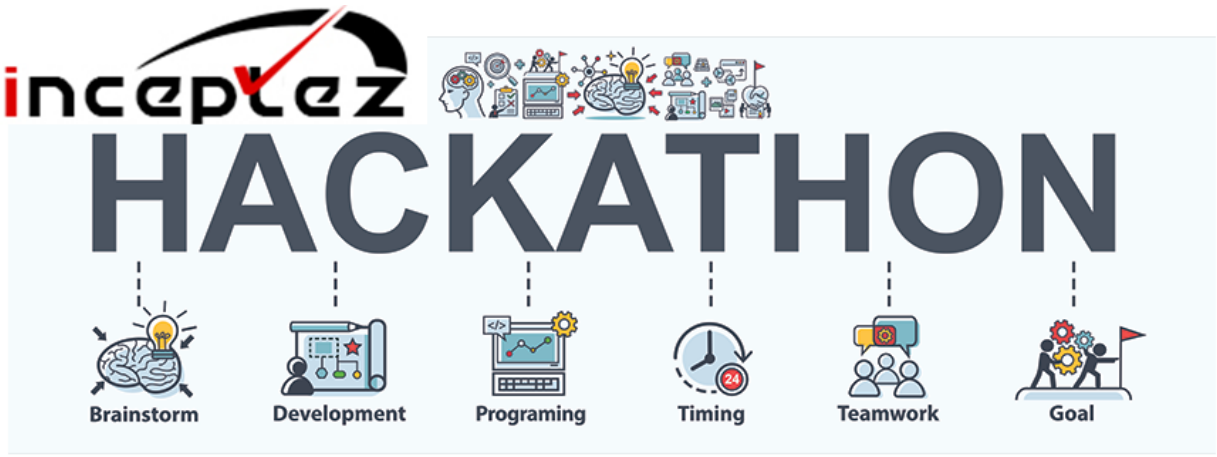


Its time for the Challenge ..... Let us Experiment, Explore, Evolve & Excel It !!!

Spark (Core, SQL, Streaming) Hackathon based on Interview scenarios

Duration of Hackathon: 5 Hours (2 hours for Core, 2 hours for DF/SQL and 1 hour for Realtime Streaming)

We got a very good response and feedback from our students after completing these usecases such as 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, transformation, ETL and schema migration of unpredicted data sets using Spark core, SQL, Dataframe functions and Realtime streaming which can help you solving interview questions by splitting, merging, applying functions in data sets. It carries lot of important SQL interview questions also..

Please read the below points before proceeding to the Hackathon:

- **Download the data into HDFS location (/user/hduser/sparkhack2) and start progress.**
- **Import required classes including sparkcontext, sqlcontext and other required objects, classes. Lets import other required libraries later as and when it is needed.**
- **Create Sparkcontext, sqlcontext or spark session.**
- **Provide the final code developed in Eclipsce, you can use REPL for development, but finally code should be in Eclipsce.**

- **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.**

Note:

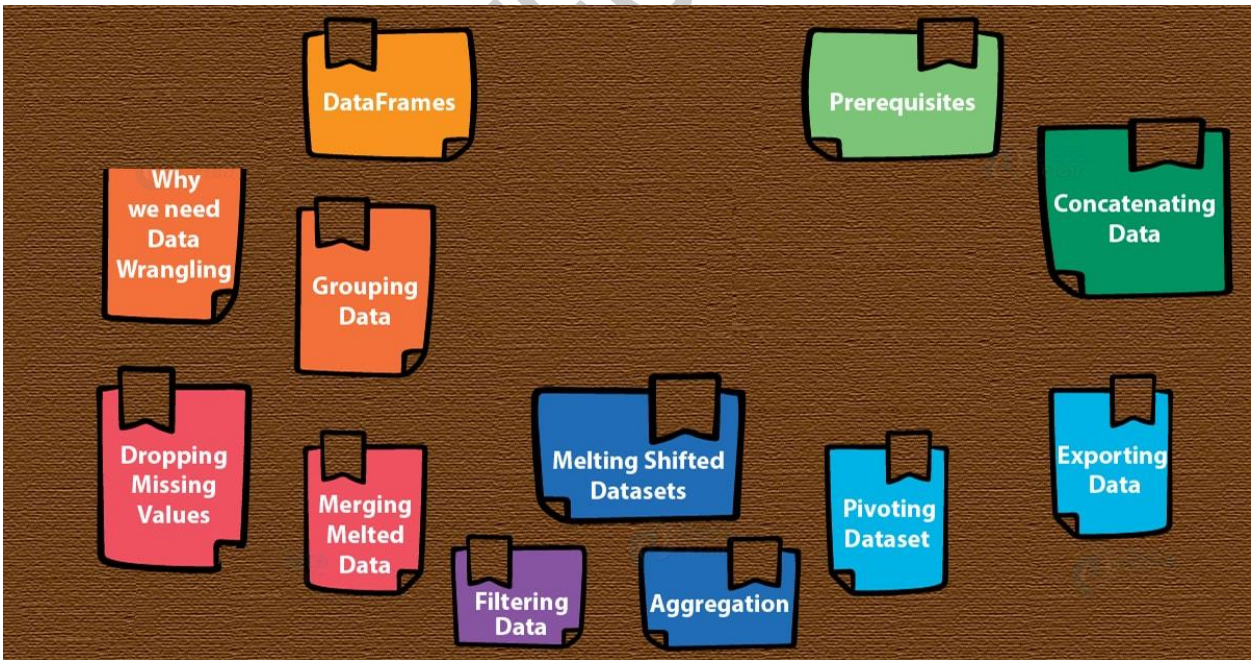
Start with Part A - Step 1 and 2 (rdd) first

or

Start with Part B - Step 3, 4 and 5 (dsl/sql) first

or

Start with Part C - Step 6 (realtime streaming) first as per your convenience, but try to complete all it's a matter of time and continuous efforts.



Part A - SPARK CORE RDD TRANSFORMATIONS / ACTIONS (Step 1 & 2)

1. Data cleaning, cleansing, scrubbing (20% Completion)

Loading RDDs, remove header, blank lines and impose caseclass schema

1. **Load** the file1 (insuranceinfo1.csv) from HDFS using textFile API into an RDD insuredata
2. **Remove the header** line from the RDD contains column names.

   Hint:

   Use rdd1.first -> rdd1.filter to remove the first line identified to remove the header.
3. Display the count and **show** few rows and check whether header is removed.
4. **Remove the blank lines** in the rdd.

   Hint: Before splitting, trim it, calculate the length and filter only row size is not = 0.
5. **Map and split** using ',' delimiter.

   Hint: To avoid getting incomplete rows truncated, use **split(",",-1)** instead of split(",")

   For Eg:

   If you take the rdd.first it should show the output like the below

   Array(21989, 21989, 2019-10-01, AK, HIOS, ODS Premier,
   https://www.modahealth.com/ProviderSearch/faces/webpages/home.xhtml?dn=ods, 13, "", "")

   Output should not be like given below

   Array(21989, 21989, 2019-10-01, AK, HIOS, ODS Premier,
   https://www.modahealth.com/ProviderSearch/faces/webpages/home.xhtml?dn=ods, 13)
6. **Filter number of fields** are equal to 10 columns only - analyze why we are doing this and provide your view here..
7. **Add case class** namely insureclass with the field names used as per the header record in the file and apply to the above data to create schemaed RDD.
8. **Take the count of the RDD** created in step 7 and step 1 and print how many rows are removed in the cleanup process of removing fields does not equals 10.
9. **Create another RDD namely rejectdata** and store the row that does not equals 10 columns.
10. **Load** the file2 (insuranceinfo2.csv) from HDFS using textFile API into an RDD insuredata2

11. **Repeat from step 2 to 9** for this file also and create the final rdd including the filtering of the records that contains blank IssuerId,IssuerId2 for

    eg: remove the records with pattern given below.

    ,,,,,,,13,Individual,Yes

2. Data merging, Deduplication, Performance Tuning & Persistance (15% Completion) – Total 35%

12. **Merge** the both header removed RDDs derived in steps 7 and 11 into an RDD namely insuredatamerged
13. **Cache** it either to memory or any other **persistence levels** you want, display only **first** few rows
14. **Calculate the count of rdds** created in step 7+11 and rdd in step 12, check whether they are matching.
15. **Remove duplicates** from this merged RDD created in step 12 and print how many duplicate rows are there.
16. **Increase the number of partitions to 8** and name it as insuredatarepart.
17. **Split the above RDD using the businessdate field** into rdd_20191001 and rdd_ 20191002 based on the BusinessDate of 2019-10-01 and 2019-10-02 respectively
18. **Store the RDDs** created in step 9, 12, 17 into HDFS locations.
19. **Convert the RDD created in step 16 above into Dataframe** namely insuredaterepartdf applying the structtype created in the step 20 given in the next usecase.

    Hint: Think of converting df to rdd then use createdataframe to apply schema some thing like this..

    Eg: createDataframe(df19.rdd,structuretype)

Part B - Spark DF & SQL (Step 3, 4 & 5)

3. DataFrames operations (20% Completion) – Total 55%

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

20. Create structuretype for all the columns as per the insuranceinfo1.csv.

    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, ensure to import sql.types library.

21. Create dataset using the csv module with option to escape ',' accessing the insuranceinfo1.csv and insuranceinfo2.csv files, apply the schema of the structure type created in the step 20.

22. Apply the below **DSL** functions in the DFs created in step 21.

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

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

    Hint : Cast to string and concat.

    c. **Remove** DentalOnlyPlan column

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

23. **Remove the rows** contains null in any one of the field and count the number of rows which contains all columns with some value.

    Hint: Use drop.na options

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

    Hint: First create the function directly and then later add inside pkg, class etc..

    a. Method should take 1 string argument and 1 return of type string

    b. Method should remove all special characters and numbers 0 to 9 - ? , / _ ( ) [ ]

    Hint: Use replaceAll function, usage of [] symbol should use \\ escape sequence.

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

25. Import the package, instantiate the class and register the method generated in step 24 as a udf for invoking in the DSL function.
26. Call the above udf in the DSL by passing NetworkName column as an argument to get the special characters removed DF.
27. Save the DF generated in step 26 in JSON into HDFS with overwrite option.
28. Save the DF generated in step 26 into CSV format with header name as per the DF and delimited by ~ into HDFS with overwrite option.
29. Save the DF generated in step 26 into hive external table and append the data without overwriting it.

4. 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:

30. **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.
31. **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:

32. **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.
33. **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 DSL function to check isnull or isnotnull to achieve the above functionality then rename and drop columns in the above 2 DFs accordingly.

Use SQL Queries:

34. **Register the above DFs as temporary views** as custview and statesview.
35. **Register the DF generated in step 22 as a tempview** namely insureview
36. **Import the package, instantiate the class and Register the method** created in step 24 in the name of remspecialcharudf using **spark udf registration**.
37. Write an **SQL query** with the below processing

    Hint: Try doing the below , step by step, skip if you can't do it, later try.

    a. Pass NetworkName to remspecialcharudf and get the new column called cleannetworkname
    b. Add current date, current timestamp fields as curdt and curts.
    c. Extract the year and month from the businessdate field and get it as 2 new fields called yr,mth respectively.
    d. Extract from the protocol either http/https from the NetworkURL column, if no protocol found then display **noprotocol.** For Eg: if http://www2.dentemax.com/ then show http if www.bridgespanhealth.com then show as **noprotocol** store in a column called protocol.
    e. 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.
38. Store the above selected Dataframe in **Parquet** formats in a HDFS location as a **single file**.
39. **Very very important interview question :** 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.

    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

40. Store the DF generated in step 39 into MYSQL table insureaggregated.

5. Visualization (5% Completion) – Total 80%

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

1. Jobs
2. Stages

   Display the below items

   a. Task locality level in the Tasks
   b. Scheduler Delay
   c. Task Deserialization Time
   d. Shuffle Read Blocked Time
   e. Shuffle Remote Reads
   f. Result Serialization Time
   g. Getting Result Time
   h. Peak Execution Memory
3. Storage
   a. Storage level
   b. Partitions info
4. Executors
5. SQL
   a. DAG
   b. AST (Abstract Syntax Tree plan)

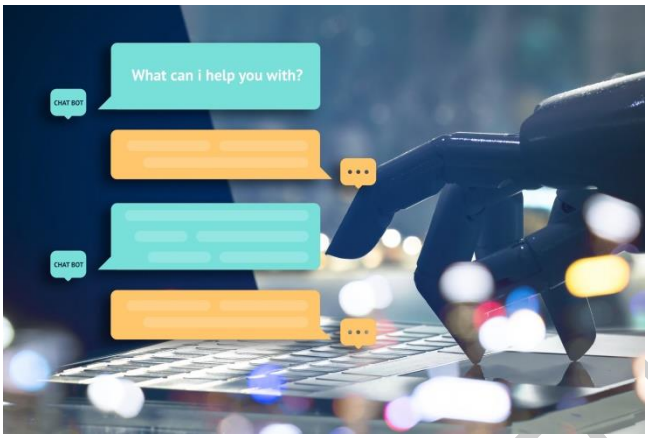
Part C - Spark Streaming with DF & SQL (Step 6)

6. Realtime Streaming (20% Completion) – Total 100%

Chat intent identification and identification of most recurring issue use case

(use Dataframe or SQL or Both), try with batch and streaming by pasting the chat in the kafka console producer and have the stopwords in the filesystem.

Note: Dataset (stopwords, chatdata) is provided in the realtimedataset folder of shared drive.



Try the below scenario with File source then try with socket or kafka streaming of the chatdata and stopwords data in the filesystem.

1. Load the chatdata into a Dataframe using ~ delimiter by pasting in kafka or streaming file source or socket with the seconds of 50.

2. Define the column names as id,chat,'type' where id is the customer who is chatting with the support agent, chat is the plain text of the chat message, type is the indicator of c means customer or a means agent.

3. Filter only the records contains 'type' as 'c' (to filter only customer interactions)

4. Remove the column 'type' from the above dataframe, hence the resultant dataframe contains only id and chat and convert to tempview.

5. Use SQL **split** function on the 'chat' column with the **delimiter** as ' ' (space) to tokenize all words for eg. if a chat looks like this (*i have internet issue from yesterday*) and convert the chat column as array type and name it as chat_tokens which has to looks like [i,have,internet,issue,from,yesterday]

id,chat_tokens

1,[i,have,internet,issue,from,yesterday]

6. Use SQL **explode** function to pivot the above data created in step 5, *then the exploded/pivoted data looks like below, register this as a tempview.*

id,chat_splits

1,i

1,have

1,internet

1,issue

1,from

1,yesterday

8. **Load the stopwords** from linux file location /home/hduser/stopwordsdir/stopwords *(A single column file found in the web that contains values such as (a,above,accross,after,an,as,at) etc. which is used in any English sentences for punctuations or as fillers)* into dataframe with the column name stopword and convert to tempview.

9. Write a **left outer join** between chat tempview created in step 6 and stopwords tempview created in step 8 and filter all nulls (or) use subquery with not in option to filter all stop words from the actual chat tempview using the stopwords tempview created above.

10. **Load** the final result into a hive table should have only result as given below using append option.

eg.

Input data in the chat file or socket stream or kafka:

1~hi~c

100~hello morning, how can i help you today~a

1~i have issues in my tv connection which is notworking~c

100~I m really sorry about that. Tell me what happened~a

1~heavy rain in the morning caused my stb and my tv is notworking~c

100~let me take your request, hopefully your issue will be resolved soon~a

Output data into hive table:

1,issues

1,tv

1,connection

1,notworking

1,heavy

1,rain

1,morning

1,caused

1,stb

1,tv

1,notworking

11. Identify the most recurring keywords used by the customer in all the chats by grouping based on the keywords used with count of keywords. use group by and count functions in the sql

for eg.

issues,2

notworking,2

tv,2

12. Store the above result in a **json** format with column names as chatkeywords, occurance

********** There you Go ☺ **********