Confidence Interval and Bootstrapping

Il-Chul Moon
Dept. of Industrial and Systems Engineering
KAIST

icmoon@kaist.ac.kr

Detour) Dataset Introduction: Credit Score

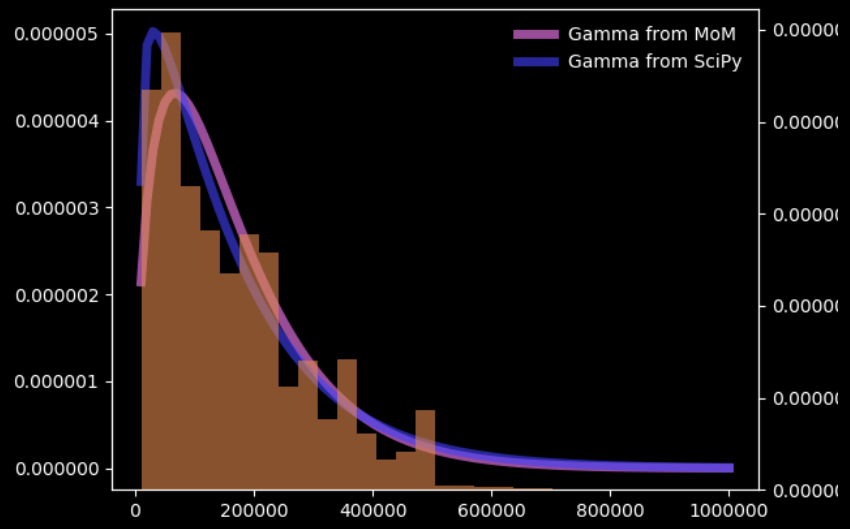
- This research aimed at the case of customers' default payments in Taiwan
- Dependent variable :
  - a binary variable, default payment (Yes = 1, No = 0), as the response variable.
- Independent variable:
  - X1: Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/her family (supplementary) credit.
  - X2: Gender (1 = male; 2 = female).
  - X3: Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).
  - X4: Marital status (1 = married; 2 = single; 3 = others).
  - X5: Age (year).
  - X6 - X11: History of past payment.
    - We tracked the past monthly payment records (from April to September, 2005) as follows: X6 = the repayment status in September, 2005; X7 = the repayment status in August, 2005; . . .;X11 = the repayment status in April, 2005.
    - The measurement scale for the repayment status is: -1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months; . . .; 8 = payment delay for eight months; 9 = payment delay for nine months and above.
  - X12-X17: Amount of bill statement (NT dollar).
    - X12 = amount of bill statement in September, 2005; X13 = amount of bill statement in August, 2005; . . .; X17 = amount of bill statement in April, 2005.
  - X18-X23: Amount of previous payment (NT dollar)
    - X18 = amount paid in September, 2005; X19 = amount paid in August, 2005; . . .;X23 = amount paid in April, 2005.
- Yeh, I. C., & Lien, C. H. (2009). The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. Expert Systems with Applications, 36(2), 2473-2480.
- Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

```
Mu:  167489.2389746325
Sigma^2:  16833730579.636848
Sigma:  129744.86725738652
Alpha:  1.6664544463029376
Beta:  9.94962098165205e-06
```

Detour) Distribution Fitting by SciPy

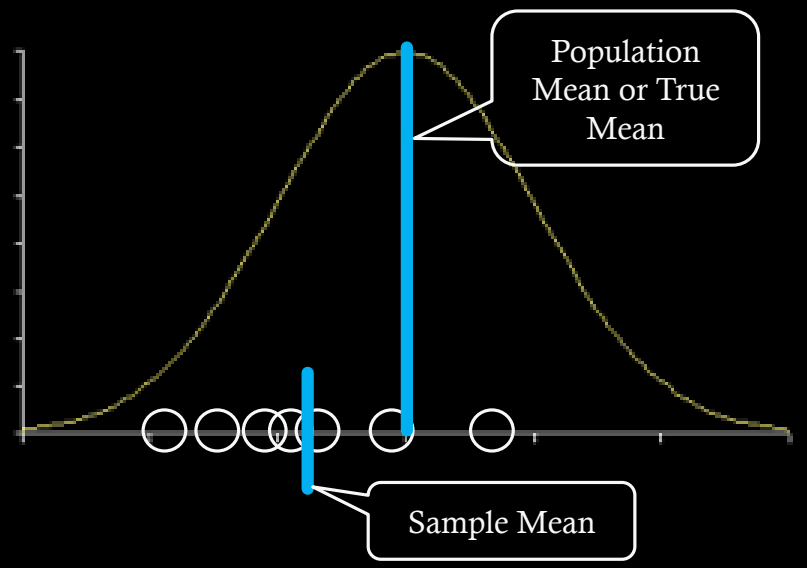
- SciPy provides a curve fitting functionality through its optimization
  - Not an exact match: SciPy uses its own optimization algorithm and appoximations
- SciPy defines the Gamma distribution with one more parameter
  - Location parameter
    - Original : $f(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}$
    - PDF with loc: $f(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} (x-\mu)^{\alpha-1} e^{-\beta(x-\mu)}$
  - With more parameterization, curve fitting becomes better

```
Mean, Sigma :  (167489.24424801697, 129744.8639212595)
Alpha from SciPy:  1.1572769644988954
Location from SciPy:  9440.977446027482
Beta from SciPy:  7.322301167530901e-06
```



Sample and Population

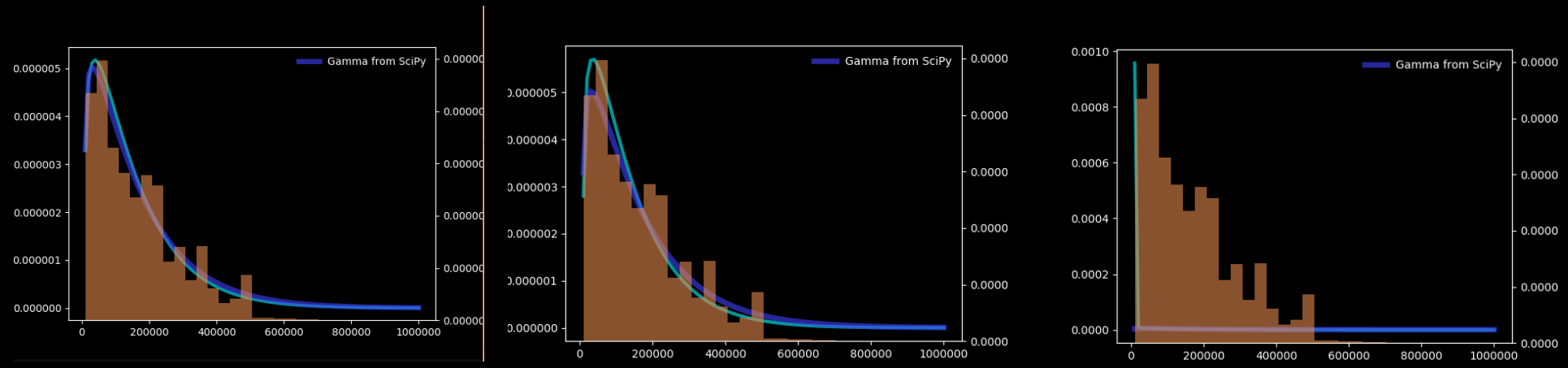
- The dataset provides 30,000 individual samples
  - This is only a proportion of population
  - Then, the question becomes whether the parameter estimates from the samples are reliable or not.
- There will be an error in the parameter estimation from samples.
  - We need to have a confidence interval to inform the validity of the sample mean with respect to population mean
  - This confidence interval is applicable to statistics, such as
    - Standard deviation, Variance
    - Median
    - Quantile….



```
Mu:  167489.2389746325
Sigma^2:  16833730579.636848
Sigma:  129744.86725738652
Alpha:  1.6664544463029376
Beta:  9.94962098165205e-06
```

Point Estimate on Single Sample

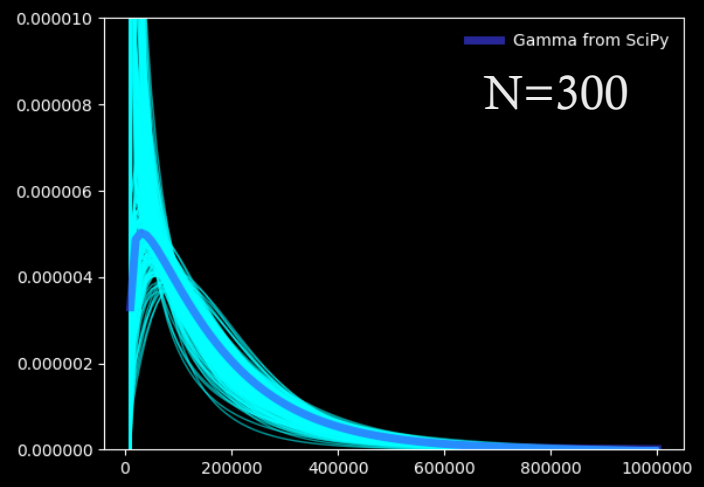
- Simulation setting!
- Let's consider 30,000 individuals as population
  - Draw 100 individuals as samples
- Calculate the population parameter and the sample parameter
  - Some cases are almost similar to the population parameter
  - Other cases are deviating from the population parameter

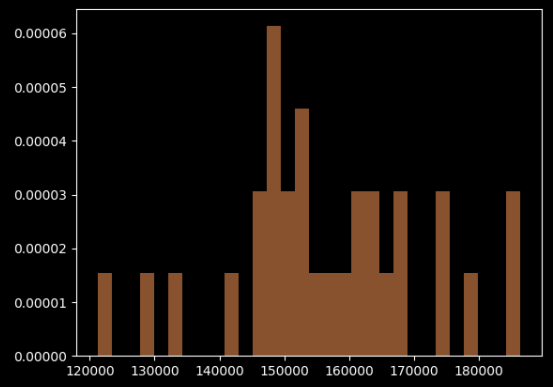


```
Mu:  167489.2389746325
Sigma^2:  16833730579.636848
Sigma:  129744.86725738652
Alpha:  1.6664544463029376
Beta:  9.94962098165205e-06
```

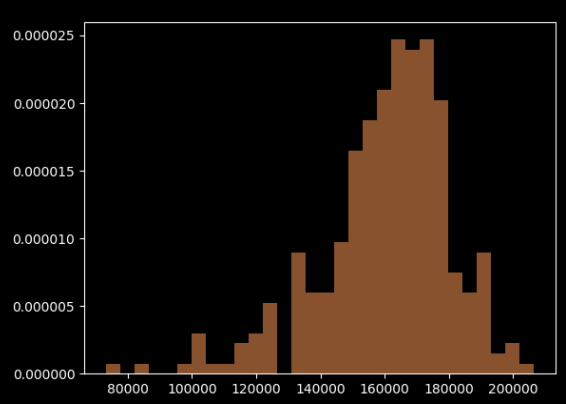
Point Estimate on Multiple Samples

- We draw a sample dataset for 30, 300, and 3000 times
  - Each dataset has 100 instances
  - We produced 300 Gamma distribution curve.
  - We created three histograms of sample means.
- Observe the distribution of sample means as the number of replications grow

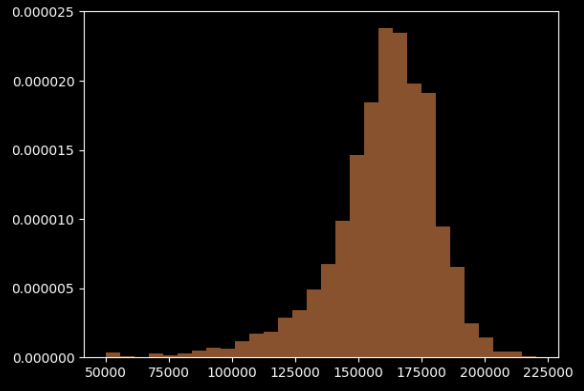




N=30



N=300



N=3000

Increasing Data Instances Per Sample Dataset

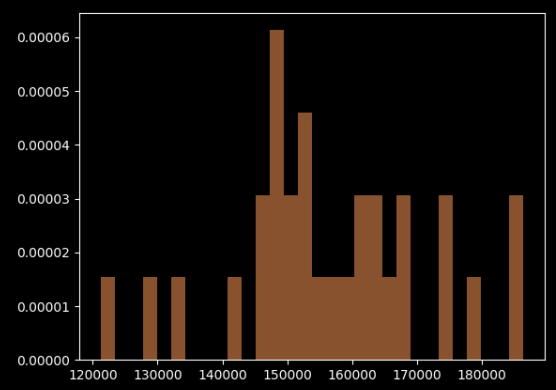
Central Limit Theorem : By the law of large numbers, the sample averages converge almost surely to the expected value μ as $n \to \infty$.

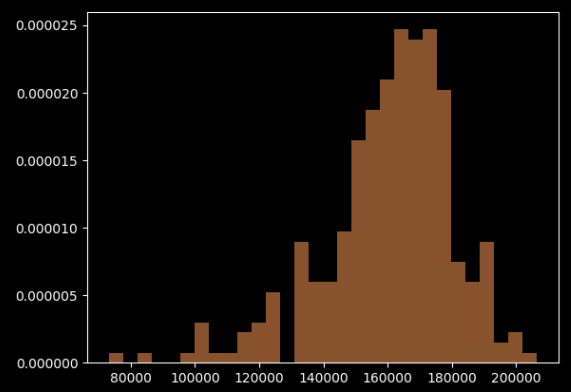
N=30

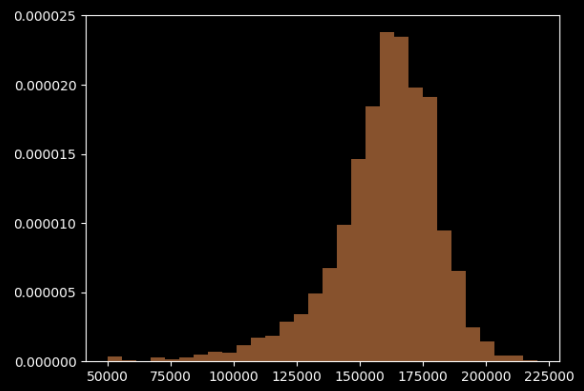
N=300

N=3000

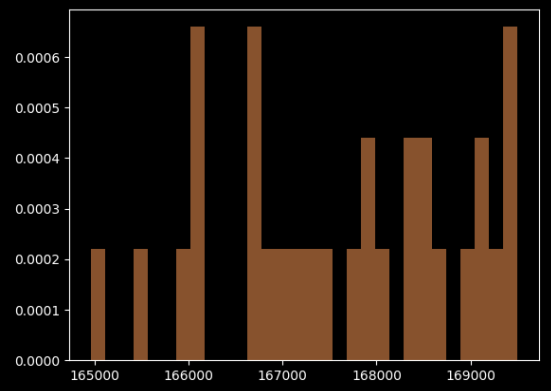
# of Samples per Dataset = 100

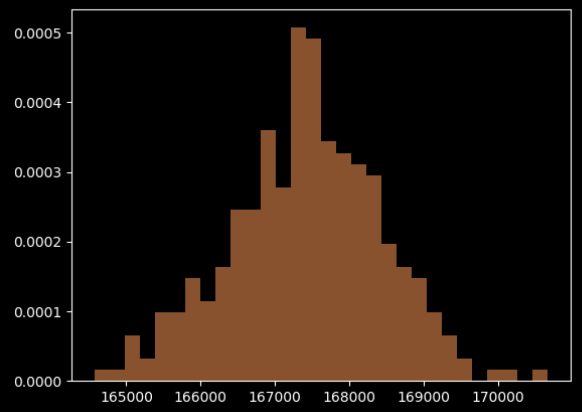


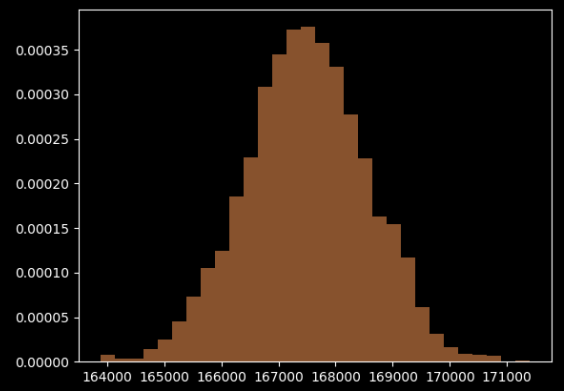




# of Samples per Dataset = 10000





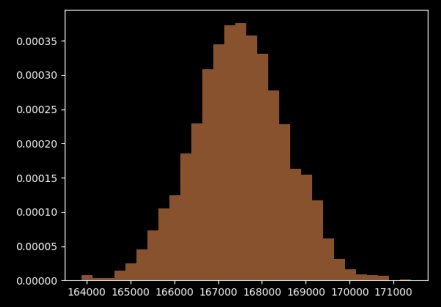


Central Limit Theorem

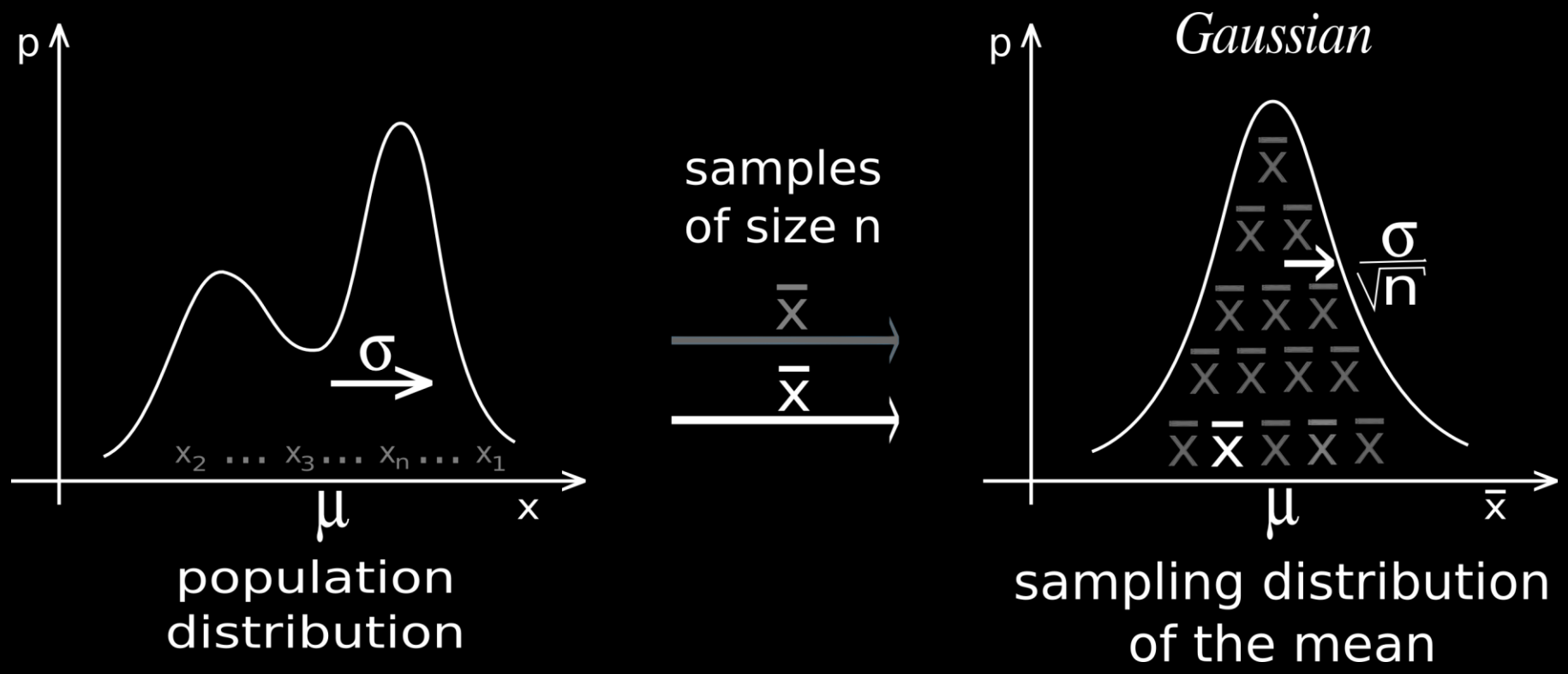
©wikipedia

N=3000

# of Samples per Dataset = 10000



Central Limit Theorem : By the law of large numbers, the sample averages converge almost surely to the expected value μ as $n \to \infty$.



Implementation of Multiple Sample Draws

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import norm, gamma
from scipy import stats
import random

objData = pd.read_excel('../data/ucidataset-credit score/default of credit card clients.xls', skiprows=[1,2])

lstIdx = list(range(len(objData['X1'])))

fig, ax1 = plt.subplots(1, 1)

numReplication = 30000
lstSampleMean = []
x = np.linspace(min(objData['X1']), max(objData['X1']), 100)
for j in range(numReplication):
    N = 100
    lstSample = []
    random.shuffle(lstIdx)
    for i in range(N):
        lstSample.append(objData['X1'][lstIdx[i]])
    objSampleResult = stats.rv_continuous.fit(gamma, lstSample)
    lstSampleMean.append(objSampleResult[0] * objSampleResult[2] + objSampleResult[1])
    ax1.plot(x, gamma.pdf(x,objSampleResult[0],objSampleResult[1],objSampleResult[2]), 'r-', lw=1, alpha=0.6)
objTrueResultGamma = stats.rv_continuous.fit(gamma, objData['X1'])
objTrueResultGaussian = stats.rv_continuous.fit(norm, objData['X1'])
print('Gamma True Parameter: ',objTrueResultGamma)
print('Gaussian True Parameter: ',objTrueResultGaussian)
print('Sample Parameter: ',objSampleResult)
print('True Mean: ',objTrueResultGamma[0]*objTrueResultGamma[2]+objTrueResultGamma[1])
print('Sample Mean: ',lstSampleMean)


ax1.plot(x, gamma.pdf(x,objTrueResultGamma[0],objTrueResultGamma[1],objTrueResultGamma[2]), 'y-', lw=5,
 alpha=0.6, label='Gamma from SciPy')

plt.ylim((0,0.00001))
ax1.legend(loc='best', frameon=False)
plt.show()

objSampleMean = pd.Series(lstSampleMean)
print(objSampleMean)

fig, ax1 = plt.subplots(1, 1)
ax1.hist(objSampleMean, bins=30, density=True, alpha=0.6)
plt.show()
```

Implementation of Multiple Sample Draws

```
numReplication = 30000
lstSampleMean = []
x = np.linspace(min(objData['X1']), max(objData['X1']), 100)
for j in range(numReplication):
    N = 100
    lstSample = []
    random.shuffle(lstIdx)
    for i in range(N):
        lstSample.append(objData['X1'][lstIdx[i]])
    objSampleResult = stats.rv_continuous.fit(gamma, lstSample)
    lstSampleMean.append(objSampleResult[0] * objSampleResult[2] + objSampleResult[1])
```

.....

```
objSampleMean = pd.Series(lstSampleMean)
print(objSampleMean)

fig, ax1 = plt.subplots(1, 1)
ax1.hist(objSampleMean, bins=30, density=True, alpha=0.6)
plt.show()
```

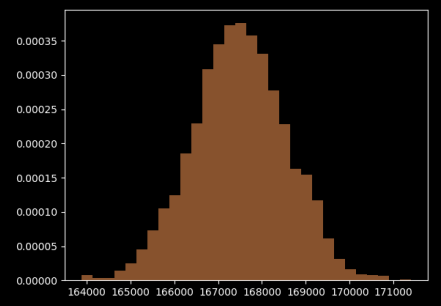
Detour) Some Properties of Expectations

- $E(k) = k$, if k is a constant
- $E(kx) = kE(x)$, if k is a constant and x is a random variable
- $E(x + y) = E(x) + E(y)$, if x and y are both random variables
- $Var(x) = E\big(x - E(x)\big)^2$
- $$\begin{aligned} Var(x+y) &= E\big((x+y) - E(x+y)\big)^2 = E\big((x+y) - E(x) - E(y)\big)^2 \\ &= E\Big(\big(x - E(x)\big) + \big(y - E(y)\big)\Big)^2 \\ &= E\Big(\big(x - E(x)\big)^2 + \big(y - E(y)\big)^2 + 2\big(x - E(x)\big)\big(y - E(y)\big)\Big) \\ &= E\big(x - E(x)\big)^2 + E\big(y - E(y)\big)^2 + E\Big(2\big(x - E(x)\big)\big(y - E(y)\big)\Big) \\ &= Var(x) + Var(y) + 2Cov(x, y) \end{aligned}$$
  - a.k.a. Variance Sum Law
  - If x and y are independent, $Cov(x, y) = 0$
- We will visit $Cov(x, y)$ in the future by
  - Expanding the univariate Gaussian distribution to the multivariate Gaussian distribution

Distribution of Sample Statistics

N=3000

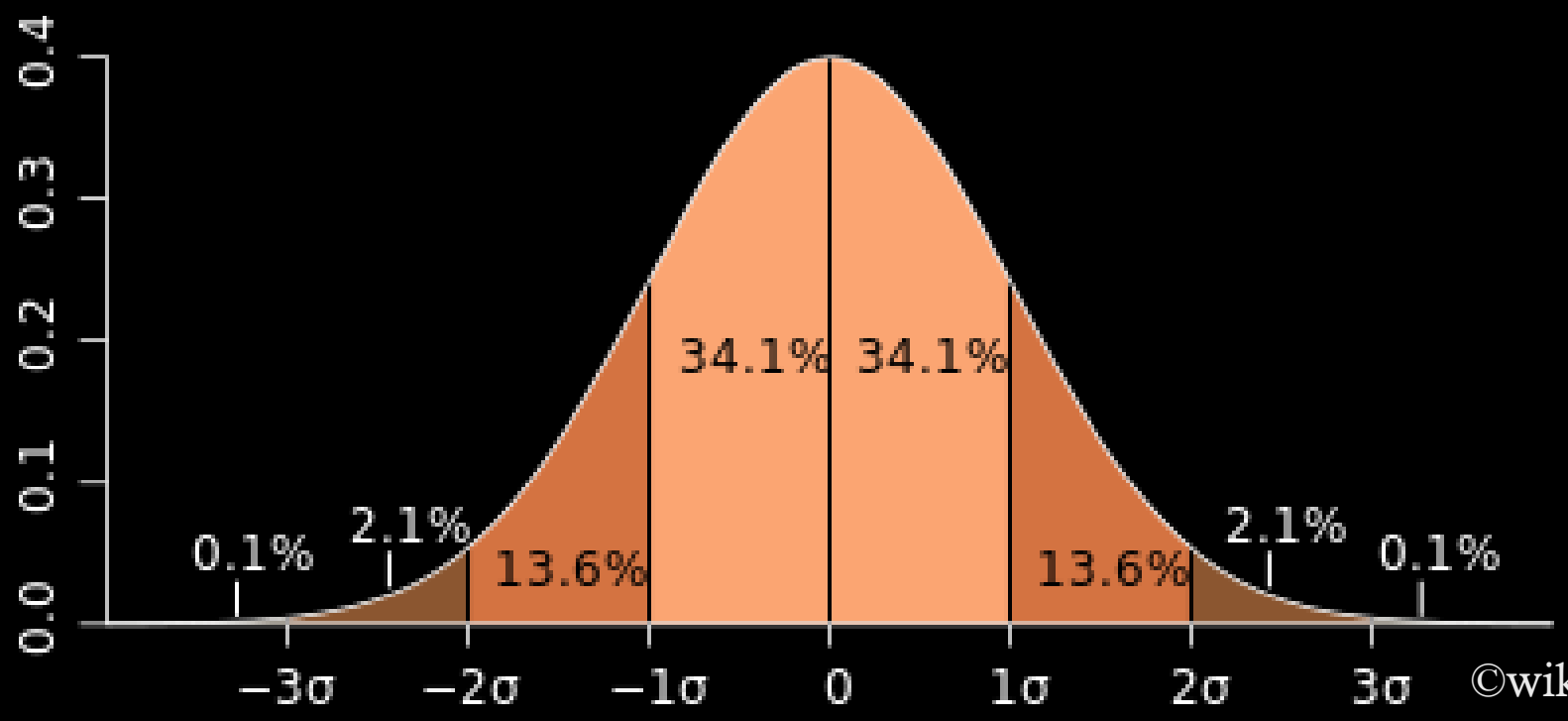
# of Samples per Dataset = 10000



- Let's say that $\hat{x}$ is sample mean and $\hat{\sigma}^2$ is sample variance
  - This is derived for a single dataset
  - $\hat{x} = E(x_i) = \frac{1}{n}\sum_{i=1}^{n} x_i$, $\hat{\sigma}^2 = Var(x_i) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \hat{x})^2$
- What would be the distribution of sample statistics?
  - We know that the distribution will eventually converge to a Gaussian distribution
  - Then, what would be the mean and the variance of the sample statistics?
- i.e.) the Gaussian distribution of the sample mean
  - $E(\hat{x}) = E\left(\frac{1}{n}\sum_{i=1}^{n} x_i\right) = \frac{1}{n}\sum_{i=1}^{n} E(x_i) = \frac{1}{n} \times n \times E(x_i) = \hat{x}$
  - $Var(\hat{x}) = Var\left(\frac{1}{n}\sum_{i=1}^{n} x_i\right) = \frac{1}{n^2} Var(\sum_{i=1}^{n} x_i) = \frac{1}{n^2}\sum_{i=1}^{n} Var(x_i) = \frac{1}{n^2} \times n \times Var(x_i) = \frac{1}{n} Var(x_i)$
    - $Var\left(\frac{1}{n} f\right) = \frac{1}{n-1}\sum_{i=1}^{n}\left(\frac{1}{n} f - E(\frac{1}{n} f)\right)^2 = \frac{1}{n-1}\sum_{i=1}^{n}\left(\frac{1}{n} f - \frac{1}{n} E(f)\right)^2$
      $$= \frac{1}{n^2}\frac{1}{n-1}\sum_{i=1}^{n}(f - E(f))^2 = \frac{1}{n^2} Var(f)$$
- $E(\hat{x})$ and $Var(\hat{x})$ are statistics of multiple sample dataset draws
  - However, they can be calculated from a single draw of sample dataset by $E(x_i)$ and $\frac{1}{n} Var(x_i)$

Confidence Interval of Normal Distribution

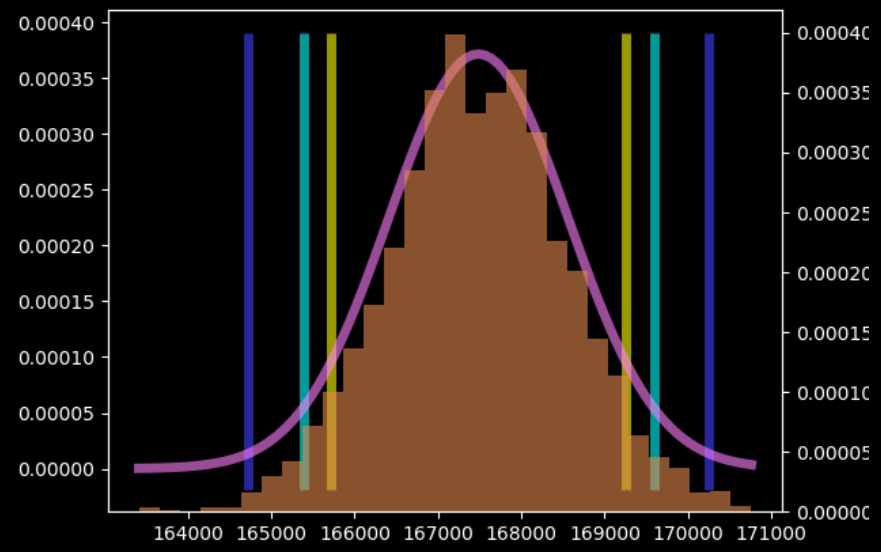
- We derived the sample mean will be distributed as $N\left(\hat{x}, \frac{1}{n}\hat{\sigma}^2\right)$
  - A draw of a sample mean fall into the confidence interval by $N\left(\hat{x}, \frac{1}{n}\hat{\sigma}^2\right)$
  - Two tail
    - $90\% : -1.645\sigma \sim 1.645\sigma$, $95\% : -1.960\sigma \sim 1.960\sigma$, $99\% : -2.576\sigma \sim 2.576\sigma$



©wikipedia

Drawing Sample Mean Distribution with Multiple Sample Simulations

$90\% : -1.645\sigma \sim 1.645\sigma$, $95\% : -1.960\sigma \sim 1.960\sigma$, $99\% : -2.576\sigma \sim 2.576\sigma$

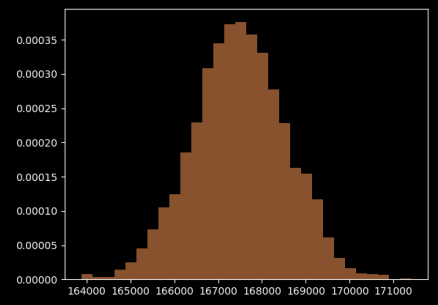



```python
objSampleMean = pd.Series(lstSampleMean)
dblMeanOfSampleMean = np.mean(objSampleMean)
dblVarOfSampleMean = np.var(objSampleMean)
print(dblMeanOfSampleMean)
print(dblVarOfSampleMean)

fig, ax1 = plt.subplots(1, 1)
ax2 = ax1.twinx()

x = np.linspace(min(objSampleMean), max(objSampleMean), 100)
ax2.hist(objSampleMean, bins=30, density=True, alpha=0.6)
ax1.plot(x,
         norm.pdf(x,dblMeanOfSampleMean,math.sqrt(dblVarOfSampleMean)),
         'g-', lw=5, alpha=0.6)

dblYMin, dblYMax = ax1.get_ylim()
ax1.vlines(x=dblMeanOfSampleMean - math.sqrt(dblVarOfSampleMean) * 1.645,
           ymin=dblYMin, ymax=dblYMax, lw=5, color='b', alpha=0.6)
ax1.vlines(x=dblMeanOfSampleMean + math.sqrt(dblVarOfSampleMean) * 1.645,
           ymin=dblYMin, ymax=dblYMax, lw=5, color='b', alpha=0.6)
ax1.vlines(x=dblMeanOfSampleMean - math.sqrt(dblVarOfSampleMean) * 1.960,
           ymin=dblYMin, ymax=dblYMax, lw=5, color='r', alpha=0.6)
ax1.vlines(x=dblMeanOfSampleMean + math.sqrt(dblVarOfSampleMean) * 1.960,
           ymin=dblYMin, ymax=dblYMax, lw=5, color='r', alpha=0.6)
ax1.vlines(x=dblMeanOfSampleMean - math.sqrt(dblVarOfSampleMean) * 2.576,
           ymin=dblYMin, ymax=dblYMax, lw=5, color='y', alpha=0.6)
ax1.vlines(x=dblMeanOfSampleMean + math.sqrt(dblVarOfSampleMean) * 2.576,
           ymin=dblYMin, ymax=dblYMax, lw=5, color='y', alpha=0.6)

plt.show()
```

Bootstrapping

N=3000

# of Samples per Dataset = 10000



- Up to this point, we utilized the given dataset as the population, and we sampled from the dataset
  - We know the importance of increasing
    - The number of samples per dataset should be large
    - The number of datasets should be large
- Resampling : simulating multiple datasets from a single dataset
  - Cross-Validation
  - Bootstrapping
- Bootstrapping : random sampling with replacement
  - Pseudo code
    - sampleBootstrap = []
    - Loop until enough number of instances in sampleBootstrap
      - SampledInstance = I.I.D Sample from a given dataset
      - sampleBootstrap.append(SampledInstance)
  - Can appear the same value for multiple, one, or zero times
    - The number of sample instances per dataset could go over the original dataset size
  - After drawing a re-sampled dataset, we can calculate the statistics that we need

Implementation of Bootstrapping

```python
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import norm, gamma
from scipy import stats
import random
```

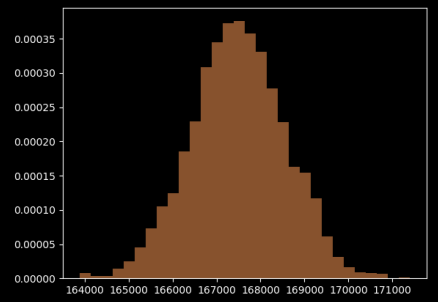
```python
def getBootstrapSamples(objSeries, intSampleNum):
    lstSample = []
    for i in range(intSampleNum):
        intRandomIdx = random.randrange(0,len(objSeries))
        lstSample.append(objSeries[intRandomIdx])
    return lstSample

def getMeanFromSample(lstSample):
    objSampleResult = stats.rv_continuous.fit(gamma, lstSample)
    return objSampleResult[0] * objSampleResult[2] + objSampleResult[1]

if __name__ == '__main__':
    objData = pd.read_excel('../data/ucidataset-credit score/default of credit card clients.xls',
                            skiprows=[1,2])
    intNumReplication = 3000
    intBootstrapSize = 1000
    intNumShowBootstrap = 3
    fig, axs = plt.subplots(intNumShowBootstrap+2)
    fig.suptitle('Bootstrapping')
    dblXMin, dblXMax = min(objData['X1']), max(objData['X1'])
    axs[0].hist(objData['X1'], range=[dblXMin, dblXMax], bins=30, density=True, alpha=0.6, color='r')

    lstSampleMean = []
    for i in range(intNumReplication):
        lstSample = getBootstrapSamples(objData['X1'], intBootstrapSize)
        dblSampleMean = getMeanFromSample(lstSample)
        lstSampleMean.append(dblSampleMean)

        if i < intNumShowBootstrap:
            axs[i+1].hist(lstSample, range=[dblXMin, dblXMax], bins=30, density=True, alpha=0.6)
            dblYMin, dblYMax = axs[i+1].get_ylim()
            axs[i+1].vlines(x=dblSampleMean, ymin=dblYMin, ymax=dblYMax, lw=5, color='b', alpha=0.6)

    axs[-1].hist(lstSampleMean,range=[dblXMin, dblXMax],bins=30,density=True,alpha=0.6, color='y')
    plt.show()
```

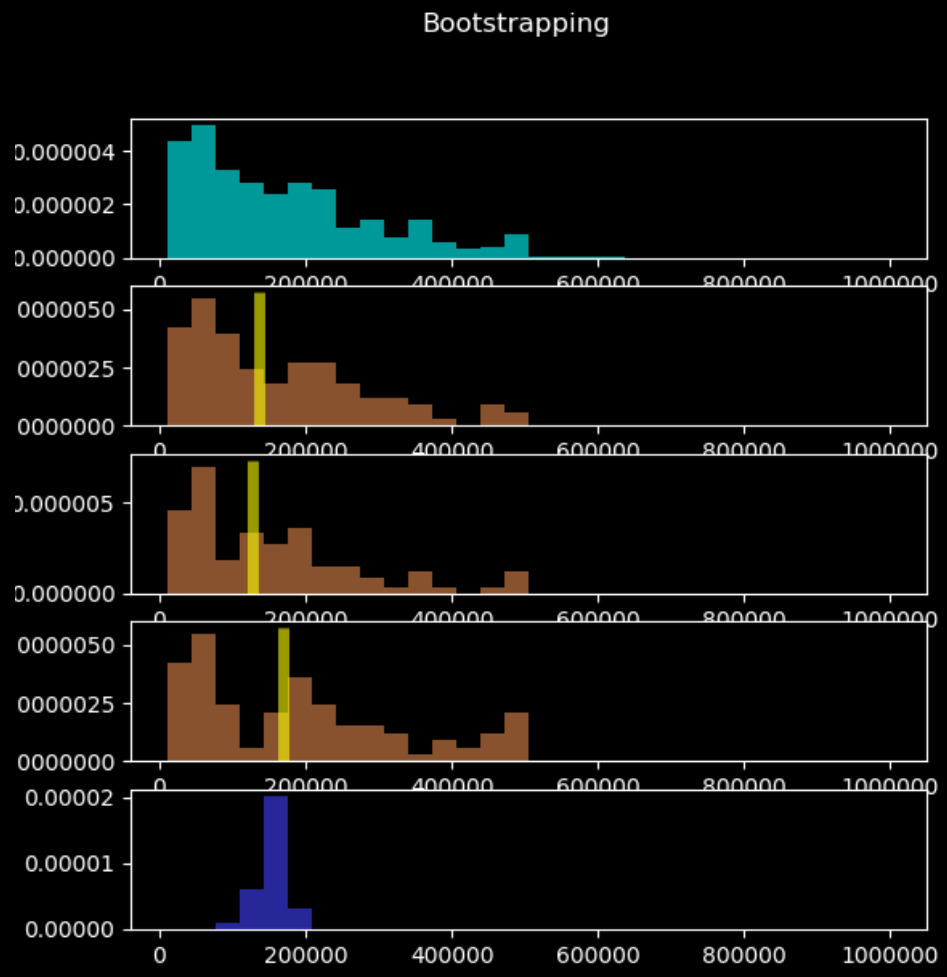
Confidence Interval of Sample Statistics

- Result of Bootstrapping
  - Original sample distribution
  - Bootstrapped re-sampled distribution
    - Bootstrap sample mean
  - Bootstrap sample mean distribution
    - Sample mean distribution follows the Normal distribution with many bootstrap simulations and many resampled datasetinstances

N=3000

# of Samples per Dataset = 10000





Confidence Interval by SciPy

- SciPy provides an API to calculate the confidence interval on sample statistics

```python
if __name__ == '__main__':
    objData = pd.read_excel('../data/ucidataset-credit score/default of credit card clients.xls',
                            skiprows=[1,2])
    intNumReplication = 3000
    intBootstrapSize = 10000

    lstSampleMean = []
    for i in range(intNumReplication):
        lstSample = getBootstrapSamples(objData['X1'], intBootstrapSize)
        dblSampleMean = getMeanFromSample(lstSample)
        lstSampleMean.append(dblSampleMean)

    objSampleMean = pd.Series(lstSampleMean)
    dblMeanOfSampleMean = np.mean(objSampleMean)
    dblVarOfSampleMean = np.var(objSampleMean)

    print("Confidence Interval on Mean From Bootstrapping : ",
          (dblMeanOfSampleMean - math.sqrt(dblVarOfSampleMean) * 1.645
           ,dblMeanOfSampleMean + math.sqrt(dblVarOfSampleMean) * 1.645))

    print("Confidence Interval on Mean From SciPy in Normal Distribution: ",
          stats.norm.interval(alpha=0.90,
                              loc=np.mean(objData['X1']), scale=stats.sem(objData['X1'])))

    print("Confidence Interval on Mean From SciPy in T-Distribution: ",
          stats.t.interval(alpha=0.90, df=len(objData['X1']) - 1,
                              loc=np.mean(objData['X1']), scale=stats.sem(objData['X1'])))

    dblSampleMean = np.mean(objData['X1'])
    dblSampleVariance = np.var(objData['X1'],ddof=1)
    print("Confidence Interval on Mean From Derivation : ",
          (dblSampleMean - math.sqrt(dblSampleVariance) * 1.645 / math.sqrt(len(objData['X1']))
           ,dblSampleMean + math.sqrt(dblSampleVariance) * 1.645 / math.sqrt(len(objData['X1']))))
```

Result From Bootstrap and APIs

- Similar results from bootstrapping, APIs, and closed-form derivations

```
intNumReplication = 3000
intBootstrapSize = 10000
```

- Why do we study bootstrapping and inside mechanism of confidence intervals?

```
print("Confidence Interval on Mean From Bootstrapping : ",
      (dblMeanOfSampleMean - math.sqrt(dblVarOfSampleMean) * 1.645
       ,dblMeanOfSampleMean + math.sqrt(dblVarOfSampleMean) * 1.645))

print("Confidence Interval on Mean From SciPy in Normal Distribution: ",
      stats.norm.interval(alpha=0.90,
                          loc=np.mean(objData['X1']), scale=stats.sem(objData['X1'])))

print("Confidence Interval on Mean From SciPy in T-Distribution: ",
      stats.t.interval(alpha=0.90, df=len(objData['X1']) - 1,
                          loc=np.mean(objData['X1']), scale=stats.sem(objData['X1'])))

dblSampleMean = np.mean(objData['X1'])
dblSampleVariance = np.var(objData['X1'],ddof=1)
print("Confidence Interval on Mean From Derivation : ",
      (dblSampleMean - math.sqrt(dblSampleVariance) * 1.645 / math.sqrt(len(objData['X1']))
       ,dblSampleMean + math.sqrt(dblSampleVariance) * 1.645 / math.sqrt(len(objData['X1']))))
```

```
Confidence Interval on Mean From Bootstrapping :  (165345.02903317523, 169593.55982756751)
Confidence Interval on Mean From SciPy in Normal Distribution:  (166257.06709710872, 168721.41085215626)
Confidence Interval on Mean From SciPy in T-Distribution:  (166257.0290445027, 168721.44890476228)
Confidence Interval on Mean From Derivation :  (166256.95744798795, 168721.52050127703)
```