**Task I – Unstructured Data Analysis**

‘Unstructured Data English.xlsx’

1. Generate code for data cleansing in Python

#load module

import pandas as pd

import re

import nltk

import codecs

import string

from pattern.en import parsetree

from pattern.en import parse

from sklearn.feature\_extraction.text import CountVectorizer

#load data

data = pd.read\_excel('Unstructured Data English.xlsx', encoding='utf-8', index\_col=0)

data.head()

# check columns

data.columns

# drop irrelevant columns

data.drop(['Unnamed: 2','Unnamed: 3'],axis=1,inplace=True)

# rename column to simple meaningful.

data.rename(columns={'Katakana text Translated': 'text'}, inplace=True)

data.head(5)

#cleanse data

# below line contains multiple regular expression for removing complex form of emojis. And each expression is explained with comment right next to it.

emoticons\_str = r"""

(?:

[:=;] # Eyes

[oO\-]? # Nose (optional)

[D\)\]\(\]/\\OpP] # Mouth

)"""

# below line contains multiple regular expression for text cleansing. And each expression is explained with comment right next to it.

regex\_str = [

emoticons\_str,

r'<[^>]+>', # HTML tags

r'(?:@[\w\_]+)', # @-mentions

r"(?:\#+[\w\_]+[\w\'\_\-]\*[\w\_]+)", # hash-tags

r'http[s]?://(?:[a-z]|[0-9]|[$-\_@.&amp;+]|[!\*\(\),]|(?:%[0-9a-f][0-9a-f]))+', # URLs

r'(?:(?:\d+,?)+(?:\.?\d+)?)', # numbers

r"(?:[a-z][a-z'\-\_]+[a-z])", # words with - and '

r'(?:[\w\_]+)', # other words

r'(?:\S)' # anything else

]

# Sad Emoticons

emoticons\_sad = set([

':L', ':-/', '>:/', ':S', '>:[', ':@', ':-(', ':[', ':-||', '=L', ':<',

':-[', ':-<', '=\\', '=/', '>:(', ':(', '>.<', ":'-(", ":'(", ':\\', ':-c',

':c', ':{', '>:\\', ';('

])

#Happy Emoticons

emoticons\_happy = set([

':-)', ':)', ';)', ':o)', ':]', ':3', ':c)', ':>', '=]', '8)', '=)', ':}',

':^)', ':-D', ':D', '8-D', '8D', 'x-D', 'xD', 'X-D', 'XD', '=-D', '=D',

'=-3', '=3', ':-))', ":'-)", ":')", ':\*', ':^\*', '>:P', ':-P', ':P', 'X-P',

'x-p', 'xp', 'XP', ':-p', ':p', '=p', ':-b', ':b', '>:)', '>;)', '>:-)',

'<3'

])

#combine sad and happy emoticons

emoticons = emoticons\_happy.union(emoticons\_sad)

# below line is the compiled regular expression , which used multiple regular expressions with OR expression to do multiple filtered operation for extracting relevant words.

tokens\_re = re.compile(r'('+'|'.join(regex\_str)+')', re.VERBOSE | re.IGNORECASE)

# below line is the compiled regular expression , which used multiple regular expressions with OR expression to do multiple filtered operation for removing emojis.

emoticon\_re = re.compile(r'^'+emoticons\_str+'$', re.VERBOSE | re.IGNORECASE)

stop\_words = nltk.corpus.stopwords.words('english')

# below two lines will add irrelevant words that needs to be filtered out from key phrases.

stop\_words.extend(['rt','via',"isn't","ain't","don't","hasn't","haven't","I've","I'VE","i've",'mr','mrs','Mr','Mrs','MR','MRS'])

stop\_words.extend([u'i', u'me', u'my', u'myself', u'we', u'our', u'ours', u'ourselves', u'you', u'your', u'yours', u'yourself', u'yourselves', u'he', u'him', u'his', u'himself', u'she', u'her', u'hers', u'herself', u'it', u'its', u'itself', u'they', u'them', u'their', u'theirs', u'themselves', u'what', u'which', u'who', u'whom', u'this', u'that', u'these', u'those', u'am', u'is', u'are', u'was', u'were', u'be', u'been', u'being', u'have', u'has', u'had', u'having', u'do', u'does', u'did', u'doing', u'a', u'an', u'the', u'and', u'but', u'if', u'or', u'because', u'as', u'until', u'while', u'of', u'at', u'by', u'for', u'with', u'about', u'against', u'between', u'into', u'through', u'during', u'before', u'after', u'above', u'below', u'to', u'from', u'up', u'down', u'in', u'out', u'on', u'off', u'over', u'under', u'again', u'further', u'then', u'once', u'here', u'there', u'when', u'where', u'why', u'how', u'all', u'any', u'both', u'each', u'few', u'more', u'most', u'other', u'some', u'such', u'no', u'nor', u'not', u'only', u'own', u'same', u'so', u'than', u'too', u'very', u's', u't', u'can', u'will', u'just', u'don', u'should', u'now'])

stop\_words = set(stop\_words)

# below is the initialization of wordnet lemmatizer for doing lemmatization. Basically a rule based approach was used to bring root form of transformed words.

wn = nltk.WordNetLemmatizer()

def get\_phrases(sentence):

phrase\_dict = {}

# this line will gives us dependency tree relations.

parse\_tree = parsetree(sentence, relations=True, lemmata=True)

# in this line we’ll iterate to extract chunk whose type is phrase. And based on that will retrieve phrases values.

for sent in parse\_tree:

for chunk in sent.chunks:

phrase\_key = chunk.type

phrase\_val = ' '.join([w.string for w in chunk.words])

if not phrase\_dict.get(phrase\_key, []):

phrase\_dict[phrase\_key] = []

if len(phrase\_val.split())>1:

phrase\_dict[phrase\_key].append(phrase\_val)

return phrase\_dict

def tokenize(s):

return tokens\_re.findall(s)

def preprocess(s, lowercase=True):

# substitute hyper link with empty string

s = re.sub(r'http\S+', '', s)

# substitute html tabs with empty string

s = re.sub(r'<.\*?>', '', s)

#remove numbers from tweet

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

s = re.sub(r'(?:(?:\d+,?)+(?:\.?\d+)?)','', s)

#remove &amp; from tweet

s = s.replace('&amp;','&')

s = s.replace('’',"'")

s = s.replace('-'," ")

phrases = []

phrases = get\_phrases(s).get('NP',[])

if lowercase:

phrases = [word\_p.lower() for word\_p in phrases]

tokens = tokenize(s)

if lowercase:

tokens = [token if emoticon\_re.search(token) else token.lower() for token in tokens]

#check tokens against stop words , emoticons and punctuations

tokens = [word.replace("'s",'') for word in tokens if word not in stop\_words and word not in emoticons and word not in string.punctuation]

#remove work with one or two chracter length.

tokens = [word for word in tokens if len(word)>2]

tokens = [wn.lemmatize(word) for word in tokens]

return (phrases, tokens)

tweet = "RT @marcobonzanini: just &amp; <abcd> an example! :D http://example.com #NLP 123 12/45/2020 987654321 SD09875 sam's"

print(preprocess(tweet))

# ['RT', '@marcobonzanini', ':', 'just', 'an', 'example', '!', ':D', 'http://example.com', '#NLP']

# add two new column next to text field. These two column will contain extracted phrases and tokens.

data[['phrases', 'cleansed\_tokens']] = data.apply(lambda row: pd.Series(preprocess(row['text'])), axis=1)

# verify sample processed data

for rec in data[:15].iterrows():

print(rec[1]['text']," --- ",rec[1]['cleansed\_tokens']," --- ",rec[1]['phrases'])

print()

from collections import Counter

terms\_all = [word for wordlist in data['cleansed\_tokens'] for word in wordlist]

count\_all = Counter()

# Update the counter

count\_all.update(terms\_all)

# Print the first 5 most frequent words

print(count\_all.most\_common(250))

# result

[('kraft', 5023), ('cheese', 1374), ('goodell', 906), ('velveeta', 802), ('robert', 744), ('video', 739), ('#cookingupgood', 645), ('#recipes', 633), ('new', 522), ('need', 468), ('patriot', 434), ('nfl', 426), ('watch', 417), ('like', 403), ('idea', 394), ('delicious', 391), ('roger', 389), ('visit', 387), ('page', 387), ('year', 386), ('mac', 359), ('food', 349), ('owner', 349), ('dinner', 321), ('apology', 320), ('pocket', 303), ('walmart', 290), ('yummy', 255), ('#sweepstakes', 253), ('bob', 250), ('deflategate', 239), ('make', 222), ('integrity', 192), ('time', 190), ('get', 176), ('relationship', 169), ('say', 166), ('investigation', 166), ('speech', 159), ('bill', 156), ('marketing', 148), ('want', 146), ('data', 145), ('#nfl', 139), ('england', 136), ('said', 135), ('harnessed', 135), ('transform', 135), ('game', 130), ('know', 126), ('center', 123), ('stand', 123), ('front', 123), ('think', 122), ('one', 121), ('news', 121), ('big', 120), ('unfairly', 120), ('attacked', 120), ('#superbowlxlix', 118)

terms\_all\_phrase = [word for wordlist in data['phrases'] for word in wordlist]

count\_all = Counter()

# Update the counter

count\_all.update(terms\_all\_phrase)

# Print the first 5 most frequent words --> meaningful phrases

print(count\_all.most\_common(25))

# Result

[('robert kraft', 433), ('the video', 383), ('visit the kraft', 381), ('cookingupgood page', 381), ('delicious ideas', 381), ('kraft foods', 279), ('yummy new year', 249), ('roger goodell', 212), ('kraft dinner', 125), ('bob kraft', 109), ('my pocket velveeta', 109), ('my pockets velveeta', 106), ('an apology', 94), ('nfl spotlight', 88), ('the new england patriots', 85), ('million investment', 78), ('big time', 75), ('kraft cheese', 73), ('kraft apology', 72), ('patriots owner kraft', 70), ('deflategate investigation kraft relationship nfl integrity boston herald', 62), ('caleb kraft prints modular thumbstick extentions', 59), ('my job', 57), ('the nfl', 56), ('velveeta cheese', 52)]

# Count hashtags only

terms\_hash = [term for term in terms\_all if term.startswith('#')]

# Count usertags only

terms\_usertag = [term for term in terms\_all if term.startswith('@')]

# Count terms only (no hashtags, no mentions)

terms\_only = [term for term in terms\_all if not term.startswith(('#', '@'))]

count\_all = Counter()

# Update the counter

count\_all.update(terms\_hash)

# Print the first 5 most frequent words --> primary theme tags

print(count\_all.most\_common(25))

# Result

[('#cookingupgood', 645), ('#recipes', 633), ('#sweepstakes', 253), ('#nfl', 139), ('#superbowlxlix', 118), ('#kraft', 115), ('#robertkraft', 114), ('#rtmusic', 107), ('#patriots', 98), ('#deflategate', 70), ('#superbowl', 43), ('#gameday', 38), ('#gossip', 38), ('#news', 28), ('#goodell', 19), ('#coupon', 18), ('#cheese', 17), ('#rogergoodell', 16), ('#ebay', 16), ('#askroger', 15), ('#velveeta', 13), ('#sports', 12), ('#forsale', 12), ('#coupons', 11), ('#immersionjournalism', 11)]

count\_all = Counter()

# Update the counter

count\_all.update(terms\_only)

# Print the first 5 most frequent words

print(count\_all.most\_common(25))

# Result

[('kraft', 5023), ('cheese', 1374), ('goodell', 906), ('velveeta', 802), ('robert', 744), ('video', 739), ('new', 522), ('need', 468), ('patriot', 434), ('nfl', 426), ('watch', 417), ('like', 403), ('idea', 394), ('delicious', 391), ('roger', 389), ('visit', 387), ('page', 387), ('year', 386), ('mac', 359), ('food', 349), ('owner', 349), ('dinner', 321), ('apology', 320), ('pocket', 303), ('walmart', 290)]

# Document-Term Matrix using CountVectorizer from Sklearn.

countVectorizer = CountVectorizer()

countVector = countVectorizer.fit\_transform([' '.join(text) for text in data['cleansed\_tokens']])

print('{} Number of tweets has {} words'.format(countVector.shape[0], countVector.shape[1]))

#print(countVectorizer.get\_feature\_names())

count\_vect\_df = pd.DataFrame(countVector.toarray(), columns=countVectorizer.get\_feature\_names())

count\_vect\_df.head()

# Term-Document Matrix

term\_doc\_df = count\_vect\_df.transpose()

term\_doc\_df.head()

term\_doc\_df.index.name='Token'

# Dump as excel

term\_doc\_df[:50].to\_excel("tfidf-output.xlsx")

import openpyxl

from openpyxl.styles import Font

#wb = openpyxl.Workbook()

wb = openpyxl.load\_workbook("tfidf-output.xlsx")

sheet = wb.active

sheet.insert\_rows(0,1)

# merge cell.

sheet.merge\_cells('B1:Z1')

sheet.cell(row = 1, column = 2).value = 'S.No.(Unique Identifier of a comment.)'

# set the font style to bold

sheet.cell(row = 1, column = 2).font = Font(size = 24, bold = True)

wb.save('tfidf-output-merge.xlsx')

**Summary of Exploratory Data Analysis**[**¶**](http://localhost:8888/notebooks/data_structure/case%20study%20L%26T/case%20study/Deloite%20Unstructured%20English%20Tweet%20EDA.ipynb#Summary-of-Exploratory-Data-Analysis)

- tweets are quite unclean. some taken care here. still more required to be handled such as spell checker, more accurate phrase extracter using rule based parser or using NER..etc.

- Most of the abbreviations can either be transformed or dropped. Needs to be taken care after Subject knowledge.

- used lemmatizer, in this dataset.

- due to lack of memory(3GB RAM) in my system, i'm dumping few samples of term-frequency records to excel. user can use same code to dump all records with having good memory system. or we can do other ways like using DASK, or writing file batch by batch.

- have mentioned comments with each code. i'm sure those are understandable. so not writing much here.

- with that 4 questions regarding to this task have been answered through code, comments, and results.