*“There’s something so paradoxical about pi. On the one hand, it represents order, as embodied by the shape of a circle, long held to be a symbol of perfection and eternity. On the other hand, pi is unruly, disheveled in appearance, its digits obeying no obvious rule, or at least none that we can perceive. Pi is elusive and mysterious, forever beyond reach. Its mix of order and disorder is what makes it so bewitching. ”*

From  Infinite Powers by Steven Strogatz

Anybody who wants to be “anybody” in Big Data must necessarily be able to work on both large structured and unstructured data.  Log analysis is critical in any enterprise which is usually unstructured. As I mentioned in my previous post [Big Data: On RDDs, Dataframes,Hive QL with Pyspark and SparkR-Part 3](https://gigadom.in/2019/04/12/big-data-on-rdds-dataframeshive-ql-with-pyspark-and-sparkr-part-3/) RDDs are typically used to handle unstructured data. Spark has the Dataframe abstraction over RDDs which performs better as it is optimized with the Catalyst optimization engine. Nevertheless, it is important to be able to process with RDDs.  This post is a continuation of my 3 earlier posts on Big Data namely

1. [Big Data-1: Move into the big league:Graduate from Python to Pyspark](https://gigadom.in/2018/10/08/big-data-1-move-into-the-big-leaguegraduate-from-python-to-pyspark/)  
2. [Big Data-2: Move into the big league:Graduate from R to SparkR](https://gigadom.in/2018/10/09/big-data-2-move-into-the-big-leaguegraduate-from-r-to-sparkr/)  
3. [Big Data: On RDDs, Dataframes,Hive QL with Pyspark and SparkR-Part 3](https://gigadom.in/2019/04/12/big-data-on-rdds-dataframeshive-ql-with-pyspark-and-sparkr-part-3/)

This post uses publicly available Webserver logs from NASA. The logs are for the months Jul 95 and Aug 95 and are a good place to start unstructured text analysis/log analysis. I highly recommend parsing these publicly available logs with regular expressions. It is only when you do that the truth of Jamie Zawinski’s pearl of wisdom

*“Some people, when confronted with a problem, think “I know, I’ll use regular expressions.” Now they have two problems.” –* Jamie Zawinksi

hits home. I spent many hours struggling with regex!!

For this post for the RDD part,  I had to refer to Dr. Fisseha Berhane’s blog post [Webserver Log Analysis](https://datascience-enthusiast.com/Python/DataFramesVsRDDsVsSQLSpark-Part3.html) and for the Pyspark part, to the Univ. of California Specialization which I had done 3 years back [Big Data Analysis with Apache Spark](https://courses.edx.org/courses/course-v1:BerkeleyX+CS110x+2T2016/course/). Once I had played around with the regex for RDDs and PySpark I managed to get SparkR and SparklyR versions to work.

The notebooks used in this post have been published and are available at

1. [logsAnalysiswithRDDs](https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/3582687578800188/2740262638283071/4574970912889634/latest.html)
2. [logsAnalysiswithPyspark](https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/3582687578800188/2740262638283126/4574970912889634/latest.html)
3. [logsAnalysiswithSparkRandSparklyR](https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/3582687578800188/2740262638283154/4574970912889634/latest.html)

You can also download all the notebooks from Github at [WebServerLogsAnalysis](https://github.com/tvganesh/WebServerLogsAnalysis)

An essential and unavoidable aspect of Big Data processing is the need to process unstructured text.Web server logs are one such area which requires Big Data techniques to process massive amounts of logs. The Common Log Format also known as the NCSA Common log format, is a standardized text file format used by web servers when generating server log files. Because the format is standardized, the files can be readily analyzed.

A publicly available webserver logs is the NASA-HTTP Web server logs. This is good dataset with which we can play around to get familiar to handling web server logs. The logs can be accessed at [NASA-HTTP](ftp://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html)

**Description** These two traces contain two month’s worth of all HTTP requests to the NASA Kennedy Space Center WWW server in Florida.

**Format** The logs are an ASCII file with one line per request, with the following columns:

-host making the request. A hostname when possible, otherwise the Internet address if the name could not be looked up.

-timestamp in the format “DAY MON DD HH:MM:SS YYYY”, where DAY is the day of the week, MON is the name of the month, DD is the day of the month, HH:MM:SS is the time of day using a 24-hour clock, and YYYY is the year. The timezone is -0400.

-request given in quotes.

-HTTP reply code.

-bytes in the reply.

**1 Parse Web server logs with RDDs**

**1.1 Read NASA Web server logs**

Read the logs files from NASA for the months Jul 95 and Aug 95

from pyspark import SparkContext, SparkConf

from pyspark.sql import SQLContext

conf = SparkConf().setAppName("Spark-Logs-Handling").setMaster("local[\*]")

sc = SparkContext.getOrCreate(conf)

sqlcontext = SQLContext(sc)

rdd = sc.textFile("/FileStore/tables/NASA\_access\_log\_\*.gz")

rdd.count()

Out[1]: 3461613

**1.2Check content**

Check the logs to identify the parsing rules required for the logs

i=0

for line in rdd.sample(withReplacement = False, fraction = 0.00001, seed = 100).collect():

i=i+1

print(line)

if i >5:

break

ix-stp-fl2-19.ix.netcom.com – – [03/Aug/1995:23:03:09 -0400] “GET /images/faq.gif HTTP/1.0” 200 263  
slip183-1.kw.jp.ibm.net – – [04/Aug/1995:18:42:17 -0400] “GET /shuttle/missions/sts-70/images/DSC-95EC-0001.gif HTTP/1.0” 200 107133  
piweba4y.prodigy.com – – [05/Aug/1995:19:17:41 -0400] “GET /icons/menu.xbm HTTP/1.0” 200 527  
ruperts.bt-sys.bt.co.uk – – [07/Aug/1995:04:44:10 -0400] “GET /shuttle/countdown/video/livevideo2.gif HTTP/1.0” 200 69067  
dal06-04.ppp.iadfw.net – – [07/Aug/1995:21:10:19 -0400] “GET /images/NASA-logosmall.gif HTTP/1.0” 200 786  
p15.ppp-1.directnet.com – – [10/Aug/1995:01:22:54 -0400] “GET /images/KSC-logosmall.gif HTTP/1.0” 200 1204

**1.3 Write the parsing rule for each of the fields**

* host
* timestamp
* path
* status
* content\_bytes

**1.21 Get IP address/host name**

This regex is at the start of the log and includes any non-white characted

import re

rslt=(rdd.map(lambda line: re.search('\S+',line)

.group(0))

.take(3)) # Get the IP address \host name

rslt

Out[3]: [‘in24.inetnebr.com’, ‘uplherc.upl.com’, ‘uplherc.upl.com’]

**1.22 Get timestamp**

Get the time stamp

rslt=(rdd.map(lambda line: re.search(‘(\S+ -\d{4})’,line)

.groups())

.take(3)) #Get the date

rslt

[(‘[01/Aug/1995:00:00:01 -0400’,),  
(‘[01/Aug/1995:00:00:07 -0400’,),  
(‘[01/Aug/1995:00:00:08 -0400’,)]

**1.23 HTTP request**

Get the HTTP request sent to Web server \w+ {GET}

# Get the REST call with ” “

rslt=(rdd.map(lambda line: re.search('"\w+\s+([^\s]+)\s+HTTP.\*"',line)

.groups())

.take(3)) # Get the REST call

rslt

[(‘/shuttle/missions/sts-68/news/sts-68-mcc-05.txt’,),  
(‘/’,),  
(‘/images/ksclogo-medium.gif’,)]

**1.23Get HTTP response status**

Get the HTTP response to the request

rslt=(rdd.map(lambda line: re.search('"\s(\d{3})',line)

.groups())

.take(3)) #Get the status

rslt

Out[6]: [(‘200’,), (‘304’,), (‘304’,)]

**1.24 Get content size**

Get the HTTP response in bytes

rslt=(rdd.map(lambda line: re.search(‘^.\*\s(\d\*)$’,line)

.groups())

.take(3)) # Get the content size

rslt

Out[7]: [(‘1839’,), (‘0’,), (‘0’,)]

**1.24 Putting it all together**

Now put all the individual pieces together into 1 big regular expression and assign to the groups

1. Host 2. Timestamp 3. Path 4. Status 5. Content\_size

rslt=(rdd.map(lambda line: re.search('^(\S+)((\s)(-))+\s(\[\S+ -\d{4}\])\s("\w+\s+([^\s]+)\s+HTTP.\*")\s(\d{3}\s(\d\*)$)',line)

.groups())

.take(3))

rslt

[(‘in24.inetnebr.com’,  
‘ -‘,  
‘ ‘,  
‘-‘,  
‘[01/Aug/1995:00:00:01 -0400]’,  
‘”GET /shuttle/missions/sts-68/news/sts-68-mcc-05.txt HTTP/1.0″‘,  
‘/shuttle/missions/sts-68/news/sts-68-mcc-05.txt’,  
‘200 1839’,  
‘1839’),  
(‘uplherc.upl.com’,  
‘ -‘,  
‘ ‘,  
‘-‘,  
‘[01/Aug/1995:00:00:07 -0400]’,  
‘”GET / HTTP/1.0″‘,  
‘/’,  
‘304 0’,  
‘0’),  
(‘uplherc.upl.com’,  
‘ -‘,  
‘ ‘,  
‘-‘,  
‘[01/Aug/1995:00:00:08 -0400]’,  
‘”GET /images/ksclogo-medium.gif HTTP/1.0″‘,  
‘/images/ksclogo-medium.gif’,  
‘304 0’,  
‘0’)]

**1.25 Add a log parsing function**

import re

def parse\_log1(line):

match = re.search('^(\S+)((\s)(-))+\s(\[\S+ -\d{4}\])\s("\w+\s+([^\s]+)\s+HTTP.\*")\s(\d{3}\s(\d\*)$)',line)

if match is None:

return(line,0)

else:

return(line,1)

**1.26 Check for parsing failure**

Check how many lines successfully parsed with the parsing function

n\_logs = rdd.count()

failed = rdd.map(lambda line: parse\_log1(line)).filter(lambda line: line[1] == 0).count()

print('Out of a total of {} logs, {} failed to parse'.format(n\_logs,failed))

# Get the failed records line[1] == 0

failed1=rdd.map(lambda line: parse\_log1(line)).filter(lambda line: line[1]==0)

failed1.take(3)

Out of a total of 3461613 logs, 38768 failed to parse  
Out[10]:  
[(‘gw1.att.com – – [01/Aug/1995:00:03:53 -0400] “GET /shuttle/missions/sts-73/news HTTP/1.0” 302 -‘,  
0),  
(‘js002.cc.utsunomiya-u.ac.jp – – [01/Aug/1995:00:07:33 -0400] “GET /shuttle/resources/orbiters/discovery.gif HTTP/1.0” 404 -‘,  
0),  
(‘pipe1.nyc.pipeline.com – – [01/Aug/1995:00:12:37 -0400] “GET /history/apollo/apollo-13/apollo-13-patch-small.gif” 200 12859’,  
0)]

**1.26 The above rule is not enough to parse the logs**

It can be seen that the single rule only parses part of the logs and we cannot group the regex separately. There is an error “AttributeError: ‘NoneType’ object has no attribute ‘group’” which shows up

#rdd.map(lambda line: re.search(‘^(\S+)((\s)(-))+\s(\[\S+ -\d{4}\])\s(“\w+\s+([^\s]+)\s+HTTP.\*”)\s(\d{3}\s(\d\*)$)’,line[0]).group()).take(4)

File “/databricks/spark/python/pyspark/util.py”, line 99, in wrapper  
return f(\*args, \*\*kwargs)  
File “”, line 1, in   
AttributeError: ‘NoneType’ object has no attribute ‘group’

at org.apache.spark.api.python.BasePythonRunner$ReaderIterator.handlePythonException(PythonRunner.scala:490)

**1.27 Add rule for parsing failed records**

One of the issues with the earlier rule is the content\_size has “-” for some logs

import re

def parse\_failed(line):

match = re.search('^(\S+)((\s)(-))+\s(\[\S+ -\d{4}\])\s("\w+\s+([^\s]+)\s+HTTP.\*")\s(\d{3}\s-$)',line)

if match is None:

return(line,0)

else:

return(line,1)

**1.28 Parse records which fail**

Parse the records that fails with the new rule

failed2=rdd.map(lambda line: parse\_failed(line)).filter(lambda line: line[1]==1)

failed2.take(5)

Out[13]:  
[(‘gw1.att.com – – [01/Aug/1995:00:03:53 -0400] “GET /shuttle/missions/sts-73/news HTTP/1.0” 302 -‘,  
1),  
(‘js002.cc.utsunomiya-u.ac.jp – – [01/Aug/1995:00:07:33 -0400] “GET /shuttle/resources/orbiters/discovery.gif HTTP/1.0” 404 -‘,  
1),  
(‘tia1.eskimo.com – – [01/Aug/1995:00:28:41 -0400] “GET /pub/winvn/release.txt HTTP/1.0” 404 -‘,  
1),  
(‘itws.info.eng.niigata-u.ac.jp – – [01/Aug/1995:00:38:01 -0400] “GET /ksc.html/facts/about\_ksc.html HTTP/1.0” 403 -‘,  
1),  
(‘grimnet23.idirect.com – – [01/Aug/1995:00:50:12 -0400] “GET /www/software/winvn/winvn.html HTTP/1.0” 404 -‘,  
1)]

**1.28 Add both rules**

Add both rules for parsing the log.

**Note it can be shown that even with both rules all the logs are not parse.Further rules may need to be added**

import re

def parse\_log2(line):

# Parse logs with the rule below

match = re.search('^(\S+)((\s)(-))+\s(\[\S+ -\d{4}\])\s("\w+\s+([^\s]+)\s+HTTP.\*")\s(\d{3})\s(\d\*)$',line)

# If match failed then use the rule below

if match is None:

match = re.search('^(\S+)((\s)(-))+\s(\[\S+ -\d{4}\])\s("\w+\s+([^\s]+)\s+HTTP.\*")\s(\d{3}\s-$)',line)

if match is None:

return (line, 0) # Return 0 for failure

else:

return (line, 1) # Return 1 for success

**1.29 Group the different regex to groups for handling**

def map2groups(line):

match = re.search('^(\S+)((\s)(-))+\s(\[\S+ -\d{4}\])\s("\w+\s+([^\s]+)\s+HTTP.\*")\s(\d{3})\s(\d\*)$',line)

if match is None:

match = re.search('^(\S+)((\s)(-))+\s(\[\S+ -\d{4}\])\s("\w+\s+([^\s]+)\s+HTTP.\*")\s(\d{3})\s(-)$',line)

return(match.groups())

**1.30 Parse the logs and map the groups**

parsed\_rdd = rdd.map(lambda line: parse\_log2(line)).filter(lambda line: line[1] == 1).map(lambda line : line[0])

parsed\_rdd2 = parsed\_rdd.map(lambda line: map2groups(line))

**2. Parse Web server logs with Pyspark**

**2.1Read data into a Pyspark dataframe**

import os

logs\_file\_path="/FileStore/tables/" + os.path.join('NASA\_access\_log\_\*.gz')

from pyspark.sql.functions import split, regexp\_extract

base\_df = sqlContext.read.text(logs\_file\_path)

#base\_df.show(truncate=False)

from pyspark.sql.functions import split, regexp\_extract

split\_df = base\_df.select(regexp\_extract('value', r'^([^\s]+\s)', 1).alias('host'),

regexp\_extract('value', r'^.\*\[(\d\d\/\w{3}\/\d{4}:\d{2}:\d{2}:\d{2} -\d{4})]', 1).alias('timestamp'),

regexp\_extract('value', r'^.\*"\w+\s+([^\s]+)\s+HTTP.\*"', 1).alias('path'),

regexp\_extract('value', r'^.\*"\s+([^\s]+)', 1).cast('integer').alias('status'),

regexp\_extract('value', r'^.\*\s+(\d+)$', 1).cast('integer').alias('content\_size'))

split\_df.show(5,truncate=False)

+———————+————————–+———————————————–+——+————+  
|host |timestamp |path |status|content\_size|  
+———————+————————–+———————————————–+——+————+  
|199.72.81.55 |01/Jul/1995:00:00:01 -0400|/history/apollo/ |200 |6245 |  
|unicomp6.unicomp.net |01/Jul/1995:00:00:06 -0400|/shuttle/countdown/ |200 |3985 |  
|199.120.110.21 |01/Jul/1995:00:00:09 -0400|/shuttle/missions/sts-73/mission-sts-73.html |200 |4085 |  
|burger.letters.com |01/Jul/1995:00:00:11 -0400|/shuttle/countdown/liftoff.html |304 |0 |  
|199.120.110.21 |01/Jul/1995:00:00:11 -0400|/shuttle/missions/sts-73/sts-73-patch-small.gif|200 |4179 |  
+———————+————————–+———————————————–+——+————+  
only showing top 5 rows

**2.2 Check data**

bad\_rows\_df = split\_df.filter(split\_df[‘host’].isNull() |

split\_df['timestamp'].isNull() |

split\_df['path'].isNull() |

split\_df['status'].isNull() |

split\_df['content\_size'].isNull())

bad\_rows\_df.count()

Out[20]: 33905

**2.3Check no of rows which do not have digits**

We have already seen that the content\_type field has ‘-‘ instead of digits in RDDs

#bad\_content\_size\_df = base\_df.filter(~ base\_df[‘value’].rlike(r’\d+$’))

bad\_content\_size\_df.count()

Out[21]: 33905

**2.4 Add ‘\*’ to identify bad rows**

To identify the rows that are bad, concatenate ‘\*’ to the content\_size field where the field does not have digits. It can be seen that the content\_size has ‘-‘ instead of a valid number

from pyspark.sql.functions import lit, concat

bad\_content\_size\_df.select(concat(bad\_content\_size\_df['value'], lit('\*'))).show(4,truncate=False)

+—————————————————————————————————————————————————+  
|concat(value, \*) |  
+—————————————————————————————————————————————————+  
|dd15-062.compuserve.com – – [01/Jul/1995:00:01:12 -0400] “GET /news/sci.space.shuttle/archive/sci-space-shuttle-22-apr-1995-40.txt HTTP/1.0” 404 -\*|  
|dynip42.efn.org – – [01/Jul/1995:00:02:14 -0400] “GET /software HTTP/1.0” 302 -\* |  
|ix-or10-06.ix.netcom.com – – [01/Jul/1995:00:02:40 -0400] “GET /software/winvn HTTP/1.0” 302 -\* |  
|ix-or10-06.ix.netcom.com – – [01/Jul/1995:00:03:24 -0400] “GET /software HTTP/1.0” 302 -\* |  
+—————————————————————————————————————————————————+

**2.5 Fill NAs with 0s**

# Replace all null content\_size values with 0.

cleaned\_df = split\_df.na.fill({‘content\_size’: 0})

**3. Webserver  logs parsing with SparkR**

library(SparkR)

library(stringr)

file\_location = "/FileStore/tables/NASA\_access\_log\_Jul95.gz"

file\_location = "/FileStore/tables/NASA\_access\_log\_Aug95.gz"

# Load the SparkR library

# Initiate a SparkR session

sparkR.session()

sc <- sparkR.session()

sqlContext <- sparkRSQL.init(sc)

df <- read.text(sqlContext,"/FileStore/tables/NASA\_access\_log\_Jul95.gz")

#df=SparkR::select(df, "value")

#head(SparkR::collect(df))

#m=regexp\_extract(df$value,'\\\\S+',1)

a=df %>%

withColumn('host', regexp\_extract(df$value, '^(\\S+)', 1)) %>%

withColumn('timestamp', regexp\_extract(df$value, "((\\S+ -\\d{4}))", 2)) %>%

withColumn('path', regexp\_extract(df$value, '(\\"\\w+\\s+([^\\s]+)\\s+HTTP.\*")', 2)) %>%

withColumn('status', regexp\_extract(df$value, '(^.\*"\\s+([^\\s]+))', 2)) %>%

withColumn('content\_size', regexp\_extract(df$value, '(^.\*\\s+(\\d+)$)', 2))

#b=a%>% select(host,timestamp,path,status,content\_type)

head(SparkR::collect(a),10)

1 199.72.81.55 – – [01/Jul/1995:00:00:01 -0400] “GET /history/apollo/ HTTP/1.0” 200 6245  
2 unicomp6.unicomp.net – – [01/Jul/1995:00:00:06 -0400] “GET /shuttle/countdown/ HTTP/1.0” 200 3985  
3 199.120.110.21 – – [01/Jul/1995:00:00:09 -0400] “GET /shuttle/missions/sts-73/mission-sts-73.html HTTP/1.0” 200 4085  
4 burger.letters.com – – [01/Jul/1995:00:00:11 -0400] “GET /shuttle/countdown/liftoff.html HTTP/1.0” 304 0  
5 199.120.110.21 – – [01/Jul/1995:00:00:11 -0400] “GET /shuttle/missions/sts-73/sts-73-patch-small.gif HTTP/1.0” 200 4179  
6 burger.letters.com – – [01/Jul/1995:00:00:12 -0400] “GET /images/NASA-logosmall.gif HTTP/1.0” 304 0  
7 burger.letters.com – – [01/Jul/1995:00:00:12 -0400] “GET /shuttle/countdown/video/livevideo.gif HTTP/1.0” 200 0  
8 205.212.115.106 – – [01/Jul/1995:00:00:12 -0400] “GET /shuttle/countdown/countdown.html HTTP/1.0” 200 3985  
9 d104.aa.net – – [01/Jul/1995:00:00:13 -0400] “GET /shuttle/countdown/ HTTP/1.0” 200 3985  
10 129.94.144.152 – – [01/Jul/1995:00:00:13 -0400] “GET / HTTP/1.0” 200 7074  
host timestamp  
1 199.72.81.55 [01/Jul/1995:00:00:01 -0400  
2 unicomp6.unicomp.net [01/Jul/1995:00:00:06 -0400  
3 199.120.110.21 [01/Jul/1995:00:00:09 -0400  
4 burger.letters.com [01/Jul/1995:00:00:11 -0400  
5 199.120.110.21 [01/Jul/1995:00:00:11 -0400  
6 burger.letters.com [01/Jul/1995:00:00:12 -0400  
7 burger.letters.com [01/Jul/1995:00:00:12 -0400  
8 205.212.115.106 [01/Jul/1995:00:00:12 -0400  
9 d104.aa.net [01/Jul/1995:00:00:13 -0400  
10 129.94.144.152 [01/Jul/1995:00:00:13 -0400  
path status content\_size  
1 /history/apollo/ 200 6245  
2 /shuttle/countdown/ 200 3985  
3 /shuttle/missions/sts-73/mission-sts-73.html 200 4085  
4 /shuttle/countdown/liftoff.html 304 0  
5 /shuttle/missions/sts-73/sts-73-patch-small.gif 200 4179  
6 /images/NASA-logosmall.gif 304 0  
7 /shuttle/countdown/video/livevideo.gif 200 0  
8 /shuttle/countdown/countdown.html 200 3985  
9 /shuttle/countdown/ 200 3985  
10 / 200 7074

**4 Webserver logs parsing with SparklyR**

install.packages("sparklyr")

library(sparklyr)

library(dplyr)

library(stringr)

#sc <- spark\_connect(master = "local", version = "2.1.0")

sc <- spark\_connect(method = "databricks")

sdf <-spark\_read\_text(sc, name="df", path = "/FileStore/tables/NASA\_access\_log\*.gz")

sdf

Installing package into ‘/databricks/spark/R/lib’

# Source: spark [?? x 1]

line

1 "199.72.81.55 - - [01/Jul/1995:00:00:01 -0400] \"GET /history/apollo/ HTTP/1…

2 "unicomp6.unicomp.net - - [01/Jul/1995:00:00:06 -0400] \"GET /shuttle/countd…

3 "199.120.110.21 - - [01/Jul/1995:00:00:09 -0400] \"GET /shuttle/missions/sts…

4 "burger.letters.com - - [01/Jul/1995:00:00:11 -0400] \"GET /shuttle/countdow…

5 "199.120.110.21 - - [01/Jul/1995:00:00:11 -0400] \"GET /shuttle/missions/sts…

6 "burger.letters.com - - [01/Jul/1995:00:00:12 -0400] \"GET /images/NASA-logo…

7 "burger.letters.com - - [01/Jul/1995:00:00:12 -0400] \"GET /shuttle/countdow…

8 "205.212.115.106 - - [01/Jul/1995:00:00:12 -0400] \"GET /shuttle/countdown/c…

9 "d104.aa.net - - [01/Jul/1995:00:00:13 -0400] \"GET /shuttle/countdown/ HTTP…

10 "129.94.144.152 - - [01/Jul/1995:00:00:13 -0400] \"GET / HTTP/1.0\" 200 7074"

# … with more rows

#install.packages(“sparklyr”)

library(sparklyr)

library(dplyr)

library(stringr)

#sc <- spark\_connect(master = "local", version = "2.1.0")

sc <- spark\_connect(method = "databricks")

sdf <-spark\_read\_text(sc, name="df", path = "/FileStore/tables/NASA\_access\_log\*.gz")

sdf <- sdf %>% mutate(host = regexp\_extract(line, '^(\\\\S+)',1)) %>%

mutate(timestamp = regexp\_extract(line, '((\\\\S+ -\\\\d{4}))',2)) %>%

mutate(path = regexp\_extract(line, '(\\\\"\\\\w+\\\\s+([^\\\\s]+)\\\\s+HTTP.\*")',2)) %>%

mutate(status = regexp\_extract(line, '(^.\*"\\\\s+([^\\\\s]+))',2)) %>%

mutate(content\_size = regexp\_extract(line, '(^.\*\\\\s+(\\\\d+)$)',2))

**5 Hosts**

**5.1  RDD**

**5.11 Parse and map to hosts to groups**

parsed\_rdd = rdd.map(lambda line: parse\_log2(line)).filter(lambda line: line[1] == 1).map(lambda line : line[0])

parsed\_rdd2 = parsed\_rdd.map(lambda line: map2groups(line))

# Create tuples of (host,1) and apply reduceByKey() and order by descending

rslt=(parsed\_rdd2.map(lambda xx[0],1))

.reduceByKey(lambda a,b:a+b)

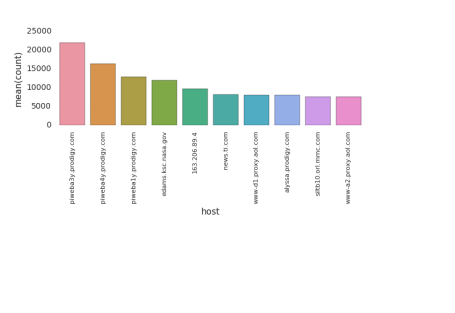
.takeOrdered(10, lambda x: -x[1]))

rslt

Out[18]:  
[(‘piweba3y.prodigy.com’, 21988),  
(‘piweba4y.prodigy.com’, 16437),  
(‘piweba1y.prodigy.com’, 12825),  
(‘edams.ksc.nasa.gov’, 11962),  
(‘163.206.89.4’, 9697),  
(‘news.ti.com’, 8161),  
(‘www-d1.proxy.aol.com’, 8047),  
(‘alyssa.prodigy.com’, 8037),  
(‘siltb10.orl.mmc.com’, 7573),  
(‘www-a2.proxy.aol.com’, 7516)]

**5.12Plot counts of hosts**

import seaborn as sns

import pandas as pd import matplotlib.pyplot as plt df=pd.DataFrame(rslt,columns=[‘host’,‘count’]) sns.barplot(x=‘host’,y=‘count’,data=df) plt.subplots\_adjust(bottom=0.6, right=0.8, top=0.9) plt.xticks(rotation=“vertical”,fontsize=8) display()  


**5.2 PySpark**

**5.21 Compute counts of hosts**

df= (cleaned\_df

.groupBy('host')

.count()

.orderBy('count',ascending=False))

df.show(5)

+——————–+—–+  
| host|count|  
+——————–+—–+  
|piweba3y.prodigy….|21988|  
|piweba4y.prodigy….|16437|  
|piweba1y.prodigy….|12825|  
| edams.ksc.nasa.gov |11964|  
| 163.206.89.4 | 9697|  
+——————–+—–+  
only showing top 5 rows

**5.22 Plot count of hosts**

import matplotlib.pyplot as plt

import pandas as pd

import seaborn as sns

df1=df.toPandas()

df2 = df1.head(10)

df2.count()

sns.barplot(x='host',y='count',data=df2)

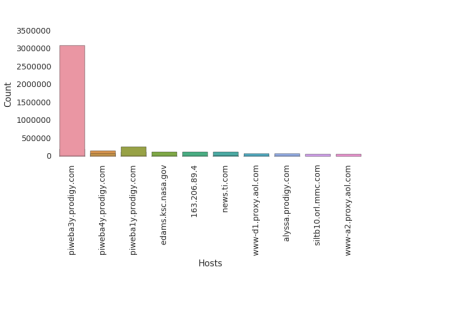
plt.subplots\_adjust(bottom=0.5, right=0.8, top=0.9)

plt.xlabel("Hosts")

plt.ylabel('Count')

plt.xticks(rotation="vertical",fontsize=10)

display()



**5.3 SparkR**

**5.31 Compute count of hosts**

c <- SparkR::select(a,a$host)

df=SparkR::summarize(SparkR::groupBy(c, a$host), noHosts = count(a$host))

df1 =head(arrange(df,desc(df$noHosts)),10)

head(df1)

                  host noHosts

1 piweba3y.prodigy.com 17572

2 piweba4y.prodigy.com 11591

3 piweba1y.prodigy.com 9868

4 alyssa.prodigy.com 7852

5 siltb10.orl.mmc.com 7573

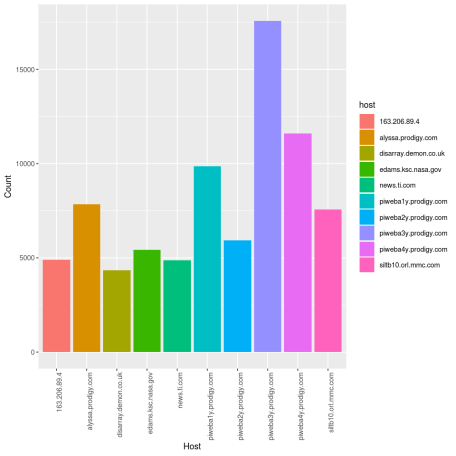
6 piweba2y.prodigy.com 5922

**5.32 Plot count of hosts**

library(ggplot2)

p <-ggplot(data=df1, aes(x=host, y=noHosts,fill=host)) + geom\_bar(stat="identity") + theme(axis.text.x = element\_text(angle = 90, hjust = 1)) + xlab('Host') + ylab('Count')

p



**5.4 SparklyR**

**5.41 Compute count of Hosts**

df <- sdf %>% select(host,timestamp,path,status,content\_size)

df1 <- df %>% select(host) %>% group\_by(host) %>% summarise(noHosts=n()) %>% arrange(desc(noHosts))

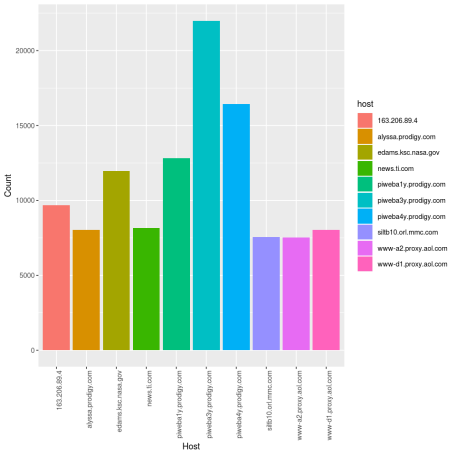
df2 <-head(df1,10)

**5.42 Plot count of hosts**

library(ggplot2)

p <-ggplot(data=df2, aes(x=host, y=noHosts,fill=host)) + geom\_bar(stat=“identity”)+ theme(axis.text.x = element\_text(angle = 90, hjust = 1)) + xlab(‘Host’) + ylab(‘Count’)

p



**6 Paths**

**6.1 RDD**

**6.11 Parse and map to hosts to groups**

parsed\_rdd = rdd.map(lambda line: parse\_log2(line)).filter(lambda line: line[1] == 1).map(lambda line : line[0])

parsed\_rdd2 = parsed\_rdd.map(lambda line: map2groups(line))

rslt=(parsed\_rdd2.map(lambda xx[5],1))

.reduceByKey(lambda a,b:a+b)

.takeOrdered(10, lambda x: -x[1]))

rslt

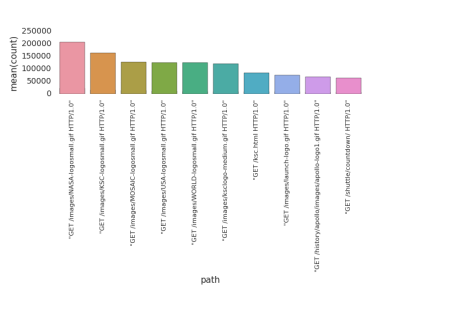
[(‘”GET /images/NASA-logosmall.gif HTTP/1.0″‘, 207520),  
(‘”GET /images/KSC-logosmall.gif HTTP/1.0″‘, 164487),  
(‘”GET /images/MOSAIC-logosmall.gif HTTP/1.0″‘, 126933),  
(‘”GET /images/USA-logosmall.gif HTTP/1.0″‘, 126108),  
(‘”GET /images/WORLD-logosmall.gif HTTP/1.0″‘, 124972),  
(‘”GET /images/ksclogo-medium.gif HTTP/1.0″‘, 120704),  
(‘”GET /ksc.html HTTP/1.0″‘, 83209),  
(‘”GET /images/launch-logo.gif HTTP/1.0″‘, 75839),  
(‘”GET /history/apollo/images/apollo-logo1.gif HTTP/1.0″‘, 68759),  
(‘”GET /shuttle/countdown/ HTTP/1.0″‘, 64467)]

**6.12 Plot counts of HTTP Requests**

import seaborn as sns

df=pd.DataFrame(rslt,columns=[‘path’,‘count’]) sns.barplot(x=‘path’,y=‘count’,data=df) plt.subplots\_adjust(bottom=0.7, right=0.8, top=0.9) plt.xticks(rotation=“vertical”,fontsize=8)

display()



**6.2 Pyspark**

**6.21 Compute count of HTTP Requests**

df= (cleaned\_df

.groupBy('path')

.count()

.orderBy('count',ascending=False))

df.show(5)

Out[20]:  
+——————–+——+  
| path| count|  
+——————–+——+  
|/images/NASA-logo…|208362|  
|/images/KSC-logos…|164813|  
|/images/MOSAIC-lo…|127656|  
|/images/USA-logos…|126820|  
|/images/WORLD-log…|125676|  
+——————–+——+  
only showing top 5 rows

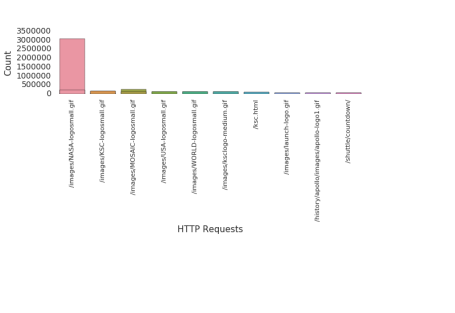
**6.22 Plot count of HTTP Requests**

import matplotlib.pyplot as plt

import pandas as pd import seaborn as sns df1=df.toPandas() df2 = df1.head(10) df2.count() sns.barplot(x=‘path’,y=‘count’,data=df2)

plt.subplots\_adjust(bottom=0.7, right=0.8, top=0.9) plt.xlabel(“HTTP Requests”) plt.ylabel(‘Count’) plt.xticks(rotation=90,fontsize=8)

display()



**6.3 SparkR**

**6.31Compute count of HTTP requests**

library(SparkR)

c <- SparkR::select(a,a$path)

df=SparkR::summarize(SparkR::groupBy(c, a$path), numRequest = count(a$path))

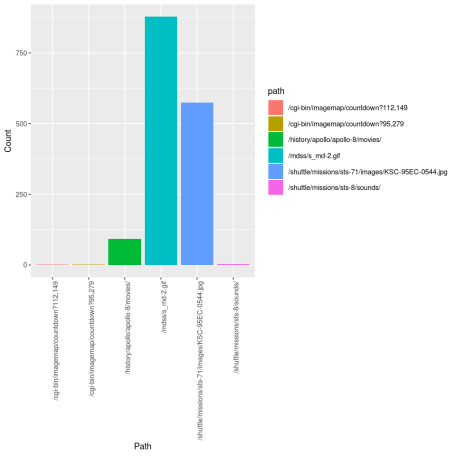
df1=head(df)

**3.14 Plot count of HTTP Requests**

library(ggplot2)

p <-ggplot(data=df1, aes(x=path, y=numRequest,fill=path)) + geom\_bar(stat="identity") + theme(axis.text.x = element\_text(angle = 90, hjust = 1))+ xlab('Path') + ylab('Count')

p



**6.4 SparklyR**

**6.41 Compute count of paths**

df <- sdf %>% select(host,timestamp,path,status,content\_size)

df1 <- df %>% select(path) %>% group\_by(path) %>% summarise(noPaths=n()) %>% arrange(desc(noPaths))

df2 <-head(df1,10)

df2

# Source: spark [?? x 2]

# Ordered by: desc(noPaths)

path noPaths

1 /images/NASA-logosmall.gif 208362

2 /images/KSC-logosmall.gif 164813

3 /images/MOSAIC-logosmall.gif 127656

4 /images/USA-logosmall.gif 126820

5 /images/WORLD-logosmall.gif 125676

6 /images/ksclogo-medium.gif 121286

7 /ksc.html 83685

8 /images/launch-logo.gif 75960

9 /history/apollo/images/apollo-logo1.gif 68858

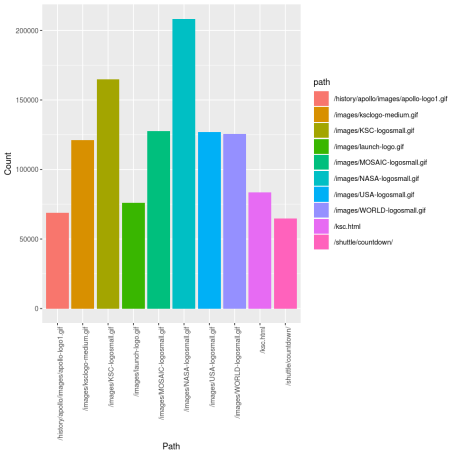
10 /shuttle/countdown/ 64695

**6.42 Plot count of Paths**

library(ggplot2)

p <-ggplot(data=df2, aes(x=path, y=noPaths,fill=path)) + geom\_bar(stat="identity")+ theme(axis.text.x = element\_text(angle = 90, hjust = 1)) + xlab('Path') + ylab('Count')

p



**7.1 RDD**

**7.11 Compute count of HTTP Status**

parsed\_rdd = rdd.map(lambda line: parse\_log2(line)).filter(lambda line: line[1] == 1).map(lambda line : line[0])

parsed\_rdd2 = parsed\_rdd.map(lambda line: map2groups(line))

rslt=(parsed\_rdd2.map(lambda xx[7],1))

.reduceByKey(lambda a,b:a+b)

.takeOrdered(10, lambda x: -x[1]))

rslt

Out[22]:

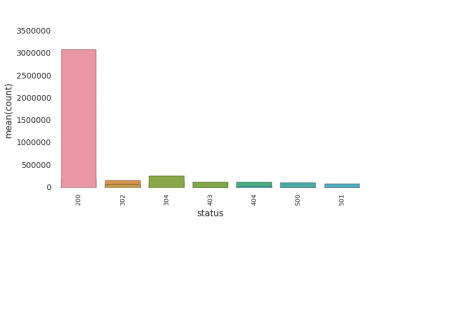
[(‘200’, 3095682),  
(‘304’, 266764),  
(‘302’, 72970),  
(‘404’, 20625),  
(‘403’, 225),  
(‘500’, 65),  
(‘501’, 41)]

**1.37 Plot counts of HTTP response status’**

import seaborn as sns

df=pd.DataFrame(rslt,columns=[‘status’,‘count’]) sns.barplot(x=‘status’,y=‘count’,data=df) plt.subplots\_adjust(bottom=0.4, right=0.8, top=0.9) plt.xticks(rotation=“vertical”,fontsize=8)

display()



**7.2 Pyspark**

**7.21 Compute count of HTTP status**

status\_count=(cleaned\_df

.groupBy('status')

.count()

.orderBy('count',ascending=False))

status\_count.show()

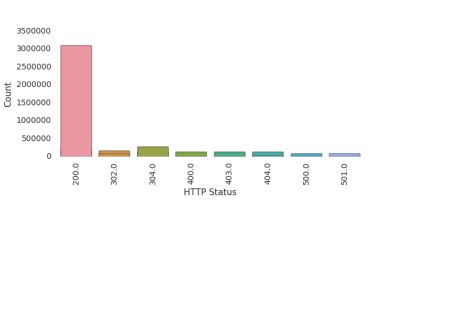
+——+——-+  
|status| count|  
+——+——-+  
| 200|3100522|  
| 304| 266773|  
| 302| 73070|  
| 404| 20901|  
| 403| 225|  
| 500| 65|  
| 501| 41|  
| 400| 15|  
| null| 1|

**7.22 Plot count of HTTP status**

Plot the HTTP return status vs the counts

df1=status\_count.toPandas()

df2 = df1.head(10) df2.count() sns.barplot(x=‘status’,y=‘count’,data=df2) plt.subplots\_adjust(bottom=0.5, right=0.8, top=0.9) plt.xlabel(“HTTP Status”) plt.ylabel(‘Count’) plt.xticks(rotation=“vertical”,fontsize=10) display()



**7.3 SparkR**

**7.31 Compute count of HTTP Response status**

library(SparkR)

c <- SparkR::select(a,a$status)

df=SparkR::summarize(SparkR::groupBy(c, a$status), numStatus = count(a$status))

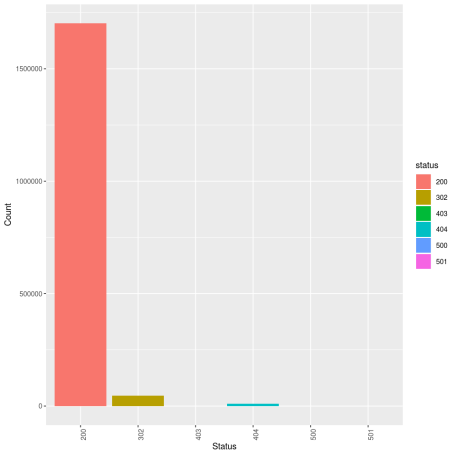
df1=head(df)

**3.16 Plot count of HTTP Response status**

library(ggplot2)

p <-ggplot(data=df1, aes(x=status, y=numStatus,fill=status)) + geom\_bar(stat="identity") + theme(axis.text.x = element\_text(angle = 90, hjust = 1)) + xlab('Status') + ylab('Count')

p



**7.4 SparklyR**

**7.41 Compute count of status**

df <- sdf %>% select(host,timestamp,path,status,content\_size)

df1 <- df %>% select(status) %>% group\_by(status) %>% summarise(noStatus=n()) %>% arrange(desc(noStatus))

df2 <-head(df1,10)

df2

# Source: spark [?? x 2]

# Ordered by: desc(noStatus)

status noStatus

1 200 3100522

2 304 266773

3 302 73070

4 404 20901

5 403 225

6 500 65

7 501 41

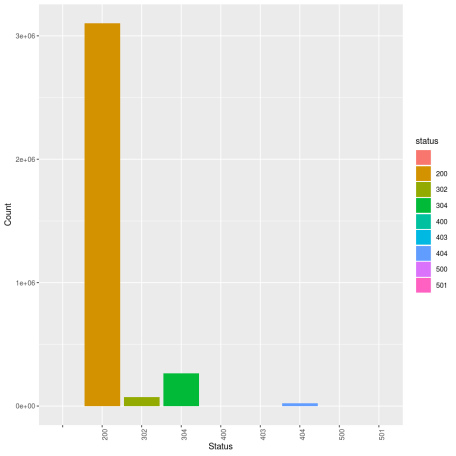
8 400 15

9 "" 1

**7.42 Plot count of status**

library(ggplot2)

p <-ggplot(data=df2, aes(x=status, y=noStatus,fill=status)) + geom\_bar(stat=“identity”)+ theme(axis.text.x = element\_text(angle = 90, hjust = 1)) + xlab(‘Status’) + ylab(‘Count’) p



**8.1 RDD**

**8.12 Compute count of content size**

parsed\_rdd = rdd.map(lambda line: parse\_log2(line)).filter(lambda line: line[1] == 1).map(lambda line : line[0])

parsed\_rdd2 = parsed\_rdd.map(lambda line: map2groups(line))

rslt=(parsed\_rdd2.map(lambda xx[8],1))

.reduceByKey(lambda a,b:a+b)

.takeOrdered(10, lambda x: -x[1]))

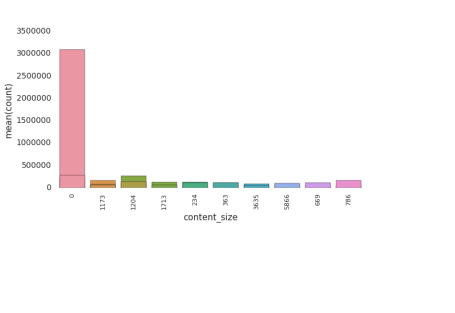
rslt

Out[24]:  
[(‘0’, 280017),  
(‘786’, 167281),  
(‘1204’, 140505),  
(‘363’, 111575),  
(‘234’, 110824),  
(‘669’, 110056),  
(‘5866’, 107079),  
(‘1713’, 66904),  
(‘1173’, 63336),  
(‘3635’, 55528)]

**8.21 Plot content size**

import seaborn as sns

df=pd.DataFrame(rslt,columns=[‘content\_size’,‘count’]) sns.barplot(x=‘content\_size’,y=‘count’,data=df) plt.subplots\_adjust(bottom=0.4, right=0.8, top=0.9) plt.xticks(rotation=“vertical”,fontsize=8) display()



**8.2 Pyspark**

**8.21 Compute count of content\_size**

size\_counts=(cleaned\_df

.groupBy('content\_size')

.count()

.orderBy('count',ascending=False))

size\_counts.show(10)

+------------+------+

|content\_size| count|

+------------+------+

| 0|313932|

| 786|167709|

| 1204|140668|

| 363|111835|

| 234|111086|

| 669|110313|

| 5866|107373|

| 1713| 66953|

| 1173| 63378|

| 3635| 55579|

+------------+------+

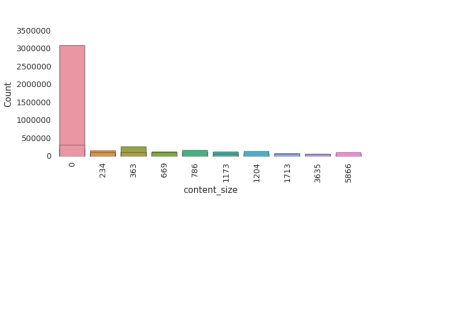
only showing top 10 rows

**8.22 Plot counts of content size**

Plot the path access versus the counts

df1=size\_counts.toPandas()

df2 = df1.head(10) df2.count() sns.barplot(x=‘content\_size’,y=‘count’,data=df2) plt.subplots\_adjust(bottom=0.5, right=0.8, top=0.9) plt.xlabel(“content\_size”) plt.ylabel(‘Count’) plt.xticks(rotation=“vertical”,fontsize=10) display()



**8.3 SparkR**

**8.31 Compute count of content size**

library(SparkR)

c <- SparkR::select(a,a$content\_size)

df=SparkR::summarize(SparkR::groupBy(c, a$content\_size), numContentSize = count(a$content\_size))

df1=head(df)

df1

     content\_size numContentSize

1 28426 1414

2 78382 293

3 60053 4

4 36067 2

5 13282 236

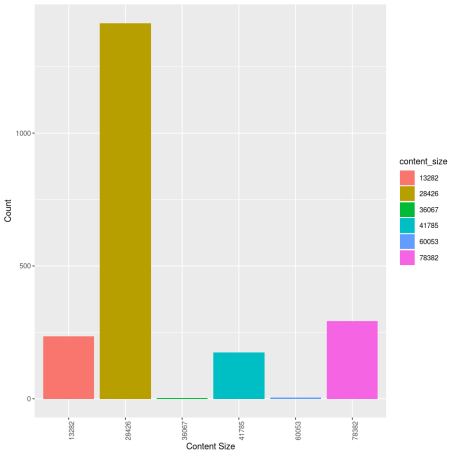
6 41785 174

8.32 Plot count of content sizes

library(ggplot2)

p <-ggplot(data=df1, aes(x=content\_size, y=numContentSize,fill=content\_size)) + geom\_bar(stat=“identity”) + theme(axis.text.x = element\_text(angle = 90, hjust = 1)) + xlab(‘Content Size’) + ylab(‘Count’)

p



**8.4 SparklyR**

**8.41Compute count of content\_size**

df <- sdf %>% select(host,timestamp,path,status,content\_size)

df1 <- df %>% select(content\_size) %>% group\_by(content\_size) %>% summarise(noContentSize=n()) %>% arrange(desc(noContentSize))

df2 <-head(df1,10)

df2

# Source: spark [?? x 2]

# Ordered by: desc(noContentSize)

content\_size noContentSize

1 0 280027

2 786 167709

3 1204 140668

4 363 111835

5 234 111086

6 669 110313

7 5866 107373

8 1713 66953

9 1173 63378

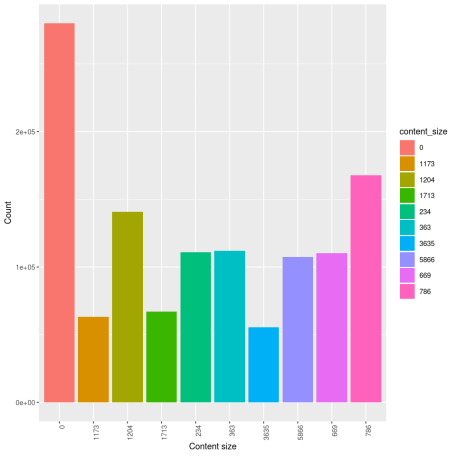
10 3635 55579

**8.42 Plot count of content\_size**

library(ggplot2)

p <-ggplot(data=df2, aes(x=content\_size, y=noContentSize,fill=content\_size)) + geom\_bar(stat="identity")+ theme(axis.text.x = element\_text(angle = 90, hjust = 1)) + xlab('Content size') + ylab('Count')

p



Conclusion: I spent many,many hours struggling with Regex and getting RDDs,Pyspark to work. Also had to spend a lot of time trying to work out the syntax for SparkR and SparklyR for parsing. After you parse the logs plotting and analysis is a piece of cake! This is definitely worth a try!

Watch this space!!