

ANALYSIS OF MEDICAL DATA

ABSTRACT

Since the COVID-19 pandemic, there have been many medical journals. This report elucidates the analysis performed on these journals in relation to the widespread.

UNIVERSITY OF SALFORD | Ambareesh
Jonnavittula

Big Data Tools & Techniques

Section 1: Setup

1.1 System Configuration

S.No	Environment	Configuration / Version
1	Databricks	Databricks Runtime version 8.1 (Apache Spark 3.1.1, Scala 2.12)
2	Operating System	Ubuntu 18.04.5 LTS
3	Python	Python 3.8.6
4	HiveQL	Apache Spark SQL 3.0 (compatible with Apache Hive)
5	AWS S3	Based on AWS CLI 2
6	AWS Athena	Athena engine version 2 (Presto 0.172)
7	AWS EMR	EMR release 6.0.0 (HBase: 1.4.13, Hadoop 2.10.1, Hive 2.3.7)

1.2 Source Data

- <https://ai2-semanticscholar-cord-19.s3-us-west-2.amazonaws.com/2020-07-01/metadata.csv>
- <https://raw.githubusercontent.com/ambareesh-j/SJR/main/scimagojr%202019.csv>

1.3 Initial Prompts

- The keyword 'OVERWRITE' wherever used is to enable reusability.
- Sorting has been done in descending order for all the tasks except for the task 6.

Section 2: Data Cleaning and Pre-processing

2.1 Outline

- **In PySpark implementation** (Dataframes and RDD) – inclusive of exception handling
 - i. Libraries are installed – bokeh, seaborn, wordcloud.
 - ii. Widgets are created to enable user interaction with the version of the datasets.
 - iii. Variables are created using inputs from the widgets given by a user.
 - iv. Quality check for website domain presence is performed.
 - v. Underlying implementation is validated.
 - vi. Datasets are downloaded and copied into DBFS file store to enable utility.
 - vii. Availability of the datasets and the underlying data is validated.
 - viii. Dataframes for CORD19 dataset are created using
 - (1) .csv file, (2) temporary view, and (3) parquet files (with partition).
 - ix. A dataframe for Scimago Journal Rankings (SJR) dataset is created using .csv file
- **In HiveQL implementation**
 - i. Hive Input Variables are set to enable version alteration and user interaction.
 - ii. Additional hive variables are created from user inputs – database, tables, file path.
 - iii. A database is created.
 - iv. Hive data structures for CORD19 dataset are created using
 - (1) .csv file to table, (2) temporary view, and (3) parquet files to table
 - v. Hive table for SJR dataset is created using .csv file
 - vi. The tables / views created are validated
- **In AWS,**
 - i. In AWS S3, buckets are created for each of input, output and miscellaneous files.
 - ii. Cord-19 and SJR .csv files are manually imported into the input buckets.
 - iii. In AWS Athena, a database is first created.
 - iv. Tables and views are later created within the database in AWS Athena, and queried.
 - v. SQL scripts are converted to .txt format and stored in S3 miscellaneous bucket.
 - vi. In AWS EMR, a cluster is created and started. The cluster then enters ‘Waiting’ state.
 - vii. Steps are added & run based on the I/O paths, where input path is where the .txt files are present, and Output path is simply a folder inside / or an S3 bucket.

2.2 PySpark DF & RDD

2.2.1 Setting the stage

- Using %pip install command, we install the necessary libraries.

0. Module setup / Library installations

```
Cmd 3
1 # Install bokeh library for Data Visualization
2 %pip install bokeh

Show result

Cmd 4
1 # Install seaborn library for Data Visualization
2 %pip install seaborn

Show result

Cmd 5
1 # Install wordcloud library for Data Visualization
2 %pip install wordcloud

Show result
```

- Using Databricks widgets utilities, dropdown widgets were setup to pick the year, month and date of a month along with SJR_Year using ranged list comprehensions.

```
1 # Choose Year
2 dbutils.widgets.dropdown("Step 1 - Choose Year", "2020", [str(x) for x in range(2019, 2026)])
3
4 # Choose Month
5 dbutils.widgets.dropdown("Step 2 - Choose Month", "07", [str(x).zfill(2) for x in range(1, 13)])
6
7 # Choose Date
8 dbutils.widgets.dropdown("Step 3 - Choose Date", "01", [str(x).zfill(2) for x in range(1, 32)])
9
10 # Choose ScimagoJr Year
11 dbutils.widgets.dropdown("Step 4 - Choose SJR Year", "2019", [str(x) for x in range(2019, 2021)])
12
```

Step 1 - Choose Year : 2020 | Step 2 - Choose Month : 07 | Step 3 - Choose Date : 01 | Step 4 - Choose SJR Year : 2019

- Later, widget dropdown variables are used to create python variables. Source paths will be part of the input variables besides others.

```
1 #Main FilePath
2 filePath = "dbfs:/FileStore/tables/CORD19/"
3
4 #####
5 # CORD-19 dataset
6 #####
7
8 # Fragments of the URL from where we will download the dataset from:
9 url_frag1 = "https://ai2-semanticscholar-cord-19.s3-us-west-2.amazonaws.com/"
10 url_frag2 = dbutils.widgets.get("Step 1 - Choose Year") + "-" +
    dbutils.widgets.get("Step 2 - Choose Month") + "-" + dbutils.widgets.get("Step 3 -
    Choose Date")
11 url_frag3 = "/metadata.csv"
12
13 # Combined variable using all the fragments of the URL
14 metadata_url = url_frag1 + url_frag2 + url_frag3
15
16 # Proper filename given to store the data
17 md_filename = str("metadata_" + url_frag2 + ".csv").replace("-", "_")
18
19 # Display status :
20 print("Dataset name : CORD-19")
21 print("Dataset type : Web-based '.CSV' file (URL)")
22 print("URL location :", str(metadata_url).strip(), "\n")
```

```

24 #####
25 # Scimago Journal Rankings dataset
26 #####
27
28 # Instead of manual Import, we uploaded the file into GitHub to use the URL here
29 # directly
30 # Data Source - https://www.scimagojr.com/journalrank.php?page=1&total_size=30891
31 # GitHub source - https://raw.githubusercontent.com/ambareesh-j/SJR/main/scimagojr%202019.csv
32 #Fragments of the location of the downloaded/imported file of SJR URL:
33 sjr_frag1 = "https://raw.githubusercontent.com/ambareesh-j/SJR/main/scimagojr%20"
34 sjr_frag2 = dbutils.widgets.get("Step 4 - Choose SJR Year") + ".csv"
35
36 # Combined variable using all the fragments of the file location of SJR URL
37 scimago_loc = sjr_frag1 + sjr_frag2
38
39 # Proper filename given to store the data
40 sjr_filename = str("scimagojr_" + sjr_frag2)
41
42 # Extra variables (during visualization)
43 scimago_year = dbutils.widgets.get("Step 4 - Choose SJR Year")
44
45 # Display status :
46 print("Dataset name : Scimago Journal Rankings")
47 print("Dataset type : Web-based '.CSV' file (URL)")
48 print("URL location: ", str(scimago_loc).strip())

```

2.2.2 Initial Validation

- ‘Requests’ library is used to validate a website. If the status code equals 200, the website exists.

1.3.1 CORD19 dataset - Quality check

```

Cmd 13

1 import requests
2 request = requests.get(metadata_url)
3 if request.status_code == 200:
4     print('Validation complete, Website exists : ', metadata_url )
5 else:
6     raise NameError('Validation failed: Website does not exist (or) Input Year/Month/Date is invalid')

Validation complete, Website exists : https://ai2-semanticscholar-cord-19.s3-us-west-2.amazonaws.com/2020-07-01/metadata.csv
Command took 5.61 seconds -- by a.jonnavittula@edu.salford.ac.uk at 5/9/2021, 12:00:39 PM on amjon

```

1.3.2 ScimagoJr dataset - Quality check

```

Cmd 15

1 import requests
2 request = requests.get(scimago_loc)
3 if request.status_code == 200:
4     print('Validation complete, Website exists : ', scimago_loc )
5 else:
6     raise NameError('Validation failed: Website does not exist (or) Input Year ScimagoJr is invalid')

Validation complete, Website exists : https://raw.githubusercontent.com/ambareesh-j/SJR/main/scimagojr%202019.csv
Command took 1.12 seconds -- by a.jonnavittula@edu.salford.ac.uk at 5/9/2021, 12:00:39 PM on amjon

```

- Python ‘imp’ library is used to find modules.

2.1 Validate the Underlying implementation

```

Cmd 18

1 # Validate the underlying implementation using imp module in Python
2 import imp
3
4 try:
5     imp.find_module('dbutils')
6 except ImportError:
7     import pyspark
8     sc = pyspark.SparkContext()
9
10 print("Validation complete : Underlying implementation using Databricks")

<command-858541245727410>:2: DeprecationWarning: the imp module is deprecated in favour of importlib;
    see the module's documentation for alternative uses
        import imp
Validation complete : Underlying implementation using Databricks
Command took 0.02 seconds -- by a.jonnavittula@edu.salford.ac.uk at 5/9/2021, 12:00:39 PM on amjon

```

2.2.3 Data Preparation

- Environment variables are set using 'OS' library.

2.2.1 Setup environmental variables to work using Shell script

Cmd 21

```
1 import os
2
3 os.environ['fileurl'] = metadata_url
4 os.environ['fileroot'] = md_filename
5
6 os.environ['fileurl2'] = scimago_loc
7 os.environ['fileroot2'] = sjr_filename
```

- Using shell script, an existing file is removed within the local file:/tmp/ directory.
- 'curl' command-line utility is used to eliminate the redundancy of manually downloading and importing the file.

2.2.2 Download the file from the URL using path variables and store it in File:/tmp/ folder

Cmd 23

```
1 %sh
2
3 rm /tmp/$fileroot
4 curl $fileurl > /tmp/$fileroot

rm: cannot remove '/tmp/metadata_2020_07_01.csv': No such file or directory
% Total    % Received % Xferd  Average Speed   Time   Time   Current
          Dload  Upload   Total Spent   Left  Speed
0       0      0      0      0      0      0 --::-- --::-- --::-- 0
5 231M    5 12.2M  0      0  45.3M  0 0:00:05 --::-- 0:00:05 45.3M
34 231M   34 80.0M  0      0  60.5M  0 0:00:03 0:00:01 0:00:02 60.5M
72 231M   72 167M  0      0  73.8M  0 0:00:03 0:00:02 0:00:01 73.8M
100 231M  100 231M  0      0  71.9M  0 0:00:03 0:00:03 --::-- 72.0M

Command took 3.33 seconds -- by a.jonnavittula@edu.salford.ac.uk at 5/9/2021, 12:00:39 PM on amjon
```

1 %sh

```
2
3 rm /tmp/$fileroot2
4 curl $fileurl2 > /tmp/$fileroot2
```

```
rm: cannot remove '/tmp/scimagojr_2019.csv': No such file or directory
% Total    % Received % Xferd  Average Speed   Time   Time   Current
          Dload  Upload   Total Spent   Left  Speed
0       0      0      0      0      0      0 --::-- --::-- --::-- 0
46 7979k  46 3675k  0      0  6315k  0 0:00:01 --::-- 0:00:01 6305k
100 7979k 100 7979k  0      0  11.0M  0 --::-- --::-- --::-- 11.0M

Command took 0.77 seconds -- by a.jonnavittula@edu.salford.ac.uk at 5/7/2021, 10:45:15 PM on amjon
```

- In order to enable utility, we copy these files and store in DBFS.

1 # Copy the files to DBFS:/

```
2 dbutils.fs.cp("file:/tmp/" + md_filename, filepath + md_filename)
3 dbutils.fs.cp("file:/tmp/" + sjr_filename, filepath + sjr_filename)
4
5 # Find the files in DBFS:/ path
6 dbutils.fs.ls(filepath)
```

```
Out[9]: [FileInfo(path='dbfs:/FileStore/tables/CORD19/CountryKeys/', name='CountryKeys/', size=0),
 FileInfo(path='dbfs:/FileStore/tables/CORD19/metadata_2020_03_27.csv', name='metadata_2020_03_27.csv', size=69689204),
 FileInfo(path='dbfs:/FileStore/tables/CORD19/metadata_2020_05_01.csv', name='metadata_2020_05_01.csv', size=89290114),
 FileInfo(path='dbfs:/FileStore/tables/CORD19/metadata_2020_07_01.csv', name='metadata_2020_07_01.csv', size=242867644),
 FileInfo(path='dbfs:/FileStore/tables/CORD19/parq/', name='parq/', size=0),
 FileInfo(path='dbfs:/FileStore/tables/CORD19/prd', name='prd', size=242867644),
 FileInfo(path='dbfs:/FileStore/tables/CORD19/scimagojr_2019.csv', name='scimagojr_2019.csv', size=8170647)]
Command took 13.84 seconds -- by a.jonnavittula@edu.salford.ac.uk at 5/7/2021, 10:45:15 PM on amjon
```

2.2.4 Quality check

- Using python, a custom function to throw a `FileNotFoundException` is created using exception handling techniques.

```
1 # Check if this file is present in /FileStore/tables/CORD19/
2
3 # Custom function to see if the file exists in a given path
4 def file_exists(path):
5     try:
6         dbutils.fs.ls(path)
7         return True
8     except Exception as e:
9         if 'java.io.FileNotFoundException' in str(e):
10             return False
11     else:
12         raise
13
```

- 'assert' is used to check whether the file exists within the try block and an `ImportError` is thrown if an exception is encountered.

```
1 #####
2 # CORD19 Dataset
3 #####
4 try:
5     imp.find_module('dbutils')
6     assert file_exists(filepath+md_filename) == True, "ERROR - The .CSV file was
7 not correctly uploaded for CORD19 metadata - " + md_filename + ", please check
8 again and upload into /FileStore/tables/CORD19/"
9 except ImportError:
10     print("Are you running this on Databricks?")
11
12 print("Validation complete, File exists in Databricks :", filepath+md_filename)
13
14 #####
15 # ScimagoJr Dataset
16 #####
17 try:
18     imp.find_module('dbutils')
19     assert file_exists(filepath+sjr_filename) == True, "ERROR - The .CSV file was
20 not correctly uploaded for Scimago Journal Rankings - " + sjr_filename + ", please
21 check again and upload into /FileStore/tables/CORD19/"
22 except ImportError:
23     print("Are you running this on Databricks?")
24
25 print("Validation complete, File exists in Databricks :", filepath+sjr_filename)
26
```

- Before the transformation begins, both the datasets are checked for data availability within.

```
1 # Temporary RDD created
2 tmp_cord_rdd = sc.textFile(filepath+md_filename)
3
4 # Assert if the number of records present are at least 2 - a header and a line item
5 assert tmp_cord_rdd.count() >= 2, "Validation failed: Missing data in the uploaded file,
6 please recheck the URL used"
7
8 # Validation status - success message
9 print("Validation complete: There is at least 1 record in the RDD, so expecting successful
10 transformation for the file : ", "\n", filepath+md_filename)
11 del tmp_cord_rdd

> (1) SparkJob
Validation complete: There is at least 1 record in the RDD, so expecting successful transformation
for the file :
dbfs:/FileStore/tables/CORD19/metadata_2020_07_01.csv
Command took 12.05 seconds -- by a.jonnavittula@edu.salford.ac.uk at 5/7/2021, 10:45:15 PM on amjon
```

2.4.2 ScimagoJr - Underlying data validation

```
Cmd 33
1 # Temporary RDD created
2 tmp_scimago_rdd = sc.textFile(filepath+sjr_filename)
3
4 # Validate if the number of records present are at least 2 - a header and a line item
5 assert tmp_scimago_rdd.count() >= 2, "Validation failed: Missing data in the uploaded file,
6 please recheck the URL used"
7
8 # Validation status - success message
9 print("Validation complete: There is at least 1 record in the RDD, so expecting successful
10 transformation for the file : ", "\n", filepath+sjr_filename)
11 del tmp_scimago_rdd

> (1) SparkJobs
Validation complete: There is at least 1 record in the RDD, so expecting successful transformation
for the file :
dbfs:/FileStore/tables/CORD19/scimagojr_2019.csv
Command took 0.85 seconds -- by a.jonnavittula@edu.salford.ac.uk at 5/9/2021, 12:00:39 PM on amjon
```

2.2.5 Dataframes creation

- Spark read options like ‘quote’ and ‘escape’ are utilized (“\”) and the delimiter is going to be a comma, since we are using a .csv file.
- A schema is dynamically inferred for the covid19 dataset because the columns may vary in the future. This observation is in accordance with the test datasets given.

```
1 # File location and type
2 file_location = filepath+md_filename
3 file_type = "csv"
4
5 # CSV options
6 infer_schema = "true"
7 first_row_is_header = "true"
8 delimiter = ","
9
10 # The applied options are for CSV files. For other file types, these will be ignored.
11 df_cord = spark.read.format(file_type) \
12     .option("inferSchema", infer_schema) \
13     .option("header", first_row_is_header) \
14     .option("quote", "\") \
15     .option("escape", "\") \
16     .option("sep", delimiter) \
17     .load(file_location)
18
19 # Dataframe df_cord displayed
20 df_cord.printSchema()
21 display(df_cord)
```

- Whereas, for ScimagoJr dataset, we create a schema assuming the metadata of the file wouldn't alter in the future. Here the delimiter is a semicolon.

```
1 from pyspark.sql.types import IntegerType, StringType, StructType, StructField
2
3 #Creating a Schema
4 scimagoSchema = StructType([
5     StructField("Rank", IntegerType(), False),
6     StructField("Sourceid", StringType(), True),
7     StructField("Title", StringType(), True),
8     StructField("Type", StringType(), True),
9     StructField("Issn", StringType(), True),
10    StructField("SJR", IntegerType(), True),
11    StructField("SJR_Best_Quartile", StringType(), True),
12    StructField("H_index", IntegerType(), True),
13    StructField("Total_Docs_"+scimago_year, IntegerType(), True),
14    StructField("Total_Docs_3years", IntegerType(), True),
15    StructField("Total_Ref", IntegerType(), True),
16    StructField("Total_Cites_3years", IntegerType(), True),
17    StructField("Citable_Docs_3years", IntegerType(), True),
18    StructField("Cites/Doc_2years", IntegerType(), True),
19    StructField("Ref/Doc", StringType(), True),
20    StructField("Country", StringType(), True),
21    StructField("Region", StringType(), True),
22    StructField("Publisher", StringType(), True),
23    StructField("Coverage", StringType(), True),
24    StructField("Categories", StringType(), True),
25])
26
27 # File location and type
28 file_location = filepath+sjr_filename
29 file_type = "csv"
```

```

31 # CSV options
32 infer_schema = "false"
33 first_row_is_header = "true"
34 delimiter = ";"
35
36 # The applied options are for CSV files. For other file types, these will be ignored.
37 df_scimagojr = spark.read.format(file_type) \
38     .option("inferSchema", infer_schema) \
39     .option("header", first_row_is_header) \
40     .option("sep", delimiter) \
41     .schema(scimagoSchema) \
42     .load(file_location)
43
44 # .option("schema", scimagoSchema) \
45
46 df_scimagojr.printSchema()
47 display(df_scimagojr)

```

- The implementation is common until here for both Dataframes & RDD.
- In parallel, a temporary view is created which is session-scoped only.

2.5.3 Temporary Views

```

Cmd 40

1 # Temporary view
2 df_cord.createOrReplaceTempView("temp_t_cord19")
3
4 print("Temporary View: 'temp_t_cord19' has been created")
5
6 # Read the data from the Temp View into a dataframe
7 df_tempView = sqlContext.sql("SELECT * FROM temp_t_cord19")

▶ df_tempView: pyspark.sql.dataframe.DataFrame = [cord_uid: string, sha: string ... 17 more fields]
Temporary View: 'temp_t_cord19' has been created
Command took 0.25 seconds -- by a.jonnavittula@edu.salford.ac.uk at 5/9/2021, 12:00:40 PM on amjon

```

- In addition, a parquet file is created and partitioned using the dataframe. This is to enable the power of chunking and predicate push-down which allows parquet to skip over records that don't match the filter criteria. This partition, however, creates performance issues.
- ***df_cord.columns** unpacks all the list of the columns from the selected dataframe, in addition to the custom column – Published_Year
- Substring function is used to take the first 4 characters in the publish_time column to create Published_Year and the parquet file is partitioned using the same.

```

1 from pyspark.sql.functions import substring
2
3 # '*' Unpacks the elements in the list
4
5 df_cord_parq = df_cord.select(*df_cord.columns,
6                               substring(df_cord["publish_time"],1,4).alias("Published_Year")) \
7
8 df_cord_parq.write.partitionBy("Published_Year").mode("overwrite").parquet("dbfs:/FileStore/tables/CORD19/parq/t_cord19.parquet")
9
10 dbutils.fs.ls("dbfs:/FileStore/tables/CORD19/parq/t_cord19.parquet")

▶ (1) Spark Jobs
▶ df_cord_parq: pyspark.sql.dataframe.DataFrame = [cord_uid: string, sha: string ... 18 more fields]
FileInfo(path='dbfs:/FileStore/tables/CORD19/parq/t_cord19.parquet/Published_Year=1998/', name='Published_Year=1998/', size=0),
FileInfo(path='dbfs:/FileStore/tables/CORD19/parq/t_cord19.parquet/Published_Year=1999/', name='Published_Year=1999/', size=0),
FileInfo(path='dbfs:/FileStore/tables/CORD19/parq/t_cord19.parquet/Published_Year=2000/', name='Published_Year=2000/', size=0),
FileInfo(path='dbfs:/FileStore/tables/CORD19/parq/t_cord19.parquet/Published_Year=2001/', name='Published_Year=2001/', size=0),
FileInfo(path='dbfs:/FileStore/tables/CORD19/parq/t_cord19.parquet/Published_Year=2002/', name='Published_Year=2002/', size=0),
FileInfo(path='dbfs:/FileStore/tables/CORD19/parq/t_cord19.parquet/Published_Year=2003/', name='Published_Year=2003/', size=0),
FileInfo(path='dbfs:/FileStore/tables/CORD19/parq/t_cord19.parquet/Published_Year=2004/', name='Published_Year=2004/', size=0),
FileInfo(path='dbfs:/FileStore/tables/CORD19/parq/t_cord19.parquet/Published_Year=2005/', name='Published_Year=2005/', size=0),
FileInfo(path='dbfs:/FileStore/tables/CORD19/parq/t_cord19.parquet/Published_Year=2006/', name='Published_Year=2006/', size=0),
FileInfo(path='dbfs:/FileStore/tables/CORD19/parq/t_cord19.parquet/Published_Year=2007/', name='Published_Year=2007/', size=0),
FileInfo(path='dbfs:/FileStore/tables/CORD19/parq/t_cord19.parquet/Published_Year=2008/', name='Published_Year=2008/', size=0),
FileInfo(path='dbfs:/FileStore/tables/CORD19/parq/t_cord19.parquet/Published_Year=2009/', name='Published_Year=2009/', size=0),
FileInfo(path='dbfs:/FileStore/tables/CORD19/parq/t_cord19.parquet/Published_Year=2010/', name='Published_Year=2010/', size=0),
FileInfo(path='dbfs:/FileStore/tables/CORD19/parq/t_cord19.parquet/Published_Year=2011/', name='Published_Year=2011/', size=0),
FileInfo(path='dbfs:/FileStore/tables/CORD19/parq/t_cord19.parquet/Published_Year=2012/', name='Published_Year=2012/', size=0),
FileInfo(path='dbfs:/FileStore/tables/CORD19/parq/t_cord19.parquet/Published_Year=2013/', name='Published_Year=2013/', size=0),

```

- Later these parquet files are read back into a new dataframe.

```

1 df_parq = spark.read.parquet("dbfs:/FileStore/tables/CORD19/parq/t_cord19.parquet")
2
3 display(df_parq)

```

▶ (3) Spark Jobs
 ▶ df_parq: pyspark.sql.dataframe.DataFrame = [cord_uid: string, sha: string ... 18 more fields]

2.2.6 RDD creation

- RDD variables are created from the above created dataframes by using .rdd() method.

2.5.1.2 CORD19 - Create RDD from Dataframe --> cord19_rdd

Cmd 37

```

1 # RDD Creation from the DF.rdd method
2 cord19_rdd = df_cord.rdd
3 cord19_rdd.take(10)

```

▶ (1) Spark Jobs

```
Out[14]: [Row(cord_uid='ug7v899j', sha='d1aafb70c066a2068b02786f8929fd9c900897fb', source_x='PMC', title='Clinical features of culture-proven Mycoplasma pneumoniae infections at King Abdulaziz University Hospital, Jeddah, Saudi Arabia', doi='10.1186/1471-2334-1-6', pmcid='PMC35282', pubmed_id='11472636', license='no-cc', abstract='OBJECTIVE: This retrospective chart review describes the epidemiology and clinical features of 40 patients with culture-proven Mycoplasma pneumoniae infections at King Abdulaziz University Hospital, Jeddah, Saudi Arabia. METHODS: Patients with positive M. pneumoniae cultures from respiratory specimens from January 1997 through December 1998 were identified through the Microbiology records. Charts of patients were reviewed. RESULTS: 40 patients were identified, 33 (82.5%) of whom required admission. Most infections (92.5%) were community-acquired. The infection affected all age groups but was most common in infants (32.5%) and pre-school children (22.5%). It occurred year-round but was most common in the fall (35%) and spring (30%). More than three-quarters of patients (77.5%) had comorbidities. Twenty-four isolates (60%) were associated with pneumonia, 14 (35%) with upper respiratory tract infections, and 2 (5%) with bronchio
```

2.5.2.2 ScimagoJr - Create RDD from Dataframe --> scimagojr_rdd

Cmd 42

```

1 scimagojr_rdd = df_scimagojr.rdd
2 scimagojr_rdd.take(10)

```

▶ (1) Spark Jobs

```
Out[65]: [Row(Rank=1, Sourceid='28773', Title='CA - A Cancer Journal for Clinicians', Type='journal', Issn='15424863, 00079235', SJR=None, SJR_Best_Quartile='Q1', H_index=156, Total_Docs_2019=36, Total_Docs_3years=129, Total_Ref=2924, Total_Cites_3years=22644, Citable_Docs_3years=89, Cites/Doc_2years=None, Ref/Doc='81,22', Country='United States', Region='Northern America', Publisher='Wiley-Blackwell', Coverage='1950-2020', Categories='Hematology (Q1); Oncology (Q1)'), Row(Rank=2, Sourceid='19434', Title='MMWR. Recommendations and reports : Morbidity and mortality weekly report. Recommendations and reports / Centers for Disease Control', Type='journal', Issn='10575987, 15458601', SJR=None, SJR_Best_Quartile='Q1', H_index=138, Total_Docs_2019=4, Total_Docs_3years=11, Total_Ref=144, Total_Cites_3years=898, Citable_Docs_3years=11, Cites/Doc_2years=None, Ref/Doc='36,00', Country='United States', Region='Northern America', Publisher='Centers for Disease Control and Prevention (CDC)', Coverage='1990-2020', Categories='Epidemiology (Q1); Health Information Management (Q1); Health (social science) (Q1); Health, Toxicology and Mutagenesis (Q1); Medicine (miscellaneous) (Q1)'),
```

2.3 HiveQL

2.3.1 Setting the stage

- Extract date for CORD19 and year for Scimago datasets are set as hive input variables assuming that the relevant files are present in the Hive metastore file system - “`dbfs:/user/hive/warehouse/`”

The screenshot shows a Hive query editor window titled "1.1 Input Variable Declaration : CORD19 - YYYY_MM_DD & ScimagoJr - YYYY". The command line (Cmd 4) contains the following code:

```
1 -- Enter CORD19 dataset date - YYYY_MM_DD format only
2 SET cord19_date = ${Enter_CORD19_Dataset_Date} ;
3
4 -- Enter SSR dataset Year - YYYY format only
5 SET sjr_year = ${Enter_ScimagoJr_Dataset_Year} ;
```

Below the command line, there are two input fields:

- "Enter_CORD19_Dataset_Date": A dropdown menu containing "2020_07_01".
- "Enter_ScimagoJr_Dataset_Year": A dropdown menu containing "2019".

- Hive variables for the names of database, metadata_table, and scimagojr_table are set using the above created input variables.

The screenshot shows a Hive query editor window containing the following code:

```
1 -- Set the Database name
2 SET db_name = cprd ;
3
4 -- Set the CORD19 table name
5 SET metadata_table = metadata_${hivevar:cord19_date} ;
6
7 -- Set the Scimago Journal Rankings table name
8 SET scimagojr_table = scimagojr_${hivevar:sjr_year} ;
```

- Hive path variables have been created for the files present in the hive metastore.

The screenshot shows a Hive query editor window with the following command:

```
1 SET cord19_hivepath = "dbfs:/user/hive/warehouse/${hivevar:metadata_table}.csv"
```

Below the command, a table is displayed:

	key	value
1	cord19_hivepath	"dbfs:/user/hive/warehouse/metadata_2020_07_01.csv"

Text below the table: "Showing all 1 rows."

At the bottom: "Command took 0.02 seconds -- by a.jonnavittula@edu.salford.ac.uk at 5/8/2021, 6:30:15 PM on amjon2"

The screenshot shows a Hive query editor window with the following command:

```
1 SET sjr_hivepath = "dbfs:/user/hive/warehouse/${hivevar:scimagojr_table}.csv"
```

Below the command, a table is displayed:

	key	value
1	sjr_hivepath	"dbfs:/user/hive/warehouse/scimagojr_2019.csv"

Text below the table: "Showing all 1 rows."

At the bottom: "Command took 0.02 seconds -- by a.jonnavittula@edu.salford.ac.uk at 5/8/2021, 6:30:15 PM on amjon2"

2.3.2 Database Creation

- The database is initially dropped if exists to ensure reusability using CASCADE which would drop the underlying data structures as well.
- Using the ‘db_name’ variable and DB properties, a database is created and listed.

2.1 Create Database from the Hive variable : db_name

Cmd 12

2.1.1 Drop, Create & Show Database

Cmd 13

```
1 DROP DATABASE IF EXISTS ${hivevar:db_name} CASCADE;
2 CREATE DATABASE ${hivevar:db_name} WITH DBPROPERTIES (
3   ID = 001,
4   Name = 'Ambareesh Jonnavittula',
5   Type = 'admin',
6   DBType = 'Production'
7 );
8 SHOW DATABASES;
```

	databaseName
1	cprd
2	cqua
3	default

Showing all 3 rows.

- A database is properly created with a .db extension as shown in extended properties.

2.1.2 Describe the Database Properties

Cmd 15

```
1 DESCRIBE DATABASE EXTENDED ${hivevar:db_name}
```

	database_description_item	database_description_value
1	Database Name	cprd
2	Comment	
3	Location	dbfs:/user/hive/warehouse/cprd.db
4	Owner	root
5	Properties	((ID,001), (Name,Ambareesh Jonnavittula), (DBType,Production), (Type,admin))

Showing all 5 rows.



Command took 0.05 seconds -- by a.jonnavittula@edu.salford.ac.uk at 5/8/2021, 6:30:15 PM on amjon2

2.3.3 Tables / Views creation

- Using the table variables, again the tables are dropped if exists and created.
- Path is given from the hive path variable created.
- Header is present, schema is inferred, delimiter is a comma.
- Additional arguments for quote and escape are given as "\\"", which will clean the raw data.

2.2 Create permanent tables in the Database

Cmd 17

2.2.1 Create a table for cord19 dataset and rename it in the format - 'metadata_YYYY_MM_DD' within the database

Cmd 18

```
1 DROP TABLE IF EXISTS ${hivevar:db_name}.${hivevar:metadata_table};
2 CREATE TABLE ${hivevar:db_name}.${hivevar:metadata_table} USING CSV OPTIONS (
3   path ${hivevar:cord19_hivepath},
4   header "true",
5   inferSchema "true",
6   delimiter ",",
7   quote "\",
8   escape "\\\"
9 );
```

► (2) Spark Jobs

OK

Command took 7.55 seconds -- by a.jonnavittula@edu.salford.ac.uk at 5/8/2021, 6:30:15 PM on amjon2

- For Scimago dataset, header is present, schema is inferred but the delimiter is a semicolon.

2.2.2 Create a table for scimagojr dataset and rename it in the format 'scimagojr_YYYY' within the database

```
Cmd 23
1 DROP TABLE IF EXISTS ${hivevar:db_name}.${hivevar:scimagojr_table};
2 CREATE TABLE ${hivevar:db_name}.${hivevar:scimagojr_table} USING CSV OPTIONS (
3   path ${hivevar:sjr_hivepath},
4   header "true",
5   inferSchema "true",
6   delimiter ";"
7 );
8
9 ▶ (2) Spark Jobs
10 OK
11
12 Command took 1.26 seconds -- by a.jonnavittula@edu.salford.ac.uk at 5/8/2021, 6:30:15 PM on amjon2
```

- A temporary view is created/replaced using the same method as for the metadata_table which was created above.

2.3 Create Temporary view for cord19 dataset

```
Cmd 27
1 CREATE OR REPLACE TEMP VIEW temp_${hivevar:metadata_table}
2   USING CSV OPTIONS (
3     path ${hivevar:cord19_hivepath},
4     header "true",
5     inferSchema "true",
6     delimiter ",",
7     quote "\"",
8     escape "\\"
9   );
10
11 ▶ (2) Spark Jobs
12 OK
13
14 Command took 7.13 seconds -- by a.jonnavittula@edu.salford.ac.uk at 5/8/2021, 6:30:15 PM on amjon2
```

- A new table stored as parquet is created from the existing metadata_table. Only this time, we create a schema limiting the number of columns. Notice that we don't use partitioning here, which would result in better performance overall.

2.4 Parquet file to Table

```
Cmd 31
1 DROP TABLE IF EXISTS ${hivevar:db_name}.parq_${hivevar:metadata_table};
2
3 CREATE TABLE ${hivevar:db_name}.parq_${hivevar:metadata_table}
4   ( cord_uid STRING,
5     sha STRING,
6     source_x STRING,
7     title STRING,
8     doi STRING,
9     pmcid STRING,
10    pubmed_id STRING,
11    license STRING,
12    abstract STRING,
13    publish_time STRING,
14    authors STRING,
15    journal STRING,
16    url STRING
17  )
18  --  LIKE ${hivevar:db_name}.${hivevar:metadata_table}
19  STORED AS PARQUET
20  LOCATION "dbfs:/user/hive/warehouse/cord19" ;
```

OK

Command took 0.23 seconds -- by a.jonnavittula@edu.salford.ac.uk at 5/8/2021, 6:30:15 PM on amjon2

- If we don't use 'overwrite', we create duplicate values.

```

1  INSERT OVERWRITE TABLE ${hivevar:db_name}.parq_${hivevar:metadata_table}
2   SELECT cord_uid,
3         sha,
4         source_x,
5         title,
6         doi,
7         pmcid,
8         pubmed_id,
9         license,
10        abstract,
11        publish_time,
12        authors,
13        journal,
14        url
15   FROM ${hivevar:db_name}.${hivevar:metadata_table};

▶ (1) Spark Jobs
OK

Command took 19.11 seconds -- by a.jonnavittula@edu.salford.ac.uk at 5/8/2021, 6:30:15 PM on amjon2

```

2.3.4 Quality check

- Count the records and view the top 2 records to see the clean data.

2.2.1.1 Quality Check for 'metadata_table'																							
Cmd 20	<pre> 1 SELECT COUNT(*) as `Num_of_Records` FROM \${hivevar:db_name}.\${hivevar:metadata_table} ▶ (2) Spark Jobs <table border="1"> <thead> <tr> <th>Num_of_Records</th> </tr> </thead> <tbody> <tr> <td>1 169821</td> </tr> </tbody> </table> Showing all 1 rows. <table border="1"> <thead> <tr> <th colspan="5"></th> </tr> </thead> <tbody> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </tbody> </table> Command took 4.68 seconds -- by a.jonnavittula@edu.salford.ac.uk at 5/8/2021, 6:30:15 PM on amjon2 </pre>	Num_of_Records	1 169821																				
Num_of_Records																							
1 169821																							
	<pre> 1 SELECT * 2 * 3 FROM 4 \${hivevar:db_name}.\${hivevar:metadata_table}; 5 LIMIT 6 2; ▶ (1) Spark Jobs <table border="1"> <thead> <tr> <th>cord_uid</th> <th>sha</th> <th>source_x</th> <th>title</th> </tr> </thead> <tbody> <tr> <td>ug7v899j</td> <td>d1aafb70c066a2068b02786f8929fd9c900897fb</td> <td>PMC</td> <td>Clinical features of culture-proven Mycoplasma pneumoniae Arabia</td> </tr> <tr> <td>02tnwd4m</td> <td>6b0567729c2143a66d737eb0a2f53f2dce2e5a7d</td> <td>PMC</td> <td>Nitric oxide: a pro-inflammatory mediator in lung disease?</td> </tr> </tbody> </table> Showing all 2 rows. <table border="1"> <thead> <tr> <th colspan="5"></th> </tr> </thead> <tbody> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </tbody> </table> </pre>	cord_uid	sha	source_x	title	ug7v899j	d1aafb70c066a2068b02786f8929fd9c900897fb	PMC	Clinical features of culture-proven Mycoplasma pneumoniae Arabia	02tnwd4m	6b0567729c2143a66d737eb0a2f53f2dce2e5a7d	PMC	Nitric oxide: a pro-inflammatory mediator in lung disease?										
cord_uid	sha	source_x	title																				
ug7v899j	d1aafb70c066a2068b02786f8929fd9c900897fb	PMC	Clinical features of culture-proven Mycoplasma pneumoniae Arabia																				
02tnwd4m	6b0567729c2143a66d737eb0a2f53f2dce2e5a7d	PMC	Nitric oxide: a pro-inflammatory mediator in lung disease?																				

- Temp view and Parquet table

2.3.1 Quality Check for temp view 'metadata_table'																	
Cmd 29	<pre> 1 SELECT COUNT(*) as `Num_of_Records` FROM temp_\${hivevar:metadata_table} ▶ (2) Spark Jobs <table border="1"> <thead> <tr> <th>Num_of_Records</th> </tr> </thead> <tbody> <tr> <td>1 169821</td> </tr> </tbody> </table> Showing all 1 rows. <table border="1"> <thead> <tr> <th colspan="5"></th> </tr> </thead> <tbody> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </tbody> </table> </pre>	Num_of_Records	1 169821														
Num_of_Records																	
1 169821																	
	<table border="1"> <thead> <tr> <th colspan="2">2.4.1 Quality Check for parquet table created</th> </tr> </thead> <tbody> <tr> <td>Cmd 34</td> <td> <pre> 1 SELECT COUNT(*) as `Num_of_Records` FROM \${hivevar:db_name}.parq_\${hivevar:metadata_table} ▶ (2) Spark Jobs <table border="1"> <thead> <tr> <th>Num_of_Records</th> </tr> </thead> <tbody> <tr> <td>1 169821</td> </tr> </tbody> </table> Showing all 1 rows. <table border="1"> <thead> <tr> <th colspan="5"></th> </tr> </thead> <tbody> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </tbody> </table> Command took 1.18 seconds -- by a.jonnavittula@edu.salford.ac.uk at 5/8/2021, 6:30:15 PM on amjon2 </pre> </td> </tr> </tbody> </table>	2.4.1 Quality Check for parquet table created		Cmd 34	<pre> 1 SELECT COUNT(*) as `Num_of_Records` FROM \${hivevar:db_name}.parq_\${hivevar:metadata_table} ▶ (2) Spark Jobs <table border="1"> <thead> <tr> <th>Num_of_Records</th> </tr> </thead> <tbody> <tr> <td>1 169821</td> </tr> </tbody> </table> Showing all 1 rows. <table border="1"> <thead> <tr> <th colspan="5"></th> </tr> </thead> <tbody> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </tbody> </table> Command took 1.18 seconds -- by a.jonnavittula@edu.salford.ac.uk at 5/8/2021, 6:30:15 PM on amjon2 </pre>	Num_of_Records	1 169821										
2.4.1 Quality Check for parquet table created																	
Cmd 34	<pre> 1 SELECT COUNT(*) as `Num_of_Records` FROM \${hivevar:db_name}.parq_\${hivevar:metadata_table} ▶ (2) Spark Jobs <table border="1"> <thead> <tr> <th>Num_of_Records</th> </tr> </thead> <tbody> <tr> <td>1 169821</td> </tr> </tbody> </table> Showing all 1 rows. <table border="1"> <thead> <tr> <th colspan="5"></th> </tr> </thead> <tbody> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </tbody> </table> Command took 1.18 seconds -- by a.jonnavittula@edu.salford.ac.uk at 5/8/2021, 6:30:15 PM on amjon2 </pre>	Num_of_Records	1 169821														
Num_of_Records																	
1 169821																	

- Scimago table

2.2.2.1 Quality Check for {scimagojr_table}

```
Cmd 25
1 SELECT
2 *
3 FROM
4   ${hivevar:db_name}.${hivevar:scimagojr_table}
5 LIMIT
6   2;
```

► (1) Spark Jobs

Rank	Sourceid	Title	Type	Issn
1	28773	CA - A Cancer Journal for Clinicians	journal	15424863, 00079235
2	19434	MMWR. Recommendations and reports : Morbidity and mortality weekly report. Recommendations and reports / Centers for Disease Control	journal	10575987, 15458601

Showing all 2 rows.

- Show the tables & views to check availability.

```
1 SHOW TABLES
2 FROM
3 ${hivevar:db_name};
```

database	tableName	isTemporary
1 cprd	metadata_2020_07_01	false
2 cprd	parq_metadata_2020_07_01	false
3 cprd	scimagojr_2019	false

Showing all 7 rows.


```
1 SHOW VIEWS LIKE 'temp_metadata*'
```

namespace	viewName	isTemporary
1	temp_metadata_2020_03_27	true
2	temp_metadata_2020_05_01	true
3	temp_metadata_2020_07_01	true

Showing all 3 rows.

2.4 AWS S3 & Athena

- In AWS S3, a new bucket – Jonnavittula.inputs-bucket is created. A folder ‘Prd1’ contains the cord19 dataset.

The screenshot shows the AWS S3 console interface. The left sidebar is collapsed. The main area shows the contents of the 'Prd1' folder within the 'jonnnavittula.inputs-bucket'. The 'Objects' tab is selected. There is one object listed:

Name	Type	Last modified	Size	Storage class
metadata_2020_07_01.csv	csv	April 25, 2021, 20:01:30 (UTC+01:00)	231.6 MB	Standard

- Similarly, scimago dataset is present within another folder ‘SJR’.

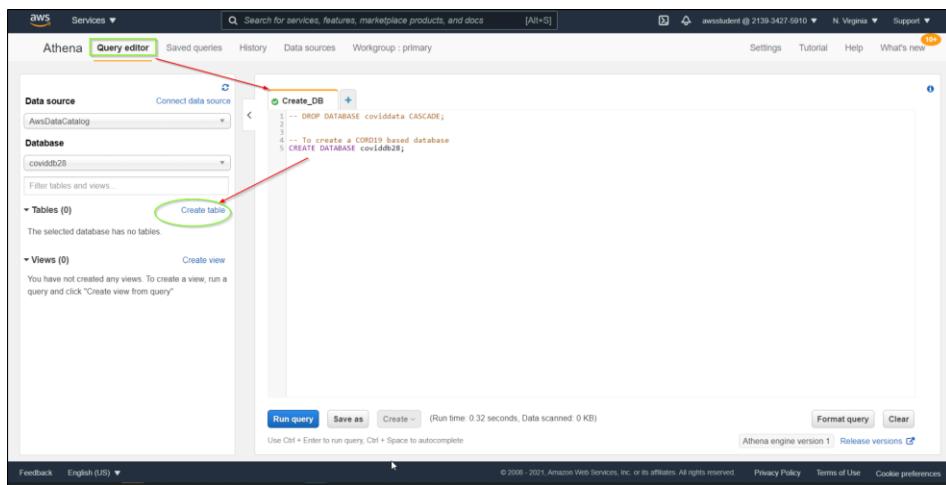
The screenshot shows the AWS S3 console interface. The left sidebar is collapsed. The main area shows the contents of the 'SJR' folder within the 'jonnnavittula.inputs-bucket'. The 'Objects' tab is selected. There is one object listed:

Name	Type	Last modified	Size	Storage class
scimagoj_2019.csv	csv	April 25, 2021, 20:01:53 (UTC+01:00)	7.8 MB	Standard

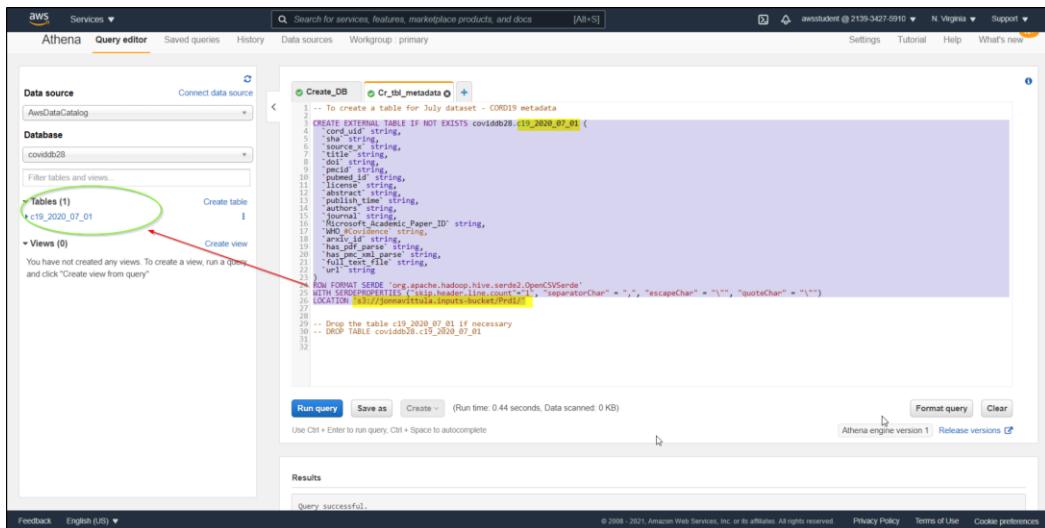
- A new S3 bucket for outputs is created – ‘s3://jonnnavittula.outputs-bucket/’ which will store our query result location within Athena. Query result location within Athena is given.

The screenshot shows the 'Settings' page for Athena. The 'Query result location and encryption' section is visible. The 'Query result location' field is highlighted with a green box and contains the value 's3://jonnnavittula.outputs-bucket/'. Below the field, a note says 'The S3 path requires a trailing slash.' There are also options for 'Encrypt query results' and 'Autocomplete'.

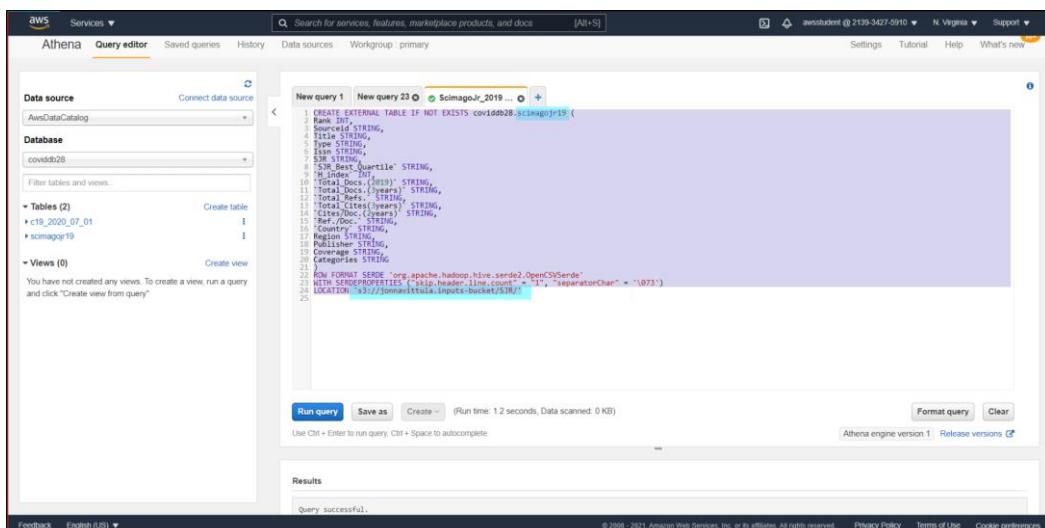
- In Athena Query editor, we proceed with the following steps –
 1. Create Database, tables and views.



- For metadata_table, we skip the header line, separatorChar is a comma, escape and quote characters are “\””



- Whereas for scimago, Presto accepts the ascii value of semi-colon '\073' and executes as expected.



- A view for raw list of authors is created by replacing the commas with a pipe for data cleaning purposes in Presto SQL which is different to HiveQL, followed by a split on semicolon.
 - Later a cleaned list of authors is created from the raw authors view using Presto's CROSS JOIN UNNEST which is an equivalent to LATERAL VIEW in HiveQL. Here 'WITH dataset as ()' enables the subquery functionality.

The screenshot shows the AWS Athena Query Editor interface. The left sidebar displays the Data source (AwsDataCatalog) and Database (coviddb28). Under Views, three items are listed: authors_cleaned (circled in green), authors_raw (circled in green), and covid_task1. A red arrow points from the 'authors_cleaned' view to the 'authors_cleaned' section in the main query editor. Another red arrow points from the 'authors_raw' view to the 'authors_raw' section in the main query editor. The main area contains a multi-line SQL script:

```
1 DROP VIEW IF EXISTS Authors_list ;
2
3 -- View 1 - Authors split on ","
4 CREATE OR REPLACE VIEW Authors_list AS
5 SELECT explode(split(trim(authors, ','), ',')) as authors_new
6 FROM "coviddb28"."c19_2020_07_01"
7
8
9 -- View 2 - Authors Exploded View
10 CREATE OR REPLACE VIEW Authors_Cleaned AS
11 WITH dataset AS
12 (
13     SELECT journal, authors_new FROM "coviddb28"."Authors_Raw"
14 )
15 SELECT journal,
16        REPLACE(TRIM(Author_Piped), ',', '') as Author
17 FROM dataset
18        CROSS JOIN UNNEST(authors_raw) as t(Author_Piped)
19
20
21
```

At the bottom, there are buttons for Run query, Save as, Create, Format query, and Clear. The status bar indicates a run time of 0.65 seconds and 0 KB scanned.

2.5 AWS S3 & EMR

- SQL files converted as .txt files and uploaded into an S3 bucket.

Name	Type	Last modified	Size	Storage class
01 - Create DB.txt	txt	April 27, 2021, 12:31:57 (UTC+01:00)	72.0 B	Standard
02 - Create Table cord19 metadata.txt	txt	April 27, 2021, 12:31:58 (UTC+01:00)	750.0 B	Standard
03 - Create Table scimagojr.txt	txt	April 27, 2021, 12:31:59 (UTC+01:00)	664.0 B	Standard
04 - Create view for Authors.txt	txt	April 27, 2021, 12:31:50 (UTC+01:00)	611.0 B	Standard
05 - Insert Data for task 1.txt	txt	April 27, 2021, 12:31:51 (UTC+01:00)	232.0 B	Standard
06 - Insert Data for task 2.txt	txt	April 27, 2021, 13:20:40 (UTC+01:00)	388.0 B	Standard
07 - Insert Data for task 3.txt	txt	April 27, 2021, 13:20:40 (UTC+01:00)	405.0 B	Standard
08 - Insert Data for task 4.txt	txt	April 27, 2021, 13:20:41 (UTC+01:00)	352.0 B	Standard
09 - Insert Data for task 5.txt	txt	April 27, 2021, 13:20:41 (UTC+01:00)	484.0 B	Standard
10 - Insert Data for task 6.txt	txt	April 27, 2021, 13:20:42 (UTC+01:00)	479.0 B	Standard

- All the scripts we used in Athena are used here in .txt format.
- Locations \${INPUT} & \${OUTPUT} variables refer to the values given in the EMR cluster steps.

```
DROP DATABASE IF EXISTS coviddb28 CASCADE;
CREATE DATABASE coviddb28;
```

```
DROP TABLE IF EXISTS coviddb28.c19_2020_07_01;
CREATE EXTERNAL TABLE coviddb28.c19_2020_07_01 (
    `cord_uid` string,
    `sha` string,
    `source_x` string,
    `title` string,
    `doi` string,
    `pmcid` string,
    `pubmed_id` string,
    `license` string,
    `abstract` string,
    `publish_time` string,
    `authors` string,
    `journal` string,
    `Microsoft_Academic_Paper_ID` string,
    `WHO_CovidEvidence` string,
    `arxiv_id` string,
    `has_pdf_parse` string,
    `has_pmc_xml_parse` string,
    `full_text_file` string,
    `url` string
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES ("skip.header.line.count"="1", "separatorChar" = ",", "escapeChar" = "\"", "quoteChar" = "\"")
LOCATION '${INPUT}';
```

03 - Create Table scimagojr.txt - Notepad

```

File Edit Format View Help
DROP TABLE IF EXISTS coviddb28.scimagojr19;

CREATE EXTERNAL TABLE coviddb28.scimagojr19 (
`Rank` INT,
`Sourceid` STRING,
`Title` STRING,
`Type` STRING,
`Issn` STRING,
`SJR` STRING,
`SJR_Best_Quartile` STRING,
`H_index` INT,
`Total_Docs(2019)` STRING,
`Total_Docs(3years)` STRING,
`Total_Ref` STRING,
`Total_Cites(3years)` STRING,
`Cites/Doc(2years)` STRING,
`Ref/Doc` STRING,
`Country` STRING,
`Region` STRING,
`Publisher` STRING,
`Coverage` STRING,
`Categories` STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES ("skip.header.line.count" = "1", "separatorChar" = '\073')
LOCATION '${INPUT}'

```

04 - Create view for Authors.txt - Notepad

```

File Edit Format View Help
DROP VIEW IF EXISTS coviddb28.Authors_Raw ;
DROP VIEW IF EXISTS coviddb28.Authors_Cleaned ;

-- View 1 - Authors split on ";"
CREATE OR REPLACE VIEW coviddb28.Authors_Raw AS
SELECT journal, SPLIT(REPLACE(authors, ',', '|'), ';') as authors_raw
FROM coviddb28.c19_2020_07_01 ;

-- View 2 - Authors Exploded View
CREATE OR REPLACE VIEW coviddb28.Authors_Cleaned AS
WITH dataset AS
(
    SELECT journal, authors_raw FROM coviddb28.Authors_Raw
)
SELECT journal,
       REPLACE(TRIM(Author_Piped), '|', ',') as Author
FROM dataset
LATERAL VIEW EXPLODE(authors_raw) t as Author_Piped
;

```

- Output bucket was kept ready with the necessary folders for tasks.

The screenshot shows the AWS S3 console with the path `Amazon S3 > jonnavitulla.outputs-bucket > AWS EMR - CORD19 tasks/`. The 'Objects' tab is selected. A table lists 12 items:

Name	Type	Last modified	Size	Storage class
T1_Frequent_Journals_\$folder\$	Folder	April 27, 2021, 02:28:39 (UTC+01:00)	0 B	Standard
T1_Frequent_Journals/	Folder	-	-	-
T2_Avg_Abstract_Lenghts_\$folder\$	Folder	April 27, 2021, 13:22:52 (UTC+01:00)	0 B	Standard
T2_Avg_Abstract_Lenghts/	Folder	-	-	-
T3_Papers_with_Highest_Authors_\$folder\$	Folder	April 27, 2021, 13:27:35 (UTC+01:00)	0 B	Standard
T3_Papers_with_Highest_Authors/	Folder	-	-	-
T4_Profile_Authors_\$folder\$	Folder	April 27, 2021, 02:35:45 (UTC+01:00)	0 B	Standard
T4_Profile_Authors/	Folder	-	-	-
T5_Author_H_Indexes_\$folder\$	Folder	April 27, 2021, 13:35:10 (UTC+01:00)	0 B	Standard
T5_Author_H_Indexes/	Folder	-	-	-
T6_Num_of_Papers_since_2020_\$folder\$	Folder	April 27, 2021, 13:49:21 (UTC+01:00)	0 B	Standard
T6_Num_of_Papers_since_2020/	Folder	-	-	-

- An EMR cluster was created with the applications – Hive, HBase, Hadoop, and instance type is m5.xlarge.

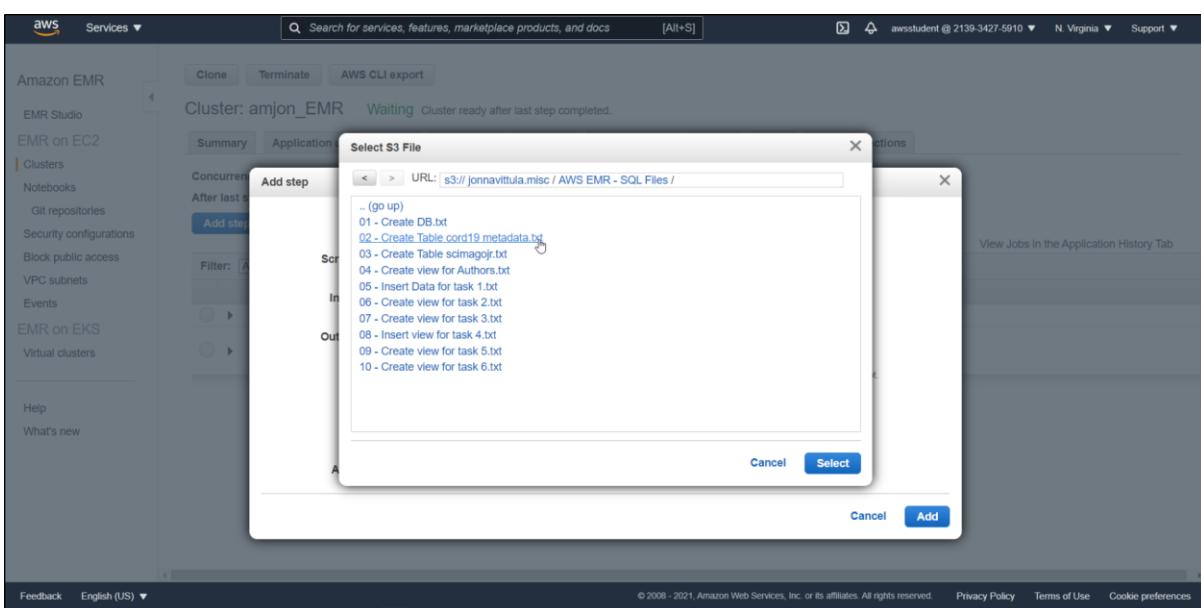
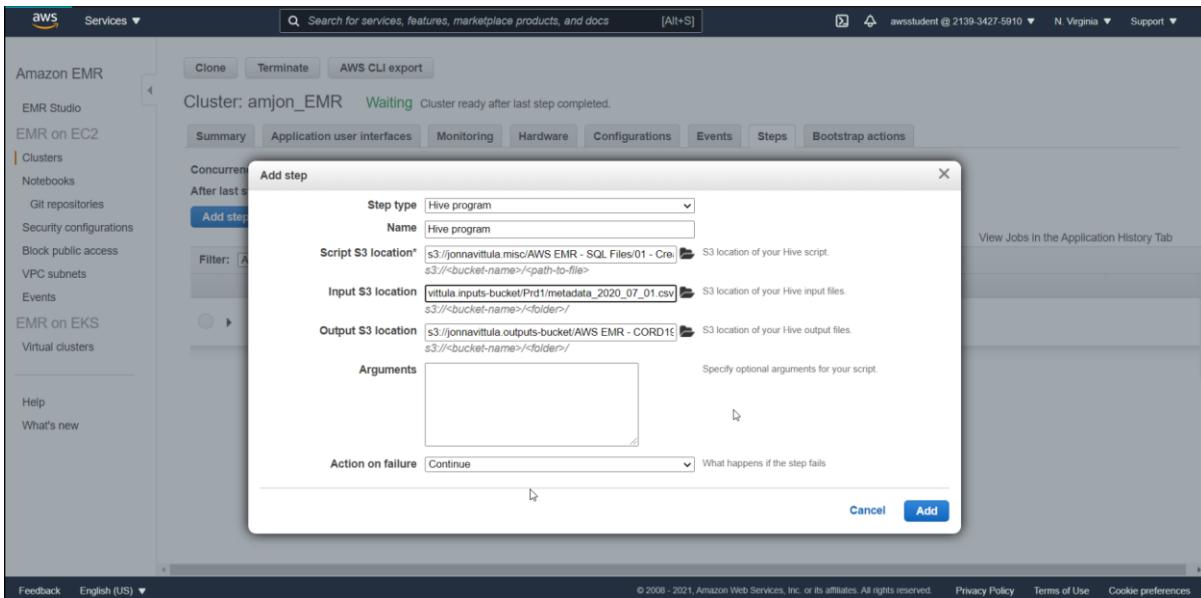
The screenshot shows the 'Create Cluster - Quick Options' page. Under 'General Configuration', the cluster name is set to `amjn_EMR`, the logging destination is `s3://aws-logs-213934275910-us-east-1/elasticsearch`, and the launch mode is set to 'Cluster'. Under 'Software configuration', the release is set to `emr-5.33.0` and the applications selected are HBase, Presto, and Spark. Under 'Hardware configuration', the instance type is set to `m5.xlarge` and the number of instances is set to 3.

- EC2 key pair is not needed in this case.

The screenshot shows the 'Create Cluster' page with detailed configuration. Under 'Applications', HBase is selected. Under 'Hardware configuration', the instance type is `m5.xlarge` and the number of instances is 3. Under 'Security and access', the EC2 key pair is set to 'Proceed without an EC2 key pair', permissions are set to 'Default', and the EMR role is `EMR_DefaultRole`. The EC2 instance profile is `EMR_EC2_DefaultRole`. At the bottom right is a 'Create cluster' button.

- Cluster remains in Waiting state until a step is performed.

- Added steps with Hive program type, script location path and I/O paths.



Section 3: Solutions given for Problems / Tasks

- The implementation was done in 3 different ways for the PySpark dataframes & HiveQL tables created from csv, temp view and parquet file, and all the results match.
- In HiveQL, it is assumed that the .csv files are already uploaded into Databricks file system – Hive metastore. To refer the tables/views, hive variables were used.
- In RDD, Lambda/anonymous functions are used while computing transformations.

3.1: Task 1 - Find the 5 most common journals

3.1.0 Outline

1. Journal column was selected excluding the nulls
2. The journals were grouped by their individual total count and sorted.
3. Aliases were applied and top 5 records were shown.

3.1.1 Assumptions made

- The top 5 journals are read by a readers' market.

3.1.2 PySpark – Dataframes

1. Nulls were dropped using `.na.drop()` method.
2. `.withColumnRenamed()` was used to give alias in Dataframes.

```
3.1 Using Dataframe created from CSV

Cmd 46
1 task1_df = df_cord.select("journal") \
2     .na.drop() \
3     .groupBy("journal") \
4     .count() \
5     .sort('count', ascending=False) \
6     .withColumnRenamed("journal", "Name of the Journal") \
7     .withColumnRenamed("count", "Frequency")
8
9 task1_df.show(5)
```

```
3.2 Using Dataframe created from Temp View

Cmd 48
1 # Perform the task
2 df_temp1 = df_tempView.select("journal") \
3     .na.drop() \
4     .groupBy("journal") \
5     .count() \
6     .sort('count', ascending=False) \
7     .withColumnRenamed("journal", "Name of the Journal") \
8     .withColumnRenamed("count", "Frequency")
9
10 df_temp1.show(5)
```

```
3.3 Using Dataframe created from Parquet Files

Cmd 50
1 # Perform the task
2 df_parqt1 = df_parq.select("journal") \
3     .na.drop() \
4     .groupBy("journal") \
5     .count() \
6     .sort('count', ascending=False) \
7     .withColumnRenamed("journal", "Name of the Journal") \
8     .withColumnRenamed("count", "Frequency")
9
10 df_parqt1.show(5)
```

3.1.3 HiveQL

1. Aliasing is done in SQL databricks using AS keyword
2. Spaces in the field names are handled using backtick / acute (`)

3.1 Using Permanent Tables

```
Cmd 44
1 SELECT
2   journal AS `Name of the Journal`,
3   count(journal) AS Frequency
4 FROM
5   ${hivevar:db_name}.${hivevar:metadata_table}
6 GROUP BY
7   journal
8 ORDER BY
9   Frequency DESC
10 LIMIT
11 5
```

3.2 Using Temp View

```
Cmd 46
1 SELECT
2   journal AS `Name of the Journal`,
3   count(journal) AS Frequency
4 FROM
5   temp_${hivevar:metadata_table}
6 GROUP BY
7   journal
8 ORDER BY
9   Frequency DESC
10 LIMIT
11 5
```

3.3 Using Table from Parquet File

```
Cmd 48
1 SELECT
2   journal AS `Name of the Journal`,
3   count(journal) AS Frequency
4 FROM
5   ${hivevar:db_name}.parq_${hivevar:metadata_table}
6 GROUP BY
7   journal
8 ORDER BY
9   Frequency DESC
10 LIMIT
11 5
```

3.1.4 PySpark – RDD

1. Filter out the nulls from Journal column using 'is not None' condition.
2. Key-value pairs are created for journal and 1, and reduced to count the journal repeats.

```
1 task1_rdd = cord19_rdd.filter(lambda row: row.journal is not None) \
2   .map(lambda row: (row.journal,1)) \
3   .reduceByKey(lambda v1,v2 : v1+v2) \
4   .sortBy(lambda line: line[1], ascending=False)
5
6 task1_rdd.take(5)
```

3.1.5 AWS

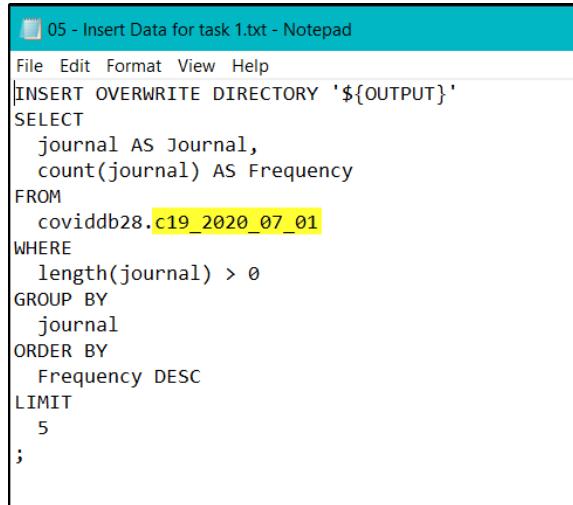
3.1.5.1 AWS S3 & Athena

```

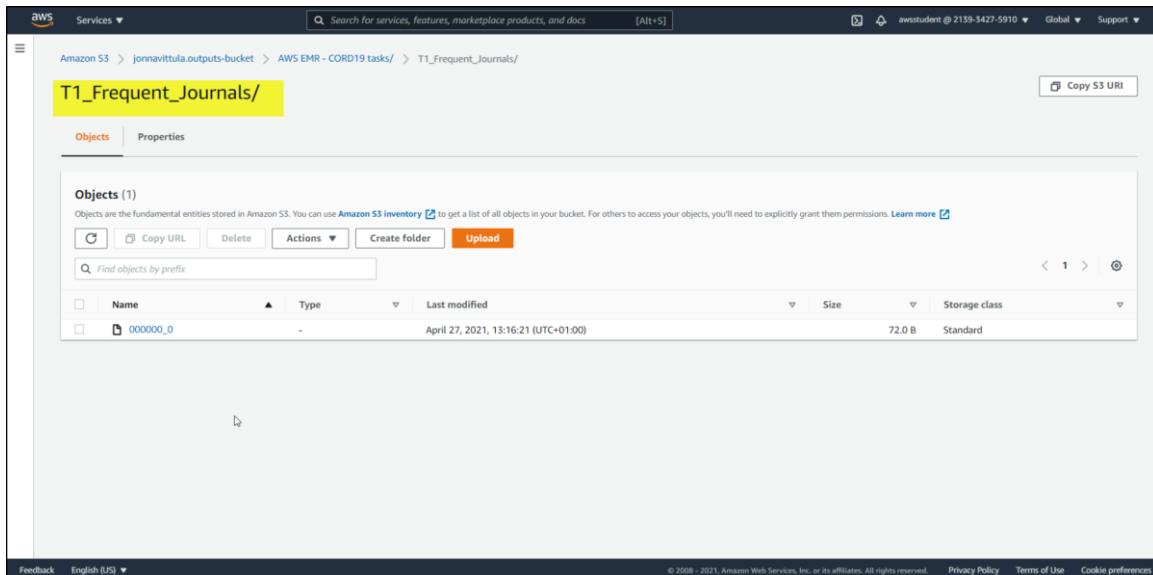
New query 1 | New query 23 | ScimagoJr_2019 ... | Task 1 | New query 30 | Authors View Cr ...
1 -- Task 1, Find the 5 most common journals, list them along with their frequencies.
2 CREATE OR REPLACE VIEW covid_task1 AS
3 SELECT
4   journal AS Journal,
5   count(journal) AS Frequency
6 FROM
7   coviddb28.covid_task1
8 WHERE
9   journal != ''
10 GROUP BY
11   journal
12 ORDER BY
13   Frequency DESC
14 LIMIT
15   5
16
17
18
Run query | Save as | Create | (Run time: 0.45 seconds, Data scanned: 0 KB)
Use Ctrl + Enter to run query, Ctrl + Space to autocomplete
Format query | Clear
Athena engine version 1 | Release versions
Feedback | English (US) | © 2006 - 2021 Amazon Web Services, Inc. or its affiliates. All rights reserved. | Privacy Policy | Terms of Use | Cookie preferences

```

3.1.5.2 AWS S3 & EMR



```
05 - Insert Data for task 1.txt - Notepad
File Edit Format View Help
INSERT OVERWRITE DIRECTORY '${OUTPUT}'
SELECT
    journal AS Journal,
    count(journal) AS Frequency
FROM
    coviddb28.c19_2020_07_01
WHERE
    length(journal) > 0
GROUP BY
    journal
ORDER BY
    Frequency DESC
LIMIT
    5
;
```



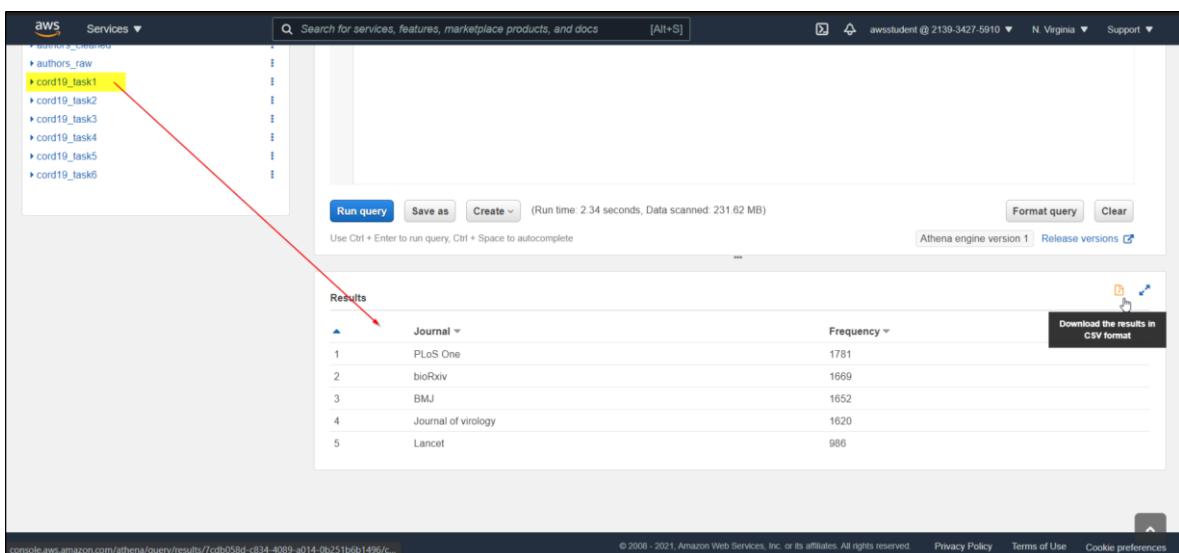
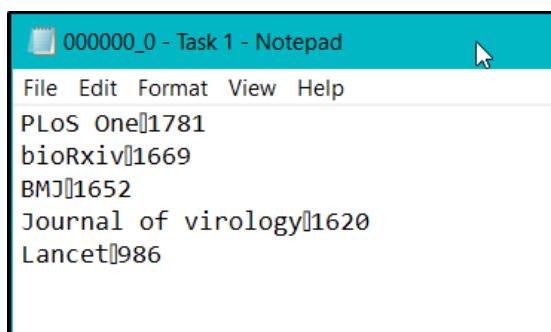
The screenshot shows the AWS S3 console interface. The URL is [Amazon S3 > jonnavittula.outputs-bucket > AWS EMR - CORD19 tasks/ > T1_Frequent_Journals/](#). The folder 'T1_Frequent_Journals/' is highlighted. The 'Objects' tab is selected, showing one object: '000000_0'. The object details show it was uploaded on April 27, 2021, at 13:16:21 (UTC+01:00), has a size of 72.0 B, and is stored in the Standard storage class.

3.1.6 Result

Name of the Journal	Frequency
PLoS One	1781
bioRxiv	1669
BMJ	1652
Journal of virology	1620
Lancet	986

	Name of the Journal	Frequency
1	PLoS One	1781
2	bioRxiv	1669
3	BMJ	1652
4	Journal of virology	1620
5	Lancet	986

```
▶ (3) Spark Jobs
Out[59]: [('PLoS One', 1781),
('bioRxiv', 1669),
('BMJ', 1652),
('Journal of virology', 1620),
('Lancet', 986)]
```



3.1.7 Result Review

- 'Journal of virology', which ranked as the most popular journal until May 2020, is now overtaken by 'PLoS One', 'bioRxiv' and 'BMJ' in that order, pushing it to 4th rank, making 'PLoS One' the most popular journal among the readers.
- It seems 'The Lancet' consistently maintained the 5th rank.
- In addition, 'Virology' and 'Emerg Infect Dis', which were on the top 5 until May, now seem to have lost their demand in the readers' market.

3.2: Task 2 - The top 5 average abstract lengths

3.2.0 Outline

1. Journal and abstract columns were selected excluding the nulls
2. Abstract column is customized to eliminate the unnecessary spaces by nullifying them first.
3. Grouping was done on journal column, while the average of the customized abstract column was aggregated and sorted.
4. In PySpark, the intermediate result is persisted to MEMORY_ONLY to optimize.
5. To this persisted result, the average abstract lengths are rounded off to a single decimal point and the columns are aliased.

3.2.1 Assumptions made

- N/A

3.2.2 PySpark – Dataframes

1. **.withColumn()** is used to customize the ‘abstract’ column to bring the length of each abstract by subtracting the additional spaces which are nullified using `regexp_replace()`.

4.1 Using Dataframe created from CSV

```
Cmd 53
1 from pyspark.sql.functions import length, regexp_replace, round
2
3 df_nospace = df_cord.select("journal", "abstract") \
4     .na.drop() \
5     .withColumn("abstract", 1 + length(df_cord.abstract) - length(regexp_replace(df_cord.abstract, ' ', ''))) \
6     .groupBy("journal") \
7     .avg("abstract").withColumnRenamed("avg(abstract)", "avg_abstract") \
8     .sort("avg_abstract", ascending=False) \
9     .persist()
10
11 task2_df = df_nospace.withColumn("avg_abstract", round(df_nospace["avg_abstract"],1)) \
12     .withColumnRenamed("journal", "Name of the Journal") \
13     .withColumnRenamed("avg_abstract", "Avg_Abstract_Length") \
14
15 task2_df.show(5)
```

4.2 Using Dataframe created from Temp View

```
Cmd 55
1 from pyspark.sql.functions import length, regexp_replace, round
2
3 df_nospace_temp = df_tempView.select("journal", "abstract") \
4     .na.drop() \
5     .withColumn("abstract", 1 + length(df_tempView.abstract) - length(regexp_replace(df_tempView.abstract, ' ', ''))) \
6     .groupBy("journal") \
7     .avg("abstract").withColumnRenamed("avg(abstract)", "avg_abstract") \
8     .sort("avg_abstract", ascending=False) \
9     .persist()
10
11 df_temp2 = df_nospace_temp.withColumn("avg_abstract", round(df_nospace_temp["avg_abstract"],1)) \
12     .withColumnRenamed("journal", "Name of the Journal") \
13     .withColumnRenamed("avg_abstract", "Avg_Abstract_Length") \
14
15 df_temp2.show(5)
```

4.3 Using Dataframe created from Parquet Files

```
Cmd 57
1 from pyspark.sql.functions import length, regexp_replace, round
2
3 df_nospace_pq = df_pq.select("journal", "abstract") \
4     .na.drop() \
5     .withColumn("abstract", 1 + length(df_pq.abstract) - length(regexp_replace(df_pq.abstract, ' ', ''))) \
6     .groupBy("journal") \
7     .avg("abstract").withColumnRenamed("avg(abstract)", "avg_abstract") \
8     .sort("avg_abstract", ascending=False) \
9     .persist()
10
11 df_pq2 = df_nospace_pq.withColumn("avg_abstract", round(df_nospace_pq["avg_abstract"],1)) \
12     .withColumnRenamed("journal", "Name of the Journal") \
13     .withColumnRenamed("avg_abstract", "Avg_Abstract_Length") \
14
15 df_pq2.show(5)
```

3.2.3 HiveQL

1. In HiveQL, we use REPLACE() function to achieve the same functionality as above.

4.1 Using Permanent Tables

```
Cmd 51

1 SELECT
2   journal AS 'Name of the Journal',
3   ROUND(
4     AVG(
5       LENGTH(abstract) - LENGTH(REPLACE(abstract, ' ', '')) + 1
6     ),1) AS 'Average Abstract lengths'
7 FROM
8   ${hivevar:db_name}.${hivevar:metadata_table}
9 GROUP BY
10  journal
11 ORDER BY
12  'Average Abstract lengths' DESC
13 LIMIT
14  5
```

4.2 Using Temp View

```
Cmd 53

1 SELECT
2   journal AS 'Name of the Journal',
3   ROUND(
4     AVG(
5       LENGTH(abstract) - LENGTH(REPLACE(abstract, ' ', '')) + 1
6     ),1) AS 'Average Abstract lengths'
7 FROM
8   temp_${hivevar:metadata_table}
9 GROUP BY
10  journal
11 ORDER BY
12  'Average Abstract lengths' DESC
13 LIMIT
14  5
```

4.3 Using Table from Parquet File

```
Cmd 55

1 SELECT
2   journal AS 'Name of the Journal',
3   ROUND(
4     AVG(
5       LENGTH(abstract) - LENGTH(REPLACE(abstract, ' ', '')) + 1
6     ),1) AS 'Average Abstract lengths'
7 FROM
8   ${hivevar:db_name}.parq_${hivevar:metadata_table}
9 GROUP BY
10  journal
11 ORDER BY
12  'Average Abstract lengths' DESC
13 LIMIT
14  5
```

3.2.4 PySpark – RDD

1. Key-value pairs are created for journal and cleaned abstract length after nullifying spaces.
2. mapValues() is used to apply average formula directly on the values in the dictionary.

```
1 task2_rdd = cord19_rdd.filter(lambda row: row.abstract is not None) \
2   .filter(lambda row: row.journal is not None) \
3   .map(lambda row: (row.journal, 1 + len(row.abstract) -
4     len(row.abstract.replace(" ", "")))) \
5   .groupByKey() \
6   .mapValues(lambda x: round(sum(x) / len(x) ,1)) \
7   .sortBy(lambda line: line[1], ascending=False)
8 task2_rdd.take(5)
```

3.2.5 AWS

3.2.5.1 AWS S3 & Athena

The screenshot shows the AWS Athena console interface. On the left, there's a sidebar with 'Data source' set to 'AwsDataCatalog' and 'Database' set to 'coviddb28'. Under 'Tables (2)', there are two entries: 'c19_2020_07_01' and 'scimagojr19'. Under 'Views (8)', there are eight entries starting with 'cord19_task'. The main area displays a query in the 'New query 1' tab:

```

1 -- Task 2. The top 5 average abstract lengths (number of words) per journal.
2 CREATE OR REPLACE VIEW cord19_task2 AS
3 SELECT
4     journal AS Journal,
5     ROUND(
6         AVG(
7             LENGTH(Abstract) - LENGTH(REPLACE(Abstract, ' ', '')) + 1
8         ),1) AS Average_Abstract_Lengths
9     FROM
10    coviddb28.c19_2020_07_01
11 WHERE
12    Abstract <> ''
13 GROUP BY
14     Journal
15 ORDER BY
16     Average_Abstract_Lengths DESC
17 LIMIT
18     5
19
20

```

Below the query, there are buttons for 'Run query', 'Save as', and 'Create'. A status message indicates '(Run time: 0.52 seconds, Data scanned: 0 KB)'. At the bottom, there are links for 'Format query', 'Clear', 'Athena engine version 2', and 'Release versions'.

3.2.5.2 AWS S3 & EMR

The screenshot shows a Notepad window titled '06 - Insert Data for task 2.txt - Notepad'. The content is a Lambda function code:

```

File Edit Format View Help
-- Task 2. The top 5 average abstract lengths (number of words) per journal.

INSERT OVERWRITE DIRECTORY '${OUTPUT}'
SELECT
    journal AS Journal,
    ROUND(
        AVG(
            LENGTH(Abstract) - LENGTH(REPLACE(Abstract, ' ', '')) + 1
        ),1) AS Average_Abstract_Lengths
FROM
    coviddb28.c19_2020_07_01
WHERE
    abstract<>''
GROUP BY
    journal
ORDER BY
    Average_Abstract_Lengths DESC
LIMIT
    5

```

The screenshot shows the AWS S3 console. The bucket path is 'Amazon S3 > jonnavittula.outputs-bucket > AWS EMR - CORD19 tasks/ > T2_Avg_Abstract_Lengths/'. The 'Objects' tab is selected, showing one object named '000000_0'. The object details are:

Name	Type	Last modified	Size	Storage class
000000_0	-	April 27, 2021, 13:22:52 (UTC+01:00)	223.0 B	Standard

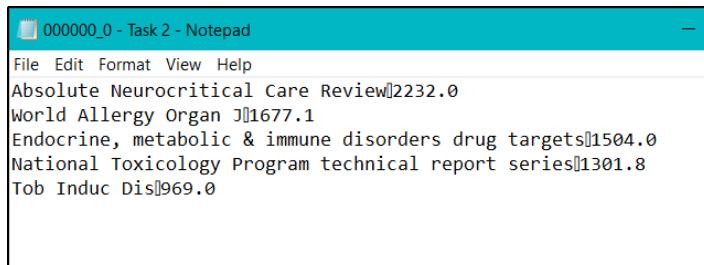
3.2.6 Result

Name of the Journal	Avg_Abstract_Length
Absolute Neurocritic...	2232.0
World Allergy Org...	1677.1
Endocrine, metabo...	1504.0
National Toxicolo...	1301.8
Tob Induc Dis	969.0

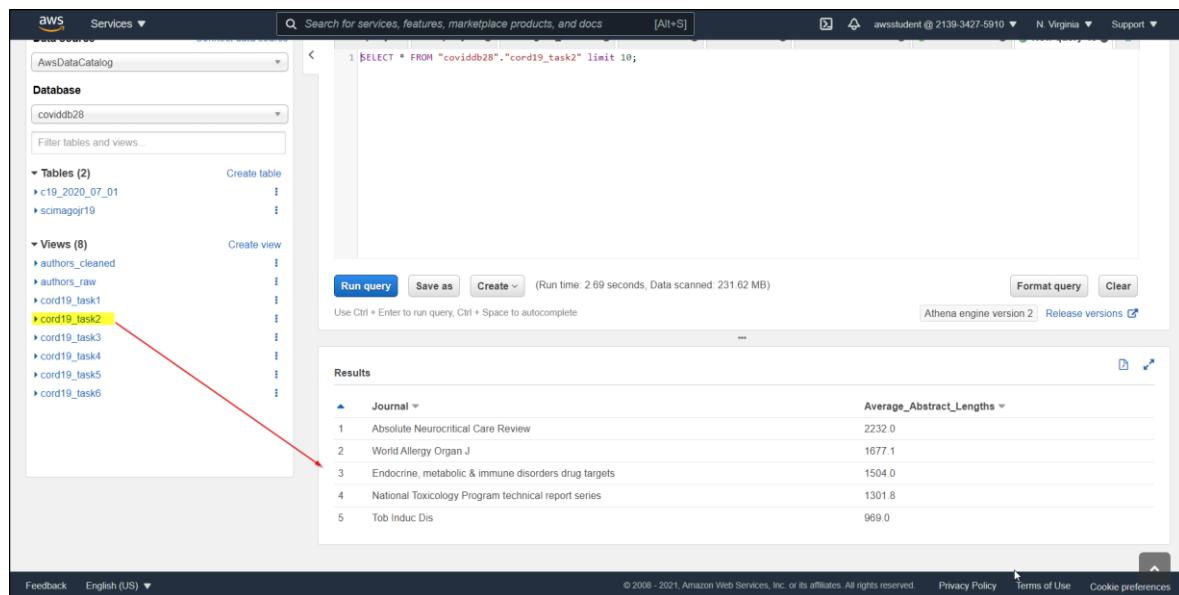
Name of the Journal	Average Abstract lengths
Absolute Neurocritical Care Review	2232
World Allergy Organ J	1677.1
Endocrine, metabolic & immune disorders drug targets	1504
National Toxicology Program technical report series	1301.8
Tob Induc Dis	969

▶ (3) Spark Jobs

```
Out[58]: [('Absolute Neurocritical Care Review', 2232.0),
 ('World Allergy Organ J', 1677.1),
 ('Endocrine, metabolic & immune disorders drug targets', 1504.0),
 ('National Toxicology Program technical report series', 1301.8),
 ('Tob Induc Dis', 969.0)]
```



```
Absolute Neurocritical Care Review||2232.0
World Allergy Organ J||1677.1
Endocrine, metabolic & immune disorders drug targets||1504.0
National Toxicology Program technical report series||1301.8
Tob Induc Dis||969.0
```



Database: coviddb28

Tables (2): c19_2020_07_01, scimagojr19

Views (8): authors_cleaned, authors_raw, cord19_task1, cord19_task2, cord19_task3, cord19_task4, cord19_task5, cord19_task6

Query:

```
1 SELECT * FROM "coviddb28"."cord19_task2" limit 10;
```

Run query | Save as | Create | (Run time: 2.69 seconds, Data scanned: 231.62 MB)

Results:

Journal	Average_Abstract_Lengths
Absolute Neurocritical Care Review	2232.0
World Allergy Organ J	1677.1
Endocrine, metabolic & immune disorders drug targets	1504.0
National Toxicology Program technical report series	1301.8
Tob Induc Dis	969.0

3.2.7 Result Review

- Until May, 'World Allergy Orggan J' has an average abstract length of 4658.5, but now in July 2020, not many words are given in these copies of the journals, hence the average length got reduced to 1677.1.
- In July 2020, 'Absolute Neurocritical Care Review' has given the highest amount of words in their copies of abstract, followed by 'Endocrine, metabolic & immune disorders drug targets' and later 'National Toxicology Program technical report series'.
- 'Tob Induc Dis' has consistently remained at 969.0, but lessened its position from 4th to 5th.

3.2.8 Further Analysis 1

- ❖ **Question** - From the 3 most recent abstracts, what are the authors trying to lay importance on / stress on?

Steps –

1. Title, Publish_time, abstract columns are selected and the nulls are dropped.
2. Using substring method, 5th and 8th characters of the publish_time column are checked for a hyphen like in a date (YYYY-MM-DD) format. This is done to ensure we have only dates.
3. Abstract column is checked not to be ‘Unknown’ which is considered as a cliché in the data, and cleaned as part of the transformation.
4. Finally, sorting is done on Publish_time in the descending order and top 3 values are shown.

4.4.1 Dataset preparation

```
Cmd 60
1 # To create a DF with Abstracts & Journal published date in descending order (Top 3)
2
3 df_taskF1 = df_cord.select("title", "publish_time", "abstract") \
4   .na.drop() \
5   .where(substring(df_cord["publish_time"],5,1) == '-') \
6   .where(substring(df_cord["publish_time"],8,1) == '-') \
7   .where(df_cord["abstract"] != 'Unknown') \
8   .sort("publish_time", ascending=False)
9
10 df_taskF1.show(3)
```

	title	publish_time	abstract
21	Respiratory vi...	2021-12-31	Abstract: Respira...
Chapter 27 Emerge...	2020-12-31	Abstract The unpr...	
Computer Vision T...	2020-12-31	Abstract Digitiza...	

only showing top 3 rows

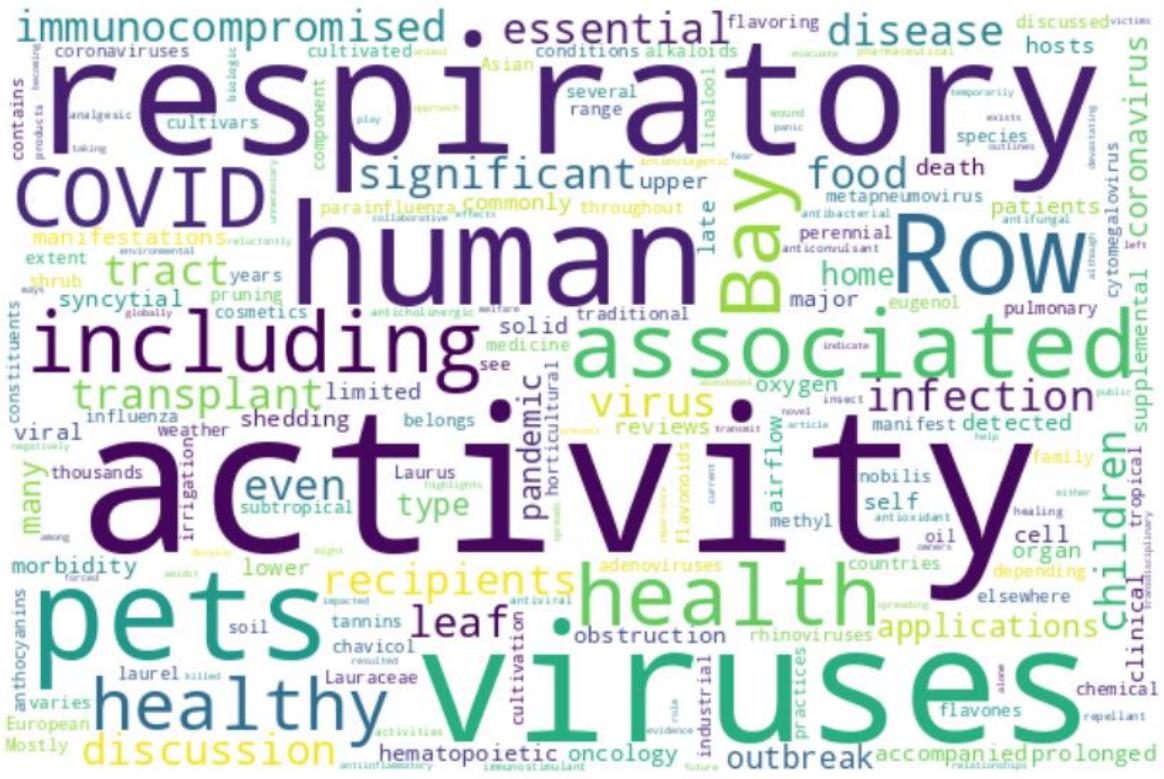
- Stopwords are used to remove words that don't add value to the plot.

4.4.2 Word Cloud plotting

```
Cmd 62
1 # Use the above created DF in a word cloud
2
3 import matplotlib.pyplot as plt
4 from wordcloud import WordCloud, STOPWORDS
5 import sys, os
6 os.chdir(sys.path[0])
7
8 # Read the top 3 abstracts into text as a "String"
9 text = str(df_taskF1.select("abstract").head(3))
10
11 # Custom Stop words
12 extra_stopwds = ['use', 'used', 'using', 'different', 'Abstract', 'need', 'due', 'method', 'system', 'data',
13 'machine', 'machining', 'chapter', 'one', 'well', 'will', 'may', 'new', 'two', 'tool', 'based', 'paper',
14 'material']
15
16 # Adding custom stopwords to the original stopwords 'set'
17 stopwords = STOPWORDS.update(extra_stopwds)
18
19 # Defining instance of WordCloud with input arguments
20 wc = WordCloud(background_color='white',
21                 stopwords=stopwords,
22                 height = 400,
23                 width=600)
24
25 # WordCloud Generate
26 wc.generate(text)
27
28 # Plot the figure using Matplotlib
29 plt.figure(figsize=[20,10])
30 plt.imshow(wc, interpolation='bilinear')
31 plt.axis("off")
32 plt.show()
```

- **Result -**

► (1) Spark Jobs



Command took 9.99 seconds -- by a.jonnevittula@edu.salford.ac.uk at 5/9/2021, 12:00:41 PM on amjon

- Result Review – By the end of the year, it seems that the journals have emphasized more on covid and associated viruses, along with their effects on the human respiratory system. In addition, they focused on the agents, especially the role of pets, which act as a catalyst sometimes, through which the people could get infected or even immunocompromised. Healthy lifestyle, food, oncology study, parainfluenza and metapneumovirus are the other topics which were accentuated.

3.3: Task 3 – 5 Titles with highest numbers of authors.

3.3.0 Outline

1. Title and Authors are selected from cord19 table
2. An exploded split is performed on Authors using the delimiter – semicolon ‘;’
3. Grouping is done on Title against the total count of authors, which is later sorted and aliased.

3.3.1 Assumptions made

- N/A

3.3.2 PySpark – Dataframes

1. Explode(split(df_cord.authors, ";")) shown below is used to first split the multiple authors within the same row and later use these authors list as an array within the dataframe.

5.1 Using Dataframe created from CSV

```
Cmd 65
1 from pyspark.sql.functions import split, explode
2
3 task3_df = df_cord.select("title", "authors") \
4     .withColumn("authors", explode(split(df_cord.authors, ";"))) \
5     .groupBy("title") \
6     .count() \
7     .sort("count", ascending=False) \
8     .withColumnRenamed("title", "Title") \
9     .withColumnRenamed("count", "Num_of_Authors")
10
11 task3_df.show(5)
```

5.2 Using Dataframe created from Temp View

```
Cmd 67
1 from pyspark.sql.functions import split, explode
2
3 df_temp3 = df_tempView.select("title", "authors") \
4     .withColumn("authors", explode(split(df_tempView.authors, ";"))) \
5     .groupBy("title") \
6     .count() \
7     .sort("count", ascending=False) \
8     .withColumnRenamed("title", "Title") \
9     .withColumnRenamed("count", "Num_of_Authors")
10
11 df_temp3.show(5)
```

5.3 Using Dataframe created from Parquet Files

```
Cmd 69
1 from pyspark.sql.functions import split, explode
2
3 df_parq3 = df_parq.select("title", "authors") \
4     .withColumn("authors", explode(split(df_parq.authors, ";"))) \
5     .groupBy("title") \
6     .count() \
7     .sort("count", ascending=False) \
8     .withColumnRenamed("title", "Title") \
9     .withColumnRenamed("count", "Num_of_Authors")
10
11 df_parq3.show(5)
```

3.2.3 HiveQL

5.1 Using Permanent Tables

```
Cmd 58
1   SELECT
2     title as Title,
3     COUNT(author) AS `Number of Authors`
4   FROM
5   (
6     SELECT
7       title,
8       explode(split(authors, ";")) as author
9     FROM
10      ${hivevar:db_name}.${hivevar:metadata_table}
11   )
12   GROUP BY
13     title
14   ORDER BY
15     `Number of Authors` DESC
16   LIMIT
17     5
18
```

5.2 Using Temp View

```
Cmd 60
1   SELECT
2     title as Title,
3     COUNT(author) AS `Number of Authors`
4   FROM
5   (
6     SELECT
7       title,
8       explode(split(authors, ";")) as author
9     FROM
10      temp_${hivevar:metadata_table}
11   )
12   GROUP BY
13     title
14   ORDER BY
15     `Number of Authors` DESC
16   LIMIT
17     5
18
```

5.3 Using Table from Parquet File

```
Cmd 62
1   SELECT
2     title as Title,
3     COUNT(author) AS `Number of Authors`
4   FROM
5   (
6     SELECT
7       title,
8       explode(split(authors, ";")) as author
9     FROM
10      ${hivevar:db_name}.parq_${hivevar:metadata_table}
11   )
12   GROUP BY
13     title
14   ORDER BY
15     `Number of Authors` DESC
16   LIMIT
17     5
18
```

3.3.4 PySpark – RDD

1. Explode is not needed in RDD, since we calculate the length of the author's array after the split at the iteration itself.

```
1 task3_rdd = cord19_rdd.filter(lambda row: row.title is not None) \
2           .filter(lambda row: row.authors is not None) \
3           .map(lambda row: [row.title, len(row.authors.split(";"))]) \
4           .sortBy(lambda line: line[1], ascending=False)
5
6 task3_rdd.take(5)
```

3.3.5 AWS

3.3.5.1 AWS S3 & Athena

1. In Presto also, we calculate the average directly at the query level.

The screenshot shows the AWS Athena Query Editor interface. On the left, the sidebar displays the data source (AwsDataCatalog), database (coviddb28), and tables (c19_2020_07_01, scimagojr19). The main area contains a query editor with the following code:

```
1 -- Task 3. Titles of the 5 papers with the highest numbers of authors. Both the numbers of authors and the
2 -- corresponding titles need to be output.
3
4 CREATE OR REPLACE VIEW cord19_task3 AS
5
6 SELECT
7     title as Title,
8     SUM(length(authors) - LENGTH(REPLACE(authors, ',', '')) +1 ) AS Num_of_Authors
9 FROM
10    "coviddb28"."c19_2020_07_01"
11 GROUP BY
12     title
13 ORDER BY
14     Num_of_Authors DESC
15 LIMIT
16     5
17
```

Below the code, the status bar indicates "Run time: 0.53 seconds, Data scanned: 0 KB". The results section shows "Query successful."

3.3.5.2 AWS S3 & EMR

The screenshot shows a Notepad window titled "07 - Insert Data for task 3.txt - Notepad". The content is a SQL query for Task 3:

```
-- Task 3. Titles of the 5 papers with the highest numbers of authors. Both the numbers of authors and the
-- corresponding titles need to be output.

INSERT OVERWRITE DIRECTORY '${OUTPUT}'
SELECT
    title as Title,
    SUM(length(authors) - LENGTH(REPLACE(authors, ',', '')) +1 ) AS Num_of_Authors
FROM
    coviddb28.c19_2020_07_01
GROUP BY
    title
ORDER BY
    Num_of_Authors DESC
LIMIT
    5
```

The screenshot shows the AWS S3 console. The path is Amazon S3 > jonnnavittula.outputs-bucket > AWS EMR - CORD19 tasks/ > T3_Papers_with_Highest_Authors/. The "Objects" tab is selected, showing one object named "000000_0".

Name	Type	Last modified	Size	Storage class
000000_0		April 27, 2021, 15:27:35 (UTC+01:00)	382.0 B	Standard

3.3.6 Result

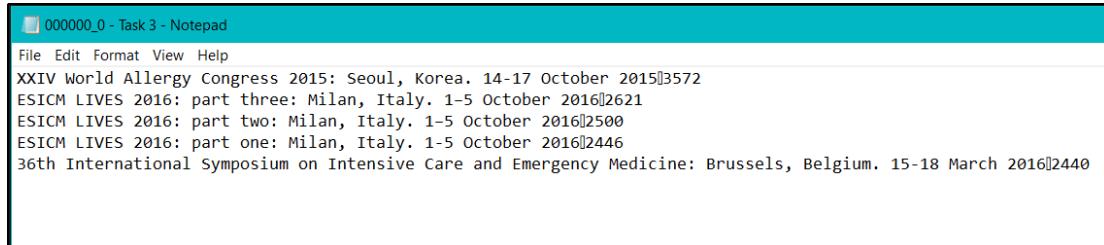
Title	Num_of_Authors
XXIV World Allerg...	3572
ESICM LIVES 2016:...	2621
ESICM LIVES 2016:...	2500
ESICM LIVES 2016:...	2446
36th Internationa...	2440

Title	Number of Authors
1 XXIV World Allergy Congress 2015: Seoul, Korea. 14-17 October 2015	3572
2 ESICM LIVES 2016: part three: Milan, Italy. 1-5 October 2016	2621
3 ESICM LIVES 2016: part two: Milan, Italy. 1-5 October 2016	2500
4 ESICM LIVES 2016: part one: Milan, Italy. 1-5 October 2016	2446
5 36th International Symposium on Intensive Care and Emergency Medicine: Brussels, Belgium. 15-18 March 2016	2440

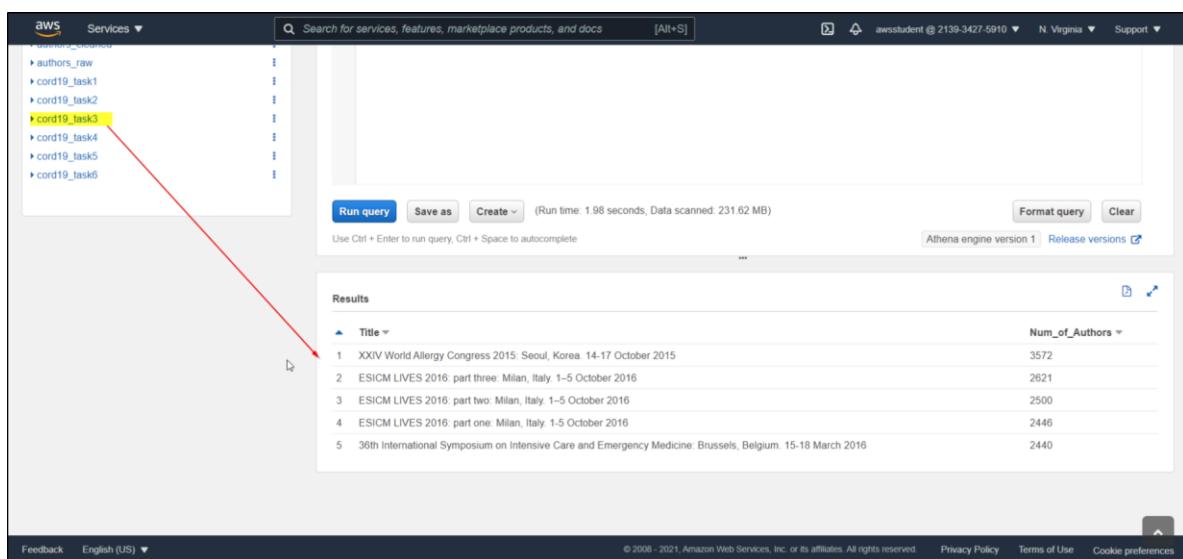
► (3) Spark Jobs

```
Out[57]: [['XXIV World Allergy Congress 2015: Seoul, Korea. 14-17 October 2015', 3572],
['ESICM LIVES 2016: part three: Milan, Italy. 1-5 October 2016', 2621],
['ESICM LIVES 2016: part two: Milan, Italy. 1-5 October 2016', 2500],
['ESICM LIVES 2016: part one: Milan, Italy. 1-5 October 2016', 2446],
['36th International Symposium on Intensive Care and Emergency Medicine: Brussels, Belgium. 15-18 March 2016',
 2440]]
```

Command took 39.35 seconds -- by a.jonnavittula@edu.salford.ac.uk at 5/8/2021, 7:11:38 PM on amjon2



```
File Edit Format View Help
XXIV World Allergy Congress 2015: Seoul, Korea. 14-17 October 2015|3572
ESICM LIVES 2016: part three: Milan, Italy. 1-5 October 2016|2621
ESICM LIVES 2016: part two: Milan, Italy. 1-5 October 2016|2500
ESICM LIVES 2016: part one: Milan, Italy. 1-5 October 2016|2446
36th International Symposium on Intensive Care and Emergency Medicine: Brussels, Belgium. 15-18 March 2016|2440
```



Title	Num_of_Authors
1 XXIV World Allergy Congress 2015: Seoul, Korea. 14-17 October 2015	3572
2 ESICM LIVES 2016: part three: Milan, Italy. 1-5 October 2016	2621
3 ESICM LIVES 2016: part two: Milan, Italy. 1-5 October 2016	2500
4 ESICM LIVES 2016: part one: Milan, Italy. 1-5 October 2016	2446
5 36th International Symposium on Intensive Care and Emergency Medicine: Brussels, Belgium. 15-18 March 2016	2440

3.3.7 Result Review

- ESICM LIVES 2016: part two and part three have gained more authors following part one from May.
- XXIV World Allergy Congress 2015 and 36th International Symposium journals have remained in the top 5, and the former ranking the first.

3.4: Task 4 – Top 5 authors and papers published

3.4.0 Outline

1. The authors column needs to be selected, split and exploded using a delimiter ';' (semicolon).
2. Then we filter out any authors that have digits in their names using regular expressions.
3. The resulted authors are now grouped and counted, and later sorted in descending order.

3.4.1 Assumptions made

- More often than not, people/authors don't have digits in their names, so this code was written to solve a general problem of removing numbers from names. Example – D039, O039, etc.

3.4.2 PySpark – Dataframes

- Generators are not supported when it's nested in expressions.
- RLIKE is a method used for pattern finding.

```
6.1 Using Dataframe created from CSV
Cmd 72

1 from pyspark.sql.functions import split, explode, trim
2
3 # Explode view is created here
4 df_notrim = df_cord.select(explode(split("authors", ";")).alias("Author"), "journal").persist()
5
6 # Filtering out any author that has numbers in their name
7 df_notrim = df_notrim.withColumn("Author_alpha", trim(df_notrim["Author"]).rlike("[0-9]")).persist()
8 df_notrim = df_notrim.filter(df_notrim["Author_alpha"]==False).persist()
9
10 # Performing Task 4 now
11 task4_df = df_notrim.withColumn("Author", trim(df_notrim["Author"])) \
12     .select("Author") \
13     .groupBy("Author") \
14     .count() \
15     .withColumnRenamed("count", "Num_of_Papers") \
16     .sort("Num_of_Papers", ascending=False)
17
18 task4_df.show(5)
19
```

```
6.2 Using Dataframe created from Temp View
Cmd 74

1 from pyspark.sql.functions import split, explode, trim
2
3 # Explode view is created here
4 df_notrim_temp = df_tempView.select(explode(split("authors", ";")).alias("Author"), "journal").persist()
5
6 # Filtering out any author that has numbers in their name
7 df_notrim_temp = df_notrim_temp.withColumn("Author_alpha", trim(df_notrim_temp["Author"]).rlike("[0-9]")).persist()
8 df_notrim_temp = df_notrim_temp.filter(df_notrim_temp["Author_alpha"]==False).persist()
9
10 # Performing Task 4 now
11 df_tempt4 = df_notrim_temp.withColumn("Author", trim(df_notrim_temp["Author"])) \
12     .select("Author") \
13     .groupBy("Author") \
14     .count() \
15     .withColumnRenamed("count", "Num_of_Papers") \
16     .sort("Num_of_Papers", ascending=False)
17
18 df_tempt4.show(5)
19
```

```
from pyspark.sql.functions import split, explode, trim
2
3 # Explode view is created here
4 df_notrim_paq = df_paq.select(explode(split("authors", ";")).alias("Author"), "journal").persist()
5
6 # Filtering out any author that has numbers in their name
7 df_notrim_paq = df_notrim_paq.withColumn("Author_alpha", trim(df_notrim_paq["Author"]).rlike("[0-9]")).persist()
8 df_notrim_paq = df_notrim_paq.filter(df_notrim_paq["Author_alpha"]==False).persist()
9
10 # Performing Task 4 now
11 df_paqt4 = df_notrim_paq.withColumn("Author", trim(df_notrim_paq["Author"])) \
12     .select("Author") \
13     .groupBy("Author") \
14     .count() \
15     .withColumnRenamed("count", "Num_of_Papers") \
16     .sort("Num_of_Papers", ascending=False)
17
18 df_paqt4.show(5)
19
```

3.4.3 HiveQL

6.1 Using Permanent Tables

Cmd 65

```
1 SELECT
2   TRIM(full_list) AS `Name of the Author`,
3   COUNT(*) AS `Number of Papers`
4 FROM
5   ${hivevar:db_name}.${hivevar:metadata_table} LATERAL VIEW EXPLODE(SPLIT(authors, ',')) AS full_list
6 WHERE
7   TRIM(full_list) NOT RLIKE '[0-9]'
8 GROUP BY
9   `Name of the Author`
10 ORDER BY
11   `Number of Papers` desc
12 LIMIT
13   5
```

6.2 Using Temp View

Cmd 67

```
1 SELECT
2   TRIM(full_list) AS `Name of the Author`,
3   COUNT(*) AS `Number of Papers`
4 FROM
5   temp_${hivevar:metadata_table} LATERAL VIEW EXPLODE(SPLIT(authors, ',')) AS full_list
6 WHERE
7   TRIM(full_list) NOT RLIKE '[0-9]'
8 GROUP BY
9   `Name of the Author`
10 ORDER BY
11   `Number of Papers` desc
12 LIMIT
13   5
```

6.3 Using Table from Parquet File

Cmd 69

```
1 SELECT
2   TRIM(full_list) AS `Name of the Author`,
3   COUNT(*) AS `Number of Papers`
4 FROM
5   ${hivevar:db_name}.parq_${hivevar:metadata_table} LATERAL VIEW EXPLODE(SPLIT(authors, ',')) AS full_list
6 WHERE
7   TRIM(full_list) NOT RLIKE '[0-9]'
8 GROUP BY
9   `Name of the Author`
10 ORDER BY
11   `Number of Papers` desc
12 LIMIT
13   5
```

3.4.4 PySpark – RDD

- Re is the regular expressions library from python. Re.match() method does pattern matching.
- .cache() is used to cache the result in memory & .flatMap() is used to flatten the list of authors.
- .subtract() is used to remove the authors that have digits in their names from the final rdd.

```
1 import re
2
3 task4_junk_rdd = cord19_rdd.filter(lambda row: row.authors is not None) \
4   .map(lambda row: row.authors.split(";")) \
5   .flatMap(lambda line: line) \
6   .filter(lambda line: re.match("^[a-zA-Z][0-9]", line)) \
7   .cache()
8
9 task4_rdd = cord19_rdd.filter(lambda row: row.authors is not None) \
10  .map(lambda row: row.authors.split(";")) \
11  .flatMap(lambda line: line) \
12  .cache()
13
14 task4_rdd = task4_rdd.subtract(task4_junk_rdd) \
15  .map(lambda line: (line.strip(),1)) \
16  .reduceByKey(lambda v1,v2: v1+v2, 15) \
17  .sortBy(lambda line: line[1], ascending=False) \
18
19 task4_rdd.take(5)
```

3.4.5 AWS

3.4.5.1 AWS S3 & Athena

The screenshot shows the AWS Athena Query Editor interface. On the left, the sidebar displays the Data source (AthenaDataCatalog) and Database (coviddb28). Under Tables, there are two entries: c19_2020_07_01 and scimagojr19. Under Views, there are several entries, with authors_cleaned highlighted in yellow. The main panel shows a query editor with the following SQL code:

```
1 -- Task 4. The top 5 most prolific authors along with the number of papers they have contributed to.
2
3 CREATE OR REPLACE VIEW cord19_task4 AS
4 SELECT author as Author,
5       COUNT(journal) as Num_of_Papers
6  FROM coviddb28.authors_cleaned
7 WHERE author <> '' AND SUBSTR(author,2,3) <> '039'
8 GROUP BY author
9 ORDER BY Num_of_Papers DESC
10 LIMIT 5;
11
12
```

Below the query, the status bar indicates: (Run time: 2.75 seconds, Data scanned: 231.62 MB). At the bottom, there are buttons for Run query, Save as, Create, Format query, and Clear.

3.4.5.2 AWS S3 & EMR

The screenshot shows a Notepad window titled "08 - Insert Data for task 4.txt - Notepad". The content of the file is the same SQL query as shown in the Athena screenshot, intended for execution on an EMR cluster.

```
-- Task 4. The top 5 most prolific authors along with the number of papers they have contributed to.

INSERT OVERWRITE DIRECTORY '${OUTPUT}'
SELECT author as Author,
       COUNT(journal) as Num_of_Papers
FROM coviddb28.authors_cleaned
WHERE author <> '' AND SUBSTR(author,2,3) <> '039'
GROUP BY author
ORDER BY Num_of_Papers DESC
LIMIT 5;
```

The screenshot shows the AWS S3 console. The path is Amazon S3 > jonnavaittula.outputs-bucket > AWS EMR - CORD19 tasks/ > T4_Prolific_Authors/. A single object named "000000_0" is listed in the Objects table. The table has columns: Name, Type, Last modified, Size, and Storage class. The object details show it is a file, last modified on April 27, 2021, at 13:32:46 (UTC+01:00), with a size of 96.0 B and a Standard storage class. There is a "Copy S3 URI" button at the top right of the object preview.

3.4.6 Result

Author	Num_of_Papers
Yuen, Kwok-Yung	260
Wang, Wei	235
Drosten, Christian	229
Wiwanitkit, Viroj	202
Jiang, Shibo	201

	Name of the Author	Number of Papers
1	Yuen, Kwok-Yung	260
2	Wang, Wei	235
3	Drosten, Christian	229
4	Wiwanitkit, Viroj	202
5	Jiang, Shibo	201

► (3) Spark Jobs

```
Out[47]: [('Yuen, Kwok-Yung', 260),  
          ('Wang, Wei', 235),  
          ('Drosten, Christian', 229),  
          ('Wiwanitkit, Viroj', 202),  
          ('Jiang, Shibo', 201)]
```

000000_0 - Task 4 - Notepad

```
File Edit Format View Help  
Yuen, Kwok-Yung|260  
Wang, Wei|235  
Drosten, Christian|229  
Wiwanitkit, Viroj|202  
Jiang, Shibo|201
```

aws Services ▾

Search for services, features, marketplace products, and docs [Alt+S]

awsstudent @ 2139-3427-5010 ▾ N. Virginia ▾ Support ▾

authors_raw
cord19_task1
cord19_task2
cord19_task3
cord19_task4
cord19_task5
cord19_task6

Run query Save as Create (Run time: 3.08 seconds, Data scanned: 231.62 MB)
Format query Clear

Athena engine version 1 Release versions

Results

	Author	Num_of_Papers
1	Yuen, Kwok-Yung	260
2	Wang, Wei	235
3	Drosten, Christian	229
4	Wiwanitkit, Viroj	202
5	Jiang, Shibo	201

Feedback English (US) ▾ © 2008 - 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use Cookie preferences

3.4.7 Result Review

- Christian Drosten and Shibo Jiang remained to be on the top 5 authors who contributed to the maximum number of papers, with Kwok-Yung Yuen in the leading position.

3.5: Task 5 – Top 5 authors with H-index

3.5.0 Outline

1. An inner join is established between cord19 dataset (field = journal) and scimago dataset (field = Title) to select the list of authors and H-index.
2. The list of authors needs to be cleaned as we did in the previous step. So we use the same result which persisted in memory.
3. From the joined dataset, authors are grouped against the total H-index value, which is later sorted and aliased.

3.5.1 Assumptions made

- The cleaned list of authors in task 4 is used in task 5 to eliminate redundancy.

3.5.2 PySpark – Dataframes

- .join() is used to join 2 dataframes.
- H_index is a string, so in order to aggregate, it was casted into Integer type.

7.1 Using Dataframe created from CSV

```
Cmd 79
1  from pyspark.sql.functions import split, explode, trim
2
3  df_joined = df_notrim.join(df_scimagojr, df_notrim.journal == df_scimagojr.Title, "inner") \
4      .select(trim("Author").alias("Author"), "H_index") \
5      .persist()
6
7  task5_df = df_joined.withColumn("H_index", df_joined["H_index"].cast("Integer")) \
8      .groupBy("Author") \
9      .sum("H_index") \
10     .withColumnRenamed("sum(H_index)", "Total_H_Index") \
11     .sort("Total_H_Index", ascending=False)
12
13 task5_df.show(5)
```

7.2 Using Dataframe created from Temp View

```
Cmd 81
1  from pyspark.sql.functions import split, explode, trim
2
3  df_joined_temp = df_notrim_temp.join(df_scimagojr, df_notrim_temp.journal == df_scimagojr.Title, "inner") \
4      .select(trim("Author").alias("Author"), "H_index") \
5      .persist()
6
7  df_temp5 = df_joined_temp.withColumn("H_index", df_joined_temp["H_index"].cast("Integer")) \
8      .groupBy("Author") \
9      .sum("H_index") \
10     .withColumnRenamed("sum(H_index)", "Total_H_Index") \
11     .sort("Total_H_Index", ascending=False)
12
13 df_temp5.show(5)
```

7.3 Using Dataframe created from Parquet Files

```
Cmd 83
1  from pyspark.sql.functions import split, explode, trim
2
3  df_joined_parq = df_notrim_parq.join(df_scimagojr, df_notrim_parq.journal == df_scimagojr.Title, "inner") \
4      .select(trim("Author").alias("Author"), "H_index") \
5      .persist()
6
7  df_parqt5 = df_joined_parq.withColumn("H_index", df_joined_parq["H_index"].cast("Integer")) \
8      .groupBy("Author") \
9      .sum("H_index") \
10     .withColumnRenamed("sum(H_index)", "Total_H_Index") \
11     .sort("Total_H_Index", ascending=False)
12
13 df_parqt5.show(5)
```

3.5.3 HiveQL

7.1 Using Permanent Tables

```
Cmd 72
1 SELECT
2   TRIM(author_list) AS `Name of the Author`,
3   SUM(`H index`) AS `Total H-index`
4 FROM
5   ${hivevar:db_name}.${hivevar:metadata_table} AS A
6   INNER JOIN
7   ${hivevar:db_name}.${hivevar:scimagojr_table} AS J
8   ON A.journal = J.title
9   LATERAL VIEW EXPLODE(SPLIT(authors, ',')) AS author_list
10 GROUP BY
11   `Name of the Author`
12 ORDER BY
13   `Total H-index` DESC
14 LIMIT
15   5
```

7.2 Using Temp View

```
Cmd 74
1 SELECT
2   TRIM(author_list) AS `Name of the Author`,
3   SUM(`H index`) AS `Total H-index`
4 FROM
5   temp_${hivevar:metadata_table} AS A
6   INNER JOIN
7   ${hivevar:db_name}.${hivevar:scimagojr_table} AS J
8   ON A.journal = J.title
9   LATERAL VIEW EXPLODE(SPLIT(authors, ',')) AS author_list
10 GROUP BY
11   `Name of the Author`
12 ORDER BY
13   `Total H-index` DESC
14 LIMIT
15   5
```

7.3 Using Table from Parquet File

```
Cmd 76
1 SELECT
2   TRIM(author_list) AS `Name of the Author`,
3   SUM(`H index`) AS `Total H-index`
4 FROM
5   ${hivevar:db_name}.parq_${hivevar:metadata_table} AS A
6   INNER JOIN
7   ${hivevar:db_name}.${hivevar:scimagojr_table} AS J
8   ON A.journal = J.title
9   LATERAL VIEW EXPLODE(SPLIT(authors, ',')) AS author_list
10 GROUP BY
11   `Name of the Author`
12 ORDER BY
13   `Total H-index` DESC
14 LIMIT
15   5
```

3.5.4 PySpark – RDD

```
1 authors_rdd = cord19_rdd.filter(lambda row: row.authors is not None) \
2   .filter(lambda row: row.journal is not None) \
3   .map(lambda row: (row.journal, row.authors)) \
4   .persist()
5
6 h_index_rdd = scimagojr_rdd.filter(lambda row: row.Title is not None) \
7   .filter(lambda row: row["H_index"] is not None) \
8   .map(lambda row: (row.Title, row["H_index"])) \
9   .persist()
10
11 task5_rdd = authors_rdd.join(h_index_rdd) \
12   .map(lambda line: (line[1][1], line[1][0].split(";"))) \
13   .flatMapValues(lambda line: line) \
14   .map(lambda line: (line[1].strip(), int(line[0]))) \
15   .groupByKey() \
16   .mapValues(lambda x : sum(x)) \
17   .sortBy(lambda line: line[1], ascending=False) \
18
19
20 task5_rdd.take(5)
```

3.5.5 AWS

3.5.5.1 AWS S3 & Athena

The screenshot shows the AWS Athena Query Editor interface. On the left, the sidebar displays the Data source (AthenaDataCatalog), Database (coviddb28), and Tables (c19_2020_07_01, scimagojr19). The main area contains the following SQL query:

```

1 -- Task 5. If an author's H index is computed by summing all the H indexes of the journals they've published
2 -- in (as included in the scimagojr dataset), list the 5 people with the top author H index values.
3
4 CREATE OR REPLACE VIEW cord19_tasks5 AS
5 SELECT author,
6       SUM(H_index) as Total_H_Index
7   FROM coviddb28.scimagojr19 as C
8   JOIN coviddb28.authors_cleaned as S
9      ON C.journal = S.Title
10     WHERE length(author) > 0
11 GROUP BY author
12 ORDER BY Total_H_Index DESC
13 LIMIT 5
14

```

Below the query, the results pane shows a single row with the author and Total_H_Index columns. The footer indicates a run time of 3.8 seconds and data scanned of 239.41 MB.

3.5.5.2 AWS S3 & EMR

The screenshot shows a Notepad window titled "09 - Insert Data for task 5.txt". The content is a SQL query for Task 5:

```

File Edit Format View Help
-- Task 5. If an author's H index is computed by summing all the H indexes of the journals they've published
-- in (as included in the scimagojr dataset), list the 5 people with the top author H index values.

INSERT OVERWRITE DIRECTORY '${OUTPUT}'
SELECT author,
       SUM(H_index) as Total_H_Index
FROM coviddb28.authors_cleaned as C
JOIN coviddb28.scimagojr19 as S
ON C.journal = S.Title
WHERE length(author) > 0
GROUP BY author
ORDER BY Total_H_Index DESC
LIMIT 5

```

The screenshot shows the AWS S3 console. The path is Amazon S3 > jonnavaittula.outputs-bucket > AWS EMR - CORD19 tasks/ > T5_Author_H_Indexes/. The Objects tab is selected, showing one object named "000000_0". The object was last modified on April 27, 2021, at 14:38:59 (UTC+01:00) and has a size of 107.0 B.

3.5.6 Result

Author	Total_H_Index
Cyranoski, David	53314
Kupferschmidt, Kai	52828
Cohen, Jon	50580
Maxmen, Amy	30134
Mallapaty, Smriti	30134

	Name of the Author	Total H-index
1	Cyranoski, David	53314
2	Kupferschmidt, Kai	52828
3	Cohen, Jon	50580
4	Maxmen, Amy	30134
5	Mallapaty, Smriti	30134

► (3) Spark Jobs

```
Out[50]: [('Cyranoski, David', 53314),
('Kupferschmidt, Kai', 52828),
('Cohen, Jon', 50580),
('Maxmen, Amy', 30134),
('Mallapaty, Smriti', 30134)]
```

```
000000_0 - Task 5 - Notepad
File Edit Format View Help
cyranoski, David|53314
Kupferschmidt, Kai|52828
Cohen, Jon|50580
Maxmen, Amy|30134
Mallapaty, Smriti|30134
```

author	Total_H_Index
1 Cyranoski, David	53314
2 Kupferschmidt, Kai	52828
3 Cohen, Jon	50580
4 Mallapaty, Smriti	30134
5 Maxmen, Amy	30134

3.5.7 Result Review

- All the top 5 authors until May are now replaced by new authors measured in terms of H-index.
- David Cyranoski's articles seem to have received the most citations over the period.

3.5.8 Further Analysis 2

- ❖ For the COVID-19 based journals, is there a correlation between the number of cites in the last 3 years and H-index?

7.4 Further Analysis 2 - Understanding Scimago Journal Rankings dataset (in relation to CORD19 data)

Question - For the COVID-19 based journals, is there a correlation between the number of cites in the last 3 years and H-index?

Linear Regression - H-index vs. Number of cites in the last 3 years since 2019

Libraries used - NumPy, Bokeh

Cmd 85

7.4.1 Dataset preparation

Cmd 86

```
1 df_joined = df_cord.join(df_scimagojr, df_cord.journal == df_scimagojr.Title, "inner") \
2     .select(trim("Region").alias("Region"),
3             trim("Country").alias("Country"),
4             trim("journal").alias("Journal"),
5             substring("publish_time",1,7).alias("Year_Month"),
6             "H_index",
7             "Total_Docs_"+scimago_year,
8             "Total_Docs_3years",
9             "Total_Ref",
10            "Total_Cites_3years",
11            "Citable_Docs_3years",
12            regexp_replace("Categories", "[(%Q1%2%3%4)]", "").alias("Categories")) \
13            .where(substring(df_cord["publish_time"],5,1) == '-')
14            .where(substring(df_cord["publish_time"],8,1) == '-')
15            .persist()
16
17 df_joined.printSchema()
```

7.4.2 Correlation Coefficient & R-squared values are calculated

Cmd 88

```
1 import numpy as np
2
3 # Setting up lists for Data Visualization / Plotting
4
5 df_regr = df_joined.select("H_index","Total_Cites_3years")
6
7 x_dep = []
8 y_indep = []
9
10 for item in df_regr.rdd.collect():
11     x_dep.append(item[0])
12     y_indep.append(item[1])
13
14 correlation_matrix = np.corrcoef(x_dep, y_indep)
15 correlation_xy = correlation_matrix[0,1]
16 r_squared = correlation_xy**2
17
18 print("R-Squared value for this regression model: ", r_squared)
19 if r_squared > 0.8:
20     print("Since R-squared value seems to be close to 1, we can proceed with plotting a graph for this correlation.")
```

▶ (1) Spark Jobs

▶ df_regr_pyspark.sql.dataframe.DataFrame = [H_index: integer, Total_Cites_3years: integer]

R-Squared value for this regression model: 0.8650663993365079

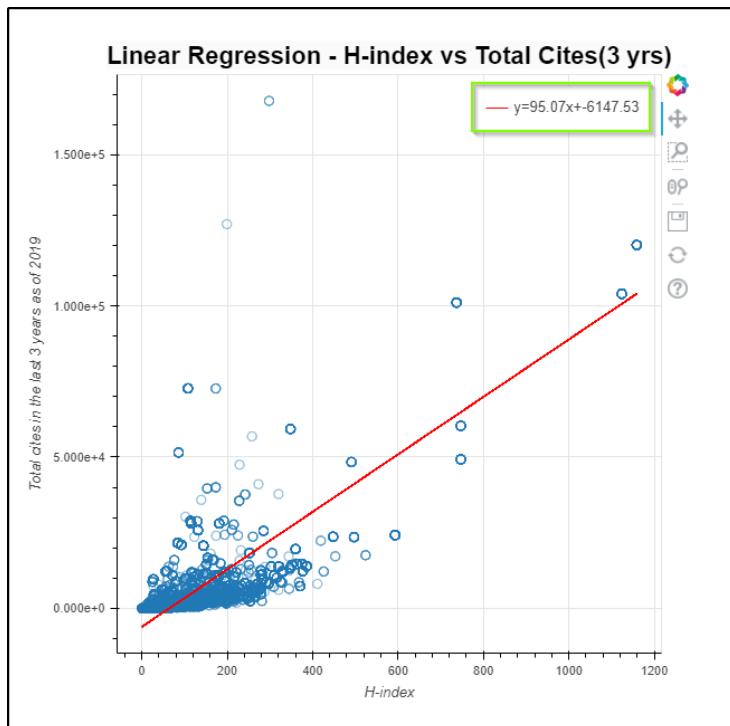
Since R-squared value seems to be close to 1, we can proceed with plotting a graph for this correlation.

Command took 0.49 seconds -- by a.jonnavittula@edu.salford.ac.uk at 5/9/2021, 1:53:08 PM on amjon

7.4.3 Linear Regression plot

```
Cmd 90
1 import numpy as np
2 from bokeh.plotting import figure
3 from bokeh.io import show
4 from bokeh.embed import components, file_html
5 from bokeh.resources import CDN
6
7 #the data
8 x=x_dep
9 y=y_indep
10
11 # determine best fit line
12 par = np.polyfit(x, y, 1, full=True)
13 slope=par[0][0]
14 intercept=par[0][1]
15 y_predicted = [slope*i + intercept for i in x]
16
17 # Legend
18 lgd = 'y='+str(np.round(slope,2))+'x'+str(np.round(intercept,2))
19
20 # Plot definition
21 fig=figure(width=600, height=600, x_axis_label='H-index', y_axis_label='Total cites in the last 3 years as of'+scimago_year)
22 fig.circle(x,y, size=8, fill_alpha=0, alpha=0.5)
23 fig.line(x,y_predicted,color='red', legend_label = lgd)
24 show(fig)
25
26 # To configure visual properties on a plot's title attribute
27 fig.title.text = "Linear Regression - H-index vs Total Cites(3 yrs)"
28 fig.title.align = "center"
29 fig.title.text_color = "black"
30 fig.title.text_font_size = "22px"
31 fig.title.background_fill_color = "#fafafa"
32
33 # To create an html document that embeds the Bokeh plot
34 html2 = file_html(fig, CDN, "my_plot2")
35
36 # To display this html
37 displayHTML(html2)
```

- Result -



- Result Review – There seems to be a linear relationship between the current H-index vs. number of citations from last 3 years. The slope seems to be steeper enough, so the rate of change is greater.

3.6: Task 6 – Number of papers since 2020-01

3.6.0 Outline

1. The substring of first 7 characters for publish_time is selected,
2. To remove jargon values, we select only values that have a hyphen in 5th and 8th characters.
3. We group by Year-Month, and sort in ascending order for the number of occurrences.
4. Arrays are used in matplotlib plot and a data dictionary is created for bokeh / seaborn plots.

3.6.1 Assumptions made

- The column ‘publish_time’ from covid19 dataset is considered as a string value.

3.6.2 PySpark – Dataframes

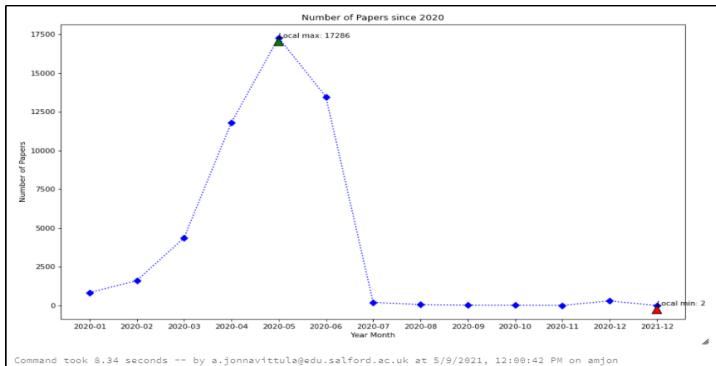
8.1 Using Dataframe created from CSV

Cmd 93

```
1  from pyspark.sql.functions import substring, length
2
3  task6_df = df_cord.select(substring("publish_time",1,7).alias("Year_Month"), "journal") \
4      .where(substring(df_cord["publish_time"],1,4) >= '2020') \
5      .where(substring(df_cord["publish_time"],5,1) == '-') \
6      .groupBy("Year_Month") \
7      .count() \
8      .sort("Year_Month", ascending=True) \
9      .withColumnRenamed("count", "Freq")
10
11 task6_df.show()
```

- Max and min values are calculated for annotations.
- Matplotlib.pyplot -

```
1  import matplotlib.pyplot as plt
2  from matplotlib.pyplot import figure
3
4  figure(figsize=(14, 8), dpi=80)
5
6  names = []
7  values = []
8
9  for item in task6_df.rdd.collect():
10    names.append(str(item[0]))
11    values.append(item[1])
12
13 ymax = max(values)
14 xpos = values.index(ymax)
15 xmax = names[xpos]
16
17 ymin = min(values)
18 xpos_1 = values.index(ymin)
19 xmin = names[xpos_1]
20
21 plt.plot(names, values, label="Number of Papers", marker = 'o', color='b', linestyle='dotted')
22
23 plt.annotate('Local max: ' + str(ymax), xy=(xmax, ymax), xytext=(xmax, ymax+5),
24               arrowprops=dict(facecolor='green'),
25               )
26
27 plt.annotate('Local min: ' + str(ymin), xy=(xmin, ymin), xytext=(xmin, ymin+5),
28               arrowprops=dict(facecolor='red'),
29               )
30
31 plt.xlabel('Year Month')
32 plt.ylabel('Number of Papers')
33 plt.title('Number of Papers since 2020')
34
35 plt.show()
```



8.2 Using Dataframe created from Temp View

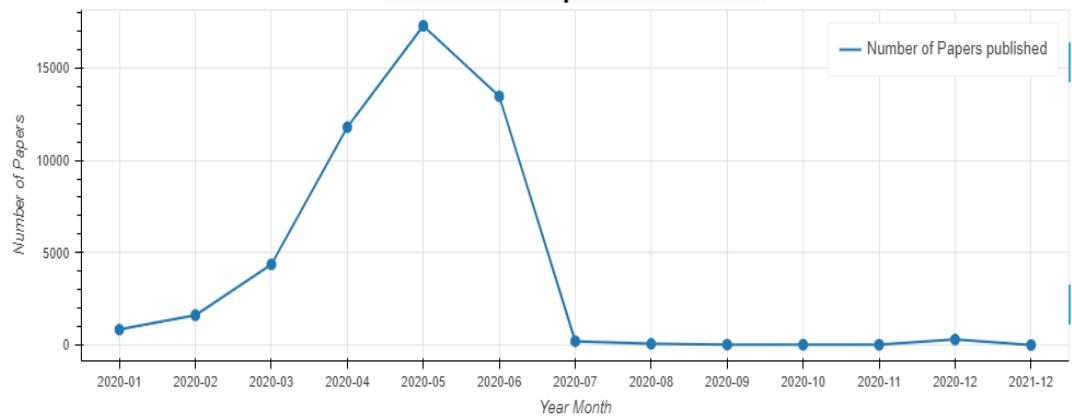
```
Cmd 97
1  from pyspark.sql.functions import substring, length
2
3  df_temp6 = df_tempView.select(substring("publish_time",1,7).alias("Year_Month"), "journal") \
4      .where(substring(df_tempView["publish_time"],1,4) >= '2020') \
5      .where(substring(df_tempView["publish_time"],5,1) == '-') \
6      .groupBy("Year_Month") \
7      .count() \
8      .sort("Year_Month", ascending=True) \
9      .withColumnRenamed("count", "Num_of_Papers")
10
11 df_temp6.show()
```

- Bokeh plot -

```
1  from bokeh.plotting import figure, output_file
2  from bokeh.models import ColumnDataSource, HoverTool
3  from bokeh.embed import components, file_html
4  from bokeh.resources import CDN
5
6  names = []
7  values = []
8
9  for item in df_temp6.rdd.collect():
10    names.append(str(item[0]))
11    values.append(item[1])
12
13 # Creating the Data Dictionary for X & Y (tooltips)
14 data = {"x": names, "y": values}
15
16 # Create a Figure() with necessary attributes
17 p = figure(x_range=names, plot_height=350, plot_width=1000,
18             x_axis_label='Year Month', y_axis_label='Number of Papers', tooltips=None)
19
20 # To render the plot into a Line chart, we use a line()
21 p.line(names, values, legend_label="Number of Papers published", line_width=2)
22
23 # To render markers where the datapoints are present
24 p.circle(names, values, width=5)
25
26 # To add HoverTool which would enable hovering across the vertical line & give tooltips
27 p.add_tools(HoverTool(tooltips=[('Year Month', '@x'), ('Num of Papers', '@y')], 
28                      # display a tooltip whenever the cursor is vertically in line with a glyph
29                      mode='vline'))
30
31 # To configure visual properties on a plot's title attribute
32 p.title.text = "Number of Papers since 2020"
33 p.title.align = "center"
34 p.title.text_color = "black"
35 p.title.text_font_size = "25px"
36 p.title.background_fill_color = "#fafafa"
37
38 # To create an html document that embeds the Bokeh plot
39 html = file_html(p, CDN, "my plot1")
40
41 # To display this html
42 displayHTML(html)
43
```

» (4) Spark Jobs

Number of Papers since 2020



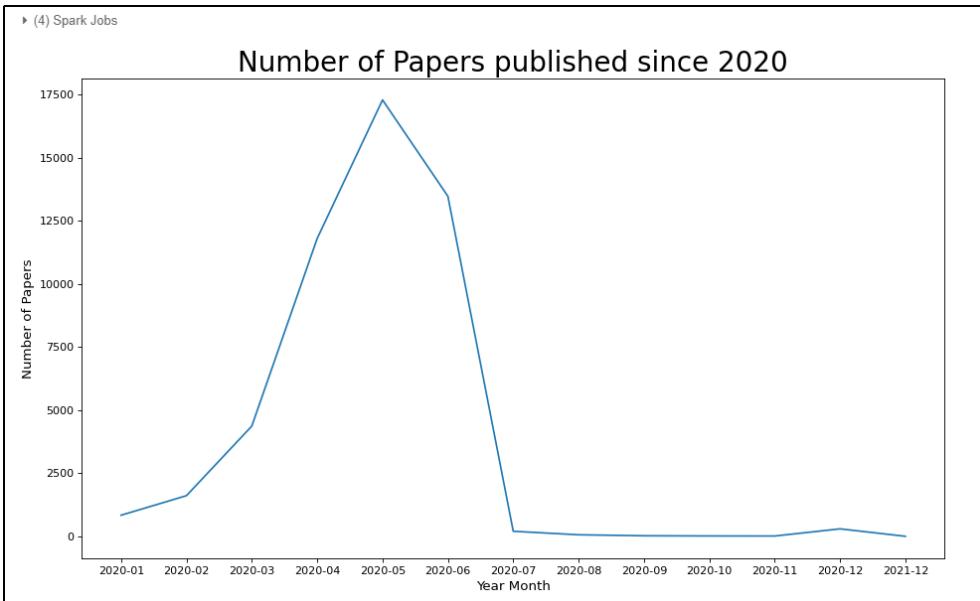
8.3 Using Dataframe created from Parquet File

Cmd 101

```
1  from pyspark.sql.functions import substring, length
2
3  df_parqt6 = df_parq.select(substring("publish_time",1,7).alias("Year_Month"), "journal") \
4      .where(substring(df_parq["publish_time"],1,4) >= '2020') \
5      .where(substring(df_parq["publish_time"],5,1) == '-') \
6      .groupBy("Year_Month") \
7      .count() \
8      .sort("Year_Month", ascending=True) \
9      .withColumnRenamed("count", "Num_of_Papers")
10
11 df_parqt6.show()
```

- Seaborn plot -

```
1  import seaborn as sns
2  import matplotlib.pyplot as plt
3  from matplotlib.pyplot import figure
4
5  figure(figsize=(14, 8), dpi=80)
6
7  names = []
8  values = []
9
10 for item in df_parqt6.rdd.collect():
11     names.append(str(item[0]))
12     values.append(item[1])
13
14 # Creating the Data Dictionary for X & Y (tooltips)
15 data_plot = {
16     "x": names,
17     "y": values
18 }
19
20 sns.lineplot(x = 'x', y = 'y', data=data_plot, ci=True)
21 sns.set_style("darkgrid")
22
23 plt.xlabel("Year Month", fontsize= 12)
24 plt.ylabel("Number of Papers", fontsize= 12)
25 plt.title("Number of Papers published since 2020", fontsize= 25)
26
27 plt.show()
```



3.6.3 HiveQL

8.1 Using Permanent Tables

Cmd 79

```
1 SELECT
2   SUBSTR(publish_time, 1, 7) as `Year Month`,
3   COUNT(*) as `Number of papers published`
4 FROM
5   ${hivevar:db_name}.${hivevar:metadata_table}
6 WHERE
7   publish_time > '2020'
8 GROUP BY
9   `Year Month`
10 ORDER BY
11   `Year Month` ASC
```

8.2 Using Temp View

Cmd 81

```
1 SELECT
2   SUBSTR(publish_time, 1, 7) as `Year Month`,
3   COUNT(*) as `Number of papers published`
4 FROM
5   temp_${hivevar:metadata_table}
6 WHERE
7   publish_time > '2020'
8 GROUP BY
9   `Year Month`
10 ORDER BY
11   `Year Month` ASC
```

8.3 Using Table from Parquet File

Cmd 83

```
1 SELECT
2   SUBSTR(publish_time, 1, 7) as `Year Month`,
3   COUNT(*) as `Number of papers published`
4 FROM
5   ${hivevar:db_name}.parq_${hivevar:metadata_table}
6 WHERE
7   publish_time > '2020'
8 GROUP BY
9   `Year Month`
10 ORDER BY
11   `Year Month` ASC
```

3.6.4 PySpark – RDD

```
1 task6_rdd = cord19_rdd.filter(lambda row: row.publish_time is not None) \
2   .map(lambda row: (row.publish_time[0:7], 1)) \
3   .reduceByKey(lambda v1,v2: v1+v2) \
4   .filter(lambda line: line[0] > '2020' and line[0][4:5] == '-')
5   .sortByKey(ascending=True) \
6   .persist()
7
8 task6_rdd.take(30)
9
```

```

1 # Setting up fields for Data Visualization / Plotting
2
3 names = []
4 values = []
5
6 for item in task6_rdd.collect():
7     names.append(str(item[0]))
8     values.append(item[1])
9
10 ymax = max(values)
11 xpos = values.index(ymax)
12 xmax = names[xpos]
13
14 ymin = min(values)
15 xpos_1 = values.index(ymin)
16 xmin = names[xpos_1]
17
18 data = {
19     "names": names,
20     "values": values
21 }

```

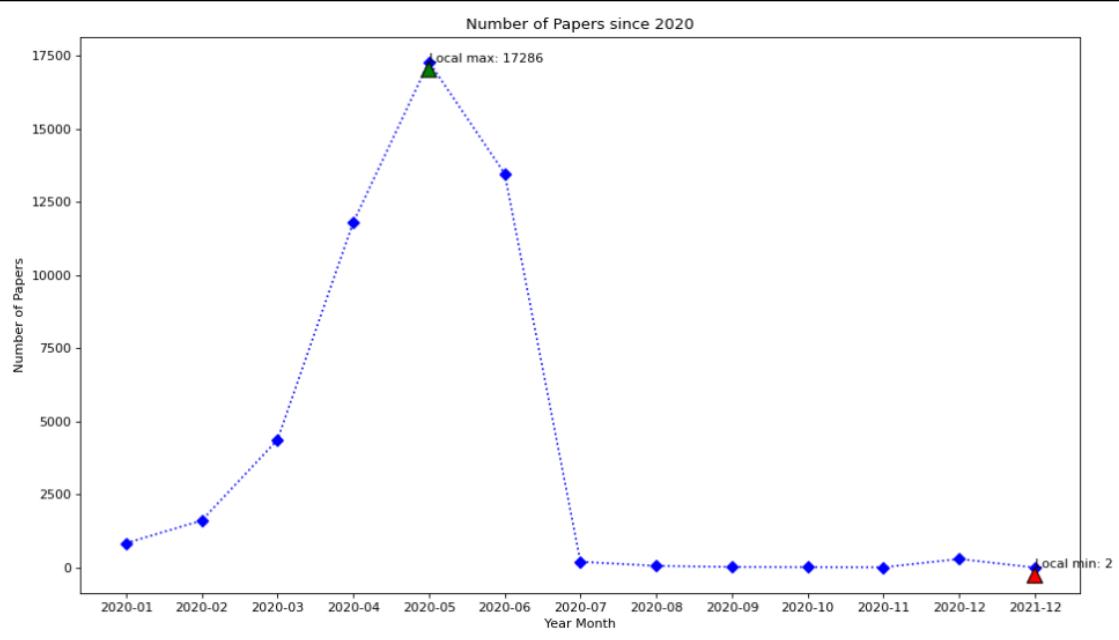
8.1 Using Matplotlib to plot the datapoints

Cmd 57

```

1 import matplotlib.pyplot as plt
2 from matplotlib.pyplot import figure
3
4 figure(figsize=(14, 8), dpi=80)
5
6 plt.plot(names, values, label="Number of Papers", marker='D', color='b', linestyle='dotted')
7
8 plt.annotate('Local max: ' + str(ymax), xy=(xmax, ymax), xytext=(xmax, ymax+5),
9             arrowprops=dict(facecolor='green'),
10            )
11
12 plt.annotate('Local min: ' + str(ymin), xy=(xmin, ymin), xytext=(xmin, ymin+5),
13             arrowprops=dict(facecolor='red'),
14            )
15
16 plt.xlabel('Year Month')
17 plt.ylabel('Number of Papers')
18 plt.title('Number of Papers since 2020')
19
20 plt.show()

```



3.6.5 AWS

3.6.5.1 AWS S3 & Athena

The screenshot shows the AWS Athena Query Editor interface. On the left, there's a sidebar with 'Data source' set to 'AwsDataCatalog' and 'Database' set to 'coviddb28'. Below these are sections for 'Tables (2)' and 'Views (8)'. The main area contains a code editor with the following SQL query:

```

1 -- Task 6. Plot the number of papers per month since 2020-01. You need to include your visualization as well
2 -- as a table of the values you have plotted for each month.
3
4 CREATE OR REPLACE VIEW cord19_task6 AS
5 SELECT
6   SUBSTR(publish_time, 1, 7) as Year_Month,
7   COUNT(*) as Published_Count
8   FROM
9   "coviddb28"."c19_2020_07_01"
10 WHERE
11   publish_time LIKE '2020%' AND publish_time <> '2020'
12 GROUP BY
13   SUBSTR(publish_time, 1, 7)
14 ORDER BY
15   SUBSTR(publish_time, 1, 7) ASC
16
17

```

Below the code editor are buttons for 'Run query', 'Save as', and 'Create'. A status bar at the bottom indicates '(Run time: 0.54 seconds, Data scanned: 0 KB)'. The bottom right corner shows 'Athena engine version 1 Release versions'.

3.6.5.2 AWS S3 & EMR

The screenshot shows a Notepad window titled '10 - Insert Data for task 6.txt'. The content is a SQL query:

```

File Edit Format View Help
-- Task 6. Plot the number of papers per month since 2020-01. You need to include your visualization as well
-- as a table of the values you have plotted for each month.

INSERT OVERWRITE DIRECTORY '${OUTPUT}'
SELECT
  SUBSTR(publish_time, 1, 7) as Year_Month,
  COUNT(*) as Published_Count
FROM
  coviddb28.c19_2020_07_01
WHERE
  publish_time >= '2020-01-01' AND publish_time LIKE '%-%-%'
GROUP BY
  SUBSTR(publish_time, 1, 7)
ORDER BY
  Year_Month ASC

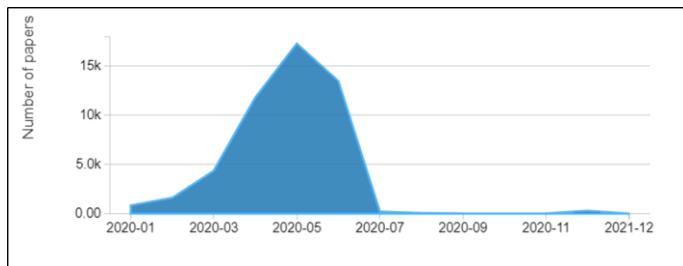
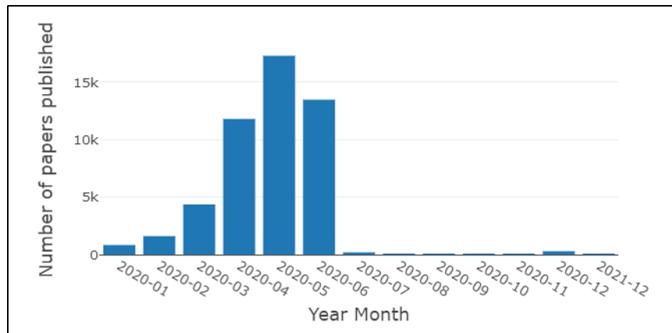
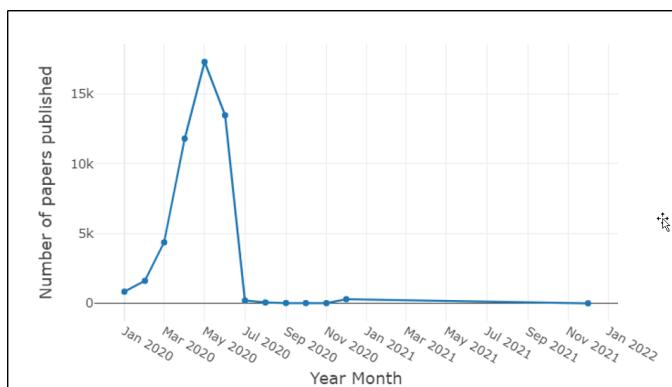
```

The screenshot shows the AWS S3 console. The path is 'Amazon S3 > jonnaivittula.outputs-bucket > AWS EMR - CORD19 tasks/ > T6_Num_of_Papers_since_2020/'. The 'Objects' tab is selected, showing a single object named '000000_0'. The object details are as follows:

Name	Type	Last modified	Size	Storage class
000000_0	-	April 27, 2021, 14:31:01 (UTC+01:00)	158.0 B	Standard

3.6.6 Result

Year_Month	Num_of_Papers
2020-01	837
2020-02	1611
2020-03	4367
2020-04	11796
2020-05	17286
2020-06	13470
2020-07	201
2020-08	62
2020-09	23
2020-10	16
2020-11	13
2020-12	297
2021-12	2

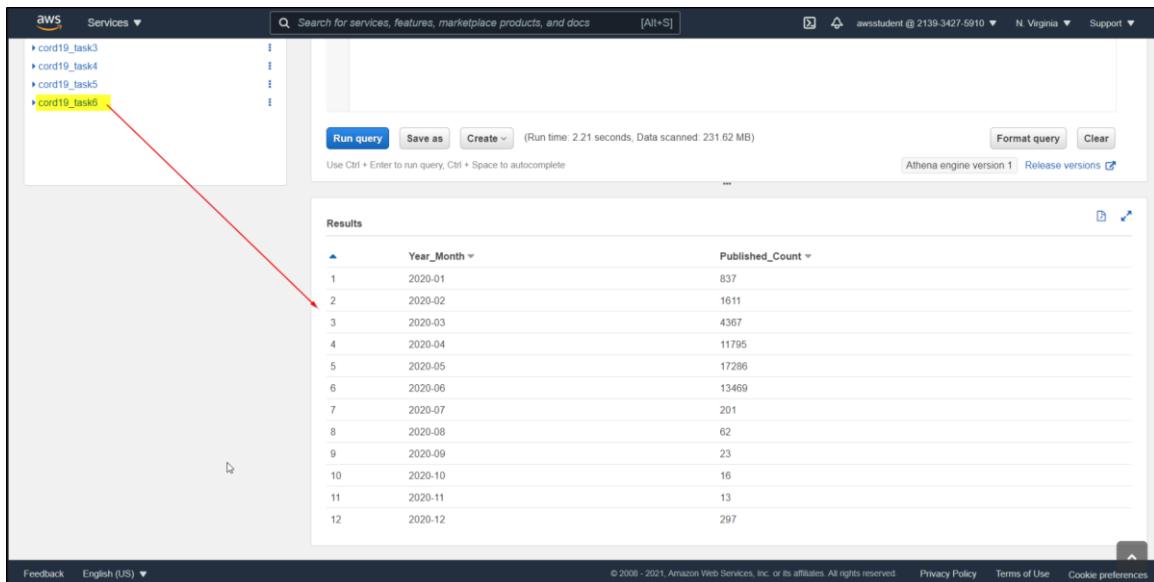


```
▶ (5) Spark Jobs
Out[22]: [('2020-01', 837),
           ('2020-02', 1611),
           ('2020-03', 4367),
           ('2020-04', 11796),
           ('2020-05', 17286),
           ('2020-06', 13470),
           ('2020-07', 201),
           ('2020-08', 62),
           ('2020-09', 23),
           ('2020-10', 16),
           ('2020-11', 13),
           ('2020-12', 297),
           ('2021-12', 2)]
```

```

000000_0 - Task 6 - Notepad
File Edit Format View Help
2020-01|837
2020-02|1611
2020-03|4367
2020-04|11795
2020-05|17286
2020-06|13469
2020-07|201
2020-08|62
2020-09|23
2020-10|16
2020-11|13
2020-12|297
2021-12|2

```



3.6.7 Result Review

- The highest number of papers were published in May-2020 (17286) and the lowest were published in 2021-12 (2).
- This dataset is as of Jul-2020, hence we see the sudden plummeting of the graph after Jun-2020.

Section 4: Additional efforts

9.1 Built-in Map chart utilization

Download and use Countrywise Alpha-3 codes from GitHub into the Databricks notebook

```
Cmd 106
1 %sh
2 curl https://raw.githubusercontent.com/ambareesh-j/world_countries/master/data/en/countries.json > /tmp/countries.json
3 ls /tmp/countries.json

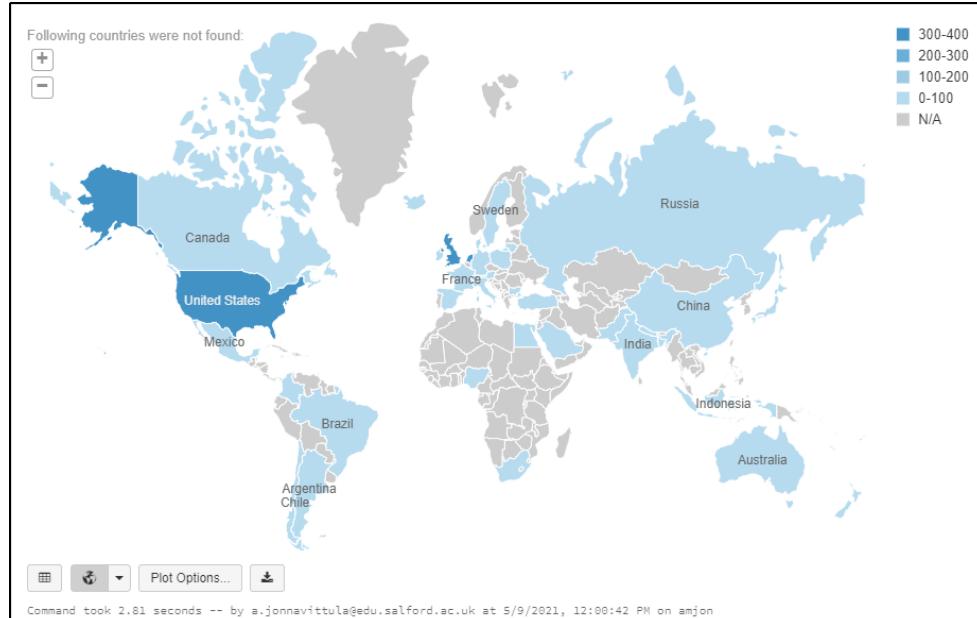
% Total    % Received % Xferd  Average Speed   Time     Time     Time  Current
          Dload  Upload Total Spent   Left Speed
0       0      0      0      0      0      0      0      0      0      0      0      0
100 11684 100 11684      0      0 59612      0      0      0      0      0      0      0
/tmp/countries.json

Command took 0.29 seconds -- by a.jonnavittula@edu.salford.ac.uk at 5/9/2021, 12:00:42 PM on amjon

Cmd 107
1 # Moving from File:/ to DBFS:/
2 dbutils.fs.cp('file:/tmp/countries.json', filepath="CountryKeys/countries.json")
3
4 dbutils.fs.ls(filepath="CountryKeys/")

Out[45]: [FileInfo(path='dbfs:/FileStore/tables/CORD19/CountryKeys/countries.json', name='countries.json', size=11684)]
Command took 0.48 seconds -- by a.jonnavittula@edu.salford.ac.uk at 5/9/2021, 12:00:42 PM on amjon
```

```
1 from pyspark.sql.functions import regexp_replace, split, explode, trim
2
3 # Read JSON file into dataframe
4 df_ctry_raw = spark.read.option("multiline","true") \
5     .json(filepath+"CountryKeys/countries.json") \
6     .persist()
7
8 df_ctry = df_ctry_raw.select("name", "alpha3").persist()
9
10 df_ctry_jrs = df_joined.select("Region", "Country", "Journal") \
11     .distinct() \
12     .groupBy("Region", "Country") \
13     .count() \
14     .withColumnRenamed("count", "Total Journals") \
15     .sort("Total Journals", ascending=False) \
16     .persist()
17
18 #Using the Country Key for Map charts to bring out Top 10 key players
19 df_ctry_key = df_ctry_jrs.join(df_ctry, df_ctry_jrs.Country == df_ctry.name, "left") \
20     .select("Country", "alpha3", "Total Journals") \
21     .sort("Total Journals", ascending=False)
22
23 # Drop from memory
24 df_ctry_raw = []
25
26 display(df_ctry_key)
```



9.2 Pivot Table to see the categories by Region/Country

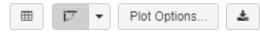
Cmd 110

```
1 df_cat_count = df_joined.select("Region", "Country", explode(split("Categories", ";")).alias("Category")) \
2     .groupBy("Region", "Country", "Category") \
3     .count()
4
5 display(df_cat_count)
```

► (10) Spark Jobs
► df_cat_count: pyspark.sql.dataframe.DataFrame = [Region: string, Country: string ... 2 more fields]

Category	Western Europe	Northern America	Asiatic Region	Eastern Europe	Latin America
1 Infectious Diseases	1768	33	18	0	24
2 Virology	1738	212	33	0	0
3 Medicine miscellaneous	1528	317	398	5	61
4 Infectious Diseases	1226	275	4	0	5
5 Medicine miscellaneous	1094	496	75	0	0
6 Public Health, Environmental and Occupational Health	866	175	6	0	0
7 Microbiology medical	740	105	4	0	28

Aggregated (by sum) in the backend.



Command took 6.83 seconds -- by a.jonnavittula@edu.salford.ac.uk at 5/9/2021, 12:00:42 PM on amjon