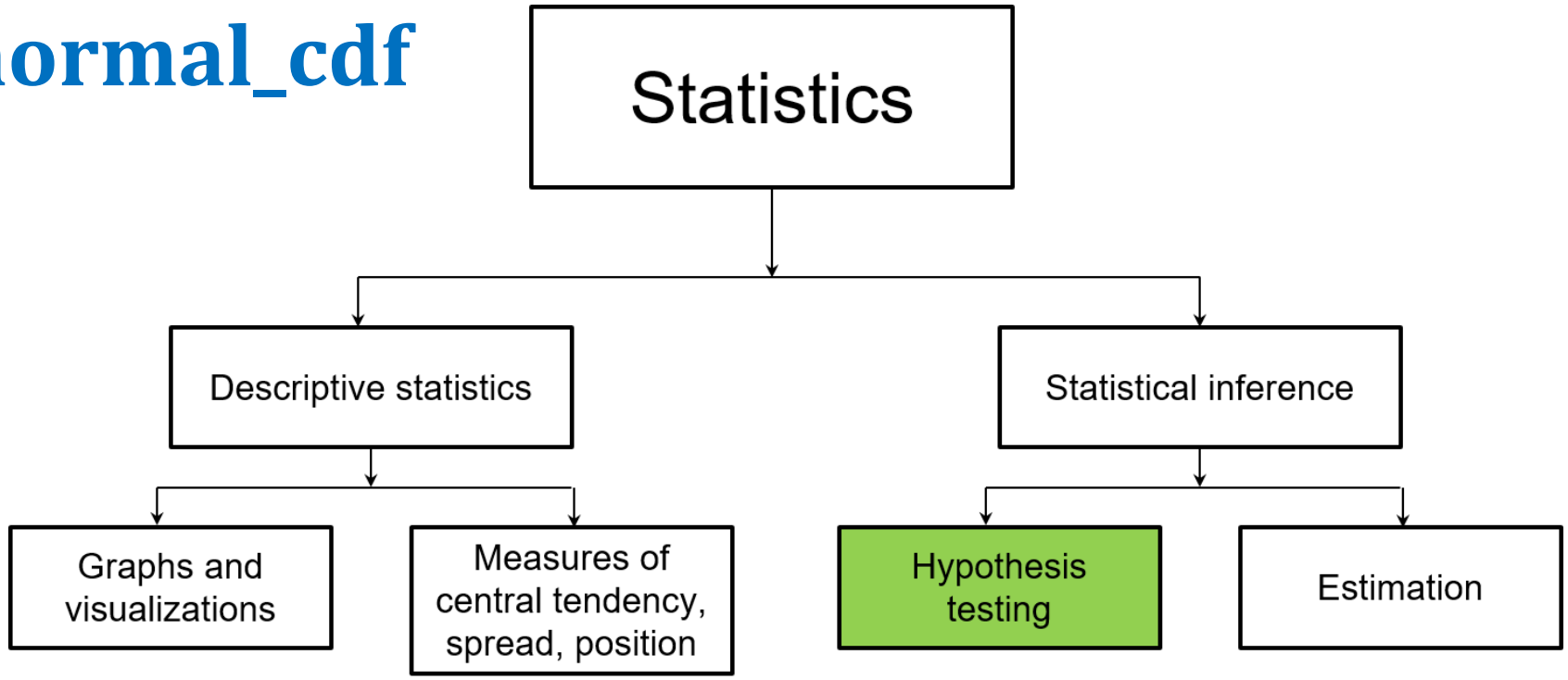
Syllabus

Unit II

Statistical Hypothesis Testing, Example: Flipping a Coin, p-Values, Confidence Intervals, p-Hacking, Example: Running an A/B Test, Bayesian Inference, Gradient Descent, The Idea Behind Gradient Descent Estimating the Gradient, Using the Gradient, Choosing the Right Step Size, Using Gradient Descent to Fit Models, Minibatch and Stochastic Gradient Descent.

normal_cdf



- Hypothesis testing plays a crucial role in statistical analysis and decision-making.
- By providing a structured framework for evaluating assumptions and drawing conclusions, it enables researchers and data analysts to make informed choices based on evidence.
- Two key concepts underpin hypothesis testing: **the null hypothesis and the alternative hypothesis.**

Statistical Hypothesis Testing

- Often, as data scientists, want to test whether a certain hypothesis is likely to be true.
- For our purposes, hypotheses are assertions like “this coin is fair” or “data scientists prefer Python to R” or “people are more likely to navigate away from the page without ever reading the content if we pop up an irritating interstitial advertisement with a tiny, hard-to-find close button” that can be translated into statistics about data.
- In the classical setup, we have a null hypothesis, H_0, that represents some default position, and some alternative hypothesis, H_1, that we’d like to compare it with.
- We use statistics to decide whether we can reject H0 as false or not.

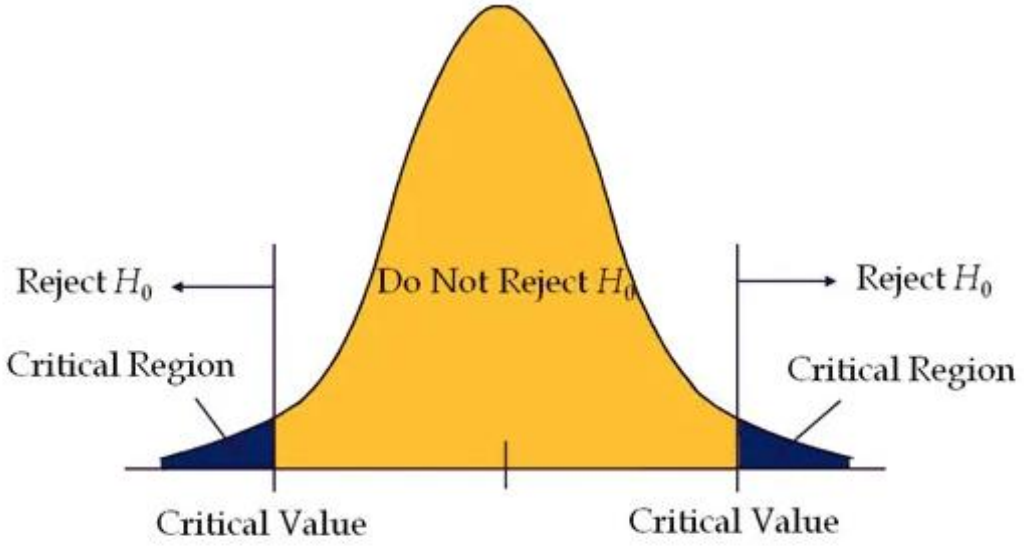
Example: Flipping a Coin

- Imagine we have a coin and we want to test whether it's fair.
- We'll make the assumption that the coin has some probability p of landing heads, and so our **null hypothesis is that the coin is fair—that is, that p = 0.5.**
- We'll test this against the **alternative hypothesis p ≠ 0.5.**
- Test will involve **flipping the coin some number, n,** times and counting the number of heads, X.
- Each coin flip is a Bernoulli trial, which means that X is a Binomial(n, p) random variable, we can approximate using the normal distribution:

```
def normal_approximation_to_binomial(n, p):
    """finds mu and sigma corresponding to a Binomial(n, p)"""
    mu = p * n
    sigma = math.sqrt(p * (1 - p) * n)
    return mu, sigma
```

Methods to Calculate Hypothesis Testing:

Rejection Region Approach:



In a hypothesis test, critical regions are ranges of the distributions where the values represent statistically significant results. If the test statistic falls in the critical region, reject the null hypothesis.

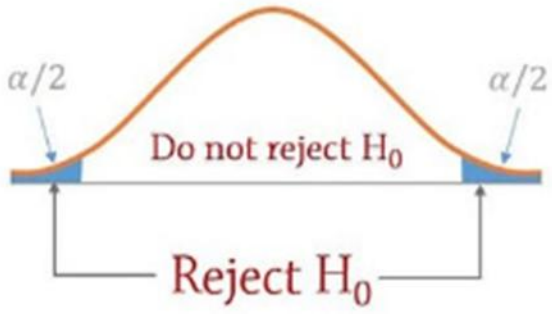
The rejection region approach is one of the commonly used methods to conduct hypothesis testing. It involves determining a critical region or rejection region, which serves as a threshold for making decisions based on test statistics.

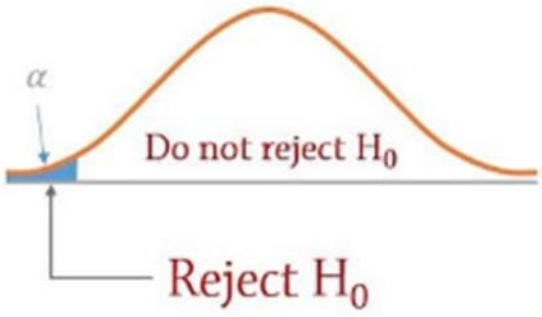
1. Setting Up the Hypothesis Test

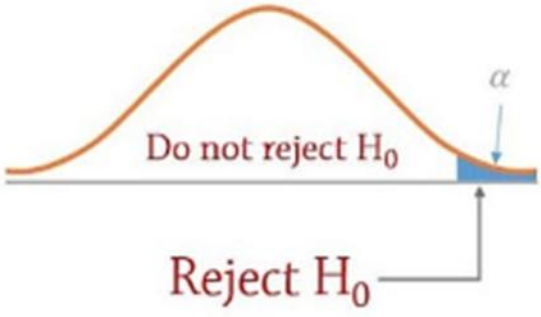
- **Null Hypothesis (H_0)**: The coin is fair (unbiased), meaning it has a 50% probability of landing on heads or tails.
- **Alternative Hypothesis (H_1)**: The coin is biased, meaning it does not have a 50% probability of landing on heads or tails.

2. Choosing the Significance Level (α)

- The significance level (α) represents the probability of rejecting H_0 when it is actually true (Type I error).
- You have set **α = 0.05 (5%)**, which is a common threshold in hypothesis testing.







Hypothesis Testing: Flipping Coin Example

3. Interpreting the p-value (2.28%)

- The **p-value (0.0228 or 2.28%)** is the probability of observing a result as extreme as (or more extreme than) the one obtained, assuming H_0 is true.
- If **p-value** $\leq \alpha$, we reject H_0.
- If **p-value** $> \alpha$, we fail to reject H_0.

Since **2.28% (p-value) is less than 5% (α)**, we **reject H_0**.

4. Conclusion: The Coin Might Be Biased

- Rejecting H_0 suggests that the observed data provides strong enough evidence against the assumption that the coin is fair.
- This does **not** prove the coin is biased with absolute certainty, but it strongly suggests that it is.

normal_cdf

- Whenever a random variable follows a normal distribution, we can use ***normal_cdf*** to figure out the probability that its realized value lies within or outside a particular interval:
- **Python program using normal_cdf to figure out that the probability value lies within or outside a particular interval.**

```
from math import erf, sqrt
from typing import Union
def normal_cdf(x: float, mu: float = 0, sigma: float = 1) -> float:
    """Computes the CDF of a normal distribution."""
    return (1 + erf((x - mu) / (sigma * sqrt(2)))) / 2
# The normal_cdf is the probability the variable is below a threshold
normal_probability_below = normal_cdf

# It's above the threshold if it's not below the threshold
def normal_probability_above(lo: float, mu: float = 0, sigma: float = 1)
-> float:
    """The probability that an N(mu, sigma) is greater than lo."""
    return 1 - normal_cdf(lo, mu, sigma)
# It's between if it's less than hi, but not less than lo
def normal_probability_between(lo: float, hi: float, mu: float = 0,
sigma: float = 1) -> float:
    """The probability that an N(mu, sigma) is between lo and hi."""
    return normal_cdf(hi, mu, sigma) - normal_cdf(lo, mu, sigma)
```

normal_cdf

```
# It's outside if it's not between
def normal_probability_outside(lo: float, hi: float, mu: float = 0,
sigma: float = 1) -> float:
    """The probability that an N(mu, sigma) is not between lo and
hi."""
    return 1 - normal_probability_between(lo, hi, mu, sigma)
print(normal_probability_below(1.0))  # Probability that X < 1
(assuming standard normal distribution)
print(normal_probability_above(1.0))  # Probability that X > 1
print(normal_probability_between(-1.0, 1.0))  # Probability that -1 < X
< 1
print(normal_probability_outside(-1.0, 1.0))  # Probability that X is
not between -1 and 1
```

normal_cdf : flipping Coin Example

Compute the Normal Approximation

Using function:

```
def normal_approximation_to_binomial(n: int, p: float) -> Tuple
[float, float]:
  """Returns mu and sigma corresponding to a Binomial(n, p)"""
  mu = p * n
  sigma = math.sqrt(p* (1-p) * n)
  return mu, sigma
```

mu, sigma = normal_approximation_to_binomial(100, 0.5)
print(mu, sigma) # Output: 50.0, 5.0
So, under H0, the expected number of heads is **50**, with a standard deviation of **5**.

Compute the Probability of Getting 60 or More Heads

We calculate:

Using the normal CDF:

probability = normal_probability_above(60, mu, sigma)
print(probability) # Output: 0.0228
This means that if the coin were fair, there would be **only a 2.28% chance** of getting 60 or more heads.

Inverse normal_cdf

- Python program using Inverse normal_cdf to find either the nontail region or the (symmetric) interval around the mean that accounts for a certain level of likelihood given the probability

```
def normal_upper_bound(probability: float, mu: float = 0, sigma: float = 1) ->
float:
  """Returns the z for which P(Z <= z) = probability"""
  return inverse_normal_cdf(probability, mu, sigma)
def normal_lower_bound(probability: float, mu: float = 0, sigma: float = 1) ->
float:
  """Returns the z for which P(Z >= z) = probability"""
  return inverse_normal_cdf(1 - probability, mu, sigma)

def normal_two_sided_bounds(probability: float, mu: float = 0, sigma: float = 1):
   """Returns the symmetric (about the mean) bounds that contain the specified
probability"""
  # Compute the tail probability before using it
  tail_probability = (1 - probability) / 2
  print(tail_probability)
  # Now use tail_probability in functions
  upper_bound = normal_lower_bound(tail_probability, mu, sigma)
  lower_bound = normal_upper_bound(tail_probability, mu, sigma)
  return lower_bound, upper_bound
mu,sigma=normal_approximation_to_binomial(1000,0.5)
print(mu,sigma)
lower,upper=normal_two_sided_bounds(0.95,mu,sigma)
```

- In particular, let's say that we choose to flip the coin n = 1,000 times.
- If our hypothesis of fairness is true, X should be distributed approximately normally with mean 500 and standard deviation 15.8:

```
mu_0, sigma_0 = normal_approximation_to_binomial(1000, 0.5)

```

- We need to make a decision about significance—how willing we are to make a type 1 error ("false positive"), in which we reject H0 even though it's true.
- For reasons lost to the annals of history, this willingness is often set at 5% or 1%. Let's choose 5%.
- Consider the test that rejects H0 if X falls outside the bounds given by:

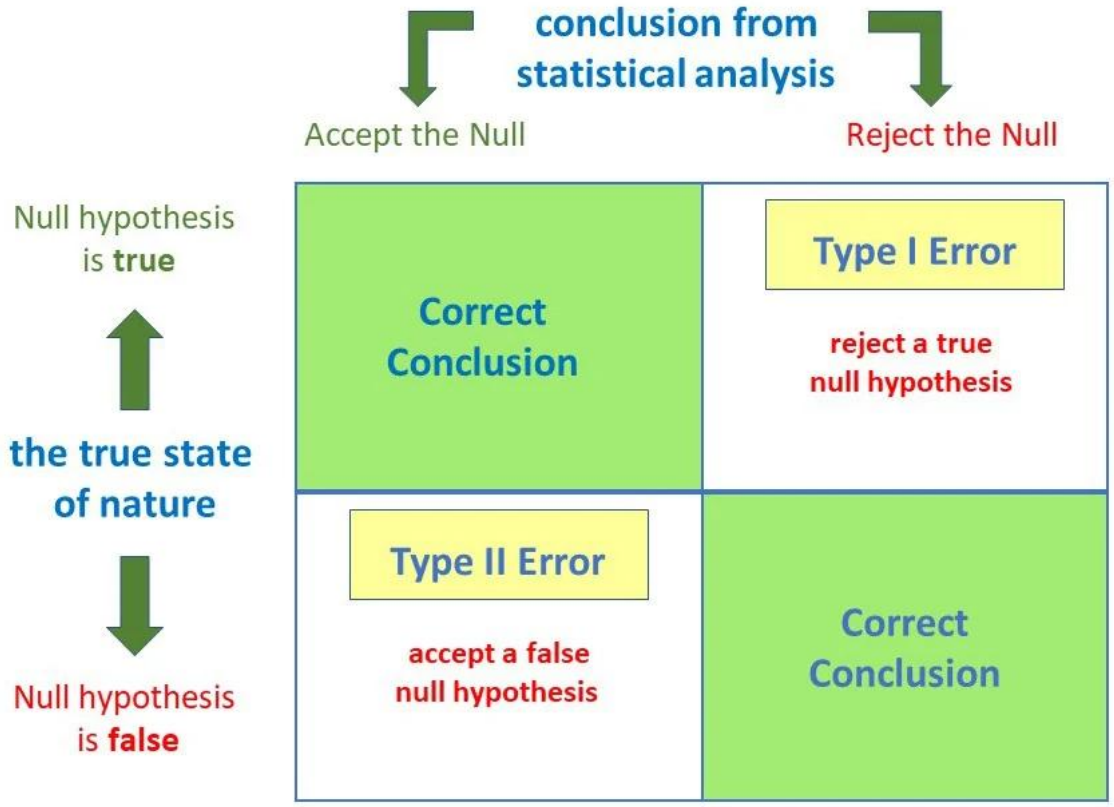
```
# (469, 531)
lower_bound, upper_bound = normal_two_sided_bounds(0.95, mu_0, sigma_0)

```

- Assuming p really equals 0.5 (i.e., H0 is true), there is just a 5% chance we observe an X that lies outside this interval, which is the exact significance we wanted.

Types of Errors

Type 1 and Type 2 errors are inherent risks in hypothesis testing.

- **Type I error** means rejecting the null hypothesis when it's actually true,
- **Type II error** means failing to reject the null hypothesis when it's actually false



Types of Errors

A **Type 1 error** occurs when we reject a true null hypothesis (H_0). Let's apply this to a **coin tossed 1,000 times** scenario.

Hypothesis Setup

- **H_0 (Null Hypothesis):** The coin is fair (P(Heads) = 0.5).
- **H_1 (Alternative Hypothesis):** The coin is biased (P(Heads) ≠ 0.5).
- **Significance Level (α):** 0.05 (5%)

Scenario

- We toss the coin **1,000 times** and record the number of heads.
- If the coin is truly fair, the expected number of heads is **500** (since 1000 × 0.5 = 500).
- However, due to randomness, we may get slightly different results, say **530 or 470 heads**, which might still be reasonable for a fair coin.
- Suppose we set a decision rule:
  - If the number of heads is **outside the range 470–530**, we reject H_0 and conclude the coin is biased.

Types of Errors

Type 1 Error (False Positive)

- The coin is actually **fair** (H_0 is true).
- Due to randomness, we observe **540 heads** (which is outside the accepted range of 470–530).
- We **incorrectly reject H_0** and conclude the coin is biased.
- **This is a Type 1 error because we rejected a true null hypothesis.**

Type 2 Error (False Negative)

- The coin is actually **biased** (H_0 is false), and suppose P(Heads) = **0.55** instead of 0.5.
- The expected number of heads is **550** (since 1000 × 0.55 = 550).
- However, we happen to observe **525 heads**, which **falls within our acceptance range (470–530).**
- We **fail to reject H_0** and incorrectly conclude that the coin is fair.
- **This is a Type 2 error because we failed to detect a biased coin when it was actually biased.**

Power of the test

```python
# 95% bounds based on assumption p is 0.5
lo, hi = normal_two_sided_bounds(0.95, mu_0, sigma_0)

# actual mu and sigma based on p = 0.55
mu_1, sigma_1 = normal_approximation_to_binomial(1000, 0.55)

# a type 2 error means we fail to reject the null hypothesis,
# which will happen when X is still in our original interval
type_2_probability = normal_probability_between(lo, hi, mu_1, sigma_1)
power = 1 - type_2_probability  # 0.887
print(power)

```

✓ 0.6s

```
0.8865480012953671
```

Power of a Hypothesis Test

The **power of a test** is the probability of correctly rejecting the null hypothesis (H_0) when the alternative hypothesis (H_1) is true. In other words, it measures how well the test detects a real effect (such as a biased coin).

$$\text{Power} = 1 - P(\text{Type 2 Error})$$

Since a **Type 2 error (β)** occurs when we fail to reject H_0 even though it is false, the power of the test is:

$$\text{Power} = 1 - \beta$$

Power Calculation for the Coin Toss Example (n = 1000, p = 0.55)

From our previous calculation:

- **Type 2 error probability (β) ≈ 0.102 (10.2%)**

Thus, the power of the test is:

$$\text{Power} = 1 - 0.102 = 0.898$$

So, the **power of this test is 89.8%**, meaning there is an **89.8% probability that we correctly detect the biased coin** when P(Heads) = 0.55.

p-Values

1. What is a p-Value?

A **p-value** (probability value) is a measure used in **hypothesis testing** to determine the strength of evidence against the **null hypothesis** (H_0). It answers the question:

> **"If the null hypothesis were true, what is the probability of observing a test statistic at least as extreme as the one obtained?"**

- **A small p-value (typically ≤ 0.05)** indicates **strong evidence against** H_0, leading us to **reject the null hypothesis**.
- **A large p-value (> 0.05)** suggests **weak evidence against** H_0, meaning we **fail to reject** H_0.

p-value ≤ 0.05 → Reject H_0 (Strong evidence against H_0).

p-value > 0.05 → Fail to reject H_0 (Not enough evidence to conclude H_1).

p-Values

Feature	Hypothesis Testing	p-Value
Definition	A full statistical method to test a claim	A probability that helps in decision-making
Purpose	To decide whether to reject H_0 or not	To measure how extreme the observed data is under H_0
Includes	Formulating hypotheses, computing test statistics, making decisions	Only a probability calculation

p-Values

- For our two-sided test of whether the coin is fair, we compute:
- **Python code snippet for two sided test to check whether coin is fair using p-values.**

```python
def two_sided_p_value(x: float, mu: float = 0, sigma: float = 1) -> float:
    """How likely are we to see a value at least as extreme as x (in either
    direction) if our values are from an N(mu, sigma)? """
    if x >= mu:
        # x is greater than the mean, so the tail is everything greater than x
        return 2 * normal_probability_above(x, mu, sigma)
    else:
        # x is less than the mean, so the tail is everything less than x
        return 2 * normal_probability_below(x, mu, sigma)


print(two_sided_p_value(529.5, mu_0, sigma_0))  # 0.062
```

✓ 0.1s

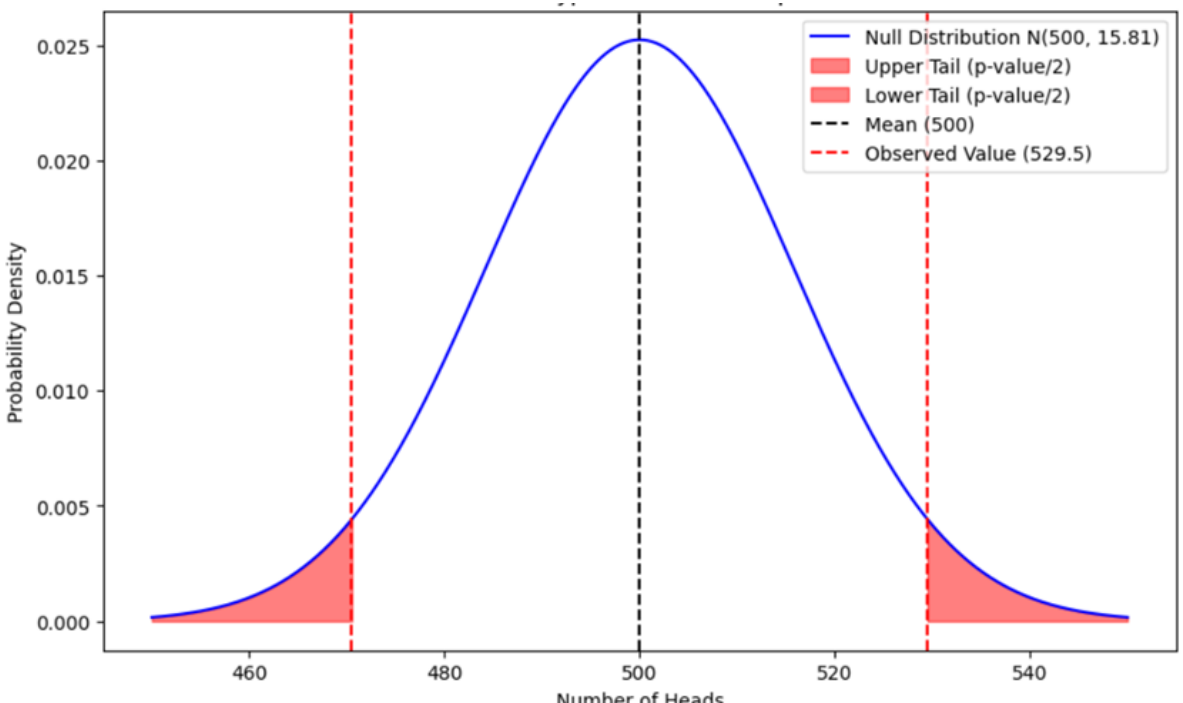
```
0.06207721579598835
```

Scenario: Testing If a Coin Is Biased

Suppose we flip a coin **1,000 times** and observe **570 heads**.

We want to test whether the coin is fair:

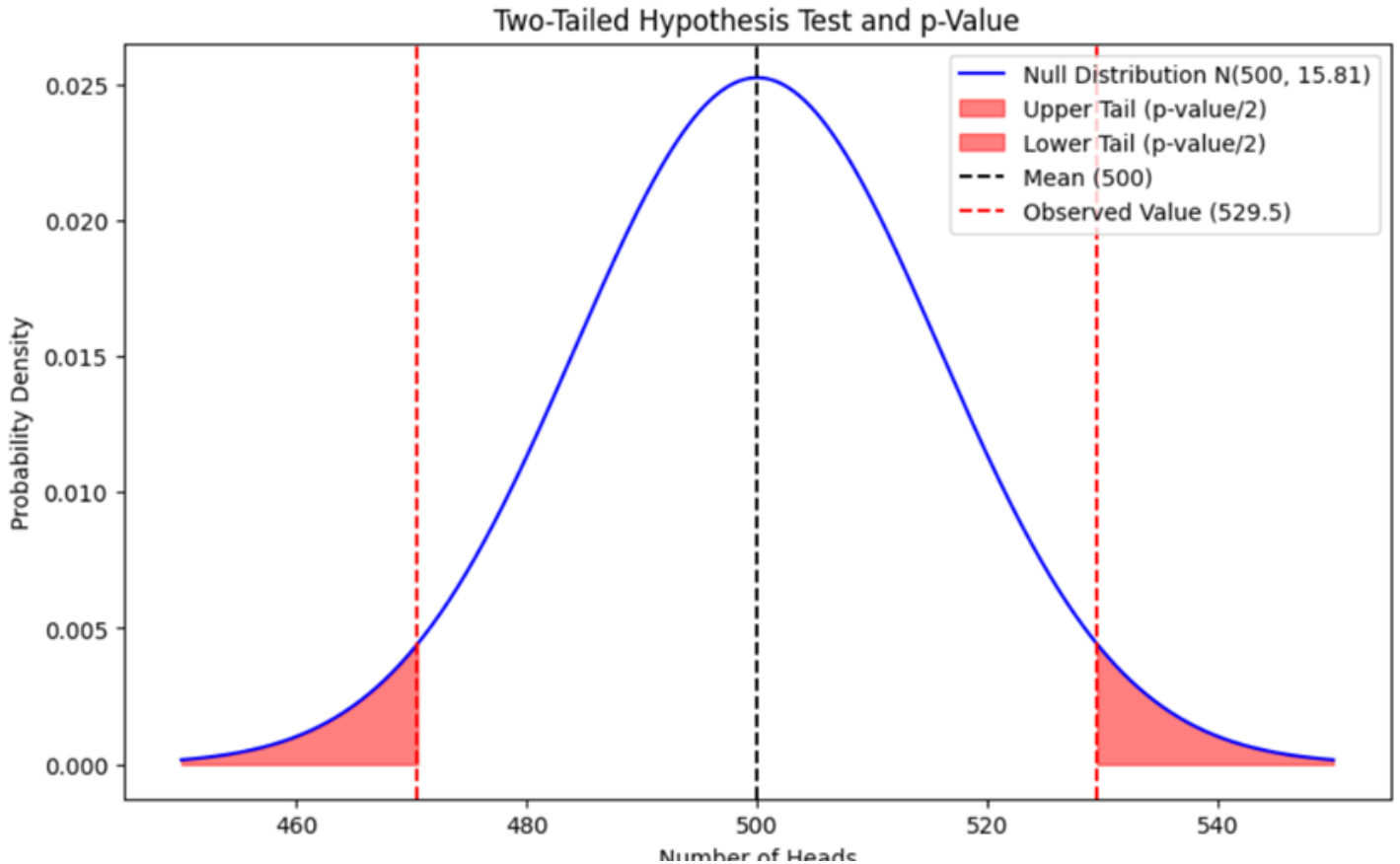
- **H_0 (Null Hypothesis):** The coin is fair → $P(\text{Heads}) = 0.5$.
- **H_1 (Alternative Hypothesis):** The coin is biased → $P(\text{Heads}) \neq 0.5$.
- **p-value** = **0.000018** (very small, much less than 0.05).
- This means that **if the coin were fair**, the probability of observing **570 heads or more (or 430 heads or fewer)** is **0.0018%**.
- Since this is an **extremely unlikely** event under H_0, we **reject H_0**.
- **Conclusion: The coin is biased**—there is **strong evidence against fairness.**



```
print(two_sided_p_value(529.5, mu_0, sigma_0))  # 0.062
```

- The p-value is **0.062 (6.2%)**.
- This means that **if H_0 were true**, the probability of observing a result as extreme as **529.5** (in either direction) is **6.2%**.
- Since **0.062 > 0.05**, we **fail to reject H_0** at the 5% significance level.

Conclusion: There is **not enough evidence** to say the coin is biased.



p-Values

- One way to convince yourself that this is a sensible estimate is with a simulation:

```python
import random
extreme_value_count = 0
for _ in range(1000):
    num_heads = sum(1 if random.random() < 0.5 else 0  # Count # of heads
                    for _ in range(1000))  # in 1000 flips,
    if num_heads >= 530 or num_heads <= 470:  # and count how often
        extreme_value_count += 1  # the # is 'extreme'

# p-value was 0.062 => ~62 extreme values out of 1000
if 59 < extreme_value_count < 65:
    print(f"{extreme_value_count}")

```

✓ 0.1s

```
63
```

Confidence Intervals

- We've been testing hypotheses about the value of the heads probability p, which is a parameter of the unknown "heads" distribution.
- Another approach is to construct a **confidence interval around the observed value of the parameter.**
- A **confidence interval (CI)** is a range of values that is likely to contain the **true population parameter** (e.g., mean, proportion) with a certain level of confidence (usually **95% or 99%**).
- In hypothesis testing, confidence intervals help us determine:
  - Whether we **reject or fail to reject H_0**.
  - The **uncertainty** in our estimate of the population parameter.

Confidence Intervals

- we observe 525 heads out of 1,000 flips, then we estimate *p* equals 0.525:
- **Python code snippet to find confidence Intervals for 525 heads observed out of 1000 coin flips**

```
p_hat = 525 / 1000
mu = p_hat
sigma = math.sqrt(p_hat * (1 - p_hat) / 1000) # 0.0158
normal_two_sided_bounds(0.95, mu, sigma)
# [0.4940, 0.5560]
```

A confidence interval is a range of values in which we are fairly confident the true population parameter lies. If 0.5 is within this range, it suggests that the sample result (525 heads out of 1000 flips) is close enough to 0.5 to be considered plausible.

'525 heads out of 1000 is not significantly different from a fair coin at the 5% significance level":

Confidence Intervals

- If instead we'd seen 540 heads, then we'd have:

Python code snippet to find confidence Intervals for 540 heads observed out of 1000 coin flips

```
p_hat = 540 / 1000
mu = p_hat
sigma = math.sqrt(p_hat * (1 - p_hat) / 1000) # 0.0158
normal_two_sided_bounds(0.95, mu, sigma)
# [0.5091, 0.5709]
```

Confidence Interval for 540 Heads Out of 1000 Flips

- 95% Confidence Interval: (0.509, 0.571)
- We are 95% confident that the true probability of heads lies between 50.9% and 57.1%.

p-Hacking

- P-hacking, also known as **data snooping**, is the misuse of statistical analysis to find patterns in data that appear significant, even if they are not, by manipulating data collection and analysis
- P-hacking involves strategies to manipulate data analysis to achieve statistically significant results ($p < 0.05$), even when no real effect exists.
- **Examples of p-hacking**:
  - **Selective reporting**: Only reporting results that show significance while ignoring non-significant findings.
  - **Data dredging**: Running multiple statistical tests on the same data and only reporting the ones that yield significant results.
  - **Outlier removal**: Removing data points that don't fit the desired outcome.
  - **Altering** sample size or statistical models after initial analyses.
- **Real-world analogy:**
- **Imagine flipping a coin 100 times in secret and only showing the part where it lands heads 8 out of 10 times.**

p-Hacking

Python code snippet to demonstrate p-Hacking

```
import random
from typing import List
def run_experiment() -> List[bool]:
    """Flips a fair coin 1000 times, True = heads, False = tails"""
    return [random.random() < 0.5 for _ in range(1000)]
def reject_fairness(experiment: List[bool]) -> bool:
    """Uses a 5% significance level (two-tailed test) to check if the coin is fair"""
    num_heads = len([flip for flip in experiment if flip])
    print(num_heads)
     return num_heads < 469 or num_heads > 531   # 5% significance range for 1000 flips
random.seed(0)  # For reproducibility
# Run 1000 experiments
experiments = [run_experiment() for _ in range(1000)]
# Count how many experiments "reject" the fair coin hypothesis
num_rejections = len([experiment for experiment in experiments if reject_fairness(experiment)])
print(f"Number of false positives (rejecting a fair coin): {num_rejections}")
```



```
Number of false positives (rejecting a fair coin): 46
```

p-Hacking

- **What it demonstrates**

This demonstration is a **simulation that explains a flaw in statistical hypothesis testing**—specifically, how **false positives** (Type I errors) can arise even when there is **no real effect**, and how this leads to **p-hacking**.

Each experiment uses a **truly fair coin**.

Still, due to **random variation**, some experiments produce results **just extreme enough to look unfair**.

Statistically, you expect about **5% of fair experiments to fall outside** the 95% confidence interval.

So:

$$5\% \times 1000 = 50 \text{ experiments}$$

will **falsely "reject" fairness**.

Researchers often **run many experiments** or **slice the data in many ways**.

If they test **enough times**, even **completely random data** can show something that looks "statistically significant."

This leads to **false discoveries**, **misleading conclusions**, and bad science.

Example: Running an A/B Test

- One of your primary responsibilities at Data Sciencester is experience optimization.
- One of the advertisers has developed a new energy drink targeted at data scientists, and the VP of Advertisements wants your help choosing between advertisement A ("tastes great!") and advertisement B ("less bias!").
- Being a scientist, you decide to run an experiment by randomly showing site visitors one of the two advertisements and tracking how many people click on each one.
- If 990 out of 1,000 A-viewers click their ad, while only 10 out of 1,000 B-viewers click their ad, you can be pretty confident that A is the better ad.

Example: Running an A/B Test

- But what if the differences are **not so stark**? Here's where you'd use statistical inference.
- Let's say that N_A people see ad A, and that n_A of them click it.
- We can think of each ad view as a Bernoulli trial where p_A is the probability that someone clicks ad A.
- $p_A = n_A/N_A$ is approximately a normal random variable with mean p_A

- standard deviation $\sigma_A = \sqrt{p_A(1-p_A)/N_A}$

Example: Running an A/B Test

- Similarly, n_B/N_B is approximately a normal random variable with mean p_B and standard deviation

$$\sigma_B = \sqrt{p_B(1-p_B)/N_B}$$

- We can express this in code as:

```
def estimated_parameters(N, n):
    p = n / N                    # observed proportion
    sigma = math.sqrt(p * (1 - p) / N)  # standard deviation of the proportion
    return p, sigma
```

This tells you how far apart the two proportions (p_B and p_A) are **in terms of standard deviation** units — basically, how surprising the difference is