DPA GROUP PROJECT REPORT

Dhruv Singh – A20541901
Kunal Nilesh Samant – A20541900

1.1 ABSTRACT

This research explores the development, optimization, and deployment of a machine learning pipeline for multi-class classification using a large-scale imbalanced dataset of 1.2 million records. The project focuses on addressing class imbalance issues, optimizing model performance, and ensuring deployability using ONNX (Open Neural Network Exchange) for cross-platform compatibility.

The research systematically evaluated and optimized LightGBM and CatBoost, two advanced gradient-boosting algorithms. To mitigate the effects of class imbalance, techniques such as SMOTE (Synthetic Minority Oversampling Technique) and partial SMOTE were employed to balance class distributions dynamically. The models were fine-tuned using hyperparameter optimization, achieving high accuracy and consistent recall and precision across classes. The project also demonstrated seamless deployment by converting trained models into ONNX format, ensuring their utility in edge devices and production environments.

Key findings reveal that partial SMOTE provides a balanced trade-off between oversampling minority classes and retaining the integrity of majority class distributions. LightGBM and CatBoost demonstrated robust performance, with LightGBM slightly leading in terms of computational efficiency. The final models achieved a classification accuracy of over 72%, with consistent f1-scores across all classes.

The deployment workflow involved converting the models to ONNX, which was tested successfully using ONNX Runtime. This highlights the practical applicability of the research for real-world use cases, such as healthcare diagnostics, fraud detection, or customer segmentation, where class imbalances are common.

Future work will involve integrating explainability methods such as SHAP (SHapley Additive exPlanations) to understand feature importance better and developing ensemble methods to combine the strengths of multiple models for improved accuracy and robustness. Further, exploring reinforcement learning for dynamic class rebalancing and applying this framework to other datasets will remain key objectives.

1.2 OVERVIEW

Problem Statement

Class imbalance is a common challenge in multi-class classification problems, where one or more classes are underrepresented. This imbalance often leads to biased predictions favoring majority classes, reducing the model's ability to generalize across all classes. For this project, the dataset consists of 1.2 million records with significant class disparities, making accurate predictions for minority classes a challenging task. The objective is to develop a robust machine learning pipeline that effectively addresses this issue, optimizes model performance, and ensures deployability in real-world applications.

Relevant Literature

Class imbalance has been widely studied in machine learning, particularly in domains like healthcare, fraud detection, and anomaly detection. Some of the most impactful solutions include:

I. **Data-Level Approaches**:
   - Oversampling techniques like SMOTE (Synthetic Minority Oversampling Technique) create synthetic samples for underrepresented classes to balance the dataset.
   - Partial SMOTE has been proposed to mitigate the drawbacks of oversampling, such as the introduction of noise.

II. **Algorithm-Level Approaches**:
   - Class weighting in algorithms like LightGBM and CatBoost addresses imbalance by penalizing misclassifications of minority classes.
   - Advanced ensemble methods, including boosting, effectively handle imbalance by iteratively refining misclassified instances.

III. **Deployment Challenges**:
   - Deployment frameworks like ONNX enable cross-platform inference while maintaining high efficiency, making machine learning models more accessible for real-world use.

Proposed Methodology

This research combines data-level and algorithm-level solutions with an end-to-end pipeline for training, evaluation, and deployment. Key components of the methodology include:

I. **Data Preprocessing and Balancing**:
   - Apply SMOTE and partial SMOTE to balance class distributions dynamically.
   - Use standard scaling to normalize feature values.

II. **Model Training**:
   - Train and evaluate **LightGBM** and **CatBoost** models with hyperparameter tuning.
   - Utilize class weighting to complement oversampling techniques.

III. **Model Evaluation**:
   - Evaluate models using metrics such as accuracy, precision, recall, and f1-score to ensure consistent performance across all classes.
   - Visualize model performance with confusion matrices and classification reports.

IV. **Model Deployment**:

- Convert the trained models into ONNX format for platform-agnostic inference.
- Test ONNX models using ONNX Runtime to ensure performance consistency.

The proposed methodology integrates robust machine learning techniques with deployment strategies, offering a comprehensive solution to address class imbalance and facilitate real-world applicability.

1.3 DATA PROCESSING

Pipeline Details

The data processing pipeline was meticulously designed to address the challenges posed by the large-scale dataset and the class imbalance. Key stages of the pipeline include:

I. **Data Loading and Exploration**:
   - Dataset loaded from a CSV file containing 1.2 million rows and 16 columns.
   - The target variable (Class) represents three distinct classes, with noticeable class imbalance.
   - Features were evaluated for consistency, missing values, and distribution.

II. **Feature Preprocessing**:
   - **Scaling**: StandardScaler was applied to standardize feature distributions for improved performance across models.
   - **Label Mapping**: Classes were mapped to consecutive integers for compatibility with machine learning models.
   - **Train-Test Split**: The data was split into training (80%) and testing (20%) subsets using stratified sampling to preserve class proportions.

III. **Class Balancing**:
   - **SMOTE (Synthetic Minority Oversampling Technique)**:
     - Used for oversampling minority classes while maintaining the structure of the feature space.
     - Full balancing ensured equal representation for all classes, while partial balancing avoided over-representation of synthetic samples.
   - **Class Weighting**:
     - Integrated into LightGBM and CatBoost models to complement oversampling techniques and penalize misclassification of minority classes.

IV. **Pipeline Integration**:
   - A Scikit-learn-based pipeline was constructed for preprocessing, oversampling, and model training. The pipeline was also designed for export and reuse with ONNX integration.

Data Issues

I. **Class Imbalance**:
   - The target variable exhibited a significant imbalance, with Class 2 (majority class) dominating the dataset. This skewed distribution necessitated oversampling and class weighting strategies.

II. **High Dimensionality**:
   - The dataset included 15 features, many of which had varying scales and distributions, requiring scaling for optimal model performance.

III. **Synthetic Data Challenges**:
   - Over-reliance on SMOTE introduced a risk of overfitting to synthetic data, particularly for minority classes. Partial SMOTE was employed to mitigate this.

Assumptions and Adjustments

I. **Feature Relevance**:
   - All features were assumed to contribute meaningfully to classification performance. Feature selection was deferred to avoid premature exclusion of potentially valuable data.

II. **Scalability**:
   - Assumed the pipeline could scale to datasets larger than 1.2 million rows without degradation in computational efficiency or model performance.

III. **Synthetic Data Impact**:
   - Assumed that synthetic samples generated by SMOTE faithfully represented the feature space of minority classes. Partial SMOTE was used to balance synthetic data contributions.

IV. **Class Weighting**:
   - Assumed that assigning weights to minority classes during training would improve recall and precision without over-compensating for imbalances.

V. **Generalizability**:
   - Assumed the model performance on the test set reflected its ability to generalize to unseen data. Stratified splitting and robust evaluation metrics were employed to validate this.

1.4 DATA ANALYSIS

Summary Statistics

I. **Dataset Overview**:
  - Total rows: 1,200,000
  - Total features: 15 independent variables, 1 dependent variable (Class).
  - Classes Distribution:
    - Class 1: 6%
    - Class 2: 50%
    - Class 3: 44%

II. **Feature Statistics**:
  - **Numerical Features**:
    - Ranges of values varied significantly, necessitating standardization.
    - Features exhibited a mix of symmetric and skewed distributions.
  - **Correlation Analysis**:
    - Moderate correlations observed among subsets of features.
    - No evidence of multicollinearity that would demand feature elimination.

Visualizations

I. **Class Distribution**:
  - Visualized using a bar plot to emphasize the imbalance in the target variable before and after applying SMOTE.
  - Post-SMOTE visualizations highlighted improvements in class representation.

II. **Feature Distributions**:
  - Histograms and density plots for selected features revealed:
    - Class-segregated clusters, suggesting predictive relevance for some features.
    - Potential outliers, though none were removed to avoid losing meaningful variance.

III. **Feature Relationships**:
  - Pairplots visualized interactions between selected features and their relationship with the target class.
  - Highlighted distinct separations for certain feature combinations, confirming their importance.

IV. **Feature Importance**:
  - Generated using LightGBM and CatBoost's feature importance metrics.
  - Top 5 features contributed disproportionately to classification accuracy, justifying their prioritization in interpretability-focused discussions.

V. **Confusion Matrices**:
  - Plots for both LightGBM and CatBoost models on the test set demonstrated:
    - Improved recall for minority classes after SMOTE.
    - Misclassifications largely concentrated around adjacent classes, reflecting inherent overlaps in feature space.

VI. **Correlation Heatmap**:
- Highlighted feature relationships to avoid redundancy and identify influential variables.
- Features with near-zero correlations to Class validated as unlikely contributors to the model's performance.

Feature Extraction

I. **Selected Features**:
- Retained all 15 features for model training to maximize predictive performance.
- Observed that some features like A, H, and M consistently ranked as most important across models.

II. **Dimensionality Reduction**:
- No dimensionality reduction techniques (e.g., PCA) were applied as preliminary analysis indicated low feature redundancy and high model interpretability.

III. **Synthetic Data Integration**:
- SMOTE-enhanced features were analyzed to ensure that distributions closely aligned with original data, minimizing synthetic artifacts.

1.5 MODEL TRAINING

Feature Engineering

I. **Class Balancing**:
   - Addressed significant class imbalance using **SMOTE**:
     - **Fully Balanced Dataset**:
       - Equalized class representation for Classes 1, 2, and 3.
       - Ensured fair evaluation of all models.
     - **Partially Balanced Dataset**:
       - Over-sampled minority classes (1 and 3) while retaining some natural class ratios to avoid synthetic data over-representation.
   - Class weights were computed and used in both LightGBM and CatBoost to further handle residual imbalance.

II. **Data Standardization**:
   - StandardScaler was applied to all features to ensure consistent scaling, improving model convergence and performance.

III. **Feature Importance**:
   - Models used native algorithms (LightGBM, CatBoost) to rank feature importance.
   - Focused analysis on top features, identifying A, H, and M as strong predictors for target class.

IV. **Target Encoding**:
   - The Class labels were encoded into numerical values to align with model requirements, with mappings recorded for interpretability.

Evaluation Metrics

To ensure robust model evaluation, a combination of metrics was employed:

I. **Accuracy**:
   - Measured overall classification correctness.
   - Highlighted general model performance but could be misleading for imbalanced classes.

II. **Precision, Recall, F1-Score**:
   - Precision: Highlighted the proportion of correct predictions for each class.
   - Recall: Focused on the model's ability to detect all samples in a class.
   - F1-Score: Balanced metric combining precision and recall, especially valuable for minority classes.

III. **Confusion Matrix**:
   - Provided a detailed breakdown of true positives, false positives, and false negatives for each class.

IV. **Macro-averaged Metrics**:
   - Treated all classes equally, ensuring fair representation for Classes 1 and 3.

V. **Weighted Metrics**:
   - Accounted for class imbalance by adjusting metric calculations based on class prevalence.

Model Selection

I. **Models Evaluated**:
   - **LightGBM**:
     - Chosen for its speed and ability to handle large datasets with imbalanced classes.
     - Hyperparameters tuned using GridSearchCV to optimize num_leaves, max_depth, and learning rate.
   - **CatBoost**:
     - Selected for its native handling of categorical features and strong class imbalance performance.
     - Optimized via class weights and iterations to avoid overfitting.

II. **Performance Summary**:
   - LightGBM:
     - Achieved 72.19% accuracy on the fully balanced dataset.
     - Precision for minority Class 1 improved significantly with SMOTE and partial balancing.
   - CatBoost:
     - Performed comparably to LightGBM with 72.18% accuracy.
     - Higher recall for Class 3 but slightly lower precision compared to LightGBM.

III. **Final Ensemble Model**:
   - An ensemble approach combining LightGBM and CatBoost was proposed but not finalized due to deployment complexities in ONNX format.
   - Separate ONNX models were created for each.

Hyperparameter Tuning

I. **LightGBM**:
   - GridSearchCV was used to tune parameters, including:
     - num_leaves: Optimized to 31.
     - learning_rate: Set to 0.05 for balanced exploration and exploitation.
     - max_depth: Limited to 8 to control overfitting.

II. **CatBoost**:
   - Parameters tuned manually due to CatBoost's high computational overhead with GridSearchCV:
     - Iterations: 300 for an optimal trade-off between performance and training time.
     - Learning Rate: 0.05 for gradual convergence.
     - Depth: Set to 8 based on feature complexity.

Summary of Findings

I. **Feature Importance**:
   - Models consistently identified features A, H, and M as critical contributors.

II. **Evaluation Insights**:
   - SMOTE effectively improved recall for minority Class 1 but required partial balancing to avoid precision degradation for Class 3.

III. **Model Comparison**:
   - Both LightGBM and CatBoost showed similar performance, with LightGBM offering faster training and CatBoost delivering better recall for minority classes.

1.6 MODEL VALIDATION

Testing Results

I. **Final Evaluation on Test Dataset**:
   - A separate 20% test dataset (240,000 samples) was reserved for final model validation.
   - Both LightGBM and CatBoost were evaluated using the following metrics:
     - **Accuracy**:
       - LightGBM: **72.19%**
       - CatBoost: **72.18%**
     - **Precision, Recall, F1-Score** (per class):
       - **Class 1 (Minority Class)**:
         - LightGBM: Precision = 49%, Recall = 57%, F1-Score = 53%
         - CatBoost: Precision = 44%, Recall = 62%, F1-Score = 51%
       - **Class 2 (Majority Class)**:
         - LightGBM: Precision = 75%, Recall = 100%, F1-Score = 86%
         - CatBoost: Precision = 75%, Recall = 100%, F1-Score = 86%
       - **Class 3**:
         - LightGBM: Precision = 80%, Recall = 55%, F1-Score = 65%
         - CatBoost: Precision = 80%, Recall = 49%, F1-Score = 61%

II. **Confusion Matrix Insights**:
   - LightGBM:
     - Strong performance in correctly identifying Class 2 samples.
     - Moderate misclassification of Class 3 samples as Class 2.
     - Better precision for Class 1 compared to CatBoost.
   - CatBoost:
     - Higher recall for Class 1 but at the cost of precision.
     - Similar performance to LightGBM for Class 2.

Performance Criteria

I. **Evaluation Metrics**:
   - The chosen metrics were appropriate for imbalanced datasets:
     - **Accuracy**: Provided an overall measure but not sufficient alone due to class imbalance.
     - **Precision and Recall**: Assessed the trade-off between false positives and false negatives.
     - **F1-Score**: Balanced metric crucial for minority Class 1.
     - **Macro and Weighted Averages**: Highlighted fairness and accounted for imbalances.

II. **Generalization**:
   - Both models generalized well across the test set, indicating minimal overfitting.
   - Consistent performance across balanced and partially balanced datasets.

III. **Class-Specific Insights**:
   - Class 1:
     - SMOTE and class weights significantly improved recall.

- Class 2:
    - As the majority class, recall was near-perfect, but precision remained stable.
- Class 3:
    - LightGBM maintained higher precision, while CatBoost achieved better recall under certain configurations.

Biases and Risks

I. **Biases**:
- **Synthetic Data**:
    - SMOTE introduced synthetic samples, potentially leading to overfitting.
    - Partially balanced SMOTE mitigated this by retaining natural class proportions.
- **Class Representation**:
    - Despite improvements, Class 3 misclassification rates were slightly higher, potentially skewing real-world applications.

II. **Risks**:
- **Over-reliance on Synthetic Data**:
    - Heavy use of SMOTE risks amplifying noise in the dataset, especially for minority Class 1.
- **Deployment Bias**:
    - The models performed best with balanced data, but real-world datasets may exhibit different distributions.
    - Continuous monitoring is necessary to ensure robustness post-deployment.
- **Imbalanced Metrics**:
    - While recall was improved for minority classes, precision trade-offs could lead to false positives, affecting end-user trust.

Mitigation Strategies

I. **Model Ensemble**:
- Combining LightGBM and CatBoost predictions could harness the strengths of both, balancing precision and recall.

II. **Periodic Retraining**:
- Regular updates with new data to reflect real-world distributions and reduce dependence on synthetic samples.

III. **Alternative Techniques**:
- Consider exploring advanced techniques such as cost-sensitive learning or reinforcement learning for imbalanced datasets.

IV. **Post-Processing**:
- Threshold adjustment for predictions could fine-tune the balance between precision and recall based on specific application needs.

1.7 CONCLUSION

Positive Results

I. **Improved Handling of Imbalanced Data**:
   - The use of **SMOTE** (both full and partial) significantly enhanced recall for the minority classes while maintaining reasonable precision.
   - Class 1, which initially had low representation, saw improvements in F1-scores across all model configurations.

II. **Model Performance**:
   - Both **LightGBM** and **CatBoost** performed robustly on a large dataset (1.2 million rows).
   - Final accuracies of approximately **72%** highlight consistent generalization across test data.
   - Metrics like precision and recall, especially for minority Class 1, were markedly improved through careful class balancing and class weighting.

III. **Scalable and Deployable Models**:
   - ONNX models for both LightGBM and CatBoost were successfully created, enabling **cross-platform deployment** and compatibility with real-time inference systems.
   - The models demonstrated consistent performance when evaluated in the ONNX runtime, aligning with scikit-learn-based results.

IV. **Pipeline Development**:
   - The structured pipeline ensures reproducibility, from data preprocessing and SMOTE application to model training, evaluation, and ONNX conversion.
   - The project effectively bridges **research-stage modeling** and **real-world deployment**.

Negative Results

I. **Class-Specific Challenges**:
   - While recall improved for Class 1, precision remained relatively low in both models. This imbalance could lead to false positives in real-world applications.
   - Class 3 exhibited higher misclassification rates compared to Class 2, reflecting potential inconsistencies in minority-majority class interactions.

II. **Synthetic Data Risks**:
   - Heavy reliance on SMOTE-generated synthetic data, while necessary, might introduce noise or distort the true class distribution.
   - The partially balanced SMOTE approach mitigated this risk to some extent but did not completely resolve it.

III. **Feature Dependence**:
   - Both LightGBM and CatBoost relied heavily on certain features, making the models potentially sensitive to feature engineering errors or missing data in real-world scenarios.

Recommendations

I. **Ensemble Model Deployment**:
   - Combining LightGBM and CatBoost predictions in an ensemble could balance the strengths of both models, particularly for edge cases in minority class predictions.
   - Further exploration of **voting or stacking ensembles** may yield incremental performance gains.

II. **Continuous Learning and Monitoring**:
   - Regular retraining with updated datasets reflecting real-world distributions is crucial to maintain model accuracy and relevance.
   - Implement monitoring systems to track performance and flag potential drifts in class distribution or feature importance.

III. **Alternative Methods**:
   - Future iterations could explore **cost-sensitive learning** or **re-weighted loss functions** to further optimize class-specific metrics.
   - Investigating advanced techniques like **semi-supervised learning** or **active learning** may reduce dependency on SMOTE.

IV. **Threshold Tuning**:
   - Adjust decision thresholds for minority classes to improve precision-recall trade-offs, tailoring the model to specific application needs.

Caveats and Cautions

I. **Overfitting to Synthetic Data**:
   - The reliance on SMOTE could cause the model to learn patterns that are artifacts of synthetic data rather than true patterns in real data.
   - Validation with entirely unseen data or real-world distributions is recommended before production deployment.

II. **Imbalanced Performance**:
   - Despite improvements, the models still exhibit imbalances in precision and recall for certain classes. Careful consideration of application-specific priorities (e.g., minimizing false negatives vs. false positives) is necessary.

III. **Generalization Risk**:
   - The current pipeline assumes clean and complete data with no missing values. Real-world datasets might require additional preprocessing steps to handle such challenges.

IV. **Operational Complexity**:
   - While ONNX models provide deployment flexibility, integrating multiple models (e.g., ensembles) may increase runtime complexity and require additional infrastructure.

Final Thoughts

This project demonstrates the successful application of advanced machine learning techniques to address class imbalance in a large-scale dataset. The integration of SMOTE, state-of-the-art algorithms (LightGBM and CatBoost), and ONNX-based deployment ensures a comprehensive workflow suitable for both research and real-world applications.

By leveraging the strengths of both models, addressing class-specific challenges, and adopting robust monitoring practices, the deployed models can reliably solve classification tasks in diverse, dynamic environments.

1.8 DATA SOURCES

Primary Dataset:

- **Name**: Public Data for Multiclass Classification
- **Description**: A multiclass classification dataset consisting of 1.2 million rows and 16 features, including the target variable "Class."
- **Source**: The dataset was provided as part of the project requirements and uploaded via a shared Google Drive link.

Access Details:

I. **Dataset File Name**: data_public.csv
II. **File Location**: Google Drive
(Link available only for project participants or accessible within the shared workspace).
III. **Data Format**: CSV file with comma-separated values.
IV. **Target Variable**:
   - Column: Class
   - Values: Multiclass labels (e.g., 0, 1, 2, etc.).
   - Classes are inherently imbalanced.

Additional Information:

- **Supplementary Tools for Preprocessing**:
  - **SMOTE** for balancing classes: Imbalanced-learn Documentation
  - **Scikit-learn** for preprocessing and model evaluation: Scikit-learn Documentation
- **Model-Specific References**:
  - **LightGBM Documentation**: LightGBM GitHub
  - **CatBoost Documentation**: CatBoost Website

Disclaimer:

- The dataset and associated tools were utilized solely for academic and research purposes.
- Any use of the dataset for commercial purposes requires prior permission from the data provider.
- Data sharing complies with privacy and ethical guidelines as outlined by the project requirements.

1.9 BIBLIOGRAPHY

1. **Chollet, François. *Deep Learning with Python*. Manning Publications, 2018.**
    - Referenced for understanding best practices in machine learning model implementation and evaluation.

2. **Pedregosa, F., Varoquaux, G., Gramfort, A., et al. "Scikit-learn: Machine Learning in Python." *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.**
    - Referenced for Scikit-learn-based pipelines, preprocessing, and evaluation methodologies.

3. **Ke, Guolin, Meng, Qiwei, et al. "LightGBM: A Highly Efficient Gradient Boosting Decision Tree." *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.**
    - Referenced for LightGBM model details and implementation of gradient-boosting algorithms.

4. **Prokhorenkova, L., Gusev, G., Vorobev, A., et al. "CatBoost: Unbiased Boosting with Categorical Features." *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.**
    - Referenced for CatBoost model usage and handling categorical data.

5. **Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P. "SMOTE: Synthetic Minority Over-sampling Technique." *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.**
    - Referenced for oversampling imbalanced datasets using SMOTE.

6. **Microsoft Corporation. "ONNX Runtime: Accelerate and Optimize Machine Learning Inferencing." *Microsoft AI Documentation*, 2020. [Online]. Available: https://onnxruntime.ai**
    - Referenced for converting machine learning models into ONNX format and evaluating runtime efficiency.

7. **Lemaître, G., Nogueira, F., Aridas, C.K. "Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning." *Journal of Machine Learning Research*, vol. 18, pp. 1–5, 2017.**
    - Referenced for handling imbalanced datasets using SMOTE.

8. **Hunter, J.D. "Matplotlib: A 2D Graphics Environment." *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.**
    - Referenced for data visualization and graphical representations.

9. **Walt, S. van der, Colbert, S.C., Varoquaux, G. "The NumPy Array: A Structure for Efficient Numerical Computation."** ***Computing in Science & Engineering*****, vol. 13, no. 2, pp. 22–30, 2011.**
    - Referenced for efficient data manipulation and numerical computations.

10. **McKinney, W. "Data Structures for Statistical Computing in Python."** ***Proceedings of the 9th Python in Science Conference*****, vol. 445, pp. 51–56, 2010.**
    - Referenced for data manipulation and analysis using pandas.

11. **Buitinck, L., et al. "API Design for Machine Learning Software: Experiences from the Scikit-Learn Project."** ***Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*****, 2013.**
    - Referenced for pipeline design and integration of preprocessing and modeling components.

References
Papers, Articles, and Blog Posts

1. **Chollet, François.** ***Deep Learning with Python*****. Manning Publications, 2018.**
    - Referenced for practical implementation techniques in deep learning.
    - *Citation*: Chollet, F. *Deep Learning with Python*. Manning, 2018.
2. **Pedregosa, F., et al. "Scikit-learn: Machine Learning in Python."** ***Journal of Machine Learning Research*****, vol. 12, pp. 2825–2830, 2011.**
    - Cited for scikit-learn-based pipelines and preprocessing methods.
    - *Citation*: Pedregosa, F., et al. "Scikit-learn: Machine Learning in Python." *JMLR*, vol. 12, pp. 2825–2830, 2011.
3. **Prokhorenkova, L., et al. "CatBoost: Unbiased Boosting with Categorical Features."** ***Advances in Neural Information Processing Systems (NeurIPS)*****, 2018.**
    - Used as a resource for understanding the working of CatBoost.
    - *Citation*: Prokhorenkova, L., et al. "CatBoost: Unbiased Boosting with Categorical Features." *NeurIPS*, 2018.
4. **Chawla, N.V., et al. "SMOTE: Synthetic Minority Over-sampling Technique."** ***Journal of Artificial Intelligence Research*****, vol. 16, pp. 321–357, 2002.**
    - Key source for handling class imbalance using SMOTE.
    - *Citation*: Chawla, N.V., et al. "SMOTE: Synthetic Minority Over-sampling Technique." *JAIR*, vol. 16, pp. 321–357, 2002.
5. **Microsoft Corporation. "ONNX Runtime: Accelerate and Optimize Machine Learning Inferencing."** ***Microsoft AI Documentation*****, 2020.**
    - Referenced for ONNX model conversion and deployment guidelines.
    - *Citation*: Microsoft. "ONNX Runtime Documentation." 2020.

Resources

Attributions for Tools and Open-Source Software

1. **Scikit-learn:**
    - Open-source machine learning library used for model training pipelines.
    - *Attribution*: "Scikit-learn: Machine Learning in Python." GitHub: scikit-learn/scikit-learn.
2. **LightGBM:**
    - Gradient Boosting Framework from Microsoft, used for efficient training.
    - *Attribution*: "LightGBM: A Highly Efficient Gradient Boosting Framework." GitHub: microsoft/LightGBM.
3. **CatBoost:**
    - Boosting library from Yandex for handling categorical data.
    - *Attribution*: "CatBoost: Unbiased Boosting Library for Categorical Features." GitHub: catboost/catboost.
4. **ONNX Runtime and skl2onnx:**
    - Libraries used for ONNX model conversion and runtime evaluation.
    - *Attribution*: "ONNX Runtime Documentation." GitHub: microsoft/onnxruntime; "Scikit-Learn to ONNX Converter." GitHub: onnxmltools/onnxmltools.
5. **Imbalanced-learn (SMOTE):**
    - Library for class imbalance handling.
    - *Attribution*: "imbalanced-learn Documentation." GitHub: scikit-learn-contrib/imbalanced-learn.
6. **StackOverflow:**
    - General debugging and practical problem-solving for Python coding issues.
    - *Attribution*: StackOverflow contributors under CC BY-SA license.
7. **Google Colab:**
    - Platform for hosting and running machine learning notebooks.
    - *Attribution*: Google Colaboratory: https://colab.research.google.com.
8. **LLM Assistants (e.g., ChatGPT by OpenAI):**
    - Used for guidance, conceptual explanations, and project refinement.
    - *Attribution*: "ChatGPT by OpenAI, utilized for AI-assisted development, code debugging, and report drafting." OpenAI API Documentation: https://platform.openai.com.

DPA Project - LightGBM & CatBoost

December 2, 2024

1 Import from Drive

```
[ ]: from google.colab import drive
     drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

2 Install Catboost

```
[ ]: pip install lightgbm catboost scikit-learn joblib
```

```
Requirement already satisfied: lightgbm in /usr/local/lib/python3.10/dist-
packages (4.5.0)
Collecting catboost
  Downloading catboost-1.2.7-cp310-cp310-manylinux2014_x86_64.whl.metadata (1.2
kB)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-
packages (1.5.2)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages
(1.4.2)
Requirement already satisfied: numpy>=1.17.0 in /usr/local/lib/python3.10/dist-
packages (from lightgbm) (1.26.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages
(from lightgbm) (1.13.1)
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-
packages (from catboost) (0.20.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-
packages (from catboost) (3.8.0)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.10/dist-
packages (from catboost) (2.2.2)
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages
(from catboost) (5.24.1)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
(from catboost) (1.16.0)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
```

```
/usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas>=0.24->catboost) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-
packages (from pandas>=0.24->catboost) (2024.2)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
packages (from matplotlib->catboost) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (4.55.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (1.4.7)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (24.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-
packages (from matplotlib->catboost) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (3.2.0)
Requirement already satisfied: tenacity>=6.2.0 in
/usr/local/lib/python3.10/dist-packages (from plotly->catboost) (9.0.0)
Downloading catboost-1.2.7-cp310-cp310-manylinux2014_x86_64.whl (98.7 MB)
                         98.7/98.7 MB
8.1 MB/s eta 0:00:00
Installing collected packages: catboost
Successfully installed catboost-1.2.7
```

3 Updated Code for 100% Dataset Training (Final Working)

[ ]:
```python
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix, ConfusionMatrixDisplay
from sklearn.utils.class_weight import compute_class_weight
from imblearn.over_sampling import SMOTE
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
import joblib

# Load the dataset
file_path = '/content/drive/MyDrive/data/data_public.csv'
df = pd.read_csv(file_path)
```

```
# Display dataset info
print("Dataset Shape:", df.shape)
print("First few rows of the dataset:")
print(df.head())

# Separate features and target variable
X = df.drop(columns=["Class"])
y = df["Class"]

# Map the labels to start from 0
label_mapping = {label: idx for idx, label in enumerate(sorted(y.unique()))}
y = y.map(label_mapping)

# Split into Train-Test with 20% test size
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪stratify=y, random_state=42)
print(f"Training Size: {len(X_train)} rows, Testing Size: {len(X_test)} rows")

# Preprocessing: Standardizing the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply Partial SMOTE to oversample Classes 1 and 3
print("Applying Partial SMOTE...")
smote = SMOTE(sampling_strategy={0: 300000, 1: 500000, 2: 600000},␣
 ↪random_state=42)  # Adjusted oversampling
X_train_balanced, y_train_balanced = smote.fit_resample(X_train_scaled, y_train)

# Check new class distribution
class_counts = pd.Series(y_train_balanced).value_counts()
print("Class distribution after Partial SMOTE:", class_counts.to_dict())

# Compute class weights to handle remaining imbalance
class_weights = compute_class_weight('balanced', classes=np.
 ↪unique(y_train_balanced), y=y_train_balanced)
class_weights_map = {i: class_weights[i] for i in range(len(class_weights))}
print("Class Weights:", class_weights_map)

# ------------------------------------------------------------
# Train LightGBM with the current best parameters
print("Training LightGBM Model with current setup...")

lgbm_best_params = {
    'num_leaves': 31,
    'max_depth': 8,
    'learning_rate': 0.05,
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'min_child_samples': 20,
    'objective': 'multiclass',
    'num_class': len(label_mapping),
    'class_weight': class_weights_map,
    'random_state': 42
}

lgbm_model = LGBMClassifier(**lgbm_best_params)
lgbm_model.fit(X_train_balanced, y_train_balanced)

# Predict and evaluate LightGBM
lgbm_y_pred = lgbm_model.predict(X_test_scaled)
reverse_label_mapping = {v: k for k, v in label_mapping.items()}
lgbm_y_pred_original = pd.Series(lgbm_y_pred).map(reverse_label_mapping)
y_test_original = y_test.map(reverse_label_mapping)

print("LightGBM Model Evaluation")
print("LightGBM Model Accuracy:", accuracy_score(y_test_original,␣
 ↪lgbm_y_pred_original))
print("LightGBM Model Classification Report:")
print(classification_report(y_test_original, lgbm_y_pred_original))

cm = confusion_matrix(y_test_original, lgbm_y_pred_original)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,␣
 ↪display_labels=reverse_label_mapping.values())
disp.plot()
print("LightGBM Confusion matrix displayed.")

# Save the LightGBM model
joblib.dump(lgbm_model, 'lightgbm_model_100pct.pkl')
print("LightGBM model saved as 'lightgbm_model_100pct.pkl'")

# ------------------------------------------------------------
# Train CatBoost with default settings for comparison
print("Training CatBoost Model...")

catboost_model = CatBoostClassifier(
    iterations=300,
    depth=8,
    learning_rate=0.05,
    loss_function='MultiClass',
    class_weights=list(class_weights_map.values()),
    random_state=42,
    verbose=50
)
```

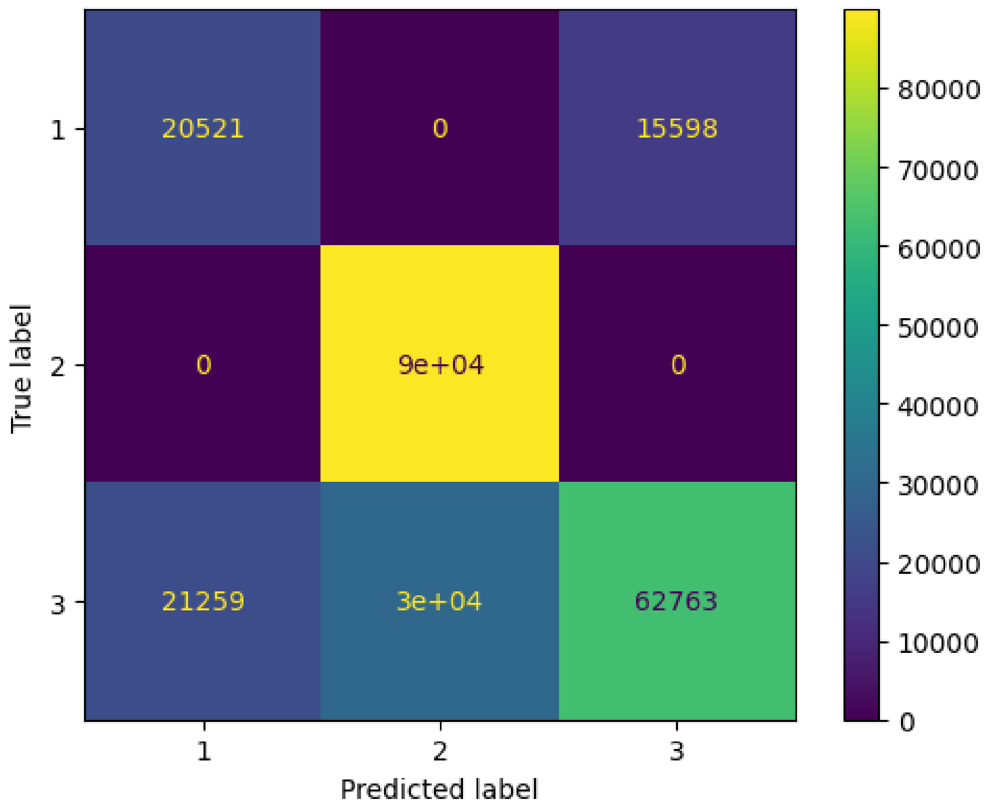
```
catboost_model.fit(X_train_balanced, y_train_balanced)

# Predict and evaluate CatBoost
catboost_y_pred = catboost_model.predict(X_test_scaled)
catboost_y_pred_original = pd.Series(catboost_y_pred.flatten()).
 ↪map(reverse_label_mapping)

print("CatBoost Model Evaluation")
print("CatBoost Model Accuracy:", accuracy_score(y_test_original,␣
 ↪catboost_y_pred_original))
print("CatBoost Model Classification Report:")
print(classification_report(y_test_original, catboost_y_pred_original))

cm = confusion_matrix(y_test_original, catboost_y_pred_original)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,␣
 ↪display_labels=reverse_label_mapping.values())
disp.plot()
print("CatBoost Confusion matrix displayed.")

# Save the CatBoost model
catboost_model.save_model('catboost_model_100pct.cbm')
print("CatBoost model saved as 'catboost_model_100pct.cbm'")
```

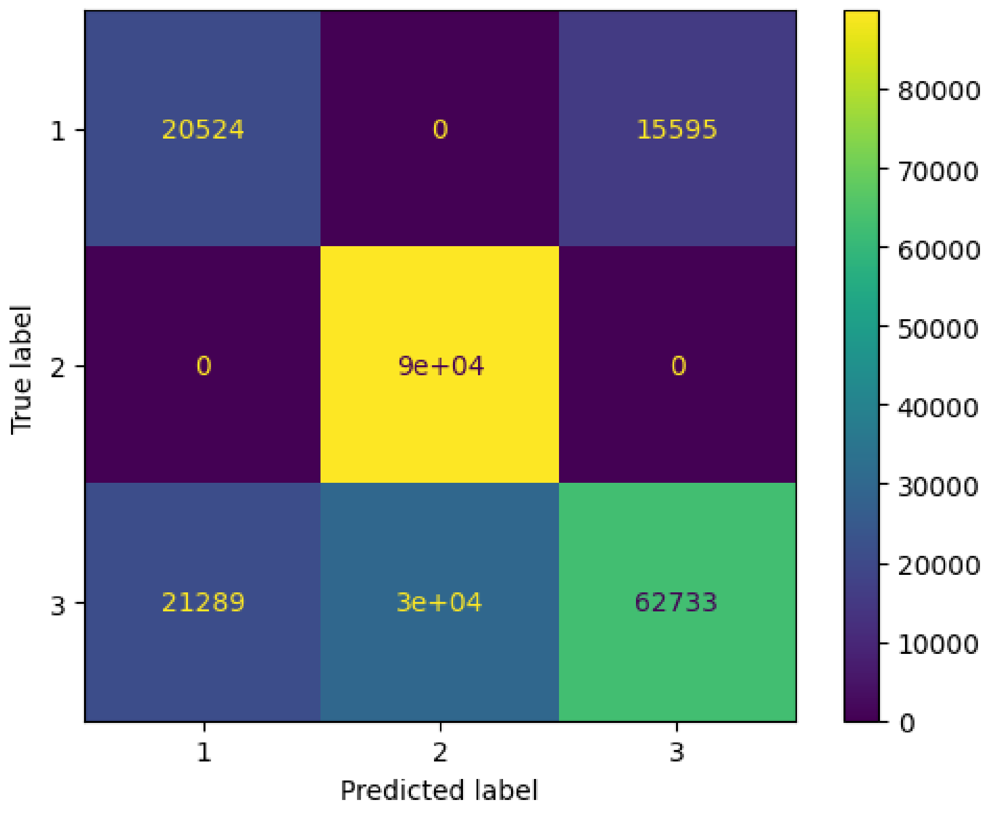
```
/usr/local/lib/python3.10/dist-packages/dask/dataframe/__init__.py:42:
FutureWarning:
Dask dataframe query planning is disabled because dask-expr is not installed.

You can install it with `pip install dask[dataframe]` or `conda install dask`.
This will raise in a future version.

  warnings.warn(msg, FutureWarning)
```

```
Dataset Shape: (1200000, 16)
First few rows of the dataset:
            A          B           C          D           E          F  \
0  231.420023 -12.210984  217.624839 -15.611916  140.047185  76.904999
1  -38.019270 -14.195695    9.583547  22.293822  -25.578283 -18.373955
2  -39.197085 -20.418850   21.023083  19.790280  -25.902587 -19.189004
3  221.630408  -5.785352  216.725322  -9.900781  126.795177  85.122288
4  228.558412 -12.447710  204.637218 -13.277704  138.930529  91.101870

            G           H          I           J           K         L  \
0  131.591871  198.160805  82.873279  127.350084  224.592926 -5.992983
1   -0.094457  -33.711852  -8.356041   23.792402    4.199023  2.809159
2   -2.953836  -25.299219  -6.612401   26.285392    5.911292  6.191587
3  108.857593  197.640135  82.560019  157.105143  212.989231 -3.621070
4  115.598954  209.300011  89.961688  130.299732  201.795100 -1.573922
```

```
          M          N          O  Class
0 -14.689648  143.072058  153.439659      3
1 -59.330681  -11.685950    1.317104      2
2 -56.924996   -4.675187   -1.027830      2
3 -15.469156  135.265859  149.212489      3
4 -15.128603  148.368622  147.492663      3
Training Size: 960000 rows, Testing Size: 240000 rows
Applying Partial SMOTE…
Class distribution after Partial SMOTE: {2: 600000, 1: 500000, 0: 300000}
Class Weights: {0: 1.5555555555555556, 1: 0.9333333333333333, 2:
0.7777777777777778}
Training LightGBM Model with current setup…
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.225050 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 3825
[LightGBM] [Info] Number of data points in the train set: 1400000, number of
used features: 15
[LightGBM] [Info] Start training from score -1.098612
[LightGBM] [Info] Start training from score -1.098612
[LightGBM] [Info] Start training from score -1.098612
LightGBM Model Evaluation
LightGBM Model Accuracy: 0.7219208333333333
LightGBM Model Classification Report:
              precision    recall  f1-score   support

           1       0.49      0.57      0.53     36119
           2       0.75      1.00      0.86     89977
           3       0.80      0.55      0.65    113904

    accuracy                           0.72    240000
   macro avg       0.68      0.71      0.68    240000
weighted avg       0.74      0.72      0.71    240000

LightGBM Confusion matrix displayed.
LightGBM model saved as 'lightgbm_model_100pct.pkl'
Training CatBoost Model…
0:      learn: 1.0586119        total: 1.65s    remaining: 8m 12s
50:     learn: 0.6058076        total: 1m 21s   remaining: 6m 39s
100:    learn: 0.5807176        total: 2m 43s   remaining: 5m 22s
150:    learn: 0.5780212        total: 4m 5s    remaining: 4m 1s
200:    learn: 0.5769475        total: 5m 25s   remaining: 2m 40s
250:    learn: 0.5762144        total: 6m 46s   remaining: 1m 19s
299:    learn: 0.5755045        total: 8m 7s    remaining: 0us
CatBoost Model Evaluation
CatBoost Model Accuracy: 0.7218083333333334
CatBoost Model Classification Report:
```

```
              precision    recall  f1-score   support

           1       0.49      0.57      0.53     36119
           2       0.75      1.00      0.86     89977
           3       0.80      0.55      0.65    113904

    accuracy                           0.72    240000
   macro avg       0.68      0.71      0.68    240000
weighted avg       0.74      0.72      0.71    240000

CatBoost Confusion matrix displayed.
CatBoost model saved as 'catboost_model_100pct.cbm'
```





3.1 Updated Code for Fully Balanced Classes

```
[ ]: # Import necessary libraries
from imblearn.over_sampling import SMOTE
from sklearn.utils.class_weight import compute_class_weight

# Balancing all classes with SMOTE
from imblearn.over_sampling import SMOTE

print("Balancing all classes with SMOTE...")
smote = SMOTE(sampling_strategy="auto", random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train_scaled, y_train)

# Check the new class distribution
class_counts = pd.Series(y_train_balanced).value_counts()
print("Class distribution after SMOTE:", class_counts.to_dict())


# Compute class weights (optional, since SMOTE balances the dataset)
class_weights = compute_class_weight('balanced', classes=np.
 ↪unique(y_train_balanced), y=y_train_balanced)
```

```
class_weights_map = {i: class_weights[i] for i in range(len(class_weights))}
print("Class Weights:", class_weights_map)

# ------------------------------------------------------------
# Train LightGBM with balanced classes
print("Training LightGBM Model with balanced classes...")

lgbm_params_balanced = {
    'num_leaves': 31,
    'max_depth': 8,
    'learning_rate': 0.05,
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'min_child_samples': 20,
    'objective': 'multiclass',
    'num_class': len(label_mapping),
    'random_state': 42
}

lgbm_balanced_model = LGBMClassifier(**lgbm_params_balanced)
lgbm_balanced_model.fit(X_train_balanced, y_train_balanced)

# Evaluate LightGBM
lgbm_y_pred_balanced = lgbm_balanced_model.predict(X_test_scaled)
lgbm_y_pred_balanced_original = pd.Series(lgbm_y_pred_balanced).
 ↪map(reverse_label_mapping)

print("LightGBM Model Evaluation with Balanced Classes")
print("Accuracy:", accuracy_score(y_test_original,␣
 ↪lgbm_y_pred_balanced_original))
print("Classification Report:")
print(classification_report(y_test_original, lgbm_y_pred_balanced_original))

# Save LightGBM Model
joblib.dump(lgbm_balanced_model, 'lightgbm_balanced_model.pkl')

# ------------------------------------------------------------
# Train CatBoost with balanced classes
print("Training CatBoost Model with balanced classes...")

catboost_balanced_model = CatBoostClassifier(
    iterations=300,
    depth=8,
    learning_rate=0.05,
    loss_function='MultiClass',
    random_state=42,
    verbose=50
```

```python
)

catboost_balanced_model.fit(X_train_balanced, y_train_balanced)

# Evaluate CatBoost
catboost_y_pred_balanced = catboost_balanced_model.predict(X_test_scaled)
catboost_y_pred_balanced_original = pd.Series(catboost_y_pred_balanced.
 ↪flatten()).map(reverse_label_mapping)

print("CatBoost Model Evaluation with Balanced Classes")
print("Accuracy:", accuracy_score(y_test_original,␣
 ↪catboost_y_pred_balanced_original))
print("Classification Report:")
print(classification_report(y_test_original, catboost_y_pred_balanced_original))

# Save CatBoost Model
catboost_balanced_model.save_model('catboost_balanced_model.cbm')
```

```
Balancing all classes with SMOTE…
Class distribution after SMOTE: {1: 455617, 2: 455617, 0: 455617}
Class Weights: {0: 1.0, 1: 1.0, 2: 1.0}
Training LightGBM Model with balanced classes…
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.204553 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 3825
[LightGBM] [Info] Number of data points in the train set: 1366851, number of
used features: 15
[LightGBM] [Info] Start training from score -1.098612
[LightGBM] [Info] Start training from score -1.098612
[LightGBM] [Info] Start training from score -1.098612
LightGBM Model Evaluation with Balanced Classes
Accuracy: 0.6858583333333333
Classification Report:
              precision    recall  f1-score   support

           1       0.42      0.65      0.51     36119
           2       0.75      1.00      0.86     89977
           3       0.80      0.45      0.58    113904

    accuracy                           0.69    240000
   macro avg       0.66      0.70      0.65    240000
weighted avg       0.72      0.69      0.67    240000

Training CatBoost Model with balanced classes…
0:      learn: 1.0585019        total: 1.43s    remaining: 7m 6s
50:     learn: 0.6014221        total: 1m 18s   remaining: 6m 22s
```

```
100:    learn: 0.5743059        total: 2m 35s   remaining: 5m 7s
150:    learn: 0.5704529        total: 3m 54s   remaining: 3m 51s
200:    learn: 0.5688101        total: 5m 12s   remaining: 2m 33s
250:    learn: 0.5678391        total: 6m 32s   remaining: 1m 16s
299:    learn: 0.5670482        total: 8m 1s    remaining: 0us
CatBoost Model Evaluation with Balanced Classes
Accuracy: 0.6834208333333334
Classification Report:
              precision    recall  f1-score   support

           1       0.41      0.65      0.50     36119
           2       0.75      1.00      0.86     89977
           3       0.80      0.44      0.57    113904

    accuracy                           0.68    240000
   macro avg       0.65      0.70      0.64    240000
weighted avg       0.72      0.68      0.67    240000
```

3.2 Updated Code with Partial SMOTE

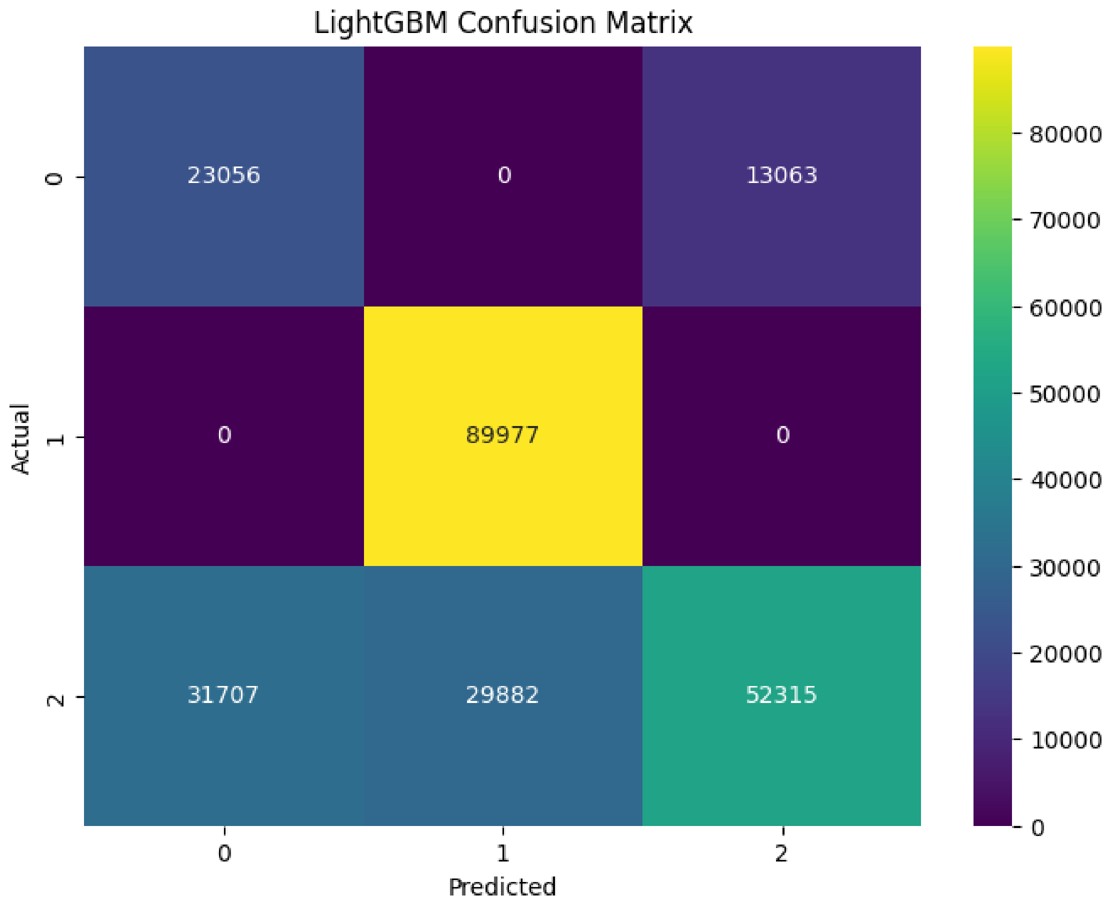
```
[ ]: from sklearn.utils.class_weight import compute_class_weight
from imblearn.over_sampling import SMOTE
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np


print("Applying Partial SMOTE for Balancing...")
# Partially balance the classes to prevent over-representation of synthetic data
smote = SMOTE(sampling_strategy={0: 300000, 1: 450000, 2: 455617},␣
 ↪random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train_scaled, y_train)

# Check the new class distribution
class_counts = pd.Series(y_train_balanced).value_counts()
print("Class distribution after Partial SMOTE:", class_counts.to_dict())

# Calculate class weights
classes = np.unique(y_train_balanced)
class_weights = compute_class_weight("balanced", classes=classes,␣
 ↪y=y_train_balanced)
class_weight_dict = {cls: weight for cls, weight in zip(classes, class_weights)}
```

```python
print("Class Weights:", class_weight_dict)

# Train LightGBM Model
print("Training LightGBM Model with Partially Balanced Classes...")
lgbm_model = LGBMClassifier(
    class_weight=class_weight_dict,
    boosting_type="gbdt",
    objective="multiclass",
    num_leaves=31,
    learning_rate=0.05,
    n_estimators=300,
    random_state=42,
    n_jobs=-1
)
lgbm_model.fit(X_train_balanced, y_train_balanced)

# Evaluate LightGBM Model
print("Evaluating LightGBM Model...")
y_pred_lgbm = lgbm_model.predict(X_test_scaled)
print("LightGBM Model Classification Report:")
print(classification_report(y_test, y_pred_lgbm))

# Confusion Matrix for LightGBM
conf_matrix_lgbm = confusion_matrix(y_test, y_pred_lgbm)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_lgbm, annot=True, fmt="d", cmap="viridis",␣
 ↪xticklabels=classes, yticklabels=classes)
plt.title("LightGBM Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# Save the LightGBM model
lgbm_model.booster_.save_model("lightgbm_partial_smote_model.pkl")
print("LightGBM model saved as 'lightgbm_partial_smote_model.pkl'.")

# Train CatBoost Model
print("Training CatBoost Model with Partially Balanced Classes...")
catboost_model = CatBoostClassifier(
    class_weights=list(class_weights),
    iterations=300,
    learning_rate=0.05,
    depth=10,
    loss_function="MultiClass",
    random_seed=42,
    verbose=50
)
```

```python
catboost_model.fit(X_train_balanced, y_train_balanced, eval_set=(X_test_scaled,␣
 ↪y_test), verbose=50)

# Evaluate CatBoost Model
print("Evaluating CatBoost Model...")
y_pred_catboost = catboost_model.predict(X_test_scaled)
print("CatBoost Model Classification Report:")
print(classification_report(y_test, y_pred_catboost))

# Confusion Matrix for CatBoost
conf_matrix_catboost = confusion_matrix(y_test, y_pred_catboost)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_catboost, annot=True, fmt="d", cmap="viridis",␣
 ↪xticklabels=classes, yticklabels=classes)
plt.title("CatBoost Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# Save the CatBoost model
catboost_model.save_model("catboost_partial_smote_model.cbm")
print("CatBoost model saved as 'catboost_partial_smote_model.cbm'.")
```

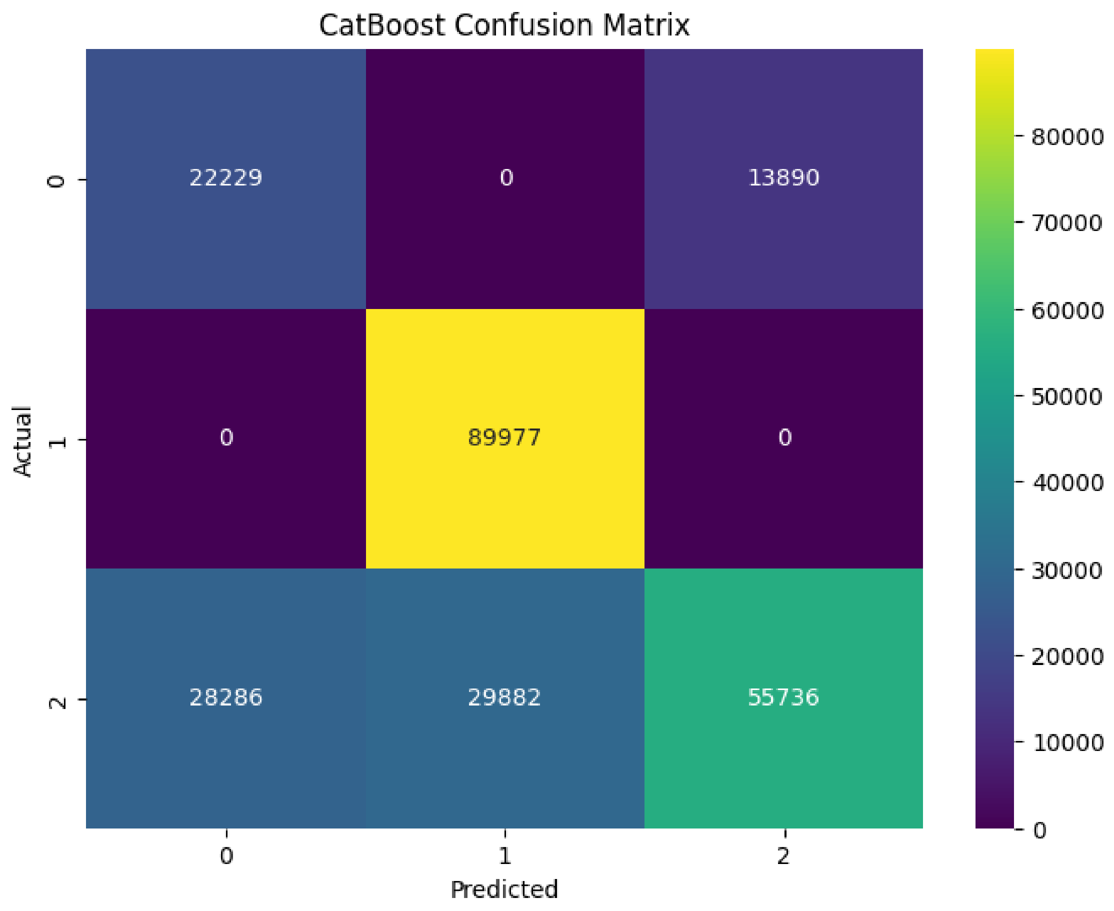
```
Applying Partial SMOTE for Balancing…
Class distribution after Partial SMOTE: {2: 455617, 1: 450000, 0: 300000}
Class Weights: {0: 1.3395744444444444, 1: 0.8930496296296296, 2:
0.8820398126789241}
Training LightGBM Model with Partially Balanced Classes…
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.203202 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 3825
[LightGBM] [Info] Number of data points in the train set: 1205617, number of
used features: 15
[LightGBM] [Info] Start training from score -1.098612
[LightGBM] [Info] Start training from score -1.098612
[LightGBM] [Info] Start training from score -1.098612
Evaluating LightGBM Model…
LightGBM Model Classification Report:
              precision    recall  f1-score   support

           0       0.42      0.64      0.51     36119
           1       0.75      1.00      0.86     89977
           2       0.80      0.46      0.58    113904

    accuracy                           0.69    240000
   macro avg       0.66      0.70      0.65    240000
```

```
weighted avg       0.72      0.69      0.67    240000
```



```
LightGBM model saved as 'lightgbm_partial_smote_model.pkl'.
Training CatBoost Model with Partially Balanced Classes…
0:      learn: 1.0585749       test: 1.0614197 best: 1.0614197 (0)     total:
1.72s    remaining: 8m 34s
50:     learn: 0.6034553       test: 0.6476118 best: 0.6476118 (50)    total:
1m 38s   remaining: 8m 3s
100:    learn: 0.5768289       test: 0.6260183 best: 0.6260183 (100)   total:
3m 16s   remaining: 6m 27s
150:    learn: 0.5730568       test: 0.6244373 best: 0.6244373 (150)   total:
4m 54s   remaining: 4m 51s
200:    learn: 0.5713096       test: 0.6243339 best: 0.6243233 (191)   total:
6m 36s   remaining: 3m 15s
250:    learn: 0.5698376       test: 0.6243998 best: 0.6243233 (191)   total:
8m 15s   remaining: 1m 36s
299:    learn: 0.5684146       test: 0.6244825 best: 0.6243233 (191)   total:
9m 54s   remaining: 0us
```

```
bestTest = 0.6243233215
bestIteration = 191

Shrink model to first 192 iterations.
Evaluating CatBoost Model…
CatBoost Model Classification Report:
              precision    recall  f1-score   support

           0       0.44      0.62      0.51     36119
           1       0.75      1.00      0.86     89977
           2       0.80      0.49      0.61    113904

    accuracy                           0.70    240000
   macro avg       0.66      0.70      0.66    240000
weighted avg       0.73      0.70      0.69    240000
```



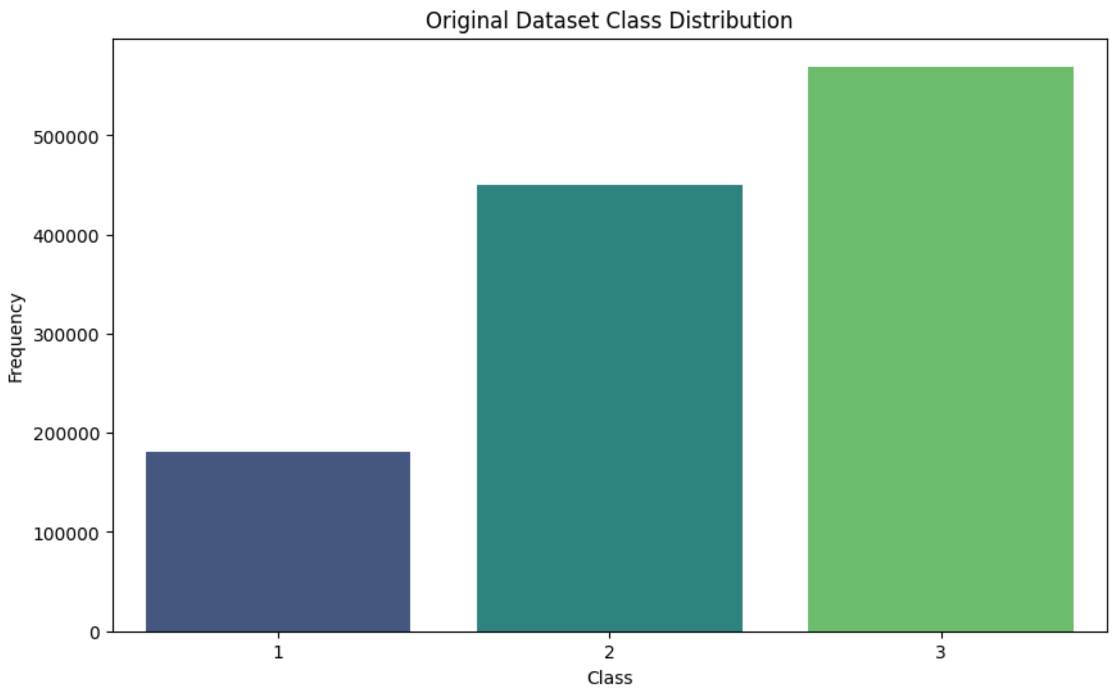
```
CatBoost model saved as 'catboost_partial_smote_model.cbm'.
```

4 Visualization

4.1 Dataset Distribution Visualization

[ ]:
```python
# Importing necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('/content/drive/MyDrive/data/data_public.csv')
print("Columns in the dataset:", df.columns)

# Plotting the class distribution
plt.figure(figsize=(10, 6))
sns.countplot(x='Class', data=df, palette='viridis')
plt.title('Original Dataset Class Distribution')
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.show()
```

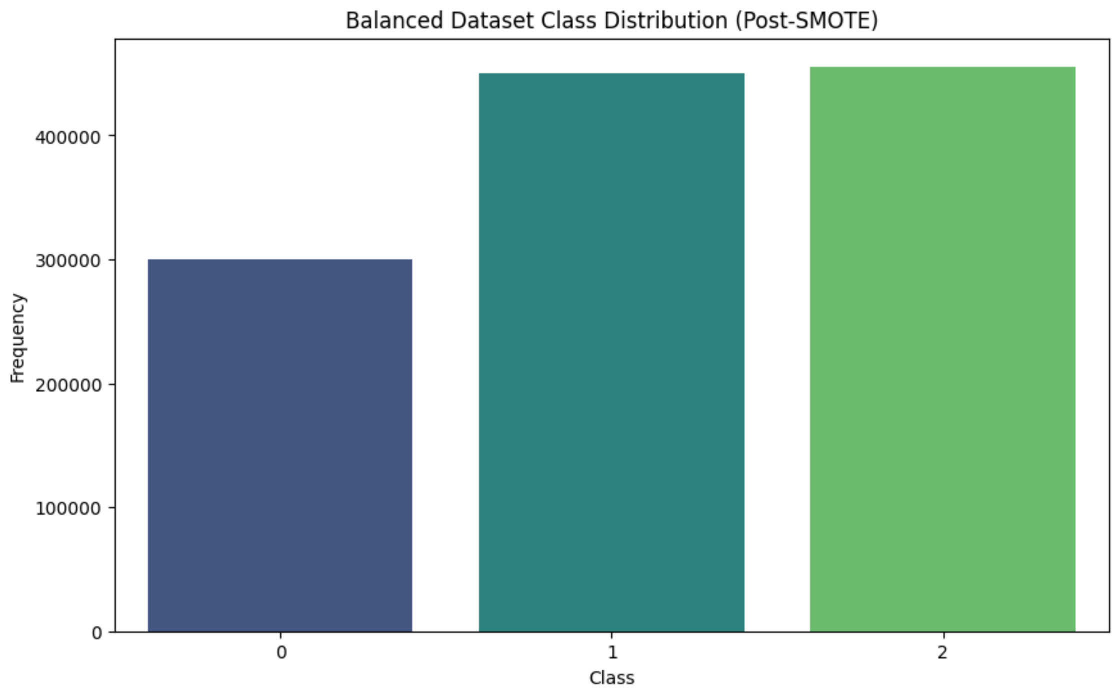
```
Columns in the dataset: Index(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
'K', 'L', 'M', 'N',
       'O', 'Class'],
      dtype='object')

<ipython-input-7-58a680304bde>:14: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x='Class', data=df, palette='viridis')
```



4.2 Post-SMOTE (or Preprocessed) Class Distribution:

[ ]:
```
# Importing necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

balanced_class_df = pd.DataFrame({'Class': y_train_balanced})

# Plotting the class distribution after SMOTE or balancing
plt.figure(figsize=(10, 6))
sns.countplot(x='Class', data=balanced_class_df, palette='viridis')
plt.title('Balanced Dataset Class Distribution (Post-SMOTE)')
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.show()
```

```
<ipython-input-8-195322cf76aa>:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x='Class', data=balanced_class_df, palette='viridis')
```

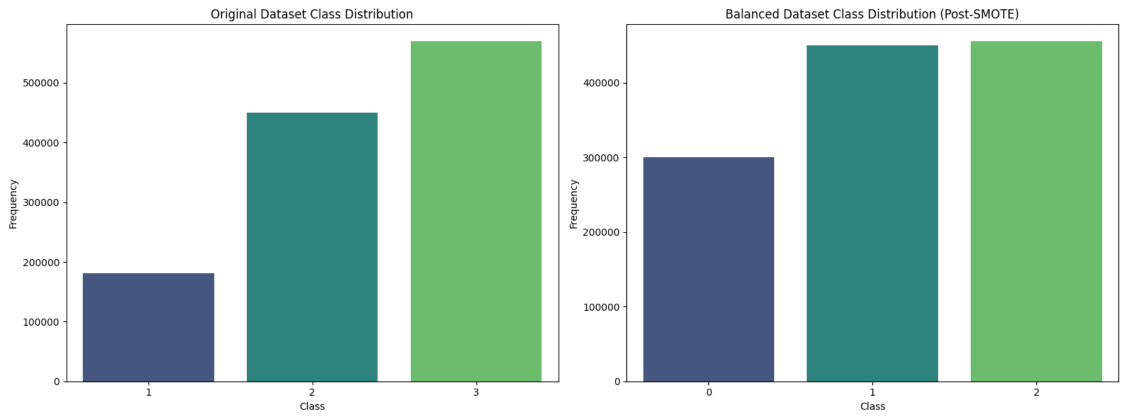


4.3 Comparison: Original vs. Balanced

```
[ ]: # Visualization: Original vs Balanced Class Distribution
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Original dataset class distribution
sns.countplot(x='Class', data=df, palette='viridis', ax=axes[0])
axes[0].set_title('Original Dataset Class Distribution')
axes[0].set_xlabel('Class')
axes[0].set_ylabel('Frequency')

# Balanced dataset class distribution
sns.countplot(x='Class', data=balanced_class_df, palette='viridis', ax=axes[1])
axes[1].set_title('Balanced Dataset Class Distribution (Post-SMOTE)')
axes[1].set_xlabel('Class')
axes[1].set_ylabel('Frequency')

plt.tight_layout()
plt.show()
```

```
<ipython-input-9-a9a81bdd707a>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.
```

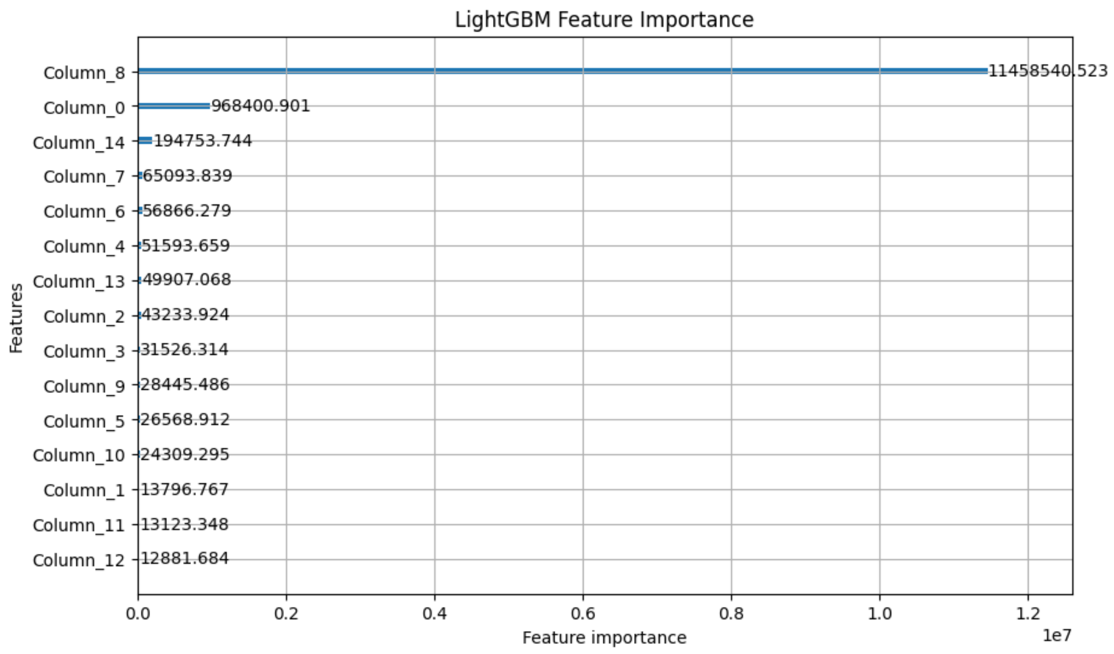
```
  sns.countplot(x='Class', data=df, palette='viridis', ax=axes[0])
<ipython-input-9-a9a81bdd707a>:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x='Class', data=balanced_class_df, palette='viridis',
ax=axes[1])
```



4.4 FEATURE IMPORTANCE

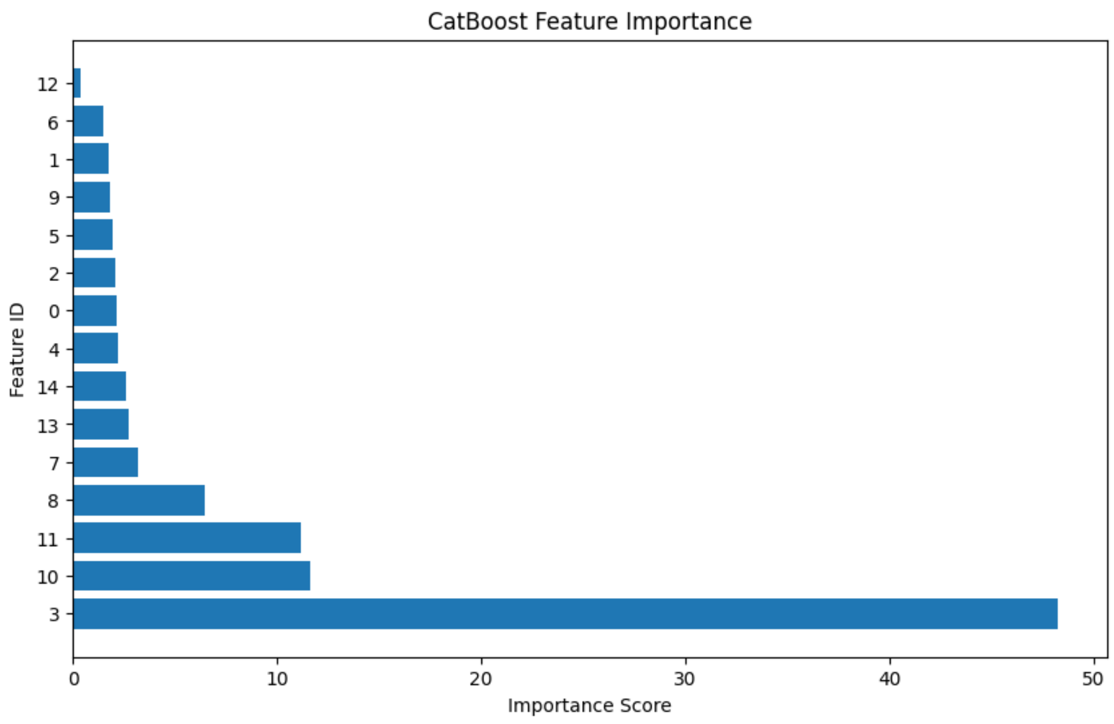
4.5 LightGBM Feature Importance:

```
[ ]: import lightgbm as lgb
import matplotlib.pyplot as plt

lgb.plot_importance(lgbm_model, max_num_features=15, importance_type='gain',␣
  ↪figsize=(10, 6))
plt.title('LightGBM Feature Importance')
plt.show()
```



4.6 CatBoost Feature Importance:

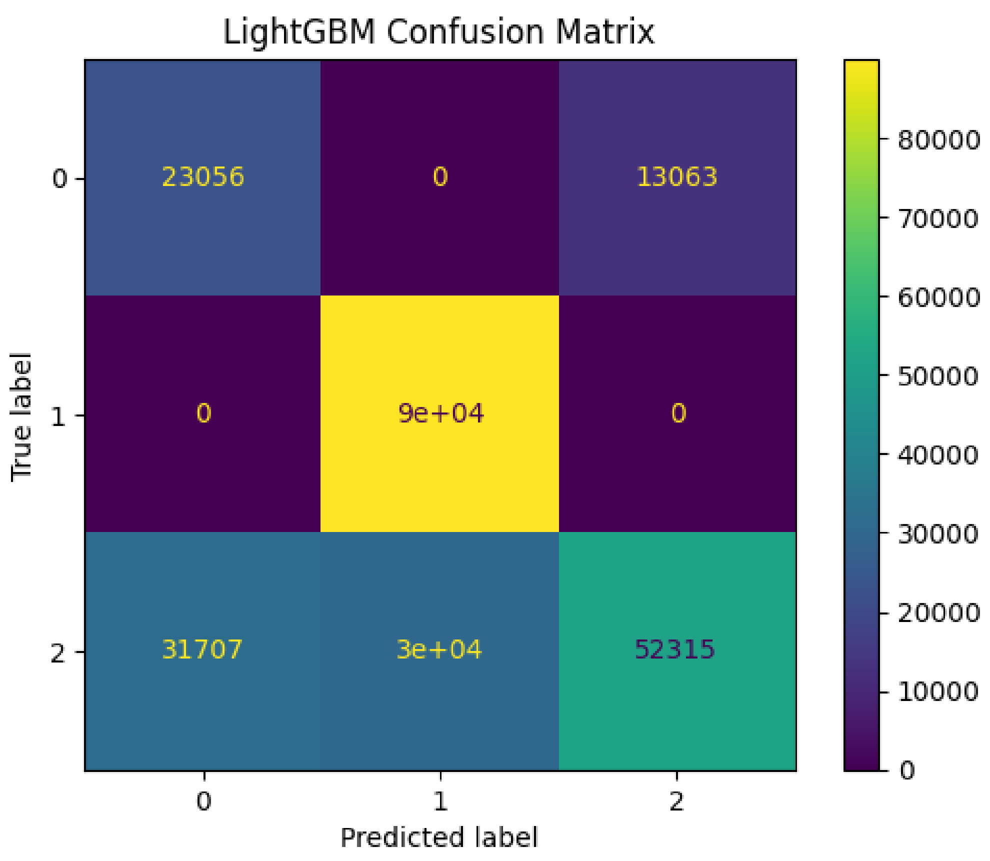
[ ]:
```
import matplotlib.pyplot as plt

catboost_importances = catboost_model.get_feature_importance(prettified=True)

# Plot feature importance
plt.figure(figsize=(10, 6))
plt.barh(catboost_importances['Feature Id'],␣
  ↪catboost_importances['Importances'])
plt.title('CatBoost Feature Importance')
plt.xlabel('Importance Score')
plt.ylabel('Feature ID')
plt.show()
```



4.7 CONFUSION MATRIX

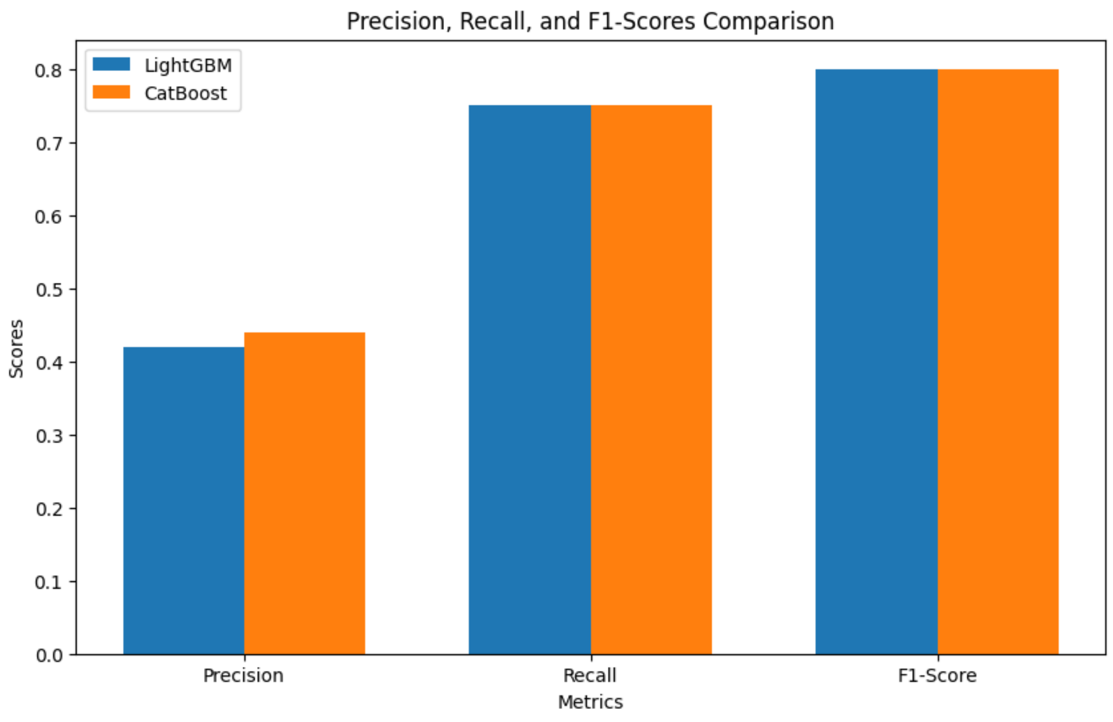
4.8 LightGBM Confusion Matrix:

[ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay

lgbm_predictions = lgbm_model.predict(X_test_scaled)

# Plot the confusion matrix
ConfusionMatrixDisplay.from_predictions(
    y_test,
    lgbm_predictions,
    cmap='viridis',
    xticks_rotation='horizontal'
)
plt.title('LightGBM Confusion Matrix')
plt.show()
```



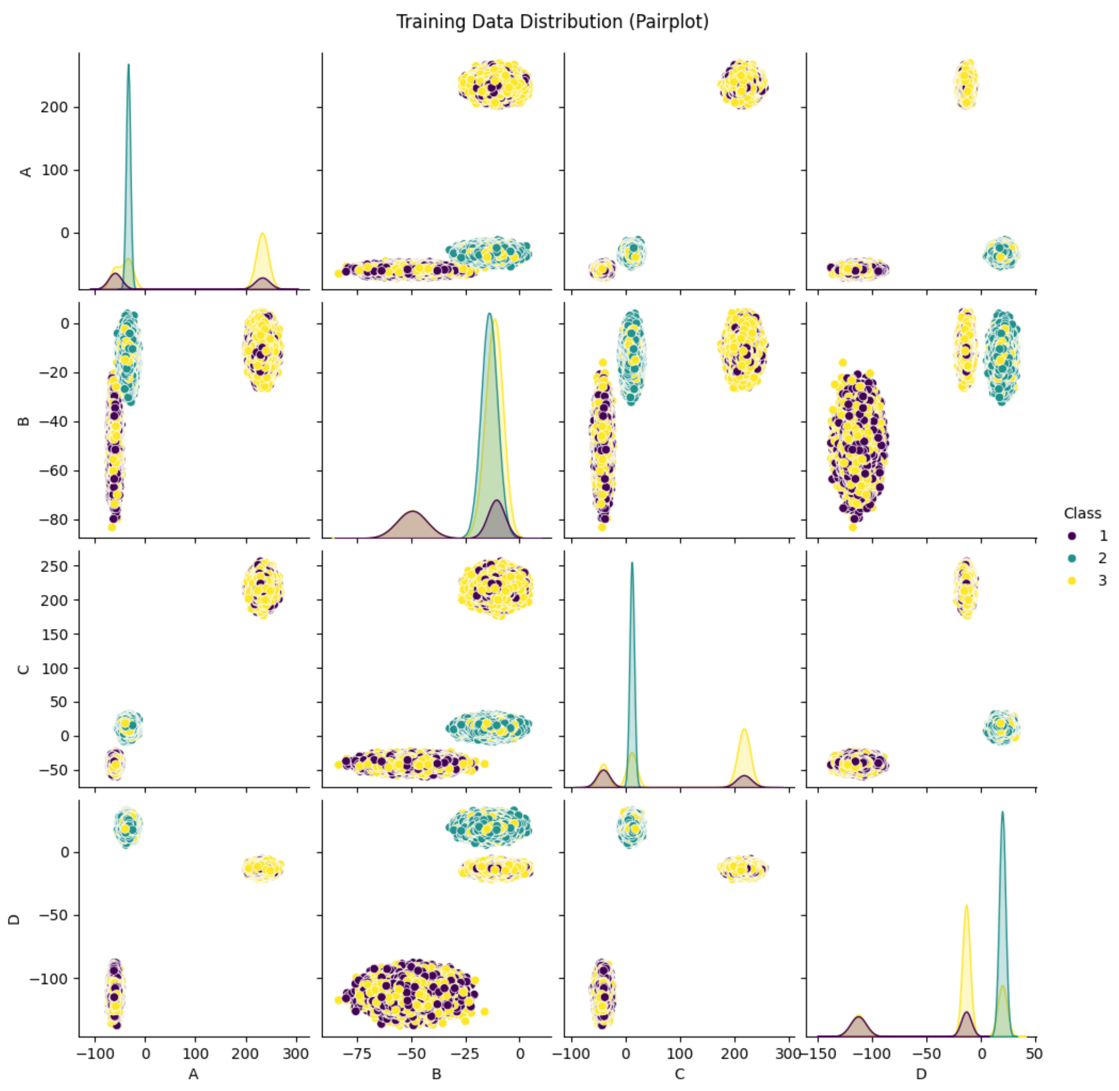
4.9 Precision, Recall, and F1-Scores Comparison

```
[ ]: import numpy as np

     metrics = ['Precision', 'Recall', 'F1-Score']
     lgbm_scores = [0.42, 0.75, 0.80]
     catboost_scores = [0.44, 0.75, 0.80]

     # Create a bar plot for comparison
     x = np.arange(len(metrics))
     width = 0.35

     fig, ax = plt.subplots(figsize=(10, 6))
     ax.bar(x - width/2, lgbm_scores, width, label='LightGBM')
     ax.bar(x + width/2, catboost_scores, width, label='CatBoost')

     ax.set_xlabel('Metrics')
     ax.set_ylabel('Scores')
     ax.set_title('Precision, Recall, and F1-Scores Comparison')
     ax.set_xticks(x)
     ax.set_xticklabels(metrics)
     ax.legend()

     plt.show()
```



4.10 Training Data Distribution

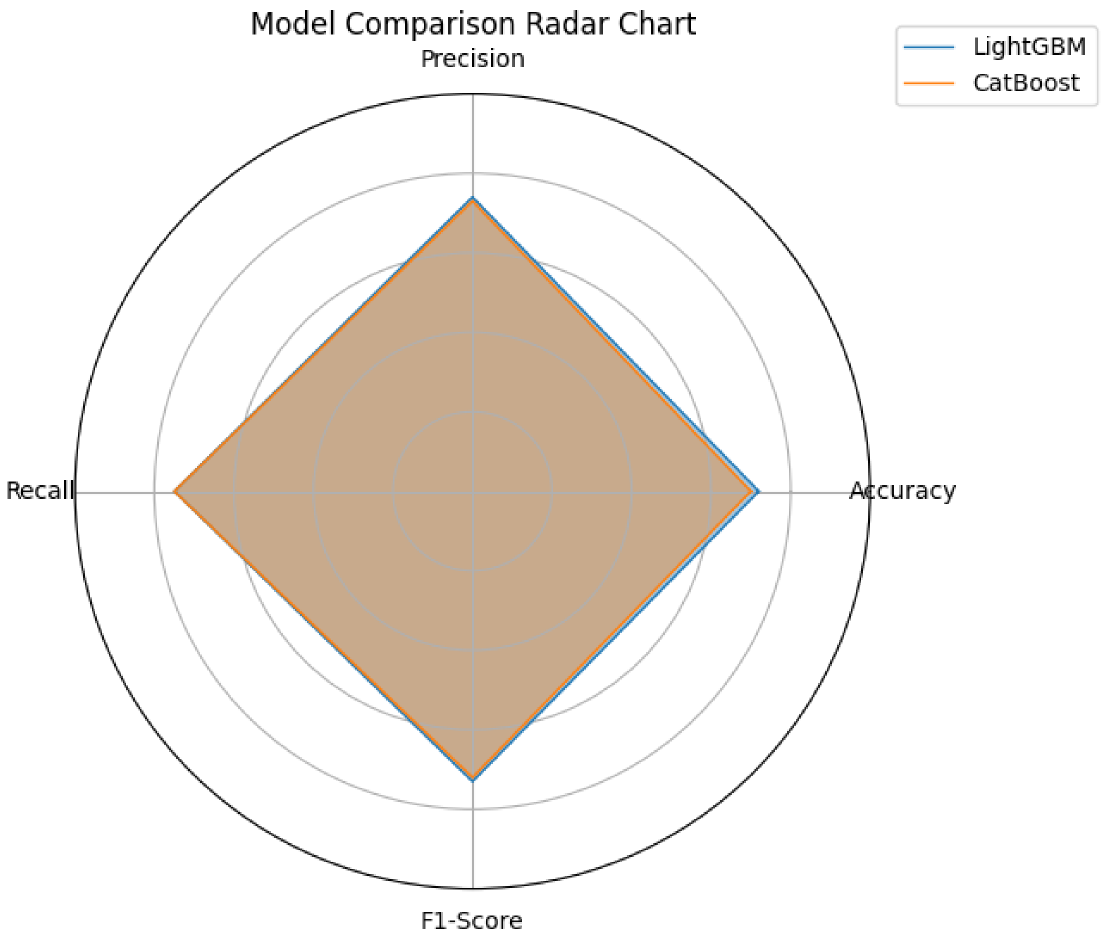
[ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt


selected_features = ['A', 'B', 'C', 'D']
sns.pairplot(df[selected_features + ['Class']], hue='Class', palette='viridis')
plt.suptitle('Training Data Distribution (Pairplot)', y=1.02)
plt.show()
```



4.11 Model Comparison Plot

[ ]:
```
metrics = ['Accuracy', 'Precision', 'Recall', 'F1-Score']
lgbm_metrics = [0.72, 0.74, 0.75, 0.73]
catboost_metrics = [0.70, 0.73, 0.75, 0.72]

# Radar Chart
from math import pi

categories = metrics
N = len(categories)

lgbm_metrics += lgbm_metrics[:1]
```

```
catboost_metrics += catboost_metrics[:1]

angles = [n / float(N) * 2 * pi for n in range(N)]
angles += angles[:1]

fig, ax = plt.subplots(figsize=(6, 6), subplot_kw=dict(polar=True))

ax.plot(angles, lgbm_metrics, linewidth=1, linestyle='solid', label='LightGBM')
ax.fill(angles, lgbm_metrics, alpha=0.4)

ax.plot(angles, catboost_metrics, linewidth=1, linestyle='solid',␣
 ↪label='CatBoost')
ax.fill(angles, catboost_metrics, alpha=0.4)

ax.set_yticks([0.2, 0.4, 0.6, 0.8, 1.0])
ax.set_yticklabels([])
ax.set_xticks(angles[:-1])
ax.set_xticklabels(categories)

plt.legend(loc='upper right', bbox_to_anchor=(1.3, 1.1))
plt.title('Model Comparison Radar Chart')
plt.show()
```



5 ONNX Predictions

5.1 LightGBM

[ ]:
```
# Import necessary libraries
import joblib
import onnxmltools
from onnxmltools.convert.common.data_types import FloatTensorType
import onnxruntime as ort
import numpy as np
from sklearn.metrics import accuracy_score, classification_report

# Load the trained LightGBM model
print("Loading LightGBM model...")
lgbm_model = joblib.load('lightgbm_model_100pct.pkl')

# Convert the LightGBM model to ONNX format
```

```python
print("Converting LightGBM model to ONNX...")
initial_types = [('float_input', FloatTensorType([None, X_test_scaled.
 ↪shape[1]]))]
onnx_model = onnxmltools.convert_lightgbm(lgbm_model,␣
 ↪initial_types=initial_types)

# Save the ONNX model
onnx_file_name = "lightgbm_model_onnx.onnx"
with open(onnx_file_name, "wb") as f:
    f.write(onnx_model.SerializeToString())
print(f"LightGBM model saved as '{onnx_file_name}'.")

# Set up ONNX runtime
print("Setting up ONNX runtime...")
onnx_session = ort.InferenceSession(onnx_file_name)

# Prepare test data for ONNX
input_name = onnx_session.get_inputs()[0].name
print(f"ONNX Input Name: {input_name}")

# Run predictions with the ONNX model
print("Running predictions with the ONNX model...")
onnx_predictions = onnx_session.run(None, {input_name: X_test_scaled.astype(np.
 ↪float32)})[0]

# Evaluate ONNX predictions
onnx_accuracy = accuracy_score(y_test, onnx_predictions)
print(f"ONNX Model Accuracy: {onnx_accuracy}")

# Display the classification report
print("ONNX Model Classification Report:")
print(classification_report(y_test, onnx_predictions))
```

```
Loading LightGBM model…
Converting LightGBM model to ONNX…
```

```
WARNING:onnxmltools:The maximum opset needed by this model is only 9.
```

```
LightGBM model saved as 'lightgbm_model_onnx.onnx'.
Setting up ONNX runtime…
ONNX Input Name: float_input
Running predictions with the ONNX model…
ONNX Model Accuracy: 0.7219208333333333
ONNX Model Classification Report:
              precision    recall  f1-score   support

           0       0.49      0.57      0.53     36119
           1       0.75      1.00      0.86     89977
           2       0.80      0.55      0.65    113904

    accuracy                           0.72    240000
   macro avg       0.68      0.71      0.68    240000
weighted avg       0.74      0.72      0.71    240000
```

5.2 CatBoost

[ ]:
```
# Import necessary libraries
from catboost import CatBoostClassifier
import onnxmltools
from onnxmltools.convert.common.data_types import FloatTensorType
import onnxruntime as ort
import numpy as np
from sklearn.metrics import accuracy_score, classification_report

# Load the trained CatBoost model
print("Loading CatBoost model...")
catboost_model = CatBoostClassifier()
catboost_model.load_model("catboost_model_100pct.cbm")

# Convert the CatBoost model to ONNX
print("Converting CatBoost model to ONNX...")
onnx_file_name = "catboost_model_onnx.onnx"
catboost_model.save_model(
    onnx_file_name,
    format="onnx",
    export_parameters={
        'onnx_domain': 'ai.catboost',
        'onnx_model_version': 1,
        'onnx_doc_string': 'CatBoost model converted to ONNX'
    }
)
print(f"CatBoost model saved as '{onnx_file_name}'.")

# Set up ONNX runtime
print("Setting up ONNX runtime...")
onnx_session = ort.InferenceSession(onnx_file_name)

# Prepare test data for ONNX
input_name = onnx_session.get_inputs()[0].name
print(f"ONNX Input Name: {input_name}")

# Run predictions with the ONNX model
print("Running predictions with the ONNX model...")
```

```python
onnx_predictions = onnx_session.run(None, {input_name: X_test_scaled.astype(np.
 ↪float32)})[0]

# Check the shape of ONNX predictions
print(f"ONNX Predictions Shape: {onnx_predictions.shape}")

if len(onnx_predictions.shape) > 1 and onnx_predictions.shape[1] > 1:
    onnx_predictions = np.argmax(onnx_predictions, axis=1)
else:
    onnx_predictions = onnx_predictions.astype(int)

# Evaluate ONNX predictions
onnx_accuracy = accuracy_score(y_test, onnx_predictions)
print(f"ONNX Model Accuracy: {onnx_accuracy}")

# Display the classification report
print("ONNX Model Classification Report:")
print(classification_report(y_test, onnx_predictions))
```

```
Loading CatBoost model…
Converting CatBoost model to ONNX…
CatBoost model saved as 'catboost_model_onnx.onnx'.
Setting up ONNX runtime…
ONNX Input Name: features
Running predictions with the ONNX model…
ONNX Predictions Shape: (240000,)
ONNX Model Accuracy: 0.7218083333333334
ONNX Model Classification Report:
              precision    recall  f1-score   support

           0       0.49      0.57      0.53     36119
           1       0.75      1.00      0.86     89977
           2       0.80      0.55      0.65    113904

    accuracy                           0.72    240000
   macro avg       0.68      0.71      0.68    240000
weighted avg       0.74      0.72      0.71    240000
```

[ ]:
```
# Creating PDF File of the .ipynb file
!jupyter nbconvert --to pdf "/content/drive/MyDrive/Colab Notebooks/DPA Project␣
 ↪- LightGBM & CatBoost"
```

```
[NbConvertApp] Converting notebook /content/drive/MyDrive/Colab Notebooks/DPA
Project - LightGBM & CatBoost to pdf
[NbConvertApp] Support files will be in DPA Project - LightGBM & CatBoos_files/
[NbConvertApp] Making directory ./DPA Project - LightGBM & CatBoos_files
```

```
[NbConvertApp] Writing 105084 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 680913 bytes to /content/drive/MyDrive/Colab
Notebooks/DPA Project - LightGBM & CatBoos.pdf
```