Bioinformatical problem solving with Python

Wednesdays 17:30-19:00, M801

alexander.nater@uni-konstanz.de

Not covered topics so far

- Error handling (exceptions)
- **Numerical and scientific libraries (SciPy, NumPy, pandas, etc.)**
- Plotting libraries (Matplotlib)
- Software design

- Data structures with labeled axes supporting automatic data alignment
- Integrated time series functionality
- Arithmetic operations pass on the axis labels
- Flexible handling of missing data
- pandas data structures: Series and DataFrame

```
from pandas import Series, DataFrame
import pandas as pd
```

- One-dimensional array-like container with associated array of data labels (index).

- Default index (0 … N-1):
  obj = Series([4, 7, -5, 3])
  obj.values → array([4, 7, -5, 3])
  obj.index → RangeIndex(start=0, stop=4, step=1)

- User-defined index:
  obj2 = Series([4, 7, -5, 3], index=[‘d’, ‘b’, ‘a’, ‘c’])
  obj2.values → array([4, 7, -5, 3])
  obj2.index → Index([d, b, a, c], dtype=object)

- Series can be constructed directly from a Python dictionary:

  sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000}
  obj3 = Series(sdata)
  obj3.values → array([35000, 71000, 16000])
  obj3.index → Index([Ohio, Texas, Oregon], dtype=object)

- Values can be accessed by their indices:

  obj3['Ohio'] → 35000
  obj3[['Texas', 'Oregon']]
  obj3['Oregon'] = 25000
  obj3['Utah'] = 5000

Given the two Series objects

```
obj1 = Series([35000, 71000, 16000, 5000],
              index=['Ohio', 'Texas', 'Oregon', 'Utah'])
obj2 = Series([6000, 12000, 9000, 17000],
              index=['Oregon', 'Utah', 'Ohio', 'California'])
```

Create a new Series object with the added values for each location. What happens with the locations that are only present in one of the objects?

- We can bring different Series object in line by calling the ‘reindex’ method with the union of the two indices:

```
new_index =  obj1.index.union(obj2.index)
obj3 = obj1.reindex(new_index, fill_value=0)
obj4 = obj2.reindex(new_index, fill_value=0)
obj3 + obj4
```

- Alternatively, we can use the ‘add’ method of Series objects with the ‘fill_value’ argument:

```
obj1.add(obj2, fill_value=0)
```

- There are other useful methods to summarize Series objects:
  obj1.sum() → 127000
  obj1.min() → 5000
  obj1.max() → 71000
  obj1.mean() → 31750.0
  obj1.std() → 28952.54738

DataFrame

- Two-dimensional data structure containing an ordered collection of columns (Series).
- Columns can be of different type.
- A DataFrame has both a row and a column index.

- DataFrame objects can be constructed from a dict of equal-length lists:

```
data = {‘state’: [‘Ohio’, ‘Ohio’, ‘Ohio’, ‘Nevada’, ‘Nevada’],
        ‘year’: [2000, 2001, 2002, 2001, 2002],
        ‘pop’: [1.5, 1.7, 3.6, 2.4, 2.9]}
df = DataFrame(data)
df.values
df.index
df.columns
```

- Specifying a list of column names will put the columns in the desired order:

```
df2 = DataFrame(data, columns=[‘year’, ‘state’, ‘pop’],
                      index=[‘one’, ‘two’, ‘three’, ‘four’, ‘five’])
```

- Columns can be accessed by dict-like notation or attribute:

  df[‘state’]
  df.year

- Rows can be accessed with the ‘ix’ indexing field:

  df.ix[1]
  df2.ix[‘three’]

  or by slicing/boolean indexing the object:

  df[:2]
  df2[data[‘year’] > 2001]

- Subset of rows and columns with the ‘ix’ field:

  df2.ix[‘three’, [‘year’, ‘state’]]

- Columns can be modified by assigning scalars, lists/arrays, or Series objects:

```
df['size'] = 6000
df['debt'] = [5.2, 6.7, 8.3, 9.0, 1.2]
df2['growth'] = Series([-1.2, 3.0, 2.7], index=['two', 'four', 'five'])
```

- The 'del' keyword removes columns:

```
del df2['growth']
```

- The 'T' attribute can be used to transpose DataFrame objects.

- The 'sort_index' method can be used to sort DataFrame objects.

Exercise

Using the two Series objects from the first exercise, create a DataFrame using appropriate column labels. Create a third column containing the mean of the first two columns. Create a new DataFrame containing only the rows with means smaller than 20000.