Introduction to Inferential Statistics with Python

This notebook is designed for beginners to understand and apply inferential statistics using Python step-by-step. It combines theory, real-life examples, and Python code. Here's what we covered so far:

1. **Inferential Statistics**: Predict population values using a sample. Compared with descriptive stats.
2. **Basic Probability**: Used coin toss and dice roll to explain probability. Plotted uniform distribution.
3. **Simulated Dice Rolls**: Rolled 1 die 10,000 times and analyzed the sum of two dice with a plot.
4. **Z-Score**: Explained what it is and how to standardize data. Plotted original vs. standardized data.
5. **Small Dataset Example**: Took a simple dataset, calculated mean, standard deviation, and standard error manually and with Python.
6. **Confidence Interval (CI)**: Explained what CI is, why it's useful, formula with real calculation steps, and implementation in Python.

What is Inferential Statistics?

Inferential Statistics is the process of using data from a **sample** to make **conclusions or predictions** about a **larger population.**

Descriptive vs Inferential Statistics

- **Descriptive Statistics**: Deals with summarizing data using mean, median, etc.
- **Inferential Statistics**: Involves **generalizing** beyond the data using probability theory.

Example:

- **Descriptive**: Average score of 50 students = 78
- **Inferential**: Estimating the average score of all 10,000 students in the city

Basic Probability

Coin Toss Example (Theory)

Suppose we toss a fair coin. It has two outcomes: **Heads** or **Tails.**

Each outcome has an equal chance of occurring:

- Probability(Heads) = $\frac{1}{2} = 0.5$
- Probability(Tails) = $\frac{1}{2} = 0.5$

In [1]:
```
# Coin toss
print("Probability of Heads in a fair coin toss =", 1/2)
```

```
Probability of Heads in a fair coin toss = 0.5
```

Dice Roll Example

When rolling a fair 6-sided die:

- Each number (1 through 6) has a probability of $\frac{1}{6} \approx 0.167$

In [2]:
```
# Dice roll (1 to 6 equally likely)
print("Probability of getting 6 in fair die =", 1/6)
```

```
Probability of getting 6 in fair die = 0.16666666666666666
```

This is called a **uniform distribution** because all outcomes are equally likely.

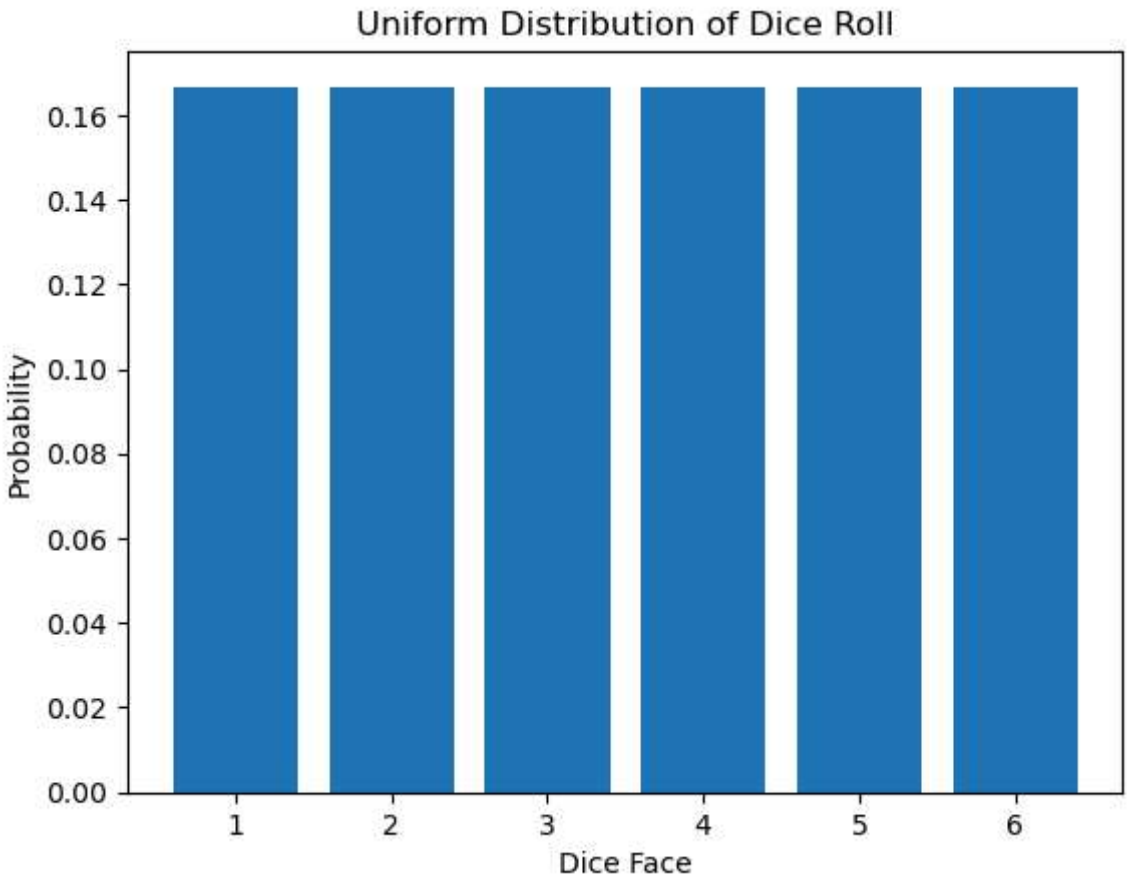
In [3]:
```
# Visualizing Uniform Distribution
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

# Coin toss
print("Probability of Heads in a fair coin toss =", 1/2)

# Dice roll (1 to 6 equally likely)
print("Probability of getting 6 in fair die =", 1/6)

# Uniform distribution (dice roll)
outcomes = [1, 2, 3, 4, 5, 6]
prob = [1/6]*6
plt.bar(outcomes, prob)
plt.title("Uniform Distribution of Dice Roll")
plt.xlabel("Dice Face")
plt.ylabel("Probability")
plt.show()
```

```
Probability of Heads in a fair coin toss = 0.5
Probability of getting 6 in fair die = 0.16666666666666666
```

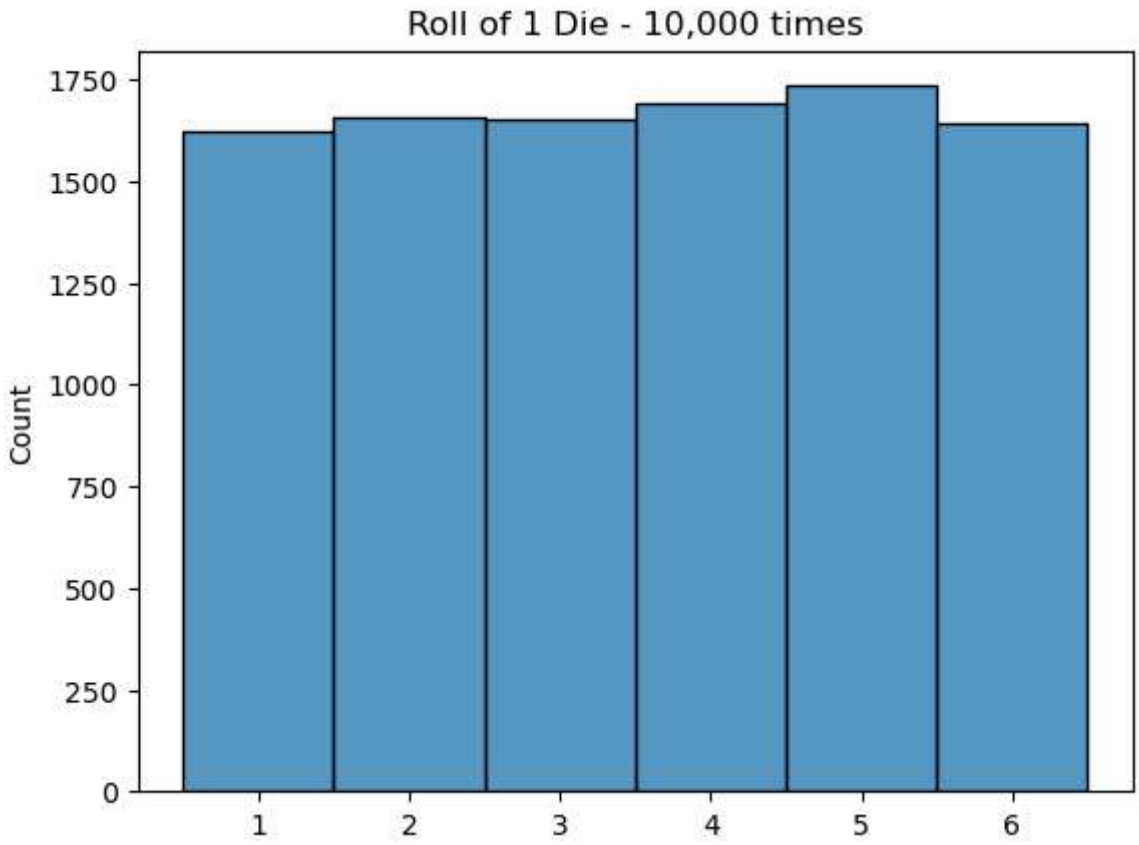


Simulate Dice Rolls

- Rolling one die gives outcomes between 1 and 6.
- Rolling two dice gives outcomes from 2 to 12 (by summing two die faces).
- Each sum has different probabilities.

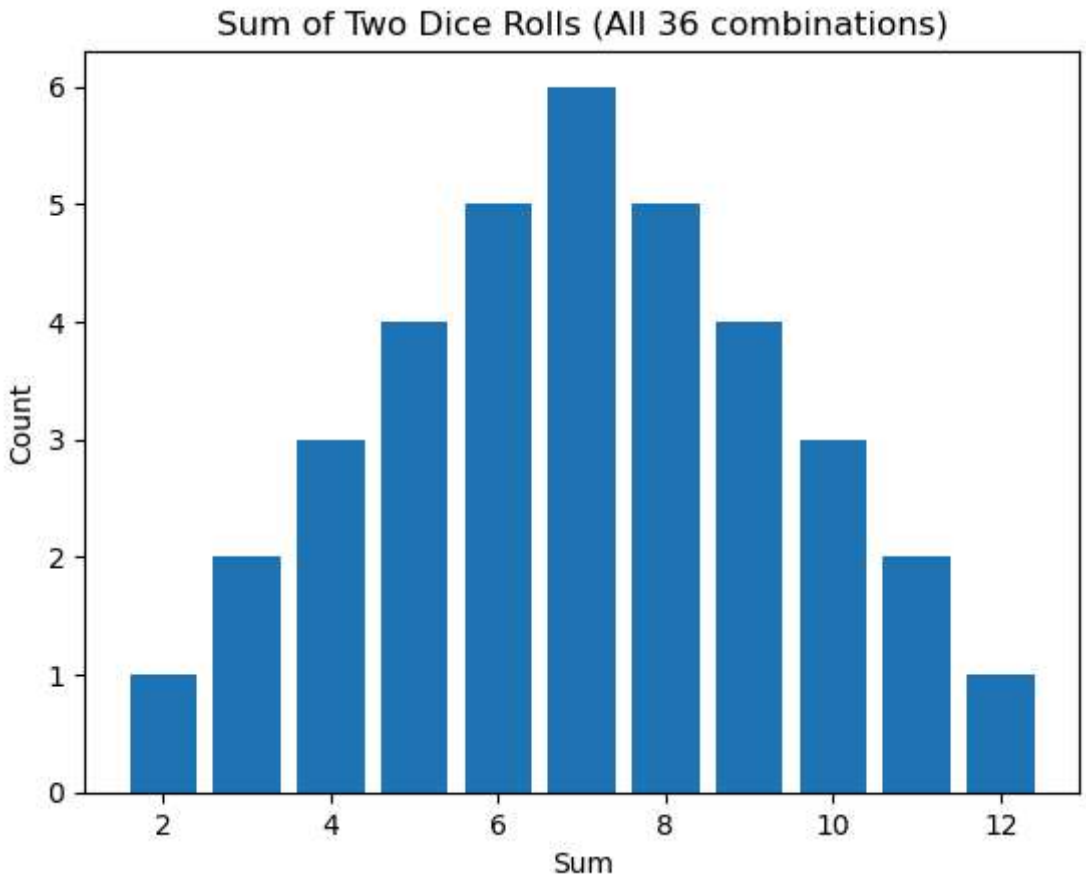
In [6]:
```
# One Die (10,000 rolls)

rolls = np.random.randint(1, 7, 10000)
sns.histplot(rolls, bins=6, discrete=True)
plt.title("Roll of 1 Die - 10,000 times")
plt.show()
```



In [5]:
```
# Two Dice Combinations

from collections import Counter
sums = [i+j for i in range(1, 7) for j in range(1, 7)]
count_sums = Counter(sums)
plt.bar(count_sums.keys(), count_sums.values())
plt.title("Sum of Two Dice Rolls (All 36 combinations)")
plt.xlabel("Sum")
plt.ylabel("Count")
plt.show()
```



Z-Score & Standardization

Z-Score Formula:

$$Z = \frac{X - \mu}{\sigma}$$

- X = value
- μ = mean
- σ = standard deviation

Z-score tells how far a value is from the mean in terms of standard deviations. Used in ML for scaling data (`StandardScaler`) or detecting outliers.

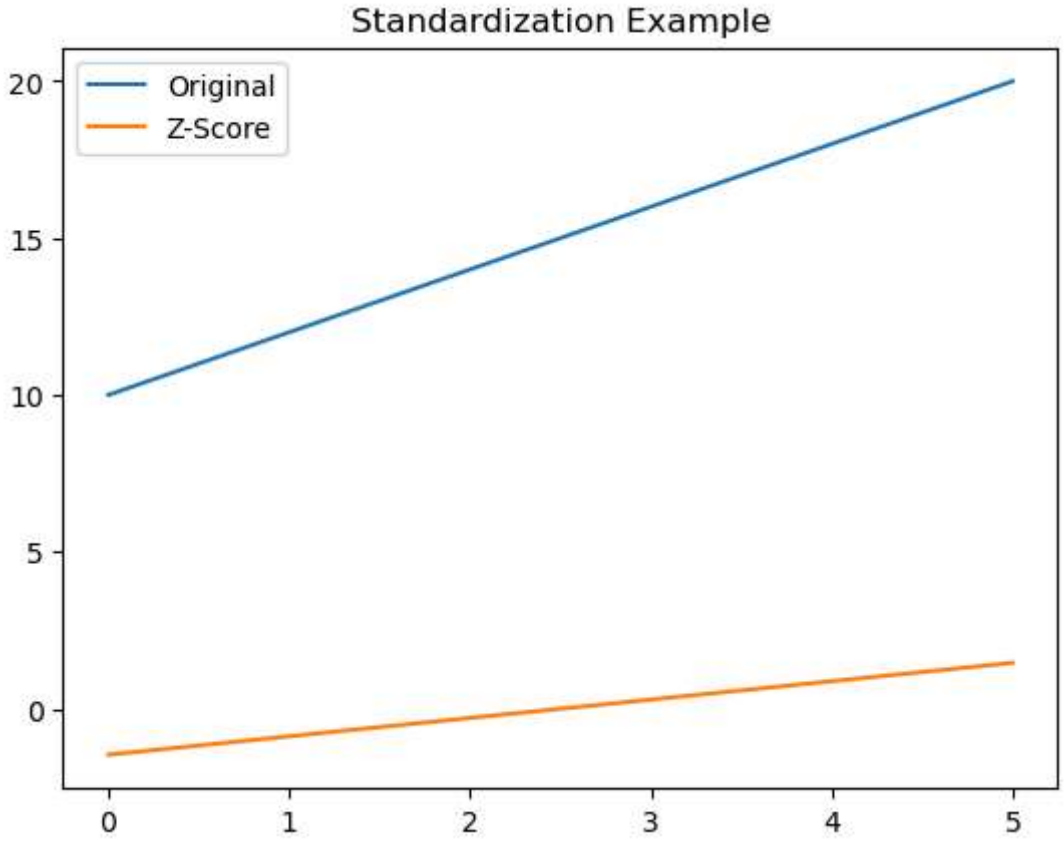
In [7]:
```
x = np.array([10, 12, 14, 16, 18, 20])
mean = x.mean()
std = x.std()
z_scores = (x - mean) / std

print("Original Data:", x)
print("Z-scores:", z_scores)

plt.plot(x, label="Original")
plt.plot(z_scores, label="Z-Score")
plt.legend()
```

```
plt.title("Standardization Example")
plt.show()
```

```
Original Data: [10 12 14 16 18 20]
Z-scores: [-1.46385011 -0.87831007 -0.29277002  0.29277002  0.87831007  1.46385011]
```



Small Dataset Example

Take a sample:

$$12, 14, 15, 16, 19, 22$$

- Mean ($\bar{x}$) = Add all values / 6
- Standard Deviation (s) = square root of variance
- Standard Error (SE) = $\frac{s}{\sqrt{n}}$

Why standard error is useful? It tells how much our sample mean is likely to differ from the true population mean.

In [8]:
```
sample = np.array([12, 14, 15, 16, 19, 22])
mean = np.mean(sample)
std = np.std(sample, ddof=1)  # sample std
df = len(sample) - 1
SE = std / np.sqrt(len(sample))

print("Mean =", mean)
```

```
print("Standard Deviation =", std)
print("Standard Error =", SE)
```

```
Mean = 16.333333333333332
Standard Deviation = 3.614784456460256
Standard Error = 1.475729574745244
```

Confidence Interval (CI)

What is a Confidence Interval? A confidence interval gives a range of values within which we expect the true population parameter (like the mean) to lie.

It answers the question: "How sure are we that our sample statistic (like mean) is close to the population mean?"

Instead of giving one value, it gives a **range** and a **confidence level**, such as 95% or 99%.

When to Use CI?

- When we want to **estimate** a population parameter using sample data
- In business, medicine, education, and research to express **uncertainty**

Manual Formula:

$$CI = \bar{x} \pm Z \cdot \frac{\sigma}{\sqrt{n}}$$

- $\bar{x}$: sample mean
- Z : critical value from Z-table (e.g. 1.96 for 95%)
- σ: std deviation (use s if population std unknown)
- n: sample size

Example (Manual Steps):

- Sample mean ($\bar{x}$) = 16.33
- Sample std deviation (s) = 3.61
- Sample size (n) = 6
- Standard Error = $\frac{3.61}{\sqrt{6}} = 1.47$
- Z for 95% = 1.96
- Margin of Error = $1.96 \times 1.47 = 2.88$
- CI = $16.33 \pm 2.88 = [13.45,\ 19.21]$

In [13]:
```
Z = 1.96
lower = mean - Z * SE
upper = mean + Z * SE
print(f"95% CI: [{lower:.2f}, {upper:.2f}]")
```

```
95% CI: [13.44, 19.23]
```

In [10]:
```
# Output explanation
print("This means we are 95% confident that the true population mean lies between",
```

```
This means we are 95% confident that the true population mean lies between 13.44 and
19.23
```

Point Estimation vs Interval Estimation

Point Estimation:

- It gives a single best guess of a population parameter.
- For example, if a sample has a mean of 70, then 70 is the point estimate for the population mean.

Interval Estimation:

- Instead of a single number, we provide a range that is likely to contain the population parameter.
- This range is calculated using confidence intervals (e.g., 95% CI = [68, 72]).

Why use interval estimation? Because point estimates don't express uncertainty. Interval estimation accounts for sample variability and gives us a confidence level.

Example

Point Estimate: Average score = 75
Interval Estimate (95% CI):

$$72, 78$$

Real-life CI Example: Board Exam Marks

You studied hard for your board exam and expect a high score. You estimate:

- **With 95% confidence**: Your score will lie between 95 and 100.
- **With 99% confidence**: Your score will lie between 94 and 100.

This is called a **confidence interval**. The wider the confidence level (like 99%), the wider the range because we want to be more certain.

Interpretation

- 95% confidence does NOT mean there's a 95% chance your score is in that interval.
- It means: If we took 100 such samples and built 100 CIs, about 95 of them would contain the true value.

Business Case: ABC Jeans Company

Scenario

ABC Jeans is launching in 3 countries (India, US, Australia). They want to decide pricing based on customer preference.

They gather small samples in each region:

- Average price willing to pay (from sample)
- But the real question: What is the **likely price preference of the whole population**?

What Should They Do?

- Use **interval estimation** via **confidence intervals**
- Use **T-test** (because population standard deviation is unknown and sample size is small)
- They can check if the average preferred price is statistically above a profitable threshold

This way, they make data-driven pricing decisions.

Z-test vs T-test

Both tests are used to compare a sample mean with a known or hypothesized population mean. The key difference lies in **what you know about the population**:

Test	Use When	Population Std (σ)	Sample Size
Z-test	Population variance is known	Known	Large $n \geq 30 preferred$
T-test	Population variance is unknown	Unknown	Small $n < 30$

Z-test Formula

$$Z = \frac{\bar{x} - \mu}{\sigma / \sqrt{n}}$$

- $\bar{x}$ = sample mean
- μ = population mean
- σ = population standard deviation
- n = sample size

T-test Formula

$$t = \frac{\bar{x} - \mu}{s/\sqrt{n}}$$

- s = sample standard deviation

Summary*

- Use **Z-test** for known variance
- Use **T-test** for unknown variance (real-world default)

Z-Table and T-Table

What Are They?

These are statistical tables that give **critical values** used to calculate margins of error or to test hypotheses.

- **Z-table**: Used with standard normal distribution (mean = 0, std = 1)
- **T-table**: Used when std deviation is estimated from the sample; depends on degrees of freedom (df = n - 1)

Common Values

Confidence Level	Z-value	T-value (df = 5)
90%	1.645	2.015
95%	1.96	2.571
99%	2.576	4.032

These values help build confidence intervals and calculate significance in hypothesis testing.

Perform Z-Test and T-Test in Python

Example : One-Sample Z-Test

In [11]:
```
import numpy as np
from scipy import stats

sample = np.array([75, 77, 74, 73, 78, 76, 77])
pop_mean = 70
pop_std = 5
```

```
n = len(sample)
mean_sample = np.mean(sample)

# Z-test statistic
z_stat = (mean_sample - pop_mean) / (pop_std / np.sqrt(n))
# P-value
p_value_z = 1 - stats.norm.cdf(z_stat)

print("Z-Statistic:", round(z_stat, 2))
print("P-Value:", round(p_value_z, 4))
```

```
Z-Statistic: 3.02
P-Value: 0.0012
```

Interpretation

- If P-value < 0.05 ⇒ reject the null hypothesis
- The sample mean is **significantly different** from the population mean

Example : One-Sample T-Test (when std unknown)

In [12]:
```
# T-test
t_stat, p_value_t = stats.ttest_1samp(sample, popmean=70)
print("T-Statistic:", round(t_stat, 2))
print("P-Value:", round(p_value_t, 4))
```

```
T-Statistic: 8.4
P-Value: 0.0002
```

Summary

In this combined notebook, we:

- Learned how inferential statistics helps us predict about a population using a sample
- Covered basics of probability using coin toss and dice
- Simulated dice rolls to visualize probability distributions
- Understood and calculated Z-score and standardized data
- Explored standard deviation, standard error, and practiced these with a small dataset
- Learned what Confidence Interval (CI) means, where to use it, and how to calculate it both manually and using Python
- Understood the difference between point and interval estimates
- Explored confidence interval through a relatable board exam example
- Applied business inference to ABC Company case
- Learned when to use Z-test vs T-test with clear logic
- Read and interpreted values from Z and T tables
- Practiced both Z-test and T-test in Python with interpretation