MACHINE LEARNING

PROJECT REPORT

 **Team: Lobia Masala** 

Submitted By

Siddharth Anil — IMT2023503
Krish Patel — IMT2023134
Chaitya Shah — IMT2023055

Project Repository

November 27, 2025

Contents

Model Training Methodology **1**

1 Financial Risk Profiling - Binomial Classification Dataset **2**
1.1 Overview . . . 2
1.2 Dataset Structure . . . 2
1.3 Target Variable Analysis . . . 2
1.3.1 Class Imbalance . . . 2
1.4 Preprocessing and Exploratory Data Analysis . . . 2
1.4.1 Outlier and Covariance Assessment . . . 2
1.4.2 Feature Preprocessing Pipeline . . . 3
1.4.3 Feature Engineering . . . 3
1.5 Models . . . 3
1.5.1 Logistic Regression Models . . . 3
1.5.2 Support Vector Machines (SVM) . . . 4
1.5.3 Neural Networks . . . 5
1.5.4 Decision Trees and Ensemble Methods . . . 7
1.6 Comparative Analysis & Conclusion . . . 8
1.6.1 Impact of Data Distribution (Skewed vs. Non-Skewed) . . . 8
1.6.2 Support Vector Machines (SVM) . . . 8
1.6.3 Neural Networks (NNs) . . . 8
1.6.4 Decision Trees (Random Forest & XGBoost) . . . 8
1.6.5 Impact of Dataset Size (80% vs. 20%) . . . 8
1.6.6 Support Vector Machines (SVM) . . . 8
1.6.7 Neural Networks (NNs) . . . 8
1.6.8 Decision Trees (Random Forest & XGBoost) . . . 8
1.7 Which models gave good results? . . . 9
1.8 Summary of Binomial Models . . . 9

2 Travel Behavior Insights - Multinomial Classification **10**
2.1 Overview . . . 10
2.2 Dataset Structure . . . 10
2.3 Target Variable Analysis . . . 10
2.3.1 Class Imbalance . . . 10
2.4 Preprocessing . . . 11
2.5 Initial Data Cleaning and Null Handling . . . 11
2.6 Ordinal Encoding of Range Features . . . 11
2.7 Outlier Removal . . . 11
2.8 Advanced Feature Engineering: Clustering . . . 11
2.9 Final Data Transformation Pipeline . . . 12
2.10 Models . . . 12
2.10.1 Decision Trees . . . 12
2.10.2 Multiclass Logistic Regression . . . 13
2.10.3 Neural Networks (MLP) . . . 14
2.10.4 Support Vector Machines (SVM) . . . 15
2.11 Conclusion . . . 16
2.11.1 Best Model Performance . . . 16
2.11.2 Impact of Skewed vs. Balanced Training Data . . . 16
2.11.3 Impact of Dataset Size (80% vs. 20% Train) . . . 17
2.12 Summary of Multiclass Models . . . 18

Model Training Methodology

Data Splitting Strategy

To rigorously evaluate model performance, we employed two distinct data splitting strategies:

1. **Split A (80/10/10)**:
    - **Training**: 80% of the dataset.
    - **Validation**: 10% of the dataset.
    - **Testing**: 10% of the dataset.
    - *Purpose*: To assess performance when ample training data is available.
2. **Split B (20/40/40)**:
    - **Training**: 20% of the dataset.
    - **Validation**: 40% of the dataset.
    - **Testing**: 40% of the dataset.
    - *Purpose*: To simulate low-resource scenarios and test model generalization with limited training data.

Data Distribution (Skewed vs. Non-Skewed)

We also experimented with the class distribution of the training data:

- **Skewed (Natural Distribution)**: The dataset is used as-is, preserving the original class imbalance. This reflects the real-world distribution of the data.
- **Non-Skewed (Balanced)**: We applied upsampling to the minority classes to match the size of the majority class. This technique aims to prevent the model from becoming biased toward the majority class.

Experimental Configurations

For every model architecture, we trained and evaluated four distinct configurations:

1. **80% Train - Skewed**
2. **80% Train - Non-Skewed**
3. **20% Train - Skewed**
4. **20% Train - Non-Skewed**

This allows us to analyze the impact of both dataset size and class balance on model performance.

1 Financial Risk Profiling - Binomial Classification Dataset

1.1 Overview

The Binomial Classification dataset is related to financial risk assessment, likely for loan applications or credit scoring. The objective is to predict the `RiskFlag`, which indicates whether an applicant poses a risk (e.g., default on a loan).

1.2 Dataset Structure

The dataset consists of the following columns:

Feature Name	Description
`ProfileID`	Unique identifier for the applicant.
`ApplicantYears`	Age of the applicant.
`AnnualEarnings`	Annual income of the applicant.
`RequestedSum`	The amount of money requested (e.g., loan amount).
`TrustMetric`	A score indicating the trustworthiness or credit score of the applicant.
`WorkDuration`	Duration of employment or work experience.
`ActiveAccounts`	Number of active financial accounts.
`OfferRate`	Interest rate offered or applicable.
`RepayPeriod`	Period for repayment (likely in months).
`DebtFactor`	A ratio or factor representing debt obligation.
`QualificationLevel`	Educational qualification (e.g., High School, Bachelor's).
`WorkCategory`	Employment type (e.g., Self-employed, Full-time).
`RelationshipStatus`	Marital or relationship status (e.g., Single, Married).
`OwnsProperty`	Binary indicator if the applicant owns property (Yes/No).
`FamilyObligation`	Binary indicator of family obligations (Yes/No).
`FundUseCase`	Purpose of the funds (e.g., Business, Education).
`JointApplicant`	Binary indicator if there is a joint applicant (Yes/No).
`RiskFlag`	**Target Variable**. Binary variable (0 or 1) indicating risk.

1.3 Target Variable Analysis

The target variable is `RiskFlag`. It is a binary variable where:

- **0**: Likely indicates low risk or non-default.
- **1**: Likely indicates high risk or default.

1.3.1 Class Imbalance

The dataset shows a **severe class imbalance**:

- **Class 0 (Low Risk)**: 88.4% (180,524 instances).
- **Class 1 (High Risk)**: 11.6% (23,753 instances).

A very high number of applicants are labeled as low risk (0). This significant skew means that a model predicting "Low Risk" for every applicant would achieve an accuracy of approximately 88.4%, but would fail to identify any high-risk applicants. This imbalance is a critical factor to consider during model training and evaluation, necessitating techniques like resampling (SMOTE, undersampling) or cost-sensitive learning.

1.4 Preprocessing and Exploratory Data Analysis

1.4.1 Outlier and Covariance Assessment

We analyzed the dataset using scatter plots and box plots to identify potential outliers. Upon visual inspection, no significant outliers were detected in the data. Additionally, we examined the covariance

between features and found it to be very low. Consequently, no features were removed, and no specific preprocessing steps such as outlier rejection or feature selection were performed.

1.4.2 Feature Preprocessing Pipeline

To prepare the data for modeling, we implemented a `ColumnTransformer` to apply specific transformations to numeric and categorical features:

- **Target Variable**: `RiskFlag`
- **Numeric Features**: The following 9 features were scaled using **StandardScaler** to ensure zero mean and unit variance:
  - `ApplicantYears`, `AnnualEarnings`, `RequestedSum`, `TrustMetric`, `WorkDuration`, `ActiveAccounts`, `OfferRate`, `RepayPeriod`, `DebtFactor`
- **Categorical Features**: The following 7 features were encoded using **OneHotEncoder**// (with `handle_unknown="ignore"` and `sparse_output=False`):
  - `QualificationLevel`, `WorkCategory`, `RelationshipStatus`, `FamilyObligation`, `OwnsProperty`, `FundUseCase`, `JointApplicant`

1.4.3 Feature Engineering

The following feature engineering steps were applied to the Binomial Classification dataset to prepare it for the ensemble model:

- **Feature Removal**: The `ProfileID` column was dropped as it is a unique identifier with no predictive value.
- **Feature Creation**:
  - `IncomeToLoanRatio`: A new feature representing the applicant's repayment capacity, calculated as:
$$IncomeToLoanRatio = \frac{AnnualEarnings}{RequestedSum + 1}$$
  - `CreditUtilization`: A proxy for credit usage relative to trustworthiness, calculated as:
$$CreditUtilization = \frac{RequestedSum}{TrustMetric + 1}$$
- **Categorical Encoding**: All categorical variables were encoded using `LabelEncoder`. The encoder was fitted on the combined training and test datasets to ensure consistent mapping of all categories.
- **Numerical Scaling**: Numerical features were standardized using `StandardScaler`, which scales data to have a mean of 0 and a standard deviation of 1.

1.5 Models

1.5.1 Logistic Regression Models

1a. Standard Binomial Logistic Regression

Method Description: A baseline Logistic Regression model is trained using the `lbfgs` solver with a maximum of 500 iterations. It models the probability of the binary outcome using the logistic function.

Results:

Dataset Size	Data Distribution	Validation Accuracy	Test Accuracy	Leaderboard
80%	Skewed	0.8854	0.8855	0.885
80%	Non-Skewed	0.6746	0.6760	0.677
20%	Skewed	0.8848	0.8855	0.885
20%	Non-Skewed	0.6788	0.6714	0.676

1b. Binomial Logistic Regression with Bayesian Optimization

Method Description: This approach utilizes **Bayesian Optimization** to exhaustively search for the optimal hyperparameters. The single set of "best" hyperparameters identified from this process was then applied to train models for all four data scenarios, testing the generalizability of the tuned configuration.

Results:

Dataset Size	Data Distribution	Validation Accuracy	Test Accuracy	Leaderboard
80%	Skewed	0.8854	0.8855	0.885
80%	Non-Skewed	0.6745	0.6761	0.677
20%	Skewed	0.8852	0.8856	0.885
20%	Non-Skewed	0.6775	0.6712	0.676

1c. Tuned Binomial Logistic Regression (Bayesian Approach + Random Search)

Method Description: This method employs a robust two-stage hyperparameter tuning process to optimize the regularization strength (`C`). First, a **Random Search** explores a wide logarithmic range of values. This is followed by **Bayesian Optimization** (using Gaussian Process via `skopt`), which refines the search around the most promising region found in the first stage. This tuning process is performed **independently** for each of the four data scenarios to find the specific optimal parameters for that distribution.

Results:

Dataset Size	Data Distribution	Validation Accuracy	Test Accuracy	Leaderboard
80%	Skewed	0.6744	0.6760	0.677
80%	Non-Skewed	0.6744	0.6760	0.677
20%	Skewed	0.6786	0.6715	0.676
20%	Non-Skewed	0.6786	0.6715	0.676

Note: The similar results across skewed and non-skewed versions in this method are because the optimal `class_weight='balanced'` found during the 80% Non-Skewed tuning was enforced on all models.

1.5.2 Support Vector Machines (SVM)

2a. Linear SVM

Method Description: The Linear SVM model is implemented using `LinearSVC`. It attempts to separate the classes with a linear hyperplane.

- **Skewed:** Trained with standard parameters (`C=1.0`).
- **Non-Skewed:** Trained with `class_weight='balanced'` to handle class imbalance.

Results:

Dataset Size	Data Distribution	Validation Accuracy	Test Accuracy	Leaderboard
80%	Skewed	0.8837	0.8837	0.884
80%	Non-Skewed	0.6725	0.6738	0.675
20%	Skewed	0.8837	0.8837	0.885
20%	Non-Skewed	0.6766	0.6698	0.674

2b. RBF SVM (Radial Basis Function)

Method Description: This method uses the `SVC` class with the RBF kernel (`kernel='rbf'`). RBF is a popular kernel that can handle complex non-linear relationships by measuring the similarity between data points.

Results:

Dataset Size	Data Distribution	Validation Accuracy	Test Accuracy	Leaderboard
80%	Skewed	0.8838	0.8837	0.884
80%	Non-Skewed	0.6997	0.7002	0.698
20%	Skewed	0.8837	0.8837	0.884
20%	Non-Skewed	0.7030	0.7125	0.710

2c. Polynomial SVM

Method Description: This method uses the `SVC` class with a polynomial kernel (`kernel='poly', degree=3`). It maps the input features into a higher-dimensional space to find a non-linear decision boundary.

Results:

Dataset Size	Data Distribution	Validation Accuracy	Test Accuracy	Leaderboard
80%	Skewed	0.8837	0.8837	0.884

*Note: Results were only found for the **80% Skewed** configuration in the artifacts as others were taking too long to train.*

2d. Sigmoid SVM

Method Description: This method uses the `SVC` class with the Sigmoid kernel (`kernel='sigmoid'`). The Sigmoid kernel is equivalent to a two-layer perceptron neural network.

Results:

Dataset Size	Data Distribution	Validation Accuracy	Test Accuracy	Leaderboard
20%	Skewed	0.5596	0.5598	0.563

*Note: Results were only found for a **20% Training Data** configuration as others were taking too long to train and also was not giving good results.*

1.5.3 Neural Networks

3a. MLP (Multi-Layer Perceptron)

Method Description: The MLP approach involves training fully connected Feed-Forward Neural Networks. Three distinct architectures were defined to test the impact of model complexity:

- **NN_Small:** A simple network with 1 hidden layer (64 units, ReLU) and Dropout (0.2).
- **NN_Medium:** A moderate network with 2 hidden layers (128, 64 units), Batch Normalization, and Dropout (0.3).
- **NN_Deep:** A complex network with 3 hidden layers (256, 128, 64 units), Batch Normalization, and Dropout (0.4/0.3/0.2).

All models use the Adam optimizer (`lr=1e-3`) and Binary Cross-Entropy loss.

Results:

NN_Small:

Dataset Size	Data Distribution	Validation Accuracy	Test Accuracy	Leaderboard
80%	Skewed	0.8871	0.8877	0.887
80%	Non-Skewed	0.6997	0.6998	0.699
20%	Skewed	0.8848	0.8859	0.887
20%	Non-Skewed	0.7019	0.7048	0.704

NN_Medium:

Dataset Size	Data Distribution	Validation Accuracy	Test Accuracy	Leaderboard
80%	Skewed	0.8871	0.8869	0.886
80%	Non-Skewed	0.7054	0.7092	0.706
20%	Skewed	0.8841	0.8861	0.886
20%	Non-Skewed	0.8185	0.8184	0.818

NN_Deep:

Dataset Size	Data Distribution	Validation Accuracy	Test Accuracy	Leaderboard
80%	Skewed	0.8867	0.8871	0.887
80%	Non-Skewed	0.6852	0.6858	0.685
20%	Skewed	0.8837	0.8856	0.885
20%	Non-Skewed	0.6850	0.6936	0.691

3b. MLP with Bayesian Optimization (Tuned)

Method Description: This method applies Bayesian Optimization (using `keras-tuner`) to find the optimal hyperparameters for the MLP architectures. The tuning search space includes the number of units, dropout rates, learning rate, and batch size.

Results:

Dataset Size	Data Distribution	Model	Validation Accuracy	Test Accuracy	Leaderboard
80%	Skewed	NN_Small	0.8868	0.8871	0.887
20%	Skewed	NN_Small	0.8837	0.8859	0.887

*Note: Results were only found for the **NN_Small** architecture on Skewed datasets as the **NN_Medium** and **NN_Deep** architectures were taking too long to train using Bayesian Tuning.*

3c. SMOTE (Synthetic Minority Over-sampling Technique)

Method Description: This approach addresses class imbalance by using SMOTE to oversample the minority class in the training data *before* training the neural networks. The same three architectures (Small, Medium, Deep) are used.

Skewed Dataset: The dataset was imbalanced, so SMOTE was directly applied to address the minority class and create a more balanced distribution. No class weights were used, as the application of SMOTE already balanced the class proportions.

Results:

NN_SMOTE_Small:

Dataset Size	Validation Accuracy	Test Accuracy	Leaderboard
80%	0.7974	0.8031	0.798
20%	0.7599	0.7653	0.765

NN_SMOTE_Medium:

Dataset Size	Validation Accuracy	Test Accuracy	Leaderboard
80%	0.8710	0.8725	0.874
20%	0.8328	0.8435	0.852

NN_SMOTE_Deep:

Dataset Size	Validation Accuracy	Test Accuracy	Leaderboard
80%	0.8780	0.8787	0.878
20%	0.8381	0.8390	0.840

1.5.4 Decision Trees and Ensemble Methods

4a. Random Forest

Method Description: A Random Forest Classifier is trained with 400 estimators. The implementation (`RandomForest.ipynb`) uses a standard pipeline with `StandardScaler` for numeric features and `OneHotEncoder` for categorical features. For the non-skewed (balanced) configuration, `class_weight='balanced'` is used.

Results:

Dataset Size	Data Distribution	Validation Accuracy	Test Accuracy	Leaderboard
80%	Skewed	0.8856	0.8854	0.886
80%	Non-Skewed	0.8847	0.8844	0.885

Note: This specific run (`RF2`) only contains results for the 80% dataset splits.

4b. XGBoost

Method Description: An XGBoost Classifier is trained with `n_estimators=400`, `learning_rate=0.1`, and `max_depth=6`. The implementation (`XGBoost.ipynb`) handles class imbalance for the non-skewed (balanced) configuration by calculating and setting the `scale_pos_weight` parameter (ratio of negative to positive samples). For skewed data, `scale_pos_weight` is set to 1.

Results:

Dataset Size	Data Distribution	Validation Accuracy	Test Accuracy	Leaderboard
80%	Skewed	0.8852	0.8859	0.887
80%	Non-Skewed	0.8852	0.8859	0.887
20%	Skewed	0.8831	0.8843	0.885
20%	Non-Skewed	0.8831	0.8843	0.885

Note: The XGBoost results are identical for Skewed and Non-Skewed configurations.

4c. XGBoost (AUC Scoring)

Method Description: This variation of the XGBoost implementation utilizes **AUC (Area Under the Curve)** as the evaluation metric during training, likely for early stopping or model selection, instead of the default logloss or error rate. The underlying model architecture and hyperparameters remain similar to the standard XGBoost method.

Results:

Dataset Size	Data Distribution	Validation Accuracy	Test Accuracy	Leaderboard
80%	Skewed	0.8852	0.8859	0.887
80%	Non-Skewed	0.8852	0.8859	0.887
20%	Skewed	0.8831	0.8843	0.885
20%	Non-Skewed	0.8831	0.8843	0.885

Note: The results for XGBoost (AUC) are identical to the standard XGBoost method (using F1 Score as evaluation metric), indicating that optimizing for AUC in this context yielded the same final accuracy performance.

1.6 Comparative Analysis & Conclusion

1.6.1 Impact of Data Distribution (Skewed vs. Non-Skewed)

We observed distinct behaviors across model families when training on Skewed (natural) versus Non-Skewed (balanced via upsampling) data.

1.6.2 Support Vector Machines (SVM)

- **Skewed Data**: SVMs achieved high accuracy ($\sim$88.4%), effectively mimicking the baseline. They prioritized overall accuracy by favoring the majority class.
- **Non-Skewed Data**: Performance dropped significantly to $\sim$67-70%. By forcing the model to treat classes equally, the SVM decision boundary shifted, likely increasing false positives for the majority class. This trade-off suggests that while balanced training helps detect risk, it comes at a heavy cost to overall precision.

1.6.3 Neural Networks (NNs)

- **Skewed Data**: Similar to SVMs, NNs on skewed data achieved baseline accuracy ($\sim$88.7%).
- **Non-Skewed Data**: Standard NNs struggled on balanced data, dropping to $\sim$70%. However, the **SMOTE-augmented NNs** (specifically Medium and Deep architectures) showed the most promise. They achieved $\sim$87.8% accuracy—slightly below the baseline—which indicates they were actually learning to classify the minority class rather than just ignoring it.

1.6.4 Decision Trees (Random Forest & XGBoost)

- **Robustness**: These models were remarkably robust to data distribution. Whether trained on Skewed or Non-Skewed data, they consistently maintained high accuracy ($\sim$88.5%).
- **Conclusion**: Tree-based ensembles handled the class imbalance naturally, likely due to their hierarchical structure which can isolate minority class regions without compromising the majority class predictions.

1.6.5 Impact of Dataset Size (80% vs. 20%)

We analyzed whether reducing the training data from 80% to 20% affected model performance.

1.6.6 Support Vector Machines (SVM)

- **Minimal Impact**: There was almost no difference in performance between the 80% and 20% splits. This suggests that the decision boundary for this dataset is relatively simple (or the overlap is consistent), and the SVM could learn it effectively even with a smaller subset of data.

1.6.7 Neural Networks (NNs)

- **Sensitivity**: NNs showed some volatility. For instance, the **NN_Medium** model on Non-Skewed data jumped from $\sim$70% (80% split) to $\sim$81% (20% split). This suggests that deep learning models were more sensitive to the specific composition of the training set when data was limited, potentially overfitting or finding different local minima.

1.6.8 Decision Trees (Random Forest & XGBoost)

- **High Efficiency**: These models showed negligible performance loss when reducing data size. They were able to capture the primary predictive patterns just as effectively with 20% of the data as with 80%, highlighting their data efficiency for this specific tabular problem.

1.7 Which models gave good results?

- **Top Performers:** XGBoost and Random Forest consistently achieved the highest accuracy, in the range of $\sim$ 88.5% to $\sim$ 88.7% across nearly all configurations.
- **Neural Networks:** The `NN_Small` model trained on skewed data also reached a similar performance level ($\sim$ 88.7%).
- **Baseline Reality:** The dataset is heavily imbalanced with $\sim$ 88.4% belonging to Class 0. Therefore, models achieving around 88.5% accuracy are likely predicting the majority class ("Low Risk") for most samples — meaning high accuracy does not necessarily indicate true learning.
- **The Real Winners:** Models trained using SMOTE + Neural Networks (particularly `Medium` and `Deep`) achieved $\sim$ 87.8%. Although slightly lower in accuracy, these models are more meaningful because they are not just memorizing the majority class. Their slight drop in accuracy suggests they are genuinely learning to detect the minority "High Risk" class.

1.8 Summary of Binomial Models

Table 2: Performance Summary of Binomial Classification Models

Model	Configuration	Val Acc	Test Acc	LB Score
Logistic Regression Models				
Logistic Regression (Std)	80% Skewed	0.8854	0.8855	0.885
Logistic Regression (Bayes)	80% Skewed	0.8854	0.8855	0.885
Logistic Regression (Tuned)	20% Skewed	0.6786	0.6715	0.676
Support Vector Machines (SVMs)				
Linear SVM	80% Skewed	0.8837	0.8837	0.884
RBF SVM	80% Skewed	0.8838	0.8837	0.884
Polynomial SVM	80% Skewed	0.8837	0.8837	0.884
Sigmoid SVM	20% Skewed	0.5596	0.5598	0.563
Neural Networks				
NN Small	80% Skewed	0.8871	0.8877	0.887
NN Medium	80% Skewed	0.8871	0.8869	0.886
NN Deep	80% Skewed	0.8867	0.8871	0.887
NN Small (Tuned)	80% Skewed	0.8868	0.8871	0.887
NN SMOTE Small	80% (SMOTE)	0.7974	0.8031	0.798
NN SMOTE Medium	80% (SMOTE)	0.8710	0.8725	0.874
NN SMOTE Deep	80% (SMOTE)	0.8780	0.8787	0.878
Decision Tree Models				
Random Forest	80% Skewed	0.8856	0.8854	0.886
XGBoost	80% Skewed	0.8852	0.8859	0.887
XGBoost (AUC)	80% Skewed	0.8852	0.8859	0.887

2 Travel Behavior Insights - Multinomial Classification

2.1 Overview

The Multiclassification dataset appears to be related to tourism and travel expenditure. The goal is likely to predict the `spend_category` of a traveler based on various demographic and trip-related features. The dataset contains information about the traveler's origin, companions, activities, and trip details.

2.2 Dataset Structure

The dataset consists of the following columns:

Feature Name	Description
`trip_id`	Unique identifier for the trip/traveler.
`country`	Country of origin or destination (likely origin based on context).
`age_group`	Age range of the traveler (e.g., 25-44, 45-64).
`travel_companions`	Who the traveler is accompanying (e.g., With Spouse, Alone).
`num_females`	Number of females in the group.
`num_males`	Number of males in the group.
`main_activity`	Primary activity during the trip (e.g., Beach Tourism, Wildlife Tourism).
`visit_purpose`	Purpose of the visit (e.g., Leisure and Holidays, Business).
`is_first_visit`	Binary indicator if it is the first visit (Yes/No).
`mainland_stay_nights`	Number of nights stayed on the mainland.
`island_stay_nights`	Number of nights stayed on islands.
`tour_type`	Type of tour (e.g., Independent, Package Tour).
`intl_transport_included`	Whether international transport is included (Yes/No).
`info_source`	Source of information about the trip (e.g., Travel agent, Friends).
`accomodation_included`	Whether accommodation is included (Yes/No).
`food_included`	Whether food is included (Yes/No).
`domestic_transport_included`	Whether domestic transport is included (Yes/No).
`sightseeing_included`	Whether sightseeing is included (Yes/No).
`guide_included`	Whether a guide is included (Yes/No).
`insurance_included`	Whether insurance is included (Yes/No).
`days_booked_before_trip`	Timeframe of booking before the trip.
`arrival_weather`	Weather conditions upon arrival.
`total_trip_days`	Duration of the trip.
`has_special_requirements`	Any special requirements (e.g., dietary needs).
`spend_category`	**Target Variable**. Categorical variable indicating the spending level (e.g., 0.0, 1.0, 2.0).

2.3 Target Variable Analysis

The target variable is `spend_category`, which is a multiclass variable. It classifies the traveler's spending behavior into distinct categories.

2.3.1 Class Imbalance

The dataset exhibits a class imbalance across the spending categories:

- **Category 0.0**: 49.5% (6,245 instances) - Represents the majority class.
- **Category 1.0**: 38.9% (4,911 instances) - A significant portion of the data.
- **Category 2.0**: 11.6% (1,464 instances) - The minority class.

This distribution indicates that the model might be biased towards predicting lower spending categories (0.0 and 1.0) if not addressed properly. The minority class (2.0) is underrepresented, constituting only about 11.6% of the dataset.

2.4 Preprocessing

The preprocessing pipeline was designed to handle the complexity of the multinomial classification task, ensuring data quality and creating informative features for the neural network models.

2.5 Initial Data Cleaning and Null Handling

- **Target Variable Cleaning**: We identified and removed 34 rows where the target variable `spend_category` was null, as these samples cannot be used for supervised learning.
- **String Sanitization**: All object-type columns were processed to strip leading/trailing whitespace and remove trailing commas, ensuring consistency in categorical values.
- **Binary Feature Standardization**:
  - Columns containing "Yes"/"No" values (e.g., `is_first_visit`, `food_included`, `intl_transport_included`) were mapped to binary integers (1/0).
  - The `has_special_requirements` column was converted to a binary flag: 0 if "None", "NaN", or empty; 1 otherwise.

2.6 Ordinal Encoding of Range Features

To preserve the order inherent in range-based features, we applied specific ordinal mappings instead of one-hot encoding:

- `days_booked_before_trip`: Mapped to an ordinal scale of 1-6:
  - "1-7" $\rightarrow$ 1, "8-14" $\rightarrow$ 2, "15-30" $\rightarrow$ 3, "31-60" $\rightarrow$ 4, "61-90" $\rightarrow$ 5, "90+" $\rightarrow$ 6.
- `total_trip_days`: Mapped to an ordinal scale of 1-4:
  - "1-6" $\rightarrow$ 1, "7-14" $\rightarrow$ 2, "15-30" $\rightarrow$ 3, "30+" $\rightarrow$ 4.
- **Imputation Strategy**: Missing values in these ordinal columns (and other categorical features) were imputed using the **mode** (most frequent value) to maintain data integrity without introducing synthetic noise.

2.7 Outlier Removal

We performed a rigorous outlier analysis and removal process based on domain knowledge and distribution tails. Approximately 71 rows were removed in total based on the following thresholds:

- `num_females`: Removed records with > 10 females (28 rows).
- `num_males`: Removed records with > 10 males (8 rows).
- `mainland_stay_nights`: Removed trips exceeding 90 nights (27 rows).
- `island_stay_nights`: Removed trips exceeding 60 nights (8 rows).

2.8 Advanced Feature Engineering: Clustering

To capture complex, non-linear relationships between traveler attributes, we employed unsupervised learning as a feature engineering step:

- **Algorithm**: K-Means Clustering.
- **Input Features**: Standardized numeric features (`num_females`, `num_males`, `mainland_stay_nights`, `island_stay_nights`, `days_booked_before_trip_ord`, `total_trip_days_ord`).
- **Configuration**: `n_clusters=6`, `random_state=42`.
- **Optimal K Selection**: We determined the optimal number of clusters (K=6) using a combination of quantitative and visual methods:

  - **Elbow Method**: Analyzed the Within-Cluster Sum of Squares (Inertia) plot to identify the "elbow" point where variance reduction slows down.
  - **Silhouette Analysis**: Calculated Silhouette Scores for K ranging from 2 to 10. The score peaked at K=6, indicating the best separation and cohesion.
  - **PCA Visualization**: Projected the data into 2D space using Principal Component Analysis (PCA) to visually confirm the distinctness of the 6 clusters.
- **Output**: A new categorical feature `kmeans_cluster` was assigned to each sample, representing the "traveler profile" cluster it belongs to. This feature allows the downstream classifier to learn cluster-specific patterns.

2.9 Final Data Transformation Pipeline

The final dataset was processed using a `ColumnTransformer` before feeding into the models:

- **Numeric Features**: Standardized using `StandardScaler` to ensure zero mean and unit variance, crucial for Neural Network convergence.
- **Categorical Features**: Encoded using `OneHotEncoder` (including the generated `kmeans_cluster` feature), with `handle_unknown='ignore'` to robustly handle unseen categories in test data.
- **Class Imbalance Handling**: We applied **SMOTE** (Synthetic Minority Over-sampling Technique) to the training set. This generated synthetic samples for minority classes in the `spend_category` target, ensuring the model does not become biased toward the majority class.

2.10 Models

2.10.1 Decision Trees

1.1 CatBoost **Method Description:** CatBoost is a gradient boosting algorithm that handles categorical features automatically.

- **Skewed**: Trained on the natural distribution of the dataset.
- **Non-Skewed**: Trained on a balanced dataset created by upsampling minority classes.

Results:

Dataset Size	Data Distribution	Train Acc	Val Acc	LB Score
80%	Skewed	0.8113	0.7625	0.679
80%	Non-Skewed	0.8991	0.7458	0.700
20%	Skewed	0.8477	0.7508	-
20%	Non-Skewed	0.9284	0.7340	-

1.2 XGBoost **Method Description:** XGBoost is an optimized distributed gradient boosting library.

- **Skewed**: Trained on the natural distribution.
- **Non-Skewed**: Trained on a balanced dataset (upsampled).

Results:

Dataset Size	Data Distribution	Train Acc	Val Acc	LB Score
80%	Skewed	0.9206	0.7450	0.675
80%	Non-Skewed	0.9559	0.7363	0.700
20%	Skewed	0.9968	0.7365	-
20%	Non-Skewed	0.9978	0.7173	0.663

1.3 Ensemble (XGBoost 0.4 + CatBoost 0.6) **Method Description:** This method combines the predictions of XGBoost and CatBoost using a weighted average (0.4 for XGBoost, 0.6 for CatBoost).

Results:

Dataset Size	Data Distribution	Train Acc	Val Acc	LB Score
80%	Skewed	0.8598	0.7554	-
80%	Non-Skewed	0.9287	0.7458	0.705
20%	Skewed	0.9398	0.7432	-
20%	Non-Skewed	0.9820	0.7339	-

1.4 Ensemble (XGBoost + CatBoost) **Method Description:** This method uses a different weighting scheme or a voting mechanism between XGBoost and CatBoost.

Results:

Dataset Size	Data Distribution	Train Acc	Val Acc	LB Score
80%	Skewed	0.8774	0.7498	0.683
80%	Non-Skewed	0.9429	0.7482	0.703
20%	Skewed	0.9737	0.7428	0.646
20%	Non-Skewed	0.9903	0.7313	0.672

1.5 Random Forest **Method Description:** This method uses Random Forests.

Results:

Dataset Size	Data Distribution	Val Acc	LB Score
80%	Skewed	0.7514	0.655
80%	Non-Skewed	0.7355	0.693
20%	Skewed	0.7402	0.629
20%	Non-Skewed	0.7279	0.664

2.10.2 Multiclass Logistic Regression

2.1 Bayesian Approach (Best) **Description:** This method utilizes Bayesian Optimization (specifically using Gaussian Processes) to tune the hyperparameters of the Logistic Regression model, primarily the inverse regularization strength `C`.

Results:

Dataset Config	Train Acc	Val Acc	LB Score
80% Training (Skewed)	0.7642	0.7580	0.664
80% Training (Non-Skewed)	0.7319	0.7112	0.690
20% Training (Skewed)	0.7938	0.7331	0.645
20% Training (Non-Skewed)	0.7882	0.6972	0.656

2.2 Multiclass LR Outlier3 (Best) **Description:** This variation applies specific outlier removal thresholds to the training data before fitting the model.

Results:

Dataset Config	Train Acc	Val Acc	LB Score
80% Training (Skewed)	0.7614	0.7575	0.665
80% Training (Non-Skewed)	0.7322	0.7131	0.690
20% Training (Skewed)	0.7869	0.7470	0.643
20% Training (Non-Skewed)	0.7835	0.7072	0.665

2.3 Multiclass LR With 6 K-Means Clustering (Best) **Description:** This method incorporates feature engineering using K-Means clustering (6 clusters) and adds the cluster labels as a new categorical feature.

Results:

Dataset Config	Train Acc	Val Acc	LB Score
80% Training (Skewed)	0.7645	0.7525	-
80% Training (Non-Skewed)	0.7332	0.7156	0.690
20% Training (Skewed)	0.7864	0.7371	-
20% Training (Non-Skewed)	0.7868	0.7072	-

2.4 Multiclass LR With 6 Clustering (Proper) (K-Means + GMM) **Description:** An extension of the clustering approach that utilizes both K-Means and Gaussian Mixture Models (GMM) for feature engineering.

Results:

Dataset Config	Train Acc	Val Acc	LB Score
80% Training (Skewed)	0.7653	0.7510	0.667
80% Training (Non-Skewed)	0.7306	0.7156	0.688
20% Training (Skewed)	0.7923	0.7331	0.642
20% Training (Non-Skewed)	0.7795	0.7072	0.658

2.5 Random + Bayesian **Description:** This method employs a two-stage hyperparameter tuning strategy (Random Search followed by Bayesian Optimization).

Results:

Dataset Config	Train Acc	Val Acc	LB Score
80% Training (Skewed)	0.7620	0.7590	0.664
80% Training (Non-Skewed)	0.7287	0.7236	0.689
20% Training (Skewed)	0.7644	0.7530	0.603
20% Training (Non-Skewed)	0.7711	0.7131	0.659

2.10.3 Neural Networks (MLP)

3.1 MLP Classifier **Description:** This approach uses Scikit-Learn's `MLPClassifier` with three different architectures (Small, Medium, Large) and K-Means clustering features.

Results:

MLP Small:

Dataset Config	Train Dist.	Val Acc	Test Acc	LB Score
80% Train	Skewed	0.6725	0.6733	0.618
80% Train	Non-Skewed	0.6582	0.6343	0.608
20% Train	Skewed	0.6649	0.6763	0.618
20% Train	Non-Skewed	0.6608	0.6661	0.612

MLP Medium:

Dataset Config	Train Dist.	Val Acc	Test Acc	LB Score
80% Train	Skewed	0.6606	0.6900	0.607
80% Train	Non-Skewed	0.6757	0.6685	0.627
20% Train	Skewed	0.6829	0.6797	0.606
20% Train	Non-Skewed	0.6681	0.6779	0.621

MLP Large:

Dataset Config	Train Dist.	Val Acc	Test Acc	LB Score
80% Train	Skewed	0.6622	0.6582	0.614
80% Train	Non-Skewed	0.6813	0.6749	0.619
20% Train	Skewed	0.6853	0.6873	0.643
20% Train	Non-Skewed	0.6673	0.6747	0.614

3.2 MLP with SMOTE **Description:** This method uses Keras `Sequential` models with similar architectures (Small, Medium, Large) but applies **SMOTE** to the training data.

Results:

Model Architecture	Dataset Config	Val Acc	Test Acc	LB Score
MLP Small	80% Train	0.7267	0.7482	0.704
MLP Small	20% Train	0.7070	0.7137	0.677
MLP Medium	80% Train	0.7100	0.7315	0.687
MLP Medium	20% Train	0.7110	0.7137	0.668
MLP Large	80% Train	0.7139	0.7219	0.678
MLP Large	20% Train	0.7072	0.7066	0.664

3.3 MLP with SMOTE+Bayesian **Description:** This method combines SMOTE for data balancing with Bayesian Optimization (via Optuna).

Results:

Validation Accuracy	Test Accuracy	LB Score
0.7506	0.7777	0.661

2.10.4 Support Vector Machines (SVM)

4.1 SVM (No Tuning) **Description:** This approach evaluates standard SVM kernels (Linear, RBF, Poly, Sigmoid) without hyperparameter tuning.

Results:

Kernel	Dataset Config	Train Dist.	Val Acc	LB Score
RBF	80% Train	Skewed	0.7418	0.571
RBF	80% Train	Non-Skewed	0.7155	0.571
RBF	20% Train	Skewed	0.7335	0.554
RBF	20% Train	Non-Skewed	0.7048	0.597
Linear	80% Train	Skewed	0.7371	0.614
Linear	80% Train	Non-Skewed	0.7020	0.690
Linear	20% Train	Skewed	0.7281	0.598
Linear	20% Train	Non-Skewed	0.6986	0.665
Sigmoid	80% Train	Skewed	0.6566	0.614
Sigmoid	80% Train	Non-Skewed	0.5992	0.682
Sigmoid	20% Train	Skewed	0.6823	0.553
Sigmoid	20% Train	Non-Skewed	0.6299	0.655
Poly	80% Train	Skewed	0.7482	0.616
Poly	80% Train	Non-Skewed	0.7108	0.667
Poly	20% Train	Skewed	0.7307	0.621
Poly	20% Train	Non-Skewed	0.6918	0.663

4.2 SVM (Tuned) **Description:** This approach uses Bayesian Optimization (via Optuna) to tune hyperparameters for Linear and RBF kernels.

Results:

Kernel	Dataset Config	Train Dist.	Best Acc	LB Score
Linear	80% Train	Skewed	0.7394	0.616
RBF	80% Train	Skewed	0.7514	0.634
Linear	80% Train	Non-Skewed	0.7203	0.680
RBF	80% Train	Non-Skewed	0.7171	0.686
Linear	20% Train	Skewed	0.7315	0.598
RBF	20% Train	Skewed	0.7382	0.621
Linear	20% Train	Non-Skewed	0.7034	0.665
RBF	20% Train	Non-Skewed	0.7175	0.656

2.11 Conclusion

2.11.1 Best Model Performance

- Tree-based **Gradient Boosting models** demonstrated the strongest results on the Kaggle leaderboard.
  - **CatBoost (Skewed 80% Train)** $\rightarrow$ 0.679
  - **XGBoost (Skewed 80% Train)** $\rightarrow$ 0.675
- The **top-ranked leaderboard submission** was achieved using an **ensemble approach**:
  - **XGBoost 0.4 + CatBoost 0.6 (Balanced 80% Train)** $\rightarrow$ **0.705** (Best Overall)
- **Neural Networks (MLP)** were competitive only at small scale:
  - **MLP Small + SMOTE (80% Train)** $\rightarrow$ 0.704 leaderboard score
  - Further tuning using SMOTE + Bayesian increased test accuracy (0.7777) but resulted in a lower leaderboard gain (0.661), showing that boosting generalizes better for leaderboard evaluation.
- **SVM models**, even when tuned, achieved moderate scores (best 0.634) and were more sensitive to data imbalance and training size.
- **Random Forest** lagged behind boosting and small MLP models, especially on reduced or balanced splits.

Overall Ranking Trend: Boosting Ensemble $>$ CatBoost $>$ XGBoost $>$ MLP Small (Balanced 80%) $>$ SVM (Tuned) $>$ Random Forest $>$ Larger MLPs

2.11.2 Impact of Skewed vs. Balanced Training Data

- **Balancing the training data improved generalization on Kaggle**, especially for:
  - SVM (Linear kernel improved from 0.614 $\rightarrow$ 0.690 when balanced)
  - Boosting Ensembles (best performer at 0.705)
- **Validation accuracy slightly dropped for boosting when balanced**, indicating synthetic noise from upsampling, but leaderboard performance still improved.
- **Medium & Large MLP networks did not gain significant leaderboard benefits from balancing** and incurred much longer training time.
- **SMOTE was more effective than simple upsampling**, especially for small neural networks, but still could not outperform boosted tree generalization at larger scales.

Key Observation:
Balancing helps fairness and leaderboard performance
But may slightly reduce raw validation accuracy due to synthetic noise

2.11.3 Impact of Dataset Size (80% vs. 20% Train)

- **80% training allowed better learning for most models**, especially boosting and small MLPs.
- **20% training significantly reduced performance**, but:
  - **XGBoost, CatBoost, and Ensembles remained resilient**, still scoring within ∼0.66–0.68 range on Kaggle.
  - **Small MLP showed stability**, maintaining nearly the same Kaggle score for 80% and 20% training (∼0.618 both skewed, ∼0.612 balanced).
  - **SVM performance dropped the most** under limited training data.

Key Observation:
Boosted trees degrade gracefully when data is limited
SVM and larger MLPs struggle when train size is extremely small

Table 4: Final Takeaways: Model Performance & Robustness

Model Category	**Leaderboard Strength**	**Data Balancing Benefit**	**Low-Resource (20%) Robustness**
Gradient Boosting	**Highest**	**High** (slight val drop)	**Most Robust**
Ensemble (XGB+Cat)	**Best Overall (0.705)**	**Maximum Benefit**	**Highly Resilient**
MLP Neural Network	Good only at Small scale	Helped small NN most	Stable only for small NN
SVM	Moderate (even tuned)	**Largest Benefit**	Least Robust
Random Forest	Lower vs Boosting	Minor/Negative	Low

Final Conclusion

The ensemble of XGBoost and CatBoost trained on 80% balanced data generalized best on Kaggle's leaderboard, while individual gradient-boosted decision trees remained the most robust to both class skew and dataset size reduction. Data balancing improved generalization and fairness, particularly for SVM and ensemble models, but introduced slightly lower validation accuracy in some cases. Training on only 20% of data caused performance degradation for most algorithms, except boosted trees and small MLP networks, which showed relative stability. Overall, gradient-boosting ensembles proved to be the best choice for leaderboard success and real-world multinomial classification generalization.

2.12 Summary of Multiclass Models

Table 5: Performance Summary of Multiclass Classification Models

Model	Best Configuration	Train Acc	Val Acc	LB Score
Decision Tree and Boosting Models				
CatBoost	80% Non-Skewed	0.8991	0.7458	0.700
XGBoost	80% Non-Skewed	0.9559	0.7363	0.700
Ensemble (XGB0.4 + Cat0.6)	80% Non-Skewed	0.9287	0.7458	**0.705**
Ensemble (XGB + Cat)	80% Non-Skewed	0.9429	0.7482	0.703
Random Forest	80% Non-Skewed	-	0.7355	0.693
Logistic Regression Models				
LR (Bayesian)	80% Non-Skewed	0.7319	0.7112	0.690
LR (Outlier3)	80% Non-Skewed	0.7322	0.7131	0.690
LR (K-Means)	80% Non-Skewed	0.7332	0.7156	0.690
LR (Proper Clustering)	80% Non-Skewed	0.7306	0.7156	0.688
LR (Random + Bayes)	80% Non-Skewed	0.7287	0.7236	0.689
Neural Network Models (MLP)				
MLP Small	80% Skewed	-	0.6725	0.618
MLP Medium	80% Non-Skewed	-	0.6757	0.627
MLP Large	20% Skewed	-	0.6853	0.643
MLP SMOTE Small	80% (SMOTE)	-	0.7267	0.704
MLP SMOTE Medium	80% (SMOTE)	-	0.7100	0.687
MLP SMOTE Deep	80% (SMOTE)	-	0.7139	0.678
MLP SMOTE+Bayesian	-	-	0.7506	0.661
Support Vector Machines (SVMs)				
SVM Linear (Tuned)	80% Non-Skewed	-	0.7203	0.680
SVM RBF (Tuned)	80% Non-Skewed	-	0.7171	0.686