Mike Fanor
Big Data Programming
Midterm Report

1- All of the lambda functions in my source code serve the purpose of processing the given text files. My approach starts off with using lambda functions to clean the RDD of the encrypted text file to remove punctuation and numbers. I extracted the words and characters from the cleaned RDD using lambda functions and flat maps. I found the word and character frequencies by using the words, characters, a lambda function and various other pyspark functions. I retrieved the most frequent letter observed in the document by using a lambda function within the map function on the character frequency RDD. The kinds of intermediate results that are generated are RDDs, a string from an RDD and integers returned from a python function.

2-

Clean the text by removing punctuation and numbers since those aren't encrypted

```
In [6]: import re

        cleaned = data1\
            .map(lambda x: re.sub(r'[^\D]', ' ', x))\
                .map(lambda y: re.sub(r'[^\w\s]', ' ', y))\
                    .map(lambda z: re.sub('_', '', z))\
                        .map(lambda word: word.strip())

        cleaned.collect()
```

```
Out[6]: ['CNEGVPHYNE CREVBQ BS SYNXVARFF BA VOZ F IARG PBECBENGR ARGJBEX PN',
         'OHG GURER NER VAQRCRAQRAG ERCBEGF BS GUR GREZ SEBZ RYFRJURER',
         '',
         '',
         'ABQR AC    ARKG        AEBSS  CERIVBHF       ABGJBEX  HC         A',
         '',
         'AC   A C  CERS',
         '',
         'RKGERZRYL  HFRQ GB ZBQVSL NQWRPGVIRF QRFPEVOVAT N YRIRY BE DHNYVGL BS',
         'QVSSVPHYGL  GUR PBAABGNGVBA VF BSGRA  ZBER FB GUNA VG FUBHYQ OR  GUVF',
         'VF TRARENYVMRQ SEBZ GUR PBZCHGRE FPVRAPR GREZF  AC UNEQ  NAQ',
         'AC PBZCYRGR   AC PBZCYRGR CEBOYRZF NYY FRRZ GB OR IREL UNEQ  OHG FB',
         'SNE AB BAR UNF SBHAQ N CEBBS GUNG GURL NER  AC VF GUR FRG BS',
         'ABAQRGREZVAVFGVP CBYLABZVNY NYTBEVGUZF  GUBFR GUNG PNA OR PBZCYRGRQ O
        L',
         'N ABAQRGREZVAVFGVP GHEVAT ZNPUVAR VA NA NZBHAG BS GVZR GUNG VF N',
         'CBYLABZVNY SHAPGVBA BS GUR FVMR BS GUR VACHG  N FBYHGVBA SBE BAR',
         'AC PBZCYRGR CEBOYRZ JBHYQ FBYIR NYY GUR BGUREF   PBQVAT N OVGOYG',
         'VZCYRZRAGNGVBA GB CRESBEZ PBEERPGYL VA RIREL PNFR VF AC NAABLVAT',
         '',
```

Test the encryption key by using a random encrypted word and shifting its letters

```
In [20]: import nltk.corpus as nc

         def test_key(encrypted_word, key):
             new_word = ''

             for letter in encrypted_word:
                 shifted_letter = ord(letter) + key

                 if shifted_letter > 90:
                     shifted_letter -= 26
                 new_word += chr(shifted_letter)

             if new_word.lower() in nc.words.words():
                 print(new_word)
```

```
In [21]: random_word = words1.sample(withReplacement=True, fraction=0.0002)
         random_word.first()

Out[21]: 'NG'
```

```
In [22]: test_key(random_word.first(), encryption_key)

         AT
```

```
In [28]: print(
             decrypt_file('Encrypted-1.txt')
         )

PARTICULAR PERIOD OF FLAKINESS ON IBM'S VNET CORPORATE NETWORK CA.
1988; BUT THERE ARE INDEPENDENT REPORTS OF THE TERM FROM ELSEWHERE.
________________________________________________________________________

NODE:NP-, NEXT:[9529]NROFF, PREVIOUS:[9530]NOTWORK, UP:[9531]= N =

NP- /N-P/ PREF.

EXTREMELY. USED TO MODIFY ADJECTIVES DESCRIBING A LEVEL OR QUALITY OF
DIFFICULTY; THE CONNOTATION IS OFTEN `MORE SO THAN IT SHOULD BE' THIS
IS GENERALIZED FROM THE COMPUTER-SCIENCE TERMS `NP-HARD' AND
`NP-COMPLETE'; NP-COMPLETE PROBLEMS ALL SEEM TO BE VERY HARD, BUT SO
FAR NO ONE HAS FOUND A PROOF THAT THEY ARE. NP IS THE SET OF
NONDETERMINISTIC-POLYNOMIAL ALGORITHMS, THOSE THAT CAN BE COMPLETED BY
A NONDETERMINISTIC TURING MACHINE IN AN AMOUNT OF TIME THAT IS A
POLYNOMIAL FUNCTION OF THE SIZE OF THE INPUT; A SOLUTION FOR ONE
NP-COMPLETE PROBLEM WOULD SOLVE ALL THE OTHERS. "CODING A BITBLT
IMPLEMENTATION TO PERFORM CORRECTLY IN EVERY CASE IS NP-ANNOYING."

NOTE, HOWEVER, THAT STRICTLY SPEAKING THIS USAGE IS MISLEADING; THERE
```

Assuming the most common encrypted letter decrypts to 'E'

```
In [17]: most_freq_letter = chars1freq.map(lambda x: x[0].upper()).first()

In [18]: def find_key(letter):
             return 26 + (ord('E') - ord(letter))

In [19]: encryption_key = find_key(most_freq_letter)
         encryption_key

Out[19]: 13
```

I started with the assumption that the most frequently occurring letter in the documents decrypted to the letter ‘E’, the most frequently occurring letter in the English language. Using a python function, I found the numerical value that would encrypt ‘E’ to the most frequently occurring letter in the documents. Using that value, I tested it against a randomly-chosen encrypted word from the document using another python function. I validated the decrypted word using the nltk corpus. I found that the encryption key found from my assumption turned out to be the correct one after the decrypted form of the randomly-chosen encrypted word was validated. My implementation did in fact give the exact Caesar Cypher values.