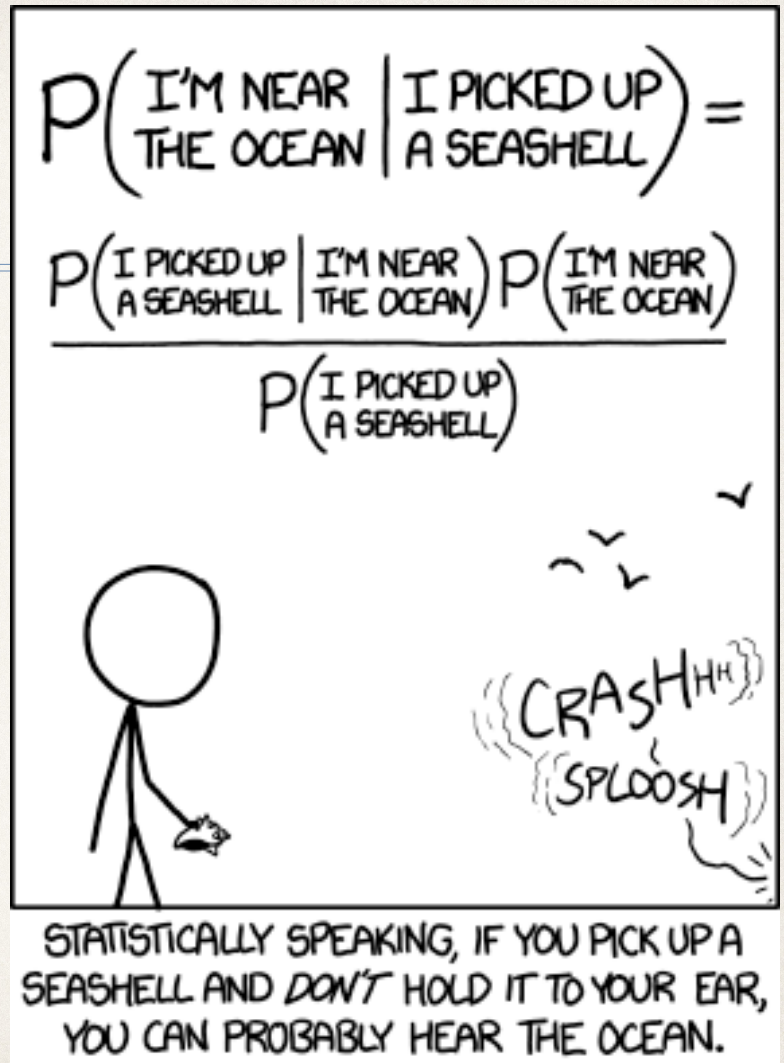
Probability Distributions

Statistics and Data Science

Spring 2025



http://xkcd.com/1236/

Goals for today: you should be able to...

- **lecture 10/11 notebook:**
  - Utilize robust statistics
  - Make more complicated Monte Carlo simulations
  - Interpret error estimates
  - Use properties of the *t* distribution to construct accurate confidence intervals
  - Explain the covariance statistic and how it relates to error estimates

Review: statistics we've been talking about

Estimators of location of data:

- Mean (`np.mean`)
- (Inverse-Variance) Weighted Mean (`np.average`)
- Mode (`mode2`)
- Median (`np.median`)

Estimators of spread of data:

- Sample Standard Deviation (`np.std`)
- Avg. Absolute Deviation
- Median absolute deviation (`scipy.stats.median_abs_deviation)`
- Interquartile Range (IQR, `scipy.stats.iqr`)

Our focus for much of today is ***robust*** statistics: ones that give meaningful results even when data is not drawn from a Normal distribution

Robust statistics

- A variety of statistics have been developed especially for their robustness.
- An example is the ***Hodges-Lehmann estimator of the mean***:
  - median $(x_i+x_j)/2$
    - where the median is calculated over all pairs (i,j), allowing duplication.
- This requires calculating the median of N^2 values for N data points, so is considerably slower than the ordinary median, but has >90% ARE. I have implemented it in the notebook:

hlmean code

```
def hlmean(data,nsamp=-1):
    ndata=len(data)

    # if the number of samples has not been provided, set it to 50*the size of the
data array

    if nsamp < 0:
        nsamp=50*ndata
    nsamp=int(nsamp)
```

hlmean code

```
def hlmean(data,nsamp=-1):
...

# create resampled version of original data
    newdata = np.random.choice(data,size=(nsamp,2))

    # average x1 + x2 from each random draw
    mn = (newdata[:,0]+newdata[:,1])/2

    # calculate the median of the averages
    return(np.median(mn))
```

Trimmed means and standard deviations

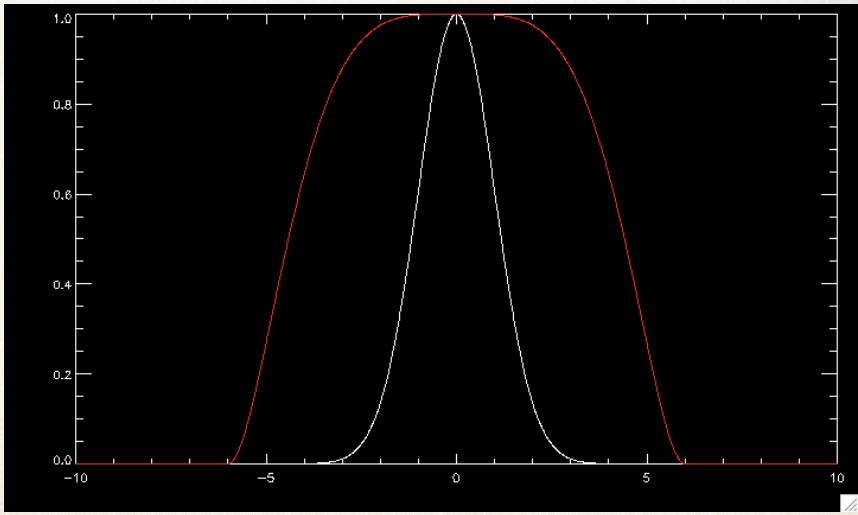
- A second common thing to do is to take an $\alpha\%$ ***trimmed mean*** (or ***trimmed standard deviation***):
  1) sort all the data by its value
  2) remove the lowest $\alpha\%$ and highest $\alpha\%$ of the data
  3) calculate the mean or standard deviation of the rest
- For the trimmed mean/std. dev., you can use `np.percentile()` to get the limits, and `scipy.stats.tmean()` or `scipy.stats.tstd()` to do the calculation. For a Gaussian distribution, the 10% trimmed standard deviation will on average be $1/1.49\ \sigma$.
- `scipy.stats.mstats.trimmed_mean()` and `scipy.stats.mstats.trimmed_std()` can optionally take the fraction to trim as inputs

Trimmed means and standard deviations

- Another related technique is sigma-clipping:
  - `scipy.stats.sigmaclip()` will yield a new array with >nσ outliers iteratively thrown out; then you can use the results with `np.mean`, `np.std`, etc.
- An alternative is ***winsorizing***: in that case, the lowest trimmed values are replaced by repeating the lowest non-trimmed value, and the highest trimmed value is replaced by repeating the highest non-trimmed value.
  - `scipy.stats.mstats.winsorize()` will yield a version of an array that is winsorized at the fractions (numbers between 0 & 1, not really percentiles) provided with the limits keyword (e.g., to winsorize 10% at each end, use limits=(0.1,0.1)).

Biweight statistics

- A third common robust statistic is the biweight (a.k.a. the bisquare or Tukey's biweight)
  - Has both high robustness and high efficiency for a variety of distributions.
- Based on an initial estimate of the mean and sigma, each data point is given a weight $(1-\Delta^2)^2$, where $\Delta=(x-\langle x\rangle)/6\sigma$, and we take $\Delta=1$ anywhere that $\Delta>1$. The weight for a unit Gaussian is plotted in red at right.



- A biweighted mean is implemented in Python as `astropy.stats.biweight_location`, while `astropy.stats.biweight_scale` calculates a biweighted estimator for standard deviation.

What's the optimal statistic to use?

- In physical and astronomical situations, we often have data that are not (or may not) be Gaussian.
- A simple example is measuring both the mean redshifts (recession velocities, in km/sec) and velocity dispersions (standard deviation of velocities, in km/sec) of galaxy clusters.
  - The redshift of a cluster provides an estimate of its distance from us, vital for interpretation
  - The velocity dispersion (i.e., RMS velocity relative to the cluster center) of a cluster provides a measure of its potential well depth: $\sigma^2 \propto GM/R$, and $M \propto \sigma^{3\text{-}4}$, for a cluster in equilibrium
  - For a discussion of robust statistics for galaxy clusters, see Beers, Flynn, & Gebhardt 1990.

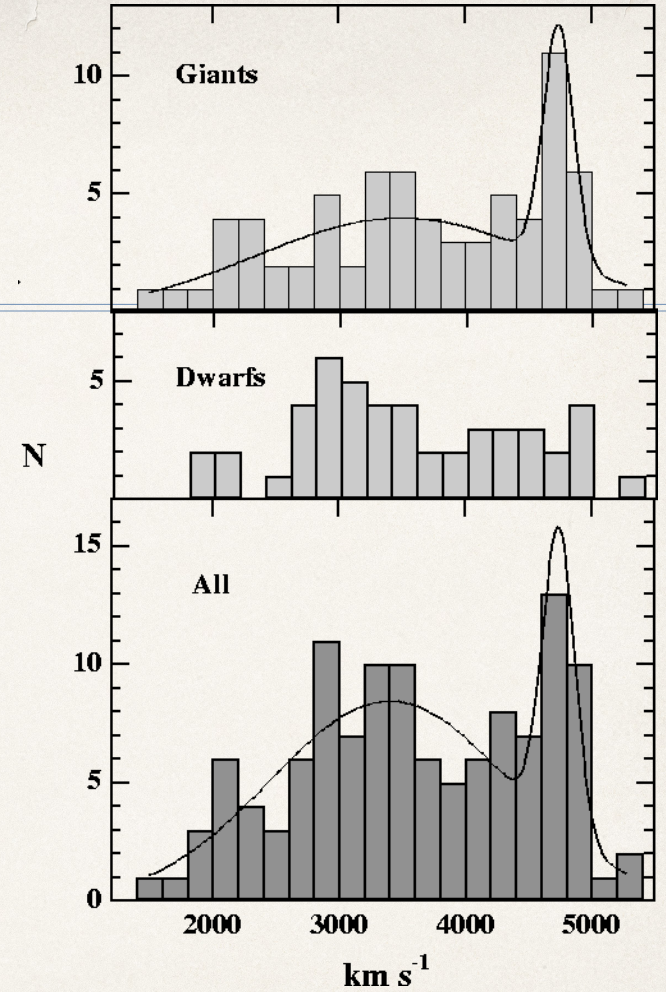
What makes this difficult?

1) Measuring redshifts for galaxies in distant clusters is hard; best case, we might have 100 spectra of galaxies around a given cluster, worst case ~5.

2) Clusters tend to be found near both other clusters and non-member galaxies

- there is a nonnegligible chance there will be another cluster within 5-10 Mpc (=~350-700 km/sec, given the Hubble-Lemaitre Law), while the velocity dispersion of a cluster can reach 1000 km/sec

3) Clusters often are still undergoing mergers with other clusters and have not reached equilibrium

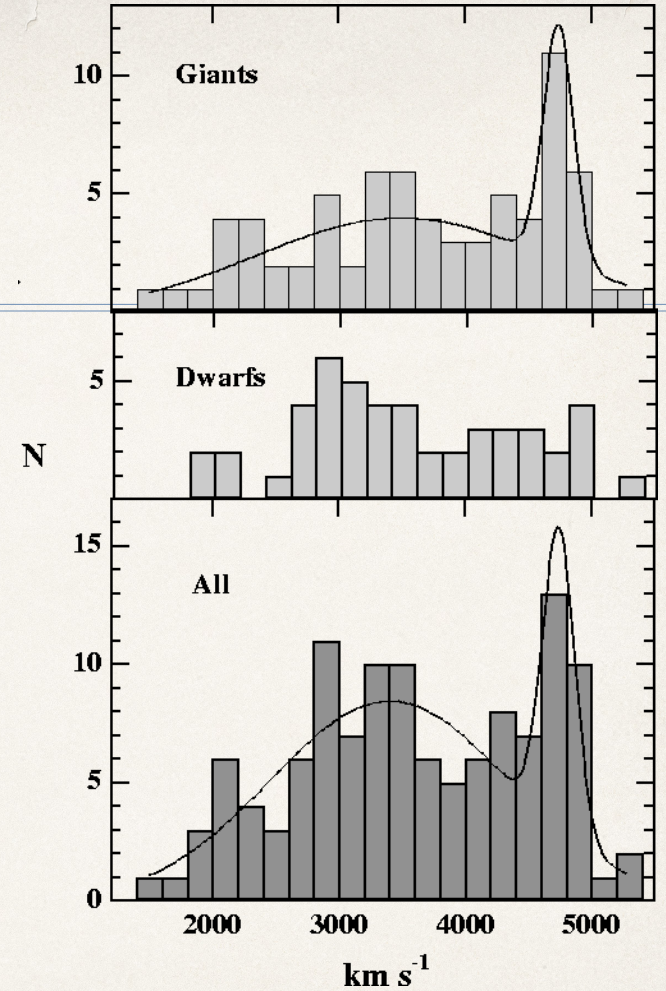


Stein, Jerjen et al. 1997

Setting up a test case

- We've generated uniform or Gaussian-distributed random numbers before. Now we want to simulate the case where we have data that may be drawn from **two** distributions: drawing from one with probability $f_{outlier}$, and from the other with probability $(1-f_{outlier})$.

- For the main cluster, let's take the distribution to be a Gaussian with mean 3150 km/sec and sigma 930 km/sec
- For the outliers, we will use a Gaussian with mean 4750 km/sec and sigma 200 km/sec.



Setting up a test case

- We want to draw from the distribution $v \sim (1\text{-}f_{\text{outlier}})N(3150,930^2)+ f_{\text{outlier}}\, N(4750,200^2)$.
- Let's do `nsims=50_000` simulations of a possible set of observations, with `ndata=100` redshifts in a set:

```
nsims=int(5E4)
ndata=100
```

- For a default, let's take the outlier fraction to be 0.1 :

```
foutlier=0.1
isoutlier=random.rand(nsims,ndata) < foutlier
fakedata=(1-isoutlier)*(random.randn(nsims,ndata)*930.+3150) \
 + (isoutlier)*(random.randn(nsims,ndata)*200.+4750)
```

Checking the results

- **Now plot a histogram of the full set of fake data; and then choose one single simulation and plot the histogram for it, with a bin size of 100 km/s.**

- **Note**: To plot histograms of a multi-dimensional array, use `np.ravel(arrayname)`, not just `arrayname`, in the call to `plt.hist`. `arrayname.ravel()` or `arrayname.flat` would also work.

Your tasks: Homework 3, due a week from Friday

1) For each of the location estimators we have considered -- mean (`np.mean`), median (`np.median`), mode (`mode2`; use a bin size of 50 km/s), Hodges-Lehmann mean (`hlmean`), 10% trimmed mean (`tmean`), and biweight mean (`biweight_location`), evaluate:

A) how efficient is each estimator, for a case with ndata=100 and foutlier=0; i.e., compare the standard deviations of the value returned by the estimator, amongst all the simulations, for perfect Gaussian data. Which yields the most accurate results (with the smallest spread **around the true value**)? You do not need to actually calculate ARE here, just compare spreads (i.e., the standard deviation of the value from each location estimator) for each.

B) Now find the bias (i.e., average offset from true mean) and spread of each estimator, for *ndata*=10 (typical case for distant clusters) and *ndata*=100 (an ideal intensive-study case), with *foutlier*=0.1 . **Explain which estimator do you think we should use in each case, and why?**

For bias, look at the mean value of (estimated location - 3150); for the spread, look at `np.std`(estimated value).

Your tasks: Homework 3, part 2

2) Set up simulations for a slightly different case: now assume outliers have a uniformly-distributed velocity between 0 and 6500 km/sec, and that we want to measure the velocity dispersion, rather than the mean velocity, of the cluster. We have a variety of estimators for dispersion: sample standard deviation (`np.std`), average absolute deviation (which we implemented), MAD (`median_abs_deviation`), biweight standard deviation (`biweight_scale`), IQR (implemented by you) & 10% trimmed standard deviation (using `scipy.stats.tstd`). Evaluate:

A) the efficiency of each estimator, with ndata=100 and foutlier=0; i.e., compare the standard deviations of the value returned by the estimator, amongst all the simulations, for perfect Gaussian data. Which yields the most accurate results (with the smallest spread **around the true value, i.e. smallest <(estimated - true)^2>**)? **Remember to apply the normalization corrections** to get the equivalent of a Gaussian sigma from non-std. deviation measures like MAD before comparing.

B) both the spread and bias of each estimator (comparing to the correct value of 930), for *ndata*=10 (typical high-z case) and *ndata*=100 (ideal intensive-study case), with *foutlier*=0.1 . **Explain which estimator do you think we should use in each case, and why?**

Homework 3 (contd).

- Homework 3 is due a week from Friday. I suggest you compare results with each other or otherwise collaborate.
- I also suggest you use print commands to make your outputs clearly understandable; e.g.:

```
means=np.mean(fakedata,axis=1)
print(f'ordinary mean: bias {np.mean(means)-3150}, \
    spread {np.std(means)}')
```

- Remember: for documentation on Python routines, use ? before the name.

Describing errors

- Because of the Central Limit Theorem, it should be a decent assumption that the distribution of measurements of the *mean* of some quantity should be Gaussian.
- If we are measuring a Gaussian-distributed quantity, then we can describe the expected results fully by just specifying the mean and true σ of the corresponding Gaussian; we might write

  mean ± *deviation*

  to describe such a result (e.g., 5 ± 2) , where *mean* is our estimated mean and *deviation* is some description of the width of the Gaussian.
- There is no one standard in the literature for *deviation*. <u>Typically</u>, it will be equal to σ or 2σ (where σ is the standard error); or will be defined such that 68% or 95% of the probability in a Gaussian would be between *mean-deviation* and *mean+deviation*.

Interpreting errors

- Typically, what we'd like to know is what true values of some parameter (which we are using the mean of measurements to determine, for instance) are possible.
- For a Bayesian analysis, this is straightforward; we can define the smallest interval/region of parameter space that contains X% of the probability as the X% high density region or X% ***credible interval***; we would then believe that the true value should fall in that region with X% probability.
- Suppose we measure a mean m and sample standard deviation of the mean $\hat{\sigma}_{\bar{x}}$ from some data, drawn from a distribution with true standard deviation σ. What would we conclude about the true mean μ, in the Frequentist view?

Considering the possibilities

- Frequentist statistics focuses on what will be observed, given an assumed truth.
- Let's consider 2 possible ideas of how we might interpret measurements:
  - 68.3% of the time we do an experiment like this, the true mean will lie between m-$\hat{\sigma}_{\bar{x}}$ and m+$\hat{\sigma}_{\bar{x}}$, where $\hat{\sigma}_{\bar{x}}$ is is the sample standard deviation of the mean determined from the data, $\frac{\sigma_x}{\sqrt{n}}$
  - 68.3% of the time we do an experiment like this, the true mean will lie between m-$\sigma_{\bar{x}}$ and m+$\sigma_{\bar{x}}$, where $\sigma_{\bar{x}}$ is the standard error we would calculate with perfect knowledge of the distribution, $\frac{\sigma}{\sqrt{n}}$

Considering the possibilities

- Let's see what's right!

```
nsims=int(1E5)
ndata=10
data=random.randn(nsims,ndata)
```

- **What should we expect the mean & sigma of the data array to be?**

Setting things up

```
means = np.mean(data, axis = 1)
sample_std=???  # we want the standard deviation of each set of 10
```

Note: just like we can calculate means along one axis of an array with the axis keyword, the same keyword works with np.std !

- We will also need to calculate the standard deviation of the mean:

```
sample_serr=sigmas/np.sqrt(ndata) # estimated standard error
true_serr=1/np.sqrt(ndata)   # standard error calculated
                             # from true sigma, =1
```

- **Now plot histograms of means and of sample_std. Are they both Normally distributed (Gaussian)?**

Testing interpretations

`means`: estimated means from each sim.

`sample_serr`: sample std. deviation of the mean from each sim.

`true_serr`: standard error determined by knowing true sigma

Write code to determine (for both N=10 and N=100):

- What fraction of the times when we do an experiment like this does the true mean lie between `means-sample_serr` and `means+sample_serr` ?

  ???

- What fraction of the times when we do an experiment like this does the true mean lie between `means-true_serr` and `means+true_serr`

  ???

Results

- 68.3% of the time we do an experiment like this, the true mean will lie between m-$\sigma_{\bar{x}}$ and m+$\sigma_{\bar{x}}$, where $\sigma_{\bar{x}}$ is the standard error we would calculate with perfect knowledge of the distribution, $\frac{\sigma}{\sqrt{n}}$
- However, the first option (68.3% between m-$\hat{\sigma}_{\bar{x}}$ and m+$\hat{\sigma}_{\bar{x}}$) wasn't too far off, and was more correct when $\hat{\sigma}_{\bar{x}}$ was determined better.
- We could say that the interval [m-$\sigma_{\bar{x}}$, m+$\sigma_{\bar{x}}$] is a ***68% confidence interval*** for the true value of μ; we could similarly define a 90%, 95%, or whatever confidence interval.
  - I.e.: 68% of the time when we make a confidence interval this way, the true value will lie within it
- Alternatively, we could say (for a 95% confidence level) that any value of the parameter μ outside this interval is significantly different from the observed value at the 5% level.

Coverage

- A related, but different, number is the ***coverage*** of an interval. If X% of the time the true value our statistic is intended to determine lies within a given interval, we'd say it has X% coverage (or it is an X% confidence interval).
- **If** we know the true standard deviation for a population which we **know** is normally-distributed (or we are in the large-N limit so the CLT applies), the interval $[m-\sigma_{\bar{x}}, m+\sigma_{\bar{x}}]$ has 68.3% coverage.
- If either of those assumptions is incorrect, we could still construct that interval, but it wouldn't have the coverage we wanted.

Doing better

- We really don't want to have to know the **true** standard error to produce a confidence interval (it's generally easier to measure locations than spreads/errors). If we at least know the distribution we're dealing with (e.g., Normal), the problem is tractable.
- Let's define a (1-α)*100% confidence level by:

  probability ($u < Z < v$) = 1-α

  where Z is some unknown parameter, and u and v are quantities constructed from some set of data (i.e., they are statistics).

Doing better

- Now go back to the case of determining the true mean of a Normal distribution underlying some data. Consider the quantity:

$$t = \frac{(\bar{x} - \mu)}{\frac{\hat{\sigma}_x}{\sqrt{n}}}$$

- i.e., the deviation of the mean statistic from the true mean, in units of the *estimated* standard error.

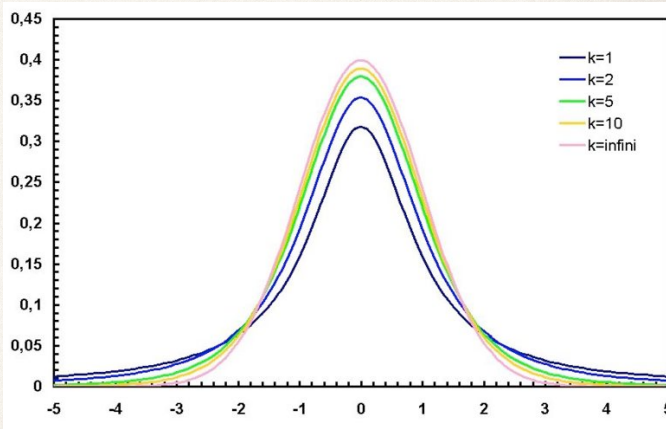
The t distribution

- $t = \frac{(\bar{x} - \mu)}{\frac{\hat{\sigma}_x}{\sqrt{n}}}$ is commonly known as Student's t

- For data drawn from a normal distribution, the PDF of t turns out to be:

$$f(t) = \frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\nu\pi}\,\Gamma(\frac{\nu}{2})}\left(1 + \frac{t^2}{\nu}\right)^{-(\frac{\nu+1}{2})},$$

where ν is the number of *degrees of freedom* (generally the # of independent datapoints minus the # of parameters) and Γ(y) is the gamma function (=(y-1)! if y is an integer).



- *f(t)* has broader tails than a Normal distribution, but converges to $N(0,1)$ as ν becomes large.

Going from data to proper confidence intervals

- Since $t = \frac{(\bar{x} - \mu)}{\frac{\hat{\sigma}_{\bar{x}}}{\sqrt{n}}}$, the probability that μ is larger than $\bar{x} + A\frac{\hat{\sigma}_{\bar{x}}}{\sqrt{n}}$ must be the same as the probability that t is greater than A (and similarly for $\bar{x} - A\frac{\hat{\sigma}_{\bar{x}}}{\sqrt{n}}$)
- Recall that the cumulative density function (CDF) $F(x)$ for some probability distribution $f(x)$ is the integral from -∞ to x of f
- so for our example case, the fraction of the time that μ will be between m-$\sigma_{\bar{x}}$ and m+$\sigma_{\bar{x}}$ will be $F(1)$-$F(-1)$, where F is the CDF for the t distribution

Going from data to proper confidence intervals

- In Python, `scipy.stats.t.cdf(x,df)` returns the CDF of the *t* distribution for `df` degrees of freedom evaluated at the value `x`.
- Let's check the cases we did before, averaging 10 or 100 values: how likely should it be that $m-\sigma_{\bar{x}}<\mu<m+\sigma_{\bar{x}}$?

```
print(stats.t.cdf(1.,9)-stats.t.cdf(-1.,9))
print(stats.t.cdf(1.,99)-stats.t.cdf(-1.,99))
```

- **How do these values compare to the results of our simulations?**

Going from data to proper confidence intervals

- We really want to figure out the value A such that $m-A\sigma_{\bar{x}} < \mu < m+A\sigma_{\bar{x}}$ is a $(1-\alpha)*100\%$ confidence interval. Then (and only then) we can turn observed means and standard errors into accurate confidence intervals.
- In Python, `scipy.stats.t.ppf(p,df)` returns the value of x such that the probability that a t-distributed variable is **less than** x is `p`, for `df` degrees of freedom

Going from data to proper confidence intervals

- For instance, a 68.3% confidence interval matching +/- 1 σ for a Gaussian should cover the range p=0.1587 -- p=0.8413, while a 95% confidence interval would range from where p=0.025 to where p=0.975.
- We know the t distribution is symmetric about 0, so we just need to calculate one of these.

```
cutoff9 = stats.t.ppf(0.8413,9) # 68.3% limit for 9 DOF
cutoff99 = stats.t.ppf(0.8413,99)
```

We can check the results, e.g.:

```
print(stats.t.cdf(1.0585,9)-stats.t.cdf(-1.0585,9))
```

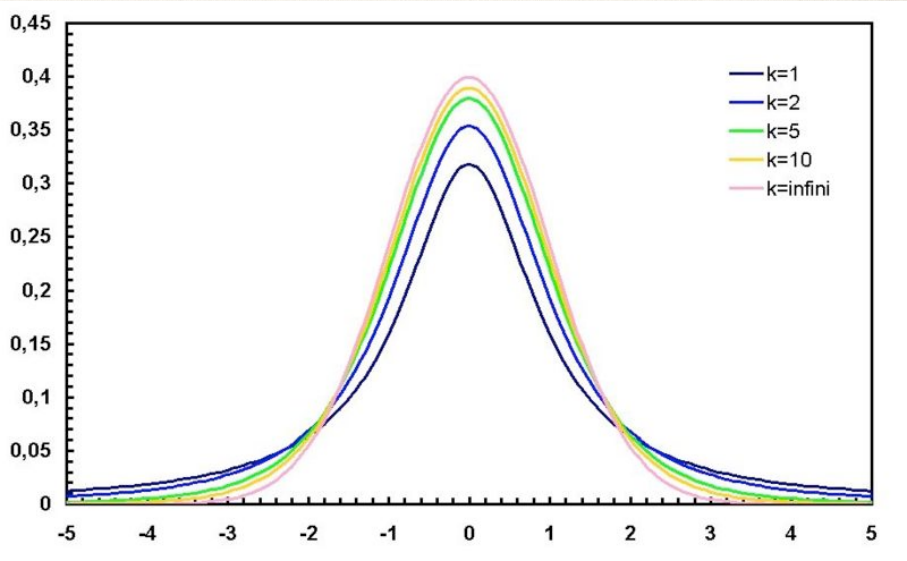
So the interval $[m\text{-}1.06\ \sigma_{\bar{x}},\ m\text{+}1.06\ \sigma_{\bar{x}}]$ will contain the true value of μ 68.3% of the time we generate a dataset & determine m & $\sigma_{\bar{x}}$.

Going from data to proper confidence intervals

- This matters most for small n and small α:

```
for i in np.arange(2,20,2):
    print(f'DOF: {i:2d}   {stats.t.ppf(0.1587,i-1):.5f},
{stats.t.ppf(0.8413,i-1):5f}')
```

- For a Gaussian distribution, 68.3% of the probability is between -1σ and +1σ, 95% is between -1.96σ and +1.96σ, 99% is between -2.56σ and +2.56σ: compare your values to those!



Errors in the mean

- We previously found by experiment that the variance of the mean of n <u>independent</u>, Gaussian-distributed measurements will be σ^2/n, where σ^2 is the variance in a single measurement. What happens if we relax these assumptions?
- The variance in the mean is: $\sigma_m^2 = \mathbb{E}(<x> - \mu)^2 = \mathbb{E}(\frac{\Sigma x_i}{n} - \mu)^2$
- Exploiting that $\mathbb{E}(a+b) = \mathbb{E}a + \mathbb{E}b$ and the definitions of mean and variance, with a lot of manipulation, we can derive:

$$\sigma_m^2 = \frac{\sigma^2}{n} + n^{-2}\Sigma_{i \neq j}\, \mathbb{E}(x_i - \mu)(x_j - \mu)$$

  - **If** all the x_i are independent of each other, then $\mathbb{E}(x_i - \mu)(x_j - \mu) = 0$ if $i \neq j$. In that case, the error on the mean will tend to get better as $n^{-1/2}$, as we found before.

Covariant Errors

- We found: $\sigma_m^2 = \frac{\sigma^2}{n} + n^{-2}\Sigma_{i\neq j}\, \mathbb{E}(x_i - \mu)(x_j - \mu)$

- There are $n(n\text{-}1)\sim n^2$ pairs of i,j with $i\neq j$, so if $\mathbb{E}(x_i - \mu)(x_j - \mu)$ is roughly a constant for all i & j, then instead σ_m^2 will be roughly constant for large n - **the uncertainty in the mean wouldn't get better with more data**.

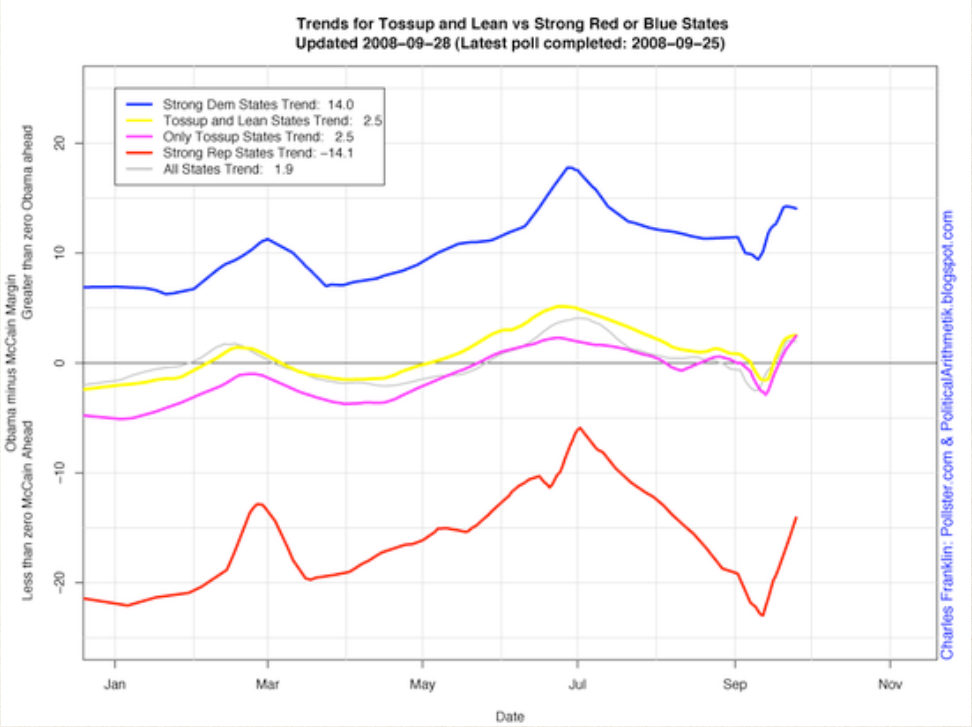
- We can define the covariance to be:

$$\mathrm{cov}[x_i, x_j] = \mathbb{E}(x_i - \mu_i)(x_j - \mu_j)$$

- Then $\mathrm{cov}[x_i, x_i] = \sigma_i^2$, and any 2 independent variables have 0 covariance.

An example

- Let's think about political polls.
- If there's some piece of national news (say, the national political conventions in a presidential election year), we'd expect the poll results in *all* states to move as a result.
- There is a *covariance* between how people respond to events in different states.



An example

- In some cases, covariance between statistics occurs because some data are in common - e.g., the trend in voter choices in the US as a whole had better reflect trends in the individual states (but with a different weighting).