```
import pandas as pd
import numpy as np
import math
from os import listdir
from urllib.parse import urlparse, parse_qs
from os.path import splitext
import user_agents
from datetime import datetime
from geolite2 import geolite2
import warnings
warnings.filterwarnings('ignore')

"""
IIS Web log processors
"""


def get_ext(url):
    """
    :param url: url stem from web log
    :return: Return the filename extension from url (ex. .asp, .jpg, .css), if no extension, ''
    """
    parsed = urlparse(url)
    root, ext = splitext(parsed.path)
    return ext.lower()

def user_agent_parse(ua_str):
    """
    using Python user_agents package to parse browser, os, and device of a user agent string
    It will also return "spider"

    :param ua_str: user-agent string from web log
    :return: parsed browser, os, and device
    """
    user_agent = user_agents.parse(ua_str)
    browser = user_agent.browser.family
    os = user_agent.os.family
    device = user_agent.device.family
    return browser, os, device

def add_season(log_data):
    """
    Add "season" field to a data frame based on the existing "month" field

    :param log_data: web log data frame
    :return: data frame with 'season' & 'yearseaon' fields
    """
    season = ['winter'] * 2  # Jan, Feb
    season.extend(['spring'] * 3)  # Mar-May, peak time for traffic
    season.extend(['summer'] * 3)  # Jun-Aug
    season.extend(['fall'] * 3)  # Sep-Nov
    season.extend(['winter'])  # Dec
    season_df = pd.DataFrame(data={'month': list(range(1, 13)), 'season': season})
    log_data = pd.merge(left=log_data, right=season_df, on='month', how='left')
    log_data['yearseason'] = log_data.year.astype(str).str.cat(log_data.season, sep=' ')
    return log_data

def log_parser(file):
    """
    Parse/process IIS log data:
    - header and text cleaning
    - parse query

    :param file: web log file
    :return: parsed/cleaned log data frame
```

```
    """

    # log parser
    log_data = pd.read_csv(file, header=None, sep=' +', skiprows=3)

    col = list(log_data.loc[0,:])[1:]
    to_underscore = ['(', '-', ' ']
    col = [x.lower() for x in col]

    for ch in to_underscore:
        col = [x.lower().replace(ch, '_').replace(')', '') for x in col]

    log_data.drop(log_data.columns[-1], axis=1, inplace=True)  # remove the last column, which
    is all nulls
    log_data.columns = col
    log_data = log_data[~log_data.date.str.startswith('#')].reset_index(drop=True)
    log_data['file'] = file[-10:-4]
    m = int(file[-8:-6])
    log_data['semester'] = 'spring' if m<=4 else 'summer' if m<=8 else'fall'

    # only keep relevent fields
    log_data = log_data[['date', 'time', 'cs_method', 'cs_uri_stem', 'cs_uri_query', 'c_ip',
    'cs_user_agent',
                         'cs_referer', 'sc_status', 'file', 'semester']]

    # trim leading and trailing white space
    for c in log_data.columns:
        log_data[c].str.strip(to_strip=None)

    # unify url with "http://" (replace https://)
    log_data.cs_referer = log_data.cs_referer.str.replace('https://', 'http://')
    log_data.cs_uri_stem = log_data.cs_uri_stem.str.replace('https://', 'http://')

    # removing any non-alphabets/numbers at the end of the strings
    uri_col = ['cs_uri_stem', 'cs_uri_query', 'cs_user_agent', 'cs_referer']
    for c in uri_col:
        log_data[c] = log_data[c].replace("[^a-zA-Z0-9-]$", '', regex=True)

    # transform null to '-', as most of the nulls are "-" in data
    log_data = log_data.fillna('-')
    log_data = log_data.replace('', '-')

    # some queries may not be parsed correctly
    ind=log_data.cs_uri_stem.str.contains('&')
    for i in log_data.cs_uri_stem[ind].index:
        strsplit = log_data.cs_uri_stem[i].split('?')
        if len(strsplit) == 2:
            log_data.cs_uri_stem[i] = strsplit[0]
            log_data.cs_uri_query[i] = strsplit[1]

    # error status
    log_data['request_status'] = 'error'
    log_data.loc[log_data.sc_status.astype('int') <= 399, 'request_status'] = 'success'

    return log_data

def log_datetime(log_data):
    """
    Parse/process IIS log data:
    - timestamp processing

    :param file: web log data frame
    :return: processed log data frame
    """
    # UTC time
```

```
    log_data['datetime_utc'] = pd.DatetimeIndex(
            pd.to_datetime(log_data.date + ' ' + log_data.time, format='%Y-%m-%d %H:%M:%S',
            errors='coerce', utc=True)).\
            tz_localize('UTC')

    # Eastern time
    # From data exploration: 80%+ visitors are from US, among which 80%+ are from EST zone
    log_data['datetime_est'] = pd.DatetimeIndex(log_data.datetime_utc).tz_convert('US/Eastern')
    # if use tz="EST", daytime saving adj will not be considered (-5 instead of -4 in summer)

    log_data['yearmonth'] = log_data.datetime_utc.map(lambda x: x.strftime('%Y-%m'))   #use utc
    time to be consistant with semester
    log_data['date'] = log_data.datetime_utc.dt.date
    log_data['year'] = log_data.datetime_utc.dt.year
    log_data['month'] = log_data.datetime_utc.dt.month
    log_data['day'] = log_data.datetime_utc.dt.day
    log_data['time_est'] = log_data.datetime_est.dt.time
    log_data['hour_est'] = log_data.datetime_est.dt.hour
    log_data = add_season(log_data)

    return log_data

def log_page_type(log_data):
    """
    Parse/process IIS log data:
    - get url extension, and identify page/non-page/pdf download

    :param file: web log data frame
    :return: processed log data frame
    """
    # Identify URL extension and page/non page
    resources = ['.jpg', '.jpeg', '.gif', '.png', '.wav', '.wma', '.wmv', '.svg',\
        '.js', '.css', '.swf', '.eot', '.svg', 'woff', '.ttf', '.cgi']
    resources = '(\\' + '|\\'.join(resources) + ')$'

    log_data['cs_uri_ext'] = log_data.cs_uri_stem.apply(lambda x: get_ext(x))

    # some pdf pages need cleaning
    ind = (log_data.cs_uri_stem.str.contains('.pdf')) & (log_data.cs_uri_ext != '.pdf')
    log_data.loc[ind, 'cs_uri_query'] = log_data.loc[ind, 'cs_uri_stem'].replace('.*\.pdf', '',
    regex=True)
    log_data.loc[ind, 'cs_uri_stem'] = log_data.loc[ind, 'cs_uri_stem'].replace('\.pdf.*$',
    '.pdf', regex=True)
    log_data.loc[ind, 'cs_uri_ext'] = '.pdf'

    # uri_type
    log_data.loc[log_data.cs_uri_stem.str.contains('/assets/'), 'cs_uri_type'] = 'non_page'
    log_data.loc[(log_data.cs_uri_type.isnull()) & (log_data.cs_uri_ext.str.contains(resources,
    regex=True)), 'cs_uri_type'] = 'non_page'
    log_data.loc[log_data.cs_uri_ext == '.pdf', 'cs_uri_type'] = 'download_pdf'
    log_data.loc[log_data.cs_uri_type.isnull(), 'cs_uri_type'] = 'page'

    return log_data

def log_ua_spider(log_data):
    """
    Parse/process IIS log data:
    - parse user agent
    - identify bots/spiders

    :param file: web log data frame
    :return: processed log data frame
    """

    # Split User Agent and Identify Spider
```

```
    log_data['browser'], log_data['os'], log_data['device'] =
    zip(*log_data['cs_user_agent'].map(user_agent_parse))
    log_data['spider'] = log_data.device == 'Spider'
    bot_ip_known = list(log_data[log_data.spider].c_ip.unique())
    log_data.loc[log_data.c_ip.isin(bot_ip_known), 'spider'] = True
    log_data['spider_known'] = log_data.spider

    bot_token = ['Mediapartners-Google', 'curl', 'daum', 'gigablastopensource',
    'go-http-client', 'httpclient',
          'libwww-perl', 'phantomjs', 'proxy', 'python', 'sitesucker', 'wada.vn', 'webindex',
          'wget',
          '\\.com', 'http:', 'https:', '8LEGS', 'Nuzzel', 'matlab']
    bot_token = '|'.join(bot_token)
    bot_ua = ['-', 'default', 'null', 'Agent']

    for i in bot_ua:
        log_data.loc[log_data.cs_user_agent == i, 'spider'] = True

    log_data.loc[log_data.cs_user_agent.str.contains(bot_token, regex=True, case=False),
    'spider'] = True
    log_data.loc[~log_data.cs_method.str.contains('GET|POST', regex=True), 'spider'] = True

    bot_ip = list(log_data[log_data.spider].c_ip.unique())
    log_data.loc[log_data.c_ip.isin(bot_ip), 'spider'] = True

    return log_data

def log_used_data(log_data):
    """
    Parse/process IIS log data:
    - select data to exclude bots and non-page

    :param file: web log file
    :return: selected data frame: exclude spider/non-page
    """
    # Selected Data Set
    log_data_use = log_data[(~log_data.spider) & (log_data.cs_uri_type !=
    'non_page')].copy().reset_index(drop=True)
    log_data_use = log_data_use.sort_values(['datetime_utc'],
    ascending=True).reset_index(drop=True)
    log_data_use['idx'] = log_data_use.file + '_' + log_data_use.index.map(str)

    return log_data_use

def ip_processor(ip_schema):
    """
    Parse geolite2 location schema
    Return a dictionary
    """

    coll = {}
    coll['c_ip'] = ip_schema['c_ip']

    # continent
    n = 'continent'
    if n in ip_schema.keys():
        k = 'code'
        coll[n+'_'+k] = ip_schema[n][k] if k in ip_schema[n].keys() else None
        k = 'geoname_id'
        coll[n+'_'+k] = ip_schema[n][k] if k in ip_schema[n].keys() else None
        k = 'names'
        coll[n+'_'+k] = ip_schema[n][k]['en'] if k in ip_schema[n].keys() else None

    # country
```

```
    n = 'country'
    if n in ip_schema.keys():
        k = 'iso_code'
        coll[n+'_'+k] = ip_schema[n][k] if k in ip_schema[n].keys() else None
        k = 'geoname_id'
        coll[n+'_'+k] = ip_schema[n][k] if k in ip_schema[n].keys() else None
        k = 'names'
        coll[n+'_'+k] = ip_schema[n][k]['en'] if k in ip_schema[n].keys() else None

    # city
    n = 'city'
    if n in ip_schema.keys():
        k = 'geoname_id'
        coll[n+'_'+k] = ip_schema[n][k] if k in ip_schema[n].keys() else None
        k = 'names'
        coll[n+'_'+k] = ip_schema[n][k]['en'] if k in ip_schema[n].keys() else None

    # subdivisions
    n = 'subdivisions'
    if n in ip_schema.keys():
        k = 'iso_code'
        coll[n+'_'+k] = ip_schema[n][0][k] if k in ip_schema[n][0].keys() else None
        k = 'geoname_id'
        coll[n+'_'+k] = ip_schema[n][0][k] if k in ip_schema[n][0].keys() else None
        k = 'names'
        coll[n+'_'+k] = ip_schema[n][0][k]['en'] if k in ip_schema[n][0].keys() else None

    # postal
    n = 'postal'
    if n in ip_schema.keys():
        k = 'code'
        coll[n+'_'+k] = ip_schema[n][k] if k in ip_schema[n].keys() else None

    # location
    n = 'location'
    if n in ip_schema.keys():
        k = 'latitude'
        coll[n+'_'+k] = ip_schema[n][k] if k in ip_schema[n].keys() else None
        k = 'longitude'
        coll[n+'_'+k] = ip_schema[n][k] if k in ip_schema[n].keys() else None
        k = 'time_zone'
        coll[n+'_'+k] = ip_schema[n][k] if k in ip_schema[n].keys() else None
        k = 'metro_code'
        coll[n+'_'+k] = ip_schema[n][k] if k in ip_schema[n].keys() else None

    return coll
```