Initial Documentation Concerning the Exercise

1. Hardware and Language Versions
    a. This exercise was developed on a **Ubuntu 16.04.4 LTS 64-bit** machine
        i. 16GB of RAM
        ii. 8 Core Intel i7 2.9 GHz Processor
    b. Python 2.7.12
    c. PostgreSQL 9.5.12
    d. No special python packages were used. All those that were used should come standard with the default Python 2.7 installation

How to Run

1. Unzip **EnronEmailProject.zip** in the directory of your choice
2. Navigate to **/cmb/code/orchestration**
3. Run **python main.py.** This will download and unpack the tarball as well
4. Run the following SQL scripts in PostgreSQL (may require tweaking if using another Database)
    a. createDatabase.sql
    b. loadData.sql
        i. NOTE: Will require path update
    c. dataAnalysis.sql
        i. If you wish to see results
    d. dataIntegrity.sql
        i. If you wish to see a quick QC of the Users and Email table

Directory Documentation

More details can be found the code comments

1. **cmb:** root directory containing code, input, and output
2. **code**: In the code directory you will find the following sub-directories and files
    a. **Load** -- Files load the processed CSV data
        i. **createDatabase.sql** -- generates the database, schemas, and tables with their appropriate DDLs
        ii. l**oadData.sql** -- copies the data from CSV files into the appropriate tables
    b. **Orchestration** -- For processing the raw Enron Email data and implementing a structure on it
        i. **fetchAndUnzipData.py** -- fetches the data using **wget** from the link provided in the exercise. It will down and unzip the data into the **input** directory
        ii. **ingestionUtils.py** -- contains utility functions for processing the data and fetching files from the **maildir** directory
        iii. **main.py** -- Parses the email files in the **maildir** directory and writes out two CSV files. One contain Users and another containing Emails
    c. **Analytics** -- For analyzing the data loaded into the database
        i. **dataAnalysis.sql** -- SQL queries for analyzing the data
    d. **QA** -- For running tests against the data

        i. **dataIntegrity.sql** -- Checks that the data from the Users table is consistent with the Email table
3. **input**: The input directory will contain the raw **enron_email_20150507.tar.gz** file and the unpacked **maildir** directory which is where all of the raw emails are stored and categories by user inbox
4. **output**: The output directory will contain the processed User and Email CSV files that will be ingested into the database. The **main.py** script will automatically write out to this location

Data Warehouse Documentation

1. We created a database **enron** which is ASCII encoded, so that the generated CSV files may be properly loaded.
    a. Since the emails where encoded in ASCII, the encoding was maintained throughout to avoid converting to UTF-8.
    b. If we are to load more data in the future from outside sources to merge with the Enron Email data, then we may consider converting to UTF-8 initially.
2. We then create three Schemas **landing_enron, source_enron, dev_enron**
    a. landing_enron: This schema loads the data as is from the CSV files
        i. Any quality assurance or duplicate removal will be done here before pushing the data to **source_enron**
    b. source_enron: This schema contains the Enron data Users and Emails that is ready for consumption by processes that will further process the data, or run analytics on it
    c. dev_enron: This schema is intended for development by power users such as Data Engineers. This intended to be a sandbox environment where additional tests can be performed on the data, or where production datasets can be developed without exposing them to other users (assumed adequate permissions are set)
    d. Additional Schemas
        i. We certainly have the option of create more schemas for different purposes. If were to ingest data on a regular basis, grant access to different parties, and report on it, then we would want to create more schemas to organize our data warehouse. Some examples include, but are not limited to:
            1. **test_enron:** Where dev datasets are pushed for testing with reporting or machine learning. This was production datasets are not impacted by a bad change
            2. **prod_enron**: Where processed data can live for consumption by the reporting and data analysis portions of the business
            3. **analytics_enron:** Where high-level business users can run their own tests and create ad-hoc tables without interfering with the development process
    e. Final Notes

i. At some point depending on the scale of this project (users, volume of data consumed, etc.) we may want to consider scaling out to multiple data warehouses. Such as one dedicated to development, one dedicated to testing, and one dedicated to production.
    1. With the right ETL tools and organizations such a design would be relatively easy to maintain

3. We have created two different tables a **Users** and an **Emails** table (based on source_enron DDLs, landing differs for Users)
    a. Users

Column	Type	Description
email_address	TEXT NOT NULL PRIMARY KEY	User email address gathered from all address fields in the Enron Email Dataset
user_name	TEXT	Use names gathered from the X-Header columns in the Enron Email Dataset. These were not always populated, and due the nature of the data the fields could not always be parsed properly
is_employee	BOOLEAN NOT NULL	Indicates where a particular user is an Enron Employee. This was determined by parsing the email address for “enron.com”

    b. Emails

Column	Type	Description
message_id	TEXT NOT NULL PRIMARY KEY	Unique identifier for an email
sender	TEXT NOT NULL	User in the From field of an email
recipients	TEXT	User(s) in the To field of an email. This field can be populated with many users and in the event that this occurs then

		email addresses are comma separated
copied_recipents	TEXT	User(s) in the CC field of an email. This field can be populated with many users and in the event that this occurs then email addresses are comma separated
blind_copied_recipients	TEXT	User(s) in the BCC field of an email. This field can be populated with many users and in the event that this occurs then email addresses are comma separated
subject	TEXT	The subject field of an email
date	TIMESTAMP NOT NULL	The date of an email. This date was converted from a typical email format into a PostgreSQL friendly format. Additionally, time has been converted to UTC.
sender_name	TEXT	The name of the sender parsed from the X-From field of an email
recipient_names	TEXT	The names of recipients parsed from the X-To field of an email. If there is more than one recipient, then they are pipe separated.
copied_names	TEXT	The names of recipients parsed from the X-cc field of an email. If there is more than one recipient, then they are pipe separated.

blind_copied_names	TEXT	The names of recipients parsed from the X-bcc field of an email. If there is more than one recipient, then they are pipe separated.
body	TEXT	The body of an email
is_forwarded	BOOLEAN NOT NULL	Indicates if an email was forwarded from a user. This field may be subject to some inaccuracy since we had to parse the emails for the ‘fw’ or ‘fwd’ tag.
is_chain	BOOLEAN NOT NULL	Indicates if an email is parse of a chain. This was determined by checking for the presence of the “Original Message” tag in an email body. Subject to inaccuracy due the nature of the parsing.
chain_count	INT NOT NULL	The count of the depth of an email chain Subject to inaccuracy due to the nature of the parsing.
source_folder	TEXT NOT NULL	The source file of the email relative to the location of the underlying folder structure of there raw data is housed.

Testing

1. The main tests can be found in the **createDatabase.sql** file and in the **dataIntegrity.sql** file
    a. The DDL design of the tables helps us check for duplicates, and critical null values at load time and catch them before ingesting the data

    b. The data integrity file runs a check for missing users in the Users table because we assume that every sender in the Emails table should be present in the Users table as they are all pulled from the same source
        i. This check actually made me realize the was a bug in my code that I needed to correct
2. Additional tests we'd want to implement would be:
    a. **Control Totals**: Did we actually capture all of the emails in the **maildir** directory?
    b. **Parser Checks**: Did we capture the right fields using the parser?
        i. Goes in line with checking for malformed data in the raw files such as missing headers, or characters that could confuse the parser

Results Summary

More details can be found in the **dataAnalysis.sql** file

1. Total Users: 71,813
2. Total Enron Users after Dup Removal: 29,342
3. Total Emails: 517,401
4. Top Sender: Kay Mann
5. Total Chains: 76,138
6. Total Chain Depth: 140,831
7. Total Forwards: 49,576
8. Emails from Enron Employees: 83%

Future Considerations

1. Given more time on this project we could consider implementing the following
    a. Fuzzy Matching on the initial Users table to determine which User to Email Address mapping is the most appropriate to keep
    b. Using something like SQLAlchemy to automate the ingestion of the data from Python into the Data Warehouse
    c. Creating an additional Emails table that has the To, CC, BCC fields spread across multiple rows
        i. E.g., insead of: **a -> b, c, d** we have:
            1. **a -> b**
            2. **a -> c**
            3. **a -> d**
        ii. This would enable us to do this things like social network analysis because we've create a relationship table that can be used as an edge table
    d. Parsing the email bodies and pulling out additional users, emails, and conversations from email chains
    e. Taking an Object Oriented approach to the code design
    f. Improving the multiprocessing aspect of the code write out CSV results much faster
    g. Improved testing and checks against the raw data

2. If we were instead ingesting Terabytes of data we would have to consider the following:
    a. Initial Processing Platform
        i. To process Terabytes of e-mail data we would need to make use of Multi-Node and Multi-Core processing, so we could look towards a technology such as Databricks to handle the initial processing of the raw data
    b. Raw File Storage
        i. We would need a place to land the raw data and store the processed CSV files, so something S3 could address our needs for storage since it can easily integrate with Databricks and most major Data Warehouse solutions
    c. Data Warehousing
        i. To actually store and query the data we've processed we'd need a more performant and scalable solution such as Amazon Redshift. This way we can ingest, store, and query data in a way that is easily accessible by users
    d. ETL Tool
        i. If we were to load this data on a regular basis we might want to consider a Third Party ETL tool, so that we easily set-up automation, QCs, and scale as the data volume grows
    e. Data Model
        i. We might also want to consider a different data model so that users are not always having to query a massive table of records
            1. For instance, we could divide up the Users and Email tables by Enron departments and external users