CONVOLVE 3.0
HACKATHON

DOCUMENTATION

Prepared by

ARYAN KUMAR

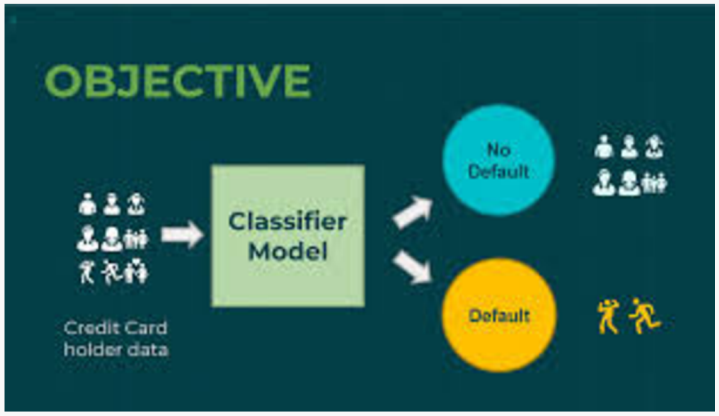
&

SMIT BHOIR

Team name: am22s042
(madras hulks)

Overview:

Due to consumers' growing preference for using credit cards as a form of payment, their use has expanded throughout time. Credit card default is caused by spending too much money, losing one's job, or experiencing financial difficulties. In addition to significant financial losses, decreased profitability, and bankruptcy, it also damages one's reputation and trust. Therefore, it's essential to accurately predict the chance of failure in order to reduce risk and costs. For financial institutions around the world, credit card default is a serious problem. Therefore, the capacity to accurately predict credit card default is crucial for the sustainability and financial stability of these organizations . To lower the risks associated with credit card default, financial organizations have traditionally forecasted default rates using statistical models. Although these models have some success, they also have a lot of problems.



Machine learning (ML) techniques have helped in overcoming these constraints. These helps in identifying variations and trends that can be utilized to forecast future defaults by analyzing historical data. These algorithms provide a number of benefits, such as the capacity to handle big datasets, nonlinear relationships, and complex interactions between variables. Therefore, ML algorithms have emerged as a powerful tool for predicting credit card default

Problem Statement:

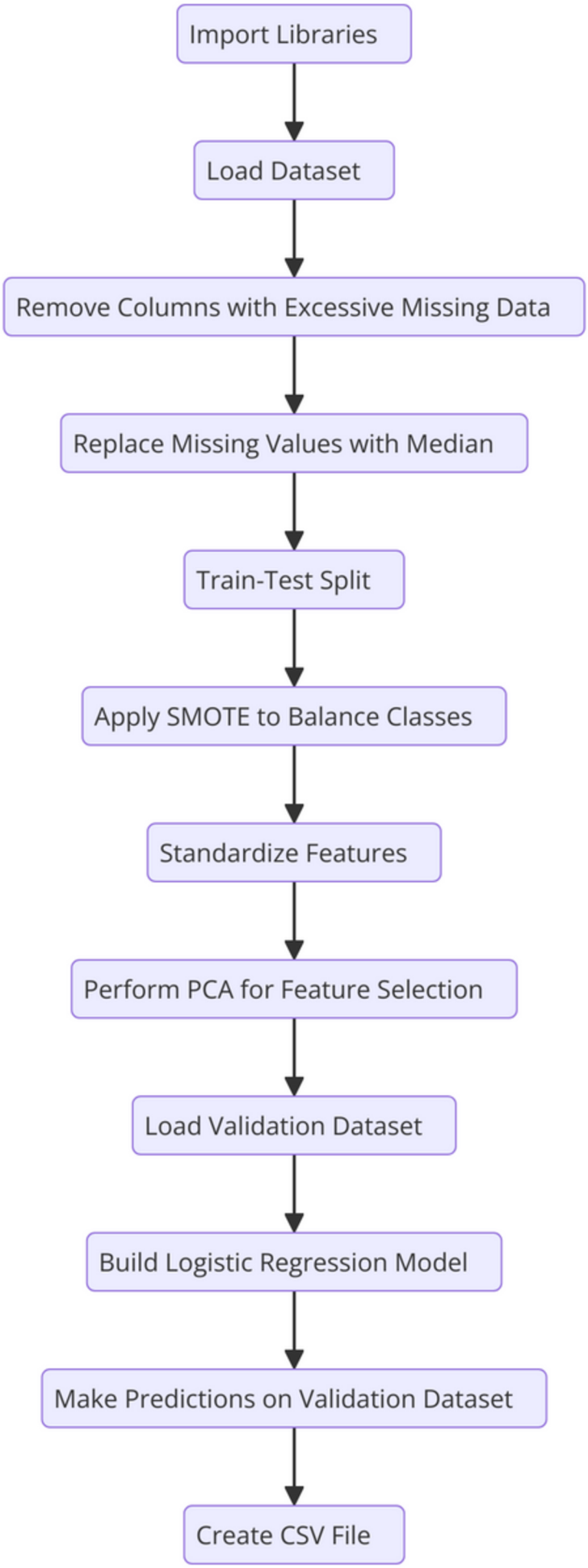
Bank A issues Credit Cards to eligible customers and uses advanced machine learning models to manage customer eligibility, limit, and interest rate assignments. These models are optimized for risk management and profitability. To further improve its risk management framework, the bank has decided to create a “Behaviour Score” for existing Credit Card customers. This score will predict the likelihood of customers defaulting on their Credit Cards, helping with portfolio risk management activities.

The Goal:

The goal is to develop a predictive "Behaviour Score" for Bank A. The model should predict the probability of a Credit Card customer defaulting based on historical data. This model will be applied to customers whose Credit Cards are open and not past due.

Complete code link

ML Architecture



Complete code link

Approach :

After downloading the dataset, we started the analysis process by checking the rows and columns, basically to check the number of features and their relationship manually before loading it into the system.

Manually looking at the dataset will give you nightmares as the data has no predefined variables or attributes, you can not just decide as to which variable to choose and which to remove. So now we decided to load the data in our python environment and continue the data analysis processes.

1. Importing All Libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score, log_loss, roc_curve, auc
from sklearn.metrics import roc_auc_score, roc_curve, classification_report, precision_score, recall_score, f1_score
```

Library Imports

Our project utilizes several key Python libraries:

- pandas and numpy for data manipulation
- scikit-learn for machine learning models and preprocessing
- imblearn for handling imbalanced datasets
- matplotlib for visualization
- xgboost for gradient boosting (though not used in final implementation)

2. Loading the Dataset

```
[3] # Provide the path to the dataset on your Google Drive
file_path = '/content/drive/MyDrive/Colab Notebooks/Dev_data_to_be_shared.csv'

data = pd.read_csv(file_path)
```

```
data.head()
```

	account_number	bad_flag	onus_attribute_1	transaction_attribute_1	transaction_attribute_2	transaction_attribute_3	transaction_attribute_4	transa
0	1	0	NaN	NaN	NaN	NaN	NaN	
1	2	0	221000.0	0.0	0.0	0.0	0.0	
2	3	0	25000.0	0.0	0.0	0.0	0.0	
3	4	0	86000.0	0.0	0.0	0.0	0.0	
4	5	0	215000.0	0.0	0.0	0.0	0.0	

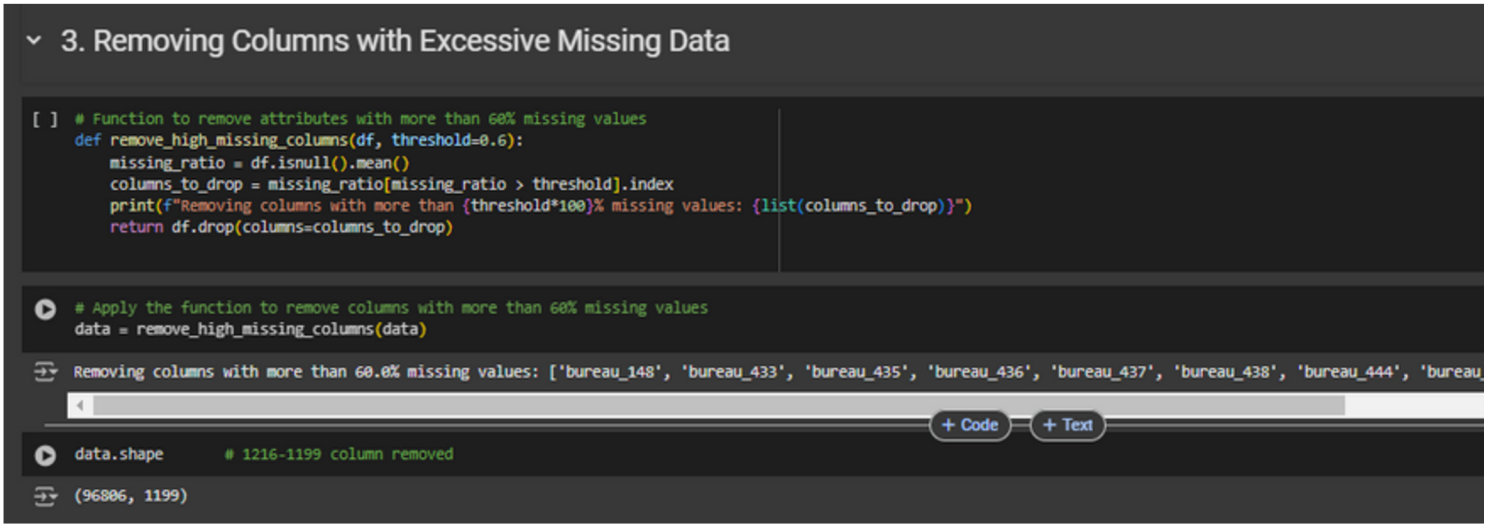
5 rows × 1216 columns

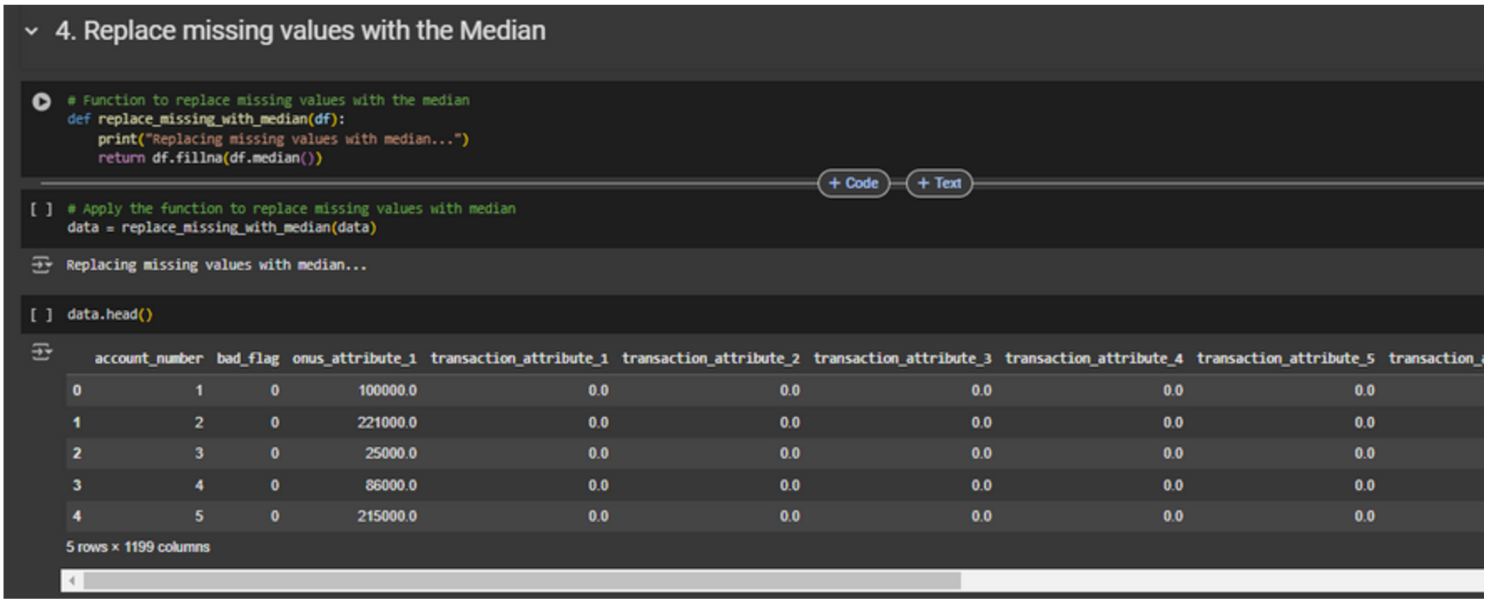
Data Loading and Initial Analysis

The dataset contains account information with multiple features. Initial analysis revealed:

- Target variable: 'bad_flag' (binary classification problem)
- Significant class imbalance in the target variable
- All features are numerical (int or float)
- No duplicate account numbers
- Multiple columns with missing values

Complete code link





Data Cleaning Strategy

Handling Missing Values

- Removed columns with more than 60% missing values
- 17 columns were dropped (reduced from 1216 to 1199 features)
- Remaining missing values were filled with median values
- This approach balances data completeness with information preservation.

At first, there are 1216 characteristics in the dataset, which is a significant number. Effectively managing missing values is essential to guaranteeing the caliber and dependability of the data used to train the model.

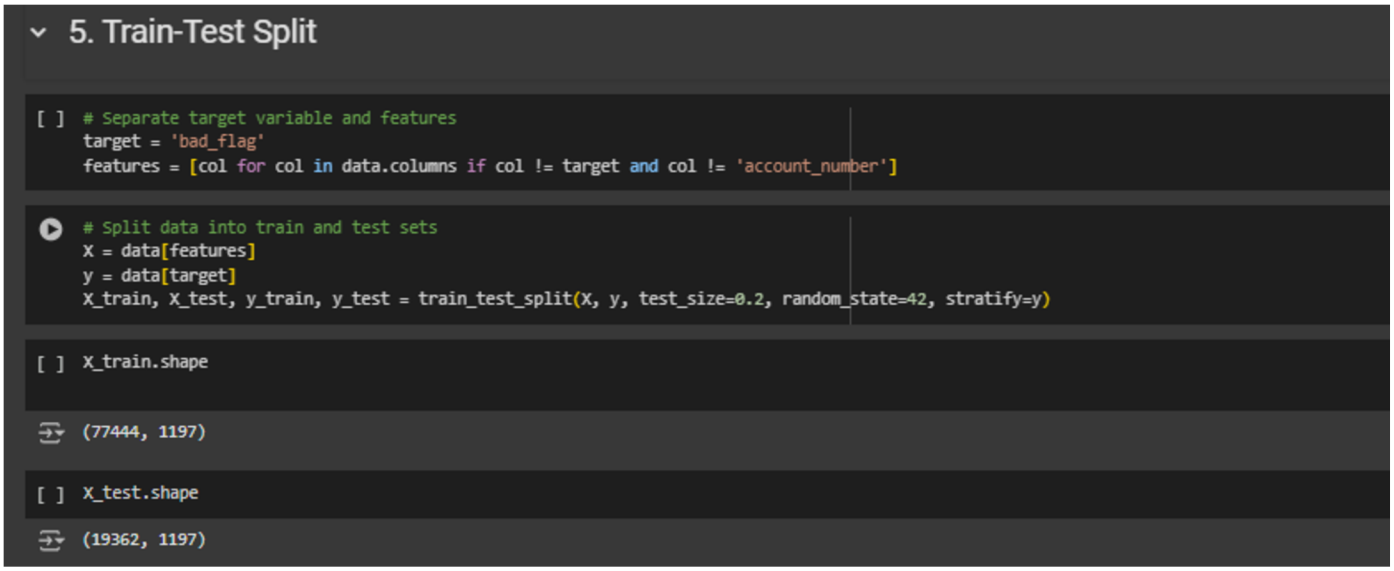
Managing Missing Values :

Machine learning models' performance can be greatly impacted by missing values. There are two primary steps in the missing values strategy

High percentages of missing values in a column can bring noise into the model and are likely to yield little relevant information. By eliminating these columns, we lower the dataset's dimensionality and concentrate on characteristics with more comprehensive data.

The median is a reliable method for filling in the missing values for the remaining columns. The median is a good option for imputation since it is less susceptible to outliers than the mean.

Complete code link



Data Preprocessing

Important considerations:

- 80-20 split for training and testing
- Stratified split to maintain class distribution
- Random seed set for reproducibility

Machine learning frequently uses an 80-20 split, in which 20% of the data is set aside for testing and 80% is used for model training. This guarantees that there will be sufficient data to train the model and retain a sizeable amount for assessing its effectiveness.

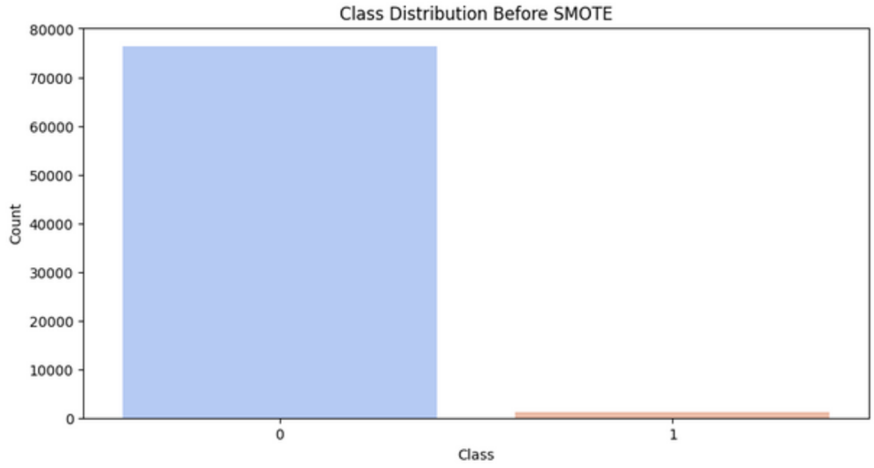
Handling Class Imbalance

- SMOTE (Synthetic Minority Over-sampling Technique) was used to address class imbalance
- This creates synthetic samples of the minority class
- Only applied to training data to prevent data leakage

When there are much more instances in one class than in another, this is referred to as class imbalance. Biased models that perform well on the majority class but badly on the minority class may result from this.

Complete code link

By producing synthetic samples of the minority class, SMOTE is a widely used method for resolving class imbalance. This enhances the model's capacity to learn from the minority class and helps to balance the distribution of classes. By interpolating across minority class samples that already exist, SMOTE creates synthetic samples. This improves the model's learning by increasing the number of minority class examples without merely replicating them.SMOTE should only be used on training data to avoid data leaks. Overly optimistic performance predictions result from data leakage, which happens when information from outside the training dataset is used to build the model.



7. Standardization

```
[ ] # Standardize the selected features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train_smote)
    X_test_scaled = scaler.transform(X_test)
```

```
[ ] X_train_scaled.shape
```

```
(152692, 1197)
```

```
X_test_scaled.shape
```

```
(19362, 1197)
```

Complete code link

Feature Scaling and Dimensionality Reduction

- StandardScaler used to normalize features
- PCA reduced dimensions from 1199 to 50 components
- This significantly reduced computational complexity while maintaining important information

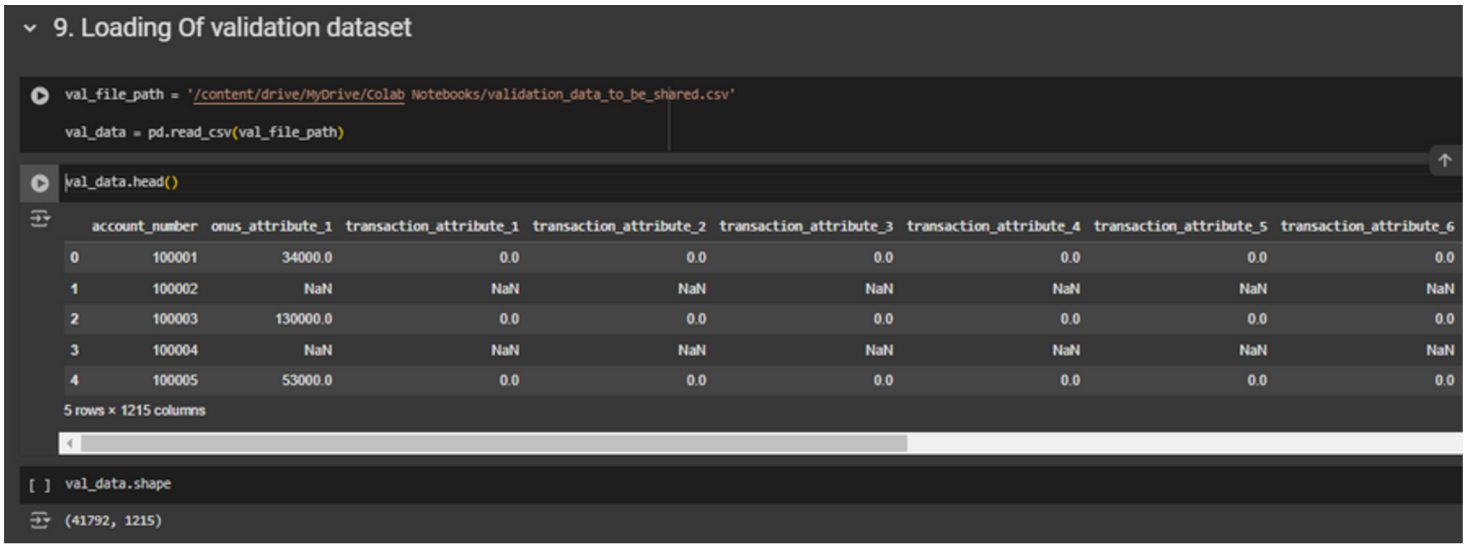
In machine learning, feature scaling is a crucial preprocessing step that guarantees each feature contributes equally to the model. It makes sure that features with greater sizes don't control the learning process and helps gradient-based algorithms converge faster. By scaling to unit variance and eliminating the mean, StandardScaler standardizes characteristics. This guarantees that the standard deviation is one and the mean is zero for every feature.
Reducing the amount of features while keeping the majority of the crucial information is made possible by dimensionality reduction. This eliminates redundant features and noise, which lowers computational complexity and can enhance model performance.

What is PCA? Why did we use it here?

There are a lot of characteristics in the dataset at first (1199 after initial cleanup). Increased processing complexity, the possibility of overfitting, and challenges with data visualization and interpretation are just a few of the problems that can arise from high-dimensional data.
Computational Complexity: computing and training models on high-dimensional data demand greater computing power. Longer training sessions and higher memory consumption may result from this.

Overfitting: Models are more likely to capture noise and unimportant patterns rather than broadly applicable trends when they have a high feature count.
The **Curse of Dimensionality** states that the volume of the feature space increases exponentially with the number of dimensions, making it more difficult to identify the best solutions and producing sparse data.By converting the original features into a new collection of orthogonal components arranged according to the amount of variation they explain, PCA is a potent dimensionality reduction technique that tackles these issues.

Complete code link



9. Loading Of validation dataset

```
val_file_path = '/content/drive/MyDrive/Colab Notebooks/validation_data_to_be_shared.csv'

val_data = pd.read_csv(val_file_path)
```

```
val_data.head()
```

	account_number	onus_attribute_1	transaction_attribute_1	transaction_attribute_2	transaction_attribute_3	transaction_attribute_4	transaction_attribute_5	transaction_attribute_6
0	100001	34000.0	0.0	0.0	0.0	0.0	0.0	0.0
1	100002	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	100003	130000.0	0.0	0.0	0.0	0.0	0.0	0.0
3	100004	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	100005	53000.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 1215 columns

```
val_data.shape
```

```
(41792, 1215)
```

```
# Droping the same columns in the validation dataset which have missing data more tha 60%
val_data.drop(columns=['bureau_148', 'bureau_433', 'bureau_435', 'bureau_436', 'bureau_437', 'bureau_438', 'bureau_444', 'bureau_446', 'bureau_447', 'burea
```

```
val_data.shape
```

```
(41792, 1198)
```

```
val_data.fillna(data.median(), inplace=True)  # Replace missing values with training data median
```

```
# Preprocess validation dataset
val_data_scaled = scaler.transform(val_data[features])  # Standardize
```

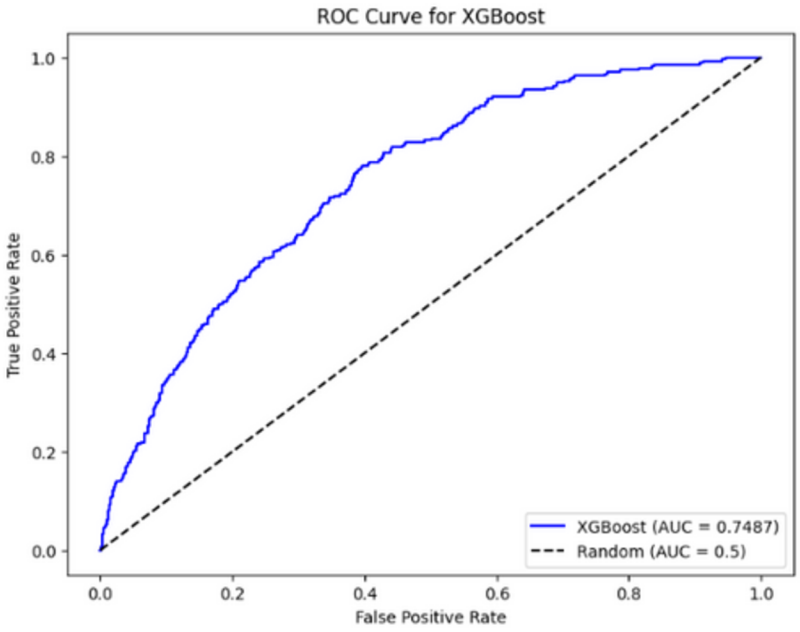
```
val_data_pca = pca.transform(val_data_scaled)  # Apply PCA
```

Loading the validation dataset for final analysis and creation of the csv file.

Complete code link

```python
import xgboost as xgb
from sklearn.metrics import classification_report, precision_score, recall_score, f1_score, roc_curve, roc_auc_score

xgb_model = xgb.XGBClassifier(
    random_state=42,
    scale_pos_weight=len(y_train_smote) / sum(y_train_smote == 0),  # Account for class imbalance
    learning_rate=0.05,
    n_estimators=100,
    max_depth=6  )

xgb_model.fit(X_train_pca, y_train_smote)

# Predict and evaluate
xgb_y_pred = xgb_model.predict(X_test_pca)
xgb_y_pred_proba = xgb_model.predict_proba(X_test_pca)[:, 1]
```

```
XGBoost Classification Report:
                precision    recall  f1-score   support

           0       0.99      0.68      0.81     19088
           1       0.03      0.67      0.06       274

    accuracy                           0.68     19362
   macro avg       0.51      0.67      0.43     19362
weighted avg       0.98      0.68      0.80     19362

Precision: 0.0292, Recall: 0.6679, F1 Score: 0.0560
XGBoost ROC AUC: 0.7487
```

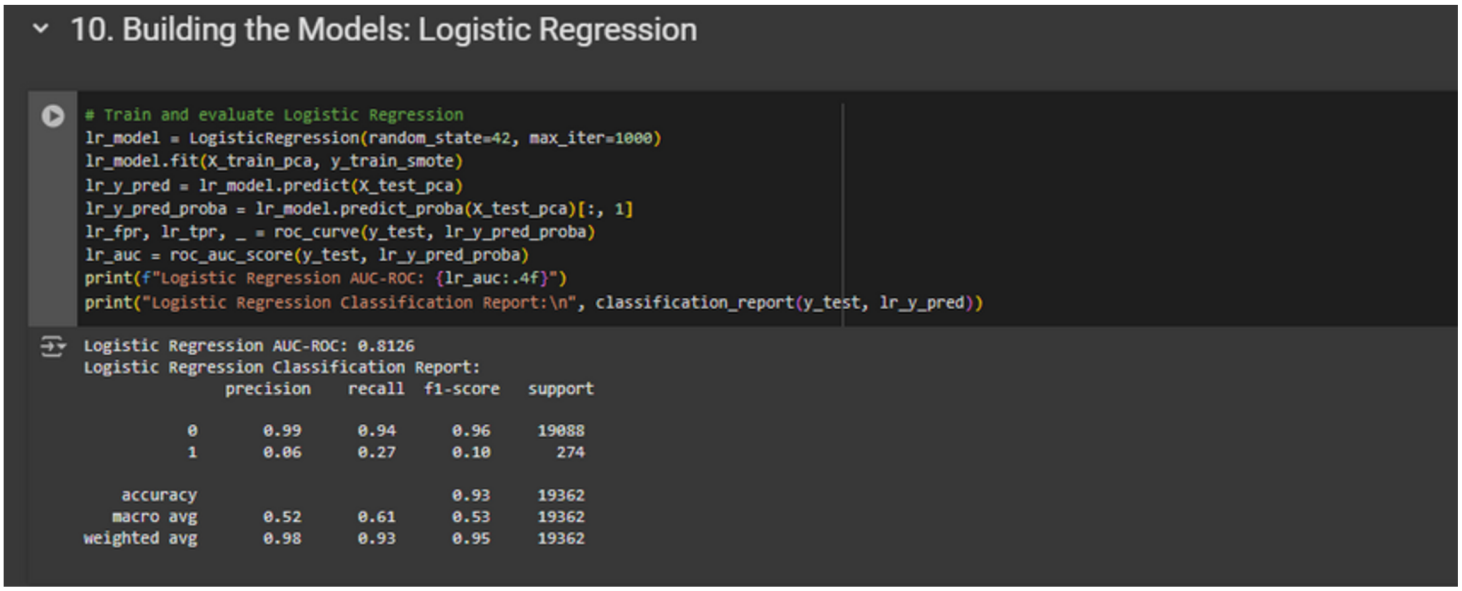


We tried to implement and train the model on XgBoost algorithm. Even, though implementing and tuning the hyperparameters manually we could not increase the accuracy .

Complete code link

```
Random Forest Model Evaluation:
AUC-ROC: 0.6838519721271659
Precision: 0.0
Recall: 0.0
F1-Score: 0.0
C:\Users\Aryan\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\Local
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
Fitting 3 folds for each of 108 candidates, totalling 324 fits
```

Before finalizing our machine learning model we implemented multiple ml model such as Random Forest , XgBoost algorithm, SVC , Naive Bayes, Grdaient Boost etc.



Although the model's overall accuracy (0.93) is high, the class imbalance makes this misleading.
Let's dissect it:
Regarding Class 0, which is probably your majority class:
Outstanding results with 0.94 recall and 0.99 precision
Thus, it hardly ever incorrectly categorizes negative instances.
Class 1 (the minority class):
Poor performance, with recall and precision of just 0.27 and 0.06, respectively
This implies that the model has trouble accurately identifying affirmative cases.
94% of the time, it is incorrect when predicting a positive case.
A more balanced perspective is offered by the AUC-ROC of 0.8126, which shows considerable discriminative capacity independent of threshold selection.

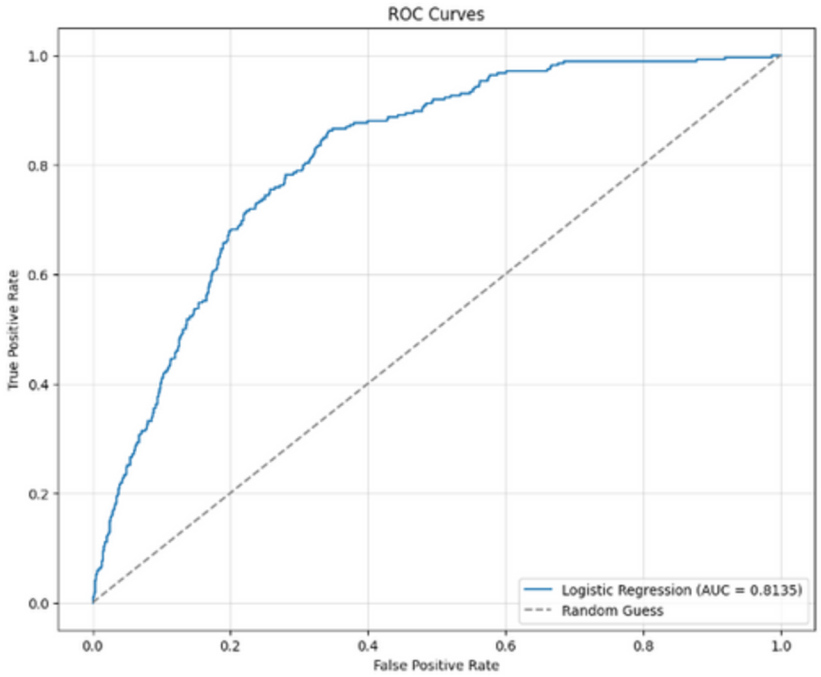
Model Implementation

Logistic Regression chosen for its:

- Interpretability
- Probability outputs
- Computational efficiency
- Model parameters:
- Increased max_iter to ensure convergence
- Default regularization parameters

Complete code link

```python
# Plot ROC Curves
plt.figure(figsize=(10, 8))
plt.plot(lr_fpr, lr_tpr, label=f'Logistic Regression (AUC = {lr_auc:.4f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves')
plt.legend(loc='lower right')
plt.grid(alpha=0.4)
plt.show()
```



Model Evaluation

- Used AUC-ROC as primary metric due to imbalanced nature of problem
- Generated classification report for detailed performance metrics
- Visualized ROC curve for model performance analysis

Overall Performance: The model performs reasonably well, as indicated by the AUC (Area Under Curve) of 0.8126. Random guessing yields an AUC of 0.5 (shown by the gray diagonal line), whereas a perfect model would have an AUC of 1.0.
Curve Form:
The curve's sharp initial climb indicates that the model can detect a small number of false positives and some true positives.
As it moves to the right, the curve progressively flattens, indicating diminishing returns—getting more true positives comes at the expense of getting more false positives.
The model is clearly learning significant patterns because the curve is continuously above the diagonal random guess line.
Lower thresholds (upper-right) will catch more true positives but generate more false alarms, while higher thresholds (lower-left) will be more selective but may miss some positive cases.

Complete code link

11. Prediction on Validation Dataset

```
[ ] lr_y_pred_val = lr_model.predict(val_data_pca)

[ ] lr_y_pred_proba_val = lr_model.predict_proba(val_data_pca)[:, 1]

    lr_y_pred_val[0:10]

    array([1, 0, 0, 0, 1, 1, 0, 0, 0, 0])

[ ] lr_y_pred_proba_val[0:10]

    array([0.5422657 , 0.07387796, 0.05654022, 0.0833386 , 0.60052156,
           0.81412922, 0.04919793, 0.11440673, 0.46997795, 0.32174341])
```

Creating the final prediction csv file with only two column(namely. account_number, predicted probability

12. Creating CSV FIle

```
# Create a DataFrame with account_number and the predicted probability
result_df = pd.DataFrame({
    'account_number': val_data['account_number'],
    'Predicted Probability': lr_y_pred_proba_val
})

[ ] result_df.head()
```

	account_number	Predicted Probability
0	100001	0.542266
1	100002	0.073878
2	100003	0.056540
3	100004	0.083339
4	100005	0.600522

Next steps: Generate code with result_df | View recommended plots | New interactive sheet

```
[ ] # Save the DataFrame to a CSV file
    result_df.to_csv('predicted_probabilities.csv', index=False)
    print("CSV file with predicted probabilities has been saved.")

    CSV file with predicted probabilities has been saved.
```

Complete code link

Thank you for your time and consideration