Section #: Text Mining Procedures

Start by saving your text files in a folder titled: “texts” This will be the “corpus” (body) of texts you are mining. Here, we can inspect contents of the corpus we just generated.

Once all corpus files prepared, text mining process can be initiated. We will present a text mining example along with explanations to the text mining procedures. The original text:

```
# Original Text
```

> *“What does your phrase "17 pieces of Tahitian South Sea pearl" mean? I am very interested in this necklace - is this your best price?”*

Step 1: Convert to lowercase

Since our goal when doing text mining project is to find what are the most frequently used terms, therefore, standardizing both lowercase and uppercase is a necessary procedure here.

```
# Convert all text to lowercase
myCorpus <- tm_map(myCorpus, content_transformer(tolower))
```

> *“what does your phrase "17 pieces of tahitian south sea pearl" mean? i am very interested in this necklace - is this your best price?”*

Step 2: Remove all numbers

Since numbers do not deliver too much information, and also difficult to generalize the meaning contained inside of numbers. At this moment, we decide to remove all numbers.

```
# Remove all numbers
myCorpus <- tm_map(myCorpus, content_transformer(removeNumbers))
```

> *what does your phrase” pieces of tahitian south sea pearl" mean? i am very interested in this necklace - is this your best price?*

Step 3: Delete all English stop-words

In English, there are a group of words are categorized as “stop-words” (common words) that usually have no analytic value. In every text, there are a lot of common, and uninteresting words (a, and, also, the, etc.). Such words are frequent by their nature, and will confound your analysis if they remain in the text. In the “tm” R package, here are the list of stop-words:

```
"i"         "me"        "my"        "myself"    "we"        "our"       "ours"      "ourselves" "you"       "your"
"yours"     "yourself"  "yourselves" "he"       "him"       "his"       "himself"   "she"       "her"       "hers"
"herself"   "it"        "its"       "itself"    "they"      "them"      "their"     "theirs"    "themselves" "what"
"which"     "who"       "whom"      "this"      "that"      "these"     "those"     "am"        "is"        "are"
"was"       "were"      "be"        "been"      "being"     "have"      "has"       "had"       "having"    "do"
"does"      "did"       "doing"     "would"     "should"    "could"     "ought"     "i'm"       "you're"    "he's"
"she's"     "it's"      "we're"     "they're"   "i've"      "you've"    "we've"     "they've"   "i'd"       "you'd"
"he'd"      "she'd"     "we'd"      "they'd"    "i'll"      "you'll"    "he'll"     "she'll"    "we'll"     "they'll"
"isn't"     "aren't"    "wasn't"    "weren't"   "hasn't"    "haven't"   "hadn't"    "doesn't"   "don't"     "didn't"
"won't"     "wouldn't"  "shan't"    "shouldn't" "can't"     "cannot"    "couldn't"  "mustn't"   "let's"     "that's"
"who's"     "what's"    "here's"    "there's"   "when's"    "where's"   "why's"     "how's"     "a"         "an"
"the"       "and"       "but"       "if"        "or"        "because"   "as"        "until"     "while"     "of"
"at"        "by"        "for"       "with"      "about"     "against"   "between"   "into"      "through"   "during"
"before"    "after"     "above"     "below"     "to"        "from"      "up"        "down"      "in"        "out"
"on"        "off"       "over"      "under"     "again"     "further"   "then"      "once"      "here"      "there"
"when"      "where"     "why"       "how"       "all"       "any"       "both"      "each"      "few"       "more"
"most"      "other"     "some"      "such"      "no"        "nor"       "not"       "only"      "own"       "same"
"so"        "than"      "too"       "very"
```

It is obvious that even though stop-words are necessary when people speaking, in a text summarizing procedure, they should be removed in order to keep the text cleaner and easier to be put in further analytics and prediction process.

```
# Delete all english stopwords. See list: stopwords("english")
myCorpus <- tm_map(myCorpus, removeWords, stopwords("english"))
```

> *phrase ” pieces tahitian south sea pearl" mean? interested necklace - best price?*

Step 4: Removing punctuation

Since computer cannot actually read meaning of punctuations. Punctuation and other special characters only look like more words to the computer and R. Use the following to methods to remove them from the text.

In some cases, if necessary, such as when working with emails, we can also remove special characters. This list has been customized to remove punctuation that you commonly find in emails. You can customize what is removed by changing them as you see fit, to meet your own unique needs.

```
# Remove all punctuation
myCorpus <- tm_map(myCorpus, content_transformer(removePunctuation))
```

> *phrase pieces tahitian south sea pearl mean interested necklace best price*

Step 5: Delete common word endings, like -s, -ed, -ing, ect.

In linguistic morphology and information retrieval, stemming is the process of reducing inflected (or sometimes derived) words to their word stem, which is a form to which affixes can be attached, base or root form – generally a written word form. Since the stem need not be identical to the morphological root of the word, and it is usually sufficient that related words map to the same stem, we want to stem out the shared parts between words with similar roots, stem or base in order to more accurately count the number of words appearing in the whole database. Another reason why we have been focusing on this part is to make sure information is not losing during text transformation process, which can misinterpret the original meaning of sentences to some extent.

A stemmer for English should be identify the string “cats” as based on the root “cat”, and “stems”, “stemmer”, “stemming”, “stemmed” as based on “stem”. A stemming algorithm reduces the words “loving”, “lover” and “loved” to the root word “love”. On the other hand, “argue”, “argued”, “argues”, “arguing”, and “argus” reduce to the stem “argu” however, “argument” and “arguments” reduce down to “argument” instead of “argu”. From this perspective, we can know that even though stemming algorithm can automatically stem words into more standardized forms, it may also mistakenly stem words into wrong stemmers, which is the biggest pain-point and limitation to this popular Natural Language Processing tool.

```
# Delete common word endings, like -s, -ed, -ing, etc.
myCorpus <- tm_map(myCorpus, stemDocument, language = "english")
```

> *phrase piece tahiti south sea pearl mean interest necklace best price*

Step 6: Reduce any whitespace

Since during the previous text-mining process, some whitespaces will be created and left in the current text. Thus, we run this part of code to remove those whitespace in case when R starting to tokenize sentences, it does not wrongly tokenize it due to continuously showing-up whitespaces.

```
# Reduce any whitespace (spaces, newlines, etc) to single spaces
myCorpus <- tm_map(myCorpus, content_transformer(stripWhitespace))
```

> *phrase piece tahiti south sea pearl mean interest necklace best price*

Now, we have our cleaned text body shown as above. In this example, we reduce 132 characters and 24 words down to 70 characters and 11 words, without losing important information hidden in this line of text. Based on this reduction, on averagely, we can therefore decrease 1/3 of processing time, which can be significantly important and helpful when facing large scale text message data.

However, when tokenizing sentences by single word, we may lose meaning of sentence. To avoid this problem, we also present N-gram tokenization process in our project, specifically both bi-gram and tri-gram. In the fields of computational linguistics and probability, an n-gram is a contiguous sequence of n items from a given sequence of text or speech. The items can be phonemes, letters, words or base pairs according to the application. The n-grams typically are collected from a corpus, which we have introduced in the previous text. When the items are words, n-grams may also be called singles. Again, using the same example,

> *phrase piece tahiti south sea pearl mean interest necklace best price*

For bigram, the sentence will be tokenized into:

> *"phrase piece", "piece tahiti", "tahiti south", "south sea", "sea pearl", "pearl mean", "mean interest", "interest necklace", "necklace best", "best price"*

For trigram, the sentence will be tokenized into:

> *"phrase piece tahiti", "piece tahiti south", "tahiti south sea", "south sea pearl", "sea pearl mean", "pearl mean interest", "mean interest necklace", "interest necklace best", "necklace best price"*

After tokenization with more grams, the meaning of sentence and terms can be more contained in the final result. The limitation of the technology side of the text mining is that information delivery rate will be decreased during transformation process. However, the greatest limitation on text mining is not only about technology, but also is related to copyright. There is a famous saying, "Fearful that their content might be freely redistributed, publishers tend to block programs that they find crawling the full text of articles, making no exceptions for users who have paid for access. They give permission only on a case-by-case basis to those who negotiate agreements on access and use." Thus, for the next step, we should consider improve our text-mining project from these two perspectives.