Visualizing Data

Data Science

Data Visualization

- There are two primary uses for data visualization:

1. To explore data
2. To communicate data

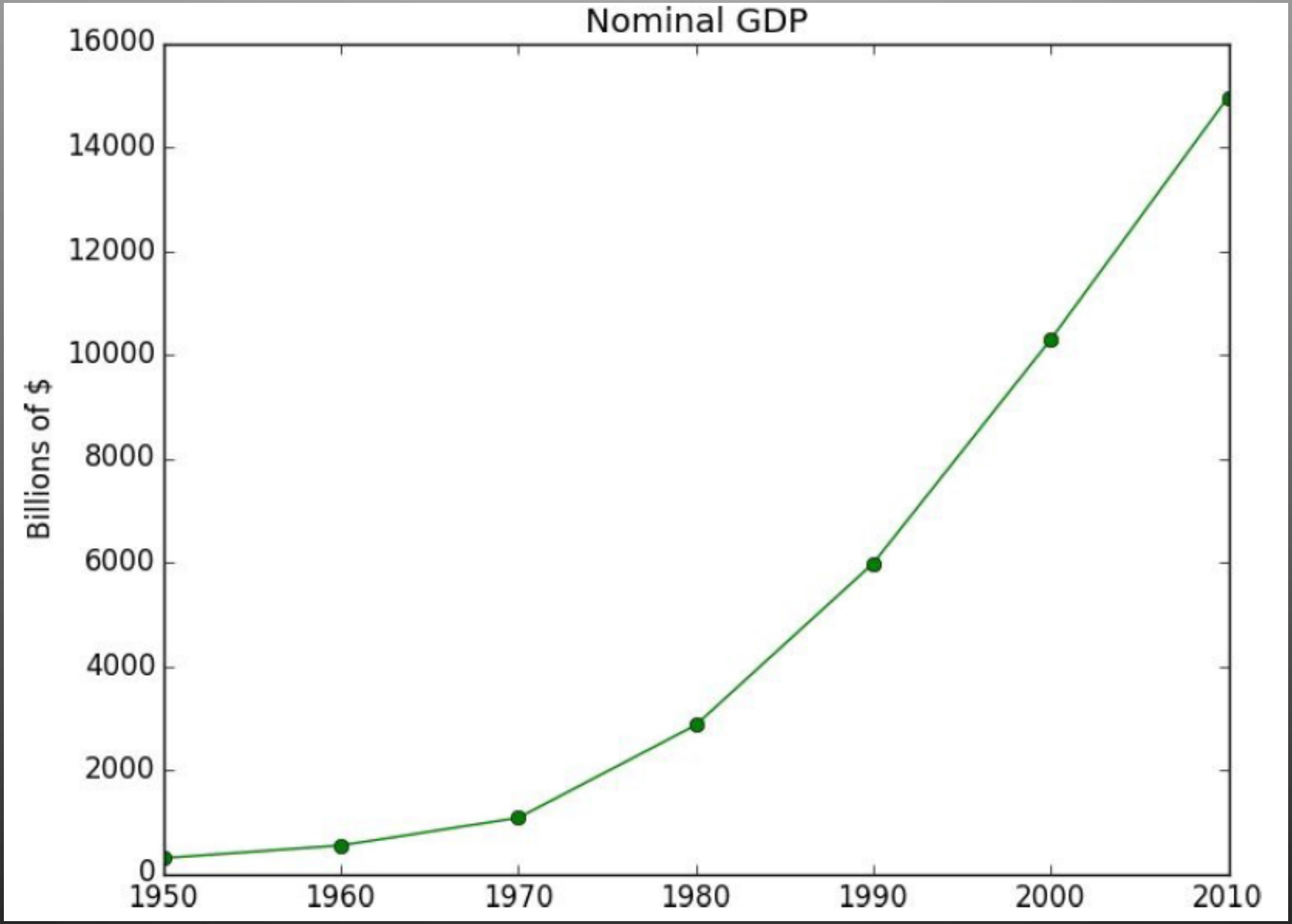
matplotlib

- We will be using the matplotlib.pyplot module.
- pyplot maintains an internal state in which you build up a visualization step by step.
- Once you are done you can save it with savefig() or display it with show().

matplotlib

```
from matplotlib import pyplot as plt
years = [1950, 1960, 1970, 1980, 1990, 2000, 2010]
gdp = [300.2, 543.3, 1075.9, 2862.5, 5979.6, 10289.7,
14958.3]
#create a line chart, years on x-axis, gdp on y
plt.plot(years, gdp, color='green', marker='o', linestyle =
'solid')
#add a title
plt.title("Nominal GDP")
#add a label to the y-axis
plt.ylabel("Billions of $")
plt.show()
```

matplotlib



Nominal GDP
Billions of $
16000
14000
12000
10000
8000
6000
4000
2000
0
1950
1960
1970
1980
1990
2000
2010

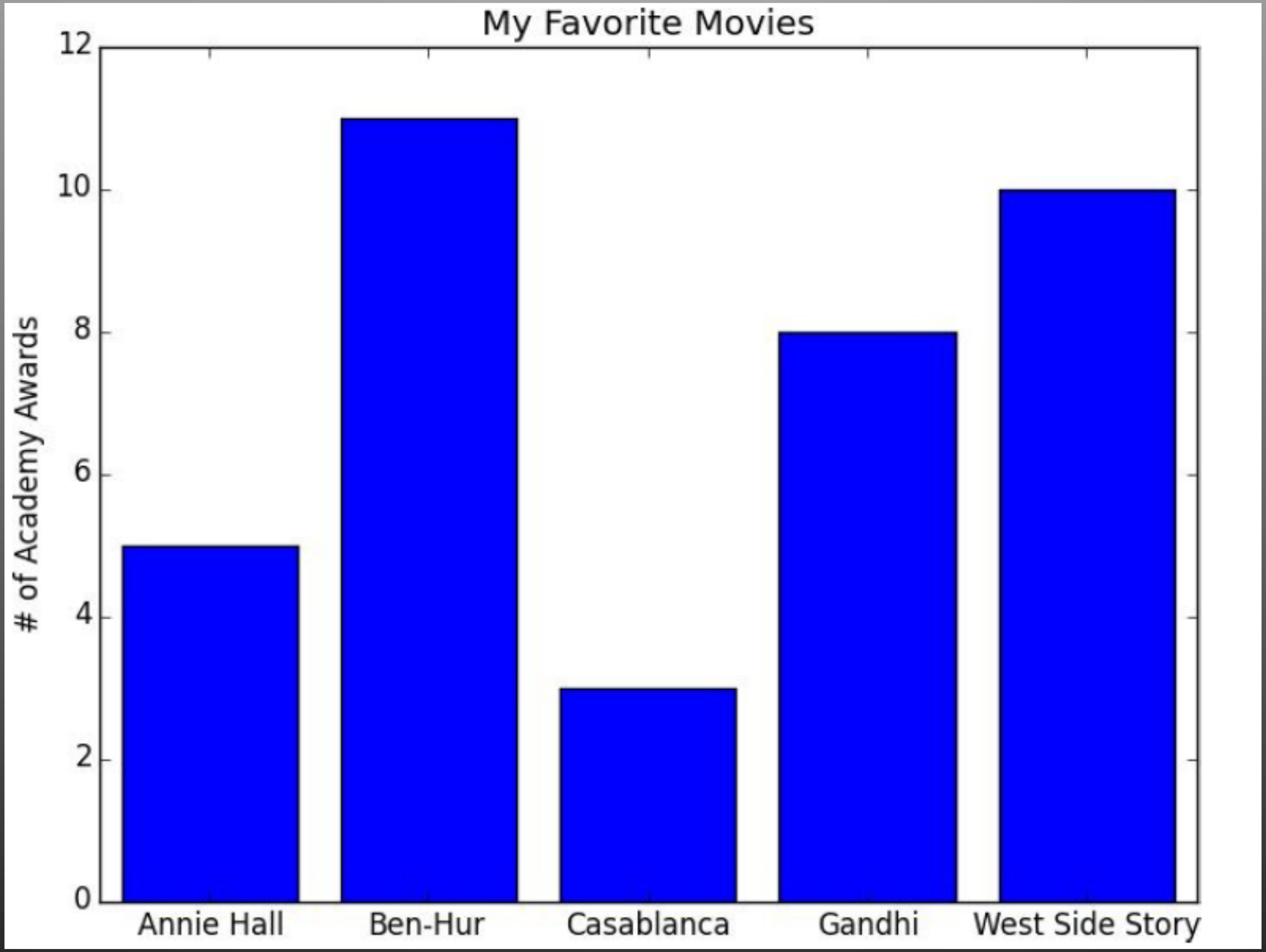
Bar Charts

- A bar chart is a good choice if you want to show how some quantity varies among some discrete set of items.

Bar Charts

```
movies = [“Annie Hall”, “Ben-Hur”, “Casablanca”, “Gandhi”, “West Side Story”]
num_oscars = [5,11,3,8,10]
#bars are by default width 0.8, so we will add 0.1 to the left coordinates so each bar is centred
xs = [i+0.1 for i, _ in enumerate(movies)]
#plot bars with left x-coordinate [xs], height [num_oscars]
plt.bar(xs, num_oscars)
plt.ylabel(“# of Academy Awards”)
plt.title(“Movies”)
#label x-axis with move names at bar centres
plt.xticks([i+0.5 for i, _ in enumerate(movies)], movies)
plt.show()
```

Barcharts



My Favorite Movies
# of Academy Awards
0
2
4
6
8
10
12
Annie Hall
Ben-Hur
Casablanca
Gandhi
West Side Story

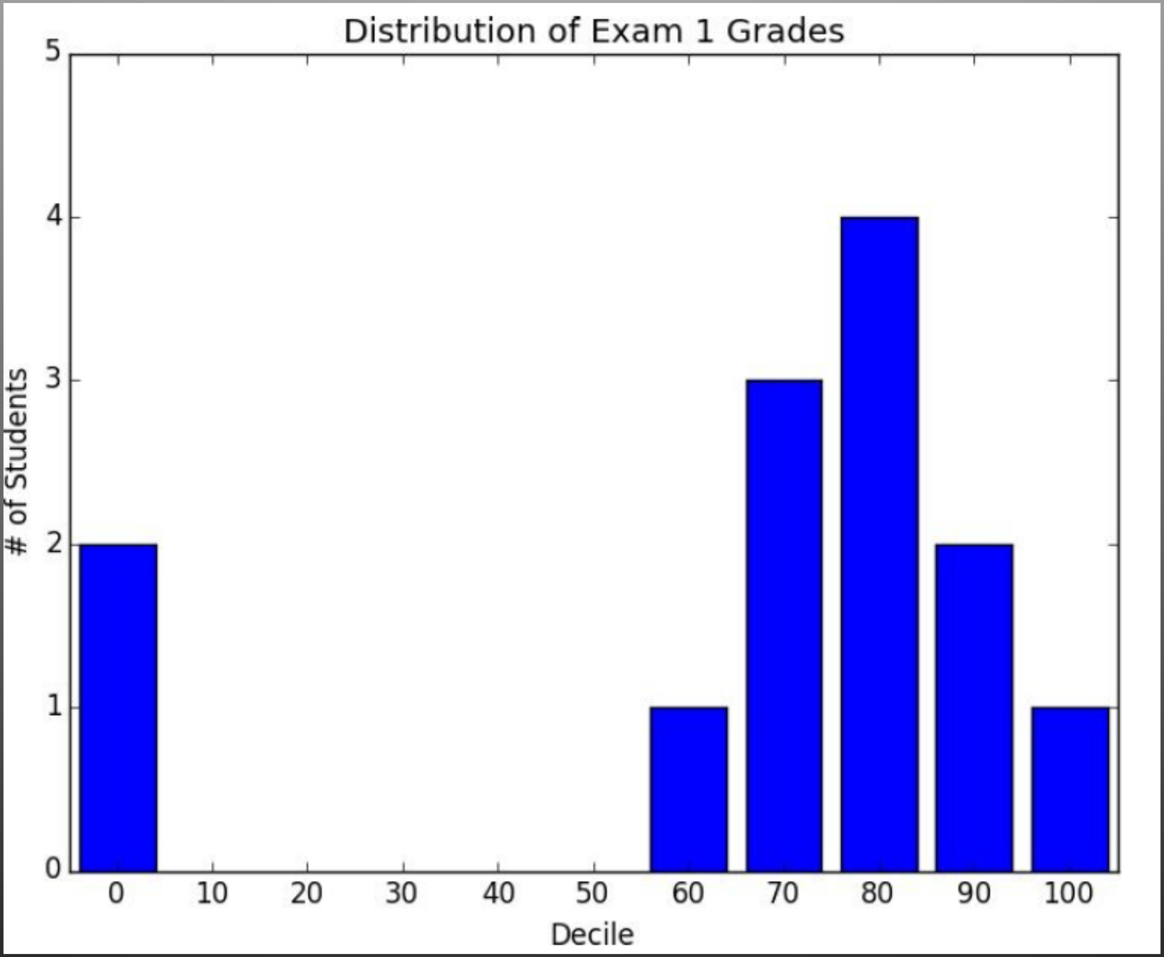
Bar Charts

- A bar chart can also be a good choice for plotting histograms of bucketed numeric values, in order to visually explore how the values are distributed.

Bar Charts

```
grades = [83,95,91,87,70,0,85,82,100,67,73,77,0]
decile = lambda grade : grade // 10 * 10
histogram = Counter(decile(grade) for grade in grades)
plt.bar([x-4 for x in histogram.keys()], histogram.values(),
8)
plt.axis([-5, 105, 0 ,5])
plt.xticks([10*i for i in range(11)])
plt.xlabel(“Decile”)
plt.ylabel(“# of Students”)
plt.title(“Distribution of Exam 1 Grades”)
plt.show()
```

Bar Charts


Distribution of Exam 1 Grades
# of Students
Decile
0
1
2
3
4
5
0
10
20
30
40
50
60
70
80
90
100

Bar Charts

- The third argument to plt.bar specifies the bar width.
- Here we chose a width of 8 and we shifted the bar left by 4 so that, for example, the 80 bar has its left and right sides at 76 and 84 and its centre at 80.
- The call to plt.axis indicates that the x-axis should range from -5 to 105 and the y-axis should range from 0 to 5.
- plt.xticks puts x-axis labels at 0, 10, 20, 30, .., 100.

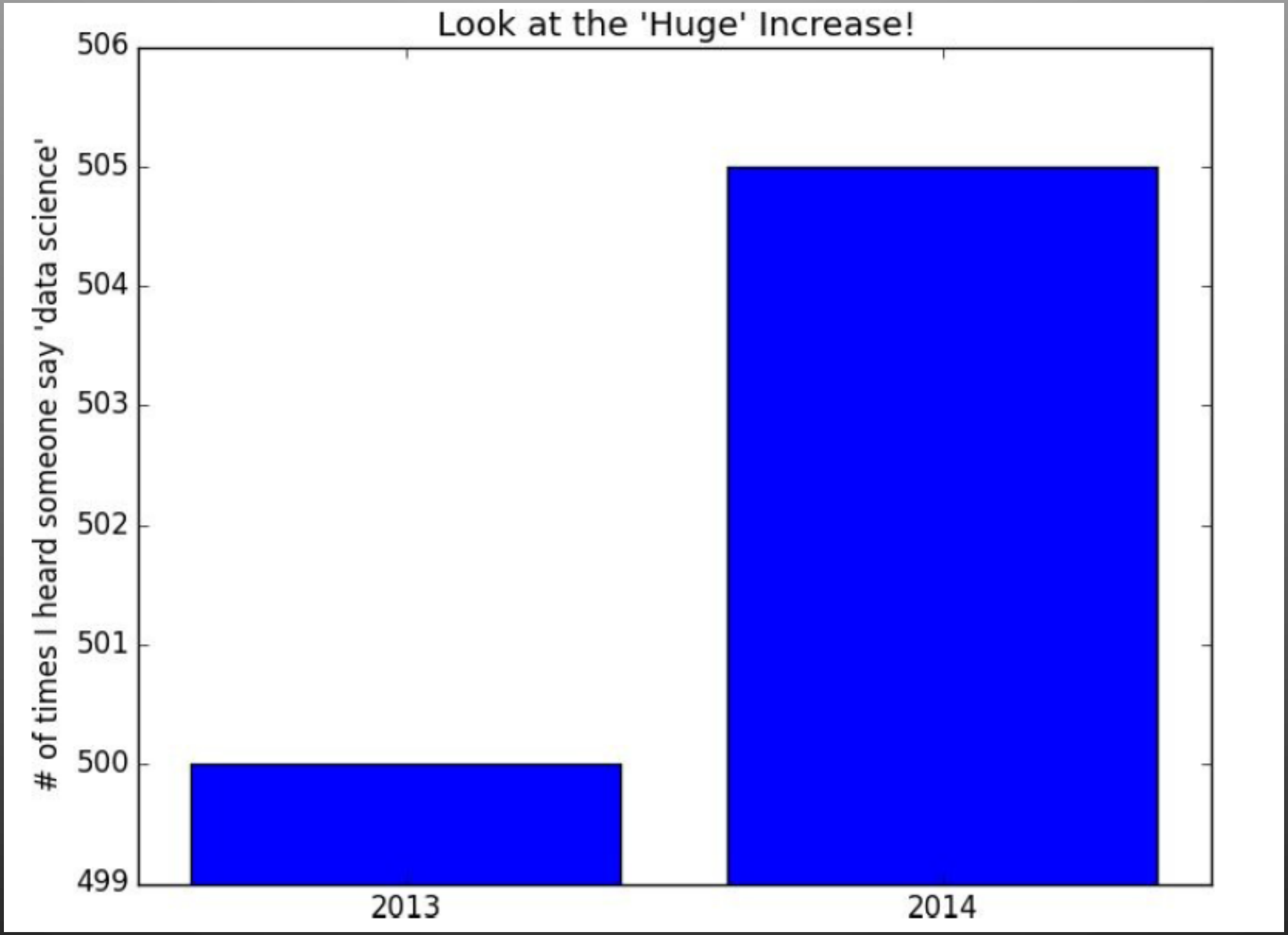
Bar Charts

- Be judicious when using plt.axis.
- When creating bar charts it is very bad form for your y-axis not to start at 0, since this is an easy way to mislead people.

Bar Charts

```
mentions = [500, 505]
years = [2013, 2014]
plt.bar([2012.6, 2013.6], mentions, 0.8)
plt.xticks(years)
plt.ylabel("#Number of times I heard someone say
'data science'")
plt.ticklabel_format(useOffset = False)
plt.axis([2012.5, 2014.5, 499, 506])
plt.title("Look at the Huge increase")
plt.show()
```

Bar Charts

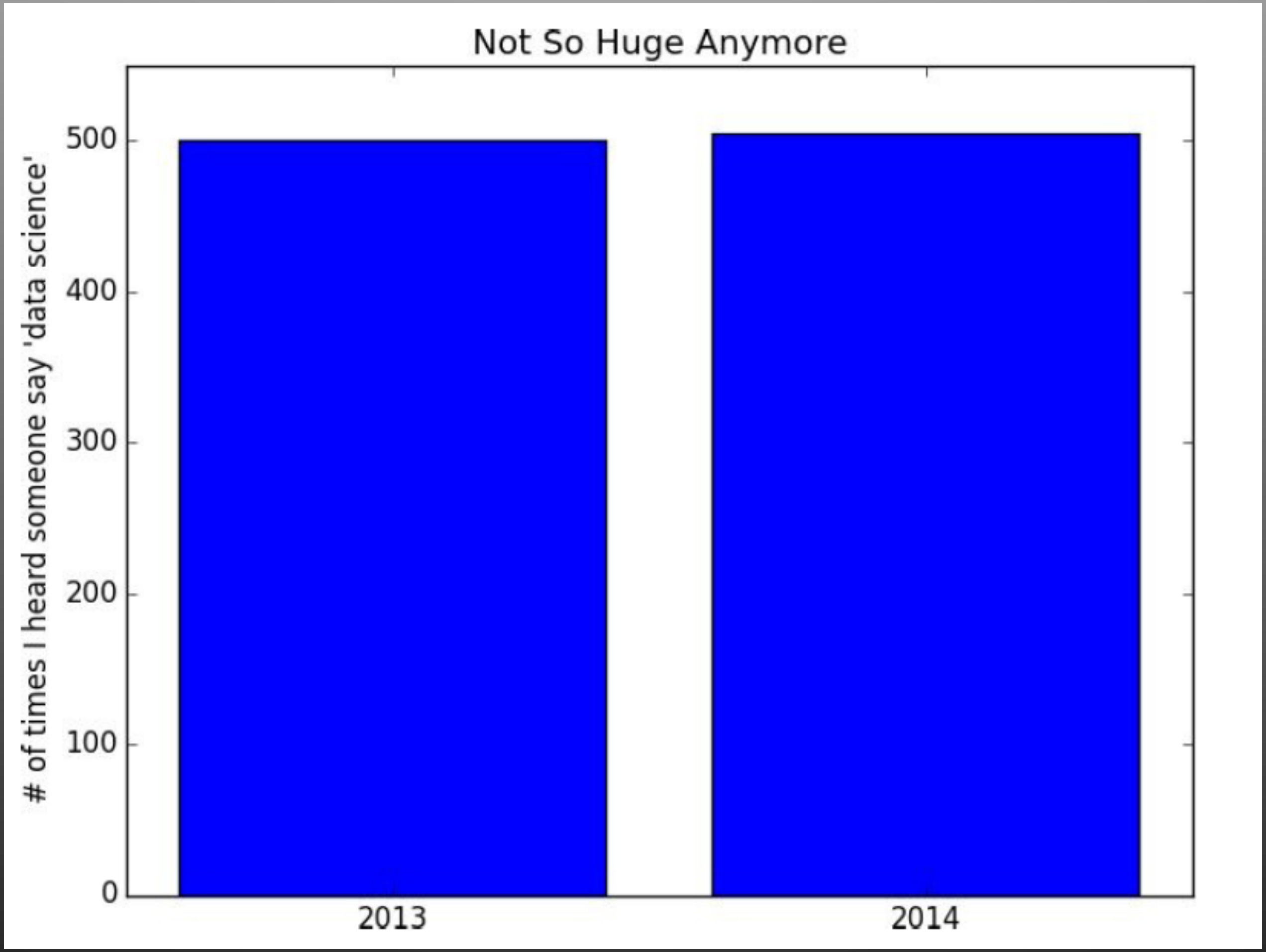

Look at the 'Huge' Increase!
# of times I heard someone say 'data science'
506
505
504
503
502
501
500
499
2013
2014

Bar Charts

- If we use more sensible aces it looks far less impressive:

```
plt.axis([2012.5, 2014.5, 0, 550])
plt.title(“Not so huge anymore”)
plt.show()
```

Bar Charts


Not So Huge Anymore
# of times I heard someone say 'data science'
0
100
200
300
400
500
2013
2014

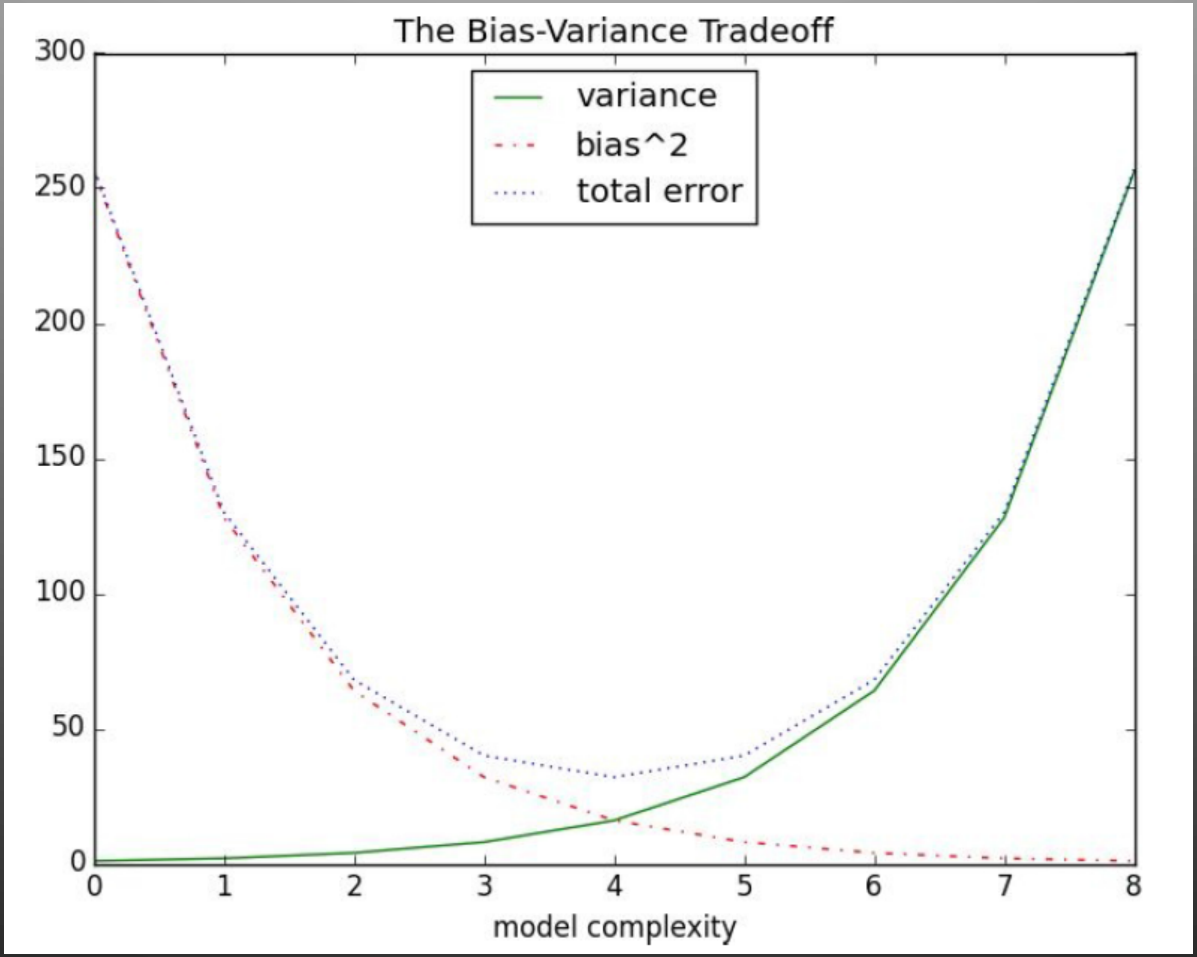
Line Charts

- We can make line charts using plt.plot()
- These are a good choice for showing trends

Line Charts

```
variance = [1,2,4,8,16,32,64,128,256]
bias_squared = [256, 128, 64,32,16,8,4,2,1]
total_error = [x+y for x,y in zip(variance, bias_squared)]
xs = [i for i,_ in enumerate(variance)]
#we can make multiple calls to plt.plot
#to show multiple series on the same plot
plt.plot(xs, variance, ‘g-’, label=‘variance’)
plt.plot(xs, bias_squared, ‘r-.’, label=‘bias^2’)
plt.plot(xs, total_error, ‘b:’, label=‘total error’)
plt.legend(loc = 9)
plt.xlabel(“model complexity”)
plt.title(“The Bias-Variance Tradeoff”)
plt.show()
```

Line Charts


The Bias-Variance Tradeoff
variance
bias^2
total error
model complexity
0
50
100
150
200
250
300
0
1
2
3
4
5
6
7
8

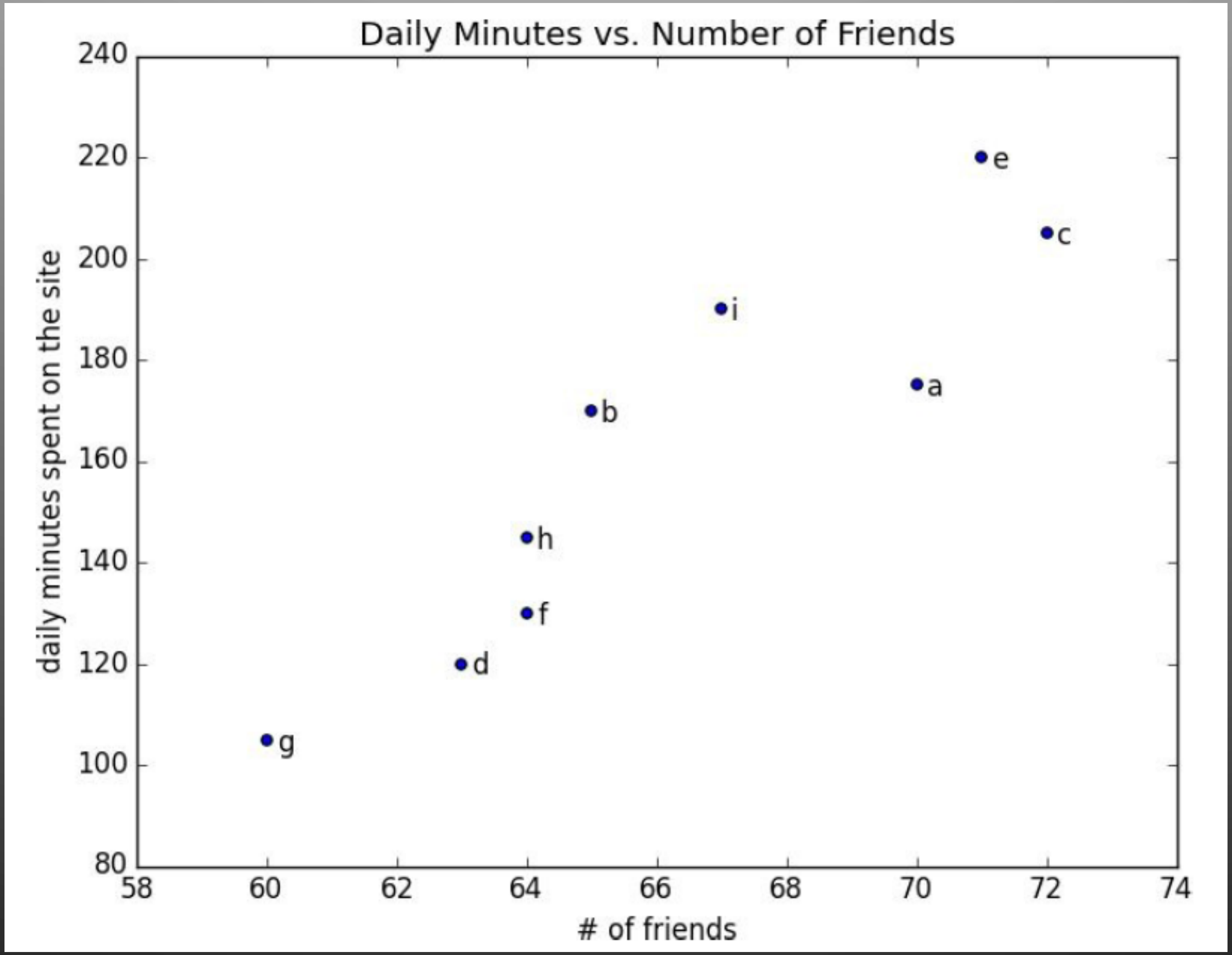
Scatterplots

- A scatterplot is the right choice for visualizing the relationship between two paired sets of data.
- The next graph shows the relationship between the number of friends your users have and the number of minutes they spend on the site every day.

Scatterplots

```
friends = [70, 65,72,63,71,64,60,64,67]
minutes = [175, 170, 205, 120, 220, 130, 105, 145, 190]
labels = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, ‘g’, ‘h’, ‘i’]
plt.scatter(friends, minutes)
for label,friend_count, minute_count in zip(labels, friends,
minutes):
   plt.annotate(label, xy=(friend_count,minute_count),
xytext=(5,-5), textcoords=‘offset points’)
plt.title(“Daily Minutes vs. Number of Friends”)
plt.xlabel(“# of friends”)
plt.ylabel(“daily minutes spend on the site”)
plt.show()
```

Scatterplots



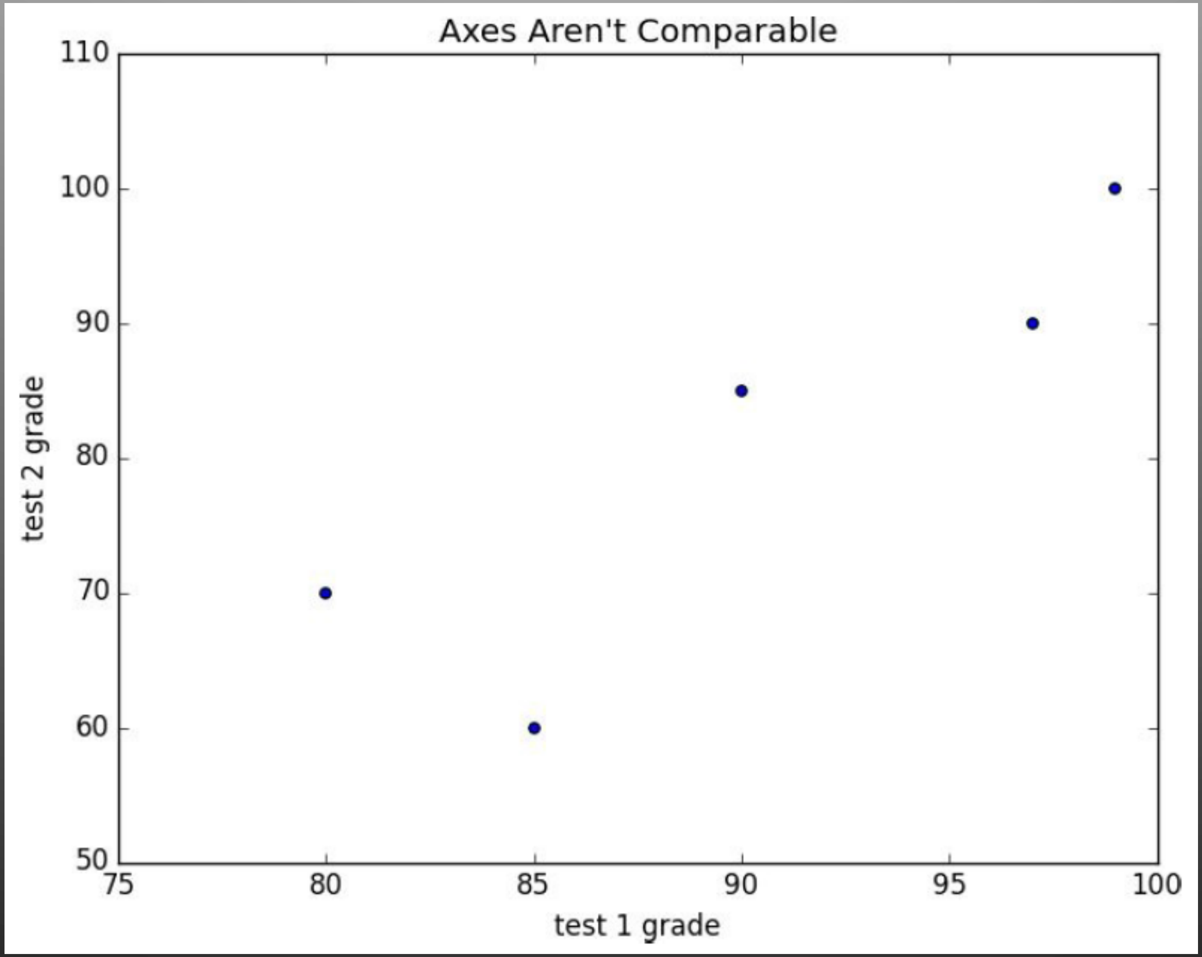
Scatterplots

- If you are scattering comparable variables, you might get a misleading picture if you let matplotlib choose the scale.

Scatterplots

```
test_1_grades = [99, 90, 85, 97, 80]
test_2_grades = [100, 85, 60, 90, 70]
plt.scatter(test_1_grades, test_2_grades)
plt.title(“Axes aren’t comparable”)
plt.xlabel(“test 1 grade”)
plt.ylabel(“test 2 grade”)
plt.show()
```

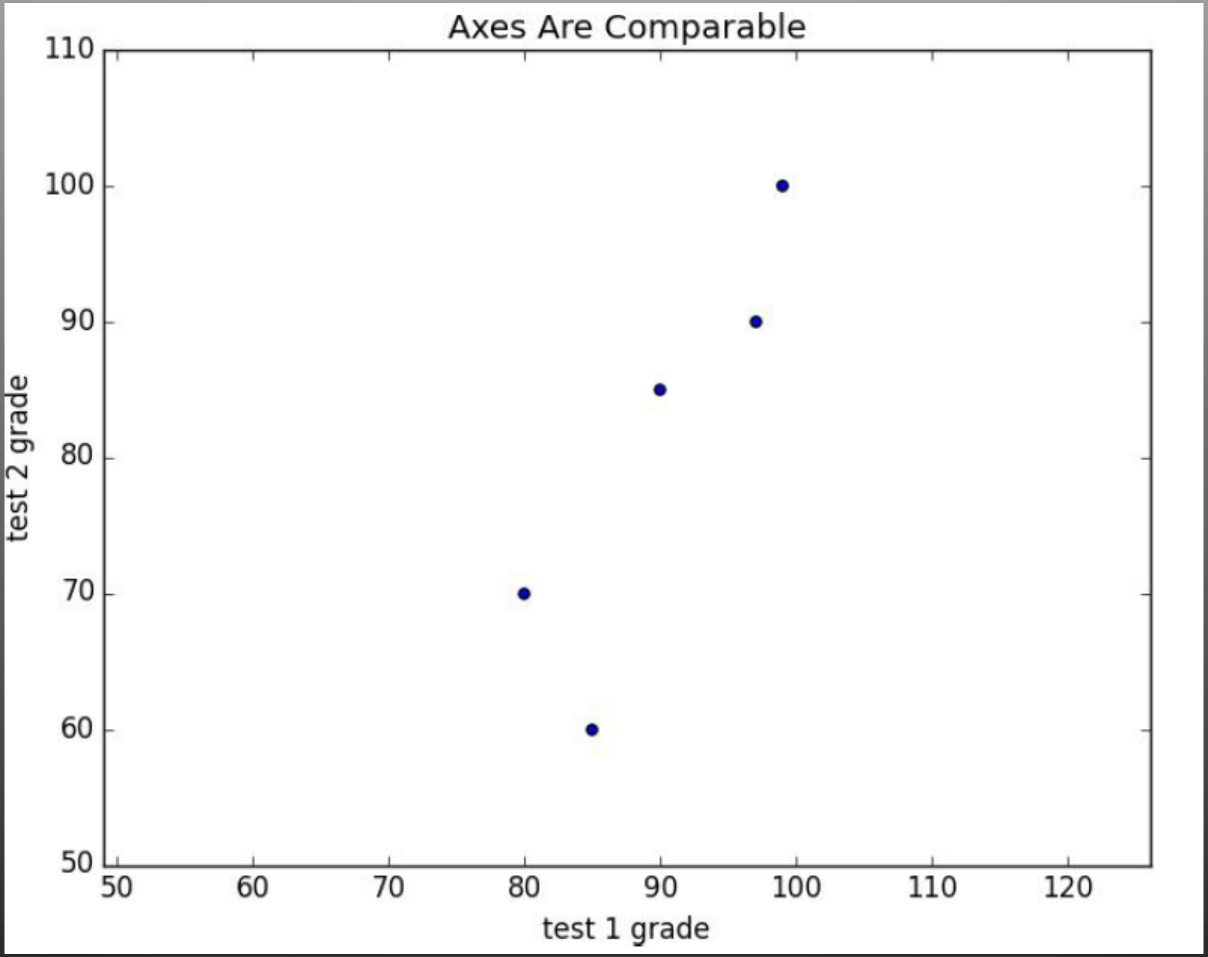
Scatterplots



Scatterplots

- If we include a call to plt.axis(“equal”), the plot more accurately shows that most of the variation occurs on test 2.

Scatterplots


Axes Are Comparable
test 2 grade
test 1 grade