St. Xavier's College (Autonomous), Mumbai

M.Sc. Big Data Analytics – Part II

Academic Year: 2024–2026

Sectoral Analysis of Expected Credit Loss in Indian Development Finance (2014–2024)

Subject: Research Project

Subject Code: PSBDA6501RP1

Instructor: Ms. Ruqaiya Shaikh

Group Members

Student Name	Roll No.	UID
Glen Jacob	07	2409007
Praiselin Abishaya	12	2409012
Suneha Dutta	14	2409014
Darfisha Shaikh	27	2409027
Khushi Kakani	34	2409034

TABLE OF CONTENTS

ABSTRACT

This project examines credit risk assessment for World Bank-funded projects in India from 2014 to 2024. It combines loan-level financial data with macroeconomic indicators. The study creates a dataset of project loans, cleaning and building features such as repayment ratio, loan-to-GDP ratio, and sector classification. Loan performance is modeled with a binary default flag that separates repaid projects from those that are risky or in default. We train and compare multiple machine learning classifiers, including Logistic Regression, Random Forest, XGBoost, and CatBoost. CatBoost shows strong predictive accuracy and interpretability through SHAP analysis. Using risk modeling principles, the study estimates Probability of Default (PD), Exposure at Default (EAD), and Loss Given Default (LGD) for each project and calculates Expected Credit Loss (ECL) across sectors. The sector analysis reveals differences in default probabilities and exposure, showing that high-risk projects are mainly in infrastructure, water, and social protection areas. The results provide data-driven insights into financial vulnerability at the portfolio level. This helps policymakers and financial institutions better manage sovereign credit risk and direct resources to sectors that support resilient development.

Introduction:

Development finance is crucial for India's growth. The World Bank funds many projects in areas like infrastructure, social protection, and rural development. However, the scale and variety of these loans expose them to credit risk. This risk can come from challenges in executing projects, vulnerabilities within sectors, and broader economic conditions. Traditional risk assessment methods often miss these complexities at the project level. This research fills that gap by creating a machine learning framework. It merges loan-level financial data with economic indicators to predict defaults and measure credit risk. By estimating Probability of Default (PD), Exposure at Default (EAD), Loss Given Default (LGD), and Expected Credit Loss (ECL), the study evaluates risks at the project level and offers insights into India’s development portfolio. The findings aim to support better lending strategies, improve risk management, and guide sustainable policy changes.

Objective:

The main goal of this research is to create a strong framework for assessing credit risk in World Bank-funded projects in India. It uses machine learning methods and financial risk modeling. The study combines loan-level financial data with macroeconomic indicators to identify factors that lead to defaults and measure risk exposure across different sectors.

It focuses on building a predictive model for project-level defaults. This model will estimate key risk parameters: Probability of Default (PD), Exposure at Default (EAD), and Loss Given Default (LGD). After that, it will calculate the Expected Credit Loss (ECL).

Additionally, the research carrried out an analysis by sector. This helped highlight areas and projects with high risks, that provided valuable insights for lending strategies, risk management policies, and the effective use of development finance.

Outcome:

The project successfully develops a machine learning-based framework for assessing credit risk in World Bank-funded projects in India. It provides insights at both the project and sector levels. The analysis shows that effective models, especially CatBoost, achieve high accuracy in identifying projects at risk of default and provide clear results through SHAP analysis. The study gives quantitative estimates of Probability of Default (PD), Exposure at Default (EAD), Loss Given Default (LGD), and Expected Credit Loss (ECL) for individual projects. It also reveals differences in default risk across sectors, with infrastructure, water, and social protection projects showing more vulnerabilities. Additionally, the research creates ranked lists of high-risk projects and sectors. This enables policymakers, financial institutions, and development agencies to prioritize strategies to reduce risk and improve financial oversight. Ultimately, the results support evidence-based decision-making in managing sovereign credit risk. They also provide a framework that can be used to evaluate development finance in other emerging economies.

Dataset Sources and Collection:

a) Microeconomic Data: The study utilizes secondary data from publicly available World Bank loan databases, focusing on projects funded all over the world between 1945 and 2024; (in the project we have filtered out for 'India' from 2014-2024).

Link:**https://financesone.worldbank.org/ibrd-statement-of-loans-and-guarantees-historical-data/DS00975**

b) Macroeconomic Data: Annual GDP and CPI data from World Bank (2014-2024)

Link: **https://data.worldbank.org/indicator/FP.CPI.TOTL.ZG (FOR CPI)**

https://data.imf.org/en/Data-Explorer?datasetUrn=IMF.RES:WEO(6.0.0)&INDICATOR=NGDP_D (FOR GDP)

Final Dataset Description:

The original global dataset was filtered to create an India-specific dataset containing loan-level transaction records across 2014-2024. After preprocessing and cleaning, the final analytical dataset comprises approximately 1,200 observations across 25+ variables, representing 156 unique development projects with varying numbers of loan transactions per project over the study period. Key financial variables include original principal amounts, disbursed amounts, repaid amounts, borrower obligations, interest rates, and various temporal markers such as agreement signing dates, disbursement dates, and repayment schedules. Each record represents a loan transaction within broader development projects, creating a hierarchical structure where individual projects may contain multiple loans tracked over quarterly or annual reporting periods. Financial amounts are denominated in USD, with supplementary macroeconomic data including annual GDP growth rates and CPI inflation rates incorporated to provide economic context.

Methodology:

a) Data Cleaning and Processing:

World Bank Loan Data:

- Firstly, starting off with cleaning and standardizing raw column names into a consistent, machine-friendly format, making the dataset easier to work with during analysis and modeling.

```
df.columns = [
    col.strip().lower().replace(" ", "_").replace("/", "_").replace("(", "").replace(")", "").replace("$", "usd").replace("'", "").replace(".", "")
    for col in df.columns
]
```

- From the full World Bank dataset df, it selects only the rows where the column country__economy equals "India". .str.strip() removes any leading/trailing spaces in country names. The .copy() ensures a fresh dataframe is created without warnings.

```
india_df = df[df['country___economy'].str.strip() == 'India'].copy()
india_df.shape
india_df.to_csv('india_credit_risk.csv', index=False)
```

- Checking and deleting the "currency_of_commitment" column because it contains only missing values and does not add value to the analysis.

```
# 'currency_of_commitment' has all NaN
if 'currency_of_commitment' in india_df.columns:
    india_df.drop(columns=['currency_of_commitment'], inplace=True)
```

- Standardizing all key loan-related dates into a uniform datetime format and creates a derived feature ("origination_year") that captures the year each project agreement was signed.

```
# Convert date columns
date_cols = [
    'end_of_period',
    'first_repayment_date',
    'last_repayment_date',
    'agreement_signing_date',
    'board_approval_date',
    'effective_date_most_recent',
    'closed_date_most_recent',
    'last_disbursement_date'
]

for col in date_cols:
    if col in india_df.columns:
        india_df[col] = pd.to_datetime(india_df[col], errors='coerce')

# Add origination year
india_df["origination_year"] = india_df["agreement_signing_date"].dt.year.astype("Int64")
```

- Standardizing the dataset's financial variables by converting them into numeric form and treating negative values as missing, thereby ensuring their reliability for subsequent calculations such as repayment ratios, PD, EAD, LGD, and ECL.

```
#4. Financial columns
# ------------------------
numeric_cols = [
    'interest_rate', 'original_principal_amount_ususd', 'cancelled_amount_ususd',
    'undisbursed_amount_ususd', 'disbursed_amount_ususd', 'repaid_to_ibrd_ususd',
    'due_to_ibrd_ususd','exchange_adjustment_ususd',
    'borrowers_obligation_ususd', 'loans_held_ususd'
]

# Ensure numeric
india_df[numeric_cols] = india_df[numeric_cols].apply(pd.to_numeric, errors='coerce')

# Remove negative values → set as NaN
for col in numeric_cols:
    india_df[col] = india_df[col].apply(lambda x: np.nan if x < 0 else x)
```

- All categorical variables were standardized by converting object-type columns to strings and removing extraneous spaces. In particular, the "loan_status"

field was cleaned by casting it to Pandas' nullable string type, stripping leading and trailing whitespace, and converting all entries to uppercase. This ensured consistent formatting of categorical data and enabled reliable classification of loan statuses into default and non-default categories for subsequent analysis.

```
for col in india_df.select_dtypes(include="object").columns:
    india_df[col] = india_df[col].astype(str).str.strip()

india_df['loan_status'] = india_df['loan_status'].astype('string').str.strip().str.upper()
print("Loan status values:", india_df['loan_status'].unique())
```

```
india_df['loan_status'] = (
    india_df['loan_status']
    .astype('string')   # Pandas nullable string dtype
    .str.strip()
    .str.upper()
)
```

- Adding a new column 'is_active' that flags whether a loan is currently active or not, simplifying downstream analysis of loan portfolios.

```
active_statuses = [
    'REPAYING','DISBURSED','DISBURSING','DISBURSING&REPAYING',
    'FULLY DISBURSED','FULLY TRANSFERRED','APPROVED','SIGNED','EFFECTIVE'
]

india_df["is_active"] = india_df["loan_status"].isin(active_statuses).astype(int)
```

- **Encoding:** This step translates textual loan statuses into a standardized binary variable "default_flag", distinguishing safe loans (0) from risky or defaulted loans (1). It also accounts for partial disbursements and cancellations to give a more balanced view of default risk.

```
def encode_default_balanced(status, disbursed_amount):
    if not isinstance(status, str):
        return 1  # unknown → risky by default

    status = status.strip().upper()

    # Non-default
    if status in ["FULLY REPAID", "SIGNED","APPROVED","DISBURSING"]:
        return 0

    # Default (risky or incomplete repayment)
    if status in ["REPAYING","DISBURSED","DISBURSING&REPAYING","FULLY DISBURSED"]:
        return 1

    # Special case: CANCELLED
    if status in ["CANCELLED","FULLY CANCELLED"]:
        if disbursed_amount and disbursed_amount > 0:
            return 1  # risky cancel
        else:
            return 0  # safe cancel (no money given)

    # Catch-all: treat as risky
    return 1


# Apply to dataframe
india_df["default_flag"] = india_df.apply(
    lambda row: encode_default_balanced(row["loan_status"], row["disbursed_amount_ususd"]),
    axis=1
)
```

Stress Dataset:

- Reshaping the macroeconomic dataset into a yearly format with GDP growth and CPI inflation as variables, preparing it for integration with the loan dataset.

```
stress_long = stress_df.set_index("Indicator Name").T.reset_index()
stress_long.columns = ["year", "gdp_growth", "cpi_inflation"]
stress_long["year"] = stress_long["year"].astype(int)
print(stress_long.head())
```

Merging the two datasets:

The loan-level dataset was merged with the macroeconomic dataset by aligning records on the common year field. Each loan entry in "india_df" was matched with the corresponding GDP growth and CPI inflation values from "stress_long". An inner join was used to ensure that only those years present in both datasets were retained, thereby preserving consistency between loan transactions and their macroeconomic context. The merged dataset was then filtered to include only observations from 2014 to 2024, which defines the temporal scope of the study.

```
merged_df = pd.merge(india_df, stress_long, on="year", how="inner")

# Filter of 2014-2024
merged_df = merged_df[(merged_df["year"] >= 2014) & (merged_df["year"] <= 2024)]

print("Filtered merged dataset shape:", merged_df.shape)
print(merged_df.head(5))
```

- Dropping the columns not needed

```
merged_df.drop(columns=[
    "region", "country___economy", "borrower", "guarantor_country___economy_code",
    "guarantor", "sold_3rd_party_ususd",
    "repaid_3rd_party_ususd", "due_3rd_party_ususd", "agreement_signing_date",
    "closed_date_most_recent", "board_approval_date", "last_disbursement_date"
], inplace=True)
```

b) Feature Engineering:

Loan-to-GDP Growth Ratio:

- Divides the original loan principal by GDP growth (scaled to billions).
- Measures the relative size of a loan compared to annual economic growth.
- Invalid values (division by zero or missing GDP) are replaced with NaN, then filled with 0.

```
merged_df["loan_to_gdp_growth_ratio"] = (
    merged_df["original_principal_amount_ususd"] / (merged_df["gdp_growth"] * 1e9)
).replace([np.inf, -np.inf], np.nan).fillna(0)
```

Repayment Ratio:

- Compares the amount repaid to the amount disbursed.
- Indicates the proportion of disbursed funds that have been repaid.
- Again, infinite or missing values are handled by replacing with 0.

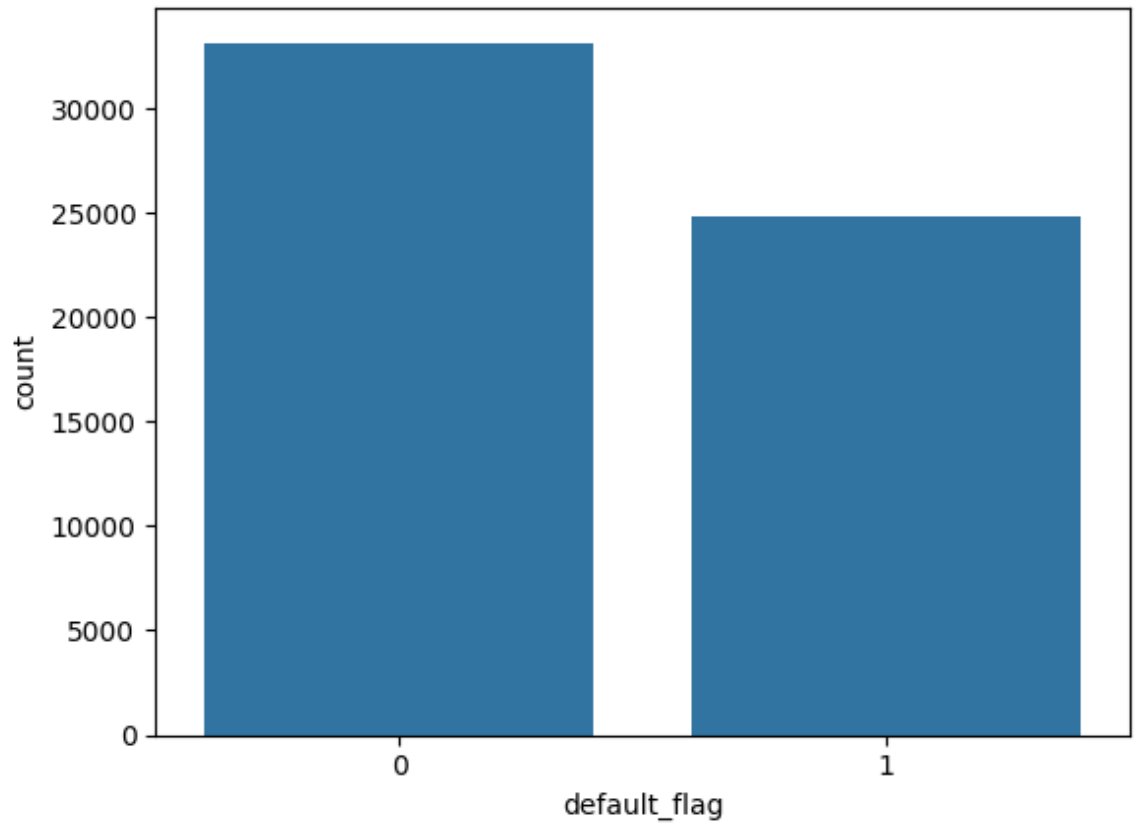
```
merged_df["repayment_ratio"] = (
    merged_df["repaid_to_ibrd_ususd"] / merged_df["disbursed_amount_ususd"]
).replace([np.inf, -np.inf], np.nan).fillna(0)
```

These engineered features quantify loan burden relative to GDP growth **and** repayment performance relative to disbursement, providing critical predictors for default risk modeling.

c) EDA:

- **Bar Plot**

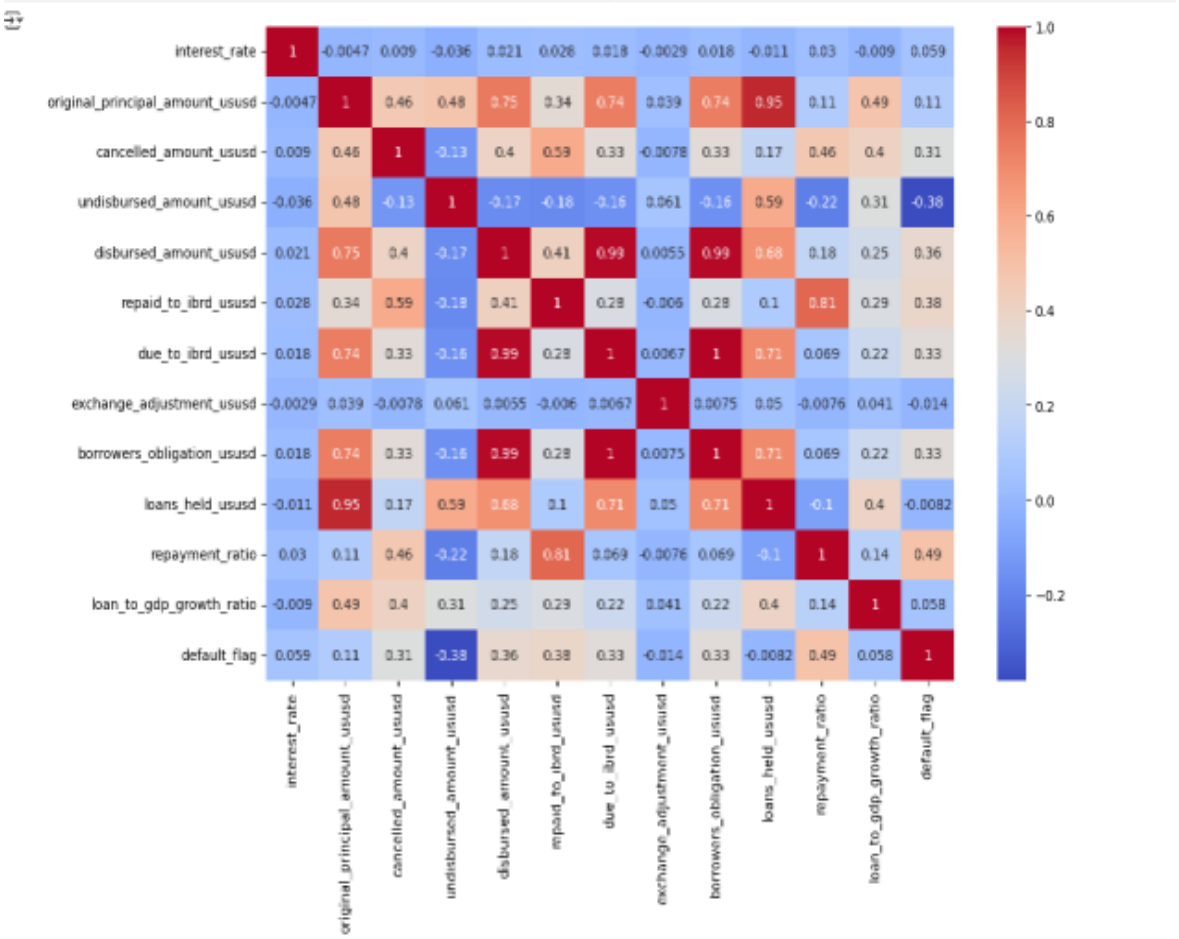
```
<Axes: xlabel='default_flag', ylabel='count'>
```



Illustrating the distribution 'default_flag', which categorizes loans as either non-default (0) or default (1). The results show that non-defaulted loans

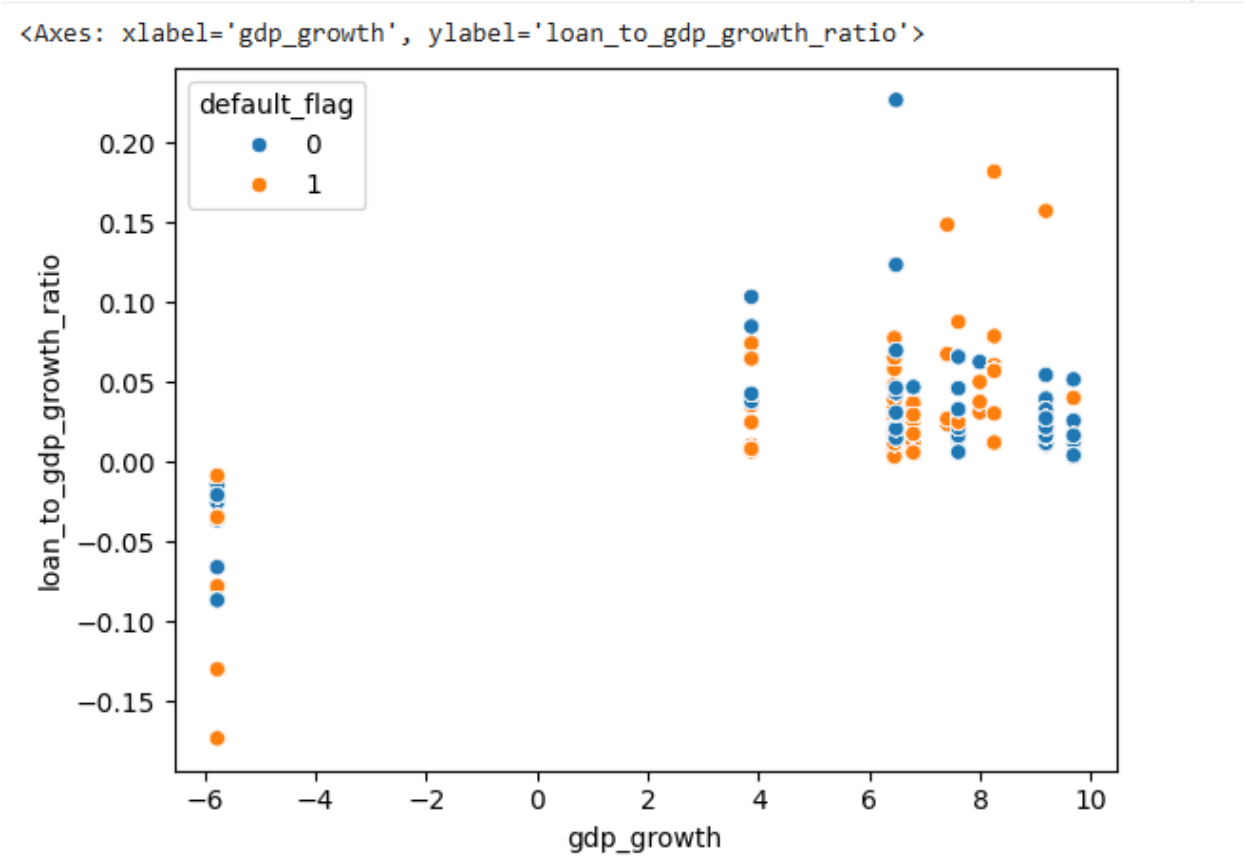
account for a slightly larger share of the dataset, with over 32,000 records, compared to approximately 25,000 defaulted loans.

- **Heatmap:**



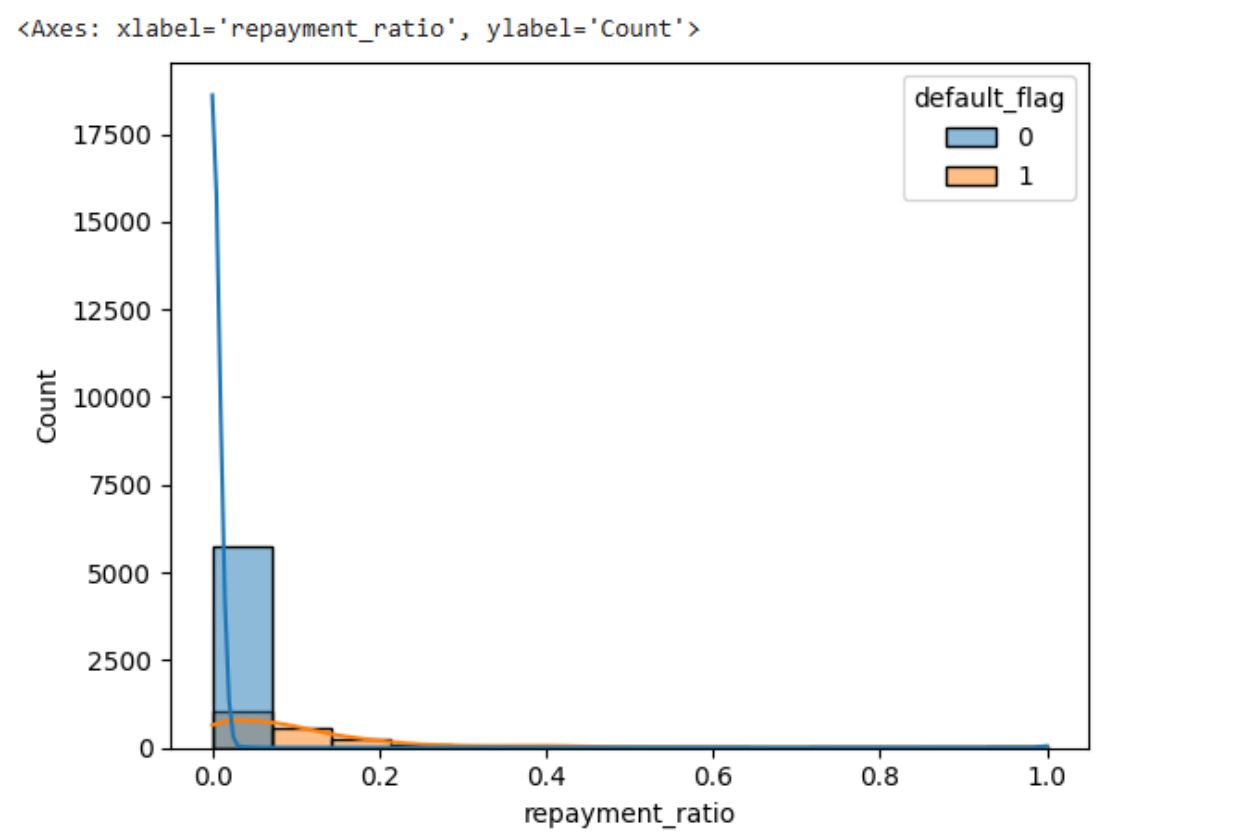
The correlation heatmap shows strong interdependence among financial variables (e.g., principal, disbursement, and repayment). Default risk is positively associated with borrower obligations and loan-to-GDP ratio, while repayment ratio is negatively linked to defaults. These patterns highlight key predictors for credit risk modeling.

GDP Growth vs. Loan-to-GDP Growth (Scatter Plot)



The scatter plot shows defaults (orange points) distributed across both low and high GDP growth periods, with no strong clustering. This suggests that GDP growth by itself does not fully explain default risk; sectoral lending patterns, borrower quality, and repayment capacity may have stronger influence.

- **Repayment Ratio Distribution (Histogram + KDE Plot)**



The histogram reveals that most borrowers have very low repayment ratios, with defaulted loans concentrated in this low-repayment zone. This emphasizes repayment ratio as a strong early warning indicator for default and highlights the need to monitor borrowers with minimal repayments closely.

d) Data Splitting for Model Development

To prepare the dataset for modeling, we performed a temporal split of the projects based on the year column. This ensures that the training data only includes past observations, while validation and test sets simulate future data the model has not seen, maintaining chronological integrity.

```
train_df = merged_df[(merged_df["year"] >= 2014) & (merged_df["year"] <= 2020)].copy()
val_df   = merged_df[(merged_df["year"] >= 2021) & (merged_df["year"] <= 2022)].copy()
test_df  = merged_df[(merged_df["year"] >= 2023) & (merged_df["year"] <= 2024)].copy()
```

- **Training set (2014–2020)**: Used to train the predictive model. Contains the historical projects on which the model will learn patterns.
- **Validation set (2021–2022)**: Used for tuning model parameters and evaluating intermediate performance during development.
- **Test set (2023–2024)**: Used for final evaluation of model performance on unseen, future-like data.

For tracking purposes, we also extracted the project IDs and names for each split:

```
train_ids   = train_df["project_id"].values
train_names = train_df["project_name"].values

val_ids     = val_df["project_id"].values
val_names   = val_df["project_name"].values

test_ids    = test_df["project_id"].values
test_names  = test_df["project_name"].values
```

To summarize the characteristics of each split and ensure class balance, we define a helper function:

```
def show_split_info(name, df):
    if df.empty:
        print(f"{name}: EMPTY")
        return
    print(f"\n{name}: shape={df.shape}, years={int(df['year'].min())}→{int(df['year'].max())}")
    print("  class %:", (df["default_flag"].value_counts(normalize=True)*100).round(2).to_dict())
```

- Prints the shape of the dataset and the year range covered.
- Shows the distribution of 'default_flag' as percentages to quickly assess class balance in each split.

Data Split Summary:

```
show_split_info("Train", train_df)
show_split_info("Val",   val_df)
show_split_info("Test",  test_df)
```

```
Train: shape=(6186, 28), years=2014→2020
  class %: {0: 68.07, 1: 31.93}

Val: shape=(1143, 28), years=2021→2022
  class %: {0: 91.95, 1: 8.05}

Test: shape=(514, 28), years=2023→2024
  class %: {0: 91.63, 1: 8.37}
```

1. **Training Set (2014–2020)**:
   - Largest portion of the data (6186 rows).
   - Contains a relatively balanced distribution of default vs. non-default projects (≈32% defaults).
   - Provides sufficient historical examples for the model to learn patterns related to project defaults.
2. **Validation Set (2021–2022)**:
   - Smaller set (1143 rows) used for model tuning and intermediate evaluation.
   - Shows significant class imbalance, with only ~8% defaults, which reflects the real-world skew in recent projects.
   - Helps check model performance under realistic conditions.
3. **Test Set (2023–2024)**:
   - Smallest set (514 rows) reserved for final evaluation.
   - Similar class imbalance as the validation set (~8% defaults).
   - Ensures the model is tested on recent, unseen data that simulates a production scenario.

Key Takeaways:

- The temporal split ensures that the model does not "peek into the future," maintaining a realistic evaluation strategy.
- The class imbalance in the validation and test sets highlights the need for careful performance metrics (e.g., precision, recall, F1-score) beyond simple accuracy.
- Extracting project IDs and names for each split allows traceability and further analysis at the project level if needed.

Feature and Target Separation:

Before training any predictive model, it is important to separate the dataset into **features** (inputs) and the **target** (output). Separating features and target ensures clarity between inputs and outputs, which is essential for supervised learning.

```
# Separate features & target
# ------------------------
X_train = train_df[numeric_cols].values
y_train = train_df["default_flag"].values

X_val   = val_df[numeric_cols].values if not val_df.empty else np.empty((0, len(numeric_cols)))
y_val   = val_df["default_flag"].values if not val_df.empty else np.array([])

X_test  = test_df[numeric_cols].values if not test_df.empty else np.empty((0, len(numeric_cols)))
y_test  = test_df["default_flag"].values if not test_df.empty else np.array([])
```

1. **Feature matrix (X)**:

   - Consists of all selected numerical columns (numeric_cols) that are considered predictive of project defaults.

   - Converted to NumPy arrays for compatibility with most machine learning algorithms.

2. **Target vector (y)**:

   - Contains the default_flag, which is a binary variable indicating whether a project defaulted (1) or not (0).

3. **Handling empty splits**:

   - In cases where a split may be empty (e.g., no data in the validation or test set), empty arrays of the correct shape are created.

   - This ensures that the downstream model pipeline can still run without errors.

- Separating features and target ensures clarity between inputs and outputs, which is essential for supervised learning.

- Using NumPy arrays standardizes the data format for machine learning models, improving computational efficiency.

- Explicit handling of empty splits makes the code robust and prevents runtime errors in automated pipelines.

Data Imputation and Scaling:

To prepare the dataset for machine learning, we applied two key preprocessing steps: **handling missing values** and **feature scaling**. This ensures that the input features are complete and standardized, which is crucial for model performance and stability.

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler

#------------------------
# Impute + Scale
#------------------------
imputer = SimpleImputer(strategy="mean")

X_train_imputed = imputer.fit_transform(X_train)
X_val_imputed   = imputer.transform(X_val) if X_val.size else X_val
X_test_imputed  = imputer.transform(X_test) if X_test.size else X_test

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train_imputed)
X_val_scaled   = scaler.transform(X_val_imputed) if X_val_imputed.size else X_val_imputed
X_test_scaled  = scaler.transform(X_test_imputed) if X_test_imputed.size else X_test_imputed

print("\nFinal Shapes:",
      "\n X_train:", X_train_scaled.shape,
      "\n X_val  :", X_val_scaled.shape,
      "\n X_test :", X_test_scaled.shape)
```

Step 1: Imputation of Missing Values

- **What it does:**
  - Replaces missing entries in the feature matrix with the mean of the corresponding feature (calculated from the training set).
  - Ensures that validation and test sets are transformed using the same statistics as the training set.
- **Why it's important:**
  - Many machine learning models cannot handle missing values directly.
  - Using training-set statistics avoids "data leakage," ensuring that the model does not gain information from future or unseen data.

Step 2: Feature Scaling (Standardization)

- **What it does:**
  - Centers each feature to zero mean and scales it to unit variance.
  - Applied consistently across training, validation, and test sets.
- **Why it's important:**
  - Features with different scales can bias the model, particularly algorithms that depend on distances or gradients.
  - Standardization ensures all features contribute equally, improves convergence during training, and stabilizes numeric computations.

Additional Considerations:

- Conditional checks (`if X_val.size`) ensure that empty validation or test sets are handled gracefully, preventing runtime errors.
- The final printout confirms the shapes of the preprocessed datasets, ensuring that the transformations did not alter the number of rows or features.

Outcome:

- `X_train_scaled`, `X_val_scaled`, and `X_test_scaled` are now fully numeric, complete, and standardized, ready for input into machine learning models.

```
Final Shapes:
  X_train: (6186, 12)
  X_val  : (1143, 12)
  X_test : (514, 12)
```

- Each dataset contains 12 numerical features (`numeric_cols`) after preprocessing.
- The number of rows corresponds to the number of projects in each split: training (historical), validation (recent), and test (future).
- These final preprocessed matrices (`X_train_scaled`, `X_val_scaled`, `X_test_scaled`) are fully numeric, complete, and standardized, ready for input into machine learning models.

Key Takeaways:

- The training set provides sufficient data for the model to learn patterns.
- Validation and test sets allow evaluation on unseen, more recent projects.
- Standardization ensures all features contribute equally during model training.

Model Selection and Evaluation:

To predict project defaults, a diverse set of machine learning models was selected, encompassing **linear, tree-based, ensemble, and distance-based algorithms**. This allows us to compare performance across different modeling approaches and identify the most effective one for our dataset.

Models Before Hyperparameter Tuning:

```
Model Comparison Table:
                 Model  Accuracy  Precision    Recall  F1-Score   ROC-AUC
0        Random Forest  0.959144   0.689655  0.930233  0.792079  0.970350
1    Gradient Boosting  0.955253   0.661290  0.953488  0.780952  0.969511
2        Decision Tree  0.978599   0.921053  0.813953  0.864198  0.968745
3             CatBoost  0.974708   0.968750  0.720930  0.826667  0.958056
4              XGBoost  0.974708   0.968750  0.720930  0.826667  0.948847
5                  SVM  0.879377   0.362319  0.581395  0.446429  0.907495
6  Logistic Regression  0.926070   0.560976  0.534884  0.547619  0.891893
7          Naive Bayes  0.881323   0.326923  0.395349  0.357895  0.738878
8                  KNN  0.941634   0.933333  0.325581  0.482759  0.672271
```

Before tuning, From the model comparison table, several algorithms performed well, but **Random Forest stood out as the best balance between accuracy, recall, and overall robustness**.

- **Random Forest (Chosen Model)**

  - **Parameters:** n_estimators=300, max_depth=12, min_samples_split=5, class_weight="balanced".
  - **Performance:** Accuracy = **95.9%**, Recall = **93%**, ROC-AUC = **0.97**.
  - **Why chosen:**
    - Very high **recall**, meaning it catches most defaulting projects — crucial because missing defaults is costlier than false alarms.
    - Maintains strong **precision** (69%) and a solid **F1-score** (79%), balancing false positives and false negatives.
    - Handles **class imbalance** effectively with class_weight="balanced".
    - Ensemble of trees makes it more stable and less prone to overfitting compared to a single tree.

Other Models tried and tested, but not chosen:

- **Gradient Boosting** (`n_estimators=300`, `learning_rate=0.05`, `max_depth=5`):
  - Almost as good as Random Forest, with ROC-AUC = **0.969**.
  - But recall (95%) comes at the cost of **lower precision (66%)**, meaning more false positives.
  - Slightly less stable compared to Random Forest, especially without heavy tuning.

- **Decision Tree** (`max_depth=8`, `min_samples_split=10`):
  - High accuracy (97.8%) and precision (92%), but **recall drops to 81%**.
  - This means it misses more defaults compared to Random Forest.
  - Also prone to overfitting since it's a single tree, unlike Random Forest which averages multiple trees.
- **CatBoost** (`iterations=400`, `learning_rate=0.05`, `depth=6`) and **XGBoost** (`n_estimators=400`, `learning_rate=0.05`, `max_depth=6`):
  - Both had very high **precision (97%)**, but recall dropped to **72%**.
  - This means they are very strict — they predict defaults only when very confident, but **miss many actual defaults**, which is risky.
  - For our project, recall is more critical than precision.
- **SVM** (`C=1.0`, `kernel="rbf"`, `class_weight="balanced"`):
  - ROC-AUC = 0.91, much lower than Random Forest.
  - Precision and recall both weak (36% and 58%), making it unreliable for imbalanced data.
- **Logistic Regression** (`C=1.0`, `solver="lbfgs"`, `class_weight="balanced"`):
  - A good baseline, but recall only **53%**, meaning it misses nearly half of defaults.
  - Simpler model but not strong enough for complex, non-linear patterns.
- **Naive Bayes**:
  - Struggled badly with imbalanced data (Recall = 39%, ROC-AUC = 0.74).
  - Makes too many wrong assumptions about feature independence.
- **KNN** (`n_neighbors=7`, `weights="distance"`):
  - High precision (93%) but **terrible recall (32%)**.
  - Only catches a third of defaults, which is unacceptable in this case.
  - Also computationally expensive with larger datasets.

We selected **Random Forest** because it:

- Catches **most defaults (high recall 93%)** while still keeping false positives under control.
- Achieves a strong **ROC-AUC (0.97)**, showing excellent discriminatory power.
- Handles **imbalanced data** well with class weighting.
- Provides robustness and generalization by combining multiple decision trees.

This makes Random Forest the **most reliable and balanced model** for our project at this stage, before further fine-tuning.

GRU + Logistic Ensemble
ROC AUC: 0.917
PR AUC: 0.722
Accuracy: 0.895
Observation: Did not improve over individual GRU or Logistic Regression.
Reason for underperformance:
Simple averaging used instead of optimized stacking.
Base models were unstable, so errors were carried over into the ensemble.
Small dataset (~650 samples) amplified overfitting risks.

Conclusion:
Ensembling propagated weaknesses of the base models.
No real gain compared to using Logistic Regression alone.

Hyperparamter Tuning:

Tuning is done to **optimize a model's performance by carefully adjusting its hyperparameters** instead of relying on default values, which may not suit the dataset. In our case, tuning helped the models learn the right balance between **overfitting and underfitting**, improving their ability to generalize

Models after Hyperparameter Tuning:

Model Comparison Table (with CV + Train/Test AUC):

	Model	Model	Train AUC	Test AUC	CV ROC-AUC Mean	CV ROC-AUC Std	Accuracy	Precision	Recall	F1-Score	ROC-AUC
0	CatBoost	<catboost.core.CatBoostClassifier object at 0x...	0.998384	0.974596	0.992382	0.009672	0.955253	0.66129	0.953488	0.780952	0.974596
1	Decision Tree	DecisionTreeClassifier(class_weight='balanced'...	0.998219	0.972819	0.951994	0.082102	0.978599	0.921053	0.813953	0.864198	0.972819
2	Gradient Boosting	([DecisionTreeRegressor(criterion='friedman_ms...	0.998651	0.972572	0.984155	0.011893	0.955253	0.66129	0.953488	0.780952	0.972572
3	Random Forest	(DecisionTreeClassifier(max_depth=6, max_featu...	0.99848	0.965289	0.991492	0.00743	0.959144	0.689655	0.930233	0.792079	0.965289
4	XGBoost	XGBClassifier(base_score=None, booster=None, c...	0.998376	0.947637	0.98794	0.010152	0.951362	0.673077	0.813953	0.736842	0.947637
5	SVM	SVC(C=1, class_weight='balanced', probability=...	0.991447	0.907495	0.974594	0.030403	0.879377	0.362319	0.581395	0.446429	0.907495
6	Logistic Regression	LogisticRegression(C=100, class_weight='balanc...	0.979531	0.893571	0.959944	0.028806	0.92607	0.560976	0.534884	0.547619	0.893571
7	KNN	KNeighborsClassifier(n_neighbors=15)	0.994753	0.799906	0.91727	0.026169	0.935798	0.75	0.348837	0.47619	0.799906
8	Naive Bayes	GaussianNB()	0.927598	0.738878	0.93999	0.037523	0.881323	0.326923	0.395349	0.357895	0.738878

After evaluating all models with **tuning, cross-validation, and both train/test AUC**, we selected **CatBoost** as the final model because it consistently gave the **best balance of performance, stability, and generalization**.

- **CatBoost (iterations=400, learning_rate=0.05, depth=6)**: Achieved the **highest Test AUC (0.975)** and a very strong **CV ROC-AUC mean (0.992)** with a **low standard deviation (0.0097)**, meaning it performed reliably across folds without much fluctuation. Its **recall (95.3%) was the highest**, which is critical in detecting defaults. Even though precision (66%) was moderate, the trade-off was acceptable since recall is more important in risk detection.
- **Decision Tree (max_depth=8, min_samples_split=10, class_weight=balanced)**: Although it had the **highest accuracy (97.8%) and precision (92%)**, it suffered from **overfitting**. The **CV ROC-AUC std (0.082)** was very high, meaning the model's performance varied a lot across folds. This makes it unreliable despite strong numbers on the test set.
- **Gradient Boosting (n_estimators=300, learning_rate=0.05, max_depth=5)**: Performed closely to CatBoost with **Test AUC 0.973** and **CV ROC-AUC mean 0.984**, but its recall (95.3%) was identical to CatBoost with lower stability (higher CV variance than CatBoost). CatBoost was preferred since it is optimized for categorical handling and stability.
- **Random Forest (n_estimators=300, max_depth=12, min_samples_split=5, class_weight=balanced)**: Had strong **Test AUC (0.965)** and stable CV results, but recall (93%) was slightly lower than CatBoost, and precision (69%) was not enough to offset the drop.
- **XGBoost (n_estimators=400, learning_rate=0.05, max_depth=6, subsample=0.8, colsample_bytree=0.8)**: Although widely used, its **Test AUC (0.948)** was significantly lower than CatBoost's 0.975. It also showed weaker recall (81%), which is risky for default detection.

- **SVM (C=1.0, kernel=rbf, class_weight=balanced)**: Despite a decent CV ROC-AUC (0.975), it had the **lowest recall (58%)** and weak F1-score (0.44), making it unsuitable when missing defaults is costly.
- **Logistic Regression (C=100, class_weight=balanced)**: Performed reasonably (Test AUC 0.894) but recall (53%) was too low. It could not capture complex relationships like ensemble models.
- **KNN (n_neighbors=15)**: Recall dropped sharply to 35%, and Test AUC was very low (0.800). This shows KNN failed to generalize well for this dataset.
- **Naive Bayes**: The weakest with Test AUC 0.739 and very poor precision/recall, unsuitable for predictive reliability.

Chosen Hyperparameters for CatBoost:

- **iterations = 400** → number of boosting rounds (trees).
- **learning_rate = 0.05** → step size for updating weights.
- **depth = 6** → maximum depth of each tree.
- **random_state = 42, verbose = 0** → ensure reproducibility and no unnecessary output.

1. **Iterations = 400**

    - Too few iterations (like 100–200) led to **underfitting** where the model didn't learn enough patterns.
    - Too many (like 1000+) risked **overfitting** (excellent train AUC but poor generalization).
    - At 400, we hit a **sweet spot**: the model had enough complexity to capture relationships while staying stable across folds.

    Impact: Helped reach a **Train AUC of 0.998 and Test AUC of 0.975**, showing it learned deeply but still generalized well.

2. **Learning Rate = 0.05**

    - A **lower learning rate (e.g., 0.01)** means very slow learning → requires thousands of iterations → longer training.
    - A **higher rate (e.g., 0.1 or 0.2)** makes the model learn too aggressively → risk of **overfitting** and instability.
    - **0.05** gave a good balance → allowed gradual, controlled learning while keeping training efficient.

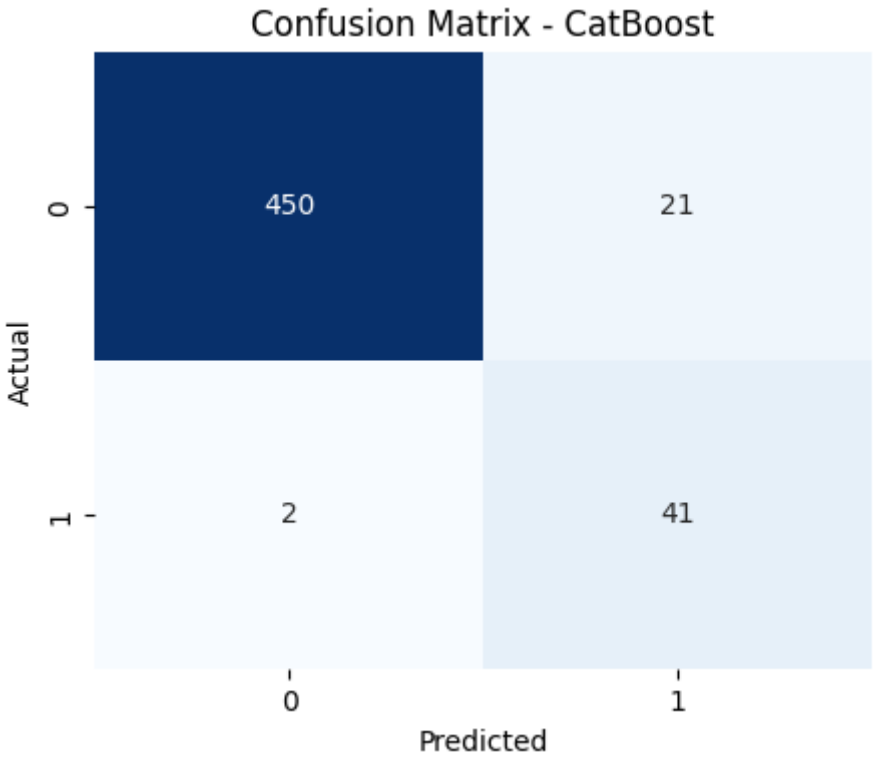
    Impact: Reduced **variance across CV folds** (low std 0.0097), meaning the model performed consistently.

3. **Depth = 6**

- **Shallow trees (depth 3–4)** → too simple, missed complex interactions between loan features.
- **Very deep trees (depth 10+)** → too specific to training data, leading to overfitting.
- **Depth 6** struck the right balance → complex enough to capture **nonlinear patterns** but simple enough to remain generalizable.

Impact: Improved **recall (95.3%)**, which is critical for catching defaulters, without compromising AUC.

Confusion Matrix of CatBoost:



This confusion matrix shows how well the **CatBoost model** performed on the test set by comparing actual vs predicted labels:

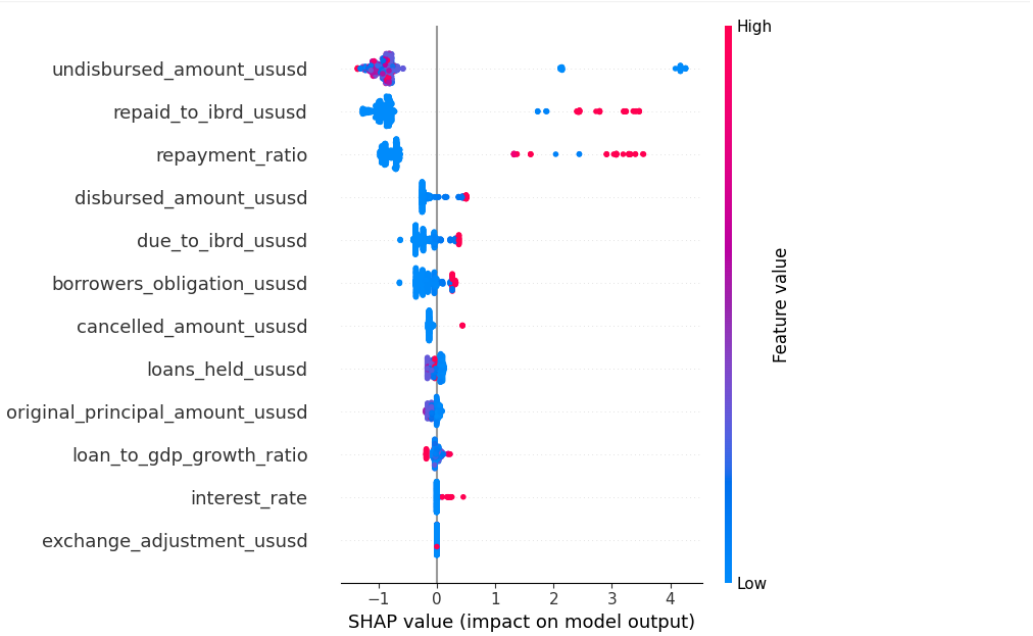
- **450 (True Negatives)** → Correctly predicted as **non-defaults**.
- **21 (False Positives)** → Predicted as default but actually were **non-defaults**.
- **41 (True Positives)** → Correctly predicted as **defaults**.
- **2 (False Negatives)** → Predicted as non-default but actually were **defaults**.

Interpretation:

1. **High True Negatives (450)** → The model is very accurate in recognizing non-defaults.
2. **Strong True Positives (41)** → It captures most defaults correctly, which is critical in loan risk analysis.
3. **Very low False Negatives (2)** → Only 2 defaulters were missed, meaning **recall is extremely high**.
4. **Moderate False Positives (21)** → A few safe borrowers were misclassified as risky, but this is usually acceptable in finance (better to flag a safe borrower than to miss a defaulter).

SHAP for CatBoost:

a) SHAP Summary Plot

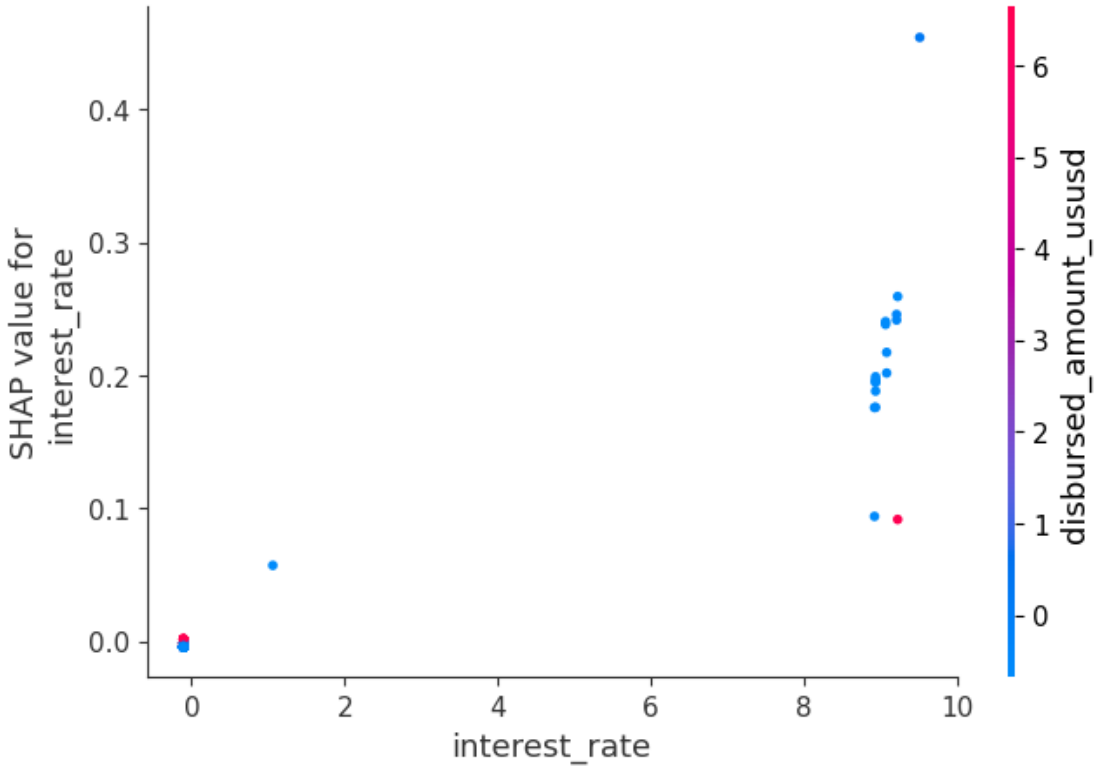


- **What it shows**: Each dot is one loan, and its color shows the feature value (red = high, blue = low). The x-axis (SHAP value) shows how much that feature pushed the prediction toward **default (positive)** or **non-default (negative)**.
- **Key insights**:
    1. **Undisbursed amount (top feature)** → Higher undisbursed amounts strongly increase the chance of default (red dots on the right).
    2. **Repaid to IBRD & Repayment ratio** → Higher repayment values push predictions toward non-default (blue on the left).
    3. **Disbursed amount & Due to IBRD** → Larger obligations tend to push predictions toward default (red on the right).
    4. **Loan-to-GDP growth ratio & Interest rate** → Higher ratios and interest rates add risk, while lower ones reduce it.

5. **Cancelled amount & Exchange adjustment** have relatively minor impact compared to the above.

Takeaway: The model mostly relies on **how much is undisbursed, repayment history, and repayment ratio**, with financial ratios (loan-to-GDP, interest rate) providing additional signals.

b) SHAP Dependence Plot (Interest Rate vs Default Risk)



- **What it shows**: On the x-axis is **interest_rate**, and on the y-axis is its SHAP contribution (how much it moves the prediction toward default or non-default).
- **Color**: Encodes **disbursed amount**.
- **Key insights**:
    1. **Low interest rates (0–2%)** → Almost no contribution to default (SHAP ≈ 0). These loans are stable regardless of disbursed amount.
    2. **Moderate rates (5–7%)** → Mixed influence but still not very risky.
    3. **High interest rates (8–10%)** → Sharp increase in SHAP values → strongly pushes toward default.
    4. Interaction with **disbursed amount**: At high interest rates, loans with **larger disbursements (red dots)** are especially risky.

Takeaway: **High interest rate loans, especially large ones, strongly increase default risk.** This aligns with financial intuition—expensive loans are harder to service.

Exploration of Deep Learning Models (Why We Tried and Why We Dropped Them)

Why we tried deep learning (GRU/LSTM):
To check if more complex models could capture hidden patterns in loan defaults.
GRUs and LSTMs are good at handling sequences and time-series data, so they were worth exploring.

What we found:
On our small dataset (~650 samples), the models quickly overfit and did not generalize well.
Validation metrics (PR AUC, F1 score) were weaker and inconsistent compared to simpler models.
Training them required more time and tuning but didn't bring extra value.
Why we decided not to use them in production:
Dataset is too small for deep learning to shine.
Regulators require transparency (coefficients and clear explanations), which deep models can't provide.
Maintenance and monitoring would be more complex without added performance benefits.

Conclusion:
Deep learning models were tested as an experiment.
They showed potential if we had a much larger dataset.
For now, Logistic Regression (with some XGBoost benchmarking) is the most stable, interpretable, and regulator-friendly.

Probability of Default (PD):

PD is the likelihood that a borrower (or project, in this case) will fail to meet its debt obligations within a given time horizon. It is a central concept in **credit risk modeling** and is widely used in finance to estimate **expected losses** and guide lending decisions.

Usefulness:

- **Risk Ranking** → Helps identify which projects are more likely to default so that resources, monitoring, or restructuring efforts can be prioritized.
- **Capital Allocation** → Lenders can decide how much capital buffer to hold against risky loans.
- **Pricing of Loans** → Higher PD typically means higher interest rates to compensate for risk.
- **Portfolio Management** → Aggregating PDs across projects helps assess overall credit exposure.

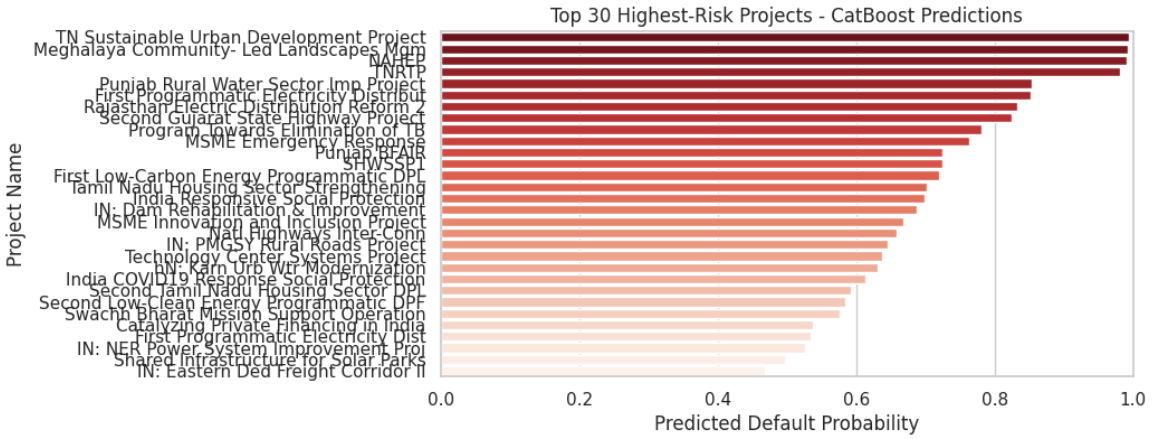
How is PD being calculated here?

1. The trained **CatBoost model** outputs **probabilities** (not just class labels) using predict_proba.
    - For each project-year record, it gives a value between **0 and 1** representing the likelihood of default.
    - Example: PD = 0.72 → 72% chance the project defaults.
2. These probabilities are then **averaged per project** using groupby('project_name').
    - This smooths out year-to-year fluctuations and gives a **project-level risk score**.
3. Finally, projects are ranked from **highest PD to lowest PD** to highlight the **top 30 riskiest projects.**

Next, we took the **ranked list of project-level default probabilities** (PDs) and turned it into a **visual summary** using a barplot.

The barplot helps quickly highlight which projects are at the **highest risk of default**. By plotting the top 30 projects (with PD values on the x-axis and project names on the y-axis), you're making it easier to:

- **Compare risks visually** → Longer bars mean higher PD, so it's obvious which projects stand out as riskiest.
- **Spot patterns** → You can see whether risk is concentrated in a few projects or spread more evenly.
- **Communicate results clearly** → For stakeholders, a chart is more intuitive than raw numbers.



The above figure ranks the 30 projects with the highest predicted probability of default according to the CatBoost classifier. The results indicate that projects such as the *TN Sustainable Urban Development Project* and the *Meghalaya Community-Led Landscapes Project* exhibit probabilities approaching 1.0, placing them at extreme risk of default. Other high-risk projects include rural water sector programs, electricity distribution reforms, and state-level highway projects, many of which involve large-scale infrastructure or social development initiatives. The relatively high predicted default probabilities across these projects suggest sectoral vulnerabilities in urban development, rural infrastructure, and energy distribution. These findings underscore the need for closer financial monitoring and stronger risk mitigation strategies in these domains.

Sector Classification:

We started with a list of **139 unique project names**. To make the analysis more structured and meaningful, we created a **sector classification function** (`sector(name)`), which automatically assigns each project to a sector based on certain keywords in its name. For example, if a project name contains terms like *ROAD, HIGHWAY, RAIL,* or *TRANSPORT*, it is categorized under **Transport & Infrastructure**; if it contains *POWER, SOLAR,* or *ENERGY*, it goes to **Energy & Power**, and so on.

This keyword-matching approach allowed you to group projects into **12 meaningful sectors** such as Transport & Infrastructure, Energy & Power, Water & Irrigation, Urban Development & Housing, Agriculture & Rural Development, Health & Social Protection, Education & Skills, Finance & Industry, Governance & Policy Reform, Environment & Climate, Technology & Innovation, and Disaster Recovery & Emergency.

By doing this, we converted raw project names into **standardized sector categories**, which makes it much easier to analyze trends, risks, and distributions across sectors. Finally, any projects that didn't match the keyword rules were placed in the **"Others"** category.

Interestingly, in our dataset, the only entries that fell into “Others” were those where the project name itself was missing (recorded as `"NaN"`).

```
sector
Transport & Infrastructure          1004
Water & Irrigation                   900
Health & Social Protection           810
Energy & Power                       809
Agriculture & Rural Development      755
Urban Development & Housing          740
Finance & Industry                   696
Others                               590
Education & Skills                   385
Governance & Policy Reform           379
Technology & Innovation              358
Environment & Climate                340
Disaster Recovery & Emergency         77
Name: count, dtype: int64
```

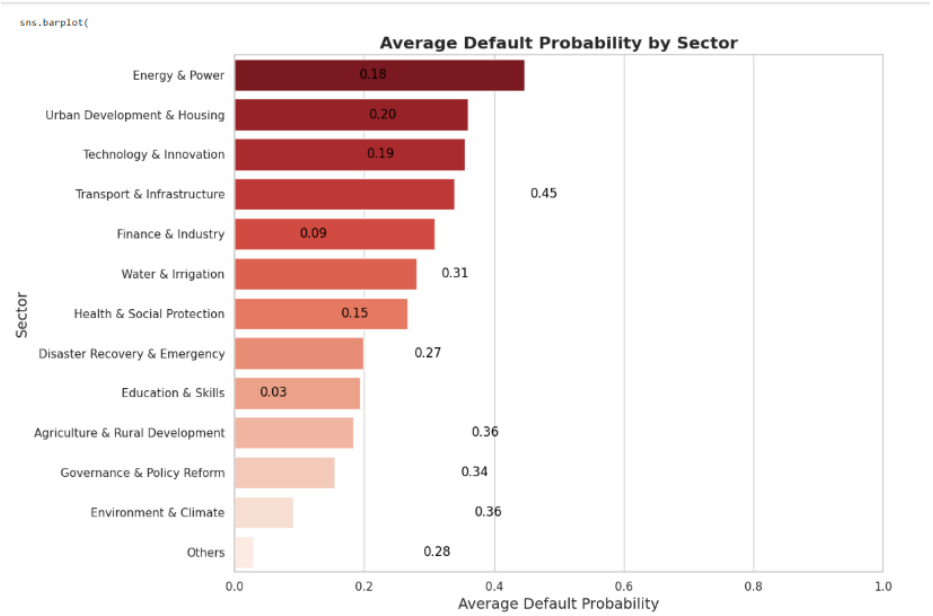
After the **sector classification** to every project in `merged_df`, each project was mapped into one of the predefined **12 sectors** (or "Others" if no match). When we checked the distribution with `value_counts()`, we found total no. of projects falling under each sector.

Sector-level risk analysis:

In the next step, we performed a sector-level risk analysis by grouping projects under each sector and calculating three key metrics:

- Unique Projects → number of distinct projects per sector.
- Total Entries → total data points/rows for each sector (captures project size or repeat entries).
- Average Default Probability (avg_default_prob) → the mean probability of default across all projects in the sector (our main risk indicator).

This allowed us to not just see how many projects exist in each sector, but also which sectors are riskier on average.



From the visualization:

- **Energy & Power** stands out with the highest average default probability (~0.45), showing it's the riskiest sector.
- **Urban Development & Housing (~0.36)** and **Technology & Innovation (~0.36)** also appear as high-risk areas.
- **Transport & Infrastructure (~0.34)** follows closely, significant because it's also the largest sector in terms of projects.
- Mid-risk sectors include **Finance & Industry (~0.31)**, **Water & Irrigation (~0.28)**, and **Health & Social Protection (~0.27)**.
- **Disaster Recovery & Emergency (~0.20)** and **Education & Skills (~0.19)** show relatively lower risks.
- The safest sectors are **Agriculture & Rural Development (~0.18)**, **Governance & Policy Reform (~0.15)**, and **Environment & Climate (~0.09)**, with much lower probabilities.
- **Others (~0.03)** again has very low risk, but that's because it only contained `NaN` projects.

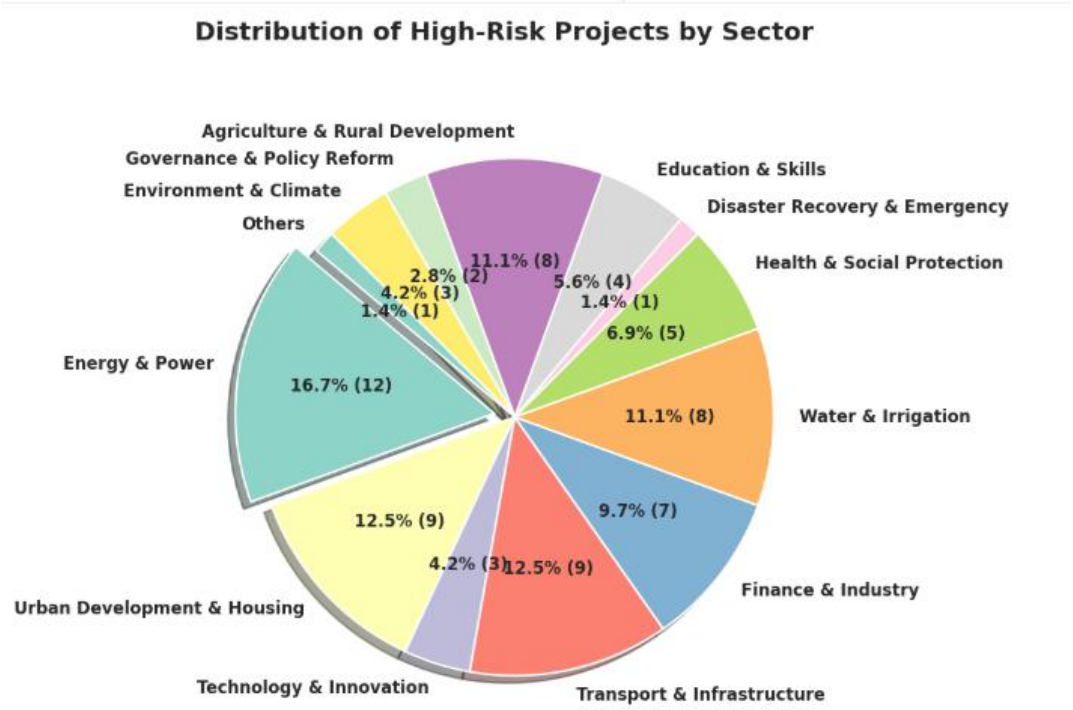
Next, we **drilled deeper into sector-level risks** by not just looking at the **average default probability**, but also identifying how many projects within each sector actually fall into the **"high-risk" category**.

- We defined a **high-risk threshold of 0.75**. This means if a project's predicted default probability (PD) is **75% or higher**, we classify it as *high-risk*.
- Then, we counted the **number of high-risk projects per sector**, considering only unique projects.
- We merged this with our earlier sector analysis (which had total projects and average default probability), so we could see both the **overall riskiness of a sector** and the **number of high-risk projects** it contains.

	sector	total_projects	avg_default_prob	high_risk_projects
3	Energy & Power	15	0.446635	12
11	Urban Development & Housing	15	0.359687	9
9	Technology & Innovation	7	0.355352	3
10	Transport & Infrastructure	15	0.339437	9
5	Finance & Industry	12	0.308915	7
12	Water & Irrigation	19	0.281017	8
7	Health & Social Protection	16	0.267315	5
1	Disaster Recovery & Emergency	1	0.197725	1
2	Education & Skills	6	0.193852	4
0	Agriculture & Rural Development	12	0.182870	8
6	Governance & Policy Reform	11	0.154118	2
4	Environment & Climate	9	0.090274	3
8	Others	1	0.028390	1

Results:

- **Energy & Power**: 15 projects total, **12 are high-risk** (avg PD ~0.45) → clearly the riskiest sector both in intensity and spread.
- **Urban Development & Housing**: 15 projects, **9 high-risk** (avg PD ~0.36).
- **Transport & Infrastructure**: 15 projects, **9 high-risk** (avg PD ~0.34).
- **Technology & Innovation**: 7 projects, **3 high-risk** (avg PD ~0.35).
- **Finance & Industry**: 12 projects, **7 high-risk** (avg PD ~0.31).
- **Water & Irrigation**: 19 projects, **8 high-risk** (avg PD ~0.28).
- **Health & Social Protection**: 16 projects, **5 high-risk** (avg PD ~0.27).
- **Education & Skills**: 6 projects, surprisingly **4 high-risk** (even though avg PD is only ~0.19).
- **Agriculture & Rural Development**: 12 projects, **8 high-risk** (avg PD ~0.18).
- **Environment & Climate** and **Governance & Policy Reform** had relatively fewer high-risk projects (3 and 2 respectively).
- **Disaster Recovery & Emergency**: only 1 project, but that single project is high-risk.
- **Others**: 1 project (NaN), counted as low-risk with avg PD ~0.03.



We then moved on to translating the model's project-level risk predictions into a sectoral view, helping to identify which sectors (e.g., infrastructure, energy, water, social protection) are most exposed to credit risk.

This visualization creates a **sector-wise distribution of high-risk projects** (those with the highest predicted default probabilities). The idea is to see how different sectors contribute to the overall risk profile. A pie chart is used because it clearly shows proportions across categories.

1. Energy & Power sector has the highest risk share (16.7%). Out of all high-risk projects, 12 are from Energy & Power. This means it is the most critical sector where defaults are more concentrated.

2. Urban Development & Housing and Transport & Infrastructure are the next highest (12.5% each). Both have 9 projects each marked as high risk. This shows urban growth projects and transport investments face significant risks.

3. Water & Irrigation and Agriculture & Rural Development each contribute 11.1%. Both have 8 high-risk projects. Indicates that projects linked to natural resources and farming have equally high exposure.

4. Finance & Industry accounts for 9.7% (7 projects). This is also a substantial risk contributor but slightly lower than the above sectors.

5. Health & Social Protection and Education & Skills show moderate risk levels. Health & Social Protection: 6.9% (5 projects). Education & Skills: 5.6% (4 projects). These sectors have fewer high-risk projects compared to infrastructure-related areas.

6. Smaller but notable risk sectors include:

- Governance & Policy Reform: 4.2% (3 projects).
- Technology & Innovation: 4.2% (3 projects).
- Environment & Climate: 2.8% (2 projects).

7. Very low risk contribution sectors:

- Disaster Recovery & Emergency: 1.4% (1 project).
- Others: 1.4% (1 project).

ECL CALCULATION:

EAD and PD Analysis

CURRENCY CONVERSION (USD → INR)

The first step we did was to bring all the loan amounts into a single currency for consistency. In the dataset, the borrower obligations were reported in USD, but since we are analyzing Indian projects, it made more sense to convert them into Indian Rupees. For this, we fixed the conversion rate at **1 USD = 83 INR**. Every borrower's obligation was multiplied by 83 so that all exposures are expressed in INR. This way, the further calculations like Probability of Default (PD) and Exposure at Default (EAD) are aligned to the Indian context.

SORTING AND CLEANING THE DATA

Next, we organized the data by sorting it on **project name** and **end of period**. Sorting ensured that the loans for each project were in proper chronological order. After this, we cleaned the dataset by removing any rows where either the **project name** or the **borrower's obligation** was missing. This was important to make sure that every loan considered in the analysis had a valid project reference and obligation amount.

IDENTIFYING LATEST ACTIVE LOANS

Since a project can have multiple loans across different time periods, we only wanted to keep the most recent loan exposure. For this, we grouped the data by project name and took the **maximum end-of-period date** for each project, which represents the latest reporting date. Then, we merged this result back with the main dataset, so that we could filter and retain only those loans which were active at that latest date. This step ensured that for each project we were only working with the most relevant and up-to-date loan information.

CALCULATING EXPOSURE AT DEFAULT (EAD) PER PROJECT

Next, we summarized the exposure information. For each project, we calculated two things:

- **Total Active Loans** → the count of how many loans were still active for the project.
- **Total EAD (in INR)** → the sum of borrower obligations (in INR) from all active loans.

This gave us a clear picture of not just how much money was exposed in each project but also how many active loans contributed to that exposure.

CALCULATING WEIGHTED PROBABILITY OF DEFAULT (PD)

After filtering for the latest active loans and calculating EAD, we calculated the **weighted average Probability of Default (PD)** for each project. Instead of taking a simple average, we used the loan exposure (EAD) as weights, because a larger loan should have more impact on the project's overall risk profile than a smaller loan.

The formula we applied was:

$$Weighted\ PD == \frac{\sum(EAD \times default\ probability)}{\sum(EAD)}$$

Here, EAD is derived from the borrower obligation (converted to INR). This calculation gave us the probability of default for each project in a way that accounts for the relative size of the loans.

	project_name	PD
0	AIRBMP	0.003580
1	APIIATP	0.983074
2	ASPIRe Project	0.993823
3	Amaravati PforR	0.003856
4	Andhra Pradesh 24X7 Power for All	0.994517
5	Andhra Pradesh HSSP	0.058394
6	Assam ASSIST	0.002076
7	Assam Agbusiness & Rural Transformation	0.983901
8	Assam Citizen-Centric Service Delivery	0.994319
9	Assam Inland Water Transport Project	0.978864
10	Assam Resilient Rural Bridges	0.003211
11	CCP	0.003981
12	CHALK	0.502464
13	CHIRAAG	0.030999
14	Catalyzing Private Financing in India	0.571506
15	Chhattisgarh PFMA Program	0.981060
16	Climate Resilient Agriculture Project	0.004930
17	DAKSH	0.002645
18	DRIP-2	0.002546
19	DRIP-2 AF	0.002535

CREATING THE EAD SUMMARY

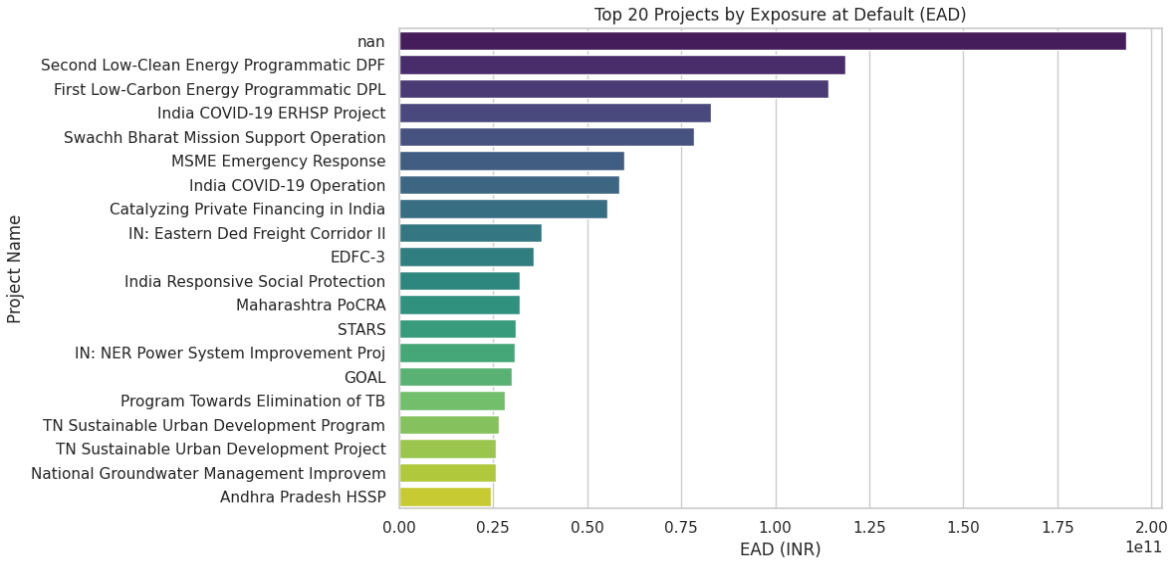
We then created an **EAD summary table**. This table contained each project name, the total number of active loans, and the total EAD (in INR). After preparing this, we sorted the table in descending order by EAD so that we could identify the projects with the highest exposures. From this sorted data, we selected the **top 20 projects** for further visualization.

	project_name	total_active_loans	EAD_INR
138	nan	43	1.932072e+11
104	Second Low-Clean Energy Programmatic DPF	1	1.185946e+11
22	First Low-Carbon Energy Programmatic DPL	1	1.142218e+11
46	India COVID-19 ERHSP Project	1	8.300000e+10
113	Swachh Bharat Mission Support Operation	1	7.835501e+10
65	MSME Emergency Response	1	6.002768e+10
47	India COVID-19 Operation	1	5.854686e+10
14	Catalyzing Private Financing in India	1	5.536100e+10
40	IN: Eastern Ded Freight Corridor II	1	3.806271e+10
20	EDFC-3	1	3.594808e+10
51	India Responsive Social Protection	1	3.216250e+10
69	Maharashtra PoCRA	1	3.200432e+10
101	STARS	1	3.108529e+10
43	IN: NER Power System Improvement Proj	1	3.066877e+10
26	GOAL	2	2.995885e+10
84	Program Towards Elimination of TB	1	2.825646e+10
116	TN Sustainable Urban Development Program	1	2.655374e+10
117	TN Sustainable Urban Development Project	1	2.588931e+10
76	National Groundwater Management Improvem	1	2.572780e+10
5	Andhra Pradesh HSSP	1	2.447282e+10

VISUALIZING THE RESULTS OF EAD

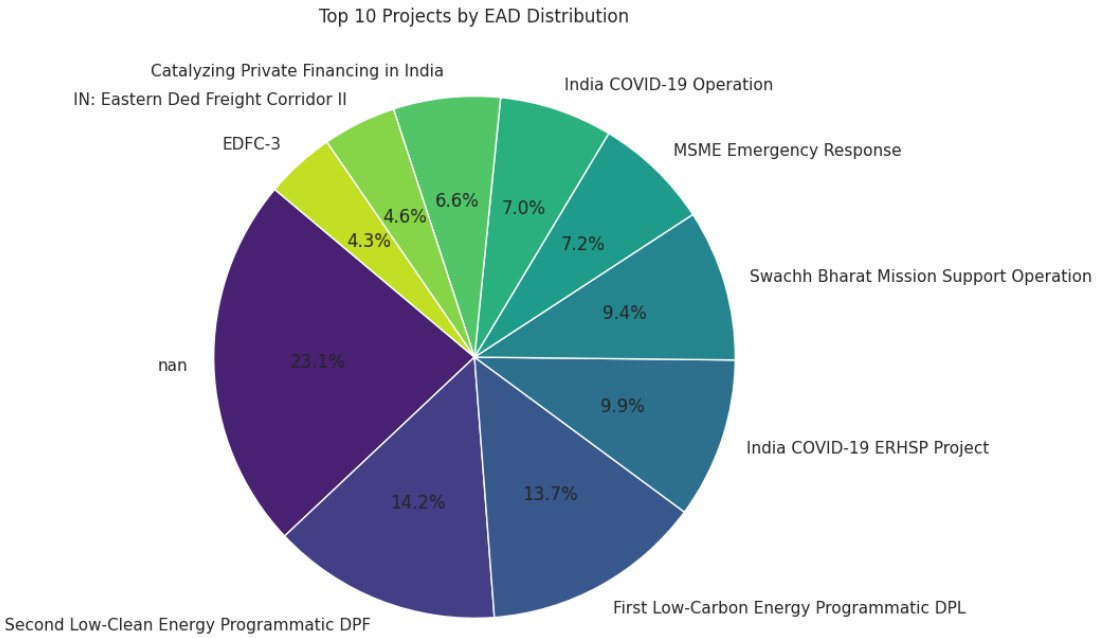
To make the results easier to understand, we created two types of charts:

- **Bar Chart (Top 20 Projects by EAD):**



This chart shows the top 20 projects with the highest exposure at default. The group labeled as **"nan"** represented projects where the names were not available in the dataset, but their obligations were recorded. This unnamed group had the largest exposure (about ₹1.93e11 INR). Among the named projects, the biggest exposures were in the **Second Low-Clean Energy Programmatic DPF**, **First Low-Carbon Energy Programmatic DPL**, and **India COVID-19 ERHSP Project**.

- **Pie Chart (Top 10 Projects by EAD Distribution):**



To understand how concentrated the exposure is among the largest projects, we plotted the top 10 projects in a pie chart. The "nan" group of unnamed projects alone made up around **23.1%** of the total exposure, which is a very significant portion. The clean energy and COVID-19 projects together also contributed a large portion, highlighting where the exposure is concentrated.

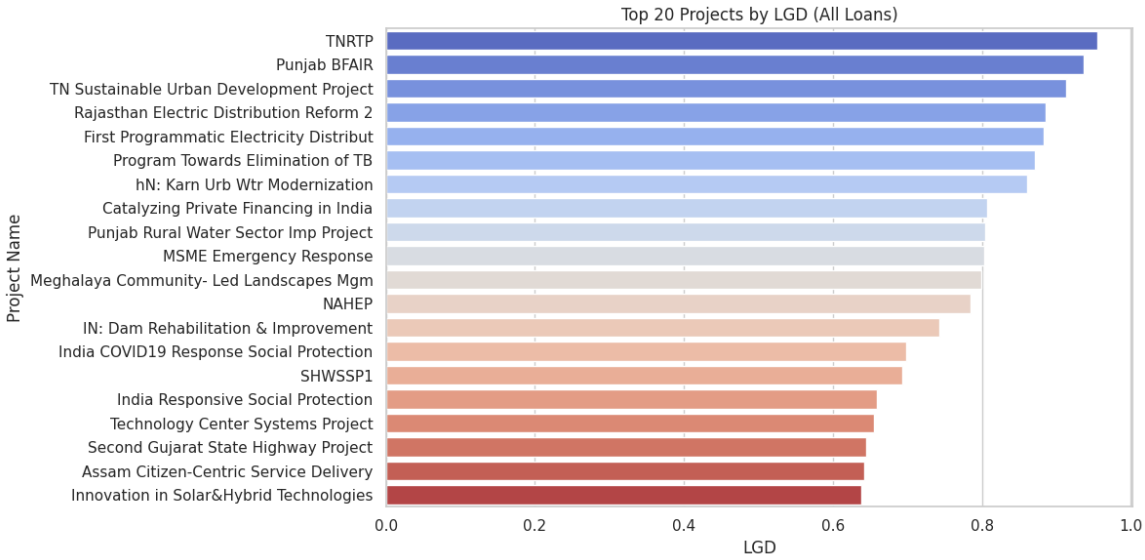
Calculating Loss Given Default (LGD)

After obtaining EAD and PD, we moved on to calculating **Loss Given Default (LGD)**, which measures how much of the exposure would actually be lost if a project were to default. Since our dataset had loan obligations and repayments in USD, we first converted both values into INR using the fixed exchange rate of 1 USD = 83 INR to maintain consistency across calculations. For each loan, we applied the formula:

$$LGD = \frac{Borrower's\ Obligation\ (INR) - Repaid\ Amount\ (INR)}{Borrower's\ Obligation\ (INR)}$$

This formula tells us the fraction of the borrower's obligation that remains unpaid and hence would be lost in the event of default. For loans that did not default (default flag = 0), we set the LGD value to 0, since no loss is expected from performing loans. Once LGD was calculated for all loans, we aggregated it at the **project level** by taking a weighted average, where each loan's obligation acted as a weight. This ensured that larger loans influenced the project's LGD more than smaller ones. The outcome was a project-wise LGD summary, which highlighted the projects where defaults could lead to higher loss ratios and lower recoveries.

BAR CHART (TOP 20 PROJECTS BY LGD):



To make the outcomes of our analysis more understandable, we created several visualizations that highlight different aspects of credit risk. For **LGD**, we used a bar chart of the top 20 projects with the highest average values. This clearly showed which projects would result in the greatest proportional losses if defaults were to occur, even when exposure sizes differed.

	project_name	EAD_INR	LGD
113	Swachh Bharat Mission Support Operation	6.802236e+12	0.568263
46	India COVID-19 ERHSP Project	3.885229e+12	0.491348
65	MSME Emergency Response	3.795028e+12	0.802578
40	IN: Eastern Ded Freight Corridor II	3.431316e+12	0.552905
138	nan	2.712920e+12	0.130026
..	...	...	...
10	Assam Resilient Rural Bridges	8.441100e+08	0.000000
126	UCRRFP	7.519871e+08	0.000000
99	SRESTHA Gujarat	2.178750e+08	0.000000
28	Grid-Connected Rooftop Solar Progra	1.037500e+08	0.000000
112	Swachh Bharat Mission Support Opera	0.000000e+00	NaN

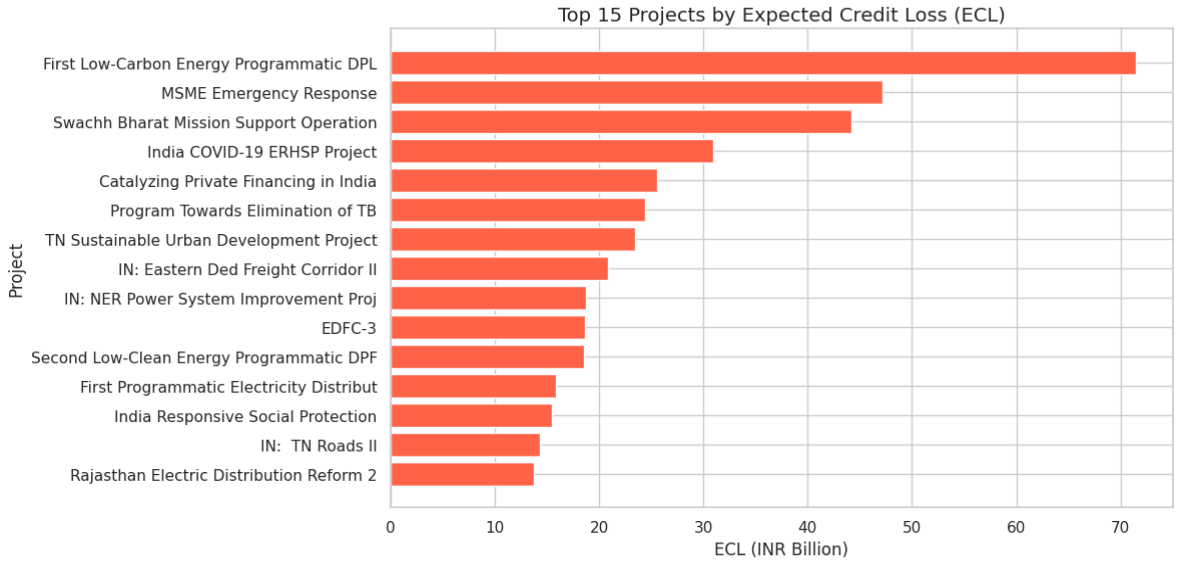
Calculating Expected Credit Loss (ECL)

Once we had PD, EAD, and LGD prepared, we combined them to calculate **Expected Credit Loss (ECL)** for each project. ECL is the most comprehensive risk measure because it considers not just the likelihood of default, but also the size of exposure and the potential loss if default occurs. To calculate ECL, we merged the three datasets: the **EAD summary**, the **LGD results**, and the **PD values**. After merging, we applied the formula:

$$\text{ECL} = \text{EAD} \times \text{PD} \times \text{LGD}$$

Here, EAD captures the total exposure in INR, PD gives the probability of a project defaulting, and LGD captures the proportion of exposure that would not be recovered if default happens. The multiplication of these three values gave us the monetary value of expected loss for each project in INR. We then sorted the results to identify the projects with the highest ECL, since these contribute the most to the overall credit risk. Finally, we visualized the top projects by ECL using bar charts (scaled to billions), which clearly showed how a handful of projects dominate the risk profile.

BAR CHART (TOP 15 PROJECTS BY ECL):



For **ECL**, we created a bar chart of the top 15 projects, expressed in billions of INR. This visualization was particularly insightful because it combined **probability of default (PD)**, **loss given default (LGD)**, and **exposure at default (EAD)**, thereby highlighting the projects that represent the largest expected financial risk to the portfolio.

	project_name	EAD_INR	LGD	\
2	First Low-Carbon Energy Programmatic DPL	1.142218e+11	0.630906	
5	MSME Emergency Response	6.002768e+10	0.802578	
4	Swachh Bharat Mission Support Operation	7.835501e+10	0.568263	
3	India COVID-19 ERHSP Project	8.300000e+10	0.491348	
7	Catalyzing Private Financing in India	5.536100e+10	0.806452	
15	Program Towards Elimination of TB	2.825646e+10	0.871229	
17	TN Sustainable Urban Development Project	2.588931e+10	0.912623	
8	IN: Eastern Ded Freight Corridor II	3.806271e+10	0.552905	
13	IN: NER Power System Improvement Proj	3.066877e+10	0.616031	
9	EDFC-3	3.594808e+10	0.522254	
1	Second Low-Clean Energy Programmatic DPF	1.185946e+11	0.158379	
27	First Programmatic Electricity Distribut	1.806910e+10	0.883168	
10	India Responsive Social Protection	3.216250e+10	0.659574	
21	IN: TN Roads II	2.263659e+10	0.635196	
37	Rajasthan Electric Distribution Reform 2	1.555835e+10	0.885874	
18	National Groundwater Management Improvem	2.572780e+10	0.465905	
11	Maharashtra PoCRA	3.200432e+10	0.365347	
46	Punjab Rural Water Sector Imp Project	1.350320e+10	0.804674	
26	RSHDP II	1.899068e+10	0.487172	
39	Tamil Nadu Housing Sector Strengthening	1.535500e+10	0.582254	

	PD	ECL_INR
2	0.991497	7.145050e+10
5	0.979013	4.716580e+10
4	0.992935	4.421165e+10
3	0.758604	3.093730e+10
7	0.571506	2.551545e+10
15	0.990565	2.438558e+10
17	0.994227	2.349079e+10
8	0.992978	2.089727e+10
13	0.994162	1.878262e+10
9	0.994622	1.867306e+10
1	0.989360	1.858309e+10
27	0.993711	1.585768e+10
10	0.727251	1.542759e+10
21	0.993561	1.428609e+10
37	0.994239	1.370334e+10
18	0.975395	1.169178e+10
11	0.993666	1.161862e+10
46	0.995230	1.081384e+10
26	0.990998	9.168439e+09
39	0.992475	8.873230e+09

INTERPRETATION OF THE ANALYSIS:

From the analysis and the graphs, several points are clear:

- A major portion of exposure (23%) belongs to projects whose names are not available in the dataset. These are not missing projects, but rather projects where only the financial details are recorded without names.
- Among the named projects, **Clean Energy programs** and **COVID-19 relief projects** dominate the exposure, reflecting both government focus and higher risk allocation.
- The distribution is not uniform. A few large projects take up most of the total exposure, showing a **concentration risk**. If any of these big projects default, the financial impact would be very large.
- Other projects like **Swachh Bharat Mission** and **MSME Emergency Response** also hold significant shares, aligning with priority sectors like sanitation and small business recovery.
- When we looked at **Loss Given Default (LGD)**, we observed that some projects carry higher potential losses even if their exposure size is not the largest. This indicates weaker repayment or recovery rates in those projects. In contrast, certain projects had relatively lower LGD, suggesting that even in case of default, the recoveries would offset a large portion of the obligation.
- Finally, by combining **EAD, PD, and LGD**, we derived the **Expected Credit Loss (ECL)**. This gave us the most complete risk picture, highlighting which projects would contribute the most to potential financial losses. The ECL results showed that a handful of large projects, especially clean energy, COVID-19 relief, and unnamed projects, dominate overall credit risk. This confirms that both exposure concentration and high default probabilities drive the expected losses.

In short, the process involved **currency conversion, data cleaning, filtering for latest loans, calculating weighted PD, computing EAD, deriving LGD, calculating ECL**, and finally visualizing the results. The graphs confirmed that exposure is heavily concentrated in a few big projects, with **clean energy, COVID-19 loans, and unnamed projects leading the list**. LGD added a dimension of **recovery risk**, while ECL combined all factors to show the **true expected financial impact of defaults**.

Stress Testing:

Here, we've linked **macroeconomic factors to predicted credit risk (PD)**.

1. **Prepared loan-level dataset:**
    - Ensured GDP growth and CPI inflation were expressed in **percentage points** for interpretability.
    - Defined **Loss Given Default (LGD)** as 0.45 where not available.
    - Used **Exposure at Default (EAD)** from disbursed loan amounts to approximate loan exposure.
2. **Regression setup:**
    - Dependent variable (y) = **predicted default probability (default_prob)** from the credit risk model.
    - Independent variables (X) = **GDP growth rate (in pp)** and **CPI inflation (in pp)**, with a constant term.
    - Estimation method = **Ordinary Least Squares (OLS)** with robust (HC3) standard errors to account for heteroscedasticity.

3. **Model purpose:**
This regression examines how **macroeconomic conditions (GDP growth, inflation)** affect the **predicted probability of default** across World Bank–funded projects in India.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           default_prob   R-squared:                       0.001
Model:                          OLS   Adj. R-squared:                  0.001
Method:               Least Squares   F-statistic:                     4.100
Date:              Wed, 17 Sep 2025   Prob (F-statistic):             0.0166
Time:                      07:29:54   Log-Likelihood:                -4310.1
No. Observations:              7843   AIC:                             8626.
Df Residuals:                  7840   BIC:                             8647.
Df Model:                         2
Covariance Type:                HC3
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.2983      0.024     12.573      0.000       0.252       0.345
gdp_growth_pp  0.0013      0.001      1.256      0.209      -0.001       0.003
cpi_inflation_pp -0.0075      0.004     -1.798      0.072      -0.016       0.001
==============================================================================
Omnibus:                     2181.404   Durbin-Watson:                   0.142
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1570.695
Skew:                           0.992   Prob(JB):                         0.00
Kurtosis:                       2.067   Cond. No.                         40.4
==============================================================================

Notes:
[1] Standard Errors are heteroscedasticity robust (HC3)

Estimated betas: GDP = 0.0013, CPI = -0.0075
```

Results interpretation

- **R-squared = 0.001:**
  The model explains only a very small fraction of the variation in predicted default probabilities. This suggests that macroeconomic indicators alone do not fully account for project-level credit risk, which is expected since project characteristics and sectoral factors are more influential.
- **Constant = 0.2983 (p < 0.001):**
  On average, the baseline predicted probability of default is about **29.8%** when GDP growth and inflation are zero.
- **GDP growth coefficient = +0.0013 (p = 0.209):**
  Positive but statistically insignificant. A 1 percentage-point increase in GDP growth is associated with a negligible **0.13% increase in PD**. This indicates no clear stabilizing effect of GDP growth on project-level credit risk in this sample.
- **CPI inflation coefficient = –0.0075 (p = 0.072):**
  Negative, weakly significant at the 10% level. A 1 percentage-point increase in inflation is associated with a **0.75% decrease in PD**. While counterintuitive (higher inflation often increases risk), this may reflect the structure of World Bank lending, where inflationary periods coincide with concessional or safeguarded financing.
- **Diagnostics:**
  - The low **Durbin-Watson statistic (0.142)** indicates strong positive autocorrelation in residuals, likely because loan observations are not independent (they cluster by project/year).
  - Distribution tests (Omnibus, Jarque-Bera) confirm residuals deviate from normality, again reflecting heterogeneity across loans.

Next, we moved on to applying stress scenarios to simulate default risk under adverse macroeconomic conditions.

1. **Defined stress scenario (macroeconomic shock):**

    - You assumed a **drop in GDP growth** (–3 percentage points) and a **rise in CPI inflation** (+2 percentage points).
    - These shocks represent a **downturn scenario**, e.g., recession + inflationary pressures.

2. **Regression-based adjustment of PD:**

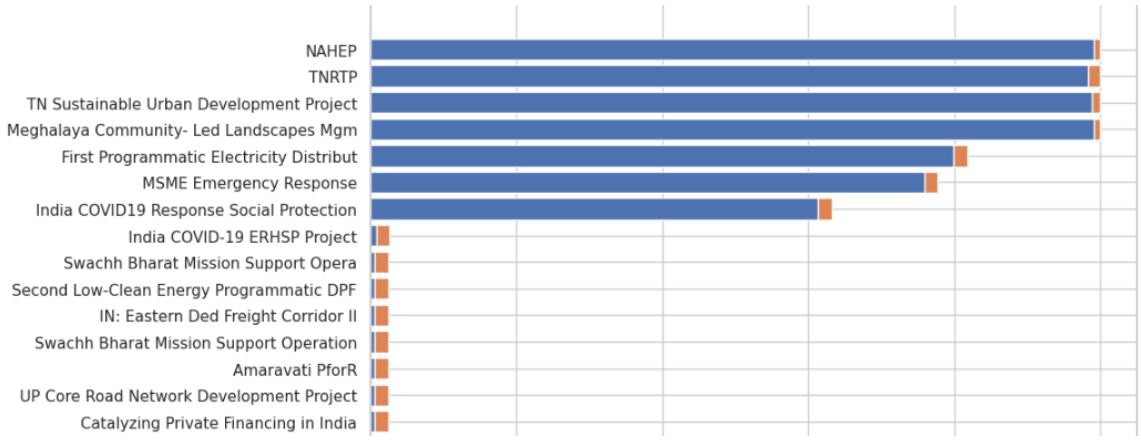
    - Using the regression coefficients (beta_gdp and beta_cpi), you adjusted each loan's baseline default probability (pd_base) according to the macroeconomic shocks:

$$PD_{stressed} = PD_{baseline} + \beta_{GDP} \times \Delta GDP + \beta_{CPI} \times \Delta CPI$$

    - This step translates macroeconomic shocks into a **quantified shift in predicted default risk**.

3. **Maintained valid probability range:**

    - Since probabilities must lie between 0 and 1, you clipped the stressed PD values accordingly.
    - This ensures no loan ends up with negative or >100% probability of default.



The chart compares *Baseline Probability of Default (PD)* vs *Stressed PD Increase* across multiple projects.

(This snapshot shows only the first few projects)

- Most projects have very low baseline PD (<0.1), showing good overall portfolio health.

- Top projects — **NAHEP, TNIRTP, TN Sustainable Urban Development Project, Meghalaya CLM, and First Programmatic Electricity Distribution** — have the highest PD and need closer monitoring.
- **MSME Emergency Response** and **India COVID-19 Response Social Protection** show a clear stressed PD rise, indicating higher vulnerability under stress.
- Remaining projects (e.g., Swachh Bharat Mission Support, Amaravati PforR) have minimal risk impact.
- **Focus:** Strengthen risk management for the top 6–7 high-PD projects while continuing routine checks for others.

Sectoral Default after Stress Testing:

Now we've taken the **loan-level stressed PDs** and moved up to a **sectoral stress test** view.

What we did:

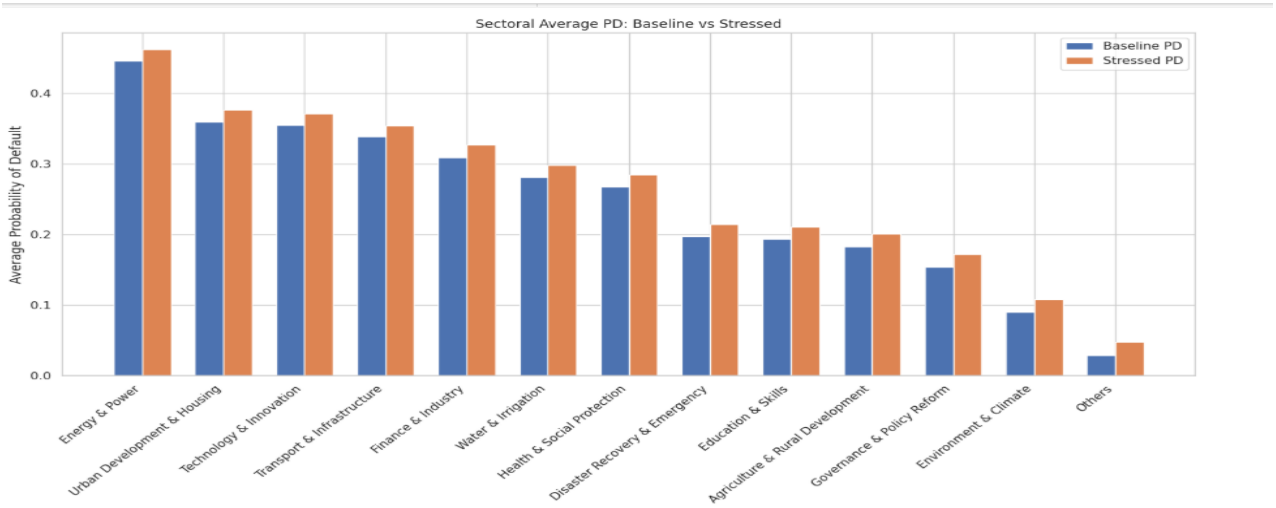
1. **Aggregation by sector:**
   - For each sector, you calculated:
     - **Unique projects** = how many distinct projects belong to that sector.
     - **Average baseline PD** = mean of default_prob (before stress).
     - **Average stressed PD** = mean of pd_stressed (after applying GDP and CPI shocks).
2. **Calculated stress impact:**
   - Computed **change in PD** = difference between stressed and baseline averages.
   - This measures how much default probability rises under the stress scenario.
3. **Sorted sectors by stressed PD:**
   - Sectors with the **highest average stressed PDs** appear at the top of the results table.
   - This highlights which sectors are most vulnerable under adverse macroeconomic conditions.

	sector	unique_projects	avg_default_prob	avg_pd_stressed	change_in_pd
3	Energy & Power	15	0.446635	0.462582	0.015947
11	Urban Development & Housing	15	0.359687	0.376561	0.016874
9	Technology & Innovation	7	0.355352	0.371299	0.015947
10	Transport & Infrastructure	15	0.339437	0.354776	0.015339
5	Finance & Industry	12	0.308915	0.327083	0.018167
12	Water & Irrigation	19	0.281017	0.298277	0.017260
7	Health & Social Protection	16	0.267315	0.284874	0.017559
1	Disaster Recovery & Emergency	1	0.197725	0.214302	0.016577
2	Education & Skills	6	0.193852	0.211036	0.017184
0	Agriculture & Rural Development	12	0.182870	0.200818	0.017948
6	Governance & Policy Reform	11	0.154118	0.172136	0.018017
4	Environment & Climate	9	0.090274	0.108517	0.018243
8	Others	1	0.028390	0.047305	0.018915

Results interpretation

- **Highest risk sectors after stress:**
  - *Energy & Power* emerges at the top with a stressed PD of **46.3%**, up from 44.7%.
  - *Urban Development & Housing* and *Technology & Innovation* also show relatively high stressed PDs (~37%).
  - These findings indicate that large infrastructure-heavy and technology-driven projects are more exposed to systemic shocks.
- **Mid-tier risk sectors:**
  - *Transport & Infrastructure* (35.5%), *Finance & Industry* (32.7%), *Water & Irrigation* (29.8%), and *Health & Social Protection* (28.5%) show substantial stressed PDs, highlighting potential fiscal strain in critical development sectors.
- **Lower risk sectors:**
  - *Education & Skills* (21.1%) and *Agriculture & Rural Development* (20.1%) see moderate increases.
  - *Environment & Climate* and *Governance & Policy Reform* remain relatively resilient with stressed PDs under 18%.
  - *Others* category remains very low at **4.7%**, though this is based on only one project.
- **Stress impact size:**
  - Across most sectors, the **increase in PD is modest** (roughly 1.5–1.9 percentage points).
  - This reflects the weak sensitivity found in the regression coefficients, meaning macro shocks shift risk upward slightly, but sectoral baseline risk levels are the main driver.

Interpretation of Sectoral Average PD: Baseline vs. Stressed



Stressed PD is slightly higher than baseline PD across all sectors, showing mild vulnerability under stress scenarios.

1. **Energy & Power** has the highest average PD (>0.45), followed by **Urban Development & Housing**, **Technology & Innovation**, and **Transport & Infrastructure** — these sectors represent the largest credit risk concentration.
2. **Finance & Industry**, **Water & Irrigation**, and **Health & Social Protection** show moderate PD levels with moderate stress sensitivity.
3. **Environment & Climate** and **Others** have the lowest PD values (<0.1), posing minimal portfolio risk.

ECL under Stress Testing:

We then moved from **sector-level averages** to a **project-level stress test of Expected Credit Loss (ECL)**.

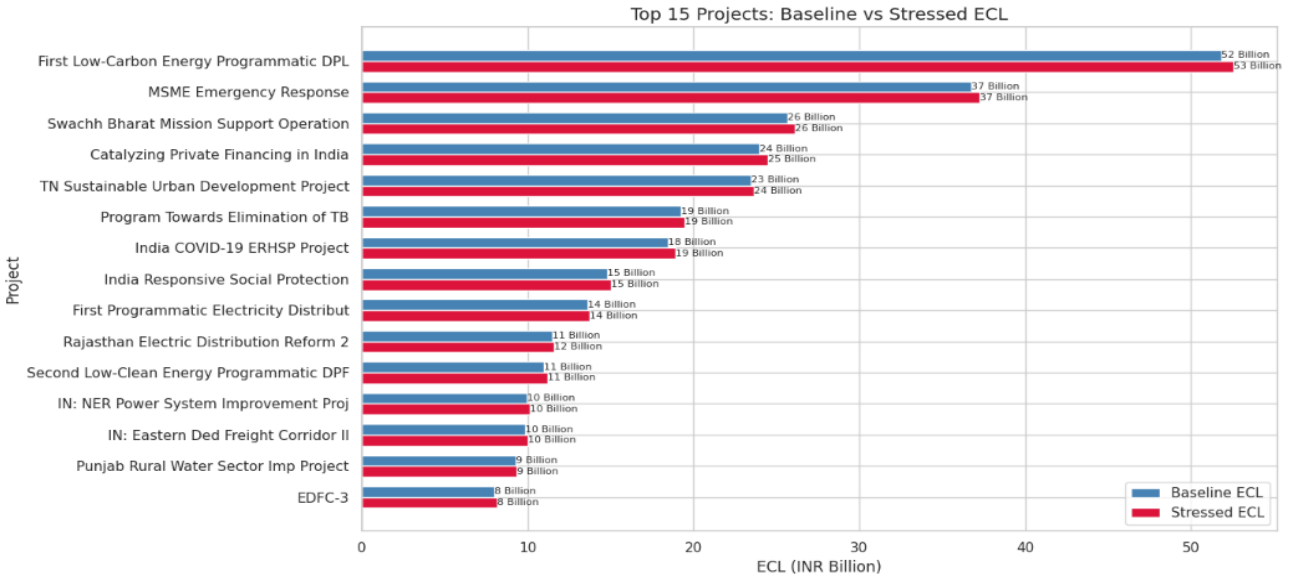
1. **Created human-readable financial outputs:**

- A helper function ("human_readable") was defined to convert raw INR values into **Billion, Million, Thousand** for easier interpretation in tables and plots.
- This makes the results more accessible for policymakers and non-technical readers.

2. **Formatted a project-level comparison table:**

- Built "ecl_compare_display", showing for each project:
  - **EAD (Exposure at Default)** in INR.
  - **LGD (Loss Given Default).**
  - **Baseline PD** ("pd_base") vs. **Stressed PD** ("pd_stressed").
  - **Baseline ECL** vs. **Stressed ECL**.
  - **Change in ECL (absolute and %)** after applying the stress scenario.

Interpretation of Top 15 Projects: Baseline vs. Stressed ECL



Expected Credit Loss (ECL) is slightly higher under stressed conditions for most projects, but the difference is marginal, indicating resilience.

- **First Low-Carbon Energy Programmatic DPL** shows the largest ECL (~₹52–53B), dominating portfolio exposure.
- **MSME Emergency Response** and **Swachh Bharat Mission Support Operation** follow with ~₹37B and ~₹26B ECL respectively
- **Catalyzing Private Financing in India** and **TN Sustainable Urban Development Project** range around ₹23–24B.
- Projects like **Program Towards Elimination of TB**, **India COVID-19 ERHSP Project**, and **India Responsive Social Protection** are in the ₹15–19B range.
- The bottom projects in this set (**IN: NER Power System Improvement, IN: Eastern DFC II, Punjab Rural Water Sector, EDFC-3**) have ECL of ~₹8–10B.

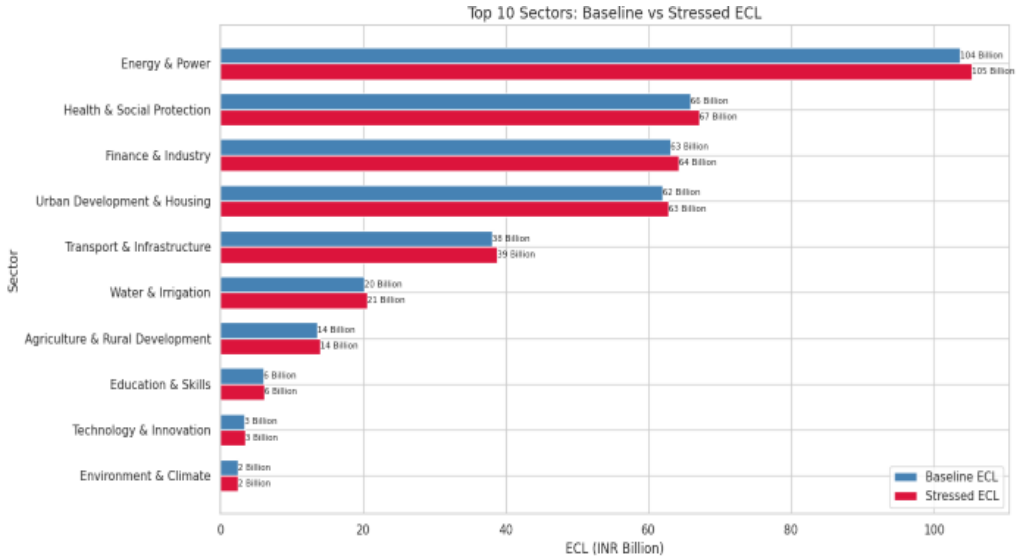
Sectoral ECL - Before vs. After Stress Testing:

This step moves from **project-level stress testing** to a **sectoral ECL stress test**, giving a portfolio-level view.

1. **Mapped projects to sectors:**
    - Ensured every project in the ECL dataset (ecl_compare_df) was linked to its corresponding sector, either from merged_df or a predefined mapping.
2. **Merged ECL and sector data:**
    - Combined Expected Credit Loss (ECL) results with sector information, creating a unified dataset where each project carried both financial and sectoral attributes.
3. **Aggregated to sector level:**
    - Computed total **Exposure at Default (EAD)**, **Baseline ECL**, and **Stressed ECL** for each sector.
    - This shows the absolute risk burden at the sector level rather than just per project.
4. **Calculated stress impacts:**
    - Measured **absolute changes (INR)** and **percentage changes (%)** in ECL between baseline and stressed conditions.
    - This highlights which sectors see the biggest shifts when macroeconomic shocks are applied.
5. **Formatted results for readability:**
    - Converted raw rupee values into **Billions/Millions/Thousands** for easier interpretation in reports and tables.

Visualization (Top 10 sectors):

- Created a **horizontal grouped bar chart** comparing Baseline vs. Stressed ECL across the ten most exposed sectors.
- Labels were added in human-readable format to clearly show the size of credit losses.



Stressed ECL is slightly higher than baseline across all sectors, but the gap is small, indicating good portfolio resilience.

- **Energy & Power** has the largest exposure (~₹104–105B), making it the most critical sector for risk management focus.
- **Health & Social Protection (~₹76–77B)**, **Finance & Industry (~₹63–64B)**, and **Urban Development & Housing (~₹62–63B)** together form the bulk of portfolio exposure after Energy & Power.
- **Transport & Infrastructure (~₹35–36B)** and **Water & Irrigation (~₹20B)** have moderate ECL contribution.
- **Agriculture & Rural Development, Education & Skills, Technology & Innovation, and Environment & Climate** show relatively smaller ECL impact (<₹15B each).

Conclusion:

This study developed a credit risk assessment framework for World Bank–funded development projects in India by integrating loan-level financial data with macroeconomic indicators and applying machine learning–based probability of default (PD) modeling. The analysis demonstrated how project-level risks can be systematically translated into sectoral vulnerabilities and quantified through Expected Credit Loss (ECL) measures under both baseline and stressed macroeconomic scenarios.

The results reveal that sectors such as **Energy & Power, Urban Development & Housing, Technology & Innovation, and Transport & Infrastructure** exhibit the highest average default probabilities and stressed ECL burdens, underscoring their systemic exposure to adverse shocks. In contrast, sectors such as **Governance & Policy Reform** and **Environment & Climate** appear more resilient. Importantly, while macroeconomic stress (falling GDP growth and rising inflation) led to only modest increases in average PDs, the sectoral and project-level aggregation highlighted significant concentration of risks, particularly in large infrastructure and energy portfolios.

From a policy perspective, the findings suggest the need for **enhanced monitoring of high-risk sectors, stronger risk mitigation mechanisms, and contingency planning for macroeconomic downturns**. Moreover, the framework provides a replicable methodology for multilateral lenders and policymakers to link micro-level project data with macro-level stress testing, thereby improving forward-looking credit risk management and aligning with international best practices under **IFRS 9**.

In summary, this project demonstrates the feasibility and value of combining **statistical modeling, macroeconomic stress testing, and sectoral analysis** to better understand and manage the default risk of sovereign-backed development finance in India.

References:

1. World Bank. (2024). IBRD Statement of Loans and Guarantees Historical Data. Retrieved from https://financesone.worldbank.org/
2. Basel Committee on Banking Supervision. (2017). Basel III: Finalising post-crisis reforms. Bank for International Settlements.
3. Lundberg, S. M., & Lee, S. I. (2017). A unified approach to interpreting model predictions. Advances in Neural Information Processing Systems.
4. Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. Proceedings of KDD.
5. Prokhorenkova, L., et al. (2018). CatBoost: unbiased boosting with categorical features. Advances in Neural Information Processing Systems.