

Enron Email Analysis

Contributors:

Hardik Chodvadiya

Jay Gala

Mihir Gandhi

Jasdeep Singh Grover

Sanjana Joshi

Problem Statement:

The Enron scandal and collapse was one of the largest corporate meltdowns in history. In the year 2000, Enron was one of the largest energy companies in America. Then, after being outed for fraud, it spiraled downward into bankruptcy within a year. Luckily for us, we have the Enron email database. It contains 500 thousand emails between 150 former Enron employees, mostly senior executives. It's also the only large public database of real emails, which makes it more valuable. In fact, data scientists have been using this dataset for education and research for years. Perform the following tasks:

1. Anomaly detection: Map the distribution of emails sent and received by hour and try to detect abnormal behavior leading up to the public scandal.
2. Social analysis: Build a model for communication between employees to find key influencers.
3. Email Analysis: Analyze the body messages in conjunction with email metadata to classify emails based on their purposes

Data Sources:

Enron Email Dataset – This is the Enron email archive hosted by CMU.

Description of Enron Data (PDF) – Exploratory analysis of Enron email data that could help you get your grounding.

<https://www.cs.cmu.edu/~enron/>

Introduction

Enron Scandal

The **Enron scandal**, publicized in October 2001, eventually led to the bankruptcy of the Enron Corporation, an American energy company based in Houston, Texas, and the *de facto* dissolution of Arthur Andersen, which was one of the five largest audit and accountancy partnerships in the world. In addition to being the largest bankruptcy reorganization in American history at that time, Enron was cited as the biggest audit failure.

Enron was formed in 1985 by Kenneth Lay after merging Houston Natural Gas and InterNorth. Several years later, when Jeffrey Skilling was hired, he developed a staff of executives that – by the use of accounting loopholes, special purpose entities, and poor financial reporting – were able to hide billions of dollars in debt from failed deals and projects. Chief Financial Officer Andrew Fastow and other executives not only misled Enron's board of directors and audit committee on high-risk accounting practices, but also pressured Arthur Andersen to ignore the issues.

Enron shareholders filed a \$40 billion lawsuit after the company's stock price, which achieved a high of US\$90.75 per share in mid-2000, plummeted to less than \$1 by the end of November 2001.^[2] The U.S. Securities and Exchange Commission (SEC) began an investigation, and rival Houston competitor Dynegy offered to purchase the company at a very low price. The deal failed, and on December 2, 2001, Enron filed for bankruptcy under Chapter 11 of the United States Bankruptcy Code. Enron's \$63.4 billion in assets made it the largest corporate bankruptcy in U.S. history until WorldCom's bankruptcy the next year.

Many executives at Enron were indicted for a variety of charges and some were later sentenced to prison. Arthur Andersen was found guilty of illegally destroying documents relevant to the SEC investigation, which voided its license to audit public companies and effectively closed the firm. By the time the ruling was overturned at the U.S. Supreme Court, the company had lost the majority of its customers and had ceased operating. Enron employees and shareholders received limited returns in lawsuits, despite losing billions in pensions and stock prices.

As a consequence of the scandal, new regulations and legislation were enacted to expand the accuracy of financial reporting for public companies. One piece of legislation, the Sarbanes–Oxley Act, increased penalties for destroying, altering, or fabricating records in federal investigations or for attempting to defraud shareholders.

The act also increased the accountability of auditing firms to remain unbiased and independent of their clients.

Enron Email Dataset

The Enron email dataset contains approximately 500,000 emails generated by employees of the Enron Corporation. It was obtained by the Federal Energy Regulatory Commission during its investigation of Enron's collapse.

This is the May 7, 2015 Version of dataset, as published at <https://www.cs.cmu.edu/~.enron/>

Anomaly Detection

In data mining, **anomaly detection** (also **outlier detection**) is the identification of rare items, events or observations which raise suspicions by differing significantly from the majority of the data. Typically the anomalous items will translate to some kind of problem such as bank fraud, a structural defect, medical problems or errors in a text. Anomalies are also referred to as outliers, novelties, noise, deviations and exceptions.

In particular, in the context of abuse and network intrusion detection, the interesting objects are often not *rare* objects, but unexpected *bursts* in activity. This pattern does not adhere to the common statistical definition of an outlier as a rare object, and many outlier detection methods (in particular unsupervised methods) will fail on such data, unless it has been aggregated appropriately. Instead, a cluster analysis algorithm may be able to detect the micro clusters formed by these patterns.

Three broad categories of anomaly detection techniques exist. **Unsupervised anomaly detection** techniques detect anomalies in an unlabeled test data set under the assumption that the majority of the instances in the data set are normal by looking for instances that seem to fit least to the remainder of the data set. **Supervised anomaly detection** techniques require a data set that has been labeled as "normal" and "abnormal" and involves training a classifier (the key difference to many other statistical classification problems is the inherent unbalanced nature of outlier detection). **Semi-supervised anomaly detection** techniques construct a model representing normal behavior from a given *normal* training data set, and then test the likelihood of a test instance to be generated by the learnt model.

Social Network Analysis

Social network analysis (SNA) is the process of investigating social structures through the use of networks and graph theory.^[1] It characterizes networked structures in terms of *nodes* (individual actors, people, or things within the network) and the *ties, edges, or links* (relationships or interactions) that connect them. Examples of social structures commonly visualized through social network analysis include social media networks,^{[2][3]} memes spread,^[4] information circulation,^[5] friendship and acquaintance networks, business networks, knowledge networks,^{[6][7]} difficult working relationships, social networks, collaboration graphs, kinship, disease transmission, and sexual relationships. These networks are often visualized through *sociograms* in which nodes are represented as points and ties are represented as lines. These visualizations provide a means of qualitatively assessing networks by varying the visual representation of their nodes and edges to reflect attributes of interest.

Social network analysis has emerged as a key technique in modern sociology. It has also gained a significant following in anthropology, biology, demography, communication studies,^{[3][13]} economics, geography, history, information science, organizational studies,^{[6][8]} political science, public health, social psychology, development studies, sociolinguistics, and computer science and is now commonly available as a consumer tool (see the list of SNA software).

Email Classification:

The dataset also involved classification of emails into documents, transactions, attorney, contractor etc. This was done using a bag of words followed by logistic regression with scikit learn. This can be described as:

In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the 'multi_class' option is set to 'ovr', and uses the cross-entropy loss if the 'multi_class' option is set to 'multinomial'. (Currently the 'multinomial' option is supported only by the 'lbfgs', 'sag', 'saga' and 'newton-cg' solvers.)

This class implements regularized logistic regression using the 'liblinear' library, 'newton-cg', 'sag', 'saga' and 'lbfgs' solvers. Note that regularization is applied by default. It can handle both dense and sparse input. Use C-ordered arrays or CSR matrices containing 64-bit floats for optimal performance; any other input format will be converted (and copied).

The 'newton-cg', 'sag', and 'lbfgs' solvers support only L2 regularization with primal formulation, or no regularization. The 'liblinear' solver supports both L1 and L2

regularization, with a dual formulation only for the L2 penalty. The Elastic-Net regularization is only supported by the 'saga' solver.

Anomaly Detection

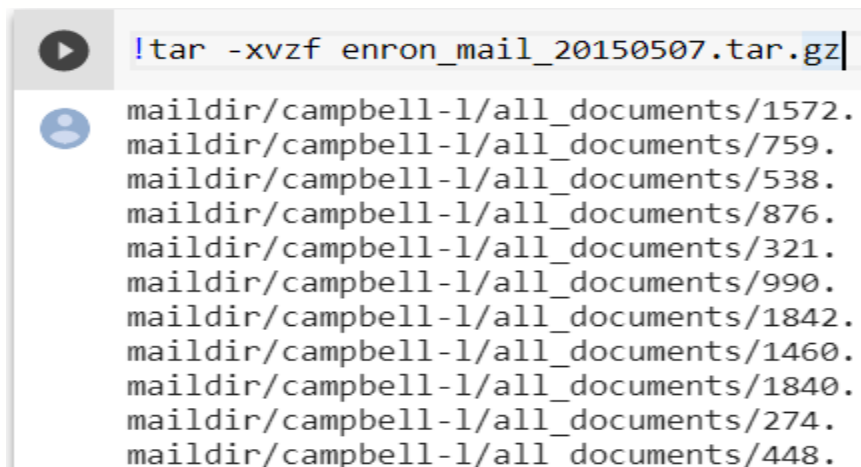
Importing Libraries: These are all the libraries needed in this part of the project for generating explanations and detecting hourly anomalous behaviour

```
import numpy as np # for data handling
import matplotlib.pyplot as plt # for visualisations
import email # for handling email data format
import os # for converting folders to dataframe
from tqdm import tqdm # for examining loops
import pandas as pd # for handling and generating dataframes
```

Downloading Data: Downloading dataset from a sub-url of the given URL

```
!wget http://www.cs.cmu.edu/~enron/enron_mail_20150507.tar.gz
```

Extracting files: Extracting all files and folders from the given folder.



```
!tar -xvzf enron_mail_20150507.tar.gz|
maildir/campbell-1/all_documents/1572.
maildir/campbell-1/all_documents/759.
maildir/campbell-1/all_documents/538.
maildir/campbell-1/all_documents/876.
maildir/campbell-1/all_documents/321.
maildir/campbell-1/all_documents/990.
maildir/campbell-1/all_documents/1842.
maildir/campbell-1/all_documents/1460.
maildir/campbell-1/all_documents/1840.
maildir/campbell-1/all_documents/274.
maildir/campbell-1/all_documents/448.
```

Preprocessing the dataset: Involves all needed transforms to interpret the data and then using for future tasks

Creating Dataframe: Converting data to table format for running various queries and processes.

```
mids = set() # for extracting only unique mails from the data as it
very large
n = 0 # for limiting the process if it is too slow
row = 0 # selecting row of the table
lists = [[0 for _ in range(int(2e6))], [0 for _ in range(int(2e6))], [0
for _ in range(int(2e6))],
```

```

        [0 for _ in range(int(2e6))], [0 for _ in range(int(2e6))], [0
for _ in range(int(2e6))],
        [0 for _ in range(int(2e6))], [0 for _ in range(int(2e6))], [0
for _ in range(int(2e6))]]
        # data is initially stored as a list of lists. This is done
because lists are comparatively faster for in allocation

# accessing all files in the root directory of the downloaded folders
for path, mid, files in tqdm(os.walk('maildir/')):
    n+=1
    if n>=400000: # limitting the process if too slow
        break
    for f in files:
        try: # rejecting the file in case of any error
            with open(path+'/'+f) as doc: # adding all email fields to the
table
                e = email.message_from_string(doc.read())
                if e.get('Date') not in mids:
                    mids.add(e.get('Message_ID'))
                    lists[0][row] = e.get('Message-ID')
                    lists[1][row] = e.get('Date')
                    lists[2][row] = e.get('From')
                    lists[3][row] = e.get('To')
                    lists[4][row] = e.get('Subject')
                    lists[5][row] = e.get('X-Folder')
                    lists[6][row] = e.get('X-Origin')
                    lists[7][row] = e.get('X-FileName')
                    lists[8][row] = e.get_payload()
                    row += 1
                mids.add(e.get('Date'))
        except:
            Pass

```


Dataframe

```
[ ] data # checking the dataframe
```

		Message-ID	Date	From	To	Subject	Path	Origin	Fi
0	<3676249.1075860770266.JavaMail.evans@thyme>		Thu, 14 Feb 2002 11:34:47 -0800 (PST)	lkunkel@trigon-sheehan.com	darrell.schoolcraft@enron.com	FW: PII LAPA	\\Darrell_Schoolcraft_Mar2002\Schoolcraft, Darr...	Schoolcraft-D	dscho Privile
1	<28120767.1075860770051.JavaMail.evans@thyme>		Mon, 10 Dec 2001 04:39:59 -0800 (PST)	darrell.schoolcraft@enron.com	steve.january@enron.com		\\Darrell_Schoolcraft_Mar2002\Schoolcraft, Darr...	Schoolcraft-D	dscho Privile
2	<26828195.1075860770221.JavaMail.evans@thyme>		Mon, 11 Feb 2002 05:15:26 -0800 (PST)	darrell.schoolcraft@enron.com	steve.january@enron.com, steven.harris@enron.c...	Transwestern scheduled volumes	\\Darrell_Schoolcraft_Mar2002\Schoolcraft, Darr...	Schoolcraft-D	dscho Privile

Anomaly Detection

Processing for anomaly detection

Adding content length as a column: This is also an important column for analysis as it indicates the length of the content of the email and will be very helpful for flagging deviations from the general trends in the data. Mails which are not replies generally have much larger size than those which have which will be shown later.

Adding a column for whether a mail is a reply or not: This column will be used later as a field for anomaly detection. This column also indicates how many small conversations are going on as a feature in the dataset and will be visualised later. The column also shows that mails in it are generally shorter than normal which will also be demonstrated later.

Generating Time Series data: This involves grouping the data according to the hour and day. This data can then be used to perform time series analysis and anomaly detection. An aggregate like mean and count is generally taken for various fields which will be shown as and when being performed over all mails in that particular hour.

```
grouped = data.groupby(by = (times.date, times.hour)) # grouping the data
```

```
x.T[1] = x.T[1]/x.T[0] # normalising the data according to the count as the points will not be scaled otherwise
x.T[2] = x.T[2]/x.T[0]
```

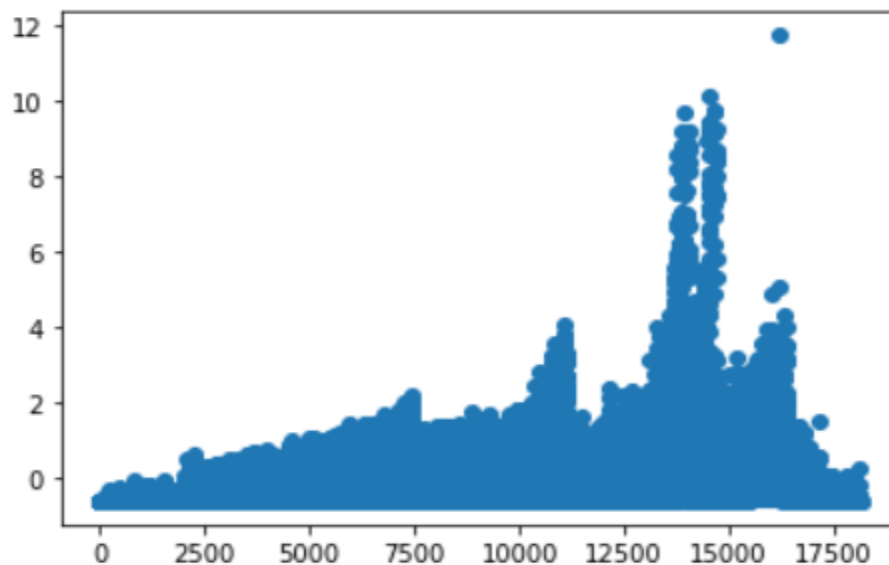
```
[ ] x
```

```
array([[ 1., 25., 0.],  
       [ 1., 942., 0.],  
       [ 1., 194., 0.],  
       ...,  
       [ 1., 215., 0.],  
       [ 1., 152., 0.],  
       [ 1., 225., 0.]])
```

In the above graph which indicate the normalised count, the number of mails shoot up very fast in a very specific time frame which indicates anomalous behaviour in that period of 2 to 3 days. Similar anomalous hours are also seen about a week after that. This can also be a strong indicator of fraudulent activities like deletion of large amounts of records and other instructions being passed and expecting immediate reporting on the same.

```
[ ] plt.scatter([i for i in range(len(x))],x.T[0])
```

```
<matplotlib.collections.PathCollection at 0x7ff10f0ff710>
```

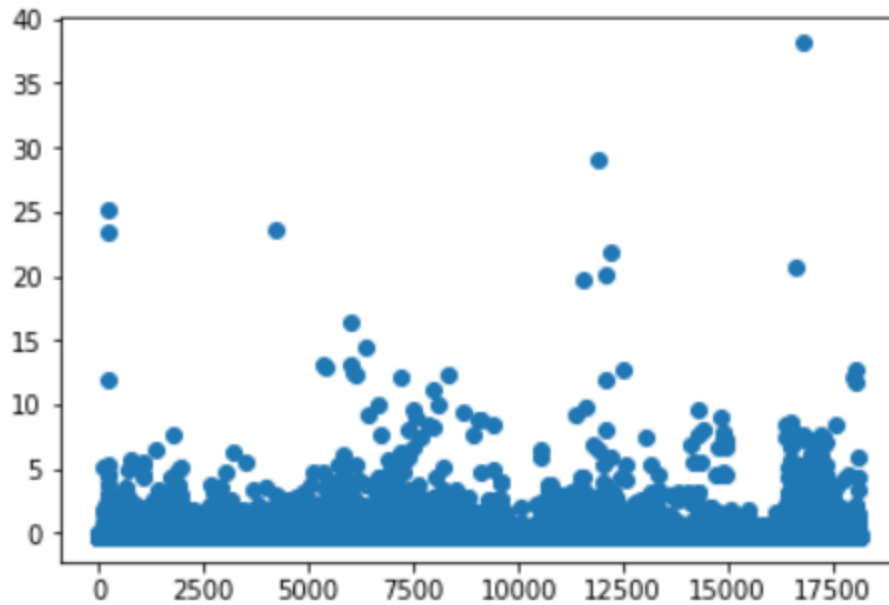


The sharp anomalous surges in the number of mails were seen to be preceded by surges in the length of mails thus it is also a very important indicator of fraudulent activity.

```
[ ] plt.scatter([i for i in range(len(x))],x.T[1])
```



<matplotlib.collections.PathCollection at 0x7ff10dce0240>

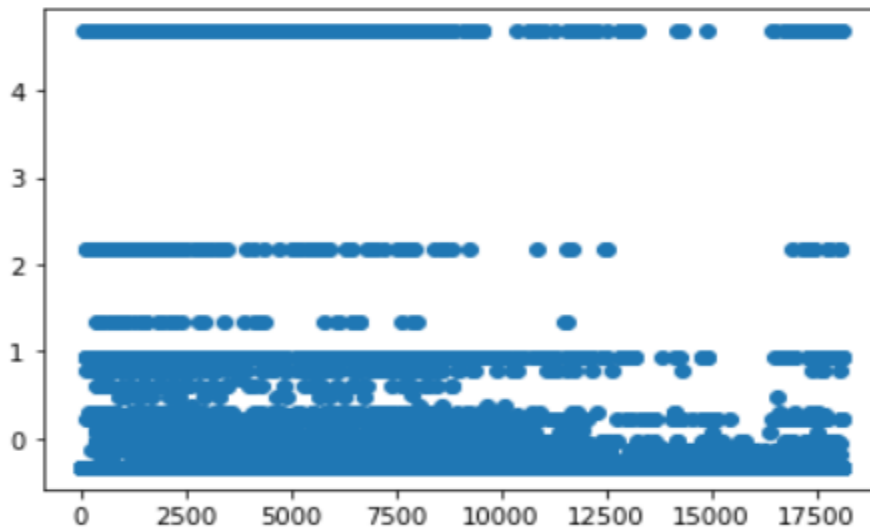


Number of replies in most conditions is very low or extremely high depending on the hour being considered. These points of multiple replies on the same topic can be of key interest in some cases.

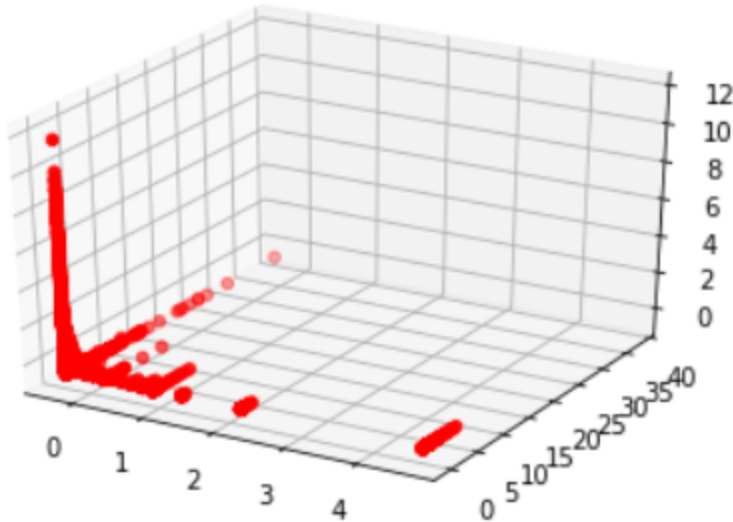
```
[ ] plt.scatter([i for i in range(len(x))],x.T[2])
```



<matplotlib.collections.PathCollection at 0x7ff10f6df6a0>



In most cases anomalies are in one of the 3 dimensions and rarely in all them together. Some points are slightly anomalous in terms of length and replies but do not deviate significantly.



Performing Anomaly Detection

Isolation Forests

This is an anomaly detection method which can be used in this case. It builds trees by performing random splits and on the data. Anomalies will be away from the data and thus will be differentiated the most from these random separation.

More

details:

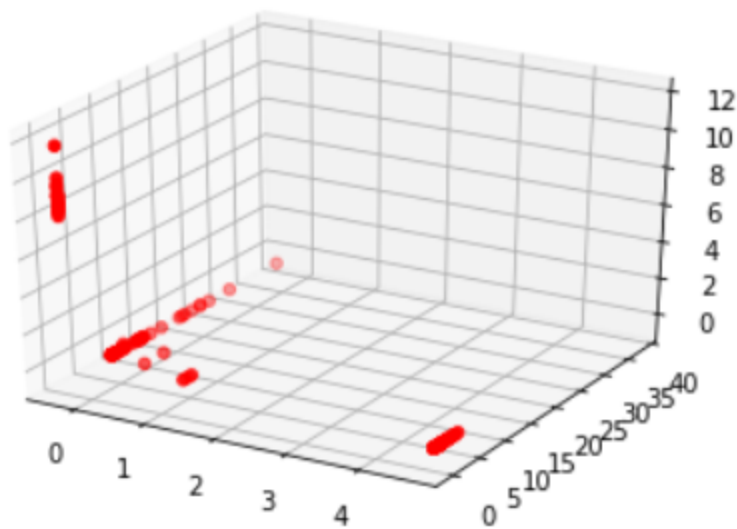
<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html#sklearn.ensemble.IsolationForest>

```
if_model = IsolationForest(max_features=3, n_estimators=1000,
verbose=2, contamination=0.01)
# verbose is to show the random separators
# n-estimators is the number of random differentiators
# contamination decides how much data should be separated as anomaly
# max_features decides how many features should be considered during
a split
```

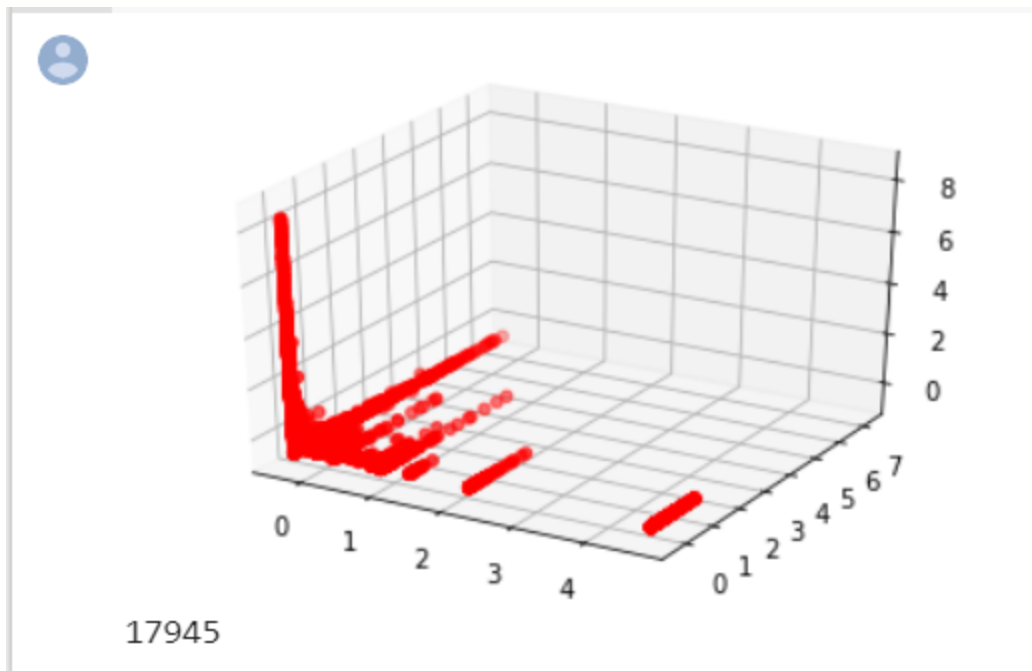
```
[ ] if_model.fit(x) # finding the anomalies
```

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
Building estimator 1 of 1000 for this parallel run (total 1000)...
Building estimator 2 of 1000 for this parallel run (total 1000)...
Building estimator 3 of 1000 for this parallel run (total 1000)...
Building estimator 4 of 1000 for this parallel run (total 1000)...
Building estimator 5 of 1000 for this parallel run (total 1000)...
Building estimator 6 of 1000 for this parallel run (total 1000)...
Building estimator 7 of 1000 for this parallel run (total 1000)...
Building estimator 8 of 1000 for this parallel run (total 1000)...
Building estimator 9 of 1000 for this parallel run (total 1000)...
Building estimator 10 of 1000 for this parallel run (total 1000)...
Building estimator 11 of 1000 for this parallel run (total 1000)...

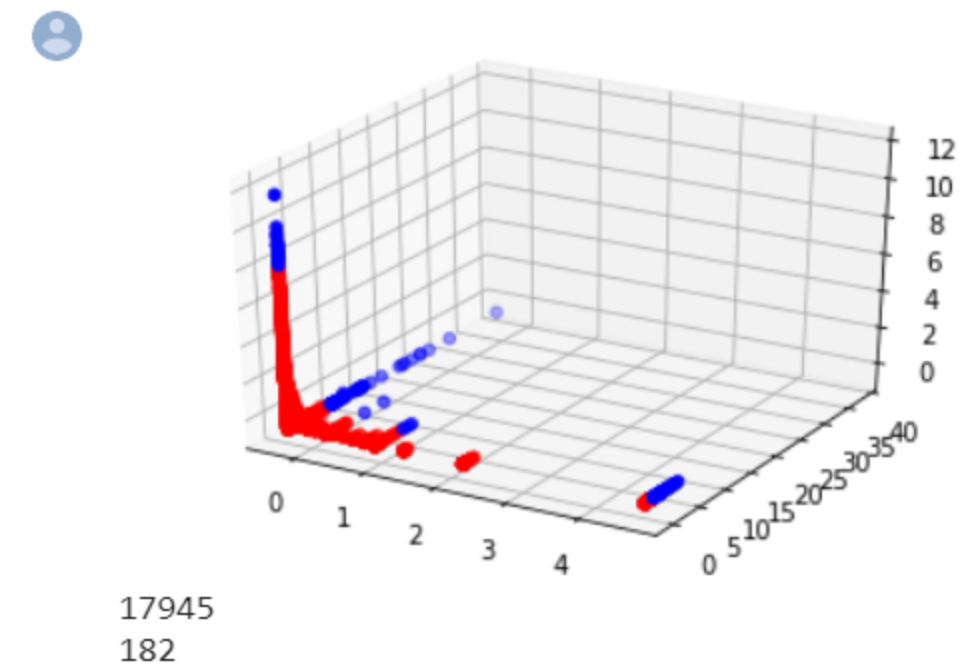
using 0.1 as a threshold and printing the anomalies



printing the considered data points



Both combined (Anomalies and Non-anomalies)



Local Outlier factor

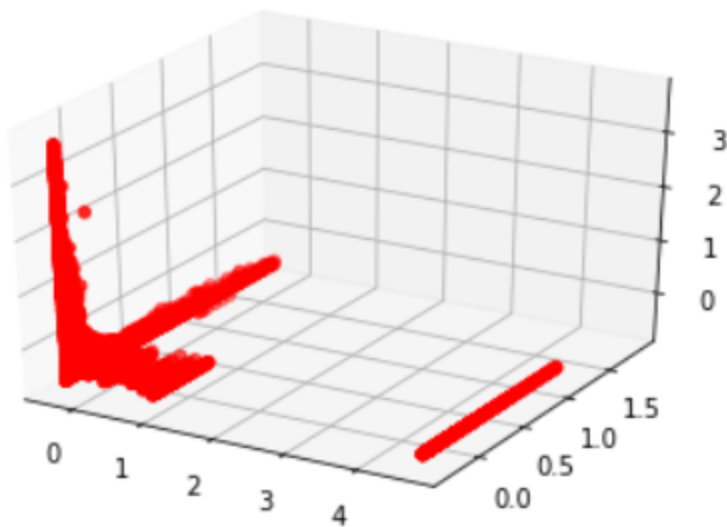
This is another method for finding outliers which uses distances from nearest neighbours and thus does separation based on density. More can be learnt at: <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.LocalOutlierFactor.html>

```
lof_model = LocalOutlierFactor(novelty=True, n_neighbors=500)
# novelty helps in detecting new features using k nearest neighbours
# n-neighbours helps in deciding the density for outlier
```

```
[ ] lof_model.fit(x)
```

```
LocalOutlierFactor(algorithm='auto', contamination='auto', leaf_size=30,
metric='minkowski', metric_params=None, n_jobs=None,
n_neighbors=500, novelty=True, p=2)
```

Data seems to have very high density in 2 major regions. Both need to be analysed. Most anomalies are way further.



16940

1187

Generated more graph based features

These features are being generated after referring various research papers which have indicated that most anomalies involving fraud had communications which between entities which were not communicating earlier. It also involved high fraction of communications with external entities which again will be considered for feature engineering.

Research papers:

- <https://www.cs.cmu.edu/~deswaran/papers/kdd18-spotlight.pdf>
- <https://www.cs.cmu.edu/~deswaran/papers/icdm18-sedanspot.pdf>
- https://www.andrew.cmu.edu/user/lakoglu/pubs/oddball_TR.pdf

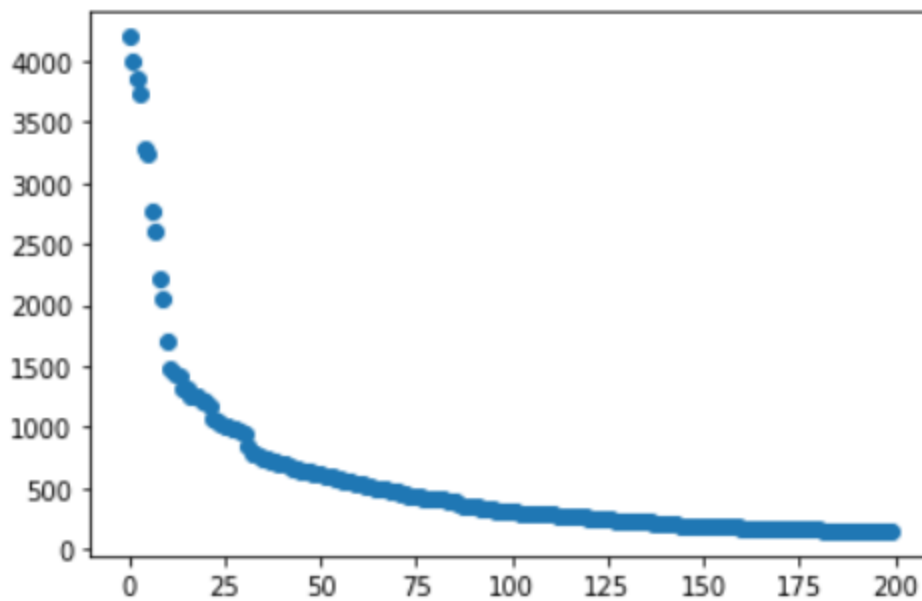
Creating count of mail sent by every source and receivers



```
from_count
{ 'lkunkel@trigon-sheehan.com': 6,
  'darrell.schoolcraft@enron.com': 504,
  'tk.lohman@enron.com': 133,
  'ron.matthews@enron.com': 59,
  'jeanne.licciardo@enron.com': 4,
  'michelle.lokay@enron.com': 426,
  'david.roensch@enron.com': 50,
  'elizabeth.brown@enron.com': 79,
  'mansoor.abdmoulaie@enron.com': 37,
  'richard.hanagriff@enron.com': 59,
  'melinda.gallishaw@enron.com': 15,
  'max.brown@enron.com': 6,
  'maria.salazar@enron.com': 1,
  'enron.payroll@enron.com': 29,
  'michael.bodnar@enron.com': 71,
  'john.sturn@enron.com': 17,
  'ramona.betancourt@enron.com': 39,
  'rc@mail.greeting-cards.com': 3,
}
```

Count of emails sent by every source

Most of the mails have been sent by the top 200 employees and thus it would be more relevant to consider only their data for general so that external factors don't influence significantly.



```
[ ] unique_to_count
```

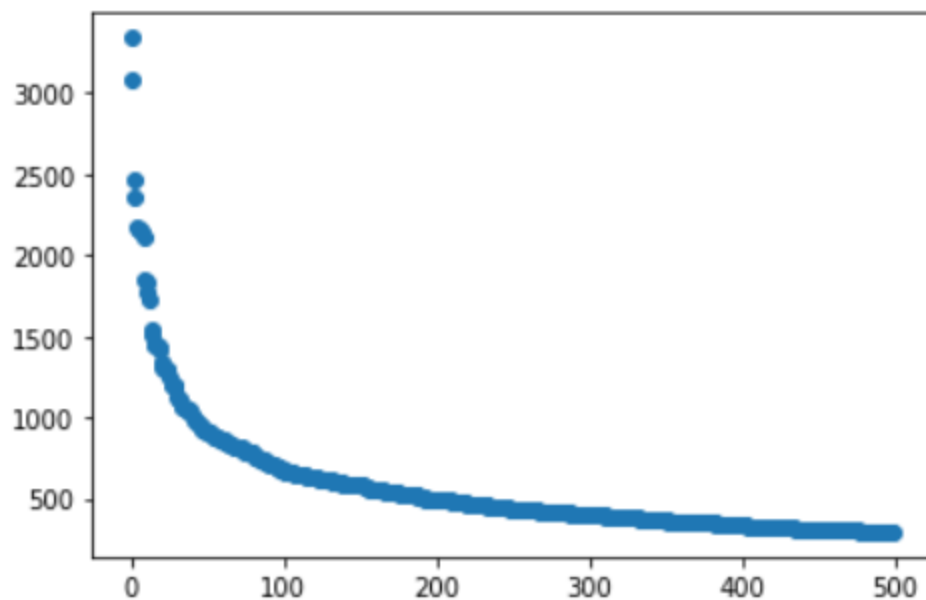


```
{'stevejanuary@enroncom': 427,  
'stevenharris@enroncom': 1291,  
'kimberlywatson@enroncom': 1210,  
'tklohman@enroncom': 1197,  
'michellelokay@enroncom': 1183,  
'debbiemoseley@enroncom': 112,  
'janmoore@enroncom': 488,  
'darrellschoolcraft@enroncom': 836,  
'berthernandez@enroncom': 298,  
'davidroensch@enroncom': 217,  
... ..
```

```
[ ] receivers
```

 ['richardshapiro@enroncom',
'jeffdasovich@enroncom',
'susanmara@enroncom',
'louisekitchen@enroncom',
'timbelden@enroncom',
'paulkaufman@enroncom',
'jamessteffes@enroncom',
'tanajones@enroncom',
'sarashackleton@enroncom',
'stevenkean@enroncom',
...]

Top 500 recipients have actually received most of the emails in the organisation.



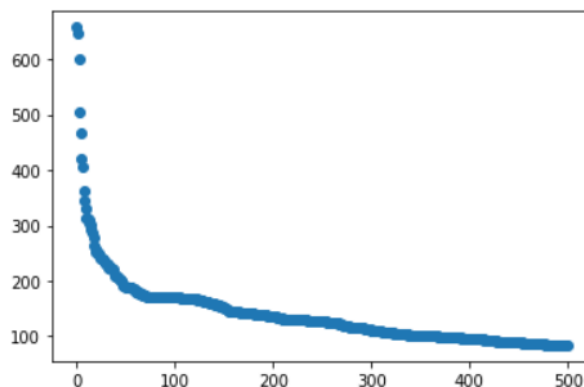
Creating count of every pair of source and destination: Creating a new feature for every pair involved in a mail. This will help in identifying new novel communication patterns.

```
[ ] unique_pair_count
```

```
{'lkunkel@trigon-sheehan.com darrell.schoolcraft@enron.com': 1,  
'darrell.schoolcraft@enron.com steve.january@enron.com': 1,  
'darrell.schoolcraft@enron.com stevejanuary@enroncom': 12,  
'darrell.schoolcraft@enron.com stevenharris@enroncom': 11,  
'darrell.schoolcraft@enron.com kimberlywatson@enroncom': 24,  
'darrell.schoolcraft@enron.com tklohman@enroncom': 21,  
'darrell.schoolcraft@enron.com michellelokay@enroncom': 12,  
'tk.lohman@enron.com darrell.schoolcraft@enron.com': 1,  
'tk.lohman@enron.com debbiemoseley@enroncom': 38,  
'tk.lohman@enron.com janmoore@enroncom': 52,  
'tk.lohman@enron.com darrellschoolcraft@enroncom': 47,  
'tk.lohman@enron.com berthernandez@enroncom': 1,
```

```
[ ] plt.scatter([i for i in range(len(pairs))][:500], [unique_pair_count[k] for k in pairs][:500])
```

```
<matplotlib.collections.PathCollection at 0x7ff0f35c8748>
```



Find unique external emails: Unique external senders and recipients have been noted as they are vents for the organisational information to escape.

```
[ ] outside_sent_mails
```


 { 'michael.hayes@beachfire.com',
'fruitsalad@quickmail.quickhosts.com',
'greg.malcolm@ontariopowergeneration.com',
'carla_j@compuserve.com',
'juliebrandimarte@hotmail.com',
'marketinfo2002460782@yahoo.com',
'hotdeals.26@reply.pm0.net',
'tom-jerilynn@worldnet.att.net',
'tz3qu@msn.com',
'dana@gablegroup.com',
'support@learningco.com',

```
[ ] outside_received_mails
```


 { 'corrier@cox-internetcom',
'csunser@worldnetattnet',
'ckaitson@midcoastenergycom',
'andersg@kealincolnacnz',
'brivers@telconetnet',
'slokopriya@aolcom',
'masterdave27@aolcom',
'sboyle@haasberkeleyedu',
'aperez@caisocom',
'zibam@theticketcompanycom',
'stephenbrown@rbcdaincom',

Creating new features


```
[ ] outside_sent_mails
```

 { 'michael.hayes@beachfire.com',
'fruitsalad@quickmail.quickhosts.com',
'greg.malcolm@ontariopowergeneration.com',
'carla_j@compuserve.com',
'juliebrandimarte@hotmail.com',
'marketinfo2002460782@yahoo.com',
'hotdeals.26@reply.pm0.net',
'tom-jerilynn@worldnet.att.net',
'tz3qu@msn.com',
'dana@gablegroup.com',
'support@learningco.com',
'gljackson2@tva.gov',

```
[ ] outside_sender
```

 [1,
0,
0,
0,
0,
0,
0,
0,

```
[ ] outside_receiver
```

 [0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0]

```
# creating new feature to detect sporadic mails which can show
significant events
fraction_sent = [0 for _ in range(len(data))]
for i in range(len(data)):
    if unique_from_count[data.From[i]]>30: # only considering cases
where the employee has significant reception
        fraction_sent[i] += 1/unique_from_count[data.From[i]]
```

[] unique_from_count

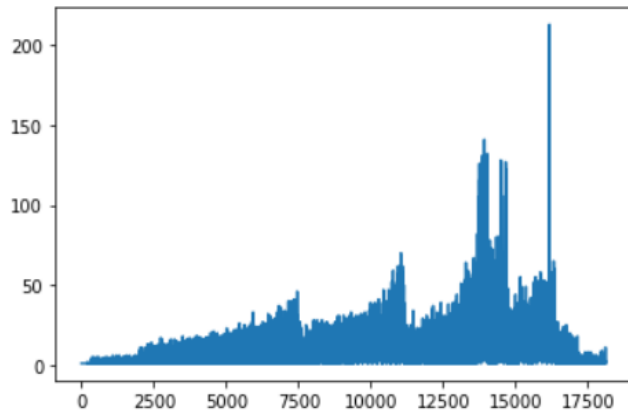
```
{'lkunkel@trigon-sheehan.com': 6,
'darrell.schoolcraft@enron.com': 504,
'tk.lohman@enron.com': 133,
'ron.matthews@enron.com': 59,
'jeanne.licciardo@enron.com': 4,
'michelle.lokay@enron.com': 426,
'david.roensch@enron.com': 50,
'elizabeth.brown@enron.com': 79,
...}
```

[] fraction_sent

```
[0,
0.001984126984126984,
0.001984126984126984,
0.007518796992481203,
0.007518796992481203,
0.001984126984126984,
0.001984126984126984,
0.001984126984126984,
0.01694915254237288,
```

```
plt.plot([i for i in range(len(grouped.nunique()['From']))], grouped.nunique()['From'])
```

```
[<matplotlib.lines.Line2D at 0x7ff0eac5d710>]
```



```
fraction_communication|
```

```
[0,  
 0,  
 0,  
 0,  
 0.0668231544491343,  
 0,  
 0,  
 0,  
 0,  
 0,  
 0,  
 0,
```

Creating data: Creating numpy array which has all features to consider for final anomaly detection

```
grouped = data.groupby(by = (times.date, times.hour))
```



```
[ ] x
```

```
array([[ 25.,  0.,  0., ...,  0.,  1.,  1.],
       [942.,  0.,  1., ...,  0.,  1.,  1.],
       [194.,  0.,  1., ...,  0.,  1.,  1.],
       ...,
       [215.,  0.,  1., ...,  0.,  1.,  1.],
       [152.,  0.,  1., ...,  0.,  1.,  1.],
       [225.,  0.,  1., ...,  0.,  1.,  1.]])
```

Visualising

```
final_features # list of all features
```

```
['Content Length',
 'Reply',
 'Outside Sender',
 'Outside Receiver',
 'Fraction Sent',
 'Fraction Received',
 'Fraction Communication',
 'Unique Senders',
 'Number of mails']
```

```
x|
```

```
array([[ 25.,  0.,  0., ...,  0.,  1.,  1.],
       [942.,  0.,  1., ...,  0.,  1.,  1.],
       [194.,  0.,  1., ...,  0.,  1.,  1.],
       ...,
       [215.,  0.,  1., ...,  0.,  1.,  1.],
       [152.,  0.,  1., ...,  0.,  1.,  1.],
       [225.,  0.,  1., ...,  0.,  1.,  1.]])
```

```
[ ] smooth_x
```

```
array([[1.61480e+02, 0.00000e+00, 1.20000e-01, ..., 0.00000e+00,
        1.00000e+00, 1.00000e+00],
       [1.62600e+02, 0.00000e+00, 1.20000e-01, ..., 0.00000e+00,
        1.00000e+00, 1.00000e+00],
       [1.25640e+02, 0.00000e+00, 8.00000e-02, ..., 0.00000e+00,
        1.00000e+00, 1.00000e+00],
       ...,
       [7.44936e+03, 4.00000e-02, 1.80000e+00, ..., 0.00000e+00,
        1.96000e+00, 2.28000e+00],
       [7.39524e+03, 4.00000e-02, 1.84000e+00, ..., 0.00000e+00,
        1.96000e+00, 2.28000e+00],
       [7.35444e+03, 4.00000e-02, 1.88000e+00, ..., 0.00000e+00,
        1.96000e+00, 2.28000e+00]])
```

Creating final Anomaly detection system using Isolation forests:

```
# normalising features by number of mails
x.T[-2] = x.T[-2]/x.T[-1]
x.T[-4] = x.T[-4]/x.T[-1]
x.T[-7] = x.T[-7]/x.T[-1]

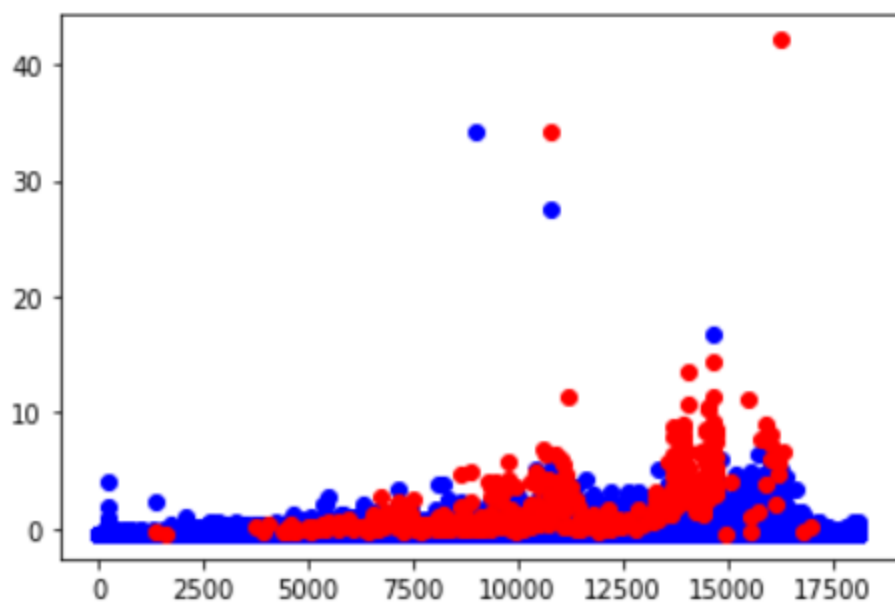
# z score normalisation
x_m = x.mean(axis=0)
x_s = x.std(axis=0)
normed_x = (x-x_m)/x_s
```

```
[ ] model = IsolationForest(contamination=0.03,n_estimators=1000, verbose=3)
```

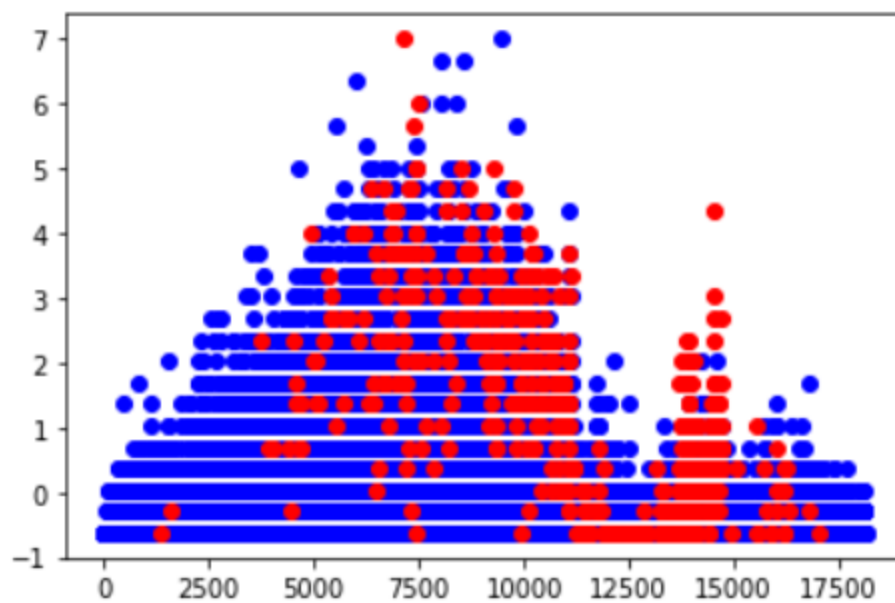
```
[ ] y = model.fit_predict(x_new)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.  
Building estimator 1 of 1000 for this parallel run (total 1000)...  
Building estimator 2 of 1000 for this parallel run (total 1000)...  
Building estimator 3 of 1000 for this parallel run (total 1000)...  
Building estimator 4 of 1000 for this parallel run (total 1000)...  
Building estimator 5 of 1000 for this parallel run (total 1000)...  
Building estimator 6 of 1000 for this parallel run (total 1000)...  
Building estimator 7 of 1000 for this parallel run (total 1000)...  
Building estimator 8 of 1000 for this parallel run (total 1000)...  
Building estimator 9 of 1000 for this parallel run (total 1000)...  
Building estimator 10 of 1000 for this parallel run (total 1000)...  
Building estimator 11 of 1000 for this parallel run (total 1000)...  
Building estimator 12 of 1000 for this parallel run (total 1000)...  
Building estimator 13 of 1000 for this parallel run (total 1000)...  
Building estimator 14 of 1000 for this parallel run (total 1000)...  
Building estimator 15 of 1000 for this parallel run (total 1000)...
```

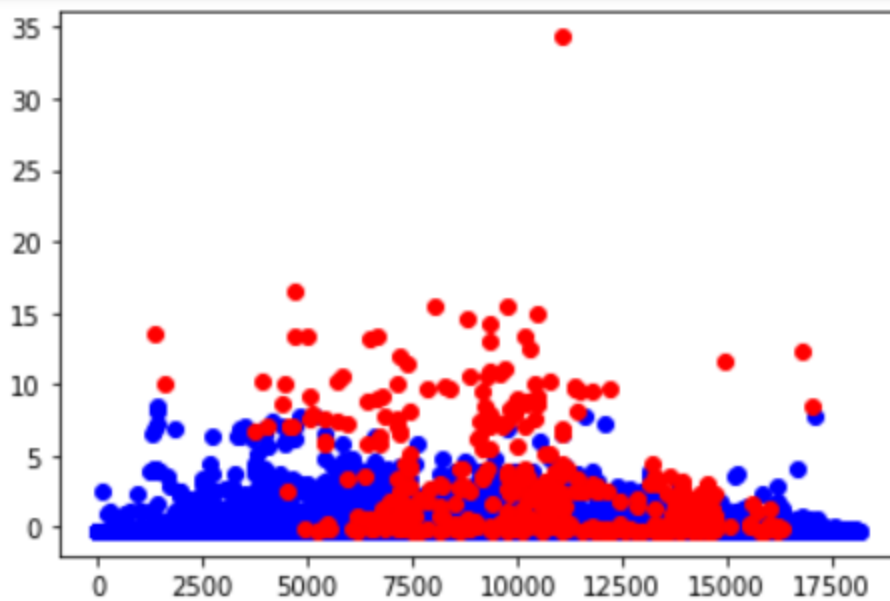
As it can be clearly seen, most of the outliers are now being detected and flagged. The number of flagged cases is about 550 hours out of 18000 hours which can be analyzed as it is only about 3% of all the cases.



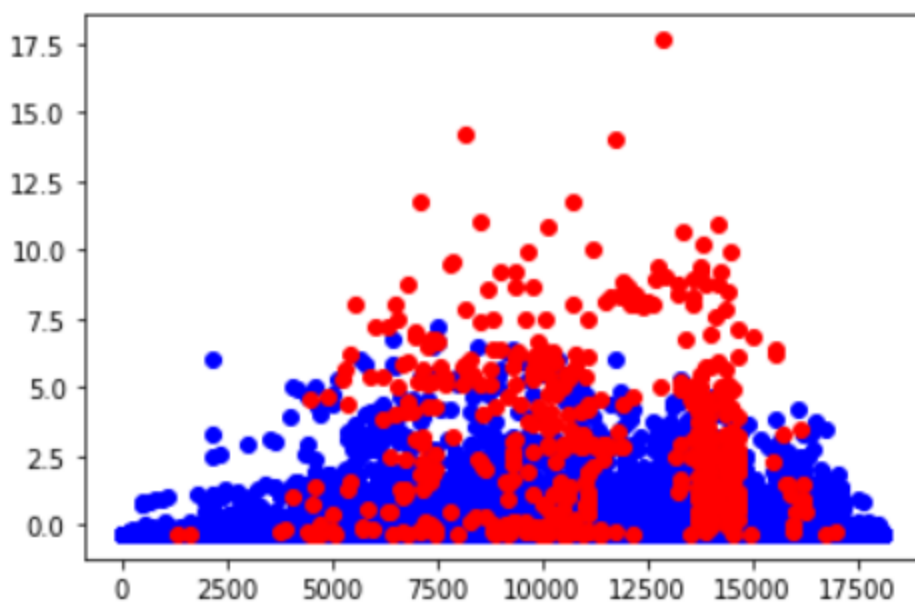
Content Length



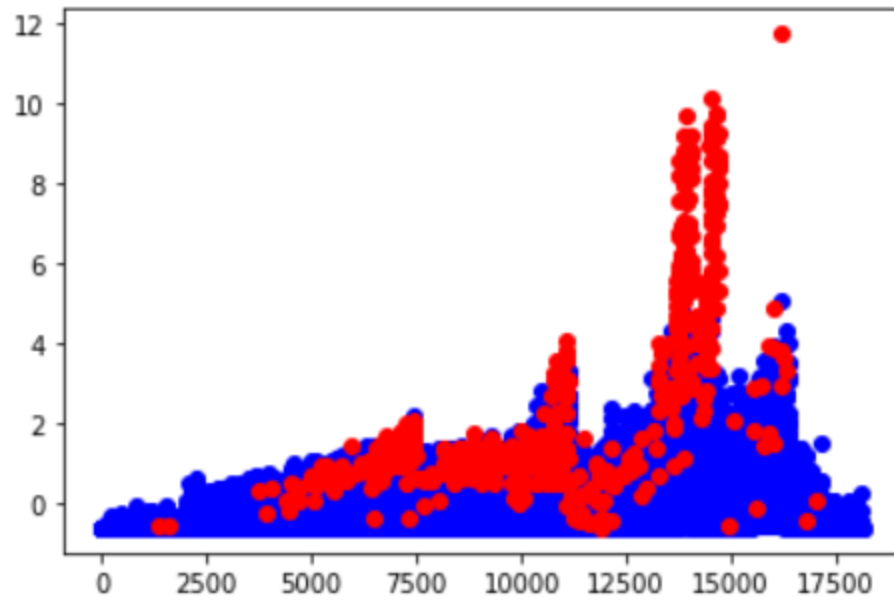
Reply




Outside Receiver




Fraction Communication



Number of mails

 550/18000|

 0.030555555555555555

Network Analysis

```
data = pd.read_csv('/content/drive/My Drive/submisisions/sem 8/ml/Enron/data6.csv')

del data['recipients']
del data['senders']
del data['file']
del data['message']

data = data[['date', 'sender', 'recipient1', 'recipient2', 'recipient3', 'subject', 'text']]

print(len(data))
print(data)
```

```
405000
      date ... text
0  2001-05-14 16:39:00-07:00 ... [' ', 'Here is our forecast', ' ', ' ']
1  2001-05-04 13:51:00-07:00 ... [' ', 'Traveling to have a business meeting tak...
2  2000-10-18 03:00:00-07:00 ... [' ', 'test successful. way to go!!!']
3  2000-10-23 06:13:00-07:00 ... [' ', 'Randy,', ' ', ' Can you send me a schedul...
4  2000-08-31 05:07:00-07:00 ... [' ', "Let's shoot for Tuesday at 11:45.  "]
...
404995 2001-01-22 00:34:00-08:00 ... [' ', 'need help.  ']
404996 2000-12-06 04:41:00-08:00 ... ['X-FileName: pallen.nsf', ' ', 'please remove ...
404997 2001-01-18 02:12:00-08:00 ... [' ', 'Jeff,', ' ', 'Here is a recent rentroll. ...
404998 2001-01-18 02:08:00-08:00 ... [' ', 'Larry,', ' ', 'The wire should go out tod...
404999 2001-01-17 07:35:00-08:00 ... [' ', 'Lucy,', ' ', 'Why did so many tenants not...

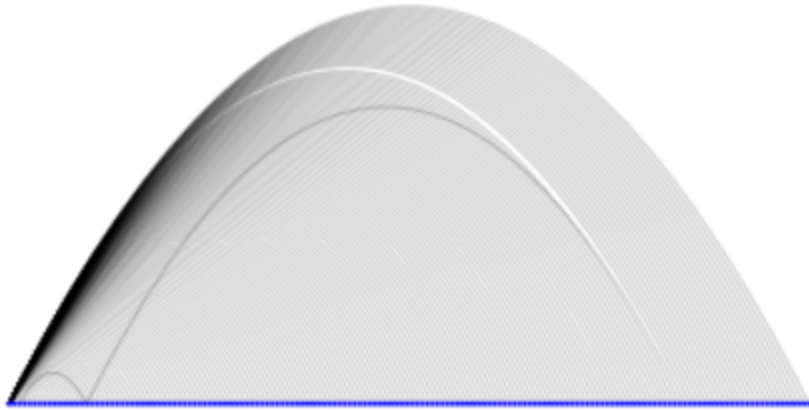
[405000 rows x 7 columns]
```

OK, looks good. we have what we want to work with now. Obviously we have discarded quite a bit of information as well, but a lot of the content in each message simply doesn't interest me at the moment. we don't care, for instance, how the messages were encoded. So for now we are happy with what we've got.

Let's draw a network. There are quite a few options available, of which I'll plot just three or four. First up...

ArcPlot

We can see our nodes (employees) at the bottom and the dominance of the one node on the right side. 517,000 row dataset here, so we are looking at emails from 150 Enron executives contained in the full dataset.



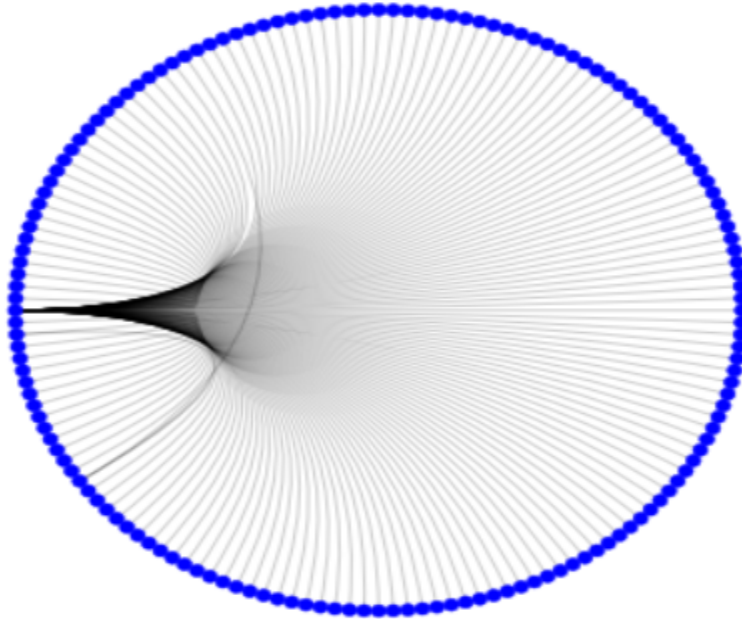
Next we'll bend the ends of this plot and join them together to form a circle, otherwise known as a

CircosPlot

Same deal here, although in my opinion a lot clearer and easier to comprehend.

Still, this leaves a lot to be desired. It just doesn't *look* like what most folks expect from a network graph


```
[ ] plot = nv.CircosPlot(G)
    plot.draw()
    plt.show()
```

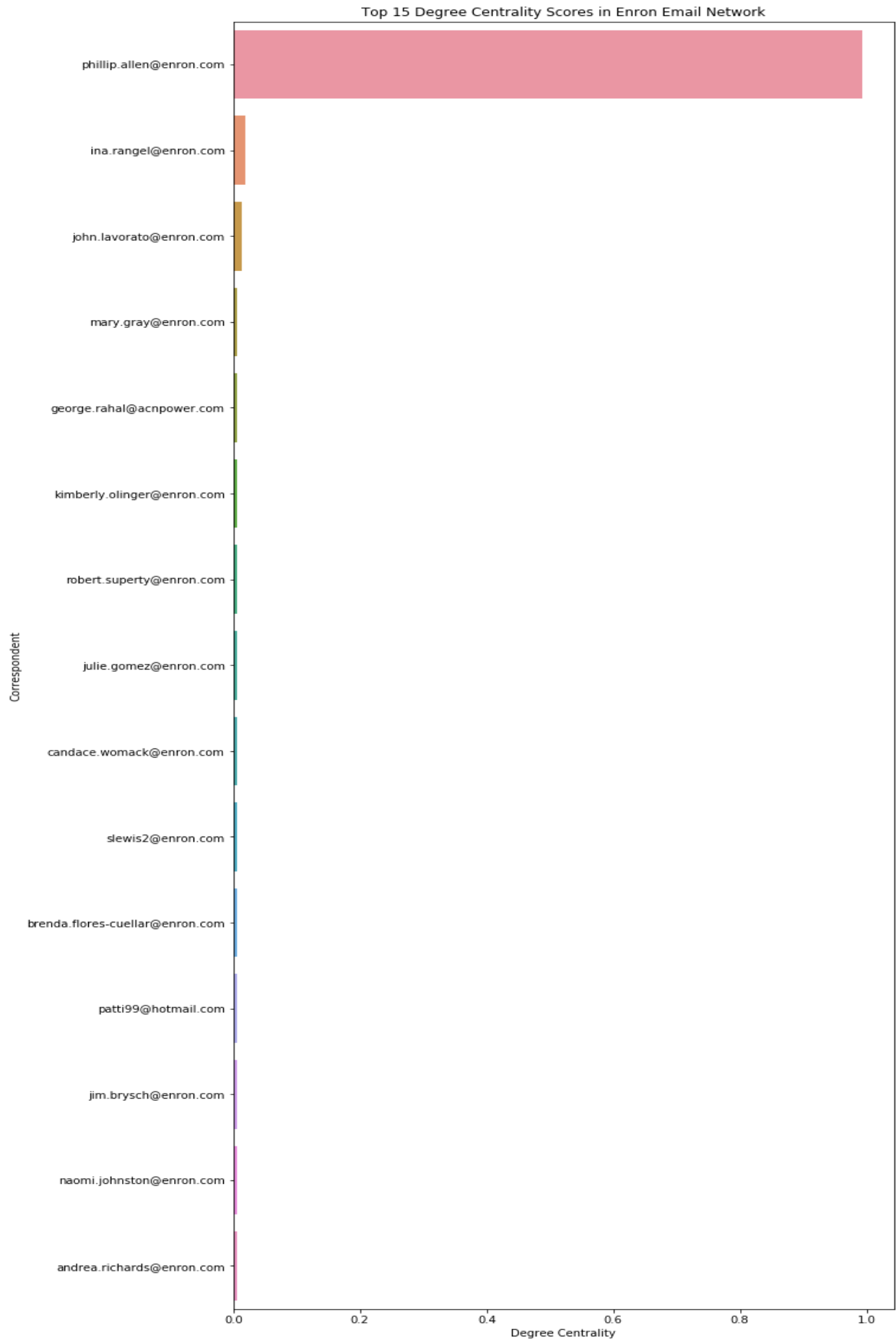


So next we will use nxviz to

Draw the Network

We plotted this a few times with different "tension" on the spring layout to show some different options for visualizing this network. We've been finding that it pays to use a small 'k' value and plot these with a large figsize (200 x 200 or larger). This produces a big enough graphic to allow you to really zoom in and examine minor details without getting overwhelmed by the number of labels. Of course, if you choose to turn the labels off with `with_labels=False` you can avoid that issue, but this is no fun (and not particularly interesting) without the names attached.

As you can see, Mr. Allen is connected to everyone. Next we can look at betweenness centrality, which measures the degree to which each node is the 'single point of contact', so to speak, between other nodes or cliques.



Betweenness Centrality



Email Classification

Emails classified according to their subject and body content in order to get insights about what is being discussed in the email. Each email gets assigned a set of classes with varying weights.

```
[ ] path = '/content/drive/My Drive/BE/Sem 8/ML/IA-2/emails.csv'
    emails = pd.read_csv(path)
    emails.head()
```



	file	message
0	allen-p/_sent_mail/1.	Message-ID: <18782981.1075855378110.JavaMail.e...
1	allen-p/_sent_mail/10.	Message-ID: <15464986.1075855378456.JavaMail.e...
2	allen-p/_sent_mail/100.	Message-ID: <24216240.1075855687451.JavaMail.e...
3	allen-p/_sent_mail/1000.	Message-ID: <13505866.1075863688222.JavaMail.e...
4	allen-p/_sent_mail/1001.	Message-ID: <30922949.1075863688243.JavaMail.e...

```
[ ] def parse_into_emails(messages):
    emails = [parse_raw_message(message) for message in messages]
    return {
        'body': map_to_list(emails, 'body'),
        'to': map_to_list(emails, 'to'),
        'from_': map_to_list(emails, 'from')
    }
```

```
[ ] # cleaning
def parse_raw_message(raw_message):
    lines = raw_message.split('\n')
    email = {}
    message = ''
    keys_to_extract = ['from', 'to']
    for line in lines:
        if ':' not in line:
            message += line.strip()
            email['body'] = message
        else:
            pairs = line.split(':')
            key = pairs[0].lower()
            val = pairs[1].strip()
            if key in keys_to_extract:
                email[key] = val
    return email
```

```
[ ] def map_to_list(emails, key):
    results = []
    for email in emails:
        if key not in email:
            results.append('')
        else:
            results.append(email[key])
    return results
```

```
[ ] email_df = pd.DataFrame(parse_into_emails(email_subset.message))
print(email_df.head())
```

```

body ... from_
0 Here is our forecast ... phillip.allen@enron.com
1 Traveling to have a business meeting takes the... ... phillip.allen@enron.com
2 test successful. way to go!!! ... phillip.allen@enron.com
3 Randy,Can you send me a schedule of the salary... ... phillip.allen@enron.com
4 ... phillip.allen@enron.com

[5 rows x 3 columns]
```

```
from gensim.models.phrases import Phrases, Phraser
```

```
# Build the bigram and trigram models
bigram = Phrases(data_words, min_count=5, threshold=100) # higher threshold fewer phrases.
trigram = Phrases(bigram[data_words], threshold=100)
```

```
/usr/local/lib/python3.6/dist-packages/gensim/models/phrases.py:598: UserWarning: For a faster
warnings.warn("For a faster implementation, use the gensim.models.phrases.Phraser class")
```

```
bigram_mod = Phraser(bigram)
trigram_mod = Phraser(trigram)
```

```
print(trigram_mod[bigram_mod[data_words[200]]])
```

```
['forwarded', 'by', 'phillip_allen', 'hou', 'ect', 'on', 'hello', 'men', 'have', 'attached', '']
```

```
def remove_stopwords(texts):
    return [[word for word in simple_preprocess(str(doc)) if word not in stop_words] for doc in texts]

def make_bigrams(texts):
    return [bigram_mod[doc] for doc in texts]

def make_trigrams(texts):
    return [trigram_mod[bigram_mod[doc]] for doc in texts]

def lemmatization(texts, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']):
    """https://spacy.io/api/annotation"""
    texts_out = []
    for sent in texts:
        doc = nlp(" ".join(sent))
        texts_out.append([token.lemma_ for token in doc if token.pos_ in allowed_postags])
    return texts_out
```

```
# create dictionary and corpus both are needed for (LDA) topic modeling
```

```
# Create Dictionary
```

```
id2word = corpora.Dictionary(data_lemmatized)
```

```
# Create Corpus
```

```
texts = data_lemmatized
```

```
# Term Document Frequency
```

```
corpus = [id2word.doc2bow(text) for text in texts]
```

```
# Build LDA model
```

```
lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,  
                                              id2word=id2word,  
                                              num_topics=20,  
                                              random_state=100,  
                                              update_every=1,  
                                              chunksize=100,  
                                              passes=10,  
                                              alpha='auto',  
                                              per_word_topics=True)
```

```
print(lda_model.print_topics())
```

```
[(0, '0.072*"document" + 0.045*"transaction" + 0.032*"attorney" + 0.025*"color" + 0.025*"contractor" +
```

Word Cloud

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

```
from wordcloud import WordCloud, STOPWORDS
```

```
import os, sys, email, re, string
```

```
from nltk.stem import WordNetLemmatizer
```

```
import nltk
```

```
nltk.download('wordnet')
```

```
nltk.download('punkt')
```

```
nltk.download('averaged_perceptron_tagger')
```

```
from collections import Counter
```

```
lemmatiser = WordNetLemmatizer()
```

```
def clean(text):
```

```
    # Function that get string and return it after cleaning as pure  
    terms
```

```
    stop = set(STOPWORDS)
```

```
    # Excludes irrelevant words
```



```
stop.update(('fwd','RE','FW','Hello','Meeting','Ga','Access','positio
ns','list','forward','floor','collar','fixed',

'enron','hou','ect','corp','please','vince','time','mail','john','kay
','day','message','week','kaminski','year',

'meeting','enronxgate','question','group','work','call','scott','chan
ge','company','let','mann','date','number',

'mark','today','david','mike','issue','houston','chris','subject','wa
y','bass','jeff','edu','office','doc','don',

'month','copy','name','comment','email','need','phone','point','thing
','request','look','ben','michael','list',

'help','delainey','fax','morning','use','tomorrow','thank','phillip',
'hotmail','guy','robert','night','lon',

'part','talk','kate','home','mailto','person','address','form','jeffr
ey','something','end','line','hour',

'place','march','love','anything','paul','giron','smith','hope','darr
on','jim','kevin','weekend','george',

'north','someone','section','richard','discus','bob','jacoby','ena','
room','see','demand','desk','area',

'everyone','greg','detail','jason','afternoon','discussion','tom','ks
law','check','basis','visit','mcconnell',

'miller','entity','location','peter','monday','response','show','page
','jennifer','lot','meet','respond',

'yesterday','pdx','house','june','larry','jan','dan','city','july','j
udy','friday','julie','shirley','meter',

'level','fyi','addition','martin','anyone','generation','department',
'type','rick','friend','period','word',

'lisa','think','class','johnson','org','robin','thompson','columbiaga
s','didn','april','william','lee','thomas',
```

```
'hey','adam','stephen','man','sender','tim','taylor','organization','center','everything','ferc','notice',
```

```
'start','davis','york','sorry','cell','return','street','hernandez','thursday','campbell','care','content',
```

```
'curve','minute','floor','stinson','janet','head','move','kind','kent','tuesday','sheila','send','suzanne',
```

```
'brenda','kim','matter','fgt','carolyn','cindy','ccampbell','tell','fwd','crenshaw','baumbach','linda','side',
```

```
'clark','mind','hain','wharton','future','errol','carlos','hand','matt','bruce','gossett','brian','try',
```

```
'wednesday','calendar','laura','nothing','doug','llc','rebecca','rob','stephanie','austin','victor','join',
```

```
    'joseph','couple','allen','kean','arnold','var','keith','lucy','grigsby'))
```

```
    # Punctuation (formerly sometimes called pointing) is the use of spacing, conventional signs,
```

```
    # and certain typographical devices as aids to the understanding and the correct reading, both silently and aloud,
```

```
    # of handwritten and printed texts.
```

```
    exclude = set(string.punctuation)
```

```
    # Lemmatize the terms
```

```
    lemma = WordNetLemmatizer()
```

```
    # The method rstrip() returns a copy of the string in
```

```
    # which all chars have been stripped from the end of the string (default whitespace characters).
```

```
    text=text.rstrip()
```

```
    # Remaind only with letters without anything else
```

```
    text = re.sub(r'^a-zA-Z|', ' ', text)
```

```
    # Removing stopwords, digits and word lenth less than 3
```

```
    stop_free = " ".join([i for i in text.lower().split() if ((i not in stop) and (not i.isdigit()) and len(i)>2)])
```

```
    # Exclude punctuation
```

```
    punc_free = ''.join(ch for ch in stop_free if ch not in exclude)
```

```
    # Adds the terms after running Lemmatizer
```

```
    normalized = " ".join(lemma.lemmatize(word) for word in punc_free.split())
```

```

    # Building tokens from the words to remain only the common nouns
as terms
    tokens = nltk.word_tokenize(normalized)
    tags = nltk.pos_tag(tokens)
    nouns = " ".join(word for word,pos in tags if (pos == 'NN' and
word not in stop and len(word)>2))
    return nouns
def word_count(str):
    # Function that get string and counts each word in the string
    counts = dict()
    words = str.split()

    for word in words:
        if word in counts:
            counts[word] += 1
        elif(len(word)>2):
            counts[word] = 1

    return counts

def get_text_from_email(msg):
    # Function that gets email and return the content as list of
strings
    '''To get the content from email objects'''
    parts = []
    for part in msg.walk():
        if part.get_content_type() == 'text/plain':
            parts.append( part.get_payload().lower() )
    return ''.join(parts)

inputrows = 510000

# Load the CSV
# emails_df = pd.read_csv("../input/emails.csv" ,nrows=inputrows)
from google.colab import drive
drive.mount('/content/drive')
path = '/content/drive/My Drive/BE/Sem 8/ML/IA-2/emails.csv'
emails_df = pd.read_csv(path)
emails_df.head()
# Transform the CSV to list of strings
messages = list(map(email.message_from_string, emails_df['message']))

keys = messages[0].keys()

```

```

print("Starting making the keys...")

# Define Keys as headers
for key in keys: emails_df[key] = [doc[key] for doc in messages]

text_clean=[]
subject_Clean=[]
emails=0

# Load the emails content to list of strings
emails_df['content'] = list(map(get_text_from_email, messages)) #
Refer to content

print("Starting looping...")

for i in range(1,inputrows):
    # Building two lists of strings from the emails contents and
    subjects, only from the sent items
        if(emails_df['X-Folder'][i] and "sent" in
emails_df['X-Folder'][i]):
            text_clean.append(clean(emails_df['content'][i].split("to:
") [0]))
            subject_Clean.append(clean(emails_df['Subject'][i]))
            emails+=1

text_clean = re.sub(r'^a-zA-Z', ' ', str(text_clean))
subject_Clean = re.sub(r'^a-zA-Z', ' ', str(subject_Clean))

print(Counter(word_count(str(text_clean))).most_common(100))
print("- - - - -")
print("In Subjects")
print("- - - - -")
print(Counter(word_count(str(subject_Clean))).most_common(100))
print("Total emails = ",emails)

wordcloud = WordCloud(
    max_font_size=50,
    background_color='white',
    max_words=100
).generate(str(text_clean))
fig = plt.figure(1)
plt.imshow(wordcloud)

```

```
plt.axis('off')
plt.show()
fig.savefig("Body")
```

```
wordcloud = WordCloud(  
    max_font_size=50,  
    max_words=100  
).generate(str(subject_Clean))  
fig = plt.figure(1)  
plt.imshow(wordcloud)  
plt.axis('off')  
plt.show()  
fig.savefig("Subjects")
```

```
Starting looping...
```

```
[('deal', 3431), ('gas', 1615), ('price', 1209), ('contract', 1127),
```

In Subjects

```
[('deal', 827), ('agreement', 703), ('gas', 531), ('report', 469), ('
Total emails = 28841
```



Results:

The analysis of all emails has been performed and verified. The anomaly detection was performed using methods like local outlier factor and Isolation forests. It allowed detection of specific hours which could indicate malicious activity as tasked. Similarly social network analysis and main contributors to Enron's social network were identified and visualised as shown. Emails were also classified as discussed above.

Future work:

The analysis done above provided very deep insights into the scandal and all associated data. This analysis can be taken further and be used with more complex combinations of anomaly detection methods and social network analysis. Special features can be extracted based on various classes of emails. Even content based features can be used for performing anomaly detection.