Statistics and Hypothesis Testing

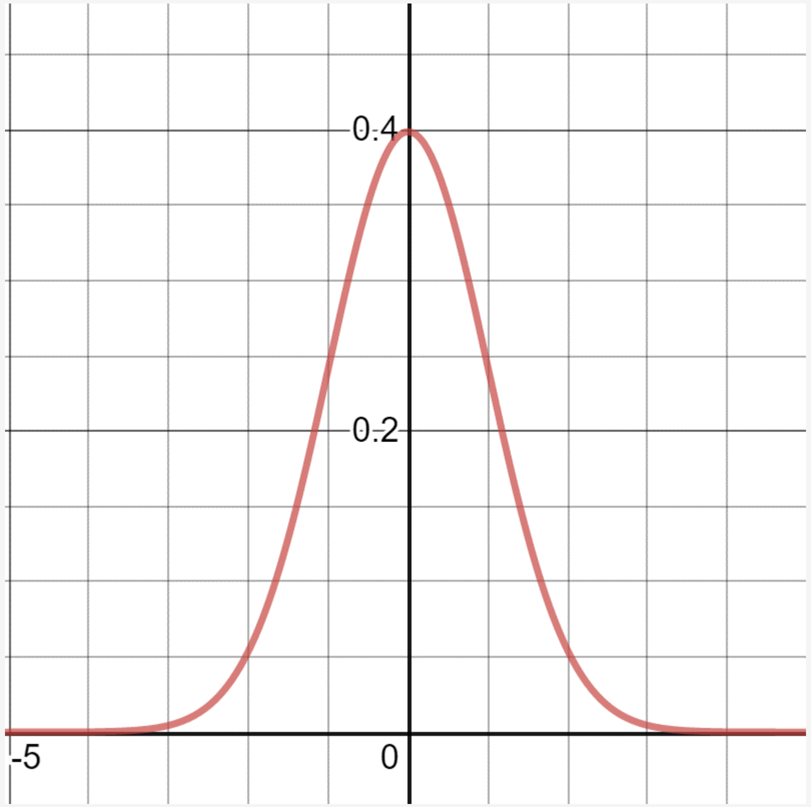
With Python

Why Learn Statistics and Hypothesis Testing?

Data is usually not useful until you can describe it with few numbers, which is one purpose of statistics.

When we want to describe larger populations (how people will vote, average car life in the US, etc) we often rely on samples to infer attributes about the larger population.

Machine learning, engineering, data science, and many other technical disciplines are driven by data and statistics.



Descriptive vs Inferential Statistics

Note there are two areas of statistics:

Descriptive Statistics – Uses tools to summarize and describe data by reducing it into single numbers such as mean, median, variance and standard deviation.

EXAMPLE*: 70% of the Panthers high school basketball team are over 6 feet tall.*

Inferential Statistics – Uses a sample of data to make predictions about a larger population, typically using hypothesis testing.

EXAMPLE*: Based on a phone survey, 62% of the U.S. population likes dogs more than cats.*

Σ

Descriptive Statistics

Populations, Samples, and Parameters

In statistics, we are often trying to infer things about a population based on a sample.

Population – A particular group of interest such as "***all*** seniors over the age of 65 in the United States" or "***all*** golden retrievers in Scotland."

Parameter – Numerical description of a population characteristic (e.g. average age of college graduates in the United States).

Sample – A subset of the population that is ideally random and unbiased (e.g. 100 randomly sampled college graduates in the United States), so we can infer parameters about the population.

Confusing Populations with Samples

IMPORTANT NOTE:

How we define "population" must be done carefully, and populations can act more like samples from something abstract.

Be mindful of this and how it impacts your objective.

EXAMPLE: We are interested in flights that depart between 2PM and 3PM, but we lack enough flights at that time to reliably predict on-time performance.

In the above example, we might consider our flights to be samples from a theoretical population that has yet to be measured.

Mean

The mean is the average of a set of values, or the sum of values divided by the number of values.

For a sample where n is the number of sample values, we define mean $\bar{x}$ as:

$$\bar{x} = \frac{x_1+x_2+x_3+\cdots+x_n}{n} = \frac{\sum x_i}{n}$$

For a population where N is the population count, we define mean μ as:

$$\mu = \frac{x_1+x_2+x_3+\cdots+x_n}{N} = \frac{\sum x_i}{N}$$

The mean is useful because it shows where the "center of gravity" exists at the center of the values.

```
# Number of pets each person owns
sample = [1, 3, 2, 5, 7, 0, 2, 3]

mean = sum(sample) / len(sample)

print(mean) # prints 2.875
```

Calculating the mean in Python for 7 randomly sampled individuals for the number of pets they own.

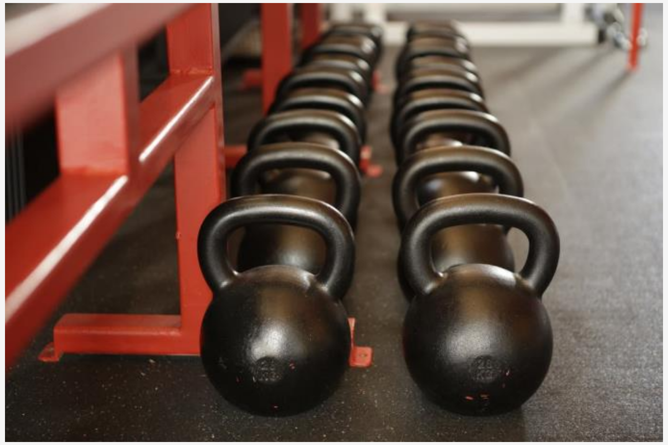
Weighted Mean

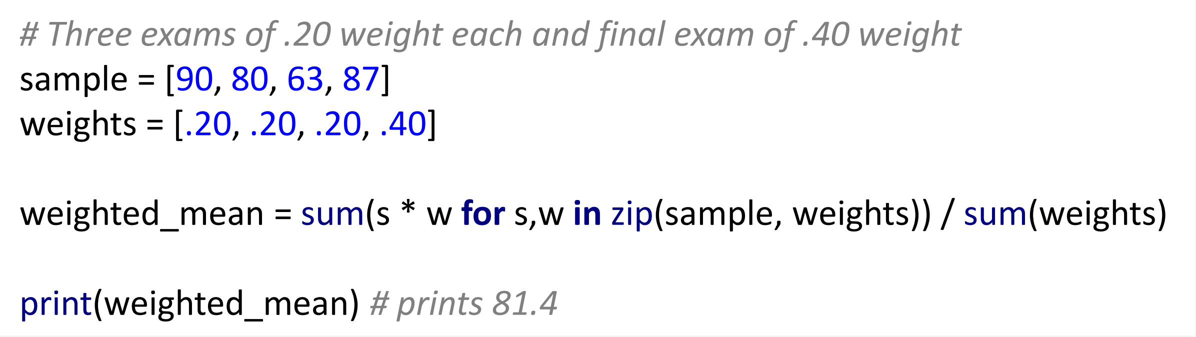
A weighted mean is a mean where each item is not given equal importance.

$$\frac{(x_1 \cdot w_1) + (x_2 \cdot w_1) + \cdots + (x_n \cdot w_n)}{w_1 + w_2 + \cdots + w_n} = \frac{\sum x_i \cdot w_i}{\sum w_i}$$

To the right we do a weighted mean of 3 exams of .20 weight and a final exam of .40 weight.

Note that the basic mean is also a weighted mean, it just gives equal importance to each item.





```python
# Three exams of .20 weight each and final exam of .40 weight
sample = [90, 80, 63, 87]
weights = [.20, .20, .20, .40]

weighted_mean = sum(s * w for s,w in zip(sample, weights)) / sum(weights)

print(weighted_mean) # prints 81.4
```

A weighted mean of 3 tests and 1 final test

Median

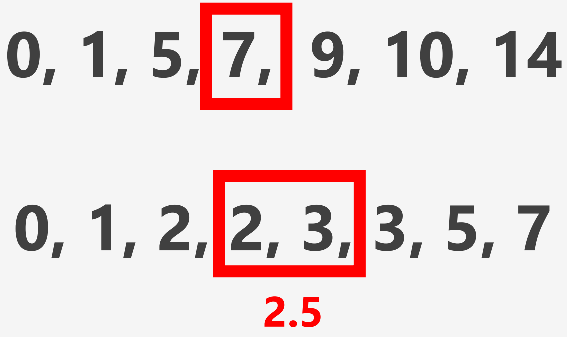
The median is the middle-most value in a set of ordered values.

It can be helpful in cases where data is skewed by **outliers**, or values that are extremely large and small compared to the rest of the values.

EXAMPLE: NBA basketball players tainting the average income of graduates from their college.

When there are an even number of values, average the two center-most values.

When the median is very different from the mean, this indicates your values are skewed.



0, 1, 5, 7, 9, 10, 14

0, 1, 2, 2, 3, 3, 5, 7

2.5

```
# Number of pets each person owns
sample = [1, 3, 2, 5, 7, 0, 2, 3]

def median(values):
   ordered = sorted(values)
   print(ordered)
   n = len(ordered)
   mid = int(n / 2) - 1

   if n % 2 == 0:
      return (ordered[mid] + ordered[mid+1]) / 2.0
   else:
      return ordered[mid]

print(median(sample)) # prints 2.5
```

Mode

The mode is the most frequently occurring value in a set of values.

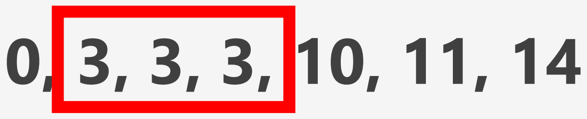
The mode primarily becomes useful when values are repetitive, and you want to find which occur the most frequently.

When no value occurs more than once, then there is no mode.

When two values occur with the same frequency, then it is bimodal.

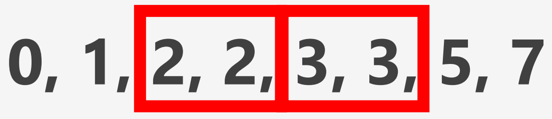
Outside of repetitive data sets, you will likely not use mode very much.

```python
# Number of pets each person owns
from collections import defaultdict

sample = [1, 3, 2, 5, 7, 0, 2, 3]

def mode(values):
    counts = defaultdict(lambda: 0)

    for s in values:
        counts[s] += 1

    max_count = max(counts.values())
    modes = [v for v in set(values) if counts[v] == max_count]
    return modes

print(mode(sample))
```



0, 3, 3, 3, 10, 11, 14

A mode of "3"



0, 1, 2, 2, 3, 3, 5, 7

Bimodal set with modes "2" and "3"

Variance

In describing data, we are often interested in measuring the differences between the mean μ and each data observation x_i

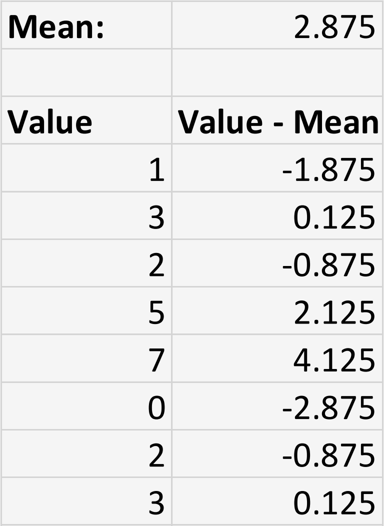
$$x_i - \mu$$

To get variance $\boldsymbol{\sigma^2}$ we sum these squared differences and divide by the number of items N.

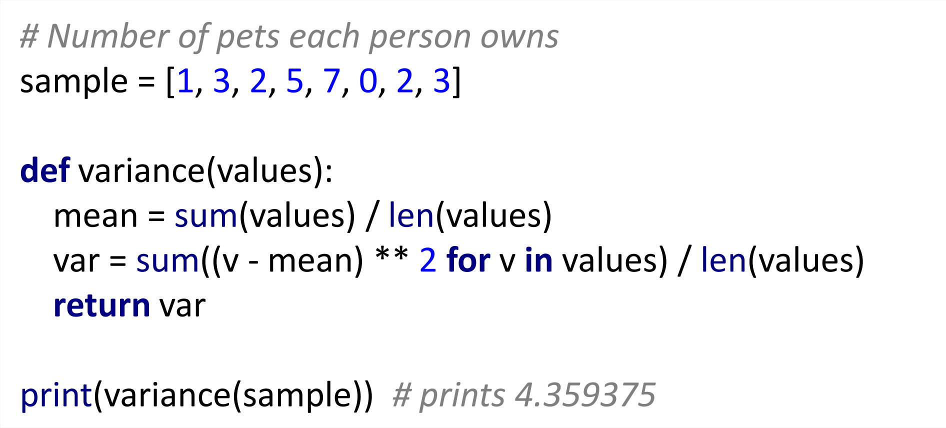
$$\sigma^2 = \frac{\sum (x_i - \mu)^2}{N}$$

In other words, we average the differences between the mean and each value.

This helps us measure how "spread out" our data is.



Mean:	2.875
Value	**Value - Mean**
1	-1.875
3	0.125
2	-0.875
5	2.125
7	4.125
0	-2.875
2	-0.875
3	0.125



```
# Number of pets each person owns
sample = [1, 3, 2, 5, 7, 0, 2, 3]

def variance(values):
    mean = sum(values) / len(values)
    var = sum((v - mean) ** 2 for v in values) / len(values)
    return var

print(variance(sample))  # prints 4.359375
```

Sample Variance versus Population Variance

One thing that needs to be called out: there is a minor adjustment in the variance formula for a population versus a sample.

SAMPLE

$$S^2 = \frac{\sum (x_i - \bar{x})^2}{n-1}$$

POPULATION

$$\sigma^2 = \frac{\sum (x_i - \mu)^2}{N}$$

We define the sample average as $\bar{x}$ and the population average as μ, and the number of items is defined by n or N respectively.

Why do we subtract “1” from n in the sample? This is to decrease the bias in the sample and not underestimate the variance of the population based on the sample.

Learn more: https://youtu.be/sHRBg6BhKjI

Standard Deviation

To make variance easier to interpret, we often want to square root it to give us the standard deviation.

SAMPLE

$$s = \sqrt{\frac{\sum(x_i - \bar{x})^2}{n - 1}}$$

POPULATION

$$\sigma = \sqrt{\frac{\sum(x_i - \mu)^2}{N}}$$

This effectively undoes the squares and back into units we can meaningfully interpret.

For example, the standard deviation for the number of pets each person owns (based on our sample) is 2.0879.

```python
from math import sqrt

# Number of pets each person owns
sample = [1, 3, 2, 5, 7, 0, 2, 3]

def variance(values):
    mean = sum(values) / len(values)
    var = sum((v - mean) ** 2 for v in values) / len(values)
    return var

def std_dev(values):
    return sqrt(variance(values))

print(std_dev(sample))  # prints 2.0879116360612584
```

Coefficient of Variation

Let’s say you have samples of home values from two neighborhoods A and B.

Neighborhood A has a sample mean of $140,000 and standard deviation of $3000.

Neighborhood B has a sample mean of $800,000 and a standard deviation of $10,000.

Which of these has more spread?

Coefficient of Variation

While you may be tempted to say neighborhood B has more spread, first divide the mean by the standard deviation for each to get the coefficient of variation.

$$cv = \frac{\sigma}{\bar{x}}$$

$$\boxed{cv_A = \frac{3{,}000}{140{,}000} = .0214}$$

$$cv_B = \frac{10{,}000}{800{,}000} = .0125$$

Surprisingly the cheaper neighborhood has higher variation despite smaller numbers, and the CV is helpful for capturing dispersion despite different numbers.

Exercise 1

You bought a spool of 1.75 mm filament for your 3D printer.

You want to measure how close the filament diameter really is to 1.75 mm.

You use a tool and sample the diameter five times on the spool:

$$\mathbf{1.78, 1.75, 1.72, 1.74, 1.77}$$

Calculate the mean and standard deviation for this set of values.

Exercise 1: ANSWER

Sample Mean

$$\bar{x} = \frac{x_1 + x_2 + x_3 + \cdots + x_n}{n} = \frac{\sum x_i}{n}$$

$$\bar{x} = \frac{1.78 + 1.75 + 1.72 + 1.74 + 1.77}{5}$$

$$\boxed{\bar{x} = 1.752}$$

MEAN

Sample Standard Deviation

$$s = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n - 1}}$$

$$s = \sqrt{\frac{(1.78 - 1.752)^2 + (1.75 - 1.752)^2 + (1.72 - 1.752)^2 + (1.74 - 1.752)^2 + (1.78 - 1.752)^2}{5 - 1}}$$

$$\boxed{s = 0.023875}$$

STANDARD DEVIATION

Exercise 1: ANSWER

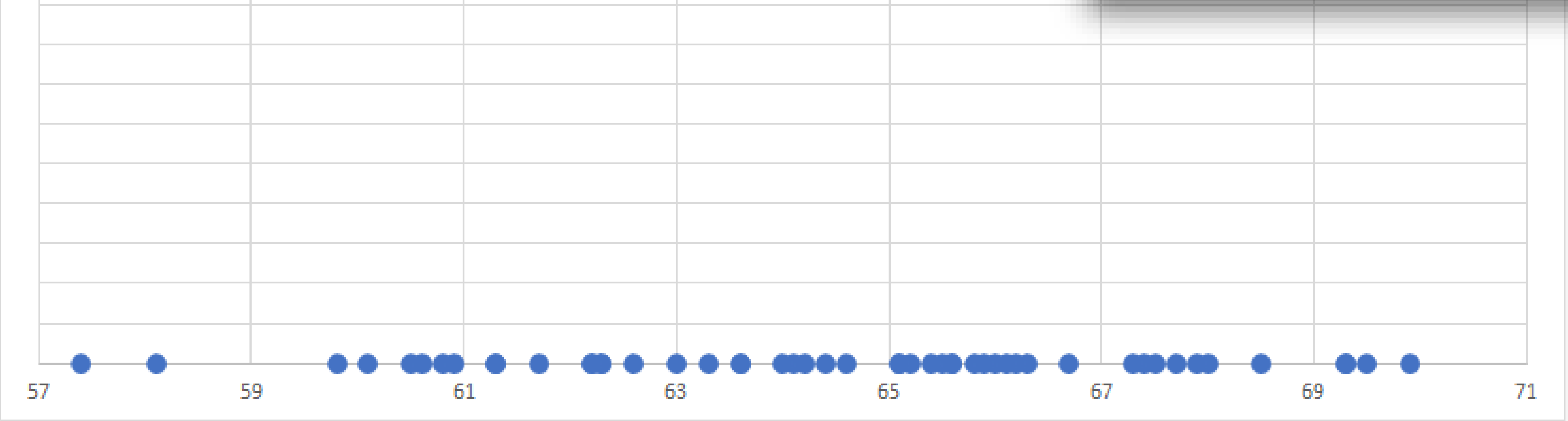
```
from math import sqrt

# Filament samples
sample = [1.78, 1.75, 1.72, 1.74, 1.77]

def mean(values):
    return sum(sample) / len(sample)

def variance_sample(values):
    mean = sum(values) / len(values)
    var = sum((v - mean) ** 2 for v in values) / (len(values) - 1.0)
    return var

def std_dev_sample(values):
    return sqrt(variance_sample(values))


print("MEAN = {}".format(mean(sample))) # 1.752
print("STD DEV = {}".format(std_dev_sample(sample))) # 0.023874672772626667
```

Normal Distribution

Discovering the Normal Distribution

Below is a number line showing recorded weights (in pounds) for 50 adult golden retrievers.

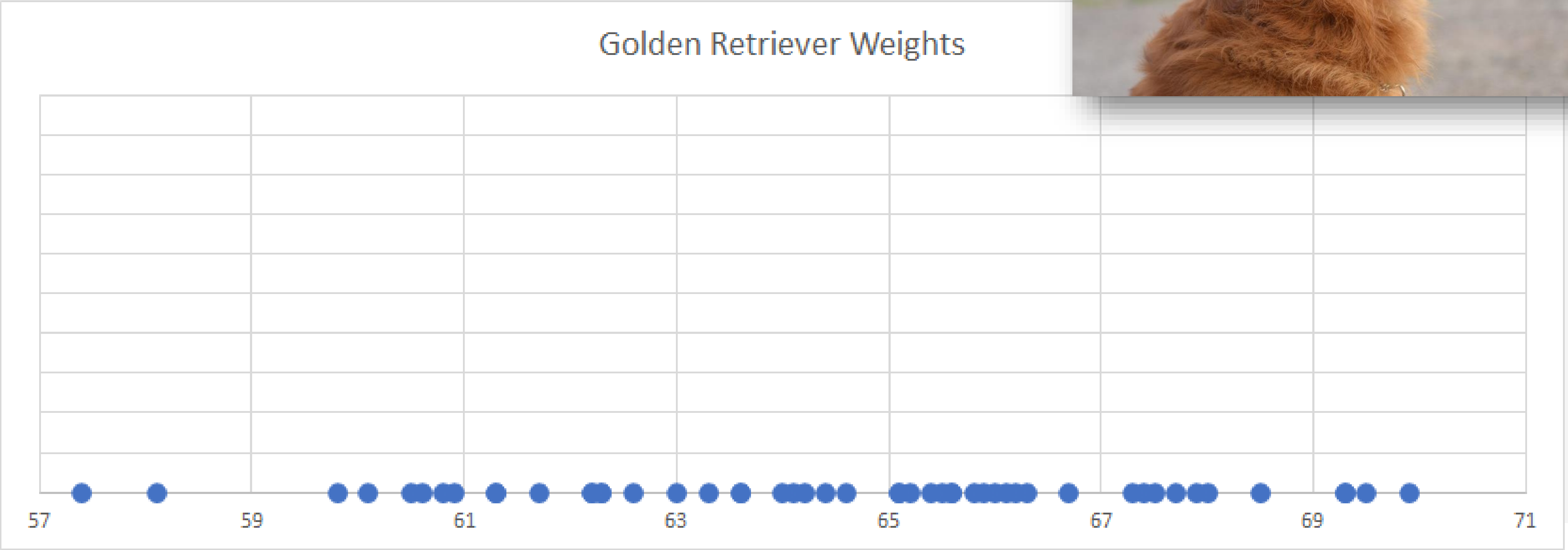
What do you notice about this data?



Discovering the Normal Distribution

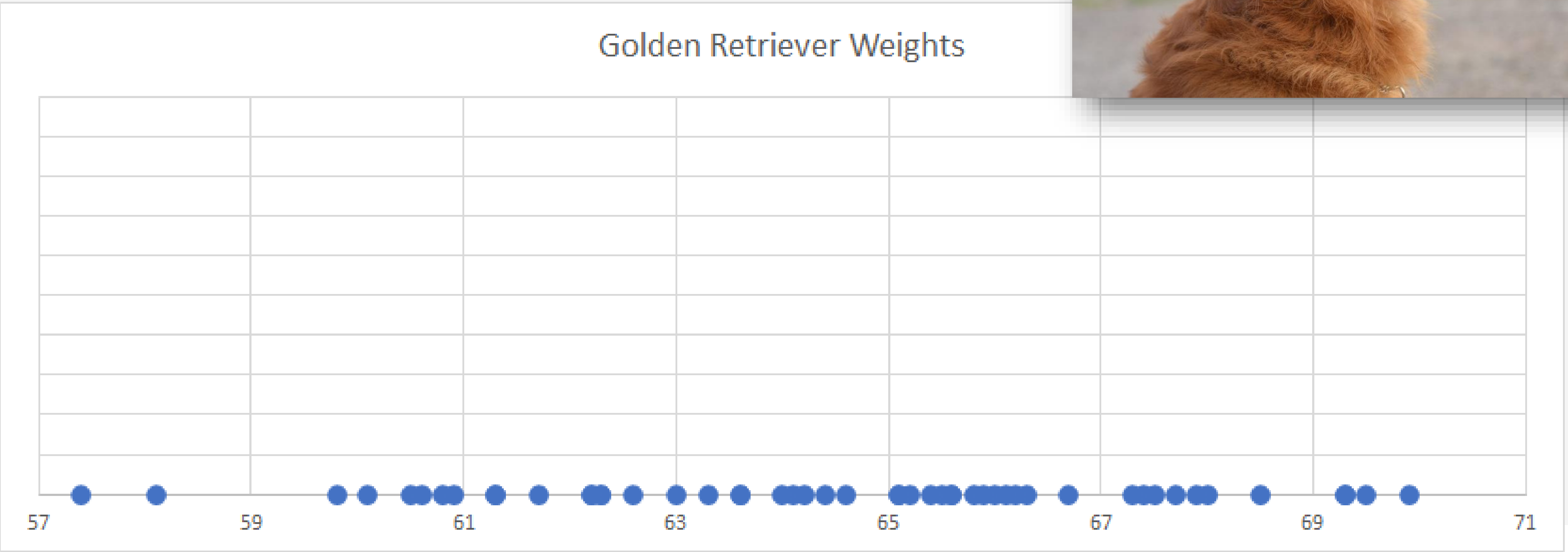
Notice that towards the "center" there are more points clustered together, while the "tails" on either end have less points.

We can infer that we are more likely to see more golden retrievers in the 61-67 lb range as opposed to lower or higher weights like 57 and 71.



Discovering the Normal Distribution

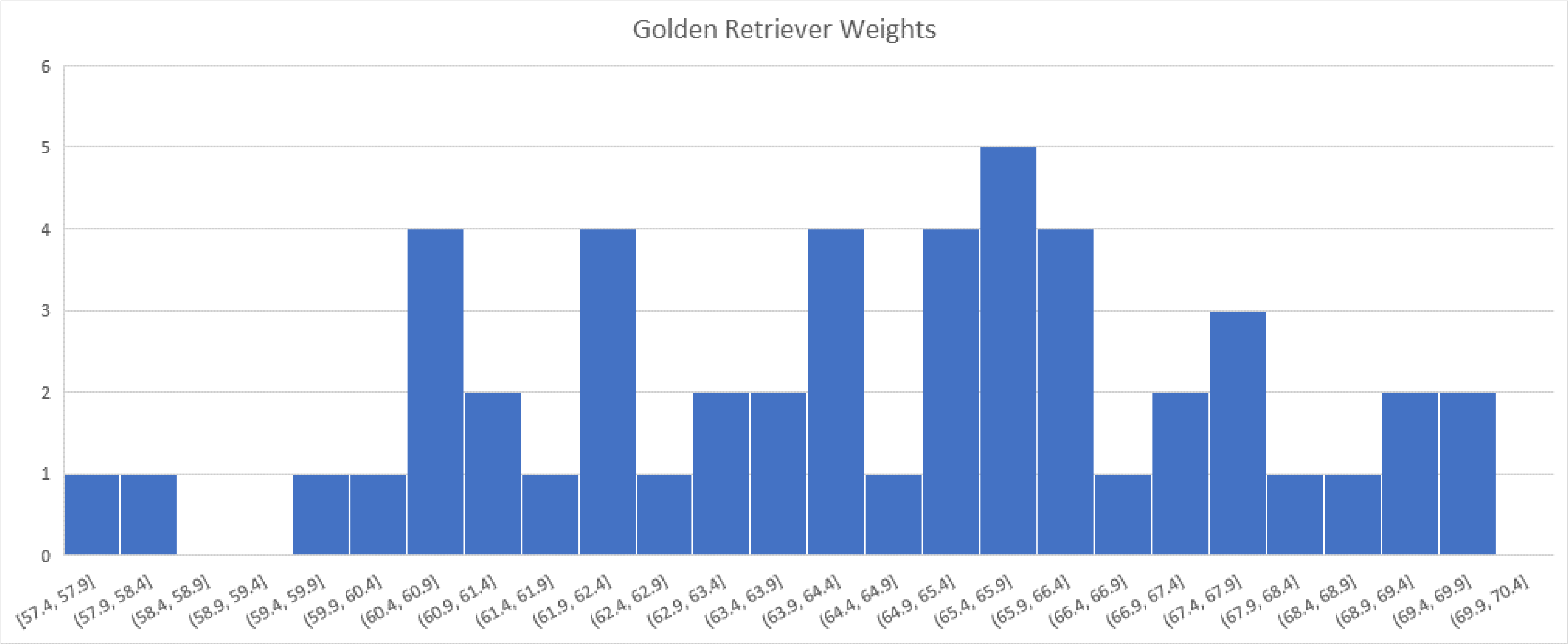
Is there a better way we can visualize this data? And get an idea where we would see more golden retrievers in terms of their weight?



Discovering the Normal Distribution

We can try to **bin** up points on equally-sized ranges, and then count the number of instances as a bar chart to create a **histogram**.

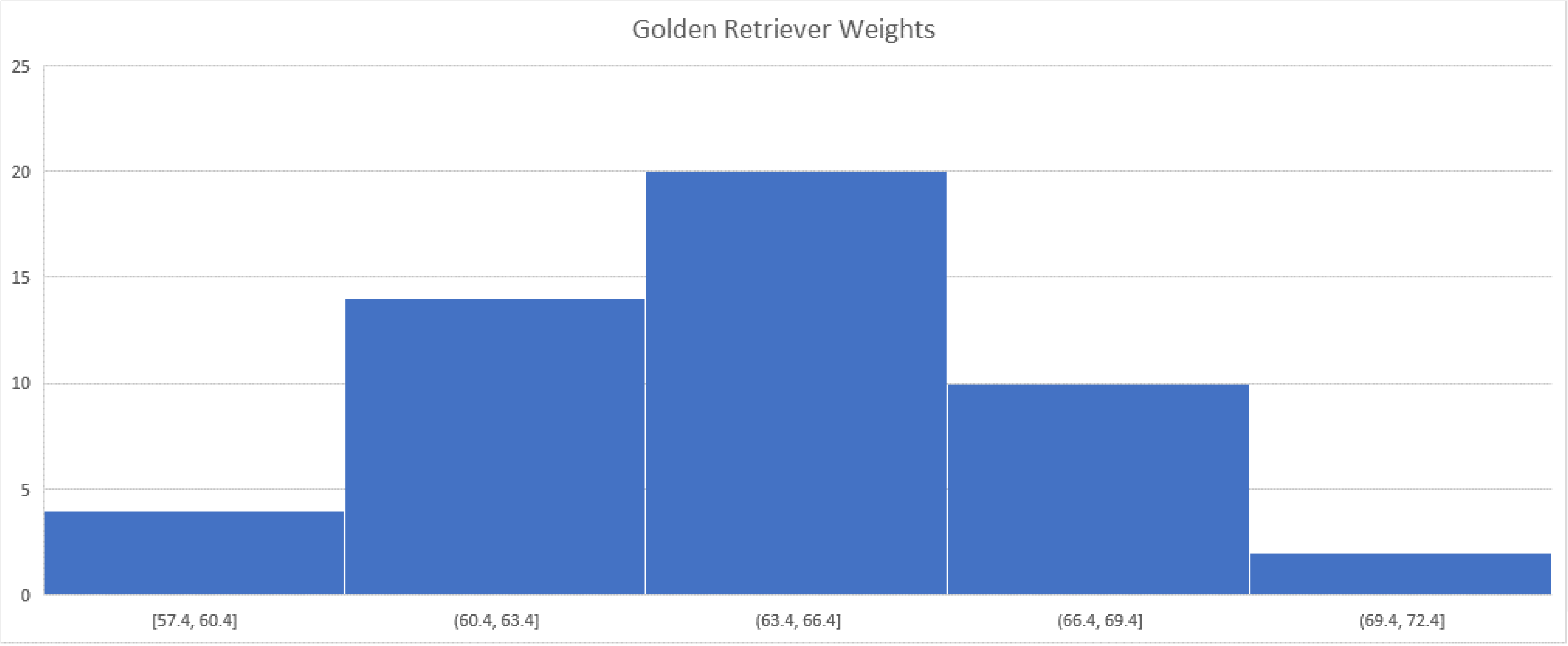
Unfortunately, we don’t have an infinite amount of data so making the bin ranges too small will not reveal much about the underlying distribution.



Discovering the Normal Distribution

But if we make the bin sizes just right, we might be able to see a shape resembling a probability distribution.

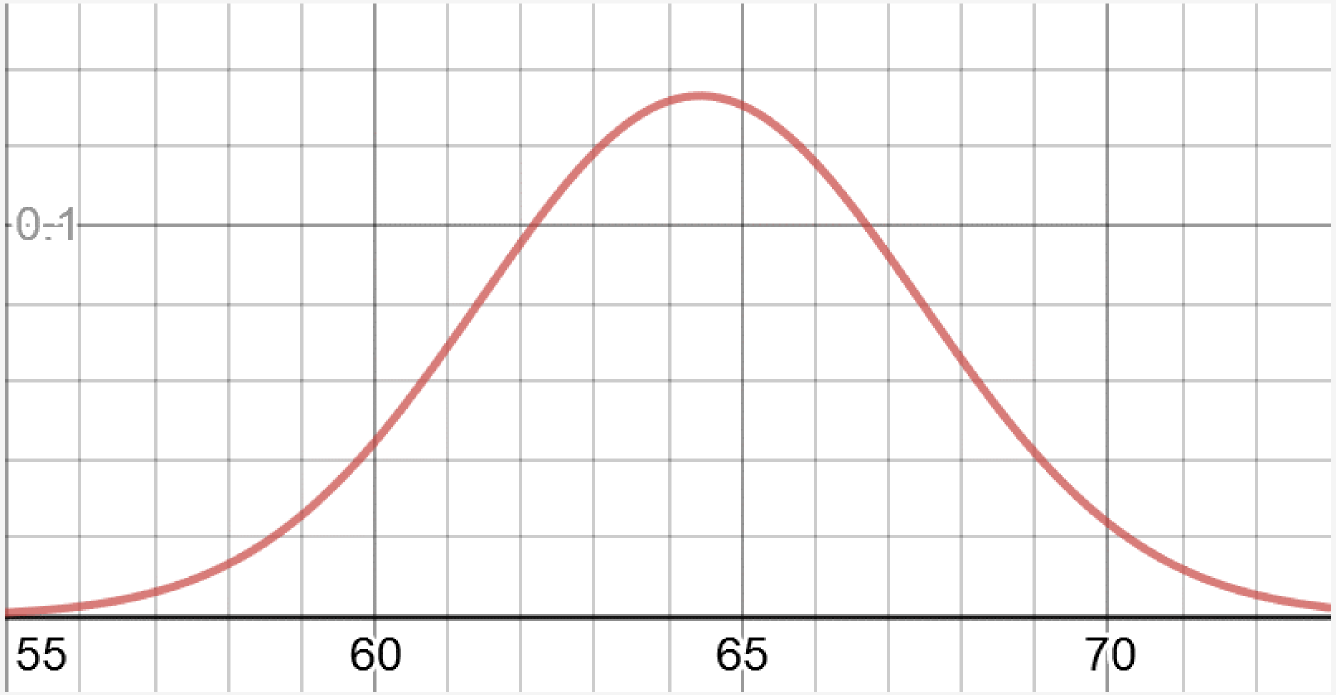
There are many types of distributions, but in this case we can see a nice bell curve known as the **normal distribution**, also called the **Gaussian Distribution**.



Discovering the Normal Distribution

We can fit this curve onto a histogram to make inferences about the entire population, as it’s unlikely I will be able to weigh every golden retriever in existence.

We can then scale this curve, so it has an area of 1.0, therefore representing a mass of probability.



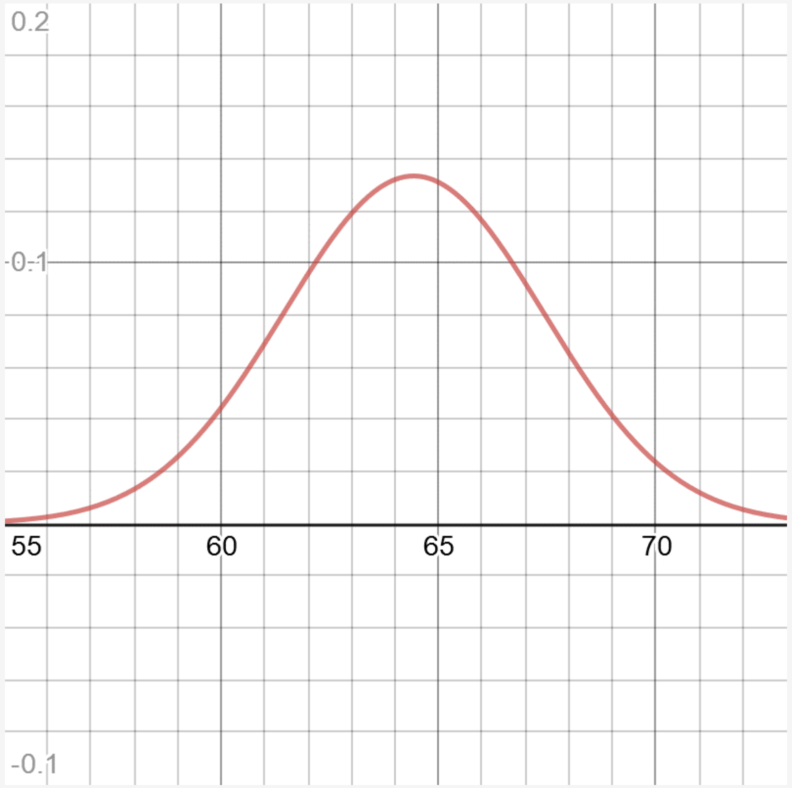
https://www.desmos.com/calculator/yftpag0tse

Discovering the Normal Distribution

The normal distribution has several important properties that make it useful:

- Symmetrical
- Most mass is at the center around the **mean**
- Has a spread (being narrow or wide) that is specified by **standard deviation**.
- The “tails” are the least likely outcomes, and approach zero infinitely but never touch zero.
- It resembles a lot of phenomena in nature and daily life, and even generalizes non-normal problems because of the central limit theorem.

We can expect any golden retriever to have a weight most likely around 64.43 (the mean), but highly unlikely around 55 or 73.



https://www.desmos.com/calculator/yftpag0tse

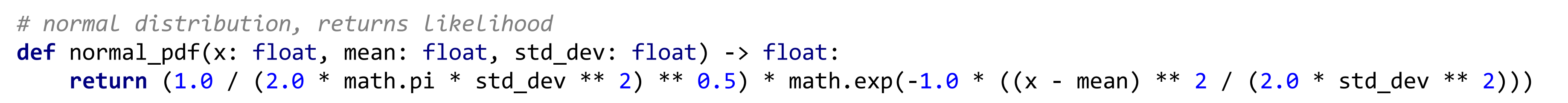
The Normal Distribution Formula

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

$$\mu = mean$$
$$\sigma = standard\ deviation$$
$$x = observed\ value$$

$$f(x) = probability\ density\ function$$

Python function:



```python
# normal distribution, returns likelihood
def normal_pdf(x: float, mean: float, std_dev: float) -> float:
    return (1.0 / (2.0 * math.pi * std_dev ** 2) ** 0.5) * math.exp(-1.0 * ((x - mean) ** 2 / (2.0 * std_dev ** 2)))
```

Discovering the Normal Distribution

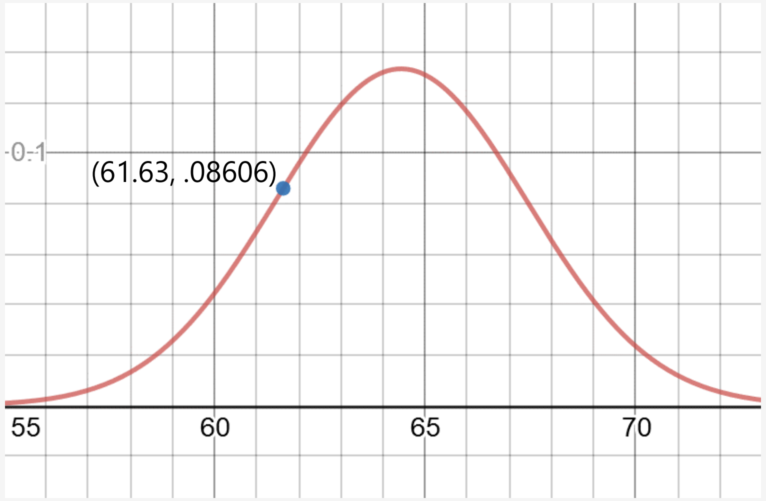
Trick question: What is the probability any golden retriever chosen at random will have a weight of exactly 61.63 lbs?

The answer may surprise you: It is 0%!

This is one of the great paradoxes of continuous distributions like the normal distribution and the beta distribution, as decimals can become so infinitely small there is an infinite number of possibilities on the curve.

The probability of getting a specific value is virtually impossible ***<u>unless</u>*** it is already observed.

The y-axis represents probability density, and to find a probability you need to use the area under a curve for a range of values.



Calculating Probabilities

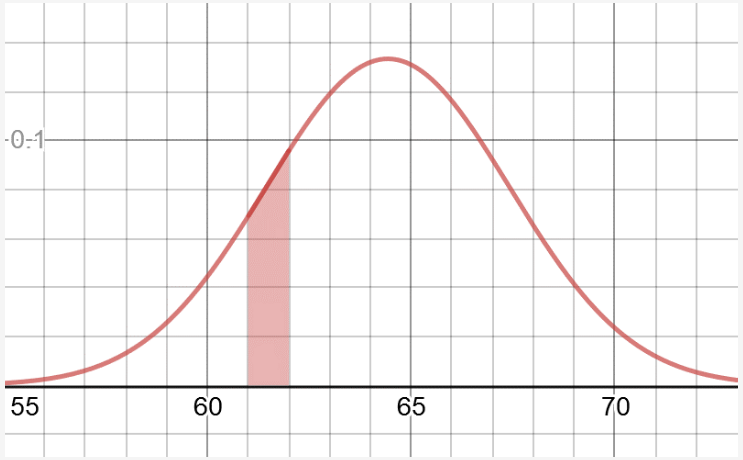
Rather than ask:

“What is the probability a golden retriever will be exactly 61.63 pounds?”

A more productive question would be:

“What is the probability a golden retriever will be ***between*** 61 and 62 pounds?”

The answer is 8.25%, and we will learn how to calculate this using integration just like we did with the beta distribution.



Empirical Rule of Standard Deviation

One of the nice features of the normal distribution is the **68-95-99.5 rule**, or **empirical rule of standard deviation**, which states the area/probability within 1, 2, and 3 standard deviations from the mean respectively.

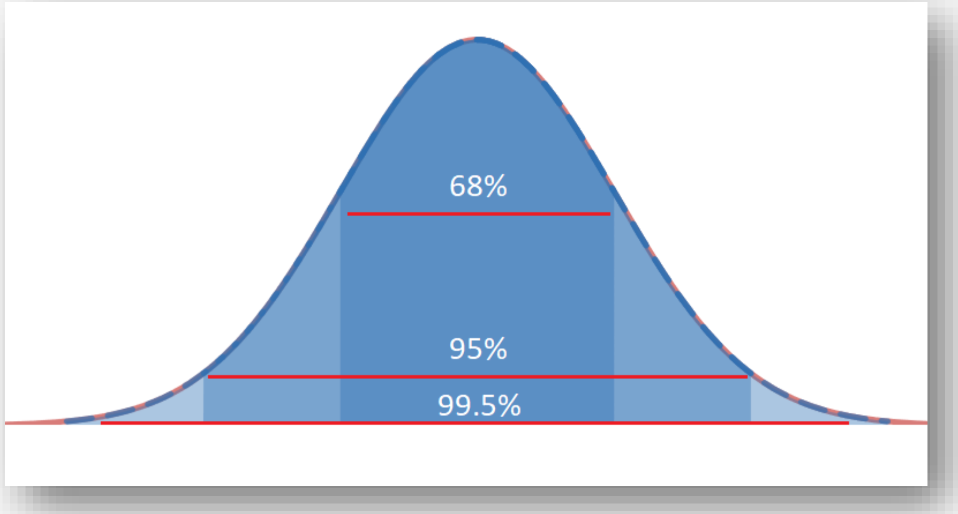
This *APPROXIMATELY* means that if I have a mean of 10 and standard deviation of 3, then…

68% of the probability will be between 7 and 13

95% will be between 4 and 16

99.5% will be between 1 and 19

You can use this rule to quickly calculate probabilities when using standard deviations.



Standard Normal Distribution and Z-Scores

A special type of normal distribution is the standard normal distribution, which has a mean of 0 and standard deviation of 1.

Sometimes we will convert a normal distribution into a standard normal distribution.

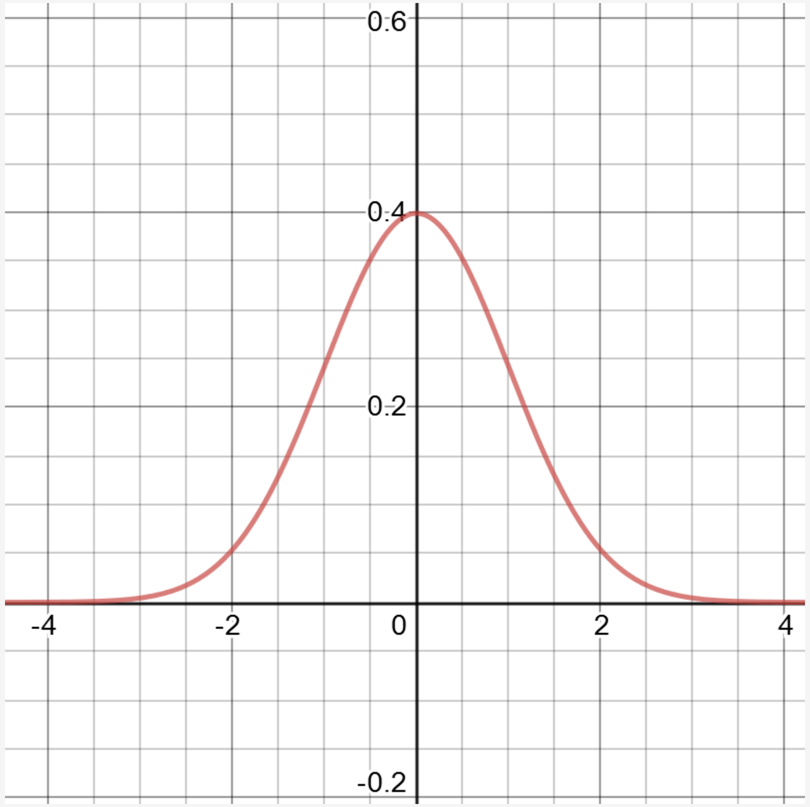
This creates a convenient way to express all values in terms of the standard deviations, known as **z scores**.

You can express any value in standard deviations (a z-score) using this conversion formula:

$$z_{population} = \frac{x-\mu}{\sigma} \qquad z_{sample} = \frac{x-\bar{x}}{s}$$

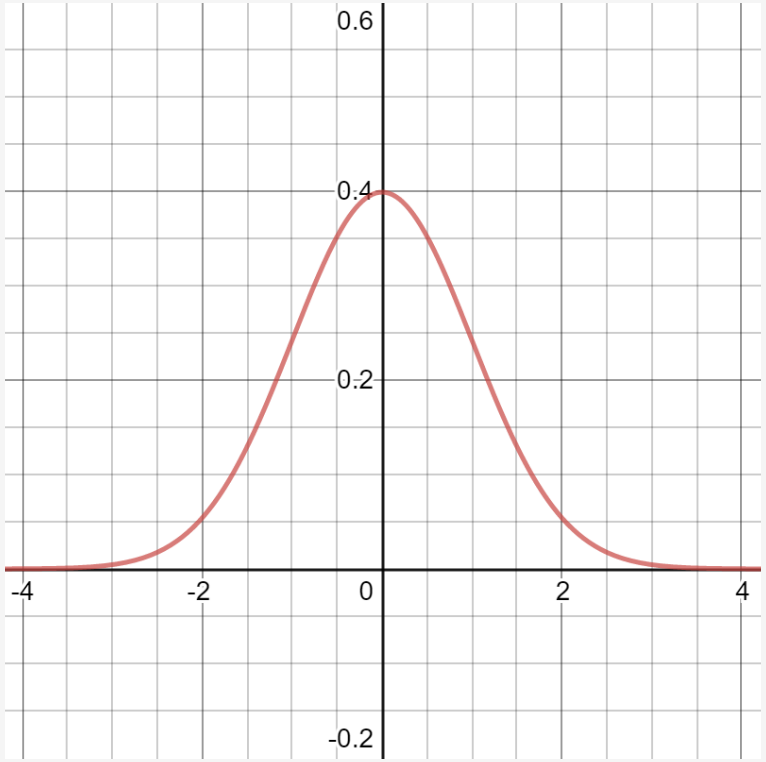
We can convert a Z-score back to the original value using the reverse formula:

$$x = z \times \sigma + \mu$$



Standard Normal Distribution and Z-Scores

```python
def z_score(x, mean, std):
   return (x - mean) / std


def z_to_x(z, mean, std):
   return (z * std) + mean


mean = 6.0
std_dev = 1.0
x = 8.0

# Convert to Z-score and then back to X
z = z_score(x, mean, std_dev)
back_to_x = z_to_x(z, mean, std_dev)

print("Z-Score: {}".format(z))  # Z-Score: 2.0
print("Back to X: {}".format(back_to_x))  # Back to X: 8.0
```



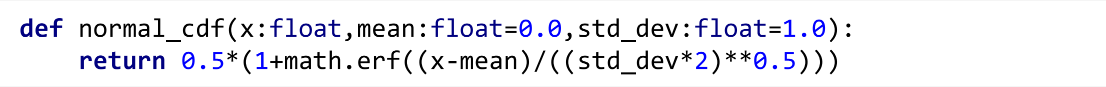
Cumulative Density Function (CDF)

To find the probability of a range of values occurring, we use a **cumulative density function**, which is a function that returns the area up to a value “x”.

$$\frac{1}{2}\left[1+\operatorname{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right]$$

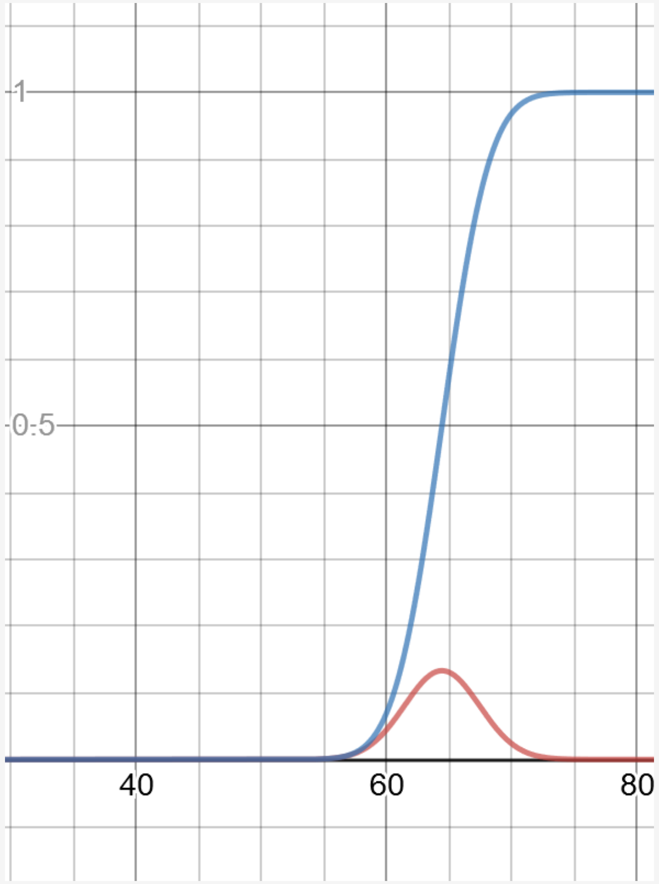
The normal distribution’s probability density function (PDF) is shown alongside its cumulative density function (CDF), and the CDF is projecting the area captured up to that x-value on the PDF.

Here is the CDF from scratch expressed in Python:



```python
def normal_cdf(x:float,mean:float=0.0,std_dev:float=1.0):
    return 0.5*(1+math.erf((x-mean)/((std_dev*2)**0.5)))
```

We do cheat a little here and use the `erf` function, and we will leave it a black box as it is beyond the scope of this class.

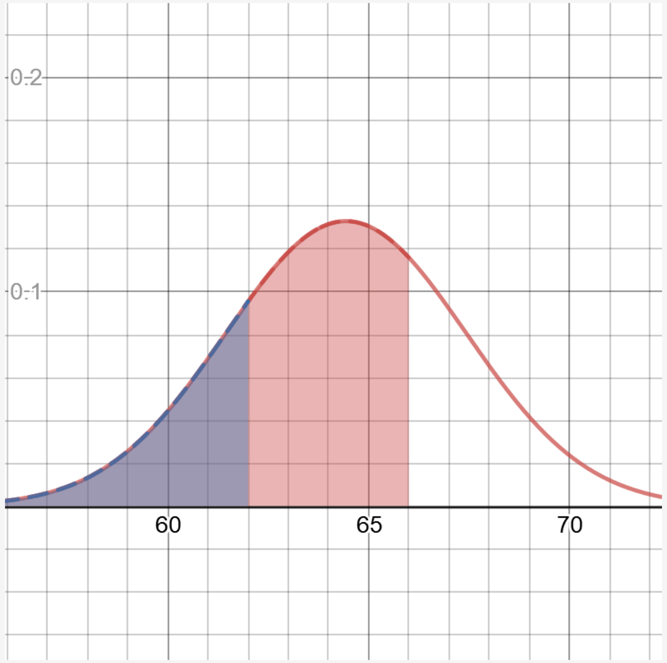


Cumulative Density Function (CDF)

I can use the CDF to find the probability of a golden retriever having a weight between 62 and 66 pounds.

I first find the area up to 66 (highlighted in red to the right) and then subtract the area up to 62 (highlighted in blue), and we should get 49.2%.

```
from scipy.stats import norm

mean = 64.43
std_dev = 2.99

x = norm.cdf(66, mean, std_dev) - norm.cdf(62, mean, std_dev)

print(x)
```



Quantile Function

If you want to look up a given “x” value that produces a given area, you will likely want to use a **quantile function** which is the inverse of the CDF function.

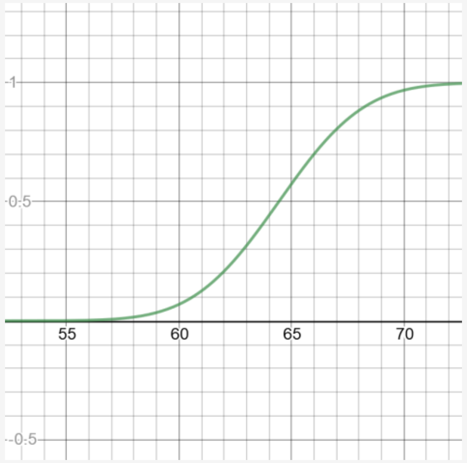
With the quantile function, you can input a probability between 0.0 and 1.0 and it will return the respective “x” value.

```python
from scipy.special import erfinv

def inv_normal_cdf(p: float, mean: float, std_dev: float):
    return mean + (std_dev * (2.0 ** 0.5) * erfinv((2.0 * p) - 1.0))
```

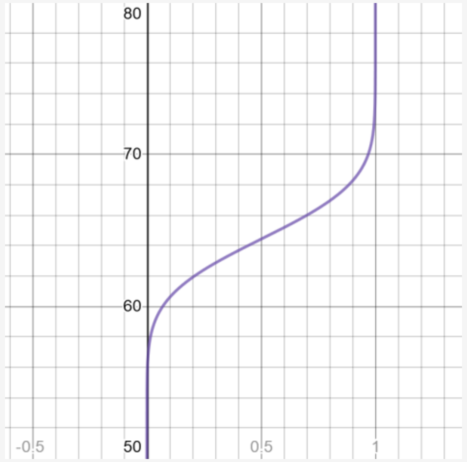
Think of it as an inverted lookup, where we look up the likelihood value on the y-axis, and then return the corresponding value from the x-axis.

You can also use Scikit-Learn (note that `loc` is mean and `scale` is standard deviation)

```python
from scipy.stats import norm

print(norm.ppf(.95, loc=0, scale=1)) # 1.6448536269514722
```



Normal CDF

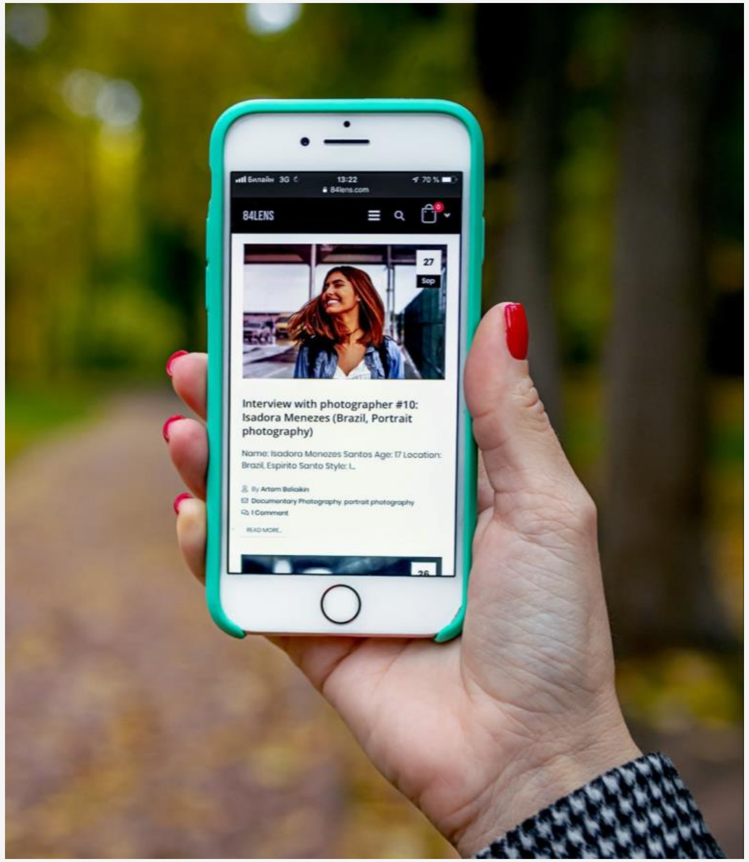


Normal Quantile

Exercise 2

A manufacturer says the Z-Phone smart phone has a mean consumer life of 42 months with a standard deviation of 8 months.

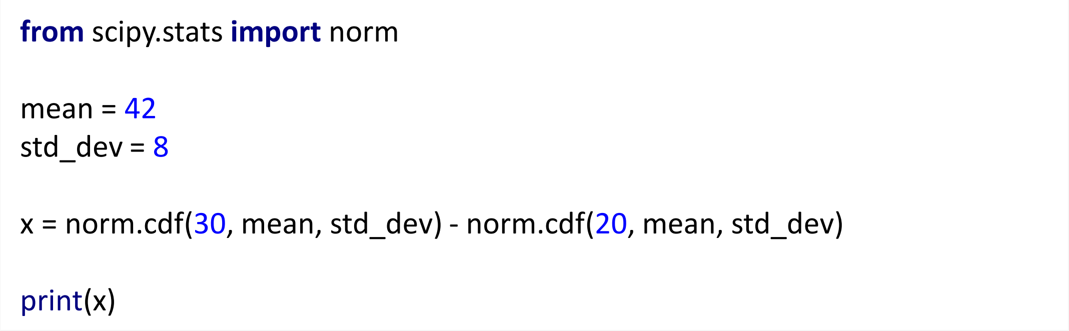
Assuming a normal distribution, what is the probability a given random Z-Phone will last between 20 and 30 months?


84LENS
Interview with photographer #10: Isadora Menezes (Brazil, Portrait photography)

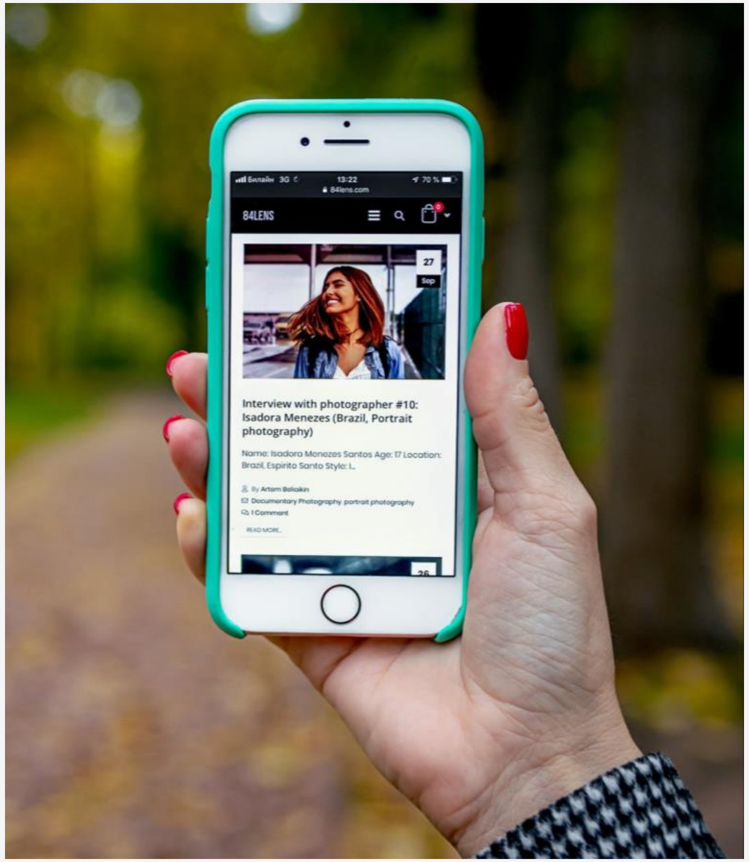
Exercise 2: ANSWER

A manufacturer says the Z-Phone smart phone has a mean consumer life of 42 months with a standard deviation of 8 months.

Assuming a normal distribution, what is the probability a given random Z-Phone will last between 20 and 30 months?

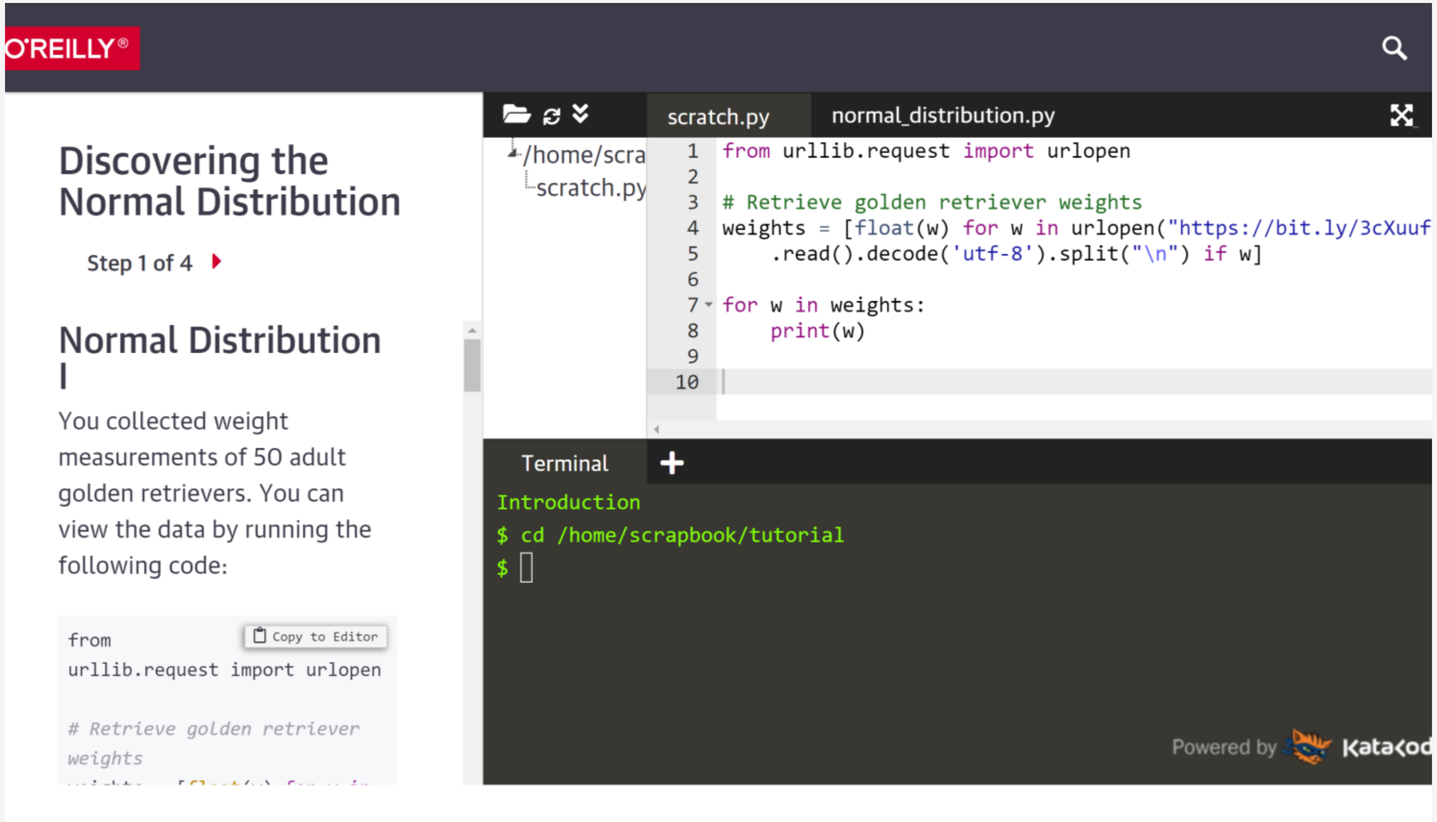


```
from scipy.stats import norm

mean = 42
std_dev = 8

x = norm.cdf(30, mean, std_dev) - norm.cdf(20, mean, std_dev)

print(x)
```


84LENS
Interview with photographer #10: Isadora Menezes (Brazil, Portrait photography)

There is a 6.3827% probability the Z-Phone will last between 20 and 30 months.

Katacoda Interactive Scenario – Normal Distribution



Discovering the Normal Distribution

Step 1 of 4

Normal Distribution I

You collected weight measurements of 50 adult golden retrievers. You can view the data by running the following code:

```
from
urllib.request import urlopen

# Retrieve golden retriever
weights
weights = [float(w) for w in
```

```python
from urllib.request import urlopen

# Retrieve golden retriever weights
weights = [float(w) for w in urlopen("https://bit.ly/3cXuuf
    .read().decode('utf-8').split("\n") if w]

for w in weights:
    print(w)

```

```
Introduction
$ cd /home/scrapbook/tutorial
$
```

https://learning.oreilly.com/scenarios/probability-from-scratch/9781492080572/

Central Limit Theorem

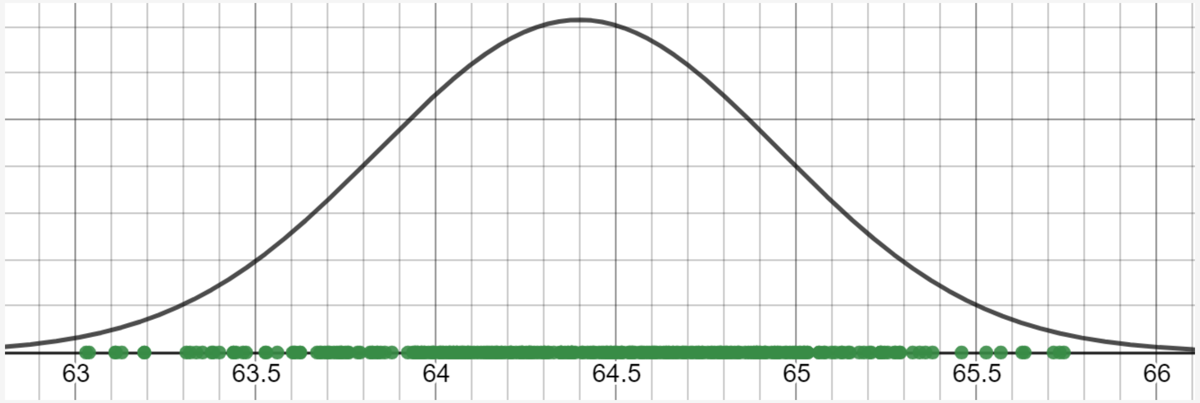
Sampling Distributions

Let's say we sample 30 golden retriever weights from a golden retriever population (sample size $n = 30$).

We obtain hundreds of more samples (each with size 30) and take the mean of each sample.

When we plot the mean of each sample to form a distribution, we will get a **sample distribution**.

To the right we see a sample distribution plotting sample means for 300 golden retriever samples (each size 30), and we happen to see a normal distribution.



Central Limit Theorem

Interesting things happen when we take all possible samples of a population, calculate the mean of each, and plot them as a distribution.

1) The mean of the sample means is equal to the population mean.

$$\mu_{\bar{x}} = \mu$$

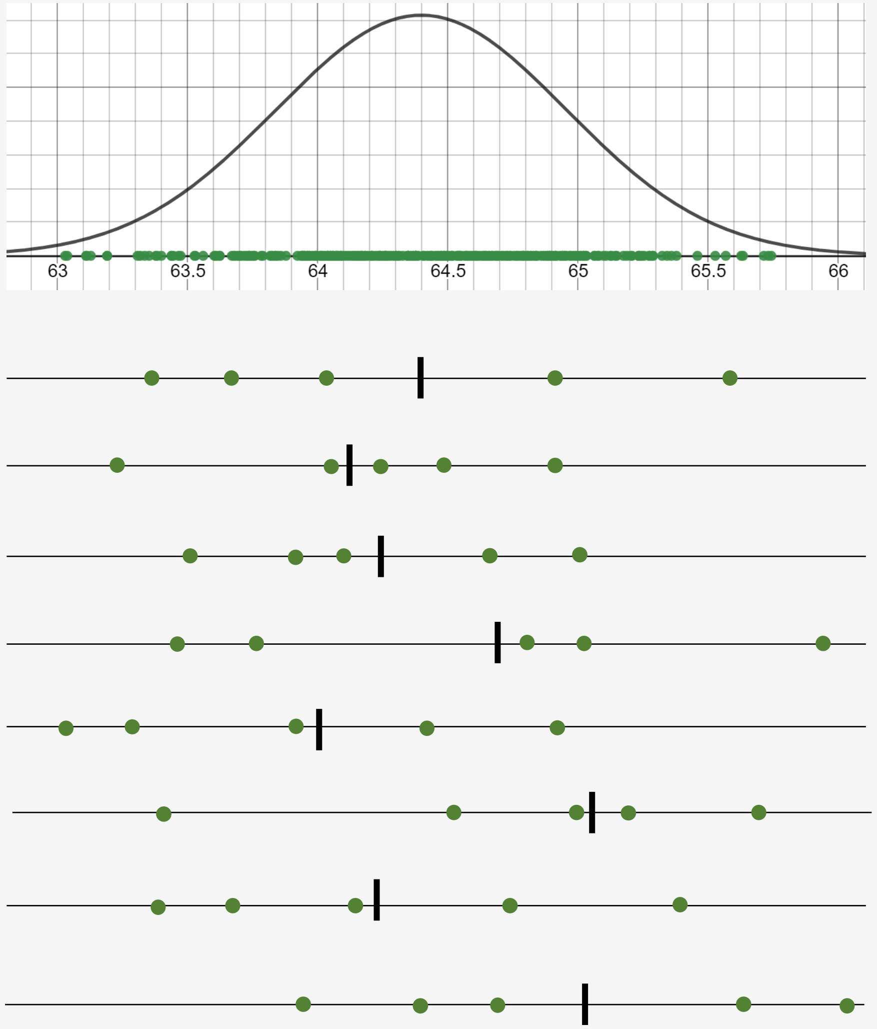
2) The standard deviation of the sample means equals the population standard deviation divided by the square root of n.

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}}$$

3) If the population is normal then the sample means will be normal.

4) If the population is not normal, but the sample size n is greater than 30, the sample means will still form a normal distribution!

Why is all the above important? It is the central limit theorem, and it allows us to infer useful things about the population based on samples, even for non-normal populations.

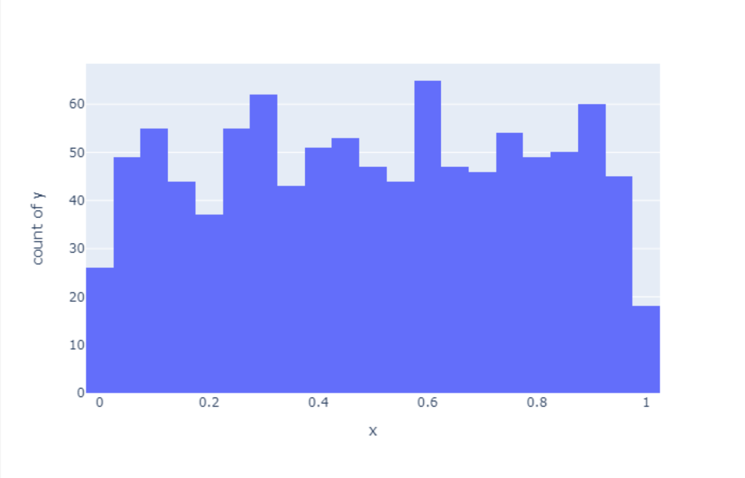


Central Limit Theorem

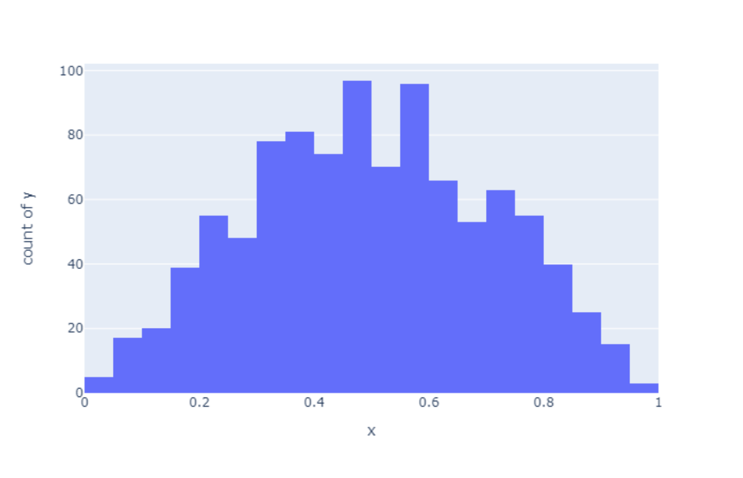
Below we generate uniformly random numbers between 0.0 and 1.0 as groups of samples and take their means.

Notice when the sample size is at least 30 we start getting a normal distribution of the means, even though the numbers are from a population that is uniform and not normal.

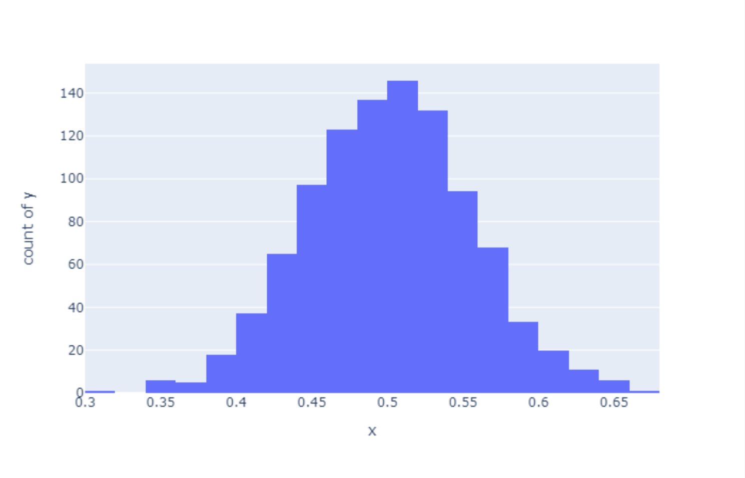
Sample size = 1



Sample size = 2



Sample size = 30



Central Limit Theorem

```
# Samples of the uniform distribution will average out to a normal distribution.

import random

import plotly.express as px

sample_size = 30
sample_count = 1000

# Central limit theorem, 1000 samples each with 30 random numbers between 0.0 and 1.0
x_values = [(sum([random.uniform(0.0, 1.0) for i in range(sample_size)]) / sample_size) for _ in range(sample_count)]
y_values = [1 for _ in range(sample_count)]

px.histogram(x=x_values, y = y_values, nbins=20).show()
```

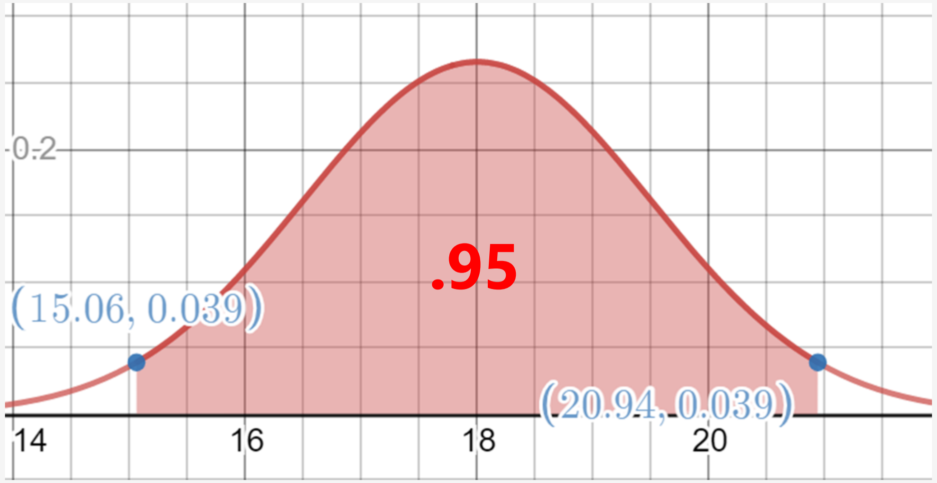
Confidence Intervals

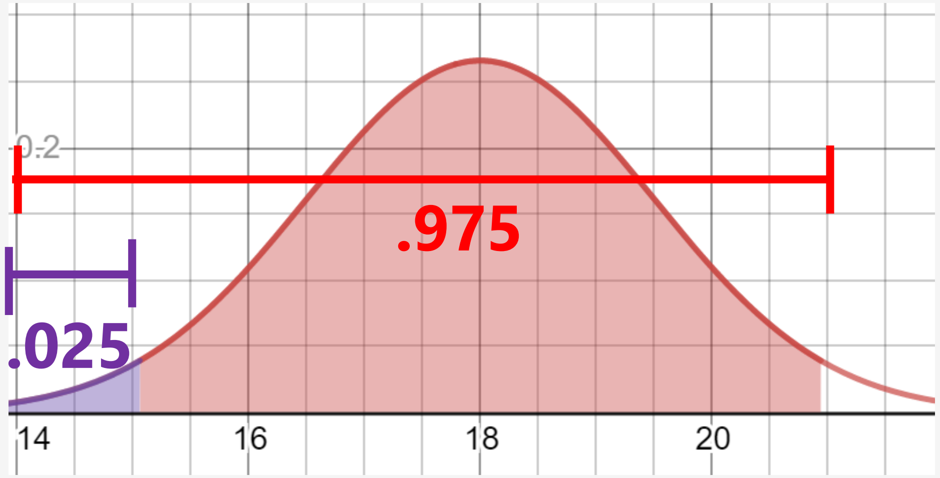
Critical Z-Value

When we work with confidence intervals, we will need to find a symmetrical range that gives us a specified area around the mean known as the critical z value.

To the right we have a normal distribution with mean of 18 and standard deviation of 1.5.

If we want to find the range that gives us .95 of the mass in the center, we need to calculate the lower x that has an area of .025 to its left, and the upper x value that has an area of .975 to its left.



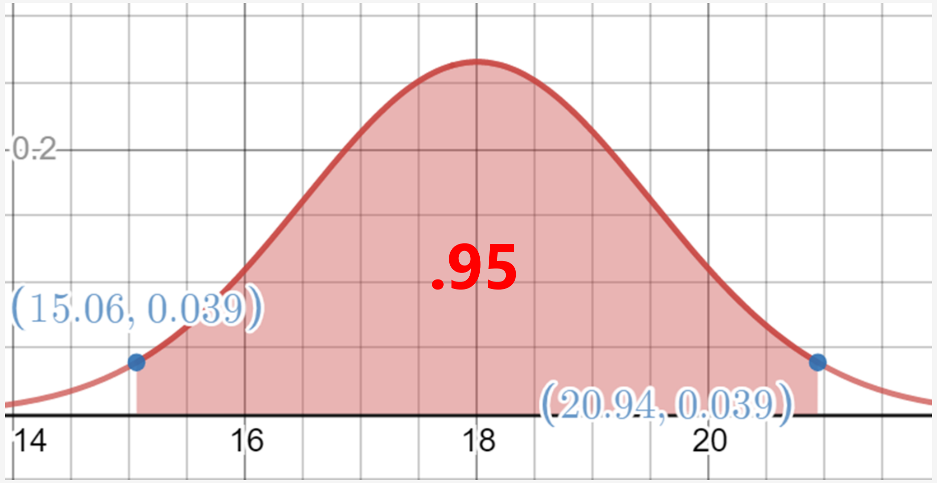


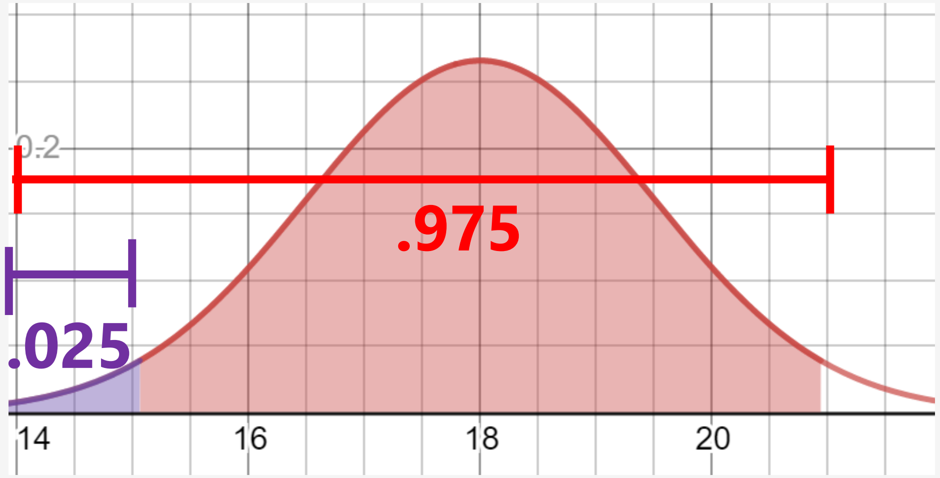
Critical Z-Value

Notice that I took the remaining .05 area (1.0 - .95 = .05) and divided it by 2, representing the remaining area in each tail.

I can then use the quantile function to look up the corresponding x values for those given areas.

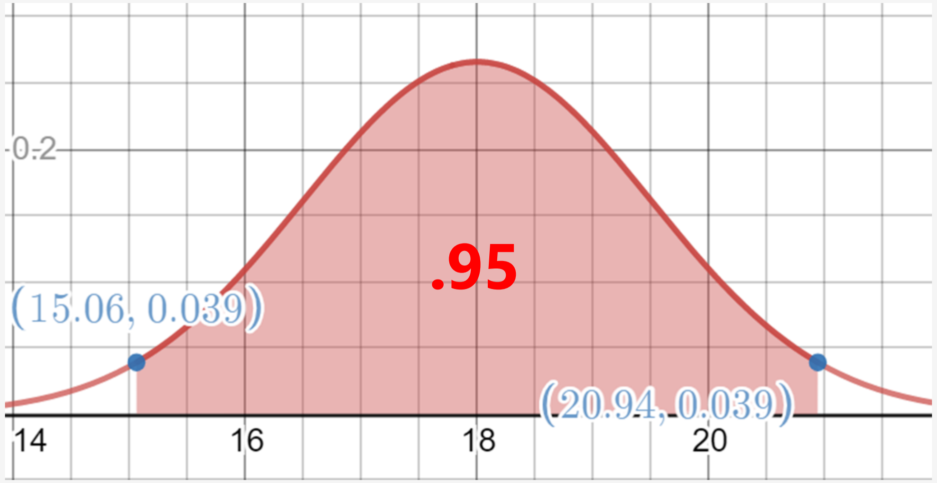
This will give us approximately 15.06 to 20.94 as our range containing 95% of density at the center of the bell curve.

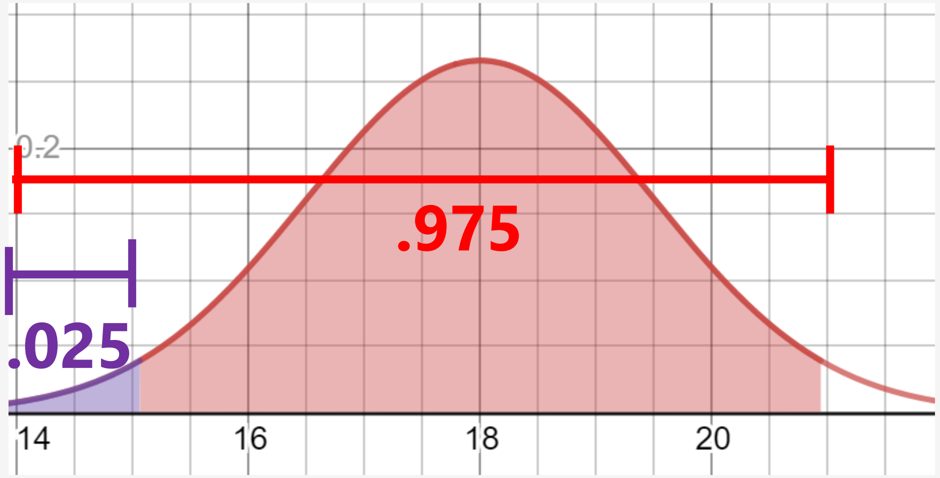




Critical Z-Value

```
from scipy.special import erfinv


def inv_normal_cdf(p: float, mean: float, std_dev: float):
    return mean + (std_dev * (2.0 ** 0.5) * erfinv((2.0 * p) - 1.0))


def z_to_x(z, mean, std):
    return (z * std) + mean


def critical_z_value(p, mean=0.0, std=1.0):
    left_area = (1.0 - p) / 2.0
    right_area = 1.0 - ((1.0 - p) / 2.0)

    lower_cutoff = inv_normal_cdf(left_area, mean, std)
    upper_cutoff = inv_normal_cdf(right_area, mean, std)

    return lower_cutoff, upper_cutoff


# What range around the mean gives an area of .95?
print(critical_z_value(p=.95, mean=18, std=1.5))
# (15.060054023189918, 20.93994597681008)
```





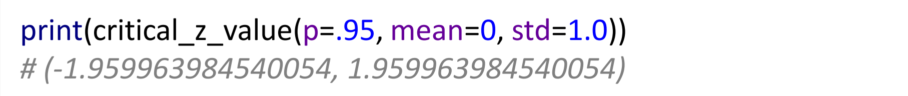
Confidence Intervals

When I take a sample of size more than 30 from a normally distributed population, I can calculate a probability that a population parameter falls in a range known as a confidence interval.

If I have a sample of 31 golden retriever weights with mean of 64.408 and standard deviation of 2.05, I know with 95% confidence the population weight mean is between 63.686 and 65.1296.

How do I calculate this?

1) I take the desired **level of confidence** (e.g. 95%) and get the critical Z-value from a standard normal distribution (mean = 0, stdev = 1).



```
print(critical_z_value(p=.95, mean=0, std=1.0))
# (-1.959963984540054, 1.959963984540054)
```

2) Then I multiply that critical z-value into his formula (right) which will give me the **margin of error**, or the range around the sample mean that contains the population mean at that confidence:

$$E = \pm z_c \frac{s}{\sqrt{n}}$$

$$\pm 1.95996 \left(\frac{2.05}{\sqrt{31}}\right) = \pm 0.72$$

Confidence Intervals

3) Finally, I take that value and add/subtract it from the sample mean to give me my confidence interval.

$$\boxed{\begin{array}{c}64.408 \pm 0.72 \\ \\ (63.686, 65.1296)\end{array}}$$

The way to interpret this:

"Based on my sample of 31 golden retriever weights with sample mean 64.408 and sample standard deviation of 2.05, I am 95% confident the population mean lies between $\boldsymbol{63.686\ and\ 65.1296}$***."***

An interesting thing to note is that the larger your sample, the narrower and more confident your confidence interval becomes!

$$E = \pm z_c \frac{s}{\sqrt{n}}$$

Confidence Intervals

```python
from scipy.special import erfinv
from math import sqrt


def inv_normal_cdf(p: float, mean: float, std_dev: float):
    return mean + (std_dev * (2.0 ** 0.5) * erfinv((2.0 * p) - 1.0))


def z_to_x(z, mean, std):
    return (z * std) + mean


def critical_z_value(p, mean=0.0, std=1.0):
    left_area = (1.0 - p) / 2.0
    right_area = 1.0 - ((1.0 - p) / 2.0)

    lower_cutoff = inv_normal_cdf(left_area, mean, std)
    upper_cutoff = inv_normal_cdf(right_area, mean, std)

    return lower_cutoff, upper_cutoff


def ci_large_sample(p, sample_mean, sample_std, n):
    # Sample size must be greater than 30

    lower, upper = critical_z_value(p)
    lower_ci = lower * (sample_std / sqrt(n))
    upper_ci = upper * (sample_std / sqrt(n))

    return sample_mean + lower_ci, sample_mean + upper_ci


# What is the confidence interval my sample of 31 golden retrievers
# with sample mean of 65.13, sample std of 2.05?
print(ci_large_sample(p=.95, sample_mean=65.13, sample_std=2.05, n=31))
# (64.40835915701992, 65.85164084298007)
```

Confidence Intervals (SciPy)

```
from math import sqrt
from scipy.stats import norm

def critical_z_value(p, mean=0.0, std=1.0):
    norm_dist = norm(loc=mean, scale=std)
    left_area = (1.0 - p) / 2.0
    right_area = 1.0 - ((1.0 - p) / 2.0)
    return norm_dist.ppf(left_area), norm_dist.ppf(right_area)


def ci_large_sample(p, sample_mean, sample_std, n):
    # Sample size must be greater than 30

    lower, upper = critical_z_value(p)
    lower_ci = lower * (sample_std / sqrt(n))
    upper_ci = upper * (sample_std / sqrt(n))

    return sample_mean + lower_ci, sample_mean + upper_ci


# What is the confidence interval my sample of 31 golden retrievers
# with sample mean of 65.13, sample std of 2.05?
print(ci_large_sample(p=.95, sample_mean=65.13, sample_std=2.05, n=31))
# (64.40835915701992, 65.85164084298007)
```

Exercise 3

I am skeptical that my 3D printer filament is not 1.75mm in average diameter as advertised.

I sampled 34 measurements with my tool and recorded them here: https://bit.ly/2JZhopE

To save you time, the sample mean is 1.715588 and the sample standard deviation is 0.029252.

What is the 99% confidence interval for the mean of my entire spool of filament?

Exercise 3: ANSWER

I am skeptical that my 3D printer filament is not 1.75mm in average diameter as advertised.

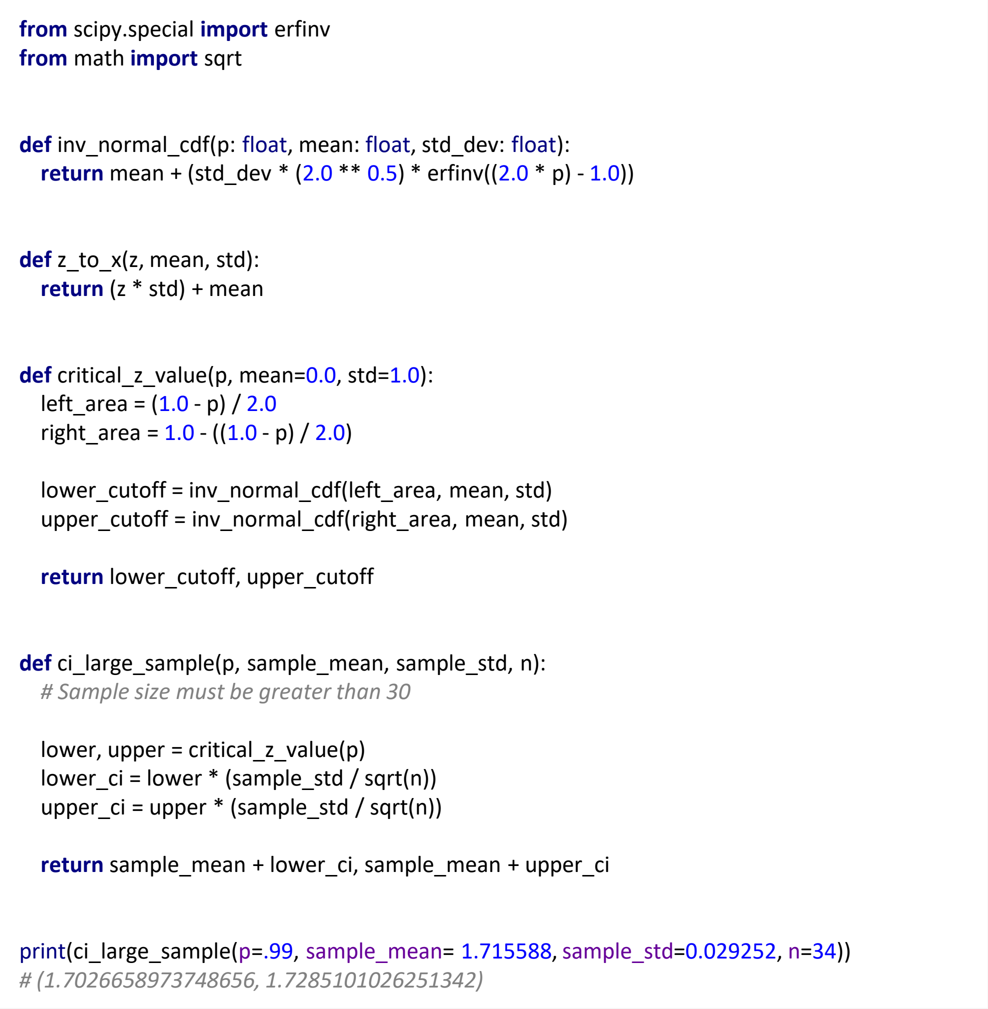
I sampled 34 measurements with my tool and recorded them here: https://bit.ly/2JZhopE

To save you time, the sample mean is 1.715588 and the sample standard deviation is 0.029252.

What is the 99% confidence interval for the mean of my entire spool of filament?

(1.7026, 1.7285)

Assuming my tool is accurate, it is unlikely the average diameter is 1.75mm because there's 99% probability it falls in this range!



```
from scipy.special import erfinv
from math import sqrt


def inv_normal_cdf(p: float, mean: float, std_dev: float):
    return mean + (std_dev * (2.0 ** 0.5) * erfinv((2.0 * p) - 1.0))


def z_to_x(z, mean, std):
    return (z * std) + mean


def critical_z_value(p, mean=0.0, std=1.0):
    left_area = (1.0 - p) / 2.0
    right_area = 1.0 - ((1.0 - p) / 2.0)

    lower_cutoff = inv_normal_cdf(left_area, mean, std)
    upper_cutoff = inv_normal_cdf(right_area, mean, std)

    return lower_cutoff, upper_cutoff


def ci_large_sample(p, sample_mean, sample_std, n):
    # Sample size must be greater than 30

    lower, upper = critical_z_value(p)
    lower_ci = lower * (sample_std / sqrt(n))
    upper_ci = upper * (sample_std / sqrt(n))

    return sample_mean + lower_ci, sample_mean + upper_ci


print(ci_large_sample(p=.99, sample_mean= 1.715588, sample_std=0.029252, n=34))
# (1.7026658973748656, 1.7285101026251342)
```

P-Values and Z-Tests

P-Values and Statistical Significance

DISCUSSION: what does it mean when someone says an outcome is “statistically significant?”

A p-value is the probability that you have gotten an outcome by random luck, rather than because of a changed variable.

The p-value was invented by Ronald Fisher in 1925, and it is best demonstrated in his party experiment:[1]

- A colleague claimed she could detect when tea was poured before milk, so he made her 8 cups of tea as an experiment.
- 4 had milk poured first, the other 4 had tea poured first.
- She identified them all correctly, and the probability of this happening by chance is 1 in 70, or 0.01428571, which is the P-value.

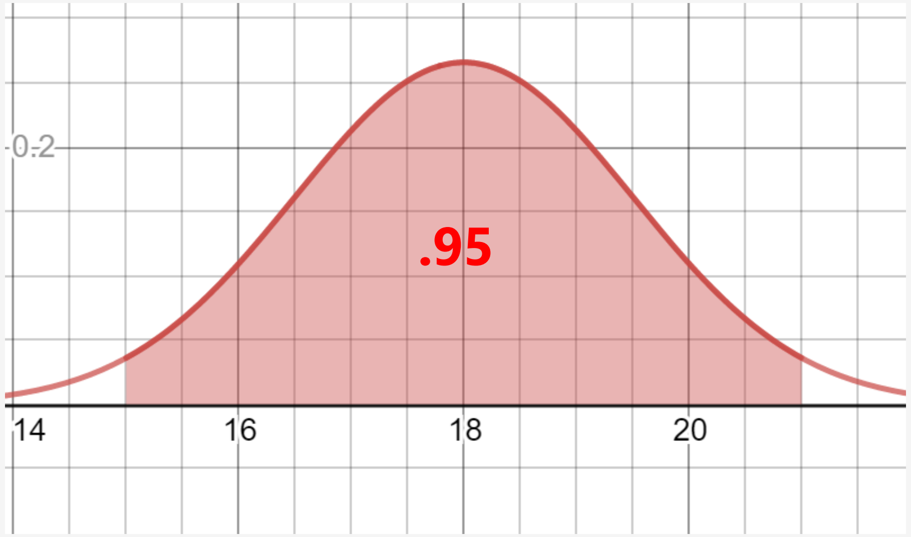
Conventionally a p-value of .05 or less is considered statistically significant, although this *arbitrary* number has come under scrutiny as data availability has made statistically significant findings (which can still occur by chance) not hard to produce.

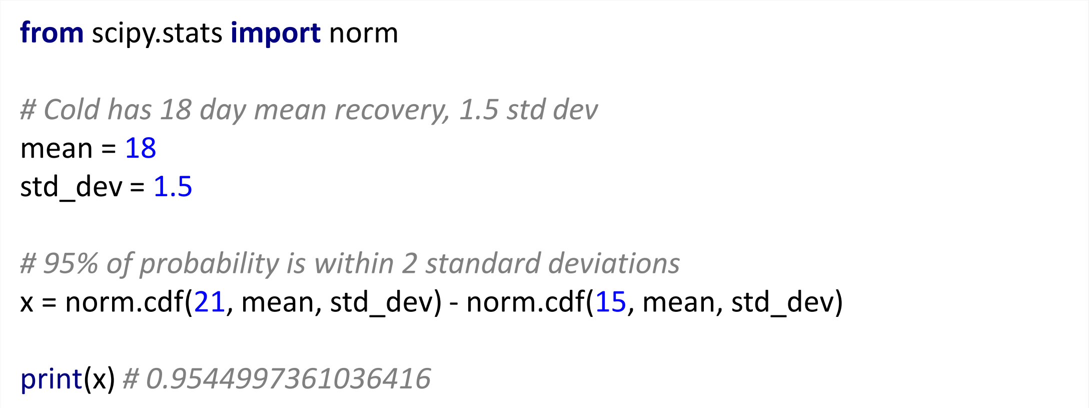
[1] https://en.wikipedia.org/wiki/Lady_tasting_tea

Z-Test

Past studies have shown the mean recovery time for a cold is 18 days with a standard deviation of 1.5 days, and follows a normal distribution

This means there is a 95% chance of recovery taking between 15 and 21 days.





```
from scipy.stats import norm

# Cold has 18 day mean recovery, 1.5 std dev
mean = 18
std_dev = 1.5

# 95% of probability is within 2 standard deviations
x = norm.cdf(21, mean, std_dev) - norm.cdf(15, mean, std_dev)

print(x) # 0.9544997361036416
```

This also means there’s a 2.5% chance of recovery taking longer than 21 days, and 2.5% chance of less than 15 days.

Z-Test

Now let's say an experimental new drug was given to a test group pf 40 people and it took an average of 16 days to recover.

Does the drug work? What if this happened by chance and the drug had no effect?

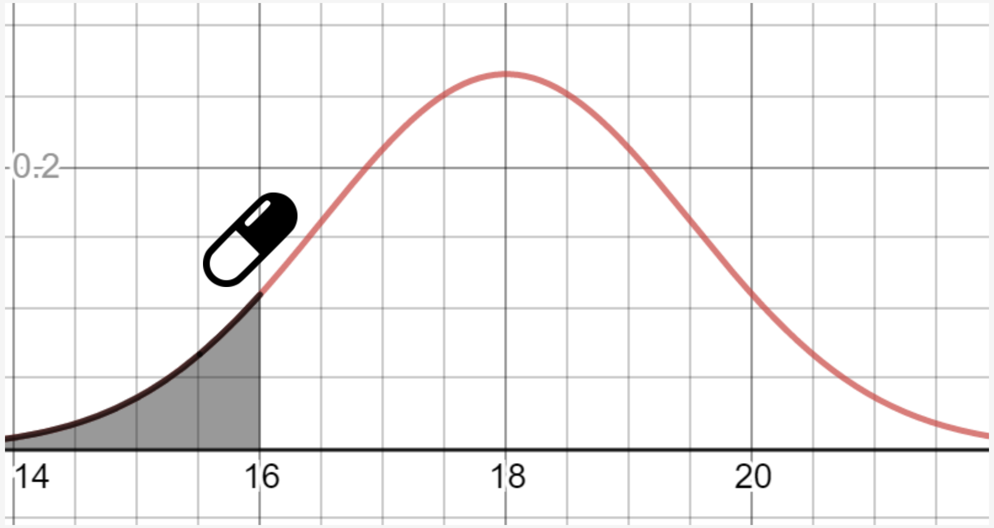
When we have two large sample studies (n > 30) and compare them, this is known as a Z-Test.

We first establish our null hypothesis (H0), the status quo/currently accepted value for a parameter (i.e. the mean recovery for a cold is 18 days).

$$H_0: \mu = 18$$

Our alternative hypothesis (H1) is the claim we are testing: our drug reduces the recovery time of a cold.

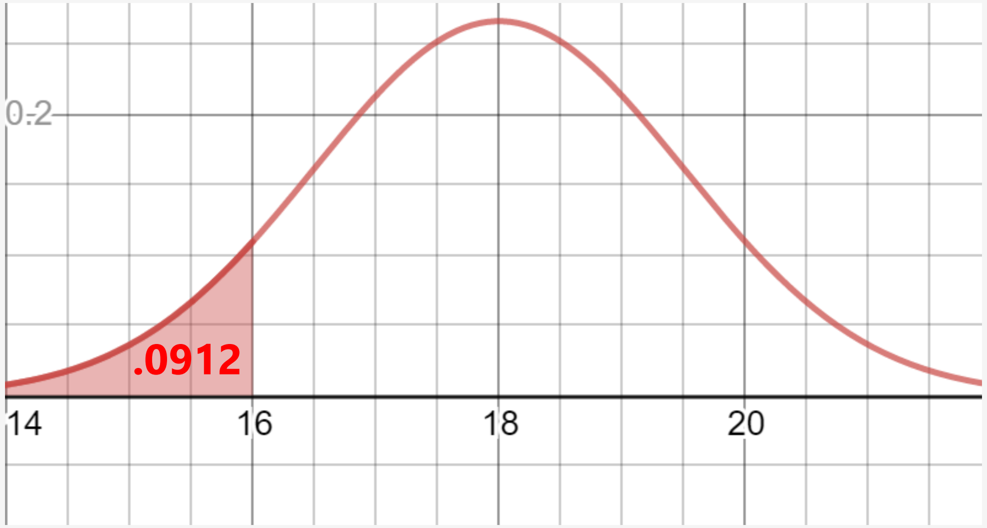
$$H_1: \mu < 18$$



Discovering the P-Value

First let's calculate the probability of recovering less than 16 days without the drug treatment (shaded on the right).

```python
from scipy.stats import norm

# Cold has 18 day mean recovery, 1.5 std dev
mean = 18
std_dev = 1.5

# Probability of 16 or less days
x = norm.cdf(16, mean, std_dev)

print(x) # 0.09121121972586788
```



There is a 9.12% probability any person would recover in less than 16 days without the drug.

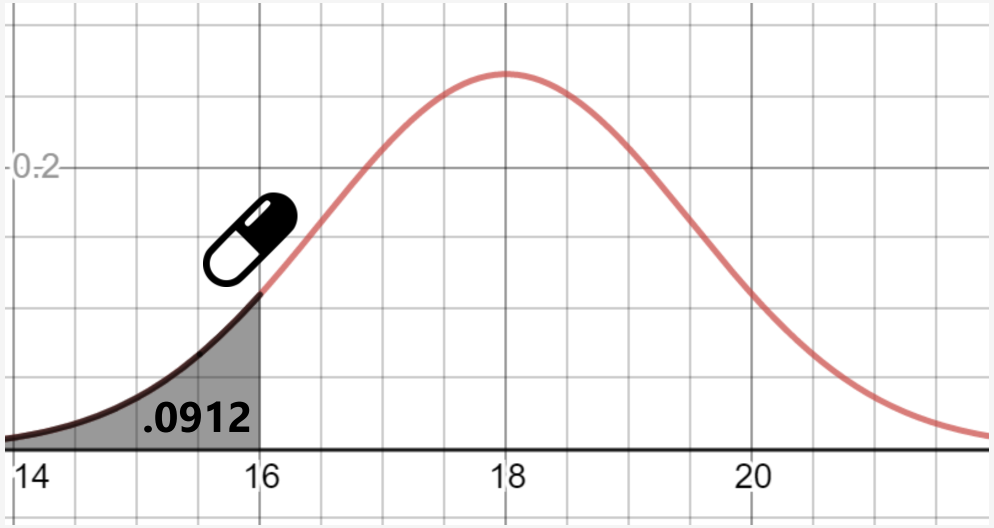
One-Sided P-Value

This 9.12% probability of observing 16 or less days of recovery is known as the one-sided p-value, the observed difference that could have just occurred by chance.

In other words, this is the equal or less likelihood we would see this observation without the drug.

By a traditional cutoff of 5% statistical significance, this would *fail to reject the null hypothesis* and conclude there's no significant evidence this drug works.

In other words, there's a 9.12% probability this pill did not work, and we just observed this by chance from the existing distribution and not a new one.



Two-Sided P-Value

What if we wanted to evaluate if the drug had *any* impact, even if it increased the duration of the cold?

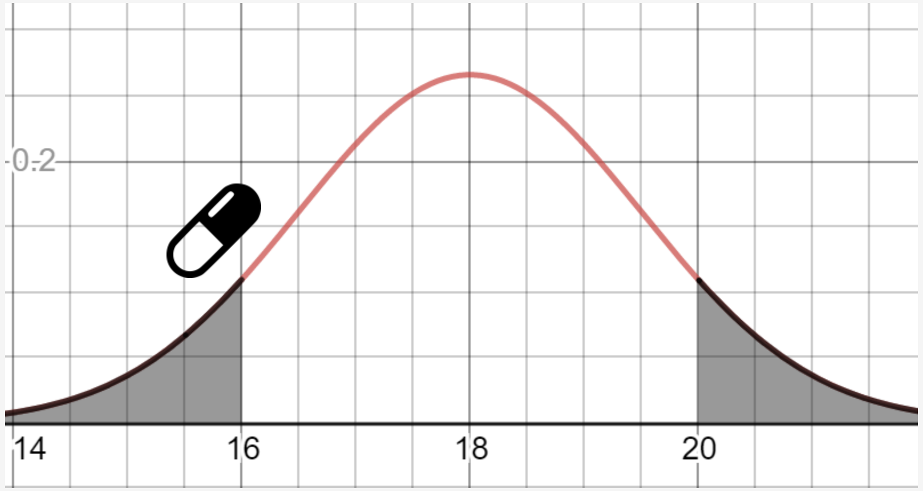
$$H_0: \mu = 18$$

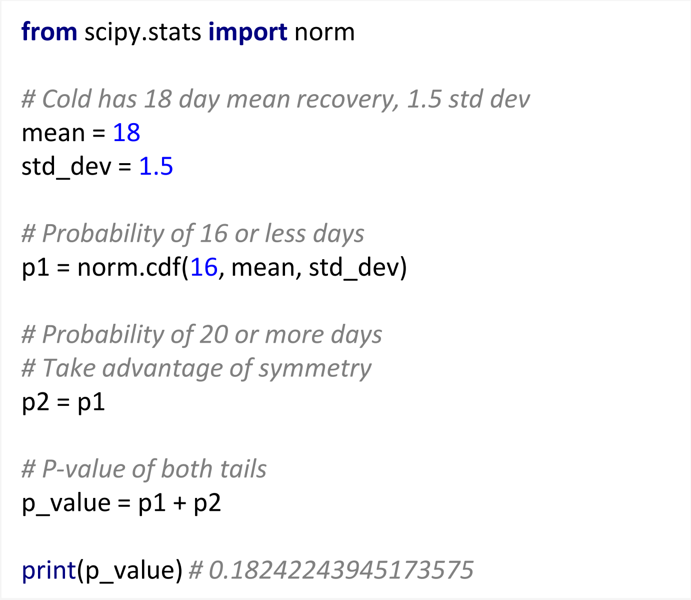
$$H_1: \mu \neq 18$$

Should we count the symmetrical probability on the other side too? Yes!

Counting both tails of equal or less likely probability is known as the **two-sided p-value**, and you should prefer it as it will more reliably capture statistical significance *REGARDLESS OF DIRECTION*.

Two-sided p-values not only sets a higher threshold for significance, but it can also detect unusual activity in both tails (what if the drug prolonged colds too?)





```
from scipy.stats import norm

# Cold has 18 day mean recovery, 1.5 std dev
mean = 18
std_dev = 1.5

# Probability of 16 or less days
p1 = norm.cdf(16, mean, std_dev)

# Probability of 20 or more days
# Take advantage of symmetry
p2 = p1

# P-value of both tails
p_value = p1 + p2

print(p_value) # 0.18242243945173575
```

Exercise

Your marketing department has started a new advertising campaign and wants to know if it affected sales, which in the past averaged $11,641 a day with a standard deviation of $552.

The new advertising campaign ran for 45 days and averaged $11,641 in sales.

Did the campaign affect sales? Why or why not?

Exercise

Yes! Or very likely. The campaign worked as the p-value is .01888, which is less than .05.

This means there is a 1.888% chance these sales numbers would have happened by random chance.

```
from scipy.stats import norm

# $10,345 in sales on average, $552 standard deviation
mean = 10345
std_dev = 552

# Probability of $11,641 or more
p1 = 1.0 - norm.cdf(11641, mean, std_dev)

# Take advantage of symmetry for other side
p2 = p1

# P-value of both tails
p_value = p1 + p2

print(p_value) # 0.01888333596496139
```

T-Distribution

Student T-Distribution

The T-distribution closely resembles the standard normal distribution, but it allows us to account for variance when we have 30 or less items in a sample.

Rather than have a mean and standard deviation, it instead only has one parameter: degrees of freedom.

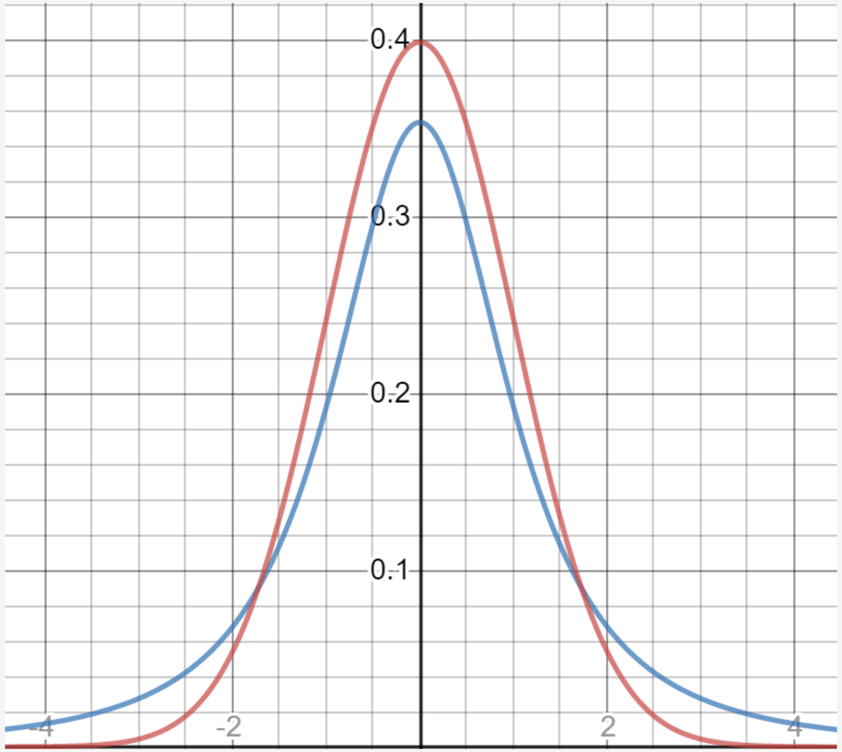
The **degrees of freedom** is sample size minus 1:

$$dof = n - 1$$

Note that it has fatter tails to capture more variance, and when degrees of freedom exceeds 30, it approaches the normal distribution.

The smaller the sample size, the more variance it has.

Desmos graph: https://www.desmos.com/calculator/h7lbfelf7b

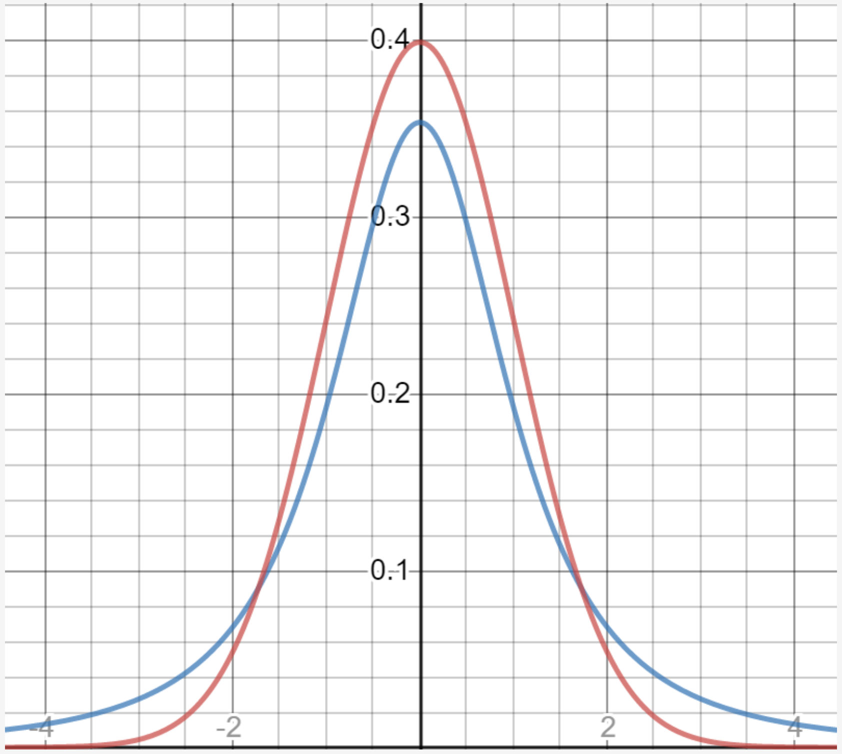


Using the T-Distribution

You might see where this can be useful… if I have a sample size that is 30 or less, I can use a T-distribution instead of the normal distribution to create a wider confidence interval.

This captures a greater uncertainty about the mean by increasing variance..

Just make sure you swap out your normal distribution for a T-distribution when calculating your critical value for a level of confidence and account the (sample size -1) as your parameter.



Confidence Intervals for Small Samples

```
from math import sqrt
from scipy.stats import t

def critical_t_value(p, n):
    t_dist = t(df=n - 1)
    left_area = (1.0 - p) / 2.0
    right_area = 1.0 - ((1.0 - p) / 2.0)
    return t_dist.ppf(left_area), t_dist.ppf(right_area)


def ci_small_sample(p, sample_mean, sample_std, n):
    # Sample size must be greater than 30

    lower, upper = critical_t_value(p, n)
    lower_ci = lower * (sample_std / sqrt(n))
    upper_ci = upper * (sample_std / sqrt(n))

    return sample_mean + lower_ci, sample_mean + upper_ci


# What is the confidence interval my sample of 15 golden retrievers
# with sample mean of 65.13, sample std of 2.05?
print(ci_small_sample(p=.95, sample_mean=64.27, sample_std=2.75,
n=15))
# (62.74710076069723, 65.79289923930276)
```

T-Test

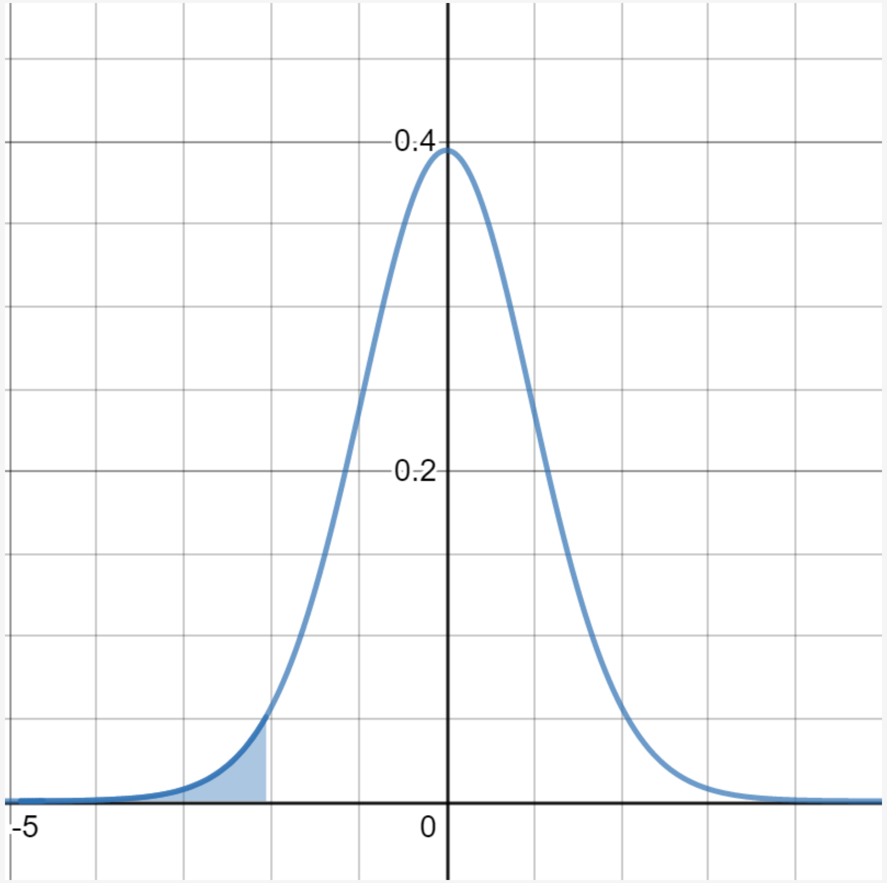
T-Test

In our Z-test example, we had established large sample parameters (mean and standard deviation) we could compare our sample to.

But what if the sample size is too small to have a known standard deviation?

We can still infer some things about the population using a T-Test, which uses a T-distribution to infer parameters about the population from a sample of 30 or less.

If you have more than 30 in your sample, use the Z-test.



T-Test

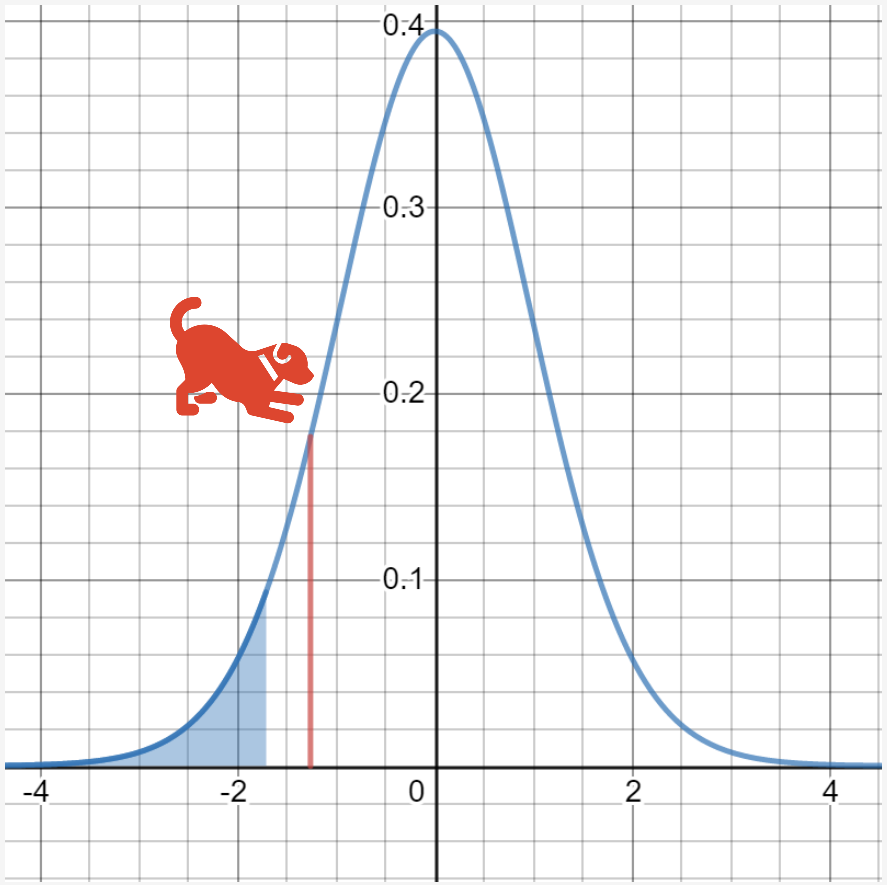
Past studies show that golden retrievers have a weight greater than 65.5 and let's say we sample 25 golden retrievers.

The sample mean is 64.7 and the sample standard deviation is 3.17. With a confidence of 95% can we say the population mean is below 65.5?

$$H_0 : \mu \geq 65.5$$

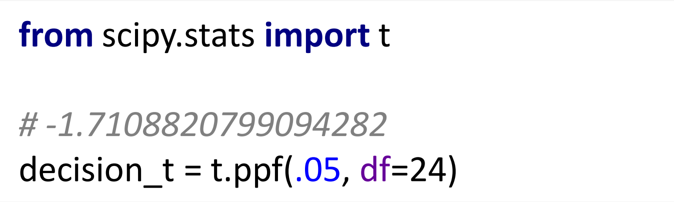
$$H_1 : \mu < 65.5$$

This is what the **T-test** is for: inferring where a population parameter lies with a certain confidence with only a small sample.



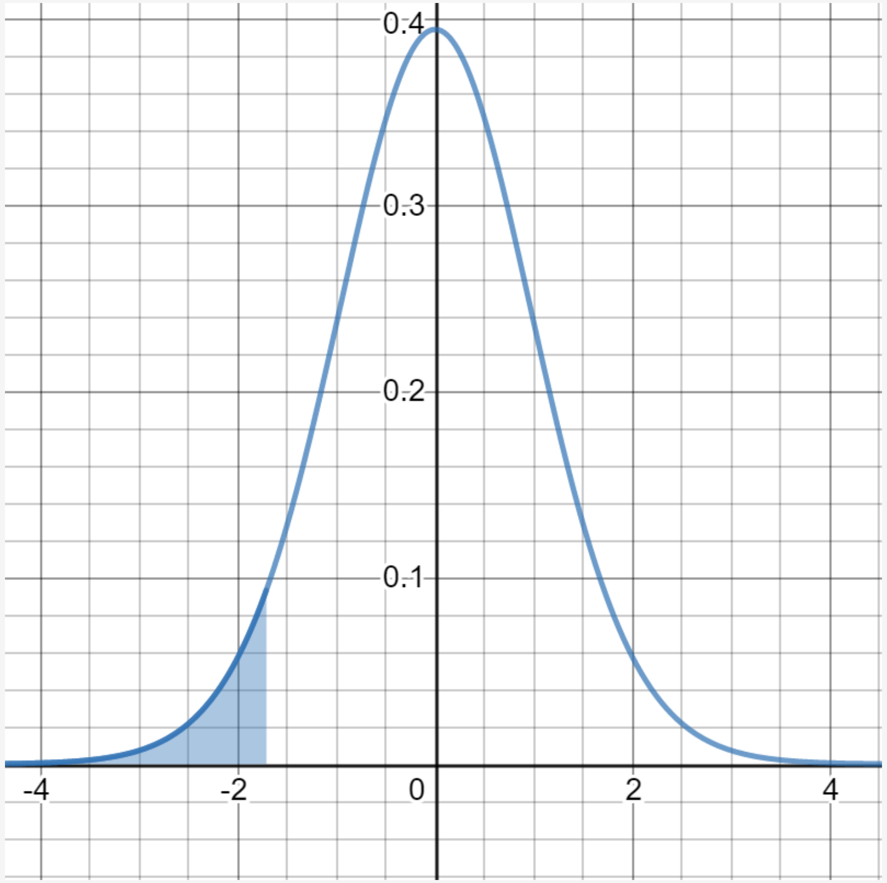
T-Test

First calculate the T-value on a T-distribution with 24 degrees of freedom (n – 1) which would put .05 of the area to its left.



```
from scipy.stats import t

# -1.7108820799094282
decision_t = t.ppf(.05, df=24)
```

If our T-value is less than -1.71 then we will reject our null hypothesis H_0 that $\mu \geq 65.5$.



T-Test

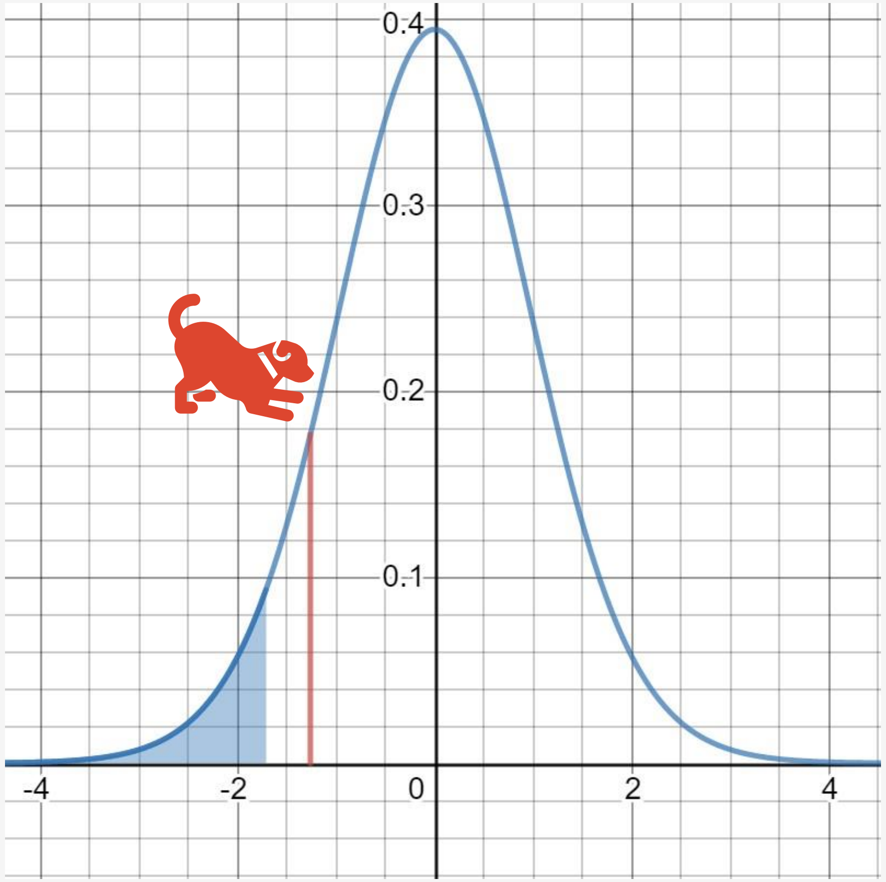
Next we need to calculate the T-value given our sample mean, sample standard deviation, and the population mean.

$$t = \frac{\bar{x} - \mu}{s/\sqrt{n}}$$

$$\frac{64.7 - 65.5}{3.17\sqrt{24}} = -1.26$$

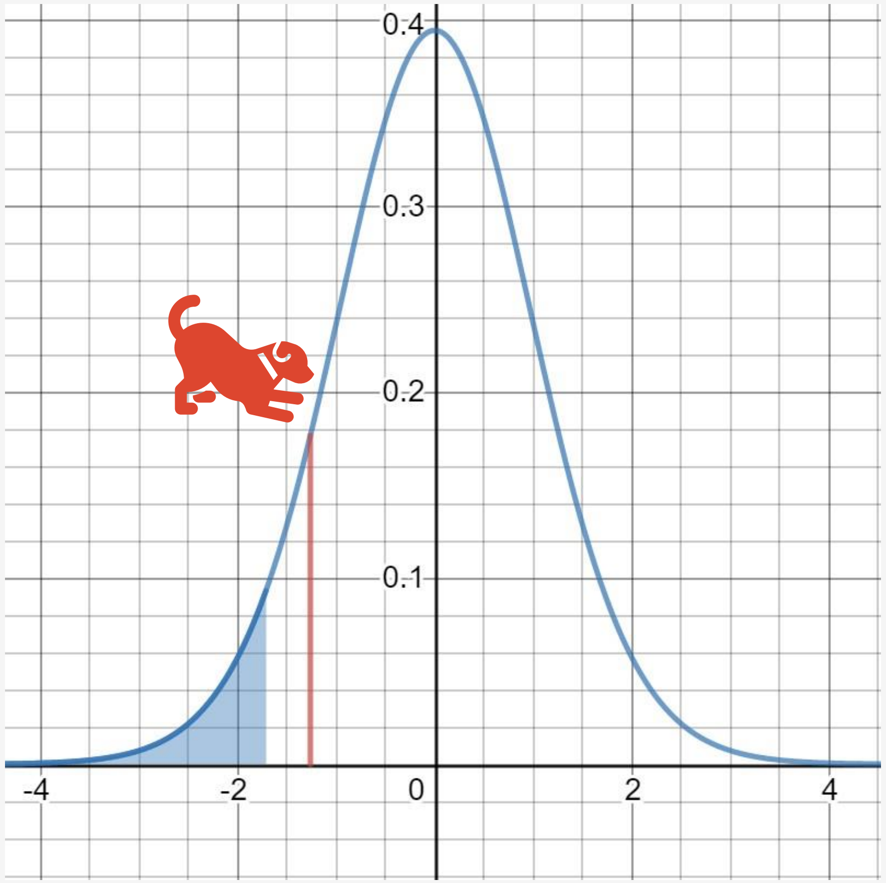
Since our T-value of -1.26 did not fall in the H_0 rejection range to the left of -1.71 we fail to reject our null hypothesis.

Therefore, we fail to reject our null hypothesis and cannot say with 95% confidence that our population mean is below 65.5.



T-Test

```
from math import sqrt
from scipy.stats import t

def t_test(sample_mean, sample_std, pop_mean, n):
    return (sample_mean - pop_mean) / (sample_std / sqrt(n))

sample_mean = 64.7
sample_std = 3.17
pop_mean = 65.5
n = 25

# -1.7108820799094282
decision_t = t.ppf(.05, df=n-1)

# -1.261829652996841
t_value = t_test(sample_mean,sample_std,pop_mean, n)

print("DECISION T: {}".format(decision_t))
print("TEST T: {}".format(t_value))

# FAILED TO REJECT H0
if decision_t <= t_value:
    print("REJECT H0")
else:
    print("FAILED TO REJECT H0")
```



Paired vs Unpaired T-Tests

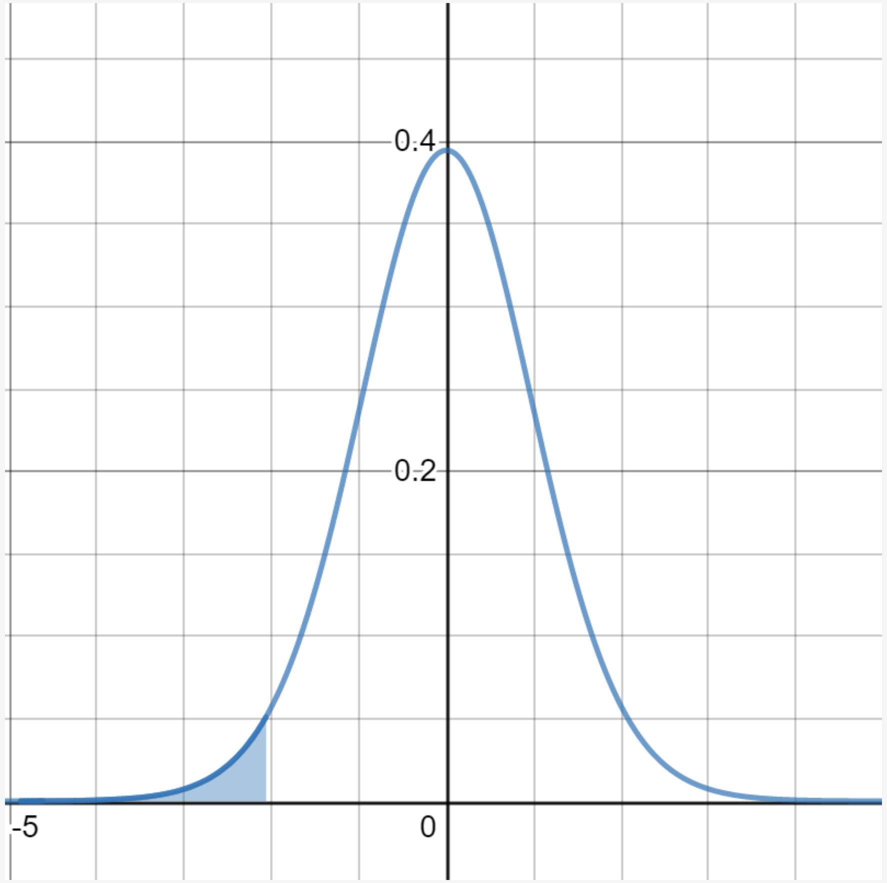
Paired T-Tests: analyzes “before” and “after” measurements from the same test subjects (e.g. testing a weight loss supplement with same group).

Unpaired T-Tests: uses two different groups, which can fall into two categories.

- Variance is assumed between equal between both groups.

- Variance is assumed different between both groups and is usually preferable as it’s more conservative (this was also our previous golden retriever example)

If you want to be conservative, always structure your hypothesis towards a two-tailed test which more reliably improves significance.



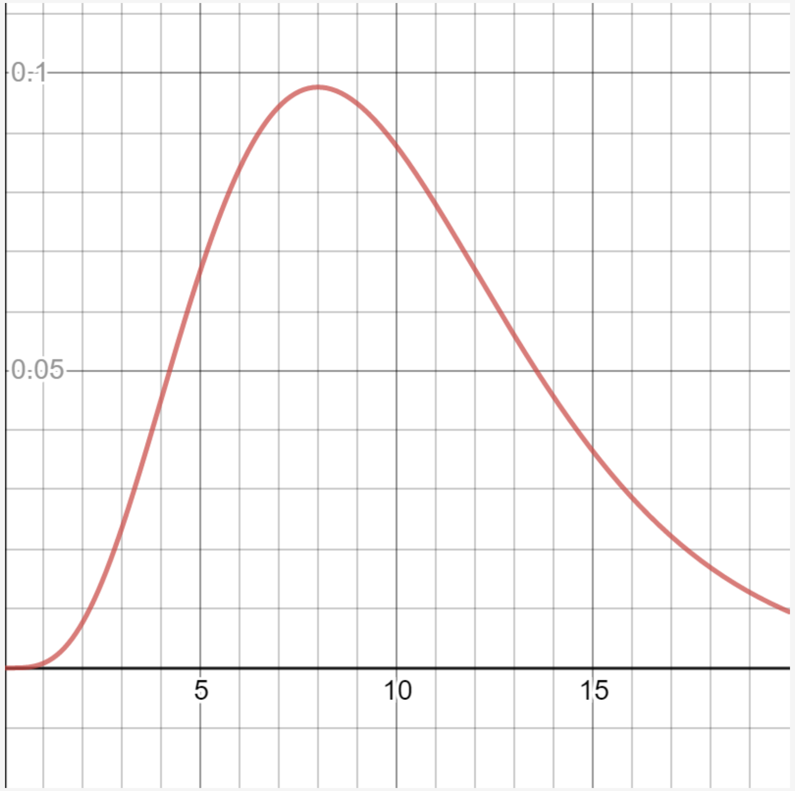
Beyond the Mean

In this class we focused primarily on confidence intervals and hypothesis testing for the mean.

However, we can adapt these tools to estimate other parameters beyond the mean.

- Proportions of a population (percentages)
- Variance/Standard Deviation of a population

Discussing these is beyond the scope of this class, but know confidence intervals and hypothesis testing can be adapted to find other parameters.



The Chi-Square distribution is useful in estimating variance/standard deviation for a population from a sample.

Big Data Considerations

Texas Sharpshooter Fallacy

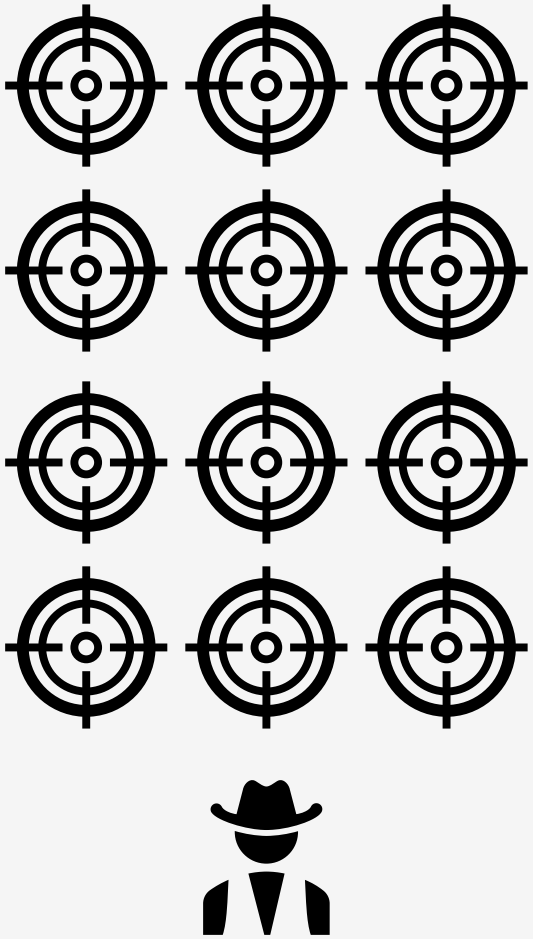
Imagine you filled a room with targets.

Not having to shoot a specific target, it is easy to hit any target.

Alternatively, what if I shoot at a blank wall and draw a target around the hole? Did I achieve anything?

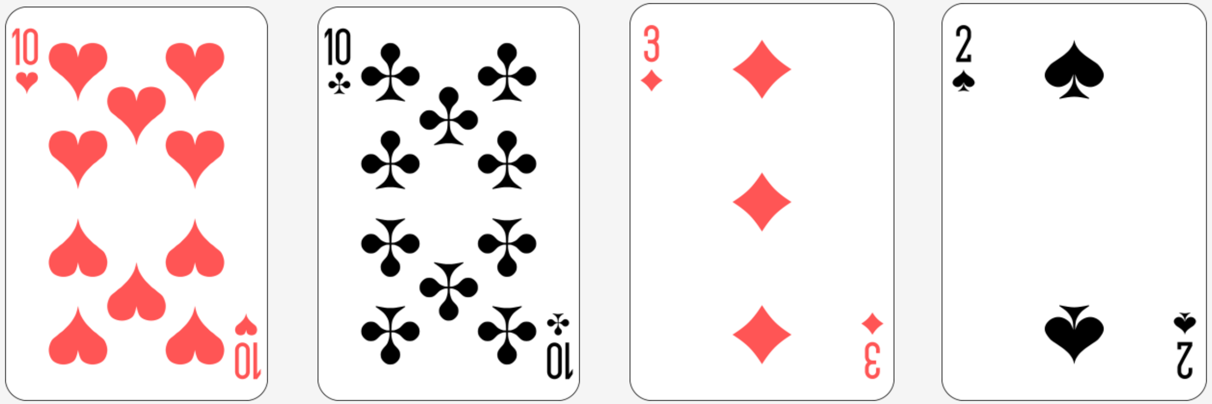
When you collect enormous amounts of data without an objective in mind, it is easy to develop multiple hypotheses afterwards based on analysis.

This can lead to the Texas Sharpshooter Fallacy, where we stumble on spurious findings that are coincidental in the gathered data, because the data was not gathered for a specific objective.



Texas Sharpshooter Fallacy

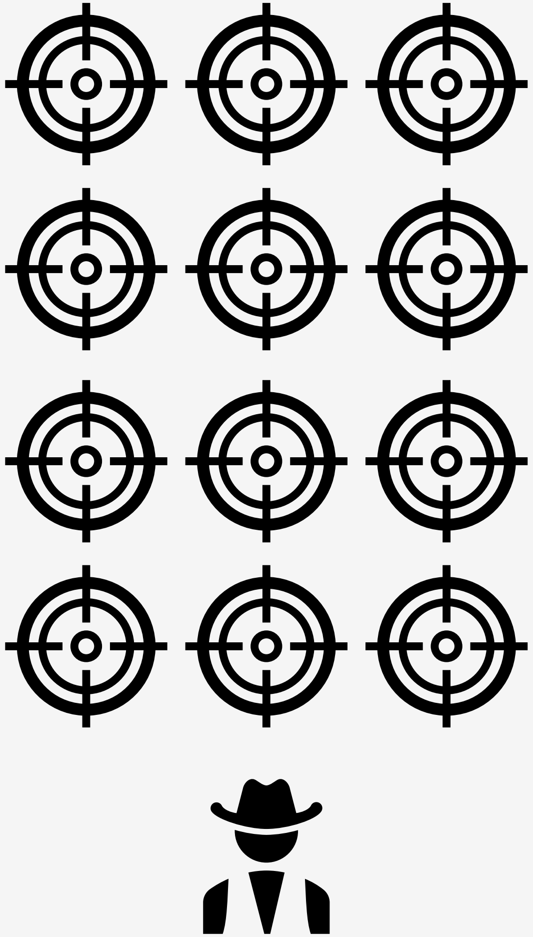
Here is another way to think about it: I draw 4 playing cards at random.



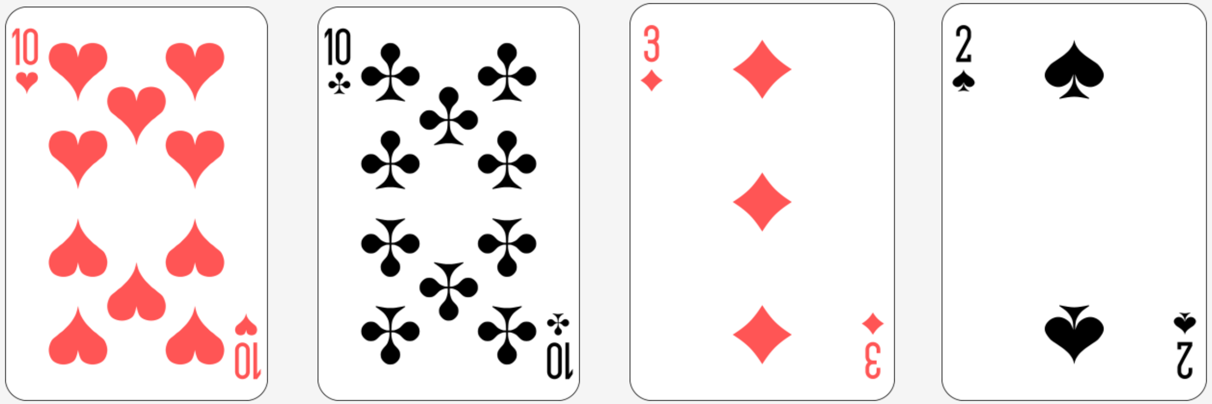
Is this not fascinating? I have two 10's and a consecutive 3 and 2! What does it mean?

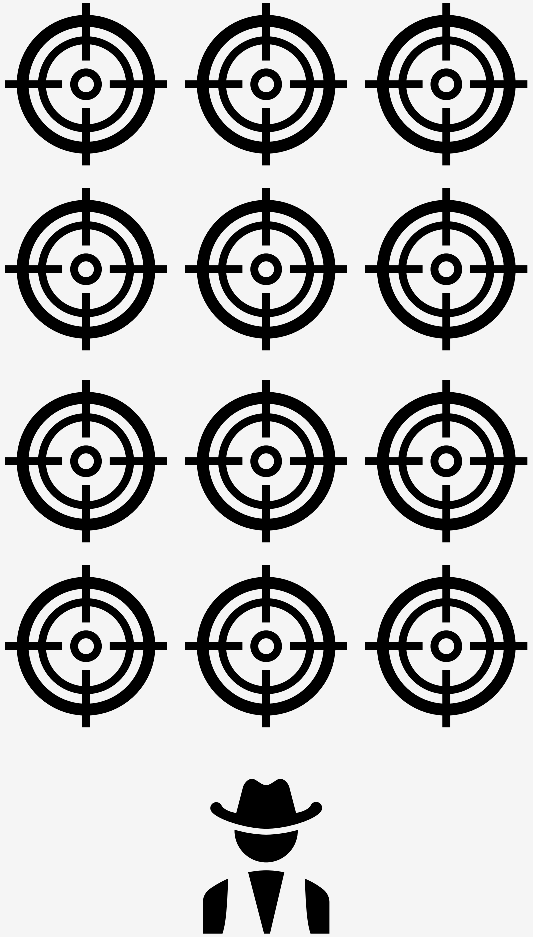
On the next draw, will I get another pair and two consecutive numbers?

I never made it my objective to achieve this outcome, I observed it ***AFTER*** it occurred.



Texas Sharpshooter Fallacy





Unless I predicted this specific outcome beforehand, there is nothing meaningful I have accomplished.

This is exactly what data mining can fall victim to: finding patterns in random events and interpreting them as meaningful.

Scientific Method versus Data Mining

The problem with the previous examples is we practiced data mining rather than the scientific method.

Data mining: Gather data *then* hypothesize

Scientific Method: Hypothesize *then* gather data

The scientific method demands a structured hypothesis, and data is deliberately gathered to prove and disprove that hypothesis, by using a test group and a control group.

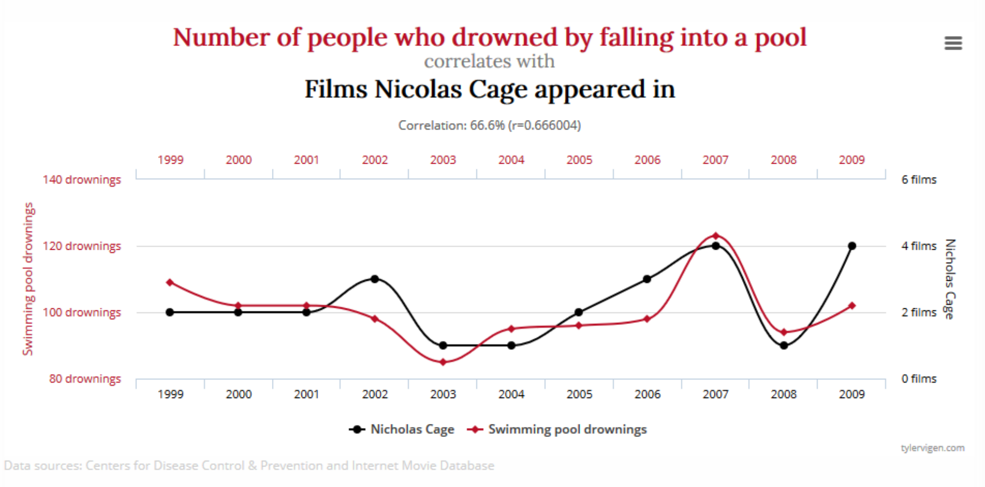
Data mining is a free-for-all where we collect lots of data, and we hope to find hidden patterns and (untested) insights that may not be obvious, or even make sense!

If data mining is used, it should ideally follow up with new data and testing, but many practitioners fail to do this.

Dr. Nancy Wertheimer at University of Colorado used data-mining to assert children living near power lines were 2-3 times as likely to develop leukemia. After decades of peer research, nobody was able to confirm any such correlation much less causality.

[*https://www.nytimes.com/1997/07/03/us/big-study-sees-no-evidence-power-lines-cause-leukemia.html*](https://www.nytimes.com/1997/07/03/us/big-study-sees-no-evidence-power-lines-cause-leukemia.html)

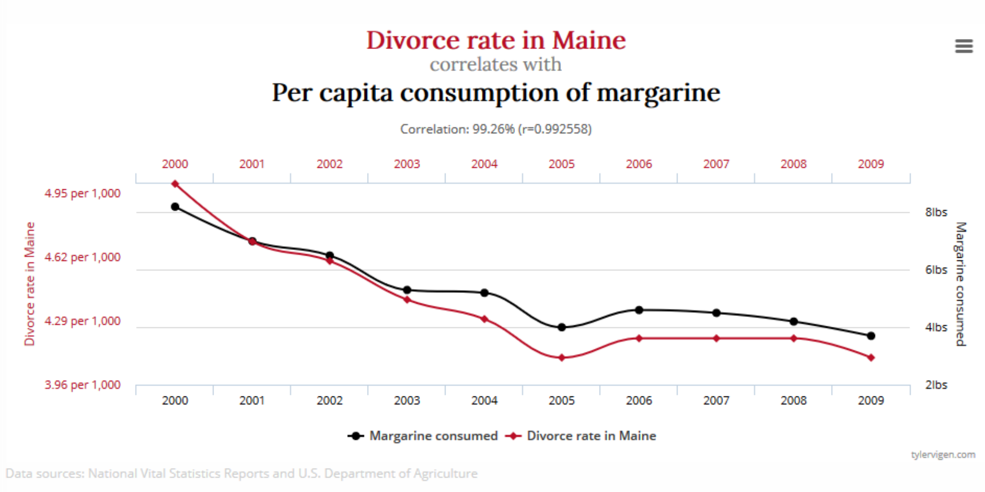
Correlation is Causation?



Number of people who drowned by falling into a pool
correlates with
Films Nicolas Cage appeared in

Correlation: 66.6% (r=0.666004)

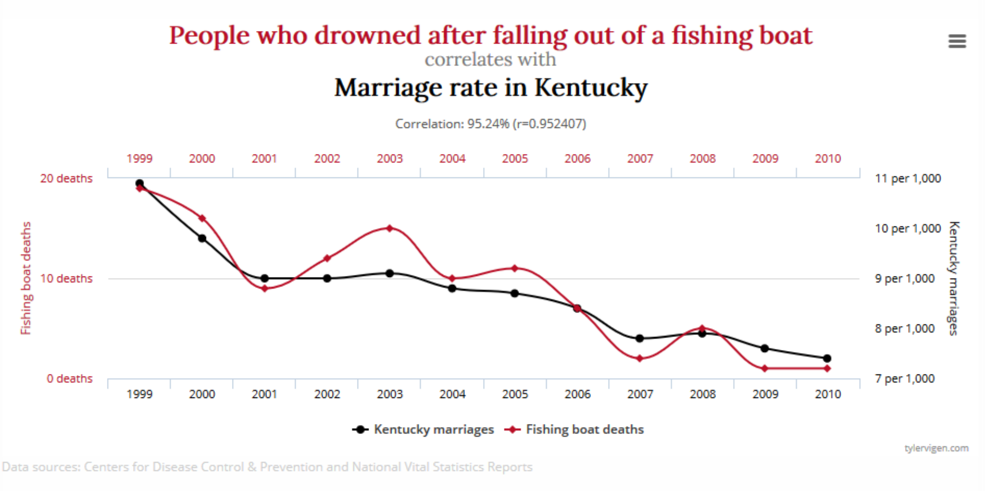
Data sources: Centers for Disease Control & Prevention and Internet Movie Database



Divorce rate in Maine
correlates with
Per capita consumption of margarine

Correlation: 99.26% (r=0.992558)

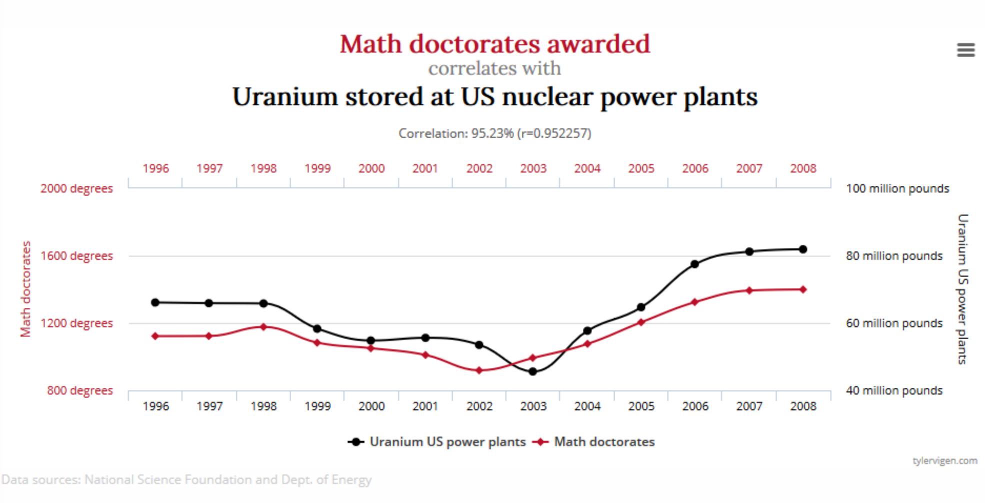
Data sources: National Vital Statistics Reports and U.S. Department of Agriculture



People who drowned after falling out of a fishing boat
correlates with
Marriage rate in Kentucky

Correlation: 95.24% (r=0.952407)

Data sources: Centers for Disease Control & Prevention and National Vital Statistics Reports



Math doctorates awarded
correlates with
Uranium stored at US nuclear power plants

Correlation: 95.23% (r=0.952257)

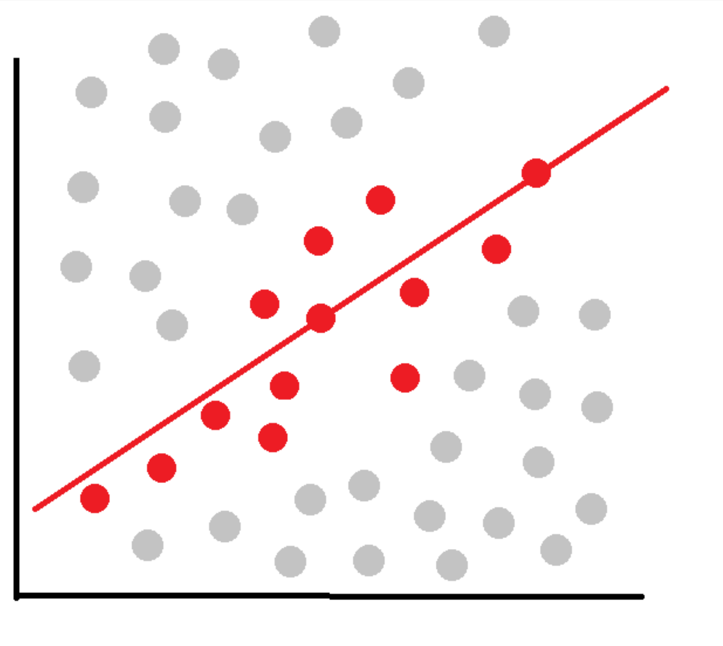
Data sources: National Science Foundation and Dept. of Energy

http://www.tylervigen.com/spurious-correlations

P-Hacking

P-hacking is cherry-picking models and data that produces a desired result rather than a realistic one.

- P-hacking is data mining for a p-value of less than .05
- Undisciplined and pressured practices are often the cause.
- Simply choosing a model because "it looks significant" or "solves my objective", rather than challenging it, is a subtle and easy way to p-hack.



This can lead to an inflation of false positives, where our model becomes too optimistic about an outcome but performs poorly in the wild.

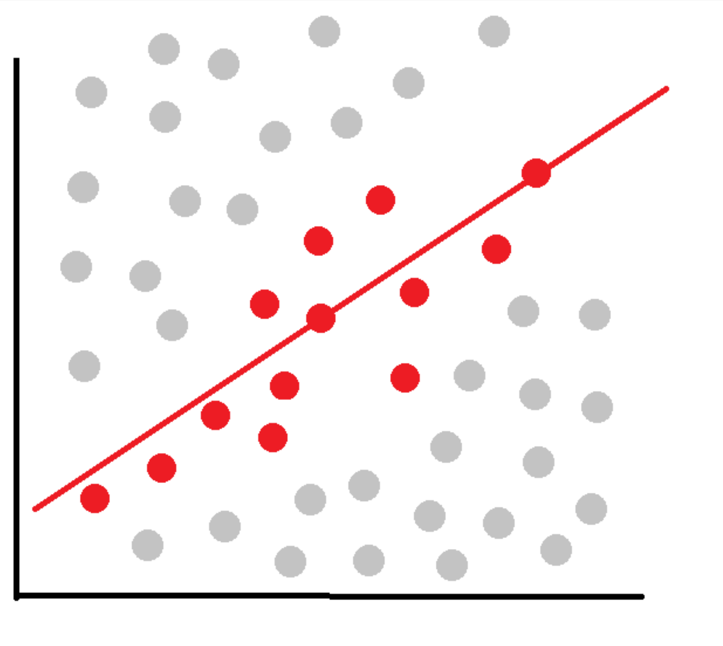
P-hacking is allegedly responsible for the replicability crisis[1], and is arguably made worse with the availability of data and machine learning.

[1] Ioannidis JP. Why most published research findings are false. PLoS Med. 2005;2(8):e124. doi:10.1371/journal.pmed.0020124

P-Hacking

Examples of P-Hacking:

- Collecting *just enough* data to get a desired result.
- Removing inconvenient data as “outliers” or “noise”
- Shopping for more than one dependent variable
- Dividing data into sub-groups, and focusing on one group
- Shopping model parameters that give the right result
- Using random seeds that produce desired outcomes



Motivations for P-Hacking:

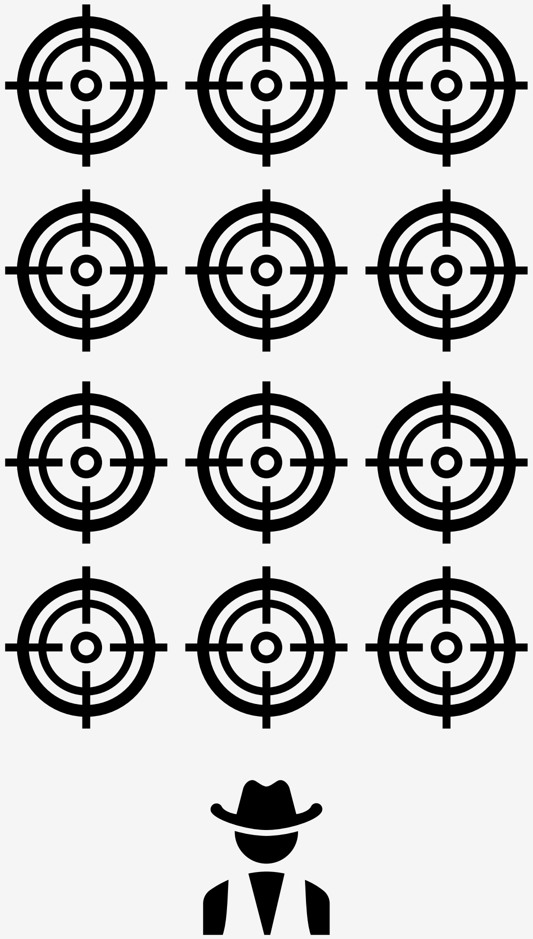
- Research pressure: “No paper, no funding”
- Job pressure: “Our client wants to see a model that predicts 10% savings in transportation costs”
- Startup pressure: “Our potential investors want a demonstration, so find a dataset that will produce good results.”

Is P-hacking malicious and deceptive? Not usually! It is often human nature operating under pressure (and even career survival).

Alleviating Data Mining Concerns

Here are a couple of things you can do to alleviate problems with data mining, Texas Sharpshooter Fallacy, and spurious correlations.

- Analyze your data and identify rational, sensible explanations.
- Apply scientific method: challenge your hypotheses by using fresh, new data.
- Seek out confounding variables, or causal variables that are omitted, which happen to correlate with collected variables.
- Utilize cross-fold validation and common sense while avoiding p-hacking, which we will talk about in later slides.



Exercise: Data Mining Gone Wild?

The next slide contains a “percent change in average travel” heat map.

It appeared in the New York Times during the COVID-19 pandemic, when stay-at-home advisories and restrictions were put in place.

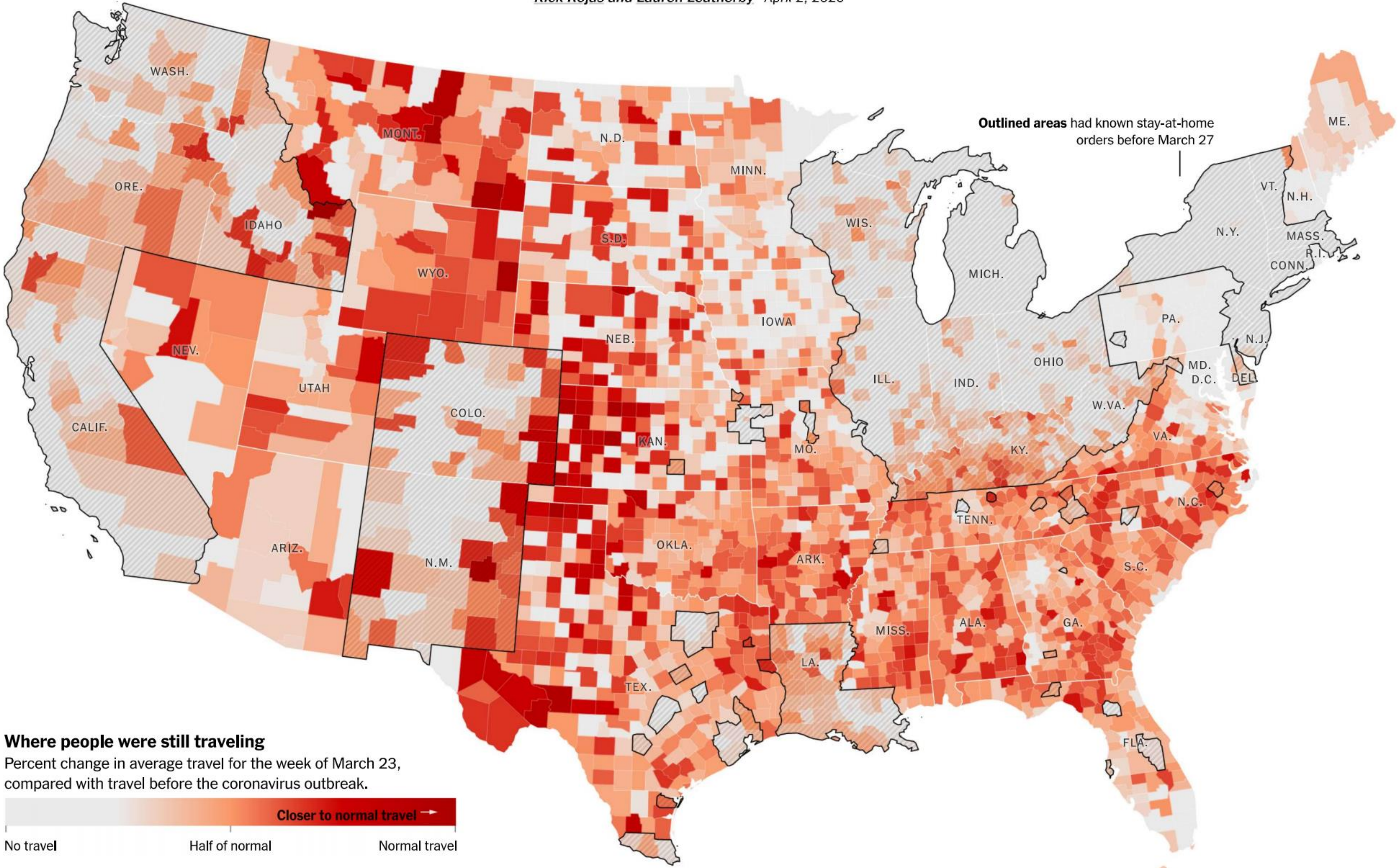
The idea behind these orders was to limit spread of the virus by having people stay at home rather than travel and be out in public.

What observations do you have about this heat map? What conclusions can you get from it?

SOURCE: https://www.nytimes.com/interactive/2020/04/02/us/coronavirus-social-distancing.html

Where America Didn’t Stay Home Even as the Virus Spread

By James Glanz, Benedict Carey, Josh Holder, Derek Watkins, Jennifer Valentino-DeVries, Rick Rojas and Lauren Leatherby April 2, 2020



Outlined areas had known stay-at-home orders before March 27

Where people were still traveling

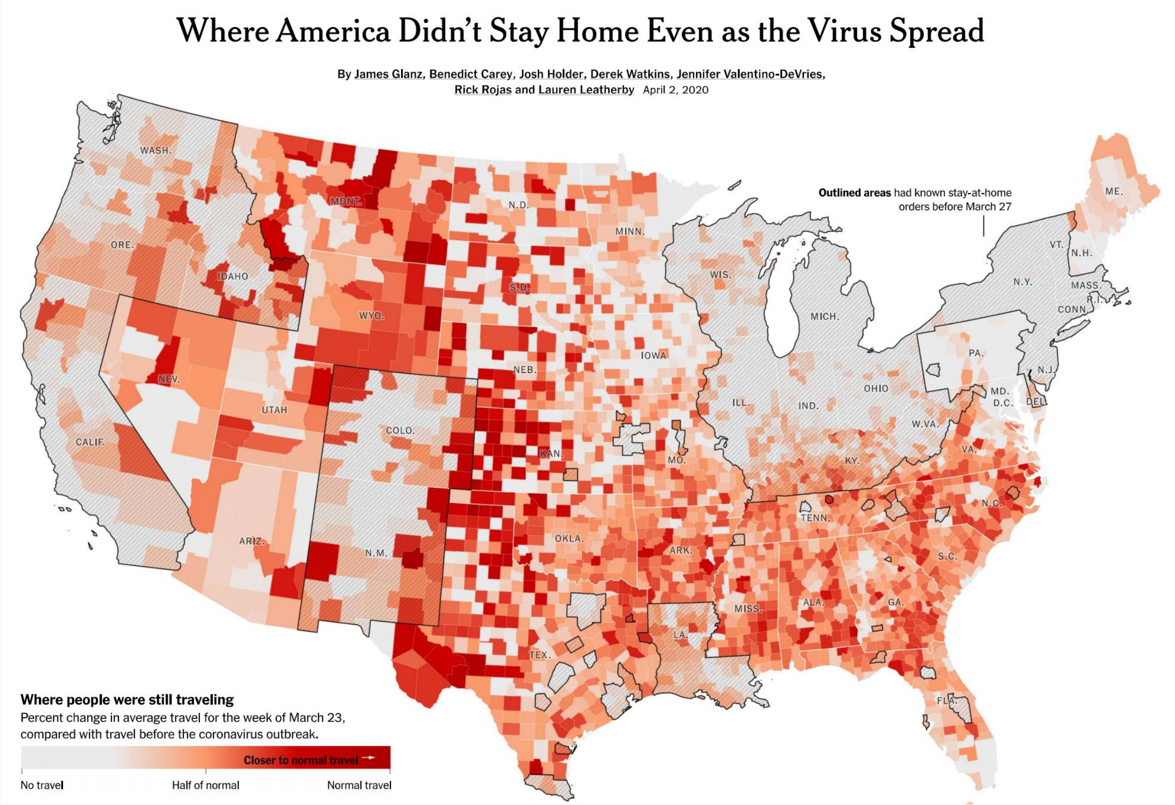
Percent change in average travel for the week of March 23, compared with travel before the coronavirus outbreak.

Exercise: Data Mining Gone Wild?

The purpose of this map is to show movement in a time people should not be moving, and while the premise of this heat map is interesting, there are some oddities:

- Why is the "percent change in average travel" the chosen variable for this heat map? Is this metric confusing and problematic?
- Why is population density ignored in the averages? A population of 5 who traveled 30 miles total (avg of 6 miles) would be more punished than a population of 150,000 who traveled 300,000 miles (avg of 2 miles).
- Is infrastructure proximity a confounding factor? Wouldn't rural and low infrastructure areas require more driving distance to get to a grocery store? Wouldn't delivery services be less available?
- Is it possible rural areas had more movement to supply grocery stores, and doing essential work? During the pandemic, demand for groceries and stockpiling were high.
- **Why are COVID-19 cases not being shown? Is that not the outcome variable we are concerned about in the first place?**

"Percentage Change of Average Travel" vs COVID-19 Cases



Where America Didn't Stay Home Even as the Virus Spread

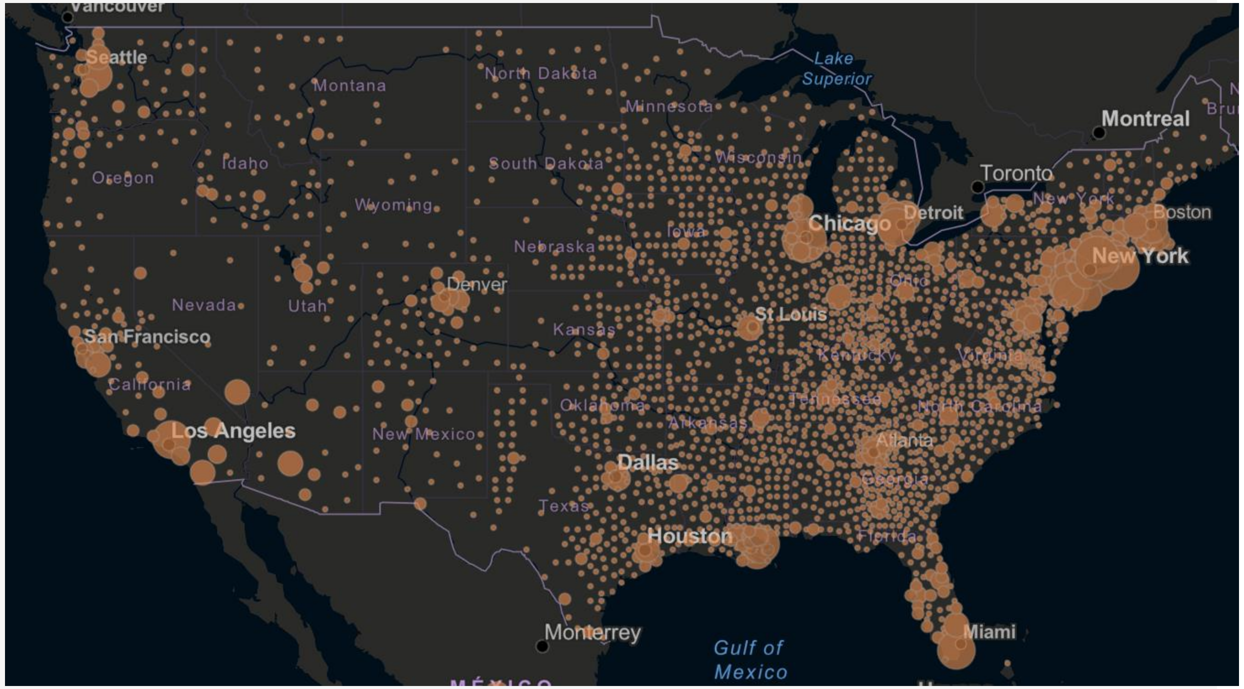
By James Glanz, Benedict Carey, Josh Holder, Derek Watkins, Jennifer Valentino-DeVries, Rick Rojas and Lauren Leatherby April 2, 2020

Outlined areas had known stay-at-home orders before March 27

Where people were still traveling

Percent change in average travel for the week of March 23, compared with travel before the coronavirus outbreak.

% change of average travel on week of 3/23/2020



COVID-19 cases on 4/6/2020

Exercise: Data Mining Gone Wild?

Just eyeballing these two maps, it is easy to see a possible negative correlation between these two metrics.

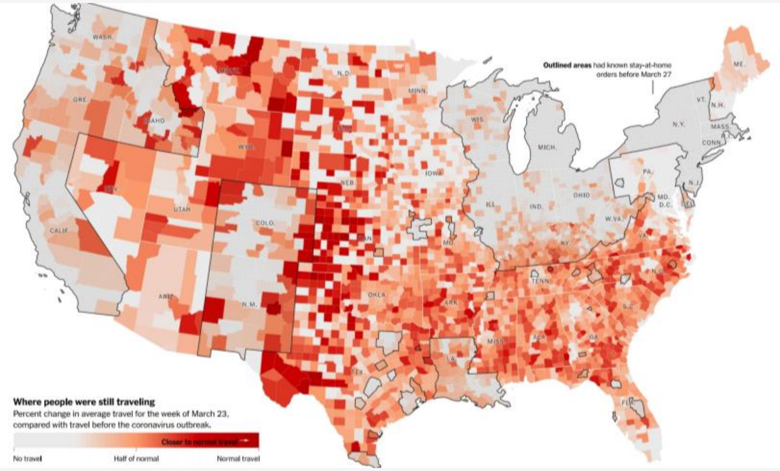
Does this mean traveling more reduces COVID-19 cases? Given this data, this would be the machine's conclusion!

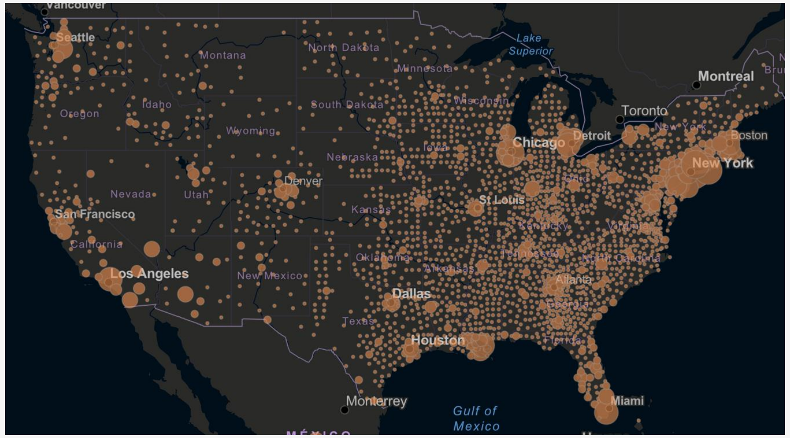
Correlation is not causation, and what we are probably seeing here is due to rural elements being a confounding factor.

So why did the New York Times focus on "percent change of average travel" as if it is a relevant driver to COVID-19 cases, when the opposite seems to be true?

We may never know, perhaps they were looking for "interesting" correlations and lost sight of the objective (what causes COVID-19 cases?)

If that was the case, this truly was the product of data-mining.


Outlined areas had known stay-at-home orders before March 27
Where people were still traveling
Percent change in average travel for the week of March 23, compared with travel before the coronavirus outbreak.
Closer to normal travel →
No travel
Half of normal
Normal travel
WASH.
ORE.
IDAHO
CALIF.
UTAH
ARIZ.
WYO.
COLO.
N.M.
N.D.
MINN.
NEB.
IOWA
KAN.
OKLA.
TEX.
MO.
ARK.
LA.
MISS.
ALA.
GA.
TENN.
KY.
ILL.
IND.
OHIO
MICH.
WIS.
W.VA.
VA.
N.C.
S.C.
FLA.
PA.
N.Y.
N.J.
MD.
D.C.
DEL.
CONN.
MASS.
R.I.
VT.
N.H.
ME.


Vancouver
Seattle
Montana
North Dakota
Lake Superior
Minnesota
Montreal
Oregon
Idaho
South Dakota
Wisconsin
Toronto
Wyoming
Chicago
Detroit
Boston
New York
Nebraska
Nevada
Utah
Denver
St Louis
San Francisco
Kansas
California
Oklahoma
Los Angeles
New Mexico
Dallas
Texas
Houston
Monterrey
Gulf of Mexico
Miami