# 20250528_01

May 28, 2025

[9]:
```python
import numpy as np
import matplotlib.pyplot as plt
```

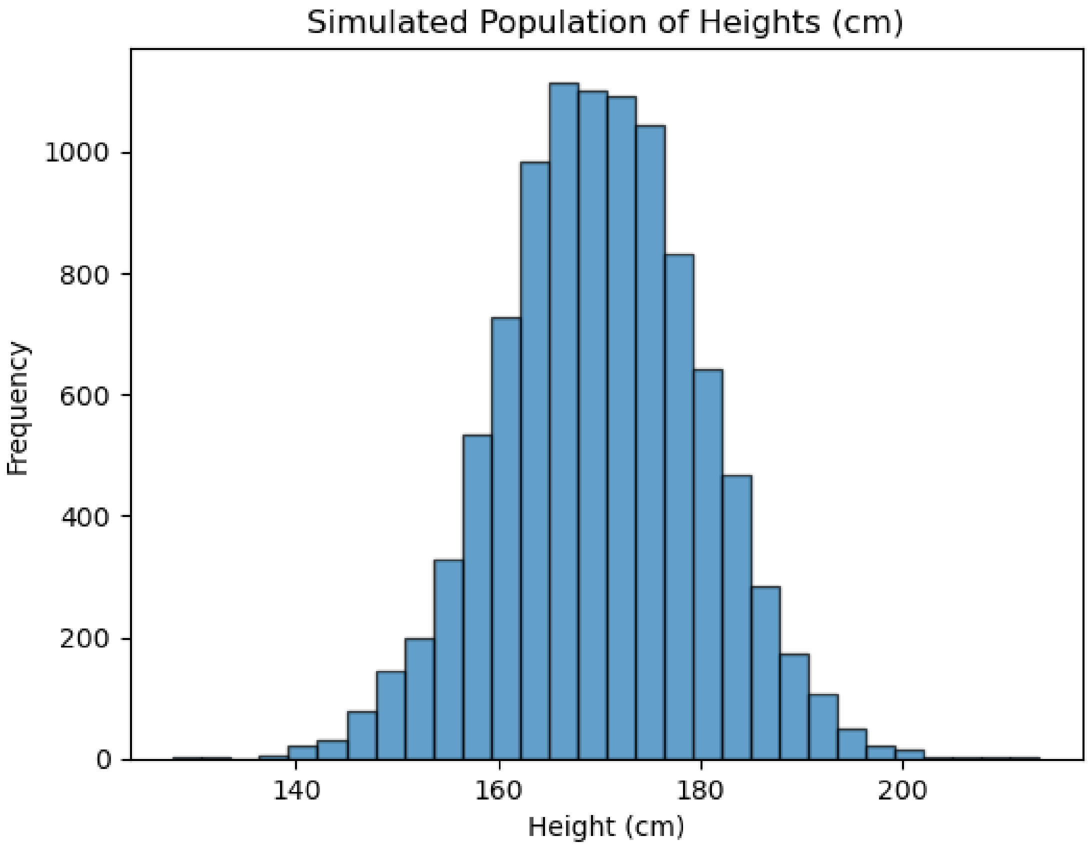
[10]:
```python
np.random.seed(420)
```

[13]:
```python
#To simulate population
population = np.random.normal(loc = 170, scale = 10, size = 10000)
```

[14]:
```python
plt.hist(population, bins = 30, edgecolor = 'black', alpha = 0.7)

plt.title("Simulated Population of Heights (cm)")
plt.xlabel("Height (cm)")
plt.ylabel("Frequency")

plt.show()
```




[24]:
```
#Store means from bootstrap
sample_means = []

#Bootstrap, 100 datas at a time, and can repeatedly chosen, because it's␣
↪bootstrap.
for _ in range(1000):
    sample = np.random.choice(population, size = 100, replace = True)
    sample_means.append(np.mean(sample))
```

[16]:
```
# Get 95% CI from percentiles
lower = np.percentile(sample_means, 2.5)
upper = np.percentile(sample_means, 97.5)

print(f"95% CI (bootstrap): [{lower:.2f}, {upper:.2f}]")
```

```
95% CI (bootstrap): [167.95, 171.89]
```

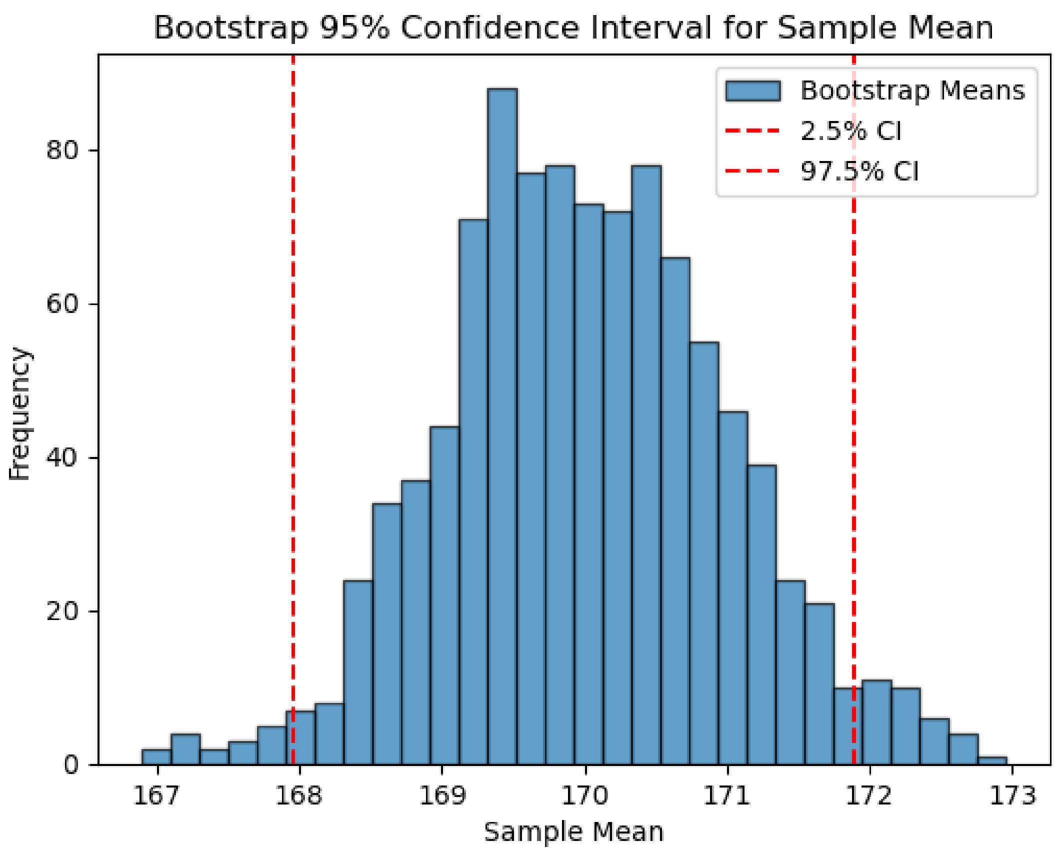
[30]:
```
# Plot histogram of bootstrap means
plt.hist(sample_means, bins = 30, edgecolor = 'black', alpha = 0.7, label =␣
↪'Bootstrap Means')
```

```python
# Add CI boundaries
plt.axvline(lower, color = 'red', linestyle = '--', label = '2.5% CI')
plt.axvline(upper, color = 'red', linestyle = '--', label = '97.5% CI')

# Decorate plot
plt.title("Bootstrap 95% Confidence Interval for Sample Mean")

plt.xlabel("Sample Mean")
plt.ylabel("Frequency")

plt.legend()
plt.show()
```




[26]:

```python
# Taking a single sample
sample = np.random.choice(population, size = 100, replace = False)

# Sample stats
sample_mean = np.mean(sample)
```

```python
sample_std = np.std(sample, ddof=1)  # ddof = Delta Degree of Freedom, sample SD means that use ddof = 1, it's a theorem.
n = len(sample)
z = 1.96  # 95% confidence, derive from standard normal distribution where P(-1.96 < Z < 1.96) = 0.95. You don't really need to calculate this, just look up the chart.

# Confidence interval formula
ci_lower = sample_mean - z * (sample_std / np.sqrt(n))
ci_upper = sample_mean + z * (sample_std / np.sqrt(n))

print(f"95% CI (formula): [{ci_lower:.2f}, {ci_upper:.2f}]")
```

```
95% CI (formula): [167.81, 172.08]
```