TASK REPORT

PRACTICE CASE 8

BIG DATA TOOLS: INTEGRATING THE DATA TO TACKLE THE BIG DATA PHENOMENON IN WONDERMART

By:

Muhammad Ridho Aulia

Student number

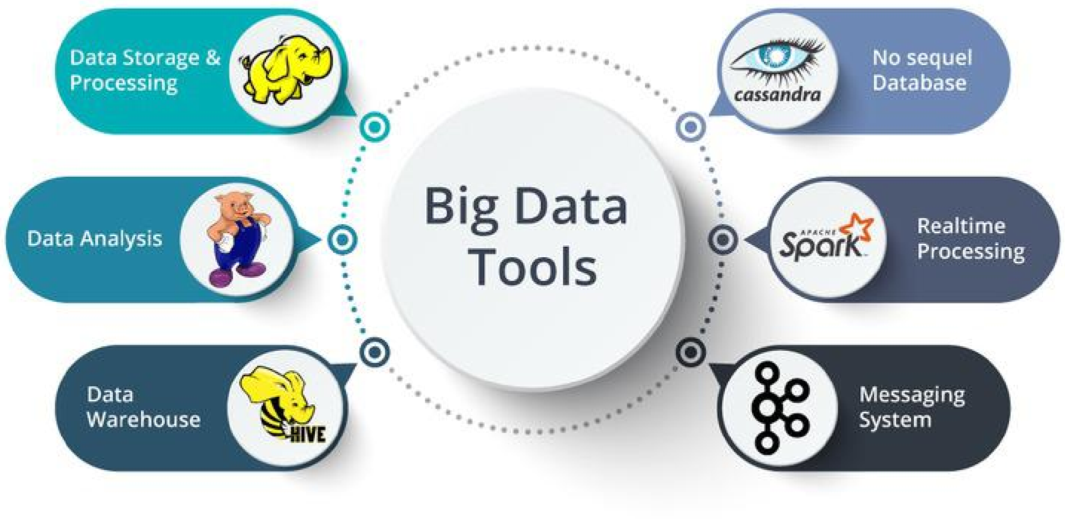
Data Engineer Student

Data Fellowship

2020

Chapter 1

Introduction



You've landed a great job with McJager Consulting (MJC) as a data engineer. MJC is proposing a digitalization data project with WonderMart, the world's largest retail store, to help address their problem of integrating the data to tackle the Big Data phenomenon. Your team of engineers have to create a data pipeline using Sqoop and HiveQL/Impala to store the data.

Your tasks are :

1. Install Cloudera Quickstart VM on your local computer

2. Store the data into HDFS using Cloudera Quickstart VM from your local machine using WinSCP or Tortoise.

3. Using Sqoop to import the data from your VM into HDFS

4. Create a table on HiveQL/Impala to represent the data. All the types of data must be customized based on HiveQL metadata.

5. Using Sqoop incremental method is a plus to inserting the data

6. Describe each step by drawing a flowchart via draw.io

Chapter 2

Progress Report

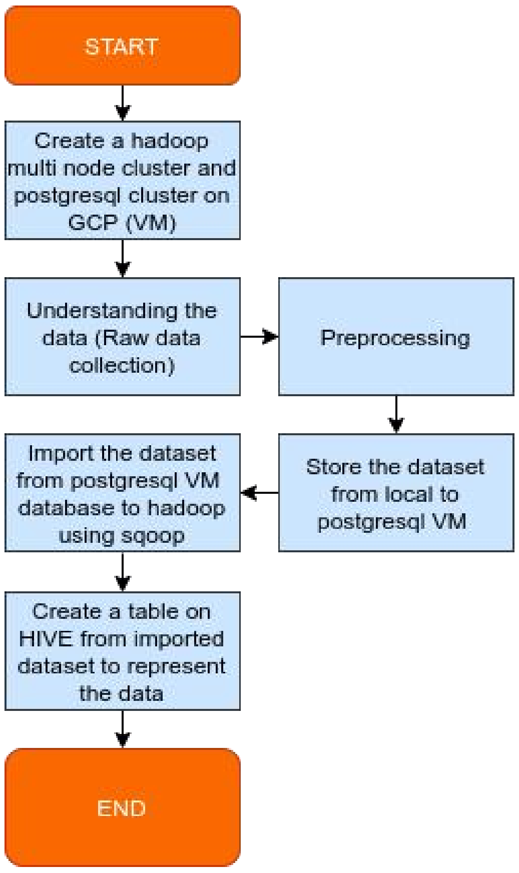
Day/Date	Task	Level	Comments
12/05/2020	Create a multinode cluster VM in GCP	Easy	1 for NameNode, 2 for DataNode
14/05/2020	Create PostgreSQL Database VM in GCP	Easy	Choose the version you need in marketplace
14/05/2020	Understanding the data (customer churn dataset)	Easy	Checking the file and structure (There is a duplicate in Id)
18/05/2020	Import the dataset to PostgreSQL database	Medium	Using python script to build the engine and connect it to postgresql
18/05/2020	Import dataset from PostgreSQL to HDFS	Medium	Using Sqoop Incremental
19/05/2020	Create table in HIVE from the data in HDFS Hard	Medium	Adjust the SQL query with HIVE query
20/05/2020	Create a machine learning model	Hard	Using random forest and xgboost
20/05/2020	Create the flowchart	Medium	Via draw.io
20/05/2020	Write the report	Medium	Using WPS

Chapter 3

Task Report

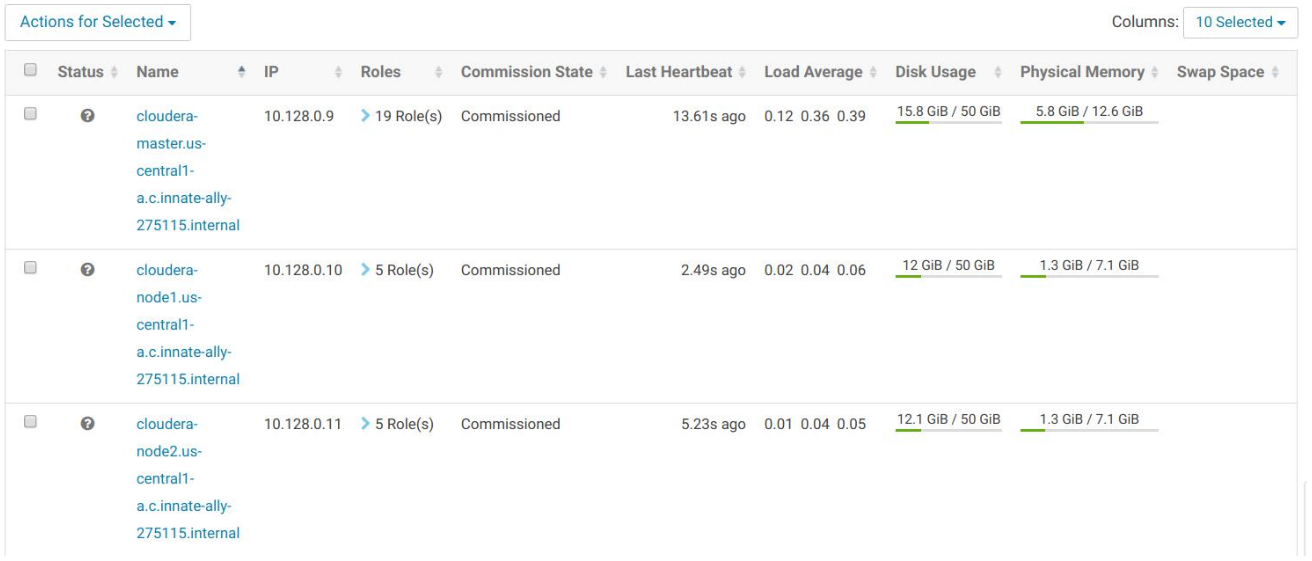
In this project, first, we must understanding the data, we must view the field and column and the purpose of each. Then if the data needs to be cleaned, we did**. In this case because the Id column is set into primary key we must ensure that the id column does not contain duplicate contents. For null contents we will handle it with SQOOP.**

Next, we create a hadoop multinode cluster and postgresql virtual machine on Google Cloud Platform. Then we store the dataset from local to postgresql with some python queries. Next we import the dataset from PostgreSQL VM database to hadoop using sqoop incremental and last we create the table on HIVE from imported dataset to represent the data. All step described in the following flowchart:


START
Create a hadoop multi node cluster and postgresql cluster on GCP (VM)
Understanding the data (Raw data collection)
Preprocessing
Store the dataset from local to postgresql VM
Import the dataset from postgresql VM database to hadoop using sqoop
Create a table on HIVE from imported dataset to represent the data
END

There is the explanation from the workflow diagram above:

1. Create multinode cluster and postgresql VM on Google Cloud Platform

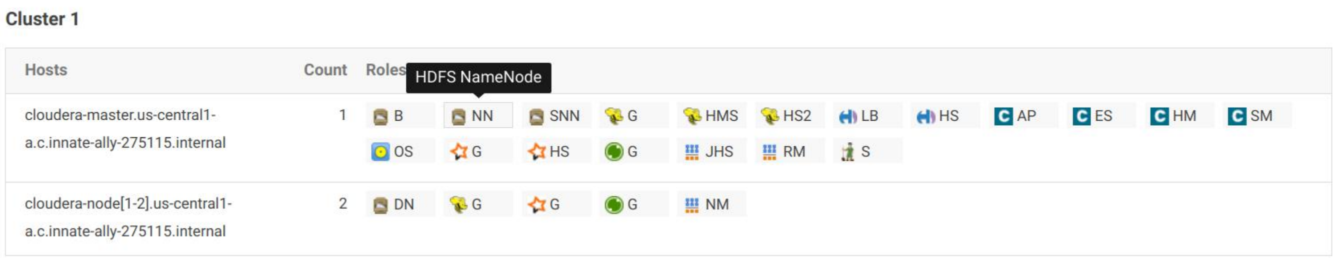

Actions for Selected

Columns: 10 Selected

	Status	Name	IP	Roles	Commission State	Last Heartbeat	Load Average	Disk Usage	Physical Memory	Swap Space
		cloudera-master.us-central1-a.c.innate-ally-275115.internal	10.128.0.9	19 Role(s)	Commissioned	13.61s ago	0.12 0.36 0.39	15.8 GiB / 50 GiB	5.8 GiB / 12.6 GiB	
		cloudera-node1.us-central1-a.c.innate-ally-275115.internal	10.128.0.10	5 Role(s)	Commissioned	2.49s ago	0.02 0.04 0.06	12 GiB / 50 GiB	1.3 GiB / 7.1 GiB	
		cloudera-node2.us-central1-a.c.innate-ally-275115.internal	10.128.0.11	5 Role(s)	Commissioned	5.23s ago	0.01 0.04 0.05	12.1 GiB / 50 GiB	1.3 GiB / 7.1 GiB	

In Google Cloud Platform, we built 4 Virtual Machines (VM), one is for MasterNode/NameNode, two is for DataNode and last one is for PostgreSQL Database cloud VM. We make 3 nodes because we run a parallel processing for Hadoop on Cloudera Manager.

Next we install Cloudera Manager on a NameNode/MasterNode (you can check it via Cloudera Manager in Roles tabs).

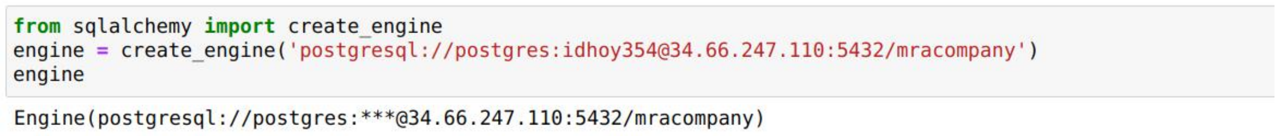

Cluster 1

HDFS NameNode

Hosts	Count	Roles
cloudera-master.us-central1-a.c.innate-ally-275115.internal	1	B, NN, SNN, G, HMS, HS2, LB, HS, AP, ES, HM, SM, OS, G, HS, G, JHS, RM, S
cloudera-node[1-2].us-central1-a.c.innate-ally-275115.internal	2	DN, G, G, G, NM

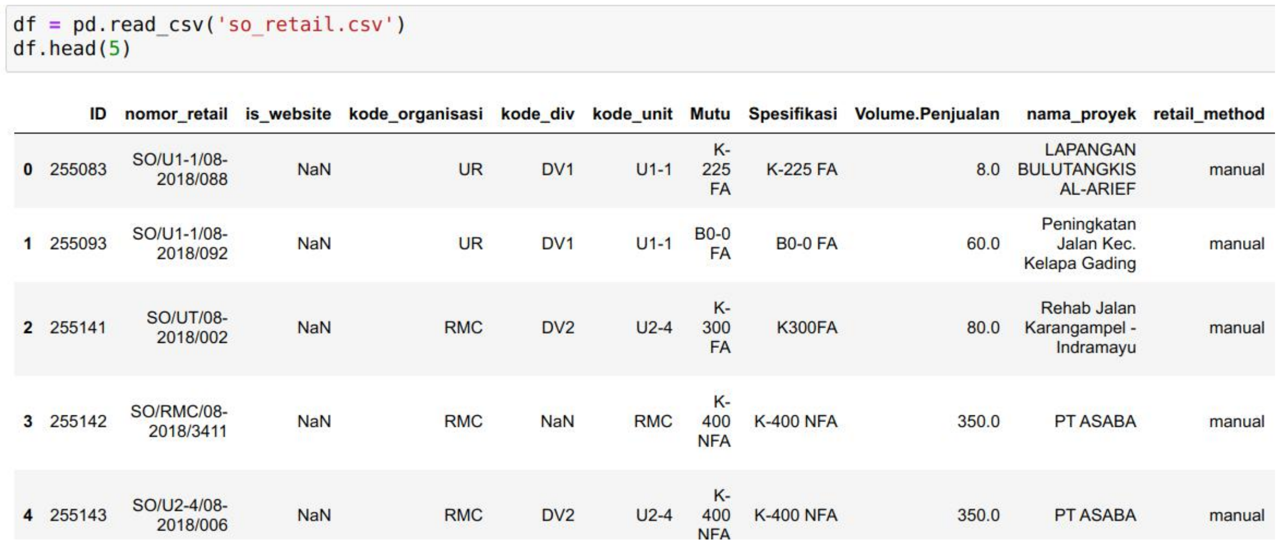
2. Understanding the data and data preprocessing

In this project the dataset is build in csv format. First we import the dataset (csv) into a local PostgreSQL table using ORM SQLalchemy in python. Then we create the engine for the connection to PostgreSQL and read the dataset using pandas.



```
from sqlalchemy import create_engine
engine = create_engine('postgresql://postgres:idhoy354@34.66.247.110:5432/mracompany')
engine
```

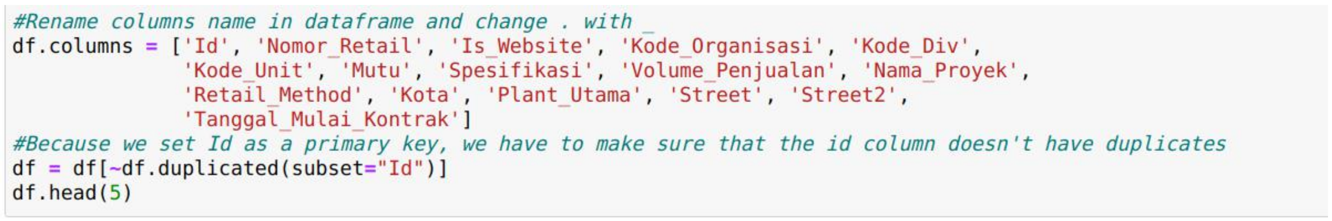
```
Engine(postgresql://postgres:***@34.66.247.110:5432/mracompany)
```



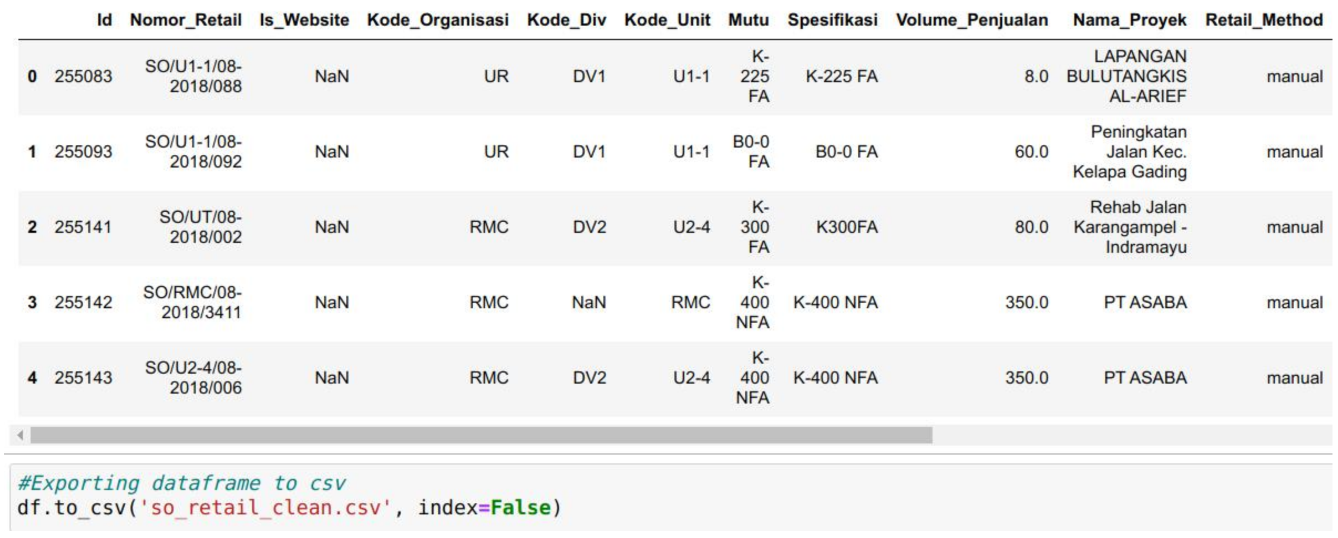
```
df = pd.read_csv('so_retail.csv')
df.head(5)
```

	ID	nomor_retail	is_website	kode_organisasi	kode_div	kode_unit	Mutu	Spesifikasi	Volume.Penjualan	nama_proyek	retail_method
0	255083	SO/U1-1/08-2018/088	NaN	UR	DV1	U1-1	K-225 FA	K-225 FA	8.0	LAPANGAN BULUTANGKIS AL-ARIEF	manual
1	255093	SO/U1-1/08-2018/092	NaN	UR	DV1	U1-1	B0-0 FA	B0-0 FA	60.0	Peningkatan Jalan Kec. Kelapa Gading	manual
2	255141	SO/UT/08-2018/002	NaN	RMC	DV2	U2-4	K-300 FA	K300FA	80.0	Rehab Jalan Karangampel - Indramayu	manual
3	255142	SO/RMC/08-2018/3411	NaN	RMC	NaN	RMC	K-400 NFA	K-400 NFA	350.0	PT ASABA	manual
4	255143	SO/U2-4/08-2018/006	NaN	RMC	DV2	U2-4	K-400 NFA	K-400 NFA	350.0	PT ASABA	manual

In this case because the Id column is set into primary key we must ensure that the Id column does not contain duplicate contents. We will dropped the rows containing duplicates. We recognize that the columns name on dataset contained non-standard naming ex: Volume.Penjualan and Plant.Utama. We change the column names into capitalize each word and replace the period with underscore. And we export the dataframe to csv file back.



```
#Rename columns name in dataframe and change . with _
df.columns = ['Id', 'Nomor_Retail', 'Is_Website', 'Kode_Organisasi', 'Kode_Div',
              'Kode_Unit', 'Mutu', 'Spesifikasi', 'Volume_Penjualan', 'Nama_Proyek',
              'Retail_Method', 'Kota', 'Plant_Utama', 'Street', 'Street2',
              'Tanggal_Mulai_Kontrak']
#Because we set Id as a primary key, we have to make sure that the id column doesn't have duplicates
df = df[~df.duplicated(subset="Id")]
df.head(5)
```



	Id	Nomor_Retail	Is_Website	Kode_Organisasi	Kode_Div	Kode_Unit	Mutu	Spesifikasi	Volume_Penjualan	Nama_Proyek	Retail_Method
0	255083	SO/U1-1/08-2018/088	NaN	UR	DV1	U1-1	K-225 FA	K-225 FA	8.0	LAPANGAN BULUTANGKIS AL-ARIEF	manual
1	255093	SO/U1-1/08-2018/092	NaN	UR	DV1	U1-1	B0-0 FA	B0-0 FA	60.0	Peningkatan Jalan Kec. Kelapa Gading	manual
2	255141	SO/UT/08-2018/002	NaN	RMC	DV2	U2-4	K-300 FA	K300FA	80.0	Rehab Jalan Karangampel - Indramayu	manual
3	255142	SO/RMC/08-2018/3411	NaN	RMC	NaN	RMC	K-400 NFA	K-400 NFA	350.0	PT ASABA	manual
4	255143	SO/U2-4/08-2018/006	NaN	RMC	DV2	U2-4	K-400 NFA	K-400 NFA	350.0	PT ASABA	manual

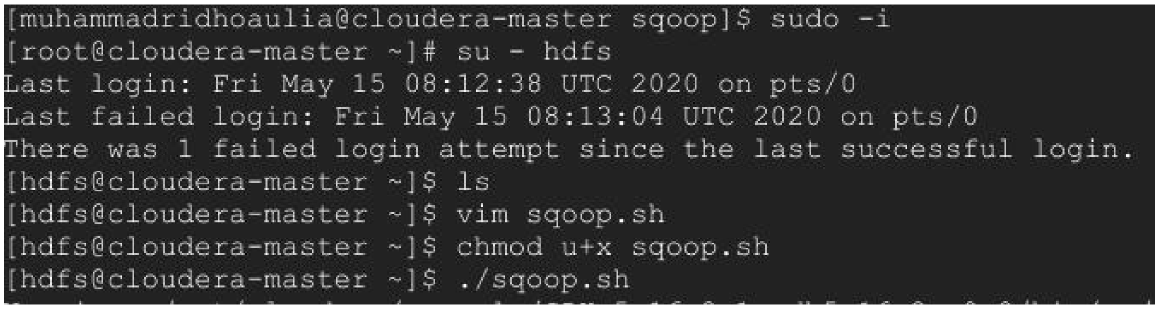
```
#Exporting dataframe to csv
df.to_csv('so_retail_clean.csv', index=False)
```

3. Import the dataset from local PostgreSQL database to HDFS using SCOOP

We build a database using PostgreSQL in this project. Based on the Sqoop book that we read, we decided to use Sqoop Incremental to import the data in PostgreSQL table into HDFS. As we mentioned earlier, we handled the null problems with scoop arguments, we put the antislash and N to transform the null data to make it easier when we do a hive query in the future to see the number of nulls, because if not changed hdfs will represent null as **'null' (string)** .

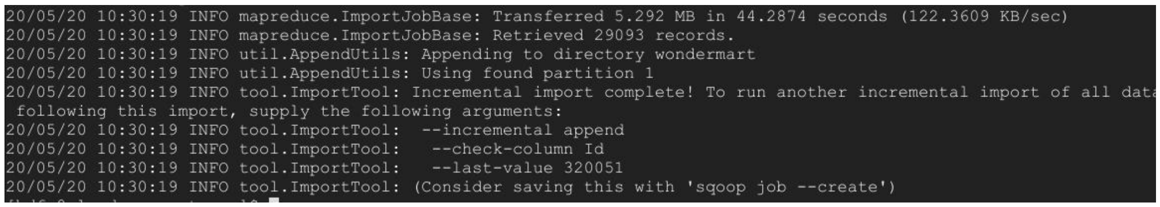
```
sqoop import \
--connect jdbc:postgresql://34.66.247.110/mracompany \
--username postgres \
--password idhoy354 \
--table retail \
--incremental append \
--check-column Id \
--last-value 1 \
--null-string '\\N' \
--null-non-string '\\N' \
--target-dir /user/hdfs/wondermart \
-- --schema public
```

We put the .sh code to MasterNode in GCP VM, change the file permission and run the .sh file:



```
[muhammadridhoaulia@cloudera-master sqoop]$ sudo -i
[root@cloudera-master ~]# su - hdfs
Last login: Fri May 15 08:12:38 UTC 2020 on pts/0
Last failed login: Fri May 15 08:13:04 UTC 2020 on pts/0
There was 1 failed login attempt since the last successful login.
[hdfs@cloudera-master ~]$ ls
[hdfs@cloudera-master ~]$ vim sqoop.sh
[hdfs@cloudera-master ~]$ chmod u+x sqoop.sh
[hdfs@cloudera-master ~]$ ./sqoop.sh
```

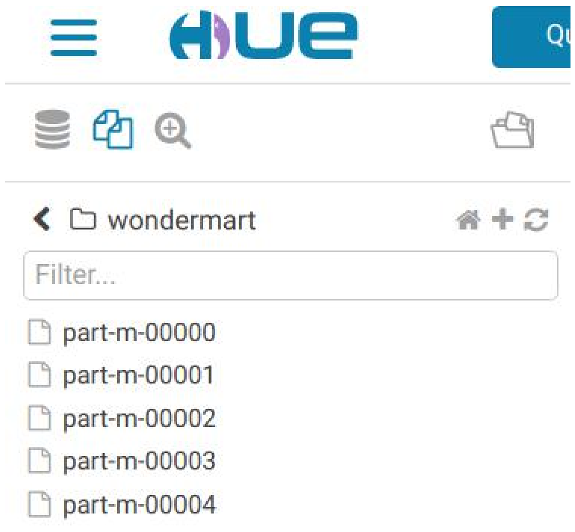
If the process is success map reduce will working and parted yhe file into 4 pieces and this the result when we implement the incremental sqoop, they will checking primary key and last value and mark it to facilitate the next import based on the last value, so it will be faster in importing data in the future



```
20/05/20 10:30:19 INFO mapreduce.ImportJobBase: Transferred 5.292 MB in 44.2874 seconds (122.3609 KB/sec)
20/05/20 10:30:19 INFO mapreduce.ImportJobBase: Retrieved 29093 records.
20/05/20 10:30:19 INFO util.AppendUtils: Appending to directory wondermart
20/05/20 10:30:19 INFO util.AppendUtils: Using found partition 1
20/05/20 10:30:19 INFO tool.ImportTool: Incremental import complete! To run another incremental import of all data
 following this import, supply the following arguments:
20/05/20 10:30:19 INFO tool.ImportTool:  --incremental append
20/05/20 10:30:19 INFO tool.ImportTool:   --check-column Id
20/05/20 10:30:19 INFO tool.ImportTool:   --last-value 320051
20/05/20 10:30:19 INFO tool.ImportTool: (Consider saving this with 'sqoop job --create')
```

You can see in HUE page at (user/hdfs/wondermart), which there is the dataset file exist.

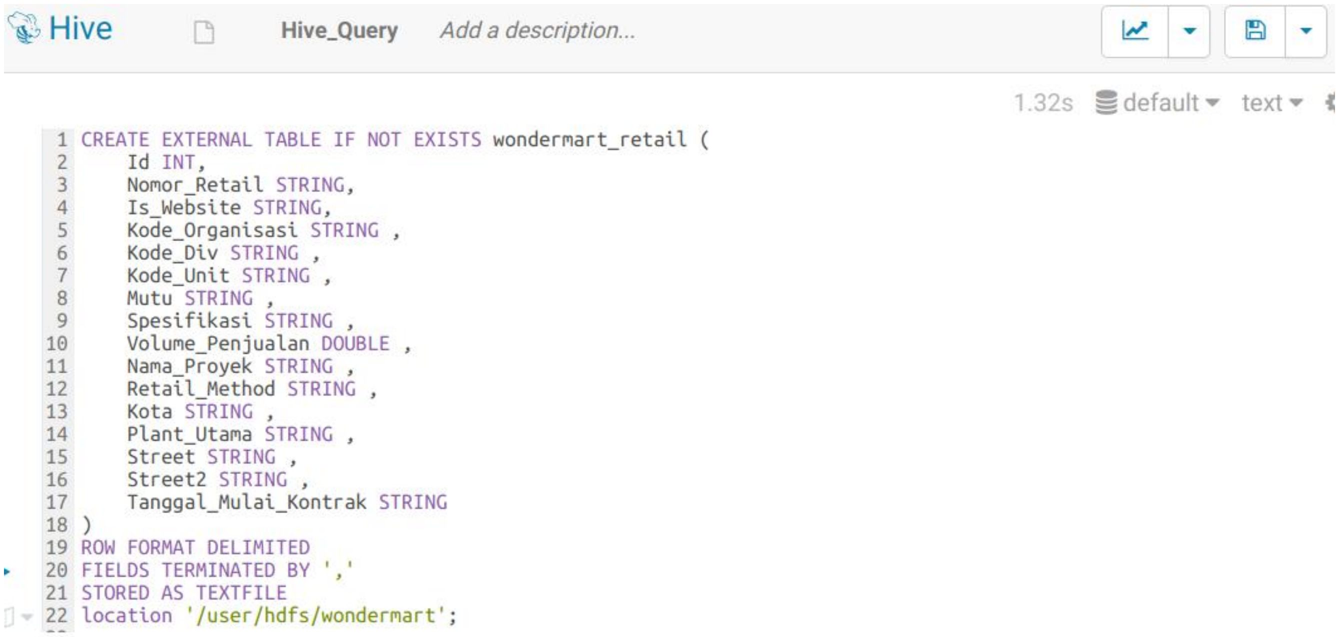
‘



part-m-00000
part-m-00001
part-m-00002
part-m-00003
part-m-00004

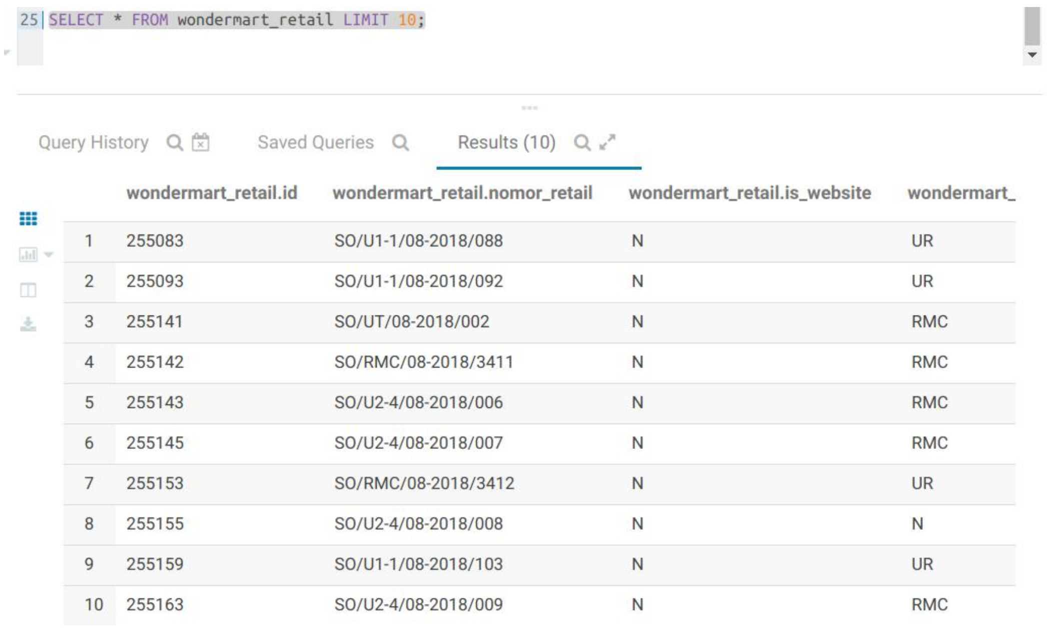
4. Create a table on HIVE to represent the data

The last steps, is we create a table for data that has been successfully imported into HADOOP. We made the table with HIVE on the HUE page in Cloudera Manager. The HIVE syntax is quite different from other RDBMS like PostgreSQL or MySQL. I f you generate DDL from Dbeaver there is some syntax and data type that you must be customized to HIVE syntax. Below is the syntax:



```
CREATE EXTERNAL TABLE IF NOT EXISTS wondermart_retail (
    Id INT,
    Nomor_Retail STRING,
    Is_Website STRING,
    Kode_Organisasi STRING ,
    Kode_Div STRING ,
    Kode_Unit STRING ,
    Mutu STRING ,
    Spesifikasi STRING ,
    Volume_Penjualan DOUBLE ,
    Nama_Proyek STRING ,
    Retail_Method STRING ,
    Kota STRING ,
    Plant_Utama STRING ,
    Street STRING ,
    Street2 STRING ,
    Tanggal_Mulai_Kontrak STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
location '/user/hdfs/wondermart';
```

Next, we try to select the table with HIVE query, there is the result, **you can see the null value handled with N (not empty).**



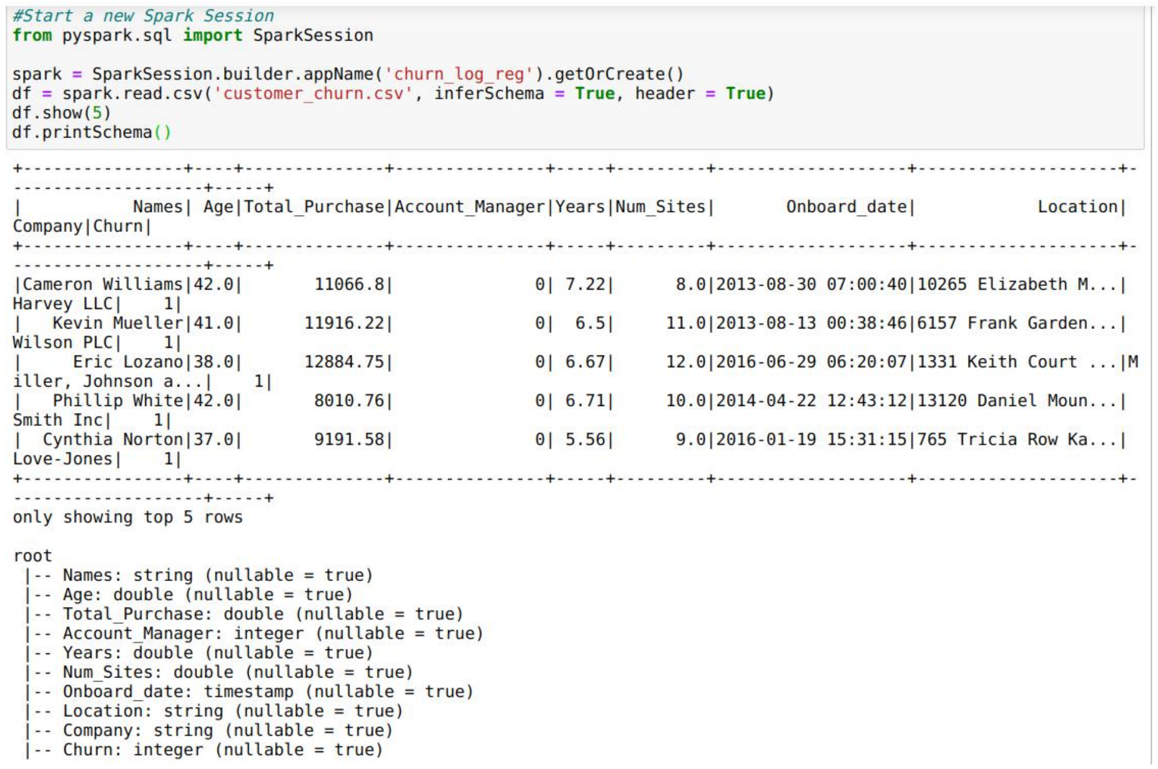
```
25 SELECT * FROM wondermart_retail LIMIT 10;
```

	wondermart_retail.id	wondermart_retail.nomor_retail	wondermart_retail.is_website	wondermart_
1	255083	SO/U1-1/08-2018/088	N	UR
2	255093	SO/U1-1/08-2018/092	N	UR
3	255141	SO/UT/08-2018/002	N	RMC
4	255142	SO/RMC/08-2018/3411	N	RMC
5	255143	SO/U2-4/08-2018/006	N	RMC
6	255145	SO/U2-4/08-2018/007	N	RMC
7	255153	SO/RMC/08-2018/3412	N	UR
8	255155	SO/U2-4/08-2018/008	N	N
9	255159	SO/U1-1/08-2018/103	N	UR
10	255163	SO/U2-4/08-2018/009	N	RMC

5. **a**

machine learning model with PySpark/SparkSQL.

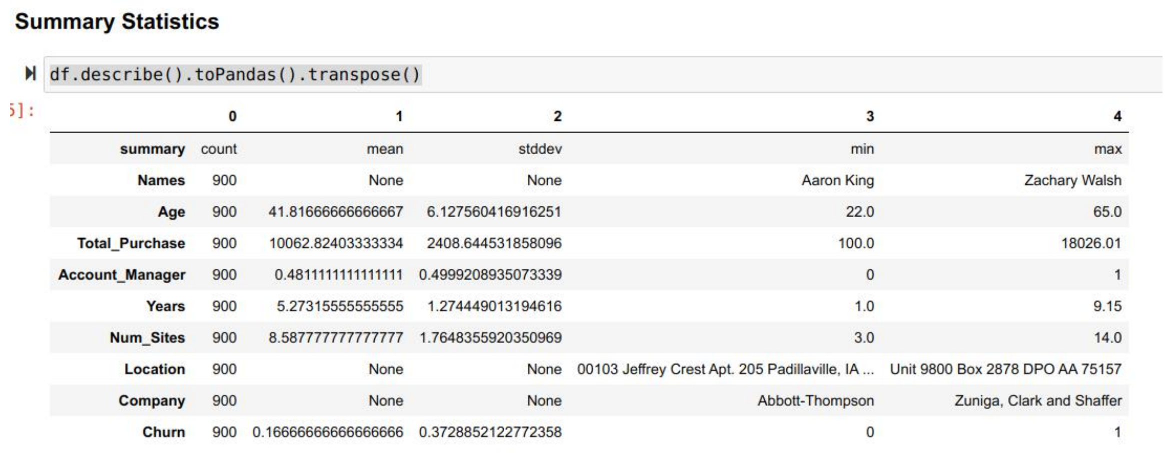
We try to make a machine learning model using Logistic Regression in PySpark. To predict the customer churn or not. First, we install Java Development Kit (JDK 8 or High) and PySpark library. Next we start a new spark session, load the dataset, showing it and print the schema of the dataset.



```
#Start a new Spark Session
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('churn_log_reg').getOrCreate()
df = spark.read.csv('customer_churn.csv', inferSchema = True, header = True)
df.show(5)
df.printSchema()
```

```
+---------------+----+--------------+---------------+-----+---------+-------------------+--------------------+-
-------------------+-----+
|          Names| Age|Total_Purchase|Account_Manager|Years|Num_Sites|       Onboard_date|            Location|
Company|Churn|
+---------------+----+--------------+---------------+-----+---------+-------------------+--------------------+-
-------------------+-----+
|Cameron Williams|42.0|       11066.8|              0| 7.22|      8.0|2013-08-30 07:00:40|10265 Elizabeth M...|
Harvey LLC|    1|
|   Kevin Mueller|41.0|      11916.22|              0|  6.5|     11.0|2013-08-13 00:38:46|6157 Frank Garden...|
Wilson PLC|    1|
|     Eric Lozano|38.0|      12884.75|              0| 6.67|     12.0|2016-06-29 06:20:07|1331 Keith Court ...|M
iller, Johnson a...|    1|
|   Phillip White|42.0|       8010.76|              0| 6.71|     10.0|2014-04-22 12:43:12|13120 Daniel Moun...|
Smith Inc|    1|
|  Cynthia Norton|37.0|       9191.58|              0| 5.56|      9.0|2016-01-19 15:31:15|765 Tricia Row Ka...|
Love-Jones|    1|
+---------------+----+--------------+---------------+-----+---------+-------------------+--------------------+-
-------------------+-----+
only showing top 5 rows

root
 |-- Names: string (nullable = true)
 |-- Age: double (nullable = true)
 |-- Total_Purchase: double (nullable = true)
 |-- Account_Manager: integer (nullable = true)
 |-- Years: double (nullable = true)
 |-- Num_Sites: double (nullable = true)
 |-- Onboard_date: timestamp (nullable = true)
 |-- Location: string (nullable = true)
 |-- Company: string (nullable = true)
 |-- Churn: integer (nullable = true)
```

We can see the summary statistics using describe function:



Summary Statistics

```
df.describe().toPandas().transpose()
```

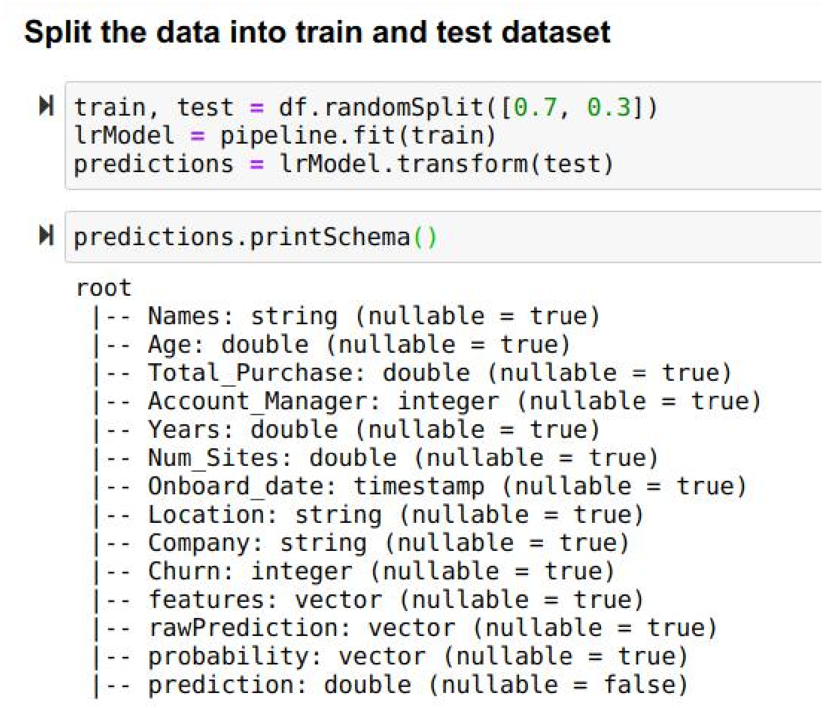
5]:

	0	1	2	3	4
summary	count	mean	stddev	min	max
Names	900	None	None	Aaron King	Zachary Walsh
Age	900	41.81666666666667	6.127560416916251	22.0	65.0
Total_Purchase	900	10062.82403333334	2408.644531858096	100.0	18026.01
Account_Manager	900	0.4811111111111111	0.4999208935073339	0	1
Years	900	5.27315555555555	1.274449013194616	1.0	9.15
Num_Sites	900	8.587777777777777	1.7648355920350969	3.0	14.0
Location	900	None	None	00103 Jeffrey Crest Apt. 205 Padillaville, IA ...	Unit 9800 Box 2878 DPO AA 75157
Company	900	None	None	Abbott-Thompson	Zuniga, Clark and Shaffer
Churn	900	0.16666666666666666	0.3728852122772358	0	1

Next we build the pipeline and use vector assembler to combine the raw features and features generated from various transforms into a single feature vector. In this project we use Logistic Regression to predict customr churn.

```
from pyspark.ml.feature import VectorAssembler

from pyspark.ml import Pipeline

from pyspark.ml.classification import LogisticRegression


assembler = VectorAssembler(inputCols = ['Age', 'Total_Purchase', 'Account_Manager', 'Years', 'Num_Sites',], outputCol = 'features')

log_reg = LogisticRegression(featuresCol = 'features', labelCol = 'Churn', maxIter=10)

pipeline = Pipeline(stages = [assembler, log_reg])
```

And then we split the dataset into train and test and make the predictions using pipeline.



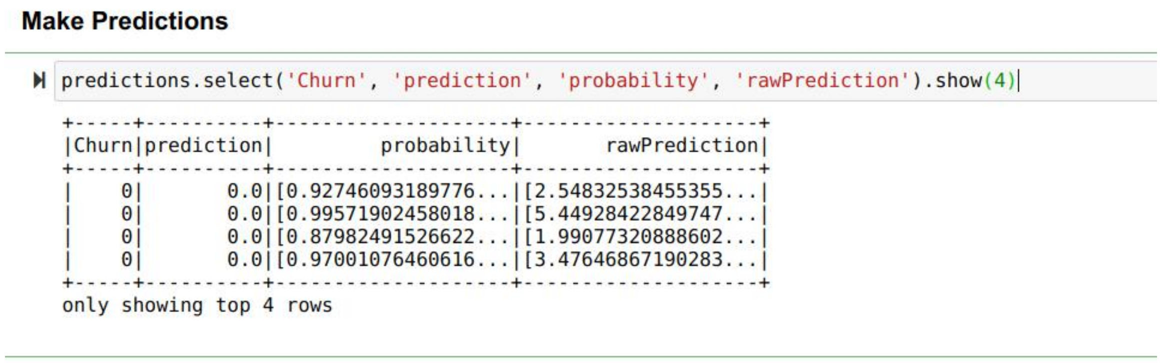
Split the data into train and test dataset

```
train, test = df.randomSplit([0.7, 0.3])
lrModel = pipeline.fit(train)
predictions = lrModel.transform(test)
```

```
predictions.printSchema()
```

```
root
 |-- Names: string (nullable = true)
 |-- Age: double (nullable = true)
 |-- Total_Purchase: double (nullable = true)
 |-- Account_Manager: integer (nullable = true)
 |-- Years: double (nullable = true)
 |-- Num_Sites: double (nullable = true)
 |-- Onboard_date: timestamp (nullable = true)
 |-- Location: string (nullable = true)
 |-- Company: string (nullable = true)
 |-- Churn: integer (nullable = true)
 |-- features: vector (nullable = true)
 |-- rawPrediction: vector (nullable = true)
 |-- probability: vector (nullable = true)
 |-- prediction: double (nullable = false)
```

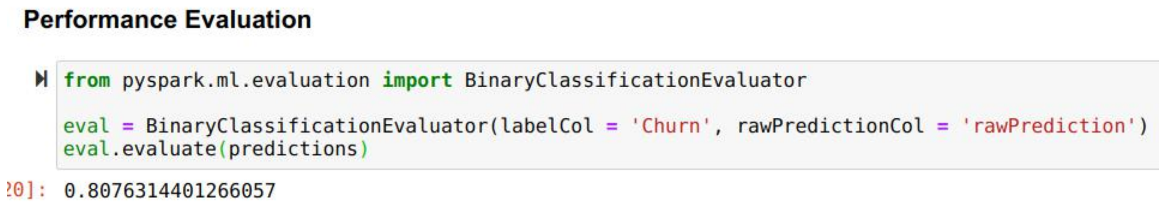
Next, we make a prediction and evaluate it, we reach 0.8, it's good:



Make Predictions

```
predictions.select('Churn', 'prediction', 'probability', 'rawPrediction').show(4)
```

```
+-----+----------+--------------------+--------------------+
|Churn|prediction|         probability|       rawPrediction|
+-----+----------+--------------------+--------------------+
|    0|       0.0|[0.92746093189776...|[2.54832538455355...|
|    0|       0.0|[0.99571902458018...|[5.44928422849747...|
|    0|       0.0|[0.87982491526622...|[1.99077320888602...|
|    0|       0.0|[0.97001076460616...|[3.47646867190283...|
+-----+----------+--------------------+--------------------+
only showing top 4 rows
```

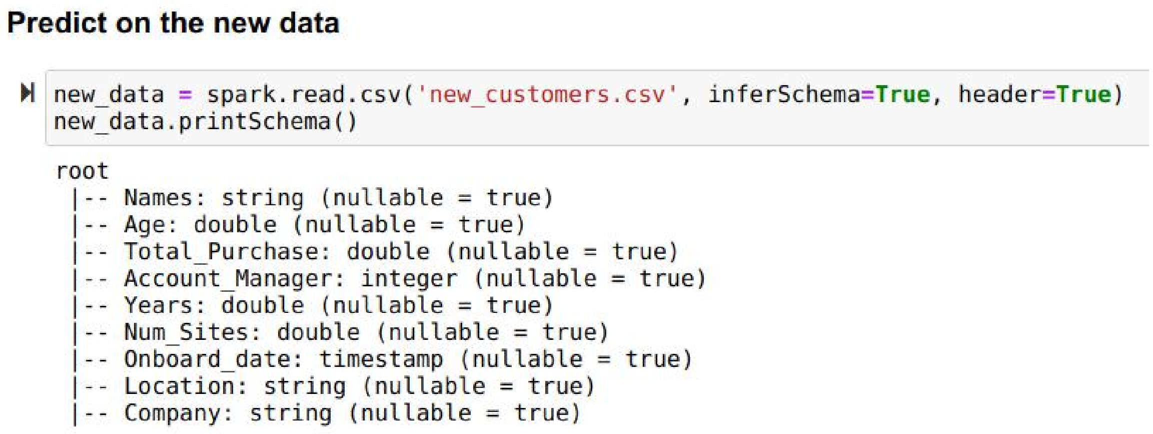


Performance Evaluation

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator

eval = BinaryClassificationEvaluator(labelCol = 'Churn', rawPredictionCol = 'rawPrediction')
eval.evaluate(predictions)
```

```
20]:  0.8076314401266057
```

We try to predict the new data:

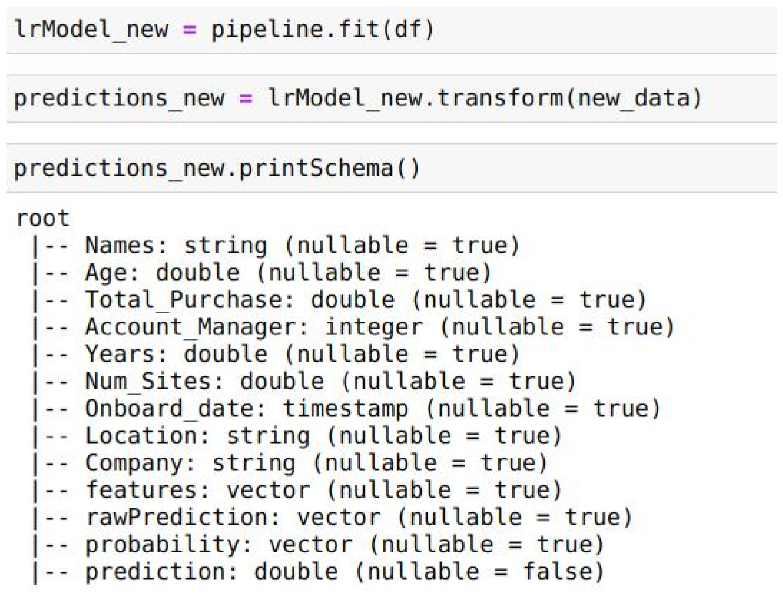


Predict on the new data

```
new_data = spark.read.csv('new_customers.csv', inferSchema=True, header=True)
new_data.printSchema()
```

```
root
 |-- Names: string (nullable = true)
 |-- Age: double (nullable = true)
 |-- Total_Purchase: double (nullable = true)
 |-- Account_Manager: integer (nullable = true)
 |-- Years: double (nullable = true)
 |-- Num_Sites: double (nullable = true)
 |-- Onboard_date: timestamp (nullable = true)
 |-- Location: string (nullable = true)
 |-- Company: string (nullable = true)
```

Use the pipeline that we build earlier and print the predictions schema:



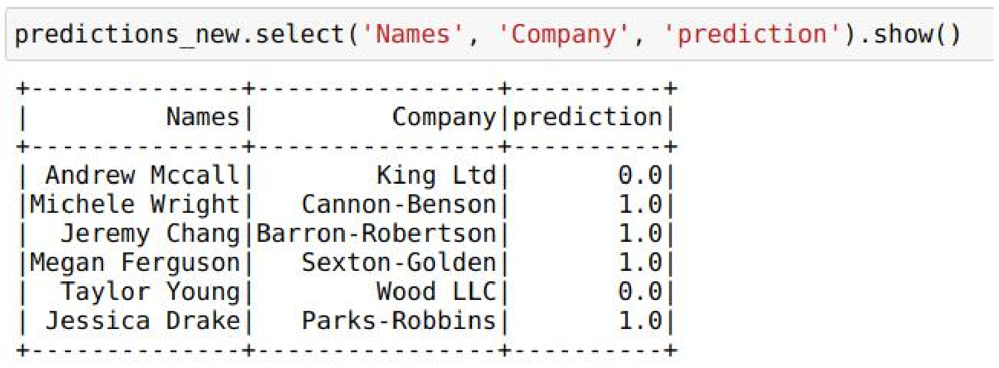
```
lrModel_new = pipeline.fit(df)
```

```
predictions_new = lrModel_new.transform(new_data)
```

```
predictions_new.printSchema()
```

```
root
 |-- Names: string (nullable = true)
 |-- Age: double (nullable = true)
 |-- Total_Purchase: double (nullable = true)
 |-- Account_Manager: integer (nullable = true)
 |-- Years: double (nullable = true)
 |-- Num_Sites: double (nullable = true)
 |-- Onboard_date: timestamp (nullable = true)
 |-- Location: string (nullable = true)
 |-- Company: string (nullable = true)
 |-- features: vector (nullable = true)
 |-- rawPrediction: vector (nullable = true)
 |-- probability: vector (nullable = true)
 |-- prediction: double (nullable = false)
```

And try to predict several rows, the result are shown below:



```
predictions_new.select('Names', 'Company', 'prediction').show()
```

```
+--------------+---------------+----------+
|         Names|        Company|prediction|
+--------------+---------------+----------+
| Andrew Mccall|       King Ltd|       0.0|
|Michele Wright|  Cannon-Benson|       1.0|
|  Jeremy Chang|Barron-Robertson|      1.0|
|Megan Ferguson|   Sexton-Golden|      1.0|
|  Taylor Young|       Wood LLC|       0.0|
| Jessica Drake|   Parks-Robbins|      1.0|
+--------------+---------------+----------+
```