PD2: Web Server Log Analysis

MTI850 - Big Data Analytics

Fall 2021

Equipe 1

```
! sudo apt-get update
! sudo mkdir -p /usr/share/man/man1
! sudo apt-get install -y openjdk-11-jdk
! pip install pyspark
```

```
Hit:1 http://deb.debian.org/debian buster InRelease
Hit:2 http://deb.debian.org/debian-security buster/updates InRelease
Hit:3 http://deb.debian.org/debian buster-updates InRelease
```

```
openjdk-11-jdk is already the newest version (11.0.16+8-1~deb10u1).
0 upgraded, 0 newly installed, 0 to remove and 28 not upgraded.
Requirement already satisfied: pyspark in
/root/venv/lib/python3.9/site-packages (3.3.0)
Requirement already satisfied: py4j==0.10.9.5 in
/root/venv/lib/python3.9/site-packages (from pyspark) (0.10.9.5)
WARNING: You are using pip version 22.0.4; however, version 22.2.2 is
available.
You should consider upgrading via the '/root/venv/bin/python -m pip
install --upgrade pip' command.
```

This assignment will show how easy it is to perform web server log analysis with Apache Spark.

Server log analysis is an ideal use case for Spark. It's a very large, common data source and contains a rich set of information. Spark allows you to store your logs in files on disk cheaply, while still providing a quick and simple way to perform data analysis on them.

This assignment will show you how to use Apache Spark on real-world text-based production logs and fully harness the power of that data.

Log data comes from many sources, such as web, file, and compute servers, application logs, user-generated content, and can be used for monitoring servers, improving business and customer intelligence, building recommendation systems, fraud detection, and much more.

How to complete this assignment

This assignment is broken up into sections examples for demonstrating Spark functionality for log processing.

It consists of 5 parts:

- *Part 1*: Introduction and Imports
- *Part 2*: Exploratory Data Analysis
- *Part 3*: Analysis Walk-Through on the Web Server Log File
- *Part 4*: Analyzing Web Server Log File
- *Part 5*: Exploring 404 Response Codes

Part 1: Introduction and Imports

```
#import findspark
#findspark.init()

# Test module for MTI850
import testmti850
```

```
# Util module for MTI850
import utilmti850

import pyspark
from pyspark.sql import SparkSession

spark = SparkSession.builder \
.master("local") \
.appName("Web Server Log Analysis") \
.config("spark.some.config.option", "some-value") \
.getOrCreate()
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
22/10/12 03:39:38 WARN NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where
applicable
```

A note about DataFrame column references

In Python, it's possible to access a DataFrame's columns either by attribute (`df.age`) or by indexing (`df['age']`). Referring to a column by attribute (`df.age`) is very Pandas-like, and it's highly convenient, especially when you're doing interactive data exploration. But it can fail, for reasons that aren't obvious. To observe this fact, uncomment the second line and run the next cell (after that, comment again the second line).

```
throwaway_df = spark.createDataFrame([('Anthony', 10), ('Julia', 20),
('Fred', 5)], ('name', 'count'))

# throwaway_df.select(throwaway_df.count).show() # This line does not
work. Please comment it out later.
```

To understand why that failed, you have to understand how the attribute-column syntax is implemented.

When you type `throwaway_df.count`, Python looks for an *existing* attribute or method called `count` on the `throwaway_df` object. If it finds one, it uses it. Otherwise, it calls a special Python function (`__getattr__`), which defaults to throwing an exception. Spark has overridden `__getattr__` to look for a column on the DataFrame.

This means you can only use the attribute (dot) syntax to refer to a column if the DataFrame does not *already* have an attribute with the column's name.

In the above example, there's already a `count()` method on the `DataFrame` class, so `throwaway_df.count` does not refer to our "count" column; instead, it refers to the `count()` *method*.

To avoid this problem, you can refer to the column using subscript notation: `throwaway_df['count']`. This syntax will *always* work.

```
throwaway_df.select(throwaway_df['count']).show()
```

```
+-----+
|count|
+-----+
|   10|
|   20|
|    5|
+-----+
```

(1a) Library Imports

We can import standard Python3 libraries (modules) the usual way. An `import` statement will import the specified module. In this assignment, we will provide any imports that are necessary.

Let's import some of the libraries we'll need:

- `re`: The regular expression library
- `datetime`: Date and time functions

```
import re
import datetime
```

```
# Quick test of the regular expression library
m = re.search('(?<=abc)def', 'abcdef')
m.group(0)
```

```
'def'
```

```
# Quick test of the datetime library
print ('This was last run on: {0}'.format(datetime.datetime.now()))
```

```
This was last run on: 2022-10-12 03:39:50.710049
```

(1b) Getting help

Remember: There are some useful Python built-ins for getting help.

You can use Python's dir() function to get a list of all the attributes (including methods) accessible through the `spark` object.

```
dir(spark)
```

```
['Builder',
 '__annotations__',
 '__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__enter__',
```

```
'__eq__',
'__exit__',
'__format__',
'__ge__',
'__getattribute__',
'__gt__',
'__hash__',
'__init__',
'__init_subclass__',
'__le__',
'__lt__',
'__module__',
'__ne__',
'__new__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__setattr__',
'__sizeof__',
'__str__',
'__subclasshook__',
'__weakref__',
'_activeSession',
'_convert_from_pandas',
'_createFromLocal',
'_createFromRDD',
'_create_dataframe',
'_create_from_pandas_with_arrow',
'_create_shell_session',
'_getActiveSessionOrCreate',
'_get_numpy_record_dtype',
'_inferSchema',
'_inferSchemaFromList',
'_instantiatedSession',
'_jconf',
'_jsc',
'_jsparkSession',
'_jvm',
'_repr_html_',
'_sc',
'builder',
'catalog',
'conf',
'createDataFrame',
'getActiveSession',
'newSession',
'range',
'read',
'readStream',
'sparkContext',
```

```
 'sql',
 'stop',
 'streams',
 'table',
 'udf',
 'version']
```

Alternatively, you can use Python's help() function to get an easier to read list of all the attributes, including examples, that the `spark` object has.

```
# Use help to obtain more detailed information
help(spark)
```

```
Help on SparkSession in module pyspark.sql.session object:

class SparkSession(pyspark.sql.pandas.conversion.SparkConversionMixin)
 |  SparkSession(sparkContext: pyspark.context.SparkContext,
jsparkSession: Optional[py4j.java_gateway.JavaObject] = None, options:
Dict[str, Any] = {})
 |
 |  The entry point to programming Spark with the Dataset and
DataFrame API.
 |
 |  A SparkSession can be used create :class:`DataFrame`,
register :class:`DataFrame` as
 |  tables, execute SQL over tables, cache tables, and read parquet
files.
 |  To create a :class:`SparkSession`, use the following builder
pattern:
 |
 |  .. autoattribute:: builder
 |     :annotation:
 |
 |  Examples
 |  --------
 |  >>> spark = SparkSession.builder \
 |  ...     .master("local") \
 |  ...     .appName("Word Count") \
 |  ...     .config("spark.some.config.option", "some-value") \
 |  ...     .getOrCreate()
 |
 |  >>> from datetime import datetime
 |  >>> from pyspark.sql import Row
 |  >>> spark = SparkSession(sc)
 |  >>> allTypes = sc.parallelize([Row(i=1, s="string", d=1.0, l=1,
 |  ...     b=True, list=[1, 2, 3], dict={"s": 0}, row=Row(a=1),
 |  ...     time=datetime(2014, 8, 1, 14, 1, 5))])
 |  >>> df = allTypes.toDF()
 |  >>> df.createOrReplaceTempView("allTypes")
 |  >>> spark.sql('select i+1, d+1, not b, list[1], dict["s"], time,
row.a '
```

```
 |  ...             'from allTypes where b and i > 0').collect()
 |  [Row((i + 1)=2, (d + 1)=2.0, (NOT b)=False, list[1]=2,
dict[s]=0, time=datetime.datetime(2014, 8, 1, 14, 1, 5), a=1)]
 |  >>> df.rdd.map(lambda x: (x.i, x.s, x.d, x.l, x.b, x.time,
x.row.a, x.list)).collect()
 |  [(1, 'string', 1.0, 1, True, datetime.datetime(2014, 8, 1, 14, 1,
5), 1, [1, 2, 3])]
 |
 |  Method resolution order:
 |      SparkSession
 |      pyspark.sql.pandas.conversion.SparkConversionMixin
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __enter__(self) -> 'SparkSession'
 |      Enable 'with SparkSession.builder.(...).getOrCreate() as
session: app' syntax.
 |
 |      .. versionadded:: 2.0
 |
 |  __exit__(self, exc_type: Optional[Type[BaseException]], exc_val:
Optional[BaseException], exc_tb: Optional[traceback]) -> None
 |      Enable 'with SparkSession.builder.(...).getOrCreate() as
session: app' syntax.
 |
 |      Specifically stop the SparkSession on exit of the with block.
 |
 |      .. versionadded:: 2.0
 |
 |  __init__(self, sparkContext: pyspark.context.SparkContext,
jsparkSession: Optional[py4j.java_gateway.JavaObject] = None, options:
Dict[str, Any] = {})
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  createDataFrame(self, data: Union[pyspark.rdd.RDD[Any],
Iterable[Any], ForwardRef('PandasDataFrameLike')], schema:
Union[pyspark.sql.types.AtomicType, pyspark.sql.types.StructType, str,
NoneType] = None, samplingRatio: Optional[float] = None, verifySchema:
bool = True) -> pyspark.sql.dataframe.DataFrame
 |      Creates a :class:`DataFrame` from an :class:`RDD`, a list or a
:class:`pandas.DataFrame`.
 |
 |      When ``schema`` is a list of column names, the type of each
column
 |      will be inferred from ``data``.
 |
 |      When ``schema`` is ``None``, it will try to infer the schema
(column names and types)
 |      from ``data``, which should be an RDD of either :class:`Row`,
```

```
 |      :class:`namedtuple`, or :class:`dict`.
 |
 |      When ``schema`` is :class:`pyspark.sql.types.DataType` or a
datatype string, it must match
 |      the real data, or an exception will be thrown at runtime. If
the given schema is not
 |      :class:`pyspark.sql.types.StructType`, it will be wrapped into
a
 |      :class:`pyspark.sql.types.StructType` as its only field, and
the field name will be "value".
 |      Each record will also be wrapped into a tuple, which can be
converted to row later.
 |
 |      If schema inference is needed, ``samplingRatio`` is used to
determined the ratio of
 |      rows used for schema inference. The first row will be used if
``samplingRatio`` is ``None``.
 |
 |      .. versionadded:: 2.0.0
 |
 |      .. versionchanged:: 2.1.0
 |         Added verifySchema.
 |
 |      Parameters
 |      ----------
 |      data : :class:`RDD` or iterable
 |          an RDD of any kind of SQL data representation
(:class:`Row`,
 |          :class:`tuple`, ``int``, ``boolean``, etc.),
or :class:`list`, or
 |          :class:`pandas.DataFrame`.
 |      schema : :class:`pyspark.sql.types.DataType`, str or list,
optional
 |          a :class:`pyspark.sql.types.DataType` or a datatype string
or a list of
 |          column names, default is None.  The data type string
format equals to
 |          :class:`pyspark.sql.types.DataType.simpleString`, except
that top level struct type can
 |          omit the ``struct<>``.
 |      samplingRatio : float, optional
 |          the sample ratio of rows used for inferring
 |      verifySchema : bool, optional
 |          verify data types of every row against schema. Enabled by
default.
 |
 |      Returns
 |      -------
 |      :class:`DataFrame`
 |
```

```
 |      Notes
 |      -----
 |      Usage with spark.sql.execution.arrow.pyspark.enabled=True is
experimental.
 |
 |      Examples
 |      --------
 |      >>> l = [('Alice', 1)]
 |      >>> spark.createDataFrame(l).collect()
 |      [Row(_1='Alice', _2=1)]
 |      >>> spark.createDataFrame(l, ['name', 'age']).collect()
 |      [Row(name='Alice', age=1)]
 |
 |      >>> d = [{'name': 'Alice', 'age': 1}]
 |      >>> spark.createDataFrame(d).collect()
 |      [Row(age=1, name='Alice')]
 |
 |      >>> rdd = sc.parallelize(l)
 |      >>> spark.createDataFrame(rdd).collect()
 |      [Row(_1='Alice', _2=1)]
 |      >>> df = spark.createDataFrame(rdd, ['name', 'age'])
 |      >>> df.collect()
 |      [Row(name='Alice', age=1)]
 |
 |      >>> from pyspark.sql import Row
 |      >>> Person = Row('name', 'age')
 |      >>> person = rdd.map(lambda r: Person(*r))
 |      >>> df2 = spark.createDataFrame(person)
 |      >>> df2.collect()
 |      [Row(name='Alice', age=1)]
 |
 |      >>> from pyspark.sql.types import *
 |      >>> schema = StructType([
 |      ...    StructField("name", StringType(), True),
 |      ...    StructField("age", IntegerType(), True)])
 |      >>> df3 = spark.createDataFrame(rdd, schema)
 |      >>> df3.collect()
 |      [Row(name='Alice', age=1)]
 |
 |      >>> spark.createDataFrame(df.toPandas()).collect()  # doctest:
+SKIP
 |      [Row(name='Alice', age=1)]
 |      >>> spark.createDataFrame(pandas.DataFrame([[1,
2]])).collect()  # doctest: +SKIP
 |      [Row(0=1, 1=2)]
 |
 |      >>> spark.createDataFrame(rdd, "a: string, b: int").collect()
 |      [Row(a='Alice', b=1)]
 |      >>> rdd = rdd.map(lambda row: row[1])
 |      >>> spark.createDataFrame(rdd, "int").collect()
```

```
 |      [Row(value=1)]
 |      >>> spark.createDataFrame(rdd, "boolean").collect() # doctest:
+IGNORE_EXCEPTION_DETAIL
 |      Traceback (most recent call last):
 |          ...
 |      Py4JJavaError: ...
 |
 |  newSession(self) -> 'SparkSession'
 |      Returns a new :class:`SparkSession` as new session, that has
separate SQLConf,
 |      registered temporary views and UDFs, but
shared :class:`SparkContext` and
 |      table cache.
 |
 |      .. versionadded:: 2.0
 |
 |  range(self, start: int, end: Optional[int] = None, step: int = 1,
numPartitions: Optional[int] = None) ->
pyspark.sql.dataframe.DataFrame
 |      Create a :class:`DataFrame` with
single :class:`pyspark.sql.types.LongType` column named
 |      ``id``, containing elements in a range from ``start`` to
``end`` (exclusive) with
 |      step value ``step``.
 |
 |      .. versionadded:: 2.0.0
 |
 |      Parameters
 |      ----------
 |      start : int
 |          the start value
 |      end : int, optional
 |          the end value (exclusive)
 |      step : int, optional
 |          the incremental step (default: 1)
 |      numPartitions : int, optional
 |          the number of partitions of the DataFrame
 |
 |      Returns
 |      -------
 |      :class:`DataFrame`
 |
 |      Examples
 |      --------
 |      >>> spark.range(1, 7, 2).collect()
 |      [Row(id=1), Row(id=3), Row(id=5)]
 |
 |      If only one argument is specified, it will be used as the end
value.
 |
```

```
 |      >>> spark.range(3).collect()
 |      [Row(id=0), Row(id=1), Row(id=2)]
 |
 |  sql(self, sqlQuery: str, **kwargs: Any) ->
pyspark.sql.dataframe.DataFrame
 |      Returns a :class:`DataFrame` representing the result of the
given query.
 |      When ``kwargs`` is specified, this method formats the given
string by using the Python
 |      standard formatter.
 |
 |      .. versionadded:: 2.0.0
 |
 |      Parameters
 |      ----------
 |      sqlQuery : str
 |          SQL query string.
 |      kwargs : dict
 |          Other variables that the user wants to set that can be
referenced in the query
 |
 |          .. versionchanged:: 3.3.0
 |             Added optional argument ``kwargs`` to specify the
mapping of variables in the query.
 |             This feature is experimental and unstable.
 |
 |      Returns
 |      -------
 |      :class:`DataFrame`
 |
 |      Examples
 |      --------
 |      Executing a SQL query.
 |
 |      >>> spark.sql("SELECT * FROM range(10) where id > 7").show()
 |      +---+
 |      | id|
 |      +---+
 |      |  8|
 |      |  9|
 |      +---+
 |
 |      Executing a SQL query with variables as Python formatter
standard.
 |
 |      >>> spark.sql(
 |      ...     "SELECT * FROM range(10) WHERE id > {bound1} AND id <
{bound2}", bound1=7, bound2=9
 |      ... ).show()
 |      +---+
```

```
|      | id|
|      +---+
|      |  8|
|      +---+
|
|      >>> mydf = spark.range(10)
|      >>> spark.sql(
|      ...     "SELECT {col} FROM {mydf} WHERE id IN {x}",
|      ...     col=mydf.id, mydf=mydf, x=tuple(range(4))).show()
|      +---+
|      | id|
|      +---+
|      |  0|
|      |  1|
|      |  2|
|      |  3|
|      +---+
|
|      >>> spark.sql('''
|      ...   SELECT m1.a, m2.b
|      ...   FROM {table1} m1 INNER JOIN {table2} m2
|      ...   ON m1.key = m2.key
|      ...   ORDER BY m1.a, m2.b''',
|      ...   table1=spark.createDataFrame([(1, "a"), (2, "b")], ["a",
"key"]),
|      ...   table2=spark.createDataFrame([(3, "a"), (4, "b"), (5,
"b")], ["b", "key"])).show()
|      +---+---+
|      |  a|  b|
|      +---+---+
|      |  1|  3|
|      |  2|  4|
|      |  2|  5|
|      +---+---+
|
|      Also, it is possible to query using class:`Column`
from :class:`DataFrame`.
|
|      >>> mydf = spark.createDataFrame([(1, 4), (2, 4), (3, 6)],
["A", "B"])
|      >>> spark.sql("SELECT {df.A}, {df[B]} FROM {df}",
df=mydf).show()
|      +---+---+
|      |  A|  B|
|      +---+---+
|      |  1|  4|
|      |  2|  4|
|      |  3|  6|
|      +---+---+
|
```

```
 |  stop(self) -> None
 |      Stop the underlying :class:`SparkContext`.
 |
 |      .. versionadded:: 2.0
 |
 |  table(self, tableName: str) -> pyspark.sql.dataframe.DataFrame
 |      Returns the specified table as a :class:`DataFrame`.
 |
 |      .. versionadded:: 2.0.0
 |
 |      Returns
 |      -------
 |      :class:`DataFrame`
 |
 |      Examples
 |      --------
 |      >>> df.createOrReplaceTempView("table1")
 |      >>> df2 = spark.table("table1")
 |      >>> sorted(df.collect()) == sorted(df2.collect())
 |      True
 |
 |
----------------------------------------------------------------------
 |  Class methods defined here:
 |
 |  getActiveSession() -> Optional[ForwardRef('SparkSession')] from
builtins.type
 |      Returns the active :class:`SparkSession` for the current
thread, returned by the builder
 |
 |      .. versionadded:: 3.0.0
 |
 |      Returns
 |      -------
 |      :class:`SparkSession`
 |          Spark session if an active session exists for the current
thread
 |
 |      Examples
 |      --------
 |      >>> s = SparkSession.getActiveSession()
 |      >>> l = [('Alice', 1)]
 |      >>> rdd = s.sparkContext.parallelize(l)
 |      >>> df = s.createDataFrame(rdd, ['name', 'age'])
 |      >>> df.select("age").collect()
 |      [Row(age=1)]
 |
 |
----------------------------------------------------------------------
 |  Readonly properties defined here:
```

```
 |
 |  catalog
 |      Interface through which the user may create, drop, alter or
query underlying
 |      databases, tables, functions, etc.
 |
 |      .. versionadded:: 2.0.0
 |
 |      Returns
 |      -------
 |      :class:`Catalog`
 |
 |  conf
 |      Runtime configuration interface for Spark.
 |
 |      This is the interface through which the user can get and set
all Spark and Hadoop
 |      configurations that are relevant to Spark SQL. When getting
the value of a config,
 |      this defaults to the value set in the
underlying :class:`SparkContext`, if any.
 |
 |      Returns
 |      -------
 |      :class:`pyspark.sql.conf.RuntimeConfig`
 |
 |      .. versionadded:: 2.0
 |
 |  read
 |      Returns a :class:`DataFrameReader` that can be used to read
data
 |      in as a :class:`DataFrame`.
 |
 |      .. versionadded:: 2.0.0
 |
 |      Returns
 |      -------
 |      :class:`DataFrameReader`
 |
 |  readStream
 |      Returns a :class:`DataStreamReader` that can be used to read
data streams
 |      as a streaming :class:`DataFrame`.
 |
 |      .. versionadded:: 2.0.0
 |
 |      Notes
 |      -----
 |      This API is evolving.
 |
```

```
 |      Returns
 |      -------
 |      :class:`DataStreamReader`
 |
 |  sparkContext
 |      Returns the underlying :class:`SparkContext`.
 |
 |      .. versionadded:: 2.0
 |
 |  streams
 |      Returns a :class:`StreamingQueryManager` that allows managing
all the
 |      :class:`StreamingQuery` instances active on `this` context.
 |
 |      .. versionadded:: 2.0.0
 |
 |      Notes
 |      -----
 |      This API is evolving.
 |
 |      Returns
 |      -------
 |      :class:`StreamingQueryManager`
 |
 |  udf
 |      Returns a :class:`UDFRegistration` for UDF registration.
 |
 |      .. versionadded:: 2.0.0
 |
 |      Returns
 |      -------
 |      :class:`UDFRegistration`
 |
 |  version
 |      The version of Spark on which this application is running.
 |
 |      .. versionadded:: 2.0
 |
 |
----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  Builder = <class 'pyspark.sql.session.SparkSession.Builder'>
 |      Builder for :class:`SparkSession`.
 |
 |
 |  __annotations__ = {'_activeSession':
typing.ClassVar[typing.Optional[F...
 |
 |  builder = <pyspark.sql.session.SparkSession.Builder object>
```

```
 |
 |
----------------------------------------------------------------------
 |  Data descriptors inherited from
pyspark.sql.pandas.conversion.SparkConversionMixin:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)


# Help can be used on any Python object
help(map)
help(testmti850.Test)

Help on class map in module builtins:

class map(object)
 |  map(func, *iterables) --> map object
 |
 |  Make an iterator that computes the function using arguments from
 |  each of the iterables.  Stops when the shortest iterable is
exhausted.
 |
 |  Methods defined here:
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __next__(self, /)
 |      Implement next(self).
 |
 |  __reduce__(...)
 |      Return state information for pickling.
 |
 |
----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate
signature.

Help on class Test in module testmti850:
```

```
class Test(builtins.object)
 |  Class methods defined here:
 |
 |  assertEquals(var, val, msg='') from builtins.type
 |
 |  assertEqualsHashed(var, hashed_val, msg='') from builtins.type
 |
 |  assertTrue(result, msg='') from builtins.type
 |
 |  printStats() from builtins.type
 |
 |  setFailFast() from builtins.type
 |
 |  setPrivateMode() from builtins.type
 |
 |
----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |
----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  failFast = False
 |
 |  numTests = 0
 |
 |  passed = 0
 |
 |  private = False
```

Part 2: Exploratory Data Analysis

Let's begin looking at our data.

For this assignment, we will use a data set from NASA Kennedy Space Center web server in Florida.

The full data set is freely available at ftp://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html, and it contains all HTTP requests for two months.

We are using a subset that only contains several days' worth of requests. To download the log and put it into HDFS, run the following commands in a terminal:

```
wget ftp://ita.ee.lbl.gov/traces/NASA_access_log_Aug95.gz
gunzip NASA_access_log_Aug95.gz
hdfs dfs -put NASA_access_log_Aug95 /NASA_access_log_Aug95.txt
```

(2a) Loading the log file

Now that we have the path to the file, let's load it into a DataFrame. We'll do this in steps. First, we'll use `spark.read.text()` to read the text file. This will produce a DataFrame with a single string column called `value`.

```
log_filename = "Nasa_access_log_Aug95.txt"
base_df = spark.read.text(log_filename)

# Let's look at the schema
base_df.printSchema()
```

```
root
 |-- value: string (nullable = true)
```

Let's take a look at some of the data.

```
base_df.show(truncate=False)
```

```
+------------------------------------------------------------------------
----------------------------------------------------------+
|value
|
+------------------------------------------------------------------------
----------------------------------------------------------+
|in24.inetnebr.com - - [01/Aug/1995:00:00:01 -0400] "GET
/shuttle/missions/sts-68/news/sts-68-mcc-05.txt HTTP/1.0" 200 1839
|
|uplherc.upl.com - - [01/Aug/1995:00:00:07 -0400] "GET / HTTP/1.0" 304
0                                                                      |
|uplherc.upl.com - - [01/Aug/1995:00:00:08 -0400] "GET
/images/ksclogo-medium.gif HTTP/1.0" 304 0
|
|uplherc.upl.com - - [01/Aug/1995:00:00:08 -0400] "GET /images/MOSAIC-
logosmall.gif HTTP/1.0" 304 0                                        |
|uplherc.upl.com - - [01/Aug/1995:00:00:08 -0400] "GET /images/USA-
logosmall.gif HTTP/1.0" 304 0                                          |
|ix-esc-ca2-07.ix.netcom.com - - [01/Aug/1995:00:00:09 -0400] "GET
/images/launch-logo.gif HTTP/1.0" 200 1713                           |
|uplherc.upl.com - - [01/Aug/1995:00:00:10 -0400] "GET /images/WORLD-
logosmall.gif HTTP/1.0" 304 0                                          |
|slppp6.intermind.net - - [01/Aug/1995:00:00:10 -0400] "GET
/history/skylab/skylab.html HTTP/1.0" 200 1687                               |
|piweba4y.prodigy.com - - [01/Aug/1995:00:00:10 -0400] "GET
/images/launchmedium.gif HTTP/1.0" 200 11853                                 |
|slppp6.intermind.net - - [01/Aug/1995:00:00:11 -0400] "GET
```

```
/history/skylab/skylab-small.gif HTTP/1.0" 200 9202                 |
|slppp6.intermind.net - - [01/Aug/1995:00:00:12 -0400] "GET
/images/ksclogosmall.gif HTTP/1.0" 200 3635                         |
|ix-esc-ca2-07.ix.netcom.com - - [01/Aug/1995:00:00:12 -0400] "GET
/history/apollo/images/apollo-logo1.gif HTTP/1.0" 200 1173   |
|slppp6.intermind.net - - [01/Aug/1995:00:00:13 -0400] "GET
/history/apollo/images/apollo-logo.gif HTTP/1.0" 200 3047           |
|uplherc.upl.com - - [01/Aug/1995:00:00:14 -0400] "GET /images/NASA-
logosmall.gif HTTP/1.0" 304 0                                      |
|133.43.96.45 - - [01/Aug/1995:00:00:16 -0400] "GET
/shuttle/missions/sts-69/mission-sts-69.html HTTP/1.0" 200 10566
|
|kgtyk4.kj.yamagata-u.ac.jp - - [01/Aug/1995:00:00:17 -0400] "GET /
HTTP/1.0" 200 7280                                                 |
|kgtyk4.kj.yamagata-u.ac.jp - - [01/Aug/1995:00:00:18 -0400] "GET
/images/ksclogo-medium.gif HTTP/1.0" 200 5866                   |
|d0ucr6.fnal.gov - - [01/Aug/1995:00:00:19 -0400] "GET
/history/apollo/apollo-16/apollo-16.html HTTP/1.0" 200 2743
|
|ix-esc-ca2-07.ix.netcom.com - - [01/Aug/1995:00:00:19 -0400] "GET
/shuttle/resources/orbiters/discovery.html HTTP/1.0" 200 6849|
|d0ucr6.fnal.gov - - [01/Aug/1995:00:00:20 -0400] "GET
/history/apollo/apollo-16/apollo-16-patch-small.gif HTTP/1.0" 200
14897  |
+-----------------------------------------------------------------------
-------------------------------------------------------+
only showing top 20 rows
```

(2b) Parsing the log file

If you're familiar with web servers at all, you'll recognize that this is in Common Log Format. The fields are:

remotehost rfc931 authuser [date] "request" status bytes

field	meaning
remotehost	Remote hostname (or IP number if DNS hostname is not available).
rfc931	The remote logname of the user. We don't really care about this field.
authuser	The username of the remote user, as authenticated by the HTTP server.
[date]	The date and time of the request.
"request"	The request, exactly as it came from the browser or client.
status	The HTTP status code the server sent back to the client.
bytes	The number of bytes (Content-Length) transferred to the client.

Next, we have to parse it into individual columns. We'll use the special built-in regexp_extract() function to do the parsing. This function matches a column against a

regular expression with one or more capture groups and allows you to extract one of the matched groups. We'll use one regular expression for each field we wish to extract.

If you can't read these regular expressions, don't worry. Trust us: They work. If you find regular expressions confusing (and they certainly *can* be), and you want to learn more about them, start with the RegexOne web site. You might also find *Regular Expressions Cookbook*, by Jan Goyvaerts and Steven Levithan, to be helpful.

Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems. (attributed to Jamie Zawinski)

```
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(truncate=False)
```

```
+----------------------------+-------------------------
+-----------------------------------------------------+------
+------------+
|host                        |timestamp                |path
|status|content_size|
+----------------------------+-------------------------
+-----------------------------------------------------+------
+------------+
|in24.inetnebr.com           |01/Aug/1995:00:00:01
-0400|/shuttle/missions/sts-68/news/sts-68-mcc-05.txt    |200   |1839
|
|uplherc.upl.com             |01/Aug/1995:00:00:07 -0400|/
|304   |0           |
|uplherc.upl.com             |01/Aug/1995:00:00:08
-0400|/images/ksclogo-medium.gif                          |304   |0
|
|uplherc.upl.com             |01/Aug/1995:00:00:08
-0400|/images/MOSAIC-logosmall.gif                        |304   |0
|
|uplherc.upl.com             |01/Aug/1995:00:00:08 -0400|/images/USA-
logosmall.gif                           |304   |0           |
|ix-esc-ca2-07.ix.netcom.com |01/Aug/1995:00:00:09
-0400|/images/launch-logo.gif                             |200   |1713
|
```

```
|uplherc.upl.com              |01/Aug/1995:00:00:10
-0400|/images/WORLD-logosmall.gif                          |304   |0
|
|slppp6.intermind.net         |01/Aug/1995:00:00:10
-0400|/history/skylab/skylab.html                          |200   |1687
|
|piweba4y.prodigy.com         |01/Aug/1995:00:00:10
-0400|/images/launchmedium.gif                             |200   |11853
|
|slppp6.intermind.net         |01/Aug/1995:00:00:11
-0400|/history/skylab/skylab-small.gif                     |200   |9202
|
|slppp6.intermind.net         |01/Aug/1995:00:00:12
-0400|/images/ksclogosmall.gif                             |200   |3635
|
|ix-esc-ca2-07.ix.netcom.com |01/Aug/1995:00:00:12
-0400|/history/apollo/images/apollo-logo1.gif              |200   |1173
|
|slppp6.intermind.net         |01/Aug/1995:00:00:13
-0400|/history/apollo/images/apollo-logo.gif               |200   |3047
|
|uplherc.upl.com              |01/Aug/1995:00:00:14 -0400|/images/NASA-
logosmall.gif                          |304   |0           |
|133.43.96.45                 |01/Aug/1995:00:00:16
-0400|/shuttle/missions/sts-69/mission-sts-69.html         |200   |10566
|
|kgtyk4.kj.yamagata-u.ac.jp  |01/Aug/1995:00:00:17 -0400|/
|200   |7280         |
|kgtyk4.kj.yamagata-u.ac.jp  |01/Aug/1995:00:00:18
-0400|/images/ksclogo-medium.gif                           |200   |5866
|
|d0ucr6.fnal.gov              |01/Aug/1995:00:00:19
-0400|/history/apollo/apollo-16/apollo-16.html             |200   |2743
|
|ix-esc-ca2-07.ix.netcom.com |01/Aug/1995:00:00:19
-0400|/shuttle/resources/orbiters/discovery.html           |200   |6849
|
|d0ucr6.fnal.gov              |01/Aug/1995:00:00:20
-0400|/history/apollo/apollo-16/apollo-16-patch-small.gif|200   |14897
|
+-----------------------------+-------------------------
+-------------------------------------------------+------
+------------+
only showing top 20 rows
```

(2c) Data Cleaning

Let's see how well our parsing logic worked. First, let's verify that there are no null rows in the original data set.

```
base_df.filter(base_df['value'].isNull()).count()
```

```
0
```

If our parsing worked properly, we'll have no rows with null column values. Let's check.

```
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.show()
bad_rows_df.count()
```

```
+--------------------+--------------------+--------------------+------
+------------+
|                host|           timestamp|                path|
status|content_size|
+--------------------+--------------------+--------------------+------
+------------+
|        gw1.att.com |01/Aug/1995:00:03...|/shuttle/missions...|
302|        null|
|js002.cc.utsunomi...|01/Aug/1995:00:07...|/shuttle/resource...|
404|        null|
|    tia1.eskimo.com |01/Aug/1995:00:28...|/pub/winvn/releas...|
404|        null|
|itws.info.eng.nii...|01/Aug/1995:00:38...|/ksc.html/facts/a...|
403|        null|
|grimnet23.idirect...|01/Aug/1995:00:50...|/www/software/win...|
404|        null|
|miriworld.its.uni...|01/Aug/1995:01:04...|/history/history.htm|
404|        null|
|      ras38.srv.net |01/Aug/1995:01:05...|/elv/DELTA/uncons...|
404|        null|
| cs1-06.leh.ptd.net |01/Aug/1995:01:17...|                    |
404|        null|
|www-b2.proxy.aol....|01/Aug/1995:01:22...|  /shuttle/countdown|
302|        null|
|    maui56.maui.net |01/Aug/1995:01:31...|            /shuttle|
302|        null|
|dialip-24.athenet...|01/Aug/1995:01:33...|/history/apollo/a...|
404|        null|
|  h96-158.ccnet.com |01/Aug/1995:01:35...|/history/apollo/a...|
404|        null|
|  h96-158.ccnet.com |01/Aug/1995:01:36...|/history/apollo/a...|
404|        null|
|  h96-158.ccnet.com |01/Aug/1995:01:36...|/history/apollo/a...|
404|        null|
|  h96-158.ccnet.com |01/Aug/1995:01:36...|/history/apollo/a...|
```

```
404|        null|
|  h96-158.ccnet.com |01/Aug/1995:01:36...|/history/apollo/a...|
404|        null|
|  h96-158.ccnet.com |01/Aug/1995:01:36...|/history/apollo/a...|
404|        null|
|  h96-158.ccnet.com |01/Aug/1995:01:36...|/history/apollo/a...|
404|        null|
|  h96-158.ccnet.com |01/Aug/1995:01:37...|/history/apollo/a...|
404|        null|
|  h96-158.ccnet.com |01/Aug/1995:01:37...|/history/apollo/a...|
404|        null|
+--------------------+--------------------+--------------------+------
+------------+
only showing top 20 rows


14178
```

Not good. We have some null values. Something went wrong. Which columns are affected?

(Note: This approach is adapted from an excellent answer on StackOverflow.)

```
from pyspark.sql.functions import col, sum

def count_null(col_name):
    return sum(col(col_name).isNull().cast('integer')).alias(col_name)

# Build up a list of column expressions, one per column.
#
# This could be done in one line with a Python list comprehension, but
we're keeping
# it simple for those who don't know Python very well.
exprs = []
for col_name in split_df.columns:
    exprs.append(count_null(col_name))

# Run the aggregation. The *exprs converts the list of expressions
into
# variable function arguments.
split_df.agg(*exprs).show() # aggregate on the entire DataFrame
(split_df) without groups using the column expressions
```

```
+----+---------+----+------+------------+
|host|timestamp|path|status|content_size|
+----+---------+----+------+------------+
|   0|        0|   0|     0|       14178|
+----+---------+----+------+------------+
```

Okay, they're all in the `content_size` column. Let's see if we can figure out what's wrong. Our original parsing regular expression for that column was:

```
regexp_extract('value', r'^.*\s+(\d+)$',
1).cast('integer').alias('content_size')
```

The \d+ selects one or more digits at the end of the input line. Is it possible there are lines without a valid content size? Or is there something wrong with our regular expression? Let's see if there are any lines that do not end with one or more digits.

Note: In the expression below, ~ means "not".

```
bad_content_size_df = base_df.filter(~ base_df['value'].rlike(r'\d+
$')) # base_df is the dataframe with the raw log (before split)

bad_content_size_df.count()
```

```
14178
```

That's it! The count matches the number of rows in `bad_rows_df` exactly.

Let's take a look at some of the bad column values. Since it's possible that the rows end in extra white space, we'll tack a marker character onto the end of each line, to make it easier to see trailing white space.

```
from pyspark.sql.functions import lit, concat

bad_content_size_df.select(concat(bad_content_size_df['value'],
lit('*'))).show(truncate=False)
```

```
+------------------------------------------------------------------------
-------------------------------------------------------+
|concat(value, *)
|
+------------------------------------------------------------------------
-------------------------------------------------------+
|gw1.att.com - - [01/Aug/1995:00:03:53 -0400] "GET
/shuttle/missions/sts-73/news HTTP/1.0" 302 -*
|
|js002.cc.utsunomiya-u.ac.jp - - [01/Aug/1995:00:07:33 -0400] "GET
/shuttle/resources/orbiters/discovery.gif HTTP/1.0" 404 -*|
|tia1.eskimo.com - - [01/Aug/1995:00:28:41 -0400] "GET
/pub/winvn/release.txt HTTP/1.0" 404 -*
|
|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 -*         |
|grimnet23.idirect.com - - [01/Aug/1995:00:50:12 -0400] "GET
/www/software/winvn/winvn.html HTTP/1.0" 404 -*                 |
|miriworld.its.unimelb.edu.au - - [01/Aug/1995:01:04:54 -0400] "GET
/history/history.htm HTTP/1.0" 404 -*                       |
|ras38.srv.net - - [01/Aug/1995:01:05:14 -0400] "GET
/elv/DELTA/uncons.htm HTTP/1.0" 404 -*
```

```
|
|cs1-06.leh.ptd.net - - [01/Aug/1995:01:17:38 -0400] "GET
/sts-71/launch/" 404 -*                                          |
|www-b2.proxy.aol.com - - [01/Aug/1995:01:22:07 -0400] "GET
/shuttle/countdown HTTP/1.0" 302 -*                               |
|maui56.maui.net - - [01/Aug/1995:01:31:56 -0400] "GET /shuttle
HTTP/1.0" 302 -*                                                  |
|dialip-24.athenet.net - - [01/Aug/1995:01:33:02 -0400] "GET
/history/apollo/apollo-13.html HTTP/1.0" 404 -*                   |
|h96-158.ccnet.com - - [01/Aug/1995:01:35:50 -0400] "GET
/history/apollo/a-001/a-001-patch-small.gif HTTP/1.0" 404 -*      |
|h96-158.ccnet.com - - [01/Aug/1995:01:36:23 -0400] "GET
/history/apollo/a-001/movies/ HTTP/1.0" 404 -*                    |
|h96-158.ccnet.com - - [01/Aug/1995:01:36:30 -0400] "GET
/history/apollo/a-001/a-001-patch-small.gif HTTP/1.0" 404 -*      |
|h96-158.ccnet.com - - [01/Aug/1995:01:36:38 -0400] "GET
/history/apollo/a-001/movies/ HTTP/1.0" 404 -*                    |
|h96-158.ccnet.com - - [01/Aug/1995:01:36:42 -0400] "GET
/history/apollo/a-001/a-001-patch-small.gif HTTP/1.0" 404 -*      |
|h96-158.ccnet.com - - [01/Aug/1995:01:36:44 -0400] "GET
/history/apollo/a-001/images/ HTTP/1.0" 404 -*                    |
|h96-158.ccnet.com - - [01/Aug/1995:01:36:47 -0400] "GET
/history/apollo/a-001/a-001-patch-small.gif HTTP/1.0" 404 -*      |
|h96-158.ccnet.com - - [01/Aug/1995:01:37:04 -0400] "GET
/history/apollo/a-004/a-004-patch-small.gif HTTP/1.0" 404 -*      |
|h96-158.ccnet.com - - [01/Aug/1995:01:37:05 -0400] "GET
/history/apollo/a-004/movies/ HTTP/1.0" 404 -*                    |
+-------------------------------------------------------------------
-----------------------------------------------------+
only showing top 20 rows
```

Ah. The bad rows correspond to error results, where no content was sent back and the server emitted a "`-`" for the `content_size` field. Since we don't want to discard those rows from our analysis, let's map them to 0.

(2d) Fix the rows with null content_size

The easiest solution is to replace the null values in `split_df` with 0. The DataFrame API provides a set of functions and fields specifically designed for working with null values, among them:

- fillna(), which fills null values with specified non-null values.
- na, which returns a DataFrameNaFunctions object with many functions for operating on null columns.

We'll use `fillna()`, because it's simple. There are several ways to invoke this function. The easiest is just to replace *all* null columns with known values. But, for safety, it's better to pass a Python dictionary containing (column_name, value) mappings. That's what we'll do.

```
# Replace all null content_size values with 0.

cleaned_df = split_df.fillna({'content_size': 0}) # in fact, fillna()
is an alias for na.fill()
```

Now, let us ensure that there are no nulls left.

```
exprs = []
for col_name in cleaned_df.columns:
  exprs.append(count_null(col_name))

cleaned_df.agg(*exprs).show()
```

```
+----+---------+----+------+------------+
|host|timestamp|path|status|content_size|
+----+---------+----+------+------------+
|   0|        0|   0|     0|           0|
+----+---------+----+------+------------+
```

(2e) Parsing the timestamp.

Okay, now that we have a clean, parsed DataFrame, we have to parse the timestamp field into an actual timestamp. The Common Log Format time is somewhat non-standard. A User-Defined Function (UDF) is the most straightforward way to parse it.

```
month_map = {
  'Jan': 1, 'Feb': 2, 'Mar':3, 'Apr':4, 'May':5, 'Jun':6, 'Jul':7,
  'Aug':8,  'Sep': 9, 'Oct':10, 'Nov': 11, 'Dec': 12
}

def parse_clf_time(s):
    """ Convert Common Log time format into a Python datetime object
    Args:
        s (str): date and time in Apache time format
[dd/mmm/yyyy:hh:mm:ss (+/-)zzzz]
    Returns:
        a string suitable for passing to CAST('timestamp')
    """
    # NOTE: We're ignoring time zone here. In a production
application, you'd want to handle that.
    return "{0:04d}-{1:02d}-{2:02d} {3:02d}:{4:02d}:{5:02d}".format(
      int(s[7:11]),
      month_map[s[3:6]],
      int(s[0:2]),
      int(s[12:14]),
      int(s[15:17]),
      int(s[18:20])
    )

u_parse_time = spark.udf.register('parse_clf_time', parse_clf_time) #
```

```
register the UDF

# sequence of operations: select all columns, add a new one (time),
and remove the timestamp column
logs_df = cleaned_df.select('*',
u_parse_time(cleaned_df['timestamp']).cast('timestamp').alias('time'))
.drop('timestamp')

total_log_entries = logs_df.count() # keep the total of log entries
for future operations

logs_df.show()

+--------------------+--------------------+------+------------
+------------------+
|                host|                path|status|content_size|
time|
+--------------------+--------------------+------+------------
+------------------+
|  in24.inetnebr.com |/shuttle/missions...|   200|        1839|1995-
08-01 00:00:01|
|    uplherc.upl.com |                   /|   304|           0|1995-
08-01 00:00:07|
|    uplherc.upl.com |/images/ksclogo-m...|   304|           0|1995-
08-01 00:00:08|
|    uplherc.upl.com |/images/MOSAIC-lo...|   304|           0|1995-
08-01 00:00:08|
|    uplherc.upl.com |/images/USA-logos...|   304|           0|1995-
08-01 00:00:08|
|ix-esc-ca2-07.ix....|/images/launch-lo...|   200|        1713|1995-
08-01 00:00:09|
|    uplherc.upl.com |/images/WORLD-log...|   304|           0|1995-
08-01 00:00:10|
|slppp6.intermind....|/history/skylab/s...|   200|        1687|1995-
08-01 00:00:10|
|piweba4y.prodigy....|/images/launchmed...|   200|       11853|1995-
08-01 00:00:10|
|slppp6.intermind....|/history/skylab/s...|   200|        9202|1995-
08-01 00:00:11|
|slppp6.intermind....|/images/ksclogosm...|   200|        3635|1995-
08-01 00:00:12|
|ix-esc-ca2-07.ix....|/history/apollo/i...|   200|        1173|1995-
08-01 00:00:12|
|slppp6.intermind....|/history/apollo/i...|   200|        3047|1995-
08-01 00:00:13|
|    uplherc.upl.com |/images/NASA-logo...|   304|           0|1995-
08-01 00:00:14|
|       133.43.96.45 |/shuttle/missions...|   200|       10566|1995-
08-01 00:00:16|
|kgtyk4.kj.yamagat...|                   /|   200|        7280|1995-
```

```
08-01 00:00:17|
|kgtyk4.kj.yamagat...|/images/ksclogo-m...|   200|        5866|1995-
08-01 00:00:18|
|    d0ucr6.fnal.gov |/history/apollo/a...|   200|        2743|1995-
08-01 00:00:19|
|ix-esc-ca2-07.ix....|/shuttle/resource...|   200|        6849|1995-
08-01 00:00:19|
|    d0ucr6.fnal.gov |/history/apollo/a...|   200|       14897|1995-
08-01 00:00:20|
+--------------------+--------------------+------+------------
+------------------+
only showing top 20 rows
```

```
logs_df.printSchema()
```

```
root
 |-- host: string (nullable = true)
 |-- path: string (nullable = true)
 |-- status: integer (nullable = true)
 |-- content_size: integer (nullable = false)
 |-- time: timestamp (nullable = true)
```

```
display(logs_df)
```

```
DataFrame[host: string, path: string, status: int, content_size: int,
time: timestamp]
```

Let's cache `logs_df`. We're going to be using it quite a bit from here forward.

```
logs_df.cache()
```

```
DataFrame[host: string, path: string, status: int, content_size: int,
time: timestamp]
```

Part 3: Analysis Walk-Through on the Web Server Log File

Now that we have a DataFrame containing the parsed log file as a set of Row objects, we can perform various analyses.

(3a) Example: Content Size Statistics

Let's compute some statistics about the sizes of content being returned by the web server. In particular, we'd like to know what are the average, minimum, and maximum content sizes.

We can compute the statistics by calling `.describe()` on the `content_size` column of `logs_df`. The `.describe()` function returns the count, mean, stddev, min, and max of a given column.

```
# Calculate statistics based on the content size.

content_size_summary_df = logs_df.describe(['content_size'])

content_size_summary_df.show()
```

```
+-------+------------------+
|summary|      content_size|
+-------+------------------+
|  count|           1569898|
|   mean|17089.225812122826|
| stddev|  67954.7639215694|
|    min|                 0|
|    max|           3421948|
+-------+------------------+
```

Alternatively, we can use SQL to directly calculate these statistics. You can explore the many useful functions within the `pyspark.sql.functions` module in the documentation.

After we apply the `.agg()` function, we call `.first()` to extract the first value, which is equivalent to `.take(1)[0]`.

```
import pyspark.sql.functions as sqlFunctions

content_size_stats =  (logs_df
                       .agg(sqlFunctions.min(logs_df['content_size']),
                            sqlFunctions.avg(logs_df['content_size']),
                            sqlFunctions.max(logs_df['content_size']))
                       .first())

print('Using SQL functions:')
print('Content Size Avg: {1:,.2f}; Min: {0:.2f}; Max:
{2:,.0f}'.format(*content_size_stats))
```

```
Using SQL functions:
Content Size Avg: 17,089.23; Min: 0.00; Max: 3,421,948
```

(3b) Example: HTTP Status Analysis

Next, let's look at the status values that appear in the log. We want to know which status values appear in the data and how many times. We again start with `logs_df`, then group by the `status` column, apply the `.count()` aggregation function, and sort by the `status` column.

```
status_to_count_df =(logs_df
                     .groupBy('status')
                     .count()
                     .sort('status')
                     .cache())
```

```
status_to_count_length = status_to_count_df.count()

print ('Found %d response codes' % status_to_count_length)

status_to_count_df.show()

assert status_to_count_length == 8
assert status_to_count_df.take(100) == [(200, 1398988), (302, 26497),
(304, 134146), (400, 10), (403, 171), (404, 10056), (500, 3), (501,
27)]
```

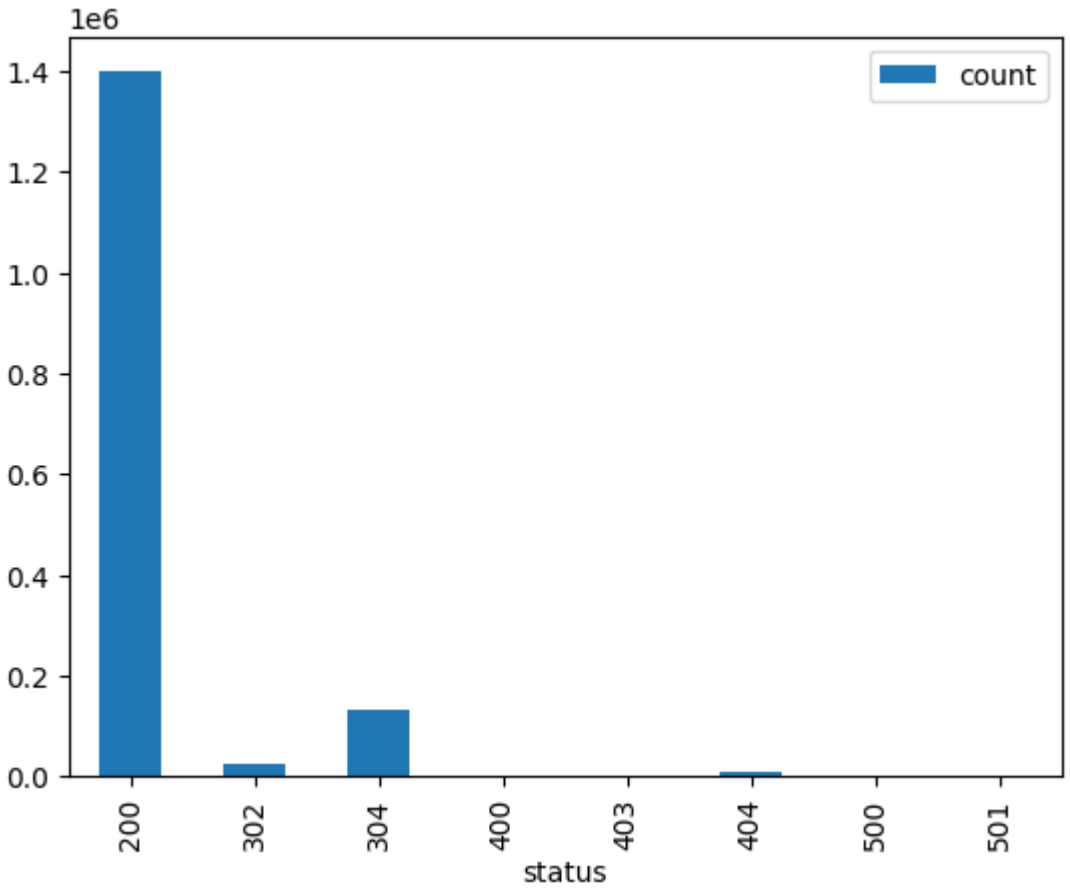
```
Found 8 response codes
+------+-------+
|status|  count|
+------+-------+
|   200|1398988|
|   302|  26497|
|   304| 134146|
|   400|     10|
|   403|    171|
|   404|  10056|
|   500|      3|
|   501|     27|
+------+-------+
```

(3c) Example: Status Graphing

Now, let's visualize the results from the last example. We can convert the original pyspark dataframe into a Pandas dataframe and then use the plot functionality. See the next cell for an example.

```
status_to_count_df_pd = status_to_count_df.toPandas()

status_to_count_df_pd.plot.bar(x='status', y='count')
```

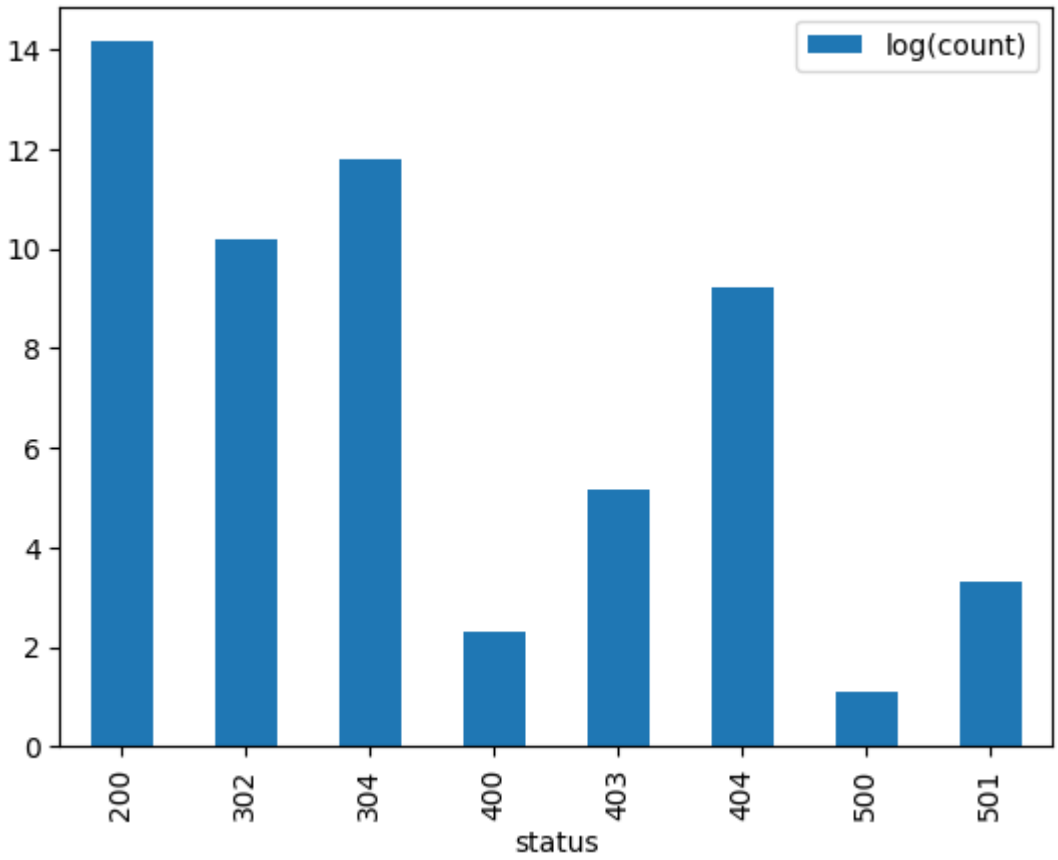
```
<AxesSubplot:xlabel='status'>
```



You can see that this is not a very effective plot. Due to the large number of '200' codes, it is very hard to see the relative number of the others. We can alleviate this by taking the logarithm of the count, adding that as a column to our DataFrame and displaying the result.

```
log_status_to_count_df = status_to_count_df.withColumn('log(count)',
sqlFunctions.log(status_to_count_df['count']))

log_status_to_count_df_pd = log_status_to_count_df.toPandas()

log_status_to_count_df_pd.plot.bar(x='status', y='log(count)')
```

```
<AxesSubplot:xlabel='status'>
```

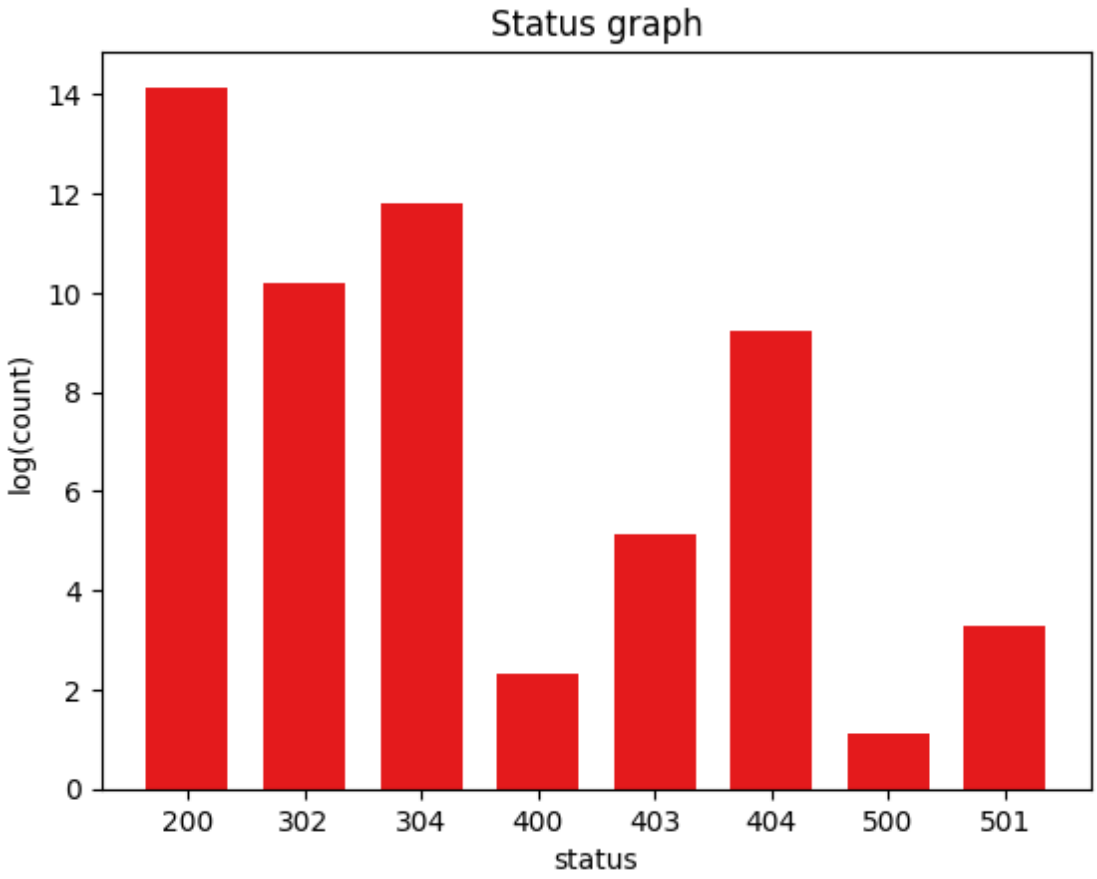


While this graph is an improvement, we might want to make more adjustments. The `matplotlib` library can give us more control in our plot. In this case, we're essentially just reproducing the Pandas graph using `matplotlib`. However, `matplotlib` exposes far more controls than the Pandas graph, allowing you to change colors, label the axes, and more.

We're using the "Set1" color map. See the list of Qualitative Color Maps at http://matplotlib.org/examples/color/colormaps_reference.html for more details. Feel free to change the color map to a different one, like "Accent".

```
import numpy as np
import matplotlib.pyplot as plt

data = log_status_to_count_df.drop('count').collect()
x, y = zip(*data) # split status (x) and count (y)
index = np.arange(len(x))
bar_width = 0.7
colorMap = 'Set1'
cmap = plt.cm.get_cmap(colorMap)

fig, ax = plt.subplots(nrows=1, ncols=1)
plt.bar(index, y, width=bar_width, color=cmap(0))
plt.xticks(ticks=index, labels=x) # set markers (ticks) and the
respective labels in the x-axis
```

```
plt.yticks(ticks=np.arange(0, 15, 2))
plt.xlabel('status')
plt.ylabel('log(count)')
plt.title('Status graph')
plt.show()
```



(3d) Example: Frequent Hosts

Let's look at hosts that have accessed the server frequently (e.g., more than ten times). As with the response code analysis in (3b), we create a new DataFrame by grouping `logs_df` by the 'host' column and aggregating by count.

We then filter the result based on the count of accesses by each host being greater than ten. Then, we select the 'host' column and show 20 elements from the result.

```
# Any hosts that has accessed the server more than 10 times.
host_sum_df =(logs_df
              .groupBy('host')
              .count())

host_more_than_10_df = (host_sum_df
                        .filter(host_sum_df['count'] > 10)
                        .select(host_sum_df['host'])) # select only
```

```
the host column

print ('Any 20 hosts that have accessed more then 10 times:\n')

host_more_than_10_df.show(truncate=False)
```

```
Any 20 hosts that have accessed more then 10 times:

+--------------------------------+
|host                            |
+--------------------------------+
|prakinf2.prakinf.tu-ilmenau.de |
|alpha2.csd.uwm.edu              |
|cjc07992.slip.digex.net         |
|n1377004.ksc.nasa.gov           |
|163.205.2.134                   |
|huge.oso.chalmers.se            |
|163.205.44.27                   |
|shark.ksc.nasa.gov              |
|etc5.etechcorp.com              |
|dd07-029.compuserve.com         |
|131.182.101.161                 |
|134.95.100.201                  |
|vab08.larc.nasa.gov             |
|ip11.iac.net                    |
|ad11-012.compuserve.com         |
|ad053.du.pipex.com              |
|204.184.6.19                    |
|p8.denver1.dialup.csn.net       |
|gate2.gdc.com                   |
|alcott.acsu.buffalo.edu         |
+--------------------------------+
only showing top 20 rows
```

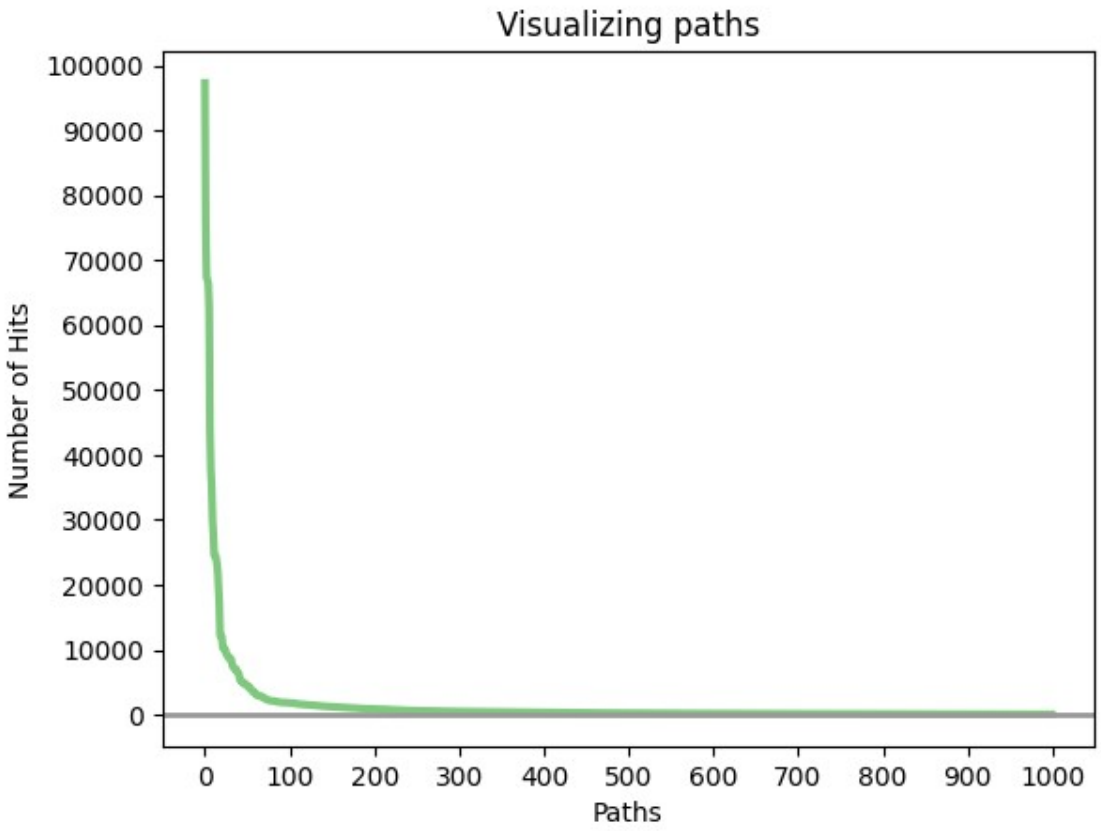
(3e) Example: Visualizing Paths

Now, let's visualize the number of hits to paths (URIs) in the log. To perform this task, we start with our `logs_df` and group by the `path` column, aggregate by count, and sort in descending order.

Next we visualize the results using `matplotlib`. We extract the paths and the counts, and unpack the resulting list of `Rows` using a `map` function and `lambda` expression.

```
paths_df = (logs_df
            .groupBy('path')
            .count()
            .sort('count', ascending=False)) # two-column dataframe

paths_counts = (paths_df
```

```
                .select('path', 'count') # in this case, it is similar
to '*'
                .rdd.map(lambda r: (r[0], r[1]))
                .collect())

paths, counts = zip(*paths_counts)

colorMap = 'Accent'
cmap = plt.cm.get_cmap(colorMap)

num_points= 1000
index = np.arange(num_points)
y = counts[:num_points]

fig, ax = plt.subplots(nrows=1, ncols=1)#prepareSubplot(np.arange(0,
6, 1), np.arange(0, 14, 2))
plt.plot(index, counts[:1000], color=cmap(0), linewidth=3)

plt.xticks(ticks=np.arange(0, 1001, 100)) # set markers (ticks) and
the respective labels in the x-axis
plt.yticks(ticks=np.arange(0, 100001, 10000))
plt.xlabel('Paths')
plt.ylabel('Number of Hits')
plt.title('Visualizing paths')

plt.axhline(linewidth=2, color='#999999') # horizontal gray line (y=0)
plt.show()
```



(3f) Example: Top Paths

For the final example, we'll find the top paths (URIs) in the log. Because we sorted `paths_df` for plotting, all we need to do is call `.show()` and pass in `n=10` and `truncate=False` as the parameters to show the top ten paths without truncating.

```
# Top Paths
print ('Top Ten Paths:')
paths_df.show(n=10, truncate=False)

expected = [
  (u'/images/NASA-logosmall.gif', 97275),
  (u'/images/KSC-logosmall.gif', 75283),
  (u'/images/MOSAIC-logosmall.gif', 67356),
  (u'/images/USA-logosmall.gif', 66975),
  (u'/images/WORLD-logosmall.gif', 66351),
  (u'/images/ksclogo-medium.gif', 62670),
  (u'/ksc.html', 43618),
  (u'/history/apollo/images/apollo-logo1.gif', 37806),
  (u'/images/launch-logo.gif', 35119),
  (u'/', 30105)
]
```

```
testmti850.Test.assertEquals(paths_df.take(10), expected, 'incorrect
Top Ten Paths')
```

```
Top Ten Paths:
+----------------------------------------+-----+
|path                                    |count|
+----------------------------------------+-----+
|/images/NASA-logosmall.gif              |97275|
|/images/KSC-logosmall.gif               |75283|
|/images/MOSAIC-logosmall.gif            |67356|
|/images/USA-logosmall.gif               |66975|
|/images/WORLD-logosmall.gif             |66351|
|/images/ksclogo-medium.gif              |62670|
|/ksc.html                               |43618|
|/history/apollo/images/apollo-logo1.gif|37806|
|/images/launch-logo.gif                 |35119|
|/                                       |30105|
+----------------------------------------+-----+
only showing top 10 rows

1 test passed.
```

Part 4: Analyzing Web Server Log File

Now it is your turn to perform analyses on the web server log files.

(4a) Exercise: Top Ten Error Paths

What are the top ten paths which did not have return code 200? Create a sorted list containing the paths and the number of times that they were accessed with a non-200 return code and show the top ten.

Think about the steps that you need to perform to determine which paths did not have a 200 return code, how you will uniquely count those paths and sort the list.

```
# TODO: Replace <FILL IN> with appropriate code

# You are welcome to structure your solution in a different way, so
long as
# you ensure the variables used in the next Test section are defined

# DataFrame containing all accesses that did not return a code 200
not200DF = logs_df.select(logs_df.path).filter(logs_df.status!=200)

# # Sorted DataFrame containing all paths and the number of times they
were accessed with non-200 return code
logs_sum_df = (not200DF.groupBy('path').count().sort('count',
ascending=False))
print('Top Ten failed URLs:')
logs_sum_df.show(n=10, truncate=False)
```

```
print(logs_sum_df.count())

Top Ten failed URLs:
+----------------------------------------+-----+
|path                                    |count|
+----------------------------------------+-----+
|/images/NASA-logosmall.gif              |19072|
|/images/KSC-logosmall.gif               |11328|
|/images/MOSAIC-logosmall.gif            |8617 |
|/images/USA-logosmall.gif               |8565 |
|/images/WORLD-logosmall.gif             |8360 |
|/images/ksclogo-medium.gif              |7722 |
|/history/apollo/images/apollo-logo1.gif |4355 |
|/shuttle/countdown/images/countclock.gif|4227 |
|/images/launch-logo.gif                 |4178 |
|/                                       |3605 |
+----------------------------------------+-----+
only showing top 10 rows

9991
```

```
# TEST Top ten error paths (4a)
top_10_err_urls = [(row[0], row[1]) for row in logs_sum_df.take(10)]
top_10_err_expected = [
  (u'/images/NASA-logosmall.gif', 19072),
  (u'/images/KSC-logosmall.gif', 11328),
  (u'/images/MOSAIC-logosmall.gif', 8617),
  (u'/images/USA-logosmall.gif', 8565),
  (u'/images/WORLD-logosmall.gif', 8360),
  (u'/images/ksclogo-medium.gif', 7722),
  (u'/history/apollo/images/apollo-logo1.gif', 4355),
  (u'/shuttle/countdown/images/countclock.gif', 4227),
  (u'/images/launch-logo.gif', 4178),
  (u'/', 3605)
]
testmti850.Test.assertEquals(logs_sum_df.count(), 9991, 'incorrect
count for logs_sum_df')
testmti850.Test.assertEquals(top_10_err_urls, top_10_err_expected,
'incorrect Top Ten failed URLs')
```

```
1 test passed.
1 test passed.
```

(4b) Exercise: Number of Unique Hosts

How many unique hosts are there in the entire log?

There are multiple ways to find this. Try to find a more optimal way than grouping by 'host'.

```
# TODO: Replace <FILL IN> with appropriate code
host_count=logs_df.groupBy('host').count()
host_count.show()

unique_host_count = host_count.count()
print ('Unique hosts: {0}'.format(unique_host_count))
```

```
+--------------------+-----+
|                host|count|
+--------------------+-----+
| grail911.nando.net |    8|
|prakinf2.prakinf....|  116|
| alpha2.csd.uwm.edu |   90|
|cjc07992.slip.dig...|   16|
|n1377004.ksc.nasa...|  323|
|      163.205.2.134 |  143|
|huge.oso.chalmers...|  251|
|    198.180.132.201 |    2|
|      163.205.44.27 |   60|
|    dial17.irco.com |    1|
| marple.harvard.edu |    2|
| shark.ksc.nasa.gov |   26|
|    xyplex16.uio.no |    6|
| etc5.etechcorp.com |   11|
|dd07-029.compuser...|   18|
|    131.182.101.161 |  103|
|ip-ts2-131.neca.com |    6|
|     134.95.100.201 |   15|
|        194.20.24.3 |    3|
|     143.166.206.88 |    5|
+--------------------+-----+
only showing top 20 rows

Unique hosts: 75060
```

```
# TEST Number of unique hosts (4b)
testmti850.Test.assertEquals(unique_host_count, 75060, 'incorrect
unique_host_count')
```

```
1 test passed.
```

(4c) Exercise: Number of Unique Daily Hosts

For an advanced exercise, let's determine the number of unique hosts in the entire log on a day-by-day basis. This computation will give us counts of the number of unique daily hosts. We'd like a DataFrame sorted by increasing day of the month which includes the day of the month and the associated number of unique hosts for that day. Make sure you cache the resulting DataFrame `daily_hosts_df` so that we can reuse it in the next exercise.

Think about the steps that you need to perform to count the number of different hosts that make requests *each* day. *Since the log only covers a single month, you can ignore the month.* You may want to use the `dayofmonth` function in the `pyspark.sql.functions` module.

Description of each variable

`day_to_host_pair_df`

A DataFrame with two columns

column
host
day

There will be one row in this DataFrame for each row in `logs_df`. Essentially, you're just trimming and transforming each row of `logs_df`. For example, for this row in `logs_df`:

```
gw1.att.com - - [23/Aug/1995:00:03:53 -0400] "GET
/shuttle/missions/sts-73/news HTTP/1.0" 302 -
```

your `day_to_host_pair_df` should have:

```
gw1.att.com 23
```

`day_group_hosts_df`

This DataFrame has the same columns as `day_to_host_pair_df`, but with duplicate (`day`, `host`) rows removed.

`daily_hosts_df`

A DataFrame with two columns:

column
day
count

```
# TODO: Replace <FILL IN> with appropriate code

from pyspark.sql.functions import dayofmonth

day_to_host_pair_df = logs_df.select("host", dayofmonth(col("time")))

day_group_hosts_df = day_to_host_pair_df.distinct()

daily_hosts_df =
day_group_hosts_df.groupBy("dayofmonth(time)").count().sort('dayofmont
h(time)', ascending=True).cache()

print ('Unique hosts per day:')
daily_hosts_df.show(n=30, truncate=False)
```

```
Unique hosts per day:
+----------------+-----+
|dayofmonth(time)|count|
+----------------+-----+
|1               |2582 |
|3               |3222 |
|4               |4191 |
|5               |2502 |
|6               |2538 |
|7               |4108 |
|8               |4406 |
|9               |4317 |
|10              |4523 |
|11              |4346 |
|12              |2865 |
|13              |2650 |
|14              |4454 |
|15              |4214 |
|16              |4340 |
|17              |4385 |
|18              |4168 |
|19              |2550 |
|20              |2560 |
|21              |4135 |
|22              |4456 |
|23              |4368 |
|24              |4077 |
|25              |4407 |
|26              |2644 |
|27              |2690 |
|28              |4215 |
|29              |4826 |
|30              |5266 |
|31              |5916 |
+----------------+-----+


# TEST Number of unique daily hosts (4c)
daily_hosts_list = (daily_hosts_df
                    .rdd.map(lambda r: (r[0], r[1]))
                    .take(30))
daily_hosts_list_expected = [(1, 2582), (3, 3222), (4, 4191), (5,
2502), (6, 2538), (7, 4108),
                             (8, 4406), (9, 4317), (10, 4523), (11,
4346), (12, 2865), (13, 2650),
                             (14, 4454), (15, 4214), (16, 4340), (17,
4385), (18, 4168), (19, 2550),
                             (20, 2560), (21, 4135), (22, 4456), (23,
4368), (24, 4077), (25, 4407),
                             (26, 2644), (27, 2690), (28, 4215), (29,
```

```
4826), (30, 5266), (31, 5916)]
testmti850.Test.assertEquals(day_to_host_pair_df.count(),
total_log_entries, 'incorrect row count for day_to_host_pair_df')
testmti850.Test.assertEquals(daily_hosts_df.count(), 30, 'incorrect
daily_hosts_df.count()')
testmti850.Test.assertEquals(daily_hosts_list,
daily_hosts_list_expected, 'incorrect daily_hosts_df')
testmti850.Test.assertTrue(daily_hosts_df.is_cached, 'incorrect
daily_hosts_df.is_cached')
```

```
1 test passed.
1 test passed.
1 test passed.
1 test passed.
```

(4d) Exercise: Visualizing the Number of Unique Daily Hosts

Using the results from the previous exercise, we will use `matplotlib` to plot a line graph of the unique hosts requests by day. We need a list of days called `days_with_hosts` and a list of the number of unique hosts for each corresponding day called `hosts`.

WARNING: Simply calling `collect()` on your transformed DataFrame won't work, because `collect()` returns a list of Spark SQL Row objects. You must *extract* the appropriate column values from the Row objects. Hint: A loop will help.

```
# TODO: Replace <FILL IN> with appropriate code
n=daily_hosts_df.count()
days_with_hosts = []

hosts = []

for i in range(0, n):
    hosts.append(daily_hosts_df.select("count").take(n)[i][0])

days_with_hosts.append(daily_hosts_df.select("dayofmonth(time)").take(
n)[i][0])

print(days_with_hosts)
print(hosts)
```

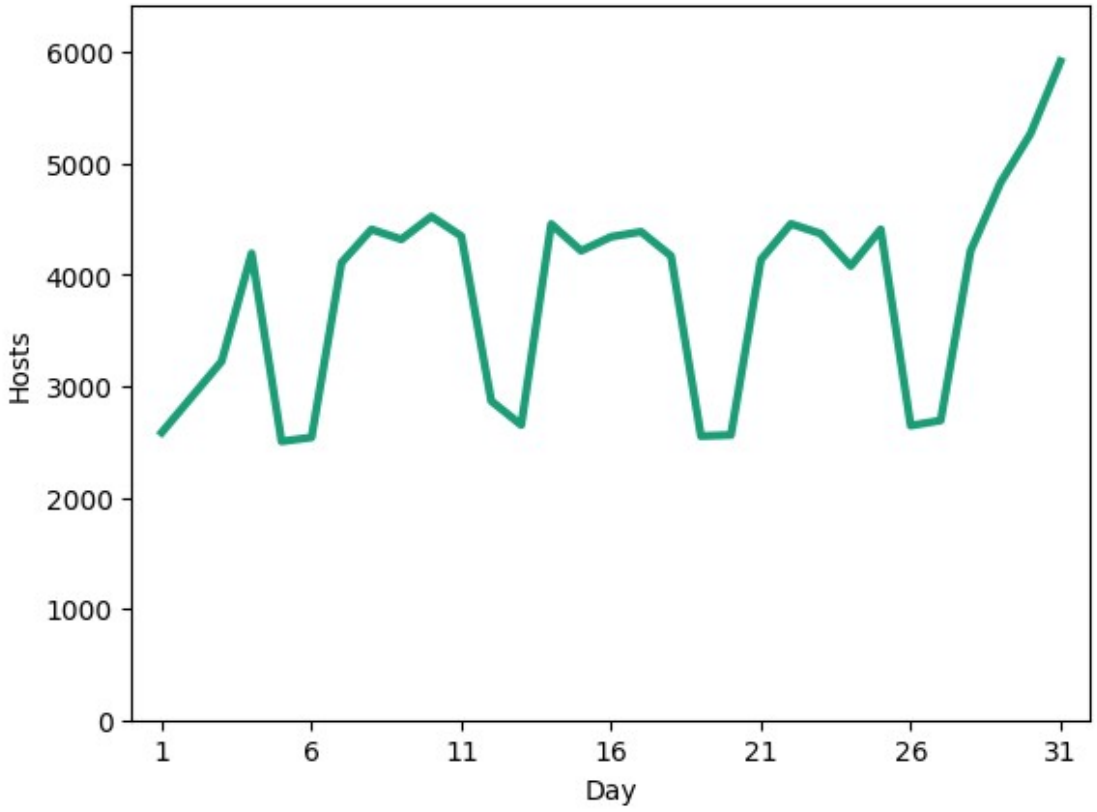
```
[1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]
[2582, 3222, 4191, 2502, 2538, 4108, 4406, 4317, 4523, 4346, 2865,
2650, 4454, 4214, 4340, 4385, 4168, 2550, 2560, 4135, 4456, 4368,
4077, 4407, 2644, 2690, 4215, 4826, 5266, 5916]
```

```
# TEST Visualizing unique daily hosts (4d)
days_with_hosts_expected = list(range(1, 32))
days_with_hosts_expected.remove(2)

hosts_expected= [2582, 3222, 4191, 2502, 2538, 4108, 4406, 4317, 4523,
```

```
4346, 2865, 2650, 4454, 4214, 4340, 4385, 4168, 2550, 2560, 4135,
4456, 4368, 4077, 4407, 2644, 2690, 4215, 4826, 5266, 5916]

testmti850.Test.assertEquals(days_with_hosts,
days_with_hosts_expected, 'incorrect days')
testmti850.Test.assertEquals(hosts, hosts_expected, 'incorrect hosts')
```

```
1 test passed.
1 test passed.
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)

plt.xticks(ticks=np.arange(1, max(days_with_hosts) + 5, 5))
plt.yticks(ticks=np.arange(0, max(hosts) + 1000, 1000))
colorMap = 'Dark2'
cmap = plt.cm.get_cmap(colorMap)
plt.plot(days_with_hosts, hosts, color=cmap(0), linewidth=3)
plt.axis([0, max(days_with_hosts) + 1, 0, max(hosts) + 500])
plt.xlabel('Day')
plt.ylabel('Hosts')
plt.show()
```



(4e) Exercise: Average Number of Daily Requests per Host

Next, let's determine the average number of requests on a day-by-day basis. We'd like a list by increasing day of the month and the associated average number of requests per host for that day. Make sure you cache the resulting DataFrame `avg_daily_req_per_host_df` so that we can reuse it in the next exercise.

To compute the average number of requests per host, find the total number of requests per day (across all hosts) and divide that by the number of unique hosts per day (which we found in part 4c and cached as `daily_hosts_df`).

Since the log only covers a single month, you can skip checking for the month.

```
# TODO: Replace <FILL IN> with appropriate code

total_req_per_day_df =
logs_df.groupBy(dayofmonth(col("time"))).count()
total_req_per_day_df =
total_req_per_day_df.withColumnRenamed("count","compteur")
total_req_per_day_df =
total_req_per_day_df.withColumnRenamed("dayofmonth(time)","dayofmonth"
)

daily_req_per_host_df =
total_req_per_day_df.join(daily_hosts_df,daily_hosts_df["dayofmonth(ti
me)"] ==  total_req_per_day_df["dayofmonth"],"inner")

avg_daily_req_per_host_df=daily_req_per_host_df.select(daily_req_per_h
ost_df["dayofmonth(time)"].alias("day"),
(daily_req_per_host_df["compteur"]/
daily_req_per_host_df["count"]).alias("avg_reqs_per_host_per_day")).so
rt('day', ascending=True).cache()

print ('Average number of daily requests per Hosts is:\n')

avg_daily_req_per_host_df.show()
```

```
Average number of daily requests per Hosts is:

+---+------------------------+
|day|avg_reqs_per_host_per_day|
+---+------------------------+
|  1|       13.166537567776917|
|  3|       12.845437616387336|
|  4|       14.210689572894298|
|  5|       12.747002398081534|
|  6|        12.77383766745469|
|  7|         13.9634858812074|
|  8|       13.653427144802542|
|  9|        14.00463284688441|
```

```
| 10|       13.541454786646032|
| 11|       14.092498849516797|
| 12|       13.288307155322862|
| 13|       13.766037735849057|
| 14|       13.443646160754378|
| 15|       13.964641670621736|
| 16|         13.0536866359447|
| 17|       13.452223489167617|
| 18|       13.494721689059501|
| 19|       12.585882352941177|
| 20|             12.876171875|
| 21|         13.4316807738815|
+---+-------------------------+
only showing top 20 rows
```

```
# TEST Average number of daily requests per hosts (4e)
from operator import itemgetter

avg_daily_req_per_host_list = (
  avg_daily_req_per_host_df.select('day',
avg_daily_req_per_host_df['avg_reqs_per_host_per_day'].cast('integer')
.alias('avg_requests'))
                           .collect()
)

values = [(row[0], row[1]) for row in avg_daily_req_per_host_list]
print(values)

values_expected = [(1, 13), (3, 12), (4, 14), (5, 12), (6, 12), (7,
13), (8, 13), (9, 14), (10, 13), (11, 14), (12, 13), (13, 13), (14,
13), (15, 13), (16, 13), (17, 13), (18, 13), (19, 12), (20, 12), (21,
13), (22, 12), (23, 13), (24, 12), (25, 13), (26, 11), (27, 12), (28,
13), (29, 14), (30, 15), (31, 15)]
testmti850.Test.assertEquals(values, values_expected, 'incorrect
avgDailyReqPerHostDF')
testmti850.Test.assertTrue(avg_daily_req_per_host_df.is_cached,
'incorrect avg_daily_req_per_host_df.is_cached')
```

```
[(1, 13), (3, 12), (4, 14), (5, 12), (6, 12), (7, 13), (8, 13), (9,
14), (10, 13), (11, 14), (12, 13), (13, 13), (14, 13), (15, 13), (16,
13), (17, 13), (18, 13), (19, 12), (20, 12), (21, 13), (22, 12), (23,
13), (24, 12), (25, 13), (26, 11), (27, 12), (28, 13), (29, 14), (30,
15), (31, 15)]
1 test passed.
1 test passed.
```

(4f) Exercise: Visualizing the Average Daily Requests per Unique Host

Using the result `avg_daily_req_per_host_df` from the previous exercise, use `matplotlib` to plot a line graph of the average daily requests per unique host by day.

`days_with_avg` should be a list of days and `avgs` should be a list of average daily requests (as integers) per unique hosts for each corresponding day. Hint: You will need to extract these from the Dataframe in a similar way to part 4d.

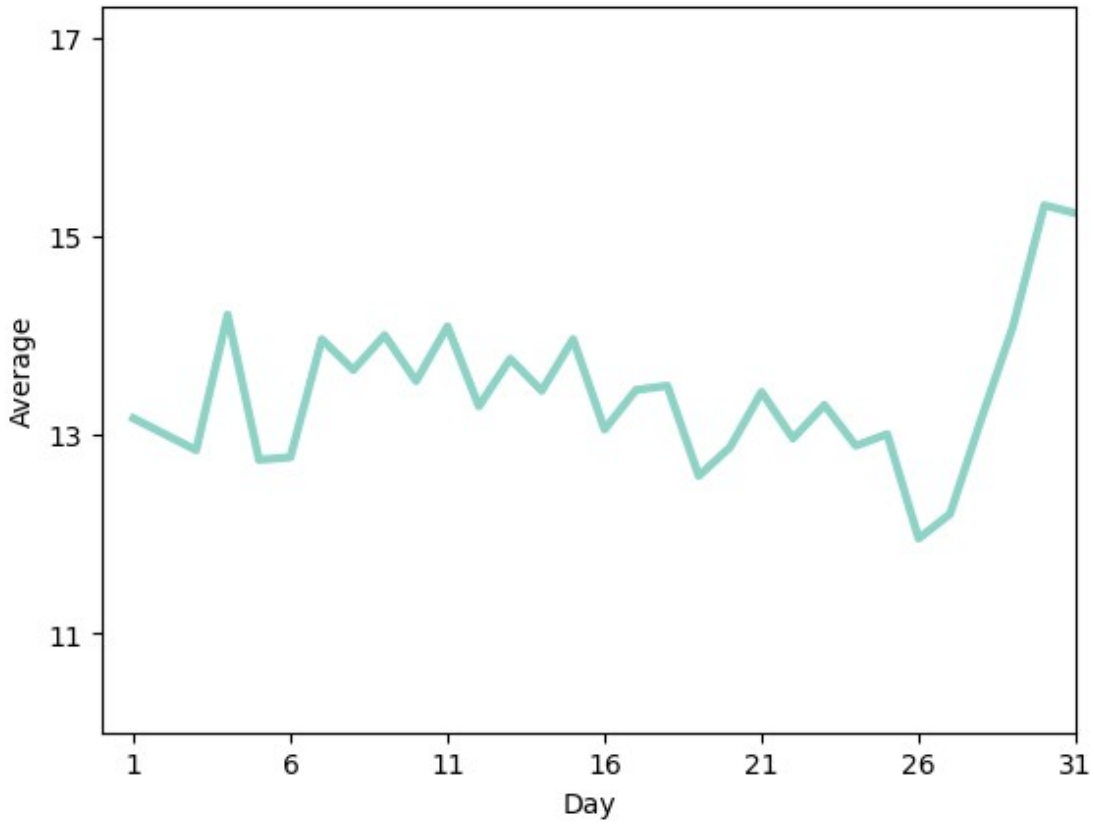
```
# TODO: Replace <FILL IN> with appropriate code
n=avg_daily_req_per_host_df.count()
days_with_avg = []

avgs = []

for i in range(0, n):
  days_with_avg.append(avg_daily_req_per_host_df.select("day").take(n)
[i][0])

avgs.append(avg_daily_req_per_host_df.select("avg_reqs_per_host_per_da
y").take(n)[i][0])

print(days_with_avg)

print(avgs)
```

```
[1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]
[13.166537567776917, 12.845437616387336, 14.210689572894298,
12.747002398081534, 12.77383766745469, 13.9634858812074,
13.653427144802542, 14.00463284688441, 13.541454786646032,
14.092498849516797, 13.288307155322862, 13.766037735849057,
13.443646160754378, 13.964641670621736, 13.0536866359447,
13.452223489167617, 13.494721689059501, 12.585882352941177,
12.876171875, 13.4316807738815, 12.962746858168762,
13.300595238095237, 12.889870002452783, 13.006807351940095,
11.954614220877458, 12.20185873605948, 13.166310794780546,
14.087857438872772, 15.313520698822636, 15.234110885733603]
```

```
# TEST Average Daily Requests per Unique Host (4f)
days_with_avg_expected = [1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]
avgs_expected = [13, 12, 14, 12, 12, 13, 13, 14, 13, 14, 13, 13, 13,
13, 13, 13, 12, 12, 13, 12, 13, 12, 13, 11, 12, 13, 14, 15, 15]

testmti850.Test.assertEquals(days_with_avg, days_with_avg_expected,
'incorrect days')
testmti850.Test.assertEquals([int(a) for a in avgs], avgs_expected,
'incorrect avgs')
```

```
1 test passed.
1 test passed.
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)

plt.xticks(ticks=np.arange(1, max(days_with_avg) + 5, 5))
plt.yticks(ticks=np.arange(int(min(avgs)), max(avgs) + 2, 2))
colorMap = 'Set3'
cmap = plt.cm.get_cmap(colorMap)
plt.plot(days_with_avg, avgs, color=cmap(0), linewidth=3)
plt.axis([0, max(days_with_avg), 10, max(avgs) + 2])
plt.xlabel('Day')
plt.ylabel('Average')
plt.show()
```



Part 5: Exploring 404 Status Codes

Let's drill down and explore the error 404 status records. We've all seen those "404 Not Found" web pages. 404 errors are returned when the server cannot find the resource (page or object) the browser or client requested.

(5a) Exercise: Counting 404 Response Codes

Create a DataFrame containing only log records with a 404 status code. Make sure you `cache()` `not_found_df` as we will use it in the rest of this exercise.

How many 404 records are in the log?

```
# TODO: Replace <FILL IN> with appropriate code

not_found_df = logs_df.filter(logs_df.status==404).cache()

print( ('Found {0} 404 URLs').format(not_found_df.count()) )
```

```
Found 10056 404 URLs
```

```
# TEST Counting 404 (5a)
testmti850.Test.assertEquals(not_found_df.count(), 10056, 'incorrect
not_found_df.count()')
testmti850.Test.assertTrue(not_found_df.is_cached, 'incorrect
not_found_df.is_cached')
```

```
1 test passed.
1 test passed.
```

(5b) Exercise: Listing 404 Status Code Records

Using the DataFrame containing only log records with a 404 status code that you cached in (5a), print out a list up to 40 *distinct* paths that generate 404 errors.

No path should appear more than once in your list.

```
# TODO: Replace <FILL IN> with appropriate code

not_found_paths_df = not_found_df.select(not_found_df.path)

unique_not_found_paths_df = not_found_paths_df.distinct()

print ('404 URLS:\n')

unique_not_found_paths_df.show(n=40, truncate=False)
```

```
404 URLS:

+-----------------------------------------------------------------+
|path                                                             |
+-----------------------------------------------------------------+
|/shuttle/missions/sts-68/images/images.html                      |
|/history/apollo/a-001/news/                                      |
|/history/apollo/a-003/movies/                                    |
|/CSMT_PageNS                                                     |
|/missions/shuttle                                                |
|/shuttle/technology/sts-newsref/stsret-newsref/stsref-toc.html|
|/pub/wiinvn/win3/ww16_99_.zip                                    |
|/public.win3/winvn                                               |
|/shuttle/sts-1/sts-1-pa.jpg                                      |
|/history/apollo/apollo/13                                        |
```

```
|/shuttle/missions/sts-61-a/images/                                  |
|/nssdc.gsfc.nasa.gov                                                |
|/shuttle/technology/images/sts-comm-small.gif                       |
|/shuttle/missions/sts-71/images/KSC-95EC-0916.txt                   |
|/shuttle/countdown/ac.html                                          |
|/pub/winvn/docs                                                     |
|/IMAGES/RSS.GIF                                                     |
|/history/apollo/-apollo-13/apollo-3.html                            |
|/shuttle/missions/sts-71/./video/livevideo.jpeg                     |
|/thunder                                                            |
|/pub/winvn/readme.txt                                               |
|/ksc.shtml                                                          |
|/img/sportstalk3.gif                                                |
|/home.html                                                          |
|/shuttle/missions/sts-61a/mission-sts-61a.html                      |
|/shuttle/technology/sts-newsref/srb.html%23srb                      |
|/astronaut.*                                                        |
|/history/apollo-12/apollo-12.html                                   |
|/elv/vidpicp.htlm                                                   |
|/history/apollo/sa-9/images/                                        |
|/elv/FACILITIES/elvhead2.gif                                        |
|/shuttle/missions/sts-86/mission-sts-86.html                        |
|/history/gemini/gemini-12.html                                      |
|/histoty/apollo/aplool-13/apollo-13.html                            |
|/hqpao/hqpao-home.html                                              |
|/winvn/winvn.html.                                                  |
|/www.quadralay.com                                                  |
|/shuttle/missions/missionshtml                                      |
|/history/apollo/-apollo-13/apollo13.html                            |
|/shuttle/miccions/sts-73/mission-sts-73.html                        |
+--------------------------------------------------------------------+
only showing top 40 rows
```

```
# TEST Listing 404 records (5b)

bad_unique_paths_40 = set([row[0] for row in
unique_not_found_paths_df.take(40)])
testmti850.Test.assertEquals(len(bad_unique_paths_40), 40,
'bad_unique_paths_40 not distinct')
```

```
1 test passed.
```

(5c) Exercise: Listing the Top Twenty 404 Response Code paths

Using the DataFrame containing only log records with a 404 response code that you cached in part (5a), print out a list of the top twenty paths that generate the most 404 errors.

Remember, top paths should be in sorted order

```
# TODO: Replace <FILL IN> with appropriate code

top_20_not_found_df =
not_found_paths_df.groupBy("path").count().sort(col("count").desc(),re
gexp_extract(col("path"), r'\/(\w+)\/', 1).asc())

print ('Top Twenty 404 URLs:\n')

top_20_not_found_df.show(n=20, truncate=False)
```

Top Twenty 404 URLs:

```
+-----------------------------------------------------------------
+-----+
|path                                                            |
count|
+-----------------------------------------------------------------
+-----+
|/pub/winvn/readme.txt                                           |
1337 |
|/pub/winvn/release.txt                                          |
1185 |
|/shuttle/missions/STS-69/mission-STS-69.html                    |682
|
|/images/nasa-logo.gif                                           |319
|
|/shuttle/missions/sts-68/ksc-upclose.gif                        |251
|
|/elv/DELTA/uncons.htm                                           |209
|
|/history/apollo/sa-1/sa-1-patch-small.gif                       |200
|
|/:://spacelink.msfc.nasa.gov                                    |166
|
|/images/crawlerway-logo.gif                                     |160
|
|/history/apollo/a-001/a-001-patch-small.gif                     |154
|
|/history/apollo/pad-abort-test-1/pad-abort-test-1-patch-small.gif|144
|
|                                                                |142
|
|/images/Nasa-logo.gif                                           |85
|
|/history/apollo/images/little-joe.jpg                           |84
|
|/shuttle/resources/orbiters/discovery.gif                       |82
|
|/shuttle/resources/orbiters/atlantis.gif                        |80
|
```

```
|/robots.txt                                                              |77
|
|/images/lf-logo.gif                                                      |77
|
|/shuttle/resources/orbiters/challenger.gif                               |77
|
|/pub                                                                     |57
|
+-------------------------------------------------------------------------
+-----+
only showing top 20 rows


# TEST Top twenty 404 URLs (5c)
top_20_not_found = [(row[0], row[1]) for row in
top_20_not_found_df.take(20)]
top_20_expected = [
 (u'/pub/winvn/readme.txt', 1337),
 (u'/pub/winvn/release.txt', 1185),
 (u'/shuttle/missions/STS-69/mission-STS-69.html', 682),
 (u'/images/nasa-logo.gif', 319),
 (u'/shuttle/missions/sts-68/ksc-upclose.gif', 251),
 (u'/elv/DELTA/uncons.htm', 209),
 (u'/history/apollo/sa-1/sa-1-patch-small.gif', 200),
 (u'/://spacelink.msfc.nasa.gov', 166),
 (u'/images/crawlerway-logo.gif', 160),
 (u'/history/apollo/a-001/a-001-patch-small.gif', 154),
 (u'/history/apollo/pad-abort-test-1/pad-abort-test-1-patch-
small.gif', 144),
 (u'', 142),
 (u'/images/Nasa-logo.gif', 85),
 (u'/history/apollo/images/little-joe.jpg', 84),
 (u'/shuttle/resources/orbiters/discovery.gif', 82),
 (u'/shuttle/resources/orbiters/atlantis.gif', 80),
 (u'/robots.txt', 77),
 (u'/images/lf-logo.gif', 77),
 (u'/shuttle/resources/orbiters/challenger.gif', 77),
 (u'/pub', 57)
]

print(top_20_not_found)
testmti850.Test.assertEquals(top_20_not_found, top_20_expected,
'incorrect top_20_not_found')

[('/pub/winvn/readme.txt', 1337), ('/pub/winvn/release.txt', 1185),
('/shuttle/missions/STS-69/mission-STS-69.html', 682), ('/images/nasa-
logo.gif', 319), ('/shuttle/missions/sts-68/ksc-upclose.gif', 251),
('/elv/DELTA/uncons.htm', 209), ('/history/apollo/sa-1/sa-1-patch-
small.gif', 200), ('/://spacelink.msfc.nasa.gov', 166),
('/images/crawlerway-logo.gif', 160), ('/history/apollo/a-001/a-001-
```

```
patch-small.gif', 154), ('/history/apollo/pad-abort-test-1/pad-abort-
test-1-patch-small.gif', 144), ('', 142), ('/images/Nasa-logo.gif',
85), ('/history/apollo/images/little-joe.jpg', 84),
('/shuttle/resources/orbiters/discovery.gif', 82),
('/shuttle/resources/orbiters/atlantis.gif', 80), ('/robots.txt', 77),
('/images/lf-logo.gif', 77),
('/shuttle/resources/orbiters/challenger.gif', 77), ('/pub', 57)]
1 test passed.
```

(5d) Exercise: Listing the Top Twenty-five 404 Response Code Hosts

Instead of looking at the paths that generated 404 errors, let's look at the hosts that encountered 404 errors. Using the DataFrame containing only log records with a 404 status codes that you cached in part (5a), print out a list of the top twenty-five hosts that generate the most 404 errors.

```
# TODO: Replace <FILL IN> with appropriate code

hosts_404_count_df =
not_found_df.groupBy("host").count().sort('count', ascending=False)

print ('Top 25 hosts that generated errors:\n')

hosts_404_count_df.show(n=25, truncate=False)
```

```
Top 25 hosts that generated errors:

+---------------------------+-----+
|host                       |count|
+---------------------------+-----+
|dialip-217.den.mmc.com     |62   |
|piweba3y.prodigy.com       |47   |
|155.148.25.4               |44   |
|maz3.maz.net               |39   |
|gate.barr.com              |38   |
|204.62.245.32              |37   |
|m38-370-9.mit.edu          |37   |
|nexus.mlckew.edu.au        |37   |
|ts8-1.westwood.ts.ucla.edu |37   |
|scooter.pa-x.dec.com       |35   |
|reddragon.ksc.nasa.gov     |33   |
|www-c4.proxy.aol.com       |32   |
|piweba5y.prodigy.com       |31   |
|www-d4.proxy.aol.com       |30   |
|piweba4y.prodigy.com       |30   |
|internet-gw.watson.ibm.com |29   |
|163.206.104.34             |28   |
|unidata.com                |28   |
|spica.sci.isas.ac.jp       |27   |
|www-d2.proxy.aol.com       |26   |
```

```
|203.13.168.17               |25   |
|203.13.168.24               |25   |
|www-d1.proxy.aol.com        |23   |
|www-c2.proxy.aol.com        |23   |
|crl5.crl.com                |23   |
+----------------------------+-----+
only showing top 25 rows


# TEST Top twenty-five 404 response code hosts (4d)

top_25_404 = [(row[0], row[1]) for row in hosts_404_count_df.take(25)]
top_25_404_expected = set([
    (u'dialip-217.den.mmc.com ', 62),
    (u'piweba3y.prodigy.com ', 47),
    (u'155.148.25.4 ', 44),
    (u'maz3.maz.net ', 39),
    (u'gate.barr.com ', 38),
    (u'204.62.245.32 ', 37),
    (u'nexus.mlckew.edu.au ', 37),
    (u'ts8-1.westwood.ts.ucla.edu ', 37),
    (u'm38-370-9.mit.edu ', 37),
    (u'scooter.pa-x.dec.com ', 35),
    (u'reddragon.ksc.nasa.gov ', 33),
    (u'www-c4.proxy.aol.com ', 32),
    (u'piweba5y.prodigy.com ', 31),
    (u'www-d4.proxy.aol.com ', 30),
    (u'piweba4y.prodigy.com ', 30),
    (u'internet-gw.watson.ibm.com ', 29),
    (u'unidata.com ', 28),
    (u'163.206.104.34 ', 28),
    (u'spica.sci.isas.ac.jp ', 27),
    (u'www-d2.proxy.aol.com ', 26),
    (u'203.13.168.17 ', 25),
    (u'203.13.168.24 ', 25),
    (u'www-c2.proxy.aol.com ', 23),
    (u'www-d1.proxy.aol.com ', 23),
    (u'crl5.crl.com ', 23)
])
testmti850.Test.assertEquals(len(top_25_404), 25, 'length of
errHostsTop25 is not 25')
testmti850.Test.assertEquals(len(set(top_25_404) -
top_25_404_expected), 0, 'incorrect hosts_404_count_df')

1 test passed.
1 test passed.
```

(5e) Exercise: Listing 404 Errors per Day

Let's explore the 404 records temporally. Break down the 404 requests by day (cache the `errors_by_date_sorted_df` DataFrame) and get the daily counts sorted by day in `errors_by_date_sorted_df`.

Since the log only covers a single month, you can ignore the month in your checks.

```
# TODO: Replace <FILL IN> with appropriate code

errors_by_date_sorted_df =
not_found_df.groupBy(dayofmonth(col("time"))).count().sort('dayofmonth
(time)', ascending=True).cache()

print ('404 Errors by day:\n')

errors_by_date_sorted_df.show()
```

```
404 Errors by day:

+----------------+-----+
|dayofmonth(time)|count|
+----------------+-----+
|               1|  243|
|               3|  304|
|               4|  346|
|               5|  236|
|               6|  373|
|               7|  537|
|               8|  391|
|               9|  279|
|              10|  315|
|              11|  263|
|              12|  196|
|              13|  216|
|              14|  287|
|              15|  327|
|              16|  259|
|              17|  271|
|              18|  256|
|              19|  209|
|              20|  312|
|              21|  305|
+----------------+-----+
only showing top 20 rows
```

```
# TEST 404 response codes per day (5e)

errors_by_date = [(row[0], row[1]) for row in
```

```
errors_by_date_sorted_df.collect()]
errors_by_date_expected = [
    (1, 243),
    (3, 304),
    (4, 346),
    (5, 236),
    (6, 373),
    (7, 537),
    (8, 391),
    (9, 279),
    (10, 315),
    (11, 263),
    (12, 196),
    (13, 216),
    (14, 287),
    (15, 327),
    (16, 259),
    (17, 271),
    (18, 256),
    (19, 209),
    (20, 312),
    (21, 305),
    (22, 288),
    (23, 345),
    (24, 420),
    (25, 415),
    (26, 366),
    (27, 370),
    (28, 410),
    (29, 420),
    (30, 571),
    (31, 526)
]
testmti850.Test.assertEquals(errors_by_date, errors_by_date_expected,
'incorrect errors_by_date_sorted_df')
testmti850.Test.assertTrue(errors_by_date_sorted_df.is_cached,
'incorrect errors_by_date_sorted_df.is_cached')

1 test passed.
1 test passed.
```

(5f) Exercise: Visualizing the 404 Errors by Day

Using the results from the previous exercise, use `matplotlib` to plot a line or bar graph of the 404 response codes by day.

Hint: You'll need to use the same technique you used in (4f).

```
# TODO: Replace <FILL IN> with appropriate code
n=errors_by_date_sorted_df.count()
```

```
days_with_errors_404 = []

errors_404_by_day = []


for i in range(0, n):

days_with_errors_404.append(errors_by_date_sorted_df.select("dayofmont
h(time)").take(n)[i][0])

errors_404_by_day.append(errors_by_date_sorted_df.select("count").take
(n)[i][0])

print (days_with_errors_404)

print (errors_404_by_day)
```

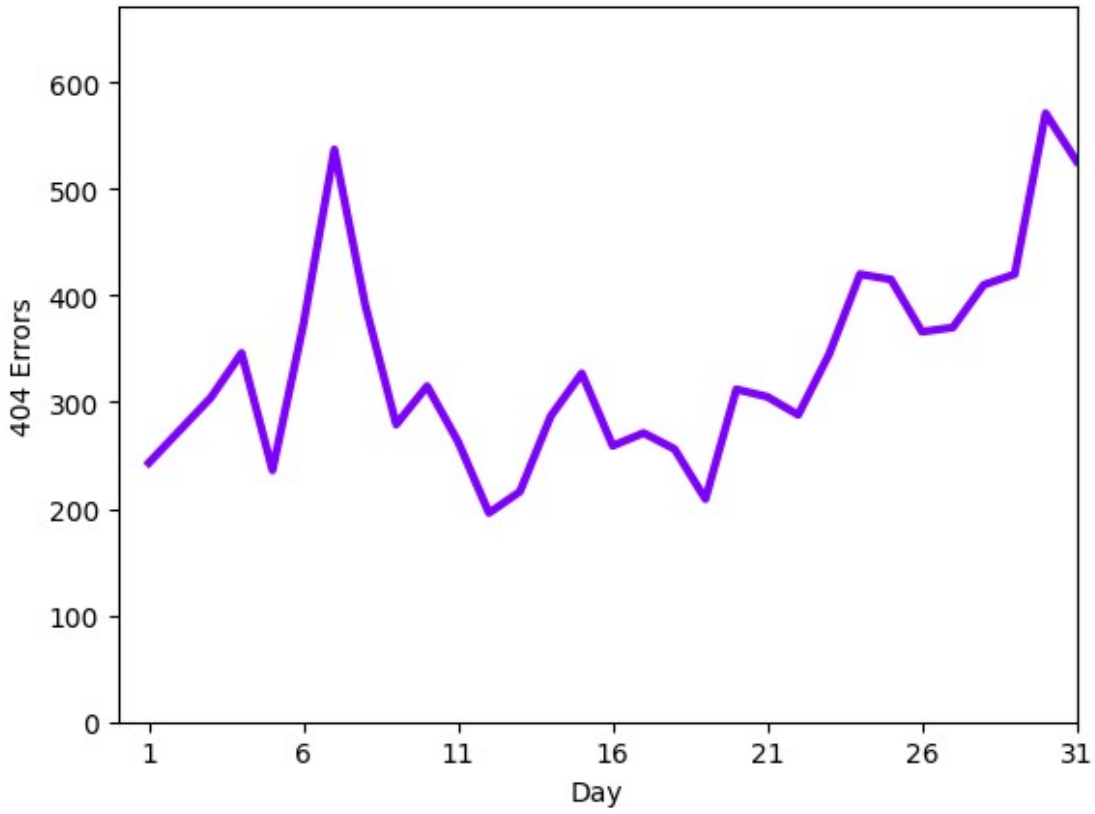
```
[1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]
[243, 304, 346, 236, 373, 537, 391, 279, 315, 263, 196, 216, 287, 327,
259, 271, 256, 209, 312, 305, 288, 345, 420, 415, 366, 370, 410, 420,
571, 526]
```

```
# TEST Visualizing the 404 Response Codes by Day (4f)
days_with_errors_404_expected = [1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
30, 31]
errors_404_by_day_expected = [243, 304, 346, 236, 373, 537, 391, 279,
315, 263, 196, 216, 287, 327, 259, 271, 256, 209, 312, 305, 288, 345,
420, 415, 366, 370, 410, 420, 571, 526]
testmti850.Test.assertEquals(days_with_errors_404,
days_with_errors_404_expected, 'incorrect days_with_errors_404')
testmti850.Test.assertEquals(errors_404_by_day,
errors_404_by_day_expected, 'incorrect errors_404_by_day')
```

```
1 test passed.
1 test passed.
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
plt.xticks(ticks=np.arange(1, max(days_with_errors_404) + 5, 5))
plt.yticks(ticks=np.arange(0, max(errors_404_by_day) + 100, 100))
colorMap = 'rainbow'
cmap = plt.cm.get_cmap(colorMap)
plt.plot(days_with_errors_404, errors_404_by_day, color=cmap(0),
linewidth=3)
plt.axis([0, max(days_with_errors_404), 0, max(errors_404_by_day) +
100])
plt.xlabel('Day')
```

```
plt.ylabel('404 Errors')
plt.show()
```



(5g) Exercise: Top Five Days for 404 Errors

Using the DataFrame `errors_by_date_sorted_df` you cached in the part (5e), what are the top five days for 404 errors and the corresponding counts of 404 errors?

```
# TODO: Replace <FILL IN> with appropriate code

top_err_date_df = errors_by_date_sorted_df.sort("count",
ascending=False)

print ('Top Five Dates for 404 Requests:\n')

top_err_date_df.show(5)
```

```
Top Five Dates for 404 Requests:

+----------------+-----+
|dayofmonth(time)|count|
+----------------+-----+
|              30|  571|
|               7|  537|
|              31|  526|
```

```
|              24|  420|
|              29|  420|
+----------------+-----+
only showing top 5 rows
```

```
# TEST Five dates for 404 requests (4g)
top_err_date_list = [(col1, col2) for col1, col2 in
top_err_date_df.take(5)]
top_err_date_list_expected = [(30, 571), (7, 537), (31, 526), (29,
420), (24, 420)]
testmti850.Test.assertEquals(top_err_date_list,
top_err_date_list_expected, 'incorrect top_err_date_df')
```

```
1 test passed.
```

(5h) Exercise: Hourly 404 Errors

Using the DataFrame `not_found_df` you cached in the part (5a) and sorting by hour of the day in increasing order, create a DataFrame containing the number of requests that had a 404 return code for each hour of the day (midnight starts at 0). Cache the resulting DataFrame `hour_records_sorted_df` and print that as a list.

```
# TODO: Replace <FILL IN> with appropriate code

from pyspark.sql.functions import hour

hour_records_sorted_df =
not_found_df.groupBy(hour(col("time"))).count().sort('hour(time)',
ascending=True).cache()

print ('Top hours for 404 requests:\n')

hour_records_sorted_df.show(24)
```

```
Top hours for 404 requests:

+----------+-----+
|hour(time)|count|
+----------+-----+
|         0|  344|
|         1|  327|
|         2|  600|
|         3|  363|
|         4|  183|
|         5|  160|
|         6|  135|
|         7|  218|
|         8|  340|
|         9|  359|
```

```
|         10|  492|
|         11|  428|
|         12|  651|
|         13|  614|
|         14|  522|
|         15|  549|
|         16|  550|
|         17|  586|
|         18|  425|
|         19|  440|
|         20|  432|
|         21|  434|
|         22|  430|
|         23|  474|
+-----------+-----+
```

```
# TEST Hourly 404 response codes (5h)

errs_by_hour = [(col1, col2) for col1, col2 in
hour_records_sorted_df.collect()]
for x in errs_by_hour:
    print(x)

errs_by_hour_expected = [
    (0, 344),
    (1, 327),
    (2, 600),
    (3, 363),
    (4, 183),
    (5, 160),
    (6, 135),
    (7, 218),
    (8, 340),
    (9, 359),
    (10, 492),
    (11, 428),
    (12, 651),
    (13, 614),
    (14, 522),
    (15, 549),
    (16, 550),
    (17, 586),
    (18, 425),
    (19, 440),
    (20, 432),
    (21, 434),
    (22, 430),
    (23, 474),
]
```

```
testmti850.Test.assertEquals(errs_by_hour, errs_by_hour_expected,
'incorrect errs_by_hour')
testmti850.Test.assertTrue(hour_records_sorted_df.is_cached,
'incorrect hour_records_sorted_df.is_cached')
```

```
(0, 344)
(1, 327)
(2, 600)
(3, 363)
(4, 183)
(5, 160)
(6, 135)
(7, 218)
(8, 340)
(9, 359)
(10, 492)
(11, 428)
(12, 651)
(13, 614)
(14, 522)
(15, 549)
(16, 550)
(17, 586)
(18, 425)
(19, 440)
(20, 432)
(21, 434)
(22, 430)
(23, 474)
1 test passed.
1 test passed.
```

(5i) Exercise: Visualizing the 404 Response Codes by Hour

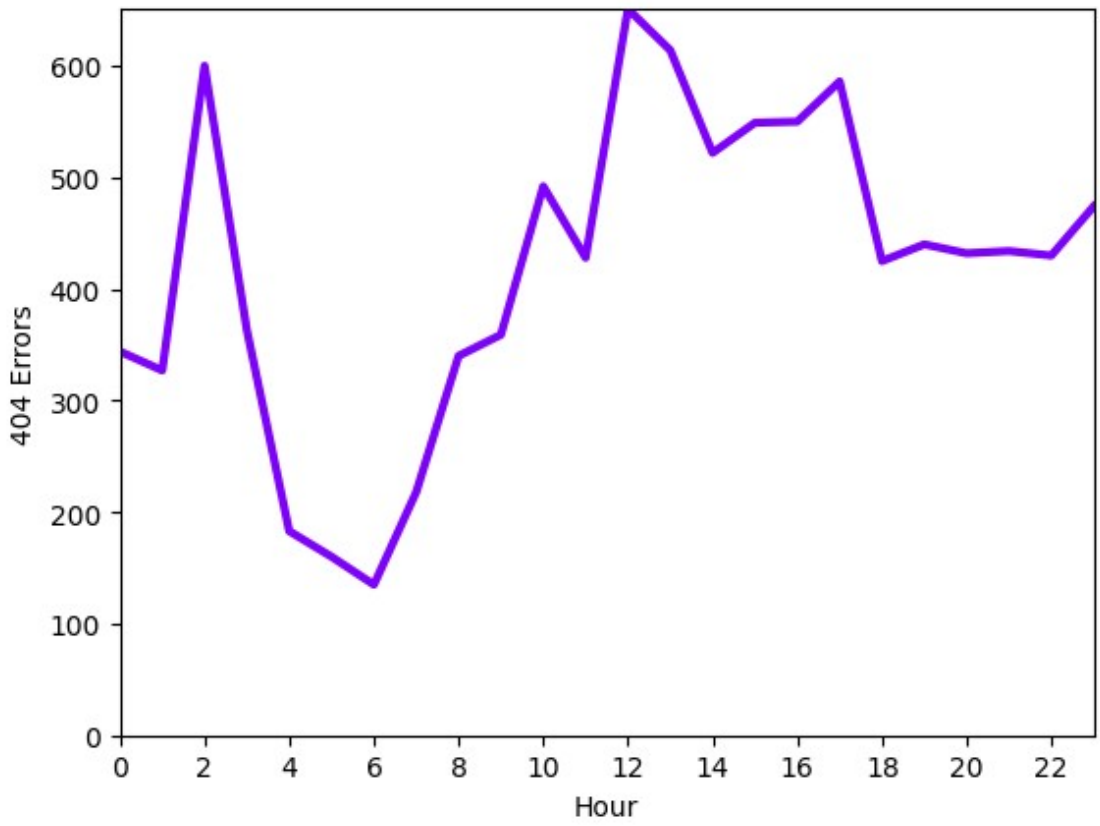
Using the results from the previous exercise, use `matplotlib` to plot a line or bar graph of the 404 response codes by hour.

```
# TODO: Replace <FILL IN> with appropriate code
n=hour_records_sorted_df.count()
hours_with_not_found = []

not_found_counts_per_hour = []

for i in range(0, n):

hours_with_not_found.append(hour_records_sorted_df.select("hour(time)"
).take(n)[i][0])

not_found_counts_per_hour.append(hour_records_sorted_df.select("count"
).take(n)[i][0])
```

```
print(hours_with_not_found)

print(not_found_counts_per_hour)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22, 23]
[344, 327, 600, 363, 183, 160, 135, 218, 340, 359, 492, 428, 651, 614,
522, 549, 550, 586, 425, 440, 432, 434, 430, 474]
```

```
# TEST Visualizing the 404 Response Codes by Hour (5i)
hours_with_not_found_expected = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]
not_found_counts_per_hour_expected = [344, 327, 600, 363, 183, 160,
135, 218, 340, 359, 492, 428, 651, 614, 522, 549, 550, 586, 425, 440,
432, 434, 430, 474]
testmti850.Test.assertEquals(hours_with_not_found,
hours_with_not_found_expected, 'incorrect hours_with_not_found')
testmti850.Test.assertEquals(not_found_counts_per_hour,
not_found_counts_per_hour_expected, 'incorrect
not_found_counts_per_hour')

fig, ax = plt.subplots(nrows=1, ncols=1)
plt.xticks(ticks=np.arange(0, 24, 2))
plt.yticks(ticks=np.arange(0, max(not_found_counts_per_hour) + 100,
100))
colorMap = 'seismic'
plt.plot(hours_with_not_found, not_found_counts_per_hour,
color=cmap(0), linewidth=3)
plt.axis([0, max(hours_with_not_found), 0,
max(not_found_counts_per_hour)])
plt.xlabel('Hour')
plt.ylabel('404 Errors')
plt.show()
```



Notebook Finished

Created in Deepnote