notebook

February 16, 2022

0.1 1. Sound it out!

Grey and Gray. Colour and Color. Words like these have been the cause of many heated arguments between Brits and Americans. Accents (and jokes) aside, there are many words that are pronounced the same way but have different spellings. While it is easy for us to realize their equivalence, basic programming commands will fail to equate such two strings.

More extreme than word spellings are names because people have more flexibility in choosing to spell a name in a certain way. To some extent, tradition sometimes governs the way a name is spelled, which limits the number of variations of any given English name. But if we consider global names and their associated English spellings, you can only imagine how many ways they can be spelled out.

One way to tackle this challenge is to write a program that checks if two strings sound the same, instead of checking for equivalence in spellings. We'll do that here using fuzzy name matching.

[78]:
```
# Importing the fuzzy package
import fuzzy

# Exploring the output of fuzzy.nysiis
print(fuzzy.nysiis("Hello"))

# Testing equivalence of similar sounding words
print(fuzzy.nysiis("colour")==fuzzy.nysiis("color"))
```

```
HAL
True
```

0.2 2. Authoring the authors

The New York Times puts out a weekly list of best-selling books from different genres, and which has been published since the 1930's. We'll focus on Children's Picture Books, and analyze the gender distribution of authors to see if there have been changes over time. We'll begin by reading in the data on the best selling authors from 2008 to 2017.

[79]:
```
# Importing the pandas module
import pandas as pd

# Reading in datasets/nytkids_yearly.csv, which is semicolon delimited.
author_df=pd.read_csv('datasets/nytkids_yearly.csv',';')
```

```
# Looping through author_df['Author'] to extract the authors first names
first_name = []
for name in author_df['Author']:
    first_name.append(name.split()[0])




# Adding first_name as a column to author_df
author_df['first_name']=first_name

# Checking out the first few rows of author_df
author_df.head()
```

[79]:

	Year	Book Title	Author	\
0	2017	DRAGONS LOVE TACOS	Adam Rubin	
1	2017	THE WONDERFUL THINGS YOU WILL BE	Emily Winfield Martin	
2	2017	THE DAY THE CRAYONS QUIT	Drew Daywalt	
3	2017	ROSIE REVERE, ENGINEER	Andrea Beaty	
4	2017	ADA TWIST, SCIENTIST	Andrea Beaty	

	Besteller this year	first_name
0	49	Adam
1	48	Emily
2	44	Drew
3	38	Andrea
4	28	Andrea

0.3 3. It's time to bring on the phonics... *again*!

When we were young children, we were taught to read using phonics; sounding out the letters that compose words. So let's relive history and do that again, but using python this time. We will now create a new column or list that contains the phonetic equivalent of every first name that we just extracted.

To make sure we're on the right track, let's compare the number of unique values in the first_name column and the number of unique values in the nysiis coded column. As a rule of thumb, the number of unique nysiis first names should be less than or equal to the number of actual first names.

[80]:
```
# Importing numpy
import numpy as np

# Looping through author's first names to create the nysiis (fuzzy) equivalent
nysiis_name = []
for name in author_df['first_name']:
    nysiis_name.append(fuzzy.nysiis(name))
```

```
# Adding nysiis_name as a column to author_df
author_df["nysiis_name"]=nysiis_name

# Printing out the difference between unique firstnames and unique nysiis_names:
print(len(author_df['first_name'].unique())-len(author_df['nysiis_name'].
 ↪unique()))
```

25

0.4 4. The inbetweeners

We'll use babynames_nysiis.csv, a dataset that is derived from the Social Security Administration's baby name data, to identify author genders. The dataset contains unique NYSIIS versions of baby names, and also includes the percentage of times the name appeared as a female name (perc_female) and the percentage of times it appeared as a male name (perc_male).

We'll use this data to create a list of gender. Let's make the following simplifying assumption: For each name, if perc_female is greater than perc_male then assume the name is female, if perc_female is less than perc_male then assume it is a male name, and if the percentages are equal then it's a "neutral" name.

[81]:
```
# Reading in datasets/babynames_nysiis.csv, which is semicolon delimited.
babies_df = pd.read_csv('datasets/babynames_nysiis.csv',';')
# Looping through babies_df to and filling up gender
gender = []
for i in range(len(babies_df)):
    if babies_df['perc_female'].iloc[i]>babies_df['perc_male'].iloc[i]:
        gender.append("F")
    elif babies_df['perc_female'].iloc[i]==babies_df['perc_male'].iloc[i]:
        gender.append("N")
    else:
        gender.append('M')


# Adding a gender column to babies_df
babies_df["gender"]=gender

# Printing out the first few rows of babies_df
babies_df.head()
```

[81]:

	babynysiis	perc_female	perc_male	gender
0	NaN	62.50	37.50	F
1	RAX	63.64	36.36	F
2	ESAR	44.44	55.56	M
3	DJANG	0.00	100.00	M
4	PARCAL	25.00	75.00	M

0.5 5. Playing matchmaker

Now that we have identified the likely genders of different names, let's find author genders by searching for each author's name in the babies_df DataFrame, and extracting the associated gender.

[82]:
```
# This function returns the location of an element in a_list.
# Where an item does not exist, it returns -1.
def locate_in_list(a_list, element):
    loc_of_name = a_list.index(element) if element in a_list else -1
    return(loc_of_name)

# Looping through author_df['nysiis_name'] and appending the gender of each
# author to author_gender.
author_gender = []
for name in author_df['nysiis_name']:
    if locate_in_list(list(babies_df['babynysiis']),name)==-1:
        author_gender.append("Unknown")
    else:
        a=babies_df['gender'].
 ↪iloc[locate_in_list(list(babies_df['babynysiis']),name)]
        author_gender.append(a)
# Adding author_gender to the author_df
author_df['author_gender']=author_gender
author_df.head()

# Counting the author's genders
author_df['author_gender'].value_counts()
```

[82]:
```
F          395
M          191
Unknown      9
N            8
Name: author_gender, dtype: int64
```

0.6 6. Tally up

From the results above see that there are more female authors on the New York Times best seller's list than male authors. Our dataset spans 2008 to 2017. Let's find out if there have been changes over time.

[83]:
```
# Creating a list of unique years, sorted in ascending order.
years = list(author_df['Year'].unique())
years.sort()


# Initializing lists
males_by_yr = []
females_by_yr = []
```

```
unknown_by_yr = []

# Looping through years to find the number of male, female and unknown authors␣
 ↪per year
for y in years:
    males_by_yr.append(len( author_df[ (author_df['author_gender']=='M') &␣
 ↪(author_df['Year']==y) ]))
    females_by_yr.append(len( author_df[ (author_df['author_gender']=='F') &␣
 ↪(author_df['Year']==y) ]))
    unknown_by_yr.append(len( author_df[␣
 ↪(author_df['author_gender']=='Unknown') & (author_df['Year']==y) ]))

# Printing out yearly values to examine changes over time
for i in range(len(years)):
    print(years[i])
    print("Males:",males_by_yr[i])
    print("Females:",females_by_yr[i])
    print("Unknown:",unknown_by_yr[i])
```

```
2008
Males: 8
Females: 15
Unknown: 1
2009
Males: 19
Females: 45
Unknown: 3
2010
Males: 27
Females: 48
Unknown: 0
2011
Males: 21
Females: 51
Unknown: 1
2012
Males: 21
Females: 46
Unknown: 0
2013
Males: 11
Females: 51
Unknown: 2
2014
Males: 21
```

```
Females: 34
Unknown: 1
2015
Males: 18
Females: 30
Unknown: 0
2016
Males: 25
Females: 32
Unknown: 0
2017
Males: 20
Females: 43
Unknown: 1
```

0.7 7. Foreign-born authors?

Our gender data comes from social security applications of individuals born in the US. Hence, one possible explanation for why there are "unknown" genders associated with some author names is because these authors were foreign-born. While making this assumption, we should note that these are only a subset of foreign-born authors as others will have names that have a match in baby_df (and in the social security dataset).

Using a bar chart, let's explore the trend of foreign-born authors with no name matches in the social security dataset.

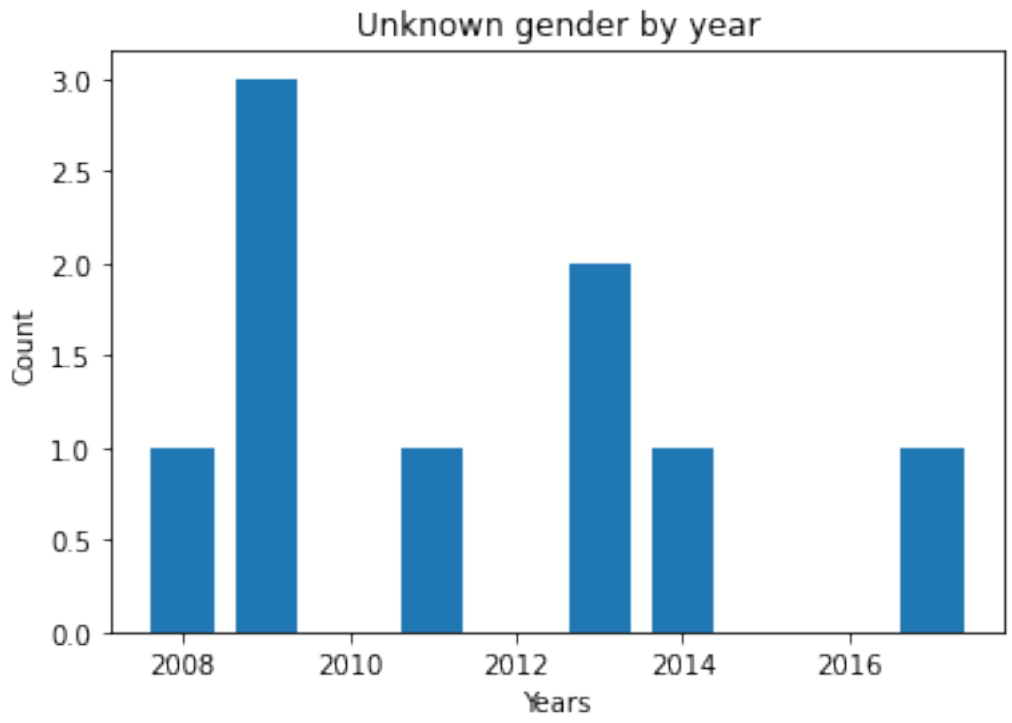
[84]:
```
# Importing matplotlib
import matplotlib.pyplot as plt

# This makes plots appear in the notebook
%matplotlib inline

# Plotting the bar chart
plt.bar(years,unknown_by_yr,width=0.75)

# [OPTIONAL] - Setting a title, and axes labels
plt.title('Unknown gender by year')
plt.xlabel('Years')
plt.ylabel('Count')
```

[84]:
```
Text(0, 0.5, 'Count')
```



0.8 8. Raising the bar

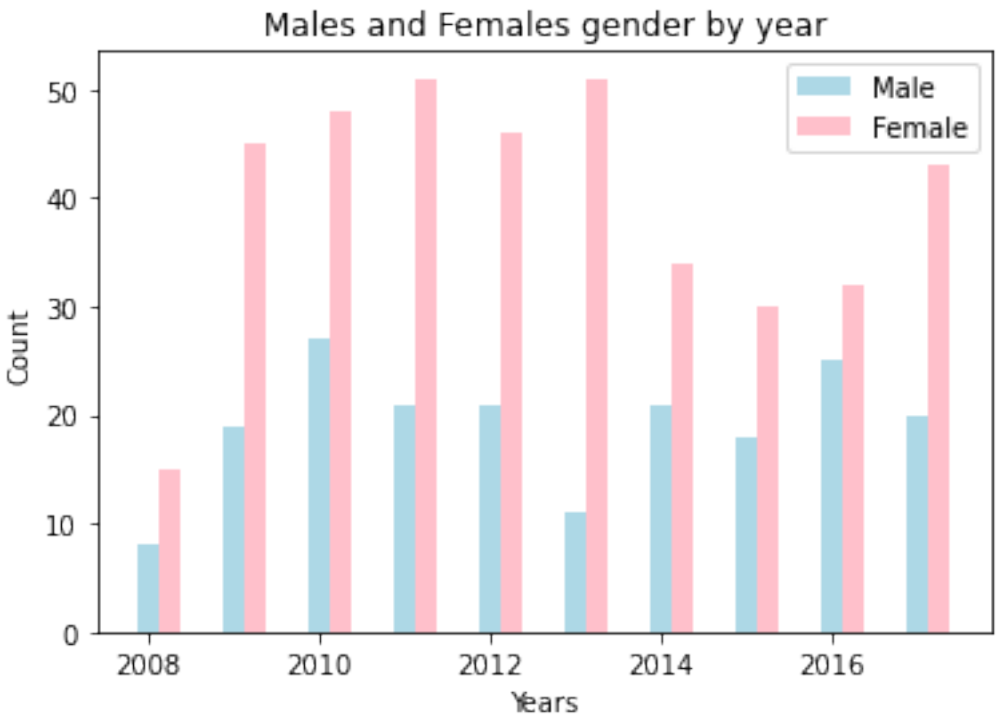
What's more exciting than a bar chart is a grouped bar chart. This type of chart is good for displaying changes over time while also comparing two or more groups. Let's use a grouped bar chart to look at the distribution of male and female authors over time.

[85]:
```
# Creating a new list, where 0.25 is added to each year
years_shifted = [i+0.25 for i in years]

# Plotting males_by_yr by year
plt.bar(years,males_by_yr,width=0.25,color='lightblue',label='Male')
# Plotting females_by_yr by years_shifted
plt.bar(years_shifted,females_by_yr,width=0.25,color='pink',label='Female')

# [OPTIONAL] - Adding relevant Axes labels and Chart Title
plt.title('Males and Females gender by year')
plt.xlabel('Years')
plt.ylabel('Count')
plt.legend(loc='best')
```

[85]:
```
<matplotlib.legend.Legend at 0x7f9932095970>
```


Males and Females gender by year
Male
Female
Count
0
10
20
30
40
50
2008
2010
2012
2014
2016
Years