Rajeev_baby_names_bystate

February 10, 2018

1 Pandas, Matplotlib, and Basemap: Coloring States by Baby Name Uniqueness

1.1 Here we will use Matplotlib and Basemap to visualize the degree of uniqueness each state has in giving baby names.

```
In [1]: import math
        import csv
        import pandas as pd
        import os
        import numpy as np
        import matplotlib as mpl
        import matplotlib.pyplot as plt
        from mpl_toolkits.basemap import Basemap
        from matplotlib.colors import rgb2hex
        from matplotlib.patches import Polygon
```

1.1.1 Here we start with a file called "namebystate" that contains a text file of baby names for each state. We will use the chdir() method in the os module to change to the directory with the names data and print out all the files with '.TXT' extensions in the directory.

```
In [2]: data_path = "rajeev_data\states"
        current_dir = os.chdir(data_path)

        for file in os.listdir(current_dir):
            if file.endswith('.TXT'):
                print(file)
```

```
AK.TXT
AL.TXT
AR.TXT
AZ.TXT
CA.TXT
CO.TXT
CT.TXT
DC.TXT
DE.TXT
```

```
FL.TXT
GA.TXT
HI.TXT
IA.TXT
ID.TXT
IL.TXT
IN.TXT
KS.TXT
KY.TXT
LA.TXT
MA.TXT
MD.TXT
ME.TXT
MI.TXT
MN.TXT
MO.TXT
MS.TXT
MT.TXT
NC.TXT
ND.TXT
NE.TXT
NH.TXT
NJ.TXT
NM.TXT
NV.TXT
NY.TXT
OH.TXT
OK.TXT
OR.TXT
PA.TXT
RI.TXT
SC.TXT
SD.TXT
TN.TXT
TX.TXT
UT.TXT
VA.TXT
VT.TXT
WA.TXT
WI.TXT
WV.TXT
WY.TXT
```

1.1.2 We can then use the enumerate( ) function to see the format for each file. Enumerate returns a tuple with the first element being the index and the second number being the line. First line will have the index of one instead of zero

```
In [3]: for lineno, line in enumerate(open('CA.TXT', 'r'), start=1):
            if lineno < 5:
                print(line)
```

```
CA,F,1910,Mary,295

CA,F,1910,Helen,239

CA,F,1910,Dorothy,220

CA,F,1910,Margaret,163
```

1.1.3 Each text file is a comma delimited file with the following fields; state, gender, date, name, counts. Now that we know what we are dealing with we can get started. Let's use the walk( ) function of the os module to iterate over the list of files in the directory.

```
In [4]: current_dir = os.getcwd()
        file_list = []
        for root, dirs, files in os.walk(current_dir):
                for filename in files:
                    if filename.endswith('.TXT'):
                        file_list.append(filename)
        print(file_list)
        print(len(file_list))
```

```
['AK.TXT', 'AL.TXT', 'AR.TXT', 'AZ.TXT', 'CA.TXT', 'CO.TXT', 'CT.TXT', 'DC.TXT', 'DE.TXT', 'FL
51
```

1.1.4 Now we have a list containing all of the .TXT files in our data directory. The length of the list is 51 corresponding to the 50 states plus the District of Columbia.

Next let's make a Pandas data frame for California in 2016. It will have columns for name and counts for 2016. We will then apply this to all 'TXT' files. We create a file object with open( ). We initialize an empty list (name_list) outside of the loop. We will add info to this list as we go through the state files. We then use a for-loop to read each line as a string. We strip the string of new lines and spaces using the strip( ) method and convert the string to a list delimited by commas using the split( ) method of lists. The comma is the default delimiter but I like to specify it explicitly. Next, we only take the data from the females for the year of 2016 using indexing and Booleans.

Inside of the loop we make a temporary list called 'temp_list' to store the counts (temp_list.append(int(line[4]))), the name (temp_list.append(line[3])) and the state (temp_list.append(line[0])).

Finally, we can append the temp_list to the name_list to create a list of lists. Later we will want to sort our pandas data frame on births so we convert the elements in the births list from strings to integers using a list comprehension, births = [int(element) for element in births]. We then sort the lists in reverse so that they are in descending rather ascending order. By default Python sorts on the first element of each list. We then create a new "name_list_top10" which contains only the top10 names for California in 2016.

```
In [5]: name_list = []
        for line in open('CA.TXT', 'r'):
            line = line.strip().split(',')
            if line[1] == 'F' and line[2] == '2013':
                temp_list = []
                temp_list.append(int(line[4]))
                temp_list.append(line[3])
                temp_list.append(line[0])
                name_list.append(temp_list)
        name_list.sort(reverse=True)
        name_list_top10 = name_list[0:10]
        #close('CA.TXT')
        print(name_list_top10)
```

```
[[3460, 'Sophia', 'CA'], [2792, 'Isabella', 'CA'], [2599, 'Mia', 'CA'], [2488, 'Emma', 'CA'],
```

1.1.5 Creating a Pandas data frame is very easy. We just pass the list of lists and give it the column headings.

```
In [6]: name_df = pd.DataFrame(name_list_top10, columns=['births', 'name', 'state'])
        name_df.head()
```

Out[6]:

	births	name	state
0	3460	Sophia	CA
1	2792	Isabella	CA
2	2599	Mia	CA
3	2488	Emma	CA
4	2292	Emily	CA

We have made a Pandas data frame from one file ('CA.txt'). We can now apply this approach to all files in the file_list we created earlier. First we initialize an empty Pandas data frame. Then we iterate through the file list and repeat what we did with the California above. Except that we will append a new data frame for each state to the growing data frame called "df". Each Pandas data frame called "data" will be overwritten when with every new state file but that is o.k. because we are constantly adding this data to the growing data frame "df". Since we initialize it outside of the for-loop we won't overwrite it.

```
In [7]: df = pd.DataFrame()
        for file in file_list:
            name_list2 = []
            for line in open(file, 'r'):
```

```
            line = line.strip().split(',')
            if line[1] == 'F' and line[2] == '2016':
                temp_list2 = []
                temp_list2.append(int(line[4]))
                temp_list2.append(line[3])
                temp_list2.append(line[0])
                name_list2.append(temp_list2)
        name_list2.sort(reverse=True)
        name_list_top10 = name_list2[0:10]
        data = pd.DataFrame(name_list_top10, columns = ['Births', 'Name', 'State'])
        df = df.append(data)
    df.head()
```

```
Out[7]:    Births       Name State
        0      47       Emma    AK
        1      45     Olivia    AK
        2      34  Charlotte    AK
        3      34     Amelia    AK
        4      33     Sophia    AK
```

You might notice some similarities to appending lists and appending a Pandas data frame. But there is one important difference. Python lists can be appended in place. This means you can initialize the empty list outside of the loop and build the list by going through the loop. Pandas Data Frames cannot be appended in place. Instead you have to store the output.

```
In [8]: df_example = pd.DataFrame()
        data2 = pd.DataFrame([['A', 'B'],[1, 2]])
        df_example.append(data2)
```

```
Out[8]:    0  1
        0  A  B
        1  1  2
```

```
In [9]: list = [1]
        df_example = pd.DataFrame()
        for i in list:
            data2 = pd.DataFrame([['A', 'B'],[1,2]])
            df_example.append(data2)
        print(df_example)
```

```
Empty DataFrame
Columns: []
Index: []
```

```
In [10]: list = [1]
         df_example = pd.DataFrame()
         for i in list:
             data2 = pd.DataFrame([['A', 'B'],[1,2]])
```

```
    df_example = df_example.append(data2)
df_example.head()
```

```
Out[10]:    0  1
         0  A  B
         1  1  2
```

Now lets see if there are any top10 names that are unique to a particular state. First lets create a data frame of just the names using slicing. Before we dive to deeply we can use the nifty describe( ) function of Data Frames to quickly see if there are any unique names in the top ten.

```
In [11]: df_name = df.loc[:,'Name']
         df_name.describe()
```

```
Out[11]: count        510
         unique        40
         top       Olivia
         freq          51
         Name: Name, dtype: object
```

1.1.6 If we had sliced the a numerical column such as 'Births' are statistics would give the max and min number of births found etc. Since we don't have numerical data we get the max number of entries in the names column which is 510 for 10 names per state (including D.C). Here we can see that there are only 40 names that are represented in the top10 across all states.

There are a lot of ways you can filter data in a data frame. I have seen the name "Brooklyn" around a lot and I would like to know which states have baby girls with this name in 2016. You can slice a column using df['name of column']. You can also filter a column using a Boolean expression.

```
In [12]: col = df['Name']
         print(df[col == "Brooklyn"])
```

```
   Births      Name State
4     113  Brooklyn    MS
9      56  Brooklyn    WV
```

We know from the describe( ) method function above that there are only 40 names in the Top10 across all states. We can use the value_count() method to quickly determine how many states have each name in their top10.

```
In [13]: name_state_counts = df['Name'].value_counts()
         name_state_counts.head()
```

```
Out[13]: Olivia       51
         Emma         51
         Ava          50
         Sophia       43
         Charlotte    40
         Name: Name, dtype: int64
```

So Olivia and Emma are rock stars. We can now use the map( ) method to add a column containing the total number of states that have a particular name in the top10. The map method takes a dictionary so we will convert our state count series to a dictionary using the to_dict( ) method.

```
In [14]: freq_to_name = name_state_counts.to_dict()
         print(freq_to_name)
```

```
{'Olivia': 51, 'Emma': 51, 'Ava': 50, 'Sophia': 43, 'Charlotte': 40, 'Isabella': 35, 'Harper':
```

Now we will use map( ) which allows you to do a transformation from values in an array, Series or DataFrame column. We will call this new column 'name_freq' for name frequency.

```
In [15]: df['name_freq'] = df['Name'].map(freq_to_name)
         df
```

```
Out[15]:     Births       Name State  name_freq
         0       47       Emma    AK         51
         1       45     Olivia    AK         51
         2       34  Charlotte    AK         40
         3       34     Amelia    AK         27
         4       33     Sophia    AK         43
         5       32  Elizabeth    AK         15
         6       32        Ava    AK         50
         7       32    Abigail    AK         29
         8       31     Aurora    AK          1
         9       29      Chloe    AK          2
         0      330        Ava    AL         50
         1      244       Emma    AL         51
         2      238     Olivia    AL         51
         3      216  Elizabeth    AL         15
         4      181     Harper    AL         35
         5      166    Madison    AL         11
         6      160     Amelia    AL         27
         7      157   Caroline    AL          1
         8      156   Isabella    AL         35
         9      152       Ella    AL          4
         0      191        Ava    AR         50
         1      164       Emma    AR         51
         2      152     Olivia    AR         51
         3      132    Abigail    AR         29
         4      120     Harper    AR         35
         5      107     Sophia    AR         43
         6       97    Paisley    AR          3
         7       95    Addison    AR          4
         8       91   Isabella    AR         35
         9       87        Mia    AR         22
         ..     ...        ...   ...        ...
```

```
0      343     Olivia    WI          51
1      319       Emma    WI          51
2      272        Ava    WI          50
3      264     Harper    WI          35
4      263  Charlotte    WI          40
5      242     Evelyn    WI          21
6      230     Amelia    WI          27
7      210     Sophia    WI          43
8      189       Nora    WI           5
9      183    Abigail    WI          29
0      102     Harper    WV          35
1       93     Olivia    WV          51
2       90        Ava    WV          50
3       88       Emma    WV          51
4       74   Isabella    WV          35
5       70    Addison    WV           4
6       68    Paisley    WV           3
7       66     Sophia    WV          43
8       61      Avery    WV          10
9       56   Brooklyn    WV           2
0       36       Emma    WY          51
1       29     Olivia    WY          51
2       24     Harper    WY          35
3       24        Ava    WY          50
4       23  Elizabeth    WY          15
5       21      Emily    WY          15
6       18      Piper    WY           1
7       18     Evelyn    WY          21
8       18      Avery    WY          10
9       17     Sophia    WY          43

[510 rows x 4 columns]
```

It is also easy to create a new data frame by pivoting our data using the crosstab( ) function.

```
In [16]: data = pd.crosstab([df.State, df.Name], df.name_freq)
         data
```

```
Out[16]: name_freq        1   2   3   4   5   8   10  11  15  21  22  27  29  35  40  \
         State Name
         AK    Abigail    0   0   0   0   0   0   0   0   0   0   0   0   1   0   0
               Amelia     0   0   0   0   0   0   0   0   0   0   0   1   0   0   0
               Aurora     1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
               Ava        0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
               Charlotte  0   0   0   0   0   0   0   0   0   0   0   0   0   0   1
               Chloe      0   1   0   0   0   0   0   0   0   0   0   0   0   0   0
               Elizabeth  0   0   0   0   0   0   0   0   1   0   0   0   0   0   0
               Emma       0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

	Olivia	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Sophia	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
AL	Amelia	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
	Ava	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Caroline	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Elizabeth	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
	Ella	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	Emma	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Harper	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
	Isabella	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
	Madison	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
	Olivia	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
AR	Abigail	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
	Addison	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	Ava	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Emma	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Harper	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
	Isabella	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
	Mia	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
	Olivia	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Paisley	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	Sophia	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
...		..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
WI	Abigail	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
	Amelia	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
	Ava	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Charlotte	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	Emma	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Evelyn	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
	Harper	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
	Nora	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
	Olivia	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Sophia	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
WV	Addison	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	Ava	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Avery	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
	Brooklyn	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	Emma	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Harper	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
	Isabella	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
	Olivia	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Paisley	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	Sophia	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
WY	Ava	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Avery	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
	Elizabeth	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
	Emily	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
	Emma	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
      Evelyn      0   0   0   0   0   0   0   0   0   1   0   0   0   0   0
      Harper      0   0   0   0   0   0   0   0   0   0   0   0   0   1   0
      Olivia      0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
      Piper       1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
      Sophia      0   0   0   0   0   0   0   0   0   0   0   0   0   0   0

name_freq        43  50  51
State Name
AK    Abigail     0   0   0
      Amelia      0   0   0
      Aurora      0   0   0
      Ava         0   1   0
      Charlotte   0   0   0
      Chloe       0   0   0
      Elizabeth   0   0   0
      Emma        0   0   1
      Olivia      0   0   1
      Sophia      1   0   0
AL    Amelia      0   0   0
      Ava         0   1   0
      Caroline    0   0   0
      Elizabeth   0   0   0
      Ella        0   0   0
      Emma        0   0   1
      Harper      0   0   0
      Isabella    0   0   0
      Madison     0   0   0
      Olivia      0   0   1
AR    Abigail     0   0   0
      Addison     0   0   0
      Ava         0   1   0
      Emma        0   0   1
      Harper      0   0   0
      Isabella    0   0   0
      Mia         0   0   0
      Olivia      0   0   1
      Paisley     0   0   0
      Sophia      1   0   0
...              ..  ..  ..
WI    Abigail     0   0   0
      Amelia      0   0   0
      Ava         0   1   0
      Charlotte   0   0   0
      Emma        0   0   1
      Evelyn      0   0   0
      Harper      0   0   0
      Nora        0   0   0
      Olivia      0   0   1
```

```
      Sophia      1   0   0
WV    Addison     0   0   0
      Ava         0   1   0
      Avery       0   0   0
      Brooklyn    0   0   0
      Emma        0   0   1
      Harper      0   0   0
      Isabella    0   0   0
      Olivia      0   0   1
      Paisley     0   0   0
      Sophia      1   0   0
WY    Ava         0   1   0
      Avery       0   0   0
      Elizabeth   0   0   0
      Emily       0   0   0
      Emma        0   0   1
      Evelyn      0   0   0
      Harper      0   0   0
      Olivia      0   0   1
      Piper       0   0   0
      Sophia      1   0   0

[510 rows x 18 columns]
```

Now we have created a data frame where the index is the state. Having a duplicate index is allowed but can make many other data manipulations more difficult.

I'm less interested in the particular names as I am in the frequency of novel names in each state. So I will reset the index followed by the groupby( ) function to group the states and sum the frequency of each name. If you try this without reseting the index you will get an error because you need a unique index for each color to do this.

```
In [17]: data_state = data.reset_index().groupby('State').sum()
         data_state
```

```
Out[17]: name_freq  1   2   3   4   5   8   10  11  15  21  22  27  29  35  40  43  50  \
         State
         AK          1   1   0   0   0   0   0   0   1   0   0   1   1   0   1   1   1
         AL          1   0   0   1   0   0   0   1   1   0   0   1   0   2   0   0   1
         AR          0   0   1   1   0   0   0   0   0   0   1   0   1   2   0   1   1
         AZ          0   0   0   0   0   1   0   0   1   0   1   0   1   1   1   1   1
         CA          0   1   0   0   0   1   0   0   1   0   1   0   1   1   0   1   1
         CO          0   0   0   0   0   0   0   0   0   1   1   0   1   2   1   1   1
         CT          0   0   0   0   0   0   0   0   1   0   1   1   1   1   1   1   1
         DC          1   0   0   1   0   1   0   0   1   0   0   0   1   1   1   0   1
         DE          1   0   0   1   0   0   0   0   1   0   0   0   0   2   1   1   1
         FL          0   0   0   0   0   1   0   0   1   0   1   1   1   1   0   1   1
         GA          0   0   0   0   0   0   0   1   0   0   0   1   1   2   1   1   1
         HI          1   0   1   1   0   0   0   0   0   0   1   1   0   1   0   1   1
```

IA	0	0	0	1	1	0	0	0	0	1	0	1	0	1	1	1	1
ID	0	0	0	0	0	0	0	0	2	1	0	1	1	1	1	0	1
IL	0	0	0	0	0	1	0	0	1	1	1	0	0	1	1	1	1
IN	0	0	0	0	0	0	0	0	0	1	0	1	1	2	1	1	1
KS	0	0	0	0	0	0	1	0	1	1	1	0	0	1	1	1	1
KY	0	0	0	0	0	0	0	0	0	0	1	1	1	2	1	1	1
LA	0	0	0	0	0	0	1	1	0	0	1	1	0	1	1	1	1
MA	0	0	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1
MD	0	0	0	0	0	0	0	1	1	0	0	0	1	2	1	1	1
ME	1	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1
MI	0	0	0	0	0	0	0	0	0	1	0	1	1	2	1	1	1
MN	0	0	1	0	1	0	0	0	0	1	0	1	0	1	1	1	1
MO	0	0	0	0	0	0	1	0	0	1	0	1	0	2	1	1	1
MS	0	3	1	0	0	0	0	1	0	0	0	0	0	2	0	0	1
MT	0	0	1	0	0	0	1	1	0	1	0	0	1	1	1	0	1
NC	0	0	0	0	0	0	0	1	1	0	0	0	1	2	1	1	1
ND	0	0	0	2	1	0	0	0	0	1	0	1	0	1	1	0	1
NE	0	0	0	0	0	0	1	0	1	1	1	0	0	1	1	1	1
NH	0	0	1	0	0	0	0	1	0	0	0	1	1	2	1	1	0
NJ	0	0	0	0	0	0	0	1	1	0	1	0	1	1	1	1	1
NM	2	0	0	0	0	1	0	0	1	0	1	0	0	1	0	1	1
NV	1	0	0	0	0	1	0	0	1	0	1	0	1	1	0	1	1
NY	0	0	0	0	0	0	0	1	1	0	1	0	1	1	1	1	1
OH	0	0	0	0	0	0	1	0	0	1	0	1	0	2	1	1	1
OK	0	0	0	0	0	0	0	0	1	0	1	0	1	2	1	1	1
OR	0	0	0	0	0	0	0	0	0	1	0	1	1	2	1	1	1
PA	0	0	0	0	0	0	0	0	1	0	1	0	1	2	1	1	1
RI	0	0	0	0	0	0	1	0	1	0	1	1	0	1	1	1	1
SC	0	1	0	1	0	0	0	1	1	0	0	0	0	2	1	0	1
SD	0	0	0	0	1	0	1	0	1	1	0	0	0	1	1	1	1
TN	0	0	0	1	0	0	0	0	1	0	0	1	0	2	1	1	1
TX	0	1	0	0	0	1	0	0	1	0	1	0	1	1	0	1	1
UT	3	0	0	0	0	0	0	0	0	1	0	1	0	1	1	0	1
VA	0	0	0	0	0	0	0	0	1	0	0	1	1	2	1	1	1
VT	0	0	1	1	0	0	0	0	0	1	0	1	0	1	1	1	1
WA	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1	1	1
WI	0	0	0	0	1	0	0	0	0	1	0	1	1	1	1	1	1
WV	0	1	1	1	0	0	1	0	0	0	0	0	0	2	0	1	1
WY	1	0	0	0	0	0	1	0	2	1	0	0	0	1	0	1	1

name_freq	51
State	
AK	2
AL	2
AR	2
AZ	2
CA	2
CO	2

```
CT          2
DC          2
DE          2
FL          2
GA          2
HI          2
IA          2
ID          2
IL          2
IN          2
KS          2
KY          2
LA          2
MA          2
MD          2
ME          2
MI          2
MN          2
MO          2
MS          2
MT          2
NC          2
ND          2
NE          2
NH          2
NJ          2
NM          2
NV          2
NY          2
OH          2
OK          2
OR          2
PA          2
RI          2
SC          2
SD          2
TN          2
TX          2
UT          2
VA          2
VT          2
WA          2
WI          2
WV          2
WY          2
```

This is great but now I want a way to rank each state for name uniqueness. I can create a new data series consisting of the sum of all name frequencies for each state. States with a lower score are more unique than states with a higher score

```
In [18]: df_sum = df.groupby('State').name_freq.sum()
         df_sum
```

```
Out[18]: State
         AK    309
         AL    280
         AR    323
         AZ    344
         CA    306
         CO    377
         CT    363
         DC    284
         DE    325
         FL    331
         GA    372
         HI    287
         IA    327
         ID    334
         IL    336
         IN    382
         KS    338
         KY    383
         LA    340
         MA    351
         MD    360
         ME    348
         MI    382
         MN    326
         MO    363
         MS    242
         MT    301
         NC    360
         ND    288
         NE    338
         NH    325
         NJ    347
         NM    277
         NV    305
         NY    347
         OH    363
         OK    371
         OR    382
         PA    371
         RI    344
         SC    294
         SD    321
```

```
TN    351
TX    306
UT    278
VA    376
VT    325
WA    349
WI    352
WV    284
WY    292
Name: name_freq, dtype: int64
```

There are some nice built in Pandas functions that allow us to calculate some common statistics. For example we can compute the mean, max and min of our name frequencies:

```
In [19]: df_sum.mean(), df_sum.max(), df_sum.min()
```

```
Out[19]: (332.54901960784315, 383, 242)
```

Now I would like to normalize these values to a 0-1 scale and sort the new data frame from highest to lowest. We will use this later.

```
In [20]: unique_score =  1-((df_sum - df_sum.min()) / (df_sum.max() - df_sum.min()))
         unique_score
```

```
Out[20]: State
         AK    0.524823
         AL    0.730496
         AR    0.425532
         AZ    0.276596
         CA    0.546099
         CO    0.042553
         CT    0.141844
         DC    0.702128
         DE    0.411348
         FL    0.368794
         GA    0.078014
         HI    0.680851
         IA    0.397163
         ID    0.347518
         IL    0.333333
         IN    0.007092
         KS    0.319149
         KY    0.000000
         LA    0.304965
         MA    0.226950
         MD    0.163121
         ME    0.248227
         MI    0.007092
         MN    0.404255
```

```
MO     0.141844
MS     1.000000
MT     0.581560
NC     0.163121
ND     0.673759
NE     0.319149
NH     0.411348
NJ     0.255319
NM     0.751773
NV     0.553191
NY     0.255319
OH     0.141844
OK     0.085106
OR     0.007092
PA     0.085106
RI     0.276596
SC     0.631206
SD     0.439716
TN     0.226950
TX     0.546099
UT     0.744681
VA     0.049645
VT     0.411348
WA     0.241135
WI     0.219858
WV     0.702128
WY     0.645390
Name: name_freq, dtype: float64
```

Now we can Concatenate our "data_state" data frame and our new "unique_score" data series using the concat( ) function. The index values are our states. We can sort the states by the name_frequency value

In [21]:
```
data_updated = pd.concat([data_state, unique_score], axis=1).sort_values(by='name_fre
data_updated
```

Out[21]:
```
       1  2  3  4  5  8  10  11  15  21  22  27  29  35  40  43  50  51  \
State
MS     0  3  1  0  0  0   0   1   0   0   0   0   0   2   0   0   1   2
NM     2  0  0  0  0  1   0   0   1   0   1   0   0   1   0   1   1   2
UT     3  0  0  0  0  0   0   0   0   1   0   1   0   1   1   0   1   2
AL     1  0  0  1  0  0   0   1   1   0   0   1   0   2   0   0   1   2
WV     0  1  1  1  0  0   1   0   0   0   0   0   0   2   0   1   1   2
DC     1  0  0  1  0  1   0   0   1   0   0   0   1   1   1   0   1   2
HI     1  0  1  1  0  0   0   0   0   0   1   1   0   1   0   1   1   2
ND     0  0  0  2  1  0   0   0   0   1   0   1   0   1   1   0   1   2
WY     1  0  0  0  0  0   1   0   2   1   0   0   0   1   0   1   1   2
SC     0  1  0  1  0  0   0   1   1   0   0   0   0   2   1   0   1   2
```

MT	0	0	1	0	0	0	1	1	0	1	0	0	1	1	1	0	1	2
NV	1	0	0	0	0	1	0	0	1	0	1	0	1	1	0	1	1	2
TX	0	1	0	0	0	1	0	0	1	0	1	0	1	1	0	1	1	2
CA	0	1	0	0	0	1	0	0	1	0	1	0	1	1	0	1	1	2
AK	1	1	0	0	0	0	0	0	1	0	0	1	1	0	1	1	1	2
SD	0	0	0	0	1	0	1	0	1	1	0	0	0	1	1	1	1	2
AR	0	0	1	1	0	0	0	0	0	0	1	0	1	2	0	1	1	2
VT	0	0	1	1	0	0	0	0	0	1	0	1	0	1	1	1	1	2
NH	0	0	1	0	0	0	0	1	0	0	0	1	1	2	1	1	0	2
DE	1	0	0	1	0	0	0	0	1	0	0	0	0	2	1	1	1	2
MN	0	0	1	0	1	0	0	0	0	1	0	1	0	1	1	1	1	2
IA	0	0	0	1	1	0	0	0	0	1	0	1	0	1	1	1	1	2
FL	0	0	0	0	0	1	0	0	1	0	1	1	1	1	0	1	1	2
ID	0	0	0	0	0	0	0	0	2	1	0	1	1	1	1	0	1	2
IL	0	0	0	0	0	1	0	0	1	1	1	0	0	1	1	1	1	2
NE	0	0	0	0	0	0	1	0	1	1	1	0	0	1	1	1	1	2
KS	0	0	0	0	0	0	1	0	1	1	1	0	0	1	1	1	1	2
LA	0	0	0	0	0	0	1	1	0	0	1	1	0	1	1	1	1	2
RI	0	0	0	0	0	0	1	0	1	0	1	1	0	1	1	1	1	2
AZ	0	0	0	0	0	1	0	0	1	0	1	0	1	1	1	1	1	2
NJ	0	0	0	0	0	0	0	1	1	0	1	0	1	1	1	1	1	2
NY	0	0	0	0	0	0	0	1	1	0	1	0	1	1	1	1	1	2
ME	1	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	2
WA	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1	1	1	2
TN	0	0	0	1	0	0	0	0	1	0	0	1	0	2	1	1	1	2
MA	0	0	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	2
WI	0	0	0	0	1	0	0	0	0	1	0	1	1	1	1	1	1	2
MD	0	0	0	0	0	0	0	1	1	0	0	0	1	2	1	1	1	2
NC	0	0	0	0	0	0	0	1	1	0	0	0	1	2	1	1	1	2
CT	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1	2
MO	0	0	0	0	0	0	1	0	0	1	0	1	0	2	1	1	1	2
OH	0	0	0	0	0	0	1	0	0	1	0	1	0	2	1	1	1	2
OK	0	0	0	0	0	0	0	0	1	0	1	0	1	2	1	1	1	2
PA	0	0	0	0	0	0	0	0	1	0	1	0	1	2	1	1	1	2
GA	0	0	0	0	0	0	0	1	0	0	0	1	1	2	1	1	1	2
VA	0	0	0	0	0	0	0	0	1	0	0	1	1	2	1	1	1	2
CO	0	0	0	0	0	0	0	0	0	1	1	0	1	2	1	1	1	2
OR	0	0	0	0	0	0	0	0	0	1	0	1	1	2	1	1	1	2
IN	0	0	0	0	0	0	0	0	0	1	0	1	1	2	1	1	1	2
MI	0	0	0	0	0	0	0	0	0	1	0	1	1	2	1	1	1	2
KY	0	0	0	0	0	0	0	0	0	0	1	1	1	2	1	1	1	2

	name_freq
State	
MS	1.000000
NM	0.751773
UT	0.744681
AL	0.730496

WV	0.702128
DC	0.702128
HI	0.680851
ND	0.673759
WY	0.645390
SC	0.631206
MT	0.581560
NV	0.553191
TX	0.546099
CA	0.546099
AK	0.524823
SD	0.439716
AR	0.425532
VT	0.411348
NH	0.411348
DE	0.411348
MN	0.404255
IA	0.397163
FL	0.368794
ID	0.347518
IL	0.333333
NE	0.319149
KS	0.319149
LA	0.304965
RI	0.276596
AZ	0.276596
NJ	0.255319
NY	0.255319
ME	0.248227
WA	0.241135
TN	0.226950
MA	0.226950
WI	0.219858
MD	0.163121
NC	0.163121
CT	0.141844
MO	0.141844
OH	0.141844
OK	0.085106
PA	0.085106
GA	0.078014
VA	0.049645
CO	0.042553
OR	0.007092
IN	0.007092
MI	0.007092
KY	0.000000

We can see that Mississippi has the highest baby name uniqueness score. Let us see what the names our. Tod do this we can make a Pandas series from the df data frame and pass it into the df data frame as a boolean.

```
In [ ]: col = df['State']
        df[col=='MS']
```

It is a good idea to change the column name of the last column as it has the same name as our rows.

```
In [23]: data_updated = data_updated.rename(columns={'name_freq':'unique score' })
         data_updated
```

Out[23]:

State	1	2	3	4	5	8	10	11	15	21	22	27	29	35	40	43	50	51	\
MS	0	3	1	0	0	0	0	1	0	0	0	0	0	2	0	0	1	2	
NM	2	0	0	0	0	1	0	0	1	0	1	0	0	1	0	1	1	2	
UT	3	0	0	0	0	0	0	0	0	1	0	1	0	1	1	0	1	2	
AL	1	0	0	1	0	0	0	1	1	0	0	1	0	2	0	0	1	2	
WV	0	1	1	1	0	0	1	0	0	0	0	0	0	2	0	1	1	2	
DC	1	0	0	1	0	1	0	0	1	0	0	0	1	1	1	0	1	2	
HI	1	0	1	1	0	0	0	0	0	0	1	1	0	1	0	1	1	2	
ND	0	0	0	2	1	0	0	0	0	1	0	1	0	1	1	0	1	2	
WY	1	0	0	0	0	0	1	0	2	1	0	0	0	1	0	1	1	2	
SC	0	1	0	1	0	0	0	1	1	0	0	0	0	2	1	0	1	2	
MT	0	0	1	0	0	0	1	1	0	1	0	0	1	1	1	0	1	2	
NV	1	0	0	0	0	1	0	0	1	0	1	0	1	1	0	1	1	2	
TX	0	1	0	0	0	1	0	0	1	0	1	0	1	1	0	1	1	2	
CA	0	1	0	0	0	1	0	0	1	0	1	0	1	1	0	1	1	2	
AK	1	1	0	0	0	0	0	0	1	0	0	1	1	0	1	1	1	2	
SD	0	0	0	0	1	0	1	0	1	1	0	0	0	1	1	1	1	2	
AR	0	0	1	1	0	0	0	0	0	0	1	0	1	2	0	1	1	2	
VT	0	0	1	1	0	0	0	0	0	1	0	1	0	1	1	1	1	2	
NH	0	0	1	0	0	0	0	1	0	0	0	1	1	2	1	1	0	2	
DE	1	0	0	1	0	0	0	0	1	0	0	0	0	2	1	1	1	2	
MN	0	0	1	0	1	0	0	0	0	1	0	1	0	1	1	1	1	2	
IA	0	0	0	1	1	0	0	0	0	1	0	1	0	1	1	1	1	2	
FL	0	0	0	0	0	1	0	0	1	0	1	1	1	1	0	1	1	2	
ID	0	0	0	0	0	0	0	0	2	1	0	1	1	1	1	0	1	2	
IL	0	0	0	0	0	1	0	0	1	1	1	0	0	1	1	1	1	2	
NE	0	0	0	0	0	0	1	0	1	1	1	0	0	1	1	1	1	2	
KS	0	0	0	0	0	0	1	0	1	1	1	0	0	1	1	1	1	2	
LA	0	0	0	0	0	0	1	1	0	0	1	1	0	1	1	1	1	2	
RI	0	0	0	0	0	0	1	0	1	0	1	1	0	1	1	1	1	2	
AZ	0	0	0	0	0	1	0	0	1	0	1	0	1	1	1	1	1	2	
NJ	0	0	0	0	0	0	0	1	1	0	1	0	1	1	1	1	1	2	
NY	0	0	0	0	0	0	0	1	1	0	1	0	1	1	1	1	1	2	
ME	1	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	2	
WA	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1	1	1	2	

TN	0	0	0	1	0	0	0	0	1	0	0	1	0	2	1	1	1	2
MA	0	0	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	2
WI	0	0	0	0	1	0	0	0	0	1	0	1	1	1	1	1	1	2
MD	0	0	0	0	0	0	0	1	1	0	0	0	1	2	1	1	1	2
NC	0	0	0	0	0	0	0	1	1	0	0	0	1	2	1	1	1	2
CT	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1	2
MO	0	0	0	0	0	0	1	0	0	1	0	1	0	2	1	1	1	2
OH	0	0	0	0	0	0	1	0	0	1	0	1	0	2	1	1	1	2
OK	0	0	0	0	0	0	0	0	1	0	1	0	1	2	1	1	1	2
PA	0	0	0	0	0	0	0	0	1	0	1	0	1	2	1	1	1	2
GA	0	0	0	0	0	0	0	1	0	0	0	1	1	2	1	1	1	2
VA	0	0	0	0	0	0	0	0	1	0	0	1	1	2	1	1	1	2
CO	0	0	0	0	0	0	0	0	0	1	1	0	1	2	1	1	1	2
OR	0	0	0	0	0	0	0	0	0	1	0	1	1	2	1	1	1	2
IN	0	0	0	0	0	0	0	0	0	1	0	1	1	2	1	1	1	2
MI	0	0	0	0	0	0	0	0	0	1	0	1	1	2	1	1	1	2
KY	0	0	0	0	0	0	0	0	0	0	1	1	1	2	1	1	1	2

	unique score
State	
MS	1.000000
NM	0.751773
UT	0.744681
AL	0.730496
WV	0.702128
DC	0.702128
HI	0.680851
ND	0.673759
WY	0.645390
SC	0.631206
MT	0.581560
NV	0.553191
TX	0.546099
CA	0.546099
AK	0.524823
SD	0.439716
AR	0.425532
VT	0.411348
NH	0.411348
DE	0.411348
MN	0.404255
IA	0.397163
FL	0.368794
ID	0.347518
IL	0.333333
NE	0.319149
KS	0.319149
LA	0.304965

```
RI       0.276596
AZ       0.276596
NJ       0.255319
NY       0.255319
ME       0.248227
WA       0.241135
TN       0.226950
MA       0.226950
WI       0.219858
MD       0.163121
NC       0.163121
CT       0.141844
MO       0.141844
OH       0.141844
OK       0.085106
PA       0.085106
GA       0.078014
VA       0.049645
CO       0.042553
OR       0.007092
IN       0.007092
MI       0.007092
KY       0.000000
```

In [24]:
```
score_dict = unique_score.to_dict()
score_dict
```

Out[24]:
```
{'AK': 0.5248226950354609 7,
 'AL': 0.73049645390070927,
 'AR': 0.42553191489361697,
 'AZ': 0.27659574468085102,
 'CA': 0.54609929078014185,
 'CO': 0.042553191489361653,
 'CT': 0.14184397163120566,
 'DC': 0.7021276595744681,
 'DE': 0.41134751773049649,
 'FL': 0.36879432624113473,
 'GA': 0.078014184397163122,
 'HI': 0.68085106382978722,
 'IA': 0.3971631205673759,
 'ID': 0.34751773049645385,
 'IL': 0.33333333333333337,
 'IN': 0.0070921985815602939,
 'KS': 0.31914893617021278,
 'KY': 0.0,
 'LA': 0.30496453900709219,
 'MA': 0.22695035460992907,
 'MD': 0.16312056737588654,
```

```
 'ME': 0.24822695035460995,
 'MI': 0.0070921985815602939,
 'MN': 0.4042553191489362,
 'MO': 0.14184397163120566,
 'MS': 1.0,
 'MT': 0.58156028368794321,
 'NC': 0.16312056737588654,
 'ND': 0.67375886524822692,
 'NE': 0.31914893617021278,
 'NH': 0.41134751773049649,
 'NJ': 0.25531914893617025,
 'NM': 0.75177304964539005,
 'NV': 0.55319148936170215,
 'NY': 0.25531914893617025,
 'OH': 0.14184397163120566,
 'OK': 0.085106382978723416,
 'OR': 0.0070921985815602939,
 'PA': 0.085106382978723416,
 'RI': 0.27659574468085102,
 'SC': 0.63120567375886527,
 'SD': 0.43971631205673756,
 'TN': 0.22695035460992907,
 'TX': 0.54609929078014185,
 'UT': 0.74468085106382986,
 'VA': 0.049645390070921946,
 'VT': 0.41134751773049649,
 'WA': 0.24113475177304966,
 'WI': 0.21985815602836878,
 'WV': 0.7021276595744681,
 'WY': 0.64539007092198575}
```

First we make a list of all of the states inluding Washington D.C. Next we make a list of the state abbreviations using a list comprehension by iterating over the indices of the unqiue_score data series.

```
In [ ]: state_list = ['Alaska', 'Alabama', 'Arkansas', 'Arizona', 'California',
                      'Colorado', 'Connecticut', 'District of Columbia', 'Delaware',
                      'Florida', 'Georgia', 'Hawaii', 'Iowa', 'Idaho', 'Illinois',
                      'Indiana', 'Kansas', 'Kentucky', 'Louisiana', 'Massachusetts',
                      'Maryland', 'Maine', 'Michigan', 'Minnesota', 'Missouri',
                      'Mississippi', 'Montana', 'North Carolina', 'North Dakota',
                      'Nebraska', 'New Hampshire', 'New Jersey', 'New Mexico',
                      'Nevada', 'New York', 'Ohio', 'Oklahoma', 'Oregon',
                      'Pennsylvania', 'Rhode Island', 'South Carolina',
                      'South Dakota', 'Tennessee', 'Texas', 'Utah', 'Virginia',
                      'Vermont', 'Washington', 'Wisconsin', 'West Virginia',
                      'Wyoming']
```

```
state_abbrev = [i for i in unique_score.index]
print(state_abbrev)
```

Now we can use the Python Zip( ) function to make a list of tuples that we can pass the dict( ) function to give us our new dictionary.

```
In [26]: combine = zip(state_abbrev, state_list)
         state_abbrev_dict = dict(combine)
         print(state_abbrev_dict)
```

```
{'AK': 'Alaska', 'AL': 'Alabama', 'AR': 'Arkansas', 'AZ': 'Arizona', 'CA': 'California', 'CO':
```

We will create a new dictionary for Basemap with the spelled out states as keys and the scores as values. We first initialize an empty dictionary. Then we use the items( ) method of dictionaries to iterate over the keys and values in our score_dict we made above. If the key matches the key in our state_abbrev_dict (The state abbreviations are the same) we use the value for that paticular key (state name spelled out) as the new key for new_dict. The value for score_dict() (the score for that state) is then assigned as the value new_dict.

```
In [27]: new_dict = {}
         for key, value in score_dict.items():
             if key in state_abbrev_dict:
                 new_dict[state_abbrev_dict[key]] = value

         print(new_dict)
         print(new_dict.get('Mississippi'))
         print(score_dict.get('MS'))
```

```
{'Alaska': 0.52482269503546097, 'Alabama': 0.73049645390070927, 'Arkansas': 0.4255319148936169
1.0
1.0
```

To check our work. We can print out any score from our dictionary using the get( ) method of dictionaries. In the above example we printed out the score for Mississippi for the new_dict and the score_dict.

Now we are ready to pass our data into Basemap but first let us prep the maps. Briefly, "llcrnlon" is the longitude of the lower left corner, "llcrnlat" is the latitude of the lower left hand corner, "urcnrlon" is the longitude of the upper right hand cornder and "urcrnrlat" is the latitude of the upper right hand corner for the map. The 'llc' projection works well for square maps. Finally lon_0 specifies the center longitude coordinate for the map. Finally, lat_1 and lat_2 allow you to define an oblique centerline. These coordinates were lifted from the Basemap example file, "fill_states.py" that is distributed with the Basemap package.

```
In [29]: # Lambert Conformal map of lower 48 states.
         #from mpl_toolkits.basemap import Basemap
         import math
         import csv
```

```
import pandas as pd
import os
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
from matplotlib.colors import rgb2hex
from matplotlib.patches import Polygon

#map_lower_48 = Basemap(llcrnrlon=-119,llcrnrlat=22,urcrnrlon=-64,urcrnrlat=49,projec

map_lower_48 = Basemap(llcrnrlon=-121,llcrnrlat=20,urcrnrlon=-62,urcrnrlat=51,project
```

This lets Basemap know where in the world we want to focus on. But Basemap does not know anything about the shapes of the U.S states. For this need to provide a shape file. This is a little confusing because although it is refered to as a shape file, there are actually three files which provide the necessary information. They all have the 'st99_d00' prefix with three different extensions, .dbf, .shp and .shx. The .shp file is a binary file containing geometric data and the .dbf and .shx are supporting files. These are maintained by the ESRI at http://www.esri.com/. Next we can use the Basemap readshapefile( ) function to read in these files. Do not include the extensions and make sure all three files are in the same directory.

```
In [34]: states_shp_info = map_lower_48.readshapefile('shapes\st99_d00', 'states', drawbounds=
         plt.show()
```

Now we have a varriable called "shp_info" describing a map of the lower forty eight states. Now let us make the Alaska and Hawaii maps

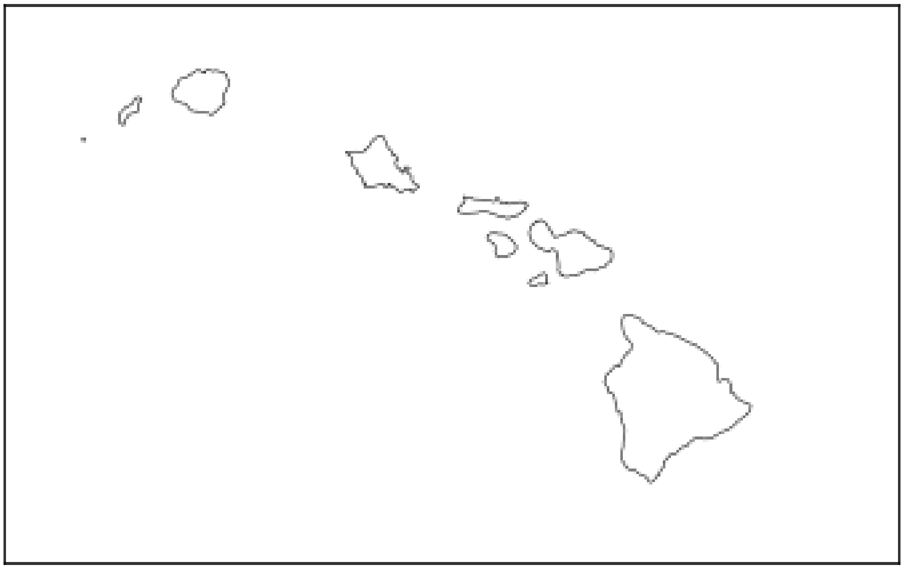
```
In [35]: m_alaska = Basemap(llcrnrlon=-173,llcrnrlat=51,urcrnrlon=-134.45,urcrnrlat=71.34,proj
```

```
In [37]: alaska_shp_info = m_alaska.readshapefile('shapes/st99_d00', 'states', drawbounds=True
         plt.show()
```

Hawaii has a very oblong shape so it will be hard to make a square like map. If we leave out the lower left hand and upper right hand corner longitude and latitude we can define a 'great circle' using two points.

```
In [38]: m_hawaii = Basemap(#llcrnrlon=-161,llcrnrlat=21,urcrnrlon=-154,urcrnrlat=22,
                            width=800000, height=500000,
                            projection='lcc',
                            lat_0=20.525, lon_0=-157.385,
                            lon_1=-158.115, lat_1=17.24,
                            lon_2=-153.479, lat_2=19.663,
                            )
```

```
In [39]: hawaii_shp_info = m_hawaii.readshapefile('shapes/st99_d00', 'states',drawbounds=True)
         plt.show()
```



Now we are ready to color the maps. First we define the color map we are going to use with, cmap=plt.com.Blues. This will give us a blue spectrum in which dark blues will be assigned a higher uniqueness score and light blues will have a lower uniqueness score. Next we can define the maximum and minimum values for our score, vmin = 0; vmax = 1. When we read the shape file into basemap, a dictionary called "shapedict" was created and stored in 'states_shp_info'. This is where the state names and corrosponding shapes reside. for example if we pass the key "NAME" we will get back the state names as values. There are multiple entries for each state.

```
In [40]: for shapedict in map_lower_48.states_info:
             print(shapedict['NAME'])
```

```
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
```

```
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
```

Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Alaska
Minnesota
Washington
Washington
Montana
Idaho
North Dakota
Washington
Washington
Washington
Washington
Washington
Washington
Washington
Washington
Michigan
Washington
Michigan
Maine
Wisconsin
Wisconsin
Wisconsin
Wisconsin
Oregon
South Dakota
Michigan
Michigan
Michigan
Wisconsin
New Hampshire
Michigan
Wisconsin
Michigan

```
Vermont
New York
Wyoming
Maine
Maine
Maine
Maine
Maine
Maine
Maine
Maine
Maine
Maine
Maine
Maine
Maine
Maine
Maine
Maine
Maine
Maine
Maine
Maine
Maine
Iowa
Nebraska
Massachusetts
Illinois
Pennsylvania
Connecticut
Rhode Island
California
Nevada
Utah
Ohio
Indiana
Ohio
Ohio
Rhode Island
Ohio
Rhode Island
Rhode Island
Ohio
Massachusetts
Rhode Island
Rhode Island
Massachusetts
Massachusetts
```

New Jersey
New York
Rhode Island
New York
Colorado
New York
New York
New York
New York
West Virginia
Missouri
Kansas
Delaware
Maryland
Delaware
Delaware
Virginia
Kentucky
District of Columbia
Maryland
Maryland
Virginia
Virginia
California
Virginia
California
California
Arizona
Oklahoma
New Mexico
Tennessee
North Carolina
Kentucky
Texas
Arkansas
North Carolina
North Carolina
South Carolina
Alabama
Georgia
Mississippi
California
California
California
California
California
California
California

Louisiana
Florida
Alabama
Mississippi
Mississippi
Mississippi
Mississippi
Louisiana
Florida
Florida
Hawaii
Texas
Florida
Hawaii
Hawaii
Hawaii
Hawaii
Hawaii
Hawaii
Florida
Hawaii
Hawaii
Florida
Hawaii
Florida
Florida
Florida
Florida
Florida
Florida
Florida
Florida
Hawaii
Hawaii
Hawaii
Hawaii
Hawaii
Hawaii
Hawaii
Hawaii
Hawaii
Hawaii
Hawaii
Hawaii
Hawaii
Hawaii
Hawaii
Hawaii

```
Hawaii
Puerto Rico
Puerto Rico
Puerto Rico
Puerto Rico
Puerto Rico
Puerto Rico
Puerto Rico
Puerto Rico
Puerto Rico
Puerto Rico
Puerto Rico
Puerto Rico
Puerto Rico
Puerto Rico
Puerto Rico
```

We want to access shapedict using a for-loop. We initialize a new empty list called 'state_names'. These names will be filled with the names from the shapedict dictionary in the order and frequency that they appear. We want to skip "Puerto Rico" so we add; if state_name is not equal to "Puerto Rico", then the score is equal to our value of new_dict for a particular state_name key. So For every state_name entry in shapedict we get a score which is the unique score taken from new_dict. Next, we define the state_name as a key in the new colors dictionary we are creating. With, 'colors[state_name] = cmap(np.sqrt(score)' We let the key to be equal to the blues spectrum mapped onto the values for score. We use the numpy square root function to use the square root of the scores. This sepearates the valus more making it easier to see color differences.

```
In [41]: cmap = plt.cm.Blues
         vmin = 0; vmax = 1
         colors = {}
         state_names = []
         for shapedict in map_lower_48.states_info:
             state_name = shapedict['NAME']
             if state_name != 'Puerto Rico':
                 score = new_dict[state_name]
                 colors[state_name] = cmap(np.sqrt(score))
             state_names.append(state_name)
```

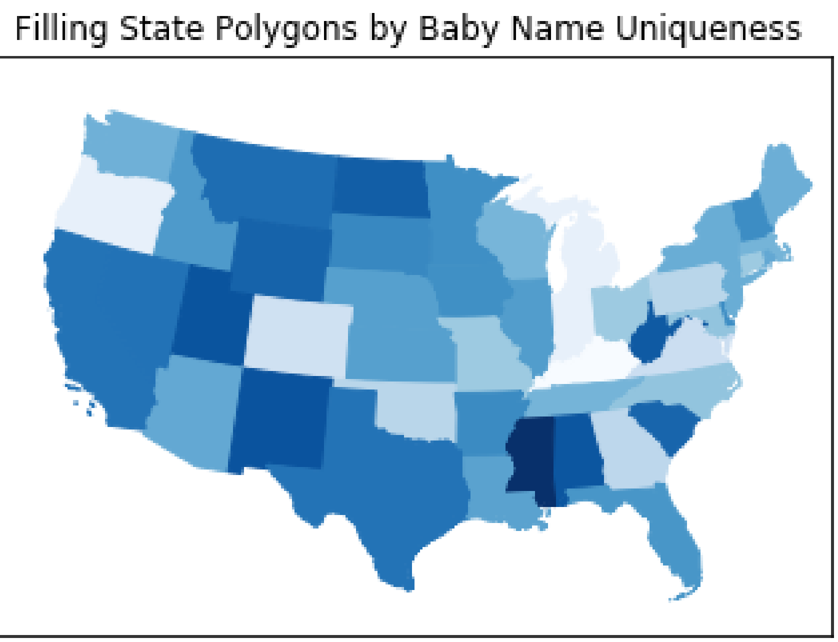
If we print the colors dictionary we can see that we have the state names as keys and RGB blue values that are scaled to the scores for each state. And we have a list of the state_names.

```
In [42]: print(colors)

{'Alaska': (0.15478662053056519, 0.46851211072664356, 0.72287581699346404, 1.0), 'Minnesota':

In [43]: print(state_names)
```

```
['Alaska', 'Alaska', 'Alaska', 'Alaska', 'Alaska', 'Alaska', 'Alaska', 'Alaska', 'Alaska', 'Ala
```

Now we can plot the data. We can add an axis with 'ax = plt.axes()'. Like "Names" above, "nshape" and "seg" are dictionary keys in the maps_lower_48 above. We iterate over these two set the color of the shapes with "color = rgb2hex(colors[state_names[nshape]])". The values for nshape are state names. So "colors[state_names[n_shape]] will give the rgb color of th a particular state. These are then converted to hex string with the matplotlib rgb2hex( ) function. The next two lines basically tell matplotlib to render the polygons with the colors just defined. Finally we can draw the states with "map_lower_48.drawstates". We can keep the color= color and the linewidth=0 so that we don't see borders around the states. Finally we show the plot with "plt.show( )"

```
In [44]: ax = plt.axes()
         for nshape, seg in enumerate(map_lower_48.states):
             if state_names[nshape] != 'Puerto Rico':
                 color = rgb2hex(colors[state_names[nshape]])
                 poly = Polygon(seg, facecolor=color, edgecolor=color)
                 ax.add_patch(poly)
         map_lower_48.drawstates(color=color, linewidth=0)
         plt.title('Filling State Polygons by Baby Name Uniqueness')
         plt.show()
```

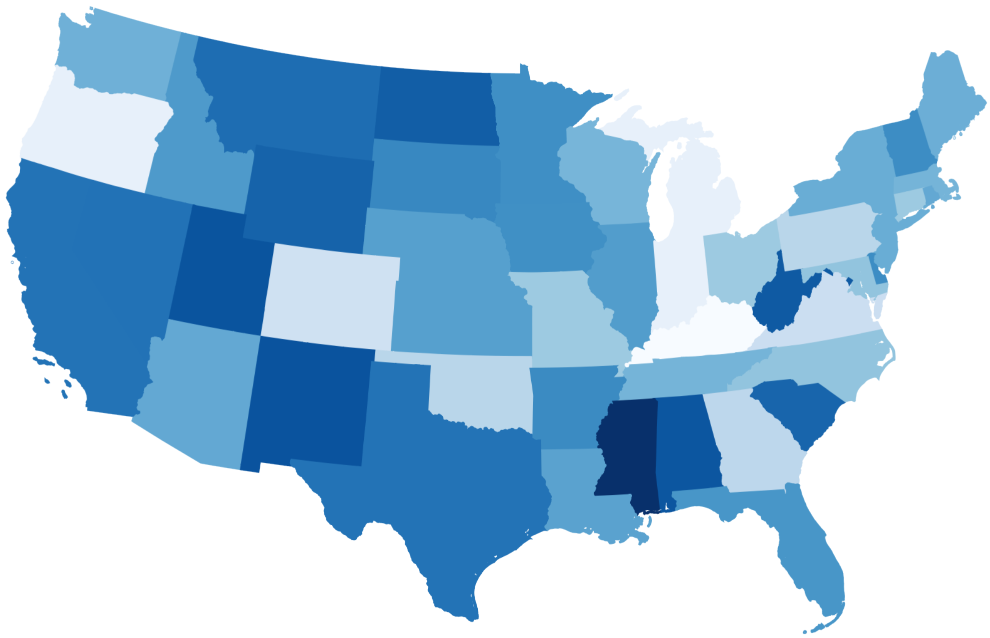


This can now be repeated for Alaska and Hawaii. However, it would be nice to do it in one shot and have a little more flexibility with labels. To do this we use the subplot2grid( ) function. We can also play around with the font by importing the font manager in the first line. We then define the figure size and resolution with "fig = plt.figure(figsize = (15, 24), dpi=300)". This requires to download the EBGaramond-Regular.ttf font and point to the directory it is in. We then set the axis properties for the mainland with "ax_mainland = plt.subplot2grid((4,4),(0,0) colspan=4". This creates a four by four grid with a column span of 4. We will not be plotting this many figures but it creates the space we need. The mainland plot will be placed in the upper left hand corner and span the entire grid. Next, we repeat what we did above for the mainland, Alaska and Hawaii.

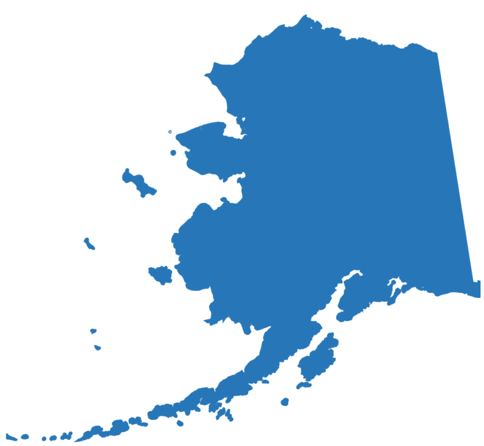
```
In [47]: import matplotlib.font_manager as fm
         import math
         cmap = plt.cm.Blues
         vmin = 0; vmax = 1

         fig = plt.figure(figsize = (15, 24), dpi=300)
         prop = fm.FontProperties(fname='shapes/EBGaramond-Regular.ttf')

         ax_mainland = plt.subplot2grid((4,4),(0,0), colspan=4)
         plt.text(0.5, 1.05, "States Colored by Baby Name Uniqueness",
                  horizontalalignment='center',
                  fontproperties=prop, fontsize=25,
                  transform = ax_mainland.transAxes)
         ax_mainland.axis("off")

         #map for the lower 48
         colors = {}
         state_names = []
         for shapedict in map_lower_48.states_info:
             state_name = shapedict['NAME']
             if state_name != 'Puerto Rico':
                 score = new_dict[state_name]
                 colors[state_name] = cmap(np.sqrt(score))
                 #maps the colors to the states in a dictionary called "colors"
             state_names.append(state_name)

         for nshape, seg in enumerate(map_lower_48.states):
             if state_names[nshape] != 'Puerto Rico':
                 color = rgb2hex(colors[state_names[nshape]])
                 mainland_poly = Polygon(seg, facecolor=color, edgecolor=color)
                 ax_mainland.add_patch(mainland_poly)
         map_lower_48.drawstates(color=color, linewidth=0)


         #Repeat for Alaska
         colors = {}
         ax_alaska = plt.subplot2grid((4,4), (1,1), colspan = 1)
         ax_alaska.set_title('Alaska', fontproperties=prop, fontsize=25)
         ax_alaska.axis("off")

         for shapedict in m_alaska.states_info:
             score = new_dict['Alaska']
             colors['Alaska'] = cmap(math.sqrt(score))
```

```
    state_names.append('Alaska')

for nshape, seg in enumerate(m_alaska.states):
    color_alaska = rgb2hex(colors['Alaska'])
    alaska_poly = Polygon(seg, facecolor=color_alaska, edgecolor=color_alaska)
    ax_alaska.add_patch(alaska_poly)
m_alaska.drawstates(color=color, linewidth=0)

#Repeat for Hawaii
colors = {}
ax_hawaii = plt.subplot2grid((4,4), (1,2), colspan = 1)
ax_hawaii.set_title('Hawaii', fontproperties=prop, fontsize=25)
ax_hawaii.axis("off")

for shapedict in m_hawaii.states_info:
    score = new_dict['Hawaii']
    colors['Hawaii'] = cmap(math.sqrt(score))
    state_names.append('Hawaii')

for nshape, seg in enumerate(m_hawaii.states):
    color_hawaii = rgb2hex(colors['Hawaii'])
    hawaii_poly = Polygon(seg, facecolor=color_hawaii, edgecolor=color_hawaii)
    ax_hawaii.add_patch(hawaii_poly)

m_hawaii.drawstates(color=color_hawaii, linewidth=0)

plt.savefig('shapes\state_name_scores.png', dpi=300) #bbox_inces='tight')
plt.show()
```

States Colored by Baby Name Uniqueness

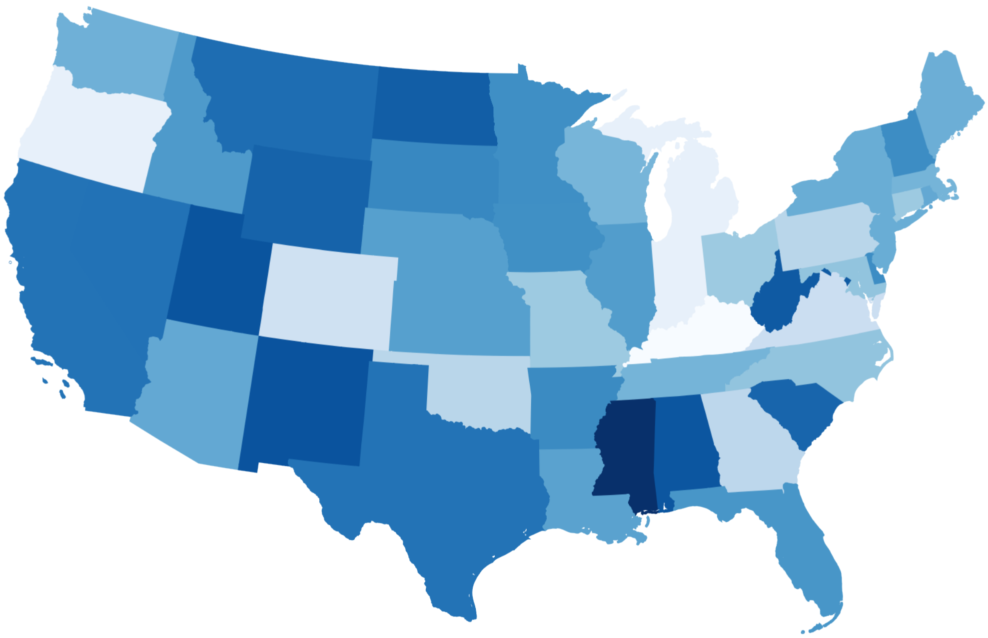


Alaska

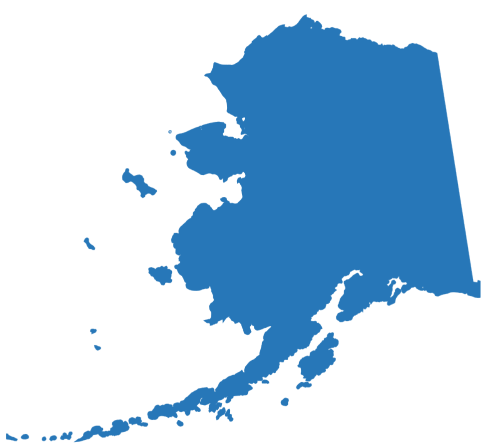


Hawaii

States Colored by Baby Name Uniqueness

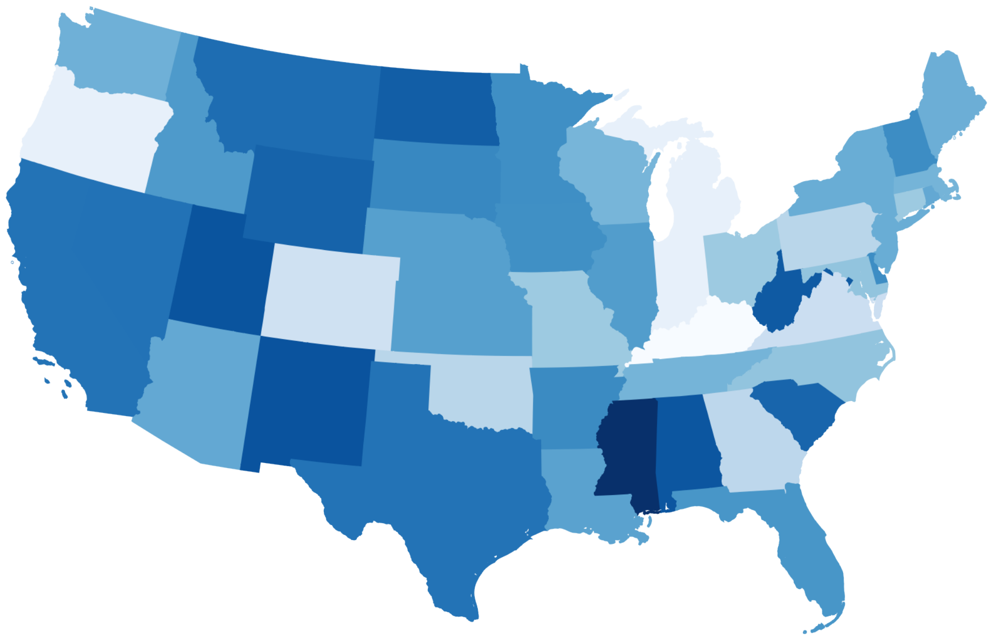


Alaska

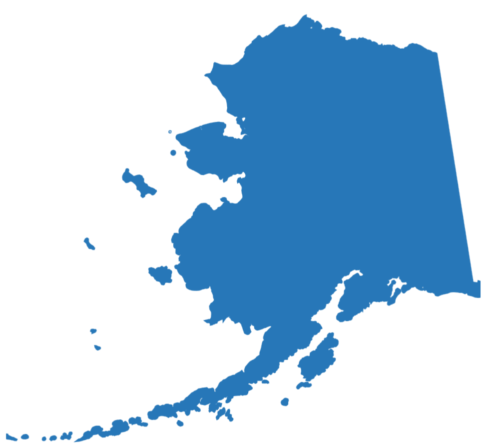


Hawaii

States Colored by Baby Name Uniqueness



Alaska



Hawaii