Credit Risk: Rate Estimation Project

Deadline: 2020-05-11 23:59 hrs

Contents

(10 pts) PR.I: Loan Rates

Create a python application capable of determining the optimal interest rate of a simple loan and an enriched amortization table.

Q1 (1 pts) Create an `Amortization` class on `models/finance.py`.

- Attributes on `__init__`:
  - `amount`: monetary value representing the loan value.
  - `rate`: interest rate of the loan.
  - `n`: number of terms (periods).
- Minimum methods:
  - `get_table`: returns the amortization table with the following columns:
    - `t`
    - `PRINCIPAL`
    - `INTEREST`
    - `ANNUITY`
    - `BALANCE`
  - `get_enriched_table`: given the `prob_of_default` and `loss_given_default` as arguments, create the amortization table with the following extra columns:
    - `IRR`: internal rate of return for each term.
    - `EL`: expected loss value using `EAD` as the `BALANCE`.
    - `PROB`: probability of scenario.
  - `expected_irr`: given the `prob_of_default` and `loss_given_default` as arguments, calculate the expected value of the IRR.

Example:

```
from models.finance import Amortization

a = Amortization(amount=10000, rate=0.15, n=5)

# Get the amortization table as a DataFrame
simple_df = a.get_table()

# Get the enriched amortization table as a DataFrame
enriched_df = a.get_enriched_table(prob_of_default=0.20, loss_given_default=0.5)

# Get the expected IRR (should be a float) considering the default scenarios.
irr = a.expected_irr(prob_of_default=0.20, loss_given_default=0.5)
```

Q2 (2 pts) Correctly calculates the enriched amortization table.

```
from models.finance import Amortization

a = Amortization(amount=10000, rate=0.15, n=5)
print(a.get_enriched_table(prob_of_default=0.20, loss_given_default=0.5))
```

Q3 (2 pts) Correctly calculates the expected IRR.

HINT: Assume the IRR at t=0 is 0.

```
from models.finance import Amortization

a = Amortization(amount=10000, rate=0.15, n=5)
print(a.expected_irr(prob_of_default=0.20, loss_given_default=0.5))
```

Q4 (2 pts) Create a `search` function that returns an interest rate value such that the expected IRR approximates an arbitrary value (default=0).

Consider the following:

- Your search space should be defined between `client_min_rate` and `client_max_rate`.
- Use a `search_samples` argument to parametrize the number of samples to be taken.

HINT: You might be able to solve this problem by using the square error concept and solving the quadratic equation.

```
from utils import search

# Feel free to use the arguments of your choice.
# The result should be a float representing an interest rate.
estimated_interest_rate = search(...)
```

Q5 (3 pts) The following command `loan-request-local` should run as expected:

```
$ python main.py loan-request-local \
    --loan-amount=10000 \
    --loan-marr=0.15 \
    --loan-terms=5 \
    --probability-of-default=0.20 \
    --loss-given-default=0.50 \
    --client-min-rate=0.06 \
    --client-max-rate=0.75 \
    --search-samples=100 \
    --save example.csv \
    --show
```

```
Loan request for 10000 due in 5 terms with 3919.15 fix payments and interest rate of 27.61% : Approved
   t   BALANCE  PRINCIPAL  INTEREST  ANNUITY   IRR       EL  PROB
0  0 10,000.00        nan       nan      nan  0.00 1,000.00  0.20
1  1  8,841.83   1,158.17  2,760.98 3,919.15 -0.61   884.18  0.16
2  2  7,363.90   1,477.94  2,441.22 3,919.15 -0.15   736.39  0.13
3  3  5,477.91   1,885.99  2,033.16 3,919.15  0.09   547.79  0.10
4  4  3,071.20   2,406.71  1,512.44 3,919.15  0.21   307.12  0.08
5  5      0.00   3,071.20    847.95 3,919.15  0.28     0.00  0.33
```

Consider the following:

- The loan-request is `Approved` IF the output of `search` (optimal interest rate) is greater than the `loan_marr` value. Otherwise, it should be `Rejected`.
- Save the enriched amortization table when `--search` ends with a `.csv` pattern. Use an empty string as the default value.
- Only print the message `Loan request for ...` and the enriched amortization table when `--show` is True.

(10 pts) PR.II: Google Integration

Integrate the **PR.I** python application into Google Sheets!

Q1 (2 pts) Complete the `GoogleSheetWrapper` class on `models/drive.py` with the following methods:

- `get_value(self, col, row)`: Given a column letter `col` (string) and a row number `row` (string or int) get the Google Sheet value. Return value should always be a string.
- `get_values(self, from_col, from_row, to_col, to_row)`: Given two cells (`from_` and `to_`) return all the values in between those cells. Consider the following scenarios:
  - If rows are the same, return a single flat list.
  - If cols are the same, return a single flat list.
  - If rows and cols are the same, call the `get_value` method which return a string.
  - Else: return a nested list were the inner lists represents the rows.

```python
from models.drive import GoogleSheetWrapper

sheet = GoogleSheetWrapper(
    spreadsheet_id="10RVyxDjdchzcl_BmjmpODC7t43P9OAKaCdY69jxa3XM",
    sheet_name="CONFIG"
)

# Single value
# RETURN: 'LABEL'
sheet.get_value(col="B", row=3)

# Multiple values but form = to should call the get_value and return a string
# RETURN: 'LABEL'
sheet.get_values(from_col="B", from_row=3, to_col="B", to_row=3)

# Multiple values but same row should return a flat list
# RETURN: ['LABEL', 'VALUE']
sheet.get_values(from_col="B", from_row=3, to_col="C", to_row=3)

# Multiple values but same column should return a flat list
# RETURN: ['PROB_OF_DEFAULT', 'LGD', 'LOAN_AMOUNT']
sheet.get_values(from_col="B", from_row=4, to_col="B", to_row=6)

# Otherwise, nested lists were the inner list represents a row are accepted.
# RETURN: [['LABEL', 'VALUE'], ['PROB_OF_DEFAULT', '0.2']]
sheet.get_values(from_col="B", from_row=3, to_col="C", to_row=4)
```

Q2 (1 pts) Complete the `ConfigTable` properties to return the correct value from a google sheet:

- `probability_of_default`
- `loss_given_default`
- `loan_amount`
- `loan_terms`
- `loan_marr`
- `client_min_rate`
- `client_max_rate`
- `search_samples`

```
from models.drive import ConfigTable

config = ConfigTable(spreadsheet_id="10RVyxDjdchzcl_BmjmpODC7t43P9OAKaCdY69jxa3XM")

# Everything should work fine
config.probability_of_default  # 0.2
config.loss_given_default  # 0.5
config.loan_amount  # 10000.0
config.loan_terms  # 5
config.loan_marr  # 0.15
config.client_min_rate  # 0.05
config.client_max_rate  # 0.85
config.search_samples  # 100
```

Q3 (2 pts) The `ConfigTable` implementation should work even with a new position on the Google Sheet.

Q4 (1 pts) The `RateValue` should contain a method `update` that updated the correct cell on the spreadsheet.

Q5 (1 pts) The `RequestValue` should contain a method `update` that updated the correct cell on the spreadsheet.

Q6 (1 pts) The `ResultTable` should contain a method `update` that updated the correct cell on the spreadsheet.

Q7 (2 pts) The following command `loan-request-cloud` should run as expected:

```
$ python main.py loan-request-cloud --sheet-id 10RVyxDjdchzcl_BmjmpODC7t43P9OAKaCdY69jxa3XM --show


Loan request for 10000.0 due in 5 terms with 3919.14 fix payments and interest rate of 27.61% : Approve
   t   BALANCE  PRINCIPAL  INTEREST  ANNUITY   IRR       EL  PROB
0  0 10,000.00        nan       nan      nan  0.00 1,000.00  0.20
1  1  8,841.83   1,158.17  2,760.97 3,919.14 -0.61   884.18  0.16
2  2  7,363.89   1,477.94  2,441.20 3,919.14 -0.15   736.39  0.13
3  3  5,477.90   1,885.99  2,033.15 3,919.14  0.09   547.79  0.10
4  4  3,071.19   2,406.71  1,512.43 3,919.14  0.21   307.12  0.08
5  5      0.00   3,071.19    847.95 3,919.14  0.28     0.00  0.33
```

Consider the following:

- The loan-request is `Approved` IF the output of `search` (optimal interest rate) is greater than the `loan_marr` value. Otherwise, it should be `Rejected`.
- Only print the message `Loan request for ...` and the enriched amortization table when `--show` is True.
- The data should be updated on the target Google Sheet.