```
# Name: Dustin Burnham
# UW NetID: dusty736
# Section: AB
# CSE 160
# Homework 7: Final Project

import json
from matplotlib import pyplot as plt


######################################################################
##########
# Problem 1: Read and clean Analyzed Quasar Spectrum
######################################################################
##########


def read_spectrum_data(filename):
    """Read in a .json file using the json package.

    Parameters:
        filename: This is the filename of the analyzed QSO spectrum

    Returns:
        The data from the json file.
    """

    with open(filename) as data_file:
        data = json.load(data_file)
    return data


def spectrum_to_list(spectrum_dict):
    """!!! This function (spectrum_to_lst) was provided by my research
group.
    I did not write it !!!

    The data read in from the json file is turned into a list with
    each entry containing all the data for one element found.

    Parameters:
        spectrum_dict: The data from the json file.

    Returns:
        A list where each entry is a element with all of their
respective data
        separated by a ';'.
    """

    systems=['z0.00000_MW']
```

```
    components=[]


    for cmp in spectrum_dict["cmps"]:
        cmp_dict=spectrum_dict["cmps"][str(cmp)]
        cmp_data=''
        if cmp[-2:]=="HI":
            systems.append(str(cmp))
        cmp_data+=str(cmp)+';'
        cmp_data+=str(cmp_dict["Reliability"])+';'
        if cmp_dict["Comment"]=='':
            cmp_data+='None;'
        else:
            cmp_data+=str(cmp_dict["Comment"])+';'
        cmp_data+=str(cmp_dict["Nfit"])+';'
        cmp_data+=str(cmp_dict["bfit"])+';'
        components.append(cmp_data)
    return components


def system_dict(spectrum_lst):
    """ Scrubs the list making the list readable.
    Maps each redshift to elements with the same redshift.

    Parameters:
        spectrum_lst: The list of data from the analyzed spectrum.

    Returns:
        A dictionary of dictionaries is returned.  The key of the
outter
        dictionary is a redshift, and it it s mapped to dictionary
containing
        that redshifts comment, column density, doppler profile,
classification,
        and element name.
    """

    system_z = {}

    for system in spectrum_lst:
        system_data = {}
        system = system.split(';')
        ID = system[0].replace('z', '')
        ID = ID.replace('-', '')
        ID = ID.split('_')
        redshift = float(ID[0])
        element = ID[1]
        system_data['element'] = element
        system_data['Classification'] = system[1]
        system_data['Comment'] = system[2]
```

```
        system_data['Column Density'] = float(system[3])
        system_data['Doppler Profile'] = float(system[4])
        system_z[redshift] = system_data
    return system_z


def element_dict(z_dict):
    """ Creates a dictionary of each element ound at a certain
redshift.

    Parameters:
        z_dict: A dictionary of redshifts mapped to dictionaries of
data
        which include element name.

    Returns:
        A dictionary of redshifts mapped to a list of elements found
at that
        redshift.
    """

    elt_dict = {}
    z_lst = []
    for z in z_dict:
        z_lst.append(z)

    for z in z_lst:
        z_up = z + 0.002
        z_down = z - 0.002
        elt_lst = []
        for z_elt in z_dict:
            if z_elt > z_down and z_elt < z_up:
                elt_lst.append(z_dict[z_elt]['element'])
        elt_dict[z] = elt_lst
    return elt_dict

######################################################################
##########
# Problem 2: Determine where the Galaxies are located
######################################################################
##########


def Hydrogen_z(z_dict):
    """ Makes a list of all the redshifts that contain Hydrogen.  This
    is used for plotting later.

    Parameters:
        z_dict: A dictionary of redshifts mapped to dictionaries of
data
```

```
        which include element name.

    Returns:
        A list of all of the redshifts where Hydrogen is found.
    """

    z_lst = []

    for z in z_dict:
        if z_dict[z]['element'] == 'HI':
            z_lst.append(z)
    return z_lst


def Hydrogen_Nc(z_dict):
    """ Makes a list of all column densities of Hydrogen.  This is
used for
    plotting later.

    Parameters:
        z_dict:  A dictionary of redshifts mapped to dictionaries of
data
        which include element name and column density.

    Returns:
        A list of all Hydrogen column densities is returned.
    """

    Nc_lst = []

    for z in z_dict:
        if z_dict[z]['element'] == 'HI':
            Nc_lst.append(z_dict[z]['Column Density'])
    return Nc_lst

def Hydrogen_sys(z_dict):
    """ Systems with hydrogen at a high column density are the systems
of
    gas that are likely connected a galaxy.  This function creates a
dictionary
    mapping redshift to column denisties greater than 14.

    Parameters:
        z_dict:  A dictionary of redshifts mapped to dictionaries of
data
        which include element name and column density.

    Returns:
        This function creates a dictionary
        mapping redshift to column denisties greater than 14.
```

```
    """

    H_dict = {}

    for z in z_dict:
        if z_dict[z]['element'] == 'HI':
            if z_dict[z]['Column Density'] > 14.0:
                H_dict[z] = z_dict[z]['Column Density']
                #print "Search for candidate galaxies at redshift:" +
str(z)
    return H_dict


######################################################################
##########
# Problem 3: Read and clean Galaxy list data from SDSS
######################################################################
##########


def Read_in_Galaxies(Galaxy_filename):
    """ Read in a text file of galaxies near the QSO and saves the
data into a
    list.  Contains Galaxy ID
    Right Ascension, Declination, redshift, and redshift uncertainty.

    Parameters:
        Galaxy_filename: Text file containing Galaxy ID (objID),
        Right Ascension (ra), Declination (dec), redshift (z), and
        redshift uncertainty (zErr).

    Returns:
        A list of each row in the text file returned.
    """

    Gal_lst = []
    Obj_data = open(Galaxy_filename)
    for Galaxy in Obj_data:
        Gal_lst.append(Galaxy)
    Obj_data.close()
    return Gal_lst


def Galaxies_dict(Galaxy_lst):
    """ This function cleans up unwanted text from the data list, and
    turns the list into a list of dictionaries of the Galaxy data.

    Parameters:
        Galaxy_lst: The list of galaxy data from the SDSS database.
```

```
    Returns:
        Returns a list of dictionaries where each the ID, ra, dec,
        z, and zErr are mapped to their respective quantities.
    """

    data = []
    param_name = Galaxy_lst[0].split("\t")
    num_par = len(param_name) - 1
    param_name[num_par] = param_name[num_par].replace('\n', "")
    for Obj in range(1,len(Galaxy_lst)):
        Gal_dict = {}
        Galaxy = Galaxy_lst[Obj].split("\t")
        Galaxy[num_par] = Galaxy[num_par].replace('\n', "")
        for param_num in range(num_par + 1):
            if param_name[param_num] != 'objID':
                Gal_dict[param_name[param_num]] =
float(Galaxy[param_num])
            else:
                Gal_dict[param_name[param_num]] = Galaxy[param_num]
        Redshift_range(Gal_dict)
        data.append(Gal_dict)
    return data


def Galaxy_Candidates(H_dict, dict_lst):
    """ Compares the dictionary of high column density hydrogen with
the
    dictionaries in the list of galaxies from the SDSS, and matches
    the systems of hydrogen with redshifts that fall within the
uncertainty
    of the galaxies from the SDSS.

    Parameters:
        H_dict: Dictionary of High column density hydrogen.
        dict_lst: List of dictionaries that contain Galxy data from
SDSS.

    Returns:
        A dictionary of redshifts mapped to a list of dictionaries
that
        contain data from galaxies that have redshifts with
unctertainties
        that contain the high column density hydrogen redshift.
    """

    H_lst = []
    for z in H_dict:
        H_lst.append(z)

    H_sys = {}
```

```
    for z in sorted(H_lst):
        z_lst = []
        for obj in dict_lst:
            upper = obj['z_up']
            lower = obj['z_down']
            z_poss = {}
            if z < upper and z > lower:
                z_poss['ID'] = obj['objID']
                z_poss['z'] = obj['z']
                z_poss['RA'] = obj['ra']
                z_poss['dec'] = obj['dec']
                z_poss['upper_z'] = upper
                z_poss['lower_z'] = lower
                z_poss['zErr'] = obj['zErr']
                z_lst.append(z_poss)
                H_sys[z] = z_lst
    return H_sys


######################################################################
##########
# Problem 4: Calculate the Actual Distance between QSO and galaxy
######################################################################
##########


def Redshift_range(Gal_dict):
    """Calculate the upper and lower extents of a galaxies redshift.

    Parameters:
        Gal_dict: list of dictionaries containing galaxies from the
SDSS.

    Returns:
        Adds keys and values of upper an lower bounds to the redshift
onto
        the input dictionaries within the list.
    """

    Gal_dict['z_up'] = Gal_dict['z'] + Gal_dict['zErr']
    Gal_dict['z_down'] = Gal_dict['z'] - Gal_dict['zErr']
    return Gal_dict


def angular_distance(ra, dec, QSO_ra=154.09418, QSO_dec=47.11204):
    """ Calculate the angular distance between two objects by using
    trigonometry.

    Parameters:
        ra: Right ascension of Galaxy (Degrees)
```

```
        dec: Declination of Galaxy (Degrees)
        QSO_ra: QSO ra (degrees)
        QSO_dec: QSO dec (degrees)

    Returns:
        returns the angluar distance between a galaxy and the QSO in
arc seconds
    """

    ang_distance = (abs(ra - QSO_ra) ** 2 + abs(dec - QSO_dec) ** 2)
** 1/2
    return ang_distance * 3600


def redshift_to_comdistance(z):
    """ Calculates the comoving distance using the redshift.

    Parameters:
        z: Redshift

    Returns:
        Returns the comoving distance of a galaxy.
    """

    # d = (z * c) / H_0 where c = speed of light (km/s), H_0 = Hubble
constant
    com_distance = z * (3 * (10 **5)) / 69.6
    return com_distance

def cos_scale(z, com_distance):
    """Calculates the cosmic scale at a redshift in kpc/arc second to
account
    for the fact that the universe is smaller at redshift z.

    Parameters:
        z: Redshift
        com_distance: Comoving distance

    Returns:
        Returns a scale in kpc/arc second
    """

    #From Cosomology Equations
    ang_dist = (1 / (1 + z)) * com_distance
    Scale = ang_dist / 206.26408
    return Scale


def Distance_of_Candidates(Candidates_dict):
    """ Calculates the distance between a candidate galaxy and the
```

```
QSO.

    Parameters:
        Candidates_dict: Dictionary of Galaxy info

    Returns:
        The angular distance between the Galaxy and the QSO
    """

    ra = Candidates_dict['RA']
    dec = Candidates_dict['dec']
    ang_dist = angular_distance(ra, dec, QSO_ra=154.09418,
QSO_dec=47.11204)
    return ang_dist


def Narrow_by_Dist(Candidates_dict):
    """Narrow the candidates by mapping the redshift where the high
column
    density hydrogen is at to the Galaxy with the smallest actual
distance
    between them.

    Parameters:
        Candidates_dict: Dictionary of redshifts mapped to list of
candidate
        quasar dictionaries.

    Returns:
        Returns a dictionary of redshifts mapped to the most likely
the
        dictionary of the most candidate based on actual distance.
    """

    Narrowed_dict = {}
    for z in Candidates_dict:
        min_dist = 1000
        for cand in Candidates_dict[z]:
            ang = Distance_of_Candidates(cand)
            cand['ang_dist'] = ang
            if ang < min_dist:
                min_dist = ang
                Narrowed_dict[z] = cand
    return Narrowed_dict


def Calc_actual_dist(Candidates_dict):
    """Calculates the Actual distance from the cosmic scale and
angular distance
```

```
    Parameters:
        Candidates_dict: list of candidate dictionaries

    Returns:

    """

    # * 1000 because we want kpc, not Mpc
    for z in Candidates_dict:
        cand_z = Candidates_dict[z]['z']
        co_move_dist = redshift_to_comdistance(cand_z)
        scale = cos_scale(cand_z, co_move_dist)
        Actual_dist = scale * Candidates_dict[z]['ang_dist']
        Candidates_dict[z]['kpc'] = Actual_dist * 1000
    return Candidates_dict


######################################################################
##########
# Problem 5: Display Findings and Plotting Galaxy/Gas Systems
######################################################################
##########


def display_results(final_dict, elt_dict,Candidates_dict):
    """Print out an easy to read list of the results.

    Parameters:
        final_dict: The dictionary with redshift mapped to the
dictionary of the
        most likely candidate galaxy.
        elt_dict: The dictionary of redshifts mapped to a list of all
elements
        found there.
        Candidates_dict: Dictionary that contains the error in the
galaxy
        redshifts.

    Returns:
        Prints Gas redshift, Galaxy redshift (with uncertainty),
Galaxy ID,
        Galaxy CGM size, and the elements found the the CGM.
    """

    for z in sorted(final_dict):
        Galaxy_z = final_dict[z]['z']
        Z_err = Candidates_dict[z]['zErr']
        Galaxy_ID = final_dict[z]['ID']
        Galaxy_Halo = final_dict[z]['kpc']
        Element_set = sorted(list(set(elt_dict[z])))
```

```
        print "Gas Location (Redshift):", z
        print "Galaxy Location (Redshift):", Galaxy_z, "+=", Z_err
        print "Galaxy ID (SDSS Catalogue):", Galaxy_ID
        print "Galaxy Halo (kpc):", Galaxy_Halo
        print "Elements Found:", Element_set
        print ""


def plot_results(final_dict,Candidates_dict,H_z,H_nc):
    """Plots all systems of hydrogen and galaxy locations (with
uncertainty)
    vs column density.

    Parameters:
        final_dict: Dictionary mapping redshift to single candidate.
        Candidates_dict: Dictionary containing the zErr of the galaxy.
        H_z: List of every Hydrogen system
        H_nc: List of every Hydrogen column density

    Returns:
        clears all previos plots.  Plots the systems of hydrogen on a
bar
        graph, where the y-axis is the column density, the galaxy
loctations,
        the upper and lower bounds of the galaxy locations.  Saves
plot as
        "QSO Galaxy Map"
    """

    Galaxy_z = []
    up_zErr = []
    down_zErr = []
    Galaxy_err = []
    for z in final_dict:
        Galaxy_z.append(final_dict[z]['z'])
        up = Candidates_dict[z]['zErr'] + final_dict[z]['z']
        down = final_dict[z]['z'] - Candidates_dict[z]['zErr']
        Galaxy_err.append(Candidates_dict[z]['zErr'])
        up_zErr.append(up)
        down_zErr.append(down)

    height = [max(H_nc) + 0.5 for i in range(len(Galaxy_z))]

    plt.clf()
    plt.xlabel('Redshift')
    plt.ylabel('Column Density')
    plt.xlim(0,0.8)
    plt.ylim(min(H_nc) - 0.5, max(H_nc) + 0.5)
    plt.title('HI column density', fontsize = 20)
    plt.bar(H_z, H_nc, 0.003, color = "green", label = "Hydrogen")
```

```
    plt.bar(Galaxy_z, height, 0.003, color = "Red", label = "Galaxy")
    plt.bar(up_zErr, height, 0.003, color = "cyan", label = "upper
Galaxy error")
    plt.bar(down_zErr, height, 0.003, color = "yellow",label = "lower
Galaxy error")
    plt.legend()
    plt.savefig("QSO Galaxy Map")
    plt.show()


def main(json_spectrum, Galaxies_csv):
    spectrum_dict = read_spectrum_data(json_spectrum)
    dat_lst = spectrum_to_list(spectrum_dict)
    dat_dict = system_dict(dat_lst)
    elements = element_dict(dat_dict)
    H_z = Hydrogen_z(dat_dict)
    H_nc = Hydrogen_Nc(dat_dict)
    Hydrogen = Hydrogen_sys(dat_dict)
    Galaxies_lst = Read_in_Galaxies(Galaxies_csv)
    Galaxy_dict = Galaxies_dict(Galaxies_lst)
    Search = Galaxy_Candidates(Hydrogen, Galaxy_dict)
    narrow = Narrow_by_Dist(Search)
    dist_dict = Calc_actual_dist(narrow)
    display_results(dist_dict, elements, narrow)
    plot_results(dist_dict,narrow,H_z,H_nc)

    assert len(elements[0.26378]) == 13
    assert elements[0.22011] == ['HI', 'HI', 'HI']
    assert len(dat_lst) == 100
    assert dat_dict[0.22011] == \
    {'Comment': 'H Lya Orphan', 'Column Density': 13.58, \
    'Doppler Profile': 24.7, 'Classification': 'b', 'element': 'HI'}
    assert Hydrogen_z([]) == []
    assert len(H_z) == len(H_nc)
    assert Search[0.1152][0]['upper_z'] == Search[0.1152][0]['z'] + \
    Search[0.1152][0]['zErr']
    assert Search[0.1152][0]['lower_z'] == Search[0.1152][0]['z'] - \
    Search[0.1152][0]['zErr']
    assert angular_distance(150.0, 30.0, 147.0, 26.0) == (12.5 * 3600)

if __name__ == "__main__":
    print "Spectrum json:"
    json_spectrum = raw_input()
    #json_spectrum = "dusty3_J1016+4706.json"
    print "Galaxies data:"
    Galaxies_csv = raw_input()
    #Galaxies_csv = "J1016_Galaxies.txt"
    main(json_spectrum, Galaxies_csv)
```