**Data Cleaning**

**Introduction**

This notebook goes through a necessary step of any data science project - data cleaning. Data cleaning is a time consuming and unenjoyable task, yet it's a very important one. Keep in mind, "garbage in, garbage out". Feeding dirty data into a model will give us results that are meaningless.

Specifically, we'll be walking through:

1. **Getting the data -**in this case, we'll be scraping data from a website
2. **Cleaning the data -**we will walk through popular text pre-processing techniques
3. **Organizing the data -**we will organize the cleaned data into a way that is easy to input into other algorithms

The output of this notebook will be clean, organized data in two standard text formats:

1. **Corpus** - a collection of text
2. **Document-Term Matrix** - word counts in matrix format

**Problem Statement**

As a reminder, our goal is to look at transcripts of various comedians and note their similarities and differences. Specifically, I'd like to know if Ali Wong's comedy style is different than other comedians, since she's the comedian that got me interested in stand up comedy.

**Getting The Data**

Luckily, there are wonderful people online that keep track of stand up routine transcripts. [Scraps From The Loft](http://scrapsfromtheloft.com/) makes them available for non-profit and educational purposes.

To decide which comedians to look into, I went on IMDB and looked specifically at comedy specials that were released in the past 5 years. To narrow it down further, I looked only at those with greater than a 7.5/10 rating and more than 2000 votes. If a comedian had multiple specials that fit those requirements, I would pick the most highly rated one. I ended up with a dozen comedy specials.

*# Web scraping, pickle imports*

**import** **requests**

**from** **bs4** **import** BeautifulSoup

**import** **pickle**

*# Scrapes transcript data from scrapsfromtheloft.com*

**def** url\_to\_transcript(url):

*'''Returns transcript data specifically from scrapsfromtheloft.com.'''*

page = requests.get(url).text

soup = BeautifulSoup(page, "lxml")

text = [p.text **for** p **in** soup.find(class\_="post-content").find\_all('p')]

print(url)

**return** text

*# URLs of transcripts in scope*

urls = ['http://scrapsfromtheloft.com/2017/05/06/louis-ck-oh-my-god-full-transcript/',

'http://scrapsfromtheloft.com/2017/04/11/dave-chappelle-age-spin-2017-full-transcript/',

'http://scrapsfromtheloft.com/2018/03/15/ricky-gervais-humanity-transcript/',

'http://scrapsfromtheloft.com/2017/08/07/bo-burnham-2013-full-transcript/',

'http://scrapsfromtheloft.com/2017/05/24/bill-burr-im-sorry-feel-way-2014-full-transcript/',

'http://scrapsfromtheloft.com/2017/04/21/jim-jefferies-bare-2014-full-transcript/',

'http://scrapsfromtheloft.com/2017/08/02/john-mulaney-comeback-kid-2015-full-transcript/',

'http://scrapsfromtheloft.com/2017/10/21/hasan-minhaj-homecoming-king-2017-full-transcript/',

'http://scrapsfromtheloft.com/2017/09/19/ali-wong-baby-cobra-2016-full-transcript/',

'http://scrapsfromtheloft.com/2017/08/03/anthony-jeselnik-thoughts-prayers-2015-full-transcript/',

'http://scrapsfromtheloft.com/2018/03/03/mike-birbiglia-my-girlfriends-boyfriend-2013-full-transcript/',

'http://scrapsfromtheloft.com/2017/08/19/joe-rogan-triggered-2016-full-transcript/']

*# Comedian names*

comedians = ['louis', 'dave', 'ricky', 'bo', 'bill', 'jim', 'john', 'hasan', 'ali', 'anthony', 'mike', 'joe']

*# # Actually request transcripts (takes a few minutes to run)*

*# transcripts = [url\_to\_transcript(u) for u in urls]*

*# # Pickle files for later use*

*# # Make a new directory to hold the text files*

*# !mkdir transcripts*

*# for i, c in enumerate(comedians):*

*# with open("transcripts/" + c + ".txt", "wb") as file:*

*# pickle.dump(transcripts[i], file)*

*# Load pickled files*

data = {}

**for** i, c **in** enumerate(comedians):

**with** open("transcripts/" + c + ".txt", "rb") **as** file:

data[c] = pickle.load(file)

*# Double check to make sure data has been loaded properly*

data.keys()

*# More checks*

data['louis'][:2]

**Cleaning The Data**

When dealing with numerical data, data cleaning often involves removing null values and duplicate data, dealing with outliers, etc. With text data, there are some common data cleaning techniques, which are also known as text pre-processing techniques.

With text data, this cleaning process can go on forever. There's always an exception to every cleaning step. So, we're going to follow the MVP (minimum viable product) approach - start simple and iterate. Here are a bunch of things you can do to clean your data. We're going to execute just the common cleaning steps here and the rest can be done at a later point to improve our results.

**Common data cleaning steps on all text:**

* Make text all lower case
* Remove punctuation
* Remove numerical values
* Remove common non-sensical text (/n)
* Tokenize text
* Remove stop words

**More data cleaning steps after tokenization:**

* Stemming / lemmatization
* Parts of speech tagging
* Create bi-grams or tri-grams
* Deal with typos
* And more...

*# Let's take a look at our data again*

next(iter(data.keys()))

*# Notice that our dictionary is currently in key: comedian, value: list of text format*

next(iter(data.values()))

*# We are going to change this to key: comedian, value: string format*

**def** combine\_text(list\_of\_text):

*'''Takes a list of text and combines them into one large chunk of text.'''*

combined\_text = ' '.join(list\_of\_text)

**return** combined\_text

*# Combine it!*

data\_combined = {key: [combine\_text(value)] **for** (key, value) **in** data.items()}

*# We can either keep it in dictionary format or put it into a pandas dataframe*

**import** **pandas** **as** **pd**

pd.set\_option('max\_colwidth',150)

data\_df = pd.DataFrame.from\_dict(data\_combined).transpose()

data\_df.columns = ['transcript']

data\_df = data\_df.sort\_index()

data\_df

*# Let's take a look at the transcript for Ali Wong*

data\_df.transcript.loc['ali']

*# Apply a first round of text cleaning techniques*

**import** **re**

**import** **string**

**def** clean\_text\_round1(text):

*'''Make text lowercase, remove text in square brackets, remove punctuation and remove words containing numbers.'''*

text = text.lower()

text = re.sub('\[.\*?\]', '', text)

text = re.sub('[**%s**]' % re.escape(string.punctuation), '', text)

text = re.sub('\w\*\d\w\*', '', text)

**return** text

round1 = **lambda** x: clean\_text\_round1(x)

*# Let's take a look at the updated text*

data\_clean = pd.DataFrame(data\_df.transcript.apply(round1))

data\_clean

*# Apply a second round of cleaning*

**def** clean\_text\_round2(text):

*'''Get rid of some additional punctuation and non-sensical text that was missed the first time around.'''*

text = re.sub('[‘’“”…]', '', text)

text = re.sub('**\n**', '', text)

**return** text

round2 = **lambda** x: clean\_text\_round2(x)

*# Let's take a look at the updated text*

data\_clean = pd.DataFrame(data\_clean.transcript.apply(round2))

data\_clean

**NOTE:** This data cleaning aka text pre-processing step could go on for a while, but we are going to stop for now. After going through some analysis techniques, if you see that the results don't make sense or could be improved, you can come back and make more edits such as:

* Mark 'cheering' and 'cheer' as the same word (stemming / lemmatization)
* Combine 'thank you' into one term (bi-grams)
* And a lot more...

## Organizing The Data

I mentioned earlier that the output of this notebook will be clean, organized data in two standard text formats:

1. **Corpus -**a collection of text
2. **Document-Term Matrix -**word counts in matrix format

### Corpus

We already created a corpus in an earlier step. The definition of a corpus is a collection of texts, and they are all put together neatly in a pandas dataframe here.

*# Let's take a look at our dataframe*

data\_df

*# Let's add the comedians' full names as well*

full\_names = ['Ali Wong', 'Anthony Jeselnik', 'Bill Burr', 'Bo Burnham', 'Dave Chappelle', 'Hasan Minhaj',

'Jim Jefferies', 'Joe Rogan', 'John Mulaney', 'Louis C.K.', 'Mike Birbiglia', 'Ricky Gervais']

data\_df['full\_name'] = full\_names

data\_df

*# Let's pickle it for later use*

data\_df.to\_pickle("corpus.pkl")

### **Document-Term Matrix**

For many of the techniques we'll be using in future notebooks, the text must be tokenized, meaning broken down into smaller pieces. The most common tokenization technique is to break down text into words. We can do this using scikit-learn's CountVectorizer, where every row will represent a different document and every column will represent a different word.

In addition, with CountVectorizer, we can remove stop words. Stop words are common words that add no additional meaning to text such as 'a', 'the', etc.

*# We are going to create a document-term matrix using CountVectorizer, and exclude common English stop words*

**from** **sklearn.feature\_extraction.text** **import** CountVectorizer

cv = CountVectorizer(stop\_words='english')

data\_cv = cv.fit\_transform(data\_clean.transcript)

data\_dtm = pd.DataFrame(data\_cv.toarray(), columns=cv.get\_feature\_names())

data\_dtm.index = data\_clean.index

data\_dtm

*# Let's pickle it for later use*

data\_dtm.to\_pickle("dtm.pkl")

*# Let's also pickle the cleaned data (before we put it in document-term matrix format) and the CountVectorizer object*

data\_clean.to\_pickle('data\_clean.pkl')

pickle.dump(cv, open("cv.pkl", "wb"))

**Additional Exercises**

1. Can you add an additional regular expression to the clean\_text\_round2 function to further clean the text?
2. Play around with CountVectorizer's parameters. What is ngram\_range? What is min\_df and max\_df?