Panda & Matplotlib

PROF. JACQUES SAVOY
UNIVERSITY OF NEUCHATEL

Working with Tables

A data representation very useful, and ubiquitous.

Basic data structure for spreadsheet and (relational) database system.

1. column: attribute, each with its domain
2. row: a record
3. cell: data type corresponding to the domain provided by the column

Access by columns, rows, or cells.

Mixed the data type across the column (not rows).

Data

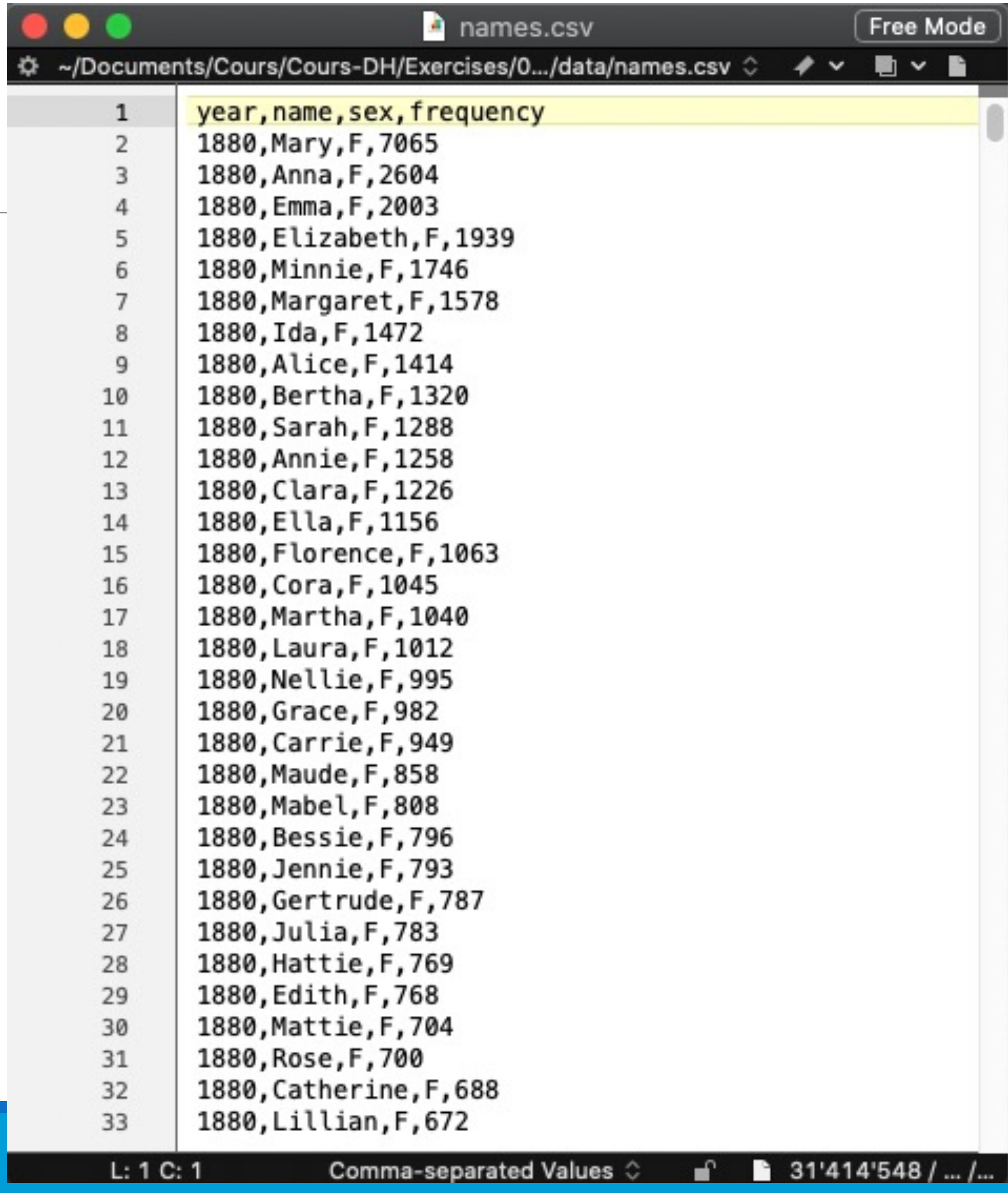
Problem: naming of the children in the US over the years (1880-2015).

United States Social Security Administration

1. the most popular in a given period
2. rate of change over the years
3. gender difference

In the folder `./data/`, we have the following file (in cvs format).

Each row represents the number of children with a given name and gender in the corresponding year.




```
year,name,sex,frequency
1880,Mary,F,7065
1880,Anna,F,2604
1880,Emma,F,2003
1880,Elizabeth,F,1939
1880,Minnie,F,1746
1880,Margaret,F,1578
1880,Ida,F,1472
1880,Alice,F,1414
1880,Bertha,F,1320
1880,Sarah,F,1288
1880,Annie,F,1258
1880,Clara,F,1226
1880,Ella,F,1156
1880,Florence,F,1063
1880,Cora,F,1045
1880,Martha,F,1040
1880,Laura,F,1012
1880,Nellie,F,995
1880,Grace,F,982
1880,Carrie,F,949
1880,Maude,F,858
1880,Mabel,F,808
1880,Bessie,F,796
1880,Jennie,F,793
1880,Gertrude,F,787
1880,Julia,F,783
1880,Hattie,F,769
1880,Edith,F,768
1880,Mattie,F,704
1880,Rose,F,700
1880,Catherine,F,688
1880,Lillian,F,672
```

Import Data

```
>>> import csv                       # the file format of the data
>>> import collections               # better dictionaries
>>> import pandas as pd              # for manipulating table
>>> import numpy as np               # for arrays and matrices

>>> with open('data/names.csv') as infile:   # first row provides the field names
...     data = list(csv.DictReader(infile))

>>> print(data[0])
{'year': '1880', 'name': 'Mary', 'sex': 'F', 'frequency': '7065'}
```

Something weird?

Import Data

```
>>> data = []       # doing by hand (without our panda library)
>>> with open('data/names.csv') as infile:
...     for row in csv.DictReader(infile):   # transform into integer the year and frequency
...         row['frequency'] = int(row['frequency'])
...         row['year'] = int(row['year'])   # each row = a dictionary
...         data.append(row)   # return an array for each record

>>> print(data[0])
{'year': 1880, 'name': 'Mary', 'sex': 'F', 'frequency': 7065}
```

Now year and frequency can be manipulated as integer.

Each row (and not each year) is a dictionary (with four keys).

Data

Some overview information

```
>>> starting_year = min(row['year'] for row in data)
>>> print(starting_year)      # the first year, and keep this variable
    1880
>>> ending_year = max(row['year'] for row in data)
>>> print(ending_year)
    2015                      # the last year
>>> print(ending_year - starting_year + 1)
    136                       # to have an idea about the time span
```

Data Manipulation

Need to select the useful names = names appearing at least ten time in a given year.

Step 1: create counter objects to store counts of girl and boy names

```
>>> name_counts_girls = collections.Counter()    # better dictionary
>>> name_counts_boys  = collections.Counter()
```

Step 2: iterate over the data and increment the counters.

For each year, the number of girls and boys

```
>>> for row in data:
        if row['sex'] == 'F':
            name_counts_girls[row['year']] += 1
        else:
         name_counts_boys[row['year']] += 1
```

Data Manipulation

Step 3: Loop over all years and assert the presence of at least 10 girl and boy names

```
>>> for year in range(starting_year, ending_year + 1):
      assert name_counts_girls[year] >= 10
      assert name_counts_boys[year] >= 10
```

Good. We have two counters with, for each year, the number of girls and boys.
Any variation requires a modification of the code

```
>>> name_counts_girls
   Counter({2007: 20553, 2008: 20442, 2009: 20168, 2006: 20044, 2010:
   19803, 2011: 19548, 2012: 19476, 2013: 19202, 2005: 19177, 2014:
   19149, 2015: 18992, …
>>> name_counts_boys[2000]
   12110
```

Data Manipulation

Trying to do the same procedure but with Panda (and to see why Panda is useful).

```
>>> df = pd.read_csv('data/names.csv')
```
Panda offers also a read for SQL, HTML, JSON, Excel

```
>>> df.head(n=5)
```
with Panda with obtain a `DataFrame` or df (for two dim. object)

```
    year        name sex  frequency
 0  1880        Mary   F       7065
 1  1880        Anna   F       2604
 2  1880        Emma   F       2003
 3  1880   Elizabeth   F       1939
 4  1880      Minnie   F       1746
```
Panda discovers that year and frequency are integers

Selecting one row, Panda will create a `Series` object

Data Selection

Verify the data type (inherited from NumPy)

```
>>> print(df.dtypes)
year              int64
name             object
sex              object
frequency         int64
dtype: object
```

Select names (`df['name']`) and return the the first 5

```
>>> df['name'].head()
0         Mary
1         Anna
2         Emma
3    Elizabeth
4       Minnie
Name: name, dtype: object
```

Select one column (to have a `Serie` object)

Return a Numpy representation of Dataframe

```
>>> print(type(df['sex'].values))
   <class 'numpy.ndarray'>
```

Data Selection

Select two columns

Returns the first 5

From an array with the selected fields
[firstFieldName, secondFieldName]

```
>>> df[['name', 'sex']].head()
```

```
         name  sex
0        Mary    F
1        Anna    F
2        Emma    F
3   Elizabeth    F
4     Minnie    F
```

Select from the 3rd row to the 6th

```
>>> df.iloc[3:7]
```

```
   year       name  sex   frequency
3  1880  Elizabeth    F        1939
4  1880     Minnie    F        1746
5  1880   Margaret    F        1578
6  1880        Ida    F        1472
```

Data Selection

Select only the 30th row (all fields)

```
>>> df.iloc[30]

year                1880
name           Catherine
sex                    F
frequency            688
Name: 30, dtype: object
```

Each `DataFrame` has an *index* (row) and column attributes

```
>>> print(df.columns)

Index(['year', 'name', 'sex', 'frequency'],
dtype='object')

>>> print(df.index)     # and the index
RangeIndex(start=0, stop=1858436, step=1)
```

Import Data

A simple solution is to specify the index when loading the data.

```
>>> df = pd.read_csv('data/names.csv', index_col=0)
>>> df.head()
             name sex  frequency
year
1880         Mary   F       7065
1880         Anna   F       2604
1880         Emma   F       2003
1880    Elizabeth   F       1939
1880       Minnie   F       1746
```

Import Data

```
>>> df = pd.read_csv('data/names.csv')   # without specifying an index, one is generated
>>> df = df.set_index('year')            # generate an index with the field year
>>> df.head()
           name  sex  frequency
year
1880       Mary    F       7065
1880       Anna    F       2604
1880       Emma    F       2003
1880  Elizabeth    F       1939
1880     Minnie    F       1746
```

Data Selection

To select a row, use the `loc` attribute (and the selected *value*(s))

```
>>> df.loc[1899].head()
          name sex  frequency
year
1899      Mary   F      13172
1899      Anna   F       5115
1899     Helen   F       5048
1899  Margaret   F       4249
1899      Ruth   F       3912
```

To select a row (by value), and only one field
Return the first 5

```
>>> df.loc[1921, 'name'].head()
year
1921        Mary
1921     Dorothy
1921       Helen
1921    Margaret
1921        Ruth
Name: name, dtype: object
```

Data Selection

Select the 10th row, and the 2nd column

Use the `iloc` attribute

```
>>> print(df.iloc[10, 2])
1258
```

Select a given index value, and two columns

Return the first 5

```
>>> df.loc[1921, ['name', 'sex']].head()
          name sex
year
1921      Mary   F
1921   Dorothy   F
1921     Helen   F
1921  Margaret   F
1921      Ruth   F
```

Data Selection

Select two years and two columns, return the first 5 names
(but for 1967, we have 5 or more rows)

```
>>> df.loc[[1921, 1967], ['name', 'sex']].head()
           name  sex
year
1921       Mary    F
1921    Dorothy    F
1921      Helen    F
1921   Margaret    F
1921       Ruth    F
```

Data Selection

Select two years and two columns, return the last 5 names

```
>>> df.loc[[1921, 1967], ['name', 'sex']].tail()
          name sex
year
1967  Zbigniew   M
1967   Zebedee   M
1967      Zeno   M
1967     Zenon   M
1967       Zev   M
```

Select two years with `iloc`, 3rd and the 10th

```
>>> df.iloc[[3, 10], [0, 2]]
           name  frequency
year
1880  Elizabeth       1939
1880      Annie       1258
```

Data Selection

Select a sequence of rows and the last two columns

Return the first 5

```
>>> df.iloc[1000:1100, -2:].head()
      sex  frequency
year
1880    M        305
1880    M        301
1880    M        283
1880    M        274
1880    M        271
```

Behind we have the NumPy data type. Thus we can access them as proposed by NumPy.

```
>>> mydata = df.values
>>> mydata_slice = array[1000:1100,-2:]
>>> array_slice
array([['M', 305],
       ['M', 301],
       ['M', 283],
        ....
```

Data Access

Transform a data frame to extract the most popular names (our *popularity* index).

From a `DataFrame`, use the given column (here `frequency`) to sort the row from the largest value to the smallest one.

Reset the index attribute.

Extract the top 20 `names` having the largest frequencies.

```
>>> def df2ranking(df, rank_col='frequency', cutoff=20):
...     df = df.sort_values(by=rank_col, ascending=False)
...     df = df.reset_index()
...     return df['name'][:cutoff]    # return only the first ones
```

Now we can use this function…

Data Access

```
>>> girl_ranks, boy_ranks = [], []
>>> for year in df.index.unique():     # consider only the distinct values
...     for sex in ('F', 'M'):         # split according to the gender
...         if sex == 'F':
...             year_df = df.loc[year]  # select the names of a given year
...             ranking = df2ranking(year_df.loc[year_df['sex'] == sex])
...             ranking.name = year
...             girl_ranks.append(ranking)  # a keep only the top 20
...         else:
...             year_df = df.loc[year]
...             ranking = df2ranking(year_df.loc[year_df['sex'] == sex])
...             ranking.name = year
...             boy_ranks.append(ranking)
```

Data Selection

```
>>> girl_ranks = pd.DataFrame(girl_ranks)  # generate two DataFrame
>>> boy_ranks  = pd.DataFrame(boy_ranks)

>>> girl_ranks
          0       1       2          3          4  ...     18        19
1880   Mary    Anna    Emma  Elizabeth     Minnie  ...  Grace    Carrie
1881   Mary    Anna    Emma  Elizabeth   Margaret  ...  Laura    Carrie
1882   Mary    Anna    Emma  Elizabeth     Minnie  ... Nellie     Maude
1883   Mary    Anna    Emma  Elizabeth     Minnie  ...  Laura     Ethel
1884   Mary    Anna    Emma  Elizabeth     Minnie  ...   Ella     Mabel
...     ...     ...     ...        ...        ...  ...    ...       ...
2015   Emma  Olivia  Sophia        Ava  Isabella  ...  Grace  Victoria
[136 rows x 20 columns]
```

Data Selection

We can generate a simpler solution (than the previous verbose one)

```
>>> df.reset_index().groupby('name')['year'].median().head()
name
Aaban          2011.5
Aabha          2013.0
Aabid          2003.0
Aabriella      2014.0
Aada           2015.0
Name: year, dtype: float64
```

1. We need the year as a column. Thus generate a new index.
2. `groupby()` aggregate a column (`name`) take the values of `year` and compute the `median()`.
3. We have the year median for each name

But we need to sort them (by default ascending order)

Data Selection

```
>>> df.reset_index().groupby('name')['year'].median().sort_values().head()
name
Roll       1881.0    #  a sequence of old names
Zilpah     1881.0
Crete      1881.5
Sip        1885.0
Ng         1885.0
Name: year, dtype: float64
```

The more compact version is the following (`level=0` means according to the index)

```
>>> boy_ranks = df.loc[df.sex == 'M'].groupby(level=0).apply(df2ranking)
>>> girl_ranks = df.loc[df.sex == 'F'].groupby(level=0).apply(df2ranking)
```

For each year, we have the top 20 most frequent names (using both `groupby()` and `apply()`).

Data Selection

```
>>> boy_ranks.head()    # keep this in mind... we'll come back here.
name     0        1       2         3         4       5        6        7        8
     ...      11       12       13    14       15       16          17      18          19
year
1880  John  William  James  Charles  George  Frank  Joseph  Thomas  Henry  ...  Harry  Walter
     Arthur  Fred  Albert  Samuel    David  Louis      Joe
1881  John  William  James  George  Charles  Frank  Joseph  Henry  Thomas  ...  Harry  Walter
     Arthur  Fred  Albert  Samuel    David  Louis  Charlie
1882  John  William  James  George  Charles  Frank  Joseph  Thomas  Henry  ...  Harry  Walter
      Arthur  Fred  Albert  Samuel    Louis  David  Clarence
1883  John  William  James  Charles  George  Frank  Joseph  Henry  Robert  ...  Harry  Walter
     Arthur  Fred  Albert  Samuel  Clarence  Louis    David
```

Data Selection

To have a better understanding, it is better to work with a smaller example

Consider this set of persons, with the gender, and a (random) value

```
>>> data = pd.DataFrame({'name': ['Jennifer', 'Claire', 'Matthew', 'Richard',
...                              'Richard', 'Claire', 'Matthew', 'Jennifer'],
...                             'sex': ['F', 'F', 'M', 'M', 'M', 'F', 'M', 'F'],
...                             'value': np.random.rand(8)})
```

Data Selection

To have a better understanding, we're working with the following example

```
>>> data
       name sex     value
0  Jennifer   F  0.142430
1    Claire   F  0.747811
2   Matthew   M  0.380427
3   Richard   M  0.979738
4   Richard   M  0.547155
5    Claire   F  0.498054
6   Matthew   M  0.833606
7  Jennifer   F  0.282659
```

Data Selection

```
>>> grouped = data.groupby('sex')    # An object with the two genders

>>> grouped.get_group('F')
       name sex     value
0  Jennifer   F  0.142430
1    Claire   F  0.747811
5    Claire   F  0.498054
7  Jennifer   F  0.282659
```

Data Selection

```
>>> for grouper, group in grouped:
...     print('grouper:', grouper)
...     print(group)
grouper: F
       name sex     value
0  Jennifer   F  0.142430
1    Claire   F  0.747811
5    Claire   F  0.498054
7  Jennifer   F  0.282659

grouper: M
 ...
```

Display the information available for each gender

```
grouper: M
      name sex     value
2   Matthew   M  0.380427
3   Richard   M  0.979738
4   Richard   M  0.547155
6   Matthew   M  0.833606
```

Data Selection

```
>>> grouped.sum()
          value
sex
F      1.670954
M      2.740926
```

Compute the sum/mean for each group (single numeric attribute in each group).

```
>>> grouped['value'].mean()
sex
F      0.417739
M      0.685231
Name: value, dtype: float64
```

Same effect as `grouped.mean()` but we specify the attribute.

Data Selection

```
>>> grouped['value'].agg(np.sum)
sex
F     1.670954
M     2.740926

Name: value, dtype: float64
```

Compute also the mean for each group with the `agg()` function.

As argument , we can provide the NumPy function (`np.mean`) or the function name (e.g., `size, mean`).

```
>>> grouped['value'].agg(['size', 'mean'])
     size      mean
sex
F       4    0.417739
M       4    0.685231
```

Data Selection

```
>>> def combine_unique(names):          # write your own function
...     return ' '.join(set(names))     # build as a set for unique names

>>> grouped['name'].agg(combine_unique)  # apply with agg() function
sex
F     Jennifer Claire
M     Richard Matthew
Name: name, dtype: object
```

Set

```
>>> A = {'Frank', 'Henry', 'James', 'Richard'}     # Operations on sets
>>> B = {'Ryan', 'James', 'Logan', 'Frank'}

>>> diff_1 = A.difference(B)
>>> diff_2 = A - B                                  # Binary operators on sets

>>> print(f"Difference of A and B = {diff_1}")
   Difference of A and B = {'Henry', 'Richard'}

>>> print(f"Difference of A and B = {diff_2}")
   Difference of A and B = {'Henry', 'Richard'}

>>> print(len(A.difference(B)))
   2
```

Set

```
>>> A = np.array([{'Isaac', 'John', 'Mark'}, {'Beth', 'Rose', 'Claire'}])

>>> B = np.array([{'John', 'Mark', 'Benjamin'}, {'Sarah', 'Anna', 'Susan'}])

>>> C = A - B

>>> print(C)

   [{'Isaac'} {'Rose', 'Beth', 'Claire'}]
```

Data Selection

Back to our problem. We have the popularity rankings for names for the two genders.

Now how can we measure the change / turnover form one year to the next.

Solution 1. (without Panda)

We have this

```
>>> boy_ranks.head()    # See Slide #25
name     0        1       2        3         4      5        6       7       8    ...
year
1880  John  William  James  Charles   George  Frank  Joseph  Thomas   Henry  ...
1881  John  William  James   George  Charles  Frank  Joseph   Henry  Thomas  ...
1882  John  William  James   George  Charles  Frank  Joseph  Thomas   Henry  ...
1883  John  William  James  Charles   George  Frank  Joseph   Henry  Robert  ...
```

Data Selection

We do a simple for-loop over the index (the years)

Compare the names in year *t* with those of year *t-1*.

Count the number of names that differ between the two years.

```
>>> def turnover(df):
...     df = df.apply(set, axis=1)
...     turnovers = {}
...     for year in range(df.index.min() + 1, df.index.max() + 1):
...         name_set, prior_name_set = df.loc[year], df.loc[year - 1]
...         turnovers[year] = len(name_set.difference(prior_name_set))
...     return pd.Series(turnovers)
```

Data Selection

```
>>> boy_ranks.apply(set, axis=1).head()     # inside the function
year
1880    {David, John, Joe, Fred, Harry, Louis, Charles...
1881    {David, John, Fred, Harry, Louis, Charles, Sam...
1882    {David, John, Fred, Harry, Louis, Charles, Sam...
1883    {David, John, Fred, Harry, Louis, Charles, Sam...
1884    {John, Fred, Harry, Louis, Charles, Samuel, He...
dtype: object
```

```
>>> boy_turnover = turnover(boy_ranks)
>>> boy_turnover.head()
1881    1    <- one difference between two years (1880-1881)
1882    1
1883    0
1884    1
1885    0
dtype: int64
```

Data Selection

Now how can we measure the change / turnover form one year to the next.

Solution 2. (with Panda)

```
>>> def turnover(df):
...     df = df.apply(set, axis=1)
...     return (df.iloc[1:] - df.shift(1).iloc[1:]).apply(len)

>>> s = boy_ranks.apply(set, axis=1)

>>> s.shift(1).head()
year
1880                                                        NaN
1881    {David, John, Joe, Fred, Harry, Louis, Charles...
1882    {David, John, Fred, Harry, Louis, Charles, Sam...
1883    {David, John, Fred, Harry, Louis, Charles, Sam...
1884    {David, John, Fred, Harry, Louis, Charles, Sam...
dtype: object
```

Data Selection

```
>>> differences = (s.iloc[1:] - s.shift(1).iloc[1:])
>>> differences.head()
year
1881      {Charlie}
1882     {Clarence}
1883             {}
1884       {Grover}
1885             {}
dtype: object
```

the same name in the top 20 except one "Charlie"

```
>>> turnovers = differences.apply(len)
>>> turnovers.head()
year
1881    1
1882    1
1883    0
1884    1
1885    0
dtype: int64
```

Data Selection

To verify this

```
>>> s[1880]

{'Harry', 'Louis', 'James', 'Albert', 'Robert', 'Walter', 'Fred',
'Joseph', 'Arthur', 'John', 'William', 'Charles', 'Joe', 'Samuel',
'Thomas', 'Henry', 'Frank', 'David', 'George', 'Edward'}

>>> s[1881]

{'Charlie', 'Harry', 'Louis', 'James', 'Albert', 'Robert', 'Walter',
'Fred', 'Joseph', 'Arthur', 'John', 'William', 'Charles', 'Samuel',
'Thomas', 'Henry', 'Frank', 'David', 'George', 'Edward'}

>>> s[1881]-s[1880]

{'Charlie'}
```

Data Selection

```
>>> boy_turnover = turnover(boy_ranks)
>>> boy_turnover.head()
year
1881    1
1882    1
1883    0
1884    1
1885    0
dtype: int64
```

```
>>> girl_turnover = turnover(girl_ranks)
>>> girl_turnover.head()
year
1881    0
1882    2
1883    1
1884    1
1885    0
dtype: int64
```

Data Visualization

Generate a graph indicating the number of differences between two years.

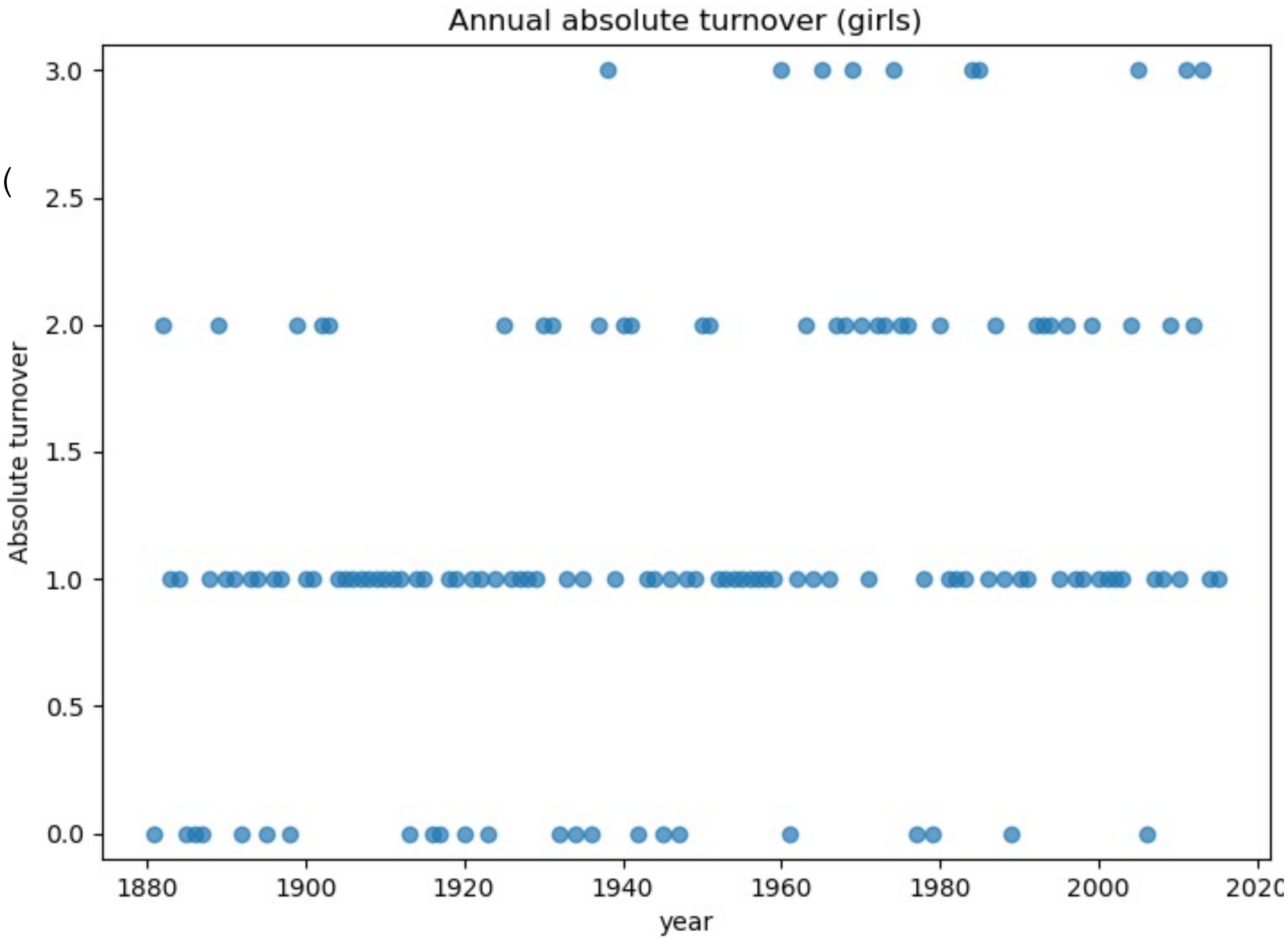
```
>>> ax = girl_turnover.plot(
...       style='o',
...       ylim=(-0.1, 3.1),
...       alpha=0.7,
...       title='Annual absolute turnover (girls)'
...       )
>>> ax.set_ylabel("Absolute turnover");
   Text(0, 0.5, 'Absolute turnover')
>>> plt.show()
```

Data Visualiza

```
>>> ax = girl_turnover.plot(
...      style='o',
...      ylim=(-0.1, 3.1),
...      alpha=0.7,
...      title='Annual
absolute turnover (girls)'
...      )
```

```
>>> ax.set_ylabel("Absolute
turnover");
Text(0, 0.5, 'Absolute
turnover')
```

```
>>> plt.show()
```

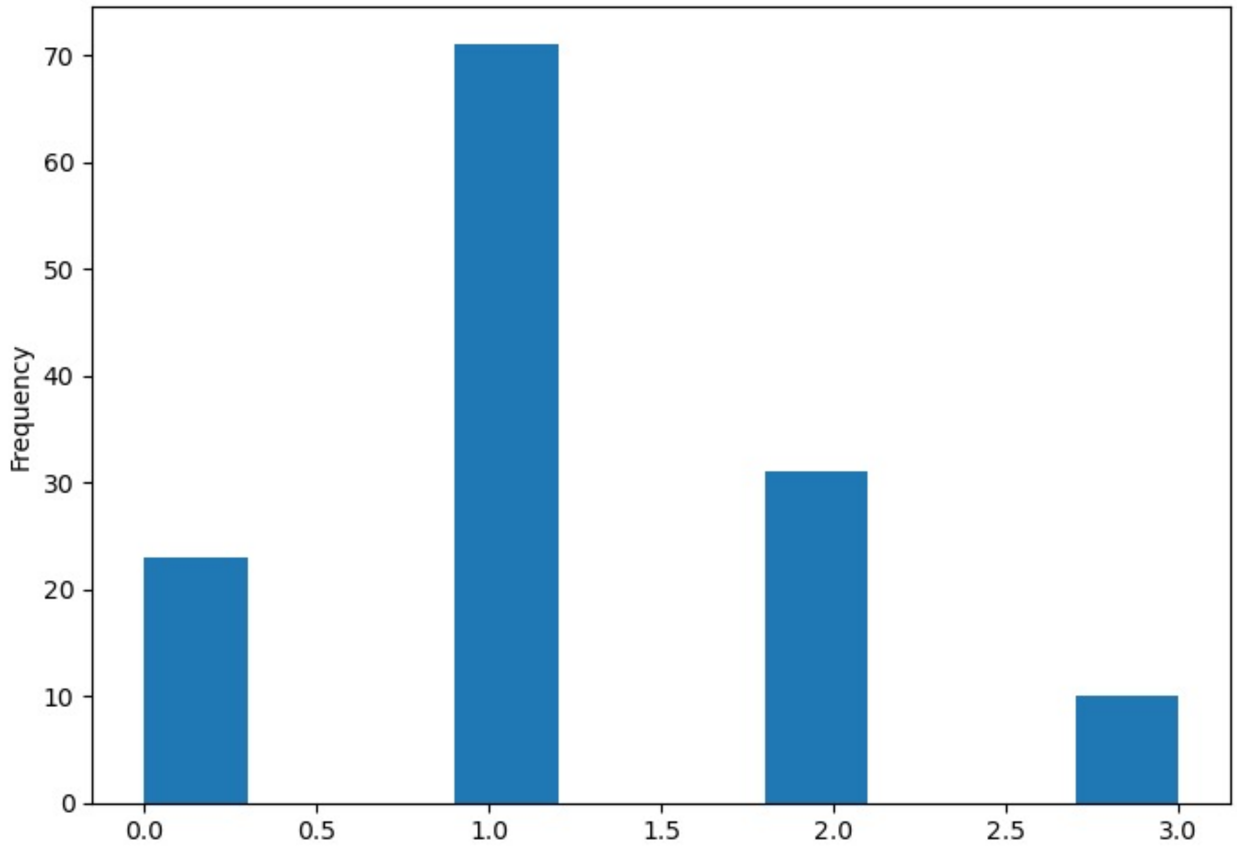


Data Selection

```
>>>
girl_turnover.plot(kind='hist')
<AxesSubplot:ylabel='Frequency'>

>>> plt.show()
```

Between two years, only a few changes in the top 20 most frequently used names.

But how can we draw a line to represent the evolution over the years (instead of having only one point per year, jumping from one integer to the next).



Data Selection

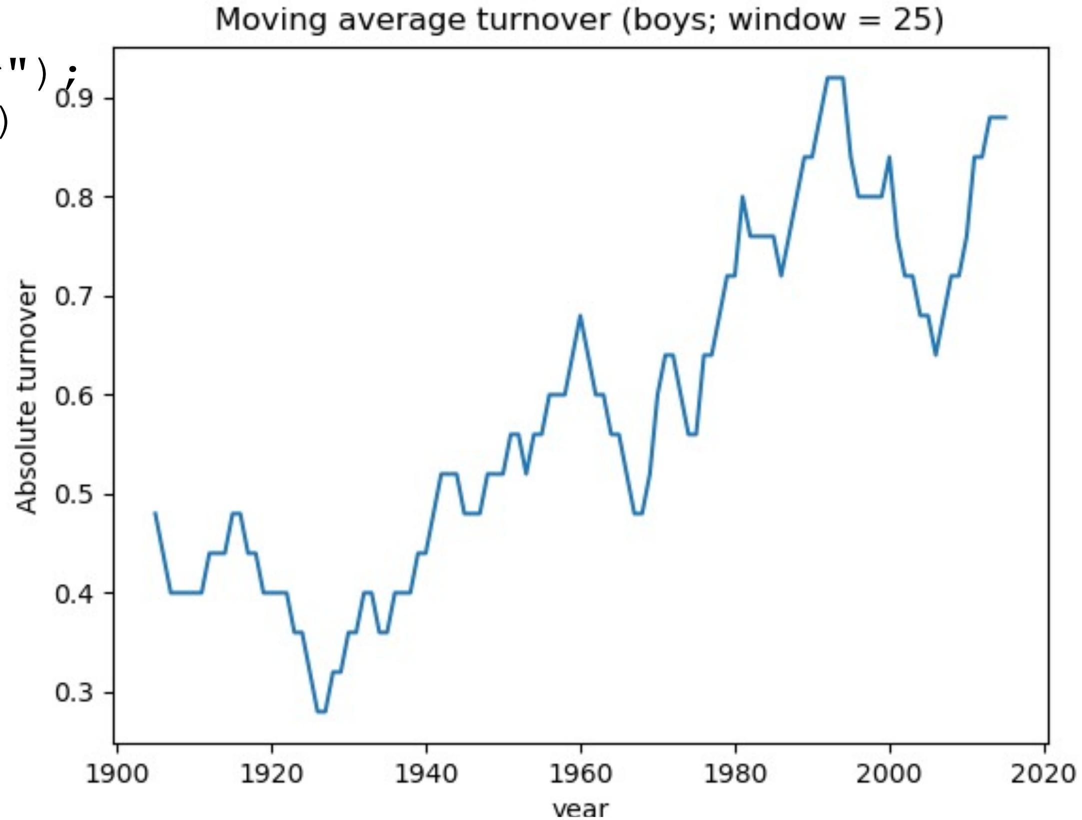
```
>>> girl_rm = girl_turnover.rolling(25).mean()

>>> ax = girl_rm.plot(title="Moving average turnover (girls; window = 25)")

>>> ax.set_ylabel("Absolute turnover");
   Text(0, 0.5, 'Absolute turnover')

>>> plt.show()
```

Applying the moving average

Few changes up to 1960.

Data Selection

```
>>> boy_rm = boy_turnover.rolling(25).mean()

>>> ax = boy_rm.plot(title="Moving average turnover (boys; window =
25)")

>>> ax.set_ylabel("Absolute turnover");
    Text(0, 0.5, 'Absolute turnover')

>>> plt.show()
```



Data Visualization

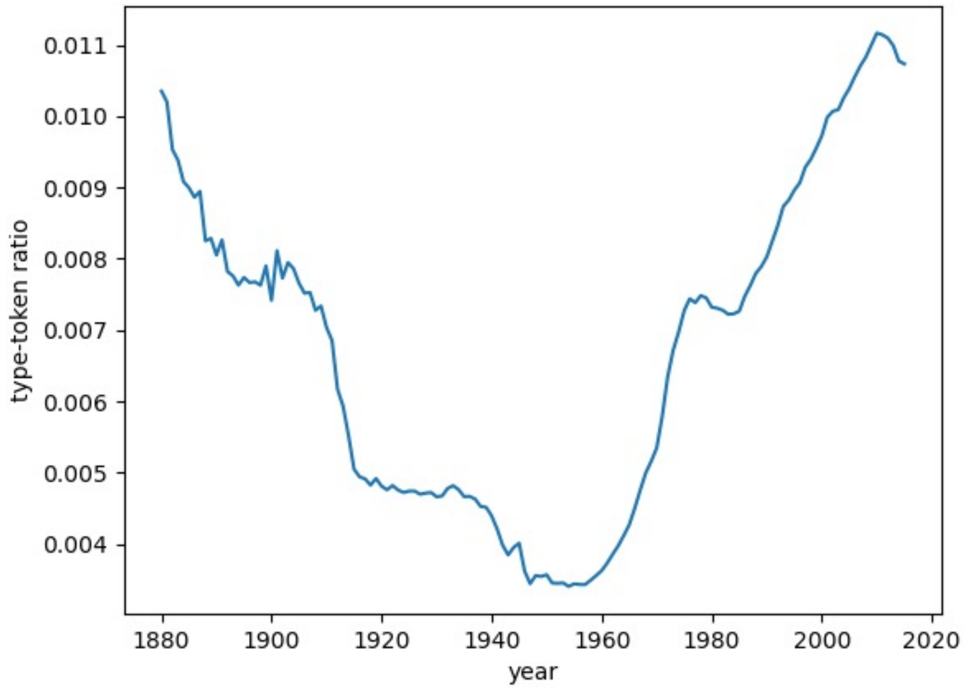
```
>>> def type_token_ratio(frequencies):    # Compute the type-token ratio of the frequencies
...     return len(frequencies) / frequencies.sum()

>>> ax = df.loc[df['sex'] == 'F'].groupby(level=0)['frequency'].apply(type_token_ratio).plot()

>>> ax.set_ylabel("type-token ratio");
  Text(0, 0.5, 'type-token ratio')
```

What is the TTR ratio?

TTR = Number of distinct species / Total number of species

Large TTR, large variability

Small TTR, we see always the same species.

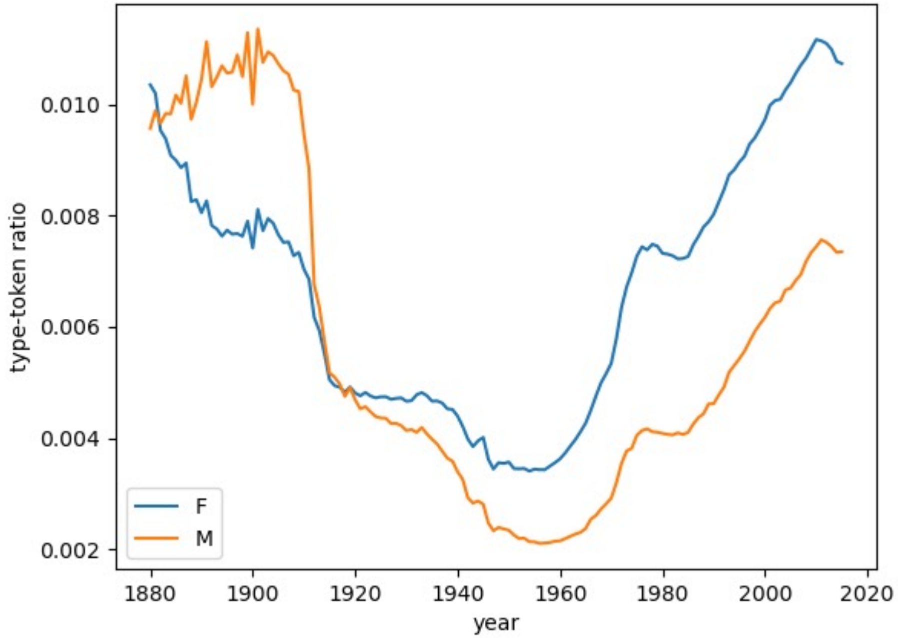


Data Visualization

```
>>> for sex in ['F', 'M']:
...     counts = df.loc[df['sex'] == sex, 'frequency']
...     tt_ratios = counts.groupby(level=0).apply(type_token_ratio)
```

Use the same axis to plot both sexes (i.e. ax=ax)

```
...     tt_ratios.plot(label=sex, legend=True, ax=ax)

>>> ax.set_ylabel("type-token ratio");
    Text(0, 0.5, 'type-token ratio')
```



Data Visualization

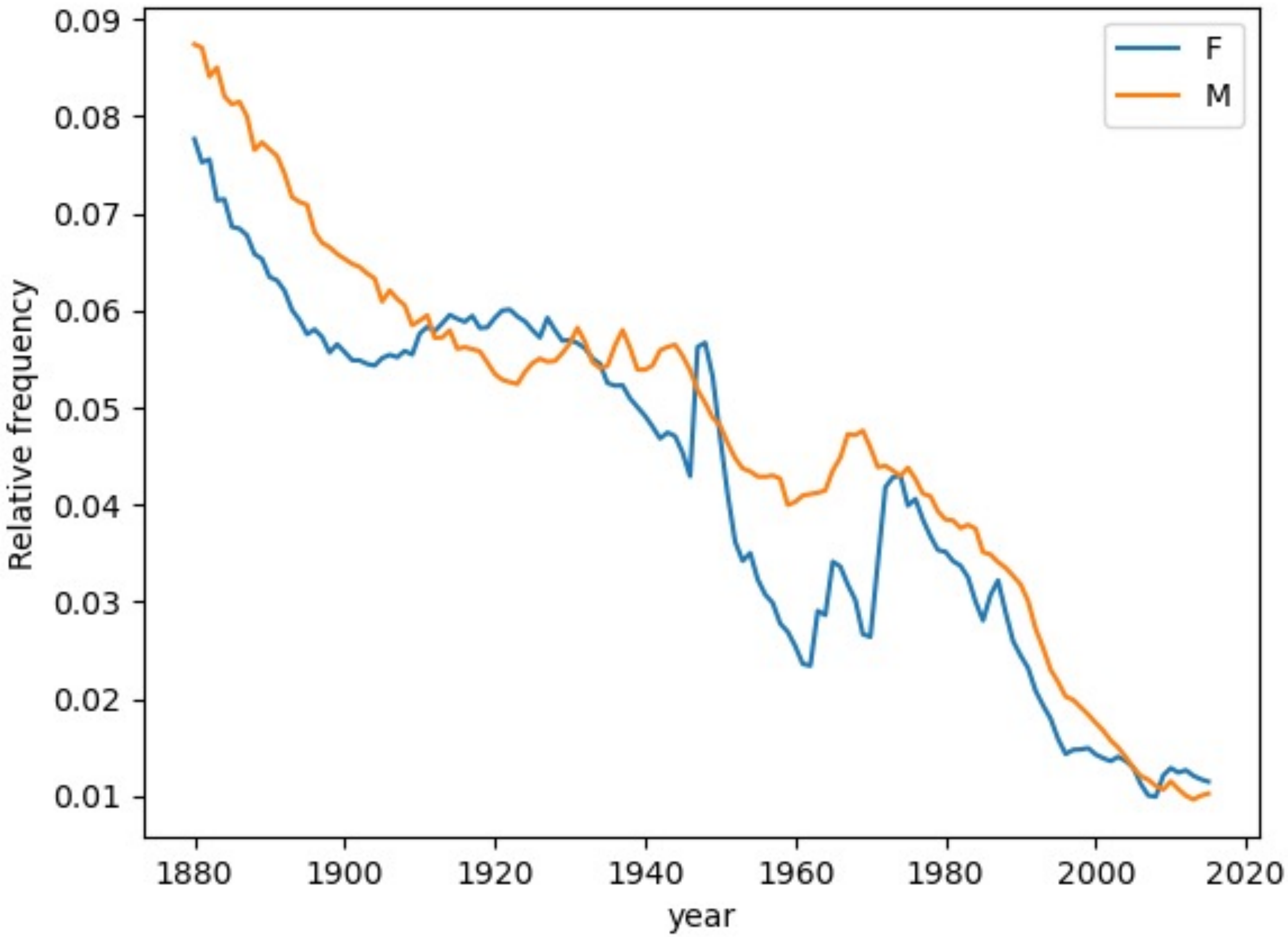
Similar plot but with the relative frequencies

```
>>> def max_relative_frequency(frequencies):
...     return (frequencies / frequencies.sum()).max()

>>> fig, ax = plt.subplots()
>>> for sex in ['F', 'M']:
...     counts = df.loc[df['sex'] == sex, 'frequency']
...     div = counts.groupby(level=0).apply(max_relative_frequency)
...     div.plot(label=sex, legend=True, ax=ax)
>>> ax.set_ylabel("Relative frequency")

>>> plt.show()
```

Data Visualiz

The resulting graph with the relative frequency of the most popular name.

The most popular name is less and less "popular" (from around 8% in 1880 to 1% in 2015).



Data Visualization

Analysing the string corresponding to names but distinctly for both genders.

Can we detect some patterns?

Inspect the string corresponding to the name.

Do we see more names starting with a 'M' than a 'J'?

Can we count the number of names ending with '-n'?

Do we observe a difference between the two genders?

Data Visualization

```
>>> boys_names = df.loc[df['sex'] == 'M', 'name']
```

```
>>> boys_names.head()
```
Splitting according to the gender

```
year
1880        John
1880     William
1880       James
1880     Charles
1880      George
Name: name, dtype: object
```

```
>>> boys_names.str.lower().head()
```
names in lowercase

```
year
1880        john
1880     william
1880       james
1880     charles
1880      george
Name: name, dtype: object
```

Data Visualization

```
>>> boys_names.loc[boys_names.str.match('[aeiou]', case=False)]
year   # RE  the name must start with a vowel, ignore upper/lowercase distinction
1880         Edward
1880         Arthur
1880         Albert
1880         Andrew
1880         Ernest
…
>>> boys_names.loc[boys_names.str.match('[^aeiou]', case=False)]
year   # RE  the name must NOT start with a vowel, ignore upper/lowercase distinction
1880         John
1880         William
1880         James
1880         Charles
1880         George
…
```

Data Visualization

```
>>> boys_names.loc[boys_names.str.match(
                  '[^aeiou]+o[^aeiou]', case=False)].head()
year                                         # Not beginning with a vowel
1880        John                             # Names with a 'o' as the first vowel
1880      Joseph                             # after the first 'o', not a vowel (e.g. Joel)
1880      Thomas
1880      Robert
1880         Roy
Name: name, dtype: object
```

```
>>> boys_names.str.get(0).head()      # the first letter of the name for each year
year
1880    J
1880    W
1880    J
1880    C
1880    G
Name: name, dtype: object
```

Data Visualization

```
>>> boys_coda = boys_names.str.get(-1)
>>> boys_coda.head()          # the last letter of the name for each year
year                          # This is a Series object
1880    n
1880    m
1880    s
1880    s
1880    e
Name: name, dtype: object
```

Return a Series containing counts of unique values.

```
>>> boys_fd = boys_coda.groupby('year').value_counts(normalize=True)
>>> boys_fd.head()
year  name
1880  n         0.181646     # the relative frequency of the last letter of the name for each year
      e         0.156102
      s         0.098392
      y         0.095553
      d         0.080416
Name: name, dtype: float64
```

Data Visualization

```
>>> boys_fd.index

MultiIndex([(1880, 'n'),
            (1880, 'e'),
            (1880, 's'),
            (1880, 'y'),
            (1880, 'd'),
            (1880, 'l'),
            (1880, 'r'),
            (1880, 't'),
            (1880, 'h'),
            (1880, 'o'),
            ...
            (2015, 'x'),
            (2015, 'b'),
```

Our Series has two values in the index

…

Data Visualization

Sort object by labels

```
>>> boys_fd.loc[1940].sort_index().head()
name
a     0.029232                  # 'a' is the first position because the sort is based on a MultiIndex
b     0.002288
c     0.004575
d     0.074479
e     0.164718                  # 'e' is more frequent at the last letter than the 'a', 'b', 'c', or 'd'
Name: name, dtype: float64  # but this is not our goal

>>> boys_fd.loc[[(1960, 'n'), (1960, 'p'), (1960, 'r')]]
year  name
1960  n        0.190891     # but over all three index values, 'n' is more frequent
      p        0.003705
      r        0.046851
Name: name, dtype: float64
```

Data Visualization

```
>>> boys_fd = boys_fd.unstack()
```

to need to have the year as index

```
>>> boys_fd.head()
   name       a          b          c          d          e          f          g   ... y          z
year                                                                                  ...

1880   0.029328   0.006623   0.006623   0.080416   0.156102   0.006623   0.007569   ...   0.095553   0.002838
1881   0.027108   0.006024   0.008032   0.076305   0.148594   0.005020   0.012048   ...   0.095382   0.001004
1882   0.025501   0.006375   0.006375   0.080146   0.166667   0.007286   0.008197   ...   0.100182   0.002732
1883   0.028183   0.004859   0.006803   0.082604   0.158406   0.006803   0.008746   ...   0.094266   0.000972
1884   0.028444   0.008889   0.006222   0.080000   0.155556   0.005333   0.007111   ...   0.100444   0.002667

[5 rows x 26 columns]
```

Be careful: With 'j' with have a lot of `NaN`

Data Visualization

```
>>> boys_fd = boys_fd.fillna(0)    # Replace NaN by 0
```

```
>>> fig, axes = plt.subplots(
...     nrows=2, ncols=4, sharey=True, figsize=(12, 6))
```

```
>>> letters = ["a", "d", "e", "i", "n", "o", "s", "t"]
```

```
>>> axes = boys_fd[letters].plot(
...     subplots=True, ax=axes, title=letters, color='C0', grid=True, legend=False)
```

```
>>> for ax in axes.flatten():
...     ax.xaxis.label.set_visible(False)
```

The x-axis of each subplots is labelled with 'year'.

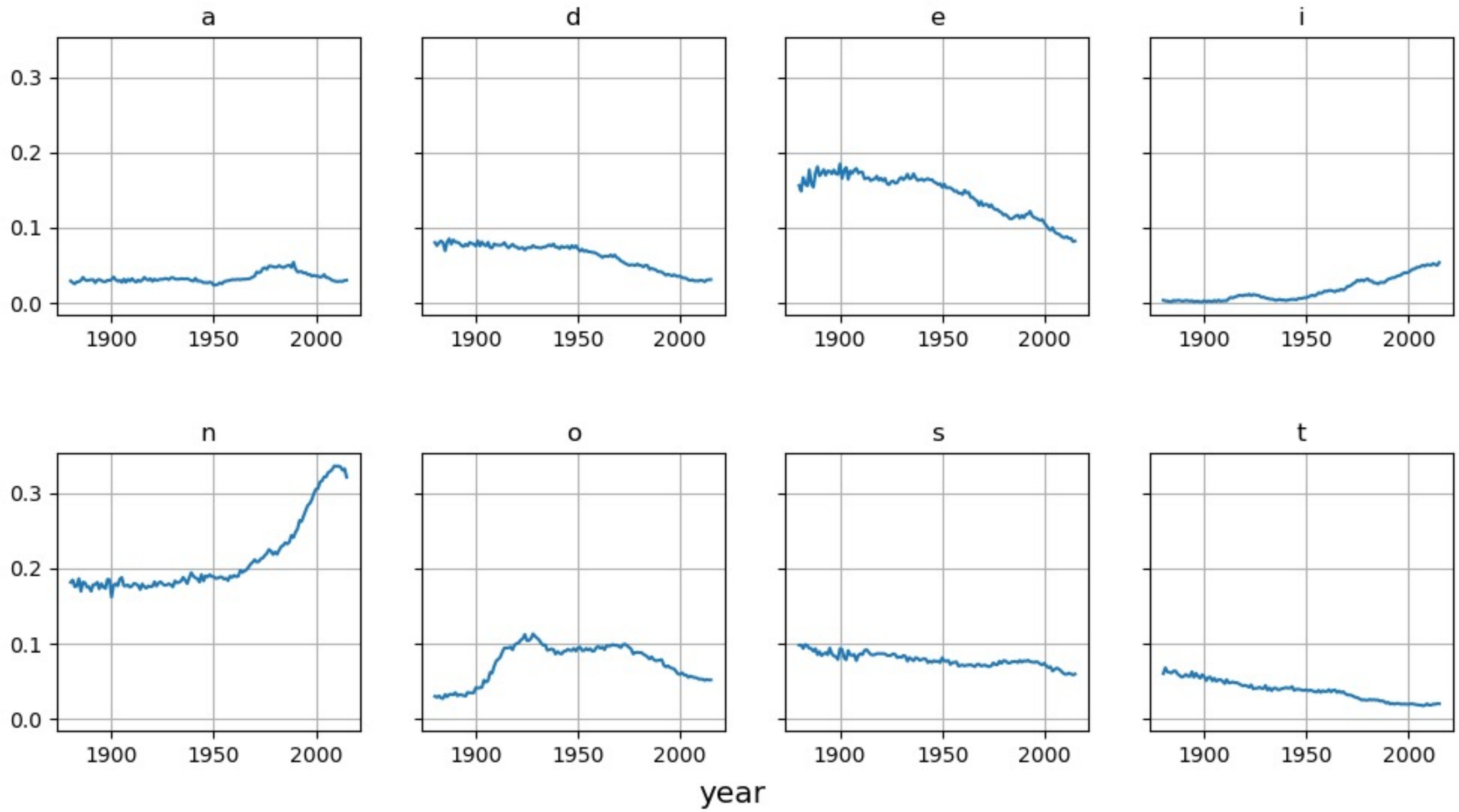
```
>>> fig.text(0.5, 0.04, "year", ha="center", va="center", fontsize="x-large")
   Text(0.5, 0.04, 'year')
```

Reserve some additional height for space between subplots

```
>>> fig.subplots_adjust(hspace=0.5)
>>> plt.show()
```

Data Visualization

The resulting graph with the last letter of boy names



Data Visualization

Can we identify unisex names?

If Mary is clearly given to a female person (or James to a boy), some name can be attribute to a girl or a boy.

As a unisex name, if we found a distribution 50-50 between the two genders, it is clearly better than a distribution 10-90. (But a 50-50 could be a very strict constraint).

Data Visualization

```
>>> d = df.loc[1910]
```
start with a subset

```
>>> d.duplicated(subset='name')
year
1910     False
1910     False
1910     False
1910     False
1910     False
…
```
in year 1910, do we have multiple lines with the same name

```
>>> sum(d.duplicated(subset='name'))
455

>>> len(d.duplicated(subset='name'))
4628
```
Usually not, but around 10% are duplicate

Data Visualization

```
>>> duplicates = d[d.duplicated(subset='name')]['name']
>>> duplicates.head()
year
1910       John
1910      James
1910    William
1910     Robert
1910     George
Name: name, dtype: object
```

Data Visualization

```
>>> d = d.loc[d.duplicated(subset='name', keep=False)]
>>> d.sort_values('name').head()  # keep=False because usually the first occurence is removed
       name sex  frequency
year
1910  Abbie   M          8    # Abbie is more frequent for girls than boys
1910  Abbie   F         79
1910  Addie   M          8
1910  Addie   F        495
1910  Adell   M          6

>>> d = d.pivot_table(values='frequency', index='name', columns='sex')
>>> d.head()    # d is still a DataFrame with index = names
sex        F    M
name
Abbie     79    8
Addie    495    8
Adell     86    6
Afton     14    6
Agnes   2163   13
```

Data Visualization

```
>>> (d['F'] / (d['F'] + d['M'])).head()
name
Abbie     0.908046      # The frequency for girls is high for this name
Addie     0.984095
Adell     0.934783
Afton     0.700000
Agnes     0.994026
dtype: float64
```

```
>>> def usage_ratio(df):     # Compute the usage ratio for unisex names
...     df = df.loc[df.duplicated(subset='name', keep=False)]
...     df = df.pivot_table(values='frequency', index='name', columns='sex')
...     return df['F'] / (df['F'] + df['M'])
```

Data Visualization

```
>>> d = df.groupby(level=0).apply(usage_ratio)
>>> d.head()
year  name
1880  Addie     0.971631   # the frequencies in girl names
      Allie     0.772059
      Alma      0.951890
      Alpha     0.812500
      Alva      0.195402
dtype: float64
```

Data Visualization

```
>>> d = d.unstack(level='name')   # reuse the year as index and names as column names
>>> d.tail()

name  Aaden      Aadi  Aadyn   Aalijah   Aaliyah    Aamari     Aamir  ...     Zyon  Zyree

year
2011    NaN       NaN    NaN  0.318182       NaN  0.428571       NaN  ...  0.155080    NaN
2012    NaN  0.081967    NaN  0.333333  0.998729       NaN       NaN  ...  0.257426    NaN
2013    NaN  0.078947    NaN  0.500000  0.997705  0.333333  0.042553  ...  0.147239    0.5
2014    NaN       NaN    NaN  0.363636  0.998975  0.400000       NaN  ...  0.209790    NaN
2015    NaN       NaN    NaN  0.413793  0.998967  0.562500       NaN  ...  0.164773    NaN

[5 rows x 9025 columns]

>>> unisex_ranking = abs(d - 0.5).fillna(0.5).mean().sort_values().index
```

Have a look step by step for this last computation

Data Visualization

```
>>> dd = abs(d - 0.5).fillna(0.5)
>>> dd
name  Aaden      Aadi  Aadyn   Aalijah   Aaliyah    Aamari     ...
year
 1880    0.5  0.500000    0.5  0.500000  0.500000  ...
 1881    0.5  0.500000    0.5  0.500000  0.500000  0.500000  ...
 1882    0.5  0.500000    0.5  0.500000  0.500000  0.500000  ...
 1883    0.5  0.500000    0.5  0.500000  0.500000  0.500000  ...
 1884    0.5  0.500000    0.5  0.500000  0.500000  0.500000  ...
 ...
[136 rows x 9025 columns]
```

Data Visualization

```
>>> dd.mean().sort_values()    # Ascending sort
name
Alva        0.121509
Jessie      0.149922
Marion      0.154971
Tommie      0.169846
Lorenza     0.172708
              ...
Alayna      0.499975      # At the end of the sort with have name with a 50-50 distribution
Aniyah      0.499975
Ezekiel     0.499976
Keira       0.499977
Ximena      0.499981
Length: 9025, dtype: float64
```

Data Visualization

```
>>> fig, axes = plt.subplots(
...     nrows=2, ncols=4, sharey=True, sharex=True, figsize=(12, 6))

>>> names = unisex_ranking[:8].tolist()     # Not all names, and form a list (of names)

>>> d[names].plot(
...     subplots=True, color="C0", ax=axes, legend=False, title=names)
array([<AxesSubplot:title={'center':'Alva'}, xlabel='year'>,
       <AxesSubplot:title={'center':'Jessie'}, xlabel='year'>,
       <AxesSubplot:title={'center':'Marion'}, xlabel='year'>,
       <AxesSubplot:title={'center':'Tommie'}, xlabel='year'>,
       <AxesSubplot:title={'center':'Lorenza'}, xlabel='year'>,
       <AxesSubplot:title={'center':'Golden'}, xlabel='year'>,
       <AxesSubplot:title={'center':'Ivory'}, xlabel='year'>,
       <AxesSubplot:title={'center':'Jonnie'}, xlabel='year'>],
      dtype=object)
```

Data Visualization

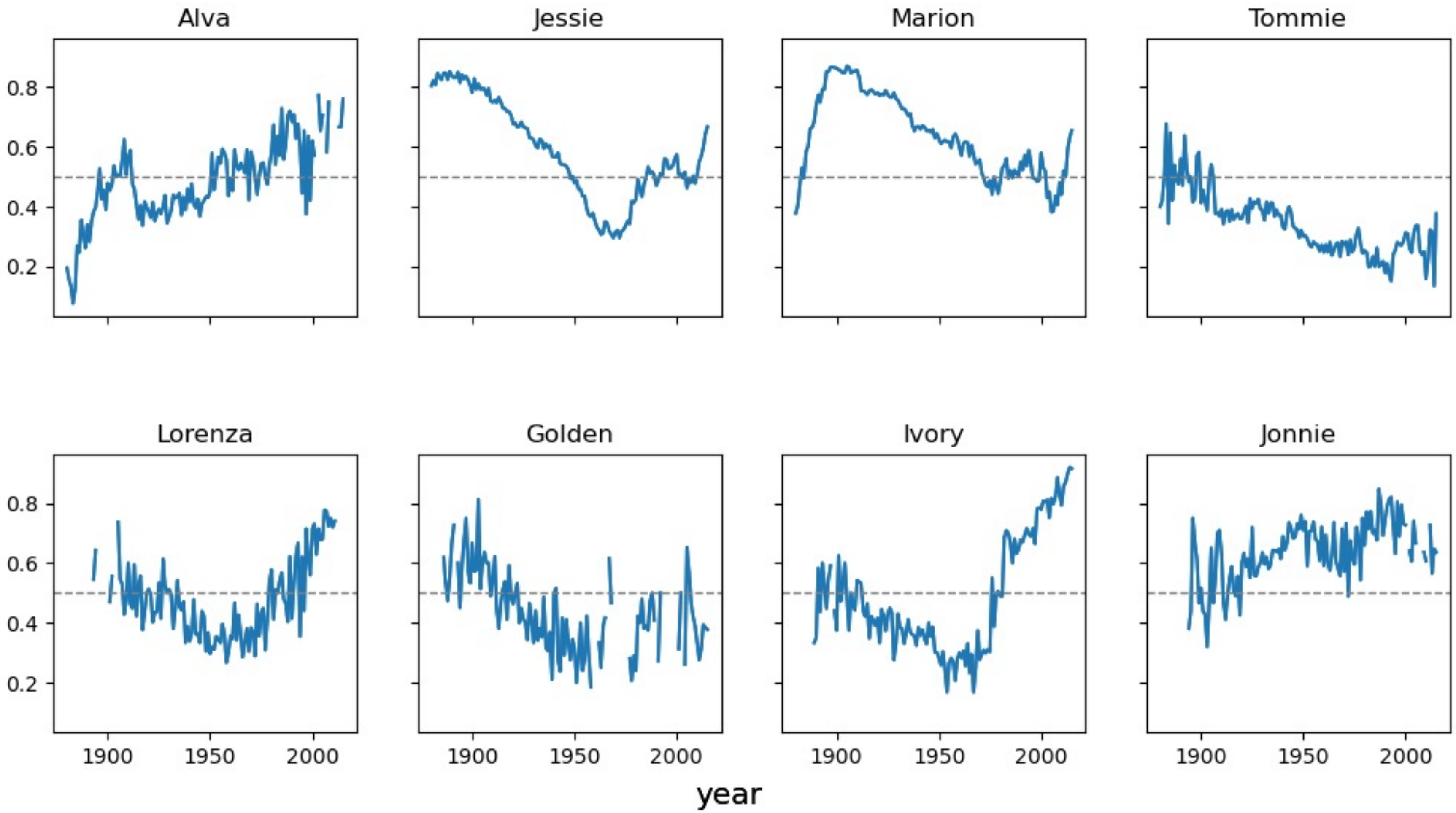
```
>>> for ax in axes.flatten():
...     ax.xaxis.label.set_visible(False)
...     ax.axhline(0.5, ls='--', color="grey", lw=1)

<matplotlib.lines.Line2D object at 0x7ff059589a30>
<matplotlib.lines.Line2D object at 0x7ff059597d90>
<matplotlib.lines.Line2D object at 0x7ff0595ed310>
<matplotlib.lines.Line2D object at 0x7ff0595ed8b0>
<matplotlib.lines.Line2D object at 0x7ff0595ede50>
<matplotlib.lines.Line2D object at 0x7ff0595f5430>
<matplotlib.lines.Line2D object at 0x7ff0595f59d0>
<matplotlib.lines.Line2D object at 0x7ff0595f5f70>

>>> fig.text(0.5, 0.04, "year", ha="center", va="center", fontsize="x-large");
Text(0.5, 0.04, 'year')

>>> fig.subplots_adjust(hspace=0.5);
```

Data Visualization

The graph of the most unisex names in our dataset

When larger than 0.5, it is more associated with girls.

This change over the years…



Data Visualization

```
>>> fig, axes = plt.subplots(
...     nrows=2, ncols=4, sharey=True, sharex=True, figsize=(12, 6))

>>> d[names].rolling(window=10).mean().plot(
...     color='C0', subplots=True, ax=axes, legend=False, title=names);

>>> for ax in axes.flatten():
...     ax.xaxis.label.set_visible(False)
...     ax.axhline(0.5, ls='--', color="grey", lw=1)

>>> fig.text(0.5, 0.04, "year", ha="center", va="center", fontsize="x-large");
    Text(0.5, 0.04, 'year')

>>> fig.subplots_adjust(hspace=0.5);

>>> plt.show()
```

Data Visualization

The same graph but with a smoothed line.