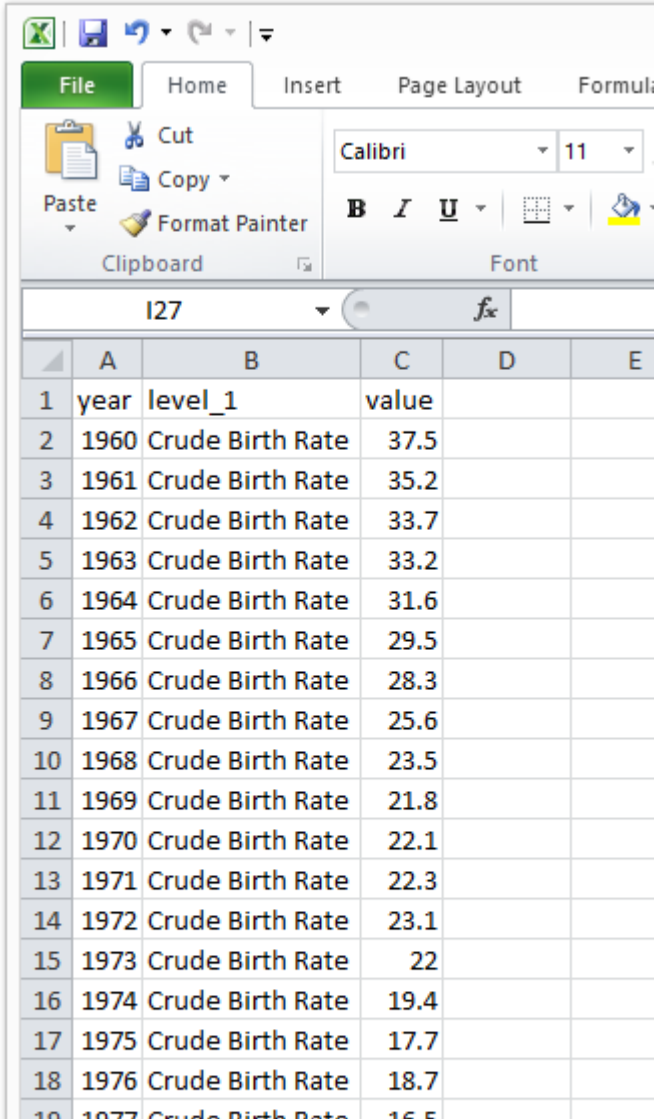
Data Visualization with Python

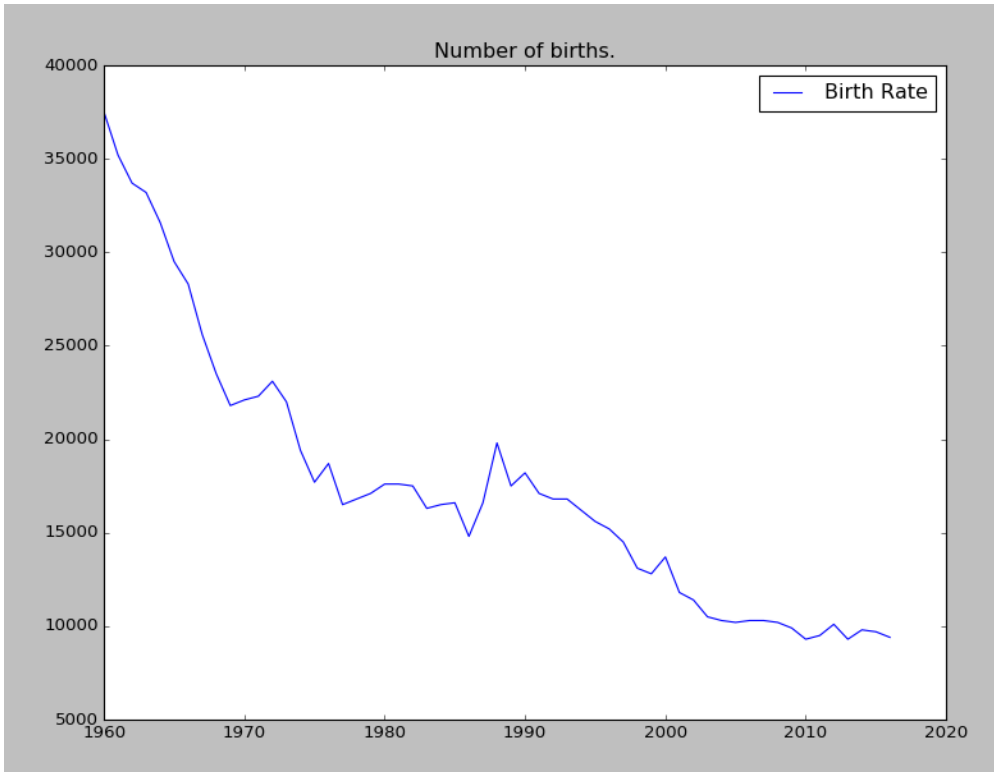
"A picture is worth a thousand words"

Why Visualization?!



	A	B	C	D	E
1	year	level_1	value		
2	1960	Crude Birth Rate	37.5		
3	1961	Crude Birth Rate	35.2		
4	1962	Crude Birth Rate	33.7		
5	1963	Crude Birth Rate	33.2		
6	1964	Crude Birth Rate	31.6		
7	1965	Crude Birth Rate	29.5		
8	1966	Crude Birth Rate	28.3		
9	1967	Crude Birth Rate	25.6		
10	1968	Crude Birth Rate	23.5		
11	1969	Crude Birth Rate	21.8		
12	1970	Crude Birth Rate	22.1		
13	1971	Crude Birth Rate	22.3		
14	1972	Crude Birth Rate	23.1		
15	1973	Crude Birth Rate	22		
16	1974	Crude Birth Rate	19.4		
17	1975	Crude Birth Rate	17.7		
18	1976	Crude Birth Rate	18.7		

vs



Why Visualization?

- Meet the need for speed
- For yourself, a tool to understand your data
- For your audience
  - Addressing data quality
  - Displaying meaningful results
  - Dealing with outliers
- For yourself, for your supervisors, for your boss!

Numpy

- A convenient package
- You can get around with "normal" Python method but Numpy can shorten your code a lot

- For example, if you want to create a list of numbers from 0 to π without Numpy you have to:

```
x = [i/100 for i in range(0,314)]
```

Numpy

```
>>> import numpy as np
>>> x1 = np.arange(0, 3.14, 0.1)
>>> x1
array([ 0. ,  0.1,  0.2,  0.3,  0.4,
  0.5,  0.6,  0.7,  0.8,  0.9,  1. ,
        1.1,  1.2,  1.3,  1.4,  1.5,
  1.6,  1.7,  1.8,  1.9,  2. ,  2.1,
        2.2,  2.3,  2.4,  2.5,  2.6,
  2.7,  2.8,  2.9,  3. ,  3.1])
>>> len(x1)
32
```

Be Careful!!!
This is not “arrange”
but “arange”

Another Way

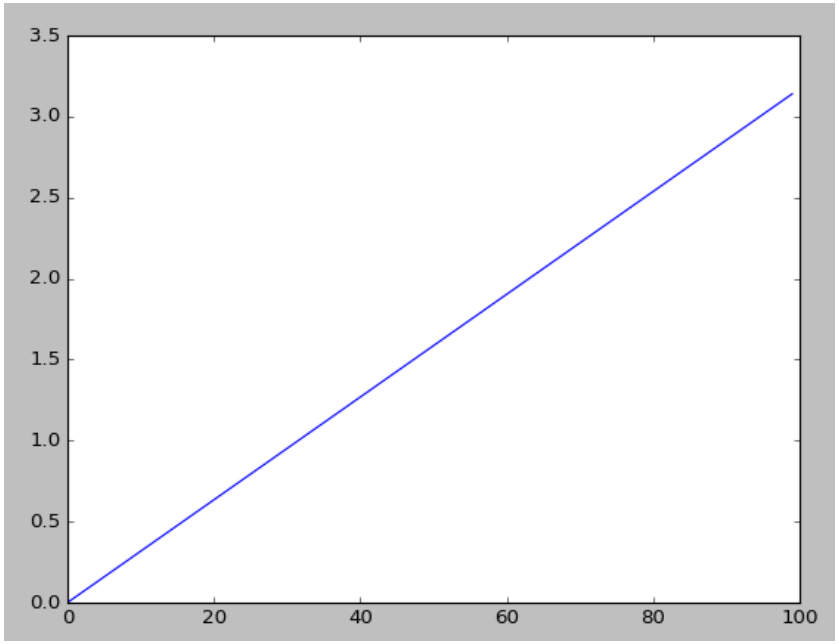
Be Careful!!!
This is not
“linespace” but
“linspace”

```
>>> x2 = np.linspace(0,3.14,100)
>>> x2
array([ 0.        ,  0.03171717,  0.063
43434,  0.09515152,  0.12686869,
        0.15858586,  0.19030303,  0.222
0202 ,  0.25373737,  0.28545455,
        0.31717172,  0.34888889,  0.380
                .
                .
                .
        2.6959596 ,  2.7276677,  2.759
39394,  2.79111111,  2.82282828,
        2.85454545,  2.88626263,  2.917
9798 ,  2.94969697,  2.98141414,
        3.01313131,  3.04484848,  3.076
56566,  3.10828283,  3.14      ])
>>> len(x2)
100
```

Using matplotlib

You can Simply

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

x1 = np.linspace(0,3.14,100)
plt.plot(x1)

plt.show()
```



Plotting cos(x)

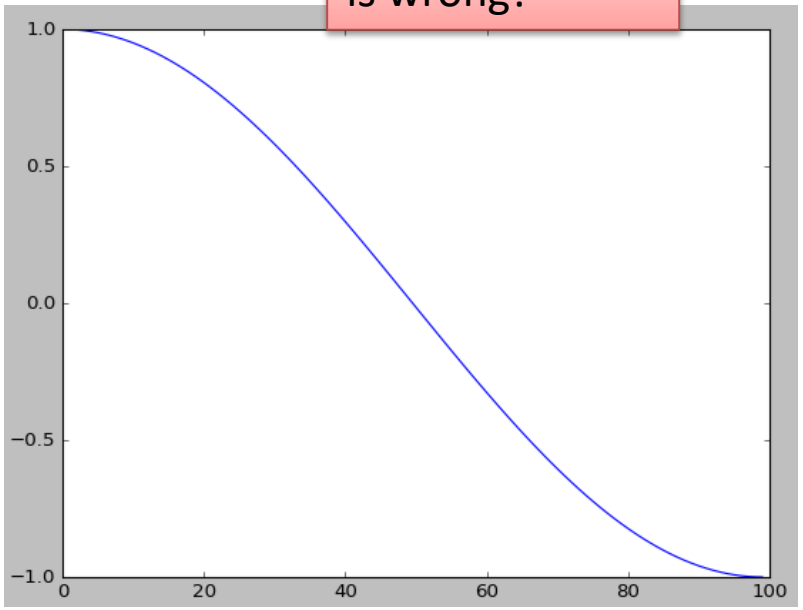
- Without Numpy, you have to

```
y = [cos(i) for i in x1]
```

- With Numpy, you simply do

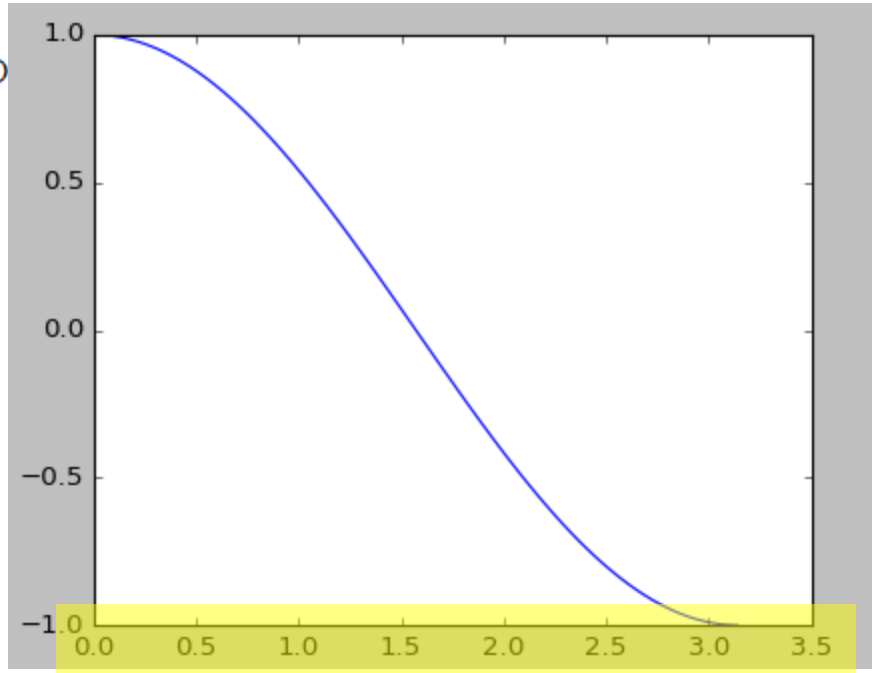
```
import numpy as np
import matplotlib.pyplot as plt

x1 = np.linspace(0,3.14,100)
y1 = np.cos(x1)

plt.plot(y1)

plt.show()
```

But the x-range is wrong!



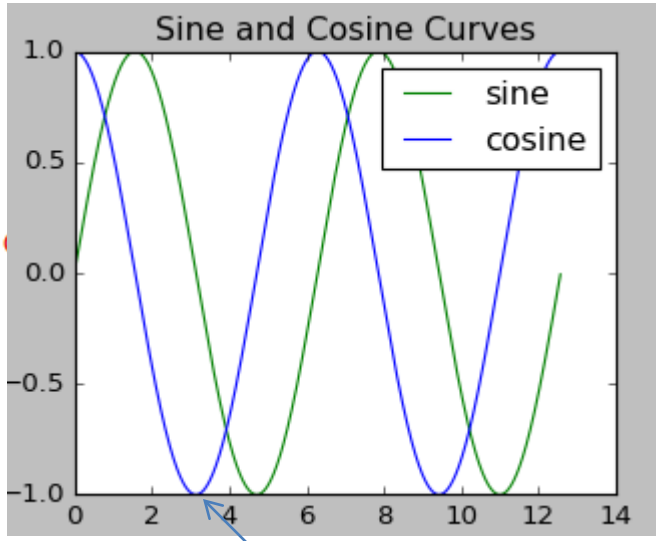
- If you only specify one list (or any sequence), each item will be plotted against just an integer

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

x1 = np.linspace(0,3.14,100)
y1 = np.cos(x1)

plt.plot(x1,y1)

plt.show()
```



More Curves (Lab 00)

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

# Create x, evenly spaced between 0 t
x = np.linspace(0, 3.14 * 4 , 1000)

y1 = np.sin(x)
y2 = np.cos(x)

# Plot the sin and cos functions
plt.plot(x , y1, "-g", label="sine")
plt.plot(x , y2, "-b", label="cosine")

# The legend should be in the top right corner
plt.legend(loc="upper right")
plt.title('Sine and Cosine Curves')

plt.show()
```



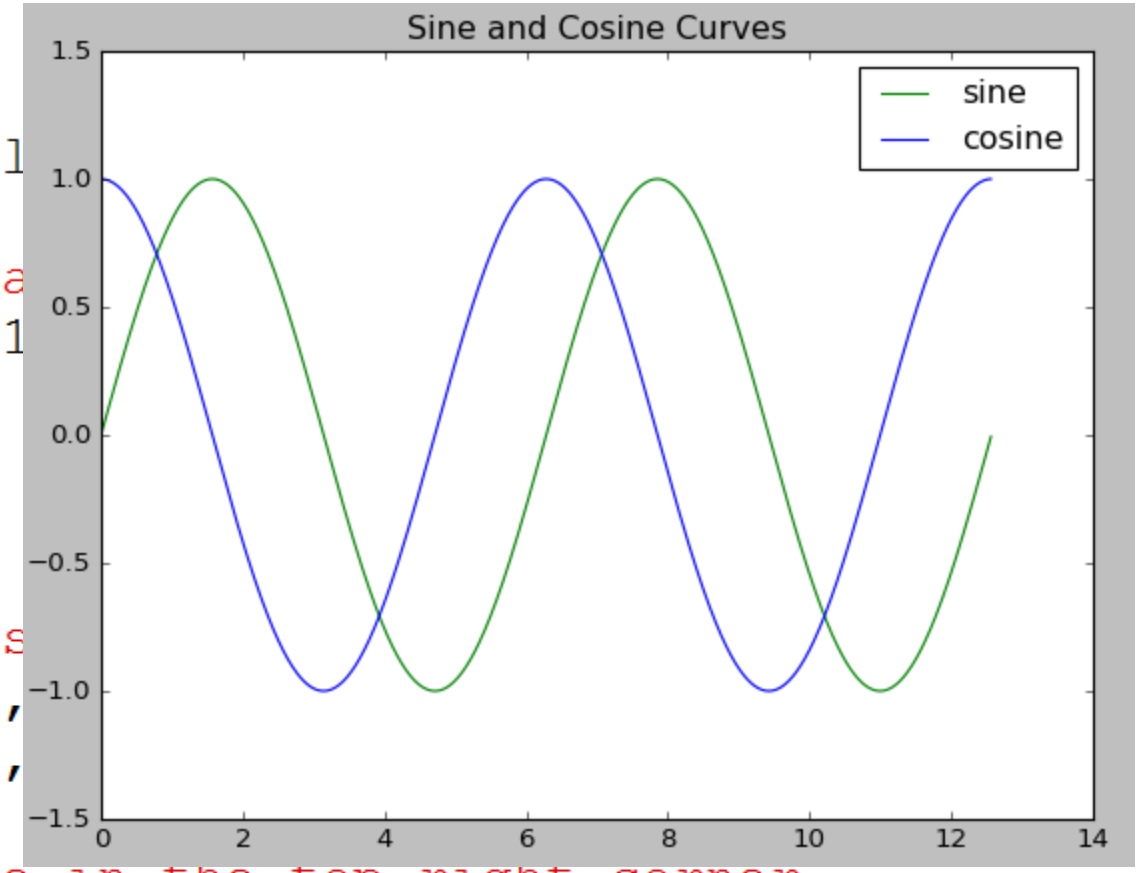
Green color and label

Not very nice looking when the curve is touching the boundary

Position of the legend

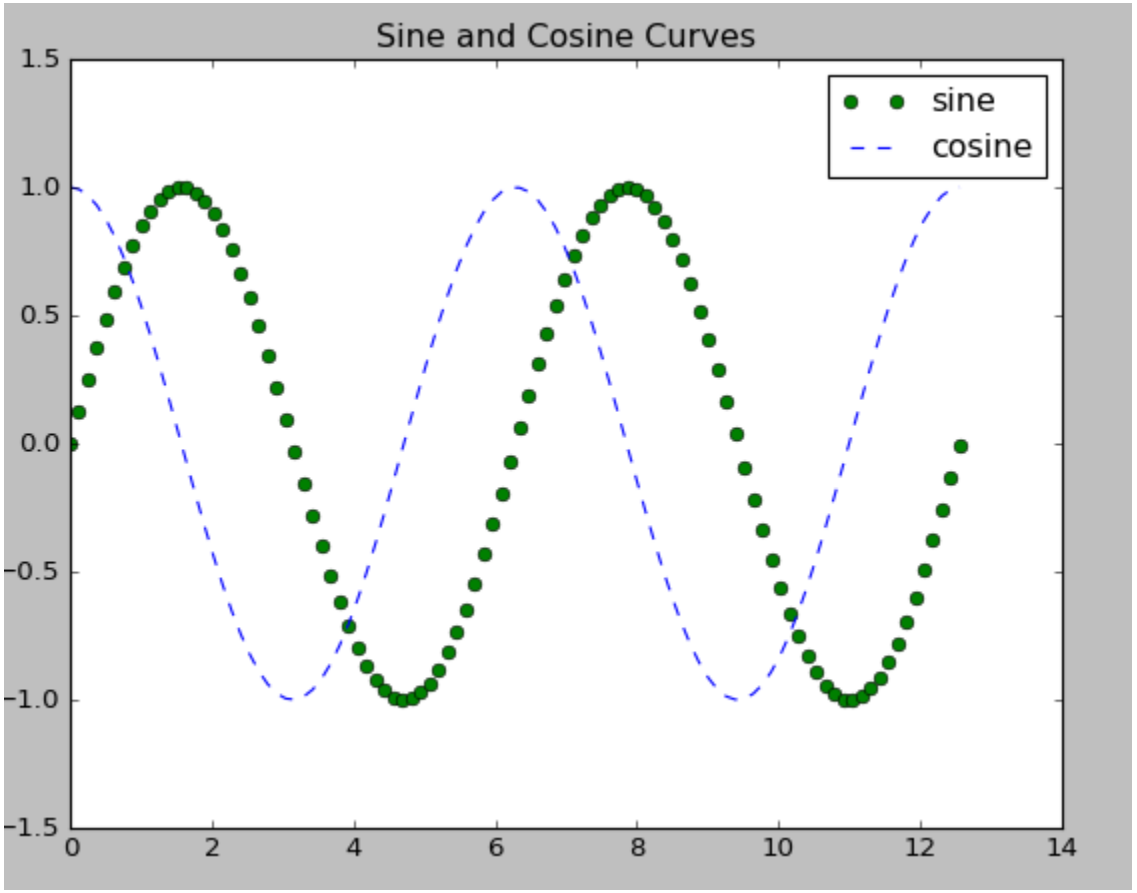
Title

```
import numpy as np
import matplotlib.pypl

# Create x, evenly spa
x = np.linspace(0, 3.1

y1 = np.sin(x)
y2 = np.cos(x)

# Plot the sin and cos
plt.plot(x , y1, "-g",
plt.plot(x , y2, "-b",

# The legend should be in the top right corner
plt.legend(loc="upper right")
plt.title('Sine and Cosine Curves')

# Limit the y axis to -1.5 to 1.5
plt.ylim(-1.5, 1.5)

plt.show()
```



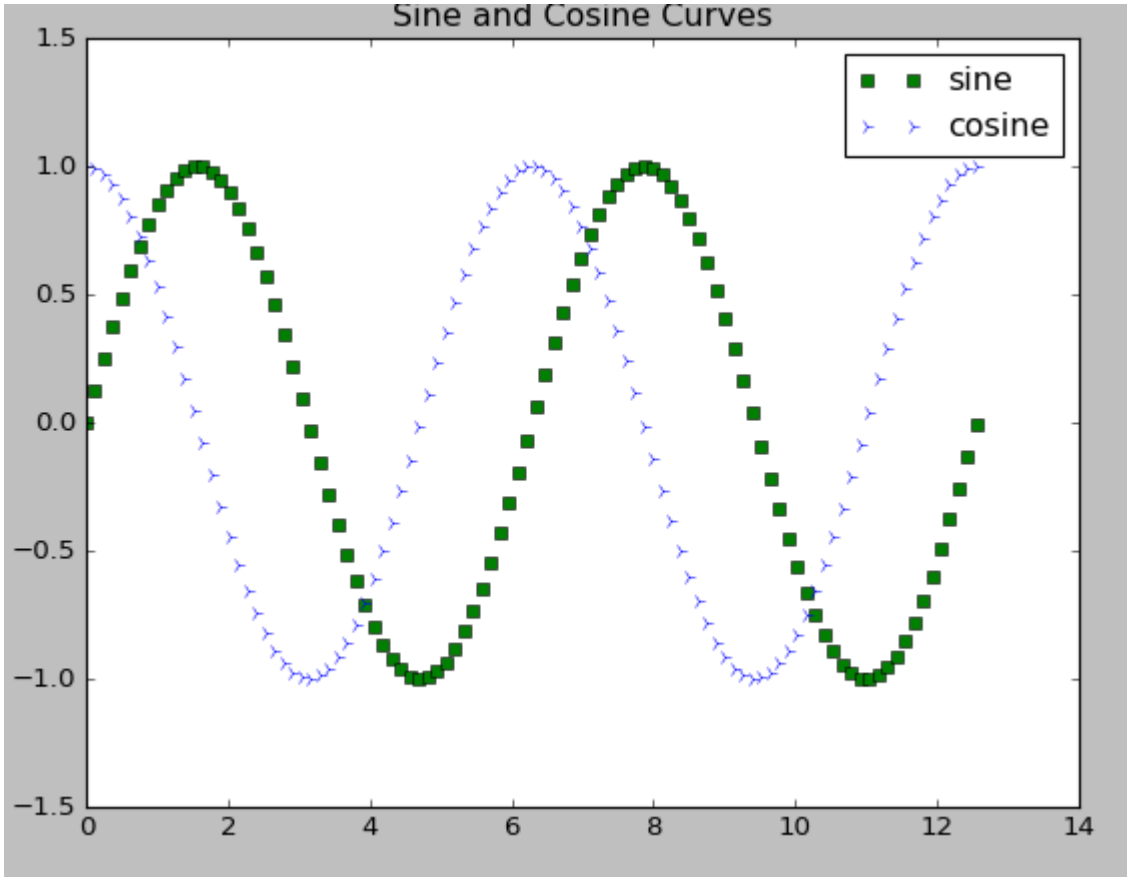
Set the vertical limits to be -1.5 to 1.5

```
# Plot the sin and cos functions
plt.plot(x , y1, "-g", label="sine")
plt.plot(x , y2, "-b", label="cosine")

plt.plot(x , y1, "og", label="sine")
plt.plot(x , y2, "--b", label="cosine")
```



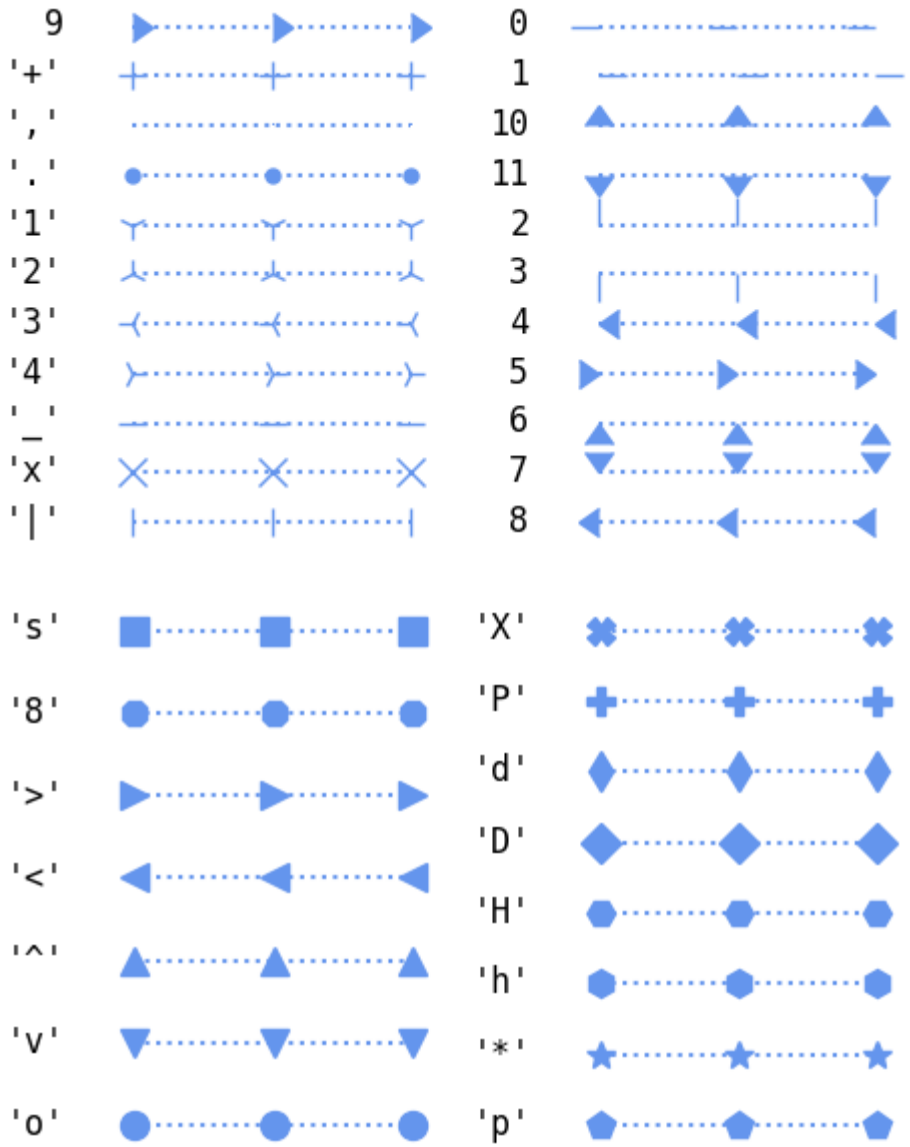
Changing Markers/Line Style

```
# Plot the sin and cos functions
plt.plot(x , y1, "sg", label="sine")
plt.plot(x , y2, "4b", label="cosine")
```



Try it out

--	dashed
:	dotted
-.	dash dotted
-	solid
o	circle
<	triangle_left
+	plus

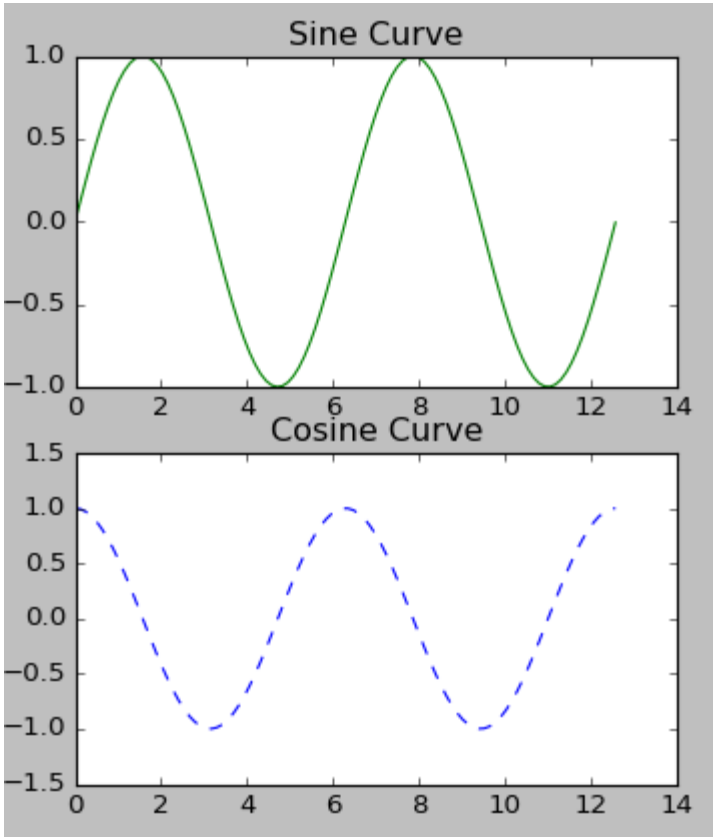


Details of ALL markers:
https://matplotlib.org/api/markers_api.html

Multiple Figures

```
# Plot the sin and cos functions
plt.subplot(211)
plt.plot(x , y1, "-g")
plt.title('Sine Curve')
plt.subplot(212)
plt.plot(x , y2, "--b")
plt.title('Cosine Curve')

# Limit the y axis to -1.5 to 1.5
plt.ylim(-1.5, 1.5)

plt.show()
```

Only affect the most recent graph



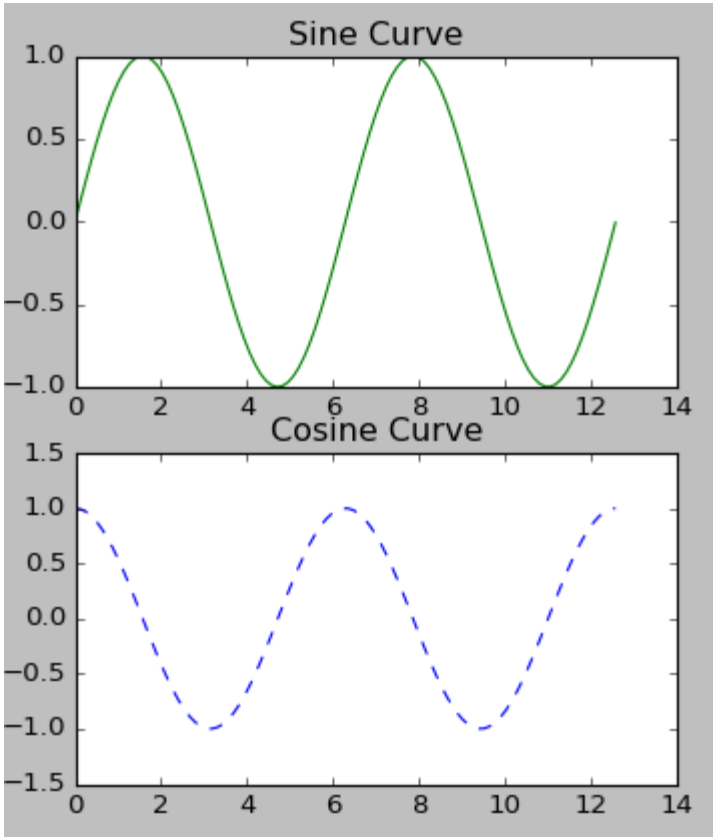
Multiple Figures

```
# Plot the sin and cos functions
plt.subplot(211)
```

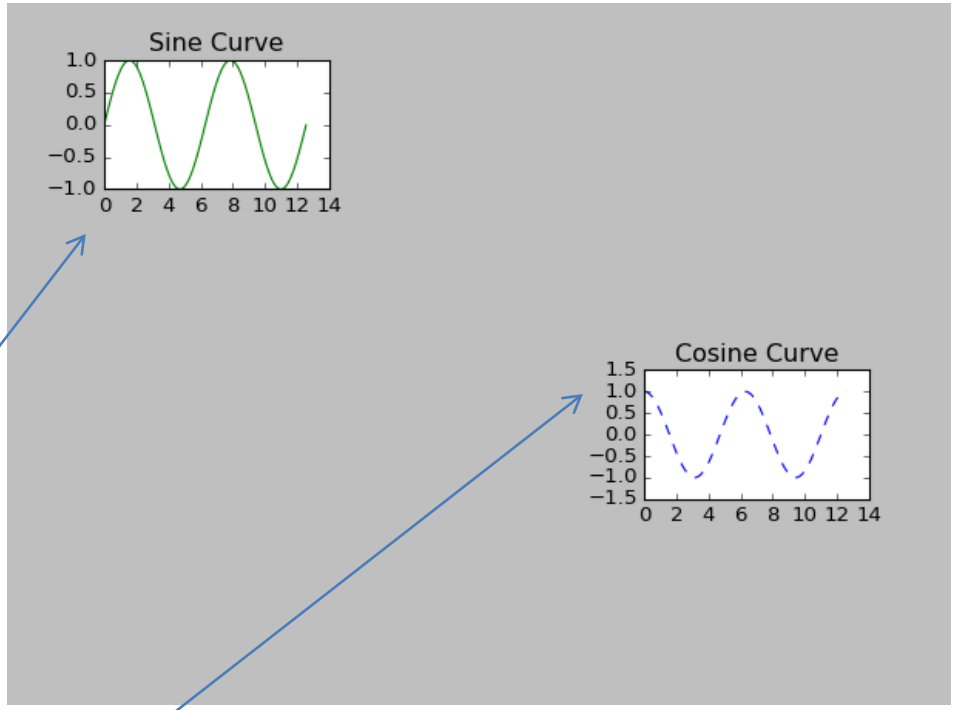
How many rows r are there

How many columns c are there

After dividing the figure into r x c places, which one do you want to place the current plotting



```
# Plot the sin and cos functions
plt.subplot(431)
plt.plot(x , y1,  "-g")
plt.title('Sine Curve')
plt.subplot(439)
plt.plot(x , y2,  "--b")
plt.title('Cosine Curve')
```



4 rows
3 columns

Position 1

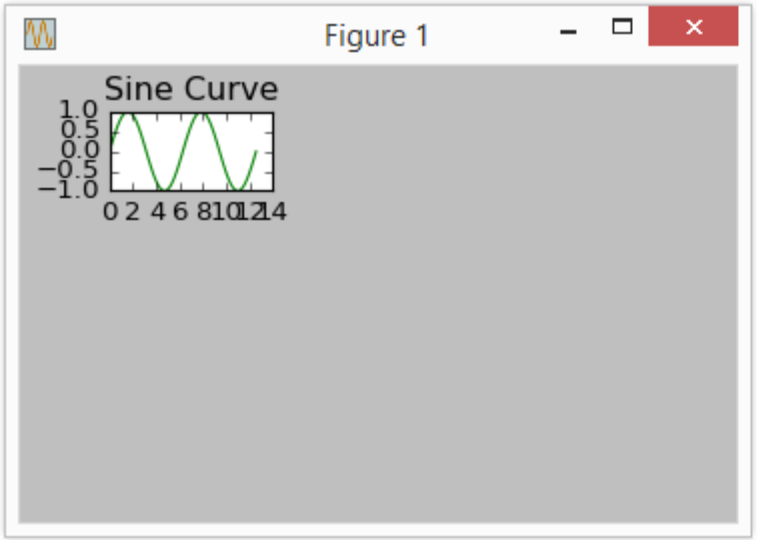
Position 9

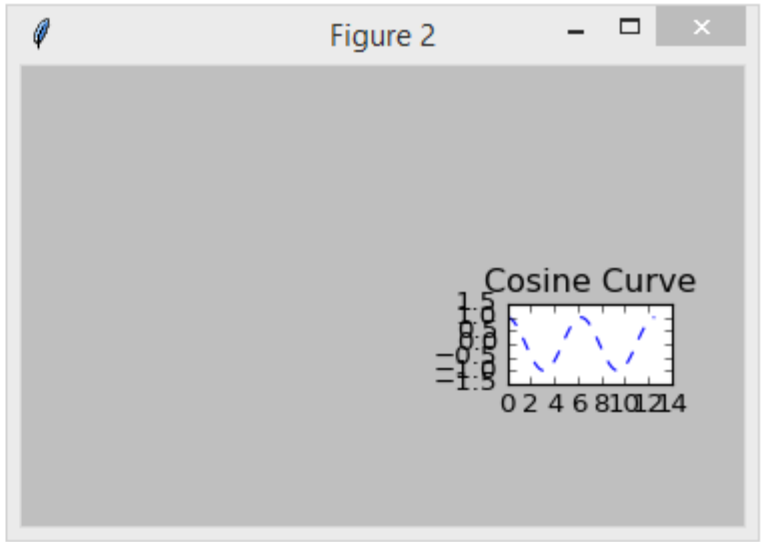
If you really have a lot of figures

```
# Plot the sin and cos functions
plt.figure(1)
plt.subplot(431)
plt.plot(x , y1, "-g")
plt.title('Sine Curve')

plt.figure(2)
plt.subplot(439)
plt.plot(x , y2, "--b")
plt.title('Cosine Curve')
```

The following plotting will be in Figure 1

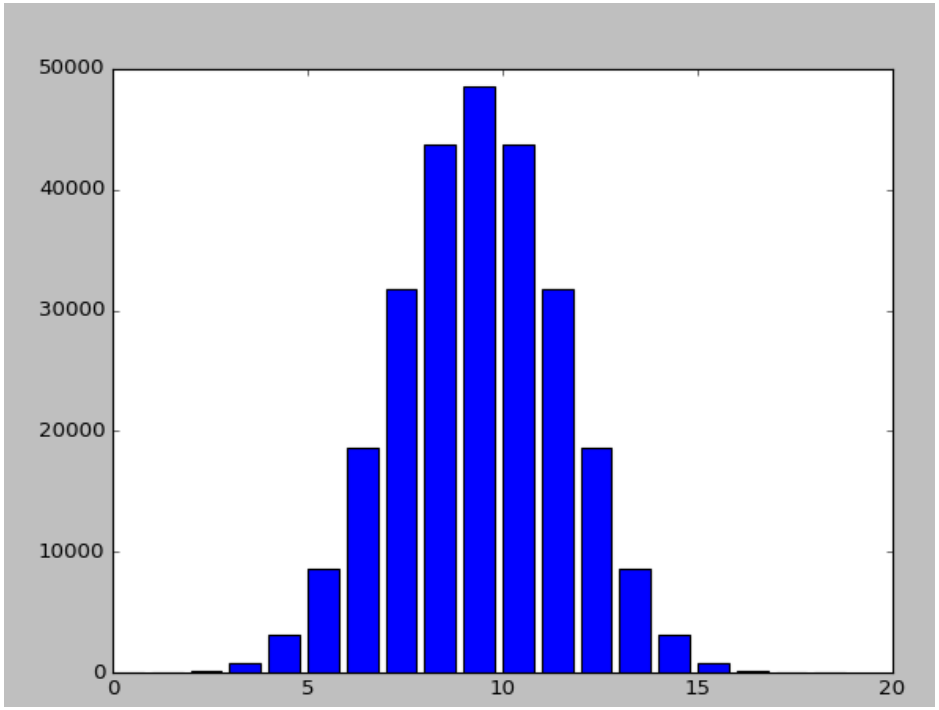
The following plotting will be in Figure 2





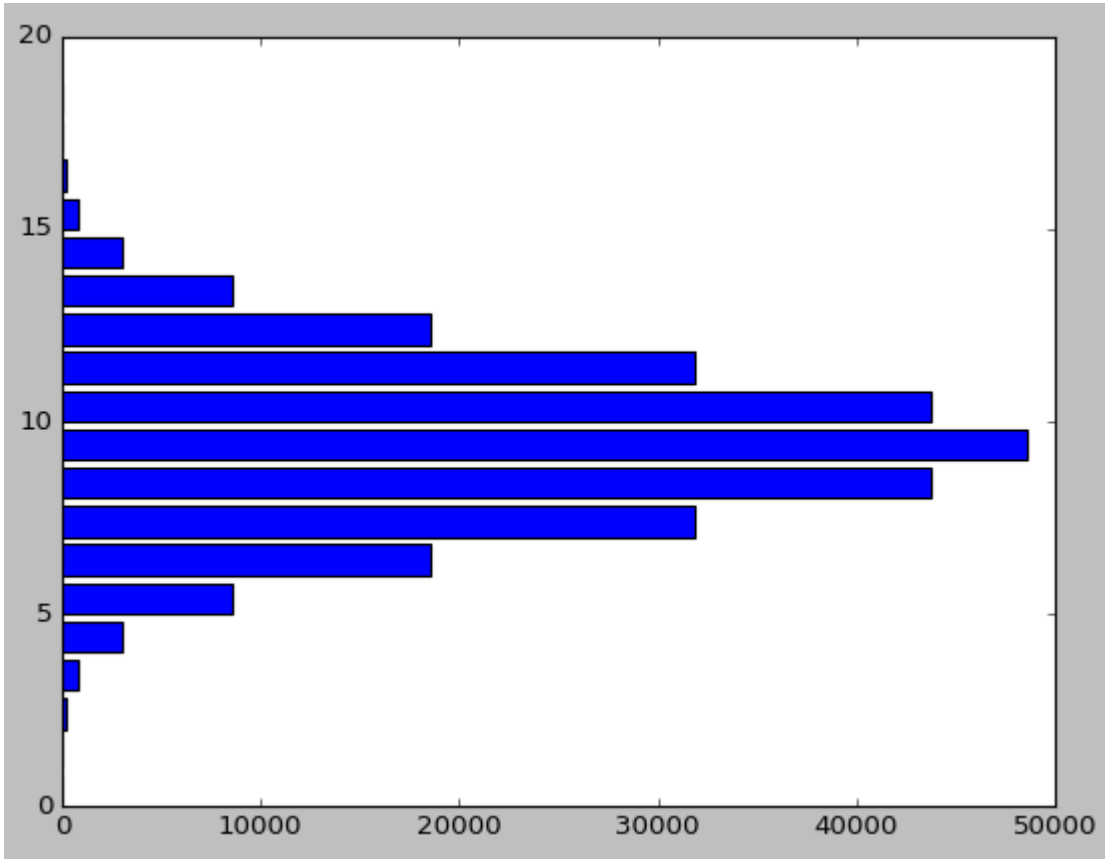
Bar Chart

```
from Lab_02_Part_A import nChooseK

N = 18
x = [i for i in range(0,N+1)]
y = [nChooseK(N,i) for i in range(0,N+1)]

plt.bar(x,y)
plt.show()
```



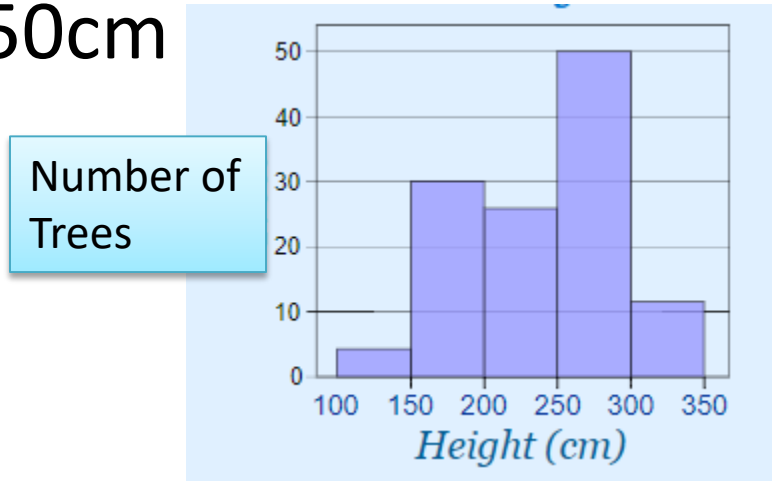
Bar Chart

```
N = 18

x = [i for i in range(0,N+1)]
y = [nChooseK(N,i) for i in range(0,N+1)]

plt.barh(x,y)

plt.show()
```

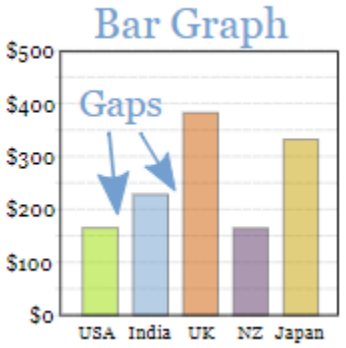


Histogram

- Similar to bar chart, but a histogram groups numbers into ranges
- E.g. Given data: heights of 30 trees from 150cm to 350cm



- Usually bar charts have gaps
  - But histograms have no gap



Let's Roll

- When you roll three dices and sum the number, which number has the highest chance?
- Application
  - E.g. in Monopoly, if you start from "GO!", which place(s) has/have the highest probability to be landed on?

Rolling 3 dices

3 = 1 + 1 + 1
4 = 1 + 1 + 2
5 = 1 + 1 + 3 = 2 + 2 + 1
6 = 1 + 1 + 4 = 1 + 2 + 3 = 2 + 2 + 2
7 = 1 + 1 + 5 = 2 + 2 + 3 = 3 + 3 + 1 = 1 + 2 + 4
8 = 1 + 1 + 6 = 2 + 3 + 3 = 4 + 3 + 1 = 1 + 2 + 5 = 2 + 2 + 4
9 = 6 + 2 + 1 = 4 + 3 + 2 = 3 + 3 + 3 = 2 + 2 + 5 = 1 + 3 + 5 = 1 + 4 + 4
10 = 6 + 3 + 1 = 6 + 2 + 2 = 5 + 3 + 2 = 4 + 4 + 2 = 4 + 3 + 3 = 1 + 4 + 5
11 = 6 + 4 + 1 = 1 + 5 + 5 = 5 + 4 + 2 = 3 + 3 + 5 = 4 + 3 + 4 = 6 + 3 + 2
12 = 6 + 5 + 1 = 4 + 3 + 5 = 4 + 4 + 4 = 5 + 2 + 5 = 6 + 4 + 2 = 6 + 3 + 3
13 = 6 + 6 + 1 = 5 + 4 + 4 = 3 + 4 + 6 = 6 + 5 + 2 = 5 + 5 + 3
14 = 6 + 6 + 2 = 5 + 5 + 4 = 4 + 4 + 6 = 6 + 5 + 3
15 = 6 + 6 + 3 = 6 + 5 + 4 = 5 + 5 + 5
16 = 6 + 6 + 4 = 5 + 5 + 6
17 = 6 + 6 + 5
18 = 6 + 6 + 6

Study Monopoly

- Rolling three dices, what is the range?
  - 3 to 18
  - Totally 16 different combinations
- All combinations have the same probability?
  - NO!
- Then what are the probabilities?
  - We can do calculations
  - But I forgot all my math already!?!?!?

Rolling 3 dices

Probability of a sum of 3: 1/216 = 0.5%
Probability of a sum of 4: 3/216 = 1.4%
Probability of a sum of 5: 6/216 = 2.8%
Probability of a sum of 6: 10/216 = 4.6%
Probability of a sum of 7: 15/216 = 7.0%
Probability of a sum of 8: 21/216 = 9.7%
Probability of a sum of 9: 25/216 = 11.6%
Probability of a sum of 10: 27/216 = 12.5%
Probability of a sum of 11: 27/216 = 12.5%
Probability of a sum of 12: 25/216 = 11.6%
Probability of a sum of 13: 21/216 = 9.7%
Probability of a sum of 14: 15/216 = 7.0%
Probability of a sum of 15: 10/216 = 4.6%
Probability of a sum of 16: 6/216 = 2.8%
Probability of a sum of 17: 3/216 = 1.4%
Probability of a sum of 18: 1/216 = 0.5%

Let's run an experiment

```
N = 100000

def roll_dice():
    return random.randint(1,6)

dice_stat = []
for i in range(N):
    dice_stat.append(roll_dice()+roll_dice()+roll_dice())
```

why not
`roll_dice()*3`?

- `dics_stat` will contains 100000 numbers of the sum of the dices
- Let's say, how many "18" are there in these 100000 numbers?

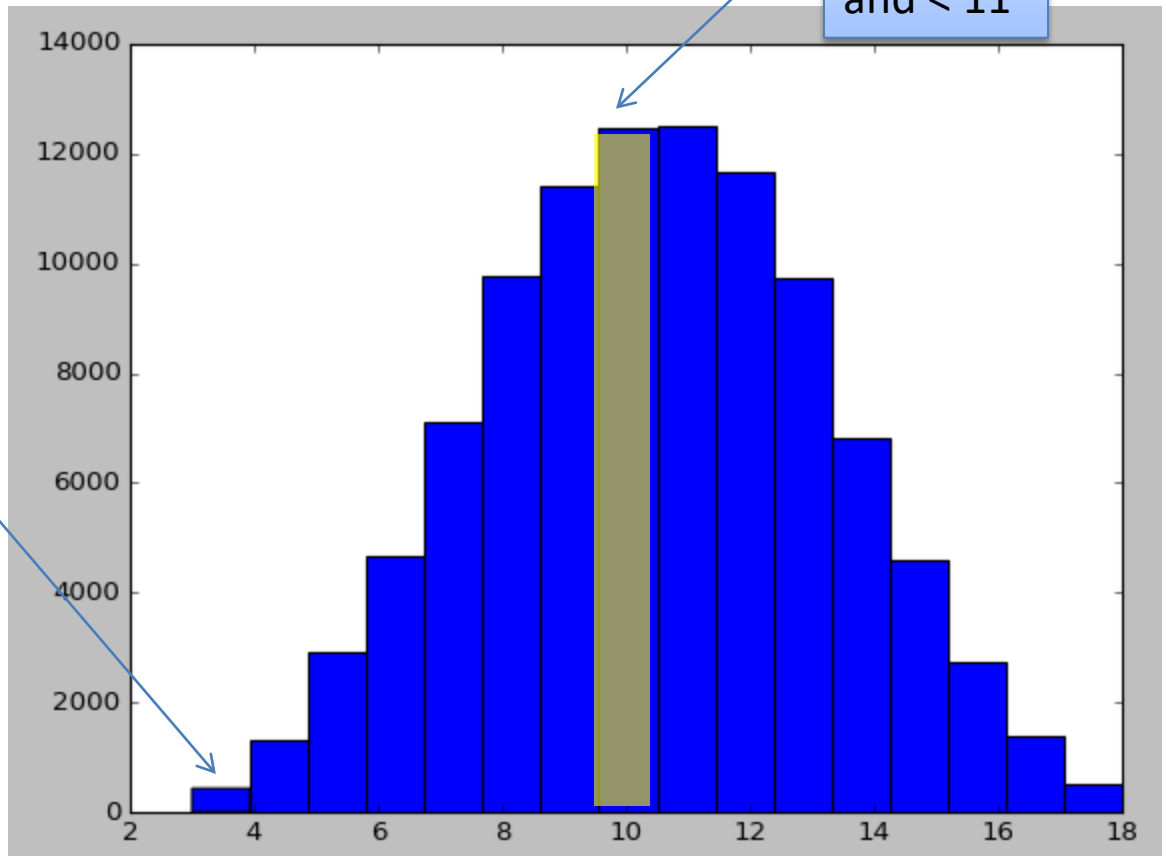
Histogram

```
plt.hist(dice_stat,16)
plt.show()
```

Number of bins

Number of times of >= 10 and < 11

Number of times of >= 3 and < 4



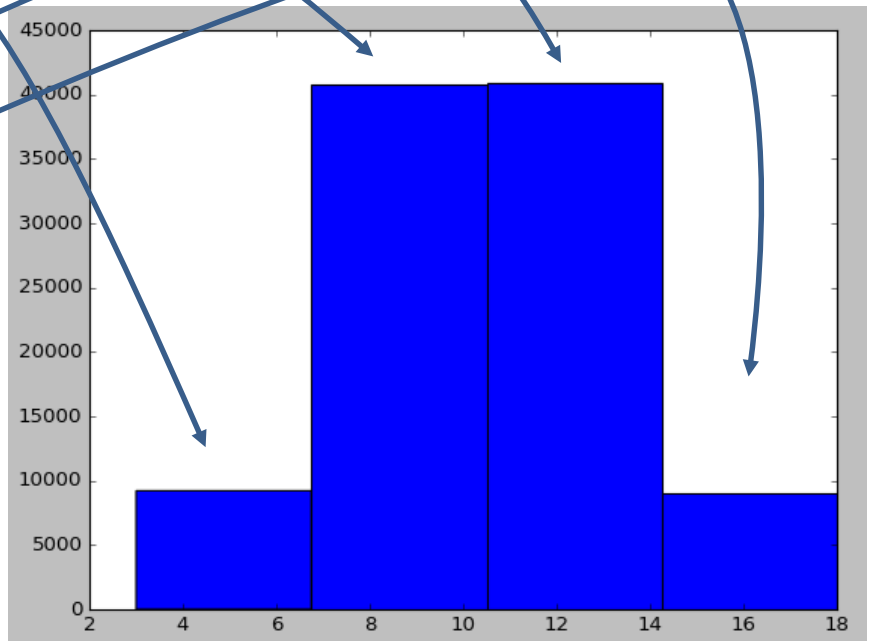
Histogram

- Usually, histograms won't have a bin for every single number x
  - 3 <= x < 4
- For example, if the data is the salary, every bin will have a larger range
  - 1000 <= x < 2000
  - 2000 <= x < 3000
  - etc.

Histogram

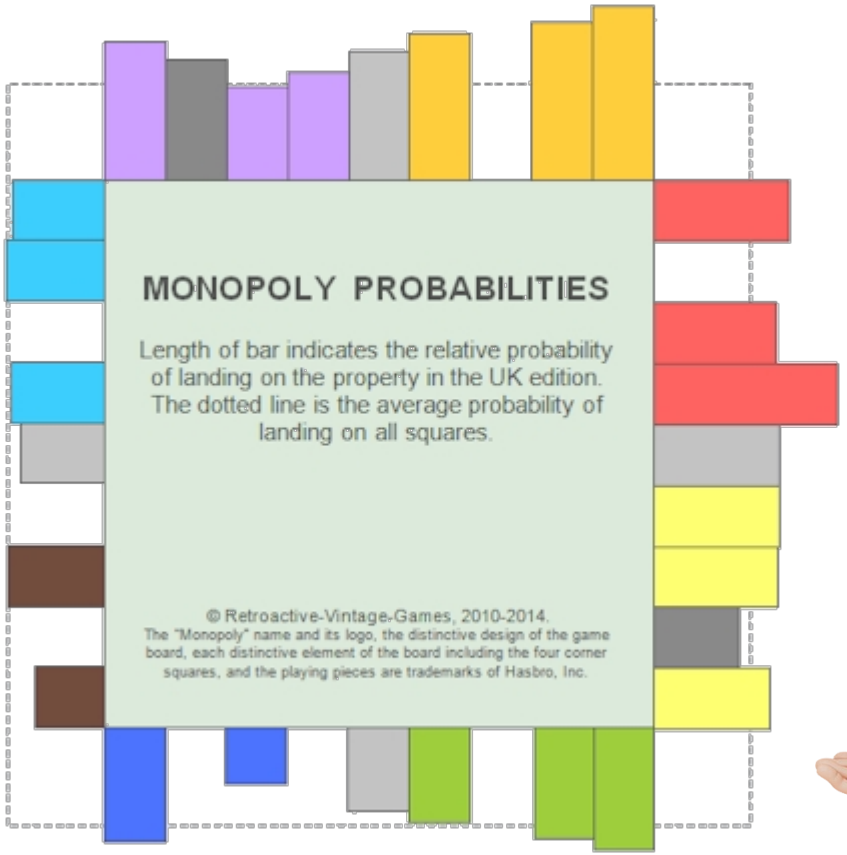
- Back to the dice example, say we want the ranges:
  - 3 <= x < 7
  - 7< = x < 11
  - 11<= x < 15
  - 15 <= x < 19
- Then we need 4 bins

```
plt.hist(dice_stat,4)
plt.show()
```



Application: Monopoly

- Is every place has a equal chance to be landed on?
  - NO!

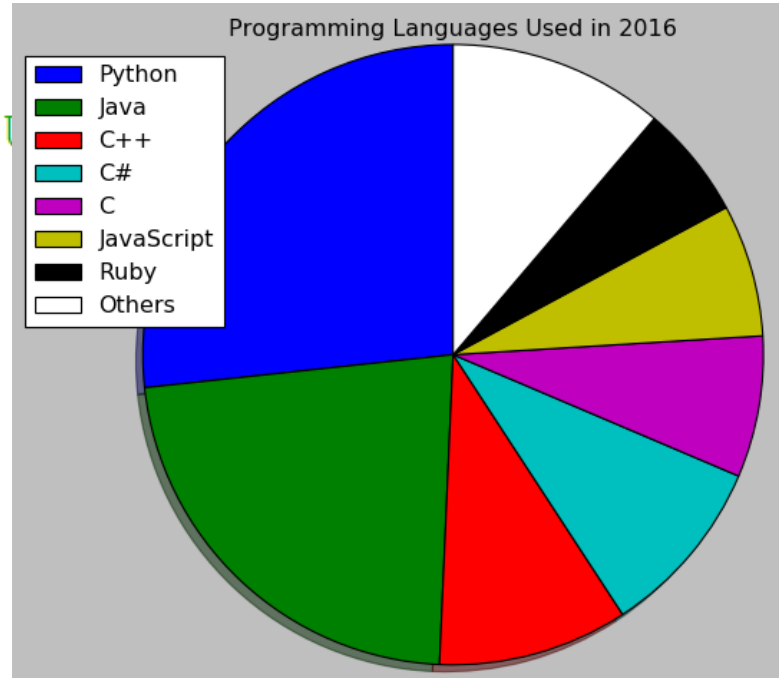


Pie Chart

```
,'Ruby','Others']
```

```
import matplotlib.pyplot as plt

labels = ['Python', 'Java','C++', 'C#', 'C','JavaScript','
sizes = [26.7,22.6,9.9,9.4,7.37,6.9,5.9,11.23]

plt.pie(sizes,shadow=True, startangle=90)
plt.legend(labels, loc="best")

plt.axis('equal')
plt.title('Programming Languages
plt.tight_layout()
plt.show()
```

Otherwise, becomes “oval” chart



Saving a Graph

- If you feel like your graph is cool and want to save it by
  - `fig.savefig('plot.png'), or`
  - `fig.savefig('plot.pdf'), or`
  - `Any format that is supported by matplotlib`

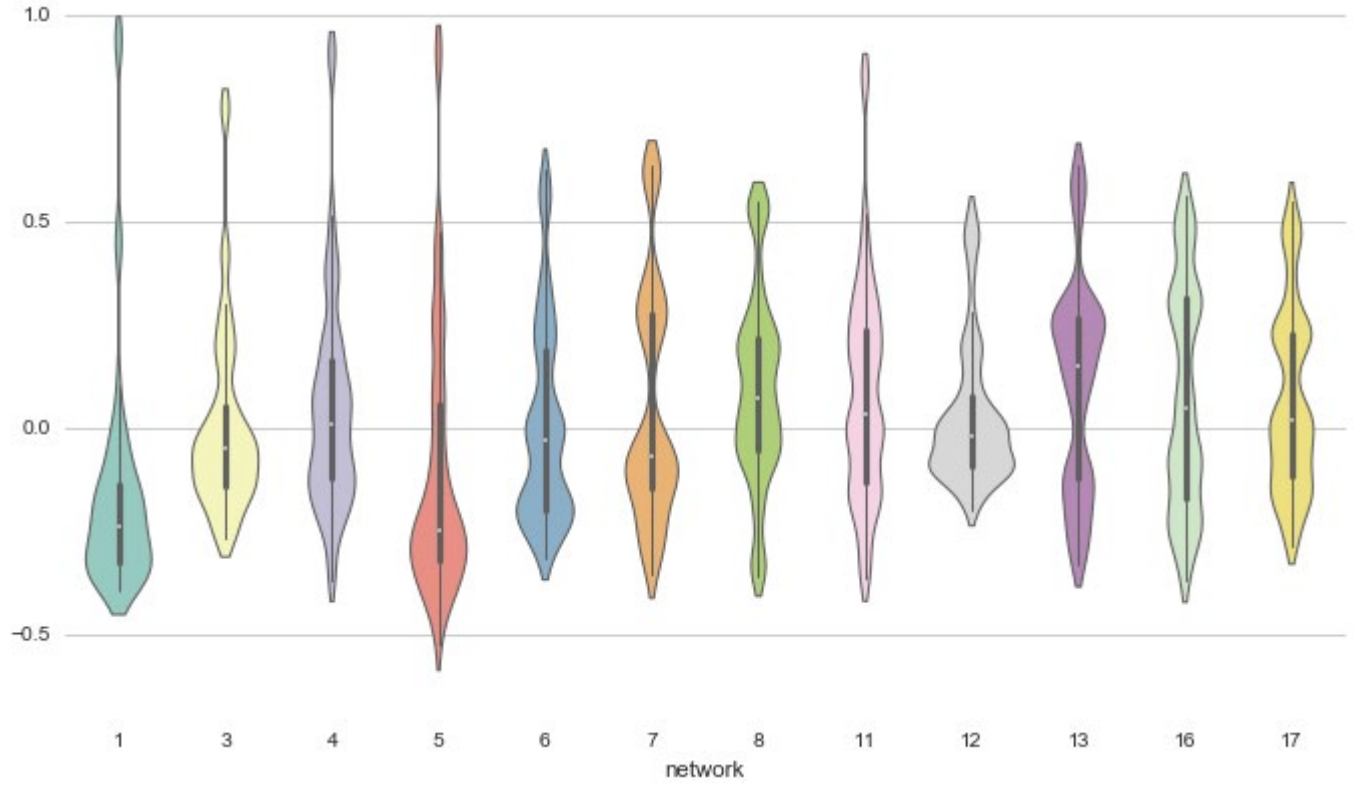
Supported Graphic Format

- You can check the supported file format by
  `plt.gcf().canvas.get_supported_filetypes()`

```
{'ps': 'Postscript', 'eps': 'Encapsulated
Postscript', 'pdf': 'Portable Document
Format', 'pgf': 'PGF code for LaTeX', 'png':
'Portable Network Graphics', 'raw': 'Raw RGBA
bitmap', 'rgba': 'Raw RGBA bitmap', 'svg':
'Scalable Vector Graphics', 'svgz': 'Scalable
Vector Graphics', 'jpg': 'Joint Photographic
Experts Group', 'jpeg': 'Joint Photographic
Experts Group', 'tif': 'Tagged Image File
Format', 'tiff': 'Tagged Image File Format'}
```

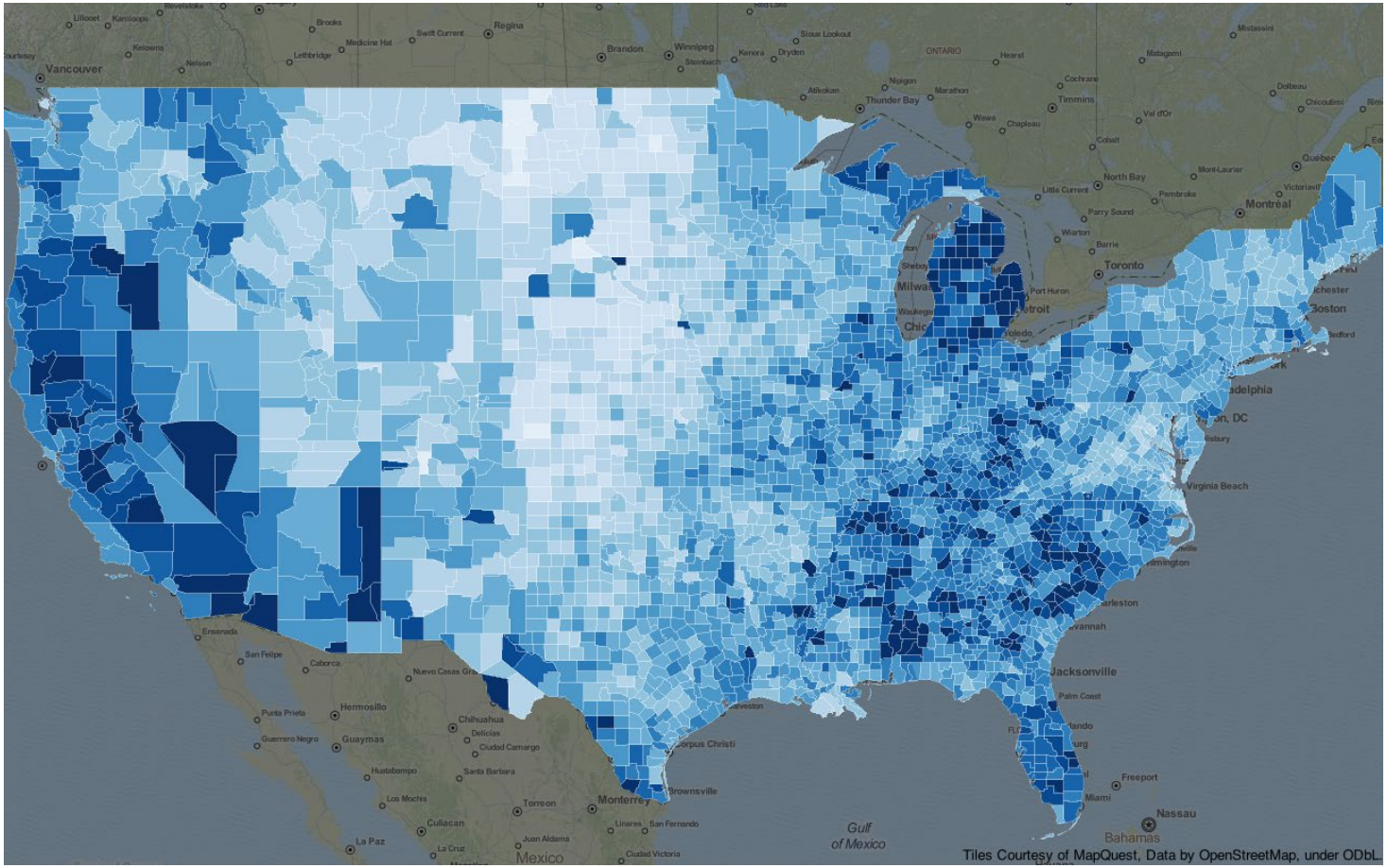
Other Visualization Packages

- Seaborn



Other Visualization Packages

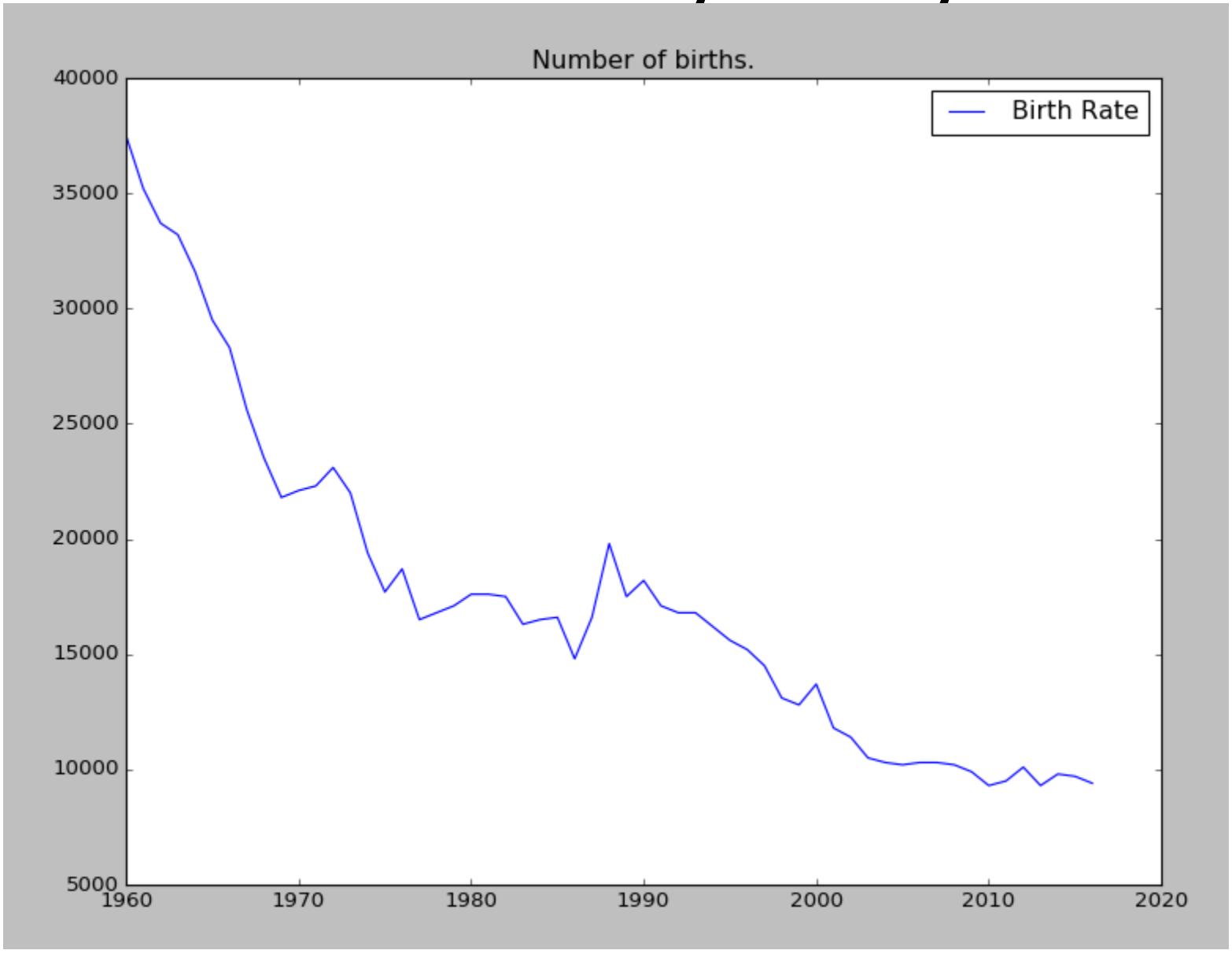
- geoplotlib



Some Tips on GOOD Visualization

- Take a step back and ask yourself, “What is my point?”

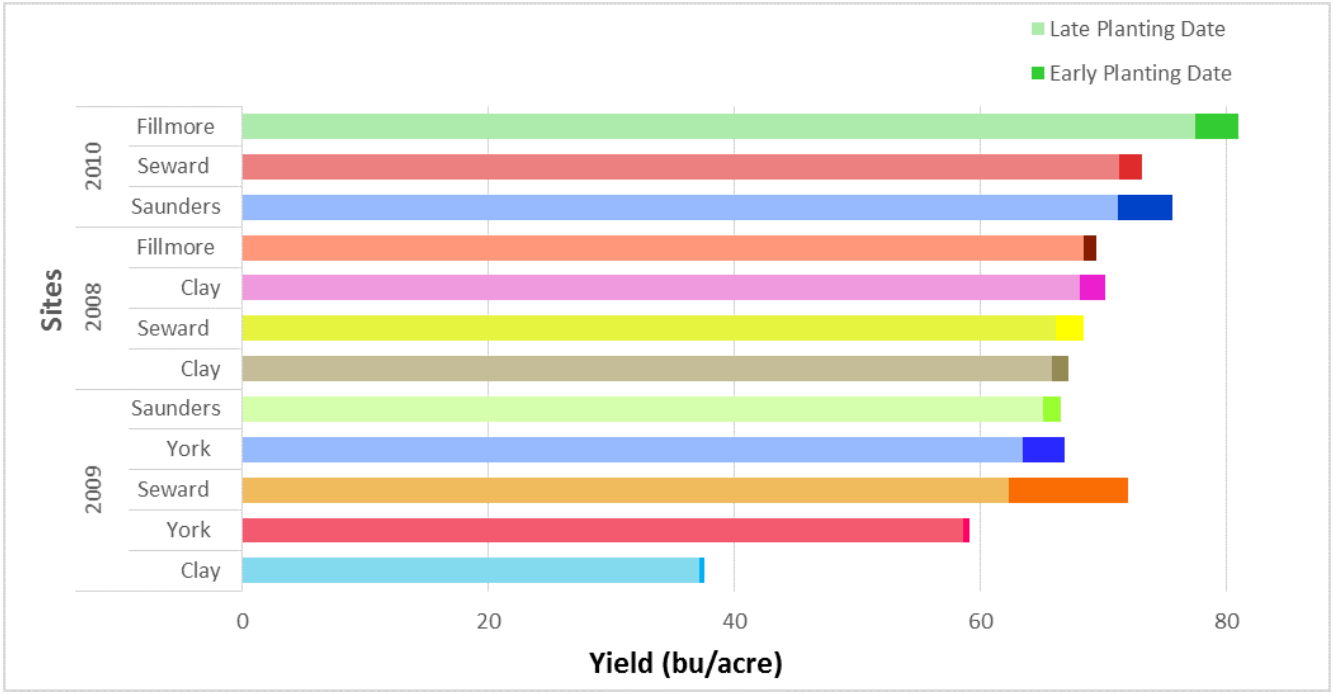
Now You Know Why “Baby Bonus”



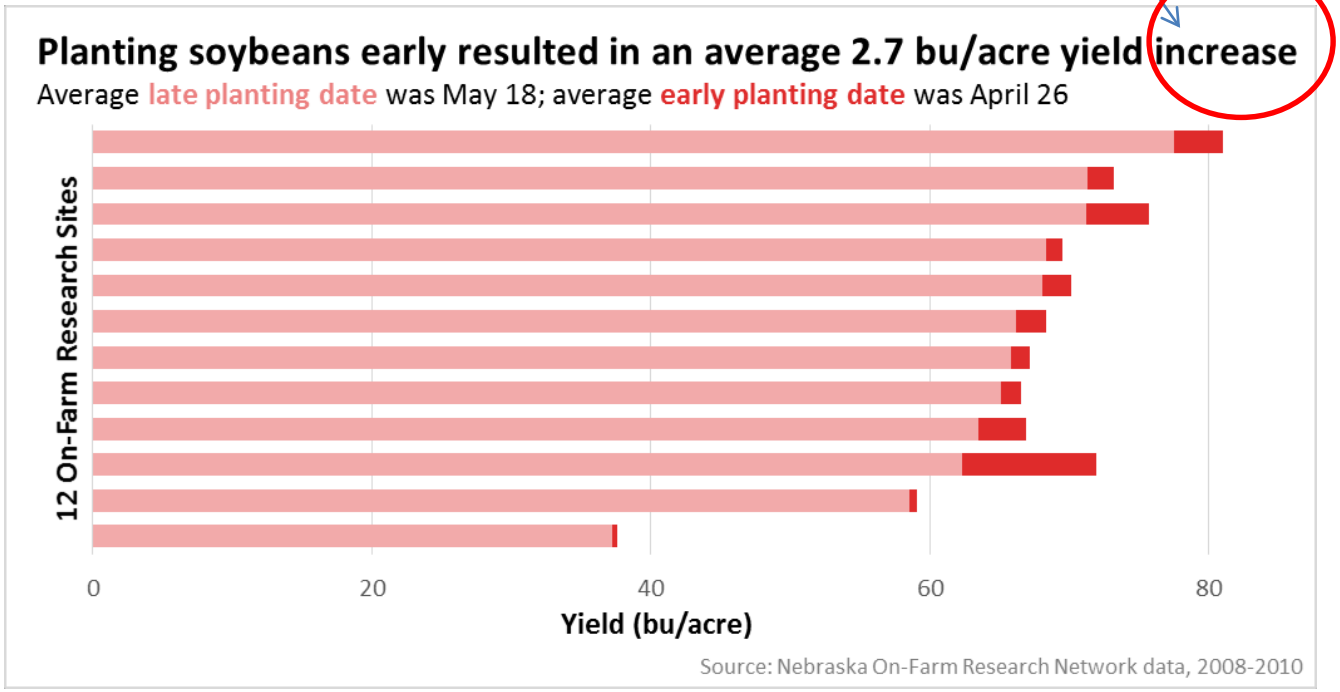
Some Tips on GOOD Visualization

- Take a step back and ask yourself, “What is my point?”
- Choose the right chart
- Use color intentionally

- The graph is bright and eye catching, yet the color is not used in a meaningful way.
- Separating the various sites with different colors is not important and only detracts from the overall point.



- Using color to make the key point stand out. In the following graph
- Use a lighter shade of red for the late planting date, and a darker shade of red for the early planting date.



Some Tips on GOOD Visualization

- Take a step back and ask yourself, “What is my point?”
- Choose the right chart
- Use color intentionally
- Create pointed titles and call out key points with text
- Get feedback and iterate
- Read up and copy other visualizations

More Help on Matplotlib

- https://matplotlib.org/users/pyplot_tutorial.html

- How to FAQ
  - https://matplotlib.org/faq/howto_faq.html