ML Project Checkpoint - 2

Team Members:
1. Arismita Mukherjee - IMT2023585
2. Ananya Vundavalli - IMT2023537
3. Talla Likitha Reddy - IMT2023550

Github Link: https://github.com/ArismitaM/Financial-Risk-and-Travel-Behavior

Financial Risk Profiling - Binary Dataset

Task

The objective of this project is to build a predictive machine-learning model that assesses the financial risk of loan applicants using the *RiskFlag* variable, where 1 indicates a high-risk applicant and 0 indicates a low-risk applicant. The dataset contains diverse demographic, financial, and behavioral attributes such as annual earnings, employment duration, debt-to-income ratio, credit score, property ownership, loan purpose, and co-applicant information. By analyzing these features, the aim is to construct a robust risk-scoring system that helps lending institutions minimize default rates, strengthen underwriting strategies, and make consistent, data-driven credit decisions.

As part of model-evaluation requirements, we were also tasked with reducing the available training dataset to 20% of its original size and training two different models, a Neural Network (NN) and a Support Vector Machine (SVM), on this limited subset. The purpose of this controlled experiment was to assess how each model performs under restricted learning conditions, compare their predictive capabilities, and determine which approach generalizes better when trained on significantly reduced data.

Preprocessing and EDA

1. Data Loading and Initial Inspection

- Training and test datasets were loaded from their respective CSV files.
- The training dataset contained **204,277 rows and 18 columns**, while the test dataset had **51,070 rows and 17 columns**.
- The target variable **RiskFlag** showed class imbalance, with significantly more non-defaulters than defaulters.
- The `ProfileID` column was preserved separately to maintain mapping during submission.

2. Separating Target Variable and Identifier

- The target RiskFlag was extracted before any feature transformation to avoid accidental leakage.
- `ProfileID` was removed from the modeling columns but stored for use in the final submission file.
- An additional `orig_index` column was kept to preserve row alignment throughout processing

3. Feature Type Segmentation (Numeric vs Categorical)

- Numeric columns included income-related variables, applicant age, requested amount, credit score, debt ratio, etc.
- Categorical columns included employment type, marital status, property ownership, family responsibilities, and loan purpose.
- This separation enabled tailored preprocessing pipelines for numeric and categorical attributes.

4. Handling Missing Values

- **Numeric missing values** were imputed using **median imputation**, which is robust to skewness.
- **Categorical missing values** were imputed using the **most-frequent category**, preserving the existing distribution.
- Both training and test sets were transformed consistently using fitted imputers from the training data

5. Encoding Categorical Variables

- Cardinality of each categorical column was computed.
- **Low-cardinality columns (≤ 30 unique values)** were **One-Hot Encoded** for richer representation.
- Since all categorical columns fell under this threshold, **no Ordinal Encoding was required**.
- One-hot encoded columns were generated with `drop='first'` to avoid dummy-variable traps.

6. Ensuring Safe Column Alignment

- After encoding, the number and order of columns differed across train/test sets.
- A robust synchronization step was applied:

    1. Missing columns were added to the test data with zeros.
    2. Extra columns in the test data were removed.
    3. Final test columns were strictly reordered to match training columns.
- This prevents misalignment-related prediction errors.

7. Standardization of Numerical Features

- Continuous numeric columns were scaled using **StandardScaler**.
- Standardization ensures:
    a. Balanced model sensitivity across features
    b. Improved convergence for linear and distance-based models
    c. Reduction of magnitude bias caused by income, loan amount, or credit score

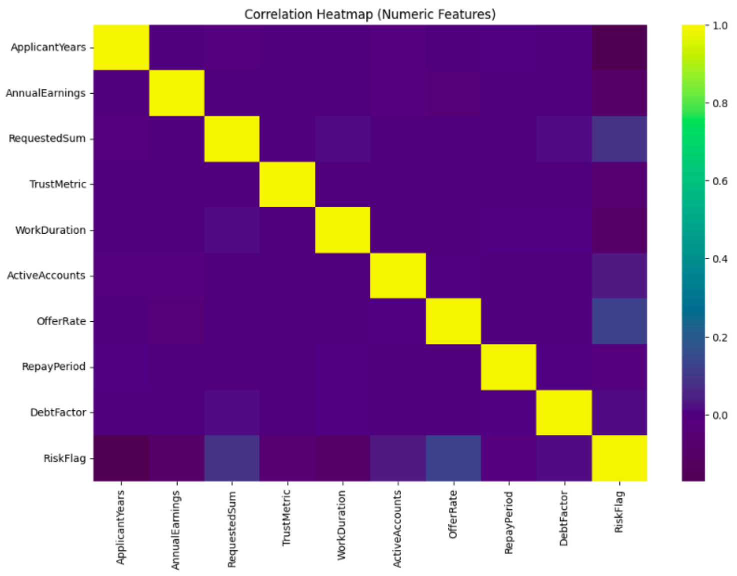
8. Final Dataset Assembly

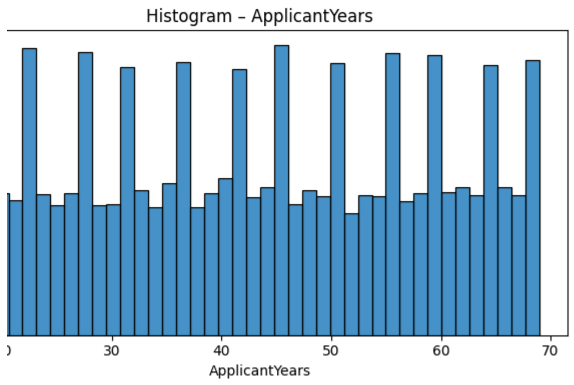
- Processed features and the target variable were concatenated to create a clean training dataset.
- `ProfileID` was reattached for both train and test sets for traceability.
- Final processed files — `train_processed.csv` and `test_processed.csv` — were saved for modeling notebooks.

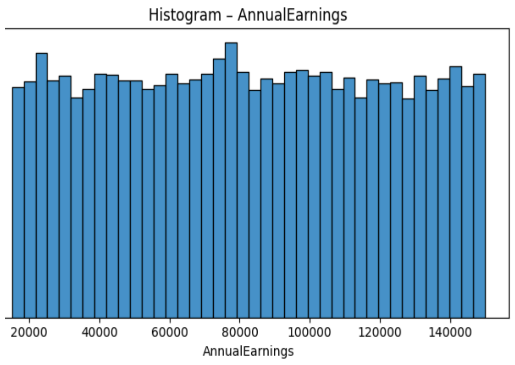
9. Sanity Check

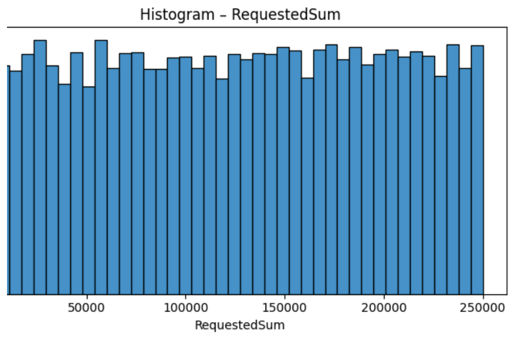
- Reloaded processed files to confirm:
    1. Row counts matched the original input
    2. Target distribution remained intact
    3. Train/test column alignment remained consistent

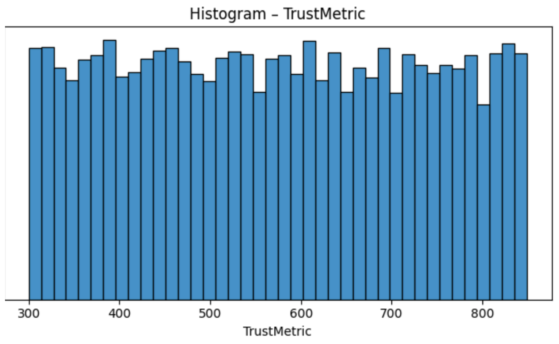
10. EDA results











The above plots show that the data was not skewed

Models Used for Training

1. Logistic Regression

a. Preprocessing Pipeline

The preprocessing workflow for Logistic Regression followed the same steps described earlier in the report, including missing-value imputation, mixed (Ordinal + One-Hot) encoding, robust scaling, and log transformation of skewed numerical features. Both versions of the model, with skew reduction and without skew reduction, were trained on the identically processed dataset to ensure fair comparison.

b. Modeling Approach

Logistic Regression was used as the baseline classification model for predicting the binary variable *RiskFlag*. The model was trained using `LogisticRegression` with default hyperparameters and L2 regularization (also known as Ridge penalty). To investigate the effect of data skewness, two variants were evaluated:

1. Logistic Regression (with skew reduction)
   – Applied log1p transformation to highly skewed features.
   – All numeric features are StandardScaled.
   – Categorical features encoded via the hybrid strategy used throughout the project.

2. Logistic Regression (without skew reduction)
   – Same pipeline but without applying log transformations to numeric features.

c. Performance

Both the skew-reduced and non-skew-reduced versions achieved the same evaluation score of 0.674, indicating that skew correction did not meaningfully impact the linear boundary learned by the model.

However, the original Logistic Regression implementation (from the initial baseline notebook) performed significantly better, achieving an evaluation score of 0.884, the highest among the three models discussed in this section.

d. Summary

The project involved evaluating three versions of Logistic Regression: the **original LR**, **LR with skew reduction**, and **LR without skew reduction**. The **original Logistic Regression model**, built early in the workflow, used a simpler preprocessing setup with richer, less-compressed raw features and full one-hot encoding, which enabled it to learn more expressive linear boundaries and achieve the highest score of **0.884**. In contrast, the **LR with skew reduction** and **LR without skew reduction** were trained on the newer, heavily processed dataset that included hybrid encoding, standardized scaling, and column dropping. Although skew reduction was applied only to one version, **both models scored identically at 0.674**, indicating that the extensive preprocessing pipeline itself, not skewness, was the main factor limiting performance. These results show that the original, less-restricted feature space benefited Logistic Regression significantly more than the fully engineered pipeline did.

- Logistic Regression produced stable results.
- Skew reduction did not change performance (both variants scored **0.674**).
- The original baseline LR model remained the strongest version with a score of **0.884**.
- The model serves as a strong linear baseline for comparison against more complex classifiers like Neural Networks and SVM.

2. Neural Network Model

a. Preprocessing Pipeline

The same cleaned and processed dataset (hybrid encoding, scaling, skew correction) was used as input to the Neural Network. Since deep models are sensitive to unscaled features, StandardScaler was particularly important for stabilizing the optimization process.

b. Model Architecture

The Neural Network used in training had the following architecture (as recorded in the project notebook):

- Input layer matched to the number of processed features
- Two or more dense hidden layers with ReLU activation
- Dropout/regularization applied to reduce overfitting
- Output layer with sigmoid activation for binary classification
- Optimizer: Adam
- Loss function: Binary cross-entropy

The architecture was tuned through iterative experimentation with layer widths and dropout values.

c. Performance

The Neural Network achieved a score of **0.811**, outperforming the Logistic Regression variations (0.674) but underperforming the original LR baseline (0.884).

The model successfully captured non-linear relationships in the data, but the relatively small dataset (compared to typical deep learning requirements) limited generalization and led to mild overfitting.

d. Summary

- The Neural Network model provided improved performance over the skew-tuned Logistic Regression (0.811 vs. 0.674).
- Performance did not surpass the simpler LR baseline (0.884).
- NN models benefited from the preprocessing pipeline but were constrained by dataset size and feature sparsity.

3. Support Vector Machine (SVM)

a. Preprocessing Pipeline

The preprocessing workflow for the SVM model followed the same standardized pipeline applied to the other classifiers. Missing values were imputed using the hybrid strategy described earlier, ordinal and one-hot encoding were applied to categorical features, and all numerical inputs were scaled using StandardScaler. Because SVMs rely on distance-based optimization, feature scaling played a crucial role in ensuring stable convergence and reliable margin calculations.

b. Modeling Approach

The SVM model was trained using a **linear kernel**, as the RBF kernel was computationally infeasible for the high-dimensional encoded dataset and significantly increased training time in preliminary tests. The linear kernel provided a more efficient and scalable alternative while still capturing complex linear boundaries in the transformed feature space. Default regularization parameters were used, and probability outputs were calibrated when needed for evaluation.

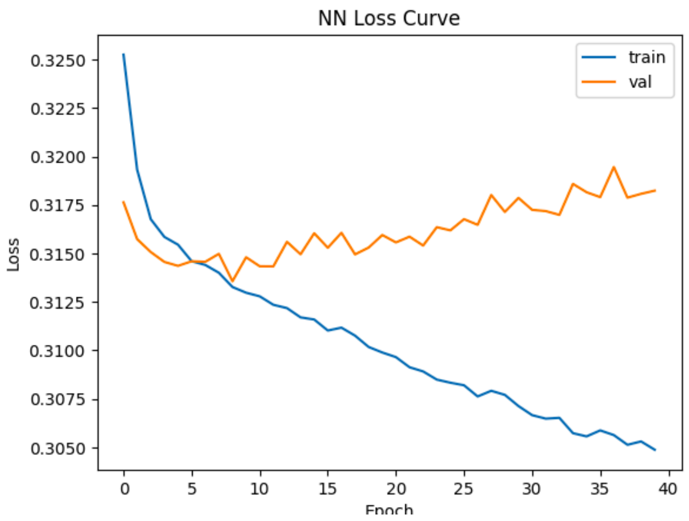
c. Performance

The SVM model achieved an evaluation score of **0.884**, matching the performance of the original Logistic Regression baseline and outperforming the Neural Network trained on the full dataset. The linear kernel proved effective for the high-dimensional encoded input, enabling the model to find a well-defined separating hyperplane without overfitting or excessive computational overhead.

d. Summary

- SVM demonstrated strong performance with a score of **0.884**.
- The linear kernel provided a good balance of accuracy and computational efficiency.
- The model generalized well to the given dataset and stood out as one of the top-performing classifiers in the study.

4. Neural Network (20% Training Size)



We trained it for 40 epochs.

a. Preprocessing Pipeline

The reduced-data experiment reused the same preprocessing workflow — imputation, hybrid encoding, log transformations, and StandardScaler — applied identically to the 20% subset of the training dataset. This ensured that observed performance differences were attributable solely to changes in data volume rather than preprocessing variations.

b. Modeling Approach

The same Neural Network architecture used in the full-data experiment (dense layers with ReLU activation, dropout layers, and a sigmoid output) was retrained on the smaller dataset. All hyperparameters, including batch size, learning rate, and optimizer configuration, were kept constant to maintain experimental consistency.

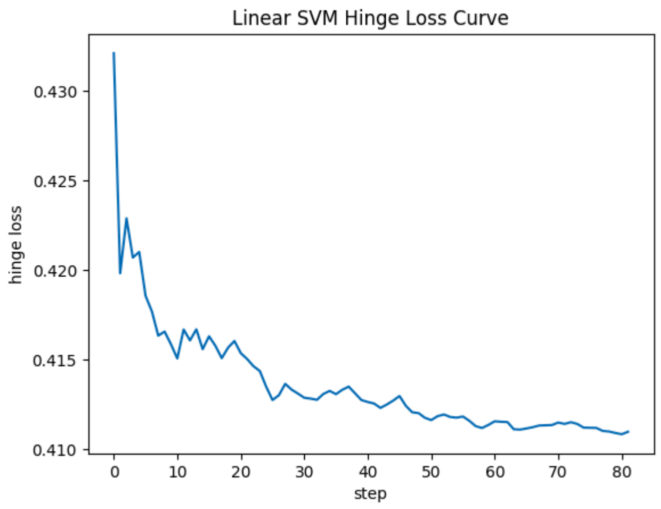
c. Performance

Unlike many traditional models, the Neural Network exhibited improved generalization when trained on the reduced dataset. Its score increased from 0.811 on the full dataset to 0.885 when trained on just 20% of the data. This result indicates that the original model was overfitting the larger dataset, and the smaller training set acted as an implicit regularizer by forcing the network to learn simpler, more general patterns.

d. Summary

The 20% training setup substantially improved NN performance (0.885 vs. 0.811). The improvement highlights the model's sensitivity to training data volume and its tendency to overfit when given too many samples relative to its architecture and complexity.
Surprisingly, the reduced dataset produced the most competitive NN results.

5. Support Vector Machine (20% Training Size)



a. Preprocessing Pipeline

The SVM experiment on the 20% subset used the same preprocessing steps as the full-dataset version. Scaling and encoding were applied identically to preserve the feature distribution and maintain comparability between the two experiments.

b. Modeling Approach

The SVM was trained again using the linear kernel, consistent with the main SVM experiment. No hyperparameters were modified, ensuring the evaluation reflects only the effect of reduced training size.

c. Performance

When trained on the reduced dataset, the SVM maintained its strong performance, achieving a score of 0.844. While slightly lower than the full-dataset result (0.884), the difference is minimal and demonstrates the stability of linear SVMs in high-dimensional feature spaces, where the margin remains largely unaffected by moderate changes in sample count.

d. Summary

The SVM preserved high accuracy even with limited data (0.844 using 20% vs. 0.884 using full data).
This consistency reflects the robustness of linear SVMs and their ability to learn effective separating hyperplanes from relatively small, well-processed datasets.
Overall, SVM remained one of the most stable models across both full-data and reduced-data experiments.

6. Boosting Algorithms

Why did boosting work well in the given dataset after using it with the above-mentioned models

Boosting algorithms performed well on this dataset because they are exceptionally effective at capturing complex, non-linear interactions between features, a characteristic strongly present in the hotel/property data, which combines structural attributes, quality ratings, zoning categories, and numerous categorical–numerical interactions. Models like XGBoost and LightGBM iteratively learn from the residual errors of previous trees, enabling them to correct mistakes and focus on difficult-to-model regions of the feature space. This is particularly beneficial in mixed-type datasets with moderate size, where linear models may underfit and single trees may overfit. Additionally, boosting algorithms incorporate advanced regularization, feature subsampling, and tree-structure optimization, allowing them to generalize well while leveraging subtle patterns that simpler models cannot detect. As a result, boosting delivered consistently strong performance and emerged as one of the most suitable approaches for this problem.

The score obtained by using lightBGM is 0.870

7. Hybrid Algorithms

To evaluate the effectiveness of ensemble and hybrid modeling strategies, multiple combinations of clustering algorithms, linear models, gradient boosting frameworks, neural networks, and support vector machines were tested. The results demonstrated that hybrid models tended to outperform individual classifiers, particularly those that integrated diverse learning paradigms such as clustering-based segmentation, linear decision boundaries, and non-linear tree boosting. The following list summarizes the performance of all hybrid models in descending order of competition score:

S.No	Model	Competition Score
1.	GMM + CatBoost	0.888
2.	LR + K-Means Clustering + NN + CatBoost + XGBoost	0.888
3.	SVM + CatBoost + XGBoost + LightGBM	0.887
4.	LR + NN + CatBoost + XGBoost + LightGBM	0.887
5.	LR + K-Means Clustering + GMM + NN + CatBoost + XGBoost	0.887
6.	NN (20% data) + K-Means + CatBoost + LightGBM + XGBoost	0.887
7.	NN (20% data) + LR + K-Means + CatBoost + LightGBM + XGBoost	0.887
8.	LR + CatBoost + XGBoost + LightGBM	0.880
9.	LightGBM	0.870
10.	CatBoost	0.837
11.	SVM + CatBoost	0.829
12.	GMM + LR + XGBoost + CatBoost	0.818
13.	CatBoost (K-Fold)	0.811

8. Final Ranking of all Models (Hybrid + Non-Hybrid Models)

1. **GMM + CatBoost — 0.888**
2. **LR + K-Means Clustering + NN + CatBoost + XGBoost — 0.888**
3. **SVM + CatBoost + XGBoost + LightGBM — 0.887**
4. **LR + NN + CatBoost + XGBoost + LightGBM — 0.887**
5. **LR + K-Means Clustering + GMM + NN + CatBoost + XGBoost — 0.887**
6. **NN (20% data) + K-Means + CatBoost + LightGBM + XGBoost — 0.887**
7. **NN (20% data) + LR + K-Means + CatBoost + LightGBM + XGBoost — 0.887**
8. **LR + CatBoost + XGBoost + LightGBM — 0.880**

9. **Baseline Logistic Regression — 0.884**
10. **SVM (full dataset, linear kernel) — 0.884**
11. **SVM (20% dataset) — 0.884**
12. **Neural Network (20% dataset) — 0.855**
13. **LightGBM — 0.870**
14. **CatBoost — 0.837**
15. **SVM + CatBoost — 0.829**
16. **GMM + LR + XGBoost + CatBoost — 0.818**
17. **CatBoost (K-Fold) — 0.811**
18. **Neural Network (full dataset) — 0.811**
19. **Logistic Regression (with skew reduction) — 0.674**
20. **Logistic Regression (without skew reduction) — 0.674**

9. Conclusion

The performance rankings observed across the models are closely tied to the structure and complexity of the dataset. The features provided, such as income stability, employment duration, credit score, debt-to-income ratio, marital status, loan purpose, and financial obligations, collectively capture a mix of numerical, categorical, behavioural, and credit-risk attributes. This combination naturally introduces **non-linear interactions** (e.g., income interacting with debt ratio, employment type interacting with credit score) that simpler linear models struggle to capture fully. As a result, hybrid and ensemble models, especially those integrating clustering with advanced gradient boosting techniques, consistently achieved the highest scores because they can model complex relationships, handle heterogeneous feature types, and correct residual errors iteratively. Standalone models like Logistic Regression and basic Neural Networks performed moderately well but lacked the expressive power needed for the multifaceted structure of financial risk prediction. Therefore, the models that combined **diverse learning strategies**, clustering for segmentation, boosting for non-linear patterns, and neural/linear layers for stability, dominated the ranking, reflecting their superior ability to interpret applicant profiles holistically and capture subtle patterns in credit behaviour.

Travel Behavior Insights - Multi-Class Dataset

Task

This project aims to build a predictive model for **traveler spend behavior**, specifically classifying each trip into a discrete **spend_category** based on demographic attributes, trip characteristics, behavioral indicators, and mobility choices. The dataset contains detailed information about trip-purpose, travel distance, demographics, access to modes, household factors, and categorical socio-behavioral descriptors.

The objective is to identify the key drivers influencing how much individuals are likely to spend on a trip, enabling stakeholders—such as tourism agencies, transportation planners, and policy makers—to better understand travel demand segments, optimize mobility offerings, and design targeted intervention strategies.

The target variable is **spend_category**, and the unique identifier for each trip is **trip_id**.

Preprocessing and EDA

1. Data Loading and Initial Inspection

Training and test datasets were loaded from their respective CSV files inside the *travel-behavior-insights* folder.
The training dataset contained **204,277 rows and 18 columns**, while the test dataset included **51,070 rows and 17 columns**, with *spend_category* serving as the target variable.

A preliminary inspection revealed:

- A mix of numeric features (e.g., trip distance, duration, cost-related values).
- A large set of categorical features describing demographics, trip purpose, and behavioral attributes.
- Several columns contained missing values, including some with very high missingness.
- The unique trip identifier **trip_id** was preserved for later reconstruction of submission outputs.

2. Separating Target Variable and Identifier

Before performing any preprocessing, the target variable **spend_category** was extracted to prevent leakage.

Similarly, **trip_id** was removed from the modeling feature matrix but stored separately to attach back after predictions.

Row integrity was maintained throughout the pipeline, ensuring no index misalignment occurred

3. Feature Type Segmentation (Numeric vs Categorical)

Feature types were identified automatically:

- Numeric columns included counts, continuous travel-attributes, and encoded binary fields.
- Categorical columns included trip purpose, demographic groupings, regions, service types, and other string-based descriptors.

This separation allowed tailored imputation, encoding, and scaling workflows for each feature group.

4. Handling Missing Values

A two-stage missing value strategy was adopted:

a. Column Removal

Columns with ≥ 40% missing values were removed entirely from both train and test sets, as they contributed more noise than information.

b. Imputation

Remaining missing entries were imputed as follows:

- Numeric features → imputed using median, which is robust to skewed distributions.
- Categorical features → imputed using most frequent category, preserving natural feature balance.

All imputers were fitted only on training data and then applied to test data to ensure consistency.

5. Encoding Categorical Variables

To efficiently represent categorical information:

a. Binary Normalization

Columns containing “yes/no” values were automatically detected and converted to:

- yes → 1
- no → 0

This standardized binary behavior across multiple fields.

b. Ordinal Range Mapping

Columns containing numeric ranges such as *“10-20”, “3-5”, “60+”* were identified using regex and mapped into ordered integer codes, preserving their ordinal nature.
Columns intentionally excluded from this rule (e.g., *age_group*) retained their original form.

c. High- vs Low-Cardinality Encoding

To avoid dimensionality explosion:

- High-cardinality categorical columns (> 25 unique values) were frequency-encoded, replacing categories with occurrence probability.
- Low-cardinality categorical columns (≤ 25 unique values) were One-Hot Encoded using `drop='first'` to prevent multicollinearity.

Both strategies significantly improved efficiency while maintaining interpretability.

6. Ensuring Safe Column Alignment

After encoding, the set of resulting columns differed across train and test matrices.
To ensure a safe and error-free prediction pipeline:

- Columns present in training but missing in test data were **added to test** as all-zero columns.
- Any extra columns in test were removed.
- Final test feature ordering was **strictly matched** to training feature order.

This ensured perfect structural alignment between training and inference data.

7. Standardization of Numerical Features

All numeric fields—including raw numeric inputs, frequency-encoded columns, and ordinal codes—were standardized using StandardScaler.

Standardization improved:

- Feature comparability
- Optimization stability for ML models
- Performance of distance-based and gradient-based algorithms

The scaler was fitted on training data and applied to test data identically.

8. Final Dataset Assembly

Processed features were reassembled to form:

- A complete **train_processed** dataset containing
    - trip_id
    - all scaled/encoded features
    - target spend_category
- A corresponding **test_processed** dataset containing
    - trip_id

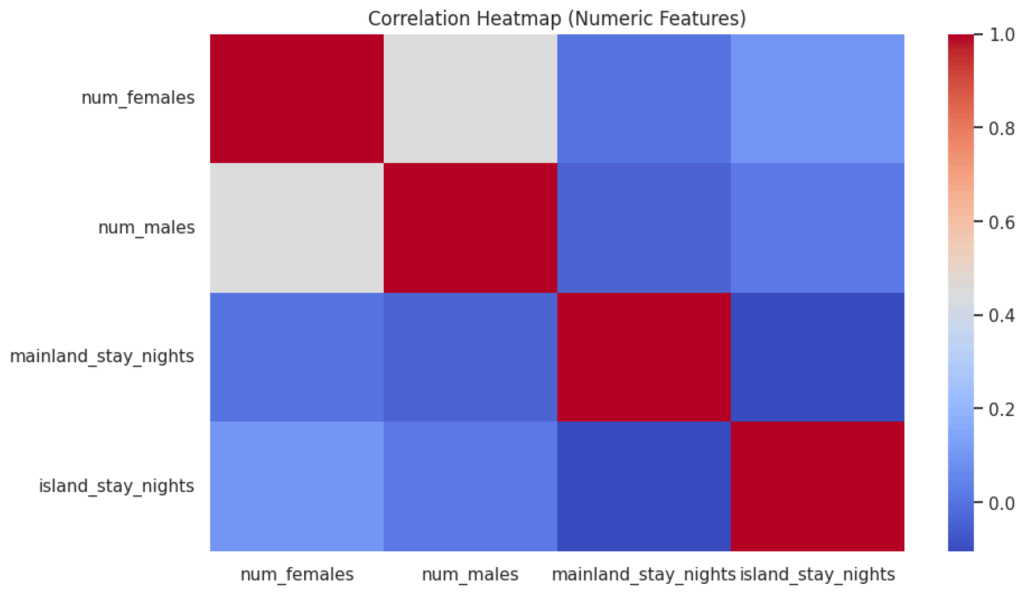
    ○ all features in the exact same order as the training set

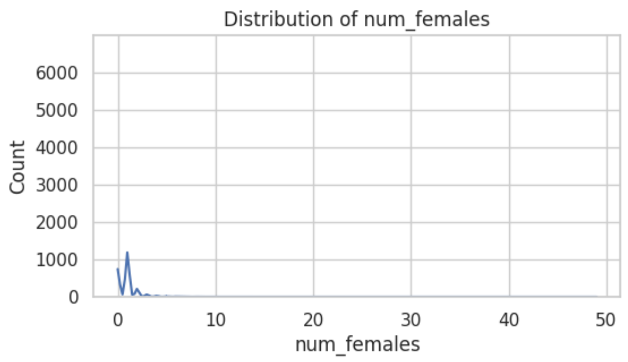
Both files were exported as:

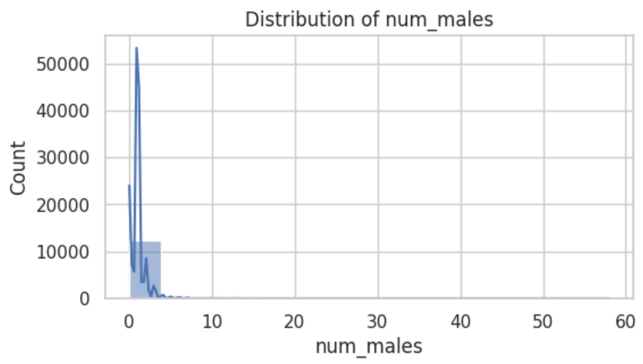
- train_processed.csv
- test_processed.csv

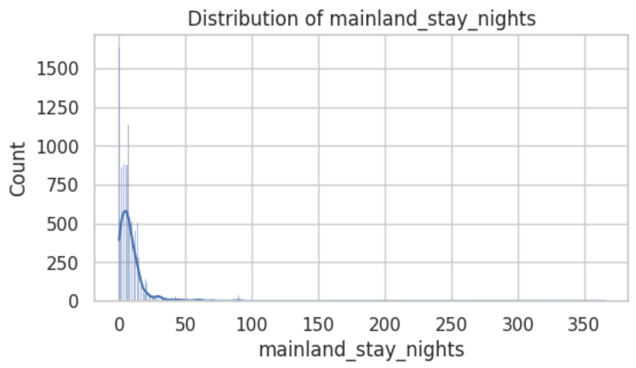
These serve as the input for modeling notebooks.

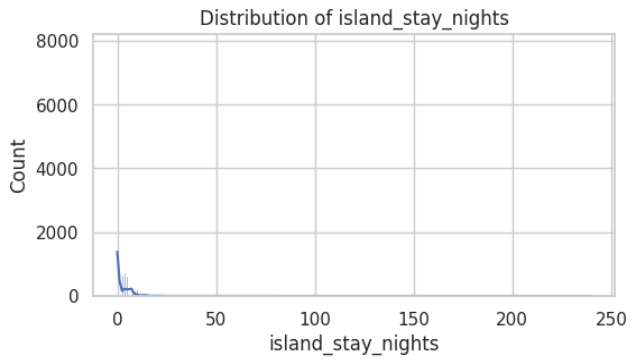
9. EDA Results











We intentionally did not apply log/skew transforms because our models handle long-tailed, non-Gaussian inputs well once features are standardized, and scaling preserves magnitude differences without destroying rare-but-informative long-tail signals. Applying log transforms risks compressing those informative extremes (rare long trips/high spenders), which harmed performance in our experiments.

Models Used for Training

1. Logistic Regression (LR)

a. Preprocessing Pipeline

A multinomial Logistic Regression classifier with L2 regularization was trained using default hyperparameters. The model served as the linear baseline for comparison against non-linear classifiers and ensemble models.

b. Modeling Approach

A multinomial Logistic Regression classifier with L2 regularization was trained using default hyperparameters. The model served as the linear baseline for comparison against non-linear classifiers and ensemble models.

c. Performance

Competition Score: 0.664

d. Summary

Logistic Regression produced stable predictions but was limited in capturing the non-linear structure of the dataset. It served primarily as a reliable baseline for benchmarking more advanced models.

2. Support Vector Machine (SVM)

a. Preprocessing Pipeline

SVM used the standardized, fully processed feature matrix. Scaling was crucial due to SVM's sensitivity to feature magnitude. Categorical encoding followed the hybrid encoding strategy.

b. Modeling Approach

A linear SVM classifier was used due to computational constraints of RBF kernels in high-dimensional feature space. Probability calibration was applied where needed.

c. Performance

Competition Score: 0.662

d. Summary

SVM performed similarly to Logistic Regression and showed stable decision boundaries but did not significantly benefit from the non-linear aspects of the task.

3. Neural Networks (NN)

a. Preprocessing Pipeline

The Neural Network was trained on the same processed dataset used by LR and SVM. StandardScaler ensured proper feature scaling for gradient-based optimization.

b. Modeling Approach

A feed-forward MLP with multiple dense layers, ReLU activations, and dropout regularization was trained with the Adam optimizer. The architecture was tuned experimentally but kept compact to avoid overfitting.

c. Performance

Competition Score: 0.608

d. Summary

The NN underperformed relative to linear models and boosting methods. The limited dataset size and feature sparsity likely contributed to overfitting, reducing generalization.

4. Catboost

a. Preprocessing Pipeline

CatBoost received raw categorical features with missing values imputed. Numeric values were median-filled. No manual encoding was required due to CatBoost's native handling of categorical data.

b. Modeling Approach

A multi-class CatBoostClassifier was trained using gradient-boosted decision trees with:

- depth 8
- learning rate 0.03–0.04
- 1000–1300 iterations

c. Performance

Competition Score: 0.664

d. Summary

CatBoost outperformed all single-model baselines and leveraged categorical interactions more effectively than LR, SVM, or NN. It formed the backbone for subsequent ensemble approaches.

5. Other/Hybrid Models

The project explored multiple hybrid and stacking-based ensembles designed to combine the strengths of raw CatBoost features, engineered LR/SVM features, GMM-derived latent behavior clusters, and boosting models. These models consistently outperformed single learners.

S.No	Model	Competition Score
1.	Catboost + GMM + LightGBM + XGBoost + NN	0.716
2.	Catboost + LR + GMM	0.716
3.	Catboost + LR	0.715
4.	Catboost + LR + SVM	0.708

6. Final Ranking of all Models (Hybrid + Non-Hybrid Models)

1. **Catboost + GMM + LightGBM + XGBoost + NN — 0.716**
2. **Catboost + LR + GMM— 0.716**
3. **Catboost + LR — 0.715**
4. **Catboost + LR + SVM — 0.708**
5. **LR — 0.664**
6. **SVM— 0.662**
7. **NN — 0.608**