Assignment - 3

Apache Spark, Real-Time data pipelines, and analysis

Mr. Rahul Midha
B00766975

Mr. Niravsinh Jadeja
B00789139

Task Description:

At present looking at the size of data that we produce every day, it requires a lot of time if we are dealing with it via normal SQL database *[1]*. The whole process gets cumbersome which would result in wastage of resources. The alternative way to get rid of all these problems is to use data processing frameworks such as Apache Sparks, Apache Storm, Samza, Flink *[2]* and so on. Any commercial company which deals with Big Data *[3]* uses such type of framework for their fast and smooth operations. Apache Sparks *[4]* is known for its in-memory cluster computing feature which provides high processing speed. Here, in this assignment, we have used Apache Sparks with Python *[5]* Integration for performing various data operations. Further, we have utilized IBM's Data Science Experience cloud facility *[6]* and local implementation of Apache Sparks & Python for the solution of the assignment.

Application Queries with Response:

Task 1:

Loading the Data

In [1]:
```
from nltk import word_tokenize
from nltk.corpus import stopwords
import time
from nltk import PorterStemmer
from pyspark import SparkContext
import re
import operator
from operator import add

#Loading the Data
start_time = time.time()
path_1 = "C:\\Users\\nirav\\Downloads\\WordCountData.txt"
sc = SparkContext()
textfile = sc.textFile(path_1)
singleListData = (textfile.collect())
```

Task 1.1 Stopping Process with time response

Part 1: Stoping Process with Response Time

In [2]:
```
time1 = time.time()
stopWords = set(stopwords.words('english'))
words = word_tokenize(str(singleListData))
wordsFiltered = []

for w in words:
    if w not in stopWords:
        wordsFiltered.append(w)

print("PART 1_STOPPING_RESPONSE TIME: %s SECONDS" % (time.time() - time1))
```

```
PART 1_STOPPING_RESPONSE TIME: 0.10533380508422852 SECONDS
```

Task 1.2 Cleaning the data part, in which we have removed selected characters

Part 2: Cleaning the data, removing unnecessary characters with Response Time

In [3]:
```
time2 = time.time()
wordsToRemove = [",","'nan", "]", "'", ".", ";", "''", "[", "``", "'I", "'of", "'is", "'are", "'in", "'In", "'by", "'the", "'and"
for i in range(0, len(wordsToRemove)):
    wordsFiltered = [x for x in wordsFiltered if x != wordsToRemove[i]]
print("PART 2_CLEANING DATA_RESPONSE TIME: %s SECONDS" % (time.time() - time2))
```

```
PART 2_CLEANING DATA_RESPONSE TIME: 0.02356410026550293 SECONDS
```

Task 1.3 Stemming Process with time response

Part 3: Stemming Process with Response time

In [4]:
```
time3 = time.time()
stemmedWords = PorterStemmer().stem(str(wordsFiltered))
print("PART 3_STEMMING_RESPONSE TIME: %s SECONDS" % (time.time() - time3))
```

```
PART 3_STEMMING_RESPONSE TIME: 0.0015034675598144531 SECONDS
```

Task 1.4 Counting of words, lines, and lines in the data using Sparks

Part 4: number of lines, characters and occurance of characters with response time

In [5]:
```
time4 = time.time()
stemmedWords = stemmedWords.split() #converting string to list
text2 = sc.parallelize(stemmedWords)
print ('NUMBER OF LINES IN FILE ARE: %s' % textfile.count())
chars = textfile.map(lambda s: len(s)).reduce(add)
print ('NUMBER OF CHARACTERS IN FILE ARE: %s' % chars)

words = text2.flatMap(lambda line: re.split('\W+', line.lower().strip())) #use of flatmap
words = words.filter(lambda x: len(x) > 3)
words = words.map(lambda w : (w,1)) #use of mapping
words = words.reduceByKey(add) #reduce by key
print("PART 4_RESPONSE TIME: %s SECONDS" % (time.time() - time4))
```

```
NUMBER OF LINES IN FILE ARE: 1449
NUMBER OF CHARACTERS IN FILE ARE: 59259
PART 4_RESPONSE TIME: 4.048681735992432 SECONDS
```

Task 1.5 Showing 50 words with their count value and whole program response time

Printing most 50 occurance words

In [6]:
```
print(sorted(words.take(50),key=operator.itemgetter(1),reverse=True))
```

```
[('would', 18), ('like', 15), ('spirit', 12), ('when', 12), ('ever', 11), ('where', 10), ('behold', 9), ('mighty', 7), ('spher
e', 7), ('free', 7), ('matter', 6), ('work', 6), ('beyond', 5), ('hast', 5), ('bosom', 4), ('guide', 4), ('stood', 3), ('broa
d', 3), ('plac', 3), ('store', 2), ('brows', 2), ('harmony', 2), ('dull', 2), ('cord', 2), ('dart', 2), ('bend', 2), ('return
s', 2), ('glorious', 2), ('space', 2), ('bespake', 2), ('sheds', 1), ('nathless', 1), ('benign', 1), ('remaining', 1), ('dive
r', 1), ('herb', 1), ('serve', 1), ('rain', 1), ('imagination', 1), ('entangled', 1), ('admiration', 1), ('binds', 1), ('order
s', 1), ('sometimes', 1), ('creature', 1), ('advent', 1), ('reveal', 1), ('fearless', 1), ('plough', 1), ('loosen', 1)]
```

The time consumed for whole program

In [7]:
```
print("WHOLE PROGRAM_RESPONSE TIME: %s SECONDS" % (time.time() - start_time))
```

```
WHOLE PROGRAM_RESPONSE TIME: 14.204959630966187 SECONDS
```

Task 2:

Task 2.1: Loading the cleaned data with Pandas

Loading the Dataset

In [1]:
```
import time
import sys
import types
import pandas as pd
from botocore.client import Config
import ibm_boto3

def __iter__(self): return 0

# @hidden_cell
# The following code accesses a file in your IBM Cloud Object Storage. It includes your credentials.
# You might want to remove those credentials before you share your notebook.
client_1f6f2e6280144b23bb3fd53cbdb85cdd = ibm_boto3.client(service_name='s3',
    ibm_api_key_id='H3RY-cRAHjH4W0KbIwe1nQnJy4oM49dgBZOdz0R3AUwB',
    ibm_auth_endpoint="https://iam.ng.bluemix.net/oidc/token",
    config=Config(signature_version='oauth'),
    endpoint_url='https://s3-api.us-geo.objectstorage.service.networklayer.com')

body = client_1f6f2e6280144b23bb3fd53cbdb85cdd.get_object(Bucket='qwerty28b58f4c38184cafbd941800e4d7f669',Key='NationalNames.csv')['Body']
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__, body )

df_data_1 = pd.read_csv(body)
df_data_1.head()
```

Out[1]:

	Id	Name	Year	Gender	Count
0	1	Mary	1880	F	7065
1	2	Anna	1880	F	2604
2	3	Emma	1880	F	2003
3	4	Elizabeth	1880	F	1939
4	5	Minnie	1880	F	1746

Task 2.2: Connecting Pandas data frame with SQL

Connecting Pandas Dataframe with SQL

In [2]:
```
time1 = time.time()
sqlContext = SQLContext(sc)
sdf = sqlContext.createDataFrame(df_data_1)
sdf.printSchema()
print("response time: %s seconds" % (time.time() - time1))
```

```
root
 |-- Id: long (nullable = true)
 |-- Name: string (nullable = true)
 |-- Year: long (nullable = true)
 |-- Gender: string (nullable = true)
 |-- Count: long (nullable = true)

response time: 110.997600079 seconds
```

Task 2.3: Find the total number of birth registered in a year

Problem 3.1 - Total number of birth registered in a year

In [3]:
```
time2 = time.time()
sdf.groupby(['Year'])\
.agg({"Count": "Sum"})\
.sort("Year", ascending=True)\
.show()
print("response time: %s seconds" % (time.time() - time2))
```

Response:

```
+----+----------+
|Year|sum(Count)|
+----+----------+
|1880|    201484|
|1881|    192699|
|1882|    221538|
|1883|    216950|
|1884|    243467|
|1885|    240855|
|1886|    255319|
|1887|    247396|
|1888|    299480|
|1889|    288950|
|1890|    301402|
|1891|    286678|
|1892|    334383|
|1893|    325223|
|1894|    338694|
|1895|    351028|
|1896|    357490|
|1897|    346960|
|1898|    381463|
|1899|    339235|
+----+----------+
only showing top 20 rows

response time: 14.9661710262 seconds
```

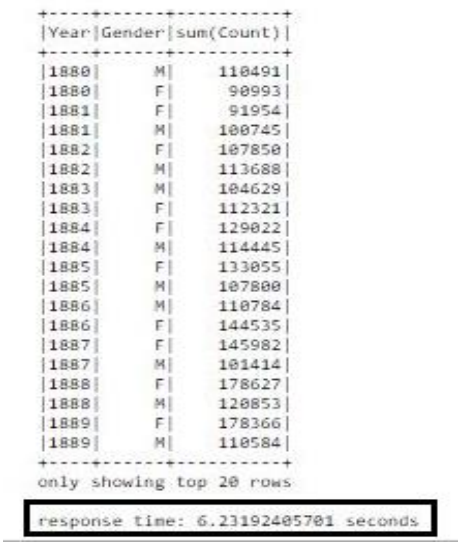
Response Time: 14.96 seconds.

Task 2.4: Find the total number of births registered in a year by gender

Problem 3.2 - Total number of births registered in a year by gender

In [4]:
```
time3 = time.time()
sdf.groupby(['Year', 'Gender'])\
.agg({"Count": "Sum"})\
.sort("Year", ascending=True)\
.show()
print("response time: %s seconds" % (time.time() - time3))
```

Response:



```
+----+------+-----------+
|Year|Gender|sum(Count)|
+----+------+-----------+
|1880|     M|     110491|
|1880|     F|      90993|
|1881|     F|      91954|
|1881|     M|     100745|
|1882|     F|     107850|
|1882|     M|     113688|
|1883|     M|     104629|
|1883|     F|     112321|
|1884|     F|     129022|
|1884|     M|     114445|
|1885|     F|     133055|
|1885|     M|     107800|
|1886|     M|     110784|
|1886|     F|     144535|
|1887|     F|     145982|
|1887|     M|     101414|
|1888|     F|     178627|
|1888|     M|     120853|
|1889|     F|     178366|
|1889|     M|     110584|
+----+------+-----------+
only showing top 20 rows

response time: 6.23192405701 seconds
```

Response Time: 6.23 seconds

Task 2.5: Input a year and populate top 5 most popular names registered that year

Problem 3.3 - Input a year and populate top 5 most popular names registered that year

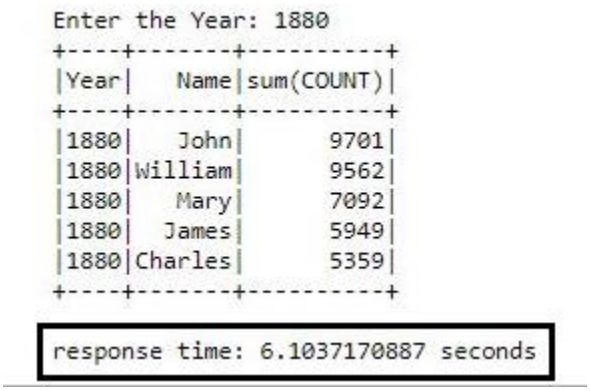
```
In [*]: time4 = time.time()
        Year = raw_input("Enter the Year: ")
        sdf.registerTempTable("abc")

        queryString = "select Year, Name, sum(COUNT) from abc where Year like %s group by Name, Year  order by sum(count) desc LIMIT 5" % Year

        abc = sqlContext.sql(queryString).show()
        print("response time: %s seconds" % (time.time() - time4))

        Enter the Year:
```

Response:



```
Enter the Year: 1880
+----+-------+-----------+
|Year|   Name|sum(COUNT)|
+----+-------+-----------+
|1880|   John|       9701|
|1880|William|       9562|
|1880|   Mary|       7092|
|1880|  James|       5949|
|1880|Charles|       5359|
+----+-------+-----------+

response time: 6.1037170887 seconds
```

Response Time: 6.10 seconds

Task 2.6: Input a child name and populate total number of birth registrations throughout the dataset for that name

Problem 3.4 - Input a child name and populate total number of birth registrations throughout the dataset for that name

```
In [*]: time5 = time.time()
        Name = raw_input("Enter your name: ")
        sdf.registerTempTable("abc")

        qwe = "SELECT Sum(Count) FROM abc WHERE Name like '%s'" % Name
        abc = sqlContext.sql(qwe).show()
        print("response time: %s seconds" % (time.time() - time5))
```

```
Enter your name:
```

Response:

```
Enter your name: Helen
+----------+
|sum(Count)|
+----------+
|   1019006|
+----------+

response time: 4.83650183678 seconds
```

Response Time: 4.83 seconds

Task 3:

Task 3.1: Loading the cleaned Data

Loading the Data Frame

In [1]:
```
import time
import sys
import types
import pandas as pd
from botocore.client import Config
import ibm_boto3

def __iter__(self): return 0

# @hidden_cell
# The following code accesses a file in your IBM Cloud Object Storage. It includes your credentials.
# You might want to remove those credentials before you share your notebook.
client_1f6f2e6280144b23bb3fd53cbdb85cdd = ibm_boto3.client(service_name='s3',
    ibm_api_key_id='H3RY-cRAHjH4W0Kbzwe1nQnJy4oM49dgBZOdz0R3AUwB',
    ibm_auth_endpoint="https://iam.ng.bluemix.net/oidc/token",
    config=Config(signature_version='oauth'),
    endpoint_url='https://s3-api.us-geo.objectstorage.service.networklayer.com')

body = client_1f6f2e6280144b23bb3fd53cbdb85cdd.get_object(Bucket='qwerty28b58f4c38184cafbd941800e4d7f669',Key='Collision_Statistics_Monthly_2000-2003.csv')['Body']
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__, body )

df_data_1 = pd.read_csv(body)
df_data_1.head()
```

Out[1]:

	ComID	GeogName	DataGroup1	DataGroup1_Order	DataGroup2	Year	Fatal	Injury	Property Damage	Total
0	cnt1201	Shelburne County	Total Reporting	1	Number	2001	3	60	148	211
1	cnt1201	Shelburne County	Total Reporting	1	Number	2002	2	47	122	171
2	cnt1201	Shelburne County	Total Reporting	1	Number	2003	1	69	141	211
3	cnt1201	Shelburne County	Total Reporting	1	Number	2000	0	49	135	184
4	cnt1202	Yarmouth County	Total Reporting	1	Number	2000	2	84	237	323

Task 3.2: Connecting the Pandas Data Frame with Sparks

Connecting Pandas DataFrame with SQL Spark

In [19]:
```
time1 = time.time()
sqlContext = SQLContext(sc)
sdf = sqlContext.createDataFrame(df_data_1)
sdf.printSchema()
print("Response Time: %s seconds" % (time.time() - time1))
```

```
root
 |-- ComID: string (nullable = true)
 |-- GeogName: string (nullable = true)
 |-- DataGroup1: string (nullable = true)
 |-- DataGroup1_Order: long (nullable = true)
 |-- DataGroup2: string (nullable = true)
 |-- Year: long (nullable = true)
 |-- Fatal: double (nullable = true)
 |-- Injury: double (nullable = true)
 |-- Property Damage: double (nullable = true)
 |-- Total: double (nullable = true)

Response Time: 0.237887859344 seconds
```

Response Time: 0.23 seconds

Task 3.3: Capture total incident counts (including Fatal, Injury and Property Damage) in a year (grouped by year).

Problem 4.1 - Capture total incident counts (including Fatal, Injury and Property Damage) in a year (grouped by year).

In [20]:
```
time2 = time.time()
sdf.registerTempTable("abc")

abc = sqlContext.sql("SELECT year, Sum(Total) FROM abc group by year order by year")
abc.show()
print("Response Time: %s seconds" % (time.time() - time2))
```

Response:

```
+----+------------------+
|year|        sum(Total)|
+----+------------------+
|2000| 60824.20000000001|
|2001|            58499.4|
|2002|            55977.2|
|2003|50492.899999999994|
+----+------------------+

Response Time: 1.15971589088 seconds
```

Response Time: 1.15 seconds.

Task 3.4: Capture sum of injuries in Nova Scotia grouped by year.

Problem 4.2 - Capture sum of injuries in Nova Scotia grouped by year.

In [21]:
```
time3 = time.time()
sdf.registerTempTable("abc")

abc = sqlContext.sql("SELECT year, Sum(Injury) FROM abc WHERE GeogName = 'Nova Scotia' group by year order by year")
abc.show()
print("Response Time: %s seconds" % (time.time() - time3))
```

Response:

```
+----+-----------------+
|year|      sum(Injury)|
+----+-----------------+
|2000|           9848.6|
|2001|8975.499999999998|
|2002|           8471.1|
|2003|7488.099999999999|
+----+-----------------+
```

```
Response Time: 0.436430931091 seconds
```

Response Time: 0.43 seconds.

Task 3.5: Capture sum of injury in Yarmouth Country in the year of 2000.

Problem 4.3 - Capture sum of injury in Yarmouth Country in the year of 2000.

In [22]:
```
time4 = time.time()
sdf.registerTempTable("abc")

abc = sqlContext.sql("SELECT year, Sum(Injury) FROM abc WHERE GeogName = 'Yarmouth County' and Year = 2000 group by year ")
abc.show()
print("Response Time: %s seconds" % (time.time() - time4))
```

Response:

```
+----+-----------+
|year|sum(Injury)|
+----+-----------+
|2000|      174.1|
+----+-----------+
```

```
Response Time: 0.544846057892 seconds
```

Response Time: 0.54 seconds.

Summary:

We have implemented first of three queries via local Spark implementation, in which we have cleaned the data first by removing selected characters and by performing several operations such as stemming and stopping. Further, we have passed this cleaned data for word count operation and it also gives the count for a number of lines and characters in the file. While solving the first query, we faced several errors on IBM Cloud platform hence we decided to switch to local implementation. We have executed this task in Ipython Notebook *[7]* as well as in python file.

For the second query, we have performed the whole operation on IBM Cloud Platform. Firstly, we have cleaned the data and created data frame using pandas then connected the data frame with Sparks internally. At last, we have provided various queries which perform desired tasks and gives the required content in iPython Notebook.

For the final query, we have utilized IBM Cloud platform. In which we have cleaned the data first. On the later part, we have created data frame using pandas and provided several queries to fulfill anticipated tasks in iPython Notebook.

Experience with Tools:

So far, we have come through this, our experience is great with Python and Apache Sparks. These are great tools when it comes to dealing with Big Data operations. We have observed that Python is very powerful language. What makes it this powerful is its packages *[8]* such as Pandas, Numpy, Matplotlib, Scifit-learn, PySpark and so on. We have faced major downtime in IBM cloud facility as it was very slow in terms of response. In addition, we faced several errors related to Apache Sparks while implementing the queries locally. Overall, it was a wonderful experience as we were dealing with Apache Sparks first time and we learned a lot by solving this assignment.

References:

[1] Wikipedia," SQL", [Online]. Available: https://en.wikipedia.org/wiki/SQL [Accessed 18th February 2018]

[2] [Online] Available: https://www.knowledgehut.com/blog/information-technology/5-best-data-processing-frameworks [Accessed 19th February 2018]

[3] "Wikipedia", [Online]. Available: https://en.wikipedia.org/wiki/Big_data [Accessed 19th February 2018]

[4] Apache, "Apache Sparks" [Online]. Available: https://spark.apache.org/ [Accessed 16th February 2018]

[5] “PySpark”, [Online]. Available: https://pypi.python.org/pypi/pyspark/2.2.1 [Accessed 20th February 2018]

[6] IBM Inc. “IBM Data Science Experience”, [Online]. Available: https://datascience.ibm.com/ [Accessed 16th February 2018]

[7] “The Jupyter Notebook”, [Online]. Available: https://ipython.org/notebook.html [Accessed 16th February 2018]

[8] “PyPI - the Python Package Index”, [Online]. Available: https://pypi.python.org/pypi [Accessed 16th February 2018]