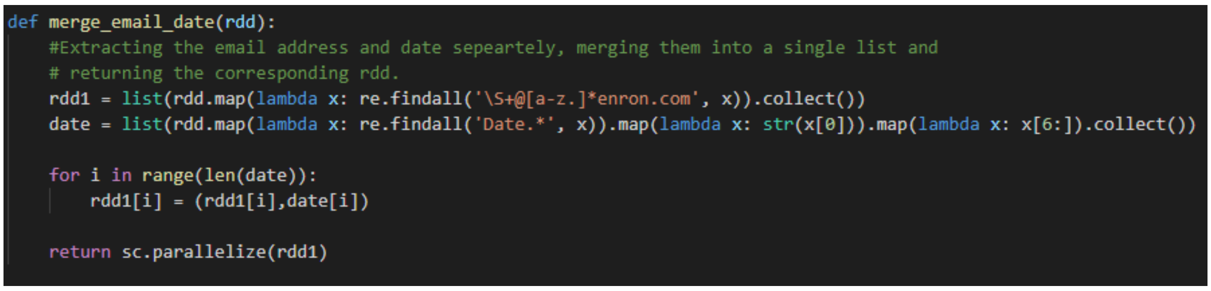
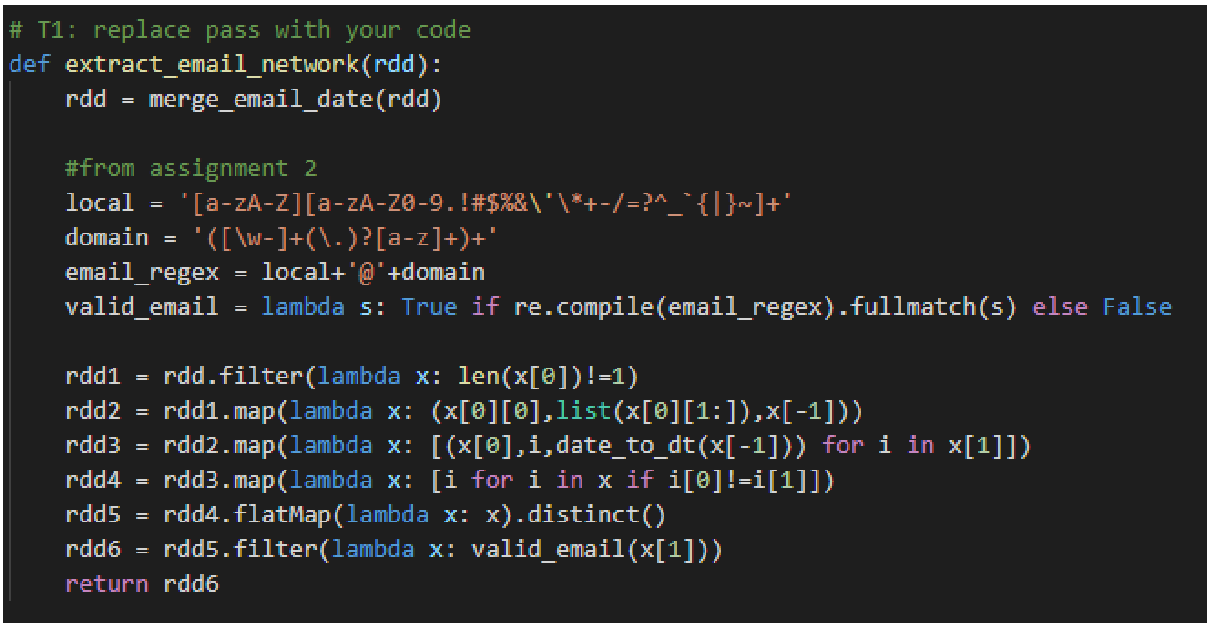
CS5234J Final Project Report

Question 1



```python
def merge_email_date(rdd):
    #Extracting the email address and date seepartely, merging them into a single list and
    # returning the corresponding rdd.
    rdd1 = list(rdd.map(lambda x: re.findall('\S+@[a-z.]*enron.com', x)).collect())
    date = list(rdd.map(lambda x: re.findall('Date.*', x)).map(lambda x: str(x[0])).map(lambda x: x[6:]).collect())

    for i in range(len(date)):
        rdd1[i] = (rdd1[i],date[i])

    return sc.parallelize(rdd1)
```

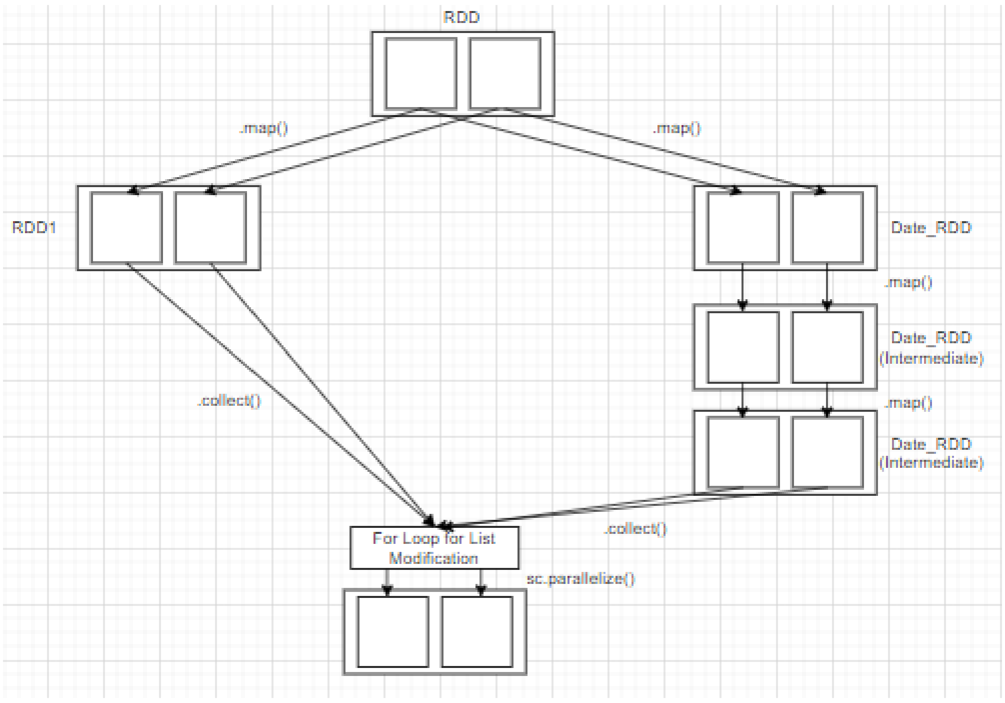


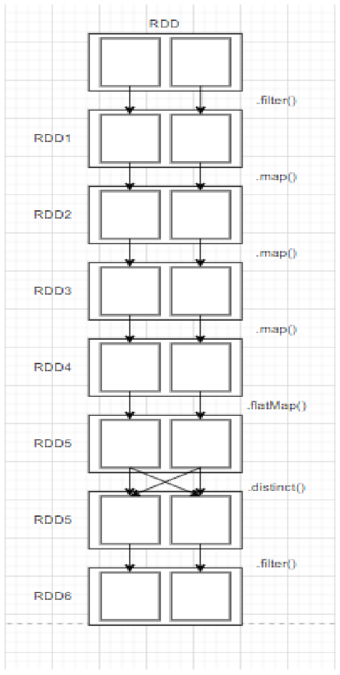
```python
# T1: replace pass with your code
def extract_email_network(rdd):
    rdd = merge_email_date(rdd)

    #from assignment 2
    local = '[a-zA-Z][a-zA-Z0-9.!#$%&\'\*+-/=?^_`{|}~]+'
    domain = '([\w-]+(\.)?[a-z]+)+'
    email_regex = local+'@'+domain
    valid_email = lambda s: True if re.compile(email_regex).fullmatch(s) else False

    rdd1 = rdd.filter(lambda x: len(x[0])!=1)
    rdd2 = rdd1.map(lambda x: (x[0][0],list(x[0][1:]),x[-1]))
    rdd3 = rdd2.map(lambda x: [(x[0],i,date_to_dt(x[-1])) for i in x[1]])
    rdd4 = rdd3.map(lambda x: [i for i in x if i[0]!=i[1]])
    rdd5 = rdd4.flatMap(lambda x: x).distinct()
    rdd6 = rdd5.filter(lambda x: valid_email(x[1]))
    return rdd6
```

For the first question, we have split it into 2 parts. First part takes care of extracting the email address and the date. This is taken care in the function 'merge_email_date'. It takes as argument the rdd corresponding to enron.seq and returns a rdd with is of the form (emai_address, date). In order to merge we first collect the email address and date individual using regex. Once we have it in the list format we merge it in form of list of tuple and then parallelize it.

The challenge faced in this snippet was to validate the email address, we realized this while coding for further task and had to come back and make the regex string more robust in order to filter some unwanted email address which were seen in tasks 2 and 3.

By far the biggest challenge was to put the email address and date together, because once we have that we could easily use spark transformations to modify or filter out the samples. In order to this we performed 2 early collect operations, merged the resulting list in the required fashion and converted the resulting list back to rdd form.





Above I have shown the linear graph of Task1, part one is for the function 'merge_email_date'. Here, there are 2 stages, the first is for RDD1 creation from the base 'RDD' (in-memory) until collect action is performed and the second stage is the one from base RDD to .collect() for through Date_RDD. For the second half again we have 2 stages, one is from base RDD till RDD6, we perform various transformations between these steps. Finally when we call distinct on RDD6 to get rid of the duplicated a new stage is created which ends with an action called on it.

1.3) We have various narror dependencies in the task like map ,filter and flatMap. A example for this is the creation of RDD4 from RDD3 where we use map to remove the samples where the sender and receiver are the same using a lambda function within a for loop.

1.4) There is one wide dependency which is ‘distinct’ used on RDD6. Wide dependency are computational heavy and are to be avoided as the data needs to be shuffled between the partitons which may or may not be on the same memory hence data might have to be moved through the network. It also leads to creation of a new stage.

1.5) In this case we will consider 2nd half of the problem.

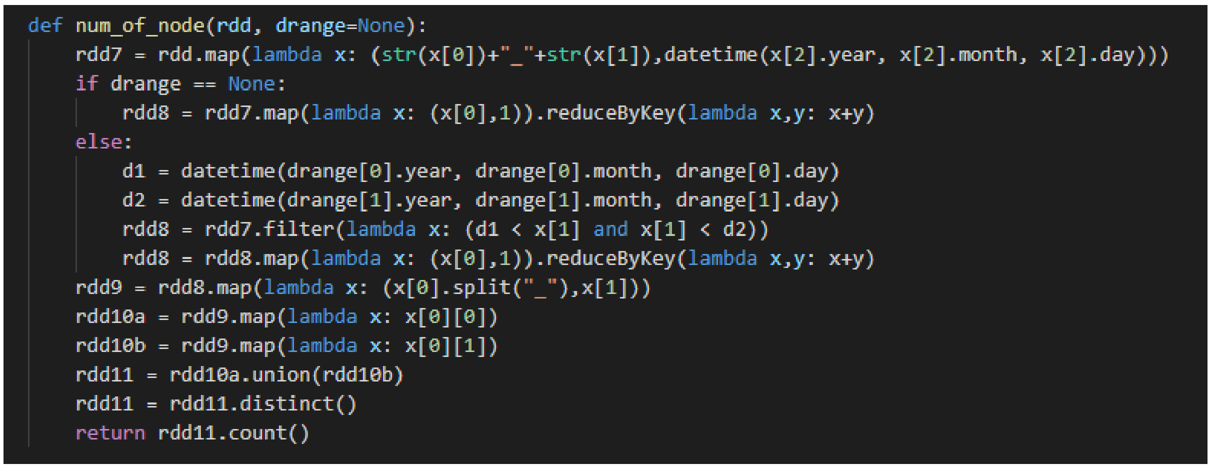
We have 2 stages, stage one has 5 tasks, in order they are

Filter -> Map -> Map-> Map-> FlatMap

Stage two is from RDD6 until the termination of the function (returning the resulting RDD to calling function). It has 2 tasks namely -

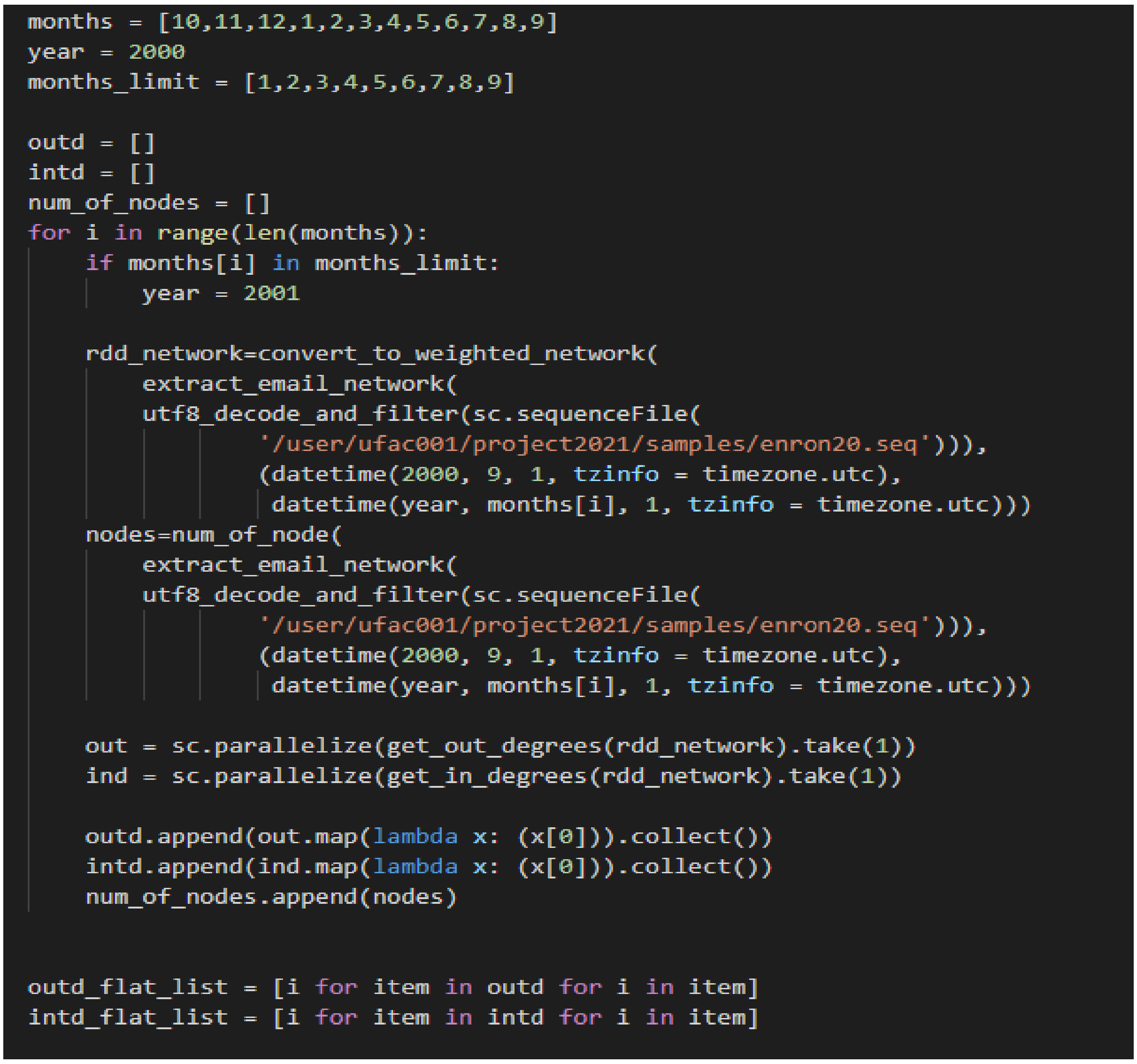
Distinct -> Filter

Question 2



```python
def num_of_node(rdd, drange=None):
    rdd7 = rdd.map(lambda x: (str(x[0])+"_"+str(x[1]),datetime(x[2].year, x[2].month, x[2].day)))
    if drange == None:
        rdd8 = rdd7.map(lambda x: (x[0],1)).reduceByKey(lambda x,y: x+y)
    else:
        d1 = datetime(drange[0].year, drange[0].month, drange[0].day)
        d2 = datetime(drange[1].year, drange[1].month, drange[1].day)
        rdd8 = rdd7.filter(lambda x: (d1 < x[1] and x[1] < d2))
        rdd8 = rdd8.map(lambda x: (x[0],1)).reduceByKey(lambda x,y: x+y)
    rdd9 = rdd8.map(lambda x: (x[0].split("_"),x[1]))
    rdd10a = rdd9.map(lambda x: x[0][0])
    rdd10b = rdd9.map(lambda x: x[0][1])
    rdd11 = rdd10a.union(rdd10b)
    rdd11 = rdd11.distinct()
    return rdd11.count()
```

Fig 2.1



```python
months = [10,11,12,1,2,3,4,5,6,7,8,9]
year = 2000
months_limit = [1,2,3,4,5,6,7,8,9]

outd = []
intd = []
num_of_nodes = []
for i in range(len(months)):
    if months[i] in months_limit:
        year = 2001

    rdd_network=convert_to_weighted_network(
        extract_email_network(
        utf8_decode_and_filter(sc.sequenceFile(
                '/user/ufac001/project2021/samples/enron20.seq'))),
                (datetime(2000, 9, 1, tzinfo = timezone.utc),
                 datetime(year, months[i], 1, tzinfo = timezone.utc)))
    nodes=num_of_node(
        extract_email_network(
        utf8_decode_and_filter(sc.sequenceFile(
                '/user/ufac001/project2021/samples/enron20.seq'))),
                (datetime(2000, 9, 1, tzinfo = timezone.utc),
                 datetime(year, months[i], 1, tzinfo = timezone.utc)))

    out = sc.parallelize(get_out_degrees(rdd_network).take(1))
    ind = sc.parallelize(get_in_degrees(rdd_network).take(1))

    outd.append(out.map(lambda x: (x[0])).collect())
    intd.append(ind.map(lambda x: (x[0])).collect())
    num_of_nodes.append(nodes)


outd_flat_list = [i for item in outd for i in item]
intd_flat_list = [i for item in intd for i in item]
```

Fig 2.2

```
In [57]: %matplotlib inline
         import matplotlib.pyplot as plt
         import numpy as np

         fig = plt.figure()
         ax = plt.axes()
         ax.plot(num_of_nodes,outd_flat_list)
         ax.plot(num_of_nodes,intd_flat_list)
         plt.ylabel("kmax")
         plt.xlabel("No. of Nodes")
         plt.title("Visualisation of kmax v No. of Nodes")
         plt.legend("OI")

Out[57]: <matplotlib.legend.Legend at 0x7f1b00a96320>
```

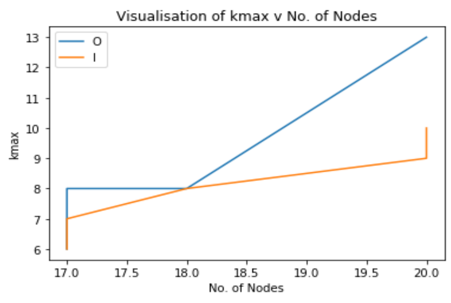
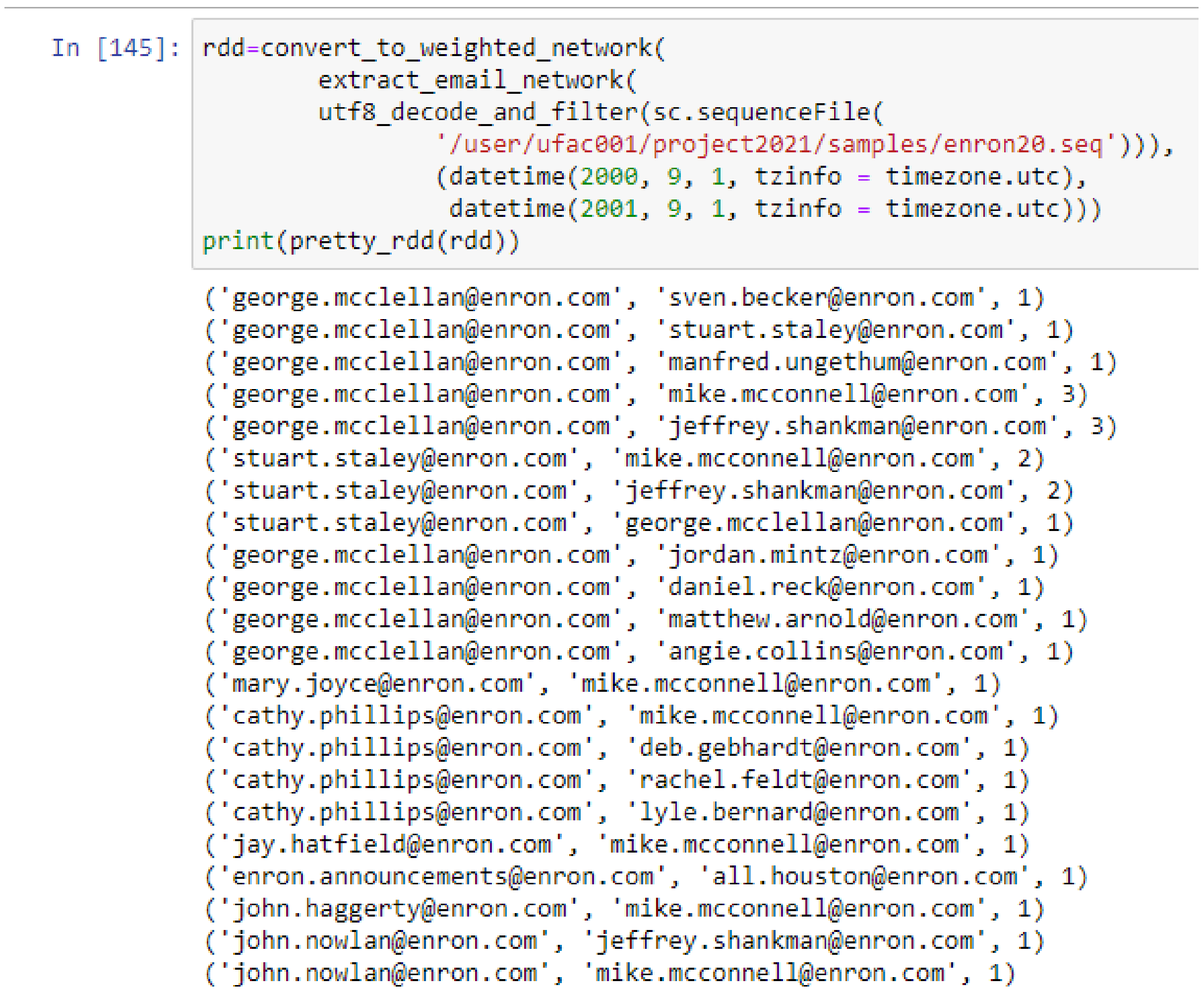


Fig 2.3

The 12 month range in consideration for this experiment is September 2000 to September 2001.



```
In [145]: rdd=convert_to_weighted_network(
                  extract_email_network(
                  utf8_decode_and_filter(sc.sequenceFile(
                          '/user/ufac001/project2021/samples/enron20.seq'))),
                          (datetime(2000, 9, 1, tzinfo = timezone.utc),
                           datetime(2001, 9, 1, tzinfo = timezone.utc)))
          print(pretty_rdd(rdd))

          ('george.mcclellan@enron.com', 'sven.becker@enron.com', 1)
          ('george.mcclellan@enron.com', 'stuart.staley@enron.com', 1)
          ('george.mcclellan@enron.com', 'manfred.ungethum@enron.com', 1)
          ('george.mcclellan@enron.com', 'mike.mcconnell@enron.com', 3)
          ('george.mcclellan@enron.com', 'jeffrey.shankman@enron.com', 3)
          ('stuart.staley@enron.com', 'mike.mcconnell@enron.com', 2)
          ('stuart.staley@enron.com', 'jeffrey.shankman@enron.com', 2)
          ('stuart.staley@enron.com', 'george.mcclellan@enron.com', 1)
          ('george.mcclellan@enron.com', 'jordan.mintz@enron.com', 1)
          ('george.mcclellan@enron.com', 'daniel.reck@enron.com', 1)
          ('george.mcclellan@enron.com', 'matthew.arnold@enron.com', 1)
          ('george.mcclellan@enron.com', 'angie.collins@enron.com', 1)
          ('mary.joyce@enron.com', 'mike.mcconnell@enron.com', 1)
          ('cathy.phillips@enron.com', 'mike.mcconnell@enron.com', 1)
          ('cathy.phillips@enron.com', 'deb.gebhardt@enron.com', 1)
          ('cathy.phillips@enron.com', 'rachel.feldt@enron.com', 1)
          ('cathy.phillips@enron.com', 'lyle.bernard@enron.com', 1)
          ('jay.hatfield@enron.com', 'mike.mcconnell@enron.com', 1)
          ('enron.announcements@enron.com', 'all.houston@enron.com', 1)
          ('john.haggerty@enron.com', 'mike.mcconnell@enron.com', 1)
          ('john.nowlan@enron.com', 'jeffrey.shankman@enron.com', 1)
          ('john.nowlan@enron.com', 'mike.mcconnell@enron.com', 1)
```

From the above result we can see there are 28 edges and 20 nodes. According to the 80/20 rule, 20% of the 20 nodes should be part of (in or out) 80% of the edges. Let us put this hypothesis to test and if it holds in out timeframe.

The top 20% (4) nodes are George, Mike, Stuart and Cathy. They should form the nodes of at least 80% of the edges which we see that they do. Hence, we can say that for out time the 80/20 rule does hold.

2.2) For the second part of question 2, we have a small code snippet and shown and submitted in project.py file. You can see in Fig 2.2 on how we performed a roll-over onto the new year (ie 2001). We kept the start date constant and kept increasing the width of the date range by one month in every iteration, to get more edges (emails between nodes).

We use the function implemented in task 3 to get the kmax values and write a modified function to get the number of nodes in the time frame. We then extract only the values relevant to us and plot a line graph as shown in Fig 2.3. We can see the the graph is slightly linear, I am even inclined to stay that it may even be exponential if were to add the in and out Kmaxes.