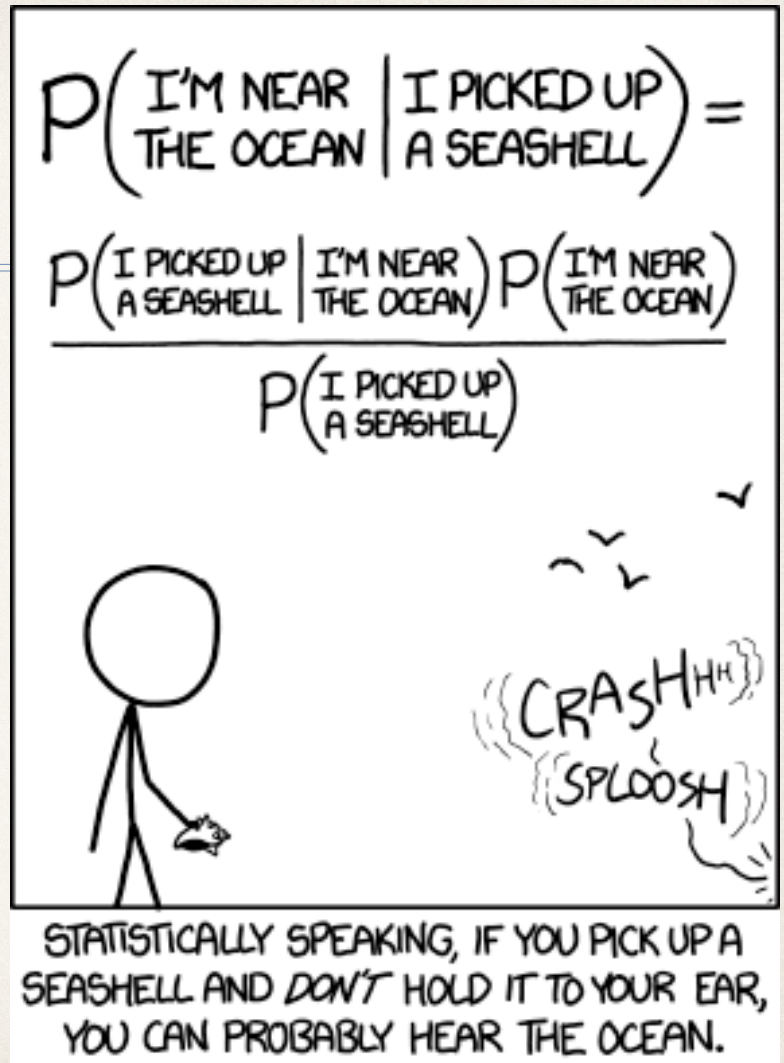
Probability Distributions

Statistics and Data Science

Spring 2025



http://xkcd.com/1236/

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

- **Lecture 9 notebook:**
  - Identify and apply major statistics (mean, mode, median, standard deviation, etc.)
- Explain what qualities we might desire in a statistic
- **lecture 10/11 notebook:**
  - Utilize robust statistics
  - Make more complicated Monte Carlo simulations
  - Interpret error estimates

Review: the sample standard deviation

- The Standard Deviation of an array is given by the Python function `np.std()`. You generally would want to call it with the keyword `ddof=1`, which calculates:

$$\sigma_s^2 = \left(\sum (x_i - \mu)^2 \right) / \ (N-1)$$

- σ_s is known as the ***sample standard deviation***, and corrects (to first order) for the fact that our estimate of the mean minimizes the squared deviation (and hence biases the second moment about it low)

Review: other measures of spread

1) The *average absolute deviation* or *average deviation*: **$< | x_i - <x> | >$**

- For a Normal distribution, the expectation value of this quantity is $\sqrt{(2/\pi)}$ times σ, or $0.7979 \times \sigma$

2) The *median absolute deviation*, or MAD: **median($| x_i$-median(x)$|$)**

- For a Normal distribution, the expectation value of this quantity is $0.6745 \times \sigma$

3) The *interquartile range*, or IQR:

IQR= 75th percentile value - 25th percentile value

= median of highest 50% of values - median of lowest 50% of values

- For a Normal distribution, the IQR = $1.349 \times \sigma$

Scale measures in Python

- **Try out `np.std()`:**

```
print( np.std(data),np.std(data,ddof=1) )
print( np.std(np.log(data)),np.std(np.log(data),ddof=1) )
```

- We need to do some work to calculate the average absolute deviation and normalize it to match sigma for a Gaussian:

```
normavgabsdev = np.mean(np.abs(data-data.mean()))/0.7979
mnlog = np.mean(np.log(data) )
normavgabsdev_log = np.mean(np.abs( np.log(data)-mnlog) )/0.7979
```

Rank-based measures

- We can also calculate MAD (and its normalization) by hand:

```
meddata=np.median(data)
normmad = np.median(np.abs(data-meddata))/0.6745
normmad_log = np.median(abs(np.log(data)-np.log(meddata)))/0.6745
```

- Alternatively, we can use `scipy.stats.median_abs_deviation()` with `scale='normal'` (NOT the default):

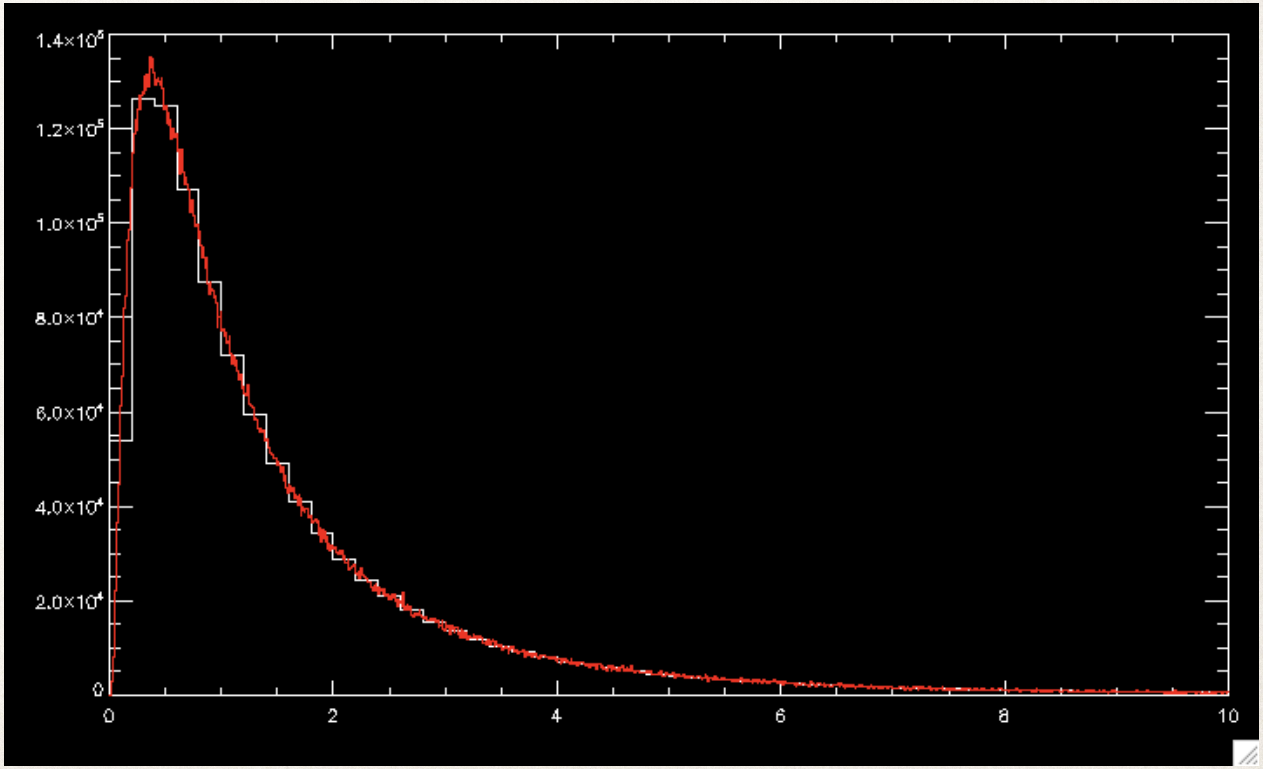
```
normmad_scipy = stats.median_abs_deviation(data,scale='normal')
```

- IQR requires us to use a new routine:

```
d25,d75 = np.percentile(data,[25,75])
normiqr = (d75-d25)/1.349
normiqr_log = (np.log(d75)-np.log(d25))/1.349
```

Results

- IF we normalize, all of these methods gave ~equal estimates for the true standard deviation for a Gaussian case (the log of our log-normal values)
- For the log-normal, the range is 0.89-2.17!
  - Compare to the true σ of the distribution = 2.16 !



Standard Deviation vs. Standard Error

- All of these methods estimate the **spread** of values that were drawn from some PDF.
  - The intrinsic spread will be the same no matter how many values we look at.
- Often, we are interested instead in **how accurately we have determined the mean** from some set of data: the "***standard error***".
  - In that case, the more data we have, the better-measured the mean should be.
- If we have 1 data point selected from *N(0,1)*, then that point will (of course) be spread around 0 as a Gaussian with sigma 1. What happens if we average N points all drawn from this Gaussian?

Averaging *n* data

- Let's try it, averaging 100 at a time:

```
nsims=int(1E5)
navg=100
data=random.randn(nsims,navg)
means=np.mean(data,axis=1)
```

- **Plot a histogram of the distribution of means, with bins 0.01 in size, over the range from -2 to +2**
- **Determine the standard deviation of the array of means**

Averaging n data

- What happens if we average 9 values at a time instead?

```
navg=9
data_9= ???
means_9=???
```

- **Overplot the histogram of the distribution of means, using plt.hist with the same binning as before.**
- **Determine the standard deviation of the array of means in each case**
- **Discuss: How does the scatter in the means scale?**

Results from averaging n data

- In each case, if we average n datapoints, the means are distributed as a Gaussian with the same mean as the true distribution (0) but spread :

$$\sigma_m = \frac{\sigma}{n^{1/2}}$$

- We could look at this as a consequence of the Central Limit Theorem:

  If you form averages M_n of samples of n from a population with finite mean and variance, then the distribution of $(M_n-\mu)/(\sigma/\sqrt{n})$ approaches a Gaussian with mean 0 and variance 1 as n goes to infinity.

- So the distribution of $M_n-\mu$ - which is the thing we just plotted (since $\mu=0$) - should be distributed as a Gaussian with mean 0 and variance σ^2/n, for large n.

The standard deviation of the mean

- In fact, the sum of 2 Gaussian-distributed variables will always be distributed as a perfect Gaussian, with $\sigma^2 = \sigma_1^2 + \sigma_2^2$ (where σ_1 and σ_2 are the standard deviations of the distributions the values x_1, x_2 are drawn from)
  - so the mean of n Gaussian-distributed variables will be distributed as a perfect Gaussian with variance $\sigma_{mean}^2 = \Sigma \frac{\sigma^2}{n^2}$(using the fact that $N(\mu, \sigma^2) = \mu + \sigma\, N(0,1)$).

- We call $\sigma_m = \frac{\sigma}{n^{1/2}}$ the ***standard deviation of the mean*** or the ***standard error***
  - It is the RMS deviation of the **mean** of n data from the **true mean** of the distribution they come from.

The standard deviation of the mean

- We would expect (in the frequentist view) that 95% of the time the **true mean**, μ, will lie in the interval $(\langle x\rangle - 2\,\sigma_m, \langle x\rangle + 2\,\sigma_m)$.* We can call that a ***95% confidence interval*** for μ.
- σ_m will **<u>always</u>** be smaller than (or equal to, for n=1) the sample standard deviation, which describes the spread of individual measurements
  - Instead, the standard error tells us how well we know the mean of the distribution
- The key thing to remember: as we acquire more data, **the standard deviation should not decrease**, as it describes the observed spread of individual values, but **our knowledge of the mean value does get better** from more data.

* IFF you know σ_m perfectly

Swimming in a sea of statistics

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`)

How do we determine the right statistic to use for our situation?

How should we choose amongst all these statistics?

- For data that really is distributed as a Gaussian, it is possible to show that the ordinary mean and sample standard deviation are the 'best' estimates of the true parameters μ and σ - for some definition of 'best'. What makes a statistic 'good' or 'better' than some other, anyway?

1) We'd like our statistics to be ***unbiased*** - i.e., to have an expectation value equal to the parameter of interest, not offset from it. For a Normal distribution, $\langle x \rangle$ is unbiased, while σ_s has a modest (max. -20%) bias for small N.

2) We'd like our statistics to be ***consistent*** - i.e., to lie in a narrower and narrower window around the correct value of some parameter for large N. An unbiased statistic is always consistent.

How should we choose amongst all these statistics?

3) A statistic should be ***impartial***: our conclusions should not depend on swapping the labels on the points/datasets (unless time is an important variable) or the units used.

- E.g., if we estimate the mean of sample A is higher than the mean of sample B by δ, using the same procedure with A and B reversed should yield $-\delta$.

4) We'd like our statistics to be ***efficient*** - to require as small a sample as possible to yield an accuracy within some threshold.

- Given a distribution, we can calculate the ***Asymptotic Relative Efficiency*** (ARE)***:***
  - If statistic A gives the same error with N_A data points as statistic B gives with N_B, the ARE of statistic A is the limit as N approaches ∞ of N_B/N_A . E.g., if N_A=1E6 yields the same errors as N_B=6E5, then statistic A has an ARE of 60%.

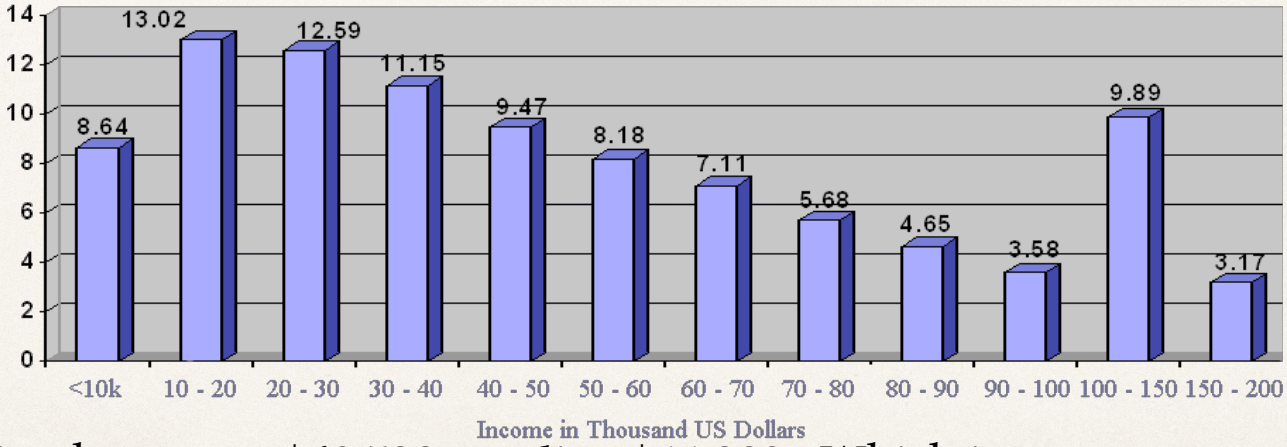
How should we choose amongst all these statistics?

5) A statistic should have ***closeness***: i.e., give a value as close as possible to the true value of some parameter of interest. However, there's lots of ways to measure closeness: do we minimize the RMS error? the average absolute deviation? etc.

- We can generalize this concept to say that a statistic should **minimize loss**, where the "***loss***" is the expectation value of some function over all possible samples.
- The estimators we derived from maximum likelihood for a Normal distribution would be equivalent to minimizing a loss given by $\sum_i -(x_i-\mu)^2 / (2\,\sigma_i^2)$. A different weighing of loss (e.g. one that depends linearly on deviations, rather than the square) would yield a different 'best' statistic.
- Some statistics minimize the maximum possible loss, instead of the expectation value; these are called ***Mini-max estimators***.

How should we choose?

6) Ideally, a statistic should be ***robust***: i.e., give the correct answer even if we have a non-Normal distribution (e.g., a Gaussian plus outliers). Although the ordinary mean has a high efficiency for normally-distributed data, **it is not robust**.



- This distribution has mean $60,528, median $44,389. Which is more representative of the population?
- What happens to each one if someone finds $10 billion stuffed in their couch?

If the median is more robust than the mean, why not always use it?

- We will try to determine the Asymptotic Relative Efficiency (ARE) of the median: how many observations do we need for the error in the mean to be equal to the error from the median of n observations?

```
nsims=int(1E4)
navg=int(5E3)
data=random.randn(nsims,navg)
means=np.mean(data,axis=1)
medians=np.median(data,axis=1)

print(f'means: {np.std(means,ddof=1):.4f}')
print(f'medians: {np.std(medians,ddof=1):.4f}')
print(f'ratio: {np.std(medians,ddof=1)
      / np.std(means,ddof=1):.4f}')
```

Interpreting our results

- The standard deviation of the median is about 1.25 times larger than the standard deviation of the mean.
- We can turn this into an ARE by remembering that the standard error scales as $n^{1/2}$
  - For the standard deviation of the mean to match the standard deviation of the median, we'd need only $\approx(1/1.25)^2=0.64$ as many data, so the ARE of the median is $\approx 64\%$ (the actual value is $2/\pi = 0.636...$).
- Using a median instead of a mean is like throwing away one-third of the data... but often the gain in robustness is worth it.
- Because the median directly depends on only 1 or 2 data points, it must have larger errors than the mean, which combines all of the data - *in the Gaussian case*.
- For non-Normal distributions, the median can have an ARE above 100%.

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

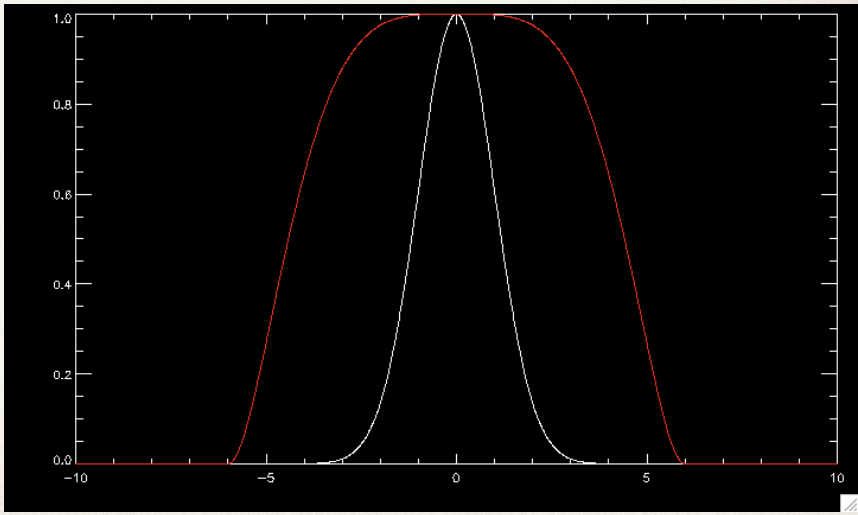
- A second common thing to do is to take an $\alpha\%$ ***trimmed mean*** (or ***trimmed standard deviation***):
  1) sort all the data by its value
  2) remove the lowest $\alpha\%$ and highest $\alpha\%$ of the data
  3) calculate the mean or standard deviation of the rest
- For the trimmed mean/std. 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.

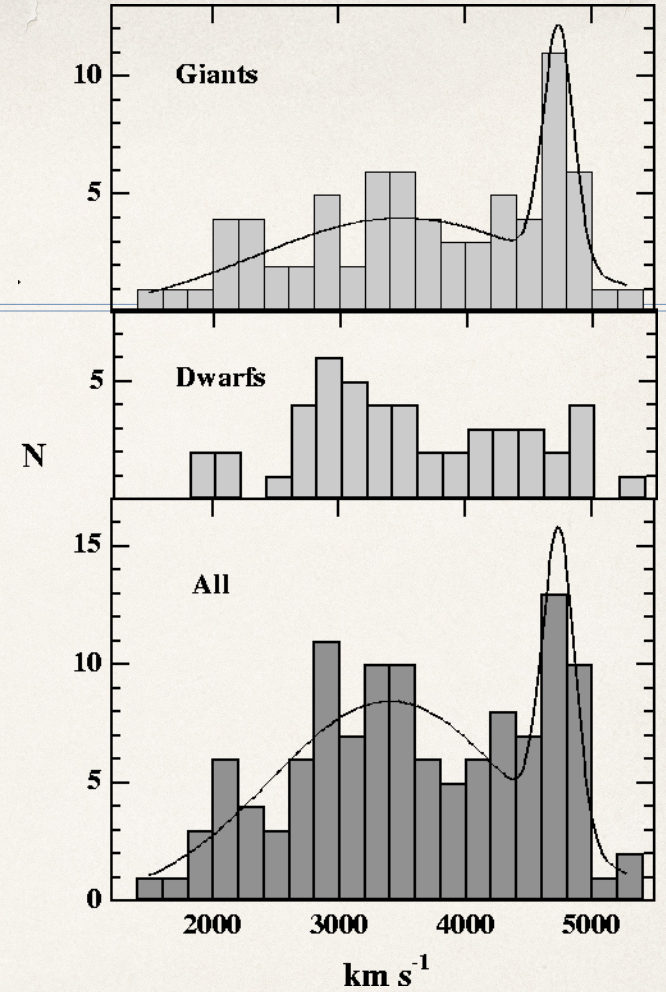
What makes this difficult?

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

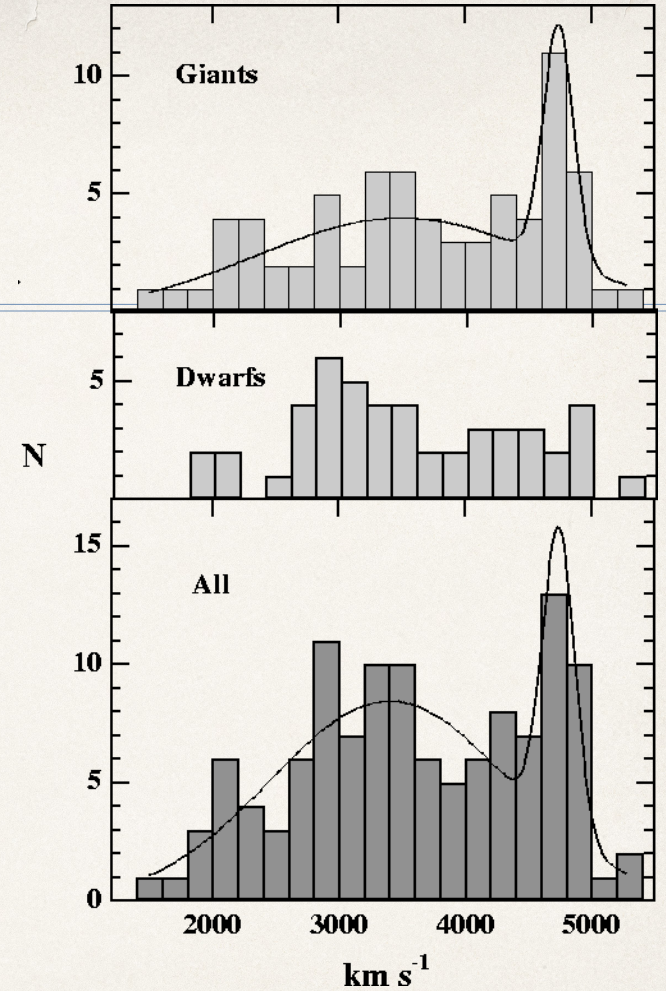


Stein, Jerjen et al. 1997

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.