Data Visualization

Christopher Simpkins

chris.simpkins@gatech.edu

Data Visualization

Lecture material taken or adapted from Phillip K. Janert, Data Analysis with Open Source Tools, O'Reilly, 2010

- Data visualization is an activity in the exploratory data analysis process in which we try to figure out what story the data has to tell
- We'll be introduced to
  – Numpy
  – Matplotlib
  – Single-variable plots
  – Two-variable plots
  – Multi-variable plots

One Variable: Shape and Distribution

- What's the spread of values?
- Are the values symmetric?
- Is the distribution even, or are there common values?
- Are there outliers? How many?
- We can answer questions like these with
  - Dot and jitter plots
  - Histograms
  - Cumulative distribution plots
- We'll learn how to construct these plots with Matplotlib and get a feel for what they tell us

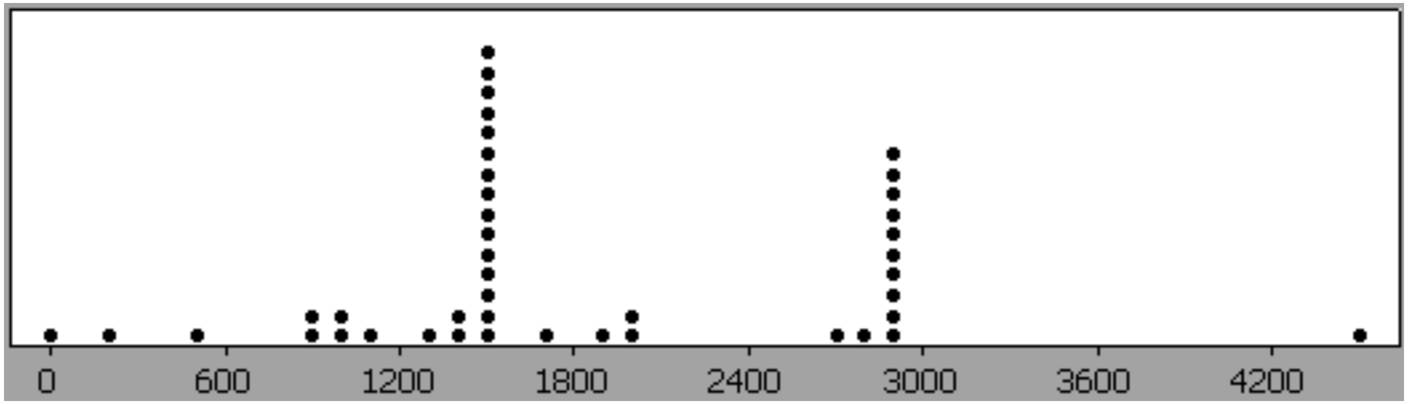
Dot and Jitter Plots

Washington	2864
Adams	1460
Jefferson	2921
Madison	2921
Monroe	2921
Adams	1460
Jackson	2921
VanBuren	1460
Harrison	31
Tyler	1427
Polk	1460
Taylor	491
Filmore	967
Pierce	1460
Buchanan	1460
Lincoln	1503
Johnson	1418
Grant	2921
Hayes	1460
Garfield	199
Arthur	1260
Cleveland	1460
Harrison	1460
Cleveland	1460
McKinley	1655
Roosevelt	2727
Taft	1460
Wilson	2921
Harding	881
Coolidge	2039
Hoover	1460
Roosevelt	4452
Truman	2810
Eisenhower	2922
Kennedy	1036
Johnson	1886
Nixon	2027
Ford	895
Carter	1461
Reagan	2922
Bush	1461
Clinton	2922
Bush	1110

- Good first step in understanding a single-variable data set is to create a dot plot
- In this data, there is only one variable: days in office (for US presidents)

From Robert W. Hayden, A Dataset that is 44% Outliers, Journal of Statistics Education, Volume 13, Number 1 (2005), www.amstat.org/publications/jse/v13n1/datasets.hayden.html

Dot and Jitter of Presidential Terms



- Every number in the data set is represented by a dot
- Dots are stacked, or “jittered” so they don’t overlap
- The y-axis has no label because we’re just getting a feel for the shape of the data set (although if we labeled the y-axis with numbers we’d essentially have a histogram with a bin size of 1 - more later)
- How can we produce this plot in matplotlib?

Dot and Jitter in Matplotlib

- Unlike some statistical packages, like R, matplotlib doesn't support dot-and-jitter plots out of the box
- We can use scatter plots to do dot-and-jitter plots
- Rather than plotting the dots in a uniform stack, we plot them randomly
  - The x values are the numbers of days in office
  - The y values are random heights above the x-axis so that we can see each point and get an idea of mass, that is, where the points are clustering - these are the frequently occurring x values
- General matplotlib procedure:
  - load data from file
  - generate y values from data
  - plot with scatter plot

Loading data from a file: `numpy.loadtxt()`

- We want to ignore the first column (president's names), so we pass usecols=(1,) so only the second column (0-based indexing) is used; usecols expects a sequence, so we add a comma after the 1
- We want integers, so we pass `dtype=int`
- We'll use ipython with the `--pylab` option
  - ipython is an enhanced interactive Python shell
  - The `--pylab` option loads numpy and matplotlib into the current namespace, making ipython a matlab-like environment

```
$ ipython -pylab
In [1]: dio = loadtxt('presidents.dat', usecols=(1,), dtype=int)

In [2]: dio
Out[2]:
array([2864, 1460, 2921, 2921, 2921, 1460, 2921, 1460,   31, 1427, 1460,
        491,  967, 1460, 1460, 1503, 1418, 2921, 1460,  199, 1260, 1460,
       1460, 1460, 1655, 2727, 1460, 2921,  881, 2039, 1460, 4452, 2810,
       2922, 1036, 1886, 2027,  895, 1461, 2922, 1461, 2922, 1110])
```

Generating the y-values from the x-values

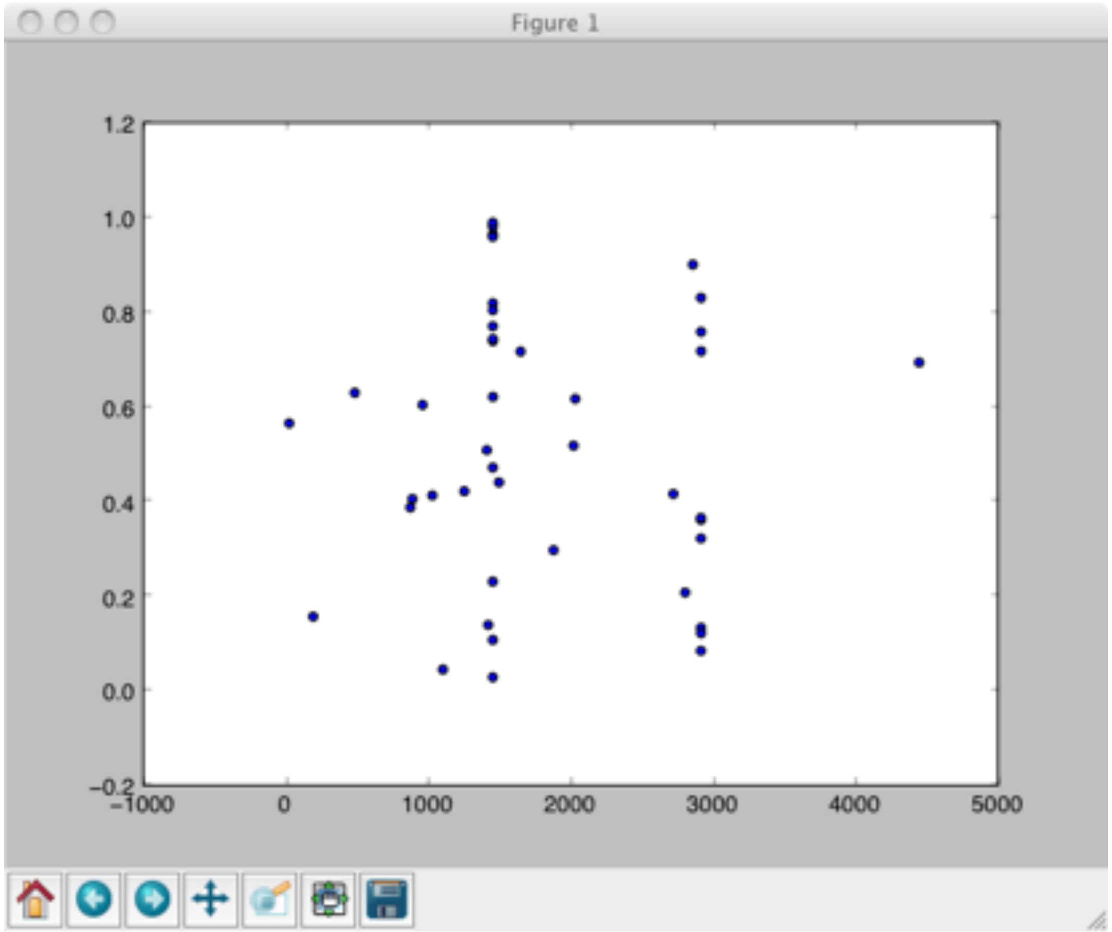
- Generate random y-values for each x-value to “jitter” them (recall that jittering means moving them slightly so they don’t cover each other up)
- Use Python’s `random.random()` function (not `numpy.random.randn()`) so we get y-values in the range [0,1).

```
In [14]: ys = [random.random() for x in range(0, len(dio))]
```

- Now we have x-values and y-values, so we’re ready to plot

```
In [14]: scatter(dio, ys)
Out[14]: <matplotlib.collections.PathCollection at 0x1169cbb50>
```

Our First Dot and Jitter Plot in Matplotlib



- We get a window like this
- Need to fix a few things:
  - Aspect ratio of plot distorts the data, which is much “wider”
  - Axis labels show small negative regions, but all data values are positive
  - Y axis needs no labels

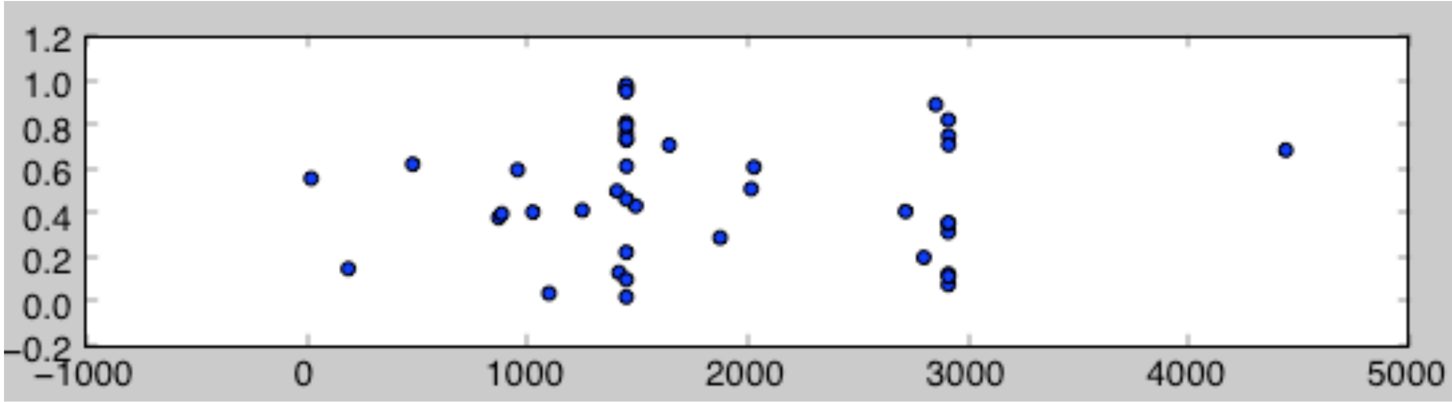
Setting the Aspect Ratio in Matplotlib

- To change the aspect ratio of the chart, we need a reference to its `Axes` object, which we get with the `axes()` method
- The method we need from the Axes object is `set_aspect`

```
In [29]: help(axes().set_aspect)

Help on method set_aspect in module matplotlib.axes:

set_aspect(self, aspect, adjustable=None, anchor=None) method of matplotlib.axes
.AxesSubplot instance
    *aspect*

      ========   ================================================
      value      description
      ========   ================================================
      'auto'     automatic; fill position rectangle with data
      'normal'   same as 'auto'; deprecated
      'equal'    same scaling from data to plot units for x and y
       num       a circle will be stretched such that the height
                 is num times the width. aspect=1 is the same as
                 aspect='equal'.
      ========   ================================================
```

Using `axes().set_aspect()`

- Matplotlib provides a stateful command-oriented wrapper around figures - as we execute functions, the current figure changes (it's meant to be like MATLAB)
- If we want to start over, we can call `clf()` to get a blank figure
- Let's try `set_aspect(1000)` (I'm saving you some effort here - it requires a few trials and errors to figure out that 1000 is what we want)



- This passes the TLAR test (That Looks About Right)
- Now let's fix the axes' ranges and add a label for the x-axis

Setting Axis Ranges and Labels

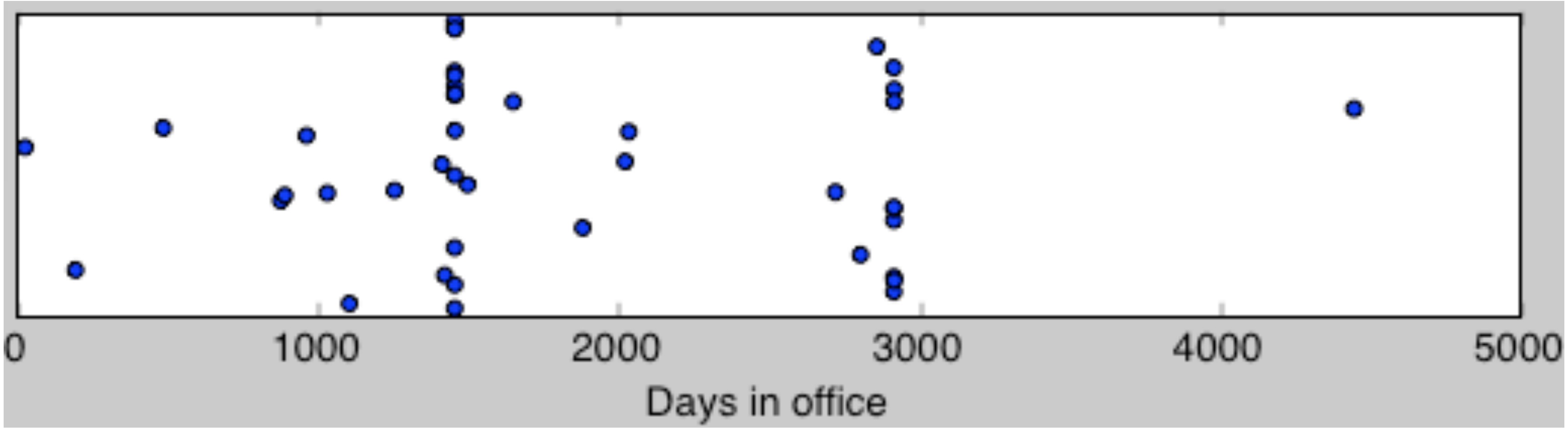
- To finish up our dot and jitter plot we'll set axis ranges, add an x-axis label, and suppress the y-axis since we don't care about the jitter values (we're only plotting dots)
  - `axis([xmin, xmax, ymin, ymax])` sets the axis ranges
  - `axes().yaxis.set_visible()` suppresses the y-axis
  - `xlabel()` sets the x-axis label

```
In [48]: axis([0,5000,0,1])
Out[48]: [0, 5000, 0, 1]

In [49]: axes().yaxis.set_visible(False)

In [50]: xlabel('Days in office')
Out[50]: <matplotlib.text.Text at 0x104b4e910>
```

The Complete pylab Dot Plot Session

```
$ ipython --pylab
In [1]: dio = loadtxt('presidents.dat', usecols=(1,), dtype=int)

In [2]: import random

In [3]: ys = [random.random() for x in range(0, len(dio))]

In [4]: scatter(dio, ys)
Out[4]: <matplotlib.collections.PathCollection at 0x105198250>

In [5]: axes().set_aspect(1000)

In [6]: axis([0, 5000, 0, 1])
Out[6]: [0, 5000, 0, 1]

In [7]: axes().yaxis.set_visible(False)
```

Our Finished Dot and Jitter Plot



- Now to the point: what does it tell us?
  - The data is bimodal, with modes at 1460 and 2921
    - » What do these modes mean? Hint: 1460 = 4 x 365
  - Almost half of the data are outliers
  - Can't just discard outliers as noise without good reason. In this data set, the outliers provide crucial information: many presidents serve partial terms.

Histograms

- Dot plots give us a big picture, but they aren’t good for showing quantitative features of the data and aren’t good for large data sets
- Histograms divide a univariate data set into bins and show how many elements are in each bin
- Good for showing the shape of a distribution
- Two main parameters: bins and alignment
- Alignment is more important for a smaller data set

Salary Data Set

199770.00
174208.37
170537.50
170060.00
162015.00
153497.52
...
49494.82
46684.35
46523.68
43604.15
40503.32
40161.62
37792.10
37513.80
37012.52
...
9378.74
8554.04
7943.66
7380.95
6666.67
6556.20
5000.00
1175.68

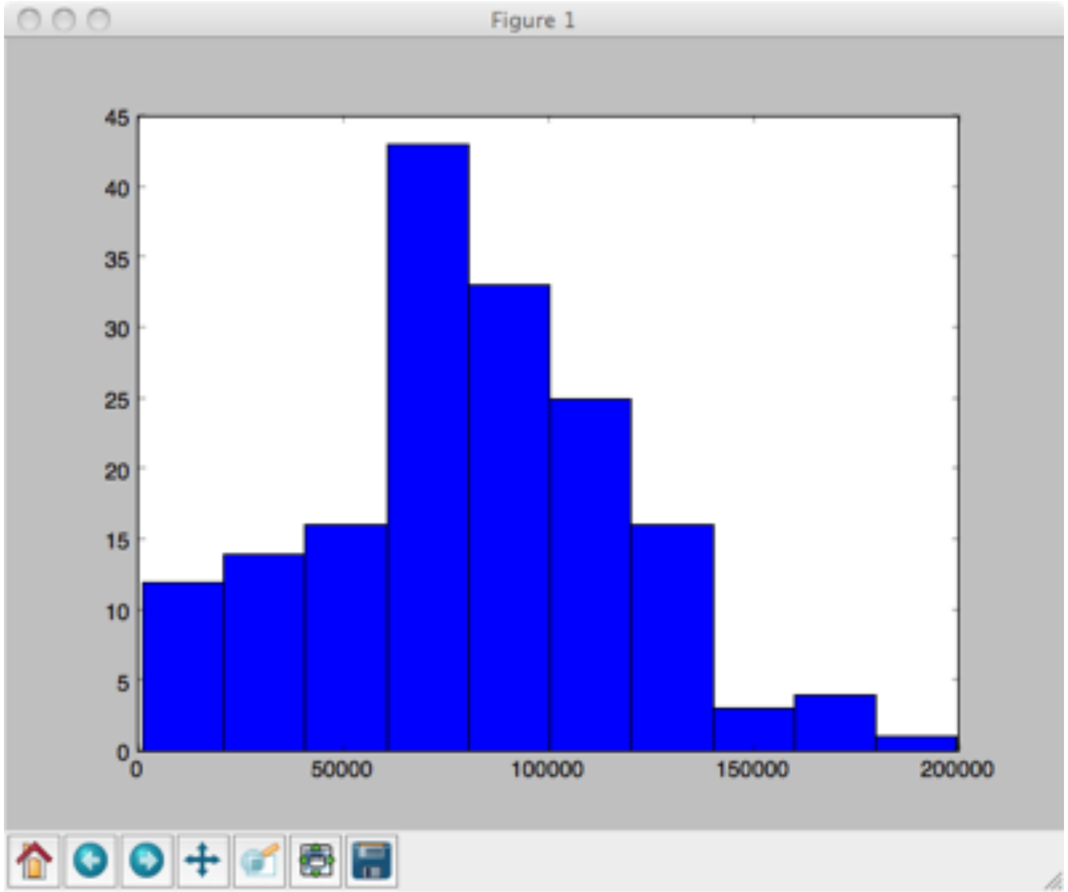
- We'll use a data set that contains a single column of data: the salaries of professionals in an organization
- Good example of anonymized data
- 167 data points

Simple Histograms in pylab

- This:

```
$ ipython --pylab

In [1]: lab1 = loadtxt('lab1-all.dat')

In [2]: hist(lab1)
```

- Gives us this:



Determining the Optimal Bin Width

- Trial and error process, but with a principled "first guess"
- Scott's Rule (assumes a Gaussian distribution, but assumption is OK since all we're doing is educating our first guess):

$$w = \frac{3.5\sigma}{\sqrt[3]{n}}$$

- Here's how we compute this in pylab:
  - Standard deviation is given by `std()` method on array
  - n is the size of the array, given by `len(lab1)`
  - We approximate cube root by raising to 1/3 power

```
In [15]: binwidth = (3.5 * lab1.std()) / (len(lab1) ** .333)
```

Using Bin Width in Matplotlib

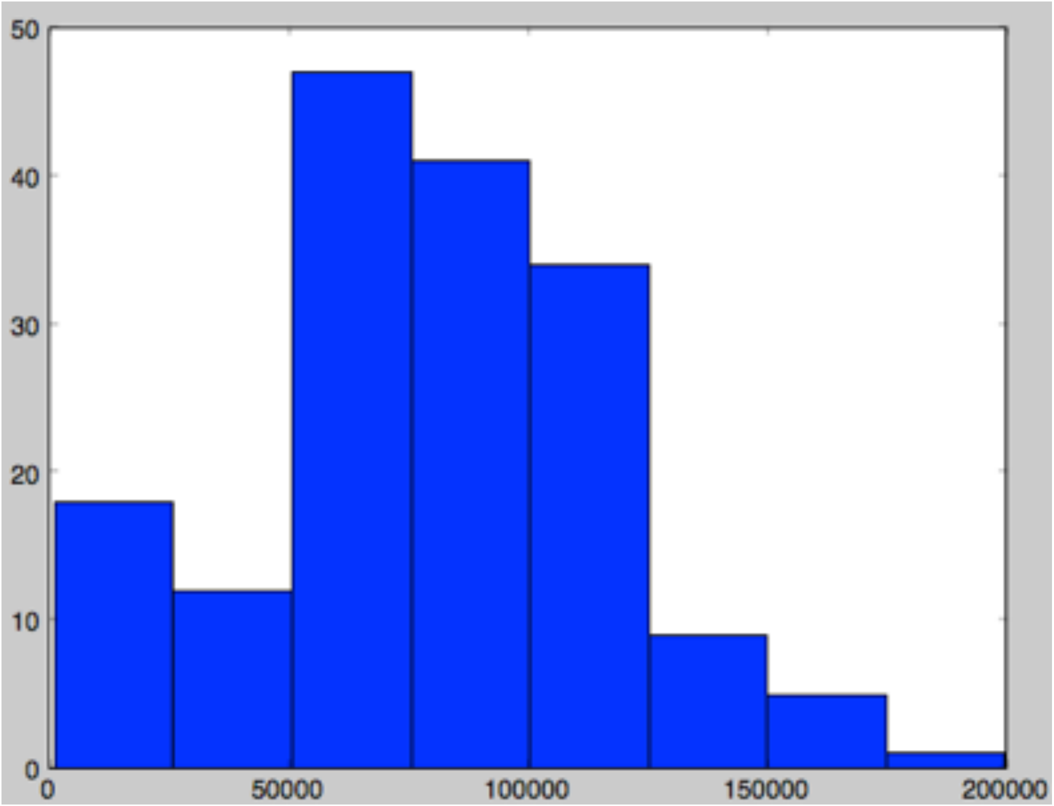
- Matplotlib's `hist()` function doesn't have a bin width parameter, it has a bins (number of bins) parameter
- So we divide the range of the data by the bin width (and convert that to an integer)

```
In [13]: bins = int( (lab1.max() - lab1.min()) / binwidth)

In [14]: bins
Out[14]: 8
```

- Let's try it:

```
In [15]: clf()

In [16]: hist(lab1, bins=8)
Out[16]:
(array([18, 12, 47, 41, 34,  9,  5,  1]),
 array([   1175.68,   25999.97,   50824.26,   75648.55,  100472.84,
        125297.13,  150121.42,  174945.71,  199770.  ]),
 <a list of 8 Patch objects>)
```

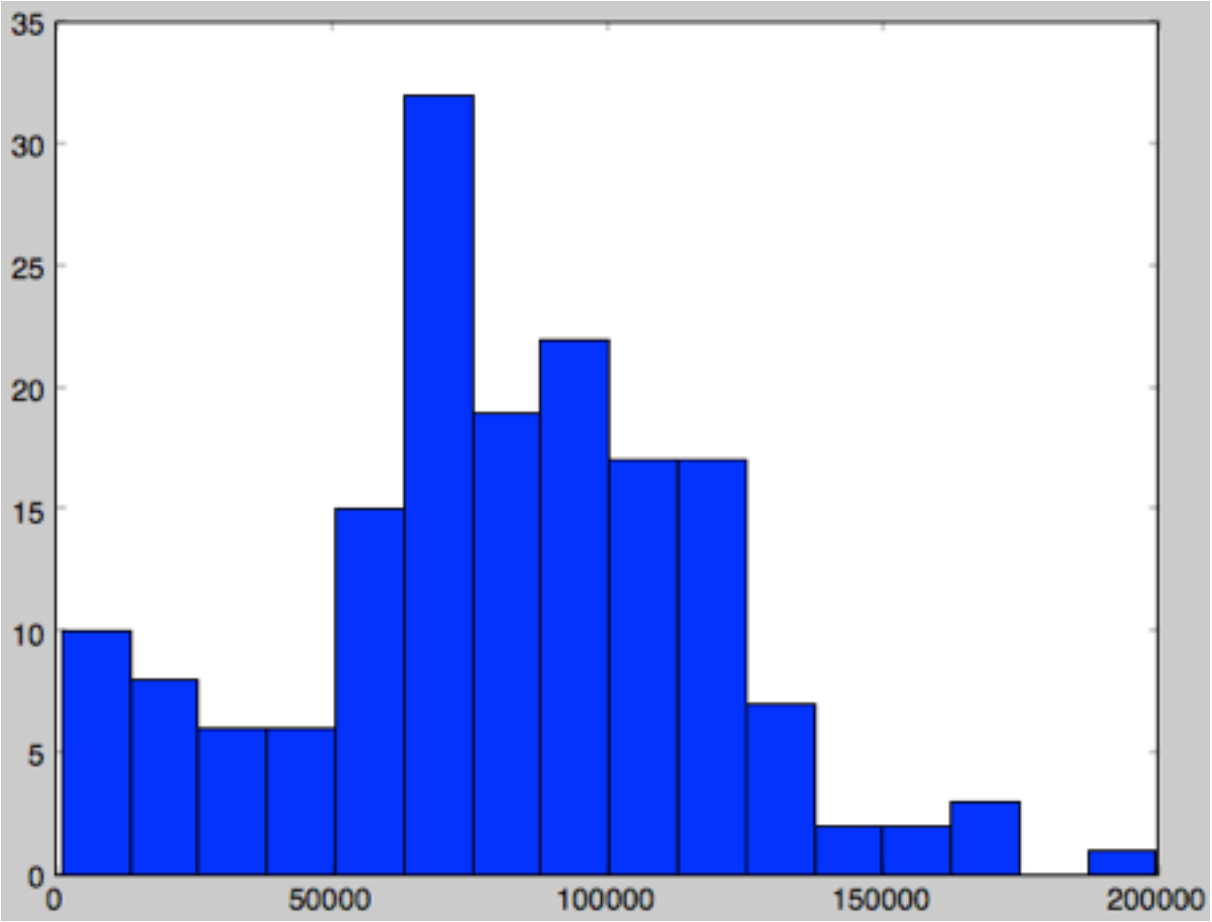
Our Salary Histogram with 8 Bins



- Scott's rule tends to give a bin-width that is too wide
- Let's try more bins:

```
In [19]: hist(lab1, bins=16)
```

Salary Histogram with 16 Bins

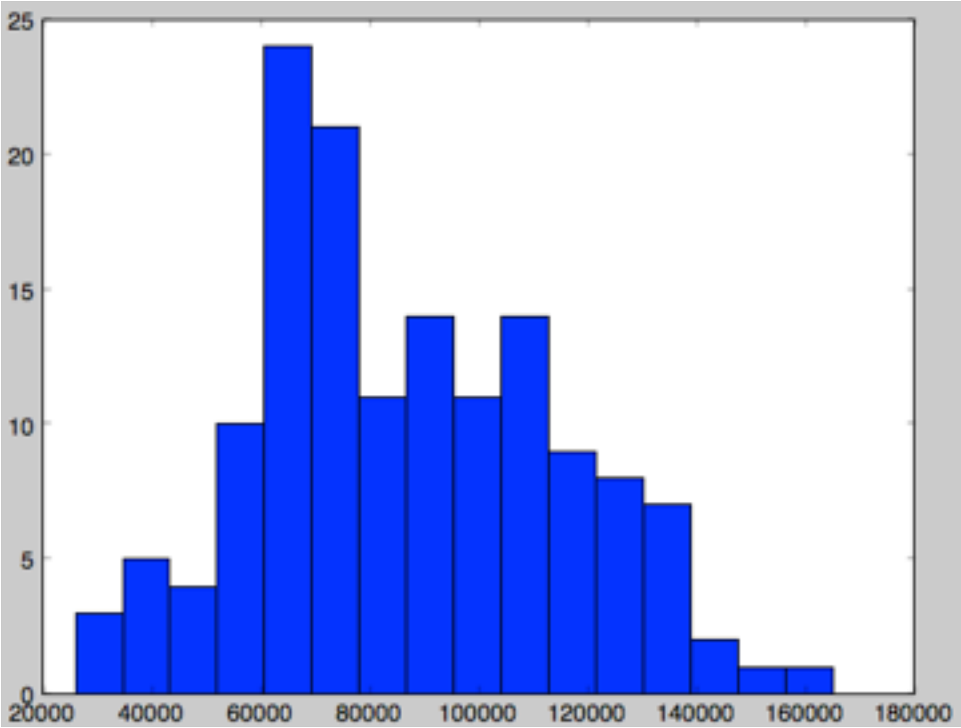


- Better. Now let's chop off the outliers by setting the range

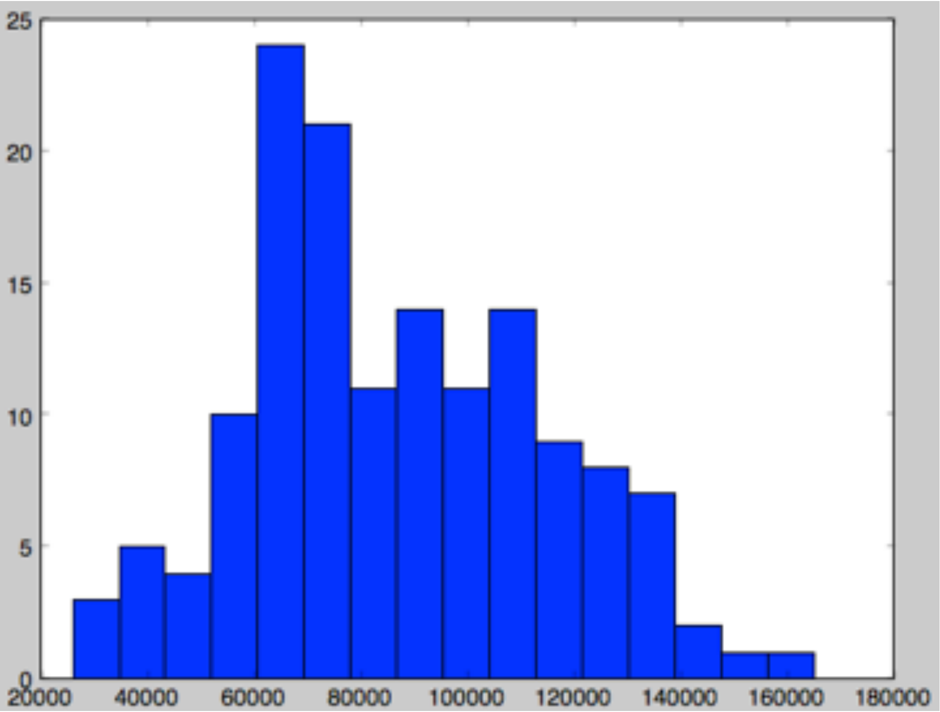
Setting the Range of a Histogram

- We can remove outliers from a histogram by restricting the range of data we plot with the `range=(min, max)` parameter

```
In [30]: clf()

In [31]: hist(lab1, bins=16, range=(26000, 165000))
```



What Does Our Histogram Tell Us?



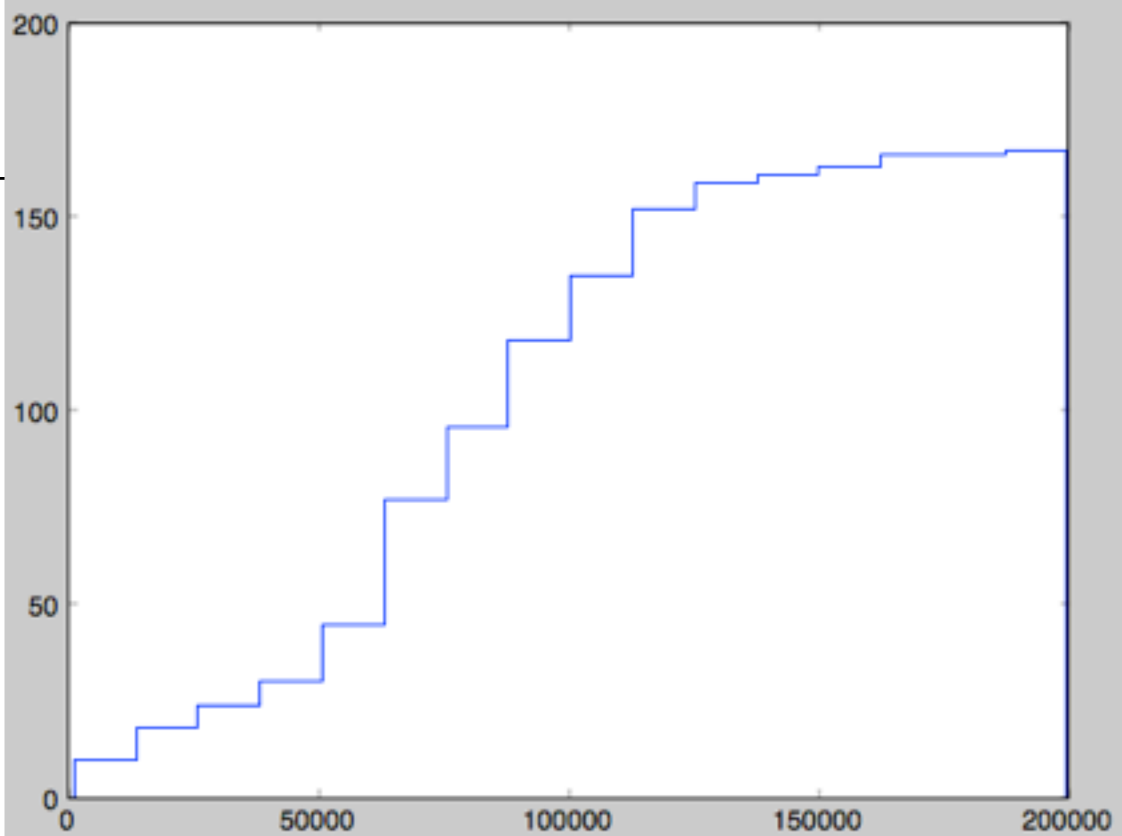
- Average salary looks like it's bout 70,000
- A few outliers with very low salaries, very few with high salaries
- Looks like the data follows a Gaussian distribution
  - In the next lecture, we'll see how to determine if the data is actually Gaussian-distributed

Cumulative Distribution Plots

- What if we want to know how many people have salaries above a certain number?
  - Hard to tell from histogram alone
- Cumulative distribution plot tells us this kind of information
- Cumulative distribution plots are done with the same hist() function in Matplotlib
  - pass the `cumulative=True` argument
  - pass `histtype='step'` to make it look like a CDF plot

CDF Plot of Salary Data

```
In [47]: clf()

In [48]: hist(lab1, bins=16, cumulative=True, histtype='step')
Out[48]: ...

In [49]: axis([0, 200000, 0, 200])
Out[49]: [0, 200000, 0, 200]
```



- Looks like more than half the professionals in this organization make over 70,000

Two Variables: Establishing Relationships

- We have two variables, perhaps one is a function of the other
- How can we tell?
- Scatter plots
  - Very simple to throw data into a scatter plot
  - Fun part comes in next lecture when we build a model of the relationship between the variables

Plotting Two Variables

1	3.385	44.500
2	0.480	15.500
3	1.350	8.100
4	465.000	423.000
5	36.330	119.500
6	27.660	115.000
7	14.830	98.200
8	1.040	5.500
9	4.190	58.000
10	0.425	6.400
11	0.101	4.000
12	0.920	5.700
13	1.000	6.600
14	0.005	0.140
...		

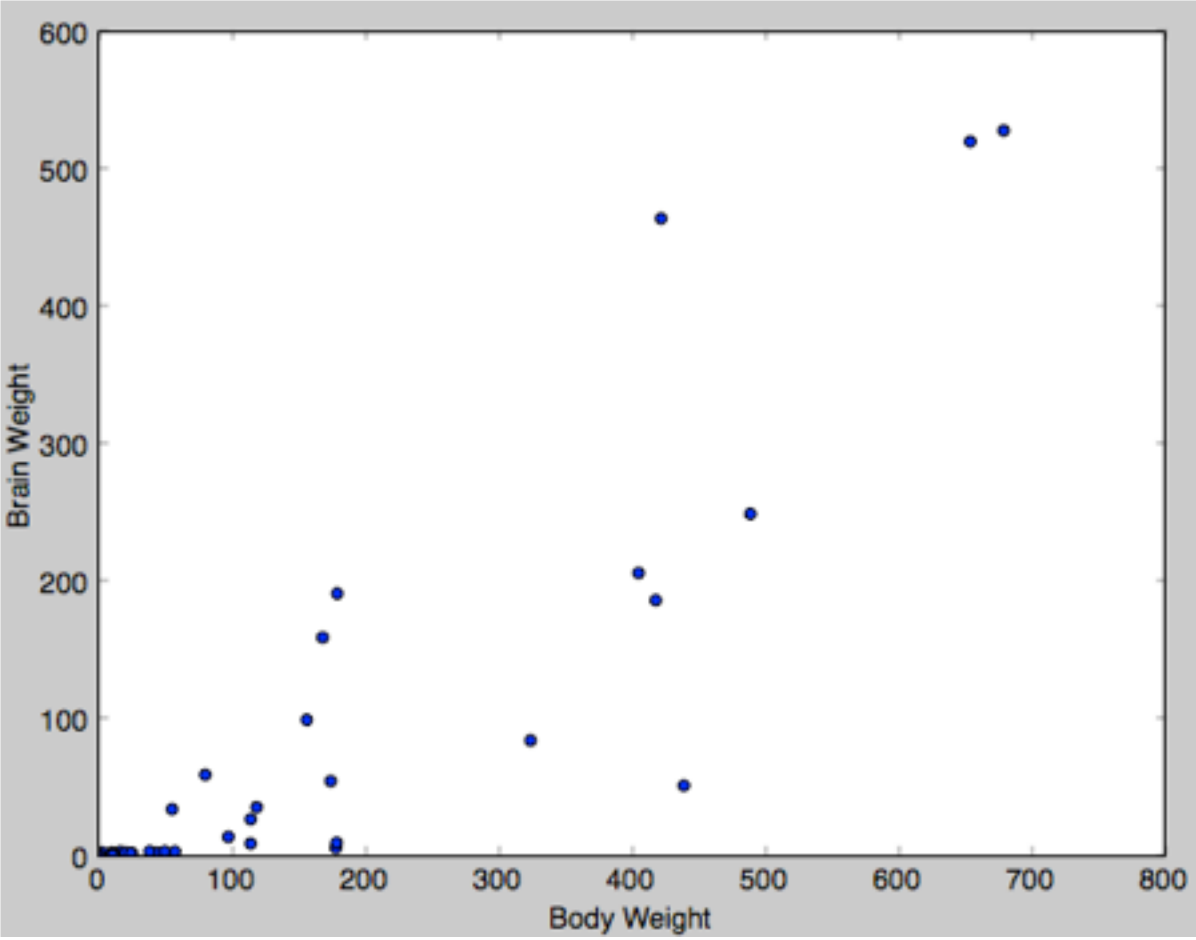
- Given a data set relating brain weight to body weight in various mammals[1]
- We can plot the brain weight versus body weight to see what relationship exists between them
- We'll take body weight as the independent variable, brain weight as a function of body weight
- Col 0 is index, col 1 is brain weight, col 2 is body weight
  - (I've removed outliers)

[1] http://orion.math.iastate.edu/burkardt/data/regression/regression.html

Scatter Plot in Matplotlib

```
In [70]: brain = loadtxt('brain-body.dat', usecols=(1,))
In [71]: body = loadtxt('brain-body.dat', usecols=(2,))
In [72]: scatter(body, brain)
Out[72]: <matplotlib.collections.PathCollection at 0x1189a1ad0>

In [73]: ylabel('Brain Weight')
Out[73]: <matplotlib.text.Text at 0x116edecd0>

In [74]: xlabel('Body Weight')
Out[74]: <matplotlib.text.Text at 0x116ee1810>

In [75]: axis([0, 800, 0, 600])
Out[75]: [0, 800, 0, 600]
```

Scatter Plot in Matplotlib



- Looks like a linear relationship. Next lecture we'll see how to confirm our intuition

Conclusion

- We've seen the vary basics of looking at single and two-variable data
- We've worked with numpy and matplotlib
- Many more topics to consider in data visualization:
  - Smoothing
  - Log plots
  - Multivariate plots
- In the next lecture we'll begin to analyze the data