# Capstone1: Write up for Data Wrangling

The purpose of this document is to explain the steps taken to clean and process the Capstone project data set and some of the findings from the process. A number of sources were used to support this work and they have been are attributed in the final section of the report.

## ## Steps

1. Set up jupyter notebook - `data\_wrangling.ipynb`

2. Load in data for reviews and businesses

3. Exploratory analysis

4. Reformat column for date into data-time category

5. Extract businesses that are restaurants

6. Extract restaurants that are in the USA using a bounding box

7. Drop business columns associated with business location

8. Merge the two dataframes

9. Explore merged dataframe

10. Pre-processing the data:

- Case conversion

- Punctuation removal

- Removal of stop words

- Lemmatization

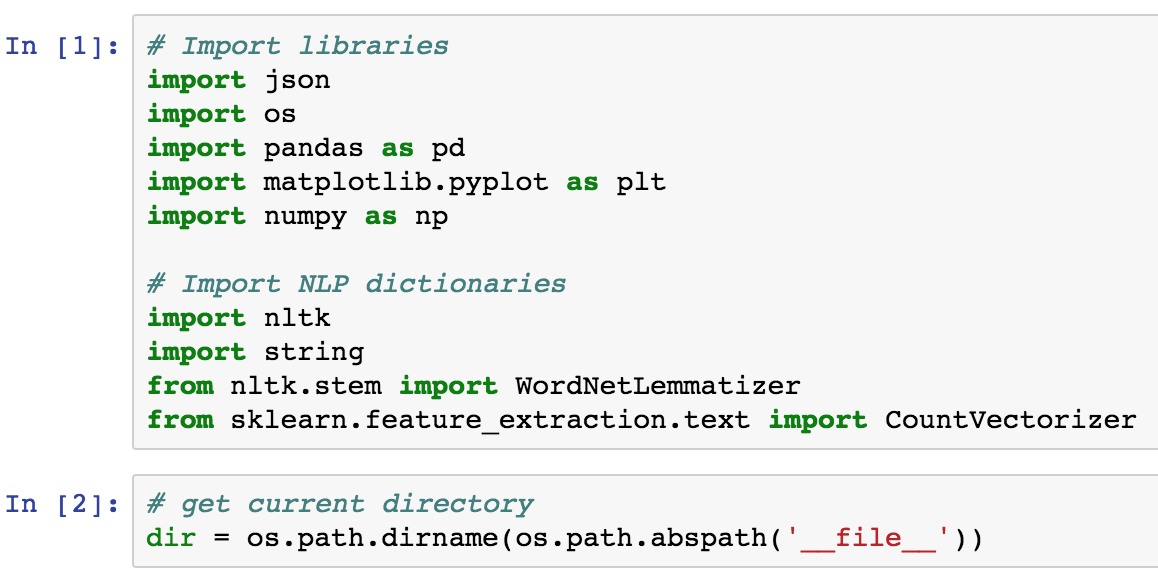
11. Vectorise text

12. Saving the output

### ### 1. Set up jupyter notebook

The data wrangling stage can be found a Jupyter notebook called `data\_wrangling.ipynb`. It will contain the key steps laid out in this document. The inputs are described in step 2, and the outputs are described in step 13.

As part of the set up, the library dependencies are loaded.



In particular:

The common DataScience libraries

- json for loading in json files

- os for non-OS dependent file paths

- pandas for manipulation of DataFrames

- matplotlib.pyplot for plots and simple visualizations

- numpy to handle simple data arrays

As well as more specific NLP libraries

- nltk for stopwords, the lemmerization,

- string to get the library of punctuations

- sklearn.feature\_extraction.text for the string to word vectorizer

The working directory is also generated for future reference.

### ### 2. Load in data

## Source of data

The dataset was obtained as part the '[Yelp dataSet Challenge - Round 10](https://www.yelp.com.sg/dataset)' (September 1, 2017 to December 31, 2017)

In particular, the `review.json` and `business.json` files from the JSON dataset.

These are currently stored in the folder labeled ` 01\_raw\_data`. The terms and conditions of using the dataset limits sharing of the raw data and it shall be excluded in the git repo.

The raw data in the original document comes in the format of dictionaries separated by lines:

```

{}

{}

{}

```

To load the data, first a connection is opened to the file, then the data is read using a list comprehension and finally the list is converted into a DataFrame. A similar process used to load business data.



### ### 3. Exploratory analysis

To understand the data set, 3 functions were used: `.shape`, `.info`, `.describe`. Then a null check was performed. And finally, a head of 50 was used to eyeball the data set.

For the review data, the following observations were made:

1. There were no null values

2. Except for the data, the data types from the column were correct

3. The full data set had ~ 4.7 million observations

4. Non-restaurant businesses were also reviewed (e.g. accommodation)

5. Reviews were not all in English

For the business data, the following observations were made:

1. There were only 2 null values (1 Lat, 1 Long)

2. The data types were not an issue but there were a number of columns related to the location of the business

3. The full data set had ~ 150 k observations

4. Non-restaurants were included

5. A number of locations were outside the USA (e.g. state: ON, postal\_code: M4K 1N7 refers to Toronto)

Unique state values and the range of lat long values were further analyzed and revealed that a number of businesses were outside the USA.

### ### 4. Reformat column for date into data-time category

To clean the review data, very little clean up was necessary. The date was converted to datetime but the stars were kept as integers vs categories.

```

reviews\_df['date'] = pd.to\_datetime(reviews\_df['date'], format='%Y-%m-%d')

```

### ### 5. Extract businesses that are restaurants

The first clean up that was done was to remove businesses that were non-restaurants. This was done first as it was expected to yield the greatest reduction in observations.

```

restaurants\_df = businesses\_df[ businesses\_df['categories'].apply(lambda categories: any(pd.Series(categories).str.contains('Restaurants')) if len(categories)>0 else False)]

```

The new dataset of businesses were ~ 50k observations (almost 1/3 the original number of businesses).

Note: the impact on the number of reviews is less severe as we will observe later.

### ### 6. Extract restaurants that are in the USA using a bounding box

The next step of the clean up was to remove restaurants outside the USA using a bounding box of min-max latitude and longitudinal values. The bounding box for the US is (49.3457868 # north lat) (24.7433195 # south lat) (-124.7844079 # west long) (-66.9513812 # east long) .

This reduced the dataset to ~48k observations.

### ### 7. Drop business columns associated with business location

Next the columns associated with the business location and opening hours were dropped.

### ### 8. Merge the two dataframes

Restaurant business data frame was mereged with the review dataframe.

### ### 9. Explore merged dataframe

To understand the merged data set, 2 functions were used: `.shape`, `.info`.

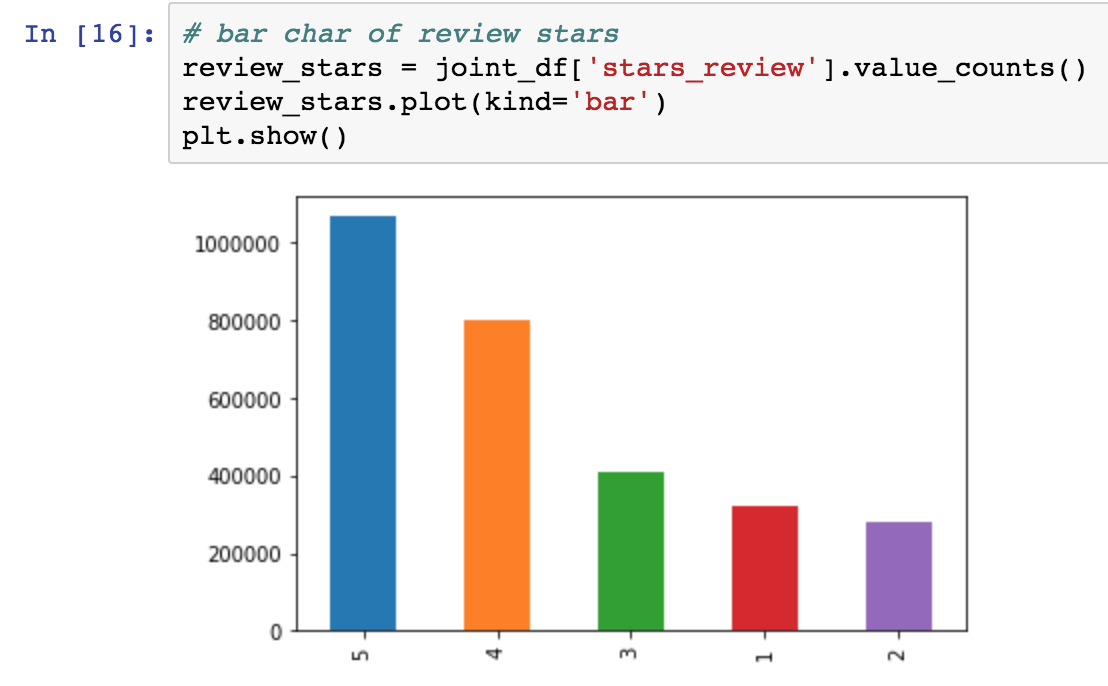
For the merged data, the following observations were made:

1. There were no null values

2. Inner join reduced the total number of observations (i.e. reviews) from ~4.7 million from all businesses to ~ 2.9 million from restaurants alone to the ~2.9 from restaurants in the USA bounding box.

By plotting a bar chart to show the distribution of star ratings from the reviews. It may be observed that there is a bias towards positive 5 star ratings.

No outliers are apparent at this stage.



### ### 10. Pre-processing the data

The pre-processing of the reviews `joint\_df['text']` involved 5 steps. These were defined in function named `pre\_process\_review`:

```

def pre\_process\_review( review\_text ):

# Remove punctuation and convert all text to lower case

nopunc = ''.join( [character for character in review\_text.lower() if character not in string.punctuation] )

# Remove stop words and lemmertize

non\_stop\_words = [lemmatizer.lemmatize(word) for word in nopunc.split() if not word in stopWords]

return non\_stop\_words

```

1. Removed punctuation

2. Convert characters to lower case

3. Removal of stop words

4. Lemmerization of words

5. Conversion for review text string to a list of words

Note that the additional step of expanding contractions was not used, but it would come as step 1 if it was needed.

These steps were then applied in to the `joint\_df['text']` column and stored into `joint\_df[‘processed\_review’]` column. Using:

```

joint\_df['processed\_review'] = joint\_df['text'].apply(lambda review\_text: pre\_process\_review( review\_text ))

```

For more information on pre-processing, please refer to references [2] [3] [4], which helped inform the steps selected above.

To make this function more efficient, I took advice from references [5] [6] and refactored the function into the following form:

```

def pre\_process\_review( review\_text ):

# Convert all text to lower case, tokenize into list of strings, remove punctuation and stop words, and lemmertize

return [lemmatizer.lemmatize(word) for word in wordpunct\_tokenize(review\_text.lower()) if not word in stop\_words\_punc]

```

Since the timing over 1 function proved too short and inconsistent, performance was measured using the cell that applied the function across all review text.

The original function took 1776 seconds, the modified function took 1389 seconds. Note for the original function, ` nopunc = ''.join( character for character in review\_text.lower() if character not in string.punctuation )` was used instead of `nopunc = ''.join( [character for character in review\_text.lower() if character not in string.punctuation] )`.

The processes was timed using:

```

import timeit

start\_time = timeit.default\_timer()

# CODE TO BE CHECKED

print(timeit.default\_timer() - start\_time)

```

### ### 11. Vectorise text

Once the review text has been preprocessed, it is ready to be converted into a vector. First, the vectorizer needs to be defined. In our case we have set the parameters as follows:

```

n\_features = 3000

vectorizer = CountVectorizer(analyzer = pre\_process\_review,

max\_features = n\_features,

max\_df=0.95,

min\_df=2).fit(joint\_df['text'])

```

max\_features: The total number of words used in the bag of words is restricted to 3000. This number is arbitrary large but can be tweaked in the future if necessary.

max\_df: This restriction is the upper threshold of the document frequency that can be expected. This has been arbitrarily set at 95% but can be lowered if commonly occurring words are not helpful.

min\_df: This is the lower threshold for the document frequency that must be met for the word to feature. If it is a float, the parameter represents a proportion of documents, if it is an integer, the parameter represents absolute counts.

For more information on the parameters of CountVectorizer, refer to reference [7] [8]

It may be observed that these parameters create a vectorizer with 3,000 features (the limit). The vectorized reviews create an array with the following properties:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Config 1 | Config 2 | Config 3 |
| n\_features | 3000 | 3000 |  |
| max\_df | 0.95 | 0.95 |  |
| min\_df | 2 | 2 |  |
| sparse matrix shape | 2876509, 3000 | 2876509, 3000 |  |
| number of non-zeros | 232154 | 123705583 |  |
| sparsity | 0.00% | 1.43% |  |

Additional combinations can be studied to find a more optimal configuration.

The vectorizer is then applied to the review text column and stored to a new numpy array:

```

# Vectorize text and convert the result to an Numpy array

vectorized\_text\_array = vectorizer.transform(joint\_df['text']).toarray()

```

The most common words do no indicate any abnormities or outliers.

### ### 12. Saving the output

The output of this analysis is the 2 sets of data saved:

1. `restaurant\_reviews.csv` - merged data of reviews and restaurants in the USA
2. `vectorized\_reviews.npy` - vectorized text array

These will be stored in new files in the folder: `02\_processed\_data`

Note that due to the restrictions of data use, these files will not be shared in a public repository.

### References:

[1] (http://en.wikipedia.org/wiki/Extreme\_points\_of\_the\_United\_States#Westernmost)

[2] (https://www.kaggle.com/c/word2vec-nlp-tutorial#part-1-for-beginners-bag-of-words)

[3] (https://radimrehurek.com/data\_science\_python/)

[4] (http://textminingonline.com/dive-into-nltk-part-iv-stemming-and-lemmatization)

[5] (https://stackoverflow.com/questions/46203023/how-can-i-make-my-python-nltk-pre-processing-code-more-efficient)

[6] (https://stackoverflow.com/questions/19130512/stopword-removal-with-nltk)

[7] (http://scikit-learn.org/stable/modules/generated/sklearn.feature\_extraction.text.CountVectorizer.html)

[8] (https://stackoverflow.com/questions/22920801/can-i-use-countvectorizer-in-scikit-learn-to-count-frequency-of-documents-that-w)