12. Expected Value & Decision Making Under Uncertainty

Phase 4: Math for Finance & Decision Making
⏱ ~45 minutes | 🎲 Decision Framework | 💡 Think in Probabilities

---

What Problem This Solves

You're facing decisions like:

- Should we build feature A (70% chance of +$50k revenue) or feature B (30% chance of +$200k)?
- Is it worth investing $10k in testing infrastructure to prevent rare but expensive bugs?
- Should we offer a money-back guarantee?
- Which API provider should we choose given different pricing + uptime SLAs?
- Should we hire 2 mid-level engineers or 1 senior?

Without expected value thinking, you make decisions on gut feeling, overweight rare outcomes ("but what if..."), or chase high-payoff low-probability bets while ignoring consistent value.

With expected value, you quantify uncertain outcomes, compare options objectively, and make long-term optimal decisions even when individual results vary.

Intuition & Mental Model

The Core Insight: Weighted Average of Possible Futures

```
Expected Value = Σ (Outcome × Probability)

Not "what will happen" but "average over many trials"
```

Mental Model: The Casino Perspective

A casino doesn't know if the next spin wins or loses. But over 10,000 spins:

```
Roulette (single number):
- Win: $350 (1/38 chance)
- Lose: -$10 (37/38 chance)

Expected Value = (1/38 × $350) + (37/38 × -$10)
               = $9.21 - $9.74
               = -$0.53 per bet

Casino wins in the long run, even though most bets win FOR the player in the short
term.
```

Software Engineering Analogy:

```
You're not making one feature decision.
You're making HUNDREDS over your career.
```

```
Expected value = long-term average success strategy.
```

Core Concepts

1. Basic Expected Value

```javascript
function expectedValue(outcomes) {
  // outcomes = [{ value: number, probability: number }, ...]
  return outcomes.reduce((sum, outcome) =>
    sum + (outcome.value * outcome.probability), 0
  );
}

// Coin flip: Win $10 (50%) or lose $5 (50%)
expectedValue([
  { value: 10, probability: 0.5 },
  { value: -5, probability: 0.5 }
]);
// $2.50 expected value
// Interpretation: Over many flips, you average $2.50 profit per flip
```

Real Example: Feature Prioritization

```javascript
function compareFeatures(features) {
  return features.map(f => ({
    name: f.name,
    expectedRevenue: expectedValue(f.outcomes),
    costToBuild: f.cost,
    netExpectedValue: expectedValue(f.outcomes) - f.cost,
    roi: ((expectedValue(f.outcomes) - f.cost) / f.cost * 100).toFixed(1) + '%'
  })).sort((a, b) => b.netExpectedValue - a.netExpectedValue);
}

compareFeatures([
  {
    name: 'Social Login',
    cost: 5000,
    outcomes: [
      { value: 20000, probability: 0.6 },  // 60% chance: +$20k revenue
      { value: 5000, probability: 0.3 },   // 30% chance: +$5k revenue
      { value: 0, probability: 0.1 }        // 10% chance: No uptake
    ]
  },
  {
    name: 'AI Recommendations',
    cost: 30000,
    outcomes: [
      { value: 150000, probability: 0.2 },  // 20% chance: Home run
      { value: 40000, probability: 0.5 },   // 50% chance: Moderate success
```

```
      { value: 10000, probability: 0.3 }    // 30% chance: Flop
    ]
  },
  {
    name: 'Dark Mode',
    cost: 3000,
    outcomes: [
      { value: 8000, probability: 0.8 },    // 80% chance: Small win
      { value: 2000, probability: 0.2 }     // 20% chance: Tiny win
    ]
  }
]);
/* [
  { name: 'AI Recommendations', expectedRevenue: 53000, cost: 30000,
netExpectedValue: 23000, roi: '76.7%' },
  { name: 'Social Login', expectedRevenue: 13500, cost: 5000, netExpectedValue:
8500, roi: '170.0%' },
  { name: 'Dark Mode', expectedRevenue: 6800, cost: 3000, netExpectedValue: 3800,
roi: '126.7%' }
]

Social Login has HIGHEST ROI (170%), but AI has highest absolute value.
Decision depends on: capital constraints, risk tolerance, strategic goals */
```

2. Decision Trees

Multi-Stage Decisions: Outcomes depend on sequences of events

```
function buildDecisionTree(options) {
  return options.map(option => {
    // Calculate expected value through multiple stages
    let ev = 0;

    option.branches.forEach(branch => {
      if (branch.subBranches) {
        // Nested probability
        branch.subBranches.forEach(subBranch => {
          ev += option.initialCost * (-1) +
                (subBranch.value * branch.probability * subBranch.probability);
        });
      } else {
        ev += (branch.value * branch.probability) - option.initialCost;
      }
    });

    return { name: option.name, expectedValue: ev.toFixed(0) };
  });
}

// Example: Launch new product
const productLaunch = {
```

```
  name: 'New API Product',
  initialCost: 50000,
  branches: [
    {
      outcome: 'Strong launch',
      probability: 0.4,
      subBranches: [
        { outcome: 'Goes viral', probability: 0.3, value: 500000 },
        { outcome: 'Steady growth', probability: 0.7, value: 150000 }
      ]
    },
    {
      outcome: 'Weak launch',
      probability: 0.6,
      subBranches: [
        { outcome: 'Recovers', probability: 0.4, value: 80000 },
        { outcome: 'Fails', probability: 0.6, value: 10000 }
      ]
    }
  ]
};

// Calculate EV manually:
const strongViralEV = 0.4 * 0.3 * 500000;       // 60,000
const strongSteadyEV = 0.4 * 0.7 * 150000;      // 42,000
const weakRecoverEV = 0.6 * 0.4 * 80000;        // 19,200
const weakFailEV = 0.6 * 0.6 * 10000;           // 3,600
const totalEV = strongViralEV + strongSteadyEV + weakRecoverEV + weakFailEV - 50000;

console.log(`Expected Value: $${totalEV}`);  // $74,800
// Despite 60% chance of "weak launch", still profitable in expectation!
```

3. Insurance & Risk Premium

Problem: How much should you pay to avoid risk?

```
function insuranceValue(loss, probability, riskAversion = 1) {
  const expectedLoss = loss * probability;
  const riskPremium = riskAversion * Math.sqrt(loss * probability * (1 -
probability));

  return {
    expectedLoss,
    maxWillingToPay: expectedLoss + riskPremium,
    breakEvenPrice: expectedLoss
  };
}

// Server crashes: $100k loss, 2% chance per year
insuranceValue(100000, 0.02, 1.5);
/* {
```

```
  expectedLoss: 2000,
  maxWillingToPay: 4099,  // Would pay up to $4k for insurance
  breakEvenPrice: 2000     // Insurance fair price = $2k
}

Interpretation:
- Expected loss: $2k/year (average)
- You'd pay $4k to avoid the risk (risk aversion)
- Insurance company charges $2.5k → Good deal for you */
```

Real Example: SLA Credits

```
function evaluateSLA(revenue, slaOptions) {
  return slaOptions.map(sla => {
    const expectedDowntime = sla.uptimePercent ? (1 - sla.uptimePercent) : 0.01;
    const expectedLoss = revenue * expectedDowntime * sla.lossMultiplier;

    return {
      name: sla.name,
      monthlyCost: sla.cost,
      expectedLoss: Math.round(expectedLoss),
      netCost: sla.cost + expectedLoss,
      decision: sla.cost + expectedLoss
    };
  }).sort((a, b) => a.netCost - b.netCost);
}

// E-commerce site: $500k/month revenue, downtime = lost sales
evaluateSLA(500000, [
  { name: 'Basic (99%)', cost: 500, uptimePercent: 0.99, lossMultiplier: 1 },
  { name: 'Pro (99.9%)', cost: 2000, uptimePercent: 0.999, lossMultiplier: 1 },
  { name: 'Enterprise (99.99%)', cost: 8000, uptimePercent: 0.9999, lossMultiplier:
1 }
]);
/* [
  { name: 'Basic', monthlyCost: 500, expectedLoss: 5000, netCost: 5500 },
  { name: 'Pro', monthlyCost: 2000, expectedLoss: 500, netCost: 2500 },   ← Best
  { name: 'Enterprise', monthlyCost: 8000, expectedLoss: 50, netCost: 8050 }
]
// Pro tier has lowest total cost (hosting + expected downtime loss) */
```

4. Kelly Criterion (Optimal Bet Sizing)

Question: How much should you "bet" on a positive EV opportunity?

```
function kellyCriterion(winProbability, winMultiplier, lossProbability, lossAmount =
1) {
  // Kelly Formula: f = (p × b - q) / b
  // f = fraction of bankroll to bet
  // p = win probability
  // b = win multiplier (how much you win per $1 bet)
```

```
  // q = lose probability

  const q = lossProbability;
  const b = winMultiplier / lossAmount;
  const f = (winProbability * b - q) / b;

  return Math.max(0, f);  // Never bet negative (if EV is negative)
}

// Startup idea: 30% chance of 10x return, 70% chance of total loss
kellyCriterion(0.30, 10, 0.70);  // 0.23 = 23% of capital
// Bet 23% of your available capital (not 100%!)

// Feature with 60% success (2x return), 40% failure
kellyCriterion(0.60, 2, 0.40);  // 0.40 = 40% of budget
```

Engineering Budget Example:

```
function allocateEngineeringBudget(budget, projects) {
  return projects.map(p => {
    const kellyFraction = kellyCriterion(p.successProb, p.returnMultiplier, 1 -
p.successProb);
    const allocation = Math.min(budget * kellyFraction, p.maxCost);
    const expectedReturn = allocation * (p.successProb * p.returnMultiplier - (1 -
p.successProb));

    return {
      name: p.name,
      kellyFraction: (kellyFraction * 100).toFixed(1) + '%',
      recommendedBudget: Math.round(allocation),
      expectedReturn: Math.round(expectedReturn)
    };
  });
}

allocateEngineeringBudget(100000, [
  { name: 'Safe Infrastructure', successProb: 0.9, returnMultiplier: 1.5, maxCost:
30000 },
  { name: 'Risky AI Feature', successProb: 0.3, returnMultiplier: 10, maxCost: 50000
},
  { name: 'Moderate Redesign', successProb: 0.6, returnMultiplier: 3, maxCost: 40000
}
]);
/* [
  { name: 'Safe Infrastructure', kellyFraction: '50.0%', recommendedBudget: 30000,
expectedReturn: 10500 },
  { name: 'Risky AI', kellyFraction: '23.0%', recommendedBudget: 23000,
expectedReturn: 46000 },
  { name: 'Moderate Redesign', kellyFraction: '40.0%', recommendedBudget: 40000,
expectedReturn: 32000 }
```

```
]
// Allocate more to safer bets, but don't ignore high-upside risks */
```

5. Gambler's Fallacy vs Law of Large Numbers

Gambler's Fallacy: "I've flipped heads 5 times, tails is DUE!"

```
function simulateFlips(n) {
  const flips = Array.from({ length: n }, () => Math.random() < 0.5 ? 'H' : 'T');
  const heads = flips.filter(f => f === 'H').length;
  return {
    flips: flips.slice(-10).join(' '),  // Last 10
    totalHeads: heads,
    totalTails: n - heads,
    percentHeads: (heads / n * 100).toFixed(1) + '%'
  };
}

// After 5 heads in a row, next flip is STILL 50/50
// Past outcomes don't affect future probability (independent events)
```

Law of Large Numbers: Over many trials, average converges to expected value

```
function demonstrateLawOfLargeNumbers(trials) {
  // Bet $1: Win $10 (5% chance), Lose $1 (95% chance)
  const expectedValue = 0.05 * 10 + 0.95 * (-1);  // -$0.45 per bet

  let total = 0;
  const results = [];

  for (let i = 1; i <= trials; i++) {
    const outcome = Math.random() < 0.05 ? 10 : -1;
    total += outcome;

    if (i % 100 === 0) {
      results.push({
        trial: i,
        averagePerBet: (total / i).toFixed(3),
        expectedValue: expectedValue.toFixed(3)
      });
    }
  }

  return results;
}

demonstrateLawOfLargeNumbers(1000);
/* [
  { trial: 100, averagePerBet: '0.200', expectedValue: '-0.450' },   // Lucky streak
  { trial: 200, averagePerBet: '-0.350', expectedValue: '-0.450' },  // Unlucky
  { trial: 500, averagePerBet: '-0.422', expectedValue: '-0.450' },  // Converging
```

```
  { trial: 1000, averagePerBet: '-0.457', expectedValue: '-0.450' }  // Very close!
]
// More trials → closer to expected value */
```

6. Opportunity Cost in Decisions

Every decision has a hidden cost: What you give up

```
function compareOpportunities(options) {
  // Sort by expected value
  const sorted = options
    .map(opt => ({
      ...opt,
      expectedValue: expectedValue(opt.outcomes),
      timeInvested: opt.timeWeeks
    }))
    .sort((a, b) => b.expectedValue - a.expectedValue);

  // Best option's EV becomes the opportunity cost for others
  const bestEV = sorted[0].expectedValue;

  return sorted.map(opt => ({
    name: opt.name,
    expectedValue: opt.expectedValue,
    timeWeeks: opt.timeWeeks,
    opportunityCost: bestEV - opt.expectedValue,
    trueValue: opt.expectedValue - (bestEV - opt.expectedValue)  // Adjusted for
opportunity cost
  }));
}

compareOpportunities([
  {
    name: 'Feature A',
    timeWeeks: 4,
    outcomes: [{ value: 50000, probability: 0.7 }, { value: 10000, probability: 0.3
}]
  },
  {
    name: 'Feature B',
    timeWeeks: 2,
    outcomes: [{ value: 25000, probability: 0.8 }, { value: 5000, probability: 0.2
}]
  },
  {
    name: 'Feature C',
    timeWeeks: 6,
    outcomes: [{ value: 80000, probability: 0.5 }, { value: 20000, probability: 0.5
}]
  }
]);
```

```
/* [
  { name: 'Feature C', expectedValue: 50000, opportunityCost: 0, trueValue: 50000 },
← Best
  { name: 'Feature A', expectedValue: 38000, opportunityCost: 12000, trueValue:
26000 },
  { name: 'Feature B', expectedValue: 21000, opportunityCost: 29000, trueValue:
-8000 }
]
// Feature B has positive EV, but negative when accounting for opportunity cost! */
```

Software Engineering Connections

1. A/B Test Decisions

```
function shouldWeShipVariant(controlConversion, variantConversion, sampleSize,
confidenceLevel = 0.95) {
  // Simplified: Expected revenue difference
  const expectedLift = variantConversion - controlConversion;
  const averageOrderValue = 50;  // $50 per conversion
  const monthlyTraffic = 10000;

  const expectedMonthlyGain = expectedLift * monthlyTraffic * averageOrderValue;

  // Risk: What if it's a false positive?
  const falsePositiveRisk = 1 - confidenceLevel;
  const riskAdjustedGain = expectedMonthlyGain * (1 - falsePositiveRisk);

  return {
    expectedMonthlyGain: Math.round(expectedMonthlyGain),
    riskAdjustedGain: Math.round(riskAdjustedGain),
    annualValue: Math.round(riskAdjustedGain * 12),
    decision: riskAdjustedGain > 0 ? 'Ship it' : 'Keep testing'
  };
}

// Control: 3% conversion, Variant: 3.5% conversion
shouldWeShipVariant(0.03, 0.035, 5000, 0.95);
/* {
  expectedMonthlyGain: 2500,
  riskAdjustedGain: 2375,  // Discounted for 5% false positive risk
  annualValue: 28500,
  decision: 'Ship it'
} */
```

2. Hiring Decisions

```
function compareHiringOptions(options) {
  return options.map(opt => {
    const productivityEV = expectedValue(opt.productivityOutcomes);
```

```
    const tenureEV = expectedValue(opt.tenureOutcomes);
    const totalValue = productivityEV * tenureEV - opt.salary;

    return {
      candidate: opt.name,
      salary: opt.salary,
      expectedProductivity: Math.round(productivityEV),
      expectedTenure: tenureEV.toFixed(1) + ' years',
      lifeValue: Math.round(totalValue),
      roi: (totalValue / opt.salary * 100).toFixed(1) + '%'
    };
  }).sort((a, b) => b.lifeValue - a.lifeValue);
}

compareHiringOptions([
  {
    name: 'Senior Engineer',
    salary: 150000,
    productivityOutcomes: [
      { value: 300000, probability: 0.7 },  // 70% chance of high productivity
      { value: 180000, probability: 0.3 }
    ],
    tenureOutcomes: [
      { value: 3, probability: 0.6 },
      { value: 5, probability: 0.4 }
    ]
  },
  {
    name: 'Junior Engineer',
    salary: 80000,
    productivityOutcomes: [
      { value: 120000, probability: 0.5 },
      { value: 80000, probability: 0.5 }
    ],
    tenureOutcomes: [
      { value: 2, probability: 0.5 },
      { value: 4, probability: 0.5 }
    ]
  }
]);
/* [
  { candidate: 'Senior', salary: 150000, expectedProductivity: 246000,
expectedTenure: '3.8 years', lifeValue: 784800, roi: '523.2%' },
  { candidate: 'Junior', salary: 80000, expectedProductivity: 100000,
expectedTenure: '3.0 years', lifeValue: 220000, roi: '275.0%' }
]
// Senior has higher absolute value, Junior has better capital efficiency */
```

3. Infrastructure Investment

```
function evaluateInfrastructureInvestment(cost, outcomes, years) {
  const annualEV = expectedValue(outcomes);
  const totalEV = annualEV * years - cost;
  const paybackPeriod = cost / annualEV;

  // Risk-adjusted (discount future uncertain savings)
  const discountRate = 0.15;
  let npv = -cost;
  for (let year = 1; year <= years; year++) {
    npv += annualEV / Math.pow(1 + discountRate, year);
  }

  return {
    upfrontCost: cost,
    expectedAnnualSavings: Math.round(annualEV),
    totalExpectedValue: Math.round(totalEV),
    paybackYears: paybackPeriod.toFixed(1),
    riskAdjustedNPV: Math.round(npv),
    decision: npv > 0 ? 'Invest' : 'Pass'
  };
}

// CI/CD investment: $50k upfront
evaluateInfrastructureInvestment(
  50000,
  [
    { value: 40000, probability: 0.5 },   // 50% chance: Great success
    { value: 20000, probability: 0.3 },   // 30% chance: Moderate
    { value: 5000, probability: 0.2 }     // 20% chance: Disappointing
  ],
  5
);
/* {
  upfrontCost: 50000,
  expectedAnnualSavings: 27000,
  totalExpectedValue: 85000,
  paybackYears: '1.9',
  riskAdjustedNPV: 40502,
  decision: 'Invest'
} */
```

4. Incident Response Tradeoffs

```
function incidentResponseStrategy(strategies) {
  return strategies.map(s => {
    const expectedDowntime = expectedValue(s.downtimeOutcomes);
    const downframeCost = expectedDowntime * 1000;  // $1000/hour downtime
    const totalCost = s.investmentCost + downframeCost;

    return {
```

```
      strategy: s.name,
      investment: s.investmentCost,
      expectedDowntimeHours: expectedDowntime.toFixed(1),
      expectedDowntimeCost: Math.round(downframeCost),
      totalExpectedCost: Math.round(totalCost)
    };
  }).sort((a, b) => a.totalExpectedCost - b.totalExpectedCost);
}

incidentResponseStrategy([
  {
    name: 'No Monitoring',
    investmentCost: 0,
    downtimeOutcomes: [
      { value: 24, probability: 0.3 },  // 30% chance: 24 hour outage
      { value: 4, probability: 0.7 }    // 70% chance: 4 hour outage
    ]
  },
  {
    name: 'Basic Monitoring',
    investmentCost: 5000,
    downtimeOutcomes: [
      { value: 2, probability: 0.2 },
      { value: 0.5, probability: 0.8 }
    ]
  },
  {
    name: 'Full Observability',
    investmentCost: 20000,
    downtimeOutcomes: [
      { value: 0.5, probability: 0.1 },
      { value: 0.1, probability: 0.9 }
    ]
  }
]);
/* [
  { strategy: 'Basic Monitoring', investment: 5000, expectedDowntimeHours: '0.8',
expectedDowntimeCost: 800, totalExpectedCost: 5800 },  ← Best
  { strategy: 'Full Observability', investment: 20000, expectedDowntimeHours: '0.1',
expectedDowntimeCost: 140, totalExpectedCost: 20140 },
  { strategy: 'No Monitoring', investment: 0, expectedDowntimeHours: '10.0',
expectedDowntimeCost: 10000, totalExpectedCost: 10000 }
]
// Basic monitoring has lowest total expected cost */
```

5. Feature Flags & Progressive Rollout

```
function rolloutStrategy(revenueImpact, userBase) {
  const strategies = [
    {
      name: 'Big Bang (100%)',
```

```
      percentRolled: 1.0,
      bugRisk: 0.15,  // 15% chance of critical bug
      bugImpact: -revenueImpact * 2  // Negative press, churn
    },
    {
      name: 'Canary (10%)',
      percentRolled: 0.10,
      bugRisk: 0.15,
      bugImpact: -revenueImpact * 0.2  // Only affects 10% of users
    },
    {
      name: 'Progressive (50%)',
      percentRolled: 0.50,
      bugRisk: 0.10,  // Caught bugs at 10%, fix before 50%
      bugImpact: -revenueImpact * 0.6
    }
  ];

  return strategies.map(s => {
    const successEV = (1 - s.bugRisk) * revenueImpact * s.percentRolled;
    const failureEV = s.bugRisk * s.bugImpact;
    const netEV = successEV + failureEV;

    return {
      strategy: s.name,
      expectedValue: Math.round(netEV),
      worstCase: Math.round(s.bugImpact),
      bestCase: Math.round(revenueImpact * s.percentRolled)
    };
  }).sort((a, b) => b.expectedValue - a.expectedValue);
}

// Feature worth $100k/month if successful
rolloutStrategy(100000, 1000000);
/* [
  { strategy: 'Progressive (50%)', expectedValue: 39000, worstCase: -60000,
bestCase: 50000 },  ← Best EV
  { strategy: 'Canary (10%)', expectedValue: 5500, worstCase: -20000, bestCase:
10000 },
  { strategy: 'Big Bang (100%)', expectedValue: 55000, worstCase: -200000, bestCase:
100000 }
]
// Progressive rollout has best risk-adjusted EV, Big Bang risks catastrophe */
```

Common Misconceptions

❌ "Expected value tells me what will happen"

No: It tells you the AVERAGE over many trials.

```
// Lottery ticket: $1 bet
const lotteryEV = expectedValue([
  { value: 1000000, probability: 0.0000001 },  // 1 in 10 million wins $1M
  { value: -1, probability: 0.9999999 }
]);

console.log(lotteryEV);  // -$0.90

// You'll MOST LIKELY lose $1 (99.99999% of the time)
// But in expectation, you lose $0.90 on average
// Over 10 million tickets, average loss is $0.90/ticket
```

❌ "Always choose the highest expected value"

Sometimes risk tolerance matters:

```
function riskToleranceExample() {
  const optionA = { name: 'Safe', ev: 10000, outcomes: [{ value: 10000, probability:
1.0 }] };
  const optionB = {
    name: 'Risky',
    ev: expectedValue([{ value: 1000000, probability: 0.02 }, { value: 0,
probability: 0.98 }]),
    outcomes: [{ value: 1000000, probability: 0.02 }, { value: 0, probability: 0.98
}]
  };

  return [optionA, optionB];
}

riskToleranceExample();
// Option A: $10k guaranteed
// Option B: $20k expected value (2% × $1M)

// If this is your ONLY shot (startup, can't afford to fail): Choose A
// If you can play many times (VC with portfolio): Choose B
```

❌ "Sunk costs should factor into decisions"

```
function sunkCostFallacy(alreadySpent, remainingCost, expectedValue) {
  const wrongDecision = {
    reasoning: "We've already spent $X, we should finish",
    calculation: alreadySpent + remainingCost,
    considersEV: false
  };

  const correctDecision = {
    reasoning: "Ignore sunk cost, compare remaining cost vs EV",
    calculation: expectedValue - remainingCost,
```

```
    shouldContinue: expectedValue > remainingCost
  };

  return { wrongDecision, correctDecision };
}

// You've spent $50k on a feature. Need $20k more to finish. Expected value: $15k.
sunkCostFallacy(50000, 20000, 15000);
/* {
  wrongDecision: { reasoning: "We've spent $50k, finish it!", calculation: 70000 },
  correctDecision: {
    reasoning: "Ignore $50k (sunk), compare $20k remaining vs $15k EV",
    shouldContinue: false  // $15k EV < $20k remaining cost
  }
}
// Correct answer: ABANDON. Spending another $20k to get $15k is bad. */
```

❌ "Rare high-impact events can be ignored"

Black Swans:

```
function blackSwanRisk(normalOutcomes, rareDisaster) {
  const normalEV = expectedValue(normalOutcomes);
  const totalEV = normalEV + (rareDisaster.value * rareDisaster.probability);

  return {
    ignoringRareEvent: normalEV,
    includingRareEvent: totalEV,
    difference: Math.abs(totalEV - normalEV),
    percentageImpact: ((Math.abs(totalEV - normalEV) / Math.abs(normalEV)) *
100).toFixed(1) + '%'
  };
}

// Startup: 95% chance of $10M exit, 4% chance of $0, 1% chance of lawsuit costing
$50M
blackSwanRisk(
  [{ value: 10000000, probability: 0.95 }, { value: 0, probability: 0.04 }],
  { value: -50000000, probability: 0.01 }
);
/* {
  ignoringRareEvent: 9500000,
  includingRareEvent: 9000000,
  difference: 500000,
  percentageImpact: '5.3%'
}
// Ignoring 1% tail risk undervalues downside by $500k! */
```

Practical Mini-Exercises

▶ **Exercise 1: Feature Prioritization** (Click to expand)
▶ **Exercise 2: Server Redundancy** (Click to expand)
▶ **Exercise 3: A/B Test** (Click to expand)

Summary Cheat Sheet

```
// EXPECTED VALUE FORMULA
EV = Σ (Outcome_i × Probability_i)

// DECISION RULE
Choose option with highest EV (if risk-neutral)
Adjust for risk tolerance (Kelly Criterion)

// KEY INSIGHTS
1. EV = long-term average, not prediction of single outcome
2. Ignore sunk costs (only future matters)
3. Account for opportunity cost
4. Rare disasters can dominate EV (1% × $1M loss = -$10k)
5. Law of Large Numbers: More trials → actual average → EV

// RISK ADJUSTMENT
Risk-Adjusted EV = EV - (Risk Premium)
Risk Premium ∝ Variance × Risk Aversion
```

Mental Models:

- **Casino**: Think like the house (many bets, EV wins)
- **Portfolio**: Diversify to play many +EV bets
- **Kelly**: Bet proportional to edge and confidence
- **Black Swans**: Don't ignore tail risks

Next Steps

✅ **You've completed**: Phase 4 (Finance & Decision Making)
➡️ **Up next**: 13. Linear Algebra for Engineers - Vectors, matrices, transformations, why ML/graphics/search use it

Before moving on, calculate:

```
// You can hire 2 engineers: Senior ($150k, 80% success, $300k value) or
// Junior ($90k, 60% success, $150k value). Budget: $200k. What's optimal?

function hiringOptimization() {
  // Your solution here
  // Hint: Consider combinations and expected values
}
```
