Norwegian 1910 Census Data Analysis using Hadoop

Bernt Andreas Eide
Department of Electrical Engineering and Computer Science (University of Stavanger, Norway)
ba.eide@stud.uis.no

Shaon Rahman
Department of Electrical Engineering and Computer Science (University of Stavanger, Norway)
s.rahman@stud.uis.no

ABSTRACT

In this project we chose to analyze census data using a Big Data framework known as Hadoop [5], Hadoop consist of multiple modules. One of the main modules utilized for this project is the HDFS (Hadoop Distributed File System), which is mainly used to process large datasets. Census data can become considerably large, depending on the target period. In our case we are analyzing Norwegian census records from 1910, preprocessing and aggregation is done with MapReduce, MRJob [17] and Spark [18]. The visualization is done using Python [15] libraries that support plotting of choropleth maps, such as Geopandas [8] & Geoplot [6].

KEYWORDS

hadoop, big data, geoplot, geopandas, python, data science, distributed computing, census, norway, visualization, preprocessing, demography, spark, mrjob, mapreduce

1 INTRODUCTION

Context and Motivation. Hadoop is mainly used for distributed computing, and can be used to process large datasets. Census data can be considerably large, especially if the data of interest is from a highly populated country. But even so, it can be applied to smaller datasets as well. Hadoop can harness the computational power of multiple computers, so that data can be processed in parallel. Census data is generally transcribed, contains lots of typos and/or missing values. Which makes census data highly relevant for distributed preprocessing, once the data has been cleaned it is very trivial to aggregate it for visualization purposes.

In this project, the records being processed are specifically Norwegian census records from 1910. Once the data has been cleaned it is possible to visualize many different areas of interest like population density, population growth, religious distribution, occupation, etc. It would also be possible to find correlation between other historical data, like precipitation rate based on location correlated with population density.

Research Problem. Publicly available and search-able census data in large quantities can be quite overwhelming. In this project a baseline is established for how one could use a Big Data framework to process census related data. And how to use the processed data for visualization and statistical purposes.

Contribution Summary.

- We have created a generic implementation of how one would use Hadoop together with MapReduce, MRJob and Spark in a census related context. And finally how to use the results for visualization on a geographical plot.

Related Work.

- Census Data Analysis using Apache Hive and Zeppelin [10]
- Finding Insights Hadoop Cluster Performance Analysis over Census Dataset Using Big-Data Analytics [4]
- Hadoop and Pig for Internet Census Data Analysis [12]

2 BACKGROUND

We have been given access to 6 GB of census data from Digitalarkivet [1], the data itself was provided in separate folders and in each folder there were one XML (Extensible Markup Language) file for each municipality. Each XML file contained data about people living in different regions of the given municipality, the folders were organized by year. We had access to 1801, 1865, 1900 and 1910 census. We chose to work with the latter, simply because the 1910 census is the most complete among the four.

The features we were most interested in were birth year, place of birth (municipality), municipality at the time of counting, gender, martial status, religion, and occupation. Using these features we were able to visualize the distribution of religion, gender, population, births, etc. The visualization can be in the form of geographical plots and graphs. Most of the visualization done for this project used geographical plots, this way it is possible to explore further details in the data.

Supervised by Jayachander Surbiryala.

Project in Data Science (DAT500), IDE, UiS
2020.

3 METHOD

3.1 Hadoop Configuration

A significant part of this project was to configure a Hadoop cluster which would utilize three slave nodes. The master node had roughly 8 GB RAM while each slave node had 4 GB RAM. The slaves ran on Ubuntu 16.04, while the master ran on Ubuntu 18.04.

Initially the master also ran on Ubuntu 16.04, however there were certain features like Jupyter Notebook Server that would be available if the master was updated to Ubuntu 18.04. Upgrading the master to run under Ubuntu 18.04 posed several issues related to Hadoop compatibility. Such as, Java 11 is the default for Ubuntu 18.04 and higher, but Java 8 is required.

Hadoop's core functionality is built on Java, and therefore running the same Java version is recommended for optimal usage. And a newer Python version was required for the slave nodes, Ubuntu 18.04 used a newer version of Python 3. The only way to ensure a version match was to build the Python version needed from source.

Software prerequisites

- Hadoop 3.1.1
- Spark 2.4.5
- Java 8
- Python 3.6.9 (built from source)
- MRJob 0.5.10
- PySpark 2.4.5

Listing 1: Building Python 3.6.9 from source

```
#!/bin/bash
apt-get install zlib1g-dev && \
cd /opt && \
wget https://www.python.org/ftp/python/3.6.9/Python-3.6.9.tgz &&↩
    \
tar -xvf Python-3.6.9.tgz && \
cd Python-3.6.9/ && \
./configure && \
make && \
make install && \
rm -R /opt/Python-3.6.9.tgz
```

By default Python 2 was the default Python interpreter, this was fixed by ensuring that the Python alias would reflect Python 3 as the default interpreter. The environment also had to be set correctly, so that Hadoop would find the required Java directory. MRJob and Spark also required the Hadoop home and config directories to be defined. This was done by modifying the /etc/environment file to define the following environment variables.

- JAVA_HOME
- SPARK_HOME
- HADOOP_CONF_DIR
- YARN_CONF_DIR

And, the PATH environment variable was updated to define paths for the Hadoop and Spark binary directories. (sbin and bin)

As listed above, Hadoop 3.1.1 was chosen. The contents were extracted into /usr/local/hadoop/ rather than the home directory. In which case setting the appropriate ownership of the directory is vital. Because directories outside of the home directory are normally tied to another user, in this case it was the root user.

Listing 2: Creating the Hadoop directory and setting the appropriate ownership

```
#!/bin/bash
mkdir /usr/local/hadoop
chown -R ubuntu /usr/local/hadoop
```

Hadoop itself only required some minor configurations, /etc/hosts lists the master and slave nodes and their respective local IP addresses so that the Hadoop environment will be able to establish communication between nodes and master. This will work if it is possible to SSH between the nodes, without the need to specify any additional credentials.

Listing 3: /etc/hosts

```
192.168.22.29 master
192.168.22.162 slave-1
192.168.22.97 slave-2
192.168.22.196 slave-3
```

The replication factor was set to the amount of slaves available (3) in the /usr/local/hadoop/etc/hadoop/hdfs-site.xml file. In /usr/local/hadoop/etc/hadoop/hadoop-env.sh, JAVA_HOME, HADOOP_HOME and HADOOP_CONF_DIR were exported to use the correct paths. This would ensure that other libraries using Hadoop would find these directories. Finally when everything was configured, the namenode was formatted. This would ensure that the HDFS was created.

Listing 4: Formatting the namenode

```
#!/bin/bash
cd /usr/local/hadoop/bin
./hdfs namenode -format
```

3.2 Spark Configuration

To correctly use and configure Spark 2.4.5 it is required to run a version of Python prior to 3.8.0 and Java 8 must be installed.

Spark was extracted into /usr/local/spark/, minor configurations were made to the settings files. A history server was enabled to enforce better logging of Spark related jobs, errors and warnings, and console logging would be limited to warnings and errors. Otherwise Spark will spam the command prompt with random info messages when running jobs via spark-submit. Some of these messages can also be suppressed through code, but this will not suppress everything.

Listing 5: /usr/local/spark/conf/log4j.properties

```
log4j.rootCategory=WARN, console
log4j.logger.org.apache.spark.repl.Main=WARN
```

Listing 6: Only print vital log messages in the console

```
spark = SparkSession.builder.appName('CensusAnalysis').getOrCreate()
sc = spark.sparkContext
sc.setLogLevel("ERROR")
```

The spark-env.sh file located in /usr/local/spark/conf/ was modified to force the PySpark library to run via Python 3, by setting PYSPARK_PYTHON and PYSPARK_DRIVER_PYTHON to *python3.6*.

Finally the spark-defaults.conf file was modified to utilize YARN as the master for job handling and scheduling, max memory for the driver, executor and yarn was set to 1 GB, 512 MB and 1 GB respectively. Due to the data being run through Spark was less than 512 MB it was not necessary to allocate more memory for Spark.

3.3 Dataset

The dataset used in this project contained roughly 2.48 million records of people across Norway in 1910 and prior. The data was fairly well structured in XML format, but was not necessarily trivial for use in Hadoop. Running MapReduce in Hadoop will split the data (the whole file) in chunks, certain information will be lost due to this split, because the information contained for a person is present on multiple lines. CSV (Comma Separated Values) and generally unstructured data where you do not rely on specific patterns in the dataset is more beneficial, which is why a transformation was necessary.

The dataset was converted into a more compact format, this would ensure that Hadoop would be able to process the data correctly. Initially the 1910 census dataset was roughly 2 GB, after this simple transformation the dataset was reduced to 200 MB. The transformation involved selecting the features of interest and doing some minor preprocessing. Fields like gender and martial status were cleaned up, since these fields are very trivial. A missing value for one of these features could easily be filled with a majority vote.

Features that were excluded from the transformation were mainly personal features that would have had no impact on the visualization step, however there were features that could have been useful if the task was to create a family tree, or ancestry tree.

```
<person id="pf01036335002889">
    <personnummer>002</personnummer>
    <husholdningsnummer>01</husholdningsnummer>
    <fornavn>Josefine Karoline</fornavn>
    <patronymikon>Kristoffersen</patronymikon>
    <kjonn>k</kjonn>
    <familiestilling>hm</familiestilling>
    <sivilstand>g</sivilstand>
    <yrke>Hustru</yrke>
    <fodselsaar>1885-10-21</fodselsaar>
    <fodested>Onsø</fodested>
    <bostatus>b</bostatus>
    <statsborgerskap>n</statsborgerskap>
    <trossamfunn>s</trossamfunn>
</person>
```

Figure 1: Original Census Data

Listing 7: Extract certain features from people in the original data and merge into CSV

```
PATTERN_EVIL_CHARS = r'[^a-zA-Z ]+'
PATTERN_WHITESPACE_SPAM = r'\s\s+'
PATTERN_NUMBERS_ONLY = r'[^0-9]+'
for line in stream:
  if not person:
    if line.find("<person") >= 0:
      person = True
      gender = profession = maritalStatus = religion = birthYear = birthPlace = '?' # Reset
      maritalStatus = 'Single'
      religion = 'Norwegian Church'
  else:
    if line.find("</person>") >= 0: # We reached the end for this person, reset.
      person = False
      data.append("{},{},{},{},{},{},{},{},{}".format(census_year, county, municipality, gender, profession, maritalStatus, religion, birthYear, birthPlace))
      continue

    idxProfession_start, idxProfession_end = line.find("<yrke>"), line.find("</yrke>")
    if idxProfession_start >= 0: # Field of Work
      profession = line[(idxProfession_start+6):idxProfession_end]
      profession = re.sub(PATTERN_EVIL_CHARS, '', profession)
      profession = re.sub(PATTERN_WHITESPACE_SPAM, ' ', profession)
      profession = profession.title()

    idxMartialStatus_start, idxMartialStatus_end = line.find("<sivilstand>"), line.find("</sivilstand>")
    if idxMartialStatus_start >= 0: # Martial Status
      maritalStatus = line[(idxMartialStatus_start+12):idxMartialStatus_end]
      maritalStatus = 'Married' if maritalStatus == 'g' else 'Single'

    idxReligion_start, idxReligion_end = line.find("<trossamfunn>"), line.find("</trossamfunn>")
    if idxReligion_start >= 0: # Religion
      religion = line[(idxReligion_start+13):idxReligion_end]
      religion = re.sub(PATTERN_EVIL_CHARS, '', religion)
      religion = re.sub(PATTERN_WHITESPACE_SPAM, ' ', religion)
      religion = 'Norwegian Church' if religion == 's' else religion.title()

    idxGender = line.find("<kjonn>")
    if idxGender  >= 0: # Gender
      gender = line[(idxGender+7):(idxGender+8)]
      gender = 'Male' if gender == 'm' else 'Female'

    year_start, year_end = line.find(">"), line.find("</fodselsaar>")
    if year_end >= 0: # Birth Year
      birthYear = line[(year_start+1):year_end]
      birthYear = re.sub(PATTERN_NUMBERS_ONLY, ',', birthYear)
      birthYear = [int(yr) for yr in birthYear.split(',') if len(yr) == 4]
      if len(birthYear) <= 0:
        birthYear = '?'
      else:
        birthYear = int(np.median(birthYear))
        if (birthYear > int(census_year)) or (birthYear < 1600): # Invalid year!
          birthYear = '?'

    birthPlace_start, birthPlace_end = line.find("<fodested>"), line.find("</fodested>")
    if birthPlace_start >= 0: # Birthplace
      birthPlace = line[(birthPlace_start+10):birthPlace_end]
      birthPlace = re.sub(PATTERN_EVIL_CHARS, '', birthPlace)
      birthPlace = re.sub(PATTERN_WHITESPACE_SPAM, ' ', birthPlace)
      birthPlace = birthPlace.title()
```

```
1910,Østfold,Fredrikshald,Male,Vrft Skibstmmermand,Married,Norwegian Church,1837,Id
1910,Østfold,Fredrikshald,Female,Hustru,Married,Norwegian Church,1839,Id
1910,Østfold,Fredrikshald,Male,Sn,Single,Norwegian Church,1871,Frhald
1910,Østfold,Fredrikshald,Female,Tjenestepike,Single,Norwegian Church,1894,Id
1910,Østfold,Fredrikshald,Male,Vrft Skipstmmermand,Married,Norwegian Church,1884,Rakkestad
1910,Østfold,Fredrikshald,Female,Hustru,Married,Norwegian Church,1878,Fredrikshald
1910,Østfold,Fredrikshald,Male,Sn,Single,Norwegian Church,1908,Fredrikshald
1910,Østfold,Fredrikshald,Male,Stenhugger,Married,Norwegian Church,1853,Sterdalen Rendalen Vre
1910,Østfold,Fredrikshald,Female,Hustru,Married,Norwegian Church,1854,Dalsland
1910,Østfold,Fredrikshald,Female,Datter,Single,Norwegian Church,1896,Id
1910,Østfold,Fredrikshald,Male,Skibs Vrftsarbeider,Married,Norwegian Church,1858,Fredrikshald
1910,Østfold,Fredrikshald,Female,Hustru,Married,Norwegian Church,1859,Fredrikshald
1910,Østfold,Fredrikshald,Female,Butikdame,Single,Norwegian Church,1885,Fredrikshald
1910,Østfold,Fredrikshald,Male,Bryggearbeider,Single,Norwegian Church,1888,Fredrikshald
1910,Østfold,Fredrikshald,Male,Snekkerlrling,Single,Norwegian Church,1891,Fredrikshald
1910,Østfold,Fredrikshald,Male,Smedjelrling,Single,Norwegian Church,1892,Fredrikshald
```

Figure 2: Transformed Census Data

3.4 Preprocessing

The data provided for this project is based on census records that have been transcribed by professionals. The original records are over 100 years old and therefore contain a lot of *outdated* information. In 1910 Norway consisted of at least 659 municipalities, since then this has been truncated to 422 municipalities in 2018. Previous smaller municipalities have been merged into other nearby municipalities and in some cases created even brand new municipalities. [2]

Since the geographical plotting is done using a shapefile [14] from 2018, the previous municipalities have to be mapped to their respective modern day municipalities. A one way conversion is done for each municipality and birthplace in the transformed dataset. Birthplaces required additional preprocessing due to a significant amount of typos and abbreviations, the birthplaces field has been transcribed directly regardless of whether or not it was valid. In this project $\approx$ 86 % of the birthplaces are valid, a large remainder are birthplaces outside of Norway.

The old municipalities and birthplace municipalities have been transformed using a dictionary which simply maps the old municipality to the new one. Municipalities have been mapped correctly with the help from this particular Wikipedia article. [3]

The field of work and religion features were also directly transcribed and required some minor preprocessing. Most Norwegians are tied to the Norwegian Church by birth, so therefore missing values and/or invalid values were set to Norwegian Church by default. The remaining religions were mostly different branches of Christianity, some Judaism and non-believers. Field of work however had lots of similar values but with typos and abbreviations, more significant cleaning could have been done to this particular feature.

```
Halden,Female,Housewife,Married,Norwegian Church,1871,Halden
Halden,Male,Sn,Single,Norwegian Church,1902,Halden
Halden,Male,Sn,Single,Norwegian Church,1903,Halden
Halden,Female,Datter,Single,Norwegian Church,1905,Halden
Halden,Male,Sn,Single,Norwegian Church,1908,Halden
Halden,Male,Bryggeformand,Married,Norwegian Church,1871,Halden
Halden,Female,Housewife,Married,Norwegian Church,1876,Oslo
Halden,Female,Datter,Single,Norwegian Church,1897,Halden
Halden,Male,Sn,Single,Norwegian Church,1899,Halden
Halden,Male,Sn,Single,Norwegian Church,1900,Halden
Halden,Female,Datter,Single,Norwegian Church,1903,Halden
Halden,Female,Datter,Single,Norwegian Church,1910,Halden
Halden,Female,Maid,Single,Norwegian Church,1893,Skee Sverige
Halden,Male,Skipsfrer,Married,Norwegian Church,1856,Hvaler
Halden,Female,Housewife,Married,Norwegian Church,1869,Nannestad
Halden,Male,Postbud,Single,Norwegian Church,1889,Halden
```

Figure 3: Preprocessed Census Data

3.5 Implementation

3.5.1 Basic MapReduce. MapReduce is a technique to map data to another set of data in order to reduce the overall data, the mapped data is reduced, sorted and merged into a more compact format. [16] A basic MapReduce job was initiated to do all the necessary preprocessing steps mentioned earlier.

Listing 8: Script to run Hadoop MapReduce Streaming

```
#!/bin/bash
hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.1.1.jar \
-mapper $1 \
-reducer $2 \
-input $3 \
-output $4 \
-file $1 \
-file $2
```

Since the input data is using the merged CSV file, split every line on comma and clean each feature separately. Map and combine the features of interest, discard the rest. The reducer will print the output of the mapper, the main purpose is to do the preprocessing in a distributed manner.

Listing 9: Preprocessing birthplaces

```
def findBestCandidate(a):
    words = a.split()

    b = [MUNICIPALITIES_TO_MODERN[w] for w in words if w in MUNICIPALITIES_TO_MODERN]
    if len(b) != 0:
        return b[0]

    b = [BIRTH_PLACE_REPLACE[w] for w in words if w in BIRTH_PLACE_REPLACE]
    if len(b) != 0:
        return MUNICIPALITIES_TO_MODERN[b[0]]

    return None

def findUpdatedBirthPlace(a):
    a = a.strip().lower().replace('aa','å').title()
    if a in MUNICIPALITIES_TO_MODERN:
        return MUNICIPALITIES_TO_MODERN[a]
    elif a in BIRTH_PLACE_REPLACE:
        return MUNICIPALITIES_TO_MODERN[BIRTH_PLACE_REPLACE[a]]

    # Still nothing? Check word for word if we find any potential combo.
    candidate = findBestCandidate(a)
    if (candidate is None) and a.endswith('en'): # Do a quick check, if for ex Rendalen->Rendal
        candidate = findBestCandidate(a[:-2])

    return (candidate if candidate else a)
```

3.5.2 Aggregation with MRJob. MRJob is a powerful Python library which can be used together with Hadoop MapReduce, MRJob can also run locally without Hadoop which makes testing even easier. It is also possible to define multiple steps that involve mapping, combining and reducing. Everything is done in a distributed fashion over the HDFS.

When MRJob is used to run a MapReduce job it will assume that the input is raw text, and anything else will be written as JSON (JavaScript Object Notation) objects using the JSONValueProtocol

protocol. Protocols can be set for input, internal and output, each protocol defines a read and write method. In some cases it may not be desirable to write the output as a JSON object, in which case a different custom or built in protocol can be used. For this project it was necessary to write the output from MRJob tasks in a CSV format.

Listing 10: Defining a custom protocol for read & write CSV

```
class CSVProtocol(object):
    def read(self, line):
        d = line.split(',')
        return (d[0], d[1]), int(d[2])

    def write(self, key, value):
        return '{},{},{}'.format(key[0], key[1], value).encode()
```

Furthermore, MRJob was used to run MapReduce to fetch population in each municipality during 1910 and also for fetching births in their respective municipalities for each given birth year.

Listing 11: Running MapReduce with MRJob to fetch all births

```
class MRAnalyzeBirths(MRJob):
    OUTPUT_PROTOCOL = CSVProtocol

    def steps(self):
        return [MRStep(mapper=self.mapper_births, reducer=self.↩
             reducer_births)]

    def mapper_births(self, k, v):
        line = v.strip().split(',')
        yield (line[-2], line[-1]), 1

    def reducer_births(self, k, v):
        data = list(v)
        yield k, len(data)
```

3.5.3 Aggregation with Spark. Spark is another tool which can utilize MapReduce, one advantage Spark has over regular MapReduce with MRJob is that it processes everything in main memory rather than reading constantly from disk. This greatly reduces latency associated with disk I/O. (Input/Output) [16]

When a dataset is loaded into Spark, the dataset will be split into several partitions across the cluster and read into main memory. An RDD (Resilient Distributed Dataframe) object is created once the dataset has been fully loaded into Spark. This object allows for filtering, mapping, creating a temporary SQL table for SQL-like syntax filtering, grouping, etc.

Listing 12: Loading data into Spark and creating a temp SQL-like table

```
sqlContext = SQLContext(sc)
df_census = sc.textFile('/data/census_preprocessed.csv')
people = (df_census.map(lambda l: l.split(",")).map(lambda p: ↩
     Row(municipality=p[0],gender=p[1],work=p[2],religion=p[4]))↩
     )
schemaPeople = sqlContext.createDataFrame(people)
schemaPeople.registerTempTable("people")
```

In this project Spark was used to generate data for common religions, common field of work and gender distribution in each municipality. For religion, the Norwegian Church was excluded due to this being the most common of them all by a significant amount.

Listing 13: Generating data using Spark

```
# Religion Distribution
excluded_religions = set(['norwegian church', '?'])
common_religions = Counter(sqlContext.sql("SELECT religion FROM ↩
     people").collect()).most_common(20)
common_religions = set([k.religion for k,_ in common_religions ↩
     if k.religion.lower() not in excluded_religions and len(k.↩
     religion) > 0])
schemaPeople.filter(schemaPeople.religion.isin(common_religions)↩
     ).groupBy('municipality', 'religion').count().orderBy("↩
     municipality").write.csv('/output/analyze_religion')

# Work Distribution
common_work = set(['Farmer', 'Fisher', 'Housewife', 'Maid'])
schemaPeople.filter(schemaPeople.work.isin(common_work)).groupBy↩
     ('municipality', 'work').count().orderBy("municipality").↩
     write.csv('/output/analyze_work')

# Gender Distribution
schemaPeople.groupBy('municipality', 'gender').count().orderBy("↩
     municipality").write.csv('/output/analyze_gender')
```

The results are written to the HDFS in CSV format, however the results contain many files due to Spark loading the data in chunks across the cluster. All that remains is to merge the output chunks.

Listing 14: Merging output from Spark into single CSV files

```
hadoop fs -getmerge -skip-empty-file /output/analyze_religion/↩
     part* ./data/census_religion.csv && \
hadoop fs -getmerge -skip-empty-file /output/analyze_work/part* ↩
     ./data/census_work.csv && \
hadoop fs -getmerge -skip-empty-file /output/analyze_gender/part↩
     * ./data/census_gender.csv
```

3.5.4 Visualization. Finally when the results from the MapReduce and SparkSQL operations have been generated it is time to visualize the data. Visualization is done using Matplotlib [9], Pandas [11], NumPy [13], Geoplot and Geopandas. A shapefile obtained from Kartverket [14] is used for the demographical plotting, the shapefile is loaded directly into Geopandas.

Listing 15: Plotting an empty geographical map of Norway

```
import geopandas as gpd
df_municipalities = gpd.read_file('shape/municipalities_2018.↩
     geojson')
df_municipalities.plot()
```

Figure 4: Empty Geographical Plot of Norway

The plot can use any colormap defined in Matplotlib, together with min and max values. Categorical plotting is also supported, plus the use of *legend* in either case.

Listing 16: Plotting a geographical map of Norway using a cmap with random values

```
fig, ax = plt.subplots(1, figsize=(95,30))
df_municipalities.plot(ax=ax,vmin=0.0,vmax=1.0,cmap='Reds',↩
      column=np.array([np.random.rand() for _ in range(len(↩
      df_municipalities))]))
ax.axis('off')
plt.tight_layout()
```

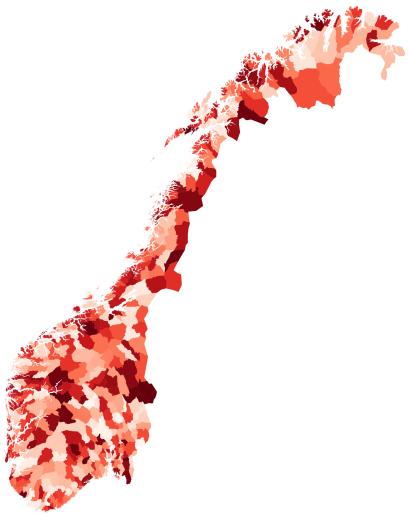

Figure 5: Geographical plot of Norway using a cmap with random values

The values supplied to the column attribute can be an array or an actual column name in the Geopandas dataframe. If an array is supplied it must have the same length as the dataframe, because each value in the dataframe is a municipality with an associated set of polygons. Together all of these elements make up the map itself. It is also possible to plot a single municipality alone, for example Oslo.

Listing 17: Plotting Oslo

```
df_municipalities.loc[df_municipalities.Municipality == 'Oslo'].↩
      plot(ax=ax)
```

Figure 6: Geographical plot of Oslo

4 RESULT & DISCUSSION

Data extracted from MapReduce and SparkSQL can now be used to visualize demographics and graphs. Visualization can be done in Jupyter Notebook or any IDE that supports Python 3. As described earlier, when using Geopandas for visualization it is necessary to provide a column name or vector. If a vector is supplied it must have equal length compared to the Geopandas dataset, and can contain either numerical or categorical values. The vector is generated by loading and aggregating the results obtained from the MapReduce and SparkSQL steps.

Listing 18: Load and visualize population data

```
MUNICIPALITIES = set(df_municipalities.Municipality)
MUNICIPALITIES_IDX = { m:i for i, m in enumerate(↩
      df_municipalities.Municipality) }

df_pop_1910 = pd.read_csv('data/census_population.csv', encoding↩
      ='utf-8', names=['Municipality', 'Population'])
pop = np.zeros(len(df_municipalities), dtype=np.uint64)
for m, c in zip(df_pop_1910.Municipality, df_pop_1910.Population↩
      ):
    pop[MUNICIPALITIES_IDX[m]] = c

fig, ax = plt.subplots(1, figsize=(95,30))
df_municipalities.plot(ax=ax, column=pop, cmap='Reds', vmin=0.0,↩
       vmax=np.ceil(pop[pop.nonzero()].mean()))
ax.axis('off')
plt.tight_layout()
```

Figure 7: Norwegian 1910 Population Density Distribution, darker shades of red implies higher density

Similarly visualizing the distribution of religion and field of work is done by picking the majority vote for each municipality. Since this is a categorical based plot, the most common religion or type of work is selected for each municipality.

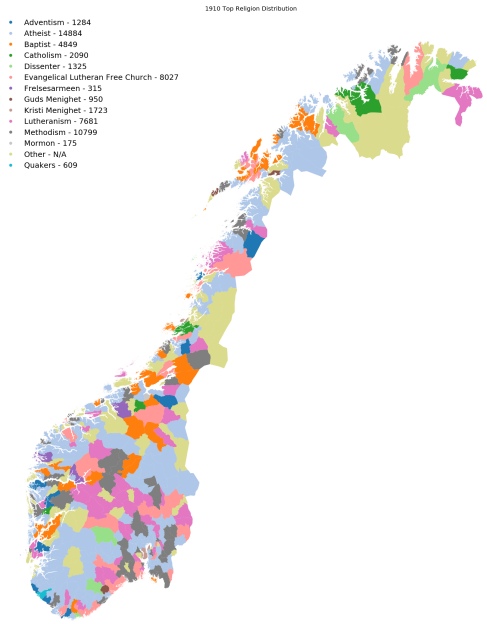

Figure 8: Popular Religions in Norway, 1910

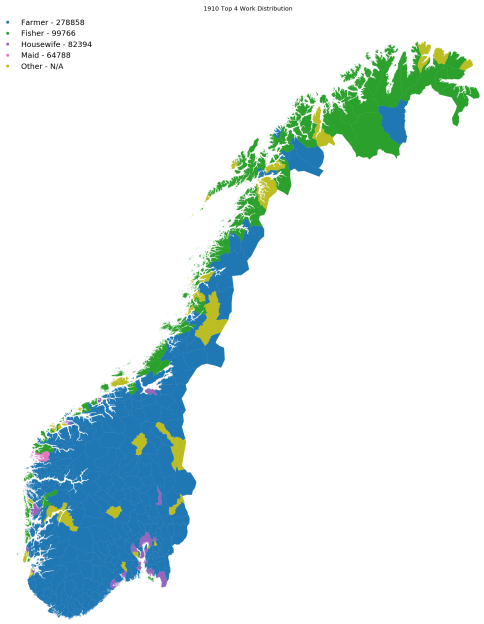

Figure 9: Popular Occupations in Norway, 1910

More preprocessing could have brought forth further details in these two previous figures. However there are some shortcomings to these plots, picking the majority vote excludes the other values. It would have been equally interesting to visualize a certain amount of religions or occupations in each municipality alone. Unfortunately due to time constraints and limitations in Geopandas itself, this was left undone.

Graphs can also be generated from the results, but will not necessarily provide as much insight in the data. Due to this project focusing mostly on demographics. However, a distribution of gender differences in each municipality was generated. The graph clearly indicates that a lot more women lived in Oslo than men. Other than that it is rather fifty-fifty which is expected.

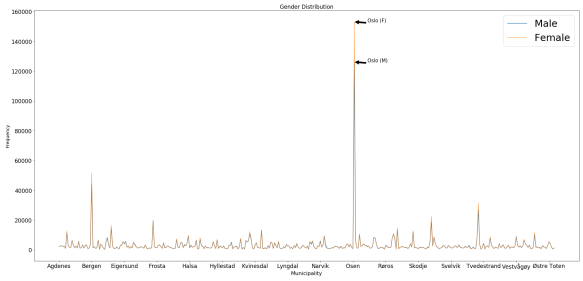

Figure 10: Gender Distribution in Norway, 1910

An animation was also created to visualize births from 1800 to 1910. This was done by creating a matrix with shape (years of interest X number of municipalities). The first row would contain all births for the year 1800 in their respective municipalities (columns).

Then simply taking the cumulative of axis 0 (the rows) would return a matrix which would represent the births in an increasing order for the municipalities over time. For each row (year) in the cumulative matrix, visualize the births (population) and create an image (frame) of the plot. Finally create a GIF (Graphics Interchange Format) of the frames generated. (The GIF itself was created using an external program, such as GIMP (GNU IMAGE MANIPULATION PROGRAM), Photoshop, etc)

Listing 19: Creating an animation of births over time

```
def animatePopulationOverTime(df, year_start, year_end, display=False):
    years_of_interest = sorted([int(year) for year in set(df.Year) if year is not '?' and int(year) >= year_start and int(year) <= year_end])
    data = np.zeros((len(years_of_interest), len(df_municipalities)), dtype=np.uint64)

    # 'Populate' the matrix with births.
    for i, year in enumerate(years_of_interest):
        records = df.loc[df.Year == str(year)]
        for m, c in zip(records.Municipality, records.Births):
            data[i,MUNICIPALITIES_IDX[m]] += c

    for i, population in enumerate(data.cumsum(axis=0)):
        year = years_of_interest[i]
        visualize(df_municipalities, population, '{}'.format(year), False, 'birth_{}'.format(year), not display)
```

As for execution time for the various jobs executed, Spark performs better if compared against some of the heavier MRJob tasks. The Spark job performed multiple tasks, while the single MRJob analysis of births were similar complexity to that of each task performed in Spark.

Library	Tasks	Time (sec)
Regular MapReduce	Preprocessing	72.10
MRJob	Analyze Population	82.30
MRJob	Analyze Births	99.00
SPARK	Analyze Religion, Jobs and Gender	281.40

Table 1: Profiling various jobs running in Hadoop, using the Unix *time* command

In table 1, analyze births took about 99 seconds to finish, while Spark ran a job like this three times (similar complexity) in only 281.40 sec. So in comparison, if the Spark tasks were run in MRJob it should have taken about ≈ 297 seconds in total to finish.

5 FURTHER WORK

Certain features in the dataset had a lot of typos and abbreviations, more preprocessing could have been done to these fields. This would have allowed for better accuracy when visualizing the results.

There were a total of 981 different occupations that had a frequency of at least 100 each, a lot of these are most likely very similar to each other.

With historical weather data (eg, fig 11) it would be possible to do some more analysis, such as finding correlation between densely populated areas and the weather conditions in those areas.

Occupation	Frequency
Farmer	278858
Datter	216859
Sn	216042
Fisher	99766
Housewife	82394
Maid	64788
Barn	56197
Husgjerning	34741
Husmor	21943
D	19204

Table 2: Raw Field of Work

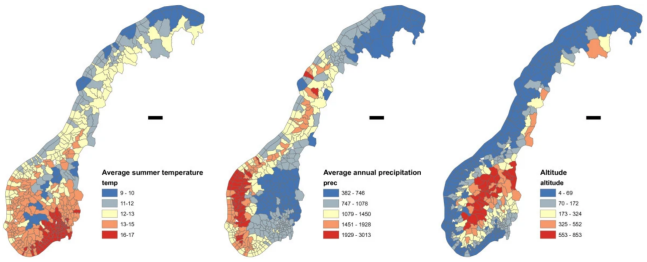

Figure 11: Temperature, Precipitation and Altitude Visualization. [7]

As mentioned in the previous preprocessing steps here 3.4, there are ≈ 86 % correct birthplaces. Which implies that there is ≈ 14 % faulty birthplaces, some of these are places outside of Norway.

Birthplace	Frequency
Sverige	25117
Ons	7600
B	6790
Liten	5796
?	5551
Inderen	4554
Herl	4345
Ed	3543
Krania	3378
Ve	2796

Table 3: Unknown or Foreign Birthplaces

Further investigation of birthplaces in table 3 would also help boosting the accuracy of the visualizations.

6 CONCLUSION

Normally it is desirable to have much larger datasets when dealing with Hadoop, in our case we could have run everything locally without using Hadoop in the first place. But this does not imply that Hadoop would be unsuitable for census data analysis, if you have enough data it is beneficial to use Hadoop. The distributed computing that Hadoop supports is a great advantage when you need to do lots of preprocessing and aggregation. This project has established somewhat of a baseline for using Hadoop in a census related context, and can be extended further to involve more complexity.

Historians and statisticians can benefit from visualizations like the ones generated in this project. These visualizations provide more depth than simple graphs, identifying patterns based on the geographical locations.

7 RAW RESULT FROM MAPREDUCE & SPARKSQL

Raw data generated from the MapReduce and SparkSQL steps, displaying the first six lines.

Municipality	Gender	Frequency
Agdenes	Female	2246
Agdenes	Male	2176
Alstahaug	Male	2586
Alstahaug	Female	2619
Alta	Male	2653
Alta	Female	2583

Table 4: Census Gender Distribution

Municipality	Frequency
Agdenes	4422
Alstahaug	5205
Alta	5236
Alvdal	4519
Andøy	4726
Aremark	1876

Table 5: Census Population Distribution

Municipality	Religion	Frequency
Agdenes	Atheist	5
Agdenes	Methodism	3
Alstahaug	Atheist	1
Alstahaug	Methodism	2
Alstahaug	Lutheranism	3
Alta	Lutheranism	1

Table 6: Census Religion Distribution

Municipality	Occupation	Frequency
Agdenes	Farmer	875
Agdenes	Fisher	366
Agdenes	Housewife	70
Agdenes	Maid	109
Alstahaug	Farmer	200
Alstahaug	Fisher	906

Table 7: Census Field of Work Distribution

Year	Birth Place (Municipality)	Frequency
1800	Askvoll	1
1800	Bergen	1
1800	Gjøvik	1
1800	Hammerfest	1
1800	Holmestrand	1
1800	Kvinnherad	1

Table 8: Census Birth Distribution

REFERENCES

[1] 2020. Digitalarkivet. https://www.digitalarkivet.no/ [Online; accessed 17-April-2020].

[2] 2020. List of former municipalities of Norway — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/List_of_municipalities_of_Norway [Online; accessed 26-April-2020].

[3] 2020. List of former municipalities of Norway — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/List_of_former_municipalities_of_Norway [Online; accessed 15-April-2020].

[4] Dharmendra Agawane, Rohit Pawar, Pavankumar Purohit, and Gangadhar Agre. 2016. Finding Insights Hadoop Cluster Performance Analysis over Census Dataset Using Big-Data Analytics. *INTERNATIONAL JOURNAL OF RESEARCH IN ADVANCE ENGINEERING* 2 (05 2016), 28. https://doi.org/10.26472/ijrae.v2i3.52

[5] Apache Software Foundation. [n. d.]. Hadoop. https://hadoop.apache.org

[6] Aleksey Bilogur. 2020. Geoplot, Geospatial Data Visualization. https://github.com/ResidentMario/geoplot

[7] Morten H Vatn Geir Aamodt, May-Bente Bengtson. 2013. Can temperature explain the latitudinal gradient of ulcerative colitis? Cohort of Norway. https://bmcpublichealth.biomedcentral.com/articles/10.1186/1471-2458-13-530/figures/2 [Online; accessed April 23, 2020].

[8] GeoPandas. 2020. GeoPandas, Python tools for geographic data. https://github.com/geopandas/geopandas

[9] John D Hunter. 2007. Matplotlib: A 2D graphics environment. *Computing in science & engineering* 9, 3 (2007), 90–95.

[10] Kiran. 2017. Census Data Analysis Using Apache Hive Zeppelin. https://acadgild.com/blog/census-data-analysis-using-apache-hive-zeppelin

[11] Wes McKinney et al. 2010. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, Vol. 445. Austin, TX, 51–56.

[12] Aleksandar Nikolić, Goran Sladic, Branko Milosavljević, Stevan Gostojić, and Zora Konjovic. 2014. Hadoop and Pig for Internet Census Data Analysis.

[13] Travis E Oliphant. 2006. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA.

[14] Kartverket (Norges Geografiske Oppmåling). 2018. Norske fylker og kommuner illustrasjonsdata 2018 (klippet etter kyst). https://kartkatalog.geonorge.no/metadata/norske-fylker-og-kommuner-illustrasjonsdata-2018-klippet-etter-kyst/cbbdf78c-fa3a-48bf-8f3f-9eec74e428fd

[15] Guido Van Rossum and Fred L. Drake. 2009. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

[16] Tomasz Wiktorski. 2019. *Introduction*. Springer International Publishing. https://doi.org/10.1007/978-3-030-04603-3_2

[17] Yelp. 2020. MRJob, the Python MapReduce library. https://github.com/Yelp/mrjob

[18] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. 2016. Apache spark: a unified engine for big data processing. *Commun. ACM* 59, 11 (2016), 56–65.