```
# coding=utf-8
# Will do also 2019
# Needs to have MAUDE_data.csv to be updated for the daVinci => Use
 DownloadMAUDE
# Will go through Event Description and Narrative of all the reports
# Extracts different information related to the malfunctions and their
 impact, such as:
# Troubleshooting, System reset, Conversion, Rescheduling, Time information

import bs4
import cookielib
import csv
import nltk
import os
import re
import string
import sys
import time
import urllib
import urllib2
import xlrd
import xlwt
from nltk.corpus import stopwords
from nltk.probability import ConditionalFreqDist
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.tag import pos_tag
from nltk.tokenize import word_tokenize

from negex import *

rfile = open(r'README.txt')
irules = sortRules(rfile.readlines())
count1 = 0
count2 = 0
fallen_cnt1 = 0
fallen_cnt2 = 0
broken_cnt = 0
missed_injuries = 0  # Event
 Type里没有记录为injury但通过分析Event确发现有injury的记录数量

# daVinci = 0 means Cardiac, daVinci = 1 means daVinci records
daVinci = 1
# Years
start_year = 2000  # 2015 #2000
current_year = 2019  # 2019 # 2014
end_year = 2019  # 2019 # 2008 # 2013
# sourcedir = './ExtractedFiles/PreFiles/'
save_dir = './ExtractedFiles/' + str(end_year) + '/'
tmpdir = './tmp/' + str(end_year) + '/'
Pre_CSV_In = save_dir + 'Pre_daVinci_Impacts_2013.csv'  #
 './Pre_daVinci_Impacts_2018.csv'
CSV_In = save_dir + 'daVinci_MAUDE_Data_2019.csv'  #
 './daVinci_MAUDE_Data_2018.csv'
```

```
CSV_Out = save_dir +
 'daVinci_MDR_Malfunction_Impacts_{}_PLOS_One.csv'.format(
    end_year)  # './daVinci_MDR_Malfunction_Impacts_2018_PLOS_One.csv'
Excel_Out = save_dir +
 'daVinci_MAUDE_Classified_{0}_{1}.xls'.format(start_year,
                                                                  end_yea
                                                        r)  #
                                                        '
                                                        ./daVinci_MAUDE_Classifie
                                                        d_2000_2019.xls'
print(os.getcwd())

f0 = open(CSV_In, 'rb')
csv_rd1 = csv.reader(f0, dialect='excel', delimiter=',')

pre_f0 = open(Pre_CSV_In, 'rb')
pre_csv_rd1 = csv.reader(pre_f0, dialect='excel', delimiter=',')

f0_1 = open(save_dir + 'Injuries_Malfunctions.csv', 'rb')  #
 伤害的原因：可通过分析Event_type字段得到此文件
# D = Death
# IN = Injury
# IL = Injury
# IJ = Injury
# M = Malfunction
# O = Other
# * = No answer provided
csv_rd2 = csv.reader(f0_1, dialect='excel', delimiter=',')
f0_2 = open(save_dir + 'Surgery_Class_Dictionary.csv', 'rb')  # 手术类型
surgery_dict = csv.reader(f0_2, dialect='excel', delimiter=',')
f0_3 = open(save_dir + 'MDR_Key_Surgery_Class_Dictionary.csv', 'rb')  #
 加入了MDR_KEY的手术类型
surgery_dict2 = csv.reader(f0_3, dialect='excel', delimiter=',')

# CSV output file
f1 = open(CSV_Out, 'wb')
csv_wr = csv.writer(f1, dialect='excel', delimiter=',')
f2 = open(tmpdir + 'Da_Vinci_MDR_Malfunction_TempResults2.csv', 'wb')
csv_wr2 = csv.writer(f2, dialect='excel', delimiter=',')
f3 = open(tmpdir + 'Da_Vinci_MDR_Malfunction_TempResults3.csv', 'wb')
csv_wr3 = csv.writer(f3, dialect='excel', delimiter=',')
f4 = open(save_dir + 'Difference.csv', 'wb')
csv_wr4 = csv.writer(f4, dialect='excel', delimiter=',')
f5 = open(save_dir + 'Negations.txt', 'wb')
f6 = open(save_dir + 'System_Errors.txt', 'wb')  # 故障为system
 error的记录里的Event描述
f7 = open(save_dir + 'Possible_Error_Codes.txt', 'wb')  # 各种错误类型出现的频率
f8 = open(save_dir + 'TestErrorCodes.txt', 'wb')  # 出现system
 error的记录的MDR_KEY
f9 = open(save_dir + 'TestInstruments.txt', 'wb')  # 仪器broken的类别

# Excel output file titles
```

```
column_titles = ['MDR_Key', 'Narrative', 'Event', 'Patient Impact',
 'Outcome', 'Injuries',
                 'Manufacturer', 'Brand_Name', 'Generic_Name',
                  'Product_Code',
                 'Manufacture_Year', 'Event_Year', 'Corrected_Event_Year',
                  'Report_Year', 'Time_to_Event',
                 'Time_to_Report',
                 'Surgery_Type', 'Surgery_Class', 'New Converted', 'New
                  Rescheduled',
                 'System Reset', 'Fallen', 'System Error',
                 'Moved', 'Arced', 'Broken', 'Tip Cover', 'Vision',
                  'Power', 'Other']

csv_wr.writerow(column_titles)

# 保存临时文件
csv_wr2.writerow(['MDR_Key', 'Narrative', 'Event', 'Patient Impact',
 'Temp'])
csv_wr3.writerow(['MDR_Key', 'Narrative', 'Event', 'Patient Impact',
 'Temp'])

workbook = xlwt.Workbook("iso-8859-2")
worksheet = workbook.add_sheet('sheet1')
column = 0
for c in column_titles:
    worksheet.write(0, column, c)
    column = column + 1

# Dictionaries, 并不懂这些字典是如何得到的
Typoed_MDRs = ['349101', '626290', '1609392', '2113223', '1227009',
 '1063554', '1063553', '2466842', '699291',
                '3530681', '3360375', '3215594', '3021230', '2861342',
                 '2402556', '2343127', '2209897',
                '1832491', '1707868', '1707868', '1695569', '1671251',
                 '2468265', '2468249', '2426441', '2940423',
                '2823390', '3306069', '2942665', '2673688', '1595504',
                 '919538', '919534', '1804859', '930610',
                '930611', '869814', '886699', '2260987', '1707866']
Other_Missed = ['2504745', '1735914', '933211', '870229', '2120175',
 '1466302', '823634', '704959', '690620', '660137',
                 '310796']

Injury_Keywords = ['injury', 'injuries', 'injured', 'torn', 'tear to',
 'tear on', 'burn', 'puncture', 'perforation',
                   'laceration', 'lacerated', 'damage to', 'organ damage',
                    'damaged', 'avulsion', 'rupture', 'repture',
                   ' necrosis', 'hole in', 'mark']  # , 'bleed']
Body_Keywords = ['patient', ' pt', 'artery', 'tissue', ' vein', ' skin',
 'vessel', 'cavity',
                 ' uterus', ' liver', 'kidney', 'tounge', ' bowel',
                  'abdomen', 'abdomin', ' colon', 'ovarian tube',
                 'bladder', ' pelvis', 'intestines', ' anatomy', 'thymus
                  gland', 'rectum', 'colon', 'nurve']
```

```
surgery_stop_adjs = ['surgical', 'surdical', 'surgery', 'operative', 's',
 'si', 'da', 'vinci', 'davinci', 'robotic',
                     'during', 'the', 'of', 'on', 'for', 'and', 'demo',
                      'a', 'an', 'intended', 'training',
                     'open', 'clinical', 'post', 'before', 'after', 'pre',
                      'post', 'previous', 'endoscopic',
                     'complete', 'laparoscopic', 'laparoscopy', 'original',
                      '2000', '1', 'hour', 'assisted']

convert_keywords = [' convert ', ' converted ', ' converting ',
                    'laparoscopically without the robot',
                    'traditional open techniques', 'traditional open
                     surgical techniques']
reschedule_keywords = [' reschedule ', ' rescheduled ', ' abort ', '
 aborted ']

Mistake_Keywords = ['accidentally', 'inadvertently', 'indidentally', 'by
 mistake', 'involuntarily',
                    'unintentionally', 'mistakenly', 'involuntarily',
                     'unwittingly']
error_keywords = ['system error', 'error code', 'error message', 'critical
 error', 'intermittent error',
                  'system fault', 'fault code', 'fault message', 'critical
                   fault', 'intermittent fault',
                  'recoverable fault', 'non-recoverable fault',
                   'nonrecoverable fault', 'unrecoverable fault',
                  'recoverable error', 'non-recoverable error',
                   'nonrecoverable error', 'unrecoverable error',
                  'recoverable safety state', 'non-recoverable safety
                   state', 'safety state',
                  '"instrument not recognized" error message', 'fault error
                   mode', 'fault alarm'

                                                 'fault appeared', 'fault
                                                 displayed',
                  'error appeared', 'error displayed']
error_keywords2 = ['error code', 'error message', 'system_fault', 'fault
 code', 'fault message',
                    'confirmed the system error', '"instrument not
                     recognized" error message',
                    'system error experienced', 'system errors experienced',
                    'system fault experienced', 'fault mode',
                    'recoverable fault', 'non-recoverable fault',
                     'nonrecoverable fault', 'unrecoverable fault',
                    'recoverable error', 'non-recoverable error',
                     'nonrecoverable error', 'unrecoverable error',
                    'recoverable safety state', 'non-recoverable safety
                     state', 'safety state',
                    'fault appeared', 'fault displayed', 'error appeared',
                     'error displayed']

vision_keywords = ['no vision', 'no video', 'no image', 'low light', '"dim"
 light', 'lack of light', 'little light',
                    'vision problem', 'vision issue', 'double vision',
                     'video issue', 'video problem', 'illuminator',
```

```
                  'vision system error', 'vision loss', 'loss of vision',
                   'loss of video', 'video loss', 'image loss',
                  'lost vision', 'lost the vision', 'lost video', 'lost
                   the video', 'lost image', 'lost the image'

                                                'lost 3D',
                  'loss of 3D', 'loss of image', 'light source was very
                   dim', 'light source was dim',
                  'vision was lost', 'video was lost', 'image was lost',
                   'vision lost', 'video lost', 'image lost',
                  'blurry', 'foggy', 'fogging', 'tint', 'right eye', 'left
                   eye', 'red eye', 'black and white balance',
                  'image was not aligned', 'black image', 'foggy image',
                   'red image', 'left eyes', 'right eyes',
                  'image', 'flicker', 'pink', '3d camera', '3d image', '3d
                   vision',
                  'unable to calibrate the camera', 'unable to calibrate
                   the endoscope',
                  'issue with the camera calibration', 'camera failed',
                   'camera error',
                  'lens was missing', 'video signal', 'video process
                   assembly',
                  'touchscreen', 'touch screen', 'assistant monitor',
                   'vision side cart shutdown',
                  'unable to focus', 'focus controller', 'not able to
                   focus',
                  'difficulty focusing', 'difficulties focusing',
                   'focusing issue',
                  'could not focus', 'could not be focused',
                  'focus was not working', 'lost focus control',
                  'focus controller not operating', 'focus controller not
                   working',
                  'problems with the focus controller', 'issue with the
                   focus controller']

fallen_keywords1 = ['fell inside', 'fell into', 'falling into', 'falling
 inside', 'fallen into', 'fallen inside',
                    'fall on the pt', 'fall on the patient', 'fall on the
                     user',
                    'fall on a pt', 'fall on a patient', 'fall on a user',
                    'into the pt', 'into the patient', 'into a pt', 'into a
                     patient',
                    "in pt's abdomen", 'fell in the pt', 'fell in the
                     patient', 'found in the pt',
                    'found in the patient',
                    'inside the patient', 'inside the pt', 'inside a pt',
                     'inside a patient',
                    'inside of the patient', 'inside of the pt', 'inside of
                     a pt', 'inside of a patient']
fallen_keywords2 = {('fall', 'V'), ('fallen', 'V'), ('fell', 'V'),
 ('falling', 'V'),
                    ('broke', 'V'), ('broken', 'V'), ('breaking', 'V'),
                    ('dropped', 'V'), ('popped', 'V'), ('smoking', 'V'),
                     ('snapped', 'V'),
```

```
                    ('in', 'IN'), ('into', 'IN'), ('inside', 'IN'), ('on',
                     'IN'), ('off', 'IN'),
                    ('pt', 'NN'), ('patient', 'NN'), ('user', 'NN'),
                     ('abdomen', 'NN')}
fallen_keywords3 = {'retrieved', 'removed from the'}
peice_keywords = {'piece', 'fragment', 'shredding', 'object', 'broken',
 'component', 'tip'}
# To exclude from what is observed in the patient when looking for fallen
 cases
burn_keywords = {'spark', 'sparking', 'burn', 'burning', 'arcing', 'arc',
 'blood', 'injury', 'hole', 'char'}

# fallen_keywords2 =
 {('fall','NN'),('fallen','VBN'),('fell','VBD'),('falling','VBG'),
 ('found','VBD'), ('lost',
#
 'VBD'),('smoking','VBG'),('broke','VBP'),('broken','VBN'),
 ('snapped','VBD'),('came','VBD'), ('in','IN'),('into',
# 'IN'),('inside','IN'),('on','IN'),('off','IN'),
 ('pt','NN'),('patient','NN'),('user','NN'),('abdomen',
# 'NN')} convert_keywords = ['convert to','converted to', 'converting to',
 'convert from', 'converted from',
# 'converting from', 'convert the procedure', 'convert procedure', 'convert
 the case', 'convert case', 'convert the
# planned procedure', 'convert the planned surgical procedure',
 'laparoscopically without the robot', 'traditional
# open techniques', 'traditional open surgical techniques']

# reschedule_keywords =['reschedule','rescheduled','was aborted', 'abort
 the procedure','abort procedure',
# 'aborted the procedure', 'aborted procedure', 'abort the planned
 procedure','abort planned procedure', 'aborted the
# planned procedure', 'aborted planned procedure', 'to abort the planned
 surgical procedure']

moving_keywords = {('not', 'NG'), ("didn't", 'NG'), ('stopped', 'NG'),
                   ('move', 'V'), ('moved', 'V'), ('moving', 'V'),
                    ('functioning', 'V'), ('go off', 'V'),
                   ('open', 'V'), ('rotate', 'V'), ('close', 'V'), ('work',
                    'V'), ('grasp', 'V'), ('stop', 'V'),
                   ('respond', 'V'),
                   ('opening', 'V'), ('rotating', 'V'), ('closing', 'V'),
                    ('working', 'V'), ('grasping', 'V'),
                   ('stopping', 'V'),
                   ('by', 'IN'), ('wrong', 'IN'), ('no', 'IN'), ('without',
                    'IN'),
                   ('non-intuitive', 'IN'), ('unintended', 'IN'),
                    ('sudden', 'IN'), ('unexpected', 'IN'),
                   ('uncontrolled', 'IN'), ('erratic', 'IN'),
                   ('themseleves', 'NN'), ('itself', 'NN'), ('command',
                    'NN'), ('direction', 'NN'),
                   ('manner', 'NN'), ('prompting', 'NN'), ('movement',
                    'NN'), ('motion', 'NN')}
```

```
arcing_keywords = [' arc ', ' arcing', ' arced', ' spark', 'sparking',
 'sparked' ' sparks',
                   ' arching', ' arch ', ' arched', 'electrical discharge']
broken_keywords = ['break ', 'broke', ' damage']
Instruments = [' piece', ' fragment', ' cable', ' wire', ' tip', 'tip
 cover', 'tip cover accessory', 'needle',
               'pin', ' clamp ', ' insulation', 'forcep', 'forceps',
                'cautery', 'screw', 'scissor',
               'distal', ' blade ', ' sleeve ', 'curved shears', 'cannula',
                'grasper', 'spatula', 'grip',
               ' distal ', 'trocar', 'electrocautery', 'tube', 'endoscope',
                'camera', 'battery',
               'insturment', 'accessory', 'instrument', 'component',
                'device']
Cardiac_Instruments = ['staple', ' valve ', ' clip', 'endoscope', ' lead ',
 'clamp', 'annuloplasty',
                       'electrosurgical', 'catheter']

reset_keywords = ['system reset', 'restart', 'reboot', 'site reset', 'reset
 the',
                  'power off', 'powered off', 'powered down', 'power down',
                   'shut down', 'turn off']
troubleshoot_keywords = ['troubleshoot', 'troubleshooting']
anesthesia_keywords = ['anesthesia', 'under anesthesia']

tip_cover_keywords = ['hole', 'burn', ' split', ' tear', 'crack', 'fail']

time_keywords = ['hour', 'hours', 'hr', 'hrs', 'minutes', 'mins']
many_keywords = ['few', 'several', 'within', ]
negation_keywords = ['no', 'not', 'nothing', 'without', 'none', "don't",
 "doesn't"]
temporal_keywords = ['during', 'before', 'after', 'into', 'prior', 'for',
 'past', 'when', 'by']

Digit_Table = {'one': 1, 'two': 2, 'three': 3, 'four': 4, 'five': 5,
               'six': 6, 'seven': 7, 'eight': 8, 'nine': 9, 'ten': 10,
               'eleven': 11, 'twelve': 12, 'thirteen': 13,
               'fourteen': 14, 'fifteen': 15, 'sixteen': 16,
               'seventeen': 17, 'eighteen': 18, 'nineteen': 19,
               'twenty': 20, 'thirty': 30, 'forty': 40, 'fifty': 50,
               'sixty': 60, 'seventy': 70, 'eighty': 80, 'ninety': 90,
               'an': 1, 'half': 0.5, '1/2': 0.5, '~1': 1, '~30': 30}


# Works as good as NegEx, it only doesn't consider these cases:
# "..., but" ".., however" "cannot be certain if ..." "cannot be determined
 if..."
# Handles the cases with "Nothing" that NegEx cannot handle.
def negation_check(s, keyword):
    loc = s.find(keyword)
    if s[0:loc - 1].find('and') > 0:
        sub_s = s[s[0:loc - 1].find('and') + 3:loc + 5]  # (and) ... s
    elif s[0:loc - 1].find('however') > 0:
        sub_s = s[s[0:loc - 1].find('however'):loc + 5]  # however ... s
    elif s[0:loc - 1].find(';') > 0:
```

```
            sub_s = s[s[0:loc - 1].find(';') + 1:loc + 5]  # (;) ... s
        else:
            sub_s = s[0:loc + 5]
        # Split the sentence to words
        regex = re.compile('[%s]' %
         re.escape('!"#$%&\'()*+,./:;<=>?@[\\]^_`{|}~'))
        Cs = regex.sub(' ', sub_s).lower()
        Ws = Cs.split(' ')
        for w in Ws:
            for n in negation_keywords:
                if (w == n):  # found negation words
                    # print Ws;
                    f5.write(sub_s + '\n')
                    return True
        return False


# Make the tables of injury_malfunctions hash
IN_MALFUNC = {'MDR_Key': 'Malfuncion_Class'}
for row in csv_rd2:
    IN_MALFUNC[row[0]] = row[1]  # 部分MDR_Key的机器人故障类型

# Make a hash for the previous classified data
Pre_Data = {'MDR_Key': []}
for row in pre_csv_rd1:
    Pre_Data[row[0]] = [row[20], row[21], row[22], row[23], row[24],
     row[25], row[26], row[27]]
    # 包含的字段:
    # New Rescheduled   System Reset    Fallen  System Error    Moved
    # Arced Broken  Tip Cover
    # 故障的类型

# Make the tables of surgery classes
Surgery_Dictionary = []
for row in surgery_dict:
    Surgery_Dictionary.append([row[0], row[1].strip()])  # 手术方式、手术类别

Surgery_Class_Hash = {'MDR_Key': ['Surgery Class', 'Surgery_Type']}
for row in surgery_dict2:
    Surgery_Class_Hash[row[0]] = [row[2], row[1]]

# Initialize
Error_Code_List = {'Error Code': (0, 'Description')}
all_data = []
all_inst_postfix = []
all_broken = []
chk_sum = 1
legal_count = 0
prior_count = 0
# Get the total number of records to read
NRows = len(list(csv_rd1))  # 总的记录
RStep = NRows / 10
# Go to the begining
f0.seek(0)
```

```
title = csv_rd1.next()
# Iterate over all MDR keys
for row in csv_rd1:
    if (daVinci == 0):
        MDR_Key = row[1]
        Surgery_Keyword = row[2]
        Event = row[3]
        Narrative = row[4]
        Impact = row[5]
        Outcome = row[6]
        Manufacture_Year = row[7]
        Event_Year = row[8]
        Report_Year = row[9]
        Time_to_Event = row[10]
        Time_to_Report = row[11]
        Manufacturer = row[12]
        Brand_Name = row[13]
        Generic_Name = row[14]
        Product_Code = row[15]
    else:
        MDR_Key = row[1]
        Event = row[2]
        Narrative = row[3]
        Impact = row[4]
        Outcome = row[5]
        Manufacture_Year = row[6]
        Event_Year = row[7]
        Report_Year = row[10]
        Time_to_Event = row[11]
        Time_to_Report = row[12]
        Manufacturer = row[13]
        Brand_Name = row[14]
        Generic_Name = row[15]
        Product_Code = row[16]

    # Correct missing Event Years based on Report Year
    if (Event_Year == 'N/A') or (Event_Year == '') or (Event_Year == ' '):
        Corrected_Event_Year = Report_Year
    else:
        Corrected_Event_Year = Event_Year

    # Clean up the Narrative and Event text (Remove punctuations)
    regex = re.compile('[%s]' %
     re.escape('!"#$%&\'()*+,./:;<=>?@[\\]^_`{|}~'))
    CNarrative = ' '.join(regex.sub(' ', Narrative).lower().split())
    CEvent = ' '.join(regex.sub(' ', Event).lower().split())
    # Split the Narrative and Event text into words
    WNarrative = CNarrative.split(' ')
    WEvent = CEvent.split(' ')  # 各个word
    # Split the Narrative and Event text into sentences
    SNarrative = Narrative.lower().split('.')
    SEvent = Event.lower().split('.')

    # Default Value for Other
```

```
Other = 'N/A'

#### Brand Name Categories for Cardiac Procedures
Device_Category = 'Other'
if (daVinci == 0):
    for k in Cardiac_Instruments:
        if Generic_Name.lower().find(k) > -1:
            Device_Category = k
            print 'Category =' + k
            break

#### Class and type of surgery
#### It is important to do this sentence by sentence, to make sure the
 first keyword is captured
Surgery_Type = ''
Surgery_Class = 'N/A'
Cardiac_Type = 'N/A'
for s in SEvent:  # iterate each sentence ( for one record )
    for entry in Surgery_Dictionary:
        ST = entry[0].lower()  # sugery type
        SC = entry[1].strip()  # sugery class
        if (s.find(ST) > -1):
            # Surgery_Type = Surgery_Type + ','+ST # find some surgery
             type, then combine it, break
            Surgery_Type = Surgery_Type + ST  # ljn changed
            Surgery_Class = SC
            break
    if (Surgery_Class != 'N/A'):
        break
if (Surgery_Class == 'N/A'):  #
 没有从Event描述中找到surgery_dict里的surgery type，那么就从Narrative描述里寻找
    for entry in Surgery_Dictionary:
        ST = entry[0].lower()
        SC = entry[1].strip()
        if (Narrative.lower().find(ST) > -1):
            Surgery_Type = Surgery_Type + ST
            Surgery_Class = SC

if (Surgery_Class == 'N/A'):  # Narrative里也没有找到，那就从已知surgery
 type和surgery class的dict里看有没有对应的MDR_kEY
    # Check previously had some records classified but I am missing
     them in this version?
    # 当Event和Narrative里都没有surgery
     type说明，数量有限的Surgery_Class_Hash dict也不一定能找到
    # This is a problem
    if (Surgery_Class_Hash.has_key(MDR_Key) and
            Surgery_Class_Hash[MDR_Key][0] != 'N/A' and
             Surgery_Class_Hash[MDR_Key][1] != 'tors'):
        print MDR_Key + ': ' + Surgery_Class_Hash[MDR_Key][0] + ' - ' +
         Surgery_Class_Hash[MDR_Key][1]
    # Check for other types of surgery that we didn't consider in our
     dictionary
    if WEvent.count('procedure') > 0:  # 有procedure
     单词的出现，从Event里提取surgery type
```

```
        procedure_index = WEvent.index('procedure')
        if not (WEvent[procedure_index - 1] in surgery_stop_adjs):  #
         procedure的前一个word
            Surgery_Type = WEvent[procedure_index - 1]
            for W in [WEvent[procedure_index - 2],
             WEvent[procedure_index - 3]]:
                if not (W in surgery_stop_adjs):
                    Surgery_Type = W + ' ' + Surgery_Type  #
                     利用procedure的前三个word组成surgery
                     type (要保证word不在surgery_stop_adjs里)
                else:
                    break
            # print Surgery_Type;

# surgery class的替换
if (Surgery_Class.find('Urology') > -1):
    Surgery_Class = 'Urologic'

if Surgery_Type.lower().find('valve') > -1:
    Cardiac_Type = 'Valve Repair'
elif Surgery_Type.lower().find('thoracic') > -1:
    Cardiac_Type = 'Thoracic'
elif Surgery_Type.lower().find('thymectomy') > -1:
    Cardiac_Type = 'Thymectomy'
elif Surgery_Type.lower().find('lobectomy') > -1:
    Cardiac_Type = 'Lobectomy'
elif Surgery_Type.lower().find('thoracoscopy') > -1:
    Cardiac_Type = 'Thoracoscopy'
elif Surgery_Type.lower().find('lung') > -1:
    Cardiac_Type = 'Lung'
else:
    for k in ['coronary artery', 'coronary arteries', 'cabg', 'bypass
     grafting', 'tecab']:
        if Surgery_Type.lower().find(k) > -1:
            Cardiac_Type = 'Coronary Bypass Grafting'

# print Surgery_Type
# print Surgery_Class

##    if (Surgery_Class == 'N/A'):
##        for s in SEvent:
##            if ((s.find('legal')>-1) or (s.find('dispute')>-1)):
##                print s
##                legal_count = legal_count +1;
##                Surgery_Class = 'N/A: legal'
##                break
##        for s in SEvent:
##            if ((s.find('prior to start')>-1) or (s.find('no patient
 use')>-1)):
##                print s
##                prior_count = prior_count +1;
##                Surgery_Class = 'N/A: prior'
##                break
```

```
if MDR_Key in ['2965965', '3404382', '3401370', '3404905', '3403409',
 '3459899',
               '773074', '2718569', '2721087', '3419846', '2913443',
                '2957935',
               '2440628', '2442419', '886632']:
    Surgery_Class = 'Urologic'
elif MDR_Key in ['3404443', '3404577', '3404550', '3345253', '3183187',
 '3113354']:
    Surgery_Class = 'Gynecologic'
elif MDR_Key in ['3385091', '2966014', '3403784', '3388777', '3320132',
 '2912030', ]:
    Surgery_Class = 'General'
elif MDR_Key in []:
    Surgery_Class = 'Colorectal'
if MDR_Key in []:
    Surgery_Class = 'Cardiothoracic'
elif MDR_Key in ['3385079', '3473319', '3128668']:
    Surgery_Class = 'Head and Neck'

if (Surgery_Type == ''):
    Surgery_Type = 'N/A'  # ljn added

#### Misscategorized Injuries
Injuries = []
found = 0
# If there is a injury keyword in those that are D or IN and if it is
 not in the filtered list
if ((Impact != 'D') and (Impact != 'IN') and (
not MDR_Key in Typoed_MDRs)):  #
 这里的Imapct就是Event_Type，D代表death，IN代表injury
    Pre_Impact = Impact
    for s in SEvent:
        for IK in Injury_Keywords:
            if ((s.find(IK) > -1) and not (negation_check(s, IK))):  #
             发现了injury等关键词，且并没有否定词
                # For injury keywords it is enough to see them
                if (IK.find('injur') > -1) or (IK.find('bleed') > -1):
                    found = 1
                # For the cut, the mistake keyword should be before and
                 the patient keyword after
                elif (IK.find('cut') > -1):
                    cut_index = s.find('cut')
                    for B in Body_Keywords:
                        if (s[cut_index:cut_index + 20].find(B) > -1):
                            for I in Mistake_Keywords:
                                if s[cut_index - 20:cut_index].find(I)
                                 > -1:
                                    found = 1
                                    break
                # For others we also need to have patient involved.
                else:
                    for B in Body_Keywords:
                        if (s.find(B) > -1):
```

```
                                # For the damage to and puncture to, the
                                 patient's keyword should be after
                                if ((IK.find('to') > -1 or IK.find('on') >
                                 -1 or
                                     IK.find('in') > -1 or IK.find('ed') >
                                      -1) and
                                        (s.find(B) - s.find(IK) <= 40) and
                                         (s.find(IK) < s.find(B))):
                                    found = 1
                                    break
                                elif ((IK.find('ed') > -1) and
                                 (s.find('was') > -1) and
                                      (s.find('was') < s.find(IK)) and
                                       (s.find(IK) - s.find('was') <= 5) and
                                      (s.find(B) < s.find('was')) and
                                       (s.find('was') - s.find(B) <= 20)):
                                    found = 1
                                    break
                                elif (IK.find('to') == -1 and IK.find('on')
                                 == -1 and
                                      IK.find('in') == -1 and IK.find('ed')
                                       == -1):
                                    found = 1
                                    break
                        if (found == 1):
                            if (Injuries.count(IK) < 1):
                                Injuries.append(IK)
                                # print Injuries;
                            break
                if (found == 1):
                    break

# If found any of the injury keywords, reclassify the event to Injury
if len(Injuries) > 0:
    missed_injuries = missed_injuries + 1
    Impact = 'IN'
# if (Impact == 'INJ' or Impact == 'INJO' or Impact == 'INJM' or Impact
 == 'BLDO'):
#    Impact = 'IN';
# if (Impact == 'COMPO' or Impact == 'CMPO' or Impact == '*'):
#    Impact = 'O';

elif (MDR_Key in Other_Missed):
    missed_injuries = missed_injuries + 1
    Injuries.append('bleed')
    Impact = 'IN'
elif (MDR_Key == '2559247'):
    Impact = 'D'

#### System Error?
System_Error = 'N/A'
Curr_Error_Codes = []
for s in SEvent:
    s_digits = []
```

```
    s_nodigit = ' '.join(''.join([i for i in s if not
     i.isdigit()]).split())
    # Look for any system errors
    for keyword in error_keywords:  # 分析故障是否是由system
     error导致，以及system error的类型
        # Check if the term is negated
        if (((s.find(keyword) != -1) and not (negation_check(s,
         keyword))) or
                ((s_nodigit.find(keyword) != -1) and not
                 (negation_check(s_nodigit, keyword)))):
            if (System_Error == 'N/A'):
                System_Error = keyword
            # Getting any possible error codes
            kindex = s.find(keyword)
            s_digits = re.findall(r'\d+', s[kindex - 10:kindex - 1]) +
             re.findall(r'\d+', s[kindex + len(
                keyword):kindex + len(keyword) + 10])
            if (s_digits != None):
                for EC in s_digits:
                    # If we didn't already see this error code before
                    if (Curr_Error_Codes.count(EC) == 0):
                        Curr_Error_Codes.append(EC)
                        if (Error_Code_List.has_key(EC)):
                            Error_Code_List[EC] =
                             (Error_Code_List[EC][0] + 1,
                             Error_Code_List[EC][1])
                        else:
                            Error_Code_List[EC] = (1, '')
            f6.write(s + '\n')
            break
    if (System_Error != 'N/A'):
        break
# If not found in Event, search in Narrative
if (System_Error == 'N/A'):
    for s in SNarrative:
        s_digits = []
        s_nodigit = ' '.join(''.join([i for i in s if not
         i.isdigit()]).split())
        # Look for any system errors
        for keyword in error_keywords2:
            # Check if the term is negated
            if (((s.find(keyword) != -1) and not (negation_check(s,
             keyword))) or
                    ((s_nodigit.find(keyword) != -1) and not
                     (negation_check(s_nodigit, keyword)))):
                if (System_Error == 'N/A'):
                    System_Error = 'N ' + keyword
                # Getting any possible error codes
                kindex = s.find(keyword)
                s_digits = re.findall(r'\d+', s[kindex - 10:kindex -
                 1]) + re.findall(r'\d+', s[kindex + len(
                    keyword):kindex + len(keyword) + 10])
                if (s_digits != None):
                    for EC in s_digits:
```

```
                        # If we didn't already see this error code
                         before
                        if (Curr_Error_Codes.count(EC) == 0):
                            Curr_Error_Codes.append(EC)
                            if (Error_Code_List.has_key(EC)):
                                Error_Code_List[EC] =
                                 (Error_Code_List[EC][0] + 1,
                                 Error_Code_List[EC][1])
                            else:
                                Error_Code_List[EC] = (1, '')
                f6.write(s + '\n')
                break
        if (System_Error != 'N/A'):
            break
# If not found in Event and Narrative, look for error codes
if (System_Error == 'N/A'):
    for s in SEvent:
        words = nltk.word_tokenize(s)
        for w in words:
            if (w.find('fault') > -1 or w.find('error') > -1):
                if words[words.index(w) - 1].isdigit():
                    System_Error = words[words.index(w) -
                     1:words.index(w)]
                    break
        if (System_Error != 'N/A'):
            break
# Recoverable or Non-Recoverable
if (System_Error != 'N/A'):
    f8.write(MDR_Key + '\n')
    # Give priority to Event Description because if to the surgeon the
     error was non-recoverable,
    # it doesn't matter if the company said it was recoverable !!!!
    if ((CEvent.find('non recoverable') != -1) or
     (CEvent.find('non-recoverable') != -1) or
            (CEvent.find('nonrecoverable') != -1) or
             (CEvent.find('irrecoverable') != -1) or
            (CEvent.find('unrecoverable') != -1) or (CEvent.find('non-
             recoverable') != -1)):
        System_Error = 'Not Recoverable'
    elif (CEvent.find('recoverable') != -1):
        System_Error = 'Recoverable'
    elif ((CNarrative.find('non recoverable') != -1) or
     (CNarrative.find('non-recoverable') != -1) or
          (CNarrative.find('nonrecoverable') != -1) or
           (CNarrative.find('irrecoverable') != -1) or
          (CNarrative.find('unrecoverable') != -1) or
           (CNarrative.find('non- recoverable') != -1)):
        System_Error = 'Not Recoverable'
    elif (CNarrative.find('recoverable') != -1):
        System_Error = 'Recoverable'

#### Vision Problems
Vision = 'N/A'
for keyword in vision_keywords:
    if (CEvent.find(keyword) != -1):
```

```
        Vision = keyword
        break
if Vision == 'N/A':
    if (CNarrative.find('was associated with the high resolution stereo
     viewer (hrsv)') != -1):
        Vision = 'reason: hrsv'
    else:
        loc = Narrative.lower().find('was associated with')
        if (loc != -1):
            rest = Narrative[loc:].split('.')[0].split('with')[1][0:40]
            if (rest.lower().find('camera') != -1) and
             (rest.lower().find('manipulator') == -1):
                Vision = 'reason: camera'
                # print rest;
# some manual fixing of false alarms.
if (MDR_Key in ['1584210', '2381732', '2664013', '961602', '3140434',
 '923148', '3489457']):
    Vision = 'N/A'
if ((Vision.find('no video') != -1) and
        ((CEvent.find('no video tape') != -1) or (CEvent.find('no video
         recording') != -1) or (
                CEvent.find('no video doc') != -1))):
    # print CEvent;
    Vision = 'N/A'

if (Pre_Data.has_key(MDR_Key) and (Pre_Data[MDR_Key][7] != 'N/A') and
 (Vision == 'N/A')):  # ljn added
    Vision = Pre_Data[MDR_Key][7]
# if((Pre_Data[MDR_Key][7] != 'N/A') and (Vision == 'N/A')): #
 对2019年的数据，这里可能会出错？？？
#     #print MDR_Key
#     #print 'was: '+Pre_Data[MDR_Key][7]+' became: '+Vision+'\n'
#     Vision = Pre_Data[MDR_Key][7]

#### Broken?
prefix_str = ''
pre_word = ''
post_word = ''
scnt = 0
# Get rid of spaces
Instruments = [inst_k.strip() for inst_k in Instruments]
Broken = 'N/A'
for s in SEvent + SNarrative:
    if scnt >= len(SEvent):
        # prefix_str = 'Narr: '
        prefix_str = ''
    for keyword in broken_keywords:
        if (s.find(keyword) > -1) and not (negation_check(s, keyword)):
            words = nltk.word_tokenize(s)
            for w in words:
                if w in ['broken', 'damaged']:
                    if (not (words[words.index(w) - 1] in ['are', 'is',
                     'was', 'were']) and
```

```
                            not (words[words.index(w) - 2] in ['are',
                             'is', 'was', 'were'])):
                        for inst_k in Instruments:
                            if (words.index(w) < len(words) - 1) and (
                                    inst_k in words[words.index(w) +
                                     1:words.index(w) + 3]):
                                Broken = prefix_str + w + ' ' + inst_k
                                broken_cnt = broken_cnt + 1
                                break
                    elif ((w == 'damage') and (words.index(w) < len(words)
                     - 1) and
                          (words[words.index(w) + 1] in ['to', 'into']) and
                          (not words[words.index(w) - 1] in Instruments)):
                        for inst_k in Instruments:
                            if (inst_k in words[words.index(w) +
                             1:words.index(w) + 3]):
                                Broken = prefix_str + w + ' ' + inst_k
                                broken_cnt = broken_cnt + 1
                                break
                    if (Broken == 'N/A') and (
                            w in ['break', 'broke', 'broken', 'damage',
                             'damaged', 'returned', 'snapped']):
                        for inst_k in Instruments:
                            if (words.index(w) < len(words) - 1) and (
                                    inst_k in words[words.index(w) +
                                     1:words.index(w) + 5]):
                                Broken = prefix_str + w + ' ' + inst_k
                                broken_cnt = broken_cnt + 1
                                break
                            if (words.index(w) > 4) and (inst_k in
                             words[words.index(w) - 5:words.index(w)]):
                                Broken = prefix_str + w + ' ' + inst_k
                                broken_cnt = broken_cnt + 1
                                break
                    if (Broken != 'N/A'):
                        break
                if (Broken != 'N/A'):
                    break
        if (Broken != 'N/A'):
            break
        else:
            for ss in ['came off', 'coming off', 'come off']:
                if (s.find(ss) > -1) and not (negation_check(s, ss)) and
                 not (Impact == 'D'):
                    Broken = ss
                    break
            if (Broken != 'N/A'):
                break
        scnt = scnt + 1
        all_broken.append(Broken)  # ljn added

    # if((Pre_Data[MDR_Key][5] == 'N/A') and (Broken != 'N/A')):
    #    print MDR_Key
    #    print 'was: '+Pre_Data[MDR_Key][5]+' became: '+Broken+'\n'
```

```
#### Fallen?
Fallen = 'N/A'
prefix_str = ''
scnt = 0
for s in SEvent:
    # Break the sentence into words
    words = nltk.word_tokenize(s)
    # tags = pos_tag(words);
    found_pairs = []
    found_tags = ''
    found_str = ''
    first_keyword = ''
    for w in words:
        # Bigram verbs
        if ((w == 'lost' and (words[words.index(w) - 1] == 'was' or
         words[words.index(w) - 1] == 'became')) or
                (w == 'found' and words[words.index(w) - 1] == 'was') or
                (w == 'noticed' and words[words.index(w) - 1] == 'was')
                 or
                # If something was observed inside the patient, but not
                 a burn or sparking
                (w == 'observed' and words[words.index(w) - 1] == 'was'
                 and
                 list(set(words[0:words.index(w) - 1]) &
                  set(burn_keywords)) == []) or
                (w == 'seen' and words[words.index(w) - 1] == 'was') or
                ((w == 'apart' or w == 'off') and words[words.index(w)
                 - 1] == 'came')):
            found_pairs.append((w, 'V'))
            found_tags = found_tags + ' ' + 'V'
            found_str = found_str + ' ' + w
        # Bigram nouns
        elif ((w == 'field' and (
                words[words.index(w) - 1] == 'surgical' or
                 words[words.index(w) - 1] == 'sterile')) or
              (w == 'cavity' and words[words.index(w) - 1] ==
               'abdominal')):
            found_pairs.append((w, 'NN'))
            found_tags = found_tags + ' ' + 'NN'
            found_str = found_str + ' ' + w
        # Unigrams
        else:
            for (keyword, tag) in fallen_keywords2:
                if w == keyword:
                    found_pairs.append((keyword, tag))
                    if tag.find('NN') != -1:
                        last_keyword = keyword
                    found_tags = found_tags + ' ' + tag
                    found_str = found_str + ' ' + keyword
    # If any of the keywords was found, look for the patterns of V, IN,
     NN
    if (found_tags != []):
        if (found_tags.find('V IN NN') != -1) or (found_tags.find('V IN
         IN NN') != -1):
            # print found_pairs
```

```
            # Check if the term is negated
            # loc = s.find(last_keyword);
            if not (negation_check(s, last_keyword)):
                Fallen = found_str
                csv_wr2.writerow([MDR_Key, Narrative, Event, Impact,
                 found_str])
                fallen_cnt1 = fallen_cnt1 + 1

    # If not found, search for any of these keywords:
    if (Fallen == 'N/A'):
        for keyword in fallen_keywords3:
            kindex = s.find(keyword)
            if (kindex > -1):
                for k in peice_keywords:
                    if (s[0:kindex].find(k) > -1):
                        Fallen = k + ' ' + keyword
                        fallen_cnt1 = fallen_cnt1 + 1
                        break;
                        # If already found, don't process the rest of
                         sentences
    else:
        break
# If not found, search for any of these keywords:
if (Fallen == 'N/A'):
    for s in SEvent + SNarrative:
        if (s.find('broke off') > -1) or (s.find('break off') > -1) or
         (s.find('broken off') > -1):
            for inst_k in Instruments:
                if (s.find(inst_k) > -1):
                    Fallen = inst_k + ' broke off'
                    fallen_cnt1 = fallen_cnt1 + 1
                    break
        elif ((s.find('broke into pieces') > -1) or (s.find('break into
         pieces') > -1) or (
                s.find('broken into pieces') > -1) or
              (s.find('pieces were removed') > -1) or (s.find('pieces
               were retrieved') > -1) or (
                      s.find('pieces were recovered') > -1) or
              (s.find('fragments were removed') > -1) or
               (s.find('fragments were retrieved') > -1) or (
                      s.find('fragments were recovered') > -1) or
              (s.find(' shredd') > -1) or (s.find('fragment was
               removed') > -1) or (
                      s.find('fragment was retrieved') > -1) or
              (s.find('piece was removed') > -1) or (s.find('piece was
               retrieved') > -1) or
              (s.find('fragment removed') > -1) or (s.find('fragment
               retrieved') > -1) or (
                      s.find('piece removed') > -1) or (s.find('piece
                       retrieved') > -1) or
              (s.find('fragments removed') > -1) or (s.find('fragments
               retrieved') > -1) or (
                      s.find('pieces removed') > -1) or (s.find('pieces
                       retrieved') > -1) or
```

```
                (s.find('foreign object was removed') > -1) or
                 (s.find('foreign object was retrieved') > -1) or
                (s.find('foreign object removed') > -1) or
                 (s.find('foreign object retrieved') > -1) or
                (s.find('recovered from the pt') > -1) or
                 (s.find('recovered from the abdom') > -1) or
                (s.find('retrieved from the pt') > -1) or
                 (s.find('retrieved from the abdom') > -1) or
                (s.find('found and removed') > -1) or (s.find('found and
                 retrieved') > -1)):
                Fallen = 'broke into pieces'
                fallen_cnt1 = fallen_cnt1 + 1
            else:
                for pword in ['piece', 'fragment']:
                    if Broken.find(pword) > -1:
                        Fallen = 'Broken ' + pword
                        fallen_cnt1 = fallen_cnt1 + 1
                        break
            # If already found, don't process the rest of sentences
            if (Fallen != 'N/A'):
                break

    # Narrative + Broken piece were successfully retrieved from the patient
     + Missing pieces found or not found
    # X-ray of CT-scan confirmed the location of pieces
    # 1416362 => decided to not remove
    if MDR_Key in ['3430596', '2691480', '2691467', '1770807',
                   '955424', '952458', '1595497', '476270', '3291549',
                   '3484699', '3360346', '1473448', '1416362', '1403415',
                   '853271', '321550', '3430596', '1595497', '3484699',
                   '3406785', '2829495', '2723133', '2399143', '1884284',
                   '1855009', '2037070', '1998148', '1028543', '799459',
                   '1566432', '3375033', '3519245', '3090826', '3368835']:
        Fallen = 'Narr/Missing Fallen'
        fallen_cnt1 = fallen_cnt1 + 1
    # Fix cases and add prefix
    if MDR_Key in ['3444257']:
        Fallen = 'N/A'

    # if ((Pre_Data[MDR_Key][1] != 'N/A') and (Fallen == 'N/A')):
    # print MDR_Key
    # print 'was: '+Pre_Data[MDR_Key][1]+' became: '+Fallen+'\n'
    # Fallen = Pre_Data[MDR_Key][1];

    #### Electrical Arcing?
    Arced = 'N/A'
    for s in SEvent + SNarrative:
        for keyword in arcing_keywords:
            # Check if the term is negated
            if ((s.find(keyword) != -1) and not (negation_check(s,
             keyword))):
                Arced = keyword
                break
        if (Arced == 'N/A'):
```

```
        if ((s.find('charr') != -1) and (s.find('tissue') == -1) and
         not (negation_check(s, 'charr'))):
            Arced = 'Charring'
    else:
        break
if (Pre_Data.has_key(MDR_Key) and (Pre_Data[MDR_Key][4] != 'N/A') and
 (Arced == 'N/A')):
    # print MDR_Key
    # print 'was: '+Pre_Data[MDR_Key][4]+' became: '+Arced+'\n'
    Arced = Pre_Data[MDR_Key][4]

#### Tip Cover Accessory?
Tip_Cover = 'N/A'
for s in SEvent + SNarrative:
    if (s.find('tip cover') != -1) or (s.find('tip cover accessory') !=
     -1) or (
            s.find('tip cover accessories') != -1):
        for keyword in tip_cover_keywords:
            if (s.find(keyword) != -1) and not (negation_check(s,
             keyword)):
                Tip_Cover = 'Tip_Cover' + keyword
                break
    if Tip_Cover != 'N/A':
        break
if MDR_Key in ['3354904']:
    Tip_Cover = 'N/A'

if (Pre_Data.has_key(MDR_Key) and (Pre_Data[MDR_Key][6] != 'N/A') and
 (Tip_Cover == 'N/A')):
    # print MDR_Key
    # print 'was: '+Pre_Data[MDR_Key][6]+' became: '+Tip_Cover+'\n'
    Tip_Cover = Pre_Data[MDR_Key][6]

#### Moving?
Moved = 'N/A'
for s in SEvent:
    words = nltk.word_tokenize(s)
    # tags = pos_tag(words);
    found_pairs = []
    found_tags = ''
    found_str = ''
    first_keyword = ''
    for w in words:
        for inst_k in ['instrument']:
            if (w == inst_k) and (words.index(w) + 1 < len(words)):
                next_w = words[words.index(w) + 1]
                if (all_inst_postfix.count(next_w) == 0):
                    all_inst_postfix.append(next_w)

        # Not Recognized
        if ((w == 'recognized' and words[words.index(w) - 1] == 'not')
         or
                (w == 'recongize' and words[words.index(w) - 1] ==
                 'not' and words[words.index(w) - 1] == 'did') or
```

```
                (w == 'recognition' and words[words.index(w) - 1] ==
                 'instrument') or
                (w == 'recognizing' and words[words.index(w) - 1] ==
                 'not')):
            Moved = w + ' ' + words[words.index(w) - 1]
            break
        # POS Patterns
        else:
            for (keyword, tag) in moving_keywords:
                if w == keyword:
                    found_pairs.append((keyword, tag))
                    found_tags = found_tags + ' ' + tag
                    found_str = found_str + ' ' + keyword
    # If any of the keywords was found, look for the patterns of: N-V,
     V-IN-NN
    if (found_tags != []):
        if (found_tags.find('NG V') != -1) or (found_tags.find('V IN
         NN') != -1) or (
                found_tags.find('IN NN') != -1):
            # print found_pairs
            Moved = found_str
            break

    # If not found, search for any of these keywords:
    if (Moved == 'N/A'):
        if (s.find('instrument') != -1) and ((s.find('not respond') !=
         -1) or (s.find('not always respond') != -1)):
            Moved = 'not respond'
        elif ((s.find('no recognize') != -1) or (s.find('not
         cauterize') > -1) or (s.find('not line up') > -1) or
              (s.find('not match up') > -1) or (s.find('not coagulate')
               > -1) or (s.find('not retract') > -1) or
              (s.find('not cut') > -1) or (s.find('not engage') > -1)
               or (s.find('not align') > -1) or
              (s.find('not seal') > -1)):
            Moved = 'not !!!'
        elif ((s.find('full range of motion') != -1) or (s.find('motion
         sensor mismatch') != -1) or
              (s.find('experience limited movement') != -1) or
               (s.find('had difficulty moving') != -1)):
            Moved = 'Motion'

    if (Moved != 'N/A'):
        break
if MDR_Key in ['2975656', '878764', '3444332', '3420104', '3348093',
               '1576670', '1559178', '271460', '2147492', '2807222']:
    Moved = 'N/A'

if (Pre_Data.has_key(MDR_Key) and (Pre_Data[MDR_Key][3] != 'N/A') and
 (Moved == 'N/A')):
    # print MDR_Key
    # print 'was: '+Pre_Data[MDR_Key][3]+' became: '+Moved+'\n'
    Moved = Pre_Data[MDR_Key][3]

#### Power Problems
```

```
Power = 'N/A'
##    power_keywords = ['power']
##    for s in SEvent:
##        for keyword in power_keywords:
##            if (s.find(keyword) != -1) and not(negation_check(s,
 keyword)):
##                Power = keyword;
##                break;
##        if Power != 'N/A':
##            break;
##    if Power == 'N/A':
##        for s in SNarrative:
##            for keyword in power_keywords:
##                if (s.find(keyword) != -1) and not(negation_check(s,
 keyword)):
##                    Power = keyword;
##                    break;
##            if Power != 'N/A':
##                break

#### Injuries that involved Malfunctions
if IN_MALFUNC.has_key(MDR_Key):
    if (IN_MALFUNC[MDR_Key].find('Fell') != -1):
        Fallen = IN_MALFUNC[MDR_Key]
    if (IN_MALFUNC[MDR_Key].find('System Error') != -1):
        System_Error = IN_MALFUNC[MDR_Key]
    if (IN_MALFUNC[MDR_Key].find('Broken') != -1):
        Broken = IN_MALFUNC[MDR_Key]
    if (IN_MALFUNC[MDR_Key].find('Arcing') != -1):
        Arced = IN_MALFUNC[MDR_Key]
    if (IN_MALFUNC[MDR_Key].find('Tip Cover') != -1):
        Tip_Cover = IN_MALFUNC[MDR_Key]
    if (IN_MALFUNC[MDR_Key].find('Unintended Operation') != -1):
        Moved = IN_MALFUNC[MDR_Key]
    if (IN_MALFUNC[MDR_Key].find('Other') != -1):
        Other = IN_MALFUNC[MDR_Key] + ' IN'

#### Other Malfunctions that we didn't review but were reported as
 malfunction
if ((System_Error == 'N/A') and (Moved == 'N/A') and (Arced == 'N/A')
 and
        (Fallen == 'N/A') and (Tip_Cover == 'N/A') and (Vision ==
         'N/A') and
        (Other == 'N/A')):
    if (Impact == 'M') or (Broken != 'N/A'):
        Other = 'Other M'
    else:
        for s in SNarrative:
            if (s.find('system issue experienced') > -1) or
             (s.find('the issue experienced by') > -1):
                Other = 'Other O'
                break

#### A System reset happened?
Reset = 'N/A'
```

```
for keyword in reset_keywords:
    if (CNarrative.find(keyword) > 0 or CEvent.find(keyword) > 0):
        Reset = 'System_Reset'
        break

#### Surgery was converted? or reschedulued?
Converted = 'N/A'
Rescheduled = 'N/A'
# Give priority to Event Description and the one that comes at the end
for keyword in convert_keywords:
    conv_loc = CEvent.find(keyword)
    if (conv_loc != -1):
        # print 'Converted'
        Converted = 'Converted'
        break
for keyword in reschedule_keywords:
    resc_loc = CEvent.find(keyword)
    if (resc_loc != -1):
        # print 'Rescheduled'
        Rescheduled = 'Rescheduled'
        break

if (Converted != 'N/A') and (Rescheduled != 'N/A'):
    # print CEvent
    if (resc_loc > conv_loc):
        Converted = 'N/A'
        # print 'Rescheduled'
    else:
        Rescheduled = 'N/A'
        # print 'Converted'
    time.sleep(10)

if (Converted == 'N/A') and (Rescheduled == 'N/A'):
    for keyword in convert_keywords:
        conv_loc = CNarrative.find(keyword)
        if (conv_loc != -1):
            # print 'Converted'
            Converted = 'Converted'
            break
    for keyword in reschedule_keywords:
        resc_loc = CNarrative.find(keyword)
        if (resc_loc != -1):
            # print 'Rescheduled'
            Rescheduled = 'Rescheduled'
            break

    if (Converted != 'N/A') and (Rescheduled != 'N/A'):
        # print CNarrative
        if (resc_loc > conv_loc):
            Converted = 'N/A'
            # print 'Rescheduled'
        else:
            Rescheduled = 'N/A'
            # print 'Converted'
        time.sleep(10)
```

```
    ### Write the output
    if (daVinci == 0):
        response = [MDR_Key, Surgery_Keyword, Narrative, Event, Impact,
         Outcome, Injuries,
                    Manufacturer, Brand_Name, Generic_Name,
                     Device_Category, Product_Code,
                    Manufacture_Year, Event_Year, Corrected_Event_Year,
                     Report_Year, Time_to_Event, Time_to_Report,
                    Surgery_Type, Surgery_Class, Cardiac_Type, Converted,
                     Rescheduled,
                    Reset, Fallen, System_Error,
                    Moved, Arced, Broken, Tip_Cover, Vision, Power, Other]
    else:
        response = [MDR_Key, Narrative, Event, Impact, Outcome, Injuries,
                    Manufacturer, Brand_Name, Generic_Name, Product_Code,
                    Manufacture_Year, Event_Year, Corrected_Event_Year,
                     Report_Year, Time_to_Event, Time_to_Report,
                    Surgery_Type, Surgery_Class, Converted, Rescheduled,
                    Reset, Fallen, System_Error,
                    Moved, Arced, Broken, Tip_Cover, Vision, Power, Other]
    # Write to CSV file
    csv_wr.writerow(response)
    # Write to Excel file
    for k in range(0, len(response)):
        worksheet.write(chk_sum, k, response[k])
        # Save the data for current row
    all_data.append(response)
    chk_sum += 1
    # Update the progress bar
    if (chk_sum % RStep == 0):
        sys.stdout.write("\r%d%%.." % (chk_sum / RStep * 10))
        sys.stdout.flush()

sys.stdout.write("\rDone!")
sys.stdout.flush()
# Close the Excel file
workbook.save(Excel_Out)

for key in Error_Code_List.keys():
    Error_Code = str(key)
    Frequency = str(Error_Code_List[key][0])
    Description = str(Error_Code_List[key][1])
    f7.write(Error_Code + ',' + Frequency + ',' + Description + '\n')

all_broken.sort()  # 根据就没有记录all_broken
for a in all_broken:
    f9.write(a.strip() + '\n')

# Close the files
f0.close()
f0_1.close()
f0_2.close()
f0_3.close()
```

```
f1.close()
f2.close()
f3.close()
f4.close()
f5.close()
f6.close()
f7.close()
f8.close()
f9.close()
print '\n' + str(chk_sum - 1) + ' reports were read and written'
print str(count1) + ' NegEx Negations'
print str(count2) + ' No/Not/Nothing'
print str(fallen_cnt1) + ' Fallen cases found by Part of Speech Tagging and
 Pattern Matching'
print str(fallen_cnt2) + ' Fallen cases found by Keyword Searching'
print str(broken_cnt) + ' Broken cases found by Keyword Searching'
print str(missed_injuries) + ' Missed Injutries found by the keyword
 searching'

'''
24393 reports were read and written
0 NegEx Negations
0 No/Not/Nothing
2260 Fallen cases found by Part of Speech Tagging and Pattern Matching
0 Fallen cases found by Keyword Searching
10102 Broken cases found by Keyword Searching
299 Missed Injutries found by the keyword searching'''
```