

Group members:

Anthony Tabet

Elias Antoine Dargham

Perla Saikaly

Rim Kfoury

Graph theory and Operational Research

Project Report

Table of Contents

[**Section 1: Enron Dataset** 3](#_Toc154216596)

[1.1 Understanding the Enron Dataset 3](#_Toc154216597)

[1.2 Significance of the Enron Dataset in Graph Theory 3](#_Toc154216598)

[**Section 2: Neo4j** 4](#_Toc154216599)

[**Section 3: Handling the Enron dataset** 4](#_Toc154216600)

[**Section 4: Queries Used** 8](#_Toc154216601)

[**Section 5: Conclusion** 8](#_Toc154216602)

[**Section 6: References** 8](#_Toc154216603)

# **Section 1: Enron Dataset**

Enron Corporation was an American energy, commodities, and services company that became infamous for one of the most notorious corporate fraud cases in history. Founded in 1985 and headquartered in Houston, Texas, Enron was initially involved in energy trading, natural gas, electricity, and communications.

The Enron dataset refers to a collection of emails and financial data acquired from the Enron Corporation, a company that infamously collapsed in 2001 due to widespread corporate fraud. This dataset is highly regarded in research and analytical circles for its comprehensive insights into corporate communication, financial records, and unethical business practices.

## 1.1 Understanding the Enron Dataset

The dataset primarily comprises approximately 500,000 emails exchanged among Enron employees, encompassing a wide spectrum of topics such as business dealings, personal interactions, financial discussions, and more. Additionally, it may include financial statements, balance sheets, and other accounting records, shedding light on Enron's financial status and the fraudulent accounting techniques employed to manipulate financial data.

## 1.2 Significance of the Enron Dataset in Graph Theory

Graph theory involves the study of graphs as mathematical structures to model relationships between entities. The Enron dataset, with its vast network of communication among employees, serves as a rich source for constructing graphs to represent relationships between individuals, departments, or topics based on email exchanges.

# **Section 2: Neo4j**

To integrate the data and have the best analysis we used Neo4j. Neo4j is a popular graph database management system that facilitates the storage and analysis of graph data. Its capabilities make it well-suited for handling interconnected datasets like the Enron emails, enabling graph-based analysis, pattern recognition, and visualization of complex relationships within the dataset. The decision to use the Enron dataset with Neo4j for a graph theory project stems from the dataset's intrinsic characteristics as a large-scale, interconnected communication network. By leveraging Neo4j's graph database functionalities, it becomes feasible to explore email relationships, identify patterns, and uncover insights into communication structures within Enron.

# 

# **Section 3: Handling the Enron dataset**

Since the Enron file we got is a csv file with just the emails, we cannot directly use Neo4j instead we had to clean and handle the data in a specific format in order to save them in the neo4j database. To access and use the needed information from the dataset, we used a python code that read these emails and saves them. First we read the csv file using the following python command:

data = pd.read\_csv(csv\_path, usecols=['message'])

data['id'] = data.index

Which gave the following result:

As seen the above result shows the first and last five rows in the csv file which gave us the message as one column. Therefore we need to extract usedful information from these messages. To do so, we used:

  data[['sender', 'recipient', 'subject', 'cc', 'body']] = data['message'].apply(

        lambda email: pd.Series(parse\_email(email))

    )

    print('Processing Emails. This may take a couple of minutes...')

    data.drop(columns=['message'], inplace=True)

    data['sender'] = data['sender'].apply(sanitize\_email\_address)

    data['recipient'] = data['recipient'].apply(filter\_recipient\_email)

    data['cc'] = data['cc'].apply(filter\_recipient\_email)

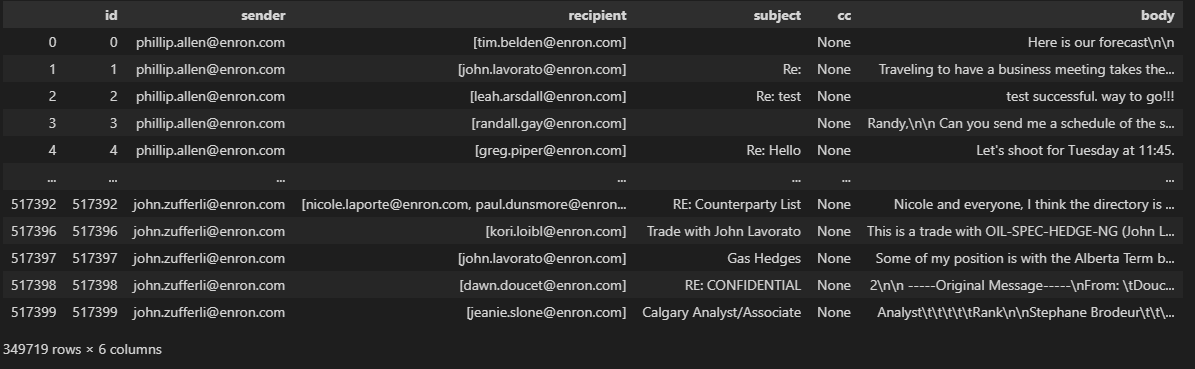
    data = data[data['recipient'] != 'None']

    data = data[data['sender'].str.contains('@enron.com')]

sanitize\_email\_address and filter\_recipient\_email are functions we wrote in order to clean and handle the information. Where “sanitize\_email\_address” takes an email address string (address) as input and aims to sanitize or clean it. The purpose of this function is to extract and return the valid email address from a given string, if it exists. If the input string does not contain a valid email address, it returns the original string. And “filter\_recipient\_email” processes a string of comma-separated email addresses, filters out the ones containing '@enron.com', sanitizes them using sanitize\_email\_address, and returns either the list of filtered email addresses or the string 'None' if there are no matches or if the input is None.

Then we took none null data as to reduce errors in nodes and lastly we made sure the emails we took are only employees by only taking emails that contain “@enron.com”. This is done to the purpose of only using a subset of the data with critical emails for analysis since what is most needed is the emails sent between employees in enron and not outsiders.

After splitting the message into useful information, we got the following data:



Column message was dropped since it was not needed anymore.

After cleaning and handling the data into the needed format. Establishing the connection between our python code and neo4j was the next step. First we created database in neo4j dbms called enron and then using the function

def connect\_to\_n4j():

    db.set\_connection(f'{config.DATABASE\_URL}/{config.DATABASE\_NAME}')

# endfunc

This function intends to set up a connection to a Neo4j database using parameters defined in the config module and using a URL derived from config.DATABASE\_URL and config.DATABASE\_NAME. To be able to establish the connection the following packages were imported “from neomodel import config, db”

config.DATABASE\_URL = app\_config['db\_connection']

config.DATABASE\_NAME = app\_config['db\_name']

 employees = set()

        for sender in data['sender'].to\_list():

            employees.add(sender)

        # endfor

        for recipients in data['recipient'].to\_list():

            for r in recipients:

                employees.add(r)

            # endfor

        # endfor

        for cced in data['cc'].to\_list():

            if cced != 'None':

                for cc in cced:

                    employees.add(cc)

                # endfor

            # endfor

        # endfor

The purpose of this code is to aggregate all unique email addresses from different columns ('sender', 'recipient', 'cc') of the DataFrame data into a set named employees. This set will contain all unique email addresses associated with this dataset,

 for employee in employees:

            ename = employee.split('@')[0]

            emp = Employee.nodes.get\_or\_none(emp\_name=ename)

            if emp is None:

                print(f'Adding Employee {ename} to database.')

                emp = Employee(emp\_name=ename, address=employee).save()

            # endif

            else:

                print(f'Employee {ename} already exists in the database (id: {emp.uid}).')

            # endelse

        #endfor

To add employees to neo4j as nodes, the following code is needed to iterate through the email addresses stored in the employees set, checks if corresponding employees exist in the database based on their names, and adds them to the database if they don't exist, or logs their existence if they are already present.

Similarly for the email message where it contain the email address as node, mid, sender recipients, subject.cc and body as attributes related to the email message.

Lastly to set relationships between the employees and email messages we added code that performs intricate operations to construct a comprehensive representation of an email communication network within a Neo4j graph database. Subsequently, it meticulously establishes relationships between these nodes to illustrate the connections in the communication network. The code handles sender, recipient, and CC details by associating 'Employee' nodes with 'EmailMessage' nodes using relationships ('SENT\_FROM', 'SENT\_TO', 'SENT\_CC'). Specifically, it identifies sender 'Employee' nodes and forms 'SENT\_FROM' relationships with corresponding 'EmailMessage' nodes, signifying the emails sent by specific individuals. For recipients and CCed individuals, it recognizes their 'Employee' nodes and creates 'SENT\_TO' and 'SENT\_CC' relationships, respectively, showcasing the recipients and CCed parties of each email. Throughout this process, the code logs the establishment of these connections, providing insights into the network's structure and email interactions. This systematic approach of node creation and relationship establishment enables the representation of email communication patterns, allowing for a detailed analysis of interactions and associations within the communication network stored in the Neo4j graph database.

Upon successfully integrating all the email communication data into Neo4j, We shifted and started working on queries to find relationships between data and figure what were the main reasons of Enron’s downfall and fraud. For the code it was a contribution between all team members however, since Neo4j did not work on all computers and some were unable to add the information from python to neo4j, we worked together and sent the python code to one computer, to Elias, in order to run the code and check its work. As for the quires we each came up and wrote several ones and tried them.

# **Section 4: Queries Used**

# **Section 5: Conclusion**

# **Section 6: References**