CAPSTONE PROJECT 1: DATA WRANGLING

Our Problem:
Can emails be used to identify the author's gender?

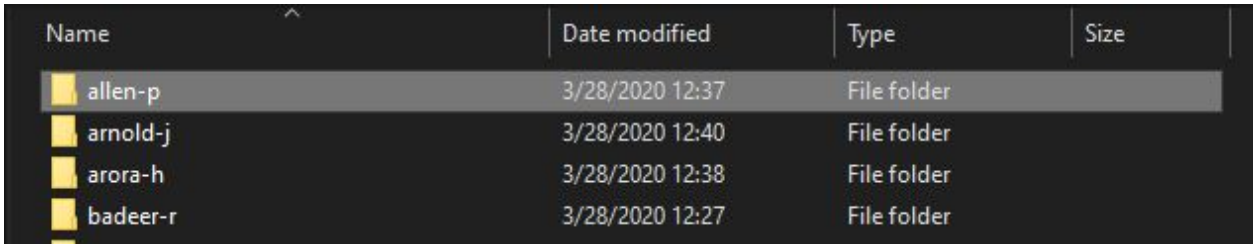
Our Dataset:
Enron Email Corpus

Approach:
To solve the problem, we'll need to arrive at a ***set of emails***, where each email can be associated with the ***gender of the author***.

SECTION 00: Acquiring the Data

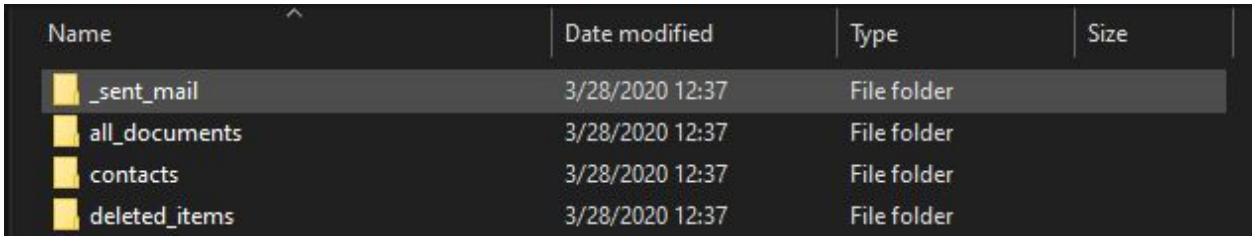
Exploring the Files
Data is downloaded from the Carnegie Mellon University School of Computer Science as a collection of *employee folders*. Each employee folder contains a collection of unstructured *subfolders*. Within the subfolders and subsequent subfolders are the email files, saved without type.

Employee folders:



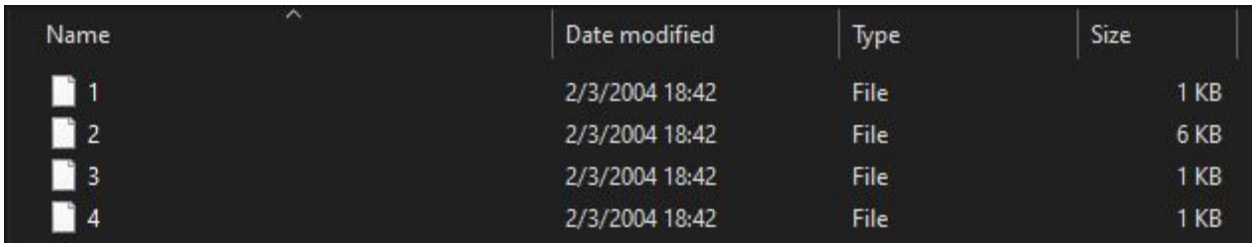
Name	Date modified	Type	Size
allen-p	3/28/2020 12:37	File folder	
arnold-j	3/28/2020 12:40	File folder	
arora-h	3/28/2020 12:38	File folder	
badeer-r	3/28/2020 12:27	File folder	

Employee subfolders:



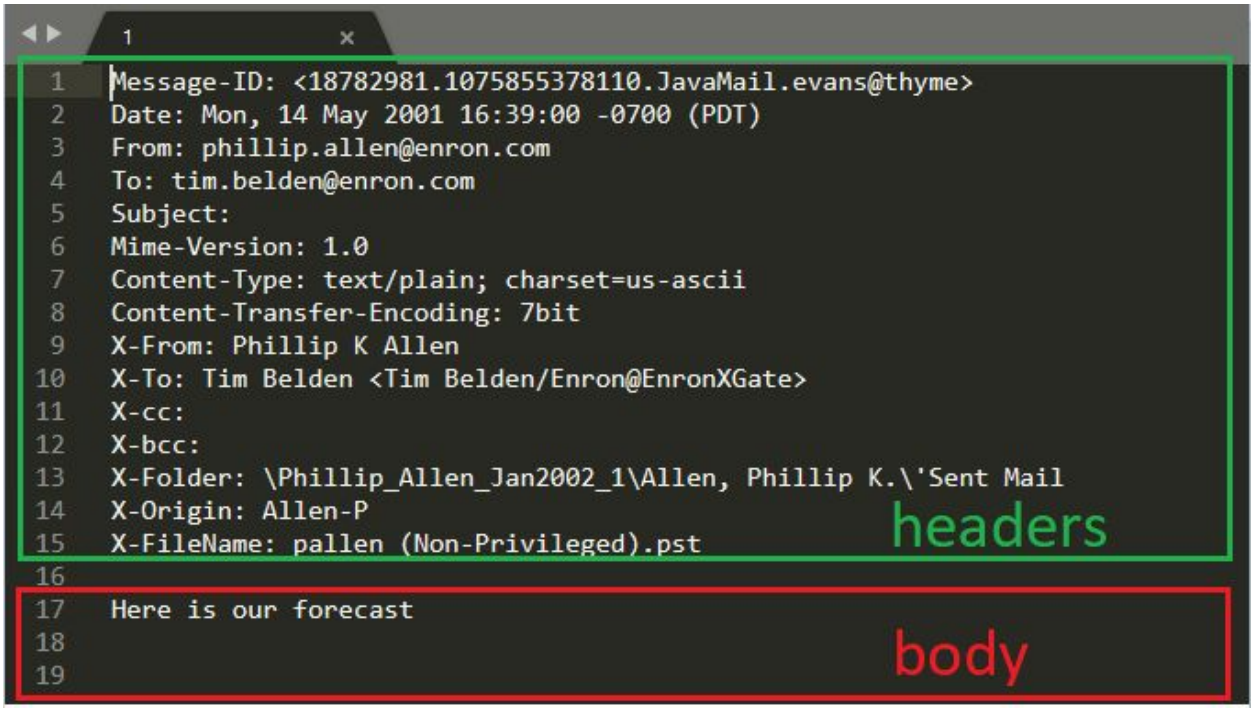
Name	Date modified	Type	Size
_sent_mail	3/28/2020 12:37	File folder	
all_documents	3/28/2020 12:37	File folder	
contacts	3/28/2020 12:37	File folder	
deleted_items	3/28/2020 12:37	File folder	

Email files:



Name	Date modified	Type	Size
1	2/3/2004 18:42	File	1 KB
2	2/3/2004 18:42	File	6 KB
3	2/3/2004 18:42	File	1 KB
4	2/3/2004 18:42	File	1 KB

Email contents:



Recovering the Files

Files were recovered as a dictionary, with the file name as the dictionary key and the email contents as the dictionary value. Files were recovered in two steps:

- Collect a list of email filenames;
- Loop through the list and recover the email text;

During recovery, *Email contents* were separated into *headers* and the email *body* using regular expressions, with each header value collected separately.

DataFrame

The resulting DataFrame contains 517401 entries, with 19 data columns (file directory + headers + email body.

Missing values were left as ‘’ and subsequently filled as Nan values during import/exporting the DataFrame between notebooks with **pandas**.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 517401 entries, 0 to 517400
Data columns (total 19 columns):
f_dir        517401 non-null object
m_id         517401 non-null object
m_date       517401 non-null object
m_from       517401 non-null object
m_to         496355 non-null object
m_cc         135166 non-null object
m_bcc        127886 non-null object
m_subj       498214 non-null object
mime_vers    517401 non-null object
cont_type    517401 non-null object
encode       517401 non-null object
x_from       517401 non-null object
x_to         508255 non-null object
x_cc         128886 non-null object
x_bcc        175 non-null object
x_fold       517401 non-null object
x_orig       517401 non-null object
x_fname      517401 non-null object
m_body       517401 non-null object
dtypes: object(19)
```

SECTION 01: First Clean

Character count: 954,786,906
Entry count: 517,401

The dataset will be cleaned multiple times, as each cleaning process can subsequently create the need to repeat a cleaning step. For example, after *RE:* and *FW:* email bodies are cleaned, some emails return empty email bodies.

Steps to clean the initial dataset are as follows.

Remove Whitespace
Whitespaces were stripped from each entry in the DataFrame using ***str.lstrip()*** and ***str.rstrip()*** passed to ***pd.DataFrame.apply()***.

Remove Redundant Characters in f_dir
When the files were recovered, the string, ‘./data/maildir/’ was used in locating files. This has been removed from all of the **f_dir** entries with ***str.replace()***.

Remove Duplicate Email Bodies
269530 duplicate emails are returned when checking ***pd.DataFrame.duplicated()***. Duplicates are removed using ***pd.DataFrame.drop_duplicates()***.

Remove non-Enron Email Addresses (@enron.com)
Non-Enron email addresses are removed by selecting the DataFrame subset that returns using ***str.endswith(‘@enron.com’)*** on the **m_from** column.

Remove RE: FW: Repeated Body Text
When users reply or forward emails, the original email text is included in the body. The original email text was removed from all emails it occurred in, keeping the user-entered email text as the email body.

First, null values are replaced with blank strings (‘’) to prevent a Nan array error from ***pd.DataFrame.apply()***.

Next, a function was created to apply ***str.split()*** method on each email body and return the first value. Strings used were:

- '-----Original Message-----'
- '----- Forwarded by'

The entire **m_body** column was passed to this function using ***pd.DataFrame.apply()***.

Remove \n Break Characters from Email Text
The '\n' character was replaced with (' ') within the email text by applying ***re.sub()*** to the **m_body** column using ***pd.DataFrame.apply()***.

Drop Duplicates and Empty Values from Email Text
Duplicates are again removed using ***pd.DataFrame.drop_duplicates()***. Empty values are removed by selecting the fields that do not return **== ''**.

Section 01 Review
Missing values were removed from the corpus to help reduce the overall size of the dataset. Outliers relative to word/character count were not addressed.

SECTION 02: Adding Gender

Character count: 135,952,755
Entry count: 169,285

To wrangle gender, first names were recovered from either the email address or the X-From column. Next, they were passed to the website www.gpeters.com/names/baby-names, a webpage that using google search returns to guess the gender of a name.

This occurred with the following steps.

Subset Relevant Data

The columns m_from (emails) and x_from (internal sender designation) were copied to return names using ***pd.DataFrame.copy()***.

Lower Case

The case of all values was lowered with ***str.lower()***.

Entry sample:

	m_from	x_from
0	phillip.allen@enron.com	phillip k allen
1	ina.rangel@enron.com	ina rangel
2	critical.notice@enron.com	critical.notice@enron.com
3	rebecca.cantrell@enron.com	rebecca w cantrell
4	paul.kaufman@enron.com	paul kaufman

Recover Names

Name patterns were recovered using ***re.search()***. 8 patterns were used with **m_from**, and 14 with **x_from**. A list of 67 stopwords was also used to filter out senders. Stopwords triggered the query to return (‘’).

Stop words:

```
['team', 'technology', 'security', 'enron', 'chairman', 'office', 'announcement',
 'wizard', 'notice', 'address', 'hr taylor', 'coo jeff', 'infrastructure', 'ubsw', 'europe',
 'human', 'resources', 'connection', 'ibuyit', 'users', 'livelink', 'registrar', 'global',
 'business', 'compensation', 'executive', 'risk', 'analytics', 'daemon', 'information',
 'management', 'helpdesk', 'project', 'sunrise', 'oncall', 'credit', 'union', 'notification',
 'central', 'communication', 'center', 'parking', 'transportation', 'international',
 'diversity', 'survey', 'automation', 'document', 'exec', 'iscinfra', 'public', 'relations',
 'controls', 'exchange', 'cms router', 'hotline', 'admin', 'pr id', 'expertfinder',
 'notes', 'the buzz', 'gpg dss', 'xi xi', 'enw piper', 'institute', 'agent', 'tarrif']
```

Name results:

	m_from	m_from_cleaned	x_from	x_from_cleaned
0	phillip.allen@enron.com	phillip allen	phillip k allen	phillip allen
1	ina.rangel@enron.com	ina rangel	ina rangel	ina rangel
2	critical.notice@enron.com		critical.notice@enron.com	
3	rebecca.cantrell@enron.com	rebecca cantrell	rebecca w cantrell	rebecca cantrell
4	paul.kaufman@enron.com	paul kaufman	paul kaufman	paul kaufman

Remove Blanks

Blanks represent entries that did not return a gender-identifiable name (ex. Initials, etc.). Blanks were removed by selecting entries where both cleaned columns **== ‘’**.

Evaluate Conflicts Between Returns

In 32 entries, one name was returned from the **m_from** column, while a different name was returned from the **x_from** column. Both names returned the same last name.

	m_from	m_from_cleaned	x_from	x_from_cleaned
168	buckner.thomas@enron.com	buckner thomas	thomas, john buckner </o=enron/ou=na/cn=recipi...	john thomas
220	jae.black@enron.com	jae black	black, tamara jae </o=enron/ou=na/cn=recipient...	tamara black
338	pinto.leite@enron.com	pinto leite	leite, francisco pinto </o=enron/ou=na/cn=reci...	francisco leite
1061	dana.davis@enron.com	dana davis	davis, mark dana </o=enron/ou=na/cn=recipients...	mark davis
1096	kay.miller@enron.com	kay miller	miller, mary kay </o=enron/ou=na/cn=recipients...	mary miller

Names were set manually using ***.loc[]*** **==** to one of the two returned options. Designations were made based on two considerations:

- Tested gender return
- Looking up the person online

Separate the First Name for Scraping

First names were isolated from the returned names using ***.str.extract()***. These names were saved to a new column, **gender_query**.

Remove Hyphen From Strings

Hyphens did not return a gender when passed to the website despite returning gender when passed without a hyphen. Hyphen names were located using ***.str.contains()***. From there, names were replaced with one of the contained names.

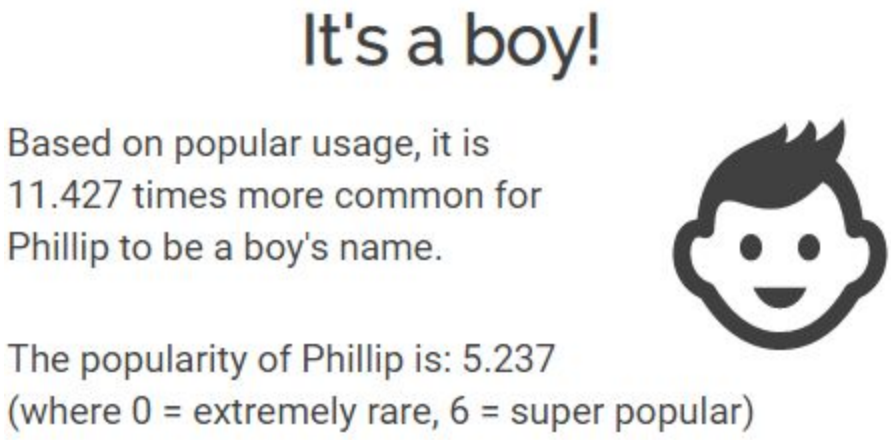
```
cd.loc[cd['gender_query'] == 'sarah-joy', 'gender_query'] = 'sarah'
```

Drop Duplicate Names
Since we need to pass each name to the website to return gender only once, a list of 1699 unique names was created using ***pd.DataFrame.drop_duplicates()***.

Web Scraping the Gender
Names were passed to the website www.gpeters.com/names/baby-names using ***requests_html*** to the ***?name=*** URL argument (https://www.gpeters.com/names/baby-names.php**?name=phillip**).

Gender was parsed from the HTML return text using ***html.search()*** to match the pattern, "It's a {}!"

Webpage return:



HTML return:

```
95      <!-- begin result-1 table -->
96      <div class="gender-result">
97          <h1>It's a boy!</h1>
98          <div class="genderimgwrapper"><img class="genderimage"
99          <!-- icons thanks to iconmonstr.com https://iconmonstr
```

61 of the 1699 names did not return a gender.

Gender Sample
A sample was taken from the gender dataset to evaluate the accuracy of gender returns relative to the actual employee gender using ***pd.DataFrame.sample(n=100, random_state=1)***.

SECTION 03: Cleaning Email Body

Character count: 135,952,755
Entry count: 169,285

Email bodies were cleaned previously in the first cleaning stage by:

- Removing duplicates
- Removing repeat text from replies, forwards

The following additional steps were taken to clean email bodies:

Remove Null, Not Found Gender

6,871 entries from the main dataset that returned a null value or 'not-found,' were removed.

Term Frequency, Inverted Document Frequency

A TF-IDF model was built to identify email duplicates that might be present but a few characters off. This included steps:

- Preprocessing
- Calculate document frequency
- Calculate tf-idf scores

Preprocessing

Preprocessing the body text for TF-IDF included X processes applied for a total of 10 steps:

- Lowercase the dataset
- Remove punctuation marks, except apostrophe
- Remove apostrophe
- Remove single characters
- Convert numbers to text
- Remove stopwords
- Lemmatise text
- Stem text
- Remove punctuation marks
- Convert numbers to text

To tokenize words, the ***nltk.tokenize*** method was used. Stopwords were collected from the ***nltk.corpus()***.

Lemmatisation was done using the ***nltk.stem WordNetLemmatizer*** and the ***nltk.corpus wordnet*** collection.

Stemming was done using the ***nltk.stem.snowball SnowballStemmer***.

Numbers were converted to text using ***num2words***.

Calculating Document Frequency
DF was calculated by passing the entire **m_body** corpus, post-processing, to a function. Inside the function, each email is tokenized, and words are added to a dictionary with their index as a value. Subsequent occurrences add their index position to the word-key.

Once all documents are passed, the dictionary counts the number of values for each word-key, and reassigns the count as the value. The resulting dictionary contains each word and its related document frequency.

Calculating TF-IDF
To calculate TF-IDF, each document is passed to a function. The function calculates the TF (frequency of unique word within document / number of words within document). Next, the function calls the document frequency and calculates the IDF with log(N/(DF+1)), where N is the total number of documents in the corpus.

The TF and IDF are multiplied together to calculate the TF-IDF score for each word relative to each document, and saved in a dictionary with the TF-IDF score.

Calculating Cosine Similarity
Cosine Similarity is calculated by taking the vectorized TF-IDF scores for two bodies of text, calculating dot multiplication in the numerator, and multiplying the normalized values in the denominator.

Using TF-IDF, Cosine Similarity to Remove Duplicates
Email bodies can be compared using Cosine Similarity to return email body pairs of higher similarity.

SECTION 04: For Future Consideration

Original Sender
No current process addresses associating the email body with the original sender explicitly. For example, if an email is forwarded and no additional text is added, an issue can arise when duplicates are dropped.

When dropping duplicates, no specific care was taken to identify the original sender - subsequently, the remaining email might not be associated with the original sender.

This might be avoided by taking the entries that do not return during a ***pandas .duplicated()*** return. However, the original, 'non-duplicate,' will also not be selected.
df = df[~pd.DataFrame.duplicated]
df = pd.DataFrame.drop_duplicates()

RE: FW: Significance
During the first major cleaning, reply and forward email body text was removed to reduce the overall size of the corpus for later processing. However, some significance might be observed from associating the amount of reply/forward emails relative to gender.

Calculating Document Frequency
During the DF calculation, the corpus is passed one document at a time. Because the DF dictionary is compiled using the word's index position within the document, there's a small chance that words in the exact same index position in two different documents would not register the occurrence of that word.

Though probably not significant, this can be corrected by combining all emails into one text body prior to processing.

Addressing Outliers
Currently, the average amount of words per sentence is 15-20 (The Acropolitan, 2017). Outliers on the lower range might be considered for corpus reduction based on word/character counts.

TF-IDF/CS Insights
Multiple returns show duplicate emails not being captured as a result of inner whitespace character position differences. Should include a ***re.replace()*** [/s]+ pattern to clean email bodies.

References

The Acropolitan. (2017). *Sentence Length Has Declined 75% in the Past 500 Years.* Retrieved from: https://medium.com/@theacropolitan/sentence-length-has-declined-75-in-the-past-500-years-2e40f80f589f