Day1

Day 1 – Detailed Understanding:

1. Project Overview – Credit Path AI

The Credit Path AI project focuses on predicting credit default risk — determining whether a borrower is likely to repay or default on their loan.
This prediction helps financial institutions minimize risk and make smarter lending decisions.

2. KPIs (Key Performance Indicators) of the Project

KPIs help evaluate the effectiveness of the predictive model. In the Credit Path AI project, the major KPIs include:

- **Accuracy** – Measures how often the model predicts correctly.
- **Precision** – The percentage of correctly predicted positive cases among all predicted positives.
- **Recall (Sensitivity)** – How well the model detects actual positives (defaulters).
- **F1 Score** – The harmonic mean of precision and recall.
- **AUC-ROC Score** – Measures how well the model distinguishes between classes (default vs. non-default).

These KPIs are used collectively to assess model performance because relying only on accuracy can be misleading when data is imbalanced.

3. Dataset Collection and Understanding

The Lending Club dataset was downloaded from Kaggle, and loaded into Python using pandas. After loading, we performed Viewing first few rows, Checking dataset size, Inspecting column types and missing values. This helps understand the structure of the data (number of features, data types, missing fields).

4. Exploratory Data Analysis (EDA)

EDA is the process of exploring and visualizing data to understand its patterns and relationships.
During EDA, we typically:

- Identify missing values.
- Check numerical distributions (using histograms, boxplots).

- Observe categorical variable frequencies (using value counts).
- Understand correlations among numerical columns (using .corr() or heatmaps).

5. Feature Engineering

Feature engineering improves the dataset so models can understand it better.

a. Handling Missing Values

Missing data can lead to inaccurate results. Techniques include:

- Replacing missing numerical values with mean or median.
- Filling categorical missing values with mode or “Unknown”.

b. Scaling Numerical Features

Since models like Logistic Regression are sensitive to value scales, we use StandardScaler to normalize features. This ensures all numerical columns have similar ranges, improving model stability.

c. Encoding Categorical Variables

Categorical variables are converted into numeric form using One-Hot Encoding.This prevents the model from misinterpreting categories as numeric order.

6. Training Base Model (Logistic Regression)

After preprocessing, a Logistic Regression model is trained to predict the loan status. This base model provides a benchmark before using advanced algorithms like XGBoost or LightGBM.

7. AUC ROC scores

ROC Curve (Receiver Operating Characteristic)

A graphical plot showing the trade-off between True Positive Rate (Recall) and False Positive Rate at different thresholds.

AUC (Area Under the ROC Curve)

- AUC represents the model’s ability to separate positive (default) and negative (non-default) classes.
- Higher AUC → better performance.

AUC Interpretation:

AUC Score	Model Performance
0.9 – 1.0	Excellent
0.8 – 0.9	Good
0.7 – 0.8	Fair
0.6 – 0.7	Poor
0.5	Random Guessing

Day2

Day 2 – Regression, Classification & Logistic Regression Concepts

Topics Covered

- Understanding Regression and its evaluation metrics.
- Understanding Classification and its evaluation metrics.
- Differences between regression and classification.
- Why Logistic Regression is used for classification tasks.
- Advantages and Limitations of Logistic Regression.
- Concept of Sigmoid Function and how it works.
- Maximum Likelihood Estimation (MLE) and the Likelihood Formula.

Detailed Understanding

1. Regression

Regression is a supervised learning technique used to predict a continuous numerical output (e.g., price, income, temperature).
The model finds the best-fitting line (or curve) that describes the relationship between input features (X) and the target variable (y).

Common Regression Algorithms:

- Linear Regression
- Ridge, Lasso Regression
- Polynomial Regression
- Decision Tree or Random Forest Regressors

Evaluation Metrics in Regression:

Metric	Description	Ideal Value
MAE (Mean Absolute Error)	Average absolute difference between actual and predicted values.	Closer to 0

MSE (Mean Squared Error)	Penalizes larger errors by squaring the difference.	Closer to 0
RMSE (Root Mean Squared Error)	Square root of MSE; interpretable in same units as target.	Closer to 0
R^2 (Coefficient of Determination)	Shows how much variance in data is explained by the model.	0–1 (Higher is better)

2. Classification

Classification is a supervised learning technique used to predict a categorical output — e.g., whether a loan will default or not, or whether an email is spam or not.

Common Classification Algorithms:

- Logistic Regression
- Decision Trees
- Random Forest
- XGBoost, LightGBM
- Support Vector Machines

Evaluation Metrics in Classification:

Metric	Description
Accuracy	Percentage of correctly classified samples.
Precision	Out of all predicted positives, how many are actually positive.
Recall (Sensitivity)	Out of all actual positives, how many are correctly predicted.
F1 Score	Harmonic mean of precision and recall (balances both).
AUC-ROC	Measures how well the model separates classes across thresholds.
Confusion Matrix	Shows counts of TP, TN, FP, FN — gives a detailed performance summary.

3. Differences between Regression and Classification

Aspect	Regression	Classification

Output Type	Continuous numerical values	Discrete categories or labels
Examples	Predicting loan amount, house price	Predicting loan default (Yes/No)
Evaluation Metrics	MAE, MSE, RMSE, R^2	Accuracy, F1, Precision, Recall, AUC
Goal	Minimize prediction error	Maximize correct classification
Algorithms	Linear, Ridge, Lasso Regression	Logistic Regression, Trees, SVM, etc.

4. Why We Use Logistic Regression for Classification

Although it has “regression” in its name, Logistic Regression is used for classification because:

- It predicts probabilities (values between 0 and 1).
- These probabilities are then converted into class labels (e.g., default or not).
- It models the relationship between input variables and the probability of a specific class.

Advantages:

- Simple and interpretable.
- Works well for linearly separable data.
- Requires less computational power.
- Provides class probabilities, not just labels.

Limitations:

- Works poorly when data is non-linear.
- Assumes features are independent and have a linear relationship with the log-odds.
- Can struggle with multicollinearity or highly imbalanced data.

5. Sigmoid (Logistic) Function

The **Sigmoid Function** is the core of logistic regression.
It maps any real number into a range between **0 and 1**, making it ideal for probability prediction.

Formula:

σ(x) = 1 / (1 + e^(-x))

Behavior:

- For large positive x → output ≈ 1
- For large negative x → output ≈ 0
- For x = 0 → output = 0.5

In Logistic Regression:

1. A linear combination of inputs is calculated:
   z = b0 + b1*x1 + b2*x2 + ... + bn*xn
2. The sigmoid function converts z into probability:
   $P(y=1 \mid x)=1 / (1 + e^{(-z)})$
3. If $P>0.5P > 0.5P>0.5$, classify as 1; else classify as 0.

6. Maximum Likelihood Estimation (MLE)

MLE is a statistical method used to estimate model parameters (weights) that make the observed data most probable.

Concept:

- The model calculates the probability of the observed outputs given its parameters.
- The best parameters are those that maximize this likelihood.

Likelihood Function:

$$L(\theta) = \Pi [P(y_i \mid x_i; \theta)]$$

To simplify optimization, we use the log-likelihood:

$$\text{Log } L(\theta) = \Sigma [y_i * \log(p_i) + (1 - y_i) * \log(1 - p_i)]$$

The model then adjusts parameters (θ) to maximize this function.

Advantages of Boosting Models over Logistic Regression

1. Captures Non-Linear Relationships
    - Logistic Regression assumes a linear relationship between features and the log-odds of the outcome.
    - Boosting algorithms (like XGBoost and LightGBM) use decision trees, allowing them to model complex, non-linear patterns in data.
2. Automatically Handles Feature Interactions
    - Boosting learns feature combinations automatically through multiple tree splits.
    - Logistic Regression cannot capture interactions unless you manually add interaction terms.
3. Better Performance on Large and Complex Datasets
    - Boosting algorithms generally achieve higher accuracy and AUC scores, especially on datasets with many variables and non-linear relationships.
    - Logistic Regression performs well only on simple, linearly separable data.
4. Robustness to Outliers and Missing Values
    - Tree-based models are less sensitive to outliers and can handle missing data internally.
    - Logistic Regression is affected strongly by outliers and requires explicit preprocessing.
5. Built-in Regularization and Feature Importance
    - XGBoost and LightGBM include L1/L2 regularization and provide feature importance scores, helping interpret which features

impact predictions the most.

- Logistic Regression supports regularization but doesn't provide feature ranking naturally.

6. Improved Generalization (Lower Bias and Variance)

    - Boosting sequentially corrects errors from previous trees, reducing both bias and variance.

    - Logistic Regression can underfit if data patterns are complex.

Day3

Day 3 – Ensemble Learning and Boosting Algorithms

1. Decision Trees

A Decision Tree is a supervised learning algorithm used for both classification and regression tasks.
It splits the dataset into branches based on feature values to make decisions.

Key Concepts:

- Each node represents a feature.
- Each branch represents a decision rule.
- Each leaf node represents the final prediction.

Advantages:

- Easy to interpret and visualize.
- Works for both numerical and categorical data.
- Requires little preprocessing.

Disadvantages:

- Prone to overfitting (memorizing training data).
- Small data changes can drastically change the tree (high variance).

2. Random Forest

A Random Forest is an ensemble of many decision trees trained on different samples of the data.
It uses Bagging (Bootstrap Aggregation) to reduce overfitting and improve accuracy.

How It Works:

1. Random subsets of data and features are chosen for each tree.
2. Each tree makes a prediction.
3. Final output = majority vote (for classification) or average (for regression).

Advantages:

- Reduces variance and overfitting.
- Works well on large datasets.
- Provides feature importance.

Disadvantages:

- Less interpretable than a single tree.
- Computationally expensive for large data.

3. Bagging (Bootstrap Aggregation)

Bagging is a technique to improve model stability and accuracy by training multiple models in parallel on random subsets of data.

Steps:

1. Create multiple random samples (with replacement).
2. Train one model on each sample (e.g., a decision tree).
3. Combine predictions (average or majority vote).

Key Point:
Bagging decreases variance without increasing bias.

4. Boosting – Concept and Working

Boosting is an ensemble technique that combines weak learners (like shallow trees) sequentially to create a strong model.

How Boosting Works (Step-by-Step):

1. Start with a weak model (e.g., a small decision tree).
2. Identify the errors (residuals) from that model.
3. Train the next model to correct those errors.
4. Continue adding models until performance stops improving.

5. Combine all models' predictions (weighted sum) to form the final output.

Key Idea:
Each new model focuses on the mistakes made by previous models.

5. Gradient Boosting

Gradient Boosting is an advanced boosting method where new models are trained to minimize the loss function's gradient (error direction).

Loss Function in Gradient Boosting:
For binary classification (e.g., loan default), a common loss function is Log Loss:

```
Loss = - [ y * log(p) + (1 - y) * log(1 - p) ]
```

The algorithm tries to minimize this loss step by step using gradient descent.

6. Important Parameters of Gradient Boosting

Parameter	Description
n_estimators	Number of trees (iterations).
learning_rate	How much each tree contributes to the final model (smaller = slower but more accurate).
max_depth	Maximum depth of each tree.
subsample	Fraction of data used for training each tree (helps prevent overfitting).
min_samples_split / leaf	Minimum samples needed to split or form a leaf.

7. Gradient Descent – Optimization Backbone

Gradient Descent is an optimization algorithm used to minimize loss functions.
It updates model parameters by moving in the direction of the negative gradient.

Update Rule:

```
θ_new = θ_old - α * (∂Loss/∂θ)
```

where:

- θ = parameter being updated
- α = learning rate
- $\partial \text{Loss}/\partial \theta$ = gradient of loss function

The process repeats until the model converges to a minimum loss.

8. XGBoost (Extreme Gradient Boosting)

XGBoost is an optimized version of Gradient Boosting designed for speed and performance.

Features:

- Handles missing data automatically.
- Uses both L1 (Lasso) and L2 (Ridge) regularization.
- Parallel processing and tree pruning.
- Highly efficient for large datasets.

Advantages:

- High accuracy and generalization.
- In-built cross-validation.
- Feature importance ranking.

9. CatBoost (Categorical Boosting)

CatBoost is a gradient boosting library that automatically handles categorical variables without manual encoding.

Features:

- Handles categorical data internally using ordered statistics.
- Prevents overfitting using ordered boosting.
- Requires minimal preprocessing.

Advantages:

- Less preprocessing needed.
- Works well with categorical-heavy datasets.
- Fast and memory-efficient.

10. LightGBM (Light Gradient Boosting Machine)

LightGBM is a gradient boosting framework developed by Microsoft that uses leaf-wise growth instead of level-wise like XGBoost.

Features:

- Extremely fast and memory efficient.
- Supports large datasets.
- Can handle both categorical and numerical features.

Advantages:

- High speed and accuracy.
- Lower memory usage.
- Great performance with imbalanced data.

Difference from XGBoost:

Feature	XGBoost	LightGBM
Tree Growth	Level-wise	Leaf-wise
Speed	Moderate	Faster
Memory	Higher	Lower
Overfitting Risk	Lower	Slightly higher (if not tuned)

Day4

Differences Between XGBoost and LightGBM

- Both XGBoost and LIghtgbm are popular gradient boosting frameworks used for machine learning tasks, particularly for structured data.

Aspect	XGBoost	LightGBM
Growth Strategy	Level-wise (grows all leaves at each level)	Leaf-wise (splits the leaf with highest loss reduction)
Training Speed	Slower due to balanced tree growth	Much faster due to histogram-based and leaf-wise growth
Memory Usage	Higher memory usage	Lower memory usage (uses histogram binning)
Overfitting	Less prone to overfitting (balanced trees)	More prone to overfitting on small datasets
Accuracy	Very high accuracy with proper tuning	Often achieves slightly better accuracy with large datasets
Handling Categorical Features	Needs manual encoding (like one-hot encoding)	Can directly handle categorical features
Parallelism	Efficient parallelization but slower than LightGBM	More efficient parallel training
Implementation Complexity	Easier to interpret and visualize	Slightly harder to interpret due to deep, unbalanced trees
Best For	Small to medium-sized datasets	Large datasets with high feature dimensions
Tree Construction	Uses pre-sorted data or histogram-based (optional)	Always uses histogram-based algorithm

Day 3 – Understanding Boosting Models: XGBoost and LightGBM

1. XGBoost (Extreme Gradient Boosting)

Concept:
XGBoost is an optimized version of gradient boosting that builds decision trees sequentially. Each new tree corrects the errors made by the previous ones using gradient-based optimization.

2. How XGBoost Develops Trees

- XGBoost builds trees level-wise.
- Each level of the tree is fully expanded before moving to the next one.
- This makes the model more balanced and generalized, reducing overfitting.
- It uses both first-order (gradient) and second-order (hessian) derivatives to minimize the loss function faster and more accurately.

3. Leaf-Wise vs Level-Wise Growth

Growth Type	Used By	How It Works	Characteristics
Level-wise Growth	**XGBoost**	Expands all leaves at the same level together	Balanced tree, stable training, less overfitting
Leaf-wise Growth	**LightGBM**	Grows the leaf with the highest loss reduction	Faster, deeper trees, better accuracy but may overfit on small data

4. Advantages of XGBoost

- Handles missing values automatically.
- Supports regularization (L1 & L2) to avoid overfitting.
- Efficient parallel computation and cache optimization.
- Provides feature importance for interpretability.
- Can handle large datasets efficiently with the tree_method='hist' option.

5. LightGBM (Light Gradient Boosting Machine)

Concept:
LightGBM is a gradient boosting framework by Microsoft that uses leaf-wise growth and histogram-based algorithms to train much faster and use less memory.

6. Growth of LightGBM Tree

- LightGBM uses leaf-wise tree growth.
- At each step, it chooses the leaf with the maximum loss reduction to split.
- This leads to deeper trees with fewer overall splits, giving higher accuracy.
- However, on small datasets, it can overfit easily, so controlling parameters like max_depth is important.

7. Advantages of LightGBM

- Very fast training speed compared to XGBoost.
- Lower memory usage due to histogram binning.
- Handles large datasets efficiently.
- Supports categorical features directly.
- Better accuracy when tuned properly.

8. Hyperparameter Tuning (Key Parameters Used)

For XGBoost:

Parameter	Purpose
n_estimators	Number of boosting rounds (trees)
max_depth	Controls depth of each tree; prevents overfitting
learning_rate	Step size for each tree; smaller → more accurate
subsample	Fraction of data used per tree (reduces overfitting)
colsample_bytree	Fraction of features used per tree
gamma	Minimum loss reduction required to make a split

reg_alpha	L1 regularization (feature selection)
reg_lambda	L2 regularization (stabilization)

For LightGBM:

Parameter	Purpose
num_leaves	Controls number of leaf nodes per tree
max_depth	Limits tree depth to avoid overfitting
learning_rate	Controls contribution of each tree
n_estimators	Number of boosting iterations
subsample	Row sampling to prevent overfitting
colsample_bytree	Column sampling for diversity
min_data_in_leaf	Minimum samples per leaf; prevents deep overfit leaves
lambda_l1, lambda_l2	Regularization parameters

XGBoost Parameters

General Parameters

Parameter	Description
booster	Type of booster to use: gbtree, gblinear, or dart
verbosity	Controls the amount of messages printed (0–3)
nthread	Number of parallel threads used for training

Tree Booster Parameters (for gbtree and dart)

Parameter	Description
eta / learning_rate	Step size shrinkage to prevent overfitting (typical 0.01–0.3)
gamma	Minimum loss reduction required to make a further partition on a leaf node

max_depth	Maximum depth of a tree (controls model complexity)
min_child_weight	Minimum sum of instance weight (hessian) needed in a child
subsample	Fraction of observations (rows) used for training each tree
colsample_bytree	Fraction of columns used for each tree
colsample_bylevel	Fraction of columns used for each level
colsample_bynode	Fraction of columns used for each split
lambda / reg_lambda	L2 regularization term on weights
alpha / reg_alpha	L1 regularization term on weights
scale_pos_weight	Controls balance of positive and negative classes (useful for imbalanced data)
max_delta_step	Maximum step allowed for each tree's weight estimation
grow_policy	Controls tree growth: depthwise (default) or lossguide
max_leaves	Maximum number of leaves per tree when using lossguide growth

Learning Task Parameters

Parameter	Description
objective	Learning objective (e.g., binary:logistic, multi:softmax, reg:squarederror)
eval_metric	Evaluation metrics like auc, error, logloss, rmse, etc.
seed	Random number seed for reproducibility

DART Booster Specific

Parameter	Description
sample_type	Type of sampling algorithm (uniform, weighted)
normalize_type	Type of normalization (tree, forest)
rate_drop	Dropout rate for trees
skip_drop	Probability of skipping dropout

LightGBM Parameters

Core Parameters

Parameter	Description
boosting_type	Type of boosting: gbdt, dart, goss, or rf
objective	Defines the learning task (binary, multiclass, regression, etc.)
metric	Evaluation metrics (auc, binary_error, logloss, rmse, etc.)
num_iterations / n_estimators	Number of boosting iterations (trees)
learning_rate	Shrinks the contribution of each tree (0.01–0.3 typical)
num_leaves	Number of leaves per tree (controls complexity)
max_depth	Limits maximum depth of a tree
min_data_in_leaf	Minimum number of samples per leaf
feature_fraction	Fraction of features used for each tree (like colsample_bytree)
bagging_fraction	Fraction of data used for each iteration (like subsample)
bagging_freq	Frequency for bagging (0 means disable)
lambda_l1	L1 regularization term
lambda_l2	L2 regularization term
min_gain_to_split	Minimum gain required for a split
max_bin	Number of bins for discretizing continuous features
scale_pos_weight	Weight of positive class (for imbalanced datasets)

Advanced / Efficiency Parameters

Parameter	Description
device	Can use cpu or gpu
verbosity	Controls logging level
n_jobs	Number of parallel threads
deterministic	Ensures reproducibility (True/False)

extra_trees	Enables extra randomization in splitting

DART Parameters (for Dropouts meet Multiple Additive Regression Trees)

Parameter	Description
drop_rate	Fraction of trees dropped in each iteration
skip_drop	Probability of skipping dropout
max_drop	Maximum number of dropped trees

GOSS Parameters (Gradient-based One-Side Sampling)

Parameter	Description
top_rate	Percentage of large gradient data kept
other_rate	Fraction of small gradient data kept

Day5

Comparison of Logistic Regression, XGBoost, and LightGBM

Aspect	Logistic Regression	XGBoost	LightGBM
Type of Model	Linear (statistical) model	Ensemble of Decision Trees (Boosting)	Ensemble of Decision Trees (Boosting)
Learning Approach	Fits a linear decision boundary using weights	Boosts weak learners sequentially to minimize errors	Boosts weak learners using leaf-wise tree growth for faster learning
Data Handling	Works best on scaled and linearly separable data	Handles both linear & non-linear data	Handles large, high-dimensional, and categorical data efficiently
Speed	Very fast and simple to train	Slower than logistic regression but optimized	Faster than XGBoost due to histogram-based and leaf-wise growth
Overfitting Tendency	Prone to underfitting on complex data	Regularized (less overfitting due to parameters like gamma, lambda)	Slightly higher risk of overfitting (deep leaf-wise trees)
Interpretability	Highly interpretable (coefficients show feature impact)	Less interpretable (complex tree ensembles)	Less interpretable (complex tree structures)
Scalability	Works well on small to medium datasets	Scales well on medium to large datasets	Excellent scalability on large datasets
Handling Missing Values	Requires preprocessing/imputation	Handles missing values internally	Handles missing values automatically
Categorical Features	Needs manual encoding (e.g., one-hot)	Needs manual encoding	Handles categorical features directly (using native support)

Performance	Moderate; baseline model	High accuracy and robustness	Higher accuracy and faster training for large data
Best Use Case	Baseline model for binary classification	Complex, structured data with moderate size	Very large datasets, real-time and production-level models

Day6

Day 6 – Hyperparameter Tuning and Model Validation

1. Hyperparameter Tuning

- Definition:
  Hyperparameter tuning is the process of finding the best set of hyperparameters (external parameters that control the learning process) to improve model performance.

- Hyperparameters are not learned automatically — they must be set manually before training (e.g., `learning_rate`, `max_depth`, `num_leaves`).

2. Grid Search

- Definition:
  Grid Search is a method that performs an exhaustive search over a specified parameter grid.
  It tries every possible combination of parameters to find the one that gives the best performance (usually based on cross-validation).

Example:

```
param_grid = {'max_depth': [3, 5, 7], 'learning_rate': [0.01, 0.1, 0.2]}
```

Advantages:

1. Simple and easy to implement.
2. Guarantees finding the best combination within the provided grid.
3. Works well for small parameter spaces.

Disadvantages:

1. Computationally expensive for large grids.
2. Time-consuming — checks every parameter combination even if many are irrelevant.
3. Inefficient when many parameters don't affect performance much.

3. Random Search

- Definition:
  Random Search samples a fixed number of random combinations from the parameter grid instead of testing all possibilities.
  It explores the search space faster and often finds a near-optimal solution.

Example:

```
param_dist = {'max_depth': [3, 5, 7], 'learning_rate': [0.01, 0.1, 0.2]}
```

Advantages:

1. Much faster than Grid Search.
2. Can find good hyperparameter combinations even with large search spaces.
3. More flexible and efficient — not all combinations are tested.

Disadvantages:

1. May miss the best parameter combination.
2. Results can vary between runs due to randomness.
3. Requires specifying a suitable number of iterations manually.

4. Cross Validation

- Definition:
  Cross-validation is a method to evaluate a model's performance by splitting the dataset into multiple folds.
  The model is trained on some folds and tested on the remaining one, and this is repeated several times.
  The final score is the average performance across all folds.

Types:

- k-Fold Cross Validation (most common)
- Stratified k-Fold (for imbalanced data)

- Leave-One-Out (LOOCV)

Advantages:

1. Provides a more reliable estimate of model performance.
2. Reduces the risk of overfitting.
3. Makes better use of available data.

Disadvantages:

1. Computationally expensive (model is trained multiple times).
2. Can take long for large datasets or complex models.
3. Slightly harder to implement for time-series data.

Day7

Day 7 – Model Explainability and Feature Insights

1. Why Explainability is Essential in Credit Risk Modeling

Explainability means understanding *why* a model makes certain predictions.
In credit risk modeling, explainability is crucial for the following reasons:

1. Regulatory Compliance:
   Financial institutions must justify lending decisions to comply with laws like Fair Credit Reporting Act (FCRA) and GDPR.

2. Transparency & Trust:
   Borrowers and stakeholders must understand why a loan was approved or rejected — it builds trust in the model.

3. Bias Detection:
   Helps identify and eliminate discrimination or bias toward certain groups.

4. Model Debugging:
   Explainable results help analysts detect data issues or overfitting.

5. Actionable Business Insights:
   By knowing which features most influence defaults, lenders can design better loan policies.

2. Feature Importance

- Definition:
  Feature importance quantifies how much each feature contributes to model predictions.
  In tree-based models like XGBoost or LightGBM, it is calculated based on:
    - How often a feature is used to split,
    - How much it improves the split quality (gain),
    - Or how it reduces impurity.

- Purpose in Credit Risk:
  It helps identify key drivers of loan defaults such as interest rate, loan amount, or debt-to-income ratio (DTI).

3. SHAP (SHapley Additive exPlanations)

- Concept:
  SHAP assigns each feature a *Shapley value* that explains its contribution to the final prediction (positive or negative).
  It's based on game theory, where each feature is treated like a "player" contributing to the model's output.

- Why Use SHAP:
  - Provides both global (overall) and local (individual prediction) explanations.
  - Works well for complex models like XGBoost and LightGBM.
  - Shows whether a feature increases or decreases the probability of default.

- Example:
  A SHAP plot can show that higher int_rate increases default probability, while higher annual_inc decreases it.

4. LIME (Local Interpretable Model-Agnostic Explanations)

- **Concept:**
  LIME explains *individual predictions* by approximating the model locally using a simpler, interpretable model (like linear regression).

- **Why Use LIME:**
  - Works with any ML model (model-agnostic).
  - Shows how changing input features affects one borrower's prediction.
  - Useful for customer-level analysis.

5. Top Five Features (from Lending Club Dataset)

Below are the most influential features (based on model importance or SHAP summary):

Feature	Expected Influence on Default Risk	Reason
int_rate (Interest Rate)	Higher → Higher default risk	High interest rates often indicate riskier borrowers; they pay more, making default more likely.
dti (Debt-to-Income Ratio)	Higher → Higher default risk	A borrower with high DTI already has more debt obligations and may struggle to repay loans.
annual_inc (Annual Income)	Higher → Lower default risk	Higher income generally means more financial stability and repayment capacity.
revol_util (Revolving Credit Utilization)	Higher → Higher default risk	High credit utilization indicates financial stress and poor credit management.
loan_amnt (Loan Amount)	Larger → Higher default risk	Bigger loans mean higher repayment pressure, increasing the chance of default.

Day8

Day 8 – Handling Imbalanced Data

1. What is Imbalanced Data?

Definition:
Imbalanced data occurs when one class (e.g., “Non-Default”) has significantly more samples than the other (e.g., “Default”).
For example, in credit risk prediction:

Fully Paid (No Default) → 85%
Charged Off (Default) → 15%

- The model tends to favor the majority class, ignoring the minority class (the actual defaults we want to detect).

2. How Imbalance Affects the Model

1. **Bias Toward Majority Class:**
   The model predicts most samples as the dominant class, achieving high accuracy but poor recall for minority class.

2. **Poor Detection of Defaults:**
   In credit risk, this means the model might miss true defaulters.

3. **Misleading Accuracy:**
   Example: 90% accuracy might sound good, but if 90% are “non-default,” the model could be predicting “non-default” for everything.

4. **Degraded Performance on Key Metrics:**
   Affects recall, precision, F1, and AUC-ROC negatively.

3. Over Sampling

- **Definition:**
  Technique where samples from the minority class are duplicated or synthetically generated to balance the dataset.

- **Example:**
  If there are 90,000 “Fully Paid” and 10,000 “Default,” oversampling increases “Default” samples to match.

- **Pros:**

- Improves model's ability to detect minority cases.
- Preserves all majority data.

- **Cons:**
  - May cause overfitting since minority samples are repeated.
  - Increases training time.

4. Under Sampling

- **Definition:**
  Technique where samples from the majority class are randomly removed to balance the dataset.

- **Example:**
  Reducing 90,000 "Fully Paid" samples down to 10,000 to match "Default."

- **Pros:**
  - Faster training and smaller dataset.
  - Reduces class imbalance quickly.

- **Cons:**
  - Risk of information loss — removes potentially useful data.
  - May reduce model accuracy.

5. SMOTE (Synthetic Minority Over-sampling Technique)

- **Definition:**
  SMOTE generates synthetic (new) samples for the minority class by interpolating between existing minority samples.
  Instead of duplicating data, it creates new, realistic data points.

- **How it works:**
  1. Picks a random minority sample.
  2. Selects one of its nearest neighbors (from same class).
  3. Creates a new synthetic point between the two.

Example (Python):

```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X, y)
```

Advantages of SMOTE

1. Creates diverse, synthetic data rather than simple duplicates.
2. Reduces overfitting compared to normal oversampling.
3. Helps models learn minority class patterns better.
4. Improves recall and AUC for imbalanced datasets.

Disadvantages of SMOTE

1. Can generate noisy or unrealistic samples if classes overlap.
2. May increase training time due to larger dataset.
3. Doesn't address class overlap — can confuse model boundaries.
4. Works only on numeric features (requires encoding categorical features first).

Summary Table

Method	Definition	Pros	Cons
Over Sampling	Duplicate minority class samples	Improves minority learning	Overfitting risk
Under Sampling	Remove majority class samples	Fast and simple	Data loss
SMOTE	Synthetic sample generation	Better generalization	Can add noise

Day9

Day 9 – Derived Features in Credit Risk Modeling

1. What are Derived Features?

Definition:
Derived features are new features created from existing raw data to help the model capture hidden patterns, relationships, or domain-specific insights.

2. Why Derived Features are Important

1. **Enhance Model Predictive Power:**
   New features often reveal patterns that the model might otherwise miss.

2. **Represent Business Logic:**
   In credit risk, relationships between income, debt, and loan size are more meaningful than their raw values.

3. **Improve Generalization:**
   Well-designed derived features help the model perform better on unseen data.

4. **Reduce Noise:**
   Combining related variables can reduce variability and simplify relationships.

3. Examples of Derived Features in Credit Risk Modeling

Derived Feature	Formula / Example	Interpretation / Usefulness
loan_to_income_ratio	loan_amnt / annual_inc	Measures how big the loan is compared to income — higher ratio → higher default risk.
installment_to_income_ratio	installment / (annual_inc / 12)	Measures how much of monthly income goes to paying the loan — higher → higher risk.
credit_utilization_ratio	revol_bal / credit_limit (or use revol_util)	Indicates how much of available credit is being used.

emp_length_num	Convert “10+ years”, “<1 year” to numbers (e.g., 10, 0.5)	Numeric representation helps models understand employment stability.
income_per_open_account	annual_inc / open_acc	Higher income per account → less financial stress.
total_credit_exposure	revol_bal + loan_amnt	Total debt burden; higher value → higher risk.
credit_age	issue_year - earliest_cr_line_year	Number of years of credit history; longer history → lower default risk.
dti_category	Bucket DTI into bins (e.g., Low, Medium, High)	Simplifies debt-to-income interpretation.
is_home_owner	home_ownership == "OWN"	Binary indicator; owners usually have lower default rates.

4. How to Create Derived Features in Code

Example using your Lending Club dataset:

```
import pandas as pd

df['loan_to_income_ratio'] = df['loan_amnt'] / df['annual_inc']
df['installment_to_income_ratio'] = df['installment'] / (df['annual_inc'] / 12)
df['credit_age'] = pd.to_datetime(df['issue_d']).dt.year -
pd.to_datetime(df['earliest_cr_line']).dt.year
df['emp_length_num'] = df['emp_length'].replace({
    '10+ years': 10, '< 1 year': 0.5, '1 year': 1, '2 years': 2, '3 years': 3,
    '4 years': 4, '5 years': 5, '6 years': 6, '7 years': 7, '8 years': 8, '9 years': 9
})
df['total_credit_exposure'] = df['revol_bal'] + df['loan_amnt']
```

5. Advantages of Derived Features

✅ Improve model accuracy and interpretability.
✅ Capture complex relationships between multiple variables.
✅ Allow domain-specific insights to influence the model.
✅ Can reduce dimensionality by combining multiple weak predictors.

6. Limitations

❌ Risk of introducing noise or multicollinearity if not designed carefully.
❌ Time-consuming and requires domain knowledge.
❌ Derived features may not generalize well across different datasets.

Day10

Day 10 – Derived Features Impact, APIs, and Credit Path AI Workflow

1. How Derived Features Affect Model Performance

- Derived features enhance model performance by helping capture complex relationships between variables.

- They provide additional predictive signals that raw data might not express clearly.

Example:
A model may not understand loan_amnt or annual_inc individually,
but creating a derived feature like loan_to_income_ratio = loan_amnt / annual_inc
helps the model assess borrower risk more directly.

Positive Impacts:

1. Improves AUC, accuracy, recall, etc.

2. Provides business-relevant insights.

3. Reduces model bias by clarifying relationships.

2. Can Derived Features Hurt the Model?

Yes — if not designed properly.
While derived features can help, **poorly chosen or redundant** ones can degrade performance.

Possible Issues:

1. **Multicollinearity:**
   Derived features may be strongly correlated with existing features, confusing the model.
   (e.g., loan_to_income_ratio and dti might overlap)

2. **Overfitting:**
   Too many derived features can make the model memorize training data instead of learning patterns.

3. **Noise Addition:**
   Irrelevant or incorrect transformations can add unnecessary noise.

Best Practice:

- Always validate the feature's importance using SHAP or feature importance plots before including it.

3. Common Mistakes While Making Derived Features

Mistake	Explanation	Impact
Creating too many derived features	Reduces interpretability, causes overfitting	Poor generalization
Using features that leak target info	Example: using "loan_status" in derived calculation	Unrealistically high accuracy
Ignoring scaling of ratios	Ratios can distort magnitude	Model instability
Not removing correlated features	Causes redundancy	Slower and less stable training
Applying transformations blindly	e.g., taking log of zero values	Errors during execution

4. Why JSON is Suitable for Credit Path AI

JSON (JavaScript Object Notation) is perfect for Credit Path AI's APIs because:

1. **Lightweight & Fast:** Easily transferred between frontend (web) and backend (FastAPI).
2. **Human & Machine Readable:** Simple structure using key–value pairs.
3. **Language Independent:** Works with Python (FastAPI), JavaScript (frontend), etc.
4. **Structured Data Exchange:** Represents borrower data (loan amount, income, etc.) neatly.
5. **Standard for REST APIs:** Most web APIs use JSON for input/output.

Example:

```
{
  "loan_id": "L001",
  "loan_amnt": 15000,
  "term": "36 months",
  "int_rate": 12.5,
  "annual_inc": 60000
}
```

5. HTTP and REST API Concepts

Concept	Description
HTTP (HyperText Transfer Protocol)	The communication protocol between client and server.
API (Application Programming Interface)	Interface allowing software applications to communicate.
REST API (Representational State Transfer)	API design that uses HTTP methods (GET, POST, PUT, DELETE) to exchange data.
FastAPI	A modern, high-performance Python framework for building REST APIs quickly and efficiently.

6. APIs Created in Credit Path AI

API Endpoint	Method	Purpose
/predict	**POST**	Accepts borrower input and returns predicted default risk + recommended actions.
/health	**GET**	Checks if the backend service is running (used for debugging).
/logs *(optional)*	**GET**	Retrieves past predictions from logs for admin use.

7. Why We Use POST for Predict in Credit Path AI

Reasons:

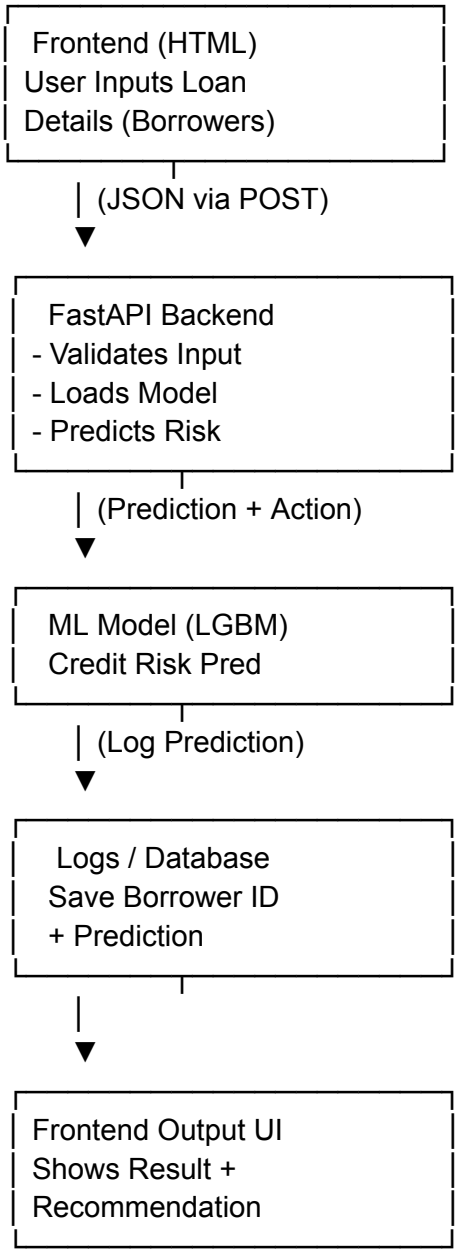
1. **POST** allows sending borrower data in request body, unlike GET (which sends in URL).
2. **Secure:** Data (like income, loan details) isn't exposed in the URL.
3. **Larger Payloads:** Can handle complex borrower objects (like JSON input).
4. **Best Practice:** For actions that change state or involve computation, POST is preferred.

Example Request:

```
import requests
data = {
    "loan_amnt": 12000,
    "term": "36 months",
    "int_rate": 13.5,
    "annual_inc": 50000,
    "dti": 18.5
}
response = requests.post("http://127.0.0.1:8000/predict", json=data)
print(response.json())
```

8. Workflow Diagram of Credit Path AI

Below is a simple conceptual **workflow (you can draw in Google Docs using arrows):**



Day11

Day 11 – API Design and Validation in Credit Path AI

1. Why Do We Choose FastAPI in This Project?

FastAPI is chosen for Credit Path AI because it is:

1. Fast and Modern — built on *Starlette* and *Pydantic*, it's faster than Flask for real-time ML predictions.
2. Easy Integration with ML Models — allows smooth integration of trained XGBoost or LightGBM models.
3. Automatic Validation — validates input and output data using *Pydantic models*.
4. Auto-Generates API Docs — provides /docs and /redoc automatically for testing APIs.
5. JSON Native — handles JSON input/output efficiently, ideal for borrower data exchange.

2. Pydantic Error Explanation

When the input JSON sent by the frontend does not match the defined schema,
FastAPI automatically raises a Pydantic Validation Error.

Example:
If "loan_amnt" should be a float but user sends "loan_amnt": "twelve thousand",
FastAPI will return:

```
{
  "detail": [
    {
      "loc": ["body", 0, "loan_amnt"],
      "msg": "value is not a valid float",
      "type": "type_error.float"
    }
  ]
}
```

This prevents invalid borrower data from entering the model and ensures data consistency.

3. JSON Format Overview

JSON (JavaScript Object Notation) is used to exchange borrower data between frontend and backend.
It uses simple key-value pairs like this:

```
{
  "loan_amnt": 15000,
```

```
  "term": "36 months",
  "annual_inc": 60000
}
```

- Keys are strings
- Values can be string, number, boolean, array, or object

JSON is lightweight, readable, and perfect for transmitting borrower data securely.

4. API Schema for Batch Prediction

We design APIs that can handle multiple borrower inputs in one request.

So the API Schema should include:

- A list of borrower objects as input
- A list of predictions as output

5. Testing API using Postman

Postman helps in sending requests to your FastAPI endpoints easily.

Steps:

1. Open Postman
2. Select POST method
3. Paste your API URL → e.g., http://127.0.0.1:8000/predict
4. Choose “Body → raw → JSON”
5. Paste borrower data JSON
6. Click Send to get the response

6. API Input Schema Design (for Batch Prediction)

We use Pydantic models in FastAPI to define structure and datatype validation.

```
from pydantic import BaseModel
from typing import List
```

```
class Borrower(BaseModel):
    loan_id: str
    loan_amnt: float
    term: str
    annual_inc: float

class BorrowerBatch(BaseModel):
    borrowers: List[Borrower]
```

Explanation:

- loan_id: string (unique ID of borrower)
- loan_amnt: float (loan amount)
- term: string (loan duration)
- annual_inc: float (annual income)
- borrowers: a list that holds multiple borrower entries

7. API Output Schema Design (for Batch Prediction)

```
class BorrowerPrediction(BaseModel):
    loan_id: str
    probability_default: float
    recommended_action: str

class PredictionResponse(BaseModel):
    predictions: List[BorrowerPrediction]
```

Explanation:

- loan_id: borrower ID (same as input)
- probability_default: model output probability of default
- recommended_action: e.g., "Approve", "Reject", "Review Manually"

8. Example JSON Input (Multiple Borrowers)

```
{
  "borrowers": [
    {
```

```
    "loan_id": "L001",
    "loan_amnt": 10000,
    "term": "36 months",
    "annual_inc": 60000
   },
   {
    "loan_id": "L002",
    "loan_amnt": 15000,
    "term": "60 months",
    "annual_inc": 50000
   },
   {
    "loan_id": "L003",
    "loan_amnt": 20000,
    "term": "36 months",
    "annual_inc": 90000
   }
  ]
}
```

9. Example JSON Output (Predictions for Borrowers)

```
{
  "predictions": [
   {
    "loan_id": "L001",
    "probability_default": 0.18,
    "recommended_action": "Approve"
   },
   {
    "loan_id": "L002",
    "probability_default": 0.67,
    "recommended_action": "Review Manually"
   },
   {
    "loan_id": "L003",
    "probability_default": 0.83,
    "recommended_action": "Reject"
   }
  ]
}
```

10. Short Notes on Data Types

Field	Data Type	Description
loan_id	str	Unique identifier of borrower
loan_amnt	float	Loan amount applied
term	str	Duration of loan
annual_inc	float	Borrower’s yearly income
probability_default	float	Model-predicted default probability
recommended_action	str	Suggested decision (Approve/Reject/Review)

11. What Should API Return if Validation Fails

If user sends invalid input, FastAPI automatically returns a 422 Unprocessable Entity error with details.

Example:


```
{
  "detail": [
    {
      "loc": ["body", "borrowers", 1, "loan_amnt"],
      "msg": "value is not a valid float",
      "type": "type_error.float"
    }
  ]
}
```


This ensures the backend is secure and reliable, preventing faulty predictions.

Day12

Day 12 – API Pseudocode and Error Handling Table

1. Pseudocode for Batch Prediction API in Credit Path AI

Goal:
Build an API that accepts multiple borrower records, performs validation, runs batch prediction using trained model, and returns probability + recommended action.

Pseudocode (10–12 Logical Steps)

1. **Start**
2. **Import required libraries**
    - FastAPI, Pydantic (for validation), joblib/pickle (for model), numpy/pandas
3. **Define Pydantic schemas**
    - Borrower → individual borrower fields
    - BorrowerBatch → list of borrowers
4. **Load pre-trained ML model** (e.g., LightGBM or XGBoost model from file)
5. **Initialize FastAPI app**
6. **Define /predict POST endpoint**
    - Input: JSON payload with multiple borrower records
    - Validate structure automatically via Pydantic
7. **For each borrower in request:**
    - Extract features into dataframe format
    - Check for missing or invalid fields
    - Handle errors (if invalid → skip or return validation message)
8. **Pass valid borrower features to ML model**
    - Generate probability_default = model.predict_proba(X)[:,1]

9. **Map probabilities to recommended actions:**
    - If prob < 0.4 → "Approve"
    - If 0.4 ≤ prob < 0.7 → "Review Manually"
    - If prob ≥ 0.7 → "Reject"
10. **Store predictions in log file**
    - Log loan_id, probability, and action with timestamp
11. **Return response JSON**
    - Include loan_id, probability_default, and recommended_action for each borrower
12. **End**

Example Output Flow

Input → Validate → Predict → Map Actions → Log Results → Return JSON

2. Error Handling Table

Below is a short list of **common API errors**, their **causes**, and the **expected API response**.

Error Name / Scenario	What Causes It	Expected API Response (Message + Code)
Validation Error	Missing or invalid data type in JSON (e.g., loan_amnt = "abc")	{ "detail": "Validation Error: Invalid input type or missing field." } **Code:** 422
Missing Field Error	Required field (like annual_inc) is not provided	{ "detail": "Missing field in borrower data." } **Code:** 400
Model File Not Found	Model file path is incorrect or model not loaded	{ "detail": "Model not found on server. Please check deployment." } **Code:** 500
Prediction Error	Unexpected data shape or preprocessing mismatch	{ "detail": "Prediction failed due to invalid input shape." } **Code:** 500

JSON Decode Error	User sends malformed JSON (syntax error)	{ "detail": "Invalid JSON format. Please check input structure." } **Code:** 400
Internal Server Error	Unexpected runtime exception in backend	{ "detail": "Internal Server Error. Please try again later." } **Code:** 500
Empty Request	No borrower data sent in request	{ "detail": "Request body cannot be empty." } **Code:** 400
Too Many Borrowers	API limit exceeded (e.g., > 100 borrowers per request)	{ "detail": "Request size too large. Limit 100 borrowers per call." } **Code:** 413

Day13

Day 13 – Recommendation System and Model Calibration

1. Recommendation System and Mapping in Credit Path AI

Definition:
A recommendation system in the Credit Path AI project is designed to map model predictions (probability of default) into actionable financial decisions.

It takes the output of the ML model — *probability of default* — and translates it into loan approval recommendations for each borrower.

2. Mapping Logic (From Prediction → Recommendation)

After the LightGBM or XGBoost model predicts the probability of default,
the system classifies borrowers into risk categories and assigns a recommended action.

Predicted Default Probability (p)	Risk Level	Recommended Action
$0.00 \leq p < 0.30$	**Low Risk**	✅ Approve Loan
$0.30 \leq p < 0.60$	**Medium Risk**	⚠️ Manual Review Required
$0.60 \leq p \leq 1.00$	**High Risk**	❌ Reject / Request Collateral

✅ Purpose:

- Converts numerical predictions into understandable and usable business actions.
- Helps financial analysts and loan officers make consistent, data-driven decisions.
- Ensures the AI model supports human decision-making, not replaces it.

3. What is a Well-Calibrated vs Poorly-Calibrated Model

Model Calibration refers to how well a model's predicted probabilities reflect real-world outcomes.

Model Type	Description	Example
Well-Calibrated Model	Model's probabilities closely match actual outcomes.	If model predicts 0.7 probability → borrower defaults 70% of the time.

Poorly-Calibrated Model	Model's probabilities are overconfident or underconfident.	Predicts 0.9 but only 50% actually default.

4. Why Calibration Matters in Credit Risk

1. **Better Decision Thresholds:** Ensures consistent risk classification (low, medium, high).
2. **Improves Trust:** Decision-makers rely more on stable probabilities.
3. **Regulatory Compliance:** Financial regulators expect explainable and calibrated risk scores.
4. **Fair Lending Decisions:** Prevents bias caused by extreme or misleading probabilities.

5. How to Check Model Calibration

Techniques:

- **Calibration Curve (Reliability Plot):** Plots predicted probability vs actual outcomes.
- **Brier Score:** Measures how close predicted probabilities are to actual results.
  Brier Score=1N∑(yi−pi^)2\text{Brier Score} = \frac{1}{N} \sum (y_i - \hat{p_i})^2Brier Score=N1∑(yi−pi^)2
  Lower Brier score → better calibration.
- **Isotonic Regression** or **Platt Scaling:**
  Post-training methods used to recalibrate model outputs.

Deployment

CreditPathAI Deployment Summary

1. Deployment Process (Render + GitHub)

This project was deployed using GitHub for source code management and Render as the cloud hosting platform.
The deployment process was simple and automated:

Step 1: Prepare Project for Deployment

- Organized project structure with backend.py, frontend/, static/, and ML models (.pkl files).
- Created a requirements.txt file for all Python dependencies.
- (Optional) Added render.yaml to automate build and start commands.

Step 2: Upload Code to GitHub

- Initialized a Git repository.

Added the project files and pushed to a new GitHub repository:

```
git add .
git commit -m "Initial deploy"
git push origin main
```

-

Step 3: Deploy on Render

- Logged into https://render.com
- Clicked New → Web Service
- Selected the GitHub repository containing the project
- Set:
    - **Build Command:** pip install -r requirements.txt
    - **Start Command:** uvicorn backend:app --host 0.0.0.0 --port 10000

    - **Environment:** Python
    - **Instance Type:** Free Tier
- Clicked **Deploy**

Render automatically:

- Built the app
- Installed all dependencies
- Started FastAPI
- Hosted the full system with a public URL

Final Output:
A production-ready, publicly accessible link to the CreditPathAI application.

2. Why I Chose Render for Deployment

Render was selected because it provides the best balance of simplicity, speed, and reliability for cloud deployment of FastAPI applications.

Reasons for Choosing Render:

1. Easy FastAPI Support

Render natively supports Python and Uvicorn deployments, making it ideal for ML + FastAPI projects.

2. Automatic GitHub Integration

Any code pushed to GitHub automatically updates the deployed app — ideal for continuous development.

3. Free Tier for Students

Render offers generous free hosting suitable for academic and internship projects.

4. Handles Backend + Frontend in One Place

The platform can serve:

- Backend APIs

- Frontend static files
- Machine learning models
  all in one environment.

5. Built-in SSL and Public URLs

Provides secure HTTPS links with zero configuration.

6. No Docker or DevOps Needed

You don’t need to set up Docker, Nginx, or server configurations — beginner friendly and fast.

3. Who Can Use My Project? (Intended Users)

CreditPathAI is designed for lending, banking, and financial organizations that need AI-based borrower risk evaluation.

Target Users:

1. Loan Officers / Credit Underwriters

- Evaluate borrower default risk instantly
- Get insights into safe vs high-risk applications
- Receive recommended actions (call borrower, restructure, etc.)

2. Financial Institutions

- Automate initial screening of loan applications
- Reduce manual workload
- Improve decision accuracy

3. Collection Teams

- Identify customers likely to default
- Prioritize follow-ups and reminders
- Reduce loss and improve recovery rate

4. FinTech Startups

- Integrate the API to offer risk-based loan decisions
- Embed ML-powered credit scoring into apps

5. Students / Researchers

- Understand ML pipeline deployment
- Learn backend integration
- Experiment with credit risk datasets

6. My Internship Mentor / Evaluation Team

- Test the complete working product
- Validate deployment, API responses, and UI/UX
- Assess model performance and engineering quality