Name: Prerna Sunil Jadhav
Sap Id: 60004220127
Batch: C2-2
Course: Big Data Infrastructure laboratory
Course Code: DJ19CEEL6011

EXPERIMENT 10

AIM: Perform sentiment analysis using Kafka.

THEORY: Data reasoning is the process of transmitting a continuous data (also known a streams) typically fed into stream processing software to derive valuable insights.
Apache Kafka is an open source, distributed streaming platform that enables the development of real time, event driven application.

Components of Apache Kafka

- Producers
- Brokers
- Topics & Partition
- Replicas
- consumers
- followers & leaders

- **Producers:** Producers in Kafka publish messages to one or more topics.

- **Brokers:** A Kafka cluster comprises one or more servers that are known as brokers. Broker works as a container that can hold multiple topics

- Topic: A stream of messages that are a part of specific category or feed name is referred to as a kafka topic.

- Partitions: Topics in kafka are divided into a configurable no. of parts, which are known as partitions.

- Replicas: Replicas are like backups for partition in kafka.

- Leaders & followers: Easy Every partition will have one server that plays the role of a leader for that partition. Leader will perform read and write operations. Followers will replicate the data of the leader.

CONCLUSION: Thus, we performed sentiment analysis using kafka.

Name:	Prerna Sunil Jadhav
Sap Id:	60004220127
Class:	T. Y. B. Tech (Computer Engineering)
Course:	Big Data Infrastructure Laboratory
Course Code:	DJ19CEEL6011
Experiment No.:	10

AIM: Perform Sentiment Analysis using Kafka.

Step 1 : As the Log Data is unstructured, we parse and create a structure from each line, which will in turn become each row while analysis.

```
import re
from pyspark.sql import Row
# This is the regex which is specific to Apache Access Logs parsing, which can be modified according to different Log formats as per the need
# Example Apache log line:
#   127.0.0.1 - - [21/Jul/2014:9:55:27 -0800] "GET /home.html HTTP/1.1" 200 2048
#   1:IP  2:client 3:user 4:date time           5:method 6:req 7:proto   8:respcode 9:size
APACHE_ACCESS_LOG_PATTERN = '^(\S+) (\S+) (\S+) \[([\w:/]+\s[+\-]\d{4})\] "(\S+) (\S+) (\S+)" (\d{3}) (\d+)'

# The below function is modelled specific to Apache Access Logs Model, which can be modified as per needs to different Logs format
# Returns a dictionary containing the parts of the Apache Access Log.
def parse_apache_log_line(logline):
    match = re.search(APACHE_ACCESS_LOG_PATTERN, logline)
    if match is None:
        raise Error("Invalid logline: %s" % logline)
    return Row(
        ip_address    = match.group(1),
        client_identd = match.group(2),
        user_id       = match.group(3),
        date = (match.group(4)[:-6]).split(":", 1)[0],
        time = (match.group(4)[:-6]).split(":", 1)[1],
        method        = match.group(5),
        endpoint      = match.group(6),
        protocol      = match.group(7),
        response_code = int(match.group(8)),
        content_size  = int(match.group(9))
    )
```

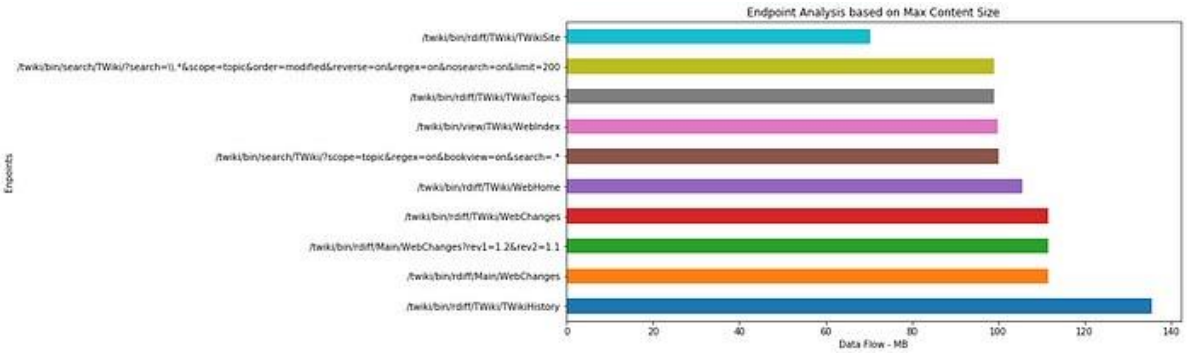
Step 2: Create Spark Context, SQL Context, DataFrame (is a distributed collection of data organized into named columns. It is conceptually equivalent to a table in a relational database)

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

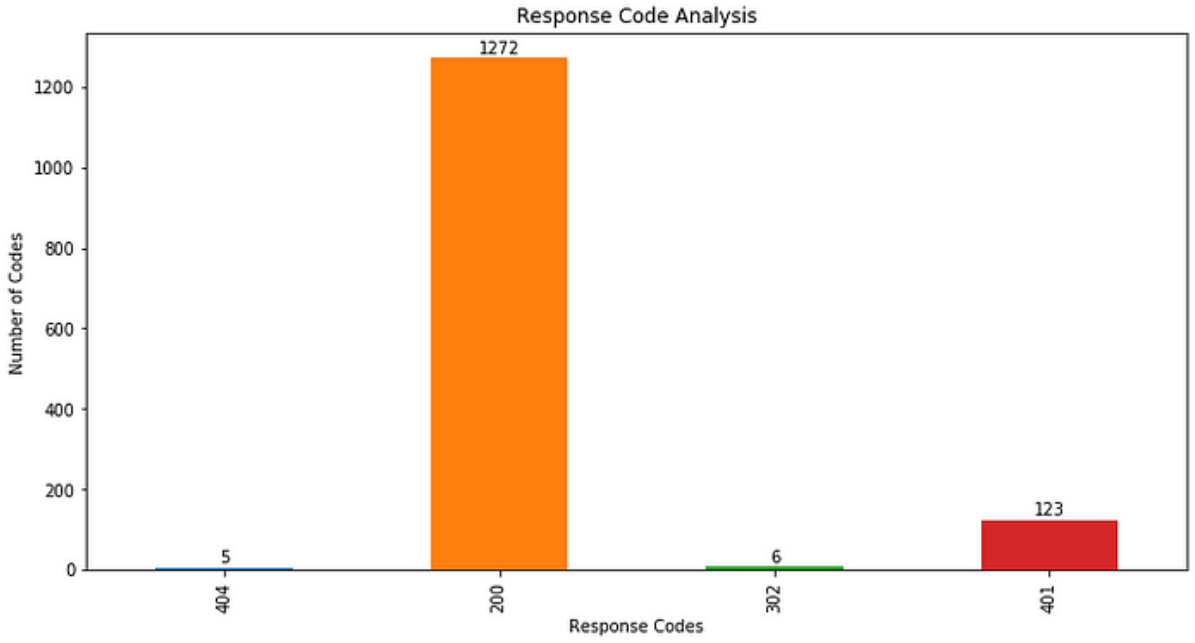
```
from pyspark import SparkContext, SparkConf
from pyspark.sql import SQLContext
import apache_access_log # This is the first file name , in which we created Data Structure of Log
import sys

# Set up The Spark App
conf = SparkConf().setAppName("Log Analyzer")
# Create Spark Context
sc = SparkContext(conf=conf)
#Create SQL Context
sqlContext = SQLContext(sc)

#Input File Path
logFile = 'Give Your Input File Path Here'

# .cache() - Persists the RDD in memory, which will be re-used again
access_logs = (sc.textFile(logFile)
               .map(apache_access_log.parse_apache_log_line)
               .cache())

schema_access_logs = sqlContext.createDataFrame(access_logs)
#Creates a table on which SQL like queries can be fired for analysis
schema_access_logs.registerTempTable("logs")
```

Step 3 : Analyze Top 10 Endpoints which Transfer Maximum Content in MB

```
#Top 10 Endpoints which Transfer Maximum Content
#.rdd.map() - Will convert the resulted rows from SQL query into a map
# .collect() - actually executes the DAG to get the overall results
topEndpointsMaxSize = (sqlContext
               .sql("SELECT endpoint,content_size/1024 FROM logs ORDER BY content_size DESC LIMIT 10")
               .rdd.map(lambda row: (row[0], row[1]))
               .collect())
# Plot Analysis Code
bar_plot_list_of_tuples_horizontal(topEndpointsMaxSize,'Data Flow - MB','Enpoints','Endpoint Analysis
    based on Max Content Size')
```

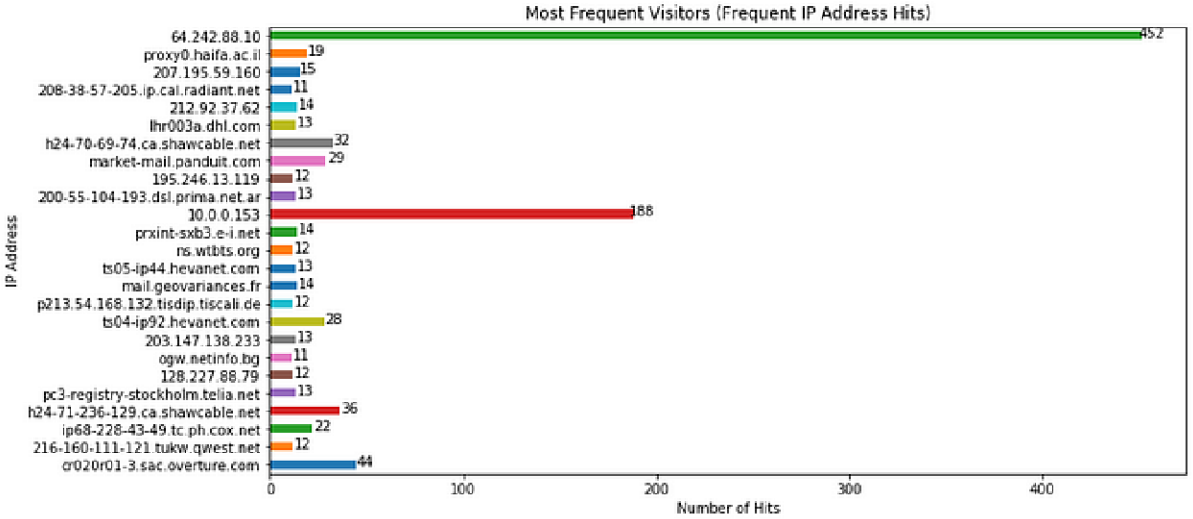


```
# Response Code Analysis
responseCodeToCount = (sqlContext
                       .sql("SELECT response_code, COUNT(*) AS theCount FROM logs GROUP BY
                           response_code")
                       .rdd.map(lambda row: (row[0], row[1]))
                       .collect())
bar_plot_list_of_tuples(responseCodeToCount,'Response Codes','Number of Codes','Response Code Analysis'
    )

# Code to Plot the results
def bar_plot_list_of_tuples(input_list,x_label,y_label,plot_title):
    x_labels = [val[0] for val in input_list]
    y_labels = [val[1] for val in input_list]
    plt.figure(figsize=(12, 6))
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.title(plot_title)
    ax = pd.Series(y_labels).plot(kind='bar')
    ax.set_xticklabels(x_labels)
    rects = ax.patches
    for rect, label in zip(rects, y_labels):
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width()/2, height + 5, label, ha='center', va='bottom')
```

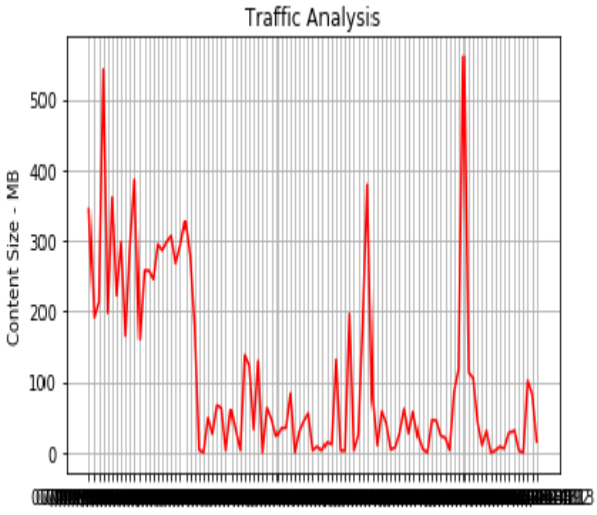


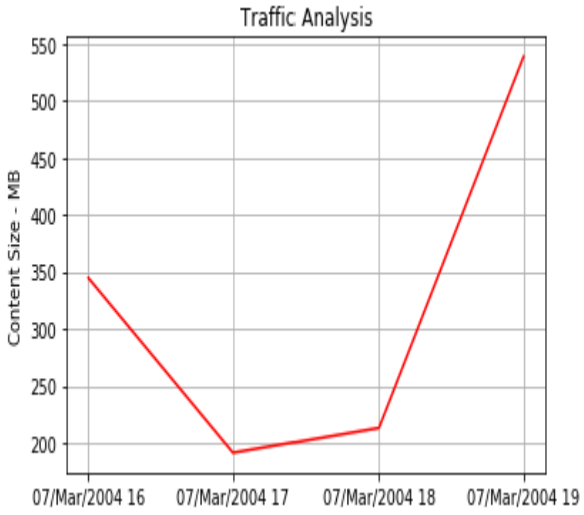

```
# Most Frequent Visitors (Most Frequent IP Address visits).
frequentIpAddressesHits = (sqlContext
                .sql("SELECT ip_address, COUNT(*) AS total FROM logs GROUP BY ip_address HAVING total >
                    10")
                .rdd.map(lambda row: (row[0], row[1]))
                .collect())
bar_plot_list_of_tuples_horizontal(frequentIpAddressesHits,'Number of Hits','IP Address','Most Frequent
    Visitors (Frequent IP Address Hits)')
```

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

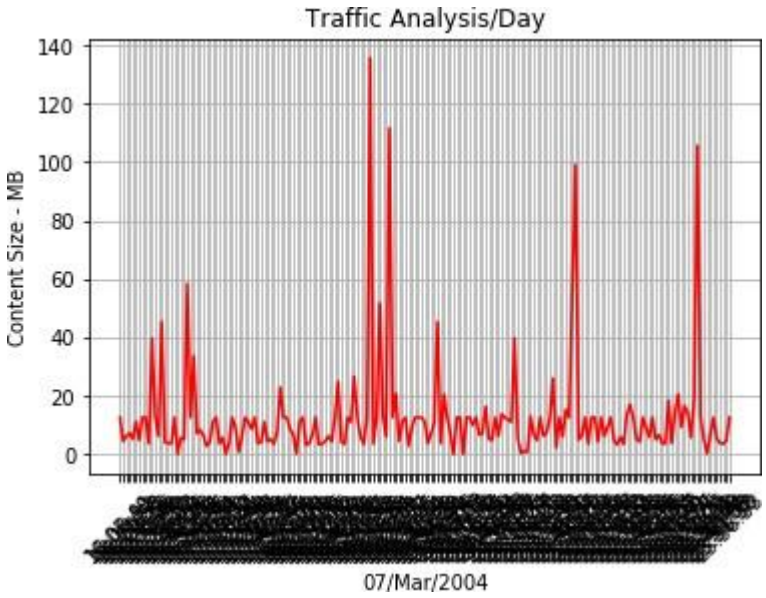



```
1  # Traffic Analysis for past One Week
2  trafficWithTime = (sqlContext
3                     .sql("SELECT date, content_size/1024 FROM logs")
4                     .rdd.map(lambda row: (row[0], row[1]))
5                     .collect())
6  time_series_plot(trafficWithTime)
7
```

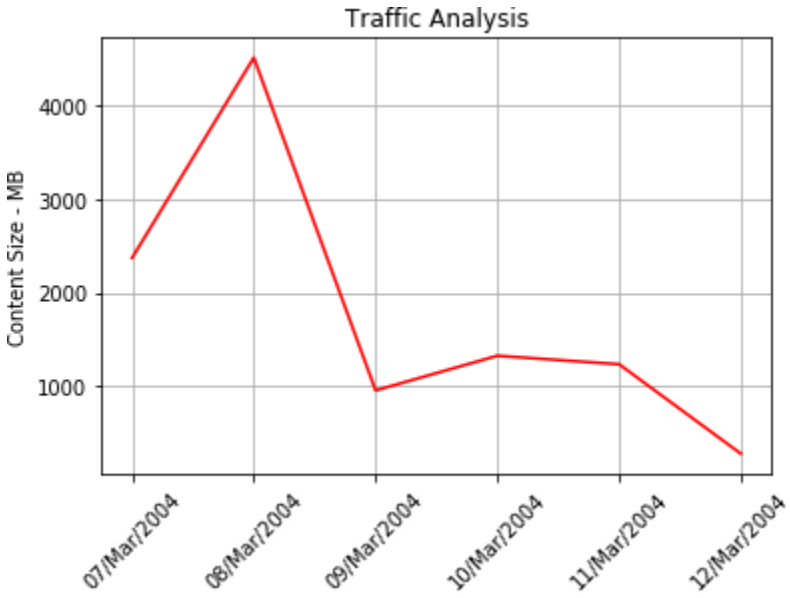




```
# Overall Traffic Analysis for a Day
Day = '07/Mar/2004'
trafficperDay = (sqlContext
                 .sql("SELECT time,content_size/1024 FROM logs where date='07/Mar/2004'")
                 .rdd.map(lambda row: (row[0], row[1]))
                 .collect())
time_series_plot(trafficperDay,Day,'Content Size - MB','Traffic Analysis/Day')
```



Outliers can be clearly detected by analysis the spikes and which end points were been hit at time by what IP Addresses.



Here, we can see an unusual spike on 8th March, which can be analyzed further for identifying discrepancy.

Code for Plot Analysis:

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

```
def time_series_plot(input_list,x_label,y_lablel,plot_title):
    x_labels = [val[0] for val in input_list]
    y_labels = [val[1] for val in input_list]
    dict_plot = OrderedDict()
    for x,y in zip(x_labels,y_labels):
       # cur_val = x.split(":", 1)[0]
        cur_val = x.split(" ")[0]
        #print(cur_val)
        dict_plot[cur_val] = dict_plot.get(cur_val, 0) + y
    input_list = list(dict_plot.items())
    x_labels = [val[0] for val in input_list]
    y_labels = [val[1] for val in input_list]
    plt.plot_date(x=x_labels, y=y_labels, fmt="r-")
    plt.xticks(rotation=45)
    plt.title(plot_title)
    plt.xlabel(x_label)
    plt.ylabel(y_lablel)
    plt.grid(True)
    plt.show()
```

```
20
21      def bar_plot_list_of_tuples_horizontal(input_list,x_label,y_label,plot_title):
22      y_labels = [val[0] for val in input_list]
23      x_labels = [val[1] for val in input_list]
24      plt.figure(figsize=(12, 6))
25      plt.xlabel(x_label)
26      plt.ylabel(y_label)
27      plt.title(plot_title)
28      ax = pd.Series(x_labels).plot(kind='barh')
29      ax.set_yticklabels(y_labels)
30      for i, v in enumerate(x_labels):
31          ax.text(int(v) + 0.5, i - 0.25, str(v),ha='center', va='bottom')
```

```
32
33      # Frequent End Points
34  topEndpoints = (sqlContext
35                  .sql("SELECT endpoint, COUNT(*) AS total FROM logs GROUP BY endpoint ORDER BY total
                        DESC LIMIT 10")
36                  .rdd.map(lambda row: (row[0], row[1]))
37                  .collect())
38  bar_plot_list_of_tuples_horizontal(topEndpoints,'Number of Times Accessed','End Points','Most
        Frequent Endpoints')
```