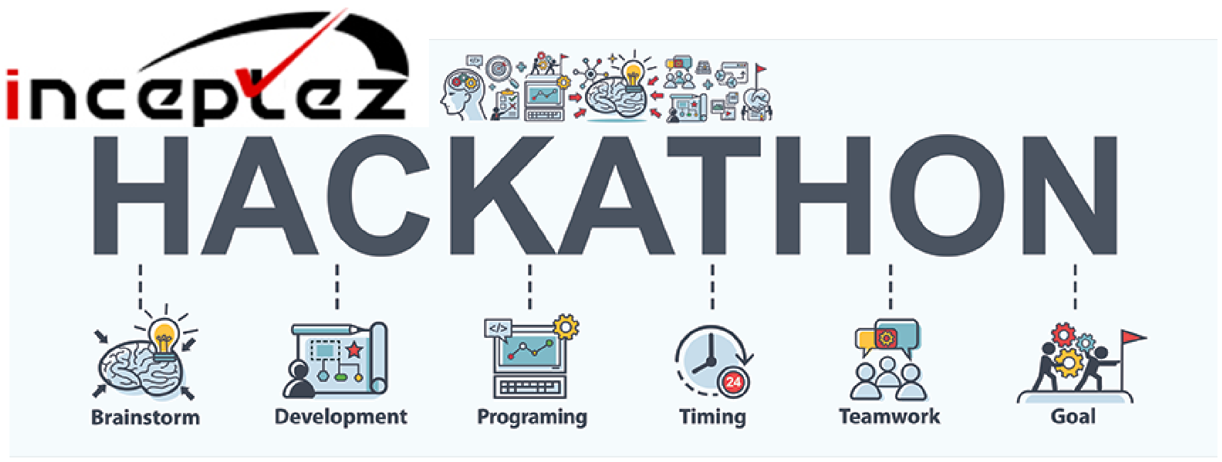


*Its time for the Challenge ….. Let us Experiment, Explore, Evolve & Excel It !!!*

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

## Hackathon Components: Spark Core, Spark DF DSL/SQL and Spark Streaming)

*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 and Realtime streaming with CDC 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 Eclipsce, you can use REPL for snippet 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.
- Provide the results as code and snapshot of results.

**Note:**

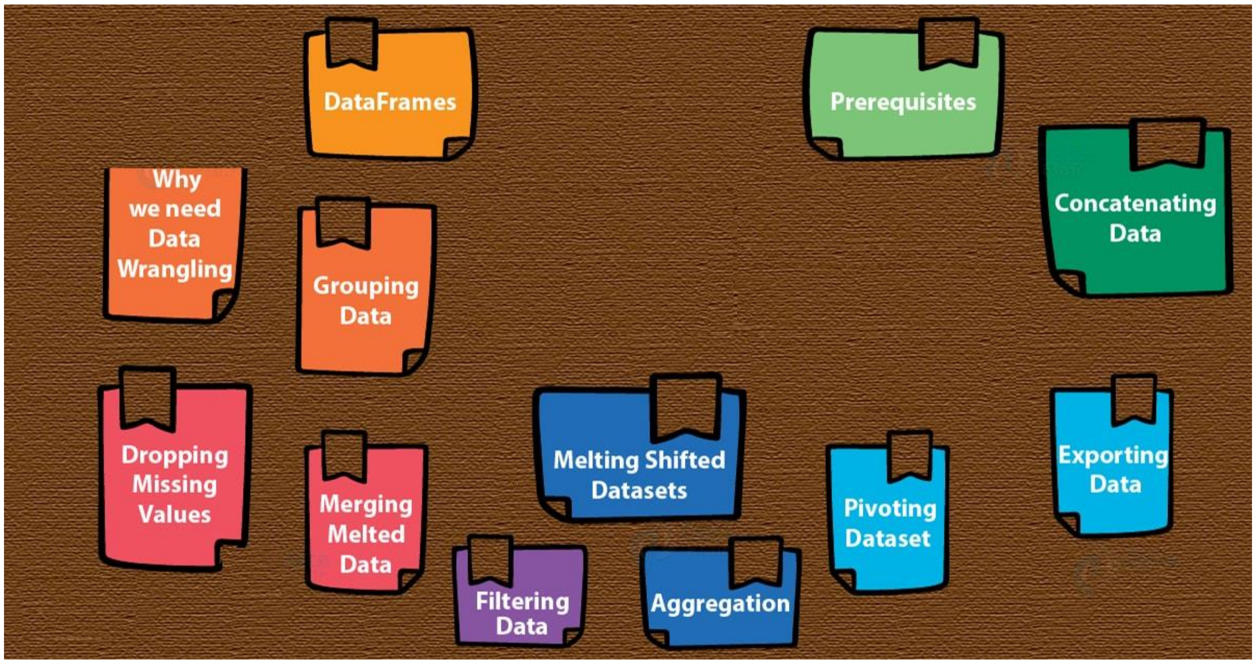
**Start with Part A - Step 1 and 2 (rdd) first**

**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 first and filter functions to achieve this by filtering the first line identified .
3. **Remove the Footer/trailer** also which contains "footer count is 404"

   Hint: Use functions like zipWithIndex, count and filter to remove the footer or use just filter to remove if you can't implement in the above way.
4. Display the **count** and **show** few rows and check whether header and footer is removed.
5. **Remove the blank lines** in the rdd.

   **Hint:** Before splitting, trim it, calculate the length and filter only row size is not = 0.
6. **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)
7. **Filter number of fields** are equal to 10 columns only - analyze why we are doing this and provide your view here..
8. **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.
9. **Take the count of the RDD** created in step 7 and step 1 and print how many rows are removed/rejected in the cleanup process of removing fields does not equals 10.

10. **Create another RDD namely rejectdata** and store the row that does not equals 10 columns. With a new column added in the first column called numcols contains number of columns in the given row **For eg - the output should be like this: (6,219) where 6 is the number of columns and 219 is the first column IssuerId of the data hence we can analyse the IssuerIds contains deficient fields.**
11. **Load** the file2 (insuranceinfo2.csv) from HDFS using textFile API into an RDD insuredata2
12. **Repeat from step 2 to 8** for this file also and create the schemaed rdd from the insuranceinfo2.csv and filter the records that contains blank or null 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%

13. **Merge** the both header and footer removed RDDs derived in steps 8 and 12 into an RDD namely insuredatamerged
14. **Persist** the step 13 RDD to memory by serializing.
15. **Calculate the count of rdds** created in step 8+12 and rdd in step 13, check whether they are matching.
16. **Remove duplicates** from this merged RDD created in step 13 and print how many duplicate rows are there.
17. **Increase the number of partitions in the above rdd to 8** and name it as insuredatarepart.
18. **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 using **Filter** function.
19. **Store the RDDs** created in step 10, 13, 18 into HDFS locations.
20. **Convert the RDD created in step 17 above into Dataframe** namely insuredaterepartdf using toDF function

## 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.**

21. Create structuretype 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, ensure to import sql.types library.

22. Create dataframes using the csv module with option to escape ‘,’ accessing the insuranceinfo1.csv and insuranceinfo2.csv files and remove the footer from both dataframes using header true, dropmalformed options and apply the schema of the structure type created in the step 21.

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

**Try the below usecases also seperately:**

i. Identify all the column names and store in an array variable – use columns function.
ii. Identify all columns with datatype and store in an array variable – use dtypes function.
iii. Identify all integer columns alone and store in an array variable.

iv. Select only the integer columns identified in the above statement and show 10 records in the screen.

24. **Take the DF created in step 23.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.

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

    **Hint:** First create the function/method 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.

26. Import the package, instantiate the class and register the method generated in step 25 as a udf for invoking in the DSL function.
27. Call the above udf in the DSL by passing NetworkName column as an argument to get the special characters removed DF.
28. Save the DF generated in step 27 in JSON into HDFS with overwrite option.
29. Save the DF generated in step 27 into CSV format with header name as per the DF and delimited by ~ into HDFS with overwrite option.
30. Save the DF generated in step 27 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:**

31. **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.
32. **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.

**<u>Use DSL functions:</u>**

33. **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.
34. **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.

**<u>Use SQL Queries:</u>**

35. **Register the above step 34 DFs as temporary views** as custview and statesview.
36. **Register the DF generated in step 23.d as a tempview** namely insureview
37. **Import the package, instantiate the class and Register the method** created in step 25 in the name of remspecialcharudf using **spark udf registration**.
38. Write an **SQL query** with the below processing – set the spark.sql.shuffle.partitions to 4
    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 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/ then **http secured** else if www.bridgespanhealth.com then show as **no protocol** 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.

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

40. 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 average age 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

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

42. *Try Package the code using maven install, take the lean and fat jar and try submit with driver memory of 512M, number of executors as 4, executor memory as 1GB and executor cores with 2 cores. Try Dynamic allocation also.*

## 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 info.
5. SQL
   a. DAG
   b. AST (Abstract Syntax Tree plan)

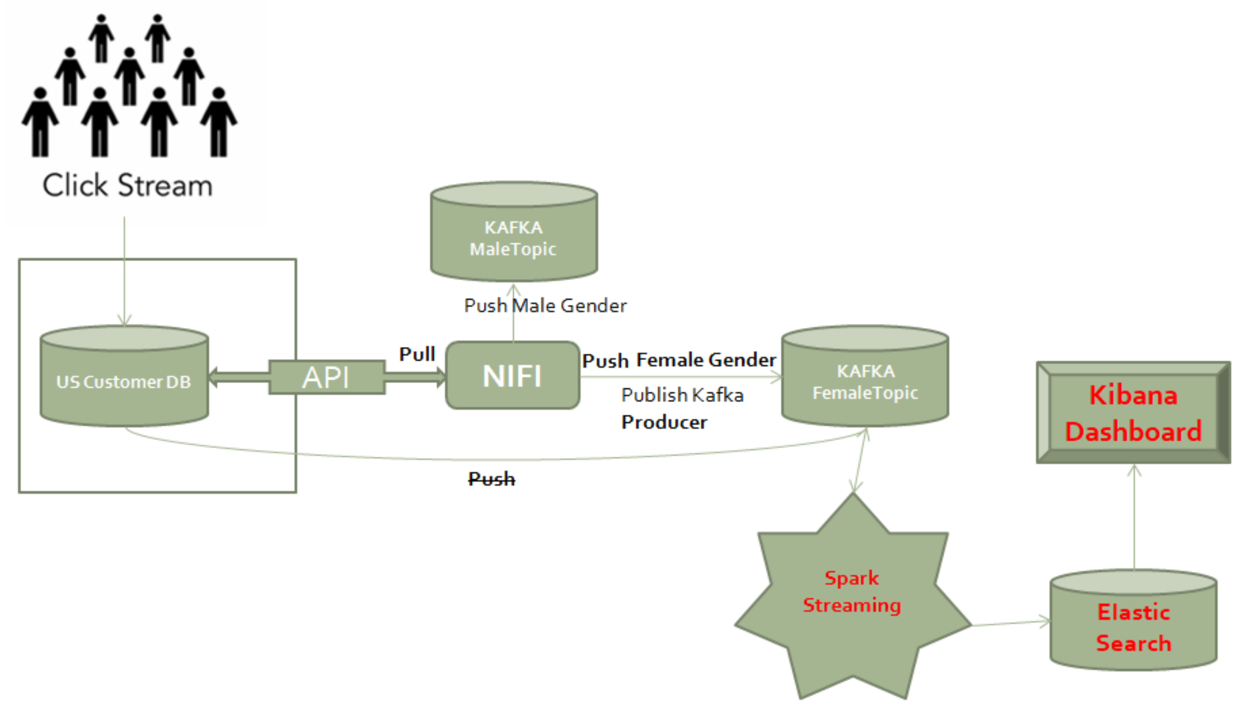
## Part C - Spark Streaming with DF & SQL (Step 6)

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

### Customer Clickstream Segmentation Analysis

**Note: The aim of the below use case is to ensure you are setting up the eclipse IDE with the given POM, so you can connect to any source/targets including cloud components in your spark code.**

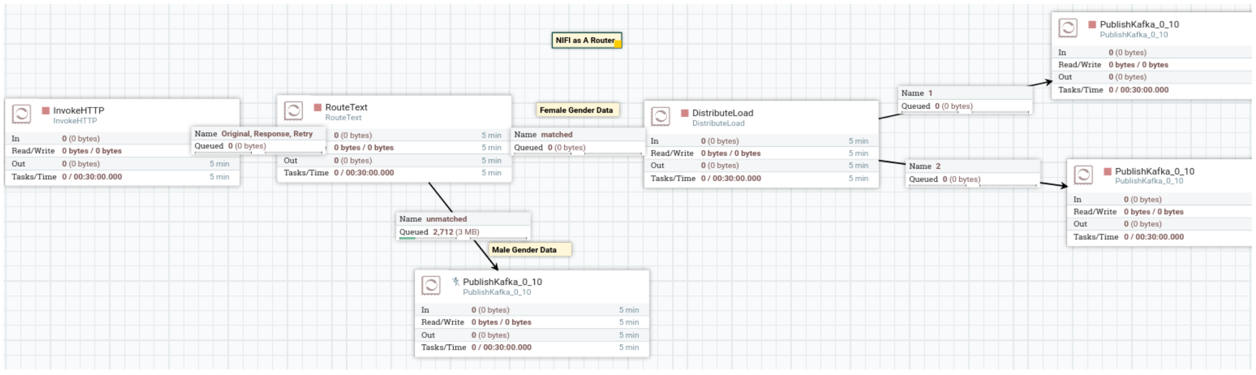
www.api.randomuser.me



**Lets go with at Realtime streaming example with the Customer Clickstream Data incrementally inserted from the source DB which is exposed as a Webservice REST API (www.api.randomuser.me)→ Hit using Nifi -> publish the events to kafka topic → fetch using spark streaming app and perform the following computation → store the result into Elastic Search and Create visualizations and dashboards as mentioned below.**

**1. Create a Nifi Workflow as per the below sample:**

**Note:** Use the nifi template xml I provided in the GDrive or refer the NIFI video shared to create the workflow in Nifi, if you are struck somewhere. Worst case, directly get the data copied from www.api.randomuser.me and paste in the Kafka console consumer.



2. Create a kafka  topic namely **custdatajson** with 2 partitions.

3. Create the below ES Index.

```
curl -XPUT 'http://localhost:9200/sparkjson' -d '

{   "settings": {

    "index": {

      "number_of_shards": "5",

      "number_of_replicas": "1"

    }

  },

  "mappings": {

    "custvisit": {

      "properties": {

        "ageflag": {

          "type": "text",

          "fields": {

            "keyword": {

              "type": "keyword",

              "ignore_above": 256

            }

          }

        },

        "age": {

          "type": "long"

        },
        "cell": {
          "type": "text",
          "fields": {
            "keyword": {
              "type": "keyword",
              "ignore_above": 256
            }
          }
        },
        "cellnumber": {
          "type": "text",
          "fields": {
            "keyword": {
              "type": "keyword",
              "ignore_above": 256
            }
          }
        },
        "coordinates": {
          "type": "geo_point",
          "fields": {
            "keyword": {
              "type": "keyword",
              "ignore_above": 256
            }
          }
        },
        "country": {
          "type": "text",
          "fields": {
            "keyword": {
              "type": "keyword",
              "ignore_above": 256
            }
          }
        },
        "curdt": {
          "type": "long"
        },
        "curts": {
          "type": "long"
        },
        "custid": {
          "type": "text",
          "fields": {
            "keyword": {
              "type": "keyword",
              "ignore_above": 256
            }
          }
        },
        "dobdt": {
          "type": "long"
        },
        "email": {
          "type": "text",
          "fields": {
            "keyword": {
              "type": "keyword",
              "ignore_above": 256
            }
          }
        },
        "first": {
          "type": "text",
          "fields": {
            "keyword": {
              "type": "keyword",
              "ignore_above": 256
            }
          }
        },
        "page": {
          "type": "long"
        },
        "state": {
          "type": "text",
          "fields": {
            "keyword": {
              "type": "keyword",
              "ignore_above": 256
            }
          }
        },
                "uscity": {
                  "type": "text",
                  "fields": {
                    "keyword": {
                      "type": "keyword",
                      "ignore_above": 256
                    }
                  }
                }
              }
            }
          }
        }'
```

4. Create a Spark streaming app (Code attached for reference) with the package org.inceptez.hack.streaming with object name as rtcustjsonanalysis to consume json data once in 10 seconds and do the followings,

**Note:** Refer the Spark Streaming code shared in the GDrive, but ensure not to copy paste fully, rather develop step by step to achieve the final code. Try with spark shell to first develop the code in a batch mode reading the data from file system with the json data copied in a file to arrive the logic mentioned below and then try with the realtime application later in eclipse.

```
spark-shell \
--conf spark.sql.hive.metastore.version=2.3.0 --conf spark.sql.hive.metastore.jars=/usr/local/hive/lib/*
```

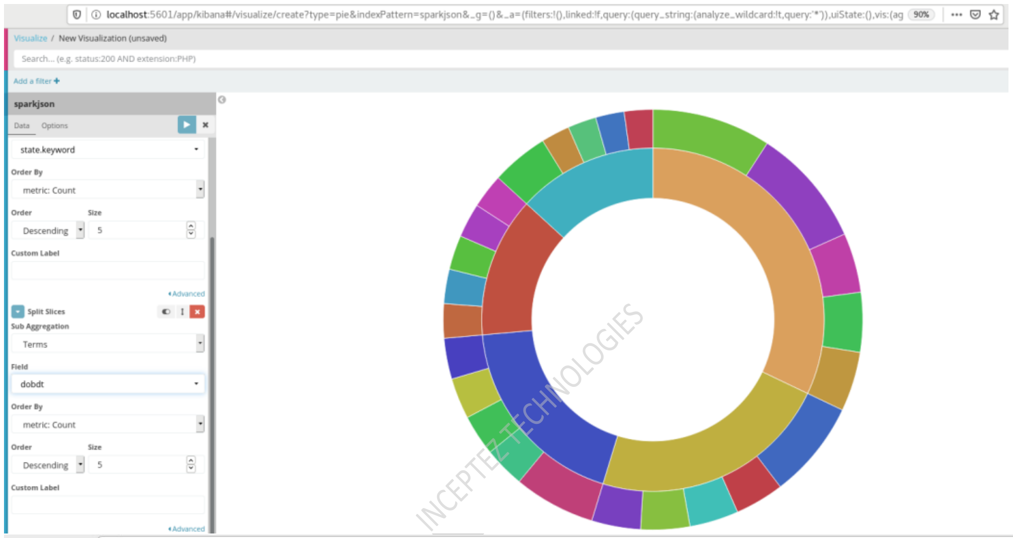
a. From the json df select the below fields. (Use SQL explode function to extract the array data)

```
explode(results) as res,
info.page as page,res.cell as cell,
res.name.first as first,
res.dob.age as age,
res.dob.date as dobdt,
res.email as email,res.location.city as uscity,res.location.coordinates.latitude as latitude,
res.location.coordinates.longitude as longitude,res.location.country as country,
res.location.state as state,
res.location.timezone as timezone,res.login.username as username,current_date as curdt,current_timestamp as curts
```

b. Filter only the records contains the age > 25
c. Write a wrapper select on top of the fields selected in step 1, username as custid, cell, first,age,email, uscity, concat(latitude,',',longitude) as coordinates, replace cell column with ()- with blank space using regexp_replace function as cellnumber,country, state, take only date portion of dobdt field in yyyy-mm-dd format as dobdt column, find the date difference in days using the built in function datediff by passing current date and dobdt as arguments and divide the output by 365 which will give the computed age from the dobdt column then find whether

valid or invalid by matching with the age field as ageflag, refer the code if you are not able to understand what is mentioned above.

d. Add the current date and timestamp fields in the last.
e. Load the above df into Elastic search and create dashboard in Kibana to analyse the country wise, city wise, age wise aggregation of data including create a coordinate map (as given below) by plotting the geo coordinates aggregation in kibana with the coordinates field generated above to see the population of customers we have across the world.
f. Try to build a fat jar with maven build and try submitting it with driver memory of 1g, number of executors as 2, executor memory with 1g, number of executors dynamically allocated with 2 to 6.


localhost:5601/app/kibana#/visualize/create?type=pie&indexPattern=sparkjson&_g=()&_a=(filters:!(),linked:!f,query:(query_string:(analyze_wildcard:!t,query:'*')),uiState:(),vis:(ag

Visualize / New Visualization (unsaved)

Search... (e.g. status:200 AND extension:PHP)

Add a filter +

sparkjson

Data Options

state.keyword

Order By: metric: Count

Order: Descending Size: 5

Custom Label

Advanced

Split Slices

Sub Aggregation: Terms

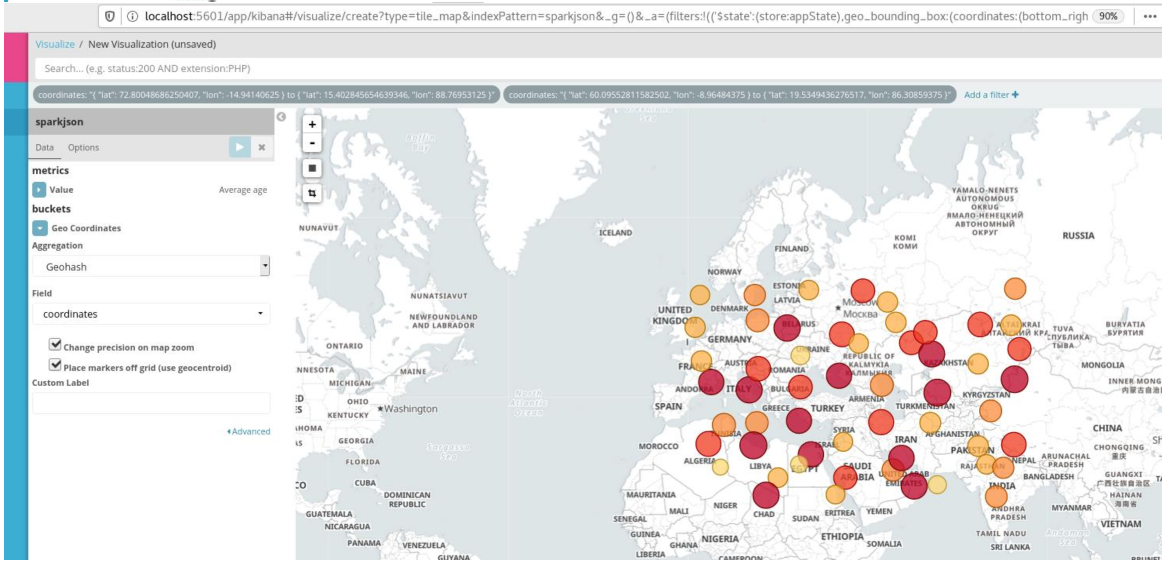
Field: dobdt

Order By: metric: Count

Order: Descending Size: 5

Custom Label

Advanced


localhost:5601/app/kibana#/visualize/create?type=tile_map&indexPattern=sparkjson&_g=()&_a=(filters:!(('$state':(store:appState),geo_bounding_box:(coordinates:(bottom_righ

Visualize / New Visualization (unsaved)

Search... (e.g. status:200 AND extension:PHP)

coordinates: "{ "lat": 72.80048686250407, "lon": -14.94140625 } to { "lat": 15.402845654639346, "lon": 88.76953125 }"

coordinates: "{ "lat": 60.09552811582502, "lon": -8.96484375 } to { "lat": 19.5349436276517, "lon": 86.30859375 }"

Add a filter +

sparkjson

Data Options

metrics

Value: Average age

buckets

Geo Coordinates

Aggregation: Geohash

Field: coordinates

[✓] Change precision on map zoom

[✓] Place markers off grid (use geocentroid)

Custom Label

Advanced

************** There you Go **************