CASE STUDY 1:

"FINDING THE WINNING STRATEGY IN A CARD GAME"

Big Data Analysis

AIT BOUIFADEN Amal

GUERRERO IBERICO Jenny

DE MACEDO Kevin

DAMAS Alexandra

INDEX

INTRODUCTION

Our aim is to play a card game in which the cards are iteratively flipped until we tell the dealer to stop. Then one additional card is flipped.

If that card is red, we win a dollar; otherwise, we lose a dollar.

Our goal is to discover a strategy that best pre-dicts a red card in the deck.

CASE PRESENTATION AND PROBLEM DEFINITION

52 cards in total

26 red cards

26 black cards

Find out the strategy that best predicts a red card in a shuffled deck.

III. DATA ANALYTICS APPROACH DEFINITION AND EXAMPLES(WITH THE HELP OF JUPITER)

$$P(A) = \frac{\text{Event seize}}{\text{Sample space seize}}$$

If 0<P(redcard)<1 → We have chances to draw a red card.

$\frac{26 \rightarrow \text{Red cards}}{52 \rightarrow \text{Total cards}}$ = 0.5 we have 50% chances to predict a red card in a shuffled deck.

If P(redc U blackc)=1 → "I bet you that the last card is either red or black"

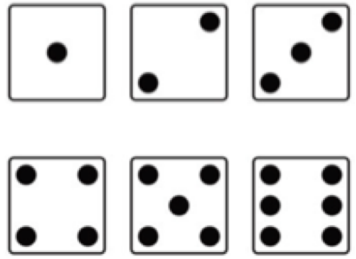
$\frac{26}{52} + \frac{26}{52} \rightarrow \frac{\text{Red + black cards}}{\text{Total cards}}$ = 1 this means that the event is certain, I'll win no matter what

If P(greencard)=0 → "I bet you that the last card is green"

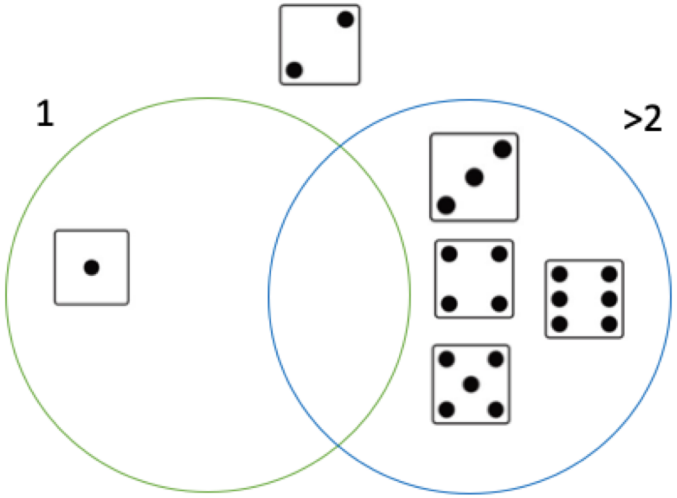
$\frac{0 \rightarrow \text{green cards}}{52 \rightarrow \text{Total cards}}$ = 0 means that the event is impossible

III. Data analytics approach definition and examples

A single dice is rolled, what is the probability of
a. Obtaining 1 or a digit greater than 2
b. Obtaining a number that is even or greater than 3

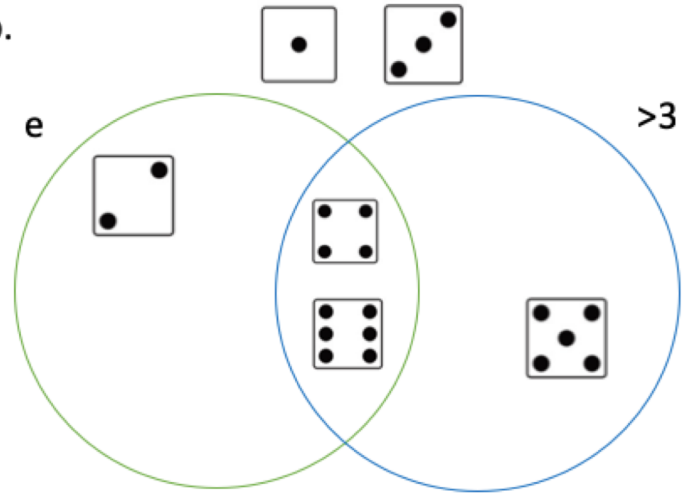


a.



$$P(1 \cup >2) = \frac{1}{6} + \frac{4}{6} = \frac{5}{6}$$

b.



$$P(e \cup >3) = \frac{3}{6} + \frac{3}{6} - \frac{2}{6} = \frac{4}{6}$$

III. Data analytics approach definition and examples

We have 7 cards face down, 5 black cards and 2 red cards. What is the probability to pick up 2 red cards by hazard

$$P(2\ redcards) = P(r) * P(r|r)$$

$$= \frac{2}{7} \times \frac{1}{6}$$

$$= \frac{2}{42} = 4.76\%$$

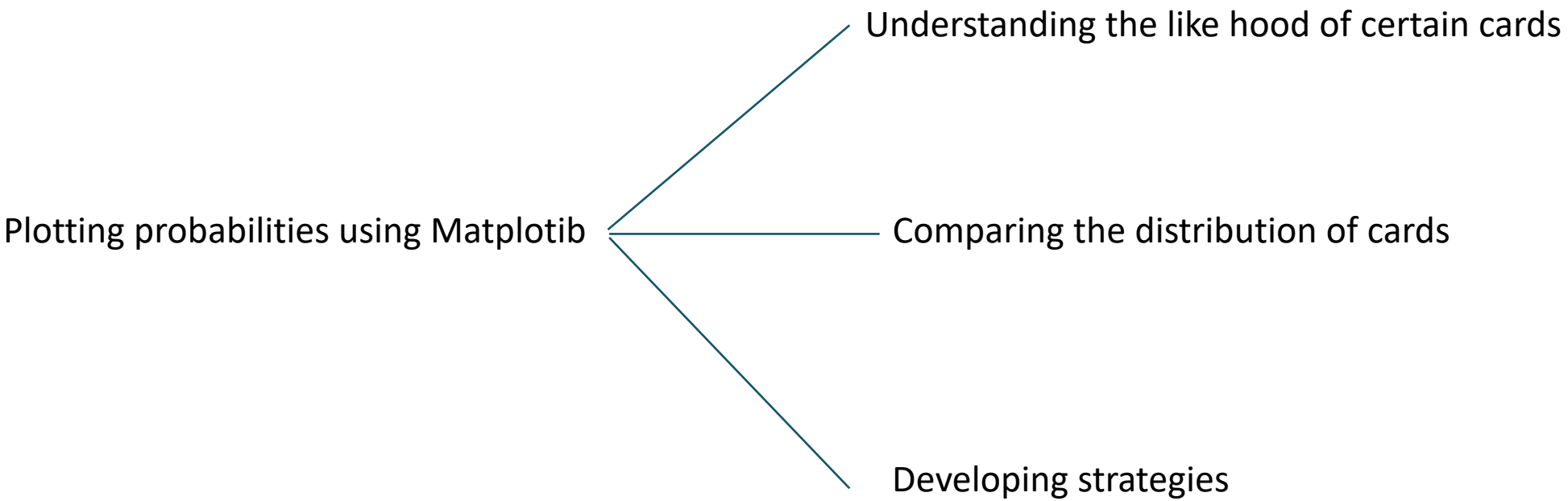
If we pick up a black card. What is the probability that this card could be an even number

$$= \frac{2}{5}$$

2 → Even numbers

5 → Total black cards

$$= 40\%$$


Plotting probabilities using Matplotib
Understanding the like hood of certain cards
Comparing the distribution of cards
Developing strategies

SOLUTION DEVELOPMENT AND ILLUSTRATION

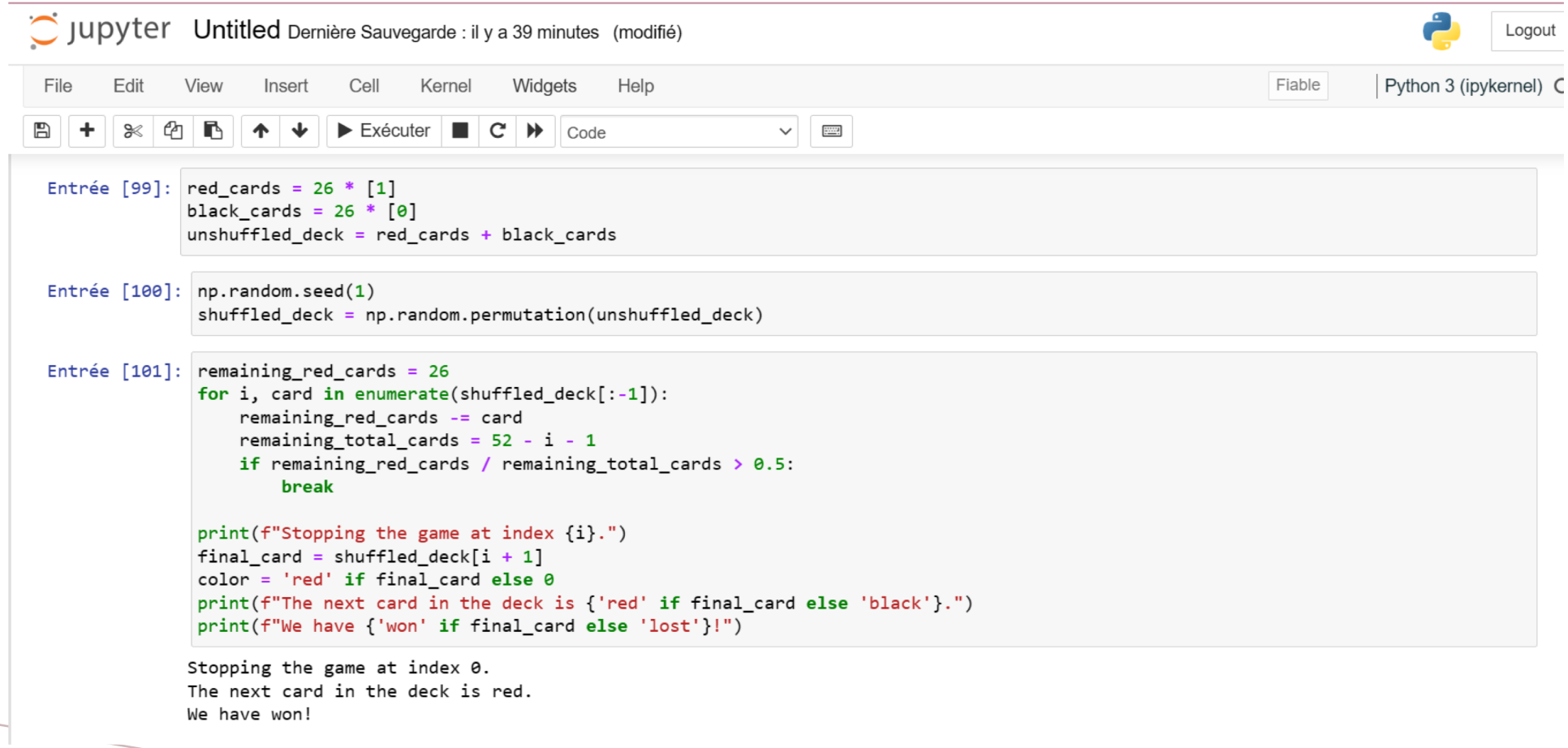
As we say, our aim is to play a card game in which the cards are iteratively flipped until we tell the dealer to stop. Then one additional card is flipped. If that card is red, we win a dollar; otherwise, we lose a dollar. Our goal is to discover a strategy that best pre-dicts a red card in the deck.

At first:

We start by creating a deck holding 26 red cards and 26 black cards. Black cards are represented by 0s, and red cards are represented by 1s.

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

WITH THE HELP OF JUPITER…



GENERALIZING THE CARD GAME STRATEGY..

jupyter Untitled Dernière Sauvegarde : il y a 41 minutes (auto-sauvegardé)

File Edit View Insert Cell Kernel Widgets Help Fiable Python 3 (ipykernel)

Exécuter Code

Entrée [102]:
```
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]
```

Entrée [103]:
```
np.random.seed(0)
observations = np.array([execute_strategy() for _ in range(1000)])
```

Entrée [104]:
```
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
```

OPTIMIZING STRATEGIES USING THE SAMPLE SPACE FOR A 10-CARD DECK

jupyter Untitled Dernière Sauvegarde : il y a une heure (auto-sauvegardé)

File Edit View Insert Cell Kernel Widgets Help Fiable Python 3 (ipykerne

Exécuter Code

Entrée [110]:
```
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
```

Entrée [111]:
```
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
```

PLOTTING STRATEGY OUTCOMES ACROSS A 52-CARD DECK

```
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()
```

This list contains all instances of losses.

Scenario where our strategy halts and we win

This list contains all instances of wins.

xecute a
y across
repeats
ulations.

where
rategy
't halt
ve win

We return the observed frequencies for all four scenarios.

We scan the scenario frequencies across multiple strategies.

Scenario where our strategy doesn't halt and we lose

Scenario where our strategy halts and we lose

The bbox_to_anchor parameter is used to position the legend above the plot to avoid overlap with the four plotted curves.

EVALUATION AND FEEDBACK: THE OPTIMAL WINNING STRATEGY

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

This function could be used as a starting point for developing a more sophisticated strategy.
For example, in this case it could be modified to return a specific card based on choosing the highest or lowest value card in the deck or to implement a more complex sequence of actions based on the cards drawn so far.

CONCLUSION

By simplifying our problem, we can utilize sample spaces to yield insights. Sample spaces allow us to test our intuition. If our intuitive solution fails on a toy version of the problem, it is also likely to fail on the actual version of the problem.