```
import pandas as pd
import re


#!/usr/bin/python3
"""
    Utility programs for accessing SEC/EDGAR
    ND-SRAF / McDonald : 201606
    https://sraf.nd.edu
    Modified to handle modern SEC access requirements
"""
import time
from urllib.request import Request, urlopen
from zipfile import ZipFile
from io import BytesIO
import random

def download_masterindex(year, qtr, flag=False):
    # Download Master.idx from EDGAR
    # Loop accounts for temporary server/ISP issues
    # ND-SRAF / McDonald : 201606 (Modified 2024)
    number_of_tries = 10
    sleep_time = 10  # Note sleep time accumulates according to err
    PARM_ROOT_PATH = 'https://www.sec.gov/Archives/edgar/full-index/'
    start = time.time()  # Using time.time() instead of time.clock()
    masterindex = []
    append_path = f"{year}/QTR{qtr}/master.zip"
    sec_url = PARM_ROOT_PATH + append_path

    headers = {
        'User-Agent': 'NYU dj2436@nyu.edu',
        'Accept-Encoding': 'gzip, deflate',
        'Host': 'www.sec.gov'
    }

    for i in range(1, number_of_tries + 1):
        try:
            req = Request(sec_url, headers=headers)
            with urlopen(req) as response:
                zipfile = ZipFile(BytesIO(response.read()))
                records = zipfile.open('master.idx').read().decode('utf-8', 'ignore').split
            break
        except Exception as exc:
            if i == 1:
                print('\nError in download_masterindex')
            print(f'  {i}. *url:  {sec_url}')
            print(f'  Warning: {str(exc)}  [{time.strftime("%c")}]')
            if '404' in str(exc):
                break
            if i == number_of_tries:
```

```
            if i == number_of_tries:
                return False
            sleep_time_with_jitter = sleep_time + random.uniform(1, 5)
            print(f'     Retry in {sleep_time_with_jitter:.2f} seconds')
            time.sleep(sleep_time_with_jitter)
            sleep_time *= 2  # Exponential backoff

    # Load m.i. records into masterindex list
    for line in records:
        mir = MasterIndexRecord(line)
        if not mir.err:
            masterindex.append(mir)

    if flag:
        print(f'download_masterindex:  {year}:{qtr} | '
              f'len() = {len(masterindex):,} | Time = {time.time() - start:.4f} seconds')

    return masterindex
class MasterIndexRecord:
    def __init__(self, line):
        self.err = False
        parts = line.split('|')
        if len(parts) == 5:
            self.cik = int(parts[0])
            self.name = parts[1]
            self.form = parts[2]
            self.filingdate = int(parts[3].replace('-', ''))
            self.path = parts[4]
        else:
            self.err = True
        return


def edgar_server_not_available(flag=False):
    from datetime import datetime as dt
    import pytz
    import time

    SERVER_BGN = 6  # Server opens at 9:00PM EST
    SERVER_END = 21   # Server closes at 6:00AM EST

    # Get UTC time from local and convert to EST
    utc_dt = pytz.utc.localize(dt.utcnow())
    est_timezone = pytz.timezone('US/Eastern')
    est_dt = est_timezone.normalize(utc_dt.astimezone(est_timezone))

    if est_dt.hour >= SERVER_BGN or est_dt.hour < SERVER_END:
        return False
    else:
        if flag:
            print('\rSleeping: because the server is not yet open' + str(dt.now()), end='',
```

```
        time.sleep(600)  # Sleep for 10 minutes
        return True


#!/un3 sr/bin/Python
"""
    Program to download EDGAR files by form type
    ND-SRAF / McDonald : 201606
    https://sraf.nd.edu
    Dependencies (i.e., modules you must already have downloaded)
      EDGAR_Forms.py
      EDGAR_Pac.py
      General_Utilities.py
"""
import pandas as pd
import os
import re
import time
import sys
# Modify the following statement to identify the path for local modules
# sys.path.append('D:\GD\Python\TextualAnalysis\Modules')
# Since these imports are dynamically mapped your IDE might flag an error...it's OK
import EDGAR_Forms  # This module contains some predefined form groups
import EDGAR_Pac
import General_Utilities
import gen_util



# * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * +

#  NOTES
#       The EDGAR archive contains millions of forms.
#       For details on accessing the EDGAR servers see:
#         https://www.sec.gov/edgar/searchedgar/accessing-edgar-data.htm
#       From that site:
#           "To preserve equitable server access, we ask that bulk FTP
#            transfer requests be performed between 9 PM and 6 AM Eastern
#            time. Please use efficient scripting, downloading only what you
#            need and space out requests to minimize server load."
#       Note that the program will check the clock every 10 minutes and only
#           download files during the appropriate time.
#       Be a good citizen...keep your requests targeted.
#
#       For large downloads you will sometimes get a hiccup in the server
#           and the file request will fail.  These errs are documented in
#           the log file.  You can manually download those files that fail.
#           Although I attempt to work around server errors, if the SEC's server
#           is sufficiently busy, you might have to try another day.
#
#      For a list of form types and counts by year:
```

```
#         "All SEC EDGAR Filings by Type and Year"
#          at https://sraf.nd.edu/textual-analysis/resources/#All%20SEC%20EDGAR%20Filings


# -----------------------
# User defined parameters
# -----------------------

# List target forms as strings separated by commas (case sensitive) or
#   load from EDGAR_Forms.  (See EDGAR_Forms module for predefined lists.)
# PARM_FORMS = EDGAR_Forms.f_10K  # or, for example, PARM_FORMS = ['8-K', '8-K/A']
PARM_FORMS = ['10-Q'] # ONLY need 10-Q report
PARM_BGNYEAR = 2019  # User selected bgn period.  Earliest available is 1994
PARM_ENDYEAR = 2023  # User selected end period.
PARM_BGNQTR = 1  # Beginning quarter of each year
PARM_ENDQTR = 4  # Ending quarter of each year
# Path where you will store the downloaded files
PARM_PATH = r'/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/Files'
# Change the file pointer below to reflect your location for the log file
#    (directory must already exist)
PARM_LOGFILE = (r'/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/' +
                r'EDGAR_Download_FORM-X_LogFile_' +
                str(PARM_BGNYEAR) + '-' + str(PARM_ENDYEAR) + '.txt')
# EDGAR parameter
PARM_EDGARPREFIX = 'https://www.sec.gov/Archives/'


#
# * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * +
with open('ticker.txt', 'r') as file:
    content = file.read()
pairs = re.findall(r'(\S+)\s+(\d+)', content)
df_cik = pd.DataFrame(pairs, columns=['Ticker', 'CIK'])
df_cik['CIK'] = df_cik['CIK'].astype(int)
df = pd.read_excel('sp500_composition.xlsx')
df['Ticker']=[tick.lower() for tick in df['Ticker']]
df['Date']=pd.to_datetime(df['Date'])
df_sub=df.loc[df['Date'] >= '2019-01-01',:]
merged_df = pd.merge(df_sub, df_cik, on='Ticker', how='left')
unique_ciks = merged_df['CIK'].dropna().unique().tolist()
sp500_cik=[int(cik) for cik in unique_ciks]

# dows=pd.read_csv("dow30.csv",header=None,index_col=0)
# ciks=pd.read_csv("cik.csv",header=None,index_col=0)
# ciks.index=[i.upper() for i in ciks.index]
# dow_cik=list(map(int,pd.concat([ciks,dows],axis=1,join="inner",sort=True).iloc[:,0].tol

def sp500filter(all_index):
    filtered=[]
    for idx in all_index:
```

```
            if idx.cik in sp500_cik:
                filtered.append(idx)
        return filtered

    def download_forms():

        # Download each year/quarter master.idx and save record for requested forms
        f_log = open(PARM_LOGFILE, 'a')
        f_log.write('BEGIN LOOPS:  {0}\n'.format(time.strftime('%c')))
        n_tot = 0
        n_errs = 0
        for year in range(PARM_BGNYEAR, PARM_ENDYEAR + 1):
            for qtr in range(PARM_BGNQTR, PARM_ENDQTR + 1):
                startloop = time.time()
                n_qtr = 0
                file_count = {}
                # Setup output path
                path = '{0}{1}/QTR{2}/'.format(PARM_PATH, str(year), str(qtr))
                if not os.path.exists(path):
                    os.makedirs(path)
                    print('Path: {0} created'.format(path))
                master_index = download_masterindex(year, qtr, True)
                masterindex= sp500filter(master_index)

    #             for record in masterindex:
    #                 if record.form=="10-Q":
    #                     print("YES")

                if masterindex:
                    for item in masterindex:
                        while edgar_server_not_available(True):  # kill time when server not
                            pass
                        if item.form in PARM_FORMS:
                            n_qtr += 1
                            # Keep track of filings and identify duplicates
                            fid = str(item.cik) + str(item.filingdate) + item.form
                            if fid in file_count:
                                file_count[fid] += 1
                            else:
                                file_count[fid] = 1
                            # Setup EDGAR URL and output file name
                            url = PARM_EDGARPREFIX + item.path
                            fname = (path + str(item.filingdate) + '_' + item.form.replace('/
                                     item.path.replace('/', '_'))
                            fname = fname.replace('.txt', '_' + str(file_count[fid]) + '.txt'

                            gen_util.respect_rate_limit()
                            return_url = gen_util.download_to_file(url, fname, f_log)
    #                         return_url = General_Utilities.download_to_file(url, fname, f_l

                            if return_url:
```

```
                                _
                            n_errs += 1
                        n_tot += 1
                        # time.sleep(1)  # Space out requests
                print(str(year) + ':' + str(qtr) + ' -> {0:,}'.format(n_qtr) + ' downloads co
                      time.strftime('%H:%M:%S', time.gmtime(time.time() - startloop)) +
                      ' | ' + time.strftime('%c'))
                f_log.write('{0} | {1} | n_qtr = {2:>8,} | n_tot = {3:>8,} | n_err = {4:>6,}
                            format(year, qtr, n_qtr, n_tot, n_errs, time.strftime('%c')))

                f_log.flush()

        print('{0:,} total forms downloaded.'.format(n_tot))
        f_log.write('\n{0:,} total forms downloaded.'.format(n_tot))


start = time.time()
print('\n' + time.strftime('%c') + '\nND_SRAF:  Program EDGAR_DownloadForms.py\n')
download_forms()
print('\nEDGAR_DownloadForms.py | Normal termination | ' +
      time.strftime('%H:%M:%S', time.gmtime(time.time() - start)))
print(time.strftime('%c'))
```

```
Thu Sep 26 13:59:41 2024
ND_SRAF:  Program EDGAR_DownloadForms.py

Path: /Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/Files2019/QTR1/ created
download_masterindex:  2019:1 | len() = 297,101 | Time = 1.4357 seconds
2019:1 -> 98 downloads completed.  Time = 00:01:21 | Thu Sep 26 14:01:03 2024
Path: /Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/Files2019/QTR2/ created
download_masterindex:  2019:2 | len() = 249,596 | Time = 0.9961 seconds
2019:2 -> 513 downloads completed.  Time = 00:06:58 | Thu Sep 26 14:08:01 2024
Path: /Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/Files2019/QTR3/ created
download_masterindex:  2019:3 | len() = 198,457 | Time = 1.5456 seconds
2019:3 -> 492 downloads completed.  Time = 00:07:10 | Thu Sep 26 14:15:11 2024
Path: /Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/Files2019/QTR4/ created
download_masterindex:  2019:4 | len() = 206,596 | Time = 1.0934 seconds
2019:4 -> 491 downloads completed.  Time = 00:06:57 | Thu Sep 26 14:22:09 2024
Path: /Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/Files2020/QTR1/ created
download_masterindex:  2020:1 | len() = 324,901 | Time = 1.5099 seconds
2020:1 -> 105 downloads completed.  Time = 00:00:36 | Thu Sep 26 14:22:45 2024
Path: /Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/Files2020/QTR2/ created
download_masterindex:  2020:2 | len() = 265,505 | Time = 1.1022 seconds
2020:2 -> 513 downloads completed.  Time = 00:02:59 | Thu Sep 26 14:25:45 2024
Path: /Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/Files2020/QTR3/ created
download_masterindex:  2020:3 | len() = 221,728 | Time = 1.4894 seconds
2020:3 -> 501 downloads completed.  Time = 00:03:49 | Thu Sep 26 14:29:35 2024
Path: /Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/Files2020/QTR4/ created
download_masterindex:  2020:4 | len() = 233,417 | Time = 1.0246 seconds
2020:4 -> 496 downloads completed.  Time = 00:03:28 | Thu Sep 26 14:33:03 2024
Path: /Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/Files2021/QTR1/ created
download_masterindex:  2021:1 | len() = 361,302 | Time = 1.5573 seconds
2021:1 -> 101 downloads completed.  Time = 00:01:23 | Thu Sep 26 14:34:27 2024
Path: /Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/Files2021/QTR2/ created
```

```
Path: /Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/Files2021/QTR2/ created
download_masterindex:  2021:2 | len() = 300,134 | Time = 1.4735 seconds
2021:2 -> 522 downloads completed.  Time = 00:07:38 | Thu Sep 26 14:42:06 2024
Path: /Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/Files2021/QTR3/ created
download_masterindex:  2021:3 | len() = 252,200 | Time = 1.2783 seconds
2021:3 -> 504 downloads completed.  Time = 00:07:18 | Thu Sep 26 14:49:25 2024
Path: /Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/Files2021/QTR4/ created
download_masterindex:  2021:4 | len() = 258,268 | Time = 1.2070 seconds
2021:4 -> 498 downloads completed.  Time = 00:07:18 | Thu Sep 26 14:56:43 2024
Path: /Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/Files2022/QTR1/ created
download_masterindex:  2022:1 | len() = 356,477 | Time = 1.3925 seconds
2022:1 -> 104 downloads completed.  Time = 00:01:19 | Thu Sep 26 14:58:03 2024
Path: /Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/Files2022/QTR2/ created
download_masterindex:  2022:2 | len() = 291,566 | Time = 1.3349 seconds
2022:2 -> 522 downloads completed.  Time = 00:07:26 | Thu Sep 26 15:05:29 2024
Path: /Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/Files2022/QTR3/ created
download_masterindex:  2022:3 | len() = 237,687 | Time = 1.1796 seconds
2022:3 -> 504 downloads completed.  Time = 00:07:18 | Thu Sep 26 15:12:48 2024
Path: /Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/Files2022/QTR4/ created
download_masterindex:  2022:4 | len() = 230,558 | Time = 1.2540 seconds
2022:4 -> 499 downloads completed.  Time = 00:07:15 | Thu Sep 26 15:20:04 2024
Path: /Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/Files2023/QTR1/ created
download_masterindex:  2023:1 | len() = 348,302 | Time = 1.4432 seconds
2023:1 -> 103 downloads completed.  Time = 00:01:25 | Thu Sep 26 15:21:29 2024
Path: /Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/Files2023/QTR2/ created
download_masterindex:  2023:2 | len() = 307,567 | Time = 1.1981 seconds
2023:2 -> 521 downloads completed.  Time = 00:08:48 | Thu Sep 26 15:30:18 2024
```

```
#!/usr/bin/python3
"""Routine to load MasterDictionary class"""
# BDM : 201510

import time
```

```
def load_masterdictionary(file_path, print_flag=False, f_log=None, get_other=False):
    _master_dictionary = {}
    _sentiment_categories = ['negative', 'positive', 'uncertainty', 'litigious', 'constra
                             'strong_modal', 'weak_modal']
    # Load slightly modified nltk stopwords.  I do not use nltk import to avoid versionin
    # Dropped from nltk: A, I, S, T, DON, WILL, AGAINST
    # Added: AMONG,
    _stopwords = ['ME', 'MY', 'MYSELF', 'WE', 'OUR', 'OURS', 'OURSELVES', 'YOU', 'YOUR',
                  'YOURSELF', 'YOURSELVES', 'HE', 'HIM', 'HIS', 'HIMSELF', 'SHE', 'H
                  'IT', 'ITS', 'ITSELF', 'THEY', 'THEM', 'THEIR', 'THEIRS', 'THEMSEL
                  'WHO', 'WHOM', 'THIS', 'THAT', 'THESE', 'THOSE', 'AM', 'IS', 'ARE'
                  'BEEN', 'BEING', 'HAVE', 'HAS', 'HAD', 'HAVING', 'DO', 'DOES', 'DI
                  'THE', 'AND', 'BUT', 'IF', 'OR', 'BECAUSE', 'AS', 'UNTIL', 'WHILE'
                  'FOR', 'WITH', 'ABOUT', 'BETWEEN', 'INTO', 'THROUGH', 'DURING', 'B
                  'AFTER', 'ABOVE', 'BELOW', 'TO', 'FROM', 'UP', 'DOWN', 'IN', 'OUT'
                  'UNDER', 'AGAIN', 'FURTHER', 'THEN', 'ONCE', 'HERE', 'THERE', 'WHE
```

```
                        'HOW', 'ALL', 'ANY', 'BOTH', 'EACH', 'FEW', 'MORE', 'MOST', 'OTHER
                        'NO', 'NOR', 'NOT', 'ONLY', 'OWN', 'SAME', 'SO', 'THAN', 'TOO', 'V
                        'JUST', 'SHOULD', 'NOW']

    with open(file_path) as f:
        _total_documents = 0
        _md_header = f.readline()
        for line in f:
            cols = line.split(',')
            _master_dictionary[cols[0]] = MasterDictionary(cols, _stopwords)
            _total_documents += _master_dictionary[cols[0]].doc_count
            if len(_master_dictionary) % 5000 == 0 and print_flag:
                print('\r ...Loading Master Dictionary' + ' {}'.format(len(_master_dictio

    if print_flag:
        print('\r', end='')  # clear line
        print('\nMaster Dictionary loaded from file: \n  ' + file_path)
        print('  {0:,} words loaded in master_dictionary.'.format(len(_master_dictionary)

    if f_log:
        try:
            f_log.write('\n\n  load_masterdictionary log:')
            f_log.write('\n    Master Dictionary loaded from file: \n       ' + file_path
            f_log.write('\n    {0:,} words loaded in master_dictionary.\n'.format(len(_ma
        except Exception as e:
            print('Log file in load_masterdictionary is not available for writing')
            print('Error = {0}'.format(e))

    if get_other:
        return _master_dictionary, _md_header, _sentiment_categories, _stopwords, _total_
    else:
        return _master_dictionary


def create_sentimentdictionaries(_master_dictionary, _sentiment_categories):

    _sentiment_dictionary = {}
    for category in _sentiment_categories:
        _sentiment_dictionary[category] = {}
    # Create dictionary of sentiment dictionaries with count set = 0
    for word in _master_dictionary.keys():
        for category in _sentiment_categories:
            if _master_dictionary[word].sentiment[category]:
                _sentiment_dictionary[category][word] = 0

    return _sentiment_dictionary


class MasterDictionary:
    def __init__(self, cols, _stopwords):
        self.word = cols[0].upper()
```

```
        self.word = cols[0].upper()
        self.sequence_number = int(cols[1])
        self.word_count = int(cols[2])
        self.word_proportion = float(cols[3])
        self.average_proportion = float(cols[4])
        self.std_dev_prop = float(cols[5])
        self.doc_count = int(cols[6])
        self.negative = int(cols[7])
        self.positive = int(cols[8])
        self.uncertainty = int(cols[9])
        self.litigious = int(cols[10])
        self.constraining = int(cols[11])
        self.superfluous = int(cols[12])
        self.interesting = int(cols[13])
        self.modal_number = int(cols[14])
        self.strong_modal = False
        if int(cols[14]) == 1:
            self.strong_modal = True
        self.moderate_modal = False
        if int(cols[14]) == 2:
            self.moderate_modal = True
        self.weak_modal = False
        if int(cols[14]) == 3:
            self.weak_modal = True
        self.sentiment = {}
        self.sentiment['negative'] = bool(self.negative)
        self.sentiment['positive'] = bool(self.positive)
        self.sentiment['uncertainty'] = bool(self.uncertainty)
        self.sentiment['litigious'] = bool(self.litigious)
        self.sentiment['constraining'] = bool(self.constraining)
        self.sentiment['strong_modal'] = bool(self.strong_modal)
        self.sentiment['weak_modal'] = bool(self.weak_modal)
        self.irregular_verb = int(cols[15])
        self.harvard_iv = int(cols[16])
        self.syllables = int(cols[17])
        self.source = cols[18]

        if self.word in _stopwords:
            self.stopword = True
        else:
            self.stopword = False
        return



import os
import pandas as pd
import wrds
from datetime import datetime, timedelta
from requests import post as requests_post, get as requests_get
```

```
headers = {
    'Authorization': 'Token caef0703c18abe50deba6e4915d2807bd72eca29',
}

def get_returns_sp500(headers,start):
    filters = f"(caldt__gte={start})"
    params = {
        'limit': 4,
        'filters':filters,
    }
    r = requests_get(
        "https://wrds-api.wharton.upenn.edu/data/crsp.dsp500_v2/",
        headers=headers,
        params=params,
    )
    exc=[]
    for row in r.json()['results']:
        vwretd = row['vwretd']
        if vwretd is not None:
            exc.append(float(vwretd))
        else:
            exc.append(0.0)
    return exc

def get_returns_4(headers,start,cusip):
    filters = f"(date__gte={start}&cusip__exact={cusip})"
    params = {
        'limit': 4,
        'filters':filters,
    }
    r = requests_get(
        "https://wrds-api.wharton.upenn.edu/data/crsp.dsf/",
        headers=headers,
        params=params,
    )
    exc=[]
    for row in r.json()['results']:
        exc.append(float((row['ret'])))
    return exc


def make_cik_cusip_dict(headers):
    headers = {
    'Authorization': 'Token caef0703c18abe50deba6e4915d2807bd72eca29',
    }
    params = {
    'limit': 999999
    }
    r = requests_get(
    "https://wrds-api.wharton.upenn.edu/data/wrdssec.wciklink_cusip/",
    headers=headers,
```

```
    params=params,
    )
    cik_cusip={}
    for row in r.json()['results']:
        cik=row['cik']
        cusip=row['cusip']
        cik_cusip[cik]=cusip
    return cik_cusip

cik_cusip=make_cik_cusip_dict(headers)


def format_date(date_string):
    year = date_string[:4]
    month = date_string[4:6]
    day = date_string[6:]
    return f"{year}-{month}-{day}"

def extract_info_from_filename(filename):
    date_str = filename[:8]
    date_str=format_date(date_str)
    cik = filename.split('edgar_data_')[1].split('_')[0]
    cik= cik.zfill(10)
    return date_str, cik




"""
Program to provide generic parsing for all files in user-specified directory.
The program assumes the input files have been scrubbed,
  i.e., HTML, ASCII-encoded binary, and any other embedded document structures that are n
  intended to be analyzed have been deleted from the file.

Dependencies:
    Python:  Load_MasterDictionary.py
    Data:    LoughranMcDonald_MasterDictionary_XXXX.csv

The program outputs:
   1.  File name
   2.  File size (in bytes)
   3.  Number of words (based on LM_MasterDictionary
   4.  Proportion of positive words (use with care - see LM, JAR 2016)
   5.  Proportion of negative words
   6.  Proportion of uncertainty words
   7.  Proportion of litigious words
   8.  Proportion of modal-weak words
   9.  Proportion of modal-moderate words
  10.  Proportion of modal-strong words
  11.  Proportion of constraining words (see Bodnaruk, Loughran and McDonald, JFQA 2015)
  12.  Number of alphanumeric characters (a-z, A-Z)
```

```
  13.  Number of digits (0-9)
  14.  Number of numbers (collections of digits)
  15.  Average number of syllables
  16.  Average word length
  17.  Vocabulary (see Loughran-McDonald, JF, 2015)

  ND-SRAF
  McDonald 2016/06 : updated 2018/03
"""

import csv
import glob
import re
import string
import sys
import time

# User defined directory for files to be parsed
# TARGET_FILES = r'/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/Files2019/QTR1'
root_dir = "/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez"
# User defined file pointer to LM dictionary
MASTER_DICTIONARY_FILE = '/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/Loughran
# User defined output file
OUTPUT_FILE = r'/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/parser.csv'
# Setup output
OUTPUT_FIELDS = ['file name,', 'file size,', 'number of words,', '% positive,', '% negati
                 '% uncertainty,', '% litigious,', '% modal-weak,', '% modal moderate,',
                 '% modal strong,', '% constraining,', '# of alphabetic,', '# of digits,'
                 '# of numbers,', 'avg # of syllables per word,', 'average word length,',

lm_dictionary = load_masterdictionary(MASTER_DICTIONARY_FILE, True)
harvard = '/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/harvard.csv'
with open(harvard, 'r') as file:
    content = file.read()
harvard_dict = set(content.split())
n_words=[]
n_words_h=[]

for word in lm_dictionary.keys():
    if(lm_dictionary[word].negative):
        n_words.append(word)

for word in list(harvard_dict):
    n_words_h.append(word)


df_neg = pd.DataFrame(columns=n_words)
df_neg_h = pd.DataFrame(columns=n_words_h)
# print(df_neg_h)
ciks = []
```

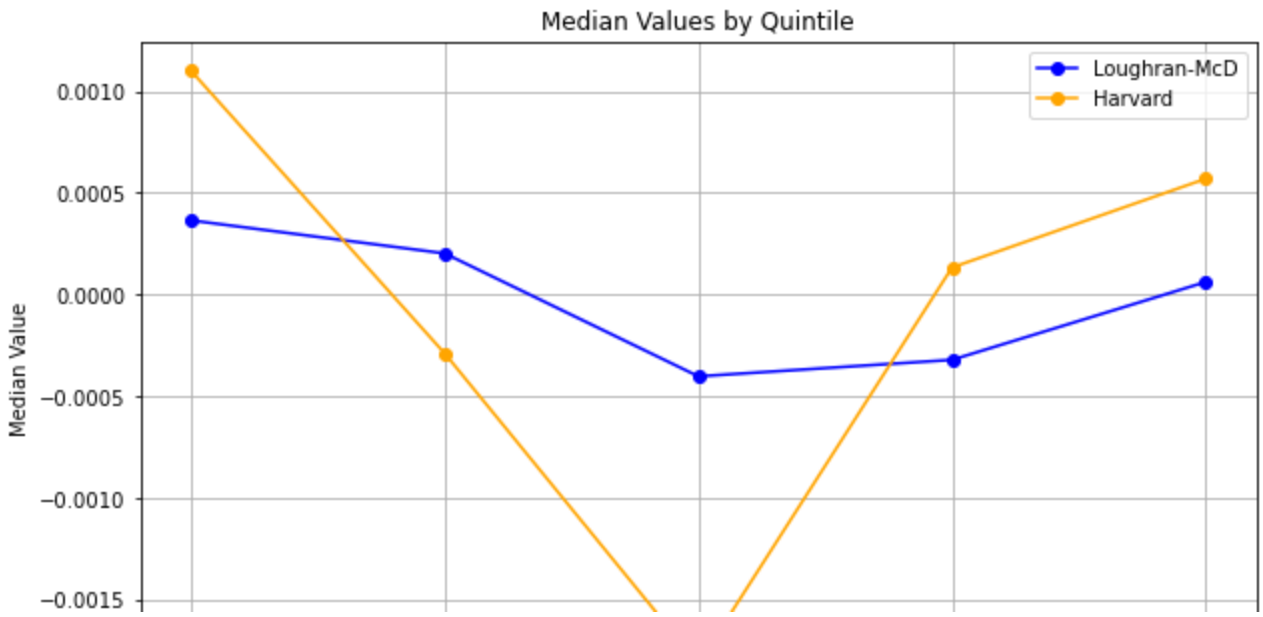
```
exc = []
tot_words=[]
neg_words=[]


def main():
    global cik_cusip
    global cik
    global exc
    global tot_words
    global neg_words

    txt_files = []
    years = range(2019, 2024)
    quarters = ['QTR1', 'QTR2', 'QTR3', 'QTR4']
    for year in years:
        year_dir = os.path.join(root_dir, f"files{year}")
        if os.path.exists(year_dir):
            for quarter in quarters:
                quarter_dir = os.path.join(year_dir, quarter)
                if os.path.exists(quarter_dir):
                    quarter_files = glob.glob(os.path.join(quarter_dir, "*.txt"))
                    txt_files.extend(quarter_files)
    print(f"Total number of .txt files found: {len(txt_files)}")

    for file in txt_files:
        print(file)
        ind=file.rfind('/')
        sub=file[ind+1:]
        start, cik = extract_info_from_filename(sub)
        result=[0.0]*4
        if(cik in cik_cusip):
            ret_stock=get_returns_4(headers,start,cik_cusip[cik])
            ret_sp500=get_returns_sp500(headers,start)
            result = [x - y for x, y in zip(ret_stock, ret_sp500)]
        ciks.append(cik)
        exc.append(result)


        with open(file, 'r', encoding='UTF-8', errors='ignore') as f_in:
            doc = f_in.read()
        doc_len = len(doc)
        doc = re.sub('(May|MAY)', ' ', doc)  # drop all May month references
        doc = doc.upper()  # for this parse caps aren't informative so shift
        output_data = get_data(doc)
        output_data[0] = file
        output_data[1] = doc_len
        tot_words.append(output_data[2])
        neg_words.append(output_data[4])
    df_exret = pd.DataFrame({
    'cik': ciks.
```

```
    'exret':exc,
    'tot_words':tot_words,
    'neg_words':neg_words
    })
    return df_exret


def get_data(doc):
    global df_neg
    global df_neg_h
    vdictionary = {}
    _odata = [0] * 17
    total_syllables = 0
    word_length = 0

    tokens = re.findall('\w+', doc)  # Note that \w+ splits hyphenated words

    word_counts = {word:0 for word in n_words}
    word_counts_h = {word:0 for word in n_words_h}######## NEW

    for token in tokens:
        if not token.isdigit() and len(token) > 1 and token in lm_dictionary:
            _odata[2] += 1  # word count
            word_length += len(token)
            if token not in vdictionary:
                vdictionary[token] = 1
            if lm_dictionary[token].negative:
                _odata[4] += 1
                word_counts[token]+=1 ###NEW
            if token in n_words_h:
                word_counts_h[token]+=1


    df_neg=df_neg.append(word_counts, ignore_index=True)
    df_neg_h=df_neg_h.append(word_counts_h, ignore_index=True)
    ####NEW

    _odata[4]/=_odata[2]*100

    return _odata


if __name__ == '__main__':
    print('\n' + time.strftime('%c') + '\nGeneric_Parser.py\n')
    df_exret=main()
    print('\n' + time.strftime('%c') + '\nNormal termination.')

      ...Loading Master Dictionary 85000
     Master Dictionary loaded from file:
       /Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/LoughranMcDonald_MasterDicti
       86,486 words loaded in master dictionary
```

```
  86,486 words loaded in master_dictionary.


Wed Oct  2 22:07:32 2024
Generic_Parser.py

Total number of .txt files found: 8075
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190130_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190130_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190207_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190206_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190130_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190205_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190206_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190205_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190102_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190204_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190130_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190318_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190301_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190315_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190205_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190228_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190206_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190207_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190130_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190207_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190104_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190206_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190206_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190205_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190305_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190207_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190130_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190208_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190131_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190129_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190315_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190207_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190207_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190204_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190207_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190221_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190201_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190208_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190306_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190207_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190305_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190207_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190131_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190131_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190326_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190307_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190109_10-Q_e
/Users/devanshjoshi/tensorflow-test/NLP- Dan Rodriguez/files2019/QTR1/20190205_10-Q_e
```

```
import math
df_tf_idf=df_neg.copy()
df_i=[0]*df_neg.shape[1]

for i in range(df_tf_idf.shape[0]):#docs
    for j in range(df_tf_idf.shape[1]):## words
        if(df_tf_idf.iloc[i,j]):
            df_tf_idf.iloc[i,j]=(1+math.log(df_tf_idf.iloc[i,j]))/(1+math.log(df_exret['t
            df_i[j]+=1

df_i=[math.log(df_tf_idf.shape[0]/i) if i else i for i in df_i]
df_tf_idf = df_tf_idf.mul(df_i, axis=1)


df_=df_tf_idf.copy()
df_['tfidf']=[sum(df_.iloc[i,:]) for i in range(df_.shape[0])]
df_['ret']=[np.mean(df_exret['exret'][i][1:]) for i in range(df_.shape[0])]
df_ = df_.drop_duplicates(subset='ret')


import math
df_tf_idf_h=df_neg_h.copy()
df_i_h=[0]*df_neg_h.shape[1]

for i in range(df_tf_idf_h.shape[0]):#docs
    for j in range(df_tf_idf_h.shape[1]):## words
        if(df_tf_idf_h.iloc[i,j]):
            df_tf_idf_h.iloc[i,j]=(1+math.log(df_tf_idf_h.iloc[i,j]))/(1+math.log(df_exre
            df_i_h[j]+=1

df_i_h=[math.log(df_tf_idf_h.shape[0]/i) if i else i for i in df_i_h]
df_tf_idf = df_tf_idf_h.mul(df_i_h, axis=1)


df_h=df_tf_idf_h.copy()
df_h['tfidf']=[sum(df_h.iloc[i,:]) for i in range(df_h.shape[0])]
df_h['ret']=[np.mean(df_exret['exret'][i]) for i in range(df_h.shape[0])]
df_h = df_h.drop_duplicates(subset='ret')


# import pandas as pd
# import matplotlib.pyplot as plt

# def quintile_analysis_and_plot(df):
#     df['quintile'] = pd.qcut(df['tfidf'], q=5, labels=False) + 1
#     quintile_medians = df.groupby('quintile')['ret'].median()
#     plt.figure(figsize=(10, 6))
#     plt.plot(quintile_medians.index, quintile_medians.values, marker='o')
#     plt.xlabel('Quintile')
#     plt.ylabel('Median Value')
#     plt.title('Median Values by Quintile')
```

```
#     plt.xticks(range(1, 6))
#     plt.grid(True)
#     plt.show()
#     return df, quintile_medians

# result_df, medians = quintile_analysis_and_plot(df_)

# print("Quintile Medians:")
# print(medians)


import pandas as pd
import matplotlib.pyplot as plt

def quintile_analysis_and_plot2(df1,df2):
    df1['quintile'] = pd.qcut(df1['tfidf'], q=5, labels=False) + 1
    quintile_medians1 = df1.groupby('quintile')['ret'].median()
    df2['quintile'] = pd.qcut(df2['tfidf'], q=5, labels=False) + 1
    quintile_medians2 = df2.groupby('quintile')['ret'].median()
    plt.figure(figsize=(10, 6))
    plt.plot(quintile_medians1.index, quintile_medians1.values, marker='o',color='blue',l
    plt.plot(quintile_medians1.index, quintile_medians2.values, marker='o',color='orange'
    plt.xlabel('Quintile')
    plt.ylabel('Median Value')
    plt.title('Median Values by Quintile')
    plt.xticks(range(1, 6))
    plt.grid(True)
    plt.legend()
    plt.show()

quintile_analysis_and_plot2(df_,df_h)

# print("Quintile Medians:")
# print(medians)
```




1
2
3
4
5
Quintile