# strategy\_calculators.py

# Enhanced version with methods for all football bet types

import logging

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

class StrategyCalculators:

"""

Provides methods to calculate profitable betting opportunities

across various football betting markets using back and lay odds

from betting exchanges.

"""

@staticmethod

def calculate\_1x2(back\_odds\_1, lay\_odds\_1, back\_odds\_x, lay\_odds\_x, back\_odds\_2, lay\_odds\_2, stake):

"""

Calculate potential profit/loss for a 1X2 (Match Result) market.

Args:

back\_odds\_1 (float): Back odds for home win

lay\_odds\_1 (float): Lay odds for home win

back\_odds\_x (float): Back odds for draw

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

back\_odds\_2 (float): Back odds for away win

lay\_odds\_2 (float): Lay odds for away win

stake (float): Initial stake amount

Returns:

dict: Dictionary containing profit scenarios and strategy details

"""

# Calculate lay stakes to balance potential profits across all outcomes

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

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

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

# Calculate potential profit for each outcome

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

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

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

# Determine the minimum profit (or maximum loss)

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

}

@staticmethod

def calculate\_over\_under(back\_over\_odds, lay\_over\_odds, back\_under\_odds, lay\_under\_odds, stake):

"""

Calculate potential profit/loss for an Over/Under Goals market.

Args:

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

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

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

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

stake (float): Initial stake amount

Returns:

dict: Dictionary containing profit scenarios and strategy details

"""

# Calculate lay stakes

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

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

# Calculate potential profit for each outcome

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

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

# Determine the minimum profit (or maximum loss)

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 Goals",

"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

}

@staticmethod

def calculate\_btts(back\_yes\_odds, lay\_yes\_odds, back\_no\_odds, lay\_no\_odds, stake):

"""

Calculate potential profit/loss for a Both Teams To Score market.

Args:

back\_yes\_odds (float): Back odds for both teams to score

lay\_yes\_odds (float): Lay odds for both teams to score

back\_no\_odds (float): Back odds for at least one team not to score

lay\_no\_odds (float): Lay odds for at least one team not to score

stake (float): Initial stake amount

Returns:

dict: Dictionary containing profit scenarios and strategy details

"""

# Calculate lay stakes

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

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

# Calculate potential profit for each outcome

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

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

# Determine the minimum profit (or maximum loss)

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": "Both Teams To Score",

"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

}

@staticmethod

def calculate\_double\_chance(back\_1x\_odds, lay\_1x\_odds, back\_x2\_odds, lay\_x2\_odds, back\_12\_odds, lay\_12\_odds, stake):

"""

Calculate potential profit/loss for a Double Chance market.

Args:

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): Initial stake amount

Returns:

dict: Dictionary containing profit scenarios and strategy details

"""

# Calculate lay stakes

lay\_stake\_1x = (stake \* back\_1x\_odds) / (lay\_1x\_odds - 1)

lay\_stake\_x2 = (stake \* back\_x2\_odds) / (lay\_x2\_odds - 1)

lay\_stake\_12 = (stake \* back\_12\_odds) / (lay\_12\_odds - 1)

# Calculate potential profit for each outcome

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

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

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

# Determine the minimum profit (or maximum loss)

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\_or\_draw": profit\_1x,

"draw\_or\_away": profit\_x2,

"home\_or\_away": profit\_12

},

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": {

"home\_or\_draw": lay\_stake\_1x,

"draw\_or\_away": lay\_stake\_x2,

"home\_or\_away": lay\_stake\_12

},

"stake": stake

}

@staticmethod

def calculate\_correct\_score(scores\_odds\_dict, stake):

"""

Calculate potential profit/loss for a Correct Score market.

Args:

scores\_odds\_dict (dict): Dictionary of score outcomes with their back and lay odds

Format: {'1-0': {'back': 8.0, 'lay': 8.5}, '2-1': {'back': 10.0, 'lay': 10.5}, ...}

stake (float): Initial stake amount

Returns:

dict: Dictionary containing profit scenarios and strategy details

"""

# Calculate lay stakes for each possible score

lay\_stakes = {}

for score, odds in scores\_odds\_dict.items():

back\_odds = odds['back']

lay\_odds = odds['lay']

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

# Calculate potential profit for each score outcome

profit\_scenarios = {}

for target\_score, odds in scores\_odds\_dict.items():

back\_odds = odds['back']

# Calculate profit if this score happens

profit = (stake \* back\_odds) - stake

# Subtract all lay stakes except for the target score

for score, lay\_stake in lay\_stakes.items():

if score != target\_score:

profit -= lay\_stake

profit\_scenarios[target\_score] = profit

# Determine the minimum profit (or maximum loss)

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

}

@staticmethod

def calculate\_ht\_ft(ht\_ft\_odds\_dict, stake):

"""

Calculate potential profit/loss for a Half-Time/Full-Time market.

Args:

ht\_ft\_odds\_dict (dict): Dictionary of HT/FT outcomes with their back and lay odds

Format: {'1/1': {'back': 3.0, 'lay': 3.2}, 'X/X': {'back': 5.0, 'lay': 5.5}, ...}

stake (float): Initial stake amount

Returns:

dict: Dictionary containing profit scenarios and strategy details

"""

# Calculate lay stakes for each possible HT/FT outcome

lay\_stakes = {}

for outcome, odds in ht\_ft\_odds\_dict.items():

back\_odds = odds['back']

lay\_odds = odds['lay']

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

# Calculate potential profit for each HT/FT outcome

profit\_scenarios = {}

for target\_outcome, odds in ht\_ft\_odds\_dict.items():

back\_odds = odds['back']

# Calculate profit if this outcome happens

profit = (stake \* back\_odds) - stake

# Subtract all lay stakes except for the target outcome

for outcome, lay\_stake in lay\_stakes.items():

if outcome != target\_outcome:

profit -= lay\_stake

profit\_scenarios[target\_outcome] = profit

# Determine the minimum profit (or maximum loss)

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

is\_profitable = min\_profit > 0

logger.info(f"HT/FT 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

}

@staticmethod

def calculate\_asian\_handicap(back\_odds, lay\_odds, handicap, stake):

"""

Calculate potential profit/loss for an Asian Handicap market.

Args:

back\_odds (dict): Dictionary of back odds for each handicap outcome

Format: {'home': 1.9, 'away': 2.0}

lay\_odds (dict): Dictionary of lay odds for each handicap outcome

Format: {'home': 2.0, 'away': 2.1}

handicap (float): The handicap value (e.g., -1.5, +0.5)

stake (float): Initial stake amount

Returns:

dict: Dictionary containing profit scenarios and strategy details

"""

# Calculate lay stakes

lay\_stake\_home = (stake \* back\_odds['home']) / (lay\_odds['home'] - 1)

lay\_stake\_away = (stake \* back\_odds['away']) / (lay\_odds['away'] - 1)

# Calculate potential profit for each outcome

profit\_home = (stake \* back\_odds['home']) - stake - lay\_stake\_away

profit\_away = (stake \* back\_odds['away']) - stake - lay\_stake\_home

# Determine the minimum profit (or maximum loss)

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

is\_profitable = min\_profit > 0

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

return {

"type": "Asian Handicap",

"handicap": handicap,

"profit\_scenarios": {

"home": profit\_home,

"away": profit\_away

},

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stakes": {

"home": lay\_stake\_home,

"away": lay\_stake\_away

},

"stake": stake

}

@staticmethod

def calculate\_dnb(back\_home\_odds, lay\_home\_odds, back\_away\_odds, lay\_away\_odds, draw\_odds, stake):

"""

Calculate potential profit/loss for a Draw No Bet market.

Args:

back\_home\_odds (float): Back odds for home win

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

back\_away\_odds (float): Back odds for away win

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

draw\_odds (float): The odds for a draw (used to calculate refund)

stake (float): Initial stake amount

Returns:

dict: Dictionary containing profit scenarios and strategy details

"""

# Calculate lay stakes

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

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

# Calculate potential profit for each outcome

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

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

profit\_draw = 0 # Stake is returned on draw

# Determine the minimum profit (or maximum loss)

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

is\_profitable = min\_profit > 0

logger.info(f"DNB 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

}

@staticmethod

def calculate\_goalscorer(back\_odds, lay\_odds, stake):

"""

Calculate potential profit/loss for a Goalscorer market (First or Anytime).

Args:

back\_odds (float): Back odds for the selected player to score

lay\_odds (float): Lay odds for the selected player to score

stake (float): Initial stake amount

Returns:

dict: Dictionary containing profit scenarios and strategy details

"""

# Calculate lay stake

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

# Calculate potential profit for each outcome

profit\_scores = (stake \* back\_odds) - stake - lay\_stake

profit\_doesnt\_score = stake - lay\_stake

# Determine the minimum profit (or maximum loss)

min\_profit = min(profit\_scores, profit\_doesnt\_score)

is\_profitable = min\_profit > 0

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

return {

"type": "Goalscorer",

"profit\_scenarios": {

"scores": profit\_scores,

"doesnt\_score": profit\_doesnt\_score

},

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stake": lay\_stake,

"stake": stake

}

@staticmethod

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

"""

Calculate potential profit/loss for a Total Corners market.

Args:

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

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

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

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

stake (float): Initial stake amount

Returns:

dict: Dictionary containing profit scenarios and strategy details

"""

# This is structurally identical to the over/under goals calculation

return StrategyCalculators.calculate\_over\_under(

back\_over\_odds, lay\_over\_odds, back\_under\_odds, lay\_under\_odds, stake

)

@staticmethod

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

"""

Calculate potential profit/loss for a Total Cards market.

Args:

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

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

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

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

stake (float): Initial stake amount

Returns:

dict: Dictionary containing profit scenarios and strategy details

"""

# This is structurally identical to the over/under goals calculation

return StrategyCalculators.calculate\_over\_under(

back\_over\_odds, lay\_over\_odds, back\_under\_odds, lay\_under\_odds, stake

)

@staticmethod

def calculate\_player\_prop(back\_odds, lay\_odds, stake):

"""

Calculate potential profit/loss for a Player Prop market.

Args:

back\_odds (float): Back odds for the player prop to occur

lay\_odds (float): Lay odds for the player prop to occur

stake (float): Initial stake amount

Returns:

dict: Dictionary containing profit scenarios and strategy details

"""

# Calculate lay stake

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

# Calculate potential profit for each outcome

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

profit\_no = stake - lay\_stake

# Determine the minimum profit (or maximum loss)

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

is\_profitable = min\_profit > 0

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

return {

"type": "Player Prop",

"profit\_scenarios": {

"yes": profit\_yes,

"no": profit\_no

},

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stake": lay\_stake,

"stake": stake

}

@staticmethod

def calculate\_win\_either\_half(back\_odds, lay\_odds, stake):

"""

Calculate potential profit/loss for a Win Either Half market.

Args:

back\_odds (float): Back odds for the team to win either half

lay\_odds (float): Lay odds for the team to win either half

stake (float): Initial stake amount

Returns:

dict: Dictionary containing profit scenarios and strategy details

"""

# Calculate lay stake

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

# Calculate potential profit for each outcome

profit\_wins\_half = (stake \* back\_odds) - stake - lay\_stake

profit\_doesnt\_win\_half = stake - lay\_stake

# Determine the minimum profit (or maximum loss)

min\_profit = min(profit\_wins\_half, profit\_doesnt\_win\_half)

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": {

"wins\_either\_half": profit\_wins\_half,

"doesnt\_win\_either\_half": profit\_doesnt\_win\_half

},

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stake": lay\_stake,

"stake": stake

}

@staticmethod

def calculate\_clean\_sheet(back\_odds, lay\_odds, stake):

"""

Calculate potential profit/loss for a Clean Sheet market.

Args:

back\_odds (float): Back odds for the team to keep a clean sheet

lay\_odds (float): Lay odds for the team to keep a clean sheet

stake (float): Initial stake amount

Returns:

dict: Dictionary containing profit scenarios and strategy details

"""

# Calculate lay stake

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

# Calculate potential profit for each outcome

profit\_clean\_sheet = (stake \* back\_odds) - stake - lay\_stake

profit\_no\_clean\_sheet = stake - lay\_stake

# Determine the minimum profit (or maximum loss)

min\_profit = min(profit\_clean\_sheet, profit\_no\_clean\_sheet)

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": {

"clean\_sheet": profit\_clean\_sheet,

"no\_clean\_sheet": profit\_no\_clean\_sheet

},

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stake": lay\_stake,

"stake": stake

}

@staticmethod

def calculate\_time\_first\_goal(time\_odds\_dict, stake):

"""

Calculate potential profit/loss for a Time of First Goal market.

Args:

time\_odds\_dict (dict): Dictionary of time brackets with their back and lay odds

Format: {'0-15': {'back': 3.5, 'lay': 3.7}, '16-30': {'back': 4.0, 'lay': 4.2}, ...}

stake (float): Initial stake amount

Returns:

dict: Dictionary containing profit scenarios and strategy details

"""

# Calculate lay stakes for each time bracket

lay\_stakes = {}

for time\_bracket, odds in time\_odds\_dict.items():

back\_odds = odds['back']

lay\_odds = odds['lay']

lay\_stakes[time\_bracket] = (stake \* back\_odds) / (lay\_odds - 1)

# Calculate potential profit for each time bracket

profit\_scenarios = {}

for target\_bracket, odds in time\_odds\_dict.items():

back\_odds = odds['back']

# Calculate profit if first goal is in this time bracket

profit = (stake \* back\_odds) - stake

# Subtract all lay stakes except for the target bracket

for bracket, lay\_stake in lay\_stakes.items():

if bracket != target\_bracket:

profit -= lay\_stake

profit\_scenarios[target\_bracket] = profit

# Add the "No Goal" scenario if not already included

if "No Goal" not in time\_odds\_dict:

# Calculate profit if no goal is scored

profit\_no\_goal = -stake

for lay\_stake in lay\_stakes.values():

profit\_no\_goal -= lay\_stake

profit\_scenarios["No Goal"] = profit\_no\_goal

# Determine the minimum profit (or maximum loss)

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

}

@staticmethod

def calculate\_outright(back\_odds, lay\_odds, stake):

"""

Calculate potential profit/loss for an Outright market.

Args:

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

lay\_odds (float): Lay odds for the team to win the tournament

stake (float): Initial stake amount

Returns:

dict: Dictionary containing profit scenarios and strategy details

"""

# Calculate lay stake

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

# Calculate potential profit for each outcome

profit\_wins = (stake \* back\_odds) - stake - lay\_stake

profit\_doesnt\_win = stake - lay\_stake

# Determine the minimum profit (or maximum loss)

min\_profit = min(profit\_wins, profit\_doesnt\_win)

is\_profitable = min\_profit > 0

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

return {

"type": "Outright",

"profit\_scenarios": {

"wins": profit\_wins,

"doesnt\_win": profit\_doesnt\_win

},

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stake": lay\_stake,

"stake": stake

}

@staticmethod

def calculate\_match\_special(back\_odds, lay\_odds, stake):

"""

Calculate potential profit/loss for a Match Special market.

Args:

back\_odds (float): Back odds for the special outcome

lay\_odds (float): Lay odds for the special outcome

stake (float): Initial stake amount

Returns:

dict: Dictionary containing profit scenarios and strategy details

"""

# Calculate lay stake

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

# Calculate potential profit for each outcome

profit\_special = (stake \* back\_odds) - stake - lay\_stake

profit\_not\_special = stake - lay\_stake

# Determine the minimum profit (or maximum loss)

min\_profit = min(profit\_special, profit\_not\_special)

is\_profitable = min\_profit > 0

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

return {

"type": "Match Special",

"profit\_scenarios": {

"special\_occurs": profit\_special,

"special\_doesnt\_occur": profit\_not\_special

},

"min\_profit": min\_profit,

"is\_profitable": is\_profitable,

"lay\_stake": lay\_stake,

"stake": stake

}

@staticmethod

def calculate\_goal\_range(range\_odds\_dict, stake):

"""

Calculate potential profit/loss for a Goal Range market.

Args:

range\_odds\_dict (dict): Dictionary of goal ranges with their back and lay odds

Format: {'0-1': {'back': 3.0, 'lay': 3.2}, '2-3': {'back': 2.5, 'lay': 2.7}, ...}

stake (float): Initial stake amount

Returns:

dict: Dictionary containing profit scenarios and strategy details

"""

# Calculate lay stakes for each goal range

lay\_stakes = {}

for goal\_range, odds in range\_odds\_dict.items():

back\_odds = odds['back']

lay\_odds = odds['lay']

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

# Calculate potential profit for each goal range

profit\_scenarios = {}

for target\_range, odds in range\_odds\_dict.items():

back\_odds = odds['back']

# Calculate profit if goals are in this range

profit = (stake \* back\_odds) - stake

# Subtract all lay stakes except for the target range

for range\_key, lay\_stake in lay\_stakes.items():

if range\_key != target\_range:

profit -= lay\_stake

profit\_scenarios[target\_range] = profit

# Determine the minimum profit (or maximum loss)

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,