Certificate in Credit Risk Analytics in Python

Credit Scoring Models

Credit scoring is the quantitative process of estimating the likelihood that a borrower will default on a loan. In the context of a Certificate in Credit Risk Analytics in Python, the terminology surrounding credit scoring models is extensive and forms the foundation for building, validating, and deploying robust predictive tools. The following exposition defines the most important terms, illustrates their practical application with Python snippets, and discusses common challenges that analysts encounter.

Probability of Default (PD) represents the estimated chance that a borrower will fail to meet its debt obligations within a specified time horizon, typically one year. PD is expressed as a decimal between 0 and 1, or as a percentage. In a logistic regression model, the PD is derived from the logistic function:

```python
Import numpy as np
Logit = np.Dot(X, beta) + intercept
Pd = 1 / (1 + np.Exp(-logit))
```

The output pd is the predicted probability of default for each observation. PD is the cornerstone of many regulatory frameworks, such as Basel II, where it directly influences capital requirements.

Loss Given Default (LGD) quantifies the proportion of exposure that is not recovered after default. LGD is also a decimal between 0 and 1, often estimated using historical recovery rates. In a simple portfolio model, the expected loss (EL) can be expressed as

```python
EL = pd * lgd * ead
```

Where ead is Exposure at Default. Accurate LGD estimation requires detailed recovery data, and practitioners often employ survival analysis or Tobit models for this purpose.

Exposure at Default (EAD) measures the amount outstanding at the time of default. For revolving credit lines, EAD may be approximated by the credit limit multiplied by a utilization factor, for example 0.5 For a 50 % utilization assumption. In Python, one might compute EAD as

```python
Df['ead'] = df['credit_limit'] * df['utilization_rate']
```

EAD is a key input to the Expected Loss calculation and influences risk-adjusted pricing decisions.

Logistic Regression is the most widely used statistical technique for binary classification in credit scoring. It models the log-odds of default as a linear combination of predictor variables. The model coefficients can be interpreted as odds ratios, providing intuitive insight into variable impact. A typical scikit-learn implementation looks like

```python
From sklearn.Linear_model import LogisticRegression
Model = LogisticRegression(max_iter=1000, penalty='l2', C=1.0)
Model.Fit(X_train, y_train)
```

The coefficients attribute contains the estimated betas, and the intercept_ attribute holds the constant term.

Weight of Evidence (WoE) is a transformation that converts categorical or binned numeric variables into a continuous scale reflecting the predictive power of each bin. WoE for a bin is calculated as

```python
Import numpy as np
Woe = np.Log((good / total_good) / (bad / total_bad))
```

Where “good” and “bad” denote the number of non-default and default observations in the bin, respectively. WoE is particularly valuable because it yields monotonic relationships with the target, facilitating the use of linear models.

Information Value (IV) measures the overall predictive strength of a variable based on its WoE distribution. IV is computed as

```python
Iv = ((good / total_good) - (bad / total_bad)) * woe
```

Variables with IV > 0.3 Are generally considered strong predictors, while those with IV Feature Engineering encompasses the creation, transformation, and selection of variables that capture borrower behavior. Common techniques include:

- Binning continuous variables into deciles or custom intervals.
- One-Hot Encoding for nominal categorical variables.
- Target Encoding which replaces categories with the mean target value, often regularized to avoid leakage.
- Interaction Terms that multiply two variables to capture joint effects.
- Polynomial Features for modeling non-linear relationships.

In Python, the category_encoders library provides a convenient TargetEncoder:

```python
From category_encoders import TargetEncoder
Enc = TargetEncoder(cols=['occupation'])
X_train_enc = enc.Fit_transform(X_train, y_train)
```

Multicollinearity occurs when predictor variables are highly correlated, inflating variance of coefficient estimates. The variance inflation factor (VIF) is a diagnostic metric; values above 5 or 10 often signal problematic collinearity. A quick VIF calculation can be performed with statsmodels:

```python
From statsmodels.Stats.Outliers_influence import variance_inflation_factor
Vif_data = pd.DataFrame()
Vif_data['feature'] = X.Columns
Vif_data['VIF'] = [variance_inflation_factor(X.Values, i) for i in range(X.Shape[1])]
```

If VIF is high, one may drop or combine variables, or apply dimensionality reduction techniques such as Principal Component Analysis (PCA).

Regularization mitigates overfitting by penalizing large coefficient values. L1 regularization (Lasso) forces some coefficients to zero, effectively performing variable selection, while L2 regularization (Ridge) shrinks coefficients towards zero but retains all variables. In scikit-learn, the C hyperparameter controls regularization strength; a smaller C implies stronger regularization.

Cross-Validation is a resampling strategy to assess model performance on unseen data. The most common form is k-fold cross-validation, where the dataset is split into k subsets, each serving once as a validation set while the remaining k−1 subsets form the training data. Stratified k-fold ensures that the proportion of defaults is preserved in each fold, which is crucial for imbalanced credit datasets:

```python
From sklearn.Model_selection import StratifiedKFold
Skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
For train_idx, val_idx in skf.Split(X, y):
    X_tr, X_val = X.Iloc[train_idx], X.Iloc[val_idx]
    Y_tr, y_val = y.Iloc[train_idx], y.Iloc[val_idx]
    # fit model on X_tr, y_tr and evaluate on X_val, y_val
```

Receiver Operating Characteristic (ROC) Curve plots the true positive rate (sensitivity) against the false

positive rate (1−specificity) across different classification thresholds. The area under the ROC curve (AUC) summarizes the model's discriminative ability; an AUC of 0.5 Indicates random guessing, while 1.0 Denotes perfect separation. In Python, ROC and AUC can be obtained via scikit-learn:

```python
From sklearn.Metrics import roc_curve, roc_auc_score
Fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
Auc = roc_auc_score(y_test, y_pred_proba)
```

Gini Coefficient is a linear transformation of AUC: Gini = 2 × AUC−1. It is frequently reported in credit risk literature and regulatory submissions.

Kolmogorov-Smirnov (KS) Statistic measures the maximum distance between the cumulative distribution functions of the default and non-default populations. A KS value above 0.4 Is generally considered strong. The KS can be computed as:

```python
Ks = max(abs(np.Cumsum(y_test==0)/np.Sum(y_test==0) - np.Cumsum(y_test==1)/np.Sum(y_test==1)))
```

Calibration assesses whether predicted probabilities align with observed default rates. A well-calibrated model will have a calibration curve that closely follows the diagonal. Calibration can be visualized using the calibration_curve function:

```python
From sklearn.Calibration import calibration_curve
Prob_true, prob_pred = calibration_curve(y_test, y_pred_proba, n_bins=10)
```

If calibration is poor, techniques such as Platt scaling or isotonic regression can be applied to adjust the probability outputs.

Scorecard is a tabular representation that translates model outputs into a point system used by business users for decision making. The scorecard construction process typically involves:

1. Converting logistic regression coefficients to points using a scaling factor. 2. Assigning points to each variable bin based on WoE values. 3. Adding a base score corresponding to the intercept.

A simple score calculation in Python might look like:

```python
Base_score = 600
```

```
Pdo = 20  # points to double odds
factor = pdo / np.Log(2)
Offset = base_score - factor * np.Log(odds)
Score = offset + factor * np.Log(odds)
```

Where odds = pd/(1-pd). The resulting score is intuitive for underwriters and can be integrated into credit policy rules.

Decision Tree models split the data recursively based on variable thresholds, creating a hierarchy of rules. Trees are easy to interpret but prone to overfitting. In scikit-learn:

```python
From sklearn.Tree import DecisionTreeClassifier
Tree = DecisionTreeClassifier(max_depth=5, min_samples_leaf=100, class_weight='balanced')
Tree.Fit(X_train, y_train)
```

The max_depth and min_samples_leaf parameters control complexity, while class_weight='balanced' addresses class imbalance by weighting the minority class more heavily.

Random Forest is an ensemble of decision trees built on bootstrap samples with random feature selection at each split. This reduces variance and improves generalization. A typical implementation:

```python
From sklearn.Ensemble import RandomForestClassifier
Rf = RandomForestClassifier(n_estimators=200, max_features='sqrt', n_jobs=-1, random_state=42)
Rf.Fit(X_train, y_train)
```

Feature importance can be extracted via rf.Feature_importances_, providing a global view of variable relevance.

Gradient Boosting Machines (GBM) sequentially fit weak learners to the residuals of previous models, focusing on difficult cases. Popular libraries include XGBoost, LightGBM, and CatBoost. Example with XGBoost:

```python
Import xgboost as xgb
Dtrain = xgb.DMatrix(X_train, label=y_train)
Params = {'objective': 'Binary:Logistic', 'eval_metric': 'Auc', 'eta': 0.05, 'Max_depth': 4}
Model = xgb.Train(params, dtrain, num_boost_round=500, early_stopping_rounds=30,
```

```
        Evals=[(xgb.DMatrix(X_val, label=y_val), 'validation')])
```

GBM models typically achieve higher AUC than logistic regression but are less transparent, prompting the need for explainability tools.

SHAP Values (SHapley Additive exPlanations) provide a unified framework for interpreting complex models. SHAP assigns each feature a contribution to the prediction, satisfying fairness and consistency properties. In Python:

```python
Import shap
Explainer = shap.TreeExplainer(rf)
Shap_values = explainer.Shap_values(X_test)
Shap.Summary_plot(shap_values, X_test)
```

The summary plot visualizes the distribution of SHAP values across features, revealing both direction and magnitude of influence.

LIME (Local Interpretable Model-agnostic Explanations) approximates the model locally with a simple surrogate (e.G., Linear) to explain individual predictions. Although less theoretically rigorous than SHAP, LIME can be useful for quick, case-by-case insight.

Imbalanced Data is a pervasive challenge in credit scoring because defaults are rare events. Standard accuracy metrics become misleading; instead, focus on AUC, precision-recall curves, or cost-sensitive measures. Techniques to address imbalance include:

- Resampling methods such as SMOTE (Synthetic Minority Over-sampling Technique) or random undersampling.
- Class weighting within the loss function, as shown earlier with class_weight='balanced'.
- Threshold optimization to minimize expected cost of misclassification, often expressed as:

```python
Cost_fp = 0.1  # cost of false positive (rejecting a good borrower)
Cost_fn = 1.0  # cost of false negative (accepting a bad borrower)
Optimal_thresh = (cost_fp) / (cost_fp + cost_fn)
```

Choosing the threshold that minimizes the weighted error aligns the model with business objectives.

Confusion Matrix tabulates true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). From these counts, various performance metrics arise:

- Sensitivity (Recall) = TP / (TP + FN)
- Specificity = TN / (TN + FP)
- Precision = TP / (TP + FP)
- F1 Score = 2 × Precision × Recall / (Precision + Recall)

These metrics can be computed with scikit-learn's classification_report.

Reject Inference addresses the bias introduced by training only on accepted applications. Since rejected applicants lack observed outcomes, analysts infer their likely behavior using methods such as:

- Augmentation where a small proportion of rejected cases are manually reviewed and labeled.
- EM Algorithm (Expectation-Maximization) to iteratively estimate the missing labels.
- Parcelling which assigns default probabilities to rejects based on the distribution of accepted defaults.

Reject inference is essential for maintaining a realistic estimate of portfolio risk, especially when acceptance rates are high.

Regulatory Frameworks impose standards on model development, validation, and governance. Key regulations include:

- Basel II/III which requires banks to estimate PD, LGD, and EAD for capital adequacy.
- IFRS 9 which mandates forward-looking expected credit loss (ECL) modeling, integrating lifetime PD and discounting.
- CCAR/DFAST stress testing scenarios that evaluate model performance under adverse macro-economic conditions.

Compliance demands thorough documentation, independent validation, and periodic back-testing.

Back-Testing compares model forecasts against realized outcomes over a hold-out period. The back-test period should be sufficiently long to capture multiple credit cycles. Metrics such as cumulative default rate, PD calibration error, and capital adequacy ratios are examined. A simple back-test in Python might involve:

```python
Df_test['predicted_pd'] = model.Predict_proba(X_test)[:,1]
Df_test['default_observed'] = y_test
Pd_bins = pd.Qcut(df_test['predicted_pd'], q=10, duplicates='drop')
Observed = df_test.Groupby(pd_bins)['default_observed'].Mean()
Predicted = df_test.Groupby(pd_bins)['predicted_pd'].Mean()
Calibration_error = np.Abs(observed - predicted).Mean()
```

Concept Drift refers to changes in the underlying data distribution over time, which can degrade model performance. Monitoring drift involves tracking statistical distances (e.G., Population Stability Index)

between training and current data:

```python
Def psi(expected, actual, buckets=10):
    Eps = 1e-6
    Expected_counts, _ = np.Histogram(expected, bins=buckets)
    Actual_counts, _ = np.Histogram(actual, bins=buckets)
    Expected_perc = expected_counts / expected_counts.Sum()
    Actual_perc = actual_counts / actual_counts.Sum()
    Return np.Sum((expected_perc - actual_perc) * np.Log((expected_perc + eps) / (actual_perc + eps)))
```

If PSI exceeds a threshold (commonly 0.1), A model refresh may be warranted.

Model Refresh is the process of retraining or updating a model to incorporate new data and address drift. A typical refresh schedule for credit scoring models is annual, aligning with regulatory reporting cycles. Automation can be achieved using MLOps pipelines that trigger retraining when drift metrics exceed preset limits.

Model Deployment translates a trained model into a production environment where it scores new applications in real time or batch mode. Common deployment patterns include:

- REST API using Flask or FastAPI, exposing an endpoint that accepts JSON payloads and returns PD predictions.
- Batch Scoring where nightly jobs process a CSV of new applications and write results to a database.
- Embedded Scoring where the model is serialized (e.G., With joblib) and loaded directly into the underwriting system.

A minimal FastAPI service might be:

```python
From fastapi import FastAPI
Import joblib
App = FastAPI()
Model = joblib.Load('credit_model.Pkl')

@App.Post('/predict')
Def predict(data: Dict):
    Import pandas as pd
    Df = pd.DataFrame([data])
    Prob = model.Predict_proba(df)[:,1][0]
    Return {'pd': Prob}
```

```

Model Governance encompasses policies, documentation, and controls that ensure model integrity throughout its lifecycle. Core components include:

- Model Documentation describing purpose, data sources, assumptions, and performance metrics.
- Version Control for code (Git) and data (DVC or MLflow), enabling reproducibility.
- Validation Checklist covering data quality, statistical tests, and business review.
- Audit Trail capturing changes, approvals, and user access logs.

Effective governance mitigates model risk and satisfies supervisory expectations.

Data Leakage occurs when information that would not be available at scoring time is inadvertently used in model training, leading to overly optimistic performance. Common sources include:

- Using post-default variables (e.G., Recovery amount) as predictors.
- Incorporating future macro-economic indicators that are not known at application.
- Performing target encoding on the full dataset before splitting.

To prevent leakage, all preprocessing steps that rely on the target must be fit on the training set only and then applied to validation/test sets.

Missing Value Imputation is essential because credit datasets often contain gaps (e.G., Missing income information). Strategies range from simple mean/median imputation to more sophisticated model-based methods such as K-Nearest Neighbors or iterative imputation. The sklearn.Impute module provides quick tools:

```python
From sklearn.Impute import SimpleImputer, KNNImputer
Median_imp = SimpleImputer(strategy='median')
X_filled = median_imp.Fit_transform(X)
```

When using model-based imputation, ensure that the imputer is trained only on the training data to avoid leakage.

Outlier Detection identifies extreme values that may distort model estimates. Techniques include:

- Z-Score filtering (|z| > 3).
- Isolation Forest for multivariate outlier detection.
- Winsorization which caps extreme values at percentile thresholds.

In Python:
```

```python
From sklearn.Ensemble import IsolationForest
Iso = IsolationForest(contamination=0.01, Random_state=42)
Outliers = iso.Fit_predict(X)
X_clean = X[outliers == 1]
```

Outlier handling should be guided by business logic; sometimes extreme values carry genuine risk information and should be retained.

Hyperparameter Tuning optimizes model settings to achieve the best predictive performance. Common approaches:

- Grid Search exhaustively evaluates a predefined parameter grid.
- Random Search samples a fixed number of random combinations, often more efficient for large spaces.
- Bayesian Optimization (e.G., Using Optuna) models the performance surface and selects promising configurations.

Example of a randomized search for XGBoost hyperparameters:

```python
From sklearn.Model_selection import RandomizedSearchCV
Param_dist = {
    'Max_depth': [3, 5, 7, 9],
    'Learning_rate': [0.01, 0.05, 0.1],
    'N_estimators': [100, 300, 500],
    'Subsample': [0.6, 0.8, 1.0],
    'Colsample_bytree': [0.6, 0.8, 1.0]
}
Xgb_clf = xgb.XGBClassifier(objective='binary:Logistic', eval_metric='auc')
Rand_search = RandomizedSearchCV(xgb_clf, param_distributions=param_dist,
                                 N_iter=30, cv=3, scoring='roc_auc', random_state=42)
Rand_search.Fit(X_train, y_train)
```

The best estimator can then be stored for deployment.

Ensemble Methods combine multiple base models to improve accuracy and robustness. Popular ensemble strategies include:

- Bagging (e.G., Random Forest) which averages predictions across bootstrapped trees.
- Boosting (e.G., XGBoost, LightGBM) which sequentially focuses on residual errors.

- Stacking where predictions from several models become inputs to a meta-learner (often a logistic regression).

A simple stacking implementation:

```python
From sklearn.Linear_model import LogisticRegression
From sklearn.Ensemble import StackingClassifier
Estimators = [('rf', RandomForestClassifier(n_estimators=200)),
              ('Xgb', xgb.XGBClassifier(use_label_encoder=False))]
Stack = StackingClassifier(estimators=estimators, final_estimator=LogisticRegression())
Stack.Fit(X_train, y_train)
```

Ensembles often achieve higher AUC but increase complexity, necessitating careful documentation and interpretability analysis.

Performance Decay is the gradual decline in predictive power as the model ages. Monitoring decay involves tracking key metrics (AUC, KS, calibration error) on a rolling window of recent data. When decay exceeds predefined thresholds, a model review or rebuild is triggered.

Cost-Benefit Analysis evaluates the financial impact of model decisions. The expected profit for a loan can be expressed as:

```python
Expected_profit = (1 - pd) * interest_income - pd * (lgd * ead) - acquisition_cost
```

Optimizing the acceptance threshold to maximize expected profit aligns the model with the institution’s risk appetite. Sensitivity analysis can be performed by varying PD, LGD, and interest rates to assess robustness.

Risk Appetite defines the level of risk the organization is willing to accept in pursuit of its strategic objectives. It influences model thresholds, portfolio limits, and pricing. Translating risk appetite into quantitative terms often involves setting a target PD or capital requirement per segment.

Portfolio Segmentation groups borrowers by risk characteristics (e.G., Credit score bands, industry, geography) to enable differentiated pricing and limit setting. Segmentation can be performed using clustering algorithms (K-Means) or rule-based approaches derived from the scorecard.

Alternative Data supplements traditional credit bureau information with non-traditional sources such as utility payments, mobile phone usage, social media activity, and transaction histories. Incorporating alternative data can improve coverage for thin-file borrowers but raises privacy and fairness concerns. When using alternative data, ensure compliance with data protection regulations and conduct bias testing.

Bias and Fairness are critical considerations in credit scoring. Disparate impact analysis examines whether protected attributes (e.G., Race, gender) influence model decisions disproportionately. Techniques to mitigate bias include:

- Removing or masking protected attributes and any proxies.
- Applying fairness-aware algorithms (e.G., Adversarial debiasing).
- Re-weighting training samples to achieve demographic parity.

Python's fairlearn library offers tools for assessing and correcting bias:

```python
From fairlearn.Metrics import demographic_parity_difference
Dp_diff = demographic_parity_difference(y_test, y_pred, sensitive_features=df_test['gender'])
```

A low demographic parity difference indicates equitable treatment across groups.

Explainable AI (XAI) bridges the gap between complex models and business stakeholders. Besides SHAP and LIME, other XAI methods include:

- Partial Dependence Plots (PDP) which show the marginal effect of a single feature on the predicted PD.
- Individual Conditional Expectation (ICE) curves that display feature effects for individual observations.
- Counterfactual Explanations that suggest minimal changes to input variables needed to flip a decision.

These visualizations help underwriters understand model behavior and build trust.

Regulatory Validation requires independent review of model methodology, data quality, and performance. Validation steps typically involve:

1. Conceptual Review confirming that the model aligns with business objectives. 2. Data Review verifying source integrity, completeness, and relevance. 3. Statistical Review assessing discrimination, calibration, and stability. 4. Back-Testing comparing predicted versus actual outcomes over a hold-out period. 5. Governance Review ensuring documentation, approvals, and change management processes are in place.

Validation reports are often submitted to senior risk committees and external regulators.

Stress Testing evaluates model performance under adverse macro-economic scenarios (e.G., Recession, high unemployment). Stress variables are incorporated into PD estimation by extending the logistic model:

```python
Df['unemployment_rate'] = macro['unemployment_rate']
Logit = beta0 + beta1 * income + beta2 * unemployment_rate
Pd_stressed = 1 / (1 + np.Exp(-logit))
```

```

Stress testing helps institutions assess capital adequacy and strategic resilience.

Scenario Analysis complements stress testing by exploring “what-if” situations, such as changes in interest rates or regulatory policy. Scenario outputs are typically reported as projected default rates, loss distributions, and capital requirements.

Time Series Features capture temporal dynamics, such as payment history trends or macro-economic indicators lagged over months. Feature engineering may involve calculating rolling averages, differences, or exponential smoothing. In pandas:

```python
Df['payment_trend_3m'] = df['payment_amount'].Rolling(window=3).Mean()
Df['gdp_lag_6m'] = macro['gdp'].Shift(6)
```

Incorporating time series features can improve early warning capabilities.

Macro-Economic Variables (GDP growth, unemployment, interest rates) are often included in PD models to reflect systematic risk. When integrating macro data, align frequencies (monthly, quarterly) and ensure that only publicly available information up to the scoring date is used.

Model Risk Management (MRM) is a discipline that oversees the entire model lifecycle, from development to retirement. Core MRM activities include:

- Maintaining a model inventory with status, owner, and version.
- Conducting periodic reviews (e.G., Quarterly) of model performance.
- Enforcing segregation of duties between model developers, validators, and users.
- Implementing controls for data access, model changes, and deployment.

A robust MRM framework reduces operational risk and supports regulatory compliance.

Data Provenance tracks the origin and transformations applied to datasets. Using tools like DVC (Data Version Control) enables reproducible pipelines:

```bash
Dvc init
Dvc add raw_data.Csv
Git add raw_data.Csv.Dvc .Gitignore
Git commit -m "Add raw data with provenance"
```
```

Each model build can then be linked to a specific data snapshot, facilitating audits.

Ethical Considerations extend beyond regulatory compliance. Analysts must balance predictive accuracy with societal impact, ensuring that models do not exacerbate financial exclusion. Transparency, stakeholder communication, and continuous monitoring of fairness metrics are essential components of an ethical credit scoring practice.

Model Monitoring Dashboard provides real-time visibility into key performance indicators (KPIs) such as AUC, drift metrics, and volume of scored applications. Tools like Streamlit or Grafana can be used to build interactive dashboards:

```python
Import streamlit as st
St.Title('Credit Model Monitoring')
St.Metric('Current AUC', round(current_auc, 3))
St.Metric('PSI (Score)', round(psi_score, 3))
```

Alerts can be configured to notify risk managers when thresholds are breached.

Threshold Optimization determines the PD cut-off that balances acceptance rates, profitability, and risk tolerance. The optimal threshold can be found by maximizing a utility function:

```python
Def utility(thresh):
    Tp = ((y_test == 1) & (y_pred_proba >= thresh)).Sum()
    Fp = ((y_test == 0) & (y_pred_proba >= thresh)).Sum()
    Fn = ((y_test == 1) & (y_pred_proba optimal_thresh aligns model decisions with the institution’s strategic goals.

Business Rules Integration combines model scores with deterministic policies (e.G., Minimum income requirement, maximum loan-to-value). Rule engines can be implemented using simple if-else logic or more sophisticated rule-based systems like Drools. In Python:

```python
Def apply_rules(row):
    If row['income'] Cost of Misclassification quantifies the financial impact of incorrectly classifying borrowers. A false negative (accepting a defaulter) typically incurs higher loss than a false positive (rejecting a good borrower). By assigning explicit monetary values to each error type, the model can be trained with a cost-sensitive loss function:

```python
```

```
From sklearn.Utils import class_weight
Weights = class_weight.Compute_sample_weight({0: Cost_fp, 1: Cost_fn}, y_train)
Model.Fit(X_train, y_train, sample_weight=weights)
```

Cost-sensitive learning aligns the objective function with business objectives.

Precision-Recall Curve is especially informative for imbalanced datasets, emphasizing performance on the minority class. The area under the precision-recall curve (AUPRC) can be higher than AUC for rare events, making it a valuable complementary metric.

```python
From sklearn.Metrics import precision_recall_curve, average_precision_score
Precision, recall, thresholds = precision_recall_curve(y_test, y_pred_proba)
Ap = average_precision_score(y_test, y_pred_proba)
```

Monitoring both ROC-AUC and AUPRC provides a fuller picture of discriminative power.

Lift Chart visualizes how much better the model is at identifying defaults compared to random selection. Lift is calculated as the ratio of the cumulative default rate in a selected percentile to the overall default rate. A lift of 3 at the top 10 % indicates that the model captures three times more defaults than random sampling.

Profit Curve plots expected profit against acceptance rate, helping decision makers choose a cutoff that maximizes net benefit. The curve is derived by sorting applications by predicted PD and computing cumulative profit as the acceptance threshold moves down the ranking.

Model Explainability is increasingly demanded by regulators and internal stakeholders. Techniques such as SHAP not only provide global feature importance but also enable local explanations for individual decisions. For a loan application, a SHAP explanation might reveal that high credit utilization and recent delinquencies contributed positively to the predicted PD, while a long credit history reduced it.

Model Documentation Template typically includes sections on:

- Model purpose and scope.
- Data sources, preprocessing steps, and variable definitions.
- Modeling methodology, hyperparameters, and training procedure.
- Performance metrics (AUC, KS, calibration) on training, validation, and test sets.
- Validation results, including back-testing and stress testing outcomes.
- Governance and change management processes.
- Limitations, assumptions, and future improvement plans.

Having a standardized template accelerates review cycles and ensures consistency across model families.

Model Risk Assessment assigns a risk rating (e.G., Low, medium, high) based on factors such as model complexity, data quality, performance stability, and usage criticality. High-risk models may require more frequent validation, tighter controls, and senior management sign-off.

Regulatory Reporting often mandates submission of model parameters, validation results, and governance evidence. For Basel II/III, institutions must provide PD, LGD, and EAD estimates, along with model governance documentation, to supervisory authorities. Automated report generation using Jinja2 templates can streamline this process.

Data Quality Checks form the first line of defense against model risk. Typical checks include: