

# Descriptive Statistics

## What Problem This Solves

Descriptive statistics summarize data so you can understand it at a glance.

When you have thousands of data points:

- What's typical?
- How spread out is the data?
- Are there outliers?
- What patterns exist?

Turns raw numbers into insights.

---

## Intuition & Mental Model

### Think: Describing a Crowd

Instead of listing everyone's height:

- Average height: ~5'8"
- Most people: Between 5'4" and 6'0"
- Few outliers: Under 5' or over 6'4"

Statistics = describing large groups concisely.

---

## Core Concepts

### 1. Measures of Center

#### Mean (Average)

Sum of all values / Number of values

```
function mean(arr) {
  return arr.reduce((sum, x) => sum + x, 0) / arr.length;
}

const responseTimes = [45, 52, 48, 51, 49, 53, 2000]; // ms
mean(responseTimes); // 328ms

// Problem: Outlier (2000ms) skews the mean!
```

When to use: Data without extreme outliers

---

#### Median (Middle Value)

Sort values, take the middle one  
(or average of two middle if even count)

```

function median(arr) {
  const sorted = [...arr].sort((a, b) => a - b);
  const mid = Math.floor(sorted.length / 2);

  if (sorted.length % 2 === 0) {
    return (sorted[mid - 1] + sorted[mid]) / 2;
  }
  return sorted[mid];
}

median(responseTimes); // 51ms
// Much more representative! Outlier doesn't affect it

```

**When to use:** Data with outliers (salaries, latencies, prices)

---

### Mode (Most Common)

```

function mode(arr) {
  const counts = {};
  let maxCount = 0;
  let modeValue = arr[0];

  for (const value of arr) {
    counts[value] = (counts[value] || 0) + 1;
    if (counts[value] > maxCount) {
      maxCount = counts[value];
      modeValue = value;
    }
  }

  return modeValue;
}

const statusCodes = [200, 200, 200, 404, 200, 500, 200];
mode(statusCodes); // 200

```

**When to use:** Categorical data (status codes, user types)

---

### Comparison:

```

const data = [1, 2, 2, 3, 4, 5, 6, 7, 8, 100];

mean(data); // 13.8 ~ Pulled up by 100
median(data); // 4.5 ~ Not affected
mode(data); // 2 ~ Most common

// Median often best for real-world data

```

---

## 2. Measures of Spread

### Range

```
function range(arr) {
  return Math.max(...arr) - Math.min(...arr);
}

const temps = [65, 68, 70, 72, 75];
range(temps); // 10 degrees
```

**Problem:** Sensitive to outliers

### Variance (average squared deviation from mean)

```
function variance(arr) {
  const avg = mean(arr);
  const squaredDiffs = arr.map(x => (x - avg) ** 2);
  return mean(squaredDiffs);
}

// Why square? Negative deviations don't cancel positive
```

### Standard Deviation (typical distance from mean)

```
function standardDeviation(arr) {
  return Math.sqrt(variance(arr));
}

const scores = [80, 82, 85, 88, 90];
mean(scores); // 85
standardDeviation(scores); // ~3.74

// "Typical score is within 3.74 points of 85"
```

### Interpretation:

Low std dev → Data clustered near mean  
High std dev → Data spread out

### Example:

```
const consistent = [100, 101, 99, 100, 100];
standardDeviation(consistent); // ~0.7 (very consistent)

const variable = [50, 75, 100, 125, 150];
standardDeviation(variable); // ~35.4 (highly variable)
```

### Percentiles

```

function percentile(arr, p) {
  const sorted = [...arr].sort((a, b) => a - b);
  const index = (p / 100) * (sorted.length - 1);
  const lower = Math.floor(index);
  const upper = Math.ceil(index);
  const weight = index - lower;

  return sorted[lower] * (1 - weight) + sorted[upper] * weight;
}

const latencies = [10, 15, 20, 25, 30, 40, 50, 100, 200, 500];

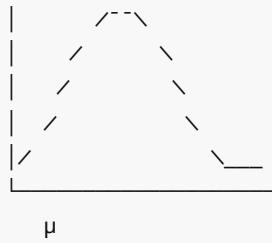
percentile(latencies, 50); // 35 (median, P50)
percentile(latencies, 95); // 350 (P95)
percentile(latencies, 99); // 480 (P99)

// "95% of requests faster than 350ms"

```

### 3. Data Distributions

**Normal Distribution** (bell curve)



**Properties:**

- Symmetric around mean ( $\mu$ )
- 68% within 1 std dev
- 95% within 2 std devs
- 99.7% within 3 std devs

**When it appears:**

- Heights, test scores, measurement errors
- Aggregates of many random factors

```

// Check if roughly normal
function isRoughlyNormal(arr) {
  const avg = mean(arr);
  const std = standardDeviation(arr);

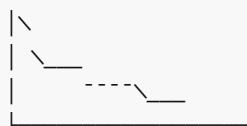
  const within1Std = arr.filter(x =>
    Math.abs(x - avg) <= std
  ).length / arr.length;

```

```
    return within1Std > 0.6 && within1Std < 0.75;
}
```

## Skewed Distributions

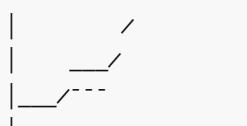
**Right-skewed** (long tail to right):



Mean > Median

Example: Income, response times

**Left-skewed** (long tail to left):



Mean < Median

Example: Test scores (easy test)

```
function skewness(arr) {
  const avg = mean(arr);
  const std = standardDeviation(arr);
  const med = median(arr);

  // Simplified skew indicator
  return (avg - med) / std;
}

// > 0: Right-skewed
// < 0: Left-skewed
// ≈ 0: Symmetric
```

## 4. Outliers

**What is an outlier?**

Value unusually far from others.

**IQR Method** (Interquartile Range):

```
function findOutliers(arr) {
  const sorted = [...arr].sort((a, b) => a - b);
  const q1 = percentile(sorted, 25);
  const q3 = percentile(sorted, 75);
  const iqr = q3 - q1;
```

```

const lowerBound = q1 - 1.5 * iqr;
const upperBound = q3 + 1.5 * iqr;

return sorted.filter(x => x < lowerBound || x > upperBound);
}

const data = [10, 12, 14, 15, 16, 18, 20, 22, 150];
findOutliers(data); // [150]

```

### Why outliers matter:

- Can skew mean dramatically
- May indicate bugs or anomalies
- Sometimes the most interesting data points

```

// Response times with one timeout
const times = [45, 48, 50, 52, 55, 5000];

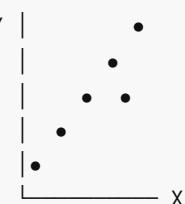
mean(times);    // 875ms   ← Misleading!
median(times); // 51ms    ← Representative

```

## 5. Correlation (Relationship Between Variables)

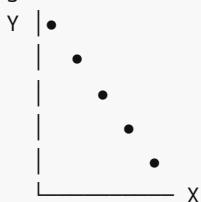
### Do two variables move together?

Positive correlation:



As X increases, Y increases

Negative correlation:



As X increases, Y decreases

No correlation:





X and Y unrelated

### Correlation coefficient (r):

```
r = 1 → Perfect positive correlation  
r = 0 → No correlation  
r = -1 → Perfect negative correlation
```

```
function correlation(x, y) {  
    const n = x.length;  
    const meanX = mean(x);  
    const meanY = mean(y);  
  
    const numerator = x.reduce((sum, xi, i) =>  
        sum + (xi - meanX) * (y[i] - meanY), 0  
    );  
  
    const denomX = Math.sqrt(x.reduce((sum, xi) =>  
        sum + (xi - meanX) ** 2, 0  
    ));  
  
    const denomY = Math.sqrt(y.reduce((sum, yi) =>  
        sum + (yi - meanY) ** 2, 0  
    ));  
  
    return numerator / (denomX * denomY);  
}  
  
// Server load vs response time  
const load = [10, 20, 30, 40, 50];  
const responseTime = [50, 75, 100, 125, 150];  
correlation(load, responseTime); // ~1.0 (strong positive)
```

⚠ Correlation ≠ Causation (more in inferential statistics)

## Software Engineering Connections

### 1. Performance Monitoring

```
class PerformanceMonitor {  
    constructor() {  
        this.latencies = [];  
    }  
  
    record(latency) {  
        this.latencies.push(latency);  
    }  
}
```

```

getStats() {
  return {
    mean: mean(this.latencies),
    median: median(this.latencies),
    p95: percentile(this.latencies, 95),
    p99: percentile(this.latencies, 99),
    stdDev: standardDeviation(this.latencies),
    outliers: findOutliers(this.latencies)
  };
}

// Report: "P95 latency: 120ms, P99: 250ms"
// Better than mean (hides outliers)

```

## 2. A/B Test Results

```

const variantA = [0.05, 0.06, 0.05, 0.07, 0.05]; // Conversion rates
const variantB = [0.08, 0.07, 0.08, 0.09, 0.07];

console.log({
  A: {
    mean: mean(variantA),      // 0.056
    stdDev: standardDeviation(variantA) // 0.008
  },
  B: {
    mean: mean(variantB),      // 0.078
    stdDev: standardDeviation(variantB) // 0.008
  }
});

// B appears better, but need statistical test
// (covered in inferential statistics)

```

## 3. Database Query Analytics

```

const queryTimes = [
  12, 15, 18, 14, 16, 13, 17, 500, 12, 14
];

const stats = {
  mean: mean(queryTimes),      // 63.1ms ← Skewed by 500ms
  median: median(queryTimes),   // 14.5ms ← More representative
  p95: percentile(queryTimes, 95), // 500ms ← Slowest 5%
  outliers: findOutliers(queryTimes) // [500ms]
};

// Alert if P95 > threshold

```

```
if (stats.p95 > 100) {
  alert("Slow queries detected");
}
```

## 4. User Behavior Analysis

```
const sessionDurations = /* thousands of values */;

const summary = {
  median: median(sessionDurations), // Typical user
  mean: mean(sessionDurations), // Average (skewed by power users)
  mode: mode(sessionDurations), // Most common behavior

  // Segments
  shortSessions: sessionDurations.filter(d => d < percentile(sessionDurations, 25)),
  longSessions: sessionDurations.filter(d => d > percentile(sessionDurations, 75))
};
```

## 5. Error Rate Tracking

```
const hourlyErrors = [2, 1, 3, 2, 1, 45, 2, 3];

const baseline = median(hourlyErrors); // 2 errors/hour
const threshold = baseline + 3 * standardDeviation(hourlyErrors);

if (hourlyErrors[hourlyErrors.length - 1] > threshold) {
  alert("Error rate spike!");
}
```

## Common Misconceptions

### ✗ "Average always represents typical value"

Mean can be misleading with skewed data:

```
const salaries = [40000, 42000, 45000, 48000, 500000];

mean(salaries); // $135,000 ← Misleading
median(salaries); // $45,000 ← Typical
```

### ✗ "Standard deviation tells you the range"

Standard deviation is about typical distance, not total range:

```
const data = [1, 2, 3, 4, 5, 100];
```

```
range(data);           // 99  
standardDeviation(data); // ~38
```

## ✗ "Correlation means one causes the other"

**Correlation ≠ Causation:**

```
Ice cream sales ↔ Drownings (correlated)  
But ice cream doesn't cause drownings!  
Both caused by summer weather.
```

## ✗ "Remove all outliers"

**Outliers can be important:**

- May indicate bugs (good to know!)
- May be legitimate rare events
- May be your most valuable customers

Always investigate before removing.

## ✗ "Normal distribution is common"

**Many real-world distributions are NOT normal:**

- Income: Right-skewed
- Response times: Right-skewed
- User engagement: Power law

Don't assume normality without checking.

---

## Practical Mini-Exercises

### Exercise 1: Interpret Metrics

You're monitoring API response times:

```
const times = [45, 48, 50, 52, 55, 58, 60, 65, 70, 500];
```

Calculate mean, median, P95, P99. Which metric best represents user experience?

► Solution

### Exercise 2: Detect Anomaly

Track hourly request counts:

```
const hourlyRequests = [  
 1200, 1150, 1180, 1220, 1190,  
 1210, 1180, 1200, 1190, 5000  
];
```

Is the last hour an anomaly?

► Solution

### Exercise 3: Compare Distributions

Two servers handle requests:

```
const serverA = [100, 110, 105, 108, 102, 107];
const serverB = [50, 150, 60, 140, 70, 130];
```

Which is more consistent?

► Solution

---

## Summary Cheat Sheet

### Measures of Center

```
mean(arr)      // Average (skewed by outliers)
median(arr)    // Middle value (robust to outliers)
mode(arr)      // Most common (for categorical data)
```

### Measures of Spread

```
range(arr)          // Max - min
variance(arr)       // Average squared deviation
standardDeviation(arr) // Typical distance from mean
percentile(arr, p)  // Value at p-th percentile
```

### When to Use Each

Metric	Best For
Mean	Normal distributions, no outliers
Median	Skewed data, outliers present
Mode	Categorical data
Std Dev	Understanding spread
Percentiles	Tail behavior (P95, P99)

### Quick Checks

```
// Outlier detection
const outliers = findOutliers(data); // IQR method

// Distribution shape
if (mean > median) {
  console.log("Right-skewed");
```

```
} else if (mean < median) {
  console.log("Left-skewed");
} else {
  console.log("Roughly symmetric");
}

// Consistency
const cv = standardDeviation(data) / mean(data); // Coefficient of variation
// Low CV = consistent, High CV = variable
```

---

## Next Steps

Descriptive statistics help you understand and summarize data. You now know how to calculate and interpret metrics that describe distributions.

Next, we'll explore **inferential statistics**—using sample data to make conclusions about larger populations and determining if differences are real or just chance.

Continue to: [08-inferential-statistics.md](#)