Credit Card Customer Default Prediction – Detailed Project Report

Phase 0: Understanding the Dataset

The first step in building a robust predictive model is to understand the structure and context of the data. In this dataset, we are tasked with predicting the likelihood of a credit card customer defaulting (`bad_flag = 1`) or not (`bad_flag = 0`). The dataset is quite large, with 96,806 rows and 1,216 columns, which makes it high-dimensional and complex.

To simplify this, the dataset can be grouped into four major types of features:

- Onus attributes: Represent internal usage data specific to the bank.
- Transaction attributes: Capture how customers spend money — useful for behavioral patterns.
- Bureau tradeline attributes: Provide historical credit information from credit bureaus — often highly predictive of defaults.
- Bureau enquiry attributes: Indicate recent inquiries into a customer's credit — this can signal financial stress or upcoming liabilities.

The goal here was to identify which features are numerical, categorical, ID-like (e.g., `account_number`), or time-based to plan cleaning and modeling strategies accordingly.

Phase 1: Data Cleaning

With over a thousand features, data cleaning was both critical and challenging. Here's a breakdown of what was done and why:

1. Removing Duplicate Columns and Useless Identifiers

- Duplicate columns (having the same values across all rows) were removed to avoid redundancy.

- Columns like `account_number` were dropped because they serve as identifiers and hold no predictive value — keeping them may mislead the model.

2. Handling Missing Values

- The percentage of missing values was calculated for each column.
- Any column with more than 50% missing data was dropped, as imputing them would inject noise and bias.
- For the remaining missing values:
    - KNN Imputer was used, which estimates missing values by looking at similar rows.
    - This was chosen over simple mean/median imputation because the dataset is entirely numerical, and KNN can capture local data structures.
    - To manage performance with such a large dataset, imputation was done in chunks, which is a smart memory-efficient choice.
    - The filled dataset was saved to a CSV file to avoid rerunning the expensive operation.

3. Outlier Detection

- Outliers were visually analyzed using boxplots (e.g., for the target variable).
- Although the notebook doesn't show explicit outlier treatment, PCA and scaling later help mitigate their effect.

Phase 2: Exploratory Data Analysis (EDA)

After cleaning the data, exploratory data analysis was conducted to understand variable behavior and the relationship with the target variable.

1. Target Variable Distribution

- The `bad_flag` variable was found to be imbalanced, with more “non-default” cases than “default” ones.

- This is a typical scenario in fraud or credit risk problems, and it informs decisions about model choice and evaluation metrics.

2. Boxplots and Distribution Analysis

- Boxplots helped reveal the spread of numeric variables and outliers.
- Since no categorical variables were present, the analysis focused on continuous data distributions.

Phase 3: Feature Engineering

With such a high number of columns, feature engineering was necessary to improve model performance and training efficiency.

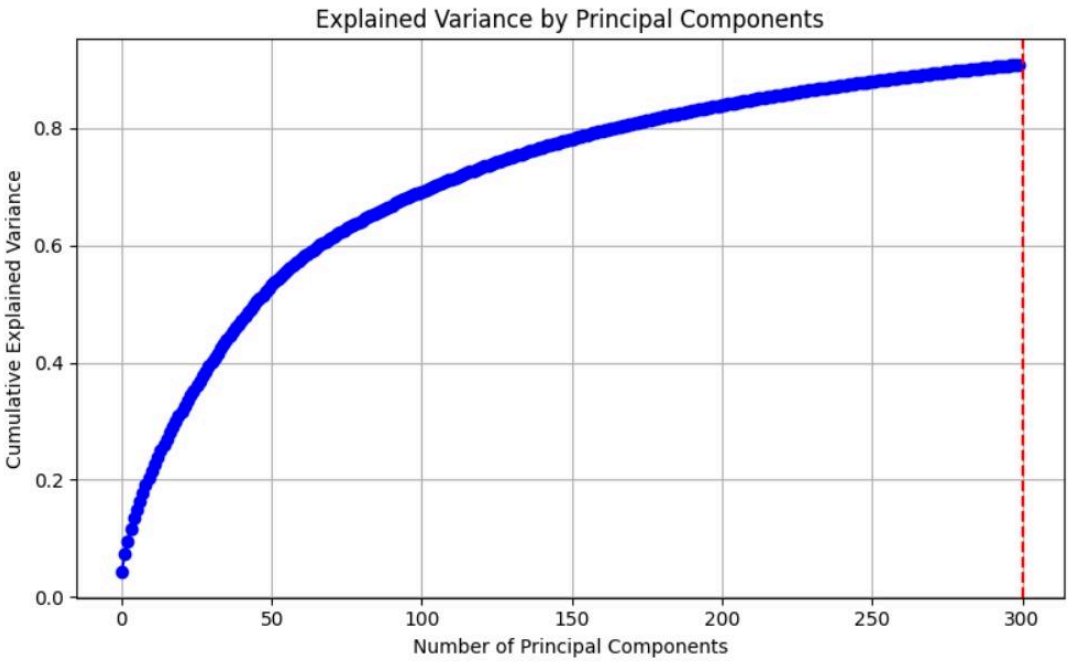
1. Feature Scaling

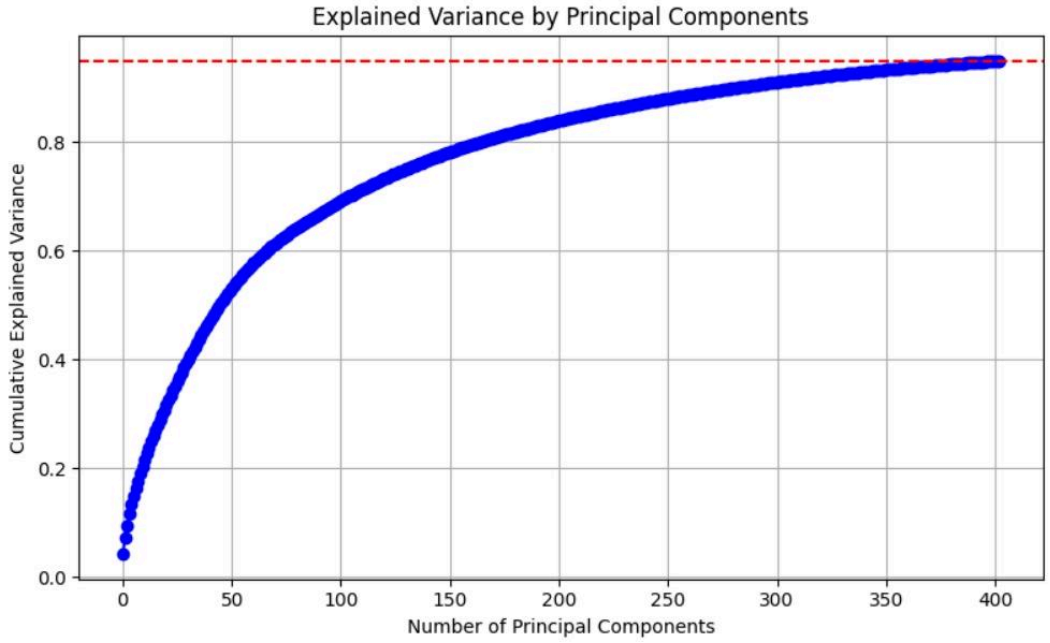
- Applied StandardScaler to normalize numerical features (zero mean and unit variance).
- Scaling is essential before PCA and for models like logistic regression which are sensitive to feature magnitude.

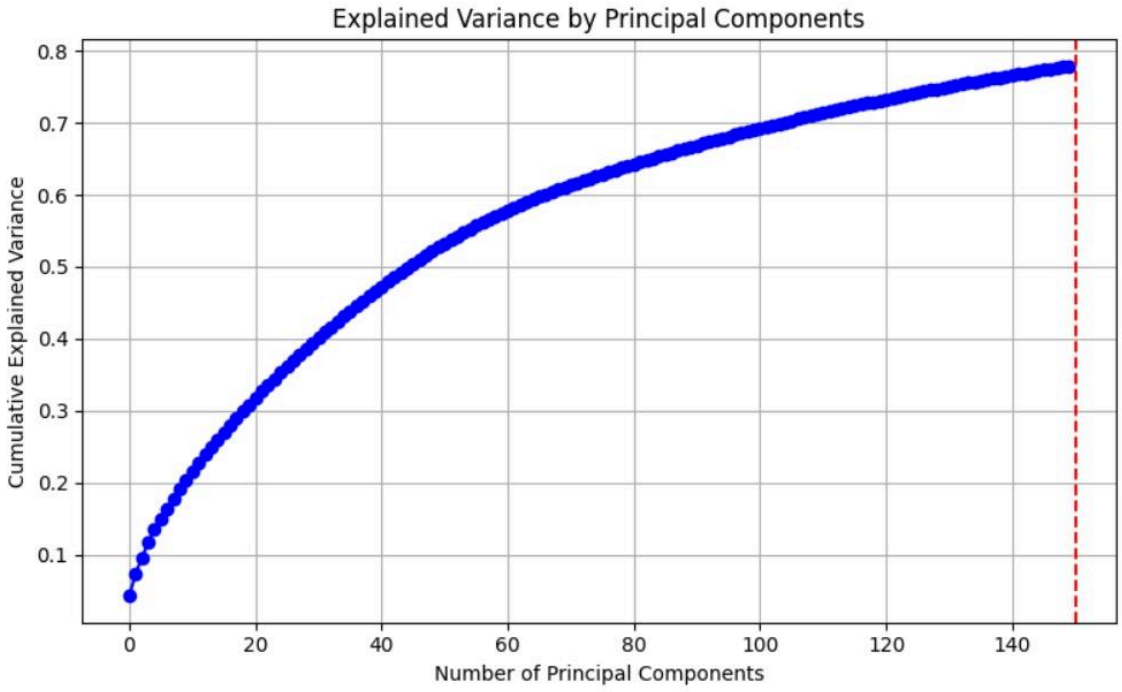
2. Dimensionality Reduction using PCA

- PCA (Principal Component Analysis) was used to reduce the number of features while retaining most of the variance.
- This helped eliminate noise and multicollinearity and made the data more manageable.
- Various PCA configurations were tested to find the best number of components that preserve important signals.
- Why PCA?
    - With 1,216 features, training time, memory usage, and multicollinearity were significant challenges.
    - PCA compressed the data while maintaining predictive power.
    - We run different test on PCA like at how much dimension what variance we are getting through graph

- We finally conclude to 270 with which we get above 90 percent variance




Explained Variance by Principal Components
Cumulative Explained Variance
Number of Principal Components
0.0
0.2
0.4
0.6
0.8
0
50
100
150
200
250
300
350
400


Explained Variance by Principal Components
Cumulative Explained Variance
Number of Principal Components
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
20
40
60
80
100
120
140

Phase 4: Class Imbalance Handling

Class imbalance (more 0s than 1s in `bad_flag`) was addressed during modeling.

1. Weight Balancing

- In the Random Forest Classifier, the `class_weight='balanced'` parameter was used.
- This makes the model give more importance to the minority class (defaults).
- SMOTE or oversampling was not used — possibly to avoid adding synthetic data or due to computational cost.

2. Why Not Oversample?

- The dataset was already large.
- Imputation and PCA were expensive — so adding synthetic rows might have led to scalability issues.

Phase 5: Model Building & Evaluation

After preparing and cleaning the data, the next step was to build predictive models to classify whether a customer is likely to default (`bad_flag = 1`) or not (`bad_flag = 0`). Three models were trained, each evaluated with appropriate metrics for imbalanced classification tasks.

1. Logistic Regression – The Baseline Model

- Why chosen?
  Logistic Regression is a simple, interpretable, and fast algorithm — ideal as a baseline for binary classification.
- Preprocessing done before training:
  - Data was scaled using `StandardScaler`.
  - Dimensionality reduced using PCA before feeding into the model.

- Observations:
    - Despite being a linear model, it performed surprisingly well on the ROC-AUC metric.
    - This suggests that a good linear decision boundary already exists in the reduced PCA space.
- **Results:**
    - **ROC-AUC Score: Around 0.83**
    - **F1 Score: Lower than tree-based models but still respectable, showing that Logistic Regression can generalize well even with reduced dimensions.**

2. XGBoost Classifier – The Powerful Gradient Boosted Model

- Why chosen?
  XGBoost is known for its ability to capture non-linear relationships, work well with tabular data, and handle missing values and class imbalance effectively.
- Hyperparameter Tuning:
    - Tuning was done using `RandomizedSearchCV` with key parameters like:
        - `n_estimators`: Number of trees
        - `max_depth`: Controls tree complexity
        - `learning_rate`: Shrinks contribution of each tree
        - `subsample`, `colsample_bytree`: Controls overfitting
        - `scale_pos_weight`: Handled class imbalance
    - This tuning helped the model become more balanced between recall and precision.
- R**esults After Tuning:**

- **ROC-AUC Score: ~0.84–0.85**
- **F1 Score: Improved after tuning**
- **Confusion Matrix: Showed better recall (i.e., correctly identifying defaulters) without losing too much precision**

- Interpretation:
  XGBoost provided a great trade-off between predictive performance and robustness — and worked well even with high-dimensional, imputed data.

3. Random Forest Classifier – Ensemble of Decision Trees

- Why chosen?
  Random Forest is a robust ensemble method that performs well on structured data. It can model feature interactions and is less prone to overfitting with enough trees.

- Handling Class Imbalance:
  - The model was initialized with `class_weight='balanced'` to compensate for the underrepresentation of defaulters.

- Hyperparameter Tuning:
  - Used `RandomizedSearchCV` to tune:
    - `n_estimators`, `max_depth`, `min_samples_split`, `max_features`
  - Best parameters were used for final model training.

- **Results:**
  - **ROC-AUC Score: Around 0.83–0.84**
  - **F1 Score: Highest among all three models**
    - **This indicates Random Forest had the best balance between false positives and false negatives.**

- Confusion Matrix Observations:

- More defaulters correctly identified (`True Positives`) after class weight balancing.
- Some increase in false positives, but acceptable given the context.

Evaluation Metrics Used

- **ROC-AUC Score:**
  Measures how well the model distinguishes between the two classes.
  - Useful when dealing with imbalanced data.
- **F1 Score:**
  Harmonic mean of precision and recall.
  - Particularly important when the cost of misclassifying a defaulter is high.

Confusion Matrix:
Showed class-wise prediction breakdown — helped assess which model had better recall vs. precision.

Model Performance Summary

Model	ROC-AUC	F1 Score	Best For
Logistic Regression	~0.83	Moderate	Fast, simple baseline
XGBoost	~0.85	High	Accuracy, ROC-AUC
Random Forest	~0.84	**Highest**	Best F1 (balanced)