TANISHA CONSULTING

Big Data Wrangling

PREPARED BY
TANISHA BATRA

PRESENTED TO
BRAINSTATION

Big Data Wrangling with Google Books Ngrams

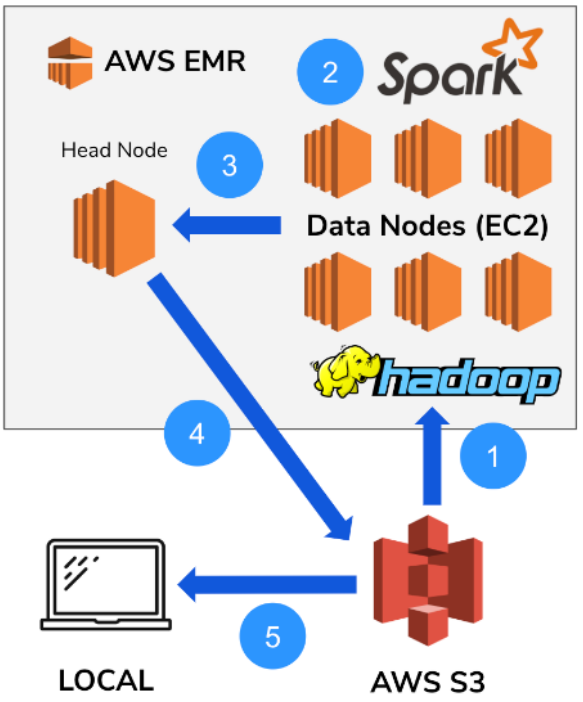
26th June 2022

OVERVIEW

In this assignment, I will apply the skills you've learned in the Big Data Fundamentals unit to load, filter, and visualize a large real-world dataset in a cloud-based distributed computing environment using Hadoop, Spark, Hive, and the S3 filesystem.

The Google Ngrams dataset was created by Google's research team by analyzing all of the content in Google Books - these digitized texts represent approximately 4% of all books ever printed, and span a time period from the 1800s into the 2000s.

As part of this workflow I will filter and reduce data down to a manageable size, and then do some analysis locally on our machine after extracting data from the Cloud and processing it using Big Data tools. The workflow and steps in the process are illustrated below:



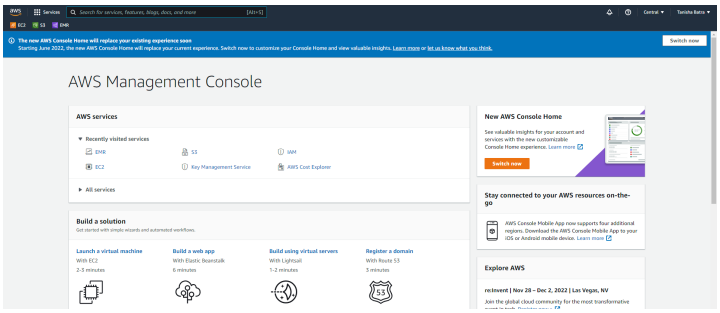
Big Data Analysis Workflow

1. Pull data from (public) S3 bucket into HDFS
2. Analyze and filter data using Spark
3. Collect filtered data to head node using *getmerge*
4. Write data to (personal) S3 bucket
5. Read data from S3 bucket for analysis on local machine

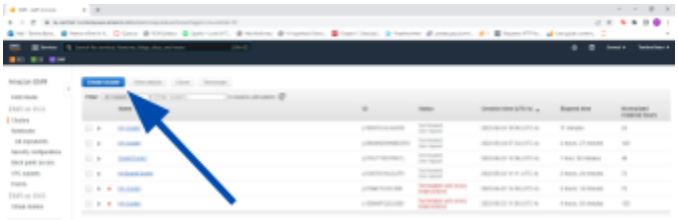
*the above overview is directly from Brainstation instructions

1. Create a Cluster

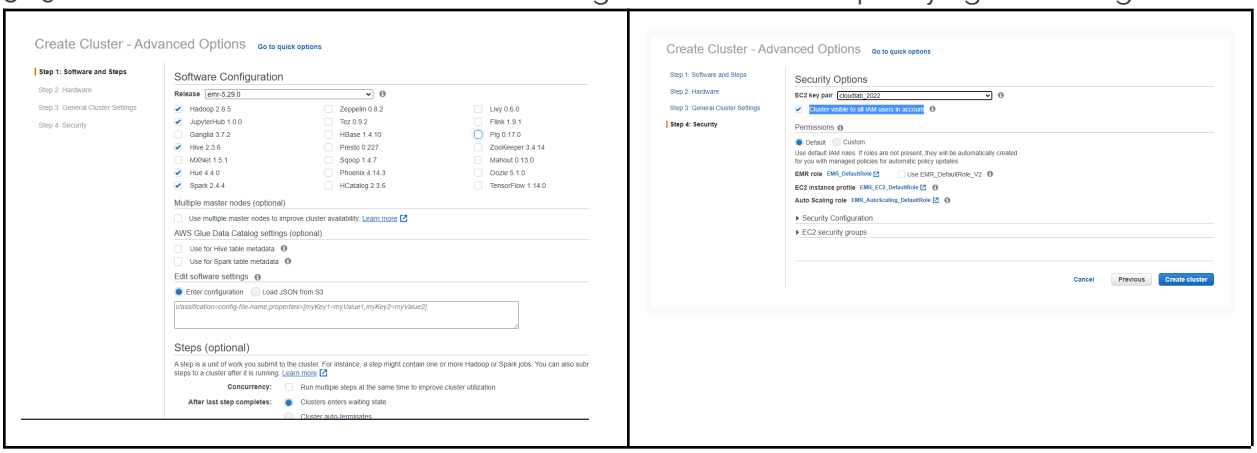
To begin, I went on https://aws.amazon.com/ and signed myself in as a root user. The below is what my console looks like:



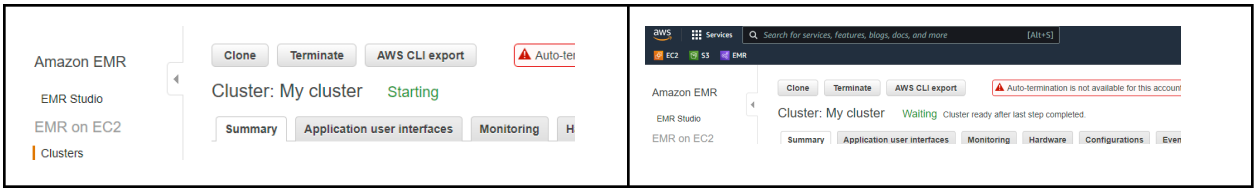
I went on the EMR services to create a new cluster, and navigated to the advanced options to choose appropriate settings for the cluster.



I made sure to include Hadoop, Space, Hive and Jupyterhub. The version I used was EMR 5.29.0 . Below are screenshots of me creating the clusters and specifying the settings.



Finally, I waited for the cluster to get into a ready state to use.



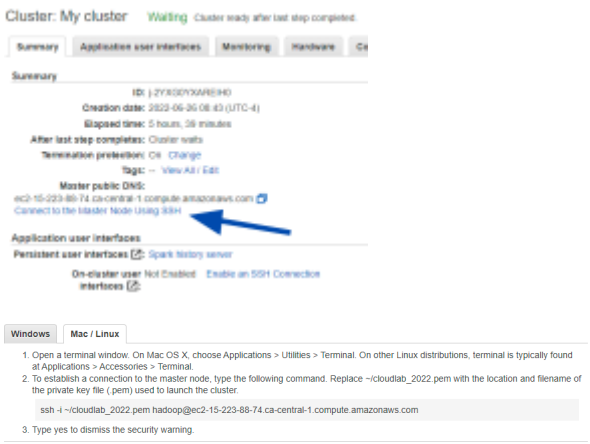
2. Connect to the head node of the cluster using SSH

Our cluster has the Hadoop framework in it. The Hadoop framework has something called a head node, which is a resource manager. It has information on the processes that run in the cluster, and knows where many files are. It's a great place to navigate from.

SSH is a network protocol. We can think of this as a tunnel to connect our computer to our cluster, so they can communicate with each other.

Finding the Command

First, I copied the command so that my system could connect to the cluster



Then, I went to the directory which had the private key file (.pem) that I could use to launch the cluster. I had saved this to my desktop.

Establishing the Connection

```
jsmba@DESKTOP-212S2PG MINGW64 ~
$ cd OneDrive
(base)
jsmba@DESKTOP-212S2PG MINGW64 ~/OneDrive
$ cd Desktop
(base)
jsmba@DESKTOP-212S2PG MINGW64 ~/OneDrive/Desktop
$ ssh -i cloudlab_2022.pem hadoop@ec2-15-223-88-74.ca-central-1.compute.amazonaw
s.com
The authenticity of host 'ec2-15-223-88-74.ca-central-1.compute.amazonaws.com (1
5.223.88.74)' can't be established.
ED25519 key fingerprint is SHA256:nDRSiwyg4J4FWzcZ2oXEZ7Sk9u4cWHNo1T2OkKvNo10.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-15-223-88-74.ca-central-1.compute.amazonaws.com'
 (ED25519) to the list of known hosts.
Last login: Sun Jun 26 13:00:54 2022

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2018.03-release-notes/
64 package(s) needed for security, out of 92 available
Run "sudo yum update" to apply all updates.

EEEEEEEEEEEEEEEEEEE MMMMMMMM           MMMMMMMM RRRRRRRRRRRRRRR
E::::::::::::::::::E M:::::::M         M:::::::M R::::::::::::::R
EE:::::EEEEEEEEE:::E M::::::::M       M::::::::M R:::::RRRRRR:::::R
  E::::E       EEEEE M:::::::::M     M:::::::::M RR::::R      R::::R
  E::::E             M::::::M:::M   M:::M::::::M   R:::R      R::::R
  E:::::EEEEEEEEEE   M:::::M M:::M M:::M M:::::M   R:::RRRRRR:::::R
  E::::::::::::::E   M:::::M  M:::M:::M  M:::::M   R:::::::::::RR
  E:::::EEEEEEEEEE   M:::::M   M:::::M   M:::::M   R:::RRRRRR::::R
  E::::E             M:::::M    M:::M    M:::::M   R:::R      R::::R
  E::::E       EEEEE M:::::M     MMM     M:::::M   R:::R      R::::R
EE:::::EEEEEEEE::::E M:::::M             M:::::M   R:::R      R::::R
E::::::::::::::::::E M:::::M             M:::::M RR::::R      R::::R
```

3. Copy data from S3 bucket to Hadoop File System

```
[hadoop@ip-172-31-32-11 ~]$ hadoop distcp s3://brainstation-dsft/eng_1M_1gram.csv /user/hadoop/eng_1M_1gram.csv
```

This code is using the distcp command. This can copy large amounts of data using MapReduce. So it is copying data from the public s3 bucker with the data, directly into HDFS. I realized I put it directly into HDFS without creating an additional directory.

Alternatively I could have run:

hadoop fs -mkdir /user/hadoop/ /user/hadoop/eng_1M_1gram

And then my code could have been:

Hadoop distcp s3://brainstation-dsft/eng_1M_1gram.csv
/user/hadoop/eng_1M_1gram/eng_1M_1gram.csv

4. Using a pyspark dataframe to garner insights, reduce the size and write it back into a HDFS directory

Creating a PySpark dataframe. Describing the size, shape and schema.

First I initiated pyspark

```
[hadoop@ip-172-31-32-11 ~]$ pyspark
```

I got a message saying I could use the pyspark by using the word 'spark'

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.4.4
      /_/

Using Python version 2.7.16 (default, Oct 14 2019 21:26:56)
SparkSession available as 'spark'.
```

Then I read the csv and saw the schema.

```
>>> df = spark.read.csv("eng_1M_1gram.csv")
>>> df.printSchema()
root
 |-- _c0: string (nullable = true)
 |-- _c1: string (nullable = true)
 |-- _c2: string (nullable = true)
 |-- _c3: string (nullable = true)
 |-- _c4: string (nullable = true)
```

Interestingly, it looks like there are five columns whose data types are strings. The columns are "nullable" which means that there could potentially have been nulls entered into these columns (source: https://stackoverflow.com/questions/39917075/pyspark-structfield-false-always-returns-nullable-true-instead-of)

I still cannot tell what the columns are about, so I will show the first 10 rows.

```
>>> df.show(10, vertical=True)
-RECORD 0--------
 _c0 | token
 _c1 | year
 _c2 | frequency
 _c3 | pages
 _c4 | books
-RECORD 1--------
 _c0 | inGermany
 _c1 | 1927
 _c2 | 2
 _c3 | 2
 _c4 | 2
-RECORD 2--------
 _c0 | inGermany
 _c1 | 1929
 _c2 | 1
 _c3 | 1
 _c4 | 1
```

It looks like the first row are header names! So instead I loaded my csv with the first row as headers.

```
>>> df = spark.read.csv("eng_1M_1gram.csv", header=True)
```

The data has 5 columns and 261823226 rows.

len(df.columns) → 5

```
>>> df.columns
['_c0', '_c1', '_c2', '_c3', '_c4']
>>> df.count()
261823226
```

Creating a new DataFrame using a SparkSQL query, only including rows with values of "data" in the token column

```
>>> spark.sql("SELECT * FROM ngrams WHERE token='data'").show(10)
+-----+----+---------+-----+-----+
|token|year|frequency|pages|books|
+-----+----+---------+-----+-----+
| data|1584|       16|   14|    1|
| data|1614|        3|    2|    1|
| data|1627|        1|    1|    1|
| data|1631|       22|   18|    1|
| data|1637|        1|    1|    1|
| data|1638|        2|    2|    1|
| data|1640|        1|    1|    1|
| data|1642|        1|    1|    1|
| data|1644|        4|    4|    1|
| data|1647|        1|    1|    1|
+-----+----+---------+-----+-----+
only showing top 10 rows
```

```
>>> df.createOrReplaceTempView("ngrams")
>>> result_df = spark.sql("SELECT * FROM ngram WHERE token='data'")
```

In the token column, in the first few rows the only value we see is data, which is a great sign! I saved this into a new dataframe called result_df.

Writing the filtered data back into HDFS directory

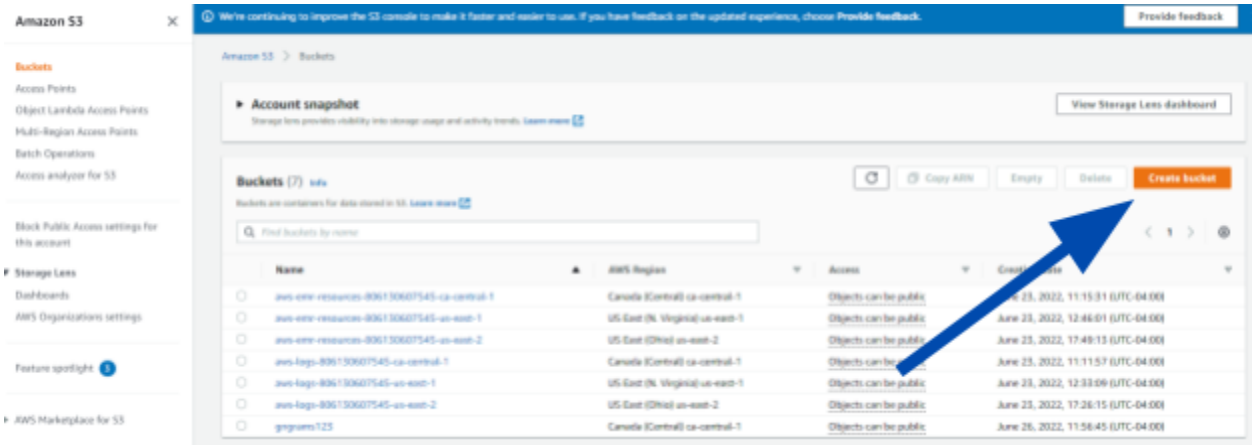
I used the write.csv method to put it into the Hadoop directory. I forgot to write header = True in this code! I had to hard code it back to normal later. Check the later sections. The I wrote hadoop fs -ls for a quick sanity check.

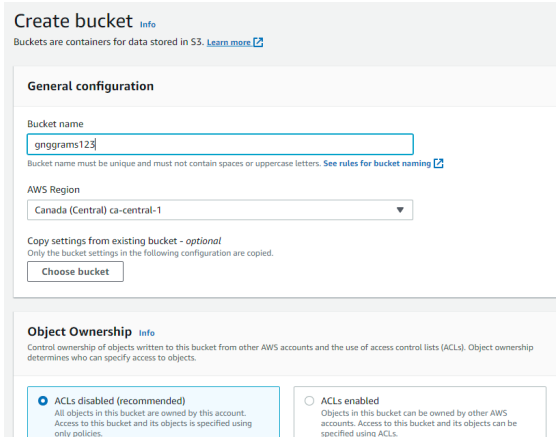
```
>>> result_df = spark.sql("SELECT * FROM ngrams WHERE token='data'")
>>> result_df.write.csv('/user/hadoop/results.csv')
>>> exit()
[hadoop@ip-172-31-32-11 ~]$ hadoop fs -ls
Found 4 items
drwxr-xr-x   - hadoop hadoop          0 2022-06-26 16:00 .sparkStaging
drwxr-xr-x   - hadoop hadoop          0 2022-06-26 13:35 eng_1M_1gram
-rw-r--r--   1 hadoop hadoop 5292105197 2022-06-26 13:51 eng_1M_1gram.csv
drwxr-xr-x   - hadoop hadoop          0 2022-06-26 15:57 results.csv
```

So yes, it is verified that results.csv is in the Hadoop directory which is what we were going for!

5. Collecting Directory contents into a single file on the local drive of head node and moving the file into the s3 bucket into my account.

First I made a bucket in my amazon account. I clicked on the S3 service, pressed "Create bucket" and gave it an original name with all the default settings.





First I will move the results_df to a local drive for the Hadoop head node.

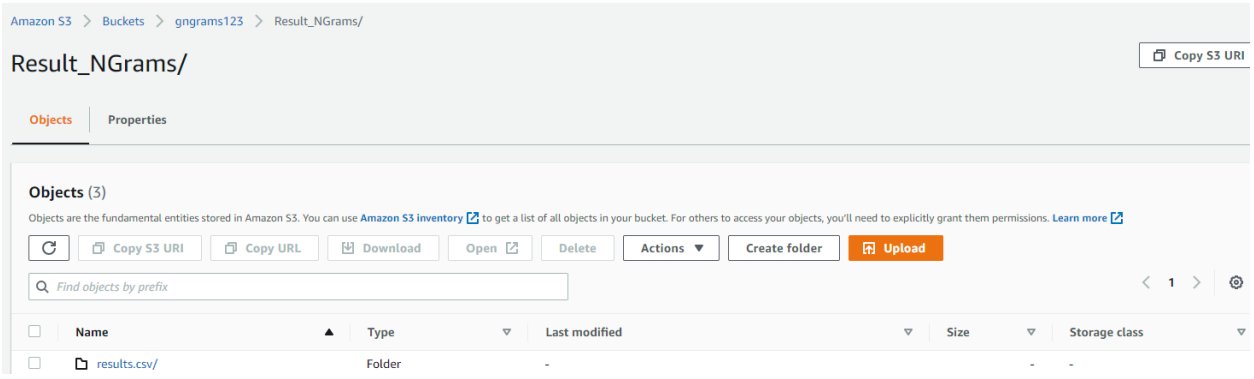
```
[hadoop@ip-172-31-32-11 ~]$ hadoop fs -getmerge /user/hadoop/results.csv results.cs
v
[hadoop@ip-172-31-32-11 ~]$ ls -lh
total 8.0K
-rw-r--r-- 1 hadoop hadoop 7.2K Jun 26 16:36 results.csv
[hadoop@ip-172-31-32-11 ~]$ which ls
alias ls='ls --color=auto'
        /bin/ls
```

In the above code I checked that in the local drive it was in a section called bins.

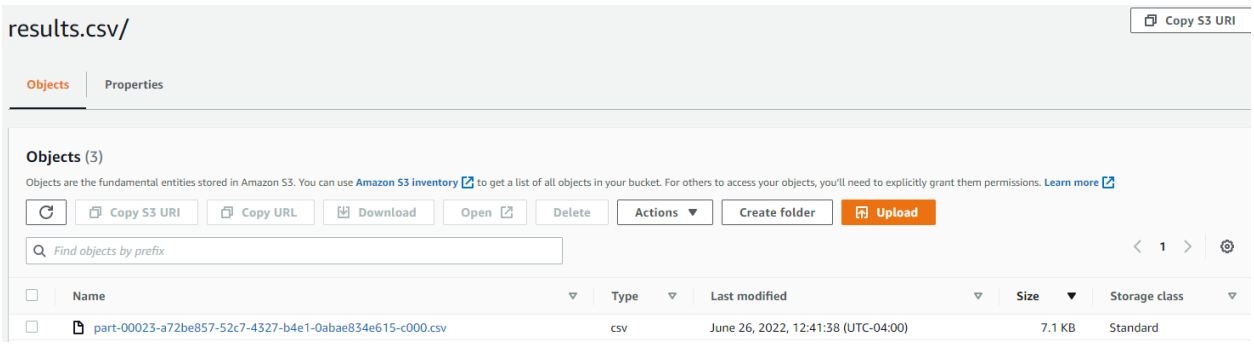
Next I moved it to the S3 bucket I made in my account, in a folder called Result_NGrams I previously made on the aws website.

```
[hadoop@ip-172-31-32-11 ~]$ hadoop distcp /user/hadoop/results.csv s3a://gngrams123/Result_NGrams
```

After a very long output, I checked to see if it was in the s3 bucket.



It was there as a folder! When I clocked on the folder I could see the csv.



6. Reading the CSV data from the S3 folder into a pandas DataFrame

First I authenticated my machine using aws configure on the command line.

```
[hadoop@ip-172-31-32-11 ~]$ aws configure
```

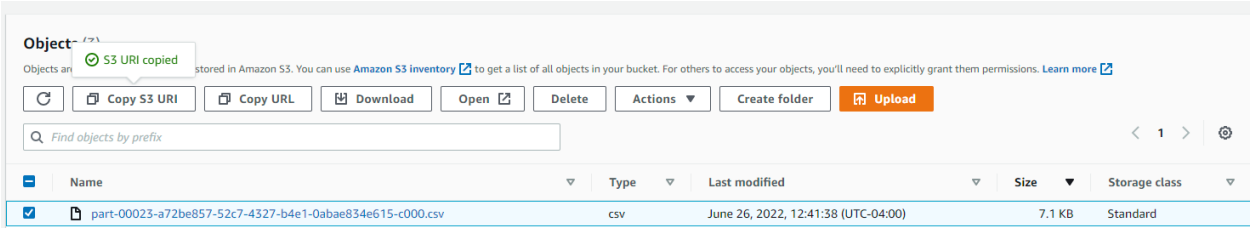
I proceeded to enter my access key and secret key, and accepted the other defaults.

This time I opened a Jupyter notebook, imported relevant libraries from my cloud_lab environment I had created a while ago, and loaded the data into a pandas dataframe.

```
import pandas as pd
```

```
pip install s3fs
```

I pressed copy s3 URI



And used it to read in the csv.

```
df = pd.read_csv('s3://gngrams123/Result_NGrams/results.csv/part-00023-a72be857-52c7-4327-b4e1-0abae834e615-c000.csv')
```

```
df.head()
```

	data	1584	16	14	1
0	data	1614	3	2	1
1	data	1627	1	1	1
2	data	1631	22	18	1
3	data	1637	1	1	1
4	data	1638	2	2	1

I did not say header = “True” when I did write.csv earlier. So I just hard coded the column names.

```
In [11]: results.csv/part-00023-a72be857-52c7-4327-b4e1-0abae834e615-c000.csv',names = ['token', 'year', 'frequency','pages','books'])

In [12]: new_df.head()
```

Out[12]:

	token	year	frequency	pages	books
0	data	1584	16	14	1
1	data	1614	3	2	1
2	data	1627	1	1	1
3	data	1631	22	18	1
4	data	1637	1	1	1

7. Plotting the number of occurrences of data over the years

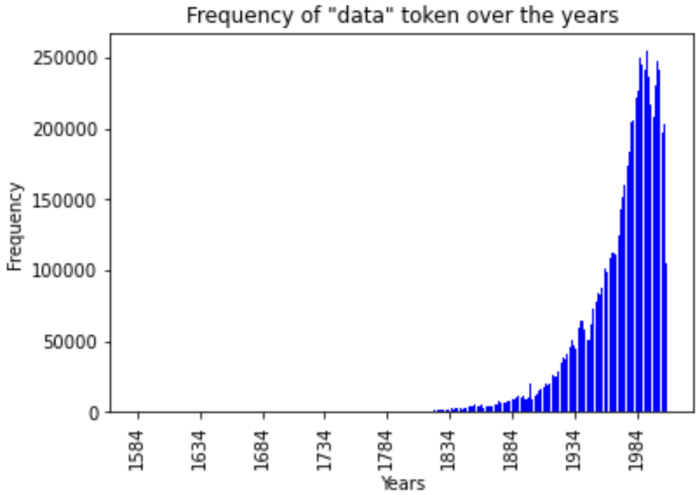
```
maxx = new_df['year'].max()
print(maxx)
print(type(maxx))
```

```
2008
<class 'int'>
```

```
minn = new_df['year'].min()
print(minn)
print(type(minn))
```

```
1584
<class 'int'>
```

```
plt.figure()
plt.bar(new_df['year'],new_df['frequency'], color = "b")
plt.xticks(rotation='vertical')
plt.xticks(np.arange(minn,maxx,step=50))
plt.xlabel('Years')
plt.ylabel('Frequency')
#plt.yticks(rotation = 10)
plt.title('Frequency of "data" token over the years')
plt.show()
```



In the above code, I potted years on the x axis and frequency on the y axis. Then appropriate labels and tick marks are added, after understanding the old and newest year.

8. Compare Hadoop and Spark as distributed file systems

Hadoop vs. Spark

Hadoop operates on a computer disk. This can make it slower, but the software is open-source and there's a lot of support online. Spark runs on computer memory, so it moves a lot faster. Hadoop is much more fault tolerant. Also Spark can help us use many coding languages like SQL to work in the cloud, whereas with Hadoop you have to code at a very low level to do simple queries.

Another advantage of Hadoop is that is

Source: https://www.youtube.com/watch?v=2PVzOHA3ktE

How HDFS stores data

This file system manages files across many nodes. The data is stored in a cluster - many computers connected to each other. However when you work with the data, it feels like it's in one computer just like a file explorer on your computer. Instead of having a super huge disk, HDFS distributes fields across many disks which reduces the processing time.

A master node manages the data and keeps track of metadata about processing, and the slave nodes process it and then store it.

In HDFS, the data is stored in blocks that are typically 64MB or 128 MB. Every block is replicated three times! So if one of your data nodes crashes, you won't face as much data loss because it will be in another node.

Source: https://www.youtube.com/watch?v=GJYEsEEfjvk

Web resources to help with this assignment:

https://www.machinelearningplus.com/plots/matplotlib-histogram-python-examples/

https://stackoverflow.com/questions/46658232/pandas-convert-column-with-year-integer-to-datetime

https://stackoverflow.com/questions/39917075/pyspark-structfield-false-always-returns-nullable-true-instead-of

https://stackoverflow.com/questions/39917075/pyspark-structfield-false-always-returns-nullable-true-instead-of

https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/FileSystemShell.html#mkdir

https://stackoverflow.com/questions/49065929/matplotlib-how-to-set-only-min-and-max-values-for-tics

https://community.cloudera.com/t5/Community-Articles/How-to-copy-between-a-cluster-and-S3-buckets/ta-p/248115

https://qiita.com/alokrawat050/items/56820afdb6968deec6a2

https://docs.aws.amazon.com/AmazonS3/latest/userguide/access-bucket-intro.html

https://docs.cloudera.com/HDPDocuments/HDP2/HDP-2.0.0.2/bk_installing_manually_book/content/rpm-chap6-4.html

https://stackoverflow.com/questions/18239567/how-can-i-download-a-file-from-an-s3-bucket-with-wget

https://docs.aws.amazon.com/config/latest/developerguide/s3-bucket-policy.html

https://saagie.zendesk.com/hc/en-us/articles/360029759552-PySpark-Read-and-Write-Files-from-HDFS