Certification Course on Sports Data Analytics – Series 1

Cricket Metrics: From Data to Insights and Strategic Vision

PROJECT REPORT

Rahul Bharadwaj

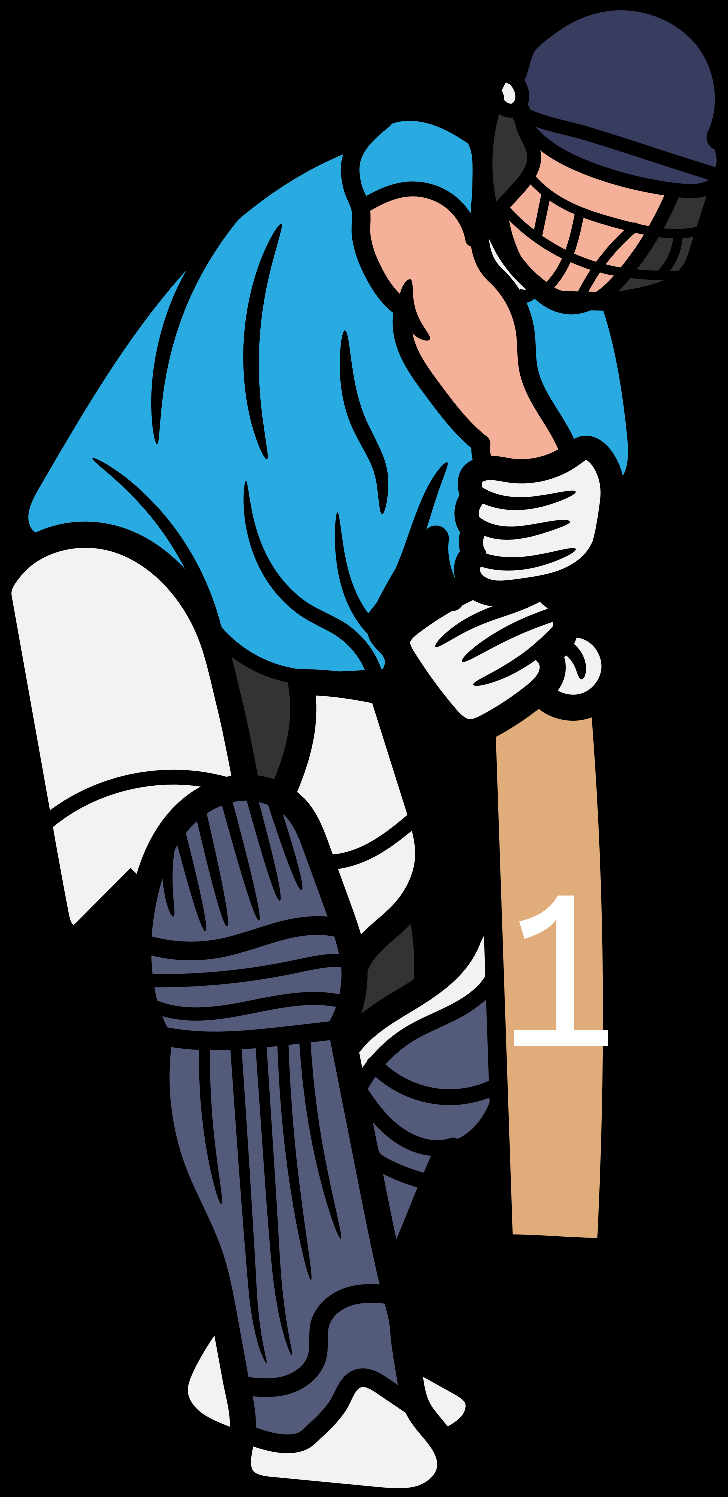
WOXSEN UNIVERSITY

Certification Course on Sports Data Analytics – Series 1
Cricket Metrics: From Data to Insights and Strategic Vision

Under the Guidence of →

Dr. Rajesh Kumar K V

USE CASE



Player Performance Analysis and Statistical Benchmarking

Player Performance Analysis and Statistical Benchmarking

Description:

This project will use synthetic_player_performance_data.csv to analyse and benchmark player performances. Statistical techniques such as z-scores, regression analysis, and ANOVA will be employed to compare players' performances against league averages and identify outliers. Power BI will be used to visualize these benchmarks, and machine learning models like K-Means Clustering will help categorize players based on performance metrics.

Steps:

Data Preprocessing: Clean the data and prepare it for analysis, including normalization of performance metrics.
Statistical Benchmarking: Use z-scores and regression analysis to benchmark individual player performance against the league or team averages.
Visualization: Create Power BI dashboards that highlight player performance against benchmarks, showing where each player stands relative to others.
Recommendation System: Use Decision Trees, enhanced with statistical insights, to provide personalized recommendations for player improvement.
Model Evaluation: Use statistical measures like silhouette score to evaluate clustering effectiveness and the accuracy of recommendations.

Expected Outcome:
A comprehensive analysis of player performance, offering statistically backed recommendations for improvement, with clear visualizations that help coaches and players understand where improvements are needed.

1. **Import Libraries :**
   - Imported `pandas` for data manipulation.
   - Imported `numpy` for numerical operations.
   - Imported `MinMaxScaler` from `sklearn.preprocessing` for normalization.

2. **Load the Dataset :**
   - Loaded the dataset from a CSV file into a DataFrame.

3. **Check for Missing Values :**
   - Checked for missing values in the DataFrame using `isnull().sum()`.

4. **Filter Data by Match Type :**
   - Filtered the DataFrame into separate DataFrames for T20, ODI, and Test match types.

5. **Define Columns for Metrics :**
   - Defined columns related to batting, bowling, and fielding metrics.

6. **Define Function to Remove Invalid Cases :**
   - Created a function `remove_invalid_cases` to filter out invalid data based on batting and bowling criteria.

7. **Define Function to Calculate and Normalize Metrics :**
   - Created a function `calculate_and_normalize_metrics` to calculate additional metrics
   - (Strike Rate, Boundary Percentage, Economy Rate) and normalize relevant metrics using MinMaxScaler.

8. **Apply Cleaning and Normalization Functions :**
   - Applied the `remove_invalid_cases` function to each match type DataFrame (T20, ODI, Test).
   - Applied the `calculate_and_normalize_metrics` function to the cleaned DataFrames for each match type.

9. **Create Separate DataFrames for Metrics :**
   - Created separate DataFrames for batting, bowling, and fielding metrics for each match type (T20, ODI, Test).

10. **Calculate Summary Statistics :**
    - Calculated summary statistics (e.g., average, count, min, max) for each match type using `describe()`.

11. **Output Summary Statistics :**
    - Printed the summary statistics for each format (T20, ODI, Test).

CELL: 1

CELL: 2

1. **Import Libraries :**
   - Imported `pandas` for data manipulation.
   - Imported `train_test_split` from `sklearn.model_selection` to split the data into training and test sets.
   - Imported `LinearRegression` from `sklearn.linear_model` for performing linear regression.
   - Imported `mean_squared_error` and `r2_score` from `sklearn.metrics` for evaluating regression models.

2. **Define Function to Calculate League Averages and Standard Deviations :**
   - Created `calculate_league_averages_and_stddev` to compute average values and standard deviations for specified columns in a DataFrame.

3. **Define Function to Calculate Z-Scores :**
   - Created `calculate_z_scores` to compute z-scores based on league averages and standard deviations. Added `Player_ID` and `Player_Name` for reference.

4. **Define Modified Function to Calculate Z-Scores and League Averages :**
   - Created `calculate_z_scores_and_averages` to filter data by format, calculate league averages and standard deviations, and then compute z-scores.

5. **Define Function for Regression Analysis :**
   - Created `perform_regression_analysis` to:
     - Split the data into training and test sets.
     - Initialize and train a `LinearRegression` model.
     - Make predictions and evaluate the model using `mean_squared_error` and `r2_score`.
     - Print the regression model coefficients, intercept, MSE, and R-squared score.

6. **Apply Regression Analysis for Batting Metrics :**
   - Defined the target (`Runs_Scored`) and feature columns (`Balls_Faced`, `Fours`, `Sixes`, `Strike_Rate`, `Boundary_Percentage`).
   - Performed regression analysis for batting metrics separately for T20, ODI, and Test formats.

7. **Apply Regression Analysis for Bowling Metrics :**
   - Defined the target (`Wickets_Taken`) and feature columns (`Overs_Bowled`, `Maiden_Overs`, `Runs_Conceded`, `Economy_Rate`).
   - Performed regression analysis for bowling metrics separately for T20, ODI, and Test formats.

8. **Apply Regression Analysis for Fielding Metrics :**
   - Defined the target (`Catches`) and feature column (`Runouts`).
   - Performed regression analysis for fielding metrics separately for T20, ODI, and Test formats.

CELL: 3

1. **Add 'Format' Column :**
   - Added a new column `Format` to each of the DataFrames (`df_t20_batting`, `df_odi_batting`, `df_test_batting`, `df_t20_bowling`, `df_odi_bowling`, `df_test_bowling`, `df_t20_fielding`, `df_odi_fielding`, `df_test_fielding`) to specify the format of the matches (T20, ODI, Test).

2. **Define Function to Print League Averages and Standard Deviations :**
   - Created the function `print_league_averages_and_stddev` which:
     - Filters the DataFrame based on the format type.
     - Calculates the league averages and standard deviations for specified columns.
     - Prints the results, including averages and standard deviations.

3. **Print League Averages and Standard Deviations :**
   - Applied the `print_league_averages_and_stddev` function to each format and metric set:
     - Batting metrics for T20, ODI, and Test formats.
     - Bowling metrics for T20, ODI, and Test formats.
     - Fielding metrics for T20, ODI, and Test formats.

CELL: 4

1. Define Recommendations

- Function : `generate_recommendations`
- Purpose : Create a dictionary of recommendations for batting, bowling, and fielding across different formats (T20, ODI, Test).
- Output : A dictionary containing specific recommendations for improving various performance metrics.

2. Add Format Column

- Purpose : Append a new column `'Format'` to each DataFrame to indicate the format (T20, ODI, Test).
- DataFrames Updated : `df_t20_batting`, `df_odi_batting`, `df_test_batting`, `df_t20_bowling`, `df_odi_bowling`, `df_test_bowling`, `df_t20_fielding`, `df_odi_fielding`, `df_test_fielding`.

3. Fetch Player Data

- Function : `get_player_data`
- Purpose : Retrieve player data from any of the DataFrames based on the `Player_ID`.
- Returns : Player data if found, otherwise returns `None`.

4. User Input

- Prompt : Request user to enter `Player_ID` and format type (`T20`, `ODI`, `Test`).
- Validation : Ensure that the format type is one of the valid options.

5. Generate and Display Recommendations

- Format Selection : Based on user input, select the appropriate DataFrames for batting, bowling, and fielding.
- Generate Recommendations : Call `generate_recommendations` with the selected DataFrames.
- Display Recommendations :
  - Print recommendations for batting , bowling , and fielding specific to the chosen format.
  - Provide actionable tips to improve performance metrics.

6. Error Handling

- Invalid Player ID : Print a message if the player ID is not found.
- Invalid Format : Print a message if the format is not one of the valid options.

Example Scenario

1. Add Format Column :
   - Each DataFrame is updated to include the `'Format'` column.
2. User Input :
   - User enters `Player_ID = 123` and `format = T20`.
3. Fetch Player Data :
   - The script looks through the DataFrames for `Player_ID = 123`.
4. Generate Recommendations :
   - If the player is found, the script generates recommendations using T20-specific data.
   - Tips for improving strike rate, boundary percentage, and runs scored are printed.
5. Display Recommendations :
   - The script displays tailored recommendations for batting, bowling, and fielding in the T20 format.
6. Handle Errors :
   - If the `Player_ID` is not found or the format is invalid, appropriate messages are displayed.

CELL: 5

1. Categorize Performance

- Function : `categorize_performance`
- Purpose : Categorize a player's performance into 'Too Good', 'Good', 'Average', or 'Poor' based on their value compared to league averages and standard deviations.
- Logic :
  - Too Good : Player's value is significantly above the league average.
  - Good : Player's value is above the league average but not significantly.
  - Average : Player's value is around the league average.
  - Poor : Player's value is below the league average.
  - Special Case : If `league_stddev` is 0 (no variation), default to 'Average'.

2. Generate Recommendations

- Function : `generate_recommendations_for_player`
- Purpose : Generate performance-based recommendations for a player based on their metrics.
- Process :
  - Retrieve the player's value for each metric.
  - Compare with league averages and standard deviations.
  - Categorize the performance using `categorize_performance`.
  - Generate actionable recommendations based on the performance category.

3. Fetch Player Data

- Function : `get_player_data`
- Purpose : Fetch player data based on `Player_ID` and format (T20, ODI, Test).
- Process :
  - Define a dictionary mapping formats to their respective DataFrames for batting, bowling, and fielding.
  - Retrieve the relevant DataFrames for the given format.
  - Filter DataFrames to get player-specific data for batting, bowling, and fielding.
  - Return a dictionary of player data for each aspect.

CELL: 5

4. Main Execution

- User Input :
  - Prompt the user to enter `Player_ID` and `format_type`.
  - Validate `format_type` (T20, ODI, Test).
- Fetch and Display Player Data :
  - Call `get_player_data` to retrieve player data.
  - If data is found, calculate league averages and standard deviations for the selected format using `calculate_league_averages_and_stddev`.
  - Print player records for the selected format.
- Generate and Display Recommendations :
  - For each aspect (batting, bowling, fielding), determine relevant metrics.
  - Call `generate_recommendations_for_player` to get performance and recommendations for each metric.
  - Print performance categorization and recommendations.
- Handle Errors :
  - If the player ID is not found, display an error message.
  - If the format is invalid, display a message prompting for valid input.

Example Execution

1. User Input :- Enter `Player_ID = 123`. - Enter `format = T20`.
2. Fetch Player Data :
   - Retrieve player data for ID 123 from T20 DataFrames.
3. Calculate League Averages and Stddevs :
   - Calculate league averages and standard deviations for T20 batting, bowling, and fielding metrics.
4. Display Player Data :
   - Print player data for batting, bowling, and fielding aspects.
5. Generate Recommendations :
   - Based on player's performance in each aspect, generate recommendations and categorize performance.
6. Display Recommendations :
   - Print performance and recommendations for each metric in batting, bowling, and fielding.

1. Data Preparation :
- The script starts by combining separate data sets (`df_t20_batting`, `df_odi_batting`, etc.) for different cricket formats (T20, ODI, Test) and player roles (batting, bowling, fielding) into a single `df_combined` DataFrame.
- It then creates a `Player_Type` column that labels the player as either a Batsman, Bowler, or All-Rounder based on their statistics.

2. Feature Selection :
- A set of relevant features (`Runs_Scored`, `Balls_Faced`, `Fours`, `Sixes`, `Wickets_Taken`, etc.) is selected for model training, focusing on core statistics for batting, bowling, and fielding.

3. Model Training :
- A Random Forest Classifier is used to train the model. The data is split into training and test sets using an 80-20 split, and the model is trained on the `X_train` set.
- After training, predictions are made on the `X_test` set, and the model's accuracy and classification report are displayed.

4. Prediction Function :
- The function `predict_player_type()` predicts the player type for each player based on their data across all formats. It retrieves the respective statistics for each format, combines them, and applies the trained model to predict the player type.
- If the player's data is missing, it returns an appropriate message.

5. Applying Predictions :
- The model is applied across all rows in the combined dataset (`df_combined`), predicting the player type for each row.
- The predictions are stored in a new column, `Predicted_Player_Type`.

6. Save Results :
- The resulting DataFrame, with actual and predicted player types, is saved to an Excel file (`player_types_with_predictions.xlsx`), enabling further analysis.

Potential Improvements and Considerations:

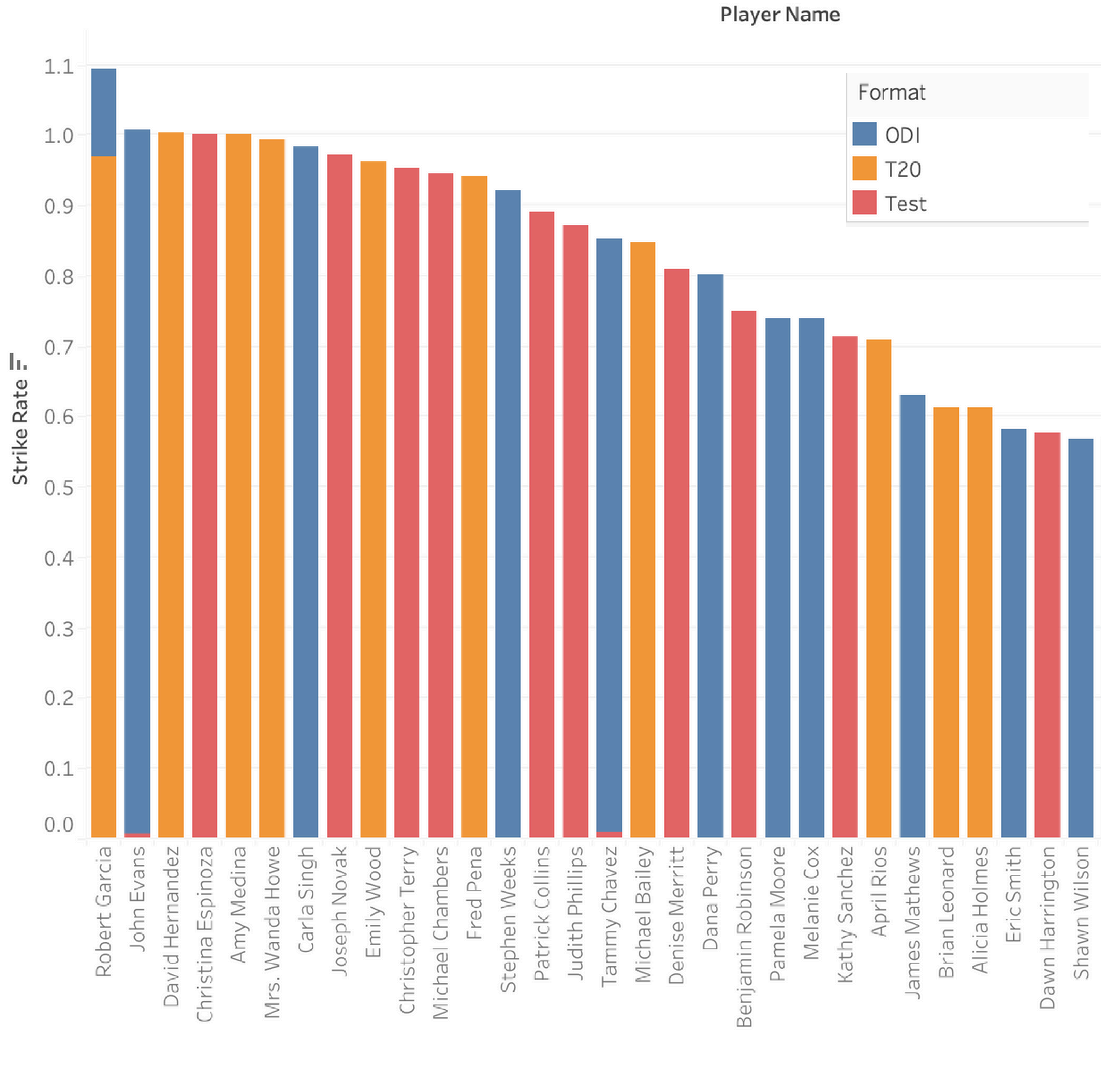
- Edge Cases : Ensure that the model works well even when certain player data is missing. The code already fills missing values with 0, but you may want to explore more sophisticated data imputation techniques.

- Model Tuning : You can improve the model by tuning hyperparameters of the `RandomForestClassifier`, such as `n_estimators` (number of trees), `max_depth` (depth of trees), etc.

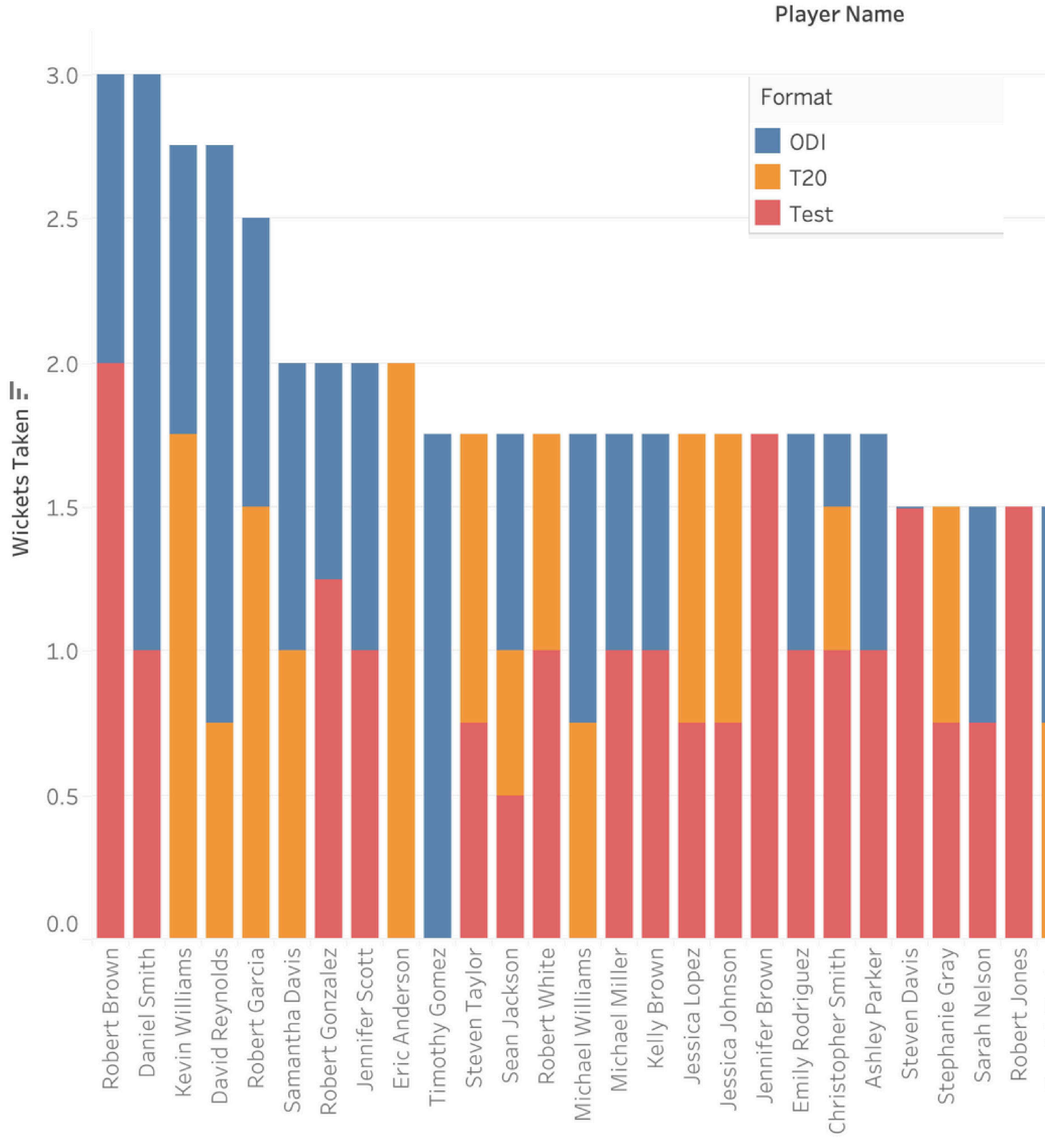
CELL: 6

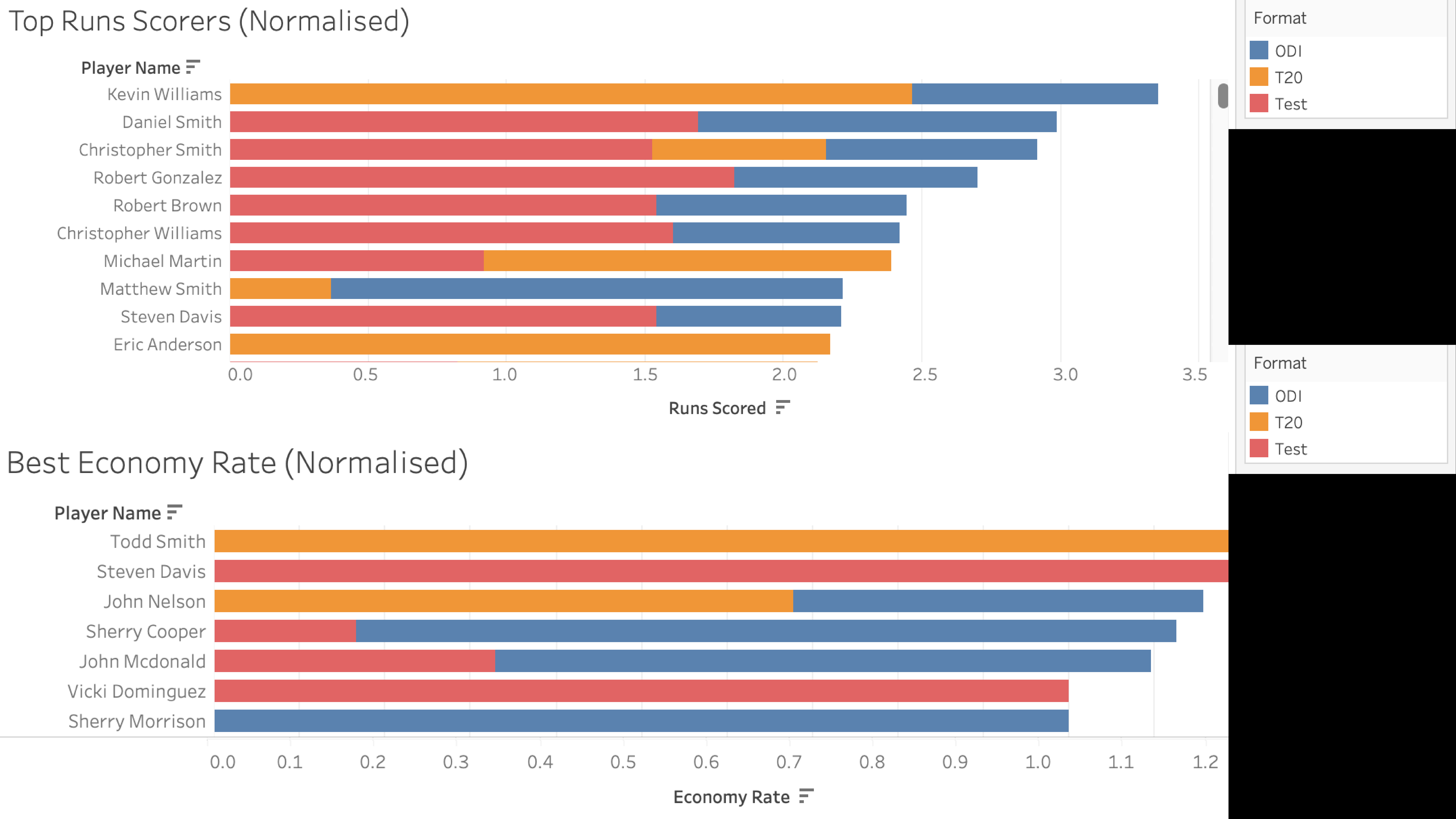
VISUALIZATIONS

Top Strike Rate (Normalised)



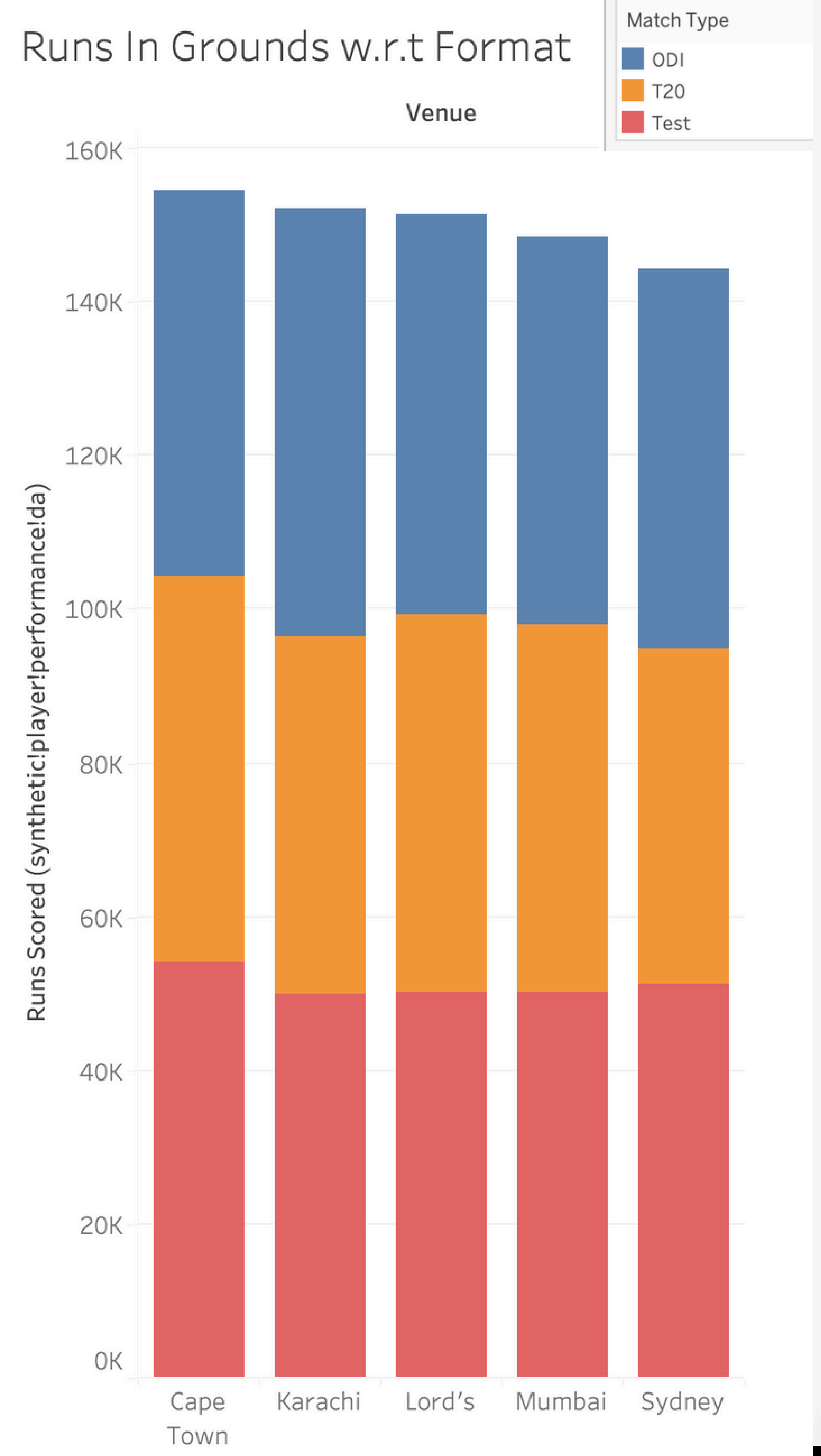
Top Wicket Takers (Normalised)

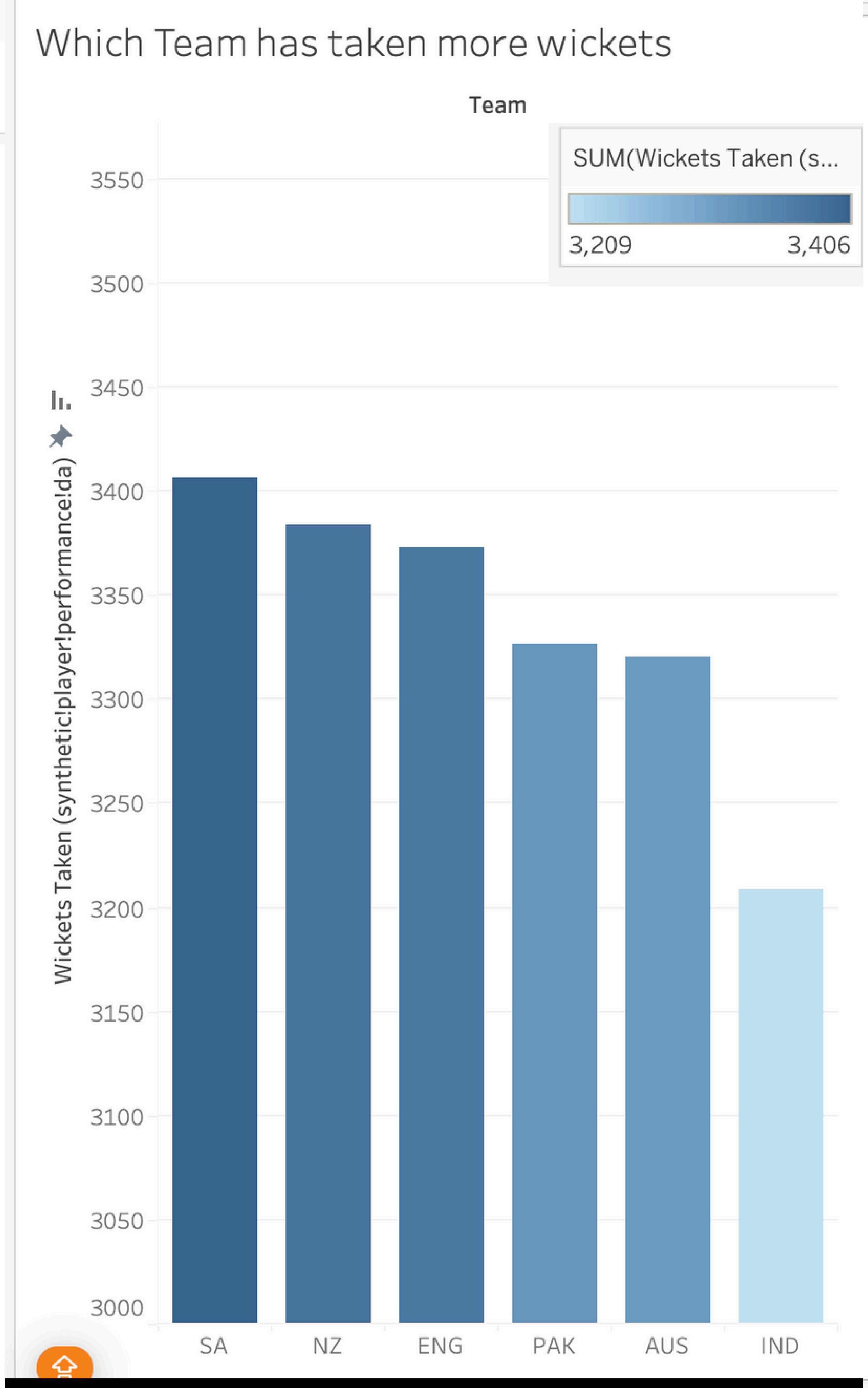


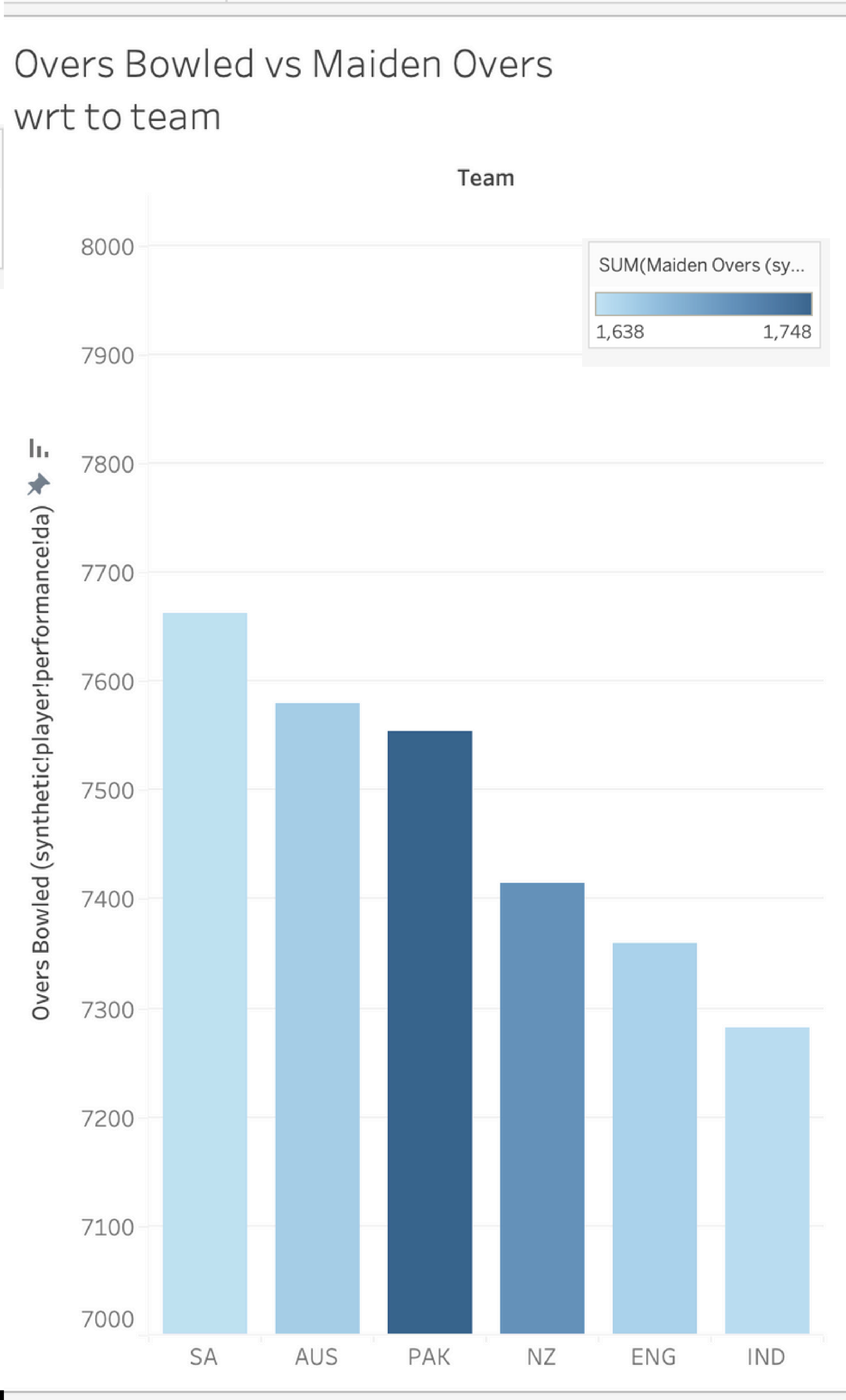


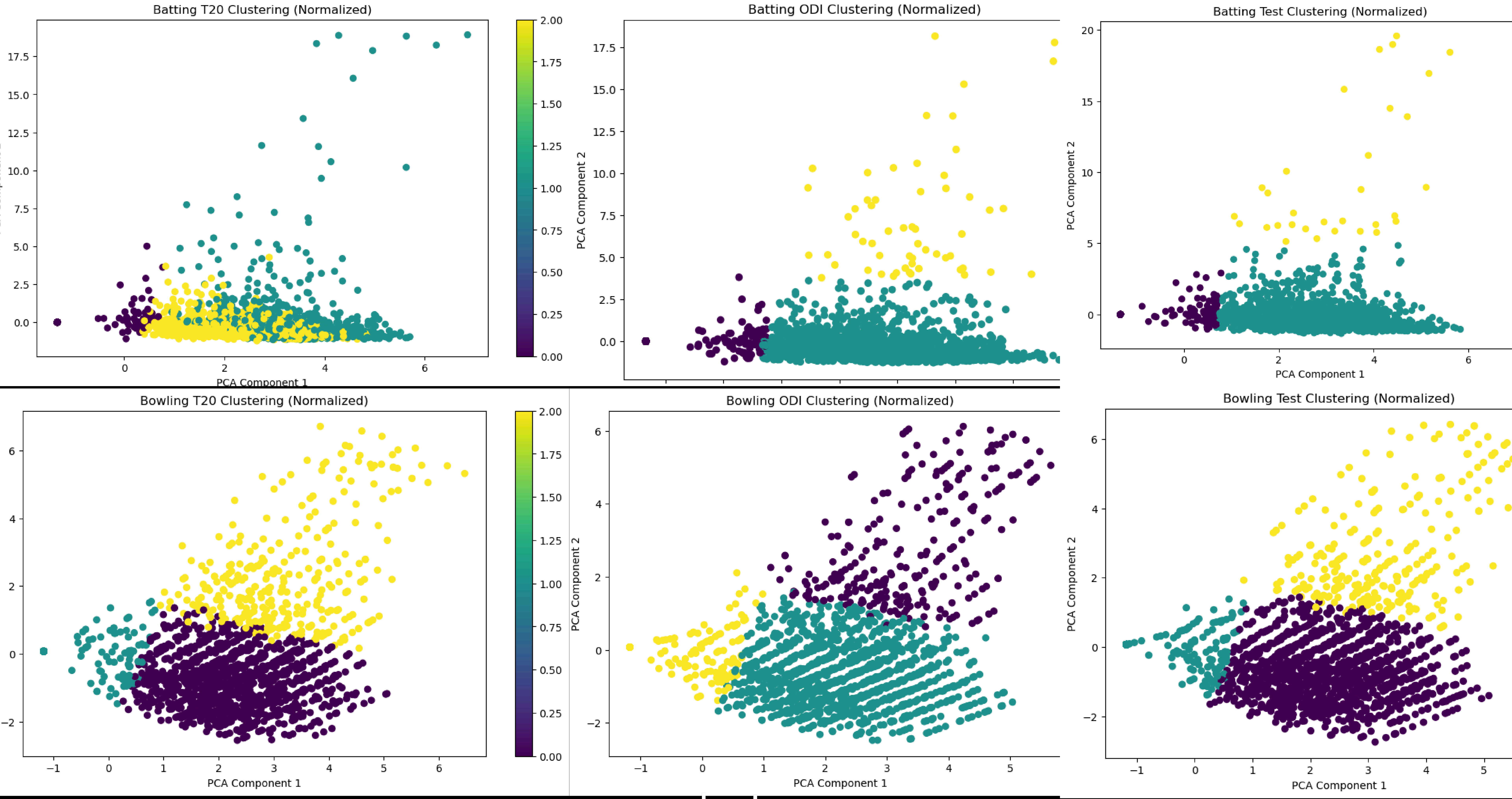
Top Runs Scorers (Normalised)

Best Economy Rate (Normalised)


Runs In Grounds w.r.t Format


Which Team has taken more wickets


Overs Bowled vs Maiden Overs wrt to team


Batting T20 Clustering (Normalized)
PCA Component 1
0
2
4
6
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Batting ODI Clustering (Normalized)
PCA Component 2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Batting Test Clustering (Normalized)
PCA Component 1
PCA Component 2
0
2
4
6
0
5
10
15
20
Bowling T20 Clustering (Normalized)
PCA Component 1
−1
0
1
2
3
4
5
6
−2
0
2
4
6
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Bowling ODI Clustering (Normalized)
PCA Component 1
PCA Component 2
−1
0
1
2
3
4
5
−2
0
2
4
6
Bowling Test Clustering (Normalized)
PCA Component 1
PCA Component 2
−1
0
1
2
3
4
5
−2
0
2
4
6

clusters

USE CASE

2

USE CASE

Team Strategy Optimization Using Statistical and Machine Learning Approaches

Team Strategy Optimization Using Statistical and Machine Learning Approaches

Problem Statement:

How can we optimize cricket team strategies using statistical analysis of historical match data combined with machine learning models?

Description:

This project will analyse the synthetic_cricket_match_data.csv to optimize team strategies. Statistical methods such as time series analysis, factor analysis, and regression models will be used to identify key strategic elements (e.g., batting order, bowling changes) that influence match outcomes. Power BI will provide visual insights into these strategies, and machine learning models like Random Forest will be used to predict the best strategies under various match conditions.

Steps:

Data Preprocessing: Prepare the data for analysis, including time series decomposition and transformation.
Statistical Analysis: Apply time series analysis and factor analysis to identify trends and key factors that impact match outcomes.
Visualization: Develop Power BI dashboards that illustrate the statistical relationships and trends in team strategies over time. (Or)
Strategy Modelling: Implement Random Forest models to predict the optimal team strategy, incorporating statistically significant factors.
Optimization and Simulation: Use simulation techniques (e.g., Monte Carlo simulation) to test the effectiveness of different strategies under varying conditions.
Model Evaluation: Evaluate the model using statistical metrics like R-squared, mean absolute error, and cross-validation results.

Expected Outcome:
A data-driven strategy optimization model that uses statistical analysis and machine learning to recommend the best team strategies, with clear visual outputs that assist in decision-making during matches.

Data Loading and Preprocessing Steps:

1. Imported Libraries:
   - pandas: For data manipulation and analysis.
   - sklearn.model_selection.train_test_split: For splitting data into training and test sets.
   - sklearn.preprocessing.LabelEncoder: For encoding categorical features.
   - sklearn.ensemble.RandomForestClassifier: For building a random forest classifier.
   - sklearn.metrics.accuracy_score, classification_report: For evaluating model performance.
   - joblib: For saving and loading models.
   - seaborn: For creating statistical graphics.
   - matplotlib.pyplot: For creating visualizations.
   - numpy: For numerical operations.
2. Data Loading:
   - Loaded data from a CSV file into a DataFrame using pandas.
3. Data Preprocessing:
   - Dropped rows with missing values and displayed the first few rows to check the DataFrame structure.

CELL: 1

CELL:2

CELL:3

Correlation Analysis:

1. Correlation Between Match Winner and Toss Winner:
   - Encoding:
     - Encoded `Match_Winner` and `Toss_Winner` using `LabelEncoder`.
   - Calculation:
     - Computed correlation between encoded `Match_Winner` and `Toss_Winner`.
   - Visualization:
     - Scatter Plot: Plotted `Toss_Winner_Encoded` vs. `Match_Winner_Encoded` with hues and styles for better visualization.
     - Heatmap: Displayed a correlation heatmap for `Toss_Winner_Encoded` and `Match_Winner_Encoded`.
   - Result: Printed the correlation value between Toss Winner and Match Winner.

2. Correlation Between Venue and Toss Decision:
   - Encoding:
     - Encoded `Venue` and `Toss_Decision` using `LabelEncoder`.
   - Calculation:
     - Computed correlation between encoded `Venue` and `Toss_Decision`.
   - Visualization:
     - Scatter Plot: Plotted `Venue_Encoded` vs. `Toss_Decision_Encoded` with hues and styles.
     - Heatmap: Displayed a correlation heatmap for `Venue_Encoded` and `Toss_Decision_Encoded`.
   - Result: Printed the correlation value between Venue and Toss Decision.

3. Correlation Between Match Winner and Toss Decision:
   - Encoding:
     - Encoded `Match_Winner` and `Toss_Decision` using `LabelEncoder`.
   - Calculation:
     - Computed correlation between encoded `Match_Winner` and `Toss_Decision`.
   - Visualization:
     - Scatter Plot: Plotted `Match_Winner_Encoded` vs. `Toss_Decision_Encoded` with hues and styles.
     - Heatmap: Displayed a correlation heatmap for `Match_Winner_Encoded` and `Toss_Decision_Encoded`.
   - Result: Printed the correlation value between Match Winner and Toss Decision.

CELL: 4

CELL:5

CELL:6

Correlation Analysis:

1. Correlation Between Runs Difference and Venue:
   - Calculate Runs Difference:
     - Computed the difference in runs between `Team1` and `Team2`.
   - Encode Venue:
     - Encoded `Venue` using `LabelEncoder`.
   - Calculate Correlation:
     - Computed the absolute correlation between `Runs_Difference` and `Venue_Encoded`.
   - Visualization:
     - Scatter Plot: Plotted `Venue_Encoded` vs. `Runs_Difference`.
     - Heatmap: Displayed a correlation heatmap for `Runs_Difference` and `Venue_Encoded`.
   - Result: Printed the correlation value.

2. Correlation Between Toss Winner and Team1/Team2 Runs:
   - Encode Toss Winner:
     - Encoded `Toss_Winner` using `LabelEncoder`.
   - Calculate Correlation:
     - Computed correlations between `Toss_Winner_Encoded` and `Team1_Runs` as well as `Toss_Winner_Encoded` and `Team2_Runs`.
   - Visualization:
     - Scatter Plots: Plotted `Toss_Winner_Encoded` vs. `Team1_Runs` and `Toss_Winner_Encoded` vs. `Team2_Runs`.
     - Heatmaps: Displayed correlation heatmaps for `Toss_Winner_Encoded` vs. `Team1_Runs` and `Toss_Winner_Encoded` vs. `Team2_Runs`.
   - Result: Printed correlation values for both `Team1` and `Team2` runs.

3. Correlation Between Match Winner and Team1/Team2 Runs:
   - Encode Match Winner:
     - Encoded `Match_Winner` using `LabelEncoder`.
   - Calculate Correlation:
     - Computed correlations between `Match_Winner_Encoded` and `Team1_Runs` as well as `Match_Winner_Encoded` and `Team2_Runs`.
   - Visualization:
     - Scatter Plots: Plotted `Match_Winner_Encoded` vs. `Team1_Runs` and `Match_Winner_Encoded` vs. `Team2_Runs`.
     - Heatmaps: Displayed correlation heatmaps for `Match_Winner_Encoded` vs. `Team1_Runs` and `Match_Winner_Encoded` vs. `Team2_Runs`.
   - Result: Printed correlation values for both `Team1` and `Team2` runs.

CELL: 7

CELL:8

CELL:9

Match Winner Prediction System:

1. Data Preparation:

- Load Data: Read data from a CSV file into a DataFrame.
- Adjust Winner Columns: Modify `Match_Winner` and `Toss_Winner` columns to reflect the actual team names.
- Create Encodings:
  - Generate encoding dictionaries for categorical columns (`Team1`, `Team2`, `Venue`, `Toss_Winner`, `Toss_Decision`, `Match_Winner`).
  - Create reverse encodings for decoding predictions.

2. Data Encoding:

- Encode DataFrame: Transform categorical data into numerical format using the encoding dictionaries.

3. Feature and Target Preparation:

- Select Features: Use encoded columns for features.
- Define Target: Use the encoded `Match_Winner` as the target variable.

4. Model Training and Evaluation:

- Train-Test Split: Split the data into training and testing sets.
- Train Model: Fit a `RandomForestClassifier` on the training data.
- Evaluate Model: Assess the model's accuracy on the test set and print the accuracy score.

5. Prediction Function:

- Predict Match Winner:
  - Encode input features.
  - Predict the match winner using the trained model.
  - Decode the prediction to get the team name.

6. User Interaction:

- Get User Input:
  - Use `tkinter` to prompt users for input on team names, venue, toss winner, and toss decision.
  - Validate user inputs and ensure they match the predefined options.
- Display Prediction:
  - Show the predicted match winner using a message box.
  - Print the result in the console for debugging.

CELL: 10

1. Confusion Matrix:
- Displays the model's accuracy in predicting match winners by comparing true labels with predicted labels for each team.
- The matrix helps identify where the model is making errors.

2. Classification Report (Precision, Recall, F1-Score):
- Visualizes precision, recall, and F1-scores for each team.
- Shows how well the model predicts the match winner for individual teams.

3. Model Accuracy Over Multiple Runs:
- Plots model accuracy over multiple training iterations, highlighting how consistent the model is in predicting match outcomes.

4. Average Runs per Venue:
- Bar plot showing the average total runs scored at each venue by both teams.
- Useful for identifying high and low-scoring grounds.

5. Team Performance by Venue:
- Bar plot illustrating how different teams perform (in terms of average runs) at various venues.
- Helps to compare venue-specific performances for different teams.

6. Toss Decision Percentage by Venue:
- Visualizes the toss decision (bat or bowl) percentages for each venue.
- Provides insights into venue-specific trends in toss decisions.

7. Impact of Toss Decision on Match Outcome:
- Bar plot showing how toss decisions (bat or bowl) impact match outcomes.
- Helps identify if batting or bowling first has a stronger correlation with winning.

8. Team Win Percentage by Venue:
- Illustrates the win percentage of different teams at various venues.
- Provides insight into which teams perform better at specific grounds.

9. Overall Match Outcome Distribution:
- A count plot showing the number of wins for each team across all matches.
- Useful for understanding overall team success rates.

CELL: 11

CELL: 12

CELL: 13

1. Team and Venue Mappings:
- Maps team abbreviations (e.g., 'PAK' to 'Pakistan') and venues (e.g., 'Karachi' to 'Pakistan') to their respective countries for easier analysis of home and away matches.

2. Loading Data:
- Loads match outcome data from a CSV file into a Pandas DataFrame.

3. Adjusting Toss and Match Winner Columns:
- Adjusts the `Toss_Winner` and `Match_Winner` columns by mapping values such as 'Team1' or 'Team2' to their actual team names.

4. Home vs. Away Assignment:
- Defines a function to determine if Team1 is playing at home or away based on the country associated with the team and the venue.
- Adds a new column, `Home_Away`, indicating whether Team1's match is a home or away game.

5. Win Percentage Calculation:
- Calculates the win percentage for home and away matches by comparing the number of wins for Team1 in home matches vs. total home matches (and similarly for away matches).

6. Average Runs Calculation:
- Calculates the average runs scored by both Team1 and Team2 in home and away matches.
- Combines these stats into a new DataFrame that includes the win percentage and an "advantage" score for home matches (difference between Team1's and Team2's runs).

7. Visualizations:
- Bar Plot for Average Runs (Home vs. Away): A bar plot comparing the average runs scored in home and away matches.
- Bar Plot for Win Percentage (Home vs. Away): A bar plot showing the win percentage for home and away matches.

8. Home Advantage:
- Prints the home advantage in terms of average runs difference between Team1 and Team2 for home matches.

The analysis highlights the relationship between home-field advantage, win percentage, and run performance in home and away matches.

1. Data Preprocessing:
- The data is loaded from a CSV file into a Pandas DataFrame.
- The 'Match_Winner' and 'Toss_Winner' columns are adjusted so that they represent the actual winning team, rather than generic labels ('Team1' or 'Team2').

2. Encoding Categorical Data:
- Categorical columns such as 'Team1', 'Team2', 'Venue', 'Toss_Winner', 'Toss_Decision', and 'Match_Winner' are encoded into numerical values using a dictionary for each column.
- Reverse encodings are also created for decoding the encoded predictions back to their original labels.

3. Feature Selection:
- The encoded columns ('Team1_Encoded', 'Team2_Encoded', 'Venue_Encoded', 'Toss_Winner_Encoded', 'Toss_Decision_Encoded') are used as features, while 'Match_Winner_Encoded' is used as the target.

4. Train-Test Split:
- The dataset is split into training and testing sets using an 80/20 ratio.

5. Model Training:
- A **RandomForestClassifier** model is trained on the training data (features and target).

6. Monte Carlo Simulation:
- A Monte Carlo simulation is run to simulate 1000 random matches between teams. For each simulation:
  - Teams, venues, toss winners, and toss decisions are randomly selected.
  - The model predicts the match winner, and win counts are tracked for each team.
  - Win percentages for all teams are calculated and displayed.

7. Plotting Simulation Results:
- A bar plot is created to visualize the win percentages from the **Monte Carlo simulation** for each team.

8. Model Evaluation:
- The model is evaluated using cross-validation (accuracy scores from **5-fold cross-validation** are displayed).
- **Mean absolute error** (**MAE**) **i**s calculated based on the model's predictions.
- A **scatter plot** of actual vs. predicted match winners is shown, with a reference diagonal line to indicate perfect predictions.

The code performs data encoding, model training, simulations, and model evaluation, with clear visualizations for both simulation results and model performance.

CELL: 14

THANKS AND REGARDS

-**Dr. Rajesh Kumar K V :** Dr. Rajesh's sessions on Power BI and Tableau were a masterclass in data visualization. His clear and engaging teaching style made complex concepts accessible **and enjoyable.**

- Shahid Mohammad Ganie: Dr. Shahid's teachings on Python fundamentals, data collection and manipulation, and advanced libraries (Seaborn for data visualization and Scikit-Learn for machine learning) provided a solid foundation in data analytics.

- Ram Kumar N.: Mr. Ram Kumar provided an invaluable opportunity to interact with cricket analysts and data scientists, offering deep insights into the industry and its real-world applications.

- Saiprasad Kagne: Mr. Saiprasad's expertise in data gathering and management, as well as his real-world project experiences, have been instrumental in enhancing our practical understanding of sports data analytics.

- Dr. Boya (**Venkatesu**)**:** Dr. Boya's sessions on statistical modeling were exceptionally enlightening. He adeptly covered both descriptive and inferential statistics, showcasing their application in cricket data analysis.