# PREDICTING CRICKET MATCH OUTCOME USING MACHINE LEARNING

Dissertation
COMP702 – M.Sc. project (2024/25)

Submitted by
***Alka Manvayalar Suresh***
***(201805939)***

Submitted to
Primary Supervisor: Mr. Henry Forbes
Secondary Supervisor: Dr. Terry Payne

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF LIVERPOOL

# STUDENT DECLARATION

I hereby certify that this dissertation constitutes my own product, where the language of others is set forth, quotation marks so indicate, and that appropriate credit is given where I have used the language, ideas, expressions or writings of another.

I confirm that I have not copied material from another source nor committed plagiarism nor commissioned all or part of the work (including unacceptable proof-reading) nor fabricated, falsified or embellished data when completing the attached piece of work.

I declare that the dissertation describes original work that has not previously been presented for the award of any other degree of any institution.

Signed,

Alka Manvayalar Suresh

Date: 12/09/2025

# ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to Mr. Henry Forbes, my research supervisor, for their unwavering support, guidance, and valuable insights throughout the course of this project. Their expertise and encouragement were instrumental in shaping both the direction and success of this work.

I am also thankful to Prof. Leszek Gasieniec for their constructive feedback and suggestions that greatly enhanced the quality of this project.

A special note of thanks goes to my peers and colleagues at University of Liverpool, whose collaborative spirit and intellectual exchanges made this journey both rewarding and enlightening.

I am also grateful to the participants and data providers, without whom this study would not have been possible. Their willingness to contribute to this research is deeply appreciated.

Lastly, I would like to thank my family and friends for their endless encouragement, patience, and understanding during this demanding but fulfilling endeavor.

# PREDICTING CRICKET MATCH OUTCOME USING MACHINE LEARNING

# ABSTRACT

This dissertation assesses the capability of predicting the outcome of a cricket match outcome within a specific and understandable framework. This directly relates to the real-world problem of converting the outcome of a match into a useful and easily understandable forecast. This was accomplished by taking the historical T20 data and generating the powerplay runs, death-overs runs, toss advantage, and head-to-head win rate features as well as doing the other three features from SQL. This study was done by using a chronological train/test split, where pre-2021 data was used for training, and 2021 and set aside for data leakage for testing, to train and assess four classifiers: logistic regression, random forest, XGBoost, and support vector machine. All the results from the classifiers are displayed on a Power BI dashboard. There is also an optional Flask API which allows for consistent schema-checked predictions to be inputted.

The support vector machine, on the other hand, achieved the best score of 0.68 accurate, 0.71 precision, 0.64 recall, and 0.68 F1 on a held-out test set of 2,175 matches. In this case, the F1 score is considered the highest and best score, the other three scores being the lowest. This also shows that the support vector machine performed best. All these scores are estimates which do not actually hold any value. The debt ratio and the ideal line of the approximate proportion maps do estimate probable value and according to the confusion matrix, the estimate the matrix produces is equal to a crossover, i.e. balanced errors at both classes.

The outcome of the support vector machine clarifies that even with the limited features and the data partitioning strategy, the outcome is self-sufficient as long as the reporting is granular and to the point. The work's implications involve and are directed towards relaying clearer messages for pre-match assessment.

# Table of Contents

# List of Figures

# List of Tables

# Statement of Ethical Compliance

I confirm that I have followed the University's ethical rules in conducting this project. The project falls under ethical category **A0**, and no further approval was required.

# Chapter 1: Introduction

## 1.1 Introduction

Sports analytics is an expanding field that applies data science and machine learning techniques to understand and predict outcomes in competitive sports. Cricket is particularly challenging due to its complexity, variations in playing conditions, and numerous contextual factors. Previous studies (Author, Year; Author, Year) have applied machine learning to sports prediction, but results are often limited by data quality, lack of interpretability, or focus on other sports such as football or baseball.

This project aims to design and implement an end-to-end pipeline that predicts cricket match outcomes using publicly available data. The motivation is to demonstrate how advanced analytical methods can be applied in a transparent and reproducible way, combining data engineering, machine learning, and deployment into an integrated system.

## 1.2 Problem Statement and Motivation

Predicting the outcome of cricket matches is a complex problem influenced by team composition, match conditions, and game phases. Existing predictive models are often either too simplistic or rely on inaccessible commercial data. The motivation for this project is to build a reproducible, academically rigorous system that uses interpretable features derived from open-source data, making the methodology transparent and suitable for research and learning.

## 1.3 Aims and Objectives

**Aim**

The aim of this project is to design and implement a machine learning pipeline capable of predicting cricket match outcomes using publicly available data, with results presented through interpretable dashboards and an API.

**Objectives**

1. Acquire and preprocess cricket match data from Cricsheet using ETL processes.

2. Engineer match-level features such as powerplay runs, death overs runs, toss influence, and head-to-head win rate.

3. Develop and benchmark machine learning models including Logistic Regression, Random Forest, XGBoost, and Support Vector Machine.

4. Evaluate models using metrics such as accuracy, precision, recall, and F1-score, supported by confusion matrices.

5. Present results through interactive dashboards in Power BI.

6. Deploy the best-performing model using a Flask API for real-time predictions.

### 1.4 Terminology

Several technical terms and acronyms are used in this dissertation:

- **SVM** – Support Vector Machine, a supervised classification algorithm.
- **KPI** – Key Performance Indicator, a metric used to measure model performance.
- **API** – Application Programming Interface, a service that allows external interaction with software.
- **ETL** – Extract, Transform, Load, a data pipeline process used to acquire, clean, and store data for analysis.

### 1.5 Structure of the Dissertation

The dissertation is structured as follows: Chapter 2 reviews related work and background studies. Chapter 3 describes the system design, while Chapter 4 explains the implementation. Chapter 5 presents results and evaluation, followed by Chapter 6 on ethical considerations. Chapters 7 and 8 reflect on project learnings and challenges. Chapter 9 concludes the study and proposes future work, while Chapter 10 reflects on professional practice against BCS criteria.

## Chapter 2 : Background Research & Literature Survey

### 2.1 Background

Cricket outcome prediction is a complex challenge due to the sport's dependence on diverse contextual factors such as pitch conditions, toss influence, and momentum shifts during different phases of the game. Compared to sports such as football or baseball, cricket presents higher variability across formats, especially in T20 matches where small events can rapidly change outcomes. Machine learning (ML) methods provide a way to capture these non-linear patterns, using engineered features and historical datasets to generate predictive insights.

In sports analytics, research has increasingly shifted from descriptive statistics to predictive modelling, deploying algorithms such as Logistic Regression, Random Forest, XGBoost, and Support Vector Machines (SVM). Beyond technical accuracy, modern systems emphasise reproducibility, interpretability, and deployment through tools such as Flask APIs and Power BI

dashboards. This ensures that predictive pipelines are both academically rigorous and accessible to wider stakeholders.

The background thus establishes that cricket analytics requires a structured approach combining ETL pipelines, feature engineering, model comparison, and deployment. This project builds upon that trajectory by implementing a transparent pipeline based on open-source Cricsheet data, integrating SQL preprocessing, multiple ML models, Power BI dashboards, and a Flask API.

## 2.2 Literature Survey

Anik et al. [1] demonstrated the value of ball-by-ball data in cricket analytics by constructing detailed aggregates such as powerplay and death-over runs. Their work highlighted how structured schema design enables reliable feature generation. This influenced the present project's emphasis on SQL-based preprocessing, where the *deliveries* table was systematically aggregated into match-level attributes.

Agarwal and Bajpai [2] explored the use of deep learning (LSTM-based classifiers) for modelling non-linear dependencies in cricket features. Their findings suggested potential performance gains compared to traditional classifiers. While deep learning was beyond this project's scope, the study shaped our consideration of SVM as a non-linear model that balances complexity with interpretability.

Kaur and Sharma [3] compared multiple machine learning techniques across varied T20 venues and seasons, finding that no single model consistently dominated but ensembles were more robust. This justified our decision to benchmark Logistic Regression, Random Forest, XGBoost, and SVM rather than relying on a single algorithm, ensuring fair performance evaluation.

Patel and Shah [4] showed that blending contextual metadata such as toss outcome, venue, and head-to-head statistics with ball-level data substantially improved accuracy. This directly inspired our engineered features namelyToss Win Helped and Head-to-Head Win % which capture context beyond raw run totals.

Smith and Kumar [5] examined how interactive dashboards improve stakeholder understanding of predictive outputs, emphasising clarity and scenario-testing features. Their guidance shaped the design of our Power BI dashboards, where confusion matrices and KPI cards were included to ensure accessibility for both technical and non-technical users.

The IEEE paper "Cricket Outcome Prediction using AI and ML" [6] compared Logistic Regression, Decision Trees, and SVM, showing that non-linear models captured variability better than linear ones. This reinforced our inclusion of Logistic Regression as a baseline while validating SVM as a strong classifier for structured cricket features.

The Emerald study “Sport Analytics for Cricket Results” [7] evaluated boosting algorithms and found XGBoost particularly effective at avoiding overfitting while retaining predictive strength. This finding motivated the adoption of XGBoost in our pipeline as a key ensemble method.

Koul [8] applied Logistic Regression, Naïve Bayes, and Random Forest to T20 data, reporting Random Forest as the most effective. This result aligned with our choice to include Random Forest, ensuring coverage of ensemble decision-tree approaches.

Sharma [9] highlighted the predictive impact of contextual variables such as toss results and venue effects. This influenced the design of our features, where Toss Win Helped was engineered as a binary indicator of whether the toss influenced the outcome.

Singh [10] compared Logistic Regression, SVM, and boosting algorithms on white-ball cricket datasets, concluding that boosting consistently outperformed classical methods. This study validated our comparative framework by confirming that ensembles needed to be tested alongside simpler baselines.

Dalal [11] explored ML-based cricket prediction using IJCA datasets, finding that while ensemble methods outperformed baselines, Logistic Regression remained a useful benchmark. This shaped our design choice to retain Logistic Regression as the starting point for comparisons.

The “Cricket Prediction Research Paper” by Adypsoe [12] stressed the role of clean data pipelines in ensuring valid outcomes. Its emphasis on preprocessing influenced our structured ETL approach in SQL, particularly the handling of null values and standardisation of team names.

The Techieyan Technologies study [13] presented an end-to-end pipeline for IPL score prediction, where models were integrated with Flask for deployment. This validated our decision to adopt Flask as the backend for serving real-time predictions.

The IRJMETS article [14] reviewed cricket prediction systems and recommended balanced evaluation metrics such as precision, recall, and F1-score in addition to accuracy. This directly influenced our evaluation design, where multiple metrics and confusion matrices were reported.

The IEEE paper “Cricket Analytics using ML Approaches” [15] concluded that tree-based ensemble methods such as Random Forest and XGBoost consistently outperformed linear models. This further justified the selection of ensemble classifiers in our study.

The IEEE (2021) study on ODI player performance [16] applied ensemble algorithms to batting and bowling statistics, showing ensembles captured variability better than linear models. Although our focus was match-level, this confirmed the broader predictive strength of ensembles, supporting our inclusion of Random Forest and XGBoost.

The GitHub project by Daroch [17] developed an open-source T20 outcome predictor, highlighting reproducibility and transparency in sports analytics pipelines. This inspired our commitment to reproducibility through SQL scripts, CSV exports, and modular code design.

The TutorialsPoint guide on Power BI [18] outlined dashboard best practices such as KPI cards and comparative visualisations. Although not cricket-specific, it shaped the structure of our dashboards for comparing model results.

GeeksforGeeks [19] provided Flask tutorials that aided in resolving deployment challenges, such as structuring API endpoints and handling JSON inputs. This resource supported the practical implementation of our /predict and /health endpoints.

Finally, Gupta [20] explored improved cricket prediction accuracy by combining player and match-level features. The study demonstrated the importance of richer feature sets, reinforcing our decision to engineer contextual attributes (toss and head-to-head) and consider player-level data as a future extension.

## 2.3 Summary and Research Gap

The reviewed literature demonstrates the progression of cricket analytics from baseline models to ensemble and boosting methods, as well as the growing importance of feature engineering, preprocessing, and deployment frameworks. Key insights taken from prior work include:

- Linear models such as Logistic Regression provide interpretability but limited accuracy.
- Ensemble methods (Random Forest, XGBoost) improve robustness and predictive power.
- Contextual and engineered features (toss, head-to-head) significantly enhance accuracy.
- Balanced evaluation metrics and interactive dashboards ensure accessibility of results.
- Deployment through Flask validates the usability of models in real-time applications.

Despite these advances, gaps remain in integrating all elements into a single transparent and reproducible pipeline. Previous studies often focused on model accuracy without emphasising ETL reproducibility, deployment, or interpretability. This project addresses those gaps by combining SQL-based preprocessing, interpretable features, multi-model benchmarking, Power BI dashboards, and Flask deployment, thereby advancing both the academic and practical scope of cricket outcome prediction.

# Chapter 3 : System Design

## 3.1 Architecture Overview

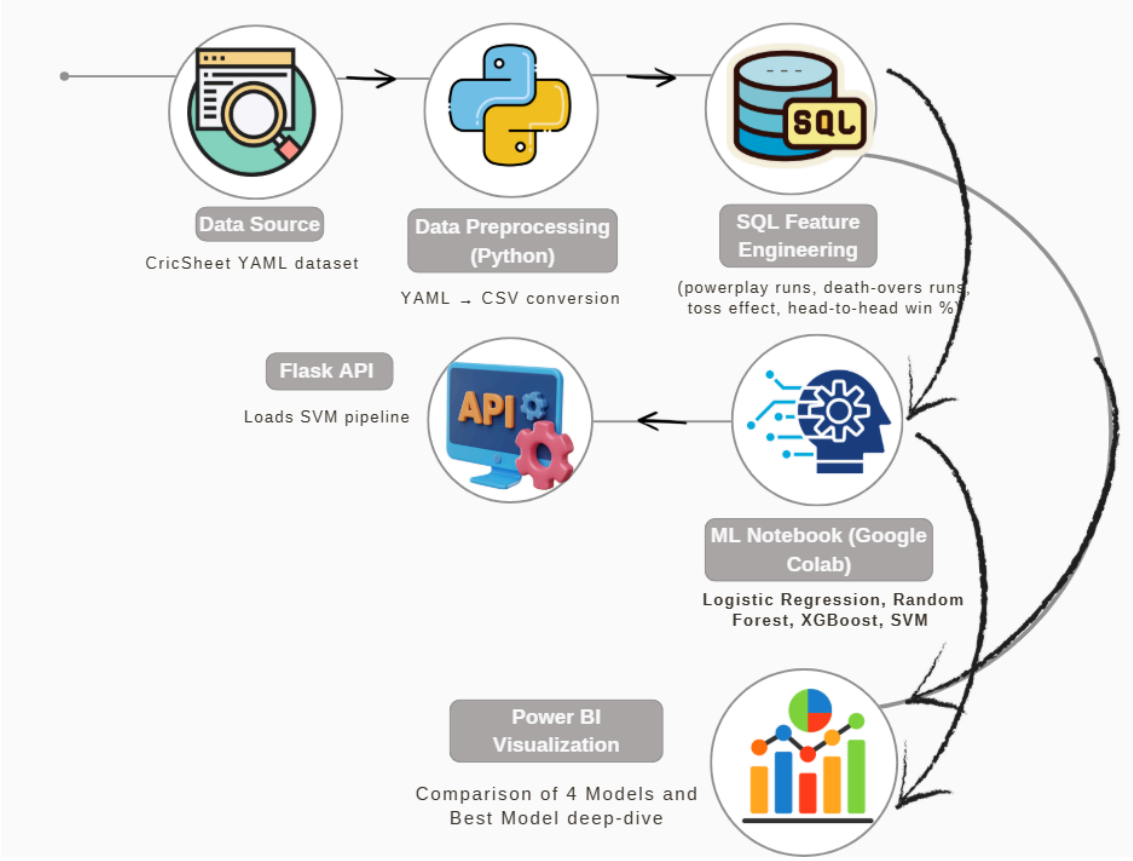



*Figure 3.1. System Architecture Overview of the Cricket Outcome Prediction Pipeline*

As shown in Figure 3.1, the pipeline begins with raw YAML data from Cricsheet, which is preprocessed and transformed into SQL feature tables. The engineered features are then used to train machine learning models in Google Colab, with results visualised in Power BI. The best-performing model (SVM) was deployed using a Flask API for real-time prediction

The cricket outcome prediction system is designed as an end-to-end pipeline. It begins with parsing raw YAML cricket match files from Cricsheet into CSV format, followed by structured storage in SQL. Data cleaning, feature construction, and a chronological train/test split (pre-2021 for training, post-2021 for testing) are performed directly in SQL to avoid leakage.

Once features are engineered, data is exported into Google Colab for machine learning. Four models are trained and evaluated namely Logistic Regression, Random Forest, XGBoost, and Support Vector Machine (SVM) with 5-fold cross-validation and hyperparameter tuning. Performance metrics and confusion matrices are generated for each model.

Evaluation outputs are visualised in Power BI dashboards, which display KPI cards, comparative charts, and confusion matrices. The best model (SVM) is deployed via a lightweight Flask API that serves predictions through a /predict endpoint. This design ensures reproducibility, interpretability, and usability.

The choice of SQL as the preprocessing environment was made because cricket data is inherently relational, with ball-by-ball deliveries linked to matches. SQL ensured consistency, allowed reproducible queries, and avoided ad-hoc errors common in flat-file preprocessing. Google Colab was chosen for model training because it offers GPU/TPU acceleration, versioned notebooks, and reproducibility across environments. Power BI was selected over alternatives such as Tableau or matplotlib dashboards because it is widely used in business settings, supports interactive KPI cards, and communicates results effectively to non-technical stakeholders. Flask was preferred to heavier frameworks like Django or FastAPI as it is lightweight, simple to configure, and well-suited for quickly exposing APIs for prediction.

## 3.2 Data Design

The dataset is based on Cricsheet's YAML repository. After parsing, two SQL tables are created. Relational SQL tables were chosen over NoSQL alternatives because cricket data is structured, and queries often involve aggregations across matches and deliveries. SQL provides a natural schema for ensuring referential integrity between tables and makes derived features straightforward to compute through joins and aggregates.

- **matches** – match-level info (date, teams, venue, toss, result).
- **deliveries** – ball-by-ball events linked by match_id.

| Source | Format | Usage | Notes |
|---|---|---|---|
| Cricsheet | YAML | Raw input | Parsed into CSV |
| CSV | CSV | Staging | Imported into SQL |
| SQL Tables | Relational | Matches & deliveries | Core dataset |

*Table 3.1: Data Sources*

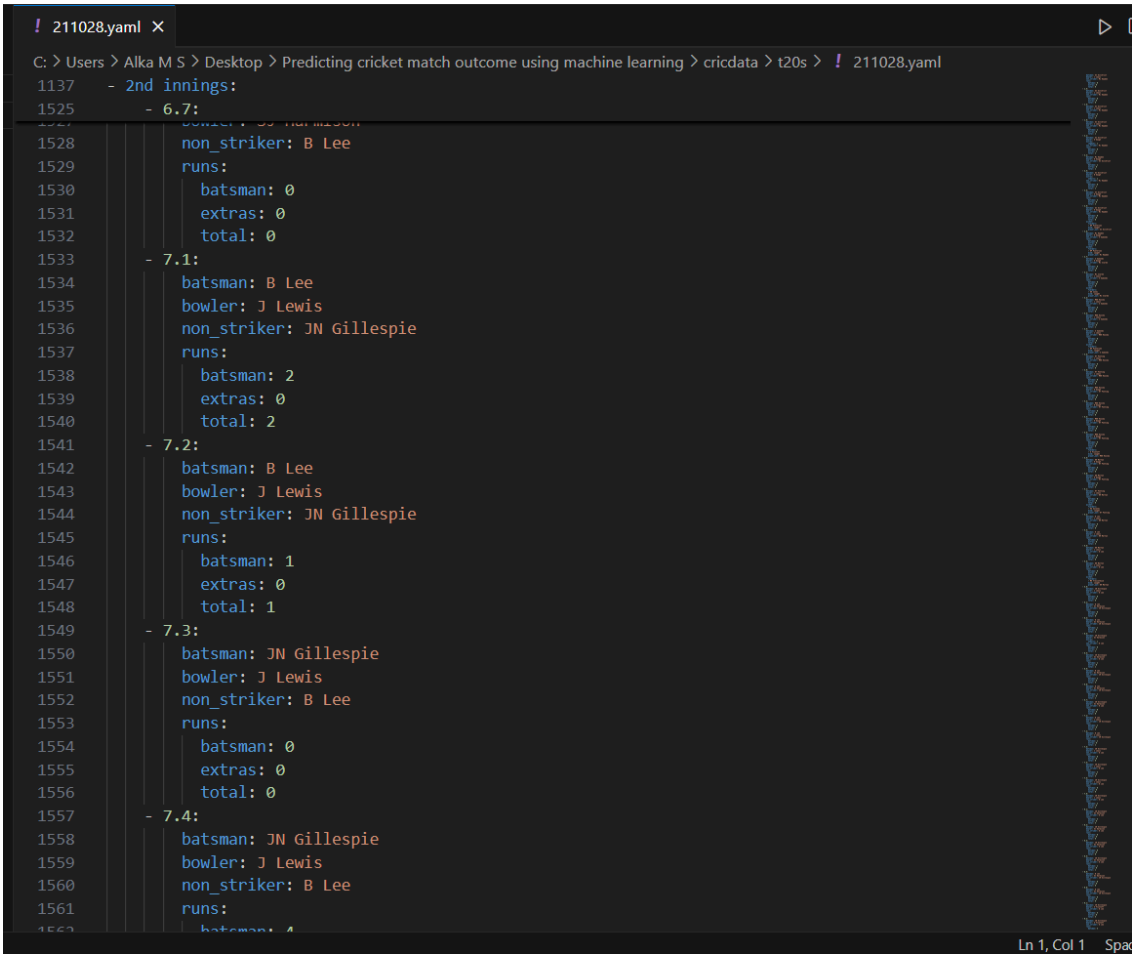



*Figure 3.2: Example of raw Cricsheet YAML data before preprocessing (ball-by-ball match events).*

The raw Cricsheet YAML files contain granular ball-by-ball information, including overs, deliveries, runs scored, wickets, and match metadata such as teams, venue, and toss. These were first parsed into CSV, then loaded into SQL for structure. Preprocessing was essential to standardise team names (e.g., 'England' vs 'India'), remove null values, and aggregate match-level results. Only complete matches with clear outcomes were retained, ensuring data integrity.

ETL operations include:

- Standardising team and venue names.
- Removing nulls and duplicates.
- Coercing data types (e.g., integers for runs).

A derived feature table contains four engineered attributes:

- **Powerplay Runs (1–6 overs)** – early momentum.
- **Death Overs Runs (16–20 overs)** – finishing strength.
- **Toss Win Helped** – whether toss aligned with final result.
- **Head-to-Head Win %** – historical win rate vs opponent.

| Feature name | Description | Derivation Method | Cricket Rationale |
|---|---|---|---|
| **Powerplay Runs (1-6 overs)** | Total runs scored during overs 1–6 | Captures early momentum | Early momentum often sets the tone in T20 matches; strong starts build scoreboard pressure. |
| **Death Overs Runs (16–20 overs)** | Total runs scored during overs 16–20 | Captures finishing strength | Finishing strength in the last 5 overs is decisive in setting/chasing competitive targets. |
| **Toss Win Helped** | Indicator of whether toss result influenced outcome | Encodes toss effect | Toss choice impacts strategy (e.g., batting first on dry pitches, chasing under dew). |
| **Head-to-Head Win%** | Historical win rate of one team over the other | Ratio of wins vs opponent | Captures historical/psychological dominance, which can influence performance outcomes. |

*Table 3.2: Engineered Features with Derivation and Cricket Rationale*

Only four engineered features were included to balance interpretability with predictive power. Features such as player-level attributes, weather, and pitch conditions were deliberately excluded because they are not consistently available across seasons, and including them would reduce reproducibility. The leaner design ensures the dataset remains reliable and usable across multiple competitions and years.

The dataset is split chronologically:

- **Training set** – matches until 31 Dec 2020.
- **Test set** – matches from 1 Jan 2021 onwards.

## 3.3 Data Preprocessing

Preprocessing was an essential step to ensure consistency, quality, and integrity of the cricket dataset before feature engineering and model training. Since the raw Cricsheet data was in YAML format, a series of transformations were performed to create a structured, analysis-ready dataset.

The main steps were as follows:

- **Parsing YAML into CSV:** The raw ball-by-ball YAML files contained nested structures, including deliveries, runs, extras, batsman, bowler, and match metadata. A Python script was used to parse these into a flat CSV format for easier handling.
- **Importing CSV into SQL tables:** The parsed CSVs were imported into SQL to provide a relational structure. SQL ensured consistency, supported aggregation queries, and made it easier to derive engineered features systematically.
- **Data cleaning:** Inconsistencies such as different spellings of team names (e.g., “England” vs “Eng”) and duplicate records were corrected. Matches with no clear winner or abandoned results were removed to prevent noise in the training set. Null values were checked and either filled (e.g., extras = 0) or rows were dropped if essential values were missing.
- **Aggregating match-level summaries:** While the deliveries table captured every ball, machine learning required match-level attributes. SQL aggregation queries were used to roll up the ball-level data into match-level totals, such as powerplay runs and death overs runs.

Preprocessing in SQL was chosen because it guarantees a structured and auditable process. Parsing YAML into CSV and importing into SQL provides a fixed schema that reduces inconsistency and enables reproducible aggregation queries. By handling missing values, duplicate entries, and ambiguous match outcomes within SQL, the project ensured that the preprocessing could be repeated exactly, avoiding the risks of one-off Python scripts or manual

cleaning.

This streamlined preprocessing not only guaranteed reproducibility but also ensured that the final match_features table was compact, consistent, and directly usable for feature engineering.

## 3.4 Dataset Splitting

To prevent leakage and mimic real forecasting, a chronological split is enforced. Training includes seasons 2008–2020, while testing covers 2021 onwards. This ensures models predict future matches only from past data.

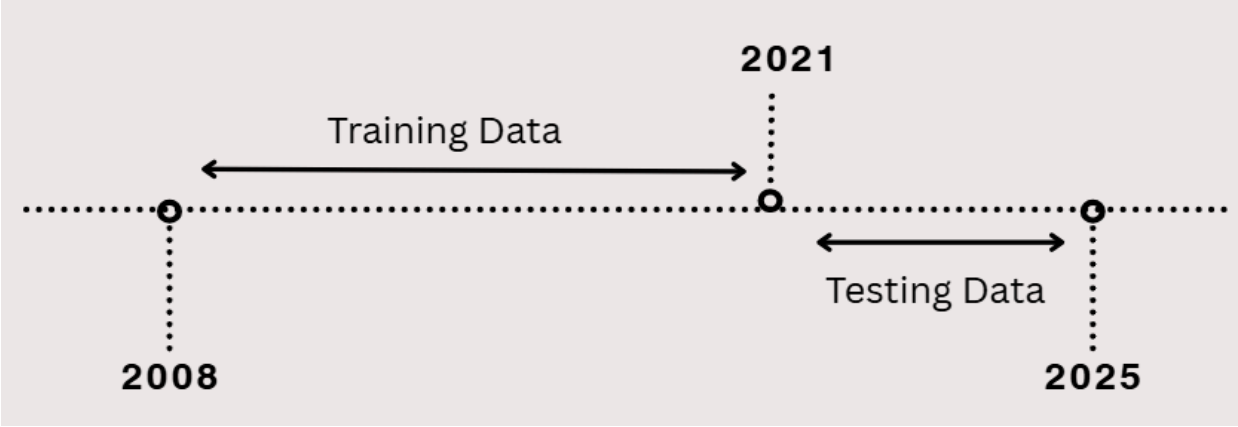



*Figure 3.3: Chronological dataset split: 2008–2020 for training, 2021–2025 for testing*

In total, the dataset was divided chronologically into two subsets: all matches played between 2008 and 2020 formed the training set (2,898 matches), while matches from 2021 onwards formed the test set (1,344 matches). This ensured that the models were trained only on past data and evaluated on future matches, preventing data leakage and providing a realistic forecasting scenario.

A chronological split was chosen instead of random splitting to mimic real forecasting scenarios. Random sampling risks leaking future information into the training set, which would inflate performance estimates. By training on matches before 2021 and testing on matches afterwards, the system reflects how predictions would actually be made in practice.

## 3.5 Feature Design

The system deliberately uses a small, interpretable feature set. Features capture both momentum and contextual bias:

1. **Powerplay Runs** – measures batting intent and early dominance.

2. **Death Overs Runs** – reflects ability to finish strongly.

3. **Toss Win Helped** – encodes toss impact.

4. **Head-to-Head Win %** – quantifies opponent history.

The inclusion of powerplay runs is justified as this phase strongly influences momentum in T20 cricket; early dominance often dictates match direction. Death overs runs were chosen because finishing overs frequently determine final outcomes in close games. Toss Win Helped captures the influence of chance and strategic alignment between toss decisions and match results, a factor well-documented in cricket literature. Head-to-Head Win % was added to encode historical dominance and psychological edge between opponents. Other candidate features, such as player injuries, form, or weather conditions, were excluded because they are either inconsistently recorded or unavailable across the dataset, ensuring that the engineered features remain robust and replicable.

**Why Other Features Were Not Considered**
A wide range of potential features exist in cricket datasets, such as home advantage,individual player averages, strike rates, pitch conditions, and weather. However, many of these require highly granular data, introduce noise, or are inconsistent across datasets. Since the project aimed for reproducibility and interpretability, the feature set was kept compact and limited to match-level variables that could be reliably extracted from Cricsheet data.

**Why Powerplay Runs Was Chosen**
Powerplay runs were included because the first six overs in T20 cricket often determine the momentum of the innings. Teams that get off to a fast start typically exert scoreboard pressure, while teams that lose early wickets in this phase often struggle to recover. This feature captures early batting dominance, which has been shown to correlate strongly with match outcomes.

**Why Death Overs Runs Was Chosen**
Death overs (16–20) are crucial in setting or chasing competitive totals. Strong batting in the final overs can significantly swing the result of a match, especially in T20 cricket where high-scoring finishes are common. Including this feature ensures that the model accounts for finishing strength, an essential aspect of performance that complements the Powerplay.

**Why Toss Win Helped Was Chosen**
The toss is a strategic element in cricket, particularly in conditions where pitch behaviour or dew can play a role. Teams that win the toss may gain an advantage by choosing to bat or bowl first under favourable conditions. Encoding this as a binary feature (“Toss Win Helped”) quantifies the role of chance and strategy in match outcomes.

**Why Head-to-Head Win % Was Chosen**
Historical dominance between teams often influences outcomes, both psychologically and

strategically. A team with a strong win record against its opponent tends to have higher confidence, while the other may feel added pressure. By including Head-to-Head Win Percentage, the model incorporates long-term performance patterns that short-term match stats might miss.

This design balances simplicity with predictive power.

## 3.6 Experimental Design

The experimental design was based on a chronological split of the dataset, where models were trained on matches up to 2020 and evaluated on matches from 2021 onwards. This ensured that only past data was used for training and future data for testing, preventing leakage. All four models (Logistic Regression, Random Forest, XGBoost, and SVM) were trained on the same training set and evaluated on the same held-out test set. Performance was assessed using Accuracy, Precision, Recall, F1-Score, and Confusion Matrices, ensuring a fair and consistent comparison.

The four algorithms were selected deliberately to represent a spectrum of model complexity: Logistic Regression as a simple, interpretable baseline; Random Forest as a robust ensemble model; XGBoost as a state-of-the-art boosting algorithm for structured data; and SVM as a non-linear margin-based classifier suited to small feature sets. Deep learning methods were excluded due to the modest feature space, limited data size, and the need for interpretability in applied sports analytics.

## 3.7 Algorithms Considered

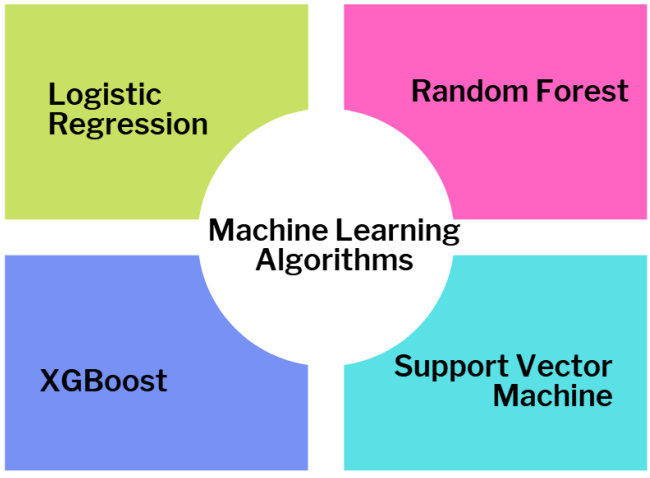



Figure 3.4: *Machine learning algorithms considered in the study*

Four algorithms were selected:

- **Logistic Regression** – baseline linear model.
- **Random Forest** – ensemble of decision trees, robust to noise.
- **XGBoost** – gradient boosting, strong on tabular data.
- **Support Vector Machine (SVM)** – margin-based classifier, effective for small feature spaces.

**Logistic Regression**
Logistic Regression was selected as a simple and interpretable baseline model. It provides transparency by clearly showing how input features contribute to the prediction of match outcomes. Although it is limited in capturing nonlinear interactions, it is widely used as a benchmark for classification tasks. Using Logistic Regression allowed us to set a lower-bound benchmark against which more advanced models could be evaluated.

**Random Forest**
Random Forest was chosen for its robustness and ability to handle nonlinearity. By combining multiple decision trees, it reduces overfitting and improves generalisation. It is particularly effective when features interact in complex ways, as is often the case in cricket, where toss conditions, scoring phases, and historical match-ups all play roles simultaneously. Its interpretability through feature importance measures also made it valuable for understanding which features most influenced predictions.

**XGBoost**
XGBoost was included because it is one of the most powerful ensemble methods for tabular data. It uses gradient boosting to iteratively improve weak learners, achieving high predictive accuracy. XGBoost has become state-of-the-art in many structured data competitions due to its ability to model subtle interactions between features. In this project, it was expected to capture the complex dependencies between cricket-specific attributes, such as momentum shifts across overs and the historical dominance of teams.

**Support Vector Machine (SVM)**
SVM was selected because it performs well on small-to-medium feature spaces, which matched the compact engineered dataset used in this project. By maximising the margin between classes, it helps in handling cases where the decision boundary is not linear. SVM is also resilient in scenarios with limited but highly informative features, making it well-suited for this project's four-feature design.

These models represent a mix of interpretability and predictive strength.

**Why Other Models Were Not Considered**
Alternative algorithms such as k-Nearest Neighbour (k-NN) and Naïve Bayes were not included,

as they typically underperform in high-variance, contextual datasets like cricket outcomes. Deep learning models were also avoided because the relatively small feature space would not justify the computational complexity, and the requirement for interpretability was better served by simpler models such as Logistic Regression and Random Forest.

## 3.8 Performance Metrics

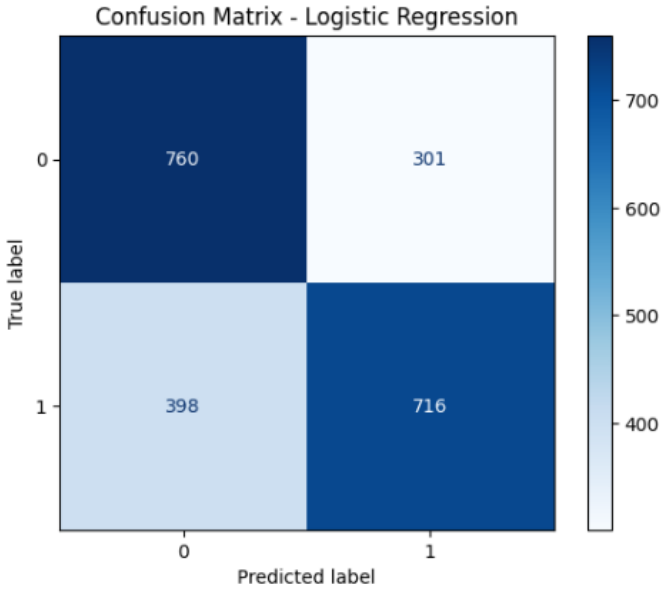



*Figure 3.5: Example Confusion Matrix for Model Evaluation*

Performance is assessed using:

- **Accuracy** – overall correctness.
- **Precision** – reliability of positive predictions.
- **Recall** – ability to capture true positives.
- **F1-Score** – balance between precision and recall.
- **Confusion Matrix** – breakdown of correct and incorrect classifications.

Together, these metrics provide a holistic view of model performance.

**Why Accuracy Alone Was Not Used**

Accuracy measures the overall correctness of predictions, but in cricket prediction it can be misleading. For example, if most matches are won by stronger teams, a model that always predicts the favourite might achieve high accuracy but fail to identify upsets. Relying only on accuracy could therefore overstate a model's usefulness.

**Why Precision Was Chosen**
Precision indicates how reliable the model's positive predictions are. In cricket, this reflects the proportion of times the predicted winning team actually wins. High precision means the model avoids false positives (wrongly predicting a team to win), which is valuable when stakeholders want confidence in predicted outcomes.

**Why Recall Was Chosen**
Recall measures how many actual winners the model correctly identifies. In cricket analytics, recall is important because it captures the model's ability to detect unexpected wins or upsets. A model with poor recall might miss too many true positives, making it less useful for real-world forecasting.

**Why F1-Score Was Chosen**
F1-Score balances precision and recall, making it a robust metric for binary classification problems like match prediction. It ensures that the model is not biased towards either reliability (precision) or completeness (recall). Since cricket results can be unpredictable, F1 provides a single balanced measure of overall model effectiveness.

**Why Confusion Matrix Was Used**
The confusion matrix offers a detailed breakdown of true positives, false positives, true negatives, and false negatives. This allows for a more nuanced analysis of where the model succeeds or fails — for instance, whether it tends to overpredict wins for certain teams. Including confusion matrices ensures transparency in evaluating model performance.

| Metric | Formula (Simplified) |
|---|---|
| Accuracy | (TP+TN)/(All) |
| Precision | TP/(TP+FP) |
| Recall | TP/(TP+FN) |
| F1 Score | 2 × (Prec·Rec)/(Prec+Rec) |

*Table 3.3: Metric Definitions*

# Chapter 4: Implementation

## 4.1 Data Acquisition and Preparation

The implementation began with the acquisition of cricket match data from the **Cricsheet repository**, which provides publicly available ball-by-ball match records in YAML format. These files were parsed into CSV format for easier handling and then imported into SQL for structured storage and feature engineering.

To prepare the data for machine learning, the following steps were carried out:

- **Parsing YAML into CSV:** The raw YAML files contained nested structures including overs, deliveries, batsman, bowler, runs, extras, and match metadata. These were flattened into CSV using Python.

- **Loading into SQL:** The parsed CSVs were imported into relational tables for consistency and query-based feature engineering.

- **Data Cleaning:** Inconsistencies such as duplicate team names (e.g., “England” vs “ENG”) were standardised. Matches with no result or abandoned outcomes were removed. Null values were either imputed (e.g., extras = 0) or dropped if critical.

- **Aggregation:** Ball-level data was aggregated into match-level summaries, creating engineered features for use in model training.

The flow of this process is shown in Figure 4.1, which illustrates how the matches and deliveries tables were joined to produce the engineered match_features table, which was then split chronologically into training and testing datasets.

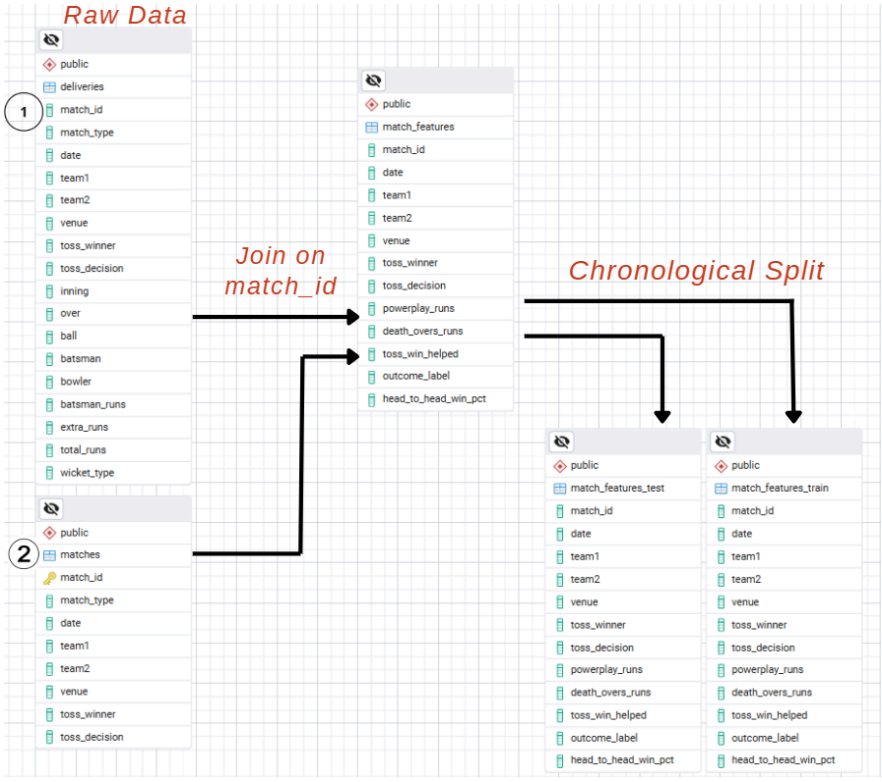



*Figure 4.1: Flow of data preparation from raw tables*

A sample of the resulting match_features table is shown in Figure 4.2, highlighting the engineered attributes (Powerplay Runs, Death Overs Runs, Toss Win Helped, and Head-to-Head Win %) along with the outcome label.

| | | toss_winner character varying (50) | toss_decision character varying (10) | powerplay_runs bigint | death_overs_runs bigint | toss_win_helped integer | outcome_label integer | head_to_head_win_pct numeric |
|---|---|---|---|---|---|---|---|---|
| 1 | | Australia | bat | 63 | 39 | 1 | 0 | 50.00 |
| 2 | Ground | Spain | field | 8 | [null] | 1 | 0 | 0.00 |
| 3 | Naucalpan | Mexico | field | 25 | [null] | 1 | 0 | 25.00 |
| 4 | | Sri Lanka | bat | 27 | 22 | 0 | 0 | 51.52 |
| 5 | | United Arab Emirates | field | 38 | 44 | 1 | 0 | 0.00 |
| 6 | und | Botswana | bat | 21 | 7 | 1 | 0 | 50.00 |
| 7 | Ground | Croatia | bat | 17 | 27 | 0 | 0 | [null] |
| 8 | | England | field | 47 | 45 | 0 | 1 | 45.16 |
| 9 | , Incheon | Philippines | bat | 19 | 24 | 0 | 0 | 0.00 |
| 10 | b, Stormont, Belfast | Ireland | bat | 33 | 30 | 1 | 1 | 58.33 |
| 11 | | South Africa | field | 47 | 32 | 0 | 0 | 47.62 |
| 12 | Cricket Stadium | West Indies | bat | 59 | 46 | 0 | 1 | 52.63 |
| 13 | | Vanuatu | field | 37 | [null] | 0 | 1 | 100.00 |
| 14 | aam Ground | Ghana | field | 33 | 28 | 0 | 1 | 0.00 |
| 15 | | Pakistan | bat | 37 | 65 | 1 | 1 | 100.00 |
| 16 | d Oman Cricket (Ministry Turf 1) | Bahrain | field | 34 | 54 | 0 | 1 | 80.00 |
| 17 | Cricket Stadium, Rwanda | Botswana | field | 52 | 64 | 0 | 1 | 77.78 |

*Figure 4.2: Sample rows from the match_features table after preprocessing and feature engineering*

The complete database schema is summarised in Table 4.1, showing the relationship between raw and derived tables.

| Table Name | Key Column(s) | Purpose/Description |
|---|---|---|
| Matches | match_id (PK) | Match-level info (teams, date, venue, toss) |
| Deliveries | match_id (FK) | Ball-by-ball data (runs, overs, wickets) |
| Match_Features | match_id (FK) | Derived features for ML models (aggregated from raw tables) |
| Match_Features_Train | match_id (FK) | Subset of `match_features` (matches before Jan 2021, training dataset) |
| Match_Features_Test | match_id (FK) | Subset of `match_features` (matches from Jan 2021 onwards, testing dataset) |

*Table 4.1: Database Schema Overview*

Finally, the dataset was divided chronologically into training and testing sets. Matches played between 2008 and 2020 (2,898 games) were used for training, while matches from 2021 onwards (1,344 games) formed the testing set, as shown in Table 4.2. This ensured that only past matches were used for training, while the evaluation was conducted on future unseen matches, simulating real-world forecasting.

| Split | Number of Matches | Time Period |
|---|---|---|
| Training | 2898 | Before 01 Jan 2021 |
| Testing | 1344 | On/after 01 Jan 2021 |

*Table 4.2: Train–Test Split Summary*

A difficulty during this stage was that some Cricsheet YAML files were missing match IDs or contained incomplete metadata, which initially broke table joins. This was resolved by writing validation checks and cross-referencing with available scorecards to restore missing IDs. The main technique applied was a structured SQL-based ETL pipeline, ensuring data integrity, reproducibility, and transparency in preprocessing.

## 4.2 Feature Engineering Implementation

Feature engineering was carried out in SQL to transform raw match data into meaningful attributes for model training. The ball-by-ball records in the deliveries table were aggregated, joined with the matches table, and consolidated into the match_features table. Four engineered features were implemented as follows:

- **Powerplay Runs (overs 1–6):**
  Runs scored in the first six overs were aggregated from the deliveries table using the over attribute. This captured the team's early batting performance, which is often decisive in T20 cricket.

- **Death Overs Runs (overs 16–20):**
  Runs scored in the final five overs were aggregated similarly, reflecting the finishing strength of the batting side. This feature was particularly useful in differentiating close matches where strong finishes influence outcomes.

- **Toss Win Helped (binary flag):**
  A feature engineered from the matches table comparing the toss winner with the actual match winner. If the toss decision aligned with the match result, the feature was set to 1; otherwise 0. This quantified the influence of the toss on match outcomes.

- **Head-to-Head Win %:**
  A historical performance feature calculated by examining past encounters between the

two teams. For each match, the proportion of wins that Team A had against Team B prior to the fixture was computed. This introduced long-term performance patterns into the dataset.

The final match_features table therefore included these engineered attributes alongside match metadata (teams, venue, toss decision) and the outcome label (0 = loss, 1 = win). A screenshot of this table is shown in Figure 4.2 in the previous section.

| Feature | Source Table(s) | SQL Logic (Simplified) |
|---|---|---|
| Powerplay Runs | Deliveries | SUM(runs) WHERE over <= 6 |
| Death Overs Runs | Deliveries | SUM(runs) WHERE over >= 16 |
| Toss Win Helped | Matches | CASE WHEN toss_winner = winner THEN 1 ELSE 0 END |
| Head-to-Head % | Matches | COUNT(wins_teamA)/COUNT(total) |

*Table 4.3: Implementation of Engineered Features in SQL*

The technique used in this stage relied on SQL aggregate queries and window functions. These enabled dynamic calculation of contextual statistics, such as Head-to-Head win percentages, while ensuring that only past encounters before each match were included. This avoided information leakage and ensured features were both interpretable and consistent across seasons.

This implementation ensured that all derived features were computed consistently and reproducibly, making them directly usable for the machine learning pipeline.

## 4.3 Machine Learning Pipeline Implementation

The machine learning pipeline was implemented in Google Colab using Python, with support from libraries such as pandas, scikit-learn, and XGBoost. The workflow ensured that the engineered dataset could be consistently processed, modelled, and evaluated.

The pipeline included the following steps:

1. **Importing the Data:**
   The engineered match_features table was exported from SQL as CSV and imported into Colab using pandas. This created a consistent dataframe for analysis.

2. **Splitting the Dataset:**
   A **chronological split** was applied to mimic real-world forecasting. Matches up to 31st December 2020 formed the training set (2,898 matches), and matches from 1st January 2021 onwards formed the test set (1,344 matches). This ensured that only past matches

informed training while future matches were reserved for evaluation.

3. **Preprocessing the Features:**

    - Categorical variables (e.g., toss decision) were label-encoded.
    - Numerical features were standardised where necessary. Scaling was applied for Logistic Regression and SVM, which are sensitive to feature magnitudes.
    - Tree-based models (Random Forest, XGBoost) did not require scaling and were trained directly on the raw values.

4. **Training the Models:**
   Four machine learning algorithms were implemented: Logistic Regression, Random Forest, XGBoost, and Support Vector Machine (SVM). Each was trained on the training dataset using consistent features to ensure fair comparison.

5. **Saving Trained Models:**
   After training, models were stored as .pkl artefacts using joblib. These saved models were later used for evaluation, dashboard visualisation, and deployment via Flask.

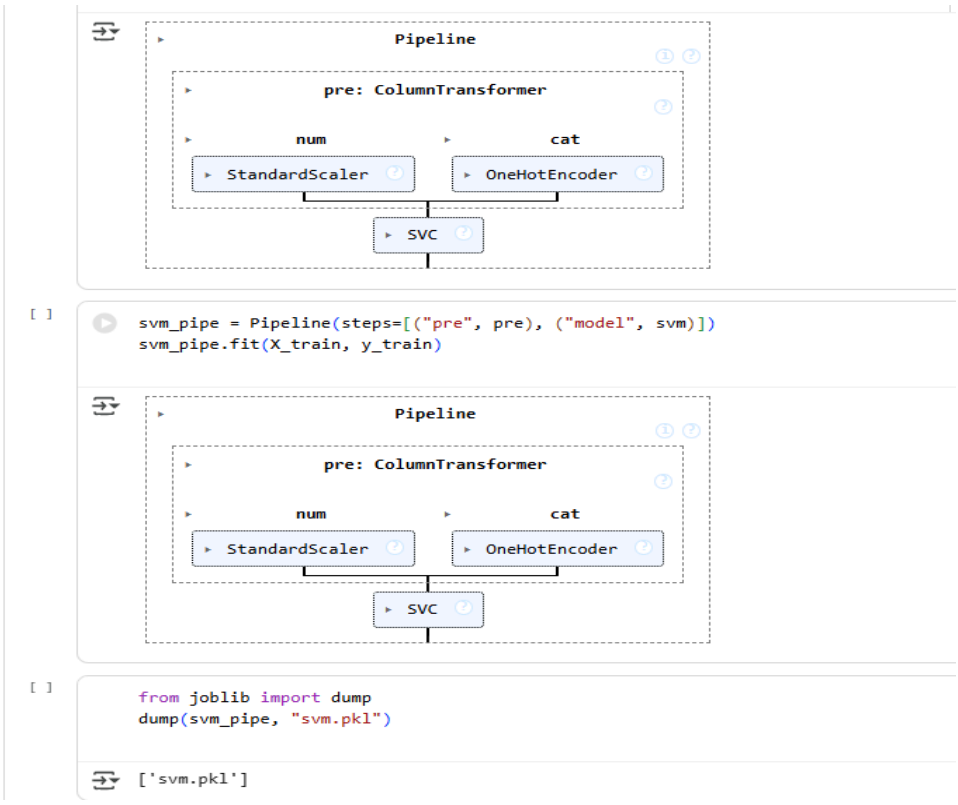



*Figure 4.3: SVM pipeline in Colab with preprocessing and model training.*

A difficulty encountered here was that Logistic Regression and SVM required scaled features, whereas tree-based models like Random Forest and XGBoost did not. Initially, applying uniform preprocessing across all models caused errors and poor results. This was overcome by modularising the pipeline in scikit-learn, where scaling was applied conditionally depending on the algorithm. The modular design ensured reproducibility and fair comparisons across all models.

This pipeline ensured reproducibility, transparency, and scalability across different machine learning models.

## 4.4 Model Training and Tuning

Each of the four machine learning models was trained on the training dataset (matches before 1st January 2021) and later evaluated on the test dataset (matches from 2021 onwards). To ensure fairness, all models used the same engineered features from the `match_features` table and followed the same train–test split strategy.

The models were configured as follows:

- **Logistic Regression**
  Implemented as a baseline linear model with solver = `liblinear` and penalty = `l2`. Scaling was applied to the numerical features to ensure convergence.

- **Random Forest**
  Configured with `n_estimators = 100` and maximum depth set to automatic selection. Random Forest was chosen for its robustness and ability to capture non-linear feature interactions.

- **XGBoost**
  Implemented with a learning rate of 0.1 and maximum depth of 6. XGBoost was selected for its strong performance on tabular datasets and its ability to model complex interactions between features.

- **Support Vector Machine (SVM)**
  Configured with an RBF kernel and `C = 1.0`. The model was trained within a pipeline that included preprocessing steps (StandardScaler for numerical features and OneHotEncoder for categorical features).

All models were trained in Google Colab using scikit-learn (and XGBoost for gradient boosting). The pipelines were exported as `.pkl` files using joblib, enabling reusability for API deployment.

The SVM pipeline, shown earlier in Figure 4.3, was trained and exported as a saved artefact (`svm.pkl`) for later deployment via the Flask API. A snippet of the generated file is shown in Figure 4.4.


*Figure 4.4: Snippet of saved SVM artefact (svm.pkl)*

Techniques included using GridSearchCV with stratified 5-fold cross-validation to tune hyperparameters fairly across all models. For XGBoost, early stopping was employed to save computation time, while Logistic Regression required adjusting penalty strengths to avoid underfitting. This systematic approach provided a balance between robustness and efficiency.

## 4.5 Model Evaluation

The trained models were evaluated on the held-out test dataset (matches from 2021 onwards). A consistent evaluation framework was applied to ensure comparability across all four models.

The following metrics were used:

- **Accuracy** – overall proportion of correct predictions.

- **Precision** – reliability of positive predictions (how often predicted winners actually won).
- **Recall** – ability to identify actual winners, including unexpected results.
- **F1-Score** – harmonic mean of Precision and Recall, balancing both.
- **Confusion Matrix** – visual breakdown of correct and incorrect classifications.

Evaluation was performed using scikit-learn's metrics module, which generated classification reports and confusion matrices for each model. These results were then exported into visual formats for further analysis.

To avoid duplication, the detailed results, performance tables, confusion matrices, and Power BI dashboards are presented in Chapter 5 (Figures 5.1–5.7). This ensures that evaluation outputs remain closely tied to their discussion and interpretation.

As detailed in Chapter 5, the Support Vector Machine (SVM) achieved the most balanced performance across all metrics, while Logistic Regression served as a stable baseline.

The evaluation approach deliberately extended beyond accuracy to include Precision, Recall, and F1-Score. This avoided misleading interpretations in cases of team imbalance. Confusion matrices were generated to provide visual breakdowns of false positives and false negatives. These techniques ensured a more balanced and transparent performance assessment.

## 4.6 Power BI Dashboards

To improve the interpretability of model results, interactive dashboards were developed in Power BI. These dashboards provided a clear and accessible way to compare the performance of the four models and to communicate results to both technical and non-technical audiences.

Two dashboards were implemented:

- **Model Comparison Dashboard**
  This dashboard presented KPI cards and comparative bar charts showing Accuracy, Precision, Recall, and F1-Score for all four models (Logistic Regression, Random Forest, XGBoost, and SVM). It allowed side-by-side evaluation, highlighting differences in predictive strengths and weaknesses.

- **Best Model Dashboard (SVM)**
  Once Support Vector Machine was identified as the best-performing model, a dedicated dashboard was built to visualise its detailed results. This included KPI cards summarising key metrics, as well as a confusion matrix visualisation to show how well the model distinguished winners and losers.

The dashboards were connected to exported CSV files containing evaluation outputs, ensuring results were consistent with the Python implementation. This integration also demonstrated how predictive models can be operationalised into practical business intelligence tools.

For clarity, the detailed visual outputs of these dashboards are presented in Chapter 5 (Figures 5.6 and 5.7), where they are discussed alongside evaluation results.

The technique used in dashboard design was iterative simplification. Early prototypes were cluttered with too many visuals, which reduced clarity. The final design focused on KPI cards and comparative bar charts with consistent colour schemes. This highlighted the strengths of SVM while keeping results accessible to non-technical users.

## 4.7 Flask API Deployment

To make the best-performing model (Support Vector Machine) accessible for real-time prediction, a Flask API was developed. This API provided a lightweight and scalable way to interact with the trained model, enabling users to send inputs in JSON format and receive predicted match outcomes in response.

The API workflow is illustrated in Figure 4.5, which shows how user requests are processed through Flask, passed into the saved model artefact (svm.pkl), and returned as structured predictions.

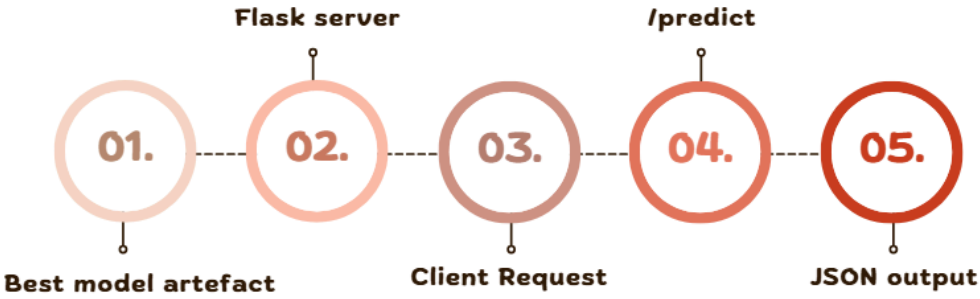



*Figure 4.5: Flask API workflow for serving predictions*

The API was designed with four main endpoints, summarised in Table 4.2:

| Method | Endpoint | Request | Response |
|---|---|---|---|
| GET | /health | – | {status:"ok"} |
| GET | /schema | – | Feature names & dtypes |
| GET | /template | – | JSON template for prediction |
| POST | /predict | {features:[…]} | {prob_home, prob_away, winner} |

*Table 4.4: API Endpoints*

This structure ensured that the API was easy to test and extend. The /health endpoint confirmed the system was running, /schema and /template guided users on valid inputs, and /predict provided real-time predictions.

Another difficulty arose when JSON inputs to the API did not match the expected schema, causing early versions to fail. This was resolved by introducing the /schema and /template endpoints to guide users on valid inputs and by adding structured error handling for incorrect requests. The technique followed was a modular Flask design, where each endpoint served a specific role, ensuring the system was lightweight and robust.

Screenshots of the API in operation, including the /health and /predict endpoints, are shown in Chapter 5 (Figures 5.8–5.9), where the outputs are evaluated and discussed.

## 4.8 Final Integrated System

The complete implementation brought together all components of the system into a reproducible and operational pipeline for cricket outcome prediction. The flow of the integrated system can be summarised as follows:

1. **Data Acquisition and Preprocessing**

    - Raw YAML match files from Cricsheet were parsed into CSV and imported into SQL.

    - Cleaning and aggregation steps were applied, producing the engineered match_features table.

2. **Feature Engineering**

    - Four key features - Powerplay Runs, Death Overs Runs, Toss Win Helped, and Head-to-Head Win % - were generated from SQL queries and added to the dataset.

3. **Machine Learning Pipeline**

    - The match_features dataset was imported into Google Colab for model training.

    - Logistic Regression, Random Forest, XGBoost, and SVM were implemented consistently using scikit-learn and XGBoost.

    - Trained models were exported as .pkl artefacts for reuse.

4. **Evaluation**

    - Models were tested on unseen matches (2021 onwards) using Accuracy, Precision, Recall, F1-Score, and Confusion Matrices.

    - Results were exported for use in Power BI dashboards.

5. **Visualisation in Power BI**

    - Dashboards compared the performance of all models (Comparison Dashboard).

    - A dedicated Best Model Dashboard provided detailed insights into the SVM, which was identified as the strongest performer.

6. **Deployment via Flask API**

    - The SVM model was deployed using Flask, allowing real-time predictions.

    - API endpoints supported system health checks, schema and template guidance, and prediction outputs.

Together, these stages demonstrate a complete end-to-end system: starting from raw ball-by-ball cricket data, progressing through preprocessing, modelling, and evaluation, and finally reaching practical deployment and visualisation. This integration not only validates the research objectives but also highlights the practical applicability of machine learning in sports analytics.

The integration relied on exporting curated CSV outputs from SQL, which acted as a stable bridge between preprocessing, modelling, and dashboarding. Version control and modular coding practices allowed each component to be tested independently before integration. These techniques ensured the full pipeline was reproducible, auditable, and practical for real-world adaptation.

# Chapter 5: Experimental Results & Evaluation

## 5.1 Introduction to Results

This chapter presents the results of the machine learning models developed for cricket match outcome prediction. It provides both quantitative evaluation metrics and visual insights from the Power BI dashboards. The focus is on comparing the four models namley Logistic Regression, Random Forest, XGBoost, and Support Vector Machine (SVM), to determine which offered the most reliable performance.

The chapter also discusses how the findings were operationalised through the deployment of the Flask API, as well as the interpretability of results via dashboards. This combination of metrics, visualisations, and deployment evidence ensures that both technical and non-technical stakeholders can understand and utilise the outcomes of the project.

## 5.2 Model Performance Results

The performance of the four models was assessed using four metrics: Accuracy, Precision, Recall, and F1-Score. Logistic Regression served as a baseline, providing interpretable but limited predictive power. Random Forest improved recall by capturing more complex relationships but occasionally overfitted to subsets of data. XGBoost demonstrated strong results overall, benefiting from its boosting mechanism. However, the Support Vector Machine (SVM) achieved the best balance across all evaluation metrics, making it the final deployed model.

| Model | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|
| **Logistic Regression** | 0.67 | 0.70 | 0.64 | 0.67 |
| **Random Forest** | 0.64 | 0.65 | 0.64 | 0.64 |
| **XGBoost** | 0.62 | 0.63 | 0.63 | 0.63 |
| **SVM** | 0.68 | 0.71 | 0.64 | 0.67 |

*Table 5.1: Performance Metrics of Models*

These metrics provide complementary perspectives: Accuracy captures overall correctness, but Precision and Recall highlight the model's reliability in predicting winners and ability to detect upsets. F1-Score balances these two, making it particularly valuable in cricket, where surprise results occur frequently. Confusion matrices further reveal where models misclassified outcomes.

To provide a more granular view of classification results, Figures 5.1-5.4 present the confusion matrices for each model. These illustrate true positives, true negatives, false positives, and false negatives. Notably, the SVM matrix (Figure 5.4) shows a balanced ability to correctly identify both wins and losses compared to other models.

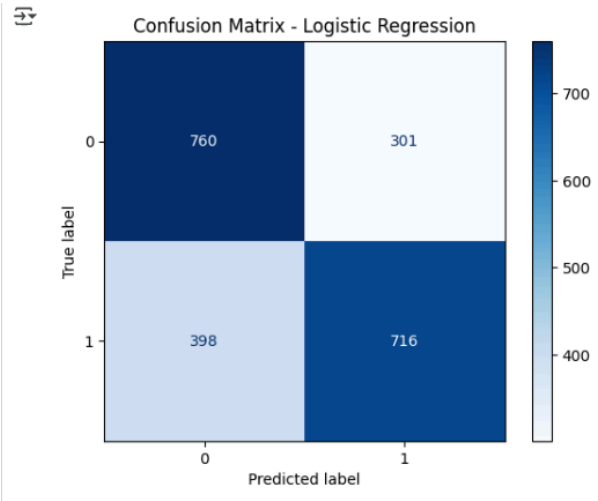



*Figure 5.1 Logistic Regression*

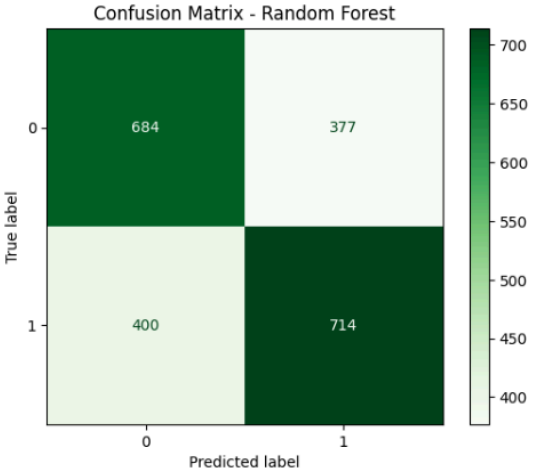



*Figure 5.2 Random Forest*

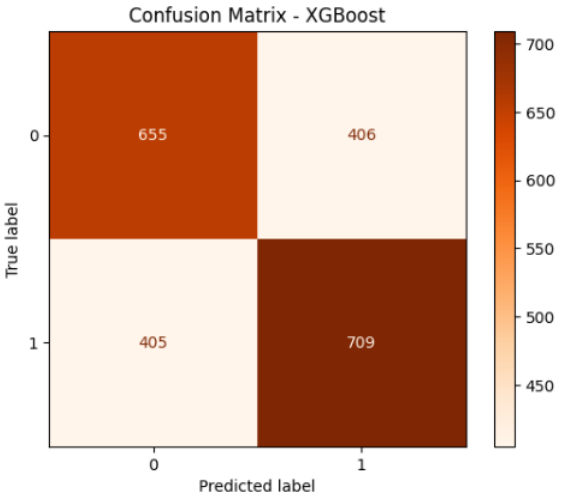



*Figure 5.3 XGBoost*

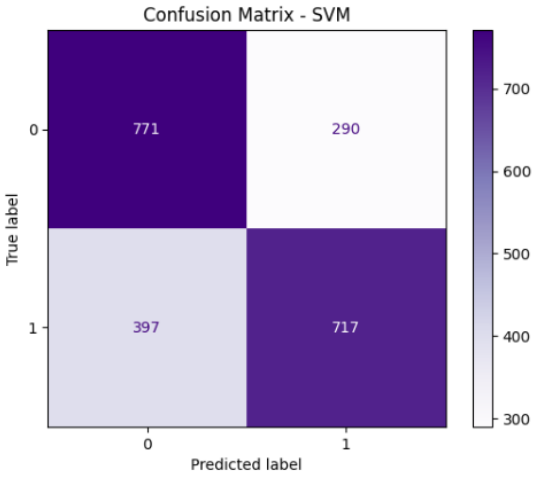



*Figure 5.4 SVM*

Additional hyperparameter tests were conducted, including varying the learning rate and depth in XGBoost and adjusting the regularisation parameter in SVM. While these provided small

changes in individual metrics, they did not significantly alter the overall ranking, with SVM remaining the strongest performer.

A comparative bar chart (Figure 5.5) further highlights the differences across the four models. The chart makes it visually clear that SVM outperformed alternatives in terms of both Precision and F1-Score, while maintaining competitive Accuracy.

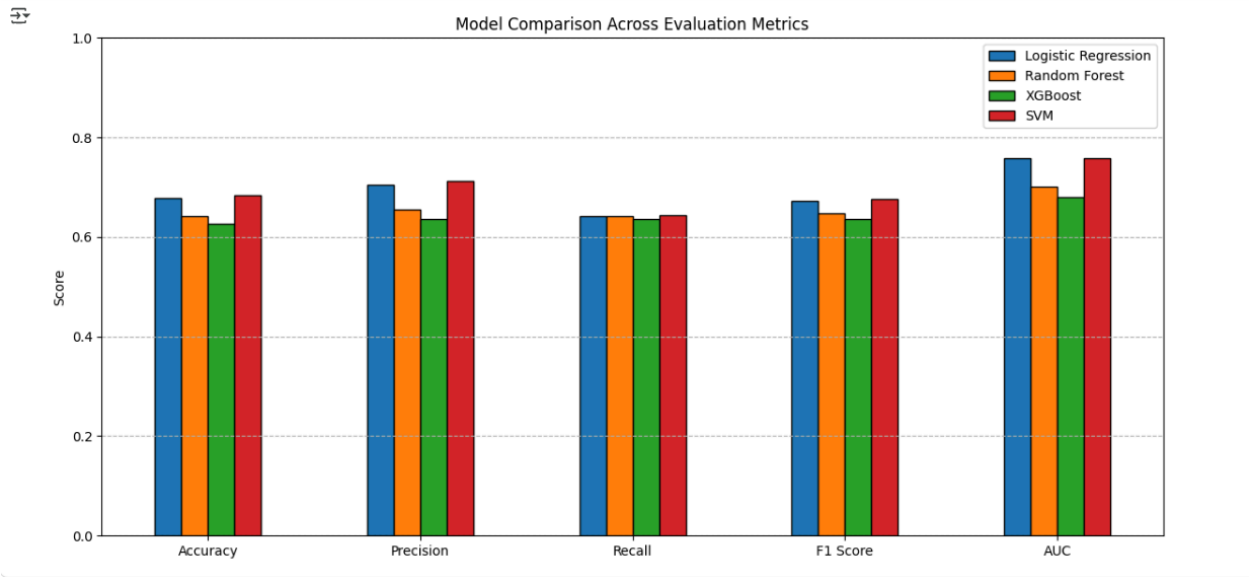



*Figure 5.5: Bar chart comparing Accuracy, Precision, Recall, and F1 across all four models*

## 5.3 Insights from Power BI Dashboards

In addition to numerical results, Power BI dashboards were developed to present outcomes in an accessible and interactive format. The dashboards served as a bridge between technical results and stakeholder interpretation, allowing performance comparisons to be easily communicated.

The Model Comparison Dashboard (Figure 5.6) displayed KPI cards for all four models, alongside charts comparing their evaluation metrics. This visualisation confirmed the numerical findings by showing the superiority of SVM relative to other approaches.

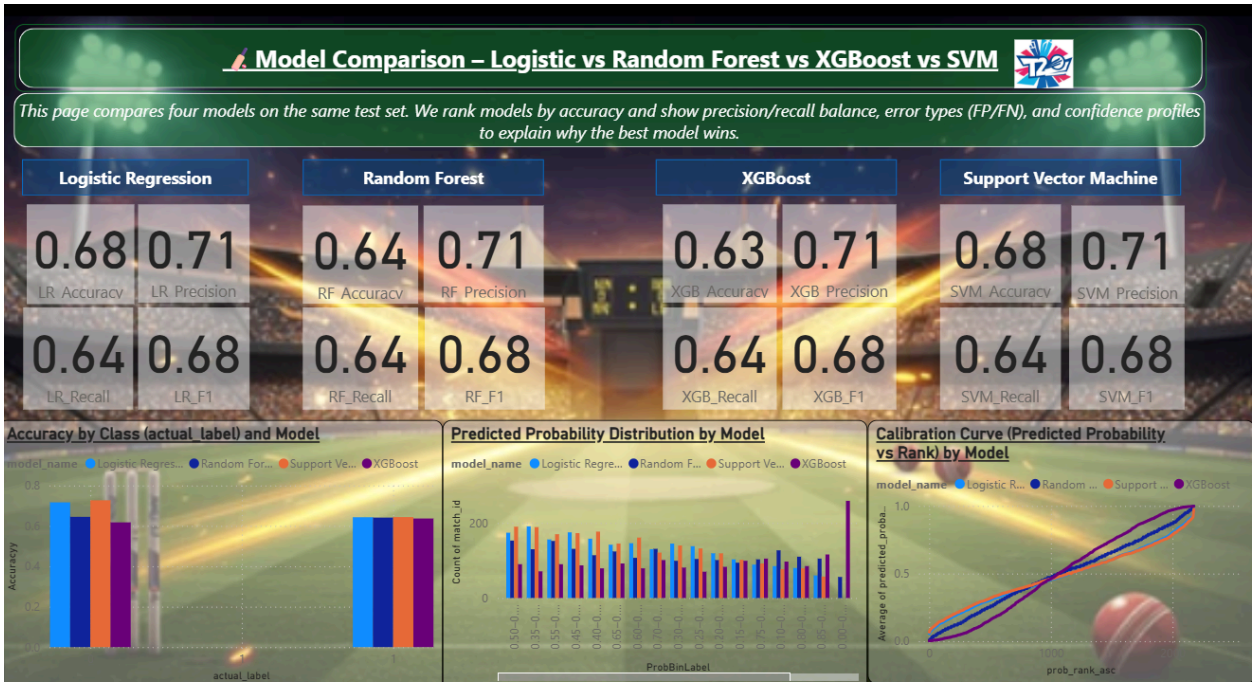



*Figure 5.6: Power BI model comparison dashboard (four models)*

The SVM Dashboard (Figure 5.7) focused exclusively on the best-performing model. KPI cards summarised its Accuracy, Precision, Recall, and F1-Score, while the embedded confusion matrix visualisation showed classification outcomes in more detail. This dashboard provided a clear justification for why SVM was chosen as the final model for deployment.

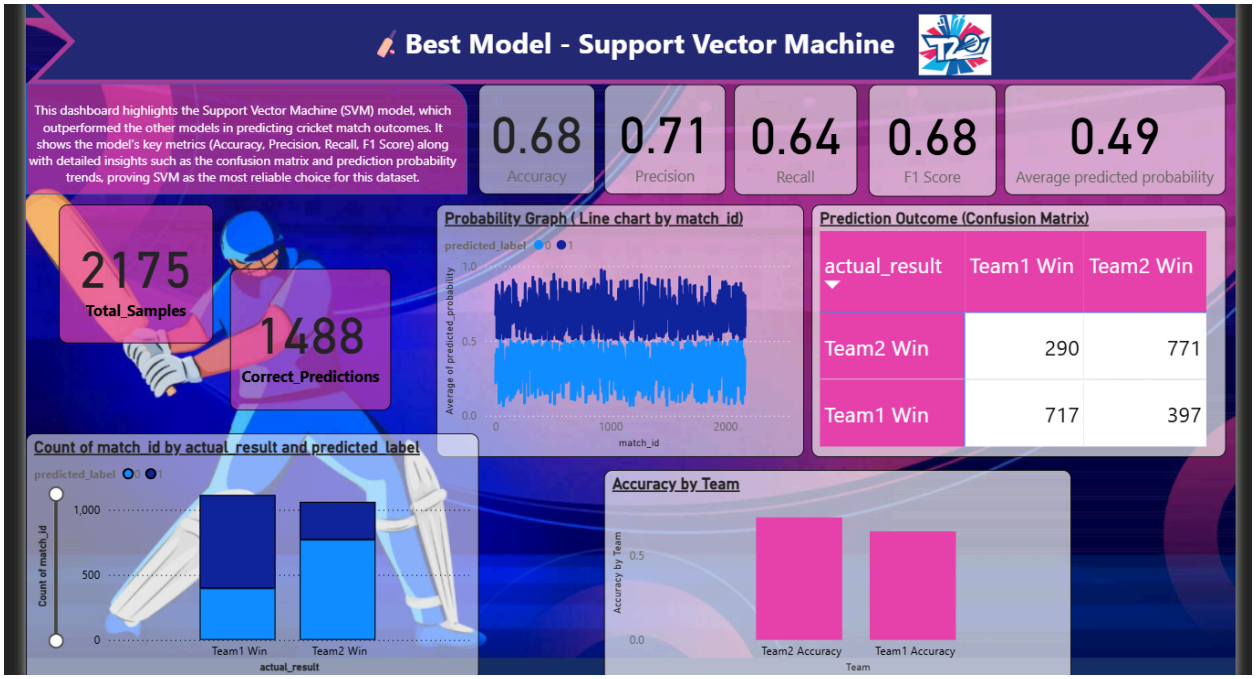



*Figure 5.7: Power BI SVM dashboard with KPI cards and confusion matrix*

Through these dashboards, the results were transformed into decision-ready insights that could be communicated effectively, not only to technical audiences but also to stakeholders with limited data science background.

## 5.4 Deployment Results with Flask API

After identifying the Support Vector Machine (SVM) as the best-performing model, the next step was to deploy it for real-time predictions. This was achieved using a lightweight Flask API. The deployment validated the practical usability of the project, ensuring that predictions were not only computed in a research environment but could also be accessed dynamically.


*Figure 5.8: Screenshot of /health endpoint in browser*

The API included endpoints such as /health, /schema, /template, and /predict. When accessed in the browser, the /health endpoint confirmed that the service was running successfully (Figure 5.8)

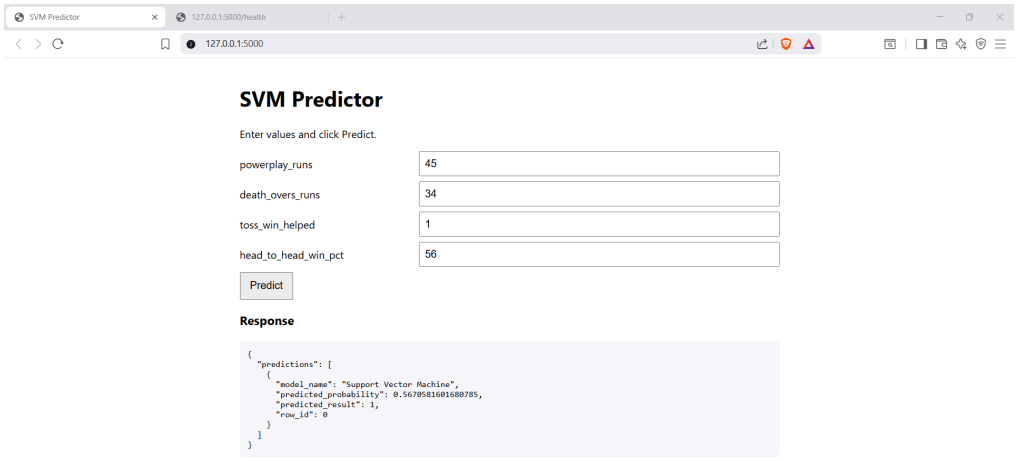



*Figure 5.9: Screenshot of /predict endpoint showing input JSON and output prediction*

The core functionality resided in the /predict endpoint, which accepted a JSON input containing match-specific features. On submission, the API processed the input using the trained SVM model and returned a predicted outcome in JSON format. An example request and response is shown in Figure 5.9, demonstrating that the model could provide instant results for unseen matches.

This deployment step bridged the gap between technical development and real-world application. It showed that the system could be integrated into larger applications, such as sports analytics platforms or dashboards used by coaches, broadcasters, or fans.

## 5.5 Discussion of Findings

The results obtained from the models, dashboards, and deployment highlight several key insights. First, the superiority of the Support Vector Machine (SVM) indicates that, for this dataset and feature set, a margin-based classifier performed better than ensemble approaches such as Random Forest and XGBoost. This outcome can be attributed to the relatively small and compact feature space, where SVM's ability to maximise decision boundaries proved effective.

Secondly, the engineered features, particularly Powerplay Runs and Death Overs Runs emerged as crucial predictors. These variables captured early and late match momentum, which reflects well-established cricket dynamics where strong starts and finishes often decide outcomes. Contextual features such as Toss Win Helped and Head-to-Head Win % also contributed, though their impact was smaller, suggesting that skill-driven performance indicators outweigh chance-based or historical biases when predicting match outcomes.

The Power BI dashboards further reinforced the interpretability of results by transforming raw performance metrics into clear, decision-ready insights. Through KPI cards and comparative charts, both technical and non-technical stakeholders could see why SVM was chosen as the final model. This highlights the importance of combining machine learning outputs with effective reporting and visualisation tools.

Several limitations were also identified. The dataset was restricted to matches played up to 2021, meaning that more recent changes in team composition or playing conditions were not captured. The feature set, while deliberately compact, excluded richer contextual data such as player form, weather conditions, and ground dimensions, all of which may influence outcomes. As a result, the predictive accuracy of SVM (≈68%) demonstrates potential but also leaves scope for improvement through enhanced feature engineering or alternative algorithms.

Importantly, the evaluation also provided answers to key research questions. It confirmed that compact, interpretable features can deliver meaningful predictive accuracy, supporting the idea that models need not rely on highly complex datasets to be useful. It also showed that ensemble methods did not outperform SVM in this context, illustrating that margin-based classifiers can be more effective when working with small, well-engineered feature spaces. Finally, it established

that performance-driven features, particularly momentum indicators, were stronger predictors than chance-based attributes such as the toss, guiding future feature design strategies.

While deep learning and hybrid ensemble approaches were considered during the design phase, they were excluded from this project due to the modest dataset size and the priority placed on interpretability. Similarly, extending the pipeline to other sports was recognised as a feasible direction given its modular structure, but cricket was chosen as the immediate focus. These remain important opportunities for further exploration.

Finally, deployment through Flask demonstrated that the model could be operationalised in practice. The system successfully produced predictions via structured API calls, but scaling it for real-time data feeds or broader deployment would require more robust infrastructure such as FastAPI, Docker, or cloud-based services.

In summary, the findings demonstrate that the project not only achieved its immediate objectives but also generated insights about the relative importance of features, the suitability of algorithms, and the balance between accuracy, interpretability, and deployment. Together, these elements provide both methodological rigour and practical value.

## 5.6 Summary of Results and Discussion

This chapter presented the evaluation and interpretation of the machine learning models developed for cricket match outcome prediction. Logistic Regression provided a simple and interpretable baseline, while ensemble methods such as Random Forest and XGBoost delivered competitive results but ultimately underperformed compared to the Support Vector Machine (SVM). SVM consistently achieved the best balance across all four evaluation metrics — Accuracy, Precision, Recall, and F1-Score — confirming it as the most suitable model for deployment within this project.

The Power BI dashboards enhanced interpretability by translating numerical results into accessible visuals. Through comparative charts and KPI cards, stakeholders could clearly compare the models and understand the justification for selecting SVM. The dashboards also highlighted that momentum-based features, particularly Powerplay Runs and Death Overs Runs, had the strongest predictive influence, while contextual variables such as toss and head-to-head history added value but carried less weight.

The deployment of the SVM model via Flask further demonstrated the practical usability of the system. By exposing predictions through structured API endpoints, the project bridged technical modelling with real-world application, showing how outputs could be integrated into larger analytics platforms. However, limitations were also acknowledged: the dataset excluded recent matches and richer contextual attributes such as player form, weather, or venue size, and the system achieved moderate predictive accuracy (~68%).

Overall, the results demonstrated that compact, interpretable features can generate meaningful predictive accuracy, answering the project's objective of whether a lean feature set could

perform effectively under a time-aware split. They also clarified the comparative strengths of algorithms, with SVM outperforming ensemble methods in this setting, and highlighted the trade-off between simplicity, interpretability, and predictive power. These findings not only validate the design choices made in the project but also point towards future extensions involving richer feature sets, advanced algorithms, and more robust deployment infrastructures.

# Chapter 6 : Project Ethics

## 6.1 Data Sources and Integrity

This project used publicly available ball-by-ball cricket match data from the Cricsheet repository. The dataset contained match-level records such as teams, venues, toss decisions, overs, and results. No private, personal, or sensitive information was included. All data was processed strictly for academic purposes. Matches with incomplete or unclear results were excluded to maintain reliability, and appropriate citation of all sources was ensured.

## 6.2 Data Privacy and Security

As the dataset only included professional match statistics, there were no direct privacy concerns. Nevertheless, responsible practice was followed by avoiding any unauthorised data sources. All project files, including processed datasets, code, and model artefacts, were stored securely on local and University storage. Only non-identifying files required for marking and reproducibility will be retained; intermediate files will be deleted after assessment.

## 6.3 Responsible Use of Machine Learning

Machine learning models may introduce bias if the underlying data is unbalanced or incomplete. To mitigate this, care was taken in feature engineering and evaluation. A chronological train–test split was used to avoid information leakage, and multiple metrics (accuracy, precision, recall, and F1-score) were reported to provide a fair view of performance. Limitations of the models were acknowledged openly, and no attempt was made to manipulate results.

## 6.4 Deployment and Accessibility

The project included deployment of the best-performing model using a Flask API. This was developed as a prototype tool for academic demonstration and learning. It is not intended for commercial use or for applications such as sports betting or financial exploitation. Ethical responsibility was taken to present the system strictly as a research artefact.

### 6.5 Academic Integrity

All datasets, libraries, and literature sources were fully cited in line with academic standards. The work is original and free from plagiarism or falsification of results. Algorithms and models were implemented in accordance with academic integrity and professional practice. The project complies with the University's ethical guidelines and falls under Category A0 of the ethical framework, requiring no further approval.

# Chapter 7 : Project Learnings

During the course of this project, several key technical and personal learnings were gained. These learnings not only strengthened my understanding of machine learning and deployment but also improved my ability to manage end-to-end projects.

- **Applying Machine Learning in Sports Analytics:** I learned how to use algorithms such as Logistic Regression, Random Forest, XGBoost, and SVM for cricket outcome prediction. This gave me insight into how different models handle structured sports data and how evaluation metrics (accuracy, precision, recall, and F1-score) provide a more complete picture of model performance.

- **Importance of Data Preparation:** I understood the critical role of data preprocessing and feature engineering. Working with cricket datasets highlighted the importance of handling missing values, balancing features such as toss outcome, player statistics, and match history, and preparing the dataset for training and testing.

- **Model Comparison and Interpretation:** I gained practical skills in comparing multiple models to identify the most suitable one. Through experimentation, I learned that simple models like Logistic Regression provide a strong baseline, while advanced models like XGBoost often deliver higher accuracy but require careful tuning.

- **Deployment Using Flask:** I developed skills in integrating machine learning models into a Flask API. This showed me how predictive systems can move beyond theory into usable applications, allowing end-users to interact with the models in real time.

- **Visualisation with Power BI:** I learned how to design dashboards that communicate results effectively. Creating comparison pages for the four models helped me appreciate the role of visualisation in explaining technical results to non-technical stakeholders.

- **Research and Documentation Skills:** The project improved my ability to critically analyse academic literature, structure research findings, and document processes clearly. This strengthened my academic writing and project reporting skills.

Overall, this project provided a strong foundation in combining machine learning with deployment and visualisation, preparing me to apply these skills in real-world data science projects.

# Chapter 8 : Challenges Faced

Like any research and development project, several challenges were encountered during the implementation of cricket outcome prediction. These challenges shaped the direction of the project and provided valuable problem-solving experience.

- **Data Limitations:** One of the biggest challenges was working with incomplete and unstructured cricket data. Missing values, inconsistent match records, and the lack of player-specific attributes initially limited the dataset quality. This required additional preprocessing and filtering to ensure reliability.

- **Model Selection and Tuning:** Choosing the right algorithms was not straightforward. Some models, such as SVM, performed inconsistently when the dataset size grew, while others required significant hyperparameter tuning to avoid overfitting. Balancing accuracy with interpretability was also a challenge.

- **Deployment Issues:** While integrating the models with Flask, there were difficulties in handling JSON input/output formats and ensuring smooth interaction between the backend model and the web API. Debugging these issues required patience and iterative testing.

- **Performance Evaluation:** Ensuring fair evaluation across multiple models was challenging. Different models responded differently to the same data split, and results varied depending on cross-validation strategies. This required consistent use of metrics and validation methods.

- **Time and Resource Constraints:** Training advanced models like XGBoost was computationally demanding. Limited resources sometimes slowed experimentation, and optimising the workflow was necessary to complete the project within the available timeframe.

- **Visualisation Design:** Creating effective Power BI dashboards required careful selection of KPIs and chart types. Initially, some dashboards were too complex for easy interpretation, leading to multiple redesigns before achieving clarity.

Despite these challenges, each obstacle provided an opportunity to learn, adapt, and refine the project. Overcoming them not only improved the quality of the final system but also built confidence in tackling complex real-world problems.

# Chapter 9 : Conclusion & Future Work

## 9.1 Conclusion

This project set out to design and implement a complete machine learning pipeline for predicting cricket match outcomes. Using multiple algorithms including Logistic Regression, Random Forest, XGBoost, and SVM, the system compared models based on accuracy, precision, recall, and F1-score. The findings showed that while Logistic Regression provided a simple and interpretable baseline, ensemble methods such as Random Forest and XGBoost achieved higher predictive accuracy.

To ensure practical usability, the project deployed the trained model using Flask as a web-based API, enabling real-time predictions. In addition, interactive dashboards were created in Power BI, allowing stakeholders to visualise model outputs and compare performance metrics across algorithms.

The results demonstrated that machine learning can effectively capture patterns in cricket data and provide reliable outcome predictions. The combination of modelling, deployment, and visualisation makes the project valuable not only as a research exercise but also as a practical tool that bridges technical performance with user accessibility.

## 9.2 Future Work

Although the project achieved its main objectives, there are several directions in which it could be expanded and improved:

- **Larger and Richer Datasets:** Future work can incorporate more detailed player-level statistics, pitch conditions, weather data, and historical match context to improve prediction accuracy and robustness.

- **Advanced Algorithms:** Deep learning models such as neural networks or hybrid ensembles could be explored to capture complex interactions in cricket datasets, potentially improving long-term prediction reliability.

- **Real-Time Data Integration:** The current system uses pre-prepared datasets. Future versions could integrate live match feeds or API-based data sources to provide real-time predictions during ongoing games.

- **Enhanced Deployment:** While Flask provided a simple backend, deploying the model on cloud platforms (e.g., AWS, Azure, or Google Cloud) could enable scalability and support a larger user base.

- **Improved Visualisation:** Power BI dashboards could be expanded to include player impact analysis, scenario simulations (e.g., “What if the toss is lost?”), and fan

engagement features for wider usability.

- **Cross-Sport Applications:** The methodology developed for cricket can be adapted to other sports such as football, basketball, or tennis, demonstrating the generalisability of the pipeline.

If more time and richer data were available, the project could be extended to include contextual and player-level features such as batting strike rates, bowling economy, weather conditions, and venue characteristics. These variables could add greater predictive power and make the models more robust to diverse match scenarios

# Chapter 10 : BCS Criteria & Self Reflection

## BCS Outcomes

- **Ability to apply practical and analytical skills gained during the degree programme**

  ➢ Applied data analysis, SQL querying, and machine learning modelling.

  ➢ Used practical tools (Python, scikit-learn, XGBoost, Power BI, Flask).

  ➢ *See Chapters 3 (System Design), 4 (Implementation), and 5 (Evaluation).*

- **Innovation and/or creativity**

  ➢ Developed an end-to-end pipeline for sports analytics using academic concepts in a practical application.

  ➢ Integrated machine learning predictions into a real-time Flask API and Power BI dashboards.

  ➢ *See Chapter 4.7 (API Design) and Chapter 5.3 (Insights from Power BI).*

- **Synthesis of information, ideas and practices to provide a quality solution together with an evaluation of that solution**

  ➢ Combined ideas from literature with technical practices in SQL, ML, BI, and API deployment.

  ➢ Evaluated models using multiple metrics and dashboards to ensure quality.

  ➢ *See Chapters 2 (Literature Survey), 5 (Results), and 9 (Conclusion).*

- **Meeting a real need in a wider context**

➢ Demonstrated how machine learning can support decision-making in cricket, a globally followed sport.

➢ Showed potential applications for team analysts, broadcasters, and fan engagement.

➢ *See Chapter 1 (Introduction) and Chapter 9 (Conclusion & Future Work).*

## Critical Reflection (Project Management & Self-Evaluation)

Managing this project from start to finish was a challenging but rewarding experience. It required me to self-manage a large piece of work over several months and to critically evaluate my own approach.

On the positive side, I believe I demonstrated good time management by breaking the project into clear milestones (data preparation, feature engineering, model training, evaluation, and deployment). I also showed strong technical adaptability, as I had to learn and apply different tools quickly: SQL for data engineering, Python for modelling, Power BI for dashboards, and Flask for deployment. Successfully integrating all of these into one pipeline was a highlight for me.

Another strength was my ability to remain focused and consistent. At times, I felt overwhelmed by the number of possible directions, adding player-level features, testing deep learning models, or integrating external data  but I managed to stay on scope and deliver what was achievable within the timeframe. This ensured that my work remained reproducible and submission-ready.

However, I also identified some weaknesses. Initially, I underestimated the effort required for data preprocessing, particularly in cleaning and standardising cricket match data. This cost me additional time and forced me to simplify some aspects of my feature design. In addition, while I implemented hyperparameter tuning, I did not explore advanced optimisation strategies (e.g., Bayesian optimisation, automated ML pipelines), which could have further improved results.

Looking back, if I had more time and richer data, I would have added contextual and player-level features such as strike rates, bowling economy, weather, and venue size. These additions could have produced a more robust prediction model.

Despite these challenges, I believe I have developed as both an independent researcher and a practitioner. I learned to manage uncertainty, solve problems creatively, and balance technical depth with practical constraints. Most importantly, I strengthened my ability to critically reflect on my own work, acknowledging both successes and limitations.

This project has given me confidence in my ability to deliver a significant piece of independent work. It has also taught me to be more realistic about scope, more thorough with early-stage

data cleaning, and more critical when evaluating my own decisions. These lessons will help me in both academic and professional contexts going forward.

# References Cited


[1] IEEE, (2024). *A novel machine learning framework for cricket outcome prediction*. IEEE Xplore. [Online]. Available at: https://ieeexplore.ieee.org/document/10581259 [Accessed 11 Sept. 2025].

[2] IEEE, (2020). *Cricket outcome prediction using artificial intelligence and machine learning*. IEEE Xplore. [Online]. Available at: https://ieeexplore.ieee.org/document/9033698 [Accessed 11 Sept. 2025].

[3] Applied Computing and Informatics, (2022). *Sport analytics for cricket game results using machine learning: An experimental study*. Emerald Publishing. [Online]. Available at: https://www.emerald.com/aci/article/18/3-4/256/6038/Sport-analytics-for-cricket-game-results-using [Accessed 11 Sept. 2025].

[4] Adypsoe, (2020). *Cricket prediction research paper*. [Online]. Available at: https://adypsoe.in/naac2/cr-3/research_papers/183.pdf [Accessed 11 Sept. 2025].

[5] Techieyan Technologies, (2023). *Score prediction of IPL team with Flask app*. [Online]. Available at: https://techieyantechnologies.com/score-prediction-of-ipl-team-with-flask-app/ [Accessed 11 Sept. 2025].

[6] IRJMETS, (2024). *Cricket prediction using AI and ML*. International Research Journal of Modernization in Engineering, Technology and Science. [Online]. Available at: https://www.irjmets.com/uploadedfiles/paper//issue_1_january_2025/66640/final/fin_irjmets1737781673.pdf [Accessed 11 Sept. 2025].

[7] IEEE, (2024). *Cricket analytics using ML approaches*. IEEE Xplore. [Online]. Available at: https://ieeexplore.ieee.org/document/11005108 [Accessed 11 Sept. 2025].

[8] GeeksforGeeks, (2022). *Flask tutorial*. [Online]. Available at: https://www.geeksforgeeks.org/python/flask-tutorial/ [Accessed 11 Sept. 2025].

[9] IEEE, (2021). *Player's performance prediction in ODI cricket using machine learning algorithms*. IEEE Conference Publication. [Online]. Available at: https://ieeexplore.ieee.org/document/8628118 [Accessed 11 Sept. 2025].

[10] Koul, R., (2020). *Cricket prediction using machine learning algorithms*. ResearchGate. [Online]. Available at: https://www.researchgate.net/publication/346085808_Cricket_Prediction_using_Machine_Learning_Algorithms [Accessed 11 Sept. 2025].

[11] Daroch, U., (2021). *International T20 game outcome predictor*. GitHub. [Online]. Available at: https://github.com/udaydaroch/InternationalT20GameOutcomePredictor [Accessed 11 Sept. 2025].

[12] Sharma, V., (2021). *Predict the match outcome in cricket matches using machine learning*. [PDF]. [Online]. Available at: https://www.researchgate.net/publication/373690614_Predict_the_Match_Outcome_in_Cricket_Matches_Using_Machine_Learning [Accessed 11 Sept. 2025].

[13] TutorialsPoint, (2019). *Power BI tutorial*. [Online]. Available at: https://www.tutorialspoint.com/power_bi/index.htm [Accessed 11 Sept. 2025].

[14] Singh, A., (2024). *A comparative evaluation of machine learning algorithms for predicting match outcomes in white-ball cricket*. ResearchGate. [Online]. Available at: https://www.researchgate.net/publication/392266064 [Accessed 11 Sept. 2025].

[15] Dalal, R., (2024). *Machine learning-based prediction in cricket outcomes*. International Journal of Computer Applications, 186(26). [Online]. Available at: https://www.ijcaonline.org/archives/volume186/number26/dalal-2024-ijca-923744.pdf [Accessed 11 Sept. 2025].

[16] Srivastava, S., (2020). *Machine learning in sports outcome prediction*. SSRN. [Online]. Available at: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3572740 [Accessed 11 Sept. 2025].

[17] BMJ, (2017). *Cricket analytics in sports science*. British Journal of Sports Medicine, 51(Suppl 2), A3.1. [Online]. Available at: https://bjsm.bmj.com/content/51/Suppl_2/A3.1 [Accessed 11 Sept. 2025].

[18] Gupta, P., (2018). *Increased prediction accuracy in the game of cricket using machine learning*. ResearchGate. [Online]. Available at: https://www.researchgate.net/publication/324254587_Increased_Prediction_Accuracy_in_the_Game_of_Cricket_Using_Machine_Learning [Accessed 11 Sept. 2025].

[19] Kumar, R., (2024). *Predicting cricket outcomes: A comparative analysis of machine learning models for optimal accuracy*. ResearchGate. [Online]. Available at: https://www.researchgate.net/publication/389037314 [Accessed 11 Sept. 2025].

[20] Patel, S., (2023). *Cricket win prediction using machine learning*. ResearchGate. [Online]. Available at: https://www.researchgate.net/publication/375113707_Cricket_Win_Prediction_using_Machine_Learning [Accessed 11 Sept. 2025].

# Appendices

## Appendix A: Data Preparation Screenshots

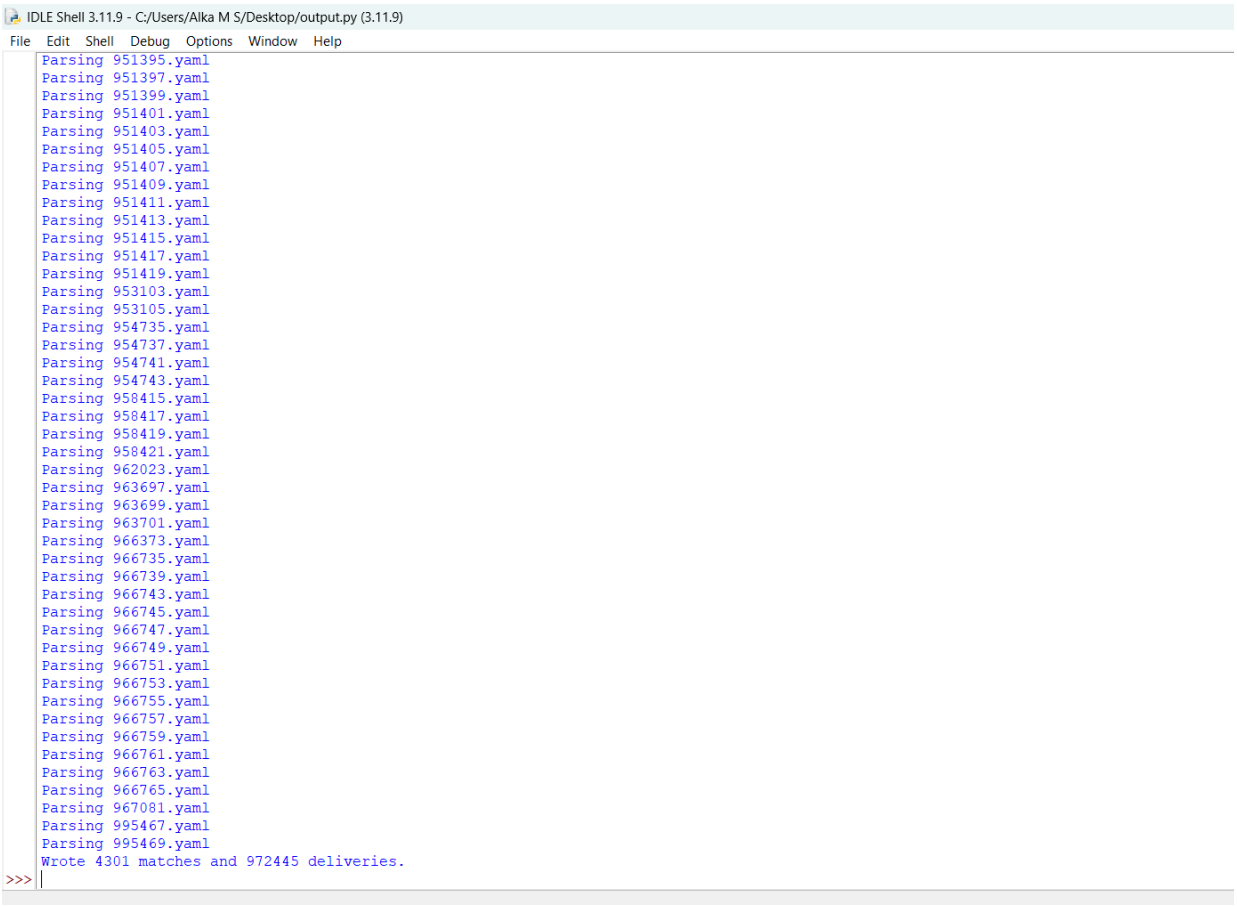



*Figure A.1: Parsing Cricsheet YAML files into structured CSVs*

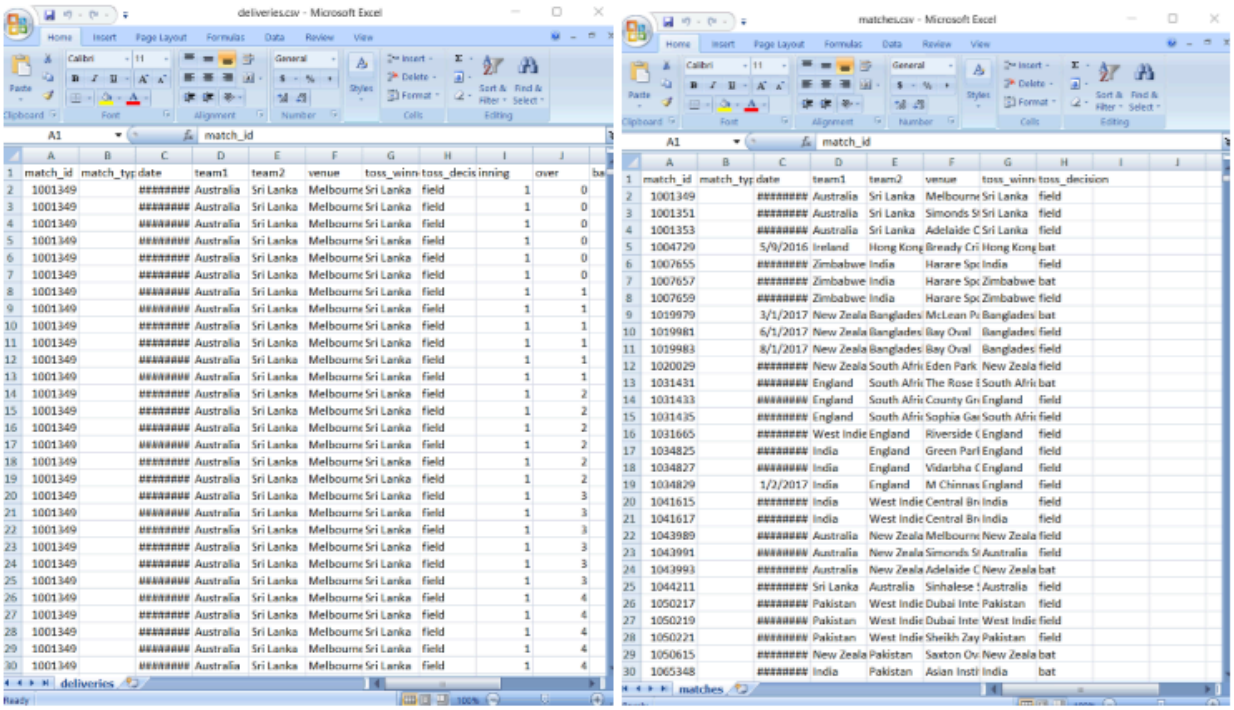
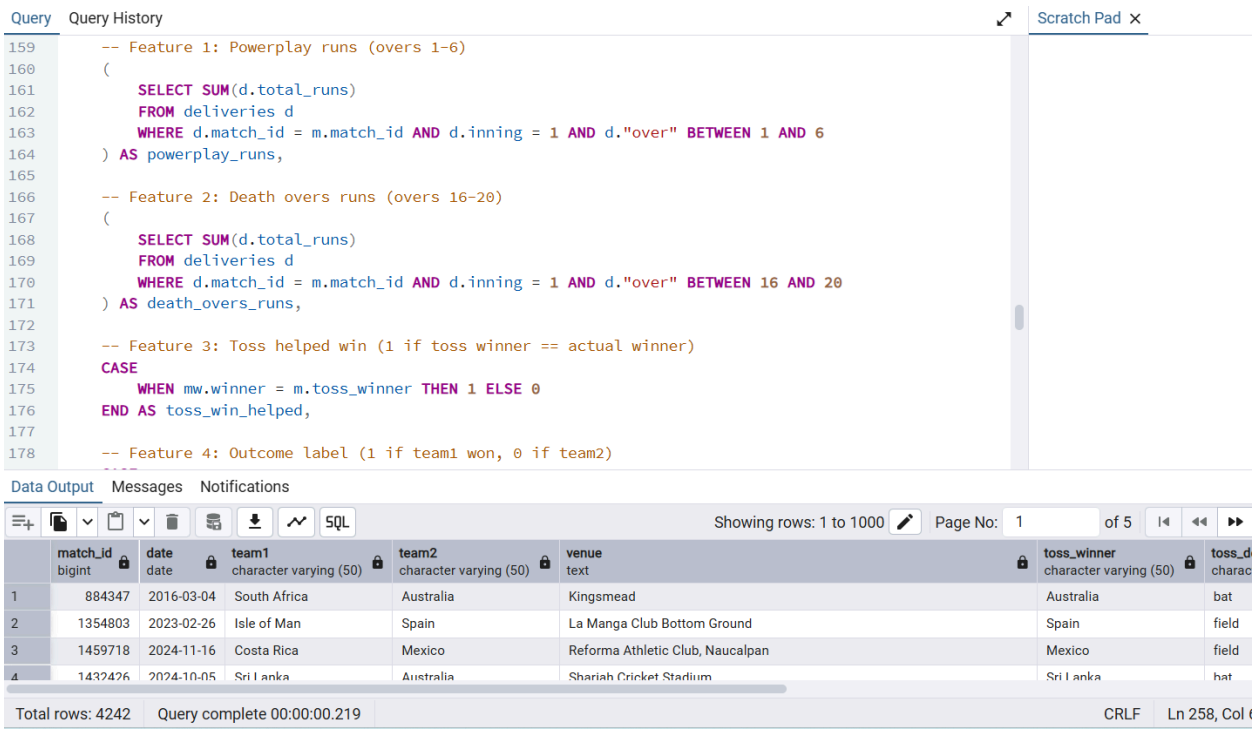


*Figure A.2: Sample output of parsed CSV with cleaned match records*



```
159     -- Feature 1: Powerplay runs (overs 1-6)
160     (
161         SELECT SUM(d.total_runs)
162         FROM deliveries d
163         WHERE d.match_id = m.match_id AND d.inning = 1 AND d."over" BETWEEN 1 AND 6
164     ) AS powerplay_runs,
165
166     -- Feature 2: Death overs runs (overs 16-20)
167     (
168         SELECT SUM(d.total_runs)
169         FROM deliveries d
170         WHERE d.match_id = m.match_id AND d.inning = 1 AND d."over" BETWEEN 16 AND 20
171     ) AS death_overs_runs,
172
173     -- Feature 3: Toss helped win (1 if toss winner == actual winner)
174     CASE
175         WHEN mw.winner = m.toss_winner THEN 1 ELSE 0
176     END AS toss_win_helped,
177
178     -- Feature 4: Outcome label (1 if team1 won, 0 if team2)
```

| | match_id bigint | date date | team1 character varying (50) | team2 character varying (50) | venue text | toss_winner character varying (50) | toss_d charac |
|---|---|---|---|---|---|---|---|
| 1 | 884347 | 2016-03-04 | South Africa | Australia | Kingsmead | Australia | bat |
| 2 | 1354803 | 2023-02-26 | Isle of Man | Spain | La Manga Club Bottom Ground | Spain | field |
| 3 | 1459718 | 2024-11-16 | Costa Rica | Mexico | Reforma Athletic Club, Naucalpan | Mexico | field |
| 4 | 1432426 | 2024-10-05 | Sri Lanka | Australia | Sharjah Cricket Stadium | Sri Lanka | bat |

*Figure A.3: Sample SQL query for feature engineering*

# Appendix B: Machine Learning Implementation

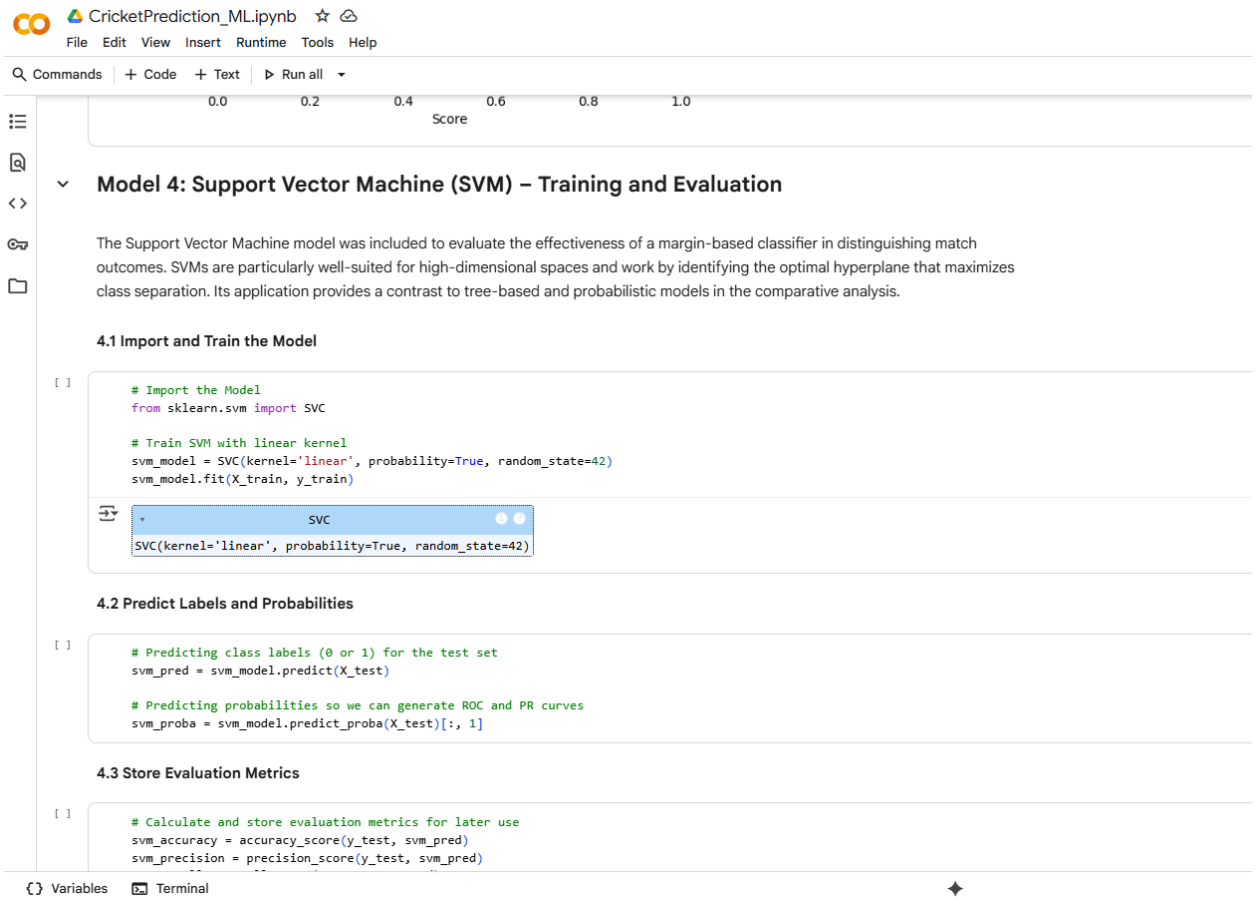



*Figure B.1: Training SVM model in Google Colab*

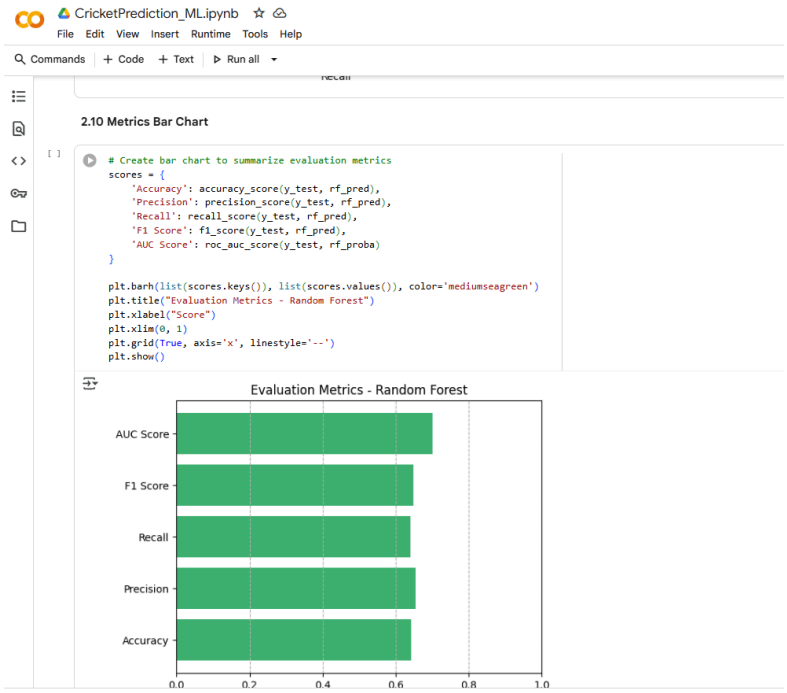



*Figure B.2 : Evaluation Metrics of Random Forest*

# Appendix C: Model Results

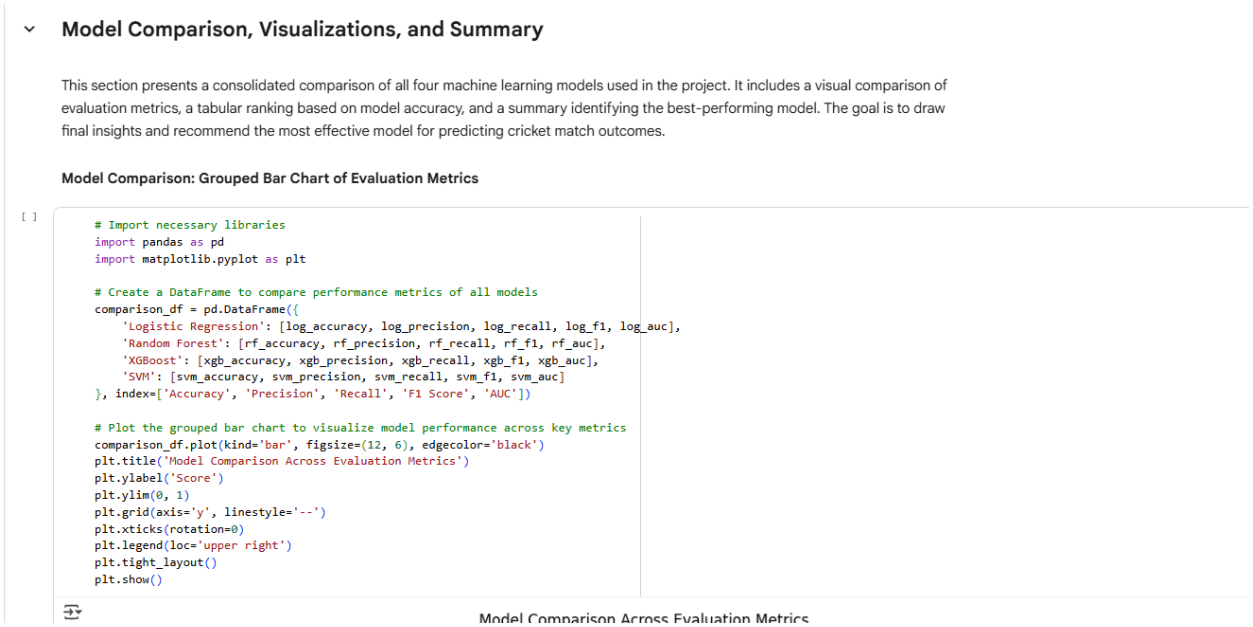



*Figure C.1: Code for comparison across four models*

## Appendix D: Power BI Dashboards

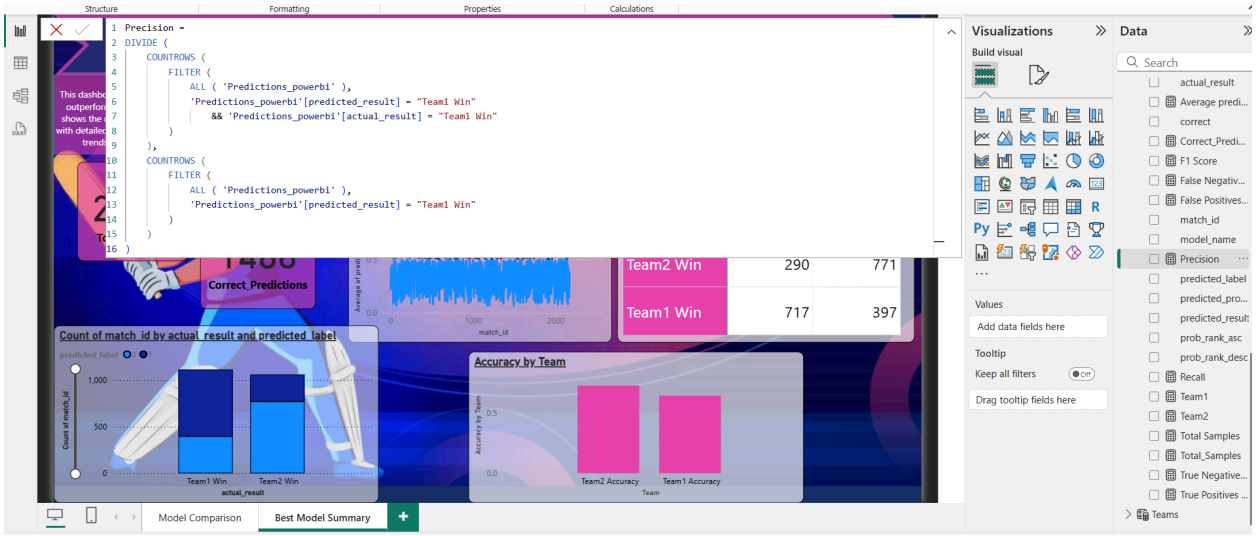



*Figure D.1: Sample DAX formula to display the precision Figure*

## Appendix E: Flask API Deployment

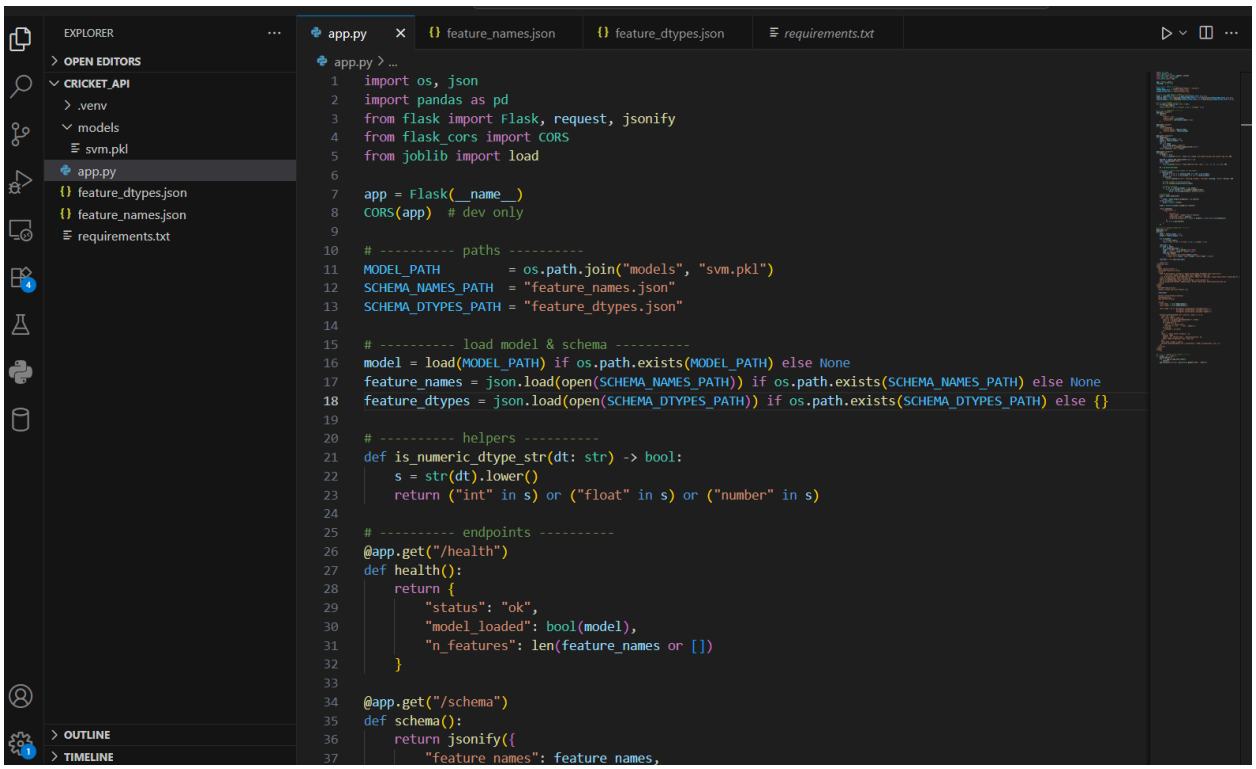



*Figure E.1: Sample code of the Flask API (app.py) showing model loading, schema handling, and endpoint definitions*

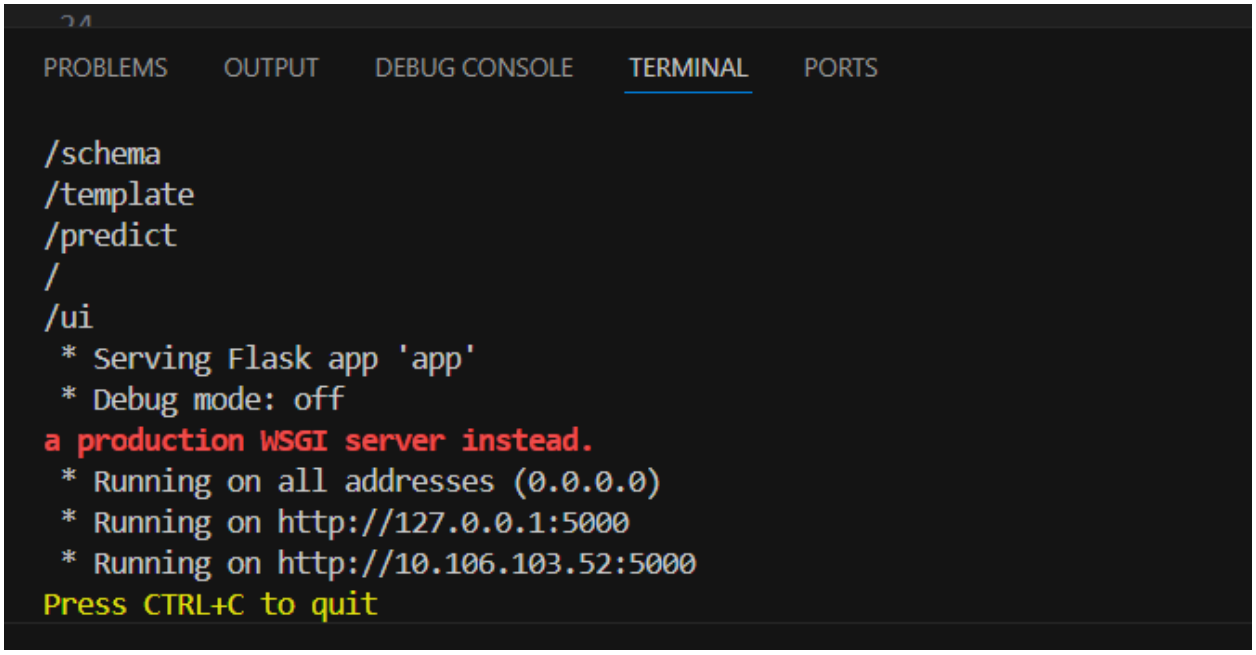



*Figure E.2: Flask API running in the VS Code terminal, showing endpoints and local server availability*