DATA ANALYSIS OF MEDICAL DATA

BDTT ASSIGNMENT

AAMIR NAWAB

Introduction

The overall purpose of this report is to explain the analysis done of the medical data supplied. The report will be alongside two .ipynb files, where one performs an analysis of medical data using databricks and the other an .ipynb that executes the entire process of loading medical data and exporting the result data for visualisation. In addition, there is terminal script that executes the entire process of loading medical data and exporting the result data for visualisation. The aim of this report is to explain the loading, cleaning and the analysis of data.

Introduction

Pre-processing and loading data

First the extracting of the supplied data is required. The terminal script and the .ipynb script both use the command:

tar xzvf ~/assignment/Aamir_Nawab/juptyer_notebook/data.tgz -C ~/assignment/Aamir_Nawab/juptyer_notebook

The supplied data provides a database setup, where the database assignment is created along with empty tables. All it requires is to execute the following command:

mysql -uroot -pcloudera < ~/assignment/Aamir_Nawab/juptyer_notebook/db_setup.sql

The requirement is to perform an analysis on the medical data provided. To perform an analysis there are 3 tables, diagnoses, hearing_evaluation and imaging that need to be loaded on to the MySQL database. Since the data that is being provided has been extracted from a raw dataset, it requires some cleaning. The reason for this is that the data has numerous NULL values that contradict the primary key constraints in the MySQL database. There cannot be duplicate or nulls values for primary keys. Therefore, attempting to load the data without filtering out, where there are primary key violations, and null values will only result in a 'Violation of PRIMARY KEY constraint' error. For that reason, any violation of primary key constraints must be filtered out. For the medical data, this is done by using the following code in the terminal:

cat ~/assignment/Aamir_Nawab/juptyer_notebook/imaging.sql | sort -u -t ',' -k 1,2 -s | grep -v 'NULL' | mysql -uroot -pcloudera assignment

(The same is done for hearing_evaluation.sql and diagnoses.sql)

The terminal command above filters outs out records where primary key is violated, as well as null values. In addition, the data will be loaded into the already created MySQL assignment database.

Once all three .sql files were loaded onto MySQL, it was then loaded onto the Hadoop structure. To do this, Sqoop was used to import all tables into HDFS. The following code was used:

***sqoop import-all-tables ***
***--connect jdbc:mysql://localhost/assignment ***
***--username root --password cloudera ***
***--fields-terminated-by "\t" ***
--warehouse-dir /assignment --as-parquetfile --driver com.mysql.jdbc.Driver --fetch-size -2147483648

The terminal command will import all tables with Sqoop fields terminated with "\t" and stored in assignment directory as parquet files. The purpose of using Apache Sqoop in this case, is to import data using the Hadoop MapReduce program and store the data into HDFS as parquet files. The reason for a parquet file rather the default comma-delimited files is the parquet format allows for a better compression where space saving on a Hadoop cluster is noticeable as well as reduced bandwidth requirement to read input i.e. parquet files save storage and processes faster. These were then converted into csv files using the following command:

mysql -uroot --password=cloudera --database=assignment --execute="SELECT 'patient_id', 'imaging_age', 'modality' UNION SELECT * INTO OUTFILE '/tmp/imaging.csv' FIELDS TERMINATED BY ',' ENCLOSED BY '\"' ESCAPED BY '\\\' LINES TERMINATED BY '\n' FROM imaging;"

(Same was done for diagnoses and hearing_evaluation, the difference being the columns selected)

CSV files are comma-separated values that are easy to programmatically manipulate i.e. files that make it easier to preform data analysis. Three sets of data analysis were performed on the medical data. One using the impala shell on the terminal command and the other two using PySpark(one on databricks and the other on juptyer notebook)

Loading data into Pyspark using juptyer notebook

The analysis of medical data was done by using PySpark in Jupyter Notebook. To perform analysis on the data, it will require to load the csv files onto a DataFrame. In the Jupyter Notebook, schemas were created for each of the csv files to be loaded into. Creating schemas is not a necessary requirement but makes each of the table loaded onto the DataFrame easier to understand improves readability and for the client compared to the default headers c0, c1, …. Consequently, the three csv files (imaging, diagnoses, hearing_evaluation) were loaded onto three different DataFrames using the relevant schema that was created for them.

Example of the schemas:

```
#Create a Schema for loading Diagnoses.csv into a DataFrame
diagnoses_schema = StructType ([StructField ("patient_id", StringType(), True),\
                                StructField ("diagnosis_code",StringType(), True),\
                                StructField ("diagnosis_age",DoubleType(), True)])
```

To load csv into a PySpark DataFrame it requires to import SQLContext and execute the following code:

diagnoses_df = sqlCtx.read.format('com.databricks.spark.csv')
.options(header = 'true').schema(diagnoses_schema)
.load('file:/home/cloudera/assignment/Aamir_Nawab/juptyer_notebook/diagnoses.csv')

This will simply load the diagnoses.csv file into variable name diagnoses_df as a DataFrame.

Loading data into Impala

Impala is a Massively Parallel Processing (MPP) SQL engine that was developed by Cloudera in 2012. The major benefit in using Impala is that its low latency and high-performance SQL engine allows for queries to process and analyse data faster with the only condition being that the data must be stored on Hadoop clusters.

Since the data is on the HDFS from using sqoop import all table from before, the Impala shell can be used to create a database. The database is simply a namespace where it helps to organise the tables. The script uses the following code:

impala-shell -q "CREATE DATABASE IF NOT EXISTS assignment

This simply creates a storage directory in HDFS as well as adds the database definition to the metastore. In addition to this, three external tables were created again using the impala shell, through the following code:

impala-shell -q "USE assignment; CREATE EXTERNAL TABLE imaging LIKE PARQUET '/assignment/imaging/*.parquet' STORED AS PARQUET LOCATION '/assignment/imaging/';"

This just creates a subdirectory in HDFS and loads parquet files onto the tables. However, loading data just simply adds files to the directory. This means files are just moved from one place to another and the data is not validated on insert. Therefore, it requires data validation and it can be done using the following code:

impala-shell -q "USE assignment; INVALIDATE METADATA diagnoses;

Once this is done the impala shell can be used to perform queries on the data. This means the analysis performed on the medical data can be done through using the impala shell through queries.

Loading data into Databricks

To use the three CSV files, it requires to be uploaded onto FileStore where a table is needed to be created for each of the CSV files. Once that is done, a cloud spark cluster will need to be created to use the databricks notebook. The following code is required to write csv file as spark DataFrame:

```
file_directory = "/FileStore/tables/imaging-2.csv/"
file_type = "csv"
infer_schema = "true"
first_row_is_header = "true"
delimiter = ","

imaging_df = spark.read.format(file_type) \
  .option("inferSchema", infer_schema) \
  .option("header", first_row_is_header) \
  .option("sep", delimiter) \
  .load(file_directory)

# This displays the dataframe created
display(imaging_df)
```

Problem statement 1

DataFrame solution

The client wished to know the 5 most common diagnosis code. The dataframe where the diagnoses csv file was loaded onto, requires to group the column 'Diagnosis_code', perform a count using the count() function and sort the order in descending to display the diagnoses codes and their frequencies, as shown below.

```
most_common_diagnosis_codes = diagnoses_df.groupby("diagnosis_code").count().sort("count", ascending=False)
most_common_diagnosis_codes.show(5)
```

As a result, it displays the following:

```
+--------------+------+
|diagnosis_code| count|
+--------------+------+
|         V20.2|476228|
|    382.900000|272497|
|    389.900000|256345|
|    381.810000|213910|
|    465.900000|175831|
+--------------+------+
only showing top 5 rows
```

RDD Solution

Before any analysis can be done using RDDs, the csv file must be loaded into an RDD. This can by the following code:

```
Imaging = sc.textFile("file:///home/cloudera/assignment/Aamir_Nawab/juptyer_notebook/imaging.csv")
```

In addition to this, some cleaning of the data required. The following code removes the header, replaces the quotes with empty space and establishes a comma delimiter.

```
imaging_rdd = imaging.filter(lambda row: row != imaging_header ).map(lambda x: x.replace('"','').split(","))
```

Once this done, the 5 most common diagnoses can be retrieved. To do this, assign the value 1 to each diagnosis code for the diagnoses RDD and use the reduceByKey() function to combine values with the same key in which

will also count the frequency of each diagnosis code. In addition, use the sortBy function, to sort in descending order to give the most frequent diagnosis code. The code:

most_common_diagnosis_codes_rdd =diagnoses_rdd.map(lambda x: (x[1],1)).reduceByKey(lambda x1,y1: x1+y1).sortBy(lambda x: x[1], False)

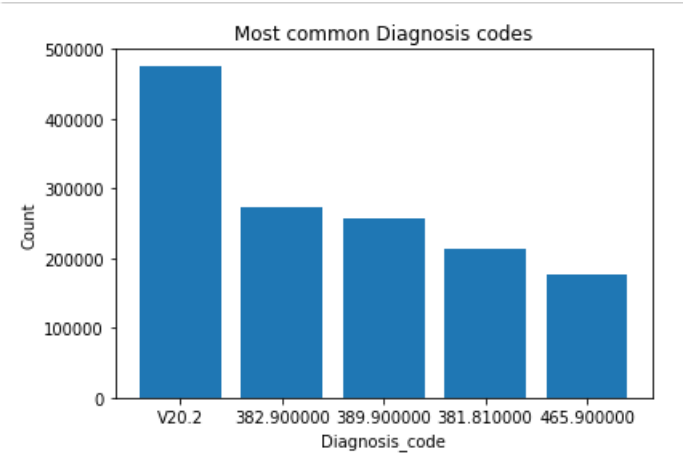
Impala Solution

The following was executed on the terminal command line:

impala-shell -B --quiet -q "use assignment; SELECT diagnosis_code, count(diagnosis_code) AS frequency FROM assignment.diagnoses GROUP BY diagnosis_code ORDER BY COUNT(diagnosis_code) DESC LIMIT 5;"

This is very similar to the DataFrame solution, except it is in SQL format. The query simply selects two columns, 'diagnosis_code' and column where count function is performed on diagnosis from the diagnoses table. Then a group by clause is used on the diagnosis code as well as an order by clause, which sorts the column in descending order.

Problem Statement 1 Visualisation



Analysis

The bar chart clearly shows the most common diagnosis code is V20.2 by far. The total count being 476,228 the V20.2 diagnosis code was used. The total number of distinct patients in the dataset is 155,029. This means that for every 1 patient, they were diagnosed with the code V20.2, around 3 times. When compared to next most common diagnosis code 382.9, with the total count being 272,497. This means that V20.2 code was used 74.76% more times than the next most common diagnosis code, 382.9. However, the dataset shows there has been 12,816,362 patients diagnosed. This means that the diagnosis code V20.2 only represents 3.7% of all patient's diagnosis.

Problem statement 2

DataFrame Solution

A diagnosis (i.e. a diagnosis code) can be assigned to a patient directly at the time of a hearing evaluation. The client desires to know the five most common diagnosis codes at this point.

For the Spark DataFrame, an inner join on the diagnoses and hearing_evaluation DataFrames. The join on both DataFrames' patient_id and join on diagnosis age and hearing_evaluation age as shown below.

diagnoses_df.join(hearing_evaluation_df,(diagnoses_df.patient_id==hearing_evaluation_df.patient_id) & (diagnoses_df.diagnosis_age==hearing_evaluation_df.evaluation_age) ,'inner')

In addition, group the column ‘Diagnosis_code’, perform the count() and sort ()function.

RDD solution

For the RDD solution it required to map out patient id, age and diagnosis code from the diagnoses RDD and map out patient id, age and unilateral_bilateral from hearing evaluation and join both rdds together, as shown below.

diagnoses_rdd.map(lambda x: ((x[0],x[2]),(x[1]))).join(hearing_evaluation_rdd.map(lambda x: ((x[0],x[1]),(x[3])))))

Like problem statement 1, assign key value pair to diagnosis codes, combine values with the same key and then count frequency.

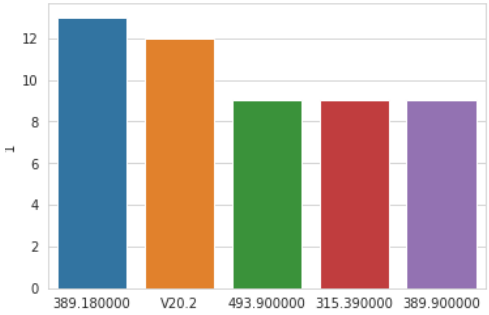
most_common_diagnosis_codes_at_hearing_rdd = join_rdd.map(lambda x: (x[1])).map(lambda x: (x[0])).map(lambda x: (x,1)).reduceByKey(lambda x1,y1: x1+y1).sortBy(lambda x: x[1], False)

Impala Solution

Similar query to problem statement 1, the only difference being a join on hearing evaluation’s patient_id and evaluation_age is made.

***impala-shell -B --quiet -q "use assignment; SELECT a.diagnosis_code, count(a.diagnosis_code) AS frequency FROM assignment.diagnoses a JOIN hearing_evaluation ON a.diagnosis_age = hearing_evaluation.evaluation_age and a.patient_id=hearing_evaluation.patient_id GROUP BY a.diagnosis_code ***
ORDER BY COUNT(a.diagnosis_code) DESC LIMIT 5;"

Visualisation



Analysis

The bar chart shows a representation of diagnosis codes given at the time of the hearing evaluation, with 389.18 diagnosis code being the most common of a total number of 13 times. These diagnoses is most likely a result of symptoms developing, since the diagnosis is being made at the time of hearing evaluation.

Problem Statement 3

DataFrame Solution

The client wishes to know the highest number of diagnoses that has been assigned to a signal patient to ensure that the dataset is ‘sensible’. To perform this type of analysis it will require to group the column name ‘patient_id’ and count the diagnosis codes assigned to them, as shown below.

diagnoses_to_a_single_patient = diagnoses_df.groupBy("patient_id").count().sort("count", ascending=False)
diagnoses_to_a_single_patient.show(1)

RDD solution

Assign the value 1 to each patient_id for the diagnoses RDD, use the reduceByKey() function to combine values with the same key in which will also count the frequency and sort in descending order to get the highest number of diagnoses perform on patients, as shown below.

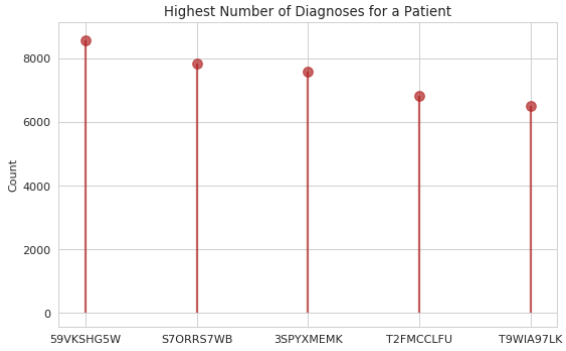
diagnoses_to_a_single_patient_rdd = diagnoses_rdd.map(lambda x: (x[0],1)).reduceByKey(lambda v1,v2: v1+v2).sortBy(lambda x: x[1], False)
diagnoses_to_a_single_patient_rdd.take(1)

Impala Solution

Select patient_id count(patient_id), use group by clause on patient and sort in descending order using the order by clause.

***impala-shell -B --quiet -q "use assignment; SELECT patient_id, count(patient_id) AS highest_diagnosis FROM assignment.diagnoses GROUP BY patient_id ***
ORDER BY COUNT(patient_id) DESC LIMIT 1;"

Visualisation



Analysis

Patient 59VKSHG5W is most diagnosed patient with a total count of 8,557. A further analysis was done, as the chart represent the 5 most diagnosed patient. This allows for a better comparison to see whether a dataset is sensible.

Problem statement 4

DataFrame Solution

The client is interested in the total number of (different) people with a hearing problem in comparison to the total number of people who have had a hearing evaluation.

Total number of different people who had a hearing evaluation is a simply performing .distinct() on patient_id, on the hearing evaluation DataFrame

unique_hearing_evaluation =
hearing_evaluation_df.select("patient_id").distinct()print(unique_hearing_evaluation.count())

The number of people with hearing problems requires to filter hearing_evaluation DataFrame where the column severity_of_hearing_loss contain the values; Slight, Mild, Moderate, Moderately Severe, Severe, or Profound. This was done using the .where() and .isin() function for the Spark DataFrame, as shown below.

total_hearing_problems = hearing_evaluation_df.where((hearing_evaluation_df["severity_of_hearing_loss"]).isin ('Moderate','Mild','Slight','Moderately Severe','Severe','Profound'))

However, to retrieve number of different people with hearing problems this requires the column patient_id to have unique values. Therefore, the distinct() function needs to be used as shown below.

distinct_hearing_problems= total_hearing_problems.select("patient_id").distinct()

To now find the percentage, it requires to convert the count of distinct_hearing_problems and unique_hearing_evaluation to be converted to floats and divided by each other as shown below.

percentage_of_hearing_problems = float(distinct_hearing_problems.count()))/float(unique_hearing_evaluation.count()) * 100

RDD solution

Total number of different(unique) people who had hearing evaluation requires mapping out the column 'patient_id' and performing the distinct() and count() function on them.

unique_hearing_evaluation_rdd = hearing_evaluation_rdd.map(lambda x: x[0]).distinct().count()
print(unique_hearing_evaluation_rdd)

Total number of people who have hearing problems requires to put the severity_of_hearing_loss values in a list. The list them can used to filter column severity_of_hearing_loss using lambda. In addition, the .distinct() can be used to get the total number of different people with hearing problems.

severity_of_hearing_loss = ['Moderate','Mild','Slight','Moderately Severe','Severe','Profound']
total_hearing_problems_rdd=hearing_evaluation_rdd.filter(lambda x: x[2] in severity_of_hearing_loss)
distinct_hearing_problems_rdd= total_hearing_problems_rdd.map(lambda x: x[0]).distinct()

To get percentage, divide the total number of different people with hearing problems by total number of different(unique) people who had hearing evaluation

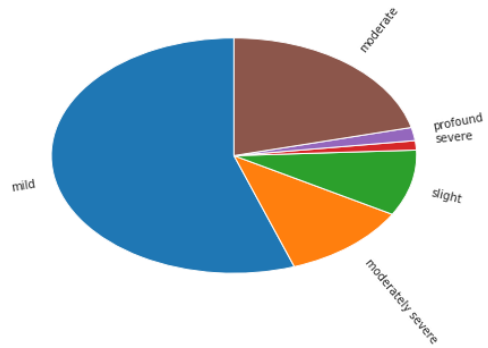
percentage_of_hearing_problems =
float(distinct_hearing_problems_rdd.count()))/float(unique_hearing_evaluation_rdd) * 100

Impala Solution

impala-shell -B --quiet -q "USE assignment; SELECT a.item, a.total_hearing_problem, b.total_patient_id, round(100 *(a.total_hearing_problem / b.total_patient_id), 2) AS 'percentage(%)' FROM (SELECT 'result' AS item, COUNT(DISTINCT(patient_id)) AS 'total_hearing_problem' FROM assignment.hearing_evaluation WHERE severity_of_hearing_loss IN ('Mild', 'Moderate', 'Moderately Severe', 'Slight', 'Profound', 'Severe')) AS a INNER JOIN (SELECT 'result' AS item, COUNT(DISTINCT(patient_id)) AS 'total_patient_id' FROM assignment.hearing_evaluation) AS b ON b.item = a.item;"

This requires selecting the number of distinct patient_id from table hearing_evaluation, filtering the column severity_of_hearing_loss for values related to hearing loss. In addition, selecting the total distinct patient_id and to get the percentage divide unique patients with hearing loss by total distinct patients who had evaluation * 100.

Visualisation




```
+-----------------------+-----+
|severity_of_hearing_loss|count|
+-----------------------+-----+
|               Profound|  323|
|      Moderately Severe| 2023|
|               Moderate| 3777|
|                 Slight| 1619|
|                 Severe|  229|
|                   Mild| 9887|
+-----------------------+-----+

Mild: 55.364543
Slight: 9.065965
Moderate: 21.150185
Moderately Severe: 11.328256
Severe: 1.282338
Profound: 1.808713
```

Analysis

To perform this analysis the distinct patient id was counted to give the total number of different people who had an evaluation. This gave a count of 33,892. Then a count of unique patient IDs of those who had hearing problems was performed. This gave a total count of 11,417. In addition to this, the client desired the percentages of those diagnosed with hearing problems compared them having hearing evaluation. This gave a percentage of 33.69%.

The pie chart represents an additional analysis undertaken. The addition analysis shows the percentage of each type hearing loss for those diagnosed with hearing loss. This shows that just over half the patients have mild hearing problems that have been diagnosed with a hearing problem.

Problem Statement 5

DataFrame Solution

The client wishes to analyse the the imaging scans for patients with hearing loss. In the analysis above, it has already been established that total number patients with hearing loss is 11,417. To find the number of patients with hearing loss that had scans performed on them requires to join the imaging dataframe and hearing evaluation dataframe on patient_id.

join_hearing_and_imaging =
distinct_hearing_problems.join(imaging_df,(distinct_hearing_problems.patient_id==imaging_df.patient_id),'inner')

Working out the perecentage of investigations performed on those with hearing problems

average = float(join_hearing_and_imaging.count())/float(distinct_hearing_problems.count())

percentage_average = average * 100

RDD Solution

Number of patients with hearing loss that scans performed on them. This requires counting the number of different people with hearing problems and joining it on imaging rdd on patient_id

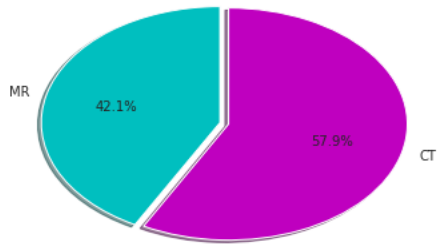
join_hearing_and_imaging_rdd = total_hearing_problems_rdd.map(lambda x: (x[0],1)).reduceByKey(lambda x1,y1: x1+y1).join(imaging_rdd.map(lambda x: ((x[0]),(x[1])))))

Working out the perecentage of investigations performed on those with hearing problems
average = (join_hearing_and_imaging_rdd.count()/distinct_hearing_problems_rdd.count()) *100

Impala Solution

This requires selecting the number of distinct patient_id from table hearing_evaluation, filtering the column severity_of_hearing_loss for values related to hearing loss. In addition, the select statement requires an inner join on imaging patient_id, which gets the total number of people with hearing problems that have had scans. The average is simply patients with hearing problems scanned divided by patients with hearing problems

```
impala-shell -B --quiet -q "USE assignment; SELECT A.item, a.total_investigation, b.total_patient_id, \
ROUND(b.total_patient_id / a.total_investigation, 2) AS 'average' \
FROM (SELECT 'result' AS item, COUNT(DISTINCT(h.patient_id)) AS 'total_investigation' \
FROM assignment.hearing_evaluation h INNER JOIN imaging im ON h.patient_id = im.patient_id \
WHERE h.severity_of_hearing_loss IN ('Mild', 'Moderate', 'Moderately Severe', 'Slight', 'Profound', 'Severe')) \
AS A INNER JOIN (SELECT 'result' AS item, COUNT(DISTINCT(patient_id)) AS total_patient_id \
FROM assignment.hearing_evaluation \
WHERE severity_of_hearing_loss IN ('Mild','Slight', 'Moderate', 'Moderately Severe', 'Severe', 'Profound')) AS b ON
b.item = A.item;" \
```



Analysis

The total number of investagtions performed were 627. This means 5.5% of people with hearing loss had imaging scans performed. Further analysis can be done, where the percentages of the different imaging scans performed can be analysed. The pie chart aims to represent this.The analysis shows that the CT scans are used the most, where 58% of scans were performed on different people. However, 42% of imaging scans performed on patients with hearing problems was MR. A 16% difference shows that there isn't much difference between the amount of imaging scans performed on the patients with hearing loss.

Problem Statement 6

DataFrame solution

The following analysis retrieves the greatest number of CTs for each age group. To perform this it requires to filter the modality column in imaging where it only shows the value CT. Then the analysis it requires to round each of the imaging age using .cast(InterType()) as well as to group them to represent years and count the CT perform for each imaging age.

```
greatest_number_of_CTs_on_age = imaging_df.where(imaging_df['modality'].isin('CT'))\
.groupBy(imaging_df.imaging_age.cast(IntegerType())).count().sort('count', ascending=False)
```

RDD solution

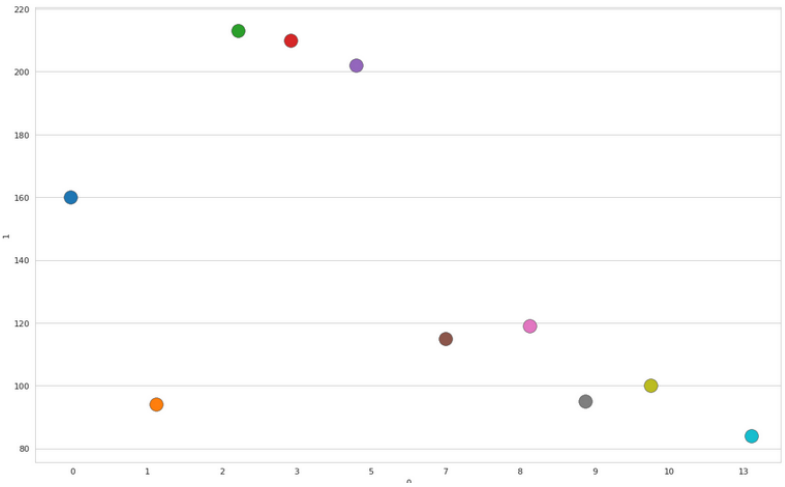
The RDD solution simply filiters imaging modality column to match value ‘CT’ , rounds age to integer and count frequency of CT for each age group.

```
greatest_number_of_CTs_on_age_rdd = imaging_rdd.filter(lambda x: x[1]=='CT').map(lambda x: (int(float(x[0])),
x[1])).map(lambda x: (x[0], 1)).reduceByKey(lambda x1,y1: x1+y1).sortBy(lambda x: x[1],False)
```

Impala solution

The required to select the count of imaging age, setting the condition where modality = CT and grouping imaging age using the floor() function to return the largest integer.

```
impala-shell -B --quiet -q "USE assignment; SELECT COUNT(imaging_age) AS total_no_CT, floor(imaging_age), modality AS
type_of_scan \
FROM assignment.imaging WHERE modality IN ('CT') GROUP BY modality, floor(imaging_age) ORDER BY total_no_CT
DESC;" \
```

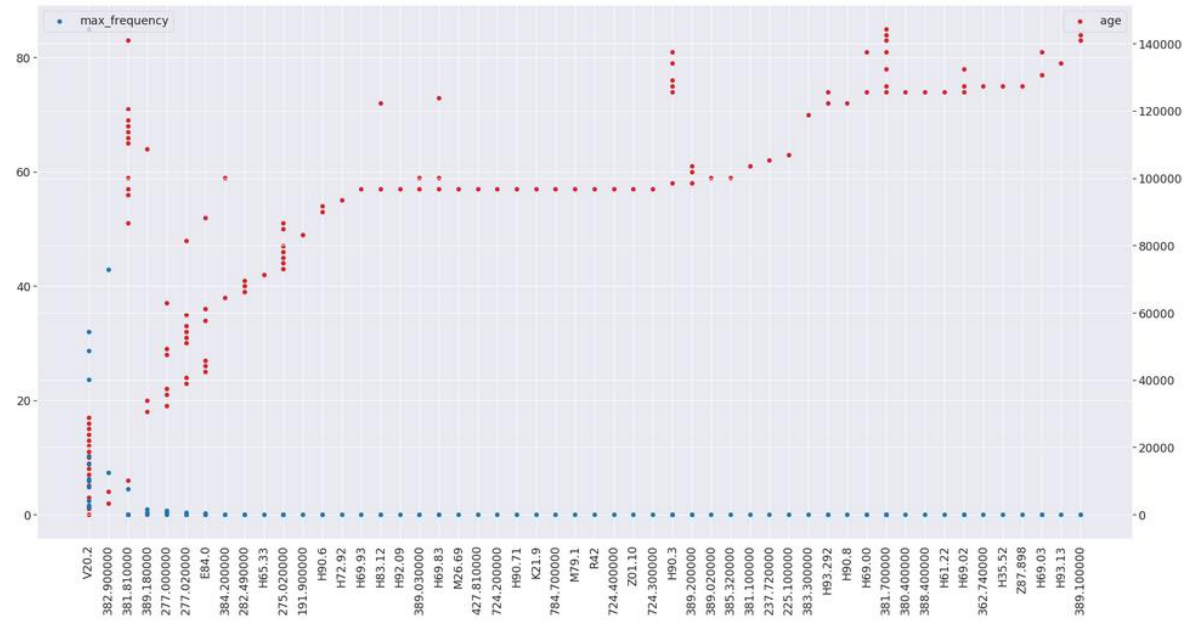


Analysis

This shows that the greatest number of CT scans in an age group is 2. The total the number of CT scans in age group 2 is, 213. However, there isn’t much difference in the highest number of CTs between three age groups. Age groups 2,3,5 have the following total CT scans 213,210,202. The total number of CT scans perform is 1,911. This means that a with the three highest age groups combined makes up around 33% of all CT scan performed. With the highest age group of CT scans making up around 12% of all CT scans.

Personal statement 7

For each year group, the client would like to visualize the most frequent diagnosis code. The scatter chart hopes to represent that.



DataFrame Solution

This required to use the HiveContext to able use the function max(). This required to group the rounded diagnosis age and diagnosis codes from the diagnoses dataframe as well as count the frequency. Then it required to partition each age group, only retrieving the max number of the frequncy of a diagnosis as well as the diagnosis code itself

```
frequency_of_diagnosis_code =
diagnoses_df_1.groupby(diagnoses_df_1.diagnosis_age.cast(IntegerType()),"diagnosis_code").count().sort('count',
ascending=False)\
.withColumnRenamed('CAST(diagnosis_age AS INT)','age_group')
most_frequent_diagnosis_code = frequency_of_diagnosis_code.withColumn("maxrow_num",
func.max("count").over(Window.partitionBy("age_group")))\
.where(func.col("count") == func.col("maxrow_num")).drop("maxrow_num").sort('count', ascending=False)
```