Microsoft

INTRO TO PYTHON FOR DATA SCIENCE

Numpy

Lists Recap

- Powerful
- Collection of values
- Hold different types
- Change, add, remove
- Need for Data Science
  - Mathematical operations over collections
  - Speed

Illustration

```
In [1]: height = [1.73, 1.68, 1.71, 1.89, 1.79]

In [2]: height
Out[2]: [1.73, 1.68, 1.71, 1.89, 1.79]

In [3]: weight = [65.4, 59.2, 63.6, 88.4, 68.7]

In [4]: weight
Out[4]: [65.4, 59.2, 63.6, 88.4, 68.7]

In [5]: weight / height ** 2
TypeError: unsupported operand type(s) for **: 'list' and 'int'
```

Solution: Numpy

- Numeric Python
- Alternative to Python List: Numpy Array
- Calculations over entire arrays
- Easy and Fast
- Installation
  - In the terminal: `pip3 install numpy`

Numpy

```
In [6]: import numpy as np

In [7]: np_height = np.array(height)

In [8]: np_height
Out[8]: array([ 1.73,  1.68,  1.71,  1.89,  1.79])

In [9]: np_weight = np.array(weight)

In [10]: np_weight
Out[10]: array([ 65.4,  59.2,  63.6,  88.4,  68.7])

In [11]: bmi = np_weight / np_height ** 2

In [12]: bmi
Out[12]: array([ 21.852,  20.975,  21.75 ,  24.747,  21.441])
```

Numpy

Element-wise calculations

```
In [6]: import numpy as np

In [7]: np_height = np.array(height)

In [8]: np_height
Out[8]: array([ 1.73,  1.68,  1.71,  1.89,  1.79])

In [9]: np_weight = np.array(weight)

In [10]: np_weight
Out[10]: array([ 65.4,  59.2,  63.6,  88.4,  68.7])

In [11]: bmi = np_weight / np_height ** 2

In [12]: bmi
Out[12]: array([ 21.852,  20.975,  21.75 ,  24.747,  21.441])
```

```
= 65.5/1.73 ** 2
```

Comparison

```
In [13]: height = [1.73, 1.68, 1.71, 1.89, 1.79]

In [14]: weight = [65.4, 59.2, 63.6, 88.4, 68.7]

In [15]: weight / height ** 2
TypeError: unsupported operand type(s) for **: 'list' and 'int'



In [16]: np_height = np.array(height)

In [17]: np_weight = np.array(weight)

In [18]: np_weight / np_height ** 2
Out[18]: array([ 21.852,  20.975,  21.75 ,  24.747,  21.441])
```

Numpy: remarks

```
In [19]: np.array([1.0, "is", True])
Out[19]:
array(['1.0', 'is', 'True'],
      dtype='<U32')

In [20]: python_list = [1, 2, 3]

In [21]: numpy_array = np.array([1, 2, 3])

In [22]: python_list + python_list
Out[22]: [1, 2, 3, 1, 2, 3]

In [23]: numpy_array + numpy_array
Out[23]: array([2, 4, 6])
```

Numpy arrays: contain only one type

Different types: different behavior!

Numpy Subsetting

```
In [24]: bmi
Out[24]: array([ 21.852,  20.975,  21.75 ,  24.747,  21.441])

In [25]: bmi[1]
Out[25]: 20.975

In [26]: bmi > 23
Out[26]: array([False, False, False,  True, False], dtype=bool)

In [27]: bmi[bmi > 23]
Out[27]: array([ 24.747])
```

Microsoft

INTRO TO PYTHON FOR DATA SCIENCE

Let’s practice!

Microsoft

INTRO TO PYTHON FOR DATA SCIENCE

2D Numpy Arrays

Type of Numpy Arrays

```
In [1]: import numpy as np

In [2]: np_height = np.array([1.73, 1.68, 1.71, 1.89, 1.79])

In [3]: np_weight = np.array([65.4, 59.2, 63.6, 88.4, 68.7])

In [4]: type(np_height)
Out[4]: numpy.ndarray

In [5]: type(np_weight)
Out[5]: numpy.ndarray
```

ndarray = N-dimensional array

2D Numpy Arrays

```
In [6]: np_2d = np.array([[1.73, 1.68, 1.71, 1.89, 1.79],
                          [65.4, 59.2, 63.6, 88.4, 68.7]])

In [7]: np_2d
Out[7]:
array([[  1.73,   1.68,   1.71,   1.89,   1.79],
       [ 65.4 ,  59.2 ,  63.6 ,  88.4 ,  68.7 ]])

In [8]: np_2d.shape
Out[8]: (2, 5)

In [9]: np.array([[1.73, 1.68, 1.71, 1.89, 1.79],
                  [65.4, 59.2, 63.6, 88.4, "68.7"]])
Out[9]:
array([['1.73', '1.68', '1.71', '1.89', '1.79'],
       ['65.4', '59.2', '63.6', '88.4', '68.7']],
      dtype='<U32')
```

2 rows, 5 columns

Single type!

Subsetting

```
                 0       1       2       3       4
array([[  1.73,   1.68,   1.71,   1.89,   1.79],   0
       [  65.4,   59.2,   63.6,   88.4,   68.7]])  1
```

```
In [10]: np_2d[0]
Out[10]: array([ 1.73,  1.68,  1.71,  1.89,  1.79])

In [11]: np_2d[0][2]
Out[11]: 1.71

In [12]: np_2d[0,2]
Out[12]: 1.71
```

Subsetting

```
                 0      1      2      3      4
array([[  1.73,  1.68,  1.71,  1.89,  1.79],   0
       [  65.4,  59.2,  63.6,  88.4,  68.7]])  1
```

```
In [10]: np_2d[0]
Out[10]: array([ 1.73,  1.68,  1.71,  1.89,  1.79])

In [11]: np_2d[0][2]
Out[11]: 1.71

In [12]: np_2d[0,2]
Out[12]: 1.71

In [13]: np_2d[:,1:3]
Out[13]:
array([[  1.68,   1.71],
       [ 59.2 ,  63.6 ]])
```

Subsetting

```
                0      1      2      3      4
array([[ 1.73,  1.68,  1.71,  1.89,  1.79],   0
       [ 65.4,  59.2,  63.6,  88.4,  68.7]])  1
```

```
In [10]: np_2d[0]
Out[10]: array([ 1.73,  1.68,  1.71,  1.89,  1.79])

In [11]: np_2d[0][2]
Out[11]: 1.71

In [12]: np_2d[0,2]
Out[12]: 1.71

In [13]: np_2d[:,1:3]
Out[13]:
array([[  1.68,   1.71],
       [ 59.2 ,  63.6 ]])

In [14]: np_2d[1,:]
Out[14]: array([ 65.4,  59.2,  63.6,  88.4,  68.7])
```

Microsoft

INTRO TO PYTHON FOR DATA SCIENCE

Let’s practice!

Microsoft

INTRO TO PYTHON FOR DATA SCIENCE

Numpy: Basic Statistics

Data analysis

- Get to know your data
- Little data -> simply look at it
- Big data -> ?

City-wide survey

```
In [1]: import numpy as np

In [2]: np_city = ... # Implementation left out

In [3]: np_city
Out[3]:
array([[  1.64,  71.78],
       [  1.37,  63.35],
       [  1.6 ,  55.09],
       ...,
       [  2.04,  74.85],
       [  2.04,  68.72],
       [  2.01,  73.57]])
```

Numpy

```
In [4]: np.mean(np_city[:,0])
Out[4]: 1.7472

In [5]: np.median(np_city[:,0])
Out[5]: 1.75

In [6]: np.corrcoef(np_city[:,0], np_city[:,1])
Out[6]:
array([[ 1.     , -0.01802],
       [-0.01803,  1.     ]])

In [7]: np.std(np_city[:,0])
Out[7]: 0.1992
```

- `sum()`, `sort()`, ...
- Enforce single data type: speed!

Generate data


```
In [8]: height = np.round(np.random.normal(1.75, 0.20, 5000), 2)

In [9]: weight = np.round(np.random.normal(60.32, 15, 5000), 2)

In [10]: np_city = np.column_stack((height, weight))
```

Microsoft

INTRO TO PYTHON FOR DATA SCIENCE

Let’s practice!

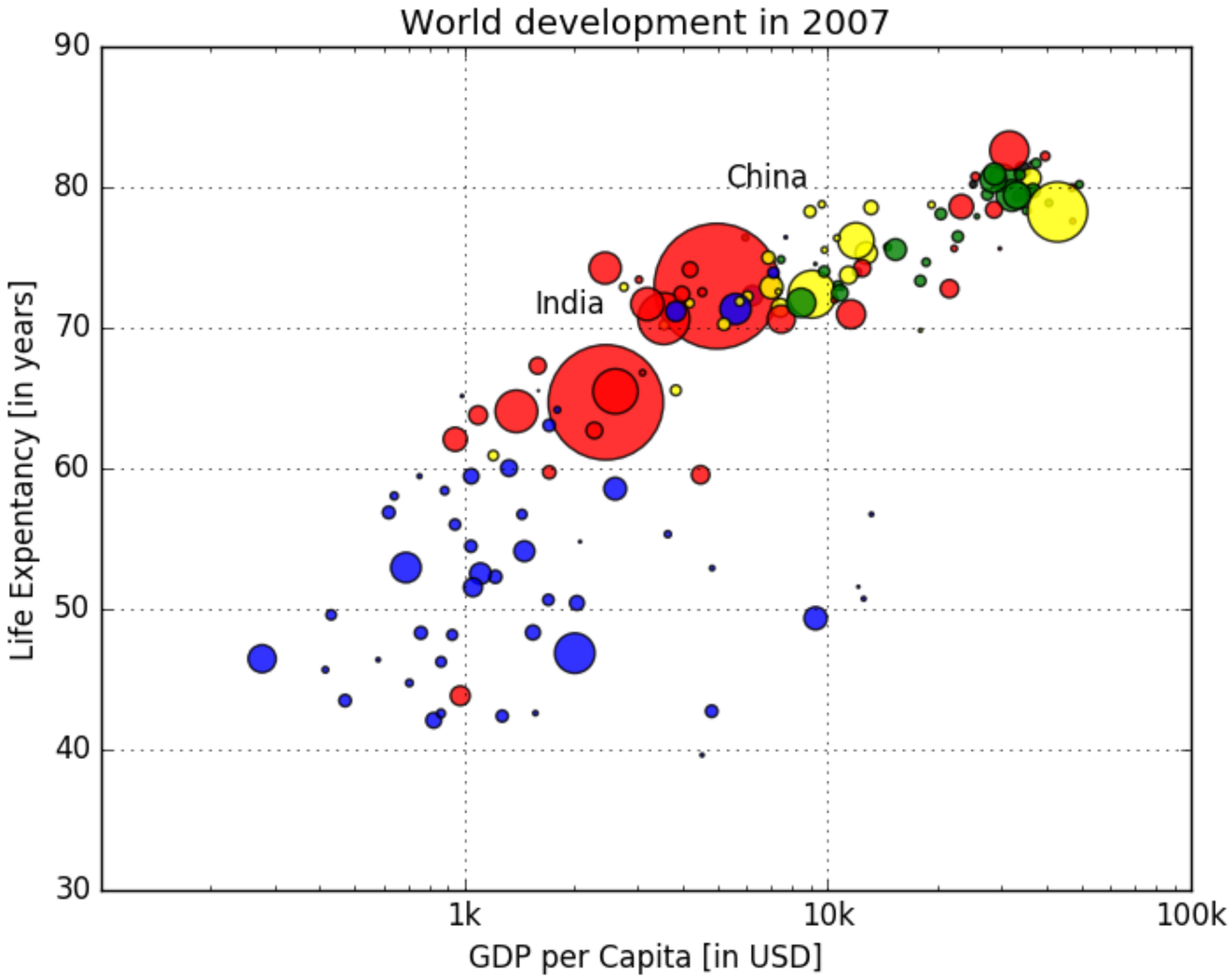
INTRO TO PYTHON FOR DATA SCIENCE

Basic Plots with Matplotlib

Data Visualization

- Very important in Data Analysis
  - Explore data
  - Report insights

Example

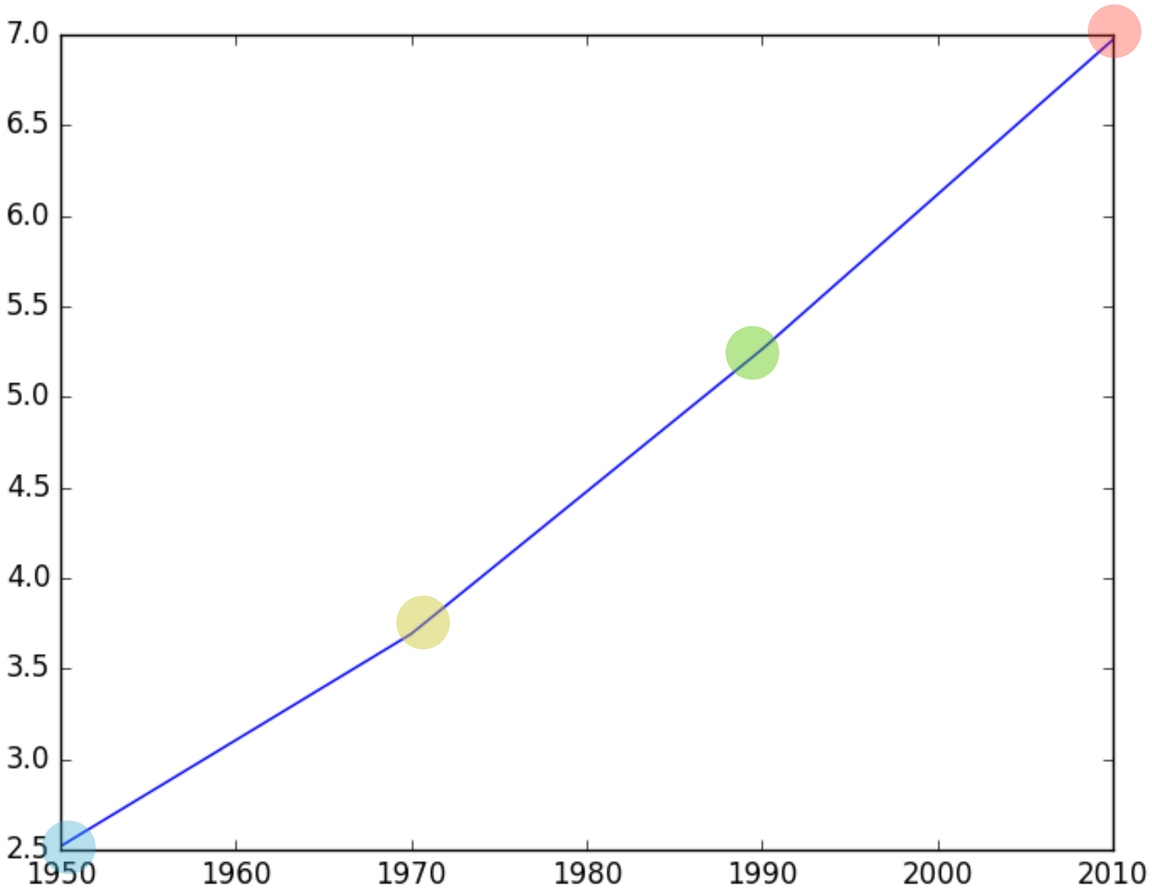

World development in 2007
Life Expentancy [in years]
GDP per Capita [in USD]
90
80
70
60
50
40
30
1k
10k
100k
China
India

Matplotlib

```
In [1]: import matplotlib.pyplot as plt

In [2]: year = [1950, 1970, 1990, 2010]

In [3]: pop = [2.519, 3.692, 5.263, 6.972]

In [4]: plt.plot(year, pop)

In [5]: plt.show()
```

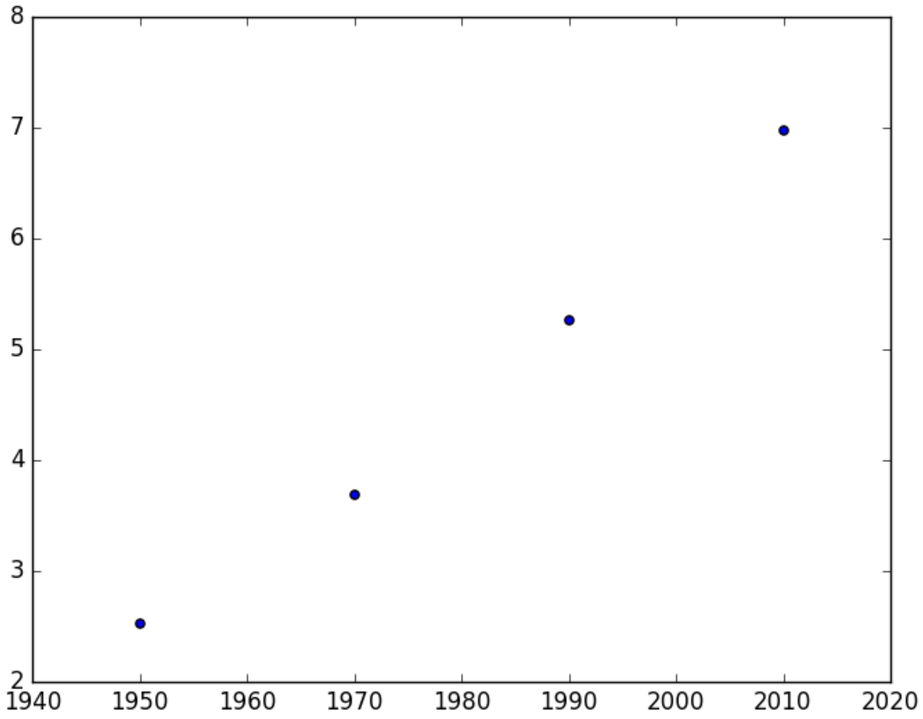
Matplotlib

```
year = [1950, 1970, 1990, 2010]
pop  = [2.519, 3.692, 5.263, 6.972]
```



Scatter plot

```
In [6]: plt.scatter(year, pop)

In [7]: plt.show()
```



Microsoft

INTRO TO PYTHON FOR DATA SCIENCE

Let’s practice!

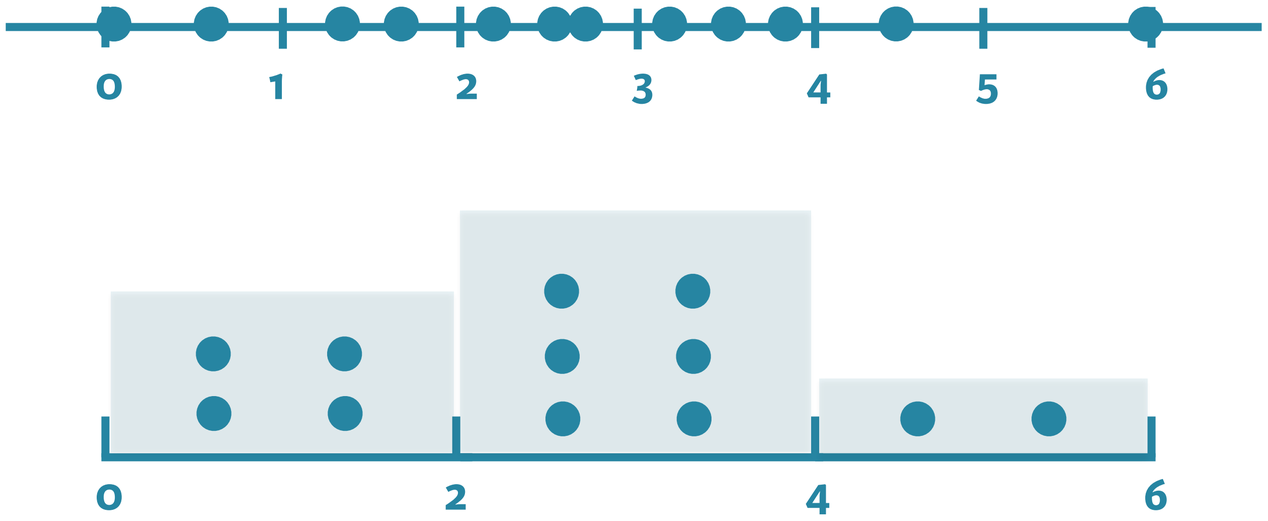
Microsoft

INTRO TO PYTHON FOR DATA SCIENCE

Histograms

Histogram

- Explore dataset
- Get idea about distribution

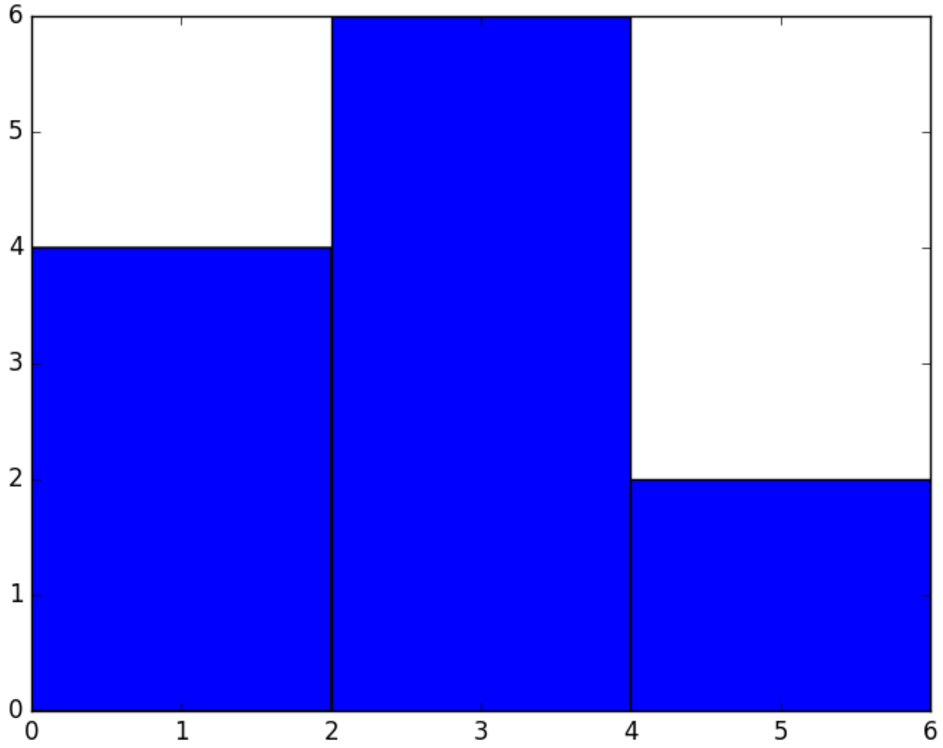


Matplotlib

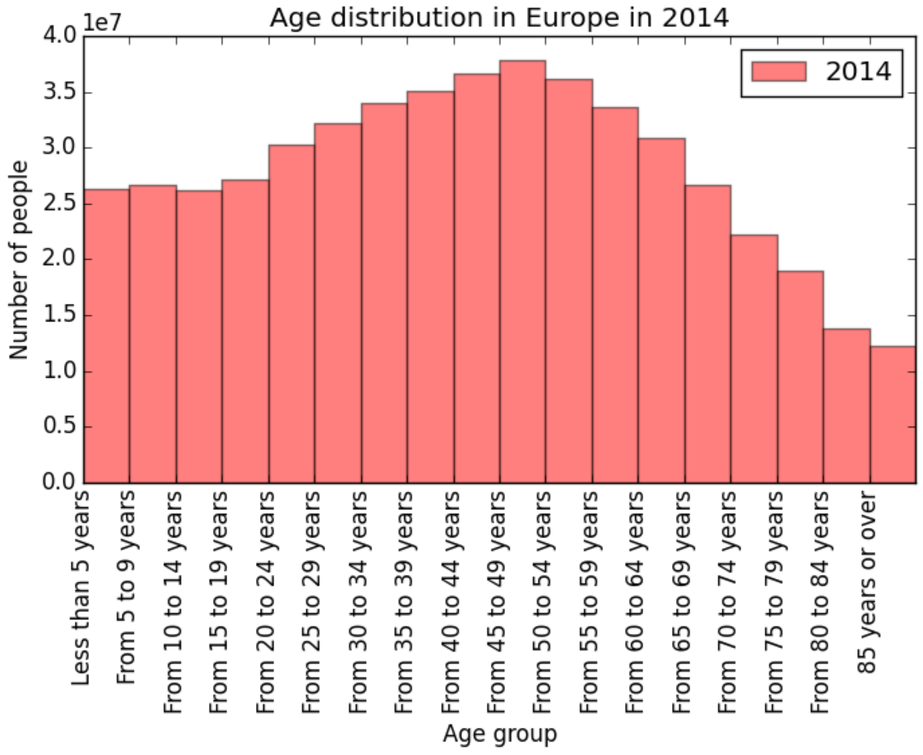
```
In [1]: import matplotlib.pyplot as plt

In [2]: help(plt.hist)

  Help on function hist in module matplotlib.pyplot:

  hist(x, bins=10, range=None, normed=False, weights=None,
  cumulative=False, bottom=None, histtype='bar', align='mid',
  orientation='vertical', rwidth=None, log=False, color=None,
  label=None, stacked=False, hold=None, data=None, **kwargs)
      Plot a histogram.

      Compute and draw the histogram of *x*. The return value is a
      tuple (*n*, *bins*, *patches*) or ([*n0*, *n1*, ...],
      *bins*, [*patches0*, *patches1*,...]) if the input contains
      multiple data.

  ...
```

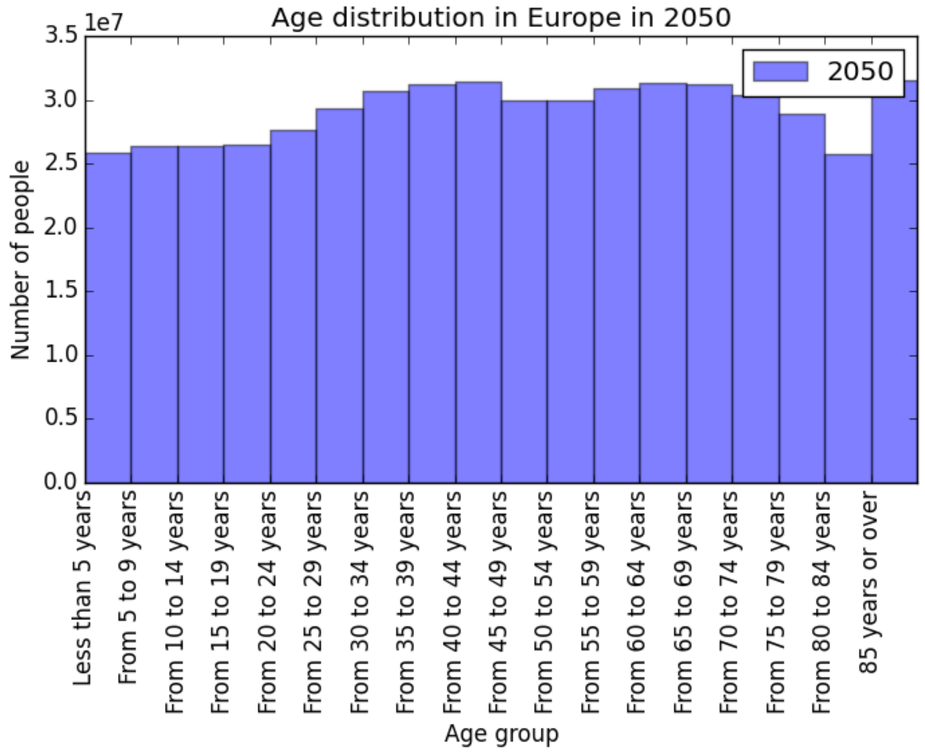
Matplotlib example

```
In [3]: values = [0,0.6,1.4,1.6,2.2,2.5,2.6,3.2,3.5,3.9,4.2,6]
In [4]: plt.hist(values, bins = 3)
In [5]: plt.show()
```



Age Distribution





Microsoft

INTRO TO PYTHON FOR DATA SCIENCE

Let’s practice!

Microsoft

INTRO TO PYTHON FOR DATA SCIENCE

Customization

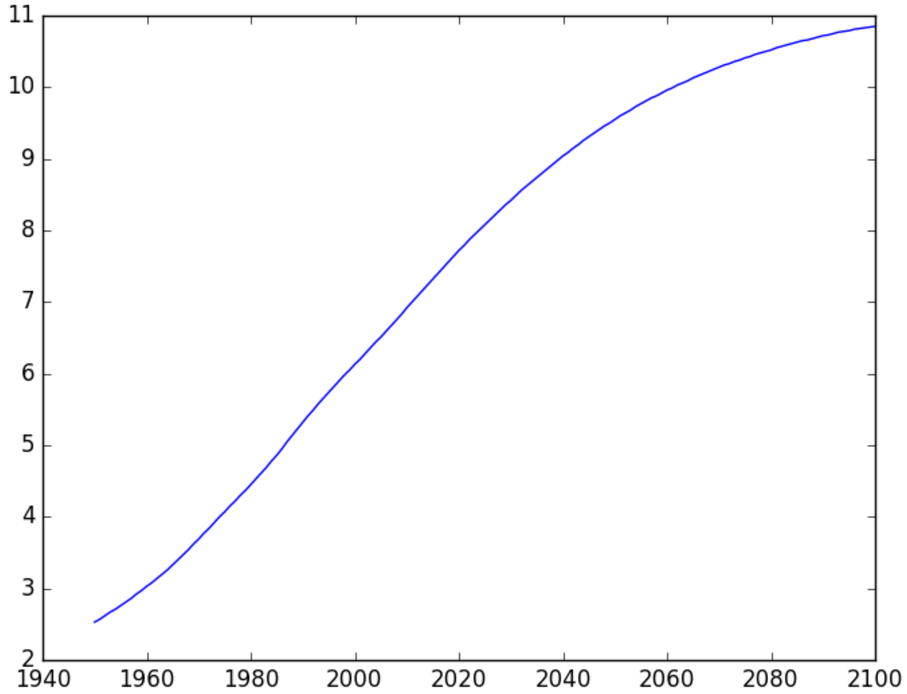
Data Visualization

- Science & Art
- Many options
  - Different plot types
  - Many customizations
- Choice depends on:
  - Data
  - Story you want to tell

Basic Plot

my_script.py

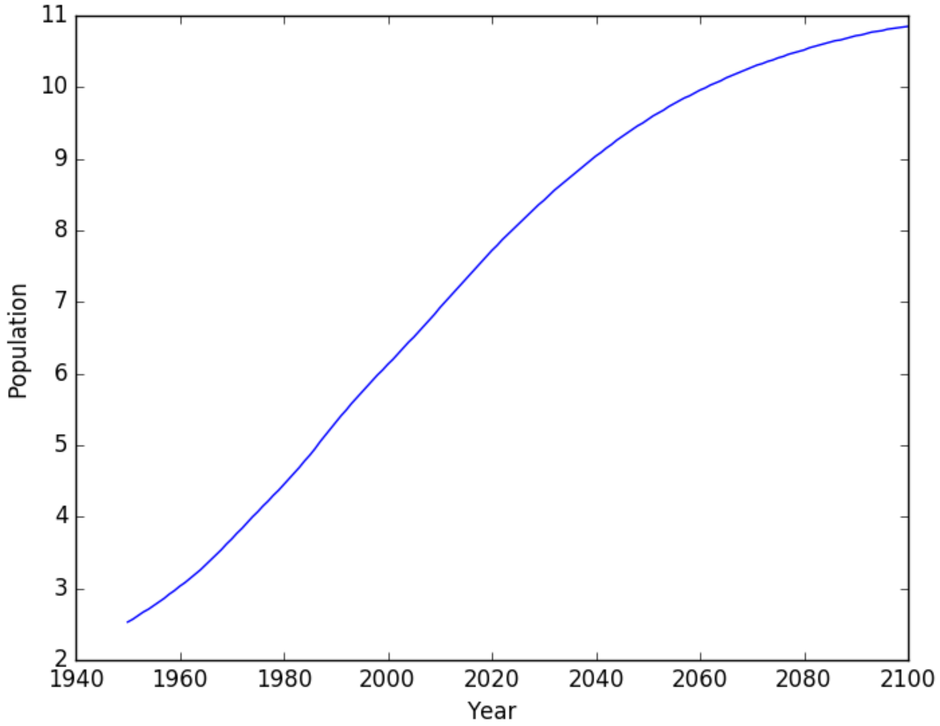
```
import matplotlib.pyplot as plt
year = ... # Implementation left out
population = ... # Implementation left out

plt.plot(year, population)
plt.show()
```



Axis labels

my_script.py

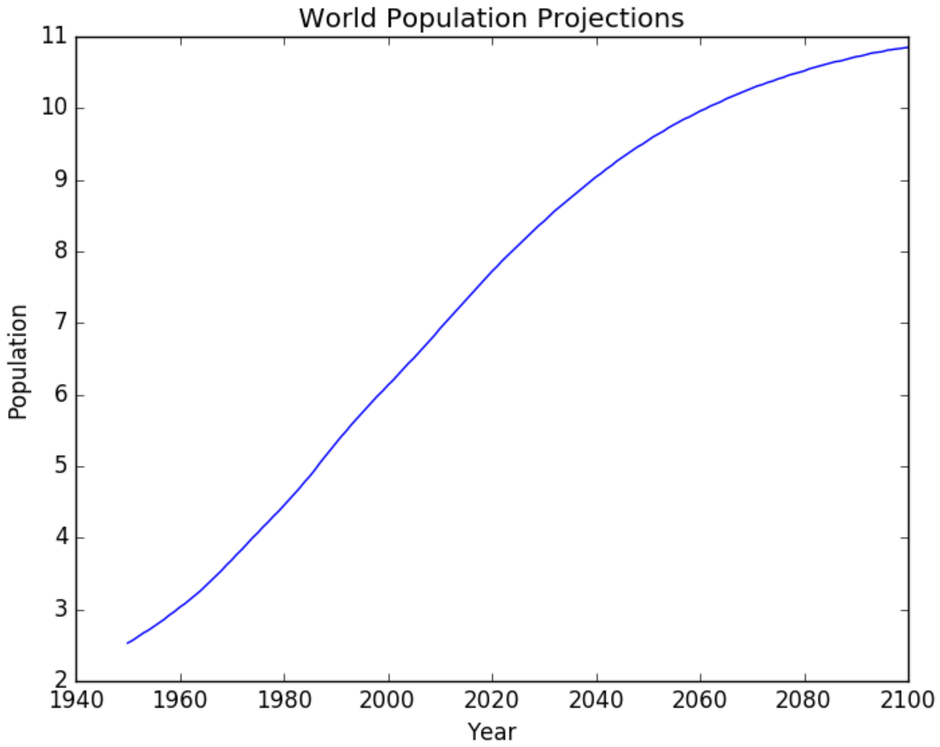
```
import matplotlib.pyplot as plt
year = ... # Implementation left out
population = ... # Implementation left out

plt.plot(year, population)

plt.xlabel('Year')
plt.ylabel('Population')

plt.show()
```



Title

my_script.py

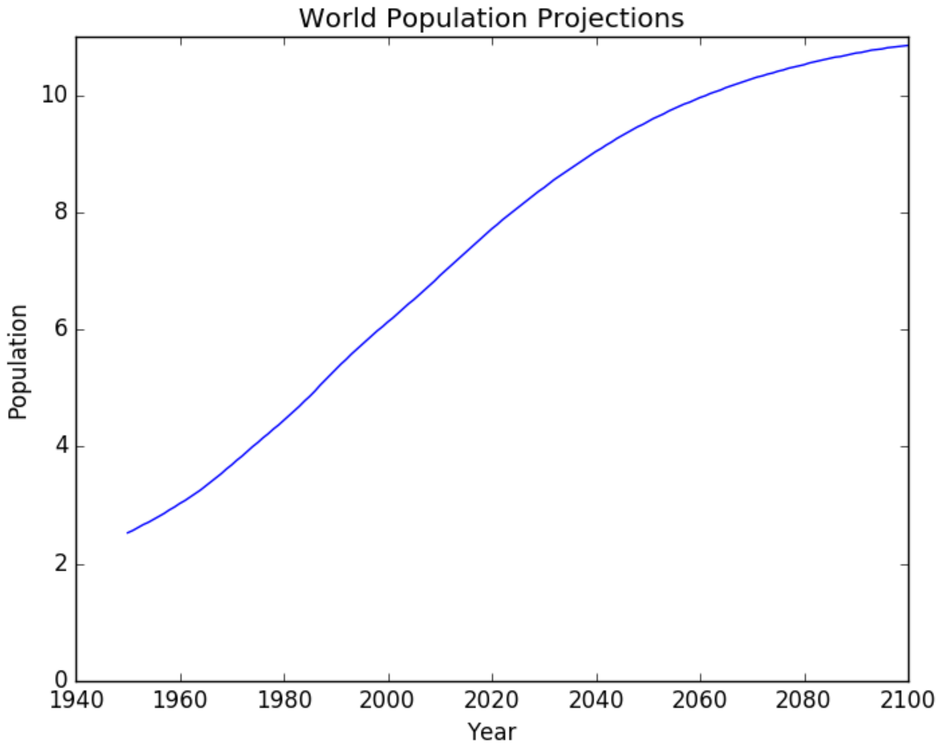
```
import matplotlib.pyplot as plt
year = ... # Implementation left out
population = ... # Implementation left out

plt.plot(year, population)

plt.xlabel('Year')
plt.ylabel('Population')
plt.title('World Population Projections')

plt.show()
```



Ticks

my_script.py

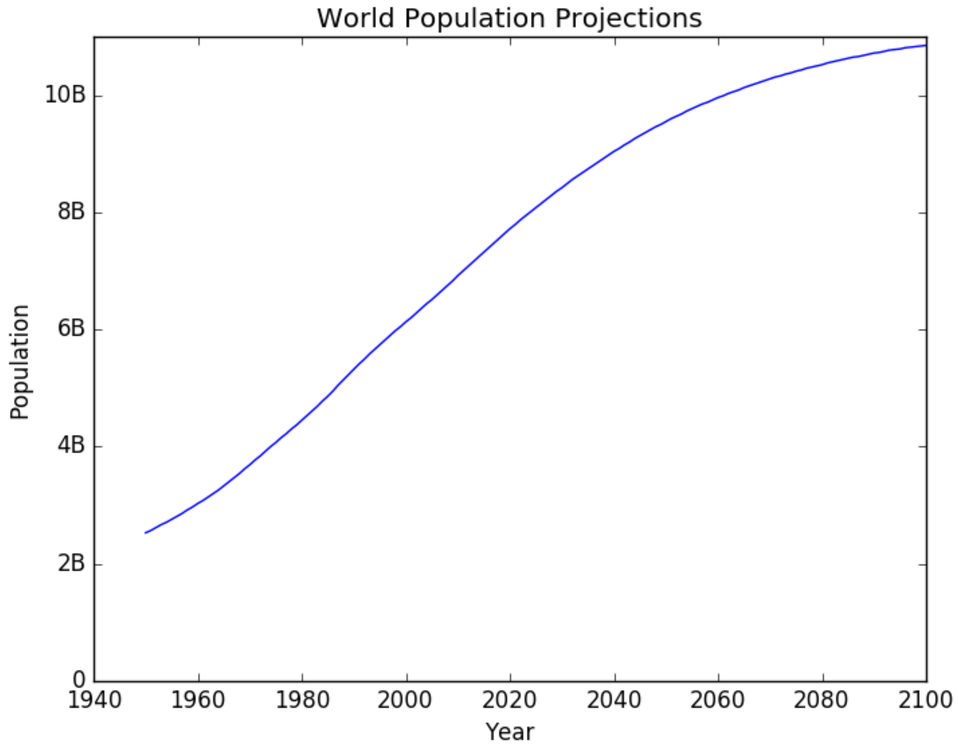
```
import matplotlib.pyplot as plt
year = ... # Implementation left out
population = ... # Implementation left out

plt.plot(year, population)

plt.xlabel('Year')
plt.ylabel('Population')
plt.title('World Population Projections')
plt.yticks([0,2,4,6,8,10])

plt.show()
```



Ticks (2)

my_script.py

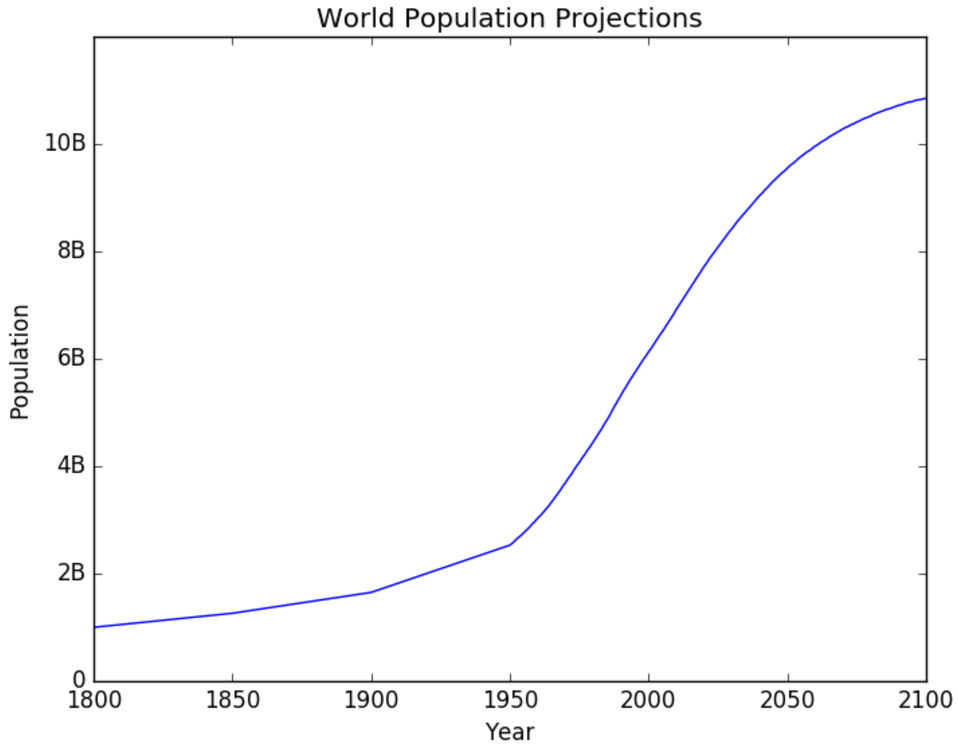
```
import matplotlib.pyplot as plt
year = ... # Implementation left out
population = ... # Implementation left out

plt.plot(year, population)

plt.xlabel('Year')
plt.ylabel('Population')
plt.title('World Population Projections')
plt.yticks([0,2,4,6,8,10],
           ['0','2B','4B','6B','8B','10B']))

plt.show()
```



Add historical data

my_script.py

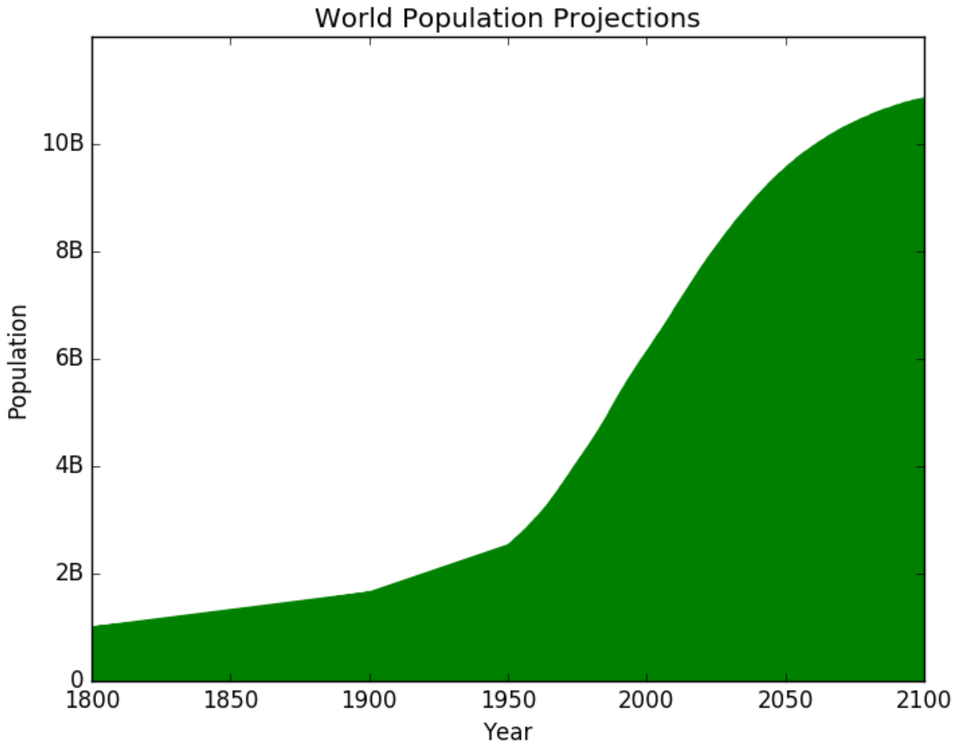
```
import matplotlib.pyplot as plt
year = ... # Implementation left out
population = ... # Implementation left out
population = [1.0,1.262,1.650] + population
year = [1800,1850,1900] + year

plt.plot(year, population)

plt.xlabel('Year')
plt.ylabel('Population')
plt.title('World Population Projections')

plt.yticks([0,2,4,6,8,10],
           ['0','2B','4B','6B','8B','10B'])

plt.show()
```



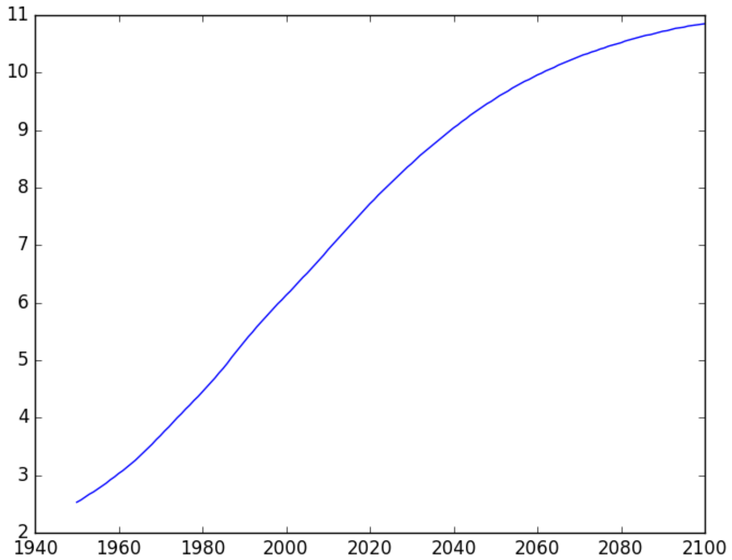
Add historical data

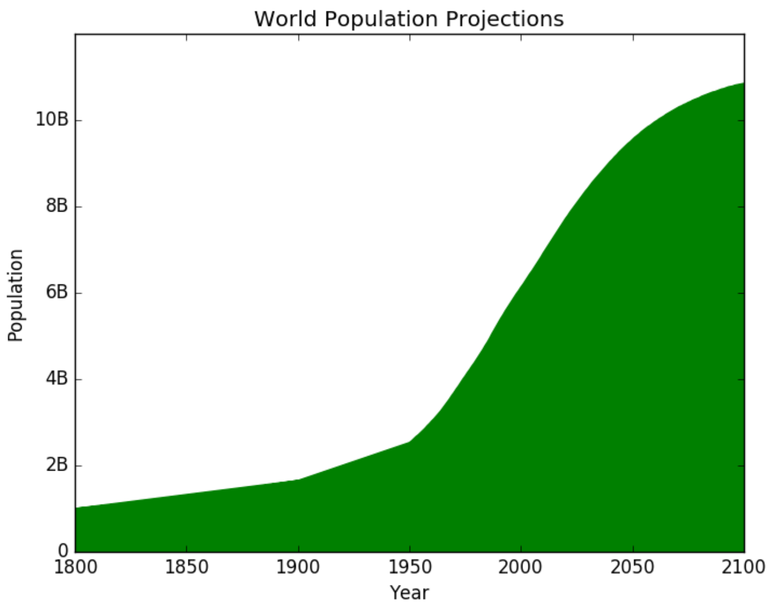
my_script.py

```
import matplotlib.pyplot as plt
year = ... # Implementation left out
population = ... # Implementation left out
population = [1.0,1.262,1.650] + population
year = [1800,1850,1900] + year

plt.fill_between(year,population,0,color='green')

plt.xlabel('Year')
plt.ylabel('Population')
plt.title('World Population Projections')

plt.yticks([0,2,4,6,8,10],
           ['0','2B','4B','6B','8B','10B'])

plt.show()
```



Before vs After





Microsoft

INTRO TO PYTHON FOR DATA SCIENCE

Let’s practice!

Microsoft

INTRO TO PYTHON FOR DATA SCIENCE

Pandas

Overview

- Huge amounts of data are common
- 2D Numpy array?
  - Only one type possible
- Pandas
  - High-level data manipulation
  - `DataFrame`

brics

```
In [1]: brics = ... # declaration left out

In [2]: brics
Out[2]:
```

column labels

	country	population	area	capital
BR	Brazil	200	8515767	Brasilia
RU	Russia	144	17098242	Moscow
IN	India	1252	3287590	New Delhi
CH	China	1357	9596961	Beijing
SA	South Africa	55	1221037	Pretoria

row labels

CSV file

brics.csv

```
,country,population,area,capital
BR,Brazil,200,8515767,Brasilia
RU,Russia,144,17098242,Moscow
IN,India,1252,3287590,New Delhi
CH,China,1357,9596961,Beijing
SA,South Africa,55,1221037,Pretoria
```

CSV file -> DataFrame

```
In [3]: import pandas as pd

In [4]: brics = pd.read_csv("path/to/brics.csv")

In [5]: brics
Out[5]:
  Unnamed: 0        country  population       area    capital
0         BR         Brazil         200    8515767   Brasilia
1         RU         Russia         144   17098242     Moscow
2         IN          India        1252    3287590  New Delhi
3         CH          China        1357    9596961    Beijing
4         SA   South Africa          55    1221037    Pretoria
```

CSV file -> DataFrame

```
In [6]: brics = pd.read_csv("path/to/brics.csv", index_col = 0)

In [7]: brics
Out[7]:
          country  population      area    capital
BR         Brazil         200   8515767   Brasilia
RU         Russia         144  17098242     Moscow
IN          India        1252   3287590  New Delhi
CH          China        1357   9596961    Beijing
SA  South Africa          55   1221037   Pretoria
```

Column access

```
In [8]: brics["country"]
Out[8]:

BR           Brazil
RU           Russia
IN            India
CH            China
SA     South Africa
Name: country, dtype: object
```

```
In [9]: brics.country
Out[9]:

BR           Brazil
RU           Russia
IN            India
CH            China
SA     South Africa
Name: country, dtype: object
```

Add Column

```
In [10]: brics["on_earth"] = [True, True, True, True, True]

In [11]: brics
Out[11]:
          country  population      area    capital on_earth
BR         Brazil         200   8515767   Brasilia     True
RU         Russia         144  17098242     Moscow     True
IN          India        1252   3287590  New Delhi     True
CH          China        1357   9596961    Beijing     True
SA  South Africa          55   1221037   Pretoria     True
```

Add Column (2)

```
In [12]: brics["density"] = brics["population"] / brics["area"] * 1000000

In [13]: brics
Out[13]:
```

	country	population	area	capital	on_earth	density
BR	Brazil	200	8515767	Brasilia	True	23.485847
RU	Russia	144	17098242	Moscow	True	8.421918
IN	India	1252	3287590	New Delhi	True	380.826076
CH	China	1357	9596961	Beijing	True	141.398928
SA	South Africa	55	1221037	Pretoria	True	45.043680

Row access

```
In [14]: brics.loc["BR"]
Out[14]:

country          Brazil
population          200
area            8515767
capital        Brasilia
density        23.48585
on earth           True
Name: BR, dtype: object
```

Element access

```
In [15]: brics.loc["CH","capital"]
Out[15]: Beijing

In [16]: brics["capital"].loc["CH"]
Out[16]: Beijing

In [17]: brics.loc["CH"]["capital"]
Out[17]: Beijing
```

Microsoft

INTRO TO PYTHON FOR DATA SCIENCE

Let’s practice!