

**Spark (DSL, SQL, Onprem & Cloud) Hackathon based on Interview scenarios**

**Hackathon Components: Spark DF DSL/SQL**

***We got a very good response, outcome and feedback from our students after completing these usecases such as it improved self learning capabilities, interview attending confidence, improved technical vibration, grip in coding, implementation of end to end pipeline etc., Kindly consider all the above factors and start work on it applying time and efforts fully by re trying several times until get the results.***

***This Hackathon helps you to manage the cleansing, scrubbing, curation, cleanup, sanitization, preprocessing, regulation, transformation, ETL/ELT, Performance tuning and schema migration of unpredicted data sets using Spark core, SQL, Dataframe functions which can help you solving interview questions by splitting, merging, applying functions in data sets. It carries lot of important interview questions also..***

**Please read the below points before proceeding to the Hackathon:**

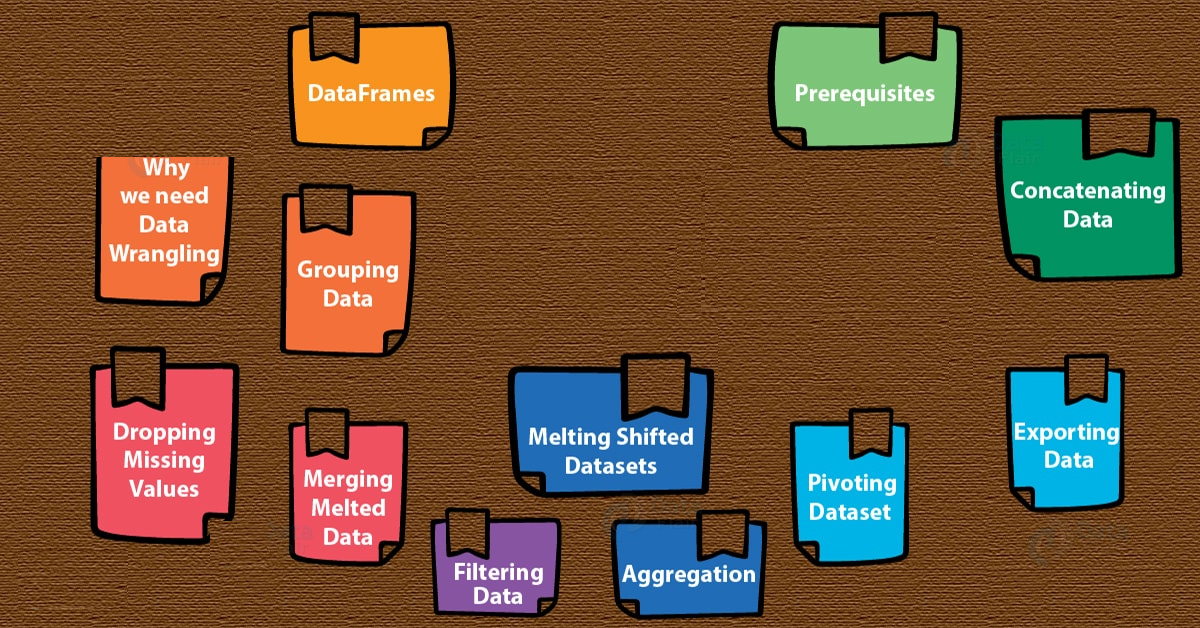
* **Download the data (custs\_states,** **insuranceinfo1,** **insuranceinfo2 ) into HDFS location (/user/hduser/sparkhack2) and start progress.**
* **Import required classes including sparkcontext, sqlcontext and other required objects, classes and import other required libraries later as and when it is needed.**
* **Create Sparkcontext, sqlcontext or spark session.**
* **Provide the final code developed in Pycharm or you can use REPL for snippet development, but finally code should be in Pycharm.**
* **Check where ever performance can be improved and add accordingly.**
* **Follow the hints if you struck anywhere for syntax or similar examples, refer pdfs, programs, online, worst case seek for help from others. If you are struck for long time in some steps ignore those steps or work on other usecases then come back and try.**
* **All the use cases given below are categorized and stated with completion %, try to achieve maximum percentage of 100 by completing all scenarios.**
* **Provide the results as code and snapshot of results.**

**Note: Give more priority to SQL then Databricks/Cloud. Minimum SQL you try to complete and send, then try other usecases if possible.**

**First start with Part A - Topic 1, 2 and 3 (dsl/sql)**

**Next start with Part B - Topic 4 (Databricks or GCP migration)**

**… as per your convenience, try to complete all - it’s a matter of time and continuous efforts.**



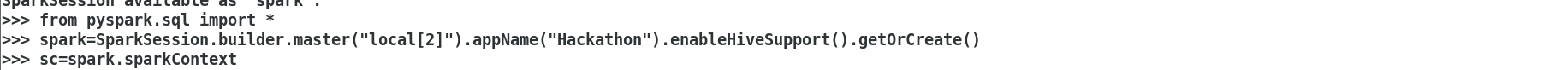
**Part A - Spark DF & SQL (Topic 1, 2 & 3)**

1. **DataFrames operations (55% Completion) – Total 55%**

**Apply Structure, DSL column management functions, transformation, custom udf & schema migration.**

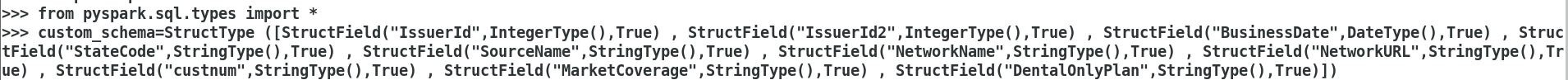
1. Dataframe creation using the built in modules ***(APPLY PERFORMANCE FEATURES LIKE CACHING, PERSISTANCE, COALESCE, REPARTITIONING ETC as per the need)***

Creating the spark session object and spark context object :



**A.** Create first structuretypes for all the columns as per the **insuranceinfo1**.csv with the columns such as IssuerId,IssuerId2,BusinessDate,StateCode,SourceName,NetworkName,NetworkURL,custnum,MarketCoverage,DentalOnlyPlan

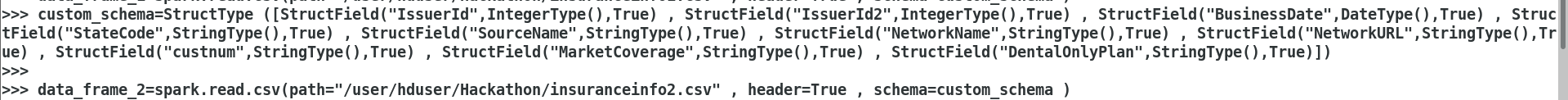
**Hint:** Do it carefully without making typo mistakes. Fields issuerid, issuerid2 should be of IntegerType, ***businessDate should be DateType*** and all other fields are StringType.

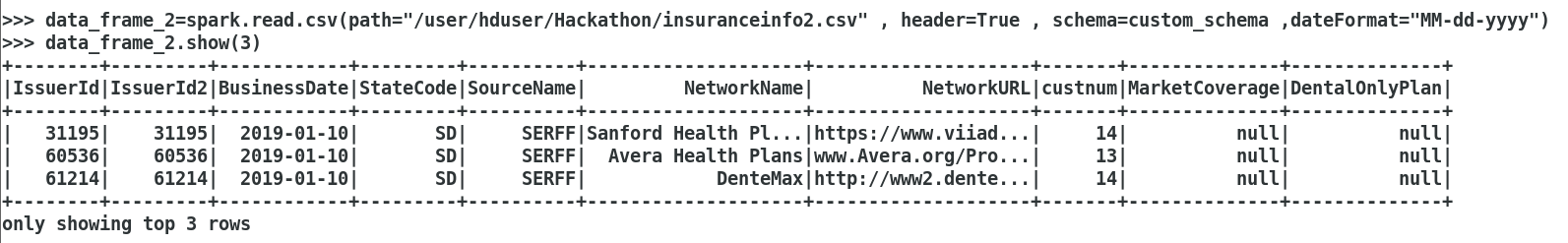




**B.** Create second structuretypes for all the columns as per the **insuranceinfo2**.csv with the columns such as IssuerId,IssuerId2,BusinessDate,StateCode,SourceName,NetworkName,NetworkURL,custnum,MarketCoverage,DentalOnlyPlan

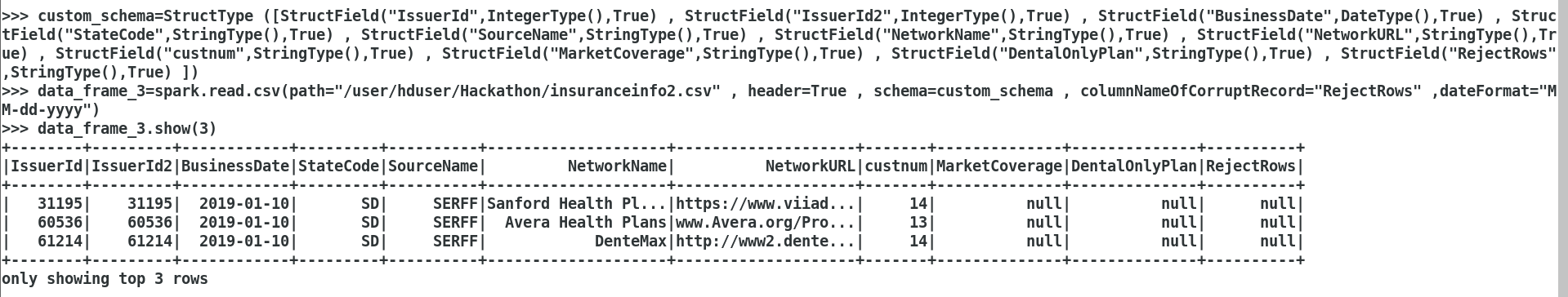
**Hint:** Do it carefully without making typo mistakes. Fields issuerid, issuerid2 should be of IntegerType, ***businessDate should be StringType*** and all other fields are StringType.





**C.** Create third structuretypes for all the columns as per the **insuranceinfo2**.csv with the columns such as IssuerId,IssuerId2,BusinessDate,StateCode,SourceName,NetworkName,NetworkURL,custnum,MarketCoverage,DentalOnlyPlan**,RejectRows**

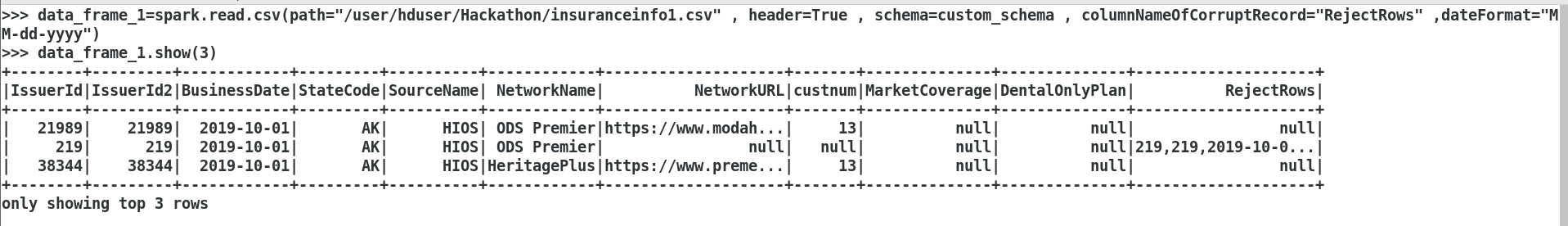
**Hint:** Do it carefully without making typo mistakes. Fields issuerid, issuerid2 should be of IntegerType, ***businessDate should be StringType*** and all other fields are StringType.

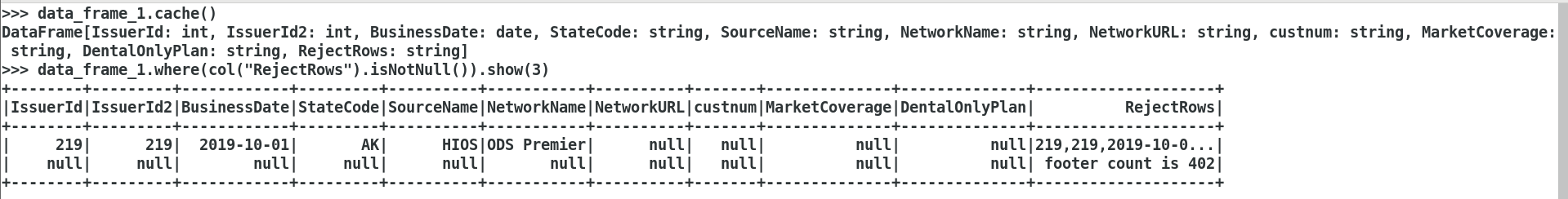


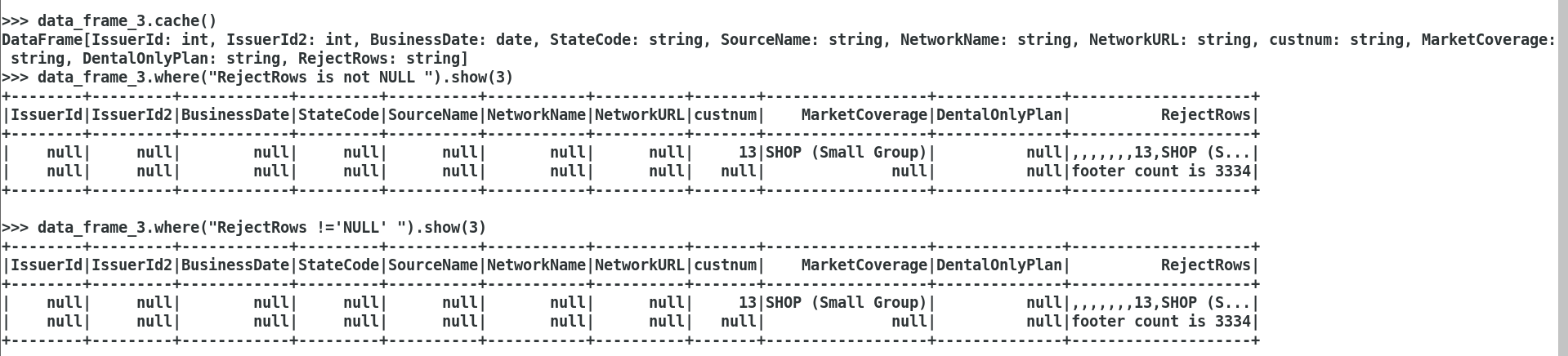
1. Create dataframe using the csv module accessing the insuranceinfo1.csv file and remove the footer from both using header true and remove the footer using dropmalformed options and apply the schema of the structure type created in the step 1.A

Create another dataframe using the csv accessing the insuranceinfo2.csv file and remove the header from both dataframe using header true and remove the footer using dropmalformed options and apply the schema of the structure type created in the step 1.B

**Note:**for insuranceinfo2.csv as the BusinessDate column format is dd-MM-yyyy, we need to explicitly convert it to yyyy-MM-dd format using to\_date() function.







Create another dataframe using the csv accessing the insuranceinfo2.csv file and remove the header from the dataframe using header true, permissive options and apply the schema of the structure type created in the step 1.C with the csv options of

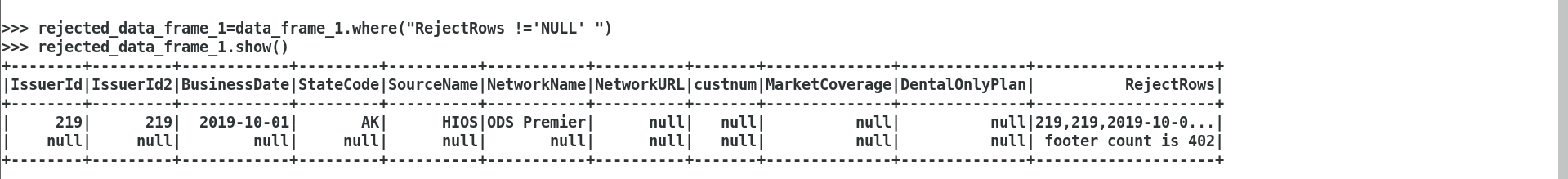
I. columnnameofcorruptrecord and store the rejected data in a new DF.

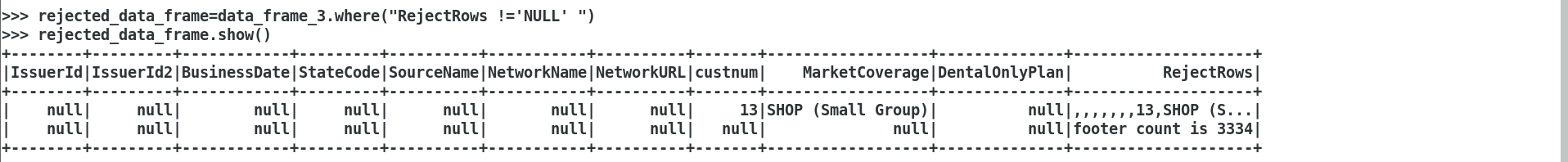
Eg: ,,,,,,,13,SHOP (Small Group) this record has to be rejected

II. Ignoreleading and trailing whitespaces

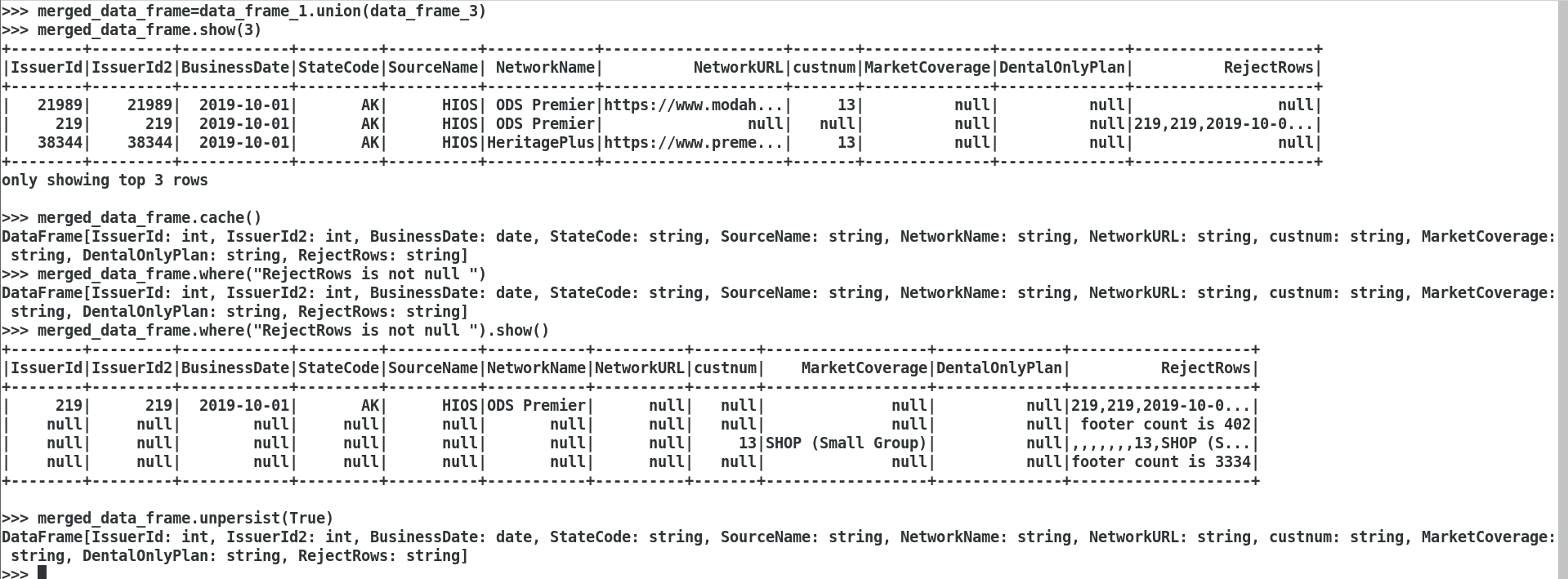
,,,,,,,13, SHOP (Small Group) ,Yes

Finally store the rejected records in a new dataframe in a single csv file.

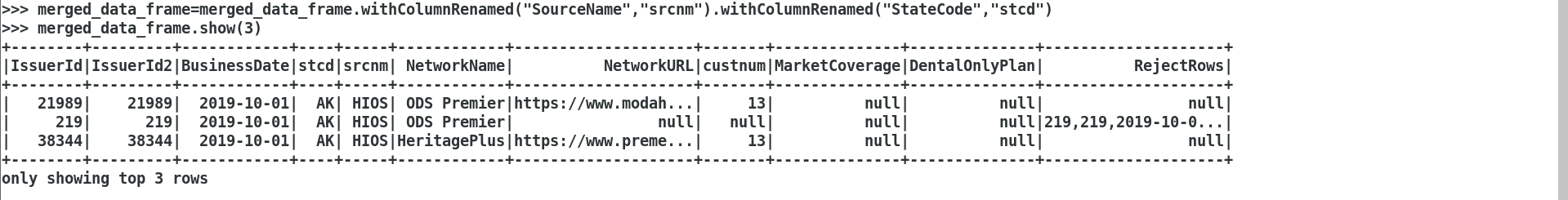




1. Apply the below **DSL** functions in the two DFs created (A and B dataframes) in step 2 , ***after merging these 2 dataframes (A and B dataframes).***

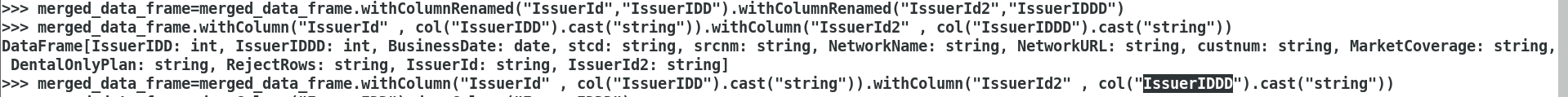
******

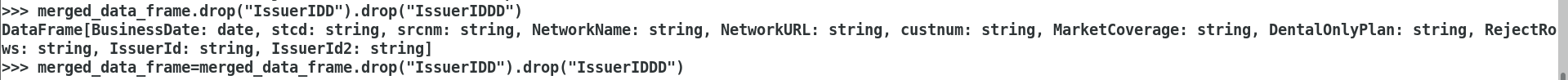
* 1. **Rename** the fields StateCode and SourceName as stcd and srcnm respectively.

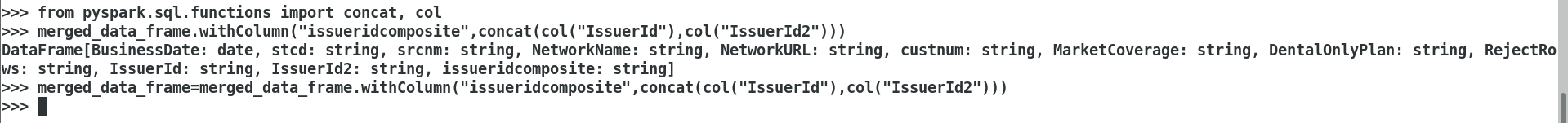


* 1. **Concat** IssuerId,IssuerId2 as issueridcomposite and make it as a new field

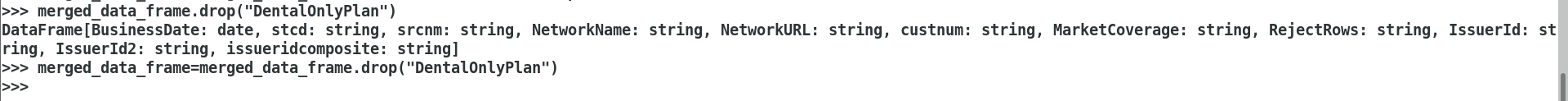
**Hint :** Cast to string and concat.



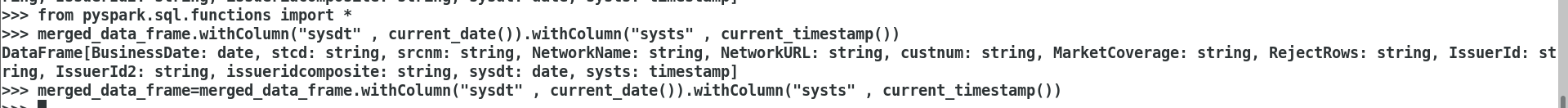




* 1. **Remove** DentalOnlyPlan column

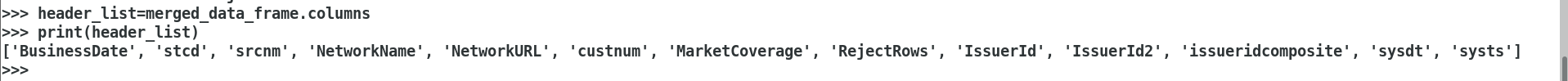


* 1. **Add columns** that should show the current system date and timestamp with the fields name of sysdt and systs respectively.

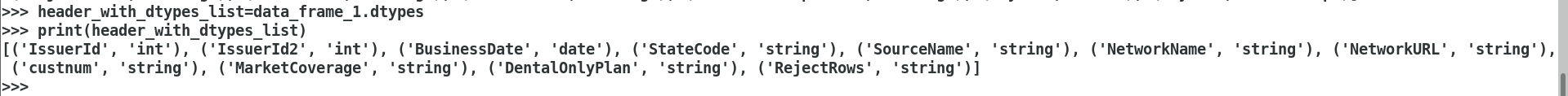


**Try the below interesting usecases seperately:**

* + 1. Identify all the column names and store in an List variable – use **columns** function.



* + 1. Identify all columns with datatype and store in a list variable and print – use **dtypes** function.



* + 1. Identify all integer columns alone and store in an list variable and print.



* + 1. Select only the integer columns identified in the above statement and show 10 records in the screen.

Hint: add all the column names in a list and apply it in df.select(list)

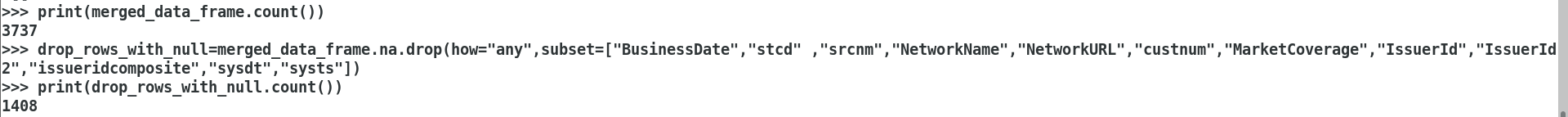


* + 1. Identify the additional column in the reject dataframe created in step 2 above by subtracting the columns between dataframe1 and dataframe3 created in step2.

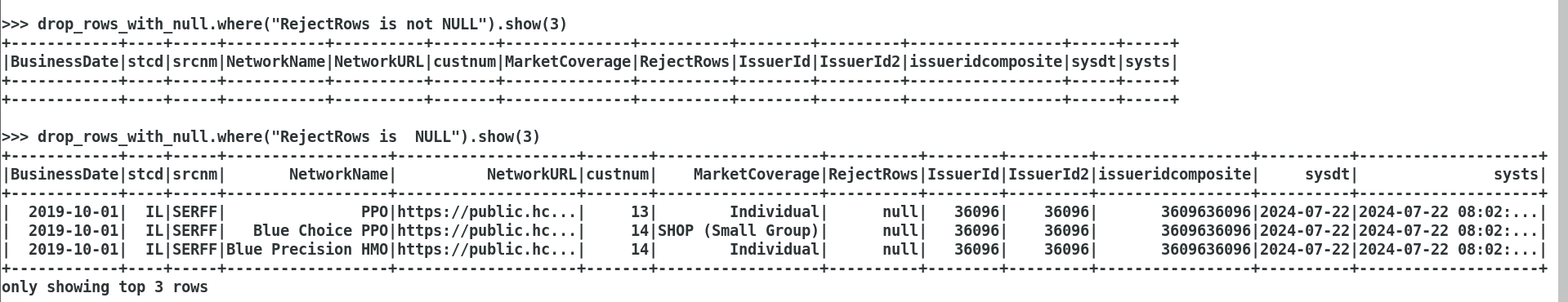
**Hint:** use columns function, then convert into set type and then do a subtraction.



1. **Take the DF created in step 3.d and Remove the rows** contains null in any one of the field and count the number of rows which contains all columns with some value.



(Below is to prove that the dataframe rows are dropped proper)



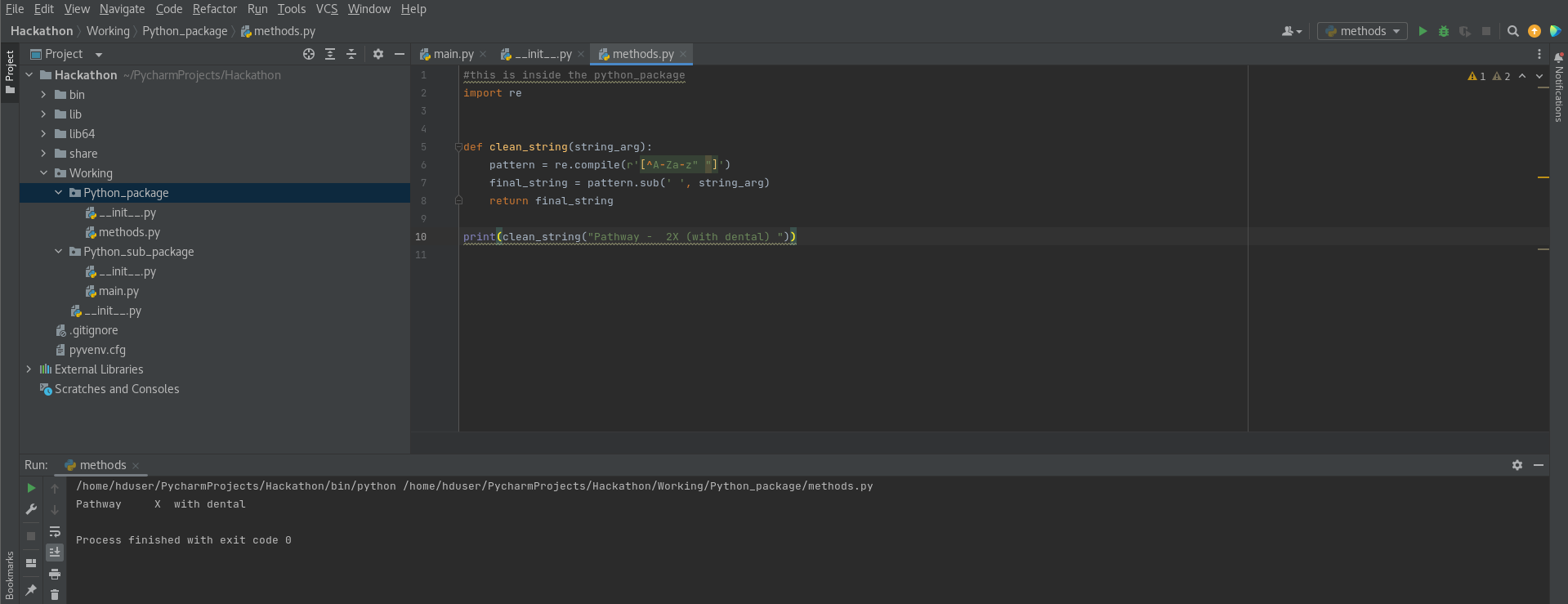
1. **Custom Method creation:** Create a package (org.inceptez.hack), module (allmethods), method (remspecialchar)

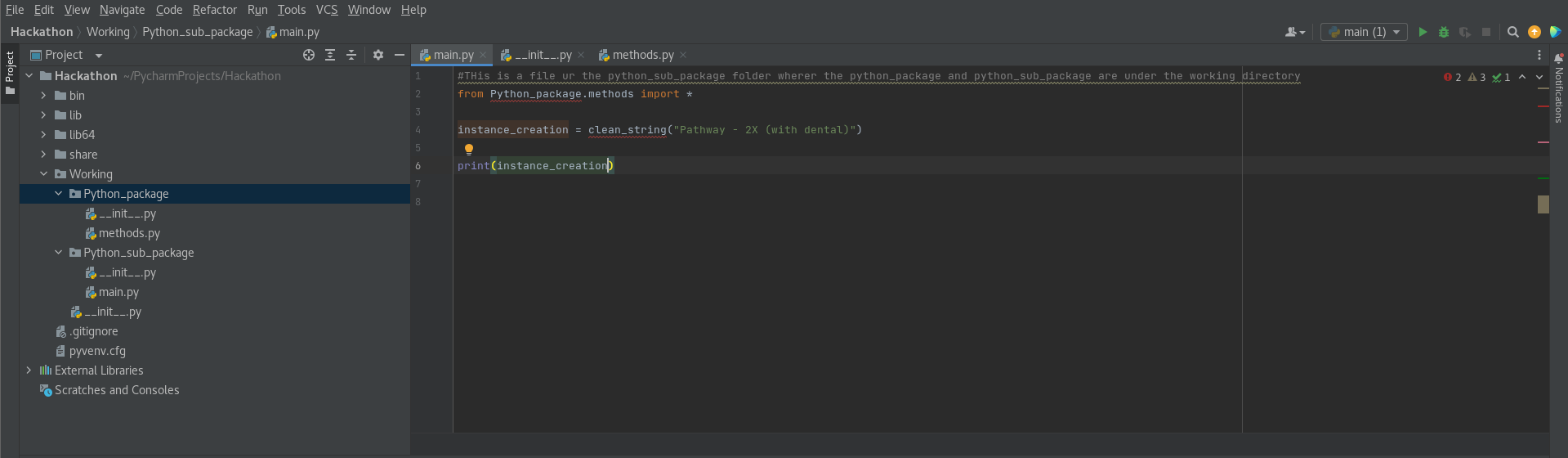
**Hint:** First create the function/method directly and then later add in a seperate pkg.module

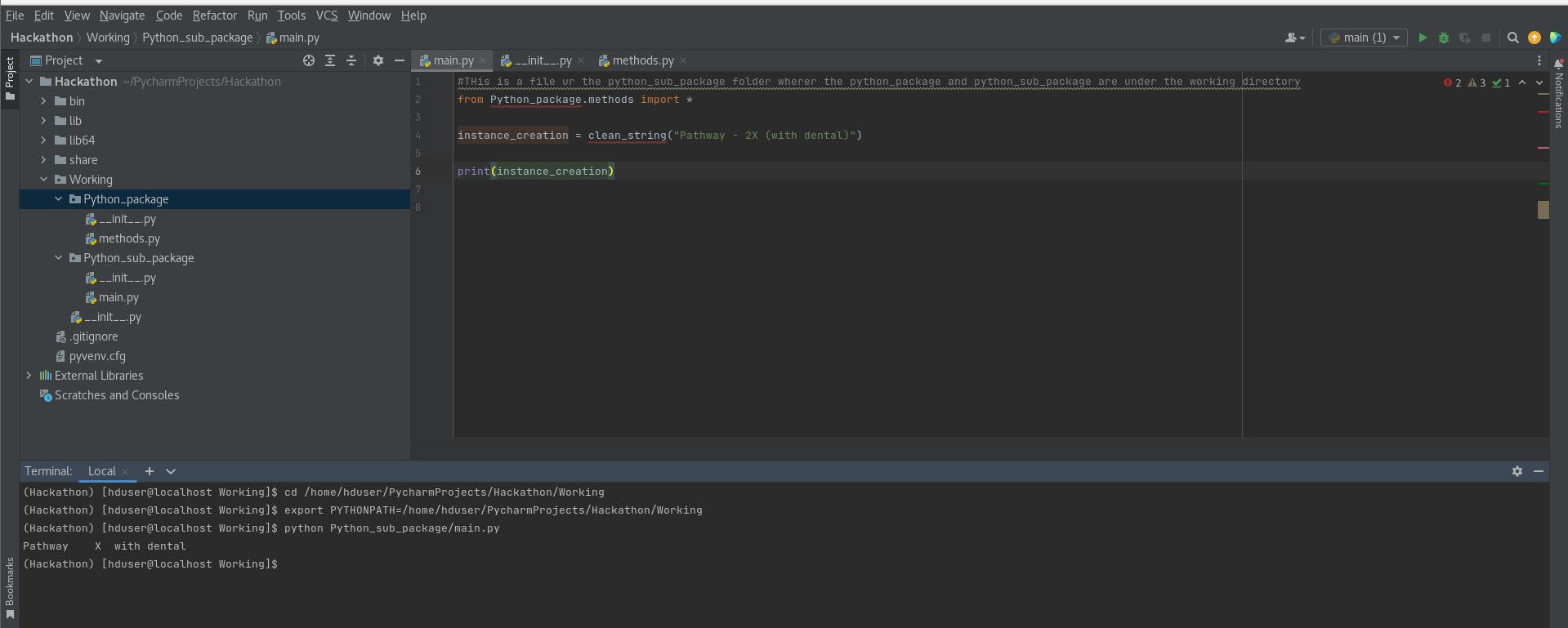
* 1. Method should take 1 string argument and 1 return of type string
  2. Method should remove all special characters and numbers 0 to 9 - ? , / \_ ( ) [ ] **Hint:** Use python regular expression function ie “re” function, usage of [] symbol should use \\ escape sequence.

Eg: regexp = re.sub("[;\\@%-^/:~,\*?\"<>|&'0-9]",'',a)

* 1. For eg. If I pass to the above method value as **Pathway - 2X (with dental)** it has to return **Pathway X with dental** as output.

****

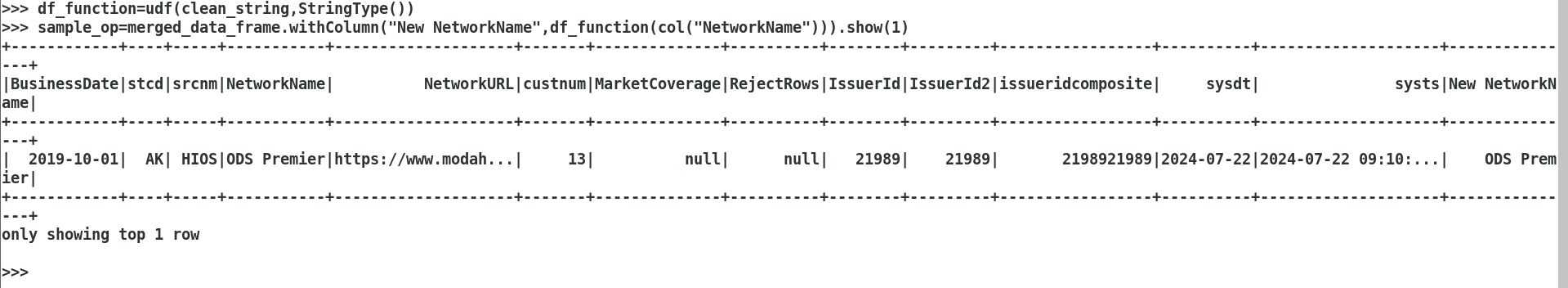
****

****

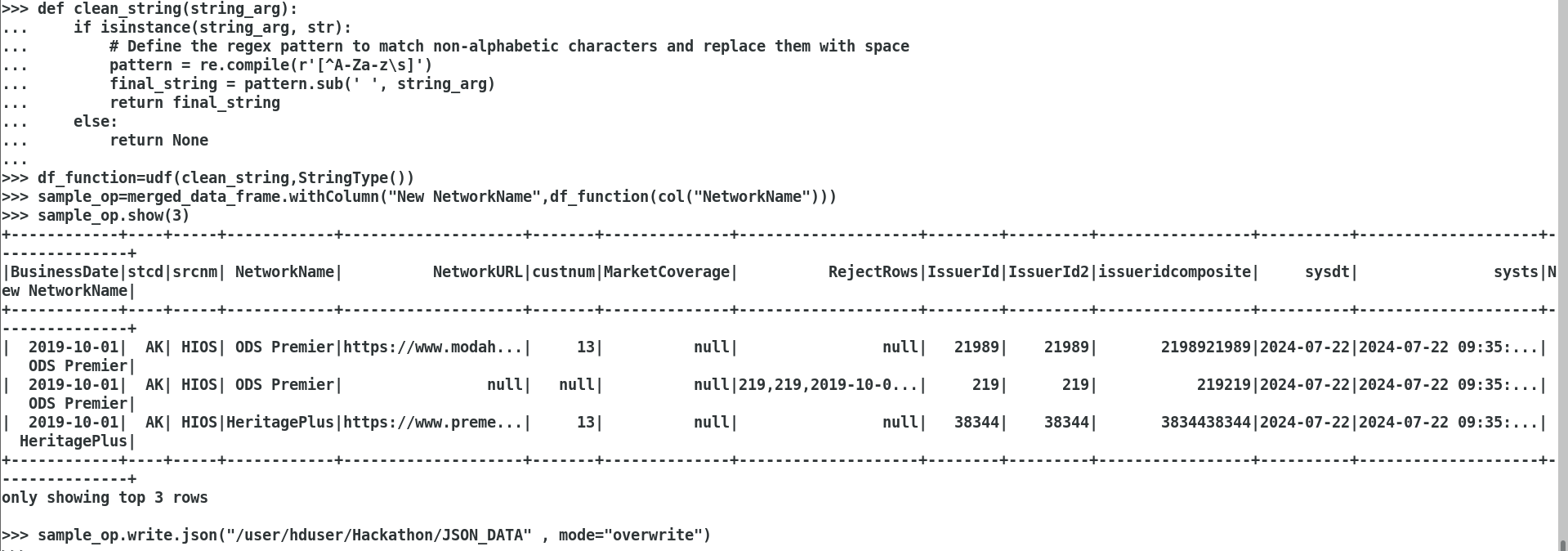
1. Import the package and refer the method generated in step 5 as a udf for invoking in the DSL function.



1. Call the above udf in the DSL by passing NetworkName column as an argument to get the special characters removed DF.



1. Save the DF generated in step 5 in JSON format into HDFS with overwrite option.

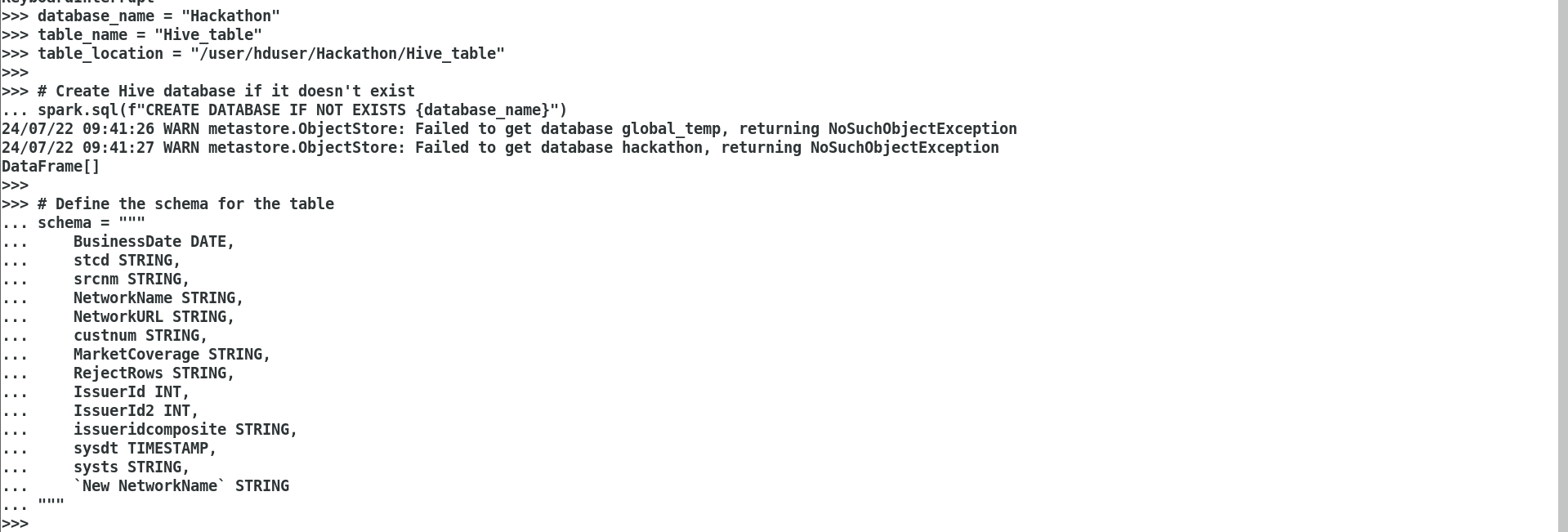


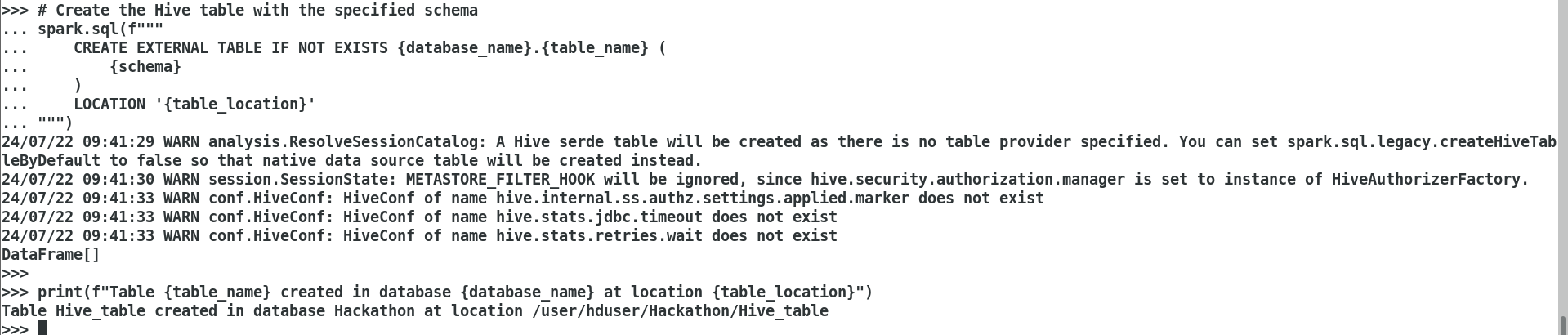
1. Save the DF generated in step 5 into CSV format with header name as per the DF and delimited by ~ into HDFS with overwrite option.

Note: Create a generic function namely writeToFile that should have 5 arguments passed as sparksession, filetype, location, delimiter, mode and call this function in the above step 26 and 27 to save the data rather than calling the write.csv and write.json directly.



1. Save the DF generated in step 5 into hive external table and append the data without overwriting it.



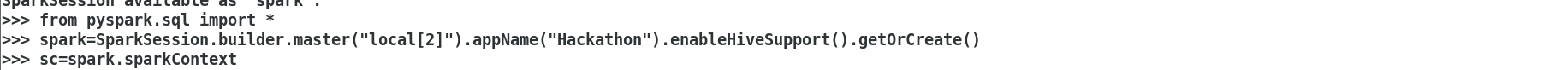


1. **Tale of handling RDDs, DFs and TempViews (20% Completion) – Total 75%**

**Loading RDDs, split RDDs, Load DFs, Split DFs, Load Views, Split Views, write UDF, register to use in Spark SQL, Transform, Aggregate, store in disk/DB**

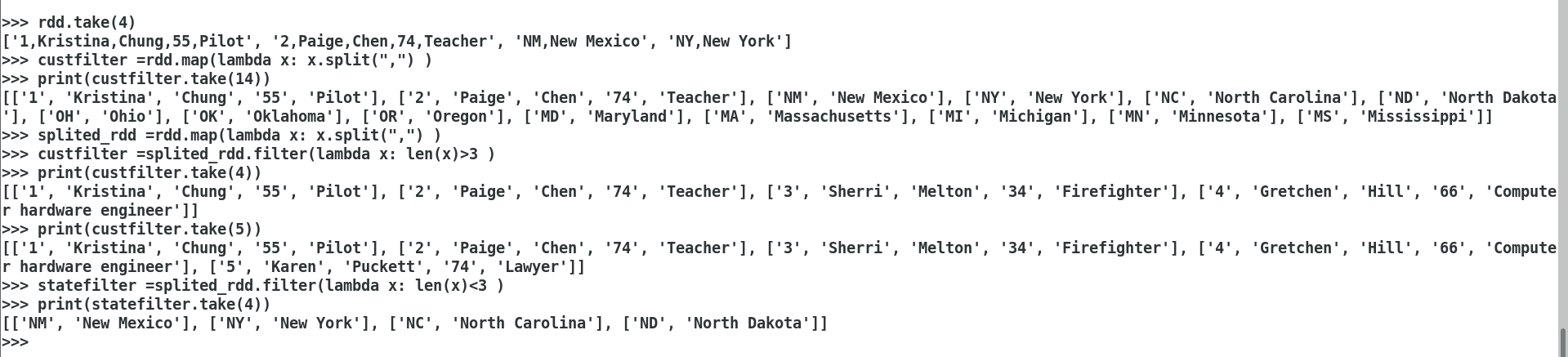
**Use RDD functions:**

1. **Load** the file3 (custs\_states.csv) from the HDFS location, using textfile API in an RDD custstates, this file contains 2 type of data one with 5 columns contains customer master info and other data with statecode and description of 2 columns.



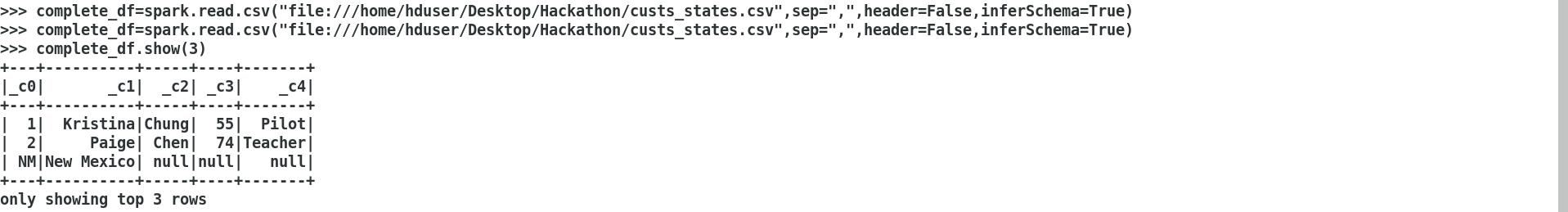


1. **Split** the above data into 2 RDDs, first RDD namely custfilter should be loaded only with 5 columns data and second RDD namely statesfilter should be only loaded with 2 columns data.



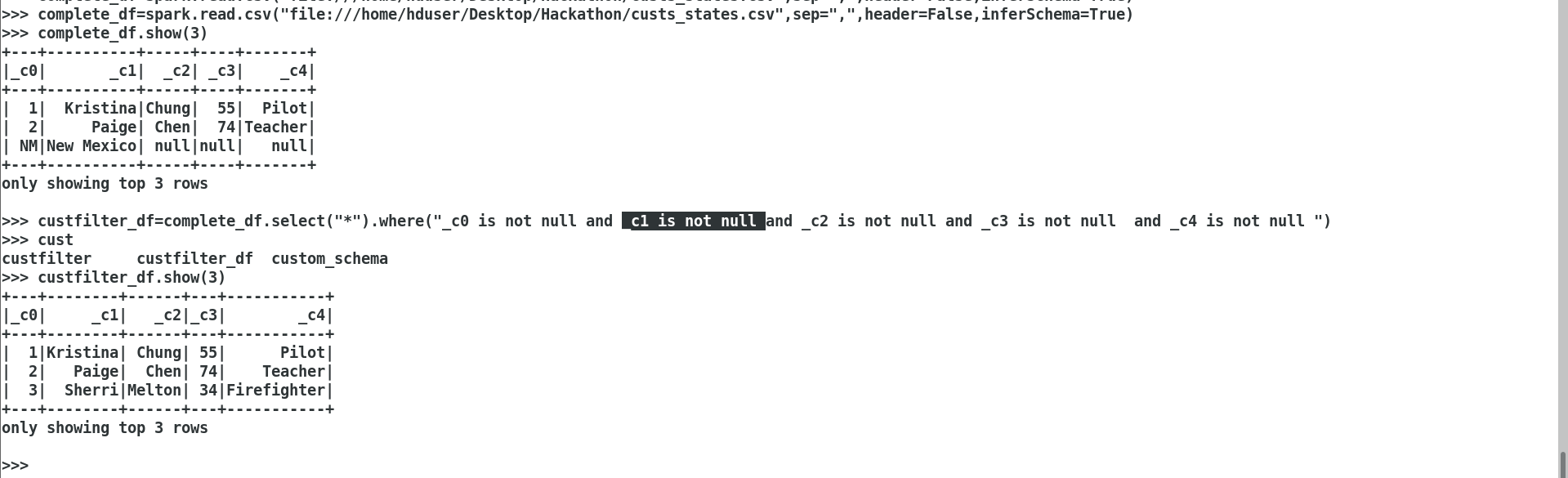
**Use DSL functions:**

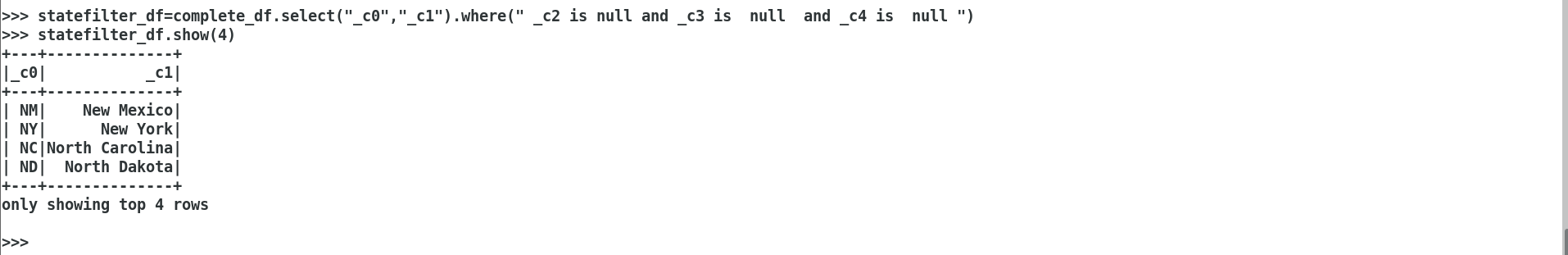
1. **Load** the file3 (custs\_states.csv) from the HDFS location, using CSV Module in a DF custstatesdf, this file contains 2 type of data one with 5 columns contains customer master info and other data with statecode and description of 2 columns.



1. **Split** the above data into 2 DFs, first DF namely custfilterdf should be loaded only with 5 columns data and second DF namely statesfilterdf should be only loaded with 2 columns data.

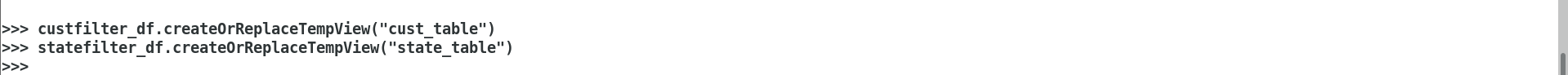
**Hint:** Use filter/where DSL function to check isnull or isnotnull to achieve the above functionality then rename, change the type and drop columns in the above 2 DFs accordingly.



****

**Use SQL Queries:**

1. **Register the above step 14 two DFs as temporary views** as custview and statesview.



1. **Register the DF generated in step 3.d as a tempview** namely insureview

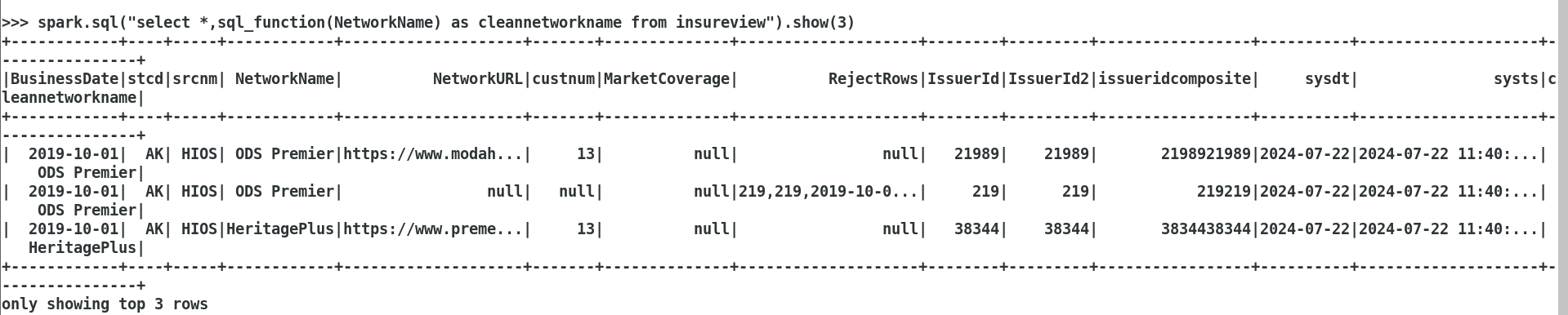


1. **Import the package and refer the method** created in step 5 in the name of remspecialcharudf using **spark udf registration**.
2. Write an **SQL query** with the below processing – set the spark.sql.shuffle.partitions to 4

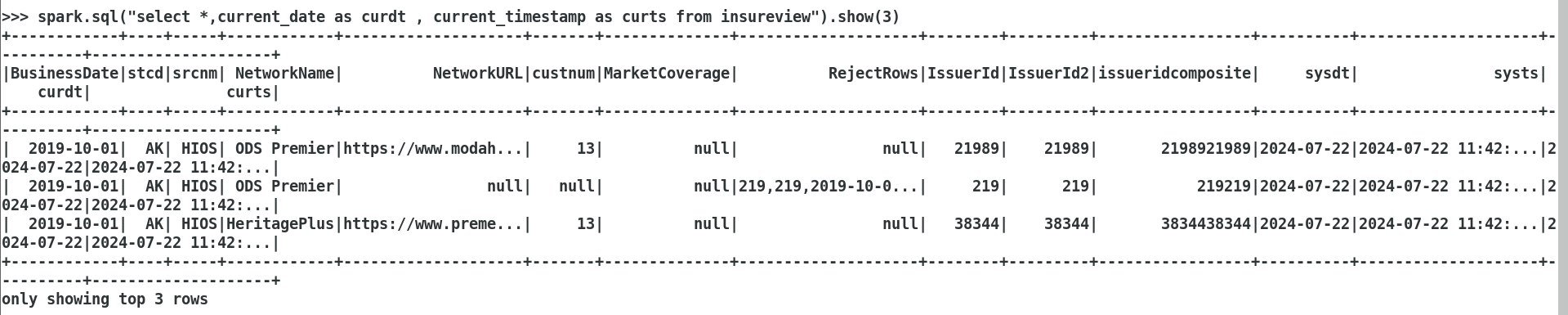


* 1. Pass NetworkName to remspecialcharudf and get the new column called cleannetworkname

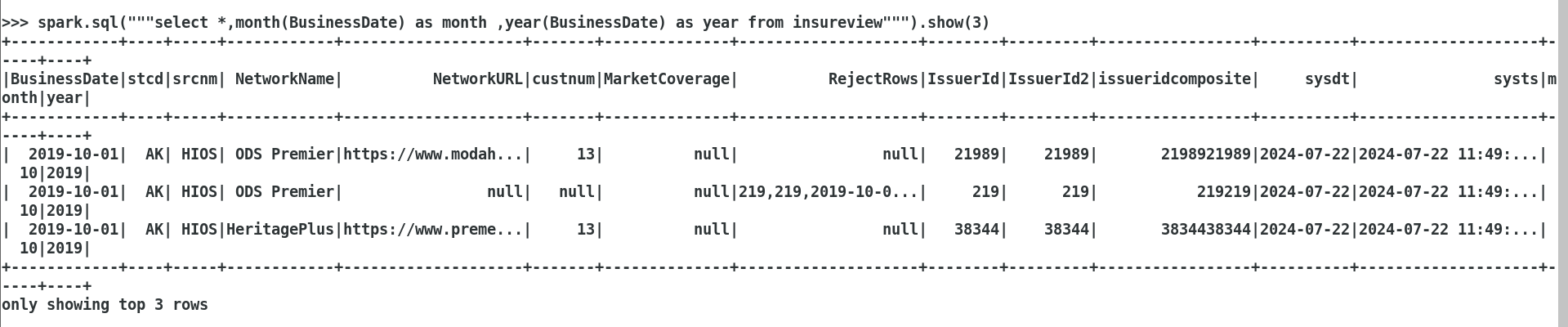




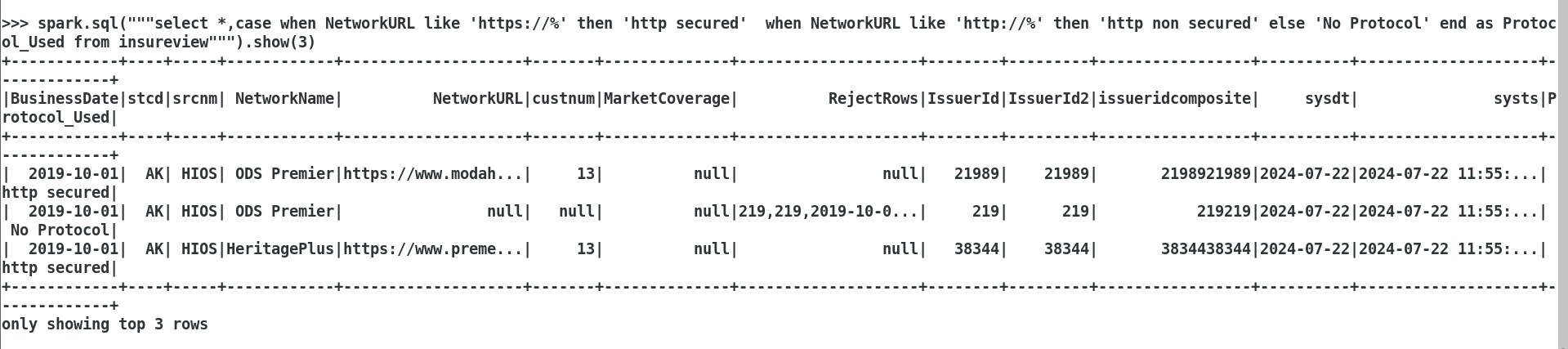
* 1. Add current date, current timestamp fields as curdt and curts.



* 1. Extract the year and month from the businessdate field and get it as 2 new fields called yr,mth respectively.



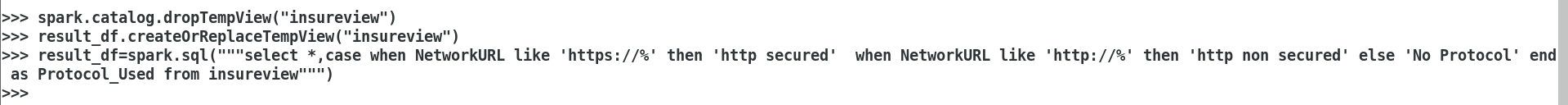
* 1. Extract from the protocol either http/https from the NetworkURL column, if http then print http non secured if https then secured else no protocol found then display **noprotocol.** For Eg: if <http://www2.dentemax.com/> then show **http non secured** else if [https://www2.dentemax.com/](http://www2.dentemax.com/) then **http secured** else if [www.bridgespanhealth.com](http://www.bridgespanhealth.com/) then show as **no protocol** store in a column called protocol.

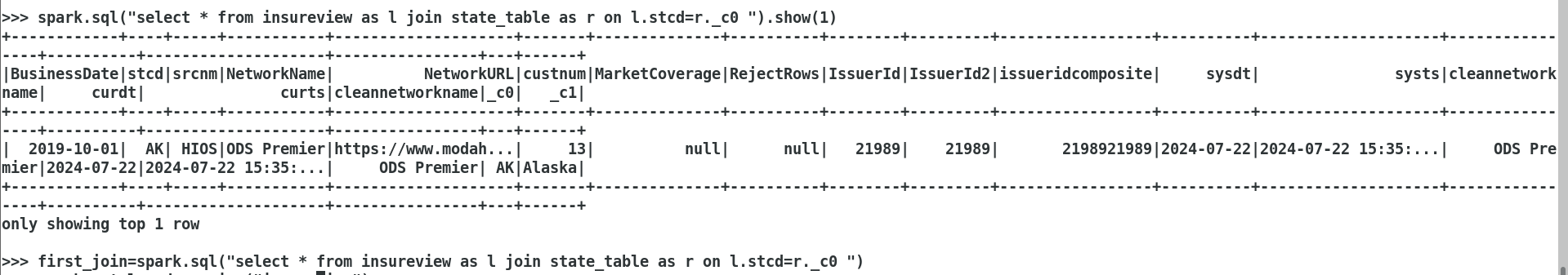


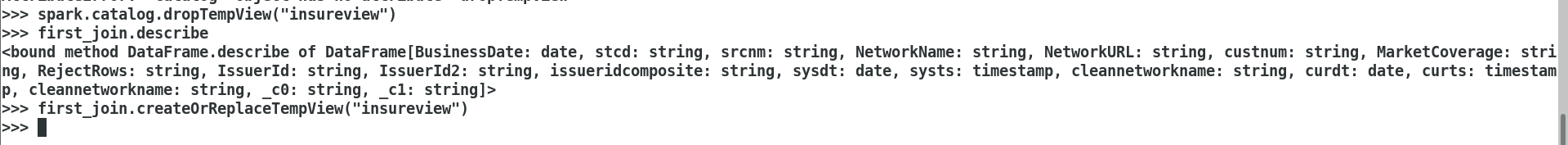
* 1. Display all the columns from insureview including the columns derived from above a, b, c, d steps with statedesc column from statesview with age,profession column from custview . Do an Inner Join of insureview with statesview using stcd=stated and join insureview with custview using custnum=custid.

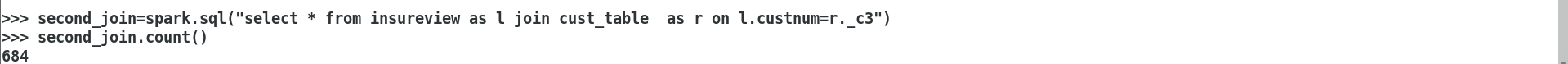












1. Store the above selected Dataframe in **Parquet** formats in a HDFS location as a **single file**.



1. Write an SQL query to identify average age, count group by statedesc, protocol, profession including a seqno column added which should have running sequence number partitioned based on protocol and ordered based on count descending and display the profession whose second highest **count** of a given state and protocol.

For eg.

**Seqno**,Avgage,count,statedesc,protocol,profession

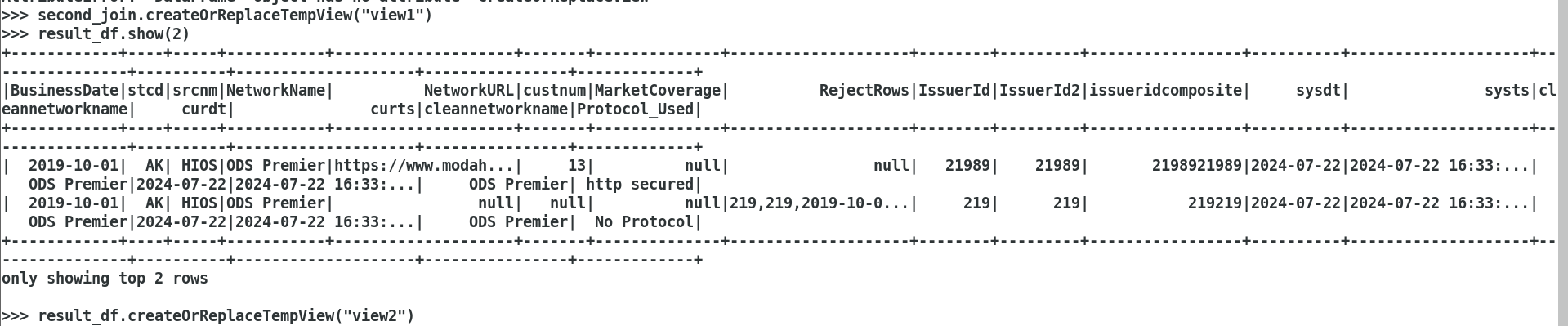
**1**,48.4,10000, Alabama,http,Pilot

**2**,72.3,300, Colorado,http,Economist

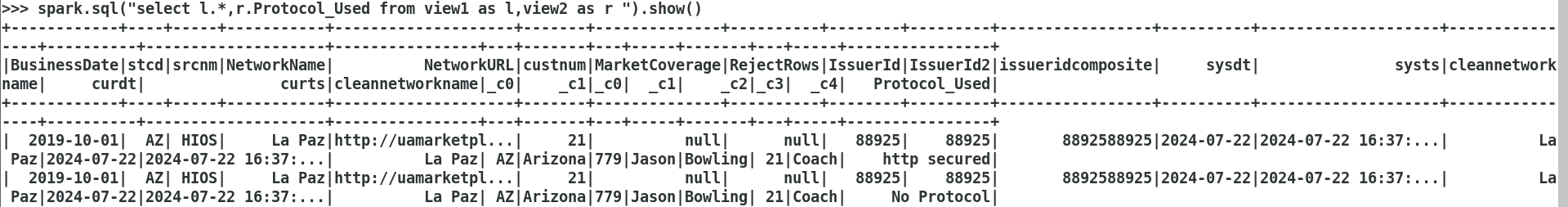
**1**,48.4,3000, Atlanta,https,Health worker

**2**,72.3,2000, New Jersey,https,Economist

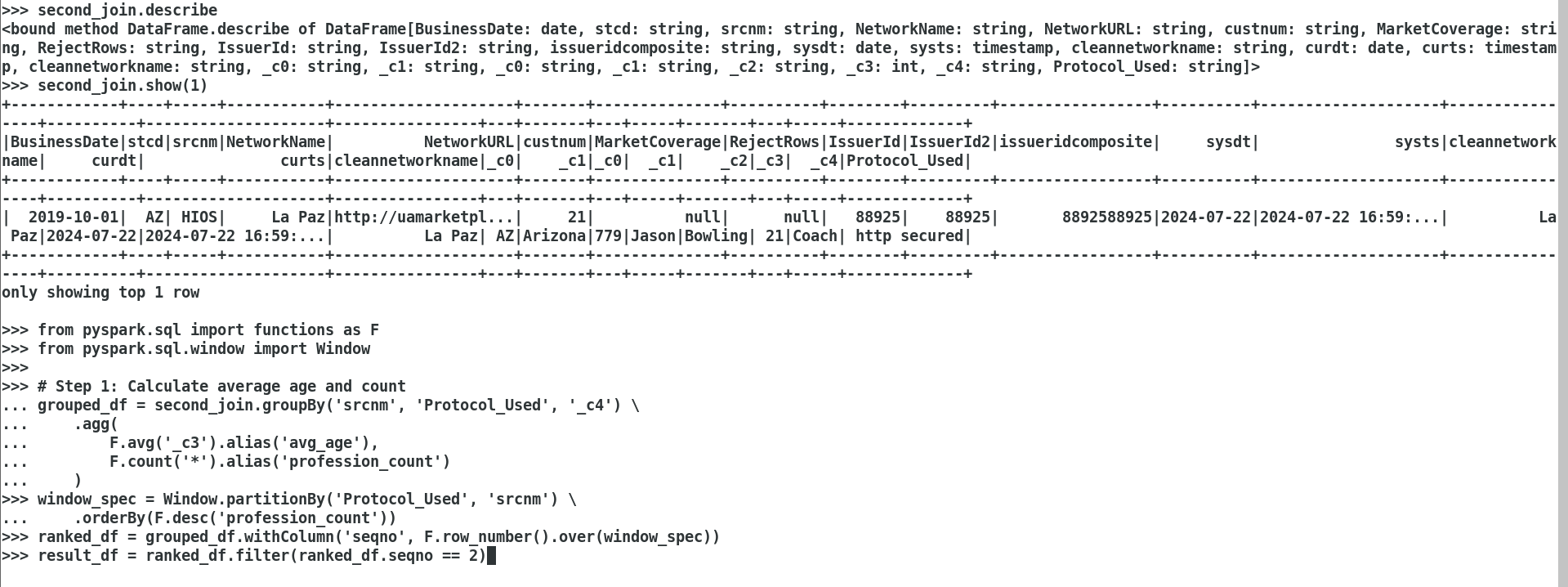
Forgot to add the protocol\_used so

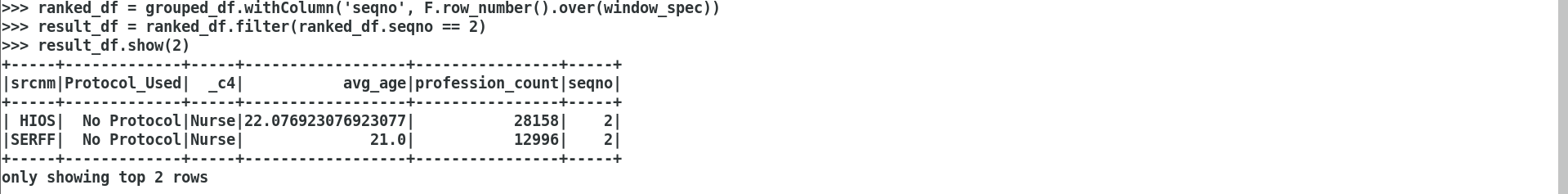


Done by joining it

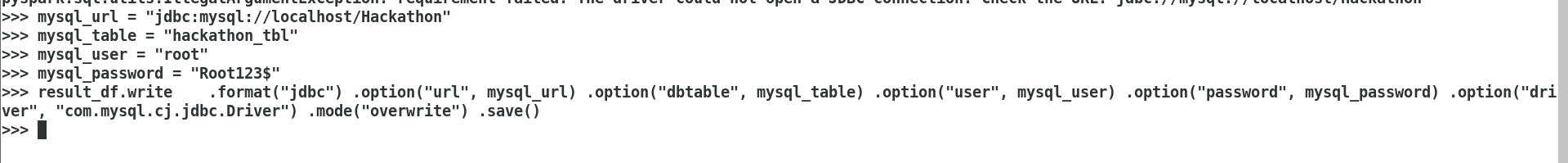


Now query is





1. Store the DF generated in step 20 into MYSQL table insureaggregated*.*

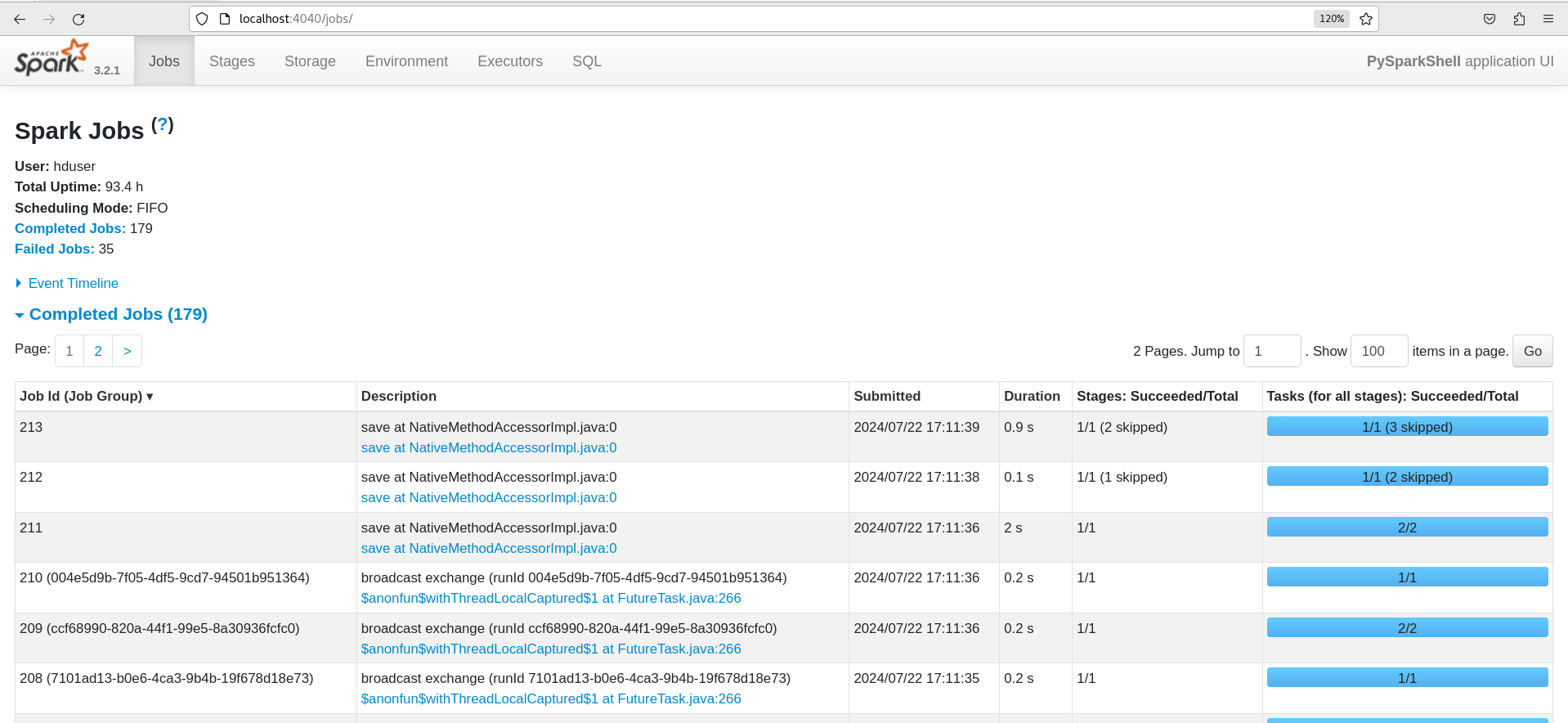


1. *Test the code in REPL/PyCharm and save it as a .py file and try submit with driver memory of 512M, number of executors as 4, executor memory as 1GB and executor cores with 2 cores.*
2. **Visualization (5% Completion) – Total 80%**

Login to spark ui and take the snapshot of the below items.

1. Jobs

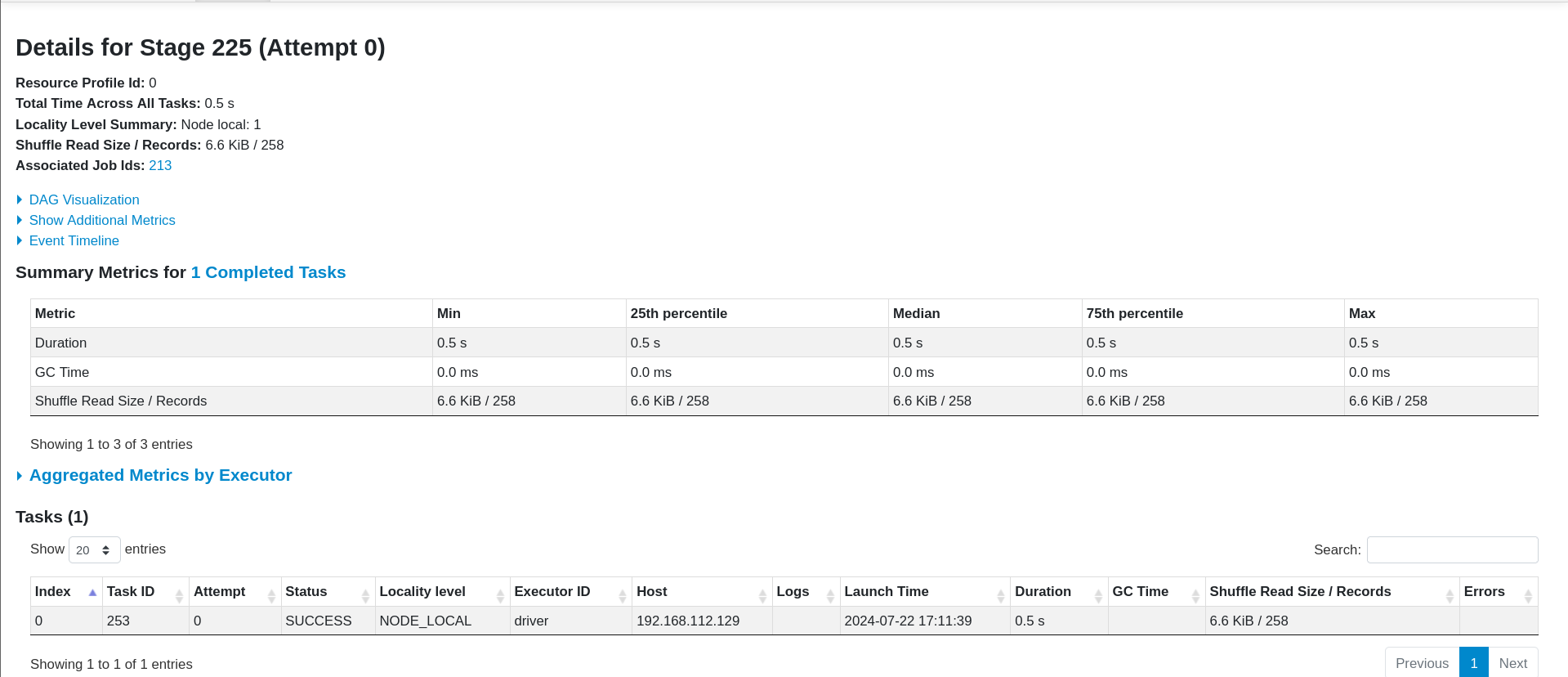
<http://localhost:4040>



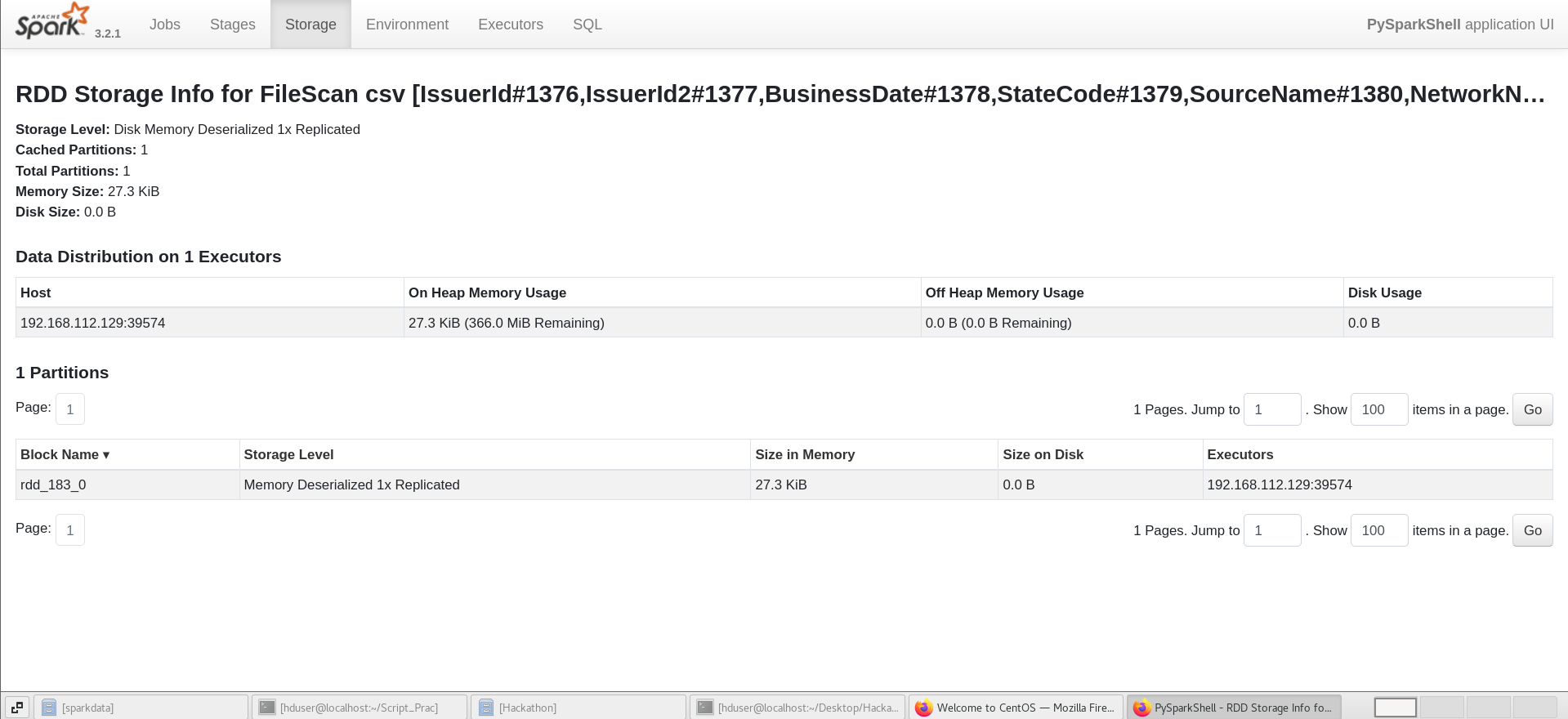
1. Stages

**Display the below items**

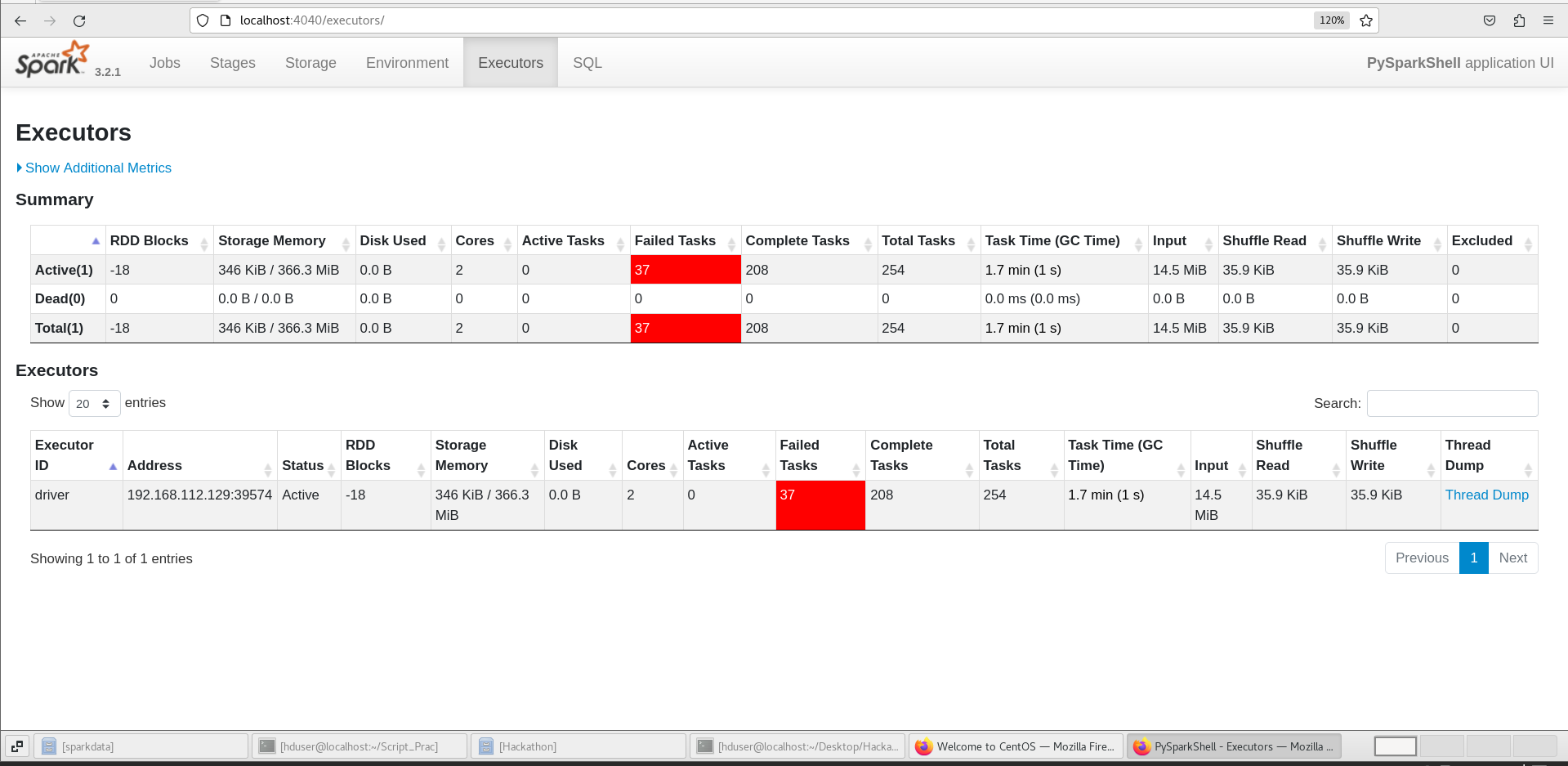
* 1. Task locality level in the Tasks
  2. Scheduler Delay
  3. Task Deserialization Time
  4. Shuffle Read Blocked Time
  5. Shuffle Remote Reads
  6. Result Serialization Time
  7. Getting Result Time
  8. Peak Execution Memory



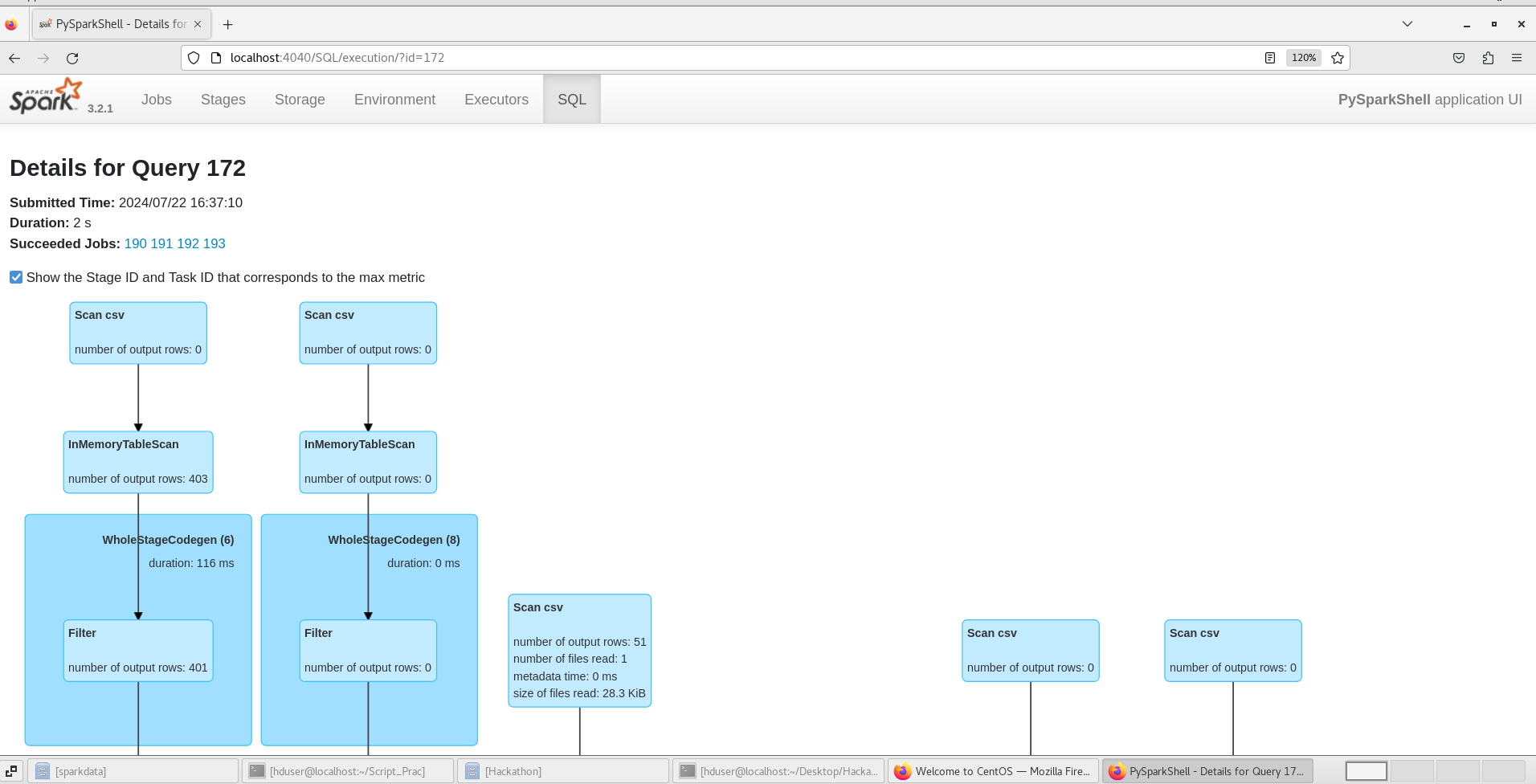
1. Storage
   1. Storage level
   2. Partitions info

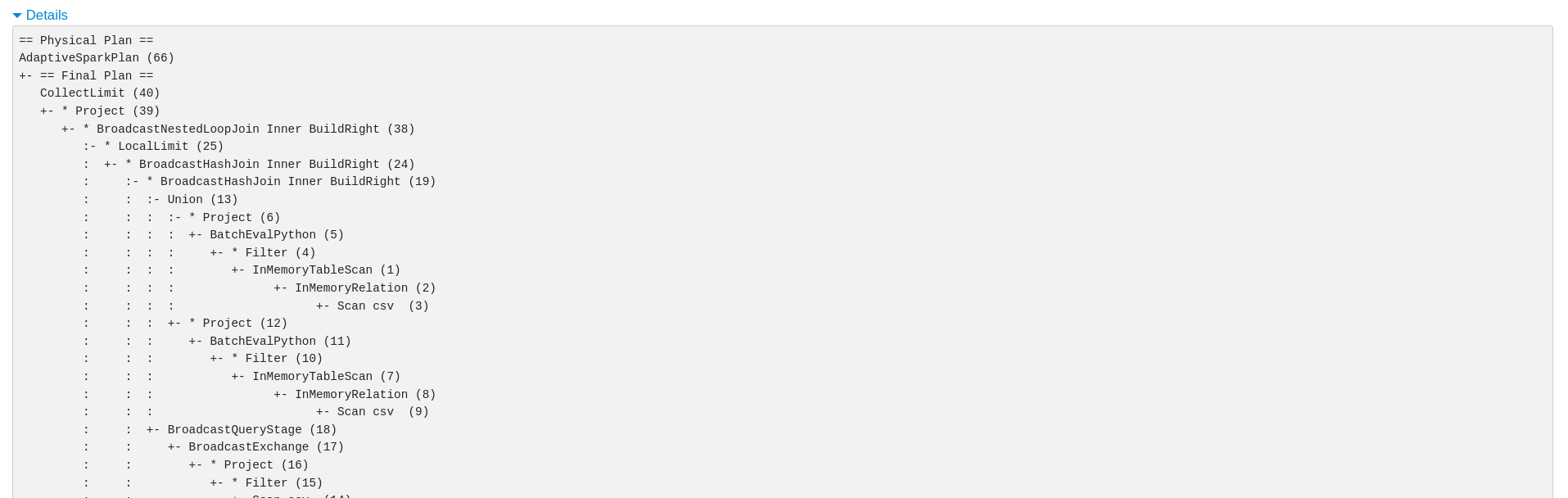


1. Executors info.



1. SQL
   1. DAG
   2. AST (Abstract Syntax Tree plan)





**Part B - Spark DF & SQL (Topic 4)**

1. **Running the above program module in Databricks or GCP Cloud (20% Completion) – Total 100%**

1. Try the above complete pyspark code generated in Databricks Environment

1. Lift and shift the above spark module into Databricks notebook
2. Upload the data into DBFS

**Or**

2. Try the above complete pyspark code generated in Databricks Environment

1. Lift and shift the above spark module into GCP Dataproc Cluster
2. Upload the data into GCS bucket

**Additional (Good to try) Hackathon Usecases to explore:**

Try the below one of both usecases as well (if you are interested):

**Usecase1:** Complex data structurization and standardization to achieve Schema Migration

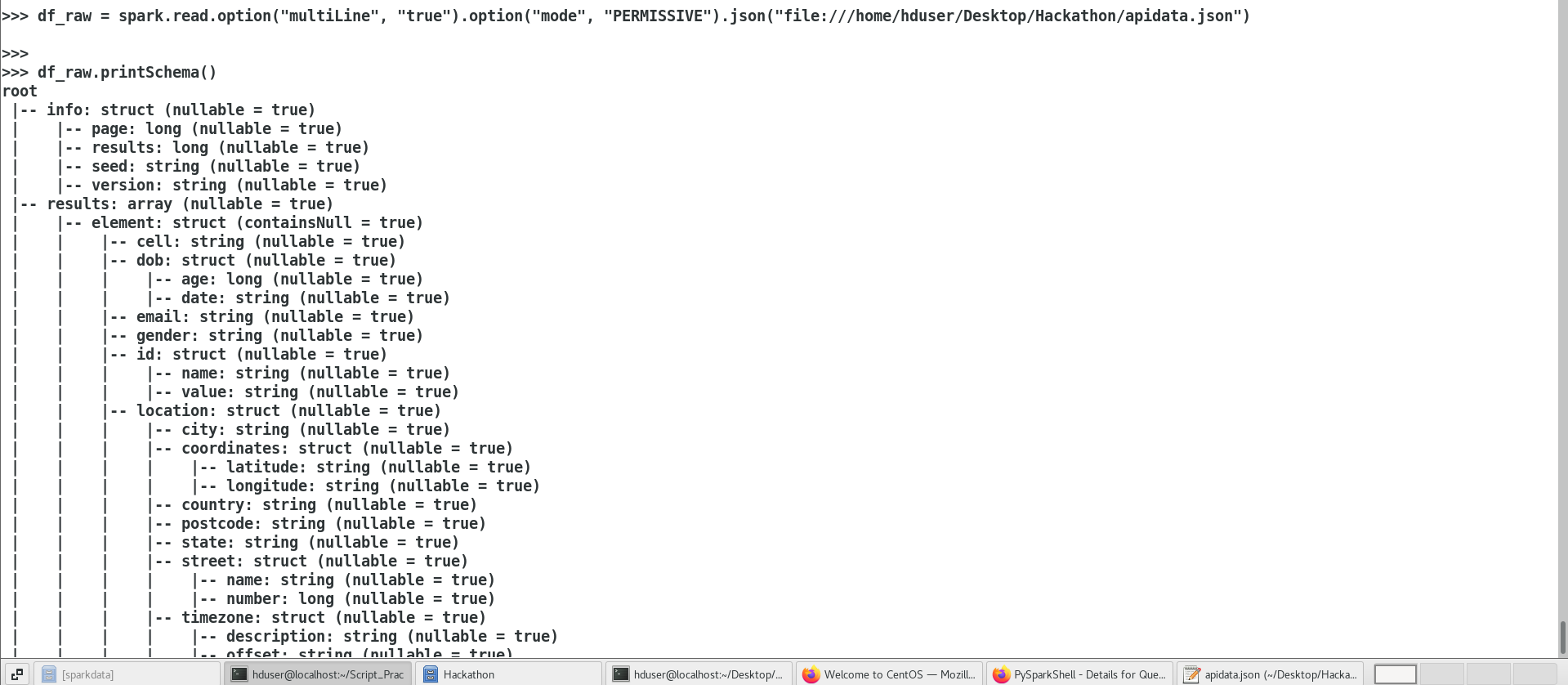
* + - 1. Make use of the structtype as mentioned below.

from pyspark.sql.types import \*  
custom\_schema = StructType([  
 StructField(**"results"**, ArrayType(StructType([  
 StructField(**"gender"**, StringType()),  
 StructField(**"location"**,  
 StructType([StructField(**"city"**, StringType()),StructField(**"state"**, StringType()),StructField(**"country"**, StringType()),StructField(**"postcode"**, StringType()),  
 StructField(**"coordinates"**, StructType([StructField(**"latitude"**, StringType()),StructField(**"longitude"**, StringType())])),])),  
 StructField(**"name"**, StructType([StructField(**"title"**, StringType()),StructField(**"first"**, StringType()),StructField(**"last"**, StringType())])),  
 StructField(**"email"**, StringType()),  
 StructField(**"login"**, StructType([StructField(**"uuid"**, StringType()),StructField(**"username"**, StringType())])),  
 StructField(**"dob"**, StructType([StructField(**"date"**, StringType()),StructField(**"age"**, StringType())])),  
 StructField(**"registered"**, StructType([StructField(**"date"**, StringType()),StructField(**"age"**, StringType())])),  
  
 ])))])



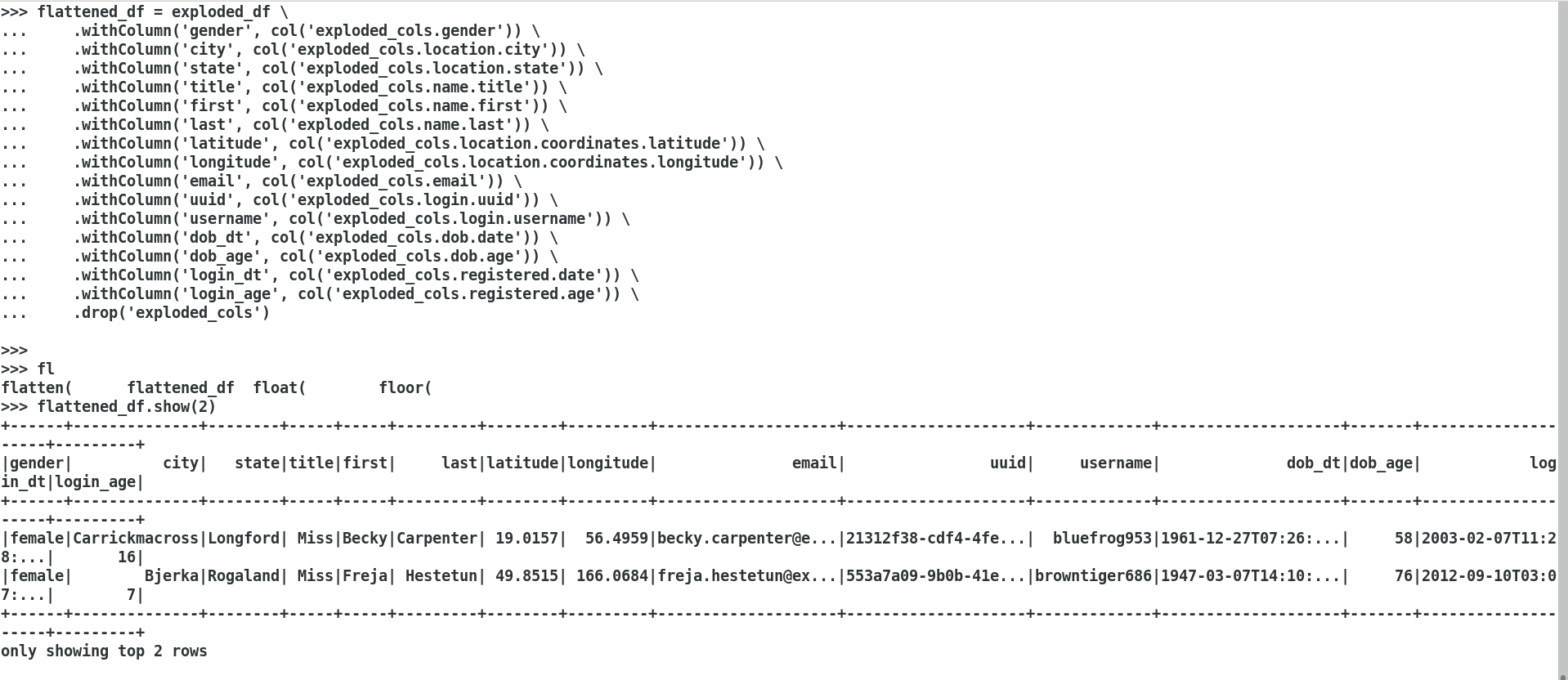
* + - 1. Read the complex json file provided (apidata.json) as a DF by applying the structure type provided above.
      2. Print the schema of the above DF created with multiline option true.

Below is for 2 and 3



* + - 1. Structurize the complex json using explode function to flatten the array to rows as given below.

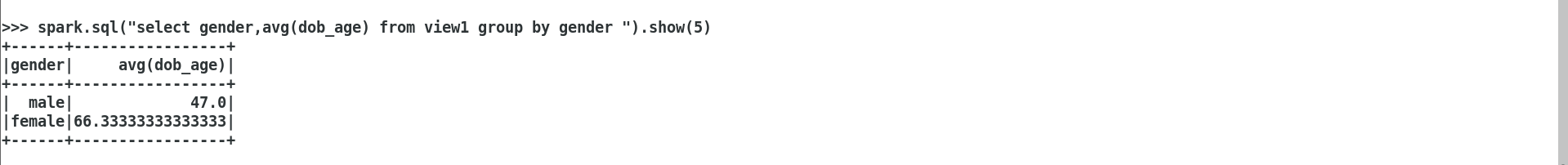
exploded\_df = df\_raw.selectExpr(**'explode(results) as exploded\_cols'**) flattened\_df = exploded\_df \  
 .withColumn(**'gender'**, col(**'exploded\_cols.gender'**)) \  
 .withColumn(**'city'**,col(**'exploded\_cols.location.city'**))\  
 .withColumn(**'state'**,col(**'exploded\_cols.location.state'**))\  
 .withColumn(**'title'**, col(**'exploded\_cols.name.title'**)) \  
 .withColumn(**'first'**,col(**'exploded\_cols.name.first'**))\  
 .withColumn(**'last'**,col(**'exploded\_cols.name.last'**))\  
 .withColumn(**'latitude'**,col(**'exploded\_cols.location.coordinates.latitude'**))\ .withColumn(**'longitude'**,col(**'exploded\_cols.location.coordinates.longitude'**))\  
 .withColumn(**'email'**, col(**'exploded\_cols.email'**)) \  
 .withColumn(**'uuid'**,col(**'exploded\_cols.login.uuid'**))\  
 .withColumn(**'username'**,col(**'exploded\_cols.login.username'**))\  
 .withColumn(**'dob\_dt'**,col(**'exploded\_cols.dob.date'**))\  
 .withColumn(**'dob\_age'**,col(**'exploded\_cols.dob.age'**))\  
 .withColumn(**'login\_dt'**, col(**'exploded\_cols.registered.date'**)) \  
 .withColumn(**'login\_age'**,col(**'exploded\_cols.registered.age'**))\  
 .drop(**'exploded\_cols'**)



* + - 1. Identify the below results out of it using DSL and SQL
         1. Identify the gender wise count



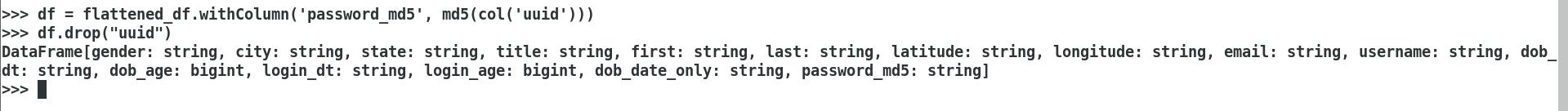
* + - * 1. Identify the gender wise average age



* + - * 1. Extract only the date portion of the dob column



* + - * 1. Apply md5 masking algorithm to the password column



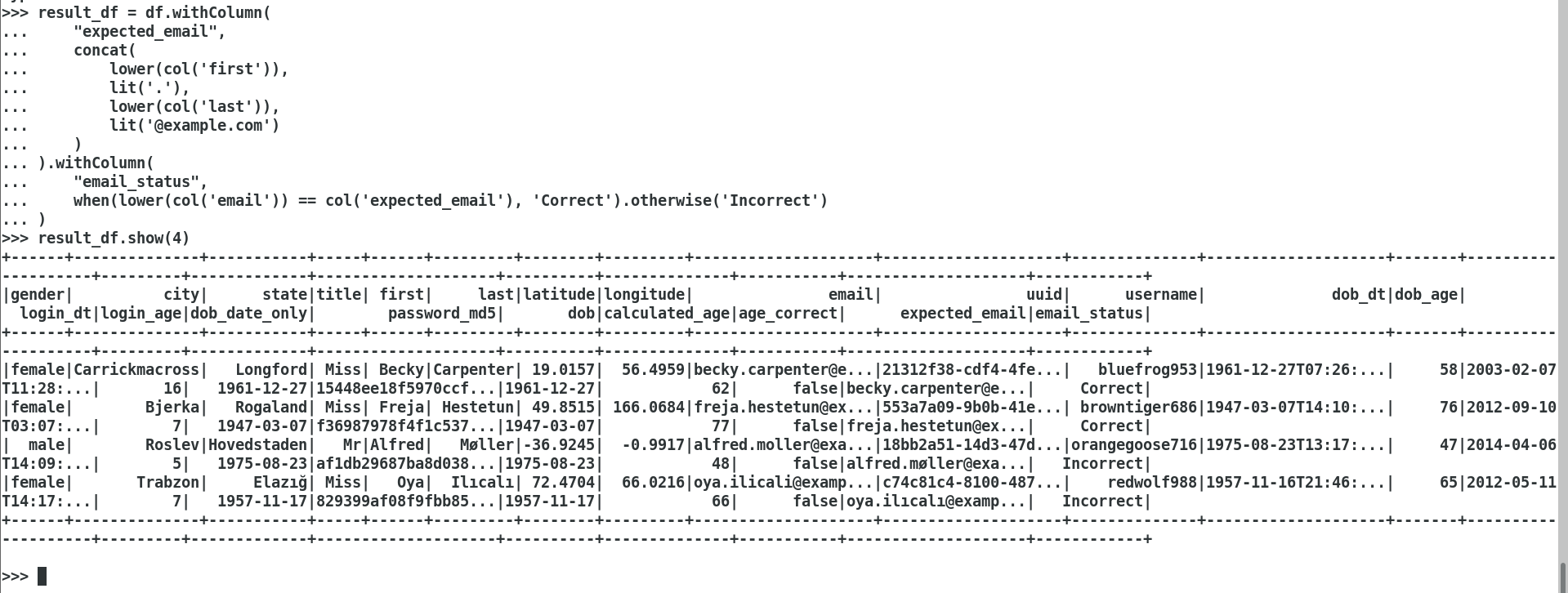
* + - * 1. Store the cell number in the format of 000-000-0000 for eg. 8256249837 has to 825-624-9837

df = df.withColumn( 'formatted\_cell\_number', regexp\_replace(col('cell\_number'), '([0-9]{3})([0-9]{3})([0-9]{4})', '$1-$2-$3') )

* + - * 1. Calculate the age is calculated correctly using the dob compared with the age column



* + - * 1. Identify the email is valid

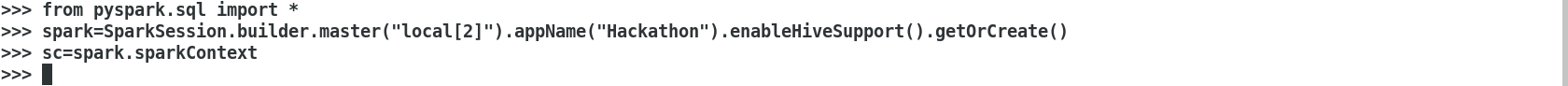


**Usecase2:**

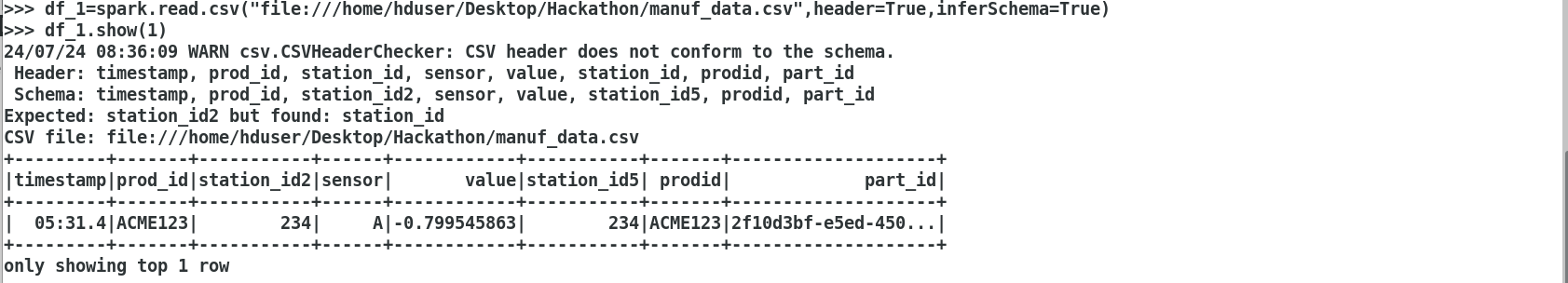
**Manufacturing Data Processing Usecase -** Data Munging, Enrichment and Wrangling functinalities.

**Requirement Spec-** Manufacturing data shall be prepared so it can be used in a machine-learning model. As part of the proof-of-concept, your person-of-contact in the plant provides you with about 1h worth of data collected at a small production line. The data-scientists building the machine-learning model have requested that you provide it to them in the "tidy" format.

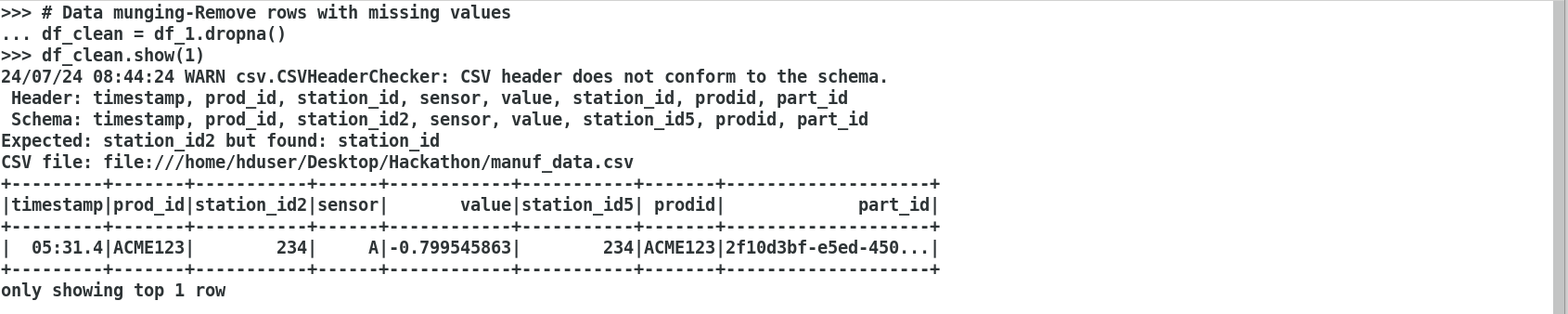
***The Tasks:***



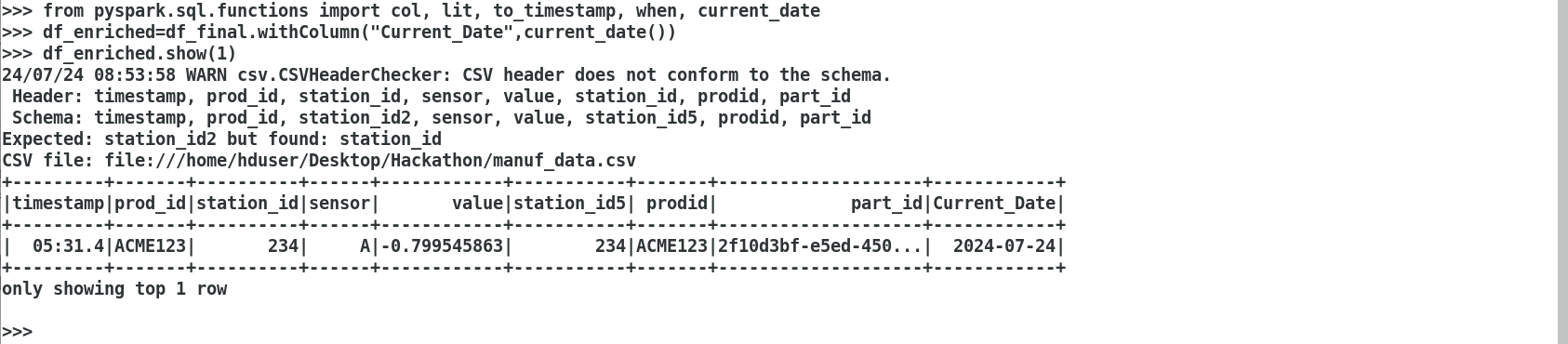
1. Please familiarize yourself with the given data set (manuf\_data.csv) and take notes of any peculiarities you discover throughout working on the dataset.

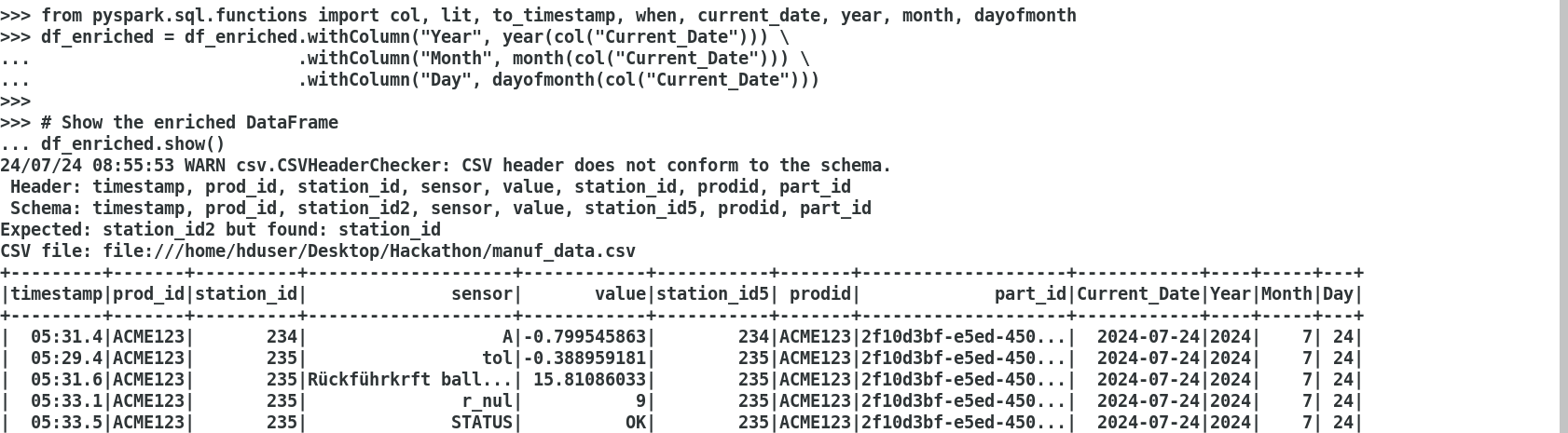


2. Implement Data munging, Enrichment and Wrangling logics using pyspark DSL/SQL.

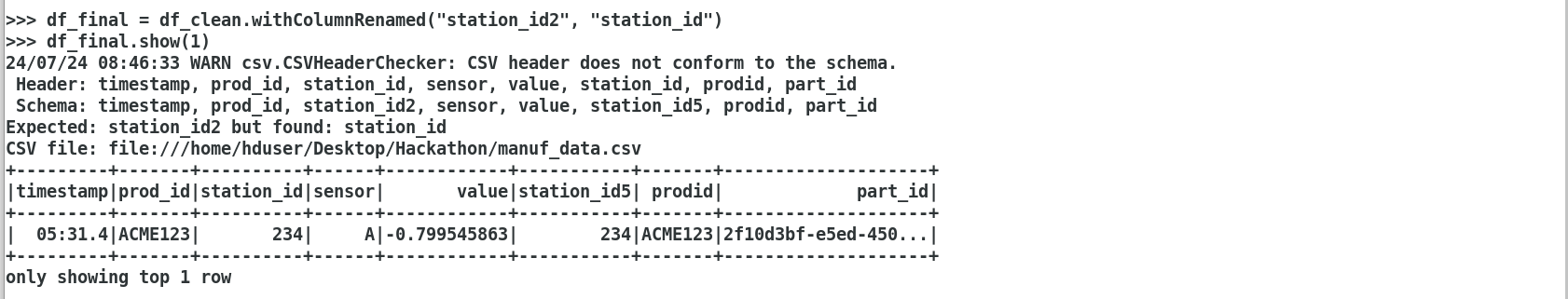


Data Enrichment:

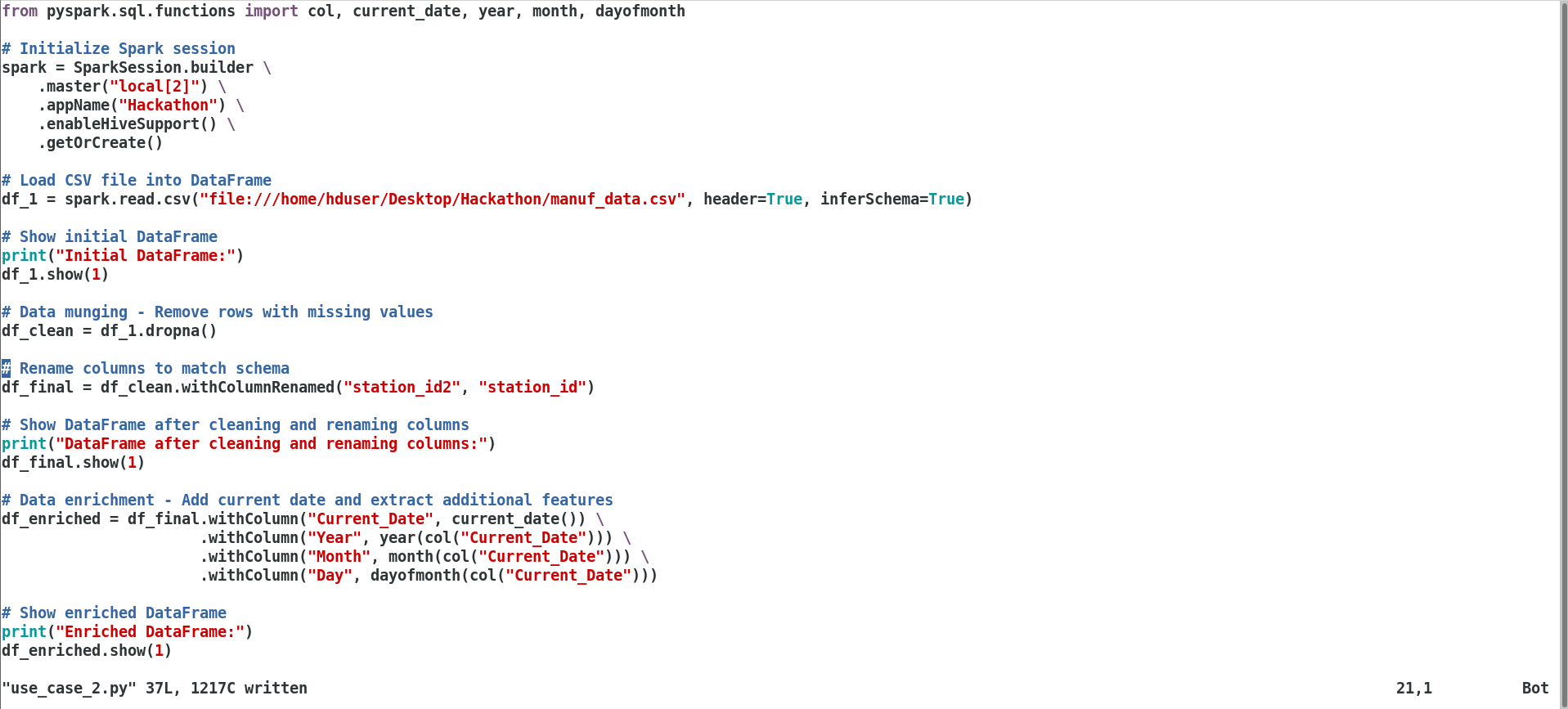




Data Wrangling :

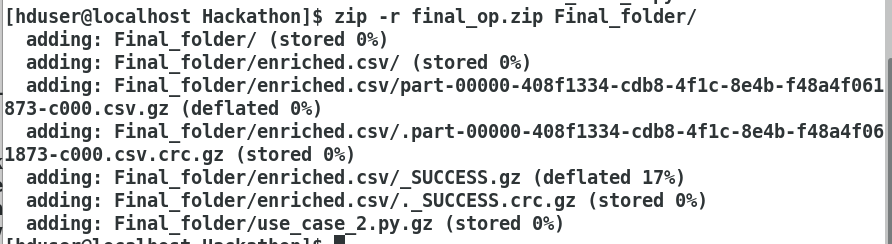


3. Provide your application logic as a script that can be called from the command line. This script shall be deployed and executed in a production environment





4. Submit the tidy data table in CSV format alongside of your solution’s source code



5. Create a presentation as like the given sample ppt (Manufacture data Code Challenge solution.pptx) mentioning the data pipeline strategies used and send it along with the hackathon usecases code.\