CSC 555 Assignment 1

Ronaldlee Ejalu
Student Number: 2020637

Suggested reading: *Mining of Massive Datasets*: Chapter 1, Chapter 2 (sections 2.1, 2.1 only).

Please be sure to submit all python code (if you are using Jupyter notebooks, please export it into the main document or into a .py file before submission).

Part 1

a) Compute (you can use any tool or software to compute answers in this part – but if you do not know to perform this computation, please talk to me about your course prerequisites):

2^{11}

```
2 raised to the power of 11 is 2048
```

$(2^4)^4$

```
2 raised to the poweer of 4 which is raised to the power of 4 is 65536.0
```

4^4

```
4 raised to the power of 4 is 256
```

8^5

```
8 raised to the power of 5 is 32768
```

842 MOD 100 (MOD is the modulo operator, a.k.a. the remainder)

```
842 MOD 100 = 42
```

837 MOD 20

```
837 MOD 20 = 17
```

22 MOD 111

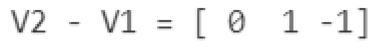
```
22 MOD 111 = 22
```

111 MOD 22

```
111 MOD 22 = 1
```

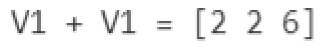
b) Given vectors V1 = (1, 1, 3) and V2 = (1, 2, 2) and a 3x3 matrix M = [(2, 1, 3), (1, 2, 1), (1, 0, 1)], compute:

V2 – V1



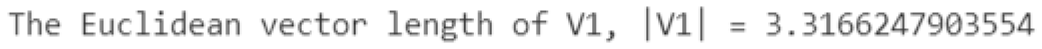
```
V2 - V1 = [ 0  1 -1]
```

V1 + V1



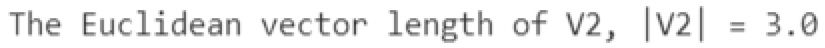
```
V1 + V1 = [2 2 6]
```

|V1| (Euclidean vector length, not the number of dimensions)



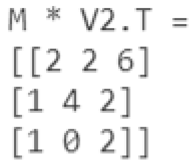
```
The Euclidean vector length of V1, |V1| = 3.3166247903554
```

|V2|



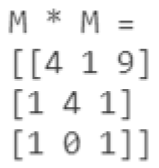
```
The Euclidean vector length of V2, |V2| = 3.0
```

M * V2 (matrix times vector, transpose it as necessary)



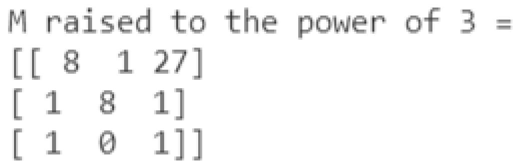
```
M * V2.T =
[[2 2 6]
[1 4 2]
[1 0 2]]
```

M * M (or M^2)



```
M * M =
[[4 1 9]
[1 4 1]
[1 0 1]]
```

M^3



```
M raised to the power of 3 =
[[ 8  1 27]
[ 1  8  1]
[ 1  0  1]]
```

c) Suppose we are flipping a coin with Head (H) and Tail (T) sides. The coin is not balanced with 0.4 probability of H coming up (and 0.6 of T). Compute the probabilities of getting:

HTHT
0.4 * 0.6 * 0.4 * 0.6 = 0.0576

THTH
0.6 * 0.4 * 0.6 * 0.4 = 0.0576
Exactly 2 Heads out of a sequence of 3-coin flips.
= 0.4 * 0.4 * 0.6 + 0.4 * 0.6 + 0.4 + 0.6 * 0.4* 0.4 = 0.096 + 0.096 + 0.096 = 0.288
Exactly 1 Tail out of sequence of 3 coin flips.
= 0.6 * 0.4 * 0.4 + 0.4 * 0.4 * 0.6 + 0.4 * 0.6 * 0.4 = 0.096 + 0.096 + 0.096 = 0.288

d) Consider a database schema consisting of two tables, Employee (<u>ID</u>, Name, Address), Project (<u>PID</u>, Name, Deadline), Assign(<u>EID</u>, <u>PID</u>, Date). Assign.EID is a foreign key referencing employee's ID and Assign.PID is a foreign key referencing the project.

Write SQL queries for:

i. Find projects that are not assigned to any employees (PID and Deadline of the project).

```
SELECT PID, NAME FROM Project WHERE PID NOT IN (SELECT PID
FROM Assign)

OR
SELECT PID, NAME FROM Project
WHERE PID IN
                (SELECT PID FROM Project
                MINUS
                SELECT PID FROM Assign
                )
```

ii. For each date, find how many assignments were made that day.

```
SELECT Date, COUNT(PID) AS numberOfAssignments
FROM Assign
GROUP BY DATE
```

iii. Find all projects that have either 0, 1, or 2 employees assigned to them (note that the <u>answer should include 0 employees</u> to be correct).

```
SELECT
       Employee.ID
       , Employee.Name
       , Employee.Address
       --, COUNT(Assign.PID) Projects
FROM Assign  RIGHT OUTER JOIN Employee
ON Assign.EID= Employee.ID
GROUP BY Employee.ID
       , Employee.Name
       , Employee.Address
HAVING COUNT(Assign.PID) < 3
```

e) Mining of Massive Datasets, Exercise 1.3.3
<u>Justify your answer</u> (giving an example only would be worth partial credit)

If the hash function h(x) = x mod 15, that the remainder when x is divided by 15, this means that h(x) can only have 15 buckets numbered from 0, 1, 2, 3, 4, 5, 6, ……….14. The bucket number in the range of 0 to B − 1, which is an integer where B represents the number of buckets; h(x) will send approximately equal numbers of hash-keys to each of the B buckets if hash keys are uniformly distributed into all buckets. So, with h(x) = x mod 15 this means that 1/15th of the integers will be assigned to each of the buckets. It is always preffered to select a prime number as B (the number of buckets) because taking B, which has any common factor with all possible hash-keys results in a nonrandom distribution into buckets. Choosing B as 11, 31, we would find that either 1/11th or 1/31th of the even integers get sent to each of the 11 or 31 buckets, which makes the hash function to operate well. The hash function is non random in its behavior if we chose B as 10 resulting in the h(x) values of 0, 2, 4, 6, 8.

f) Hadoop Distributed Filesystem.

i. What are the guarantees offered by a replication factor of 3 (3 copies of each block)?

So that we don't lose all copies of blocks due to a rack failure and this results in making HDFS fault-tolerant. This implies that blocks are replicated with two more copies at three different compute nodes in the cluster.

ii. What action does NameNode have to take when a machine in the Hadoop cluster fails/crashes?

The NameNode detects any failed Map workers since it periodically pings the worker processes. All the Map tasks that were assigned to this worker will have to be redone, even if they had been completed. Not only does the NameNode sets the status of each of these Map tasks to idle and will schedule them on a Worker when one becomes available, but also must inform each reduce task that the location of its input from that Map task has changed.

iii. What is the storage cost for a file of size X MBs, when the HDFS replication factor is set to 3?

If I had a file of size X MBs, it would be divided into 4 blocks and this amounts to 4 * X * 3 = 12XMBs.

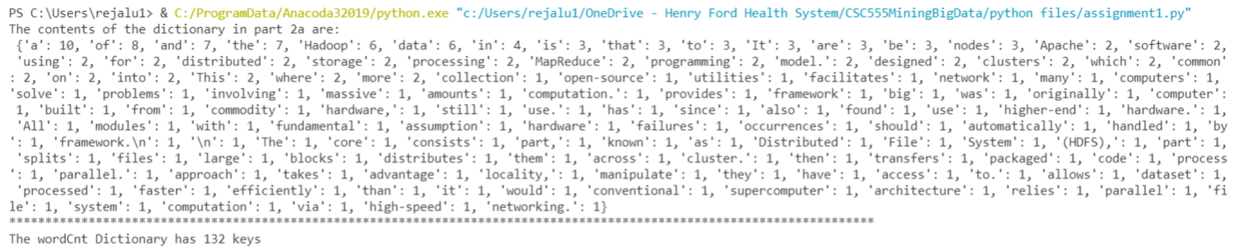
Part 2

a) Write python code that is going to read a text file and compute a total word count using a dictionary (e.g., {'Hadoop':3, 'Cloud': 2, 'MapReduce':4}. For our purposes, a word is anything split by space (.split(' ')), even if it includes things like punctuation.

Test the code on HadoopBlurb.txt (attached to the homework, from Apache Hadoop Wikipedia entry).

```
def sortDict(consolidatedDict):
    """Function that returns a sorted dictionary in descending order  """

    sortedMergedDict = {key: value for (key, value) in sorted(consolidatedDict.it
ems(), key=lambda wordCnt: wordCnt[1], reverse=True)}
    return sortedMergedDict                                 # return the sorted d
ictionary


fileLocation = 'C:\\Users\\rejalu1\\OneDrive -
 Henry Ford Health System\\CSC555MiningBigData\\data\\HadoopBlurb.txt'

def readFile(fileLocation):
    """ Function that generates a list of words"""
    wordL = []                                              # declare an empty li
st of words
    with open(fileLocation) as fN:
        lines = fN.readlines()                              # read the file and g
enerate a list of strings

        for item in range(len(lines)):                      # Loop through the li
st of strings
            splittedLinesL = lines[item].split(' ')         # split the string to
 create a list of words

            for splittedItem in range(len(splittedLinesL)): # loop through the li
st of words and add words to the wordL(list)
                if len(splittedLinesL[splittedItem]) == 0:  # if there are empty
spaces just pass
                    pass
                else:
                    wordL.append(splittedLinesL[splittedItem])
        return wordL                                        # return a list of wo
rds

listOfWords = readFile(fileLocation)                        # return a list of wo
rds
# print(listOfWords)
def wordCnt(listOfWords):
    wordCnt = {}                                                # define an empty
 dictionary
    for i in range(len(listOfWords)):                           # iterate through
 the list of words and count the number of words
        if listOfWords[i] in wordCnt.keys():
            wordCnt[listOfWords[i]] = wordCnt[listOfWords[i]] + 1
```

How many keys does your dictionary have?



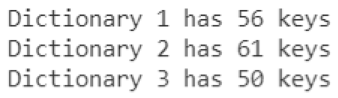
```
PS C:\Users\rejalu1> & C:/ProgramData/Anacoda32019/python.exe "c:/Users/rejalu1/OneDrive - Henry Ford Health System/CSC555MiningBigData/python files/assignment1.py"
The contents of the dictionary in part 2a are:
 {'a': 10, 'of': 8, 'and': 7, 'the': 7, 'Hadoop': 6, 'data': 6, 'in': 4, 'is': 3, 'that': 3, 'to': 3, 'It': 3, 'are': 3, 'be': 3, 'nodes': 3, 'Apache': 2, 'software': 2,
 'using': 2, 'for': 2, 'distributed': 2, 'storage': 2, 'processing': 2, 'MapReduce': 2, 'programming': 2, 'model.': 2, 'designed': 2, 'clusters': 2, 'which': 2, 'common'
: 2, 'on': 2, 'into': 2, 'This': 2, 'where': 2, 'more': 2, 'collection': 1, 'open-source': 1, 'utilities': 1, 'facilitates': 1, 'network': 1, 'many': 1, 'computers': 1,
'solve': 1, 'problems': 1, 'involving': 1, 'massive': 1, 'amounts': 1, 'computation.': 1, 'provides': 1, 'framework': 1, 'big': 1, 'was': 1, 'originally': 1, 'computer':
 1, 'built': 1, 'from': 1, 'commodity': 1, 'hardware,': 1, 'still': 1, 'use.': 1, 'has': 1, 'since': 1, 'also': 1, 'found': 1, 'use': 1, 'higher-end': 1, 'hardware.': 1,
 'All': 1, 'modules': 1, 'with': 1, 'fundamental': 1, 'assumption': 1, 'hardware': 1, 'failures': 1, 'occurrences': 1, 'should': 1, 'automatically': 1, 'handled': 1, 'by
': 1, 'framework.\n': 1, '\n': 1, 'The': 1, 'core': 1, 'consists': 1, 'part,': 1, 'known': 1, 'as': 1, 'Distributed': 1, 'File': 1, 'System': 1, '(HDFS),': 1, 'part': 1,
 'splits': 1, 'files': 1, 'large': 1, 'blocks': 1, 'distributes': 1, 'them': 1, 'across': 1, 'cluster.': 1, 'then': 1, 'transfers': 1, 'packaged': 1, 'code': 1, 'process
': 1, 'parallel.': 1, 'approach': 1, 'takes': 1, 'advantage': 1, 'locality,': 1, 'manipulate': 1, 'they': 1, 'have': 1, 'access': 1, 'to.': 1, 'allows': 1, 'dataset': 1,
 'processed': 1, 'faster': 1, 'efficiently': 1, 'than': 1, 'it': 1, 'would': 1, 'conventional': 1, 'supercomputer': 1, 'architecture': 1, 'relies': 1, 'parallel': 1, 'fi
le': 1, 'system': 1, 'computation': 1, 'via': 1, 'high-speed': 1, 'networking.': 1}
************************************************************************************************************************
The wordCnt Dictionary has 132 keys
```

The dictionary has 132 keys as shown in the screen shot output above.

b) Write python code that is going to create three different word count dictionaries, splitting the words at random between the three. Each time you process the word, choose at random which count dictionary to add it to (that means some words will appear in multiple dictionaries simultaneously).

```
def randomSplitOfWords(listOfWords):
    """
    Function that creates three different word count dictionaries
    splitting the words at random between the three dicts
    """
    wordCnt1 = {}                                                   # define an empt
y dictionary
    wordCnt2 = {}
    wordCnt3 = {}
    for i in range(len(listOfWords)):                               # iterate throug
h the list of words and count the number of words
        randomNum = random.randint(0,2)                             # generate a ran
dom number between 0 and 2 inclusive.
        if randomNum == 0:
            if listOfWords[i] in wordCnt1.keys():
                wordCnt1[listOfWords[i]] = wordCnt1[listOfWords[i]] + 1
            else:
                wordCnt1[listOfWords[i]] = 1
        elif randomNum == 1:
            if listOfWords[i] in wordCnt2.keys():
                wordCnt2[listOfWords[i]] = wordCnt2[listOfWords[i]] + 1
            else:
                wordCnt2[listOfWords[i]] = 1
        elif randomNum == 2:
            if listOfWords[i] in wordCnt3.keys():
                wordCnt3[listOfWords[i]] = wordCnt3[listOfWords[i]] + 1
            else:
                wordCnt3[listOfWords[i]] = 1
    return wordCnt1, wordCnt2, wordCnt3                         # return the three di
ctionaries of words


# Invoke the randomSplitOfWords helper function, which takes the list of words as
 a parameter and
# create three different word count dictionaries splitting the words
# at random between the three dicts
wCnt1, wCnt2, wCnt3 = randomSplitOfWords(listOfWords)
print('Dictionary 1 has %s keys' %(len(wCnt1.keys())))
print('Dictionary 2 has %s keys' %(len(wCnt2.keys())))
print('Dictionary 3 has %s keys' %(len(wCnt3.keys())))
```

How many keys does each dictionary have?



```
Dictionary 1 has 56 keys
Dictionary 2 has 61 keys
Dictionary 3 has 50 keys
```

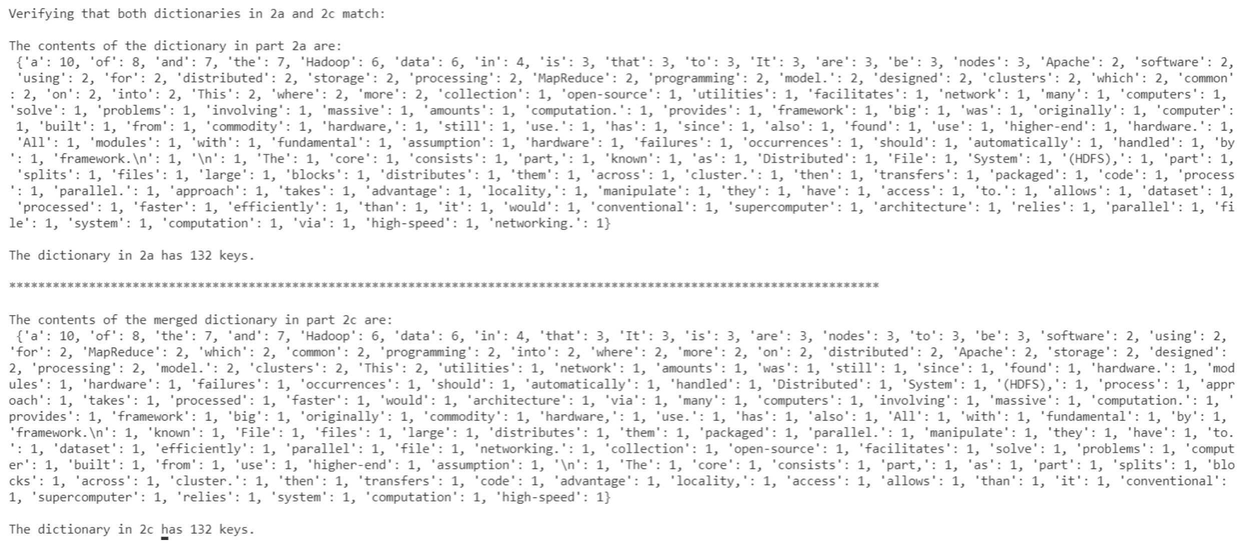
From the screenshot output above:

- Dictionary 1 had 56 keys.
- Dictionary 2 had 61 keys.
- Dictionary 3 had 50 keys.

c) Write python code to merge the three dictionaries into one and verify that it matches the dictionary from Part 2-a.

```
def mergeDicts(dict1, dict2, dict3):
    """
    Function that returns a merged dictionary after using
    Counter function from the collections package to merge
    multiple dictionaries across a common key
    """
    tempMergedDict = {}                                     # declare empty dicti
onaries
    mergedDict = {}

    tempMergedDict = Counter(dict1) + Counter(dict2)        # Use the Counter fun
ction to merge dict1 and dict2 together
    mergedDict = Counter(tempMergedDict) + Counter(dict3)   # again merge the thi
rd dictionary to the merged dict1 and dict2

    return mergedDict                                       # return the merged D
ictionary, which contents all the contents of three dictionaries.

consolidatedDict = mergeDicts(wCnt1, wCnt2, wCnt3)             # helper function
which takes three dictionaries as parameters and merges them together
sortedMergedDict = sortDict(consolidatedDict)                  # helper function
that returns a sorted dictionary
print('Verifying that both dictionaries in 2a and 2c match:\n')
print('The contents of the dictionary in part 2a are: \n %s\n'%(cntWordsDict))

print('The dictionary in 2a has %s\n' %(len(cntWordsDict.keys())))
print('****'*30)
print('\nThe contents of the merged dictionary in part 2c are: \n %s\n' %(sortedM
ergedDict))
print('The dictionary in 2c has %s' %(len(sortedMergedDict.keys())))
```

A screenshot below verifies the contents of both dictionaries in 2a and 2c:

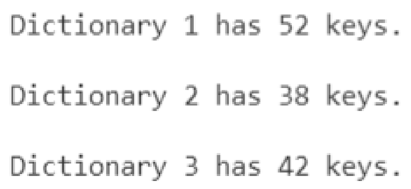


```
Verifying that both dictionaries in 2a and 2c match:

The contents of the dictionary in part 2a are:
 {'a': 10, 'of': 8, 'and': 7, 'the': 7, 'Hadoop': 6, 'data': 6, 'in': 4, 'is': 3, 'that': 3, 'to': 3, 'It': 3, 'are': 3, 'be': 3, 'nodes': 3, 'Apache': 2, 'software': 2,
 'using': 2, 'for': 2, 'distributed': 2, 'storage': 2, 'processing': 2, 'MapReduce': 2, 'programming': 2, 'model.': 2, 'designed': 2, 'clusters': 2, 'which': 2, 'common'
: 2, 'on': 2, 'into': 2, 'This': 2, 'where': 2, 'more': 2, 'collection': 1, 'open-source': 1, 'utilities': 1, 'facilitates': 1, 'network': 1, 'many': 1, 'computers': 1,
'solve': 1, 'problems': 1, 'involving': 1, 'massive': 1, 'amounts': 1, 'computation.': 1, 'provides': 1, 'framework': 1, 'big': 1, 'was': 1, 'originally': 1, 'computer':
 1, 'built': 1, 'from': 1, 'commodity': 1, 'hardware,': 1, 'still': 1, 'use.': 1, 'has': 1, 'since': 1, 'also': 1, 'found': 1, 'use': 1, 'higher-end': 1, 'hardware.': 1,
 'All': 1, 'modules': 1, 'with': 1, 'fundamental': 1, 'assumption': 1, 'hardware': 1, 'failures': 1, 'occurrences': 1, 'should': 1, 'automatically': 1, 'handled': 1, 'by
': 1, 'framework.\n': 1, '\n': 1, 'The': 1, 'core': 1, 'consists': 1, 'part,': 1, 'known': 1, 'as': 1, 'Distributed': 1, 'File': 1, 'System': 1, '(HDFS),': 1, 'part': 1,
 'splits': 1, 'files': 1, 'large': 1, 'blocks': 1, 'distributes': 1, 'them': 1, 'across': 1, 'cluster.': 1, 'then': 1, 'transfers': 1, 'packaged': 1, 'code': 1, 'process
': 1, 'parallel.': 1, 'approach': 1, 'takes': 1, 'advantage': 1, 'locality,': 1, 'manipulate': 1, 'they': 1, 'have': 1, 'access': 1, 'to.': 1, 'allows': 1, 'dataset': 1,
 'processed': 1, 'faster': 1, 'efficiently': 1, 'than': 1, 'it': 1, 'would': 1, 'conventional': 1, 'supercomputer': 1, 'architecture': 1, 'relies': 1, 'parallel': 1, 'fi
le': 1, 'system': 1, 'computation': 1, 'via': 1, 'high-speed': 1, 'networking.': 1}

The dictionary in 2a has 132 keys.

******************************************************************************************************************

The contents of the merged dictionary in part 2c are:
 {'a': 10, 'of': 8, 'the': 7, 'and': 7, 'Hadoop': 6, 'data': 6, 'in': 4, 'that': 3, 'It': 3, 'is': 3, 'are': 3, 'nodes': 3, 'to': 3, 'be': 3, 'software': 2, 'using': 2,
'for': 2, 'MapReduce': 2, 'which': 2, 'common': 2, 'programming': 2, 'into': 2, 'where': 2, 'more': 2, 'on': 2, 'distributed': 2, 'Apache': 2, 'storage': 2, 'designed':
2, 'processing': 2, 'model.': 2, 'clusters': 2, 'This': 2, 'utilities': 1, 'network': 1, 'amounts': 1, 'was': 1, 'still': 1, 'since': 1, 'found': 1, 'hardware.': 1, 'mod
ules': 1, 'hardware': 1, 'failures': 1, 'occurrences': 1, 'should': 1, 'automatically': 1, 'handled': 1, 'Distributed': 1, 'System': 1, '(HDFS),': 1, 'process': 1, 'appr
oach': 1, 'takes': 1, 'processed': 1, 'faster': 1, 'would': 1, 'architecture': 1, 'via': 1, 'many': 1, 'computers': 1, 'involving': 1, 'massive': 1, 'computation.': 1, '
provides': 1, 'framework': 1, 'big': 1, 'originally': 1, 'commodity': 1, 'hardware,': 1, 'use.': 1, 'has': 1, 'also': 1, 'All': 1, 'with': 1, 'fundamental': 1, 'by': 1,
'framework.\n': 1, 'known': 1, 'File': 1, 'files': 1, 'large': 1, 'distributes': 1, 'them': 1, 'packaged': 1, 'parallel.': 1, 'manipulate': 1, 'they': 1, 'have': 1, 'to.
': 1, 'dataset': 1, 'efficiently': 1, 'parallel': 1, 'file': 1, 'networking.': 1, 'collection': 1, 'open-source': 1, 'facilitates': 1, 'solve': 1, 'problems': 1, 'comput
er': 1, 'built': 1, 'from': 1, 'use': 1, 'higher-end': 1, 'assumption': 1, '\n': 1, 'The': 1, 'core': 1, 'consists': 1, 'part,': 1, 'as': 1, 'part': 1, 'splits': 1, 'blo
cks': 1, 'across': 1, 'cluster.': 1, 'then': 1, 'transfers': 1, 'code': 1, 'advantage': 1, 'locality,': 1, 'access': 1, 'allows': 1, 'than': 1, 'it': 1, 'conventional':
1, 'supercomputer': 1, 'relies': 1, 'system': 1, 'computation': 1, 'high-speed': 1}

The dictionary in 2c has 132 keys.
```

d) Write python code that is going to deterministically assign each word to one of the three dictionaries instead. For example, you can make that assignment using the remainder (YourNumber % 3 will always return 0, 1, or 2 depending on the number). You can convert a string into a numeric value using hash (e.g., hash('Hadoop.')). We will talk about hashing in more detail later in the quarter.

```
# Part 2d
# Deterministically assign each word to one of the three dictionaries
def deterministicSplitOfWords(listOfWords):
    """
    Function that creates three different word count dictionaries
    splitting the words at random between the three dicts
    """
    dWordCnt1 = {}                                                  # define an emp
ty dictionary
    dWordCnt2 = {}
    dWordCnt3 = {}
    for i in range(len(listOfWords)):                              # iterate throug
h the list of words and count the number of words
        detValue = hash(listOfWords[i]) % 3                        # derive the det
erministic value

        if detValue == 0:
            if listOfWords[i] in dWordCnt1.keys():
                dWordCnt1[listOfWords[i]] = dWordCnt1[listOfWords[i]] + 1
            else:
                dWordCnt1[listOfWords[i]] = 1

        elif detValue == 1:
            if listOfWords[i] in dWordCnt2.keys():
                dWordCnt2[listOfWords[i]] = dWordCnt2[listOfWords[i]] + 1
            else:
                dWordCnt2[listOfWords[i]] = 1

        else: # when the deterministic value is 2
            # print('detValue: %s'%(detValue))
           # for debugging purposes
            if listOfWords[i] in dWordCnt3.keys():
                dWordCnt3[listOfWords[i]] = dWordCnt3[listOfWords[i]] + 1
            else:
                dWordCnt3[listOfWords[i]] = 1

    return dWordCnt1, dWordCnt2, dWordCnt3
           # return the three dictionaries of words

deterDict1, deterDict2, deterDict3 = deterministicSplitOfWords(listOfWords)

print('Dictionary 1 has %s keys.\n' %(len(deterDict1.keys())))
print('Dictionary 2 has %s keys.\n' %(len(deterDict2.keys())))
print('Dictionary 3 has %s keys.\n' %(len(deterDict3.keys())))
```

How many keys does each dictionary have?



```
Dictionary 1 has 52 keys.

Dictionary 2 has 38 keys.

Dictionary 3 has 42 keys.
```

Dictionary 1 has 52 keys.

Dictionary 2 has 38 keys.

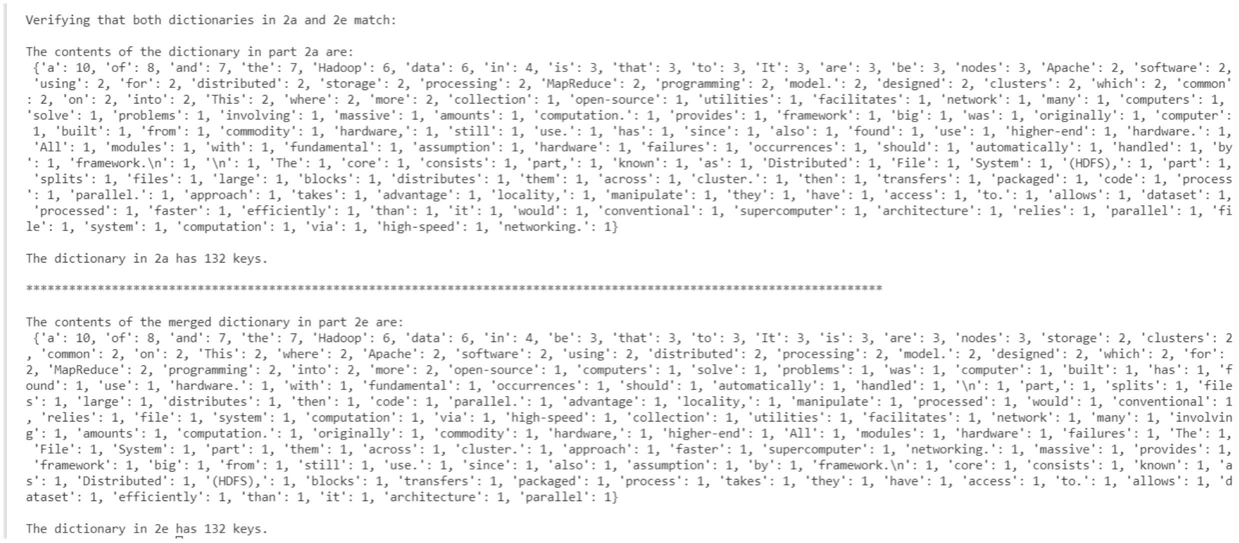
Dictionary 3 has 42 keys.

e) Write python code to merge the three dictionaries into one and verify that it matches the dictionary from Part 2-a.

```
def mergeDicts(dict1, dict2, dict3):
    """
    Function that returns a merged dictionary after using
    Counter function from the collections package to merge
    multiple dictionaries across a common key
    """
    tempMergedDict = {}                                     # declare empty dicti
onaries
    mergedDict = {}

    tempMergedDict = Counter(dict1) + Counter(dict2)        # Use the Counter fun
ction to merge dict1 and dict2 together
    mergedDict = Counter(tempMergedDict) + Counter(dict3)   # again merge the thi
rd dictionary to the merged dict1 and dict2

    return mergedDict                                       # return the merged D
ictionary, which contents all the contents of three dictionaries.


# Part 2e
# Merge the three dictionaries in part 2d into one.
partEMergedDict = mergeDicts(deterDict1, deterDict2, deterDict3)                # he
lper function which takes three dictionaries as parameters and merges them togeth
er
sortedEMergedDict = sortDict(partEMergedDict)                                    # h
elper function that returns a sorted dictionary
print('Verifying that both dictionaries in 2a and 2e match:\n')
print('The contents of the dictionary in part 2a are: \n %s\n'%(cntWordsDict))

print('The dictionary in 2a has %s keys.\n' %(len(cntWordsDict.keys())))
print('****'*30)
print('\nThe contents of the merged dictionary in part 2e are: \n %s\n' %(sortedE
MergedDict))
print('The dictionary in 2e has %s keys.' %(len(sortedEMergedDict.keys())))
```

Verifying that the merged dictionary matches with the dictionary in part2a.

The screenshot below verifies that:



```
Verifying that both dictionaries in 2a and 2e match:

The contents of the dictionary in part 2a are:
 {'a': 10, 'of': 8, 'and': 7, 'the': 7, 'Hadoop': 6, 'data': 6, 'in': 4, 'is': 3, 'that': 3, 'to': 3, 'It': 3, 'are': 3, 'be': 3, 'nodes': 3, 'Apache': 2, 'software': 2,
 'using': 2, 'for': 2, 'distributed': 2, 'storage': 2, 'processing': 2, 'MapReduce': 2, 'programming': 2, 'model.': 2, 'designed': 2, 'clusters': 2, 'which': 2, 'common'
: 2, 'on': 2, 'into': 2, 'This': 2, 'where': 2, 'more': 2, 'collection': 1, 'open-source': 1, 'utilities': 1, 'facilitates': 1, 'network': 1, 'many': 1, 'computers': 1,
'solve': 1, 'problems': 1, 'involving': 1, 'massive': 1, 'amounts': 1, 'computation.': 1, 'provides': 1, 'framework': 1, 'big': 1, 'was': 1, 'originally': 1, 'computer':
 1, 'built': 1, 'from': 1, 'commodity': 1, 'hardware,': 1, 'still': 1, 'use.': 1, 'has': 1, 'since': 1, 'also': 1, 'found': 1, 'use': 1, 'higher-end': 1, 'hardware.': 1,
 'All': 1, 'modules': 1, 'with': 1, 'fundamental': 1, 'assumption': 1, 'hardware': 1, 'failures': 1, 'occurrences': 1, 'should': 1, 'automatically': 1, 'handled': 1, 'by
': 1, 'framework.\n': 1, '\n': 1, 'The': 1, 'core': 1, 'consists': 1, 'part,': 1, 'known': 1, 'as': 1, 'Distributed': 1, 'File': 1, 'System': 1, '(HDFS),': 1, 'part': 1,
 'splits': 1, 'files': 1, 'large': 1, 'blocks': 1, 'distributes': 1, 'them': 1, 'across': 1, 'cluster.': 1, 'then': 1, 'transfers': 1, 'packaged': 1, 'code': 1, 'process
': 1, 'parallel.': 1, 'approach': 1, 'takes': 1, 'advantage': 1, 'locality,': 1, 'manipulate': 1, 'they': 1, 'have': 1, 'access': 1, 'to.': 1, 'allows': 1, 'dataset': 1,
 'processed': 1, 'faster': 1, 'efficiently': 1, 'than': 1, 'it': 1, 'would': 1, 'conventional': 1, 'supercomputer': 1, 'architecture': 1, 'relies': 1, 'parallel': 1, 'fi
le': 1, 'system': 1, 'computation': 1, 'via': 1, 'high-speed': 1, 'networking.': 1}

The dictionary in 2a has 132 keys.

****************************************************************************************************************************

The contents of the merged dictionary in part 2e are:
 {'a': 10, 'of': 8, 'and': 7, 'the': 7, 'Hadoop': 6, 'data': 6, 'in': 4, 'be': 3, 'that': 3, 'to': 3, 'It': 3, 'is': 3, 'are': 3, 'nodes': 3, 'storage': 2, 'clusters': 2
, 'common': 2, 'on': 2, 'This': 2, 'where': 2, 'Apache': 2, 'software': 2, 'using': 2, 'distributed': 2, 'processing': 2, 'model.': 2, 'designed': 2, 'which': 2, 'for':
2, 'MapReduce': 2, 'programming': 2, 'into': 2, 'more': 2, 'open-source': 1, 'computers': 1, 'solve': 1, 'problems': 1, 'was': 1, 'computer': 1, 'built': 1, 'has': 1, 'f
ound': 1, 'use': 1, 'hardware.': 1, 'with': 1, 'fundamental': 1, 'occurrences': 1, 'should': 1, 'automatically': 1, 'handled': 1, '\n': 1, 'part,': 1, 'splits': 1, 'file
s': 1, 'large': 1, 'distributes': 1, 'then': 1, 'code': 1, 'parallel.': 1, 'advantage': 1, 'locality,': 1, 'manipulate': 1, 'processed': 1, 'would': 1, 'conventional': 1
, 'relies': 1, 'file': 1, 'system': 1, 'computation': 1, 'via': 1, 'high-speed': 1, 'collection': 1, 'utilities': 1, 'facilitates': 1, 'network': 1, 'many': 1, 'involvin
g': 1, 'amounts': 1, 'computation.': 1, 'originally': 1, 'commodity': 1, 'hardware,': 1, 'higher-end': 1, 'All': 1, 'modules': 1, 'hardware': 1, 'failures': 1, 'The': 1,
 'File': 1, 'System': 1, 'part': 1, 'them': 1, 'across': 1, 'cluster.': 1, 'approach': 1, 'faster': 1, 'supercomputer': 1, 'networking.': 1, 'massive': 1, 'provides': 1,
 'framework': 1, 'big': 1, 'from': 1, 'still': 1, 'use.': 1, 'since': 1, 'also': 1, 'assumption': 1, 'by': 1, 'framework.\n': 1, 'core': 1, 'consists': 1, 'known': 1, 'a
s': 1, 'Distributed': 1, '(HDFS),': 1, 'blocks': 1, 'transfers': 1, 'packaged': 1, 'process': 1, 'takes': 1, 'they': 1, 'have': 1, 'access': 1, 'to.': 1, 'allows': 1, 'd
ataset': 1, 'efficiently': 1, 'than': 1, 'it': 1, 'architecture': 1, 'parallel': 1}

The dictionary in 2e has 132 keys.
```

Both dictionaries in 2a and 2e have the same contents and number of keys, which is 132 keys.

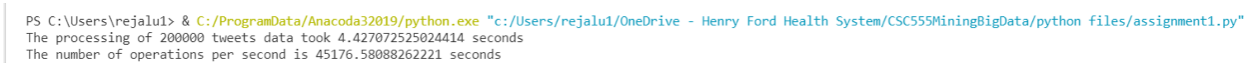
Part 3

Write (and test) python code that is going to measure the speed of reading from the web (using urllib or similar), reading from a file and writing to a file on your computer. That means your code will read or write some amount of data, time the operation, and compute the read or write rate (in MBytes/sec). Each of the measuring operations has to execute for at least 4 seconds.

a) Compute the speed of reading from disk

```
def extractLine():
    """Reading the file in chunks"""
    with open(fileName, 'rb') as f:
        for item in f:
            yield item

startTime = time.time()
chunkSize = 200000
generatedLines = extractLine()                                 # invoke a helper ext
ractLine to read  the file in chunks
screenNameDict = {}
fileItemsL = [i for i, j in zip(generatedLines, range(chunkSize))]
endTime = time.time()
print('The processing of 200000 tweets data took %s seconds' %(endTime-
startTime))
print('The number of operations per second is %s seconds' %(200000/(endTime-
startTime)))
```

From the screenshot below:

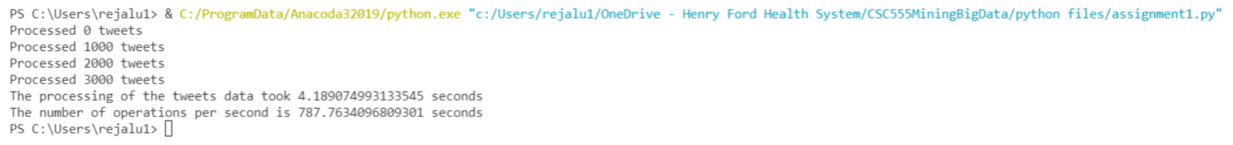


```
PS C:\Users\rejalu1> & C:/ProgramData/Anacoda32019/python.exe "c:/Users/rejalu1/OneDrive - Henry Ford Health System/CSC555MiningBigData/python files/assignment1.py"
The processing of 200000 tweets data took 4.427072525024414 seconds
The number of operations per second is 45176.58088262221 seconds
```

The reading of 200,000 tweets from the file took 4 seconds.

b) Compute the speed of reading from the web

```
os.chdir('C:/Users/rejalu1/OneDrive -
 Henry Ford Health System/CSC555MiningBigData/data')

tweetdata = """http://dbgroup.cdm.depaul.edu/DSC450/OneDayOfTweets.txt"""
startTime = time.time()                                                     # sta
rt time of processing the file in web

webFD = urllib.request.urlopen(tweetdata)

for i in range(3300):
    if i % 1000 == 0: # Print a message every 500th tweet read
        print ("Processed " + str(i) + " tweets")
    try:
        itemResponse = webFD.readline()                                       # r
ead one line at a time
    except Exception:
        continue
endTime = time.time()                                                        # en
d time of processing of writing the tweets data to a file.
print('The processing of the tweets data took %s seconds' %(endTime-startTime))
print('The number of operations per second is %s seconds' %(3300/(endTime-
startTime)))
```



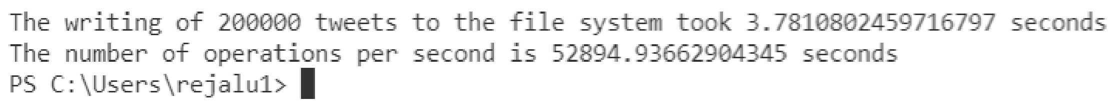
```
PS C:\Users\rejalu1> & C:/ProgramData/Anacoda32019/python.exe "c:/Users/rejalu1/OneDrive - Henry Ford Health System/CSC555MiningBigData/python files/assignment1.py"
Processed 0 tweets
Processed 1000 tweets
Processed 2000 tweets
Processed 3000 tweets
The processing of the tweets data took 4.189074993133545 seconds
The number of operations per second is 787.7634096809301 seconds
PS C:\Users\rejalu1> 
```

From the above screenshot, the processing of the tweets data took around 4 seconds.

c) Compute the speed of writing to disk

```
# Part 3(c)
# Compute the speed of writing to disk
startTime = time.time()
csvf = open('C:/Users/rejalu1/OneDrive -
 Henry Ford Health System/CSC555MiningBigData/data/WrittenFileOfTweets.csv', 'wb'
)
for i in range(200000):
    if i % 10000== 0: # Print a message every 10000th tweet read
        print ("Processed " + str(i) + " tweets")
    try:
        csvf.write(fileItemsL[i])

    except Exception:
        continue
csvf.close()
endTime = time.time()
print('The writing of 200000 tweets to the file system took %s seconds' %(endTime
-startTime))
print('The number of operations per second is %s seconds' %(200000/(endTime-
startTime)))
```

From the screenshot below:

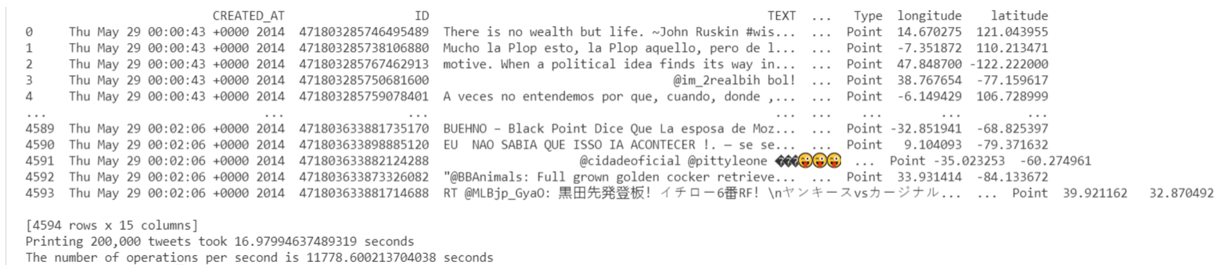


```
The writing of 200000 tweets to the file system took 3.7810802459716797 seconds
The number of operations per second is 52894.93662904345 seconds
PS C:\Users\rejalu1>
```

The speed of writing 200, 000 tweets to the local file system took 3.7 seconds

d) Finally, add a print statement in part 3-a (i.e., print everything you read from the file) and measure the new throughput in MBytes/sec.

```
# Part 3(d)
# Add a final print in 3a and print
# everything you read from the file.
def transformExtraneousValues(fileDictkey):
    """A function that takes a dictionary key and
    checks if the value is null, an empty string or []
    and it replaces it with None otherwise it assigns
    the actual value to a variable which is returned
    """

    valuestr = ''
    if fileDictkey =='null' or fileDictkey =='' or fileDictkey =='[]':
        valuestr = None
    else:
        valuestr = fileDictkey
    return valuestr

newGeoRows = [] # hold individual values of to-be-inserted row
newTweetRows = [] # hold individual values of to-be-inserted row

tweetCounter = 0
geoCounter = 0

startTime = time.time()
for i in range(200000):
    if i % 10000 == 0: # Print a message every 50th tweet read
        print ("Processed " + str(i) + " tweets")
    try:
        if fileItemsL[i]:    # check if the item is empty before hand
            # tweetLine is a byte object which needs to be decoded.
            # the loads() function in the json object lets you convert the string
 into the json object which acts like a dictionary.
            # then decode the line that come back from the web into a string.

            fileDict = json.loads(fileItemsL[i].decode('utf-
8'))    # using decode() and loads to convert each item to a dictionary


            geoV = fileDict['geo']

            NoneType=type(None)


            if geoV: # check if geoV is not null
```



```
                         CREATED_AT                   ID                                              TEXT  ...   Type  longitude    latitude
0     Thu May 29 00:00:43 +0000 2014  471803285746495489  There is no wealth but life. ~John Ruskin #wis...  ...  Point  14.670275  121.043955
1     Thu May 29 00:00:43 +0000 2014  471803285738106880  Mucho la Plop esto, la Plop aquello, pero de l...  ...  Point  -7.351872  110.213471
2     Thu May 29 00:00:43 +0000 2014  471803285767462913  motive. When a political idea finds its way in...  ...  Point  47.848700 -122.222000
3     Thu May 29 00:00:43 +0000 2014  471803285750681600                                  @im_2realbih bol!  ...  Point  38.767654  -77.159617
4     Thu May 29 00:00:43 +0000 2014  471803285759078401  A veces no entendemos por que, cuando, donde ,...  ...  Point  -6.149429  106.728999
...                              ...                 ...                                               ...  ...    ...        ...         ...
4589  Thu May 29 00:02:06 +0000 2014  471803633881735170  BUEHNO - Black Point Dice Que La esposa de Moz...  ...  Point -32.851941  -68.825397
4590  Thu May 29 00:02:06 +0000 2014  471803633898885120  EU  NAO SABIA QUE ISSO IA ACONTECER !. - se se...  ...  Point   9.104093  -79.371632
4591  Thu May 29 00:02:06 +0000 2014  471803633882124288                    @cidadeoficial @pittyleone ���😝😝😝  ...  Point -35.023253  -60.274961
4592  Thu May 29 00:02:06 +0000 2014  471803633873326082  "@BBAnimals: Full grown golden cocker retrieve...  ...  Point  33.931414  -84.133672
4593  Thu May 29 00:02:06 +0000 2014  471803633881714688  RT @MLBjp_GyaO: 黒田先発登板! イチロー6番RF! \nヤンキースvsカージナル...  ...  Point  39.921162   32.870492

[4594 rows x 15 columns]
Printing 200,000 tweets took 16.97994637489319 seconds
The number of operations per second is 11778.600213704038 seconds
```

From the above screenshot, printing everything from the file took 16.97 seconds because I had to decode and convert each item to a dictionary, do some data transformation, derive the expected tuples, which I added to the respective list and used the list to derive a pandas data frame and finally print the contents of the data frame. There is no way this process would take 4 seconds to finish.

<u>Submit a single document containing your written answers. Be sure that this document contains your name and “CSC 555 Assignment 1” at the top.</u>