DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Session: Aug 2020 – Dec 2020

UE18CS322 – BIG DATA

PROJECT REPORT

TOPIC: Health Care Analysis

TEAM MEMBERS:

1. Sumukh R R -PES2201800078 (SEC C)
2. K Rahul Reddy -PES2201800267 (SEC A)
3. Pramath R Rao -PES2201800364 (SEC C)
4. Nandha Kumar Gowda -PES2UG19CS805(SEC E)

INTRODUCTION:

The introduction of clinical information systems (CIS) in Intensive Care Units (ICUs) offers the possibility of storing a huge amount of machine-ready clinical data that can be used to improve patient outcomes and the allocation of resources, as well as suggest topics for randomized clinical trials. Clinicians, however, usually lack the necessary training for the analysis of large databases. In addition, there are issues referred to patient privacy and consent, and data quality. Multidisciplinary collaboration among clinicians, data engineers, machine-learning experts, statisticians, epidemiologists and other information scientists may overcome these problems.
Health care analysis is very important in this present situation and to do that we have to build models and analyse various aspects .It is very important to first analyse the user/patient diagnosis data because many insights can be found from that and lot of things can be done or brought to conclusion. We can study the occurrence of the fever based on the symptoms and based on these things we can declare other person with the same symptoms as particular disease.
To maintain a healthy and normal life its very important for us to analyse our body and take care of it.

DATASET

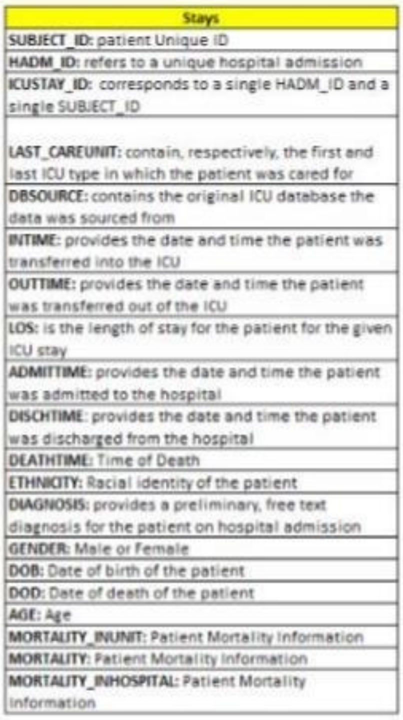
MIMIC-III (‘Medical Information Mart for Intensive Care’) is a large, single-center database comprising information relating to patients admitted to critical care units at a large tertiary care hospital. Data includes vital signs, medications, laboratory measurements, observations and notes charted by care providers, fluid balance, procedure codes, diagnostic codes, imaging reports, hospital length of stay, survival data, health.csv which contains details of hospital, footnote, measure_id and more. The database supports applications including academic and industrial research, quality improvement initiatives, and higher education coursework
5,718 records consisted of in-hospital mortality information.
188714 records were found in health.csv file.
Our study was to analyze the health.csv file.
The health.csv file is basically a dataset of hospitals in USA. It contains details of patient, hospital name, city, state, Footnote (brief report on patient), Condition.
Footnote we have numbers from 1-6 each number represents different condition of patient.

DATA PREPROCESSING

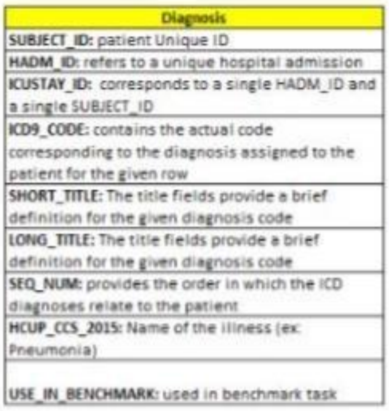
We have a total of 26 datasets downloaded from the MIMIC III database.

(2). These datasets have a mixture of both categorical and continuous variables. Since the total size of the datasets is around 43.3 GB with over a million records, we used hadoop spark SQL context to show the used cases and make changes to the csv files which are required for the project. Despite using spark, the overall data extraction process was computationally very expensive. Each patient in the database are referred to as subjects and are given a unique subject ID.We created folders for each subject ID and in each folder,west or edspecific patient information that will aid in predicting the mortality. Subject’s hospital stays, diagnosis, and events information was stored in each of those folders.The below figure shows the attributes from each of the csv files used and its descriptions. The attributes in the below figure are extracted from several datasets.The number of events prior to removing missing information was 253,116,833. Events with missing HADM_ID (5,162,703) and ICUSTAY_ID (15,735,688) information in the three csv files were removed and the total number of events after this step turned out to be 232,218,442. Every time a subject is admitted to the ICU, their ICU stay is represented with an ICU stay ID. A subject may have multiple ICU stay IDs which indicates that they have been admitted to the ICU multiple times. ICU stay medical information such as, capillary refill rate, diastolic blood pressure, fraction inspired oxygen, glascow coma scale eye opening, glascow coma scale motor response, glascow coma scale total, glascow coma scalever balresponse, glucose heartrate, height, mean blood pressure, oxygensaturation, respiratoryrate, systolicbloodpressure, temperature, weight, andpH, was extracted forevery ICU stayID in different csv files (based on the number of ICU visits) for every subject ID.

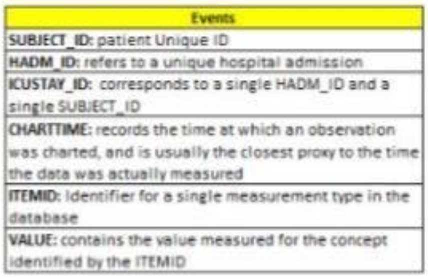
3.All the subject level ICU stay information csv files were distributed to three folders namely, Train (60% ~14,681), Validation (20% ~ 3,222),and Test (20% ~ 3,236). In this process, three separate csv files were generated namely, train_listfile, val_listfile, and test_listfile. These 3 csv files containsthe names of all the ICU stay csvfiles in the three folders and their corresponding mortality information represented by 0(dead) and 1(alive).



Stays
SUBJECT_ID: patient Unique ID
HADM_ID: refers to a unique hospital admission
ICUSTAY_ID: corresponds to a single HADM_ID and a single SUBJECT_ID
LAST_CAREUNIT: contain, respectively, the first and last ICU type in which the patient was cared for
DBSOURCE: contains the original ICU database the data was sourced from
INTIME: provides the date and time the patient was transferred into the ICU
OUTTIME: provides the date and time the patient was transferred out of the ICU
LOS: is the length of stay for the patient for the given ICU stay
ADMITTIME: provides the date and time the patient was admitted to the hospital
DISCHTIME: provides the date and time the patient was discharged from the hospital
DEATHTIME: Time of Death
ETHNICITY: Racial identity of the patient
DIAGNOSIS: provides a preliminary, free text diagnosis for the patient on hospital admission
GENDER: Male or Female
DOB: Date of birth of the patient
DOD: Date of death of the patient
AGE: Age
MORTALITY_INUNIT: Patient Mortality Information
MORTALITY: Patient Mortality Information
MORTALITY_INHOSPITAL: Patient Mortality Information



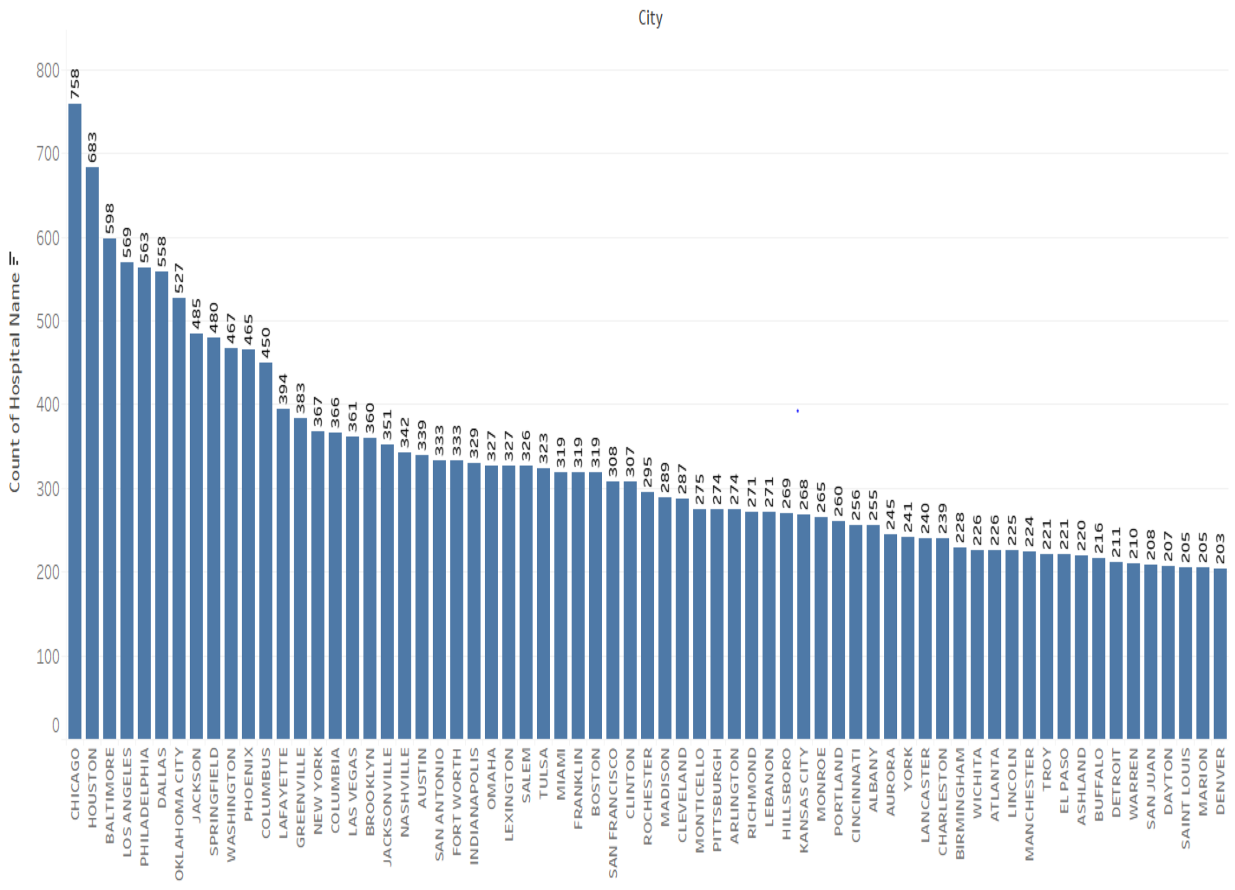
Diagnosis
SUBJECT_ID: patient Unique ID
HADM_ID: refers to a unique hospital admission
ICUSTAY_ID: corresponds to a single HADM_ID and a single SUBJECT_ID
ICD9_CODE: contains the actual code corresponding to the diagnosis assigned to the patient for the given row
SHORT_TITLE: The title fields provide a brief definition for the given diagnosis code
LONG_TITLE: The title fields provide a brief definition for the given diagnosis code
SEQ_NUM: provides the order in which the ICD diagnoses relate to the patient
HCUP_CCS_2015: Name of the illness (ex: Pneumonia)
USE_IN_BENCHMARK: used in benchmark task



Events
SUBJECT_ID: patient Unique ID
HADM_ID: refers to a unique hospital admission
ICUSTAY_ID: corresponds to a single HADM_ID and a single SUBJECT_ID
CHARTTIME: records the time at which an observation was charted, and is usually the closest proxy to the time the data was actually measured
ITEMID: Identifier for a single measurement type in the database
VALUE: contains the value measured for the concept identified by the ITEMID

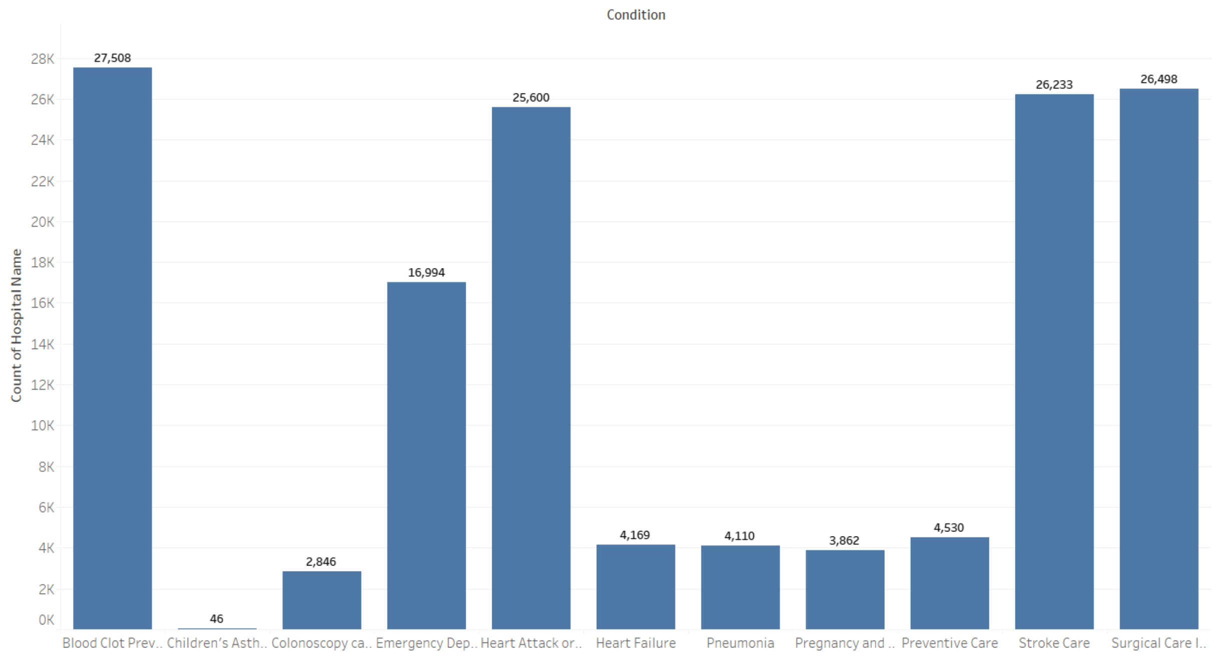
DATA VISUALIZATION

Patients in each city



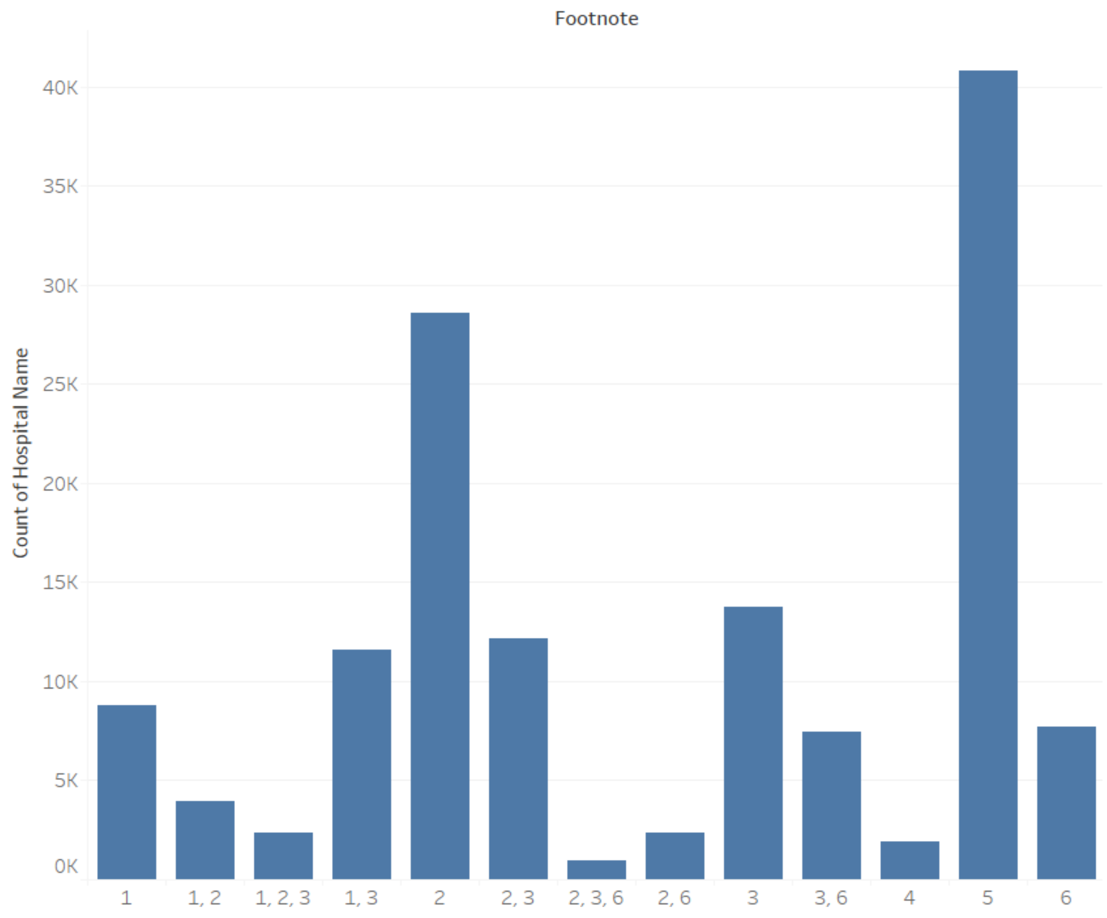
This graph shows the number of patients admitted per each city. From this graph, we can tell that Chicago has the highest number of patients per each city.

Conditions of patients



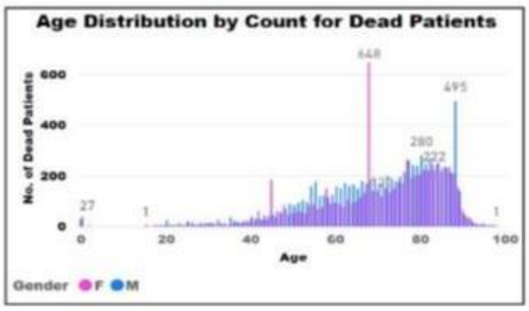
This graph shows the number of patients admitted for a given medical condition. Blood clot prevention is the highest and children's asthma is the lowest.

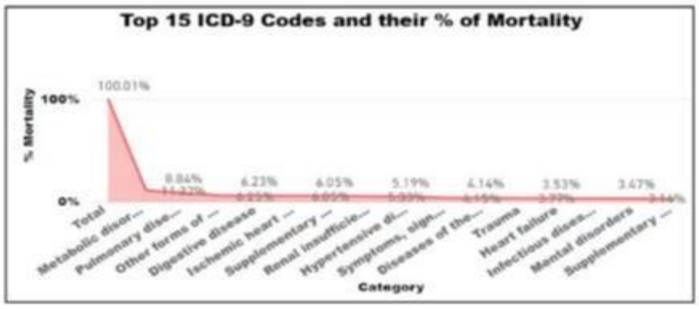
About Footnote



1 - The number of cases/patients are too few to report.
2 - The data submitted are based on the sample of cases/patients.
3 - The results are based on shorter time period than required.
4 - The data suppressed by CMS for one or more quarters.

5 - Results are not available for this reporting period.
6 - No cases met the criteria for this measure.





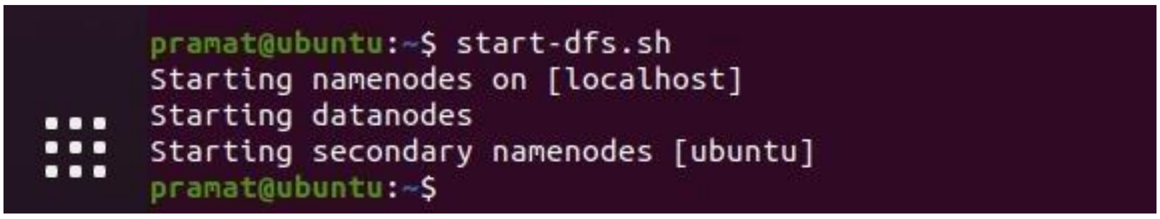
❖ As we can see most male patients have an average age of 85 and most female patients have average age of 65.
❖ Mortality % vs Category is represented in 2nd graph.As we can see 11.32% is highest with the patients having metabolic disorder and the least is 3.14% 000with the supplementary disorder.

Hadoop Distributed File System (HDFS)

HDFS is a distributed file system designed to run on commodity hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets.

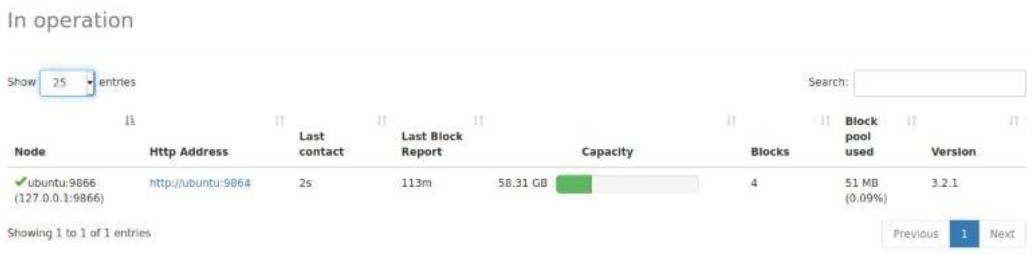
HDFS is a distributed file system that handles large data sets running on commodity hardware. It is used to scale a single Apache Hadoop cluster to hundreds (and even thousands) of nodes.

- **Hadoop cluster is up and running**



```
pramat@ubuntu:~$ start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [ubuntu]
pramat@ubuntu:~$
```

- **Our health.csv dataset was uploaded into hdfs, under user/healthdata**

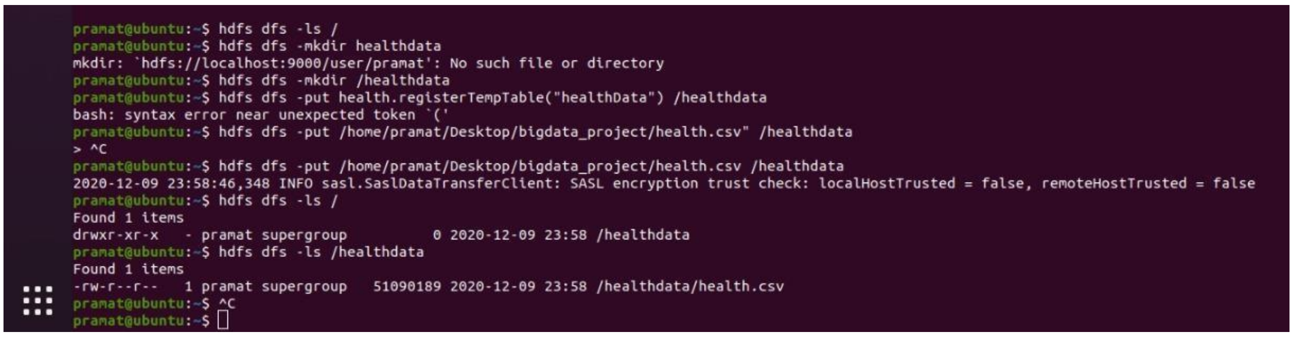


In operation

Show 25 entries Search:

Node	Http Address	Last contact	Last Block Report	Capacity	Blocks	Block pool used	Version
✓ubuntu:9866 (127.0.0.1:9866)	http://ubuntu:9864	2s	113m	58.31 GB	4	51 MB (0.09%)	3.2.1

Showing 1 to 1 of 1 entries



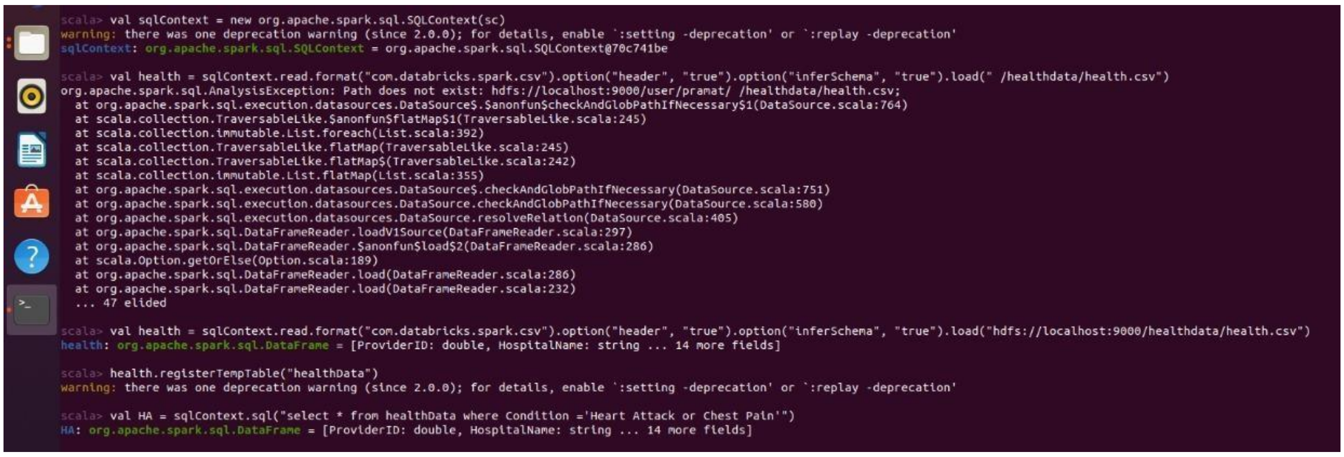
```
pramat@ubuntu:~$ hdfs dfs -ls /
pramat@ubuntu:~$ hdfs dfs -mkdir healthdata
mkdir: `hdfs://localhost:9000/user/pramat': No such file or directory
pramat@ubuntu:~$ hdfs dfs -mkdir /healthdata
pramat@ubuntu:~$ hdfs dfs -put health.registerTempTable("healthData") /healthdata
bash: syntax error near unexpected token `('
pramat@ubuntu:~$ hdfs dfs -put /home/pramat/Desktop/bigdata_project/health.csv" /healthdata
> ^C
pramat@ubuntu:~$ hdfs dfs -put /home/pramat/Desktop/bigdata_project/health.csv /healthdata
2020-12-09 23:58:46,348 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
pramat@ubuntu:~$ hdfs dfs -ls /
Found 1 items
drwxr-xr-x   - pramat supergroup          0 2020-12-09 23:58 /healthdata
pramat@ubuntu:~$ hdfs dfs -ls /healthdata
Found 1 items
-rw-r--r--   1 pramat supergroup   51090189 2020-12-09 23:58 /healthdata/health.csv
pramat@ubuntu:~$ ^C
pramat@ubuntu:~$ 
```

Apache Spark

Apache Spark is an open-source, distributed processing system used for big data workloads. It utilizes in-memory caching and optimized query execution for fast queries against data of any size.

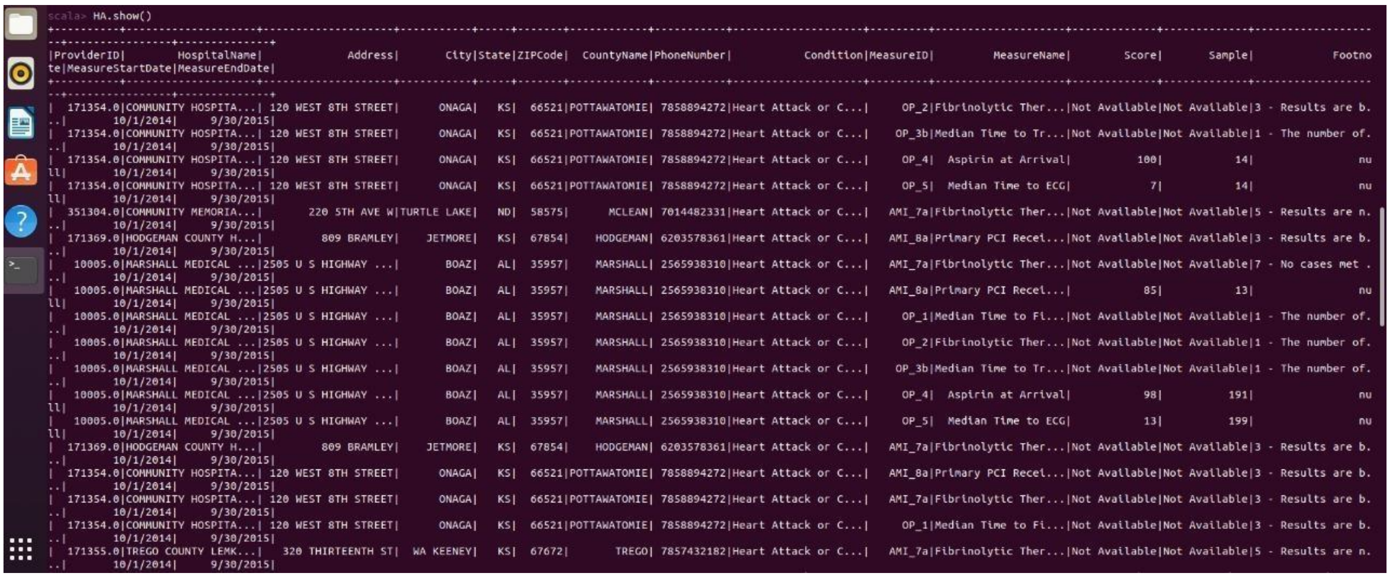
It lets us process big data sets faster by splitting the work up into chunks and assigning those chunks across computational resources.

We have used SQLcontext for building our basic commands.
SQLContext is a class and is used for initializing the functionalities of **Spark** SQL. SparkContext class object (sc) is required for initializing **SQLContext** class object. ... By default, the SparkContext object is initialized with the name sc when the **spark**-shell starts.



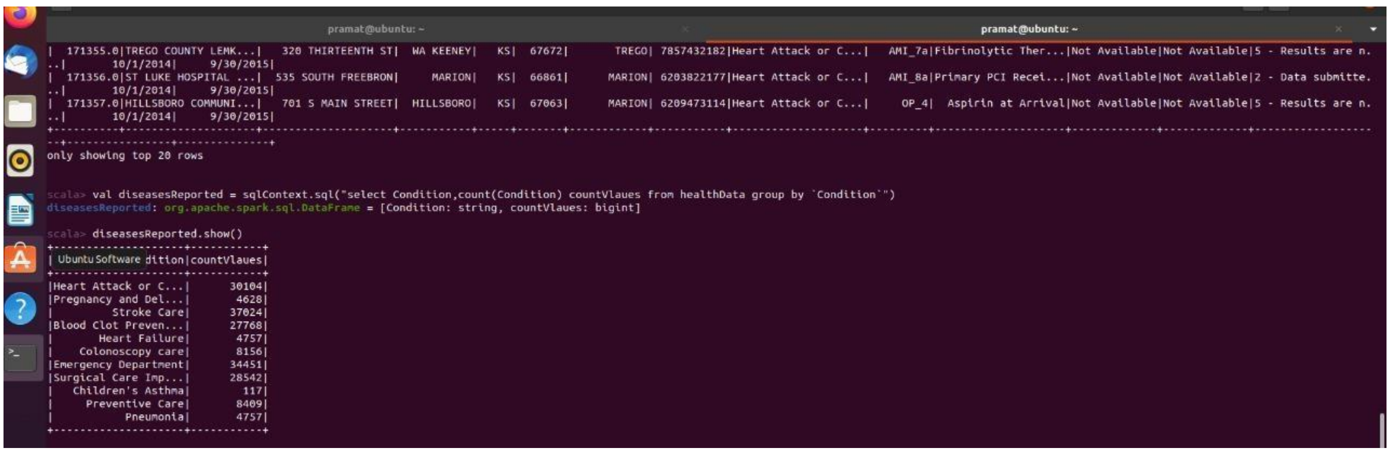
```
scala> val sqlContext = new org.apache.spark.sql.SQLContext(sc)
warning: there was one deprecation warning (since 2.0.0); for details, enable `:setting -deprecation' or `:replay -deprecation'
sqlContext: org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@70c741be

scala> val health = sqlContext.read.format("com.databricks.spark.csv").option("header", "true").option("inferSchema", "true").load(" /healthdata/health.csv")
org.apache.spark.sql.AnalysisException: Path does not exist: hdfs://localhost:9000/user/pramat/ /healthdata/health.csv;
  at org.apache.spark.sql.execution.datasources.DataSource$.$anonfun$checkAndGlobPathIfNecessary$1(DataSource.scala:764)
  at scala.collection.TraversableLike.$anonfun$flatMap$1(TraversableLike.scala:245)
  at scala.collection.immutable.List.foreach(List.scala:392)
  at scala.collection.TraversableLike.flatMap(TraversableLike.scala:245)
  at scala.collection.TraversableLike.flatMap$(TraversableLike.scala:242)
  at scala.collection.immutable.List.flatMap(List.scala:355)
  at org.apache.spark.sql.execution.datasources.DataSource$.checkAndGlobPathIfNecessary(DataSource.scala:751)
  at org.apache.spark.sql.execution.datasources.DataSource.checkAndGlobPathIfNecessary(DataSource.scala:580)
  at org.apache.spark.sql.execution.datasources.DataSource.resolveRelation(DataSource.scala:405)
  at org.apache.spark.sql.DataFrameReader.loadV1Source(DataFrameReader.scala:297)
  at org.apache.spark.sql.DataFrameReader.$anonfun$load$2(DataFrameReader.scala:286)
  at scala.Option.getOrElse(Option.scala:189)
  at org.apache.spark.sql.DataFrameReader.load(DataFrameReader.scala:286)
  at org.apache.spark.sql.DataFrameReader.load(DataFrameReader.scala:232)
  ... 47 elided

scala> val health = sqlContext.read.format("com.databricks.spark.csv").option("header", "true").option("inferSchema", "true").load("hdfs://localhost:9000/healthdata/health.csv")
health: org.apache.spark.sql.DataFrame = [ProviderID: double, HospitalName: string ... 14 more fields]

scala> health.registerTempTable("healthData")
warning: there was one deprecation warning (since 2.0.0); for details, enable `:setting -deprecation' or `:replay -deprecation'

scala> val HA = sqlContext.sql("select * from healthData where Condition ='Heart Attack or Chest Pain'")
HA: org.apache.spark.sql.DataFrame = [ProviderID: double, HospitalName: string ... 14 more fields]
```

- In first line we have created a class object (sc) which is required for initializing SQL Context class Object. By default , the SparkContext object is initialized with the name sc when the spark-shell starts.
- Next we have created a variable named health and we have initiated the variable with health.csv and the dataset is read using "com.databricks.spark.csv" which is inbuilt package in spark to read csv files.The path of health.csv which was uploaded in HDFS is given in load() command.
- We have registered the temporary table with the name healthData for health variable.
- We have extracted the name of patient and their details using the select command from sql and the condition was "Heart Attack or Chest Pain") and that was initialized to a variable named HA.



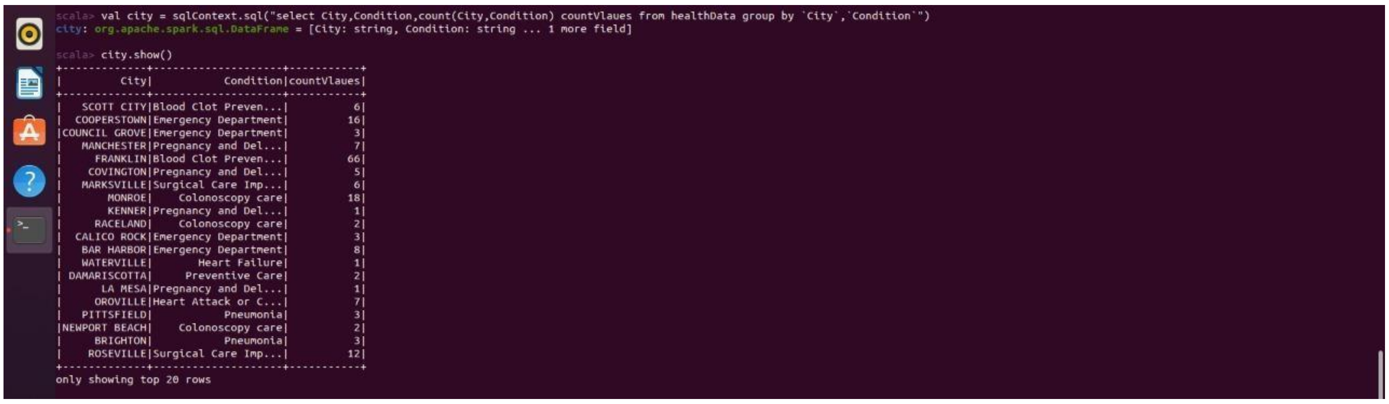
```
scala> HA.show()
+----------+--------------------+--------------------+----------+-----+-------+------------+-----------+--------------------+---------+--------------------+-------------+-------------+------------------
--+----------------+--------------+
|ProviderID|        HospitalName|             Address|      City|State|ZIPCode|  CountyName|PhoneNumber|           Condition|MeasureID|         MeasureName|        Score|       Sample|            Footno
te|MeasureStartDate|MeasureEndDate|
+----------+--------------------+--------------------+----------+-----+-------+------------+-----------+--------------------+---------+--------------------+-------------+-------------+------------------
--+----------------+--------------+
|  171354.0|COMMUNITY HOSPITA...| 120 WEST 8TH STREET|     ONAGA|   KS|  66521|POTTAWATOMIE| 7858894272|Heart Attack or C...|     OP_2|Fibrinolytic Ther...|Not Available|Not Available|3 - Results are b.
..|       10/1/2014|     9/30/2015|
|  171354.0|COMMUNITY HOSPITA...| 120 WEST 8TH STREET|     ONAGA|   KS|  66521|POTTAWATOMIE| 7858894272|Heart Attack or C...|    OP_3b|Median Time to Tr...|Not Available|Not Available|1 - The number of.
..|       10/1/2014|     9/30/2015|
|  171354.0|COMMUNITY HOSPITA...| 120 WEST 8TH STREET|     ONAGA|   KS|  66521|POTTAWATOMIE| 7858894272|Heart Attack or C...|     OP_4|  Aspirin at Arrival|          100|           14|                nu
ll|       10/1/2014|     9/30/2015|
|  171354.0|COMMUNITY HOSPITA...| 120 WEST 8TH STREET|     ONAGA|   KS|  66521|POTTAWATOMIE| 7858894272|Heart Attack or C...|     OP_5|  Median Time to ECG|            7|           14|                nu
ll|       10/1/2014|     9/30/2015|
|  351304.0|COMMUNITY MEMORIA...|       220 5TH AVE W|TURTLE LAKE|  ND|  58575|      MCLEAN| 7014482331|Heart Attack or C...|   AMI_7a|Fibrinolytic Ther...|Not Available|Not Available|5 - Results are n.
..|       10/1/2014|     9/30/2015|
|  171369.0|HODGEMAN COUNTY H...|         809 BRAMLEY|   JETMORE|   KS|  67854|    HODGEMAN| 6203578361|Heart Attack or C...|   AMI_8a|Primary PCI Recei...|Not Available|Not Available|3 - Results are b.
..|       10/1/2014|     9/30/2015|
|   10005.0|MARSHALL MEDICAL ...|2505 U S HIGHWAY ...|      BOAZ|   AL|  35957|    MARSHALL| 2565938310|Heart Attack or C...|   AMI_7a|Fibrinolytic Ther...|Not Available|Not Available|7 - No cases met .
..|       10/1/2014|     9/30/2015|
|   10005.0|MARSHALL MEDICAL ...|2505 U S HIGHWAY ...|      BOAZ|   AL|  35957|    MARSHALL| 2565938310|Heart Attack or C...|   AMI_8a|Primary PCI Recei...|           85|           13|                nu
ll|       10/1/2014|     9/30/2015|
|   10005.0|MARSHALL MEDICAL ...|2505 U S HIGHWAY ...|      BOAZ|   AL|  35957|    MARSHALL| 2565938310|Heart Attack or C...|     OP_1|Median Time to Fi...|Not Available|Not Available|1 - The number of.
..|       10/1/2014|     9/30/2015|
|   10005.0|MARSHALL MEDICAL ...|2505 U S HIGHWAY ...|      BOAZ|   AL|  35957|    MARSHALL| 2565938310|Heart Attack or C...|     OP_2|Fibrinolytic Ther...|Not Available|Not Available|1 - The number of.
..|       10/1/2014|     9/30/2015|
|   10005.0|MARSHALL MEDICAL ...|2505 U S HIGHWAY ...|      BOAZ|   AL|  35957|    MARSHALL| 2565938310|Heart Attack or C...|    OP_3b|Median Time to Tr...|Not Available|Not Available|1 - The number of.
..|       10/1/2014|     9/30/2015|
|   10005.0|MARSHALL MEDICAL ...|2505 U S HIGHWAY ...|      BOAZ|   AL|  35957|    MARSHALL| 2565938310|Heart Attack or C...|     OP_4|  Aspirin at Arrival|           98|          191|                nu
ll|       10/1/2014|     9/30/2015|
|   10005.0|MARSHALL MEDICAL ...|2505 U S HIGHWAY ...|      BOAZ|   AL|  35957|    MARSHALL| 2565938310|Heart Attack or C...|     OP_5|  Median Time to ECG|           13|          199|                nu
ll|       10/1/2014|     9/30/2015|
|  171369.0|HODGEMAN COUNTY H...|         809 BRAMLEY|   JETMORE|   KS|  67854|    HODGEMAN| 6203578361|Heart Attack or C...|   AMI_7a|Fibrinolytic Ther...|Not Available|Not Available|3 - Results are b.
..|       10/1/2014|     9/30/2015|
|  171354.0|COMMUNITY HOSPITA...| 120 WEST 8TH STREET|     ONAGA|   KS|  66521|POTTAWATOMIE| 7858894272|Heart Attack or C...|   AMI_8a|Primary PCI Recei...|Not Available|Not Available|3 - Results are b.
..|       10/1/2014|     9/30/2015|
|  171354.0|COMMUNITY HOSPITA...| 120 WEST 8TH STREET|     ONAGA|   KS|  66521|POTTAWATOMIE| 7858894272|Heart Attack or C...|   AMI_7a|Fibrinolytic Ther...|Not Available|Not Available|3 - Results are b.
..|       10/1/2014|     9/30/2015|
|  171354.0|COMMUNITY HOSPITA...| 120 WEST 8TH STREET|     ONAGA|   KS|  66521|POTTAWATOMIE| 7858894272|Heart Attack or C...|     OP_1|Median Time to Fi...|Not Available|Not Available|3 - Results are b.
..|       10/1/2014|     9/30/2015|
|  171355.0|TREGO COUNTY LEMK...|  320 THIRTEENTH ST| WA KEENEY|   KS|  67672|       TREGO| 7857432182|Heart Attack or C...|   AMI_7a|Fibrinolytic Ther...|Not Available|Not Available|5 - Results are n.
..|       10/1/2014|     9/30/2015|
```

- As we can see the condition here is Heart Attck Or Chest Pain and its executed using show() command.



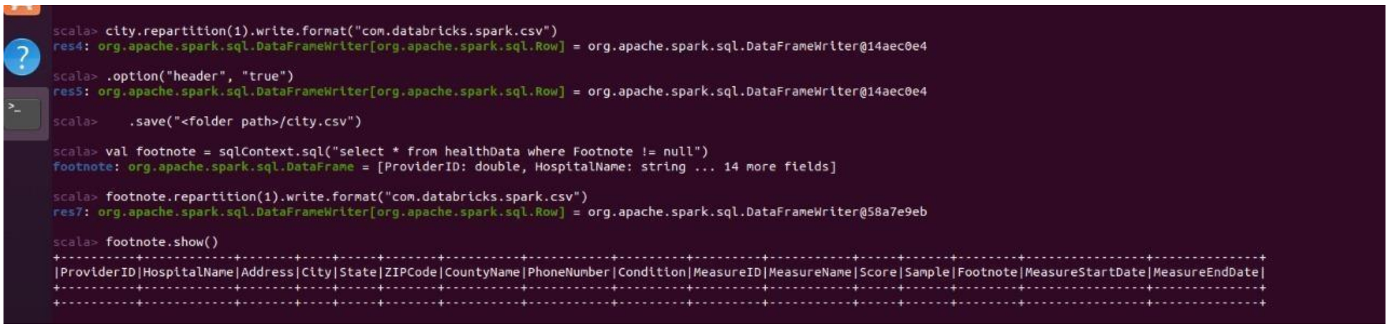
```
|  171355.0|TREGO COUNTY LEMK...|  320 THIRTEENTH ST| WA KEENEY|   KS|  67672|       TREGO| 7857432182|Heart Attack or C...|   AMI_7a|Fibrinolytic Ther...|Not Available|Not Available|5 - Results are n.
..|       10/1/2014|     9/30/2015|
|  171356.0|ST LUKE HOSPITAL ...|  535 SOUTH FREEBRON|    MARION|   KS|  66861|      MARION| 6203822177|Heart Attack or C...|   AMI_8a|Primary PCI Recei...|Not Available|Not Available|2 - Data submitte.
..|       10/1/2014|     9/30/2015|
|  171357.0|HILLSBORO COMMUNI...|   701 S MAIN STREET| HILLSBORO|   KS|  67063|      MARION| 6209473114|Heart Attack or C...|     OP_4|  Aspirin at Arrival|Not Available|Not Available|5 - Results are n.
..|       10/1/2014|     9/30/2015|
+----------+--------------------+--------------------+----------+-----+-------+------------+-----------+--------------------+---------+--------------------+-------------+-------------+------------------
--+----------------+--------------+
only showing top 20 rows


scala> val diseasesReported = sqlContext.sql("select Condition,count(Condition) countVlaues from healthData group by `Condition`")
diseasesReported: org.apache.spark.sql.DataFrame = [Condition: string, countVlaues: bigint]

scala> diseasesReported.show()
+--------------------+-----------+
| Ubuntu Software dition|countVlaues|
+--------------------+-----------+
|Heart Attack or C...|      30104|
|Pregnancy and Del...|       4628|
|          Stroke Care|      37024|
|Blood Clot Preven...|      27768|
|        Heart Failure|       4757|
|     Colonoscopy care|       8156|
|Emergency Department|      34451|
|Surgical Care Imp...|      28542|
|    Children's Asthma|        117|
|      Preventive Care|       8409|
|            Pneumonia|       4757|
+--------------------+-----------+
```

- Now we have created variable diseasesReported and we have counted the number of patients for each disease condition.



```
scala> val city = sqlContext.sql("select City,Condition,count(City,Condition) countVlaues from healthData group by `City`,`Condition`")
city: org.apache.spark.sql.DataFrame = [City: string, Condition: string ... 1 more field]

scala> city.show()
+-------------+--------------------+-----------+
|         City|           Condition|countVlaues|
+-------------+--------------------+-----------+
|   SCOTT CITY|Blood Clot Preven...|          6|
|  COOPERSTOWN|Emergency Department|         16|
|COUNCIL GROVE|Emergency Department|          3|
|   MANCHESTER|Pregnancy and Del...|          7|
|     FRANKLIN|Blood Clot Preven...|         66|
|    COVINGTON|Pregnancy and Del...|          5|
|   MARKSVILLE|Surgical Care Imp...|          6|
|       MONROE|    Colonoscopy care|         18|
|       KENNER|Pregnancy and Del...|          1|
|     RACELAND|    Colonoscopy care|          2|
|  CALICO ROCK|Emergency Department|          3|
|   BAR HARBOR|Emergency Department|          8|
|   WATERVILLE|       Heart Failure|          1|
|  DAMARISCOTTA|     Preventive Care|          2|
|      LA MESA|Pregnancy and Del...|          1|
|     OROVILLE|Heart Attack or C...|          7|
|   PITTSFIELD|           Pneumonia|          3|
|NEWPORT BEACH|    Colonoscopy care|          2|
|     BRIGHTON|           Pneumonia|          3|
|    ROSEVILLE|Surgical Care Imp...|         12|
+-------------+--------------------+-----------+
only showing top 20 rows
```

- Now we have created a variable named city and we have segregated city with the condition and the number of patients with that condition.



```
scala> city.repartition(1).write.format("com.databricks.spark.csv")
res4: org.apache.spark.sql.DataFrameWriter[org.apache.spark.sql.Row] = org.apache.spark.sql.DataFrameWriter@14aec0e4

scala> .option("header", "true")
res5: org.apache.spark.sql.DataFrameWriter[org.apache.spark.sql.Row] = org.apache.spark.sql.DataFrameWriter@14aec0e4

scala>    .save("<folder path>/city.csv")

scala> val footnote = sqlContext.sql("select * from healthData where Footnote != null")
footnote: org.apache.spark.sql.DataFrame = [ProviderID: double, HospitalName: string ... 14 more fields]

scala> footnote.repartition(1).write.format("com.databricks.spark.csv")
res7: org.apache.spark.sql.DataFrameWriter[org.apache.spark.sql.Row] = org.apache.spark.sql.DataFrameWriter@58a7e9eb

scala> footnote.show()
+----------+------------+-------+----+-----+-------+----------+-----------+---------+---------+-----------+-----+------+--------+----------------+--------------+
|ProviderID|HospitalName|Address|City|State|ZIPCode|CountyName|PhoneNumber|Condition|MeasureID|MeasureName|Score|Sample|Footnote|MeasureStartDate|MeasureEndDate|
+----------+------------+-------+----+-----+-------+----------+-----------+---------+---------+-----------+-----+------+--------+----------------+--------------+
+----------+------------+-------+----+-----+-------+----------+-----------+---------+---------+-----------+-----+------+--------+----------------+--------------+
```

- Now we have repartitioned the city variable using the repartition command.Repartition method can be used to either increase or decrease the number of partitions ina DataFrame. The repartition algorithm does a full data shuffle and equally distributes the data among the partitions.It does not attempt to minimize data movement like the coalesce algorithm.

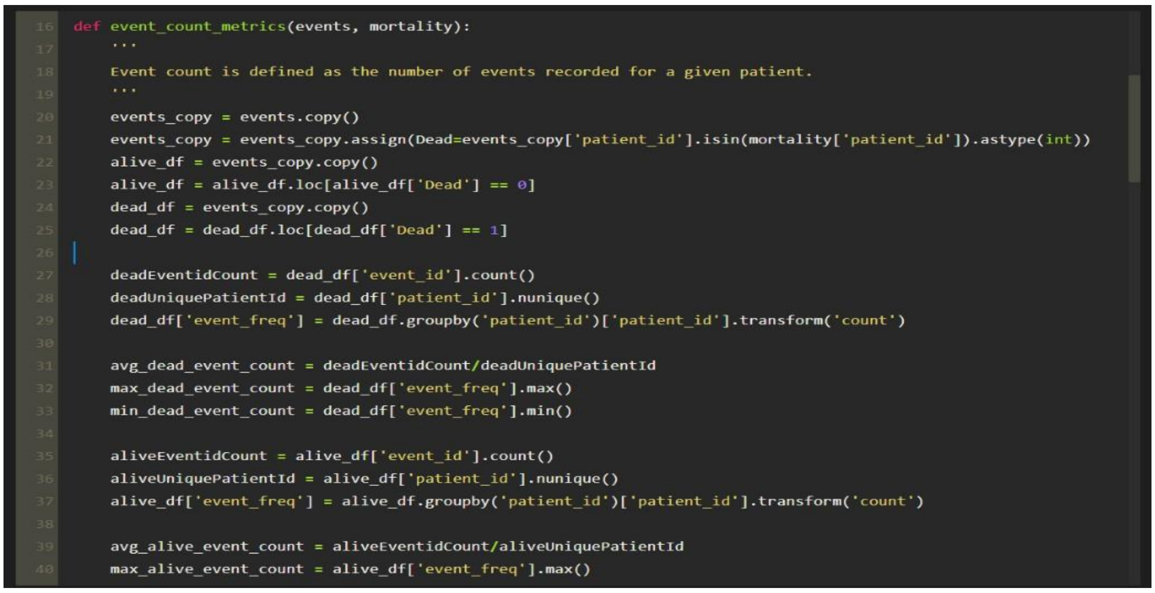
- Then we have created a variable named footnote and we have stored all the footnote values which has null value its basically like preprocessing and later we have removed all the values from the dataset with the value null.

- We can see that when we use show() command on footnote all the values are removed and the table and the table empty.

<u>Mortality analysis (Model Building):</u>

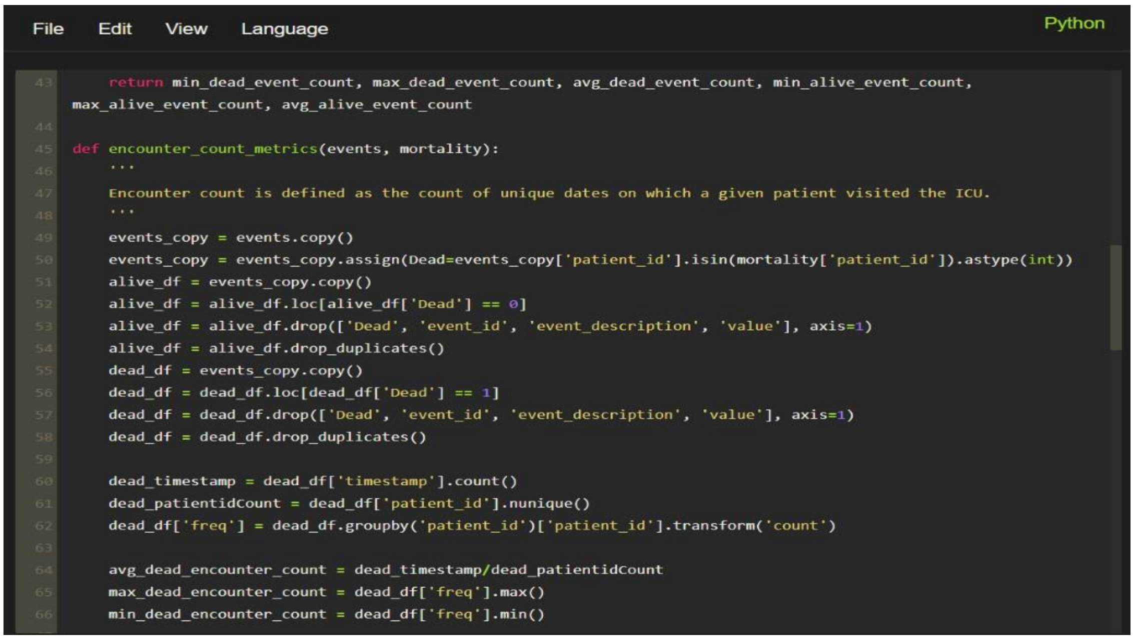
We Used Python in predicting patient mortality analysis. Did preliminary descriptive statistics analysis, constructed features selection process, created machine learning models using logistic regression, support vector machine learning, and decision tree. Analyzed performance of the models, using accuracy, area under the curve (AUC), precision, recall, F-score, K-fold and randomized K-fold validation.

Event statistics:



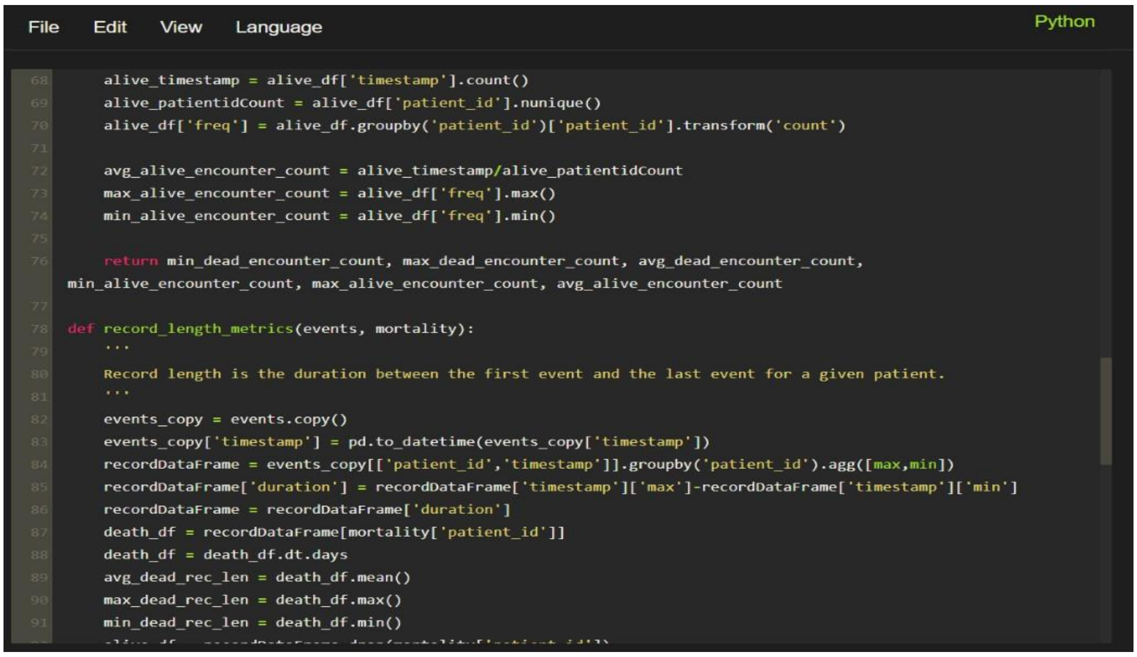
```python
def event_count_metrics(events, mortality):
    '''
    Event count is defined as the number of events recorded for a given patient.
    '''
    events_copy = events.copy()
    events_copy = events_copy.assign(Dead=events_copy['patient_id'].isin(mortality['patient_id']).astype(int))
    alive_df = events_copy.copy()
    alive_df = alive_df.loc[alive_df['Dead'] == 0]
    dead_df = events_copy.copy()
    dead_df = dead_df.loc[dead_df['Dead'] == 1]

    deadEventidCount = dead_df['event_id'].count()
    deadUniquePatientId = dead_df['patient_id'].nunique()
    dead_df['event_freq'] = dead_df.groupby('patient_id')['patient_id'].transform('count')

    avg_dead_event_count = deadEventidCount/deadUniquePatientId
    max_dead_event_count = dead_df['event_freq'].max()
    min_dead_event_count = dead_df['event_freq'].min()

    aliveEventidCount = alive_df['event_id'].count()
    aliveUniquePatientId = alive_df['patient_id'].nunique()
    alive_df['event_freq'] = alive_df.groupby('patient_id')['patient_id'].transform('count')

    avg_alive_event_count = aliveEventidCount/aliveUniquePatientId
    max_alive_event_count = alive_df['event_freq'].max()
```

Here we have written the code for event_count metrics which gives the output of average event count,max_count and min_count for the deceased patients and the alive patients.
We have choosen patients with unique patient id.



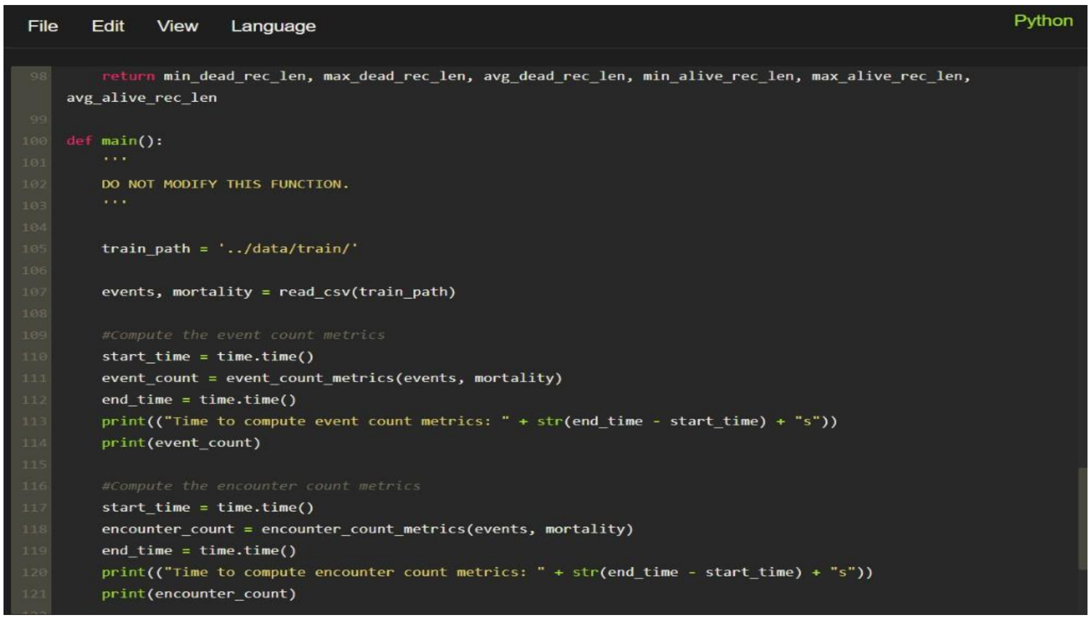
```python
    return min_dead_event_count, max_dead_event_count, avg_dead_event_count, min_alive_event_count,
max_alive_event_count, avg_alive_event_count

def encounter_count_metrics(events, mortality):
    '''
    Encounter count is defined as the count of unique dates on which a given patient visited the ICU.
    '''
    events_copy = events.copy()
    events_copy = events_copy.assign(Dead=events_copy['patient_id'].isin(mortality['patient_id']).astype(int))
    alive_df = events_copy.copy()
    alive_df = alive_df.loc[alive_df['Dead'] == 0]
    alive_df = alive_df.drop(['Dead', 'event_id', 'event_description', 'value'], axis=1)
    alive_df = alive_df.drop_duplicates()
    dead_df = events_copy.copy()
    dead_df = dead_df.loc[dead_df['Dead'] == 1]
    dead_df = dead_df.drop(['Dead', 'event_id', 'event_description', 'value'], axis=1)
    dead_df = dead_df.drop_duplicates()

    dead_timestamp = dead_df['timestamp'].count()
    dead_patientidCount = dead_df['patient_id'].nunique()
    dead_df['freq'] = dead_df.groupby('patient_id')['patient_id'].transform('count')

    avg_dead_encounter_count = dead_timestamp/dead_patientidCount
    max_dead_encounter_count = dead_df['freq'].max()
    min_dead_encounter_count = dead_df['freq'].min()
```

Here we have written the code for encounter_count_metrics which gives the output of average encounter count,max_count and min_count for the deceased patients and the alive patients.
We have choosen patients with unique patient id.We have also dropped some of the columns named Dead, event_id, event_description, value.

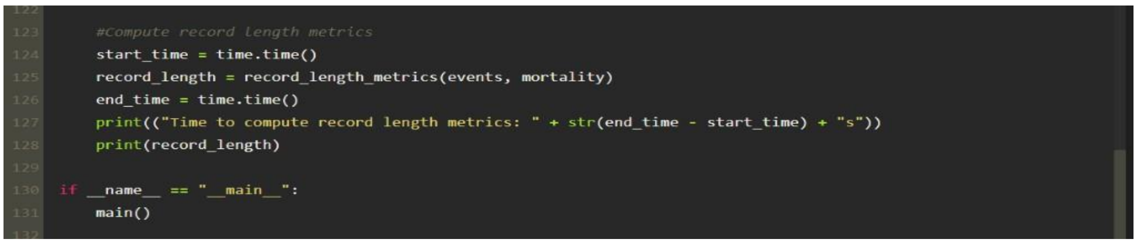


```python
    alive_timestamp = alive_df['timestamp'].count()
    alive_patientidCount = alive_df['patient_id'].nunique()
    alive_df['freq'] = alive_df.groupby('patient_id')['patient_id'].transform('count')

    avg_alive_encounter_count = alive_timestamp/alive_patientidCount
    max_alive_encounter_count = alive_df['freq'].max()
    min_alive_encounter_count = alive_df['freq'].min()

    return min_dead_encounter_count, max_dead_encounter_count, avg_dead_encounter_count,
min_alive_encounter_count, max_alive_encounter_count, avg_alive_encounter_count

def record_length_metrics(events, mortality):
    '''
    Record length is the duration between the first event and the last event for a given patient.
    '''
    events_copy = events.copy()
    events_copy['timestamp'] = pd.to_datetime(events_copy['timestamp'])
    recordDataFrame = events_copy[['patient_id','timestamp']].groupby('patient_id').agg([max,min])
    recordDataFrame['duration'] = recordDataFrame['timestamp']['max']-recordDataFrame['timestamp']['min']
    recordDataFrame = recordDataFrame['duration']
    death_df = recordDataFrame[mortality['patient_id']]
    death_df = death_df.dt.days
    avg_dead_rec_len = death_df.mean()
    max_dead_rec_len = death_df.max()
    min_dead_rec_len = death_df.min()
```

Here we have written the code for record_length_metrics which gives the output of average record length,max_length and min_length for the deceased patients and the alive patients.
We have choosen patients with unique patient id. Length was calculated with parameters events and mortality.



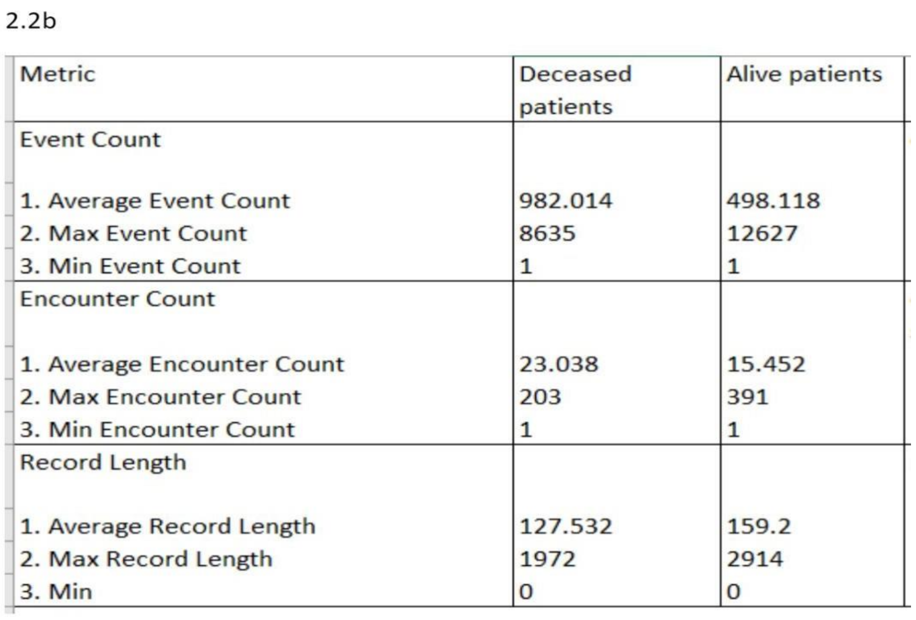
```python
    return min_dead_rec_len, max_dead_rec_len, avg_dead_rec_len, min_alive_rec_len, max_alive_rec_len,
avg_alive_rec_len

def main():
    '''
    DO NOT MODIFY THIS FUNCTION.
    '''

    train_path = '../data/train/'

    events, mortality = read_csv(train_path)

    #Compute the event count metrics
    start_time = time.time()
    event_count = event_count_metrics(events, mortality)
    end_time = time.time()
    print(("Time to compute event count metrics: " + str(end_time - start_time) + "s"))
    print(event_count)

    #Compute the encounter count metrics
    start_time = time.time()
    encounter_count = encounter_count_metrics(events, mortality)
    end_time = time.time()
    print(("Time to compute encounter count metrics: " + str(end_time - start_time) + "s"))
    print(encounter_count)
```



```python

    #Compute record length metrics
    start_time = time.time()
    record_length = record_length_metrics(events, mortality)
    end_time = time.time()
    print(("Time to compute record length metrics: " + str(end_time - start_time) + "s"))
    print(record_length)

if __name__ == "__main__":
    main()
```

The whole code is uploaded on github: https://github.com/sumukhrr/Big-Data-Project-on-Health-Care./blob/main/event_statistics.py

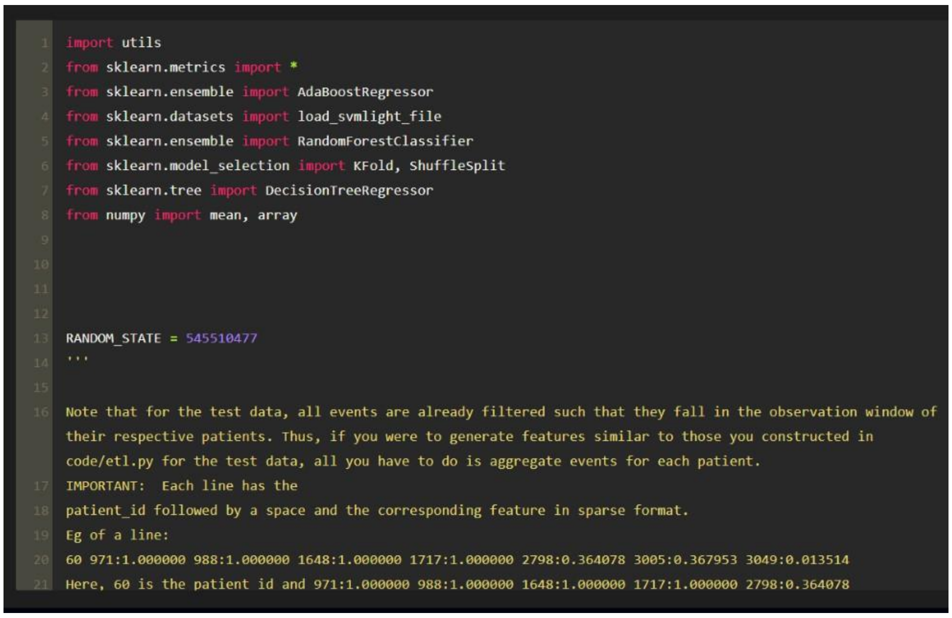
Output:



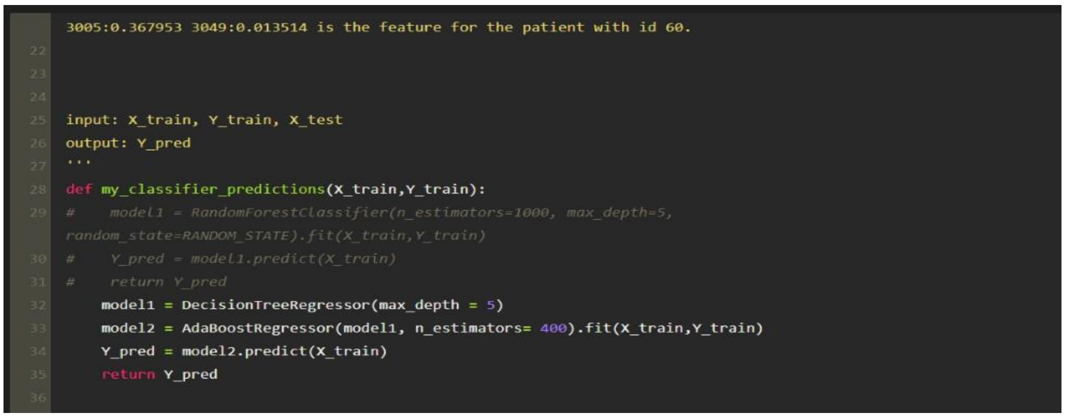
2.2b

Metric	Deceased patients	Alive patients
Event Count 1. Average Event Count 2. Max Event Count 3. Min Event Count	 982.014 8635 1	 498.118 12627 1
Encounter Count 1. Average Encounter Count 2. Max Encounter Count 3. Min Encounter Count	 23.038 203 1	 15.452 391 1
Record Length 1. Average Record Length 2. Max Record Length 3. Min	 127.532 1972 0	 159.2 2914 0

Model Training:

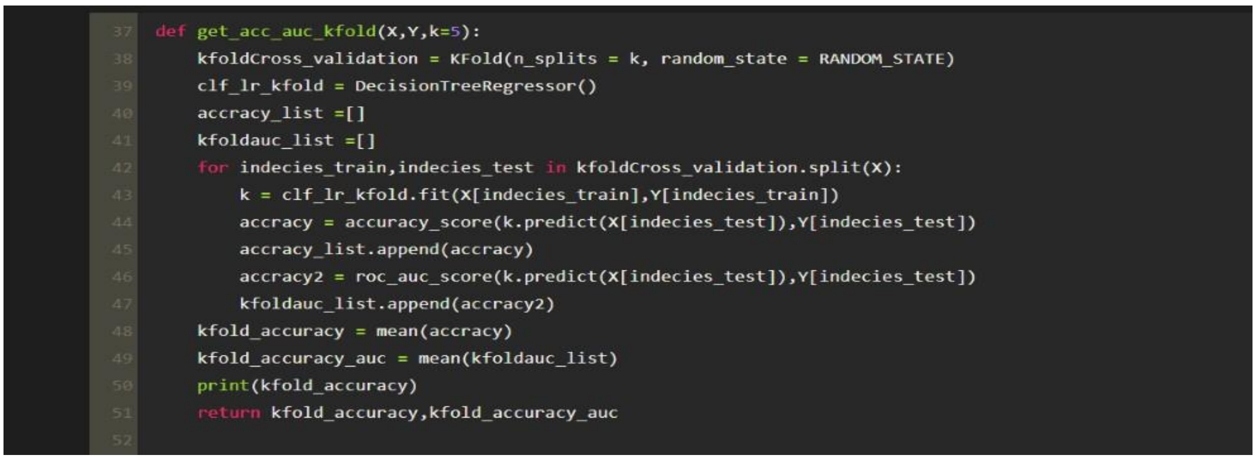


```
import utils
from sklearn.metrics import *
from sklearn.ensemble import AdaBoostRegressor
from sklearn.datasets import load_svmlight_file
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold, ShuffleSplit
from sklearn.tree import DecisionTreeRegressor
from numpy import mean, array




RANDOM_STATE = 545510477
'''

Note that for the test data, all events are already filtered such that they fall in the observation window of
their respective patients. Thus, if you were to generate features similar to those you constructed in
code/etl.py for the test data, all you have to do is aggregate events for each patient.
IMPORTANT:  Each line has the
patient_id followed by a space and the corresponding feature in sparse format.
Eg of a line:
60 971:1.000000 988:1.000000 1648:1.000000 1717:1.000000 2798:0.364078 3005:0.367953 3049:0.013514
Here, 60 is the patient id and 971:1.000000 988:1.000000 1648:1.000000 1717:1.000000 2798:0.364078
```



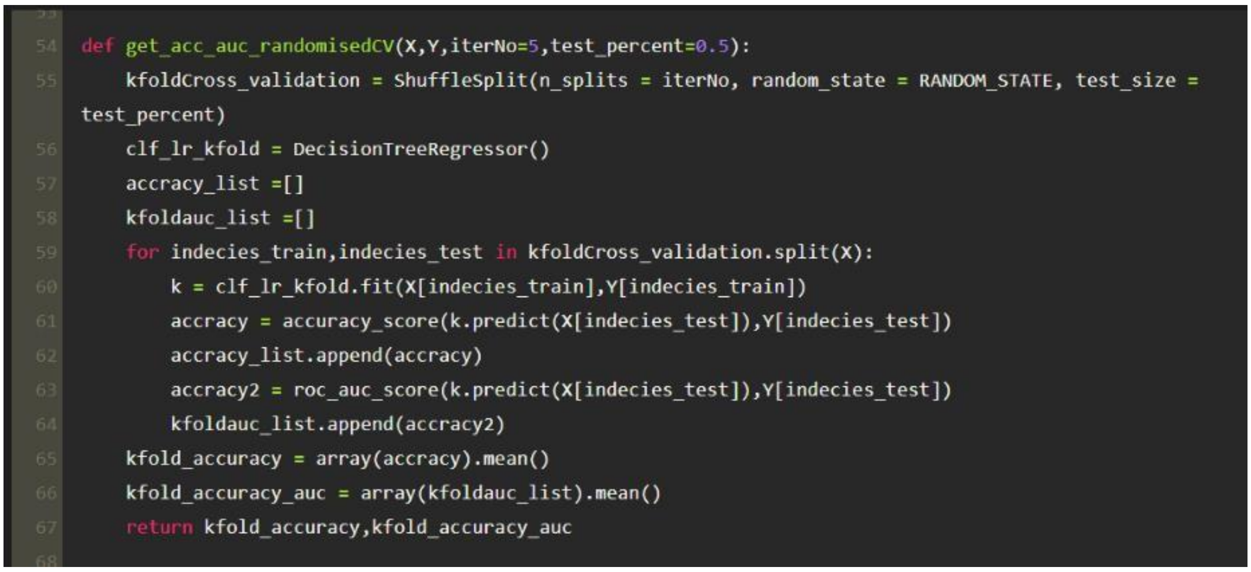
```
3005:0.367953 3049:0.013514 is the feature for the patient with id 60.



input: X_train, Y_train, X_test
output: Y_pred
'''
def my_classifier_predictions(X_train,Y_train):
#    model1 = RandomForestClassifier(n_estimators=1000, max_depth=5,
random_state=RANDOM_STATE).fit(X_train,Y_train)
#    Y_pred = model1.predict(X_train)
#    return Y_pred
    model1 = DecisionTreeRegressor(max_depth = 5)
    model2 = AdaBoostRegressor(model1, n_estimators= 400).fit(X_train,Y_train)
    Y_pred = model2.predict(X_train)
    return Y_pred
```

In my_classifier_prediction we have used decision tree regressor with max_depth of 5 and the number of estimators used was 400 and this was conducted on training dataset.



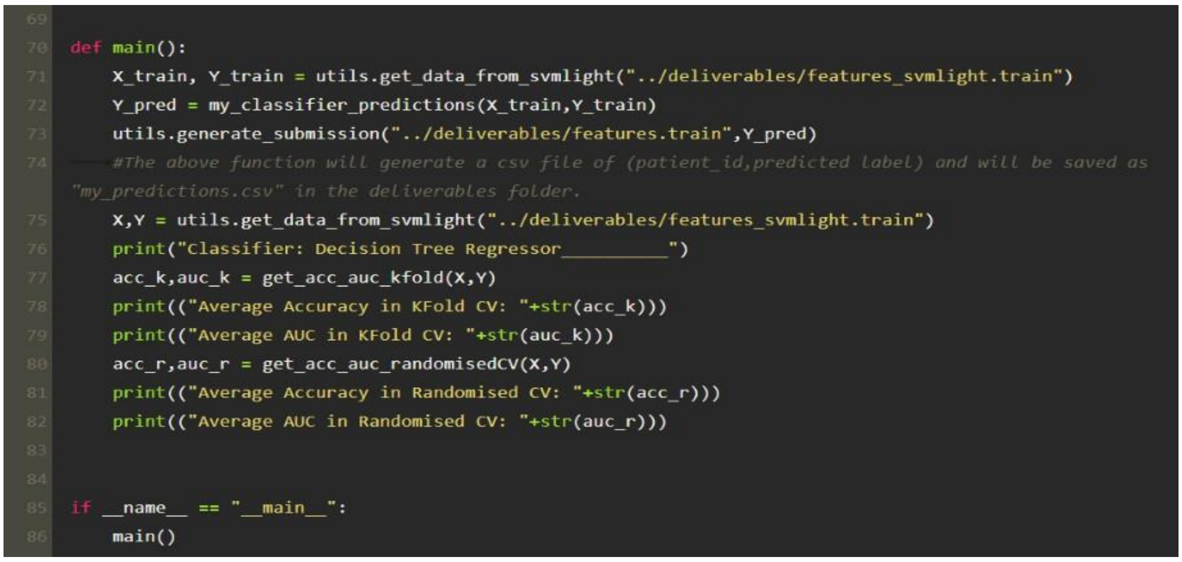
```
def get_acc_auc_kfold(X,Y,k=5):
    kfoldCross_validation = KFold(n_splits = k, random_state = RANDOM_STATE)
    clf_lr_kfold = DecisionTreeRegressor()
    accracy_list =[]
    kfoldauc_list =[]
    for indecies_train,indecies_test in kfoldCross_validation.split(X):
        k = clf_lr_kfold.fit(X[indecies_train],Y[indecies_train])
        accracy = accuracy_score(k.predict(X[indecies_test]),Y[indecies_test])
        accracy_list.append(accracy)
        accracy2 = roc_auc_score(k.predict(X[indecies_test]),Y[indecies_test])
        kfoldauc_list.append(accracy2)
    kfold_accuracy = mean(accracy)
    kfold_accuracy_auc = mean(kfoldauc_list)
    print(kfold_accuracy)
    return kfold_accuracy,kfold_accuracy_auc
```

Using kfold cross validation with k=5 number of split.and we have also calculated kfold for AUC (area under curve) and calculated accuracy for both the things.



```python
def get_acc_auc_randomisedCV(X,Y,iterNo=5,test_percent=0.5):
    kfoldCross_validation = ShuffleSplit(n_splits = iterNo, random_state = RANDOM_STATE, test_size = test_percent)
    clf_lr_kfold = DecisionTreeRegressor()
    accracy_list =[]
    kfoldauc_list =[]
    for indecies_train,indecies_test in kfoldCross_validation.split(X):
        k = clf_lr_kfold.fit(X[indecies_train],Y[indecies_train])
        accracy = accuracy_score(k.predict(X[indecies_test]),Y[indecies_test])
        accracy_list.append(accracy)
        accracy2 = roc_auc_score(k.predict(X[indecies_test]),Y[indecies_test])
        kfoldauc_list.append(accracy2)
    kfold_accuracy = array(accracy).mean()
    kfold_accuracy_auc = array(kfoldauc_list).mean()
    return kfold_accuracy,kfold_accuracy_auc
```

Now when we take test dataset to tain we have taken 5 iterations and calculated kfold cross validation of area under curve.We have calculated accuracy for both.

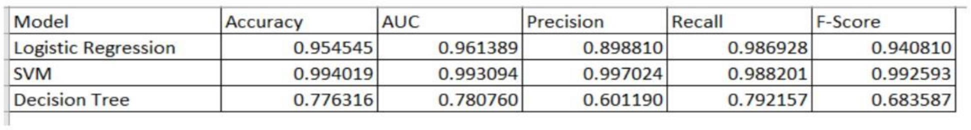


```python
def main():
    X_train, Y_train = utils.get_data_from_svmlight("../deliverables/features_svmlight.train")
    Y_pred = my_classifier_predictions(X_train,Y_train)
    utils.generate_submission("../deliverables/features.train",Y_pred)
    #The above function will generate a csv file of (patient_id,predicted label) and will be saved as "my_predictions.csv" in the deliverables folder.
    X,Y = utils.get_data_from_svmlight("../deliverables/features_svmlight.train")
    print("Classifier: Decision Tree Regressor__________")
    acc_k,auc_k = get_acc_auc_kfold(X,Y)
    print(("Average Accuracy in KFold CV: "+str(acc_k)))
    print(("Average AUC in KFold CV: "+str(auc_k)))
    acc_r,auc_r = get_acc_auc_randomisedCV(X,Y)
    print(("Average Accuracy in Randomised CV: "+str(acc_r)))
    print(("Average AUC in Randomised CV: "+str(auc_r)))


if __name__ == "__main__":
    main()
```

The path given here is deliverable folder change it to your path where you want it to be stored. We have calculated normal accuracy and Area under curve Accuracy in randomised CV.

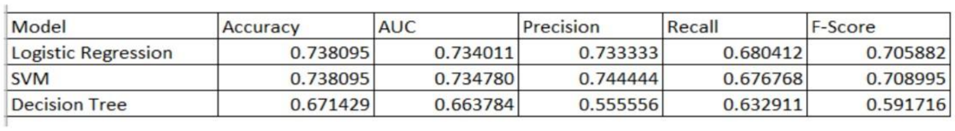
Whole code is uploaded on : https://github.com/sumukhrr/Big-Data-Project-on-Health-Care./blob/main/event_statistics.py.
Output:

4.1b Training Data



Model	Accuracy	AUC	Precision	Recall	F-Score
Logistic Regression	0.954545	0.961389	0.898810	0.986928	0.940810
SVM	0.994019	0.993094	0.997024	0.988201	0.992593
Decision Tree	0.776316	0.780760	0.601190	0.792157	0.683587

4.1c Test Data



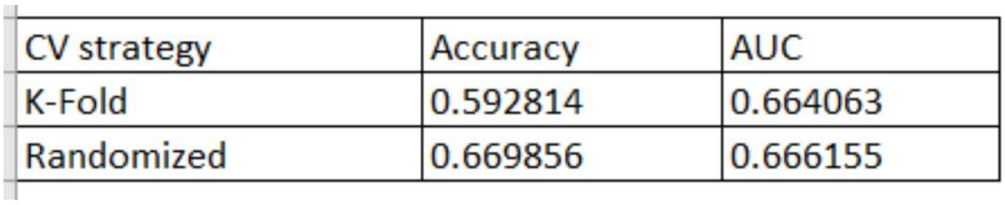
Model	Accuracy	AUC	Precision	Recall	F-Score
Logistic Regression	0.738095	0.734011	0.733333	0.680412	0.705882
SVM	0.738095	0.734780	0.744444	0.676768	0.708995
Decision Tree	0.671429	0.663784	0.555556	0.632911	0.591716

The test dataset is 50 percent of the whole dataset and the rest was training dataset.We can see that Training dataset gives more accuracy that test data which totally depends on the random elements which is chosen for test dataset.SVM has the highest accuracy of 99.40%.

We have also tested our model with random forest classifier and decision tree classifier also .with the same code but with the different parameters which resulted in somewhat better accuracy than before.
Code is uploaded on: https://github.com/sumukhrr/Big-Data-Project-on-Health-Care./blob/main/event_statistics.py.

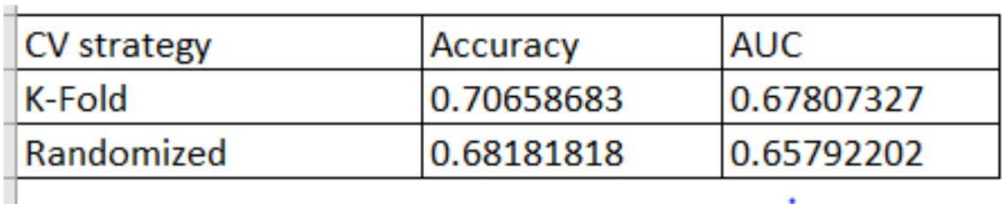
Outputs are:

Results of Random Forest Classifier



CV strategy	Accuracy	AUC
K-Fold	0.592814	0.664063
Randomized	0.669856	0.666155

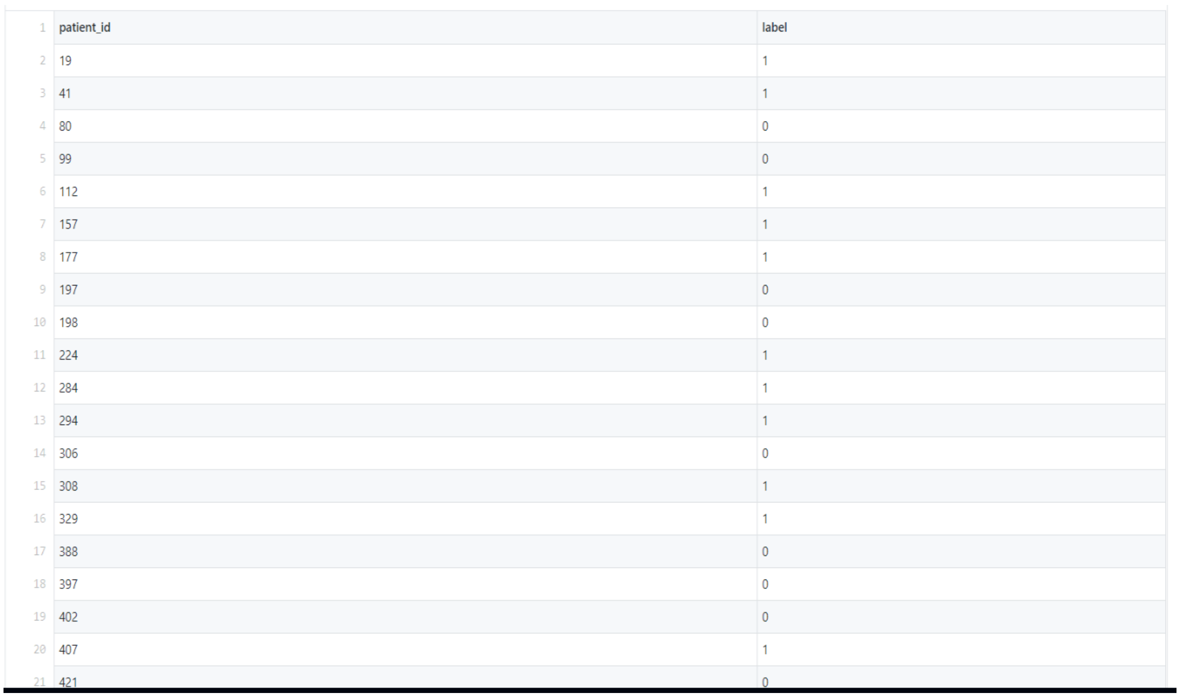
Results of Decision Tree Regressor



CV strategy	Accuracy	AUC
K-Fold	0.70658683	0.67807327
Randomized	0.68181818	0.65792202

As we can see that decision tree regressor gives more accuracy than Random forest classifier in this case and the accuracy is 70.65%.

Look at Predictions.csv:



patient_id	label
19	1
41	1
80	0
99	0
112	1
157	1
177	1
197	0
198	0
224	1
284	1
294	1
306	0
308	1
329	1
388	0
397	0
402	0
407	1
421	0

Here 1 represents the patient is alive and 0 tells that the patient is dead.

IMPORTANT: All codes of sparkSQL and csv files of model building are uploaded on github https://github.com/sumukhrr/Big-Data-Project-on-Health-Care./blob/main/event_statistics.py.

Further Work

1) We can improve our models performance by following means
    - Outlier detection and removing.
    - Collecting and adding more data from hospitals.
    - Using parameter tuning to find optimal value for each parameter which will improve model accuracy.
    - Use principle component analysis for eliminating correlated features.
    - Using other features of spark ,hive and kafka .
    - Building more features and analysis on the dataset.

CONCLUSION

In healthcare, large amounts of heterogeneous medical data have become available in various healthcare organizations (payers, providers, pharmaceuticals). This data could be an enabling resource for deriving insights for improving care delivery and reducing waste. The enormity and complexity of these data-sets present great challenges in analyses and subsequent applications to a practical clinical environment.

In this Project we were introduced to various technologies which can be used to easily analyse the data and come to a conclusion.

We have used sparkSQL to make changes to dataset and to find various insights on health.csv dataset.

We have used python for model building and various ml models and we can choose the best model for mortality rate analysis.

These tools are very useful for any big data analysis as these help providing great insights on dataset which will be helpful for analysis.

REFERENCES:

- https://www.researchgate.net/publication/334690194_Using_Data_Analytics_to_Predict_Hospital_Mortality_in_Sepsis_Patients
- https://arxiv.org/ftp/arxiv/papers/1705/1705.03508.pdf
- https://journals.sagepub.com/doi/full/10.1177/1550147720928897
- https://crvsgateway.info/Tool-for-the-analysis-of-statistics-on-mortality-and-causes-of-death-ANACONDA~360
- https://bmjopen.bmj.com/content/10/8/e034524
- https://www.nature.com/articles/s41598-020-75767-2
- https://www.thelancet.com/journals/landig/article/PIIS2589-7500(20)30217-X/fulltext