**Continuous Pipeline**

**Heart Attack Predicitons**

Andrew Pfeifer

**Introduction:**

The goal of this project is to take a data set and process it through a continuous data pipeline in a big data environment. The first step of the process was to load a dataset into HDFS from NiFi. Next, the data will be loaded into hive then pulled into pyspark  and have machine learning applied to it. Lastly, the metrics of machine learning were to be uploaded and retrieved from HBase. The following will discuss this process, what I tried and any technical issues I ran into.

**Data set:**

The data set I chose to work with is heart attack data from Kaggle. This can be located from the following link:

<https://www.kaggle.com/datasets/fatemehmohammadinia/heart-attack-dataset-tarik-a-rashid/data>

This data set contains 9 columns:

Age: The age of the patient.

Gender: The sex of the patient. This is set as a binary variable.

Heart rate: The heart rate of the patients' heartbeats per minute.

Systolic blood pressure: This is the pressure in the heart during muscle contractions.

Diastolic blood pressure This is the pressure in the heart in-between the heartbeats.

Blood sugar: This is the blood glucose level of the patient.

CK-MB: This is an enzyme that is released during muscle damage to the heart.

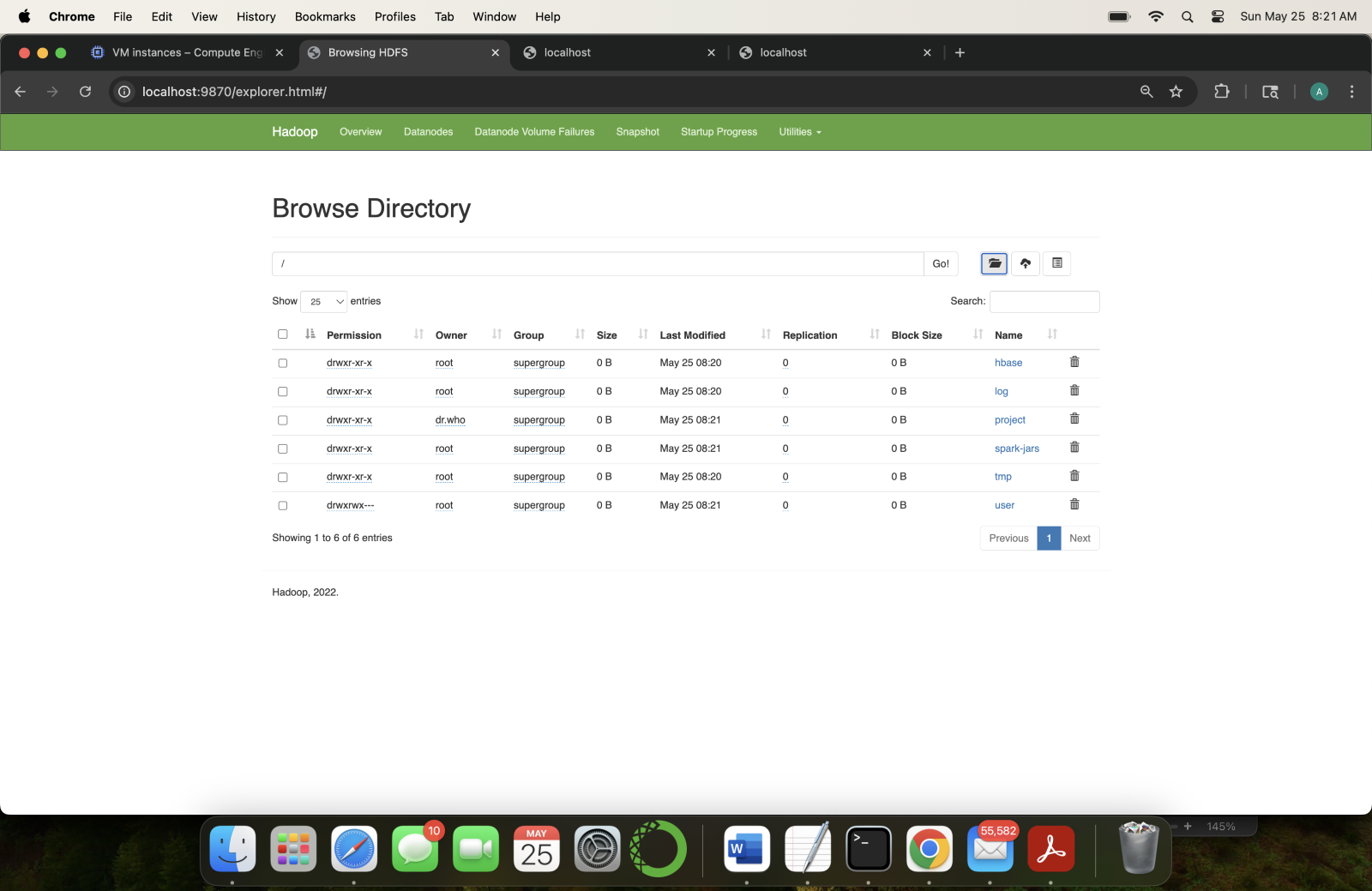
Troponin: This is a biomarker that indicates heart damage/injury.

Result: Whether the patient had a heart attack or not. Binary, but uses positive to indicate a heart attack, negative otherwise.

**Step 1: Loading data into HDFS from NiFi.**

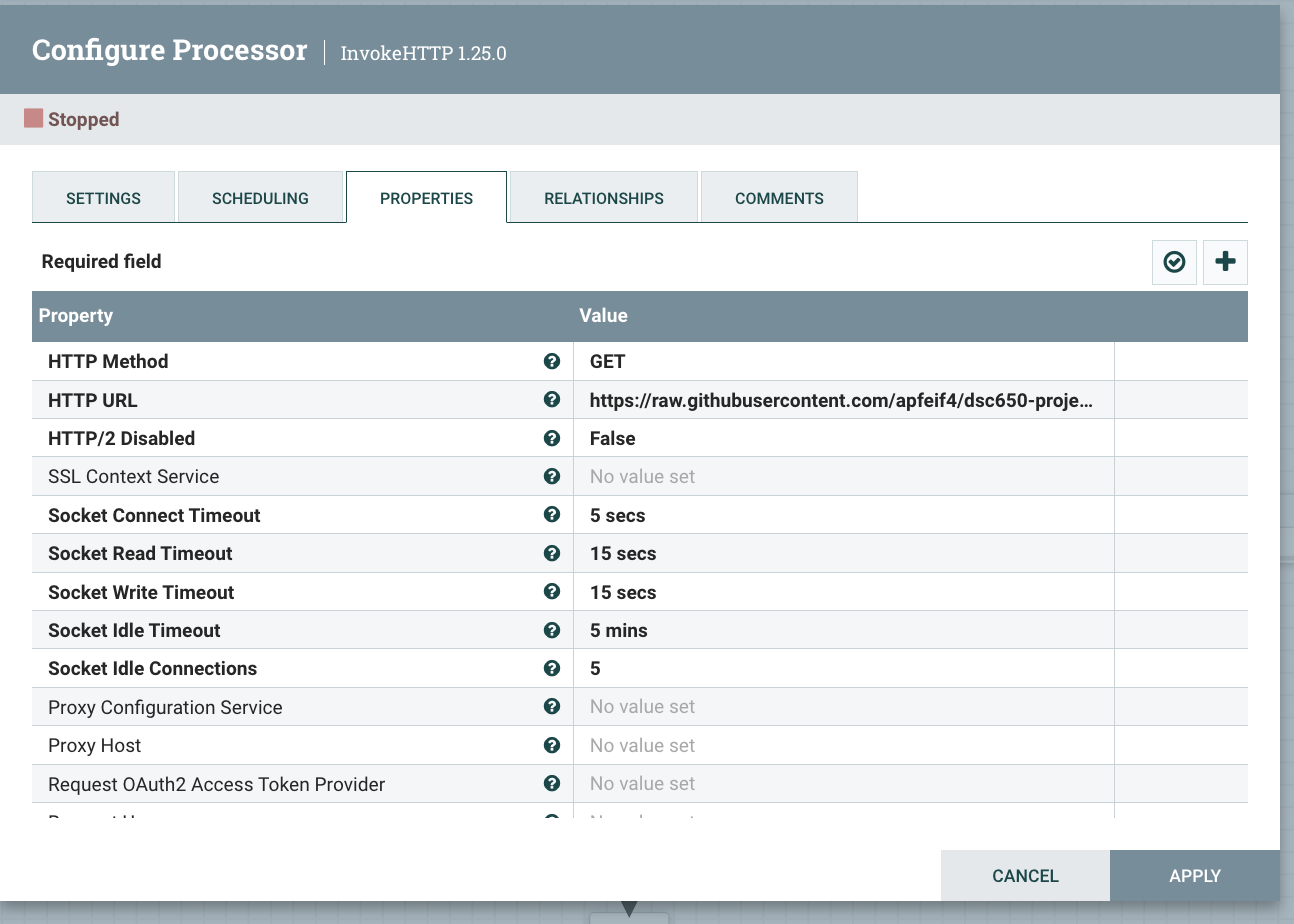
*New directory opened in HDFS:*

The first thing I did was to start the docker containers in the Hadoop environment. I opened the local host and waited for all the directories to open before proceeding. Once they were open, I made a new directory for the project called “project.”

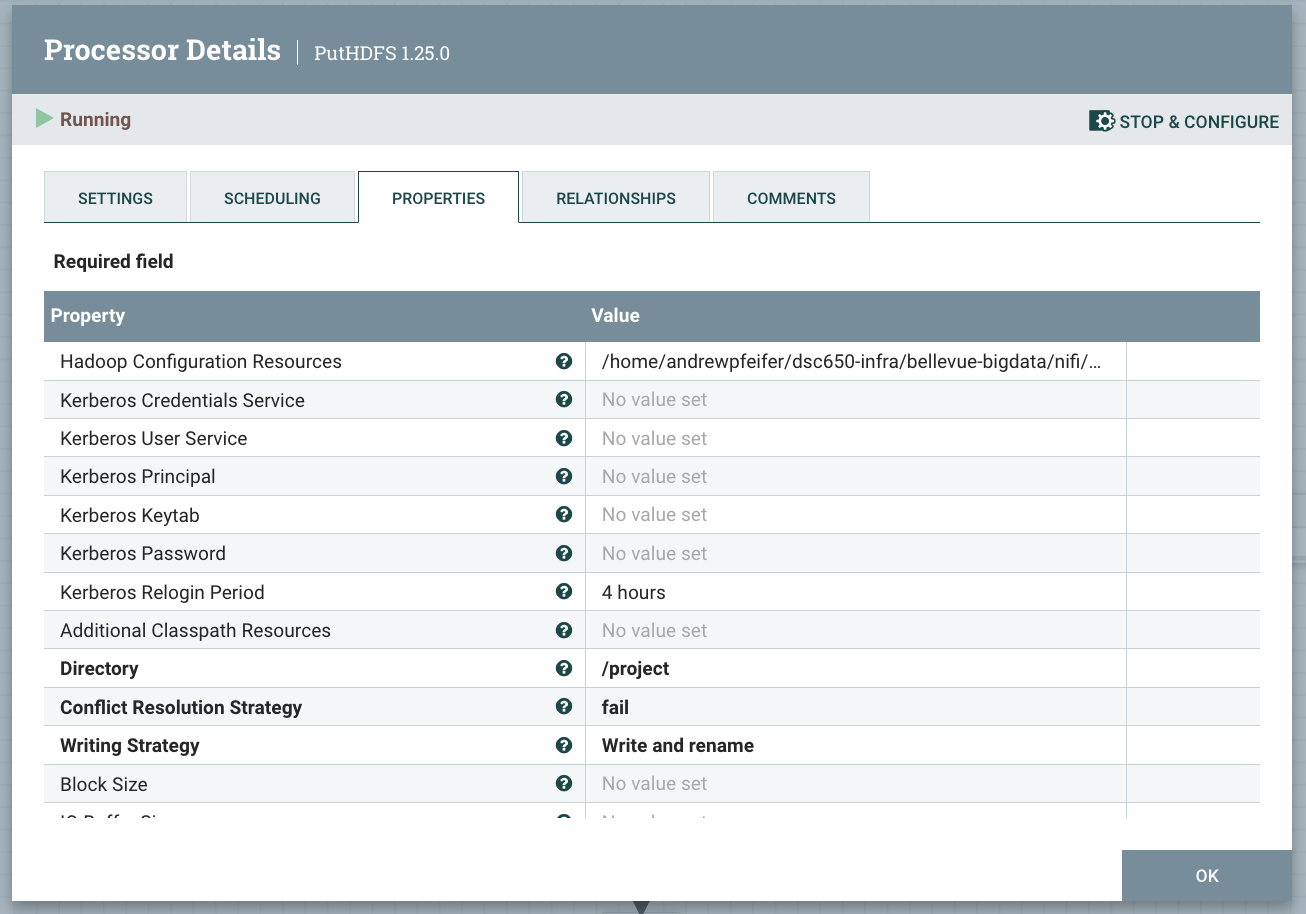


*Data flow in NiFi to HDFS:*

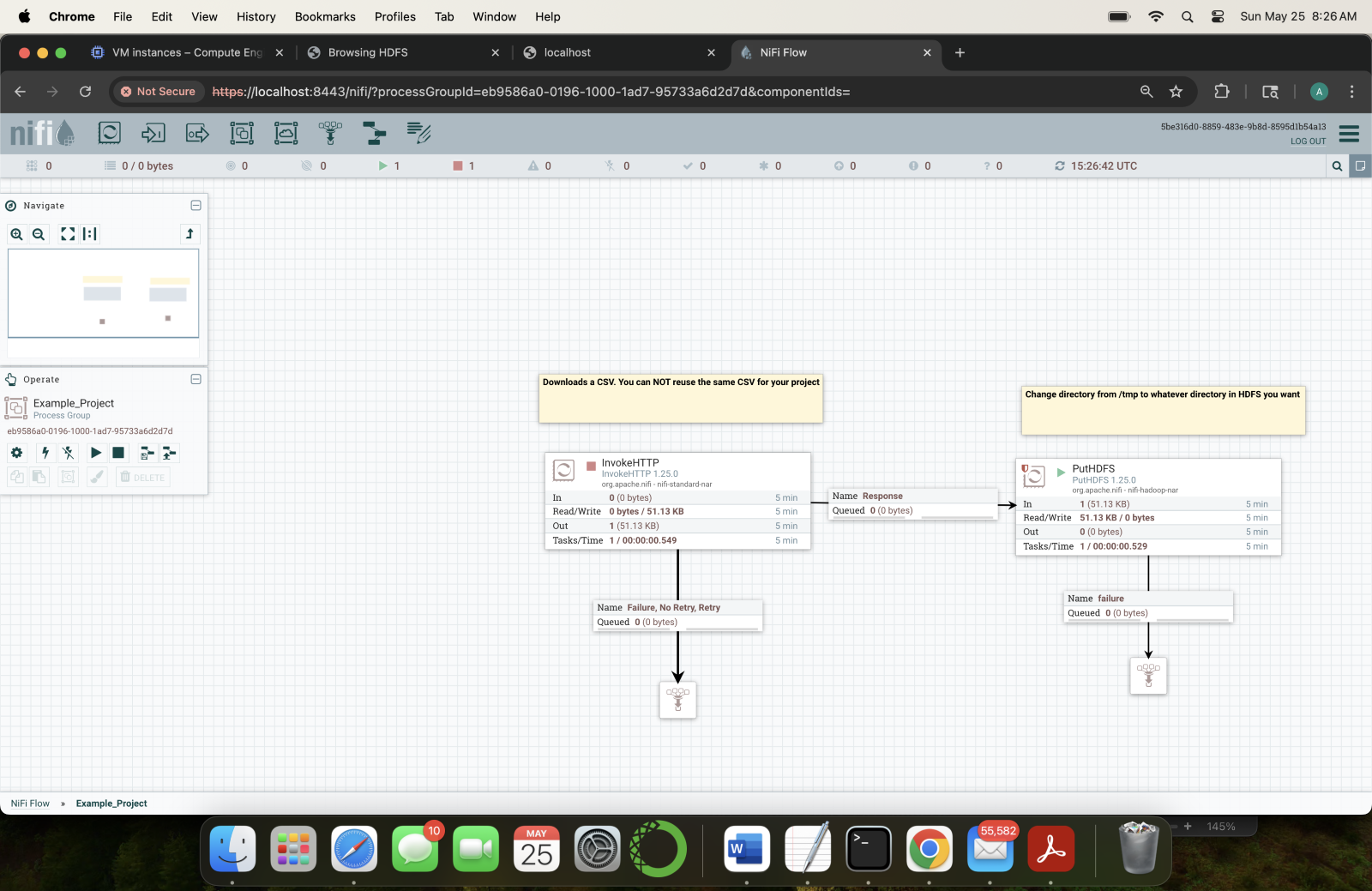
Before the data can be processed through NIFI, we need to obtain a raw JSON file of the data. This can be done by uploading the csv file to GitHub and obtaining a URL for the raw data. Then start the Nifi containers. Once this is done, the HTTP URL in the invoke HTTP processor group needs to be updated with the address as shown below.



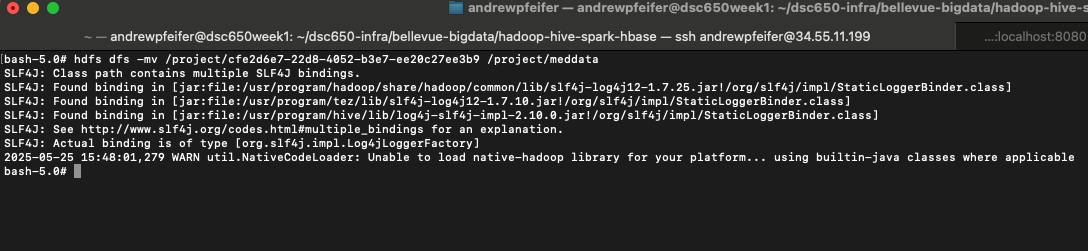
The next step is to go into the configuration for the PutHDFS processor group and update the directory name that the data is to be uploaded to. Also, in the terminal, one would need to run the whoami command so that you know your username. Next, the username replaces the word ‘ubuntu’ in the Hadoop Configuration Resources line.



Once the processor groups are updated, the Put processor can be started and the invoke processor should be started only once. After refreshing, one can see that the data will have successfully flowed to the HDFS directory, and the Nifi containers can be stopped.

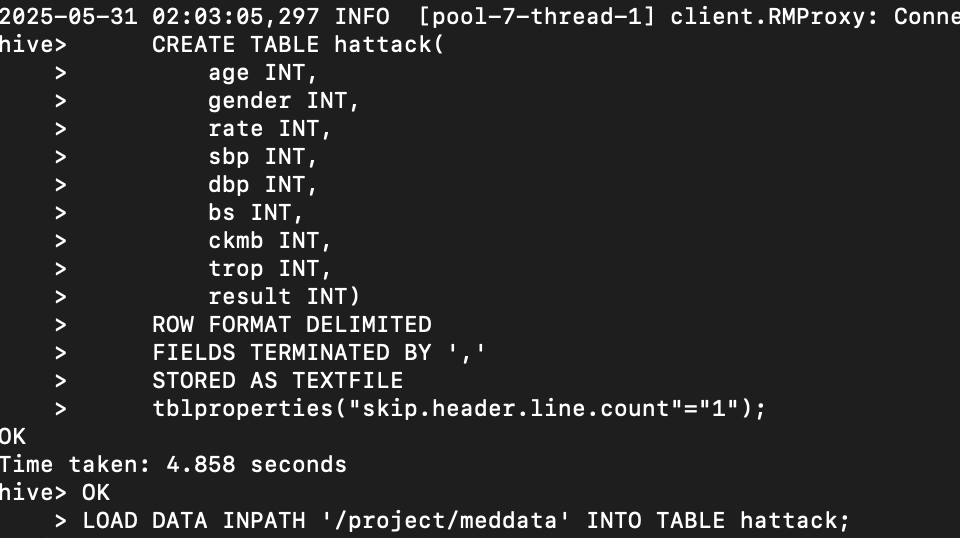


The last step in this process is to rename the file in HDFS. When the data is processed through NiFi to HDFS, it is saved as a randomly generated string of characters and numbers. To change the name of the data, enter the Hadoop environment in the terminal and enter the master bash. Once here the data’s name can be changed by using the command hdfs dfs –mv /path/oldname /path/ newname. Below is a screenshot of this successfully being completed to change the name to meddata.

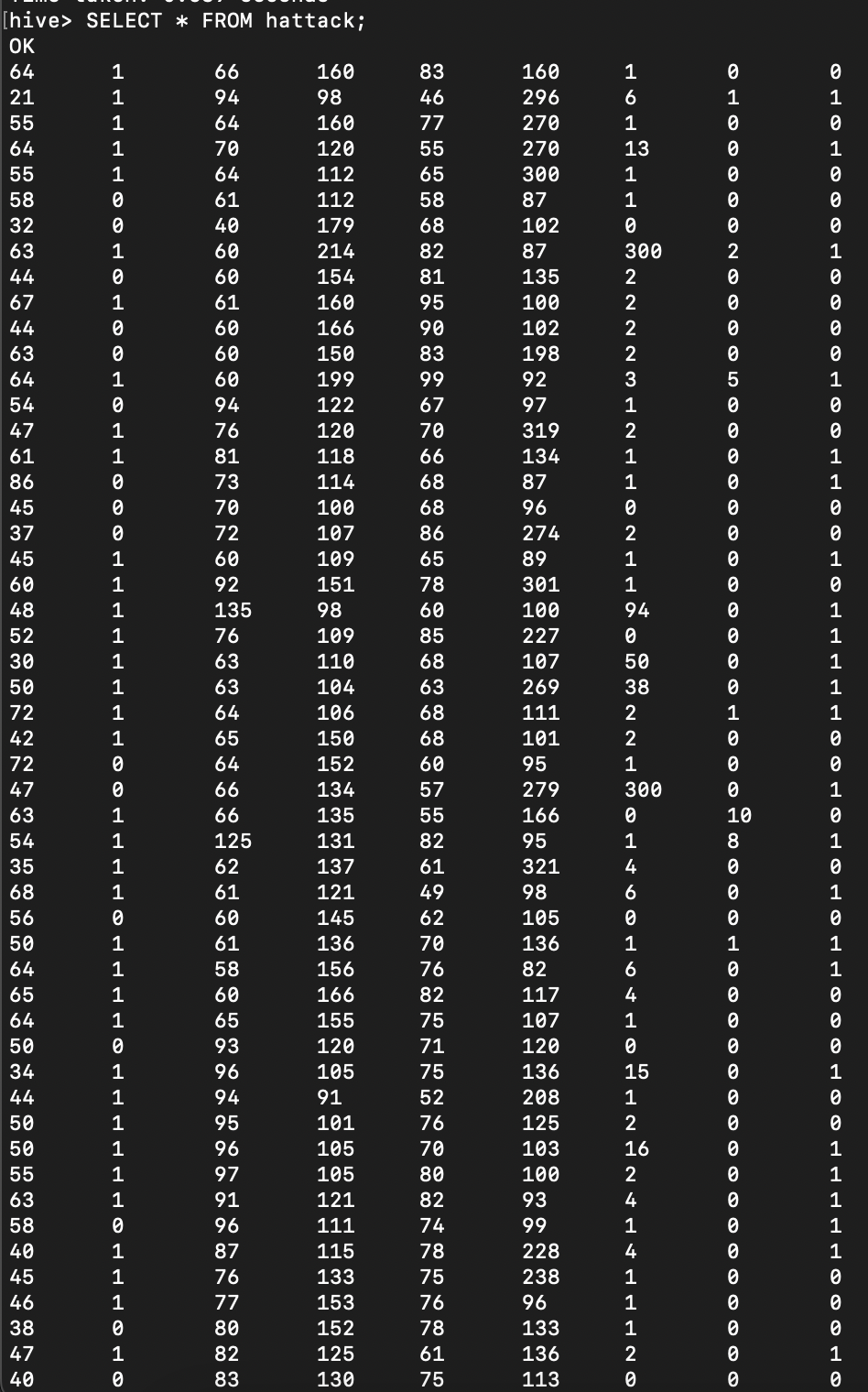


**File sent to hive:**

The next step in the process is to send the file from HDFS to Hive. To start, a table of columns needs to be created. This will need to be made from all the existing columns in the data set as well as the data type. The variable names were changed and abbreviated to save space in the output table. Also, spaces were removed as these can cause issues when calling the data in the upcoming systems. When the table is made, the data can be sent to fill in the table. Both commands are shown below as well as an output showing that this was successfully completed.

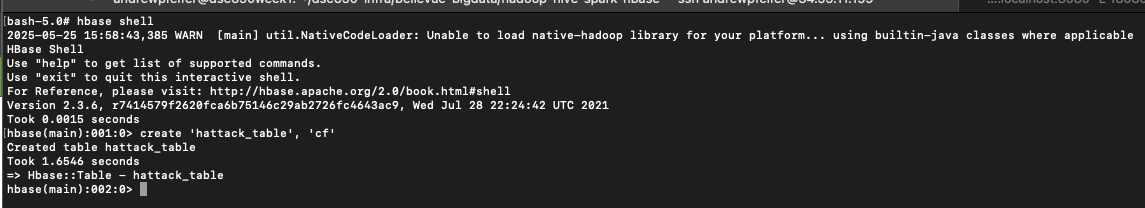


*Output Select all data:*



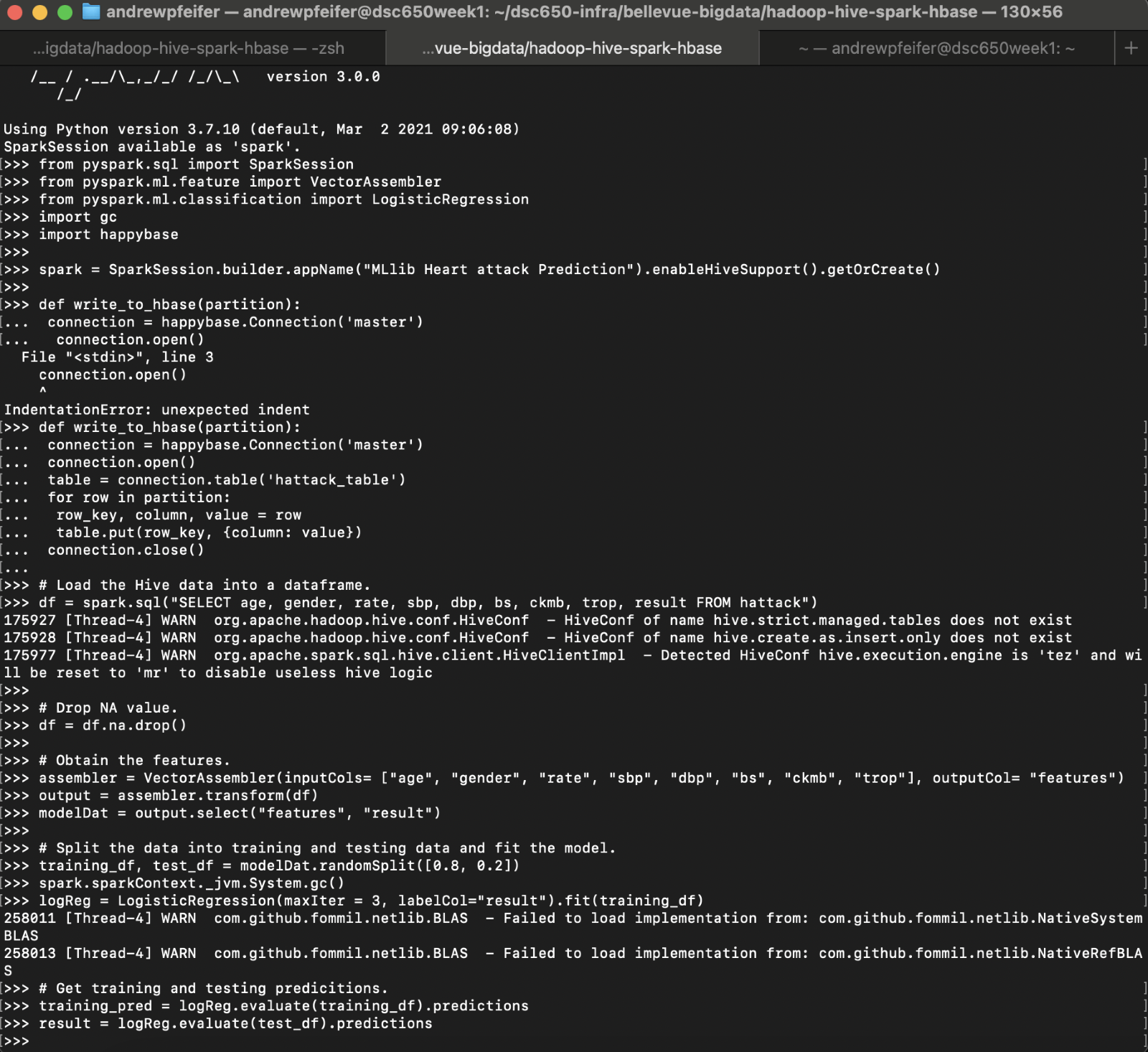
**Create and apply machine learning:**

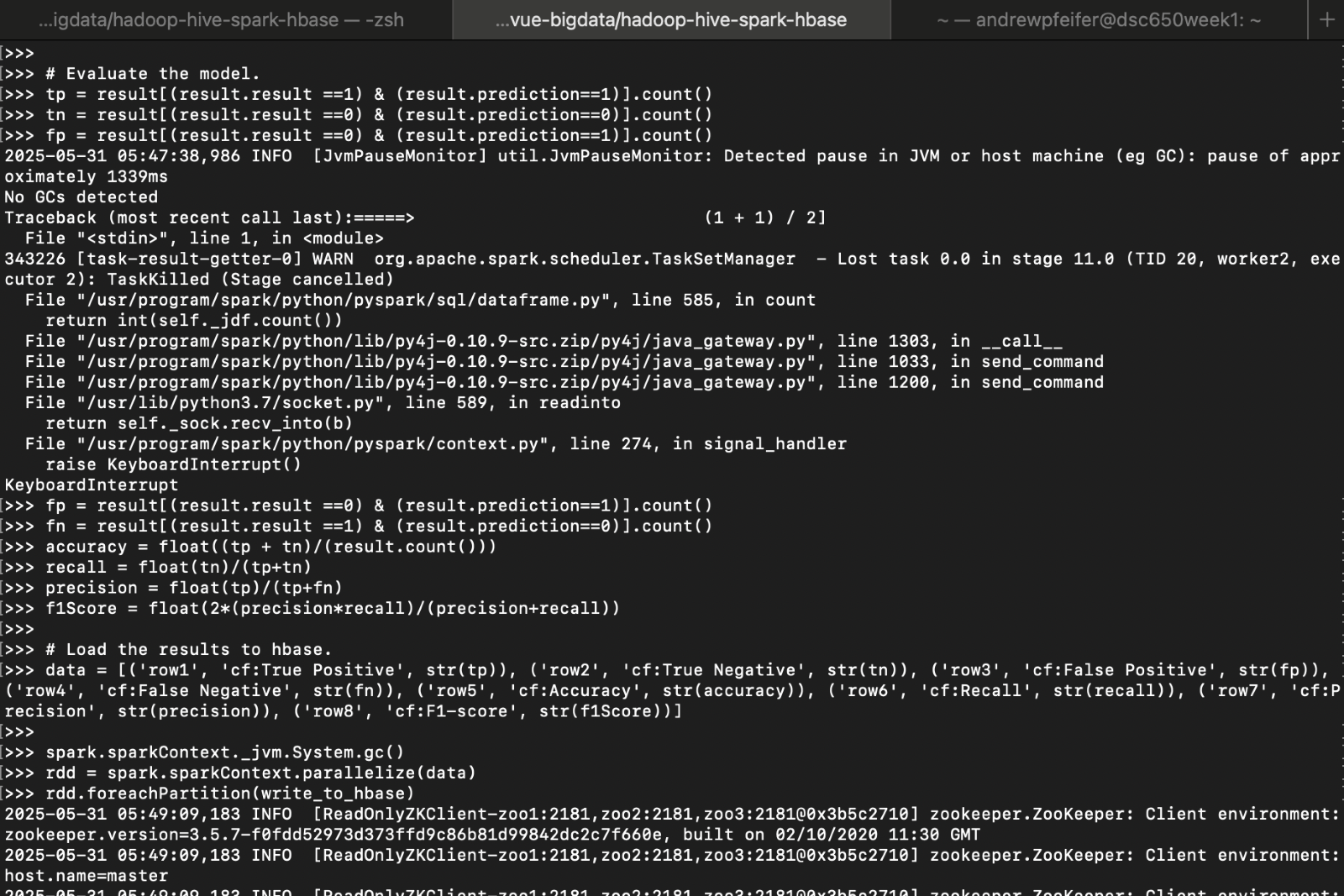
Prior to starting the machine learning, an HBase table was created called hattack\_table. This was created prior to the machine learning algorithm so that there is a place to store the metrics once done.



*Python ML code*

The first step in this process is to enter the pyspark environment. Once in, you will want to load in the packages you intend on using and create a spark session. Then the data can be imported from Hive. When this is done, there are null values entered into the top row of the data frame which need to be removed. When using logistic regression, the algorithm does not work with strings. Any strings would need to be dropped or converted to integers. The vector assembler is used to make vectors of the input columns and called features when completed. A new data frame can be created using the features and the results of the algorithm. The model data is then split into training and testing sets, and the machine learning model is trained. Once trained, the training results and testing results from the model can be obtained, and the model can be evaluated. The full code for this is provided below.



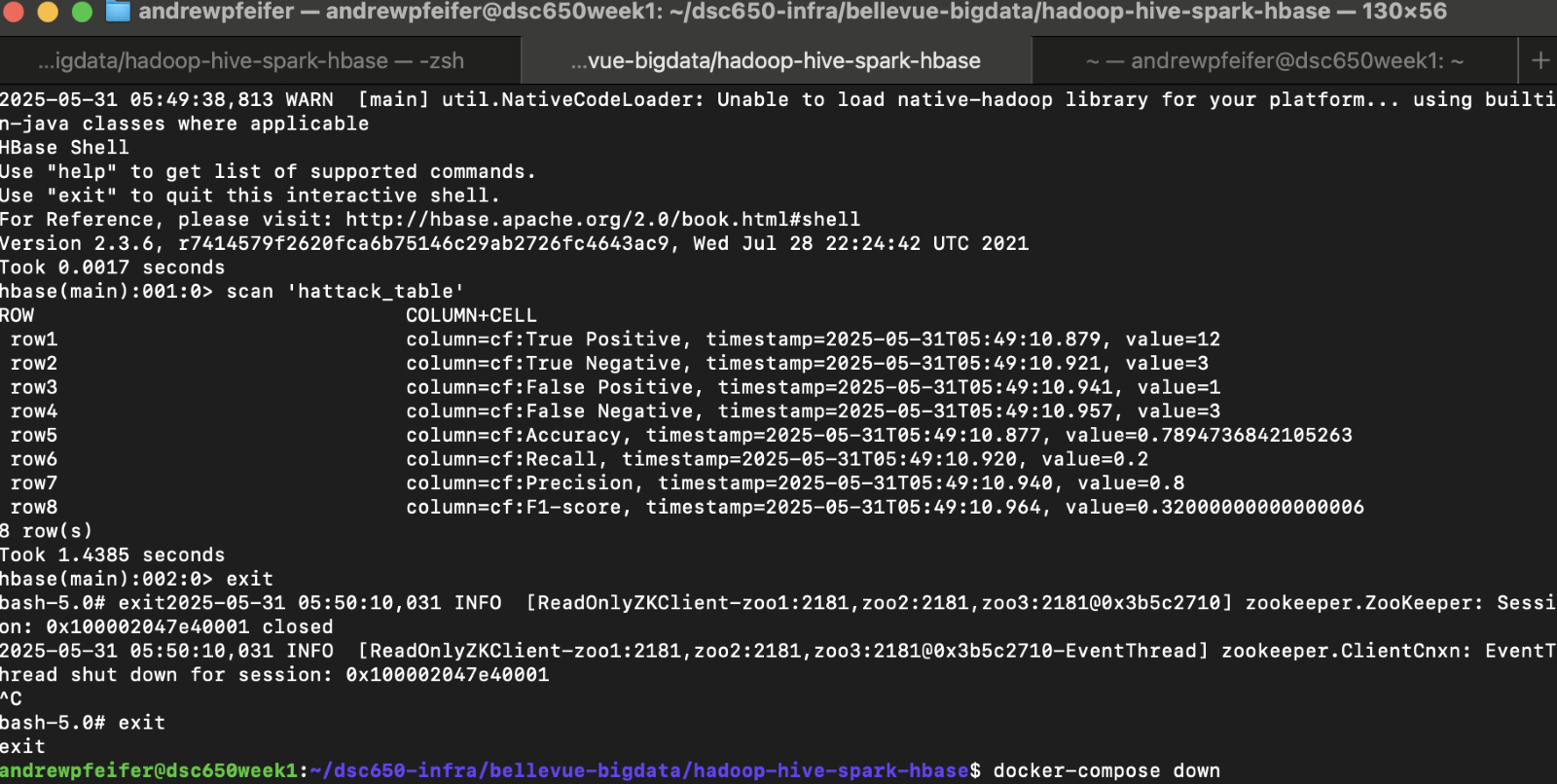


**Writing to HBase from spark:**

Prior to writing to HBase from spark, the command “hbase thrift start &” needs to be entered into the master bash environment. When in the pyspark  session, import happybase, and make a data frame consisting of the row, name of the metric, and the metric value. With the “write to HBase” definition and the RDD, the metrics will flow into the HBase table that was previously created. I received an error in this code initially and needed to update the data to include “cf:” in the column. The code for this can be found in the pyspark code above.

**Retrieve data from HBase:**

The last step in the pipeline process is to retrieve the metrics from HBase. This can be done by entering the HBase shell and running the command scan ‘hattack\_table’.



**What worked and didn’t as well as technical issue:**

Early in the project I had issues with the Hadoop configuration in Nifi. I replaced the whole line a couple of times and got an error. I then reviewed the line in Nifi against the debugging tip and noticed that there was only one place for me to change the username, and this resolved the issue. Again, I had another issue with NiFi. I was letting both processor groups run. This was causing the file to be uploaded to HDFS hundreds of thousands of times. I needed to re-read the debugging tip. I ran the Invoke processor once with the put processor stopped. This allowed me to check the queue and review the data that was being moved. I saw that the entire data set was moved by running one instance of the invoke processor group.

After this I did not run into any issues until the machine learning algorithm portion. First, I forgot to eliminate white space when saving the data to Hive. I needed to go back to Hive and redo the column names. Next, I added duplicate code that did not need to be there, and my containers kept crashing during this process. I cleaned up the duplicated code but was still having issues with the containers crashing when I was trying to make the ML algorithm and upload the results to HBase in the same spark session. We are working in a limited processing power environment, and my thought was that running the thrift server in the background while running logistic regression was causing the containers to break. I then tried running the ML algorithm, then staring at the thrift server, which helped. However, this was not accurate. I then proceeded to manually reduce the iterations, size of the data set, and I converted the string values outside of the spark session to lighten the load. This still did not work, but in the troubleshooting docs I found that I can import a garbage collector, and this solved my issues. I needed to manually run the garbage collector periodically to prevent an HBase disconnect.  The last issue I had was that I did not realize that “cf:” needed to be in the column data that was to be uploaded as well as additional errors of spacing/ spelling in my code. This was a quick fix as I just needed to redefine the data and correct the spacing/spelling issues, then have it uploaded to Hbase.

**Conclusion:**

This project allowed me to understand how HDFS, Hive, NiFi, Spark, and HBase all work together to process data. I was able to ingest the data with NiFi, store it in HDFS, move it to hive, manipulate it in Spark, and store the results in HBase. I did have many errors during this process, but they were mostly user errors with the typos and spacing causing most of the issues. However, I was able to successfully make a continuous data pipeline

As for the Machine Learning model, the results weren’t bad for predicting heart attacks in patients. However, this is a model that I wouldn’t want to deploy in a live environment due to the lack of data that was used to create it. In the future when working in a live environment, I feel confident in my abilities to make a continuous pipeline for batch and real-time data.