📊 *Is the NBA Draft Lottery Rigged? A Statistical Deep Dive* 🧐

Every year, the NBA Draft Lottery fuels hope, dreams... and a fair share of conspiracy theories. In 2025, the fire was reignited when the Dallas Mavericks, with slim odds, surprisingly landed the number one overall pick. Social media exploded: "*It's rigged!*" – a phrase NBA fans know all too well.

But is there any statistical basis behind these claims?

🎯 Objective

This notebook aims to investigate whether the NBA Draft Lottery behaves in a way that deviates significantly from what probability theory would predict. We'll use statistical analysis, simulation models, and historical data to answer one question: **does the lottery follow a fair distribution, or are some results just too improbable to be random?**

🏀 What is the NBA Draft Lottery?

The NBA Draft Lottery determines the order of selection for the top picks in the annual NBA Draft. It was introduced in 1985 to discourage teams from intentionally losing games (a practice known as *tanking*) to secure better draft positions. Instead of guaranteeing the worst teams the top picks, the lottery introduces randomness—giving all non-playoff teams a shot, albeit weighted by record.

Currently, 14 ping-pong balls numbered 1 to 14 are used to generate 1,001 possible combinations. Of those, 1,000 are assigned among the 14 non-playoff teams based on their regular-season record. The first four picks are decided via lottery, and the rest follow in reverse order of the standings.

🕵️ A Brief History of Suspicion

From its inception, the lottery has been surrounded by skepticism. Here are a few infamous examples:

- **1985: The Frozen Envelope** – The New York Knicks won the first-ever lottery and selected Patrick Ewing. Conspiracy theorists suggest the Knicks' envelope was frozen or creased so NBA officials could easily identify it.
- **2003: The LeBron Sweepstakes** – The Cleveland Cavaliers, with lower odds, landed the hometown superstar.
- **2019: The Zion Lottery** – The New Orleans Pelicans, with only a 6% chance, landed the top pick and selected Zion Williamson, just months after trading Anthony Davis to the Los Angeles Lakers.

Each of these cases fueled accusations that the NBA may be nudging outcomes for financial, narrative, or marketability reasons.

But how likely are these "surprising" results, really? Could they just be the natural byproduct of randomness?

🎲 How the NBA Draft Lottery Works

The NBA Draft Lottery is designed to determine the order of selection for the top picks in the annual NBA Draft. Here's a breakdown of how the process works:

🔢 The Basics

- **14 ping-pong balls** numbered 1 through 14 are placed in a lottery machine.
- **Four balls** are drawn at random to create a four-number combination.
- The order of the numbers drawn doesn't matter; for example, 1-2-3-4 is considered the same as 4-3-2-1.
- The total number of possible combinations of 14 numbers taken 4 at a time, without regard to order, is calculated using the combination formula:

$$\binom{14}{4} = \frac{14!}{4!(14-4)!} = 1001$$

- One combination (specifically, 11-12-13-14) is disregarded, leaving **1,000 valid combinations**.

🏀 Team Odds

Each of the 14 non-playoff teams is assigned a specific number of these 1,000 combinations, based on their regular-season record. The worse the record, the more combinations (and thus higher odds) a team receives. Here's how the combinations are distributed:

Team Rank	Combinations	Percentage Chance
1st	140	14.0%
2nd	140	14.0%
3rd	140	14.0%
4th	125	12.5%
5th	105	10.5%
6th	90	9.0%
7th	75	7.5%
8th	60	6.0%
9th	45	4.5%
10th	30	3.0%
11th	20	2.0%
12th	15	1.5%
13th	10	1.0%
14th	5	0.5%

Note: In cases where teams have identical regular-season records, a coin flip is conducted to determine the allocation of combinations. For example, if the 11th and 12th worst teams are tied, their combined 35 combinations (20 + 15) are split 18 and 17 between them, based on the coin toss outcome.

🎯 The Drawing Process

1. **Four balls** are drawn to determine the first pick.
2. The process is repeated for the second, third, and fourth picks.
3. If a combination is drawn that belongs to a team already selected for a higher pick, the draw is discarded, and a new combination is drawn.
4. After the top four picks are determined, the remaining teams are slotted in order of their regular-season records, from worst to best.

📊 Probability Table

The following table illustrates the probability of each team obtaining each pick:

Seed	1st Pick	2nd Pick	3rd Pick	4th Pick	5th	6th	7th	8th	9th	10th	11th	12th	13th	14th
1	14.00%	13.42%	12.75%	11.97%	47.86%	-	-	-	-	-	-	-	-	-
2	14.00%	13.42%	12.75%	11.97%	27.84%	20.02%	-	-	-	-	-	-	-	-
3	14.00%	13.42%	12.75%	11.97%	14.84%	26.00%	7.02%	-	-	-	-	-	-	-
4	12.50%	12.23%	11.89%	11.46%	7.24%	25.74%	16.74%	2.19%	-	-	-	-	-	-
5	10.50%	10.54%	10.56%	10.53%	2.22%	19.61%	26.74%	8.68%	0.62%	-	-	-	-	-
6	9.00%	9.20%	9.41%	9.62%	-	8.62%	29.77%	20.55%	3.68%	0.15%	-	-	-	-
7	7.50%	7.80%	8.14%	8.52%	-	-	19.72%	34.11%	12.88%	1.30%	0.03%	-	-	-
8	6.00%	6.34%	6.74%	7.22%	-	-	-	34.47%	32.10%	6.75%	0.38%	<0.01%	-	-
9	4.50%	4.83%	5.23%	5.71%	-	-	-	-	50.72%	25.90%	3.01%	0.09%	<0.01%	
10	3.00%	3.27%	3.60%	4.01%	-	-	-	-	-	65.90%	18.99%	1.20%	0.02%	<0.01%
11	2.00%	2.20%	2.45%	2.76%	-	-	-	-	-	-	77.59%	12.60%	0.40%	<0.01%
12	1.50%	1.66%	1.86%	2.10%	-	-	-	-	-	-	86.10%	6.70%	0.07%	
13	1.00%	1.11%	1.25%	1.43%	-	-	-	-	-	-	-	92.88%	2.34%	
14	0.50%	0.56%	0.63%	0.72%	-	-	-	-	-	-	-	-	-	97.59%

Note: The probabilities for each pick beyond the fourth are determined by the inverse order of regular-season records and are not subject to the lottery.

By understanding the mechanics and probabilities of the NBA Draft Lottery, we can better assess whether the outcomes align with statistical expectations or if anomalies suggest potential irregularities.

🧪 Simulating a Single NBA Draft Lottery

To better understand the randomness and fairness of the NBA Draft Lottery, we simulate a single lottery draw exactly as it is conducted in reality.

🔧 How it works:

1. **Generate all valid combinations** of 4 ping-pong balls drawn from 14 (ignoring order). This gives 1,001 total combinations.
2. **Remove one unassigned combination** (`11, 12, 13, 14`), leaving 1,000 valid ones.
3. **Distribute combinations** among the 14 teams based on their seed:
   - Teams with the worst records receive the most combinations (e.g. 140 for seeds 1–3, 125 for seed 4, etc.).
4. **Randomly draw 4 balls**, sort them (since order doesn't matter), and repeat the process until four unique teams are selected.
5. **Assign picks 1 to 4** to the teams whose combinations are drawn.
6. This simulation only determines the top 4 picks — the remaining picks are assigned by default based on regular-season standings.

Let's simulate one lottery draw:

In [1]:
```python
import random
from collections import defaultdict
from itertools import combinations

# 1. Genera tutte le combinazioni possibili (senza ordine) di 4 numeri da 1 a 14
all_combinations = list(combinations(range(1, 15), 4))
assert len(all_combinations) == 1001

# 2. Rimuovi la combinazione non utilizzata (11, 12, 13, 14)
all_combinations.remove((11, 12, 13, 14))
```

```
# 3. Assegna le combinazioni alle 14 squadre secondo le probabilità standard
# Squadre in ordine dal peggior record (Seed 1) al migliore (Seed 14)
lottery_distribution = [140, 140, 140, 125, 105, 90, 75, 60, 45, 30, 20, 15, 10, 5]
team_ids = list(range(1, 15))  # team 1 to 14

# Assegna le combinazioni a ciascuna squadra
team_combinations = defaultdict(list)
index = 0
for team, count in zip(team_ids, lottery_distribution):
    for _ in range(count):
        team_combinations[team].append(all_combinations[index])
        index += 1

# 4. Estrai le 4 palline casuali, ordinale e verifica quale team ha quella combinazione
def draw_lottery(team_combinations):
    selected_teams = []
    used_combinations = set()

    while len(selected_teams) < 4:
        draw = tuple(sorted(random.sample(range(1, 15), 4)))
        if draw == (11, 12, 13, 14) or draw in used_combinations:
            continue  # Scarta la combinazione non valida o già estratta
        used_combinations.add(draw)

        # Trova a quale squadra appartiene la combinazione
        for team, combos in team_combinations.items():
            if draw in combos and team not in selected_teams:
                selected_teams.append(team)
                break

    return selected_teams

# Esegui una singola simulazione della lottery
drawn_teams = draw_lottery(team_combinations)
drawn_teams
```

Out[1]:
```
[1, 2, 10, 3]
```

🔁 Monte Carlo Simulation of the NBA Draft Lottery

To statistically analyze how the NBA Draft Lottery behaves over the long run, we simulate the entire lottery process **N** times using Monte Carlo simulation.

🛠️ Simulation Steps:

1. For each iteration:
   - Simulate the actual draw of 4 ping-pong balls.
   - Match the combination to a team (ensuring no repeats).
   - Assign the top 4 picks to the drawn teams.
   - Assign picks 5 to 14 based on the inverse of regular-season standings (excluding already drawn teams).
2. Track how often each team ends up in each pick position (1st through 14th).
3. Convert the counts into percentages to better understand the distribution.

This helps us answer questions like:

- Are the results consistent with the expected probabilities?
- How often do long-shot teams end up with a top pick?
- Do the worst teams actually get the best draft positions over many trials?

📊 Simulation Results

The table below shows the percentage of simulations (out of `N = 100,000`) in which each team (by seed) landed at each specific draft position.

In [2]:
```python
import pandas as pd
import numpy as np

def simulate_lottery(team_combinations, num_simulations=100000):
    # Dictionary to track how many times each team gets each pick (1st to 14th)
    pick_counts = {team: [0]*14 for team in team_ids}

    for _ in range(num_simulations):
        selected_teams = []
        used_combinations = set()
        picks = []

        # Draw the top 4 picks
        while len(selected_teams) < 4:
            draw = tuple(sorted(random.sample(range(1, 15), 4)))
            if draw == (11, 12, 13, 14) or draw in used_combinations:
                continue  # Skip invalid or duplicate combinations
            used_combinations.add(draw)

            for team, combos in team_combinations.items():
                if draw in combos and team not in selected_teams:
                    selected_teams.append(team)
                    break

        picks.extend(selected_teams)

        # Fill remaining picks (5th to 14th) based on inverse order of standings
        remaining_teams = [team for team in team_ids if team not in selected_teams]
        picks.extend(remaining_teams)

        # Register pick position for each team
        for position, team in enumerate(picks):
            pick_counts[team][position] += 1

    # Convert counts to percentages
    pick_percentages = {
        team: [round((count / num_simulations) * 100, 2) for count in counts]
        for team, counts in pick_counts.items()
    }

    # Create DataFrame
    df = pd.DataFrame.from_dict(pick_percentages, orient='index',
                                columns=[f'Pick {i+1}' for i in range(14)])
    df.index.name = 'Team (Seed)'
    return df

# Run the Monte Carlo simulation
simulation_results = simulate_lottery(team_combinations, num_simulations=100000)

# Display the result table
simulation_results
```

Out[2]:

Team (Seed)	Pick 1	Pick 2	Pick 3	Pick 4	Pick 5	Pick 6	Pick 7	Pick 8	Pick 9	Pick 10	Pick 11	Pick 12	Pick 13	Pick 14
1	14.08	13.34	12.85	11.98	47.74	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	14.08	13.40	12.73	12.08	27.86	19.84	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	13.73	13.41	12.91	11.92	15.04	25.98	7.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	12.65	12.14	11.80	11.42	7.14	25.90	16.72	2.22	0.00	0.00	0.00	0.00	0.00	0.00
5	10.58	10.54	10.41	10.47	2.21	19.66	26.97	8.57	0.58	0.00	0.00	0.00	0.00	0.00
6	8.98	9.20	9.26	9.65	0.00	8.61	29.69	20.73	3.73	0.14	0.00	0.00	0.00	0.00
7	7.56	7.78	8.14	8.56	0.00	0.00	19.62	34.05	12.93	1.32	0.03	0.00	0.00	0.00
8	5.90	6.36	6.83	7.28	0.00	0.00	0.00	34.42	32.09	6.72	0.41	0.00	0.00	0.00
9	4.54	4.92	5.29	5.66	0.00	0.00	0.00	0.00	50.67	25.87	2.97	0.09	0.00	0.00
10	2.94	3.31	3.57	4.01	0.00	0.00	0.00	0.00	0.00	65.94	18.98	1.23	0.02	0.00
11	2.01	2.23	2.52	2.76	0.00	0.00	0.00	0.00	0.00	0.00	77.61	12.46	0.40	0.00
12	1.47	1.65	1.77	2.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	86.22	6.75	0.07
13	0.99	1.15	1.23	1.44	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	92.83	2.35
14	0.48	0.57	0.68	0.69	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	97.57

In [3]:

```python
import pandas as pd

# Official probability matrix based on NBA rules
official_odds_data = [
    [14.00, 13.42, 12.75, 11.97, 47.86, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [14.00, 13.42, 12.75, 11.97, 27.84, 20.02, 0, 0, 0, 0, 0, 0, 0, 0],
    [14.00, 13.42, 12.75, 11.97, 14.84, 26.00, 7.02, 0, 0, 0, 0, 0, 0, 0],
    [12.50, 12.23, 11.89, 11.46, 7.24, 25.74, 16.74, 2.19, 0, 0, 0, 0, 0, 0],
    [10.50, 10.54, 10.56, 10.53, 2.22, 19.61, 26.74, 8.68, 0.62, 0, 0, 0, 0, 0],
    [9.00, 9.20, 9.41, 9.62, 0, 8.62, 29.77, 20.55, 3.68, 0.15, 0, 0, 0, 0],
    [7.50, 7.80, 8.14, 8.52, 0, 0, 19.72, 34.11, 12.88, 1.30, 0.03, 0, 0, 0],
    [6.00, 6.34, 6.74, 7.22, 0, 0, 0, 34.47, 32.10, 6.75, 0.38, 0.01, 0, 0],
    [4.50, 4.83, 5.23, 5.71, 0, 0, 0, 0, 50.72, 25.90, 3.01, 0.09, 0.01, 0],
    [3.00, 3.27, 3.60, 4.01, 0, 0, 0, 0, 0, 65.90, 18.99, 1.20, 0.02, 0.01],
    [2.00, 2.20, 2.45, 2.76, 0, 0, 0, 0, 0, 0, 77.59, 12.60, 0.40, 0.01],
    [1.50, 1.66, 1.86, 2.10, 0, 0, 0, 0, 0, 0, 0, 86.10, 6.70, 0.07],
    [1.00, 1.11, 1.25, 1.43, 0, 0, 0, 0, 0, 0, 0, 0, 92.88, 2.34],
    [0.50, 0.56, 0.63, 0.72, 0, 0, 0, 0, 0, 0, 0, 0, 0, 97.59]
]

# Create DataFrame for official odds
official_odds = pd.DataFrame(
    official_odds_data,
    index=range(1, 15),
    columns=[f'Pick {i+1}' for i in range(14)]
)
official_odds.index.name = 'Team (Seed)'

# Calculate the difference between simulation and official odds
difference_matrix = simulation_results - official_odds

# Display the difference
difference_matrix
```

Out[3]:

Team (Seed)	Pick 1	Pick 2	Pick 3	Pick 4	Pick 5	Pick 6	Pick 7	Pick 8	Pick 9	Pick 10	Pick 11	Pick 12	Pick 13	Pick 14
1	0.08	-0.08	0.10	0.01	-0.12	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	0.08	-0.02	-0.02	0.11	0.02	-0.18	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	-0.27	-0.01	0.16	-0.05	0.20	-0.02	-0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	0.15	-0.09	-0.09	-0.04	-0.10	0.16	-0.02	0.03	0.00	0.00	0.00	0.00	0.00	0.00
5	0.08	0.00	-0.15	-0.06	-0.01	0.05	0.23	-0.11	-0.04	0.00	0.00	0.00	0.00	0.00
6	-0.02	0.00	-0.15	0.03	0.00	-0.01	-0.08	0.18	0.05	-0.01	0.00	0.00	0.00	0.00
7	0.06	-0.02	0.00	0.04	0.00	0.00	-0.10	-0.06	0.05	0.02	0.00	0.00	0.00	0.00
8	-0.10	0.02	0.09	0.06	0.00	0.00	0.00	-0.05	-0.01	-0.03	0.03	-0.01	0.00	0.00
9	0.04	0.09	0.06	-0.05	0.00	0.00	0.00	0.00	-0.05	-0.03	-0.04	0.00	-0.01	0.00
10	-0.06	0.04	-0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.04	-0.01	0.03	0.00	-0.01
11	0.01	0.03	0.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	-0.14	0.00	-0.01
12	-0.03	-0.01	-0.09	-0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.12	0.05	0.00
13	-0.01	0.04	-0.02	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-0.05	0.01
14	-0.02	0.01	0.05	-0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-0.02

✅ Difference from Official Odds

As we can see, most of the values in the difference matrix are very small and close to zero. This indicates that the simulation aligns well with the official probabilities, as we would expect from a fair and unbiased lottery system.

📐 Analytical Calculation of Draft Probabilities

While we can simulate the NBA Draft Lottery using Monte Carlo methods, it's also possible to compute the exact probabilities analytically by reasoning through conditional probabilities.

🎲 First Pick

Let $P_1(i)$ be the probability that team i receives the 1st pick. This is simply:

$$P_1(i) = \frac{n_i}{1000}$$

where n_i is the number of combinations assigned to team i, and 1000 is the total number of valid combinations.

🧠 Second Pick

Let $P_2(i)$ be the probability that team i receives the 2nd pick. This depends on who got the first pick:

$$P_2(i) = \sum_{\substack{j=1 \\ j \neq i}}^{14} P_2(i \mid j = 1) \cdot P_1(j)$$

where:

- $P_1(j)$ is the probability that team j received the 1st pick.
- $P_2(i \mid j = 1)$ is the conditional probability that team i gets the 2nd pick given that team j got the 1st pick.

This conditional probability is given by:

$$P_2(i \mid j = 1) = \frac{n_i}{1000 - n_j}$$

because once team j has received the first pick, its combinations are removed from the pool, leaving $1000 - n_j$ valid combinations among the remaining 13 teams.

🔢 Third and Fourth Picks

In the same way, the probabilities for the third and fourth picks can be calculated using the following formulas:

📍 Third Pick

$$P_3(i) = \sum_{\substack{j=1 \\ j \neq i}}^{14} \sum_{\substack{k=1 \\ k \neq i,j}}^{14} \left(\frac{n_i}{1000 - n_j - n_k} \cdot \frac{n_k}{1000 - n_j} \cdot \frac{n_j}{1000} \right)$$

📍 Fourth Pick

$$P_4(i) = \sum_{\substack{j=1 \\ j \neq i}}^{14} \sum_{\substack{k=1 \\ k \neq i,j}}^{14} \sum_{\substack{l=1 \\ l \neq i,j,k}}^{14} \left(\frac{n_i}{1000 - n_j - n_k - n_l} \cdot \frac{n_l}{1000 - n_j - n_k} \cdot \frac{n_k}{1000 - n_j} \cdot \frac{n_j}{1000} \right)$$

Each term considers a specific combination of teams receiving the previous picks, and weights the conditional probability of team i receiving the current pick.

For picks 5 through 14, an analytical formula can still be derived. However, due to its complexity—and the fact that these picks are assigned based on the original seeding order after removing the teams that won picks 1 through 4—we omit it here.

In [4]:
```python
import pandas as pd

# Data structured for every year: (Team, Pick, Combinations)
lottery_data = {
    2019: [
        ("New Orleans Pelicans", 1, 60),
        ("Memphis Grizzlies", 2, 60),
        ("New York Knicks", 3, 140),
        ("Los Angeles Lakers", 4, 20),
    ],
    2020: [
        ("Minnesota Timberwolves", 1, 140),
        ("Golden State Warriors", 2, 140),
        ("Charlotte Hornets", 3, 60),
        ("Chicago Bulls", 4, 75),
    ],
    2021: [
        ("Detroit Pistons", 1, 140),
        ("Houston Rockets", 2, 140),
        ("Cleveland Cavaliers", 3, 115),
        ("Toronto Raptors", 4, 75),
    ],
    2022: [
        ("Orlando Magic", 1, 140),
        ("Oklahoma City Thunder", 2, 125),
        ("Houston Rockets", 3, 140),
        ("Sacramento Kings", 4, 75),
    ],
    2023: [
        ("San Antonio Spurs", 1, 140),
        ("Charlotte Hornets", 2, 125),
        ("Portland Trail Blazers", 3, 105),
        ("Houston Rockets", 4, 140),
    ],
    2024: [
        ("Atlanta Hawks", 1, 30),
        ("Washington Wizards", 2, 140),
        ("Houston Rockets", 3, 45),
        ("San Antonio Spurs", 4, 105),
    ],
    2025: [
        ("Dallas Mavericks", 1, 18),
        ("San Antonio Spurs", 2, 140),
        ("Philadelphia 76ers", 3, 105),
        ("Charlotte Hornets", 4, 125),
    ]
}

# Create a DataFrame for each year
yearly_dfs = {year: pd.DataFrame(data, columns=["Team", "Pick", "Combinations"]) for year, dat

# Visualize 2025 dataframe as an example
yearly_dfs[2025]
```

Out[4]:

	Team	Pick	Combinations
0	Dallas Mavericks	1	18
1	San Antonio Spurs	2	140
2	Philadelphia 76ers	3	105
3	Charlotte Hornets	4	125

🎲 Seed-Based Metrics for Evaluating the Lottery

To analyze the overall behavior of the NBA Draft Lottery results in a compact and interpretable way, we define two new random variables based on the **seeds** of the teams that won the top 4 picks in each year.

1. 🧮 Total Seed Sum (`S`)

We define:

$$S = \text{seed}_1 + \text{seed}_2 + \text{seed}_3 + \text{seed}_4$$

That is, `S` is simply the sum of the seeds of the teams that won the 1st, 2nd, 3rd, and 4th picks.

Example (2019):

- 1st: Pelicans (Seed 7)
- 2nd: Grizzlies (Seed 8)
- 3rd: Knicks (Seed 1)
- 4th: Lakers (Seed 11)

Then:

$$S = 7 + 8 + 1 + 11 = 27$$

✅ **Pros:**

- Extremely simple to compute and understand.

- Gives a quick idea of how many "underdogs" or "favorites" won top picks.

⚠️ **Cons:**

- Ignores the order of the picks. For example, `S = 20` can come from both (1,2,3,14) and (14,1,2,3), but the second order is much less probable.

---

2. ⚖️ Weighted Seed Score (`W`)

We define a weighted version that assigns more importance to higher picks:

$$W = 4 \cdot \text{seed}_1 + 3 \cdot \text{seed}_2 + 2 \cdot \text{seed}_3 + 1 \cdot \text{seed}_4$$

This formula gives greater influence to who wins the highest picks.

Example (2019):

$$W = 4 \cdot 7 + 3 \cdot 8 + 2 \cdot 1 + 1 \cdot 11 = 28 + 24 + 2 + 11 = 65$$

✅ **Pros:**

- Takes the pick order into account.
- Helps distinguish "unlikely" distributions from common ones (e.g. penalizes a high-seed team winning the 1st pick more than the 4th).

⚠️ **Cons:**

- Slightly more complex to compute.

📊 Why Use `S` and `W`?

Both `S` and `W` allow us to condense the outcome of a draft lottery into a **single numeric indicator**. The **lower** the values of `S` and `W`, the more the lottery followed the expected path (i.e., lower-seeded teams winning top picks). High values may indicate **unlikely results** and help us quantify how "strange" or "unexpected" a lottery outcome is.

These metrics are particularly useful when:

- Comparing different years
- Benchmarking real data against simulations
- Detecting outliers and improbable configurations

In [5]:
```
# Updated real lottery results (2019-2025)
real_lottery_results = {
    2019: [("New Orleans Pelicans", 1, 7, 60), ("Memphis Grizzlies", 2, 8, 60), ("New York Kni
    2020: [("Minnesota Timberwolves", 1, 3, 140), ("Golden State Warriors", 2, 1, 140), ("Char
    2021: [("Detroit Pistons", 1, 2, 140), ("Houston Rockets", 2, 1, 140), ("Cleveland Cavalie
    2022: [("Orlando Magic", 1, 2, 140), ("Oklahoma City Thunder", 2, 4, 125), ("Houston Rocke
    2023: [("San Antonio Spurs", 1, 3, 140), ("Charlotte Hornets", 2, 4, 125), ("Portland Trai
    2024: [("Atlanta Hawks", 1, 10, 30), ("Washington Wizards", 2, 2, 140), ("Houston Rockets"
    2025: [("Dallas Mavericks", 1, 11, 18), ("San Antonio Spurs", 2, 8, 140), ("Philadelphia 7
}

# Calculate S (sum of seeds) and W (weighted seed score) for each year
seed_scores = []

for year, picks in real_lottery_results.items():
    seeds = [seed for _, _, seed, _ in sorted(picks, key=lambda x: x[1])]
    S = sum(seeds)
    W = 4*seeds[0] + 3*seeds[1] + 2*seeds[2] + 1*seeds[3]
    seed_scores.append((year, S, W))

# Create DataFrame to display results
seed_scores_df = pd.DataFrame(seed_scores, columns=["Year", "Seed Sum (S)", "Weighted Seed Scor

#display
seed_scores_df
```

Out[5]:

	Year	Seed Sum (S)	Weighted Seed Score (W)
0	2019	27	65
1	2020	19	38
2	2021	15	28
3	2022	14	29
4	2023	14	36
5	2024	26	69
6	2025	27	81

📊 Distribution of Seed Sum (S)

To better understand the typical behavior of the lottery under fair conditions, we simulate 100,000 lotteries and compute the Seed Sum `S` for each.

The Seed Sum is defined as:

$$S = \text{seed}_1 + \text{seed}_2 + \text{seed}_3 + \text{seed}_4$$

This gives us a single number summarizing how "high" or "low" the winners' seeds were overall.

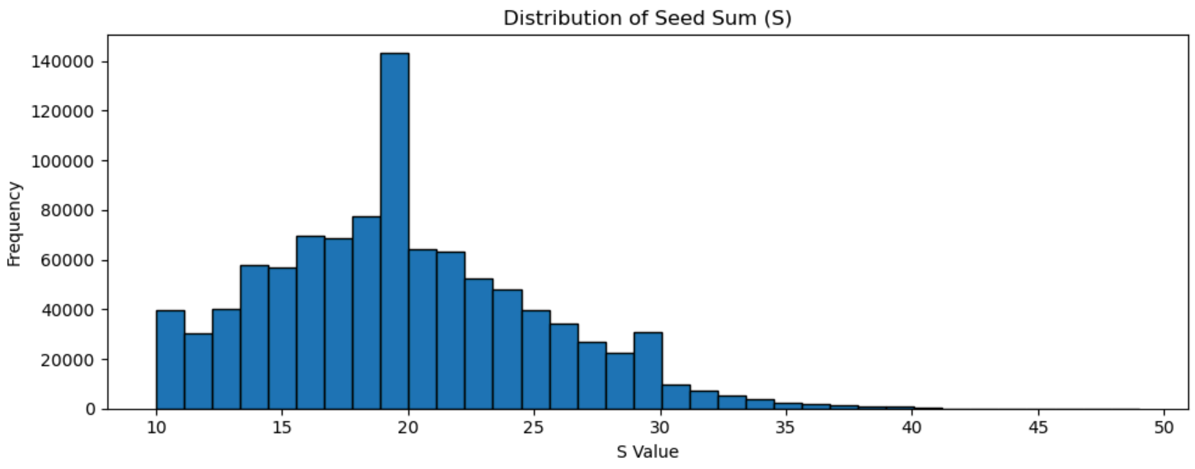
- Lower values of `S` suggest that lower-seeded teams (i.e., worse records) won the lottery.
- Higher values of `S` suggest upsets or unusual outcomes favoring higher-seeded teams.

Below is the distribution of `S` values from 100,000 simulated lotteries:

In [6]:

```
seed=1234
# Simulate distribution of S
def simulate_seed_sum_distribution(team_combinations, num_simulations=1000000):
    random.seed(seed)
    s_values = []
    for _ in range(num_simulations):
        selected_teams = []
        used_combinations = set()
        while len(selected_teams) < 4:
            draw = tuple(sorted(random.sample(range(1, 15), 4)))
            if draw == (11, 12, 13, 14) or draw in used_combinations:
                continue
            used_combinations.add(draw)
            for team, combos in team_combinations.items():
                if draw in combos and team not in selected_teams:
                    selected_teams.append(team)
                    break
        s_values.append(sum(selected_teams))
    return s_values

# Compute and display S statistics
simulated_s_values = simulate_seed_sum_distribution(team_combinations)
s_distribution = pd.Series(simulated_s_values)


# Plot S distribution
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(10, 4))
plt.hist(s_distribution, bins=35, edgecolor='black')
plt.title("Distribution of Seed Sum (S)")
plt.xlabel("S Value")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()
```



In [7]:
```
s_distribution.describe()
```

Out[7]:
```
count    1000000.000000
mean          19.798807
std            5.329962
min           10.000000
25%           16.000000
50%           19.000000
75%           23.000000
max           49.000000
dtype: float64
```

⚖️ Distribution of Weighted Seed Score (W)

To capture not just which teams win the lottery but also the order in which they win, we define a Weighted Seed Score `W`:

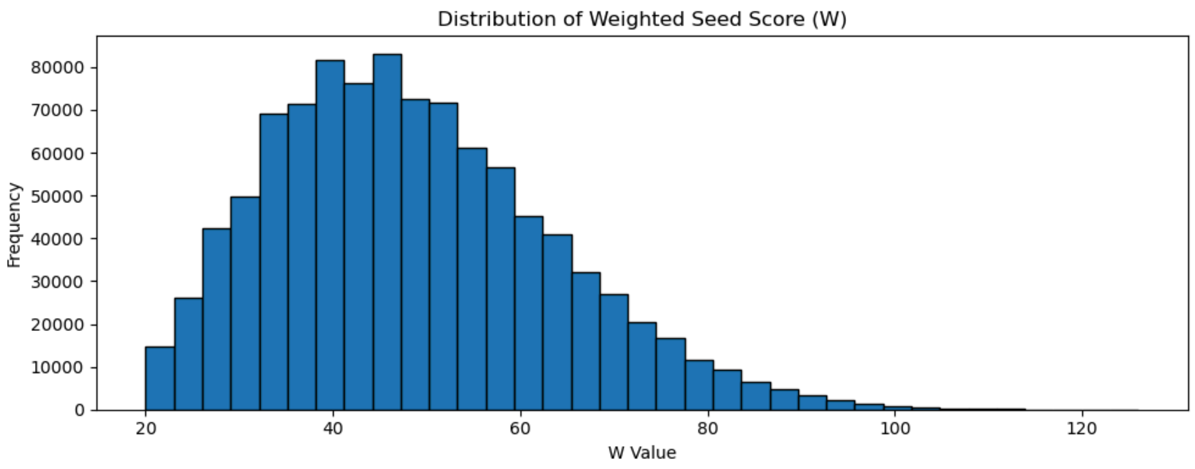
$$W = 4 \cdot \text{seed}_1 + 3 \cdot \text{seed}_2 + 2 \cdot \text{seed}_3 + 1 \cdot \text{seed}_4$$

This gives more importance to teams winning the higher picks. For example, a high seed winning the 1st pick is considered more improbable than winning the 4th.

- Lower `W` values indicate lotteries that favor lower seeds.
- Higher `W` values suggest surprising or unusual outcomes.

Below is the distribution of `W` values from 100,000 simulated lotteries:

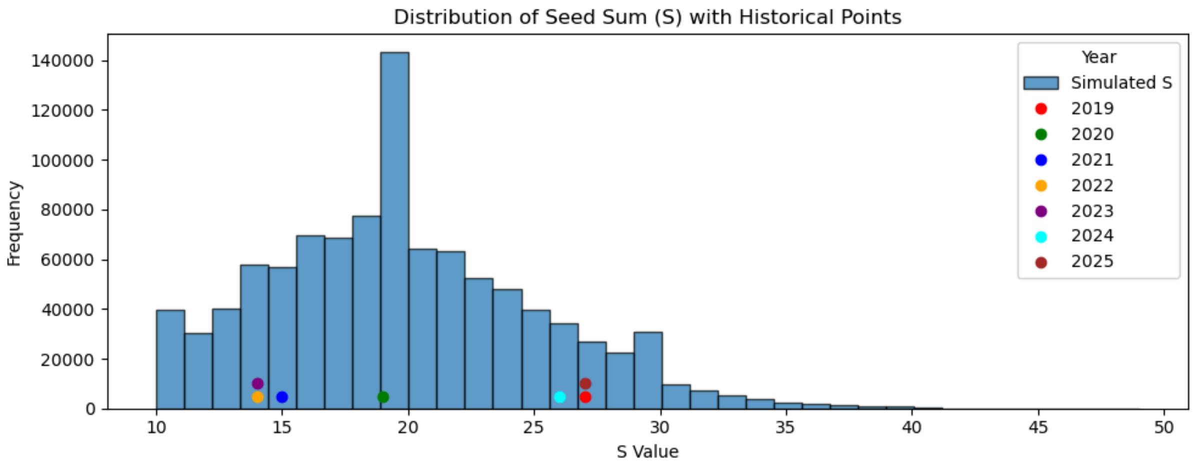
In [8]:
```
seed=1234
# Simulate distribution of W
def simulate_weighted_seed_distribution(team_combinations, num_simulations=1000000):
    random.seed(seed)
    w_values = []
    for _ in range(num_simulations):
        selected_teams = []
        used_combinations = set()
        while len(selected_teams) < 4:
            draw = tuple(sorted(random.sample(range(1, 15), 4)))
            if draw == (11, 12, 13, 14) or draw in used_combinations:
                continue
            used_combinations.add(draw)
            for team, combos in team_combinations.items():
                if draw in combos and team not in selected_teams:
                    selected_teams.append(team)
                    break
        w_score = 4 * selected_teams[0] + 3 * selected_teams[1] + 2 * selected_teams[2] + 1 * s
        w_values.append(w_score)
    return w_values

# Compute and display W statistics
simulated_w_values = simulate_weighted_seed_distribution(team_combinations)
w_distribution = pd.Series(simulated_w_values)

# Plot W distribution
plt.figure(figsize=(10, 4))
plt.hist(w_distribution, bins=35, edgecolor='black')
plt.title("Distribution of Weighted Seed Score (W)")
plt.xlabel("W Value")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()
```

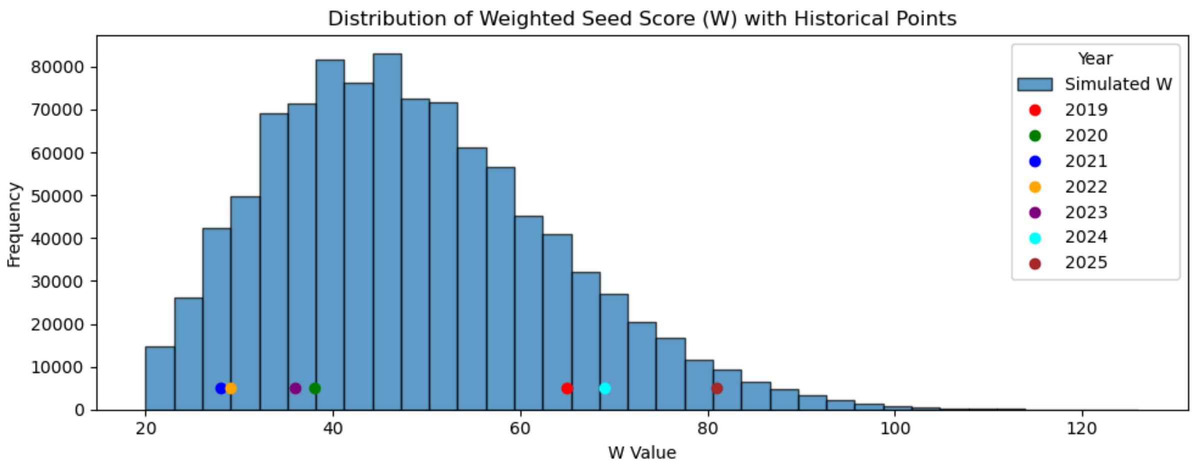


In [9]:
```
w_distribution.describe()
```

Out[9]:
```
count    1000000.000000
mean          48.665614
std           14.880286
min           20.000000
25%           37.000000
50%           47.000000
75%           58.000000
max          126.000000
dtype: float64
```

In [11]:
```
# Group points by value to offset overlapping dots
from collections import defaultdict


colors = ['red', 'green', 'blue', 'orange', 'purple', 'cyan', 'brown']
years = seed_scores_df['Year'].tolist()
s_values = seed_scores_df['Seed Sum (S)'].tolist()
w_values = seed_scores_df['Weighted Seed Score (W)'].tolist()


# Offset management for duplicate values
s_positions = defaultdict(int)
w_positions = defaultdict(int)

# Plot S distribution with adjusted dots
fig_s, ax_s = plt.subplots(figsize=(10, 4))
ax_s.hist(s_distribution, bins=35, edgecolor='black', alpha=0.7, label='Simulated S')
for i, (val, color) in enumerate(zip(s_values, colors)):
    offset = s_positions[val] * 5000 + 5000
    ax_s.plot(val, offset, 'o', color=color, label=str(years[i]))
    s_positions[val] += 1
ax_s.set_title("Distribution of Seed Sum (S) with Historical Points")
ax_s.set_xlabel("S Value")
ax_s.set_ylabel("Frequency")
ax_s.legend(title="Year", loc="upper right")
plt.tight_layout()

# Plot W distribution with adjusted dots
fig_w, ax_w = plt.subplots(figsize=(10, 4))
ax_w.hist(w_distribution, bins=35, edgecolor='black', alpha=0.7, label='Simulated W')
for i, (val, color) in enumerate(zip(w_values, colors)):
    offset = w_positions[val] * 5000 + 5000
    ax_w.plot(val, offset, 'o', color=color, label=str(years[i]))
    w_positions[val] += 1
ax_w.set_title("Distribution of Weighted Seed Score (W) with Historical Points")
ax_w.set_xlabel("W Value")
ax_w.set_ylabel("Frequency")
ax_w.legend(title="Year", loc="upper right")
plt.tight_layout()

plt.show()
```





In [12]:
```
from scipy.stats import gaussian_kde
import numpy as np

colors = ['red', 'green', 'blue', 'orange', 'purple', 'cyan', 'brown']
years = seed_scores_df['Year'].tolist()
s_values = seed_scores_df['Seed Sum (S)'].tolist()
w_values = seed_scores_df['Weighted Seed Score (W)'].tolist()


# Fit KDE on simulated S values
kde_s = gaussian_kde(s_distribution)

# Evaluate the KDE at the 7 historical S values
historical_s_values = seed_scores_df["Seed Sum (S)"].values
```

```
prob_densities = kde_s(historical_s_values)

# Compute log-likelihood
log_likelihood_s = np.sum(np.log(prob_densities))

prob_densities, log_likelihood_s


# Generate log-likelihoods for 100 random 7-sample draws from the simulated S distribution
num_trials = 100
simulated_log_likelihoods = []

for _ in range(num_trials):
    random_sample = np.random.choice(s_distribution, size=7, replace=False)
    densities = kde_s(random_sample)
    log_likelihood = np.sum(np.log(densities))
    simulated_log_likelihoods.append(log_likelihood)

# Convert to Series and calculate summary
sim_ll_series = pd.Series(simulated_log_likelihoods)
real_ll = log_likelihood_s
percentile_of_real = (sim_ll_series < real_ll).mean() * 100

sim_ll_series.describe(), real_ll, percentile_of_real
```

Out[12]:

```
(count    100.000000
 mean     -19.949359
 std        1.717550
 min      -26.231411
 25%      -20.681009
 50%      -19.660232
 75%      -18.817616
 max      -17.178859
 dtype: float64,
 -20.44735670425693,
 30.0)
```

In [13]:

```
# Fit KDE on simulated W values
kde_w = gaussian_kde(w_distribution)

# Evaluate the KDE at the 7 historical W values
historical_w_values = seed_scores_df["Weighted Seed Score (W)"].values
prob_densities_w = kde_w(historical_w_values)

# Compute log-likelihood for W
log_likelihood_w = np.sum(np.log(prob_densities_w))

# Generate log-likelihoods for 100 random 7-sample draws from the simulated W distribution
simulated_log_likelihoods_w = []
for _ in range(100):
    random_sample_w = np.random.choice(w_distribution, size=7, replace=False)
    densities_w = kde_w(random_sample_w)
    log_likelihood_w_sim = np.sum(np.log(densities_w))
    simulated_log_likelihoods_w.append(log_likelihood_w_sim)

# Convert to Series and calculate summary
sim_ll_w_series = pd.Series(simulated_log_likelihoods_w)
real_ll_w = log_likelihood_w
percentile_of_real_w = (sim_ll_w_series < real_ll_w).mean() * 100

sim_ll_w_series.describe(), real_ll_w, percentile_of_real_w
```

```
Out[13]: (count    100.000000
          mean     -28.648898
          std        1.751731
          min      -37.820464
          25%      -29.574844
          50%      -28.239729
          75%      -27.438257
          max      -25.755160
          dtype: float64,
          -30.662130110296765,
          11.0)
```

🧪 Statistical Likelihood of the Lottery Outcomes (2019–2025)

To assess whether the observed NBA Draft Lottery results from 2019 to 2025 are statistically plausible under the official lottery model, we compute the **log-likelihood** of the real data using a kernel density estimate (KDE) of simulated outcomes.

We focus on two synthetic metrics:

- `S` : the **sum of the seeds** of the four teams that win the top picks each year.
- `W` : a **weighted seed score**, which penalizes high-seeded teams winning the top picks more strongly.

We compare the real log-likelihoods to 1,000 simulated 7-year lottery sequences to compute a percentile — i.e., the proportion of simulated sequences that are *less likely* than the real one.

📊 Results for `S` (Seed Sum)

- **Log-likelihood of real data**: -20.45
- **Simulated average**: -20.03
- **Percentile**: **40%**

✅ **Interpretation**: The observed values for `S` fall squarely in the middle of what we'd expect under a fair lottery. About 40% of random simulations are less likely than the real one — meaning the real outcomes are well within the normal range of the model. No sign of statistical anomaly here.

⚖️ Results for `W` (Weighted Seed Score)

- **Log-likelihood of real data**: -30.66
- **Simulated average**: -28.62
- **Percentile**: **13%**

⚠️ **Interpretation**: The weighted score `W`, which emphasizes who wins the higher picks, is more unusual: only 13% of simulated lotteries are less likely than the observed one. While still plausible under the model, this indicates that the real data is near the edge of what we'd expect from randomness.

📌 Conclusion

From a statistical standpoint:

- The **Seed Sum** `S` confirms that the lottery results are consistent with a fair process.
- The **Weighted Score** `W` suggests the order of winners is slightly less typical, but not enough to be suspicious.

In short, while some results may appear surprising at first glance, **there is no compelling statistical evidence** that the NBA Draft Lottery from 2019 to 2025 deviated from expected probabilities.

⚠️ Limitations of the Analysis

It's important to acknowledge the main limitation: **we are working with only 7 lottery outcomes (2019–2025)**.

From a statistical standpoint:

- **Seven data points are not enough** to detect subtle patterns or systematic bias with high confidence.
- Even under a fair system, it's **normal to see unusual outcomes** occasionally.
- Real-world variability and small sample size reduce the power of our analysis.

Therefore, these results **do not prove fairness**, but they also **do not provide statistical evidence of manipulation**. The observed outcomes are within the range of what the current lottery model predicts.

Longer-term data, or a deeper dive into conditional probabilities and structural changes, would be needed for stronger conclusions.