Tutorial 3

Statistical Computation and Analysis

Spring 2024

Tutorial Outline

- Plug-in principle
- Biased / unbiased estimator

Data and Population

Goal: to infer the population parameter from the sampled data.

- **Parameter:** numerical value that describes a population. It is fixed and unknown but can be estimated.
- **Statistic:** numerical value that describes a sample. It varies from sample to sample and can be calculated from sample data.
- **Estimator:** a formula or rule that uses sample data to estimate an unknown parameter. Provides a best guess of the true value of the parameter, based on the available data.

Plugin principle

- The estimator value is calculated on a sample using the same formula as the population parameter.

	Parameter	Plugin estimator
Mean	$\mu = E(x) = \int_{-\infty}^{\infty} xp(x)dx$	$\bar{x} = \sum_{i=1}^{N} x_i p(x_i) = \frac{1}{N}\sum_{i=1}^{N} x_i$
Standard Deviation	$\sigma = \sqrt{E\left((x-\mu)^2\right)}$	$s_{plugin} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2}$

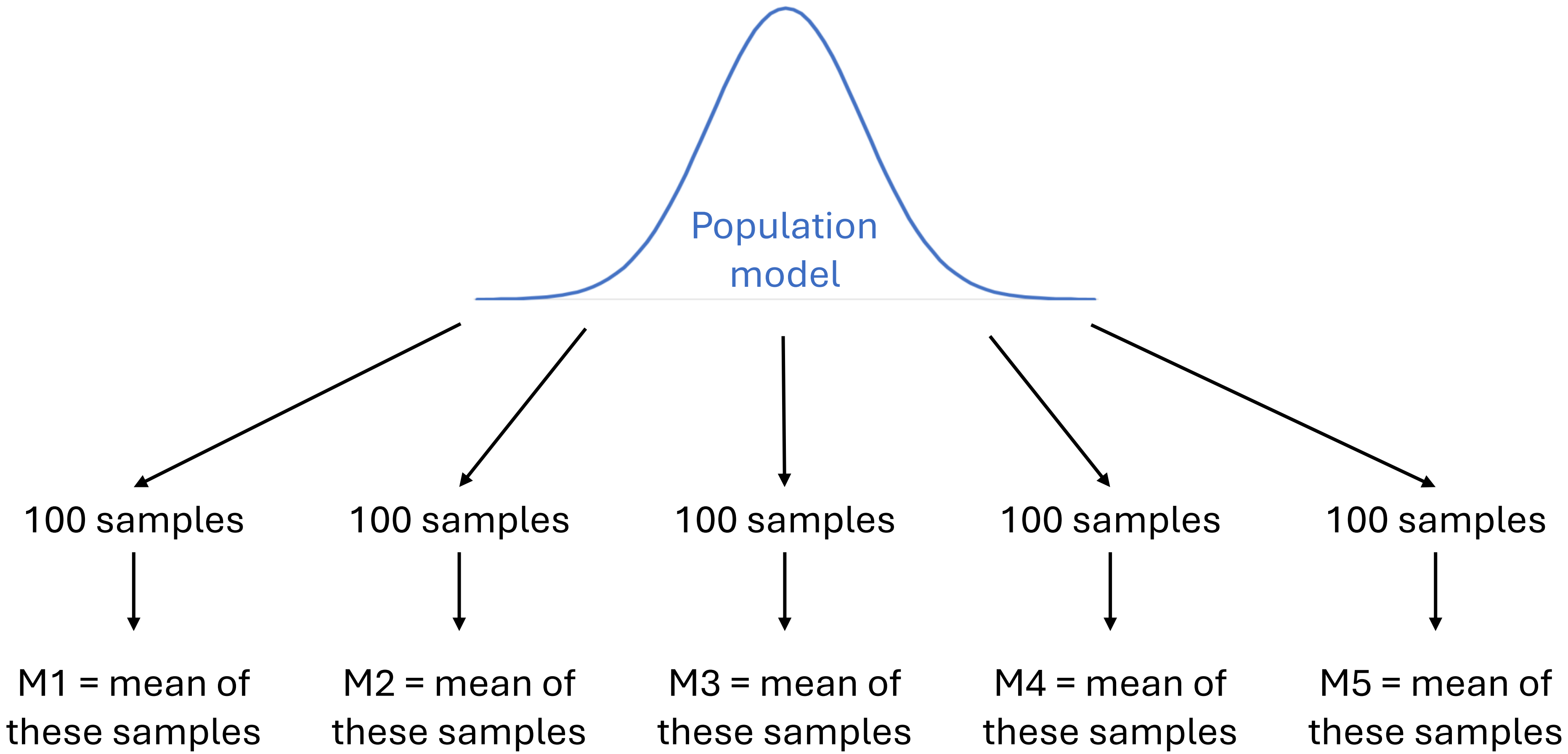
Usually underestimates the real standard deviation ($s_{plugin} < \sigma$)

Plugin principle

- The plugin principle is one method of generating an estimator.
- It does not always give the best estimator.
    - For example, the standard deviation.

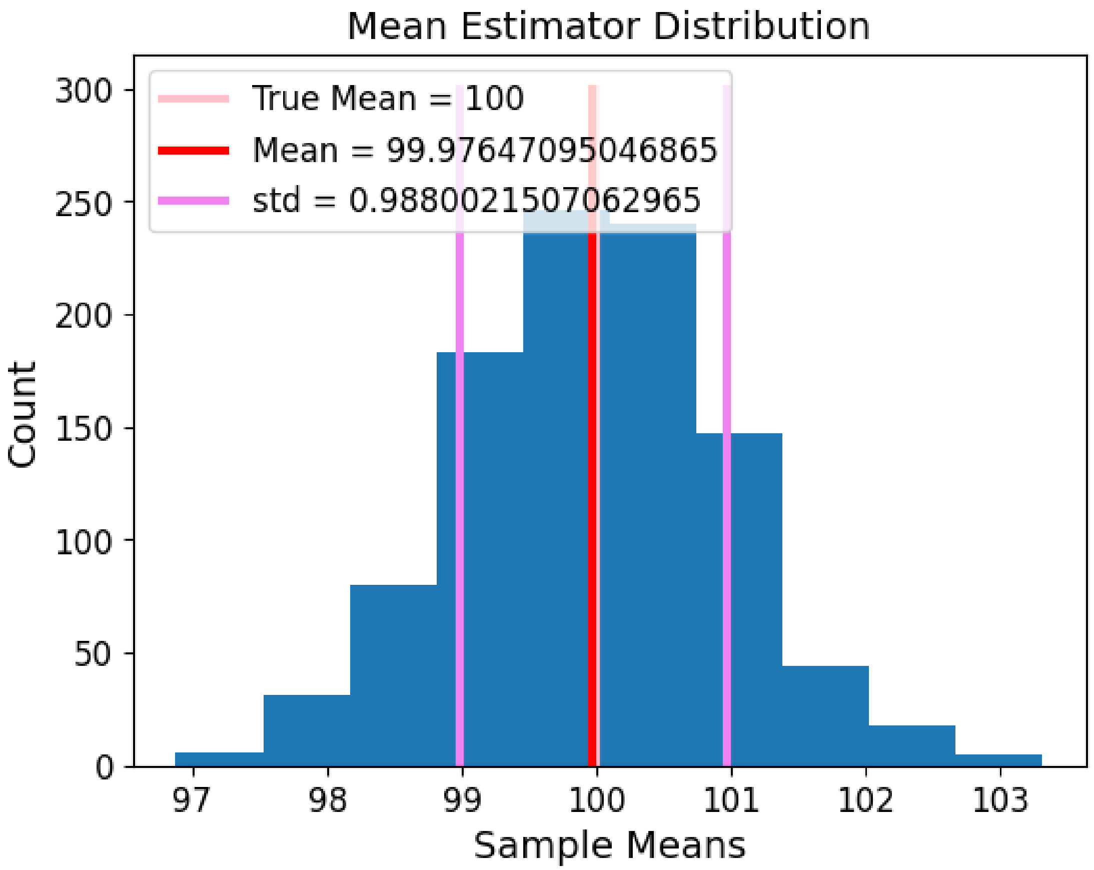
Bias and Variance

- Every time we sample, we get a different estimate.
    - Example, estimate the mean.



Bias and Variance

- The estimator has a distribution:
    - Plot all the means of the samples we took.



Bias and Variance

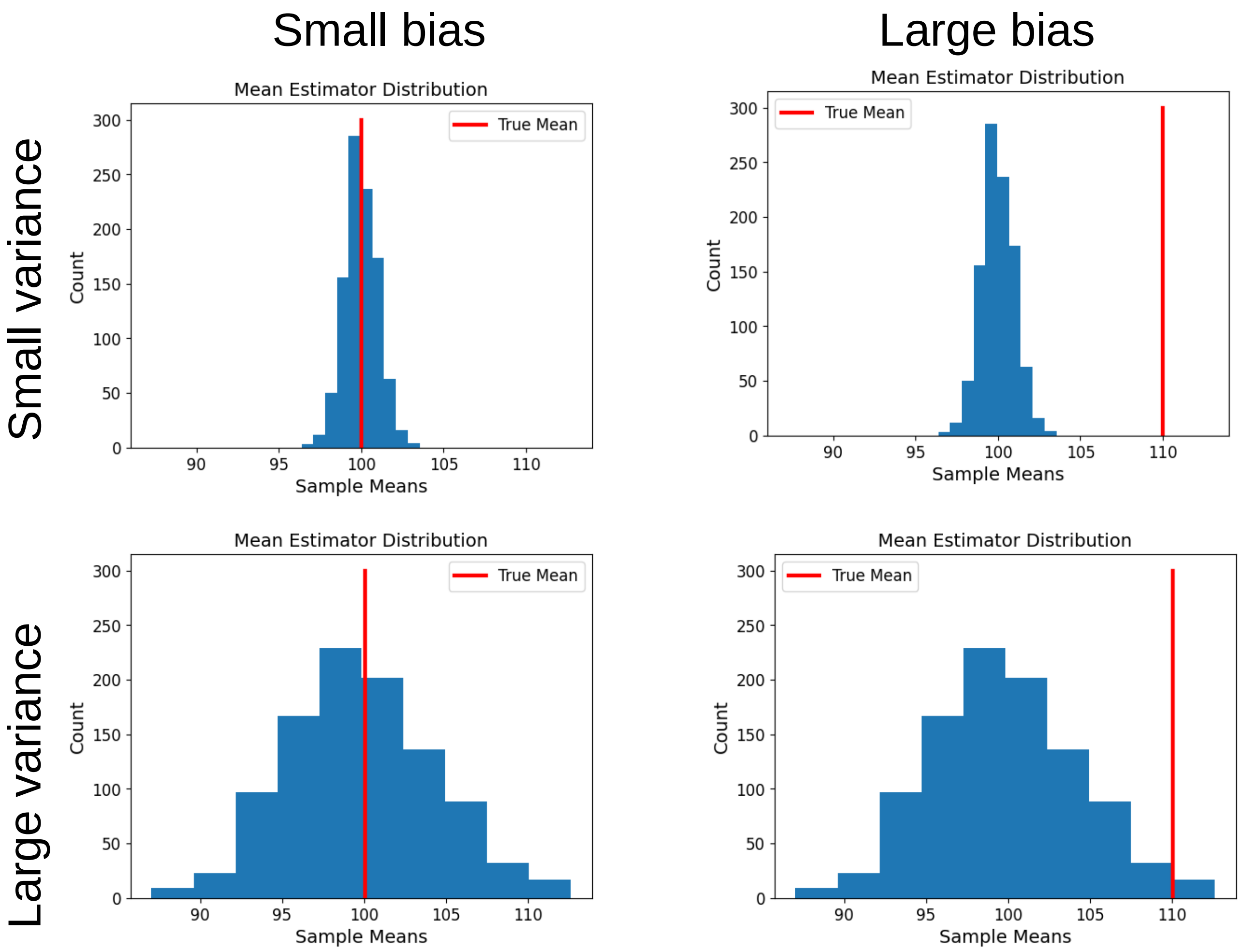
```
#mean estimator distribution

#population is normal distribution with mean of 100 and standard deviation of 10
#sample 100 datapoints each time
#repeat 1000 times to create estimator distribution
samples1000 = np.random.normal(100, 10, size = (100,1000))
AllMeans = np.mean(samples1000, axis = 0)
```

Bias and Variance

```
plt.figure()
plt.hist(AllMeans)
plt.xlabel('Sample Means', fontsize = 14)
plt.ylabel('Count', fontsize = 14)
plt.title('Mean Estimator Distribution', fontsize = 14)
plt.xticks(fontsize = 12)
plt.yticks(fontsize = 12)
plt.plot([np.mean(AllMeans), np.mean(AllMeans)], [0, 300], color = 'red', lw = 3, label = f'Mean = {np.mean(AllMeans)}')
plt.plot([np.mean(AllMeans) - np.std(AllMeans), np.mean(AllMeans) - np.std(AllMeans)], [0, 300], color = 'violet', lw = 3, label = f'std = {np.std(AllMeans)}')
plt.plot([np.mean(AllMeans) + np.std(AllMeans), np.mean(AllMeans) + np.std(AllMeans)], [0, 300], color = 'violet', lw = 3)
plt.xlabel('Sample Means', fontsize = 14)
plt.ylabel('Count', fontsize = 14)
plt.title('Mean Estimator Distribution', fontsize = 14)
plt.xticks(fontsize = 12)
plt.yticks(fontsize = 12)
plt.legend(fontsize = 12)
```

Bias and Variance

- Every time we sample, we get a different estimate.
- Goal: no bias, minimal variance



Bias

Unbiased estimator

- Expected value of the statistic = expected value of the population: $E\left[\widehat{\theta}\right] = \theta$

Biased estimator

- $E\left[\widehat{\theta}\right] \neq \theta$

Variance and standard deviation

Plug-in estimator is biased

$$s_{plugin}^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2$$

$$E(s_{plugin}^2) = \left(1 - \frac{1}{N}\right)\sigma^2 \neq \sigma^2$$

Unbiased estimator

$$s^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2$$

Standard deviation

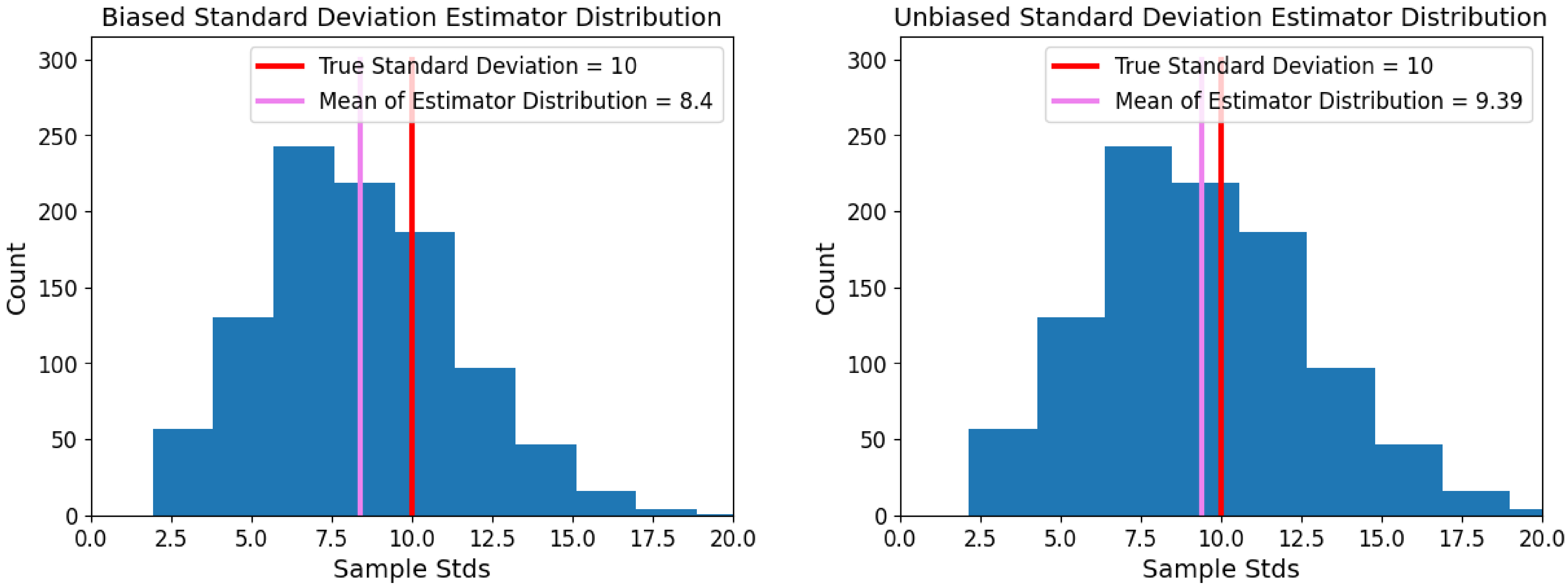
$$s = \sqrt{s^2}$$

Biased and Unbiased Estimators

Biased	Unbiased
Maximum	Mean
Minimum	Not plug-in variance
Plug-in variance	Not plug-in standard deviation
Plug-in standard deviation	

Bias and Variance

- Distribution of biased and unbiased standard deviation estimates.



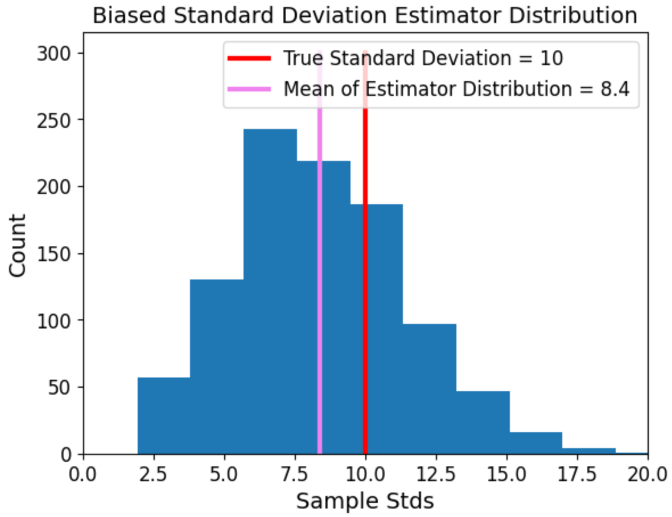
Bias and Variance

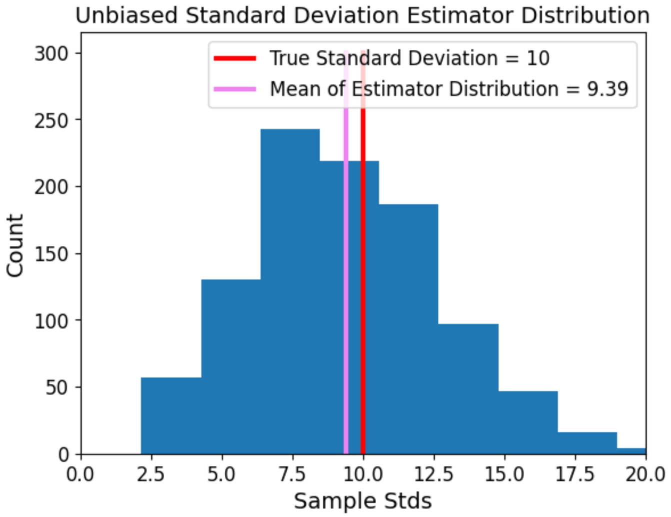
```
#standard distribution estimator distribution
samples = np.random.normal(100, 10, size = (5,1000))
AllStdsBiased = np.std(samples, axis = 0)
AllStdsunBiased = np.std(samples, axis = 0, ddof = 1)
```

Bias and Variance

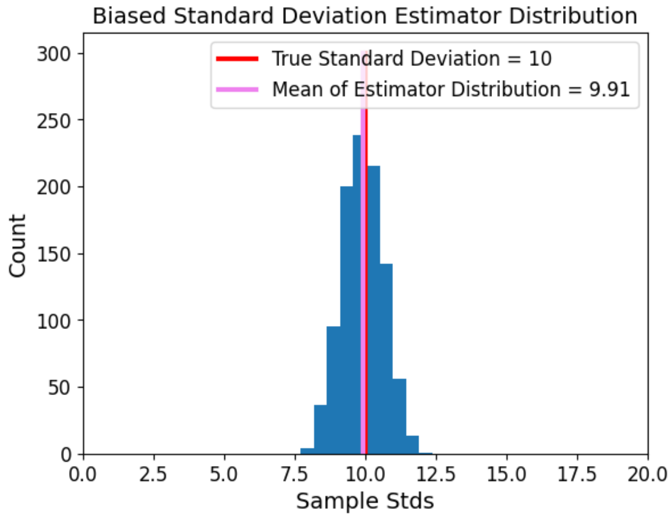
- Increasing the sample size will decrease the bias.

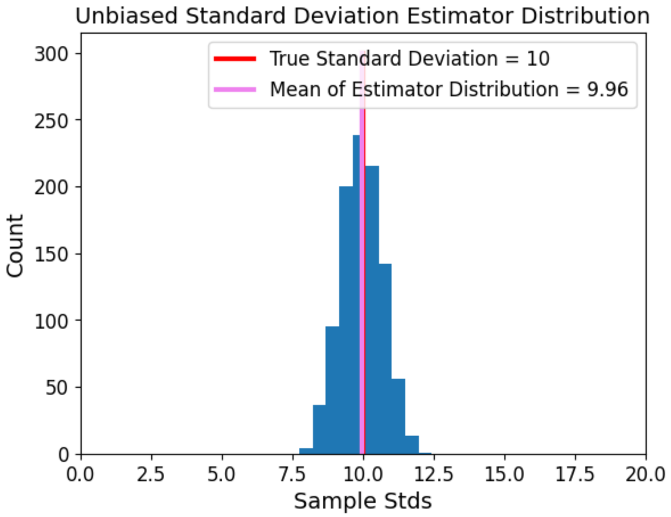
N = 5:





N = 100:





Bias and Variance

- There are different algorithms for generating estimators:
  - Plugin estimator
  - Minimum variance unbiased estimator
  - Least squares estimator
  - Maximum likelihood estimator

Simulation – Number of Girls Born

There is a 1/125 chance that a birth event results in fraternal twins, of which each has an approximate 49.5% chance of being a girl, and a 1/300 chance of identical twins, which have an approximate 49.5% chance of being a pair of girls. Simulate 400 birth events.

Simulation – Number of Girls Born

```
#brith types (twins and single births)
birth_types = np.array([0, 1, 2]) #fraternal twins, identical twins, single births
NumGirls = np.zeros((1000, 1))
for i in range(1000): #1000 repetitions
  #400 births in each simulation
  births = np.random.choice(birth_types, 400, replace = True, p = np.array([1/125, 
1/300, 1 - 1/125 - 1/300]))
  girlboy = np.zeros((births.shape[0], 1)) #1 = girl, 0 = boy
  for j in range(births.shape[0]):
    if births[j] == 0: #fraternal twins
      girlboy[j] = np.random.binomial(2, 0.495, 1) #there are two born
    if births[j] == 1: #identical twins
      girlboy[j] = 2*np.random.binomial(1, 0.495, 1) #either both are twins or 
neither are
    else:
      girlboy[j] = np.random.binomial(1, 0.488, 1)
  NumGirls[i] = np.sum(girlboy)
```