6th Week Review

Thomas Schwarz, SJ

Topics

- Babynames homework
- Numpy arrays
  - Loading from text
  - Indexing
  - Extracting Information
  - Application: Simple recursion

Babynames

- Tasks:
  - Find / open files, extract year from the file name
  - Find the data in each file and add to a data structure
  - Use the data structure in order to display result
- Note:
  - You could add more files to the directory, so we cannot assume that the years are fixed

Babynames

- Finding all files
  - We are interested in html files in a directory
  - Use os

```
>>> import os
>>> os.listdir('/Users/thomasschwarz/Documents/
My website/Classes/PDS2021/Babynames')
['baby2004.html', 'baby2012.html',
'baby2008.html', 'baby1998.html',
'baby1982.html', 'baby1994.html',
'baby2002.html', 'baby2014.html',
'baby1984.html', 'baby1992.html',
'baby1988.html', 'baby2016.html',
'baby2000.html', 'baby1990.html',
'baby1986.html', 'baby2010.html',
'baby2006.html', 'baby1996.html',
'baby1980.html']
```

Babynames

- We can also filter for the extension
- Please get used to comprehension:
  - It is used in many data science scripts

```
def get_files(directory_name):
    file_names = [file for file in
                  os.listdir(directory_name)
                  if file.endswith('.html')]
    return file_names
```

Babynames

- We need to extract the year from the file name
  - A typical file name

b	a	b	y	2	0	1	2	.	h	t	m	l

- We are looking at:
  - the last four letters before the dot
  - or: letters 4-8

```
def get_year(filename):
    return int(filename.split('.')[0][-4:])
```

Babynames

- Deciding on a data structure
  - We want to answer queries on popularity of a name
  - Or even draw it

```
>>> display(dd, yy, 'thomas')
1980 25
1982 23
1984 25
1986 26
1988 23
1990 26
1992 27
1994 27
1996 28
1998 31
2000 33
2002 36
2004 37
2006 51
2008 52
2010 62
2012 63
2014 54
2016 48
```

```
>>> display(dd, yy, 'emil')
1980 955
1982 995
1984 969
1986 not ranked
1988 not ranked
1990 not ranked
1992 not ranked
1994 not ranked
1996 not ranked
1998 not ranked
2000 not ranked
2002 not ranked
2004 not ranked
2006 not ranked
```

Babynames

- My answer:
  - Get a list of years
  - Get a dictionary that associates a name with a dictionary year —> ranking

Babynames

- Extracting Information from the file
  - Need to look at a file
    - All information is in an html table

```
<tr align="right">
 <td>1</td> <td>Michael</td> <td>Jessica</td>
</tr>
<tr align="right">
 <td>2</td> <td>Christopher</td> <td>Ashley</td>
</tr>
<tr align="right">
 <td>3</td> <td>Matthew</td> <td>Emily</td>
</tr>
<tr align="right">
 <td>4</td> <td>Joshua</td> <td>Samantha</td>
```

Babynames

- Relevant pattern:
  - `<td>rank</td> <td>male_name</td> <td>female_name</td>`
- Split on the empty space!
  - Check that this gives a list of three instances

```
<tr align="right">
 <td>1</td> <td>Michael</td> <td>Jessica</td>
</tr>
<tr align="right">
 <td>2</td> <td>Christopher</td> <td>Ashley</td>
</tr>
<tr align="right">
```

Babynames

```
for line in infile:
    if line.strip().startswith("<td>"):
    items = line.split()
    if len(items)==3:
```

- We extract based on the items

```
        rank = items[0][4:-5]
        male = items[1][4:-5].lower()
        female = items[2][4:-5].lower()
```

- And add / create to the dictionary

```
        if male not in result:
            result[male] = {year: rank}
        else:
            result[male][year] = rank
```

Babynames

- Display:

```
def display(dictionary, years, name):
    if name not in dictionary:
        print('never ranked')
    else:
        for year in years:
            print(year, dictionary[name].get(year,'not ranked'))
```

- Notice: get allows to specify a default value

Babynames

- How to make a plot:

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

dd, yy = create_dictionary('Babynames')
```

- Create a figure and an axis object

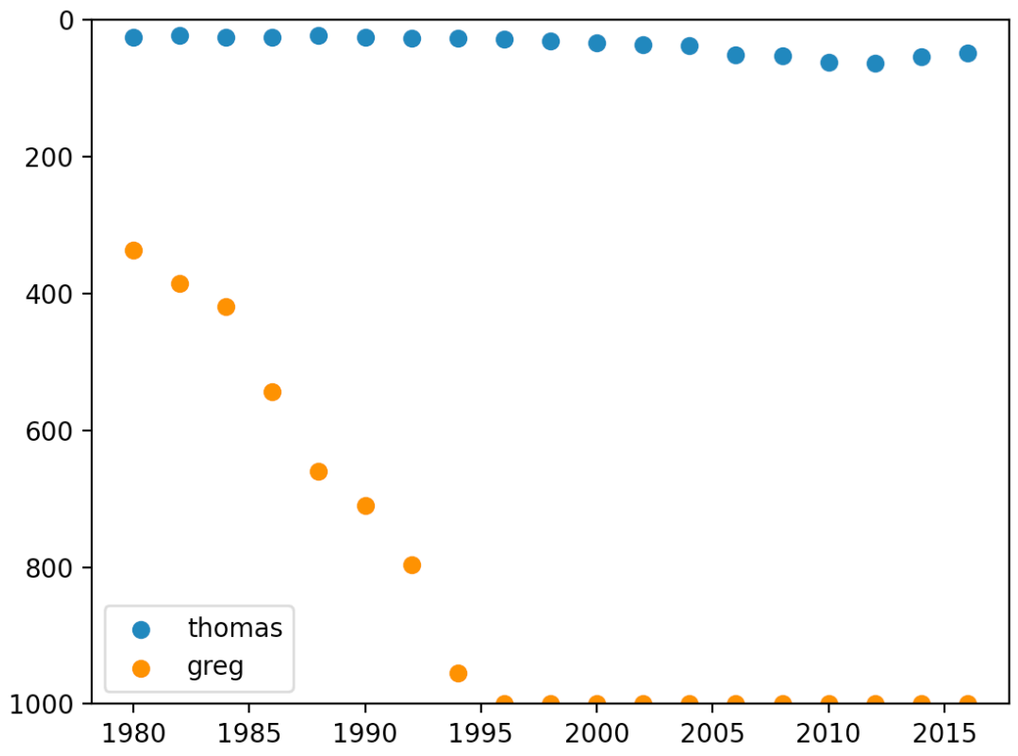
```
fig = plt.figure()
axes = fig.add_subplot()
axes.set_ylim(1000, 0)
```

Babynames

- Provide a scatter plot by calling the dictionary

```
axes.scatter(yy, [int(dd['thomas'].get(year, 1000))
                  for year in yy], label='thomas')
axes.scatter(yy, [int(dd['greg'].get(year, 1000))
                  for year in yy], label='greg')
```

- And plot the values



Penguins

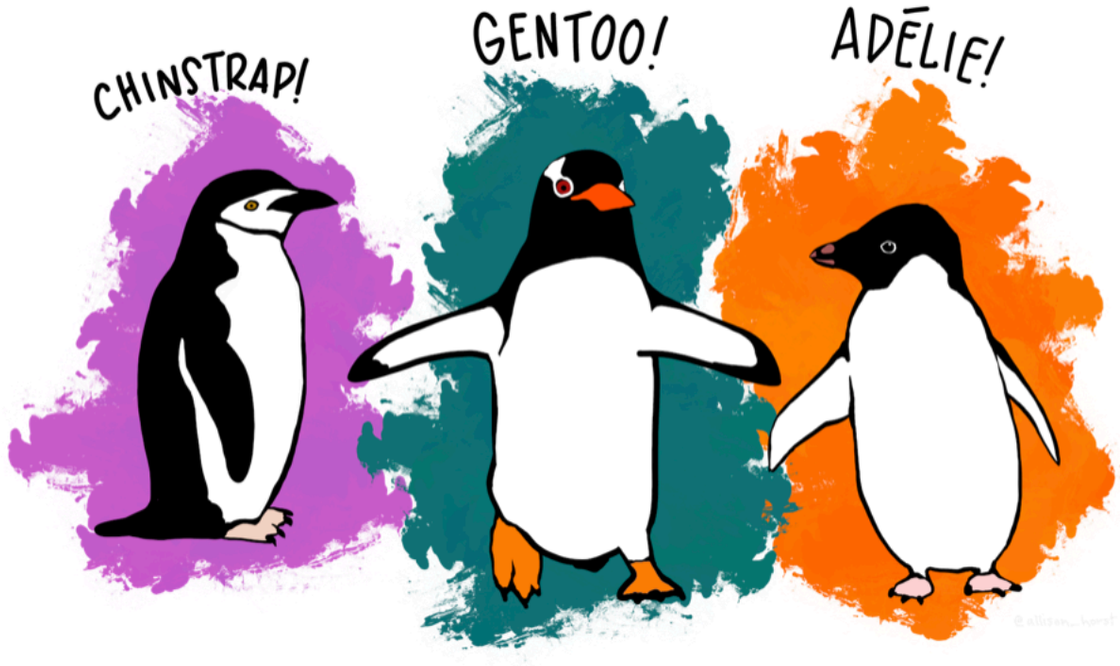
- The Iris data set has been over-analyzed
  - and published in a Eugenics journal
- So, there is an alternative:
  - Allison Horst collected penguin data from
    - Gorman, Williams, and Fraser (2014)
    - Palmer Island Station
- https://allisonhorst.github.io/palmerpenguins/

Penguins

- I publish the data set as a csv file
  - Need to import it with loadtext
    - Which is not easy

penguins

species	island	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g	sex
Adelie	Torgersen	39.1	18.7	181	3750	MALE
Adelie	Torgersen	39.5	17.4	186	3800	FEMALE
Adelie	Torgersen	40.3	18	195	3250	FEMALE
Adelie	Torgersen	NA	NA	NA	NA	NA
Adelie	Torgersen	36.7	19.3	193	3450	FEMALE
Adelie	Torgersen	39.3	20.6	190	3650	MALE
Adelie	Torgersen	38.9	17.8	181	3625	FEMALE
Adelie	Torgersen	39.2	19.6	195	4675	MALE
Adelie	Torgersen	34.1	18.1	193	3475	NA
Adelie	Torgersen	42	20.2	190	4250	NA



Artwork by @allison_horst.

Penguins

- We skip over the island (column 1)
- We code species and sex using converters

```
converters={
    0: lambda astring: (0 if astring == 'Adelie' else 1 if
                        astring == 'Chinstrap' else 2),
    6: lambda astring: (0 if astring == 'MALE' else 1 if
astring == 'FEMALE' else 2)
    }
```

Penguins

- This means column 0 and 6 are integers, the rest is float

```
np.genfromtxt(
    'penguins.csv',
    usecols=(0,2,3,4,5,6),
    dtype = [('type','>i4'), ('cl',float), ('cd', float),
             ('fl',float), ('bm', float), ('sex', '>i4')],
    encoding = None,
    delimiter = ',',
    converters = converters,
    skip_header=1))
```

Penguins

- Now we get rid of rows with a NAN value in them
  - isnan is the numpy function
  - We can use .any / .all to specify whether one or more or all are NAs
- This is our boolean condition

```
~np.isnan(penguins).any(axis=1)
```

  - The tilde indicates negation, because these are the value we want

```
peng = penguins[~np.isnan(penguins).any(axis=1),]
```

Penguins

- Now let's use only Adelie penguins
  - First column has a 0 in it
    - Boolean condition is
      - `peng[:,0]==0`
  - We select:
    - `peng[peng[:,0]==0]`
  - And throw away column 0 (axis = 1 because column)
    - `adelie = np.delete(peng[peng[:,0]==0], 0, axis = 1)`

Penguins

- Now we can calculate things:
  - Mean values by hand
  - Recall that shape[0] is the number of rows

```
def get_mean(array):
    return [np.sum(array[:,i])/array.shape[0] for
            i in range(array.shape[1])]
```

Penguins

- We can also use the np.mean function
  - Where we need to specify the axis, depending on whether we want to get the mean of columns or of rows

```
>>> get_mean(adelie)
[38.79139072847682, 18.346357615894039, 189.95364238410596,
3700.662251655629, 0.5496688741721855]
>>> np.mean(adelie, axis=0)
array([3.87913907e+01, 1.83463576e+01, 1.89953642e+02,
3.70066225e+03, 5.49668874e-01])
```

Penguins

- Calculate the variance $(\sum_{i=1}^{n}(x_i - \bar{x})^2)/n$
- Or better: the sample variance $(\sum_{i=1}^{n}(x_i - \bar{x})^2)/(n-1)$

```
def get_var(array):
    result = []
    for i in range(array.shape[1]):
        mu = np.sum(array[:,i])/array.shape[0]
        var = (np.sum( (array[:,i]-mu)**2 )
               / (array.shape[0]-1))
        result.append(var)
    return result
```

Penguins

- Sample variance: ddof (degrees of freedom) = 1

```
>>> np.var(adelie, axis=0, ddof=1)
array([7.09372539e+00, 1.48023664e+00, 4.27645033e+01,
2.10282892e+05, 3.15849890e-01])
```

Penguins

- Correlation between two columns x and y
- $$\frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{(\sum_{i=1}^{n}(x_i-\bar{x})^2\sum_{i=1}^{n}(y_i-\bar{y})^2}}$$

```
def cov(array, i, j):
    means = np.mean(array, axis=0)
    numerator = np.sum( (array[:,i]-means[i])*
                        (array[:,j]-means[j]) )
    var1 = np.sum( (array[:,i]-means[i])**2 )
    var2 = np.sum( (array[:,j]-means[j])**2 )
    return numerator/sqrt(var1*var2)
```

Penguins

- Simple regression:
  - Two data sets X, Y
  - Assume $Y = \beta X + \alpha + \epsilon$
    - α — intercept
    - β — slope
    - ϵ — error
- Minimize the expectation of ϵ^2

Penguins

- Turns out:
  - $\beta = \frac{\sum_{i=1}^{n}(X_i - \bar{X}) \cdot (Y_i - \bar{Y})}{\sum_{i=1}^{n}(X_i - \bar{X})^2}$
  - $\alpha = \bar{Y} - \beta\bar{X}$

Penguins

- Thus:

```
def slope(array, i, j):
    means = np.mean(array, axis=0)
    numerator = np.sum( (array[:,i]-means[i])*
                        (array[:,j]-means[j]) )
denominator = np.sum( (array[:,i]-means[i])**2 )
return numerator / denominator
```

Penguins

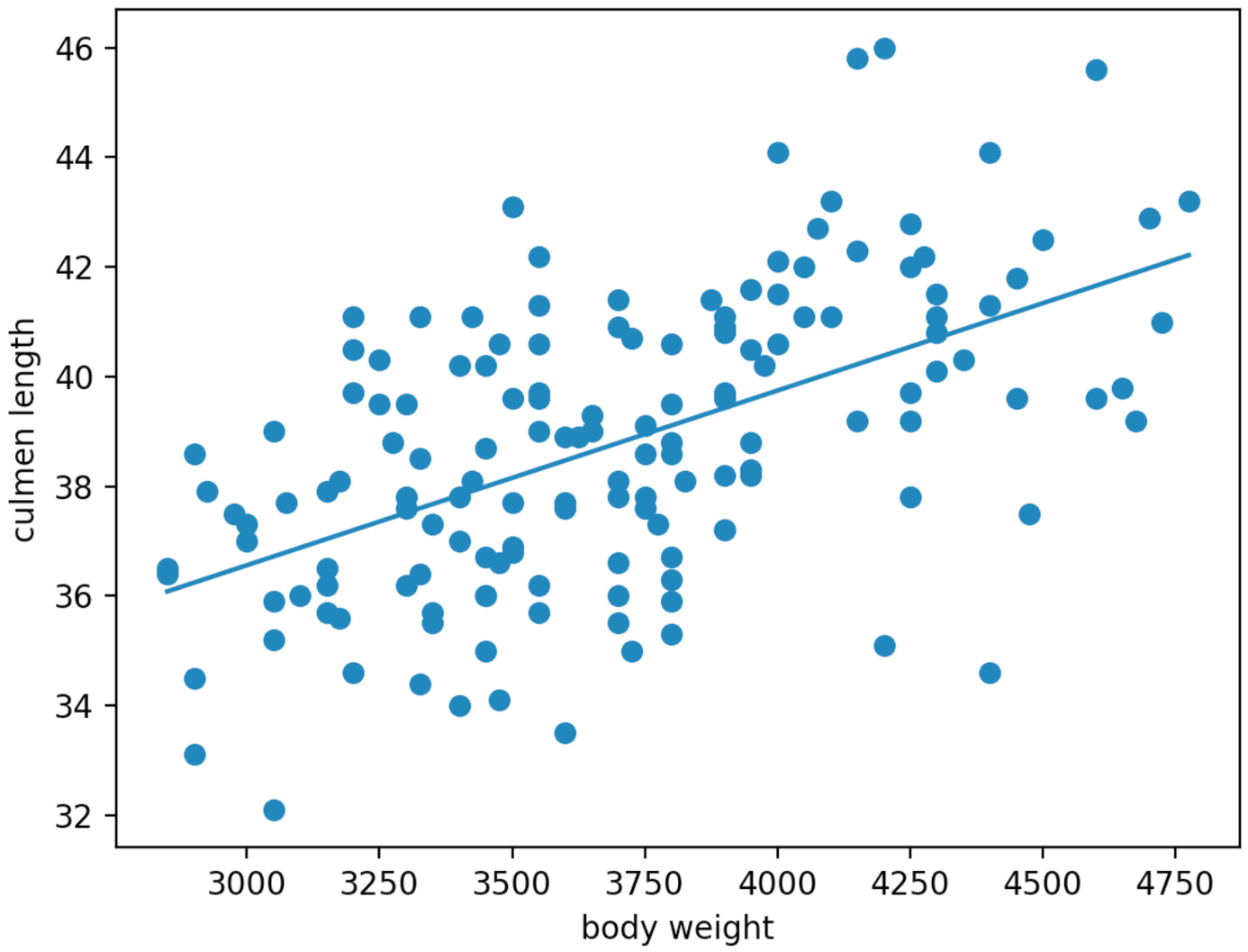
- And:
  - ```
    def intercept(array, i, j):
        beta = slope(array, i, j)
        means = np.mean(array, axis=0)
        return means[j]-beta*means[i]
    ```
    ```

Penguins

- To visualize:
  - Create a prediction function:
    - ```
      def prediction(x):
          return alpha + beta*x
      ```
  - And a lin-space

```
plt.scatter(adelie[:,3], adelie[:,0])
X = np.linspace(np.min(adelie, axis=0)[3],
                np.max(adelie, axis=0)[3],101)
plt.plot(X, prediction(X))
plt.show()
```
```

Penguins


46
44
42
40
38
36
34
32
culmen length
3000
3250
3500
3750
4000
4250
4500
4750
body weight