Loan Investor Demonstration

November 13, 2023

1 Loan Investor Demonstration

This document demonstrates the motivations and functionality of the loan investor class. The investors are the primary players that interact with one another and hold, buy and sell loans. They are each assigned a trader and a target score which is their weighted average risk return based on interest rate. The goal of investors in the simulation will be to chase their target score based on a prespecified risk/return profiles. They would start with a random portfolio allocation of loans and then trade to achieve their target score. The simulation is specified in units of cycles which correspond to months in that loans have maturities based on 12 months. The following notebook illustrates some of the key attributes of the loan investor class.

[1]:
```
import Agents.LoanInvestor as LoanInvestor
import Agents.Loan as Loan
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
# creating a set of loans to allocate to investors
loan_list = [Loan.Loan() for i in range(1000)]

investor = LoanInvestor.LoanInvestor()
investor.initialize_portfolio(loan_list)
```

[3]:
```
vars(investor)
```

[3]:
```
{'id': 'I3d514c27-4bf1-4231-b547-a928297d6290',
 'capital': 48427619.743449286,
 'min_capital_pct': 0.15,
 'capital_history': [48427619.743449286],
 'target_score': 0.21394099127695954,
 'current_score': 0.24402408145593624,
 'current_cycle': 0,
 'loan_fair_values': [12725733.256550714],
 'portfolio_values': [61153353.0],
 'portfolio': [<Agents.Loan.Loan at 0x11f7197d0>,
  <Agents.Loan.Loan at 0x11f709e00>,
  <Agents.Loan.Loan at 0x11f757720>,
  <Agents.Loan.Loan at 0x11f6f5bf0>],
 'matured_loans': [],
```

```
 'trader': None,
 'interest_received': [],
 'loans_for_sale': []}
```

1.1 Initial Allocation of Loans

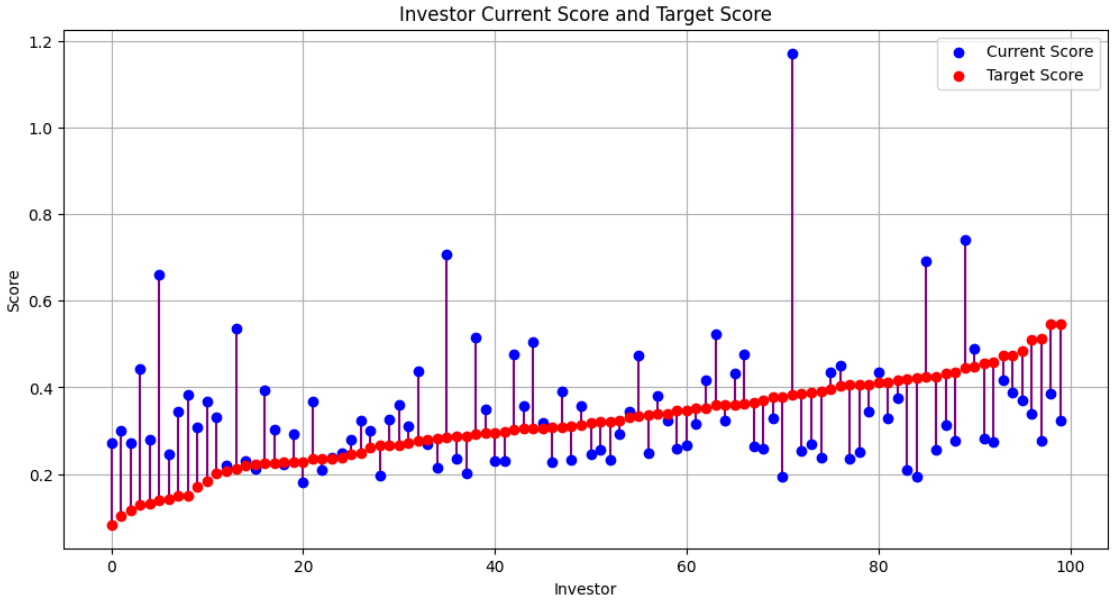
When we are allocating loans to investors, we want to make sure that the loans are allocated in a way that ensures that the number of investors above and below their target score is roughly the same. We tune the parameter for target score to match this target. This can be another aspect to tweak in the simulation to witness the differences when investors have different expectations than potential market reality and what will occur.

[4]:
```
%%script false --no-raise-error

def get_unsold_loans(num_loans, num_investors):
    loans = [Loan.Loan() for _ in range(num_loans)]
    investors = [LoanInvestor.LoanInvestor() for _ in range(num_investors)]

    for investor in investors:
        # regenerating list of available loans
        available_loans = [loan for loan in loans if loan.current_owner == "no␣
 ↪owner"]
        investor.generate_initial_capital()
        investor.initialize_portfolio(available_loans)

    unsold_loan_num = len([loan for loan in loans if loan.current_owner == "no␣
 ↪owner"])

    return unsold_loan_num


# generating a plot that shows how unsold loans scale with the number of loans␣
 ↪versus investors
unsold_loans = [get_unsold_loans(num_loans, 100) for num_loans in range(100,␣
 ↪10000, 100)]
plt.plot(range(100, 10000, 100), unsold_loans)
plt.xlabel("Number of Loans")
plt.ylabel("Number of Unsold Loans")
plt.title("Number of Unsold Loans vs. Number of Loans for a fixed 100␣
 ↪investors")
plt.show()
```

[5]:
```
# creating a universe of investors and loans
num_investors = 100
num_loans = 1000

investor_list = [LoanInvestor.LoanInvestor() for i in range(num_investors)]
```

```
loan_list = [Loan.Loan() for i in range(num_loans)]

# initializing the investor portfolios
for investor in investor_list:
    investor.generate_initial_capital()
    investor.initialize_portfolio([loan for loan in loan_list if loan.
 ↪current_owner == "no owner"])

# Sorting the scores for a more organized visualization
current_scores = np.array([investor.current_score for investor in␣
 ↪investor_list])
target_scores= np.array([investor.target_score for investor in investor_list])

# Sorting the scores for a more organized visualization
indices = np.argsort(target_scores)
current_scores_sorted = current_scores[indices]
target_scores_sorted = target_scores[indices]

# printing the number of investors that are above their target score
print("Number of investors above target score: ", sum([1 for a, b in␣
 ↪zip(current_scores, target_scores) if a > b]))
print("Number of investors below target score: ", sum([1 for a, b in␣
 ↪zip(current_scores, target_scores) if a < b]))

# Creating a scatter plot with connected lines
plt.figure(figsize=(12, 6))
for i in range(num_investors):
    plt.plot([i, i], [current_scores_sorted[i], target_scores_sorted[i]],␣
 ↪color='purple')  # lines
plt.scatter(range(num_investors), current_scores_sorted, color='blue',␣
 ↪label='Current Score', zorder=5)
plt.scatter(range(num_investors), target_scores_sorted, color='red',␣
 ↪label='Target Score', zorder=5)

plt.ylabel("Score")
plt.xlabel("Investor")
plt.title("Investor Current Score and Target Score")
plt.legend()
plt.grid(True)
plt.show()
```

```
Number of investors above target score:  51
Number of investors below target score:  49
```



[6]:
```
# getting the average number of investors above and below their target score
trials = 100
num_investors = 100
num_loans = 1000
num_investors_above = []
num_investors_below = []

for i in range(trials):
    investor_list = [LoanInvestor.LoanInvestor() for i in range(num_investors)]
    loan_list = [Loan.Loan() for i in range(num_loans)]

    current_scores = []
    target_scores = []

    # initializing the investor portfolios
    for investor in investor_list:
        investor.generate_initial_capital()
        investor.initialize_portfolio([loan for loan in loan_list if loan.
 ↪current_owner == "no owner"])
        current_scores.append(investor.current_score)
        target_scores.append(investor.target_score)

    num_investors_above.append(sum([1 for a, b in zip(current_scores,␣
 ↪target_scores) if a > b]))
    num_investors_below.append(sum([1 for a, b in zip(current_scores,␣
 ↪target_scores) if a < b]))
```

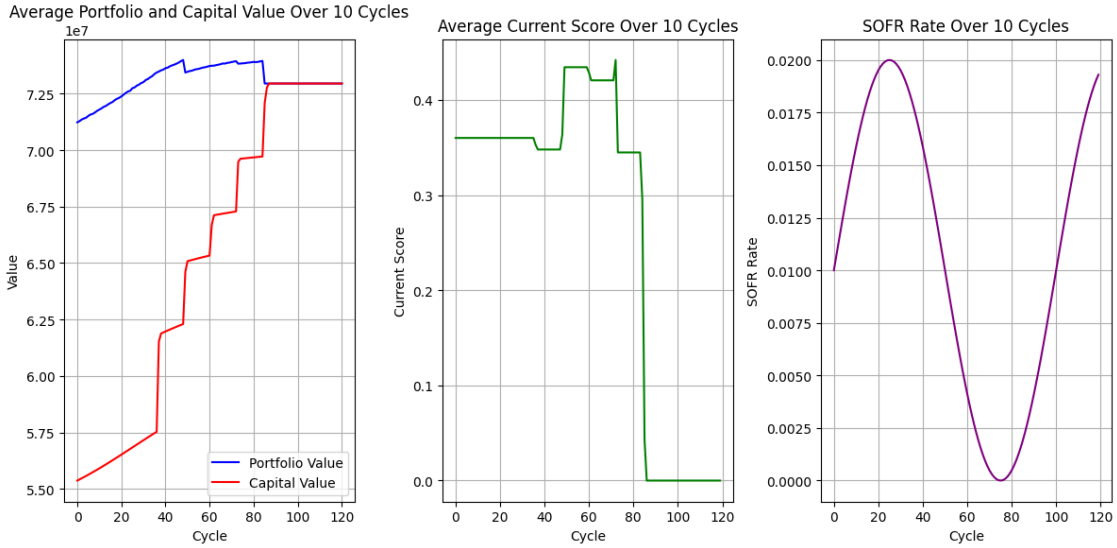
```
print("Average number of investors above target score: ", np.
 ↪mean(num_investors_above))
print("Average number of investors below target score: ", np.
 ↪mean(num_investors_below))
```

```
Average number of investors above target score:  51.86
Average number of investors below target score:  48.14
```

2 Updating: Portfolio and capital over cycles in a non-complex market

[7]:
```
# creating a universe of investors and loans
num_investors = 10
num_loans = 100
cycles = 120

investor_list = [LoanInvestor.LoanInvestor() for i in range(num_investors)]
loan_list = [Loan.Loan() for i in range(num_loans)]

# initializing the investor portfolios
for investor in investor_list:
    investor.generate_initial_capital()
    investor.initialize_portfolio([loan for loan in loan_list if loan.
 ↪current_owner == "no owner"])

avg_current_scores = []
portfolio_values = []
capital_values = []
sofr_rates = [0.01 + 0.01 * np.sin(2 * np.pi * i / 100) for i in range(cycles)]

# Update the market and gather data
for cycle in range(120):
    [loan.update(cycle) for loan in loan_list]
    [investor.update(float_interest=sofr_rates[cycle]) for investor in␣
 ↪investor_list]
    avg_current_scores.append(np.mean([investor.current_score for investor in␣
 ↪investor_list]))

for investor in investor_list:
    portfolio_values.append(investor.portfolio_values)
    capital_values.append(investor.capital_history)



# Taking the per cycle average of portfolio and capital
```

```
avg_portfolio_values = np.mean(portfolio_values, axis=0)
avg_capital_values = np.mean(capital_values, axis=0)

# Plotting the average portfolio, capital and current score over 10 cycles␣
 ↪using subplots
fig, axs = plt.subplots(1, 3, figsize=(12, 6))

# Plotting both portfolio and capital values on the same subplot
axs[0].plot(range(121), avg_portfolio_values, label='Portfolio Value',␣
 ↪color='blue')
axs[0].plot(range(121), avg_capital_values, label='Capital Value', color='red')
axs[0].set_title("Average Portfolio and Capital Value Over 10 Cycles")
axs[0].set_ylabel("Value")
axs[0].set_xlabel("Cycle")
axs[0].legend()
axs[0].grid(True)

# Plotting the current score on a separate subplot
axs[1].plot(range(120), avg_current_scores, color='green')
axs[1].set_title("Average Current Score Over 10 Cycles")
axs[1].set_ylabel("Current Score")
axs[1].set_xlabel("Cycle")
axs[1].grid(True)

# Plotting the SOFR rate on a separate subplot
axs[2].plot(range(120), sofr_rates, color='purple')
axs[2].set_title("SOFR Rate Over 10 Cycles")
axs[2].set_ylabel("SOFR Rate")
axs[2].set_xlabel("Cycle")
axs[2].grid(True)


plt.tight_layout()
plt.show()
```



2.1 Investor Trading: Buying and Selling Loans

The goal of this simulation is to create a plausible market in which investors are individually making intelligent decisions about their loan portfolios. In order to do this, we have investors make deicsions on selling loans based on deviation of the loans score absent of weight and decide to buy loans based on their 'maximized' price in which the investor takes the attributes of their loan in conjunction with their target score to generate a maximum price that they would be willing to pay for the loan.

2.1.1 Investor Buying Loans by generating bid prices

[8]:
```
# creating a universe of investors and loans
num_investors = 10
num_loans = 100

investor_list = [LoanInvestor.LoanInvestor() for i in range(num_investors)]
loan_list = [Loan.Loan() for i in range(num_loans)]

# initializing the investor portfolios
for investor in investor_list:
    investor.generate_initial_capital()
    investor.initialize_portfolio([loan for loan in loan_list if loan.
 ↪current_owner == "no owner"])

loan_for_sale = Loan.Loan()
# printing out the relevant information for the loan for sale
print("Loan for Sale: ", (loan_for_sale.as_dict()))
```

```
Loan for Sale:  {'id': '3bb2737b-8db9-4513-91e0-41eaacfcb811', 'maturity': 72,
'current_cycle': 0, 'starting_cycle': 0, 'ending_cycle': 72, 'time_to_maturity':
72, 'pd': 0.003652493524299095, 'size': 3526275.053513838, 'interest_rate':
0.019868293144482005, 'fair_value': 99.31466600350396, 'market_price':
99.31466600350396, 'current_owner': 'no owner', 'maturity_bool': False,
'fair_value_history': [99.31466600350396], 'market_price_history':
[99.31466600350396], 'ownership_history': ['no owner'], 'sale_price_history':
[None], 'reserve_price': None}
```

[9]:
```
loan_for_sale = Loan.Loan()
# printing out the relevant information for the loan for sale
print("Loan for Sale: ", loan_for_sale.as_dict())

# getting the bid prices for a new loan from every investor using the portfolio␣
 ↪weighted method
bid_prices_p_inc = [investor.get_bid_price(loan_for_sale,␣
 ↪pricing_method='portfolio_included') for investor in investor_list]
bid_prices_p_n = [investor.get_bid_price(loan_for_sale,␣
 ↪pricing_method='portfolio_neutral') for investor in investor_list]

print("\n Bid Prices Portfolio Included: ", bid_prices_p_inc)
print("\n Bid Prices Portfolio Neutral: ", bid_prices_p_n)
```

```
Loan for Sale:  {'id': '42fa58bf-2908-4b72-a409-6d8d9bce74d4', 'maturity': 60,
'current_cycle': 0, 'starting_cycle': 0, 'ending_cycle': 60, 'time_to_maturity':
60, 'pd': 0.07591736571950287, 'size': 4408727.50106454, 'interest_rate':
0.027233317054347308, 'fair_value': 99.99532223268774, 'market_price':
99.99532223268774, 'current_owner': 'no owner', 'maturity_bool': False,
'fair_value_history': [99.99532223268774], 'market_price_history':
[99.99532223268774], 'ownership_history': ['no owner'], 'sale_price_history':
[None], 'reserve_price': None}

 Bid Prices Portfolio Included:  [87.49717317141642, 84.25805102546083,
86.07179870918652, 84.95039024102016, 87.56345542522878, 89.89864853008773,
88.66641111388628, 91.76409041234554, 86.63912675127075, 86.54393333967121]

 Bid Prices Portfolio Neutral:  [87.48712061624913, 84.26812117778955,
86.09844047973057, 84.96781709621695, 87.59651320632906, 89.90644396517341,
88.65837866999448, 91.7215287416379, 86.64735832976221, 86.5441074054902]
```

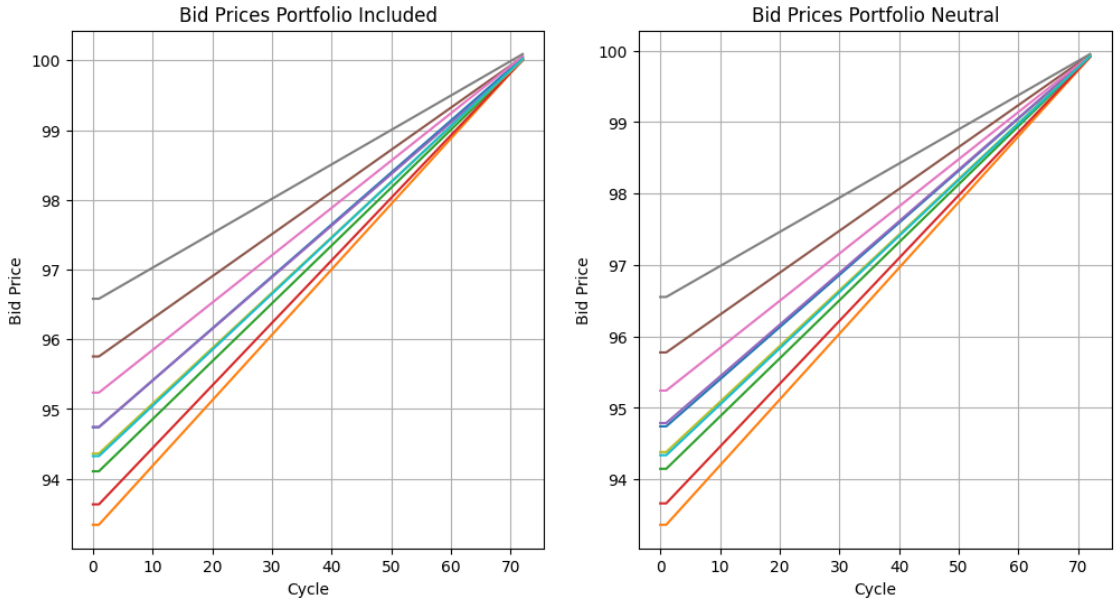
[10]:
```
loan_for_sale = Loan.Loan(current_cycle=0)

# printing out the relevant information for the loan for sale
print("Loan for Sale Pre-update: ", loan_for_sale.as_dict())

# plotting bid prices until the loan matures
bid_prices_p_inc = []
bid_prices_p_n = []
```

```
cycles = 120

for cycle in range(cycles):
    bid_prices_p_inc.append([investor.get_bid_price(loan_for_sale,␣
 ↪pricing_method='portfolio_included') for investor in investor_list])
    bid_prices_p_n.append([investor.get_bid_price(loan_for_sale,␣
 ↪pricing_method='portfolio_neutral') for investor in investor_list])
    loan_for_sale.update(current_cycle=cycle)
    [investor.update() for investor in investor_list]

print("\nLoan for Sale post-updates: ", list(loan_for_sale.as_dict().items())[:
 ↪-2])

print("\n Investor Capital: ", [investor.capital for investor in investor_list]␣
 ↪)

# plotting the bid prices over time where each list is a list of lists for each␣
 ↪investor as 2 subplots. Each investor is colored differently

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(range(cycles), bid_prices_p_inc)
plt.title("Bid Prices Portfolio Included")
plt.xlabel("Cycle")
plt.ylabel("Bid Price")
plt.grid(True)

plt.subplot(1, 2, 2)
plt.plot(range(cycles), bid_prices_p_n)
plt.title("Bid Prices Portfolio Neutral")
plt.xlabel("Cycle")
plt.ylabel("Bid Price")
plt.grid(True)
```

```
Loan for Sale Pre-update:  {'id': '46166e8d-96e1-4af7-bce3-b875b7ec1517',
'maturity': 72, 'current_cycle': 0, 'starting_cycle': 0, 'ending_cycle': 72,
'time_to_maturity': 72, 'pd': 0.027066961742515352, 'size': 4994077.134665801,
'interest_rate': 0.025385328834027096, 'fair_value': 99.34131098817127,
'market_price': 99.34131098817127, 'current_owner': 'no owner', 'maturity_bool':
False, 'fair_value_history': [99.34131098817127], 'market_price_history':
[99.34131098817127], 'ownership_history': ['no owner'], 'sale_price_history':
[None], 'reserve_price': None}

Loan for Sale post-update:  [('id', '46166e8d-96e1-4af7-bce3-b875b7ec1517'),
('maturity', 72), ('current_cycle', 72), ('starting_cycle', 0), ('ending_cycle',
72), ('time_to_maturity', 0), ('pd', 0.027066961742515352), ('size',
```

4994077.134665801), ('interest_rate', 0.025385328834027096), ('fair_value', 100), ('market_price', 100), ('current_owner', 'no owner'), ('maturity_bool', True), ('fair_value_history', [99.34131098817127, 99.42888869070835, 100.01181234236512, 99.90881466016987, 99.92289705170087, 99.42046003908767, 99.2730909687915, 100.04536643127146, 99.69571089026489, 100.27597137098435, 99.60051507843396, 100.80684520863137, 98.7257716381672, 99.29547623754623, 100.11689228364563, 100.31466609670906, 100.07671438703564, 99.85093255325155, 99.65925862807767, 99.69760222087763, 99.89244437257493, 100.6663822096993, 99.40845677528073, 99.84945717853668, 98.63721020211118, 99.56590691689777, 99.16174711449698, 100.0413411942977, 99.0290889781052, 100.24879714016768, 99.42652087707242, 99.26888216119731, 99.02579953511757, 98.88148884722776, 100.4053223130948, 99.31747673253848, 100.16054445973661, 99.86591892112608, 99.65438958601834, 98.93760740759929, 99.79933307461927, 99.45818413355896, 99.42695742428462, 100.08577724655254, 99.43748255031268, 100.45556444278397, 99.79368101119871, 99.14676232578904, 101.11593157354925, 98.67830709601897, 98.92764930388273, 99.9061277304383, 100.04579948768009, 99.72441541830034, 100.37651197469938, 100.4238669212983, 100.26248499409732, 99.02928208236412, 99.61645646053445, 99.17892420119561, 99.49606908144635, 99.66397215338478, 101.06268492491573, 99.72235943300358, 100.56345028234907, 100.11188873912434, 98.74285517492504, 100.14596741054524, 100.61222973807779, 99.71275093841216, 99.91299479492714, 99.57960445960751, 99.6941486178305, 100]), ('market_price_history', [99.34131098817127, 99.34131098817127]), ('ownership_history', ['no owner'])]

Investor Capital: [61084625.96158689, 89765694.2434825, 82843968.01583278, 85354770.82932183, 58284091.435220264, 121570097.04822196, 118286519.35354528, 47287777.34283248, 44167541.93832011, 62557917.82899326]



2.1.2 Investor getting a loan for sale out of it's portfolio

The basic way that this works is to find the loans that contribute to the current score in the wrong direction by the most possible value. This is completed absent of size where the investors are looking to dump the loans with the largest wrong-way interest rate over size. The reason it is done absent of size is to minimize only the largest loans from being dumped from investors.

[11]:
```
# creating a universe of investors and loans
num_investors = 10
num_loans = 100

investor_list = [LoanInvestor.LoanInvestor() for i in range(num_investors)]
loan_list = [Loan.Loan() for i in range(num_loans)]

# initializing the investor portfolios
for investor in investor_list:
    investor.generate_initial_capital()
    investor.initialize_portfolio([loan for loan in loan_list if loan.
 ↪current_owner == "no owner"])

print('Loan For Sale Attributes', investor_list[0].get_loan_to_sell().as_dict())
print('\nInvestor Attributes', vars(investor_list[0]))
```

```
Loan For Sale Attributes {'id': '1451cc2e-cf2f-4323-8d3f-e721bbdd7ae4',
'maturity': 84, 'current_cycle': 0, 'starting_cycle': 0, 'ending_cycle': 84,
'time_to_maturity': 84, 'pd': 0.06990910301112045, 'size': 598928.9045898146,
'interest_rate': 0.016200261306326548, 'fair_value': 99.0543195035587,
'market_price': 99.0543195035587, 'current_owner':
```

```
'I9252826b-0352-4da7-a342-1b544e52573c', 'maturity_bool': False,
'fair_value_history': [99.0543195035587], 'market_price_history':
[99.0543195035587], 'ownership_history': ['no owner',
'I9252826b-0352-4da7-a342-1b544e52573c'], 'sale_price_history': [None],
'reserve_price': 79.24345560284696}

Investor Attributes {'id': 'I9252826b-0352-4da7-a342-1b544e52573c', 'capital':
45246010.52604283, 'min_capital_pct': 0.15, 'capital_history':
[45246010.52604283], 'target_score': 0.22692343332089965, 'current_score':
0.31744579387442945, 'current_cycle': 0, 'loan_fair_values':
[14400831.473957174], 'portfolio_values': [59646842.0], 'portfolio':
[<Agents.Loan.Loan object at 0x12e44aeb0>, <Agents.Loan.Loan object at
0x12e44a3b0>, <Agents.Loan.Loan object at 0x12e264ca0>, <Agents.Loan.Loan object
at 0x12e448930>, <Agents.Loan.Loan object at 0x12e44d3b0>], 'matured_loans': [],
'trader': None, 'interest_received': [], 'loans_for_sale': [<Agents.Loan.Loan
object at 0x12e44a3b0>]}
```

[11]: