ASTR 519 Final Project: Cannon Dataset Mining

by Nick Konz, Henry Gu, and Roark Habegger

Here, we've combined (and cleaned up) all of the notebooks that we've used in this project.

0. CSV Creation/Data Cleaning (Roark)

In [3]:

```
# some of the usual imports
from astropy.io import fits
from astropy.table import Table
import numpy as np
import matplotlib.pyplot as plt
import os
%matplotlib inline

# files
file1 = "allStarCannon-l31c.2.fits"
file2 = "allStar-l31c.2.fits"

hdu_list = fits.open(file1)
abun_data = Table(hdu_list[1].data)
```

In [4]:

```
#This cell takes a while because the table takes up a lot of data
hdu_list2 = fits.open(file2)
vel_data = Table(hdu_list2[1].data)
```

In [5]:

```
cols = abun_data.colnames
myCols = []
myColNames = ["RA_HRS","DEC_DEG"]
for i in range(len(cols)):
    #Ditch all the raw errors
    if not ("RAW" in str(cols[i])):
        #Get all abundances
        if ("_H" in str(cols[i])):
            myCols.append(i)
            myColNames.append(str(cols[i]))
        #Get important non-abundance values that could be useful
        elif ("TEFF" in str(cols[i])) or ("LOGG" in str(cols[i])) or ("_
M" in str(cols[i])):
            myCols.append(i)
            myColNames.append(str(cols[i]))

myColNames.append('RAD_VEL')
myColNames.append('RAD_VEL_ERR')
print(myColNames)
```

```
['RA_HRS', 'DEC_DEG', 'TEFF', 'LOGG', 'M_H', 'ALPHA_M', 'FE_H', 'C_H',
'CI_H', 'N_H', 'O_H', 'NA_H', 'MG_H', 'AL_H', 'SI_H', 'P_H', 'S_H', 'K_
H', 'CA_H', 'TI_H', 'TIII_H', 'V_H', 'CR_H', 'MN_H', 'CO_H', 'NI_H', 'T
EFF_ERR', 'LOGG_ERR', 'M_H_ERR', 'ALPHA_M_ERR', 'FE_H_ERR', 'C_H_ERR',
'CI_H_ERR', 'N_H_ERR', 'O_H_ERR', 'NA_H_ERR', 'MG_H_ERR', 'AL_H_ERR',
'SI_H_ERR', 'P_H_ERR', 'S_H_ERR', 'K_H_ERR', 'CA_H_ERR', 'TI_H_ERR', 'T
III_H_ERR', 'V_H_ERR', 'CR_H_ERR', 'MN_H_ERR', 'CO_H_ERR', 'NI_H_ERR',
'RAD_VEL', 'RAD_VEL_ERR']
```

In [6]:

```
#Ditch entries without data
ii = (abun_data['FILENAME'] != '')

print(len(abun_data['APOGEE_ID']))
print(len(abun_data['APOGEE_ID'][ii]))
print(len(abun_data['APOGEE_ID'][ii])/len(abun_data['APOGEE_ID']))

#can parse ratios by error
```

```
277371
164074
0.5915326403985998
```

In [24]:

```
#Make array of current vals
myArr = np.array(abun_data[:][ii])
newArr = np.zeros([len(abun_data[:][ii]),len(myCols)+4],)
print(len(myArr))
print(newArr.shape)
locArr = np.array(vel_data['APOGEE_ID'][1:])
radArr = np.array(vel_data['VHELIO_AVG'][1:])
radErrArr = np.array(vel_data['VERR'][1:])
```

```
164074
(164074, 52)
```

In [25]:
```
i=0
l1=0

for i in range(len(myArr)):
    myrow = myArr[i]

    #Parse RA
    locStr = myrow[0]
    raStr = locStr[2:10]
    hrs = int(raStr[0:2])
    mins = int(raStr[2:4])
    secs = int(raStr[4:6])+ int(raStr[6:8])/100.0
    ra = hrs+mins/60.0+secs/3600.0

    #Parse Dec
    decStr = locStr[10:]
    if decStr[0] == "+":
        decSgn = 1.0
    else:
        decSgn = -1.0
    deg = int(decStr[1:3])*decSgn
    arcMins = int(decStr[3:5])*decSgn
    arcSecs = (int(decStr[5:7])+int(decStr[7])/10.0)*decSgn
    dec = deg+arcMins/60.0+arcSecs/3600.0


    #Save RA and Dec
    newArr[i,:2] = [ra,dec]

    ##Get Rad velocity
    ind = -1
    for l in range(l1,len(locArr)):
        if locArr[l]  == locStr:
            ind = l
            l1 = l
            break
    newArr[i,-2] = radArr[ind]
    newArr[i,-1]= radErrArr[ind]

    #Save Abundance Values
    k=0
    for j in myCols:
        colNom = myColNames[2+k]
        val=0.0
        #IF the value is bad, replace with a NaN to take up less data
        if (colNom == "NA_H_ERR"):
            valInd = 2+k-24
            if (newArr[i,valInd]<=-100.0):
                newArr[i,valInd] = np.nan
                val=np.nan
            else:
                val=myrow[j]

        if ("_ERR" in colNom) and (myrow[j] >=1) and not ("TEFF" in colN
om):
            val = np.nan
            valInd = 2+k-24
            newArr[i,valInd]=np.nan

        else:
            val = myrow[j]

        newArr[i,2+k] = val
        k+=1
    perc = i/len(myArr)

    #Sometimes print out the fraction of data we have parsed and organiz
ed
    if int(i/20000.0) == i/20000.0:
        print(perc)
```

```
0.0
0.1218962175603691
0.2437924351207382
0.3656886526811073
0.4875848702414764
0.6094810878018455
0.7313773053622146
0.8532735229225837
0.9751697404829528
```

In [26]:

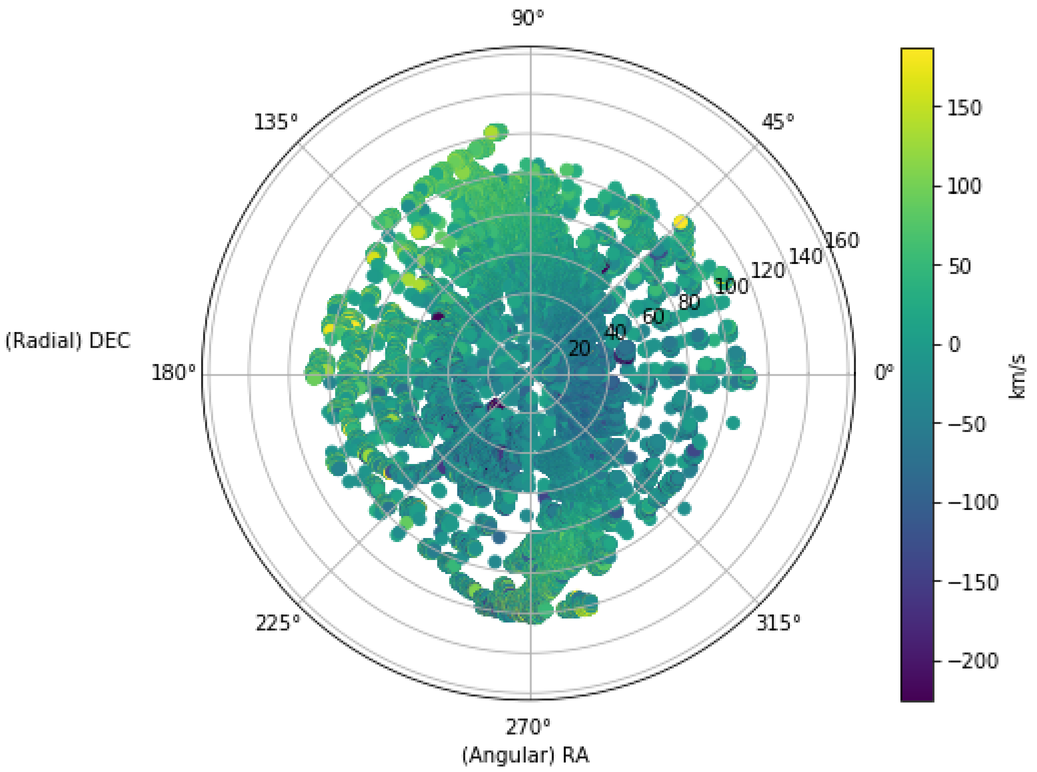
```
#Save the File to Cannon.csv with proper header
saveFile = "Cannon.csv"
headStr = ""
for entry in myColNames:
    headStr += entry
    headStr += ','
np.savetxt(saveFile,newArr,delimiter=',',fmt="%1.8e",header=headStr[:-1
],comments='')
```

In [27]:

```
# Plot velocities vs. equitorial coordinates:
myArr = np.loadtxt(saveFile,delimiter=',',skiprows=1)
headArr = np.loadtxt(saveFile,delimiter=',',max_rows=1,dtype=str)
data = {}
for i in range(len(headArr)):
    data[headArr[i]] = myArr[:,i]
```

In [28]:

```
maxVel = np.max(np.abs(data['RAD_VEL']))

scaledVel = data['RAD_VEL']/maxVel
#plt.figure(1)
#plt.hist(scaledVel,range=[-0.24,0.2],bins=100)

th = data['RA_HRS']*2*np.pi/24.0
rad = -data['DEC_DEG']+90

fig = plt.figure(2,figsize=[8,6])
ax = fig.add_subplot(111, projection='polar')
c = ax.scatter(th, rad, c=data['RAD_VEL'], cmap='viridis',vmin=-0.24*max
Vel,vmax=0.2*maxVel)

#c.colorbar(label='km/s')
ax.set_xlabel("(Angular) RA ")
ax.set_ylabel("(Radial) DEC          \n        ",rotation=0,ha='right')
fig.colorbar(c,label='km/s')
plt.show()
```



2. K-Means (Henry)

Method I

Below is a brief introduction of what I did in Method I.

1. I focus on 20 different elements' chemical abundance of each star

2. I use 19 elements' chemical abundance as the training set. Then I feed the chemical abundance to the K-Mean to get a result of the clustering. Here, the problem is: in K-Mean, you must tell the algorithm how many clusters are you looking for in advance, but I don't know, so at this step, I start from just looking for 2 clusters and use a for loop to increase the number of clusters(up to 100). Therefore, I get a result each time I increase the cluster number.

3. I use the 1 element's chemical abundance left as the validation set. Then I feed the chemical abundance to the K-Mean to get another result of the clustering. Like what I described in step 2, I get a result each time I increase the cluster number.

4. I define the result of step 2 as the Predicted labels, and the result of step 3 as the True labels. Then I use the accuracy_socre in K-Mean to get the accuracy score of my fit, and use the score as the uncertainty. The idea is: if two stars are in the same group, the algorithm should give me the same clustering result for both the 19-element training set and the 1-element validation set.

5. I compare the accuracy score I get for different cluster numbers I am looking for and pick the highest score as the answer. For example, if I reach the highest score when I divided all stars into 8 clusters, then I think the stars should be partitioned into 8 parts.

6. Then I use the cluster numbers I got in step 5 and and apply it to the scatter plot of the position of the stars by using different colors to see if the clustering result output using chemical abundance will lead to an obvious grouping in physical space.

In [55]:

```
# Read in data
data, header = fits.getdata("allStarCannon-l31c.2.fits", header=True)
fits_inf = fits.open("allStarCannon-l31c.2.fits")
```

In [56]:

```
# Trim the data to get all "good" rows
good_data_index = np.where(data["LOCATION_ID"] > 0)[0] # index of the "good" rows which contains useful data
data_refined = data[good_data_index]
```

In [57]:

```
# Get the chemical aboundance data
M_H = data["M_H"][good_data_index]
ALPHA_M = data["ALPHA_M"][good_data_index]
FE_H = data["FE_H"][good_data_index]
C_H = data["C_H"][good_data_index]
CI_H = data["CI_H"][good_data_index]
N_H = data["N_H"][good_data_index]
O_H = data["O_H"][good_data_index]
NA_H = data["NA_H"][good_data_index]
MG_H = data["MG_H"][good_data_index]
AL_H = data["AL_H"][good_data_index]
SI_H = data["SI_H"][good_data_index]
P_H = data["P_H"][good_data_index]
S_H = data["S_H"][good_data_index]
K_H = data["K_H"][good_data_index]
CA_H = data["CA_H"][good_data_index]
TI_H = data["TI_H"][good_data_index]
TIII_H = data["TIII_H"][good_data_index]
V_H = data["V_H"][good_data_index]
CR_H = data["CR_H"][good_data_index]
MN_H = data["MN_H"][good_data_index]
CO_H = data["CO_H"][good_data_index]
NI_H = data["NI_H"][good_data_index]
```

In [58]:

```
# Store the abundance data into a very large array, each row represents
 a star and each column represents an element in the star
chemical_abundance = np.zeros((data_refined.size, 20))
chemical_abundance[:, 0] = FE_H
chemical_abundance[:, 1] = C_H
chemical_abundance[:, 2] = CI_H
chemical_abundance[:, 3] = N_H
chemical_abundance[:, 4] = O_H
chemical_abundance[:, 5] = NA_H
chemical_abundance[:, 6] = MG_H
chemical_abundance[:, 7] = AL_H
chemical_abundance[:, 8] = SI_H
chemical_abundance[:, 9] = P_H
chemical_abundance[:, 10] = S_H
chemical_abundance[:, 11] = K_H
chemical_abundance[:, 12] = CA_H
chemical_abundance[:, 13] = TI_H
chemical_abundance[:, 14] = TIII_H
chemical_abundance[:, 15] = V_H
chemical_abundance[:, 16] = CR_H
chemical_abundance[:, 17] = MN_H
chemical_abundance[:, 18] = CO_H
chemical_abundance[:, 19] = NI_H
```

In [59]:

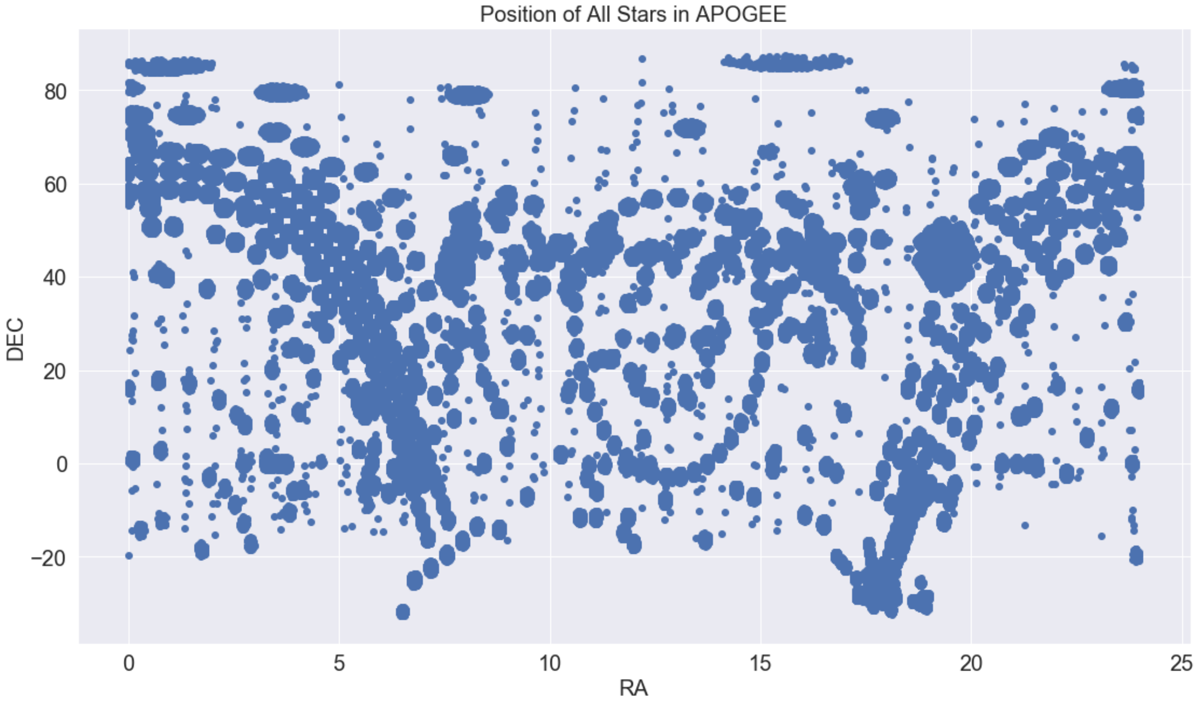
```
chemical_abundance_training_set = chemical_abundance[:, 1:] # traning se
t includes all chemical abundance except the abundance for Fe
chemical_abundance_validation_set = chemical_abundance[:, 0:1] # validat
ion set is the Fe abundance
```

In [60]:

```
# Get the position(RA & DEC) information of each star
ID = data["APOGEE_ID"][good_data_index]
RA = []
DEC = []
for i in ID:
    RA.append(float(i[2:4]) + float(i[4:6]) / 60 + float(i[6:8]) / 3600
+ float(i[8:10]) / 360000)
    dec = float(i[11:13]) + float(i[13:15]) / 60 + float(i[15:17]) / 360
0 + float(i[17:18]) / 36000
    if (i[10:11] == '+'):
        DEC.append(dec)
    else:
        DEC.append(-dec)
RA = np.array(RA)
DEC = np.array(DEC)
```

In [61]:

```
# Plot the stars' positions
fig = plt.figure()
fig.set_size_inches(18.5, 10.5)
plt.scatter(RA, DEC)
plt.xlabel('RA', fontsize = 20)
plt.xticks(fontsize = 20)
plt.ylabel('DEC', fontsize = 20)
plt.yticks(fontsize = 20)
plt.title('Position of All Stars in APOGEE', fontsize = 20)
plt.show()
```



In [62]:

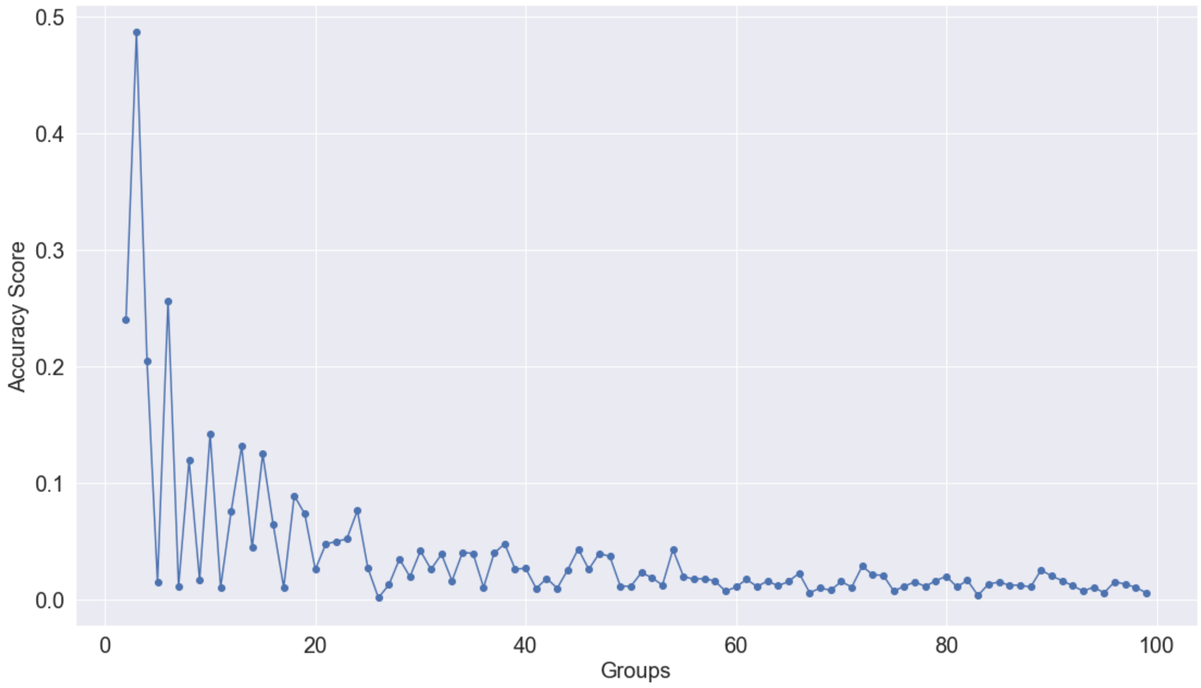
```
# This is the step 2, 3, and 4 in my introduction, which is also very time consuming
max_possible_groups = 100
score = []
for i in range(2, max_possible_groups):
    est = KMeans(i)
    est.fit(chemical_abundance_training_set)
    y_kmeans_training = est.predict(chemical_abundance_training_set)

    est.fit(chemical_abundance_validation_set)
    y_kmeans_validation = est.predict(chemical_abundance_validation_set)
    score.append(accuracy_score(y_kmeans_validation, y_kmeans_training))
    print(str('%.3f'%(((i - 1) / (max_possible_groups - 2)) * 100) + " %complete"))

score = np.array(score)
```

```
1.020 % complete
2.041 % complete
3.061 % complete
4.082 % complete
5.102 % complete
6.122 % complete
7.143 % complete
8.163 % complete
9.184 % complete
10.204 % complete
11.224 % complete
12.245 % complete
13.265 % complete
14.286 % complete
15.306 % complete
16.327 % complete
17.347 % complete
18.367 % complete
19.388 % complete
20.408 % complete
21.429 % complete
22.449 % complete
23.469 % complete
24.490 % complete
25.510 % complete
26.531 % complete
27.551 % complete
28.571 % complete
29.592 % complete
30.612 % complete
31.633 % complete
32.653 % complete
33.673 % complete
34.694 % complete
35.714 % complete
36.735 % complete
37.755 % complete
38.776 % complete
39.796 % complete
40.816 % complete
41.837 % complete
42.857 % complete
43.878 % complete
44.898 % complete
45.918 % complete
46.939 % complete
47.959 % complete
48.980 % complete
50.000 % complete
51.020 % complete
52.041 % complete
53.061 % complete
54.082 % complete
55.102 % complete
56.122 % complete
57.143 % complete
58.163 % complete
59.184 % complete
60.204 % complete
61.224 % complete
62.245 % complete
63.265 % complete
64.286 % complete
65.306 % complete
66.327 % complete
67.347 % complete
68.367 % complete
69.388 % complete
70.408 % complete
71.429 % complete
72.449 % complete
73.469 % complete
74.490 % complete
75.510 % complete
```

```
76.531 % complete
77.551 % complete
78.571 % complete
79.592 % complete
80.612 % complete
81.633 % complete
82.653 % complete
83.673 % complete
84.694 % complete
85.714 % complete
86.735 % complete
87.755 % complete
88.776 % complete
89.796 % complete
90.816 % complete
91.837 % complete
92.857 % complete
93.878 % complete
94.898 % complete
95.918 % complete
96.939 % complete
97.959 % complete
98.980 % complete
100.000 % complete
```

In [63]:

```
# This is the step 5 in my introduction
# Plot the relation between the number of groups divided and the accurac
y score
fig = plt.figure()
fig.set_size_inches(18.5, 10.5)
groups = np.arange(2, max_possible_groups, 1)
plt.plot(groups, score, 'o-')
plt.xlabel('Groups', fontsize = 20)
plt.xticks(fontsize = 20)
plt.ylabel('Accuracy Score', fontsize = 20)
plt.yticks(fontsize = 20)
plt.show()
```



Below is Fig1: Relation between the number of clusters and its corresponding accuracy score. Here, the accuracy score is calculated by using the 19(except Fe) elements' chemical abundance as the training set and the rest 1(Fe) element's chemical abundance as the validation set.

title

From this figure, I can see that when I partition the stars into 5 groups, the results from the training set and the validation set is the closest, and an accuracy score of over 0.7 is not bad. Therefore, from the chemical abundance, I conclude that the stars should be put into 5 groups.

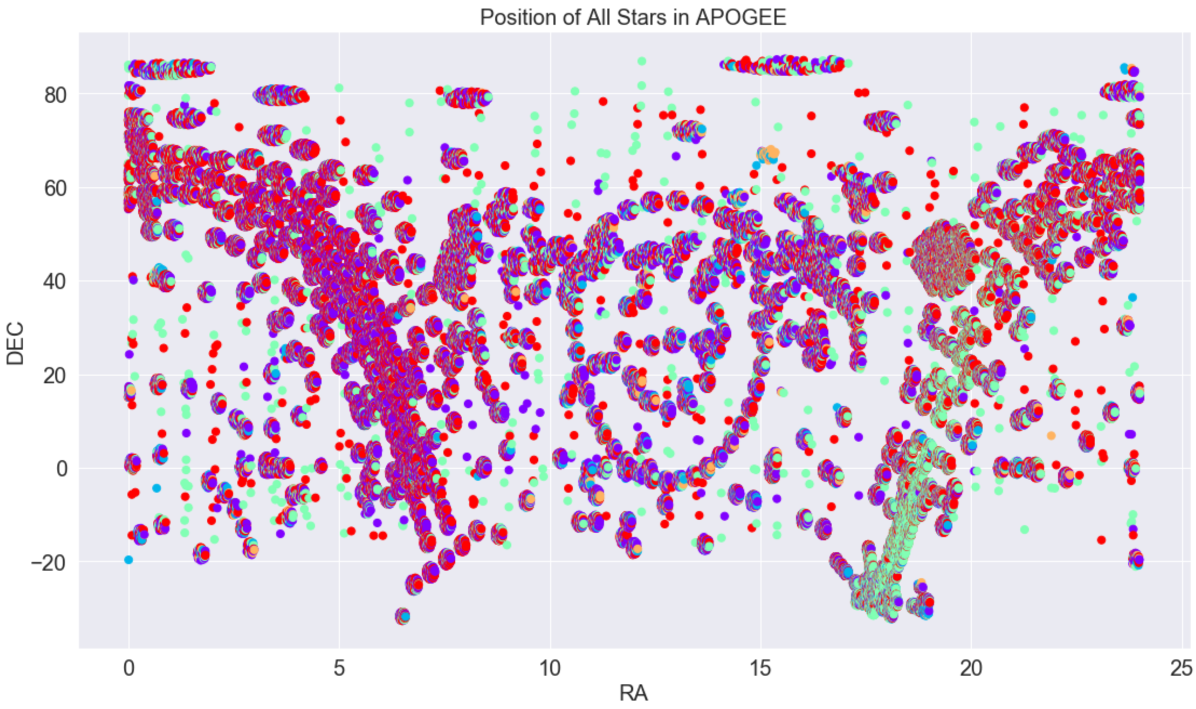
In [64]:

```
# See how good the fits are
print("The highest score corresponds to a partition of " + str(2 + np.wh
ere(score == np.max(score))[0][0]) + " groups")
print("At that partition, the accuracy score is " + str(np.max(score)))
```

```
The highest score corresponds to a partition of 3 groups
At that partition, the accuracy score is 0.4864268561746529
```

In [65]:

```
# This is the step 6 in my introduction
est = KMeans(5) # partition into 5 groups is the best choice
est.fit(chemical_abundance_training_set)
y_kmeans_training = est.predict(chemical_abundance_training_set)

fig = plt.figure()
fig.set_size_inches(18.5, 10.5)
plt.scatter(RA, DEC, c=y_kmeans_training, s=50, cmap='rainbow');
plt.xlabel('RA', fontsize = 20)
plt.xticks(fontsize = 20)
plt.ylabel('DEC', fontsize = 20)
plt.yticks(fontsize = 20)
plt.title('Position of All Stars in APOGEE', fontsize = 20)
plt.show()
```



Below is Fig2: Distribution of the members of all 5 groups in physical space. In this plot, members from different groups are represented using different colors.

title

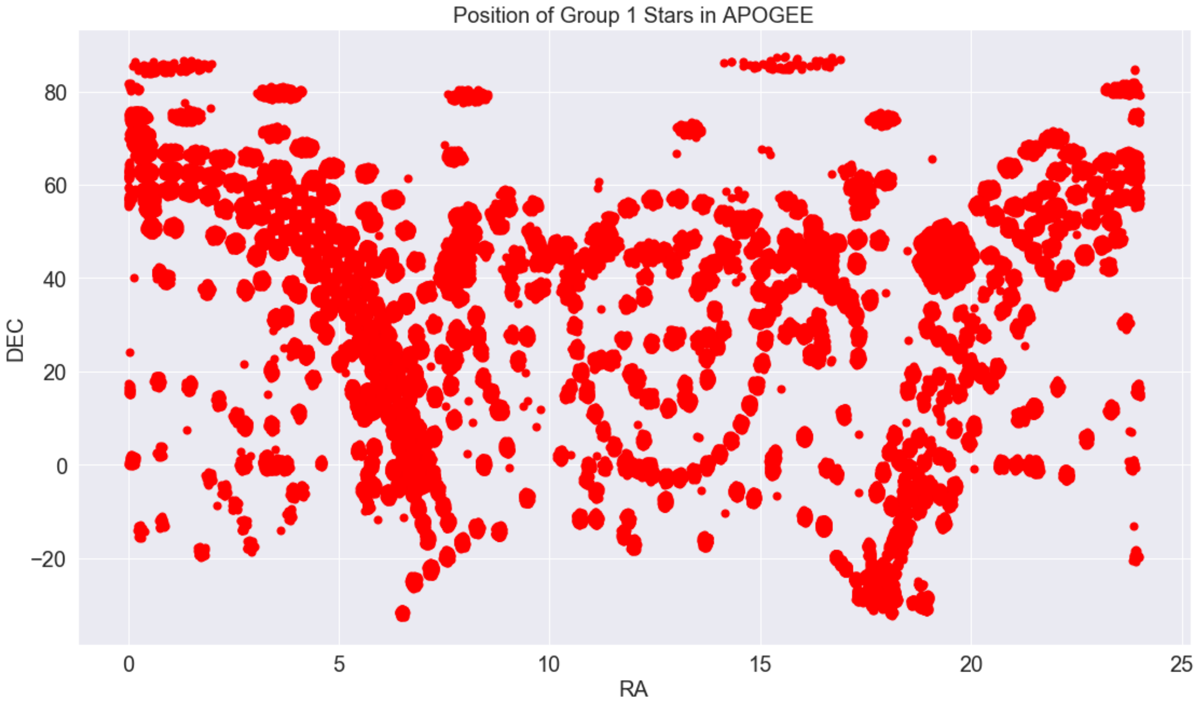
In [66]:

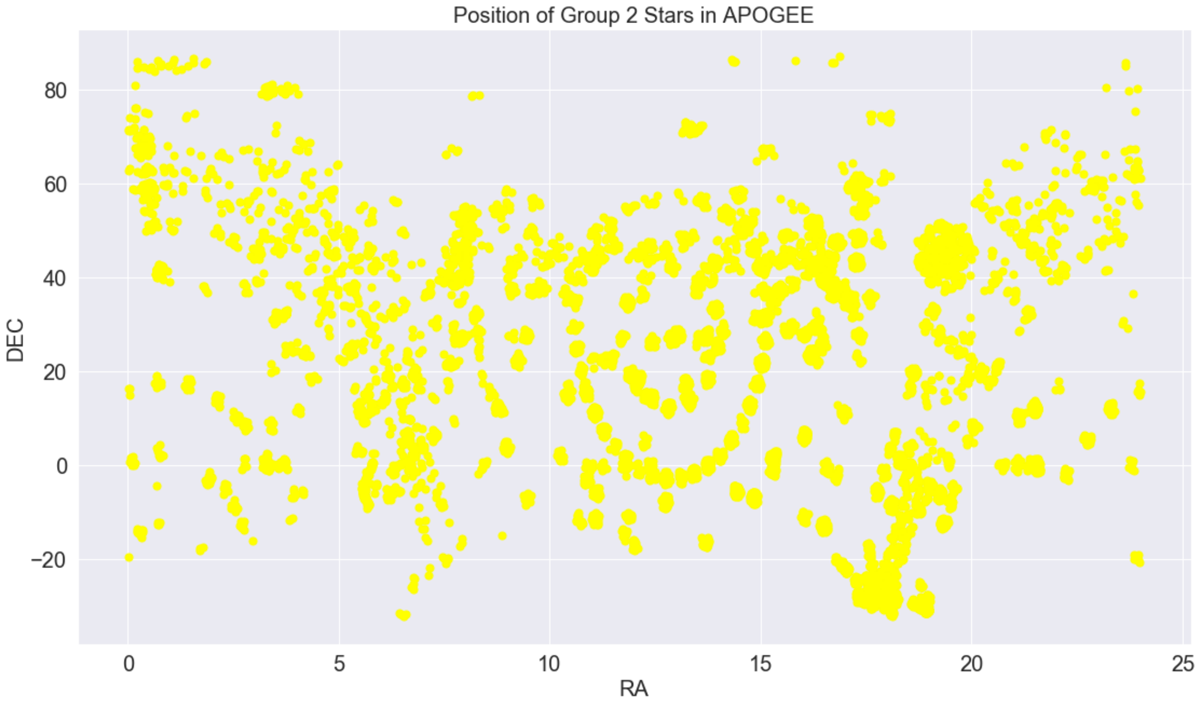
```
# Check how many stars are in each group
group_1_index = np.where(y_kmeans_training == 0)[0]
group_2_index = np.where(y_kmeans_training == 1)[0]
group_3_index = np.where(y_kmeans_training == 2)[0]
group_4_index = np.where(y_kmeans_training == 3)[0]
group_5_index = np.where(y_kmeans_training == 4)[0]
print("There are " + str(y_kmeans_training.size) + " stars in total")
print("Group 1 has " + str(len(group_1_index)) + " members, which is " +
str('%.3f'%(len(group_1_index) / y_kmeans_training.size * 100)) + " % of
the total stars")
print("Group 2 has " + str(len(group_2_index)) + " members, which is " +
str('%.3f'%(len(group_2_index) / y_kmeans_training.size * 100)) + " % of
the total stars")
print("Group 3 has " + str(len(group_3_index)) + " members, which is " +
str('%.3f'%(len(group_3_index) / y_kmeans_training.size * 100)) + " % of
the total stars")
print("Group 4 has " + str(len(group_4_index)) + " members, which is " +
str('%.3f'%(len(group_4_index) / y_kmeans_training.size * 100)) + " % of
the total stars")
print("Group 5 has " + str(len(group_5_index)) + " members, which is " +
str('%.3f'%(len(group_5_index) / y_kmeans_training.size * 100)) + " % of
the total stars")
```

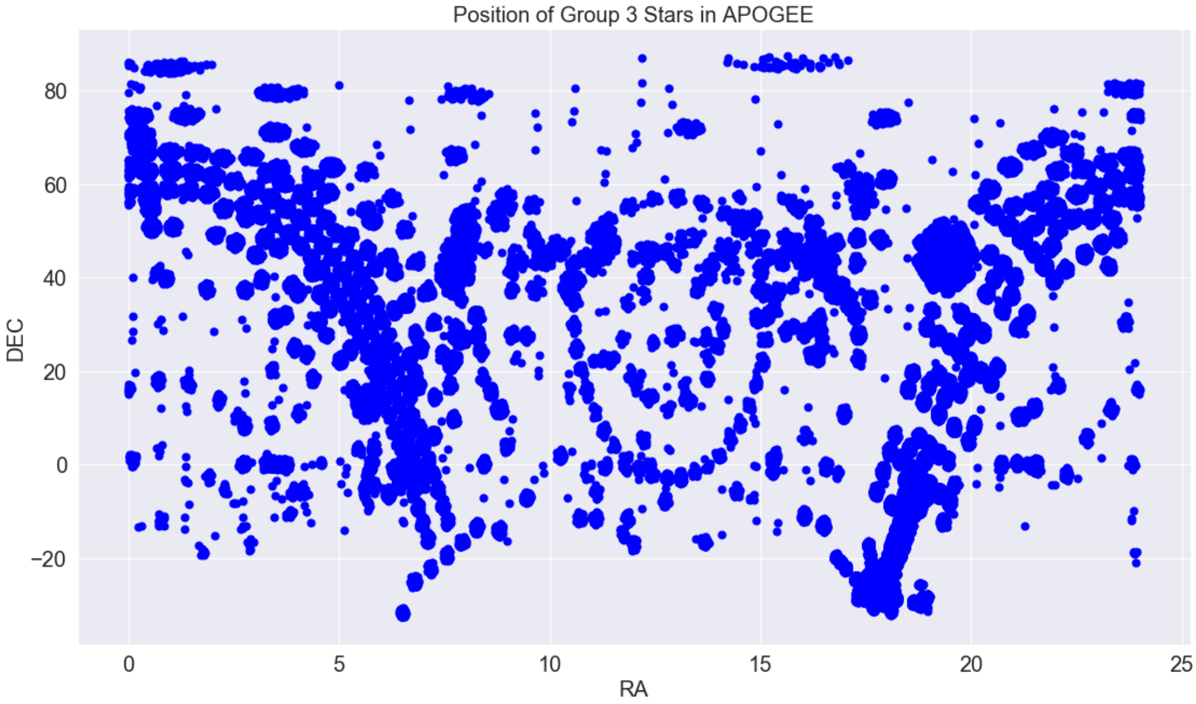
```
There are 164074 stars in total
Group 1 has 50439 members, which is 30.742 % of the total stars
Group 2 has 7227 members, which is 4.405 % of the total stars
Group 3 has 43428 members, which is 26.469 % of the total stars
Group 4 has 5314 members, which is 3.239 % of the total stars
Group 5 has 57666 members, which is 35.146 % of the total stars
```

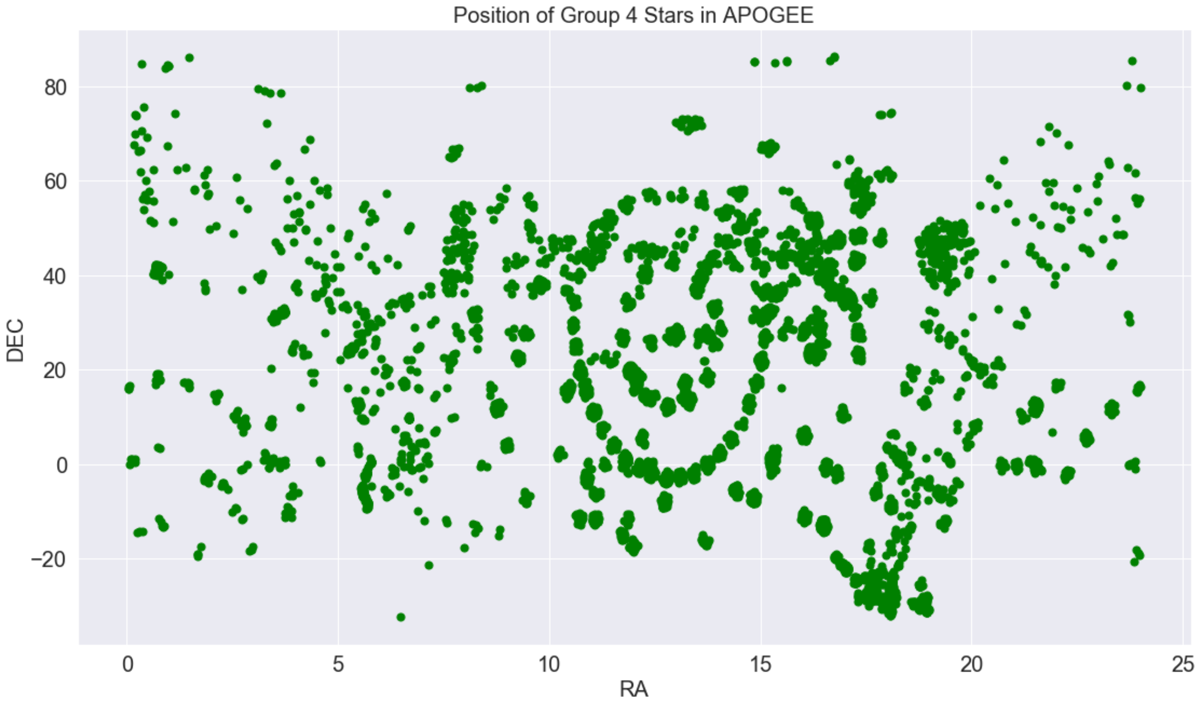
In [67]:

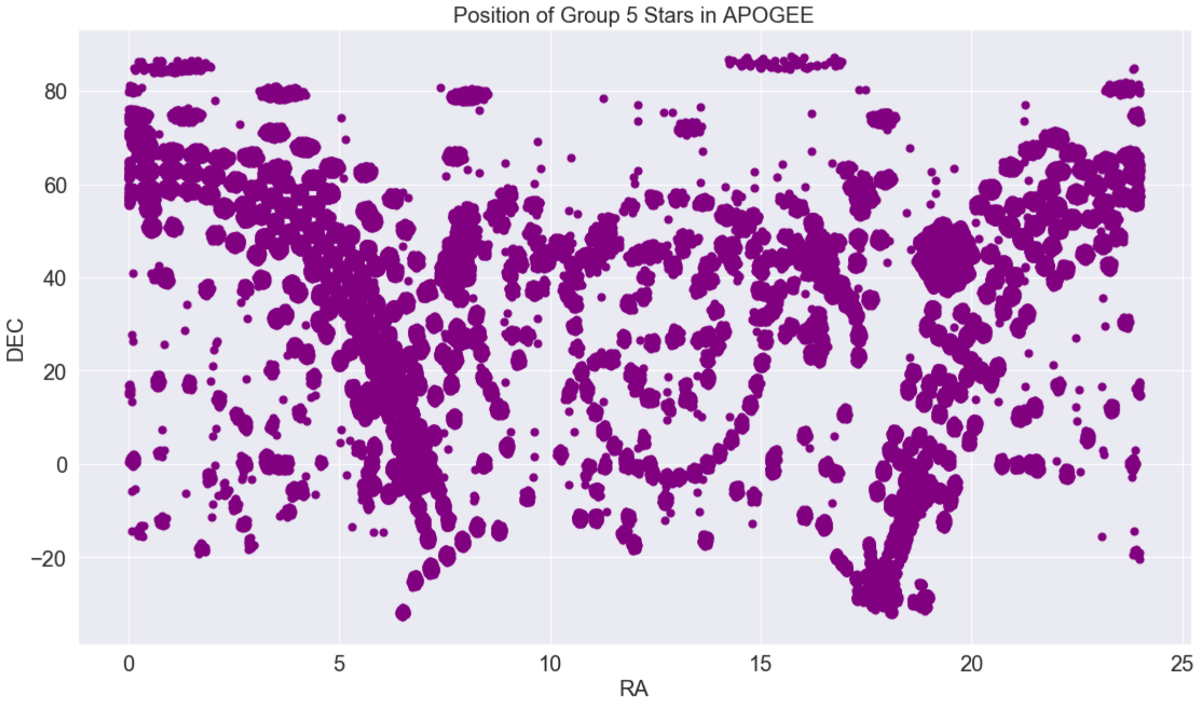
```
# Plot the positions of stars in each group
# Group 1
fig = plt.figure()
fig.set_size_inches(18.5, 10.5)
plt.scatter(RA[group_1_index], DEC[group_1_index], c='red', s=50, cmap=
'rainbow');
plt.xlabel('RA', fontsize = 20)
plt.xticks(fontsize = 20)
plt.ylabel('DEC', fontsize = 20)
plt.yticks(fontsize = 20)
plt.title('Position of Group 1 Stars in APOGEE', fontsize = 20)
plt.show()

# Group 2
fig = plt.figure()
fig.set_size_inches(18.5, 10.5)
plt.scatter(RA[group_2_index], DEC[group_2_index], c='yellow', s=50, cma
p='rainbow');
plt.xlabel('RA', fontsize = 20)
plt.xticks(fontsize = 20)
plt.ylabel('DEC', fontsize = 20)
plt.yticks(fontsize = 20)
plt.title('Position of Group 2 Stars in APOGEE', fontsize = 20)
plt.show()

# Group 3
fig = plt.figure()
fig.set_size_inches(18.5, 10.5)
plt.scatter(RA[group_3_index], DEC[group_3_index], c='blue', s=50, cmap=
'rainbow');
plt.xlabel('RA', fontsize = 20)
plt.xticks(fontsize = 20)
plt.ylabel('DEC', fontsize = 20)
plt.yticks(fontsize = 20)
plt.title('Position of Group 3 Stars in APOGEE', fontsize = 20)
plt.show()

# Group 4
fig = plt.figure()
fig.set_size_inches(18.5, 10.5)
plt.scatter(RA[group_4_index], DEC[group_4_index], c='green', s=50, cmap
='rainbow');
plt.xlabel('RA', fontsize = 20)
plt.xticks(fontsize = 20)
plt.ylabel('DEC', fontsize = 20)
plt.yticks(fontsize = 20)
plt.title('Position of Group 4 Stars in APOGEE', fontsize = 20)
plt.show()

# Group 5
fig = plt.figure()
fig.set_size_inches(18.5, 10.5)
plt.scatter(RA[group_5_index], DEC[group_5_index], c='purple', s=50, cma
p='rainbow');
plt.xlabel('RA', fontsize = 20)
plt.xticks(fontsize = 20)
plt.ylabel('DEC', fontsize = 20)
plt.yticks(fontsize = 20)
plt.title('Position of Group 5 Stars in APOGEE', fontsize = 20)
plt.show()
```


Position of Group 1 Stars in APOGEE
DEC
80
60
40
20
0
−20
RA
0
5
10
15
20
25


Position of Group 2 Stars in APOGEE
DEC
80
60
40
20
0
−20
RA
0
5
10
15
20
25


Position of Group 3 Stars in APOGEE
DEC
80
60
40
20
0
−20
RA
0
5
10
15
20
25





Below are Fig3 to Fig7: Distribution of the members of the 5 groups in physical space.

title

title

title

title

title

Really, there seems no obvious physical boundaries between those 5 groups. Therefore, the clustering result output using chemical abundance will NOT lead to an obvious grouping in physical space.

Method II

Now add the information about the RA and DEC

This time, I still use the 20 elements' chemical abundance used in methid I, but together with the RA and DEC informatiuon

I change the training set to 10 out of 20 elements' chemical abundance together with RA and DEC, and the validation set to the rest 10 elements' chemical abundance together with RA and DEC

I repeat the experiment at most 184756 times(which is almost impossible), each time I use a different combination of 10 elements for the validation set

Then I use the similar methods described in Method I(step 4 and 5) to plot the relation between the number of groups and the accuracy score for each training and validation set, and get a peak for each plot, which indicates the optimal groups of clusters the K-Mean algorithm suggests.

Finally, I compare all the peaks I get and see if most of them indicates the same number of groups.

First, I focused on finding cluster numbers from 2 to 10(Warning: This needs 2-3 mins to generate 1 plot, and there will be 184756 total combinations, which will make the code running forever. For testing purpose, just let it generate 2-3 plots to get a basic feeling and manually terminate the algorithm.)

In [68]:

```python
total_combination = 184756 # All possible ways of choosing 10 elements from 20 elements
max_possible_groups = 10
training_set_column_collection = []
validation_set_column_collection = []
optimal_number_of_clusters_collection = []
highest_accuracy_score_collection = []
score_collection = []

while (len(training_set_column_collection) < total_combination):
    training_set_column = [] # First, generate 10 random numbers from 0 to 19, which indicates which column of the chemical abundance will be used as the training set
    # The rest 10 columns will be used as the validation set
    while (len(training_set_column) < 10):
        random_int = random.randint(0, 19) # 0 to 19, both included
        if random_int not in training_set_column:
            training_set_column.append(random_int)
    training_set_column.sort()
    if training_set_column not in training_set_column_collection:
        training_set_column_collection.append(training_set_column)
        # print("I've found " + str(len(training_set_column_collection)) + " different combinations")
    else:
        continue

    # Now we've had one 10-random-int. Pick the corresponding columns as the training set
    # First, figure out the validation set columns(int not in the 10-random-int)
    validation_set_column = []
    for i in range(0, 20):
        if i not in training_set_column:
            validation_set_column.append(i)
    validation_set_column_collection.append(validation_set_column)

    chemical_abundance_training_set = np.zeros((RA.size, 12))
    chemical_abundance_validation_set = np.zeros((RA.size, 12))

    for i in range(0, 10):
        chemical_abundance_training_set[:, i] = chemical_abundance[:, training_set_column[i]]
    chemical_abundance_training_set[:, 10] = RA
    chemical_abundance_training_set[:, 11] = DEC

    for i in range(0, 10):
        chemical_abundance_validation_set[:, i] = chemical_abundance[:, validation_set_column[i]]
    chemical_abundance_validation_set[:, 10] = RA
    chemical_abundance_validation_set[:, 11] = DEC

    score = [] # accuracy score for one combination of training and validation set
    for i in range(2, max_possible_groups + 1):
        est = KMeans(i)
        est.fit(chemical_abundance_training_set)
        y_kmeans_training = est.predict(chemical_abundance_training_set)

        est.fit(chemical_abundance_validation_set)
        y_kmeans_validation = est.predict(chemical_abundance_validation_set)
        score.append(accuracy_score(y_kmeans_validation, y_kmeans_training))
        print("Combination " + str(len(training_set_column_collection)) + ": " + str('%.3f'%(((i - 1) / (max_possible_groups - 1)) * 100) + " % complete"))
    print()
    score_collection.append(score)

    optimal_number_of_clusters = int(np.where(score == np.max(score))[0][0]) + 2
    optimal_number_of_clusters_collection.append(optimal_number_of_clusters)
    highest_accuracy_score = np.max(score)
    highest_accuracy_score_collection.append(highest_accuracy_score)

    # Plot the result for this combination
    fig = plt.figure()
    fig.set_size_inches(18.5, 10.5)
    groups = np.arange(2, max_possible_groups + 1, 1)
    plt.plot(groups, score, 'o-')
    plt.xlabel('Groups', fontsize = 20)
    plt.xticks(fontsize = 20)
    plt.ylabel('Accuracy Score', fontsize = 20)
    plt.yticks(fontsize = 20)
    plt.vlines(optimal_number_of_clusters, 0, highest_accuracy_score, co
lor = 'green', linestyle = 'dashed')
    plt.show()

    print()
    print("Combination " + str(len(training_set_column_collection)) + ":
")
    print("The optimal number of clusters is: " + str(optimal_number_of_
clusters))
    print("Its corresponding accuracy score is: " + str(highest_accuracy
_score))
    print()
```

```
-----------------------------------------------------------------------
----
NameError                                 Traceback (most recent call l
ast)
<ipython-input-68-ed1285829a5f> in <module>
     11     # The rest 10 columns will be used as the validation set
     12     while (len(training_set_column) < 10):
---> 13         random_int = random.randint(0, 19) # 0 to 19, both incl
uded
     14         if random_int not in training_set_column:
     15             training_set_column.append(random_int)

NameError: name 'random' is not defined
```

In [ ]:

```
optimal_number_of_clusters_collection = np.array(optimal_number_of_clust
ers_collection)
print("Total combinations tested: " + str(optimal_number_of_clusters_col
lection.size))
print(stats.mode(optimal_number_of_clusters_collection))

highest_accuracy_score_collection = np.array(highest_accuracy_score_coll
ection)
print(np.mean(highest_accuracy_score_collection))
```

In [ ]:

```
fig = plt.figure()
fig.set_size_inches(18.5, 10.5)
plt.hist(optimal_number_of_clusters_collection, bins = np.arange(2, 11,
1))
plt.xlabel('Groups', fontsize = 20)
plt.xticks(fontsize = 20)
plt.ylabel('Counts', fontsize = 20)
plt.yticks(fontsize = 20)
plt.show()
```

Below is Fig8: Counts of the optimal number of clusters(highest accuracy score) for all 552 combinations.

title

```
In [ ]: fig = plt.figure()
        fig.set_size_inches(18.5, 10.5)
        plt.hist(highest_accuracy_score_collection, bins = np.arange(0, 1, 0.1))
        plt.xlabel('Accuracy Score', fontsize = 20)
        plt.xticks(fontsize = 20)
        plt.ylabel('Counts', fontsize = 20)
        plt.yticks(fontsize = 20)
        plt.show()
```

Below is Fig9: Distribution of the accuracy scores of all 552 combinations. Here, each accuracy score is calculated by using the 10 randomly picked elements' chemical abundance(together with RA and DEC) as the training set and the rest 10 element's chemical abundance(together with RA and DEC) as the validation set.

title

Below is Fig10: Relation between the accuracy score and the cluster numbers(2 to 10). Here, the accuracy score is calculated by using the 10 randomly picked elements' chemical abundance(together with RA and DEC) as the training set and the rest 10 element's chemical abundance(together with RA and DEC) as the validation set. 2 is picked as the optimal cluster number, which is picked most frequently by the algorithm.

title

Below is Fig11: Relation between the accuracy score and the cluster numbers(2 to 10). Here, the accuracy score is calculated by using the 10 randomly picked elements' chemical abundance(together with RA and DEC) as the training set and the rest 10 element's chemical abundance(together with RA and DEC) as the validation set. 5 is picked as the cluster number since it has the highest accuracy score, which is slightly higher than 2.

title

Then I focused on finding cluster numbers from 20 to 30

In [ ]:

```
total_combination_2 = 184756 # All possible ways of choosing 10 elements from 20 elements
max_possible_groups_2 = 30
training_set_column_collection_2 = []
validation_set_column_collection_2 = []
optimal_number_of_clusters_collection_2 = []
highest_accuracy_score_collection_2 = []
score_collection_2 = []

while (len(training_set_column_collection_2) < total_combination_2):
    training_set_column_2 = [] # First, generate 10 random numbers from 0 to 19, which indicates which column of the chemical abundance will be used as the training set
    # The rest 10 columns will be used as the validation set
    while (len(training_set_column_2) < 10):
        random_int_2 = random.randint(0, 19) # 0 to 19, both included
        if random_int_2 not in training_set_column_2:
            training_set_column_2.append(random_int_2)
    training_set_column_2.sort()
    if training_set_column_2 not in training_set_column_collection_2:
        training_set_column_collection_2.append(training_set_column_2)
        # print("I've found " + str(len(training_set_column_collection)) + " different combinations")
    else:
        continue

    # Now we've had one 10-random-int. Pick the corresponding columns as the training set
    # First, figure out the validation set columns(int not in the 10-random-int)
    validation_set_column_2 = []
    for i in range(0, 20):
        if i not in training_set_column_2:
            validation_set_column_2.append(i)
    validation_set_column_collection_2.append(validation_set_column_2)

    chemical_abundance_training_set_2 = np.zeros((RA.size, 12))
    chemical_abundance_validation_set_2 = np.zeros((RA.size, 12))

    for i in range(0, 10):
        chemical_abundance_training_set_2[:, i] = chemical_abundance[:, training_set_column_2[i]]
    chemical_abundance_training_set_2[:, 10] = RA
    chemical_abundance_training_set_2[:, 11] = DEC

    for i in range(0, 10):
        chemical_abundance_validation_set_2[:, i] = chemical_abundance[:, validation_set_column_2[i]]
    chemical_abundance_validation_set_2[:, 10] = RA
    chemical_abundance_validation_set_2[:, 11] = DEC

    score_2 = [] # accuracy score for one combination of training and validation set
    for i in range(21, max_possible_groups_2 + 1):
        est_2 = KMeans(i)
        est_2.fit(chemical_abundance_training_set_2)
        y_kmeans_training_2 = est_2.predict(chemical_abundance_training_set_2)

        est_2.fit(chemical_abundance_validation_set_2)
        y_kmeans_validation_2 = est_2.predict(chemical_abundance_validation_set_2)
        score_2.append(accuracy_score(y_kmeans_validation_2, y_kmeans_training_2))
        print("Combination " + str(len(training_set_column_collection_2)) + ": " + str('%.3f'%(((i - 20) / (max_possible_groups_2 - 20)) * 100)) + " % complete"))
    print()
    score_collection_2.append(score_2)

    optimal_number_of_clusters_2 = int(np.where(score_2 == np.max(score_2))[0][0]) + 21
    optimal_number_of_clusters_collection_2.append(optimal_number_of_clu
sters_2)

    highest_accuracy_score_2 = np.max(score_2)
    highest_accuracy_score_collection_2.append(highest_accuracy_score_2)

    # Plot the result for this combination
    fig = plt.figure()
    fig.set_size_inches(18.5, 10.5)
    groups_2 = np.arange(21, max_possible_groups_2 + 1, 1)
    plt.plot(groups_2, score_2, 'o-')
    plt.xlabel('Groups', fontsize = 20)
    plt.xticks(fontsize = 20)
    plt.ylabel('Accuracy Score', fontsize = 20)
    plt.yticks(fontsize = 20)
    plt.vlines(optimal_number_of_clusters_2, 0, highest_accuracy_score_2
, color = 'green', linestyle = 'dashed')
    plt.show()

    print()
    print("Combination " + str(len(training_set_column_collection_2)) +
": ")
    print("The optimal number of clusters is: " + str(optimal_number_of_
clusters_2))
    print("Its corresponding accuracy score is: " + str(highest_accuracy
_score_2))
    print()
```

In [ ]:

```
optimal_number_of_clusters_collection_2 = np.array(optimal_number_of_clu
sters_collection_2)
print("Total combinations tested: " + str(optimal_number_of_clusters_col
lection_2.size))
```

In [ ]:

```
group_21 = len(np.where(optimal_number_of_clusters_collection_2 == 21)[0
])
group_22 = len(np.where(optimal_number_of_clusters_collection_2 == 22)[0
])
group_23 = len(np.where(optimal_number_of_clusters_collection_2 == 23)[0
])
group_24 = len(np.where(optimal_number_of_clusters_collection_2 == 24)[0
])
group_25 = len(np.where(optimal_number_of_clusters_collection_2 == 25)[0
])
group_26 = len(np.where(optimal_number_of_clusters_collection_2 == 26)[0
])
group_27 = len(np.where(optimal_number_of_clusters_collection_2 == 27)[0
])
group_28 = len(np.where(optimal_number_of_clusters_collection_2 == 28)[0
])
group_29 = len(np.where(optimal_number_of_clusters_collection_2 == 29)[0
])
group_30 = len(np.where(optimal_number_of_clusters_collection_2 == 30)[0
])
```

In [ ]:

```
fig = plt.figure()
fig.set_size_inches(18.5, 10.5)
plt.hist(optimal_number_of_clusters_collection_2, bins = np.arange(21, 3
1, 1))
plt.xlabel('Groups', fontsize = 20)
plt.xticks(fontsize = 20)
plt.ylabel('Counts', fontsize = 20)
plt.yticks(fontsize = 20)
plt.show()
```

Below is Fig12: Counts of the optimal number of clusters(highest accuracy score) for all 173 combinations.

title

In [ ]:

```
# Get the mean accuracy score of all tests
highest_accuracy_score_collection_2 = np.array(highest_accuracy_score_co
llection_2)
print(np.mean(highest_accuracy_score_collection_2))
```

In [ ]:

```
fig = plt.figure()
fig.set_size_inches(18.5, 10.5)
plt.hist(highest_accuracy_score_collection_2, bins = np.arange(0, 0.2,
0.01))
plt.xlabel('Accuracy Score', fontsize = 20)
plt.xticks(fontsize = 20)
plt.ylabel('Counts', fontsize = 20)
plt.yticks(fontsize = 20)
plt.show()
```

Below is Fig13: Distribution of the accuracy scores of all 173 combinations. Here, each accuracy score is calculated by using the 10 randomly picked elements' chemical abundance(together with RA and DEC) as the training set and the rest 10 element's chemical abundance(together with RA and DEC) as the validation set.}

title

Below is Fig14: Relation between the accuracy score and the cluster numbers(21 to 30). Here, the accuracy score is calculated by using the 10 randomly picked elements' chemical abundance(together with RA and DEC) as the training set and the rest 10 element's chemical abundance(together with RA and DEC) as the validation set. There are multiple peaks in this plot and the accuracy scores for all cluster numbers are very low(< 0.1).

title

Method III

Now change the training set to contain all chemical abundances, and the validation set to contain the RA, DEC, and radio velocity information

In [ ]:
```
chemical_abundance_training_set_4 = np.zeros((RA.size, 20))
chemical_abundance_validation_set_4 = np.zeros((RA.size, 3))

chemical_abundance_training_set_4 = chemical_abundance
chemical_abundance_validation_set_4[:, 0] = RA
chemical_abundance_validation_set_4[:, 1] = DEC
chemical_abundance_validation_set_4[:, 2] = radio_velocity

score_4 = []
for i in range(21, 31):
    est_4 = KMeans(i)
    est_4.fit(chemical_abundance_training_set_4)
    y_kmeans_training_4 = est_4.predict(chemical_abundance_training_set_
4)

    est_4.fit(chemical_abundance_validation_set_4)
    y_kmeans_validation_4 = est_4.predict(chemical_abundance_validation_
set_4)
    score_4.append(accuracy_score(y_kmeans_validation_4, y_kmeans_traini
ng_4))
    print(str('%.3f'%(((i - 20) / 10) * 100) + " % complete"))

score_4 = np.array(score_4)

optimal_number_of_clusters_4 = int(np.where(score_4 == np.max(score_4))[
0][0]) + 21
highest_accuracy_score_4 = np.max(score_4)

# Plot the result
fig = plt.figure()
fig.set_size_inches(18.5, 10.5)
groups_4 = np.arange(21, 31, 1)
plt.plot(groups_4, score_4, 'o-')
plt.xlabel('Groups', fontsize = 20)
plt.xticks(fontsize = 20)
plt.ylabel('Accuracy Score', fontsize = 20)
plt.yticks(fontsize = 20)
plt.vlines(optimal_number_of_clusters_4, 0, highest_accuracy_score_4, co
lor = 'green', linestyle = 'dashed')
plt.show()

print()
print("The optimal number of clusters is: " + str(optimal_number_of_clus
ters_4))
print("Its corresponding accuracy score is: " + str(highest_accuracy_sco
re_4))
print()
```

Below is Fig15: Relation between the accuracy score and the cluster numbers(21 to 30). Here, the accuracy score is calculated by using all 20 elements' chemical abundance as the training set and the RA, DEC, and radio velocity as the validation set. The peak is very vague and all cluster numbers have an extremely low accuracy score(< 0.06).

title

3. GMMs and UMAP (Nick)

In [15]:
```
# IMPORTS
import pandas as pd
import matplotlib as mpl
from matplotlib.colors import LogNorm

from sklearn.mixture import GaussianMixture
import umap
```

In [29]:

```
# READ IN DATA
master_filename = "Cannon.csv"

master_df = pd.read_csv(master_filename)
master_df = master_df.drop(columns=[master_df.columns[-1]]) # get rid of
last column of NaNs
master_df = master_df.drop(columns=[col for col in master_df.columns if
"ERR" in col])# get rid of err columns (don't care about these)
# get rid of rows with NaNs
master_df = master_df.dropna()

print(master_df.columns)
master_df
```

```
Index(['RA_HRS', 'DEC_DEG', 'TEFF', 'LOGG', 'M_H', 'ALPHA_M', 'FE_H',
'C_H',
       'CI_H', 'N_H', 'O_H', 'NA_H', 'MG_H', 'AL_H', 'SI_H', 'P_H', 'S_
H',
       'K_H', 'CA_H', 'TI_H', 'TIII_H', 'V_H', 'CR_H', 'MN_H', 'CO_H',
'NI_H',
       'RAD_VEL'],
      dtype='object')
```

Out[29]:

	RA_HRS	DEC_DEG	TEFF	LOGG	M_H	ALPHA_M	FE_H	C_H
0	0.000006	74.285389	3727.69779	0.531083	-0.075315	-0.001618	-0.095524	-0.131261
1	0.000189	57.173139	5009.32634	3.328694	-0.111417	-0.001791	-0.116588	-0.204118
2	0.000586	63.463056	4657.17842	2.245739	0.026965	-0.013804	0.053893	-0.195703
4	0.000881	58.360639	3890.75048	0.857215	-0.178921	0.050077	-0.175300	-0.163705
5	0.001239	58.909139	4747.95337	2.377376	-0.001440	-0.025889	0.000664	-0.181968
...	...	...	...	...	...	...	...	...
164066	23.998717	64.715583	4068.17983	1.063511	-0.476503	-0.021310	-0.479274	-0.584773
164067	23.998750	64.049222	4342.72311	1.814609	-0.176498	0.016720	-0.172619	-0.250403
164068	23.998758	63.650556	5044.91507	2.831689	0.137040	-0.056630	0.148526	-0.045335
164069	23.998906	62.853361	4303.95700	1.826523	-0.013265	0.004226	-0.000182	-0.081725
164071	23.999100	79.302000	4306.96552	1.752330	-0.419879	0.202183	-0.387951	-0.344293

138893 rows × 27 columns

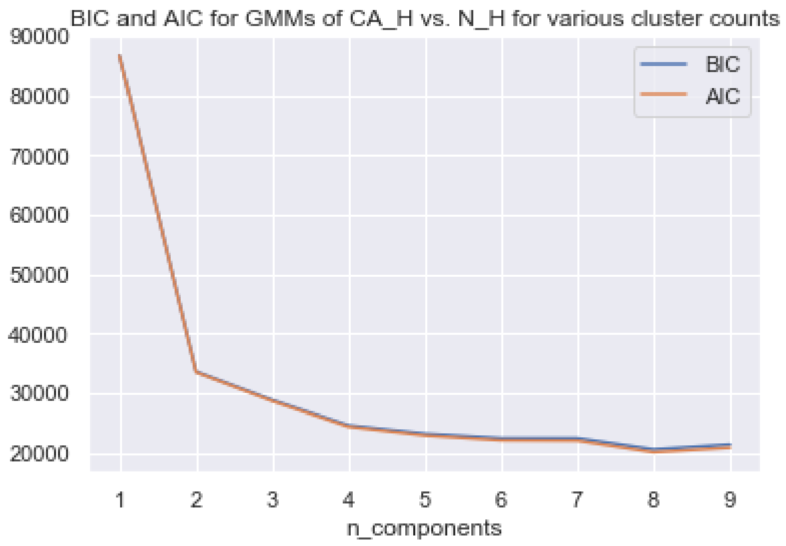
3.1 GMMs

Fig. 3.1.1

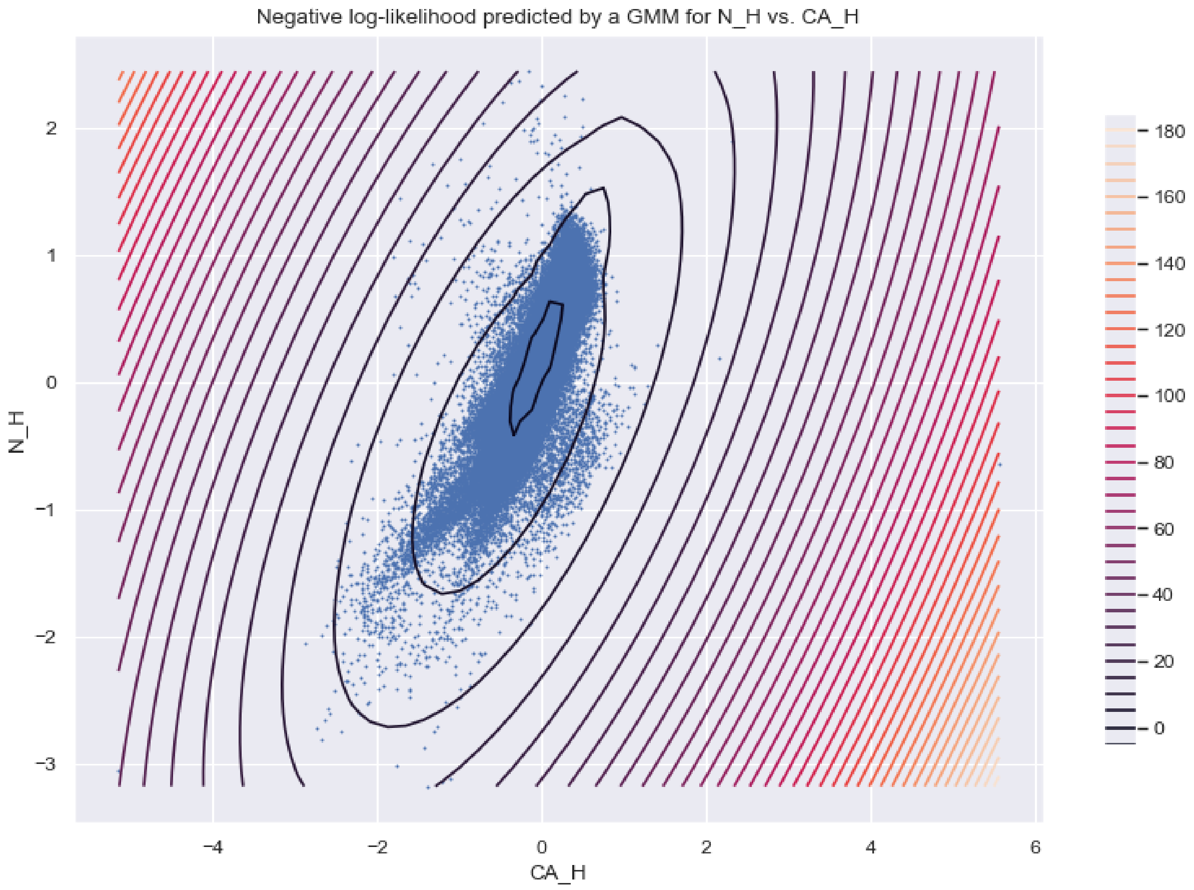
In [38]:

```
# Here, I tried to determine correlations or clustering between individual parameters
# Don't actually run this for everything; it will take forever
# I just have it run for a single set, as an example
example_params = ["CA_H", "N_H"]

for col1 in master_df.columns:
    for col2 in master_df.columns:
        if col1 != col2 and col1 == example_params[0] and col2 == example_params[1]:
            print("Analyzing {} vs {}...".format(col1, col2))

            u = np.column_stack([master_df[col1], master_df[col2]])

#             print(u.shape)

            n_estimators = np.arange(1, 10)
            clfs = [GaussianMixture(n_components=n, covariance_type='full').fit(u) for n in n_estimators]
            bics = [clf.bic(u) for clf in clfs]
            aics = [clf.aic(u) for clf in clfs]

            plt.plot(n_estimators, bics, label='BIC')
            plt.plot(n_estimators, aics, label='AIC')
            plt.title("BIC and AIC for GMMs of {} vs. {} for various cluster counts".format(col1, col2))
            plt.xlabel("n_components")
            plt.legend()
            plt.show()

            best_n = np.argmax(bics[1:]) + 2

            print("optimal n_components = {}".format(best_n))


            clf = GaussianMixture(n_components=best_n, covariance_type='full')
            clf.fit(u)

            # display predicted scores by the model as a contour plot
            x = np.linspace(np.min(u[:,0]), np.max(u[:,0]))
            y = np.linspace(np.min(u[:,1]), np.max(u[:,1]))
            X, Y = np.meshgrid(x, y)
            XX = np.array([X.ravel(), Y.ravel()]).T
            Z = -clf.score_samples(XX)
            Z = Z.reshape(X.shape)

            plt.figure(figsize=(12,8), dpi=80)
            CS = plt.contour(X, Y, Z, levels=42)
            CB = plt.colorbar(CS, shrink=0.8, extend='both')
            plt.scatter(u[:,0], u[:,1], s=0.5)

            plt.title('Negative log-likelihood predicted by a GMM for {} vs. {}'.format(col2, col1))
            plt.axis('tight')
            plt.xlabel(col1)
            plt.ylabel(col2)
            plt.show()
```

```
Analyzing CA_H vs N_H...
```



```
optimal n_components = 2
```



In [32]:

```
# Also attempted something similar with seaborn linear correlation plott
ing,
# again with all possible 2-combinations (again excluded here for brevit
y)
import scipy.stats as stats
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns

for col1 in master_df.columns:
    for col2 in master_df.columns:
        if col1 != col2 and col1 == example_params[0] and col2 == exampl
e_params[1]:
            u = np.column_stack([master_df[col1], master_df[col2]])

            print("Analyzing Correlation of {} vs. {}:".format(col2, col
1))
            sns.set(style="darkgrid", color_codes=True)
            g = sns.jointplot(u[:,0], u[:,1], kind="reg")
            g.annotate(stats.pearsonr)
            plt.show()
```

```
Analyzing Correlation of N_H vs. CA_H:
```

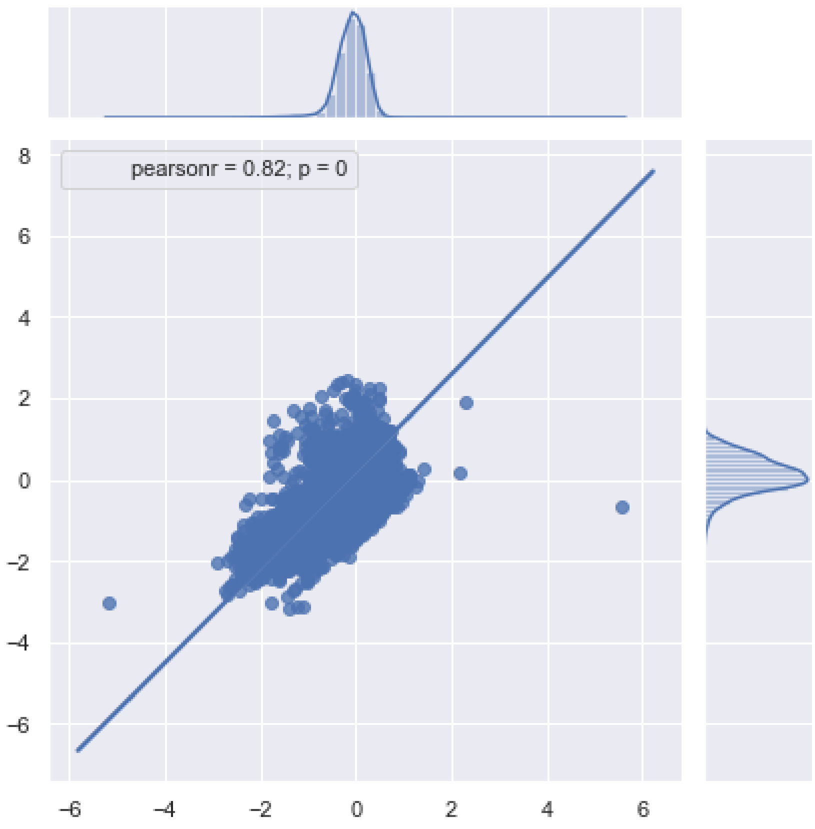


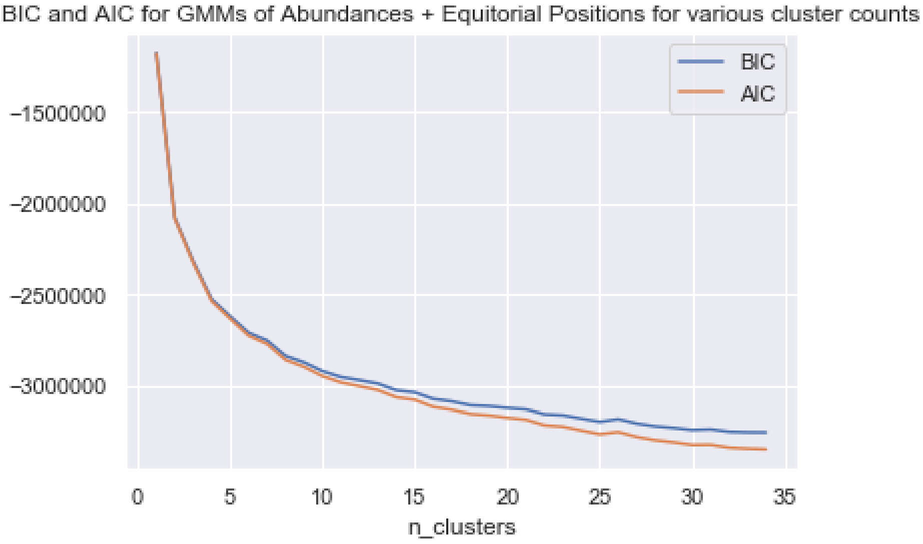
Fig. 3.1.2

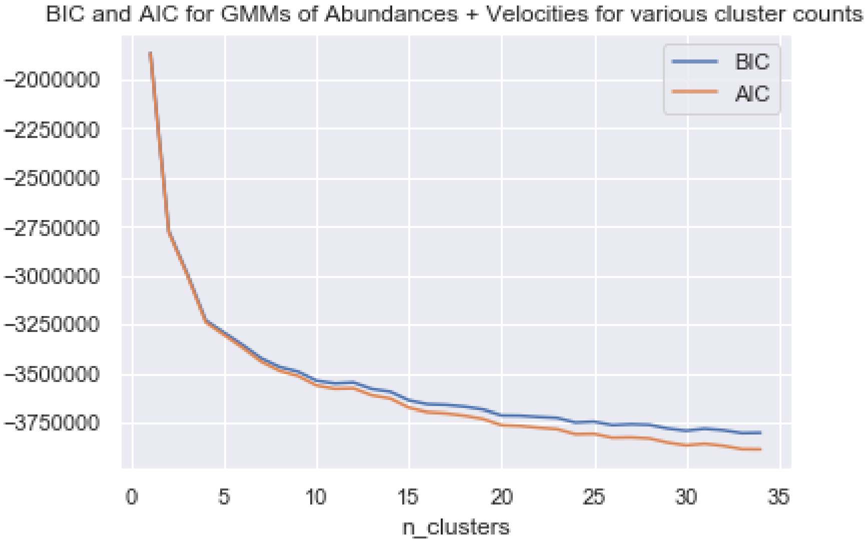
In [33]:

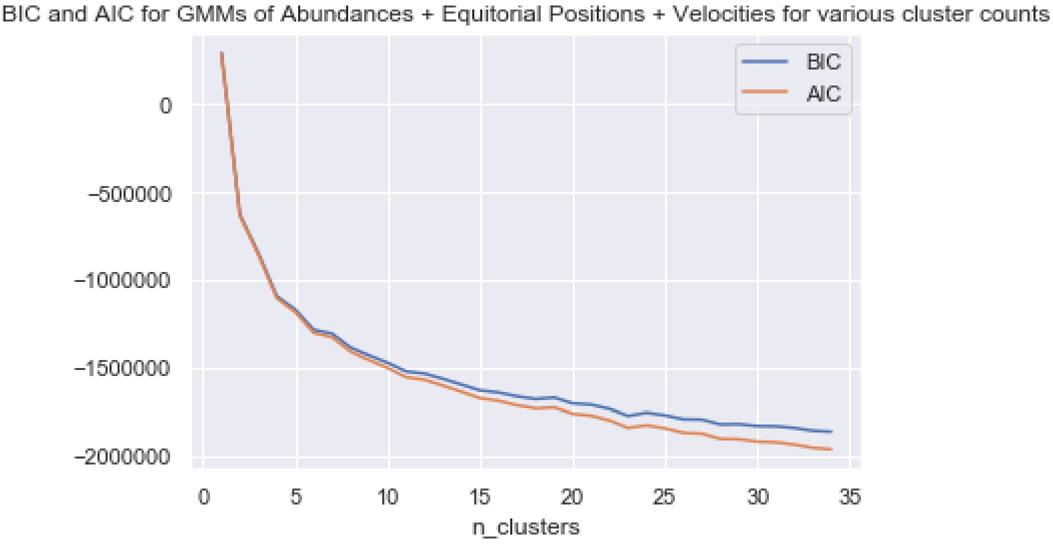
```
# create high-dimensional datasets for (excluding effective temperature, surface gravity,
# and overall abundance measurements as we didn't think they'd be relavant):

badcols_ab_pos = ["TEFF", "LOGG", "M_H", "ALPHA_M", 'RAD_VEL']  # abundances + equitorial positions
badcols_ab_vel = ["TEFF", "LOGG", "M_H", "ALPHA_M", 'RA_HRS', 'DEC_DEG'] # abundances + velocity
badcols_ab_phase = ["TEFF", "LOGG", "M_H", "ALPHA_M"] # abundances + positions + velocity

ab_pos_df = master_df.drop(columns=[col for col in master_df.columns if col in badcols_ab_pos])# get rid of err columns (don't care about these)
ab_vel_df = master_df.drop(columns=[col for col in master_df.columns if col in badcols_ab_vel])# get rid of err columns (don't care about these)
ab_phase_df = master_df.drop(columns=[col for col in master_df.columns if col in badcols_ab_phase])# get rid of err columns (don't care about these)

# get rid of rows with NaNs
ab_pos_df = ab_pos_df.dropna()
ab_vel_df = ab_vel_df.dropna()
ab_phase_df = ab_phase_df.dropna()
```

In [43]:

```
# Run GMMs over 1 - 35 clusters on the above three datasets
# THESE TAKE A WHILE, so we included imgs of the outputs below.

max_clusters = 35
names = {"ab_pos_df": "Abundances + Equitorial Positions",
        "ab_vel_df": "Abundances + Velocities",
         "ab_phase_df": "Abundances + Equitorial Positions + Velocities"
        }
dfs = {"ab_pos_df": ab_pos_df,
        "ab_vel_df": ab_vel_df,
         "ab_phase_df": ab_phase_df
        }

for k in dfs:
    u = dfs[k]
    n_estimators = np.arange(1, max_clusters)
    clfs = [GaussianMixture(n_components=n, covariance_type='full').fit(u) for n in n_estimators]
    bics = [clf.bic(u) for clf in clfs]
    aics = [clf.aic(u) for clf in clfs]

    plt.title("BIC and AIC for GMMs of {} for various cluster counts".format(names[k]))
    plt.plot(n_estimators, bics, label='BIC')
    plt.plot(n_estimators, aics, label='AIC')
    plt.xlabel("n_clusters")
    plt.legend()
    plt.show()
```







Outputs:

1. Abundances + Equitorial Positions:

<div class="container"; width:10px; margin:0 auto;>

</div>

2. Abundances + Velocities:

<div class="container"; width:10px; margin:0 auto;>

</div>

3. Abundances + Equitorial Positions + Velocities:

<div class="container"; width:10px; margin:0 auto;>

</div>

3.2 UMAP

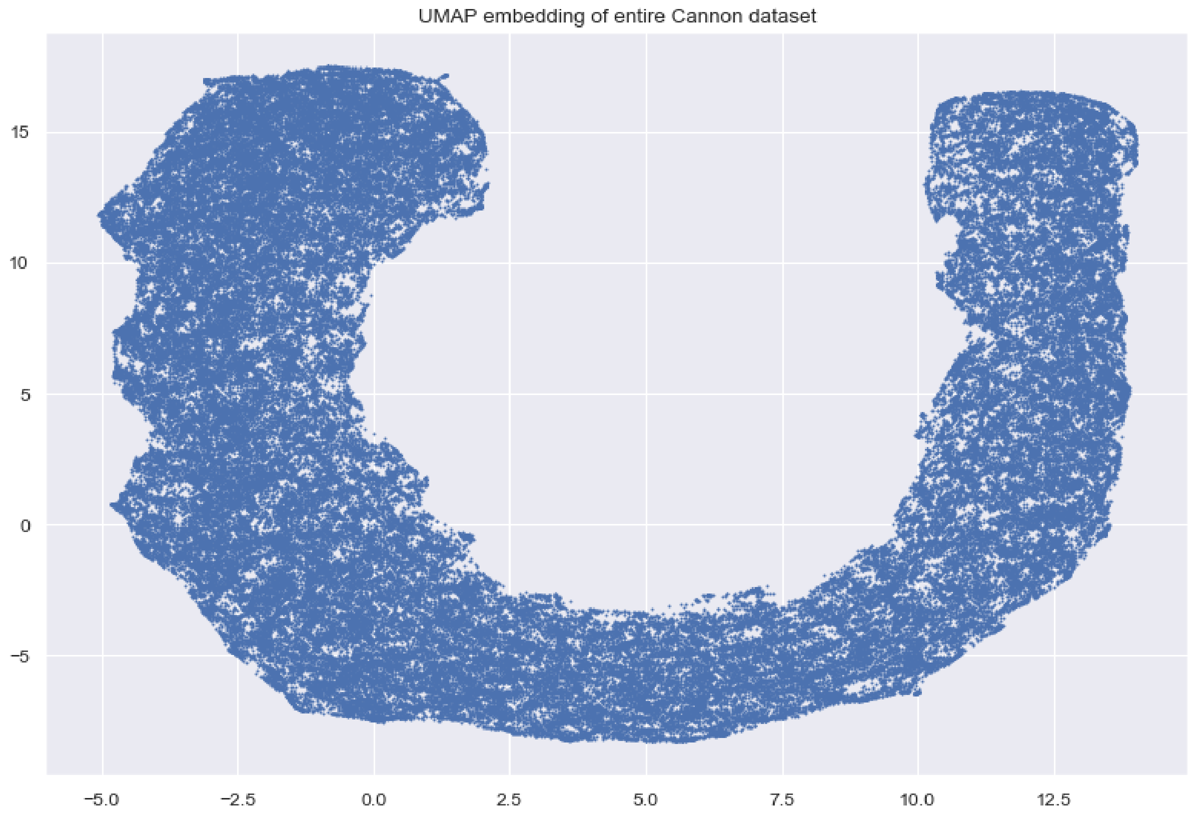
Fig. 3.2.1

```
In [41]: # First, we simply ran UMAP on the entire dataset,
         # with parameters that are best for clustering (creating overall structu
         re):
         # We then ran a GMM on this to attempt to see any
         # Again, this takes a while to run, so we've included the output below
         fit = umap.UMAP(n_neighbors=50, min_dist=0.0, n_components=2)
         %time um = fit.fit_transform(master_df)
```

```
CPU times: user 6min 53s, sys: 12.9 s, total: 7min 6s
Wall time: 2min 20s
```

```
In [42]: plt.figure(figsize=(12,8), dpi=80)
         plt.scatter(um[:,0], um[:,1], s=0.5)
         plt.title('UMAP embedding of entire Cannon dataset')
```

```
Out[42]: Text(0.5, 1.0, 'UMAP embedding of entire Cannon dataset')
```



Output example (note that actual output shape is fairly random, although number of clusters generally isn't):

<div class="container"; width:10px; margin:0 auto;>

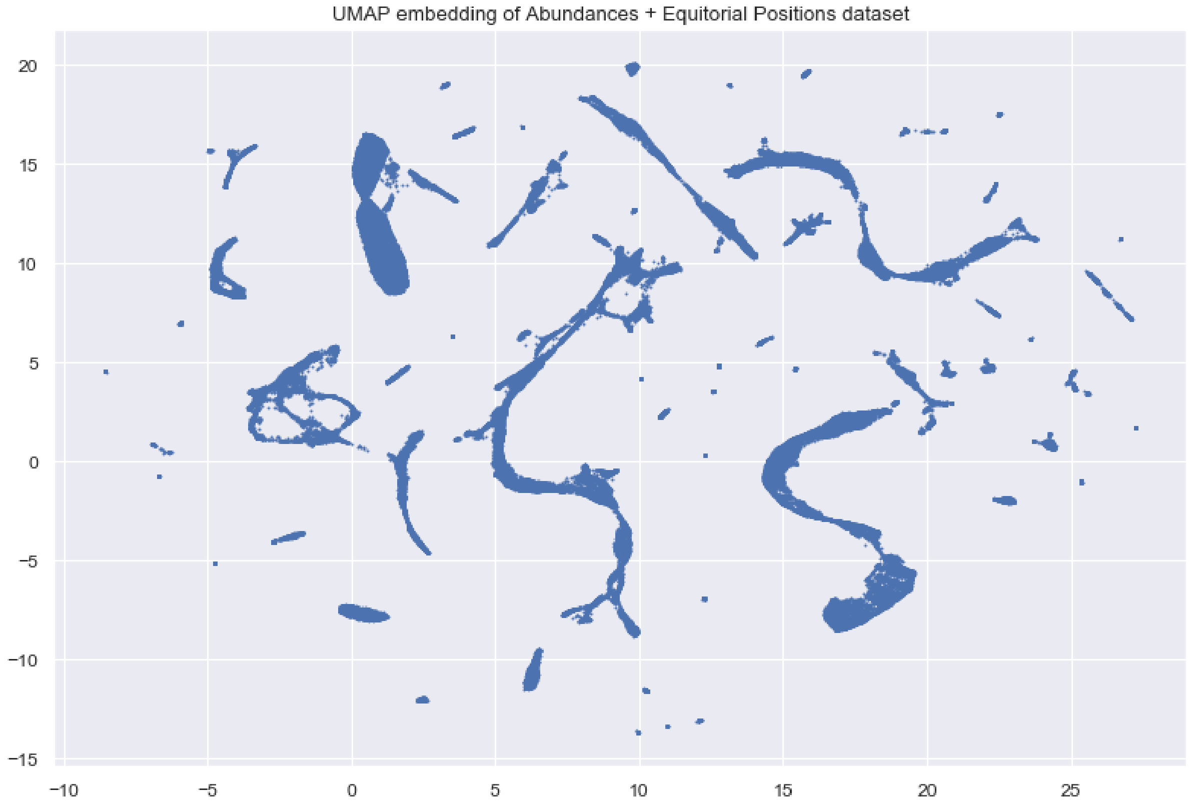
</div>

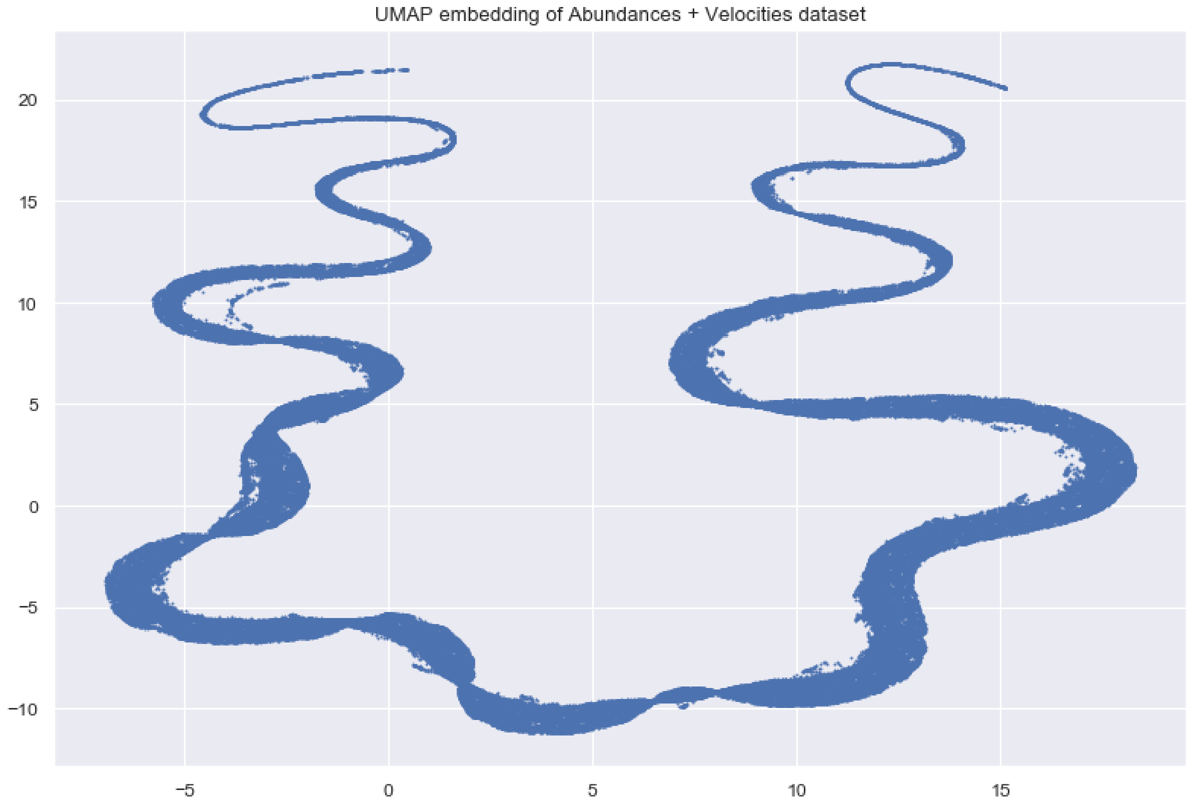
Fig. 3.2.2:

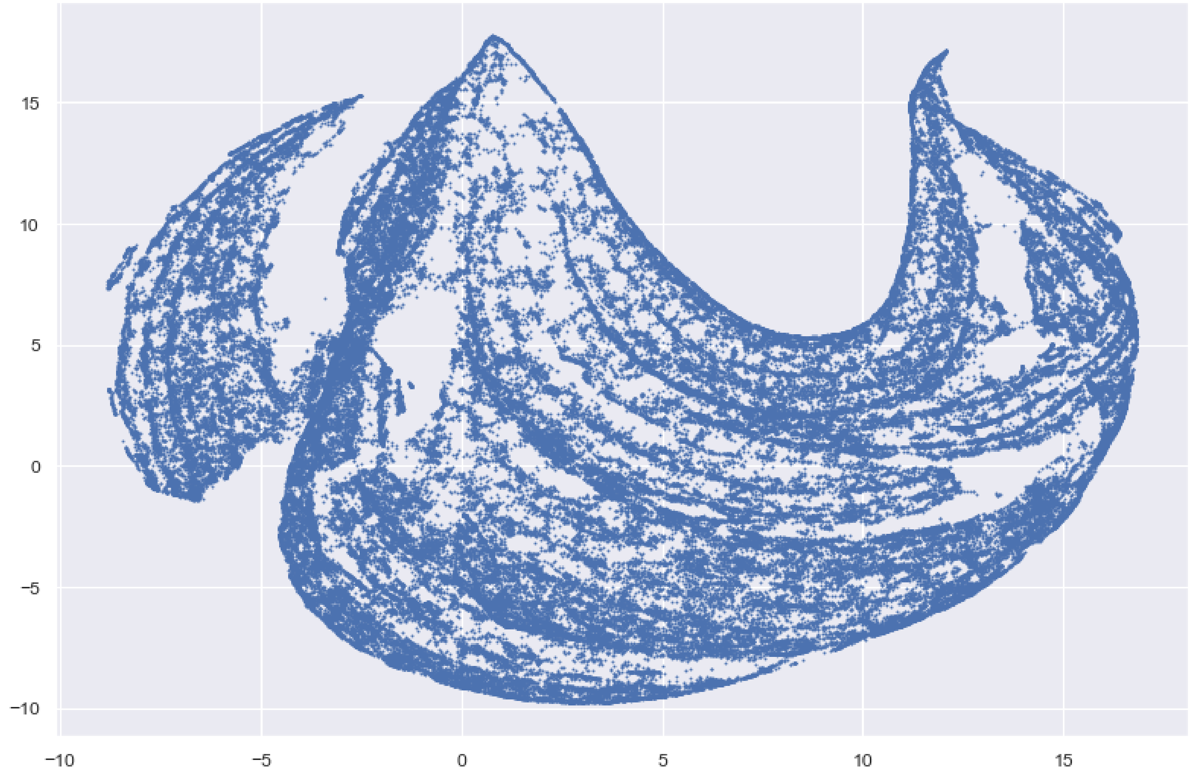
In [44]:

```
# We then attempted the same, but on the same three datasets that the
# GMM analysis was performed on
# Again, these take a while to run, so we've included output images in t
he cell below
for k in dfs:
    df = dfs[k]
    fit = umap.UMAP(n_neighbors=50, min_dist=0.0, n_components=2)
    %time u = fit.fit_transform(df)

    plt.figure(figsize=(12,8), dpi=80)
    plt.scatter(u[:,0], u[:,1], s=0.5)
    plt.title('UMAP embedding of {} dataset'.format(names[k]))
```

```
CPU times: user 10min 34s, sys: 22.7 s, total: 10min 57s
Wall time: 3min 7s
CPU times: user 16min 2s, sys: 39.1 s, total: 16min 41s
Wall time: 4min 36s
CPU times: user 7min 57s, sys: 11.7 s, total: 8min 9s
Wall time: 2min 18s
```






UMAP embedding of Abundances + Equitorial Positions + Velocities dataset

Example Outputs:

<div class="container"; width:10px; margin:0 auto;>

</div>

<div class="container"; width:10px; margin:0 auto;>

</div>

<div class="container"; width:10px; margin:0 auto;>

</div>

4. DBSCAN (Roark)

from Scikit.cluster

I am attempting to apply the `scikit.cluster.DBSCAN` algorithm on the Cannon data set we have parsed and organized, but first I visualize the dataset using 2 feature plots and Nearest Neighbors from Scikit learn

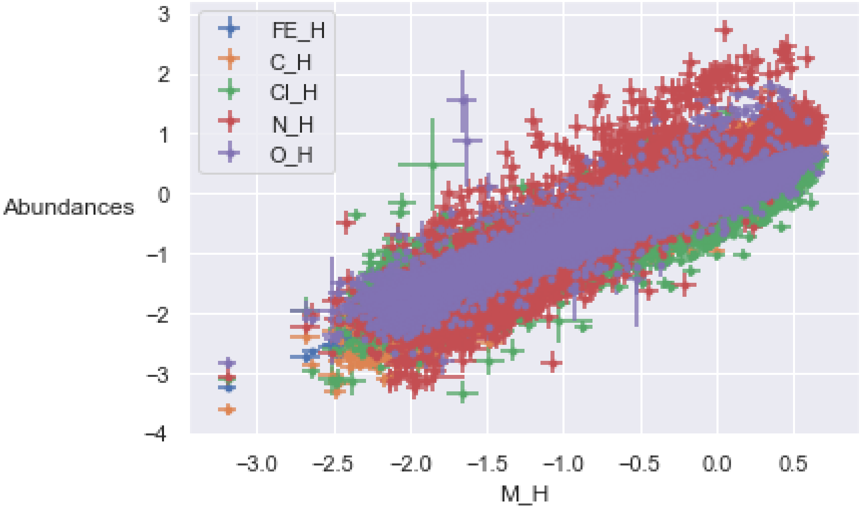
In [70]:

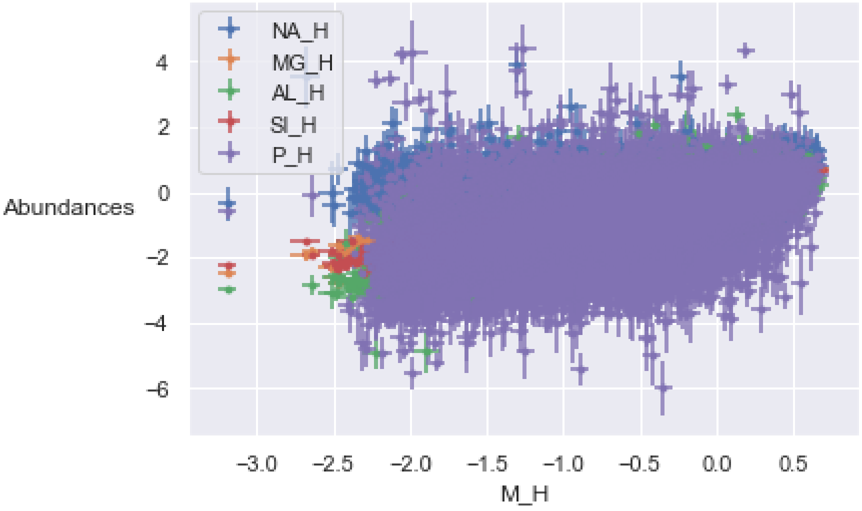
```
import numpy as np
import matplotlib.pyplot as plt

fold = './'
file1 = 'Cannon.csv'

myArr = np.loadtxt(fold+file1,delimiter=',',skiprows=1)
headArr = np.loadtxt(fold+file1,delimiter=',',max_rows=1,dtype=str)
data = {}
for i in range(len(headArr)):
    data[headArr[i]] = myArr[:,i]
print(headArr)
```

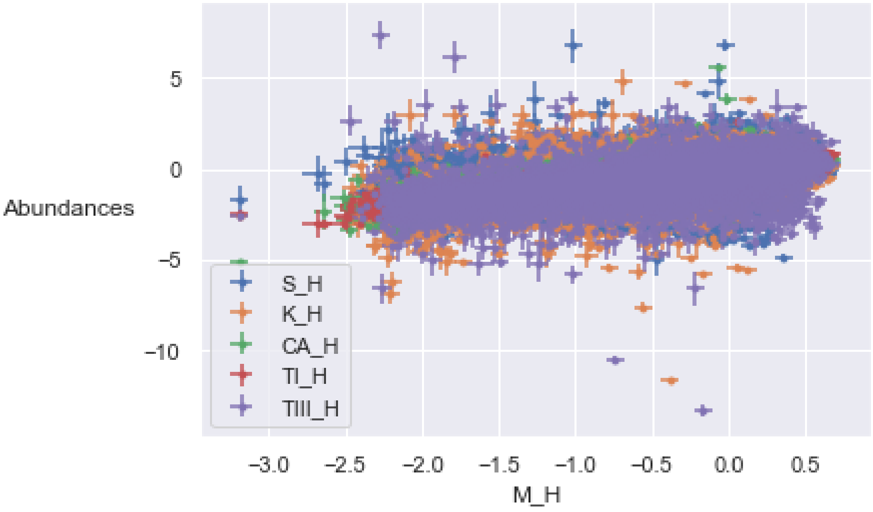
```
['RA_HRS' 'DEC_DEG' 'TEFF' 'LOGG' 'M_H' 'ALPHA_M' 'FE_H' 'C_H' 'CI_H'
 'N_H' 'O_H' 'NA_H' 'MG_H' 'AL_H' 'SI_H' 'P_H' 'S_H' 'K_H' 'CA_H' 'TI_
H'
 'TIII_H' 'V_H' 'CR_H' 'MN_H' 'CO_H' 'NI_H' 'TEFF_ERR' 'LOGG_ERR'
 'M_H_ERR' 'ALPHA_M_ERR' 'FE_H_ERR' 'C_H_ERR' 'CI_H_ERR' 'N_H_ERR'
 'O_H_ERR' 'NA_H_ERR' 'MG_H_ERR' 'AL_H_ERR' 'SI_H_ERR' 'P_H_ERR' 'S_H_E
RR'
 'K_H_ERR' 'CA_H_ERR' 'TI_H_ERR' 'TIII_H_ERR' 'V_H_ERR' 'CR_H_ERR'
 'MN_H_ERR' 'CO_H_ERR' 'NI_H_ERR' 'RAD_VEL' 'RAD_VEL_ERR']
```

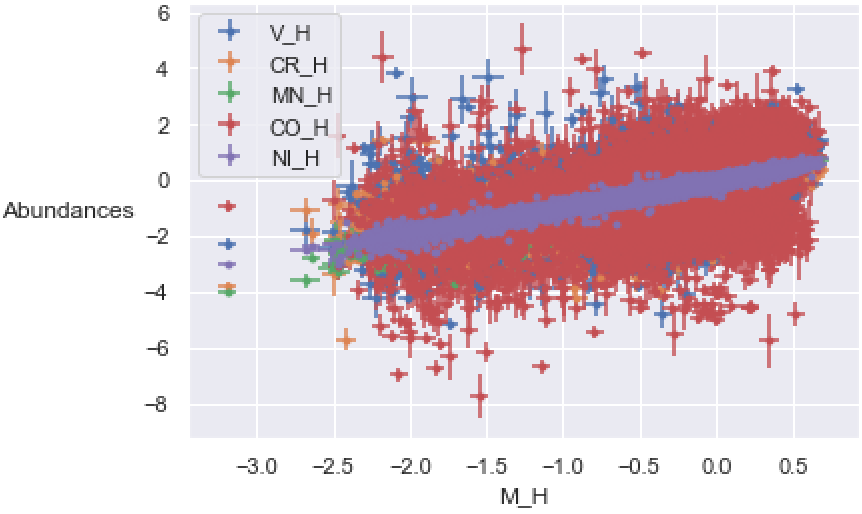
In [71]:

```
#Plot out each abundance vs. entryX, with abunPerPlot abundances on each plot
entryX = 'M_H'
i = 0
abunPerPlot = 5
plt.figure(i)
for entry in headArr[6:26]:
    entryY = entry
    xdata  = data[entryX]
    ydata  = data[entryY]
    xdataE = data[entryX+"_ERR"]
    ydataE = data[entryY+"_ERR"]

    plt.ylabel("Abundances",rotation=0,ha='right')
    plt.xlabel(entryX)

    plt.errorbar(x=xdata,y=ydata,xerr=xdataE,yerr=ydataE,fmt='.',label=entry)
    i += 1
    if i % abunPerPlot == 0:
        plt.legend()
        plt.show()
        if i !=20:
            plt.figure(i/abunPerPlot+1)
```


FE_H
C_H
CI_H
N_H
O_H
Abundances
3
2
1
0
−1
−2
−3
−4
−3.0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
M_H


NA_H
MG_H
AL_H
SI_H
P_H
Abundances
4
2
0
−2
−4
−6
−3.0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
M_H


S_H
K_H
CA_H
TI_H
TIII_H
Abundances
5
0
−5
−10
−3.0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
M_H


V_H
CR_H
MN_H
CO_H
NI_H
Abundances
6
4
2
0
−2
−4
−6
−8
−3.0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
M_H

In [72]:

```
from sklearn.neighbors import NearestNeighbors
from matplotlib import cm


# Get Data we want to run through DBSCAN
testLst = []
testLst.append(data['RA_HRS'])
testLst.append(data['DEC_DEG'])
testLst.append(data['RAD_VEL'])
for entry in headArr[6:26]:
    #print(entry)
    testLst.append(data[entry])


testArr = np.transpose(np.array(testLst))
delLst = []
print(testArr.shape)
for k in range(len(testArr[:,0])):
    for j in range(len(testArr[0,:])):
        if (testArr[k,j] != float(testArr[k,j])):
            delLst.append(k)
testArr = np.delete(testArr,delLst,0)
print(testArr.shape)
```

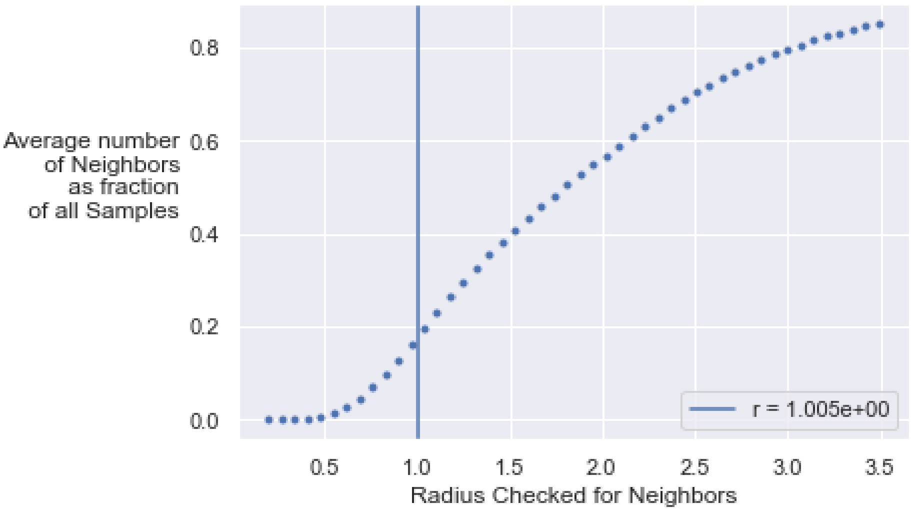
```
(164074, 23)
(138893, 23)
```

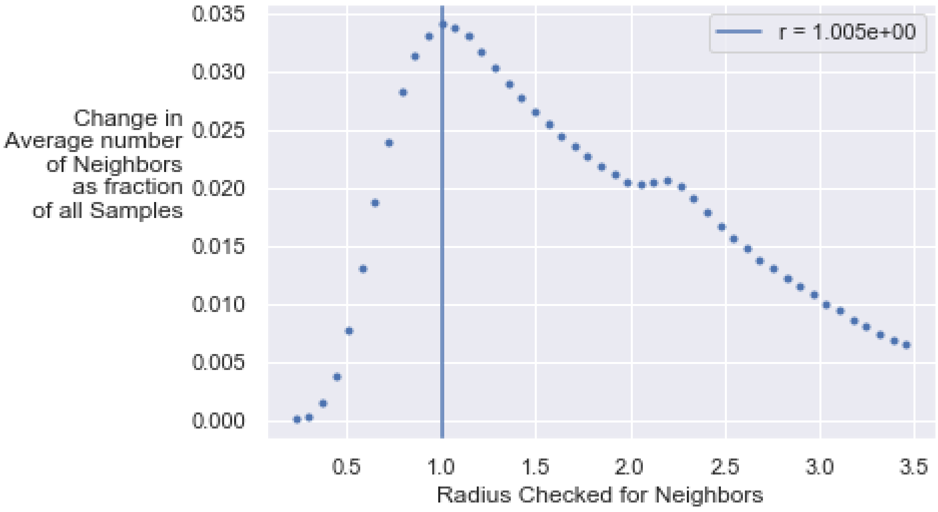
In [73]:

```
#Run Nearest Neighbors to find ideal cluster radius (or eps)
#This takes a while. You can un comment the print statement
                     #to watch the loop tick by with fraction of radii c
omplete
radArr = np.arange(0.2,3.5,0.07)
meanArr = []
skpArr = np.arange(0,len(testArr[:,0]),5000)
for rad in radArr:
    neigh = NearestNeighbors(radius = rad)
    myFit = neigh.fit(testArr[:,3:])

    mySlice = testArr[skpArr,3:]
    dists, inds = neigh.radius_neighbors(X=mySlice)
    numNeighbors = np.zeros(len(inds))
    for i in range(len(inds)):
        numNeighbors[i] = len(inds[i])

    meanArr.append(np.mean(numNeighbors)/len(testArr[:,0]))
    #print(format(len(meanArr)/len(radArr),'1.5f'))
```

In [74]:

```
plt.figure(1)
meanArr = np.array(meanArr)
derivRad = 0.5*(radArr[1:]+radArr[:-1])
deriv = meanArr[1:]-meanArr[:-1]
maxI = np.where(deriv == np.max(deriv))[0]
plt.axvline(x=derivRad[maxI],label="r = "+ format(derivRad[maxI][0],'2.3e'))

plt.plot(radArr,meanArr,'.')
plt.xlabel('Radius Checked for Neighbors')
plt.ylabel('Average number\n of Neighbors\n as fraction\n of all Samples',rotation=0,ha='right')
plt.legend()
plt.figure(2)

plt.plot(derivRad,deriv,'.')

plt.axvline(x=derivRad[maxI],label="r = "+ format(derivRad[maxI][0],'2.3e'))
plt.xlabel('Radius Checked for Neighbors')
plt.ylabel('Change in\n Average number\n of Neighbors\n as fraction\n of all Samples',rotation=0,ha='right')
plt.legend()
plt.show()
```

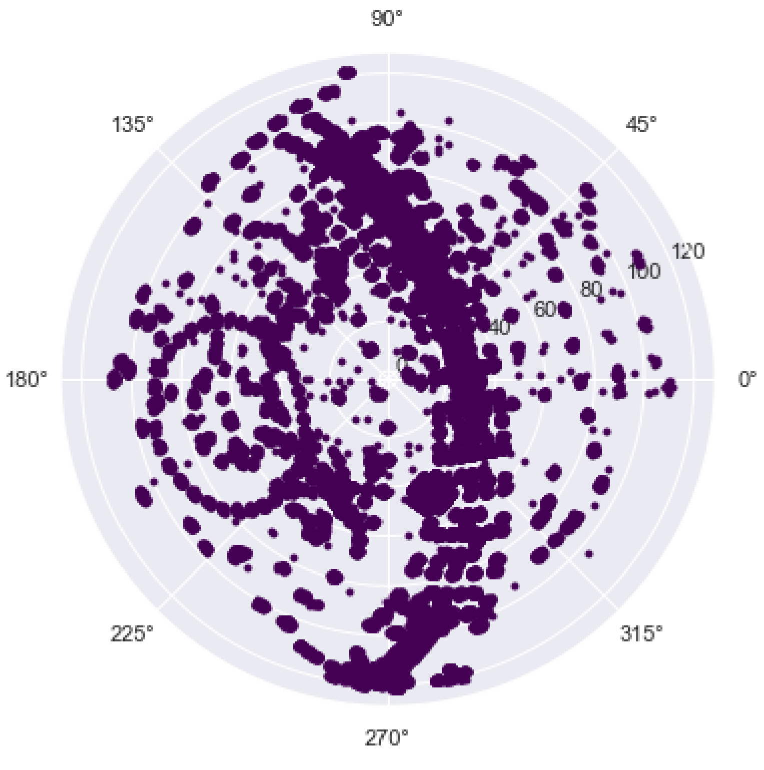




```
In [75]: from sklearn.cluster import DBSCAN
         from matplotlib import cm

         #Run DBSCAN on my eps from above. This code comes from scikit's DBSCAN D
         emo
         #Only run on some of the points... Kernel dies if trying to run all poin
         ts
         skpArr = np.arange(0,len(testArr[:,0]),3)
         e = 1.075
         dbLabels = DBSCAN(eps=e, min_samples=100).fit_predict(testArr[skpArr,3
         :])

         # Number of clusters in labels, ignoring noise if present.
         n_clusters_ = len(set(dbLabels)) - (1 if -1 in dbLabels else 0)
         n_noise_ = list(dbLabels).count(-1)

         print(e,n_clusters_)
```

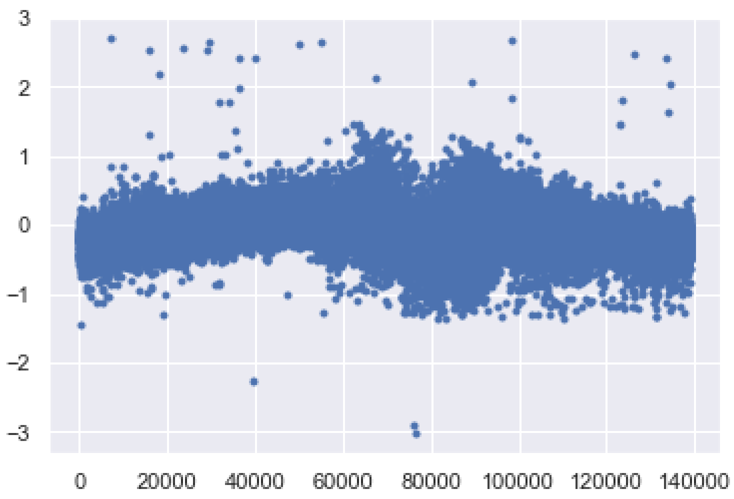
```
1.075 1
```

```
In [77]: #Idea here is to plot nPlt clusters per plot, and color each cluster usi
         ng the viridis map

         myCMap = cm.get_cmap('viridis',n_clusters_)
         nPlt = 5
         #plt.plot(testArr[:,0],testArr[:,1],'b.')
         for n in range(n_clusters_):
             mm = (dbLabels==n)
             th = testArr[skpArr,0][mm]*2*np.pi/24.0
             rad = -testArr[skpArr,1][mm]+90
             fig = plt.figure(1,figsize=[6,6])
             plt.polar(th,rad,'.',color=myCMap((float(n)%nPlt+1)/(nPlt+1)))

             #plt.plot(testArr[:,0][mm],testArr[:,1][mm],'.')

             if (n+1) % nPlt == 0:
                 #plt.xlim([np.min(testArr[:,0]),np.max(testArr[:,0])])
                 #plt.ylim([np.min(testArr[:,1]),np.max(testArr[:,1])])
                 plt.xlabel("(Angular) RA ")
                 plt.ylabel("(Radial) DEC            \n         ",rotation=0,ha='rig
         ht')
                 plt.show()
```



In [78]:

```
##Now Let's try to include radial velocity
maxVel = np.max(np.abs(testArr[:,2]))
testArr2 = testArr.copy()
testArr2[:,2] = testArr[:,2]/(0.33*maxVel)
plt.plot(testArr2[:,2],'.')
plt.show()
```



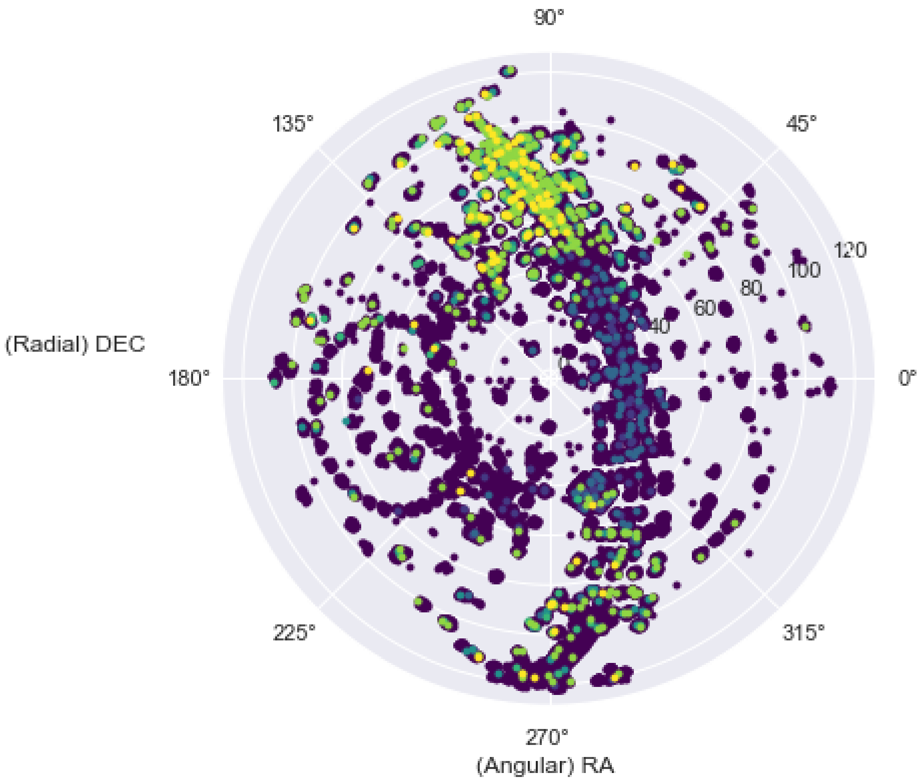
In [79]:

```
from sklearn.cluster import DBSCAN
from matplotlib import cm

#Run DBSCAN on my eps from above. This code comes from scikit's DBSCAN Demo
#Only run on some of the points... Kernel dies if trying to run all points
skpArr = np.arange(0,len(testArr[:,0]),3)

#Keep Same epsilon
e = 1.075
#Run on array with radial velocity values normalized, and only for some of the abundances
dbLabels = DBSCAN(eps=e, min_samples=100).fit_predict(testArr[skpArr,2:])

# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(dbLabels)) - (1 if -1 in dbLabels else 0)
n_noise_ = list(dbLabels).count(-1)

print(e,n_clusters_)
```

```
1.075 7
```

In [80]:

```
#Idea here is to plot nPlt clusters per plot, and color each cluster using the viridis map

myCMap = cm.get_cmap('viridis',n_clusters_)
nPlt = 7
#plt.plot(testArr[:,0],testArr[:,1],'b.')
for n in range(n_clusters_):
    mm = (dbLabels==n)
    th = testArr2[skpArr,0][mm]*2*np.pi/24.0
    rad = -testArr2[skpArr,1][mm]+90
    fig = plt.figure(1,figsize=[6,6])
    plt.polar(th,rad,'.',color=myCMap((float(n)%nPlt+1)/(nPlt+1)))

    #plt.plot(testArr[:,0][mm],testArr[:,1][mm],'.')

    if (n+1) % nPlt == 0:
        #plt.xlim([np.min(testArr[:,0]),np.max(testArr[:,0])])
        #plt.ylim([np.min(testArr[:,1]),np.max(testArr[:,1])])
        plt.xlabel("(Angular) RA ")
        plt.ylabel("(Radial) DEC          \n        ",rotation=0,ha='right')
        plt.show()
```



In [ ]: