Quantitative Risk Management
FIN-417

Homework 2

by William Jallot (SCIPER 341540)
Antoine Garin (SCIPER 327295)
Matthias Wyss (SCIPER 329884)

Quantitative Risk Management, MA3

Dr. Urban Ulrych

Contents

Autumn Semester - 2025

General instructions. You may work in groups of up to three students (smaller groups are also allowed). Your submission must contain:

1. A single **PDF report** including a clear description of your methodology, the results of your experiments (tables and figures), and your *commented code* in an appendix.
2. The corresponding **code files** in Python (.ipynb or .py).

The PDF should be fully self-contained: all results, plots, and tables must be visible in the report without executing the code. When the provided scripts are executed, they should exactly reproduce the results shown in your PDF. Please list all group members on the title page of your report.

Grading. This project is worth 10% of the final grade. You will be evaluated based on the correctness of your implementation, the clarity and completeness of your explanations, and the quality of your analysis and interpretation of results.

Project description. This project focuses on the quantitative modeling of credit risk for retail loans. You will work with *synthetic* data where each loan applicant is described by three characteristics: age, income, and employment type. Your goal is to build and evaluate statistical learning models that predict repayment probabilities and to assess the profitability and risk of different lending strategies.

1 Exercise 1: Feature generation

1. **Feature generation.** Simulate $m+n$ feature vectors $x^i = (x_1^i, x_2^i, x_3^i) \in \mathbb{R}^3$, $i = 1, \dots, m+n$, with $m = 20000$ and $n = 10000$, where:
   - x_1^i: age, uniformly distributed on $[18, 80]$,
   - x_2^i: monthly income (in thousands of CHF), uniformly distributed on $[1, 15]$,
   - x_3^i: employment status in $\{0, 1\}$ with $\mathbb{P}(x_3^i = 1) = 0.1$ (self-employed) and $\mathbb{P}(x_3^i = 0) = 0.9$ (salaried).

   Assume independence across the three coordinates.

   a) Compute empirical means and standard deviations of each feature based on the first m samples.

   b) Suggest other variables that would realistically be relevant in credit scoring (beyond age, income, and employment type).

In this exercise we construct synthetic feature vectors for retail loan applicants and compute basic summary statistics on the training sample.
We fix the total number of observations to be

$$m = 20000, \qquad n = 10000, \qquad m + n = 30000,$$

where m denotes the size of the training set and n the size of the test set. For each applicant $i = 1, \dots, m+n$ we generate a three-dimensional feature vector

$$x_i = (x_{i1}, x_{i2}, x_{i3})^\top \in \mathbb{R}^3,$$

where the coordinates represent:

$$x_{i1} = \text{age}, \quad x_{i2} = \text{monthly income (in thousand of CHF)}, \quad x_{i3} = \text{employment status}.$$

The features are simulated under the following probabilistic model, assuming independence across coordinates:

$$x_{i1} \sim \text{Unif}[18, 80], \qquad x_{i2} \sim \text{Unif}[1, 15], \qquad \mathbb{P}(x_{i3} = 1) = 0.1,\ \mathbb{P}(x_{i3} = 0) = 0.9.$$

Here $x_{i3} = 0$ encodes a salaried employee and $x_{i3} = 1$ a self-employed person. In the Python implementation, a fixed random seed is set via `np.random.seed(0)` to ensure reproducibility of all numerical results.

1.A Empirical means and Standard deviations

For each feature $j = 1, 2, 3$ we compute its empirical mean and empirical standard deviation on the training sample. Denoting the j-th coordinate of x_i by x_{ij}, the empirical mean is

$$\hat{\mu}_j = \frac{1}{m} \sum_{i=1}^{m} x_{ij},$$

and the (unbiased) empirical standard deviation is given by

$$\hat{\sigma}_j = \sqrt{\frac{1}{m-1} \sum_{i=1}^{m} \big(x_{ij} - \hat{\mu}_j\big)^2}.$$

In the code this corresponds to `X_train.mean(axis=0)` for the means and `X_train.std(axis=0, ddof=1)` for the standard deviations, where the option `ddof=1` implements the factor $1/(m-1)$ in the denominator.
For our simulated dataset, the resulting estimates (rounded to four decimal places) are:

$$\hat{\mu} \approx (48.7427,\ 7.9865,\ 0.1017),$$

$$\hat{\sigma} \approx (18.0079,\ 4.0309,\ 0.3023).$$

These values are reasonable given the underlying distributions: the means for age and income are close to the midpoints of their respective uniform intervals, and the mean of the employment status is close to the theoretical probability 0.1 of being self-employed. The standard deviations reflect the spread of the uniform distributions and the Bernoulli nature of the employment indicator.

1.B Other variables relevant in credit scoring

Beyond age, income, and employment type, many other borrower characteristics are typically important in credit scoring because they provide additional information about both the ability and the willingness to repay. Examples include variables summarizing the applicant's credit history, such as the number of past defaults, the number of open credit lines, and the timing of any late payments, which directly capture historical repayment behaviour. Another key quantity is the debt-to-income ratio, measuring the share of the applicant's income already committed to servicing existing debt: a high ratio suggests limited capacity to take on new obligations.
Employment stability is often highly informative. This can be represented by the tenure in the current job and the total number of job changes over a given period, as more stable employment generally implies more stable income. Demographic and household information, such as marital status or the number of dependents, can affect the level of mandatory expenses and thus the disposable income available for loan repayment. Education level is sometimes used as a proxy for long-term income potential and employment prospects.
Finally, information about housing and wealth, for instance whether the applicant is a renter or homeowner and the presence of savings or other financial assets, can provide a buffer against income shocks. Together, such additional variables allow a more nuanced assessment of credit risk than using age, income, and employment type alone, and can therefore improve the accuracy and robustness of credit scoring models.

2 Exercise 2: Default model and data generation

2. **Default model and data generation.** Let $(\xi^i)_{i=1}^{m+n}$ be i.i.d. Unif(0,1). Define the sigmoid function as $\psi(z) = 1/(1+e^{-z})$. Consider two repayment probability functions $p_1, p_2 : \mathbb{R}^3 \to (0,1)$:

$$p_1(x) = \psi(13.3 - 0.33x_1 + 3.5x_2 - 3x_3),$$
$$p_2(x) = \psi\big(5 - 10\big[\mathbf{1}_{(-\infty,25)}(x_1) + \mathbf{1}_{(75,\infty)}(x_1)\big] + 1.1x_2 - x_3\big).$$

Construct two datasets (x^i, y_1^i) and (x^i, y_2^i), $i = 1, \dots, m+n$, by setting

$$y_s^i = \begin{cases} 1, & \text{if } \xi^i \le p_s(x^i), \\ 0, & \text{otherwise,} \end{cases} \qquad s = 1, 2.$$

Interpretation: $y_s^i = 1$ means borrower i repays, $y_s^i = 0$ means default. For each dataset $s = 1, 2$:

a) Fit a logistic regression model $\hat{p}_s^{\log} : \mathbb{R}^3 \to \mathbb{R}$ on the training data (x^i, y_s^i), $i = 1, \dots, m$. Report cross-entropy loss on both training and test sets ($i = m+1, \dots, m+n$). You can use the Python function `sklearn.linear_model.LogisticRegression`.

b) Standardize features by dividing each coordinate by its empirical standard deviation from the training set. Fit a Support Vector Machine (SVM) classifier with Gaussian kernel

$$k(x, x') = \exp\left(-\frac{1}{10}\|x - x'\|_2^2\right),$$

using hinge loss and regularization parameter $\lambda = \frac{5}{2m}$. In `sklearn.svm.SVC`, this corresponds to $C = 0.2$. Use the option `probability=True` to obtain probability estimates $\hat{p}_s^{\text{svm}}$ (Platt scaling[1]). Evaluate cross-entropy loss of $\hat{p}_s^{\text{svm}}$ on both training and test sets.

c) Plot the ROC curves, i.e., graphs of the True Positive Rate (TPR) versus the False Positive Rate (FPR), using the test data. Additionally, compute the corresponding Area Under the ROC Curve (AUC) for $\hat{p}_s^{\log}$ and $\hat{p}_s^{\text{svm}}$.

In this exercise we construct two synthetic default models and corresponding datasets, and then fit logistic regression classifiers and Support Vector Machine (SVM) classifier with Gaussian kernel to estimate repayment probabilities. We evaluate the quality of these probabilistic predictions using the cross-entropy (negative conditional log-likelihood) on both training and test sets.
We first define the sigmoid (logistic) function

$$\psi(z) = \frac{1}{1+e^{-z}},$$

which maps any real-valued input to the open interval $(0,1)$ and will be used to convert linear predictors into probabilities.
Given the feature vectors $x_i = (x_{i1}, x_{i2}, x_{i3})^\top$ constructed in Exercise 1, we introduce an auxiliary sequence $(\xi_i)_{i=1}^{m+n}$ of i.i.d. random variables with

$$\xi_i \sim \text{Unif}(0,1),$$

generated once for all observations. These uniforms are used to simulate Bernoulli repayment indicators conditional on the features.
We consider two repayment probability functions $p_1, p_2 : \mathbb{R}^3 \to (0,1)$.
The first model is a standard logistic regression-type specification with a linear predictor in the features:

$$p_1(x_1, x_2, x_3) = \psi\big(13.3 - 0.33x_1 + 3.5x_2 - 3x_3\big).$$

Given p_1, the corresponding binary repayment indicator y_{i1} for borrower i is defined by

$$y_{i1} = \begin{cases} 1, & \text{if } \xi_i \le p_1(x_i), \\ 0, & \text{otherwise,} \end{cases} \qquad i = 1, \dots, m+n.$$

Thus y_{i1} is a Bernoulli random variable with success probability $p_1(x_i)$, representing full repayment ($y_{i1} = 1$) versus default ($y_{i1} = 0$). In the code this is implemented via the comparison `(xi <= p1(x1, x2, x3)).astype(int)`.
The second model has a more complex, non-linear dependence on age through indicator functions:

$$p_2(x_1, x_2, x_3) = \psi\Big(5 - 10\mathbf{1}_{\{x_1<25\}} - 10\mathbf{1}_{\{x_1>75\}} + 1.1x_2 - x_3\Big).$$

Here the coefficients -10 applied to the indicators for very young ($x_1 < 25$) and very old ($x_1 > 75$) borrowers sharply reduce the log-odds of repayment for those age groups, creating strong non-linearity in x_1. The repayment indicator y_{i2} is defined analogously by

$$y_{i2} = \begin{cases} 1, & \text{if } \xi_i \le p_2(x_i), \\ 0, & \text{otherwise,} \end{cases} \qquad i = 1, \dots, m+n,$$

again implemented as `(xi <= p2(x1, x2, x3)).astype(int)`. This yields two labelled datasets (x_i, y_{i1}) and (x_i, y_{i2}) for $i = 1, \dots, m+n$.
We then split both datasets into training and test sets using the same partition as in Exercise 1. Writing $X \in \mathbb{R}^{(m+n)\times 3}$ for the full feature matrix and $y_s \in \{0,1\}^{m+n}$ for the labels of dataset $s \in \{1, 2\}$, we set

$$X_{\text{train}} = X_{1:m,:}, \qquad X_{\text{test}} = X_{m+1:m+n,:},$$
$$y_{s,\text{train}} = (y_{1s}, \dots, y_{ms})^\top, \qquad y_{s,\text{test}} = (y_{m+1,s}, \dots, y_{m+n,s})^\top.$$

2.A Logistic regression model

For each dataset $s \in \{1, 2\}$, we fit a logistic regression model $\hat{p}_s^{\text{log}}$. The model assumes that, conditional on the features, the repayment indicator is Bernoulli with probability

$$\hat{p}_s^{\text{log}}(x) = \psi\big(\hat{\beta}_{s0} + \hat{\beta}_{s1}x_1 + \hat{\beta}_{s2}x_2 + \hat{\beta}_{s3}x_3\big),$$

where the parameters $\hat{\beta}_s = (\hat{\beta}_{s0}, \dots, \hat{\beta}_{s3})$ are estimated by maximizing the conditional log-likelihood on the training data. In practice this is done by calling

$$\texttt{logreg} = \texttt{LogisticRegression(max\_iter=1000)}, \qquad \texttt{logreg.fit}(X_{\text{train}}, y_{s,\text{train}}).$$

The option `max_iter=1000` ensures that the numerical optimizer has enough iterations to converge.
Once the model is fitted, we obtain predicted class probabilities on both training and test sets via

$$\hat{P}_s^{\text{train}} = \hat{p}_s^{\text{log}}(X_{\text{train}}), \qquad \hat{P}_s^{\text{test}} = \hat{p}_s^{\text{log}}(X_{\text{test}}),$$

implemented as `logreg.predict_proba(X_train)` and `logreg.predict_proba(X_test)`. The function `predict_proba` returns for each observation a two-dimensional vector $(\hat{P}(Y = 0 \mid X), \hat{P}(Y = 1 \mid X))$.
To evaluate how well these probabilistic predictions match the true labels, we use the cross-entropy loss (negative conditional log-likelihood). For a generic dataset with observations (x_i, y_i) and predicted probabilities $\hat{p}_i = \hat{p}(Y = 1 \mid x_i)$, the empirical cross-entropy loss is

$$\ell_{\text{CE}} = -\frac{1}{N}\sum_{i=1}^{N}\big[y_i \log \hat{p}_i + (1 - y_i)\log(1 - \hat{p}_i)\big],$$

where N is the sample size. This quantity is computed using `sklearn.metrics.log_loss` on both training and test sets:

$$\texttt{log\_loss}(y_{s,\text{train}}, \hat{P}_s^{\text{train}}), \qquad \texttt{log\_loss}(y_{s,\text{test}}, \hat{P}_s^{\text{test}}).$$

For dataset 1, the cross-entropy loss is very small on both training and test data:

$$\ell_{\text{CE, train}}^{(1)} \approx 0.0297, \qquad \ell_{\text{CE, test}}^{(1)} \approx 0.0349.$$

This is expected because the true repayment probability p_1 is exactly a logistic function of the three features, so the logistic regression model is correctly specified and can closely recover the generating mechanism.
For dataset 2, the losses are significantly larger:

$$\ell^{(2)}_{\text{CE, train}} \approx 0.1535, \qquad \ell^{(2)}_{\text{CE, test}} \approx 0.1486.$$

Here the logistic regression model is misspecified: the true p_2 contains non-linear effects of age through indicator functions, which cannot be exactly represented by a linear predictor in (x_1, x_2, x_3). As a result, the fitted model provides a cruder approximation of the true repayment probabilities, leading to higher cross-entropy loss even though performance remains similar between training and test, indicating limited overfitting.
Overall, this exercise illustrates how logistic regression behaves when the underlying data-generating process is correctly specified (dataset 1) versus when important non-linearities are present but not explicitly modeled (dataset 2), and how the cross-entropy loss provides a natural quantitative measure of predictive calibration for probabilistic classifiers.

2.B SVM model

We now replace logistic regression by a nonlinear classifier based on support vector machines (SVMs) with a Gaussian radial basis function (RBF) kernel, using the same two synthetic datasets as in Section 2. The goal is again to approximate the conditional repayment probabilities and to evaluate predictive performance via the cross-entropy loss on training and test sets.
Since SVMs with RBF kernels are sensitive to the scaling of the input, we first standardize each feature using the training data. Let $X_{\text{train}} \in \mathbb{R}^{m\times 3}$ and denote by x_{ij} the value of feature j in observation i. For $j = 1, 2, 3$ we compute

$$\hat{\mu}_j = \frac{1}{m}\sum_{i=1}^{m} x_{ij}, \qquad \hat{\sigma}_j = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}(x_{ij}-\hat{\mu}_j)^2},$$

and define standardized features

$$z_{ij} = \frac{x_{ij}-\hat{\mu}_j}{\hat{\sigma}_j},$$

yielding $X_{\text{train,scaled}}$ and $X_{\text{test,scaled}}$. In the code this is done via `StandardScaler` with `fit_transform` on the training set and `transform` on the test set.
For each dataset $s \in \{1, 2\}$ we then fit an SVM with Gaussian RBF kernel

$$K(x, x') = \exp\bigl(-\gamma \|x - x'\|_2^2\bigr),$$

using $\gamma = 1/10$ and regularization parameter $C = 0.2$:

```
SVC(kernel='rbf', C=0.2, gamma=0.1, probability=True).
```

The `probability=True` option performs a post-hoc logistic calibration (Platt scaling) so that the signed distances $f_s(x)$ to the decision boundary are mapped to estimated probabilities

$$\hat{p}_s(x) \approx \frac{1}{1+\exp\bigl(A_s f_s(x) + B_s\bigr)} \approx \mathbb{P}(Y = 1 \mid X = x).$$

The function `predict_proba` returns $\hat{P}_s^{\text{train}}$ and $\hat{P}_s^{\text{test}}$ for training and test sets, respectively.
To assess the quality of these probabilistic predictions, we use the cross-entropy (negative log-likelihood) loss. For a sample (x_i, y_i) with predicted probability $\hat{p}_i = \hat{p}(Y = 1 \mid x_i)$, the empirical loss is

$$\ell_{\text{CE}} = -\frac{1}{N}\sum_{i=1}^{N}\Bigl[y_i \log \hat{p}_i + (1-y_i)\log\bigl(1-\hat{p}_i\bigr)\Bigr],$$

computed in practice via `log_loss` on both training and test predictions.
For dataset 1, the SVM with RBF kernel yields

$$\ell^{(1,\text{SVM})}_{\text{CE, train}} \approx 0.0319, \qquad \ell^{(1,\text{SVM})}_{\text{CE, test}} \approx 0.0388,$$

which is close to, but slightly worse than, the logistic regression losses from Section 2, consistent with the fact that the true repayment probability is exactly logistic in (x_1, x_2, x_3).
For dataset 2, the SVM performs significantly better than logistic regression, with

$$\ell^{(2,\text{SVM})}_{\text{CE, train}} \approx 0.0711, \qquad \ell^{(2,\text{SVM})}_{\text{CE, test}} \approx 0.0671.$$

This substantial reduction in cross-entropy loss reflects the ability of the Gaussian kernel to capture the strong nonlinear dependence of the default probability on age, which a purely linear logistic model cannot represent.

2.C ROC curves

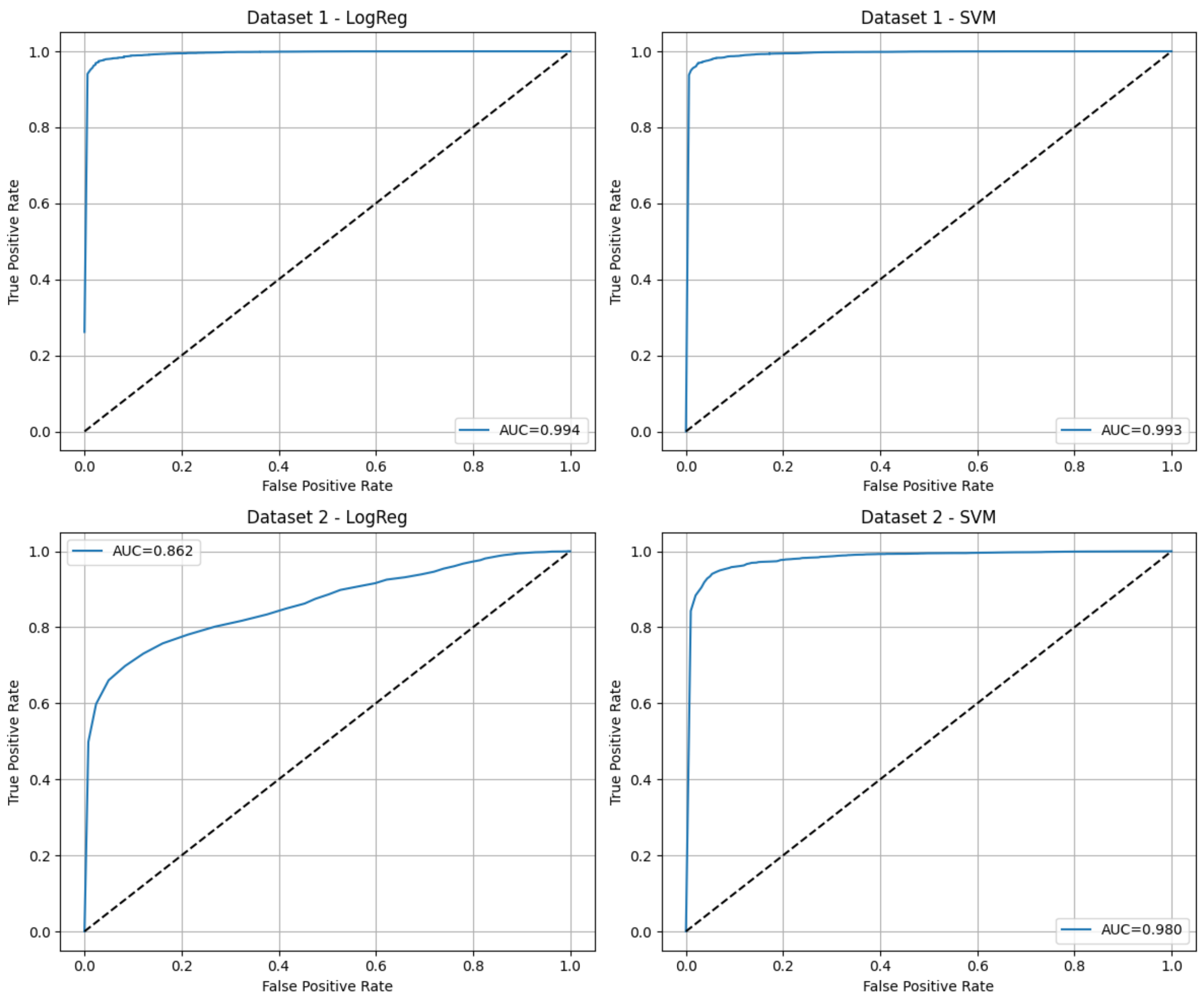

Figure 1: Plot of the ROC curves, i.e., graphs of the True Positive Rate (TPR) versus the False Positive Rate (FPR), using the test data.

Dataset	Logistic Regression (AUC)	SVM (AUC)
Dataset 1	0.994	0.993
Dataset 2	0.862	0.980

Table 1: Comparison of AUC scores for Logistic Regression and SVM across two datasets

Table 1 reports the Area Under the ROC Curve (AUC) obtained by Logistic Regression and Support Vector Machine (SVM) classifiers on two different datasets.
For **Dataset 1**, both models achieve near-perfect discrimination performance, with AUC values of 0.994 for Logistic Regression and 0.993 for SVM. This indicates that the classes are highly separable and that even a simple linear classifier is sufficient to capture the decision boundary.

In contrast, for **Dataset 2**, a significant difference in performance is observed between the two models. Logistic Regression achieves an AUC of 0.862, while SVM reaches an AUC of 0.980. The lower AUC obtained by Logistic Regression suggests limited capacity to model the data structure. The RBF kernel successfully captures the non-linear age thresholds at 25 and 75 encoded in the indicator functions of p_2, whereas logistic regression, due to its linear decision boundary, is unable to model such non-linear effects.

3 Exercise 3: Lending startegies

3. **Lending strategies.** Focus now on dataset 2, i.e., (x^i, y_2^i). Assume each loan is of CHF 1000 and that repayment is all-or-nothing (full repayment with interest, or total default). Consider the following lending policies applied to the test set $(i = m+1, \ldots, m+n)$:

   (i) Lend to everyone at 5.5% interest.

   (ii) Lend selectively at 1% interest, but only to applicants with $\hat{p}_2^{\log}(x^i) \geq 95\%$.

   (iii) Same as (ii), but selection based on $\hat{p}_2^{\mathrm{svm}}$.

   To evaluate performance, simulate 50000 scenarios of repayment outcomes: for each test applicant i, draw independent $\xi^{i,k}$ ~Unif(0, 1) and set

$$D_{i,k} = \mathbf{1}\{\xi^{i,k} \leq p_2(x^{m+i})\}, \qquad i = 1, \ldots, n,\ k = 1, \ldots, 50000.$$

   Here $D_{i,k} = 1$ indicates repayment in scenario k. For each strategy (i)–(iii):

   a) Plot the histogram of profits & losses across the 50000 scenarios and compute the expected profit & loss.

   b) Estimate the 95%-VaR and 95%-ES of the profit & loss distribution.

In this exercise we evaluate three lending policies on the test sample of dataset 2. Each loan has a face value of CHF 1000 and the repayment outcome is all-or-nothing (full repayment plus interest, or total loss). We simulate $K = 50{,}000$ independent repayment scenarios using the true repayment probability $p_2(x)$ defined in Exercise 2. For each test applicant i and scenario k, we draw

$$\xi_{i,k} \sim \mathrm{Unif}(0,1), \qquad D_{i,k} = \mathbf{1}\{\xi_{i,k} \leq p_2(x_{m+i})\},$$

so that $D_{i,k} = 1$ indicates full repayment and $D_{i,k} = 0$ corresponds to default.

3.A Lending policies

We compare three portfolio rules applied to the test set $(i = m+1, \ldots, m+n)$:

1. **Strategy (i): Lend to everyone at 5.5% interest.** All applicants receive a loan. Gross repayment factor: $r = 1.055$.

2. **Strategy (ii): Logistic regression screening at 1% interest.** Grant a loan only if the logistic regression estimate satisfies $\hat{p}_2^{\log}(x) \geq 0.95$. Gross factor: $r = 1.01$.

3. **Strategy (iii): SVM (RBF) screening at 1% interest.** Grant a loan only if the calibrated SVM predicts $\hat{p}_2^{\mathrm{svm}}(x) \geq 0.95$. Gross factor: $r = 1.01$.

Let N_{sel} denote the number of accepted applicants for a strategy, and let R_k be the number of repaid loans in scenario k. The scenario-wise P&L is

$$\Pi_k = 1000\big(rR_k - (N_{\text{sel}} - R_k)\big) = 1000\big((r+1)R_k - N_{\text{sel}}\big).$$

3.B Simulation outputs and risk measures

For each strategy, let $\Pi_1, \ldots, \Pi_K$ be the simulated profits. Define the loss as

$$L_k = -\Pi_k.$$

- **Expected Profit:**

$$\widehat{\mathbb{E}}[\Pi] = \frac{1}{K}\sum_{k=1}^{K}\Pi_k.$$

- **95% Value-at-Risk (VaR):**

$$\text{VaR}_{95\%} = \text{Quantile}_{95\%}(L) = q_{0.95}(L),$$

  i.e., the minimal loss exceeded with 5% probability.

- **95% Expected Shortfall (ES):**

$$\text{ES}_{95\%} = \mathbb{E}[L \mid L \geq \text{VaR}_{95\%}] = \frac{1}{|\{k : L_k \geq q_{0.95}(L)\}|}\sum_{\{k:L_k\geq q_{0.95}(L)\}} L_k.$$

3.C Results and interpretation

Figure 2, Figure 3 and Figure 4 display the empirical P&L distributions obtained from the simulation.

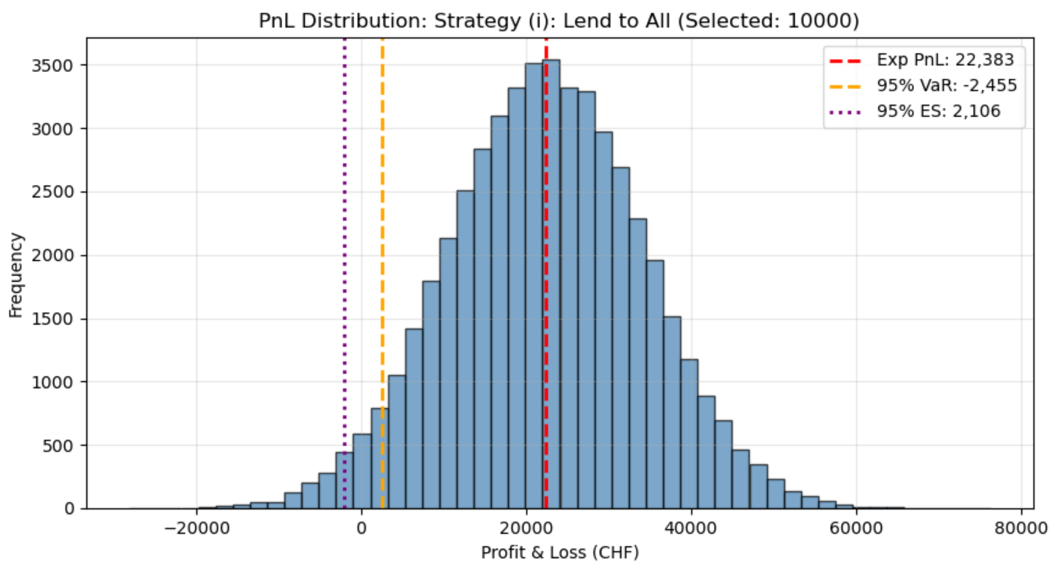

Figure 2: P&L distribution for Strategy (i): lend to all at 5.5%.

Table 2 summarizes the expected profit and tail risk for the three lending strategies. Recall that both VaR and ES are defined in terms of losses, so negative values indicate that even in the worst scenarios the portfolio still generates a gain.

- **Strategy (i) – Lend to all at 5.5%:** The expected profit is CHF 22,383. The histogram shows P&L ranging from roughly -20,000 to 6,000 CHF. The 95% VaR of -2,455 CHF and 95% ES of 2,106 CHF indicate that even in the 5% worst-case scenarios, losses are negligible or even gains, making this a low-risk strategy in terms of potential losses.

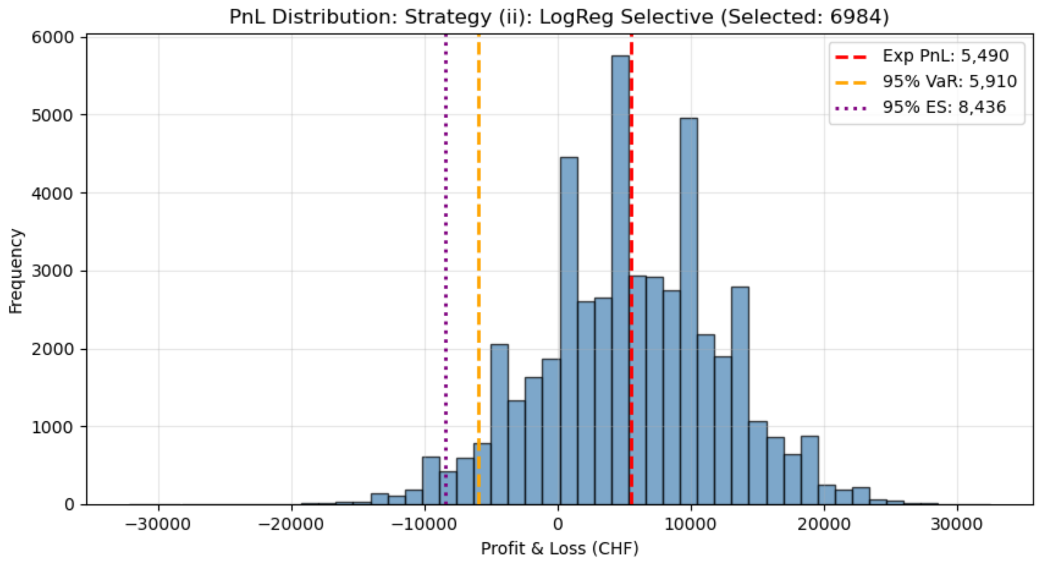

Figure 3: P&L distribution for Strategy (ii): logistic regression screening (threshold 95%).

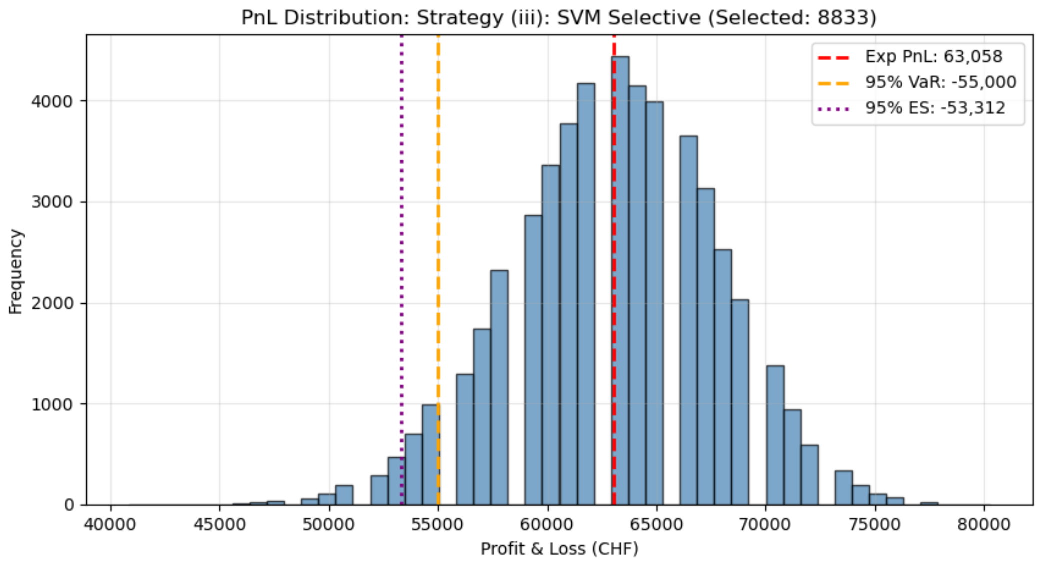

Figure 4: P&L distribution for Strategy (iii): SVM screening (threshold 95%).

Table 2: Profitability and risk metrics for the three lending strategies.

Strategy	Applicants	Exp. PnL (CHF)	$\mathrm{VaR}_{95\%}$ (CHF)	$\mathrm{ES}_{95\%}$ (CHF)
Lend to All (5.5%)	10000	22,383.19	-2,455.00	2,106.14
Logistic Regression (1%)	6984	5,489.51	5,910.00	8,435.51
SVM Selective (1%)	8833	63,057.76	-55,000.00	-53,312.36

- **Strategy (ii) – Logistic regression selective at 1%:** The expected profit is CHF 5,489, lower than the first strategy. P&L ranges approximately from -20,000 to 30,000 CHF. The 95% VaR of 5,910 CHF and 95% ES of 8,436 CHF indicate that the worst-case 5% of scenarios can incur significant losses. While strict screening reduces the number of loans and potential gains, it introduces a non-negligible tail risk.
- **Strategy (iii) - SVM selective at 1%:** This strategy yields the highest expected profit, CHF 63,058, with P&L between roughly 45,000 and 80,000 CHF. The 95% VaR of -55,000 CHF and ES of -53,312 CHF indicate that even in the worst-case 5% of scenarios, there are no losses; instead, large gains are realized. SVM screening effectively selects high-probability repayees, maximizing expected profit while maintaining low tail risk.

Overall, these strategies illustrate the trade-off between expected profit and tail risk. Lending to all provides moderate expected profit with minimal tail losses, logistic regression screening reduces expected profit but introduces significant potential losses in adverse scenarios, and SVM screening maximizes expected profit with highly favorable tail outcomes under the current simulation setup.

A Code Appendix

```
# %% [markdown]
# # Project 2: Credit Risk and Statistical Learning

# %% [markdown]
# **Names of all group members:**
# - Matthias Wyss (matthias.wyss@epfl.ch)
# - William Jallot (william.jallot@epfl.ch)
# - Antoine Garin (antoine.garin@epfl.ch)
#
#
# ---
#
# All code below is only suggestive and you may as well use different approaches.

# %% [markdown]
# ## Exercise 1 - Feature Generation

# %%
# Exercise 1.
import numpy as np
import os
np.random.seed(0) # for reproducibility

# simulate explanatory variables x
m, n = 20000, 10000 # training and test sizes
total = m + n

# x1: age (18-80)
x1 = np.random.uniform(18, 80, size=total)

# x2: monthly income in kCHF (1-15)
x2 = np.random.uniform(1, 15, size=total)

# x3: employment status (0 = salaried, 1 = self-employed)
x3 = np.random.choice([0, 1], size=total, p=[0.9, 0.1])

# stack into a feature matrix
X = np.column_stack((x1, x2, x3))
```

```
# a) calculate empirical means and standard deviations over training data
# Select first m samples as training data
X_train = X[:m]

means = X_train.mean(axis=0)

# We use ddof=1 for sample std deviation
stds = X_train.std(axis=0, ddof=1)

print("Empirical␣means␣(training␣data):", means)
print("Empirical␣stds␣␣(training␣data):", stds)


# b) Suggest other variables that would realistically be relevant in credit scoring.
# (you do not have to implement those of course, just explain your answer in writing)
"""
Other␣variables␣that␣could␣be␣relevant␣for␣credit␣scoring␣include:
-␣Credit␣history:␣past␣defaults,␣number␣of␣open␣loans,␣payment␣history.
-␣Debt-to-income␣ratio:␣proportion␣of␣income␣already␣committed␣to␣debt␣payments.
-␣Employment␣stability:␣length␣of␣current␣job,␣number␣of␣job␣changes.
-␣Marital␣status␣/␣dependents:␣may␣affect␣financial␣obligations.
-␣Education␣level:␣can␣correlate␣with␣income␣stability.
-␣Age␣brackets␣or␣life␣stage:␣young␣vs.␣near␣retirement␣may␣carry␣different␣risk.
-␣Housing␣situation:␣renter,␣owner,␣mortgage␣payments.
-␣Other␣financial␣indicators:␣savings,␣assets,␣or␣investments.
These␣features␣help␣better␣capture␣the␣borrower's␣ability␣and␣likelihood␣to␣repay.
"""

# %% [markdown]
# ## Exercise 2 - Default Model and Data Generation

# %%
# Exercise 2.
# Building the datasets:
np.random.seed(1)
sigmoid = lambda x: 1. / (1. + np.exp(-x))

# generate uniform random variables for thresholds
xi = np.random.uniform(0, 1, size=total)

# --- build the first dataset ---
# repayment probability function p1
p1 = lambda x1, x2, x3: sigmoid(13.3 - 0.33*x1 + 3.5*x2 - 3*x3)

# labels: 1 = repayment, 0 = default
y1 = lambda xi, x1, x2, x3: (xi <= p1(x1, x2, x3)).astype(int)

# --- build the second dataset ---
# repayment probability function p2
# Here we use a or logic instead of + because the two events are mutually exclusive
# (age below 25 or above 75)
p2 = lambda x1, x2, x3: sigmoid(5 - 10*((x1 < 25) | (x1 > 75)) + 1.1*x2 - x3)

y2 = lambda xi, x1, x2, x3: (xi <= p2(x1, x2, x3)).astype(int)

# %%
# Exercise 2. a)
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss
# "model = LogisticRegression().fit(X_data, Y_data)" fits a model
# "pred_X = model.predict_proba(X)" evaluates the model
# (note that it outputs both P(Y=0|X) and P(Y=1|X))
# "log_loss(Y, pred_X)" evaluates the negative conditional log likelihood (also called cross
    -entropy loss)

# Evaluate labels using the lambda functions
y1_vals = y1(xi, x1, x2, x3)
y2_vals = y2(xi, x1, x2, x3)

# Split train and test
X_train, X_test = X[:m], X[m:]
y1_train, y1_test = y1_vals[:m], y1_vals[m:]
y2_train, y2_test = y2_vals[:m], y2_vals[m:]

# --- Logistic Regression for dataset 1 ---
logreg1 = LogisticRegression().fit(X_train, y1_train)
pred1_train_logreg = logreg1.predict_proba(X_train)
pred1_test_logreg = logreg1.predict_proba(X_test)

loss1_train_logreg = log_loss(y1_train, pred1_train_logreg)
loss1_test_logreg = log_loss(y1_test, pred1_test_logreg)

print("Dataset␣1␣-␣Logistic␣Regression:")
print("Train␣cross-entropy␣loss:", f"{loss1_train_logreg:.4f}")
print("Test␣cross-entropy␣loss:␣", f"{loss1_test_logreg:.4f}")

# --- Logistic Regression for dataset 2 ---
logreg2 = LogisticRegression().fit(X_train, y2_train)

pred2_train_logreg = logreg2.predict_proba(X_train)
pred2_test_logreg = logreg2.predict_proba(X_test)

loss2_train_logreg = log_loss(y2_train, pred2_train_logreg)
loss2_test_logreg = log_loss(y2_test, pred2_test_logreg)

print("\nDataset␣2␣-␣Logistic␣Regression:")
print("Train␣cross-entropy␣loss:", f"{loss2_train_logreg:.4f}")
print("Test␣cross-entropy␣loss:␣", f"{loss2_test_logreg:.4f}")

# %%
# Exercise 2.b)
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import log_loss
# Calculate normalized data

# "model = SVC(kernel='rbf', gamma=GAMMA, C=C, probability=True)" creates
# a model with kernel exp(-GAMMA \|x-x'\|_2^2) and regul. parameter C (note the relation
    between C and the parameter lambda).
# "probability=True" enables the option "model.predict_proba(X)" to predict probabilities
    from the regression function \hat{f}^{svm}.
# "model.fit(X, Y)" optimizes the model parameters (using hinge loss)

# Fit the models for both datasets (this can take up to 60 seconds with SVC)

# "model.predict_proba(X)" predicts probabilities from features (note that it outputs both P
    (Y=0|X) and P(Y=1|X))

# Calculate cross-entropy loss on both datasets for train and test
# --- Standardize features using training set statistics ---
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# regularization parameter for SVM
C = 0.2
GAMMA = 1/10

# --- SVM for dataset 1 ---
svm1 = SVC(kernel='rbf', C=C, gamma=GAMMA, probability=True).fit(X_train_scaled, y1_train)

# Collect probabilities for dataset 1
pred1_train_svm = svm1.predict_proba(X_train_scaled)
pred1_test_svm = svm1.predict_proba(X_test_scaled)

loss1_train_svm = log_loss(y1_train, pred1_train_svm)
loss1_test_svm = log_loss(y1_test, pred1_test_svm)

print("Dataset␣1␣-␣SVM␣(RBF␣kernel):")
print("Train␣cross-entropy␣loss:", f"{loss1_train_svm:.4f}")
print("Test␣cross-entropy␣loss:␣", f"{loss1_test_svm:.4f}")

# --- SVM for dataset 2 ---
svm2 = SVC(kernel='rbf', C=C, gamma=GAMMA, probability=True).fit(X_train_scaled, y2_train)

# Collect probabilities for dataset 2
pred2_train_svm = svm2.predict_proba(X_train_scaled)
pred2_test_svm = svm2.predict_proba(X_test_scaled)

loss2_train_svm = log_loss(y2_train, pred2_train_svm)
loss2_test_svm = log_loss(y2_test, pred2_test_svm)

print("\nDataset␣2␣-␣SVM␣(RBF␣kernel):")
print("Train␣cross-entropy␣loss:", f"{loss2_train_svm:.4f}")
print("Test␣cross-entropy␣loss:␣", f"{loss2_test_svm:.4f}")

# %%
# Exercise 2.c)
import matplotlib.pyplot as plt
# To calculate the curves, it is fine to take 100 threshold values c, i.e.,
ths = np.linspace(0, 1, 100)

# To approximately calculate the AUC, it is fine to simply use Riemann sums.
# This means, if you have 100 (a_i, b_i) pairs for the curves, a_1 <= a_2 <= ...
# then you may simply use the sum
# sum_{i=1}^99 (b_i + b_{i+1})/2 * (a_{i+1}-a_i)
# as the approximation of the integral (or AUC)


# first data set & logistic regression:
# (the code should be reusable for all cases, only exchanging datasets and predicted
    probabilities depending on the model)

# Compute and plot the ROC and AUC cruves


# second data set & logistic regression:


# first data set and SVM:
```

```
# second data set and SVM:

np.random.seed(0)
# second data set and SVM:

def compute_roc_auc(y_true, y_pred_probs, thresholds):
    """
␣␣␣␣Compute␣ROC␣curve␣points␣(FPR,␣TPR)␣and␣approximate␣AUC␣using␣Riemann␣sum.

␣␣␣␣Parameters:
␣␣␣␣␣␣␣␣y_true:␣array␣of␣true␣binary␣labels␣(0␣or␣1)
␣␣␣␣␣␣␣␣y_pred_probs:␣array␣of␣predicted␣probabilities␣for␣Y=1
␣␣␣␣␣␣␣␣thresholds:␣array␣of␣threshold␣values␣to␣evaluate

␣␣␣␣Returns:
␣␣␣␣␣␣␣␣fpr:␣array␣of␣false␣positive␣rates
␣␣␣␣␣␣␣␣tpr:␣array␣of␣true␣positive␣rates
␣␣␣␣␣␣␣␣auc:␣approximate␣AUC
␣␣␣␣"""
    tpr = []
    fpr = []

    P = np.sum(y_true == 1)
    N = np.sum(y_true == 0)

    for c in thresholds:
        y_pred = (y_pred_probs >= c).astype(int)
        TP = np.sum((y_pred == 1) & (y_true == 1))
        FP = np.sum((y_pred == 1) & (y_true == 0))
        tpr.append(TP / P)
        fpr.append(FP / N)

    tpr = np.array(tpr)
    fpr = np.array(fpr)

    # Sort by FPR (ascending) before applying trapezoidal rule
    idx = np.argsort(fpr)
    fpr = fpr[idx]
    tpr = tpr[idx]

    # Riemann sum (trapezoidal) AUC
    auc = np.sum((tpr[:-1] + tpr[1:]) / 2 * (fpr[1:] - fpr[:-1]))

    return fpr, tpr, auc

# Plot all 4 cases:
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
axes = axes.ravel() # flatten to a 1D array for easy looping

cases = [
    ("Dataset␣1␣-␣LogReg", y1_test, pred1_test_logreg[:,1]),
    ("Dataset␣1␣-␣SVM",  y1_test, pred1_test_svm[:,1]),
    ("Dataset␣2␣-␣LogReg", y2_test, pred2_test_logreg[:,1]),
    ("Dataset␣2␣-␣SVM",  y2_test, pred2_test_svm[:,1]),
]

for ax, (title, y_true, preds) in zip(axes, cases):
    fpr, tpr, auc = compute_roc_auc(y_true, preds, ths)
    # ROC curve
    ax.plot(fpr, tpr, label=f"AUC={auc:.3f}")

    # Random classifier line
    ax.plot([0, 1], [0, 1], "k--")

    # Labels and style
    ax.set_title(title)
    ax.set_xlabel("False␣Positive␣Rate")
    ax.set_ylabel("True␣Positive␣Rate")
    ax.grid(True)
    ax.legend()

plt.tight_layout()

os.makedirs("outputs", exist_ok=True)
plt.savefig(os.path.join("outputs", "roc_subplots.png"))
plt.show()

# %% [markdown]
# ## Exercise 3 - Lending Strategies

# %%
# Exercise 3.

# Set model parameters and define matrix D
n_scenarios = 50000
loan_amount = 1000

x1_test = X_test[:, 0]
x2_test = X_test[:, 1]
x3_test = X_test[:, 2]

xi_test = np.random.uniform(0, 1, (len(X_test), n_scenarios))

true_probs_test = p2(x1_test, x2_test, x3_test)

print(f"Generating␣simulation␣matrix␣D␣({len(X_test)}␣applicants␣x␣{n_scenarios}␣scenarios)
    ...")
# Draw independent random variables xi ~ Unif(0,1)
# We generate the boolean matrix D directly:
# D_ik = 1 if xi <= p2(x), else 0
D = (xi_test <= true_probs_test[:, None]).astype(int)
print("Matrix␣D␣generated.\n")

# %%
def evaluate_strategy(strategy_name, selection_mask, interest_rate, D_matrix):
    """
␣␣␣␣Helper␣function␣to␣calculate␣PnL,␣plot␣histogram,␣and␣compute␣metrics.
␣␣␣␣"""
    # Filter the D matrix for only the selected applicants
    # resulting shape: (n_selected, n_scenarios)
    D_selected = D_matrix[selection_mask, :]

    n_selected = D_selected.shape[0]

    if n_selected == 0:
        print(f"---␣{strategy_name}␣---")
        print("No␣applicants␣selected.␣PnL␣is␣0.")
        return
    # Calculate PnL for each scenario
    # Profit = (Repayments * Loan * Interest) - (Defaults * Loan)
    # Repayment count per scenario (sum columns)
    repayments = np.sum(D_selected, axis=0) # shape: (n_scenarios,)
    defaults = n_selected - repayments

    # PnL Vector
    pnl = (repayments * loan_amount * interest_rate) - (defaults * loan_amount)

    # Expected Profit & Loss
    expected_pnl = np.mean(pnl)

    # 95% VaR (Value at Risk)
    # We take the 5th percentile of the PnL distribution.
    loss = -pnl
    var_95 = np.percentile(loss, 95)

    # 95% ES (Expected Shortfall)
    # The average of the PnL values that fall below the VaR threshold.
    es_95 = loss[loss >= var_95].mean()

    # Plotting
    plt.figure(figsize=(10, 5))
    plt.hist(pnl, bins=50, alpha=0.7, color='steelblue', edgecolor='black')
    plt.axvline(expected_pnl, color='red', linestyle='--', linewidth=2, label=f'Exp␣PnL:␣{
        expected_pnl:,.0f}')
    plt.axvline(-var_95, color='orange', linestyle='--', linewidth=2, label=f'95%␣VaR:␣{
        var_95:,.0f}')
    plt.axvline(-es_95, color='purple', linestyle=':', linewidth=2, label=f'95%␣ES:␣{es_95
        :,.0f}')

    plt.title(f"PnL␣Distribution:␣{strategy_name}␣(Selected:␣{n_selected})")
    plt.xlabel("Profit␣&␣Loss␣(CHF)")
    plt.ylabel("Frequency")
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.savefig(os.path.join("outputs", f"pnl_{strategy_name.replace('␣',␣'_').lower()}.png"
        ))
    plt.show()

    # Print results
    print(f"---␣{strategy_name}␣---")
    print(f"Applicants␣Selected:␣{n_selected}␣over␣{len(X_test)}")
    print(f"Interest␣Rate:␣␣␣␣␣␣␣{interest_rate*100}%")
    print(f"Expected␣PnL:␣␣␣␣␣␣␣␣CHF␣{expected_pnl:,.2f}")
    print(f"95%␣VaR:␣␣␣␣␣␣␣␣␣␣␣␣␣CHF␣{var_95:,.2f}")
    print(f"95%␣ES:␣␣␣␣␣␣␣␣␣␣␣␣␣␣CHF␣{es_95:,.2f}")
    print("-" * 30 + "\n")

# %%
# Scenario 1: Lend to everyone at 5.5% interest
# Define Portfolio and possible outcomes for this portfolio using matrix D
# Plot the histogram of profits and losses
# Calculate expected profit and losses, compute 95%-VaR and 95%-ES

mask_all = np.ones(len(X_test), dtype=bool)
evaluate_strategy("Strategy␣(i):␣Lend␣to␣All", mask_all, 0.055, D)

# %%
# Scenario 2: Lend selectively (LogReg >= 95%) at 1% interest
# Define Portfolio and possible outcomes using the matrix D and the predicted default
```

```
    probabilities from the logistic regression model
# Plot the histogram of profits and losses
# Calculate expected profit and losses, compute 95%-VaR and 95%-ES
mask_logreg = pred2_test_logreg[:, 1] >= 0.95
evaluate_strategy("Strategy␣(ii):␣LogReg␣Selective", mask_logreg, 0.01, D)

# %%
# Scenario 3: Lend selectively (SVM >= 95%) at 1% interest
# Define Portfolio and possible outcomes using the matrix D and the predicted default
    probabilities from the SVM model
# Plot the histogram of profits and losses
# Calculate expected profit and losses, compute 95%-VaR and 95%-ES
mask_svm = pred2_test_svm[:, 1] >= 0.95
evaluate_strategy("Strategy␣(iii):␣SVM␣Selective", mask_svm, 0.01, D)

# %%
```