```
#!/usr/bin/env python
# coding: utf-8
# download_rockland_raw_bids_ver2.py
#
# Modified by: Yiwen Tian 2021
# Based on Ver1 by: Daniel Clark, John Pellman 2015/2016, Carlos G.Candano 2017/2018


'''
This script downloads data from the NKI Rockland Public releases
stored in the cloud in BIDS format.  You can specify sex, age range, handedness,
session, scan type (anatomical, functional, dwi) and to limit your download to a subset of
the sample.
If no options are specified, all available files are downloaded.
Use the '-h' to get more information about command line usage.
'''


SESSIONS = ['BAS1','BAS2','FLU1','TRT','FLU2']


SCANS = ['anat', 'func', 'dwi']


SERIES_MAP={'CHECKERBOARD1400':'task-CHECKERBOARD_acq-1400',
            'CHECKERBOARD645':'task-CHECKERBOARD_acq-645',
            'RESTCAP':'task-rest_acq-CAP',
            'REST1400':'task-rest_acq-1400',
            'BREATHHOLD1400':'task-BREATHHOLD_acq-1400',
            'REST645':'task-rest_acq-645',
            'RESTPCASL':'task-rest_pCASL'}


def files(client, bucket, prefix=''):
    """ Return the path to the participants.tsv file in the bucket """
    paginator = client.get_paginator('list_objects')
    for result in paginator.paginate(Bucket=bucket, Prefix=prefix,
Delimiter='participants.tsv'):
        for prefix in result.get('CommonPrefixes', []):
            yield prefix.get('Prefix')


def generate_Subfolders(s3_client):
    """ Generates a list of the contents specified in the prefix """
    gen_subfolders = files(s3_client, 'fcp-indi', prefix=
('data/Projects/RocklandSample/RawDataBIDSLatest/'))#+extra
    genSubfoldersList=list(gen_subfolders)
    print(genSubfoldersList)
    return genSubfoldersList


# Main collect and download function
def collect_and_download(out_dir,
                         aws_links=0,
                         less_than=0, greater_than=0, sex='', handedness='',
                         sessions=SESSIONS,
                         scans=SCANS,
                         series=SERIES_MAP.keys(),
                         derivatives=False,
                         dryrun=False):
    '''
    Function to collect and download images from the Rockland sample
    directory on FCP-INDI's S3 bucket
    Parameters
```

```
    ----------
    out_dir : string
        filepath to a local directory to save files to
    aws_links : string
        filepath of aws_links.csv from
http://fcon_1000.projects.nitrc.org/indi/enhanced/aws_links.csv
    less_than : float
        upper age (years) threshold for participants of interest
    greater_than : float
        lower age (years) threshold for participants of interest
    sex : string
        'M' or 'F' to indicate whether to download male or female data
    handedness : string
        'R' or 'L' to indicate whether to download right-handed or
        left-handed participants
    sessions : list
        the session names (e.g.,'BAS1','FLU1')
    scan : list
        the scan types to download.  Can be 'anat','func' or 'dwi'.
    series : list
        the series to download (for functional scans)
    derivatives : boolean
        whether or not to download data derivatives for functional scans
    dryrun : boolean
        whether or not to perform a dry run (i.e., no actual downloads,
        just listing files that would be downloaded)
    Returns
    -------
    boolean
        Returns true if the download was successful, false otherwise.
    '''
    # Import packages
    import pandas
    import boto3
    import botocore
    # For anonymous access to the bucket.
    from botocore import UNSIGNED
    from botocore.client import Config
    from botocore.handlers import disable_signing

    # Init variables
    s3_bucket_name = 'fcp-indi'
    s3_prefix = 'data/Projects/RocklandSample/RawDataBIDSLatest'

    # Fetch bucket
    s3 = boto3.resource('s3')
    s3.meta.client.meta.events.register('choose-signer.s3.*', disable_signing)
    s3_bucket = s3.Bucket(s3_bucket_name)

    # Remove series that aren't in the series map keys.
    series = [ s for s in series if s in SERIES_MAP.keys()]

    # Generate a list of series to filter on. (e.g. task-CHECKERBOARD..)
    series_filt = [SERIES_MAP[s] for s in series]
    # If output path doesn't exist, create it
    if not os.path.exists(out_dir) and not dryrun:
        print ('Could not find %s, creating now...' % out_dir)
        os.makedirs(out_dir)
    if os.path.exists(aws_links):
        participants_df = pandas.read_csv(aws_links, na_values=['n/a'])
    else:
        print('Downloading aws_links.csv from
http://fcon_1000.projects.nitrc.org/indi/enhanced......')
        os.system('wget http://fcon_1000.projects.nitrc.org/indi/enhanced/aws_links.csv .')
        print('Saved to current folder')
```

```
        participants_df = pandas.read_csv('aws_links.csv', na_values=['n/a'])

    s3_client = boto3.client('s3', config=Config(signature_version=UNSIGNED))
    print ('Collecting images of interest...')
    # Remove the participant rows from whose age range, handedness and sex do not conform to
the criteria.
    if less_than:
        participants_df = participants_df[participants_df['age'] < less_than]
    if greater_than:
        participants_df = participants_df[participants_df['age'] > greater_than]
    if sex == 'M':
        participants_df = participants_df[participants_df['gender'] == 'M']
    elif sex == 'F':
        participants_df = participants_df[participants_df['gender'] == 'F']
    if handedness == 'R':
        participants_df = participants_df[participants_df['handedness'] == 'RIGHT']
    elif handedness == 'L':
        participants_df = participants_df[participants_df['handedness'] == 'LEFT']
    participants_df = participants_df[participants_df['session'].isin(sessions)]
    participants_df =
participants_df[participants_df['filepath'].str.contains('|'.join(scans))]
    participants_df =
participants_df[(participants_df['filepath'].str.contains('|'.join(series_filt))==True)|
(participants_df['filepath'].str.contains('func')==False)]
    if len(participants_df) == 0:
        print ('No participants meet the criteria given.  No download will be initiated.')
        return

    # Generate a list of participants, single column list to filter on.
    participants_filt = ['sub-' + label + '/' for label in
participants_df['subject'].tolist()]
    participants_filt = list(set(participants_filt))

    s3_keylist = list(participants_df['filepath'])
    s3_keylist = list(set(s3_keylist))

    # Append as well the dataset_desc json path to the end of s3_keylist
    s3_keylist.append('/'.join(['s3://fcp-indi', s3_prefix, 'dataset_description.json']))
    # Verify that the participants Datframe Has Only the Subjects that appear in the
s3_keylist
    # NOT all participants have all the Scan Types.
    newParticipantsDf=pandas.DataFrame(columns=participants_df.columns)
    for row in s3_keylist:
        rowDivided=row.split('/')
        #Get only the rows with sub-A000
        justTheSub=[x for x in rowDivided if x.startswith("sub-A") and len(x)==13]
        if len(justTheSub)>0:
            # Remove the sub- part
            ursi=justTheSub[0][4:]
            particRow=participants_df[participants_df.iloc[:,0].str.contains(ursi)]
            #Build the dataframe and remove repeated rows
            newParticipantsDf=pandas.DataFrame.append(newParticipantsDf,particRow)
            newParticipantsDf=pandas.DataFrame.drop_duplicates(newParticipantsDf)
    participants_df = newParticipantsDf

    # Re-create the participants list after verifying the particpants that have
    #The correct scan types. This corss checking with s3_keylist
    participants_filt = ['sub-' + label + '/' for label in
participants_df['subject'].tolist()]
    # And download the items. All the items are the Total number of rows in s3_keylist
    total_num_files = len(s3_keylist)
    files_downloaded = len(s3_keylist)
    # For each of the paths and each index in s3_keylist
    for path_idx, s3_path in enumerate(s3_keylist):
        s3_path = s3_path.replace('s3://fcp-indi/','')
```

```
        print (s3_path)
        # Remove the string /data/Projects/RocklandSample/RawDataBIDSLatest for each path in
list
        rel_path = s3_path.replace(s3_prefix, '')
        # Remove the FIRST slash from string
        rel_path = rel_path.lstrip('/')
        # Create a location path  for the folder and file path
        download_file = os.path.join(out_dir, rel_path)
        download_dir = os.path.dirname(download_file)
        # Create the folder in the path specified
        if not os.path.exists(download_dir) and not dryrun:
            os.makedirs(download_dir)
        try:
            if not os.path.exists(download_file):
                # Dryrun will not download the files
                if dryrun:
                    print ('Would download to: %s' % download_file)
                else:
                    print ('Downloading to: %s' % download_file)
                    # Download the files in the just created folder
                    with open(download_file, 'wb') as f:
                        s3_client.download_fileobj(s3_bucket_name, s3_path, f)
                    print ('%.3f%% percent complete' %                              (100*
(float(path_idx+1)/total_num_files)))
            else:
                print ('File {} already exists, skipping...'.format(download_file))
                files_downloaded -= 1
        except Exception as exc:
            print ('There was a problem downloading %s.\n'                    'Check input
arguments and try again.' % s3_path)
            print (exc)

    if dryrun:
        print ('%d files would be downloaded for %d participant(s).' %
(files_downloaded,len(participants_df)))
    else:
        print ('%d files downloaded for %d participant(s).' %
(files_downloaded,len(participants_df)))
    if not dryrun:
        print ('Saving out revised participants.tsv and session tsv files.')
        # Save out revised participants.tsv to output directory, if a participants.tsv
already exists, open it and append it to the new one.
        if os.path.isfile(os.path.join(out_dir, 'participants.tsv')):
            old_participants_df = pandas.read_csv(os.path.join(out_dir, 'participants.tsv'),
delimiter='\t', na_values=['n/a', 'N/A'])
            participants_df = participants_df.append(old_participants_df, ignore_index=True)
            participants_df.drop_duplicates(inplace=True)
            os.remove(os.path.join(out_dir, 'participants.tsv'))
        participants_df.to_csv(os.path.join(out_dir, 'participants.tsv'), sep="\t",
na_rep="n/a", index=False)
    print ('Done!')


# Make module executable
if __name__ == '__main__':
    # Import packages
    import argparse
    import sys
    import os

    # Init arparser
    parser = argparse.ArgumentParser(description=__doc__)

    # Required arguments
    parser.add_argument('-o', '--out_dir', required=True, type=str,
```

```
                        help='Path to local folder to download files to')

    # Optional arguments
    parser.add_argument('-al', '--aws_links', required=False,
                        type=str, help='Path to aws_links.csv. Leave it empty '\
                         'to let the script search and download to current directory')
    parser.add_argument('-lt', '--less_than', required=False,
                         type=float, help='Upper age threshold (in years) of '\
                                          'particpants to download (e.g. for '\
                                          'subjects 30 or younger, \'-lt 31\')')
    parser.add_argument('-gt', '--greater_than', required=False,
                         type=float, help='Lower age threshold (in years) of '\
                                       'particpants to download (e.g. for '\
                                       'subjects 31 or older, \'-gt 30\')')
    parser.add_argument('-x', '--sex', required=False, type=str,
                         help='Participant sex of interest to download only '\
                              '(e.g. \'M\' or \'F\')')
    parser.add_argument('-m', '--handedness', required=False, type=str,
                         help='Participant handedness to download only '\
                              '(e.g. \'R\' or \'L\')')
    parser.add_argument('-v', '--sessions', required=False, nargs='*', type=str,
                         help='A space-separated list of session (visit) codes '\
                              'to download (e.g. \'NFB3\',\'CLG2\')')
    parser.add_argument('-t', '--scans', required=False, nargs='*', type=str,
                         help='A space-separated list of scan types '\
                              'to download (e.g. \'anat\',\'dwi\')')
    parser.add_argument('-e', '--series', required=False, nargs='*', type=str,
                         help='A space-separated list of series codes '\
                              'to download (e.g. \'DMNTRACKINGTRAIN\',\'DMNTRACKINGTEST\')')
    parser.add_argument('-d', '--derivatives', required=False, action='store_true',
                         help='Download derivatives (despiked physio, masks) in addition to
raw data?')
    parser.add_argument('-n', '--dryrun', required=False, action='store_true',
                         help='Perform a dry run to see how many files would be downloaded.')

    # Parse and gather arguments
    args = parser.parse_args()
    #########
    '''
    class Namespace:
        def __init__(self, **kwargs):
            self.__dict__.update(kwargs)
    args=Namespace(out_dir='/home/cgutierrez/RunScripts/down_rock', less_than=7,
greater_than=None,sex='M', handedness=None,sessions=None, scans=['func'], series=
["CHECKERBOARD645","REST1400"], dryrun=False, derivatives=None)
    ##########
    '''
    # Init variables
    out_dir = os.path.abspath(args.out_dir)
    kwargs = {}
    if args.aws_links:
        kwargs['aws_links'] = args.aws_links
    elif os.path.exists('aws_links.csv'):
        print ('Found aws_links.csv in current working directory.')
        kwargs['aws_links'] = 'aws_links.csv'

    if args.less_than:
        kwargs['less_than'] = args.less_than
        print ('Using upper age threshold of %d...' % kwargs['less_than'])
    else:
        print ('No upper age threshold specified')
    if args.greater_than:
        kwargs['greater_than'] = args.greater_than
        print ('Using lower age threshold of %d...' % kwargs['greater_than'])
    else:
```

```
        print ('No lower age threshold specified')
    if args.sex:
        kwargs['sex'] = args.sex.upper()
        if kwargs['sex'] == 'M':
            print ('Downloading only male participants...')
        elif kwargs['sex'] == 'F':
            print ('Downloading only female participants...')
        else:
            print ('Input for sex \'%s\' was not \'M\' or \'F\'.' % kwargs['sex'])
            print ('Please check script syntax and try again.')
            sys.exit(1)
    else:
        print ('No sex specified, using all sexes...')
    if args.handedness:
        kwargs['handedness'] = args.handedness.upper()
        if kwargs['handedness'] == 'R':
            print ('Downloading only right-handed participants...')
        elif kwargs['handedness'] == 'L':
            print ('Downloading only left-handed participants...')
        else:
            print ('Input for handedness \'%s\' was not \'L\' or \'R\'.' %
kwargs['handedness'])
            print ('Please check script syntax and try again.')
            sys.exit(1)
    if args.sessions:
        kwargs['sessions'] = args.sessions
        for session in kwargs['sessions']:
            if session not in SESSIONS:
                print ('Session \'%s\' is not a valid session name.' % session)
                print ('Please check script syntax and try again.')
                sys.exit(1)
        print ('Sessions to download: ' + ' '.join(kwargs['sessions']))
    if args.scans:
        kwargs['scans'] = args.scans
        for scan in kwargs['scans']:
            if scan not in SCANS:
                print ('Scan \'%s\' is not a valid scan name.' % scan)
                print ('Please check script syntax and try again.')
                sys.exit(1)
        print ('Scans to download: ' + ' '.join(kwargs['scans']))
    if args.series:
        kwargs['series'] = args.series
        for series in kwargs['series']:
            if series not in SERIES_MAP.keys():
                print ('Series \'%s\' is not a valid series name.' % series)
                print ('Please check script syntax and try again.')
                sys.exit(1)
        print ('Series to download: ' + ' '.join(kwargs['series']))
    if args.derivatives:
        kwargs['derivatives'] = args.derivatives
        print ('Data derivatives will be downloaded.')
    if args.dryrun:
        kwargs['dryrun'] = args.dryrun
        print ('Running download as a dry run.')

    # Call the collect and download routine
    collect_and_download(out_dir, **kwargs)
```