1 Решение практического задания 1

В этой главе

- ✓ Симуляции карточной игры.
- ✓ Оптимизация стратегии вычисления вероятности.
- ✓ Доверительные интервалы.

Мы планируем сыграть в игру, в которой карты поочередно открываются до тех пор, пока мы не говорим дилеру: «Стоп». После этого открывается еще одна карта: если она оказывается красной, мы выигрываем доллар, в противном случае теряем его. Задача — определить стратегию, которая наилучшим образом прогнозирует открытие именно красной карты. Для этого сделаем следующее.

1. Разработаем несколько стратегий для прогнозирования выпадения красных карт в случайно перетасованной колоде.
2. Применим каждую стратегию для нескольких симуляций, вычислив вероятность успеха с высоким доверительным интервалом. Если эти вычисления окажутся неподъемными, мы переключимся на те стратегии, которые лучше всего справятся с пространством элементарных исходов для десяти карт.
3. Вернем простейшую стратегию, связанную с наивысшей вероятностью успеха.

ВНИМАНИЕ!

Спойлер! Вскоре вы узнаете решение для рассматриваемого исследования. Я настоятельно рекомендую вам попытаться решить эту задачу самостоятельно, прежде чем изучать готовое решение. Описание задачи находится в начале этого исследования.

4.1. ПРОГНОЗИРОВАНИЕ КРАСНЫХ КАРТ В ПЕРЕТАСОВАННОЙ КОЛОДЕ

Начнем с создания колоды, содержащей 26 красных и 26 черных карт (листинг 4.1). Черные карты представлены нулями, а красные — единицами.

Листинг 4.1. Моделирование колоды из 52 карт

```
red_cards = 26 * [1]
black_cards = 26 * [0]
unshuffled_deck = red_cards + black_cards
```

Далее колода перетасовывается (листинг 4.2).

Листинг 4.2. Перетасовывание колоды из 52 карт

```
np.random.seed(1)
shuffled_deck = np.random.permutation(unshuffled_deck)
```

Теперь мы поочередно открываем карты, останавливаясь, когда очередная с наибольшей вероятностью должна оказаться красной. После этого открываем очередную карту и выигрываем, если она действительно оказывается красной.

Теперь осталось лишь решить, когда же останавливаться. Одной из предполагаемых простых стратегий будет закончить игру, когда количество оставшихся в колоде красных карт будет больше числа оставшихся черных. Применим эту стратегию к перетасованной колоде (листинг 4.3).

Листинг 4.3. Перенос в код стратегии для карточной игры

```
remaining_red_cards = 26
for i, card in enumerate(shuffled_deck[:-1]):
    remaining_red_cards -= card
    remaining_total_cards = 52 - i - 1
    if remaining_red_cards / remaining_total_cards > 0.5:
        break

print(f"Stopping the game at index {i}.")
final_card = shuffled_deck[i + 1]
color = 'red' if final_card else 0
print(f"The next card in the deck is {'red' if final_card else 'black'}.")
print(f"We have {'won' if final_card else 'lost'}!")
```

Вычитает общее число извлеченных карт из 52. Это число равно i+1, поскольку отсчет i начался с нуля. В качестве альтернативы можно выполнить enumerate(shuffled_deck[:-1], 1), чтобы отсчет i начинался с единицы

```
Stopping the game at index 1.
The next card in the deck is red.
We have won!
```

Такая стратегия привела к победе на первой же попытке. По ее правилам мы останавливаемся, когда доля красных карт оказывается больше половины общего числа оставшихся карт. Эту долю можно обобщить как равную параметру `min_red_fraction`, чтобы останавливаться, когда соотношение красных карт окажется

больше указанного значения параметра. В общем виде эта стратегия реализована в листинге 4.4 при `min_red_fraction`, равном `0.5`.

Листинг 4.4. Обобщение победной стратегии для карточной игры

```
np.random.seed(0)
total_cards = 52
total_red_cards = 26
def execute_strategy(min_fraction_red=0.5, shuffled_deck=None,
                     return_index=False):
    if shuffled_deck is None:
        shuffled_deck = np.random.permutation(unshuffled_deck)

    remaining_red_cards = total_red_cards

    for i, card in enumerate(shuffled_deck[:-1]):
        remaining_red_cards -= card
        fraction_red_cards = remaining_red_cards / (total_cards - i - 1)
        if fraction_red_cards > min_fraction_red:
            break
    return (i+1, shuffled_deck[i+1]) if return_index else shuffled_deck[i+1]
```

Если входной колоды нет, перемешивает неперетасованную колоду

При необходимости возвращает вместе с последней картой ее индекс

4.1.1. Оценка вероятности успеха стратегии

Применим нашу базовую стратегию к серии из 1000 случайных перемешиваний карт, проще говоря, игр (листинг 4.5).

Листинг 4.5. Выполнение стратегии в 1000 игр

```
observations = np.array([execute_strategy() for _ in range(1000)])
```

Общая доля единиц в результатах соответствует полученной доле красных карт, а значит, доле побед. Вычислить ее можно, сложив единицы в `observations` и разделив результат на размер массива. В качестве альтернативы этот расчет можно выполнить с помощью `observations.mean()` (листинг 4.6).

Листинг 4.6. Вычисление частоты побед

```
frequency_wins = observations.sum() / 1000
assert frequency_wins == observations.mean()
print(f"The frequency of wins is {frequency_wins}")
```

```
The frequency of wins is 0.511
```

Мы победили в 51,1 % игр! Похоже, что наша стратегия работает: 511 побед и 489 поражений принесли нам общую прибыль 22 доллара (листинг 4.7).

Выбранная стратегия отлично сработала при 1000 игр. Теперь нарисуем график сходимости частоты побед для серии разных количеств игр — от 1 до 10 000 (листинг 4.8; рис. 4.1).

Листинг 4.7. Вычисление общей прибыли

```
dollars_won = frequency_wins * 1000
dollars_lost = (1 – frequency_wins) * 1000
total_profit = dollars_won – dollars_lost
print(f"Total profit is ${total_profit:.2f}")

Total profit is $22.00
```

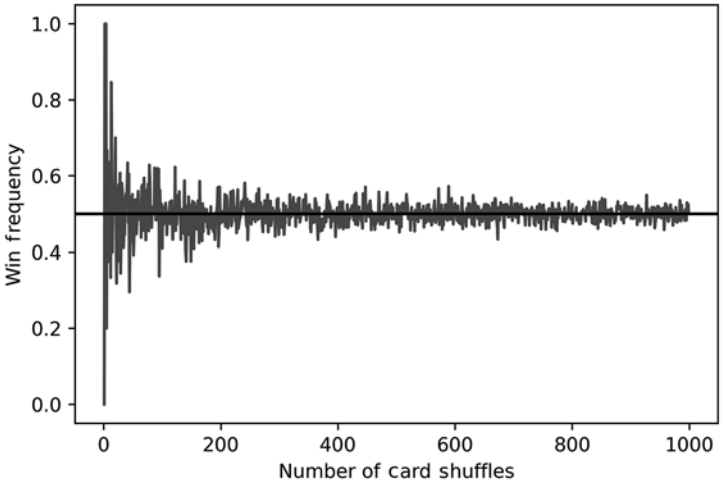

Рис. 4.1. Соотношение количества сыгранных игр и частоты побед. Частоты колеблются вокруг значения 0,5. Здесь нельзя сказать, находится вероятность победы выше или ниже 0,5

Листинг 4.8. Отрисовка симулированных частот побед

```
np.random.seed(0)
def repeat_game(number_repeats):
    observations = np.array([execute_strategy()
                             for _ in range(number_repeats)])
    return observations.mean()

frequencies = []
for i in range(1, 1000):
    frequencies.append(repeat_game(i))

plt.plot(list(range(1, 1000)), frequencies)
plt.axhline(0.5, color='k')
plt.xlabel('Number of Card Shuffles')
plt.ylabel('Win-Frequency')
plt.show()
print(f"The win-frequency for 10,000 shuffles is {frequencies[-1]}")

The win-frequency for 10,000 shuffles is 0.5035035035035035
```

Возвращает частоту побед в заданном числе игр

При 10 000 игр эта стратегия дает частоту побед более 50 %. Тем не менее в процессе анализа она иногда проседает ниже 50 %. Какова же наша уверенность в том, что вероятность победы фактически окажется больше 0,5? Выяснить это можно с помощью анализа доверительного интервала (рис. 4.2). Этот интервал мы вычисляем путем 300-кратной оценки 10 000 перетасовываний, что в сумме дает 3 млн игр. Перемешивание массива вычислительно затратно, поэтому код листинга 4.9 выполняется примерно 40 с.

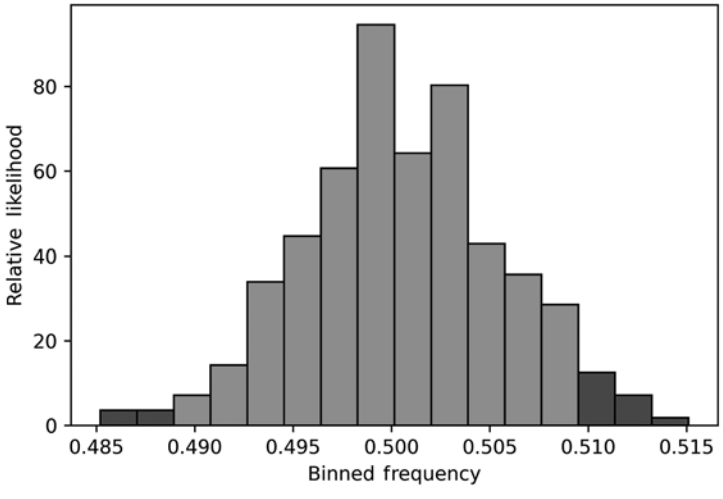

Рис. 4.2. Гистограмма 300 частот, распределенных по интервалам в соотношении с их относительными вероятностями. Выделенные столбцы обозначают доверительный интервал 95 %, охватывающий диапазон частот примерно от 0,488 до 0,508

Листинг 4.9. Вычисление доверительного интервала для 3 млн игр

```
np.random.seed(0)
frequency_array = np.array([repeat_game(10000) for _ in range(300)])
likelihoods, bin_edges, patches = plt.hist(frequency_array, bins='auto',
                                           edgecolor='black', density=True)
bin_width = bin_edges[1] – bin_edges[0]
start_index, end_index = compute_high_confidence_interval(likelihoods,
    bin_width)
for i in range(start_index, end_index):
    patches[i].set_facecolor('yellow')
plt.xlabel('Binned Frequency')
plt.ylabel('Relative Likelihood')

plt.show()

The frequency range 0.488938 – 0.509494 represents a 97.00 % confidence interval
```

Напомню, что функцию compute_high_confidence_interval мы определили в главе 3

Создается довольно высокая уверенность в том, что фактическая вероятность находится где-то между 0,488 и 0,509. Тем не менее до сих пор неизвестно, выше

она 0,5 или ниже. И это проблема, так как даже малейшая неточность в определении истинной вероятности может привести к потере денег.

Представьте, что истинная вероятность составляет 0,5001. Если мы применим нашу стратегию к 1 млрд игр, то можем ожидать выигрыш 200 000 долларов. А теперь предположим, что мы ошибались и фактическая вероятность равна 0,4999. В таком случае мы эти 200 000 долларов потеряем. Едва заметная ошибка в четвертом десятичном разряде обойдется не в одну сотню тысяч долларов.

Поэтому нам необходима абсолютная уверенность в том, что истинная вероятность находится выше 0,5. Значит, нужно сузить доверительный интервал путем увеличения количества игр в одной серии, за что мы поплатимся более длительным временем выполнения. Код листинга 4.10 анализирует 50 000 игр 3000 раз.

ПРЕДУПРЕЖДЕНИЕ

Следующий код выполняется примерно час.

Листинг 4.10. Вычисление доверительного интервала для 150 млн игр

```
np.random.seed(0)

frequency_array = np.array([repeat_game(50000) for _ in range(3000)])
likelihoods, bin_edges = np.histogram(frequency_array, bins='auto', density=True)
bin_width = bin_edges[1] – bin_edges[0]
compute_high_confidence_interval(likelihoods, bin_width)

The frequency range 0.495601 – 0.504345 represents a 96.03 % confidence interval
```

Мы выполнили анализ. К сожалению, новый доверительный интервал по-прежнему не проясняет, находится ли истинная вероятность выше 0,5. Что же делать? Дальнейшее увеличение числа перетасовываний будет уже вычислительно нецелесообразным (если только вы не горите желанием пару дней подождать завершения симуляции). Возможно, улучшения удастся добиться, увеличив `min_red_fraction` с 0,50 до 0,75. Обновим нашу стратегию и пойдем погуляем, пока симуляция выполняется в течение очередного часа.

ПРЕДУПРЕЖДЕНИЕ

Код листинга 4.11 выполняется около часа.

Листинг 4.11. Вычисление доверительного интервала для обновленной стратегии

```
np.random.seed(0)
def repeat_game(number_repeats, min_red_fraction):
    observations = np.array([execute_strategy(min_red_fraction)
                             for _ in range(number_repeats)])
    return observations.mean()

frequency_array = np.array([repeat_game(50000, 0.75) for _ in range(3000)])
likelihoods, bin_edges = np.histogram(frequency_array, bins='auto', density=True)
```

```
bin_width = bin_edges[1] – bin_edges[0]
compute_high_confidence_interval(likelihoods, bin_width)

The frequency range 0.495535 – 0.504344 represents a 96.43 % confidence interval
```

Ни в какую! Охват доверительного интервала по-прежнему не дает конкретики, включая в себя как выгодные, так и невыгодные вероятности.

Возможно, удастся прояснить этот вопрос, применив наши стратегии к колоде из десяти карт. Ее пространство элементарных исходов можно изучить полноценно, что позволит вычислить точные шансы на победу.

4.2. ОПТИМИЗАЦИЯ СТРАТЕГИЙ С ПОМОЩЬЮ ПРОСТРАНСТВА ИСХОДОВ ДЛЯ КОЛОДЫ ИЗ ДЕСЯТИ КАРТ

Код листинга 4.12 просчитывает пространство элементарных исходов для колоды из десяти карт, после чего применяет к нему базовую стратегию. Итоговым выводом является вероятность получения выигрыша с помощью этой стратегии.

Листинг 4.12. Применение базовой стратегии к колоде из десяти карт

```
total_cards = 10
total_red_cards = int(total_cards / 2)
total_black_cards = total_red_cards
unshuffled_deck = [1] * total_red_cards + [0] * total_black_cards
sample_space = set(itertools.permutations(unshuffled_deck))
win_condition = lambda x: execute_strategy(shuffled_deck=np.array(x))
prob_win = compute_event_probability(win_condition, sample_space)
print(f"Probability of a win is {prob_win}")

Probability of a win is 0.5
```

Напомню, что itertools мы импортировали в главе 3

Условие события, при котором наша базовая стратегия ведет к выигрышу

Функцию compute_event_probability мы определили в главе 1

Удивительно, но наша основная стратегия дает победу лишь в 50 % случаев. Это не лучше, чем выбирать первую карту наугад! Возможно, параметр `min_red_fraction` был недостаточно мал. Выяснить это можно, проанализировав все значения с двумя десятичными разрядами между 0,5 и 1,0. Код листинга 4.13 вычисляет вероятности победы для диапазона значений `min_red_fraction`, возвращая минимальные и максимальные вероятности.

Листинг 4.13. Применение нескольких стратегий к колоде из десяти карт

```
def scan_strategies():
    fractions = [value / 100 for value in range(50, 100)]
    probabilities = []
    for frac in fractions:
        win_condition = lambda x: execute_strategy(frac, shuffled_deck=np.array(x))
```

```
        probabilities.append(compute_event_probability(win_condition, sample_space))
    return probabilities

probabilities = scan_strategies()
print(f"Lowest probability of win is {min(probabilities)}")
print(f"Highest probability of win is {max(probabilities)}")

Lowest probability of win is 0.5
Highest probability of win is 0.5
```

И минимальная, и максимальная вероятности равны 0,5. Ни одна из наших стратегий не превзошла случайный выбор карты. Возможно, корректировка размера колоды позволит добиться улучшения. Проанализируем пространства элементарных исходов колод, состоящих из двух, четырех, шести и восьми карт. Применим обе стратегии к каждому пространству и вернем их вероятности победы, среди которых найдем ту, которая не равна 0,5 (листинг 4.14).

Листинг 4.14. Применение нескольких стратегий к нескольким колодам

```
for total_cards in [2, 4, 6, 8]:
    total_red_cards = int(total_cards / 2)
    total_black_cards = total_red_cards
    unshuffled_deck = [1] * total_red_cards + [0] * total_black_cards

    sample_space = set(itertools.permutations(unshuffled_deck))
    probabilities = scan_strategies()
    if all(prob == 0.5 for prob in probabilities):
        print(f"No winning strategy found for deck of size {total_cards}")
    else:
        print(f"Winning strategy found for deck of size {total_cards}")

No winning strategy found for deck of size 2
No winning strategy found for deck of size 4
No winning strategy found for deck of size 6
No winning strategy found for deck of size 8
```

Все стратегии, будучи примененными к небольшим колодам, дают вероятность победы 0,5. При каждом увеличении размера колоды мы добавляем в нее две карты, но к улучшению показателей это не ведет. Стратегия, которая не справляется с колодой из двух карт, не справляется и с колодой из четырех, а стратегия, которая проваливается для восьми карт, проваливается также для десяти. Эту логику можно экстраполировать и дальше. Стратегия, которая не справляется с колодой из десяти карт, наверняка не справится и с 12, 14 и 16 картами.

В конечном итоге она не подойдет и для колоды из 52 карт. С качественной точки зрения это индуктивное рассуждение имеет смысл, и его можно подтвердить математически. Но сейчас мы не хотим нагружать себя математикой. Нам важно опровергнуть сигналы своей интуиции. Наши стратегии не работают для колоды из десяти карт, и у нас нет убедительных оснований рассчитывать, что они сработают для колоды из 52 карт. Почему же они не справляются?

Казалось, что начальная стратегия была разумна: если в колоде остается больше красных карт, чем черных, то вероятность извлечь красную выше. Однако мы допустили оплошность, не приняв во внимание сценарии, в которых красные карты никогда не превосходят по количеству черные. Предположим, что первые 26 карт красные, а остальные черные. В таких условиях наши стратегии не дадут сигнал остановки игры и мы проиграем. Или возьмем, к примеру, перетасованную колоду, в которой первые 25 карт красные, следующие 26 — черные и последняя — красная. В этом случае стратегии также не найдут удачного места для остановки, но мы все равно победим. Получается, что каждая стратегия может привести к одному из четырех сценариев.

- Стратегия останавливает игру, и очередная карта оказывается красной. Победа.
- Стратегия останавливает игру, и очередная карта оказывается черной. Проигрыш.
- Стратегия не останавливает игру, и последняя карта оказывается красной. Победа.
- Стратегия не останавливает игру, и последняя карта оказывается черной. Проигрыш.

Давайте проанализируем, как часто эти четыре сценария возникают среди 50 000 игр. Мы запишем эти частоты по всем значениям, входящим в диапазон `min_red_fraction`. После этого на графике отобразим каждое значение `min_red_fraction` относительно частоты наблюдения каждого из четырех сценариев (листинг 4.15; рис. 4.3).

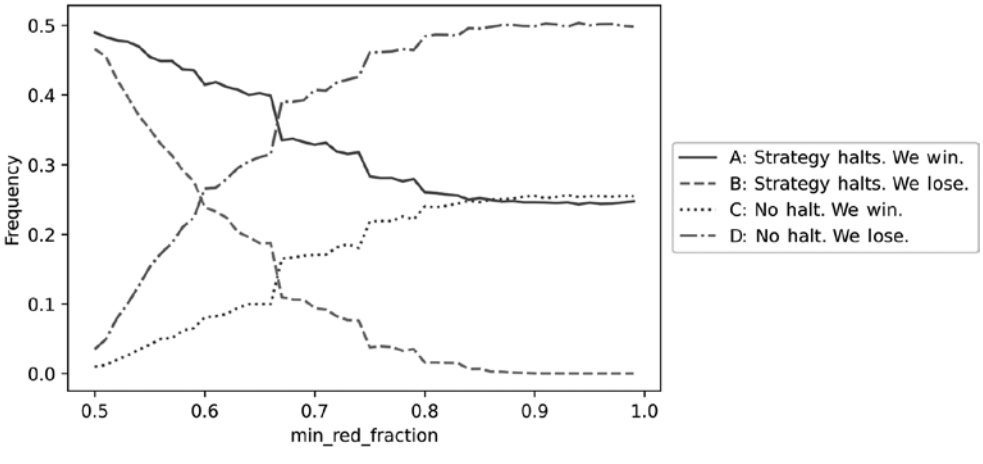

Рис. 4.3. Параметр min_red_fraction, отображенный относительно частот возникновения всех четырех сценариев. Частота сценария A изначально равна примерно 0,49, но в итоге падает до 0,25. Частота сценария C начинается примерно с 0,01 и в результате доходит до 0,25. Сумма частот A и C сохраняется около 0,5, тем самым отражая 50%-ный шанс на победу

Листинг 4.15. Построение графика результатов стратегий для колоды из 52 карт

```
np.random.seed(0)
total_cards = 52
total_red_cards = 26
unshuffled_deck = red_cards + black_cards

def repeat_game_detailed(number_repeats, min_red_fraction):
    observations = [execute_strategy(min_red_fraction, return_index=True)
                    for _ in range(num_repeats)]
    successes = [index for index, card, in observations if card == 1]
    halt_success = len([index for index in successes if index != 51])
    no_halt_success = len(successes) - halt_success
    failures = [index for index, card, in observations if card == 0]
    halt_failure = len([index for index in failures if index != 51])
    no_halt_failure = len(failures) - halt_failure
    result = [halt_success, halt_failure, no_halt_success, no_halt_failure]
    return [r / number_repeats for r in result]

fractions = [value / 100 for value in range(50, 100)]
num_repeats = 50000
result_types = [[], [], [], []]

for fraction in fractions:
    result = repeat_game_detailed(num_repeats, fraction)
    for i in range(4):
        result_types[i].append(result[i])

plt.plot(fractions, result_types[0],
         label='A) Strategy Halts. We Win.')
plt.plot(fractions, result_types[1], linestyle='--',
         label='B) Strategy Halts. We Lose.')
plt.plot(fractions, result_types[2], linestyle=':',
         label='C) No Halt. We Win.')
plt.plot(fractions, result_types[3], linestyle='-.',
         label='D) No Halt. We Lose.')
plt.xlabel('min_red_fraction')
plt.ylabel('Frequency')
plt.legend(bbox_to_anchor=(1.0, 0.5))
plt.show()
```

Список всех случаев поражений

Сценарий, в котором остановка происходит и мы побеждаем

Список всех случаев побед

Мы выполняем стратегию для num_repeats симуляций

Сценарий, в котором остановки не происходит и мы побеждаем

Возвращение полученных частот всех четырех сценариев

Просмотр частот сценариев в нескольких стратегиях

Сценарий, в котором остановки не происходит и мы проигрываем

Сценарий, в котором остановка происходит и мы проигрываем

Параметр bbox_to_anchor используется для размещения легенды над графиком, чтобы исключить наложение на четыре кривые

Рассмотрим график при значении `min_red_fraction`, равном `0.5`. Здесь сценарий A (стратегия приводит к остановке игры и победе) является самым частым исходом с частотой примерно 0,49. В то же время остановка ведет к поражению примерно в 46 % случаев (сценарий B). Почему же мы сохраняем 50%-ный шанс на победу? Что ж, в 1 % случаев наша стратегия к остановке игры не приводит, но мы все

равно побеждаем (сценарий C). Слабость стратегии уравновешивается элементом случайности.

На графике по мере увеличения `min_red_fraction` частота сценария A также увеличивается. Чем мы сдержаннее играем, тем с меньшей вероятностью останавливаем игру преждевременно, благодаря чему побеждаем. В то же время растет показатель успеха стратегии C. Чем сдержаннее мы действуем, тем выше вероятность дойти до последней карты и выиграть по воле случая.

По ходу увеличения `min_red_fraction` сценарии A и C сходятся к частоте 0,25. Таким образом, вероятность победы остается 50 %. Иногда наша стратегия ведет к остановке и мы побеждаем. В других случаях она также ведет к остановке, но мы проигрываем. Любое преимущество, обеспечиваемое любой из стратегий, автоматически этими поражениями аннулируется. Тем не менее время от времени нам везет — стратегия не приводит к остановке, но в игре мы побеждаем. Эти счастливые победы компенсируют поражения, и вероятность выигрыша остается прежней. Что бы мы ни делали, вероятность выиграть будет 50/50 (листинг 4.16). Следовательно, оптимальной стратегией будет открытие первой карты перетасованной колоды.

Листинг 4.16. Оптимальная стратегия для победы

```
def optimal_strategy(shuffled_deck):
return shuffled_deck[0]
```

РЕЗЮМЕ

- Вероятности не всегда совпадают с нашей интуицией. По наитию мы предположили, что спланированная нами игра в карты пройдет удачнее, чем случайная. На деле же оказалось не так. Нужно быть внимательными, работая со случайными процессами. Стоит тщательно протестировать все интуитивные предположения, прежде чем делать ставку на будущий исход.
- Иногда даже крупномасштабные симуляции не способны выявить вероятность в рамках требуемого уровня точности. Однако за счет упрощения задачи можно получить более точное представление с помощью пространств элементарных исходов. Эти пространства позволяют проверить наше чутье. Если интуитивное решение не годится для упрощенной версии задачи, то наверняка оно не справится и с реальной ее версией.