DS-GA 1007
Programming for Data Science

Lecture 10

pandas I – Operations on Tables

Package that combines array operations and queries on tabular data

DS-GA 1007
Programming for Data Science

Lecture 10
pandas I – Operations on Tables

Announcements

- Homework 7 due **Sunday November 10** at 11:59pm
- Survey 3 due **Sunday November 10** at 11:59pm
- Project
  - Milestone due **Thursday November 28** at 11:59pm
  - Background
  - Plans
    - Some Components of the Software
    - Some Relevant Datasets and Approaches

Announcements

How was lab on Monday?!

- Homework 7 due **Sunday November 10** at 11:59pm
- Survey 3 due **Sunday November 10** at 11:59pm
- Project
  - Milestone due **Thursday November 28** at 11:59pm
  - Background
  - Plans
    - Some Components of the Software
    - Some Relevant Datasets and Approaches

Agenda

- Review
  - numpy
  - matplotlib
- Lesson
  - notebooks and scripts
  - pandas
- Readings
  - Python for Data Analysis by Wes McKinney
  - http://pandas.pydata.org/pandas-docs/stable/index.html

Objectives

- How can we perform operations on arrays?
- How can we choose appropriate charts for data types?
- Is it possible to link notebook and scripts?
- How can we arrange rows and columns of a table?

Agenda

- Review
  - numpy
  - matplotlib
- Lesson
  - notebooks and scripts
  - pandas
- Readings
  - Python for Data Analysis by Wes McKinney
  - http://pandas.pydata.org/pandas-docs/stable/index.html

Objectives

- Dimensions, Shape and Data Type
- Retrieving Data
  - Access, Iterate
  - Slice, Mask
- Broadcasting
- Views vs Copies

Agenda

- Review
  - numpy
  - matplotlib
- Lesson
  - notebooks and scripts
  - pandas
- Readings
  - Python for Data Analysis by Wes McKinney
  - http://pandas.pydata.org/pandas-docs/stable/index.html

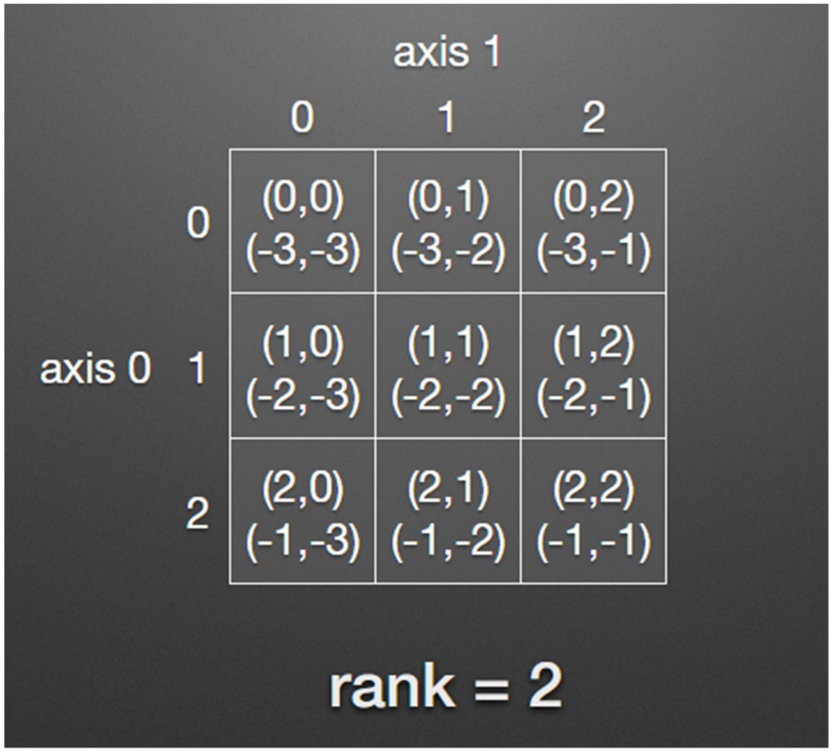
Objectives

- Dimensions, Shape and Data Type
- Retrieving Data
  - Access, Iterate
  - Slice, Mask
- Broadcasting
- Views vs Copies

```
import numpy
import numpy as np
from numpy import *
```

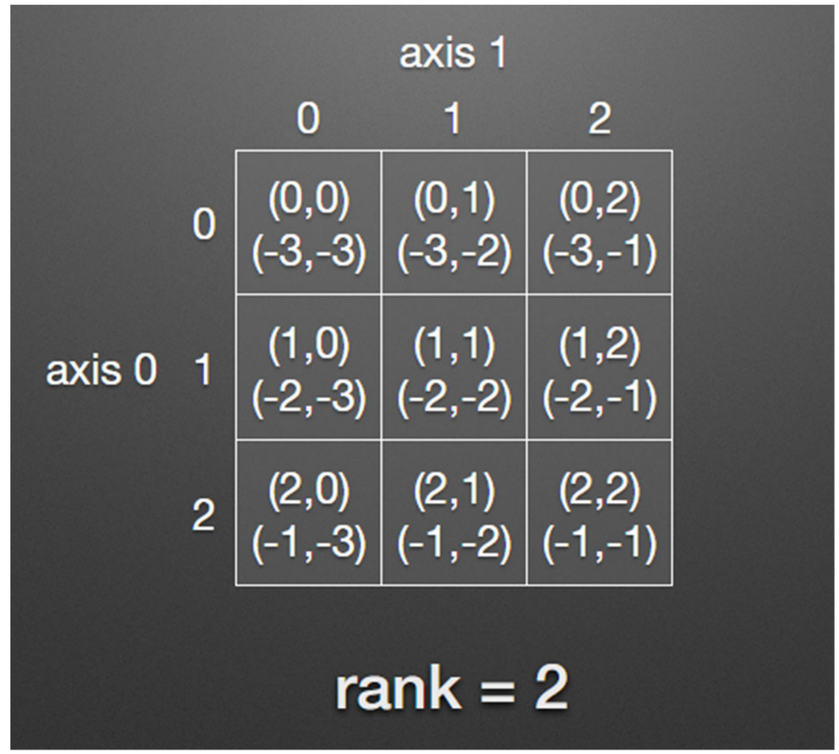
Dimension, Shape and Data Type

- Dimension
  - Number of nested arrays
  - Each array dimension called ***axis***
  - Axes start at 0
  - Number of Axes is ***rank***
  - ***ndarray.ndim***



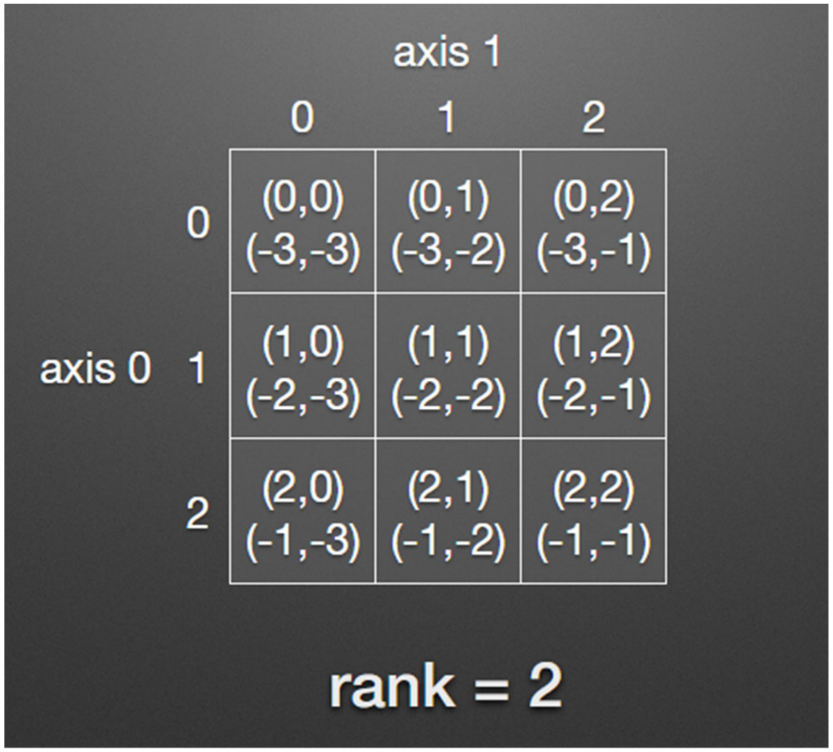
Dimension, Shape and Data Type

- Shape
  - Tuple of integers indicating size of each axis
  - ***Size*** of array is total number of elements
  - ***ndarray.shape***
  - ***ndarray.size***



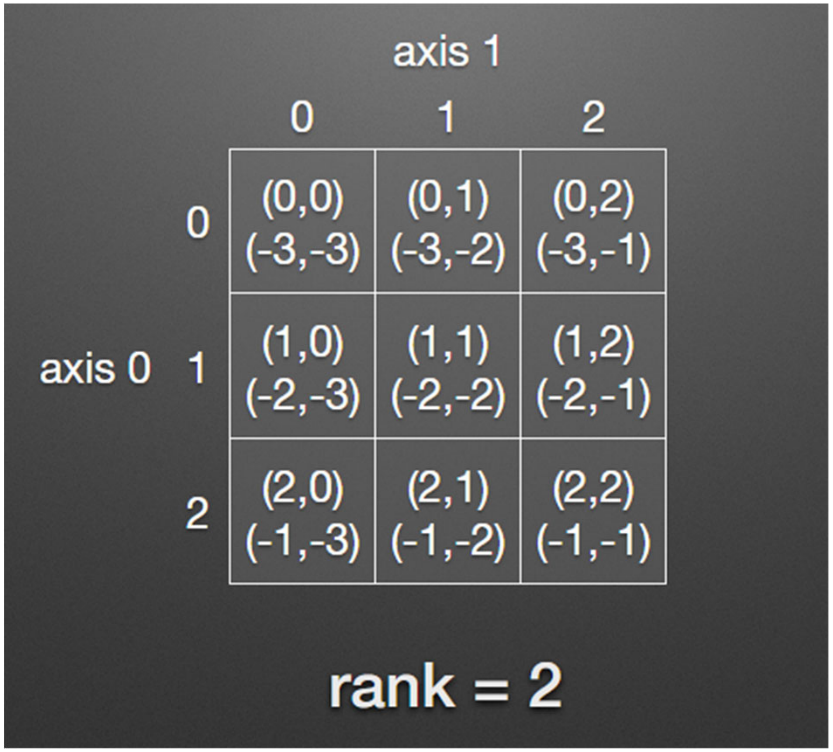
Dimension, Shape and Data Type

- Data Type
  - The package will try to detect the data type from the entries
  - Many data types and precision supported
    - float32
    - int8
    - complex64
  - ***ndarray.dtype***



Retrieving Data

- Access
  - Arrays indexed by tuple of integers
  - Use brackets to get entries
  - Negative indices are supported
  - ***x[i,j] or x[(i,j)]***



Retrieving Data

- Iterate
  - In loops iteration performed over first axes
  - Can convert from two dimensional to one dimensional in row major or column major order

```
a2d = np.array([[1,3,5,7,9,11],
                [2,4,6,8,10,12],
                [0,1,2,3,4,5]])
```

```
for r in a2d:
    print(r)
```

```
[ 1  3  5  7  9 11]
[ 2  4  6  8 10 12]
[0 1 2 3 4 5]
```

```
for i in a2d.flat:
    print(i)
```

```
1
3
5
7
9
```

Retrieving Data

- Slicing
  - Specified by start, end and stride
  - Omitting an index means from the beginning, to the end or default stride
  - A slice is a ***view*** of the original array. Views are like references

```
a2d
```

```
array([[ 1,  3,  5,  7,  9, 11],
       [ 2,  4,  6,  8, 10, 12],
       [ 0,  1,  2,  3,  4,  5]])
```

```
print('fixing a column and traversing rows\n',a2d[:,2])

print('traversing an array block\n',a2d[1:,2:5])

print('traversing a subset of rows \n',a2d[[0,2]])

print('traversing a subset of columns \n',a2d[:,[0,2,5]])

print('traversing a subset of array elements \n',a2d[[0,1,2],[0,2,5]])
```

Retrieving Data

- Slicing
  - Specified by start, end and stride
  - Omitting an index means from the beginning, to the end or default stride
  - A slice is a ***view*** of the original array. Views are like references
- Fancy Indexing
  - Passing multiple pairs of indices

```
a2d
```

```
array([[ 1,  3,  5,  7,  9, 11],
       [ 2,  4,  6,  8, 10, 12],
       [ 0,  1,  2,  3,  4,  5]])
```

```
print('fixing a column and traversing rows\n',a2d[:,2])

print('traversing an array block\n',a2d[1:,2:5])

print('traversing a subset of rows \n',a2d[[0,2]])

print('traversing a subset of columns \n',a2d[:,[0,2,5]])

print('traversing a subset of array elements \n',a2d[[0,1,2],[0,2,5]])
```

Retrieving Data

- Slicing
  - Specified by start, end and stride
  - Omitting an index means from the beginning, to the end or default stride
  - A slice is a ***view*** of the original array. Views are like references
- Fancy Indexing
  - Passing multiple pairs of indices

```
a2d
```

```
array([[ 1,  3,  5,  7,  9, 11],
       [ 2,  4,  6,  8, 10, 12],
       [ 0,  1,  2,  3,  4,  5]])
```

```
print('fixing a column and traversing rows\n',a2d[:,2])

print('traversing an array block\n',a2d[1:,2:5])

print('traversing a subset of rows \n',a2d[[0,2]])

print('traversing a subset of columns \n',a2d[:,[0,2,5]])

print('traversing a subset of array elements \n',a2d[[0,1,2],[0,2,5]])
```

```
traversing an array block
 [[ 6  8 10]
 [ 2  3  4]]
traversing a subset of rows
 [[ 1  3  5  7  9 11]
 [ 0  1  2  3  4  5]]
traversing a subset of columns
 [[ 1  5 11]
 [ 2  6 12]
 [ 0  2  5]]
traversing a subset of array elements
 [1 6 5]
```

Retrieving Data

- Masking
  - Array of True and False useful to select elements in another array
  - Can make assignments using True and False

```
x = np.arange(18).reshape(3,6)
print(x)
mask = (x > 7)
print(mask)
print(x[mask])
x[mask]=0
print(x)
```

```
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]]
[[False False False False False False]
 [False False  True  True  True  True]
 [ True  True  True  True  True  True]]
[ 8  9 10 11 12 13 14 15 16 17]
[[0 1 2 3 4 5]
 [6 7 0 0 0 0]
 [0 0 0 0 0 0]]
```

Views vs Copies

- Slicing an array gives a view
  - Like a reference to part to an array
  - Changing elements of the view will change the original array
- Accessing an array through indexing gives a copy
- Use ***copy.deepcopy*** to avoid side-effects

```
x = np.arange(18).reshape(3,6)
print(x)
mask = (x > 7)
print(mask)
print(x[mask])
x[mask]=0
print(x)
```

```
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]]
[[False False False False False False]
 [False False  True  True  True  True]
 [ True  True  True  True  True  True]]
[ 8  9 10 11 12 13 14 15 16 17]
[[0 1 2 3 4 5]
 [6 7 0 0 0 0]
 [0 0 0 0 0 0]]
```

Broadcasting

- Allows for element by element operations on arrays with different sizes
  - Package transforms arrays so that they all have the same size

```
A = np.arange(25).reshape(5,5)
s = 5
B = s + A
B
```

```
array([[ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24],
       [25, 26, 27, 28, 29]])
```

$$B = s + A = \begin{bmatrix} 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 & 9 \\ 10 & 11 & 12 & 13 & 14 \\ 15 & 16 & 17 & 18 & 19 \\ 20 & 21 & 22 & 23 & 24 \end{bmatrix}$$

Broadcasting

- The two arrays must be compatible for broadcasting
  - the axis lengths match, or
  - either of the lengths is 1
- Broadcasting is then performed over the missing and/or length 1 dimensions

```
v = np.arange(5)
print(v.shape)

w = np.arange(3).reshape(3,1)
print(w.shape)

Z =  v + w

print(v,'\n')
print(w,'\n')
print(Z)
```

```
(5,)
(3, 1)
[0 1 2 3 4]

[[0]
 [1]
 [2]]

[[0 1 2 3 4]
 [1 2 3 4 5]
 [2 3 4 5 6]]
```

Broadcasting

- The two arrays must be compatible for broadcasting
  - the axis lengths match, or
  - either of the lengths is 1
- Broadcasting is then performed over the missing and/or length 1 dimensions

$$Z = v + w$$

$$\downarrow$$

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$$

$$\downarrow$$

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ 0 & 1 & 2 & 3 & 4 \\ 0 & 1 & 2 & 3 & 4 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

Vectorized Operations

- Repeated operations in loops are slow
- Package supports C extensions that allow for execution across arrays element by element
- Processing concurrently instead of sequentially saves time

```python
import numpy as np
np.random.seed(0)

def compute_reciprocals(values):
    output = np.empty(len(values))
    for i in range(len(values)):
        output[i] = 1.0 / values[i]
    return output

values = np.random.randint(1, 10, size=5)
compute_reciprocals(values)
```

Vectorized Operations

- Repeated operations in loops are slow
- Package supports C extensions that allow for execution across arrays element by element
- Processing concurrently instead of sequentially saves time

```
import numpy as np
np.random.seed(0)

def compute_reciprocals(values):
    output = np.empty(len(values))
    for i in range(len(values)):
        output[i] = 1.0 / values[i]
    return output

values = np.random.randint(1, 10, size=5)
compute_reciprocals(values)
```

```
big_array = np.random.randint(1, 100, size=1000000)
%timeit compute_reciprocals(big_array)
```

```
1 loop, best of 3: 2.91 s per loop
```

Vectorized Operations

- Repeated operations in loops are slow
- Package supports C extensions that allow for execution across arrays element by element
- Processing concurrently instead of sequentially saves time

```python
print(compute_reciprocals(values))
print(1.0 / values)
```

```
[ 0.16666667  1.          0.25        0.25        0.125     ]
[ 0.16666667  1.          0.25        0.25        0.125     ]
```

Vectorized Operations

- Repeated operations in loops are slow
- Package supports C extensions that allow for execution across arrays element by element
- Processing concurrently instead of sequentially saves time

```
print(compute_reciprocals(values))
print(1.0 / values)
```

```
[ 0.16666667  1.          0.25        0.25        0.125       ]
[ 0.16666667  1.          0.25        0.25        0.125       ]
```

```
%timeit (1.0 / big_array)
```

```
100 loops, best of 3: 4.6 ms per loop
```

Vectorized Operations

- Possible to create your own operations through the ***numba*** package
  - Write a function meant for numbers
  - Decorate the function indicating data types
  - Call the function on arrays

```python
from numba import vectorize, float64


@vectorize([float64(float64, float64)])
def f(x, y):
    return x + y
```

Vectorized Operations

- Possible to create your own operations through the ***numba*** package
  - Write a function meant for numbers
  - Decorate the function indicating data types
  - Call the function on arrays

```
from numba import vectorize, float64

@vectorize([float64(float64, float64)])
def f(x, y):
    return x + y
```

```
>>> a = np.arange(6)
>>> f(a, a)
array([ 0,  2,  4,  6,  8, 10])
```

Vectorized Operations

- Possible to create your own operations through the ***numba*** package
  - Write a function meant for numbers
  - Decorate the function indicating data types
  - Call the function on arrays

```
from numba import vectorize, float64

@vectorize([float64(float64, float64)])
def f(x, y):
    return x + y
```

```
>>> a = np.arange(6)
>>> f(a, a)
array([ 0,  2,  4,  6,  8, 10])
```

Agenda

- Review
  - numpy
  - matplotlib
- Lesson
  - notebooks and scripts
  - pandas
- Readings
  - Python for Data Analysis by Wes McKinney
  - http://pandas.pydata.org/pandas-docs/stable/index.html

Objectives

- What are different data types? How can we choose appropriate charts for data types?
- What are good practices for visualization
  - Conditioning
  - Scale
  - Jiggling Baseline
  - Transformation

Agenda

- Review
  - numpy
  - matplotlib
- Lesson
  - notebooks and scripts
  - pandas
- Readings
  - Python for Data Analysis by Wes McKinney
  - http://pandas.pydata.org/pandas-docs/stable/index.html

```
import matplotlib as mpl
import matplotlib.pyplot as plt
```

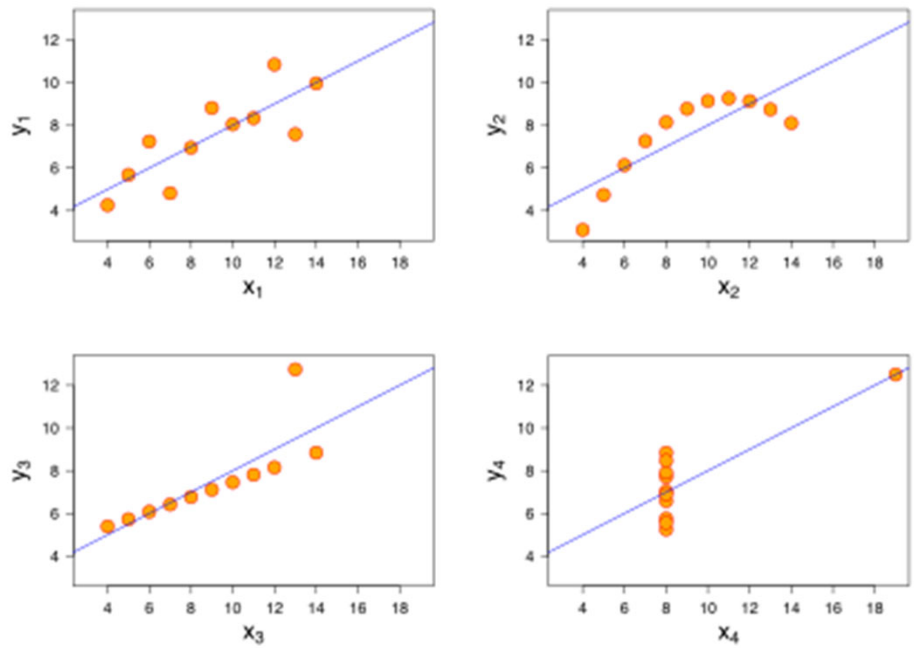
Objectives

- What are different data types? How can we choose appropriate charts for data types?
- What are good practices for visualization
  - Conditioning
  - Scale
  - Jiggling Baseline
  - Transformation

matplotlib

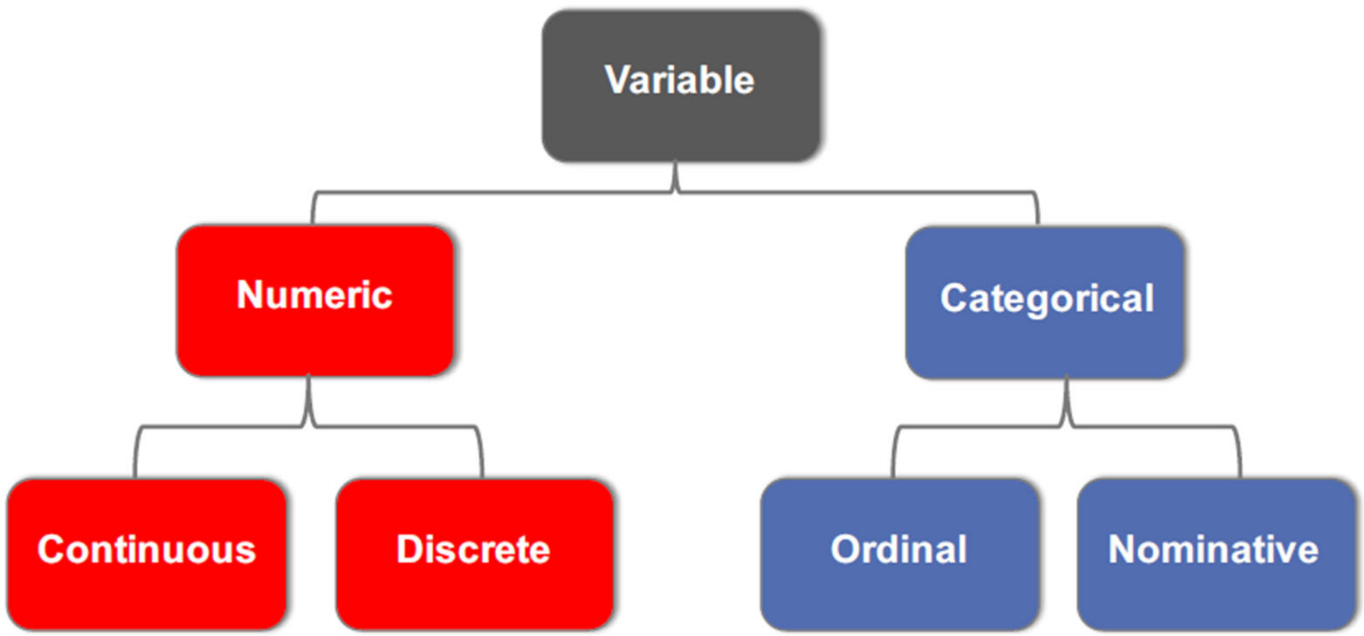
- Why visualization?

Property	Value	Accuracy
Mean of *x*	9	exact
Sample variance of *x*	11	exact
Mean of *y*	7.50	to 2 decimal places
Sample variance of *y*	4.125	±0.003



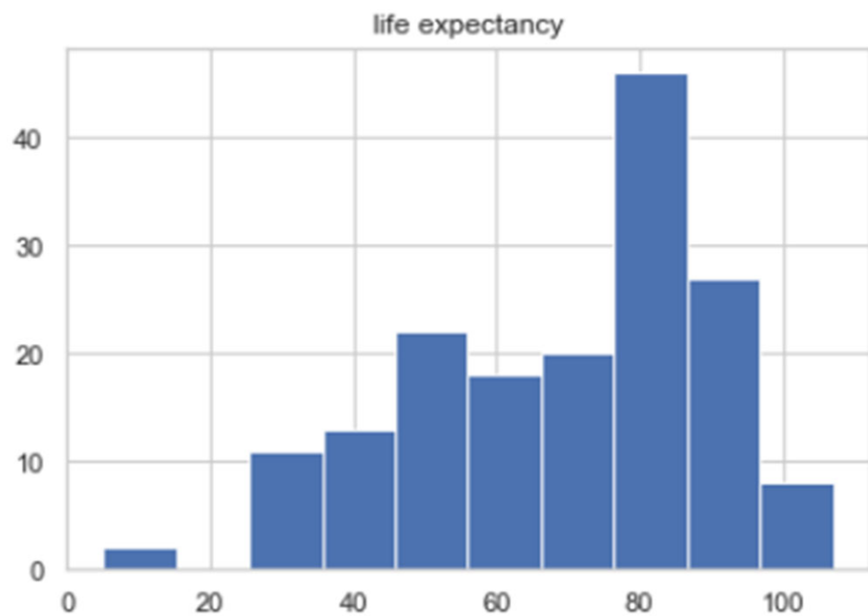
matplotlib

- Data Types



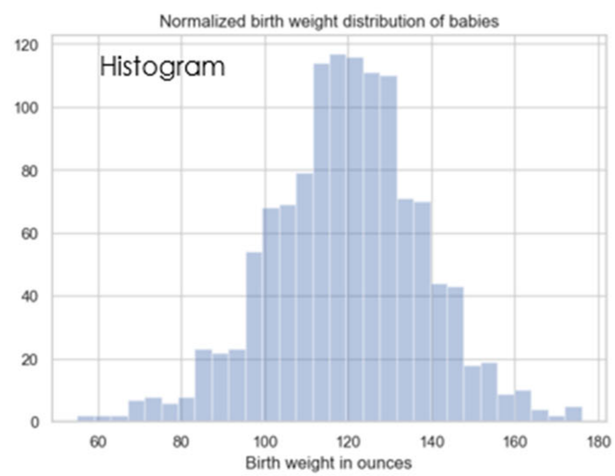
matplotlib

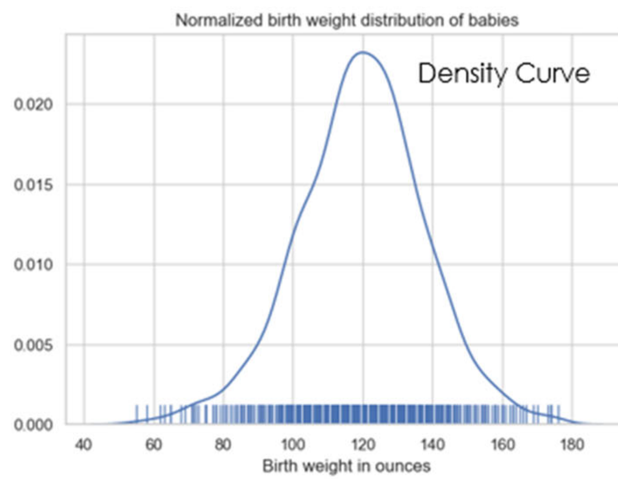
- One Quantitative Variable



matplotlib

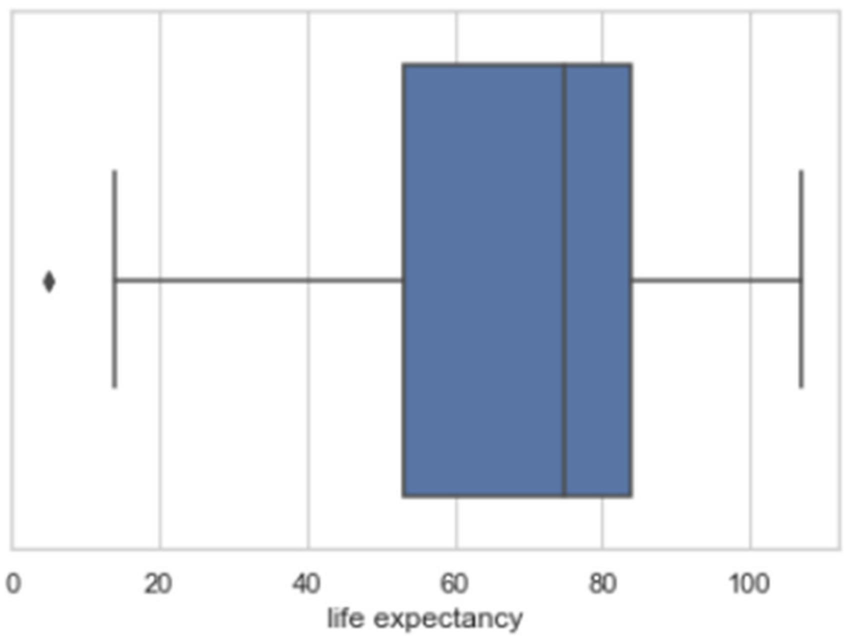
- One Quantitative Variable





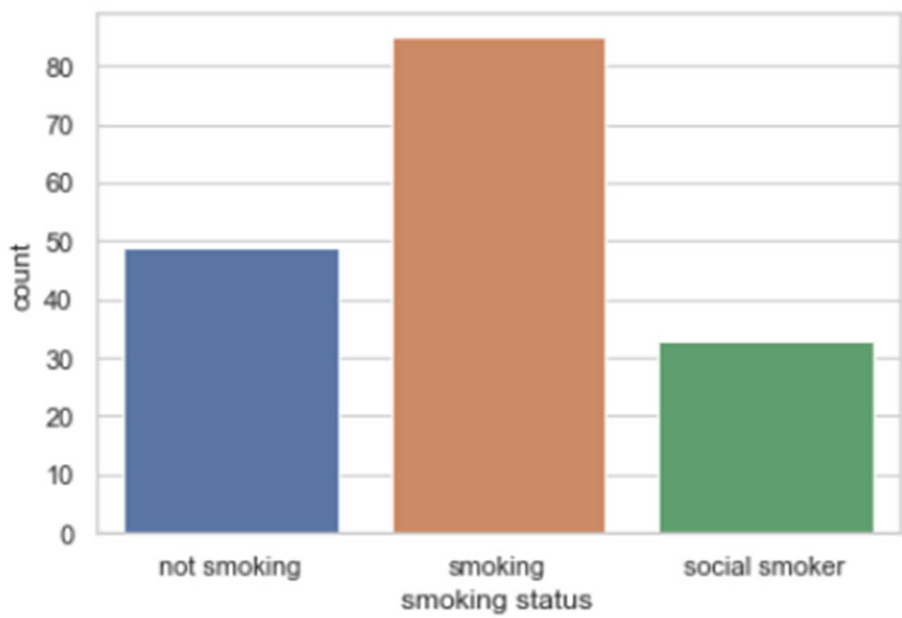
matplotlib

- One Quantitative Variable



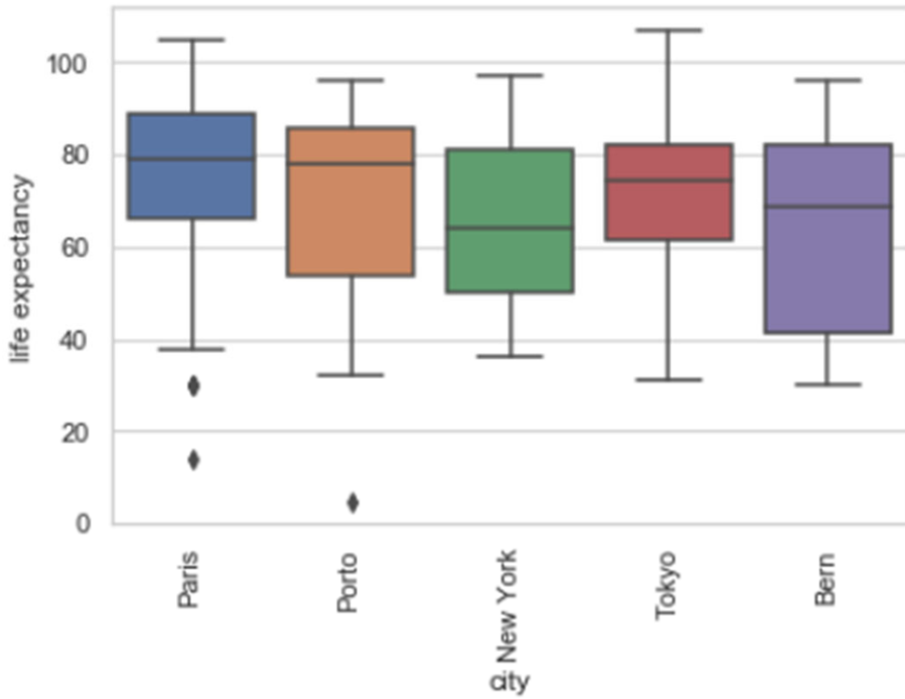
matplotlib

- One Qualitative Variable



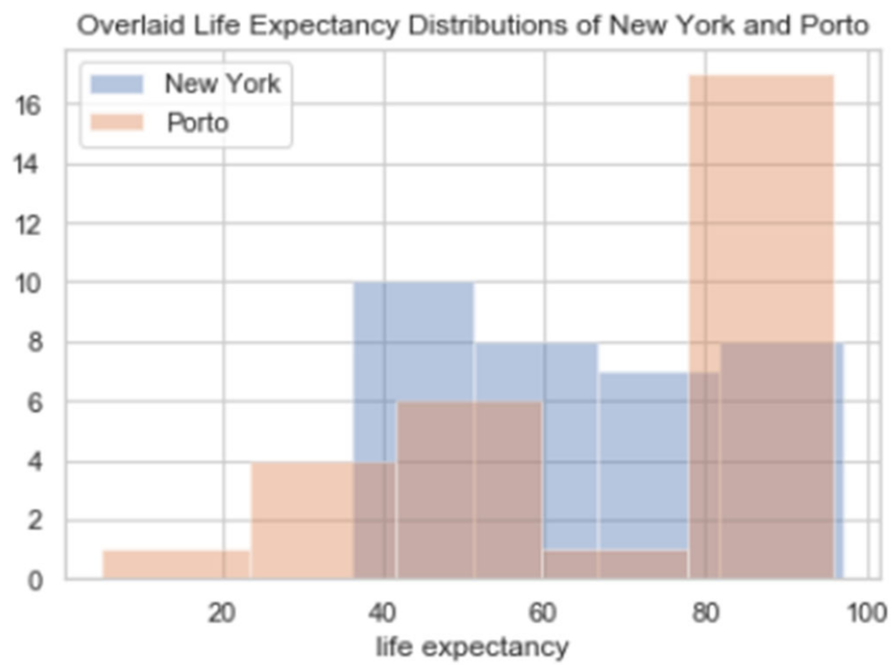
matplotlib

- One Qualitative Variable and One Quantitative Variable



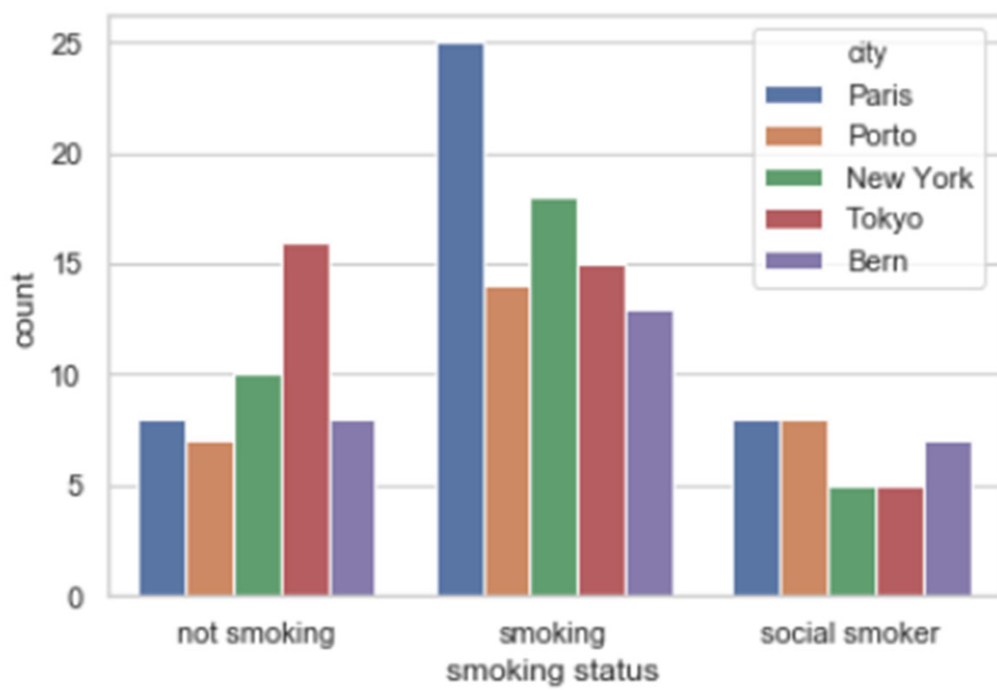
matplotlib

- One Qualitative Variable and One Quantitative Variable



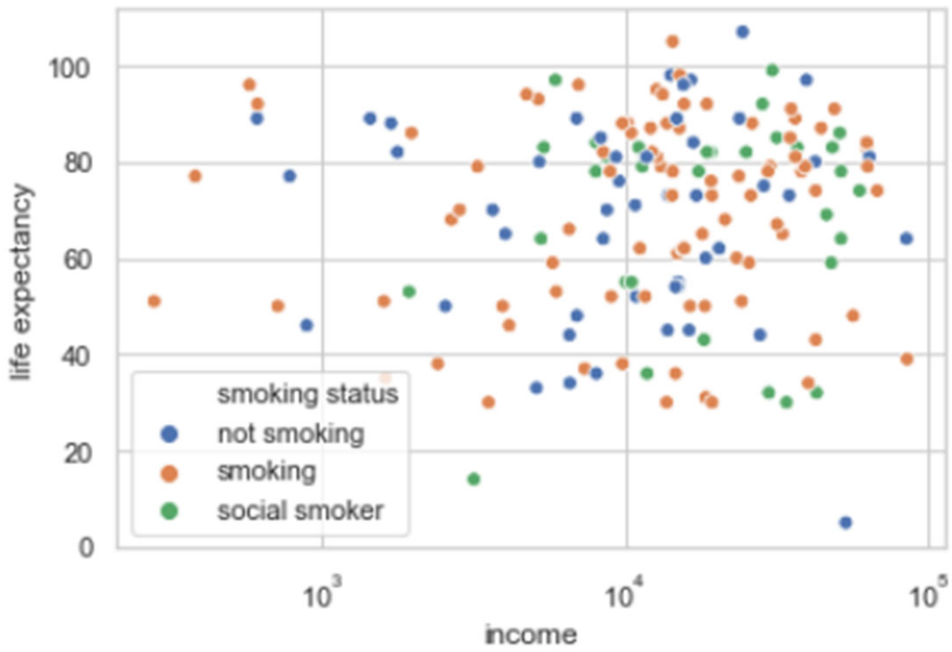
matplotlib

- Multiple Qualitative Variables



matplotlib

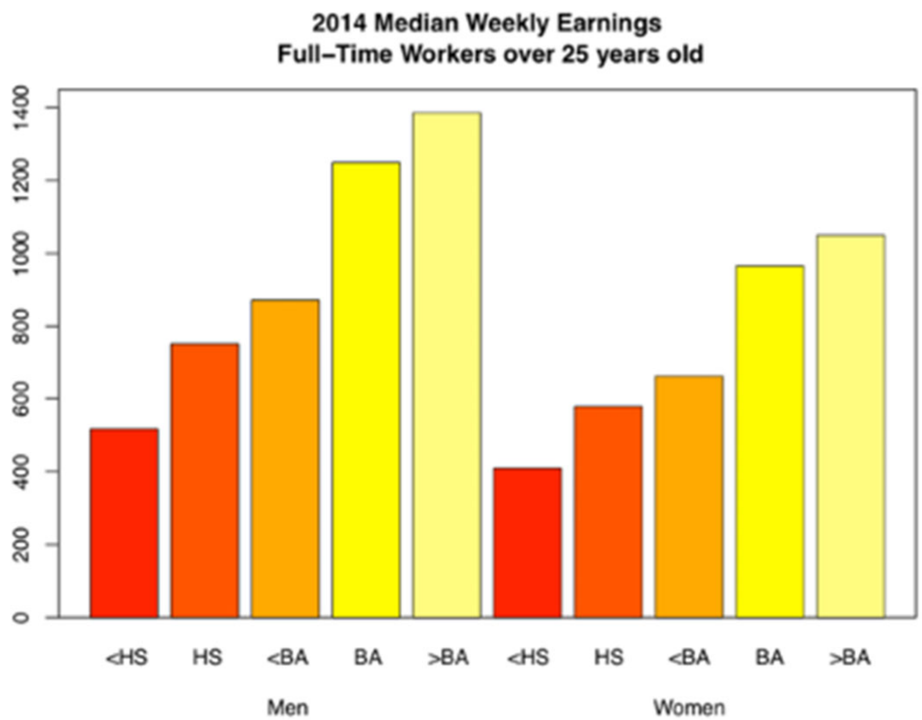
- Multiple Quantitative Variables
- Check out sns.pairplot !



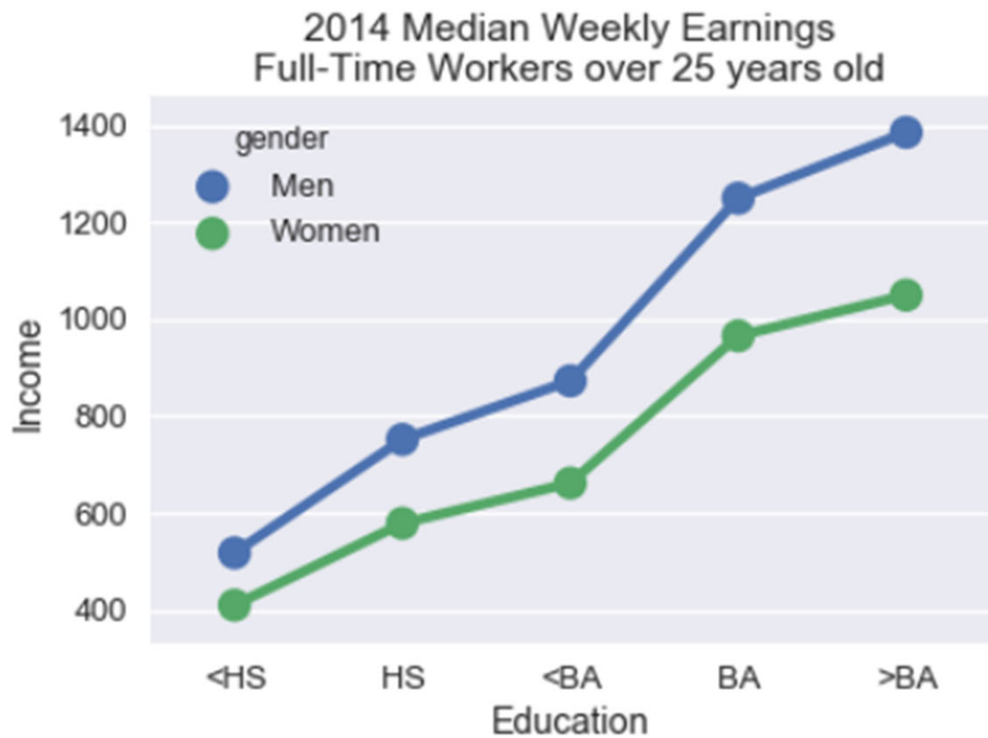
matplotlib

- Multiple Quantitative Variables

matplotlib

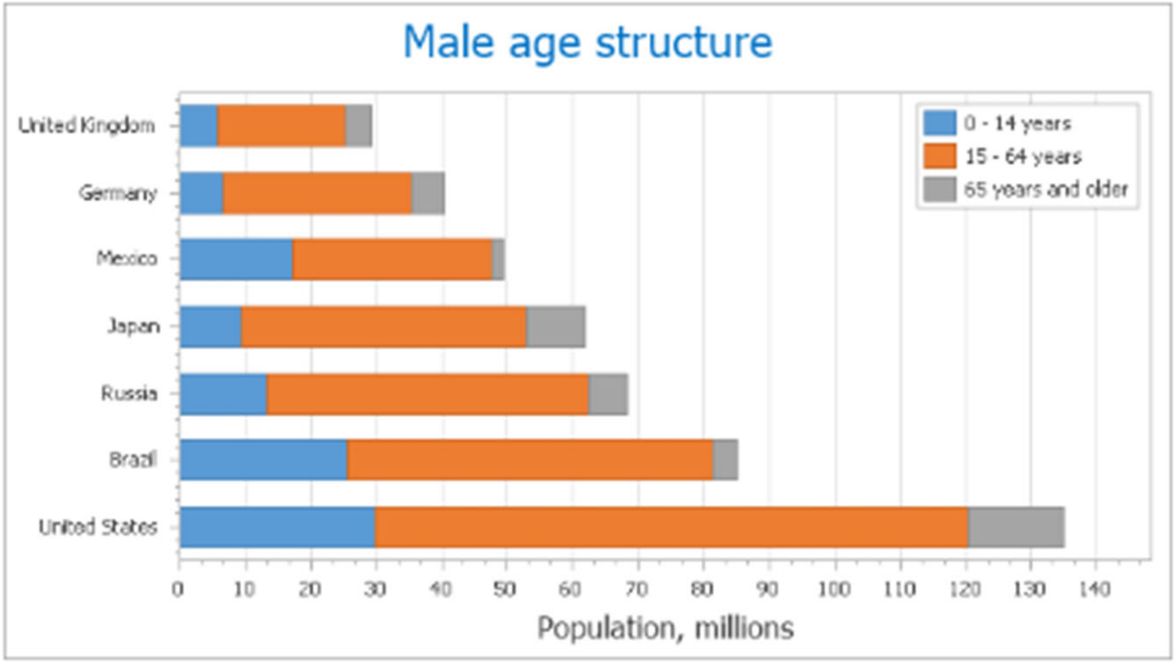

2014 Median Weekly Earnings
Full−Time Workers over 25 years old
0
200
400
600
800
1000
1200
1400
<HS
HS
<BA
BA
>BA
<HS
HS
<BA
BA
>BA
Men
Women

matplotlib

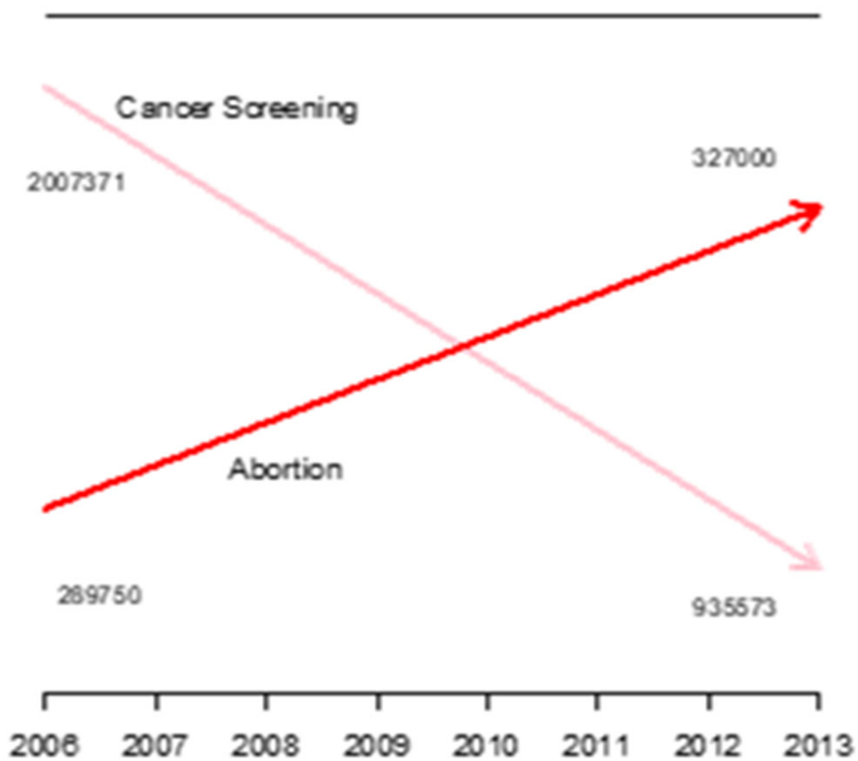


2014 Median Weekly Earnings
Full-Time Workers over 25 years old
gender
Men
Women
Income
1400
1200
1000
800
600
400
<HS
HS
<BA
BA
>BA
Education

matplotlib

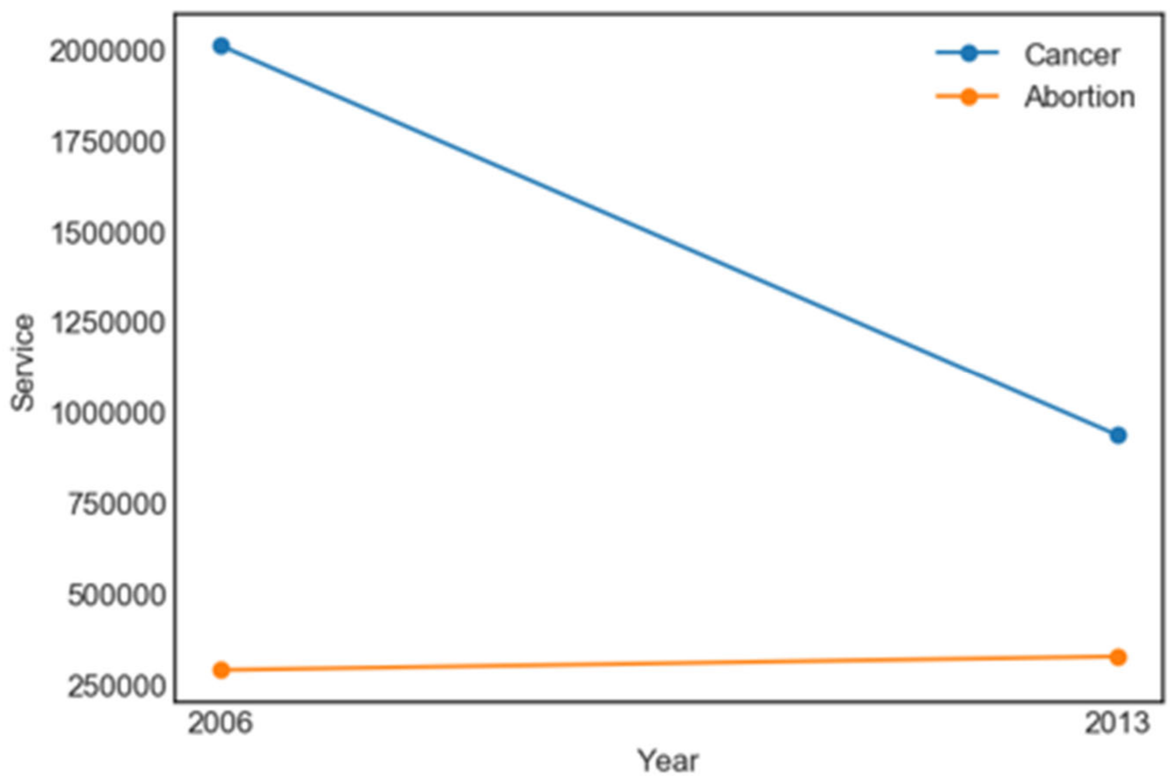

Male age structure
United Kingdom
Germany
Mexico
Japan
Russia
Brazil
United States
0 - 14 years
15 - 64 years
65 years and older
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
Population, millions

matplotlib



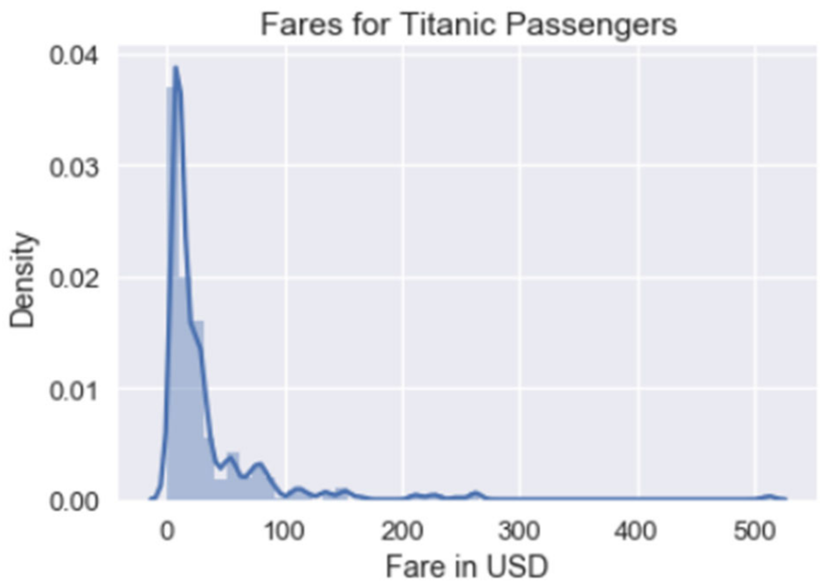
Cancer Screening
2007371
327000
Abortion
289750
935573
2006
2007
2008
2009
2010
2011
2012
2013

matplotlib



Cancer
Abortion
2000000
1750000
1500000
1250000
1000000
750000
500000
250000
Service
2006
2013
Year

matplotlib



Fares for Titanic Passengers
0.04
0.03
0.02
0.01
0.00
Density
0
100
200
300
400
500
Fare in USD

Agenda

- Review
  - numpy
  - matplotlib
- Lesson
  - notebooks and scripts
  - pandas
- Readings
  - Python for Data Analysis by Wes McKinney
  - http://pandas.pydata.org/pandas-docs/stable/index.html

Objectives

- What are the advantages and disadvantages of scripts and notebooks?
- Is it possible to link notebook and scripts?
- What are the command line tools to convert between scripts and notebooks?

Jupytext

- Notebooks
  - Advantages
    - Combine text, code and charts
    - Support multiple languages including Python and R

Jupytext

- Notebooks
  - Advantages
    - Combine text, code and charts
    - Support multiple languages including Python and R
  - Disadvantages
    - JSON format makes version control difficult
    - Lacking debugger to determine errors
    - Refactoring difficult across multiple cells which may not reflect logical order of program

Jupytext

Link a notebook and a script so changes in one become changes in the other!

- Notebooks
  - Advantages
    - Combine text, code and charts
    - Support multiple languages including Python and R
  - Disadvantages
    - JSON format makes version control difficult
    - Lacking debugger to determine errors
    - Refactoring difficult across multiple cells which may not reflect logical order of program

Agenda

- Review
  - numpy
  - matplotlib
- Lesson
  - notebooks and scripts
  - pandas
- Readings
  - Python for Data Analysis by Wes McKinney
  - http://pandas.pydata.org/pandas-docs/stable/index.html

import pandas as pd

Objectives

- What are the components of pandas for storing data?
  - Series
  - DataFrame
- How can we access data in a table?

pandas

- Tabular data consisting of rows and columns common in data analysis
  - Rows are observations in sample
  - Columns are features of the data
- Often ***panel data*** with rows consisting of timestamps

pandas

- Tabular data consisting of rows and columns common in data analysis
  - Rows are observations in sample
  - Columns are features of the data
- Often ***panel data*** with rows consisting of timestamps

- pandas packages takes data structures and operations from languages like R and SQL
- Built on top of
  - numpy
  - scipy
  - some components of matplotlib

pandas

- Tabular data consisting of rows and columns common in data analysis
  - Rows are observations in sample
  - Columns are features of the data
- Often ***panel data*** with rows consisting of timestamps

```
SELECT e.emp_id,
       e.emp_name,
       d.dept_name
FROM Employee e
INNER JOIN Department d ON e.dept_id = d.dept_id
WHERE d.dept_name = 'finance'
  AND e.emp_name LIKE '%A%'
  AND e.salary > 500;
```

- pandas packages takes data structures and operations from languages like R and SQL
- Built on top of
  - numpy
  - scipy
  - some components of matplotlib

Series

- One-dimensional object containing
  - Data
  - Labels (called ***index***)
- Like array in numpy

```
# Creating a series
index = ['a','b','c','d','e']
series = pd.Series(np.arange(5), index=index)
print(series)
```

```
a    0
b    1
c    2
d    3
e    4
dtype: int64
```

Series

- One-dimensional object containing
  - Data
  - Labels (called ***index***)
  - Default index is range(N) where N is the length
- Index used for
  - lookups
  - aligning tables
  - supports hierarchical indexes, where each label is a tuple

```python
# Creating a series
index = ['a','b','c','d','e']
series = pd.Series(np.arange(5), index=index)
print(series)
```

```
a    0
b    1
c    2
d    3
e    4
dtype: int64
```

DataFrame

- Two-dimensional object containing
  - Data
  - Labels (called ***index***)
  - Columns (ordered)
- Like a dictionary of Series where all Series have the same index

```
# Creating a dataframe with a dictionary
d = {'state' : ['FL', 'FL', 'GA', 'GA', 'GA'],
     'year' :  [2010, 2011, 2008, 2010, 2011],
     'pop' :   [18.8, 19.1, 9.7, 9.7, 9.8]}

df_d = pd.DataFrame(d)
print(df_d)
```

```
    pop state  year
0  18.8    FL  2010
1  19.1    FL  2011
2   9.7    GA  2008
3   9.7    GA  2010
4   9.8    GA  2011
```

Input and Output

- We can store tabular data in many formats
  - Comma Separated Values (CSV)
  - Tab Separated Values (TSV)
- Note that these file formats are not nested
  - Each row and column contains one entry

```
# the first row becomes the column indices
df = pd.read_csv('simple.csv')
print(df)

print(df.columns.values)
```

```
   a   b   c   d message
0  1   2   3   4   hello
1  5   6   7   8   world
2  9  10  11  12     foo
['a' 'b' 'c' 'd' 'message']
```

Demo

Take-Aways

- What is Series
- What is DataFrame
- How does pandas support operations like numpy?
- How does pandas support operations like query languages?

Summary

- Review
  - numpy
  - matplotlib
- Demos
  - notebooks and scripts
  - pandas
- Readings
  - http://wiki.scipy.org/Tentative_NumPy_Tutorial
  - http://matplotlib.org/Matplotlib.pdf