

Machine Learning: Data Foundations + Algorithms & Applications

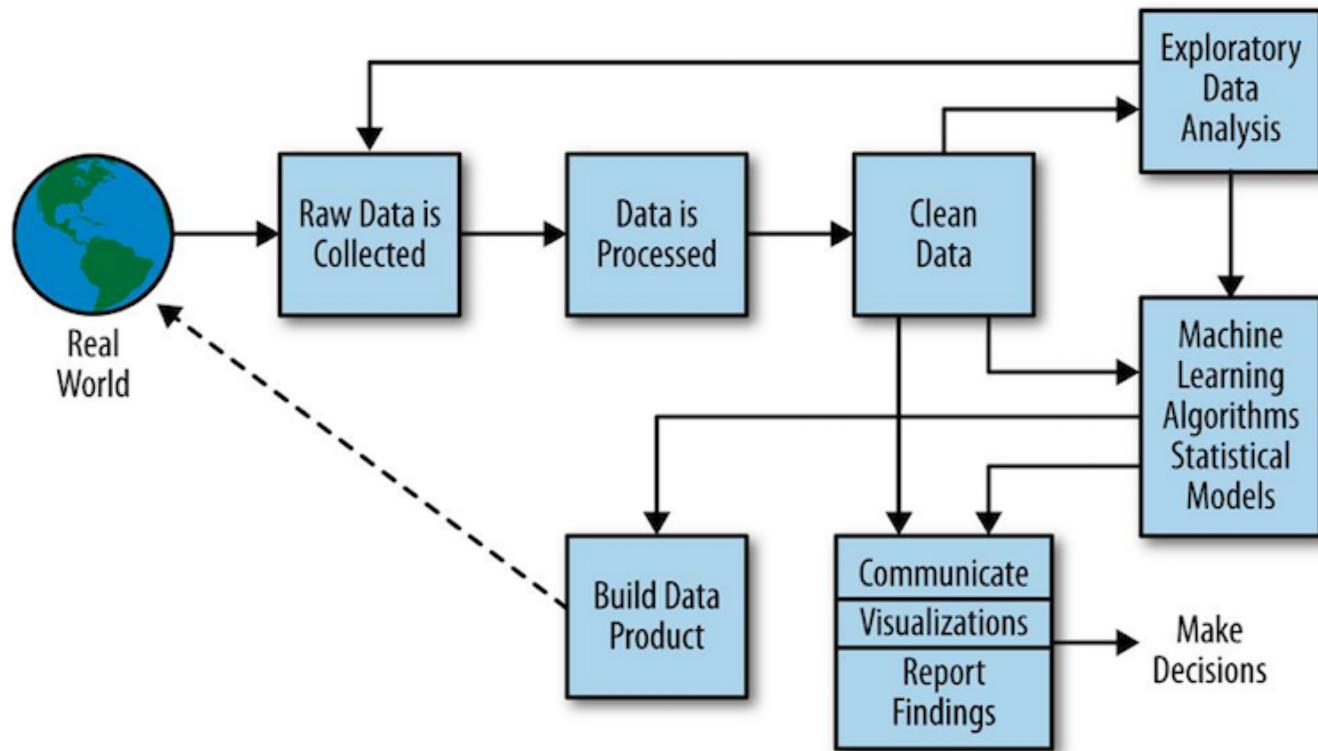
Day 3

Agenda - Day 3

- Introduction
- "Homework" (k-Means Iris) Review
- Python Numerics
- More! Pandas
- More! Data-Driven
- More! SciKit-Learn

Introduction

Where We Came From



Big Concepts

- Data as an Asset
- Role of Context
- Bad Data + ML = No Good
- Data Familiarity
- Curse of Dimensionality

Demo: "Homework" Review

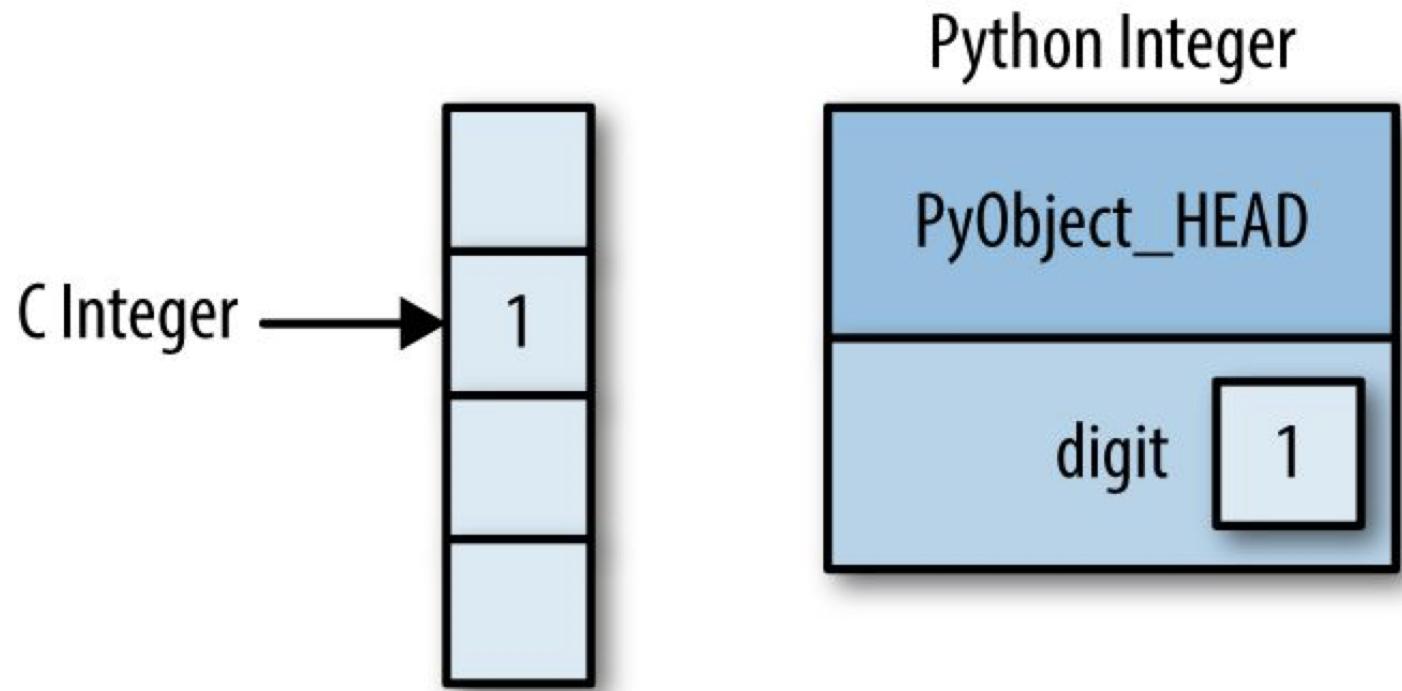
"Homework" Review

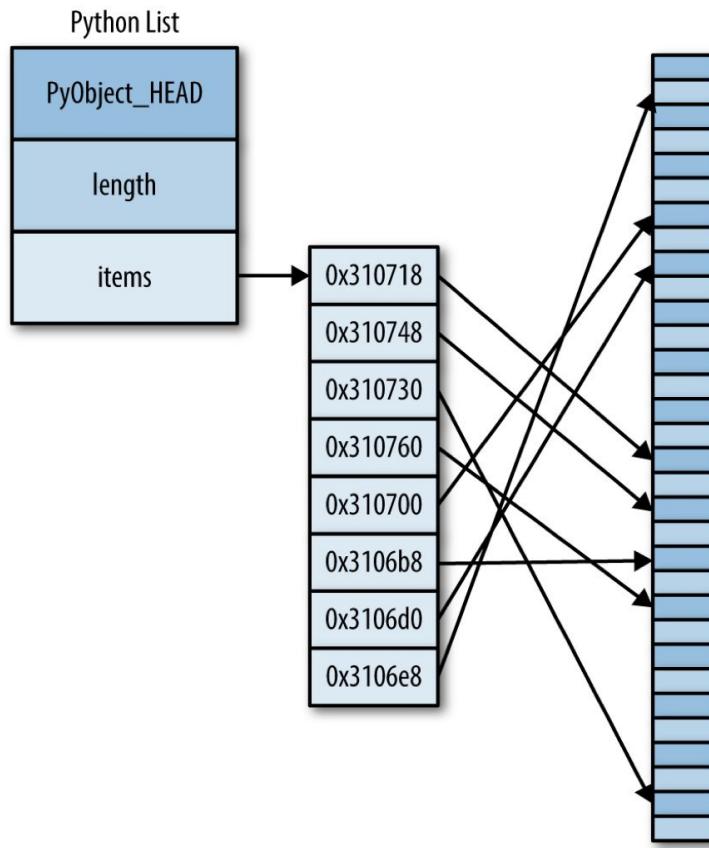
- let's open the notebook named **Demo - k-Means IRIS.ipynb** and go through it together

Demo: Python Numerics

Python Numerics

- **numpy** is the de facto numerical toolkit for Python (<http://www.numpy.org>)
- there are screenshots in this presentation to maintain continuity, but let's open the notebook named **Demo - Python Numerics.ipynb** and go through it together
- when done, click [here](#) to skip screenshots





```
import numpy as np
np.random.seed(0)

def compute_reciprocals(values):
    output = np.empty(len(values))
    for i in range(len(values)):
        output[i] = 1.0 / values[i]

    return output

values = np.random.randint(1, 10, size = 5)
compute_reciprocals(values)
```

```
Out[1]: array([ 0.16666667, 1. , 0.25 , 0.25 , 0.125 ])
```

```
In[2]: big_array = np.random.randint(1, 100, size = 1000000)
        %timeit compute_reciprocals(big_array)

1 loop, best of 3: 2.91 s per loop
```

```
import numpy as np
```

```
>>> x = 2.345
```

```
>>> x - 2.345  
0.0
```

```
>>> x - 2.345 == 0.0  
True
```

np.trunc()

- nearest integer i which is closer to zero than x is

```
>>> np.trunc(x)  
2.0
```

np.floor()

- the largest integer i , such that $i \leq x$

```
>>> np.floor(x)  
2.0
```

```
>>> np.floor(2.01)  
2.0
```

```
>>> np.floor(2.00)  
2.0
```

np.ceil()

- the smallest integer i , such that $i \geq x$

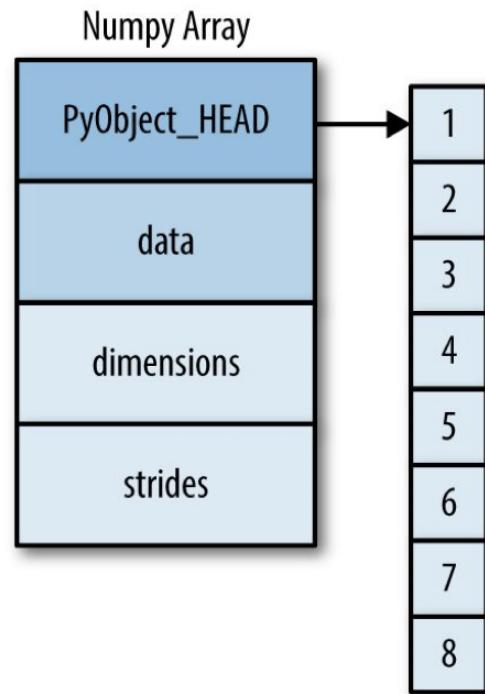
```
>>> np.ceil(x)  
3.0
```

```
>>> np.ceil(2.01)  
3.0
```

```
>>> np.ceil(2.0)  
2.0
```

```
>>> np.ceil(x) - 1  
2.0
```

```
>>> np.ceil(2.01) - 1  
2.0
```



```
>>> import numpy as np  
>>> np.array([1, 4, 2, 5, 3])  
array([1, 4, 2, 5, 3])
```

```
>>> np.array([3.14, 4, 2, 3])  
array([ 3.14,  4. ,  2. ,  3. ])
```

```
>>> np.array([1, 2, 3, 4], dtype='float32')  
array([ 1.,  2.,  3.,  4.], dtype=float32)
```

```
>>> np.array([range(i, i + 3) for i in [2, 4, 6]])  
array([[2, 3, 4],  
       [4, 5, 6],  
       [6, 7, 8]])
```

```
>>> np.zeros(10, dtype=int)  
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
>>> np.ones((3, 5), dtype=float)  
array([[ 1.,  1.,  1.,  1.,  1.],  
       [ 1.,  1.,  1.,  1.,  1.],  
       [ 1.,  1.,  1.,  1.,  1.]])
```

```
>>> np.full((3, 5), 3.14)  
array([[ 3.14,  3.14,  3.14,  3.14,  3.14],  
       [ 3.14,  3.14,  3.14,  3.14,  3.14],  
       [ 3.14,  3.14,  3.14,  3.14,  3.14]])
```

```
>>> np.arange(0, 20, 2)
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

```
>>> np.linspace(0, 1, 5)
array([ 0. ,  0.25,  0.5 ,  0.75,  1. ])
```

```
>>> np.random.random((3, 3))
array([[ 0.33687321,  0.57661167,  0.77955889],
       [ 0.7441428 ,  0.53766224,  0.62966105],
       [ 0.27348138,  0.27234265,  0.71558399]])
```

```
>>> np.random.normal(0, 1, (3, 3))
array([[ -2.02151788, -0.04433702, -0.16415692],
       [  0.28498092,  0.4638266 ,  0.64989753],
       [ -0.93943869, -1.90040282, -0.94711293]])
```

```
>>> np.random.randint(0, 10, (3, 3))
array([[ 7,  3,  0],
       [ 3,  3,  4],
       [ 9,  2,  6]])
```

```
>>> np.eye(3)
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

Converting array types

```
>>> x = np.linspace(0,10,50)
>>> x
array([ 0.          ,  0.20408163,  0.40816327,  0.6122449 ,
       0.81632653,  1.02040816,  1.2244898 ,  1.42857143,
       1.63265306,  1.83673469,  2.04081633,  2.24489796,
       2.44897959,  2.65306122,  2.85714286,  3.06122449,
       3.26530612,  3.46938776,  3.67346939,  3.87755102,
       4.08163265,  4.28571429,  4.48979592,  4.69387755,
       4.89795918,  5.10204082,  5.30612245,  5.51020408,
       5.71428571,  5.91836735,  6.12244898,  6.32653061,
       6.53061224,  6.73469388,  6.93877551,  7.14285714,
       7.34693878,  7.55102041,  7.75510204,  7.95918367,
       8.16326531,  8.36734694,  8.57142857,  8.7755102 ,
       8.97959184,  9.18367347,  9.3877551 ,  9.59183673,
       9.79591837,  10.         ])
```

```
>>> x.astype(int)
array([ 0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  2,  2,  2,  2,  2,  3,
       3,  3,  3,  4,  4,  4,  4,  4,  4,  5,  5,  5,  5,  5,  5,  6,  6,
       6,  6,  7,  7,  7,  7,  7,  8,  8,  8,  8,  8,  9,  9,  9,  9,  9,  10])
```

Multi-dimensional Arrays

```
>>> x2 = np.random.randint(10, size=(3, 4))
array([[3, 3, 0, 9],
       [0, 9, 7, 0],
       [5, 6, 2, 5]])
```

```
>>> x2[0, 0]
3
>>> x2[2, 0]
5
>>> x2[2, -1]
5
```

```
>>> x2[0, 0] = 12
>>> x2
array([[12,  3,  0,  9],
       [ 0,  9,  7,  0],
       [ 5,  6,  2,  5]])
```

```
>>> np.arange(0,9).reshape(3,3)
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

Numpy Array Slicing

```
>>> x = np.arange(10)
>>> x[:5]
array([0, 1, 2, 3, 4])
>>> x[5:]
array([5, 6, 7, 8, 9])
```

```
>>> x[4:7]
array([4, 5, 6])
```

```
>>> x[::-2]
array([0, 2, 4, 6, 8])
```

```
>>> x[1::2]
array([1, 3, 5, 7, 9])
```

```
>>> x[::-1]
array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

```
>>> x[5::-2]
array([5, 3, 1])
```

Filtering 1 dimensional data

```
>>> x = np.array([ 1, 1, 2, 1, 0, 3, 0, 0 ])
```

```
>>> np.nonzero(x)
(array([0, 1, 2, 3, 5]),)
```

```
>>> x[np.nonzero(x)]
array([1, 1, 2, 1, 3])
```

```
>>> x[np.nonzero(x < 3)]
array([1, 1, 2, 1, 0, 0, 0])
```

Filtering 2-dimensional data

```
>>> x = np.array([[1,0,0], [0,2,0], [1,1,0]])  
>>> x  
array([[1, 0, 0],  
       [0, 2, 0],  
       [1, 1, 0]])
```

```
>>> np.nonzero(x)  
(array([0, 1, 2, 2]), array([0, 1, 0, 1]))
```

```
>>> x[ np.nonzero(x) ]  
array([1, 2, 1, 1])
```

```
>>> np.transpose(np.nonzero(x))  
array([[0, 0],  
       [1, 1],  
       [2, 0],  
       [2, 1]])
```

Multi-dimensional Subarrays

```
>>> x2  
array([[12,  3,  0,  9],  
       [ 0,  9,  7,  0],  
       [ 5,  6,  2,  5]])
```

```
>>> x2[:2, :3]  
array([[12,  3,  0],  
       [ 0,  9,  7]])
```

```
>>> x2[:3, ::2]  
array([[12,  0],  
       [ 0,  7],  
       [ 5,  2]])
```

```
>>> x2[::-1, ::-1]  
array([[ 5,  2,  6,  5],  
       [ 0,  7,  9,  0],  
       [ 9,  0,  3, 12]])
```

Subarray Views

```
>>> x2
array([[12,  3,  0,  9],
       [ 0,  9,  7,  0],
       [ 5,  6,  2,  5]])
```

```
>>> x2_sub = x2[:2, :2]
>>> x2_sub
array([[12,  3],
       [ 0,  9]])
```

```
>>> x2_sub[0, 0] = 99
>>> x2_sub
array([[99,  3],
       [ 0,  9]])
```

```
>>> x2
array([[99,  3,  0,  9],
       [ 0,  9,  7,  0],
       [ 5,  6,  2,  5]])
```

Vectorized Operations

```
big_array = np.random.randint(1, 100, size=1000000)
```

```
In [2]: big_array = np.random.randint(1, 100, size=1000000)
....: %timeit (1.0 / big_array)
....:
100 loops, best of 3: 3.51 ms per loop
```

```
In [4]: big_array = np.random.rand(1000000)
....: %timeit sum(big_array)
....: %timeit np.sum(big_array)
10 loops, best of 3: 85.2 ms per loop
1000 loops, best of 3: 400 µs per loop
```

```
>>> x = np.arange(9).reshape((3, 3))
>>> 2 ** x
array([[ 1,   2,   4],
       [ 8,  16,  32],
       [ 64, 128, 256]])
```

```
>>> x = np.arange(4)
>>> -(0.5*x + 1) ** 2
array([-1. , -2.25, -4. , -6.25])
```

Operator	ufunc	Description
+	np.add	Addition (e.g., $1 + 1 = 2$)
-	np.subtract	Subtraction (e.g., $3 - 2 = 1$)
-	np.negative	Unary negation (e.g., -2)
*	np.multiply	Multiplication (e.g., $2 * 3 = 6$)
/	np.divide	Division (e.g., $3 / 2 = 1.5$)
//	np.floor_divide	Floor division (e.g., $3 // 2 = 1$)
**	np.power	Exponentiation (e.g., $2 ** 3 = 8$)
%	np.mod	Modulus/remainder (e.g., $9 \% 4 = 1$)

Aggregate ufuncs

```
>>> x = np.arange(1, 6)
>>> np.add.reduce(x)
15
>>> np.multiply.reduce(x)
120
>>> np.add.accumulate(x)
array([ 1,  3,  6, 10, 15])
>>> np.multiply.accumulate(x)
array([ 1,  2,  6, 24, 120])
```

Exercise: Python Numerics

(open the notebook named `Exercise 8 - Python Numerics.ipynb`)

Demo: More! Pandas

More! Pandas

- **pandas** is the de facto data analysis toolkit for Python (<http://pandas.pydata.org>)
- there are screenshots in this presentation to maintain continuity, but let's open the notebook named **Demo - More! Pandas.ipynb** and go through it together
- when done, click [here](#) to skip screenshots

Pandas Series

```
>>> import pandas as pd  
>>> data = pd.Series([0.25, 0.5, 0.75, 1.0])  
>>> data  
0    0.25  
1    0.50  
2    0.75  
3    1.00  
dtype: float64
```

```
>>> data.values  
array([ 0.25,  0.5 ,  0.75,  1. ])  
>>> data.index  
RangeIndex(start=0, stop=4, step=1)
```

```
>>> data[1]  
0.5
```

```
>>> data[1:3]  
1    0.50  
2    0.75  
dtype: float64
```

```
>>> data = pd.Series([0.25, 0.5, 0.75, 1.0], index=['a', 'b', 'c', 'd'])
```

```
>>> data
a    0.25
b    0.50
c    0.75
d    1.00
dtype: float64
```

```
>>> data['c']
0.75
```

```
>>> data = pd.Series([0.25, 0.5, 0.75, 1.0], index=[2, 5, 3, 7])
```

```
>>> data
2    0.25
5    0.50
3    0.75
7    1.00
dtype: float64
```

```
>>> data[5]
0.5
```

Implicit and Explicit Indexing

```
>>> data = pd.Series(['a', 'b', 'c'], index=[1, 3, 5])
>>> data
1    a
3    b
5    c
dtype: object
```

```
>>> data[1]
'a'
```

```
>>> data[1:3]
3    b
5    c
dtype: object
```

loc and iloc

```
>>> data  
1    a  
3    b  
5    c  
dtype: object
```

```
>>> data.loc[1]  
'a'  
>>> data.loc[1:3]  
1    a  
3    b  
dtype: object
```

```
>>> data.iloc[1]  
'b'  
>>> data.iloc[1:3]  
3    b  
5    c  
dtype: object
```

Python dicts as Series

```
>>> population_dict = {'California': 38332521,  
                      'Texas': 26448193,  
                      'New York': 19651127,  
                      'Florida': 19552860,  
                      'Illinois': 12882135}  
>>> population = pd.Series(population_dict)
```

```
>>> population  
California    38332521  
Florida       19552860  
Illinois      12882135  
New York      19651127  
Texas         26448193  
dtype: int64
```

```
>>> population['California']  
38332521
```

```
>>> population['California':'Illinois']  
California    38332521  
Florida       19552860  
Illinois      12882135  
dtype: int64
```

Pandas DataFrame

```
>>> area_dict = {'California': 423967, 'Texas': 695662, 'New York': 141297,
                 'Florida': 170312, 'Illinois': 149995}
>>> area = pd.Series(area_dict)
>>> area
California    423967
Florida       170312
Illinois      149995
New York      141297
Texas         695662
dtype: int64
```

```
>>> states = pd.DataFrame({'population': population,
                           'area': area})
>>> states
            area  population
California  423967    38332521
Florida     170312    19552860
Illinois    149995    12882135
New York    141297    19651127
Texas       695662    26448193
```

```
>>> states.index  
Index(['California', 'Florida', 'Illinois', 'New York', 'Texas'],  
      dtype='object')
```

```
>>> states.columns  
Index(['area', 'population'], dtype='object')
```

```
>>> states['area']  
California    423967  
Florida       170312  
Illinois      149995  
New York      141297  
Texas         695662  
Name: area, dtype: int64
```

```
>>> states.values  
array([[ 423967, 38332521],  
       [ 170312, 19552860],  
       [ 149995, 12882135],  
       [ 141297, 19651127],  
       [ 695662, 26448193]])
```

Sales Data

```
>>> dat = pd.read_csv("data/WA_Fn-UseC_-Sales-Win-Loss.csv")
>>> dat.columns
Index(['Opportunity Number', 'Supplies Subgroup', 'Supplies Group', 'Region',
       'Route To Market', 'Elapsed Days In Sales Stage', 'Opportunity Result',
       'Sales Stage Change Count', 'Total Days Identified Through Closing',
       'Total Days Identified Through Qualified', 'Opportunity Amount USD',
       'Client Size By Revenue', 'Client Size By Employee Count',
       'Revenue From Client Past Two Years', 'Competitor Type',
       'Ratio Days Identified To Total Days',
       'Ratio Days Validated To Total Days',
       'Ratio Days Qualified To Total Days', 'Deal Size Category'],
      dtype='object')
```

```
>>> dat['Opportunity Result']
0      Won
1     Loss
2      Won
3     Loss
4     Loss
5     Loss
6      Won
7     Loss
...
...
```

Counting Values

```
>>> dat['Opportunity Result'].value_counts()
Loss      60398
Won       17627
Name: Opportunity Result, dtype: int64
```

```
>>> dat['Supplies Group'].value_counts()
Car Accessories        49810
Performance & Non-auto    27325
Tires & Wheels            609
Car Electronics           281
Name: Supplies Group, dtype: int64
```

```
>>> dat['Elapsed Days In Sales Stage'].value_counts()
16      5010
44      2388
62      1738
7       1629
23      1455
37      1412
45      1238
24      1233
35      1226
...
...
```

Top Five values

```
>>> dat['Supplies Subgroup'].value_counts()[:5]
Motorcycle Parts      15174
Exterior Accessories 13876
Garage & Car Care    9733
Shelters & RV         9606
Batteries & Accessories 9192
Name: Supplies Subgroup, dtype: int64
```

Extracting Columns

```
>>> region_results = dat[["Region", "Opportunity Result"]]
>>> region_results.shape
(78025, 2)
>>> region_results.head()
0      Northwest      Won
1      Pacific      Loss
2      Pacific      Won
3          NaN      Loss
4      Pacific      Loss
```

Exercise: More! Pandas

(open the notebook named `Exercise 9 - More! Pandas.ipynb`)

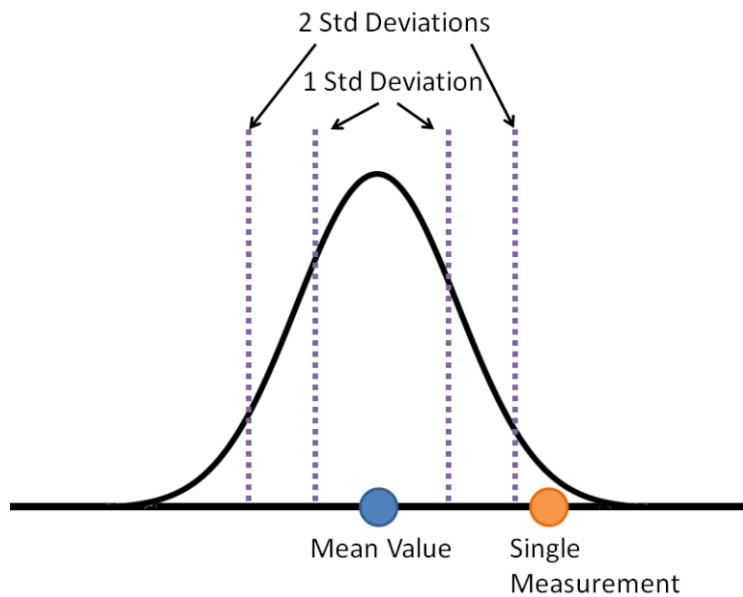
Data-Driven

Statistical Significance

- we'd like to determine how (un)likely is it that an outcome occurs by random chance
- if that probability is unlikely enough we conclude something other than random chance caused that outcome
- the two most important things for computing statistical significance are...
 - the distance the average value of your measured data is from what you are comparing it against
 - the standard deviation of what you are measuring

Standard Deviation

- if we make another measurement of the same type of data...
- ...it will fall somewhere on that curve with decreasing likelihood the farther you get away from the mean value

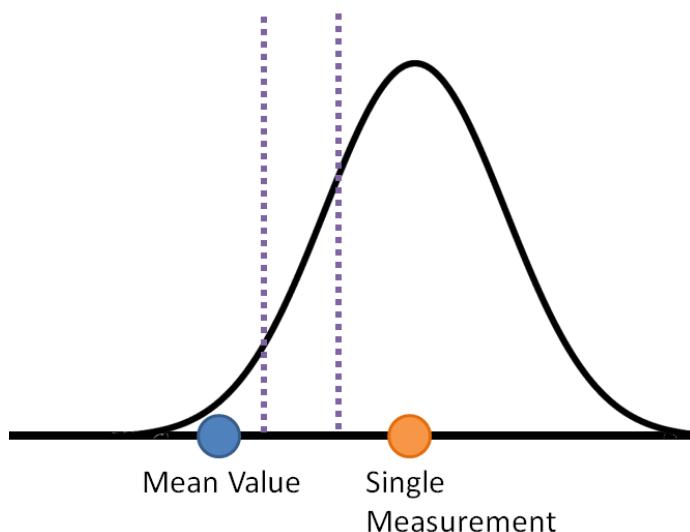


- let's turn this chart around and ask the question in reverse...

<http://www.fairlynerdy.com/intuitive-guide-statistical-significance/>

Hypothesis Testing

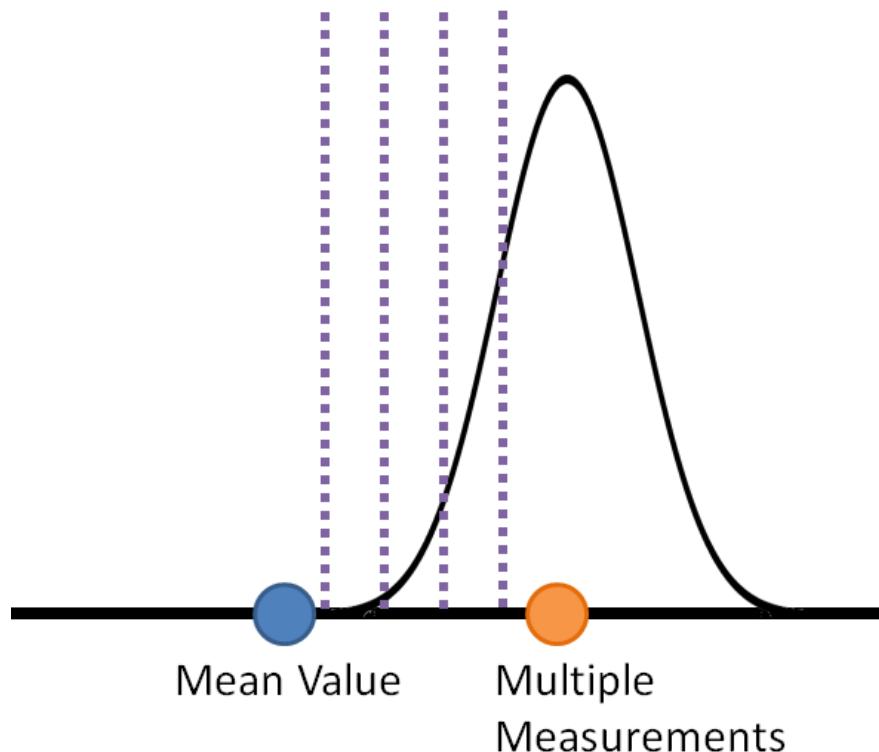
- Put a normal curve around our measured data and ask—"How likely is it that this came from the same source as the reference data?"
- i.e., how many standard deviations is it from our mean value?



- But how is this any different?
 - it isn't when we're talking about a single measurement...

What happens as we make more measurements?

- we don't care about the standard deviation of our data
- ...we care about the standard deviation of the mean value of all of our measurements
- ...which can change as we make additional measurements

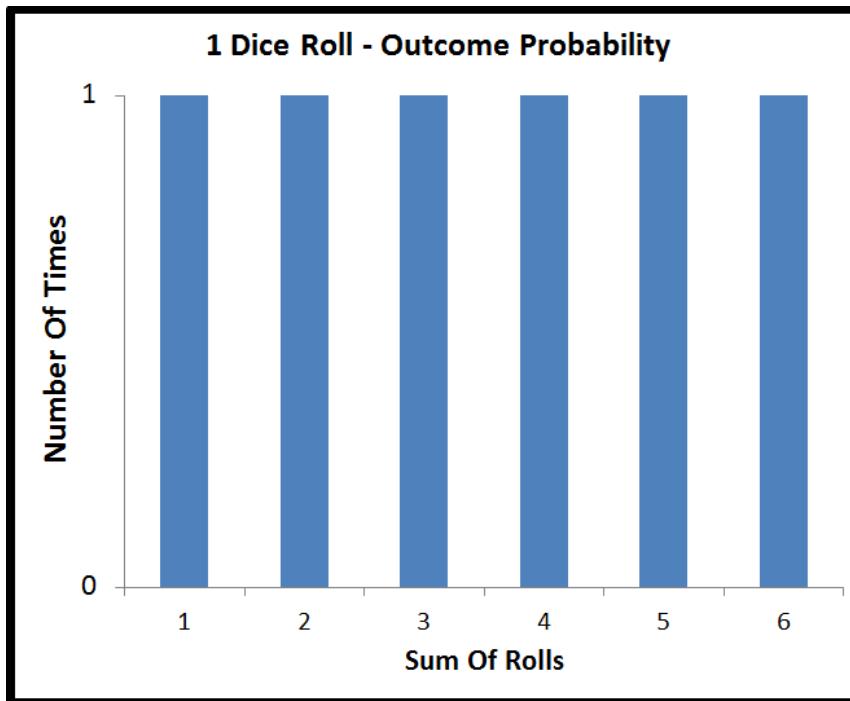


Why does the standard deviation of the mean decrease with more measurements?

- Let's what happens when you roll dice...



Roll a Single Die

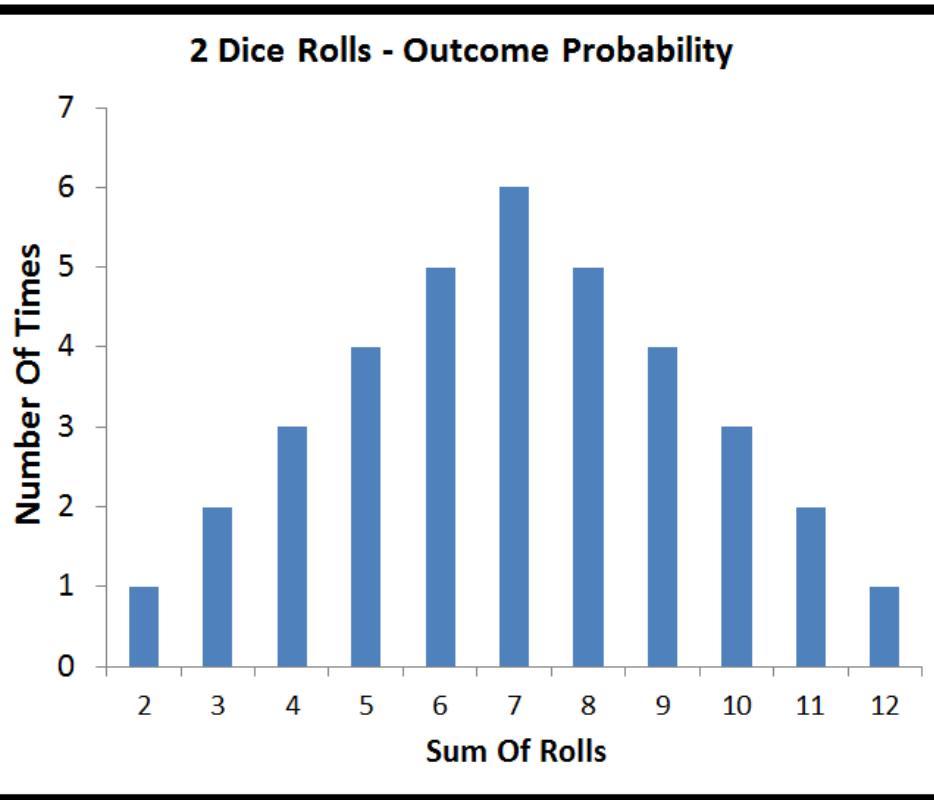


- No way to predict the outcome of a single die roll
- (i.e., no outcome is more likely than any other)
- What's the average roll?

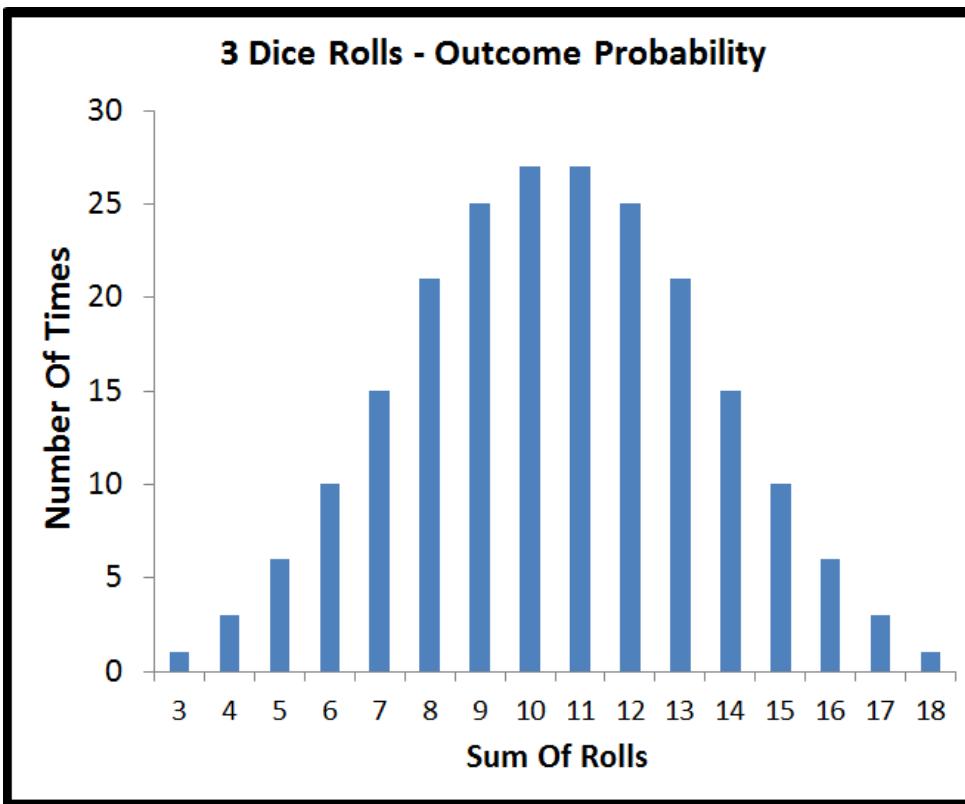
<http://www.fairlynerdy.com/intuitive-guide-statistical-significance/>

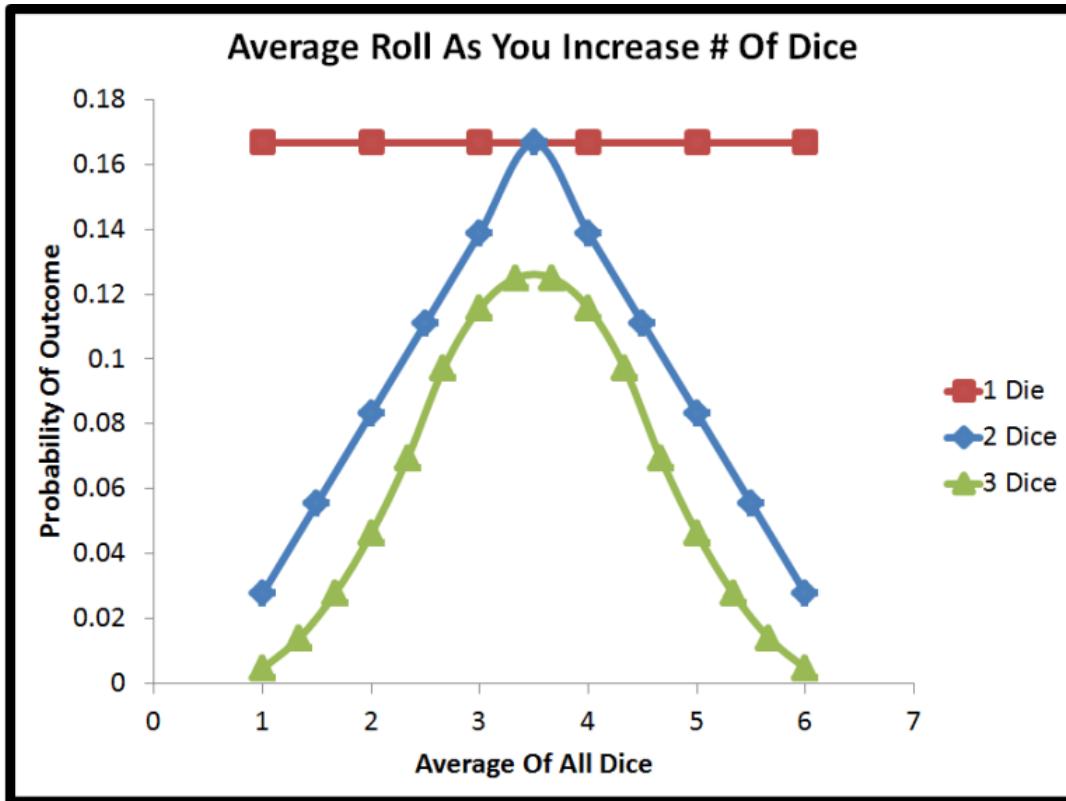


©
Copyright - Photomiqs

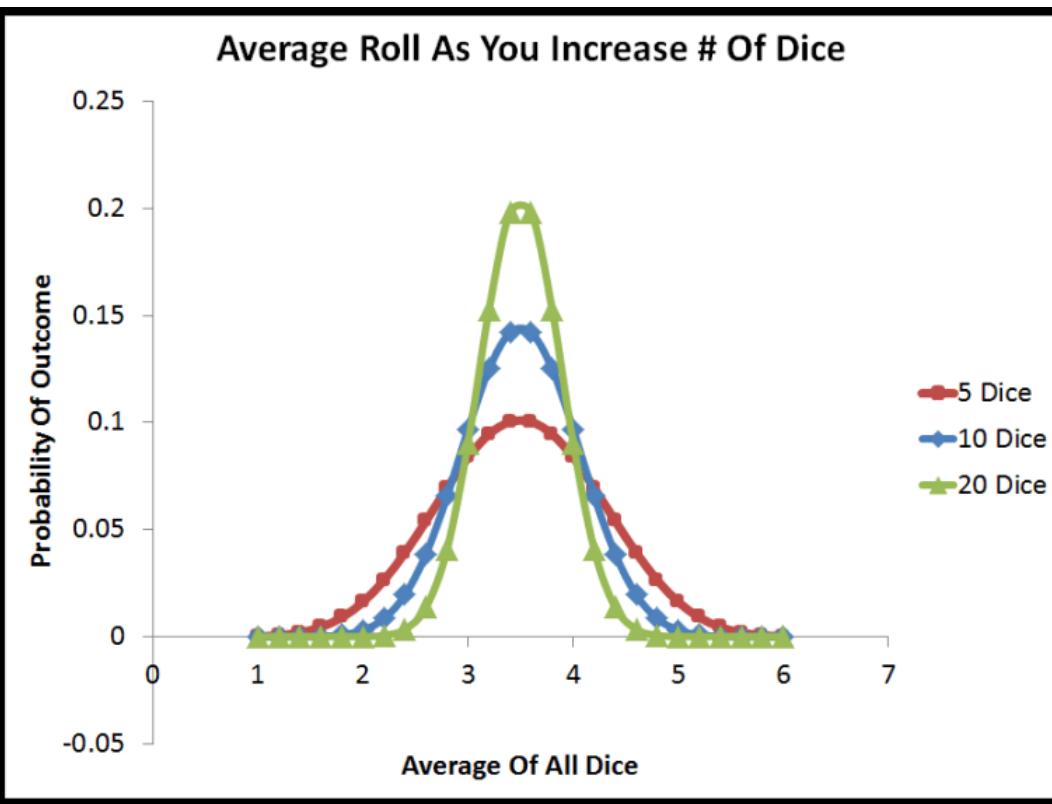


- $6 \times 6 = 36$ combinations, but only 11 values
- i.e., all of a sudden we can start to predict the sum of 2 dice
- Average roll?

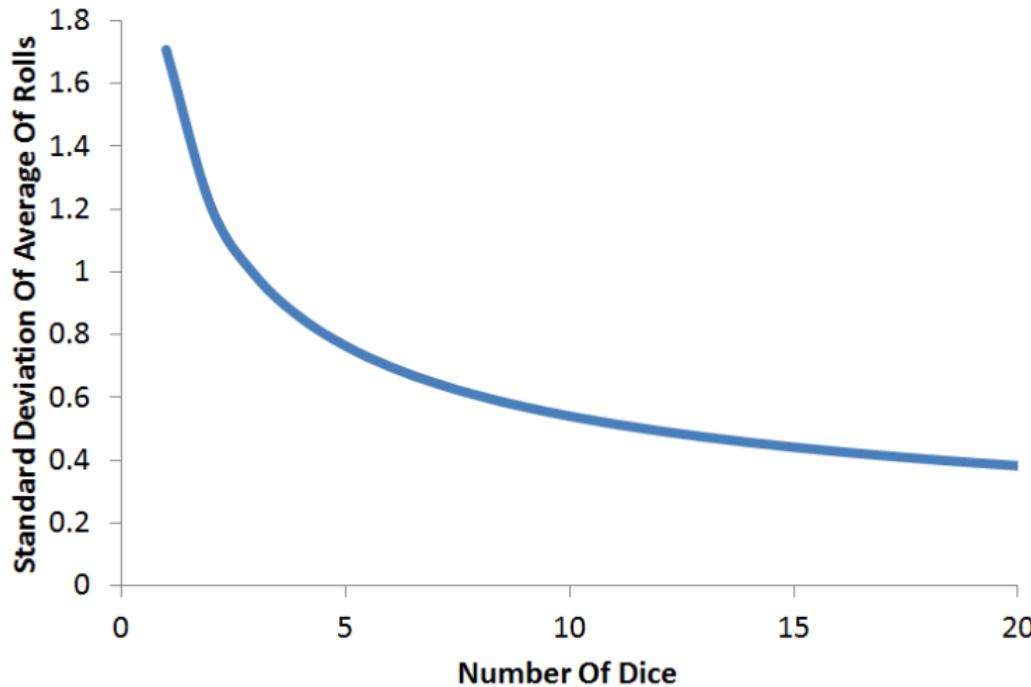




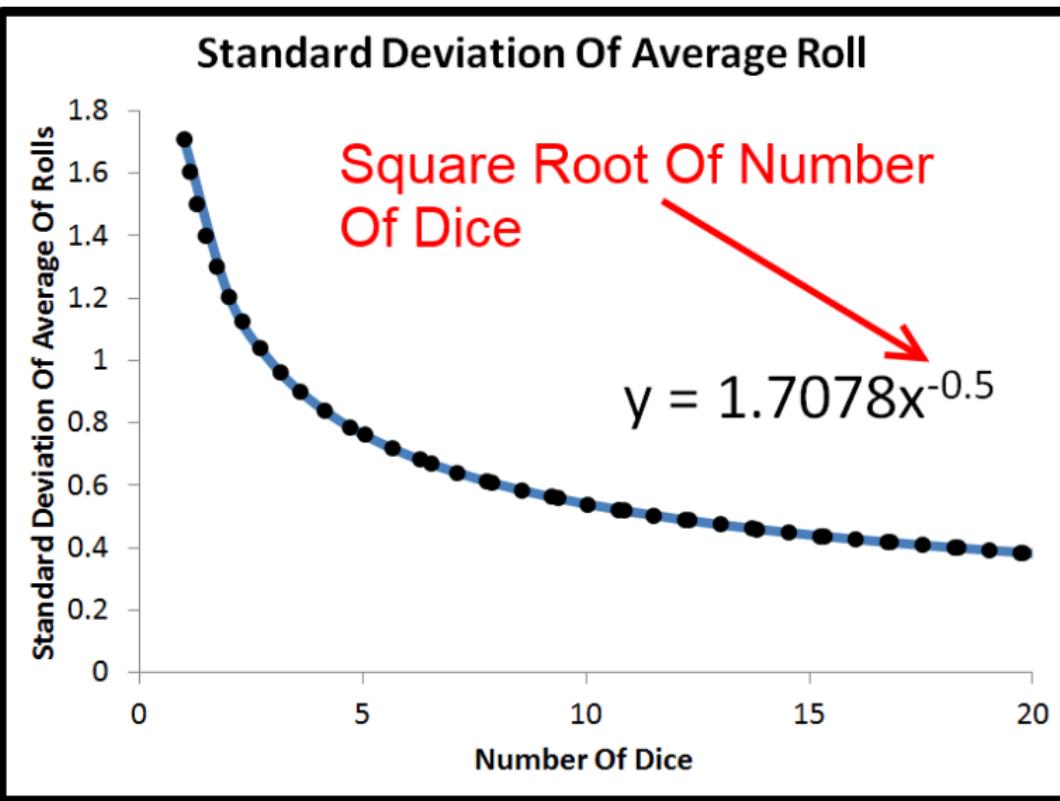
- by making multiple measurements, we can be more confident of their average value than we can in the result of any single measurement
- range of possible outcomes for the average is same as range of possible outcomes for a single roll, but distribution is more clustered in the center
- standard deviation of the mean of multiple measurements is lower than standard deviation of a single measurement



Standard Deviation Of Average Roll



```
>>> import numpy as np  
# compute standard deviation of one die roll  
>>> np.std((1, 2, 3, 4, 5, 6))  
1.707825127659933
```



$$y = 1.7078x^{-0.5}$$

$$y = \frac{1.7078}{\sqrt{x}}$$

$$y = \frac{\sigma}{\sqrt{n}}$$

Making Claims about a Population: Hypothesis Testing

Hypothesis Testing

- What we're asking is—"Could we expect to see this result simply by chance?"
- "...or are we demonstrating a specific result?"

Example: Evaluating Sales Training

- Suppose you have a bunch of salespeople, you hear about a training that makes people better at selling software—how do you know if it's worth it?
 - ...send a few people, see if it works
- How could software sold be different for the people you've sent?
 - What if they sell more?
 - What if they sell the same?
 - What if they sell less?
- When evaluating whether or not it's worth it maybe you need to delay measurement

Example: Birth Weight of Babies



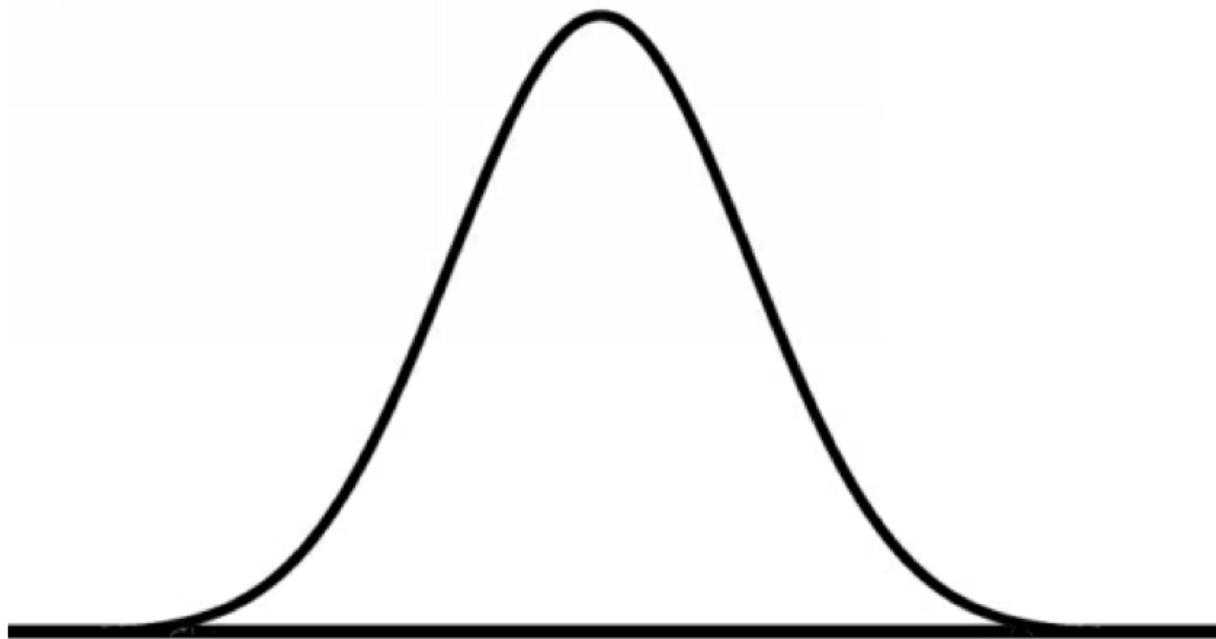
- Langston is one of 30 babies born in a particular hospital
- Suppose we want to know whether male babies born in that hospital are larger than average at birth?

Baby Number	Weight (lbs)
1	5.5
2	5.9
3	6
4	6
5	6.1
6	6.5
7	6.5
8	6.8
9	7
10	7.2
11	7.4
12	7.5
13	7.5
14	7.6
15	7.7
16	7.7
17	7.8
18	7.9
19	8
20	8
21	8.2
22	8.3
23	8.5
24	8.9
25	9
26	9.1
27	9.4
28	10
29	11
30	12
Average	7.8333

Compare Our Hospital to Population Mean

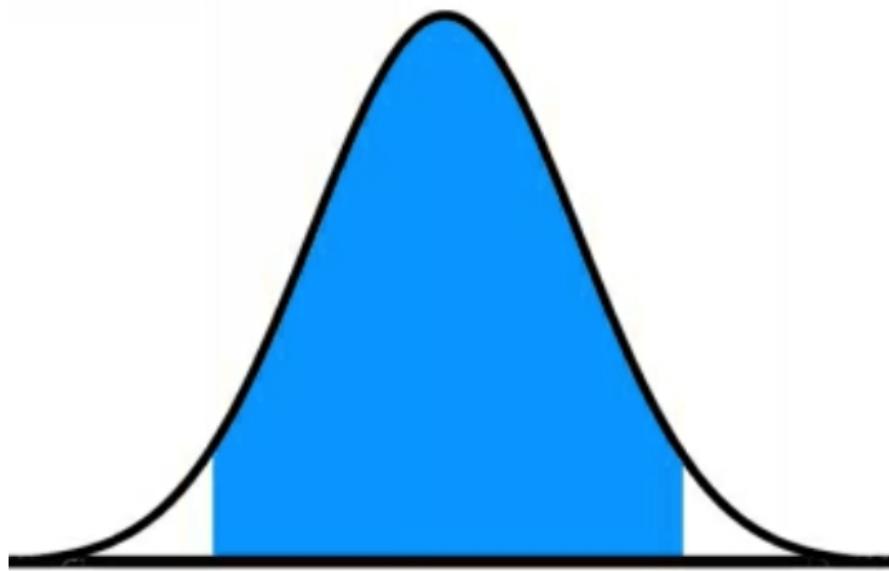
- avg. birth weight for a male baby in U.S. is 7.5 lbs. (population mean, μ)
- standard deviation of 1.25 lbs. (population standard deviation, σ)
- we desire a 95% confidence level that our babies are bigger

Standard Normal Curve



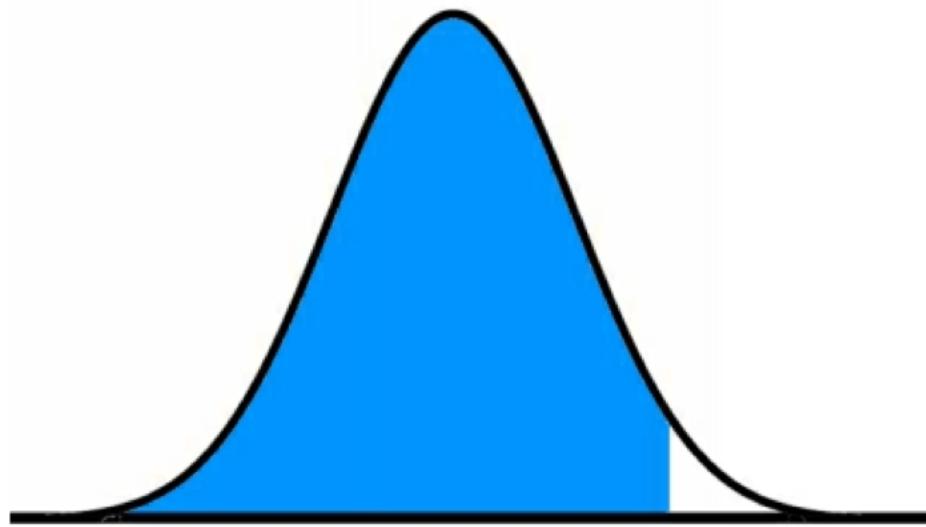
How do our hospital's babies compare to the overall population mean?

Standard Normal Curve
Filled From Middle
2 Tailed



- If we want to test that our babies do not weigh the same (i.e., more or less) as the average US male baby, then a two-tailed test would be appropriate

Standard Normal Curve
Filled From Left
i.e. 1 Tailed From Left



In our case, we are interested in seeing if our babies are *bigger* so we use a one-tailed test

Making the Claim: The Hypothesis

- To perform a hypothesis test, a null (H_0) and alternative (H_a) hypothesis needed to be stated.
- H_0 : Our babies are not significantly heavier than the population mean
- H_a : They are
- To determine if the difference is significant we use a Z-Test

Z Test

$$Z = \frac{\bar{x} - \mu}{\frac{\sigma}{\sqrt{n}}}$$

Variable	Meaning
\bar{x}	sample mean
μ	population mean
σ	population standard deviation
n	test sample size

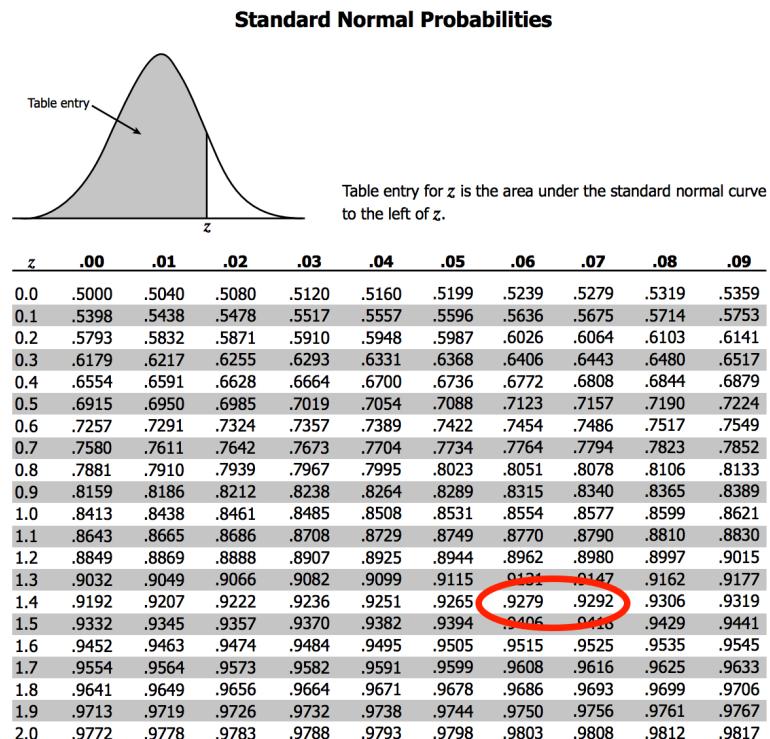
Z Test (Rewritten)

$$Z = \frac{\bar{x} - \mu}{\sigma} \times \sqrt{n}$$

$$Z = \frac{7.8333 - 7.5}{1.25} \times \sqrt{30}$$

$$Z = 1.461$$

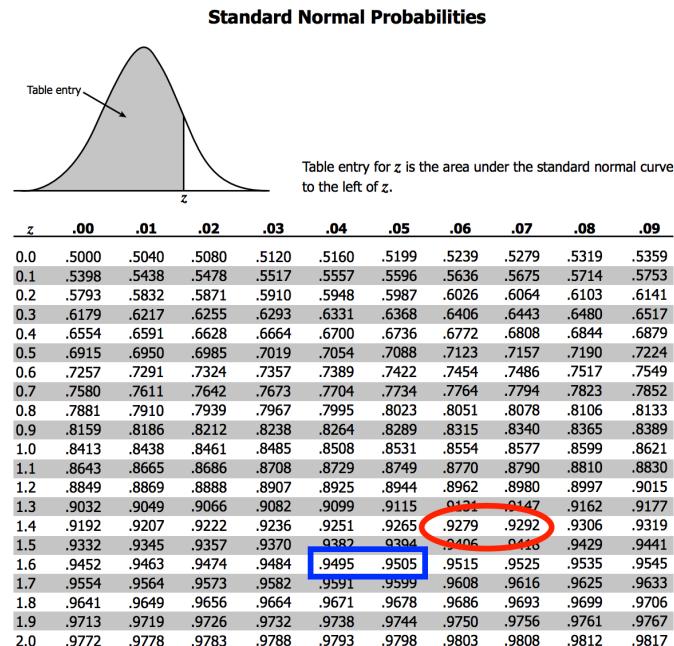
What Does Our Z Score Mean?



- it means we believe—with 92% confidence—our babies are bigger
- therefore...our results are not statistically significant

What Do We Do Now?

- We could get more data...
- Let's see go back to the Z table and see why...



- We need a Z score of 1.65—what sample size would be needed to reject the null hypothesis?

Z Test (Rewritten)

$$Z = \frac{\bar{x} - \mu}{\sigma} \times \sqrt{n}$$

$$1.65 = \frac{7.8333 - 7.5}{1.25} \times \sqrt{n}$$

$$n = 38.3$$

$$n = 39$$

- To be 95% confident our babies are bigger, we need a sample size of 39 (and of course the mean would have to be the same or larger)



Should we send our kid to an SAT prep course amid claims the course will increase your kid's score?

$$Z = \frac{\bar{x} - \mu_0}{\frac{\sigma}{\sqrt{n}}}$$

```
>>> import numpy as np
>>> import math
>>> mu = 500 # population mean
>>> sigma = 100 # population standard deviation
```

```
>>> samples = [434, 694, 457, 534, 720, 400, 484, 478, 610, 641, 425, 636,
              454, 514, 563, 370, 499, 640, 501, 625, 612, 471, 598, 509, 531]
>>> len(samples)
25
>>> np.mean(samples)
536.0
>>> xbar = np.mean(samples)
```

```
>>> z_score = (xbar - mu) / (sigma / math.sqrt(len(samples)))
>>> z_score
1.8
```

Standard Normal Probabilities

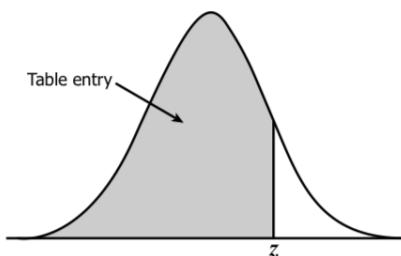


Table entry for z is the area under the standard normal curve to the left of z .

z	.00	.01	.02	.03	.04	.05	.06	.07	.08	.09
0.0	.5000	.5040	.5080	.5120	.5160	.5199	.5239	.5279	.5319	.5359
0.1	.5398	.5438	.5478	.5517	.5557	.5596	.5636	.5675	.5714	.5753
0.2	.5793	.5832	.5871	.5910	.5948	.5987	.6026	.6064	.6103	.6141
0.3	.6179	.6217	.6255	.6293	.6331	.6368	.6406	.6443	.6480	.6517
0.4	.6554	.6591	.6628	.6664	.6700	.6736	.6772	.6808	.6844	.6879
0.5	.6915	.6950	.6985	.7019	.7054	.7088	.7123	.7157	.7190	.7224
0.6	.7257	.7291	.7324	.7357	.7389	.7422	.7454	.7486	.7517	.7549
0.7	.7580	.7611	.7642	.7673	.7704	.7734	.7764	.7794	.7823	.7852
0.8	.7881	.7910	.7939	.7967	.7995	.8023	.8051	.8078	.8106	.8133
0.9	.8159	.8186	.8212	.8238	.8264	.8289	.8315	.8340	.8365	.8389
1.0	.8413	.8438	.8461	.8485	.8508	.8531	.8554	.8577	.8599	.8621
1.1	.8643	.8665	.8686	.8708	.8729	.8749	.8770	.8790	.8810	.8830
1.2	.8849	.8869	.8888	.8907	.8925	.8944	.8962	.8980	.8997	.9015
1.3	.9032	.9049	.9066	.9082	.9099	.9115	.9131	.9147	.9162	.9177
1.4	.9192	.9207	.9222	.9236	.9251	.9265	.9279	.9292	.9306	.9319
1.5	.9332	.9345	.9357	.9370	.9382	.9394	.9406	.9418	.9429	.9441
1.6	.9452	.9463	.9474	.9484	.9495	.9505	.9515	.9525	.9535	.9545
1.7	.9554	.9564	.9573	.9582	.9591	.9599	.9608	.9616	.9625	.9633
1.8	.9641	.9649	.9656	.9664	.9671	.9678	.9686	.9693	.9699	.9706
1.9	.9713	.9719	.9726	.9732	.9738	.9744	.9750	.9756	.9761	.9767

Standard Normal Probabilities

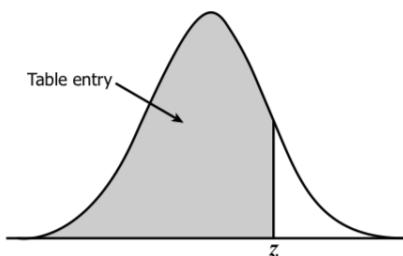


Table entry for z is the area under the standard normal curve to the left of z .

z	.00	.01	.02	.03	.04	.05	.06	.07	.08	.09
0.0	.5000	.5040	.5080	.5120	.5160	.5199	.5239	.5279	.5319	.5359
0.1	.5398	.5438	.5478	.5517	.5557	.5596	.5636	.5675	.5714	.5753
0.2	.5793	.5832	.5871	.5910	.5948	.5987	.6026	.6064	.6103	.6141
0.3	.6179	.6217	.6255	.6293	.6331	.6368	.6406	.6443	.6480	.6517
0.4	.6554	.6591	.6628	.6664	.6700	.6736	.6772	.6808	.6844	.6879
0.5	.6915	.6950	.6985	.7019	.7054	.7088	.7123	.7157	.7190	.7224
0.6	.7257	.7291	.7324	.7357	.7389	.7422	.7454	.7486	.7517	.7549
0.7	.7580	.7611	.7642	.7673	.7704	.7734	.7764	.7794	.7823	.7852
0.8	.7881	.7910	.7939	.7967	.7995	.8023	.8051	.8078	.8106	.8133
0.9	.8159	.8186	.8212	.8238	.8264	.8289	.8315	.8340	.8365	.8389
1.0	.8413	.8438	.8461	.8485	.8508	.8531	.8554	.8577	.8599	.8621
1.1	.8643	.8665	.8686	.8708	.8729	.8749	.8770	.8790	.8810	.8830
1.2	.8849	.8869	.8888	.8907	.8925	.8944	.8962	.8980	.8997	.9015
1.3	.9032	.9049	.9066	.9082	.9099	.9115	.9131	.9147	.9162	.9177
1.4	.9192	.9207	.9222	.9236	.9251	.9265	.9279	.9292	.9306	.9319
1.5	.9332	.9345	.9357	.9370	.9382	.9394	.9406	.9418	.9429	.9441
1.6	.9452	.9463	.9474	.9484	.9495	.9505	.9515	.9525	.9535	.9545
1.7	.9554	.9564	.9573	.9582	.9591	.9599	.9608	.9616	.9625	.9633
1.8	.9641	.9649	.9656	.9664	.9671	.9678	.9686	.9693	.9699	.9706
1.9	.9713	.9719	.9726	.9732	.9738	.9744	.9750	.9756	.9761	.9767

Standard Normal Probabilities

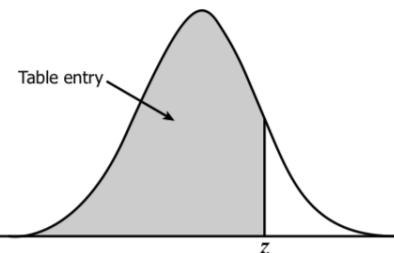
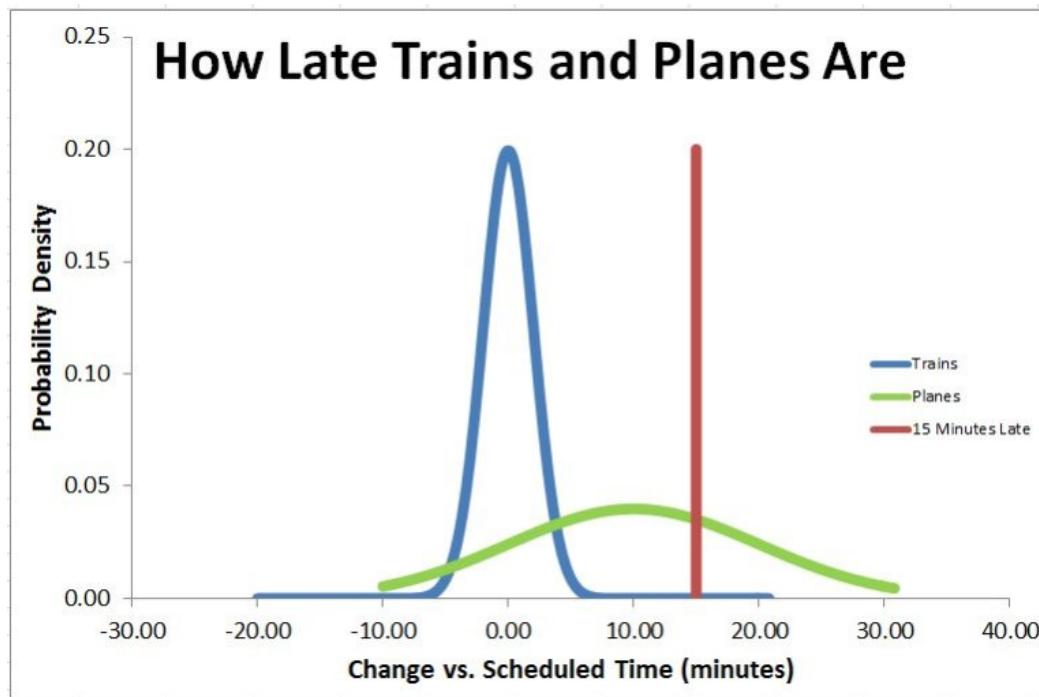


Table entry for z is the area under the standard normal curve to the left of z .

z	.00	.01	.02	.03	.04	.05	.06	.07	.08	.09
0.0	.5000	.5040	.5080	.5120	.5160	.5199	.5239	.5279	.5319	.5359
0.1	.5398	.5438	.5478	.5517	.5557	.5596	.5636	.5675	.5714	.5753
0.2	.5793	.5832	.5871	.5910	.5948	.5987	.6026	.6064	.6103	.6141
0.3	.6179	.6217	.6255	.6293	.6331	.6368	.6406	.6443	.6480	.6517
0.4	.6554	.6591	.6628	.6664	.6700	.6736	.6772	.6808	.6844	.6879
0.5	.6915	.6950	.6985	.7019	.7054	.7088	.7123	.7157	.7190	.7224
0.6	.7257	.7291	.7324	.7357	.7389	.7422	.7454	.7486	.7517	.7549
0.7	.7580	.7611	.7642	.7673	.7704	.7734	.7764	.7794	.7823	.7852
0.8	.7881	.7910	.7939	.7967	.7995	.8023	.8051	.8078	.8106	.8133
0.9	.8159	.8186	.8212	.8238	.8264	.8289	.8315	.8340	.8365	.8389
1.0	.8413	.8438	.8461	.8485	.8508	.8531	.8554	.8577	.8599	.8621
1.1	.8643	.8665	.8686	.8708	.8729	.8749	.8770	.8790	.8810	.8830
1.2	.8849	.8869	.8888	.8907	.8925	.8944	.8962	.8980	.8997	.9015
1.3	.9032	.9049	.9066	.9082	.9099	.9115	.9131	.9147	.9162	.9177
1.4	.9192	.9207	.9222	.9236	.9251	.9265	.9279	.9292	.9306	.9319
1.5	.9332	.9345	.9357	.9370	.9382	.9394	.9406	.9418	.9429	.9441
1.6	.9452	.9463	.9474	.9484	.9495	.9505	.9515	.9525	.9535	.9545
1.7	.9554	.9564	.9573	.9582	.9591	.9599	.9608	.9616	.9625	.9633
1.8	.9641	.9649	.9656	.9664	.9671	.9678	.9686	.9693	.9699	.9706
1.9	.9713	.9719	.9726	.9732	.9738	.9744	.9750	.9756	.9761	.9767

More Statistical Significance...



- your plane being 15 minutes late wouldn't be unusual
- but if you were at a Japanese bullet train station...

"In the past century, data collection often was carried out by statistical sampling since data was manually collected and therefore expensive. The prudent researcher would design the process to maximize the information that could be extracted from the data. In this century, data are arriving by firehose and without design."

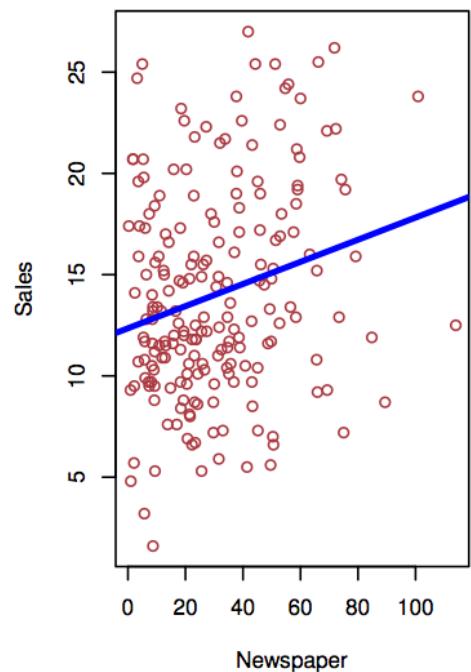
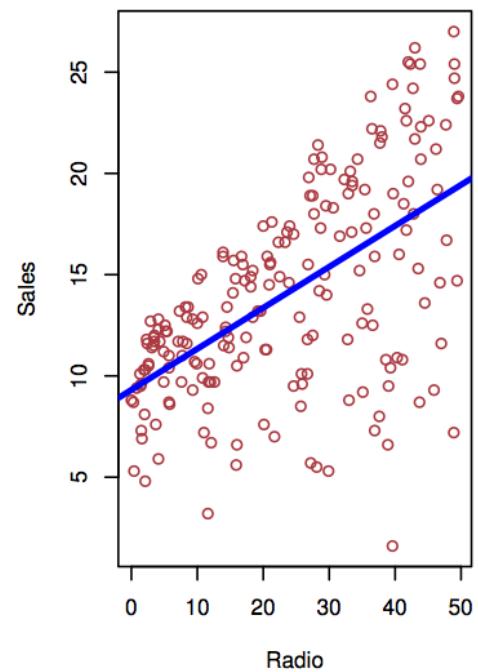
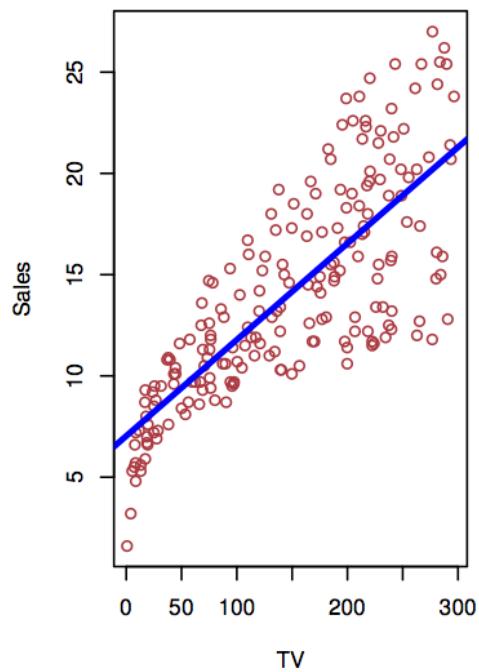
"Algorithms for Data Science", Steele et al

"Since statistical science revolves around the analysis of relatively small and information-rich data, statistical philosophy and principles are at times marginally relevant to the task at hand."

"Steele, Brian; Chandler, John; Reddy, Swarna. Algorithms for Data Science (Kindle Locations 711-712). Springer International Publishing. Kindle Edition."

Statistical Learning

Advertising Effectiveness



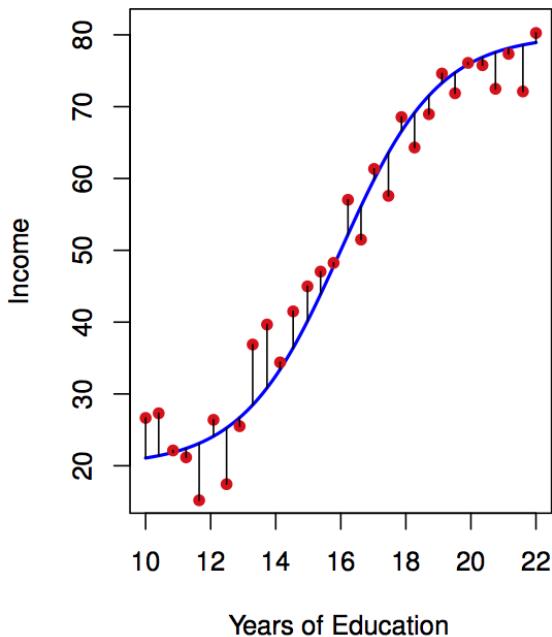
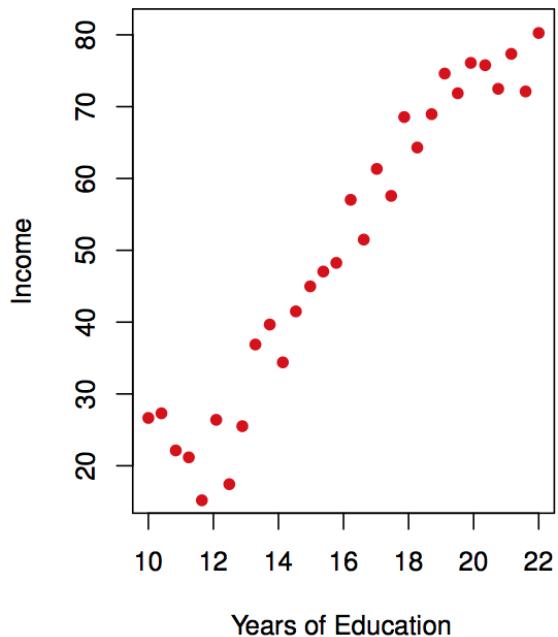
Reality

$$Y = f(X) + \epsilon$$

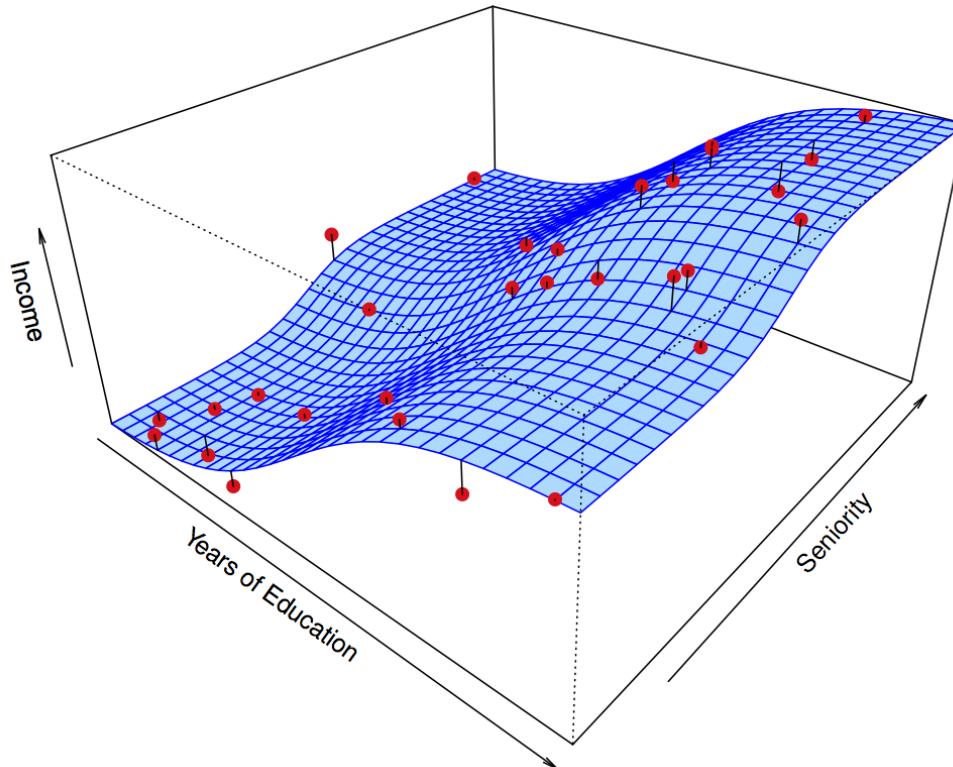
Some function maps the inputs to the output

ϵ is a random error term, which is independent of X and has mean zero

Does Income Relate to Years of Education?



What if we add another dimension?



Prediction

$$\hat{Y} = \hat{f}(X)$$

We build a model which approximates the true function

Why *isn't* there an error term here?

Types of Errors

- Reducible
 - How do we reduce this?
- Irreducible

Compare Reality to Our Predictions

$$E(Y - \hat{Y})^2 = E[f(X) + \epsilon - \hat{f}(X)]^2$$

$$= \underbrace{[f(X) - \hat{f}(X)]^2}_{\textit{Reducible}} + \underbrace{\textit{Var}(\epsilon)}_{\textit{Irreducible}}$$

Estimating $f()$

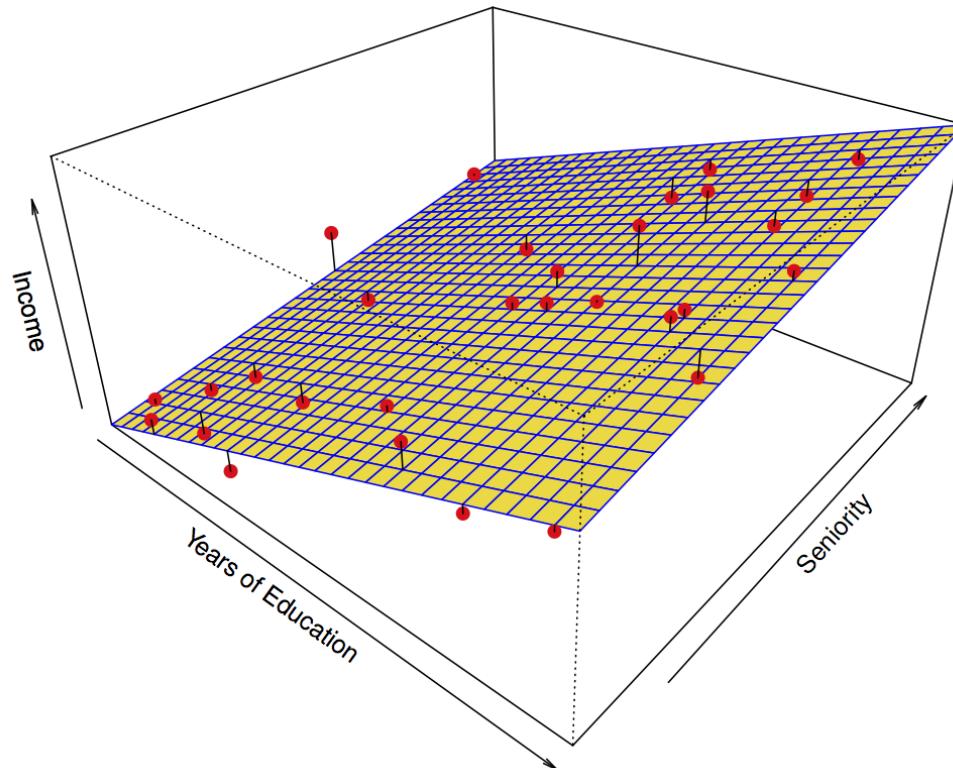
- Parametric methods
- Non-parametric methods

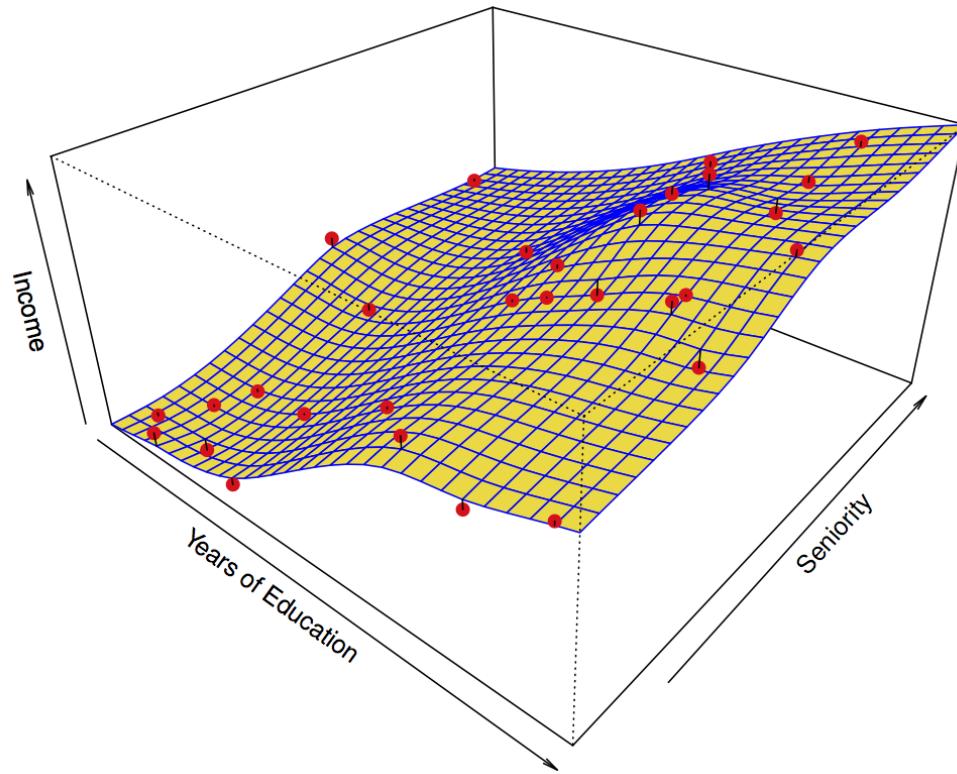
Multivariate Linear Regression

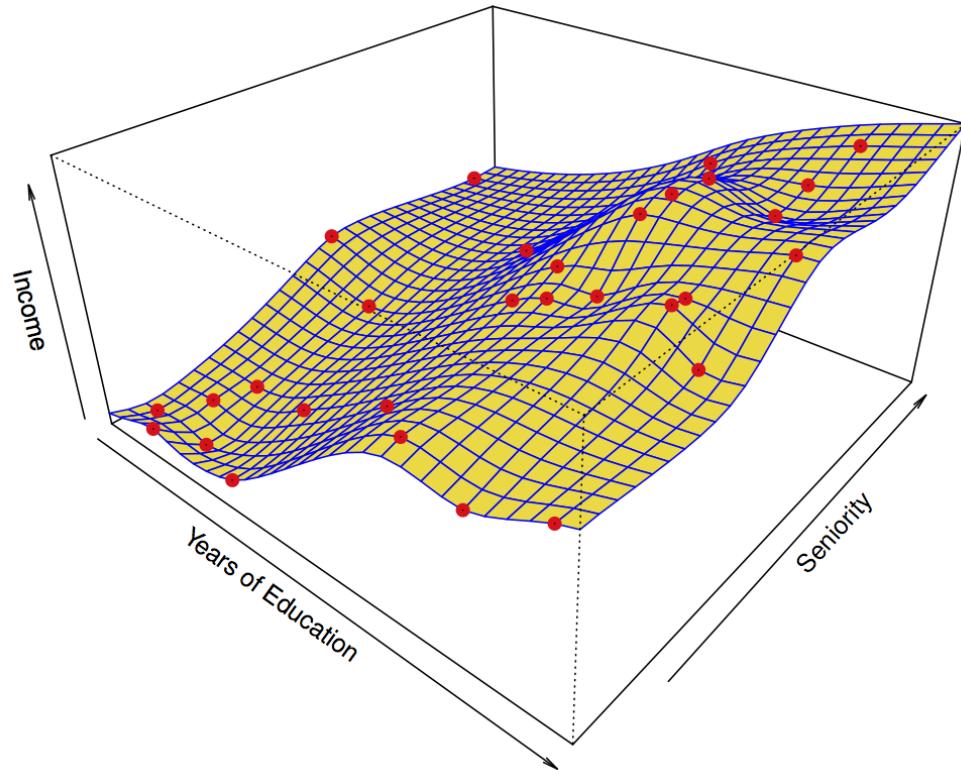
$$f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p$$

$$Y \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p$$

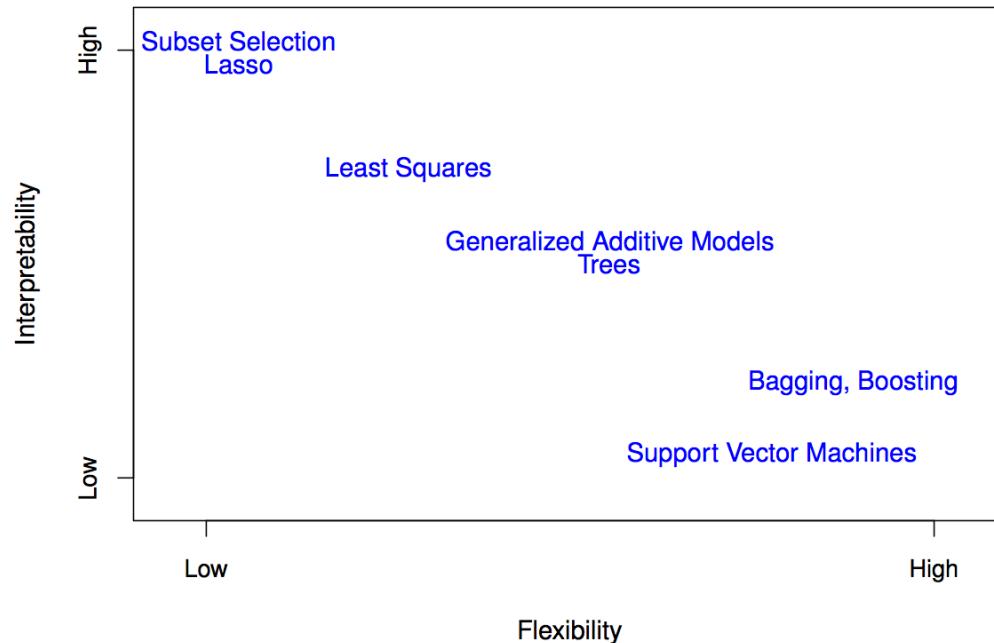
$$\text{income} \approx \beta_0 + \beta_1 \times \text{education} + \beta_2 \times \text{seniority}$$







Interpretability vs. Flexibility Tradeoff



Measuring Fitness: Mean Squared Error

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{f}(x_i))^2$$

Training MSE

Training data = $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$

Compute $\hat{f}(x_1), \hat{f}(x_2), \dots, \hat{f}(x_n)$

$\hat{f}(x_i) \approx y_i$ means training MSE is small

Training MSE (cont'd)

However, training MSE doesn't really matter

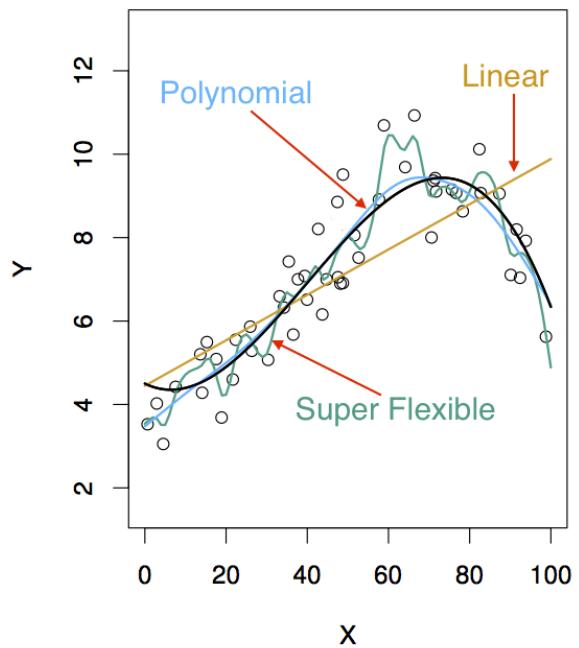
What's much more important is...

test MSE!

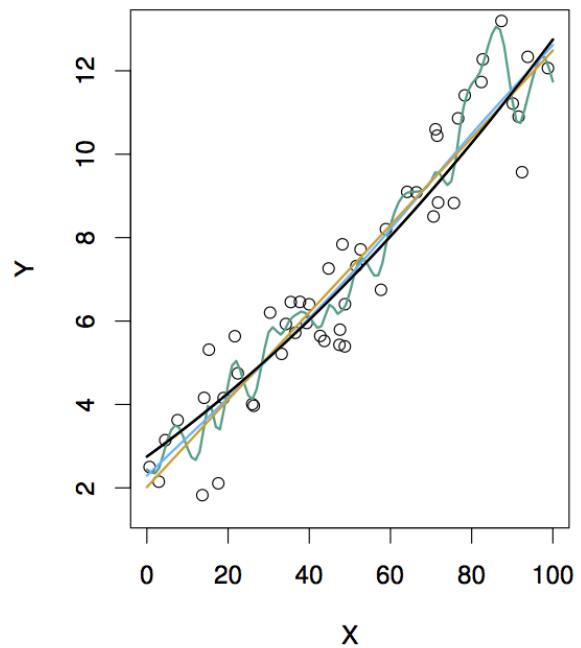
$$\hat{f}(x_{n+1}), \dots, \hat{f}(x_{n+m}) \approx y_{n+1}, \dots, y_{n+m}$$

$$\frac{1}{m} \sum_{j=n+1}^{n+m} (\hat{f}(x_j) - y_j)^2$$

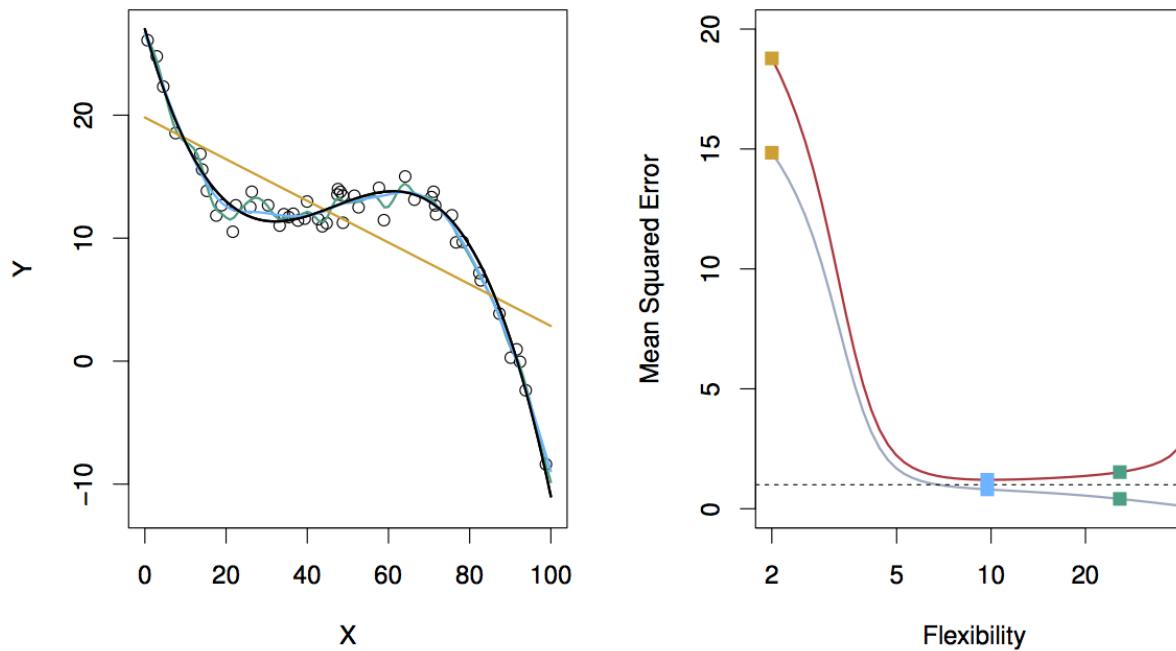
Comparing Models with MSE Curves



More MSE Curves



Even More MSE Curves

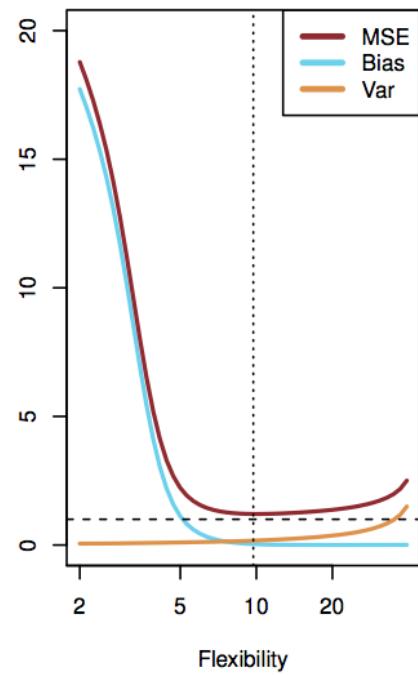
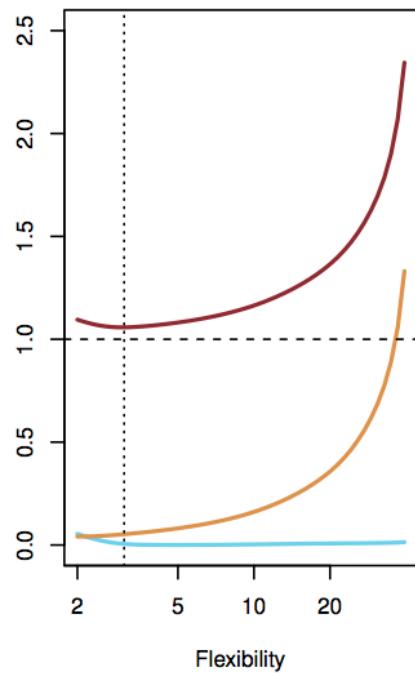
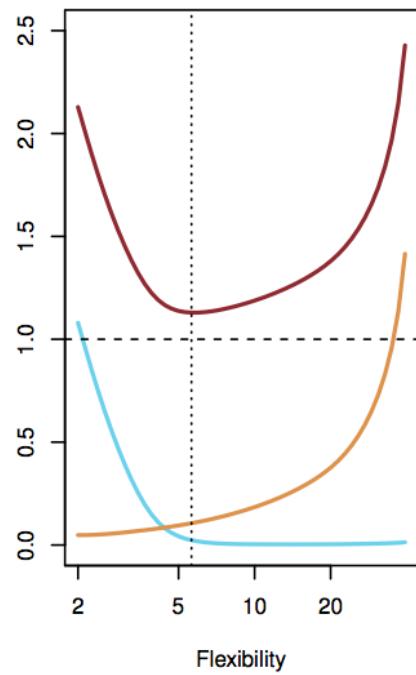


Bias-Variance Trade-Off

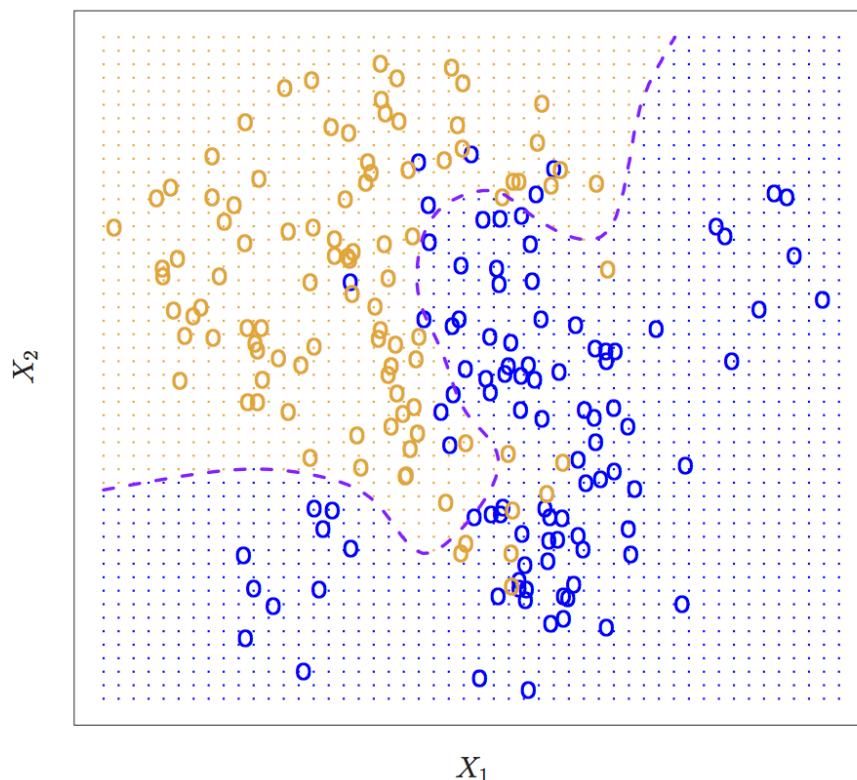
$$E(y_j - \hat{f}(x_j))^2 = Var(\hat{f}(x_j)) + [Bias(\hat{f}(x_j))]^2 + Var(\epsilon)$$

Term	Meaning
$E(y_j - \hat{f}(x_j))^2$	expected test MSE (j = 1..m are test items)
$Var(\hat{f}(x_j))$	amount estimate would change with different training data
$[Bias(\hat{f}(x_j))]^2$	error introduced in approximation
$Var(\epsilon)$	irreducible error

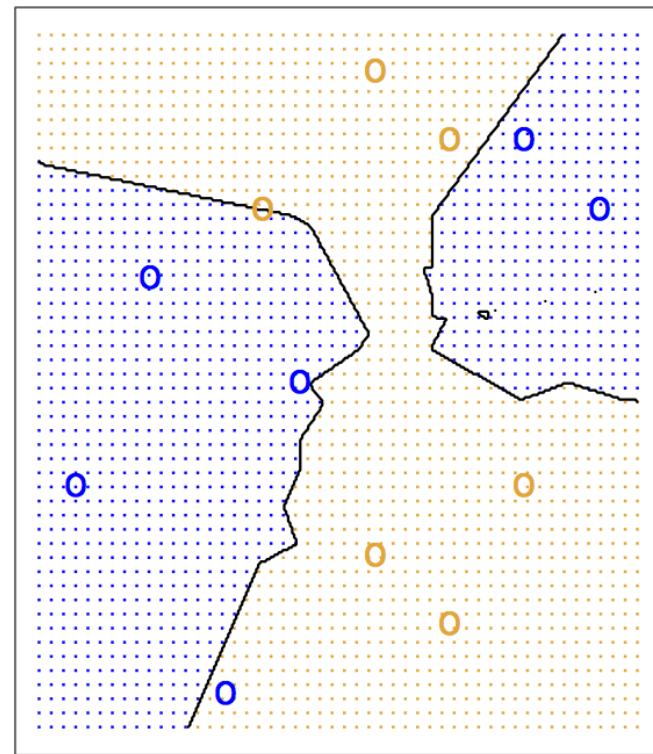
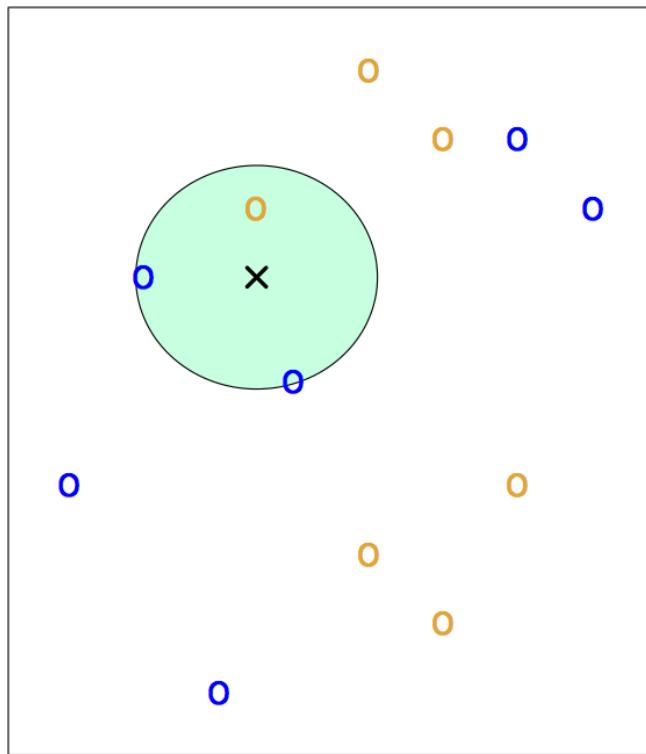
MSE, Bias, and Variance Relationships



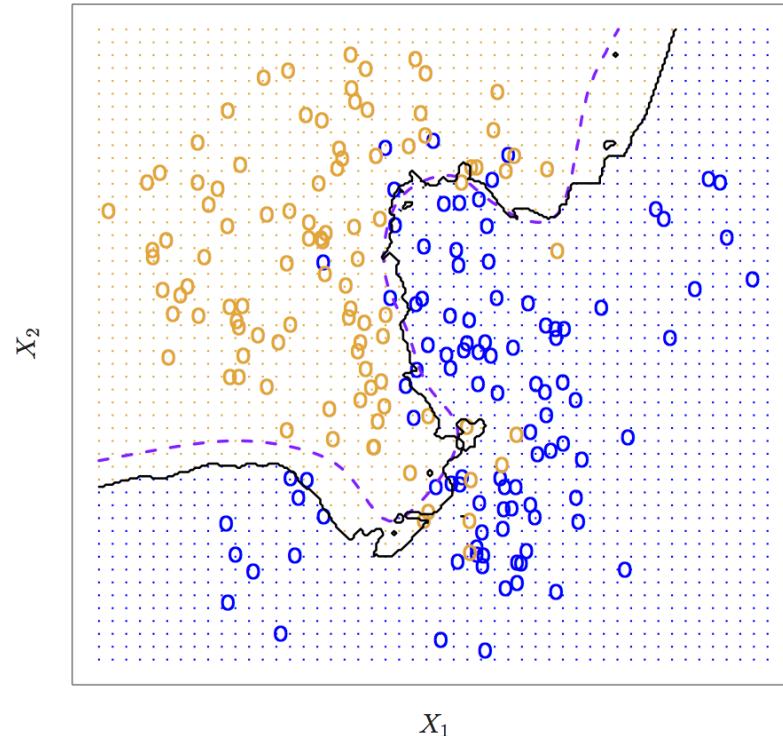
Comparing Classification Models



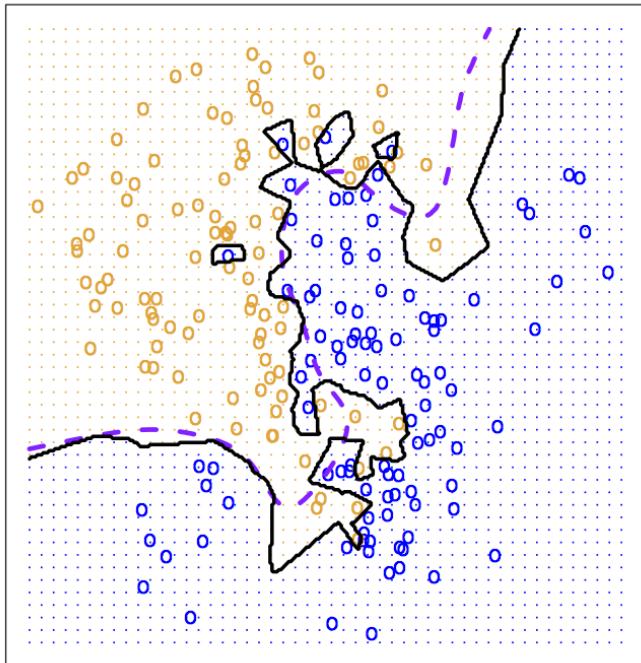
K-Nearest Neighbors ($k = 3$)



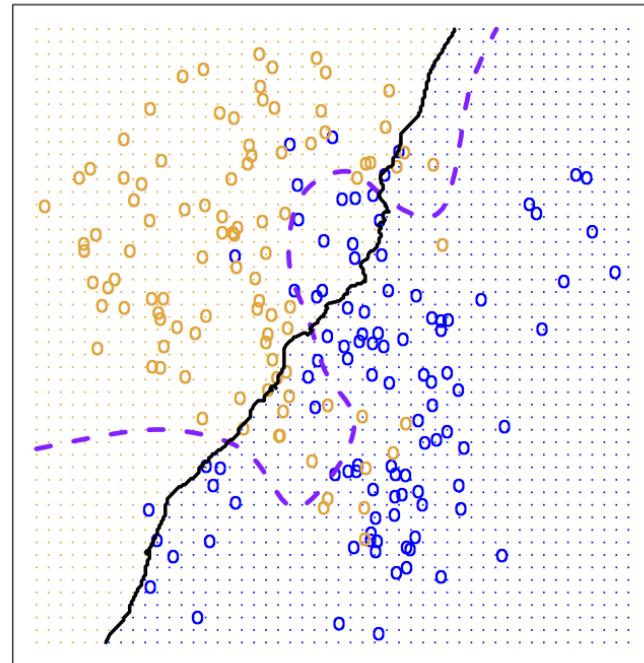
KNN: K=10

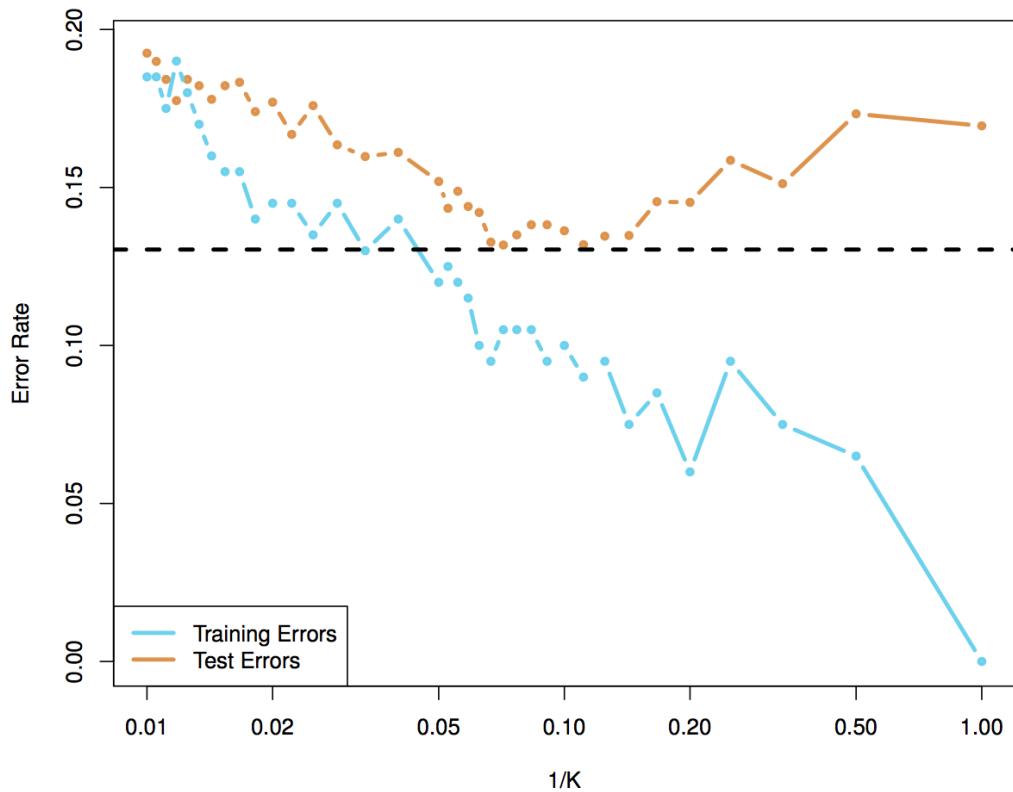


KNN: K=1



KNN: K=100



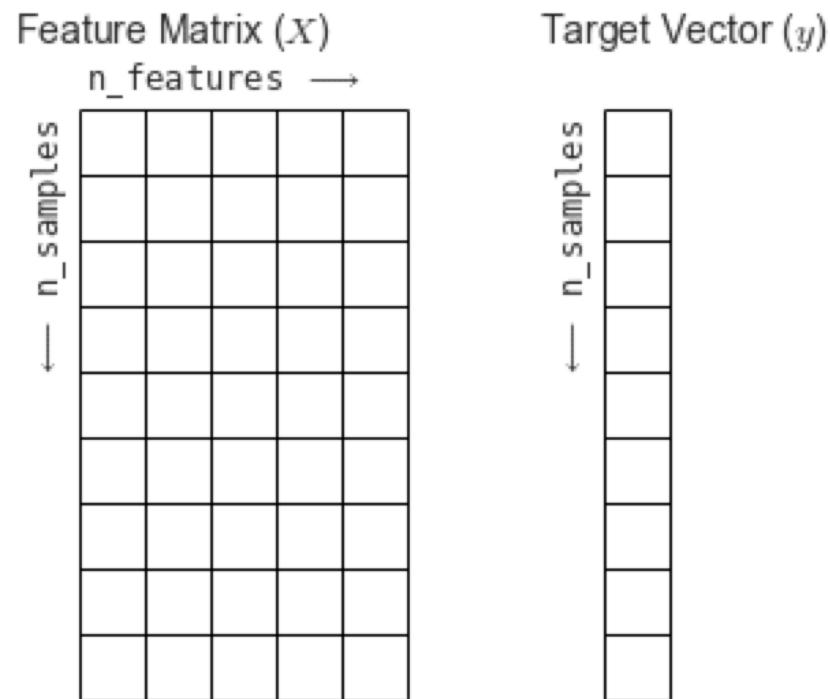


Demo: SciKit-Learn

SciKit Learn

- popular Python library providing efficient implementation of a large number of machine learning algorithms
- purposely designed to be clean and uniform across tools
- consistent data representation and common interface
- there are screenshots in this presentation to maintain continuity, but let's open the notebook named **Demo - SciKit Learn.ipynb** and go through it together
- when done, click [here](#) to skip screenshots

SciKit-Learn Data Representation



```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets

np.random.seed(5)

iris = datasets.load_iris()
dat = pd.DataFrame(data=np.c_[iris['data'], iris['target']],
                     columns=iris['feature_names'] + ['target'])

```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0
5	5.4	3.9	1.7	0.4	0.0
6	4.6	3.4	1.4	0.3	0.0
7	5.0	3.4	1.5	0.2	0.0
8	4.4	2.9	1.4	0.2	0.0
9	4.9	3.1	1.5	0.1	0.0
10	5.4	3.7	1.5	0.2	0.0
11	4.8	3.4	1.6	0.2	0.0

```
# Set up the feature matrix  
  
">>>> X_iris = dat.drop('target', axis=1)  
>>> X_iris.shape  
(150, 4)
```

```
# Set up the target vector  
  
">>>> y_iris = iris['target']  
>>> y_iris.shape  
(150,)
```

scikit-learn Objects

"All objects within scikit-learn share a uniform common basic API consisting of three complementary interfaces: an estimator interface for building and fitting models, a predictor interface for making predictions and a transformer interface for converting data."

Estimator API

- Driven by a set of principles documented in a paper:
<https://arxiv.org/pdf/1309.0238.pdf>
 - Consistency
 - Allow Inspection
 - Limited object hierarchies
 - Composition
 - Sensible defaults

General Flow

- Choose a model
- Choose model hyperparameters
- Arrange data into a features matrix and target vector
- Fit the model to the data with the fit() method
- Apply the model to test data (predict() or transform())

Examples

```
rng = np.random.RandomState(1)
x = 10 * rng.rand(50)
y = 2 * x - 5 + rng.randn(50)
plt.scatter(x, y);

from sklearn.linear_model import LinearRegression
model = LinearRegression(fit_intercept=True)
model.fit(x[:, np.newaxis], y)
xfit = np.linspace(0, 10, 1000)
yfit = model.predict(xfit[:, np.newaxis])
```

```
from sklearn.tree import DecisionTreeClassifier
tree_clf = DecisionTreeClassifier(max_depth=2)
tree_clf.fit(X_iris, y_iris)
tree_clf.predict([[5, 1.5, 2, 2.0]])
```

```
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
X, y = make_blobs(random_state=1)
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
kmeans.labels_
```

Exercise: SciKit-Learn