Apache Hive-Based Big Data Analysis of HealthCare

Abstract

The Electronic Health Record (EHR) stores valuable information on patient records in digital form. The amount of data in the EHR is increasing due to government mandates and technological innovation. Patient data are recorded using sensors and medical reports. Given huge amounts of heterogeneous data in the EHR, there is a need for effective methods to store and analyze these data for meaningful interpretations. This study focuses on various analysis techniques for analyzing and retrieving required information from big data in the EHR. Many Hive queries are conducted in the Hadoop distributed file system to extract valuable information. The study also proposes and demonstrates the use of Tableau as a data analysis technique for effectively deducing valuable information in the form of visual graphs.

1. Introduction

A digital version of patient medical reports known as the Electronic Health Record (EHR) provides benefits such as data sharing, easy access to patient health history, and powerful analysis techniques for extracting valuable information from big data in the EHR. Given such benefits, the U.S. government made it mandatory to maintain the EHR for all patients under the American Recovery and Reinvestment Act (ARRA) of 2009. As a result, electronic data in the EHR has become heterogenous and is rapidly growing. Therefore, it is difficult for a single system to support huge amounts of big data. Unstructured traditional DBMS tools have been shown to be insufficient in handling such large data. All these requirements have led to the use of distributed systems for the efficient processing of big data. The Hadoop Distributed File System (HDFS) is a technique in which big data can be processed in a distributed manner using a number of systems.

However, revealing hidden patterns and unseen relationships among vast health record data in terms of their volume and variety remains a key challenge for data analytics. Big data improve on legacy warehousing by providing massive volume, velocity, and variety in corporate data. Pattern extraction from various databases entails many data optimization issues, so velocity and variety represent another key issue in big data. Moreover, big data can also help manage scaling through technology migration during data expansion. In general, big data encompass scientific mining and business intelligence models for enhanced insights, process automation, and decision making for data-oriented organizations [1].

This paper presents different SQL-like statements for querying and managing useful information using Apache Hive on top of the HDFS. The paper also proposes the integration of Apache Hive with Tableau as a powerful visualization tool to enable deeper insights into big data from the EHR.

The rest of this paper is organized as follows: Section 2 describes important big data tools and techniques for analyzing and extracting useful information and offers details on Apache Hadoop, its architecture, and the role of Apache HIVE in big data analytics. Section 3 presents related work on the analysis of big data and their issues. Section 4 describes the proposed work by explaining the dataset and experimental methodology. Section 5 presents the output of various Apache Hive queries and discusses different metrics for measuring the error. It also details the integration of Tableau for better visualization of Apache Hive query results. The section also graphically presents forecasted states of high alert for high rate deaths. Finally, Section 6 concludes with some avenues for future research.

2. Big Data Tools and Techniques

Big data sources consist of scientific repositories such as the EHR, forecast analysis, and social media repositories such as Facebook and Twitter for various textual, voice, video, and image patterns [2]. Big data analytics involves scientific mining with business intelligence models to for deeper data analysis for useful pattern extraction, inferences for hidden insights along with predictive analysis, and filtering of huge amounts of information from big data. In addition to explanatory and discovery analytics, big data also offer golden path analysis consisting of a new technique of behavioral data analysis of human actions [3][4].

The big data landscape reflects vast changes from the traditional era of healthcare analytics in the use of advanced data management tools. Big data analytic tools in healthcare domains currently include Apache Sqoop, MongoDB, NoSQL, Apache Flume, Cassandra, Apache Hadoop, MapReduce, and Spark, among many others. This section provides the basis of Apache Hadoop and Map Reduce as well as the Hive Query Interface [18].

Apache Hadoop

Apache Hadoop is a powerful distributed software framework configured over servers for massive parallel clusters. Apache Hadoop is an open-source distributed system written in Java and can process data across clusters of computers. A conceptual view of the Hadoop architecture is described in Fig. 1.

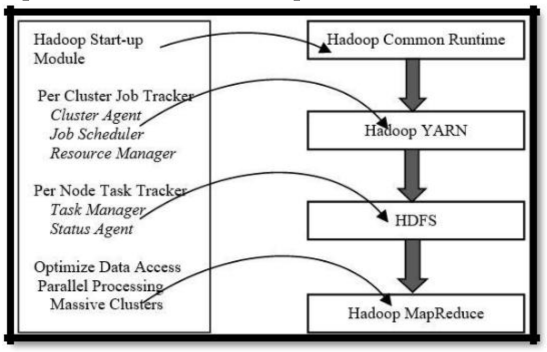


Fig.1. Conceptual view of Hadoop architecture [9] [10]

The Hadoop architecture consists of four types of modules including Hadoop Common Runtime, Hadoop YARN, HDFS, and Hadoop MapReduce [6]:

Hadoop Common Runtime: This includes Java libraries and utilities required for Hadoop modules. These libraries provide access to the Hadoop distributed file system and basic runtime libraries for the initialization of Hadoop start-up modules.

Hadoop YARN: This provides a layer for job scheduling and resource management.

HDFS: To optimize data access, the HDFS enables high throughput access to application data. Hadoop MapReduce interface focuses on parallelism to process large datasets.

Hadoop MapReduce: This is a software framework with two basic operations: Map Task and Reduce Task. Map Task is the first task and must be performed before Reduce Task. It accepts input data and converts it into a set of data tuples containing key value pairs. Reduce Task accepts the output of Map Task as input data and combines a set of relevant data tuples.

In general, MapReduce is a Java-based API following Hadoop YARN, which includes a scheduler for task scheduling and monitoring and re-executes any failed tasks. MapReduce consists of a single master Job Tracker for each cluster unit and a Task Tracker for each cluster node. Master Job Tracker is responsible for resource allocation to cluster units and for component task allocation to each cluster slave node. In addition, it monitors the node status and executes failed tasks to other available nodes. Task Tracker follows the node's operational working and collects results to send them back to Job Tracker. This provides confirmation feedback to the master about the node status and its efficiency [9] [10].

Role of Apache HIVE in Big Data Analytics

Apache Hive is a query and analysis tool constructed over databases and file systems that are integrated with the Hadoop architecture. Hive provides an SQL-like interface to query Hadoop-oriented databases and file systems. Traditional SQL queries are transformed using the MapReduce Java API in Hive, known as HiveQL. HiveQL provides a layer of abstraction to the programmer by integrating SQL queries into Java MapReduce, thereby eliminating the re-implementation of traditional query statements within MapReduce. This Hive interface transforms hive queries into an underlying MapReduce Java API and is executed over cluster nodes to provide distributed parallel database processing. In effect, Hive is a structured query operational tool that works over Hadoop. Hive resides on top of the Hadoop structure to query and summarize big data [14]. The Hadoop MapReduce operational flow is depicted in Fig. 2.

In addition to Hive, the Hadoop ecosystem includes other sub-tools such as Sqoop and Pig. Sqoop provides data migration from the traditional RDBMS schema to the HDFS schema. It provides import and export operations to and from HDFS and RDBMS. The programmer can transform RDBMS data definitions into HDFS datasets and vice versa. Pig is a Hadoop procedural language for constructing own scripts to map-reduce operations. Pig is like SQL scripting in RDBMS systems. Pig is required to make more advanced scripting operations and achieve dynamicity through the power of the MapReduce API.

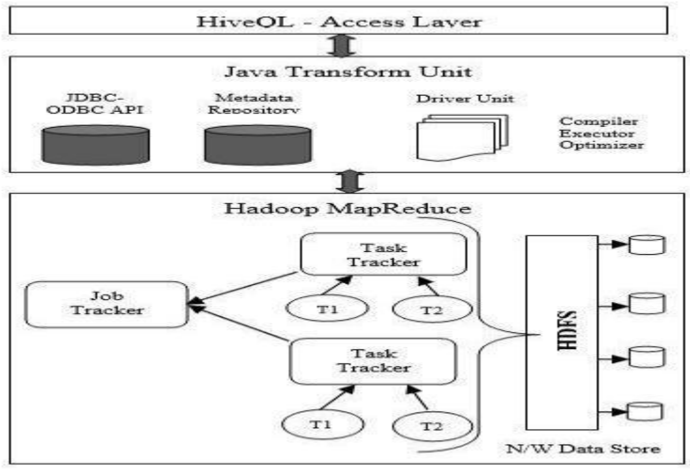


Fig.2. Hadoop MapReduce operational flow of Hive-based query using Hadoop MapReduce [9] [10]

3. Literature Survey

Data analytics, a field originating from data science, provides a strategic decision-making tool. Previous studies have focused on enhancing business efficiency in the decision-making process. Big data represent an enhancement to traditional data mining and warehousing approaches by incorporating not only mathematics and scientific techniques but also business intelligence and maturity models for business decision-making processes. This section provides a review of recent studies in the field of big data that focus on improving business decision-making processes.

Russom [1] surveyed big data analytics methods for classifying various industries and compared various big data analysis tools. Maltby [2] summarized various techniques and tools in data mining, text analytics, Hadoop MapReduce, and NoSQL and their application to the big data landscape in the industrial domain.

Manyika et al. [3] presented various data initiatives in using digital data or electronically stored documents and offered several ways to use big data in business analytics and intelligence domains. Zikopoulos and Eaton [4] focused on IBM research on big data in terms of their efforts to leverage the open-source big data technology infosphere.

Chen et al. [5] suggested the significant influence of big data trends on business intelligence and analytics. They provided an introduction to business intelligence and its trends. Holmes [6] highlighted various practical aspects of Hadoop MapReduce and social databases such as Facebook, Twitter, and LinkedIn.

Dittrich and Quiane [7] focused on big data efficiency within Hadoop and MapReduce for the underlying structure of the HDFS to store massive volumes of data and the optimization of data access. Madden [8] presented various ways to migrate traditional database technologies to big data and described major factors influencing relationships between traditional warehousing and online processing with big data analytics.

Patel et al. [9] addressed big data issues using Hadoop and MapReduce and covered the management of the massive volume from the exponential growth of corporate databases from day-to-day business transactions. Several issues have been listed as requiring processing of a terabyte of data on a daily basis. Because of the inability of existing database architectures and schema structures, this has produced serious big data issues in the industry. The authors covered Hadoop clusters implemented with the HDFS-based distributed data storage system in addition to a MapReduce parallel processing framework to address terabyte/petabyte of data.

Bakshi [10] described the architecture and approach of Hadoop with Hadoop common runtime API routines and the Hadoop Yarn-based parallel computing framework. This framework incorporates the MapReduce API and implements a Job Scheduler for each cluster Job and Task Trackers for each node. Job Tracker monitors the whole cluster operation, and Task Tracker monitors the operation of each cluster node and returns the status back to Job Tracker for each task as completed or failed. Failed tasks are rescheduled by Job Tracker in the following cycle.

O'Driscoll et al. [11] provided a list of applications for biotechnology and bioinformatics for processing large amounts of genome data. Biological data require massive levels of storage and processing for their analysis using matching algorithms and pattern-filtering techniques.

Katal et al. [13] highlighted big data issues, tools, challenges, and good practices and analyzed existing technology migration techniques in handling data-scaling issues. They also specified issues in the Hadoop system.

Qin and Li [14] described Hive, a Hadoop query interface, and explored HiveQL processing and its translation into a Java-enabled MapReduce API without requiring the programmer to design distributed parallel queries in Hadoop data.

Song et al. [15] listed various challenges in enhancing big data technology in cluster/grid leading. Smart grid operation includes heterogeneous data storage and multi-state data, which requires some degree of interoperability.

Wu et al. [16] focused on data mining using big data technologies and described data mining methodologies in big data analytics. Marz and Warren [17] specified services such as social networks, e-commerce, and web analytics for large-scale data management.

Zakir et al. [18] described issues in information systems and covered various big data technologies including Apache Sqoop, MongoDB, NoSQL, Apache Flume, Cassandra, MapReduce, and Spark. Oracle has also published a white paper on advanced analytics for the Oracle database, describing the merger of big data, statistical analysis, linkage with Hadoop, and sentiment analysis to transform a traditional business structure into big data technology.

4. The Proposed Work

This paper analyzes healthcare data using Apache Hive. The study uses a number of Hive queries for data analysis. For improved data analytics, Apache Hive is integrated with Tableau as a powerful analytics tool. This helps to examine the conversion of data into useful information. The use of Tableau helps analyze high-alert states with maximum deaths, major diseases resulting in more deaths, and deaths corresponding to deceasing diseases. Exponential smoothing is used to forecast disease trends. This section presents the proposed work and experiments for the analysis of big data in the EHR.

Experiments

We conducted a set of experiments to analyze EHR big data. The proposed technique was validated based on the dataset from the Centres for Disease Control and Prevention. The dataset was publicly available under the Open Data Commons Open Database License (ODbL) [20][21][22]. This dataset provided numbers for various causes of deaths in the United States and the reasons.

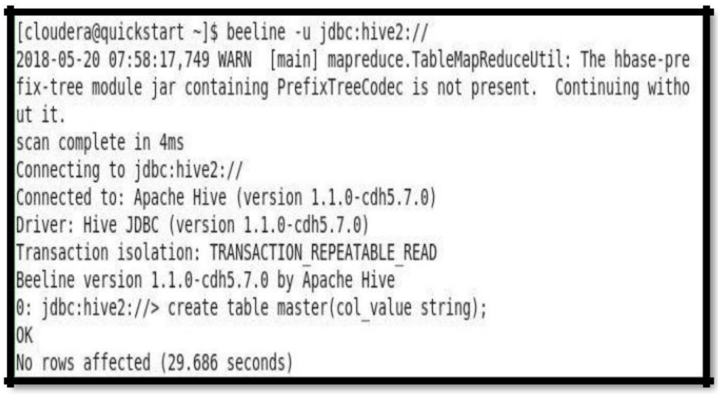
Experimental Setup

The proposed work involved the use of pre-built images of the Hadoop ecosystem by Cloudera QuickStart [23]. Virtual machine (single-node cluster) CDH5.13 was used to execute all Hive queries. This included the predefined installation of all prerequisites for using Hadoop and offered vast facilities for various software programs such as Hive, Pig, Spark, and Impala. For virtualization, VMware Player was used to distribute resources across guest operating systems for the parallel execution of different operating systems. It operates on top of the Windows environment and acts as a host for pre-built Hadoop images of Cloudera. HIVE integration was done using Tableau for better visualization [24].

Experiment Methodology

The proposed work was based on the following predefined steps:

Step 1. **Connection Establishment:** Beeline connections are established using the HIVE server with Java Database Connectivity (JDBC), as depicted in Fig. 3.



```
[cloudera@quickstart ~]$ beeline -u jdbc:hive2://
2018-05-20 07:58:17,749 WARN  [main] mapreduce.TableMapReduceUtil: The hbase-pre
fix-tree module jar containing PrefixTreeCodec is not present.  Continuing witho
ut it.
scan complete in 4ms
Connecting to jdbc:hive2://
Connected to: Apache Hive (version 1.1.0-cdh5.7.0)
Driver: Hive JDBC (version 1.1.0-cdh5.7.0)
Transaction isolation: TRANSACTION_REPEATABLE_READ
Beeline version 1.1.0-cdh5.7.0 by Apache Hive
0: jdbc:hive2://> create table master(col_value string);
OK
No rows affected (29.686 seconds)
```

Fig.3: Connection establishment to Hive Server by Beeline using JDBC

Step 2. Downloading Dataset: The dataset is downloaded from the Internet using a browser in a virtual environment and saved in *downloads* as a default location.

Step 3. Creation of Empty Tables: A table is created to store downloaded datasets using the following command:

```
create table tempp_master(col_value string);
```

Step 4. Loading the Dataset: The dataset is loaded from the downloads folder to the Hadoop environment using the following command:

```
LOAD DATA local INPATH
'/home/cloudera/Downloads/Health_Dataa.csv' OVERWRITE
INTO TABLE tempp_master;
```

Step 5. Creation of Table: A new table is created with each column having the same datatype as columns of the original dataset through the following command:

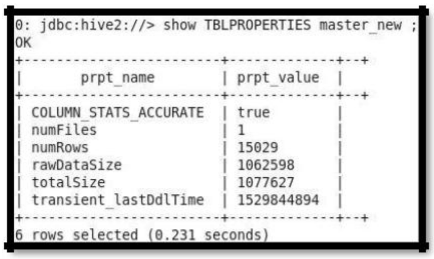
```
create table master_new (Year INT,Cause
VARCHAR(100),CnameSTRING,StateSTRING,DeathsINT,AdjustedDeat
h Rate FLOAT );
```

Step 6. Mapping of columns: Each column is mapped from the table storing the whole dataset to the table created in Step 5 using a regular expression based on the following command:

```
insert overwrite table masterr_new SELECT
regexp_extract(col_value,'^(?:([^,]*),?){1}',1)
Year,regexp_extract(col_value,'^(?:([^,]*),?){2}',1)
Cause,regexp_extract(col_value,'^(?:([^,]*),?){3}',1)
CName,regexp_extract(col_value,'^(?:([^,]*),?){4}',1)
State,regexp_extract(col_value,'^(?:([^,]*),?){5}',1)
Deaths,regexp_extract(col_value,'^(?:([^,]*),?){6}',1) AdjustedDeathRate
from tempp_master;
```

Step 7. Display the Table: To check the properties of the table created in Step 6 with all datasets stored in the columns, use the following command for the result shown in Fig. 4:

```
show TBLPROPERTIES master_new;
```



```
0: jdbc:hive2://> show TBLPROPERTIES master_new ;
OK
+------------------------+--------------+--+
|       prpt_name        |  prpt_value  |
+------------------------+--------------+--+
| COLUMN_STATS_ACCURATE  | true         |
| numFiles               | 1            |
| numRows                | 15029        |
| rawDataSize            | 1062598      |
| totalSize              | 1077627      |
| transient_lastDdlTime  | 1529844894   |
+------------------------+--------------+--+
6 rows selected (0.231 seconds)
```

Fig.4. Properties of the table used in the proposed work

5. Results and Discussion

We executed a large number of Hive queries to analyze based on the experimental methodology. We recorded various factors related to the Hadoop environment, including the number of Map, Reducers, cumulative CPU usage, HDFS Read, and HDFS Write, while executing and analyzed as described in the next sections.

Analysis Based on Queries Executed in Hive

We executed a number of HiveQL-based queries to analyze the identified dataset [25] [26] [27]. We also executed various queries in the proposed work, and their definitions are described as follows:

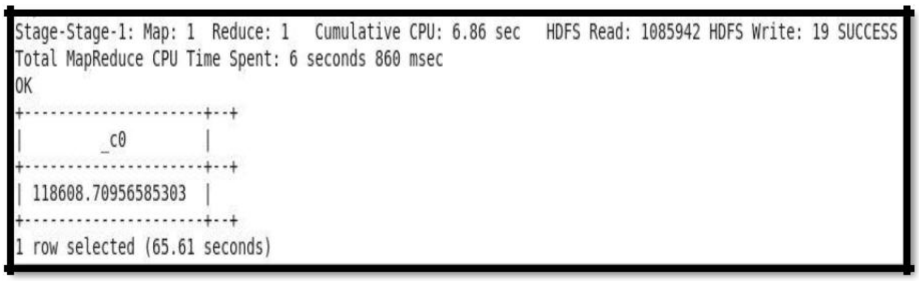
a. **Population Standard Deviation:**

This is a measure of the spread of scores for a given variable and is known as the population standard deviation:

$\sigma = \text{sqrt} [\sum (Y_i - \mu)^2 / M]$

The results are extracted using the following commands, and corresponding outputs are graphically depicted:

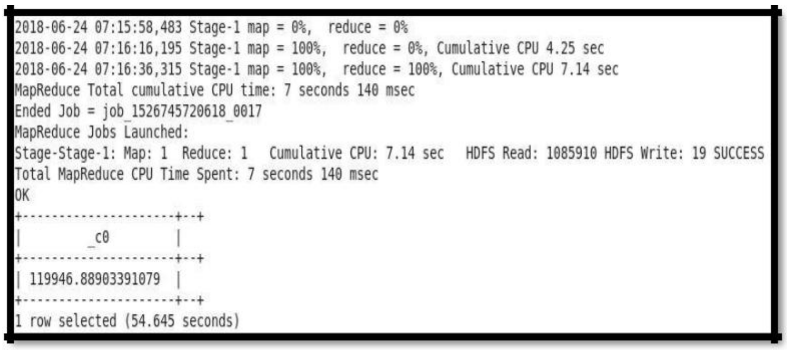
select stddev_pop(deaths) from master_new where year='2013';



```
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 6.86 sec   HDFS Read: 1085942 HDFS Write: 19 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 860 msec
OK
+---------------------+--+
|         _c0         |
+---------------------+--+
| 118608.70956585303  |
+---------------------+--+
1 row selected (65.61 seconds)
```

Fig.5. Output for population standard deviation

select stddev(deaths) from master_new where year='2014';



```
2018-06-24 07:15:58,483 Stage-1 map = 0%,  reduce = 0%
2018-06-24 07:16:16,195 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 4.25 sec
2018-06-24 07:16:36,315 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 7.14 sec
MapReduce Total cumulative CPU time: 7 seconds 140 msec
Ended Job = job_1526745720618_0017
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 7.14 sec   HDFS Read: 1085910 HDFS Write: 19 SUCCESS
Total MapReduce CPU Time Spent: 7 seconds 140 msec
OK
+---------------------+--+
|         _c0         |
+---------------------+--+
| 119946.88903391079  |
+---------------------+--+
1 row selected (54.645 seconds)
```

Fig.6. Output for standard deviation query

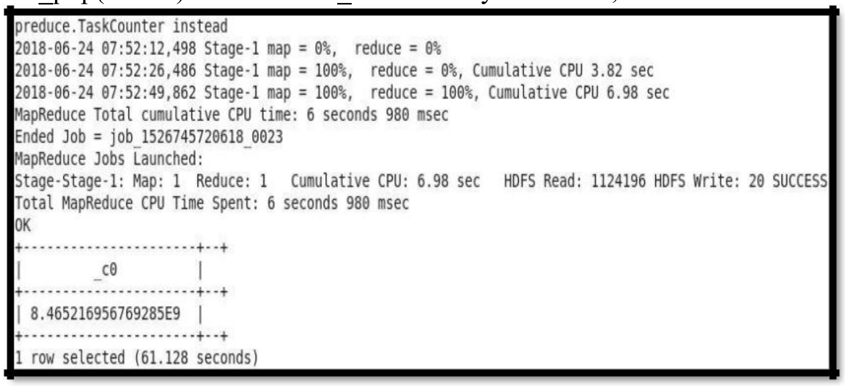
b. Population Variance:

This is the square of the population standard deviation and known as the population variance:

$\sigma^2 = \sum (Y_i - \mu)^2 / M$

The results are extracted using the following command, and corresponding outputs are graphically depicted in Fig 7:

select var_pop(deaths) from masterr_new where year='2010';



```
preduce.TaskCounter instead
2018-06-24 07:52:12,498 Stage-1 map = 0%,  reduce = 0%
2018-06-24 07:52:26,486 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 3.82 sec
2018-06-24 07:52:49,862 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.98 sec
MapReduce Total cumulative CPU time: 6 seconds 980 msec
Ended Job = job_1526745720618_0023
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 6.98 sec   HDFS Read: 1124196 HDFS Write: 20 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 980 msec
OK
+----------------------+--+
|         _c0          |
+----------------------+--+
| 8.465216956769285E9  |
+----------------------+--+
1 row selected (61.128 seconds)
```

Fig.7. Output for population variance query

c. Sample Standard Deviation:

This is a measure of the spread of scores in the sample for a given variable and is known as a sample standard deviation:

SSD= sqrt [$\sum (y_i - \text{Y\_bar})^2 / (m - 1)$]

The results are extracted using following command.

select stddev_samp (deaths) from master_new where year='2003';

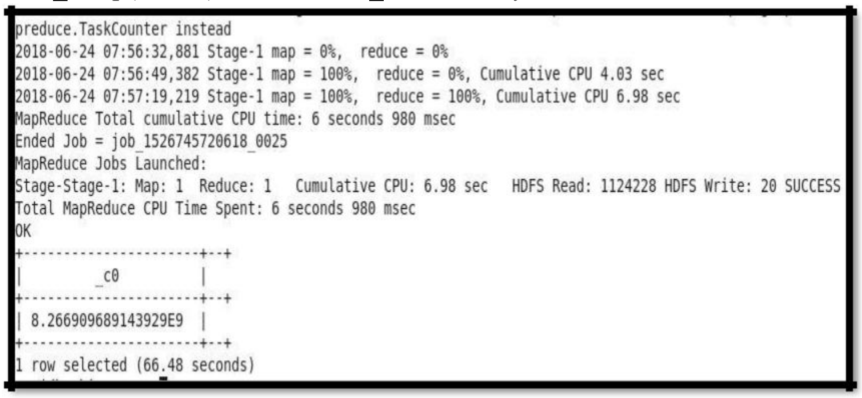
d. Sample Variance:

This is the square of the sample standard deviation and is known as the sample variance:

$$s^2 = \sum (y_i - Y\_bar)^2 / (m - 1)$$

The results are extracted using the following commands, and corresponding outputs are graphically depicted in Figs. 8 and 9.

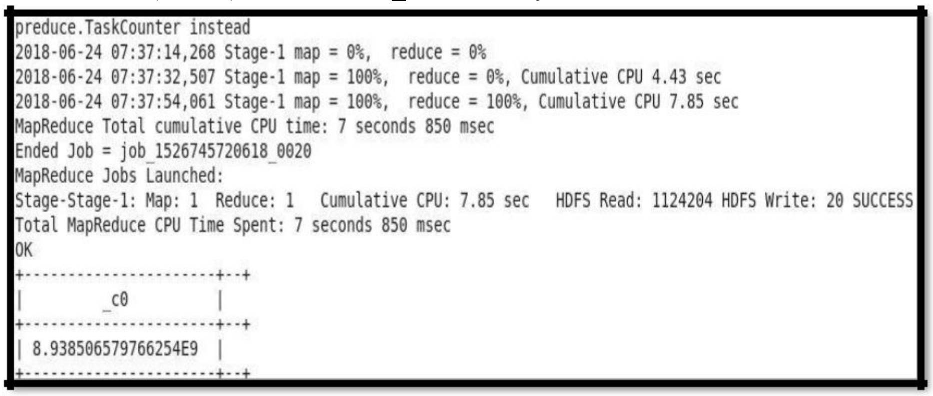
select var_samp(deaths) from masterr_new where year='2009';



```
preduce.TaskCounter instead
2018-06-24 07:56:32,881 Stage-1 map = 0%,  reduce = 0%
2018-06-24 07:56:49,382 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 4.03 sec
2018-06-24 07:57:19,219 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.98 sec
MapReduce Total cumulative CPU time: 6 seconds 980 msec
Ended Job = job_1526745720618_0025
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 6.98 sec   HDFS Read: 1124228 HDFS Write: 20 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 980 msec
OK
+----------------------+--+
|         _c0          |
+----------------------+--+
| 8.266909689143929E9  |
+----------------------+--+
1 row selected (66.48 seconds)
```

Fig.8. Output for sample variance query

select variance(deaths) from masterr_new where year='2012';

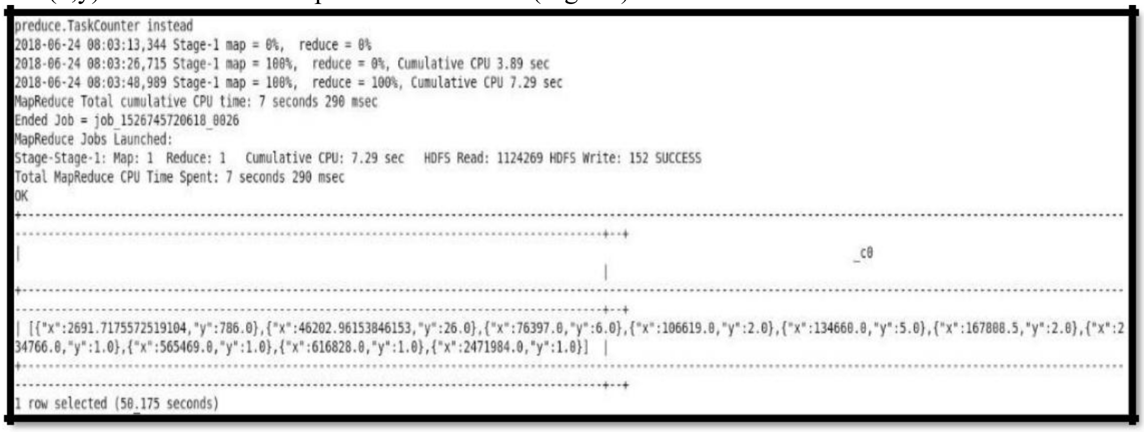


```
preduce.TaskCounter instead
2018-06-24 07:37:14,268 Stage-1 map = 0%,  reduce = 0%
2018-06-24 07:37:32,507 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 4.43 sec
2018-06-24 07:37:54,061 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 7.85 sec
MapReduce Total cumulative CPU time: 7 seconds 850 msec
Ended Job = job_1526745720618_0020
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 7.85 sec   HDFS Read: 1124204 HDFS Write: 20 SUCCESS
Total MapReduce CPU Time Spent: 7 seconds 850 msec
OK
+----------------------+--+
|         _c0          |
+----------------------+--+
| 8.938506579766254E9  |
+----------------------+--+
```

Fig.9. Output for variance query

e. Histogram_numeric (col, b):

This computes a histogram of a numeric column in the group using b non-uniformly spaced bins such that the output is an array of size b of double-valued (x,y) coordinates that represent bin centers (Fig. 10).



```
preduce.TaskCounter instead
2018-06-24 08:03:13,344 Stage-1 map = 0%,  reduce = 0%
2018-06-24 08:03:26,715 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 3.89 sec
2018-06-24 08:03:48,989 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 7.29 sec
MapReduce Total cumulative CPU time: 7 seconds 290 msec
Ended Job = job_1526745720618_0026
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 7.29 sec   HDFS Read: 1124269 HDFS Write: 152 SUCCESS
Total MapReduce CPU Time Spent: 7 seconds 290 msec
OK
+------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------+--+
|                                                                                                          _c0
                                                                              |
+------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------+--+
| [{"x":2691.7175572519104,"y":786.0},{"x":46202.96153846153,"y":26.0},{"x":76397.0,"y":6.0},{"x":106619.0,"y":2.0},{"x":134660.0,"y":5.0},{"x":167808.5,"y":2.0},{"x":2
34766.0,"y":1.0},{"x":565469.0,"y":1.0},{"x":616828.0,"y":1.0},{"x":2471984.0,"y":1.0}]  |
+------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------+--+
1 row selected (50.175 seconds)
```

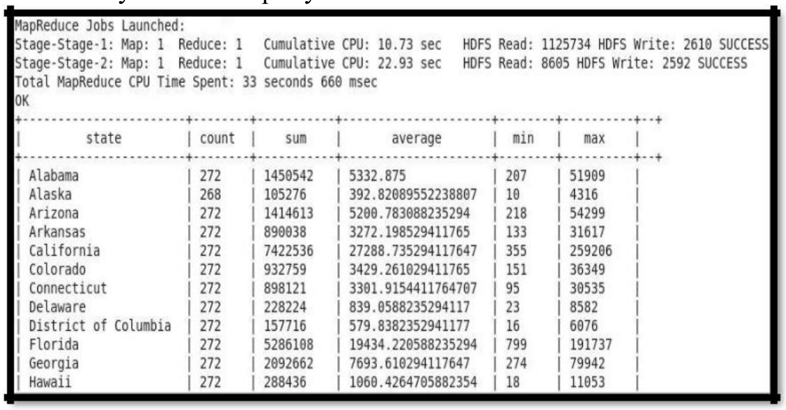
Fig.10. Output of histogram query

f. Query Using Group by and Order by Clause:

This provides data in a logical order. The results are extracted using the following commands, and corresponding outputs are graphically depicted in Fig. 11:

select state, count(deaths) as Count, sum(deaths) as Sum, avg(deaths) as Average,min(deaths) as Min,max(deaths)as Max from masterr_new group by state order by state;

Total time taken by the above query is *33 sec and 660 msec.*



```
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 10.73 sec   HDFS Read: 1125734 HDFS Write: 2610 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 22.93 sec   HDFS Read: 8605 HDFS Write: 2592 SUCCESS
Total MapReduce CPU Time Spent: 33 seconds 660 msec
OK
+-----------------------+--------+-----------+---------------------+--------+----------+--+
|         state         | count  |    sum    |       average       |  min   |   max    |
+-----------------------+--------+-----------+---------------------+--------+----------+--+
| Alabama               | 272    | 1450542   | 5332.875            | 207    | 51909    |
| Alaska                | 268    | 105276    | 392.82089552238807  | 10     | 4316     |
| Arizona               | 272    | 1414613   | 5200.783088235294   | 218    | 54299    |
| Arkansas              | 272    | 890038    | 3272.198529411765   | 133    | 31617    |
| California            | 272    | 7422536   | 27288.735294117647  | 355    | 259206   |
| Colorado              | 272    | 932759    | 3429.261029411765   | 151    | 36349    |
| Connecticut           | 272    | 898121    | 3301.9154411764707  | 95     | 30535    |
| Delaware              | 272    | 228224    | 839.0588235294117   | 23     | 8582     |
| District of Columbia  | 272    | 157716    | 579.8382352941177   | 16     | 6076     |
| Florida               | 272    | 5286108   | 19434.220588235294  | 799    | 191737   |
| Georgia               | 272    | 2092662   | 7693.610294117647   | 274    | 79942    |
| Hawaii                | 272    | 288436    | 1060.4264705882354  | 18     | 11053    |
```

Fig.11. Output for query using group by and order by clause

select * from masterr_new where YEAR='2015' AND state='Alaska';

Time Taken by above *query is 4.123 seconds*

g. Queries Using Partitioning:

A number of queries are executed using partitioning. Example queries are as follows:

create table masterr_new_part (Cause VARCHAR (100),CnameSTRING,DeathsINT,AdjustedDeathRate FLOAT)PARTITIONED BY (YEAR INT,State STRING);

insert overwrite TABLE masterr_new_part PARTITION (YEAR,State) select Cause,Cname,Deaths,AdjustedDeathRate,Year,State from masterr_new ;

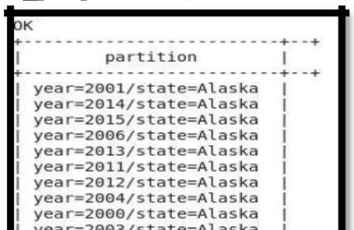
Whenever the amount of data is too large to be inserted in a portioned table, the following two parameters are set:

set hive.exec.max.dynamic.partitions=3000

set hive.exec.max.dynamic.partitions.pernode=1000;

This sets the value of the total number of dynamic partitions and dynamic partitions per node, as depicted in Fig. 12:

show partitions masterr_new_partpartition(State='Alaska');



```
OK
+--------------------------+--+
|        partition         |
+--------------------------+--+
| year=2001/state=Alaska   |
| year=2014/state=Alaska   |
| year=2015/state=Alaska   |
| year=2006/state=Alaska   |
| year=2013/state=Alaska   |
| year=2011/state=Alaska   |
| year=2012/state=Alaska   |
| year=2004/state=Alaska   |
| year=2000/state=Alaska   |
| year=2003/state=Alaska   |
| year=2005/state=Alaska   |
| year=2002/state=Alaska   |
| year=2010/state=Alaska   |
| year=2007/state=Alaska   |
| year=1999/state=Alaska   |
| year=2008/state=Alaska   |
| year=2009/state=Alaska   |
+--------------------------+--+
17 rows selected (0.085 seconds)
```

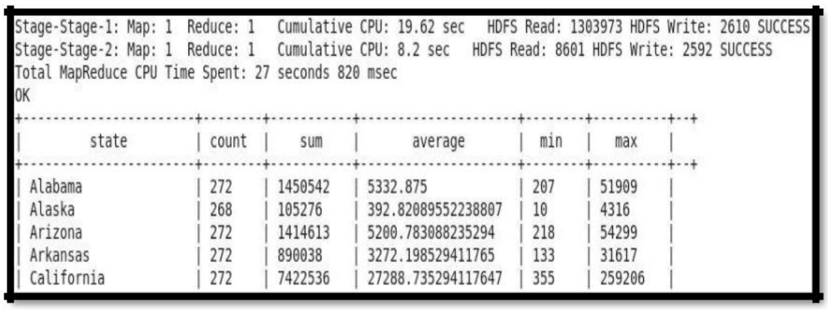
Fig.12.Hive portioned table

select * from masterr_new_part where YEAR='2015' AND state='Alaska';
Time taken 2.952 seconds

select state, count(deaths) as Count, sum(deaths) as Sum, avg(deaths) as

Average,min(deaths) as Min,max(deaths)as Max from masterr_new_part group by state order by state;

Time taken 27 sec 820msec



```
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 19.62 sec   HDFS Read: 1303973 HDFS Write: 2610 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 8.2 sec   HDFS Read: 8601 HDFS Write: 2592 SUCCESS
Total MapReduce CPU Time Spent: 27 seconds 820 msec
OK
+---------------------+-------+----------+-------------------+--------+---------+--+
|        state        | count |   sum    |      average      |  min   |   max   |
+---------------------+-------+----------+-------------------+--------+---------+--+
| Alabama             | 272   | 1450542  | 5332.875          | 207    | 51909   |
| Alaska              | 268   | 105276   | 392.82089552238807 | 10    | 4316    |
| Arizona             | 272   | 1414613  | 5200.783088235294 | 218    | 54299   |
| Arkansas            | 272   | 890038   | 3272.198529411765 | 133    | 31617   |
| California          | 272   | 7422536  | 27288.735294117647 | 355   | 259206  |
```

Fig.13. Output for the query using group by and order by clause for the portioned table

The advantage of partitioning is that it provides faster execution of queries, especially for large datasets, and results are required from a portion of the dataset. The reason for this faster execution is that partitioning creates virtual boundaries in the whole dataset such that when a query is executed, it is done so only within that boundary.

In Hive, query plans are executed in stages, and if the result for the next stage is not dependent on the previous stage, then parallel executions can be performed for a given query by setting values for the following parameters:

set hive.exec.parallel=true;

set hive.exec.parallel.thread.number=8;

From the first statement, parallel execution is enabled, and the second query enables the running of eight threads in parallel. That is, eight stages can be executed in parallel if they are independent. Therefore, parallel executions substantially increase cluster utilization.

Integration of Hive and Tableau

Tableau is a powerful data visualization and business intelligence tool with a user-friendly interface for a deep analysis. With Tableau, the user can quickly obtain valuable insights from vast Hadoop datasets. Tableau also enables deep knowledge of advanced query languages and makes big data more manageable through visual analysis. Hadoop is a type of data source for Tableau, and by using simple native connectors, Tableau can easily be connected to Hadoop, Hive, and Impala. With simple drag-and-drop data from Cloudera Enterprise with Tableau, visualization is made easy for given data.

Heat maps are used for insightful visualization. They use color and size to express values, so the user can quickly obtain a wide array of information. Heat maps have wide application when comparisons are required using large numbers of datasets.

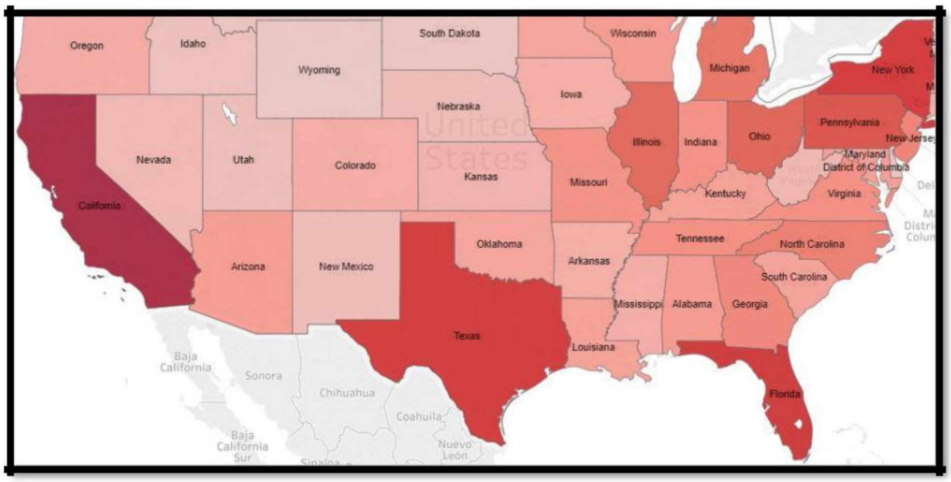


Fig.14.Heat map for different states corresponding to the number of deaths

In Fig.14, the states of California, Texas, Florida, and New York are on high alert, as indicated by the brightness of the color red is greater for these states. Therefore, all further research is focused on these states. With Tableau, main reasons for deaths in these states and numbers of deaths can be inferred through the visualization provided by Tableau.

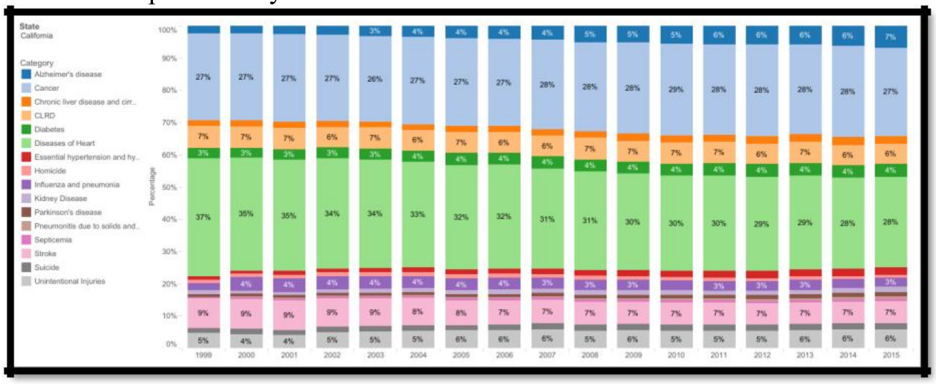


Fig.15. Visualization of reasons for death in California and its percentage distribution

From Fig. 15, the following findings can be summarized:

i. Cancer, heart disease, and septicaemia are the main causes of death in California.

ii. The number of deaths from cancer and diabetes remains constant since 1999.

iii. Although diseases related to the heart represent one of the main causes of death in California, but the percentage decreases over time.

iv. Unexpectedly, Alzheimer's disease and unintentional injuries(accidents) come into existence, increasing the number of deaths.

v. The number of deaths from septicaemia, suicide, and stroke shows a decreasing trend.

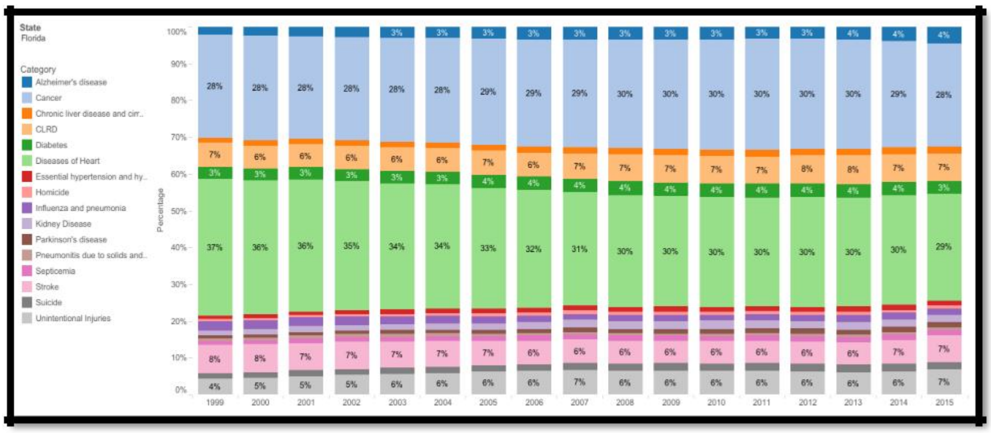


Fig.16. Visualization of reasons for death in Florida and its percentage distribution

From Fig. 16, the following findings can be summarized:

i. The number of deaths from Alzheimer's disease and unintentional injuries(accidents) is increasing.

ii. Cancer and heart disease are the main causes of death in Florida. The number of deaths from heart disease is decreasing, and that of cancer remains constant.

iii. Deaths from chronic liver disease, cirrhosis, diabetics, and septicaemia remains constant.

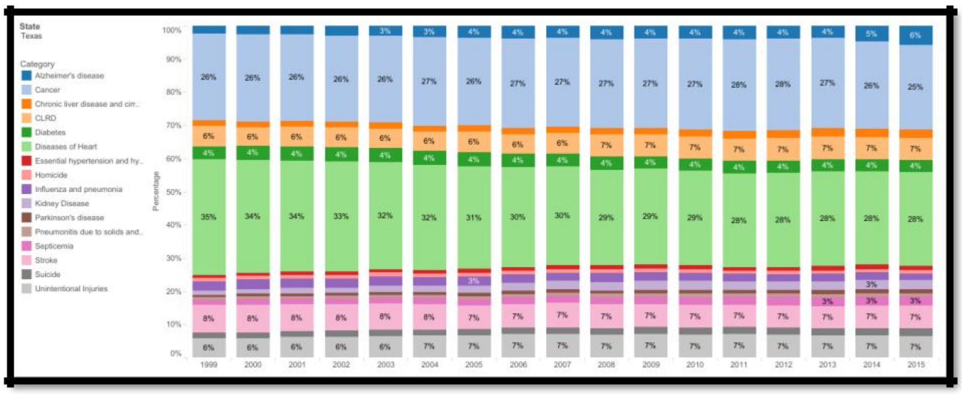


Fig.17.Visualization of reasons for death in Texas and its percentage distribution

From Fig. 17, the following findings can be summarized:

i. The number of deaths from Alzheimer's disease is increasing.

ii. Cancer and heart disease are the main causes of death in Texas. The number of deaths from heart disease and cancer is decreasing over time.

iii. Deaths from diabetes, stroke, and unintentional injuries(accidents) remain constant.

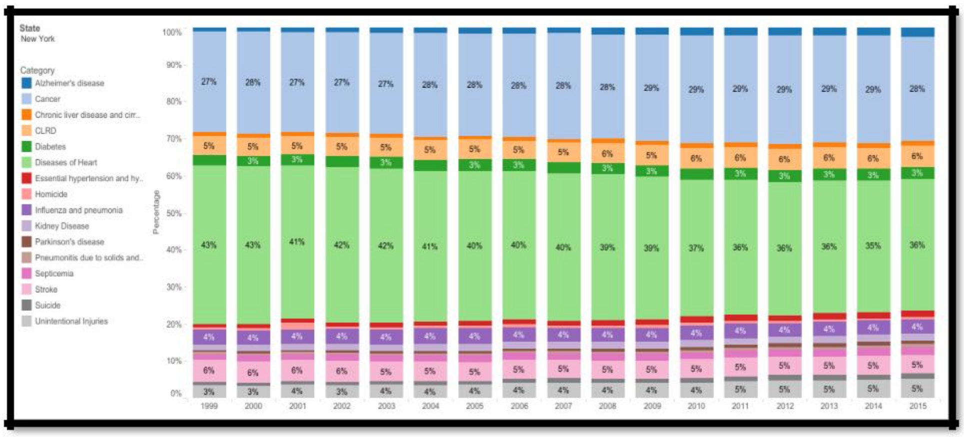


Fig.18.Visualization of reasons for death in New York and its percentage distribution

From Fig. 18, the following findings can be summarized:

i. Cancer and heart disease are the main causes of death in Texas. The number of deaths from heart diseases is decreasing, and that of cancer remains constant.

ii. Deaths from influenza and pneumonia remain constant.

iii. Deaths from diabetes, stroke, unintentional injuries(accidents), and chronic lower respiratory diseases remain constant.

Forecasting Four States with Maximum Deaths

This study proposes the use of exponential smoothing for forecasting states with maximum deaths and the number of deaths. In exponential smoothing, more weight is given to recent observations than older ones. In this study, we used various quality and quantity metrics to measure the error [12]:

(t) Index of a period in a time series,

(N) Time series length,

(M) A number of periods in a season/cycle,

(A(t)) The actual value of the time series at period t,

(F(t)) Fitted or forecasted value at period t.

The residuals are e(t) = F(t)-A(t)

We used the following metrics to measure the error:

i. The root mean squared error (RMSE) is calculated as

$$\sqrt{\left(\frac{1}{n}\right)\sum e(t)^2}.$$

ii. The mean absolute error (MAE) is calculated as

$$\frac{1}{n}\sum \left|e(t)\right|.$$

iii. The mean absolute scaled error (MASE) measures the magnitude of the error of a naive one-step ahead and that of the error forecast as a ratio:

$$\frac{\frac{1}{n}\sum \left|e(t)\right|}{\frac{1}{(n-1)}\sum_{2}^{n}\left|Y(t)-Y(t-1)\right|}.$$

iv. The mean absolute percentage error (MAPE) measures the error's magnitude by comparing it with given data as a percentage. The lower the measure of MAPE, the better it is:

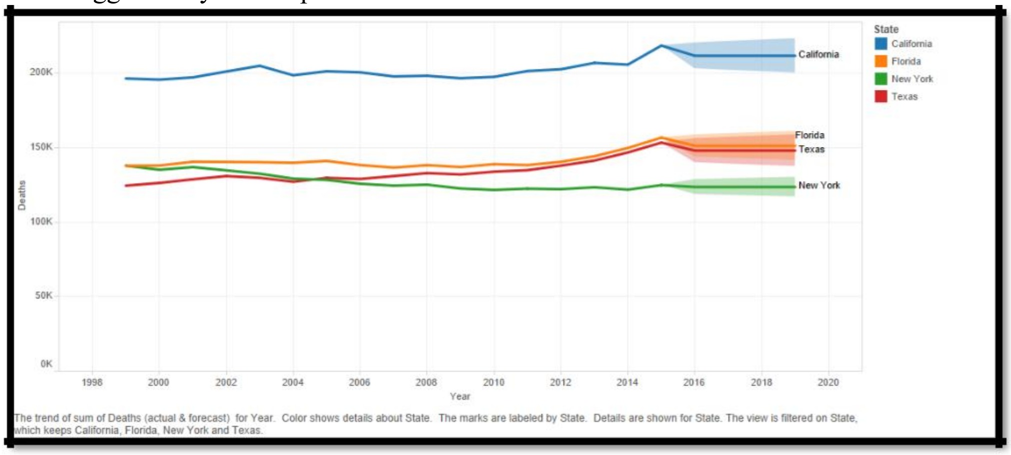
$$100\frac{1}{n}\sum\left|\frac{e(t)}{A(t)}\right|$$

v. The Akaike information criterion (AIC) is a model quality measure that penalizes complex models to prevent overfitting:

$$n * \log(SSE/n) + 2 * (k+1),$$

where SSE is the sum of squared errors and k is the number of estimated parameters, including initial states.

Forecasted states from exponential smoothing are shown in Fig. 19, and error values are shown in Table 1. The number of deaths is expected to further increase in California and Florida, indicating a need for appropriate preventive measures. It has also been observed that certain diseases require proper attention, as suggested by their rapid increases over time.



The trend of sum of Deaths (actual & forecast) for Year. Color shows details about State. The marks are labeled by State. Details are shown for State. The view is filtered on State, which keeps California, Florida, New York and Texas.

Fig.19.Forecasting of four states using exponential smoothing

Table 1.Values of various quality metrics for proposed work

Detail	Quality Metrics				
State	RMSE	MAE	MASE	MAPE	AIC
Texas	4,096	3,049	1.23	2.20%	289
New York	2,540	2,192	1.25	1.70%	273
Florida	3,743	2,395	1.16	1.60%	286
California	4,424	3,094	1.01	1.50%	291

6. Conclusions and Future Research

This study analyzes EHR big data using Hive queries and forecasts high-alert states with maximum death rates. We analyze big data extracted from the given dataset to forecast high-alert states using exponential smoothing and visualize the results using Tableau.

The results suggest an urgent need to address the alarming situation of high numbers of deaths in four states, namely California, Florida, New York, and Texas. Major causes of diseases such as cancer and heart disease remain constant, and in some cases such as Alzheimer’s disease and unintentional injuries(accidents), the numbers are increasing.

This study uses exponential smoothing for forecasting purposes, and as an extension to this study, machine learning techniques such as neural networks may produce better prediction results.