Visualization

Thomas Schwarz, SJ

Data Visualization

- We use Matplotlib
  - Import with standard names

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

  - Can set style

```
plt.style.use('classic')
```

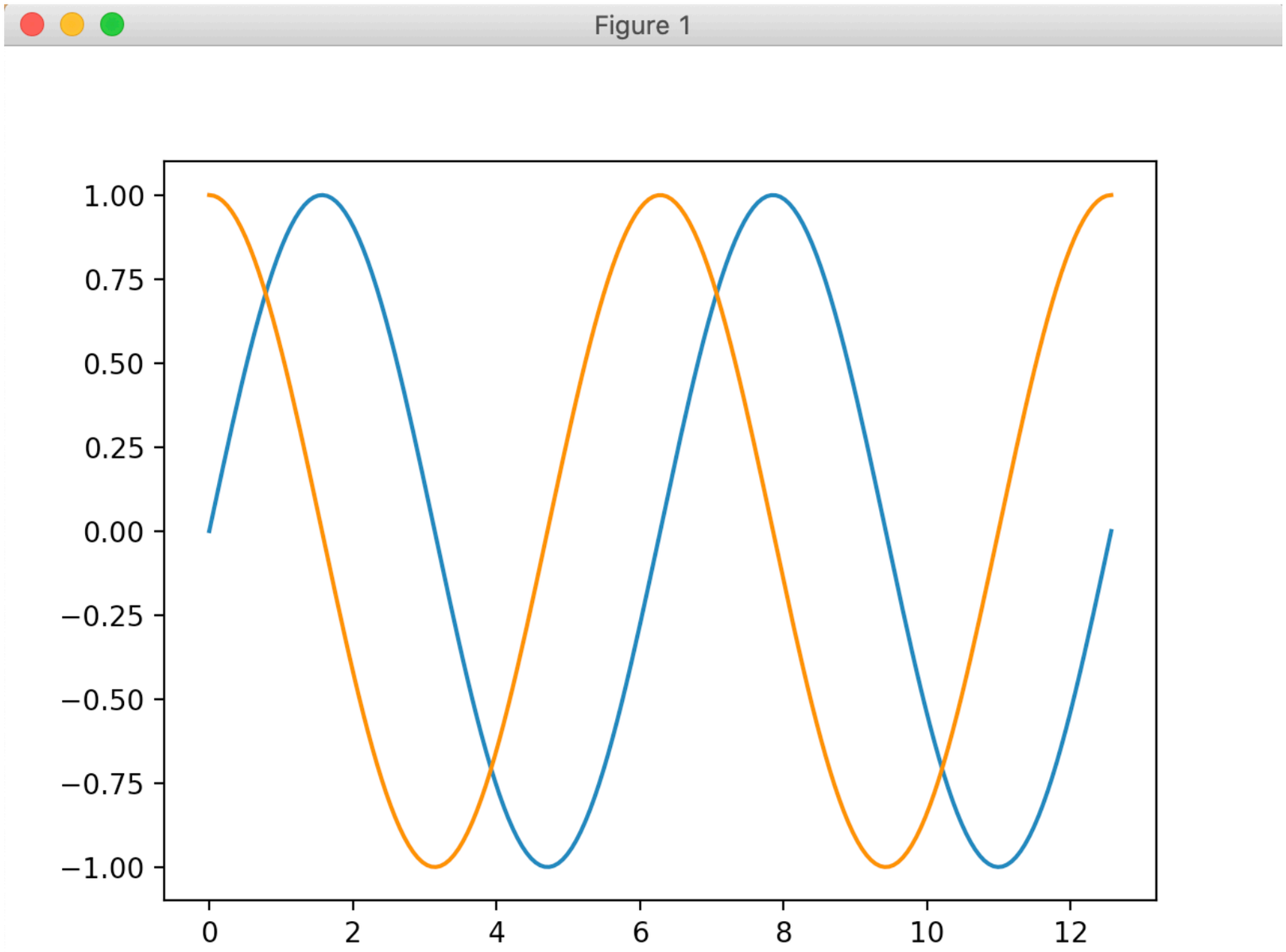
Matplotlib

- Display your results from a script
  - Use `plt.show( )`
    - Starts an event loop
    - Looks at all figure objects and opens interactive windows

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

x = np.linspace(0, 4*math.pi, 200)
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
plt.show()
```

Matplotlib



Matplotlib

- Plotting from iPython
  - Need to specify matplotlib mode
    - `In [1]: %matplotlib`
    - `In [2]: import matplotlib.pyplot as plt`
  - Specify `%matplotlib notebook`
    - Interactive plots embedded in notebook
  - Specify %matplotlib interactive
    - static images of our plots

Matplotlib

- Can save figures with the `savefig('myfigure.png')` command
  - File format inferred from the extension

Histograms

- Let's use the iris data set
  - A pre-processed version is in `sklearn.datasets`
    - Which means importing it

```
iris = load_iris()
features = iris.data.T
```

  - We better look at it first:

```
>>> features[:,:10]
array([[5.1, 4.9, 4.7, 4.6, 5. , 5.4, 4.6, 5. , 4.4, 4.9],
       [3.5, 3. , 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1],
       [1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5],
       [0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1]])
```

Histograms

- Let's create histograms for all four properties of an Iris set
  - We can define a simple figure with four different panels

```
fig, axs = plt.subplots(2,2,  squeeze = True)
```

- We then load the axes elements

```
axs[0][0]          axs[0][1]

axs[1][0]          axs[1][1]
```

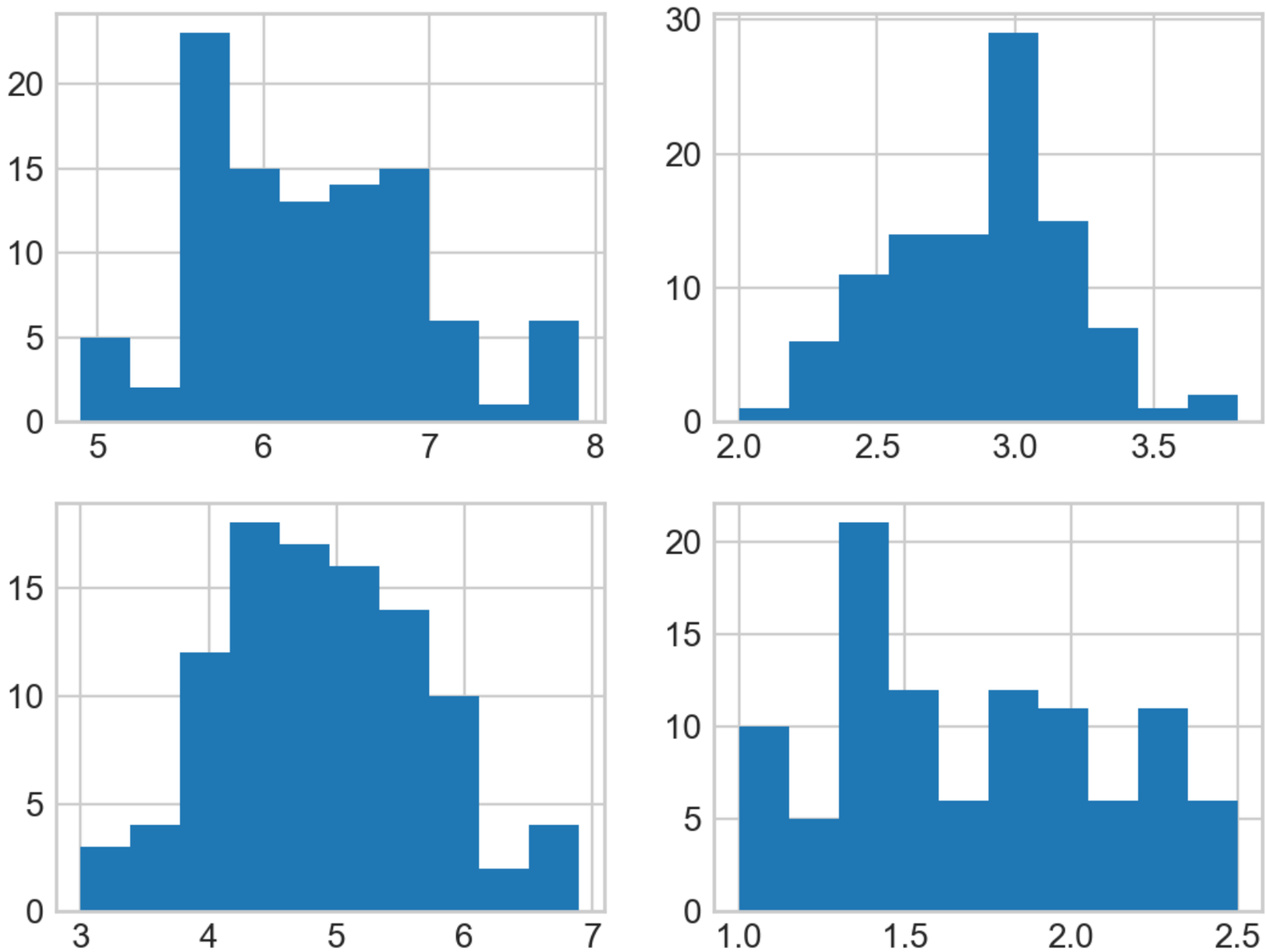
Histograms

- For a histogram, we use the hist method of an axes
  - Needs a np.array and a number of bins

```
axs[0][0].hist(features[0,50:150], bins = 10)
axs[0][1].hist(features[1,50:150], bins = 10)
axs[1][0].hist(features[2,50:150], bins = 10)
axs[1][1].hist(features[3,50:150], bins = 10)
```

Histograms

- Result so far:



Histograms

- The axs objects are actually a tuple
  - N, bins, patches
    - with N the count in each bin
    - bins the number in each bin
    - patches gives us access to the properties drawn

Histograms

- Here is how we get at them

```
N00,bins00,patches00 = axs[0][0].hist(features[0,50:150], bins=10)
N01,bins01,patches01 = axs[0][1].hist(features[1,50:150], bins=10)
N10,bins10,patches10 = axs[1][0].hist(features[2,50:150], bins=10)
N11,bins11,patches11 = axs[1][1].hist(features[3,50:150], bins=10)
```

- And we can see what is in them
  - N00, bins00

```
array([ 5.,  2., 23., 15., 13., 14., 15.,  6.,  1.,  6.])
array([4.9, 5.2, 5.5, 5.8, 6.1, 6.4, 6.7, 7. , 7.3, 7.6, 7.9])
```

Histograms

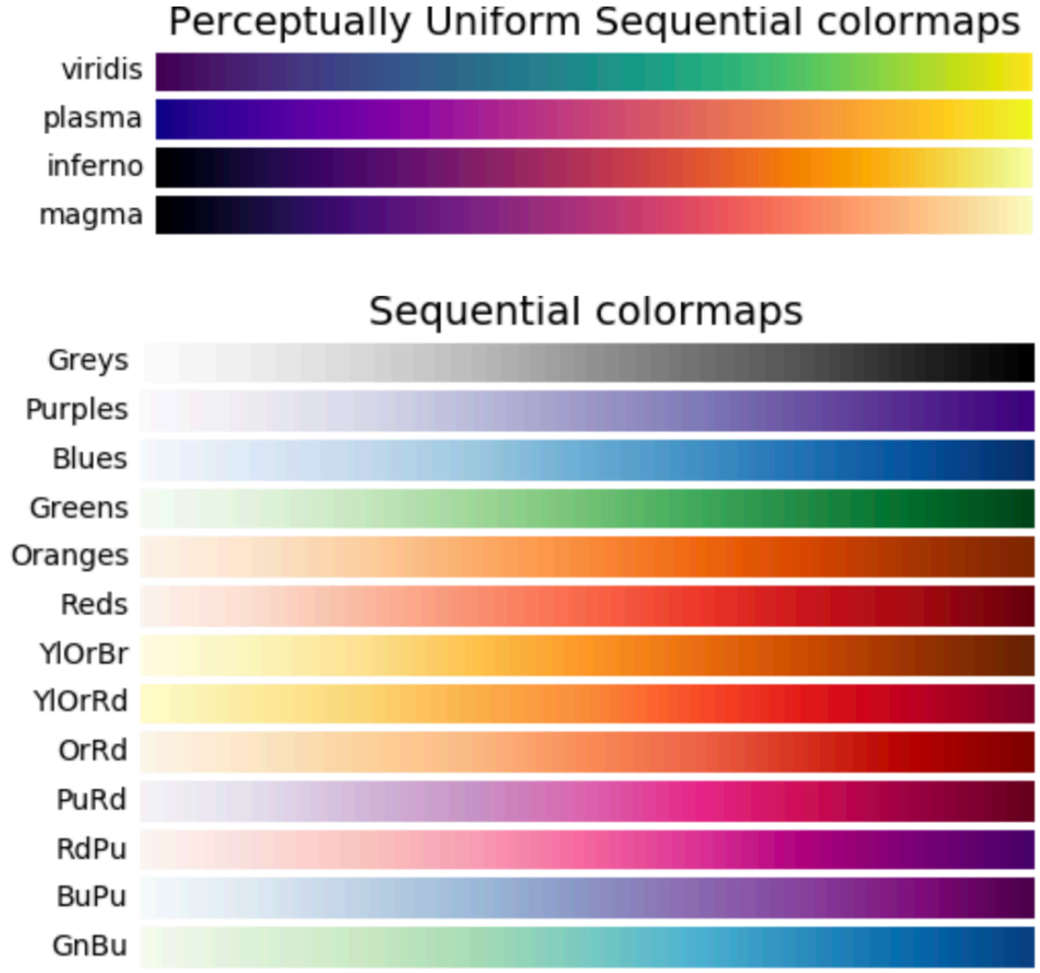
- The patches is a list of patch objects, one for each bin rectangle
- We can for example update the color
  - Use `patch.set_facecolor(color)`
    - Where color is a number between 0 and 256
    - Which we can select according to colormaps

Colormaps

matplotlib

color example code: colormaps_reference.py

(Source code)



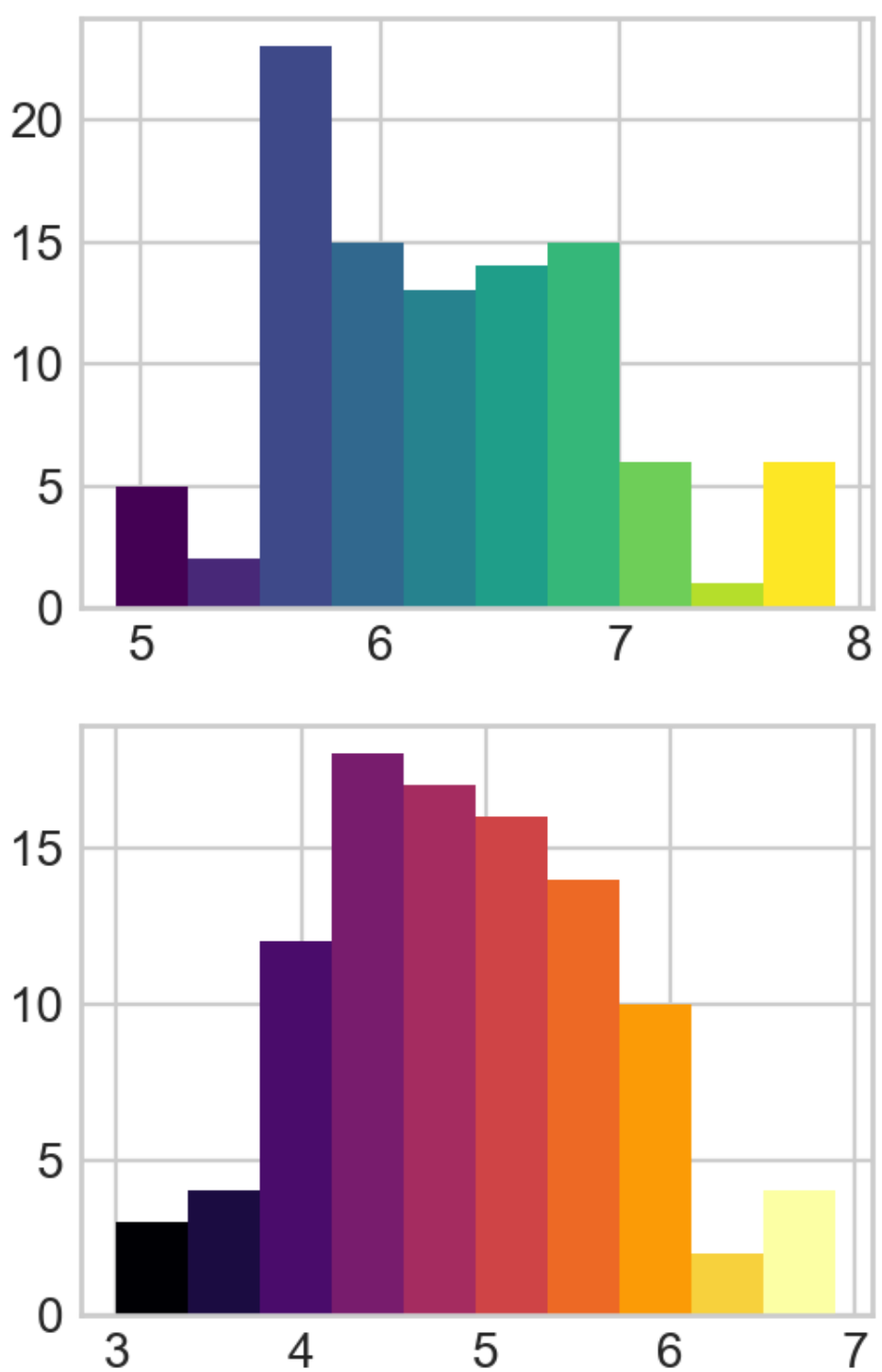
Histograms

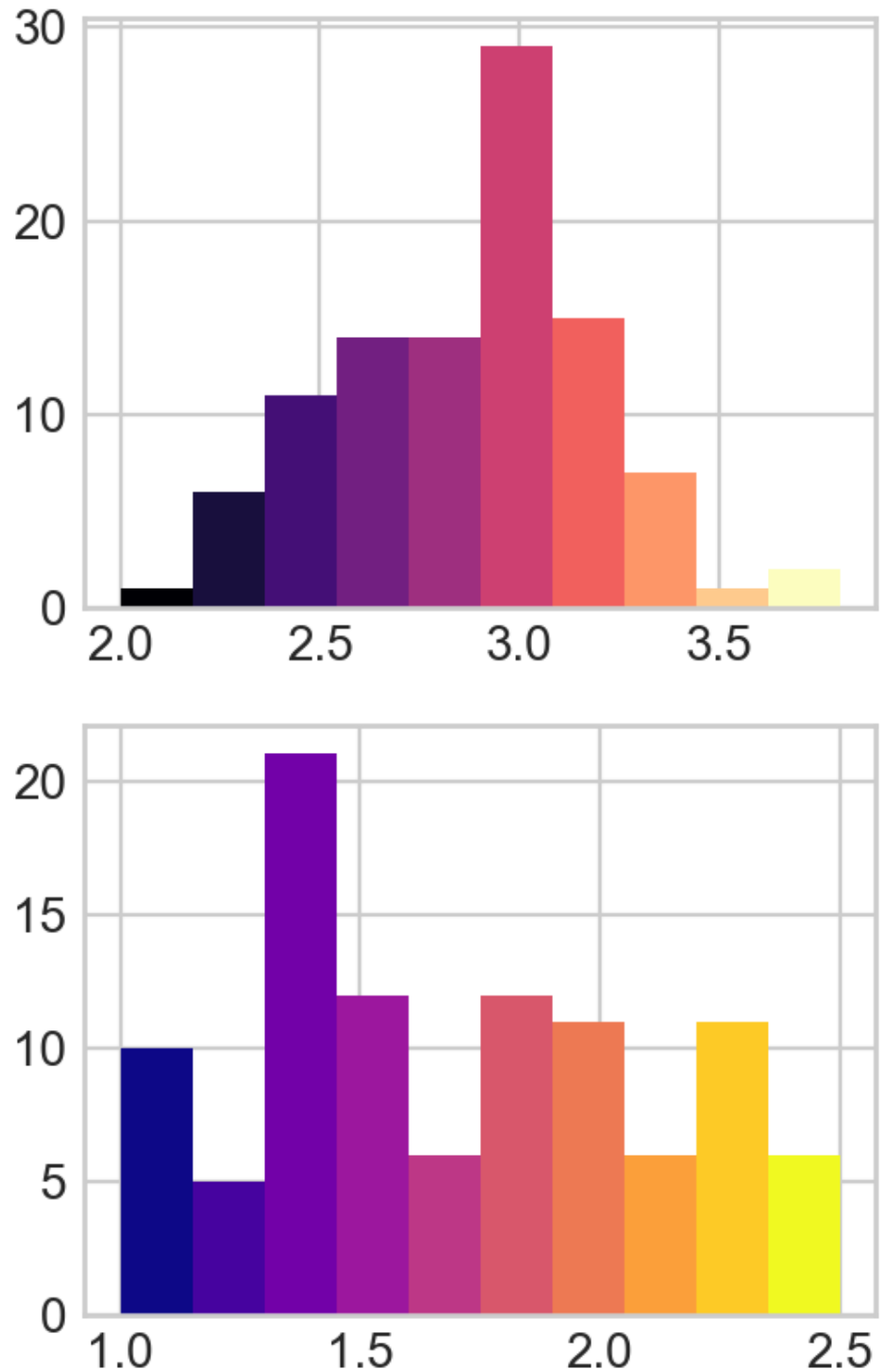
- For each of the four histograms, we can use a different color map
  - We pick the number uniformly through 1 … 256

```
for i, patch in enumerate(patches00):
    color = plt.cm.viridis(i*256//9)
    patch.set_facecolor(color)

for i, patch in enumerate(patches01):
    color = plt.cm.magma(i*256//9)
    patch.set_facecolor(color)
```

Histograms


20
15
10
5
0
5
6
7
8


30
20
10
0
2.0
2.5
3.0
3.5

15
10
5
0
3
4
5
6
7

20
15
10
5
0
1.0
1.5
2.0
2.5

Python enumerate

- Remember that Python has some great list tools
  - `enumerate(a_list_or_sequence)`
  - will generate tuples of index and element

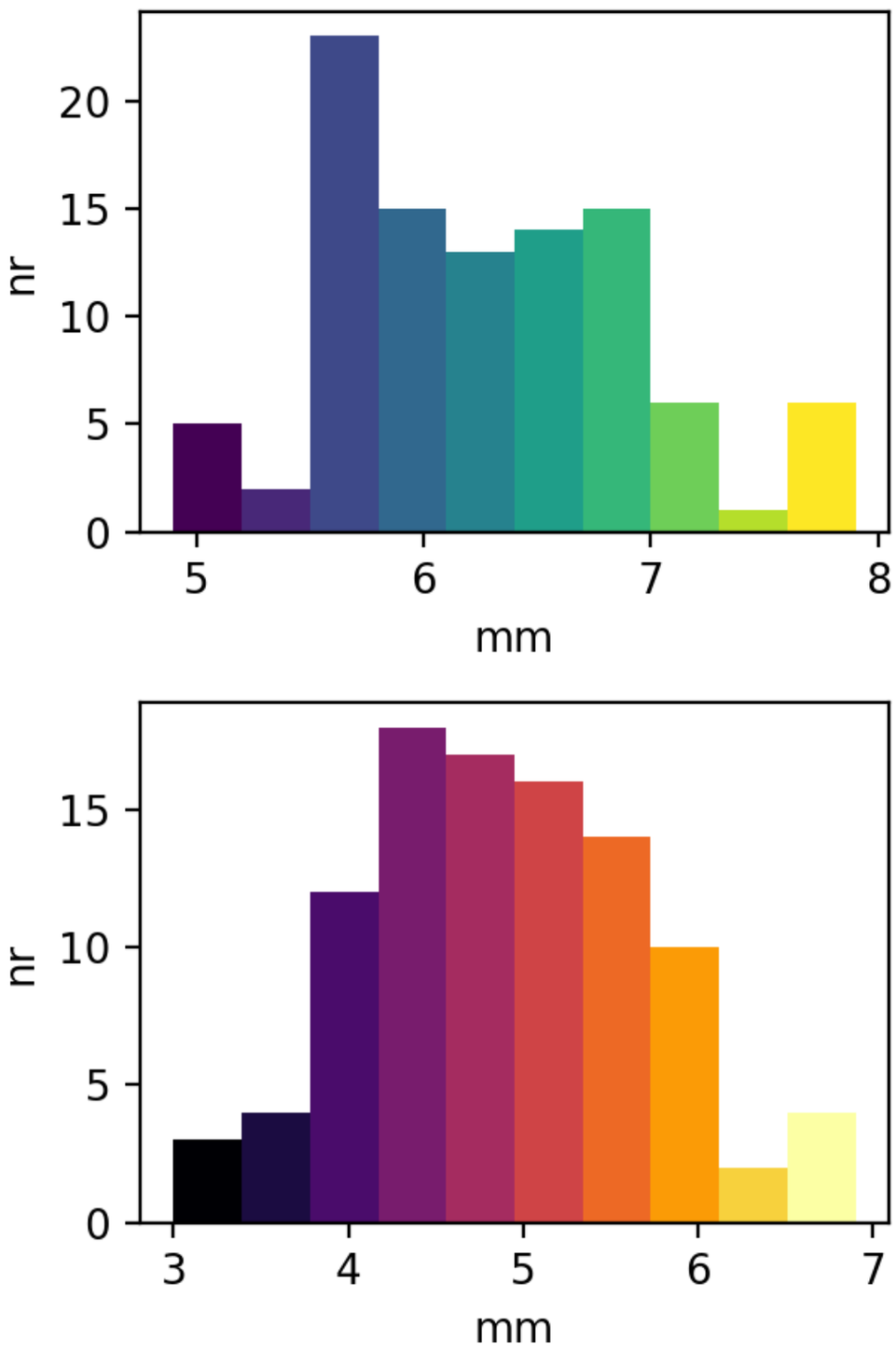
Histograms

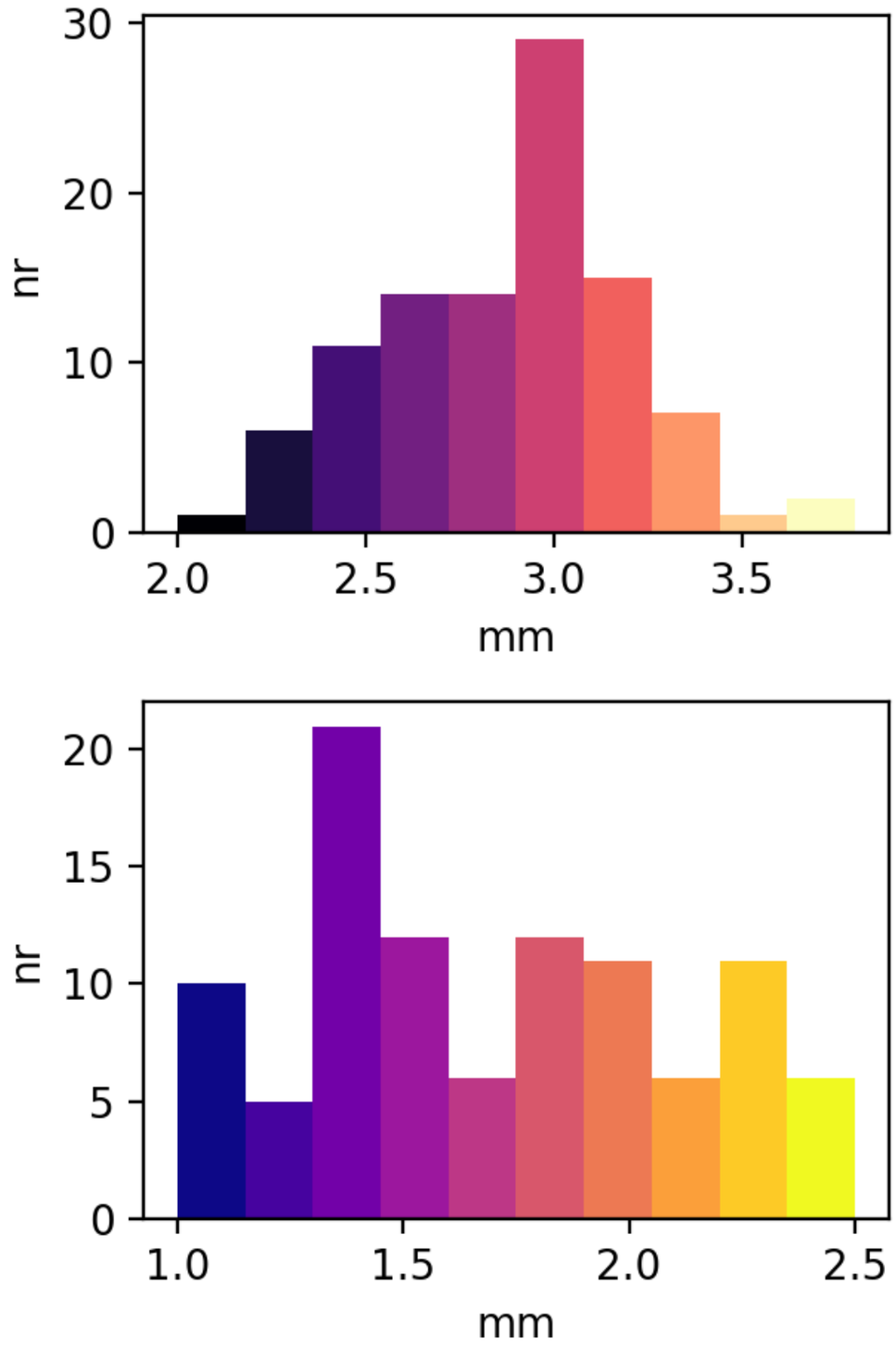
- We should describe what we are displaying
  - First, let's add labels for the axes
    - Turns out that the rows are too close together
    - Search the web: can be adjusted with tight_layout

```
for ax in axs:
    for a in ax:
        a.set_xlabel('mm')
        a.set_ylabel('nr')

fig.tight_layout(pad=1.0)
```

Histograms


20
15
10
5
0
nr
5
6
7
8
mm


30
20
10
0
nr
2.0
2.5
3.0
3.5
mm

15
10
5
0
nr
3
4
5
6
7
mm

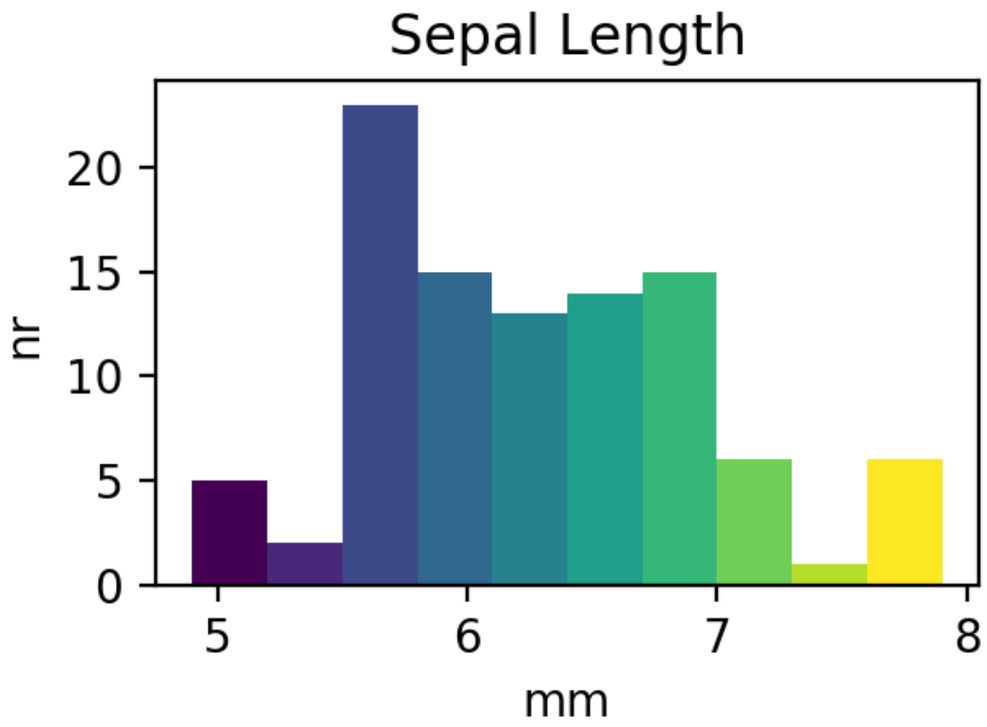
20
15
10
5
0
nr
1.0
1.5
2.0
2.5
mm

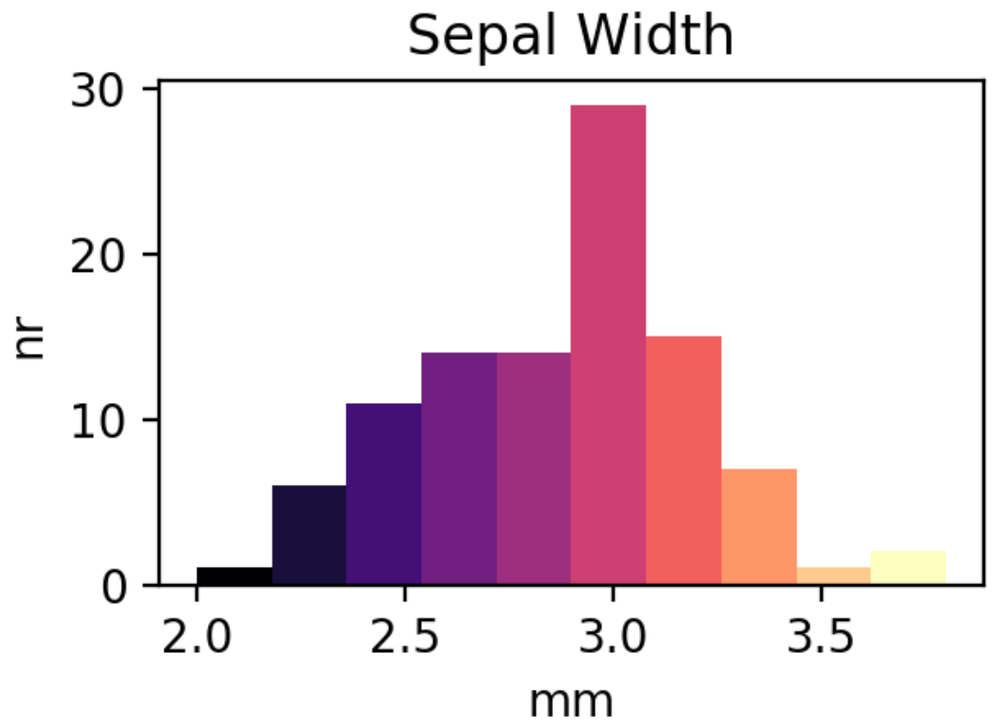
Histograms

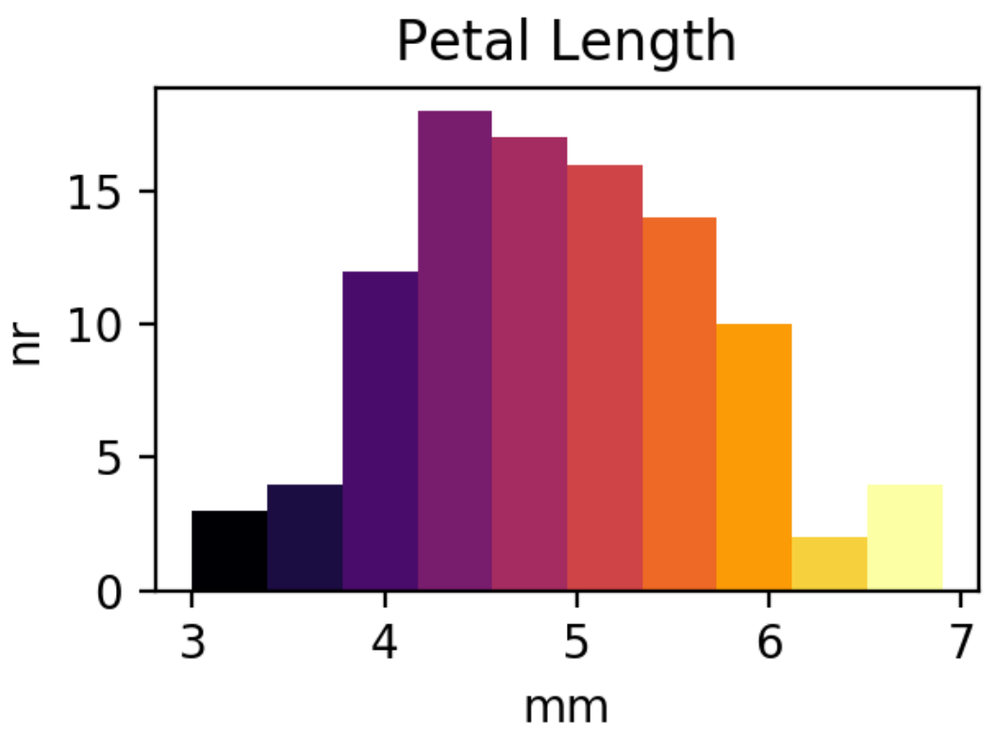
```
for ax in axs:
    for a in ax:
        a.set_xlabel('mm')
        a.set_ylabel('nr')
axs[0][0].set_title('Sepal Length')
axs[0][1].set_title('Sepal Width')
axs[1][0].set_title('Petal Length')
axs[1][1].set_title('Petal Width')

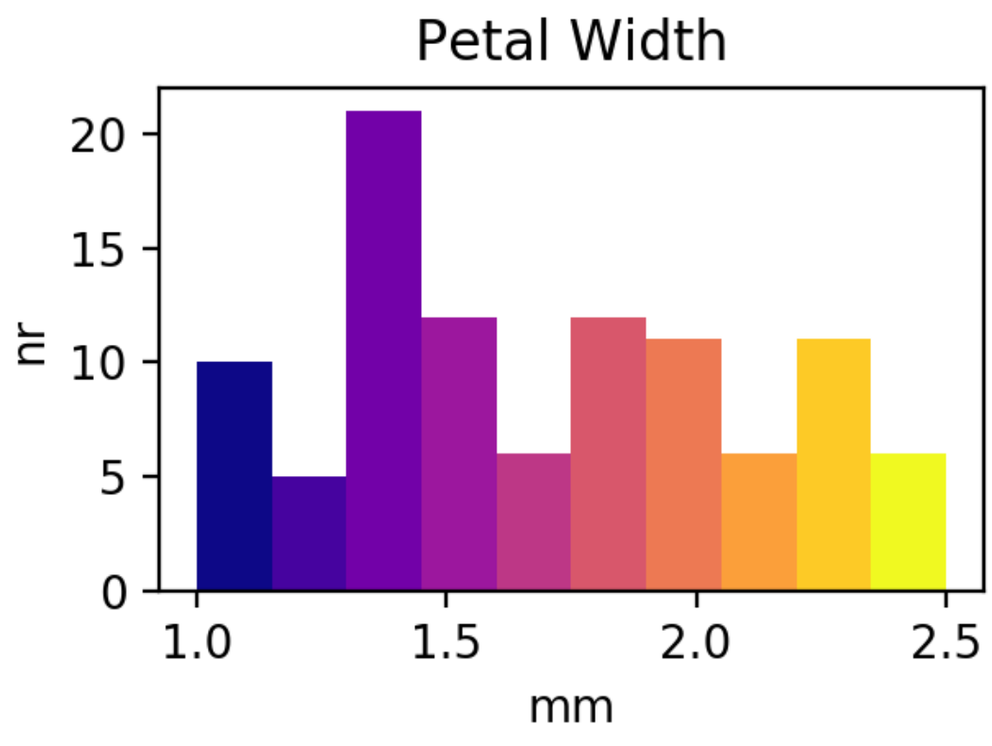
fig.tight_layout(pad=1.0)
```

Histograms


Sepal Length
nr
20
15
10
5
0
5
6
7
8
mm


Sepal Width
nr
30
20
10
0
2.0
2.5
3.0
3.5
mm


Petal Length
nr
15
10
5
0
3
4
5
6
7
mm


Petal Width
nr
20
15
10
5
0
1.0
1.5
2.0
2.5
mm

2D-Histograms

- Two-dimensional histograms use color to show the numbers into a two dimensional bin

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

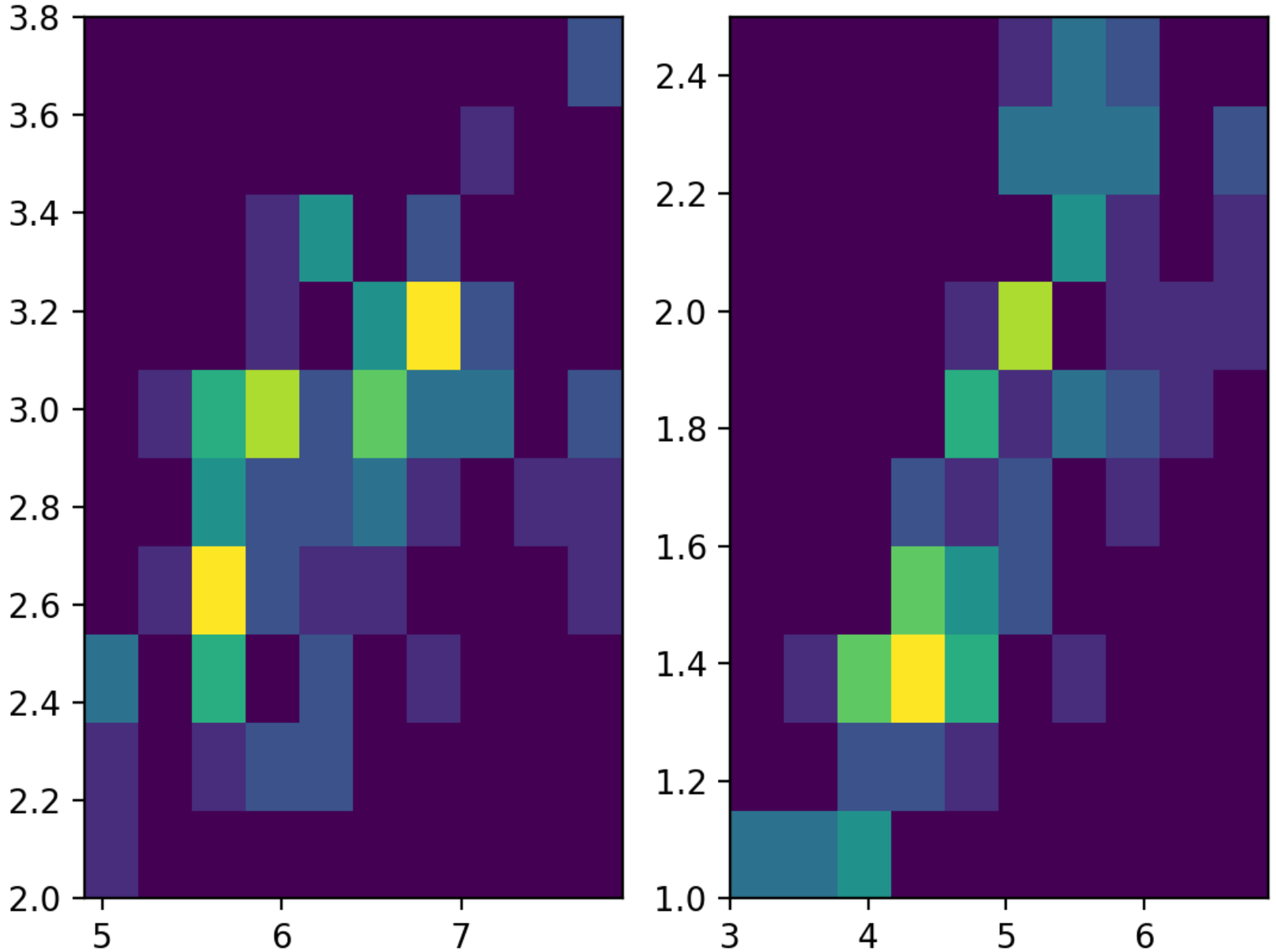

iris = load_iris()
features = iris.data.T

fig, axs = plt.subplots(1,2)
axs[0].hist2d(features[0,50:150],features[1,50:150])
axs[1].hist2d(features[2,50:150],features[3,50:150])

plt.show()
```

2D-Histograms

- The result is harder to read:



2D-Histograms

- We can adjust the number of bins on each side
- And adjust the color scheme
  - `from matplotlib import colors`
  - set color map to something easily interpretable

2D-Histograms

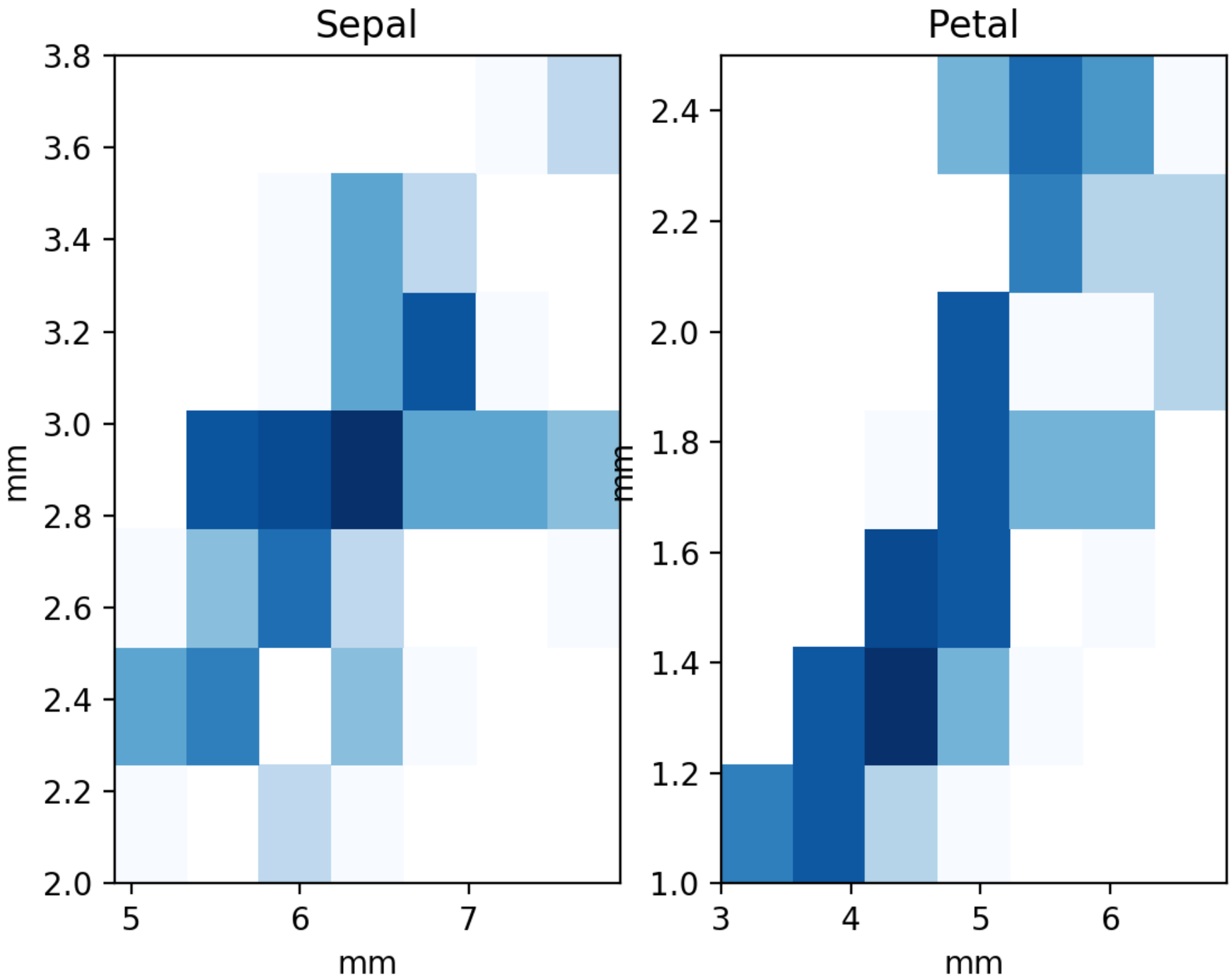
```
fig, axs = plt.subplots(1,2)
axs[0].hist2d(features[0,50:150],
              features[1,50:150],
              bins=7,
              norm=colors.LogNorm(),
              cmap = 'Blues')
axs[1].hist2d(features[2,50:150],
              features[3,50:150],
              bins=7,
              norm=colors.LogNorm(),
              cmap = 'Blues')
```

2D-Histograms

- Then add labels and titles

```
for i in range(2):
    axs[i].set_xlabel('mm')
    axs[i].set_ylabel('mm')
axs[0].set_title('Sepal')
axs[1].set_title('Petal')
```

2D-Histograms



2D-Histograms

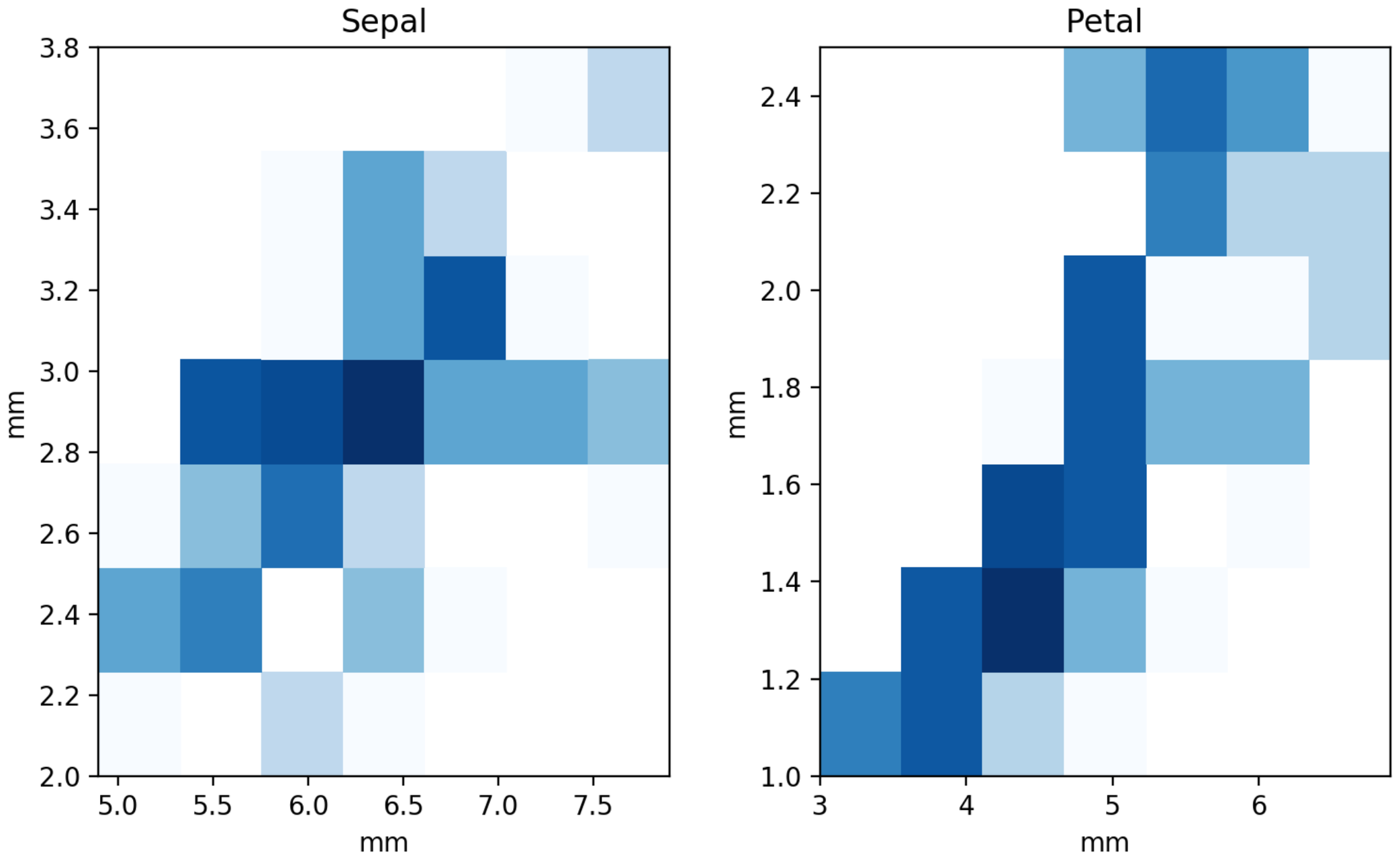
- Need to adjust padding

```
for a in axs:
    a.set_xlabel('mm')
    a.set_ylabel('mm')

axs[0].set_title('Sepal')
axs[1].set_title('Petal')

fig.tight_layout(pad=1.0)
```

2D-Histograms



Plot Legends

- Let's create a simple plot: compare arctan and the logistic functions
- Provide a simple legend
- Give a label to the plot
- call legend and the axes object

Plot Legends

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

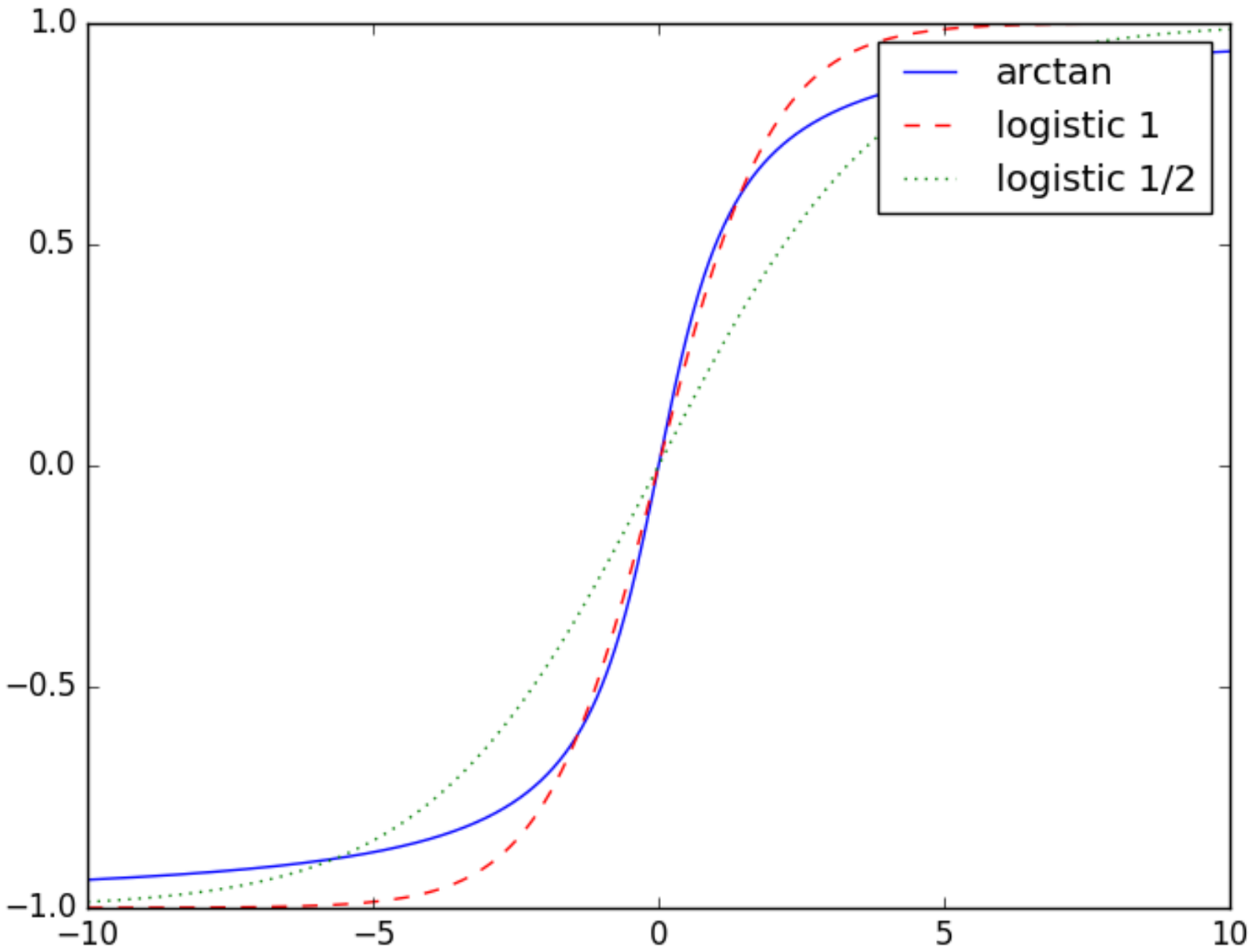
plt.style.use('classic')
```

```
x = np.linspace(-10,10,1001)
fig, ax = plt.subplots()
ax.plot(x, 2*np.arctan(x)/math.pi, 'b-', label='arctan')
ax.plot(x, 2/(np.exp(-x)+1)-1, 'r--', label='logistic 1')
ax.plot(x, 2/(np.exp(-x/2)+1)-1, 'g:', label='logistic 1/2')

ax.legend()

plt.show()
```

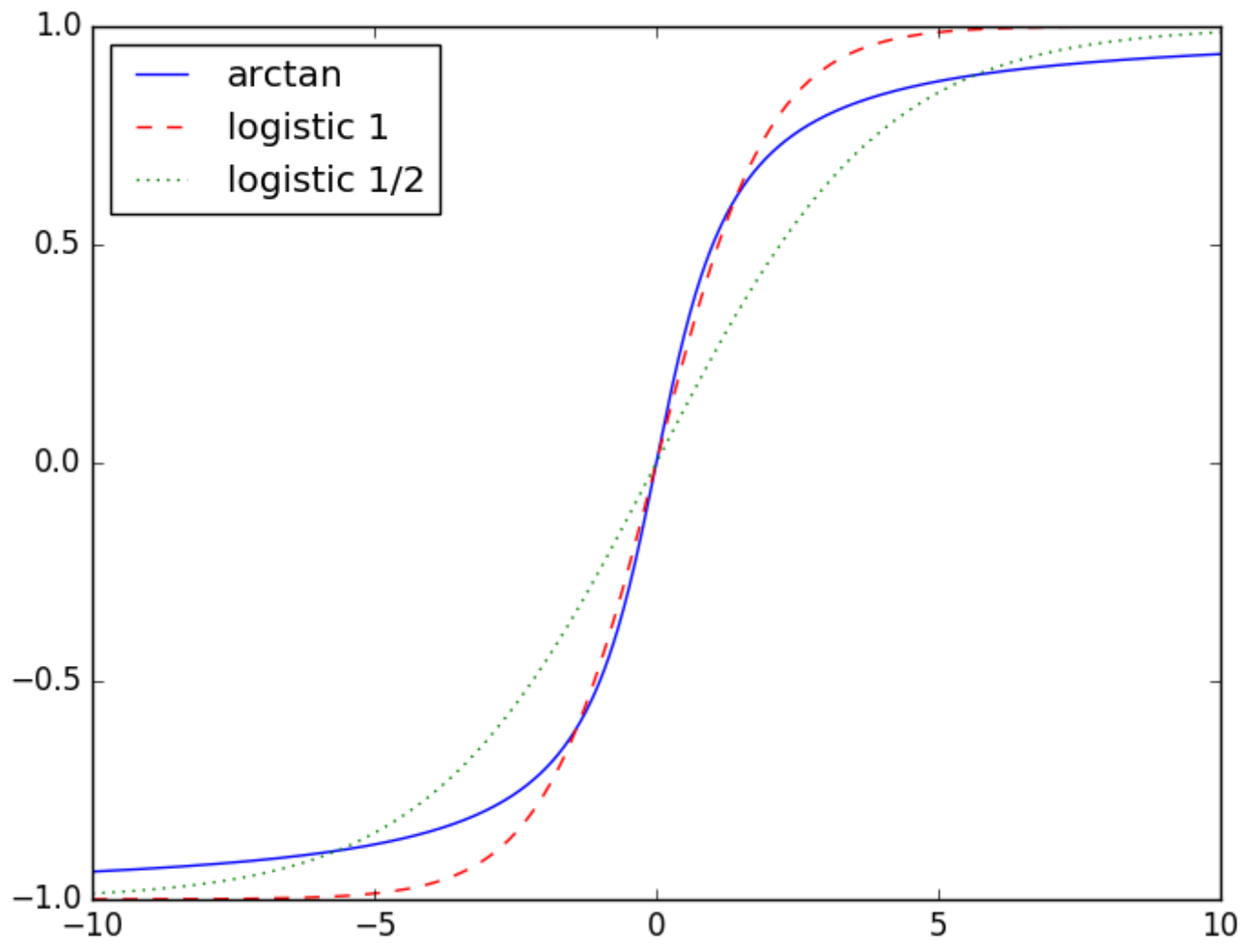
Plot Legends



Plot Legends

- Can specify placement of the legend

```
ax.legend(loc='upper left')
```



Plot Legends

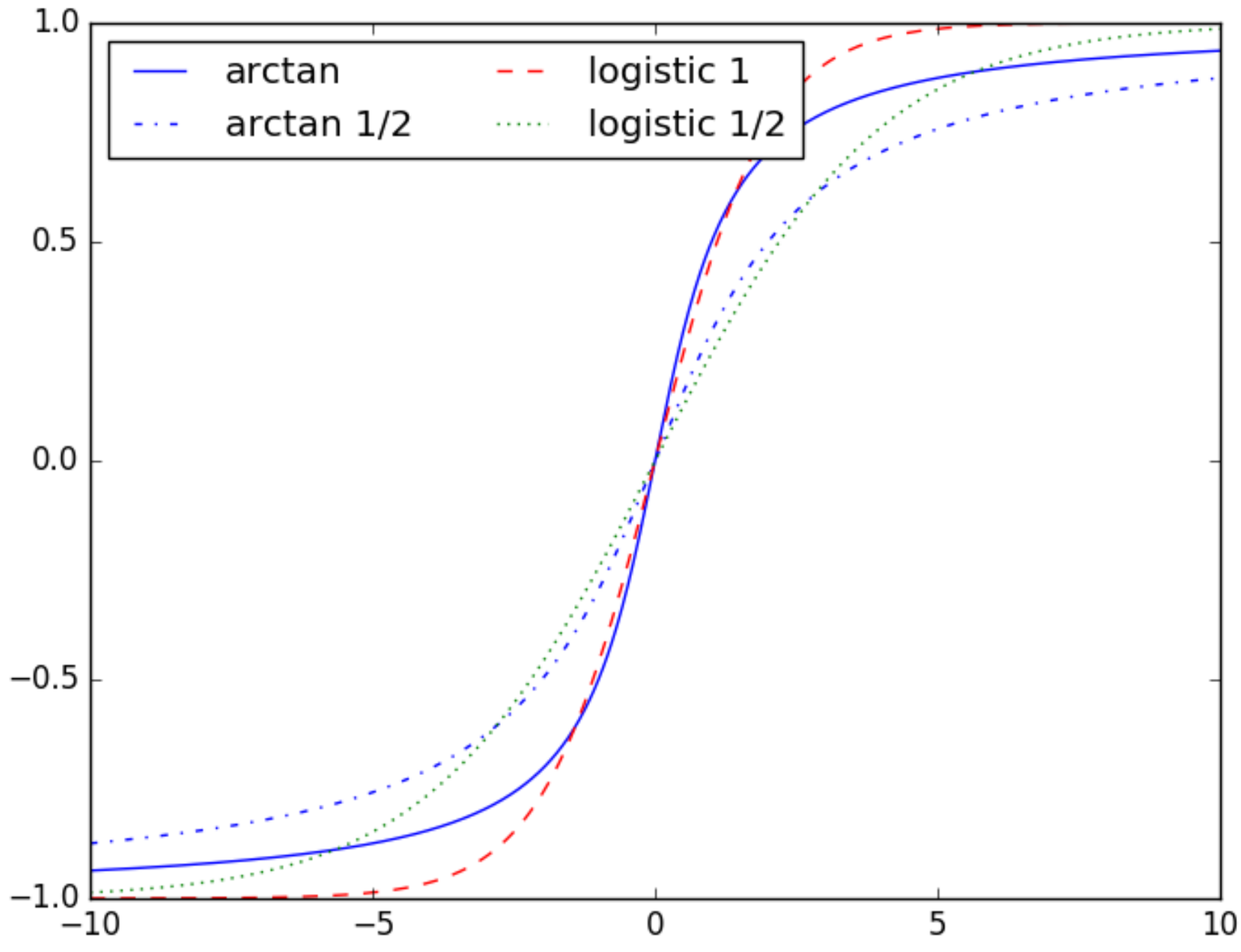
- Can specify the number of columns in the legend

```
x = np.linspace(-10,10,1001)
fig, ax = plt.subplots()
ax.plot(x, 2*np.arctan(x)/math.pi, 'b-', label='arctan')
ax.plot(x, 2*np.arctan(x/2)/math.pi, 'b-.', label='arctan 1/2')
ax.plot(x, 2/(np.exp(-x)+1)-1, 'r--', label='logistic 1')
ax.plot(x, 2/(np.exp(-x/2)+1)-1, 'g:', label='logistic 1/2')


ax.legend(loc='upper left', ncol = 2)

plt.show()
```

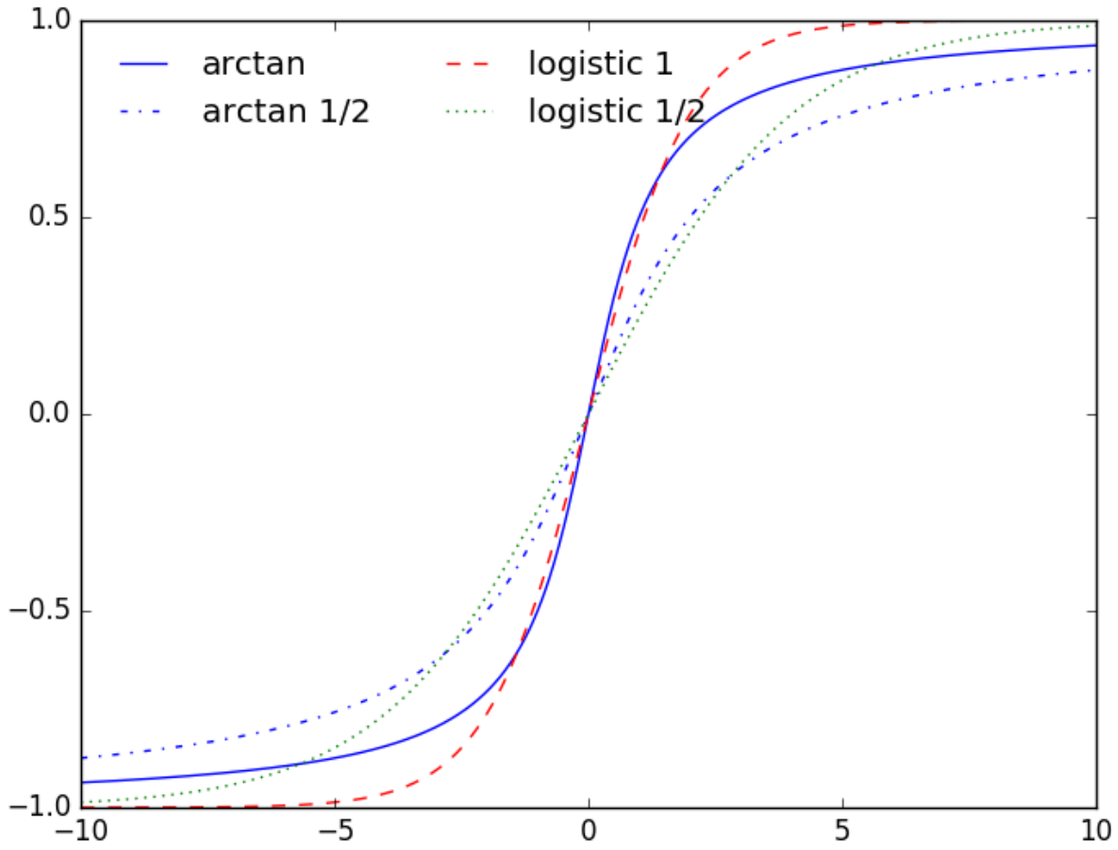
Plot Legends



Plot Legends

- Turn off the frame

```
ax.legend(loc='upper left',
          ncol = 2,
          frameon=False)
```



Plot Legends

- If we do not provide a label in a plot, then it will be ignored

Plot Legends

- Can use labels to provide additional information
  - Open california_cities.csv
  - Create a scatter plot based on latitude and longitude
  - Set the color to the population (decadic logarithm)
  - Set the size to the area of the city
  - Deploy a colorbar with a label (in LaTeX)

Plot Legends

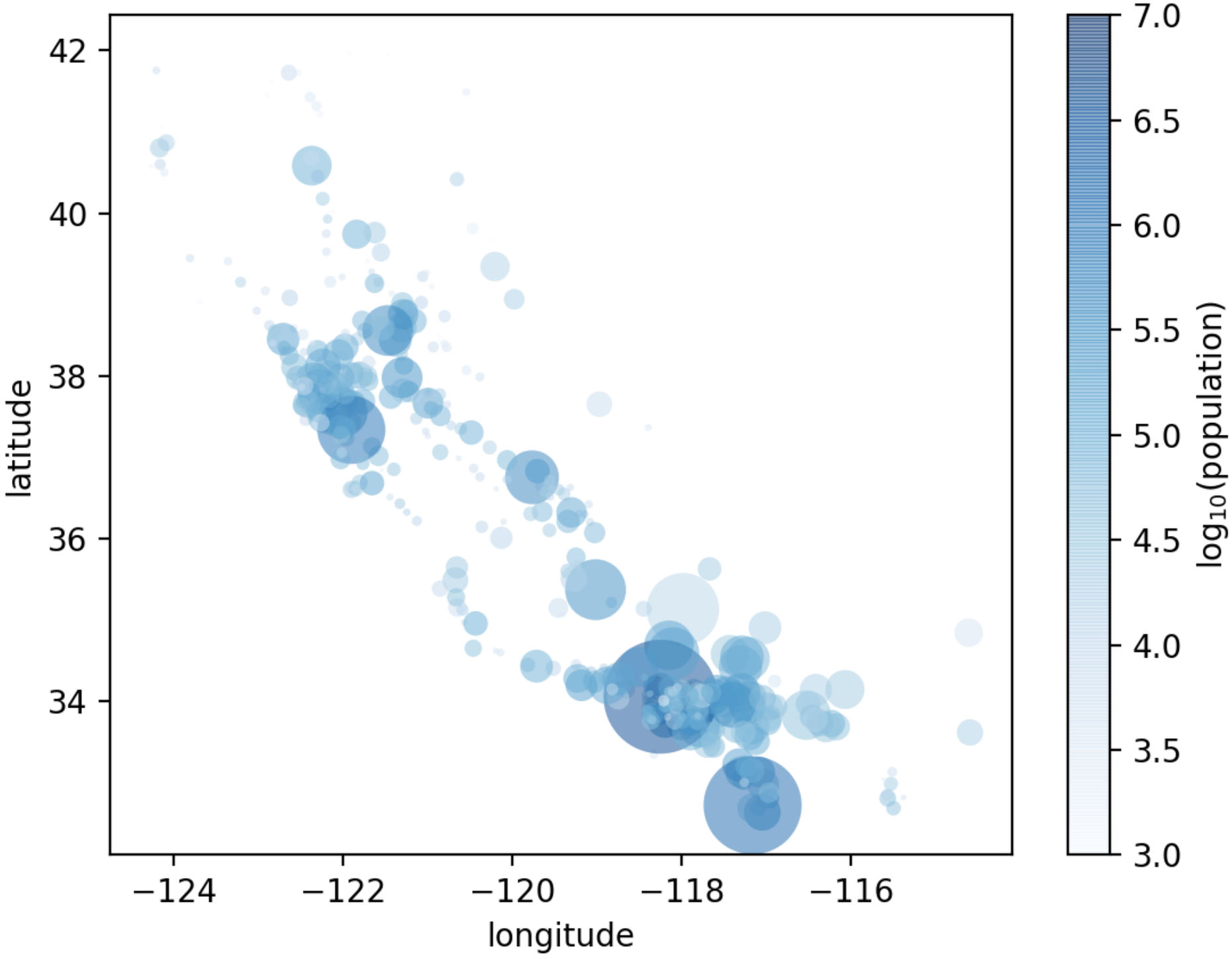
```
cities = pd.read_csv('california_cities.csv')

lat, lon = cities['latd'], cities['longd']
population, area = cities['population_total'],
cities['area_total_km2']
```

```
plt.scatter(lon, lat, label=None,
            c = np.log10(population), cmap='Blues',
            s = area, linewidth = 0, alpha = 0.5)

plt.axis(aspect='equal')
plt.xlabel('longitude')
plt.ylabel('latitude')
plt.colorbar(label='$\log_{10}$(population)')
plt.clim(3,7)  #color scaling
```

Plot Legends



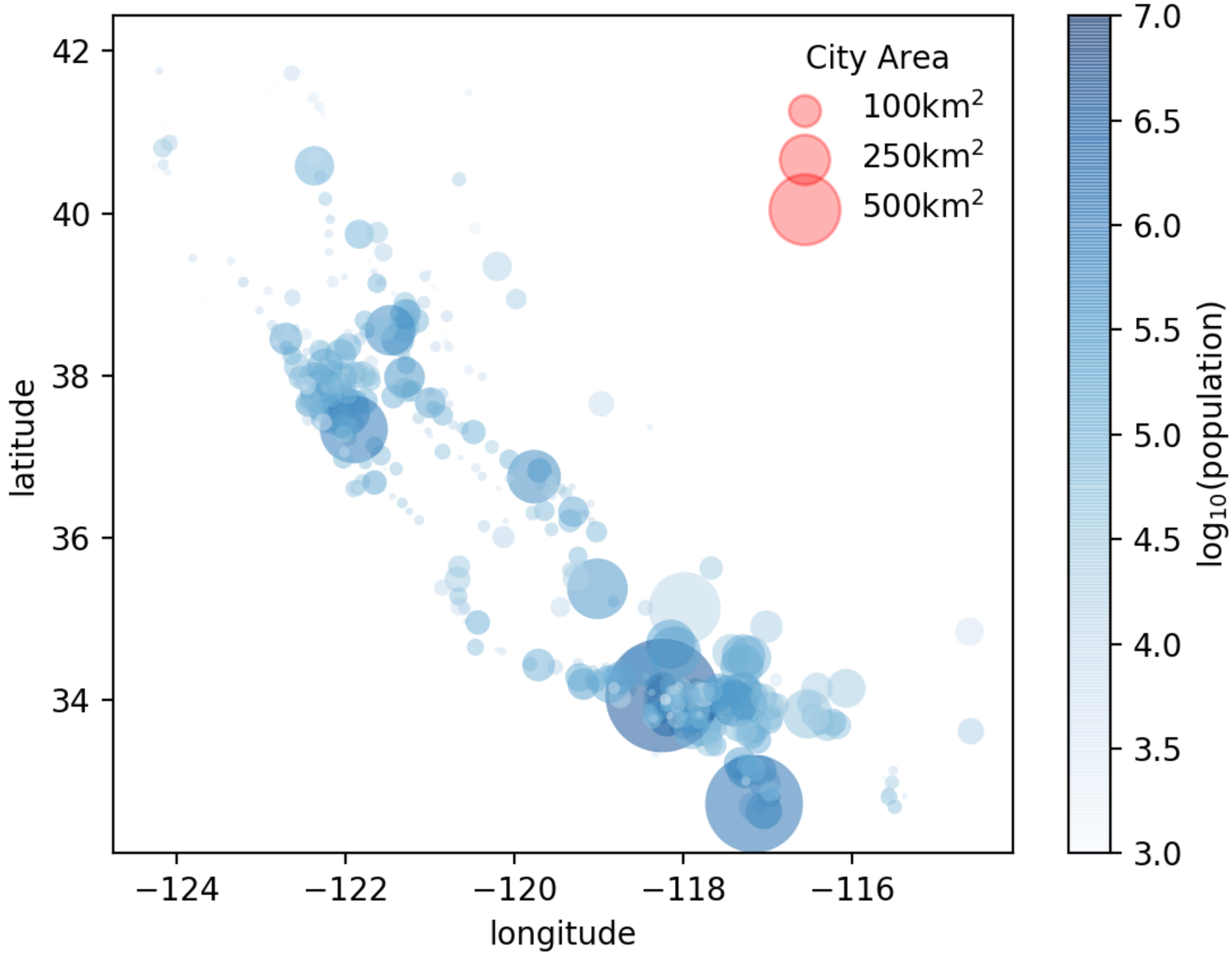
Plot Legends

- This lacks an explanation of the areas of the cities
  - In order to create a legend, we need to plot comparison cities with 'label'
  - But these cities do not have to exist

```
for area in [100, 250, 500]:
    plt.scatter([],[], c='r',
                alpha = 0.3, s=area,
                label = str(area)+'km$^2$')
plt.legend(frameon = False, title='City Area')
```

Plot Legends

- The label isn't so great

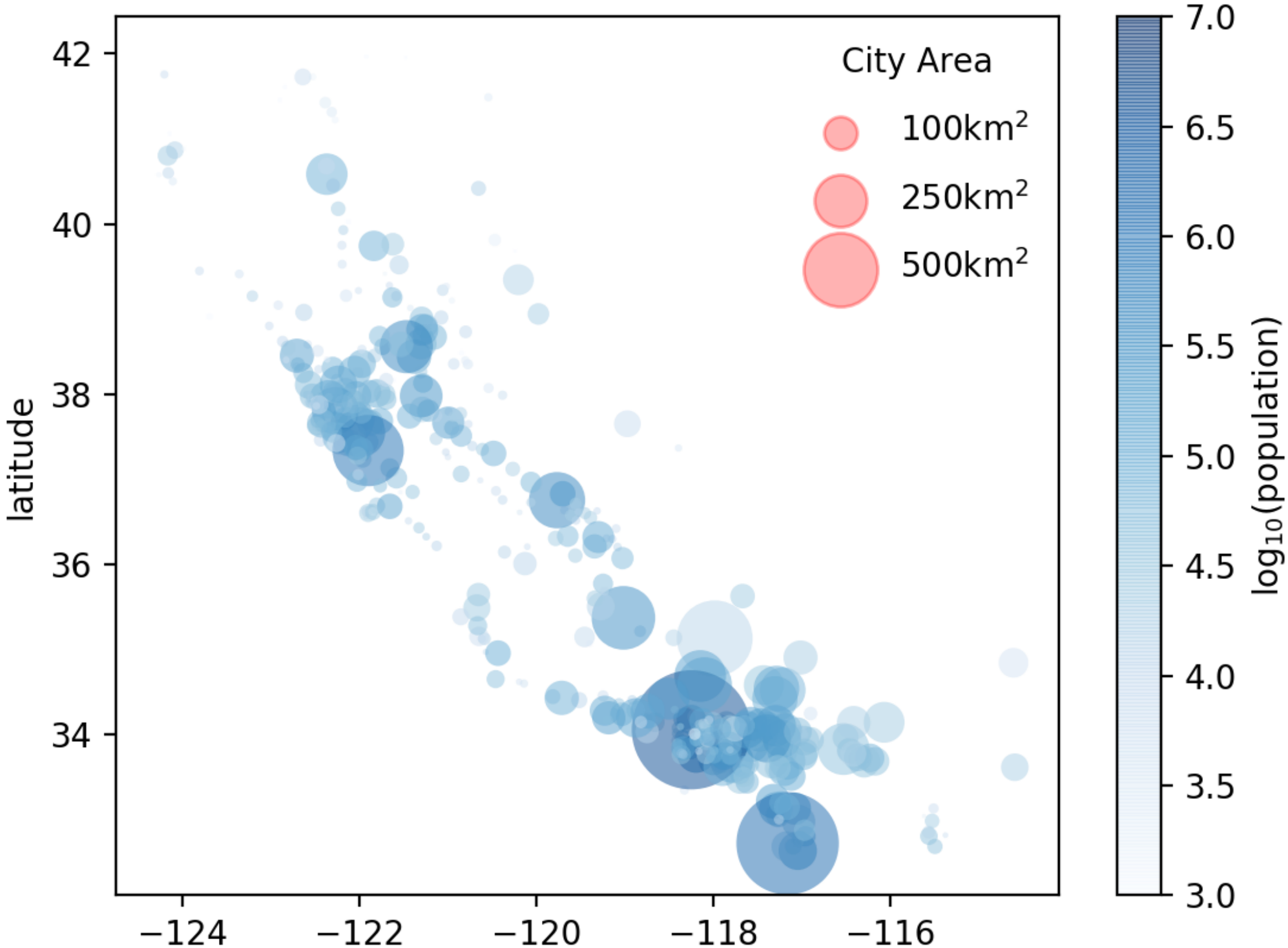


Plot Legends

- Use labelspacing to avoid overlap

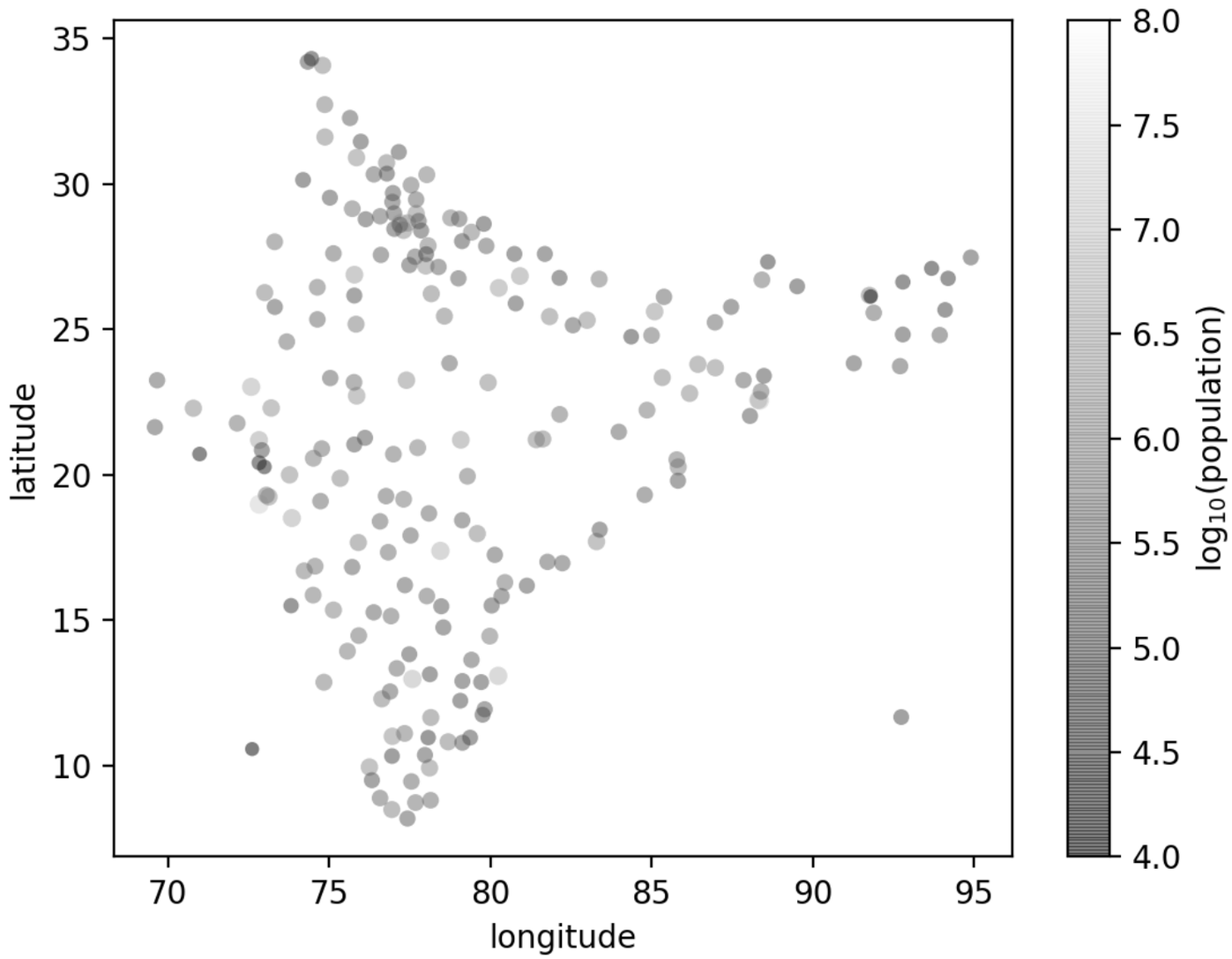
```
plt.legend(frameon = False,
                  title='City Area',
                  labelspacing=1)
```

Plot Legends



Homework

- Use the in.csv map to create a map of India's cities



Homework

- Restrict the map to cities of more than 2 million population
- Use a better color-scheme

Text and Annotation

- We download a database on births per day in the US

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

```
births = pd.read_csv('births.csv')
```

```
>>> births.head()
   year  month  day gender  births
0  1969      1  1.0      F    4046
1  1969      1  1.0      M    4440
2  1969      1  2.0      F    4454
3  1969      1  2.0      M    4548
4  1969      1  3.0      F    4548
```

Aggregation & Grouping

- Pandas has a number of aggregation functions, such as means, sum, count, …
- Use grouby to group by 'gender' and calculate the total number in the file

```
>>> births.groupby('gender')[['births']].sum()
           births
gender
F        74035823
M        77738555
```

Aggregation & Grouping

- Or subdivide by year

```
>>> births.groupby(['gender', 'year'])[['births']].sum()
                births
gender year
F      1969   1753634
       1970   1819164
       1971   1736774
       1972   1592347
       1973   1533102
...                ...
M      2004   2108197
       2005   2122727
       2006   2188268
       2007   2212118
       2008   2177227

[80 rows x 1 columns]
```

Aggregation & Grouping

- pivot_table is much simpler to use:

```
>>> births.pivot_table('births',
                       index='year',
                       columns = 'gender',
                       aggfunc='sum')
gender          F         M
year
1969      1753634   1846572
1970      1819164   1918636
1971      1736774   1826774
1972      1592347   1673888
1973      1533102   1613023
1974      1543005   1627626
1975      1535546   1618010
1976      1547613   1628863
1977      1623363   1708796
```

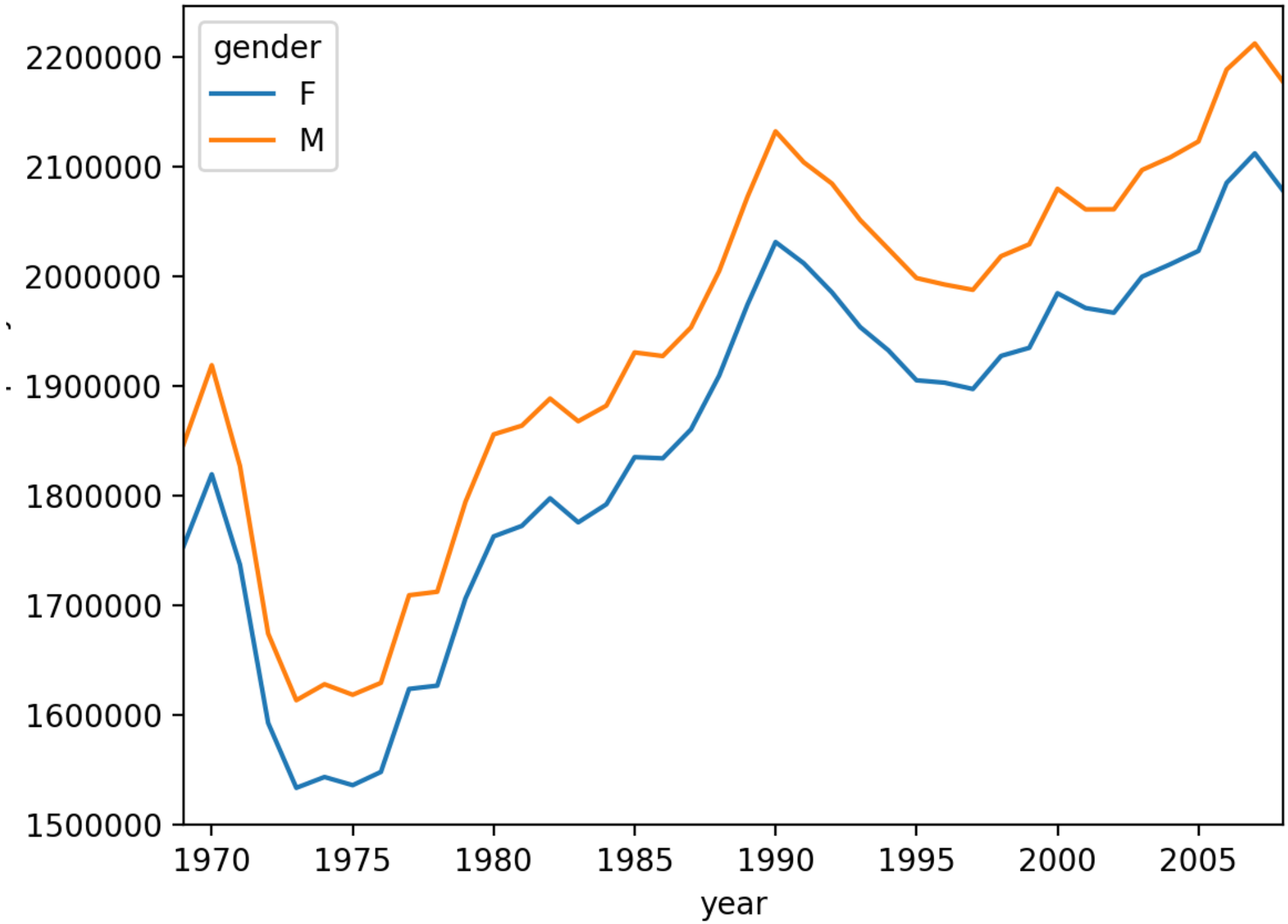
Aggregation & Grouping

- Here is the data per year

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

births = pd.read_csv('births.csv')
births.pivot_table('births',
                   index='year',
                   columns='gender',
                   aggfunc='sum').plot()
plt.ylabel('total births per year');
plt.show()
```

Aggregation & Grouping



Aggregation & Grouping

- The data set has some problems:
  - Missing values or typos (June 31)
  - One possibility is to programmatically find outliers
  - First, we estimate the standard deviation of the values

```
births = pd.read_csv('births.csv')
quartiles = np.percentile(births['births'],
                          [25, 50, 75])
mu = quartiles[1]
sigma = 0.74*(quartiles[2]-quartiles[0])
print(mu, sigma)
```

Aggregation & Grouping

- Now we want to filter out all values that are more than 5σ away from μ
- This can be done in pandas using the query function

```
births = births.query('(births > @mu - 5*@sigma) &
(births < @mu + 5*@sigma)')
```

- Alternatively, write a Python script to clean the csv file

Aggregation & Grouping

- To get the mean number of births per day,
    - need to set the day column to integers
        - there are some 'null' values
        - this works because we already got rid of bad data

```
births['day'] = births['day'].astype(int)
```

Aggregation & Grouping

- Pandas has its own datetime converter
  - trick:
    - day, month, year are integers
    - calculate an integer in form YYYYMMDD
    - it is then interpreted as a string

```
births.index = pd.to_datetime(
    10000*births.year+100*births.month+births.day,
    format='%Y%m%d'
)
```

Aggregation & Grouping

- Result is success:
-

```
>>> births.head()
                year  month  day gender  births
1969-01-01  1969      1      1      F    4046
1969-01-01  1969      1      1      M    4440
1969-01-02  1969      1      2      F    4454
1969-01-02  1969      1      2      M    4548
1969-01-03  1969      1      3      F    4548
```

Aggregation & Grouping

- Create an additional column 'decade'
  - Divide year as integer by 10, then multiply by 10
    - 1974 -> 197 -> 1970
      - Then group by decades and calculate mean of births

```
>>> births.pivot_table('births',
      index='decade', columns = 'gender',
      aggfunc='mean').head()
gender             F             M
decade
1960    4802.290411  5056.827397
1970    4452.450164  4687.509584
1980    4968.524027  5222.411800
```

Aggregation & Grouping

- Can also pivot according to day of the week

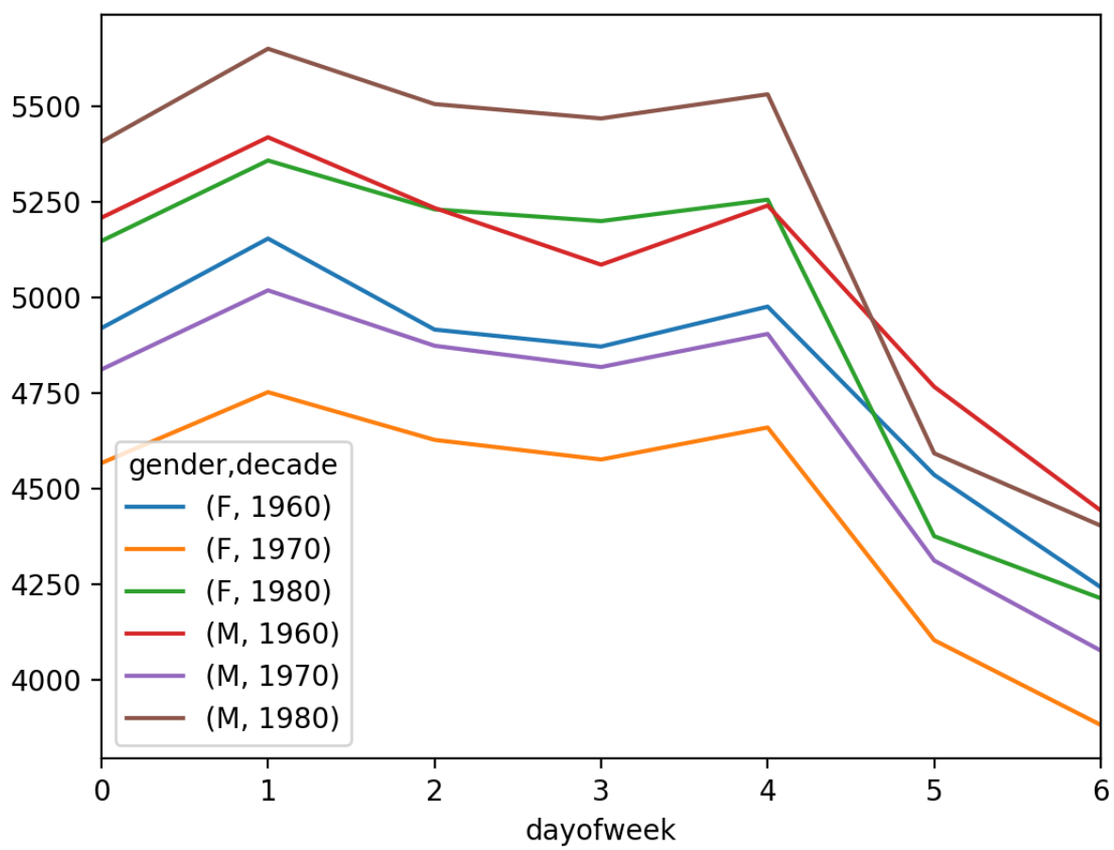
```
births['dayofweek'] = births.index.dayofweek
```

```
>>> births.pivot_table('births', index='dayofweek', columns =
['gender', 'decade'], aggfunc='mean')
gender                F                    ...              M
decade             1960          1970    ...           1970          1980
dayofweek                                 ...
0           4919.500000  4566.764368  ...  4811.431034  5406.437100
1           5153.692308  4752.195777  ...  5018.309021  5649.948936
2           4915.660377  4627.401152  ...  4873.351248  5505.321277
3           4871.230769  4576.057471  ...  4817.789272  5467.665957
4           4975.769231  4659.810345  ...  4904.381226  5530.895745
5           4535.884615  4103.703065  ...  4311.865900  4592.244681
6           4242.115385  3882.128352  ...  4076.429119  4403.017058
```

Text and Annotation

- We can separate genders:

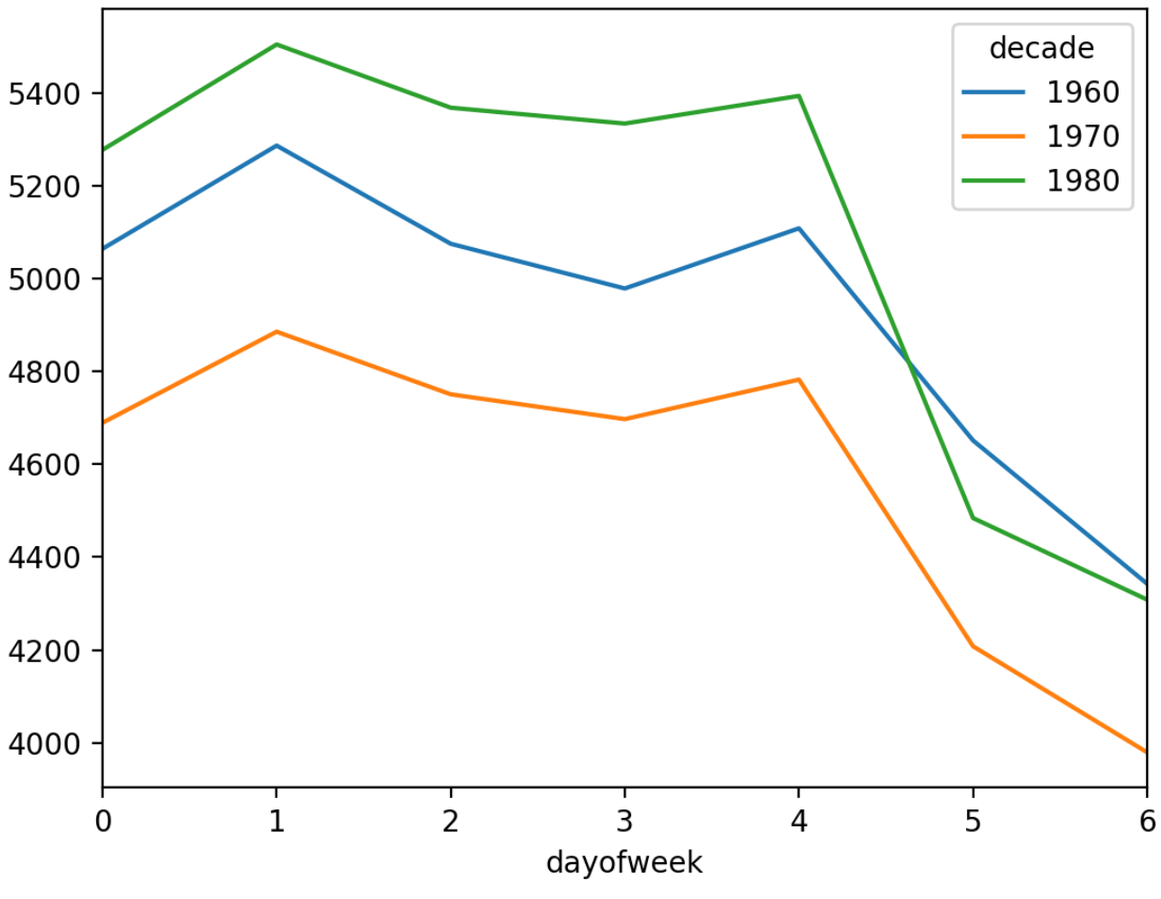
```
births.pivot_table('births',
                   index='dayofweek',
                   columns = ['gender', 'decade'],
                   aggfunc='mean').plot()
```



Text and Annotation

- Or not

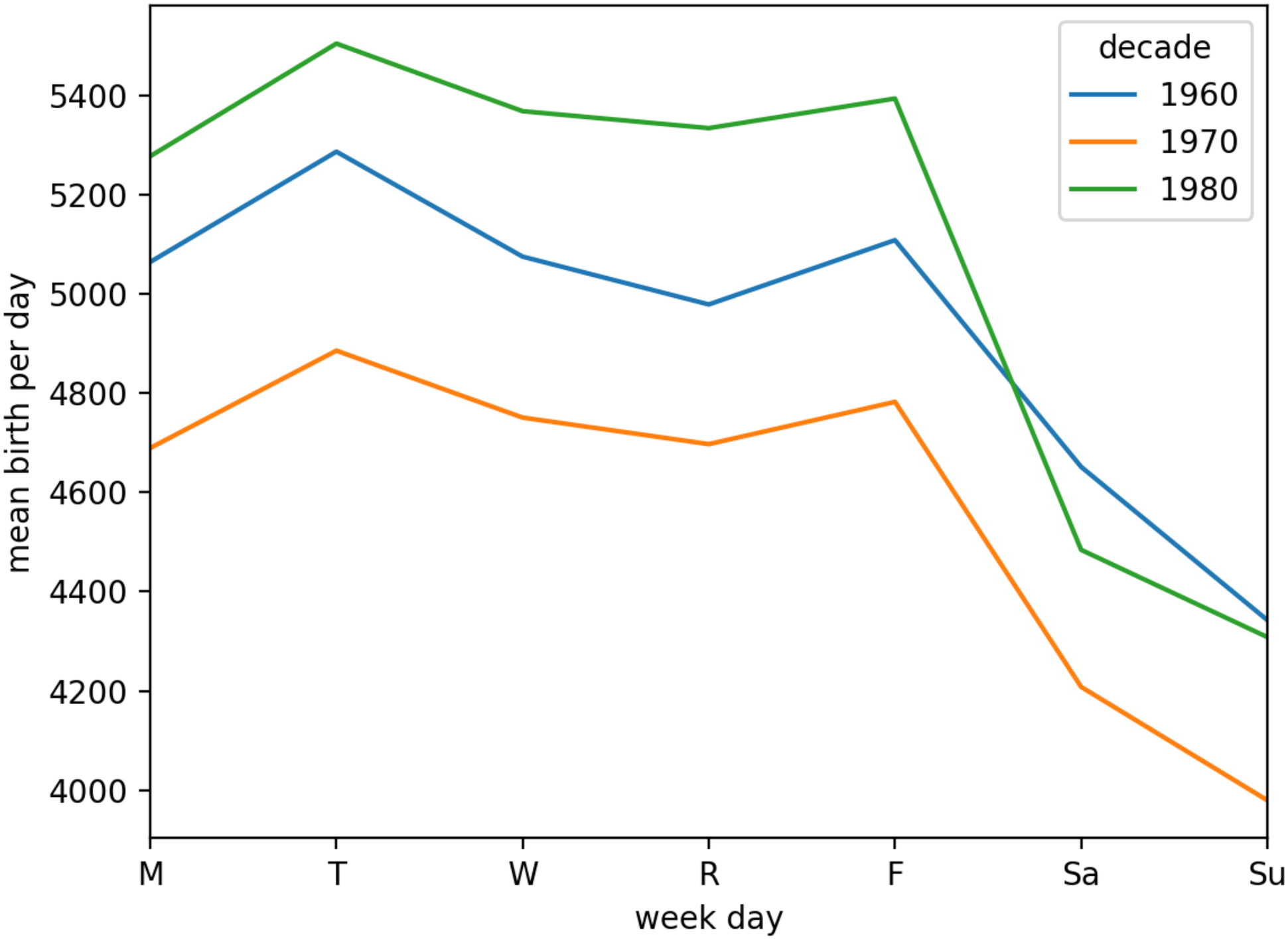
```
births.pivot_table('births',
                   index='dayofweek',
                   columns = ['gender', 'decade'],
                   aggfunc='mean').plot()
```



Text and Annotation

- This is not so great: The day of the week is a number
  - We can actually access labels, but this is in the inside of matplotlib itself
  - Use gca to set ticks
- ```
  plt.gca().set_xticklabels(['M', 'T',
                             'W', 'R', 'F', 'Sa', 'Su'])
  plt.ylabel('mean birth per day')
  plt.xlabel('week day')
  ```
  ```

Text and Annotation



Text and Annotation

- Apparently, birth dates are controlled by the day of the week

Text and Annotation

- What about the effects of holidays
  - Create a table with a new index corresponding to the days in 2000 (which was a leap year)

```
births_by_date = births.pivot_table('births',
                    [births.index.month, births.index.day])

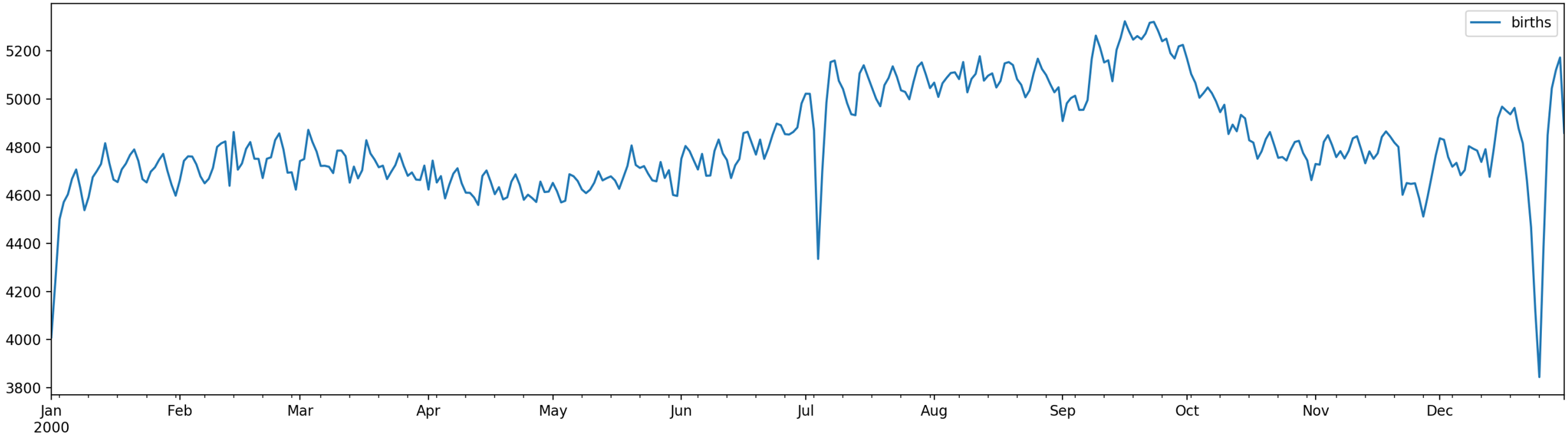
births_by_date.index = [pd.datetime(2000, month, day)
                for (month, day) in births_by_date.index]
```

Text and Annotation

- Then plot them

```
fig, ax = plt.subplots(figsize=(20,5))
births_by_date.plot(ax=ax)
```

Text and Annotation



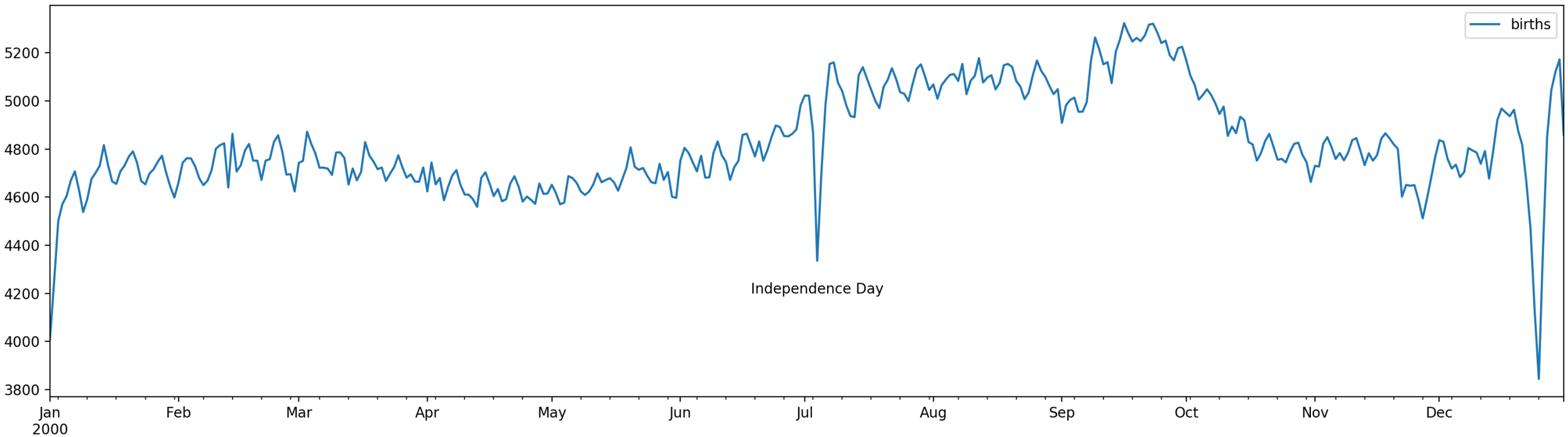
Text and Annotation

- Can observe:
  - Couples procreate more in January
  - Three big dips: July 4 and December 24/25, and January 1
  - Effect of Thanksgiving and Memorial day are not as visible

Text and Annotation

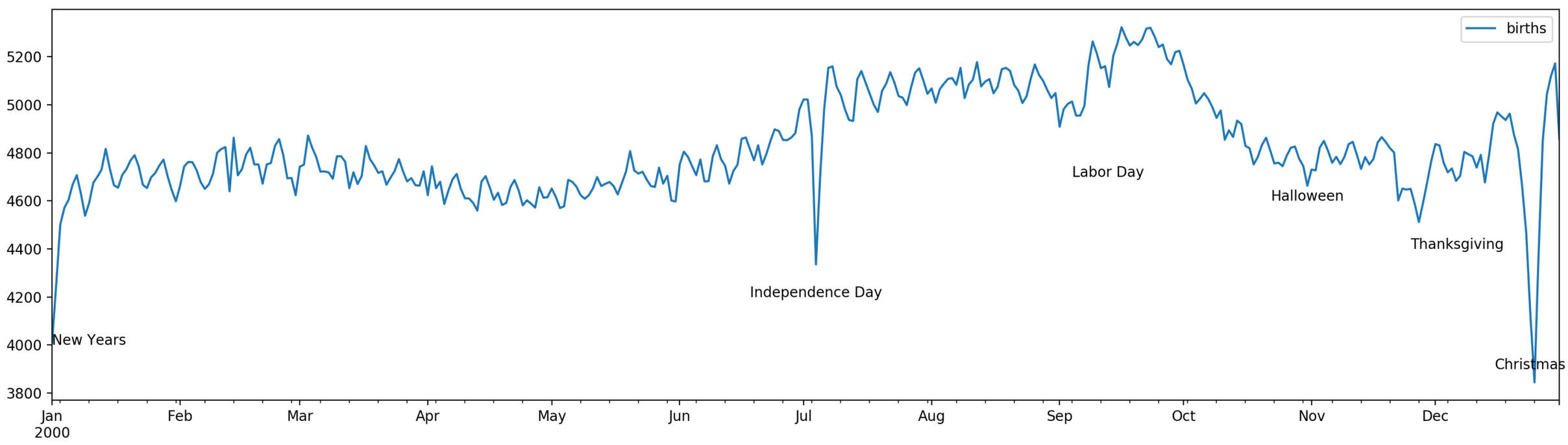
- Can use plt.text or ax.text to place a text at a certain coordinate
-
```
my_style = dict(size=10, color='black')
ax.text('2000-7-4', 4200,
'Independence Day',
ha='center',
**my_style)
```

Text and Annotation



Text and Annotation

```
my_style = dict(size=10, color='black')
ax.text('2000-7-4', 4200, 'Independence Day', ha='center', **my_style)
ax.text('2000-10-31', 4600, 'Halloween', ha='center', **my_style)
ax.text('2000-12-24', 3900, 'Christmas', ha='center', **my_style)
ax.text('2000-1-1', 4000, 'New Years', ha='left', **my_style)
ax.text('2000-9-4', 4700, 'Labor Day', ha='left', **my_style)
ax.text('2000-11-25', 4400, 'Thanksgiving', ha='left', **my_style)
```



Text and Annotation

- Can also use ax.annotate

```
ax.annotate('Look here', xy = ('2000-7-4',4350),
xytext=('2000-8-1',
        4000),
arrowprops = dict(facecolor='black', shrink = 0.05))
```