1.1

First did a very simple calculation

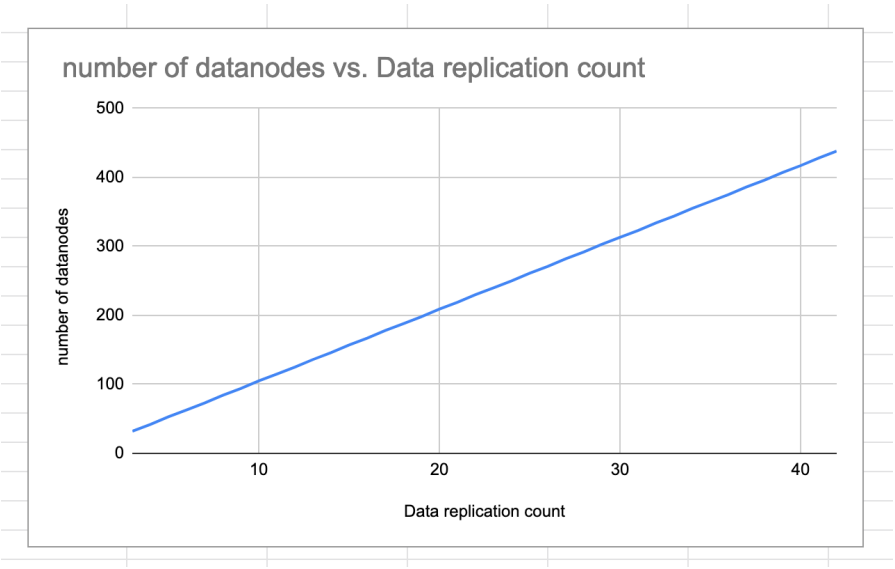
Datanode size	64	TB
Utilize rate	25%	
Replication	3	
original size	500	TB
increase	0.02	
Actual size	48	TB

Then build a table like this

1.1			
	Data replication count	Total storage in HDFS	number of datanodes
	3	1500	32
	4	2000	42
	5	2500	53
	6	3000	63
	7	3500	73
	8	4000	84
	9	4500	94
	10	5000	105
	11	5500	115
	12	6000	125
	13	6500	136
	14	7000	146
	15	7500	157
	16	8000	167
	17	8500	178
	18	9000	188

Example: I got number of data nodes 32 by 1500/48=31.25 and round up to 32

Then get the graph

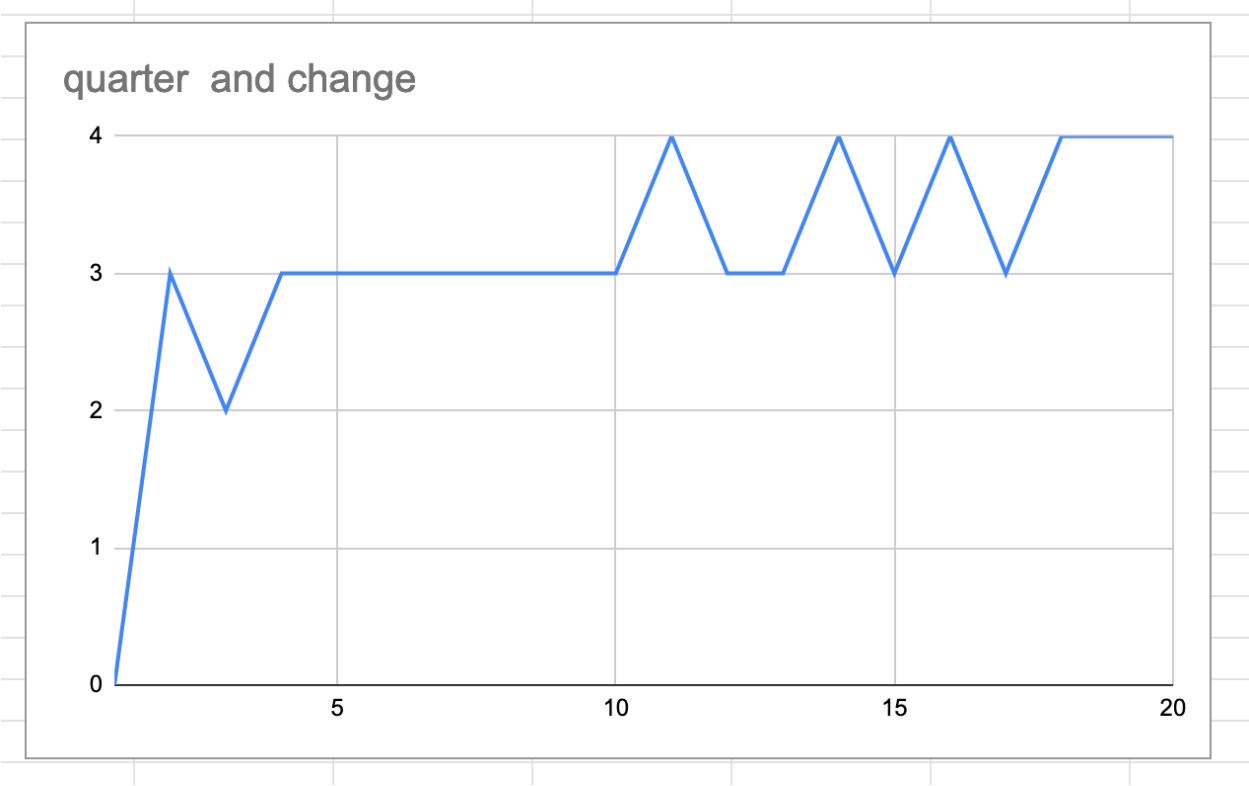


1.2

Draw the graph based on

data replication c	quarter	Total storage in HDFS	number of datanodes	change	price
13	1	6500	136	0	8339.52
	2	6630	139	3	8523.48
	3	6762.6	141	2	8646.12
	4	6897.852	144	3	8830.08
	5	7035.80904	147	3	9014.04
	6	7176.525221	150	3	9198
	7	7320.055725	153	3	9381.96
	8	7466.45684	156	3	9565.92
	9	7615.785977	159	3	9749.88
	10	7768.101696	162	3	9933.84
	11	7923.46373	166	4	10179.12
	12	8081.933005	169	3	10363.08
	13	8243.571665	172	3	10547.04
	14	8408.443098	176	4	10792.32
	15	8576.61196	179	3	10976.28
	16	8748.144199	183	4	11221.56

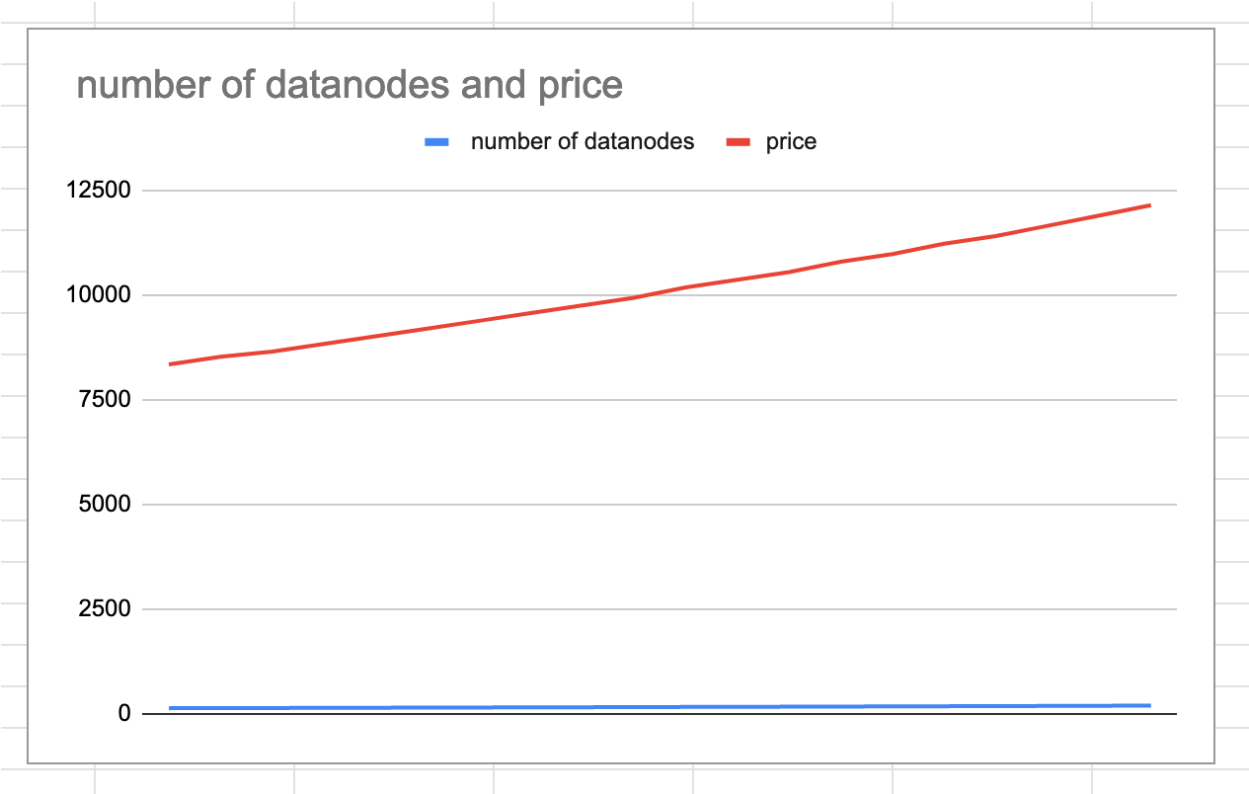
I randomly choose data replication 13 and then draw the graph



1.3

First get the unit price is 61.32 from aws website and then calculate the total price(refer to graph in 1.2) then get the graph.

	price	
61.32	8339.52	
	8523.48	
	8646.12	
	8830.08	
	9014.04	
	9198	
	9381.96	
	9565.92	
	9749.88	
	9933.84	
	10179.12	
	10363.08	
	10547.04	
	10792.32	
	10976.28	
	11221.56	
	11405.52	
	11650.8	
	11896.08	
	12141.36	



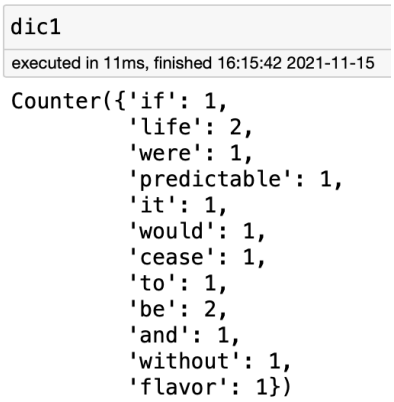
2. Assuming you have the following 4 data items on a datanode.

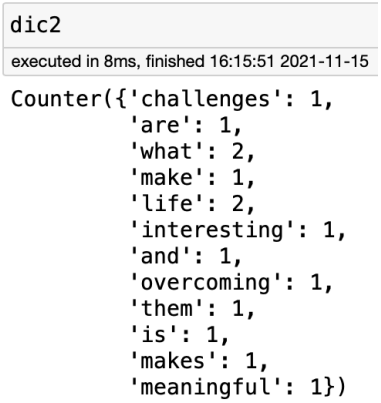
2.1. Write and submit a mapper code (in Python or pseudocode) that creates a sorted dictionary of each word (key) and frequency (value) in each document.

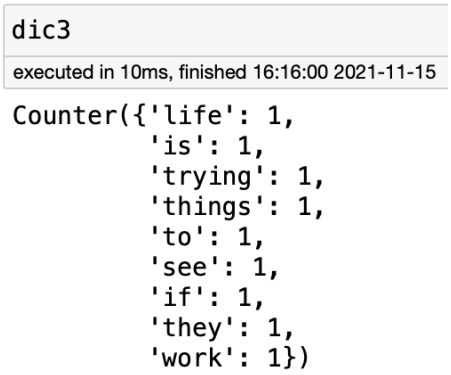
```
# Import packages we need in python
import pandas as pd
import numpy as np
import string
from collections import Counter

# Load the data items
d1 = "If life were predictable it would cease to be life, and be without flavor"
d2 = "Challenges are what make life interesting and overcoming them is what makes life meaningful"
d3 = "Life is trying things to see if they work"
d4 = "You will face many defeats in life, but never let yourself be defeated"

# We used lowercase and ignored the comma. In that case, “life”, “life,”, “Life,” and “Life” are the same.
d1 = d1.lower().replace(",","")
d2 = d2.lower().replace(",","")
```

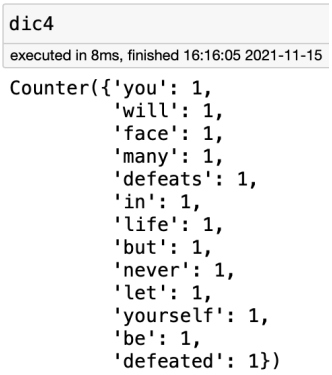
```
d3 = d3.lower().replace(",","")
d4 = d4.lower().replace(",","")

# Split the sentences into words
dic1 = Counter(d1.split())
dic2 = Counter(d2.split())
dic3 = Counter(d3.split())
dic4 = Counter(d4.split())
```

The dictionary of each word and frequency in each document is as follows.









2.2. Write and submit a reducer code (in Python or pseudocode) that takes the mapped dictionaries and gives the TOTAL frequency of ALL words.

```
# We used counter package to give total frequency of all words
dicts = [dic1, dic2, dic3, dic4]
dic = dict(sum(map(Counter, dicts), start = Counter()))
```

The total frequency of all words is:

'if': 2, 'life': 6, 'were': 1, 'predictable': 1, 'it': 1, 'would': 1, 'cease': 1, 'to': 2, 'be': 3, 'and': 2, 'without': 1, 'flavor': 1, 'challenges': 1, 'are': 1, 'what': 2, 'make': 1, 'interesting': 1, 'overcoming': 1, 'them': 1, 'is': 2, 'makes': 1, 'meaningful': 1, 'trying': 1, 'things': 1, 'see': 1, 'they': 1, 'work': 1, 'you': 1, 'will': 1, 'face': 1, 'many': 1, 'defeats': 1, 'in': 1, 'but': 1, 'never': 1, 'let': 1, 'yourself': 1, 'defeated': 1

2.3. Write and submit a reducer code (in Python or pseudocode) that takes the mapped dictionaries and gives the AVERAGE frequency of word “life” across all documents.

```
# Get the dictionary as it is in question 2 which has TOTAL frequency of ALL words
dicts = [dic1, dic2, dic3, dic4]
dic = dict(sum(map(Counter, dicts), start = Counter()))

# Average frequency = total frequency / number of documents
print("The average frequency of 'life' across all documents is: ")
print(float(dic['life']/len(dicts)))
```

The average frequency of 'life' across all documents is 1.5.

3. Based on a use case published in a paper, *Pig Vs. Hive Use Case Analysis*, Kendal, Noren, & Perel (*Int. Journal of Database Theory and Application*, 2016), the authors recommend using Hive over Pig Latin. Briefly discuss the use case where Hive performed better than Pig and discuss what industries this use case would be applicable to.

Hive outperforms Pig Latin:

In *Pig Vs. Hive Use Case Analysis*, (*Int. Journal of Database Theory and Application*, 2016), Kendal, Noren, & Perel benchmark Pig Latin against Hive by repeatedly running a query on a 1 gig and 2 gig dataset. Because of how each language handles dates, the dates for Pig were modified to include a timestamp, leading to 1.2 and 2.5 gig files. This may have impacted the results, however, it is worth noting that both datasets had the same number of rows.

Results for 30 runs of query

	1 Gig Mean	1 Gig sd	2 Gig Mean	2 Gig sd
Pig	11.79950	.382402	22.83227	.564224
Hive	10.46663	.305627	15.50277	.609189

It appears likely that the increased size caused by Pig’s need for a timestamp is not the sole reason for the performance disparity, especially as the file size increased.

The query itself was a join of three tables, with concatenation of year and month columns, and grouped by year-month, customer, and item, in order to show the number of and

total spent on each item for each customer each month.

Industrial Applications:

The query tested is very general and we expect could be used by any business that performs sales. Furthermore, a firm large enough to have invested in a hadoop database is probably dealing with *much* more than just 2 gigs of data, increasing the query-performance-gap between Pig and Hive. Finally, the use of Hive implies the company has Big Data-sized data needs. For these reasons, this case is most applicable to large retailers, technology companies (eg. Apple), and financial companies.