ISOM 671: Managing Big Data - Homework 3

Sean Jung

There are 3 numbered questions. Please submit your assignment as a single PDF or Word file by uploading it to course canvas page. You should provide: all commands, results of any commands, and answers to questions, if any.

1. (10 points) Discuss the differences between Hive, Impala, and Spark.

Hive	Impala	Spark
• Data warehouse software for querying and managing large, distributed datasets built on Hadoop • Created to make it possible for analysts with strong SQL skills to run queries on the huge volumes of data that Facebook stored in HDFS • A framework for data warehousing on top of Hadoop • Grew from a need to manage and learn from the huge volumes of data that Facebook was producing every day from its burgeoning social network	• It is a SQL engine, launched by Cloudera in 2012 • Open-source massively parallel processing interactive SQL engine (magnitude performance boost compared to Hive running on MapReduce) • Requires the database to be stored in clusters of computers that are running Apache Hadoop • Interactive, low-latency SQL queries on HDFS or HBase (In the years since Hive was created, many other SQL-on-Hadoop engines have emerged to address some of Hive's limitations	• a cluster computing framework for large-scale data processing (Unlike most of the other processing frameworks, Spark does not use MapReduce as an execution engines) instead, it uses its own distributed runtime for executing work on a cluster. • Nonetheless, Spark has many parallels with MapReduce, in terms of both API and runtime. • Spark is closely integrated with Hadoop: it can run on YARN and works with Hadoop file formats and storage backends like HDFS. • Best known for its ability to keep large working datasets in memory between jobs, allowing Spark to outperform the equivalent MapReduce workflow, where datasets are always loaded from disk

	Advantage	Disadvantage
Hive	• stable query engine • open-source engine with a vast community • uses SQL-like and Hive QL languages that are easy-to-understand by RDBMS professionals	• slow execution time compared to Impala or Spark due to its MapReduce concept • can only process structured data (not recommended for unstructured data)
Impala	• performs real-time query execution on data stored in Hadoop clusters • absence of Map Reduce makes it faster than Hive • uses HiveQL and SQL-92 (makes it easier for data analyst using RDMBS)	• only supports few type formats (RCFile, Parquet, Avro file and SequenceFile) • supports limited amount of platform (CDH, AWS and MapR)
Spark	• a fast query execution engine that can execute batch queries as well (notably, anywhere between 10 to 100 times faster than Hive) • fully compatible with hive data queries and User Defined Functions (UDF)	• requires lots of RAM power and hence, increases the usability cost • cannot be considered a stable engine due to numerous undergoing development

- Load the NY Taxi tripdata.csv from canvas to your S3 bucket and from there into Hive and Spark.

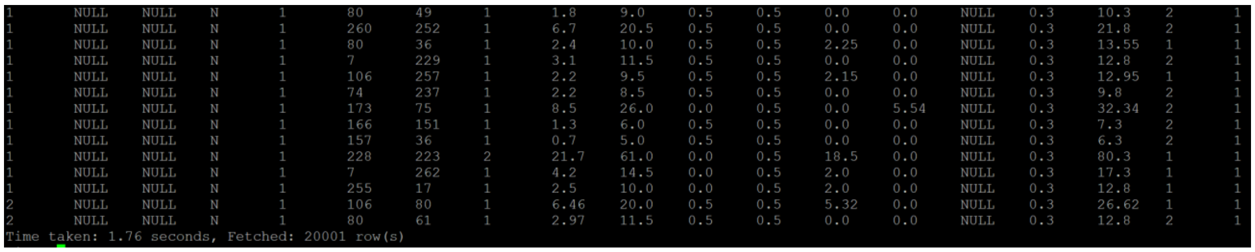
HIVE:

```
hive

DROP TABLE IF EXISTS nyTaxi;

CREATE EXTERNAL TABLE nyTaxi (VendorID INT, lpep_pickup_datetime DATE,
lpep_dropoff_datetime DATE, store_and_fwd_flag STRING, RatecodeID INT, PULocationID
INT, DOLocationID INT, passenger_count INT, trip_distance FLOAT, fare_amount FLOAT,
extra FLOAT, mta_tax FLOAT, tip_amount FLOAT, tolls_amount FLOAT, ehail_fee FLOAT,
improvement_surcharge FLOAT, total_amount FLOAT, payment_type INT, trip_type INT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE
LOCATION 's3n://aws-logs-281175518142-us-east-1/homework_3/';

SELECT * FROM nyTaxi;
```

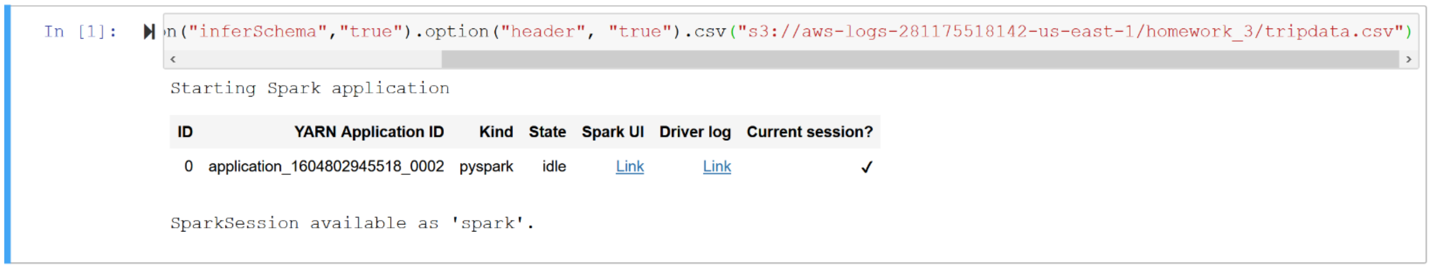


```
1       NULL    NULL    N       1       80      49      1       1.8     9.0     0.5     0.5     0.0     0.0     NULL    0.3     10.3    2       1
1       NULL    NULL    N       1       260     252     1       6.7     20.5    0.5     0.5     0.0     0.0     NULL    0.3     21.8    2       1
1       NULL    NULL    N       1       80      36      1       2.4     10.0    0.5     0.5     2.25    0.0     NULL    0.3     13.55   1       1
1       NULL    NULL    N       1       7       229     1       3.1     11.5    0.5     0.5     0.0     0.0     NULL    0.3     12.8    2       1
1       NULL    NULL    N       1       106     257     1       2.2     9.5     0.5     0.5     2.15    0.0     NULL    0.3     12.95   1       1
1       NULL    NULL    N       1       74      237     1       2.2     8.5     0.5     0.5     0.0     0.0     NULL    0.3     9.8     2       1
1       NULL    NULL    N       1       173     75      1       8.5     26.0    0.0     0.5     0.0     5.54    NULL    0.3     32.34   2       1
1       NULL    NULL    N       1       166     151     1       1.3     6.0     0.5     0.5     0.0     0.0     NULL    0.3     7.3     2       1
1       NULL    NULL    N       1       157     36      1       0.7     5.0     0.5     0.5     0.0     0.0     NULL    0.3     6.3     2       1
1       NULL    NULL    N       1       228     223     2       21.7    61.0    0.0     0.5     18.5    0.0     NULL    0.3     80.3    1       1
1       NULL    NULL    N       1       7       262     1       4.2     14.5    0.0     0.5     2.0     0.0     NULL    0.3     17.3    1       1
1       NULL    NULL    N       1       255     17      1       2.5     10.0    0.0     0.5     2.0     0.0     NULL    0.3     12.8    1       1
2       NULL    NULL    N       1       106     80      1       6.46    20.0    0.5     0.5     5.32    0.0     NULL    0.3     26.62   1       1
2       NULL    NULL    N       1       80      61      1       2.97    11.5    0.5     0.5     0.0     0.0     NULL    0.3     12.8    2       1
Time taken: 1.76 seconds, Fetched: 20001 row(s)
```

20,001 rows

SPARK:

```
nytaxi = spark.read.option("inferSchema","true").option("header",
"true").csv("s3://aws-logs-281175518142-us-east-1/homework_3/tripdata.csv")
```

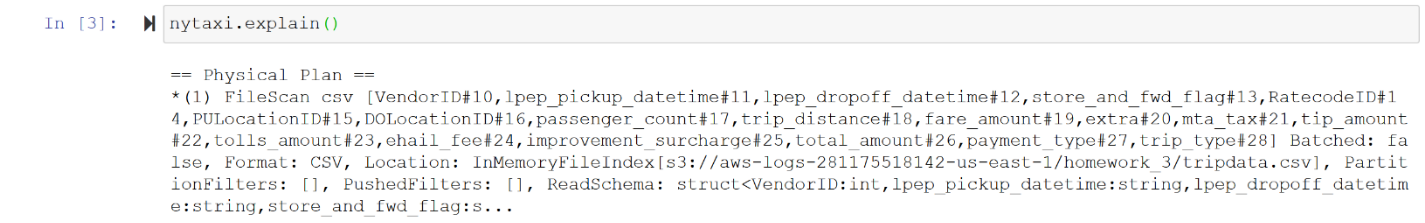


```
In [1]:  n("inferSchema","true").option("header", "true").csv("s3://aws-logs-281175518142-us-east-1/homework_3/tripdata.csv")

Starting Spark application
```

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
0	application_1604802945518_0002	pyspark	idle	Link	Link	✓

```
SparkSession available as 'spark'.
```



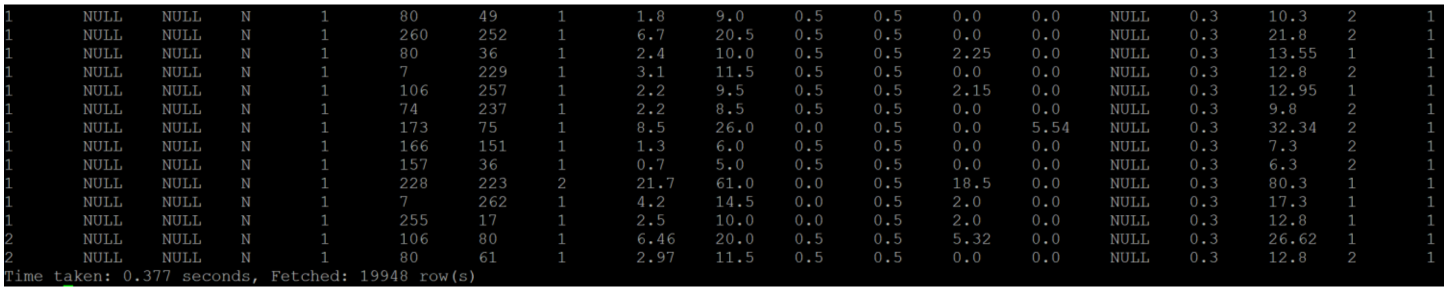
```
In [3]:  nytaxi.explain()

== Physical Plan ==
*(1) FileScan csv [VendorID#10,lpep_pickup_datetime#11,lpep_dropoff_datetime#12,store_and_fwd_flag#13,RatecodeID#1
4,PULocationID#15,DOLocationID#16,passenger_count#17,trip_distance#18,fare_amount#19,extra#20,mta_tax#21,tip_amount
#22,tolls_amount#23,ehail_fee#24,improvement_surcharge#25,total_amount#26,payment_type#27,trip_type#28] Batched: fa
lse, Format: CSV, Location: InMemoryFileIndex[s3://aws-logs-281175518142-us-east-1/homework_3/tripdata.csv], Partit
ionFilters: [], PushedFilters: [], ReadSchema: struct<VendorID:int,lpep_pickup_datetime:string,lpep_dropoff_datetim
e:string,store_and_fwd_flag:s...
```

- Write a code in Hive and Spark to find total records (for Hive, I found total record in the previous step) in the data and find number of records with rate codes: 1, 3, and 5.

HIVE:

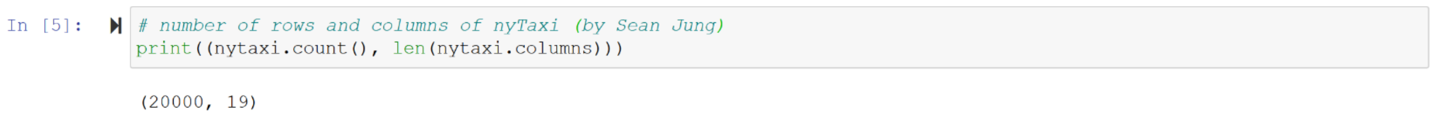
```
# Find where RatecodeID = 1, 3, 5
SELECT * FROM nyTaxi
WHERE RatecodeID IN ('1', '3' ,'5');
```



```
1       NULL    NULL    N       1       80      49      1       1.8     9.0     0.5     0.5     0.0     0.0     NULL    0.3     10.3    2       1
1       NULL    NULL    N       1       260     252     1       6.7     20.5    0.5     0.5     0.0     0.0     NULL    0.3     21.8    2       1
1       NULL    NULL    N       1       80      36      1       2.4     10.0    0.5     0.5     2.25    0.0     NULL    0.3     13.55   1       1
1       NULL    NULL    N       1       7       229     1       3.1     11.5    0.5     0.5     0.0     0.0     NULL    0.3     12.8    2       1
1       NULL    NULL    N       1       106     257     1       2.2     9.5     0.5     0.5     2.15    0.0     NULL    0.3     12.95   1       1
1       NULL    NULL    N       1       74      237     1       2.2     8.5     0.5     0.5     0.0     0.0     NULL    0.3     9.8     2       1
1       NULL    NULL    N       1       173     75      1       8.5     26.0    0.0     0.5     0.0     5.54    NULL    0.3     32.34   2       1
1       NULL    NULL    N       1       166     151     1       1.3     6.0     0.5     0.5     0.0     0.0     NULL    0.3     7.3     2       1
1       NULL    NULL    N       1       157     36      1       0.7     5.0     0.5     0.5     0.0     0.0     NULL    0.3     6.3     2       1
1       NULL    NULL    N       1       228     223     2       21.7    61.0    0.0     0.5     18.5    0.0     NULL    0.3     80.3    1       1
1       NULL    NULL    N       1       7       262     1       4.2     14.5    0.0     0.5     2.0     0.0     NULL    0.3     17.3    1       1
1       NULL    NULL    N       1       255     17      1       2.5     10.0    0.0     0.5     2.0     0.0     NULL    0.3     12.8    1       1
2       NULL    NULL    N       1       106     80      1       6.46    20.0    0.5     0.5     5.32    0.0     NULL    0.3     26.62   1       1
2       NULL    NULL    N       1       80      61      1       2.97    11.5    0.5     0.5     0.0     0.0     NULL    0.3     12.8    2       1
Time taken: 0.377 seconds, Fetched: 19948 row(s)
```

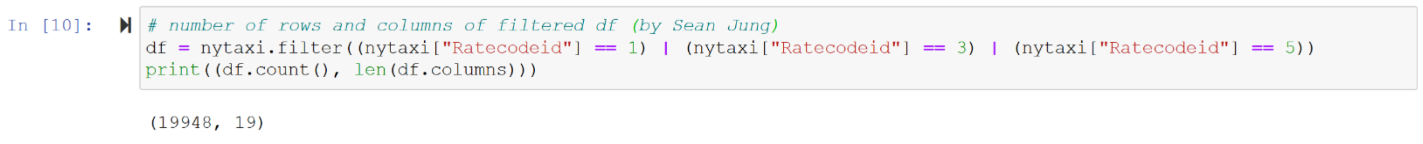
There are 19,948 records of which RatecodeID is equal to 1,3, or 5

SPARK:



```
In [5]:  # number of rows and columns of nyTaxi (by Sean Jung)
         print((nytaxi.count(), len(nytaxi.columns)))

(20000, 19)
```

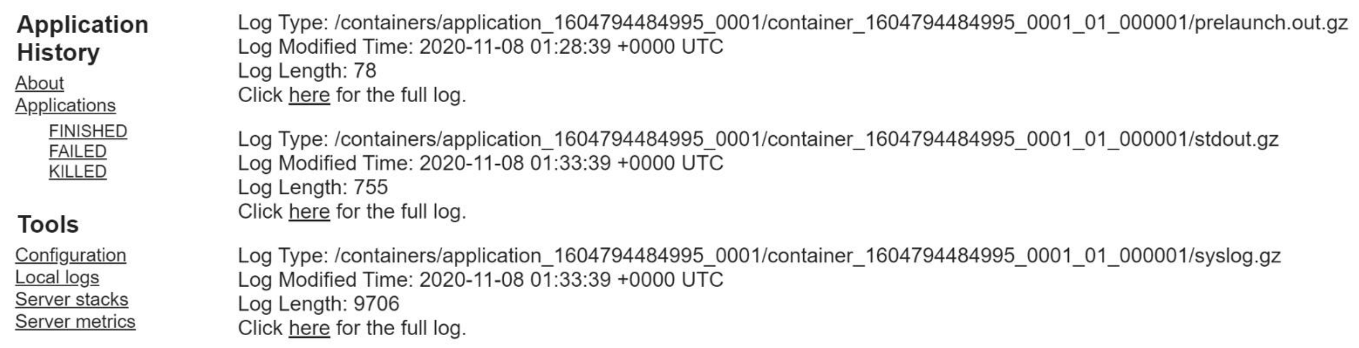


```
In [10]:  # number of rows and columns of filtered df (by Sean Jung)
          df = nytaxi.filter((nytaxi["Ratecodeid"] == 1) | (nytaxi["Ratecodeid"] == 3) | (nytaxi["Ratecodeid"] == 5))
          print((df.count(), len(df.columns)))

(19948, 19)
```

- Look at the log file in YARN timeline server and Spark history server and take a snapshot of the entry that is returning the number of records requested by your code.

Sean's YARN server log



Application History
About
Applications
FINISHED
FAILED
KILLED

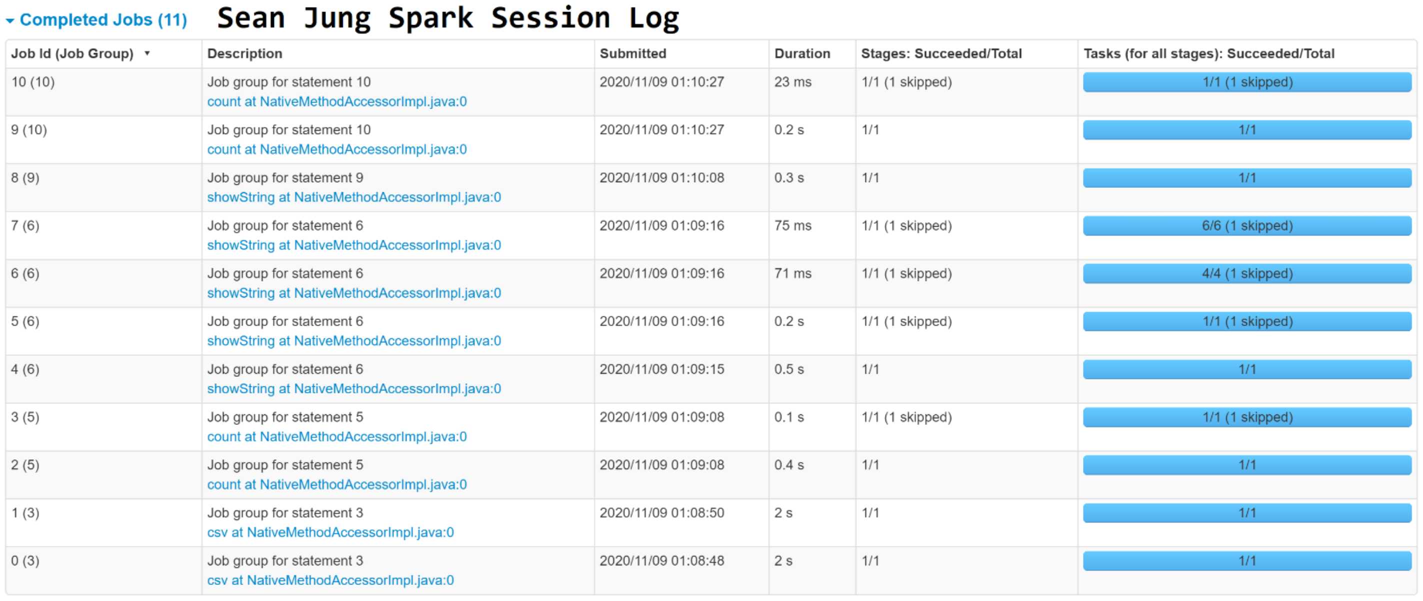
Tools
Configuration
Local logs
Server stacks
Server metrics

```
Log Type: /containers/application_1604794484995_0001/container_1604794484995_0001_01_000001/prelaunch.out.gz
Log Modified Time: 2020-11-08 01:28:39 +0000 UTC
Log Length: 78
Click here for the full log.

Log Type: /containers/application_1604794484995_0001/container_1604794484995_0001_01_000001/stdout.gz
Log Modified Time: 2020-11-08 01:33:39 +0000 UTC
Log Length: 755
Click here for the full log.

Log Type: /containers/application_1604794484995_0001/container_1604794484995_0001_01_000001/syslog.gz
Log Modified Time: 2020-11-08 01:33:39 +0000 UTC
Log Length: 9706
Click here for the full log.
```

Sean Jung Spark Session Log

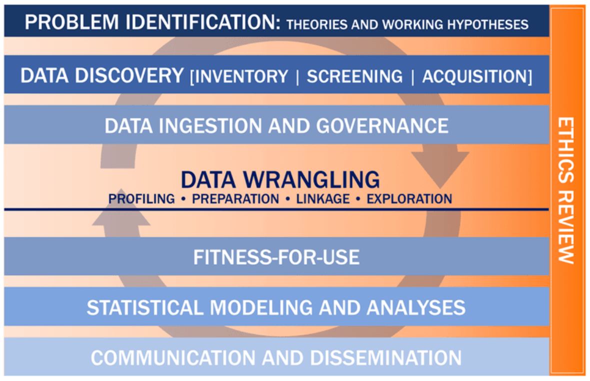


Completed Jobs (11)

Job Id (Job Group) ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
10 (10)	Job group for statement 10 count at NativeMethodAccessorImpl.java:0	2020/11/09 01:10:27	23 ms	1/1 (1 skipped)	1/1 (1 skipped)
9 (10)	Job group for statement 10 count at NativeMethodAccessorImpl.java:0	2020/11/09 01:10:27	0.2 s	1/1	1/1
8 (9)	Job group for statement 9 showString at NativeMethodAccessorImpl.java:0	2020/11/09 01:10:08	0.3 s	1/1	1/1
7 (6)	Job group for statement 6 showString at NativeMethodAccessorImpl.java:0	2020/11/09 01:09:16	75 ms	1/1 (1 skipped)	6/6 (1 skipped)
6 (6)	Job group for statement 6 showString at NativeMethodAccessorImpl.java:0	2020/11/09 01:09:16	71 ms	1/1 (1 skipped)	4/4 (1 skipped)
5 (6)	Job group for statement 6 showString at NativeMethodAccessorImpl.java:0	2020/11/09 01:09:16	0.2 s	1/1 (1 skipped)	1/1 (1 skipped)
4 (6)	Job group for statement 6 showString at NativeMethodAccessorImpl.java:0	2020/11/09 01:09:15	0.5 s	1/1	1/1
3 (5)	Job group for statement 5 count at NativeMethodAccessorImpl.java:0	2020/11/09 01:09:08	0.1 s	1/1 (1 skipped)	1/1 (1 skipped)
2 (5)	Job group for statement 5 count at NativeMethodAccessorImpl.java:0	2020/11/09 01:09:08	0.4 s	1/1	1/1
1 (3)	Job group for statement 3 csv at NativeMethodAccessorImpl.java:0	2020/11/09 01:08:50	2 s	1/1	1/1
0 (3)	Job group for statement 3 csv at NativeMethodAccessorImpl.java:0	2020/11/09 01:08:48	2 s	1/1	1/1

2. (10 points) Read the article Doing Data Science (https://hdsr.mitpress.mit.edu/pub/hnptx6lq/release/8).

- Assuming you are the CIO at Coke, what database systems (Hive, Impala, and Spark) would you implement for different functional groups (e.g. marketing, operations, and finance) within the organization. Please try to align your answer to the data science framework (fig 1) as presented in the article (Note: the points will be allocated not on the correctness of the answer but the reasoning and analysis behind the answer)


PROBLEM IDENTIFICATION: THEORIES AND WORKING HYPOTHESES
DATA DISCOVERY [INVENTORY | SCREENING | ACQUISITION]
DATA INGESTION AND GOVERNANCE
DATA WRANGLING
PROFILING • PREPARATION • LINKAGE • EXPLORATION
FITNESS-FOR-USE
STATISTICAL MODELING AND ANALYSES
COMMUNICATION AND DISSEMINATION
ETHICS REVIEW

Introduction:

Hive, Impala, and Spark are all part of the SQL interfaces on Hadoop. Hive is mainly used for querying, processing, and maintaining structured data

Impala is used for Business Intelligence and SQL analytics; Lastly, Spark is a processing engine that’s used for structured data processing and procedure development.

Marketing:

Coke also might have had limited insights to thousands of marketing campaign across channels. For example, clients may want real-time campaign updates with 3-second SLA or limited self-service BI restricts it from scaling the data type demands.

By using HiveQL and SQL-92, Impala performs real-time query execution on data stored in Hadoop clusters. In this instance, Impala would be the best option for the marketing team at Coke. Even though fundamental concept under which it performs queries may be similar to that of Hive, Impala logs files so much faster.

Impala is widely used for business intelligence tool, allowing the marketing team to quickly form strategic decisions about customer segmentation, targeting, advertisement selection, product selection, location selection, etc.

Additionally, Impala is most likely to maximize the efficiency in the process of maintaining and processing marketing data that entails analysis of consumer demographic and preferences and outputting log files that maintains the integrity and security of private, sensitive data such as customer contact information and consumer preferences.

Lastly, by adopting such Impala infrastructure in the process of marketing, Coke can also build digital marketing platform for 360-degree customer view that improves query performance from minutes to seconds to meet SLAs, enhance modeling with combined online and offline data, and optimize real-time data through interactive, self-service access.

Operations:

When it comes to the operation task where the speed, ease of use, extensibility, and modularity of the query matters much more, Spark is most likely to be the safe bet. Not only the operations within Spark can be applied across many types of workload tasks but Spark can be also supported in a variety of supported programming languages (such as Python, R, SQL, Scala, Java).

Also, Spark's versatility to read/import data from the diverse locations can be useful as the data structure of the operation task is most likely to be different from each other attributed from having different location or different divisions of the same company.

Even though Spark process the data in RAM that could be problematic in the case of data leakage, the consequences of such adversity may not be too severe compared to that of losing secure financial data.

Lastly, operational data is primarily considered in the fitness-for-use, statistical modelling, and analysis phases, allowing Coke and other firms to figure out which parts of their operations experience issue and hence requires improvement.

Finance:

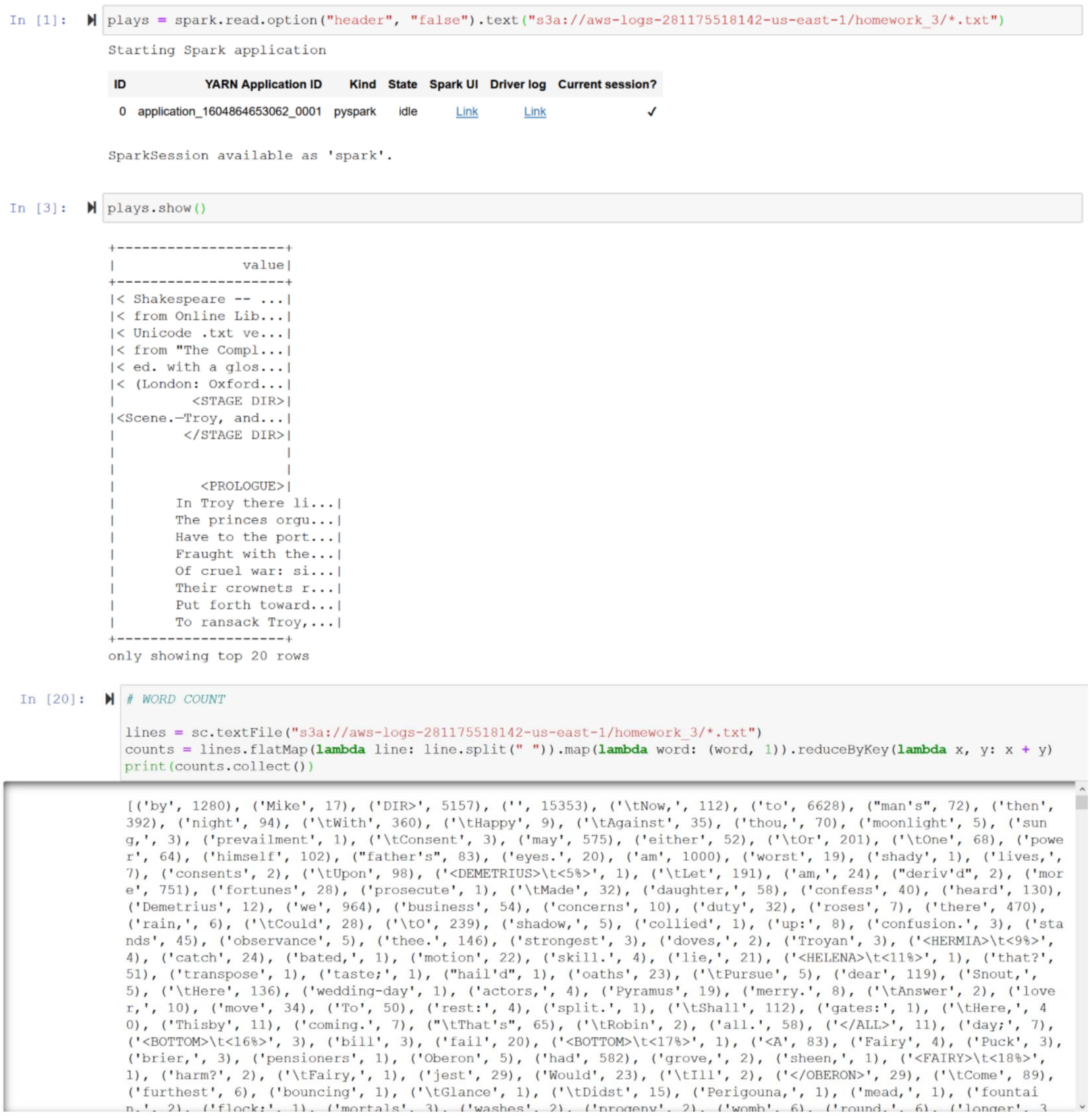
Coke previously might have needed to collect, process, and analyze financial data from growing sales/revenue of their brands efficiently and securely. Hence, it is likely that they also may have experienced the difficulty of not being able to keep up with the speed of expansion for the structured data demands.

In this instance, Hive is most likely to be the best one to use. Most financial data are structured, and Hive is efficient in maintaining and processing the structured data. Not only Hive provides data warehouse software for querying and managing large, it can also directly yield outputs to a file system without having to require extensive software development background to operate (as long as one is proficient in SQL languages), entailing that users (who are savvy in finance but not so much in computer science) can easily and safely query financial data from or to the Hive database.

Additionally, Hive will provide storage to the large amount of financial data Coke generates with its data warehousing feature. Financial data is mostly used in the data ingestion and governance phase. Once financial data is stored into a database, it is seldom analyzed afterwards. It may go through all the other phases, but the ingestion and governance are the main parts of the process where Hive will be primarily utilized. Thus, by implementing Hive database infrastructure, they would be able to securely integrate thousands of financial data through unified enterprise data hub.

3. (10 points) Load multiple Shakespeare plays in your S3 bucket and load all of those in Spark. (data source: https://github.com/Pseudomanifold/Shakespeare/tree/master/Plays/comedies)

- Write a PySpark code that calculates word frequency across all documents
- Write a PySpark code that find the frequency for word "love" in each document and results are presented in descending order of count



In [1]:
```
plays = spark.read.option("header", "false").text("s3a://aws-logs-281175518142-us-east-1/homework_3/*.txt")
```

```
Starting Spark application
```

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
0	application_1604864653062_0001	pyspark	idle	Link	Link	✓

```
SparkSession available as 'spark'.
```

In [3]:
```
plays.show()
```

```
+--------------------+
|               value|
+--------------------+
|< Shakespeare -- ...|
|< from Online Lib...|
|< Unicode .txt ve...|
|< from "The Compl...|
|< ed. with a glos...|
|< (London: Oxford...|
|          <STAGE DIR>|
|<Scene.—Troy, and...|
|         </STAGE DIR>|
|                    |
|                    |
|           <PROLOGUE>|
|        In Troy there li...|
|        The princes orgu...|
|        Have to the port...|
|        Fraught with the...|
|        Of cruel war: si...|
|        Their crownets r...|
|        Put forth toward...|
|        To ransack Troy,...|
+--------------------+
only showing top 20 rows
```

In [20]:
```
# WORD COUNT

lines = sc.textFile("s3a://aws-logs-281175518142-us-east-1/homework_3/*.txt")
counts = lines.flatMap(lambda line: line.split(" ")).map(lambda word: (word, 1)).reduceByKey(lambda x, y: x + y)
print(counts.collect())
```

```
[('by', 1280), ('Mike', 17), ('DIR>', 5157), ('', 15353), ('\tNow,', 112), ('to', 6628), ("man's", 72), ('then',
392), ('night', 94), ('\tWith', 360), ('\tHappy', 9), ('\tAgainst', 35), ('thou,', 70), ('moonlight', 5), ('sun
g,', 3), ('prevailment', 1), ('\tConsent', 3), ('may', 575), ('either', 52), ('\tOr', 201), ('\tOne', 68), ('powe
r', 64), ('himself', 102), ("father's", 83), ('eyes.', 20), ('am', 1000), ('worst', 19), ('shady', 1), ('lives,',
7), ('consents', 2), ('\tUpon', 98), ('<DEMETRIUS>\t<5%>', 1), ('\tLet', 191), ('am,', 24), ("deriv'd", 2), ('mor
e', 751), ('fortunes', 28), ('prosecute', 1), ('\tMade', 32), ('daughter,', 58), ('confess', 40), ('heard', 130),
('Demetrius', 12), ('we', 964), ('business', 54), ('concerns', 10), ('duty', 32), ('roses', 7), ('there', 470),
('rain,', 6), ('\tCould', 28), ('\tO', 239), ('shadow,', 5), ('collied', 1), ('up:', 8), ('confusion.', 3), ('sta
nds', 45), ('observance', 5), ('thee.', 146), ('strongest', 3), ('doves,', 2), ('Troyan', 3), ('<HERMIA>\t<9%>',
4), ('catch', 24), ('bated,', 1), ('motion', 22), ('skill.', 4), ('lie,', 21), ('<HELENA>\t<11%>', 1), ('that?',
51), ('transpose', 1), ('taste;', 1), ("hail'd", 1), ('oaths', 23), ('\tPursue', 5), ('dear', 119), ('Snout,',
5), ('\tHere', 136), ('wedding-day', 1), ('actors,', 4), ('Pyramus', 19), ('merry.', 8), ('\tAnswer', 2), ('love
r,', 10), ('move', 34), ('To', 50), ('rest:', 4), ('split.', 1), ('\tShall', 112), ('gates:', 1), ('\tHere,', 4
0), ('Thisby', 11), ('coming.', 7), ("\tThat's", 65), ('\tRobin', 2), ('all.', 58), ('</ALL>', 11), ('day;', 7),
('<BOTTOM>\t<16%>', 3), ('bill', 3), ('fail', 20), ('<BOTTOM>\t<17%>', 1), ('<A', 83), ('Fairy', 4), ('Puck', 3),
('brier,', 3), ('pensioners', 1), ('Oberon', 5), ('had', 582), ('grove,', 2), ('sheen,', 1), ('<FAIRY>\t<18%>',
1), ('harm?', 2), ('\tFairy,', 1), ('jest', 29), ('Would', 23), ('\tIll', 2), ('</OBERON>', 29), ('\tCome', 89),
('furthest', 6), ('bouncing', 1), ('\tGlance', 1), ('\tDidst', 15), ('Perigouna,', 1), ('mead,', 1), ('fountai
n.', 2), ('flock:', 1), ('mortals', 3), ('washes', 2), ('progeny', 2), ('womb', 6), ('round.', 6), ('longer', 3
```

In [21]:

```
# Count 'Love' in a whole document
love = plays.filter(plays.value.contains("love"))
love.count()
```

```
1194
```

```
# Question 3.3 - Count the word 'love' by each document [SEAN JUNG]

# 1. Assign each files
one = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/A Midsummer-Night's
two = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/All's Well That End
three = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/As You Like It.tx
four = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/Cymbeline.txt")
five = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/Love's Labour's Lo
six = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/Measure for Measure
seven = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/Much Ado about No
eight = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/Pericles, Prince
nine = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/The Comedy of Erro
ten = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/The Merchant of Ven
eleven = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/The Merry Wives
twelve = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/The Taming of th
thirteen = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/The Tempest.tx
fourteen = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/The Two Gentle
fifteen = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/The Winter's Ta
sixteen = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/Troilus and Cre
seventeen = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/Twelfth-Night

# Put all of them in a list
collection = [one, two, three, four, five, six, seven, eight, nine, ten, eleven, twelve, thirteen, fourteen, fifteen


# Initialize
x=[]

# Define a mapper function
def f(iterator):
    for y in iterator:
        z = (y.filter(y.value.contains('love')).count())
        x.append(z)

f(collection)


# Call a function to sort them in descending order
print(sorted(x, reverse=True))
```

```
[163, 141, 122, 99, 94, 88, 78, 69, 66, 63, 46, 36, 36, 30, 29, 17, 17]
```

Or I can do it manually one by one.

```
In [17]: # Counting 'Love' from 'The Two Gentlemen of Verona.txt'
fourteen = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/The Two Gentlemen of Veron
a.txt")
fourteen = fourteen.filter(fourteen.value.contains("love"))
fourteen.count()
```

```
163
```

```
In [4]: # Counting 'Love' from 'A Midsummer-Night's Dream.txt'
one = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/A Midsummer-Night's Dream.txt")
one = one.filter(one.value.contains("love"))
one.count()
```

```
141
```

```
In [6]: # Counting 'Love' from 'As You Like It.txt'
three = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/As You Like It.txt")
three = three.filter(three.value.contains("love"))
three.count()
```

```
122
```

```
In [8]: # Counting 'Love' from 'Love's Labour's Lost.txt'
five = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/Love's Labour's Lost.txt")
five = five.filter(five.value.contains("love"))
five.count()
```

```
99
```

```
In [10]: # Counting 'Love' from 'Much Ado about Nothing.txt'
seven = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/Much Ado about Nothing.txt")
seven = seven.filter(seven.value.contains("love"))
seven.count()
```

```
94
```

```
In [20]: # Counting 'Love' from 'Twelfth-Night; or What You Will.txt'
seventeen = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/Twelfth-Night; or What Yo
u Will.txt")
seventeen = seventeen.filter(seventeen.value.contains("love"))
seventeen.count()
```

```
88
```

```
In [19]: # Counting 'Love' from 'Troilus and Cressida.txt'
sixteen = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/Troilus and Cressida.txt")
sixteen = sixteen.filter(sixteen.value.contains("love"))
sixteen.count()
```

```
78
```

```
In [15]: # Counting 'Love' from 'The Taming of the Shrew.txt'
twelve = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/The Taming of the Shrew.tx
t")
twelve = twelve.filter(twelve.value.contains("love"))
twelve.count()
```

```
69
```

```
In [5]: # Counting 'Love' from 'All's Well That Ends Well.txt'
        two = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/All's Well That Ends Well.txt")
        two = two.filter(two.value.contains("love"))
        two.count()

        66
```

```
In [13]: # Counting 'Love' from 'The Merchant of Venice.txt'
         ten = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/The Merchant of Venice.txt")
         ten = ten.filter(ten.value.contains("love"))
         ten.count()

         63
```

```
In [14]: # Counting 'Love' from 'The Merry Wives of Windsor.txt'
         eleven = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/The Merry Wives of Windsor.t
         xt")
         eleven = eleven.filter(eleven.value.contains("love"))
         eleven.count()

         46
```

```
In [7]: # Counting 'Love' from 'Cymbeline.txt'
        four = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/Cymbeline.txt")
        four = four.filter(four.value.contains("love"))
        four.count()

        36
```

```
In [18]: # Counting 'Love' from 'The Winter's Tale.txt'
         fifteen = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/The Winter's Tale.txt")
         fifteen = fifteen.filter(fifteen.value.contains("love"))
         fifteen.count()

         36
```

```
In [11]: # Counting 'Love' from 'Pericles, Prince of Tyre.txt'
         eight = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/Pericles, Prince of Tyre.tx
         t")
         eight = eight.filter(eight.value.contains("love"))
         eight.count()

         30
```

```
In [9]: # Counting 'Love' from 'Measure for Measure.txt'
        six = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/Measure for Measure.txt")
        six = six.filter(six.value.contains("love"))
        six.count()

        29
```

```
In [12]: # Counting 'Love' from 'The Comedy of Errors.txt'
         nine = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/The Comedy of Errors.txt")
         nine = nine.filter(nine.value.contains("love"))
         nine.count()

         17
```

```
In [16]: # Counting 'Love' from 'The Tempest.txt'
         thirteen = spark.read.option("header", "false").text("s3://aws-logs-281175518142-us-east-1/homework_3/The Tempest.txt")
         thirteen = thirteen.filter(thirteen.value.contains("love"))
         thirteen.count()

         17
```