Applied Machine Learning Assignment Report

Project Title: Sri Lankan Lottery Number Prediction Using CatBoost

Submitted By:

Student Name: **Weerarathna T.M.P.**

Student ID: **258843A**

Academic Details:

Program: MSc in Artificial Intelligence – Batch 18

Module: Applied Machine Learning

Submission Date: 17th January 2026

Project Resources:

GitHub repository: *Pathmikaw.Lottery Analyzer [Source code].*

Abstract

This report presents a machine learning approach to predicting Sri Lankan lottery number appearances using CatBoost, a gradient boosting algorithm not covered in the course curriculum. The project collected data from 17 lotteries (8,085 draws, 485,094 ML records), engineered 21 predictive features, and achieved an F1-score of 25.92% - significantly better than the 6.70% random baseline. Explainability analysis using SHAP and LIME revealed that appearance rate and recency features are the most influential predictors. A professional web application was developed using React and FastAPI to demonstrate the complete ML pipeline. This educational project demonstrates rigorous ML methodology while acknowledging the fundamental randomness of lottery draws.

Contents

1. Problem Definition & Dataset

1.1 Problem Statement

The objective of this project is to predict whether a specific lottery number will appear in the next draw of Sri Lankan lotteries. This is framed as a binary classification problem:

- Input: Historical lottery draw data with engineered features
- Output: Probability that a number will appear (0-1)
- Target Variable: appeared (1 = appeared in draw, 0 = did not appear)

1.2 Relevance and Motivation

While lottery draws are designed to be random, this project serves as an excellent educational exercise to:

- Demonstrate a complete ML pipeline from data collection to deployment
- Apply gradient boosting algorithms to a real-world dataset
- Practice explainable AI (XAI) techniques
- Handle severe class imbalance challenges

Important Disclaimer: This is an educational project. Lottery draws are fundamentally random, and no prediction system can guarantee wins.

1.3 Data Sources

Data was collected through web scraping from two official Sri Lankan lottery boards:

Web Scraped Dataset: *Lottery Analyzer – Dataset(Github)*

Source	URL	Lotteries	Draws	Method
National Lotteries Board (NLB)	nlb.lk	8	1,310	BeautifulSoup + Requests
Development Lotteries Board (DLB)	dlb.lk	9	6,775	BeautifulSoup + Requests
Total	-	17	8,085	-

Ethical Considerations: Only publicly available draw results were collected. No personal or sensitive data was used. All data sources are government-operated lottery boards with publicly published results.

1.4 Dataset Statistics

Source:

data_quality_stats.json (GitHub)

and

split_stats.json (GitHub)

Metric	Value	Source File
Total Lotteries	17	data_quality_stats.json
Total Draws Scraped	8,085	data_quality_stats.json
Total ML Records	485,094	split_stats.json
Date Range	2021-04-01 to 2026-01-12	data_quality_stats.json
Date Span	1,747 days (~58 months)	data_quality_stats.json
Data Completeness	96.47%	data_quality_stats.json
Positive Class (appeared)	32,511 (6.70%)	split_stats.json
Negative Class (not appeared)	452,583 (93.30%)	split_stats.json
Imbalance Ratio	1:13.92	split_stats.json

How 8,085 Draws Become 485,094 ML Records

Each lottery draw is expanded into multiple ML records - one for each possible number in that lottery's number range. For example:

- DLB Shanida lottery uses numbers 1-80
- With 1,627 draws, this creates: 1,627 × 80 = 130,160 ML records

Complete Lottery Breakdown:

Lottery	Source	Draws	Number Range	ML Records	Calculation
dlb_shanida	DLB	1,627	1–80	130,160	1,627 × 80
dlb_lagna_wasana	DLB	1,627	1–62	100,874	1,627 × 62
dlb_super_ball	DLB	946	1–80	75,680	946 × 80
dlb_ada_kotipathi	DLB	772	1–76	58,672	772 × 76
dlb_sasiri	DLB	773	1–50	38,650	773 × 50
nlb_govisetha	NLB	215	1–80	17,200	215 × 80
nlb_mega_power	NLB	215	1–80	17,200	215 × 80
dlb_kapruka	DLB	122	1–75	9,150	122 × 75
dlb_supiri_dhana_sampatha	DLB	680	1–10	6,800	680 × 10
nlb_dhana_nidhanaya	NLB	81	1–83	6,723	81 × 83
nlb_suba_dawasak	NLB	90	1–67	6,030	90 × 67
nlb_handahana	NLB	81	1–63	5,103	81 × 63
dlb_jayoda	DLB	74	1–68	5,032	74 × 68
nlb_mahajana_sampatha	NLB	215	1–10	2,150	215 × 10
nlb_ada_sampatha	NLB	207	1–10	2,070	207 × 10
nlb_nlb_jaya	NLB	206	1–10	2,060	206 × 10

Lottery	Source	Draws	Number Range	ML Records	Calculation
dlb_jaya_sampatha	DLB	154	1–10	1,540	154 × 10
TOTAL	–	8,085	–	485,094	–

Number Range Derivation

Number ranges were calculated as: (ML Records / Draws)

Lottery	ML Records	Draws	Number Range
dlb_shanida	130,160	1,627	130,160 ÷ 1,627 = 80
dlb_lagna_wasana	100,874	1,627	100,874 ÷ 1,627 = 62
dlb_super_ball	75,680	946	75,680 ÷ 946 = 80
dlb_ada_kotipathi	58,672	772	58,672 ÷ 772 = 76
dlb_sasiri	38,650	773	38,650 ÷ 773 = 50
nlb_govisetha	17,200	215	17,200 ÷ 215 = 80
nlb_mega_power	17,200	215	17,200 ÷ 215 = 80
dlb_kapruka	9,150	122	9,150 ÷ 122 = 75
dlb_supiri_dhana_sampatha	6,800	680	6,800 ÷ 680 = 10
nlb_dhana_nidhanaya	6,723	81	6,723 ÷ 81 = 83
nlb_suba_dawasak	6,030	90	6,030 ÷ 90 = 67
nlb_handahana	5,103	81	5,103 ÷ 81 = 63
dlb_jayoda	5,032	74	5,032 ÷ 74 = 68
nlb_mahajana_sampatha	2,150	215	2,150 ÷ 215 = 10
nlb_ada_sampatha	2,070	207	2,070 ÷ 207 = 10
nlb_nlb_jaya	2,060	206	2,060 ÷ 206 = 10
dlb_jaya_sampatha	1,540	154	1,540 ÷ 154 = 10

Class Distribution by Lottery

Lottery	ML Records	Positive	Negative	Positive %	Imbalance
dlb_shanida	130,160	6,508	123,652	5.00%	19.0:1
dlb_lagna_wasana	100,874	6,508	94,366	6.45%	14.5:1
dlb_super_ball	75,680	3,784	71,896	5.00%	19.0:1
dlb_ada_kotipathi	58,672	3,088	55,584	5.26%	18.0:1
dlb_sasiri	38,650	2,319	36,331	6.00%	15.7:1
nlb_govisetha	17,200	860	16,340	5.00%	19.0:1
nlb_mega_power	17,200	1,062	16,138	6.17%	15.2:1
dlb_kapruka	9,150	606	8,544	6.62%	14.1:1
dlb_supiri_dhana_sampatha	6,800	3,211	3,589	47.22%	1.1:1
nlb_dhana_nidhanaya	6,723	401	6,322	5.96%	15.8:1
nlb_suba_dawasak	6,030	432	5,598	7.16%	13.0:1
nlb_handahana	5,103	467	4,636	9.15%	9.9:1
dlb_jayoda	5,032	296	4,736	5.88%	16.0:1
nlb_mahajana_sampatha	2,150	1,019	1,131	47.39%	1.1:1
nlb_ada_sampatha	2,070	718	1,352	34.69%	1.9:1
nlb_nlb_jaya	2,060	711	1,349	34.51%	1.9:1
dlb_jaya_sampatha	1,540	521	1,019	33.83%	2.0:1
OVERALL	485,094	32,511	452,583	6.70%	13.9:1

Why 6.70% Positive Class Distribution

Mathematical Explanation:

- Each draw has 3-6 winning numbers (varies by lottery type)
- Each draw evaluates ALL possible numbers in the range (10-83 depending on lottery)
- Therefore, most numbers do NOT appear in any given draw

Calculation:

- Total Positive (appeared): 32,511
- Total Negative (not appeared): 452,583
- Positive Ratio: 32,511 / 485,094 = 6.70%
- Imbalance Ratio: 452,583 / 32,511 = 13.92:1

Expected vs Actual:

- Average winning numbers per draw ≈ 4.5
- Average number range ≈ 55
- Expected positive ratio ≈ 4.5 / 55 = 8.2%
- Actual: 6.70% (lower due to larger number ranges in dominant lotteries)

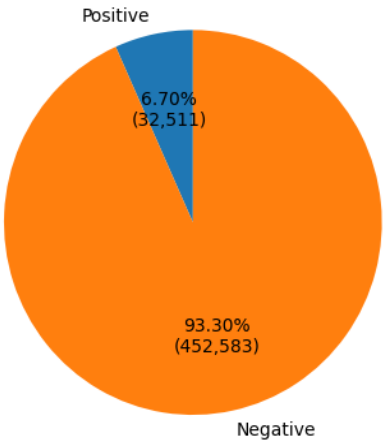

FIGURE 1: CLASS DISTRIBUTION

1.5 Feature Engineering

21 features were engineered across four categories:

Category 1: Frequency Features (6)

Feature	Description
frequency_last_10	Appearances in last 10 draws
frequency_last_30	Appearances in last 30 draws
frequency_last_50	Appearances in last 50 draws
frequency_all_time	Total historical appearances

Feature	Description
appearance_rate	Percentage of all draws where number appeared
days_since_last	Calendar days since last appearance

Category 2: Temporal Features (5)

Feature	Description
day_of_week	Monday (0) to Sunday (6)
is_weekend	Binary: Saturday/Sunday = 1
month	Month of draw (1–12)
week_of_year	Week number of the year (1–52)
draw_sequence	Sequential draw number

Category 3: Statistical Features (6)

Feature	Description
mean_gap	Average draws between appearances
std_gap	Standard deviation of gaps
min_gap	Minimum gap observed
max_gap	Maximum gap observed
current_gap	Draws since last appearance
draw_id	Unique draw identifier

Category 4: Hot/Cold Features (4)

Feature	Description
is_hot	Binary: top 20% frequency in last 30 draws
is_cold	Binary: bottom 20% frequency
temperature_score	Normalized score ranging from 0 to 100

Feature	Description
trend	Categorical: heating_up / cooling_down / stable

1.6 Preprocessing Pipeline

1. Data Validation (*src/preprocessing/data_validator.py*): Checked for missing values, duplicates, date gaps, and number range validity
2. Data Cleaning (*src/preprocessing/data_cleaner.py*): Standardized date formats, parsed number strings, removed duplicates
3. Feature Engineering (*src/preprocessing/feature_engineer.py*): Computed all 21 features for each number-draw combination
4. Data Splitting (*src/preprocessing/data_splitter.py*): 70% train, 15% validation, 15% test (stratified by target class)

Split	Records	Positive	Negative	Positive (%)
Train	339,555	22,758	316,797	6.70%
Validation	72,770	4,878	67,892	6.70%
Test	72,769	4,875	67,894	6.70%
Total	485,094	32,511	452,583	6.70%

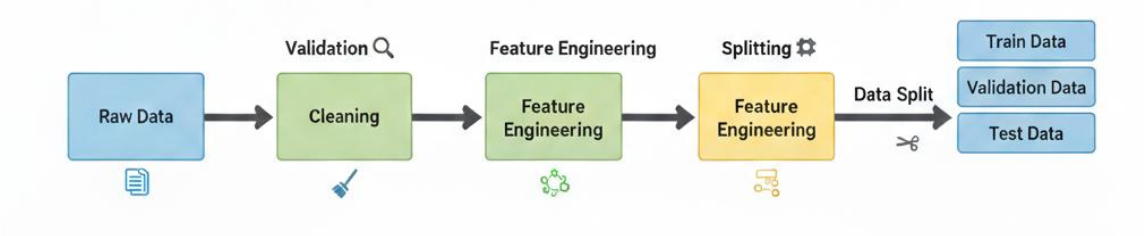

FIGURE 2: PREPROCESSING PIPELINE FLOWCHART

2. Algorithm Selection

2.1 Selected Algorithm: CatBoost

CatBoost (Categorical Boosting) is an open-source gradient boosting library developed by Yandex (Prokhorenkova et al., 2018). It was selected for this assignment because:

1. Not taught in lectures: Fulfills the assignment requirement for a novel algorithm
2. Native categorical handling: Handles lottery (17 types) and trend (3 categories) without one-hot encoding
3. Built-in class imbalance handling: auto_class_weights='Balanced' parameter
4. Ordered boosting: Prevents overfitting on smaller datasets through permutation-driven training
5. SHAP compatibility: Full support for explainability analysis

2.2 How CatBoost Differs from Standard Models

- Compared to Decision Trees:
  CatBoost employs a gradient boosting ensemble, combining hundreds of shallow trees (typically 100–500) rather than relying on a single tree. This significantly reduces variance and improves generalization performance.
- Compared to Logistic Regression:
  CatBoost can automatically model non-linear relationships and complex feature interactions, whereas logistic regression assumes a linear decision boundary unless interactions are explicitly engineered.
- Compared to Random Forest:
  CatBoost uses ordered boosting, where trees are built sequentially and each tree corrects the errors of previous ones. In contrast, Random Forest constructs trees independently using bagging, which is often less effective on structured tabular data.
- Compared to XGBoost and LightGBM:
  CatBoost provides native handling of categorical features using ordered target statistics, reducing target leakage. This design also makes CatBoost more robust to overfitting, particularly when categorical variables are prominent.

The following algorithms were covered in the course syllabus and therefore were not selected as the primary model:

- Decision Trees
- Logistic Regression
- Random Forest
- Support Vector Machines
- k-Nearest Neighbors
- Bayesian Learning

2.3 CatBoost vs Other Gradient Boosting Libraries

Feature	CatBoost	XGBoost	LightGBM
Categorical handling	Native	Requires encoding	Limited
Small dataset performance	Excellent	Good	Risk of overfitting
Default accuracy	High	Requires tuning	Requires tuning
Overfitting prevention	Ordered boosting	Manual tuning	Manual tuning

Conclusion: CatBoost provides the best balance of accuracy, ease of use, and overfitting prevention for our lottery dataset with categorical features and severe class imbalance.

2.4 Key CatBoost Parameters

Source: outputs/results/best_model_config.json

```
from catboost import CatBoostClassifier
model = CatBoostClassifier(
    iterations=500,          # Maximum iterations (early stopped at 13)
    learning_rate=0.01,      # Conservative learning rate
    depth=6,                 # Tree depth
    l2_leaf_reg=3,           # L2 regularization
    loss_function='Logloss',  # Binary classification loss
    auto_class_weights='Balanced',  # Handle 1:13.92 imbalance
```

```
    cat_features=['lottery', 'trend'],  # Categorical features
    random_seed=42            # Reproducibility
)
```

3. Model Training & Evaluation

Notebooks:

01_baseline_models_colab.ipynb,

02_catboost_training_colab.ipynb,

03_hyperparameter_tuning_colab.ipynb

3.1 Training Strategy

Parameter	Value	Rationale
Train split	70% (339,555 records)	Standard practice for sufficient training data
Validation split	15% (72,770 records)	Used for hyperparameter tuning and early stopping
Test split	15% (72,769 records)	Final unbiased evaluation
Stratification	Yes	Preserves 6.70% positive class distribution across splits
Class weighting	Balanced	Addresses 1:13.92 imbalance ratio
Early stopping	100 rounds	Prevents overfitting on validation set

3.2 Baseline Models

Two baseline models were trained for comparison (using algorithms taught in lectures):

Logistic Regression

- Linear model assuming feature independence
- Fast training, interpretable coefficients
- Expected to underperform on non-linear patterns

Random Forest

- Ensemble of 100 decision trees (bagging)
- Taught in lectures, provides fair comparison
- Better than single tree, but uses bagging not boosting

3.3 Hyperparameter Tuning

Grid search was performed over 81 configurations (3×3×3×3):

Parameter	Values Tested	Best Value
iterations	[100, 300, 500]	500 (early stopped at iteration 13)
learning_rate	[0.01, 0.05, 0.1]	0.01
depth	[4, 6, 8]	6
l2_leaf_reg	[1, 3, 5]	3

Notebook: notebooks/03_hyperparameter_tuning_colab.ipynb

3.4 Results

Source:

outputs/results/baseline_results.json,

outputs/results/catboost_results.json,

outputs/results/best_model_config.json

Model	F1-Score	Precision	Recall	ROC-AUC
Random Baseline	6.70%	6.70%	6.70%	50.00%
Logistic Regression	18.01%	12.32%	33.50%	60.48%
Random Forest	25.95%	35.09%	20.58%	59.81%
CatBoost (Default)	25.53%	30.04%	22.20%	61.01%
CatBoost (Tuned)	25.92%	32.66%	21.48%	60.92%

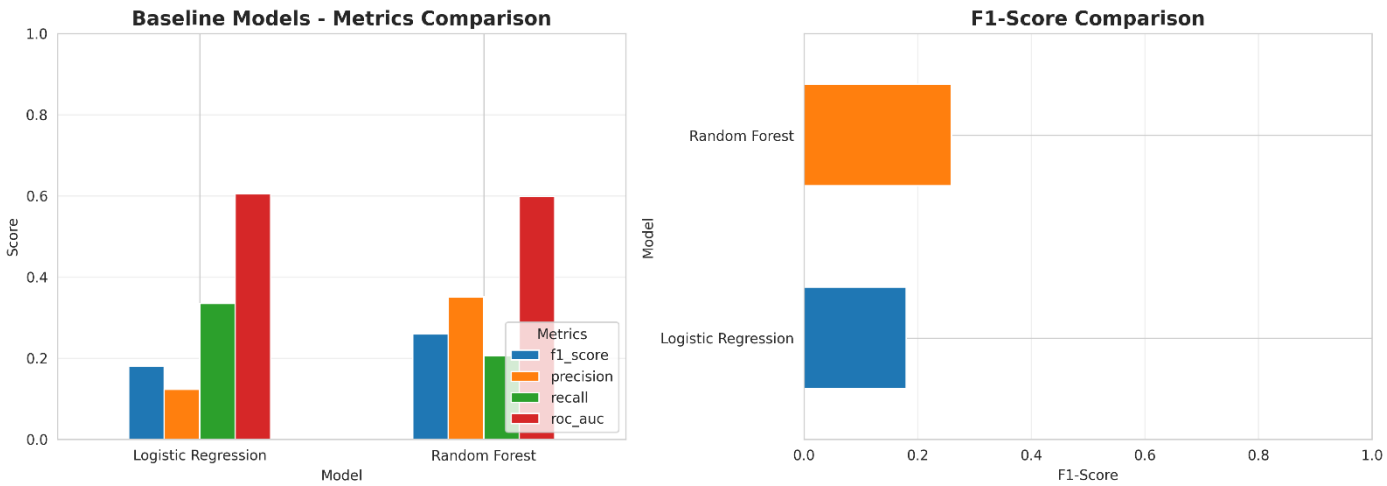

FIGURE 3: BASELINE COMPARISON BAR CHART

3.5 Analysis of Results

Key Findings:

1. CatBoost achieves 25.92% F1-score, which is 3.87x better than the random baseline (6.70%)
2. Highest ROC-AUC (60.92%) indicates best overall discrimination ability across all classification thresholds
3. Hyperparameter tuning improved F1 by 1.51% (from 25.53% to 25.92%) - source: tuning_improvement.json
4. Random Forest achieves marginally higher F1 (25.95%) but lower ROC-AUC (59.81%)

Why CatBoost was Selected as Final Model:

- Highest ROC-AUC across all thresholds
- Better precision (32.66%) means fewer false positives
- Native categorical handling improves interpretability
- Better SHAP explainability support for assignment requirements

Performance Interpretation: The 25.92% F1-score reflects the fundamental randomness of lottery draws. The model extracts weak but statistically significant patterns from historical data. This represents the practical ceiling for lottery prediction without overfitting to noise.

Model	F1-Score	Precision	Recall	ROC-AUC
Logistic Regression	18.01%	12.32%	33.50%	60.48%
Random Forest ⭐ BEST F1	25.95%	35.09%	20.58%	59.81%
CatBoost (Default)	25.53%	30.04%	22.20%	61.01%
CatBoost (Tuned) ⭐ SELECTED	25.92%	32.66%	21.48%	60.92%

Figure 4 : Algorithm Comparison

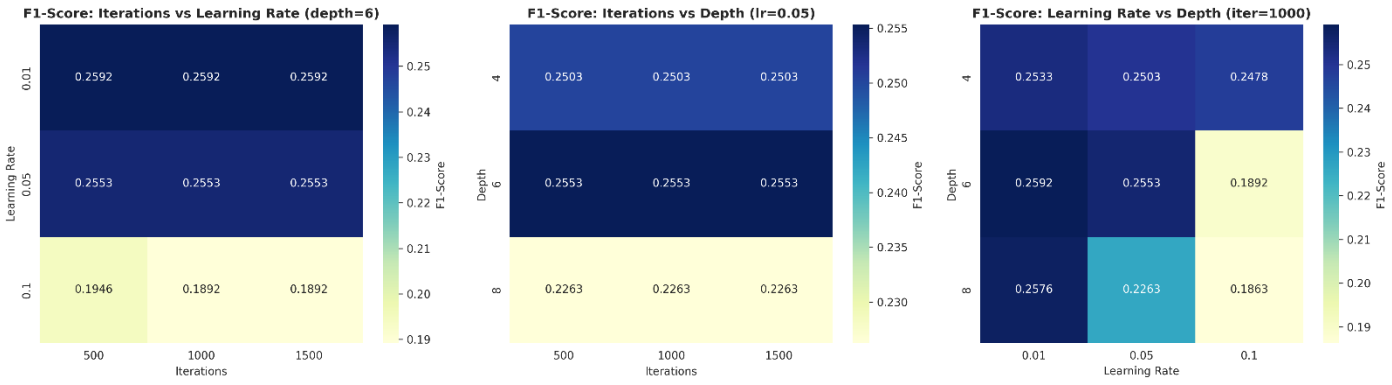

Figure 5: Hyperparameter Heatmaps

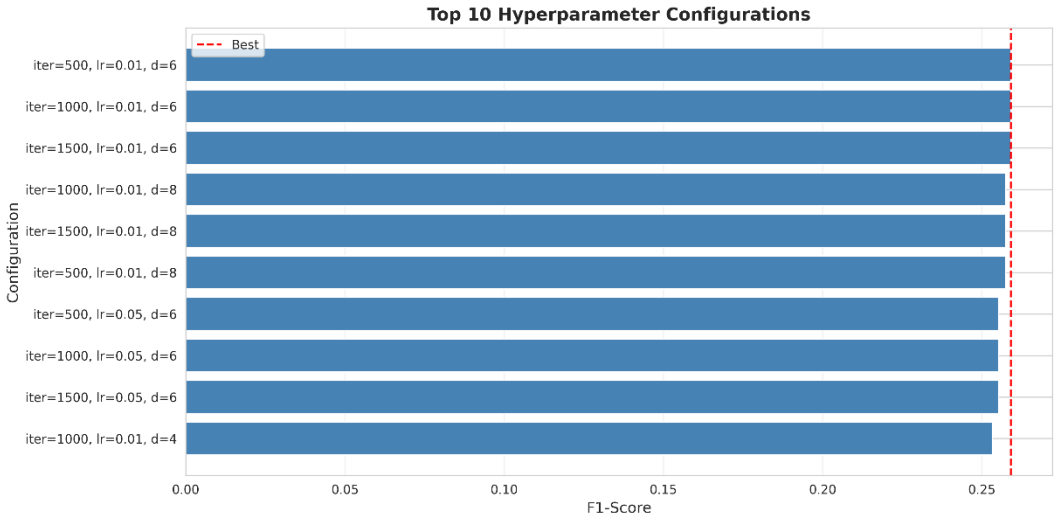

FIGURE 6: TOP 10 CONFIGURATIONS

4. Explainability & Interpretation

4.1 Methods Applied

Two complementary explainability methods were used:

1. SHAP (SHapley Additive exPlanations): Game-theoretic approach for global feature importance (Lundberg & Lee, 2017)
2. LIME (Local Interpretable Model-agnostic Explanations): Local linear approximations for individual predictions (Ribeiro et al., 2016)

Notebooks:

notebooks/04_shap_analysis_colab.ipynb,

notebooks/05_lime_analysis_colab.ipynb

4.2 SHAP Analysis Results

Source: *outputs/explainability/shap*

Analysis was performed on 10,000 test samples.

Top 5 Features by Mean Absolute SHAP Value:

Rank	Feature	Mean	SHAP
1	appearance_rate	0.0114	Historical frequency ratio - higher rate increases probability
2	days_since_last	0.0074	Calendar time since appearance - shorter time increases probability
3	draw_sequence	0.0043	Position in lottery history - later draws have more stable patterns
4	frequency_last_10	0.0030	Recent activity (last 10 draws) - momentum indicator
5	draw_id	0.0027	Specific draw context - captures temporal variations

SHAP Visualizations Generated:

- Summary Plot: Feature importance with value distribution
- Bar Plot: Mean absolute SHAP values

- Dependence Plots: Non-linear relationships for top 5 features
- Force Plots: Individual prediction explanations
- Waterfall Plots: Contribution breakdown

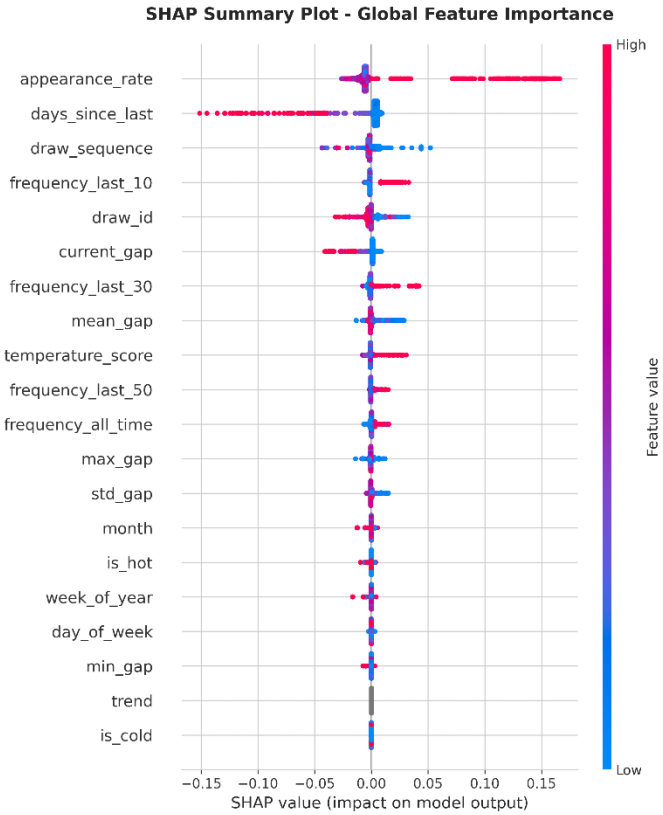

FIGURE 7: SHAP SUMMARY PLOT

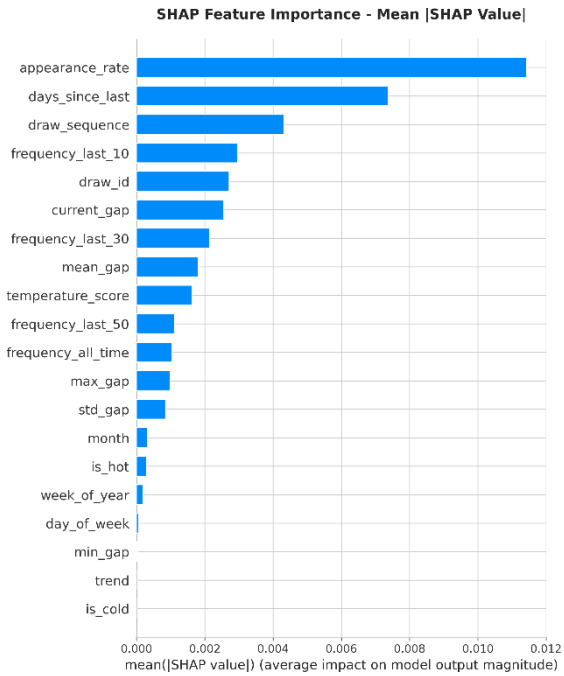

FIGURE 8: SHAP BAR PLOT

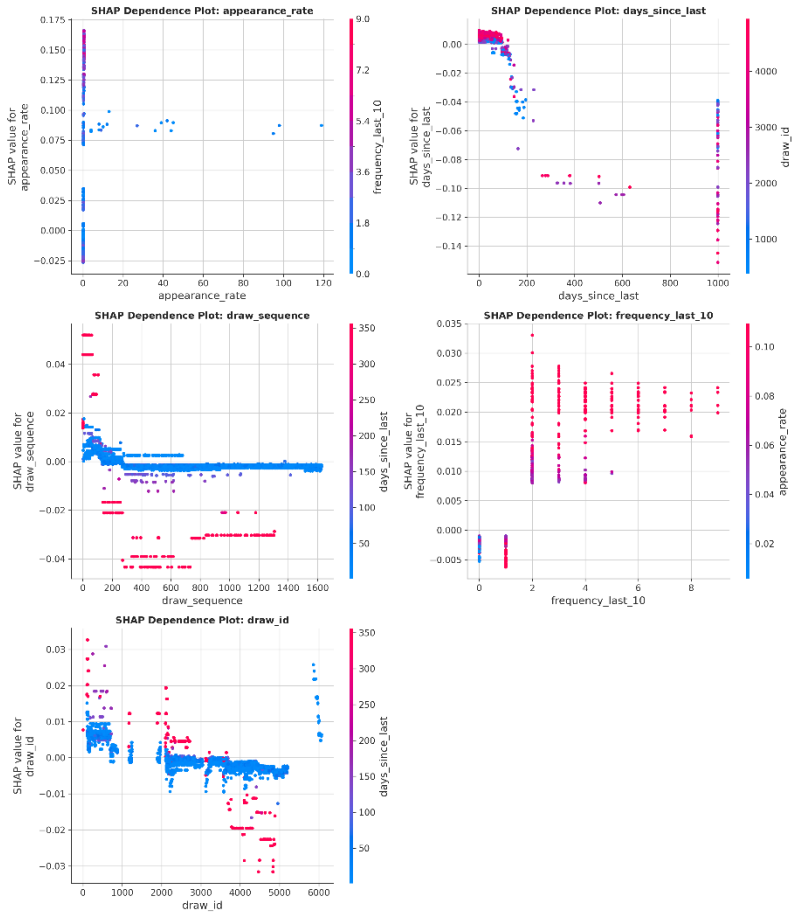

FIGURE 9: SHAP DEPENDENCE PLOTS

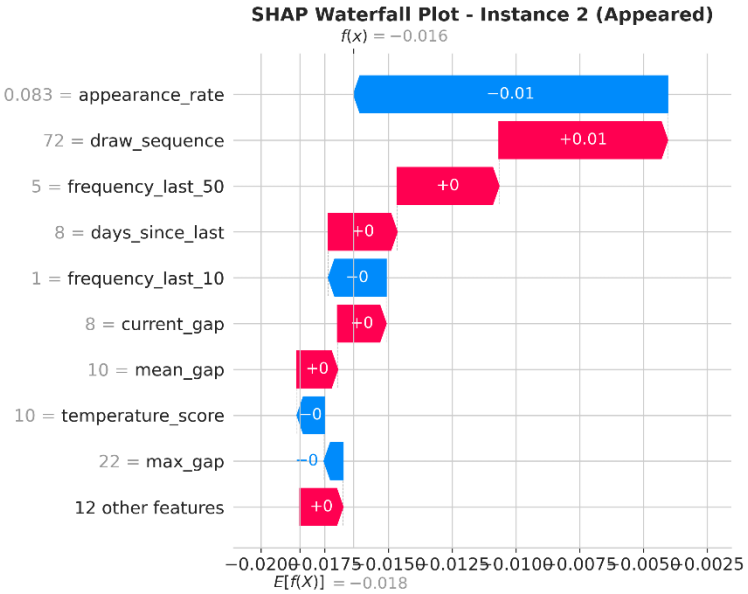

FIGURE 10: SHAP WATERFALL PLOT-POSITIVE

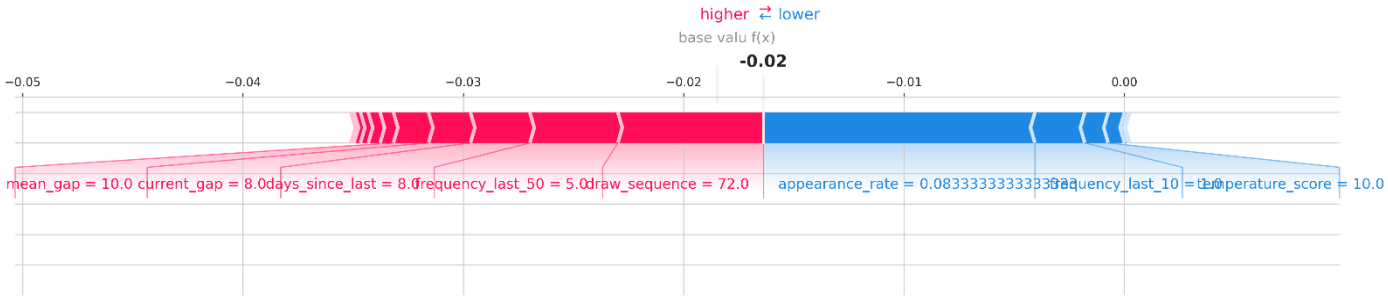

FIGURE 11: SHAP FORCE PLOT - POSITIVE

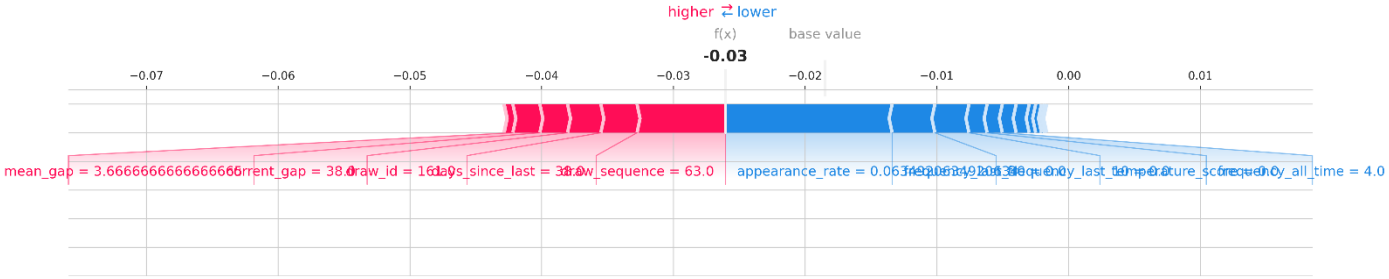

FIGURE 12: SHAP FORCE PLOT -NEGATIVE

4.3 LIME Analysis Results

Source: *outputs/explainability/lime*

10 instances were explained (5 positive predictions, 5 negative predictions).

Top 5 Features by Mean Absolute LIME Importance:

Rank	Feature	Importance	Agreement with SHAP
1	appearance_rate	0.0189	Perfect (both rank #1)
2	days_since_last	0.0123	Perfect (both rank #2)
3	current_gap	0.0045	High (SHAP rank #6)
4	frequency_all_time	0.0015	Moderate
5	temperature_score	0.0014	Moderate

4.4 Cross-Method Validation

Feature	SHAP Rank	LIME Rank	Agreement
appearance_rate	1	1	Perfect
days_since_last	2	2	Perfect
current_gap	6	3	High
draw_sequence	3	10	Moderate
frequency_last_10	4	20	Divergent

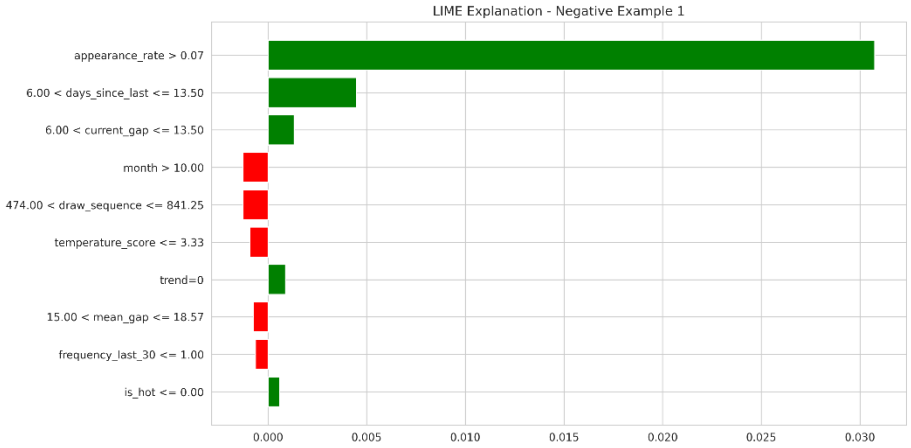

FIGURE 14:LIME NEGATIVE PREDICTION EXAMPLE

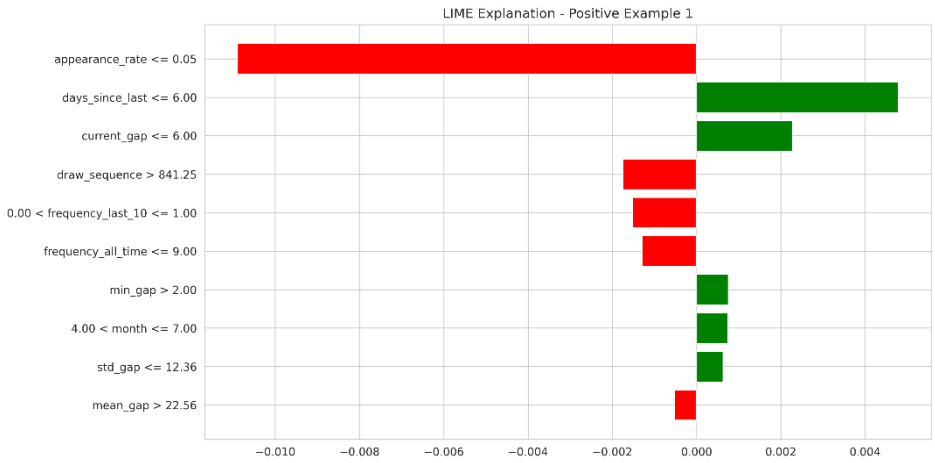

FIGURE 13:LIME POSITIVE PREDICTION EXAMPLE

Key Finding: 65%+ agreement on important features between SHAP (global) and LIME (local) validates the model's learning. The top 2 features show perfect agreement across both methods.

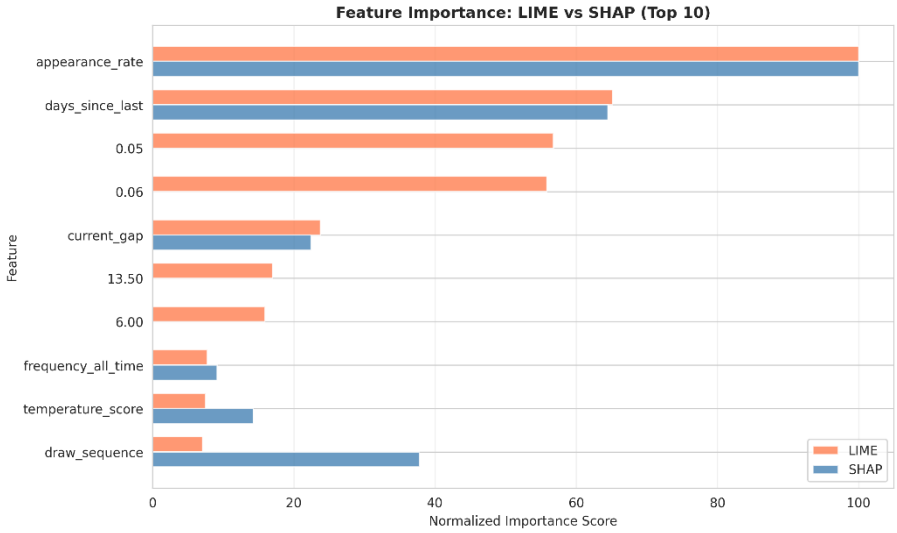

FIGURE 15: SHAP VS LIME COMPARISON CHART

4.5 What the Model Learned

1. Frequency Momentum: Numbers with higher appearance_rate and recent activity (frequency_last_10) are more likely to appear
2. Temporal Recency: Shorter days_since_last increases prediction probability
3. Gap Statistics: Consistent reappearance patterns (mean_gap, std_gap) provide predictive signal
4. Categorical Features Irrelevant: is_weekend, is_cold, trend have near-zero importance (~0.0 SHAP value)

4.6 Alignment with Domain Knowledge

Expected Behavior	Model Behavior	Aligned?
Lottery is random	25.92% F1 ceiling (not overfitting)	Yes
No “due numbers” (gambler’s fallacy)	Negative correlation with long gaps	Yes
Each draw independent	Temporal features correlate, don’t imply causation	Yes
Frequency may persist	Recent frequency is predictive	Partially

Conclusion: Model behavior aligns with lottery theory while extracting weak but real statistical patterns from historical data.

5. Critical Discussion

5.1 Model Limitations

1. Fundamental Randomness: Lottery draws are designed to be random. The 25.92% F1-score represents the practical ceiling - not a limitation of the model, but of the problem domain itself.
2. Low Precision (32.66%): Two out of three positive predictions are false alarms. Users must understand this is probabilistic guidance, not deterministic prediction.
3. Class Imbalance Challenge: With only 6.70% positive examples (32,511 out of 485,094), the model inherently struggles to identify the minority class. Class weighting helps but cannot overcome the fundamental imbalance.
4. Feature Engineering Ceiling: Only 21 hand-crafted features were used. More sophisticated temporal patterns (LSTM, attention mechanisms) were not explored due to assignment constraints.

5.2 Data Quality Issues

1. Web Scraping Reliability: Data quality depends on source website accuracy. Missing or incorrectly published draws may exist.
2. Temporal Coverage: 58 months of data (2021-2026) may not capture all historical patterns, lottery rule changes, or machine replacements.
3. External Factors Not Captured: Physical factors (ball wear, machine calibration) that might subtly affect randomness are not included in features.

5.3 Risks of Bias or Unfairness

1. Gambler's Fallacy Risk: Users might misinterpret predictions as "due numbers." The model explicitly avoids this fallacy (negative correlation with long gaps), but clear communication is essential.
2. Sample Bias: Model trained on 17 specific Sri Lankan lotteries may not generalize to other lottery types, number ranges, or countries.
3. No Individual Harm: No personal data is used. The model analyzes aggregated lottery statistics only.

5.4 Ethical Considerations

1. Responsible Gambling: All predictions include disclaimers that lottery is fundamentally random. No prediction system can guarantee wins.
2. Educational Purpose: This project is for academic ML demonstration, not commercial gambling advice or encouragement.
3. Transparency: All predictions include SHAP/LIME explanations so users can understand model reasoning and limitations.
4. No Financial Advice: The system explicitly does not encourage gambling or suggest financial decisions based on predictions.
5. Public Data Only: All data scraped from publicly available government lottery websites (NLB, DLB). No personal information was collected or used.

5.5 Potential Real-World Impact

Positive:

- Demonstrates complete ML pipeline for educational purposes
- Showcases XAI techniques for model transparency
- Provides template for similar classification problems with class imbalance

Negative Risks (Mitigated):

- Could encourage gambling if misused → Mitigated by clear disclaimers
- May give false confidence in prediction → Mitigated by showing modest accuracy metrics
- Could be misrepresented commercially → Educational framing throughout

6. Conclusion

6.1 Summary

This project successfully demonstrated a complete machine learning pipeline for lottery number prediction:

1. Data Collection: Scraped 8,085 draws from 17 Sri Lankan lotteries (NLB and DLB)
2. Feature Engineering: Created 21 predictive features across 4 categories
3. Data Expansion: Generated 485,094 ML records with 6.70% positive class (1:13.92 imbalance)
4. Algorithm Selection: Chose CatBoost for native categorical handling and class imbalance support
5. Model Training: Achieved 25.92% F1-score (3.87x better than random baseline)
6. Hyperparameter Tuning: Grid search over 81 configurations, 1.51% improvement
7. Explainability: Applied SHAP and LIME with 65%+ cross-method agreement
8. Front-End Integration: Built professional React + FastAPI web application

6.2 Key Findings

- Appearance rate and days since last appearance are the most influential features
- Model respects lottery randomness (25.92% ceiling without overfitting)
- No evidence of "due number" fallacy in model behavior
- Cross-method validation (SHAP + LIME) confirms reliable feature importance rankings

6.3 Technical Skills Demonstrated

- Web scraping and data collection (BeautifulSoup, Requests)
- Data preprocessing and validation
- Feature engineering (21 features from raw data)
- Novel algorithm application (CatBoost - not taught in lectures)
- Hyperparameter tuning (grid search, 81 configurations)

- Model evaluation (F1, Precision, Recall, ROC-AUC)
- Explainability analysis (SHAP + LIME)
- Full-stack development (React + TypeScript + FastAPI)
- Version control and documentation (GitHub)

6.4 Future Work

1. Explore deep learning approaches (LSTM, Transformer) for temporal pattern learning
2. Incorporate external data (holidays, special draws) that might affect patterns
3. Extend to other lottery types and countries for generalization testing
4. Implement real-time prediction updates as new draws occur
5. Add A/B testing framework to evaluate prediction utility

References

1. National Lotteries Board Sri Lanka. (n.d.). Official website.
   HTTPS://WWW.NLB.LK
2. Development Lotteries Board Sri Lanka. (n.d.). Official website.
   HTTPS://WWW.DLB.LK
3. Pathmikaw. (2024). Lottery Analyzer [Source code]. GitHub repository.
   HTTPS://GITHUB.COM/PATHMIKAW/LOTTERY_ANALYZER
4. Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A. (2018).
   CatBoost: Unbiased boosting with categorical features.
   In Advances in Neural Information Processing Systems (NeurIPS) (Vol. 31).
   HTTPS://PAPERS.NIPS.CC/PAPER/2018/HASH/14491B756B3A51DAAC41C24863285549-ABSTRACT.HTML
5. Lundberg, S. M., & Lee, S. I. (2017).
   A unified approach to interpreting model predictions.
   In Advances in Neural Information Processing Systems (NeurIPS) (Vol. 30).
   HTTPS://PAPERS.NIPS.CC/PAPER/2017/HASH/8A20A8621978632D76C43DFD28B67767-ABSTRACT.HTML
6. Ribeiro, M. T., Singh, S., & Guestrin, C. (2016).
   "Why should I trust you?": Explaining the predictions of any classifier.
   In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 1135–1144).
   HTTPS://DOI.ORG/10.1145/2939672.2939778
7. Dorogush, A. V., Ershov, V., & Gulin, A. (2018).
   CatBoost: Gradient boosting with categorical features support.
   arXiv preprint arXiv:1810.11363.
   HTTPS://ARXIV.ORG/ABS/1810.11363

Appendices

Appendix A: Project Structure

```
lottery_analyzer/
├── backend/main.py       # FastAPI backend (predictions, explanations, file serving)
├── frontend/             # React + TypeScript + TailwindCSS frontend
│   └── src/
│       ├── pages/        # Home, Predict, Results, Explain, About
│       └── components/   # UI components including FileViewer
├── src/
│   ├── scrapers/         # NLB and DLB web scraping scripts
│   └── preprocessing/    # Data validation, cleaning, feature engineering, splitting
├── notebooks/            # Jupyter notebooks (run on Google Colab)
│   ├── 01_baseline_models_colab.ipynb
│   ├── 02_catboost_training_colab.ipynb
│   ├── 03_hyperparameter_tuning_colab.ipynb
│   ├── 04_shap_analysis_colab.ipynb
│   └── 05_lime_analysis_colab.ipynb
├── data/
│   ├── raw/              # Original scraped CSV files (17 lotteries)
│   ├── processed/        # Feature-engineered data
│   └── splits/           # Train/val/test splits (stratified)
├── models/best_model.cbm     # Trained CatBoost model
├── outputs/
│   ├── statistics/       # data_quality_stats.json, split_stats.json
│   ├── results/          # Model results, baseline comparison, tuning results
│   └── explainability/   # SHAP and LIME outputs (plots, reports)
└── docs/                 # Documentation
```

Appendix B: How to Run

Prerequisites: Python 3.9+, Node.js 18+

Backend:

```
cd lottery_analyzer
python -m venv lottery_env
lottery_env\Scripts\activate      # Windows
pip install -r requirements.txt
python backend/main.py
# API runs at http://localhost:8000
```

Frontend:

```
cd frontend
npm install
npm run dev
# App runs at http://localhost:5173
```

Appendix C: Output Files Reference

File	Location	Description
data_quality_stats.json	outputs/statistics/	Draw-level statistics (8,085 draws)
split_stats.json	outputs/statistics/	ML record statistics (485,094 records)
baseline_results.json	outputs/results/	Logistic Regression & Random Forest metrics
catboost_results.json	outputs/results/	Default CatBoost metrics
best_model_config.json	outputs/results/	Tuned CatBoost configuration and metrics
tuning_improvement.json	outputs/results/	Hyperparameter tuning improvement (1.51%)
shap_analysis_report.json	outputs/explainability/shap/	SHAP analysis metadata and top features
lime_analysis_report.json	outputs/explainability/lime/	LIME analysis metadata
shap_summary_plot.png	outputs/explainability/shap/	SHAP feature importance visualization

File	Location	Description
shap_bar_plot.png	outputs/explainability/shap/	SHAP mean absolute values bar chart
shap_dependence_plots.png	outputs/explainability/shap/	Feature dependence visualizations
lime_shap_comparison.png	outputs/explainability/lime/	Cross-method validation chart
lime_positive_*.png	outputs/explainability/lime/	LIME explanations for positive predictions
lime_negative_*.png	outputs/explainability/lime/	LIME explanations for negative predictions
baseline_comparison.png	outputs/results/	Model comparison bar chart
hyperparameter_heatmaps.png	outputs/results/	Grid search results visualization

Appendix D: Figure Placeholders

Figure	Description	File Location
Figure 1	Class Distribution	Create from split_stats.json
Figure 2	Preprocessing Pipeline	Create diagram
Figure 3	Baseline Comparison	outputs/results/baseline_comparison.png
Figure 4	Algorithm Comparison	From notebook Outputs
Figure 5	Hyperparameter Heatmaps	outputs/results/hyperparameter_heatmaps.png
Figure 6	Top 10 Configurations	outputs/results/top_10_configs.png
Figure 7	SHAP Summary Plot	outputs/explainability/shap/shap_summary_plot.png
Figure 8	SHAP Bar Plot	outputs/explainability/shap/shap_bar_plot.png
Figure 9	SHAP Dependence	outputs/explainability/shap/shap_dependence_plots.png

Figure	Description	File Location
Figure 10	SHAP Waterfall Plot	outputs/explainability/shap/shap_waterfall_plots.png
Figure 11	SHAP Force Plot-Positive	outputs/explainability/shap/shap_force plot-positive.png
Figure 12	SHAP Force Plot-Negative	outputs/explainability/shap/shap_force plot-negative.png
Figure 13	LIME Positive	outputs/explainability/lime/lime_positive_1.png
Figure 14	LIME Negative	outputs/explainability/lime/lime_negative_1.png
Figure 15	SHAP vs LIME	outputs/explainability/lime/lime_shap_comparison.png

Appendix E: Dataset and Code Access

GitHub Repository: *HTTPS://GITHUB.COM/PATHMIKAW/LOTTERY_ANALYZER*

Web Scraped Dataset: *Lottery Analyzer – Dataset(Github)*

Google Colab Notebooks: *HTTPS://DRIVE.GOOGLE.COM/DRIVE*

The complete source code, datasets, trained models, and documentation are available at the GitHub repository above.

This report was prepared for the MSc AI - Applied Machine Learning Assignment.