README.md

Wednesday, December 7, 2022 3:27 PM

Project Goals:

Create price transparency for medical procedures for individual's receiving medical care, ultimately driving towards empowering individuals to demand price transparency and opportunities to negotiate prices. Design an application which enables this information to be presented and accessed in a scalable way.

Within the application users can lookup a hospital procedure and see the low and high-end range of a negotiated charge for a Chicago hospital performing the procedure. Furthermore, someone can enter in the bill amount they paid for a procedure at a Chicago hospital.

Project Video:

https://drive.google.com/file/d/17336n0zDnYbjaDwHhld_z7Ix5DG-I9tY/view
WEMB file has also been provided in the zip file.

Data & Design:

Chargemaster | Northwestern Medicine
Standard Charges| Rush University Medical Center
2022 CPT Codes Provided by the CDC

Healthcare data is complex and intentionally obscured to make transparency initiatives by the public extremely difficult. In addition to the Big Data Application architecture, a significant piece of this project was thinking through how to connect the data in a meaningful way.

On July 1, 2022, a federal rule went into effect requiring health plans to disclose the negotiated prices they pay physicians and facilities for each item and service they provide. As a result, most hospitals provide some information about the negotiated charges of their procedures. This data is intentionally presented in a manner that makes it difficult to understand and most-likely is not comprehensive of all the services performed at the hospital. To parse through and determine how I would best link datasets across hospitals, I downloaded each hospital dataset to my computer to evaluate and clean locally.

Medical services performed at a hospital are reviewed and categorized according to a series of established guidelines. One of these guidelines is established by the American Medical Association and uses CPT Codes to describe different medical services. Both hospitals provided at least some of their medical procedures with a CPT Code. This felt like a good way to filter for like procedures across hospitals. In order to enforce common service descriptions, I joined each hospital's table with documentation given by the CDC, defining the category and operative procedure by CPT Code.

Finally, It was not possible to match insurances across hospital. However, common fields related to charged prices for each hospital were the <minimum negotiated charge>, <maximum negotiated charge>, and <discounted cash price>. This information enabled me to present the minimum and maximum range of negotiated charges for a given procedure by hospital. I then matched the lowest cost negotiated price and maximum negotiated price with the respective insurer.

Big Data Architecture:

This project leveraged the lambda architecture. The Batch Layer was constructed using Hive and Java Script. The Speed Layer enables an interface for users to input a hospital bill they received for a hospital procedure. The speed layer is written with Scala and uses manually entered data to update the Serving Layer with an increment object.

Batch Layer:

I used a Secure Copy Protocol (SCP) to transfer locally stored CSV files over to the cluster's name node and then put into HDFS. For the reasons mentioned above, the format of the data required initial processing therefore this project does not follow the Sushi Principal of Big Data. I did download and maintain raw copies of data on my local computer, therefore if I needed to revisit a raw data file I'd have access. Each hospital does not make historical archived records available online,

therefore maintaining a raw data source in another location is important. Overtime, it would also be interesting to analyze trends in negotiated rates across insurers as well.

Once placed in hdfs as a csv file, I created externally managed hive data tables for each file. As more data is released by each hospital I could follow the same steps to clean and add CSVs to each respective folder and rerun the batch layer queries without any changes to them and update the serving layer's hbase table with the latest negotiated rates.

Next, I setup an ORC table to run further calculations on each hospital-level data. I needed to get the maximum and minimum charged rates by insurer. Although in retrospect, ORC was not the correct choice for this. The calculations I ran looked at each individual row to identify the maximum and minimum charges. An ORC table, is more efficient when calculations are done at the column level. Therefore, on a future iteration I would go through and not change the data to an ORC structure.

Lastly in the query to transfer the joined Hive called chi_hospitals over to Hbase, I added two static column <num_inputted_bils> and <inputted_bills_sum> to serve as a placeholder when user inputted data comes through via the speed layer.

Serving Layer:
I decided to represent each database row in my Serving Layer Hbase table as a procedure performed at one of the two hospitals I selected to review in Chicago: Northwestern Medicine and Rush University Medical Center. This would be best represented by concatenating the procedure code with the hospital for my mapping key. Since Hbase sorts its data alphabetically by mapping key, this increased the efficiency of searching for data by CPT procedure code. Hospitals that shared a CPT code are found next to each other. The query a user performs will search for negotiated charges by procedure, this lent to a more efficient query by using a prefix filter to isolate the data for a requested procedure.

Speed Layer:
The speed layer asks users to enter in the cost of their latest hospital bill. I generated a Kakfa topic ecjackson_hospital_input to receive the data. From the Kafka streaming I directly create an increment object which is used to update the hbase table with the number of invoices that have been entered for a given procedure and the total sum of all charges paid for the same procedure. This enabled data per hospital per procedure to need only ever need one row within the Hbase table.

It was not necessary for me to first generate a scratch Hbase table since the data entered at the speed layer is processed directly into the serving layer. Drop downs protect data entry and ensure any value entered by a user will be able to match back to an existing mapping key in the Hbase table. If I had more time, I'd strengthen data integrity protocols on the front end interface by requiring 100% completion before the form can be submitted.

The speed layer increment object directly updates the serving layer. For each procedure in which hospital input data is recorded the user will now see a calculated avg of paid charges for a procedure and can compare these against the negotiated charges.

-- *launch kafka streaming to increment 'chi_hosp' with user-inputted paid bills data*
-- *within hadoop*
cd ecjackson/src/target

spark-submit --master local[2] --driver-java-options "-Dlog4j.configuration=file:///home/hadoop/ss.log4j.properties" --class StreamBills uber-speedlayer_bills-1.0-SNAPSHOT.jar b-2.mpcs530142022.7vr20l.c19.kafka.us-east-1.amazonaws.com:9092,b-1.mpcs530142022.7vr20l.c19.kafka.us-east-1.amazonaws.com:9092,b-3.mpcs530142022.7vr20l.c19.kafka.us-east-1.amazonaws.com:9092

Scalable User Access:
The project uses Code Deploy and the two webpages /index.html and /submit-hospital-bill.html for scalable access if traffic

were to increase.

Query Hospital Procedure Charges
Submit Healthcare Invoice Data