



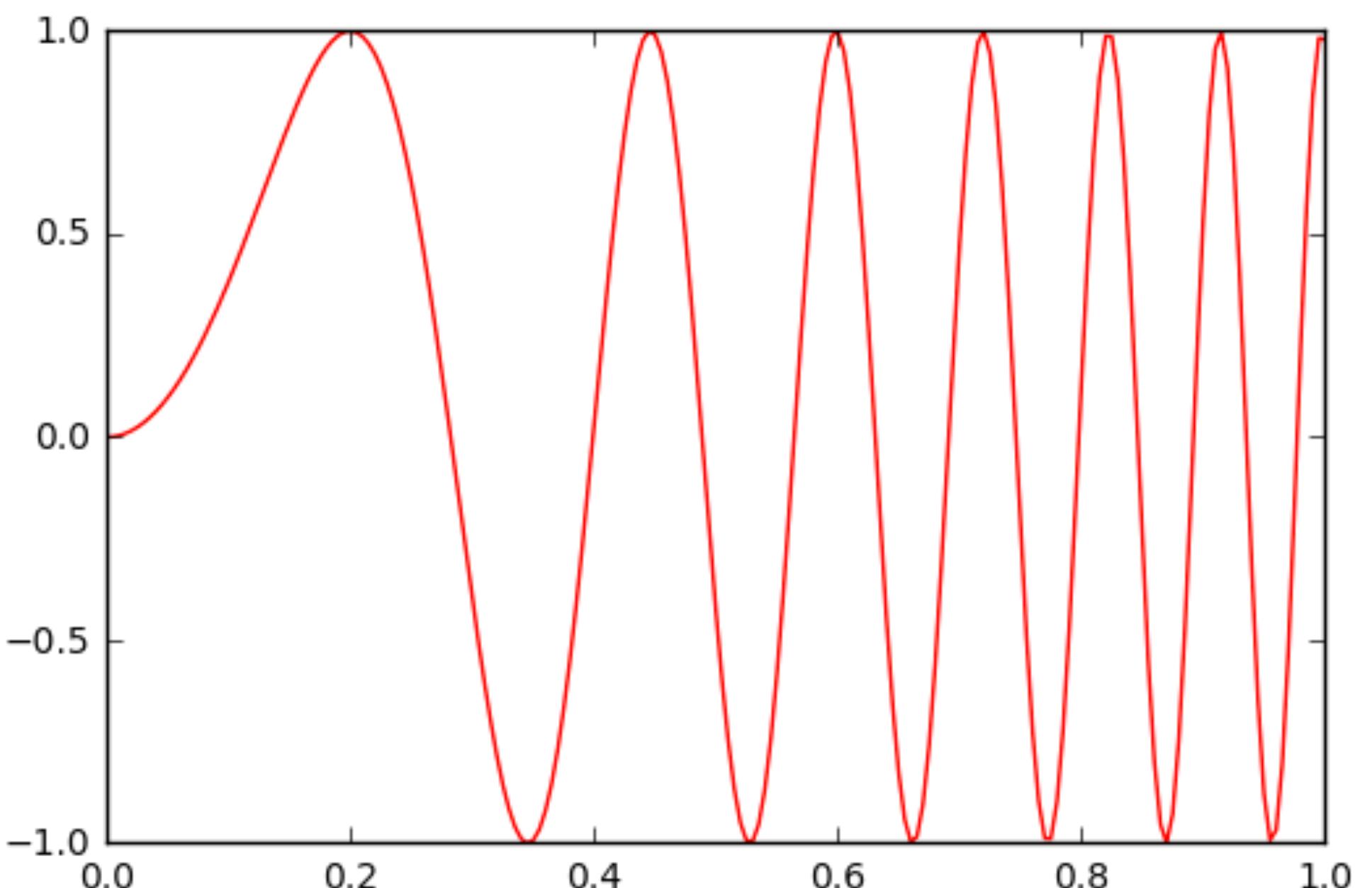
INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

# Introduction to Data Visualization with Python



# Reminder: Line plots

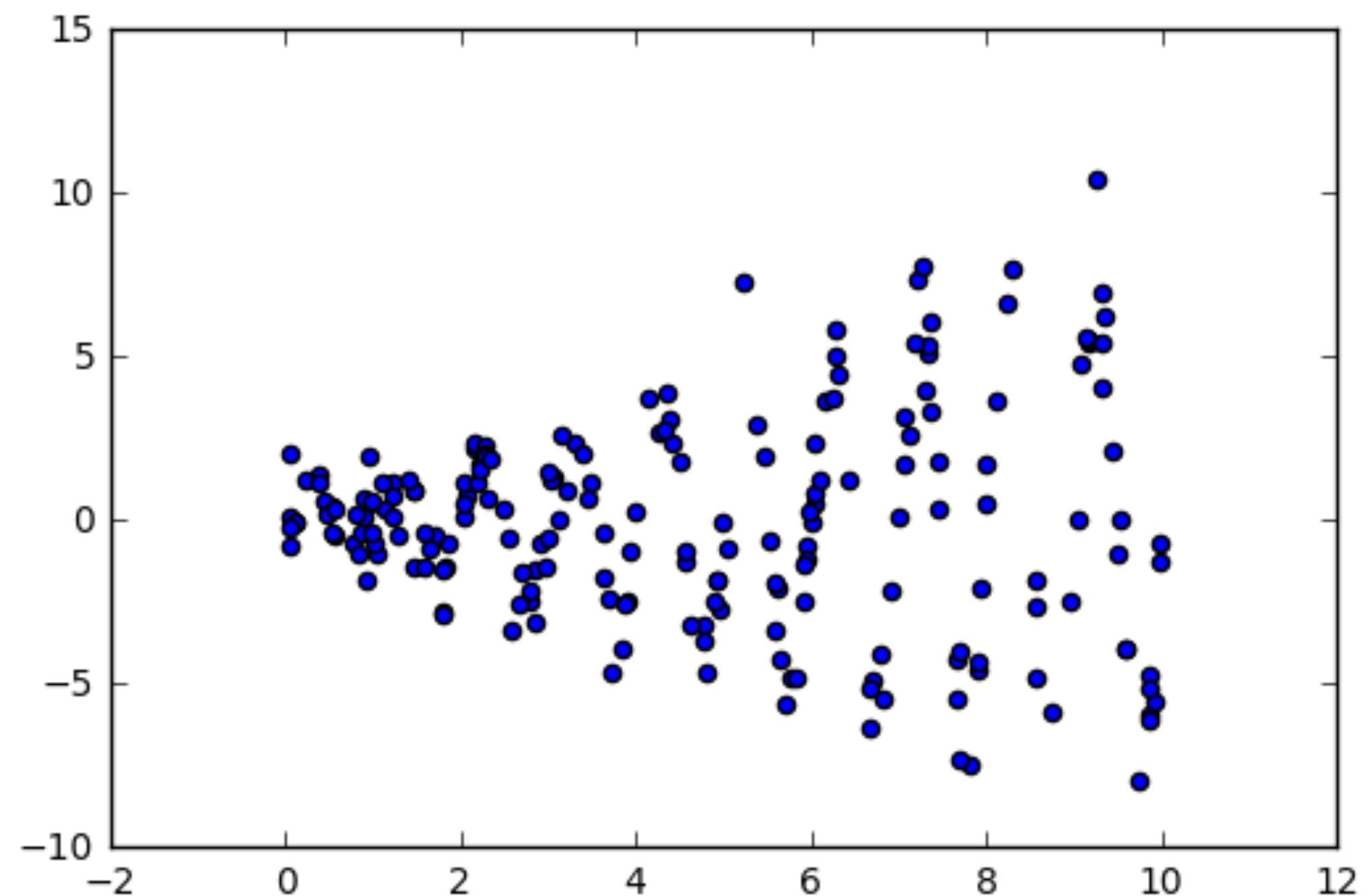
```
In [1]: import numpy as np  
  
In [2]: import matplotlib.pyplot as plt  
  
In [3]: x = np.linspace(0, 1, 201)  
  
In [4]: y = np.sin((2*np.pi*x)**2)  
  
In [5]: plt.plot(x, y, 'red')  
  
In [6]: plt.show()
```





# Reminder: Scatter plots

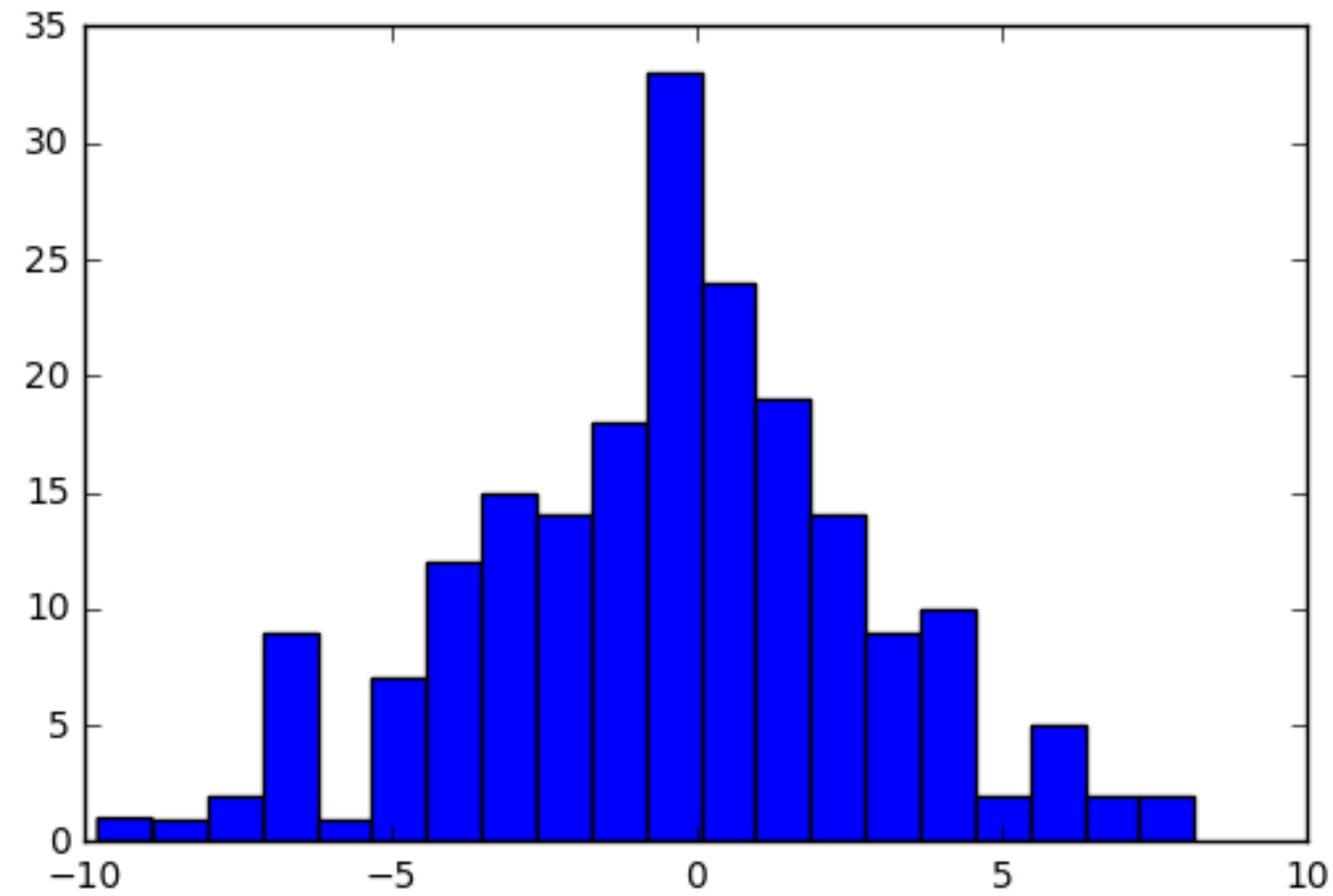
```
In [1]: import numpy as np  
  
In [2]: import matplotlib.pyplot as plt  
  
In [3]: x = 10*np.random.rand(200,1)  
  
In [4]: y = (0.2 + 0.8*x) * np.sin(2*np.pi*x) + \  
...:     np.random.randn(200,1)  
  
In [5]: plt.scatter(x,y)  
  
In [6]: plt.show()
```





# Reminder: Histograms

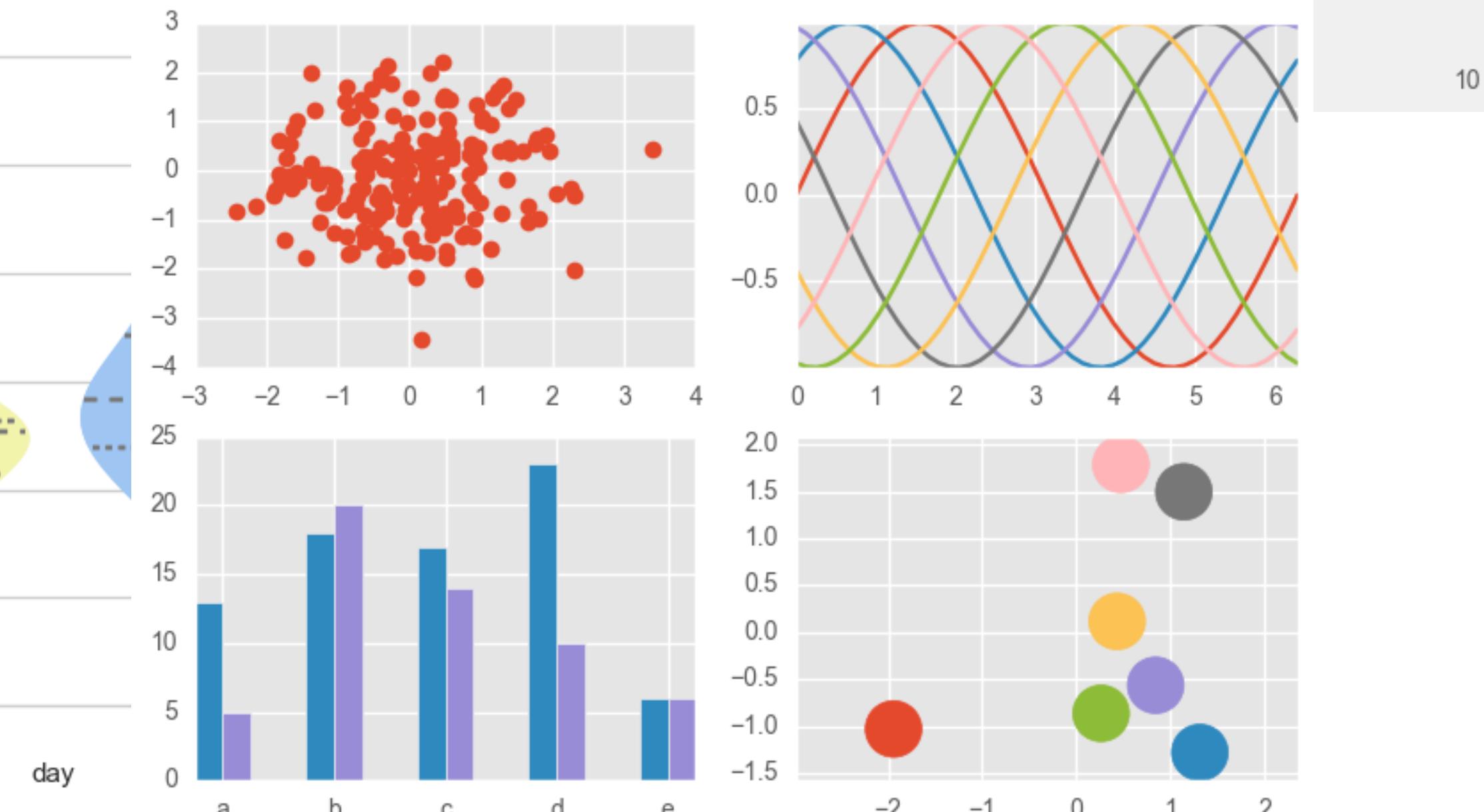
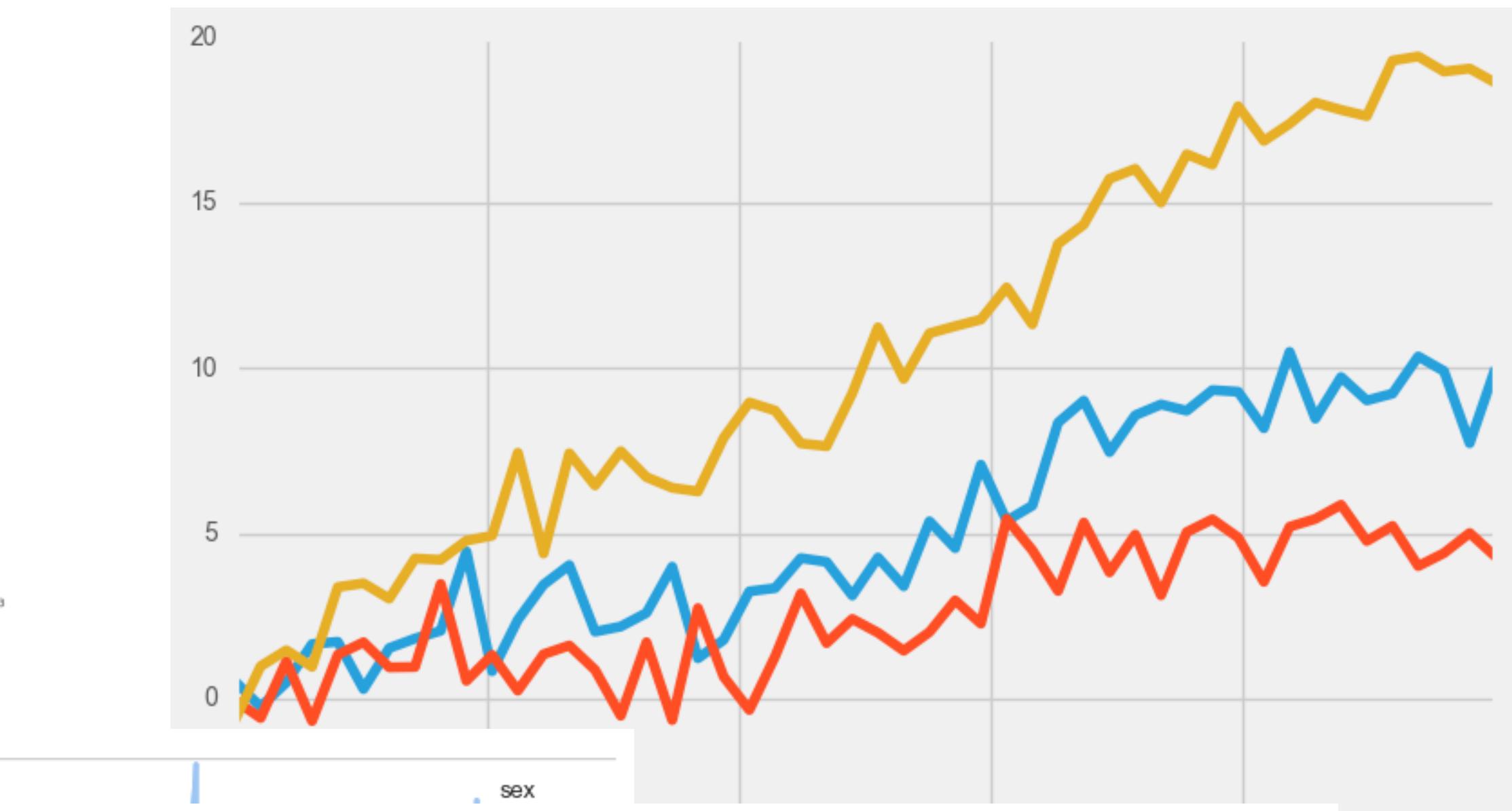
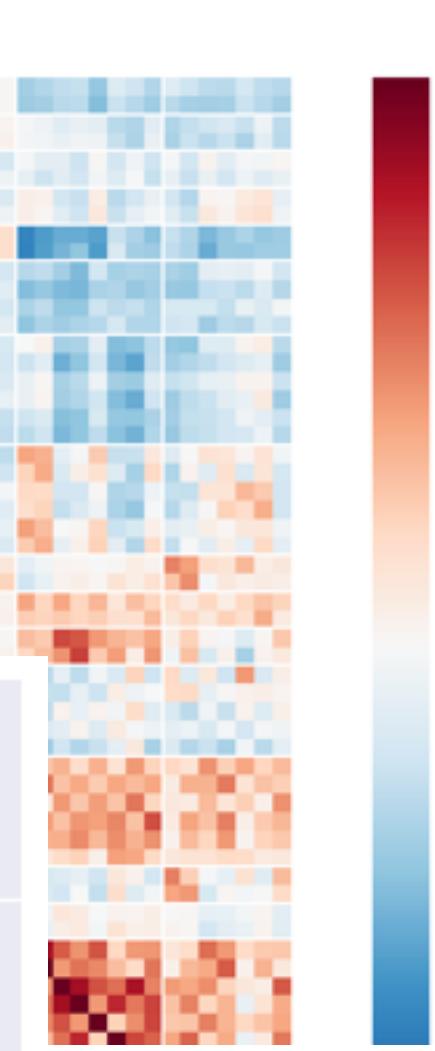
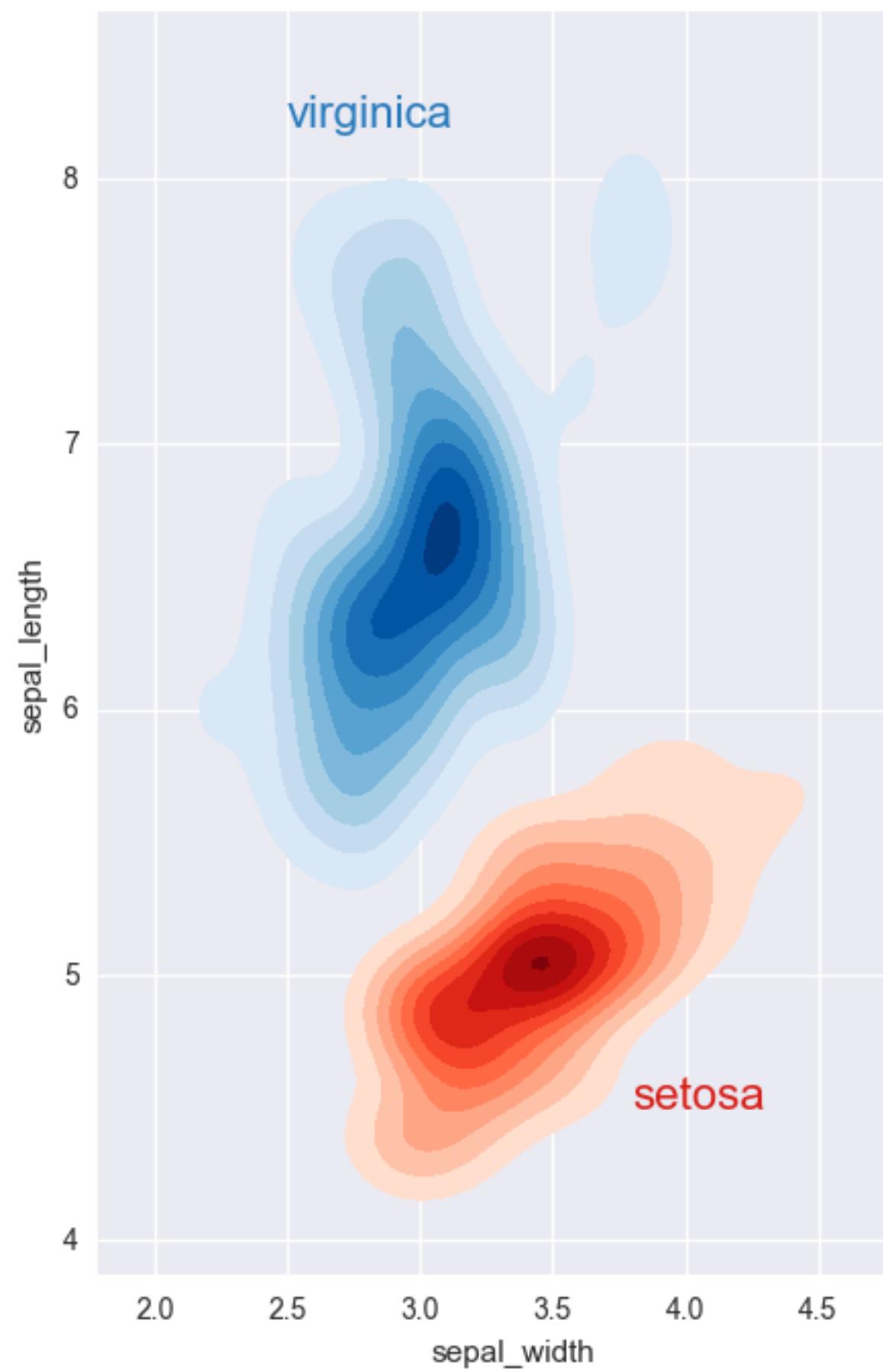
```
In [1]: import numpy as np  
  
In [2]: import matplotlib.pyplot as plt  
  
In [3]: x = 10*np.random.rand(200,1)  
  
In [4]: y = (0.2 + 0.8*x) * \  
...:     np.sin(2*np.pi*x) + \  
...:     np.random.randn(200,1)  
  
In [5]: plt.hist(y, bins=20)  
  
In [6]: plt.show()
```





# What you will learn

- Customizing of plots: axes, annotations, legends
- Overlaying multiple plots and subplots
- Visualizing 2D arrays, 2D data sets
- Working with color maps
- Producing statistical graphics
- Plotting time series
- Working with images





## INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

**See you in  
the course!**



INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

# Plotting multiple graphs



# Strategies

- Plotting many graphs on common axes
- Creating axes within a figure
- Creating subplots within a figure

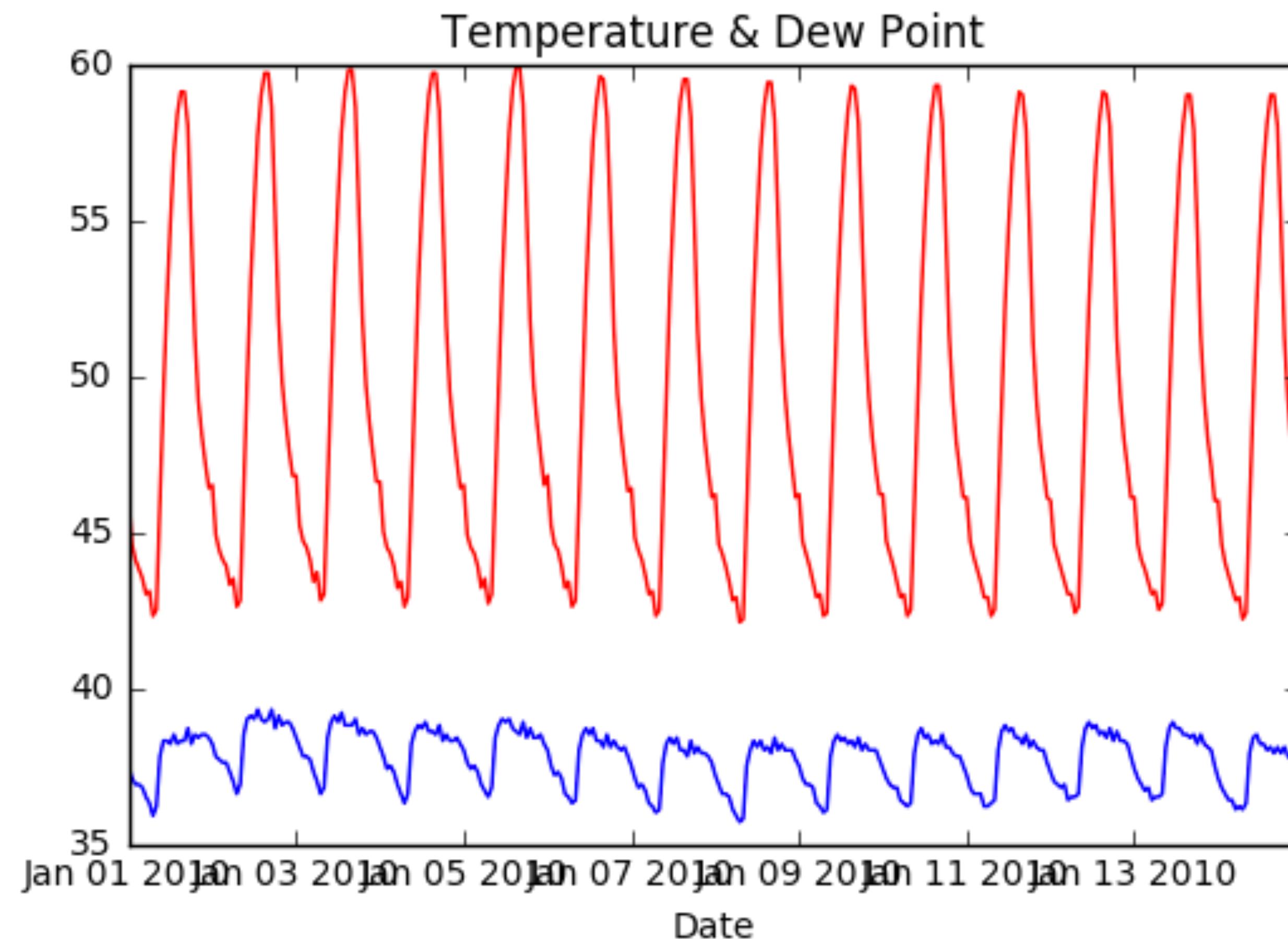


# Graphs on common axes

```
In [1]: import matplotlib.pyplot as plt  
  
In [2]: plt.plot(t, temperature, 'red')  
  
In [3]: plt.plot(t, dewpoint, 'blue') # Appears on same axes  
  
In [4]: plt.xlabel('Date')  
  
In [5]: plt.title('Temperature & Dew Point')  
  
In [6]: plt.show() # Renders plot objects to screen
```



# Graphs on common axes





# Using axes()

```
In [1]: plt.axes([0.05,0.05,0.425,0.9])
```

```
In [2]: plt.plot(t, temperature, 'red')
```

```
In [3]: plt.xlabel('Date')
```

```
In [4]: plt.title('Temperature')
```

```
In [5]: plt.axes([0.525,0.05,0.425,0.9])
```

```
In [6]: plt.plot(t, dewpoint, 'blue')
```

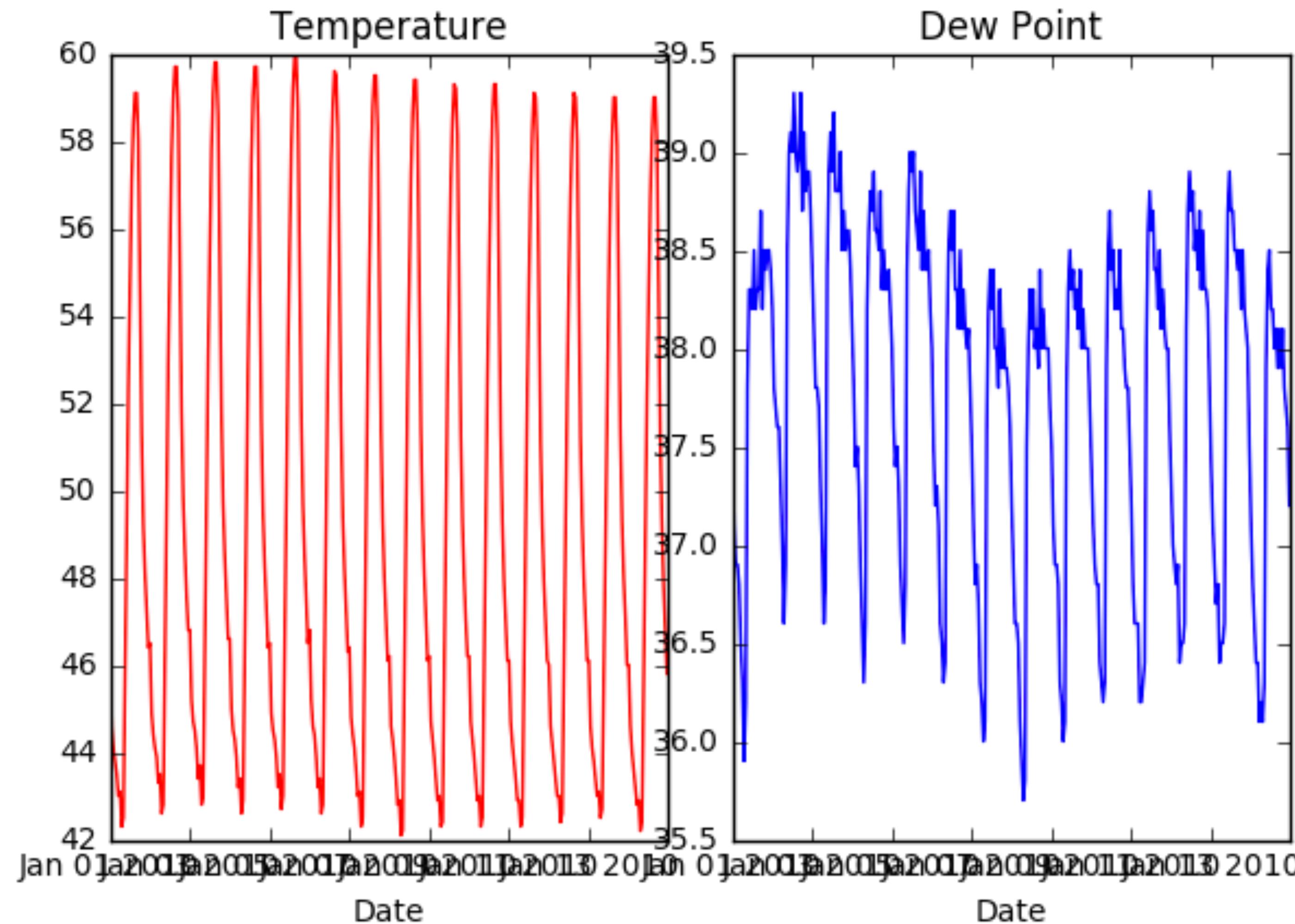
```
In [7]: plt.xlabel('Date')
```

```
In [8]: plt.title('Dew Point')
```

```
In [9]: plt.show()
```

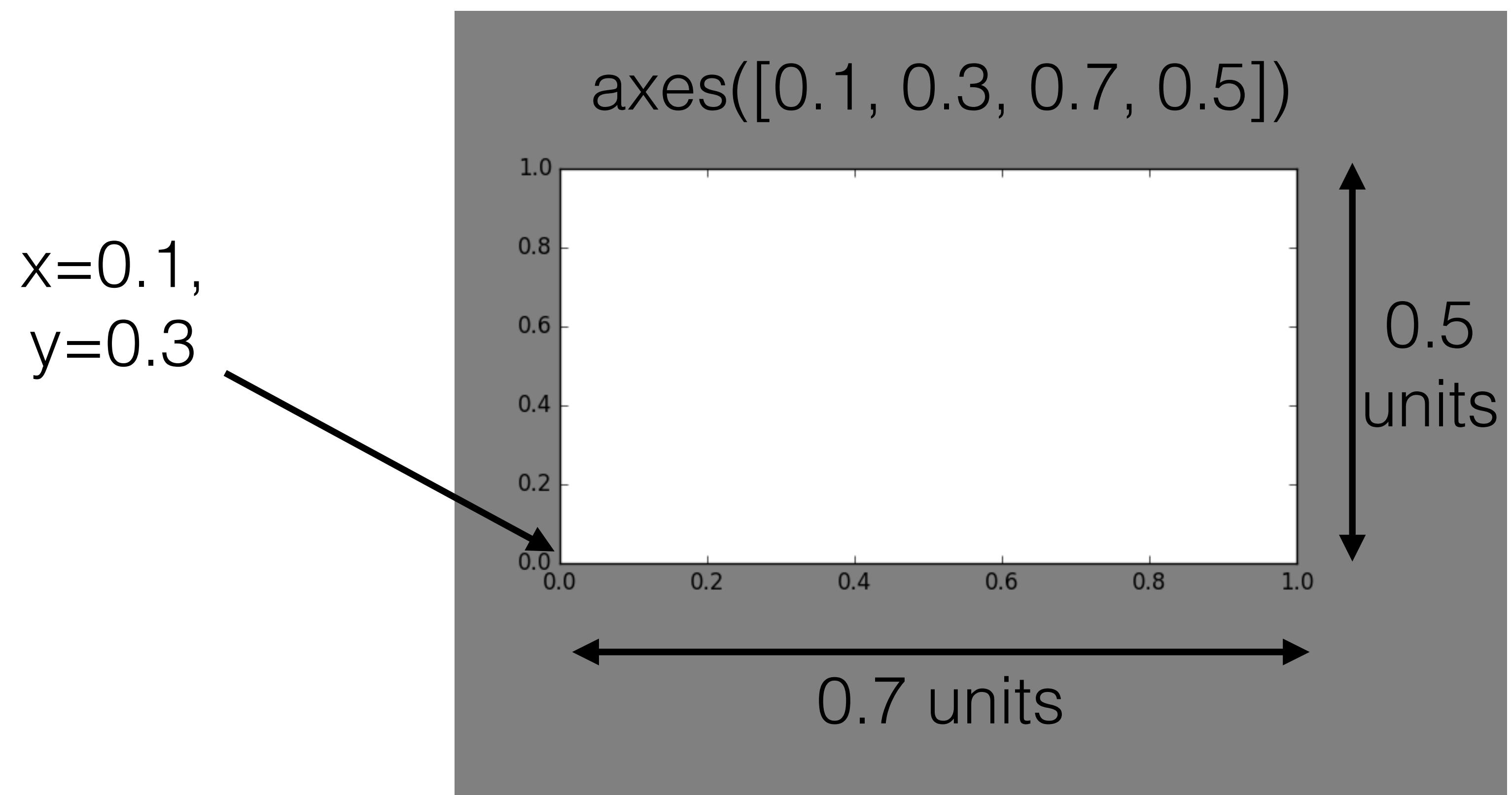


# Using axes()

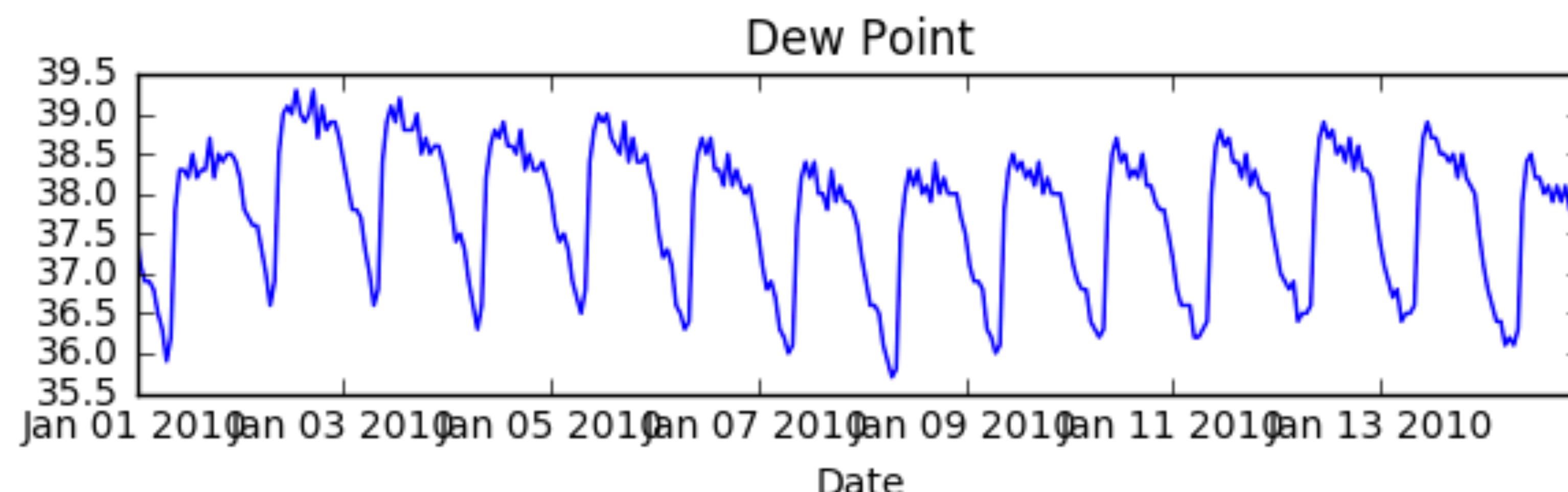
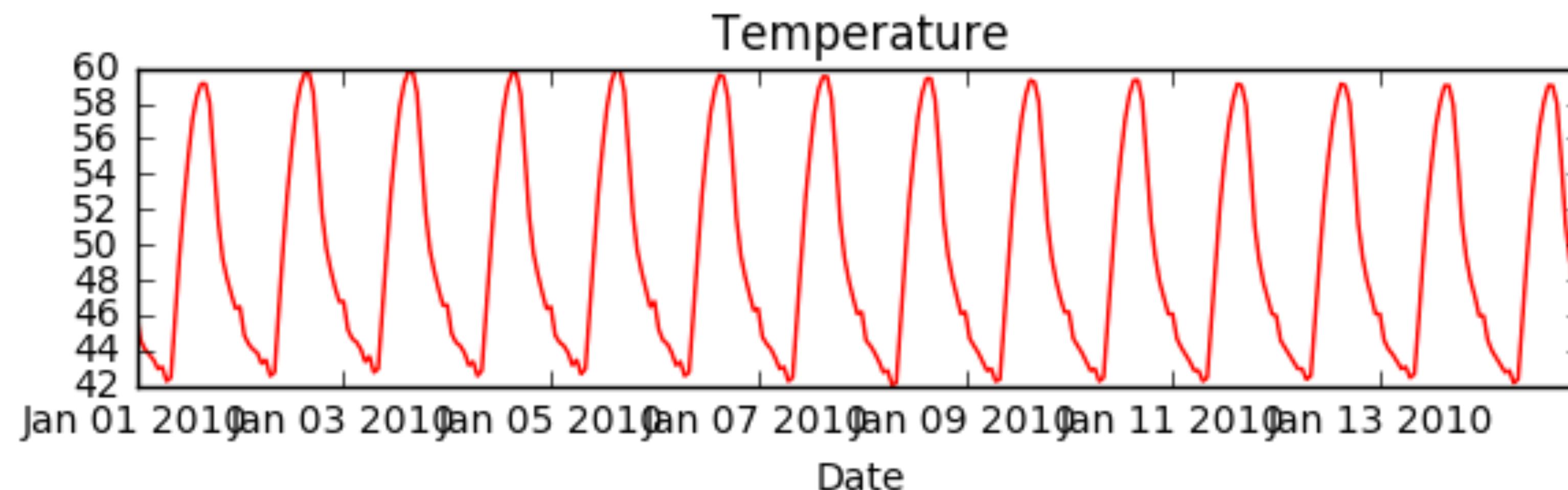


# The axes() command

- Syntax: `axes([x_lo, y_lo, width, height])`
- Units between 0 and 1 (figure dimensions)



# Using subplot()





# Using subplot()

```
In [1]: plt.subplot(2, 1, 1)
```

```
In [2]: plt.plot(t, temperature, 'red')
```

```
In [3]: plt.xlabel('Date')
```

```
In [4]: plt.title('Temperature')
```

```
In [5]: plt.subplot(2, 1, 2)
```

```
In [6]: plt.plot(t, dewpoint, 'blue')
```

```
In [7]: plt.xlabel('Date')
```

```
In [8]: plt.title('Dew Point')
```

```
In [9]: plt.tight_layout()
```

```
In [10]: plt.show()
```



# The subplot() command

- Syntax: `subplot(nrows, ncols, nsubplot)`
- Subplot ordering:
  - Row-wise from top left
  - Indexed from 1



INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

**Let's practice!**



INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

# Customizing axes



# Controlling axis extents

- `axis([xmin, xmax, ymin, ymax])` sets axis extents
- Control over individual axis extents
  - `xlim([xmin, xmax])`
  - `ylim([ymin, ymax])`
- Can use tuples, lists for extents
  - e.g., `xlim((-2, 3))` works
  - e.g., `xlim([-2, 3])` works also

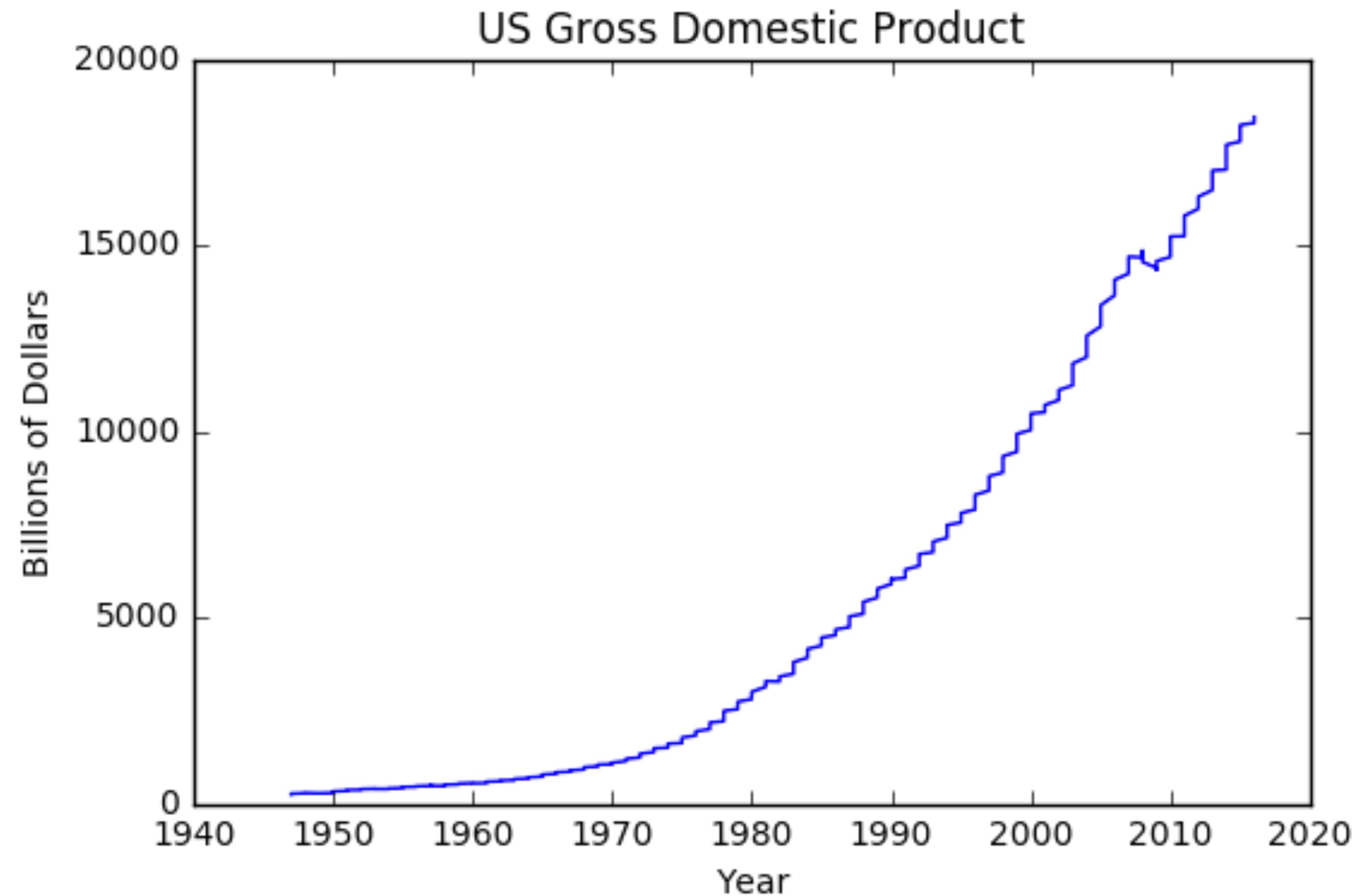


# GDP over time

```
In [1]: import matplotlib.pyplot as plt  
  
In [2]: plt.plot(yr, gdp)  
  
In [3]: plt.xlabel('Year')  
  
In [4]: plt.ylabel('Billions of Dollars')  
  
In [5]: plt.title('US Gross Domestic Product')  
  
In [6]: plt.show()
```



# GDP over time





# Using xlim()

```
In [1]: plt.plot(yr, gdp)

In [2]: plt.xlabel('Year')

In [3]: plt.ylabel('Billions of Dollars')

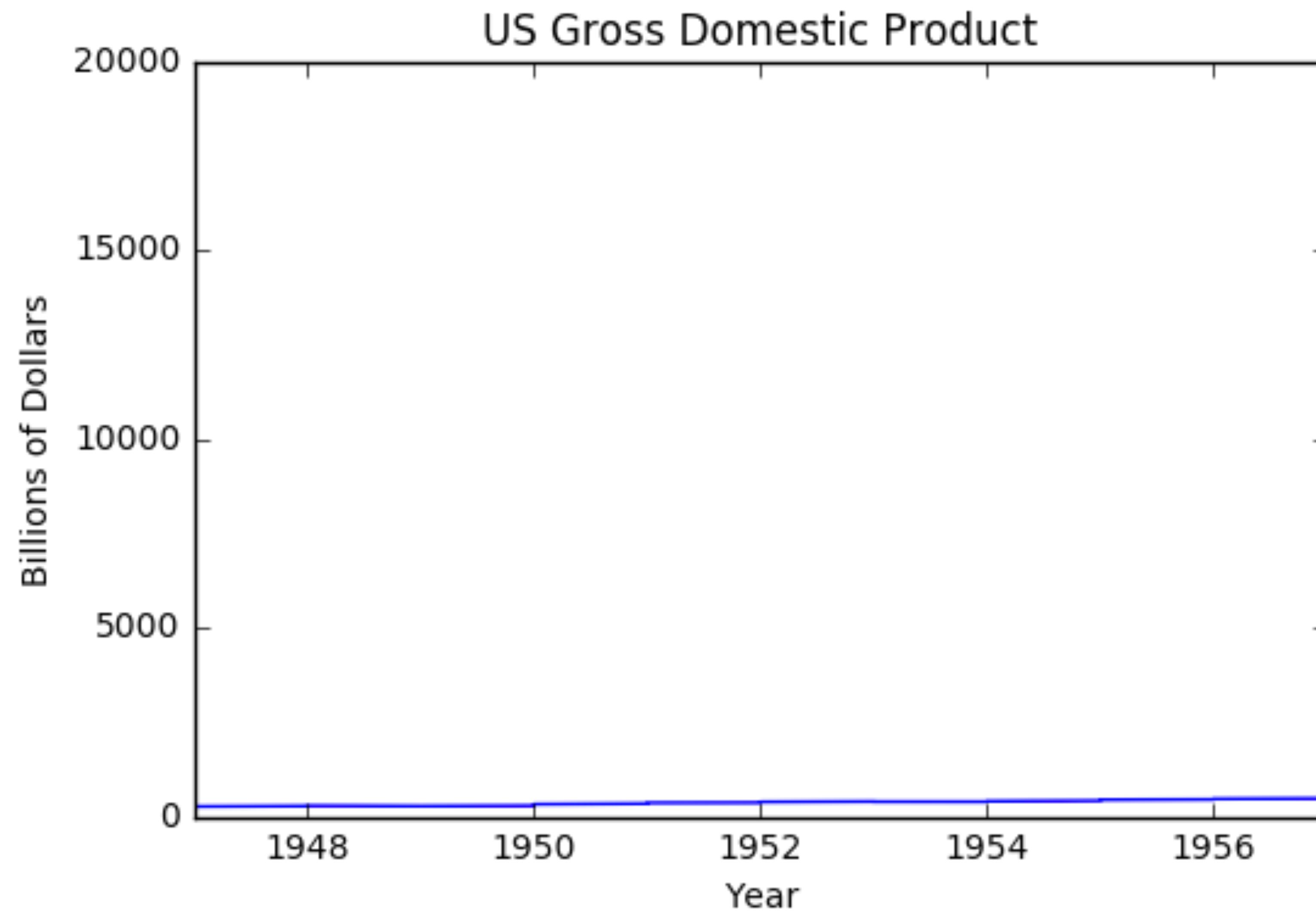
In [4]: plt.title('US Gross Domestic Product')

In [5]: plt.xlim((1947, 1957))

In [6]: plt.show()
```



# Using xlim()





# Using xlim() & ylim()

```
In [1]: plt.plot(yr, gdp)

In [2]: plt.xlabel('Year')

In [3]: plt.ylabel('Billions of Dollars')

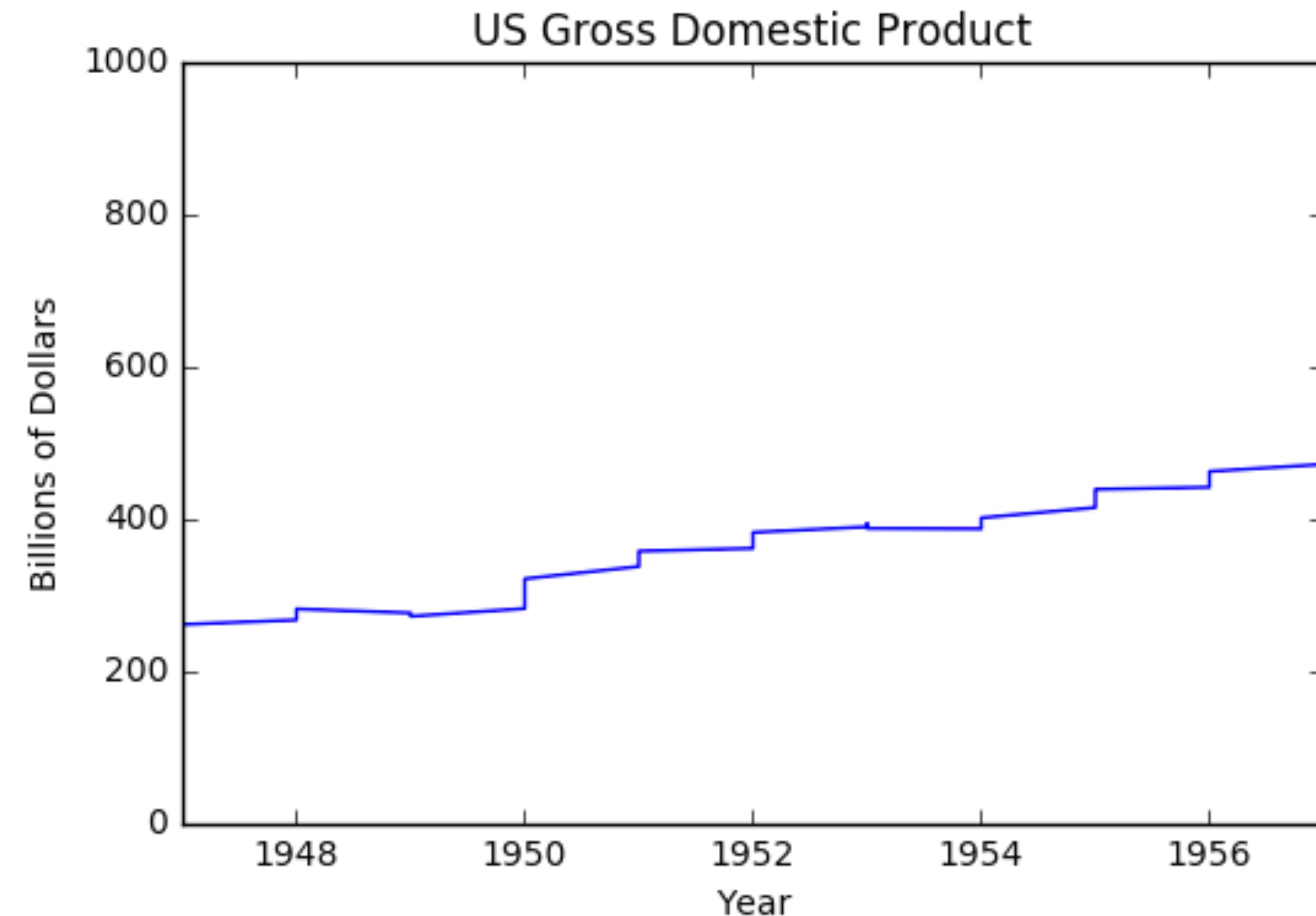
In [4]: plt.title('US Gross Domestic Product')

In [5]: plt.xlim((1947, 1957))

In [6]: plt.ylim((0, 1000))

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

# Using xlim() & ylim()





# Using axis()

```
In [1]: plt.plot(yr, gdp)

In [2]: plt.xlabel('Year')

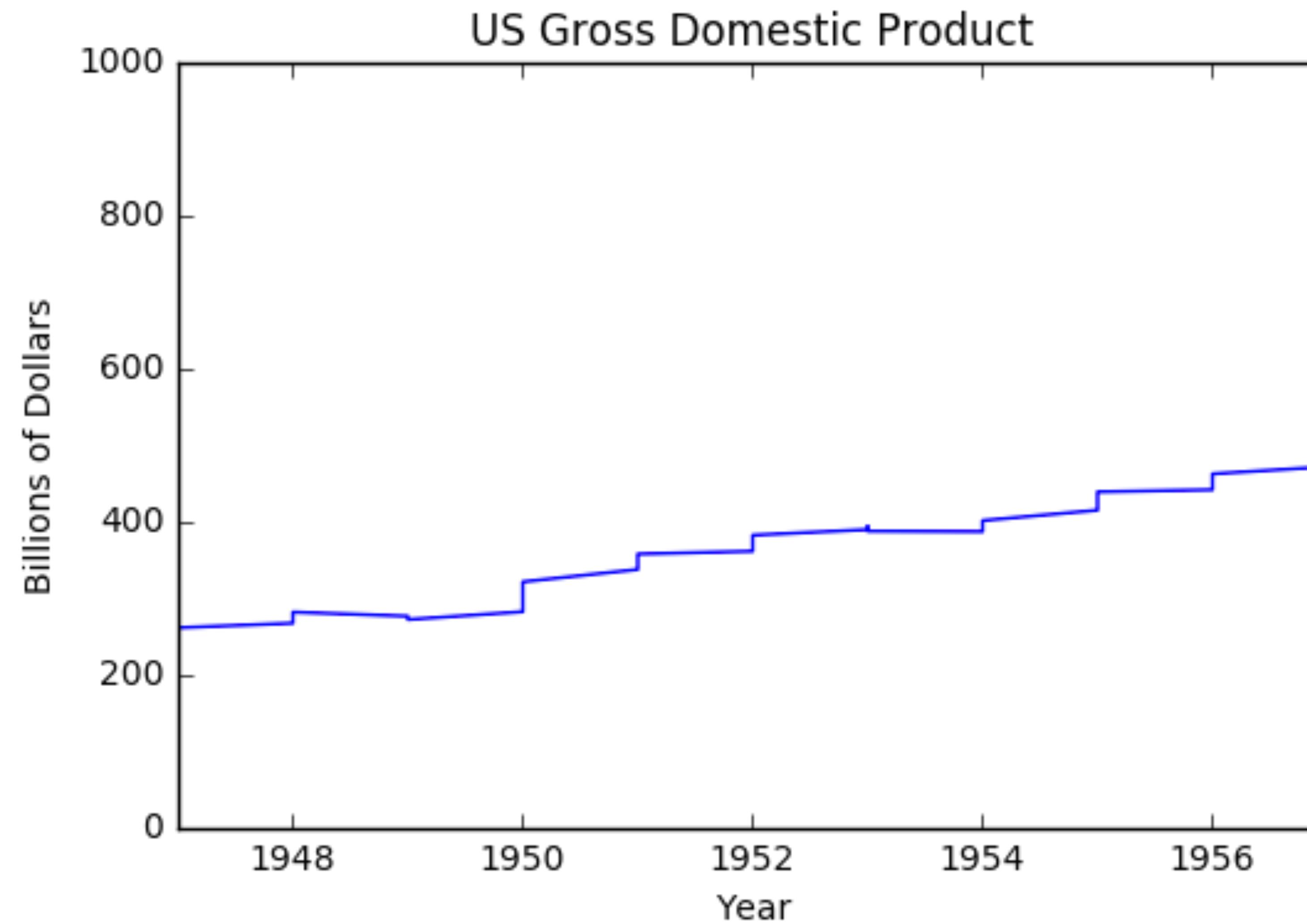
In [3]: plt.ylabel('Billions of Dollars')

In [4]: plt.title('US Gross Domestic Product')

In [5]: plt.axis((1947, 1957, 0, 600))

In [6]: plt.show()
```

# Using axis()

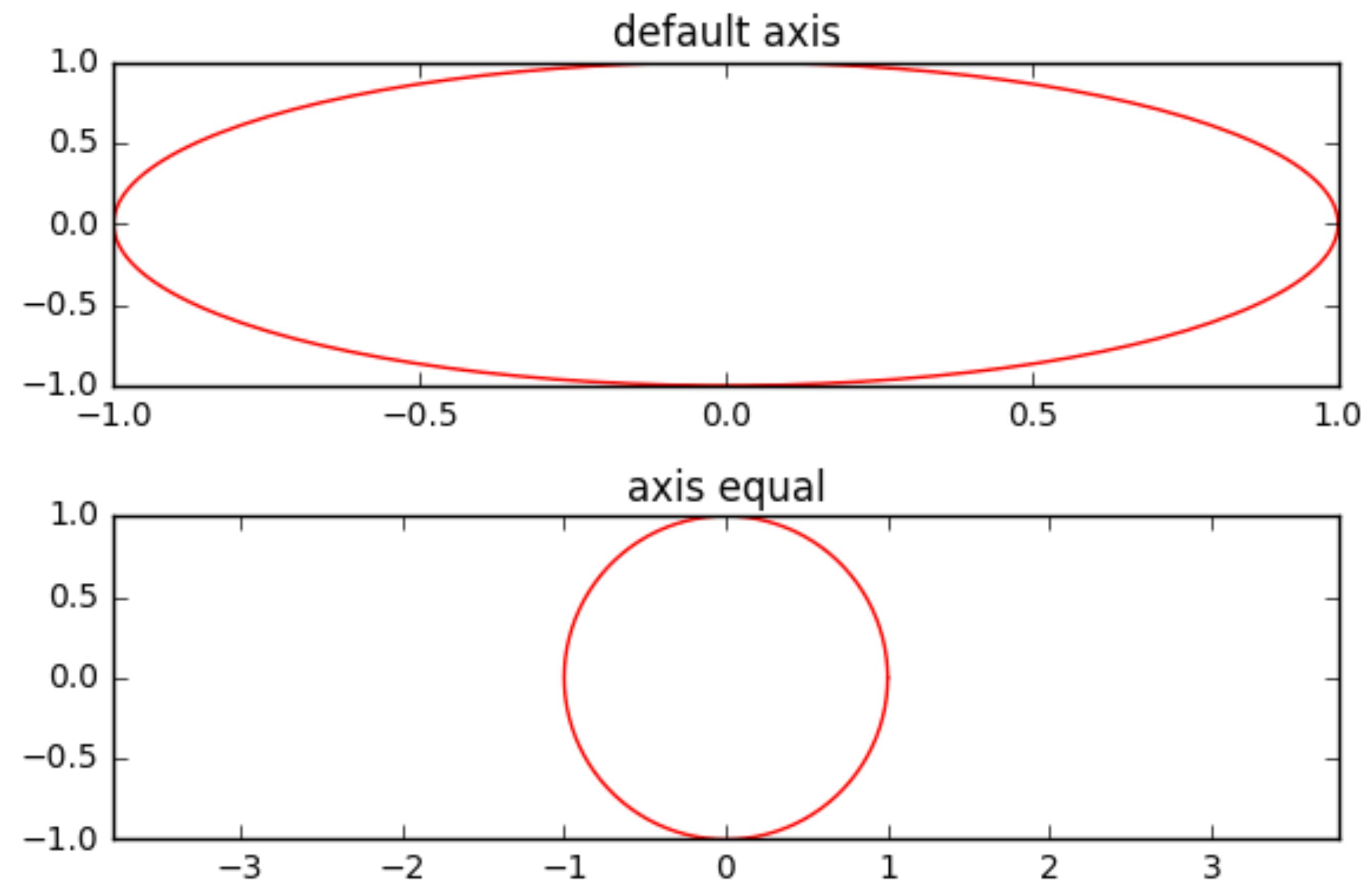




# Other axis() options

Invocation	Result
<code>axis('off')</code>	turns off axis lines, labels
<code>axis('equal')</code>	equal scaling on x, y axes
<code>axis('square')</code>	forces square plot
<code>axis('tight')</code>	sets <code>xlim()</code> , <code>ylim()</code> to show all data

# Using axis('equal')





# Using axis('equal')

```
In [1]: plt.subplot(2, 1, 1)
```

```
In [2]: plt.plot(x, y, 'red')
```

```
In [3]: plt.title('default axis')
```

```
In [4]: plt.subplot(2, 1, 2)
```

```
In [5]: plt.plot(x, y, 'red')
```

```
In [6]: plt.axis('equal')
```

```
In [7]: plt.title('axis equal')
```

```
In [8]: plt.tight_layout()
```

```
In [9]: plt.show()
```



INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

**Let's practice!**



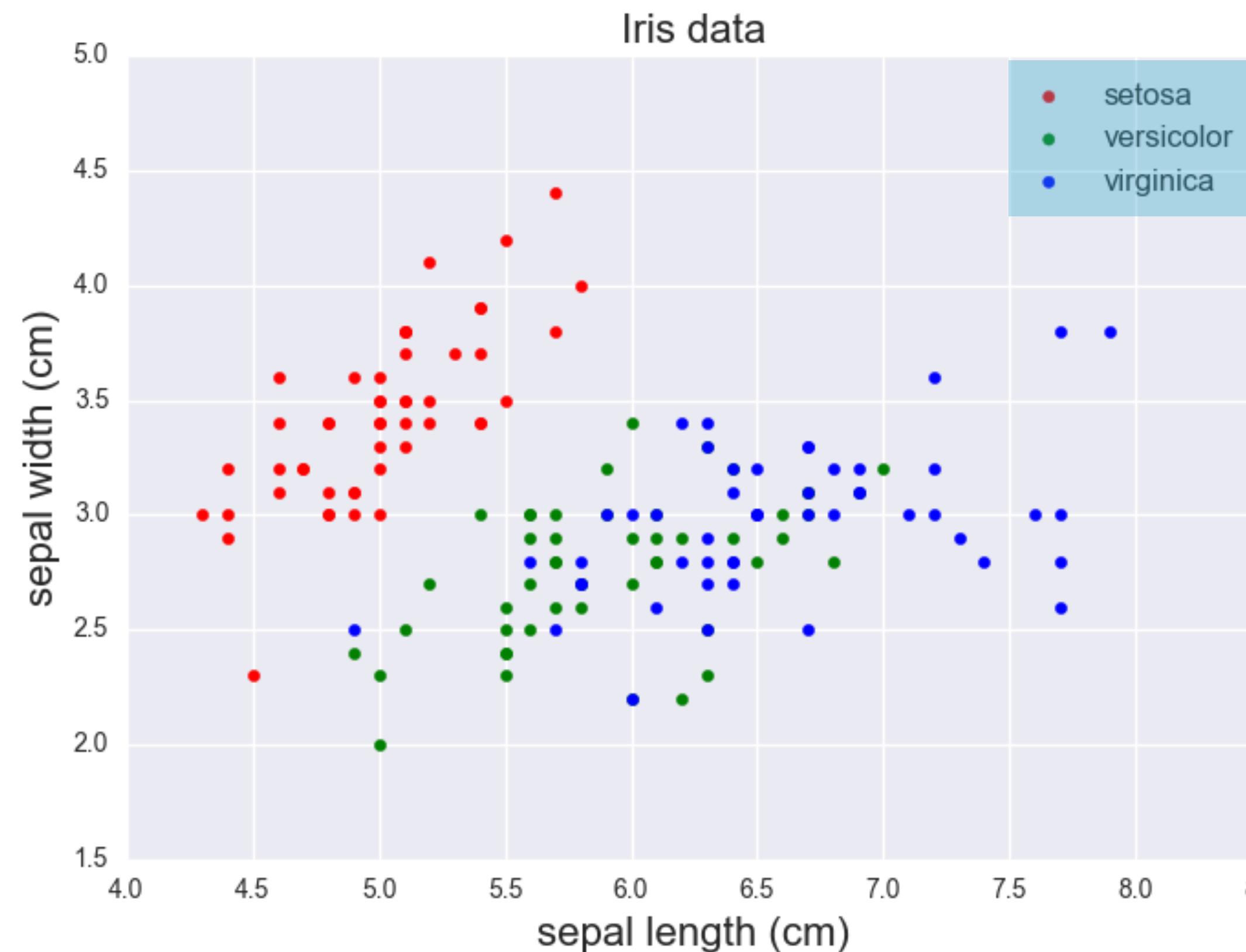
INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

# Legends, annotations, and styles



# Legends

- provide labels for overlaid points and curves



Legend



# Using legend()

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

```
In [2]: plt.scatter(setosa_len, setosa_wid,  
...:                 marker='o', color='red', label='setosa')
```

```
In [3]: plt.scatter(versicolor_len, versicolor_wid,  
...:                 marker='o', color='green', label='versicolor')
```

```
In [4]: plt.scatter(virginica_len, virginica_wid,  
...:                 marker='o', color='blue', label='virginica')
```



# Using legend()

```
In [5]: plt.legend(loc='upper right')
```

```
In [6]: plt.title('Iris data')
```

```
In [7]: plt.xlabel('sepal length (cm)')
```

```
In [8]: plt.ylabel('sepal width (cm)')
```

```
In [9]: plt.show()
```



# Legend locations

string	code	string	code	string	code
'upper left'	2	'upper center'	9	'upper right'	1
'center left'	6	'center'	10	'center right'	7
'lower left'	3	'lower center'	8	'lower right'	4
'best'	0			'right'	5



# Plot annotations

- Text labels and arrows using `annotate()` method
- Flexible specification of coordinates
- Keyword `arrowprops`: dict of arrow properties
  - `width`
  - `color`
  - etc.



# Using `annotate()` for text

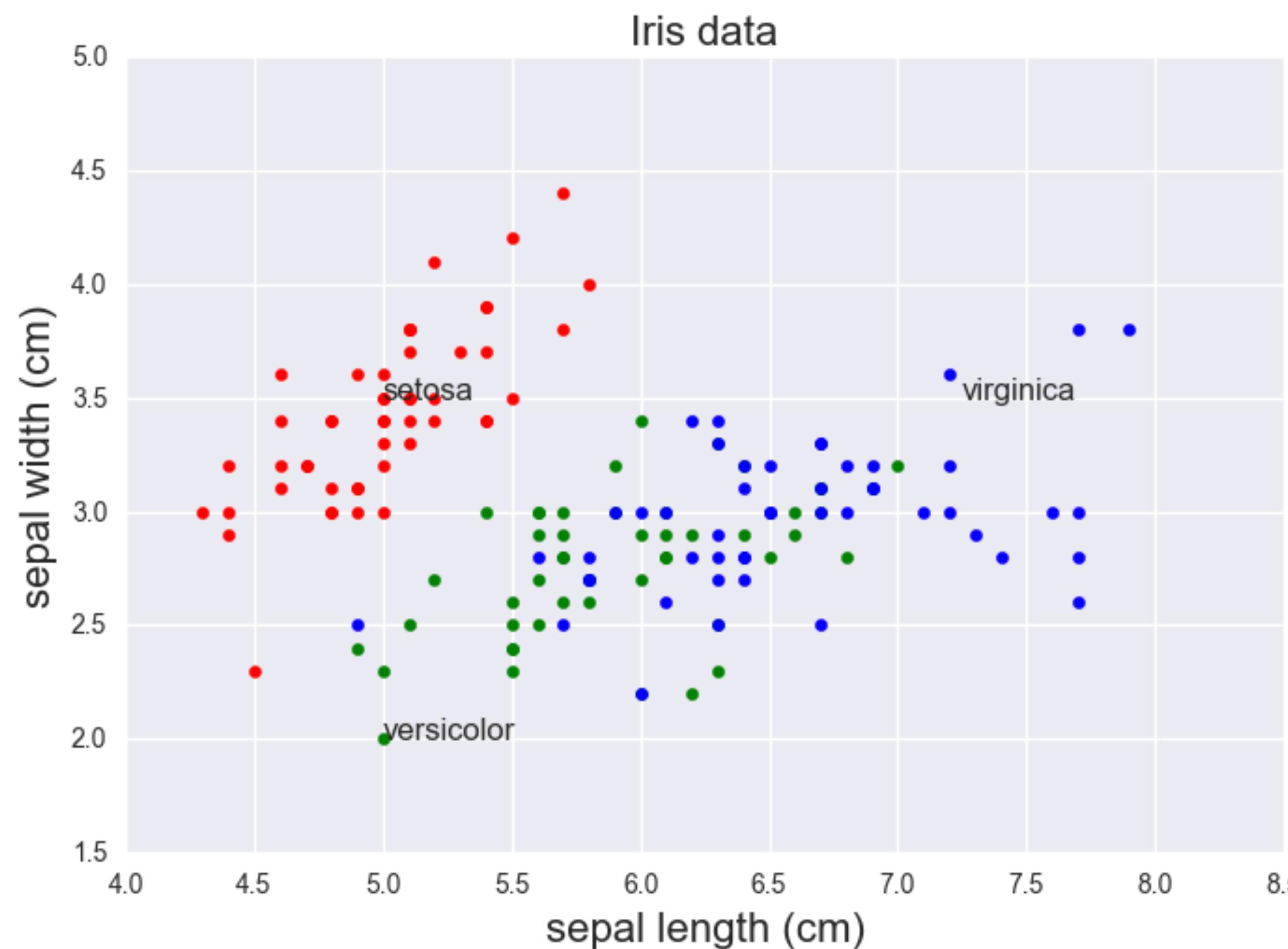
```
In [1]: plt.annotate('setosa', xy=(5.0, 3.5))
```

```
In [2]: plt.annotate('virginica', xy=(7.25, 3.5))
```

```
In [3]: plt.annotate('versicolor', xy=(5.0, 2.0))
```

```
In [4]: plt.show()
```

# Using `annotate()` for text



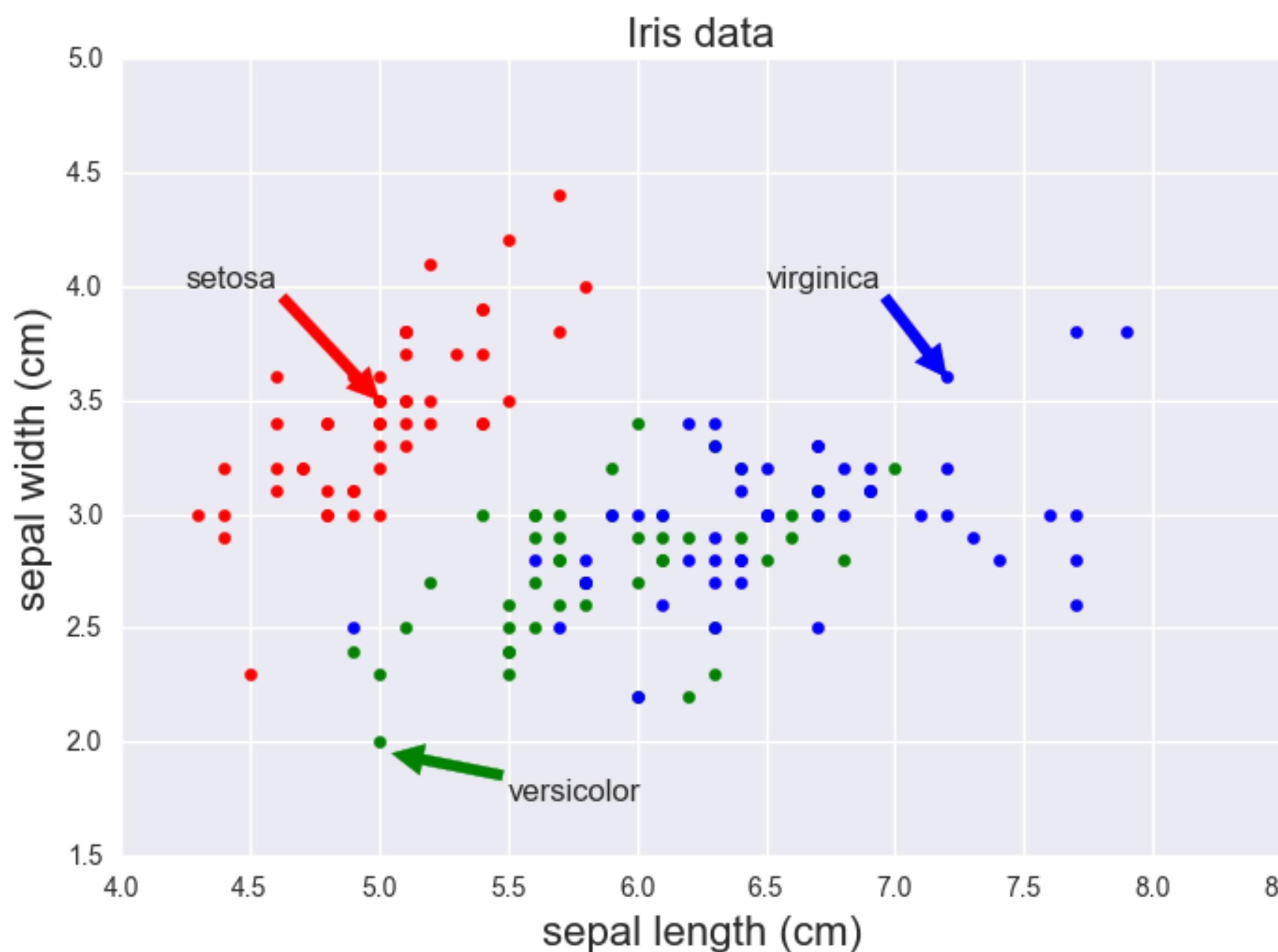


# Options for `annotate()`

option	description
<code>s</code>	text of label
<code>xy</code>	coordinates to annotate
<code>xytext</code>	coordinates of label
<code>arrowprops</code>	controls drawing of arrow



# Using `annotate()` for arrows





# Using annotate() for arrows

```
In [1]: plt.annotate('setosa', xy=(5.0, 3.5),  
...:                  xytext=(4.25, 4.0), arrowprops={'color':'red'})
```

```
In [2]: plt.annotate('virginica', xy=(7.2, 3.6),  
...:                   xytext=(6.5, 4.0), arrowprops={'color':'blue'})
```

```
In [3]: plt.annotate('versicolor', xy=(5.05, 1.95),  
...:                   xytext=(5.5, 1.75),  
...:                   arrowprops={'color':'green'})
```

```
In [4]: plt.show()
```



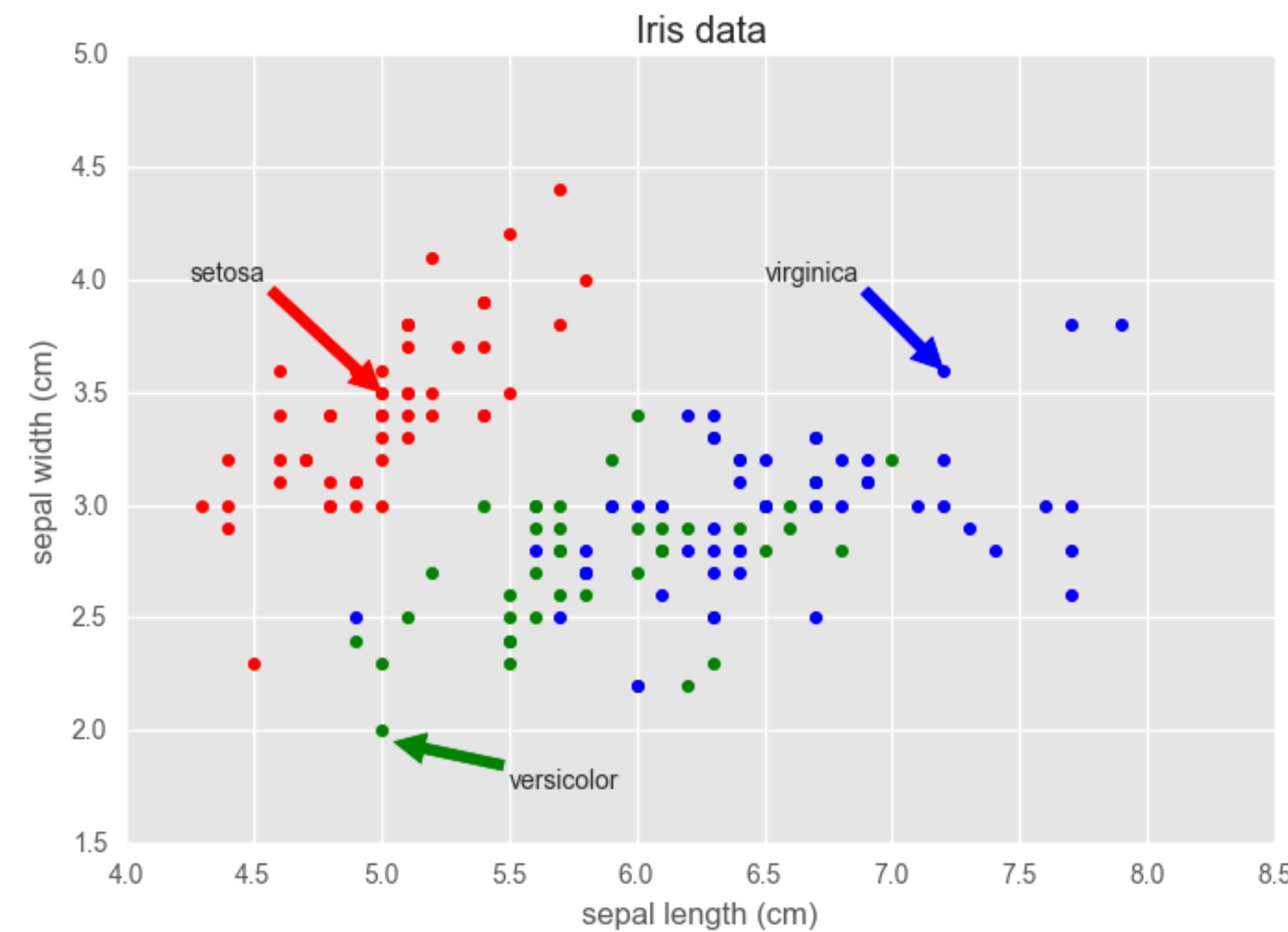
# Working with plot styles

- Style sheets in Matplotlib
- Defaults for lines, points, backgrounds, etc.
- Switch styles globally with `plt.style.use()`
- `plt.style.available`: list of styles

# ggplot style

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

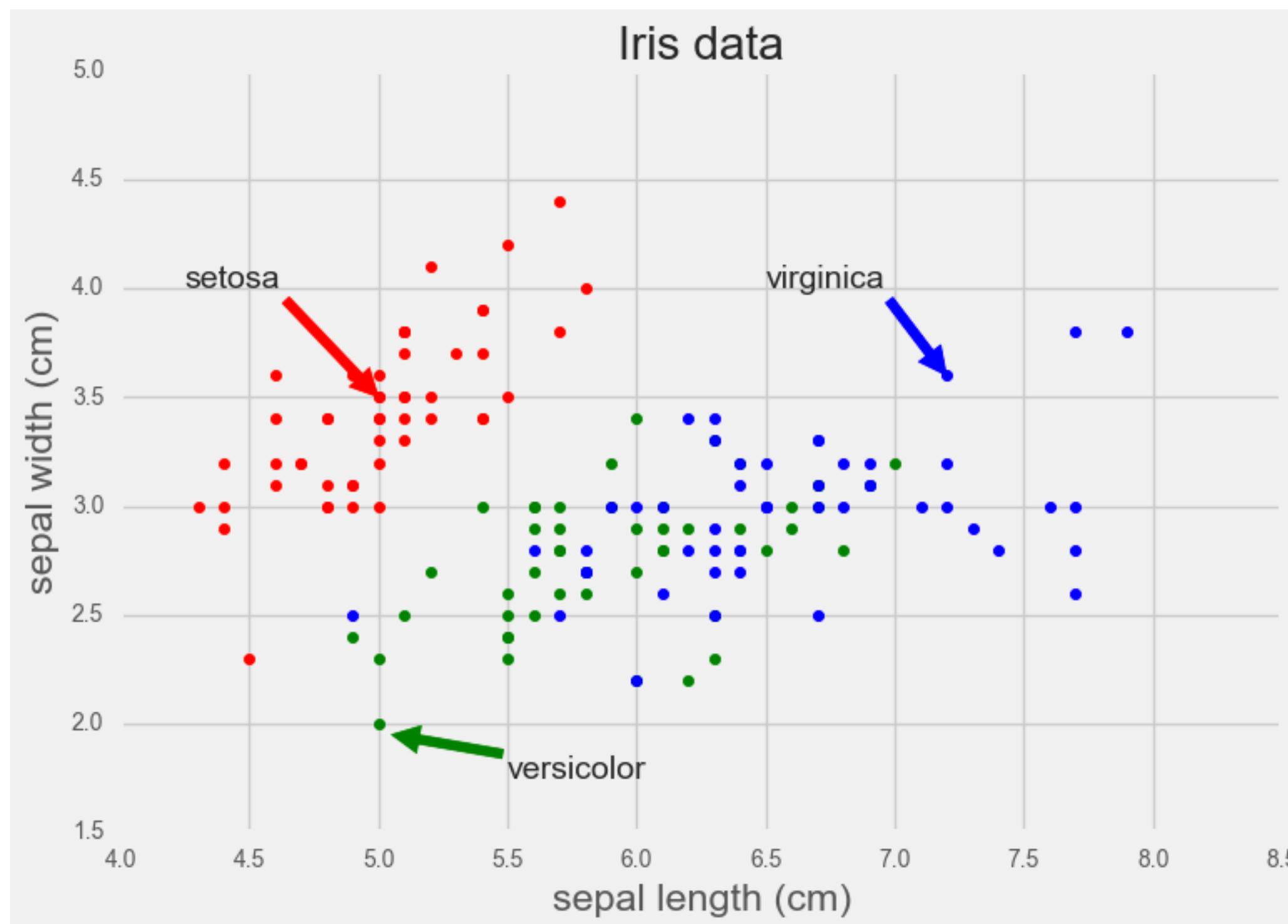
```
In [2]: plt.style.use('ggplot')
```



# fivethirtyeight style

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

```
In [2]: plt.style.use('fivethirtyeight')
```





INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

**Let's practice!**



INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

# Working with 2D arrays



# Reminder: NumPy arrays

- Homogeneous in type
- Calculations all at once
- Indexing with brackets:
  - $A[index]$  for 1D array
  - $A[index_0, index_1]$  for 2D array

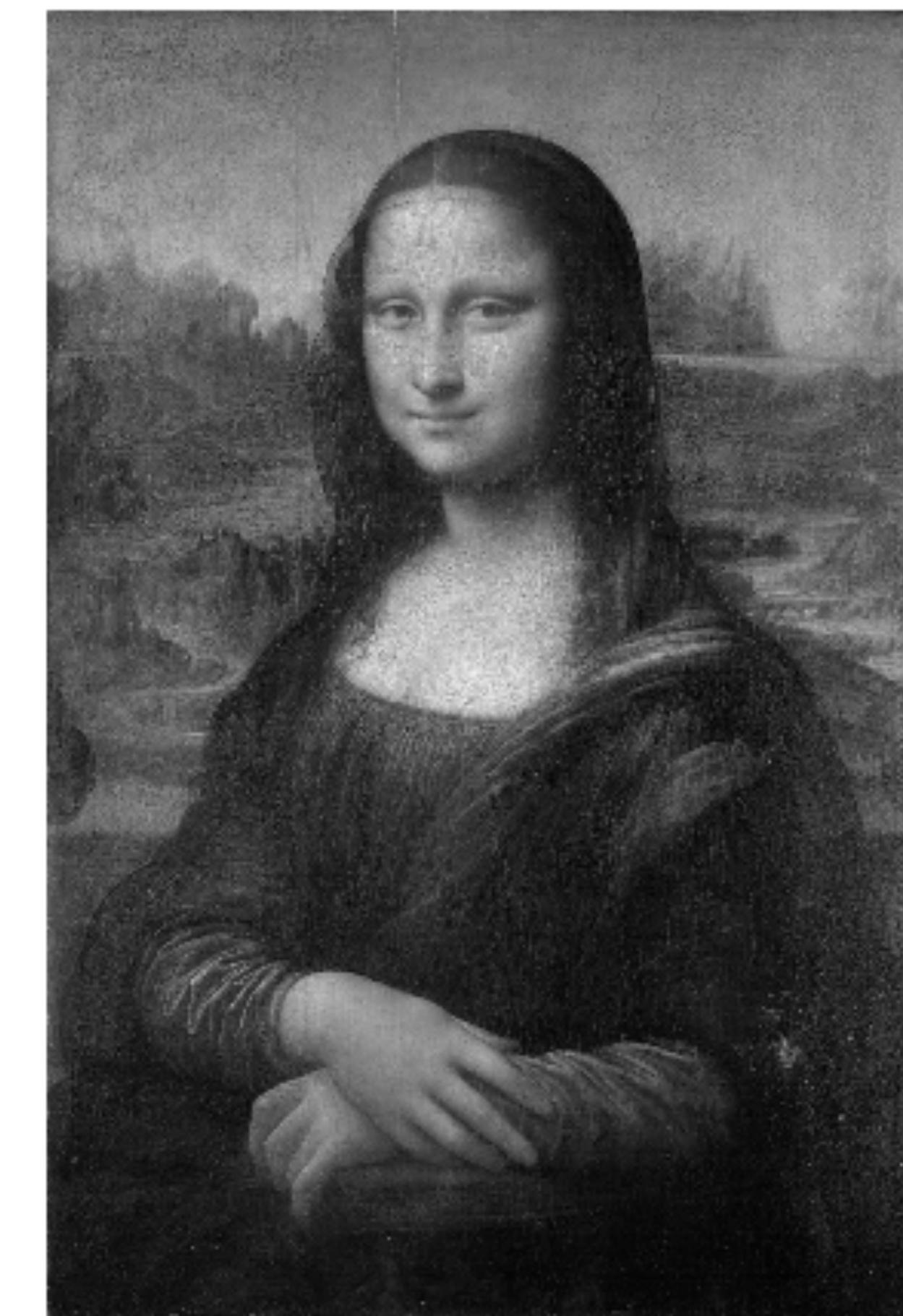


# Reminder: slicing arrays

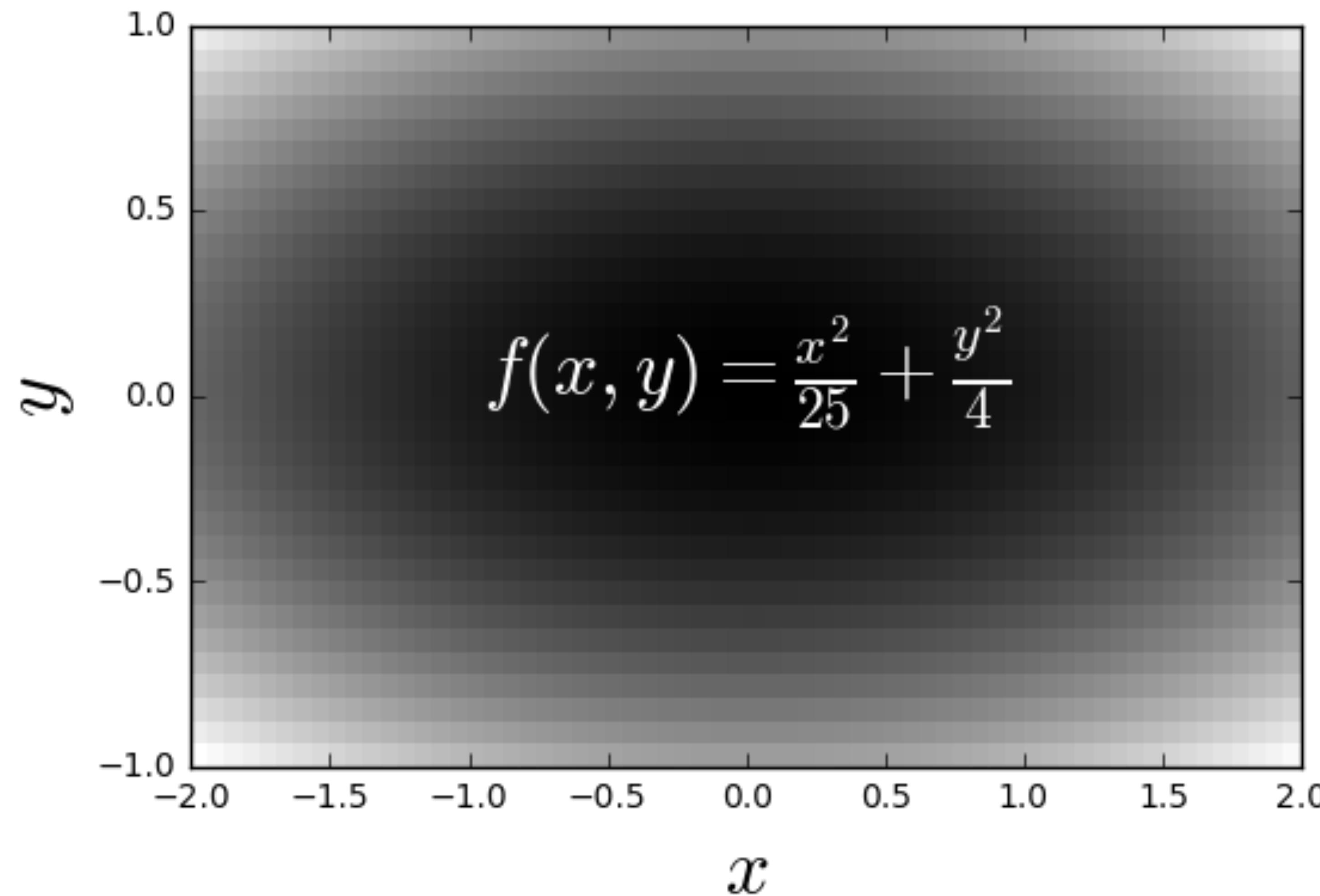
- Slicing: 1D arrays:  $A[slice]$ , 2D arrays:  $A[slice0, slice1]$
- Slicing:  $slice = start:stop:stride$ 
  - Indexes from *start* to *stop-1* in steps of *stride*
  - Missing *start*: implicitly at *beginning* of array
  - Missing *stop*: implicitly at *end* of array
  - Missing *stride*: implicitly *stride 1*
- Negative indexes/slices: count from *end of array*

# 2D arrays & images

0.434	0.339	0.337	0.367	...
0.434	0.421	0.404	0.395	...
0.350	0.388	0.340	0.340	...
0.328	0.384	0.308	0.308	...
...	...	...	...	...



# 2D arrays & functions





# Using meshgrid()



meshgrids.py

```
import numpy as np
```

```
u = np.linspace(-2, 2, 3)
```

```
v = np.linspace(-1, 1, 5)
```

```
X, Y = np.meshgrid(u, v)
```

```
Z = X**2/25 + Y**2/4
```



# Using meshgrid()

X:

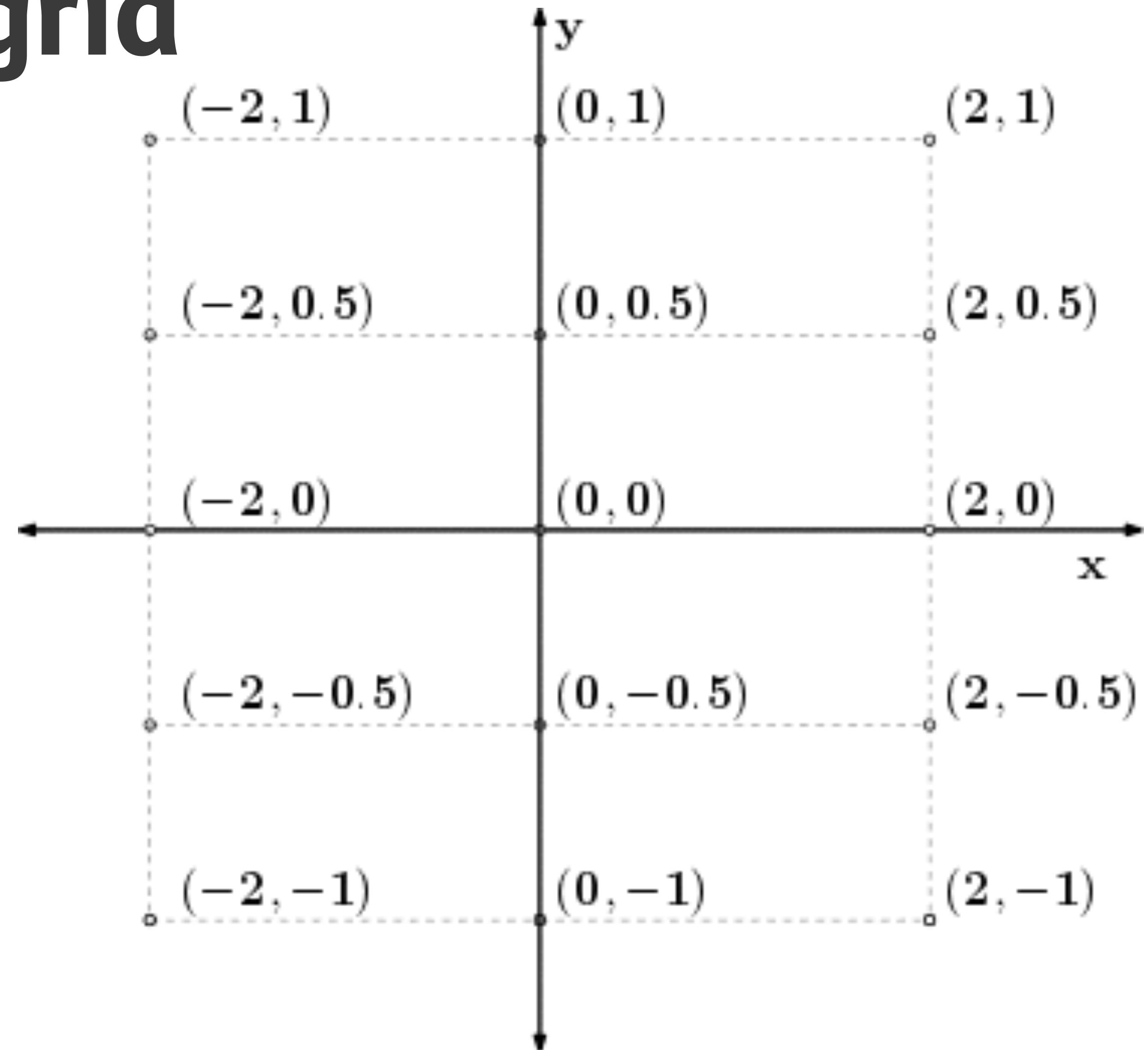
```
[[ -2.  0.  2.]
 [-2.  0.  2.]
 [-2.  0.  2.]
 [-2.  0.  2.]
 [-2.  0.  2.]]
```

Y:

```
[[ -1. -1. -1. ]
 [ -0.5 -0.5 -0.5]
 [ 0.  0.  0. ]
 [ 0.5  0.5  0.5]
 [ 1.  1.  1. ]]
```



# Meshgrid





# Sampling on a grid



meshgrids.py

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

u = np.linspace(-2, 2, 3)
v = np.linspace(-1, 1, 5)
X,Y = np.meshgrid(u, v)
Z = X**2/25 + Y**2/4
```

```
print('Z:\n', Z)
plt.set_cmap('gray')
plt.pcolor(Z)
plt.show()
```



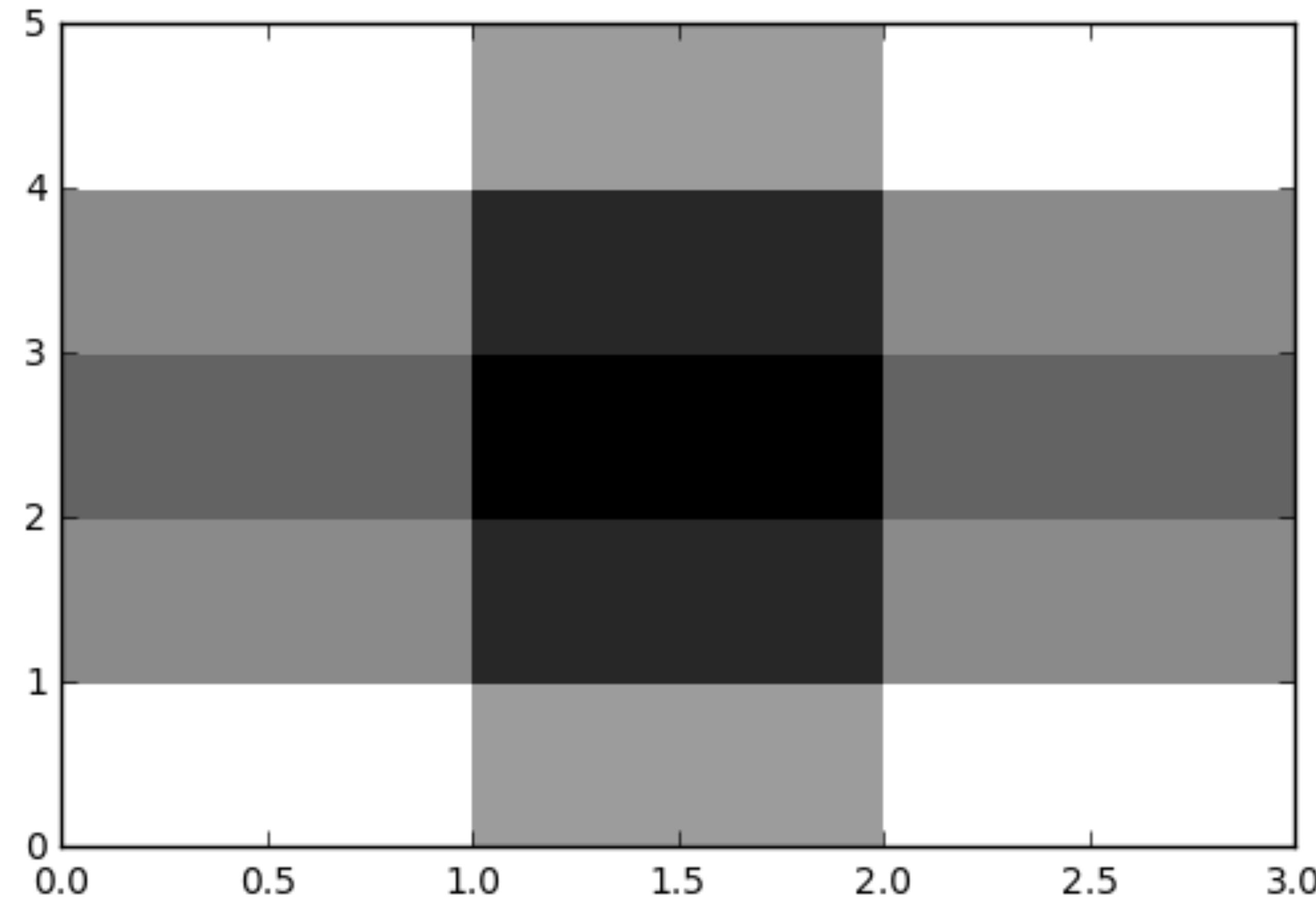
# Sampling on a grid

Z:

```
[[ 0.41      0.25      0.41      ]  
 [ 0.2225    0.0625    0.2225]  
 [ 0.16      0.        0.16      ]  
 [ 0.2225    0.0625    0.2225]  
 [ 0.41      0.25      0.41      ]]
```

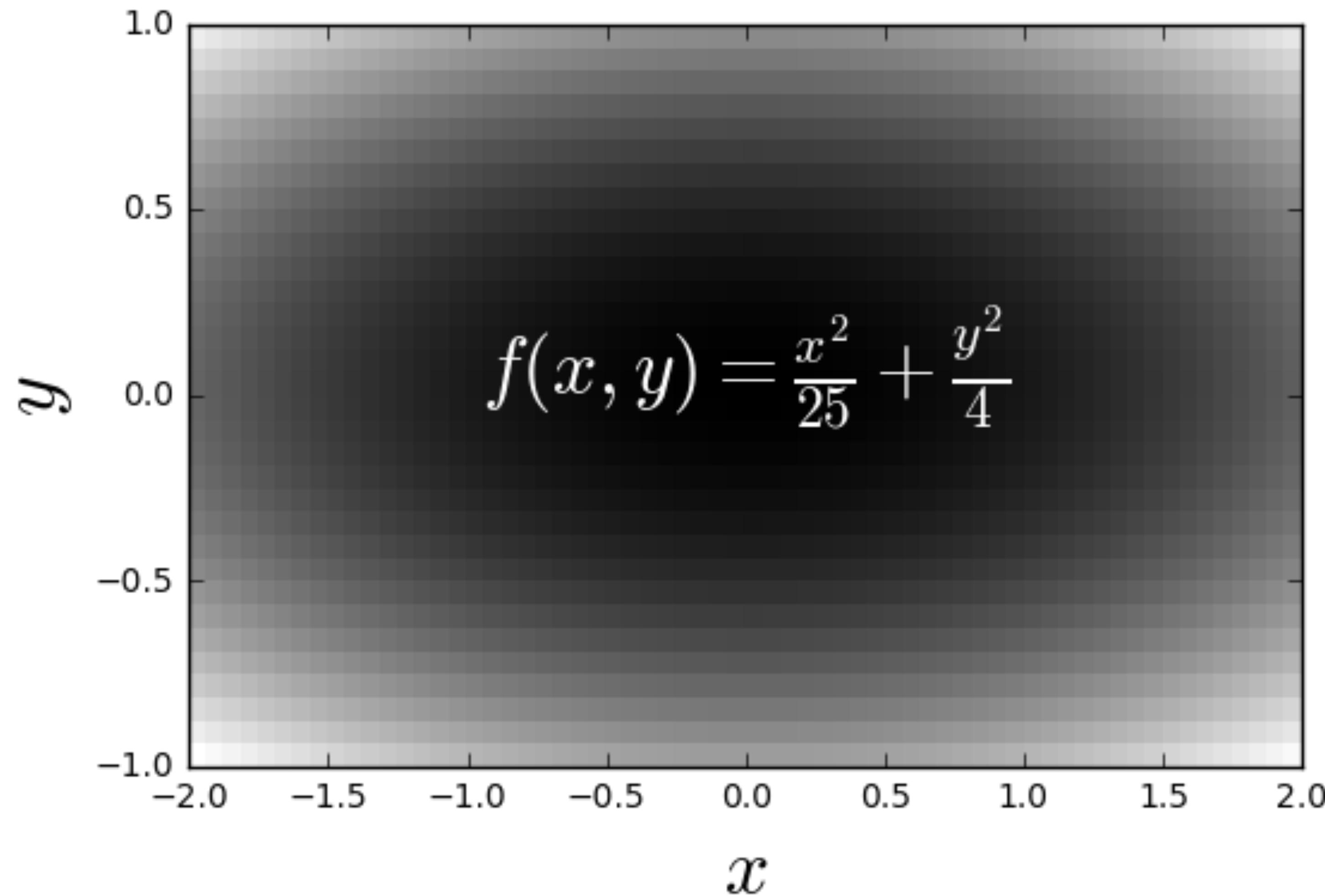


# Sampling on a grid





# Sampling on a grid





# Orientations of 2D arrays & images



orientation.py

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

Z = np.array([[1, 2, 3], [4, 5, 6]])
print('Z:\n', z)

plt.pcolor(Z)
plt.show()
```

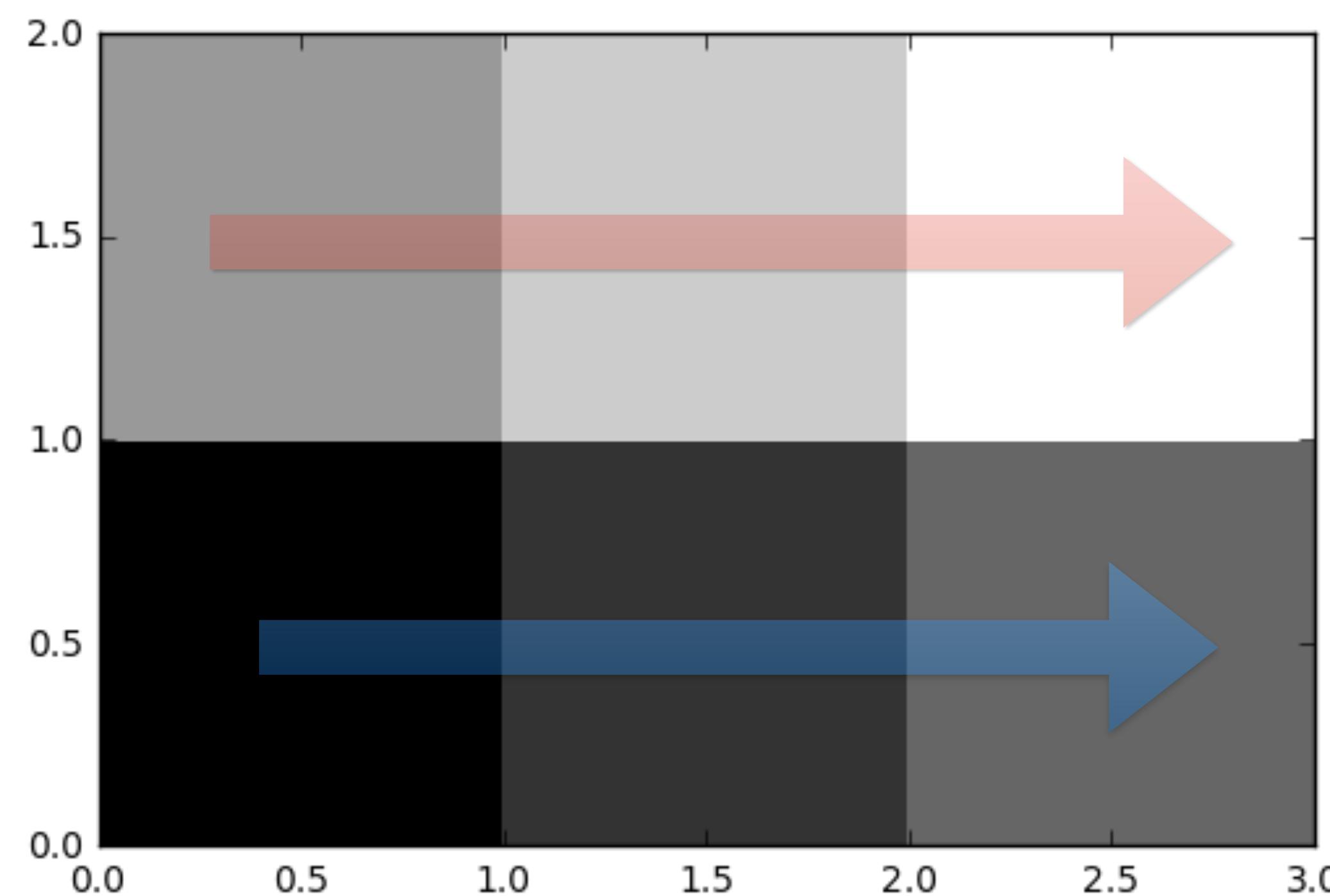


# Orientations of 2D arrays & images

Output:

Z:

```
[[1 2 3] →  
 [4 5 6]] →
```





INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

**Let's practice!**

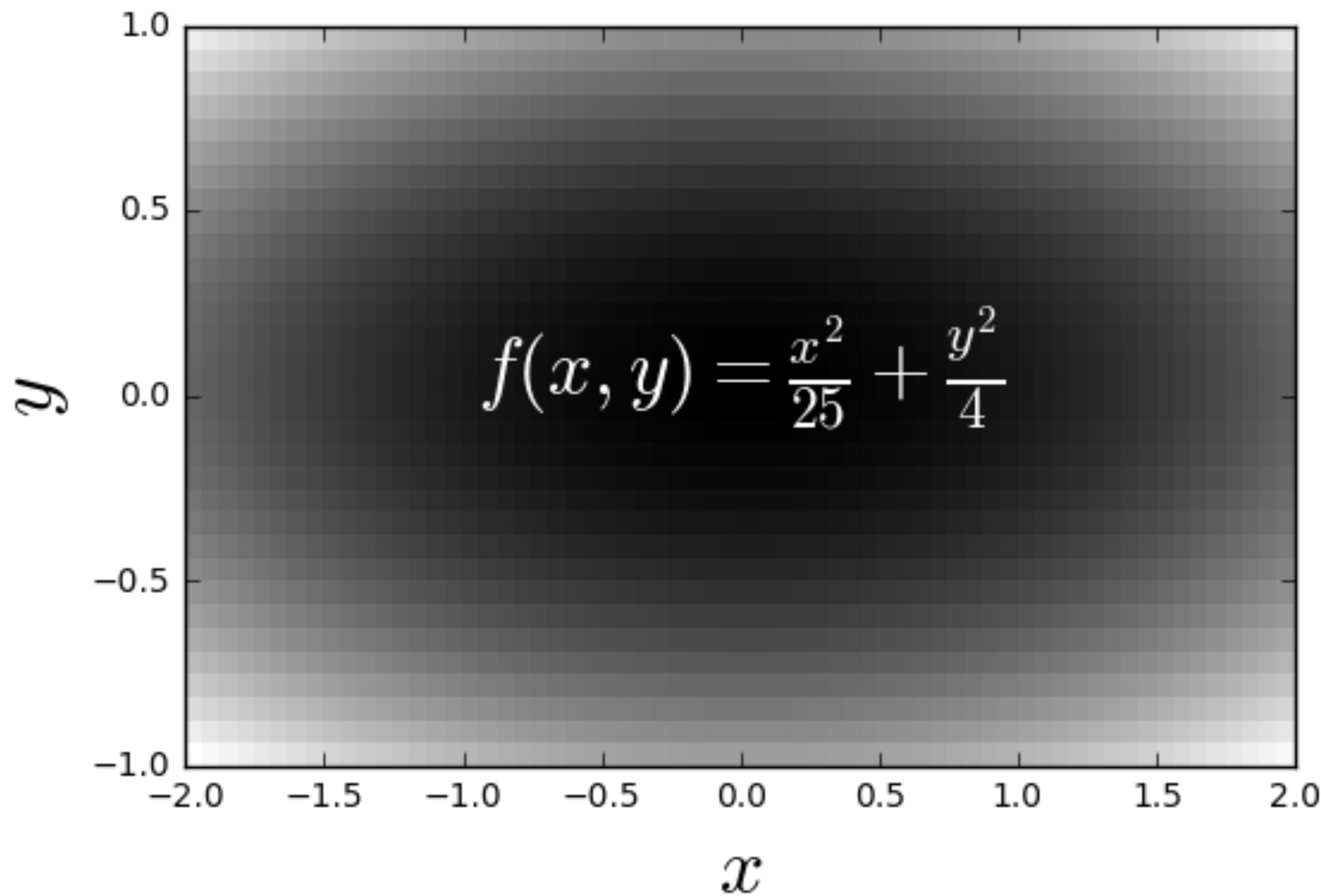


INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

# Visualizing bivariate functions



# Bivariate functions





# Pseudocolor plot

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

```
In [2]: import matplotlib.pyplot as plt
```

```
In [3]: u = np.linspace(-2, 2, 65)
```

```
In [4]: v = np.linspace(-1, 1, 33)
```

```
In [5]: X,Y = np.meshgrid(u, v)
```

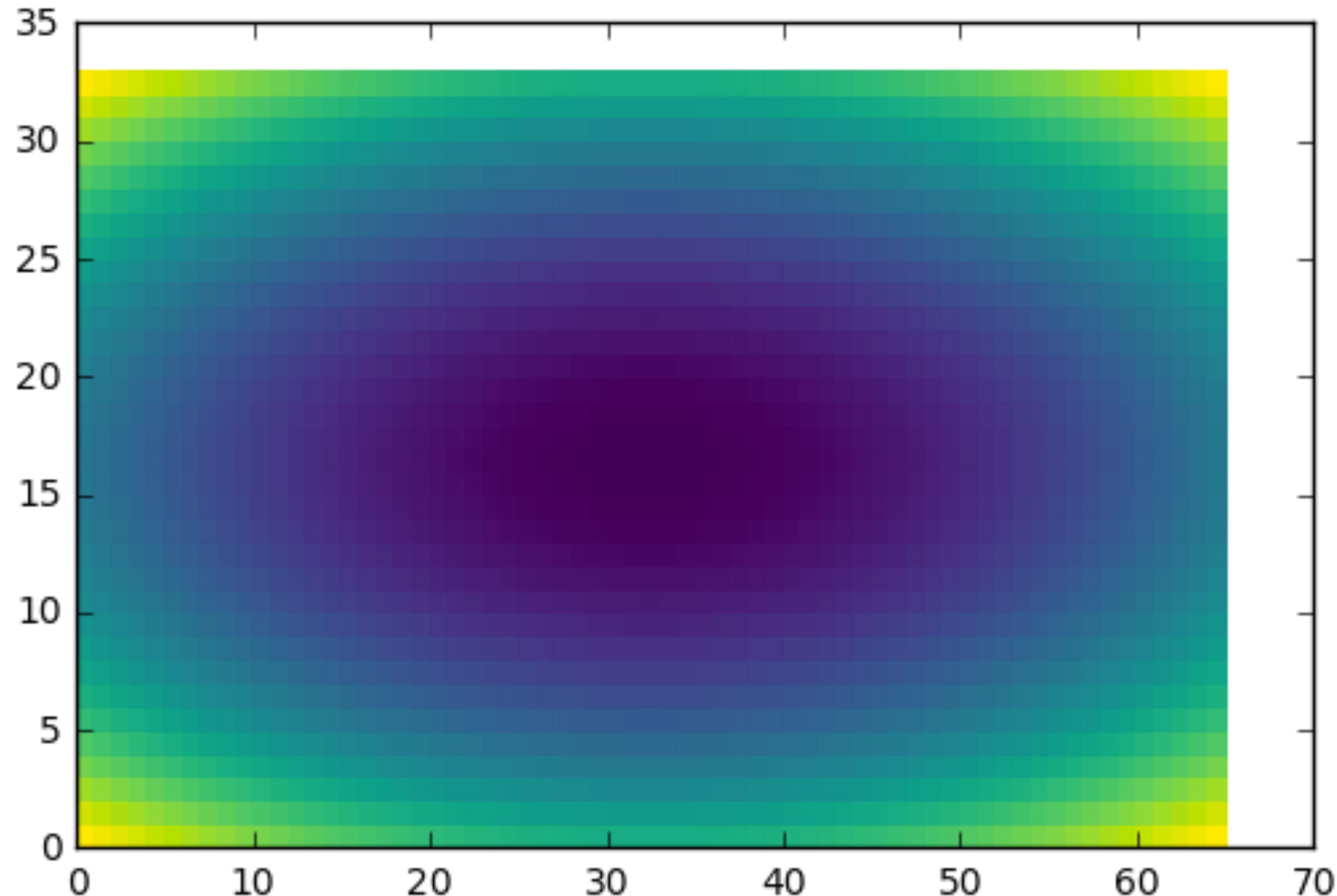
```
In [6]: Z = X**2/25 + Y**2/4
```

```
In [7]: plt.pcolor(Z)
```

```
In [8]: plt.show()
```



# Pseudocolor plot





# Color bar

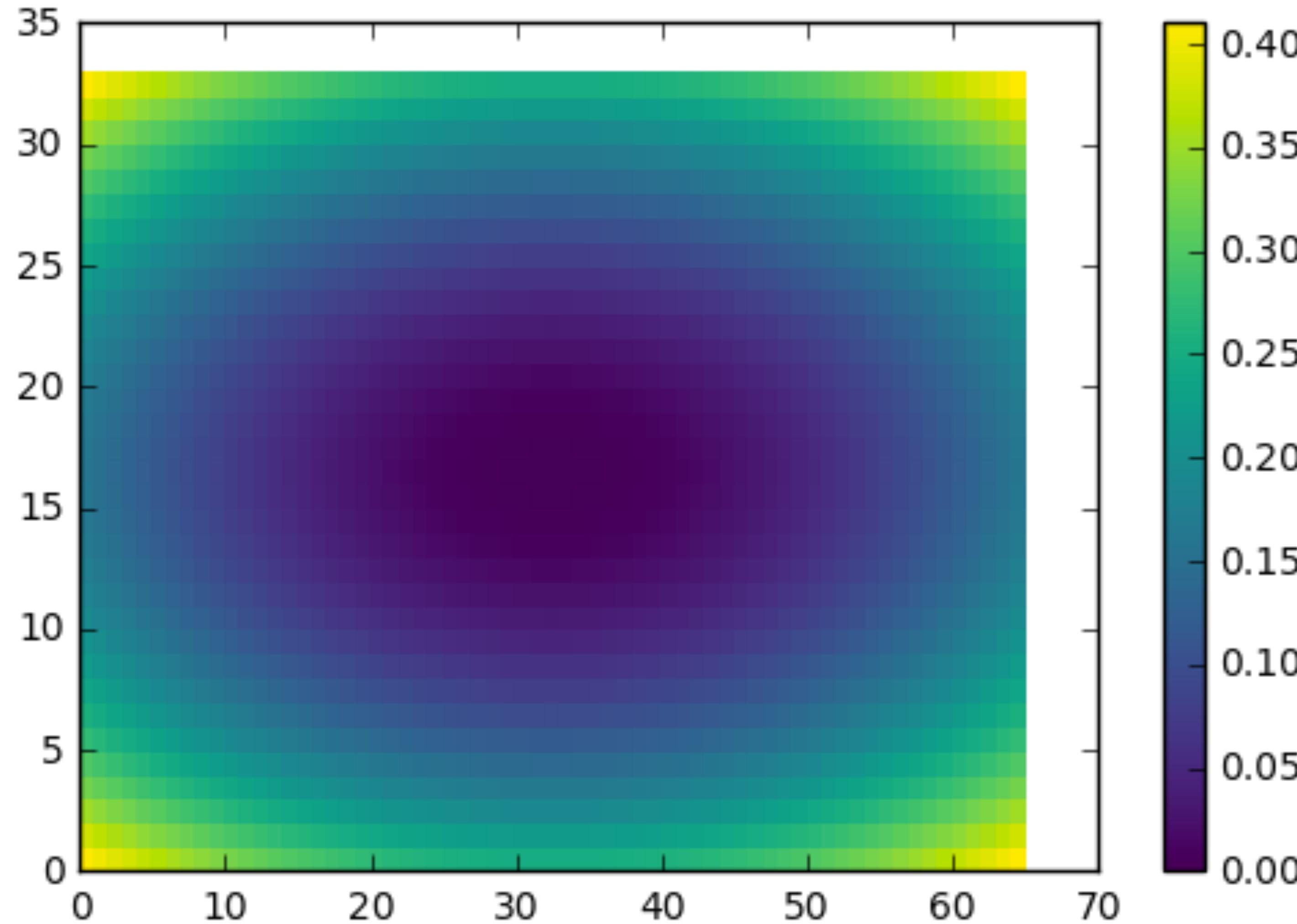
```
In [9]: plt.pcolor(Z)
```

```
In [10]: plt.colorbar()
```

```
In [11]: plt.show()
```



# Color bar





# Color map

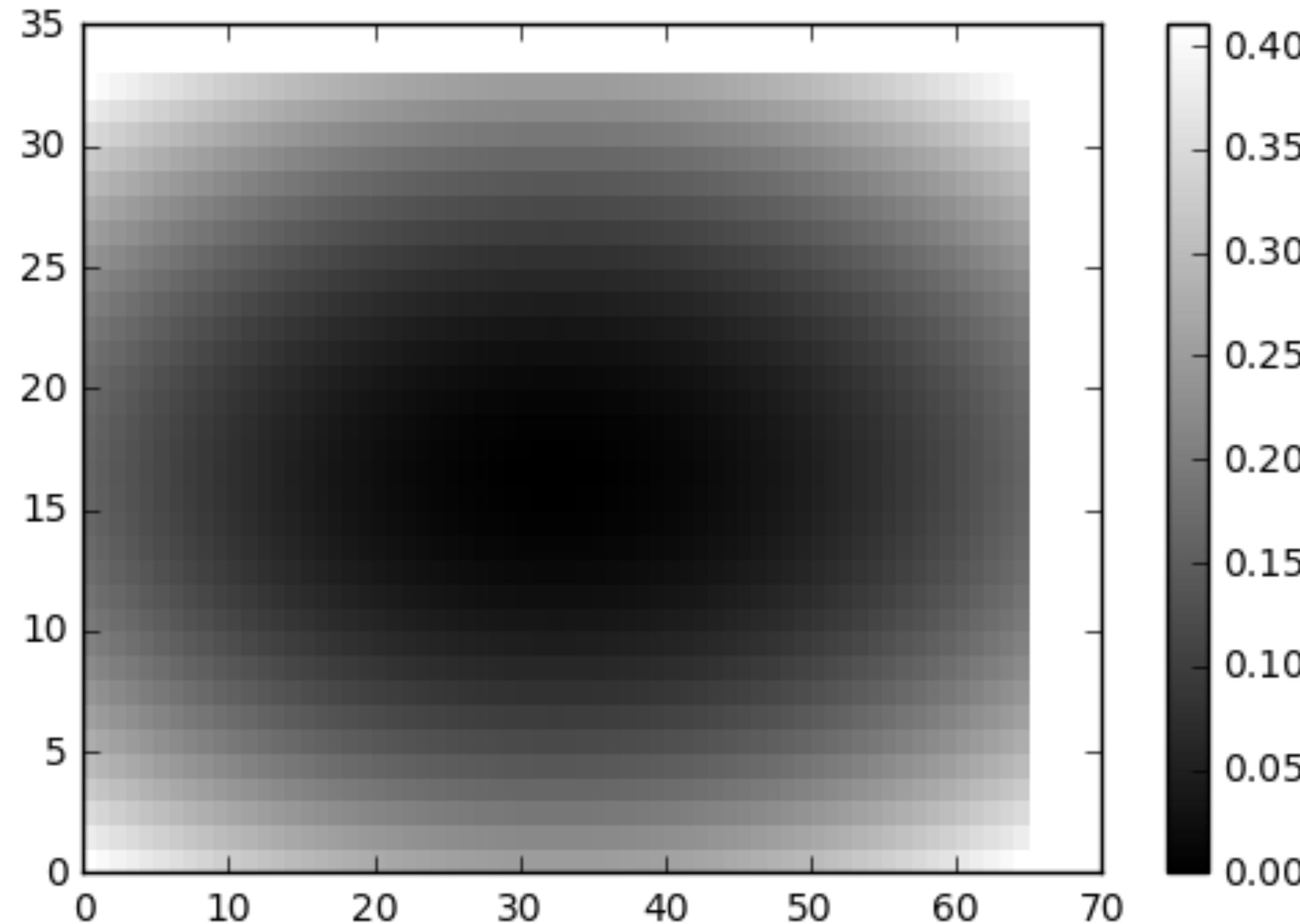
```
In [12]: plt.pcolor(Z, cmap= 'gray')
```

```
In [13]: plt.colorbar()
```

```
In [14]: plt.show()
```



# Color map





# Color map

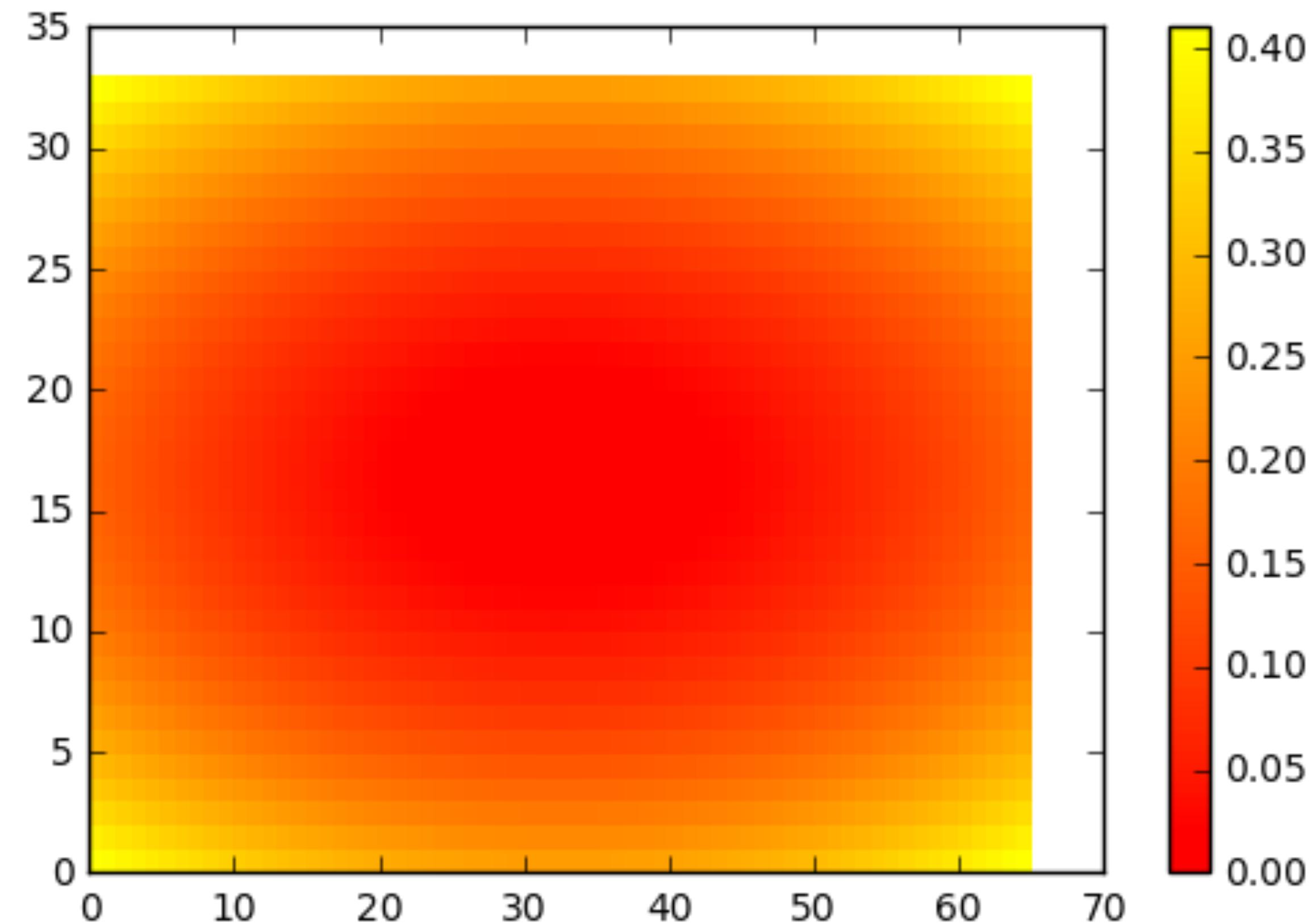
```
In [15]: plt.pcolor(Z, cmap= 'autumn')
```

```
In [16]: plt.colorbar()
```

```
In [17]: plt.show()
```



# Color map





# Axis tight

```
In [18]: plt.pcolor(Z)
```

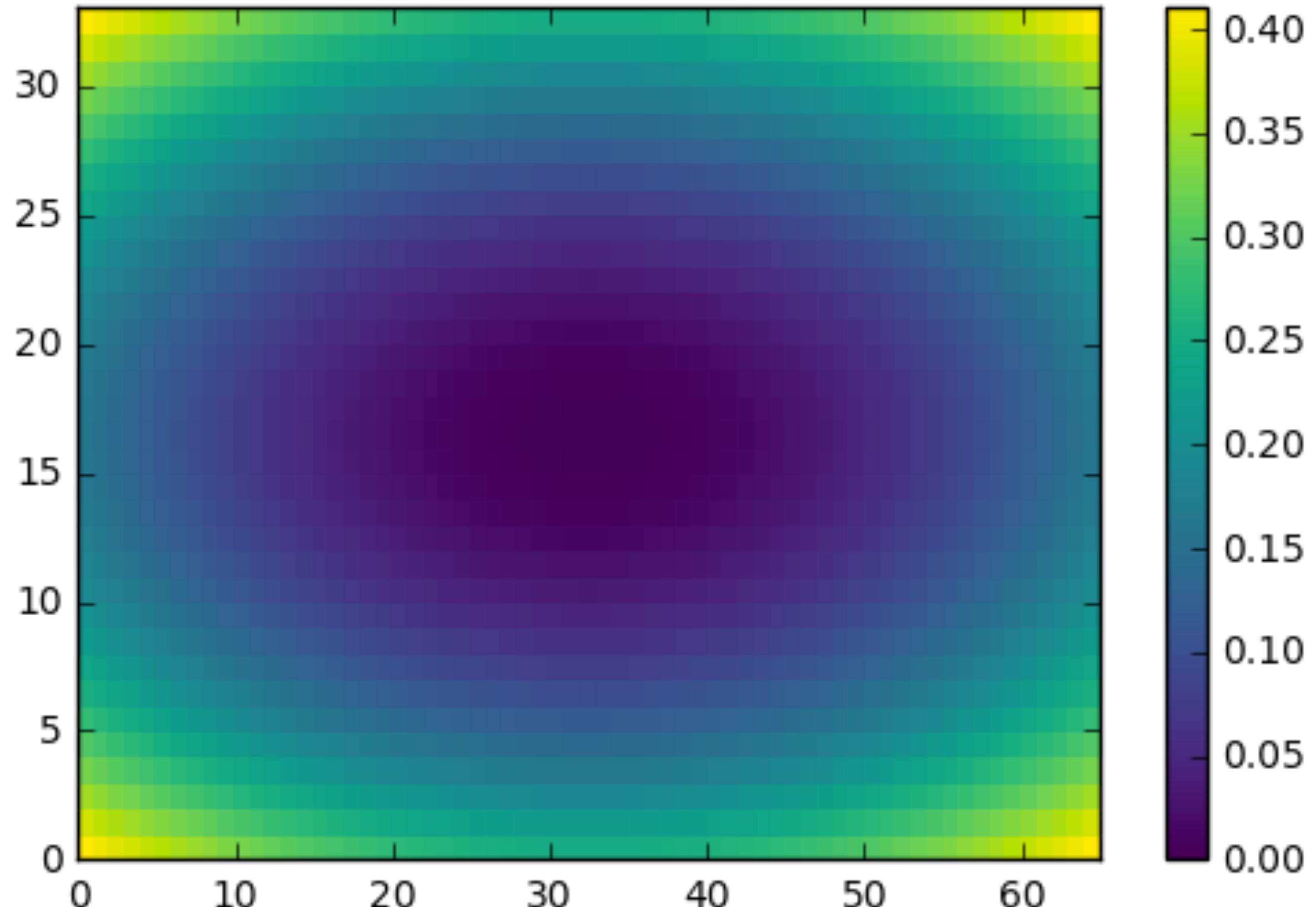
```
In [19]: plt.colorbar()
```

```
In [20]: plt.axis('tight')
```

```
In [21]: plt.show()
```



# Axis tight





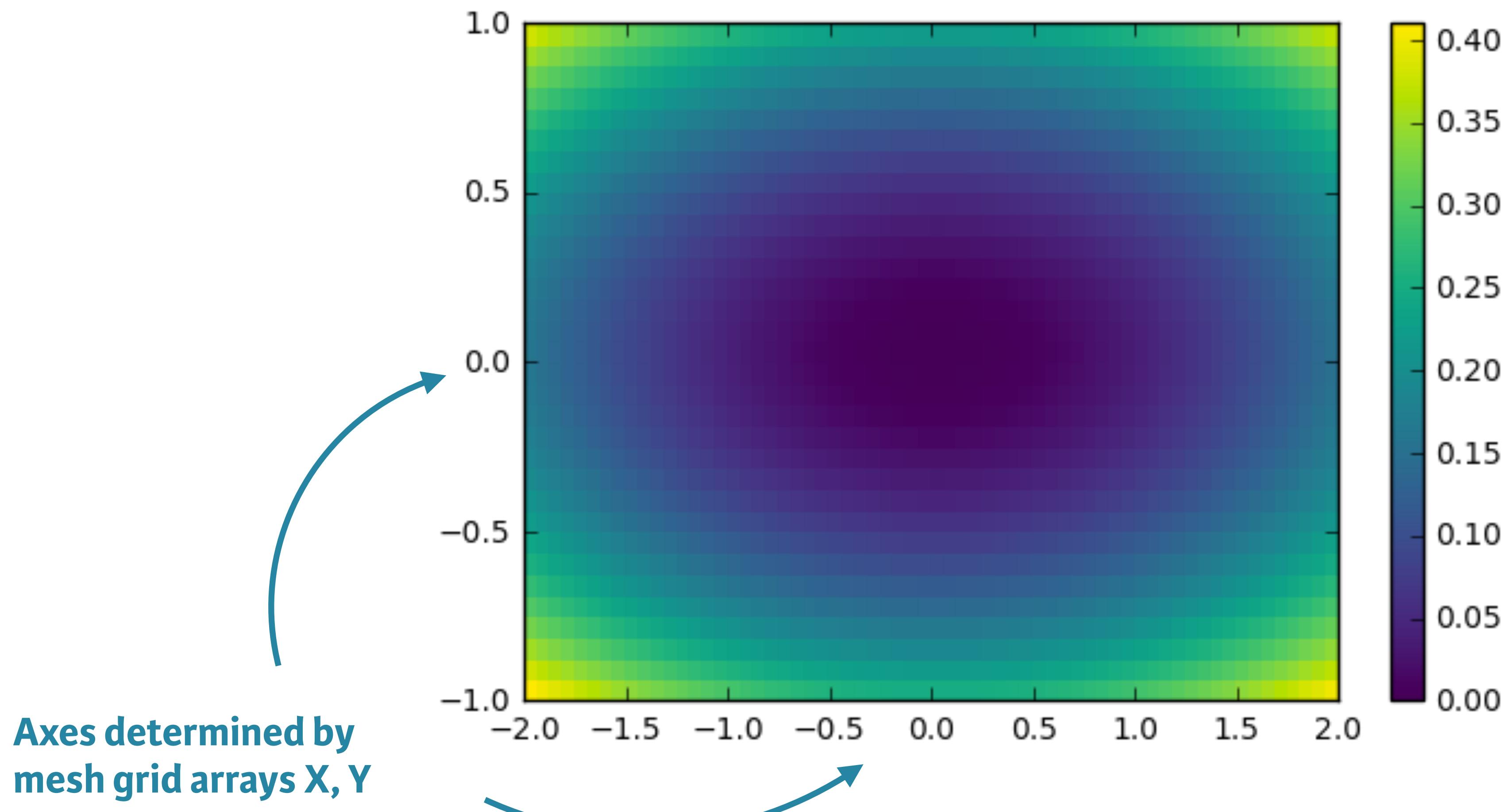
# Plot using mesh grid

```
In [22]: plt.pcolor(X, Y, Z) # X, Y are 2D meshgrid
```

```
In [23]: plt.colorbar()
```

```
In [24]: plt.show()
```

# Plot using mesh grid





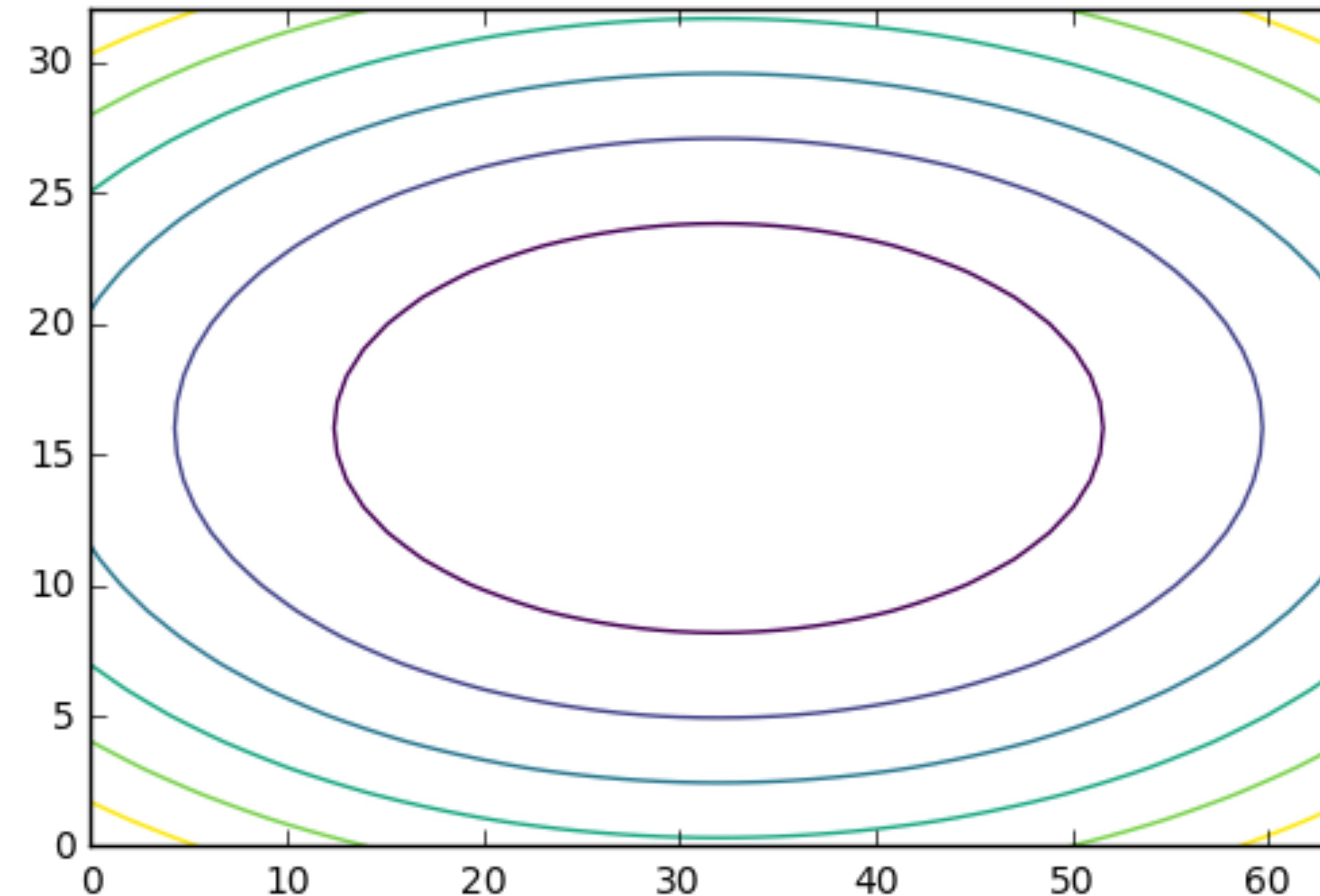
# Contour plots

```
In [25]: plt.contour(Z)
```

```
In [26]: plt.show()
```



# Contour plots





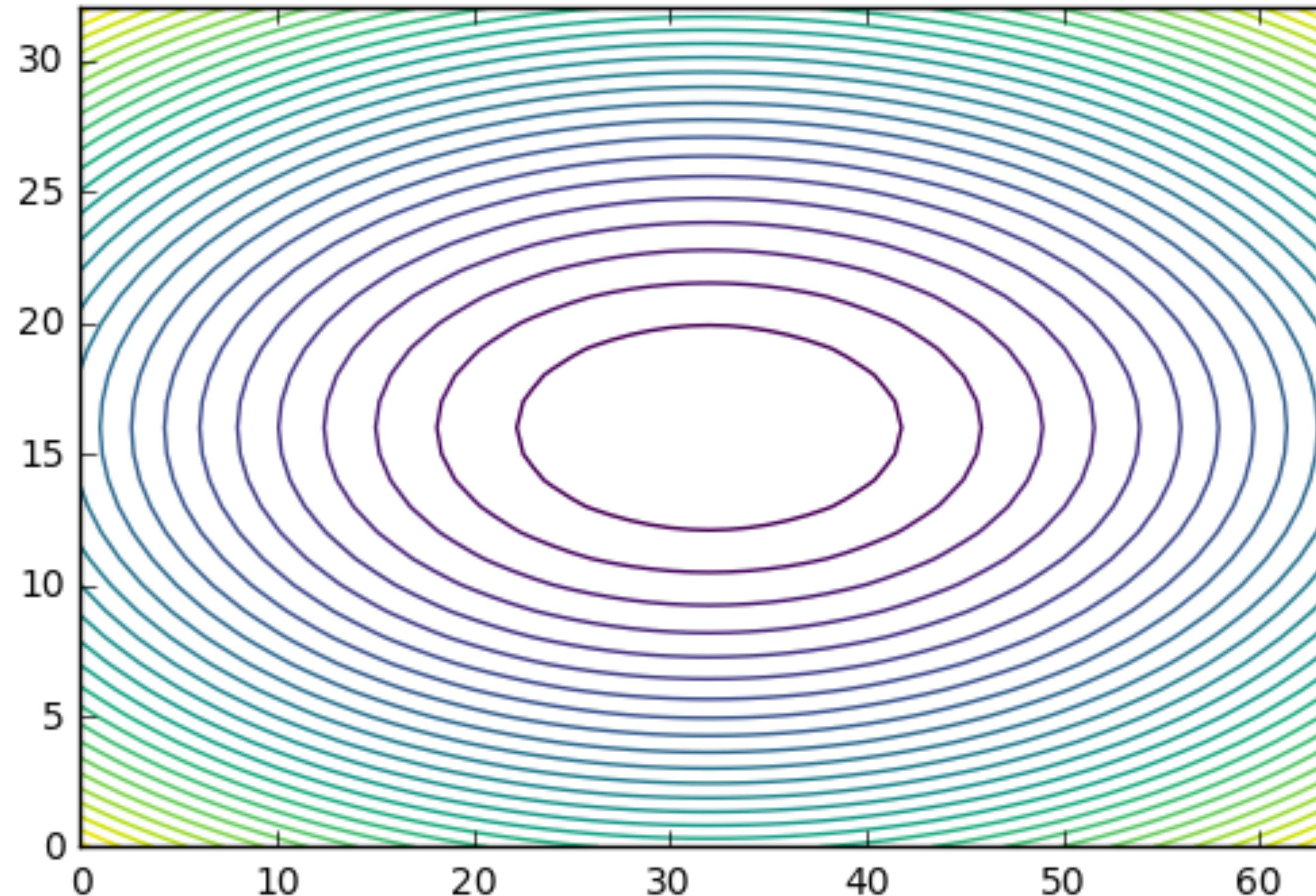
# More contours

```
In [27]: plt.contour(Z, 30)
```

```
In [28]: plt.show()
```



# More contours





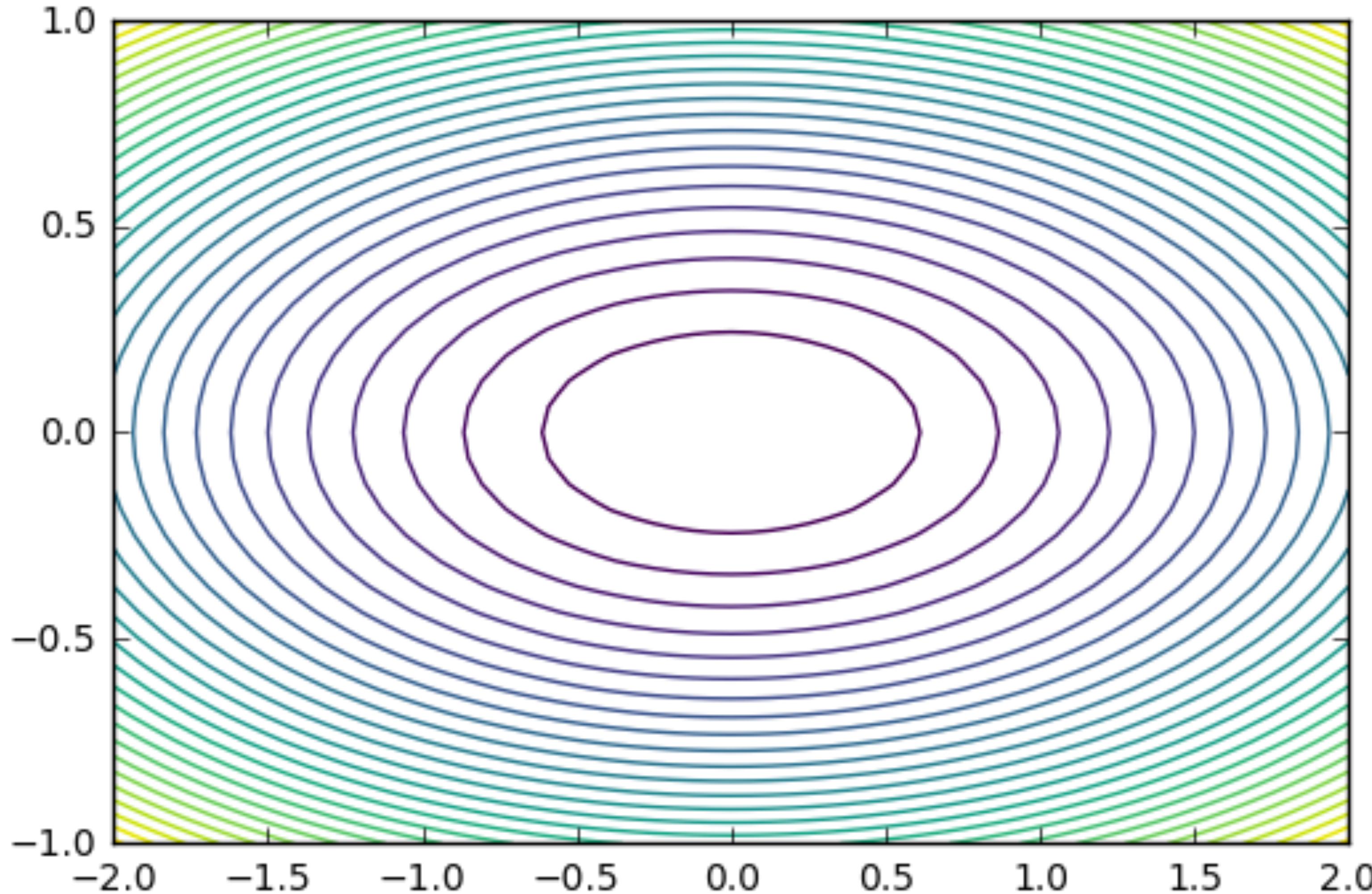
# Contour plot using meshgrid

```
In [29]: plt.contour(X, Y, Z, 30)
```

```
In [30]: plt.show()
```



# Contour plot using meshgrid





# Filled contour plots

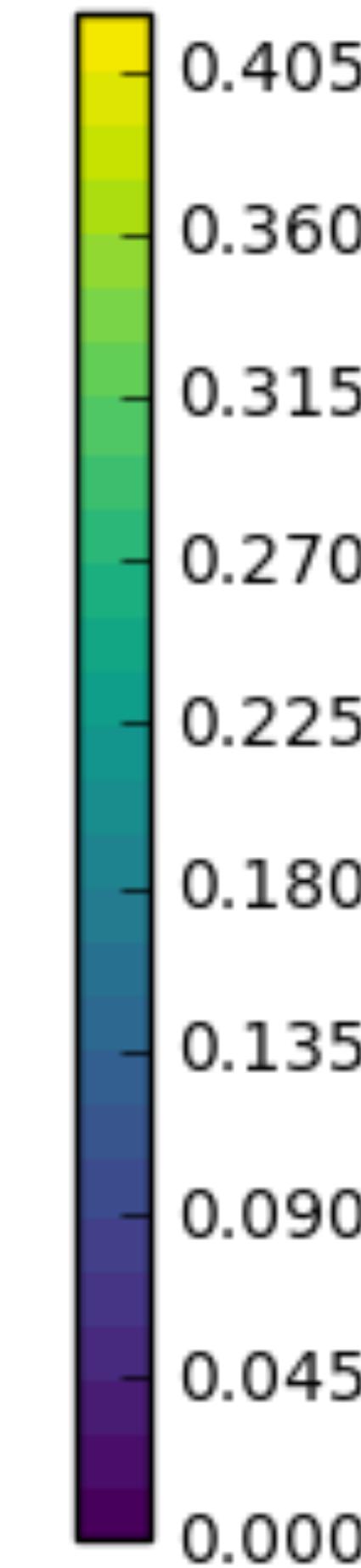
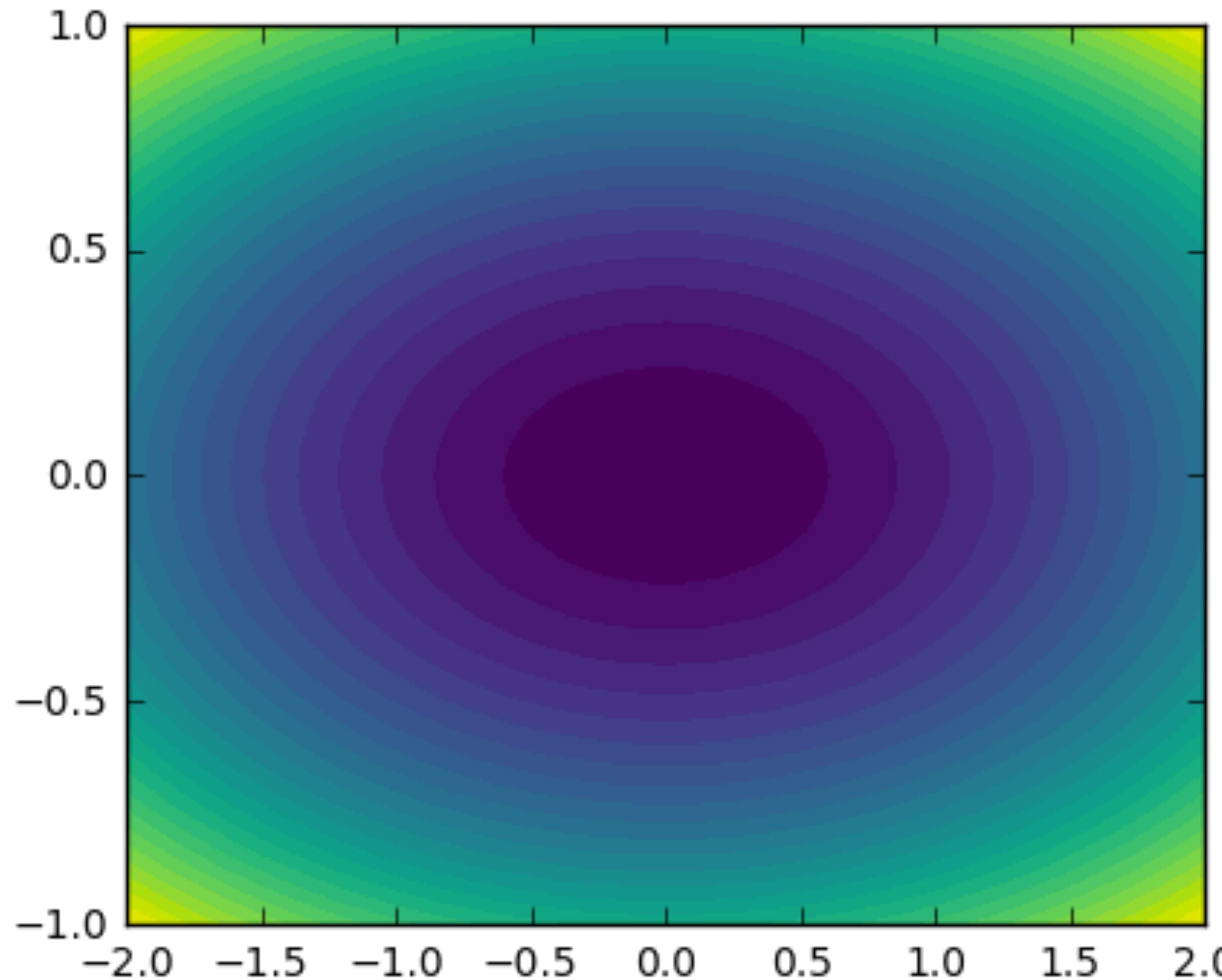
```
In [31]: plt.contourf(X, Y, Z, 30)
```

```
In [32]: plt.colorbar()
```

```
In [33]: plt.show()
```



# Filled contour plots





# More information

- API has many (optional) keyword arguments
- More in `matplotlib.pyplot` documentation
- More examples: <http://matplotlib.org/gallery.html>



INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

**Let's practice!**

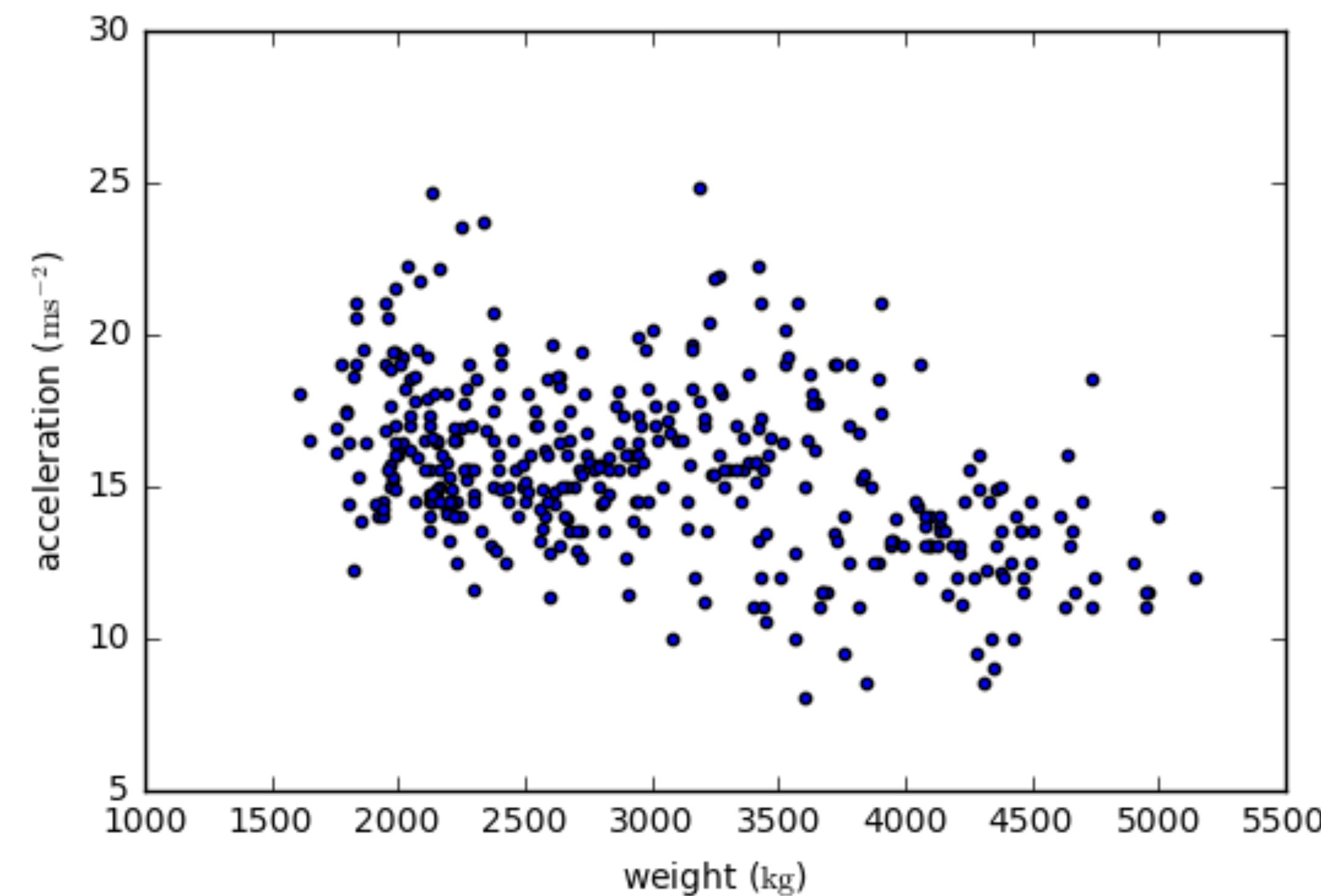


INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

# Visualizing bivariate distributions

# Distributions of 2D points

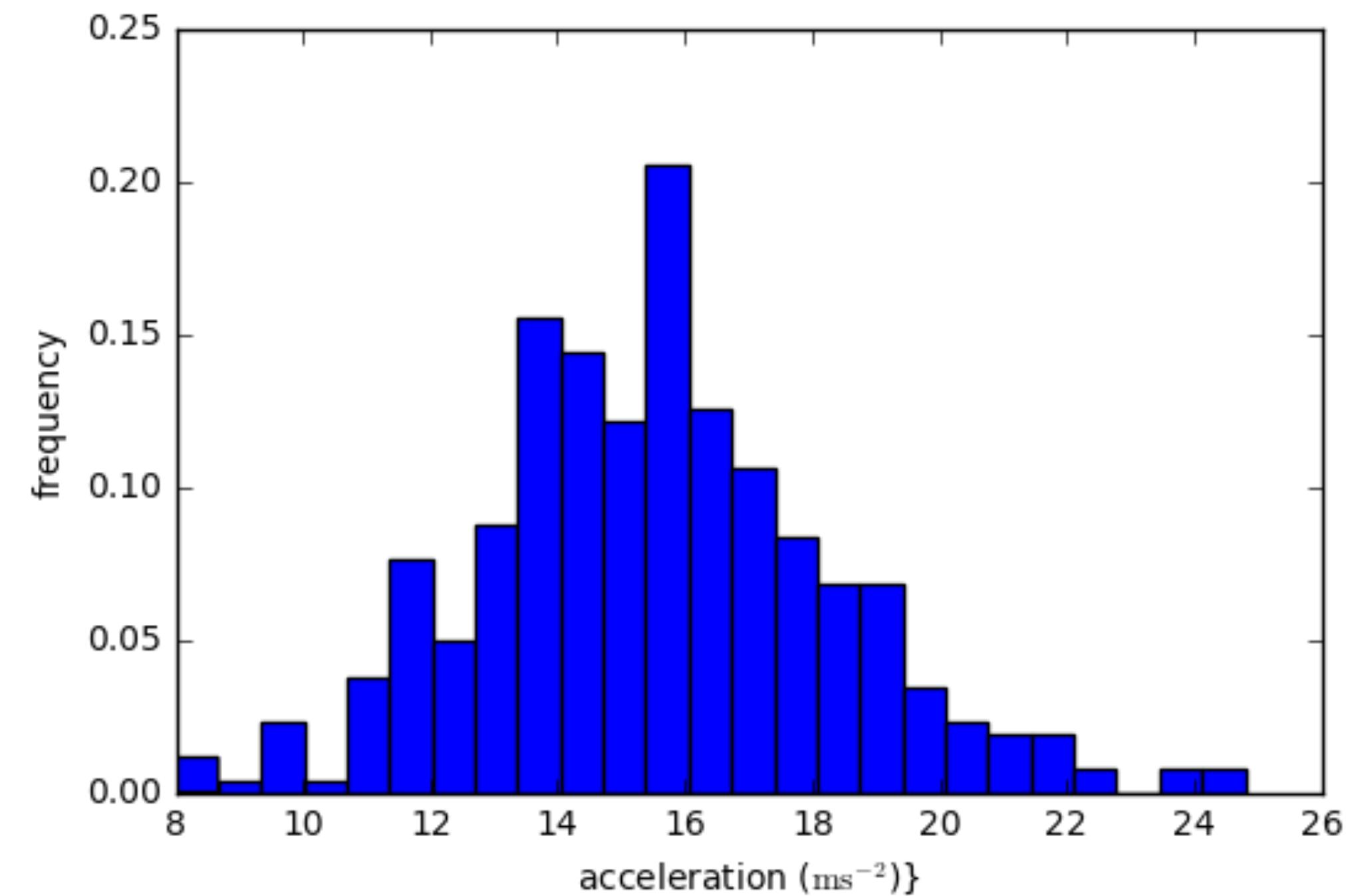
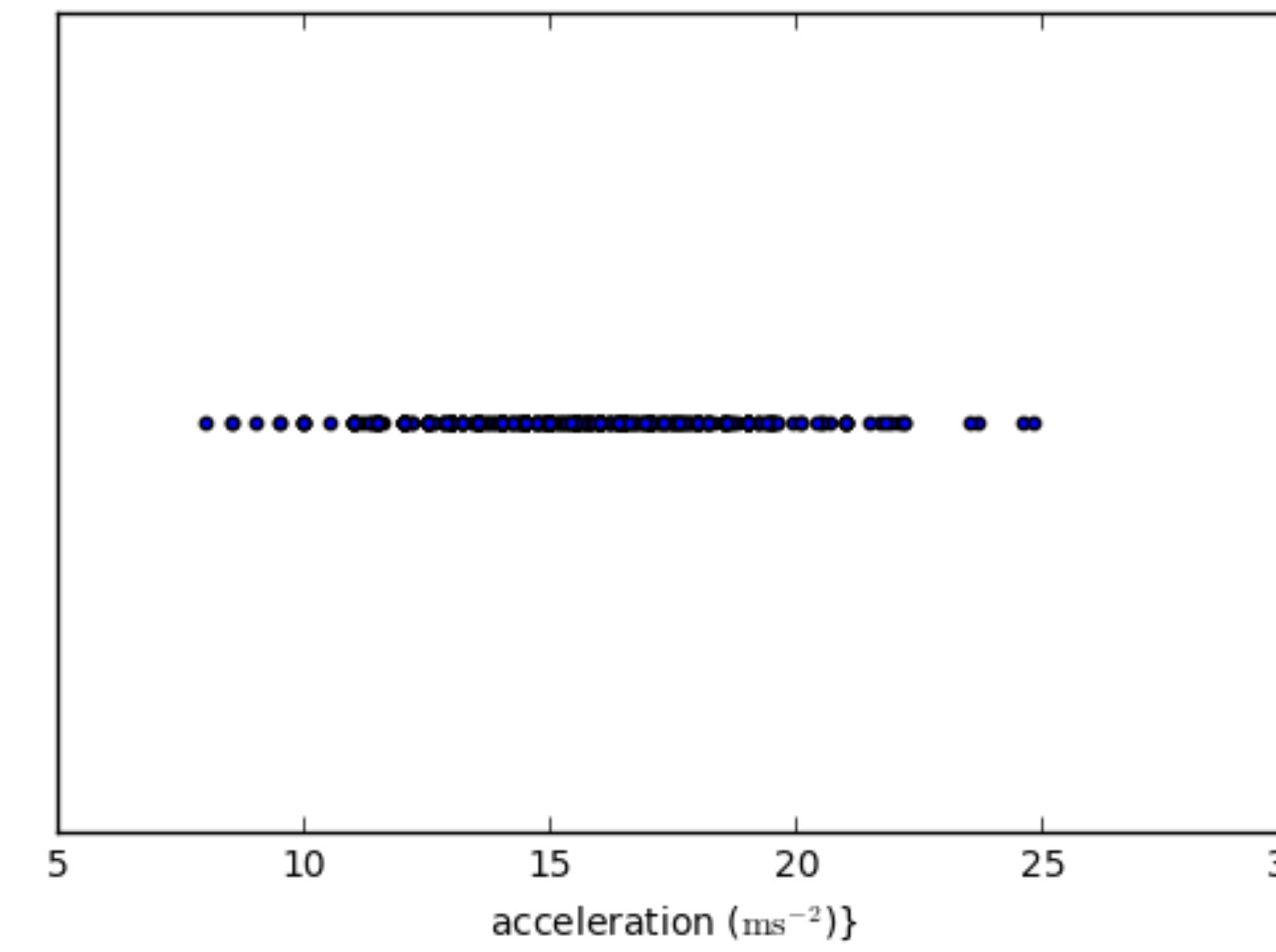
- 2D points given as two 1D arrays  $x$  &  $y$
- Goal: generate a 2D histogram from  $x$  &  $y$





# Histograms in 1D

- Choose bins (intervals)
- Count realizations within bins & plot





# Histograms in 1D

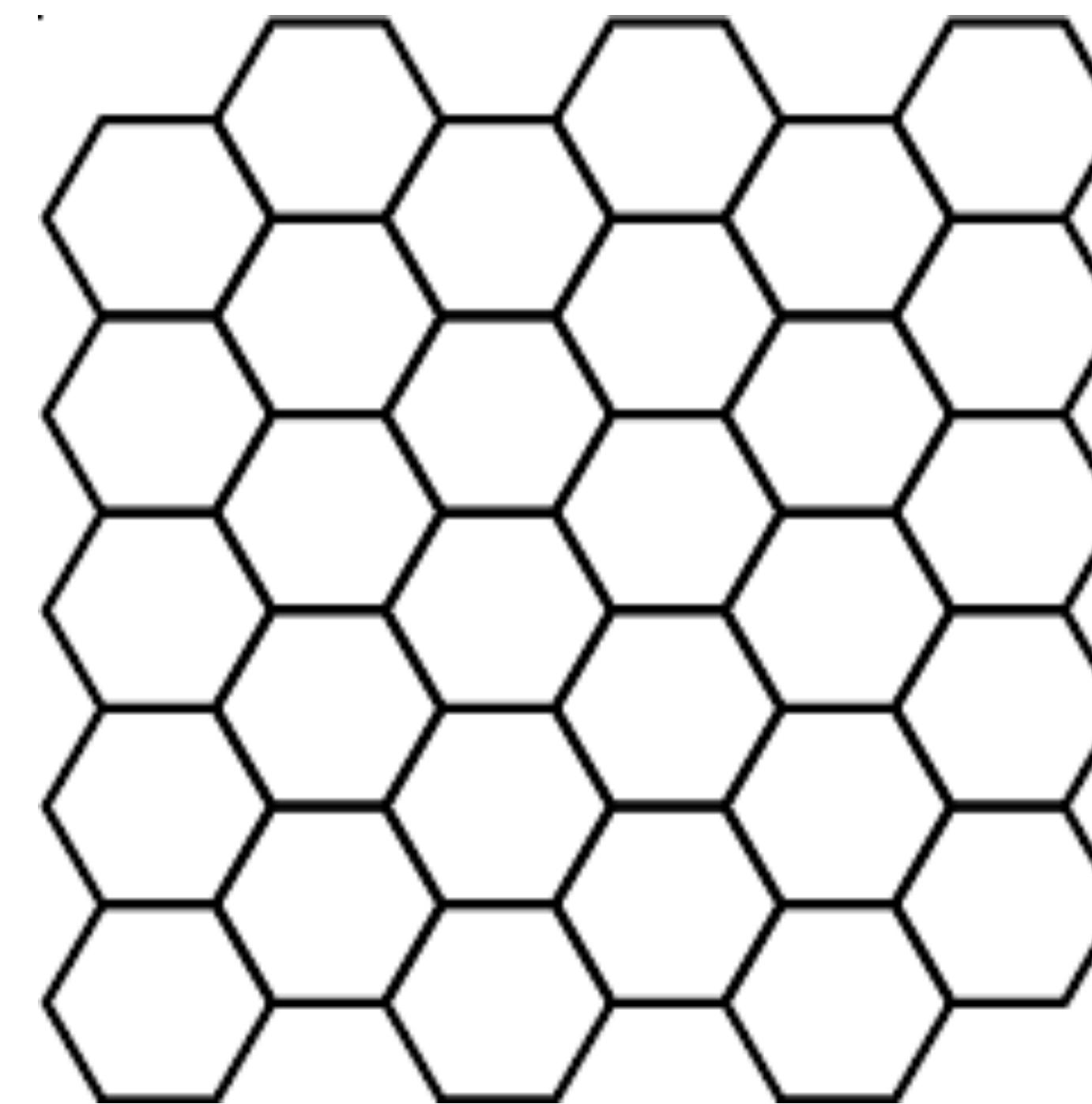
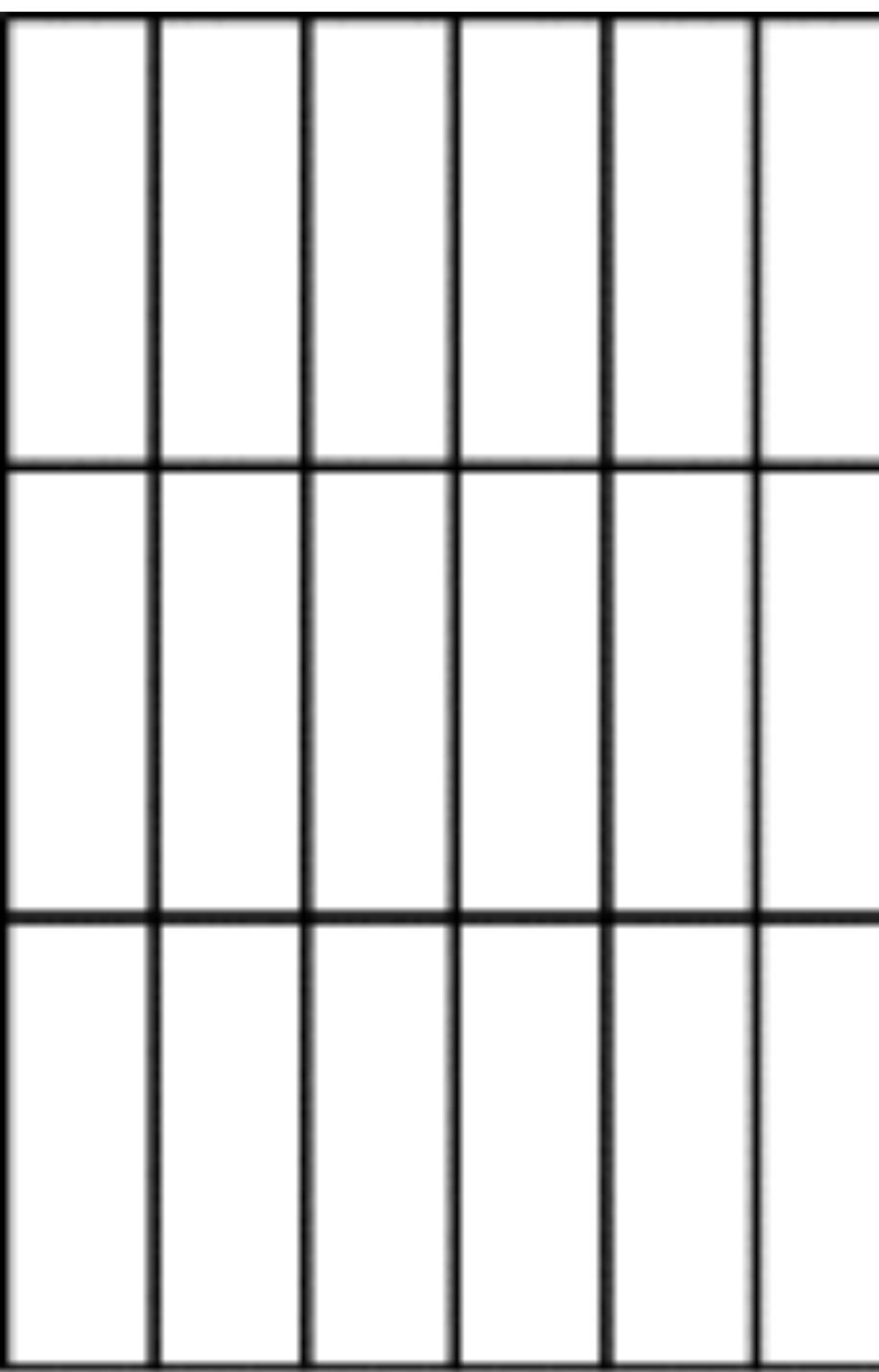
```
In [1]: counts, bins, patches = plt.hist(x, bins=25)
```

```
In [2]: plt.show()
```



# Bins in 2D

- Different shapes available for binning points
- Common choices: rectangles & hexagons

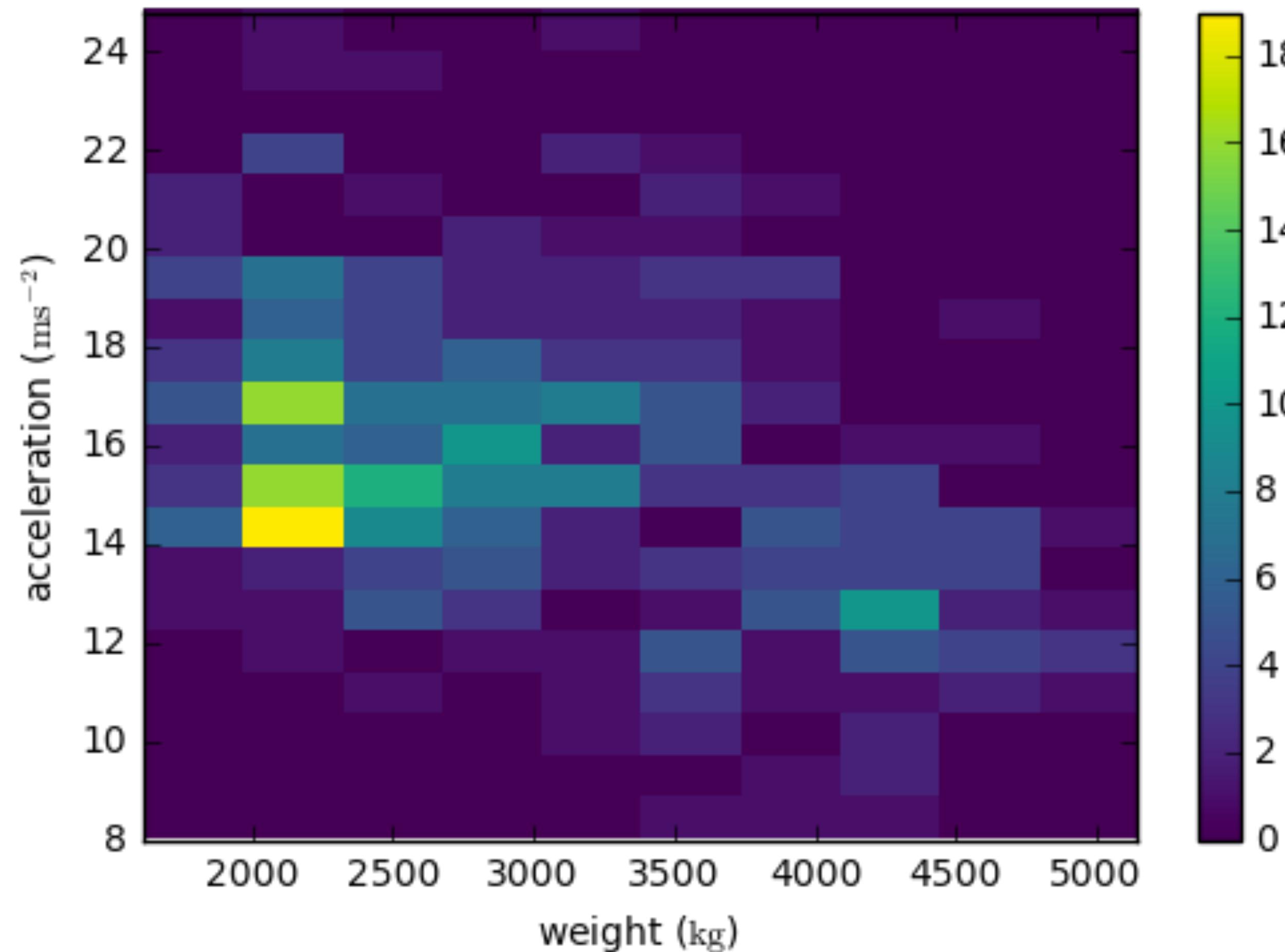




# hist2d(): Rectangular binning

```
In [1]: plt.hist2d(x, y, bins=(10, 20)) # x & y are 1D arrays of  
...: same length  
  
In [2]: plt.colorbar()  
  
In [3]: plt.xlabel('weight ($\mathsf{kg}$)')  
  
In [4]: plt.ylabel('acceleration ($\mathsf{m}\mathsf{s}^{-2}$)')  
  
In [5]: plt.show()
```

# hist2d(): Rectangular binning





# hexbin(): Hexagonal binning

```
In [1]: plt.hexbin(x, y, gridsize=(15,10))
```

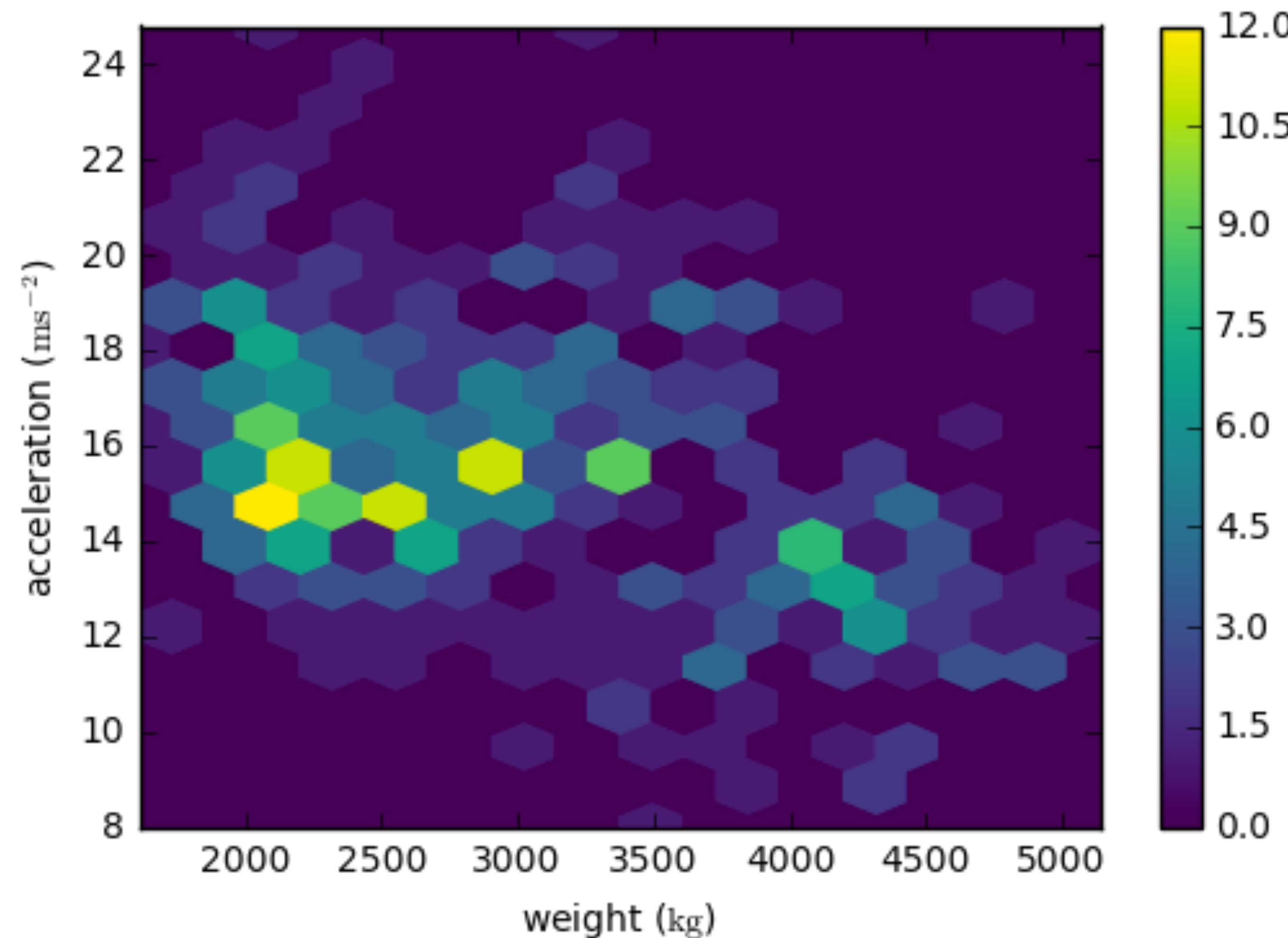
```
In [2]: plt.colorbar()
```

```
In [3]: plt.xlabel('weight ($\mathsf{kg}$)')
```

```
In [4]: plt.ylabel('acceleration ($\mathsf{m s}^{-2}$)')
```

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

# hexbin(): Hexagonal binning





INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

**Let's practice!**



INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

# Working with images



# Images

- Grayscale images: rectangular 2D arrays
- Color images: typically three 2D arrays (channels)
  - RGB (Red-Green-Blue)
- Channel values:
  - 0 to 1 (floating-point numbers)
  - 0 to 255 (8 bit integers)



# Loading images

```
In [1]: img = plt.imread('sunflower.jpg')
```

```
In [2]: print(img.shape)  
(480, 640, 3)
```

```
In [3]: plt.imshow(img)
```

```
In [4]: plt.axis('off')
```

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



# Loading images





# Reduction to gray-scale image

```
In [6]: collapsed = img.mean(axis=2)
```

```
In [7]: print(collapsed.shape)
(480, 640)
```

```
In [8]: plt.set_cmap('gray')
```

```
In [9]: plt.imshow(collapsed, cmap='gray')
```

```
In [10]: plt.axis('off')
```

```
In [11]: plt.show()
```



# Reduction to gray-scale image





# Uneven samples

```
In [12]: uneven = collapsed[::-4, ::2] # nonuniform subsampling
```

```
In [13]: print(uneven.shape)
(120, 320)
```

```
In [14]: plt.imshow(uneven)
```

```
In [15]: plt.axis('off')
```

```
In [16]: plt.show()
```



# Uneven samples





# Adjusting aspect ratio

```
In [17]: plt.imshow(uneven, aspect=2.0)
```

```
In [18]: plt.axis('off')
```

```
In [19]: plt.show()
```



# Adjusting aspect ratio





# Adjusting extent

```
In [20]: plt.imshow(uneven, cmap='gray', extent=(0,640,0,480))
```

```
In [21]: plt.axis('off')
```

```
In [22]: plt.show()
```



# Adjusting extent





INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

**Let's practice!**



INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

# Visualizing Regressions



# Seaborn



Seaborn is a Python visualization library based on matplotlib. It provides a high-level interface for drawing attractive statistical graphics.

[Documentation](#)

[Features](#)

<https://stanford.edu/~mwaskom/software/seaborn/>



# Recap: Pandas DataFrames

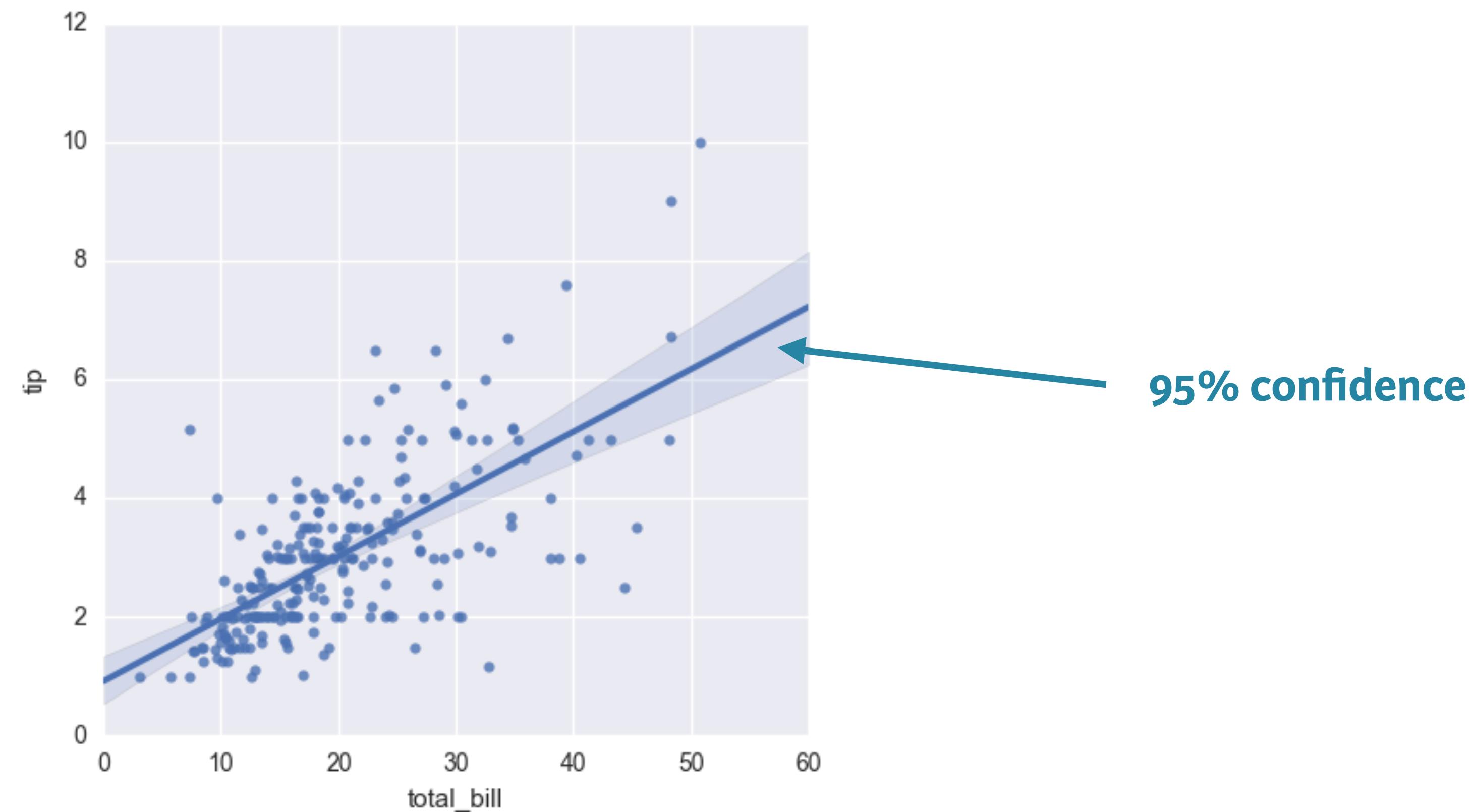
- Labelled tabular data structure
- Labels on rows: *index*
- Labels on columns: *columns*
- Columns are Pandas *Series*



# Recap: Pandas DataFrames



# Linear regression plots





# Using lmplot()

```
In [1]: import pandas as pd
```

```
In [2]: import matplotlib.pyplot as plt
```

```
In [3]: import seaborn as sns
```

```
In [4]: tips =sns.load_dataset('tips')
```

```
In [5]: sns.lmplot(x= 'total_bill', y='tip', data=tips)
```

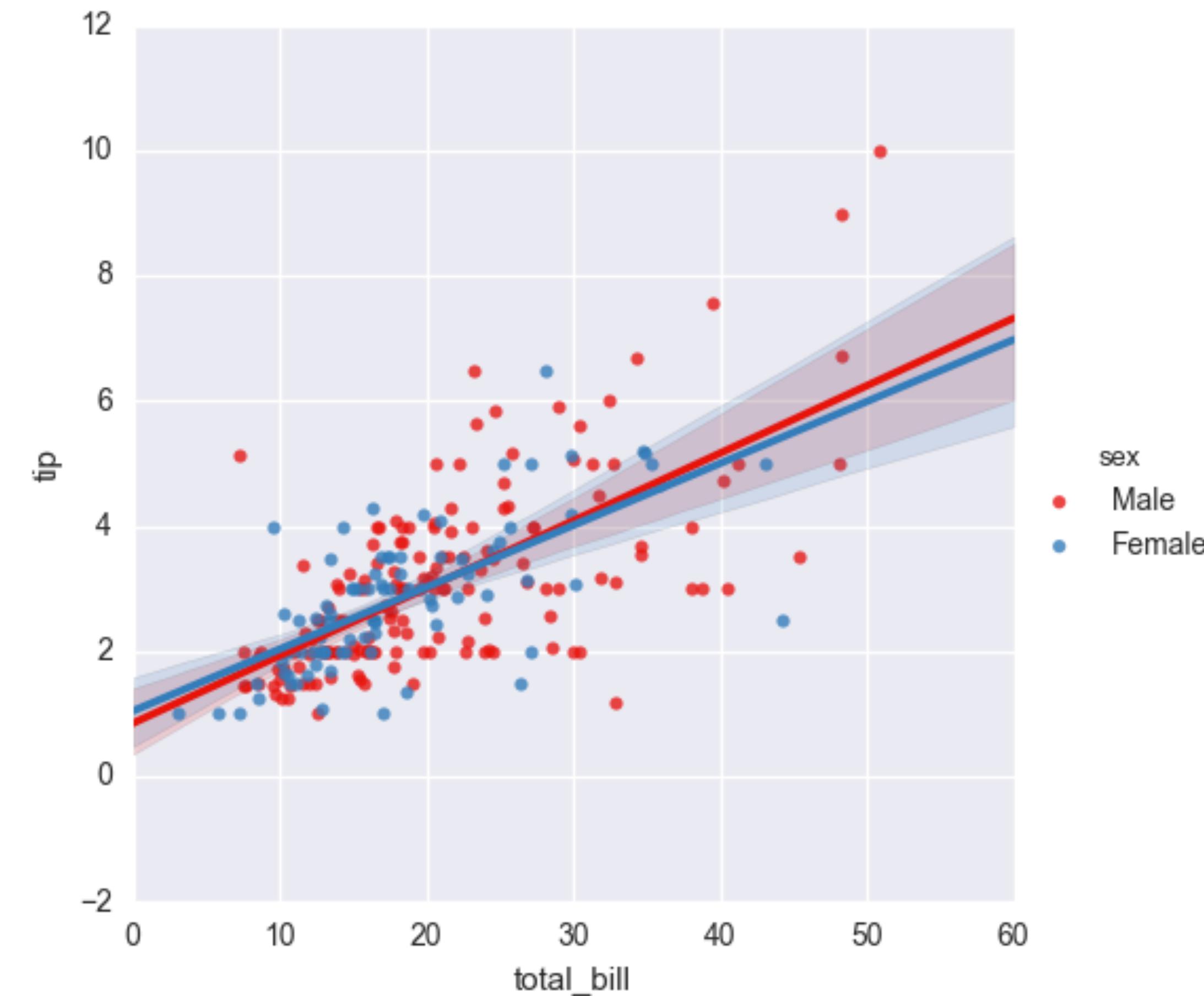
```
In [6]: plt.show()
```



# Factors



# Grouping factors (same plot)





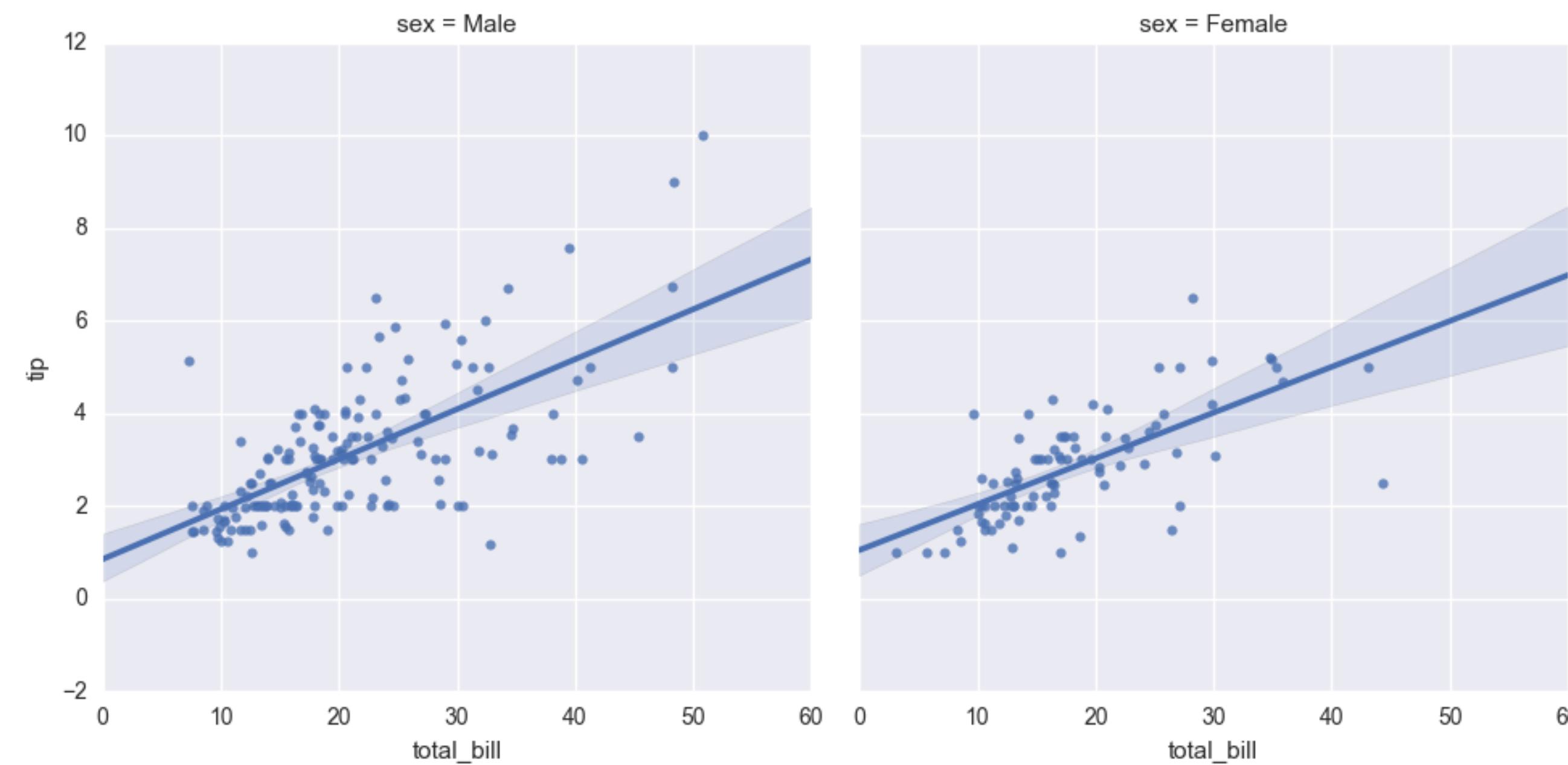
# Using hue=...

```
In [7]: sns.lmplot(x='total_bill', y='tip', data=tips, hue='sex',
...:                 palette='Set1')
```

```
In [8]: plt.show()
```



# Grouping factors (subplots)





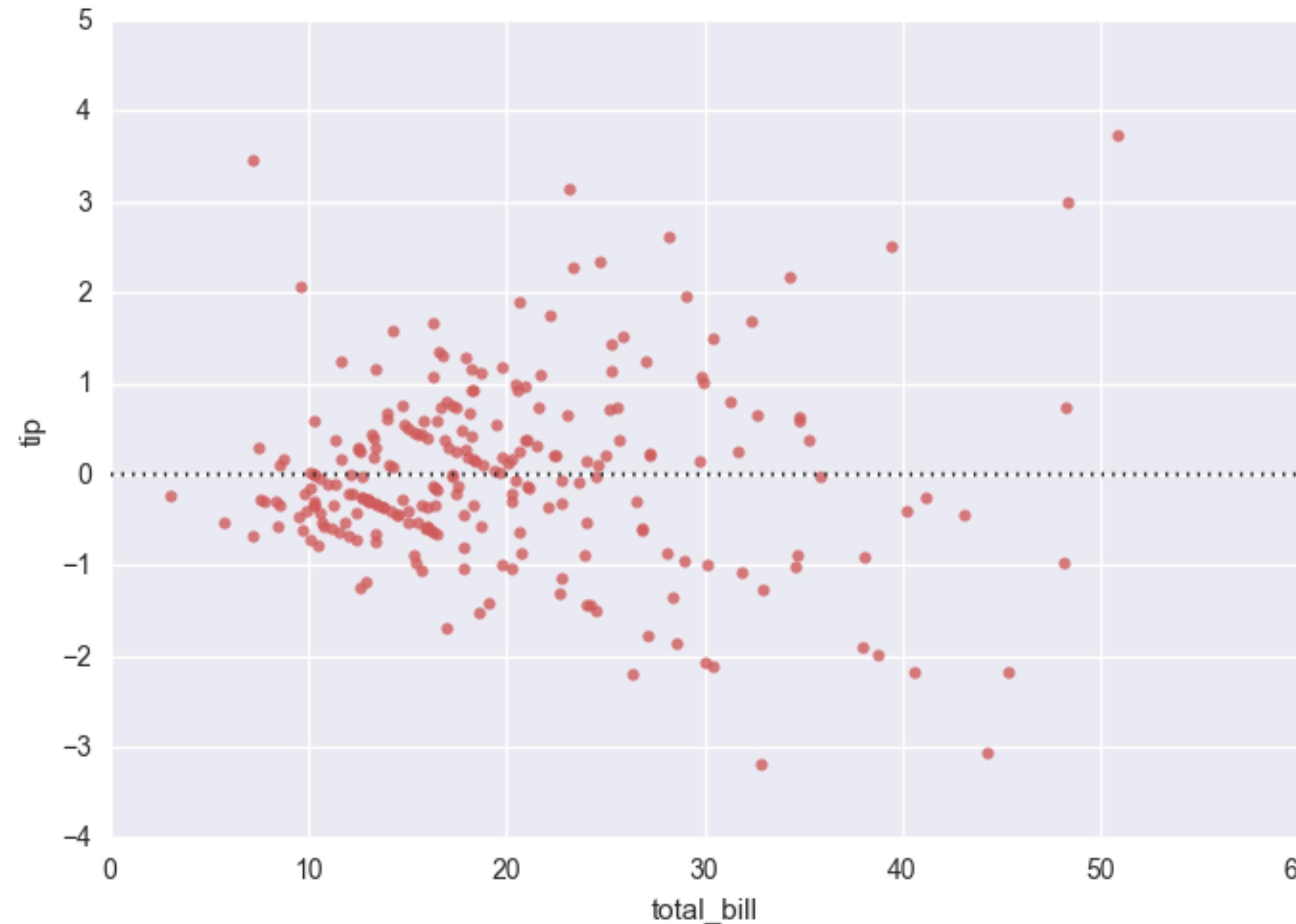
# Using col=...

```
In [9]: sns.lmplot(x='total_bill', y='tip', data=tips, col='sex')
```

```
In [10]: plt.show()
```



# Residual plots





# Using residplot()

```
In [11]: sns.residplot(x='age',y='fare',data=tips,color='indianred')
```

```
In [12]: plt.show()
```

- Similar arguments as lmplot() but more flexible
  - x, y can be *arrays or strings*
  - data is DataFrame (optional)
- Optional arguments (e.g., color) as in Matplotlib



INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

**Let's practice!**



INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

# Visualizing univariate distributions

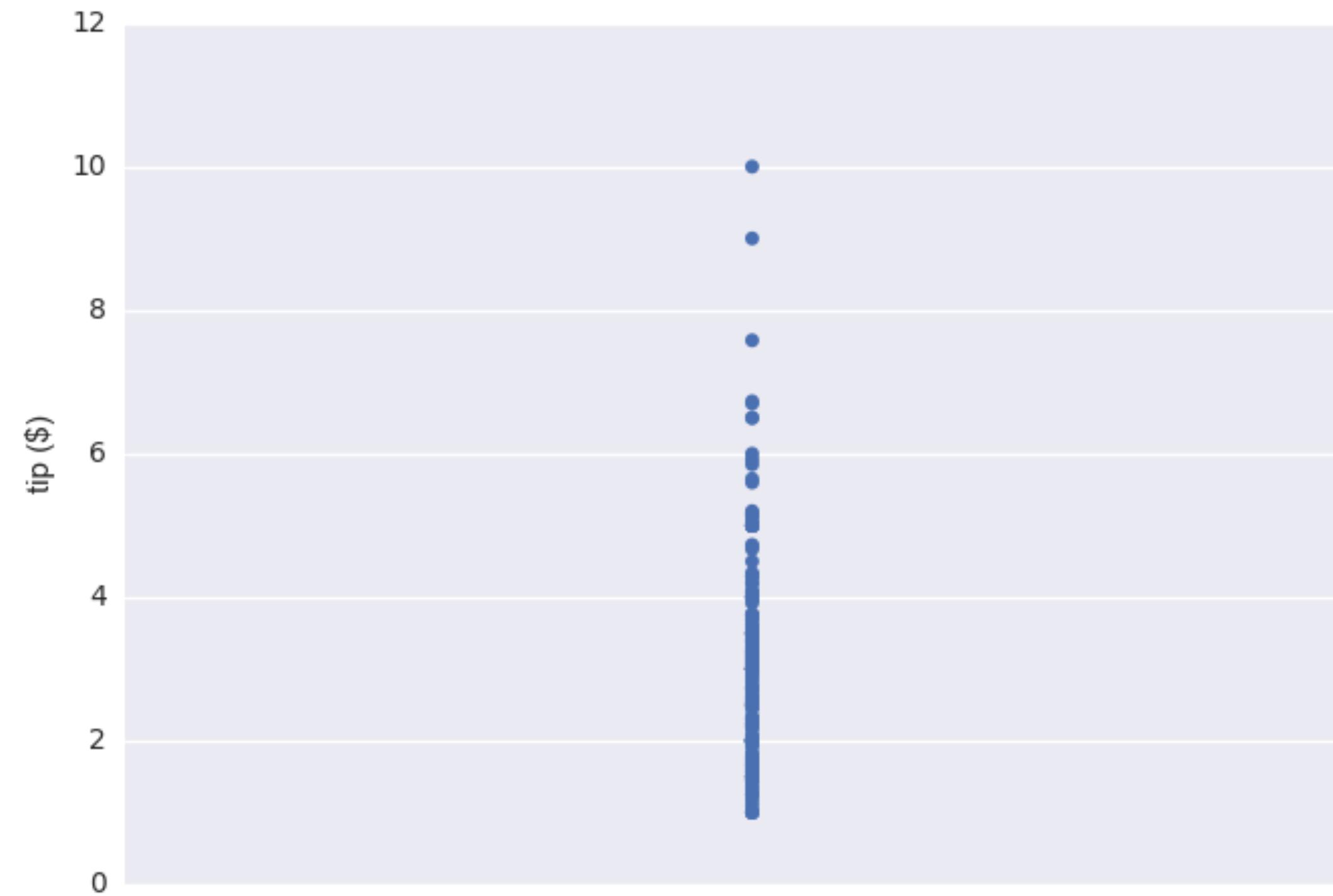


# Visualizing data

- Univariate → “one variable”
- Visualization techniques for sampled univariate data
  - Strip plots
  - Swarm plots
  - Violin plots



# Strip plot





# Using stripplot()

```
In [1]: sns.stripplot(y= 'tip', data=tips)
```

```
In [2]: plt.ylabel('tip ($)')
```

```
In [3]: plt.show()
```



# Grouping with stripplot()

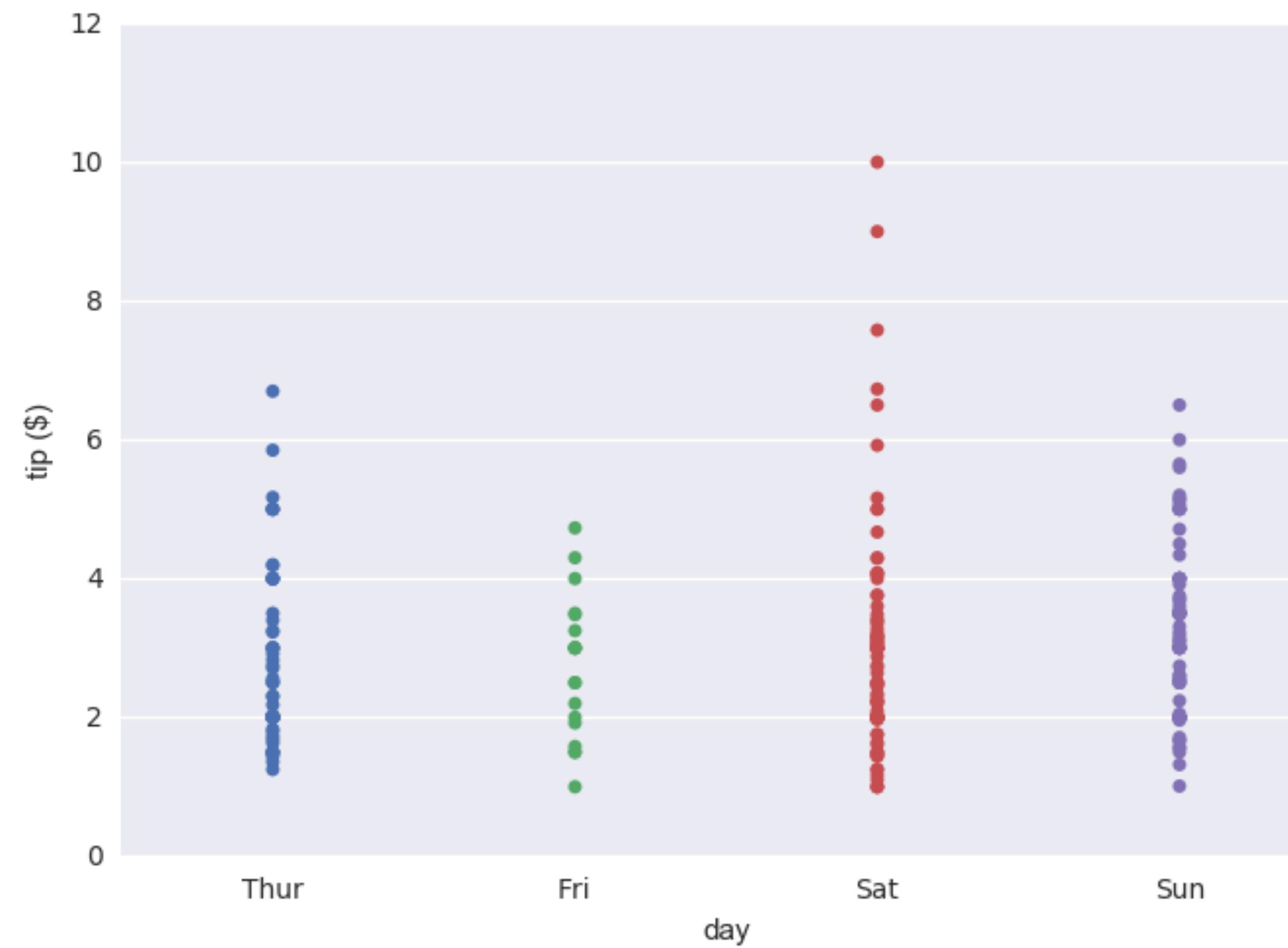
```
In [4]: sns.stripplot(x='day', y='tip', data=tip)
```

```
In [5]: plt.ylabel('tip ($)')
```

```
In [6]: plt.show()
```

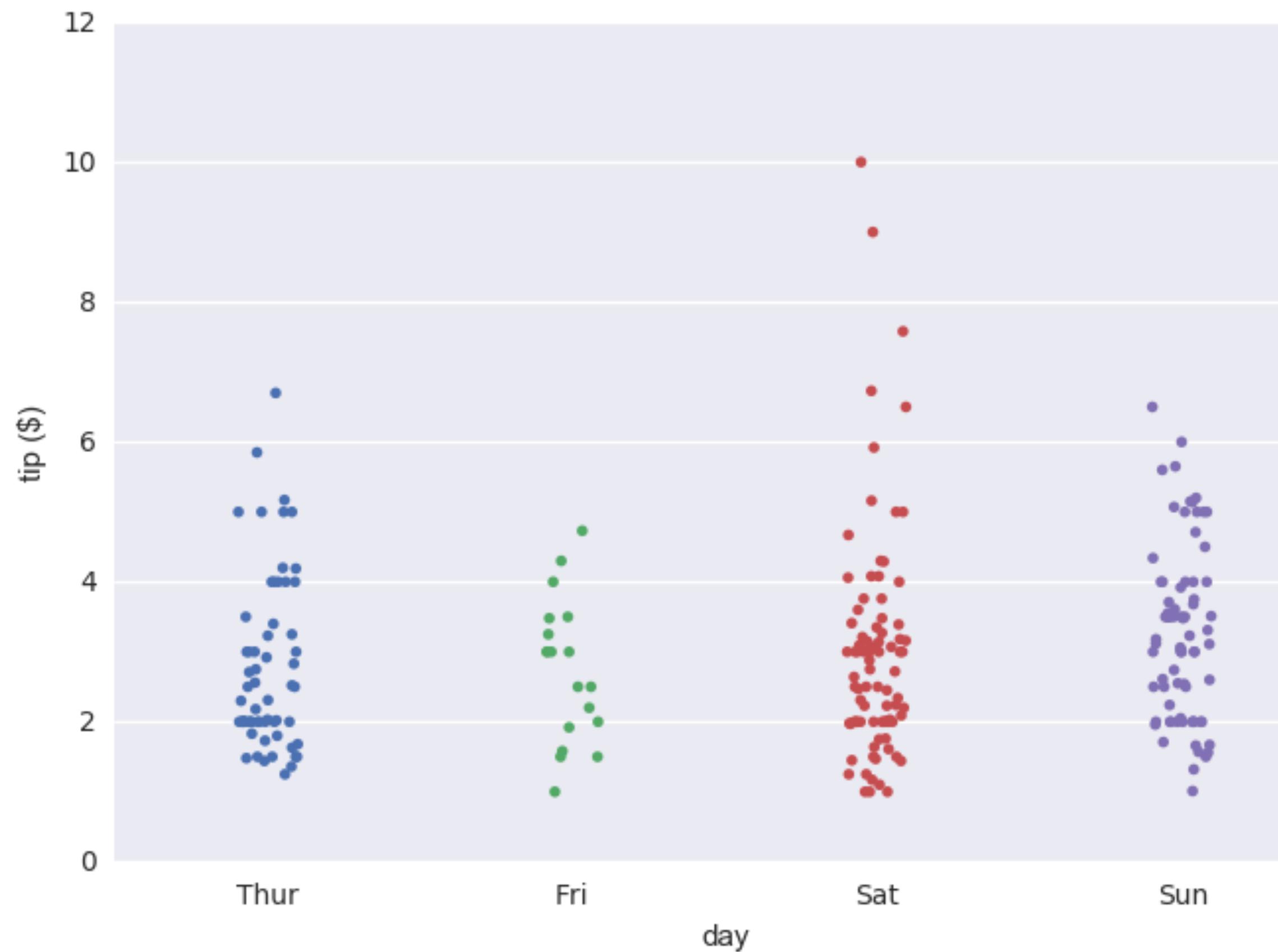


# Grouped strip plot





# Spreading out strip plots





# Spreading out strip plots

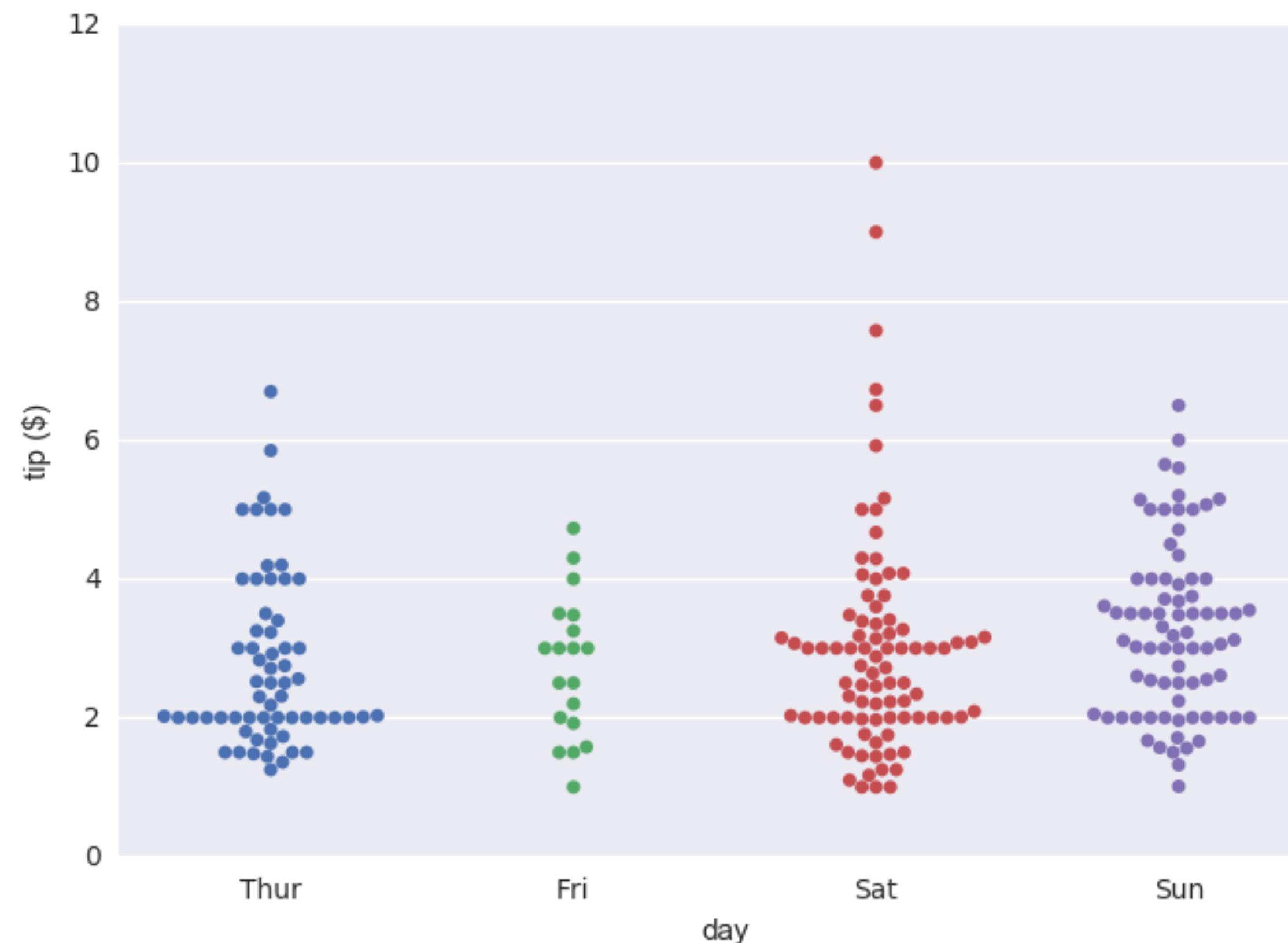
```
In [7]: sns.stripplot(x='day', y='tip', data=tip, size=4,  
...:                      jitter=True)
```

```
In [8]: plt.ylabel('tip ($)')
```

```
In [9]: plt.show()
```



# Swarm plot





# Using swarmplot()

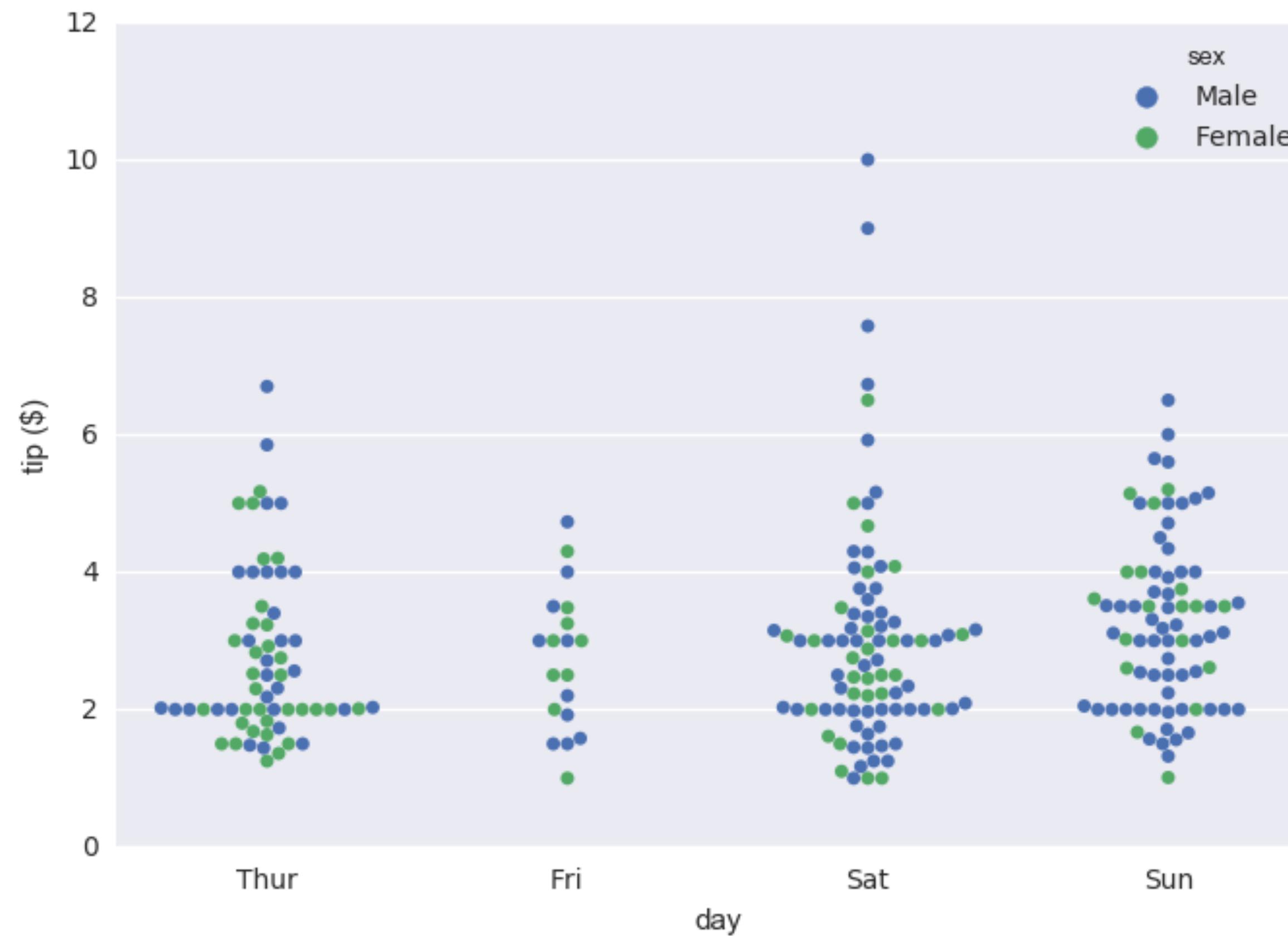
```
In [10]: sns.swarmplot(x='day', y='tip', data=tips)
```

```
In [11]: plt.ylabel('tip ($)')
```

```
In [12]: plt.show()
```



# More grouping





# More grouping with swarmplot()

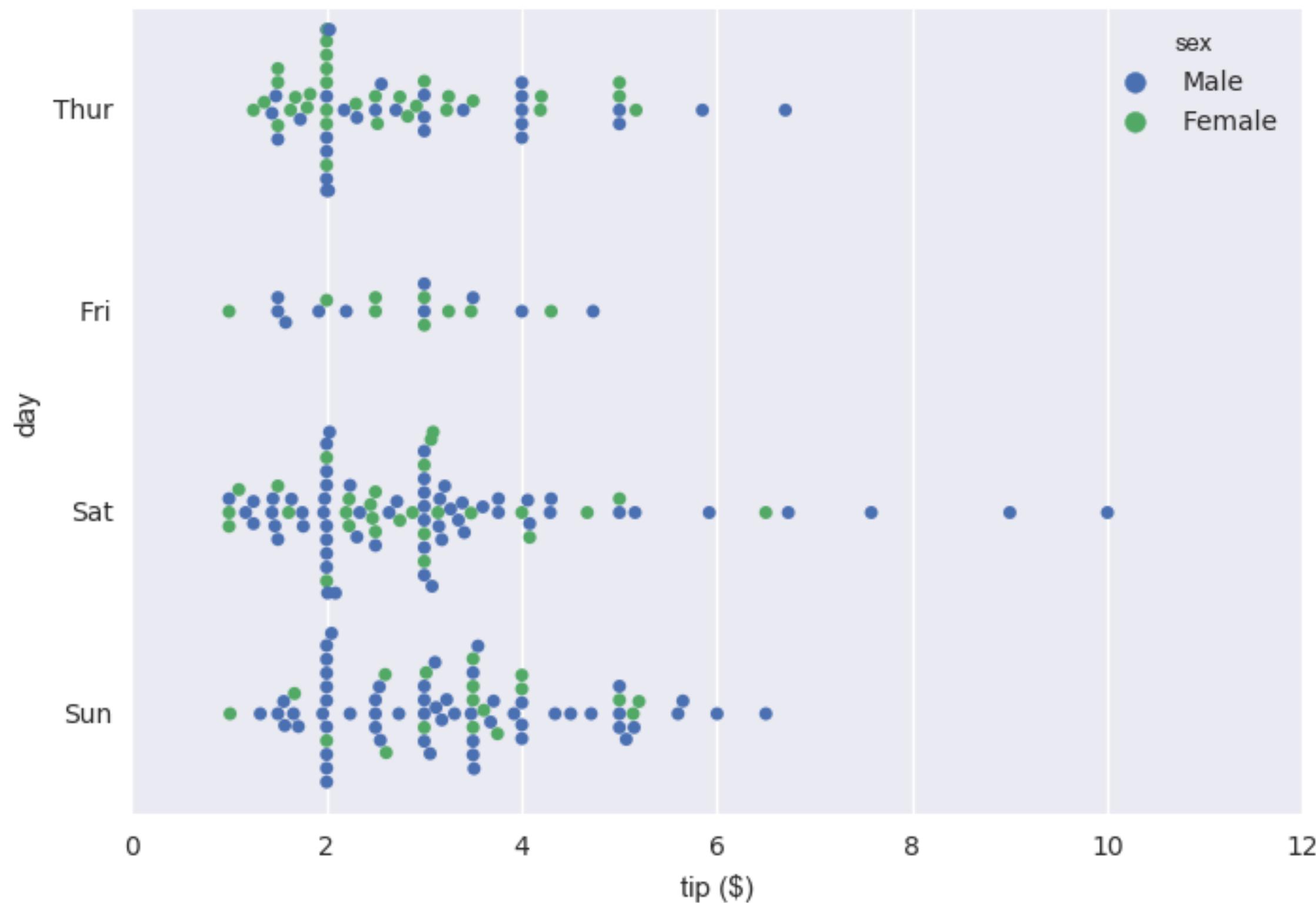
```
In [13]: sns.swarmplot(x='day', y='tip', data=tips, hue='sex')
```

```
In [14]: plt.ylabel('tip ($)')
```

```
In [15]: plt.show()
```



# Changing orientation





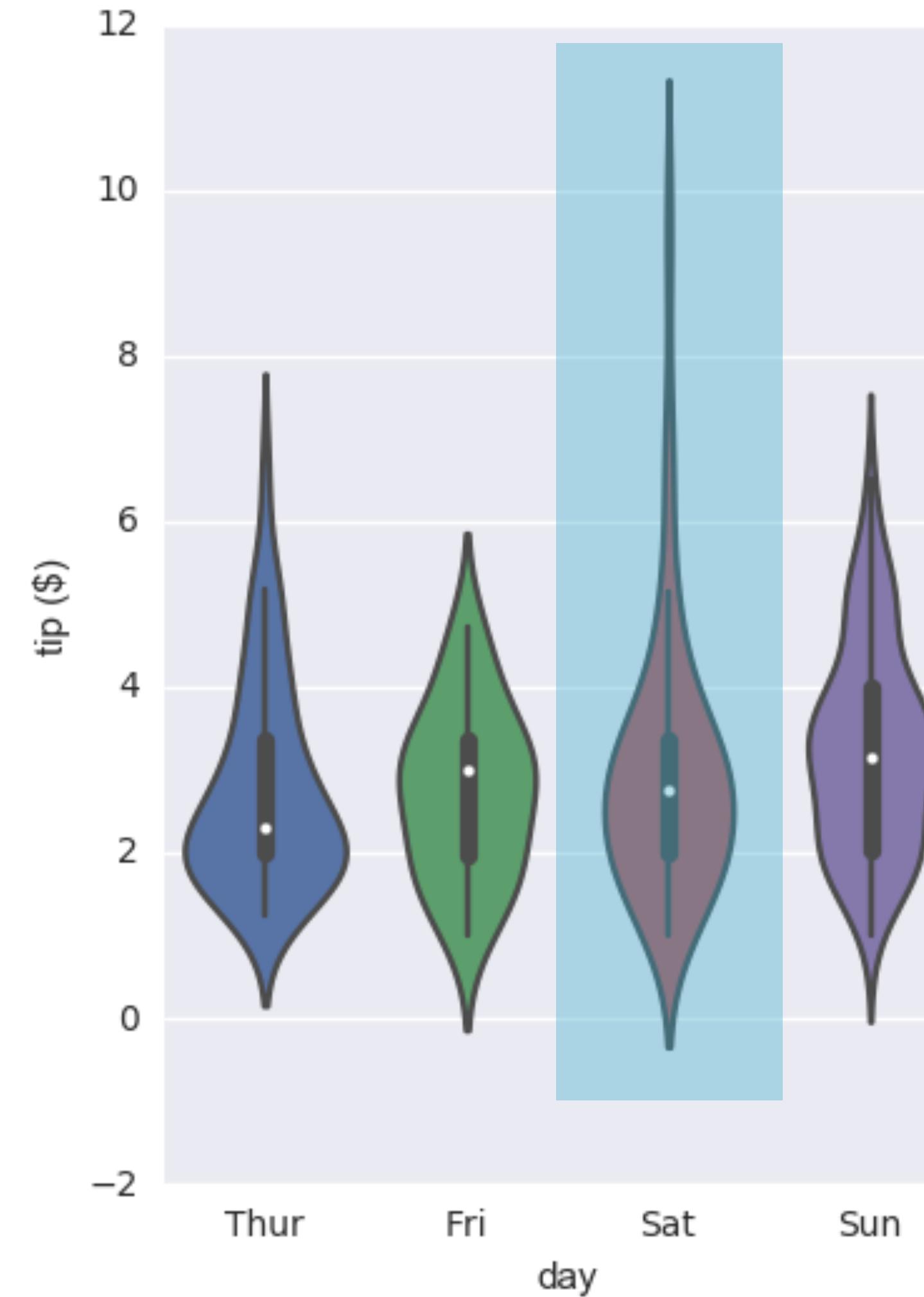
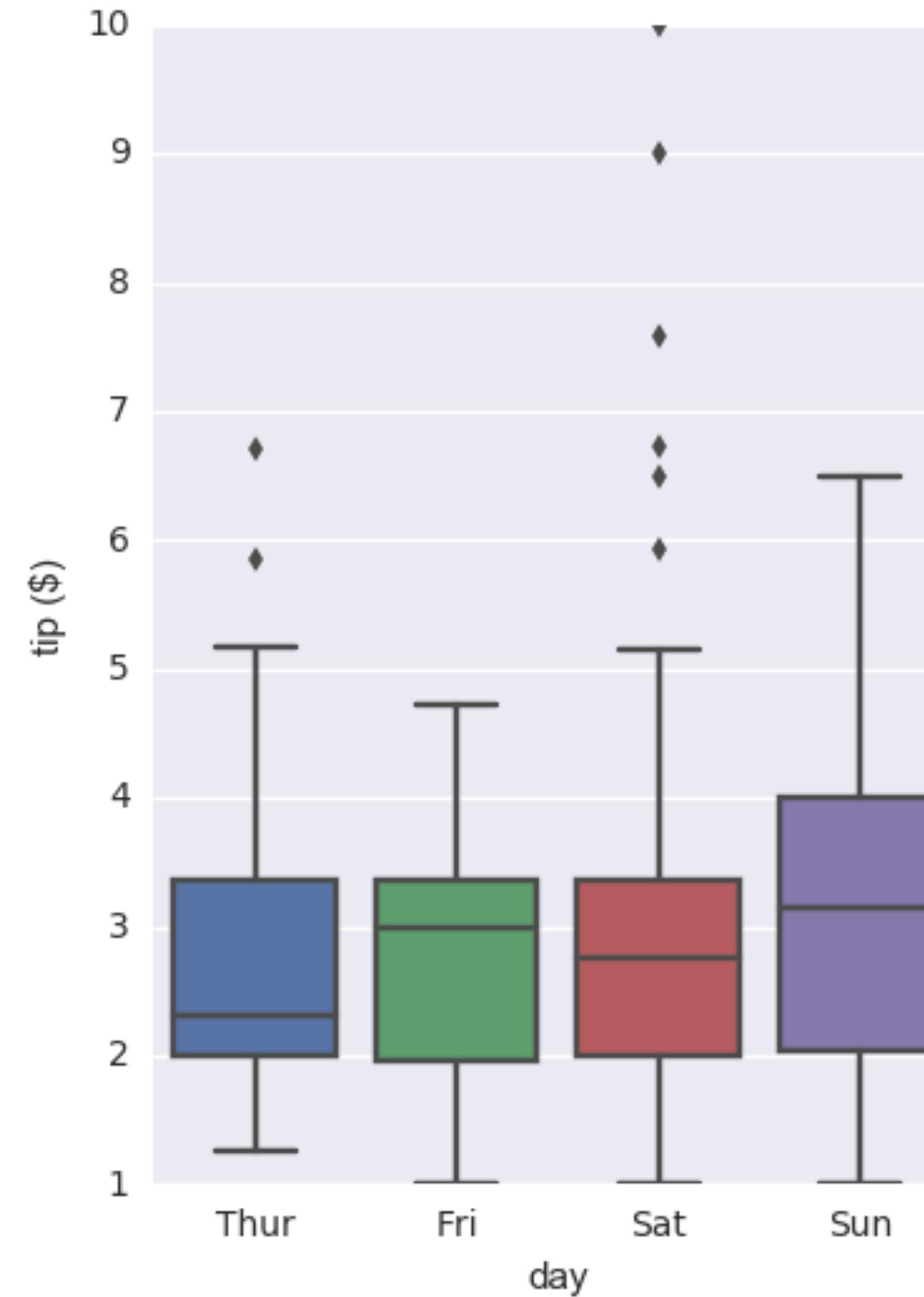
# Changing orientation

```
In [16]: sns.swarmplot(x='tip', y='day', data=tips, hue='sex',  
....:                  orient='h')
```

```
In [17]: plt.xlabel('tip ($)')
```

```
In [18]: plt.show()
```

# Violin plot





# Using violinplot()

```
In [19]: plt.subplot(1,2,1)
```

```
In [20]: sns.boxplot(x='day', y='tip', data=tips)
```

```
In [21]: plt.ylabel('tip ($)')
```

```
In [22]: plt.subplot(1,2,2)
```

```
In [23]: sns.violinplot(x='day', y='tip', data=tips)
```

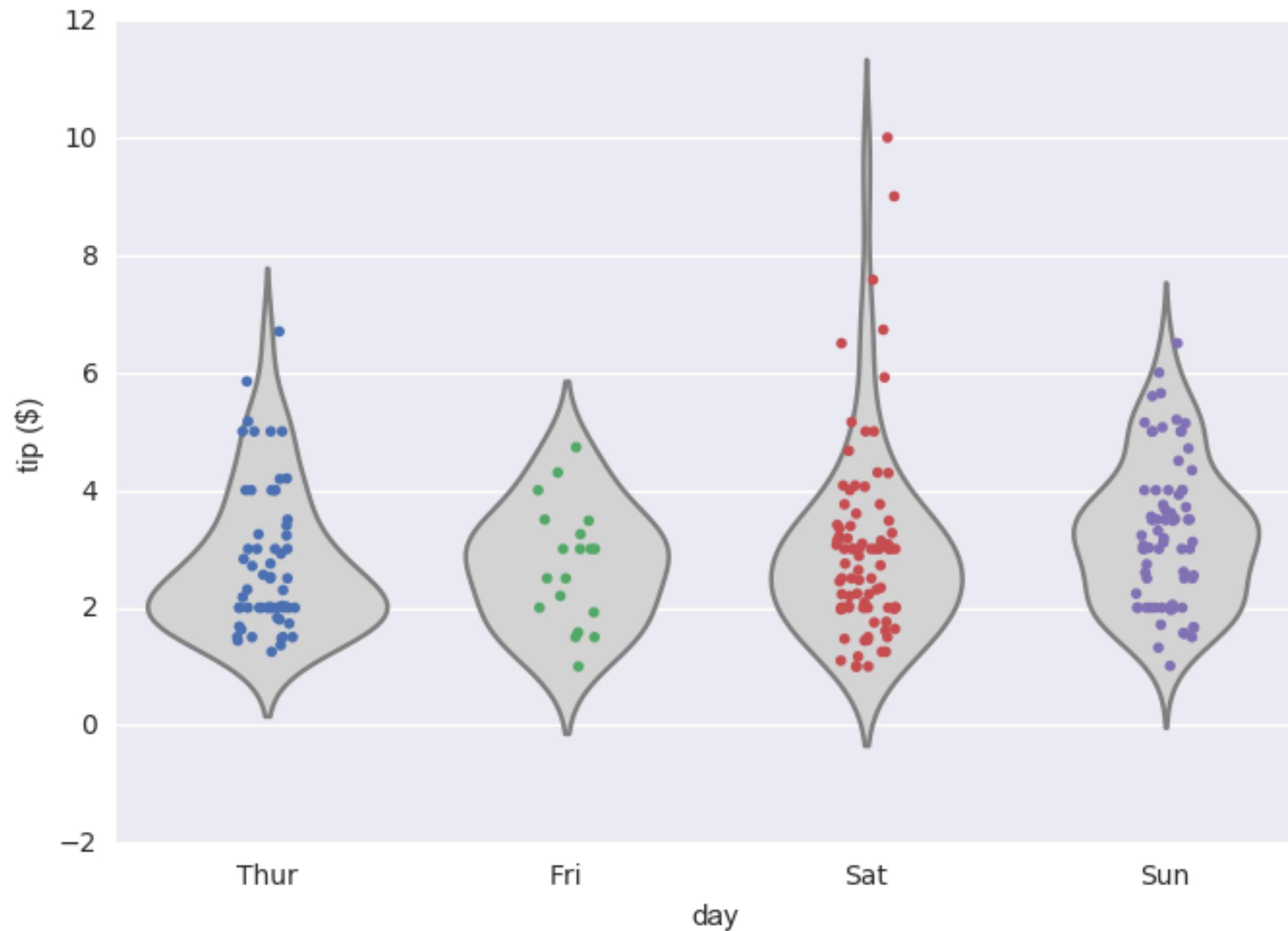
```
In [24]: plt.ylabel('tip ($)')
```

```
In [25]: plt.tight_layout()
```

```
In [26]: plt.show()
```



# Combining plots





# Combining plots

```
In [27]: sns.violinplot(x='day', y='tip', data=tips, inner=None,  
...:                      color='lightgray')
```

```
In [28]: sns.stripplot(x='day', y='tip', data=tips, size=4,  
...:                      jitter=True)
```

```
In [29]: plt.ylabel('tip ($)')
```

```
In [30]: plt.show()
```



INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

**Let's practice!**



INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

# Visualizing Multivariate Distributions

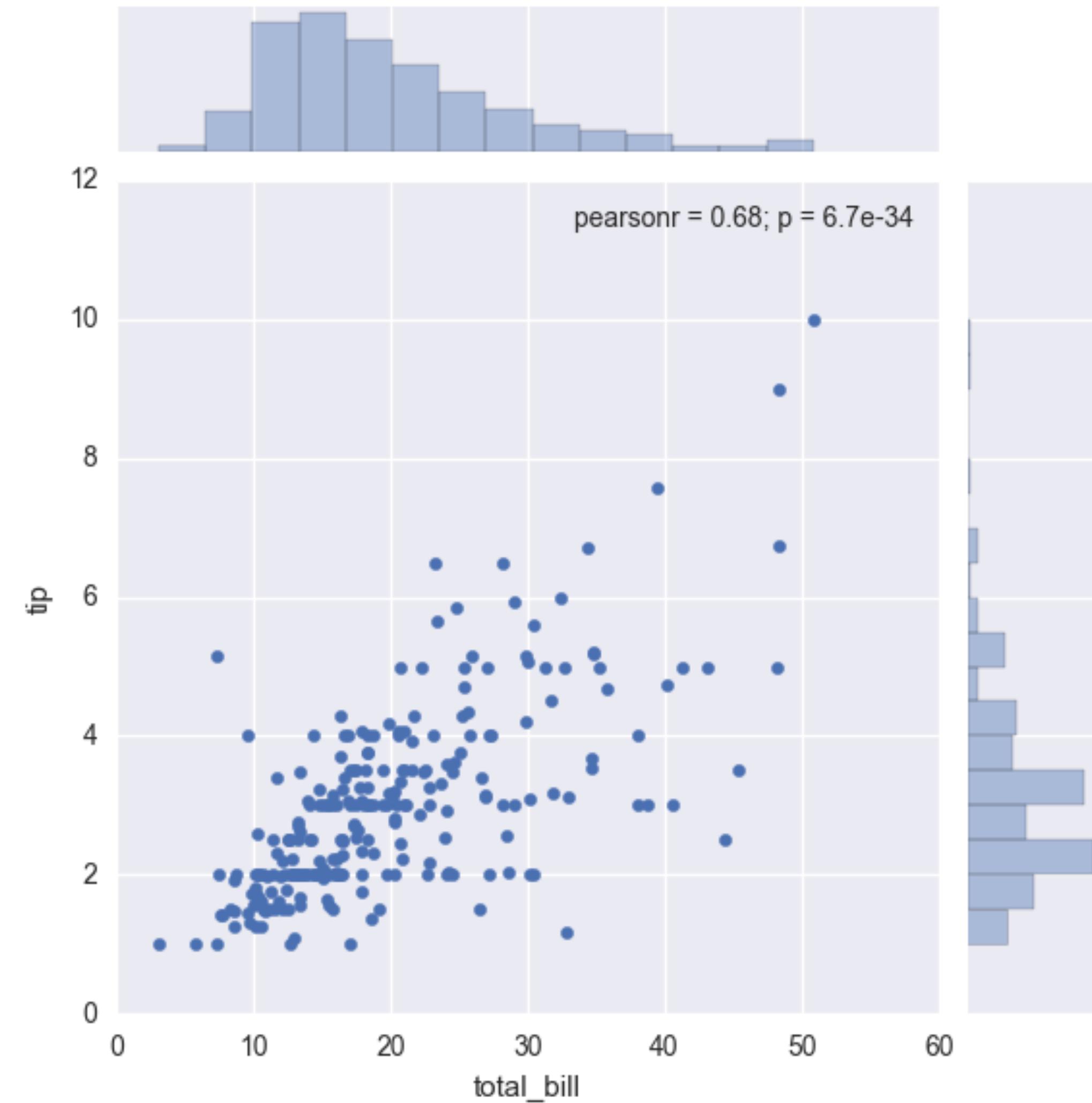


# Visualizing data

- Bivariate → “two variables”
- Multivariate → “multiple variables”
- Visualizing relationships in multivariate data
  - Joint plots
  - Pair plots
  - Heat maps



# Joint plot





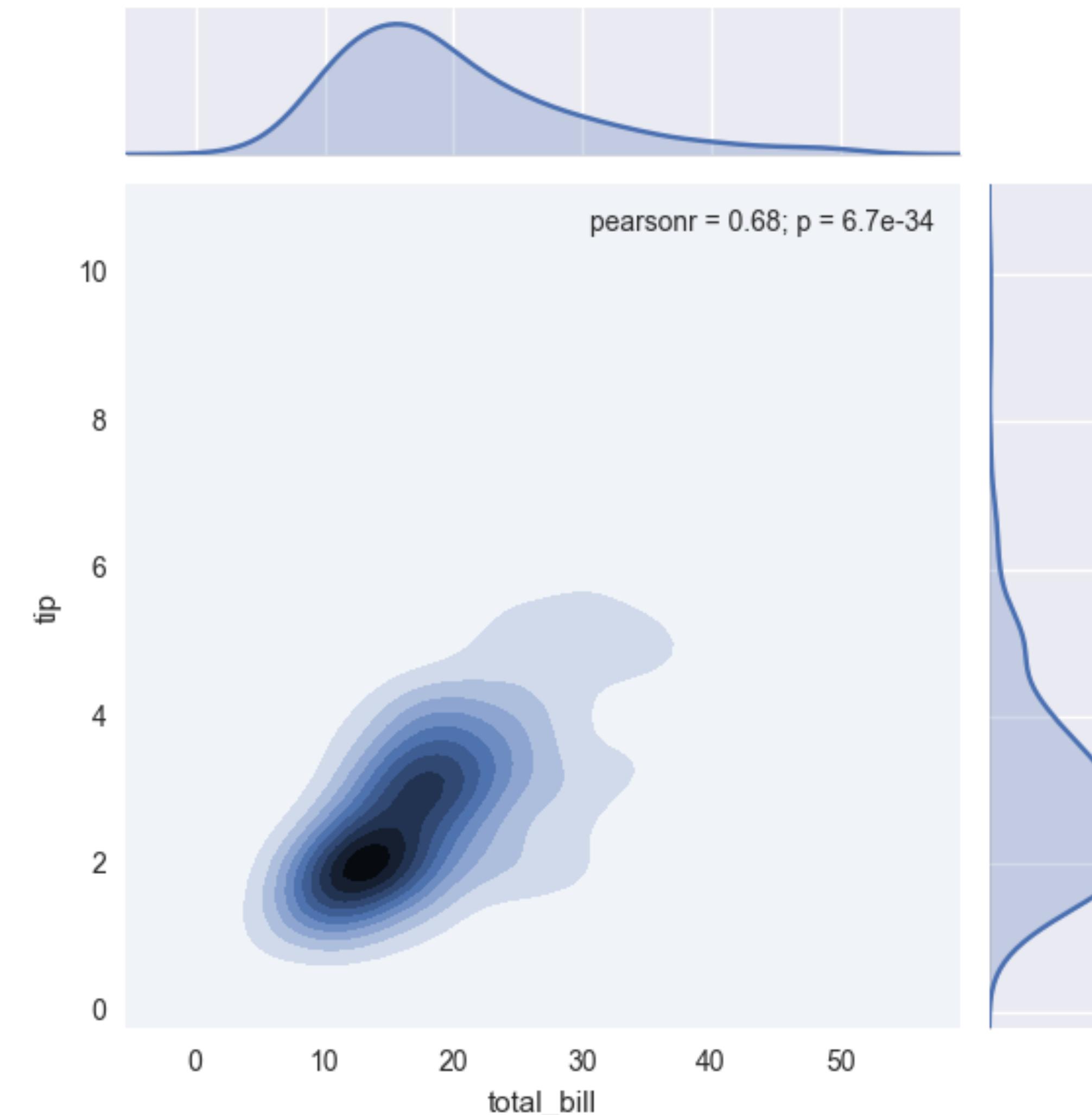
# Using jointplot()

```
In [1]: sns.jointplot(x= 'total_bill', y= 'tip', data=tips)
```

```
In [2]: plt.show()
```



# Joint plot using KDE



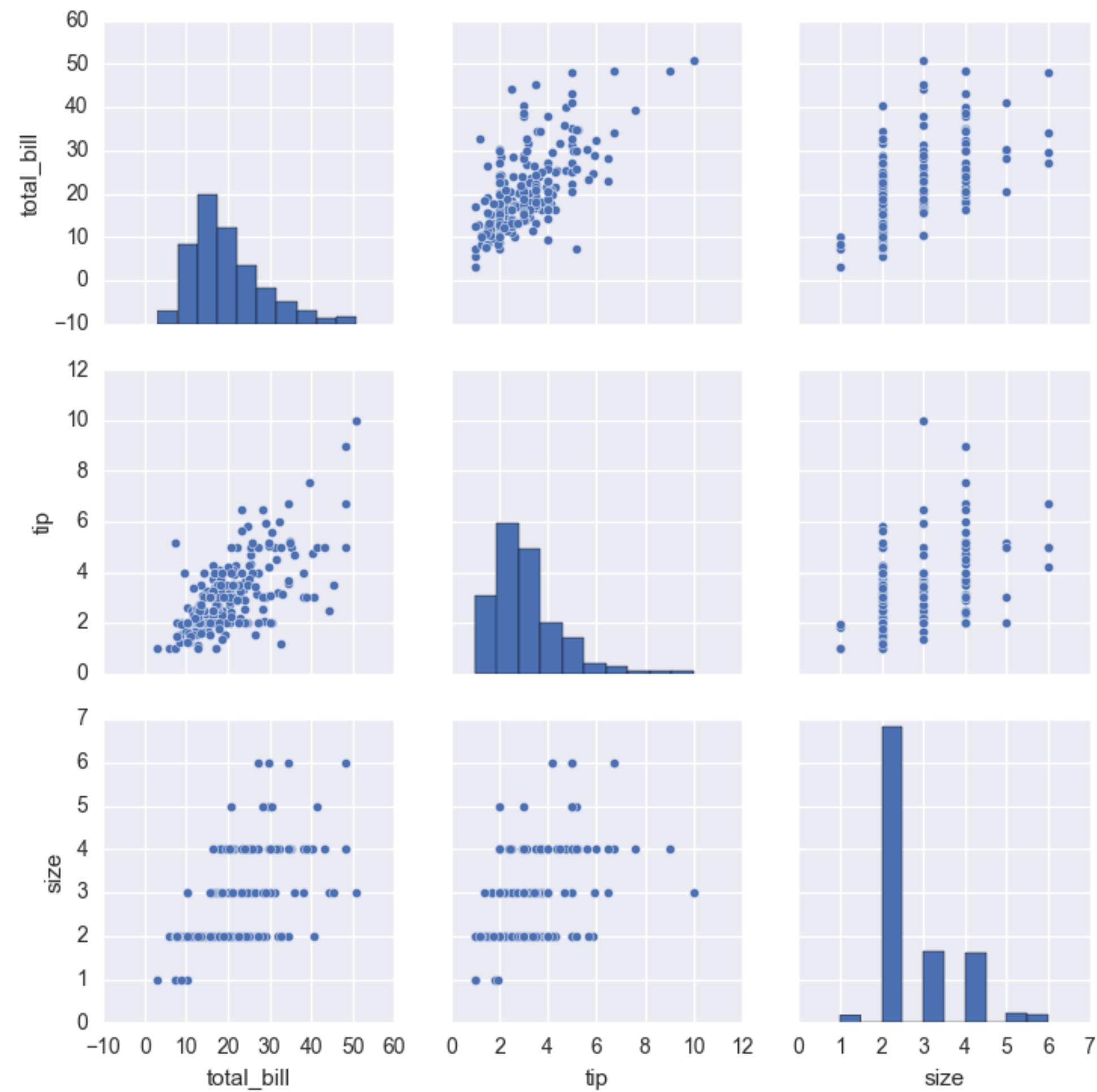


# Using kde=True

```
In [3]: sns.jointplot(x='total_bill', y= 'tip', data=tips,  
....:                 kind='kde')
```

```
In [4]: plt.show()
```

# Pair plot





# Using pairplot()

```
In [5]: sns.pairplot(tips)
```

```
In [6]: plt.show()
```

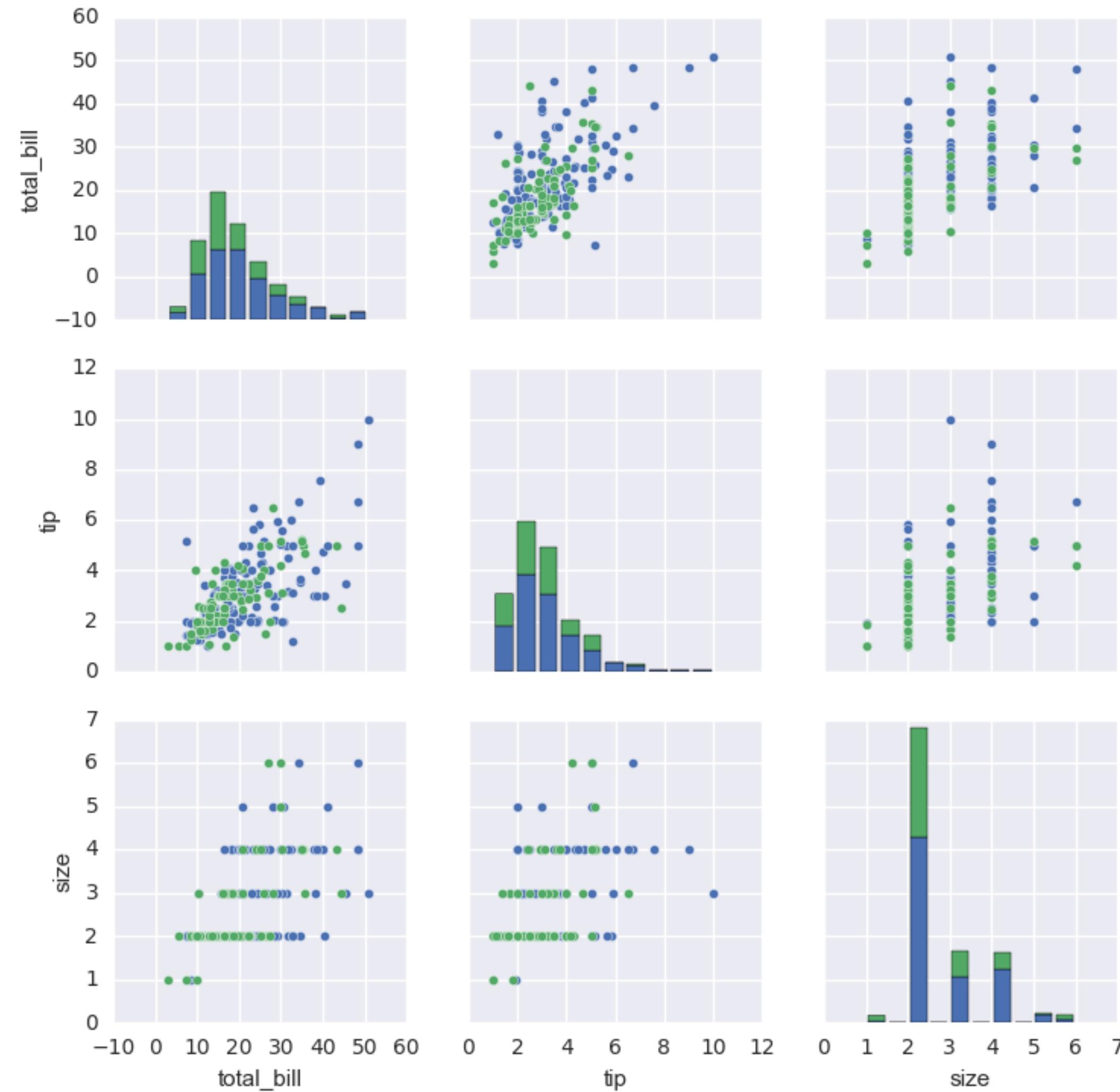


# Using pairplot() with hue

```
In [7]: sns.pairplot(tips, hue='sex')
```

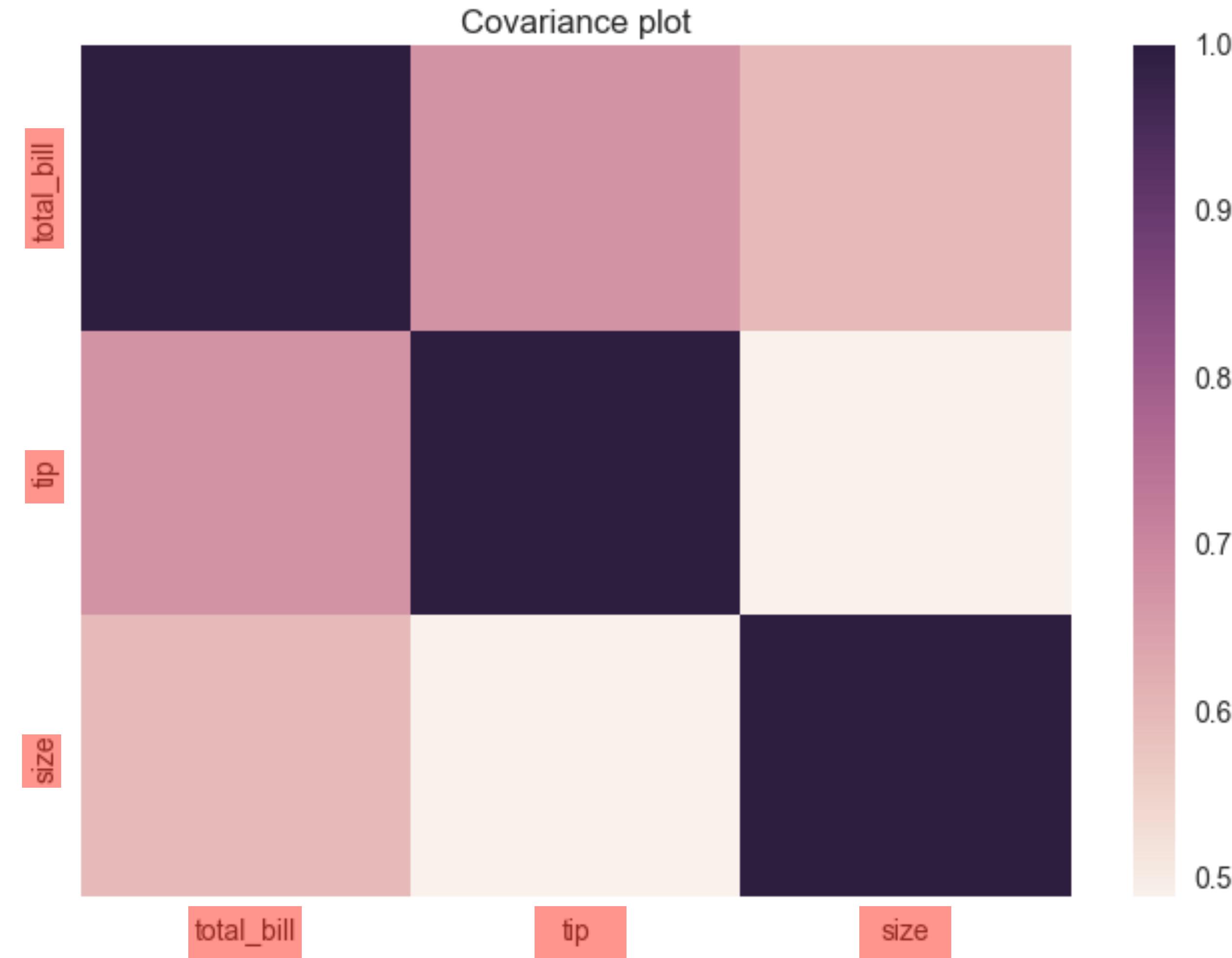
```
In [8]: plt.show()
```

# Using pairplot() with hue





# Covariance heat map of tips data





# Using heatmap()

```
In [9]: print(covariance)
      total_bill      tip      size
total_bill    1.000000  0.675734  0.598315
tip          0.675734  1.000000  0.489299
size         0.598315  0.489299  1.000000
```

```
In [10]: sns.heatmap(covariance)
```

```
In [11]: plt.title('Covariance plot')
```

```
In [12]: plt.show()
```



INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

**Let's practice!**



INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

# Visualizing time series



# Datetimes & time series

```
In [1]: type(weather)  
Out[1]: pandas.core.frame.DataFrame
```

```
In [2]: type(weather.index)  
Out[2]: pandas.tseries.index.DatetimeIndex
```

Date	Temperature	DewPoint	Pressure
2010-01-01 00:00:00	46.2	37.5	1.0
2010-01-01 01:00:00	44.6	37.1	1.0
2010-01-01 02:00:00	44.1	36.9	1.0
2010-01-01 03:00:00	43.8	36.9	1.0
2010-01-01 04:00:00	43.5	36.8	1.0
...	...	...	...

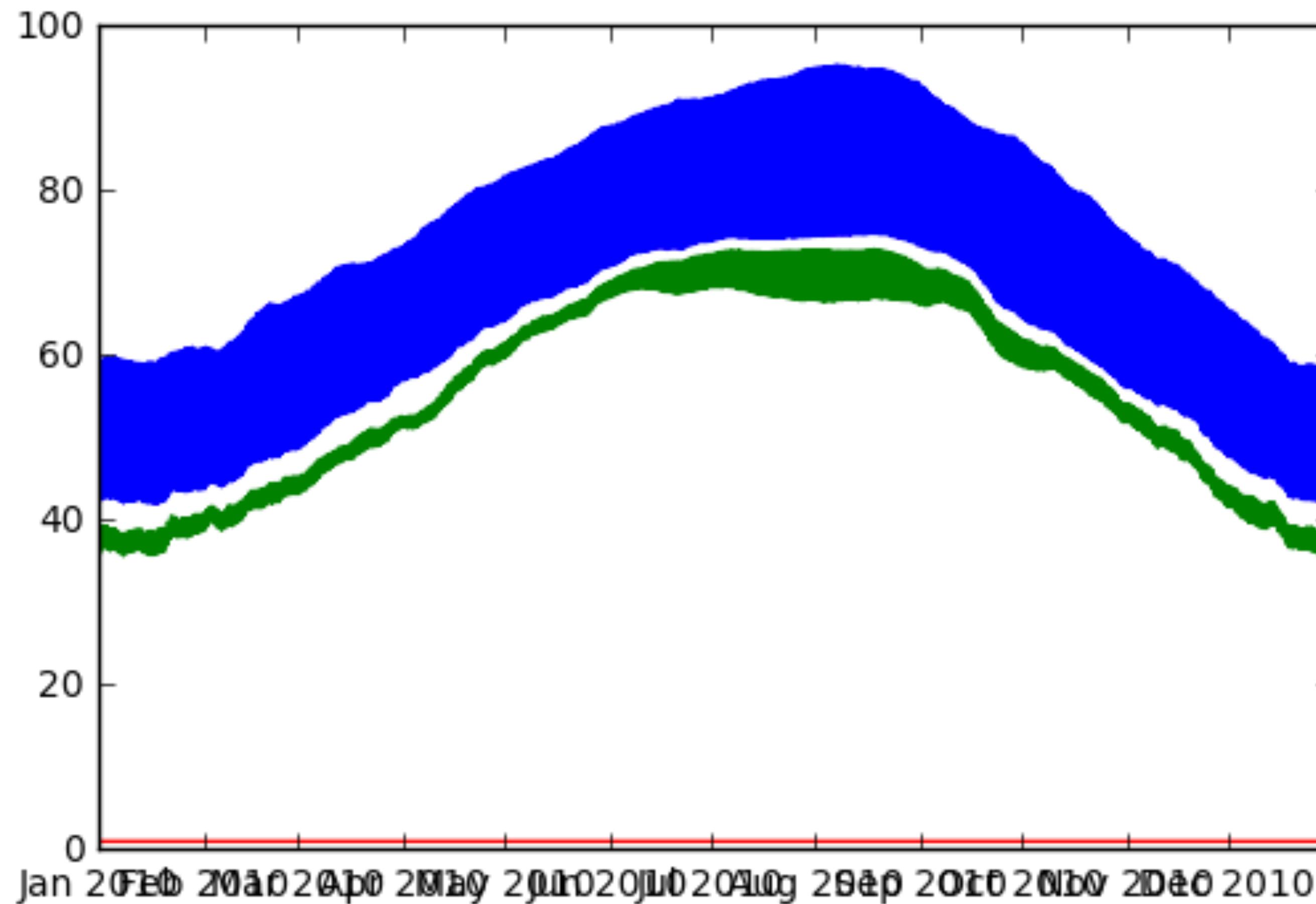


# Plotting DataFrames

```
In [1]: plt.plot(weather)
```

```
In [2]: plt.show()
```

# Plotting DataFrames





# Time series

- Pandas time series: *datetime* as *index*
- Datetime: represents periods or time-stamps
- Datetime index: specialized slicing
  - e.g., `weather['2010-07-04']`
  - e.g., `weather['2010-03':'2010-04']`
  - e.g., `weather['2010-05']`



# Slicing time series

```
In [1]: temperature = weather['Temperature']
```

```
In [2]: march_apr = temperature['2010-03':'2010-04'] # data of  
....: March & April 2010 only
```

```
In [3]: march_apr.shape  
Out[3]: (1463,)
```

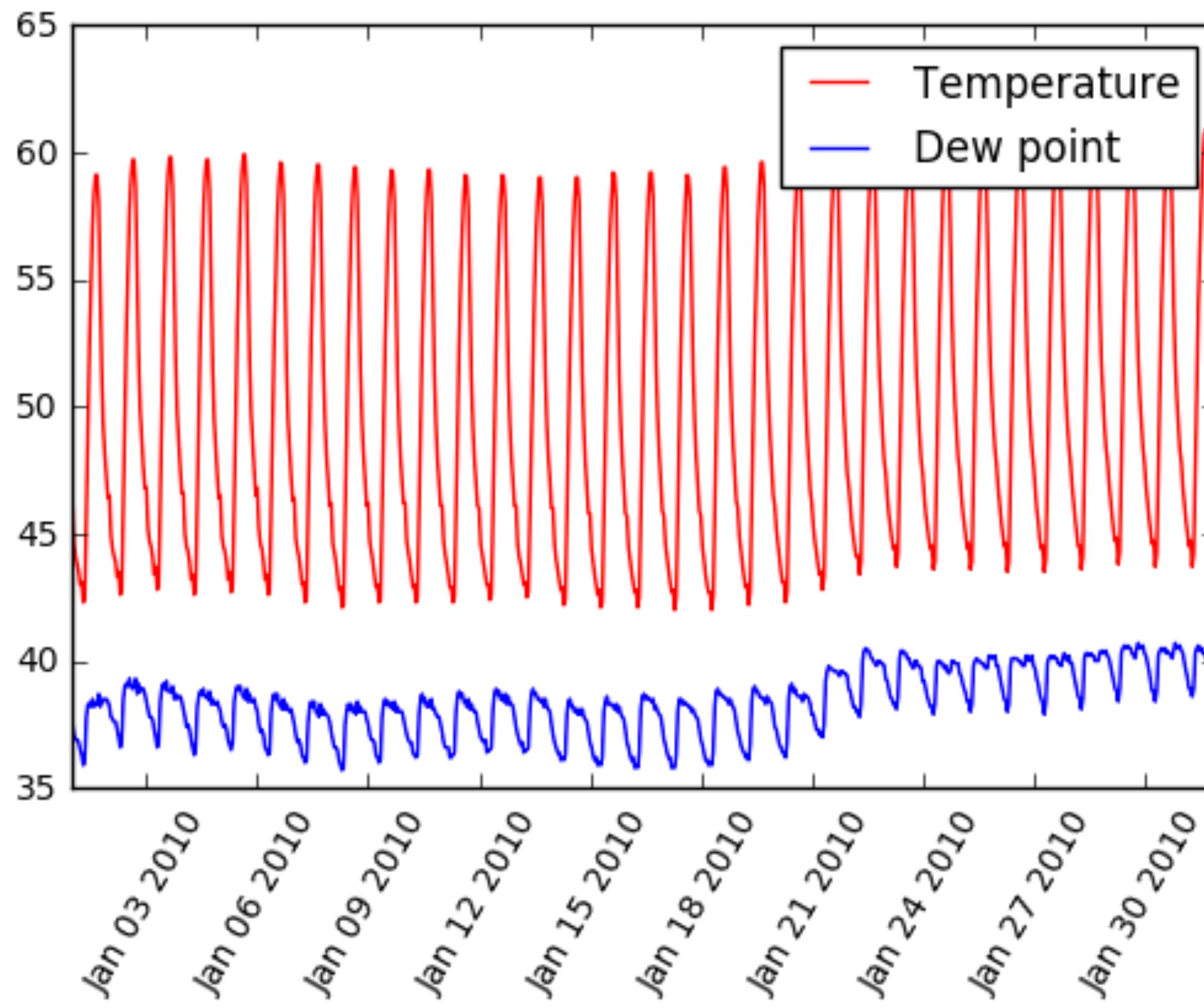
```
In [4]: march_apr.iloc[-4:] #extract last 4 entries from time  
series
```

```
Out[4]:  
Date  
2010-04-30 20:00:00    73.3  
2010-04-30 21:00:00    71.3  
2010-04-30 22:00:00    69.7  
2010-04-30 23:00:00    68.5
```

```
Name: Temperature, dtype: float64
```



# Plotting time series slices





# Plotting time series slices

```
In [1]: plt.plot(temperature['2010-01'], color='red',
...:                 label='Temperature')
```

```
In [2] dew point = weather['DewPoint']
```

```
In [3]: plt.plot(dewpoint['2010-01'], color='blue',
...:                 label='Dewpoint')
```

```
In [4]: plt.legend(loc='upper right')
```

```
In [5]: plt.xticks(rotation=60)
```

```
In [6]: plt.show()
```



# Selecting & formatting dates

```
In [1]: jan = temperature['2010-01']
```

```
In [2]: dates = jan.index[::96] # Pick every 4th day
```

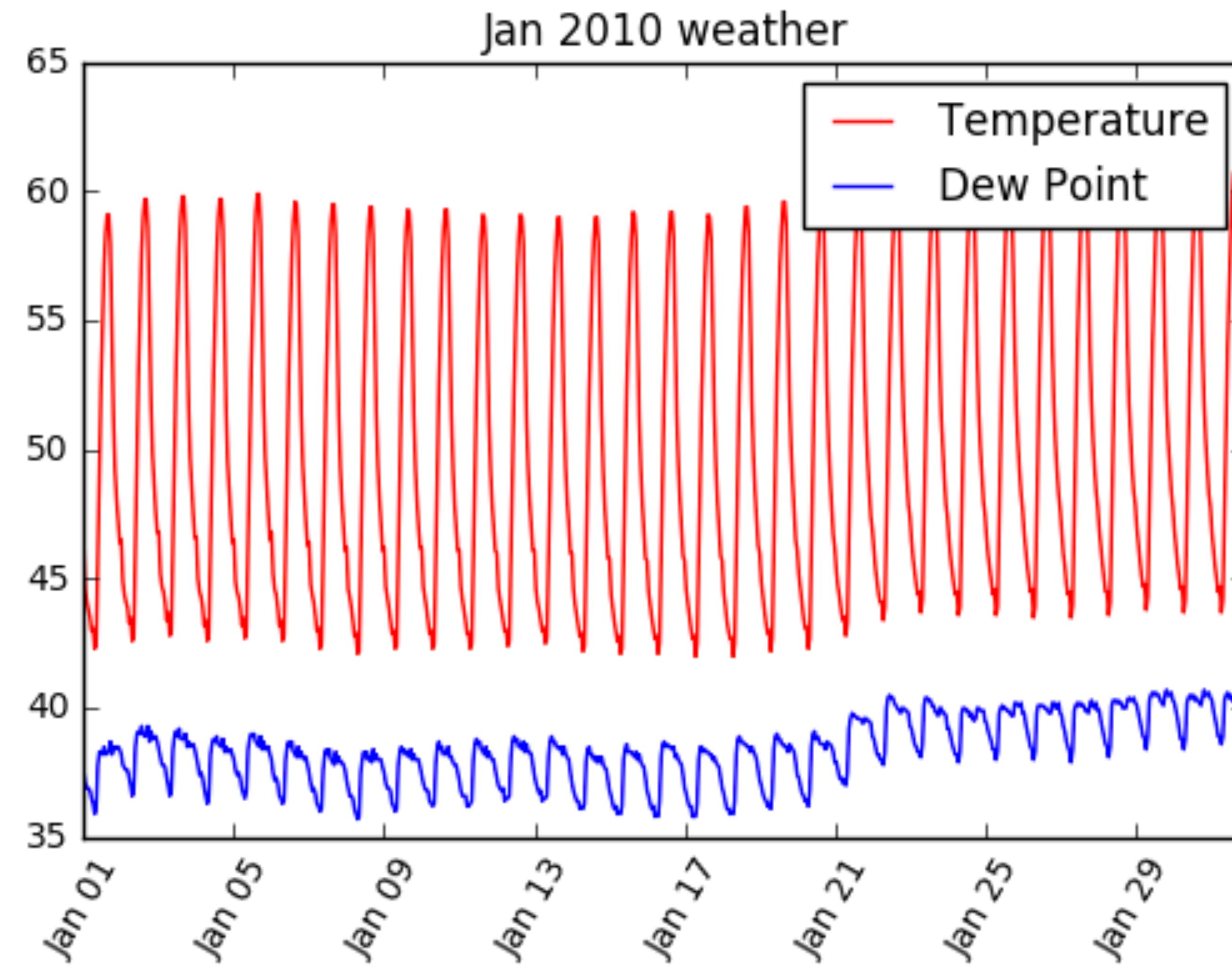
```
In [3]: print(dates)
DatetimeIndex(['2010-01-01', '2010-01-05', '2010-01-09',
               '2010-01-13', '2010-01-17', '2010-01-21', '2010-01-25',
               '2010-01-29'], dtype='datetime64[ns]', name='Date', freq=None)
```

```
In [4]: labels = dates.strftime('%b %d') # Make formatted labels
```

```
In [5]: print(labels)
['Jan 01' 'Jan 05' 'Jan 09' 'Jan 13' 'Jan 17' 'Jan 21' 'Jan 25'
 'Jan 29']
```



# Cleaning up ticks on axis





# Cleaning up ticks on axis

```
In [1]: plt.plot(temperature['2010-01'], color='red',  
...:             label='Temperature')
```

```
In [2]: plt.plot(dewpoint['2010-01'], color='blue',  
...:             label='Dewpoint')
```

```
In [3]: plt.xticks(dates, labels, rotation=60)
```

```
In [4]: plt.legend(loc='upper right')
```

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



INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

**Let's practice!**

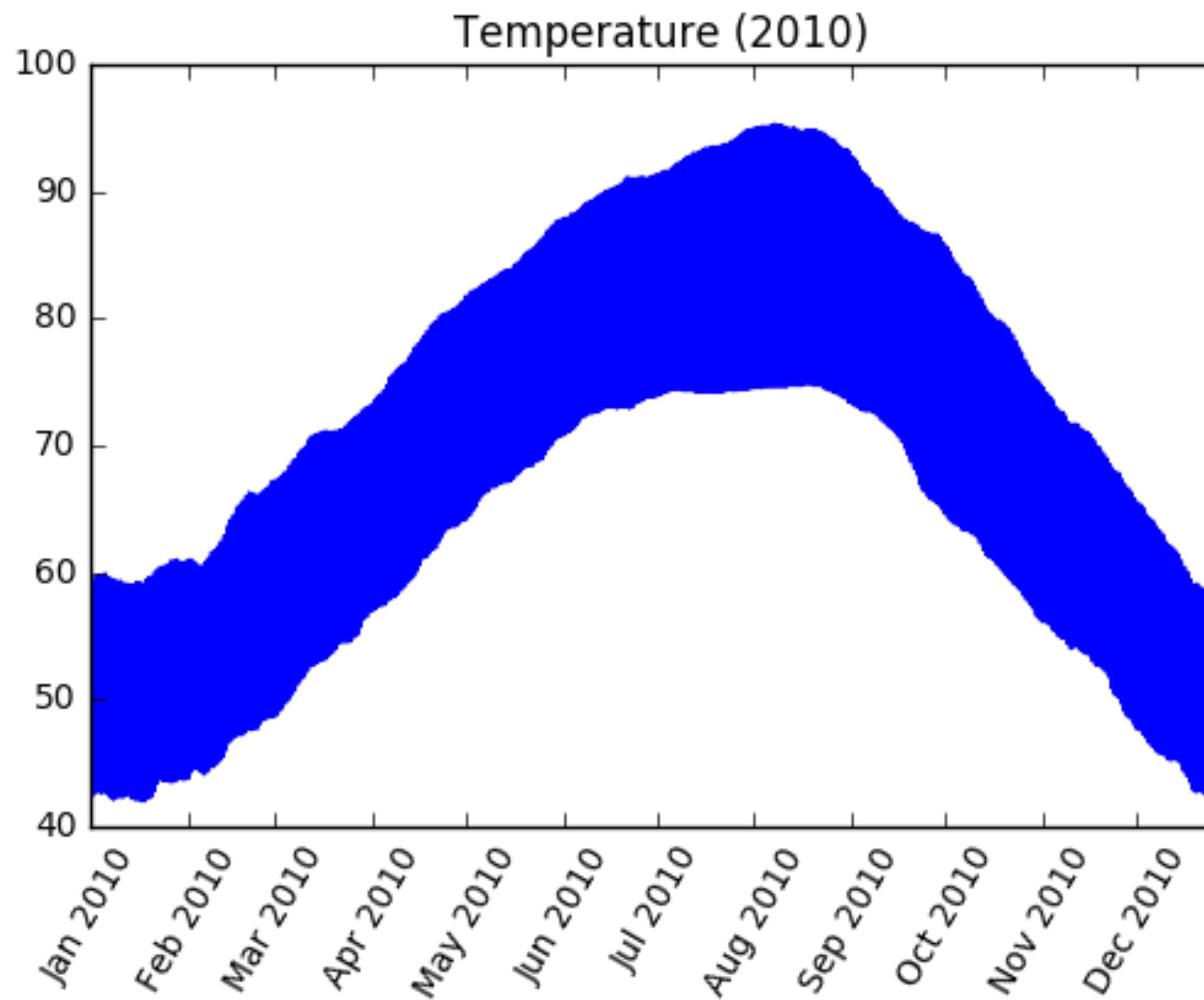


INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

# Time series with moving windows

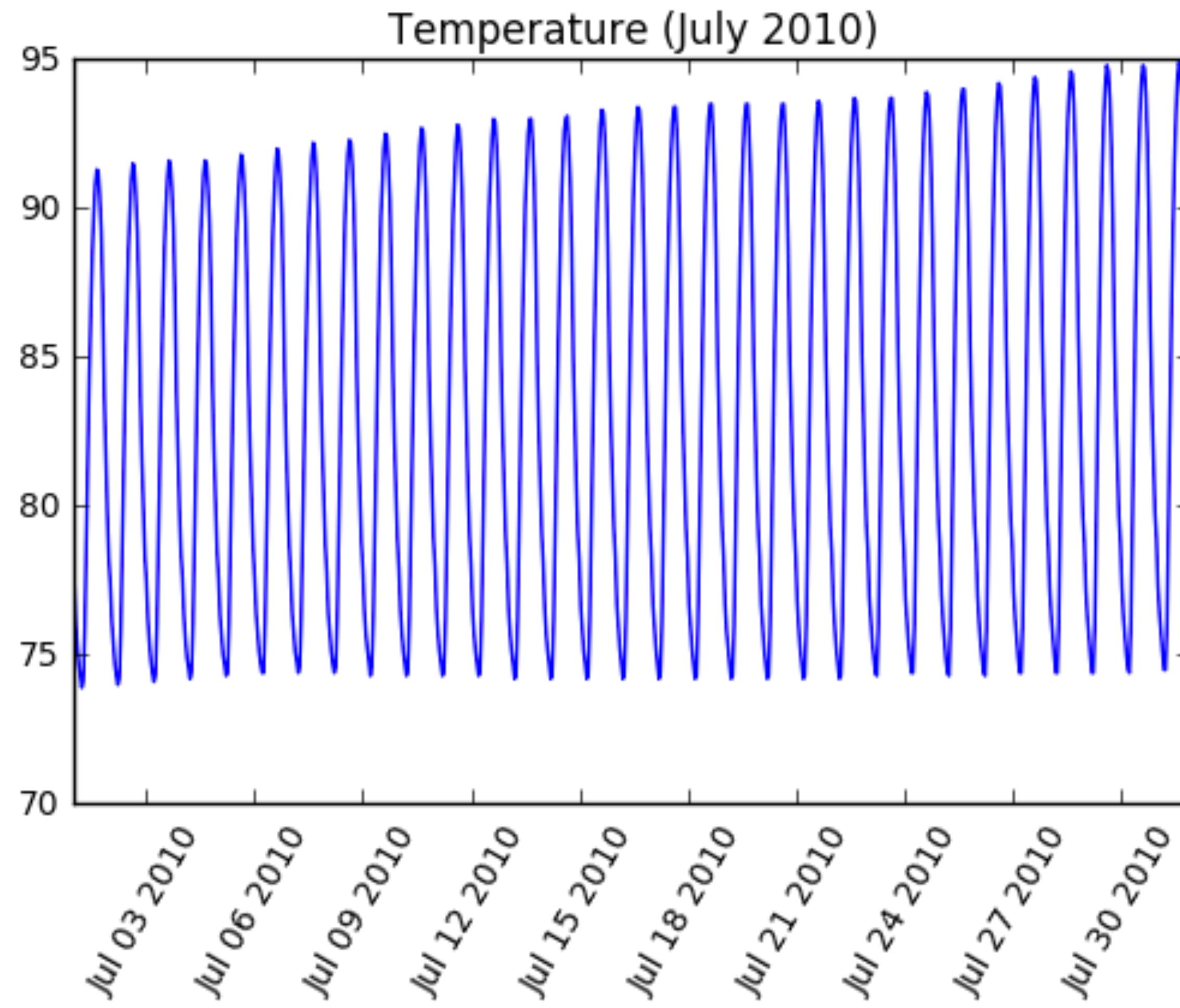


# Hourly data over a year





# Zooming in





# Moving windows & time series

- Moving window calculations
  - Averages
  - Medians
  - Standard deviations
- Extracts information on longer time scales
- See Pandas courses on how to compute



# Moving averages

```
In [1]: smoothed.info() # smoothed computing using moving averages
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 8759 entries, 2010-01-01 00:00:00 to 2010-12-31
23:00:00
Data columns (total 5 columns):
14d            8424 non-null float64
1d             8736 non-null float64
3d             8688 non-null float64
7d             8592 non-null float64
dtypes: float64(5)
memory usage: 410.6 KB
```

```
In [2]: print(smoothed.iloc[:3,:])
        14d   1d   3d   7d   Temperature
Date
2010-01-01 00:00:00    NaN  NaN  NaN  NaN      46.2
2010-01-01 01:00:00    NaN  NaN  NaN  NaN      44.6
2010-01-01 02:00:00    NaN  NaN  NaN  NaN      44.1
```

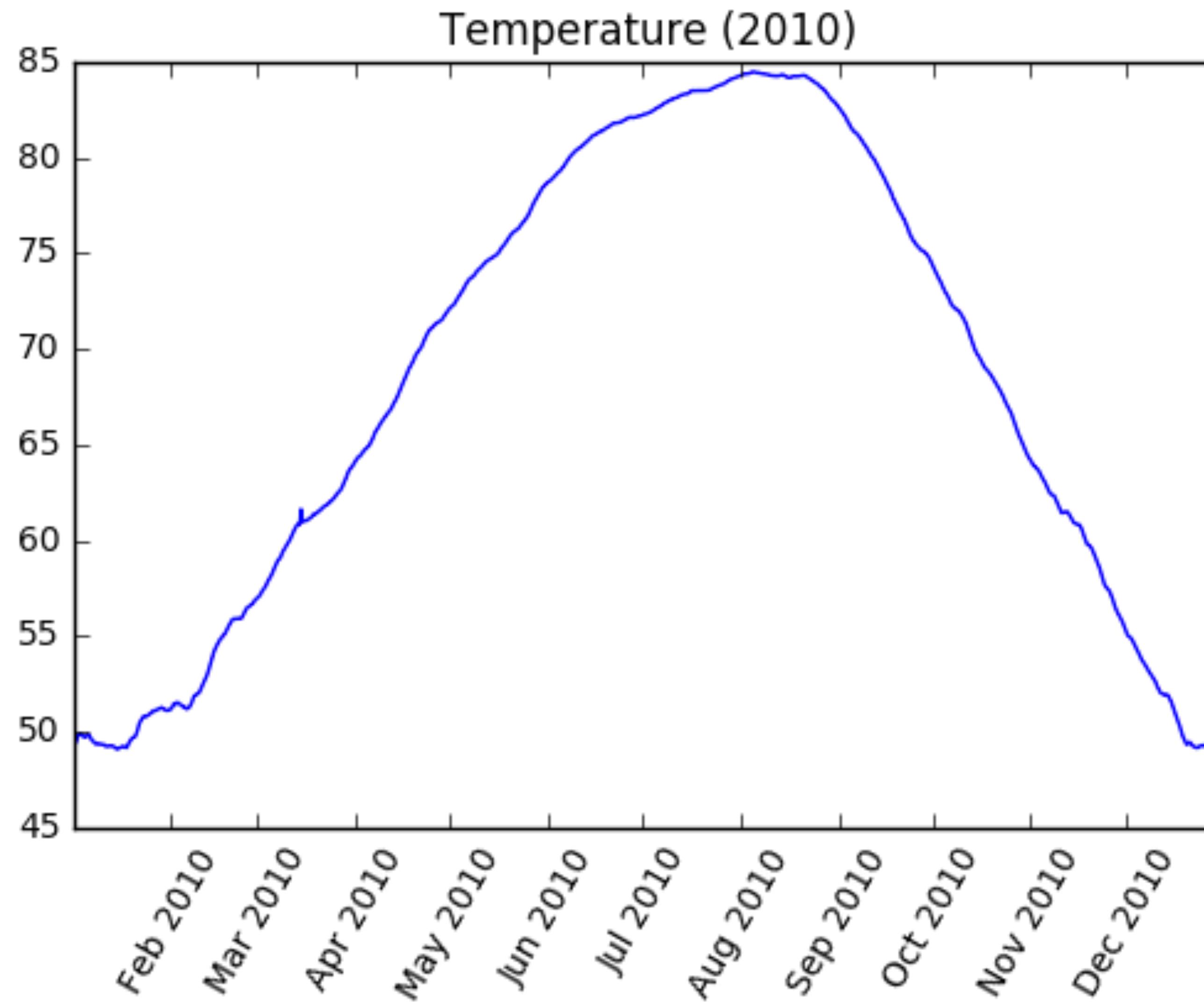


# Viewing 24 hour averages

```
In [1]: plt.plot(smoothed['1d']) # moving average over 24 hours  
  
In [2]: plt.title('Temperature (2010)')  
  
In [3]: plt.xticks(rotation=60)  
  
In [4]: plt.show()
```

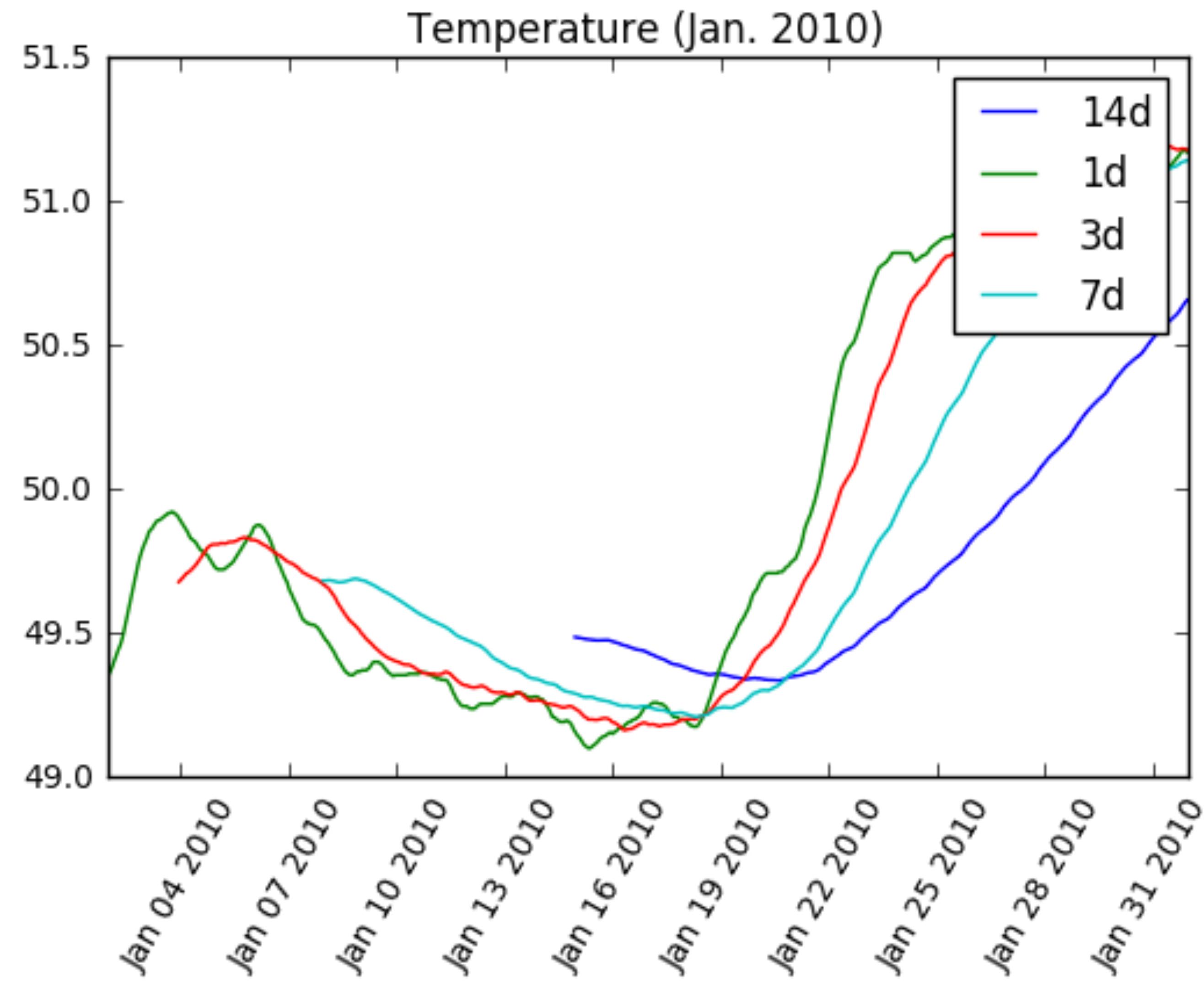


# Viewing 24 hour averages





# Viewing all moving averages





# Viewing all moving averages

```
In [1]: plt.plot(smoothed['2010-01']) # plot  
...: DataFrame for January
```

```
In [2]: plt.legend(smoothed.columns)
```

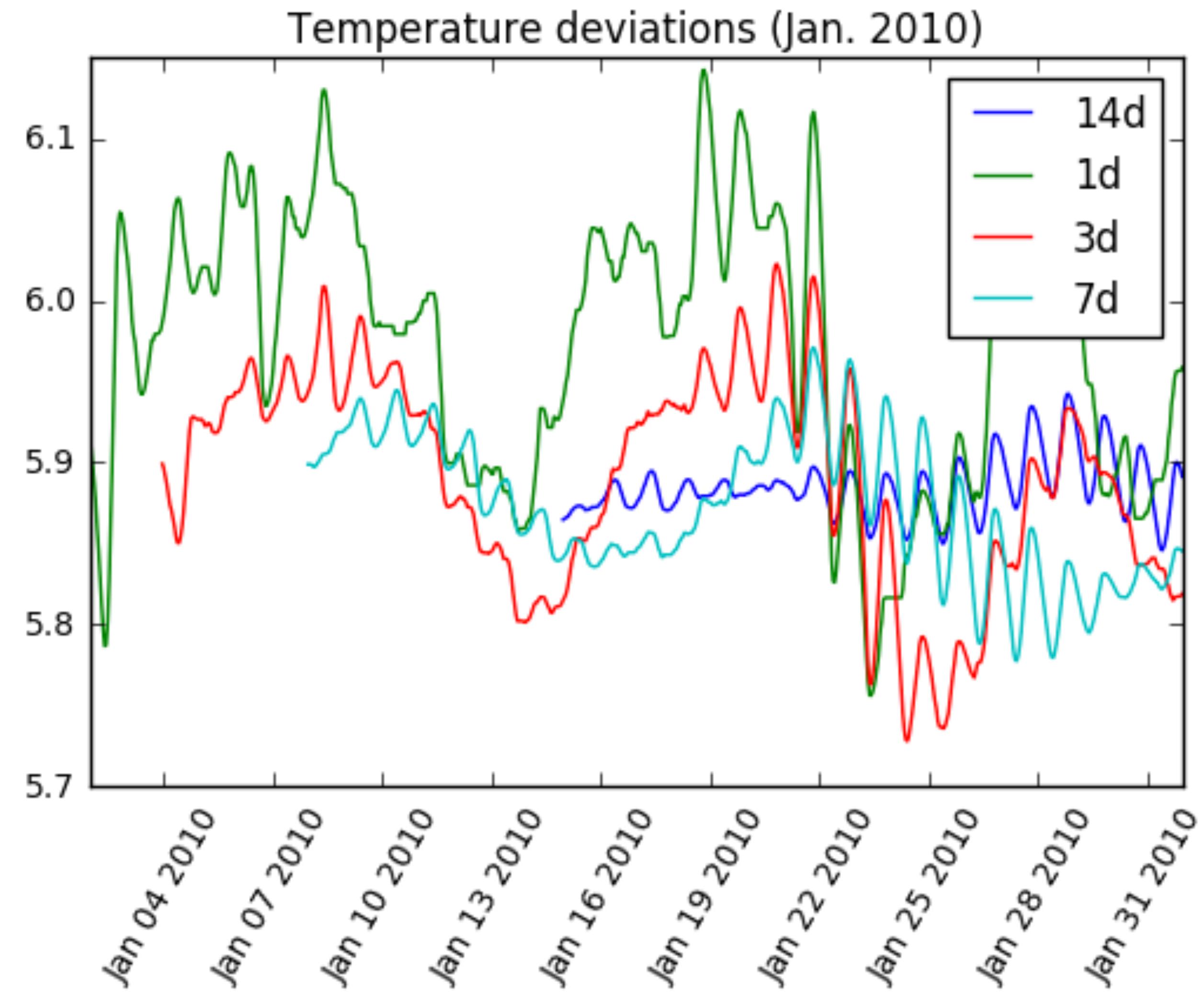
```
In [3]: plt.title('Temperature (Jan. 2010)')
```

```
In [4]: plt.xticks(rotation=60)
```

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



# Moving standard deviations





# Moving standard deviations

```
In [1]: plt.plot(variances['2010-01'])
```

```
In [2]: plt.legend(variances.columns)
```

```
In [3]: plt.title('Temperature deviations (Jan. 2010)')
```

```
In [4]: plt.xticks(rotation=60)
```

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



INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

**Let's practice!**



INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

# Histogram equalization in images



# Original image





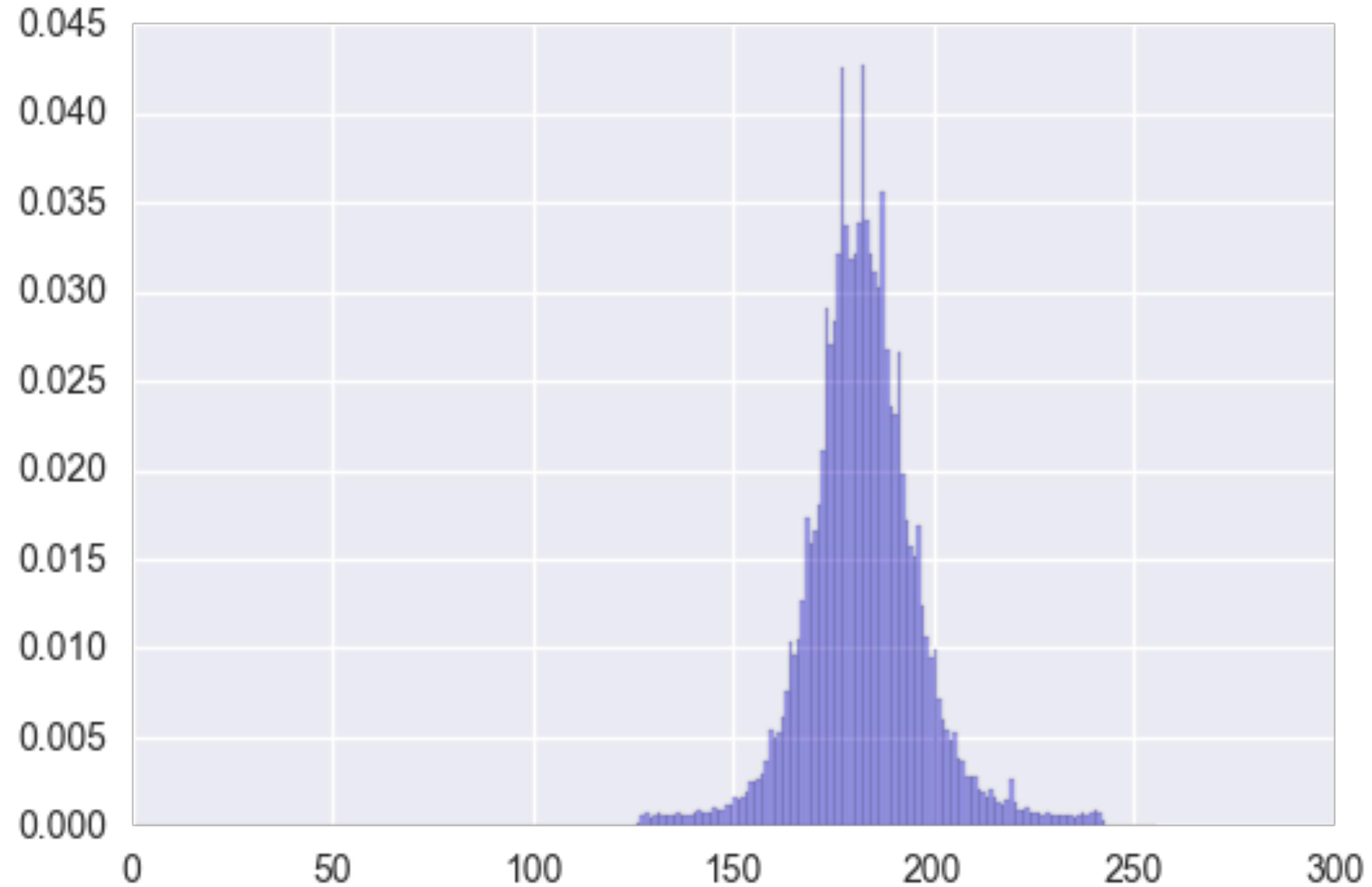
# Equalized image

Equalized image





# Image histograms





# Image histograms

```
In [1]: orig = plt.imread('low-contrast-moon.jpg')
```

```
In [2]: pixels = orig.flatten()
```

```
In [3]: plt.hist(pixels, bins=256, range=(0,256), normed=True,  
...:           color='blue', alpha=0.3)
```

```
In [4]: plt.show()
```

```
In [5]: minval, maxval = orig.min(), orig.max()
```

```
In [6]: print(minval, maxval)  
125 244
```



# Rescaling the image

```
In [1]: minval, maxval = orig.min(), orig.max()
```

```
In [2]: print(minval, maxval)  
125 244
```

```
In [3]: rescaled = (255/(maxval-minval)) * (pixels - minval)
```

```
In [4]: print(rescaled.min(), rescaled.max())  
0.0 255.0
```

```
In [5]: plt.imshow(rescaled)
```

```
In [6]: plt.axis('off')
```

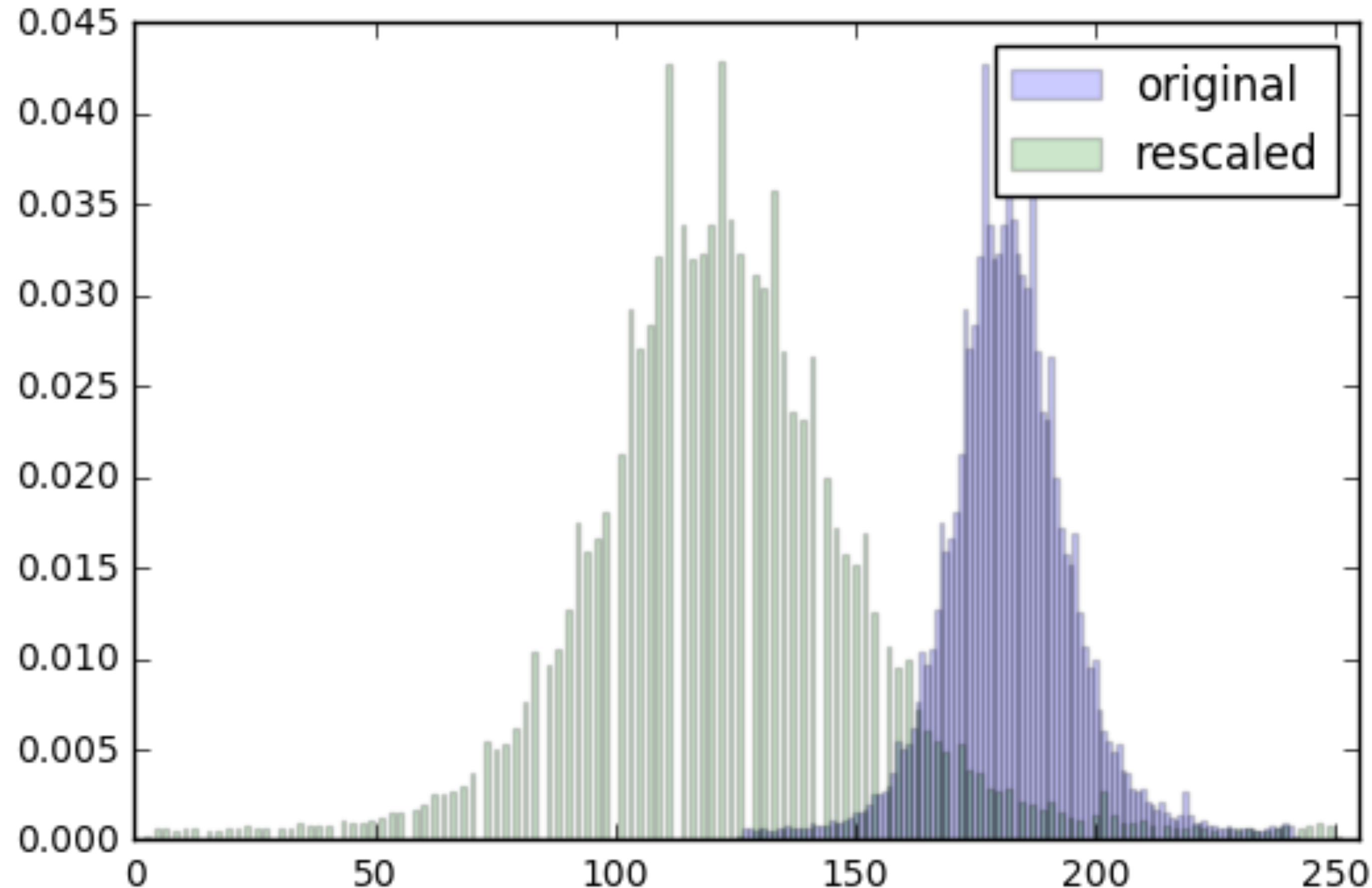
```
In [7]: plt.show()
```



# Rescaled image



# Original & rescaled histograms





# Original & rescaled histograms

```
In [1]: plt.hist(orig.flatten(), bins=256, range=(0,255),  
...:             normed=True, color='blue', alpha=0.2))
```

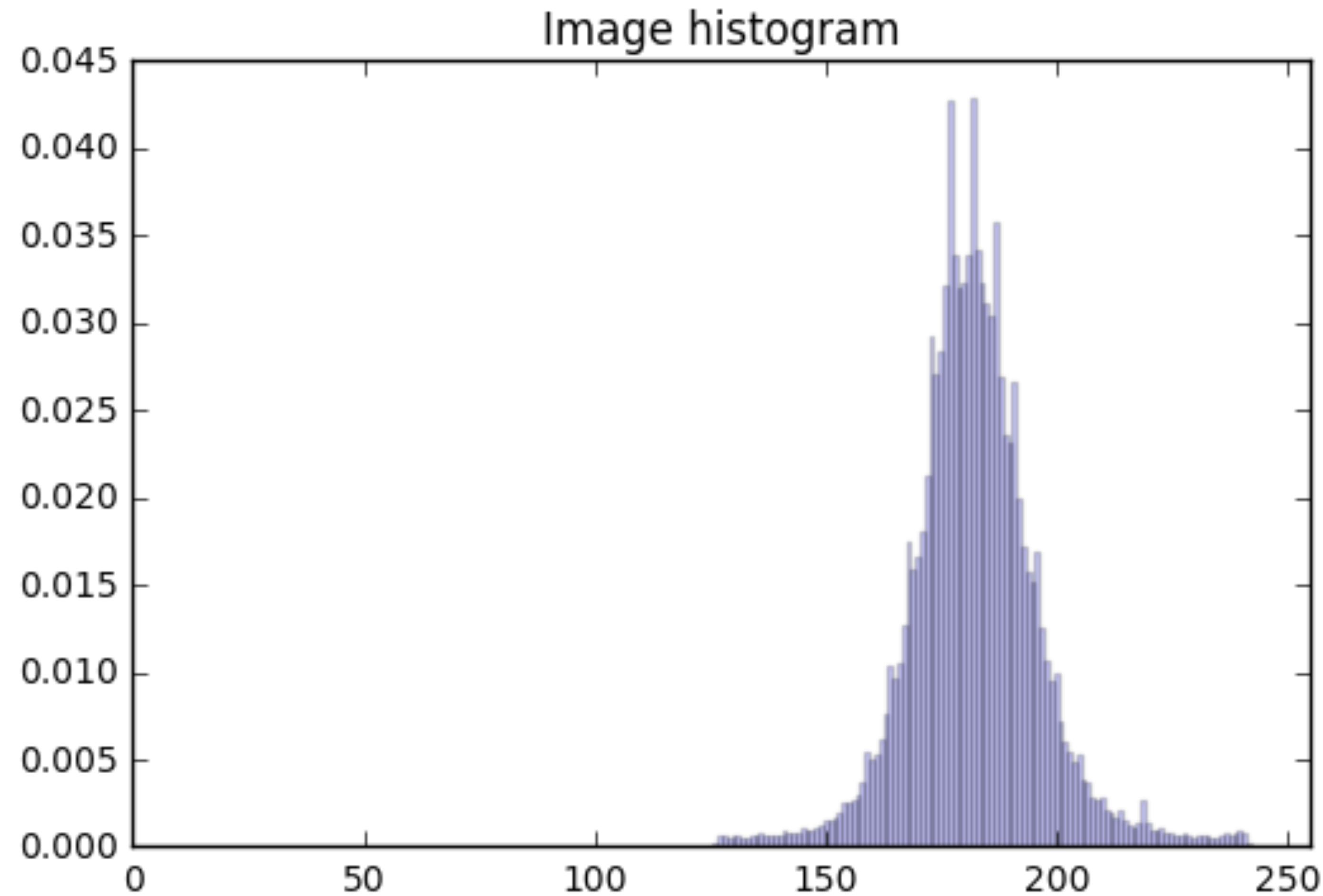
```
In [2]: plt.hist(rescaled.flatten(), bins=256, range=(0,255),  
...:             normed=True, color='green', alpha=0.2))
```

```
In [3]: plt.legend(['original', 'rescaled'])
```

```
In [4]: plt.show()
```

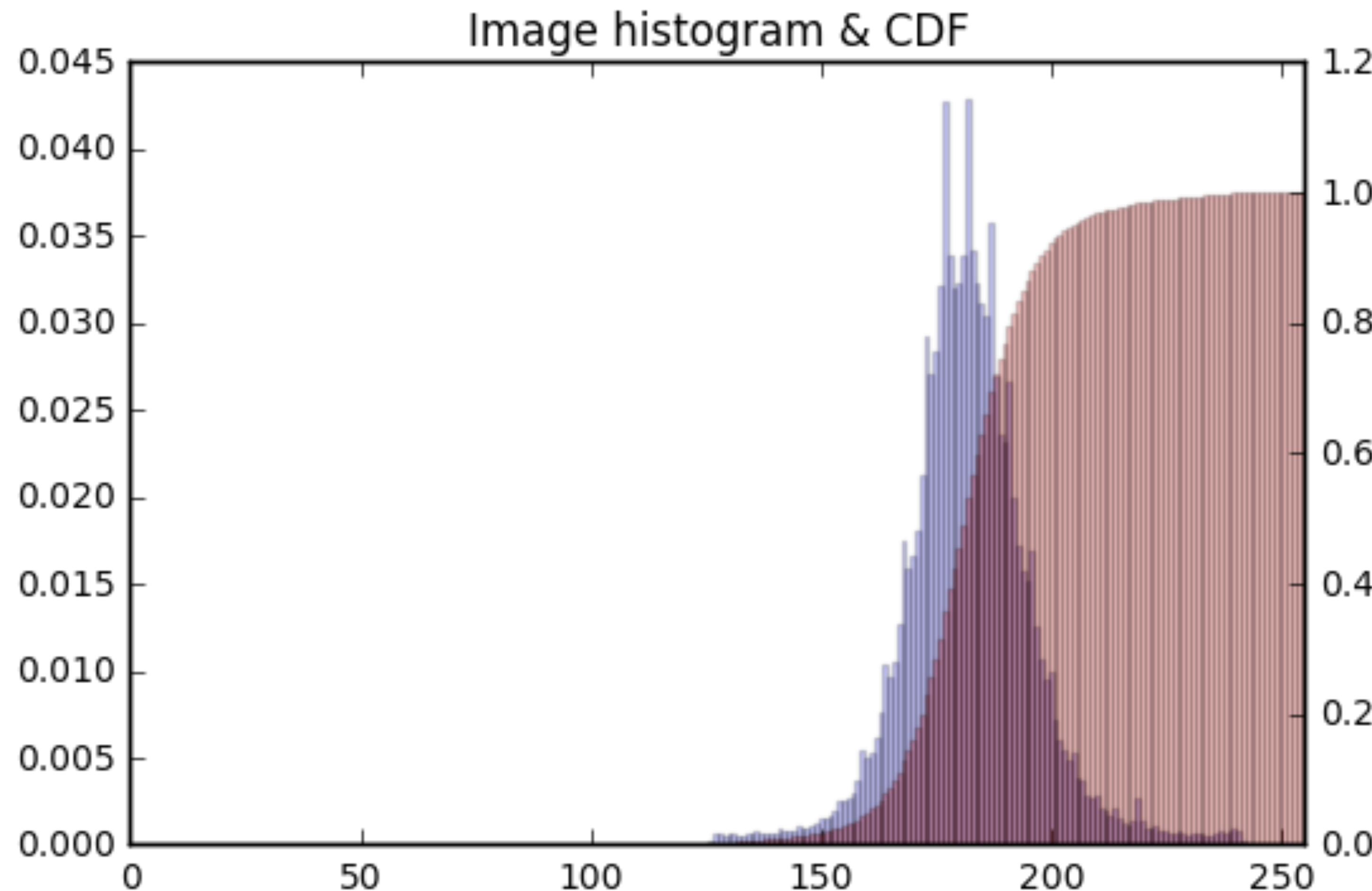


# Image histogram & CDF





# Image histogram & CDF





# Image histogram & CDF

```
In [1]: plt.hist(pixels, bins=256, range=(0,256), normed=True,  
...:           color='blue', alpha=0.3)
```

```
In [2]: plt.twinx()
```

```
In [3]: orig_cdf, bins, patches = plt.hist(pixels,  
...: cumulative=True, bins=256, range=(0,256), normed=True,  
...: color='red', alpha=0.3)
```

```
In [4]: plt.title('Image histogram and CDF')
```

```
In [5]: plt.xlim(0, 255)
```

```
In [6]: plt.show()
```



# Equalizing intensity values

```
In [1]: new_pixels = np.interp(pixels, bins[:-1], orig_cdf*255)
```

```
In [2]: new = new_pixels.reshape(orig.shape)
```

```
In [3]: plt.imshow(new)
```

```
In [4]: plt.axis('off')
```

```
In [5]: plt.title('Equalized image')
```

```
In [6]: plt.show()
```



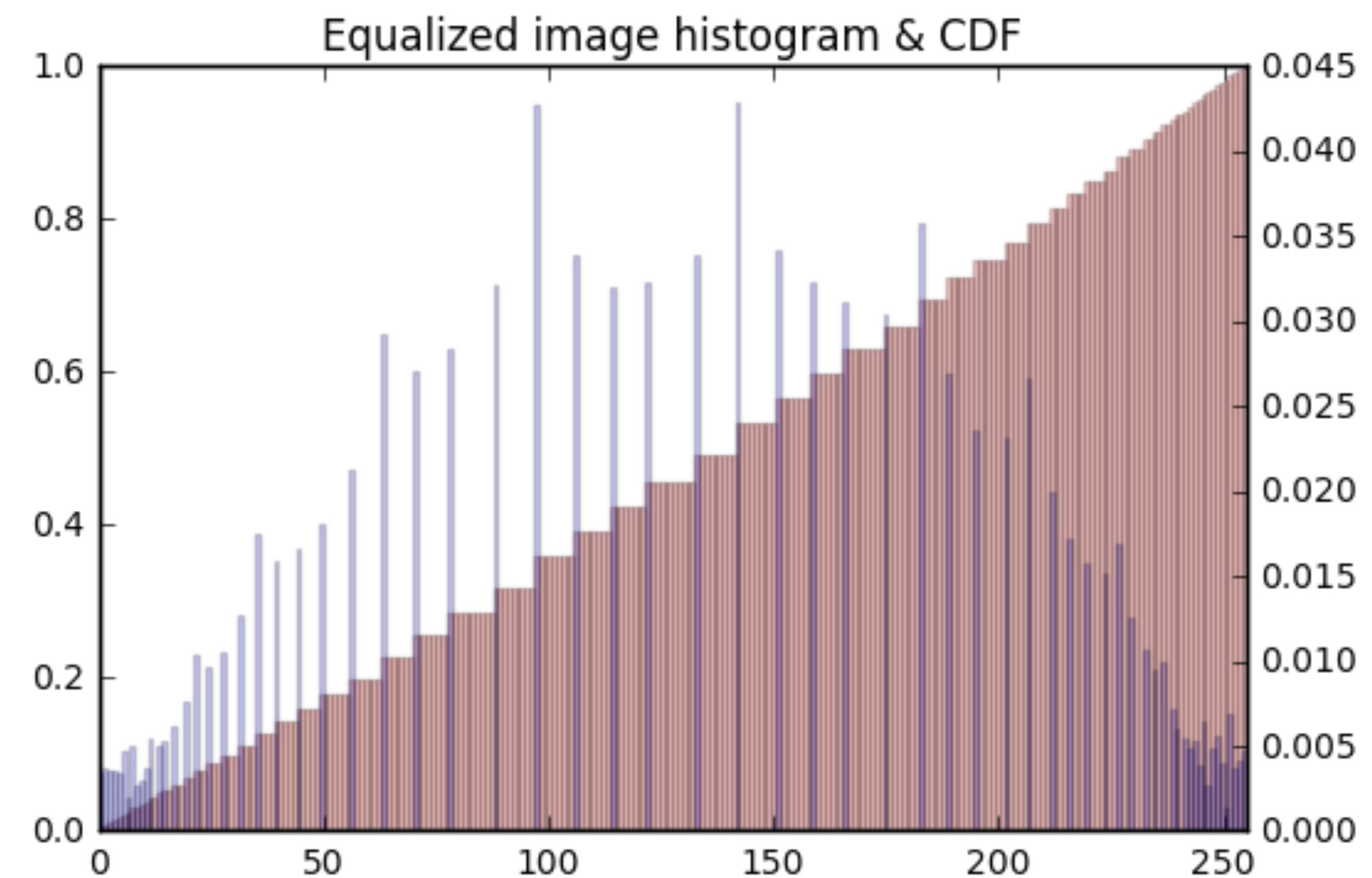
# Equalized image

Equalized image





# Equalized histogram & CDF





# Equalized histogram & CDF

```
In [1]: plt.hist(new_pixels, bins=256, range=(0,256),  
...:             normed=True, color='blue', alpha=0.3)  
  
In [2]: plt.twinx()  
  
In [3]: plt.hist(new_pixels, cumulative=True, bins=256,  
...:             range=(0,256), normed=True, color='red', alpha=0.1)  
  
In [4]: plt.title('Equalized image histogram and CDF')  
  
In [5]: plt.xlim(0, 255)  
  
In [6]: plt.show()
```



INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

**Let's practice!**



INTRODUCTION TO DATA VISUALIZATION WITH PYTHON

# Congratulations!