**CS2806**

**Machine Learning Techniques Lab**

**IPL\_Score\_Predictor PREDI-KET: employing various regressing models to predict score of an IPL match**

***S.Arjun(22011101093), S.Vigneshram(22011101097)***

**Abstract:** 1.1 Cricket is a very familiar and exciting sport that people of all age groups are insane to see and play. For many it's a billion-dollar market as they speculate financially, hoping to be able to earn profit in the form of gambling and various other ways. In this project, a model using machine learning algorithms is proposed to predict the score of each match based on past datasets available from2008 to 2020 IPL matches in Kaggle. This proposed methodology includes the following steps like Pre-processing of collected datasets, Feature selection from raw data, Conversion of categorical data into numerical data, Partitioning of samples into training and test samples, Training, and classification. Few machine learning algorithms like Support Vector Machine, Random Forest, Naive Bayes were already used in previous papers. In this project, algorithms like Linear Regression, K-Nearest Neighbor Regressor, XGBoost Regressor, Random Forest Regressor, SVR, Decision Tree Regressor models are proposed for a score prediction. The accuracy of the above machine learning algorithms is used to predict the score of an IPL match along with its Precision.

**Introduction**: 2.1 Cricket is the second-most popular sport in the world after football, but it is cricket that is most adored in India. Numerous research papers have been published in recent years, and a great deal of work has been done to predict the outcome of a cricket match using the variables that affect the outcome of the match. They use supervised machine learning algorithms to predict the outcome of the match, such as Linear regression, Support vector machines, Logistic regression, Decision Tree, Bayes Network, and Random Forest. Cricket is a fast-paced version of the game that draws spectators to the stadium and watchers at home, the Twenty20 format in particular is immensely popular. The Board of Control for Cricket in India oversees the professional Twenty20 cricket league known as the Indian Premier League (IPL) (BCCI). Each year, the Indian Premier League is held, with each team representing an Indian city. A massive industry like fantasy 11 and betting websites has given the model a lot of importance in addition to the excitement created by the media and several natural variables that have an impact on the game. The ability to accurately anticipate the outcome of a cricket match is greatly influenced by the game's regulations, the players' ability, their physical condition, and numerous other natural elements. People will use the predictions made by the machine learning algorithm as technology advances and apps like Fantasy 11 and betting sites become more popular. In many ways, using machine learning makes life easier. We won't rely on just one machine learning algorithm to forecast the result of a cricket match; rather, we'll use them all. Unsupervised learning and supervised learning are the two types of learning used in machine learning. When using unsupervised learning, the computer must sort the data based on patterns and combinations without any prior training. This is because the data is not properly labelled. In contrast, the data in supervised learning is labelled with the appropriate categorization so that the computer can quickly analyze it and come up with the desired outcome. Due to the well labelled cricket match data, unsupervised learning models are useless for our purpose. So, we'll employ supervised learning models. Regression and classification are the two categories used in supervised learning. Regression is used when the result is a real quantity, such as rupees or height, and classification is used to classify between categories, such as red or blue. It is of the number type. Regression will be used in our model because the winning percentage will be the outcome and numbers represent the outcome's type. Finding the critical elements that influence match outcome is our main goal, and we'll do this by choosing the machine learning model that best matches the data and produces the greatest outcomes. In the field of forecasting the outcome of a cricket match, certain works have already been published. In several papers, the accuracy suffers because only a few significant criteria are used to make predictions. The machine learning model is inappropriate in some papers, though. In order to select the appropriate model for training and testing the data, it is crucial to consider all the important variables that may have a bearing on the match's outcome. The prediction accuracy will dramatically increase as a result.

**Model Architecture:**

**2.1 Obtain the dataset:**

.The dataset for analysis and prediction was obtained from www.kaggle.com, which included data from past IPL editions from 2008 to 2020.

The dataset comprises of over by over details of matches and runs from 2008 to 2020. Dataset Used: ipl\_data.csv

Attributes include:

mid - match id

date - when matches are played

venue - place where matches are played

bat\_team - batting team

bowl\_team - bowling team

batsman - batsman

bowler - bowler

runs - runs scored

wickets - wickets

overs - overs - next 3 are based on this

run\_last\_5 - runs scored in last 5 overs

wicket\_last\_5 - wickets in last 5 overs

stricker - batsman playing as main 1

non-striker - batsman playing as runner up - not main 0

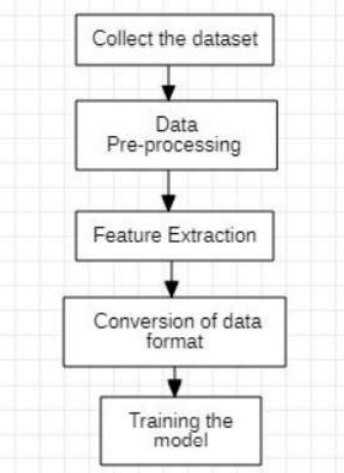
total - total score (target variable)

**2.2 Information Feature extraction and pre-processing:**

The pre-processing stage cleans the dataset by deleting data that isn't necessary for obtaining results. During the pre-processing stage, data that has not been declared or tagged is eliminated. To extract the essential analysis, as well as for the prediction module, the data must be pre-processed and cleaned. The dataset was created using records from the last 12 years, or from season 2008 to 2020. Methods such as eliminating outliers, normalizing, and standardization are used to pre-process the data.

**2.3 Data format conversion:**

Because a few of the dataset's attributes are categorical, classification is rather difficult. It may potentially have an impact on the model, resulting in incorrect predictions. Except for the target attribute (Winner), all categorical data in the dataset has been transformed to numeric format and standardized on a scale basis in this step. Label encoding and one-hot encoding are the two most used approaches.



**Fig1: basic architecture**

A diagram of a data system

Description automatically generated

**Fig2: Architecture of the model**

**2.4 Model Training:**

The datasets were divided into two sections for training and testing before the model was trained. On the datasets, six regression models are used to predict the score: Linear Regression, K-Nearest Neighbor Regressor, XGBoost Regressor, Random Forest Regressor, SVR, Decision Tree Regressor models

**2.4.1 Linear Regression:**

Linear regression is a simple and widely-used statistical technique for modeling the relationship between a dependent variable (target) and one or more independent variables (features). It assumes that the relationship between the variables can be represented by a linear equation. The model aims to find the best-fitting straight line through the data points, minimizing the sum of the squared differences between the observed and predicted values. Linear regression is sensitive to outliers and assumes a linear relationship between variables.

**2.4.2 K-Nearest Neighbors (KNN) Regressor:**

KNN is a non-parametric and instance-based learning algorithm used for both classification and regression tasks. In KNN regression, the predicted value for a data point is the average of the target values of its k nearest neighbors. The value of k, the number of neighbors, is a hyperparameter that needs to be specified. KNN is simple to implement and can capture complex patterns in the data. However, it can be computationally expensive, especially with large datasets.

**2.4.3 XGBoost Regressor:**

XGBoost (Extreme Gradient Boosting) is an implementation of gradient boosting decision trees designed for speed and performance. It builds an ensemble of weak learners (decision trees) sequentially, where each tree corrects the errors made by the previous ones. XGBoost uses a gradient descent algorithm to minimize a loss function, typically the mean squared error (MSE), and adds regularization terms to prevent overfitting. XGBoost is known for its high performance, scalability, and ability to handle large datasets with high-dimensional features.

**2.4.4 Random Forest:**

Random forest is an ensemble-based supervised learning methodology. Ensemble learning is a type of learning in which many decision trees are constructed and then combined to produce more accurate prediction models. The resulting of trees termed "Random Forest" is a mix of multiple decision trees. The random forest algorithm is not biassed because it is based on a majority vote and delivers the final prediction based on that voting. Random Forest and Decision Tree both employ the identical Equation(1) and Equation(2) formulas.

The Random Forest method is based on the following principle:

Phase 1: In this step, random rows from the training data set will be selected by assigning an arbitrary value.

Phase 2: The decision tree is built based on the selection of random rows in this step. The output is then created from each decision tree.

Phase 3: Using the frequency method, voting will be done on the created output.

Phase 4: Based on the number of votes collected from the decision trees, the ultimate result is anticipated from step 3.

**2.4.5 Support Vector Regression (SVR):**

SVR is a variant of support vector machines (SVM) used for regression tasks. It finds the hyperplane that best fits the data, while also minimizing the margin violations (deviations of the actual target values from the predicted values). SVR uses a kernel function to map the input features into a higher-dimensional space, where the hyperplane is constructed. SVR is effective in capturing complex relationships in the data and is less sensitive to outliers compared to linear regression.

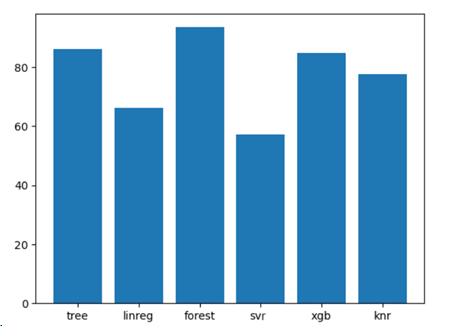
**2.4.6 Decision Tree Regressor:**

Decision trees are hierarchical tree structures used for both classification and regression tasks. In decision tree regression, the feature space is partitioned into regions, and for each region, the predicted value is the average (or another aggregation) of the target values of the training samples within that region. Decision trees recursively split the feature space based on the feature that provides the best split according to some criterion (e.g., information gain, Gini impurity). Decision trees are easy to interpret and visualize but are prone to overfitting, especially with deep trees and noisy data.

**PERFORMANCE ANALYSIS AND RESULT:**

3.1 Accuracy: The fraction of correct forecast in all predictions is known as accuracy. In this experiment, random forest classifier outruns all the algorithms by predicting the result with highest accuracy of

93.47%. The figure 3 shows the accuracies of the various algorithms implemented.



**Fig3:** Accuracy of various algorithms

Table 1 shows about the various algorithms and their accuracies obtained. It is clear from the table that the random forest classifier performed better than other algorithm

|  |  |
| --- | --- |
| **ALGORITHM** | **ACCURACY** |
| Decision tree regression | 86.10570822655737 |
| Linear regression | 66.06950355728935 |
| Random forest regression | 93.47093055358451 |
| Support vector regressor | 57.088777652694 |
| XGBoost regression | 84.77182323206365 |
| k-nearest neighbour regression | 77.73370051418556 |

*Table1:* Dataset Statistics for 3.1

**Code:**

# Importing Necessary Libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

#Importing dataset

ipl\_df = pd.read\_csv('/content/ipl\_data.csv')

print(f"Dataset successfully Imported of Shape : {ipl\_df.shape}")

**# DATA PRE-PROCESSING AND FEATURE EXTRACTION**

#removing irrelevant data columns

# Names of all columns

ipl\_df.columns

#Here, we can see that columns ['mid', 'date', 'venue', 'batsman', 'bowler', 'striker', 'non-striker'] won't provide any relevant information for our model to train

import pandas as pd

# Importing dataset with corrected file path

ipl\_df = pd.read\_csv(r"C:\Users\Arjun\Downloads\ipl\_data.csv")

irrelevant = ['mid', 'date', 'venue', 'batsman', 'bowler', 'striker', 'non-striker']

print(f'Before Removing Irrelevant Columns : {ipl\_df.shape}')

# Check if all irrelevant columns exist in DataFrame

existing\_columns = set(ipl\_df.columns)

missing\_columns = [col for col in irrelevant if col not in existing\_columns]

if missing\_columns:

print(f"The following columns are not present in the DataFrame: {missing\_columns}")

else:

# Drop Irrelevant Columns

ipl\_df = ipl\_df.drop(irrelevant, axis=1)

print(f'After Removing Irrelevant Columns : {ipl\_df.shape}')

print(ipl\_df.head())

#Keeping only Consistent Teams

const\_teams = ['Kolkata Knight Riders', 'Chennai Super Kings', 'Rajasthan Royals','Mumbai Indians', 'Kings XI Punjab', 'Royal Challengers Bangalore','Delhi Daredevils', 'Sunrisers Hyderabad']

print(f'Before Removing Inconsistent Teams : {ipl\_df.shape}')

ipl\_df = ipl\_df[(ipl\_df['bat\_team'].isin(const\_teams)) & (ipl\_df['bowl\_team'].isin(const\_teams))]

print(f'After Removing Irrelevant Columns : {ipl\_df.shape}')

print(f"Consistent Teams : \n{ipl\_df['bat\_team'].unique()}")

ipl\_df.head()

#Remove First 5 Overs of every match

print(f'Before Removing Overs : {ipl\_df.shape}')

ipl\_df = ipl\_df[ipl\_df['overs'] >= 5.0]

print(f'After Removing Overs : {ipl\_df.shape}')

ipl\_df.head()

#plotting correlation matrix of current data

from seaborn import heatmap

heatmap(data=ipl\_df.corr(), annot=True)

A screenshot of a graph

Description automatically generated

**#DATA FORMAT CONVERSION**

#performing label encoding

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

le = LabelEncoder()

for col in ['bat\_team', 'bowl\_team']:

  ipl\_df[col] = le.fit\_transform(ipl\_df[col])

ipl\_df.head()

#ONE HOT ENCODING  ANC COLUMN TRANSFORMATION

from sklearn.compose import ColumnTransformer

columnTransformer = ColumnTransformer([('encoder',

                                        OneHotEncoder(),

                                        [0, 1])],

                                      remainder='passthrough')

ipl\_df = np.array(columnTransformer.fit\_transform(ipl\_df))

# Save the Numpy Array in a new DataFrame with transformed columns

cols = ['batting\_team\_Chennai Super Kings', 'batting\_team\_Delhi Daredevils', 'batting\_team\_Kings XI Punjab',

'batting\_team\_Kolkata Knight Riders', 'batting\_team\_Mumbai Indians', 'batting\_team\_Rajasthan Royals',

'batting\_team\_Royal Challengers Bangalore', 'batting\_team\_Sunrisers Hyderabad',

'bowling\_team\_Chennai Super Kings', 'bowling\_team\_Delhi Daredevils', 'bowling\_team\_Kings XI Punjab',

'bowling\_team\_Kolkata Knight Riders', 'bowling\_team\_Mumbai Indians', 'bowling\_team\_Rajasthan Royals',

'bowling\_team\_Royal Challengers Bangalore', 'bowling\_team\_Sunrisers Hyderabad', 'runs', 'wickets', 'overs',

'runs\_last\_5', 'wickets\_last\_5', 'total']

df = pd.DataFrame(ipl\_df, columns=cols)

**Model Training:**

Preparing training and testing data

features = df.drop(['total'], axis=1)

labels = df['total']

from sklearn.model\_selection import train\_test\_split

train\_features, test\_features, train\_labels, test\_labels = train\_test\_split(features, labels, test\_size=0.20, shuffle=True)

print(f"Training Set : {train\_features.shape}\nTesting Set : {test\_features.shape}")

## ML Algorithms

**1. Decision Tree Regressor**

from sklearn.tree import DecisionTreeRegressor

tree = DecisionTreeRegressor()

# Train Model

tree.fit(train\_features, train\_labels)

# Evaluate Model

train\_score\_tree = str(tree.score(train\_features, train\_labels) \* 100)

test\_score\_tree = str(tree.score(test\_features, test\_labels) \* 100)

print(f'Train Score : {train\_score\_tree[:5]}%\nTest Score : {test\_score\_tree[:5]}%')

models["tree"] = test\_score\_tree

from sklearn.metrics import mean\_absolute\_error as mae, mean\_squared\_error as mse

print("---- Decision Tree Regressor - Model Evaluation ----")

print("Mean Absolute Error (MAE): {}".format(mae(test\_labels, tree.predict(test\_features))))

print("Mean Squared Error (MSE): {}".format(mse(test\_labels, tree.predict(test\_features))))

print("Root Mean Squared Error (RMSE): {}".format(np.sqrt(mse(test\_labels, tree.predict(test\_features)))))

**2.Linear Regression**

from sklearn.linear\_model import LinearRegression

linreg = LinearRegression()

# Train Model

linreg.fit(train\_features, train\_labels)

# Evaluate Model

train\_score\_linreg = str(linreg.score(train\_features, train\_labels) \* 100)

test\_score\_linreg = str(linreg.score(test\_features, test\_labels) \* 100)

print(f'Train Score : {train\_score\_linreg[:5]}%\nTest Score : {test\_score\_linreg[:5]}%')

models["linreg"] = test\_score\_linreg

print("---- Linear Regression - Model Evaluation ----")

print("Mean Absolute Error (MAE): {}".format(mae(test\_labels, linreg.predict(test\_features))))

print("Mean Squared Error (MSE): {}".format(mse(test\_labels, linreg.predict(test\_features))))

print("Root Mean Squared Error (RMSE): {}".format(np.sqrt(mse(test\_labels, linreg.predict(test\_features)))))

**3.Random Forest Regression**

from sklearn.ensemble import RandomForestRegressor

forest = RandomForestRegressor()

# Train Model

forest.fit(train\_features, train\_labels)

# Evaluate Model

train\_score\_forest = str(forest.score(train\_features, train\_labels)\*100)

test\_score\_forest = str(forest.score(test\_features, test\_labels)\*100)

print(f'Train Score : {train\_score\_forest[:5]}%\nTest Score : {test\_score\_forest[:5]}%')

models["forest"] = test\_score\_forest

print("---- Random Forest Regression - Model Evaluation ----")

print("Mean Absolute Error (MAE): {}".format(mae(test\_labels, forest.predict(test\_features))))

print("Mean Squared Error (MSE): {}".format(mse(test\_labels, forest.predict(test\_features))))

print("Root Mean Squared Error (RMSE): {}".format(np.sqrt(mse(test\_labels, forest.predict(test\_features)))))

**4.Support Vector Machine**

from sklearn.svm import SVR

svm = SVR()

# Train Model

svm.fit(train\_features, train\_labels)

#evaluate model

train\_score\_svm = str(svm.score(train\_features, train\_labels)\*100)

test\_score\_svm = str(svm.score(test\_features, test\_labels)\*100)

print(f'Train Score : {train\_score\_svm[:5]}%\nTest Score : {test\_score\_svm[:5]}%')

models["svm"] = test\_score\_svm

print("---- Support Vector Regression - Model Evaluation ----")

print("Mean Absolute Error (MAE): {}".format(mae(test\_labels, svm.predict(test\_features))))

print("Mean Squared Error (MSE): {}".format(mse(test\_labels, svm.predict(test\_features))))

print("Root Mean Squared Error (RMSE): {}".format(np.sqrt(mse(test\_labels, svm.predict(test\_features)))))

**5.XGBoost**

from xgboost import XGBRegressor

xgb = XGBRegressor()

# Train Model

xgb.fit(train\_features, train\_labels)

#evaluate model

train\_score\_xgb = str(xgb.score(train\_features, train\_labels)\*100)

test\_score\_xgb = str(xgb.score(test\_features, test\_labels)\*100)

print(f'Train Score : {train\_score\_xgb[:5]}%\nTest Score : {test\_score\_xgb[:5]}%')

models["xgb"] = test\_score\_xgb

print("---- XGB Regression - Model Evaluation ----")

print("Mean Absolute Error (MAE): {}".format(mae(test\_labels, xgb.predict(test\_features))))

print("Mean Squared Error (MSE): {}".format(mse(test\_labels, xgb.predict(test\_features))))

print("Root Mean Squared Error (RMSE): {}".format(np.sqrt(mse(test\_labels, xgb.predict(test\_features)))))

**6.K NEAREST NEIGHBOURS:**

from sklearn.neighbors import KNeighborsRegressor

knr = KNeighborsRegressor()

# Train Model

knr.fit(train\_features, train\_labels)

#evaluate model

train\_score\_knr = str(knr.score(train\_features, train\_labels)\*100)

test\_score\_knr = str(knr.score(test\_features, test\_labels)\*100)

print(f'Train Score : {train\_score\_knr[:5]}%\nTest Score : {test\_score\_knr[:5]}%')

models["knr"] = test\_score\_knr

print("---- KNR - Model Evaluation ----")

print("Mean Absolute Error (MAE): {}".format(mae(test\_labels, knr.predict(test\_features))))

print("Mean Squared Error (MSE): {}".format(mse(test\_labels, knr.predict(test\_features))))

print("Root Mean Squared Error (RMSE): {}".format(np.sqrt(mse(test\_labels, knr.predict(test\_features)))))

**Best Model**

%matplotlib inline

import matplotlib.pyplot as plt

model\_names = list(models.keys())

accuracy = list(map(float, models.values()))

# creating the bar plot

plt.bar(model\_names, accuracy)

A graph of blue bars

Description automatically generated with medium confidence

print(accuracy)

OUTPUT: [84.54550911987303, 66.06334375169997, 93.43831973912555, 57.2555162561484, 84.81572056878687, 77.07084803147988]

# Predictions

def score\_predict(batting\_team, bowling\_team, runs, wickets, overs, runs\_last\_5, wickets\_last\_5, model=forest):

  prediction\_array = []

  # Batting Team

  if batting\_team == 'Chennai Super Kings':

    prediction\_array = prediction\_array + [1,0,0,0,0,0,0,0]

  elif batting\_team == 'Delhi Daredevils':

    prediction\_array = prediction\_array + [0,1,0,0,0,0,0,0]

  elif batting\_team == 'Kings XI Punjab':

    prediction\_array = prediction\_array + [0,0,1,0,0,0,0,0]

  elif batting\_team == 'Kolkata Knight Riders':

    prediction\_array = prediction\_array + [0,0,0,1,0,0,0,0]

  elif batting\_team == 'Mumbai Indians':

    prediction\_array = prediction\_array + [0,0,0,0,1,0,0,0]

  elif batting\_team == 'Rajasthan Royals':

    prediction\_array = prediction\_array + [0,0,0,0,0,1,0,0]

  elif batting\_team == 'Royal Challengers Bangalore':

    prediction\_array = prediction\_array + [0,0,0,0,0,0,1,0]

  elif batting\_team == 'Sunrisers Hyderabad':

    prediction\_array = prediction\_array + [0,0,0,0,0,0,0,1]

  # Bowling Team

  if bowling\_team == 'Chennai Super Kings':

    prediction\_array = prediction\_array + [1,0,0,0,0,0,0,0]

  elif bowling\_team == 'Delhi Daredevils':

    prediction\_array = prediction\_array + [0,1,0,0,0,0,0,0]

  elif bowling\_team == 'Kings XI Punjab':

    prediction\_array = prediction\_array + [0,0,1,0,0,0,0,0]

  elif bowling\_team == 'Kolkata Knight Riders':

    prediction\_array = prediction\_array + [0,0,0,1,0,0,0,0]

  elif bowling\_team == 'Mumbai Indians':

    prediction\_array = prediction\_array + [0,0,0,0,1,0,0,0]

  elif bowling\_team == 'Rajasthan Royals':

    prediction\_array = prediction\_array + [0,0,0,0,0,1,0,0]

  elif bowling\_team == 'Royal Challengers Bangalore':

    prediction\_array = prediction\_array + [0,0,0,0,0,0,1,0]

  elif bowling\_team == 'Sunrisers Hyderabad':

    prediction\_array = prediction\_array + [0,0,0,0,0,0,0,1]

  prediction\_array = prediction\_array + [runs, wickets, overs, runs\_last\_5, wickets\_last\_5]

  prediction\_array = np.array([prediction\_array])

  pred = model.predict(prediction\_array)

  return int(round(pred[0]))

### **Test 1**

* Batting Team : **Delhi Daredevils**
* Bowling Team : **Chennai Super Kings**
* Final Score : **147/9**

batting\_team='Delhi Daredevils'

bowling\_team='Chennai Super Kings'

score = score\_predict(batting\_team, bowling\_team, overs=10.2, runs=68, wickets=3, runs\_last\_5=29, wickets\_last\_5=1)

print(f'Predicted Score : {score} || Actual Score : 147')

**OUTPUT:**

Predicted Score : 151 || Actual Score : 147

### **Test 2**

* Batting Team : **Mumbai Indians**
* Bowling Team : **Kings XI Punjab**
* Final Score : **176/7**

batting\_team='Mumbai Indians'

bowling\_team='Kings XI Punjab'

score = score\_predict(batting\_team, bowling\_team, overs=12.3, runs=113, wickets=2, runs\_last\_5=55, wickets\_last\_5=0)

print(f'Predicted Score : {score} || Actual Score : 176')

**OUTPUT:**

Predicted Score : 187 || Actual Score : 176

### **Test 3**

* Batting Team : **Kolkata Knight Riders**
* Bowling Team : **Chennai Super Kings**
* Final Score : **172/5**

batting\_team="Kings XI Punjab"

bowling\_team="Rajasthan Royals"

score =score\_predict(batting\_team, bowling\_team, overs=14.0, runs=118, wickets=1, runs\_last\_5=45, wickets\_last\_5=0)

print(f'Predicted Score : {score} || Actual Score : 185')

**OUTPUT:**

Predicted Score : 172 || Actual Score : 172

**Conclusion:**

The aim of this study is to leverage historical data to predict both the final score and the outcome of matches. This research will encompass various stages of the data science process, including data preprocessing, visualization, preparation, selection, and the implementation of machine learning models. By combining these fields, we seek to accurately forecast the innings scores. To achieve this goal, we will apply a range of machine learning models to the relevant datasets.

As our approach well predicts the IPL in the current scenario that is based on the historical records, it can be further extended after youngsters join the team, their history records are made available. Moreover, new season data can be added, and adding some new features like head-to-head win which are beneficial in increasing accuracy.

As for the scope of the future, the focus can be on each player’s performance and evaluate that on a regular basis for the season. His ratings for bowling and batting can also be predicted. There can be a chance to predict the man of the match for the two teams.

**Related Works and References:**

This project was worked with certain important highlighting inputs from different research publications on the same, some of which are listed below.

1.

Preetham HK1 , Prajwal R2 , Prince Kumar 3 , Naveen Kumar4 1234UG-Students, Department of Computer Science Engineering, Dayananda Sagar Academy of Technology and Management, Bangalore, India. Cricket Score Prediction Using Machine Learning

<https://ijirt.org/master/publishedpaper/IJIRT157821_PAPER.pdf>

2.

"Prediction of IPL Match Score and Winner Using Machine Learning Algorithms", International Journal of Emerging Technologies and Innovative Research (www.jetir.org | UGC and issn Approved), ISSN:2349-5162, Vol.8, Issue 6, page no. ppc437-c444, June-2021, Available at : <http://www.jetir.org/papers/JETIR2106333.pdf>

3.

Prediction of IPL Match Outcome Using Machine Learning Techniques.

Department of CSE, SJB Institute of Technology, Affilated to Visveswaraya Technological University BGS Health &

Education City, Bengaluru-560060, Karnataka, India.Srikantiah K.C

<https://www.researchgate.net/publication/355061139_Prediction_of_IPL_Match_Outcome_Using_Machine_Learning_Techniques>

**VIVA VOICE QUESTION:**

1. **What is random forest algorithm?**

Random forest is an ensemble-based supervised learning methodology. Ensemble learning is a type of learning in which many decision trees are constructed and then combined to produce more accurate prediction models. The resulting of trees termed "Random Forest" is a mix of multiple decision trees. The random forest algorithm is not biassed because it is based on a majority vote and delivers the final prediction based on that voting.

1. **What is the difference between Linear Regression and SVR?**

Linear regression is a simple and widely-used statistical technique for modeling the relationship between a dependent variable (target) and one or more independent variables (features). It assumes that the relationship between the variables can be represented by a linear equation. The model aims to find the best-fitting straight line through the data points, minimizing the sum of the squared differences between the observed and predicted values. Linear regression is sensitive to outliers and assumes a linear relationship between variables. SVR is a variant of support vector machines (SVM) used for regression tasks. It finds the hyperplane that best fits the data, while also minimizing the margin violations (deviations of the actual target values from the predicted values). SVR uses a kernel function to map the input features into a higher-dimensional space, where the hyperplane is constructed. SVR is effective in capturing complex relationships in the data and is less sensitive to outliers compared to linear regression.

1. **What is XGBoost Regression?**

XGBoost (Extreme Gradient Boosting) is an implementation of gradient boosting decision trees designed for speed and performance. It builds an ensemble of weak learners (decision trees) sequentially, where each tree corrects the errors made by the previous ones. XGBoost uses a gradient descent algorithm to minimize a loss function, typically the mean squared error (MSE), and adds regularization terms to prevent overfitting. XGBoost is known for its high performance, scalability, and ability to handle large datasets with high-dimensional features

1. **Why did you choose Random Forest over other regressing algorithms?**

The reason why we chose Random Forest over other regressing methods was because it gave the highest accuracy out of all ( %).