**📘 Fantasy Points Prediction System — Full Documentation**

**📌 Overview**

This project trains a Machine Learning model using historical cricket **ball-by-ball commentary data** to predict **fantasy points** for players in upcoming matches. It supports appending new data, retraining, and UI-based prediction.

**Core files:**

* model1.py: ML pipeline + feature engineering
* train\_cricinfo\_model.py: Full training script
* cricinfo.csv: Combined raw match commentary data
* fantasy\_model.pkl: Saved trained model
* app\_with\_model.py: Streamlit prediction app

**🅰️ A. Integrating New Match Data**

**1. Convert SQL to CSV**

If you receive data in .sql format (e.g., INSERT INTO commentary VALUES (...)):

python convert\_sql\_to\_csv.py

This creates a file called new\_data.csv.

**2. Append New Data to Main Dataset**

import pandas as pd

old = pd.read\_csv("cricinfo.csv")

new = pd.read\_csv("new\_data.csv")

combined = pd.concat([old, new], ignore\_index=True)

combined.to\_csv("cricinfo.csv", index=False)

Make sure all columns match and types are consistent.

**🅱️ B. Training the ML Model**

Run this script to clean, preprocess, engineer features, and train the model:

python train\_cricinfo\_model.py

Outputs:

* feature\_importance.csv — feature impact ranking
* player\_predictions.csv — predictions per player
* fantasy\_model.pkl — saved model, encoders, and data for reuse

**🅲 C. Predicting Fantasy Points**

**Option 1: From Python Script**

from model1 import predict\_player\_fantasy\_points

predictions = predict\_player\_fantasy\_points(model, encoders, input\_df)

Where input\_df is a DataFrame with the same structure as X.

**Option 2: Via Streamlit UI**

You can use app\_with\_model.py to launch an interactive fantasy points predictor.

streamlit run app\_with\_model.py

**🅳 D. Streamlit App Integration (with Trained Model)**

**🔹 Save Model After Training**

Add this to the end of train\_cricinfo\_model.py:

import joblib

joblib.dump((model, encoders, X), "fantasy\_model.pkl")

This saves your trained model for reuse.

**🔹 Streamlit App: app\_with\_model.py**

import streamlit as st

import pandas as pd

import joblib

@st.cache\_resource

def load\_model():

model, encoders, X = joblib.load("fantasy\_model.pkl")

return model, encoders, X

st.set\_page\_config(page\_title="Fantasy Predictor", page\_icon="🏏")

st.title("🏏 Fantasy Points Predictor (Using Trained Model)")

model, encoders, X\_full = load\_model()

player\_names = sorted(X\_full['player\_name'].unique())

st.header("Team A Playing XI")

team\_a = st.multiselect("Select 11 players for Team A", player\_names, max\_selections=11)

st.header("Team B Playing XI")

team\_b = st.multiselect("Select 11 players for Team B", player\_names, max\_selections=11)

col1, col2 = st.columns(2)

with col1:

batting\_first = st.selectbox("Who is batting first?", ["Team A", "Team B"])

with col2:

toss\_winner = st.selectbox("Who won the toss?", ["Team A", "Team B"])

group\_id = st.text\_input("Enter Group/Match Name", "Qualifier 1")

if st.button("Predict Fantasy Points"):

if len(team\_a) != 11 or len(team\_b) != 11:

st.error("❌ Please select 11 players for each team.")

else:

innings\_map = {"Team A": 1, "Team B": 2}

all\_players = team\_a + team\_b

input\_df = X\_full[X\_full['player\_name'].isin(all\_players)].copy()

input\_df['innings\_number'] = input\_df['player\_name'].apply(

lambda name: innings\_map[batting\_first] if name in all\_players else 0

)

input\_df['toss\_winner'] = toss\_winner

input\_df['group\_id'] = group\_id

from model1 import predict\_player\_fantasy\_points

pred\_df = predict\_player\_fantasy\_points(model, encoders, input\_df)

pred\_df = pred\_df[['player\_name', 'innings\_number', 'predicted\_fantasy\_points']]

pred\_df = pred\_df.sort\_values(by='predicted\_fantasy\_points', ascending=False)

st.success("✅ Fantasy points predicted.")

st.dataframe(pred\_df)

st.download\_button("📥 Download CSV", data=pred\_df.to\_csv(index=False), file\_name="predictions.csv")

**🅴 E. File Structure**

project/

├── cricinfo.csv # Combined commentary data

├── convert\_sql\_to\_csv.py # SQL to CSV converter

├── train\_cricinfo\_model.py # Full model training script

├── model1.py # ML pipeline

├── fantasy\_model.pkl # Saved model + encoders

├── app\_with\_model.py # Streamlit app for prediction

├── feature\_importance.csv # Output after training

├── player\_predictions.csv # Output after training

├── requirements.txt # Python dependencies

**🅵 F. Installation & Requirements**

**Install dependencies:**

pip install -r requirements.txt

**Add streamlit if you want to use the UI:**

pip install streamlit

**requirements.txt includes:**

pandas

numpy

scikit-learn

matplotlib

seaborn

**🧠 Notes**

* Uses GradientBoostingRegressor as the prediction model
* Visual outputs include:
  + feature\_importance.png
  + predictions\_vs\_actual.png
* Retraining can be automated via cron jobs, GitHub Actions, or notebooks
* Use .pkl file to **load model instantly** instead of retraining

**📘 Machine Learning Model – Technical Documentation**

**🔹 Objective**

Predict fantasy points for cricket players based on ball-by-ball match data using supervised regression.

**🔧 Model Architecture**

| **Component** | **Details** |
| --- | --- |
| **Type** | Regression Model |
| **Algorithm** | Gradient Boosting Regressor (via sklearn.ensemble.GradientBoostingRegressor) |
| **Purpose** | Predict total fantasy points per player in each match |

**🔢 Input Data**

Each row represents a single **player’s performance in a match**.

**🧱 Feature Engineering**

All features are engineered in model1.py → functions like engineer\_player\_features, create\_player\_history\_features, and add\_match\_context\_features.

**🔹 Core Features per Player**

| **Feature** | **Description** |
| --- | --- |
| runs | Total runs scored |
| balls\_faced | Number of balls faced |
| strike\_rate | (runs / balls) \* 100 |
| wickets | Total wickets taken |
| overs | Total overs bowled (approx. from balls) |
| economy | Runs conceded / overs bowled |
| dot\_balls | Balls with zero runs |
| runs\_conceded | Total runs given as bowler |
| fours, sixes | Count of 4s and 6s |
| form\_direction | +1, 0, or -1 based on fantasy score trend |
| rolling\_avg\_fantasy, rolling\_avg\_sr, rolling\_avg\_economy, etc. | Rolling average over past 3 matches |
| player\_role | Categorical role: All-rounder, Bowler, etc. |
| team\_id, opponent\_team\_id | Contextual info from match metadata |

**🧠 Fantasy Point Calculation Rules (During Feature Engineering)**

Fantasy points are calculated using these cricket rules:

**🔹 Batting Points**

* +1 per run
* +2 per four
* +6 per six
* +20 bonus for 50s
* +40 bonus for 100s
* -5 for a duck (dismissed on 0)

**🔹 Bowling Points**

* +25 per wicket
* +4 per dot ball
* +8 per maiden over (approx.)
* Bonus:
  + +4 for 3-wicket haul
  + +8 for 4-wicket haul
  + +16 for 5-wicket haul

**🧪 Training Setup**

**📂 File: train\_cricinfo\_model.py**

Steps:

1. Load and clean cricinfo.csv
2. Generate match/player tables
3. Run full feature engineering
4. Split data into train/test (80/20)
5. Train model using GridSearchCV on:

param\_grid = {

'model\_\_n\_estimators': [100, 200],

'model\_\_learning\_rate': [0.1, 0.05],

'model\_\_max\_depth': [3, 5]

}

1. Evaluate using:

* MAE (Mean Absolute Error)
* RMSE (Root Mean Square Error)
* R² Score

1. Save:
   * fantasy\_model.pkl (model + encoders)
   * feature\_importance.csv
   * player\_predictions.csv
   * feature\_importance.png, predictions\_vs\_actual.png

**✅ Model Output**

* Fantasy points predicted per player (predicted\_fantasy\_points)
* Sorted list used for fantasy team recommendations
* Results usable in Streamlit or automated systems

**💾 Saving and Loading Model**

Saved using:

import joblib

joblib.dump((model, encoders, X), "fantasy\_model.pkl")

Loaded in Streamlit with:

model, encoders, X = joblib.load("fantasy\_model.pkl")

**🛠️ Extending or Retuning the Model**

Future contributors can:

* Modify model1.py to include new features (e.g., fielding, match location)
* Change the ML model to:
  + RandomForestRegressor
  + XGBoost
  + LightGBM
* Replace GridSearchCV with RandomizedSearchCV or Optuna
* Export model to ONNX or TensorFlow Lite for mobile apps

**🚀 Deployment**

* ✅ Compatible with Streamlit (app\_with\_model.py)
* 🔜 Easy to containerize with Docker (optional)
* 📦 Can be deployed to:
  + Streamlit Cloud
  + AWS EC2
  + GCP App Engine

**📂 File Overview (Related to ML)**

| **File** | **Role** |
| --- | --- |
| model1.py | Core ML logic (features + training) |
| train\_cricinfo\_model.py | Calls model1.py to train on full data |
| fantasy\_model.pkl | Saved trained model |
| app\_with\_model.py | Streamlit app that uses .pkl |
| cricinfo.csv | Cleaned training data |