Report for CS564 Final Project

Title of the Project: Cricket Pulse

Name of the students: Andrew Sabee, Sriharsha Devarapu, Harsh Sahu

I. Introduction

Report Organization

The report below is broken up into three sections: We begin with the introduction, which will give an outline of the project, alongside discussing how and why we were motivated to create the app we did. The second section covers in more detail the implementation of our app, the dataset used, and the tests performed to ensure our app was up to par. Finally, the report concludes with findings and lessons learned from the creation of this app. All group members contributed to all parts of this report.

Motivation

Cricket's popularity has been increasing rapidly over the past few years, and it is now the second most popular sport worldwide. However, there is a lack of analytical and visualization tools available for the sport. The purpose of our app is to provide an analytical tool for fans and teams to make data-driven decisions and observations based on the history of their players. This will allow for more precise analysis and adjustments by teams, and allow fans interested in data-gathering to understand on a deeper level the players they are cheering for. Team owners can analyze player performance statistics and make informed decisions, especially during player auctions or transfers. The motivation for making this a database-driven app lies in the size of the data – our ball-by-ball table alone is 150,000 entries, far too large to access efficiently with conventional methods.

Application Description

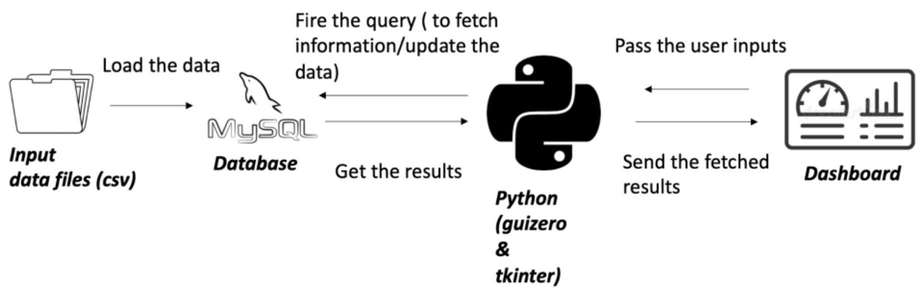
Our application will be a chart of play-by-play data of the cricket league (IPL - Indian Premier League) for games up to 2017. The user will be able to sort/query data by player, team, match and ball-by-ball metrics to be able to see breakdowns across different areas of the sport. The goal is to allow the user to see how players and teams performed both across multiple seasons of play and also at different conditions within a single match (i.e, maybe a player performs better (with a higher hit rate) at over 1 compared to over 10). Users will be able to add notes related to a player or a team. These notes would contain the user's thoughts or opinions on the data they were looking at.

We have broken down the query options into 3 different pages, shown below in the "Implementation" section of this report. The first page is Match Search, which will display information related to either a specific match or an aggregate of matches. Depending on the filters selected, it could display anything from a single match's data up to the sum across the entire dataset. Data displayed on this page is not designed for team- or player-specific, but rather looks at offensive and defensive stats, such as runs and wickets by over. The second page is Player Comparison, which allows the user to compare two players against each other in abstract, or compare their performance against a specific team. Individual stats are prioritized here, such as the player's strike rate and the number of wickets they have taken. The final page is Team Comparison, which performs a very similar task to the Player Comparison page, but for teams instead of individual players. Screenshots of all of these pages are provided below.

II. Our Implementation

II.1 Description of the system architecture

While compartmentalization of our code would have been ideal, the time constraints of this project forced us to make some compromises regarding the separation between GUI and backend code. All of our code is within two files, GUI.ipynb and utils.py. GUI.ipynb is the central code for the application, and constructs the GUI for each individual page of our app. It also contains much of the code for updating these pages when a query is run. utils.py serves loosely as our "backend," and constructs the SQL queries based on the filters the user has selected to use on a given page, and passes that query into the database to get the data the user is looking for. We also have SQL query files as .txt files (like page_1_query1.txt) for running individual queries to fetch the corresponding information from the Database.



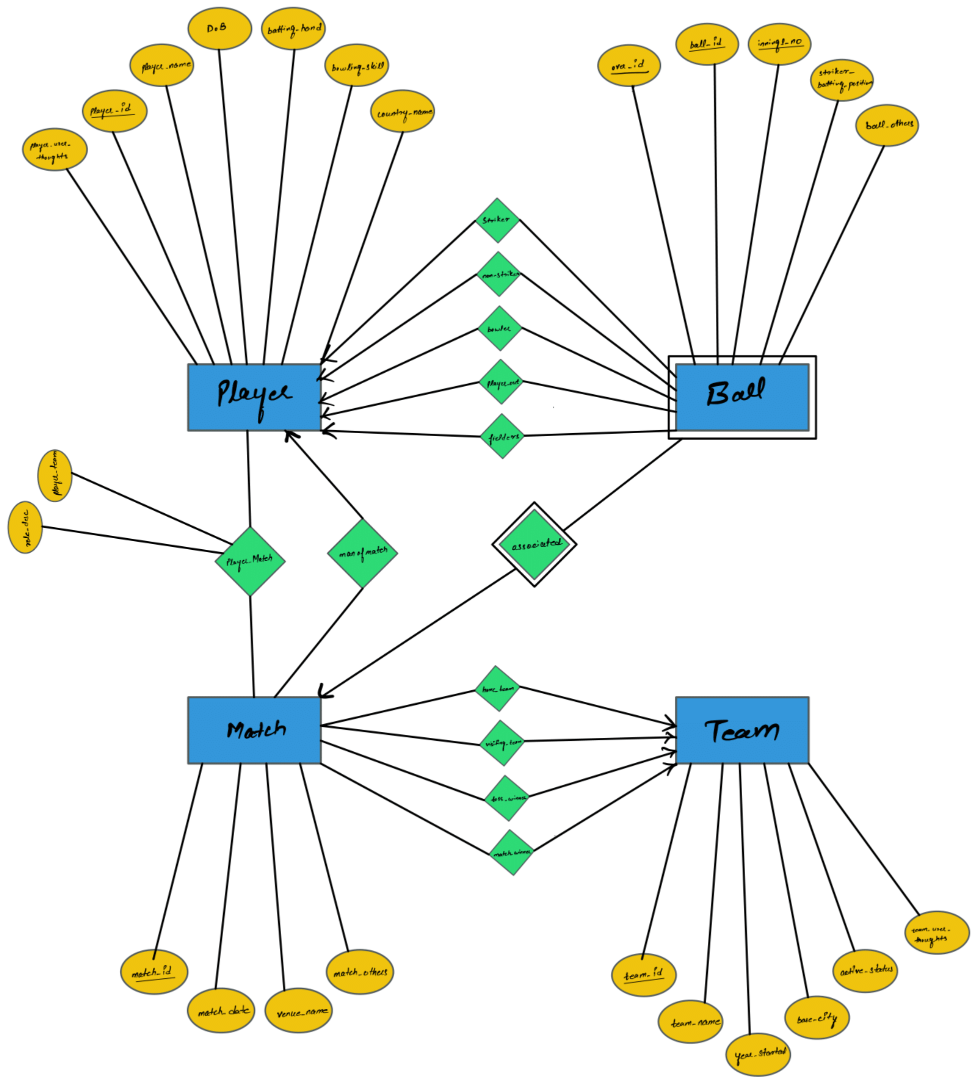
Overall architecture

II.2 Description of the dataset

Our dataset comes from data.world, and initially included 5 tables, one of which had over 150,000 entries in it. After cleaning the data (by replacing team and player names with IDs wherever applicable, correcting the date format) and normalizing our schema, our final dataset had 7 tables, stored as .csv files.

Team.csv is relatively simple, storing information about a team's ID, home city, and the user's thoughts on the team. Staidum.csv is simply the stadium name and city. Player.csv stores information about a player such as their batting/bowling skill, and also the user's thoughts on said player. Player_match.csv represents the relationship between a player and a given match, and stores information such as who played in what match, and what team they played for in that match. Match.csv stores information on each match that was played, such as what teams were involved and the margin of victory of the match. Location.csv is simply a list of all the locations (both city and country) where matches have been hosted. Ball_by_ball.csv, the largest file in our dataset by far, contains the information for each ball bowled in IPL games from 2008-2017. It contains information about who was bowling, who was batting, when this ball was bowled within the match, and the end result of the ball.

II.3 ER diagram (High-resolution PDF here)



- Additional attributes that could not be mentioned in the ER Diagram:

ball_others : extra_type, runs_scored, extra_runs, out_type

match_others : city_name, country_name, toss_decision, outcome_type, manofmatch, win_margin

II.4 Relational model

Underlined attributes are the key attributes of the given relation

Player (Player_ID:Number, Player_user_thoughts:String, Player_name:String, DoB:yyyy-mm-dd, batting_hand:String, bowling_skill:string, country_name:string)

Ball (Match_ID:number, Over_no:number, Ball_no:number, Innings_no:number, striker_batting_position:number(1-11), extra_type:string, runs_scored:number, extra_runs:number, out_type:string, Striker:number, non_striker:number, Bowler:number, Player_out:number, Fielders:number)

Match (Match_ID:number, Match_date:yyyy-mm-dd, Venue_name:string, toss_decision:string, Outcome_type:string, Win_margin:number, man_of_match_player_id:number, home_team_id:number, visiting_team_id:number, toss_winner_id:number, match_winner_id:number)

(home_team, match_date), (visiting_team, match_date), are secondary keys

Stadium (Venue_name:string, City_name:string)

Location (City_name:string, Country_name:string)

Team (Team_ID:number, Team_name:string, Year_started:number, Base_city:string, active_status:string, Team_user_thoughts:string)

Team_name is a secondary key

Player_Match (Match_ID:number, Player_ID:number, Role_desc:string, player_team_id:number)

Normalization

We had to do some normalization of the data from the initial dataset in order to have our schema in BCNF. Below we detail the decomposition process and demonstrate that each table ends up in BCNF.

Table: Ball
FD = {match_id, over_id, ball_id, innings_no → striker_batting_position, extra_type, runs_scored, extra_runs, out_type, striker, non_striker, bowler, player_out, fielders}
This table is in both 3NF and BCNF as the LHS of the only FD (match_id, over_id, ball_id, innings_no) is a super key

Table: Match

Note: Assuming one team plays only one match in one day. One city name belongs to one country
FD = {match_id → home_team, visiting_team, match_date, venue_name, city_name, country_name, toss_winner, match_winner, toss_decision, outcome_type, manofmatch, win_margin
venue_name → city_name, country_name
city_name → country_name
home_team, match_date → match_id
visiting_team, match_date → match_id }

Based on the above-defined functional dependencies, Match is *not in 3NF*, and the following functional dependencies violate 3NF: venue_name → city_name, country_name & city_name → country_name
As part of 3NF decomposition, we will apply BCNF decomposition first
We will take *venue_name → city_name, country_name* to begin the decomposition:

- Decompose Match into R1 = {city_name, country_name, venue_name} & R2 = {venue_name, match_id, home_team, visiting_team, match_date, toss_winner, match_winner, toss_decision, outcome_type, manofmatch, win_margin}
- Now R2 is in BCNF but R1 is not (since city_name → country_name still violates BCNF)

We will take city_name → country_name to decompose R1:

- Decompose R1 into R3 = {country_name, city_name} & R4 = {city_name, venue_name}

Now all the new decomposed relations of Match (R2, R3 & R4) are in BCNF

We will not proceed with the further steps of 3NF decomposition as the functional dependencies are already preserved.
Therefore, new decomposed relations/tables of Match (which are in 3NF) are:

- R2 = {venue_name, match_id, home_team, visiting_team, match_date, toss_winner, match_winner, toss_decision, outcome_type, manofmatch, win_margin}
- R3 = {country_name, city_name} (Location)
- R4 = {city_name, venue_name} (Stadium)

Table: Team
Note: Assuming each team will have a unique name

FD = {team_id → team_name, year_started, base_city, active_status, Team_user_thoughts team_name → team_id }

This table is in both 3NF and BCNF as the LHS of both the functional dependencies is a super key

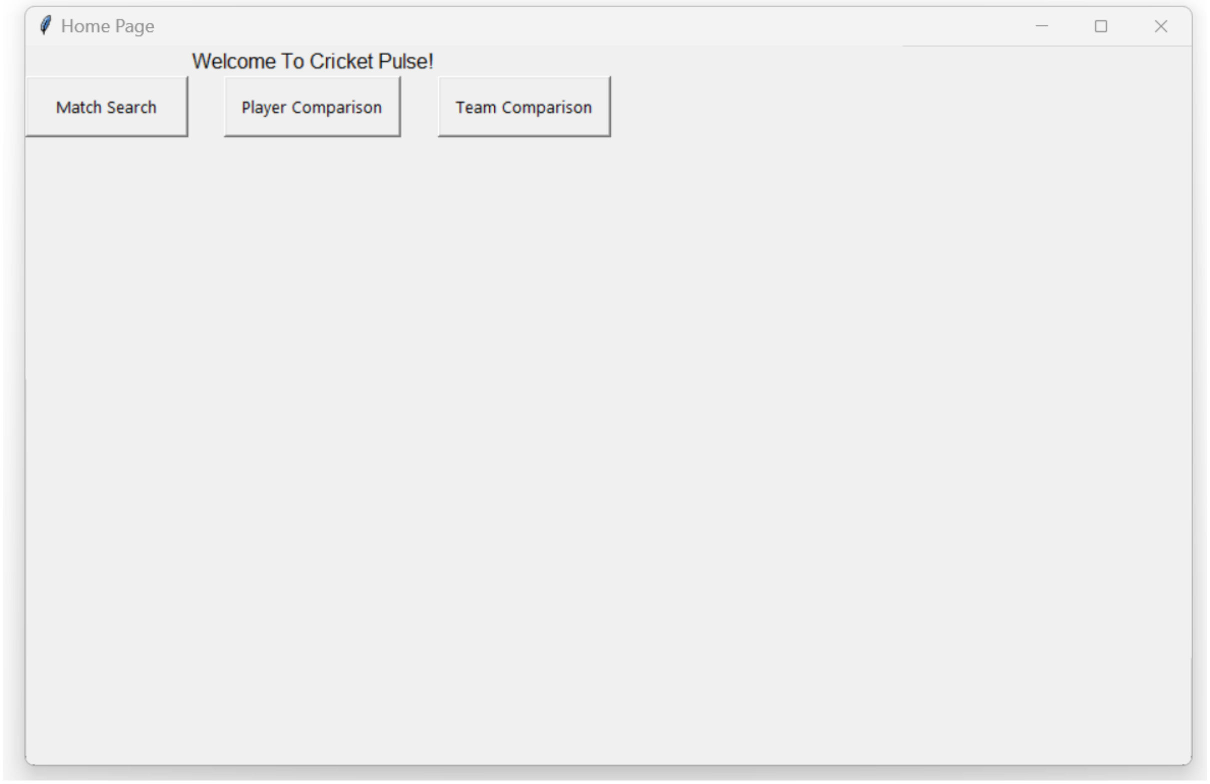
Table: Player
FD = { player_id → player_name, dob, batting_hand, bowling_skill, country_name, Player_user_thoughts}
This table is in both 3NF and BCNF as the LHS of the only functional dependency is a super key

Table: Player Match
FD = { match_id, player_id → role_desc, player_team}
This table is in both 3NF and BCNF as the LHS of the only functional dependency is a super key

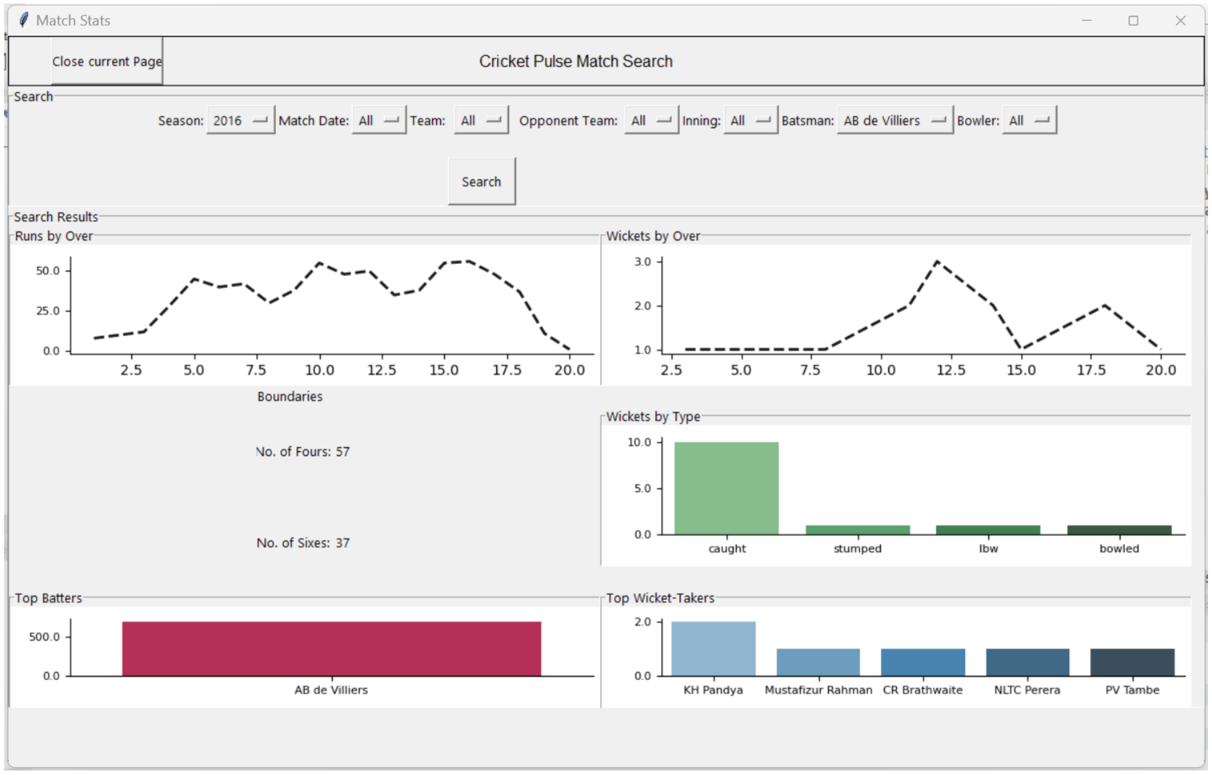
II.5 Implementation

A video walkthrough of the application is attached

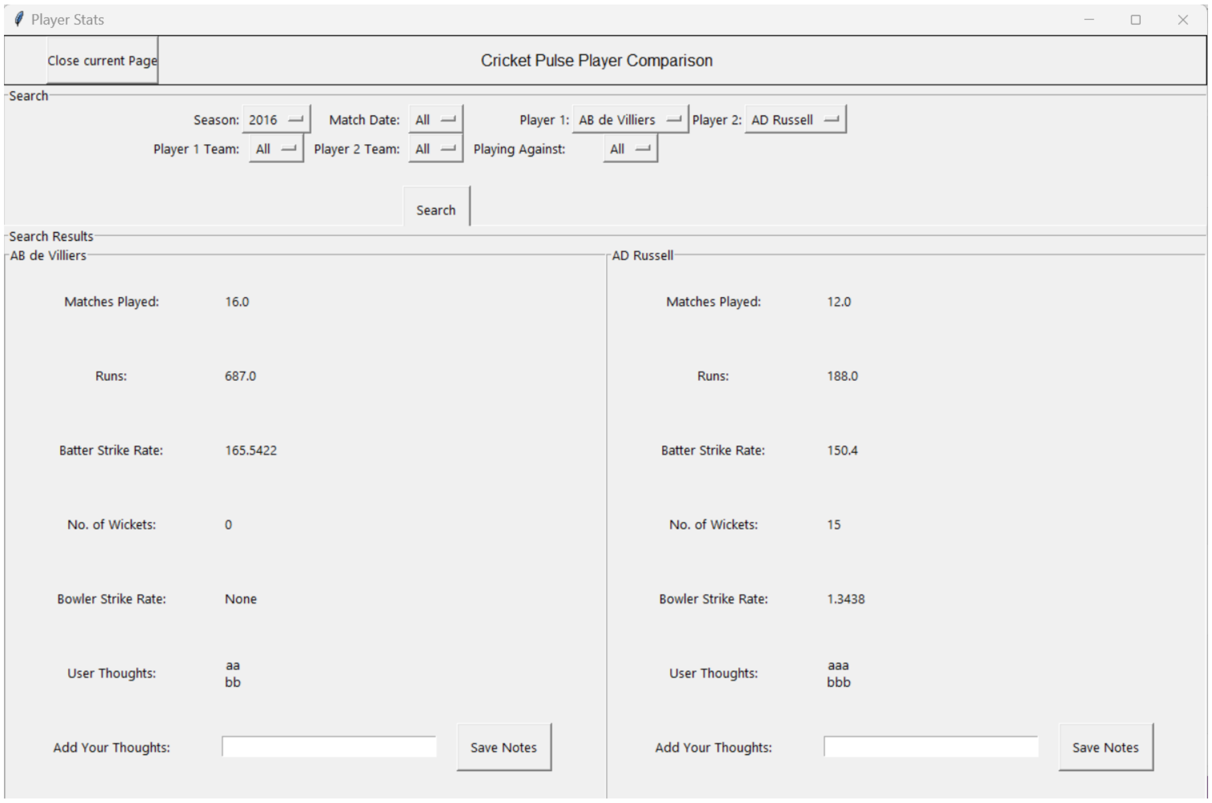
Home Page



Match Stats Page (Note: All wickets sections are bowlers based and all runs sections (rest) are batters based)

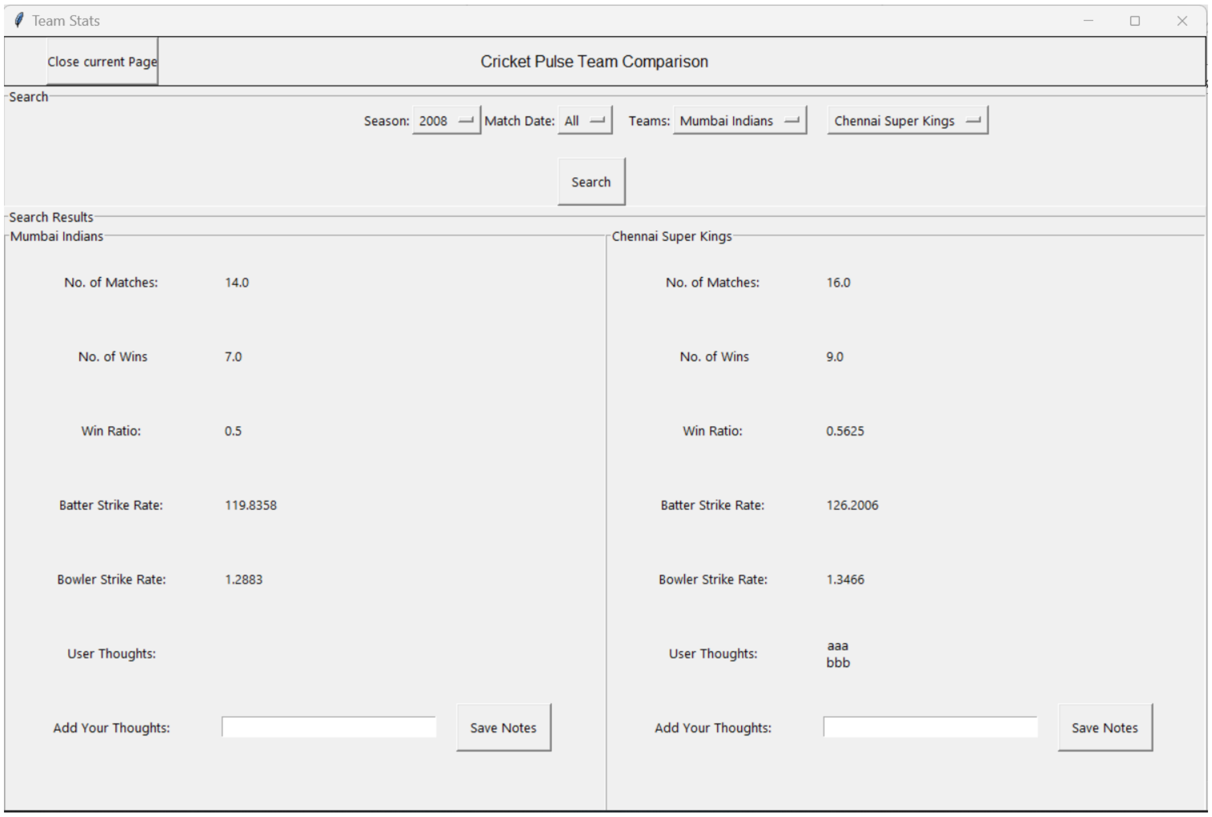


Player Comparison Page



	AB de Villiers		AD Russell
Matches Played:	16.0	Matches Played:	12.0
Runs:	687.0	Runs:	188.0
Batter Strike Rate:	165.5422	Batter Strike Rate:	150.4
No. of Wickets:	0	No. of Wickets:	15
Bowler Strike Rate:	None	Bowler Strike Rate:	1.3438
User Thoughts:	aa bb	User Thoughts:	aaa bbb
Add Your Thoughts:		Add Your Thoughts:	

Team Comparison Page



Cricket Pulse Team Comparison

Search

Season: 2008 Match Date: All Teams: Mumbai Indians Chennai Super Kings

Search Results

Mumbai Indians		Chennai Super Kings	
No. of Matches:	14.0	No. of Matches:	16.0
No. of Wins	7.0	No. of Wins	9.0
Win Ratio:	0.5	Win Ratio:	0.5625
Batter Strike Rate:	119.8358	Batter Strike Rate:	126.2006
Bowler Strike Rate:	1.2883	Bowler Strike Rate:	1.3466
User Thoughts:		User Thoughts:	aaa bbb
Add Your Thoughts:		Add Your Thoughts:	

II.6 Evaluation

We plan to test the application for the functionality (UI) and the data. Below are the test cases for both.

UI

Test Case	Status
Clicking on buttons on the home page takes you to the respective pages (Page 1, Page 2, Page 3)	Passed
Navigate back to the home page from each of the three pages.	Passed
Validate the functionality of all the filters on Page 1 by ensuring they contain relevant values and filter the data in the charts/sections whenever the filters are modified or updated.	Passed
Validate the functionality of all the filters on Page 2 by ensuring they contain relevant values and filter the data in the charts/sections whenever the filters are modified or updated.	Passed
Validate the functionality of all the filters on Page 3 by ensuring they contain relevant values and filter the data in the charts/sections whenever the filters are modified or updated.	Passed
Verify if the notes added by users for players are updated in the database	Passed
Verify if the notes added by users for team are updated in the database	Passed

Data

Test Case	Status
Verify if the data for runs scored by over is matching with the raw data	Passed
Verify if the data for # boundaries by type is matching with the raw data	Passed
Verify if the data for Top 5 batters is matching with the raw data	Passed
Verify if the data for # wickets by over is matching with the raw data	Passed
Verify if the data for # wickets by type is matching with the raw data	Passed
Verify if the data for Top 5 wicket takers is matching with the raw data	Passed
Verify if player's # matches, # runs, batter strike rate is matching with the raw data	Passed
Verify if player's # wickets is matching with the raw data	Passed
Verify if player's bowling strike rate is matching with the raw data	Passed
Verify if team's # matches, # wins , win ratio is matching with the raw data	Passed
Verify if team's batting and bowling strike rate is matching with the raw data	Passed

Note: For all of the above data test cases, we have matched the UI data with the raw data in cvs files.

III. Conclusion

During the creation of our app, we learned a lot about both the process itself and the way that real-world data can impact your plans for implementation. The dataset that we used for our project makes some very common operations difficult, such as finding a team's roster for a given season, or figuring out what team a player played for each year of his career. The way that our data was collected and represented had a tangible effect on what our app's functionality was – we had to drop the idea of displaying a team's roster across seasons due to the complexity of the SQL required.

That being said, our SQL queries remained very complex due to the number of filters involved in one individual search. Given 3 different pages, with 5+ filters for queries, some of which were optional for a query, the SQL required to make sure those filters were translated into the correct query was very complex. It was eye-opening to see how useful and flexible SQL can be when done correctly.

We also learned a lot about the timeline of a project's construction. Due to the condensed nature of summer courses, coordinating our efforts and making sure everyone was always contributing in one way or another was vital to ensuring that our project was finished in time for our presentation. We were forced to divide up the labor of our project efficiently and make decisions quickly about what we could and could not complete within the timeframe.

Our project also forced all of us to hone our technical skills in one way or another. Sriharsha and Harsh had never created a GUI prior to this project, and Andrew had never worked with Python prior to this. Each of us had to learn and close the gaps in our knowledge while creating this project in order for it to come together correctly. It also encouraged us to cooperate and help cover each other's blind spots.