# Only keeping the version with back\_commission parameter and improved implementation

@staticmethod

def calculate\_score\_cast(back\_odds\_dict, lay\_odds\_dict, stake, commission=0.02, back\_commission=0.0):

"""Calculate potential profit/loss for Score Cast market with commission.

This market combines first goalscorer with correct score for higher odds. It allows betting on

which player will score first and what the final score will be, as a single bet.

Parameters:

back\_odds\_dict (dict): Dictionary of scorer+score combo -> back odds (e.g. {'Kane 2-0': 25.0, 'Salah 1-0': 20.0, ...})

lay\_odds\_dict (dict): Dictionary of scorer+score combo -> lay odds

stake (float): Stake amount for back bet

commission (float): Commission rate for lay bets (default 0.02 for Smarkets)

back\_commission (float): Commission rate for back bets (default 0.0 for Smarkets, 0.05 for Betfair)

Returns:

dict: Profit scenarios and other calculation details

Edge cases:

- Returns None if any odds are < 1.0

- Returns None if division by zero occurs during lay stake calculation

- Returns None if dictionaries have different keys

Example usage:

back\_odds = {'Kane 2-0': 25.0, 'Kane 1-0': 20.0, 'Salah 2-0': 30.0}

lay\_odds = {'Kane 2-0': 26.0, 'Kane 1-0': 21.0, 'Salah 2-0': 32.0}

result = calculate\_score\_cast(back\_odds, lay\_odds, 100, 0.02, 0.05)

"""

if not back\_odds\_dict or not lay\_odds\_dict:

logger.warning("Empty odds dictionaries in calculate\_score\_cast")

return None

if any(odds <= 0 for odds in back\_odds\_dict.values()) or any(odds <= 0 for odds in lay\_odds\_dict.values()):

logger.warning(f"Zero or negative odds in calculate\_score\_cast: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

if any(odds < 1.0 for odds in back\_odds\_dict.values()) or any(odds < 1.0 for odds in lay\_odds\_dict.values()):

logger.warning(f"Invalid odds in calculate\_score\_cast: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

# Check that both dictionaries have the same keys

if set(back\_odds\_dict.keys()) != set(lay\_odds\_dict.keys()):

logger.warning(f"Mismatched keys in calculate\_score\_cast: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

combinations = list(back\_odds\_dict.keys())

lay\_stakes = {}

# Calculate lay stakes for each combination

for combo in combinations:

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake, back\_odds\_dict[combo], lay\_odds\_dict[combo], commission)

if lay\_stake is None:

return None

lay\_stakes[combo] = lay\_stake

# Calculate profit scenarios using helper method

profit\_scenarios = StrategyCalculators.calculate\_profit\_scenarios(

stake, back\_odds\_dict, lay\_odds\_dict, lay\_stakes, commission, back\_commission

)

min\_profit = min(profit\_scenarios.values())

is\_profitable = min\_profit > 0

logger.info(f"Score Cast calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Score Cast",

"profit\_scenarios": profit\_scenarios,

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": lay\_stakes,

"stake": stake,

"commission": commission,

"back\_commission": back\_commission

} @staticmethod

def calculate\_goal\_range(back\_odds\_dict, lay\_odds\_dict, stake, commission=0.02, back\_commission=0.0):

"""Calculate potential profit/loss for Goal Range market with commission.

This market allows betting on the total number of goals falling within specific ranges

(e.g., 0-1 goals, 2-3 goals, 4+ goals).

Parameters:

back\_odds\_dict (dict): Dictionary of range -> back odds (e.g. {'0-1': 3.5, '2-3': 2.7, '4+': 4.0})

lay\_odds\_dict (dict): Dictionary of range -> lay odds

stake (float): Stake amount for back bet

commission (float): Commission rate for lay bets (default 0.02 for Smarkets)

back\_commission (float): Commission rate for back bets (default 0.0 for Smarkets, 0.05 for Betfair)

Returns:

dict: Profit scenarios and other calculation details

Edge cases:

- Returns None if any odds are < 1.0

- Returns None if division by zero occurs during lay stake calculation

- Returns None if dictionaries have different keys

Example usage:

back\_odds = {'0-1': 3.5, '2-3': 2.2, '4+': 4.0}

lay\_odds = {'0-1': 3.7, '2-3': 2.3, '4+': 4.2}

result = calculate\_goal\_range(back\_odds, lay\_odds, 100, 0.02, 0.05)

"""

if not back\_odds\_dict or not lay\_odds\_dict:

logger.warning("Empty odds dictionaries in calculate\_goal\_range")

return None

if any(odds <= 0 for odds in back\_odds\_dict.values()) or any(odds <= 0 for odds in lay\_odds\_dict.values()):

logger.warning(f"Zero or negative odds in calculate\_goal\_range: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

if any(odds < 1.0 for odds in back\_odds\_dict.values()) or any(odds < 1.0 for odds in lay\_odds\_dict.values()):

logger.warning(f"Invalid odds in calculate\_goal\_range: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

# Check that both dictionaries have the same keys

if set(back\_odds\_dict.keys()) != set(lay\_odds\_dict.keys()):

logger.warning(f"Mismatched keys in calculate\_goal\_range: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

ranges = list(back\_odds\_dict.keys())

lay\_stakes = {}

# Calculate lay stakes for each range

for goal\_range in ranges:

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake, back\_odds\_dict[goal\_range], lay\_odds\_dict[goal\_range], commission)

if lay\_stake is None:

return None

lay\_stakes[goal\_range] = lay\_stake

# Calculate profit scenarios using helper method

profit\_scenarios = StrategyCalculators.calculate\_profit\_scenarios(

stake, back\_odds\_dict, lay\_odds\_dict, lay\_stakes, commission, back\_commission

)

min\_profit = min(profit\_scenarios.values())

is\_profitable = min\_profit > 0

logger.info(f"Goal Range calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Goal Range",

"profit\_scenarios": profit\_scenarios,

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": lay\_stakes,

"stake": stake,

"commission": commission,

"back\_commission": back\_commission

} # Only keep the version with back\_commission parameter and improved implementation

@staticmethod

def calculate\_time\_of\_first\_goal(back\_odds\_dict, lay\_odds\_dict, stake, commission=0.02, back\_commission=0.0):

"""Calculate potential profit/loss for Time of First Goal market with commission.

This market allows betting on the time bracket when the first goal will be scored.

Parameters:

back\_odds\_dict (dict): Dictionary of time bracket -> back odds (e.g. {'0-15': 4.0, '16-30': 4.5, ...})

lay\_odds\_dict (dict): Dictionary of time bracket -> lay odds

stake (float): Stake amount for back bet

commission (float): Commission rate for lay bets (default 0.02 for Smarkets)

back\_commission (float): Commission rate for back bets (default 0.0 for Smarkets, 0.05 for Betfair)

Returns:

dict: Profit scenarios and other calculation details

Edge cases:

- Returns None if any odds are < 1.0

- Returns None if division by zero occurs during lay stake calculation

- Returns None if dictionaries have different keys

Example usage:

back\_odds = {'0-15': 4.0, '16-30': 4.5, '31-45': 5.0, '46-60': 5.5, '61-75': 6.0, '76-90': 7.0, 'No Goal': 12.0}

lay\_odds = {'0-15': 4.2, '16-30': 4.7, '31-45': 5.2, '46-60': 5.7, '61-75': 6.2, '76-90': 7.3, 'No Goal': 13.0}

result = calculate\_time\_of\_first\_goal(back\_odds, lay\_odds, 100, 0.02, 0.05)

"""

if not back\_odds\_dict or not lay\_odds\_dict:

logger.warning("Empty odds dictionaries in calculate\_time\_of\_first\_goal")

return None

if any(odds <= 0 for odds in back\_odds\_dict.values()) or any(odds <= 0 for odds in lay\_odds\_dict.values()):

logger.warning(f"Zero or negative odds in calculate\_time\_of\_first\_goal: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

if any(odds < 1.0 for odds in back\_odds\_dict.values()) or any(odds < 1.0 for odds in lay\_odds\_dict.values()):

logger.warning(f"Invalid odds in calculate\_time\_of\_first\_goal: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

# Check that both dictionaries have the same keys

if set(back\_odds\_dict.keys()) != set(lay\_odds\_dict.keys()):

logger.warning(f"Mismatched keys in calculate\_time\_of\_first\_goal: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

time\_brackets = list(back\_odds\_dict.keys())

lay\_stakes = {}

# Calculate lay stakes for each time bracket

for bracket in time\_brackets:

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake, back\_odds\_dict[bracket], lay\_odds\_dict[bracket], commission)

if lay\_stake is None:

return None

lay\_stakes[bracket] = lay\_stake

# Calculate profit scenarios using helper method

profit\_scenarios = StrategyCalculators.calculate\_profit\_scenarios(

stake, back\_odds\_dict, lay\_odds\_dict, lay\_stakes, commission, back\_commission

)

min\_profit = min(profit\_scenarios.values())

is\_profitable = min\_profit > 0

logger.info(f"Time of First Goal calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Time of First Goal",

"profit\_scenarios": profit\_scenarios,

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": lay\_stakes,

"stake": stake,

"commission": commission,

"back\_commission": back\_commission

} @staticmethod

def calculate\_profit\_scenarios(stake, back\_odds\_dict, lay\_odds\_dict, lay\_stakes, commission, back\_commission):

"""Helper method to calculate profit scenarios for multi-outcome bets.

Parameters:

stake (float): Stake amount for back bet

back\_odds\_dict (dict): Dictionary of outcome -> back odds

lay\_odds\_dict (dict): Dictionary of outcome -> lay odds

lay\_stakes (dict): Dictionary of outcome -> lay stakes

commission (float): Commission rate for lay bets

back\_commission (float): Commission rate for back bets

Returns:

dict: Dictionary of outcome -> profit

Example usage:

profit\_scenarios = calculate\_profit\_scenarios(100, {'1-0': 7.0, '2-0': 9.0},

{'1-0': 7.5, '2-0': 9.5},

{'1-0': 14.3, '2-0': 12.5}, 0.02, 0.05)

"""

profit\_scenarios = {}

total\_lay\_liability = sum(lay\_stakes[outcome] \* (lay\_odds\_dict[outcome] - 1) \* (1 - commission) for outcome in lay\_stakes)

for target\_outcome in back\_odds\_dict:

target\_liability = lay\_stakes[target\_outcome] \* (lay\_odds\_dict[target\_outcome] - 1) \* (1 - commission)

profit = (stake \* (back\_odds\_dict[target\_outcome] - 1) \* (1 - back\_commission)) - (total\_lay\_liability - target\_liability)

profit\_scenarios[target\_outcome] = profit

return profit\_scenarios

def calculate\_odd\_even\_goals(back\_odd\_odds, lay\_odd\_odds, back\_even\_odds, lay\_even\_odds, stake, commission=0.02, back\_commission=0.0):

"""Calculate potential profit/loss for Odd/Even Goals market with commission.

This market allows betting on whether the total number of goals in a match will be odd or even.

Parameters:

back\_odd\_odds (float): Back odds for odd number of goals

lay\_odd\_odds (float): Lay odds for odd number of goals

back\_even\_odds (float): Back odds for even number of goals

lay\_even\_odds (float): Lay odds for even number of goals

stake (float): Stake amount for back bet

commission (float): Commission rate for lay bets (default 0.02 for Smarkets)

back\_commission (float): Commission rate for back bets (default 0.0 for Smarkets, 0.05 for Betfair)

Returns:

dict: Profit scenarios and other calculation details

Edge cases:

- Returns None if any odds are < 1.0

- Returns None if division by zero occurs during lay stake calculation

Example usage:

result = calculate\_odd\_even\_goals(1.9, 2.0, 2.0, 2.1, 100, 0.02, 0.05)

"""

if any(odds <= 0 for odds in [back\_odd\_odds, lay\_odd\_odds, back\_even\_odds, lay\_even\_odds]):

logger.warning(f"Zero or negative odds in calculate\_odd\_even\_goals: back\_odd\_odds={back\_odd\_odds}, lay\_odd\_odds={lay\_odd\_odds}, back\_even\_odds={back\_even\_odds}, lay\_even\_odds={lay\_even\_odds}")

return None

if any(odds < 1.0 for odds in [back\_odd\_odds, lay\_odd\_odds, back\_even\_odds, lay\_even\_odds]):

logger.warning(f"Invalid odds in calculate\_odd\_even\_goals: back\_odd\_odds={back\_odd\_odds}, lay\_odd\_odds={lay\_odd\_odds}, back\_even\_odds={back\_even\_odds}, lay\_even\_odds={lay\_even\_odds}")

return None

lay\_stake\_odd = StrategyCalculators.calculate\_lay\_stake(stake, back\_odd\_odds, lay\_odd\_odds, commission)

lay\_stake\_even = StrategyCalculators.calculate\_lay\_stake(stake, back\_even\_odds, lay\_even\_odds, commission)

if lay\_stake\_odd is None or lay\_stake\_even is None:

return None

profit\_odd = (stake \* (back\_odd\_odds - 1) \* (1 - back\_commission)) - (lay\_stake\_even \* (lay\_even\_odds - 1) \* (1 - commission))

profit\_even = (stake \* (back\_even\_odds - 1) \* (1 - back\_commission)) - (lay\_stake\_odd \* (lay\_odd\_odds - 1) \* (1 - commission))

min\_profit = min(profit\_odd, profit\_even)

is\_profitable = min\_profit > 0

logger.info(f"Odd/Even Goals calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Odd/Even Goals",

"profit\_scenarios": {"odd": profit\_odd, "even": profit\_even},

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": {"odd": lay\_stake\_odd, "even": lay\_stake\_even},

"stake": stake,

"commission": commission,

"back\_commission": back\_commission

} @staticmethod

def calculate\_method\_of\_first\_goal(back\_odds\_dict, lay\_odds\_dict, stake, commission=0.02, back\_commission=0.0):

"""Calculate potential profit/loss for Method of First Goal market with commission.

This market allows betting on how the first goal will be scored (e.g., header, shot, penalty).

Parameters:

back\_odds\_dict (dict): Dictionary of method -> back odds (e.g. {'header': 5.0, 'shot': 2.0, ...})

lay\_odds\_dict (dict): Dictionary of method -> lay odds

stake (float): Stake amount for back bet

commission (float): Commission rate for lay bets (default 0.02 for Smarkets)

back\_commission (float): Commission rate for back bets (default 0.0 for Smarkets, 0.05 for Betfair)

Returns:

dict: Profit scenarios and other calculation details

Edge cases:

- Returns None if any odds are < 1.0

- Returns None if division by zero occurs during lay stake calculation

- Returns None if dictionaries have different keys

Example usage:

back\_odds = {'header': 5.0, 'shot': 2.0, 'penalty': 8.0}

lay\_odds = {'header': 5.2, 'shot': 2.1, 'penalty': 8.5}

result = calculate\_method\_of\_first\_goal(back\_odds, lay\_odds, 100, 0.02, 0.05)

"""

if not back\_odds\_dict or not lay\_odds\_dict:

logger.warning("Empty odds dictionaries in calculate\_method\_of\_first\_goal")

return None

if any(odds <= 0 for odds in back\_odds\_dict.values()) or any(odds <= 0 for odds in lay\_odds\_dict.values()):

logger.warning(f"Zero or negative odds in calculate\_method\_of\_first\_goal: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

if any(odds < 1.0 for odds in back\_odds\_dict.values()) or any(odds < 1.0 for odds in lay\_odds\_dict.values()):

logger.warning(f"Invalid odds in calculate\_method\_of\_first\_goal: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

# Check that both dictionaries have the same keys

if set(back\_odds\_dict.keys()) != set(lay\_odds\_dict.keys()):

logger.warning(f"Mismatched keys in calculate\_method\_of\_first\_goal: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

methods = list(back\_odds\_dict.keys())

lay\_stakes = {}

# Calculate lay stakes for each method

for method in methods:

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake, back\_odds\_dict[method], lay\_odds\_dict[method], commission)

if lay\_stake is None:

return None

lay\_stakes[method] = lay\_stake

# Calculate profit scenarios using helper method

profit\_scenarios = StrategyCalculators.calculate\_profit\_scenarios(

stake, back\_odds\_dict, lay\_odds\_dict, lay\_stakes, commission, back\_commission

)

min\_profit = min(profit\_scenarios.values())

is\_profitable = min\_profit > 0

logger.info(f"Method of First Goal calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Method of First Goal",

"profit\_scenarios": profit\_scenarios,

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": lay\_stakes,

"stake": stake,

"commission": commission,

"back\_commission": back\_commission

} - commission) for method in methods)

# Calculate profit for each method outcome

for target\_method in methods:

target\_liability = lay\_stakes[target\_method] \* (lay\_odds\_dict[target\_method] - 1) \* (1import logging

logger = logging.getLogger(\_\_name\_\_)

class StrategyCalculators:

"""Provides methods to calculate profitable betting opportunities."""

@staticmethod

def calculate\_lay\_stake(stake, back\_odds, lay\_odds, commission):

"""Helper method to calculate lay stake with division by zero protection.

Parameters:

stake (float): Stake amount for back bet

back\_odds (float): Back odds

lay\_odds (float): Lay odds

commission (float): Commission rate

Returns:

float or None: Calculated lay stake or None if invalid

Example:

lay\_stake = calculate\_lay\_stake(100, 2.0, 2.1, 0.02)

"""

if back\_odds <= 0 or lay\_odds <= 0:

logger.warning(f"Zero or negative odds in calculate\_lay\_stake: back\_odds={back\_odds}, lay\_odds={lay\_odds}")

return None

denominator = lay\_odds \* (1 - commission) - 1

if denominator <= 0:

logger.warning(f"Division by zero in lay stake calculation: lay\_odds={lay\_odds}, commission={commission}")

return None

return (stake \* back\_odds) / denominator

def calculate\_arbitrage(back\_odds, lay\_odds, back\_platform, stake, commission=0.02):

"""Calculate arbitrage opportunity between back and lay odds."""

if back\_odds < 1.0 or lay\_odds < 1.0 or back\_platform not in ["betfair", "smarkets"]:

return None

lay\_stake = (stake \* back\_odds) / (lay\_odds \* (1 - commission) - 1)

profit\_if\_back\_wins = (stake \* (back\_odds - 1)) - (lay\_stake \* (lay\_odds - 1) \* (1 - commission))

profit\_if\_lay\_wins = stake - (lay\_stake \* (lay\_odds - 1) \* (1 - commission))

min\_profit = min(profit\_if\_back\_wins, profit\_if\_lay\_wins)

is\_profitable = min\_profit > 0

logger.info(f"{half\_name} {market\_type.capitalize()} calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": f"{half\_name} {market\_type.capitalize()}",

"profit\_scenarios": profit\_scenarios,

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": lay\_stakes,

"stake": stake,

"commission": commission

}"Team {market\_type.capitalize()} calculation for {team}: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": f"Team {market\_type.capitalize()} ({team})",

"profit\_scenarios": {"over": profit\_over, "under": profit\_under},

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": {"over": lay\_stake\_over, "under": lay\_stake\_under},

"stake": stake,

"commission": commission

}

@staticmethod

def calculate\_half\_markets(back\_odds\_dict, lay\_odds\_dict, stake, half="first", market\_type="goals", commission=0.02):

"""Calculate potential profit/loss for Half-specific markets with commission.

Parameters:

back\_odds\_dict (dict): Dictionary of outcome -> back odds

lay\_odds\_dict (dict): Dictionary of outcome -> lay odds

stake (float): Stake amount for back bet

half (str): "first" or "second" to specify which half

market\_type (str): Type of market (e.g., "goals", "corners", "cards")

commission (float): Commission rate (default 0.02)

Returns:

dict: Profit scenarios and other calculation details

"""

if not back\_odds\_dict or not lay\_odds\_dict:

return None

if any(odds < 1.0 for odds in back\_odds\_dict.values()) or any(odds < 1.0 for odds in lay\_odds\_dict.values()):

return None

# Check that both dictionaries have the same keys

if set(back\_odds\_dict.keys()) != set(lay\_odds\_dict.keys()):

return None

outcomes = list(back\_odds\_dict.keys())

lay\_stakes = {}

profit\_scenarios = {}

# Calculate lay stakes for each outcome

for outcome in outcomes:

lay\_stakes[outcome] = (stake \* back\_odds\_dict[outcome]) / (lay\_odds\_dict[outcome] \* (1 - commission) - 1)

# Calculate profit for each outcome

for target\_outcome in outcomes:

profit = (stake \* (back\_odds\_dict[target\_outcome] - 1))

# Subtract liabilities for all other outcomes

for other\_outcome in outcomes:

if other\_outcome != target\_outcome:

profit -= (lay\_stakes[other\_outcome] \* (lay\_odds\_dict[other\_outcome] - 1) \* (1 - commission))

profit\_scenarios[target\_outcome] = profit

min\_profit = min(profit\_scenarios.values())

is\_profitable = min\_profit > 0

half\_name = "First Half" if half == "first" else "Second Half"

logger.info(f"Odd/Even Goals calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Odd/Even Goals",

"profit\_scenarios": {"odd": profit\_odd, "even": profit\_even},

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": {"odd": lay\_stake\_odd, "even": lay\_stake\_even},

"stake": stake,

"commission": commission

}

@staticmethod

def calculate\_team\_corners\_cards(back\_over\_odds, lay\_over\_odds, back\_under\_odds, lay\_under\_odds, stake, market\_type="corners", team="home", commission=0.02):

"""Calculate potential profit/loss for Team Corners or Cards market with commission."""

if any(odds < 1.0 for odds in [back\_over\_odds, lay\_over\_odds, back\_under\_odds, lay\_under\_odds]):

return None

lay\_stake\_over = (stake \* back\_over\_odds) / (lay\_over\_odds \* (1 - commission) - 1)

lay\_stake\_under = (stake \* back\_under\_odds) / (lay\_under\_odds \* (1 - commission) - 1)

profit\_over = (stake \* (back\_over\_odds - 1)) - (lay\_stake\_under \* (lay\_under\_odds - 1) \* (1 - commission))

profit\_under = (stake \* (back\_under\_odds - 1)) - (lay\_stake\_over \* (lay\_over\_odds - 1) \* (1 - commission))

min\_profit = min(profit\_over, profit\_under)

is\_profitable = min\_profit > 0

logger.info(f"Method of First Goal calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Method of First Goal",

"profit\_scenarios": profit\_scenarios,

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": lay\_stakes,

"stake": stake,

"commission": commission

}

@staticmethod

def calculate\_odd\_even\_goals(back\_odd\_odds, lay\_odd\_odds, back\_even\_odds, lay\_even\_odds, stake, commission=0.02):

"""Calculate potential profit/loss for Odd/Even Goals market with commission."""

if any(odds < 1.0 for odds in [back\_odd\_odds, lay\_odd\_odds, back\_even\_odds, lay\_even\_odds]):

return None

lay\_stake\_odd = (stake \* back\_odd\_odds) / (lay\_odd\_odds \* (1 - commission) - 1)

lay\_stake\_even = (stake \* back\_even\_odds) / (lay\_even\_odds \* (1 - commission) - 1)

profit\_odd = (stake \* (back\_odd\_odds - 1)) - (lay\_stake\_even \* (lay\_even\_odds - 1) \* (1 - commission))

profit\_even = (stake \* (back\_even\_odds - 1)) - (lay\_stake\_odd \* (lay\_odd\_odds - 1) \* (1 - commission))

min\_profit = min(profit\_odd, profit\_even)

is\_profitable = min\_profit > 0

logger.info(f"Arbitrage calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Arbitrage",

"profit": min\_profit,

"lay\_stake": lay\_stake,

"is\_profitable": is\_profitable,

"stake": stake,

"commission": commission

}

@staticmethod

def calculate\_score\_cast(back\_odds\_dict, lay\_odds\_dict, stake, commission=0.02, back\_commission=0.0):

"""Calculate potential profit/loss for Score Cast market with commission.

This market combines first goalscorer with correct score for higher odds. It allows betting on

which player will score first and what the final score will be, as a single bet.

Parameters:

back\_odds\_dict (dict): Dictionary of scorer+score combo -> back odds (e.g. {'Kane 2-0': 25.0, 'Salah 1-0': 20.0, ...})

lay\_odds\_dict (dict): Dictionary of scorer+score combo -> lay odds

stake (float): Stake amount for back bet

commission (float): Commission rate for lay bets (default 0.02 for Smarkets)

back\_commission (float): Commission rate for back bets (default 0.0 for Smarkets, 0.05 for Betfair)

Returns:

dict: Profit scenarios and other calculation details

Edge cases:

- Returns None if any odds are < 1.0

- Returns None if division by zero occurs during lay stake calculation

- Returns None if dictionaries have different keys

Example usage:

back\_odds = {'Kane 2-0': 25.0, 'Kane 1-0': 20.0, 'Salah 2-0': 30.0}

lay\_odds = {'Kane 2-0': 26.0, 'Kane 1-0': 21.0, 'Salah 2-0': 32.0}

result = calculate\_score\_cast(back\_odds, lay\_odds, 100, 0.02, 0.05)

"""

if not back\_odds\_dict or not lay\_odds\_dict:

logger.warning("Empty odds dictionaries in calculate\_score\_cast")

return None

if any(odds <= 0 for odds in back\_odds\_dict.values()) or any(odds <= 0 for odds in lay\_odds\_dict.values()):

logger.warning(f"Zero or negative odds in calculate\_score\_cast: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

if any(odds < 1.0 for odds in back\_odds\_dict.values()) or any(odds < 1.0 for odds in lay\_odds\_dict.values()):

logger.warning(f"Invalid odds in calculate\_score\_cast: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

# Check that both dictionaries have the same keys

if set(back\_odds\_dict.keys()) != set(lay\_odds\_dict.keys()):

logger.warning(f"Mismatched keys in calculate\_score\_cast: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

combinations = list(back\_odds\_dict.keys())

lay\_stakes = {}

# Calculate lay stakes for each combination

for combo in combinations:

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake, back\_odds\_dict[combo], lay\_odds\_dict[combo], commission)

if lay\_stake is None:

return None

lay\_stakes[combo] = lay\_stake

# Calculate profit scenarios using helper method

profit\_scenarios = StrategyCalculators.calculate\_profit\_scenarios(

stake, back\_odds\_dict, lay\_odds\_dict, lay\_stakes, commission, back\_commission

)

min\_profit = min(profit\_scenarios.values())

is\_profitable = min\_profit > 0

logger.info(f"Score Cast calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Score Cast",

"profit\_scenarios": profit\_scenarios,

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": lay\_stakes,

"stake": stake,

"commission": commission,

"back\_commission": back\_commission

}0.02, back\_commission=0.0):

"""Calculate potential profit/loss for Score Cast market with commission.

This market combines first goalscorer with correct score for higher odds. It allows betting on

which player will score first and what the final score will be, as a single bet.

Parameters:

back\_odds\_dict (dict): Dictionary of scorer+score combo -> back odds (e.g. {'Kane 2-0': 25.0, 'Salah 1-0': 20.0, ...})

lay\_odds\_dict (dict): Dictionary of scorer+score combo -> lay odds

stake (float): Stake amount for back bet

commission (float): Commission rate for lay bets (default 0.02 for Smarkets)

back\_commission (float): Commission rate for back bets (default 0.0 for Smarkets, 0.05 for Betfair)

Returns:

dict: Profit scenarios and other calculation details

Edge cases:

- Returns None if any odds are < 1.0

- Returns None if division by zero occurs during lay stake calculation

- Returns None if dictionaries have different keys

Example usage:

back\_odds = {'Kane 2-0': 25.0, 'Kane 1-0': 20.0, 'Salah 2-0': 30.0}

lay\_odds = {'Kane 2-0': 26.0, 'Kane 1-0': 21.0, 'Salah 2-0': 32.0}

result = calculate\_score\_cast(back\_odds, lay\_odds, 100, 0.02, 0.05)

"""

if not back\_odds\_dict or not lay\_odds\_dict:

logger.warning("Empty odds dictionaries in calculate\_score\_cast")

return None

if any(odds <= 0 for odds in back\_odds\_dict.values()) or any(odds <= 0 for odds in lay\_odds\_dict.values()):

logger.warning(f"Zero or negative odds in calculate\_score\_cast: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

if any(odds < 1.0 for odds in back\_odds\_dict.values()) or any(odds < 1.0 for odds in lay\_odds\_dict.values()):

logger.warning(f"Invalid odds in calculate\_score\_cast: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

# Check that both dictionaries have the same keys

if set(back\_odds\_dict.keys()) != set(lay\_odds\_dict.keys()):

logger.warning(f"Mismatched keys in calculate\_score\_cast: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

combinations = list(back\_odds\_dict.keys())

lay\_stakes = {}

# Calculate lay stakes for each combination

for combo in combinations:

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake, back\_odds\_dict[combo], lay\_odds\_dict[combo], commission)

if lay\_stake is None:

return None

lay\_stakes[combo] = lay\_stake

# Calculate profit scenarios using helper method

profit\_scenarios = StrategyCalculators.calculate\_profit\_scenarios(

stake, back\_odds\_dict, lay\_odds\_dict, lay\_stakes, commission, back\_commission

)

min\_profit = min(profit\_scenarios.values())

is\_profitable = min\_profit > 0

logger.info(f"Score Cast calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Score Cast",

"profit\_scenarios": profit\_scenarios,

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": lay\_stakes,

"stake": stake,

"commission": commission,

"back\_commission": back\_commission

}0.02):

"""Calculate potential profit/loss for Score Cast market with commission.

Parameters:

back\_odds\_dict (dict): Dictionary of scorer+score combo -> back odds

lay\_odds\_dict (dict): Dictionary of scorer+score combo -> lay odds

stake (float): Stake amount for back bet

commission (float): Commission rate (default 0.02)

Returns:

dict: Profit scenarios and other calculation details

"""

if not back\_odds\_dict or not lay\_odds\_dict:

return None

if any(odds < 1.0 for odds in back\_odds\_dict.values()) or any(odds < 1.0 for odds in lay\_odds\_dict.values()):

return None

# Check that both dictionaries have the same keys

if set(back\_odds\_dict.keys()) != set(lay\_odds\_dict.keys()):

return None

combinations = list(back\_odds\_dict.keys())

lay\_stakes = {}

profit\_scenarios = {}

# Calculate lay stakes for each combination

for combo in combinations:

lay\_stakes[combo] = (stake \* back\_odds\_dict[combo]) / (lay\_odds\_dict[combo] \* (1 - commission) - 1)

# Calculate profit for each combination outcome

for target\_combo in combinations:

profit = (stake \* (back\_odds\_dict[target\_combo] - 1))

# Subtract liabilities for all other combinations

for other\_combo in combinations:

if other\_combo != target\_combo:

profit -= (lay\_stakes[other\_combo] \* (lay\_odds\_dict[other\_combo] - 1) \* (1 - commission))

profit\_scenarios[target\_combo] = profit

min\_profit = min(profit\_scenarios.values())

is\_profitable = min\_profit > 0

logger.info(f"Score Cast calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Score Cast",

"profit\_scenarios": profit\_scenarios,

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": lay\_stakes,

"stake": stake,

"commission": commission

}

@staticmethod

def calculate\_team\_to\_score\_first(back\_home\_odds, lay\_home\_odds, back\_away\_odds, lay\_away\_odds, back\_no\_goal\_odds, lay\_no\_goal\_odds, stake, commission=0.02, back\_commission=0.0):

"""Calculate potential profit/loss for Team To Score First market with commission.

This market allows betting on which team will score the first goal in the match, or whether

there will be no goals.

Parameters:

back\_home\_odds (float): Back odds for home team to score first

lay\_home\_odds (float): Lay odds for home team

back\_away\_odds (float): Back odds for away team to score first

lay\_away\_odds (float): Lay odds for away team

back\_no\_goal\_odds (float): Back odds for no goals in the match

lay\_no\_goal\_odds (float): Lay odds for no goals

stake (float): Stake amount for back bet

commission (float): Commission rate for lay bets (default 0.02 for Smarkets)

back\_commission (float): Commission rate for back bets (default 0.0 for Smarkets, 0.05 for Betfair)

Returns:

dict: Profit scenarios and other calculation details

Edge cases:

- Returns None if any odds are < 1.0

- Returns None if division by zero occurs during lay stake calculation

Example usage:

result = calculate\_team\_to\_score\_first(2.0, 2.1, 2.2, 2.3, 11.0, 12.0, 100, 0.02, 0.05)

"""

if any(odds <= 0 for odds in [back\_home\_odds, lay\_home\_odds, back\_away\_odds, lay\_away\_odds,

back\_no\_goal\_odds, lay\_no\_goal\_odds]):

logger.warning(f"Zero or negative odds in calculate\_team\_to\_score\_first: back\_home\_odds={back\_home\_odds}, lay\_home\_odds={lay\_home\_odds}, " +

f"back\_away\_odds={back\_away\_odds}, lay\_away\_odds={lay\_away\_odds}, " +

f"back\_no\_goal\_odds={back\_no\_goal\_odds}, lay\_no\_goal\_odds={lay\_no\_goal\_odds}")

return None

if any(odds < 1.0 for odds in [back\_home\_odds, lay\_home\_odds, back\_away\_odds, lay\_away\_odds,

back\_no\_goal\_odds, lay\_no\_goal\_odds]):

logger.warning(f"Invalid odds in calculate\_team\_to\_score\_first: back\_home\_odds={back\_home\_odds}, lay\_home\_odds={lay\_home\_odds}, " +

f"back\_away\_odds={back\_away\_odds}, lay\_away\_odds={lay\_away\_odds}, " +

f"back\_no\_goal\_odds={back\_no\_goal\_odds}, lay\_no\_goal\_odds={lay\_no\_goal\_odds}")

return None

# Calculate lay stakes using helper method

lay\_stake\_home = StrategyCalculators.calculate\_lay\_stake(stake, back\_home\_odds, lay\_home\_odds, commission)

lay\_stake\_away = StrategyCalculators.calculate\_lay\_stake(stake, back\_away\_odds, lay\_away\_odds, commission)

lay\_stake\_no\_goal = StrategyCalculators.calculate\_lay\_stake(stake, back\_no\_goal\_odds, lay\_no\_goal\_odds, commission)

if lay\_stake\_home is None or lay\_stake\_away is None or lay\_stake\_no\_goal is None:

return None

# Calculate profit for each outcome scenario

profit\_home = (stake \* (back\_home\_odds - 1) \* (1 - back\_commission)) - (lay\_stake\_away \* (lay\_away\_odds - 1) \* (1 - commission)) - (lay\_stake\_no\_goal \* (lay\_no\_goal\_odds - 1) \* (1 - commission))

profit\_away = (stake \* (back\_away\_odds - 1) \* (1 - back\_commission)) - (lay\_stake\_home \* (lay\_home\_odds - 1) \* (1 - commission)) - (lay\_stake\_no\_goal \* (lay\_no\_goal\_odds - 1) \* (1 - commission))

profit\_no\_goal = (stake \* (back\_no\_goal\_odds - 1) \* (1 - back\_commission)) - (lay\_stake\_home \* (lay\_home\_odds - 1) \* (1 - commission)) - (lay\_stake\_away \* (lay\_away\_odds - 1) \* (1 - commission))

min\_profit = min(profit\_home, profit\_away, profit\_no\_goal)

is\_profitable = min\_profit > 0

logger.info(f"Team To Score First calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Team To Score First",

"profit\_scenarios": {"home": profit\_home, "away": profit\_away, "no\_goal": profit\_no\_goal},

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": {"home": lay\_stake\_home, "away": lay\_stake\_away, "no\_goal": lay\_stake\_no\_goal},

"stake": stake,

"commission": commission,

"back\_commission": back\_commission

}

@staticmethod

def calculate\_race\_to\_goals(back\_home\_odds, lay\_home\_odds, back\_away\_odds, lay\_away\_odds, back\_neither\_odds, lay\_neither\_odds, stake, goals=2, commission=0.02, back\_commission=0.0):

"""Calculate potential profit/loss for Race to X Goals market with commission.

Parameters:

back\_home\_odds (float): Back odds for home team to reach goal target first

lay\_home\_odds (float): Lay odds for home team

back\_away\_odds (float): Back odds for away team to reach goal target first

lay\_away\_odds (float): Lay odds for away team

back\_neither\_odds (float): Back odds for neither team to reach goal target

lay\_neither\_odds (float): Lay odds for neither team

stake (float): Stake amount for back bet

goals (int): Target number of goals to reach (e.g., 2)

commission (float): Commission rate for lay bets (default 0.02)

back\_commission (float): Commission rate for back bets (default 0.0)

Returns:

dict: Profit scenarios and other calculation details

"""

# Validate goals parameter

if not isinstance(goals, int) or goals <= 0:

logger.warning(f"Invalid goals parameter in calculate\_race\_to\_goals: goals={goals}")

return None

if any(odds < 1.0 for odds in [back\_home\_odds, lay\_home\_odds, back\_away\_odds, lay\_away\_odds,

back\_neither\_odds, lay\_neither\_odds]):

logger.warning(f"Invalid odds in calculate\_race\_to\_goals: back\_home\_odds={back\_home\_odds}, lay\_home\_odds={lay\_home\_odds}, " +

f"back\_away\_odds={back\_away\_odds}, lay\_away\_odds={lay\_away\_odds}, " +

f"back\_neither\_odds={back\_neither\_odds}, lay\_neither\_odds={lay\_neither\_odds}")

return None

lay\_stake\_home = StrategyCalculators.calculate\_lay\_stake(stake, back\_home\_odds, lay\_home\_odds, commission)

lay\_stake\_away = StrategyCalculators.calculate\_lay\_stake(stake, back\_away\_odds, lay\_away\_odds, commission)

lay\_stake\_neither = StrategyCalculators.calculate\_lay\_stake(stake, back\_neither\_odds, lay\_neither\_odds, commission)

if lay\_stake\_home is None or lay\_stake\_away is None or lay\_stake\_neither is None:

return None

profit\_home = (stake \* (back\_home\_odds - 1) \* (1 - back\_commission)) - (lay\_stake\_away \* (lay\_away\_odds - 1) \* (1 - commission)) - (lay\_stake\_neither \* (lay\_neither\_odds - 1) \* (1 - commission))

profit\_away = (stake \* (back\_away\_odds - 1) \* (1 - back\_commission)) - (lay\_stake\_home \* (lay\_home\_odds - 1) \* (1 - commission)) - (lay\_stake\_neither \* (lay\_neither\_odds - 1) \* (1 - commission))

profit\_neither = (stake \* (back\_neither\_odds - 1) \* (1 - back\_commission)) - (lay\_stake\_home \* (lay\_home\_odds - 1) \* (1 - commission)) - (lay\_stake\_away \* (lay\_away\_odds - 1) \* (1 - commission))

min\_profit = min(profit\_home, profit\_away, profit\_neither)

is\_profitable = min\_profit > 0

logger.info(f"Race to {goals} Goals calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": f"Race to {goals} Goals",

"profit\_scenarios": {"home": profit\_home, "away": profit\_away, "neither": profit\_neither},

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": {"home": lay\_stake\_home, "away": lay\_stake\_away, "neither": lay\_stake\_neither},

"stake": stake,

"commission": commission,

"back\_commission": back\_commission

}

@staticmethod

def calculate\_method\_of\_first\_goal(back\_odds\_dict, lay\_odds\_dict, stake, commission=0.02):

"""Calculate potential profit/loss for Method of First Goal market with commission.

Parameters:

back\_odds\_dict (dict): Dictionary of method -> back odds (e.g. {'header': 5.0, 'shot': 2.0, ...})

lay\_odds\_dict (dict): Dictionary of method -> lay odds

stake (float): Stake amount for back bet

commission (float): Commission rate (default 0.02)

Returns:

dict: Profit scenarios and other calculation details

"""

if not back\_odds\_dict or not lay\_odds\_dict:

return None

if any(odds < 1.0 for odds in back\_odds\_dict.values()) or any(odds < 1.0 for odds in lay\_odds\_dict.values()):

return None

# Check that both dictionaries have the same keys

if set(back\_odds\_dict.keys()) != set(lay\_odds\_dict.keys()):

return None

methods = list(back\_odds\_dict.keys())

lay\_stakes = {}

profit\_scenarios = {}

# Calculate lay stakes for each method

for method in methods:

lay\_stakes[method] = (stake \* back\_odds\_dict[method]) / (lay\_odds\_dict[method] \* (1 - commission) - 1)

# Calculate profit for each method outcome

for target\_method in methods:

profit = (stake \* (back\_odds\_dict[target\_method] - 1))

# Subtract liabilities for all other methods

for other\_method in methods:

if other\_method != target\_method:

profit -= (lay\_stakes[other\_method] \* (lay\_odds\_dict[other\_method] - 1) \* (1 - commission))

profit\_scenarios[target\_method] = profit

min\_profit = min(profit\_scenarios.values())

is\_profitable = min\_profit > 0

logger.info(f

@staticmethod

def calculate\_handicap\_result(back\_home\_odds, lay\_home\_odds, back\_draw\_odds, lay\_draw\_odds, back\_away\_odds, lay\_away\_odds, stake, commission=0.02):

"""Calculate potential profit/loss for 3-Way Handicap Result market with commission."""

if any(odds < 1.0 for odds in [back\_home\_odds, lay\_home\_odds, back\_draw\_odds, lay\_draw\_odds, back\_away\_odds, lay\_away\_odds]):

return None

lay\_stake\_home = (stake \* back\_home\_odds) / (lay\_home\_odds \* (1 - commission) - 1)

lay\_stake\_draw = (stake \* back\_draw\_odds) / (lay\_draw\_odds \* (1 - commission) - 1)

lay\_stake\_away = (stake \* back\_away\_odds) / (lay\_away\_odds \* (1 - commission) - 1)

profit\_home = (stake \* (back\_home\_odds - 1)) - (lay\_stake\_draw \* (lay\_draw\_odds - 1) \* (1 - commission)) - (lay\_stake\_away \* (lay\_away\_odds - 1) \* (1 - commission))

profit\_draw = (stake \* (back\_draw\_odds - 1)) - (lay\_stake\_home \* (lay\_home\_odds - 1) \* (1 - commission)) - (lay\_stake\_away \* (lay\_away\_odds - 1) \* (1 - commission))

profit\_away = (stake \* (back\_away\_odds - 1)) - (lay\_stake\_home \* (lay\_home\_odds - 1) \* (1 - commission)) - (lay\_stake\_draw \* (lay\_draw\_odds - 1) \* (1 - commission))

min\_profit = min(profit\_home, profit\_draw, profit\_away)

is\_profitable = min\_profit > 0

logger.info(f"Handicap Result calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Handicap Result",

"profit\_scenarios": {"home": profit\_home, "draw": profit\_draw, "away": profit\_away},

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": {"home": lay\_stake\_home, "draw": lay\_stake\_draw, "away": lay\_stake\_away},

"stake": stake,

"commission": commission

}

@staticmethod

def calculate\_multi\_goals(back\_odds\_dict, lay\_odds\_dict, stake, commission=0.02, back\_commission=0.0):

"""Calculate potential profit/loss for Multi-Goals market with commission.

This market allows betting on multiple over/under goals outcomes combined, such as

"over 1.5 goals and under 3.5 goals" as a single bet.

Parameters:

back\_odds\_dict (dict): Dictionary of multi-goal outcome -> back odds (e.g. {'1.5-2.5': 3.0, '2.5-3.5': 2.5, ...})

lay\_odds\_dict (dict): Dictionary of multi-goal outcome -> lay odds

stake (float): Stake amount for back bet

commission (float): Commission rate for lay bets (default 0.02 for Smarkets)

back\_commission (float): Commission rate for back bets (default 0.0 for Smarkets, 0.05 for Betfair)

Returns:

dict: Profit scenarios and other calculation details

Edge cases:

- Returns None if any odds are < 1.0

- Returns None if division by zero occurs during lay stake calculation

- Returns None if dictionaries have different keys

Example usage:

back\_odds = {'1.5-2.5': 3.0, '2.5-3.5': 2.5, '3.5-4.5': 4.0}

lay\_odds = {'1.5-2.5': 3.2, '2.5-3.5': 2.7, '3.5-4.5': 4.3}

result = calculate\_multi\_goals(back\_odds, lay\_odds, 100, 0.02, 0.05)

"""

if not back\_odds\_dict or not lay\_odds\_dict:

logger.warning("Empty odds dictionaries in calculate\_multi\_goals")

return None

if any(odds <= 0 for odds in back\_odds\_dict.values()) or any(odds <= 0 for odds in lay\_odds\_dict.values()):

logger.warning(f"Zero or negative odds in calculate\_multi\_goals: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

if any(odds < 1.0 for odds in back\_odds\_dict.values()) or any(odds < 1.0 for odds in lay\_odds\_dict.values()):

logger.warning(f"Invalid odds in calculate\_multi\_goals: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

# Check that both dictionaries have the same keys

if set(back\_odds\_dict.keys()) != set(lay\_odds\_dict.keys()):

logger.warning(f"Mismatched keys in calculate\_multi\_goals: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

outcomes = list(back\_odds\_dict.keys())

lay\_stakes = {}

# Calculate lay stakes for each outcome

for outcome in outcomes:

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake, back\_odds\_dict[outcome], lay\_odds\_dict[outcome], commission)

if lay\_stake is None:

return None

lay\_stakes[outcome] = lay\_stake

# Calculate profit scenarios using helper method

profit\_scenarios = StrategyCalculators.calculate\_profit\_scenarios(

stake, back\_odds\_dict, lay\_odds\_dict, lay\_stakes, commission, back\_commission

)

min\_profit = min(profit\_scenarios.values())

is\_profitable = min\_profit > 0

logger.info(f"Multi-Goals calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Multi-Goals",

"profit\_scenarios": profit\_scenarios,

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": lay\_stakes,

"stake": stake,

"commission": commission,

"back\_commission": back\_commission

}

@staticmethod

def calculate\_team\_goals(back\_over\_odds, lay\_over\_odds, back\_under\_odds, lay\_under\_odds, stake, team="home", commission=0.02, back\_commission=0.0):

"""Calculate potential profit/loss for Team Goals market with commission.

This market involves betting on whether a specific team will score over or under a certain number of goals.

This method is a wrapper around calculate\_team\_market with market\_type="goals".

Parameters:

back\_over\_odds (float): Back odds for over

lay\_over\_odds (float): Lay odds for over

back\_under\_odds (float): Back odds for under

lay\_under\_odds (float): Lay odds for under

stake (float): Stake amount for back bet

team (str): Which team ("home" or "away")

commission (float): Commission rate for lay bets (default 0.02 for Smarkets)

back\_commission (float): Commission rate for back bets (default 0.0 for Smarkets, 0.05 for Betfair)

Returns:

dict: Profit scenarios and other calculation details

Example usage:

result = calculate\_team\_goals(2.0, 2.1, 1.9, 2.0, 100, "home", 0.02, 0.05)

"""

return StrategyCalculators.calculate\_team\_market(

back\_over\_odds,

lay\_over\_odds,

back\_under\_odds,

lay\_under\_odds,

stake,

market\_type="goals",

team=team,

commission=commission,

back\_commission=back\_commission

)

@staticmethod

def calculate\_winning\_margin(back\_odds\_dict, lay\_odds\_dict, stake, commission=0.02, back\_commission=0.0):

"""Calculate potential profit/loss for Winning Margin market with commission.

This market allows betting on the exact margin of victory for a team (e.g., win by 1 goal,

win by 2 goals, etc.).

Parameters:

back\_odds\_dict (dict): Dictionary of margin -> back odds (e.g. {'1': 4.0, '2': 6.0, ...})

lay\_odds\_dict (dict): Dictionary of margin -> lay odds

stake (float): Stake amount for back bet

commission (float): Commission rate for lay bets (default 0.02 for Smarkets)

back\_commission (float): Commission rate for back bets (default 0.0 for Smarkets, 0.05 for Betfair)

Returns:

dict: Profit scenarios and other calculation details

Edge cases:

- Returns None if any odds are < 1.0

- Returns None if division by zero occurs during lay stake calculation

- Returns None if dictionaries have different keys

Example usage:

back\_odds = {'1': 4.0, '2': 6.0, '3+': 9.0, 'Draw': 3.5, '-1': 4.5, '-2': 8.0, '-3+': 12.0}

lay\_odds = {'1': 4.2, '2': 6.3, '3+': 9.5, 'Draw': 3.7, '-1': 4.7, '-2': 8.4, '-3+': 13.0}

result = calculate\_winning\_margin(back\_odds, lay\_odds, 100, 0.02, 0.05)

"""

if not back\_odds\_dict or not lay\_odds\_dict:

logger.warning("Empty odds dictionaries in calculate\_winning\_margin")

return None

if any(odds <= 0 for odds in back\_odds\_dict.values()) or any(odds <= 0 for odds in lay\_odds\_dict.values()):

logger.warning(f"Zero or negative odds in calculate\_winning\_margin: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

if any(odds < 1.0 for odds in back\_odds\_dict.values()) or any(odds < 1.0 for odds in lay\_odds\_dict.values()):

logger.warning(f"Invalid odds in calculate\_winning\_margin: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

# Check that both dictionaries have the same keys

if set(back\_odds\_dict.keys()) != set(lay\_odds\_dict.keys()):

logger.warning(f"Mismatched keys in calculate\_winning\_margin: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

margins = list(back\_odds\_dict.keys())

lay\_stakes = {}

# Calculate lay stakes for each margin

for margin in margins:

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake, back\_odds\_dict[margin], lay\_odds\_dict[margin], commission)

if lay\_stake is None:

return None

lay\_stakes[margin] = lay\_stake

# Calculate profit scenarios using helper method

profit\_scenarios = StrategyCalculators.calculate\_profit\_scenarios(

stake, back\_odds\_dict, lay\_odds\_dict, lay\_stakes, commission, back\_commission

)

min\_profit = min(profit\_scenarios.values())

is\_profitable = min\_profit > 0

logger.info(f"Winning Margin calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Winning Margin",

"profit\_scenarios": profit\_scenarios,

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": lay\_stakes,

"stake": stake,

"commission": commission,

"back\_commission": back\_commission

}

@staticmethod

def calculate\_time\_of\_first\_goal(back\_odds\_dict, lay\_odds\_dict, stake, commission=0.02):

"""Calculate potential profit/loss for Time of First Goal market with commission.

Parameters:

back\_odds\_dict (dict): Dictionary of time bracket -> back odds (e.g. {'0-15': 4.0, '16-30': 4.5, ...})

lay\_odds\_dict (dict): Dictionary of time bracket -> lay odds

stake (float): Stake amount for back bet

commission (float): Commission rate (default 0.02)

Returns:

dict: Profit scenarios and other calculation details

"""

if not back\_odds\_dict or not lay\_odds\_dict:

return None

if any(odds < 1.0 for odds in back\_odds\_dict.values()) or any(odds < 1.0 for odds in lay\_odds\_dict.values()):

return None

# Check that both dictionaries have the same keys

if set(back\_odds\_dict.keys()) != set(lay\_odds\_dict.keys()):

return None

time\_brackets = list(back\_odds\_dict.keys())

lay\_stakes = {}

profit\_scenarios = {}

# Calculate lay stakes for each time bracket

for bracket in time\_brackets:

lay\_stakes[bracket] = (stake \* back\_odds\_dict[bracket]) / (lay\_odds\_dict[bracket] \* (1 - commission) - 1)

# Calculate profit for each time bracket outcome

for target\_bracket in time\_brackets:

profit = (stake \* (back\_odds\_dict[target\_bracket] - 1))

# Subtract liabilities for all other time brackets

for other\_bracket in time\_brackets:

if other\_bracket != target\_bracket:

profit -= (lay\_stakes[other\_bracket] \* (lay\_odds\_dict[other\_bracket] - 1) \* (1 - commission))

profit\_scenarios[target\_bracket] = profit

min\_profit = min(profit\_scenarios.values())

is\_profitable = min\_profit > 0

logger.info(f"Time of First Goal calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Time of First Goal",

"profit\_scenarios": profit\_scenarios,

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": lay\_stakes,

"stake": stake,

"commission": commission

}

@staticmethod

def calculate\_outright(back\_odds\_dict, lay\_odds\_dict, stake, commission=0.02):

"""Calculate potential profit/loss for Outright market with commission.

Parameters:

back\_odds\_dict (dict): Dictionary of team/player -> back odds

lay\_odds\_dict (dict): Dictionary of team/player -> lay odds

stake (float): Stake amount for back bet

commission (float): Commission rate (default 0.02)

Returns:

dict: Profit scenarios and other calculation details

"""

if not back\_odds\_dict or not lay\_odds\_dict:

return None

if any(odds < 1.0 for odds in back\_odds\_dict.values()) or any(odds < 1.0 for odds in lay\_odds\_dict.values()):

return None

# Check that both dictionaries have the same keys

if set(back\_odds\_dict.keys()) != set(lay\_odds\_dict.keys()):

return None

selections = list(back\_odds\_dict.keys())

lay\_stakes = {}

profit\_scenarios = {}

# Calculate lay stakes for each selection

for selection in selections:

lay\_stakes[selection] = (stake \* back\_odds\_dict[selection]) / (lay\_odds\_dict[selection] \* (1 - commission) - 1)

# Calculate profit for each selection outcome

for target\_selection in selections:

profit = (stake \* (back\_odds\_dict[target\_selection] - 1))

# Subtract liabilities for all other selections

for other\_selection in selections:

if other\_selection != target\_selection:

profit -= (lay\_stakes[other\_selection] \* (lay\_odds\_dict[other\_selection] - 1) \* (1 - commission))

profit\_scenarios[target\_selection] = profit

min\_profit = min(profit\_scenarios.values())

is\_profitable = min\_profit > 0

logger.info(f"Outright calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Outright",

"profit\_scenarios": profit\_scenarios,

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": lay\_stakes,

"stake": stake,

"commission": commission

}

@staticmethod

def calculate\_match\_specials(back\_odds\_dict, lay\_odds\_dict, stake, commission=0.02, back\_commission=0.0):

"""Calculate potential profit/loss for Match Specials market with commission.

This market allows betting on special combinations of outcomes in a match, such as

"team wins and both teams score" or "team wins to nil".

Parameters:

back\_odds\_dict (dict): Dictionary of special -> back odds (e.g. {'Win & BTTS': 5.0, 'Win to nil': 3.2, ...})

lay\_odds\_dict (dict): Dictionary of special -> lay odds

stake (float): Stake amount for back bet

commission (float): Commission rate for lay bets (default 0.02 for Smarkets)

back\_commission (float): Commission rate for back bets (default 0.0 for Smarkets, 0.05 for Betfair)

Returns:

dict: Profit scenarios and other calculation details

Edge cases:

- Returns None if any odds are < 1.0

- Returns None if division by zero occurs during lay stake calculation

- Returns None if dictionaries have different keys

Example usage:

back\_odds = {'Win & BTTS': 5.0, 'Win to nil': 3.2, 'Win & Over 2.5': 4.0}

lay\_odds = {'Win & BTTS': 5.5, 'Win to nil': 3.5, 'Win & Over 2.5': 4.3}

result = calculate\_match\_specials(back\_odds, lay\_odds, 100, 0.02, 0.05)

"""

if not back\_odds\_dict or not lay\_odds\_dict:

logger.warning("Empty odds dictionaries in calculate\_match\_specials")

return None

if any(odds <= 0 for odds in back\_odds\_dict.values()) or any(odds <= 0 for odds in lay\_odds\_dict.values()):

logger.warning(f"Zero or negative odds in calculate\_match\_specials: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

if any(odds < 1.0 for odds in back\_odds\_dict.values()) or any(odds < 1.0 for odds in lay\_odds\_dict.values()):

logger.warning(f"Invalid odds in calculate\_match\_specials: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

# Check that both dictionaries have the same keys

if set(back\_odds\_dict.keys()) != set(lay\_odds\_dict.keys()):

logger.warning(f"Mismatched keys in calculate\_match\_specials: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

specials = list(back\_odds\_dict.keys())

lay\_stakes = {}

# Calculate lay stakes for each special

for special in specials:

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake, back\_odds\_dict[special], lay\_odds\_dict[special], commission)

if lay\_stake is None:

return None

lay\_stakes[special] = lay\_stake

# Calculate profit scenarios using helper method

profit\_scenarios = StrategyCalculators.calculate\_profit\_scenarios(

stake, back\_odds\_dict, lay\_odds\_dict, lay\_stakes, commission, back\_commission

)

min\_profit = min(profit\_scenarios.values())

is\_profitable = min\_profit > 0

logger.info(f"Match Specials calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Match Specials",

"profit\_scenarios": profit\_scenarios,

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": lay\_stakes,

"stake": stake,

"commission": commission,

"back\_commission": back\_commission

}

@staticmethod

def calculate\_goal\_range(back\_odds\_dict, lay\_odds\_dict, stake, commission=0.02):

"""Calculate potential profit/loss for Goal Range market with commission.

Parameters:

back\_odds\_dict (dict): Dictionary of range -> back odds (e.g. {'0-1': 3.5, '2-3': 2.7, ...})

lay\_odds\_dict (dict): Dictionary of range -> lay odds

stake (float): Stake amount for back bet

commission (float): Commission rate (default 0.02)

Returns:

dict: Profit scenarios and other calculation details

"""

if not back\_odds\_dict or not lay\_odds\_dict:

return None

if any(odds < 1.0 for odds in back\_odds\_dict.values()) or any(odds < 1.0 for odds in lay\_odds\_dict.values()):

return None

# Check that both dictionaries have the same keys

if set(back\_odds\_dict.keys()) != set(lay\_odds\_dict.keys()):

return None

ranges = list(back\_odds\_dict.keys())

lay\_stakes = {}

profit\_scenarios = {}

# Calculate lay stakes for each range

for goal\_range in ranges:

lay\_stakes[goal\_range] = (stake \* back\_odds\_dict[goal\_range]) / (lay\_odds\_dict[goal\_range] \* (1 - commission) - 1)

# Calculate profit for each range outcome

for target\_range in ranges:

profit = (stake \* (back\_odds\_dict[target\_range] - 1))

# Subtract liabilities for all other ranges

for other\_range in ranges:

if other\_range != target\_range:

profit -= (lay\_stakes[other\_range] \* (lay\_odds\_dict[other\_range] - 1) \* (1 - commission))

profit\_scenarios[target\_range] = profit

min\_profit = min(profit\_scenarios.values())

is\_profitable = min\_profit > 0

logger.info(f"Goal Range calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Goal Range",

"profit\_scenarios": profit\_scenarios,

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": lay\_stakes,

"stake": stake,

"commission": commission

}

@staticmethod

def calculate\_lay\_the\_draw(lay\_odds, back\_odds, lay\_stake, commission=0.02, back\_commission=0.0):

"""Calculate lay-the-draw strategy profit/loss.

This strategy involves laying the draw and backing a team if they go behind,

to secure a profit regardless of the final outcome.

Parameters:

lay\_odds (float): Lay odds for the draw

back\_odds (float): Back odds for the team

lay\_stake (float): Stake amount for lay bet

commission (float): Commission rate for lay bets (default 0.02 for Smarkets)

back\_commission (float): Commission rate for back bets (default 0.0 for Smarkets, 0.05 for Betfair)

Returns:

dict: Strategy calculation details including profit scenarios

Edge cases:

- Returns None if any odds are < 1.0

- Returns None if division by zero occurs during back stake calculation

Example usage:

result = calculate\_lay\_the\_draw(3.5, 1.5, 100, 0.02, 0.05)

"""

if lay\_odds <= 0 or back\_odds <= 0:

logger.warning(f"Zero or negative odds in calculate\_lay\_the\_draw: lay\_odds={lay\_odds}, back\_odds={back\_odds}")

return None

if lay\_odds < 1.0 or back\_odds < 1.0:

logger.warning(f"Invalid odds in calculate\_lay\_the\_draw: lay\_odds={lay\_odds}, back\_odds={back\_odds}")

return None

# Prevent division by zero

if back\_odds - 1 <= 0:

logger.warning(f"Division by zero in calculate\_lay\_the\_draw: back\_odds={back\_odds}")

return None

back\_stake = (lay\_stake \* (lay\_odds - 1) \* (1 - commission)) / (back\_odds - 1)

profit\_if\_not\_draw = (back\_stake \* (back\_odds - 1) \* (1 - back\_commission)) - (lay\_stake \* (lay\_odds - 1) \* (1 - commission))

profit\_if\_draw = -lay\_stake \* (lay\_odds - 1) \* (1 - commission)

min\_profit = min(profit\_if\_not\_draw, profit\_if\_draw)

is\_profitable = min\_profit > 0

logger.info(f"Lay-the-Draw calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Lay-the-Draw",

"profit\_scenarios": {"not\_draw": profit\_if\_not\_draw, "draw": profit\_if\_draw},

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"back\_stake": back\_stake,

"lay\_stake": lay\_stake,

"commission": commission,

"back\_commission": back\_commission

}

@staticmethod

def calculate\_1x2(back\_odds\_1, lay\_odds\_1, back\_odds\_x, lay\_odds\_x, back\_odds\_2, lay\_odds\_2, stake, commission=0.02, back\_commission=0.0):

"""Calculate potential profit/loss for a 1X2 market with commission."""

if any(odds < 1.0 for odds in [back\_odds\_1, lay\_odds\_1, back\_odds\_x, lay\_odds\_x, back\_odds\_2, lay\_odds\_2]):

return None

# Prevent division by zero

if (lay\_odds\_1 \* (1 - commission) - 1 <= 0 or

lay\_odds\_x \* (1 - commission) - 1 <= 0 or

lay\_odds\_2 \* (1 - commission) - 1 <= 0):

return None

lay\_stake\_1 = (stake \* back\_odds\_1) / (lay\_odds\_1 \* (1 - commission) - 1)

lay\_stake\_x = (stake \* back\_odds\_x) / (lay\_odds\_x \* (1 - commission) - 1)

lay\_stake\_2 = (stake \* back\_odds\_2) / (lay\_odds\_2 \* (1 - commission) - 1)

profit\_1 = (stake \* (back\_odds\_1 - 1) \* (1 - back\_commission)) - (lay\_stake\_x \* (lay\_odds\_x - 1) \* (1 - commission)) - (lay\_stake\_2 \* (lay\_odds\_2 - 1) \* (1 - commission))

profit\_x = (stake \* (back\_odds\_x - 1) \* (1 - back\_commission)) - (lay\_stake\_1 \* (lay\_odds\_1 - 1) \* (1 - commission)) - (lay\_stake\_2 \* (lay\_odds\_2 - 1) \* (1 - commission))

profit\_2 = (stake \* (back\_odds\_2 - 1) \* (1 - back\_commission)) - (lay\_stake\_1 \* (lay\_odds\_1 - 1) \* (1 - commission)) - (lay\_stake\_x \* (lay\_odds\_x - 1) \* (1 - commission))

min\_profit = min(profit\_1, profit\_x, profit\_2)

is\_profitable = min\_profit > 0

logger.info(f"1X2 calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "1X2",

"profit\_scenarios": {"home\_win": profit\_1, "draw": profit\_x, "away\_win": profit\_2},

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": {"home": lay\_stake\_1, "draw": lay\_stake\_x, "away": lay\_stake\_2},

"stake": stake,

"commission": commission,

"back\_commission": back\_commission

}

@staticmethod

def calculate\_over\_under(back\_over\_odds, lay\_over\_odds, back\_under\_odds, lay\_under\_odds, stake, commission=0.02, back\_commission=0.0):

"""Calculate potential profit/loss for Over/Under goals market with commission."""

if any(odds < 1.0 for odds in [back\_over\_odds, lay\_over\_odds, back\_under\_odds, lay\_under\_odds]):

return None

# Prevent division by zero

if (lay\_over\_odds \* (1 - commission) - 1 <= 0 or

lay\_under\_odds \* (1 - commission) - 1 <= 0):

return None

lay\_stake\_over = (stake \* back\_over\_odds) / (lay\_over\_odds \* (1 - commission) - 1)

lay\_stake\_under = (stake \* back\_under\_odds) / (lay\_under\_odds \* (1 - commission) - 1)

profit\_over = (stake \* (back\_over\_odds - 1) \* (1 - back\_commission)) - (lay\_stake\_under \* (lay\_under\_odds - 1) \* (1 - commission))

profit\_under = (stake \* (back\_under\_odds - 1) \* (1 - back\_commission)) - (lay\_stake\_over \* (lay\_over\_odds - 1) \* (1 - commission))

min\_profit = min(profit\_over, profit\_under)

is\_profitable = min\_profit > 0

logger.info(f"Over/Under calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Over/Under",

"profit\_scenarios": {"over": profit\_over, "under": profit\_under},

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": {"over": lay\_stake\_over, "under": lay\_stake\_under},

"stake": stake,

"commission": commission,

"back\_commission": back\_commission

}

@staticmethod

def calculate\_btts(back\_yes\_odds, lay\_yes\_odds, back\_no\_odds, lay\_no\_odds, stake, commission=0.02, back\_commission=0.0):

"""Calculate potential profit/loss for Both Teams To Score market with commission."""

if any(odds < 1.0 for odds in [back\_yes\_odds, lay\_yes\_odds, back\_no\_odds, lay\_no\_odds]):

return None

# Prevent division by zero

if (lay\_yes\_odds \* (1 - commission) - 1 <= 0 or

lay\_no\_odds \* (1 - commission) - 1 <= 0):

return None

lay\_stake\_yes = (stake \* back\_yes\_odds) / (lay\_yes\_odds \* (1 - commission) - 1)

lay\_stake\_no = (stake \* back\_no\_odds) / (lay\_no\_odds \* (1 - commission) - 1)

profit\_yes = (stake \* (back\_yes\_odds - 1) \* (1 - back\_commission)) - (lay\_stake\_no \* (lay\_no\_odds - 1) \* (1 - commission))

profit\_no = (stake \* (back\_no\_odds - 1) \* (1 - back\_commission)) - (lay\_stake\_yes \* (lay\_yes\_odds - 1) \* (1 - commission))

min\_profit = min(profit\_yes, profit\_no)

is\_profitable = min\_profit > 0

logger.info(f"BTTS calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "BTTS",

"profit\_scenarios": {"yes": profit\_yes, "no": profit\_no},

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": {"yes": lay\_stake\_yes, "no": lay\_stake\_no},

"stake": stake,

"commission": commission,

"back\_commission": back\_commission

}

@staticmethod

def calculate\_double\_chance(back\_1x\_odds, lay\_1x\_odds, back\_x2\_odds, lay\_x2\_odds, back\_12\_odds, lay\_12\_odds, stake, commission=0.02, back\_commission=0.0):

"""Calculate potential profit/loss for Double Chance market with commission.

This market allows betting on two of the three possible outcomes in a match (home win or draw,

draw or away win, home win or away win).

Parameters:

back\_1x\_odds (float): Back odds for home win or draw

lay\_1x\_odds (float): Lay odds for home win or draw

back\_x2\_odds (float): Back odds for draw or away win

lay\_x2\_odds (float): Lay odds for draw or away win

back\_12\_odds (float): Back odds for home win or away win

lay\_12\_odds (float): Lay odds for home win or away win

stake (float): Stake amount for back bet

commission (float): Commission rate for lay bets (default 0.02 for Smarkets)

back\_commission (float): Commission rate for back bets (default 0.0 for Smarkets, 0.05 for Betfair)

Returns:

dict: Profit scenarios and other calculation details

Edge cases:

- Returns None if any odds are < 1.0

- Returns None if division by zero occurs during lay stake calculation

Example usage:

result = calculate\_double\_chance(1.5, 1.55, 1.7, 1.75, 1.3, 1.35, 100, 0.02, 0.05)

"""

if any(odds <= 0 for odds in [back\_1x\_odds, lay\_1x\_odds, back\_x2\_odds, lay\_x2\_odds, back\_12\_odds, lay\_12\_odds]):

logger.warning(f"Zero or negative odds in calculate\_double\_chance: back\_1x\_odds={back\_1x\_odds}, lay\_1x\_odds={lay\_1x\_odds}, " +

f"back\_x2\_odds={back\_x2\_odds}, lay\_x2\_odds={lay\_x2\_odds}, " +

f"back\_12\_odds={back\_12\_odds}, lay\_12\_odds={lay\_12\_odds}")

return None

if any(odds < 1.0 for odds in [back\_1x\_odds, lay\_1x\_odds, back\_x2\_odds, lay\_x2\_odds, back\_12\_odds, lay\_12\_odds]):

logger.warning(f"Invalid odds in calculate\_double\_chance: back\_1x\_odds={back\_1x\_odds}, lay\_1x\_odds={lay\_1x\_odds}, " +

f"back\_x2\_odds={back\_x2\_odds}, lay\_x2\_odds={lay\_x2\_odds}, " +

f"back\_12\_odds={back\_12\_odds}, lay\_12\_odds={lay\_12\_odds}")

return None

# Calculate lay stakes using helper method

lay\_stake\_1x = StrategyCalculators.calculate\_lay\_stake(stake, back\_1x\_odds, lay\_1x\_odds, commission)

lay\_stake\_x2 = StrategyCalculators.calculate\_lay\_stake(stake, back\_x2\_odds, lay\_x2\_odds, commission)

lay\_stake\_12 = StrategyCalculators.calculate\_lay\_stake(stake, back\_12\_odds, lay\_12\_odds, commission)

if lay\_stake\_1x is None or lay\_stake\_x2 is None or lay\_stake\_12 is None:

return None

# Calculate profit for each outcome scenario

profit\_1x = (stake \* (back\_1x\_odds - 1) \* (1 - back\_commission)) - (lay\_stake\_x2 \* (lay\_x2\_odds - 1) \* (1 - commission)) - (lay\_stake\_12 \* (lay\_12\_odds - 1) \* (1 - commission))

profit\_x2 = (stake \* (back\_x2\_odds - 1) \* (1 - back\_commission)) - (lay\_stake\_1x \* (lay\_1x\_odds - 1) \* (1 - commission)) - (lay\_stake\_12 \* (lay\_12\_odds - 1) \* (1 - commission))

profit\_12 = (stake \* (back\_12\_odds - 1) \* (1 - back\_commission)) - (lay\_stake\_1x \* (lay\_1x\_odds - 1) \* (1 - commission)) - (lay\_stake\_x2 \* (lay\_x2\_odds - 1) \* (1 - commission))

min\_profit = min(profit\_1x, profit\_x2, profit\_12)

is\_profitable = min\_profit > 0

logger.info(f"Double Chance calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Double Chance",

"profit\_scenarios": {"home\_draw": profit\_1x, "draw\_away": profit\_x2, "home\_away": profit\_12},

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": {"home\_draw": lay\_stake\_1x, "draw\_away": lay\_stake\_x2, "home\_away": lay\_stake\_12},

"stake": stake,

"commission": commission,

"back\_commission": back\_commission

}

@staticmethod

def calculate\_correct\_score(back\_odds\_dict, lay\_odds\_dict, stake, commission=0.02, back\_commission=0.0):

"""Calculate potential profit/loss for Correct Score market with commission.

Parameters:

back\_odds\_dict (dict): Dictionary of score -> back odds (e.g. {'1-0': 7.0, '2-1': 9.0, ...})

lay\_odds\_dict (dict): Dictionary of score -> lay odds

stake (float): Stake amount for back bet

commission (float): Commission rate (default 0.02)

back\_commission (float): Commission rate for back bets (default 0.0)

Returns:

dict: Profit scenarios and other calculation details

"""

if not back\_odds\_dict or not lay\_odds\_dict:

return None

if any(odds < 1.0 for odds in back\_odds\_dict.values()) or any(odds < 1.0 for odds in lay\_odds\_dict.values()):

return None

# Check that both dictionaries have the same keys

if set(back\_odds\_dict.keys()) != set(lay\_odds\_dict.keys()):

return None

scores = list(back\_odds\_dict.keys())

lay\_stakes = {}

profit\_scenarios = {}

# Check for division by zero in lay odds

for score in scores:

if lay\_odds\_dict[score] \* (1 - commission) - 1 <= 0:

return None

# Calculate lay stakes for each score

for score in scores:

lay\_stakes[score] = (stake \* back\_odds\_dict[score]) / (lay\_odds\_dict[score] \* (1 - commission) - 1)

# Calculate profit for each score outcome

for target\_score in scores:

profit = (stake \* (back\_odds\_dict[target\_score] - 1) \* (1 - back\_commission))

# Subtract liabilities for all other scores

for other\_score in scores:

if other\_score != target\_score:

profit -= (lay\_stakes[other\_score] \* (lay\_odds\_dict[other\_score] - 1) \* (1 - commission))

profit\_scenarios[target\_score] = profit

min\_profit = min(profit\_scenarios.values())

is\_profitable = min\_profit > 0

logger.info(f"Correct Score calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Correct Score",

"profit\_scenarios": profit\_scenarios,

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": lay\_stakes,

"stake": stake,

"commission": commission,

"back\_commission": back\_commission

}

@staticmethod

def calculate\_halftime\_fulltime(back\_odds\_dict, lay\_odds\_dict, stake, commission=0.02, back\_commission=0.0):

"""Calculate potential profit/loss for Half Time/Full Time market with commission.

This market allows betting on the result at half time combined with the result at full time,

with 9 possible outcomes (e.g., Home/Home, Home/Draw, Home/Away, etc.).

Parameters:

back\_odds\_dict (dict): Dictionary of HT-FT result -> back odds (e.g. {'1-1': 4.5, 'X-1': 5.5, ...})

lay\_odds\_dict (dict): Dictionary of HT-FT result -> lay odds

stake (float): Stake amount for back bet

commission (float): Commission rate for lay bets (default 0.02 for Smarkets)

back\_commission (float): Commission rate for back bets (default 0.0 for Smarkets, 0.05 for Betfair)

Returns:

dict: Profit scenarios and other calculation details

Edge cases:

- Returns None if any odds are < 1.0

- Returns None if division by zero occurs during lay stake calculation

- Returns None if dictionaries don't have all 9 expected combinations

Example usage:

back\_odds = {'1-1': 4.5, '1-X': 15.0, '1-2': 30.0, 'X-1': 6.0, 'X-X': 5.0, 'X-2': 6.5, '2-1': 25.0, '2-X': 15.0, '2-2': 3.5}

lay\_odds = {'1-1': 4.7, '1-X': 16.0, '1-2': 32.0, 'X-1': 6.3, 'X-X': 5.2, 'X-2': 6.8, '2-1': 26.0, '2-X': 16.0, '2-2': 3.7}

result = calculate\_halftime\_fulltime(back\_odds, lay\_odds, 100, 0.02, 0.05)

"""

if not back\_odds\_dict or not lay\_odds\_dict:

logger.warning("Empty odds dictionaries in calculate\_halftime\_fulltime")

return None

if any(odds <= 0 for odds in back\_odds\_dict.values()) or any(odds <= 0 for odds in lay\_odds\_dict.values()):

logger.warning(f"Zero or negative odds in calculate\_halftime\_fulltime: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

if any(odds < 1.0 for odds in back\_odds\_dict.values()) or any(odds < 1.0 for odds in lay\_odds\_dict.values()):

logger.warning(f"Invalid odds in calculate\_halftime\_fulltime: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

# Check that both dictionaries have all 9 combinations

expected\_keys = ['1-1', '1-X', '1-2', 'X-1', 'X-X', 'X-2', '2-1', '2-X', '2-2']

if not all(key in back\_odds\_dict for key in expected\_keys) or not all(key in lay\_odds\_dict for key in expected\_keys):

logger.warning(f"Missing keys in calculate\_halftime\_fulltime: back\_odds\_dict keys={back\_odds\_dict.keys()}, lay\_odds\_dict keys={lay\_odds\_dict.keys()}")

return None

lay\_stakes = {}

# Calculate lay stakes for each HT-FT combination

for result in expected\_keys:

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake, back\_odds\_dict[result], lay\_odds\_dict[result], commission)

if lay\_stake is None:

return None

lay\_stakes[result] = lay\_stake

# Calculate profit scenarios using helper method

profit\_scenarios = StrategyCalculators.calculate\_profit\_scenarios(

stake, back\_odds\_dict, lay\_odds\_dict, lay\_stakes, commission, back\_commission

)

min\_profit = min(profit\_scenarios.values())

is\_profitable = min\_profit > 0

logger.info(f"Half Time/Full Time calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Half Time/Full Time",

"profit\_scenarios": profit\_scenarios,

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": lay\_stakes,

"stake": stake,

"commission": commission,

"back\_commission": back\_commission

}

@staticmethod

def calculate\_asian\_handicap(back\_odds\_dict, lay\_odds\_dict, stake, commission=0.02, back\_commission=0.0):

"""Calculate potential profit/loss for Asian Handicap market with commission.

This market involves one team starting with a handicap (e.g., -1.5, +0.5). A team with a negative

handicap must win by more than that margin, while a team with a positive handicap can lose by less

than that margin and still win the bet.

Parameters:

back\_odds\_dict (dict): Dictionary of handicap -> back odds (e.g. {'-1.5': 3.2, '+1.5': 1.4})

lay\_odds\_dict (dict): Dictionary of handicap -> lay odds

stake (float): Stake amount for back bet

commission (float): Commission rate for lay bets (default 0.02 for Smarkets)

back\_commission (float): Commission rate for back bets (default 0.0 for Smarkets, 0.05 for Betfair)

Returns:

dict: Profit scenarios and other calculation details

Edge cases:

- Returns None if any odds are < 1.0

- Returns None if division by zero occurs during lay stake calculation

- Returns None if dictionaries have different keys

Example usage:

back\_odds = {'-1.5': 3.2, '+0.5': 1.8, '+1.5': 1.4}

lay\_odds = {'-1.5': 3.4, '+0.5': 1.9, '+1.5': 1.5}

result = calculate\_asian\_handicap(back\_odds, lay\_odds, 100, 0.02, 0.05)

"""

if not back\_odds\_dict or not lay\_odds\_dict:

logger.warning("Empty odds dictionaries in calculate\_asian\_handicap")

return None

if any(odds <= 0 for odds in back\_odds\_dict.values()) or any(odds <= 0 for odds in lay\_odds\_dict.values()):

logger.warning(f"Zero or negative odds in calculate\_asian\_handicap: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

if any(odds < 1.0 for odds in back\_odds\_dict.values()) or any(odds < 1.0 for odds in lay\_odds\_dict.values()):

logger.warning(f"Invalid odds in calculate\_asian\_handicap: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

# Check that both dictionaries have the same keys

if set(back\_odds\_dict.keys()) != set(lay\_odds\_dict.keys()):

logger.warning(f"Mismatched keys in calculate\_asian\_handicap: back\_odds\_dict={back\_odds\_dict}, lay\_odds\_dict={lay\_odds\_dict}")

return None

handicaps = list(back\_odds\_dict.keys())

lay\_stakes = {}

# Calculate lay stakes for each handicap

for handicap in handicaps:

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake, back\_odds\_dict[handicap], lay\_odds\_dict[handicap], commission)

if lay\_stake is None:

return None

lay\_stakes[handicap] = lay\_stake

# Calculate profit scenarios using helper method

profit\_scenarios = StrategyCalculators.calculate\_profit\_scenarios(

stake, back\_odds\_dict, lay\_odds\_dict, lay\_stakes, commission, back\_commission

)

min\_profit = min(profit\_scenarios.values())

is\_profitable = min\_profit > 0

logger.info(f"Asian Handicap calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Asian Handicap",

"profit\_scenarios": profit\_scenarios,

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": lay\_stakes,

"stake": stake,

"commission": commission,

"back\_commission": back\_commission

}

@staticmethod

def calculate\_draw\_no\_bet(back\_home\_odds, lay\_home\_odds, back\_away\_odds, lay\_away\_odds, stake, commission=0.02, back\_commission=0.0):

"""Calculate potential profit/loss for Draw No Bet market with commission."""

if any(odds < 1.0 for odds in [back\_home\_odds, lay\_home\_odds, back\_away\_odds, lay\_away\_odds]):

return None

# Prevent division by zero

if (lay\_home\_odds \* (1 - commission) - 1 <= 0 or

lay\_away\_odds \* (1 - commission) - 1 <= 0):

return None

lay\_stake\_home = (stake \* back\_home\_odds) / (lay\_home\_odds \* (1 - commission) - 1)

lay\_stake\_away = (stake \* back\_away\_odds) / (lay\_away\_odds \* (1 - commission) - 1)

profit\_home = (stake \* (back\_home\_odds - 1) \* (1 - back\_commission)) - (lay\_stake\_away \* (lay\_away\_odds - 1) \* (1 - commission))

profit\_away = (stake \* (back\_away\_odds - 1) \* (1 - back\_commission)) - (lay\_stake\_home \* (lay\_home\_odds - 1) \* (1 - commission))

profit\_draw = 0 # Draw means stake is returned for both back and lay

min\_profit = min(profit\_home, profit\_away, profit\_draw)

is\_profitable = min\_profit > 0

logger.info(f"Draw No Bet calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Draw No Bet",

"profit\_scenarios": {"home": profit\_home, "away": profit\_away, "draw": profit\_draw},

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": {"home": lay\_stake\_home, "away": lay\_stake\_away},

"stake": stake,

"commission": commission,

"back\_commission": back\_commission

}

@staticmethod

def calculate\_goalscorer(back\_odds\_dict, lay\_odds\_dict, stake, commission=0.02):

"""Calculate potential profit/loss for First/Anytime Goalscorer market with commission.

Parameters:

back\_odds\_dict (dict): Dictionary of player -> back odds (e.g. {'Kane': 5.0, 'Salah': 4.5, ...})

lay\_odds\_dict (dict): Dictionary of player -> lay odds

stake (float): Stake amount for back bet

commission (float): Commission rate (default 0.02)

Returns:

dict: Profit scenarios and other calculation details

"""

if not back\_odds\_dict or not lay\_odds\_dict:

return None

if any(odds < 1.0 for odds in back\_odds\_dict.values()) or any(odds < 1.0 for odds in lay\_odds\_dict.values()):

return None

# Check that both dictionaries have the same keys

if set(back\_odds\_dict.keys()) != set(lay\_odds\_dict.keys()):

return None

players = list(back\_odds\_dict.keys())

lay\_stakes = {}

profit\_scenarios = {}

# Calculate lay stakes for each player

for player in players:

lay\_stakes[player] = (stake \* back\_odds\_dict[player]) / (lay\_odds\_dict[player] \* (1 - commission) - 1)

# Calculate profit for each player outcome

for target\_player in players:

profit = (stake \* (back\_odds\_dict[target\_player] - 1))

# Subtract liabilities for all other players

for other\_player in players:

if other\_player != target\_player:

profit -= (lay\_stakes[other\_player] \* (lay\_odds\_dict[other\_player] - 1) \* (1 - commission))

profit\_scenarios[target\_player] = profit

min\_profit = min(profit\_scenarios.values())

is\_profitable = min\_profit > 0

logger.info(f"Goalscorer calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Goalscorer",

"profit\_scenarios": profit\_scenarios,

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": lay\_stakes,

"stake": stake,

"commission": commission

}

@staticmethod

def calculate\_total\_corners(back\_over\_odds, lay\_over\_odds, back\_under\_odds, lay\_under\_odds, stake, commission=0.02):

"""Calculate potential profit/loss for Total Corners market with commission."""

if any(odds < 1.0 for odds in [back\_over\_odds, lay\_over\_odds, back\_under\_odds, lay\_under\_odds]):

return None

lay\_stake\_over = (stake \* back\_over\_odds) / (lay\_over\_odds \* (1 - commission) - 1)

lay\_stake\_under = (stake \* back\_under\_odds) / (lay\_under\_odds \* (1 - commission) - 1)

profit\_over = (stake \* (back\_over\_odds - 1)) - (lay\_stake\_under \* (lay\_under\_odds - 1) \* (1 - commission))

profit\_under = (stake \* (back\_under\_odds - 1)) - (lay\_stake\_over \* (lay\_over\_odds - 1) \* (1 - commission))

min\_profit = min(profit\_over, profit\_under)

is\_profitable = min\_profit > 0

logger.info(f"Total Corners calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Total Corners",

"profit\_scenarios": {"over": profit\_over, "under": profit\_under},

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": {"over": lay\_stake\_over, "under": lay\_stake\_under},

"stake": stake,

"commission": commission

}

@staticmethod

def calculate\_total\_cards(back\_over\_odds, lay\_over\_odds, back\_under\_odds, lay\_under\_odds, stake, commission=0.02):

"""Calculate potential profit/loss for Total Cards market with commission."""

if any(odds < 1.0 for odds in [back\_over\_odds, lay\_over\_odds, back\_under\_odds, lay\_under\_odds]):

return None

lay\_stake\_over = (stake \* back\_over\_odds) / (lay\_over\_odds \* (1 - commission) - 1)

lay\_stake\_under = (stake \* back\_under\_odds) / (lay\_under\_odds \* (1 - commission) - 1)

profit\_over = (stake \* (back\_over\_odds - 1)) - (lay\_stake\_under \* (lay\_under\_odds - 1) \* (1 - commission))

profit\_under = (stake \* (back\_under\_odds - 1)) - (lay\_stake\_over \* (lay\_over\_odds - 1) \* (1 - commission))

min\_profit = min(profit\_over, profit\_under)

is\_profitable = min\_profit > 0

logger.info(f"Total Cards calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Total Cards",

"profit\_scenarios": {"over": profit\_over, "under": profit\_under},

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": {"over": lay\_stake\_over, "under": lay\_stake\_under},

"stake": stake,

"commission": commission

}

@staticmethod

def calculate\_player\_props(back\_odds\_dict, lay\_odds\_dict, stake, commission=0.02):

"""Calculate potential profit/loss for Player Props market with commission.

Parameters:

back\_odds\_dict (dict): Dictionary of prop -> back odds (e.g. {'2+ shots': 1.8, '1+ assist': 3.2, ...})

lay\_odds\_dict (dict): Dictionary of prop -> lay odds

stake (float): Stake amount for back bet

commission (float): Commission rate (default 0.02)

Returns:

dict: Profit scenarios and other calculation details

"""

if not back\_odds\_dict or not lay\_odds\_dict:

return None

if any(odds < 1.0 for odds in back\_odds\_dict.values()) or any(odds < 1.0 for odds in lay\_odds\_dict.values()):

return None

# Check that both dictionaries have the same keys

if set(back\_odds\_dict.keys()) != set(lay\_odds\_dict.keys()):

return None

props = list(back\_odds\_dict.keys())

lay\_stakes = {}

profit\_scenarios = {}

# Calculate lay stakes for each prop

for prop in props:

lay\_stakes[prop] = (stake \* back\_odds\_dict[prop]) / (lay\_odds\_dict[prop] \* (1 - commission) - 1)

# Calculate profit for each prop outcome

for target\_prop in props:

profit = (stake \* (back\_odds\_dict[target\_prop] - 1))

# Subtract liabilities for all other props

for other\_prop in props:

if other\_prop != target\_prop:

profit -= (lay\_stakes[other\_prop] \* (lay\_odds\_dict[other\_prop] - 1) \* (1 - commission))

profit\_scenarios[target\_prop] = profit

min\_profit = min(profit\_scenarios.values())

is\_profitable = min\_profit > 0

logger.info(f"Player Props calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Player Props",

"profit\_scenarios": profit\_scenarios,

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": lay\_stakes,

"stake": stake,

"commission": commission

}

@staticmethod

def calculate\_win\_either\_half(back\_home\_odds, lay\_home\_odds, back\_away\_odds, lay\_away\_odds, stake, commission=0.02):

"""Calculate potential profit/loss for Win Either Half market with commission."""

if any(odds < 1.0 for odds in [back\_home\_odds, lay\_home\_odds, back\_away\_odds, lay\_away\_odds]):

return None

lay\_stake\_home = (stake \* back\_home\_odds) / (lay\_home\_odds \* (1 - commission) - 1)

lay\_stake\_away = (stake \* back\_away\_odds) / (lay\_away\_odds \* (1 - commission) - 1)

profit\_home = (stake \* (back\_home\_odds - 1)) - (lay\_stake\_away \* (lay\_away\_odds - 1) \* (1 - commission))

profit\_away = (stake \* (back\_away\_odds - 1)) - (lay\_stake\_home \* (lay\_home\_odds - 1) \* (1 - commission))

profit\_neither = -stake - (lay\_stake\_home \* (lay\_home\_odds - 1) \* (1 - commission)) - (lay\_stake\_away \* (lay\_away\_odds - 1) \* (1 - commission))

min\_profit = min(profit\_home, profit\_away, profit\_neither)

is\_profitable = min\_profit > 0

logger.info(f"Win Either Half calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Win Either Half",

"profit\_scenarios": {"home": profit\_home, "away": profit\_away, "neither": profit\_neither},

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": {"home": lay\_stake\_home, "away": lay\_stake\_away},

"stake": stake,

"commission": commission

}

@staticmethod

def calculate\_clean\_sheet(back\_yes\_odds, lay\_yes\_odds, back\_no\_odds, lay\_no\_odds, stake, commission=0.02):

"""Calculate potential profit/loss for Clean Sheet market with commission."""

if any(odds < 1.0 for odds in [back\_yes\_odds, lay\_yes\_odds, back\_no\_odds, lay\_no\_odds]):

return None

lay\_stake\_yes = (stake \* back\_yes\_odds) / (lay\_yes\_odds \* (1 - commission) - 1)

lay\_stake\_no = (stake \* back\_no\_odds) / (lay\_no\_odds \* (1 - commission) - 1)

profit\_yes = (stake \* (back\_yes\_odds - 1)) - (lay\_stake\_no \* (lay\_no\_odds - 1) \* (1 - commission))

profit\_no = (stake \* (back\_no\_odds - 1)) - (lay\_stake\_yes \* (lay\_yes\_odds - 1) \* (1 - commission))

min\_profit = min(profit\_yes, profit\_no)

is\_profitable = min\_profit > 0

logger.info(f"Clean Sheet calculation: min\_profit={min\_profit:.2f}, is\_profitable={is\_profitable}")

return {

"type": "Clean Sheet",

"profit\_scenarios": {"yes": profit\_yes, "no": profit\_no},

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": {"yes": lay\_stake\_yes, "no": lay\_stake\_no},

"stake": stake,

"commission": commission

}