

    protected void cleanup(Context context) throws IOException, InterruptedException {
        for (Text t : topEmployers.descendingMap().values())
            context.write(NullWritable.get(), t);
    }
}
}

```

5. Code: Most Data Engineer jobs for each year for each Location (Top N Filtering Pattern, Partitioner)

```
public class H1BTop10DataEngineerPerYear {

    public static void main(String[] args) throws IOException, ClassNotFoundException,
InterruptedException {
        // TODO Auto-generated method stub
        // Most Data Engineer jobs for each year for each Location(TOP N FILTERING PATTERN)
        Configuration conf = new Configuration();

        if (args.length != 2) {
            System.err.println("Usage: H1BDataAnalysis <input> <output>");
            System.exit(2);
        }

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

        Job job = new Job(conf, "H1BDataAnalysis Top 10 Employers");
        job.setJarByClass(H1BAppInsPerEmployeePerYearMapper.class);

        job.setMapperClass(H1BAppInsPerEmployeePerYearMapper.class);
        job.setPartitionerClass(CustomPartioner.class);
        job.setReducerClass(H1BAppInsPerEmployeePerYearReducer.class);
        job.setNumReduceTasks(7);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(LongWritable.class);
        job.setOutputKeyClass(NullWritable.class);
        job.setOutputValueClass(Text.class);

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

        // Delete output if exists
        FileSystem hdfs = FileSystem.get(conf);
        if (hdfs.exists(outputDir))
            hdfs.delete(outputDir, true);

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

        System.exit(code);
    }

    public static class CustomPartioner extends Partitioner<Text, LongWritable> {

        @Override
        public int getPartition(Text key, LongWritable value, int numReduceTasks) {
            String[] str = key.toString().split("\\t");
            if (str[1].equals("2011"))
                return 0;
            if (str[1].equals("2012"))
                return 1;
            if (str[1].equals("2013"))
                return 2;
        }
    }
}
```

```

        if (str[1].equals("2014"))
            return 3;
        if (str[1].equals("2015"))
            return 4;
        if (str[1].equals("2016"))
            return 5;
        else
            return 6;
    }
}

public static class H1BAppInsPerEmployeePerYearMapper extends Mapper<LongWritable, Text, Text,
LongWritable> {
    LongWritable one = new LongWritable(1);

    public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
        if (key.get() > 0)
        {
            String[] values = value.toString().split(",(?=[^\"]*\"[^\"]*\")*\"[^\"]*$)");

            if (values[4] != null && values[4].contains("DATA ENGINEER") && values[8] !=
null
                && !values[8].equals("NA")) {
                Text answer = new Text(values[8].replaceAll("\\", "") + "\\t" +
values[7]); // Location and Year
                context.write(answer, one);
            }
        }
    }
}

public static class H1BAppInsPerEmployeePerYearReducer extends Reducer<Text, LongWritable,
NullWritable, Text> {

    private TreeMap<LongWritable, Text> Top10DataEngineer = new TreeMap<LongWritable,
Text>();

    long sum = 0;

    public void reduce(Text key, Iterable<LongWritable> values, Context context)
throws IOException, InterruptedException {
        sum = 0;
        for (LongWritable val : values) {
            sum += val.get();
        }
        Top10DataEngineer.put(new LongWritable(sum), new Text(key + ", " + sum));
        if (Top10DataEngineer.size() > 10)
            Top10DataEngineer.remove(Top10DataEngineer.firstKey());
    }

    protected void cleanup(Context context) throws IOException, InterruptedException {
        for (Text t : Top10DataEngineer.descendingMap().values())
            context.write(NullWritable.get(), t);
    }
}
}

```

6. Code: Categorizing petitions filed for a "CEO" for each year (Binning Data Organization Pattern)

```
public class H1BCEOPetitionsPerYear {

    public static void main(String[] args) throws IOException, ClassNotFoundException,
InterruptedException {
        // TODO Auto-generated method stub

        // Categorizing petitions filed for a "CEO" for each year(BINNING ORGANIZATION
        // PATTERN)

        Configuration conf = new Configuration();

        if (args.length != 2) {
            System.err.println("Usage: H1BDataAnalysis <input> <out>");
            System.exit(2);
        }

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

        Job job = new Job(conf, "H1B Data Analysis");
        job.setJarByClass(H1BCEOPetitionsPerYear.class);
        job.setMapperClass(H1BCEOPetitionsPerYearMapper.class);
        job.setNumReduceTasks(0);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);

        MultipleOutputs.addNamedOutput(job, "bins", TextOutputFormat.class, Text.class,
NullWritable.class);
        MultipleOutputs.setCountersEnabled(job, true);

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

        // Delete output if exists
        FileSystem hdfs = FileSystem.get(conf);
        if (hdfs.exists(outputDir))
            hdfs.delete(outputDir, true);

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

        System.exit(code);
    }

    public static class H1BCEOPetitionsPerYearMapper extends Mapper<Object, Text, Text, NullWritable> {

        private MultipleOutputs<Text, NullWritable> mos = null;

        protected void setup(Context context) {
            mos = new MultipleOutputs<Text, NullWritable>(context);
        }
    }
}
```

```

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

        String[] values = value.toString().split("(?=[^\\]*\\\"[^\\]*\\\"|\\\"[^\\]*\\\"$)");
        if (values == null) {
            return;
        }

        if (!values[7].equalsIgnoreCase("YEAR")) {
            if (values[4].substring(1, values[4].length()-1).equalsIgnoreCase("CEO") ||
values[4].substring(1, values[4].length()-1).equalsIgnoreCase("CHIEF EXECUTIVE OFFICER")
|| values[4].substring(1, values[4].length()-1).equalsIgnoreCase("CHEIF EXECUTIVE OFFICER")) {
                if (values[7].equalsIgnoreCase("2011")) {
                    mos.write("bins", value, NullWritable.get(), "Year-2011");
                } else if (values[7].equalsIgnoreCase("2012")) {
                    mos.write("bins", value, NullWritable.get(), "Year-2012");
                } else if (values[7].equalsIgnoreCase("2013")) {
                    mos.write("bins", value, NullWritable.get(), "Year-2013");
                } else if (values[7].equalsIgnoreCase("2014")) {
                    mos.write("bins", value, NullWritable.get(), "Year-2014");
                } else if (values[7].equalsIgnoreCase("2015")) {
                    mos.write("bins", value, NullWritable.get(), "Year-2015");
                } else if (values[7].equalsIgnoreCase("2016")) {
                    mos.write("bins", value, NullWritable.get(), "Year-2016");
                }
            }
        }
    }

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

```

7. Successful Petitions based on the Wages (Inverted Index Pattern)

```
public class H1BWageRangeSummarization {

    public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException {
        // TODO Auto-generated method stub

        // Normal Counting of Number of Full time and Part time Petitions

        Configuration conf = new Configuration();

        if (args.length != 2) {
            System.err.println("Usage: H1BDataAnalysis <input> <out>");
            System.exit(2);
        }

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

        Job job = new Job(conf, "H1B Data Analysis");
        job.setJarByClass(H1BWageRangeSummarizationMapper.class);
        job.setMapperClass(H1BWageRangeSummarizationMapper.class);
        job.setReducerClass(H1BWageRangeSummarizationReducer.class);
        job.setNumReduceTasks(1);

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

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

        // Delete output if exists
        FileSystem hdfs = FileSystem.get(conf);
        if (hdfs.exists(outputDir))
            hdfs.delete(outputDir, true);

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

        System.exit(code);
    }

    public static class H1BWageRangeSummarizationMapper extends Mapper<Object, Text, Text, IntWritable> {

        private Text wageRange = new Text();
        private IntWritable ONE = new IntWritable(1);

        public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
            // String[] values = value.toString().split(",");
            String[] values = value.toString().split("(?=[^\\"]*" "[^"]*" )*"[^"]*$)");
            if (values == null) {
```

```

        return;
    }

    if (!values[1].equals("\CASE_STATUS\")) {
        String case_status = values[1].trim();
        if ((case_status.equalsIgnoreCase("\CERTIFIED\")
            || case_status.equalsIgnoreCase("\CERTIFIED-WITHDRAWN\"))) {
            double employeeWage = Double.parseDouble(values[6]);
            // jobtitle_outValue.set(job_title);
            if (employeeWage >= 0 && employeeWage <= 30000) {
                wageRange.set("0-30000 Success Range");
            } else if (employeeWage > 30000 && employeeWage <= 60000) {
                wageRange.set("30000-60000 Success Range");
            } else if (employeeWage > 60000 && employeeWage <= 70000) {
                wageRange.set("60000-70000 Success Range");
            } else if (employeeWage > 70000 && employeeWage <= 80000) {
                wageRange.set("70000-80000 Success Range");
            } else if (employeeWage > 80000 && employeeWage <= 90000) {
                wageRange.set("80000-90000 Success Range");
            } else if (employeeWage > 90000 && employeeWage <= 100000) {
                wageRange.set("90000-100000 Success Range");
            } else if (employeeWage > 100000) {
                wageRange.set("100000- Above Success Range");
            }

        } else {
            if (!values[6].equals("NA")) {
                double employeeWage = Double.parseDouble(values[6]);
                // jobtitle_outValue.set(job_title);
                if (employeeWage >= 0 && employeeWage <= 30000) {
                    wageRange.set("0-30000 Failure Range");
                } else if (employeeWage > 30000 && employeeWage <= 60000) {
                    wageRange.set("30000-60000 Failure Range");
                } else if (employeeWage > 60000 && employeeWage <= 70000) {
                    wageRange.set("60000-70000 Failure Range");
                } else if (employeeWage > 70000 && employeeWage <= 80000) {
                    wageRange.set("70000-80000 Failure Range");
                } else if (employeeWage > 80000 && employeeWage <= 90000) {
                    wageRange.set("80000-90000 Failure Range");
                } else if (employeeWage > 90000 && employeeWage <= 100000) {
                    wageRange.set("90000-100000 Failure Range");
                } else if (employeeWage > 100000) {
                    wageRange.set("100000- Above Failure Range");
                }

            }

        }
        context.write(wageRange, ONE);
    }
}

}

}

public static class H1BWageRangeSummarizationReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

```

```
private IntWritable outvalue = new IntWritable();
private Text outkey = new Text();

public void reduce(Text key, Iterable<IntWritable> values, Context context)
    throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable s : values) {
        sum += s.get();
    }
    outvalue.set(sum);
    context.write(key, outvalue);
}
}
```