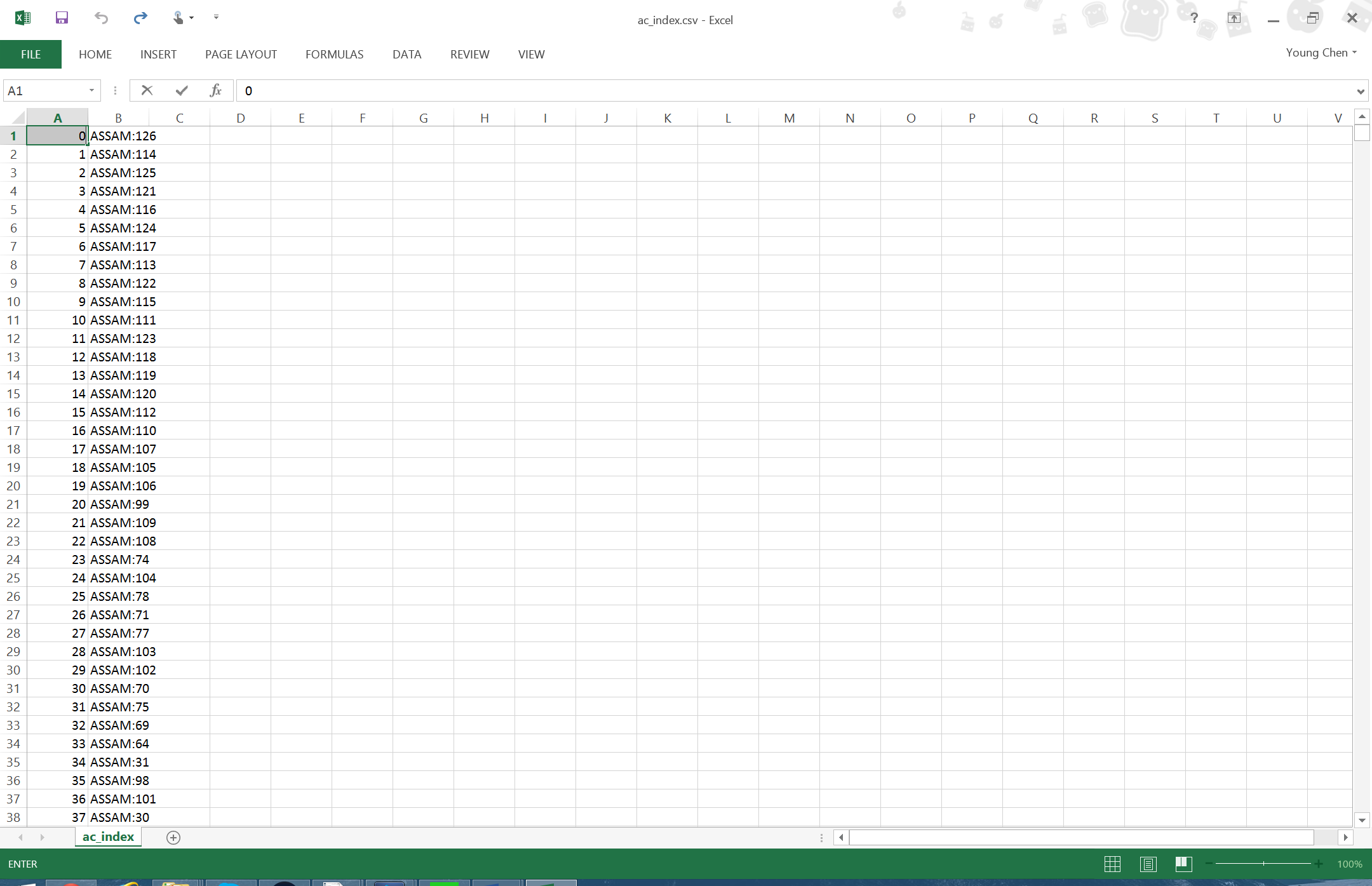
Start: 5:50 p.m.

Ac\_index generating code finished. Output:



\*This is the order of the polygons in the shape file. By mapping the order of the data to the shape file we can eliminate searching for the data, speeding up the animation significantly, as searches are computationally rather expensive.

Code:

1. **import** shapefile
2. **import** re
3. **import** csv
4. with open("C:/Users/Young/Documents/GitHub/Research-2015-Energy-Data/data/generated/ac\_index.csv", 'w', newline='') as ac\_index:
5. ac\_index\_writer = csv.writer(ac\_index, delimiter=',',
6. quotechar='|', quoting=csv.QUOTE\_MINIMAL)
7. pattern = re.compile("([a-zA-Z& ]+)([0-9]+)") #regular expression matching to split state from AC ID
8. sf = shapefile.Reader("C:/Users/Young/Documents/Github/Research-2015-Energy-Data/data/AC\_India\_Geo/AC\_ALLINDIA")
9. **print**("Record format:", sf.fields)
10. unique\_id = 0;
11. **for** r **in** sf.iterRecords():
12. result = pattern.match(r[4])
13. **if** result:
14. stateid = pattern.match(r[4]).groups()
15. #print("INDEX:", unique\_id, "STATE:", stateid[0], "ID:", stateid[1]);
16. ac\_index\_writer.writerow([unique\_id, stateid[0] + ':' + stateid[1]]);
17. unique\_id += 1
18. **else**:
19. **print**("ERROR:", r[4], "could not be matched");
20. exit();

Break: 6:20 p.m.

Resume 9:20 p.m.

Reading in the original data and preparing to format it. This will consist of setting up a two dimensional array indexed via [unique\_id][month].

Choosing data structures

One option is to use an ordered dictionary as the top level, where they key is the state:id, and the value is a 2d list indexed via year/month to get the vis value. However, since there are so many state:id pairs, the memory efficiency of this approach is questionable (up to 2n, but we’re going to end up with a very large n). Another difficulty is iterating and printing in the correct order. O(n) space and O(1) lookup time is very promising though.

We have an array of our state:id’s in the correct order. Using a dict will be much faster when building the formatted data, as we won’t have to keep searching for the state:id that we want to insert at. Printing will be okay as we can just iterate off of our ac\_array and retrieve the value we’re looking for in O(1) time.

Current code:

1. **import** shapefile
2. **import** re
3. **import** csv
4. **from** itertools **import** repeat
5. #start and end year defined in case we ever need to change it (assumed to be inclusive)
6. start\_year = 1993;
7. end\_year = 2013;
8. ac\_data = {};
9. ac\_array = [];
10. #Open the csv we're going to write to
11. with open("C:/Users/Young/Documents/GitHub/Research-2015-Energy-Data/data/generated/AC\_India\_Index.csv", 'w', newline='') as ac\_index:
12. #our csv writer object
13. ac\_index\_writer = csv.writer(ac\_index, delimiter=',',
14. quotechar='|', quoting=csv.QUOTE\_MINIMAL)
15. #regular expression matching to split state from AC ID
16. pattern = re.compile("([a-zA-Z& ]+)([0-9]+)")
17. #our shapefile reader
18. sf = shapefile.Reader("C:/Users/Young/Documents/Github/Research-2015-Energy-Data/data/AC\_India\_Geo/AC\_ALLINDIA")
19. #print the record format we've found for easy debugging
20. **print**("Record format:", sf.fields)
21. #start our easy unique id counter at 0
22. unique\_id = 0;
23. #iterate through the records, and parse stateid into state:id.
24. #write it into our csv file output, and add it to our ac\_array
25. #since we're going to use it later to reformat the data
26. **for** r **in** sf.iterRecords():
27. result = pattern.match(r[4])
28. **if** result:
29. stateid = pattern.match(r[4]).groups()
30. #print("INDEX:", unique\_id, "STATE:", stateid[0], "ID:", stateid[1]);
31. ac\_array.append(stateid[0] + ':' + stateid[1]);
32. ac\_index\_writer.writerow([unique\_id, stateid[0] + ':' + stateid[1]]);
33. unique\_id += 1
34. **else**:
35. **print**("ERROR:", r[4], "could not be matched");
36. exit();
37. #print(ac\_array);
38. #define indexing functions
39. **def** date\_to\_index(year, month):
40. **return** (year-start\_year)\*12+month-1
41. **def** index\_to\_date(index):
42. **return** [index//12+start\_year, index%12+1]
43. #open the data we're going to read, and the data we're going to write
44. with  open("C:/Users/Young/Documents/GitHub/Research-2015-Energy-Data/data/AC\_India\_Data/AC\_India\_Data.csv", newline='') as ac\_data\_in:
45. **print**("Data in opened")
46. #set up reader object
47. datareader = csv.DictReader(ac\_data\_in)
48. #set up our ac\_data (initialize an empty array in each of the valid state:id's)
49. **for** ac\_id **in** ac\_array:
50. # data is a [21][12] list, indexed via [year since 1993][month]
51. data = [[-1.0 **in** range(12)] **for** x **in** range(end\_year-start\_year+1)]
52. ac\_data[ac\_id] = data
53. #start reading in the data!
54. **for** row **in** datareader:
55. row\_key = row["State"] + ':' + row["Constituency"]
56. **if** row\_key **in** ac\_data:
57. ac\_data[row\_key][int(row["Year"])][int(row["Month"])] = float(row["NCS\_20K\_Pred\_Month"])
58. **else**:
59. **print**("Formtted key:", row\_key, "not found")
61. with open("C:/Users/Young/Documents/GitHub/Research-2015-Energy-Data/data/generated/AC\_India\_Data\_Formatted.csv", 'w', newline='') as ac\_data\_out:
62. #set up writer object
63. fieldnames = ["Year", "Month"]
64. fieldnames.extend(ac\_array)
65. #print("Writing CSV with header:", fieldnames)
66. datawriter = csv.DictWriter(ac\_data\_out, fieldnames = fieldnames, lineterminator = '\n')

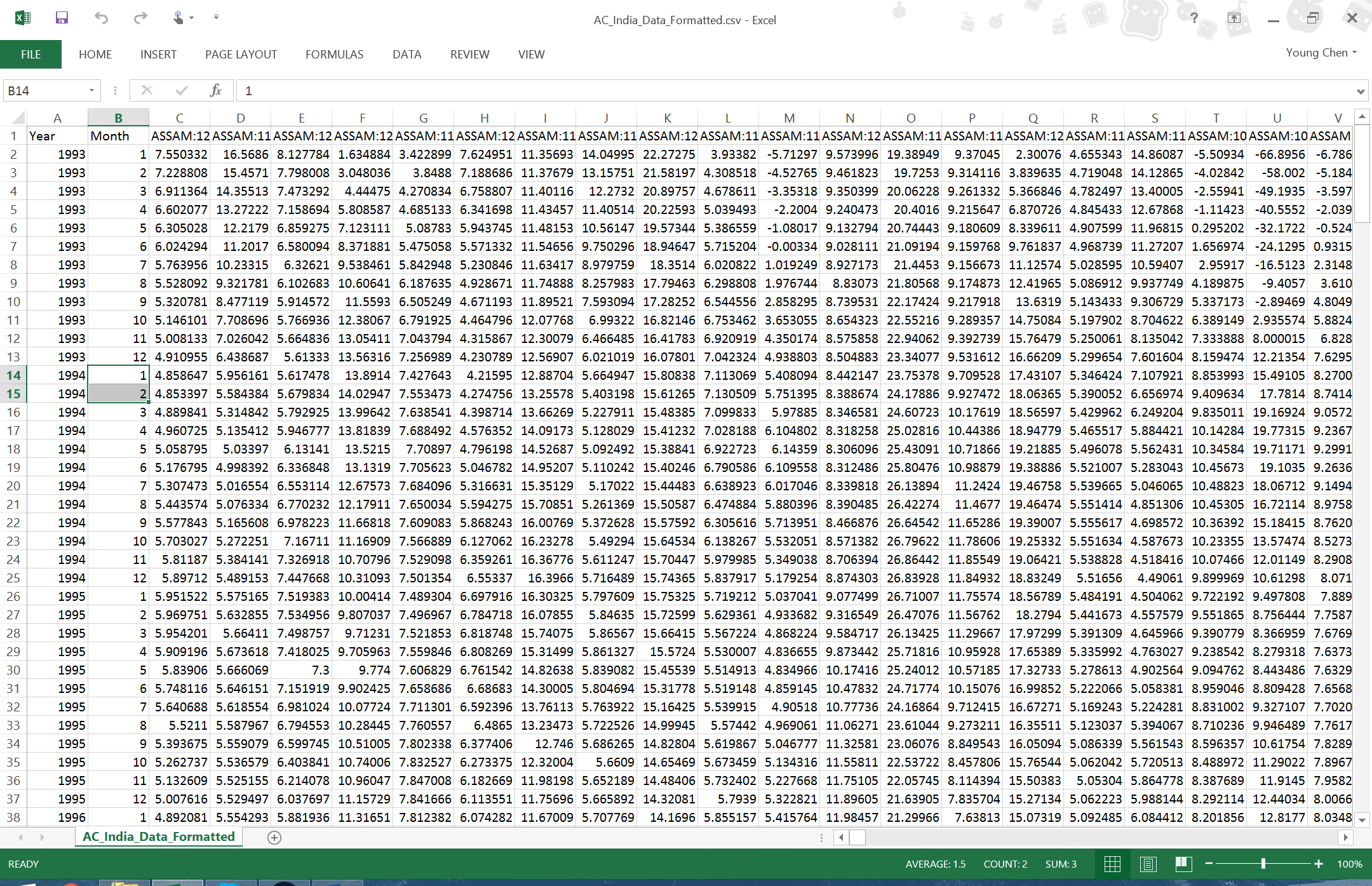
Script is reporting that almost every state:id pair is mismatched. Upon closer inspection, it’s definitely due to the state names having different case in the shape file vs in the data. I’m going to format everything to upper case for consistency

Break: 11:10 p.m.

Resume: 12:40 a.m.

Note: exception occurred. Turns out there was data after 2013 (unexpected!)

Script finalize and finished. Formatted data:



Finished code:

1. **import** shapefile
2. **import** re
3. **import** csv
4. #start and end year defined in case we ever need to change it (assumed to be inclusive)
5. start\_year = 1993;
6. end\_year = 2014;
7. ac\_data = {};
8. ac\_array = [];
9. #Open the csv we're going to write to
10. with open("C:/Users/Young/Documents/GitHub/Research-2015-Energy-Data/data/generated/AC\_India\_Index.csv", 'w', newline='') as ac\_index:
11. #our csv writer object
12. ac\_index\_writer = csv.writer(ac\_index, delimiter=',',
13. quotechar='|', quoting=csv.QUOTE\_MINIMAL)
14. #regular expression matching to split state from AC ID
15. pattern = re.compile("([a-zA-Z& ]+)([0-9]+)")
16. #our shapefile reader
17. sf = shapefile.Reader("C:/Users/Young/Documents/Github/Research-2015-Energy-Data/data/AC\_India\_Geo/AC\_ALLINDIA")
18. #print the record format we've found for easy debugging
19. **print**("Record format:", sf.fields)
20. #start our easy unique id counter at 0
21. unique\_id = 0;
22. #iterate through the records, and parse stateid into state:id.
23. #write it into our csv file output, and add it to our ac\_array
24. #since we're going to use it later to reformat the data
25. **for** r **in** sf.iterRecords():
26. result = pattern.match(r[4])
27. **if** result:
28. stateid = pattern.match(r[4]).groups()
29. #print("INDEX:", unique\_id, "STATE:", stateid[0], "ID:", stateid[1]);
30. ac\_array.append((stateid[0] + ':' + stateid[1]).upper());
31. ac\_index\_writer.writerow([unique\_id, (stateid[0] + ':' + stateid[1]).upper()])
32. unique\_id += 1
33. **else**:
34. **print**("ERROR:", r[4], "could not be matched");
35. exit();
36. #print(ac\_array);
37. #define indexing functions
38. **def** date\_to\_index(year, month):
39. **return** (year-start\_year)\*12+month-1
40. **def** index\_to\_date(index):
41. **return** [index//12+start\_year, index%12+1]
42. #open the data we're going to read, and the data we're going to write
43. with  open("C:/Users/Young/Documents/GitHub/Research-2015-Energy-Data/data/AC\_India\_Data/AC\_India\_Data.csv", newline='') as ac\_data\_in:
44. **print**("Data in opened")
45. #set up reader object
46. datareader = csv.DictReader(ac\_data\_in)
47. #set up our ac\_data (initialize an empty array in each of the valid state:id's)
48. **for** ac\_id **in** ac\_array:
49. # data is a [21][12] list, indexed via [year since 1993][month]
50. data = [ [-1.0]\*12 **for** x **in** range(end\_year-start\_year+1)]
51. ac\_data[ac\_id] = data
52. #start reading in the data!
53. **for** row **in** datareader:
54. row\_key = (row["State"] + ':' + row["Constituency"]).upper()
55. **if** row\_key **in** ac\_data:
56. #print("Reading key", row\_key)
57. **try**:
58. ac\_data[row\_key][int(row["Year"])-start\_year][int(row["Month"])-1] = float(row["NCS\_20K\_Pred\_Month"])
59. **except** IndexError:
60. **print**("ERROR: year", int(row["Year"])-start\_year, "month", int(row["Month"])-1, "out of bounds with ac\_data entry size", len(ac\_data[row\_key]), "by", len(ac\_data[row\_key][0]))
61. exit()
62. #else:
63. #print("Formatted key", row\_key, "not found")
65. with open("C:/Users/Young/Documents/GitHub/Research-2015-Energy-Data/data/generated/AC\_India\_Data\_Formatted.csv", 'w', newline='') as ac\_data\_out:
66. #set up writer object
67. fieldnames = ["Year", "Month"]
68. fieldnames.extend(ac\_array)
69. #print("Writing CSV with header:", fieldnames)
70. datawriter = csv.DictWriter(ac\_data\_out, fieldnames = fieldnames, lineterminator = '\n')
71. datawriter.writeheader()
72. **for** year **in** range(start\_year, end\_year+1):
73. **for** month **in** range(12):
74. row = {"Year":year, "Month":month+1}
75. **for** ac\_id **in** ac\_array:
76. **try**:
77. row[ac\_id] = ac\_data[ac\_id][year-start\_year][month]
78. **except** IndexError:
79. **print**("ERROR: year", int(row["Year"])-start\_year, "month", int(row["Month"])-1, "out of bounds with ac\_data entry size", len(ac\_data[row\_key]), "by", len(ac\_data[row\_key][0]))
80. exit()
81. **print**("Writing row for", month+1, "/", year)
82. datawriter.writerow(row)
83. print(“Finished”)

Finished: 1:20 a.m.

Total: 3 hours