CPSC-531 Adv. Database Management Systems

Topic – Big Data Analytics for Insurance Company

Prof. Shen, Tseng-Ching

Team Members –

Name: Vaibhav Rastogi (885190280)

Mail ID - vaibhav1102@csu.fullerton.edu

Name: Sanket Mungikar (885209239)

Mail ID – sanketmungikar@csu.fullerton.edu

Table of Contents –

Introduction –

When beginning with our project for CSPC-531, we thought of the problems that Insurance companies are currently facing to make a stand in the market.Due to increased competition insurance company is facing difficulty expanding its revenue and understanding their customer base. Also another problem faced by insurance company is that to meet their annual targets insurance companies are facing difficulties in attracting customers from different regions.

Objective –

The main goal or objective of the project was to provide analytics solutions for the insurance company which will help them make appropriate business proposals to enhance their revenue by analyzing different age-group customers buying pattern and find the maximum premium captured by the sourcing channel.

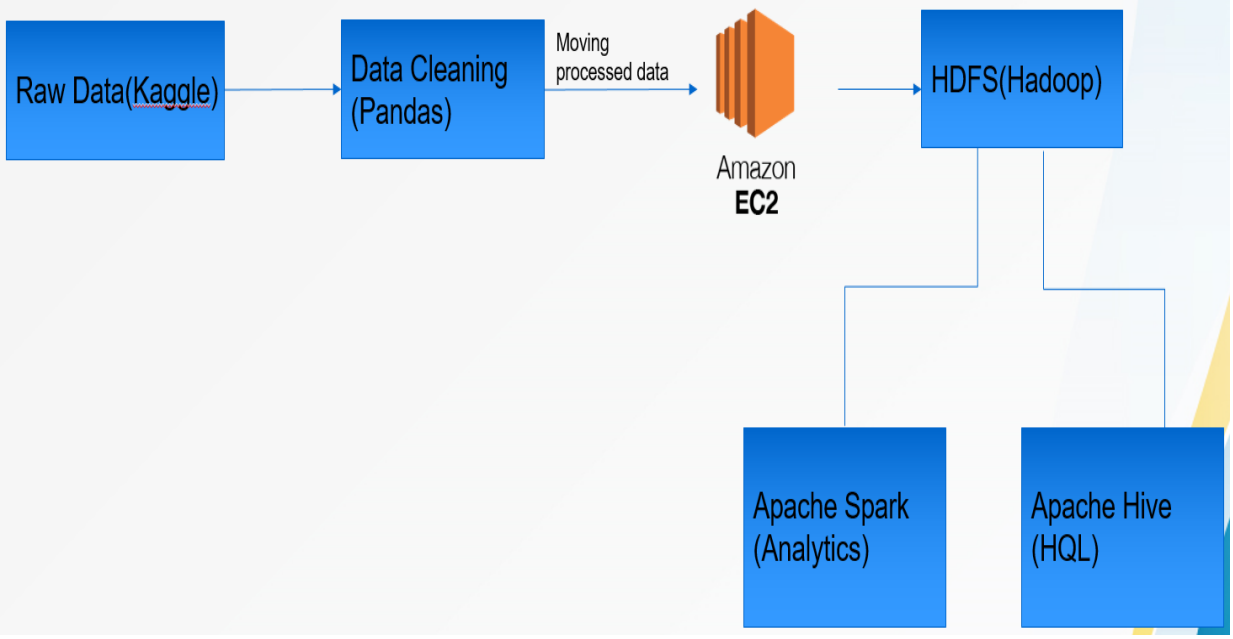
Functionalities –

We performed the big data analytics for Insurance Company based on the following use cases which help the company to analyze and help them in increasing their revenue –

1. Which sourcing channel has generated the Max revenue for the company?
2. Which sourcing channel has got best customers with best possible premium policies (application underwriting score vs. sourcing channel).

Our project will perform the analysis based on this use cases and perform the analytics using Apache Spark.

Architectural Diagram –



Architectural flow of project -

1. **Data Gathering (Raw data) from kaggle-**
   Firstly to begin with our analytics process we need to collect raw data (semi-structured form). So to achieve this requirement we collected/ gathered the data from Kaggle.com.
2. **Data cleaning of raw data using pandas-**
   Secondly the collected raw data was transformed by performing data cleaning operations by using Pandas library where null and undefined values from data where truncated.

3. **Moving data from local machine to hadoop clusters in AWS EC-2 instances –**
   The transformed csv data was then moved from our local machine to determined location in hadoop cluster which was configured in AWS cloud.
4. **Loading data from hdfs AWS EC2 instances into Apache Spark for analytics-**
   The data loaded into the hadoop cluster was then used to perform analytics using Apache Spark. To achieve this the transformed data on the hadoop cluster was used and the spark analytics was performed.

Technologies Used to implement our project –

1. AWS EC2
2. Pandas for data cleaning
3. Apache hadoop
4. Apache Hive
5. Apache Spark- Analytics
6. Stand-alone configuration using Apache Ambari (Horton Works Sandbox)

Configuration required for setting up AWS EC-2 instances –

Steps to set-up EC-2 instances –

IP-Address for every node set-up on AWS -

Name Node ec2-54-189-101-173.us-west-2.compute.amazonaws.com

SNN ec2-18-237-128-174.us-west-2.compute.amazonaws.com

DataNode1 ec2-54-149-20-91.us-west-2.compute.amazonaws.com

DataNode2 ec2-35-165-118-140.us-west-2.compute.amazonaws.com

Connecting to every node created using the .pem file(Identity File) -

1. ssh -i AWSEC2.pem ubuntu@ec2-54-189-101-173.us-west-2.compute.amazonaws.com nn
2. ssh -i AWSEC2.pem ubuntu@ec2-18-237-128-174.us-west-2.compute.amazonaws.com snn
3. ssh -i AWSEC2.pem ubuntu@ec2-54-149-20-91.us-west-2.compute.amazonaws.com d1
4. ssh -i AWSEC2.pem ubuntu@ec2-35-165-118-140.us-west-2.compute.amazonaws.com d2

Configuring Name Node (NN), Secondary Name Node (SNN), Data Nodes (DN-1, DN-2) –

vi ~/.ssh/config

```
Host nnode
    HostName ec2-54-189-101-173.us-west-2.compute.amazonaws.com
    User ubuntu
    IdentityFile ~/.ssh/ida_rsa
Host snn
    HostName ec2-18-237-128-174.us-west-2.compute.amazonaws.com
    User ubunutu
    IdentityFile ~/.ssh/ida_rsa
Host datanode1
    HostName ec2-54-149-20-91.us-west-2.compute.amazonaws.com
    User ubuntu
    IdentityFile ~/.ssh/ida_rsa
Host datanode2
    HostName ec2-35-165-118-140.us-west-2.compute.amazonaws.com
    User ubuntu
    IdentityFile ~/.ssh/ida_rsa
```

Configuring hadoop - hdfs on AWS EC-2:

1. install java sudo apt-get -y install openjdk-8-jdk-headless
2. wget link.tar (downloading hadoop tar file)
3. tar xvzf filename (extracting downloaded tar file)
4. mv extracted filename hadoop (moving extracted folder to hadoop)

Commands to connect data nodes and name node

1. ssh-keygen
2. copy key generated to other nodes from name node
3. scp -i AWSEC2.pem /home/ubuntu/.ssh/id_rsa.pub ubuntu@ec2-18-237-128-174.us-west-2.compute.amazonaws.com:/home/ubuntu/.ssh/id_rsa.pub (snn,d1,d2 from nn)
4. cat ~/.ssh/id_rsa_pub >> ~/.ssh/authorized_keys (all nodes)
5. vi ~/.ssh/config

Exporting it to the PATHS:

Steps to configure hadoop -

1. sudo vi ~/.bashrc
- export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
- export HADOOP_HOME=/home/ubuntu/hadoop
- export HADOOP_CONF=$HADOOP_HOME/conf
- export PATH=$PATH:$JAVA_HOME:$HADOOP_HOME/bin
- source ~/.bashrc

changing user access permission
sudo chown ubuntu:ubuntu /usr/local/hadoop/hdfs/data

Modify hadoop config files located-

HADOOP_HOME/etc/hadoop/

Editing hadoop-env.sh

vi $HADOOP_HOME/etc/hadoop/hadoop-env.sh (SET JAVA_HOME)
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64

Editing core-site.xml

vi $HADOOP_HOME/etc/hadoop/core-site.xml

```
<property>
        <name>fs.defaultFS</name>
        <value>
            hdfs://ec2-54-189-101-173.us west2.compute.amazonaws.com:9000
        </value>
</property>
```

Editing hdfs-site.xml -

vi $HADOOP_HOME/etc/hadoop/hdfs-site.xml (replication factor)

```
<property>
        <name>dfs.replication</name>
        <value>2</value>
    </property>
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>file:///usr/local/hadoop/hdfs/data</value>
```

```
</property>
```

Editing mapred-site.xml

vi $HADOOP_HOME/etc/hadoop/mapred-site.xml

```
<property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
</property>
```

Editing yarn-site.xml

vi $HADOOP_HOME/etc/hadoop/yarn-site.xml

```
<property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapred_shuffle</value>
</property>
<property>
        <name>yarn.resourcemanager.hostname</name>
        <value>ec2-54-189-101-173.us-west-2.compute.amazonaws.com</value>
</property>
```

Copy all these files from Name Node to secondary name node, data node-1, data node – 2:

scp hadoop-env.sh core-site.xml hdfs-site.xml mapred-site.xml yarn-site.xml ubuntu@ec2-18-237-128-174.us-west-2.compute.amazonaws.com:~/hadoop/etc/hadoop

Configuring Master-Slave Nodes:

Name Node -
cd ~/hadoop/etc/hadoop
vi masters – IPV4 address of Name Node.
vi slaves IPV4 address of Data Node1,DataNode2

Secondary Name Node -
cd ~/hadoop/etc/hadoop
vi masters – IPV4 address of Name Node.
vi slaves IPV4 address of Data Node1,DataNode2

Data Node 1 -
cd ~/hadoop/etc/hadoop
vi slaves IPV4 address of Data Node1

Data Node 2 -
cd ~/hadoop/etc/hadoop
vi slaves IPV4 address of Data Node2

After configuring the Name Node, Secondary Name Node and Data Nodes commands to run the services hdfs and yarn:

1. ssh -i AWSEC2.pem ubuntu@ec2-35-88-101-162.us-west-2.compute.amazonaws.com
2. hdfs namenode -format
3. $HADOOP_HOME/sbin/start-dfs.sh
4. $HADOOP_HOME/sbin/start-yarn.sh

Command to stop all the running resources and services-

HADOOP_HOME/sbin/stop-all.sh
Command to check the running resources on hadoop cluster – **Jps**

Steps to Apache Spark on hadoop AWS EC-2 cluster -

Installing python - sudo apt install python3-pip

Command to install jupyter notebook - pip3 install jupyter

Install java - sudo apt-get install default.jre

Install scala - sudo apt-get install scala
Install py4j - pip3 install py4j

Download Spark -
wget http://archive.apache.org/dist/spark/spark-3.0.0/spark-3.0.0-bin-hadoop3.2.tgz

Extract downloaded tar file - sudo tar -zxvf spark-3.0.0-bin-hadoop3.2.tgz

Install findspark - pip3 install findspark

Writing default config to jupyter notebook --generate-config

Create directory for certificates –

1. mkdir certs
2. cd certs
3. writing new private key to mycert.pem

- sudo openssl req -x509 -nodes -days 365 -newkey rsa :1024 -keyout mycert.pem -out mycert.pem

Config jupyter notebook –

1. cd ~/.jupyter/
2. setting permission - ls-lrt
3. vi jupyter_notebook_config.py
   - c=get_config()
   - c.NotebookApp.certfile='u/home/ubuntu/cert/mycert.pem'
   - c.NotebookApp.ip='*'
   - c.NotebookApp.open_browser=False
   - c.NotebookApp.port=8888
4. run notebook - jupyter notebook

Apache Ambari – Stand-Alone clusters –

Steps to set up Apache Ambari –

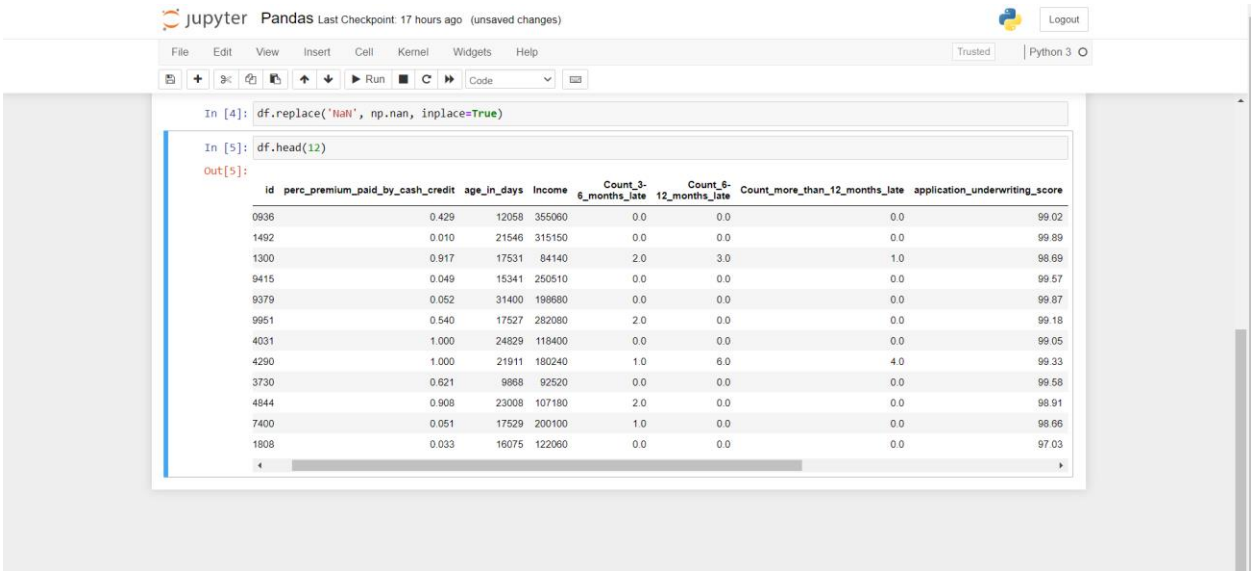
1. Download suitable virtual Box compatible with operating system
2. Download Horton 2.6.5 form hortonworks.com/downloads
3. Import it on your installed VM
4. Start Horton sandbox
5. Login to localhost using username as maria_dev and password as maria_dev

Configuring Spark on Ambari –

Ambari UI was used to set-up the Spark –

https://docs.cloudera.com/HDPDocuments/HDP2/HDP-2.5.3/bk_spark-component-guide/content/install-spark-over-ambari.html

Screenshot for Data Cleaning using Pandas –



jupyter Pandas Last Checkpoint: 17 hours ago (unsaved changes)

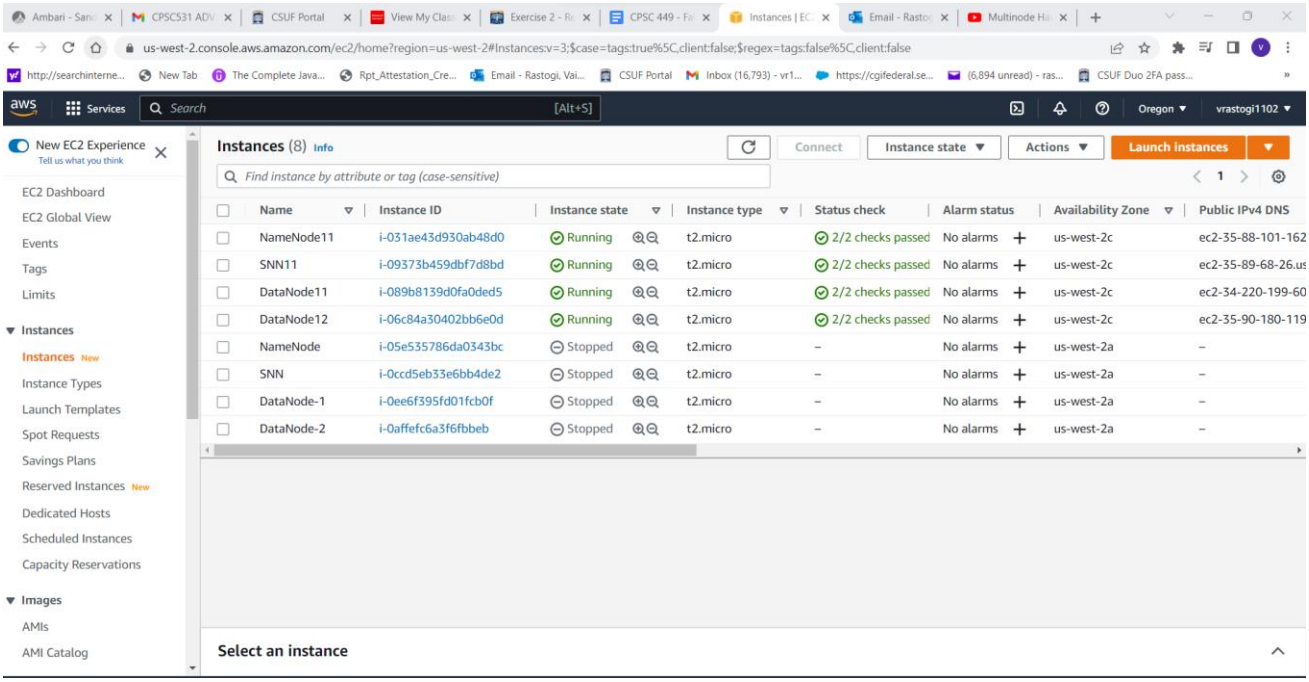
File Edit View Insert Cell Kernel Widgets Help

```
In [4]: df.replace('NaN', np.nan, inplace=True)

In [5]: df.head(12)
```

Out[5]:

id	perc_premium_paid_by_cash_credit	age_in_days	Income	Count_3-6_months_late	Count_6-12_months_late	Count_more_than_12_months_late	application_underwriting_score
0936	0.429	12058	355060	0.0	0.0	0.0	99.02
1492	0.010	21546	315150	0.0	0.0	0.0	99.89
1300	0.917	17531	84140	2.0	3.0	1.0	98.69
9415	0.049	15341	250510	0.0	0.0	0.0	99.57
9379	0.052	31400	198680	0.0	0.0	0.0	99.87
9951	0.540	17527	282080	2.0	0.0	0.0	99.18
4031	1.000	24829	118400	0.0	0.0	0.0	99.05
4290	1.000	21911	180240	1.0	6.0	4.0	99.33
3730	0.621	9868	92520	0.0	0.0	0.0	99.58
4844	0.908	23008	107180	2.0	0.0	0.0	98.91
7400	0.051	17529	200100	1.0	0.0	0.0	98.66
1808	0.033	16075	122060	0.0	0.0	0.0	97.03

Screenshots of running instances on AWS EC-2 cluster –



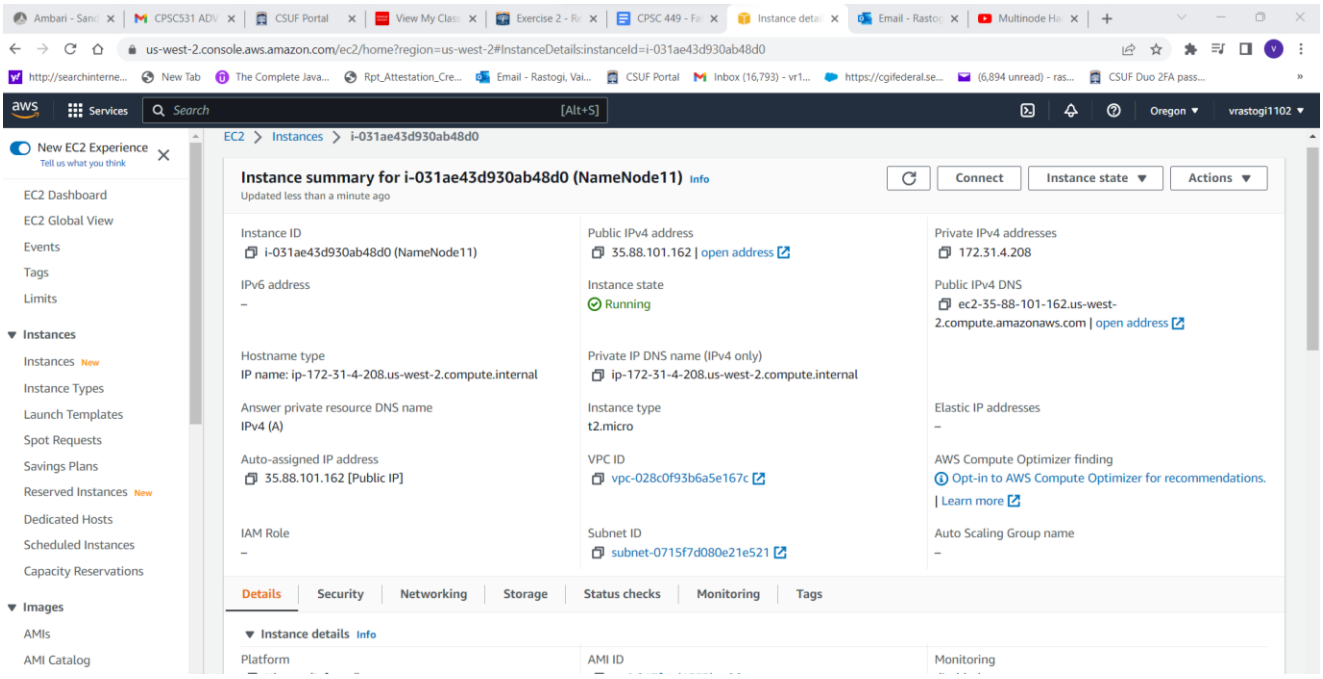
us-west-2.console.aws.amazon.com/ec2/home?region=us-west-2#Instances:v=3;$case=tags:true%5C,client:false;$regex=tags:false%5C,client:false

Instances (8) Info

Connect | Instance state | Actions | Launch instances

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
NameNode11	i-031ae43d930ab48d0	Running	t2.micro	2/2 checks passed	No alarms	us-west-2c	ec2-35-88-101-162
SNN11	i-09373b459dbf7d8bd	Running	t2.micro	2/2 checks passed	No alarms	us-west-2c	ec2-35-89-68-26.us
DataNode11	i-089b8139d0fa0ded5	Running	t2.micro	2/2 checks passed	No alarms	us-west-2c	ec2-34-220-199-60
DataNode12	i-06c84a30402bb6e0d	Running	t2.micro	2/2 checks passed	No alarms	us-west-2c	ec2-35-90-180-119
NameNode	i-05e535786da0343bc	Stopped	t2.micro	–	No alarms	us-west-2a	–
SNN	i-0ccd5eb33e6bb4de2	Stopped	t2.micro	–	No alarms	us-west-2a	–
DataNode-1	i-0ee6f395fd01fcb0f	Stopped	t2.micro	–	No alarms	us-west-2a	–
DataNode-2	i-0affefc6a3f6fbbeb	Stopped	t2.micro	–	No alarms	us-west-2a	–

Select an instance



Apache Spark –

Apache Spark is a data processing framework that can quickly perform processing tasks on very large data sets, and can also distribute data processing tasks across multiple computers, either on its own or in tandem with other distributed computing tools.

Advantages of Spark over Hadoop and Hive –

1) Provides memory based solutions – retain as much memory in RAM.

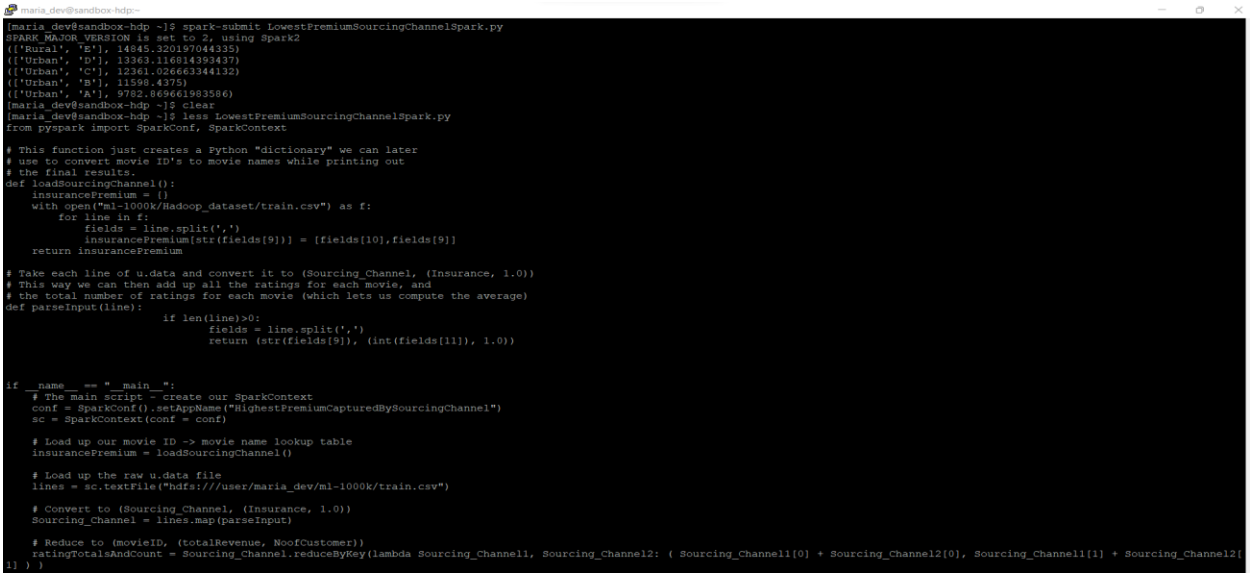
2)11 times faster than Hadoop Map Reduce in memory processing.

3) Use of Directed Acyclic Graph for workflow optimization.

4) Perform analytics on Unstructured Data.

Use Case-1: Which sourcing channel has generated the Max revenue for the company?

Input –



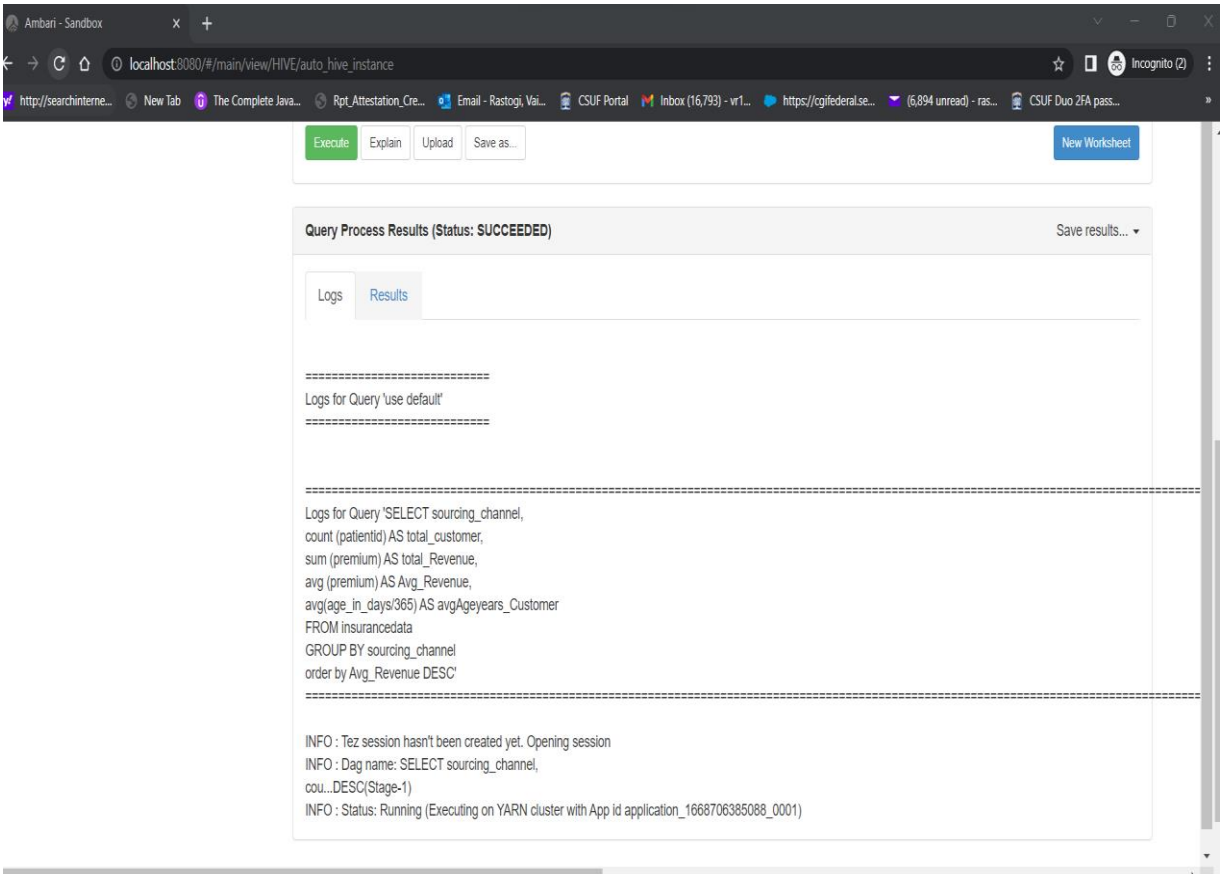
```
[maria_dev@sandbox-hdp ~]$ spark-submit LowestPremiumSourcingChannelSpark.py
SPARK_MAJOR_VERSION is set to 2, using Spark2
(['Rural', 'E'], 14845.320197044335)
(['Urban', 'D'], 13363.116814393437)
(['Urban', 'C'], 12361.026663344132)
(['Urban', 'B'], 11598.4375)
(['Urban', 'A'], 9782.869661983586)
[maria_dev@sandbox-hdp ~]$ clear
[maria_dev@sandbox-hdp ~]$ less LowestPremiumSourcingChannelSpark.py
from pyspark import SparkConf, SparkContext

# This function just creates a Python "dictionary" we can later
# use to convert movie ID's to movie names while printing out
# the final results.
def loadSourcingChannel():
    insurancePremium = {}
    with open("ml-1000k/Hadoop_dataset/train.csv") as f:
        for line in f:
            fields = line.split(',')
            insurancePremium[str(fields[9])] = [fields[10],fields[9]]
    return insurancePremium

# Take each line of u.data and convert it to (Sourcing_Channel, (Insurance, 1.0))
# This way we can then add up all the ratings for each movie, and
# the total number of ratings for each movie (which lets us compute the average)
def parseInput(line):
                    if len(line)>0:
                        fields = line.split(',')
                        return (str(fields[9]), (int(fields[11]), 1.0))



if __name__ == "__main__":
    # The main script - create our SparkContext
    conf = SparkConf().setAppName("HighestPremiumCapturedBySourcingChannel")
    sc = SparkContext(conf = conf)

    # Load up our movie ID -> movie name lookup table
    insurancePremium = loadSourcingChannel()

    # Load up the raw u.data file
    lines = sc.textFile("hdfs:///user/maria_dev/ml-1000k/train.csv")

    # Convert to (Sourcing_Channel, (Insurance, 1.0))
    Sourcing_Channel = lines.map(parseInput)

    # Reduce to (movieID, (totalRevenue, NoofCustomer))
    ratingTotalsAndCount = Sourcing_Channel.reduceByKey(lambda Sourcing_Channel1, Sourcing_Channel2: ( Sourcing_Channel1[0] + Sourcing_Channel2[0], Sourcing_Channel1[1] + Sourcing_Channel2[
1] ) )
```

Use Case-2: Which sourcing channel has got best customers with best possible premium policies (application underwriting score vs. sourcing channel):

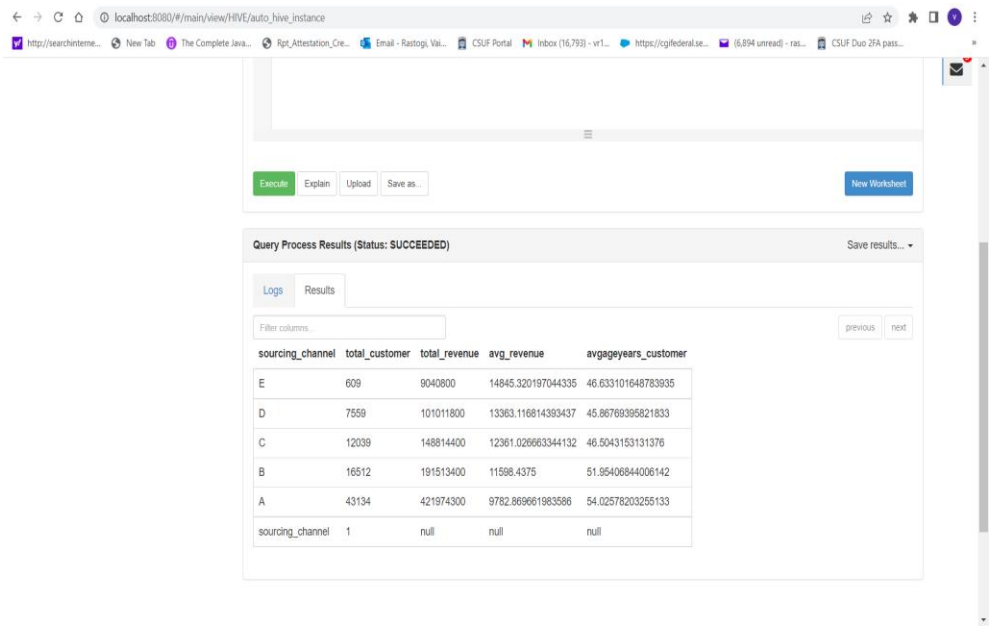
Input -

```
[maria_dev@sandbox-hdp ~]$ less LowestPremiumSourcingChannelSpark.py

[1]+  Stopped                 less LowestPremiumSourcingChannelSpark.py
[maria_dev@sandbox-hdp ~]$ clear
[maria_dev@sandbox-hdp ~]$ spark-submit LowestIncomeid3Spark.py
SPARK_MAJOR_VERSION is set to 2, using Spark2
('E', 99.02827586206897)
('D', 98.99886096044463)
('B', 98.99233298207367)
('A', 98.97689363378979)
('C', 98.94178004817)
[maria_dev@sandbox-hdp ~]$ less  LowestIncomeid3Spark.py
from pyspark import SparkConf, SparkContext

# This function just creates a Python "dictionary" we can later
# use to convert Sourcing Channel to Channel and Region  names while printing out
# the final results.
def loadSourcingChannel():
    insurancePremium = {}
    with open("ml-1000k/Hadoop_dataset/train.csv") as f:
        for line in f:
            fields = line.split(',')
            insurancePremium[str(fields[9])] = [fields[10],fields[9]]
    return insurancePremium

# Take each line of u.data and convert it to (Sourcing_Channel, (Application_underwritingscore, 1.0))

# the total number of ratings for each movie (which lets us compute the average)
def parseInput(line):
                    if len(line)>0:
                        fields = line.split(',')
                        return (str(fields[9]), (float(fields[7]), 1.0))


if __name__ == "__main__":
    # The main script - create our SparkContext
    conf = SparkConf().setAppName("HighestApplicationscoreBySourcingChannel")
    sc = SparkContext(conf = conf)

    # Load up our movie ID -> movie name lookup table
    insurancePremium = loadSourcingChannel()

    # Load up the raw u.data file
    lines = sc.textFile("hdfs:///user/maria_dev/ml-1000k/train.csv")

    # Convert to (Sourcing_Channel, (underwritingscore, 1.0))
    Sourcing_Channel = lines.map(parseInput)
```

Use-Case-1 - Which sourcing channel has generated the Max revenue for the company?

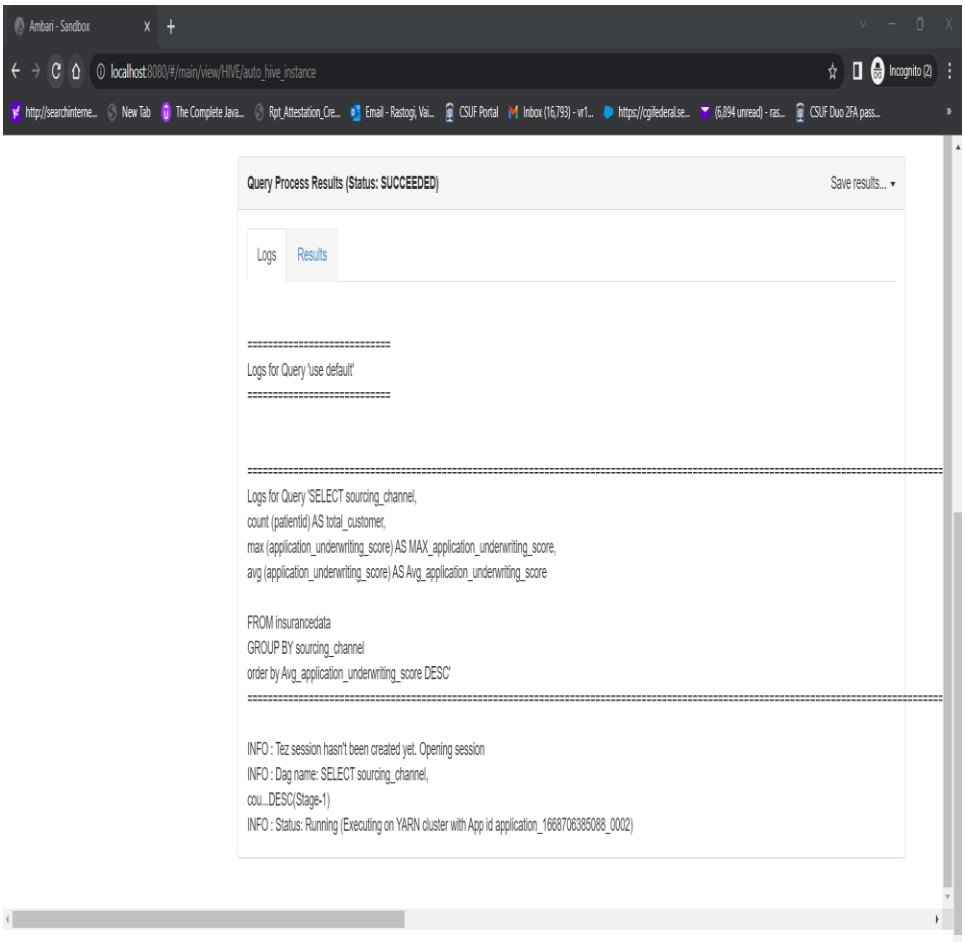


Output –

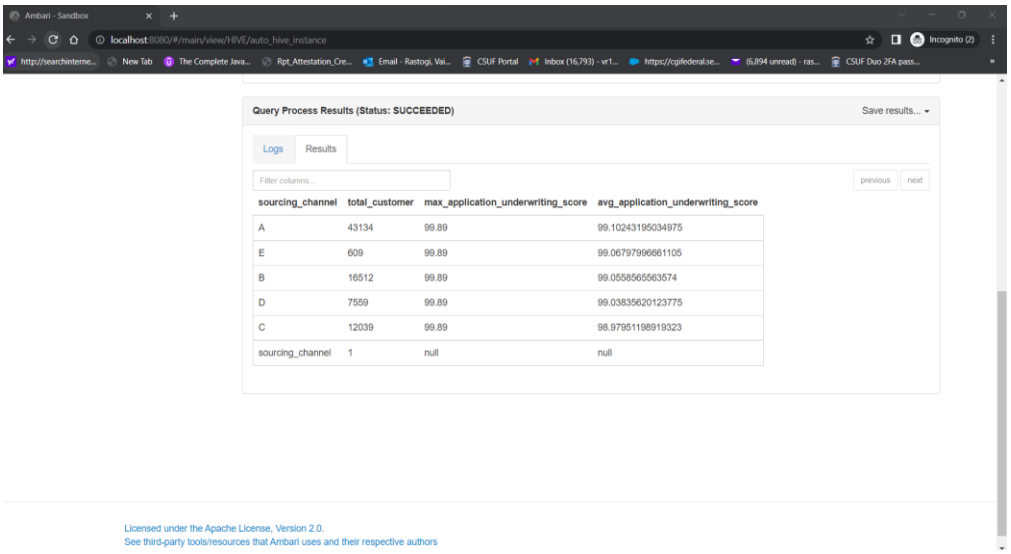


sourcing_channel	total_customer	total_revenue	avg_revenue	avgageyears_customer
E	609	9040800	14845.320197044335	46.633101648783935
D	7559	101011800	13363.116814393437	45.86769395821833
C	12039	148814400	12361.026663344132	46.5043153131376
B	16512	191513400	11598.4375	51.95406844006142
A	43134	421974300	9782.869661983586	54.02578203255133
sourcing_channel	1	null	null	null

Use-Case2: Which sourcing channel has got best customers with best possible premium policies (application underwriting score vs. sourcing channel)



Output –



GitHub Location of Code –

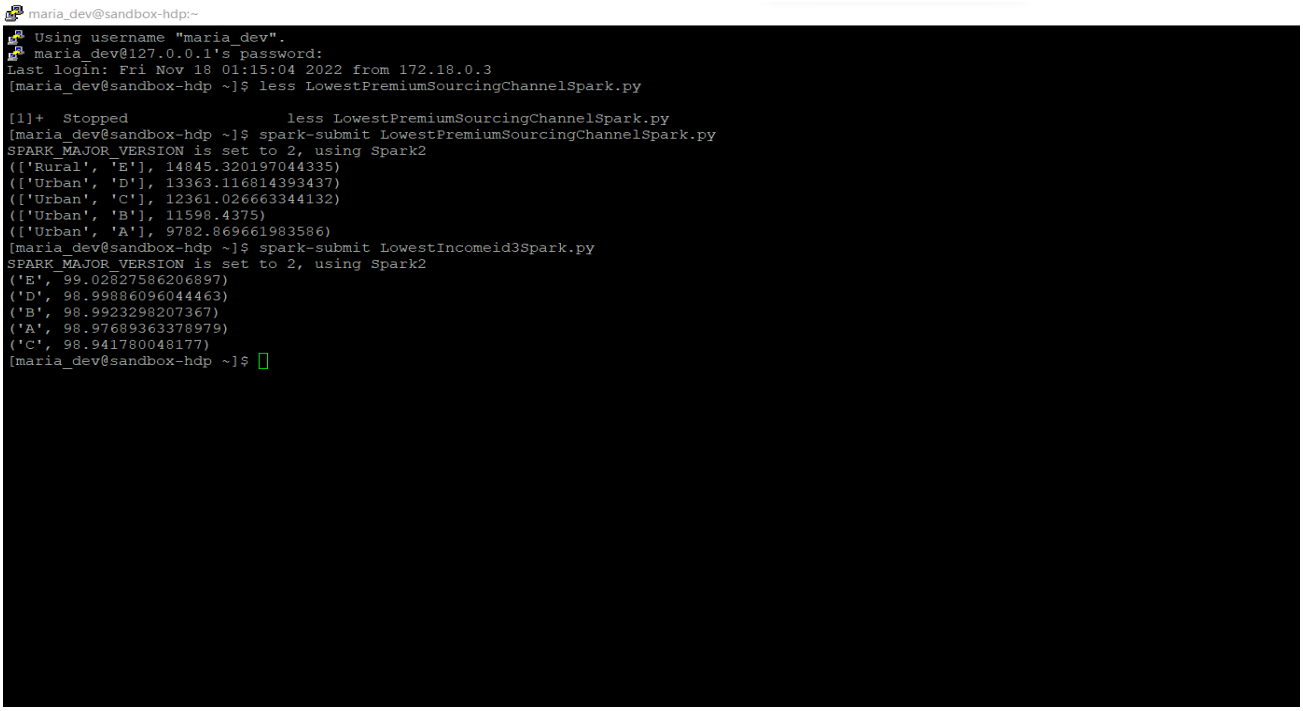
https://github.com/Sanket2596/CPSC-531-Final_Project_BigData_Analytics_For_Insurance_Company

Steps to run the application –

1) Connect to the hadoop configured clusters as mentioned in the above steps.
2) Move the data to hadoop clusters.
3) Import code to jupyter notebook and run **spark-submit command** to run the spark analytics**.**
4) For running Hive queries - run zookeeper and then execute the HQL queries using Apache Ambari (Horton Works Sand Box).

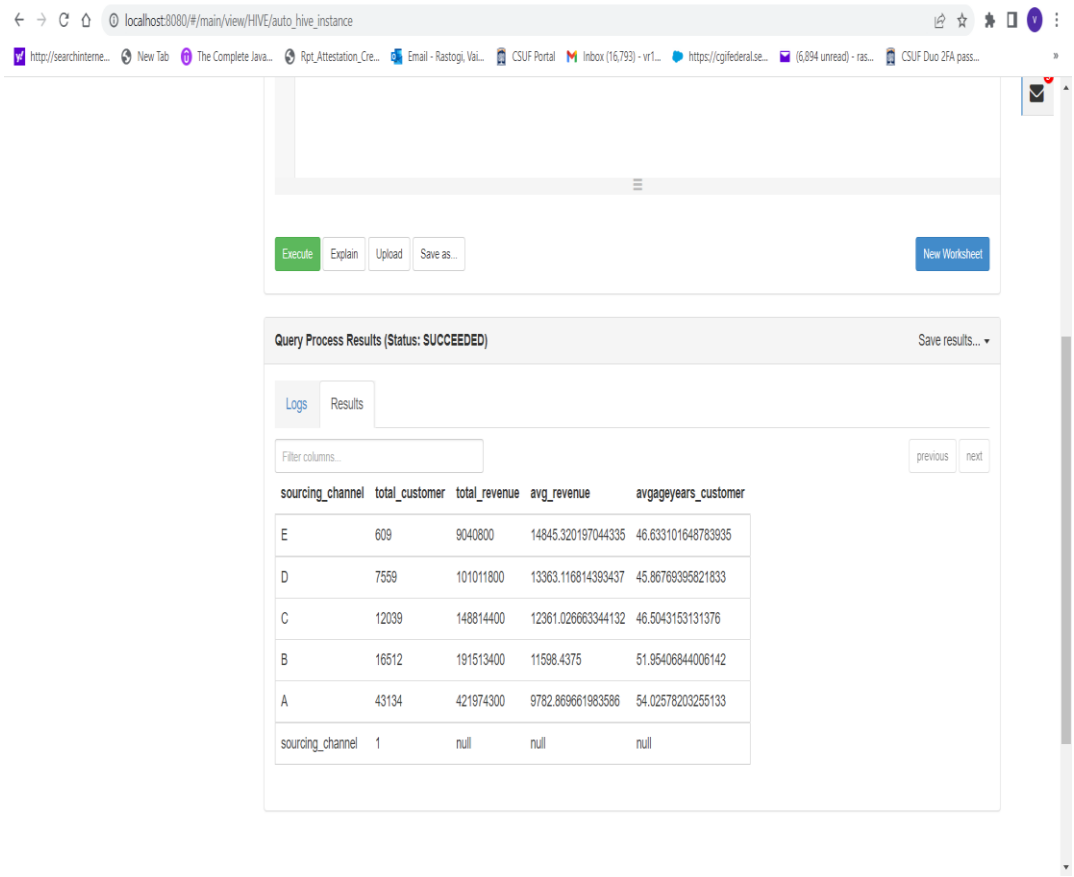
Test Results –

1) Test results for Spark Analytics –



Test results for Hive Analytics –

Result -Use Case -1: Which sourcing channel has generated the Max revenue for the company?



sourcing_channel	total_customer	total_revenue	avg_revenue	avgageyears_customer
E	609	9040800	14845.320197044335	46.633101648783935
D	7559	101011800	13363.116814393437	45.86769395821833
C	12039	148814400	12361.026663344132	46.5043153131376
B	16512	191513400	11598.4375	51.95406844006142
A	43134	421974300	9782.869661983586	54.02578203255133
sourcing_channel	1	null	null	null

Result - Use Case-2: Which sourcing channel has got best customers with best possible premium policies (application underwriting score vs. sourcing channel).