Experiment 10

Aim : Perform Sentiment Analysis Using Kafka

BDT Experiment 10 DATE:

Kreena Shah
60004210243
C'32

Aim : Perform Sentiment Analysis using Kafka

Theory :

Apache Kafka is an open source, distributed streaming platform that enables the development of real time, event-driven applications

A streaming platform enables developers to build applications that continuously consume & process these streams at extremely high speeds with a high level of fidelity & accuracy based on the correct order of their occurence.

Components of Apache Kafka

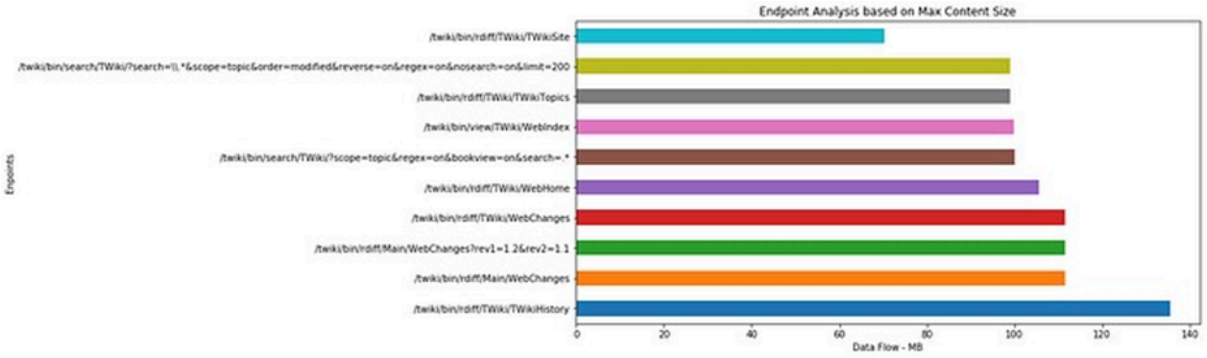
(1) Topics
(2) Brokers
(3) Consumers & Consumer Groups
(4) Producers
(5) Partitions
(6) Partition Offset
(7) Replicas
(8) Leaders & Followers

Sentiment Analysis is the process of analyzing digital text to determine if the emotional tone of message is +ve, -ve, neutral.

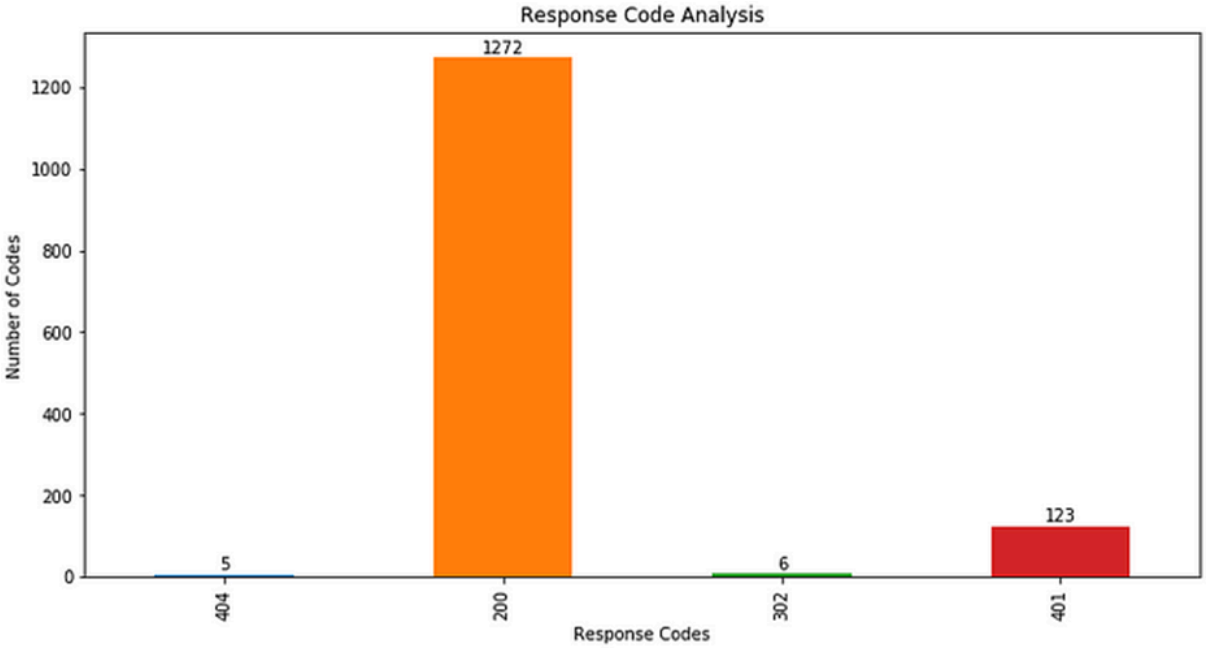
```
import re
from pyspark.sql import Row
# This is the regex which is specific to Apache Access Logs parsing, which can be modified according to
    different Log formats as per the need
# Example Apache log line:
#   127.0.0.1 - - [21/Jul/2014:9:55:27 -0800] "GET /home.html HTTP/1.1" 200 2048
#   1:IP  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))
    )
```

```
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")
```

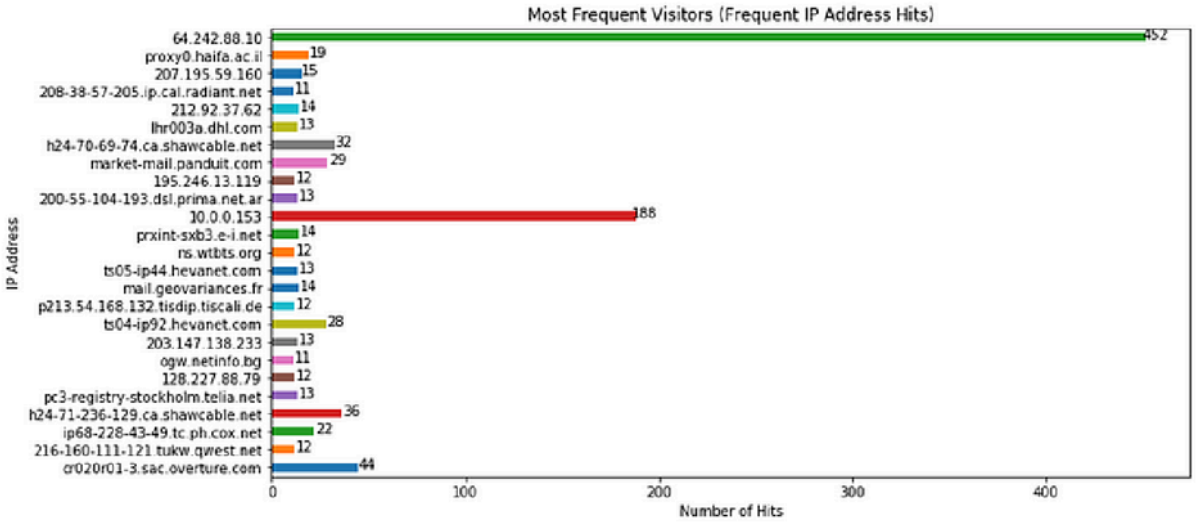
```
#Top 10 Endpoints which Transfer Maximum Content
#.rdd.map() - Will convert the resulted rows from SQL query into a map
# .collect() - actually executes the DAG to get the overall results
topEndpointsMaxSize = (sqlContext
               .sql("SELECT endpoint,content_size/1024 FROM logs ORDER BY content_size 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')
```



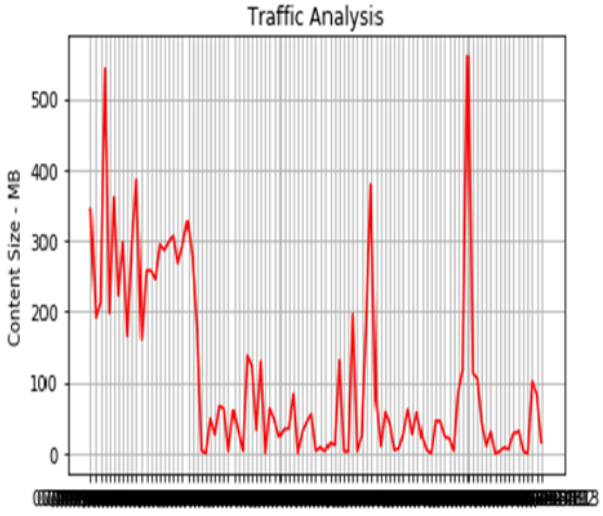

```
# Response Code Analysis
responseCodeToCount = (sqlContext
                       .sql("SELECT response_code, COUNT(*) AS theCount FROM logs GROUP BY
                           response_code")
                       .rdd.map(lambda row: (row[0], row[1]))
                       .collect())
bar_plot_list_of_tuples(responseCodeToCount,'Response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

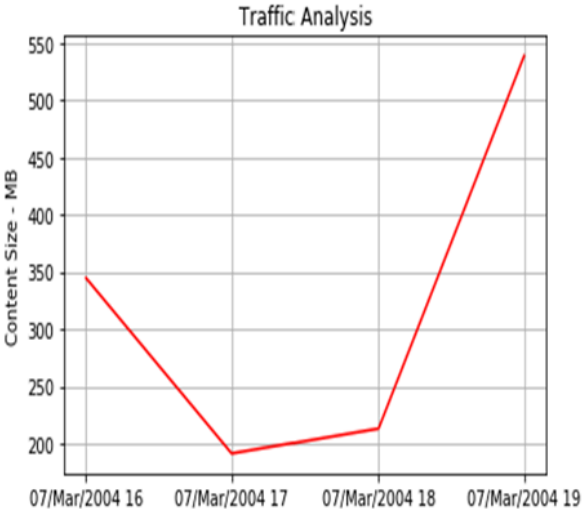


```
# Most Frequent Visitors (Most Frequent IP Address visits).
frequentIpAddressesHits = (sqlContext
                .sql("SELECT ip_address, COUNT(*) AS total FROM logs GROUP BY ip_address HAVING total >
                    10")
                .rdd.map(lambda row: (row[0], row[1]))
              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

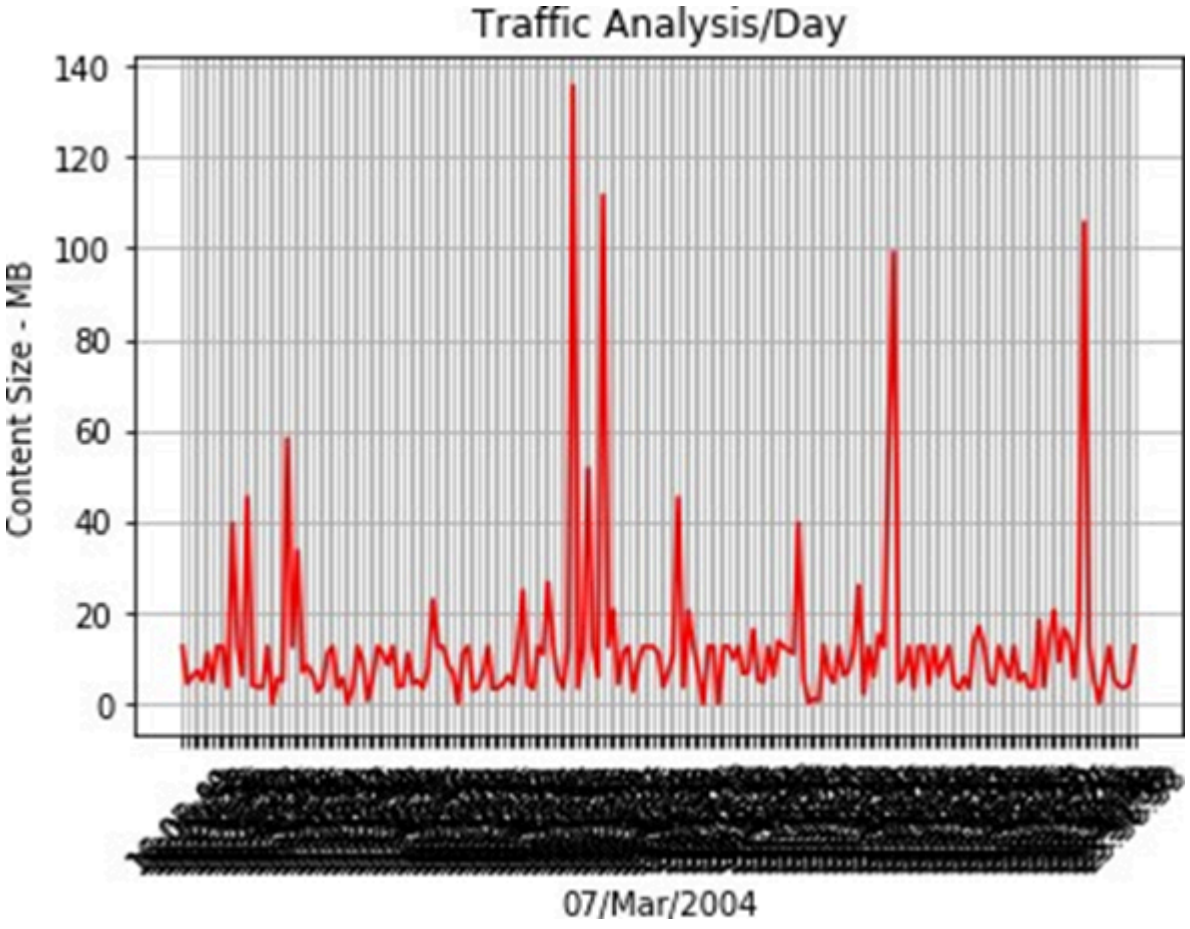


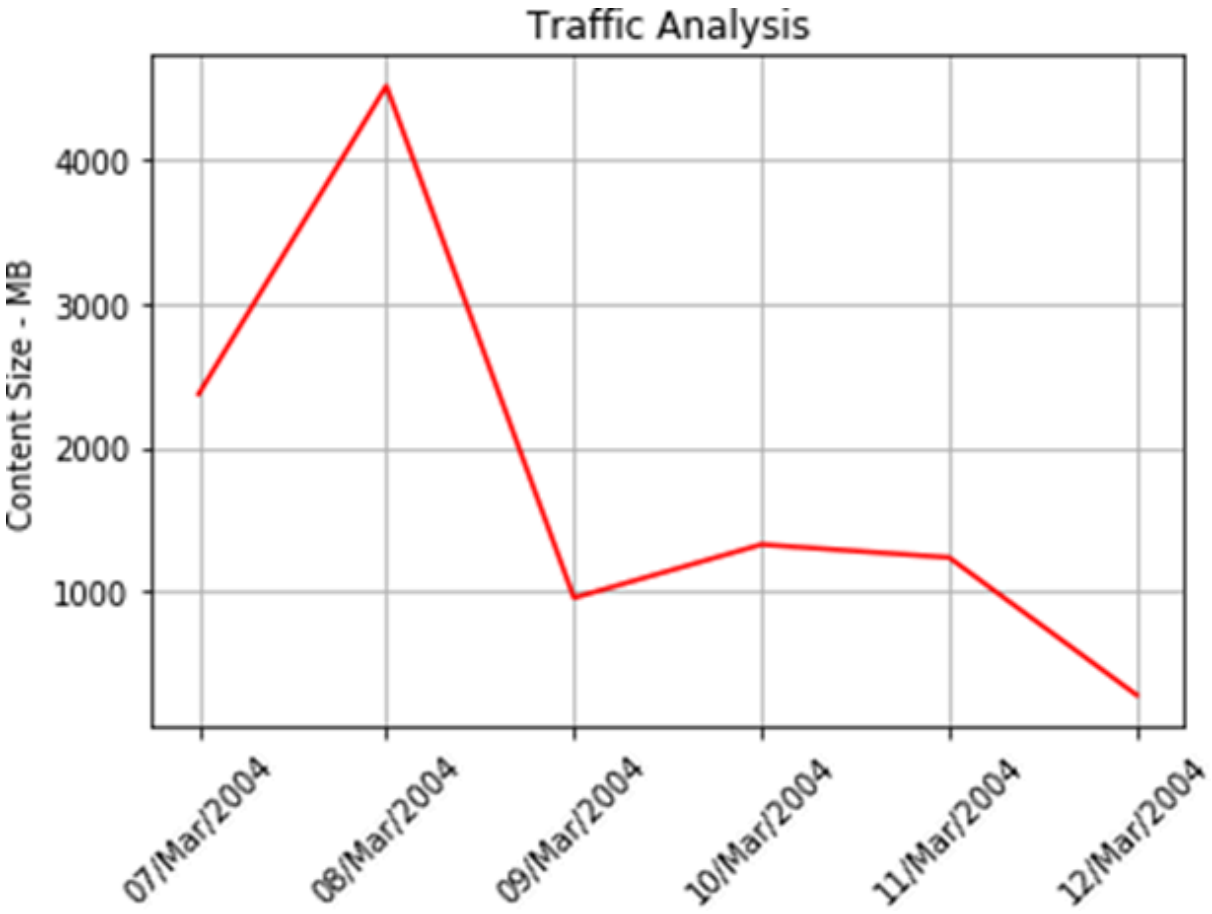

```
# Traffic Analysis for past One Week
trafficWithTime = (sqlContext
                        .sql("SELECT date, content_size/1024 FROM logs")
                        .rdd.map(lambda row: (row[0], row[1]))
                        .collect())
time_series_plot(trafficWithTime)
```





```
# 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')
```

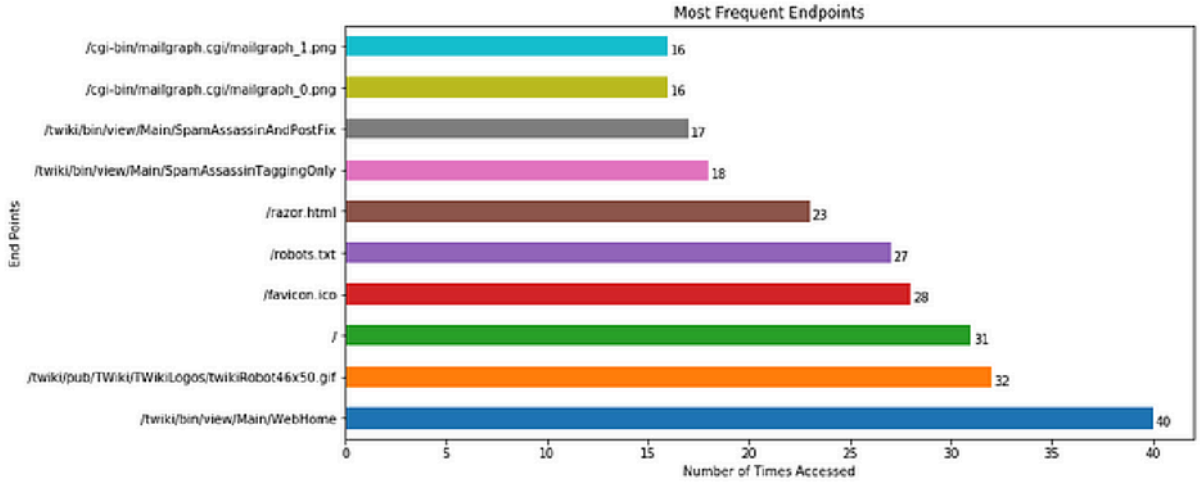





```
def time_series_plot(input_list,x_label,y_lablel,plot_title):
    x_labels = [val[0] for val in input_list]
    y_labels = [val[1] for val in input_list]
    dict_plot = OrderedDict()
    for x,y in zip(x_labels,y_labels):
       # cur_val = x.split(":", 1)[0]
        cur_val = x.split(" ")[0]
        #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()
```

```
def bar_plot_list_of_tuples_horizontal(input_list,x_label,y_label,plot_title):
    y_labels = [val[0] for val in input_list]
    x_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(x_labels).plot(kind='barh')
    ax.set_yticklabels(y_labels)
    for i, v in enumerate(x_labels):
        ax.text(int(v) + 0.5, i - 0.25, str(v),ha='center', va='bottom')
```

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



Conclusion : Thus, we performed sentiment analysis using Kafka