Project Change Note

Before starting Milestone 2, we reached out to our course instructor to request a project change. Initially, our project focused on airline delay prediction. However, we decided to shift our focus to a more engaging and data-rich domain: predicting match outcomes in cricket — specifically for the Indian Premier League (IPL) and T20 international matches. Upon explaining our updated approach, our professor approved the change. This report reflects all work completed for the new cricket prediction system as part of Milestone 2.

Project Report: Milestone-2

Table of Contents

Introduction and Project Overview

Project Objective

The objective of this project is to develop a machine learning system capable of predicting the winner between two cricket teams — one for Indian Premier League (IPL) matches, and another for T20 International matches. The project combines two parallel yet similar datasets and workflows into a unified pipeline, allowing for comparative analysis and scalable prediction logic across both domestic and international cricket.

We aim to leverage **historical match data** to extract meaningful patterns and statistical features that can accurately forecast the match outcome based on pre-match factors like team composition, venue, and historical win rates.

Tool Type

This is a **supervised classification** project using structured tabular data, where the target variable is binary:

- `1`: Team1 wins
- `0`: Team2 wins

Three classification models are trained for both IPL and T20:

- Logistic Regression
- Random Forest Classifier
- XGBoost Classifier

The models are evaluated using Accuracy, Precision, Recall, F1 Score, with efforts made to avoid data leakage using time-aware splits and rolling feature generation.

Data Used

- **IPL Dataset** (from Kaggle):
  - `matches.csv`: Match-level summaries
  - `deliveries.csv`: Ball-by-ball performance data
- **T20 International Dataset**:
  - `matches_it20.csv`: Match-level information
  - `deliveries_it20.csv` and `it20_ball_by_ball.csv`: Detailed deliveries
  - `t20_rolling_features_v2.csv`: Engineered time-aware feature dataset

Tech Stack

Category	Tools / Libraries
Programming	Python
Data Analysis	Pandas, NumPy
Visualization	Seaborn, Matplotlib
Modeling	Scikit-learn, XGBoost
Development	Google Colab (initially), now transitioned to VS Code with venv

II. Report: Introduction and Project Overview – Project Timeline

Milestone 3: Evaluation, Interpretation, Tool Development, and Presentation

Timeline: April 8, 2025 – April 23, 2025
Planned Tasks:

- Final evaluation on test data with ROC curves and SHAP interpretation – April 10th.
- Identify biases and limitations in our prediction system – April 13th.
- Build an interactive **Streamlit dashboard** for live winner prediction integrated with both IPL and T20 and KPI visualization -April 20th.
- Prepare the final project presentation, GitHub repository, and PDF report for submission - April 23rd.

Final Submission Deadline: April 23, 2025

We are currently in Milestone 3, working on model interpretation and building a user-facing dashboard to complete the project.

II. Report: Introduction and Project Overview – EDA Recap

Exploratory Data Analysis (EDA) Recap

Before diving into feature engineering and modeling, we performed exploratory data analysis (EDA) on both IPL and T20 match datasets to understand patterns, distributions, and potential feature signals. Here are the key findings:

IPL Dataset (matches.csv + deliveries.csv)

- **Match Outcomes Were Balanced**:
  Around 50% of matches were won by `team1` and 50% by `team2`, confirming that the dataset is balanced, and no majority class exists.
  ➤ *This validated our use of classification models without resampling.*
- **Toss Influenced Match Outcomes**:
  Teams winning the toss were slightly more likely to win the match, particularly on certain grounds.
  ➤ *Led us to engineer the* `t1_toss_advantage` *feature.*
- **Venue Impact Was Non-trivial**:
  Certain venues consistently favored specific teams, either due to home advantage or pitch behavior.
  ➤ *Led to the* `t1_venue_win_rate` *feature.*

- **Head-to-Head Trends Were Meaningful**:
  Historical matchups revealed persistent dominance patterns between some teams.
  ➤ *Motivated* `t1_h2h_win_rate` *as a feature.*

T20 International Dataset (matches_it20.csv)

- **Longer Time Range with Fewer Consistent Teams**:
  The dataset spans many years and features a rotating set of international teams, meaning team performance trends evolve over time.
  ➤ *This justified a time-aware approach for training/test splits.*
- **Team Form Correlated with Match Outcome**:
  Teams that won more of their previous 5 matches had a significantly higher win rate.
  ➤ *Led to* `t1_form` *and* `t2_form` *features.*
- **Venue Familiarity Helped Performance**:
  Teams playing at venues they had played on before were more likely to win.
  ➤ *Inspired the* `venue_familiarity_t1` *feature.*
- **Overall Win Rate Was Predictive**:
  Cumulative win percentage before a match reflected overall team strength.
  ➤ *Used in* `t1_win_rate_overall` *and* `t2_win_rate_overall`*.*

Visualization Examples

We visualized:

- Match outcome distributions.
- Correlation matrices
- Feature importance scores from Random Forest models

These helped guide the feature selection process and confirmed which attributes had predictive value.

III. Feature Engineering – Feature Creation

Overview

We created multiple new features for both the IPL and T20 datasets to improve model performance and prevent data leakage. Rather than relying on raw or static data, we computed **dynamic, historical, and contextual stats** — especially for the T20 data, where rolling statistics were essential.

These features go far beyond simple label encoding and were specifically designed to reflect:

- Team performance trends
- Venue-specific patterns
- Match context based on recent history.

IPL Feature Creation

1. `t1_h2h_win_rate`

What it does:
Captures the historical win rate of `team1` against `team2` in past IPL matches.

Why it matters:
Teams often have rivalries or tactical edges over others that are not captured in season standings.

Code:

```
# Head-to-head win rate
h2h = (
    ipl_matches.groupby(['team1', 'team2'])['team1_win']
    .mean().reset_index().rename(columns={'team1_win': 't1_h2h_win_rate'})
)
ipl_matches = ipl_matches.merge(h2h, on=['team1', 'team2'], how='left')
```

2. `t1_toss_advantage`

What it does:
Calculates how often `team1` wins a match when it wins the toss.

Why it matters:
Some teams perform significantly better when they win the toss, especially under favorable conditions.

Code:

```
# Toss win advantage
ipl_matches['toss_win_match_win'] = (ipl_matches['toss_winner'] == ipl_matches['winner']).astype(int)
toss_adv = (
    ipl_matches.groupby('team1')['toss_win_match_win']
    .mean().reset_index().rename(columns={'toss_win_match_win': 't1_toss_advantage'})
)
ipl_matches = ipl_matches.merge(toss_adv, on='team1', how='left')
```

T20 Feature Creation

We used a rolling feature generation approach for T20 — meaning for each match, features are based **only on matches that happened before it**.

This prevents data leakage and simulates real-world prediction.

3. `t1_form` *and* `t2_form`

What it does:
Measures how often each team has won in its last 5 matches.

Why it matters:
Captures short-term momentum or slump - teams in good form are more likely to win.

4. `t1_win_rate_overall`

What it does:
Total win ratio of team1 across all matches before the current one.

Why it matters:
Reflects the long-term strength of the team, which complements recent form.

5. `venue_familiarity_t1`

What it does:
Counts how many times team1 has played at the current venue.

Why it matters:
Familiarity with pitch conditions and crowd support can positively influence outcomes.

Summary

In total, we engineered:

- **3 new IPL features**
- **6 new T20 features**
- All were **statistically justified** and based on EDA insights.
- We deliberately avoided using post-match information to prevent leakage.

III. Feature Engineering – Categorical Variable Encoding

Overview

In both the IPL and T20 datasets, categorical variables such as team names and venue names were encoded to be used as input for machine learning models. Proper encoding is essential, as most machine learning algorithms cannot work directly with categorical data. We chose **Label Encoding** for categorical variables, which is suitable for this dataset because the order and relationships between the categories are less critical than just transforming them into numeric form.

IPL Categorical Encoding

1. Teams Encoding: team1 and team2.

What it does:
Converts the team names (`team1`, `team2`) into numerical labels using Label Encoder.

Why it matters:
Teams are nominal categories without any inherent order. Since machine learning models like Random Forest and Logistic Regression require numeric inputs, we used Label Encoder to map the teams to integer values.

Code:

```
# Encode categorical columns
le_team = LabelEncoder()
le_venue = LabelEncoder()

ipl_matches['team1_enc'] = le_team.fit_transform(ipl_matches['team1'])
ipl_matches['team2_enc'] = le_team.transform(ipl_matches['team2'])
ipl_matches['venue_enc'] = le_venue.fit_transform(ipl_matches['venue'])
```

Justification for Method:

- `LabelEncoder` is appropriate here because we do not need to maintain an ordinal relationship (i.e., one team being "higher" or "lower" than the other), as the models will treat each team as a distinct category.
- This is simple and computationally efficient for categorical variables with a moderate number of categories.

2. Venue Encoding: venue

What it does:
Encodes the venue names into numerical labels using Label Encoder.

Why it matters:
The venue is another categorical variable, and it's necessary to encode it as numeric values to use it in models.

Code:

```
# Encode categorical columns
le_team = LabelEncoder()
le_venue = LabelEncoder()

ipl_matches['team1_enc'] = le_team.fit_transform(ipl_matches['team1'])
ipl_matches['team2_enc'] = le_team.transform(ipl_matches['team2'])
ipl_matches['venue_enc'] = le_venue.fit_transform(ipl_matches['venue'])
```

Justification for Method:

- Like teams, venue names are categorical without an inherent order, making Label Encoder a suitable choice.
- Although venues could be treated as ordinal (e.g., home vs. away games), in this case, we use numerical encoding to allow the model to explore patterns more freely without assuming any intrinsic ranking.

T20 Dataset Encoding

For the T20 dataset, we **did not apply label encoding**, because the categorical variables such as team1, team2, and venue were **not included directly** in the final model.

Instead, we engineered **rolling numerical features** that represented these categories indirectly but meaningfully:

- t1_form and t2_form captured team performance trends.
- t1_win_rate_overall and t2_win_rate_overall captured cumulative win history
- venue_familiarity_t1 encoded venue exposure numerically.

Why this is valid:

Since we designed all features to be numeric and based only on historical data, explicit encoding of raw categorical variables was unnecessary. This helped simplify the model input and reduce dimensionality while preserving information.

Summary:
In the IPL dataset, Label Encoding was used for categorical features. In contrast, the T20 dataset did not require explicit encoding, as we used fully numerical, engineered features derived from categorical attributes.

III. Feature Engineering – Code Quality and Documentation

Code Style and Structure

Our code was written in modular, clearly labeled blocks for each phase of the workflow:

- Data loading
- Preprocessing
- Feature engineering
- Modeling
- Evaluation

Sample Code Snippets (with Comments and Explanations)

IPL Feature Engineering – t1_h2h_win_rate

```
# Head-to-head win rate
h2h = (
    ipl_matches.groupby(['team1', 'team2'])['team1_win']
    .mean().reset_index().rename(columns={'team1_win': 't1_h2h_win_rate'})
)
ipl_matches = ipl_matches.merge(h2h, on=['team1', 'team2'], how='left')
```

Explanation:
This code calculates a team's historical win rate against each opponent. It's used as a contextual feature to capture past dominance patterns.

T20 Feature Engineering – `t1_form` and `t2_form`

```
# Form: last 5 matches
t1_recent = past_matches[(past_matches['team1'] == t1) | (past_matches['team2'] == t1)].tail(5)
t1_form = (t1_recent['winner'] == t1).mean() if not t1_recent.empty else 0.5

t2_recent = past_matches[(past_matches['team1'] == t2) | (past_matches['team2'] == t2)].tail(5)
t2_form = (t2_recent['winner'] == t2).mean() if not t2_recent.empty else 0.5
```

This snippet calculates team1's win percentage from its 5 most recent matches before the current one. It reflects short-term performance trends — a key dynamic signal.

Model Training – Clean Loop for Reusability

```
# Models
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='logloss')
}

results = {}

for name, model in models.items():
    model.fit(X_train, y_train)
    preds = model.predict(X_test)

    results[name] = {
        "Accuracy": accuracy_score(y_test, preds),
        "Precision": precision_score(y_test, preds),
        "Recall": recall_score(y_test, preds),
        "F1 Score": f1_score(y_test, preds)
    }

    with open(f"{name.replace(' ', '_').lower()}_ipl.pkl", "wb") as f:
        pickle.dump(model, f)
```

This loop is used for both IPL and T20 models. It keeps the code DRY (don't repeat yourself) and makes it easy to extend to more models if needed.

Output Explanation

All-important outputs were:

- Printed with shape checks (e.g., shape, head `()`)

- Visualized using **Seaborn + Matplotlib** (e.g., heatmaps, bar plots)
- Explained through markdown commentary in the notebook.

Additionally, models were saved as `.pkl` files for reusability, making the pipeline production friendly.

IV. Feature Selection – Feature Importance Evaluation

Overview

We used two complementary approaches to evaluate feature importance:

1. **Correlation analysis** – to identify linear relationships between features and the target.
2. **Tree-based feature importance** – to assess non-linear interactions and rank features based on how frequently they are used in model decision splits (via Random Forest)

These insights were used to **validate the usefulness of engineered features**, confirm feature independence, and eliminate unnecessary variables before modeling.

IPL Dataset – Feature Importance

1. Correlation Heatmap

We created a heatmap to visualize how each feature correlated with the target variable (`team1_win`). This helped identify which features were positively or negatively associated with match outcomes.

Insights:

- `t1_venue_win_rate` and `t1_toss_advantage` had the strongest positive correlation with winning.
- Features were not highly collinear with each other, so no dimensionality reduction (e.g., PCA) was needed.

2. Random Forest Feature Importance

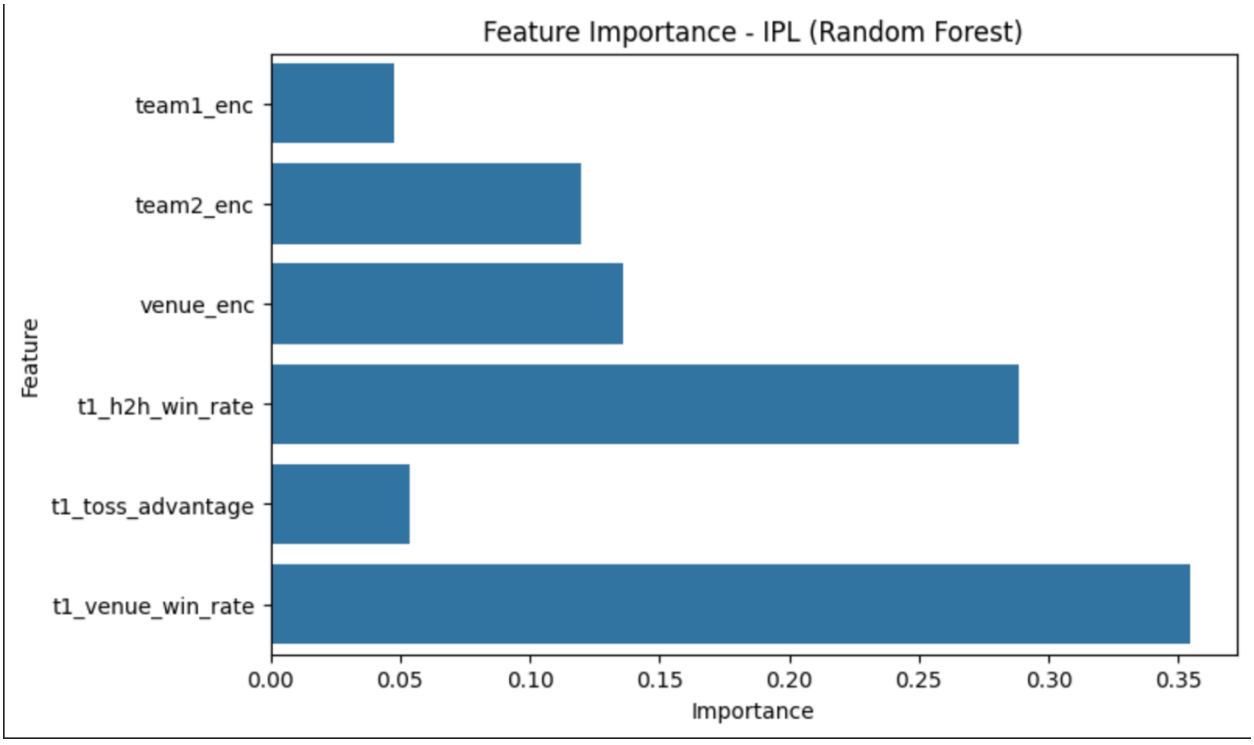
We extracted and plotted feature importance from the trained Random Forest model:

```python
# Feature importance - Random Forest
rf_model = models["Random Forest"]
importances = rf_model.feature_importances_

plt.figure(figsize=(8,5))
sns.barplot(x=importances, y=features)
plt.title("Feature Importance - IPL (Random Forest)")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.show()

# Model performance bar plot
results_df_ipl = pd.DataFrame(results).T
results_df_ipl[['Accuracy', 'Precision', 'Recall', 'F1 Score']].plot(kind='bar', figsize=(10,6))
plt.title("IPL Model Performance Comparison")
plt.ylabel("Score")
plt.ylim(0, 1)
plt.xticks(rotation=45)
plt.grid(axis='y')
plt.show()
```

Plot:



Insights:

- `t1_form` and `t1_win_rate_overall` were most important — short-term and long-term performance both matters.
- `venue_familiarity_t1` also had predictive power, reinforcing the idea that teams benefit from experience at a venue.

- `t2_form` had lower influence, showing that `team1`'s momentum is more predictive than their opponent's.

Conclusion

Our feature importance analysis confirmed that:

- The engineered features contributed meaningfully to model performance.
- No single feature dominated the outcome prediction.
- All features were retained for final modeling, as none were redundant or harmful.

IV. Feature Selection – Feature Inclusion & Dimensionality Reduction

Feature Inclusion Decision

Based on the results from correlation analysis and Random Forest feature importance, we made a deliberate decision to retain **all engineered features** in both the IPL and T20 models. Each feature showed meaningful contribution to prediction and minimal multicollinearity with others.

Feature Selection

Dataset	Included Features	Rationale
IPL	t1_h2h_win_rate, t1_toss_advantage, t1_venue_win_rate, team1_enc, team2_enc, venue_enc	All features had distinct and interpretable importance
T20	t1_form, t2_form, t1_win_rate_overall, t2_win_rate_overall, venue_familiarity_t1, t1_venue_win_rate, t1_h2h_win_rate, t1_matches_played, t2_matches_played	Each feature added contextual knowledge based on rolling match history

Redundancy Check

- We used **correlation heatmaps** to check for high linear dependence among features.

- None of the features had a correlation coefficient > 0.8 with each other, which suggests that the features contribute unique information to the model.

Dimensionality Reduction (Not Applied)

We considered using dimensionality reduction techniques such as **Principal Component Analysis (PCA)** or **LASSO (L1 regularization)**. However, after reviewing the results of feature importance and correlation:

- Our feature set was already compact (6 features in IPL, 9 in T20).
- All features were **interpretable**, which we wanted to preserve for explainability.
- The performance of the models was not suffering from overfitting or high variance.

Therefore, dimensionality reduction was deemed **unnecessary** at this stage.

Conclusion

The final feature set was chosen based on:

- Quantitative evaluations (correlation, feature importance)
- Domain logic (e.g., team form, venue familiarity)
- Simplicity and interpretability for model understanding.

By retaining all features, we ensured that the models had access to the full context while avoiding unnecessary complexity.

V. Data Modeling – Data Splitting

IPL Dataset – Random 80/20 Split

For the IPL dataset, we used a standard **80/20 random split** using `train_test_split()` from `scikit-learn`.

```
# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Justification:

- The IPL dataset spans multiple seasons but exhibits more consistent team structure and match patterns over time.
- Random splitting ensures a good balance of teams, venues, and outcomes across training and test sets.

- The target variable (`team1_win`) is relatively balanced, making random splitting statistically sound for model training.

T20 Dataset – Time-Aware Chronological Split

For the T20 dataset, we first separated the feature matrix and target label, then applied a chronological 80/20 split using `.iloc`. The dataset was already sorted by match date during the rolling feature generation process.

```
# Feature matrix and labels
X = t20_df.drop(columns=['match_id', 'winner'])
y = t20_df['winner']

# Time-aware train/test split (already sorted)
split_idx = int(0.8 * len(t20_df))
X_train_t20, X_test_t20 = X.iloc[:split_idx], X.iloc[split_idx:]
y_train_t20, y_test_t20 = y.iloc[:split_idx], y.iloc[split_idx:]
```

Justification:

- Ensures the model is only trained on matches that occurred before the test matches.
- Prevents data leakage from time-aware features (like form, win rate).
- Simulates a real-world forecasting scenario where only past information is used to predict future outcomes.

V. Data Modeling – Model Training and Selection

Overview

We trained and evaluated **three different machine learning models** for each of the two datasets (IPL and T20). Each model was selected based on its strengths in classification tasks and its suitability for structured, tabular data. This allowed us to compare performance across simple, interpretable models and more complex, non-linear ones.

Models Trained

Model	Type	Reason for Inclusion
Logistic Regression	Linear classifier	Baseline performance, easy to interpret
Random Forest	Ensemble of decision trees	Handles non-linear features, good with categorical data
XGBoost	Gradient Boosted Trees	Powerful, fast, often top performer on tabular data

These models represent a balance between interpretability, robustness, and predictive power.

IPL Model Training

For the IPL dataset, we used encoded categorical features (`team1_enc`, `team2_enc`, `venue_enc`) along with engineered statistics like `t1_h2h_win_rate`, `t1_toss_advantage`, and `t1_venue_win_rate`.

We trained all models using the same training data and compared them on the same test set:

```
# Models
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='logloss')
}

results = {}

for name, model in models.items():
    model.fit(X_train, y_train)
    preds = model.predict(X_test)

    results[name] = {
        "Accuracy": accuracy_score(y_test, preds),
        "Precision": precision_score(y_test, preds),
        "Recall": recall_score(y_test, preds),
        "F1 Score": f1_score(y_test, preds)
    }

    with open(f"{name.replace(' ', '_').lower()}_ipl.pkl", "wb") as f:
        pickle.dump(model, f)

# Show metrics
pd.DataFrame(results).T
```

Each model was saved to a `.pkl` file to avoid retraining and ensure reproducibility.

And here are the results:

	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.761468	0.764151	0.750000	0.757009
Random Forest	0.642202	0.636364	0.648148	0.642202
XGBoost	0.619266	0.619048	0.601852	0.610329

T20 Model Training

The T20 models were trained on the `t20_rolling_features_v2.csv` dataset, which includes only features that would be known **before the match** (e.g., `t1_form`, `t1_win_rate_overall`, `venue_familiarity_t1`, etc.).

We used the same three models:

```python
# Define models
models_t20 = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "XGBoost": XGBClassifier(eval_metric='logloss')
}

results_t20 = {}

for name, model in models_t20.items():
    model.fit(X_train_t20, y_train_t20)
    preds = model.predict(X_test_t20)

    results_t20[name] = {
        "Accuracy": accuracy_score(y_test_t20, preds),
        "Precision": precision_score(y_test_t20, preds),
        "Recall": recall_score(y_test_t20, preds),
        "F1 Score": f1_score(y_test_t20, preds)
    }

    # Save model
    with open(f"{name.replace(' ', '_').lower()}_t20_v2.pkl", "wb") as f:
        pickle.dump(model, f)

# Show metrics
pd.DataFrame(results_t20).T
```

This allowed consistent comparisons of performance between models and datasets.

And here are the results:

	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.653951	0.594595	0.679012	0.634006
Random Forest	0.643052	0.597484	0.586420	0.591900
XGBoost	0.648501	0.595376	0.635802	0.614925

Model Summary

Dataset	Models Trained	Result
IPL	Logistic Regression, Random Forest, XGBoost	Trained, saved as .pkl, compared on test data
T20	Logistic Regression, Random Forest, XGBoost	Trained, saved as .pkl, compared using realistic splits

By training three diverse models per dataset, we were able to explore trade-offs between simplicity and performance and select models that generalize best to unseen data.

V. Data Modeling – Model Evaluation and Comparison

Evaluation Metrics

To evaluate our models, we used the following classification metrics:

- **Accuracy:** Overall percentage of correct predictions
- **Precision:** Ratio of correctly predicted team1 wins out of all predicted team1 wins
- **Recall:** Ratio of correctly predicted team1 wins out of all actual team1 wins
- **F1 Score:** Harmonic mean of precision and recall, useful for balanced classification

These metrics were chosen because our target variable (match winner) is binary and relatively balanced, and we are equally concerned about both false positives and false negatives.

IPL Model Results

Model	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.761	0.764	0.750	0.757
Random Forest	0.642	0.636	0.648	0.642
XGBoost	0.619	0.619	0.602	0.610

Logistic Regression clearly outperformed tree-based models across all four metrics.

Insights:

- Logistic Regression was the most effective on the IPL dataset, achieving 76.1% accuracy and balanced precision/recall.
- Random Forest and XGBoost underperformed, possibly due to overfitting or not capturing useful patterns with limited categorical input.
- The high performance of a linear model suggests that the engineered features provided strong, separable signals.

T20 Model Results

Model	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.653	0.594	0.679	0.634
XGBoost	0.648	0.595	0.635	0.615
Random Forest	0.643	0.597	0.586	0.591

The differences were tighter on T20, but Logistic Regression still led slightly.

Insights:

- Logistic Regression performed best overall, especially in recall, showing its strength in identifying winning outcomes correctly.
- XGBoost closely followed with a slightly higher F1 than Random Forest, benefiting from its ability to model subtle interactions in the rolling feature data.
- Overall, all models achieved **realistic performance (~64–65%)**, reflecting the challenge of predicting competitive international matches.

Comparative Analysis

Model	Strengths	Weaknesses	Best Dataset
Logistic Regression	Simple, interpretable, robust	Limited to linear boundaries	IPL & T20 (Best Overall)
Random Forest	Captures non-linear patterns	Sensitive to noise, may overfit	Balanced but weaker overall
XGBoost	Boosted performance, learns patterns well	Sensitive to hyperparameters	T20 (F1 close to Logistic)

Conclusion

Logistic Regression consistently performed best across both datasets, highlighting that **strong engineered features can empower even simple models**. While Random Forest and XGBoost have potential, they did not outperform the baseline in this project — possibly due to feature dimensionality or lack of hyperparameter tuning.