CRICKET PREDICTION API

PRESENTED BY

Salva Nadeem

Objective

This report documents the complete machine learning pipeline developed for **predicting T20 cricket match outcomes**. The project encompasses exploratory data analysis, feature engineering, model training and evaluation, API development, and integration with an LLM for explainable AI.

1. Exploratory Data Analysis

1.1 Dataset Overview

The dataset contains T20 cricket match data with the following structure:

- Dataset shape: (15691, 5) after cleaning
- Features: total_runs, wickets, target, balls_left, won (target variable)

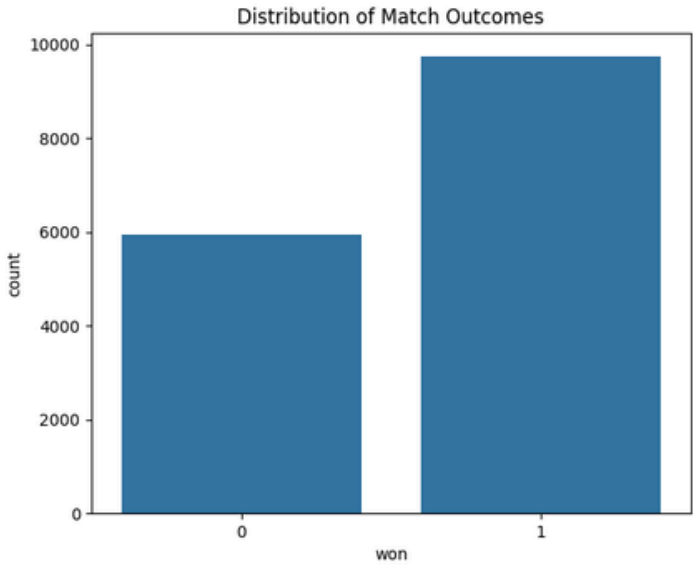
```
Dataset shape: (15691, 5)

First few rows:
   total_runs  wickets  target  balls_left  won
0         0.0      0.0     125       119.0    1
1         0.0      0.0     125       118.0    1
2         1.0      0.0     125       117.0    1
3         1.0      1.0     125       116.0    1
4         1.0      1.0     125       115.0    1
```

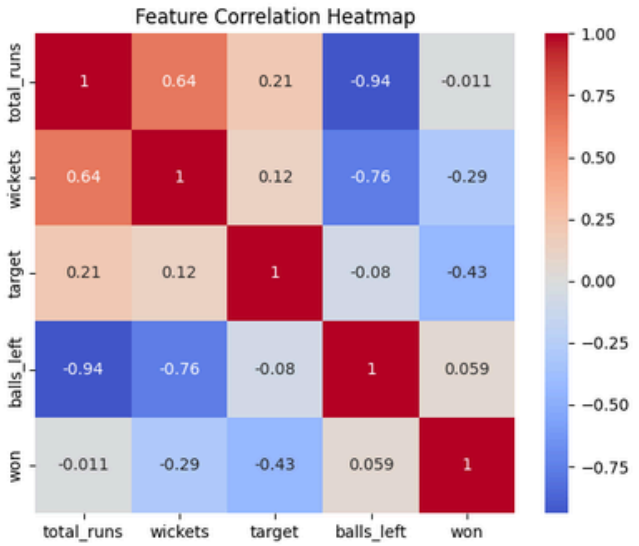
```
Descriptive statistics:
         total_runs       wickets        target    balls_left           won
count  15689.000000  15689.000000  15691.000000  15689.000000  15691.000000
mean      71.395691      2.538530    156.716462     63.766652      0.621184
std       45.430853      2.153691     28.713984     33.006166      0.485108
min        0.000000      0.000000     59.000000     -3.000000      0.000000
25%       34.000000      1.000000    136.000000     36.000000      0.000000
50%       68.000000      2.000000    156.000000     65.000000      1.000000
75%      105.000000      4.000000    180.000000     92.000000      1.000000
max      203.000000     10.000000    238.000000    119.000000      1.000000

Missing values:
total_runs    2
wickets       2
target        0
balls_left    2
won           0
dtype: int64

Correlation matrix:
             total_runs   wickets    target  balls_left       won
total_runs     1.000000  0.638929  0.210841   -0.938889 -0.010967
wickets        0.638929  1.000000  0.121617   -0.761451 -0.291894
target         0.210841  0.121617  1.000000   -0.079779 -0.432581
balls_left    -0.938889 -0.761451 -0.079779    1.000000  0.058960
won           -0.010967 -0.291894 -0.432581    0.058960  1.000000
```

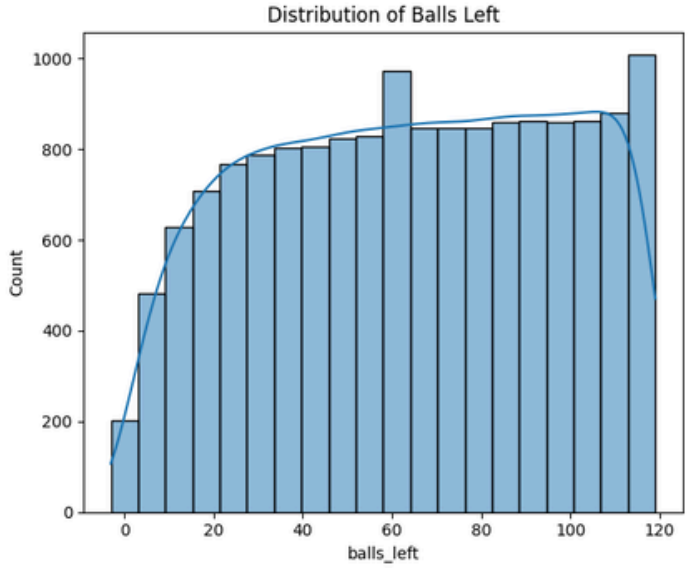
The dataset shows that around **62% of matches are wins.** Correlation analysis highlights that higher targets (-0.43) and more wickets lost (-0.29) significantly reduce win probability, while balls_left alone has very weak predictive power. Overall, the data captures cricket dynamics well, but feature engineering and cleaning are essential to improve model performance and interpretability.

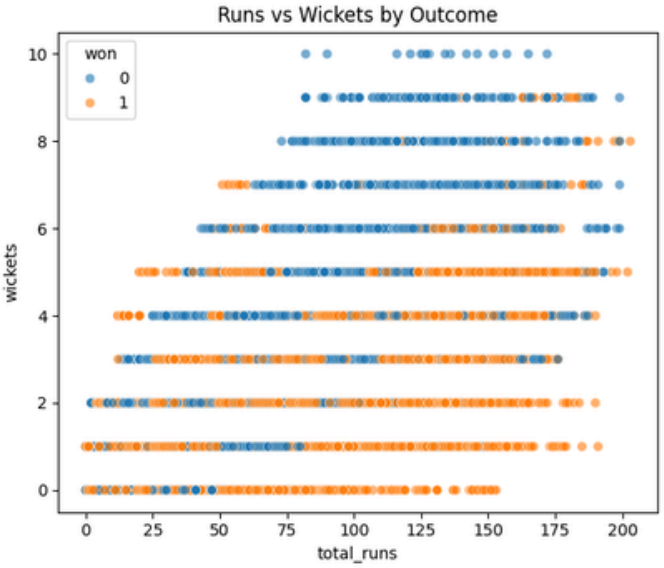
After cleaning, the dataset contains 15,680 matches, with a class distribution of ~62% wins and ~38% losses. Correlation analysis revealed strong multicollinearity between total_runs and balls_left, highlighting the need for engineered features (e.g., run rate, required run rate) for better model performance.

Data Visualization





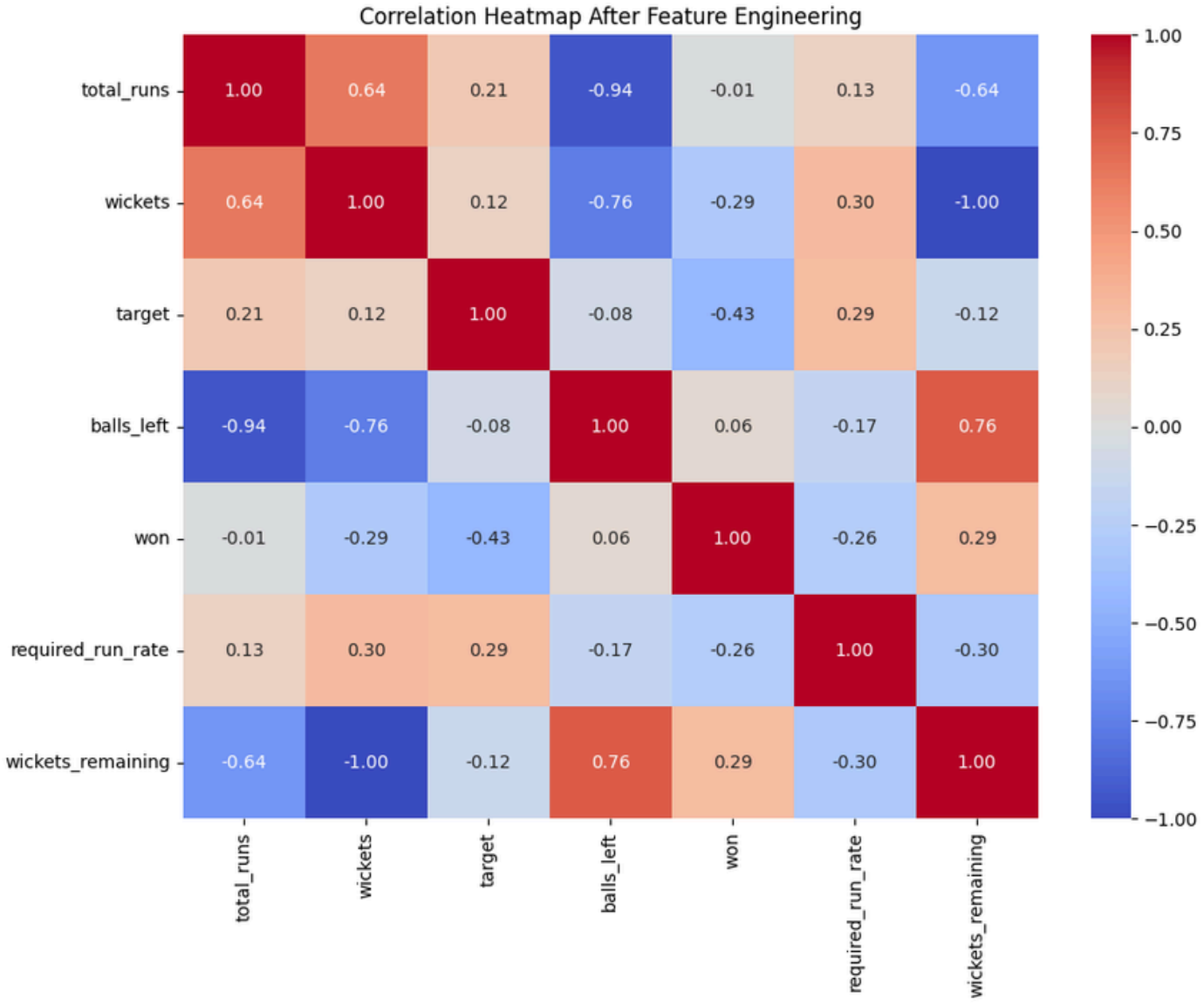




Feature Engineering

Two new features were engineered to capture cricket-specific match dynamics:

1. Required Run Rate (RRR): Calculated as remaining runs divided by overs left.
    - Correlation with winning: -0.28 → Higher RRR reduces the chance of winning, as expected.
2. Wickets Remaining: Derived as 10 – wickets.
    - Correlation with winning: +0.29 → More wickets in hand increases the chance of winning.



Model Training & Comparison

Three different algorithms were selected for comparison:

1. **Logistic Regression**: Baseline model for binary classification
2. **Random Forest**: Ensemble method handling non-linear relationships
3. **XGBoost**: Gradient boosting known for high performance on structured data

Training Methodology

- Train-Test Split: 80-20 split to maintain class distribution
- Cross-Validation: 5-fold cross-validation to ensure robust performance estimates
- Evaluation Metrics: Focused on **F1-score** as the primary metric due to unbalanced classes

```
=== Model Comparison ===
Logistic Regression:
  CV F1: 0.8313
  Test F1: 0.8275
  Test Accuracy: 0.7790
  Test ROC AUC: 0.8556

Random Forest:
  CV F1: 0.9717
  Test F1: 0.9778
  Test Accuracy: 0.9723
  Test ROC AUC: 0.9957

XGBoost:
  CV F1: 0.9724
  Test F1: 0.9749
  Test Accuracy: 0.9688
  Test ROC AUC: 0.9972


Best model based on Test F1 Score: Random Forest
Saved Random Forest as trained_model.pkl
```

Production API Development

FastAPI Implementation:

A production-ready API was developed with the following endpoints:

1. POST /predict: Accepts CSV uploads and returns predictions
2. POST /explain/{prediction_id}: Provides human-readable explanations for predictions
3. GET /health: Health check endpoint
4. GET /predictions/{filename}: Allows downloading prediction results

Key Features

- **Input Validation**: Comprehensive validation for CSV structure and content
- **Filtering Logic**: Processes only rows where balls_left < 60 and target > 120
- **Error Handling**: Robust error handling with meaningful error messages
- **Logging:** Comprehensive logging for debugging and monitoring
- **Model Versioning**: Designed with model versioning considerations for future updates

Prompt Engineering Integration

The **/explain endpoint** integrates with Groq's API to provide natural language explanations of model predictions.

The prompt template was carefully engineered to:

- Include all relevant match situation context
- Request concise, insightful explanations
- Consider different scenarios (high vs. low confidence predictions)
- Provide cricket-specific analysis

Test Coverage:

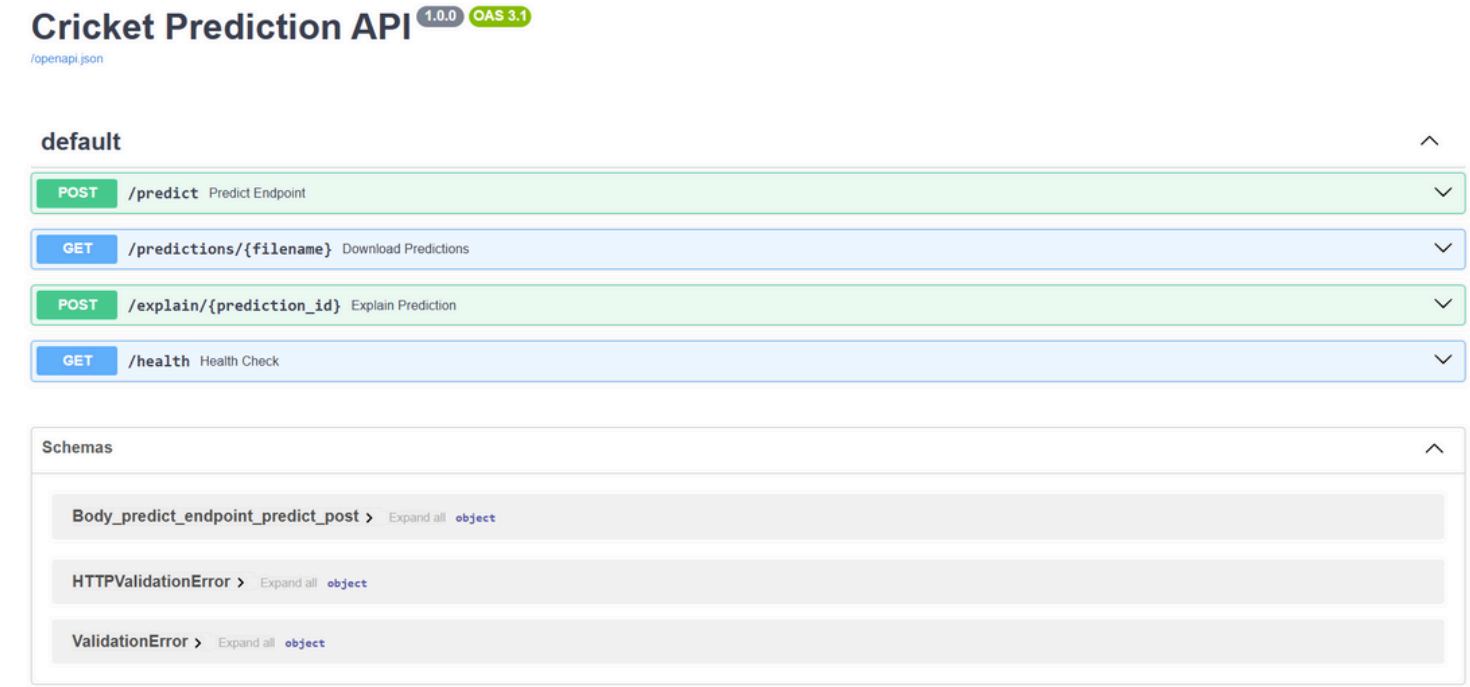
The test suite covers:

- API endpoint functionality
- Error handling for malformed inputs
- Model prediction consistency
- Explanation generation reliability

Limitations:

1. Feature Limitations: Does not incorporate team strength or player form
2. API Constraints: Currently supports only CSV file inputs

Swagger UI



Predict Endpoint

Explain Endpoint

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/explain/1' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{"prediction": 1,
    "confidence": 0.78,
    "total_runs": 45,
    "wickets": 2,
    "target": 128,
    "balls_left": 99}'
```

Request URL

```
http://127.0.0.1:8000/explain/1
```

Server response

Code | Details

200

Response body


```
{
  "prediction": 1,
  "confidence": 0.78,
  "explanation": "Having scored 45 runs in just 21 balls, the side is cruising at a 12.9 RPO, well above the modest 5.0 RPO needed to chase the remaining 83 runs from 99 balls. With eight wickets in hand and more than 16 overs left, they have both the firepower and the time to accelerate. Historically, teams in a similar situation—requiring under 6 RPO with 8+ wickets and >15 overs remaining—win around 85-90 % of the time. The model therefore assigns a 78 % win probability, reflecting the strong but not absolute advantage."
}
```


Download

Response headers

Unit Testing

```
PS D:\cricket_api> python tests/test_unit.py
Prediction: 1
Confidence: 1.0
Explanation: The chase sits at 92/2 with 59 balls left, so 36 runs are needed at just about 3.6 runs per over – well below the 8-9 runs per over the side has already been scoring. With essentially all ten wickets in hand and almost a full quota of overs, the situation mirrors countless T20 chases where teams have a >95 % win rate. Historical data shows that when a side needs under 40 runs with >8 overs and >5 wickets, they win more than nine times out of ten. Hence the model confidently predicts a win with 1.00 confidence.
```

Prediction File

	total_runs	wickets	target	balls_left	won	prediction	prediction_confidence
2	82	2	125	59	1	1	1
3	82	2	125	58	1	1	1
4	84	2	125	57	1	1	1
5	86	2	125	56	1	1	1
6	87	2	125	55	1	1	1
7	91	2	125	54	1	1	1
8	92	2	125	53	1	1	1
9	93	2	125	52	1	1	1
10	93	2	125	51	1	1	1
11	94	2	125	50	1	1	1
12	95	2	125	49	1	1	1
13	99	2	125	48	1	1	1
14	100	2	125	47	1	1	1
15	100	2	125	46	1	1	1