Enron email data set exploration

In [1]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import networkx as nx
```

In [2]:
```python
# Get better looking pictures
%config InlineBackend.figure_format = 'retina'
```

In [3]:
```python
df = pd.read_feather('enron.feather')
df = df.sort_values(['Date'])
#df
```

Email traffic over time

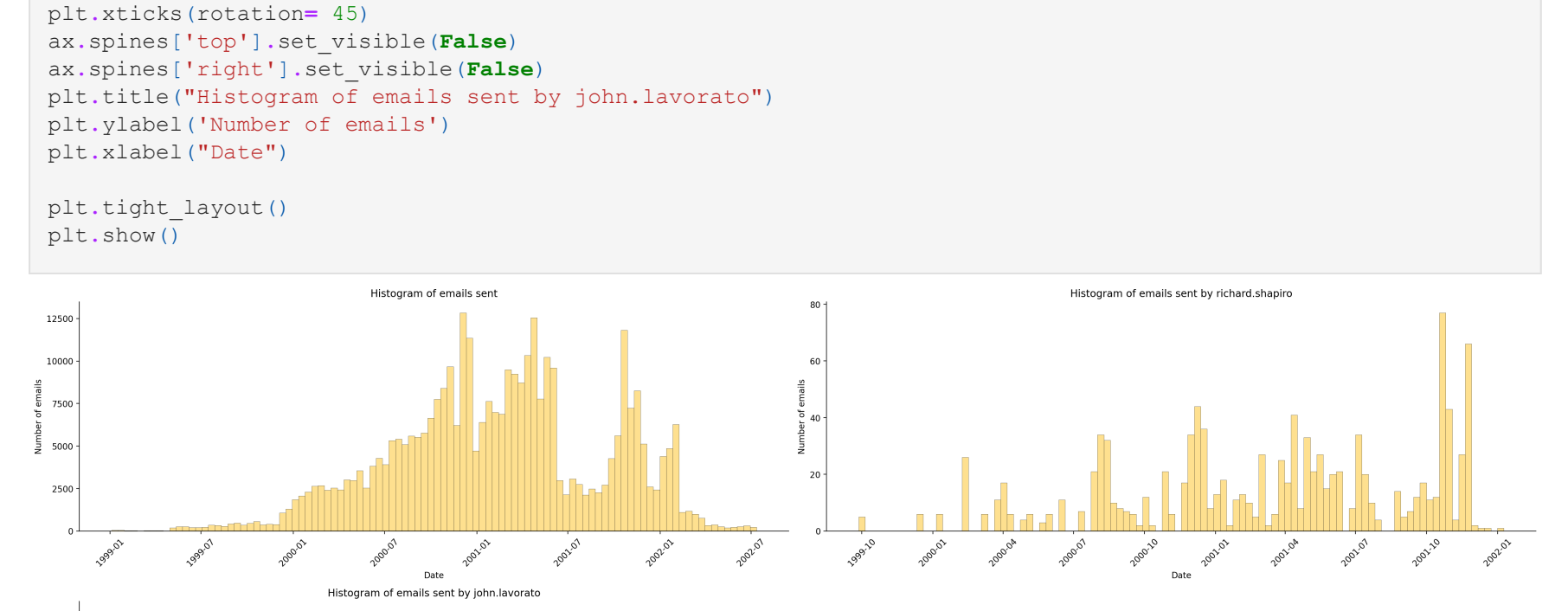
Group the data set by `Date` and `MailID`, which will get you an index that collects all of the unique mail IDs per date. Then reset the index so that those date and mail identifiers become columns and then select for just those columns; we don't actually care about the counts created by the `groupby` (that was just to get the index). Create a histogram that shows the amount of traffic per day. Then specifically for email sent from `richard.shapiro` and then `john.lavorato`. Because some dates are set improperly (to 1980), filter for dates greater than January 1, 1999.

In [4]:
```python
etot = df.groupby(['Date','MailID']).count().reset_index()
etot = etot[['Date','MailID']]
etot['Date']  = pd.to_datetime(etot['Date'])
etot = etot[etot['Date']>='1999-01-01']
#etot
```

In [5]:
```python
rsdf = df.query("From=='richard.shapiro' and Date >='1999-01-01'").groupby(['Date','MailID']).count().reset_inc
rsdf= rsdf[['Date','MailID']]
rsdf['Date']  = pd.to_datetime(rsdf['Date'])
#rsdf
```

In [6]:
```python
jldf = df.query("From=='john.lavorato' and Date >='1999-01-01'").groupby(['Date','MailID']).count().reset_index
jldf= jldf[['Date','MailID']]
jldf['Date']  = pd.to_datetime(jldf['Date'])
#jldf
```

In [7]:
```python
yr =['1999-01-01','1999-07-01','2000-01-01','2000-07-01','2001-01-01','2001-07-01','2002-01-01','2002-07-01']
count = [0,2500,5000,7500,10000,12500]

plt.figure(figsize=(25,10))

ax = plt.subplot(221)
ax.hist(data = etot, x= 'Date',bins =100,color = '#fee08f',edgecolor ='black',lw =.2)
plt.yticks(count,count)
plt.xticks(rotation= 45)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
plt.title("Histogram of emails sent")
plt.ylabel('Number of emails')
plt.xlabel("Date")

ax = plt.subplot(222)
ax.hist(data = rsdf, x= 'Date',bins =100,color = '#fee08f',edgecolor ='black',lw =.2)
plt.yticks([0,20,40,60,80])
plt.xticks(rotation= 45)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
plt.title("Histogram of emails sent by richard.shapiro")
plt.ylabel('Number of emails')
plt.xlabel("Date")

ax = plt.subplot(223)
ax.hist(data = jldf, x= 'Date',bins =100,color = '#fee08f',edgecolor ='black',lw =.2)
plt.yticks([0,20,40,60,80])
plt.xticks(rotation= 45)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
plt.title("Histogram of emails sent by john.lavorato")
plt.ylabel('Number of emails')
plt.xlabel("Date")

plt.tight_layout()
plt.show()
```



Received emails

Count the number of messages received per user and then sort in reverse order. Make a bar chart showing the top 30 email recipients.

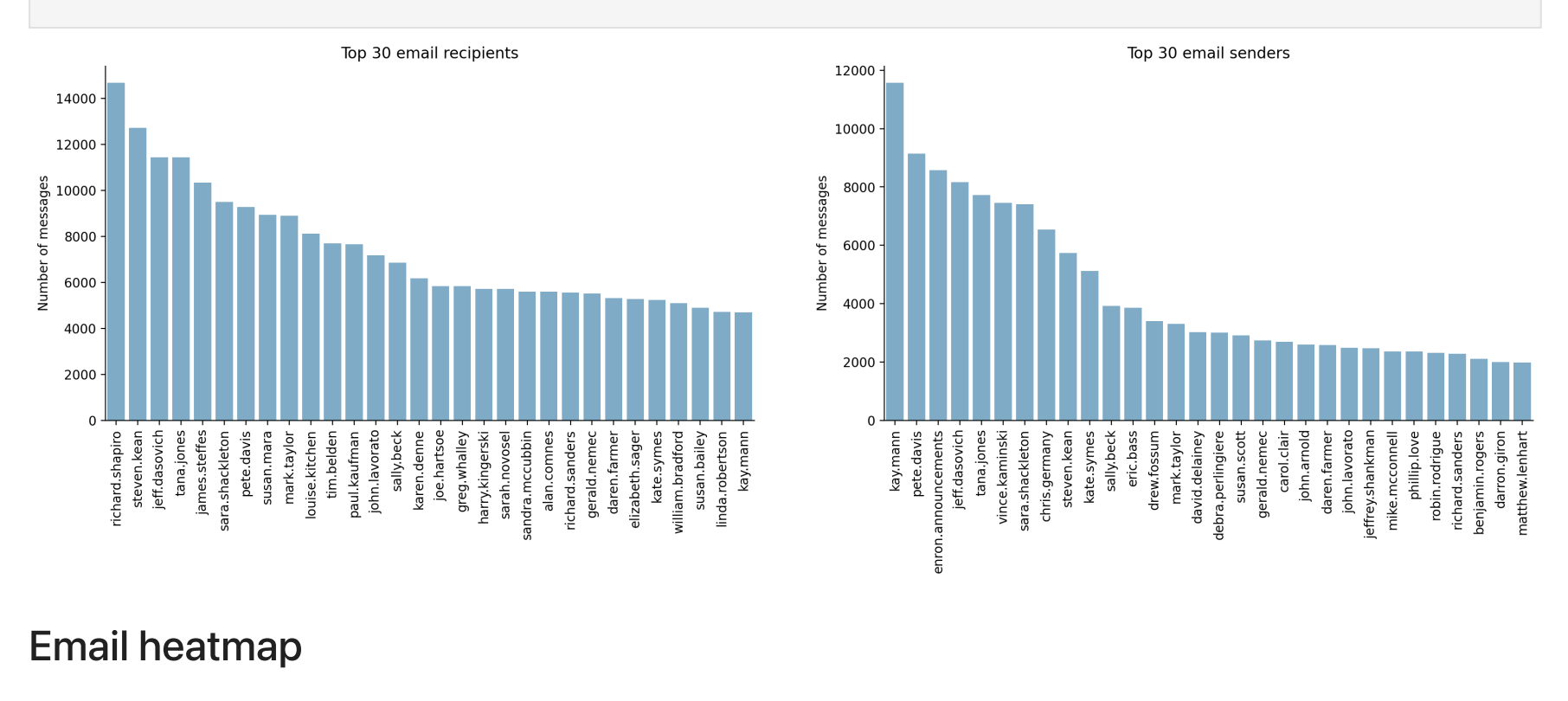
Sent emails

Make a bar chart indicating the top 30 mail senders. This is more complicated than the received emails because a single person can email multiple people in a single email. So, group by `From` and `MailID`, convert the index back to columns and then group again by `From` and get the count.

In [8]:
```python
recieve = df.groupby("To")["MailID"].count().sort_values()[::-1][:30]
#recieve
```

In [9]:
```python
sender = df.groupby(["From",'MailID']).count().reset_index().groupby("From")['MailID'].count()
sender = sender.sort_values()[::-1][:30]
#sender
```

In [10]:
```python
plt.figure(figsize=(20,5))

plt.subplot(121)
ax = sns.barplot(x = recieve.index,y = recieve.values, color = '#73aed2')
plt.xticks(rotation=90)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
plt.title("Top 30 email recipients")
plt.ylabel('Number of messages')
plt.xlabel(None)

plt.subplot(122)
ax = sns.barplot(x = sender.index,y = sender.values, color = '#73aed2')
plt.xticks(rotation=90)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
plt.title("Top 30 email senders")
plt.ylabel('Number of messages')
plt.xlabel(None)
plt.show();
```



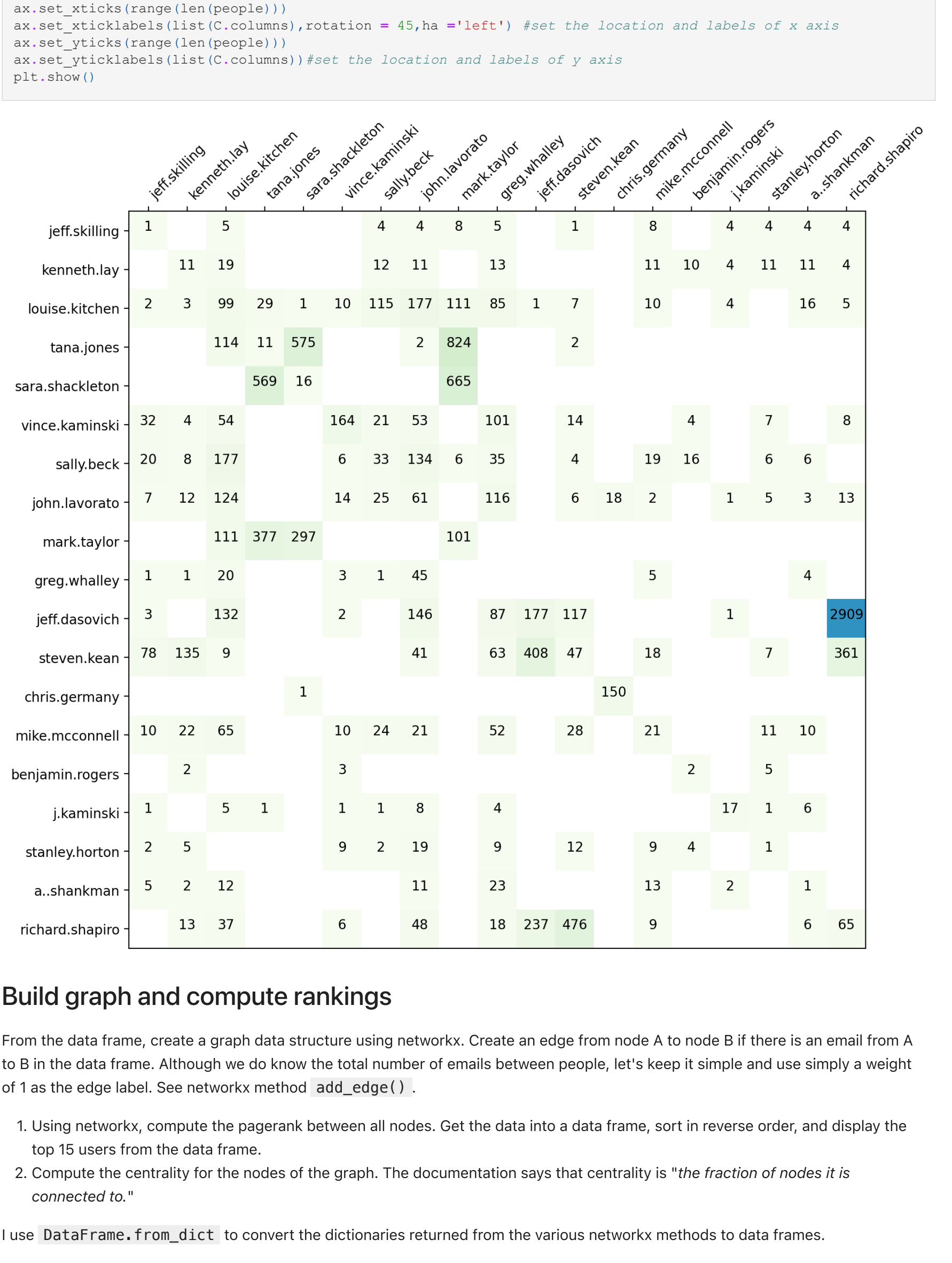
Email heatmap

Given a list of Enron employees, compute a heat map that indicates how much email traffic went between each pair of employees. The heat map is not symmetric because Susan sending mail to Xue is not the same thing as Xue sending mail to Susan. The first step is to group the data frame by `From` and `To` columns in order to get the number of emails from person i to person j. Then, create a 2D numpy matrix, C, of integers and set $C_{i,j}$ to the count of person i to person j. Using matplotlib, `ax.imshow(C, cmap='GnBu', vmax=4000)`, show the heat map and add tick labels at 45 degrees for the X axis. Set the labels to the appropriate names. Draw the number of emails in the appropriate cells of the heat map, for all values greater than zero. Please note that when you draw text using `ax.text()`, the coordinates are X,Y whereas the coordinates in the C matrix are row,column so you will have to flip the coordinates.

In [11]:
```python
people = ['jeff.skilling', 'kenneth.lay', 'louise.kitchen', 'tana.jones',
          'sara.shackleton', 'vince.kaminski', 'sally.beck', 'john.lavorato',
          'mark.taylor', 'greg.whalley', 'jeff.dasovich', 'steven.kean',
          'chris.germany', 'mike.mcconnell', 'benjamin.rogers', 'j.kaminski',
          'stanley.horton', 'a..shankman', 'richard.shapiro']
```

In [12]:
```python
eh = df[(df['From'].isin(people)) & (df["To"].isin(people))]
C = eh.groupby(['From','To'])['MailID'].count().unstack().loc[people,people]
#C
```

In [13]:
```python
f,ax = plt.subplots(figsize = (10,10))
ax.imshow(C, cmap='GnBu', vmax=4000)

# Add correlation as text to each box, with 0 decimal places
for i in range(len(people)):
    for j in range(len(people)):
        if C.iloc[j,i]>0:
            ax.text(i, j, f"{C.iloc[j,i]:.0f}", horizontalalignment='center')

ax.xaxis.tick_top()
ax.set_xticks(range(len(people)))
ax.set_xticklabels(list(C.columns),rotation = 45,ha ='left') #set the location and labels of x axis
ax.set_yticks(range(len(people)))
ax.set_yticklabels(list(C.columns))#set the location and labels of y axis
plt.show()
```



Build graph and compute rankings

From the data frame, create a graph data structure using networkx. Create an edge from node A to node B if there is an email from A to B in the data frame. Although we do know the total number of emails between people, let's keep it simple and use simply a weight of 1 as the edge label. See networkx method `add_edge()`.

1. Using networkx, compute the pagerank between all nodes. Get the data into a data frame, sort in reverse order, and display the top 15 users from the data frame.
2. Compute the centrality for the nodes of the graph. The documentation says that centrality is "*the fraction of nodes it is connected to.*"

I use `DataFrame.from_dict` to convert the dictionaries returned from the various networkx methods to data frames.

Node PageRank

In [14]:
```python
G = nx.DiGraph()
for f, t in zip(df['From'],df["To"]):
    G.add_edge(f,t, weight=1)
```

In [15]:
```python
pk = pd.DataFrame.from_dict(nx.pagerank(G), orient = 'index',columns=['PageRank']).sort_values(ascending=False,
pk = pk[:15]
pk
```

Out[15]:

	PageRank
jeff.skilling	0.004966
kenneth.lay	0.004579
louise.kitchen	0.004494
tana.jones	0.004299
sara.shackleton	0.004022
vince.kaminski	0.003850
sally.beck	0.003577
john.lavorato	0.003467
gerald.nemec	0.002643
rod.hayslett	0.002540
mark.taylor	0.002514
greg.whalley	0.002280
jeff.dasovich	0.002264
daren.farmer	0.002007
steven.kean	0.002003

Centrality

In [16]:
```python
cen = pd.DataFrame.from_dict(nx.degree_centrality(G), orient = 'index',columns=['PageRank']).sort_values(ascenc
cen = cen[:15]
cen
```

Out[16]:

	PageRank
sally.beck	0.088048
outlook.team	0.082401
david.forster	0.079988
kenneth.lay	0.076137
technology.enron	0.063662
jeff.skilling	0.058220
tana.jones	0.054677
louise.kitchen	0.053445
jeff.dasovich	0.048773
sara.shackleton	0.047952
tracey.kozadinos	0.047387
john.lavorato	0.046822
julie.clyatt	0.045847
bodyshop	0.044922
david.oxley	0.043793

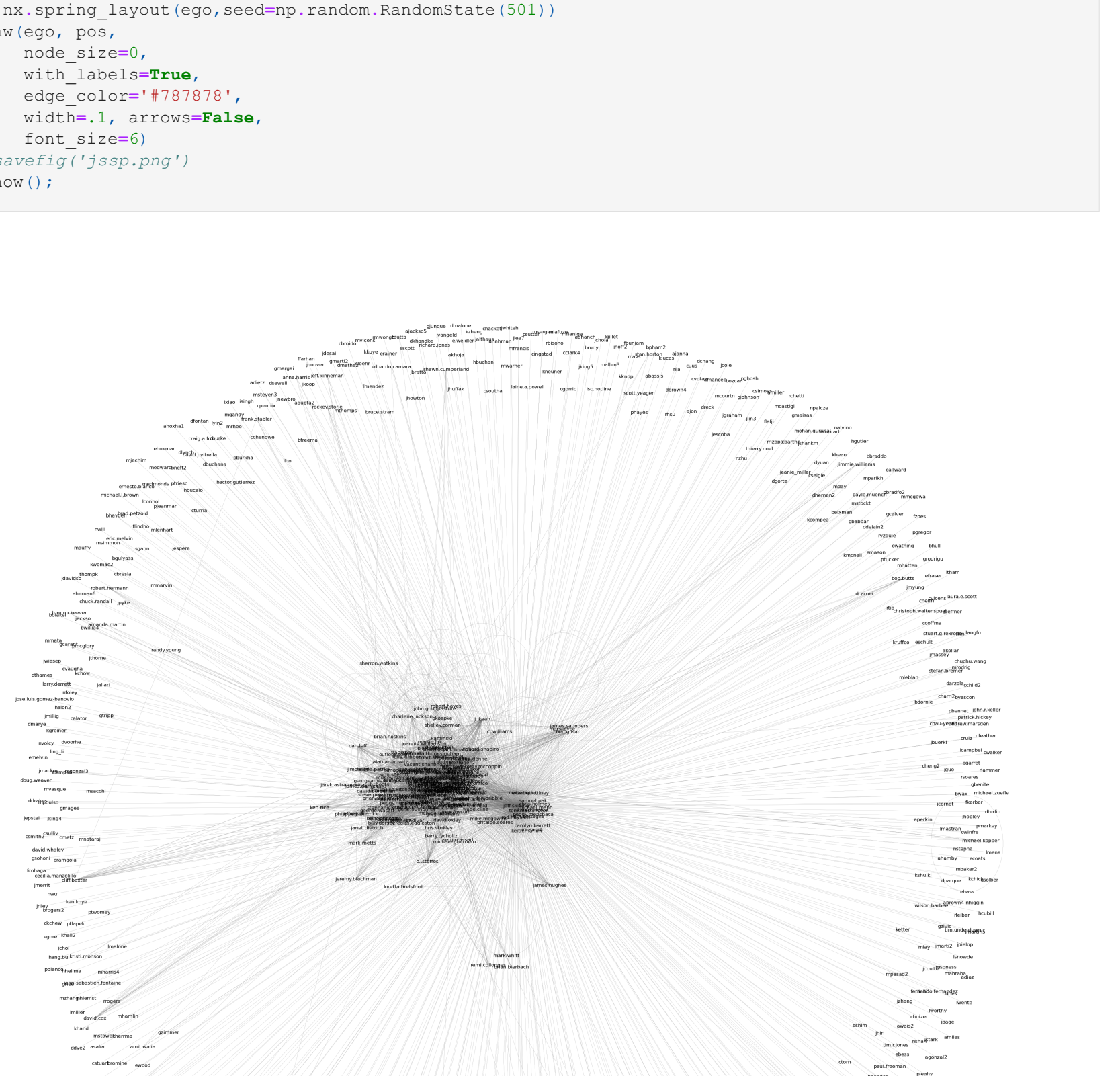
Plotting graph subsets

The email graph is way too large to display the whole thing and get any meaningful information out. However, we can look at subsets of the graph such as the neighbors of a specific node. To visualize it we can use different strategies to layout the nodes. In this case, we will use two different layout strategies: *spring* and *kamada-kawai*. According to Wikipedia, these force directed layout strategies have the characteristic: "...*the edges tend to have uniform length (because of the spring forces), and nodes that are not connected by an edge tend to be drawn further apart*...".

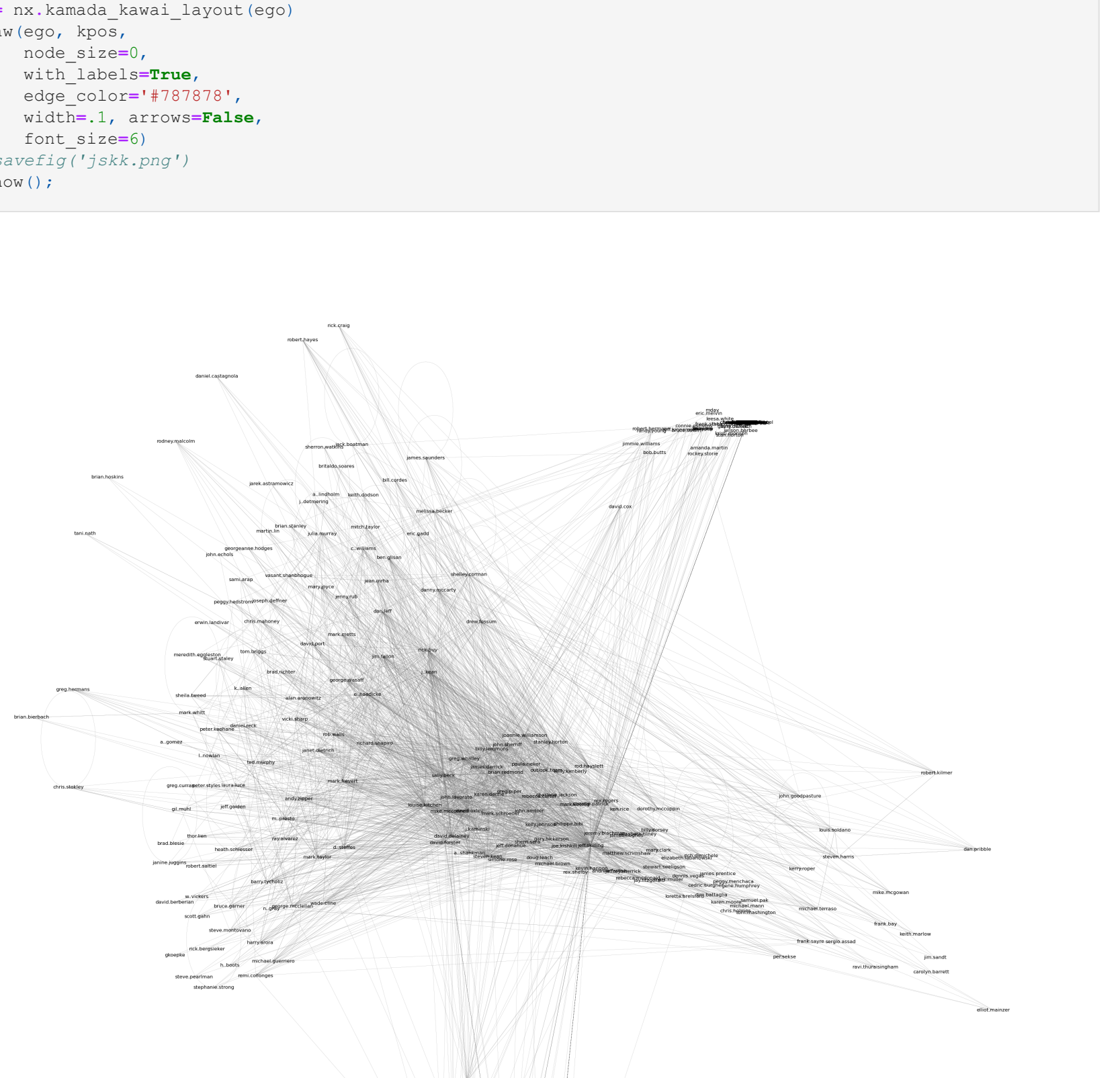
Use networkx `ego_graph()` method to get a radius=1 neighborhood around `jeff.skilling` and draw the spring graph with a plot that is 20x20 inch so we can see details. Then, draw the same subgraph again using the kamada-kawai layout strategy. Finally, get the neighborhood around kenneth.lay and draw kamada-kawai.

spring_layout G object

In [17]:
```python
plt.figure(figsize=(20,20))

ego = nx.ego_graph(G,'jeff.skilling')
pos = nx.spring_layout(ego,seed=np.random.RandomState(501))
nx.draw(ego, pos,
        node_size=0,
        with_labels=True,
        edge_color='#787878',
        width=.1, arrows=False,
        font_size=6)
#plt.savefig('jssp.png')
plt.show();
```



In [18]:
```python
plt.figure(figsize=(20,20))

ego = nx.ego_graph(G,'jeff.skilling')
kpos = nx.kamada_kawai_layout(ego)
nx.draw(ego, kpos,
        node_size=0,
        with_labels=True,
        edge_color='#787878',
        width=.1, arrows=False,
        font_size=6)
#plt.savefig('jskk.png')
plt.show();
```



In [19]:
```python
plt.figure(figsize=(20,20))

ego = nx.ego_graph(G,'kenneth.lay')
kpos = nx.kamada_kawai_layout(ego)
nx.draw(ego, kpos,
        node_size=0,
        with_labels=True,
        edge_color='#787878',
        width=.1, arrows=False,
        font_size=6)
#plt.savefig('klkk.png')
plt.show();
```