RUNS PREDICTION

In this code, I predicted the cricket runs scored using a machine learning model (specifically a neural network implemented with TensorFlow/Keras) and handling missing player predictions. Here's an overview of what the code does:

Data Import and Preprocessing:

- The code starts by importing necessary libraries, reading data from 'cricket2.csv' into a DataFrame 'df', and displaying the initial rows and data information.
- It creates a new DataFrame 'new_df' by dropping columns 'player_name', 'match_date', and 'opposition' from 'df'.
- The feature matrix 'X' is created by dropping the 'runs_scored' column, and the target variable 'y' is set to 'runs_scored'.

Machine Learning Model (TensorFlow - Neural Network):

- The data is split into training and testing sets.
- Standardization is applied to scale the features using StandardScaler.
- A neural network model is defined using TensorFlow's Keras API. It consists of two hidden layers with ReLU activation functions and an output layer for regression.
- The model is compiled with 'adam' optimizer and 'mean_squared_error' loss function.
- The model is trained on the scaled training data with 60 epochs and a batch size of 32.

Model Evaluation:

- Predictions for the number of runs scored are made on the scaled test data using the trained neural network model.
- The R-squared (R2) score is calculated to evaluate the model's performance.

Player-Level Predictions:

- Predictions for runs scored are aggregated at the player level by calculating the mean and mode of predictions for each player.
- A DataFrame called 'player_predictions' is created to store player-wise predictions and actual runs scored.

Player-Level Evaluation:

- Mean Squared Error (MSE) and R2 scores are calculated separately for both mean and mode predictions at the player level.

Handling Missing Player Predictions:

- The code identifies missing predictions for specific player IDs (15, 18, 24, 28) and imputes default values (23, 26, 10, 15) for these missing predictions.

Submission Preparation:

- A previously generated sample submission CSV file ('cricket_predictionstf.csv') is loaded into a DataFrame named 'sub'.
- Predicted run values from 'runs_predd' are assigned to the 'runs' column in the submission DataFrame.

Saving Predictions:

- The final predictions are saved to a new CSV file named 'cricket_predictionstf.csv'.

The code focuses on predicting cricket runs scored using a neural network model implemented with TensorFlow/Keras. It also includes data preprocessing, model training, evaluation, player-level aggregation of predictions, handling of missing predictions, and saving the predictions for submission.

## WICKETS PREDICTION

Here's a summary of what this code does:

Data Import and Preprocessing:

- It starts by importing necessary libraries and reading data from 'cricket2--.csv' into a DataFrame 'df'.
- The initial rows and data information are displayed.

Data Preprocessing:

- A new DataFrame 'new_df' is created by dropping the 'player_name' and 'match_date' columns. Additionally, one-hot encoding is applied to the 'opposition' column to convert categorical data into a suitable format for machine learning.

Data Splitting:

- The feature matrix 'X' is defined by dropping the 'wickets' column, and the target variable 'y' is set to 'wickets'.
- The data is split into training and testing sets using train_test_split.

Hyperparameter Tuning:

- A parameter grid is defined with various hyperparameter options, including the number of estimators (trees), maximum depth of trees, and learning rate for boosting.

- An XGBoost Regressor is initialized, and a GridSearchCV is set up to perform hyperparameter tuning. GridSearchCV is used to find the best combination of hyperparameters that minimize the negative mean squared error.
- The best hyperparameters obtained from the grid search are used to initialize an XGBoost Regressor with the 'objective' set to "reg:squarederror."

Model Training and Evaluation:

- The model with the best hyperparameters is trained on the training data.
- Predictions for the number of wickets are made on the test data.
- The code evaluates the model's performance by calculating the Mean Squared Error (MSE) and R-squared (R2) scores.

Feature Importance Analysis:

- Feature importances are extracted from the trained model to understand the significance of different features in making wicket predictions. These importances are stored in a DataFrame and sorted by importance.

Player-Level Predictions:

- Predictions for wickets are aggregated at the player level by calculating the mean and mode of predictions for each player.
- A DataFrame named 'player_predictions' is created to store player-wise predictions and actual wickets taken.

Player-Level Evaluation:

- Mean Squared Error (MSE) and R2 scores are calculated separately for both mean and mode predictions at the player level.

Handling Missing Player Predictions:

- The code identifies missing predictions for specific player IDs (18, 28, 30) and imputes default values (0, 0, 2) for these missing predictions.

Submission Preparation:

- A sample submission CSV file ('sample_submission_br9x6st.csv') is loaded into a DataFrame named 'sub'.
- Predicted wicket values from 'wickets_predd' are assigned to the 'wickets' column in the submission DataFrame.

Saving Predictions:

- The final predictions are saved to a new CSV file named 'cricket_predictions2--.csv'.