```
#!/usr/bin/env python2.7
# Copyright (C) 2016 Daniel Asarnow
# University of California, San Francisco
#
# Simple program for parsing and altering Relion .star files.
# See help text and README file for more information.
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/
>.
from __future__ import print_function
import argparse
import sys
import json
import numpy as np
import pandas as pd
from pyem.star import write_star
from pyem.star import transform_star


general = {u'uid': None,
           u'ctf_params.akv': "rlnVoltage",
           u'ctf_params.angast_deg': "rlnDefocusAngle",
           u'ctf_params.angast_rad': None,
           u'ctf_params.cs': "rlnSphericalAberration",
           u'ctf_params.detector_psize': "rlnDetectorPixelSize",
           u'ctf_params.df1': "rlnDefocusU",
           u'ctf_params.df2': "rlnDefocusV",
           u'ctf_params.mag': "rlnMagnification",
           u'ctf_params.psize': None,
           u'ctf_params.wgh': "rlnAmplitudeContrast",
           u'data_input_relpath': "rlnImageName",
           u'data_input_idx': None}

model = { u'alignments.model.U': None,
          u'alignments.model.dr': None,
          u'alignments.model.dt': None,
          u'alignments.model.ess_R': None,
          u'alignments.model.ess_S': None,
          u'alignments.model.phiC': None,
```

```
          u'alignments.model.r.0': "rlnAngleRot",
          u'alignments.model.r.1': "rlnAngleTilt",
          u'alignments.model.r.2': "rlnAnglePsi",
          u'alignments.model.t.0': "rlnOriginX",
          u'alignments.model.t.1': "rlnOriginY"}


def main(args):
    meta = parse_metadata(args.input)  # Read cryosparc metadata file.
    meta["data_input_idx"] = ["%.6d" % (i+1) for i in
meta["data_input_idx"]]  # Reformat particle idx for Relion.

    if "data_input_relpath" not in meta.columns:
        if args.data_path is None:
            print("Data path missing, use --data-path to specify
particle stack path")
            return 1
        meta["data_input_relpath"] = args.data_path

    meta["data_input_relpath"] =
meta["data_input_idx"].str.cat(meta["data_input_relpath"], sep="@")  #
Construct _rlnImageName field.
    # Take care of trivial mappings.
    rlnheaders = [general[h] for h in meta.columns if h in general and
general[h] is not None]
    star = meta[[h for h in meta.columns if h in general and
general[h] is not None]].copy()
    star.columns = rlnheaders

    # general class assignments and other model parameters.
    phic = meta[[h for h in meta.columns if "phiC" in h]]  # Posterior
probability over class assignments.
    if len(phic.columns) > 1:  # Check class assignments exist in
input.
        # phic.columns = [int(h[21]) for h in meta.columns if "phiC"
in h]
        phic.columns = range(len(phic.columns))
        cls = phic.idxmax(axis=1)
        for p in model:
            if model[p] is not None:
                pspec = p.split("model")[1]
                param = meta[[h for h in meta.columns if pspec in h]]
                param.columns = phic.columns
                star[model[p]] = param.lookup(param.index, cls)
        star["rlnClassNumber"] = cls + 1  # Compute most probable
classes and add one for Relion indexing.
    else:
        for p in model:
            if model[p] is not None and p in meta.columns:
                star[model[p]] = meta[p]
```

```
        star["rlnClassNumber"] = 1

    if args.minphic is not None:
        mask = np.all(phic < args.minphic, axis=1)
        if args.drop_bad:
           star.drop(star[mask].index, inplace=True)  # Delete low-
confidence particles.
        else:
           star.loc[mask, "rlnClassNumber"] = 0  # Set low-confidence
particles to dummy class.

    if args.transform is not None:
        r = np.array(json.loads(args.transform))
        star = transform_star(star, r, inplace=True)

    # Write Relion .star file with correct headers.
    write_star(args.output, star, reindex=True)
    return 0


def parse_metadata(csvfile):
    with open(csvfile, 'rU') as f:
        lines = enumerate(f)
        idx = -1
        headers = None
        for i, l in lines:
            if l.startswith("_header"):
                _, headers = next(lines)
                headers = headers.rstrip().split(",")
            if l.startswith("_dtypes"):
                _i, dtypes = next(lines)
                dtypes = dtypes.rstrip().split(",")
                idx = _i + 1
                break
    meta = pd.read_csv(csvfile, skiprows=idx, header=None,
skip_blank_lines=True)
    if headers is None:
        return None
    meta.columns = headers
    return meta


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("input", help="Input Cryosparc metadata .csv
file")
    parser.add_argument("output", help="Output .star file")
    parser.add_argument("--minphic", help="Minimum posterior
probability for class assignment", type=float)
    parser.add_argument("--drop-bad", help="Drop particles instead of
```

```
assigning dummy class", action="store_true")
    parser.add_argument("--data-path", help="Path to single particle
stack", type=str)
    parser.add_argument("--transform", help="Apply rotation matrix or
3x4 rotation plus translation matrix to particles (Numpy format)",
type=str)
    sys.exit(main(parser.parse_args()))
```