import logging

import json

from decimal import Decimal, ROUND\_HALF\_UP

from typing import Dict, Union, Optional, List, Any

from functools import lru\_cache

# Configure logging

logging.basicConfig(

level=logging.INFO,

format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'

)

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

# Custom exceptions

class BettingCalculationError(Exception):

pass

class InvalidInputError(BettingCalculationError):

pass

class DivisionByZeroError(BettingCalculationError):

pass

class UnsupportedMarketError(BettingCalculationError):

pass

class StrategyCalculators:

@staticmethod

def validate\_inputs(inputs: Dict[str, Any], required\_keys: List[str], type\_checks: Optional[Dict[str, tuple]] = None) -> None:

for key in required\_keys:

if key not in inputs:

logger.warning(f"Missing required parameter: {key}")

raise InvalidInputError(f"Missing required parameter: {key}")

if type\_checks:

for key, expected\_types in type\_checks.items():

if key in inputs and not isinstance(inputs[key], expected\_types):

logger.warning(f"Invalid type for {key}. Expected {expected\_types}, got {type(inputs[key])}")

raise InvalidInputError(f"Invalid type for {key}. Expected {expected\_types}, got {type(inputs[key])}")

for key, value in inputs.items():

if isinstance(value, (int, float)):

inputs[key] = Decimal(str(value)).quantize(Decimal('0.01'), rounding=ROUND\_HALF\_UP)

if key in ['commission', 'back\_commission'] and inputs[key] >= 1:

logger.warning(f"Commission {inputs[key]} >= 1 is invalid")

raise InvalidInputError("Commission must be between 0 and 1")

@staticmethod

@lru\_cache(maxsize=1000)

def calculate\_lay\_stake(stake: Union[int, float, Decimal], back\_odds: Union[int, float, Decimal], lay\_odds: Union[int, float, Decimal], commission: Union[int, float, Decimal] = Decimal('0.02')) -> Decimal:

inputs = {'stake': stake, 'back\_odds': back\_odds, 'lay\_odds': lay\_odds, 'commission': commission}

StrategyCalculators.validate\_inputs(inputs, ['stake', 'back\_odds', 'lay\_odds', 'commission'], {k: (int, float, Decimal) for k in inputs.keys()})

stake\_d, back\_odds\_d, lay\_odds\_d, commission\_d = (inputs['stake'], inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

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

logger.warning(f"Invalid inputs: stake={stake\_d}, back\_odds={back\_odds\_d}, lay\_odds={lay\_odds\_d}")

raise InvalidInputError("Stake and odds must be positive")

if back\_odds\_d < 1 or lay\_odds\_d < 1:

logger.warning(f"Odds below 1: back\_odds={back\_odds\_d}, lay\_odds={lay\_odds\_d}")

raise InvalidInputError("Odds must be >= 1.0")

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

if denominator <= 0:

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

raise DivisionByZeroError("Invalid lay odds or commission leading to division by zero")

lay\_stake = (stake\_d \* back\_odds\_d / denominator).quantize(Decimal('0.01'), rounding=ROUND\_HALF\_UP)

logger.debug(f"Calculated lay\_stake: {lay\_stake}")

return lay\_stake

@staticmethod

def calculate\_profit\_scenarios(stake: Union[int, float, Decimal], back\_odds\_dict: Dict[str, Union[int, float, Decimal]], lay\_odds\_dict: Dict[str, Union[int, float, Decimal]], lay\_stakes: Dict[str, Decimal], commission: Union[int, float, Decimal], back\_commission: Union[int, float, Decimal]) -> Dict[str, Decimal]:

inputs = {'stake': stake, 'back\_odds\_dict': back\_odds\_dict, 'lay\_odds\_dict': lay\_odds\_dict, 'lay\_stakes': lay\_stakes, 'commission': commission, 'back\_commission': back\_commission}

StrategyCalculators.validate\_inputs(inputs, ['stake', 'back\_odds\_dict', 'lay\_odds\_dict', 'lay\_stakes', 'commission', 'back\_commission'], {'stake': (int, float, Decimal), 'back\_odds\_dict': (dict,), 'lay\_odds\_dict': (dict,), 'lay\_stakes': (dict,), 'commission': (int, float, Decimal), 'back\_commission': (int, float, Decimal)})

stake\_d = inputs['stake']

back\_odds = {k: Decimal(str(v)) for k, v in back\_odds\_dict.items()}

lay\_odds = {k: Decimal(str(v)) for k, v in lay\_odds\_dict.items()}

commission\_d = inputs['commission']

back\_commission\_d = inputs['back\_commission']

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

logger.warning("Mismatched keys in back\_odds\_dict, lay\_odds\_dict, and lay\_stakes")

raise InvalidInputError("All dictionaries must have the same keys")

profit\_scenarios = {}

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

for outcome in back\_odds:

target\_liability = lay\_stakes[outcome] \* (lay\_odds[outcome] - 1) \* (1 - commission\_d)

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

profit\_scenarios[outcome] = profit.quantize(Decimal('0.01'), rounding=ROUND\_HALF\_UP)

return profit\_scenarios

@staticmethod

def create\_standard\_result(bet\_type: str, profit\_scenarios: Dict[str, Decimal], lay\_stakes: Dict[str, Decimal], stake: Decimal, commission: Decimal, back\_commission: Decimal, extra\_context: Optional[Dict[str, Any]] = None) -> Dict[str, Union[str, bool, Decimal, Dict]]:

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

result = {"type": bet\_type, "profit\_scenarios": profit\_scenarios, "min\_profit": min\_profit, "is\_profitable": min\_profit > 0, "lay\_stakes": lay\_stakes, "stake": stake, "commission": commission, "back\_commission": back\_commission}

if extra\_context:

result.update(extra\_context)

logger.info(f"{bet\_type} calculation: min\_profit={min\_profit:.2f}, is\_profitable={min\_profit > 0}")

return result

# Bet Type 1: 1X2

@staticmethod

def calculate\_1x2(

back\_odds\_dict: Dict[str, Union[int, float, Decimal]],

lay\_odds\_dict: Dict[str, Union[int, float, Decimal]],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds\_dict': back\_odds\_dict,

'lay\_odds\_dict': lay\_odds\_dict,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds\_dict', 'lay\_odds\_dict', 'stake'],

{'stake': (int, float, Decimal), 'back\_odds\_dict': (dict,), 'lay\_odds\_dict': (dict,), 'commission': (int, float, Decimal), 'back\_commission': (int, float, Decimal)}

)

stake\_d = inputs['stake']

lay\_stakes = {outcome: StrategyCalculators.calculate\_lay\_stake(stake\_d, back\_odds\_dict[outcome], lay\_odds\_dict[outcome], inputs['commission']) for outcome in back\_odds\_dict}

profit\_scenarios = StrategyCalculators.calculate\_profit\_scenarios(stake\_d, back\_odds\_dict, lay\_odds\_dict, lay\_stakes, inputs['commission'], inputs['back\_commission'])

return StrategyCalculators.create\_standard\_result("1X2", profit\_scenarios, lay\_stakes, stake\_d, inputs['commission'], inputs['back\_commission'])

# Bet Type 2: Double Chance 1X

@staticmethod

def calculate\_double\_chance\_1x(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Double Chance 1X",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 3: Double Chance X2

@staticmethod

def calculate\_double\_chance\_x2(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Double Chance X2",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 4: Double Chance 12

@staticmethod

def calculate\_double\_chance\_12(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Double Chance 12",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 5: Home Team Win Either Half

@staticmethod

def calculate\_home\_team\_win\_either\_half(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team Win Either Half",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 6: Home Team to Win from Behind

@staticmethod

def calculate\_home\_team\_to\_win\_from\_behind(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win from Behind",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 7: Home Team to Win to Nil

@staticmethod

def calculate\_home\_team\_to\_win\_to\_nil(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win to Nil",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 8: Match to Go to Extra Time

@staticmethod

def calculate\_match\_to\_go\_to\_extra\_time(

back\_odds\_dict: Dict[str, Union[int, float, Decimal]],

lay\_odds\_dict: Dict[str, Union[int, float, Decimal]],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds\_dict': back\_odds\_dict,

'lay\_odds\_dict': lay\_odds\_dict,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds\_dict', 'lay\_odds\_dict', 'stake'],

{'stake': (int, float, Decimal), 'back\_odds\_dict': (dict,), 'lay\_odds\_dict': (dict,), 'commission': (int, float, Decimal), 'back\_commission': (int, float, Decimal)}

)

stake\_d = inputs['stake']

lay\_stakes = {outcome: StrategyCalculators.calculate\_lay\_stake(stake\_d, back\_odds\_dict[outcome], lay\_odds\_dict[outcome], inputs['commission']) for outcome in back\_odds\_dict}

profit\_scenarios = StrategyCalculators.calculate\_profit\_scenarios(stake\_d, back\_odds\_dict, lay\_odds\_dict, lay\_stakes, inputs['commission'], inputs['back\_commission'])

return StrategyCalculators.create\_standard\_result("Match to Go to Extra Time", profit\_scenarios, lay\_stakes, stake\_d, inputs['commission'], inputs['back\_commission'])

# Bet Type 9: Match to Go to Penalties

@staticmethod

def calculate\_match\_to\_go\_to\_penalties(

back\_odds\_dict: Dict[str, Union[int, float, Decimal]],

lay\_odds\_dict: Dict[str, Union[int, float, Decimal]],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds\_dict': back\_odds\_dict,

'lay\_odds\_dict': lay\_odds\_dict,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds\_dict', 'lay\_odds\_dict', 'stake'],

{'stake': (int, float, Decimal), 'back\_odds\_dict': (dict,), 'lay\_ods\_dict': (dict,), 'commission': (int, float, Decimal), 'back\_commission': (int, float, Decimal)}

)

stake\_d = inputs['stake']

lay\_stakes = {outcome: StrategyCalculators.calculate\_lay\_stake(stake\_d, back\_odds\_dict[outcome], lay\_odds\_dict[outcome], inputs['commission']) for outcome in back\_odds\_dict}

profit\_scenarios = StrategyCalculators.calculate\_profit\_scenarios(stake\_d, back\_odds\_dict, lay\_odds\_dict, lay\_stakes, inputs['commission'], inputs['back\_commission'])

return StrategyCalculators.create\_standard\_result("Match to Go to Penalties", profit\_scenarios, lay\_stakes, stake\_d, inputs['commission'], inputs['back\_commission'])

# Bet Type 10: Home Team to Win a Penalty Shootout

@staticmethod

def calculate\_home\_team\_to\_win\_a\_penalty\_shootout(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win a Penalty Shootout",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 11: Half-Time Result

@staticmethod

def calculate\_half\_time\_result(

back\_odds\_dict: Dict[str, Union[int, float, Decimal]],

lay\_odds\_dict: Dict[str, Union[int, float, Decimal]],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds\_dict': back\_odds\_dict,

'lay\_odds\_dict': lay\_odds\_dict,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds\_dict', 'lay\_odds\_dict', 'stake'],

{'stake': (int, float, Decimal), 'back\_odds\_dict': (dict,), 'lay\_odds\_dict': (dict,), 'commission': (int, float, Decimal), 'back\_commission': (int, float, Decimal)}

)

stake\_d = inputs['stake']

lay\_stakes = {outcome: StrategyCalculators.calculate\_lay\_stake(stake\_d, back\_odds\_dict[outcome], lay\_odds\_dict[outcome], inputs['commission']) for outcome in back\_odds\_dict}

profit\_scenarios = StrategyCalculators.calculate\_profit\_scenarios(stake\_d, back\_odds\_dict, lay\_odds\_dict, lay\_stakes, inputs['commission'], inputs['back\_commission'])

return StrategyCalculators.create\_standard\_result("Half-Time Result", profit\_scenarios, lay\_stakes, stake\_d, inputs['commission'], inputs['back\_commission'])

# Bet Type 12: Full-Time Result with Handicap

@staticmethod

def calculate\_full\_time\_result\_with\_handicap(

back\_odds\_dict: Dict[str, Union[int, float, Decimal]],

lay\_odds\_dict: Dict[str, Union[int, float, Decimal]],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds\_dict': back\_odds\_dict,

'lay\_odds\_dict': lay\_odds\_dict,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds\_dict', 'lay\_odds\_dict', 'stake'],

{'stake': (int, float, Decimal), 'back\_odds\_dict': (dict,), 'lay\_odds\_dict': (dict,), 'commission': (int, float, Decimal), 'back\_commission': (int, float, Decimal)}

)

stake\_d = inputs['stake']

lay\_stakes = {outcome: StrategyCalculators.calculate\_lay\_stake(stake\_d, back\_odds\_dict[outcome], lay\_odds\_dict[outcome], inputs['commission']) for outcome in back\_odds\_dict}

profit\_scenarios = StrategyCalculators.calculate\_profit\_scenarios(stake\_d, back\_odds\_dict, lay\_odds\_dict, lay\_stakes, inputs['commission'], inputs['back\_commission'])

return StrategyCalculators.create\_standard\_result("Full-Time Result with Handicap", profit\_scenarios, lay\_stakes, stake\_d, inputs['commission'], inputs['back\_commission'])

# Bet Type 13: Home Team to Win by Exactly 1 Goal

@staticmethod

def calculate\_home\_team\_to\_win\_by\_exactly\_1\_goal(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win by Exactly 1 Goal",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 14: Home Team to Win by 2+ Goals

@staticmethod

def calculate\_home\_team\_to\_win\_by\_2\_plus\_goals(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win by 2+ Goals",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 15: Home Team to Win by 3+ Goals

@staticmethod

def calculate\_home\_team\_to\_win\_by\_3\_plus\_goals(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win by 3+ Goals",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 16: No Goalscorer

@staticmethod

def calculate\_no\_goalscorer(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="No Goalscorer",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 17: Home Team to Lead at Half-Time

@staticmethod

def calculate\_home\_team\_to\_lead\_at\_half\_time(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Lead at Half-Time",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 18: Home Team to Lead at Full-Time

@staticmethod

def calculate\_home\_team\_to\_lead\_at\_full\_time(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Lead at Full-Time",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 19: Home Team to Come Back from 2 Goals Down

@staticmethod

def calculate\_home\_team\_to\_come\_back\_from\_2\_goals\_down(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Come Back from 2 Goals Down",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 20: Match to End in Draw After Extra Time

@staticmethod

def calculate\_match\_to\_end\_in\_draw\_after\_extra\_time(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Match to End in Draw After Extra Time",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 21: Home Team to Win After Extra Time

@staticmethod

def calculate\_home\_team\_to\_win\_after\_extra\_time(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win After Extra Time",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 22: Home Team to Win in Regular Time Only

@staticmethod

def calculate\_home\_team\_to\_win\_in\_regular\_time\_only(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win in Regular Time Only",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 23: Match to End in a Shootout Draw

@staticmethod

def calculate\_match\_to\_end\_in\_a\_shootout\_draw(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Match to End in a Shootout Draw",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 24: Home Team to Win After Conceding First

@staticmethod

def calculate\_home\_team\_to\_win\_after\_conceding\_first(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win After Conceding First",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 25: Home Team to Win Without Scoring in First Half

@staticmethod

def calculate\_home\_team\_to\_win\_without\_scoring\_in\_first\_half(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win Without Scoring in First Half",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 26: Home Team to Win After Trailing at Half-Time

@staticmethod

def calculate\_home\_team\_to\_win\_after\_trailing\_at\_half\_time(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win After Trailing at Half-Time",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 27: Home Team to Win by 4+ Goals

@staticmethod

def calculate\_home\_team\_to\_win\_by\_4\_plus\_goals(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win by 4+ Goals",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 28: Home Team to Win by 5+ Goals

@staticmethod

def calculate\_home\_team\_to\_win\_by\_5\_plus\_goals(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win by 5+ Goals",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 29: Home Team to Win After Being Level at 75 Minutes

@staticmethod

def calculate\_home\_team\_to\_win\_after\_being\_level\_at\_75\_minutes(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win After Being Level at 75 Minutes",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 30: Home Team to Win Without Conceding in Second Half

@staticmethod

def calculate\_home\_team\_to\_win\_without\_conceding\_in\_second\_half(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win Without Conceding in Second Half",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 31: Home Team to Win After a Red Card

@staticmethod

def calculate\_home\_team\_to\_win\_after\_a\_red\_card(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win After a Red Card",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 32: Home Team to Win with a Clean Sheet in Extra Time

@staticmethod

def calculate\_home\_team\_to\_win\_with\_a\_clean\_sheet\_in\_extra\_time(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win with a Clean Sheet in Extra Time",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 33: Home Team to Win After Scoring First

@staticmethod

def calculate\_home\_team\_to\_win\_after\_scoring\_first(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win After Scoring First",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 34: Home Team to Win Without a Shot in First 15 Minutes

@staticmethod

def calculate\_home\_team\_to\_win\_without\_a\_shot\_in\_first\_15\_minutes(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win Without a Shot in First 15 Minutes",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 35: Home Team to Win After VAR Goal Decision

@staticmethod

def calculate\_home\_team\_to\_win\_after\_var\_goal\_decision(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win After VAR Goal Decision",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 36: Home Team to Win with Last Kick of the Game

@staticmethod

def calculate\_home\_team\_to\_win\_with\_last\_kick\_of\_the\_game(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win with Last Kick of the Game",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 37: Home Team to Win After Missing a Penalty

@staticmethod

def calculate\_home\_team\_to\_win\_after\_missing\_a\_penalty(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win After Missing a Penalty",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 38: Home Team to Win Without a Corner in First Half

@staticmethod

def calculate\_home\_team\_to\_win\_without\_a\_corner\_in\_first\_half(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win Without a Corner in First Half",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 39: Home Team to Win After Opponent Hits Woodwork

@staticmethod

def calculate\_home\_team\_to\_win\_after\_opponent\_hits\_woodwork(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win After Opponent Hits Woodwork",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 40: Home Team to Win with Fewer Shots on Target

@staticmethod

def calculate\_home\_team\_to\_win\_with\_fewer\_shots\_on\_target(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win with Fewer Shots on Target",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 41: Home Team to Win After Opponent Red Card

@staticmethod

def calculate\_home\_team\_to\_win\_after\_opponent\_red\_card(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win After Opponent Red Card",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 42: Home Team to Win Without Leading at Half-Time

@staticmethod

def calculate\_home\_team\_to\_win\_without\_leading\_at\_half\_time(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win Without Leading at Half-Time",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 43: Home Team to Win After 0-0 First Half

@staticmethod

def calculate\_home\_team\_to\_win\_after\_0\_0\_first\_half(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win After 0-0 First Half",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 44: Home Team to Win with Goal in Stoppage Time

@staticmethod

def calculate\_home\_team\_to\_win\_with\_goal\_in\_stoppage\_time(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win with Goal in Stoppage Time",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 45: Home Team to Win After Draw at 60 Minutes

@staticmethod

def calculate\_home\_team\_to\_win\_after\_draw\_at\_60\_minutes(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win After Draw at 60 Minutes",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 46: Home Team to Win Without Card Against Them

@staticmethod

def calculate\_home\_team\_to\_win\_without\_card\_against\_them(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win Without Card Against Them",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 47: Home Team to Win After Substitution Goal

@staticmethod

def calculate\_home\_team\_to\_win\_after\_substitution\_goal(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win After Substitution Goal",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 48: Home Team to Win with Fewer Corners

@staticmethod

def calculate\_home\_team\_to\_win\_with\_fewer\_corners(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win with Fewer Corners",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 49: Home Team to Win After Goalkeeper Save

@staticmethod

def calculate\_home\_team\_to\_win\_after\_goalkeeper\_save(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win After Goalkeeper Save",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

# Bet Type 50: Home Team to Win Without Possession Advantage

@staticmethod

def calculate\_home\_team\_to\_win\_without\_possession\_advantage(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Home Team to Win Without Possession Advantage",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)