Machine Learning in Finance: Mock exam 1

Duration: 180 minutes. Part 1 and Part 2 are worth 50 points each. Work locally in Jupyter with downloaded course materials and personal notes. Internet and AI are not allowed.

Submission. Put all code and written answers in the answer notebook. Save it as `MLFIN_MOCK_2026_1_answers.ipynb`. Submit that notebook together with the supplied `MLFIN_MOCK_2026_1_blank_submission.pdf`. The PDF is intentionally blank; do not put answers in it.

The supplied `MLFIN_MOCK_2026_1_code_solutions.ipynb` is for self-checking. It contains code only; written explanations and answers to explanation-only questions are intentionally omitted.

Partial solutions can earn points. Show completed steps even if unfinished. Later Part 2 work that correctly follows an earlier answer can earn marks; state your assumptions. If a question tells you to use a particular method, use it. Otherwise, any correct method is accepted.

Part 1 (50 points)

Part 1 contains four independent programming problems. Each block supplies the inputs and definitions needed for that block; its later subparts may use your earlier result in the same block.

(A) **A tiered investment fee (12 points)**

An investment account has a nonnegative balance b in euros. Its annual fee $F(b)$, also in euros, is

$$F(b) = \begin{cases} 0, & 0 \le b \le 1000, \\ 0.006(b - 1000), & 1000 < b \le 5000, \\ 24 + 0.003(b - 5000), & b > 5000. \end{cases}$$

(i) **7 points.** Write `fee(balance)` from scratch. It must return one float equal to the displayed fee for any nonnegative balance, including the tier boundaries.

(ii) **5 points.** Call your function on balances `0`, `1000`, `5000`, and `6500`, in that order. Print the four fees. State how many extra euros of fee arise when the balance rises from 5,000 to 6,500 euros.

(B) **Present values with NumPy (14 points)**

Rows of `cashflows` are Bond A and Bond B; its columns are payments at the end of years 1, 2, and 3. All amounts are euros. The length-3 array `factors` supplies a weight for each year. For bond i, define its present value and net value by

$$V_i = \sum_{j=1}^{3} c_{ij} f_j, \qquad N_i = V_i - P_i,$$

where c_{ij} is that bond's payment in year j, f_j is the matching factor, and P_i is its purchase price. Use the supplied factors directly.

```
import numpy as np
cashflows = np.array([[20, 20, 120],
                      [10, 10, 110]])
factors = np.array([0.95, 0.90, 0.85])
price = np.array([135, 115])
```

(i) **5 points.** Use NumPy's matrix product `@` to calculate the two present values in bond order. Print them to two decimals.

(ii) **6 points.** Complete the four blanks. The elementwise method must give the same two present values, and `net_value` must contain the two net values. Print the agreement result and the net values to two decimals.

```
check = (cashflows * ...).sum(axis=...)
print(np.allclose(present_value, ...))
net_value = present_value - ...
print(net_value.round(2))
```

(iii) **3 points.** Print the name of the bond with the larger net value. In one sentence, explain why adding the three cash payments without their factors can give a different conclusion about value.

(C) **When is an invoice paid? (12 points)**

`invoice` maps each customer name to the euro amount due. `payments` maps the same name to a list of nonnegative euro payments on days 1, 2, and so on. For a customer, the total paid by the end of day d is

$$S_d = \sum_{j=1}^{d} p_j,$$

where p_j is the payment on day j. The invoice is paid on the first day for which S_d is at least the amount due. If that never happens, there is no paid day.

```
invoice = {'Ada': 120, 'Ben': 90, 'Cleo': 100}
payments = {'Ada': [40, 30, 50],
            'Ben': [30, 20, 10],
            'Cleo': [80, 20]}
```

(i) **6 points.** Write `first_paid_day(amount,daily_payments)`. Return the first paid day as an integer starting at 1, or `None` if the invoice is not fully paid.

(ii) **6 points.** Build and print `paid_day`, a dictionary mapping every customer to the result of your function. Then print the customer paid earliest and that day. If several customers tie, choose the name that comes first alphabetically.

(D) **Electricity-price changes (12 points)**

The lists give electricity prices in euros per MWh on six consecutive days. `days[i]` and `prices[i]` refer to the same day. A one-day return from day $i-1$ to day i is $p_i/p_{i-1}-1$. A two-day return starting on day i is p_{i+2}/p_i-1. All prices are positive.

```
days = [1, 2, 3, 4, 5, 6]
prices = [40, 45, 42, 47, 49, 46]
```

(i) **4 points.** With a loop, build `daily` containing the five one-day returns in date order. Print the list rounded to three decimals.

(ii) **5 points.** With a loop, build `two_day` containing every two-day return in start-day order. Print the list rounded to three decimals.

(iii) **3 points.** Print the starting day and return of the largest two-day increase, with the return formatted as a percentage to two decimals. If there is a tie, use the earliest starting day.

Part 2 (50 points)

A bond desk reviews new issues before investing. You are helping it identify issuers that may need closer review. Use a **RandomForestClassifier** to forecast whether an issuer will default within one year. The simulated CSV has one issuer at issuance per row. It records the issuer's leverage, interest cover, cash ratio, and profit margin, along with the bond's yield spread above a reference yield. `default_1y` records a default in the following year (1) or no default (0); `issuer_id` identifies the issuer.

Column	Meaning
`issuer_id`	Unique issuer label; identifier, not a predictor.
`leverage_ratio`	Debt divided by assets; a fraction.
`interest_cover`	Earnings divided by interest expense; a positive ratio.
`cash_ratio`	Cash divided by assets; a fraction.
`bond_spread_bp`	Bond yield spread above a reference yield, in basis points.
`profit_margin`	Profit divided by revenue; a fraction that may be negative.
`default_1y`	Target: 1 if the issuer defaults within one year, otherwise 0.

(A) **Inspect and prepare bond issuers (13 points)**

A basis point is one hundredth of a percentage point. For each bond, let s^{bp} be `bond_spread_bp` and let s^{pct} be the new `bond_spread_pct`. Calculate

$$s^{\text{pct}} = \frac{s^{\text{bp}}}{100}.$$

Use 25% of rows as test and the rest as train. A default is the positive class.

(i) **2 points.** Load the CSV into `df`. Print its shape and first three rows.

(ii) **4 points.** Add `bond_spread_pct` to `df`. Print its first three values to two decimals. Print how many bonds have a spread of at least 1.5 percentage points.

(iii) **4 points.** Make `X` from leverage, interest cover, cash ratio, `bond_spread_pct`, and profit margin, in that order; make `y` from `default_1y`. Split into train and test with 25% test, `random_state=17`, and stratification by `y`. Print the train and test row counts.

(iv) **3 points.** A constant reference gives every test bond the train default share as its score. Calculate and print its test ROC AUC to three decimals.

(B) **Choose a forest leaf size (10 points)**

A forest's `min_samples_leaf` is the smallest number of train rows allowed in a leaf. Compare 5, 20, and 60. Use 100 trees, `max_features='sqrt'`, and `random_state=31` for each forest. Calculate three-fold train/val ROC AUC for each setting. Choose the highest mean; if exactly tied, choose the smaller leaf size.

(i) **6 points.** Calculate and print the mean three-fold val ROC AUC for each leaf size, labelled by size and rounded to three decimals.

(ii) **4 points.** Select the leaf size by the stated rule. Instantiate `forest` with that size and the same fixed settings, fit it on all train rows, and print the selected size.

(C) **Rank test default risk (7 points)**

Use the selected fitted forest from the previous stage.

(i) **4 points.** Store the forest's default probabilities for test issuers as `p_test`. Calculate and print their test ROC AUC to three decimals.

(ii) **3 points.** Compare the test ROC AUC with the constant reference and the selected mean train/val ROC AUC. Explain in plain language what ROC AUC measures in this setting.

(D) **Which inputs does the forest use? (12 points)**

Built-in importance measures the forest's accumulated split improvement. Permutation importance measures the drop in test ROC AUC after a predictor is shuffled.

(i) **7 points.** Create one horizontal bar chart of the forest's built-in importances, ordered from smallest at the bottom to largest at the top. Put predictor names on the vertical axis and importance on the horizontal axis. Give the figure a title. Print the ordered values to three decimals.

(ii) **5 points.** Calculate permutation importance on the test rows using ROC AUC, three repeats, and `random_state=19`. Print each predictor's mean importance to three decimals, largest first. Name the top predictor under each importance measure and state whether they agree.

(E) **An issuer-review note (8 points)**

Write a short conclusion for a bond-risk desk using your measured results.

(i) **8 points.** In your own words, assess whether the forest is useful for deciding which issuers to review. Support your view with its measured test ROC AUC against the reference, one financially meaningful predictor from the importance results, and one limitation of those results. Give a sensible next step for the desk in at most five sentences.