Cricket Innings Score Prediction Using Machine Learning Algorithms

High National Diploma in Software Engineering

Machine Learning for Artificial Intelligence

Final Report Documentation

23.2F

S M A D V D SAMMANDAPPERUMA	COHNDSE232F-044
K D R SILVA	COHNDSE232F-045
W M D P Weerakoon	COHNDSE232F-091
K A I Perera	COHNDSE232F-100

School of Computing and Engineering

National Institute of Business Management

Colombo – 7

Abstract

Predicting the first innings score in T20 cricket matches is a challenging and useful task. It helps in understanding the game better, making decisions during matches, and providing understanding for fans and analysts. This project uses machine learning techniques to predict scores based on historical match data.

The dataset has detailed information about each ball and overall match statistics. To prepare the data, we cleaned it and created useful features like runs, wickets, overs, batting team, bowling team, runs in last 5 overs. We also analyzed the data to find important patterns that affect scores.

We tested different machine learning models, including Linear Regression, Random Forest, Gradient Boosting, XGB Regressor, and Artificial Neural Networks (ANN). These models were evaluated using measures like how close the predictions were to actual scores. Among them, Gradient Boosting and XGB Regressor worked the best because they could handle complex data relationships.

This study shows how machine learning can help predict cricket scores accurately. In the future, we could make the models even better by adding real-time match data and player performance. This project proves how technology can bring new insights into sports like cricket.

Introduction

1. Cricket is one of the most loved sports around the world, especially the fast-paced T20 format. Predicting the score of a team's first innings can help teams make better decisions during the match. It is also useful for fans, analysts, and fantasy cricket platforms to understand and enjoy the game better. A good score prediction can provide insights into how the match might progress, making the game even more exciting.

Figure 1:T20 Match

1.1 Problem Statement

Cricket first-inning scores are difficult to predict because of various factors, including pitch condition, player form, and weather. Cricket is a difficult game to study because of these aspects. We can identify trends in past match data and generate precise forecasts using machine learning techniques. This can assist broadcasters with analysis, teams with strategy, and fantasy sports platforms with user experience enhancement.

1.2 Objectives

The main goal of this project is to Predict the score of the first inning of a T20 match using a machine learning model.

We trained different models using algorithms such as Linear Regression, Random Forest Regression, XGBooster Regression, Gradient Boosting, and Artificial Neural Network.

1.3 Dataset Overview

The dataset used for this project was taken from Kaggle. It contains detailed match data, including ball-by-ball statistics like runs scored, wickets lost, overs bowled, and details about the players and teams. These features are used to train the machine learning models to understand how scores are built in a T20 match. Before using the data, it was cleaned and prepared by handling missing values and creating new features like run rate and performance in specific overs.

1.4 Challengers

Predicting the first-innings score in T20 cricket matches comes with several challenges:

1. **Dynamic Nature of the Game**
   T20 cricket is highly unpredictable due to the fast-paced format. Factors like sudden collapses, powerplays, and death-over performances can significantly impact scores, making prediction difficult.

2. **Limited Data**
   While historical match data is available, the variability in player performance, weather conditions, and pitch behavior may not always be fully captured, leading to gaps in the dataset.

3. **Feature Selection**
   Identifying the most important features, such as player form, team strategies, and match conditions, is crucial for building accurate models but can be challenging due to the complexity of the game.

4. **Data Imbalance**
   Certain match scenarios, such as low-scoring or extremely high-scoring games, might be underrepresented in the dataset, causing the model to struggle in predicting these cases accurately.

5. **Real-Time Adaptation**
   Cricket matches are influenced by real-time factors like weather changes, injuries, and on-field decisions. Incorporating such real-time data into the model is complex and often unavailable during training.

6. **Overfitting**
   With detailed ball-by-ball data, models can sometimes be overfit to the training data, reducing their ability to generalize to unseen matches.

7. **Handling Nonlinear Relationships**
   Cricket performance metrics often involve nonlinear relationships, such as how the strike rate increases towards the end of an innings. Capturing these dynamics requires sophisticated algorithms and careful tuning.

8. **External Factors**
   Variables like crowd pressure, dew, and player fatigue are difficult to quantify but can significantly affect scores, leading to potential inaccuracies in predictions.

2. Literature Review

2.1 Related Works

Several studies and projects have explored the use of machine learning in sports analytics, particularly in cricket. Early efforts primarily relied on statistical techniques to analyze match data and predict outcomes. Recent advancements in machine learning have enabled more accurate predictions by leveraging large datasets and complex algorithms. For example, models like regression, Random Forest, and neural networks have been used to predict match results, player performances, and even batting or bowling strategies. However, predicting first-innings scores in T20 cricket remains less explored due to the format's fast-paced and unpredictable nature.

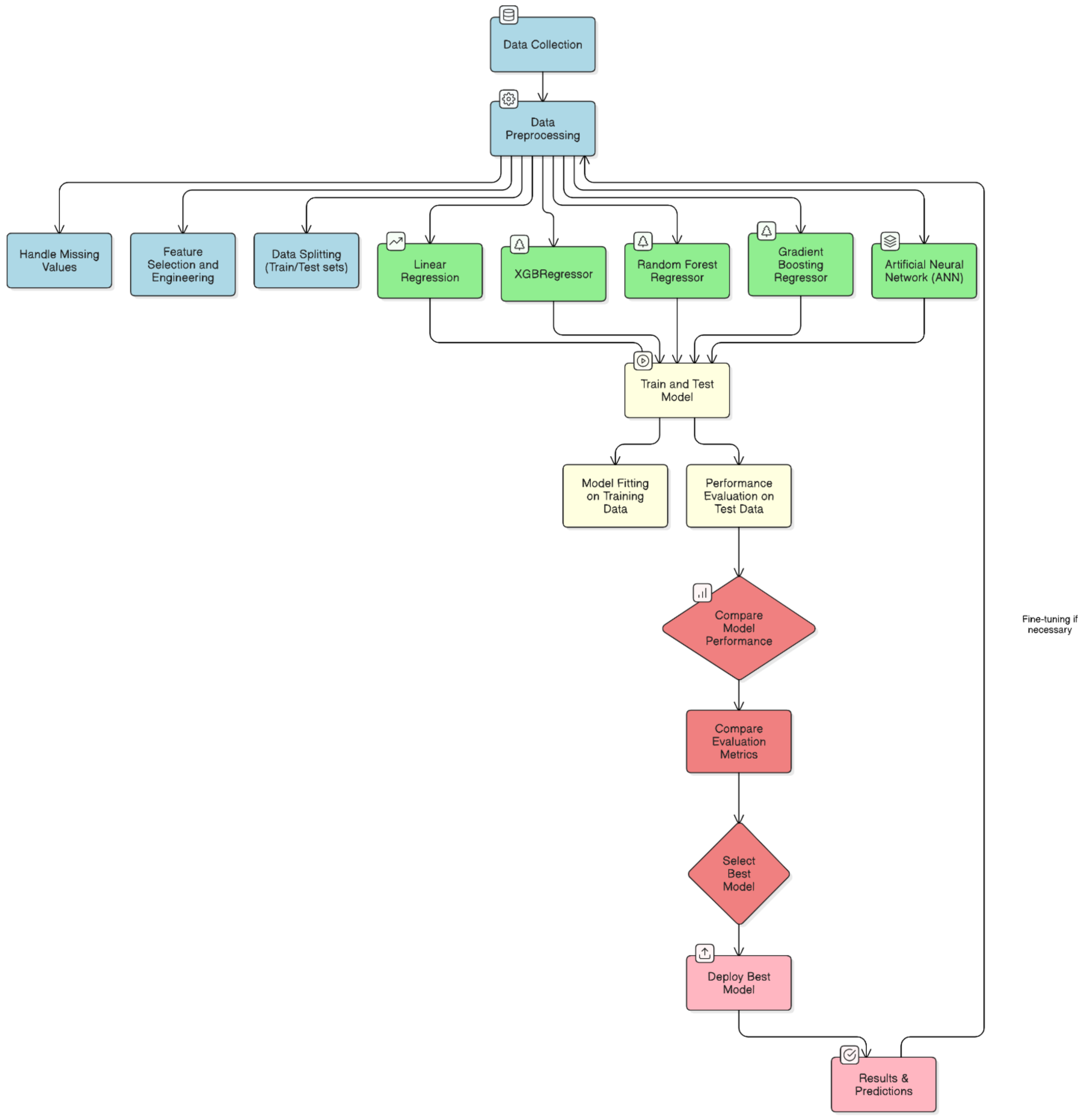
2.2 Machine Learning in Sports Analytics

Machine learning has become a vital tool in sports analytics, offering new insights into performance metrics, game strategies, and outcome predictions. In cricket, machine learning applications include player performance tracking, injury prediction, and real-time strategy formulation. Advanced algorithms such as Gradient Boosting and XGBoost are widely used for their ability to handle nonlinear relationships and large datasets. Artificial Neural Networks (ANN) are also gaining popularity for their capacity to model complex patterns, such as the impact of overs, strike rates, and partnerships on scores. These advancements highlight the growing potential of machine learning to transform how sports data is analyzed and utilized.

2.3 Why use the amount of runs in the last 5 overs as an input parameter?

We can get a clear snapshot of the team’s recent scoring moment and pace by looking at how many runs they scored in the last 5 overs. For example, if the current over is 13, this means the runs scored from over 8 to over 12. This period helps the model understand if the team is maintaining a good run rate or building a solid base for the later overs. Adding this feature makes the prediction more accurate by showing how consistently the team is performing.

3. Methodologies



Data Collection
Data Preprocessing
Handle Missing Values
Feature Selection and Engineering
Data Splitting (Train/Test sets)
Linear Regression
XGBRegressor
Random Forest Regressor
Gradient Boosting Regressor
Artificial Neural Network (ANN)
Train and Test Model
Model Fitting on Training Data
Performance Evaluation on Test Data
Compare Model Performance
Compare Evaluation Metrics
Select Best Model
Deploy Best Model
Results & Predictions
Fine-tuning if necessary

Figure 2:Work flow diagram

3.1Data Preprocessing

Data preprocessing is an essential step to clean and prepare the dataset for model training. And these are the steps we did in data preprocessing:

1. Handling Missing Values: This dataset had some missing values. Therefore we removed columns with missing values or it the column was not needed such as ‘city’. This ensures that this model only trained on accurate and complete information.
2. Feature Engineering: Feature engineering was used to create new variables that provided more useful insights for the model. The added features are:
   - Overs Completed: Current over number which helps the model to understand the current stage of the inning.

- Runs in the last 5 Overs: This is runs scored in the last 5 overs which helps the model to understand the team scoring pace.

3. Data Normalization and Scaling: Features like runs, overs, and strike rates had different ranges of values, which could affect the model's performance. These features were scaled to ensure all inputs were in a similar range. This helped models like Neural Networks perform better by preventing large numbers from dominating small ones.

3.2Exploratory Data Analysis

EDA was conducted to better understand the dataset and uncover patterns that could help improve the model.

- Teams that score steadily in the middle overs (overs 7-15) often achieve higher total scores.

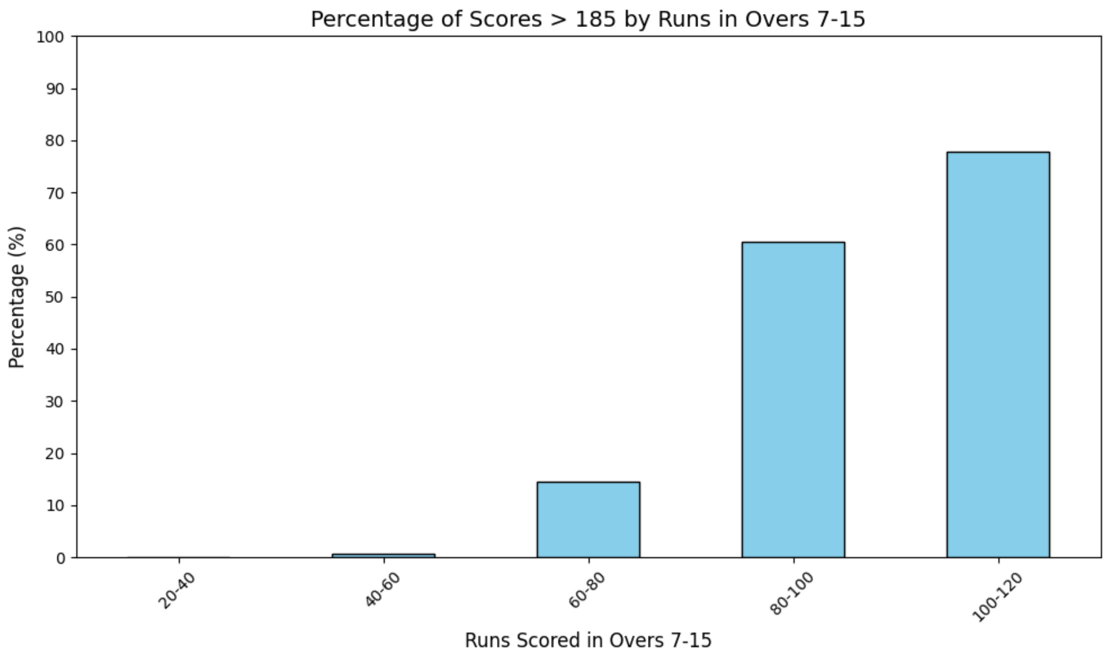


Figure 3:Distribution of Percentage of Scores Greater than 185 Based on Runs Scored in Overs 7-15

- Losing too many wickets early in the innings usually leads to lower final scores.

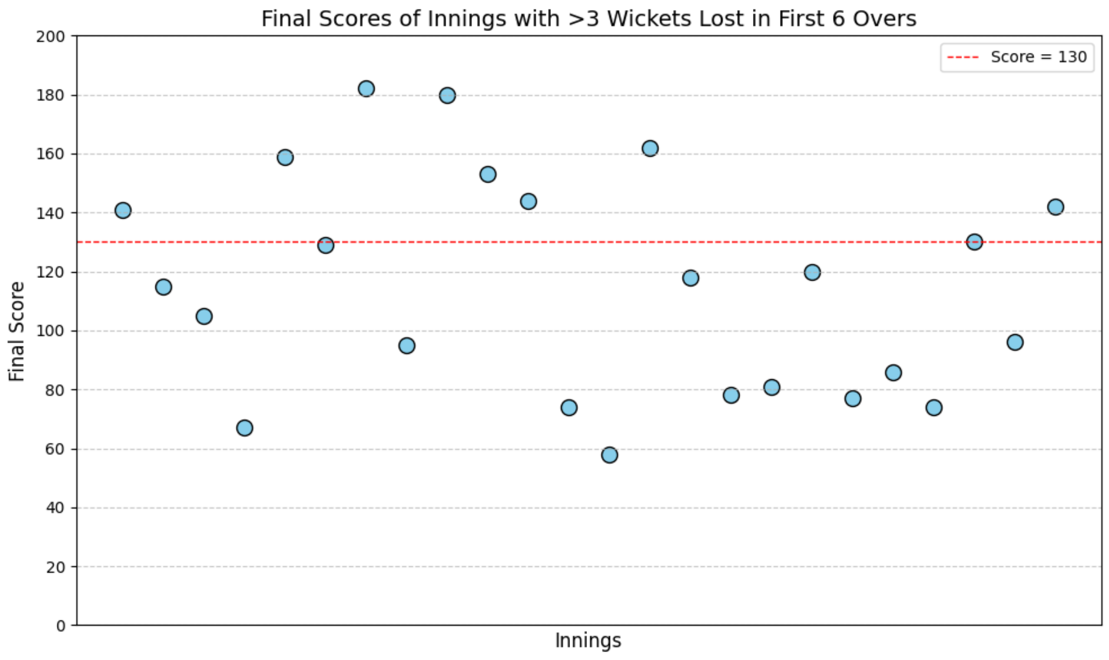


Figure 4:Scatter Plot of Final Scores for Innings with More Than 3 Wickets Lost in the First 6 Overs

- The runs scored in the last 5 overs can have a major impact on the final score, as teams try to accelerate their scoring.

3.3Model Selection

We chose Several machine learning models for this project. Each model has unique strengths, making them suitable for different aspects of the problem:

1. Linear Regression

   Linear Regression was chosen because it is simple and easy to interpret. It works well when there is a straight-line relationship between inputs and outputs. For example, it can easily identify how features like "current runs" affect the final score.

2. Random Forest Regression

   Random Forest was selected because it handles complex relationships between features and can measure feature importance. It combines many decision trees to improve prediction accuracy. This model is especially good at understanding non-linear patterns, such as how "wickets lost" impacts the final score.

3. Gradient Boosting Regression

   Gradient Boosting builds models iteratively, learning from mistakes in the previous step. It was included because it is effective for capturing small improvements in predictions and works well on datasets with many features.

4. XGBooster Regression

   XGBRegressor is a high-performance version of Gradient Boosting. It is optimized to handle large datasets efficiently and produce accurate predictions. It was selected because it is fast and can handle complex patterns, like changes in scoring trends across different overs.

5. Artificial Neural Networks (ANN)

   ANN was chosen because it can capture complex and non-linear relationships that other models might miss. The model used multiple layers, activation functions, and optimization techniques to predict scores. ANN is particularly useful for identifying patterns in features like "runs in the last 5 overs" and "current run rate."

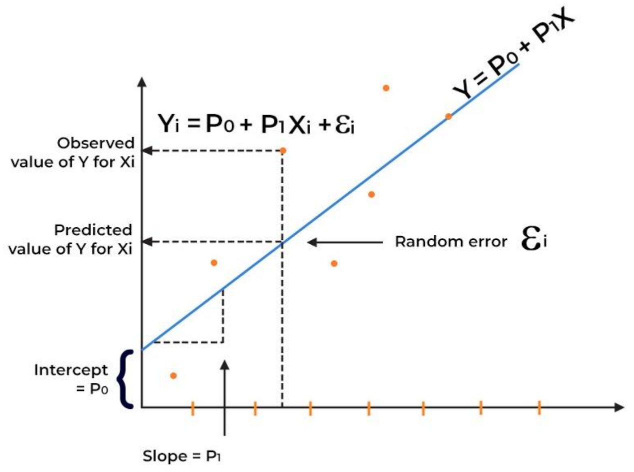
3.4Algorithms and Techniques

In this section, we explain the algorithms used for predicting first-innings scores in T20 cricket matches. We discuss how each algorithm works, its strengths, and its limitations, with a focus on their application in this project.

3.4.1 Linear Regression

Linear Regression is one of the simplest and most widely used algorithms for regression tasks. It assumes a linear relationship between the input features (independent variables) and the target variable (dependent variable). The algorithm works by finding the best-fit line that minimizes the sum of squared differences between the actual values and the predicted values. This method is called the least squares technique.

In this project, the Linear Regression model was implemented using a pipeline that included data transformation, feature scaling, and the regression model itself. After training the model on the dataset, it predicted total scores based on features like current runs, current over, wickets lost, and runs in the last 5 overs.



$Y = P_0 + P_1X$
$Y_i = P_0 + P_1X_i + \varepsilon_i$
Observed value of Y for Xi
Predicted value of Y for Xi
Random error ε_i
Intercept = P_0
Slope = P_1

Figure 5:Visual representation of a simple linear regression model

Advantages:

- Easy to interpret and explain
- Efficient to train
- Effective for datasets that have clear linear datasets

Limitations

- Performs poorly when the relationship between features and the target variable is non-linear.
- Sensitive to outliers, which can distort the model's predictions.

3.4.2 Random Forest Regressor

Random Forest is a kind of ensemble learning that uses many decision trees to predict. Each tree is grown on a random subset of the data using a technique called bootstrapping. The final prediction is done by averaging the outputs of all individual trees, which reduces overfitting and increases accuracy.

In this project, the Random Forest Regressor was implemented with 1000 decision trees (n_estimators=1000) and a maximum depth

of 12 (max_depth=12). The pipeline included data transformations and feature scaling before training the model.

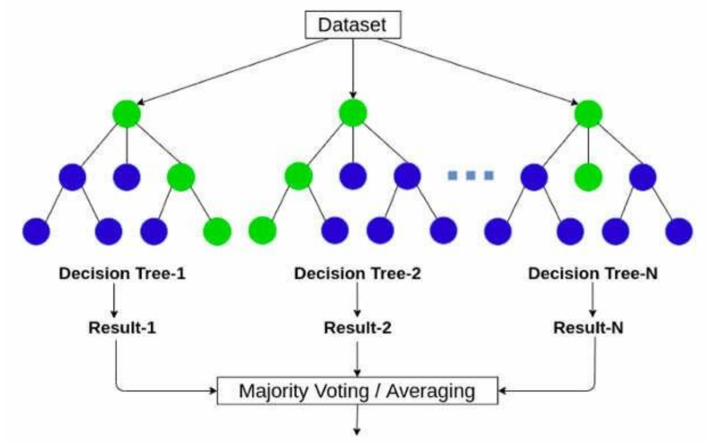


Figure 6:Ensemble Learning with Decision Trees

Strengths

- Handles both linear and non-linear relationships effectively.
- Can determine the importance of features, helping to identify which inputs (e.g., current runs, runs in the last 5 overs) influence predictions the most.
- Less likely to overfit compared to individual decision trees when properly tuned.

Drawbacks

- Computationally expensive, especially when using a large number of trees or deep trees.

- May require significant parameter tuning (e.g., number of trees, tree depth) for optimal performance.

3.4.3 Artificial Neural Networks (ANN)

Artificial Neural Networks (ANN) is a machine learning model inspired by the human brain. It consists of layers of interconnected units that can train to make predictions. They are designed to recognize patterns and relationships in data which makes them especially useful for regression and complex pattern recognitions. So these nodes have three layers the input layer, hidden layer, and output layer. A layer can have only a dozen units or millions of units as this depends on how the complex neural networks will be required to learn the hidden patterns in the dataset. And the neurons that are in ANN call artificial neurons which is a concept from biological neurons that comes from animal brains so they share a lot of similarities in the structure. Applications that use Aritifical Neural Network are Social Media platforms, Marketing and sales tools, Healthcare tools and personal assistant tools.

- **Input Layer**: This input layer receives the features that describe the current state of the match. In this scenario those are current runs, overs completed, wickets that have been lost, and runs in the last 5 overs (which helps to understand the team scoring pace).
- **Hidden Layer**: This is where this ANN model learn complex patterns from the input data. We gave neurons as 100 therefore they are responsible from learning the relationship between input features (runs, wickets, overs completed) and capturing non-linear patterns. This Hidden layer performs mathematical transformations.
- **Output Layer**: Then this Final output layer predicts the final score of the inning based on the features processed by the previous layers. This output layer takes data that is learned from hidden layer and transforms it into a specific output. For this scenario that outputs the score of the inning.

Advantages

- Can model highly complex, non-linear relationships between input features and the target variable.
- Flexible and capable of learning from large datasets with diverse features.

Limitations

- Computationally expensive and requires significant resources to train.
- Highly sensitive to hyperparameter tuning, such as the number of neurons, learning rate, and regularization terms.
- Requires a larger amount of data to perform well compared to simpler models like Linear Regression.

3.4.4 Gradient Boosting Regressor

In this project, the Gradient Boosting Regressor was implemented with 1000 estimators (n_estimators=1000) and a learning rate of 0.1 (learning_rate=0.1). The model learned iteratively, focusing on improving predictions by addressing errors from earlier iterations.

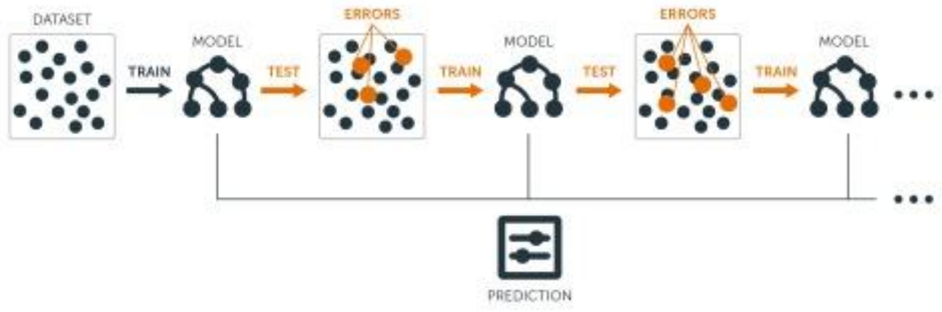


Figure 7:Boosting Workflow in Machine Learning

XGBRegressor

XGBRegressor is an advanced implementation of Gradient Boosting designed to enhance both efficiency and predictive performance. Built on the principles of boosting weak learners iteratively, it improves on traditional Gradient Boosting by incorporating several key optimizations:

- **Regularization Techniques**
  XGBRegressor introduces L1 (Lasso) and L2 (Ridge) regularization to reduce the risk of overfitting, which is common in boosting algorithms. These techniques penalize large coefficients, encouraging the model to focus on generalizable patterns rather than noise in the data.
- **Parallel Processing**
  Unlike traditional Gradient Boosting, XGBRegressor supports parallel processing during training. This significantly reduces computation time by allowing multiple trees to be built simultaneously, making it highly suitable for large datasets.
- **Tree Pruning**
  The algorithm uses a unique tree pruning technique known as "max depth" or "maximum number of splits" to control tree growth. Instead of expanding trees greedily, it uses a heuristic called the Minimum Loss Reduction (gamma parameter) to determine whether further splits improve the model's performance.
- **Custom Objective Functions**
  XGBRegressor allows the use of custom loss functions, providing flexibility to optimize for domain-specific metrics. For this project, a standard regression objective (Mean Squared Error) was used, but the algorithm can also accommodate other objectives.
- **Handling Missing Data**
  XGBoost natively handles missing data by learning default directions in the trees for missing values. This is particularly useful for datasets where imputing missing values could introduce bias or reduce accuracy.

- **Weighted Quantile Sketch**
  The algorithm uses a weighted quantile sketch method for better handling of weighted datasets, ensuring more accurate split calculations when working with uneven distributions of data.
- **Scalability**
  XGBRegressor is highly scalable, capable of handling datasets with millions of rows and hundreds of features efficiently. It achieves this through memory-efficient data storage and computation.
- **Cross-Validation Integration**
  XGBoost integrates cross-validation into its training process with the cv() function. This makes it easier to test different hyperparameter combinations and select the best-performing configuration.
- **Hyperparameter Tuning**
  For this project, XGBRegressor was optimized using advanced hyperparameter tuning techniques:

  - **Learning Rate (eta)**: Controlled the step size of updates, balancing the trade-off between speed and accuracy of learning.

  - **Number of Estimators (n_estimators)**: Determined the number of boosting rounds to iteratively improve predictions.

  - **Maximum Depth (max_depth)**: Limited tree growth to prevent overfitting while capturing meaningful data interactions.

  - **Subsample Ratio (subsample)**: Reduced the proportion of data samples used for each tree, adding randomness and preventing overfitting.

- **GPU Support**
  For datasets with high-dimensional features or requiring rapid predictions, XGBRegressor offers GPU acceleration, further enhancing its speed.

In this project, XGBRegressor outperformed other models due to its ability to efficiently handle complex, nonlinear relationships in the data while maintaining high speed and scalability. By fine-tuning its hyperparameters, the model achieved remarkable accuracy and robustness, making it the most suitable choice for predicting first-innings scores in T20 cricket matches.

For this project we used these values:

- n_estimators=1000
- learning_rate=0.1
- max_depth=14
- random_state=1

Comparison Between Gradient Boosting and XGBooster Regression

Gradient boosting regression is easy to implement and interpret but it's not good for larger datasets. But XGBRegression is Optimized for larger datasets with good speed and performance especially for larger datasets.

Strengths

- Highly accurate for larger and more complex datasets
- Can handle both linear and non-linear relationships effectively.
- Tunable for many datasets and problems.

Drawbacks

- Takes a lot of computational power, especially for larger datasets.
- It requires very careful tuning to prevent overfitting.

4. Implementation

The implementation phase focused on using Python and its machine learning libraries to build, train, and evaluate models for predicting first-innings scores in T20 cricket matches. This section outlines the tools and libraries used, the training process, testing and validation, and key code snippets.

4.1 Tools and Libraries

To implement the project, several Python libraries and tools were utilized, each serving a specific purpose:

- *pandas:* For data cleaning, preprocessing, and manipulation. It was essential for handling the cricket dataset efficiently.
- *NumPy:* Used for mathematical operations and handling numerical arrays.
- *matplotlib and seaborn:* For data visualization. These libraries helped create plots like histograms, scattered plots, and heatmaps to understand data trends.
- *Scikit-learn:* Provided a wide range of tools for preprocessing, model building, and evaluation, such as Linear Regression, Random Forest, and Gradient Boosting.
- *XGBoost:* Specifically used for implementing the XGBRegressor model, known for its performance and efficiency.
- *TensorFlow/Keras:* For building and training Artificial Neural Networks (ANN) to capture complex data relationships.
- *Joblib:* For saving trained models for future use.
- *Jupyter Notebook:* Used as the development environment for testing and running the models.

4.2 Model Training

The model training process involved selecting features, splitting the dataset, and optimizing model parameters to achieve accurate predictions.

Dataset Splitting
The dataset was divided into training (80%) and testing (20%) sets using scikit-learn's `train_test_split` function. This ensured the models were evaluated on unseen data to avoid overfitting.

Training the Models
Each machine learning model was trained using the training dataset. The models included Linear Regression, Random Forest, Gradient Boosting, XGBRegressor, and ANN. Default parameters were used initially, followed by hyperparameter tuning.

Hyperparameter Tuning
To optimize the models, hyperparameter tuning was performed using Grid Search or Random Search. For example:

- **Random Forest**: The number of trees (`n_estimators`) and the maximum depth of each tree (`max_depth`) were tuned.
- **XGBRegressor**: Parameters like learning rate (`eta`), maximum depth (`max_depth`), and the number of boosting rounds (`n_estimators`) were optimized.
- **ANN**: The number of hidden layers, neurons, and learning rate were adjusted using trial and error.

Example of Grid Search for Random Forest:

The best parameters were used to retrain the models for improved accuracy.

4.3 Testing and Validation

To evaluate the models, multiple metrics were used to measure their performance on the test data. These metrics included:

- **Mean Absolute Error (MAE)**: Measures the average magnitude of errors in the predictions.
- **Root Mean Squared Error (RMSE)**: Provides insight into the standard deviation of the prediction errors.
- **R^2 Score**: Indicates how well the model explains the variance in the target variable.

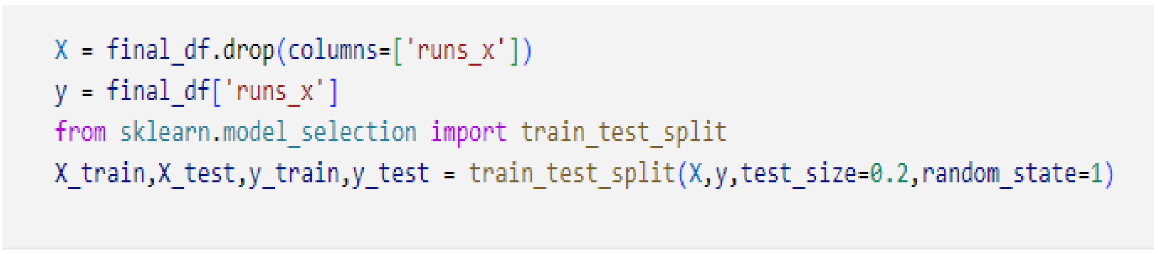
Example of Evaluation Metrics:

Each model's performance was compared based on these metrics, and the most suitable models were selected for deployment.

4.4 Code Snippets

Key code snippets highlight the implementation process:

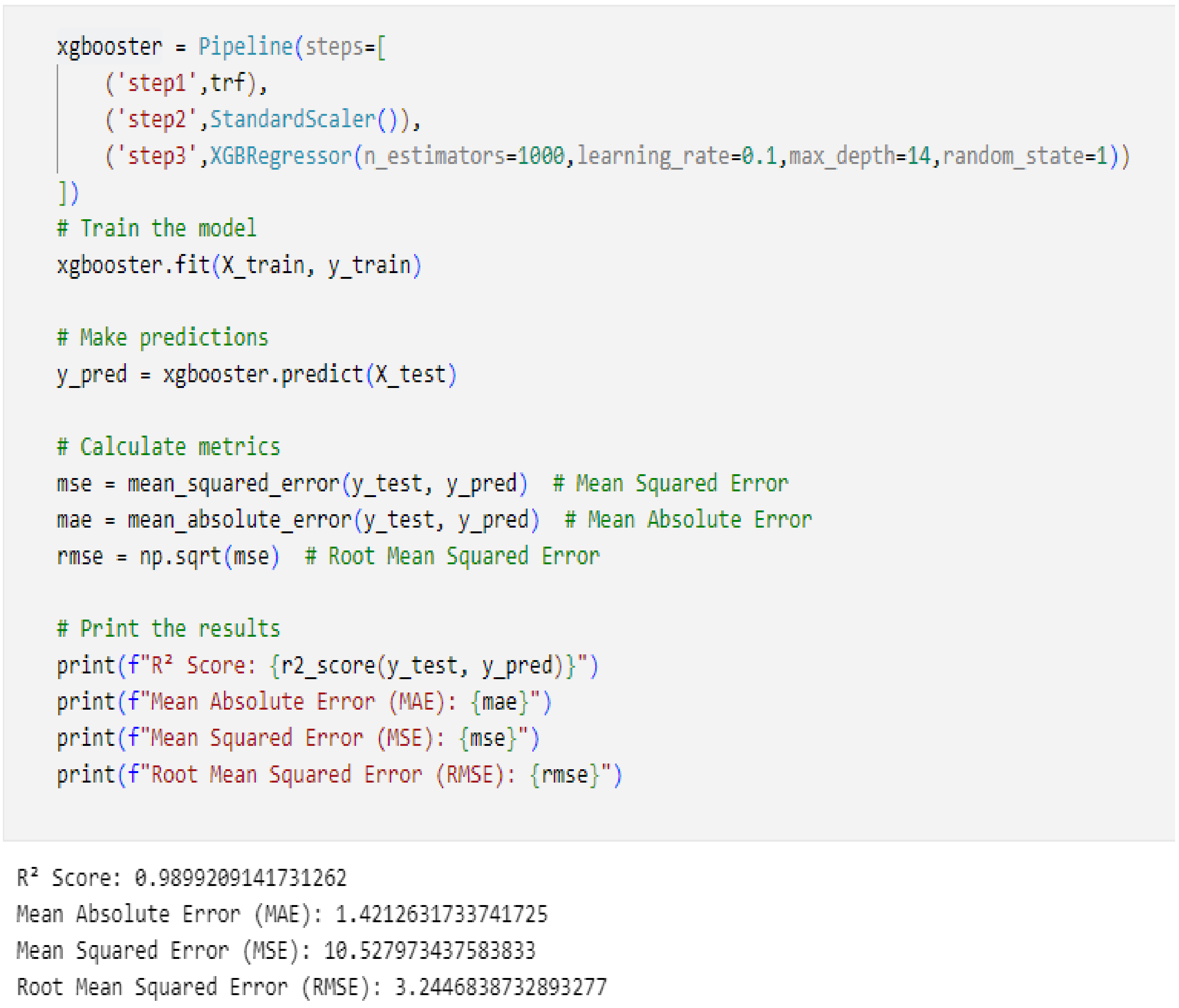
1. Train Test Split the Dataset



```
X = final_df.drop(columns=['runs_x'])
y = final_df['runs_x']
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=1)
```

Figure 8: Train test split the data code

2. Training an XGBRegressor Model

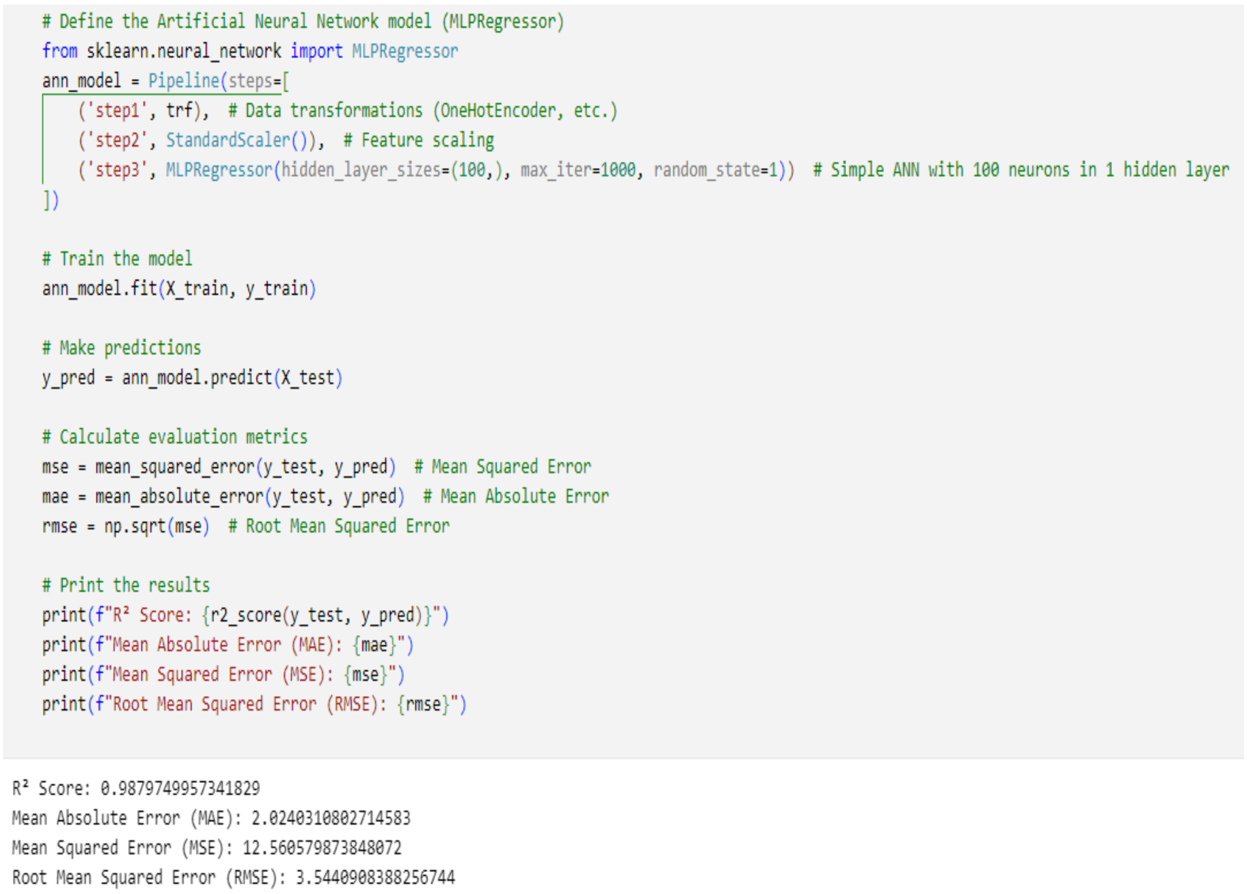


```
xgbooster = Pipeline(steps=[
    ('step1',trf),
    ('step2',StandardScaler()),
    ('step3',XGBRegressor(n_estimators=1000,learning_rate=0.1,max_depth=14,random_state=1))
])
# Train the model
xgbooster.fit(X_train, y_train)

# Make predictions
y_pred = xgbooster.predict(X_test)

# Calculate metrics
mse = mean_squared_error(y_test, y_pred)  # Mean Squared Error
mae = mean_absolute_error(y_test, y_pred)  # Mean Absolute Error
rmse = np.sqrt(mse)  # Root Mean Squared Error

# Print the results
print(f"R² Score: {r2_score(y_test, y_pred)}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
```

```
R² Score: 0.9899209141731262
Mean Absolute Error (MAE): 1.4212631733741725
Mean Squared Error (MSE): 10.527973437583833
Root Mean Squared Error (RMSE): 3.2446838732893277
```

Figure 9: Training an XGBRegressor Model

3. Training an Artificial Neural Network (ANN)

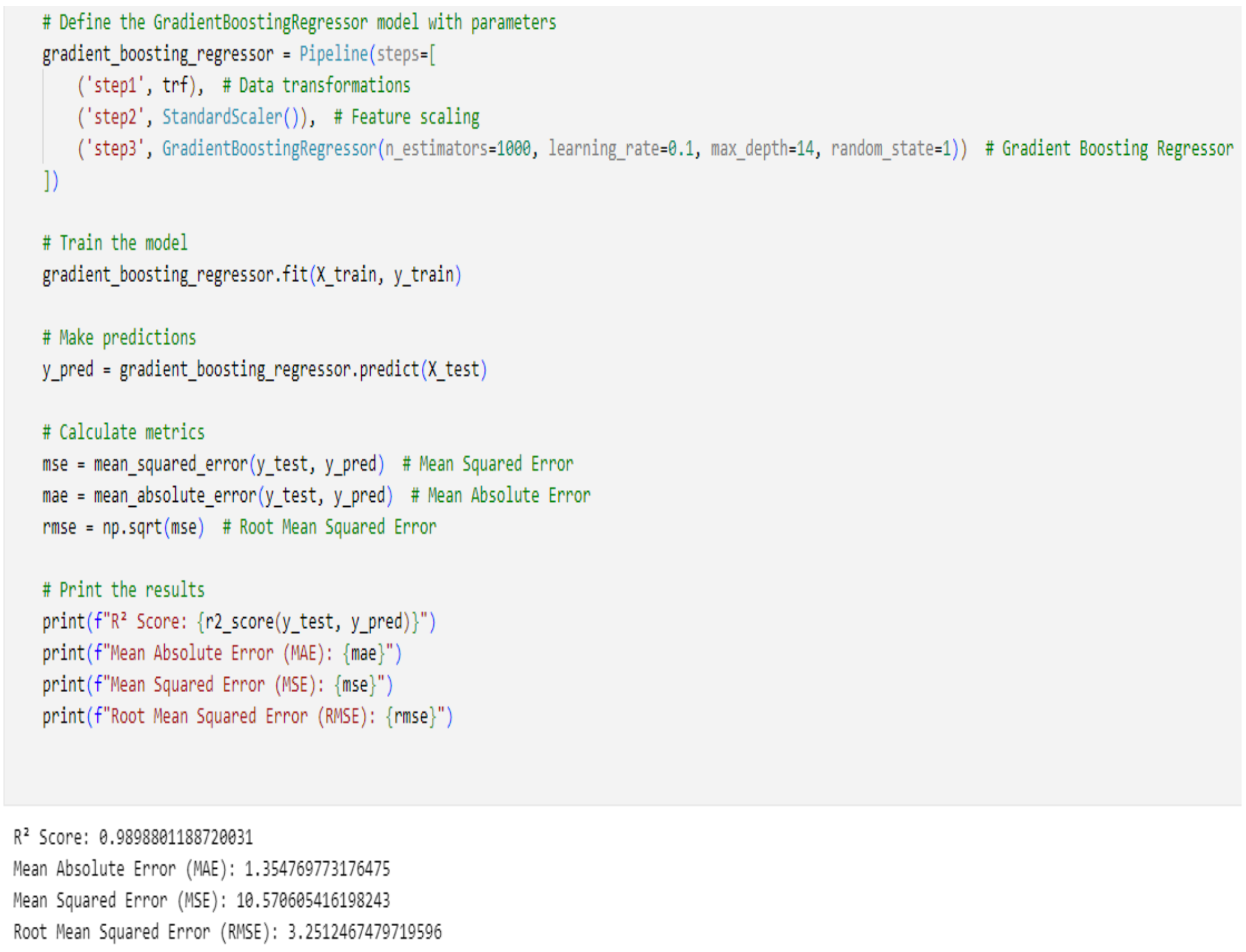


```python
# Define the Artificial Neural Network model (MLPRegressor)
from sklearn.neural_network import MLPRegressor
ann_model = Pipeline(steps=[
    ('step1', trf),  # Data transformations (OneHotEncoder, etc.)
    ('step2', StandardScaler()),  # Feature scaling
    ('step3', MLPRegressor(hidden_layer_sizes=(100,), max_iter=1000, random_state=1))  # Simple ANN with 100 neurons in 1 hidden layer
])

# Train the model
ann_model.fit(X_train, y_train)

# Make predictions
y_pred = ann_model.predict(X_test)

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)  # Mean Squared Error
mae = mean_absolute_error(y_test, y_pred)  # Mean Absolute Error
rmse = np.sqrt(mse)  # Root Mean Squared Error

# Print the results
print(f"R² Score: {r2_score(y_test, y_pred)}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
```

```
R² Score: 0.9879749957341829
Mean Absolute Error (MAE): 2.0240310802714583
Mean Squared Error (MSE): 12.560579873848072
Root Mean Squared Error (RMSE): 3.5440908388256744
```

Figure 10:Training an Artificial Neural Network (ANN) Model

4. Training the Gradient Booster regression Model

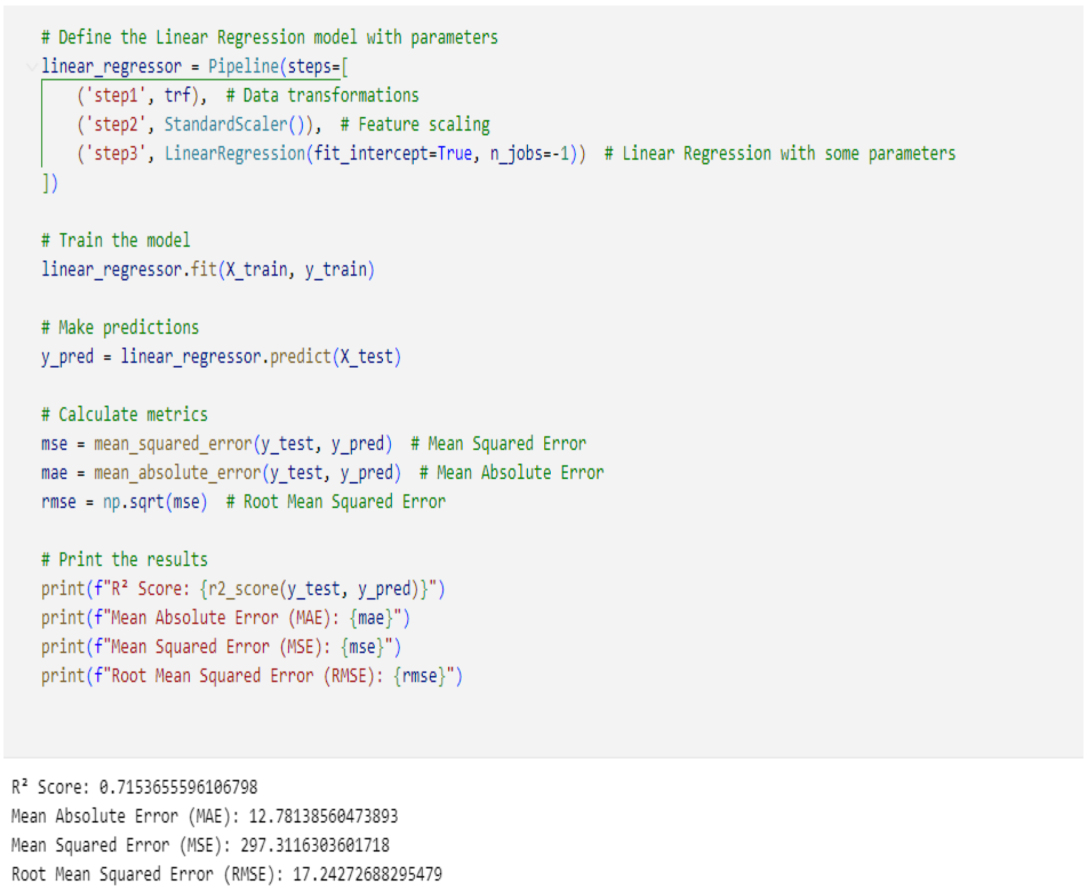


```
# Define the GradientBoostingRegressor model with parameters
gradient_boosting_regressor = Pipeline(steps=[
    ('step1', trf),  # Data transformations
    ('step2', StandardScaler()),  # Feature scaling
    ('step3', GradientBoostingRegressor(n_estimators=1000, learning_rate=0.1, max_depth=14, random_state=1))  # Gradient Boosting Regressor
])

# Train the model
gradient_boosting_regressor.fit(X_train, y_train)

# Make predictions
y_pred = gradient_boosting_regressor.predict(X_test)

# Calculate metrics
mse = mean_squared_error(y_test, y_pred)  # Mean Squared Error
mae = mean_absolute_error(y_test, y_pred)  # Mean Absolute Error
rmse = np.sqrt(mse)  # Root Mean Squared Error

# Print the results
print(f"R² Score: {r2_score(y_test, y_pred)}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
```

```
R² Score: 0.9898801188720031
Mean Absolute Error (MAE): 1.354769773176475
Mean Squared Error (MSE): 10.570605416198243
Root Mean Squared Error (RMSE): 3.2512467479719596
```

Figure 11:Training the Gradient Booster regression Model

5. Training the LinearRegression Model

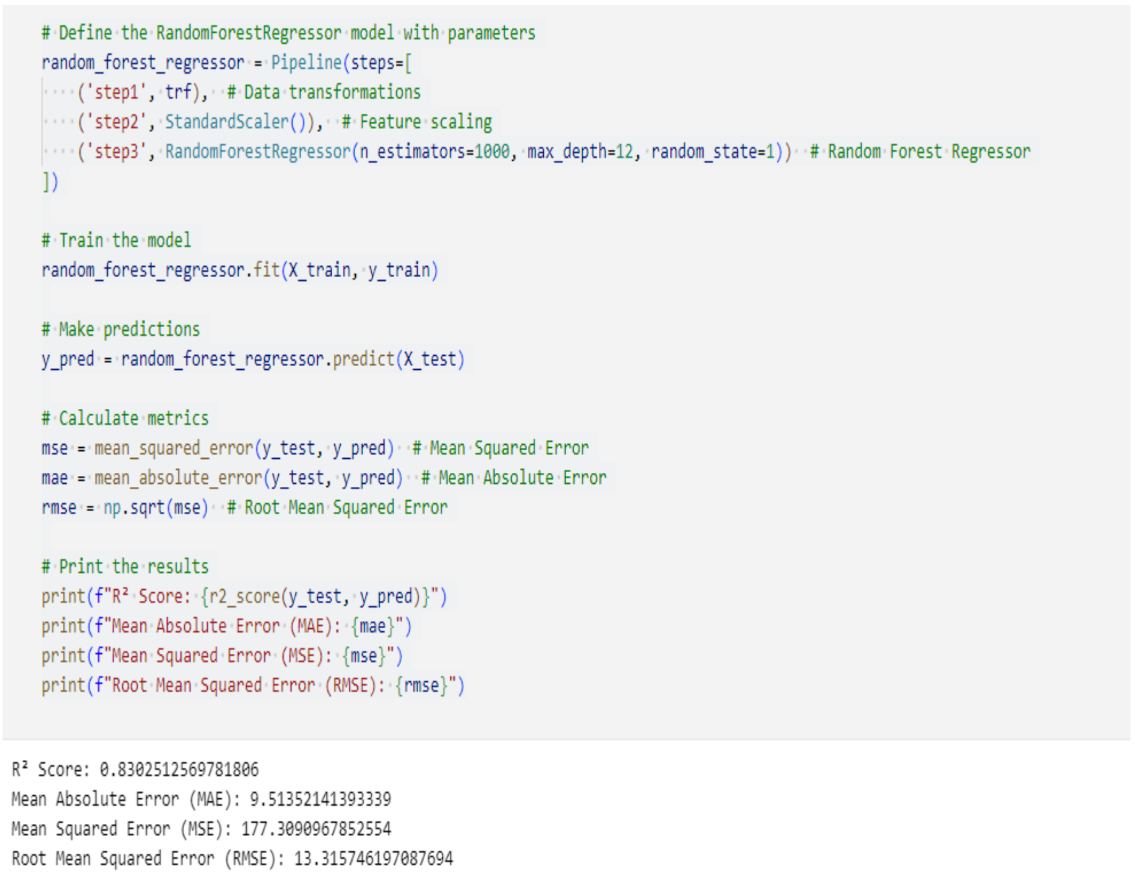


```python
# Define the Linear Regression model with parameters
linear_regressor = Pipeline(steps=[
    ('step1', trf),  # Data transformations
    ('step2', StandardScaler()),  # Feature scaling
    ('step3', LinearRegression(fit_intercept=True, n_jobs=-1))  # Linear Regression with some parameters
])

# Train the model
linear_regressor.fit(X_train, y_train)

# Make predictions
y_pred = linear_regressor.predict(X_test)

# Calculate metrics
mse = mean_squared_error(y_test, y_pred)  # Mean Squared Error
mae = mean_absolute_error(y_test, y_pred)  # Mean Absolute Error
rmse = np.sqrt(mse)  # Root Mean Squared Error

# Print the results
print(f"R² Score: {r2_score(y_test, y_pred)}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
```

```
R² Score: 0.7153655596106798
Mean Absolute Error (MAE): 12.78138560473893
Mean Squared Error (MSE): 297.3116303601718
Root Mean Squared Error (RMSE): 17.24272688295479
```

Figure 12:Training the LinearRegression Model code

6. Training the Random Forest Regression Model

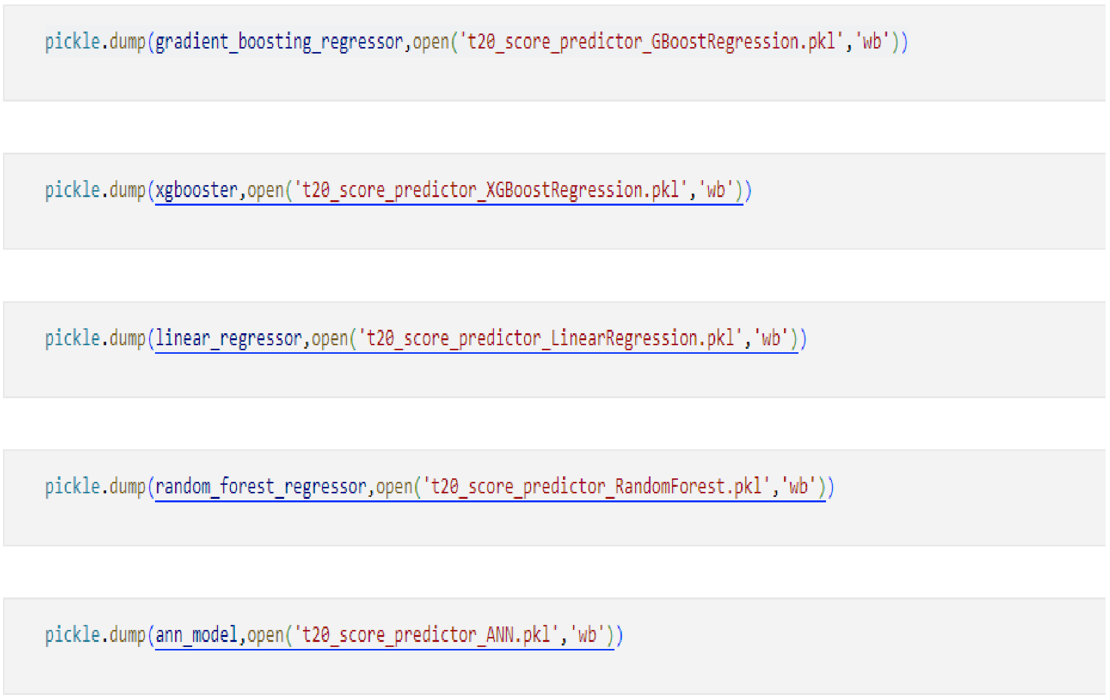


```python
# Define the RandomForestRegressor model with parameters
random_forest_regressor = Pipeline(steps=[
    ('step1', trf),  # Data transformations
    ('step2', StandardScaler()),  # Feature scaling
    ('step3', RandomForestRegressor(n_estimators=1000, max_depth=12, random_state=1))  # Random Forest Regressor
])

# Train the model
random_forest_regressor.fit(X_train, y_train)

# Make predictions
y_pred = random_forest_regressor.predict(X_test)

# Calculate metrics
mse = mean_squared_error(y_test, y_pred)  # Mean Squared Error
mae = mean_absolute_error(y_test, y_pred)  # Mean Absolute Error
rmse = np.sqrt(mse)  # Root Mean Squared Error

# Print the results
print(f"R² Score: {r2_score(y_test, y_pred)}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
```

```
R² Score: 0.8302512569781806
Mean Absolute Error (MAE): 9.51352141393339
Mean Squared Error (MSE): 177.3090967852554
Root Mean Squared Error (RMSE): 13.315746197087694
```

Figure 13:Training the Random Forest Regression Model Code

7. Saving the Model
Trained models were saved using joblib for later use:



```python
pickle.dump(gradient_boosting_regressor,open('t20_score_predictor_GBoostRegression.pkl','wb'))
```

```python
pickle.dump(xgbooster,open('t20_score_predictor_XGBoostRegression.pkl','wb'))
```

```python
pickle.dump(linear_regressor,open('t20_score_predictor_LinearRegression.pkl','wb'))
```

```python
pickle.dump(random_forest_regressor,open('t20_score_predictor_RandomForest.pkl','wb'))
```

```python
pickle.dump(ann_model,open('t20_score_predictor_ANN.pkl','wb'))
```

Figure 14:Saving the Model code

5. Result and Discussion

The performance of the models used to forecast cricket scores is covered in this part, along with the conclusions and observations drawn from the model evaluation. Using performance criteria such as Mean Squared Error (MSE), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE), the models under consideration are linear regression, random forest regression, gradient boosting regression, and XGBoost regression, and Artificial Neural Network. We also explore the significance of particular traits and their impact on the forecasts.

5.1Model Performance

5.1.1 Linear Regression

- MSE: 297.31
- MAE: 12.78
- RMSE: 17.24

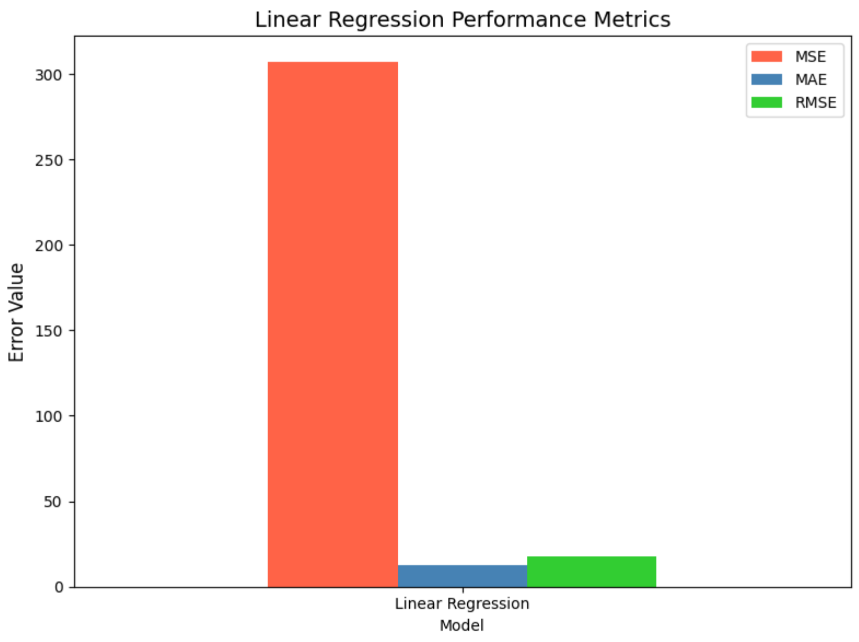


Figure 15:Linear Regression Chart

5.1.2 Random Forest Regression

- MSE: 177.30
- MAE: 9.51
- RMSE: 13.31

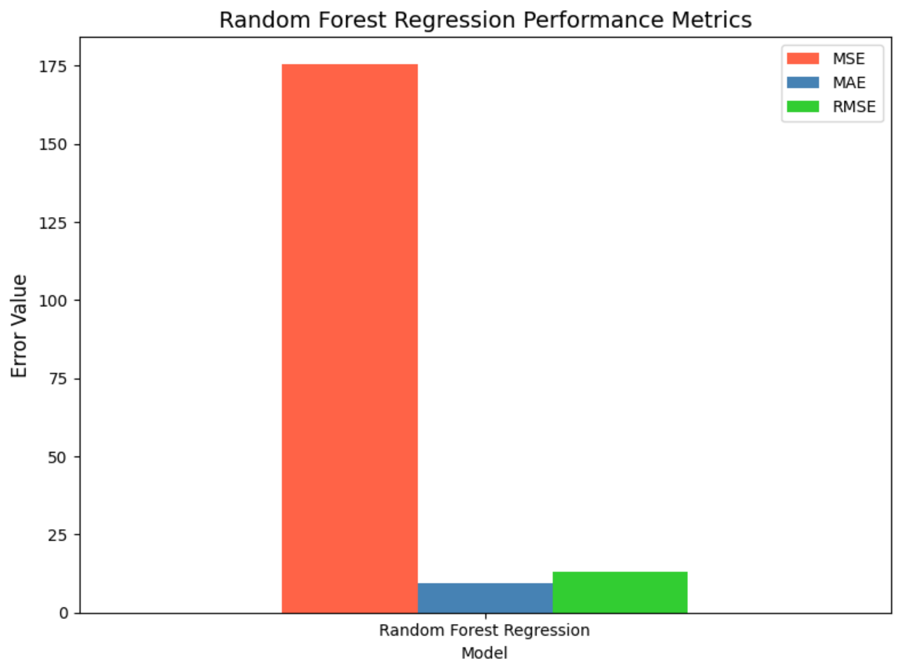


Figure 16: Random Forest Regression Performance Metrics Chart

5.1.3 XGB Regression

- MSE:10.52
- MAE: 1.42
- RMSE: 3.24

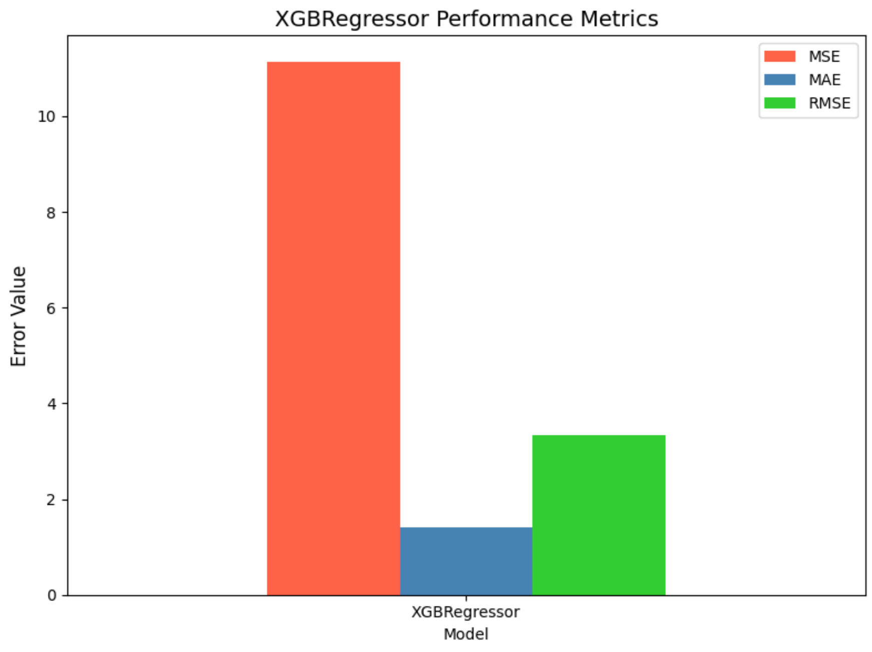


Figure 17: XGBRegressor Perfomance Metrics Chart

5.1.4 Gradient Booster Regression

- MSE:10.57
- MAE: 1.35
- RMSE: 3.25

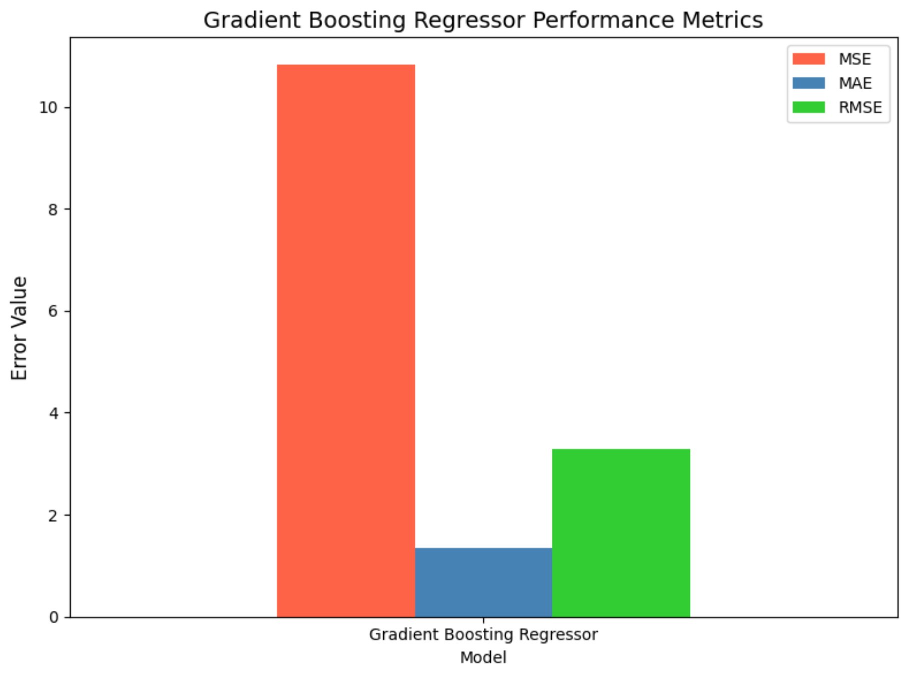


Figure 18: Gradient Boosting Regressor Performance Metrics Chart

5.1.5 Artificial Neural Network (ANN)

- MSE: 12.56
- MAE: 2.02
- RMSE: 3.54

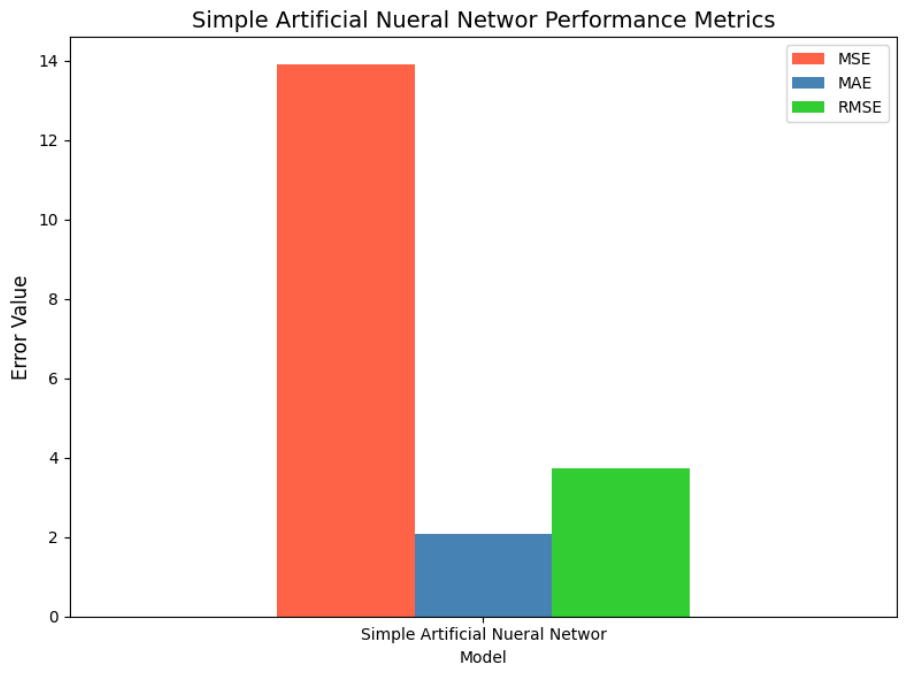


Figure 19: ANN Performance Metrics Chart

5.2Comparison between all the models.

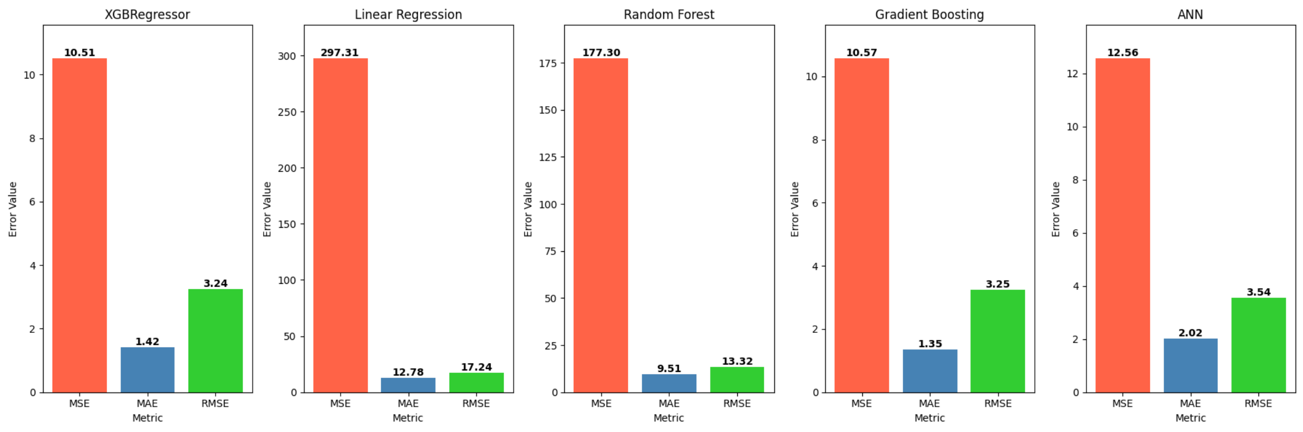


Figure 20: Comparison between all the models Chart

The above chart shows the MSE, MAE, and RMSE values of all 5models. Here is a side-by-side comparison.

- Linear Regression has the highest MSE and MAE values of **297.31** and **12.78**, indicating poor performance.
- Random Forest has an MSE value of **177.30** and an MAE value of **9.51**, performing better than Linear Regression but still unsuitable.
- Artificial Neural Network (ANN) has an MSE value of **12.56** and an MAE value of **2.02**, significantly better than the previous models but higher than XGB and Gradient Booster.
- XGB Regression has an MSE value of **10.51** and an MAE value of **1.42**, marginally outperforming Gradient Booster.
- Gradient Booster Regression has an MSE value of **10.57** and an MAE value of **1.35**, showing excellent performance but slightly lower than XGB Regression.

XGB Regression is the best model for this score prediction system due to its lower MSE and MAE values.

5.2.1 Observations

1. **Gradient Boosting** and **XGBRegressor** are the **top performers**, showing their suitability for this dataset.
2. Linear Regression struggles, likely due to the dataset’s nonlinear relationships.
3. Random Forest improves over Linear Regression but is less effective than boosting-based models.
4. ANN is competitive but requires more tuning to surpass Gradient Boosting and XGBRegressor.

5.3 Insights and Observations

The T20 score prediction system processes raw match data into a structured format suitable for machine learning models. Key features such as current score, balls left, wickets left, current run rate, and runs in the last 5 overs are derived to capture the dynamics of T20 cricket. The final dataset includes essential attributes like batting team, bowling team, and venue, with the final score as the target variable. The preprocessing pipeline incorporates one-hot encoding for categorical features and scaling for numerical ones, ensuring compatibility with regression models.

Multiple models were evaluated, including Linear Regression, Random Forest, Gradient Boosting, XGB Regressor, and an Artificial Neural Network (ANN). Linear Regression and Random Forest models performed poorly due to high error rates. Gradient Boosting and XGB Regressors achieved the best results, with XGB Regressor marginally outperforming the rest in terms of Mean Squared Error (MSE) and Mean Absolute Error (MAE). The ANN showed promise but did not surpass the ensemble models. XGB Regressor was selected as the optimal model for its accuracy and ability to handle complex data relationships.

The trained models were serialized using pickle for future use, and visualizations like scatter plots were used to analyze model performance. The system could be further enhanced by integrating external factors such as weather conditions or pitch characteristics and conducting hyperparameter tuning for neural networks. These improvements could refine predictions and expand the system's applicability for real-world scenarios.

Relevance of Particular Features
The following characteristics were especially important in determining an innings' ultimate score:

Current Runs: It goes without saying that a key factor in forecasting the ultimate score is the total amount of runs scored thus far. This aspect was emphasized as one of the most important by all models.

Wickets: The ultimate score is influenced by the number of wickets lost, which indicates how many overs a team is likely to bat. Because it aids in estimating the team's remaining potential, models that took this attribute into account fared better.

Overs Completed: Another crucial component was the quantity of overs that had already been finished. It is a crucial metric for determining how much time a team has remaining to score runs.

Runs in the Last Five Overs: This statistic aids in determining the team's momentum. Models that took this into account, particularly XGBoost and Random Forest, scored better. Teams that score rapidly in the last overs are likely to achieve a higher final score.

6. Conclusion and Future Work

6.1 Summary of Findings

The goal of this study was to use machine learning techniques to forecast first-inning scores in Twenty 20 cricket matches. Using historical match data, preprocessing it for insights, and testing different machine learning models to find the best predictors were the goals. Cleaning, feature engineering, and scaling were all part of the data preprocessing step, which got the data ready for analysis. Important variables that have a big impact on scores were identified via exploratory data analysis (EDA), including partnerships, strike rates, and the number of overs left.

Artificial Neural Networks (ANN), Linear Regression, Random Forest, Gradient Boosting, and XGBRegressor were among the machine learning models that were trained and assessed. Because of their capacity to manage intricate patterns and nonlinear interactions in the data, **Gradient Boosting** and **XGBRegressor** produced the **most accurate predictions** out of all of them. The study effectively illustrated how cutting-edge machine learning algorithms may be used to enhance real-time decision-making and offer actionable insights in sports analytics.

6.2 Future Enhancements

While the project achieved its objectives, there is room for improvement and expansion. Future work could include the following enhancements:

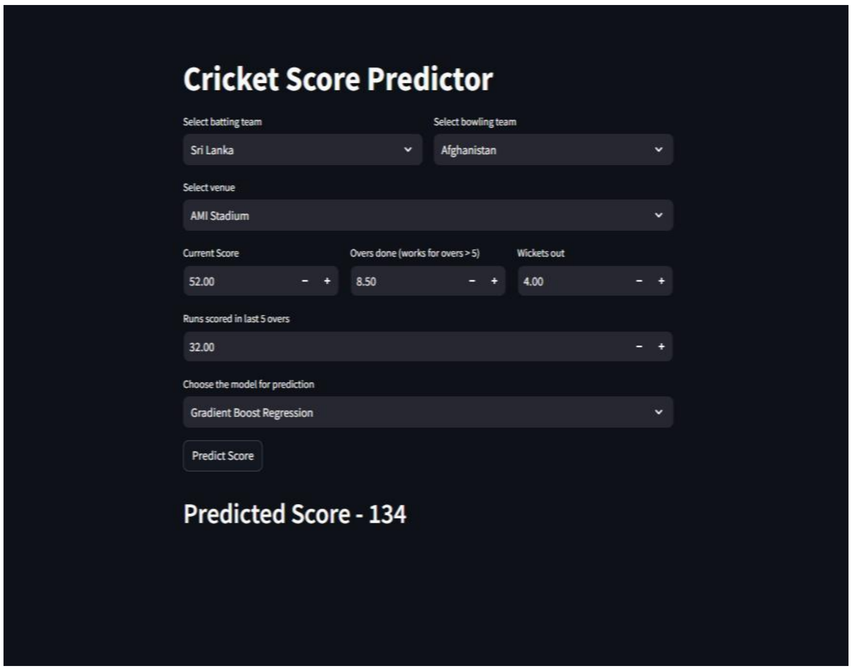
- **Incorporating Real-Time Data**: Prediction accuracy might be increased by including real-time match updates, such as player fitness, pitch reports, and weather.
- **Player-Specific Metrics**: More detailed information might be obtained by incorporating player-specific historical performance indicators, such as batting and bowling averages against certain opponents.
- **Advanced Algorithms**: Exploring more advanced models like Long Short-Term Memory (LSTM) networks or other time-series analysis methods could capture temporal dependencies in cricket matches.
- **Data Expansion**: Increasing the size and diversity of the dataset by including data from domestic leagues and international tournaments could make the models more robust.
- **Feature Enhancement**: Introducing additional features, such as team strategies, bowling speeds, and field placements, could capture more subtle influences on scoring.
- **Deployment**: Developing a real-time application or dashboard for coaches, analysts, and fans to use these predictions during live matches.

This project establishes the groundwork for machine learning in cricket analytics, creating opportunities for more study and real-world implementations. Future revisions can offer even more precise and insightful information by expanding on the current work, improving the game's comprehension and enjoyment.

7. References

- GitHub
- Lecture Materials

8. Appendices



Cricket Score Predictor

Select batting team: Sri Lanka

Select bowling team: Afghanistan

Select venue: AMI Stadium

Current Score: 52.00

Overs done (works for overs > 5): 8.50

Wickets out: 4.00

Runs scored in last 5 overs: 32.00

Choose the model for prediction: Gradient Boost Regression

Predicted Score - 134

Figure 21: Cricket Score Predictor UI