Loan Class Visualizations

The following notebook explains the function of the starting loan class and breaks down each of its functions

Attributes

The loan class is the base class for all loans in the simulation and initialized with the following attributes:

- `id` **(str)**:
  - Unique identifier for the loan.
- `maturity` **(int)**:
  - Number of cycles until the loan matures.
- `current_cycle` **(int)**:
  - The current cycle of the simulation.
- `starting_cycle` **(int)**:
  - The cycle when the loan was initiated.
- `ending_cycle` **(int)**:
  - The cycle when the loan will end.
- `time_to_maturity` **(int)**:
  - Number of cycles until the loan matures.
- `pd` **(float)**:
  - Probability of default for the loan.
- `size` **(float)**:
  - Size of the loan.
- `interest_rate` **(float)**:
  - Interest rate of the loan.
- `fair_value` **(float)**:
  - Fair value of the loan.
- `market_price` **(float)**:
  - Market price of the loan.
- `current_owner` **(str)**:

    - The current owner of the loan.
- **`maturity_bool` (bool)**:
    - Indicates if the loan has matured.
- **`fair_value_history` (list)**:
    - History of the loan's fair values.
- **`market_price_history` (list)**:
    - History of the loan's market prices.
- **`ownership_history` (list)**:
    - History of the loan's ownership.

Methods

`__init__(self, current_cycle=0, current_owner="no owner")`

Initializes the Loan with random values for its attributes.

- **Parameters**:
    - `current_cycle` (int, default = 0): The current cycle of the simulation.
    - `current_owner` (str, default = "no owner"): The initial owner of the loan.

Attribute Initializations:

- `maturity`: Generated randomly using a uniform distribution, $U(3 \times 12, 8 \times 12)$.
- `pd`: Generated using a beta distribution, $\text{Beta}(1, 50)$.
- `size`: Generated using a uniform distribution, $U(500,000, 5,000,000)$.

`generate_interest_rate(self)`

Generates a random interest rate based on some factors and noise.

- **Returns**:
    - A float representing the generated interest rate.

Interest Rate Calculation:

- Base noise component: $\mathcal{N}(0.02, 0.005)$
- Correlation with probability of default: $\text{correlation\_factor} \times \text{pd} \times \text{influence\_factor}$

`calculate_price(self)`

Calculates the price of the loan based on its attributes.

- **Returns**:
  - A float representing the calculated price of the loan.

Price Calculation Influences:

- Probability of Default (pd) effect is doubled for loans with a PD of greater than 20%:
  - $-7 \times \text{pd}$ if $\text{pd} \leq 0.2$
  - $-14 \times \text{pd}$ otherwise
- Interest Rate (ir) effect: $5 \times \text{interest\_rate}$
- Size effect: $-0.0000002 \times \text{size}$

`generate()`

Static method to generate a new `Loan` instance.

- **Returns**:
  - A new `Loan` object.

`update_owner(self, new_owner)`

Updates the current owner of the loan.

- **Parameters**:
  - `new_owner` (str): The new owner of the loan.

`update(self, current_cycle, new_owner=None, new_market_price=None)`

Updates the loan's attributes for a new cycle. If the loan has matured, checks for default and adjusts fair value and market price accordingly.

- **Parameters**:
  - `current_cycle` (int): The new current cycle.
  - `new_owner` (str, optional): The new owner of the loan, if it has changed.
  - `new_market_price` (float, optional): The new market price of the loan, if it has changed.

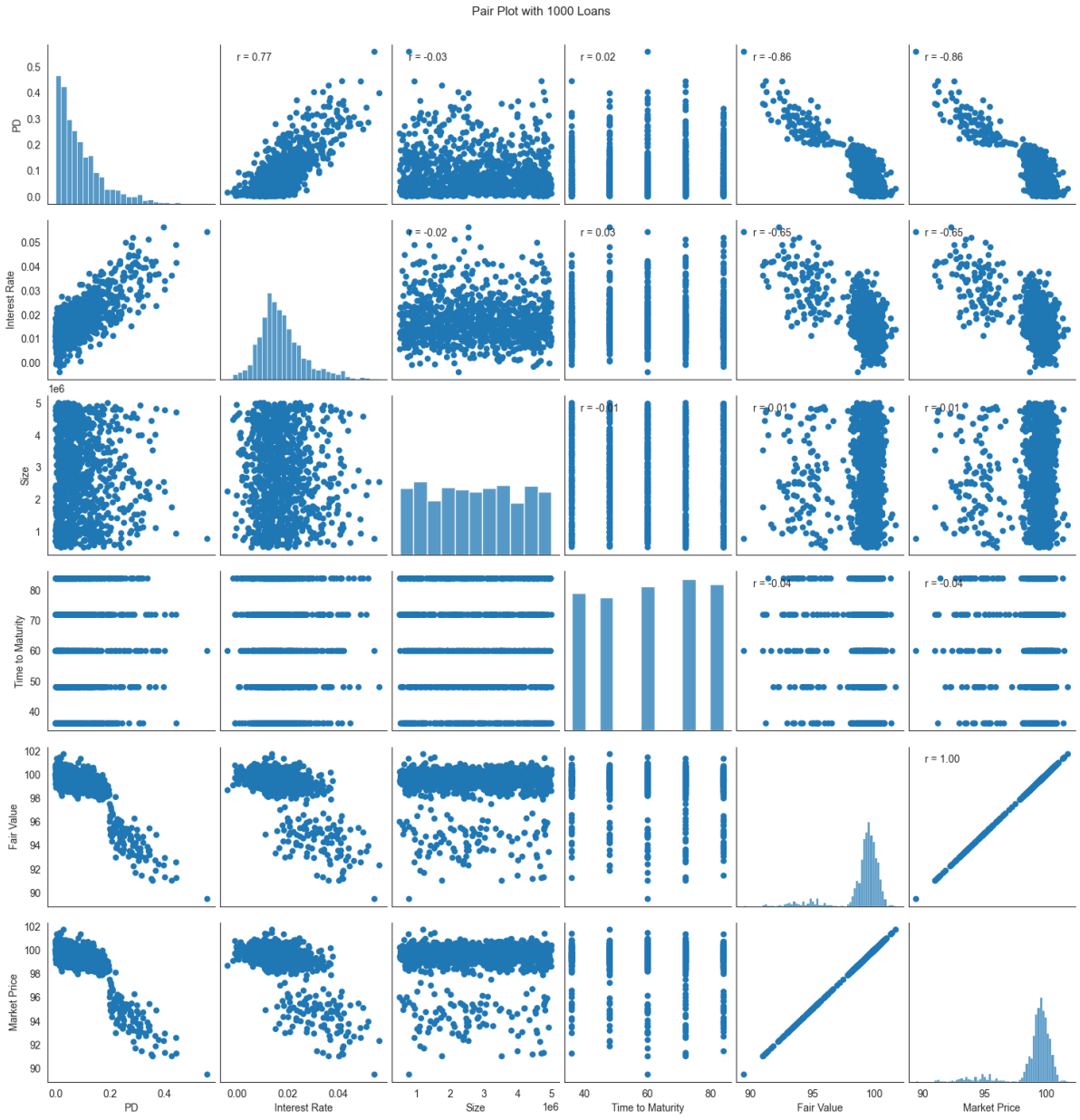
Maturity Logic:

- If the loan has matured:
  - Checks for default using a random value: $\text{default\_outcome} = \text{Random} < \text{pd}$
  - If default occurs:

        - Fair value and market price are set based on a normal distribution: $\mathcal{N}(10, 2)$
    - Otherwise:
        - Fair value and market price are set to 100 (par value).

In [1]:

```
import Agents.Loan as Loan
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt


def corr_annotate(x, y, **kws):
    r = np.corrcoef(x, y)[0, 1]
    ax = plt.gca()
    ax.annotate(f"r = {r:.2f}", xy=(.1, .9), xycoords=ax.transAxes)

# Regenerate the loans with a sample size of 50 and extract their attributes
loan_num = 1000

loans = [Loan.Loan.generate() for _ in range(loan_num)]
attributes = {
    "PD": [loan.pd for loan in loans],
    "Interest Rate": [loan.interest_rate for loan in loans],
    "Size": [loan.size for loan in loans],
    "Time to Maturity": [loan.time_to_maturity for loan in loans],
    "Fair Value": [loan.fair_value for loan in loans],
    "Market Price": [loan.market_price for loan in loans]
}

df_loans_updated = pd.DataFrame(attributes)

# Create the pair plot without gridlines and with circle markers
sns.set_style("white")
grid = sns.pairplot(df_loans_updated, plot_kws={"edgecolor": "none", "marker
grid.map_upper(corr_annotate)
plt.suptitle("Pair Plot with "+str(loan_num)+" Loans", y=1.02)
plt.show()
```


Pair Plot with 1000 Loans
PD
Interest Rate
Size
Time to Maturity
Fair Value
Market Price
r = 0.77
r = -0.03
r = 0.02
r = -0.86
r = -0.86
r = -0.02
r = 0.03
r = -0.65
r = -0.65
r = -0.01
r = 0.01
r = 0.01
r = -0.04
r = -0.04
r = 1.00
1e6

In [1]: