1. INTRODUCTION

In the realm of modern sports analytics, the integration of technology has revolutionized the way teams make decisions. "Cricket Data Analysis: Player Performance and Strategy Insights" is a project aimed at providing teams with valuable insights through the analysis of player performance and strategic recommendations. This system leverages machine learning algorithms to analyze various cricketing parameters and provide data-driven suggestions, empowering teams to make informed choices for improved performance and success on the field.

1.1 System Definition

The "Cricket Data Analysis" project is a sophisticated system designed to analyze Indian Premier League (IPL) cricket data spanning from 2008 to 2020. It incorporates a user interface developed in Python using the Tkinter library for the frontend, complemented by MongoDB serving as the robust backend database. This project caters to two distinct user roles: User and Admin.

1.2 Project Description

Admin Features:

User Management: The admin has the authority to create, update, read, and delete user accounts.

User Features:

Upon successful login, the user gains access to the following cricket analysis:

- Head to Head Analysis: Displays the total number of wins and losses among two teams.
- Lucky Venues Analysis: Shows the number of wins at different venues for particular teams.
- Top 10 Run Scorers of All Time: Provides insights into the top 10 run-scorers in IPL history.
- Highest Wicket-Taker Analysis: Highlights the highest wicket-taker in IPL history.
- Toss Decision Across Seasons: Examines the impact of toss decisions over different IPL seasons.

2. SYSTEM STUDY

In this section, we delve into the comparison between traditional cricket coaching methods and the proposed "Cricket Data Analysis: Player Performance and Strategy Insights" system. By highlighting the limitations of conventional coaching techniques and the advantages of the new data-driven approach, we see how technology can revolutionize cricket coaching for the better.

2.1 Existing System

Before the implementation of this project, there was a lack of a centralized system for efficiently analyzing and visualizing IPL data. Users had to rely on manual processes or third-party tools, often lacking integration and user-friendly interfaces.

2.2 Proposed System

The proposed system revolutionizes the analysis of IPL data, providing users with a dedicated and streamlined platform. The frontend, developed in Python using Tkinter, offers a seamless and interactive interface. MongoDB, as the backend, ensures reliable data storage and retrieval, with scalability to accommodate future growth.

2.3 Data Flow Diagram(level 0 and level 1)

The Data Flow Diagrams (DFDs) are used for structure analysis and design. DFDs show the flow of data from external entities into the system. DFDs also show how the data moves and are transformed from one process to another, as well as its logical storage. The following symbols are used within DFDs.

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modelling its process aspects. A DFD is often used as a preliminary step to create an overview of the system, which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).

A DFD shows what kind of information will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of process or information about whether processes will operate in sequence or in parallel.

PHYSICAL VS. LOGICAL DFD

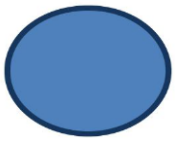
A logical DFD captures the data flows that are necessary for a system to operate. It describes the processes that are undertaken, the data required and produced by each process, and the stores needed to hold the data. On the other hand, a physical DFD shows how the system is implemented, either now (Current Physical DFD), or how the designer intends it to be in the future (Required Physical DFD).

Thus, a Physical DFD may be used to describe the set of data items that appear on each piece of paper that move around an office, and the fact that a set of pieces of paper are stored together in a filing cabinet. It is quite possible that a Physical DFD will include references to data that are duplicated, or redundant, and that the data stores, if implemented as a set of database tables, would constitute an un-normalized (or de-normalized) relational database. In contrast, a Logical DFD attempts to capture the data flow aspects of a system in a form that has neither redundancy nor duplication.

DATA FLOW SYMBOLS AND THEIR MEANINGS: -

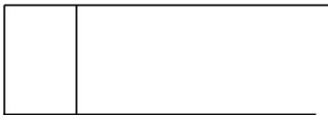
An entity: A source of data or a destination for data.

Source/Sink: Represented by rectangles in the diagram. Sources and Sinks are external entities which are sources or destinations of data, respectively.

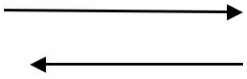


Process: Represented by circles in the diagram. Processes are responsible for manipulating the data. They take data as input and output an altered version of the data.

Data Store: Represented by a segmented rectangle with an open end on the right. Data Stores are both electronic and physical locations of data. Examples include databases, directories, files, and even filing cabinets and stacks of paper.

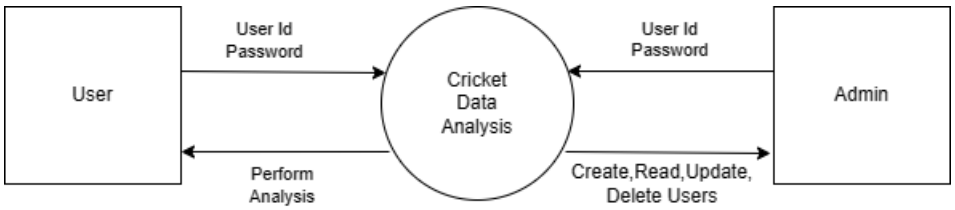


Data Flow: Represented by a unidirectional arrow. Data Flows show how data is moved through the System. Data Flows are labeled with a description of the data that is being passed through it.



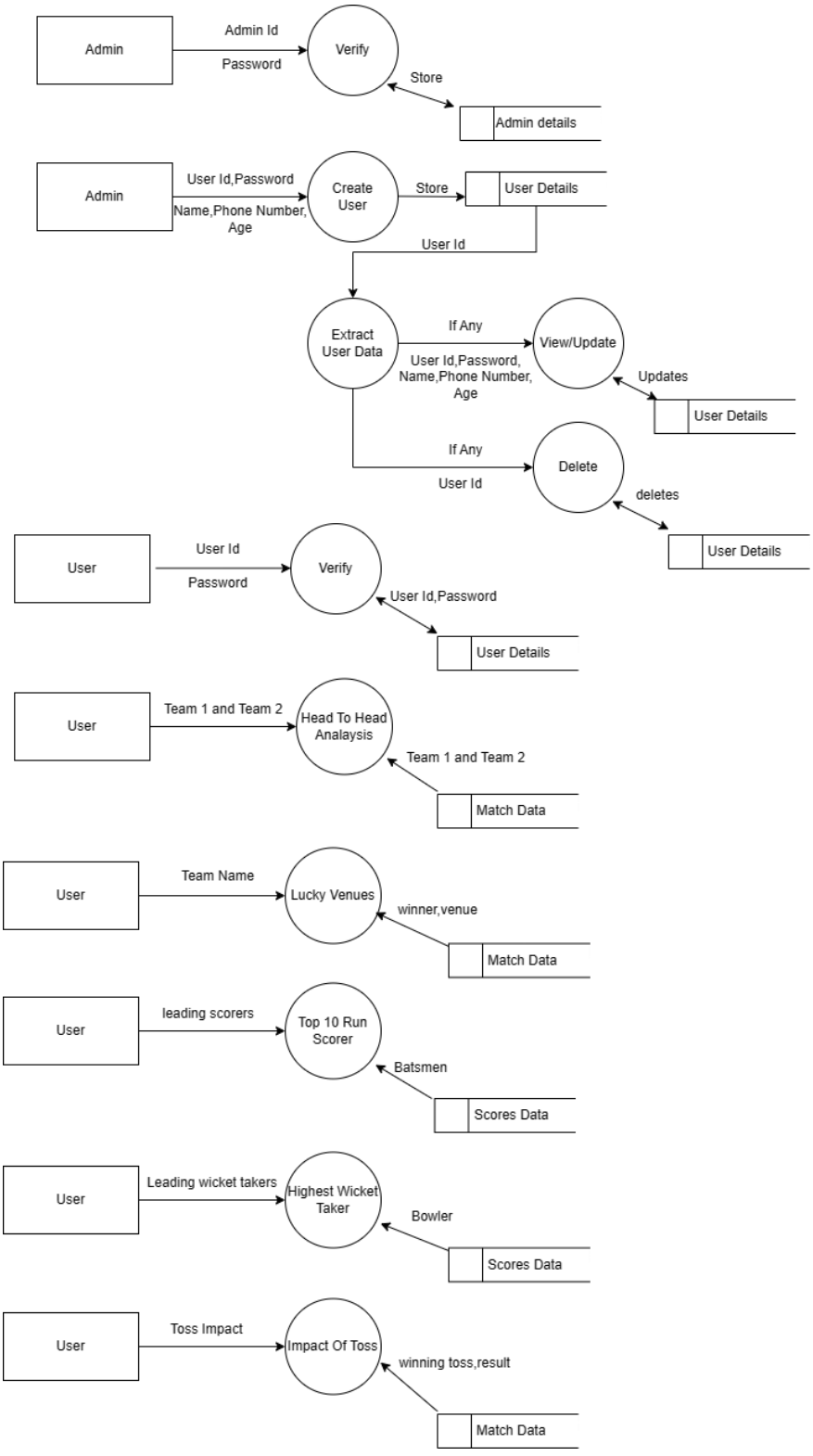
A level-0 DFD is the most basic form of DFD. It aims to show how the entire system works at a glance. There is only one process in the system and all the data flows either into or out of this process. Level-0 DFD's demonstrates the interactions

When drawing Level-0 DFD's, we must first identify the process, all the external entities and all the data flows. We must also state any assumptions we make about the system. It is advised that we draw the process in the middle of the page. We then draw our external entities in the corners and finally connect our entities to our process with the data flows between the process and external entities. They do not contain Data Stores.



Level 1 DFD's:

Level 1 DFD's aim is to give an overview of the full system. They look at the system in more detail. Major processes are broken down into sub-processes. Level 1 DFD's also identifies data stores that are used by the major processes.



3. SYSTEM CONFIGURATION

In the design phase, the architecture of "Cricket Data Analysis: Player Performance and Strategy Insights" is meticulously established. This phase, rooted in the requirements document from the earlier stage, transforms requirements into a concrete architecture. The architecture delineates the system's components, interfaces, and behaviors, captured in the design document.

3.1 Hardware Configuration

Processor	Intel(r)core i5-4210u CPU @ 1.70ghz
Ram	16 Gb
ClockSpeed	1.80 ghz
Memory	4gb

3.2 Software Configuration

Front-End	Python
Back-End	Mongodb
Documentation	MS Word
OS	Windows 11

4. DETAILS OF SOFTWARE

Diving into the software architecture of "Cricket Data Analysis: Player Performance and Strategy Insights," this section provides an in-depth look at both the frontend and backend components. From the intuitive user interface to the robust backend processing, each element is carefully designed to enhance the coaching experience for users.

4.1 Overview of Front End

The frontend of the project utilizes the Tkinter library in Python. Tkinter, a powerful GUI toolkit, facilitates the creation of a visually appealing and user-friendly interface. It seamlessly interacts with the backend to retrieve and display IPL data for various analyses.

The frontend of the "Cricket Data Analysis" project is developed using the Tkinter library in Python. Tkinter provides a robust and user-friendly graphical user interface (GUI) toolkit, allowing for the creation of interactive windows, buttons, and visual elements. This choice ensures a seamless and engaging user experience. The frontend is responsible for presenting analysis options to users, capturing their input, and communicating with the backend to retrieve and display relevant IPL data.

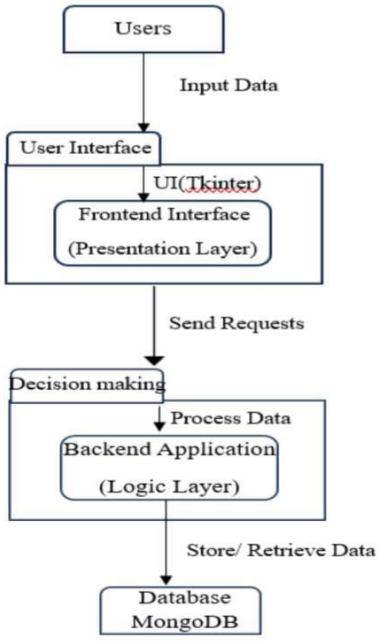
4.2 Overview of Back End

MongoDB serves as the backend database for the project. MongoDB is a NoSQL database that offers flexibility in handling unstructured data, making it ideal for storing and managing IPL cricket data. The backend manages user authentication, ensuring secure access to the system. It processes user requests, retrieves data from the database, and sends the necessary information back to the frontend for display. MongoDB's scalability ensures efficient storage and retrieval of data, accommodating the growing volume of IPL statistics over the years.

5. SYSTEM DESIGN

5.1 Architectural Design

- Creating and maintaining architectural diagrams to provide accurate and valuable content is not easy. Most of the time we create either too much, too little or irrelevant documentation because we fail to identify the proper beneficiaries and their real needs.
- One of the biggest mistakes is to create detailed architectural diagrams for parts of the system with high volatility. It is a burden to manually maintain them unless they are automatically generated.
- In practice, most stakeholders are not interested in detailed diagrams, but rather in one or two high-level diagrams which reflect the modularity and boundaries of the system. Beyond these, for a deeper understanding, the code should be the source of truth, which in most of the cases only developers are interested in.
- To find the appropriate amount of quantity and quality of architectural diagrams, brainstorm and agree with the team what is really useful for them, whatever that means! Do not try to create diagrams for things that are self-explanatory in the source code or for the sake of any comprehensive architectural methodology.
- The main purpose of architectural diagrams should be to facilitate collaboration, to increase communication, and to provide vision and guidance.



5.2 Input Design

login.py:

```
class LoginWindow:
    def __init__(self, root, user_type):
        self.root = root
        self.user_type = user_type
        self.init_ui()

    def init_ui(self):
        self.login_window = tk.Toplevel(self.root)
        self.login_window.title("Cricket Data Analysis - Login")

        window_width = 300
        window_height = 150
        screen_width = self.root.winfo_screenwidth()
        screen_height = self.root.winfo_screenheight()
        x = (screen_width - window_width) // 2
        y = (screen_height - window_height) // 2
        self.login_window.geometry(f"{window_width}x{window_height}+{x}+{y}")

        self.create_widgets()

    def create_widgets(self):
        username_label = tk.Label(self.login_window, text="User ID:")
        username_label.grid(row=0, column=0, padx=10, pady=5, sticky=tk.E)

        self.username_entry = tk.Entry(self.login_window)
        self.username_entry.grid(row=0, column=1, padx=10, pady=5)

        password_label = tk.Label(self.login_window, text="Password:")
        password_label.grid(row=1, column=0, padx=10, pady=5, sticky=tk.E)

        self.password_entry = tk.Entry(self.login_window, show="*")
        self.password_entry.grid(row=1, column=1, padx=10, pady=5)

        login_button = tk.Button(self.login_window, text="Login",
command=self.perform_login,
                                 width=15, height=2, relief=tk.RAISED, bg="#4CAF50",
fg="white", font=("Helvetica", 12, "bold"))
        login_button.grid(row=2, column=1, pady=10)
if __name__ == "__main__":
    db = Database()

    root = tk.Tk()
    root.title("Cricket Data Analysis")

    try:
        image_path = r"D:\Photos\Index.jpg"  # Replace with your image file path
```

```
            background_image = Image.open(image_path)
            background_photo = ImageTk.PhotoImage(background_image)
        except FileNotFoundError:
            messagebox.showerror("Image Error", f"Image file not found at: {image_path}")
            root.destroy()
        else:
            screen_width = root.winfo_screenwidth()
            screen_height = root.winfo_screenheight()

            root.geometry(f"{screen_width}x{screen_height}+0+0")
            root.attributes("-fullscreen", True)

            background_label = tk.Label(root, image=background_photo)
            background_label.place(relwidth=1, relheight=1)

            button_width = 15
            button_height = 2
            button_relx = 0.5
            user_button = tk.Button(root, text="User", command=open_user_login,
                                    width=button_width, height=button_height, relief=tk.RAISED,
bg="#3366CC", fg="white",
                                    font=("Helvetica", 12, "bold"), bd=3, highlightthickness=0)
            admin_button = tk.Button(root, text="Admin", command=open_admin_login,
                                     width=button_width, height=button_height, relief=tk.RAISED,
bg="#FF6347", fg="white",
                                     font=("Helvetica", 12, "bold"), bd=3, highlightthickness=0)

            user_button.place(relx=button_relx, rely=0.49, anchor="center")
            admin_button.place(relx=button_relx, rely=0.72, anchor="center")

            root.mainloop()

        db.close_connection()
    def open_login_page(user_type, user_radio_var, admin_radio_var):
        login_window = tk.Toplevel(root)
        login_window.title("Cricket Data Analysis - Login")

        # Calculate the window position for centering
        window_width = 300
        window_height = 150
        screen_width = root.winfo_screenwidth()
        screen_height = root.winfo_screenheight()
        x = (screen_width - window_width) // 2
        y = (screen_height - window_height) // 2

        # Set the window dimensions and position
        login_window.geometry(f"{window_width}x{window_height}+{x}+{y}")

        # Create widgets for login page
        username_label = tk.Label(login_window, text="User Id:")
```

```
        username_label.grid(row=0, column=0, padx=10, pady=5, sticky=tk.E)

        username_entry = tk.Entry(login_window)
        username_entry.grid(row=0, column=1, padx=10, pady=5)

        password_label = tk.Label(login_window, text="Password:")
        password_label.grid(row=1, column=0, padx=10, pady=5, sticky=tk.E)

        password_entry = tk.Entry(login_window, show="*")
        password_entry.grid(row=1, column=1, padx=10, pady=5)

        login_button = tk.Button(login_window, text="Login", command=lambda:
login(user_radio_var, admin_radio_var, username_entry, password_entry),
                                 width=15, height=2, relief=tk.RAISED, bg="#4CAF50", fg="white",
font=("Helvetica", 12, "bold"))
        login_button.grid(row=2, column=1, pady=10)
    def open_user_login():
        update_radio_vars("user", user_radio_var, admin_radio_var)
        open_login_page("User", user_radio_var, admin_radio_var)

    # Function to open login page for Admin
    def open_admin_login():
        update_radio_vars("admin", user_radio_var, admin_radio_var)
        open_login_page("Admin", user_radio_var, admin_radio_var)

    # Load the background image using PIL
    try:
        image_path = r"D:\Photos\Index.jpg"  # Replace with your image file path
        background_image = Image.open(image_path)
        background_photo = ImageTk.PhotoImage(background_image)
    except FileNotFoundError:
        messagebox.showerror("Image Error", f"Image file not found at: {image_path}")
        root.destroy()
    else:
        # Set window size and position
        screen_width = root.winfo_screenwidth()
        screen_height = root.winfo_screenheight()

        root.geometry(f"{screen_width}x{screen_height}+0+0")

        # Set window to full screen without borders
        root.attributes("-fullscreen", True)

        # Create a label to display the background image
        background_label = tk.Label(root, image=background_photo)
        background_label.place(relwidth=1, relheight=1)

        # Customize button appearance and size
        button_width = 15
        button_height = 2
```

```
        button_relx = 0.5
        user_button = tk.Button(root, text="User", command=open_user_login,
width=button_width, height=button_height, relief=tk.RAISED,
                                bg="#3366CC", fg="white", font=("Helvetica", 12, "bold"), bd=3,
highlightthickness=0)
        admin_button = tk.Button(root, text="Admin", command=open_admin_login,
width=button_width, height=button_height, relief=tk.RAISED,
                                 bg="#FF6347", fg="white", font=("Helvetica", 12, "bold"), bd=3,
highlightthickness=0)

        # Place buttons in the center of the window with some spacing
        user_button.place(relx=button_relx, rely=0.49, anchor="center")
        admin_button.place(relx=button_relx, rely=0.72, anchor="center")

        # Run the main loop
        root.mainloop()
```

crud.py:

```
    class AdminPage:
        def __init__(self, root):
            self.root = root
            self.root.title("Admin Page")

            # MongoDB connection
            self.client = MongoClient('mongodb://localhost:27017/')
            self.db = self.client['cricket']
            self.collection = self.db['users']

            # Background Image
            image_path = 'D:\\Photos\\Index.jpg'
            if not os.path.isfile(image_path):
                messagebox.showerror("Image Error", f"Image file not found at: {image_path}")
            else:
                self.img = Image.open(image_path)
                self.img = ImageTk.PhotoImage(self.img)

                background_label = tk.Label(self.root, image=self.img)
                background_label.place(relwidth=1, relheight=1)

            self.frame = tk.Frame(self.root)
            self.frame.pack(padx=20, pady=20)

            self.label = tk.Label(self.frame, text="Admin Page", font=("Helvetica", 18, "bold"))
            self.label.grid(row=0, column=0, columnspan=2, pady=10)

            # Buttons for different modules
            create_users_button = tk.Button(self.frame, text="Create Users",
command=self.create_users)
```

```
        create_users_button.grid(row=1, column=0, padx=10, pady=10)

        view_users_button = tk.Button(self.frame, text="View/Update Users",
command=self.view_users)
        view_users_button.grid(row=1, column=1, padx=10, pady=10)

        delete_users_button = tk.Button(self.frame, text="Delete Users",
command=self.delete_users)
        delete_users_button.grid(row=2, column=0, columnspan=2, padx=10, pady=10)

        # User database (in-memory representation)
        self.users = []




    def add_user_to_mongodb(self, user_id, password, name, phone, age,
create_window):
        user_info = {
            "User ID": user_id,
            "Password": password,
            "Name": name,
            "Phone Number": phone,
            "Age": age
        }

        try:
            # Insert user information into MongoDB
            result = self.collection.insert_one(user_info)

            if result.inserted_id:
                messagebox.showinfo("User Created", "User created successfully and added to
MongoDB!")
                create_window.destroy()  # Close the create user window
            else:
                messagebox.showerror("Error", "Failed to add user to MongoDB.")
        except Exception as e:
            messagebox.showerror("Error", f"An error occurred: {str(e)}")

    def create_users(self):
        create_window = tk.Toplevel(self.root)
        create_window.title("Create User")

        # Create entry widgets for user input
        user_id_label = tk.Label(create_window, text="User ID:")
        user_id_label.grid(row=0, column=0, padx=10, pady=5, sticky=tk.E)
        user_id_entry = tk.Entry(create_window)
        user_id_entry.grid(row=0, column=1, padx=10, pady=5)

        password_label = tk.Label(create_window, text="Password:")
```

```
        password_label.grid(row=1, column=0, padx=10, pady=5, sticky=tk.E)
        password_entry = tk.Entry(create_window, show="*")
        password_entry.grid(row=1, column=1, padx=10, pady=5)

        name_label = tk.Label(create_window, text="Name:")
        name_label.grid(row=2, column=0, padx=10, pady=5, sticky=tk.E)
        name_entry = tk.Entry(create_window)
        name_entry.grid(row=2, column=1, padx=10, pady=5)

        phone_label = tk.Label(create_window, text="Phone Number:")
        phone_label.grid(row=3, column=0, padx=10, pady=5, sticky=tk.E)
        phone_entry = tk.Entry(create_window)
        phone_entry.grid(row=3, column=1, padx=10, pady=5)

        age_label = tk.Label(create_window, text="Age:")
        age_label.grid(row=4, column=0, padx=10, pady=5, sticky=tk.E)
        age_entry = tk.Entry(create_window)
        age_entry.grid(row=4, column=1, padx=10, pady=5)

        submit_button = tk.Button(create_window, text="Submit", command=lambda:
self.add_user_to_mongodb(
            user_id_entry.get(), password_entry.get(), name_entry.get(), phone_entry.get(),
age_entry.get(), create_window))
        submit_button.grid(row=5, column=1, pady=10)

    def view_users(self):
        view_window = tk.Toplevel(self.root)
        view_window.title("View/Update Users")

        # Fetch all users from MongoDB
        users_from_mongo = self.collection.find()

        for i, user in enumerate(users_from_mongo):
            user_info_label = tk.Label(view_window, text=f"User {i + 1}:")
            user_info_label.grid(row=i * 2, column=0, padx=10, pady=5)

            # Display user information
            user_id_label = tk.Label(view_window, text=f"User ID: {user['User ID']}")
            user_id_label.grid(row=i * 2 + 1, column=0, padx=10, pady=5)

            password_label = tk.Label(view_window, text=f"Password: {user['Password']}")
            password_label.grid(row=i * 2 + 1, column=1, padx=10, pady=5)

            name_label = tk.Label(view_window, text=f"Name: {user['Name']}")
            name_label.grid(row=i * 2 + 1, column=2, padx=10, pady=5)

            phone_label = tk.Label(view_window, text=f"Phone Number: {user['Phone
Number']}")
            phone_label.grid(row=i * 2 + 1, column=3, padx=10, pady=5)
```

```
            age_label = tk.Label(view_window, text=f"Age: {user['Age']}")
            age_label.grid(row=i * 2 + 1, column=4, padx=10, pady=5)

            # Add an "Update" button for each user
            update_button = tk.Button(view_window, text="Update", command=lambda
u=user: self.open_update_window(u))
            update_button.grid(row=i * 2 + 1, column=5, padx=10, pady=5)

            # Separator below each user
            separator = tk.Label(view_window, text="------------------------")
            separator.grid(row=i * 2 + 2, column=0, columnspan=6, padx=10, pady=5)

        if users_from_mongo.count() == 0:
            messagebox.showinfo("No Users", "No users found in MongoDB.")

    def open_update_window(self, user):
        update_window = tk.Toplevel(self.root)
        update_window.title("Update User Info")

        # Center the update window on the screen
        window_width = 300
        window_height = 200
        screen_width = update_window.winfo_screenwidth()
        screen_height = update_window.winfo_screenheight()
        x = (screen_width - window_width) // 2
        y = (screen_height - window_height) // 2

        update_window.geometry(f"{window_width}x{window_height}+{x}+{y}")

        # Bring the update window to the front
        update_window.lift(self.root)
        update_window.attributes('-topmost', True)
        update_window.attributes('-topmost', False)  # Allow other windows to be on top

        # Populate the entry fields with the current user information
        password_entry = tk.Entry(update_window, show="*")
        password_entry.insert(0, user["Password"])
        password_entry.grid(row=1, column=1, padx=10, pady=5)

        name_entry = tk.Entry(update_window)
        name_entry.insert(0, user["Name"])
        name_entry.grid(row=2, column=1, padx=10, pady=5)

        phone_entry = tk.Entry(update_window)
        phone_entry.insert(0, user["Phone Number"])
        phone_entry.grid(row=3, column=1, padx=10, pady=5)

        age_entry = tk.Entry(update_window)
        age_entry.insert(0, user["Age"])
        age_entry.grid(row=4, column=1, padx=10, pady=5)
```

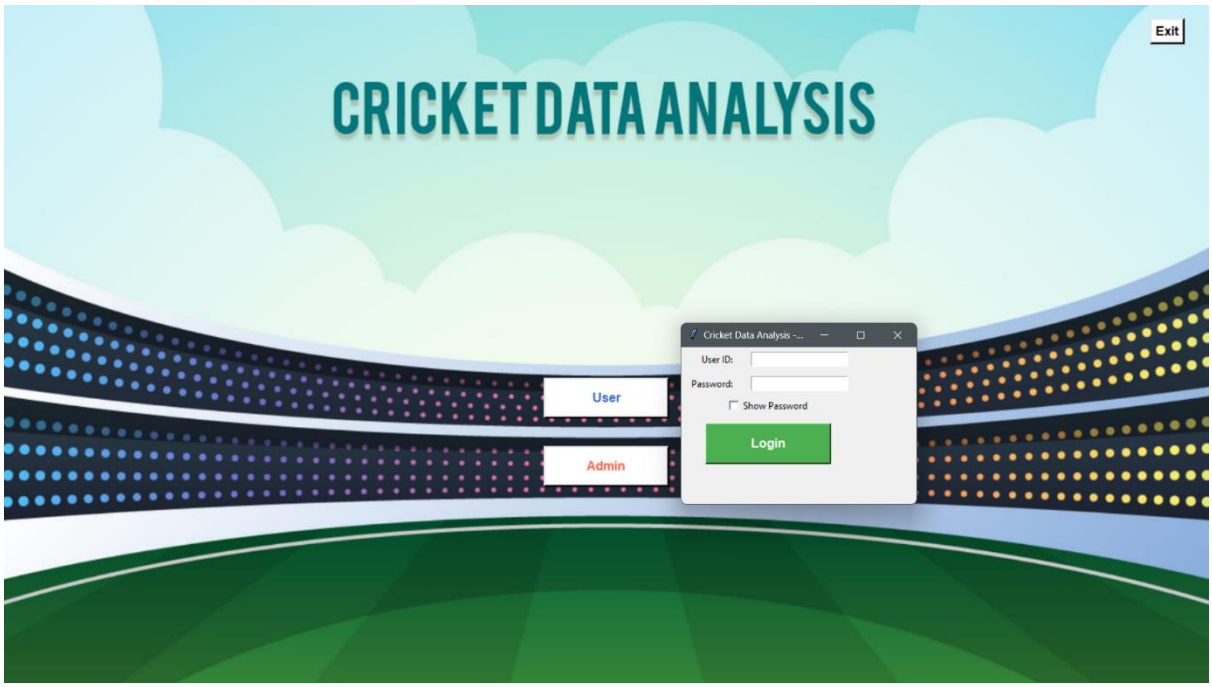
```
        update_button = tk.Button(update_window, text="Update",
                                  command=lambda: self.update_user_info(user, password_entry.get(),
name_entry.get(),
                                                                        phone_entry.get(), age_entry.get(),
update_window))
        update_button.grid(row=5, column=1, pady=10)

        messagebox.showinfo("Update", "User information update initiated. Click 'Update' to
confirm changes.")
```

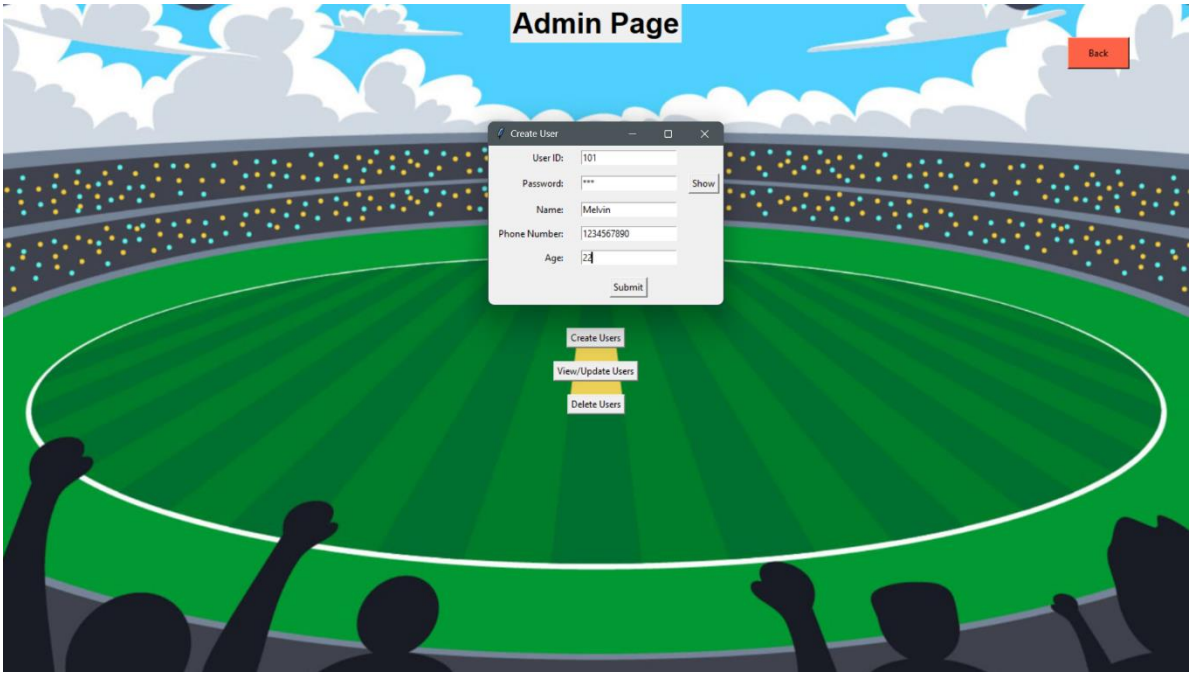
analysis.py:

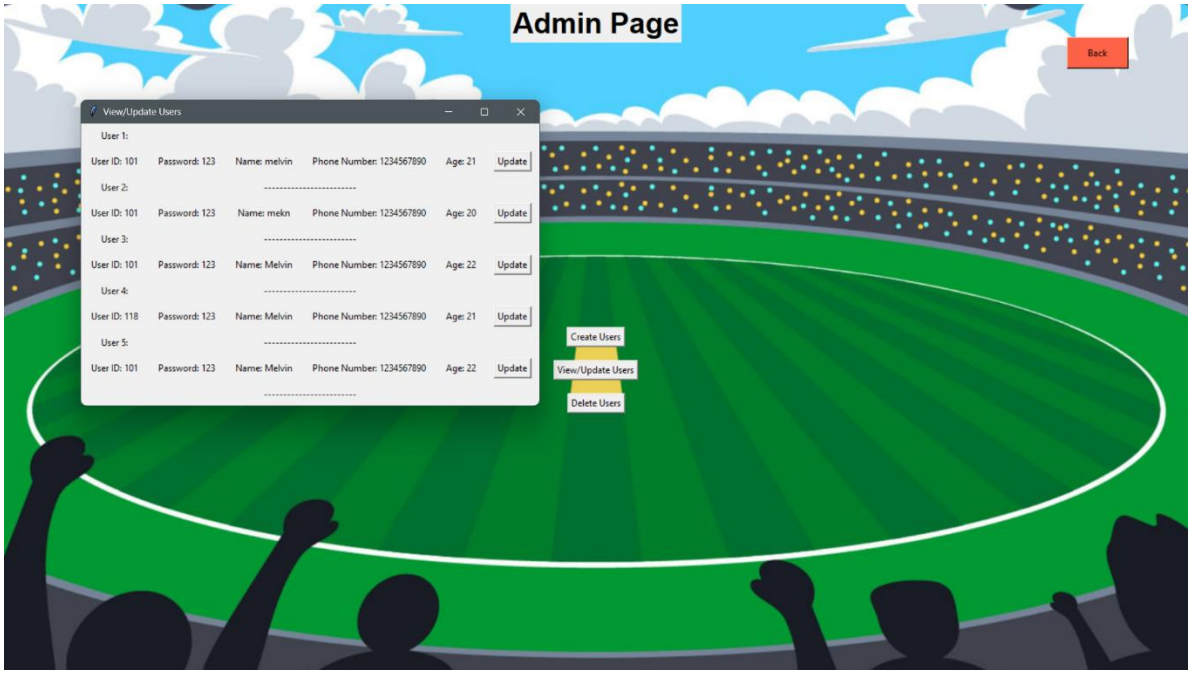
```
    def create_section(self, frame, section):
        # Add title label
        title_label = ttk.Label(frame, text=section["title"], font=("Helvetica", 16, "bold"))
        title_label.pack(pady=(10, 0))

        # Add description label
        description_label = ttk.Label(frame, text=section["description"], wraplength=300,
padding=(10, 10))
        description_label.pack()

        # Get screen dimensions
        screen_width = self.winfo_screenwidth()
        screen_height = self.winfo_screenheight()

        # Add Canvas for the background image
        canvas = tk.Canvas(frame, width=screen_width, height=screen_height)
        canvas.pack(fill=tk.BOTH, expand=True)  # Fill the entire frame

        # Add image as background
        if "image_path" in section:
            image = Image.open(section["image_path"])
            # Resize the image to fit the screen
            image = image.resize((screen_width, screen_height), Image.LANCZOS)
            section["photo_image"] = ImageTk.PhotoImage(image)
            canvas.create_image(0, 0, anchor=tk.NW, image=section["photo_image"])

        # Create a frame for buttons
        button_frame = ttk.Frame(canvas)
        button_frame.pack(pady=10)

        # Add button to lead to a specific page
        button = ttk.Button(button_frame, text="Go to Page", command=lambda:
self.show_specific_page(section["title"]))
        button.pack()
```

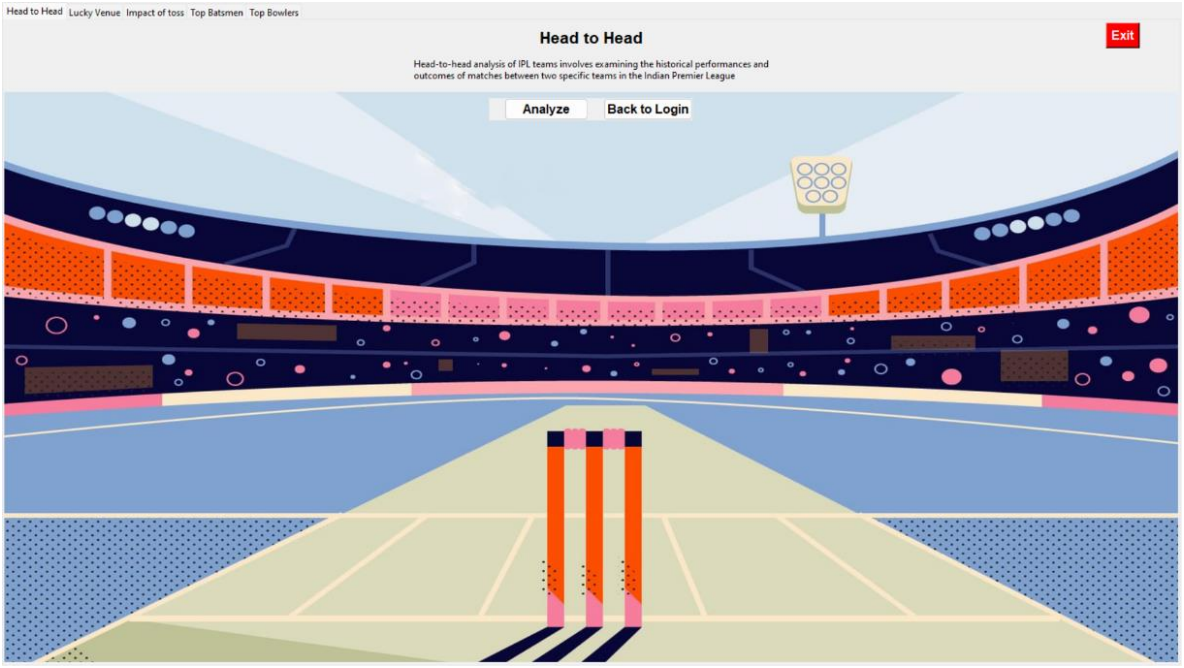
```
    def show_specific_page(self, section_title):
        if section_title == "Head to Head":
            # Launch HeadToHead.py as a separate process
            subprocess.Popen(["python", "HeadToHead.py"])
        elif section_title == "Impact of toss":
            subprocess.Popen(["Python", "ImpactOfToss.py"])
        elif section_title == "Lucky Venue":
            subprocess.Popen(["Python", "LuckyVenues.py"])
        elif section_title == "Top Batsmen":
            subprocess.Popen(["Python", "TopBatsmen.py"])
        elif section_title == "Top Bowlers":
            subprocess.Popen(["Python", "TopBowler.py"])
        else:
            print(f"Button clicked for {section_title}")

if __name__ == "__main__":
    app = AnalysisPage()
    app.mainloop()
```

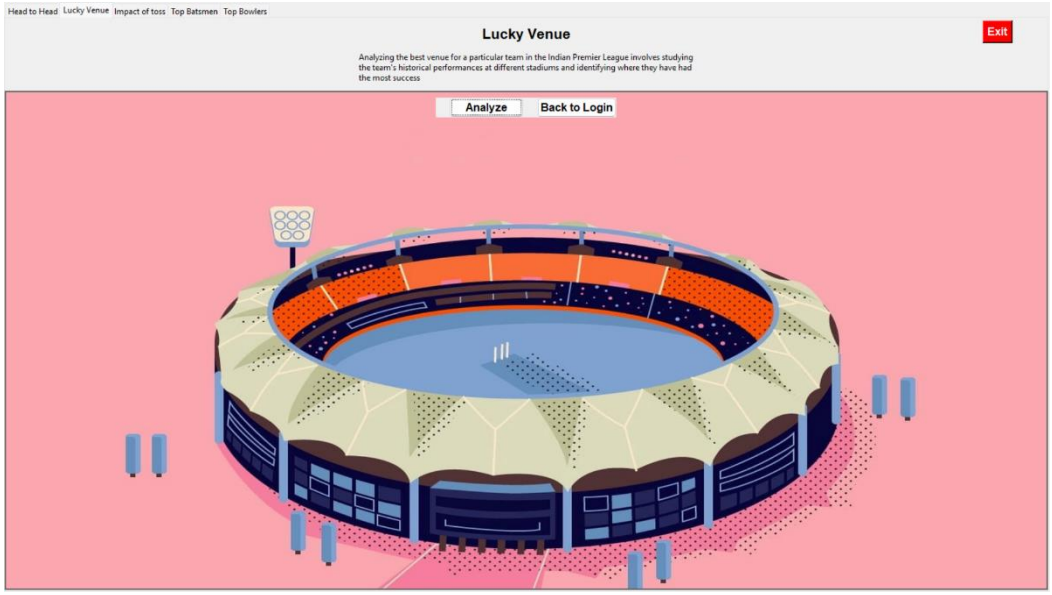
5.3 Output Design

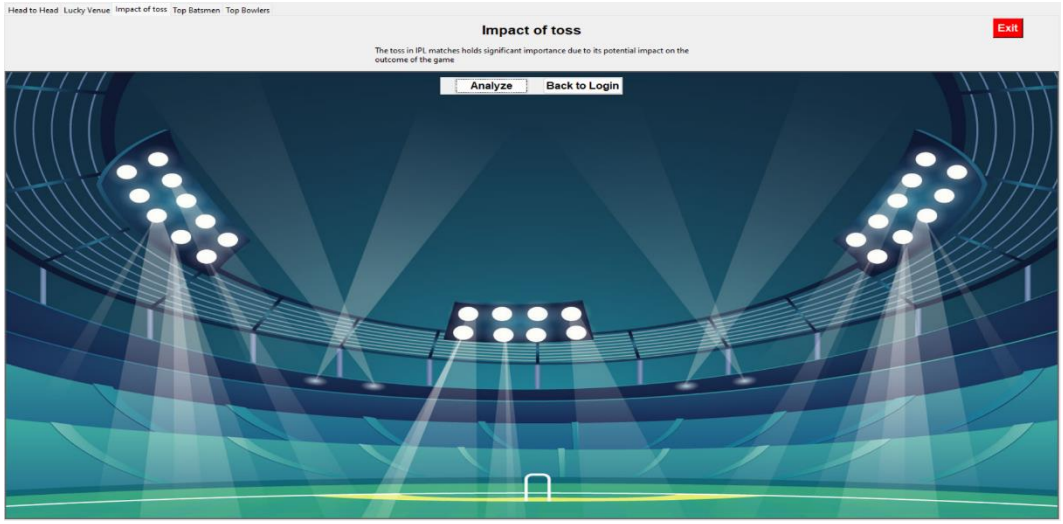


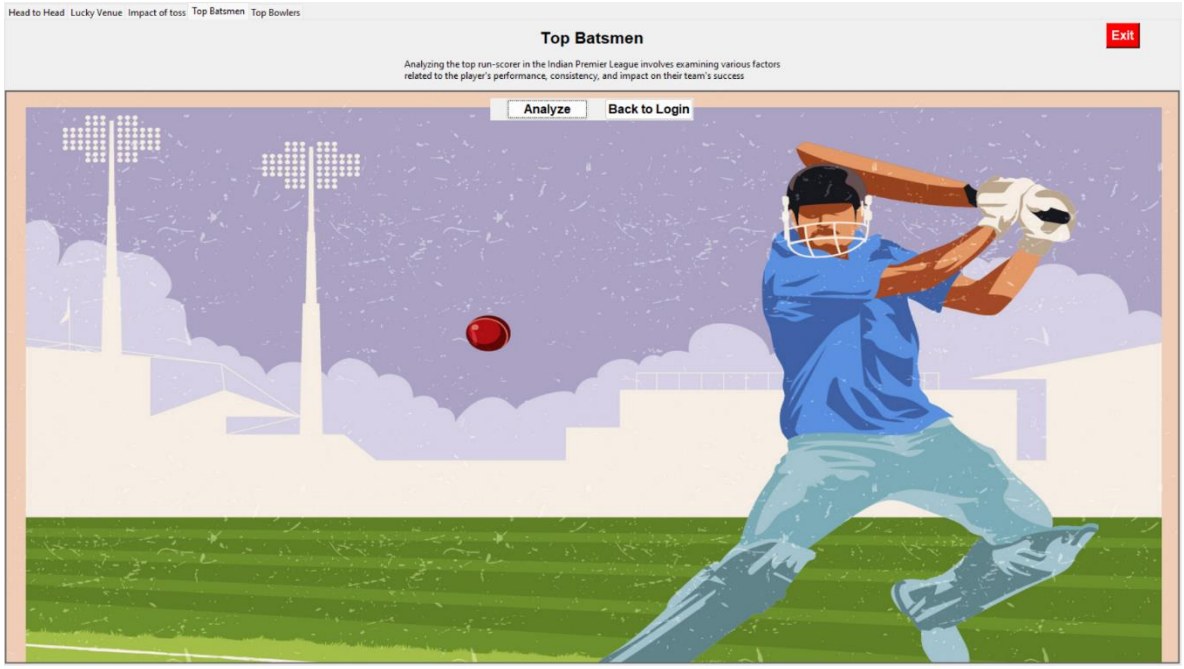


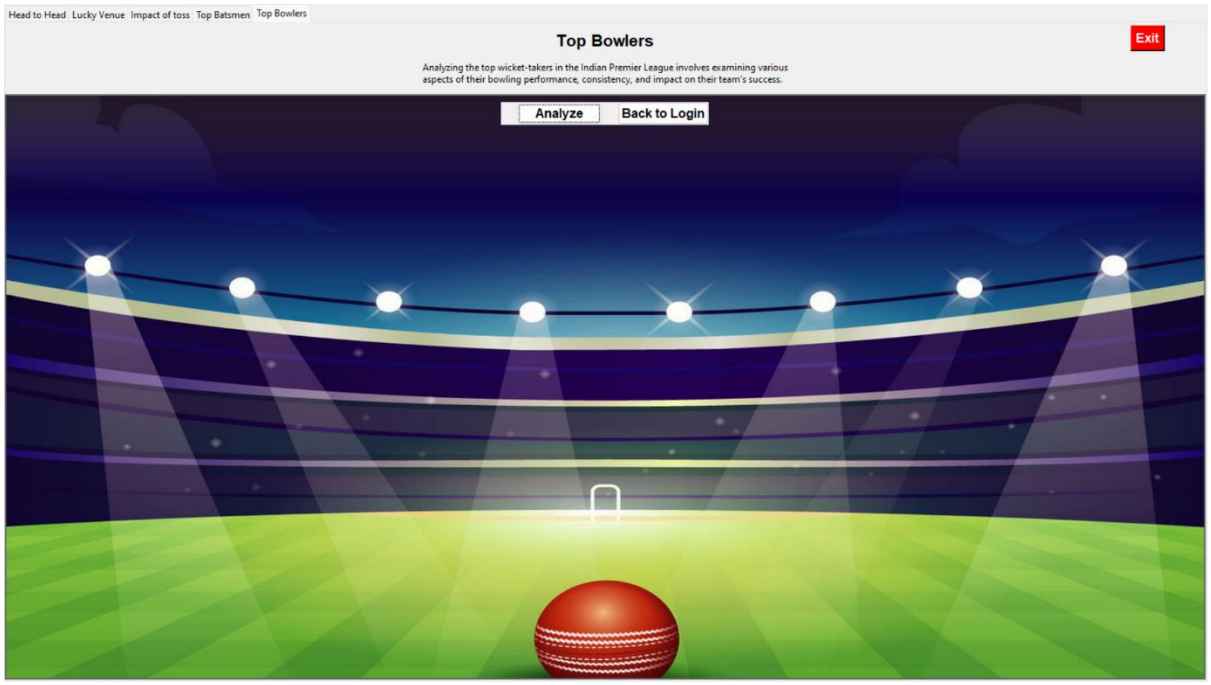

Admin Page
Back
View/Update Users
User 1:
User ID: 101 Password: 123 Name: melvin Phone Number: 1234567890 Age: 21 Update
User 2:
User ID: 101 Password: 123 Name: mekn Phone Number: 1234567890 Age: 20 Update
User 3:
User ID: 101 Password: 123 Name: Melvin Phone Number: 1234567890 Age: 22 Update
User 4:
User ID: 118 Password: 123 Name: Melvin Phone Number: 1234567890 Age: 21 Update
User 5:
User ID: 101 Password: 123 Name: Melvin Phone Number: 1234567890 Age: 22 Update
Create Users
View/Update Users
Delete Users

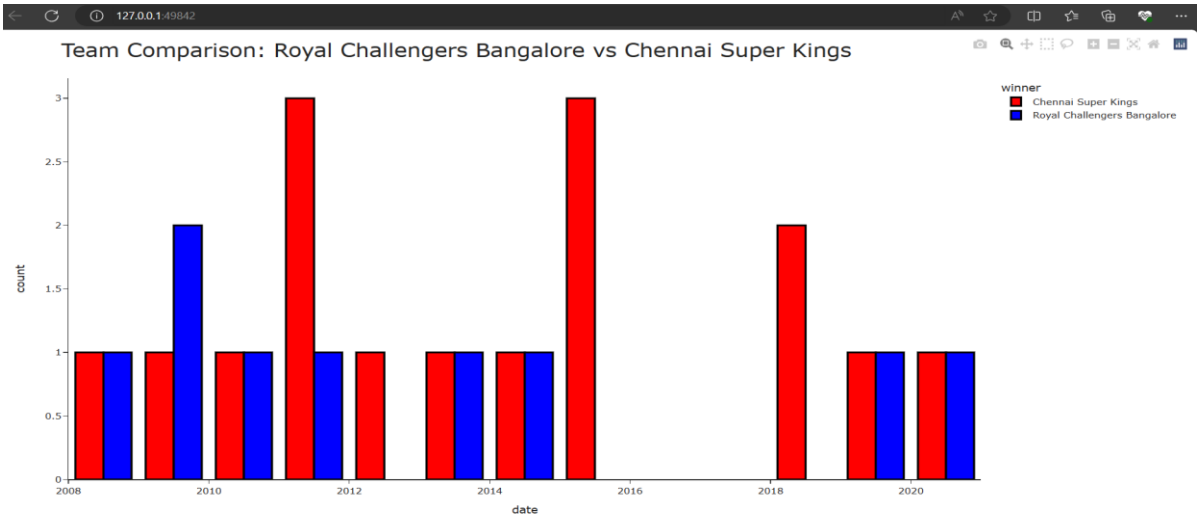

Head to Head
Lucky Venue
Impact of toss
Top Batsmen
Top Bowlers
Head to Head
Exit
Head-to-head analysis of IPL teams involves examining the historical performances and outcomes of matches between two specific teams in the Indian Premier League
Analyze
Back to Login

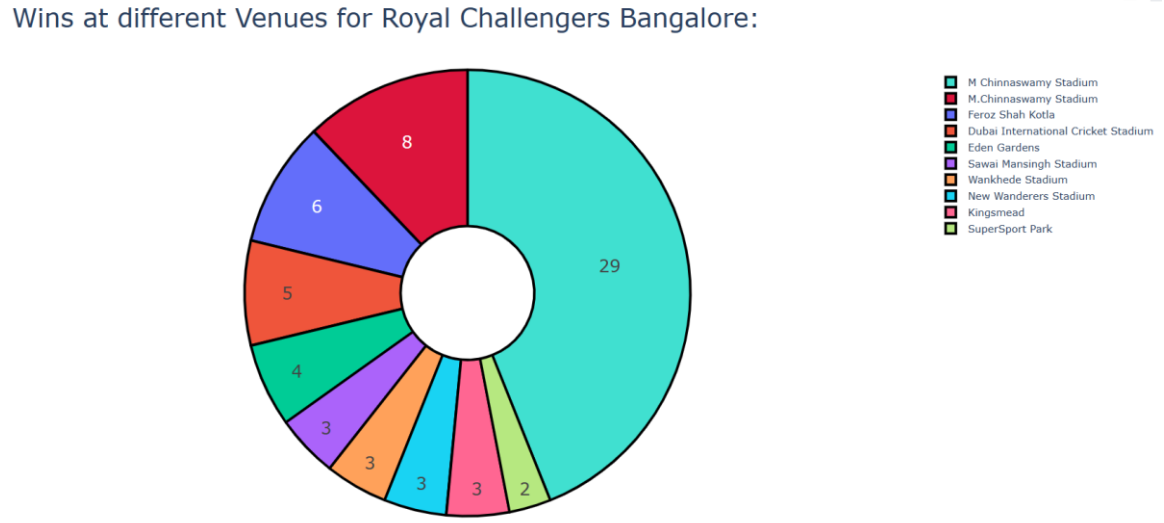

Head to Head
Lucky Venue
Impact of toss
Top Batsmen
Top Bowlers
Lucky Venue
Exit
Analyzing the best venue for a particular team in the Indian Premier League involves studying the team's historical performances at different stadiums and identifying where they have had the most success
Analyze
Back to Login

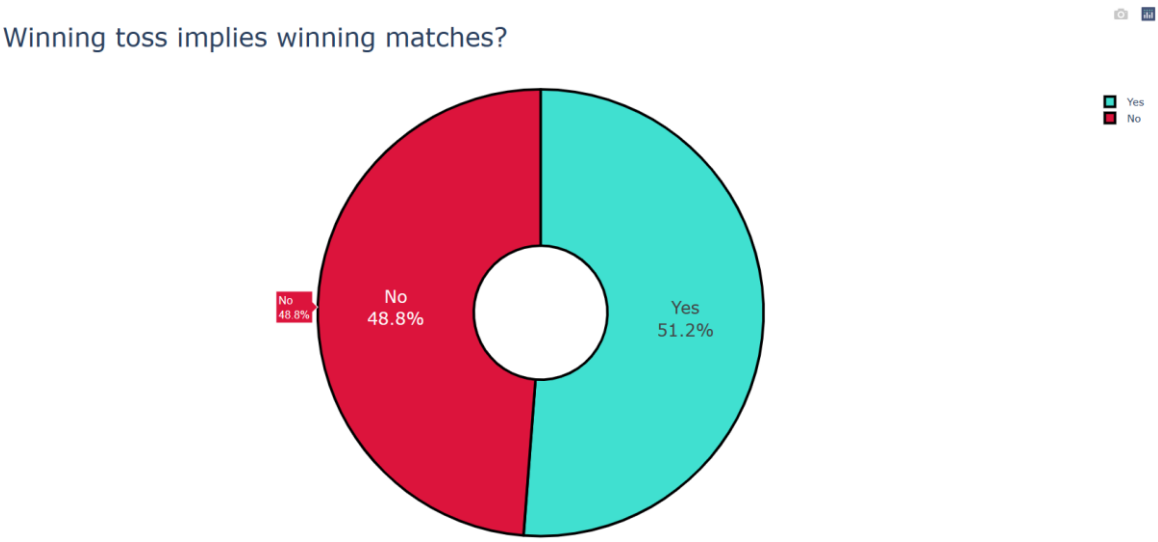

Head to Head
Lucky Venue
Impact of toss
Top Batsmen
Top Bowlers
Impact of toss
Exit
The toss in IPL matches holds significant importance due to its potential impact on the outcome of the game
Analyze
Back to Login

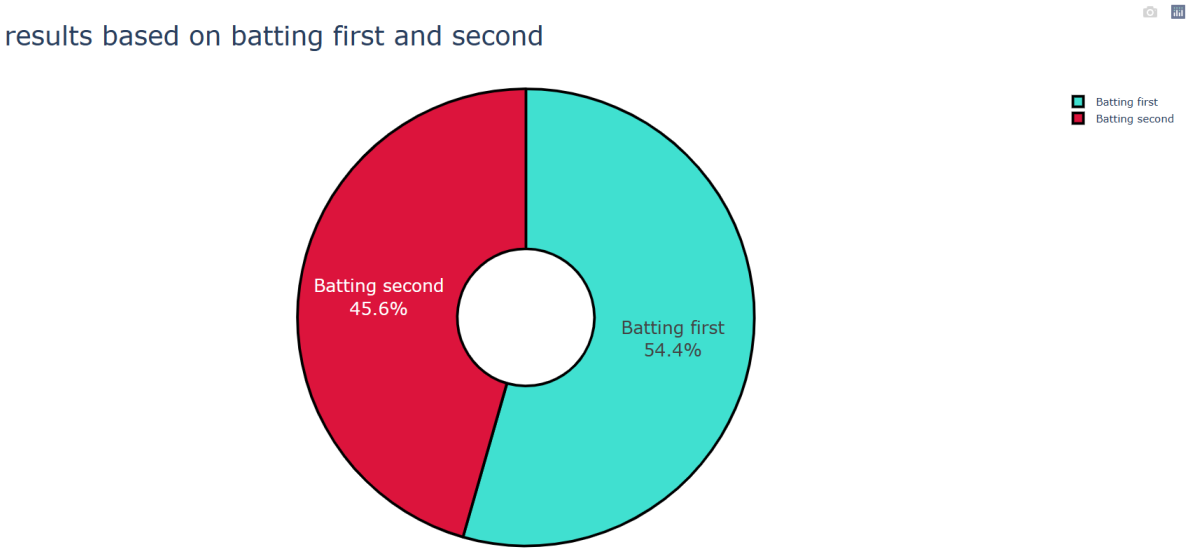

Head to Head
Lucky Venue
Impact of toss
Top Batsmen
Top Bowlers
Top Batsmen
Exit
Analyzing the top run-scorer in the Indian Premier League involves examining various factors related to the player's performance, consistency, and impact on their team's success
Analyze
Back to Login

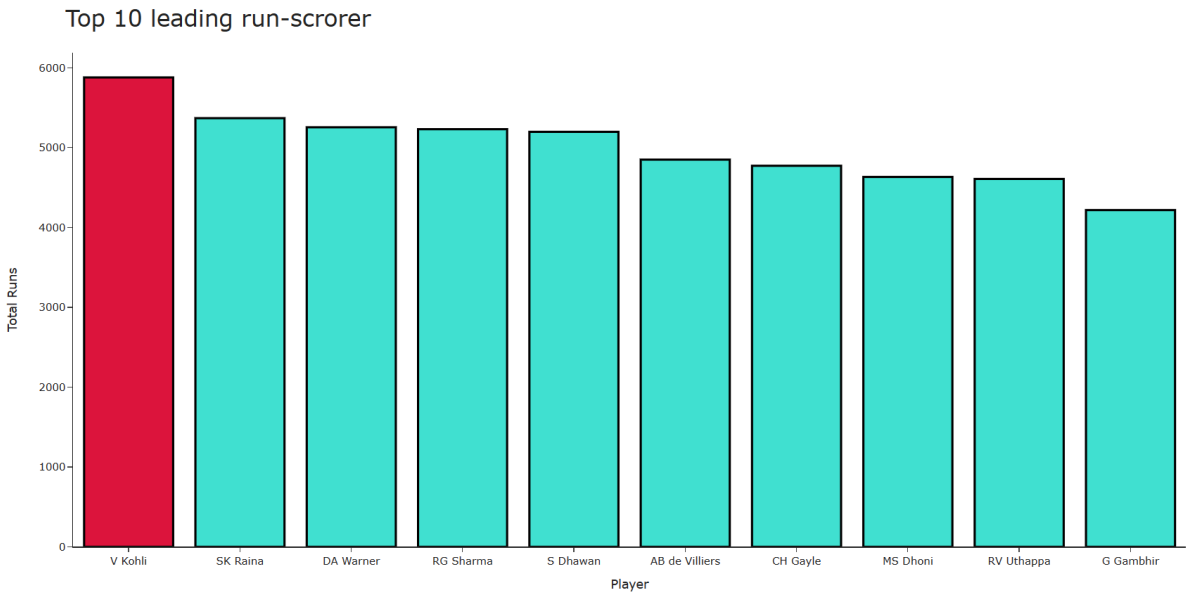

Head to Head
Lucky Venue
Impact of toss
Top Batsmen
Top Bowlers
Top Bowlers
Exit
Analyzing the top wicket-takers in the Indian Premier League involves examining various aspects of their bowling performance, consistency, and impact on their team's success.
Analyze
Back to Login

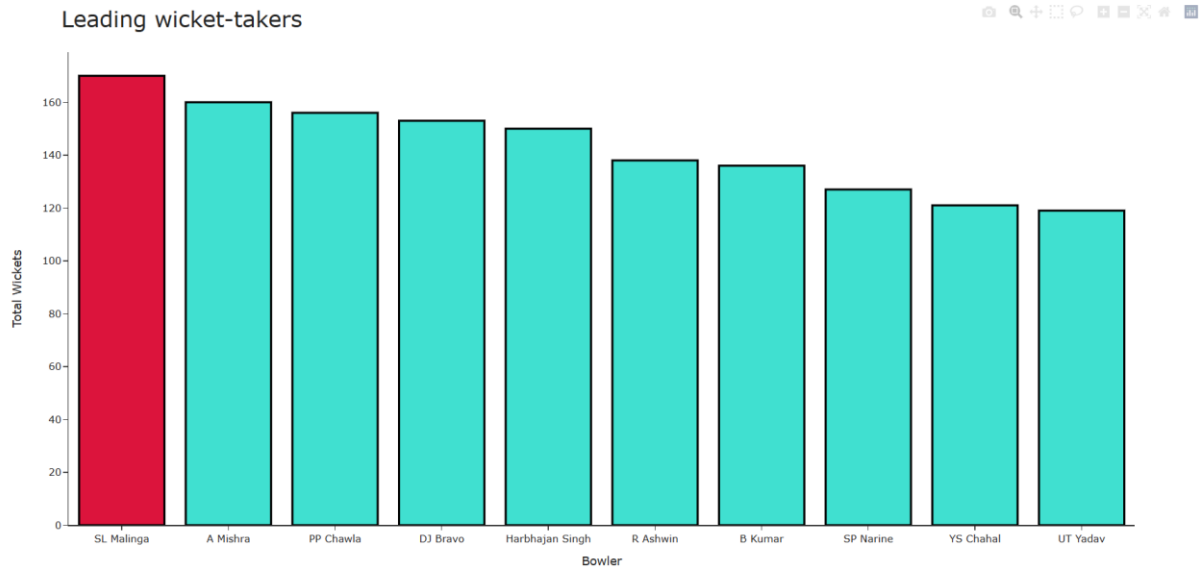

127.0.0.1:49842
Team Comparison: Royal Challengers Bangalore vs Chennai Super Kings
winner
Chennai Super Kings
Royal Challengers Bangalore
count
date
0
0.5
1
1.5
2
2.5
3
2008
2010
2012
2014
2016
2018
2020


Wins at different Venues for Royal Challengers Bangalore:
M Chinnaswamy Stadium
M.Chinnaswamy Stadium
Feroz Shah Kotla
Dubai International Cricket Stadium
Eden Gardens
Sawai Mansingh Stadium
Wankhede Stadium
New Wanderers Stadium
Kingsmead
SuperSport Park
29
8
6
5
4
3
3
3
3
2


Winning toss implies winning matches?
Yes
No
No
48.8%
No
48.8%
Yes
51.2%


results based on batting first and second
Batting first
Batting second
Batting second
45.6%
Batting first
54.4%


Top 10 leading run-scrorer
Total Runs
6000
5000
4000
3000
2000
1000
0
V Kohli
SK Raina
DA Warner
RG Sharma
S Dhawan
AB de Villiers
CH Gayle
MS Dhoni
RV Uthappa
G Gambhir
Player


Leading wicket-takers
Total Wickets
160
140
120
100
80
60
40
20
0
SL Malinga
A Mishra
PP Chawla
DJ Bravo
Harbhajan Singh
R Ashwin
B Kumar
SP Narine
YS Chahal
UT Yadav
Bowler

5.4 Database Design

I. Admin Collection:

Field name	Datatype
admin_id	int
password	string

II.User Collection:

Field name	Datatype
user_id	int
password	string
name	char
phone number	int

III. Match Collection:

Field name	Datatype
id	int
city	char
date	date
player of match	char
venue	char
neutral venue	int
team1	char
team2	char
toss winner	char

winner	char
result	char
result margin	int
umpire1	char
umpire2	char

IV. Scores Collection:

Field name	Datatype
id	int
inning	int
ball	int
batsmen	char
non striker	char
bowler	char
batsmen	int
extra runs	int
total runs	int
non boundary	int
Is wicket	int
batting_team	char
bowling team	char

6. SOURCE CODE

1.Login.py

```
import tkinter as tk
from tkinter import messagebox
from PIL import Image, ImageTk
from pymongo import MongoClient
import subprocess

# MongoDB connection
mongo_client = MongoClient('mongodb://localhost:27017/')
mongo_db = mongo_client['cricket']
mongo_collection = mongo_db['users']

# Function to handle login button click
import pymongo
import tkinter as tk
from tkinter import messagebox
from PIL import Image, ImageTk
from pymongo import MongoClient
import subprocess

class Database:
    def __init__(self):
        self.client = MongoClient('mongodb://localhost:27017/')
        self.db = self.client['cricket']
        self.users_collection = self.db['users']
        self.admin_collection = self.db['admin']

    def close_connection(self):
        self.client.close()

    def authenticate_user(self, user_id, password):
        return self.users_collection.find_one({"User ID": user_id, "Password": password})

    def authenticate_admin(self, admin_id, password):
        return self.admin_collection.find_one({"admin_id": admin_id, "password":
password})

class LoginWindow:
    def __init__(self, root, user_type):
        self.root = root
        self.user_type = user_type
        self.init_ui()

    def init_ui(self):
        self.login_window = tk.Toplevel(self.root)
        self.login_window.title("Cricket Data Analysis - Login")
```

```
        window_width = 300
        window_height = 150
        screen_width = self.root.winfo_screenwidth()
        screen_height = self.root.winfo_screenheight()
        x = (screen_width - window_width) // 2
        y = (screen_height - window_height) // 2
        self.login_window.geometry(f"{window_width}x{window_height}+{x}+{y}")

        self.create_widgets()

    def create_widgets(self):
        username_label = tk.Label(self.login_window, text="User ID:")
        username_label.grid(row=0, column=0, padx=10, pady=5, sticky=tk.E)

        self.username_entry = tk.Entry(self.login_window)
        self.username_entry.grid(row=0, column=1, padx=10, pady=5)

        password_label = tk.Label(self.login_window, text="Password:")
        password_label.grid(row=1, column=0, padx=10, pady=5, sticky=tk.E)

        self.password_entry = tk.Entry(self.login_window, show="*")
        self.password_entry.grid(row=1, column=1, padx=10, pady=5)

        login_button = tk.Button(self.login_window, text="Login",
command=self.perform_login,
                                 width=15, height=2, relief=tk.RAISED, bg="#4CAF50",
fg="white", font=("Helvetica", 12, "bold"))
        login_button.grid(row=2, column=1, pady=10)

    def perform_login(self):
        user_id = self.username_entry.get()
        password = self.password_entry.get()

        if self.user_type == "user":
            user_data = db.authenticate_user(user_id, password)
            self.handle_login_result(user_data, "User", user_id)
        elif self.user_type == "admin":
            admin_data = db.authenticate_admin(user_id, password)
            self.handle_login_result(admin_data, "Admin", "Admin")

    def handle_login_result(self, data, role, user_id):
        if data:
            messagebox.showinfo("Login Successful", f"Welcome, {user_id} ({role})!")
            launch_next_file("analysis.py") if role == "User" else
launch_next_file("crud.py")
            self.login_window.destroy()
        else:
            messagebox.showerror("Login Failed", f"Invalid {role} ID or password for
{user_id}")
```

```
    def launch_next_file(filename):
        try:
            subprocess.run(["python", filename], check=True)
        except subprocess.CalledProcessError as e:
            messagebox.showerror("Error", f"Failed to launch the next file: {e}")

    def open_user_login():
        LoginWindow(root, "user")

    def open_admin_login():
        LoginWindow(root, "admin")

    if __name__ == "__main__":
        db = Database()

        root = tk.Tk()
        root.title("Cricket Data Analysis")

        try:
            image_path = r"D:\Photos\Index.jpg"  # Replace with your image file path
            background_image = Image.open(image_path)
            background_photo = ImageTk.PhotoImage(background_image)
        except FileNotFoundError:
            messagebox.showerror("Image Error", f"Image file not found at: {image_path}")
            root.destroy()
        else:
            screen_width = root.winfo_screenwidth()
            screen_height = root.winfo_screenheight()

            root.geometry(f"{screen_width}x{screen_height}+0+0")
            root.attributes("-fullscreen", True)

            background_label = tk.Label(root, image=background_photo)
            background_label.place(relwidth=1, relheight=1)

            button_width = 15
            button_height = 2
            button_relx = 0.5
            user_button = tk.Button(root, text="User", command=open_user_login,
                                    width=button_width, height=button_height, relief=tk.RAISED,
bg="#3366CC", fg="white",
                                    font=("Helvetica", 12, "bold"), bd=3, highlightthickness=0)
            admin_button = tk.Button(root, text="Admin", command=open_admin_login,
                                     width=button_width, height=button_height, relief=tk.RAISED,
bg="#FF6347", fg="white",
                                     font=("Helvetica", 12, "bold"), bd=3, highlightthickness=0)

            user_button.place(relx=button_relx, rely=0.49, anchor="center")
            admin_button.place(relx=button_relx, rely=0.72, anchor="center")
```

```
            root.mainloop()

        db.close_connection()

    # Function to handle login button click
    import pymongo

    # Function to handle login button click
    import pymongo
    from tkinter import messagebox

    # Function to handle login button click
    def login(user_radio_var, admin_radio_var, username_entry, password_entry):
        entered_user_id = username_entry.get()  # Assuming the input field is still named
'username_entry'
        entered_password = password_entry.get()

        # MongoDB connection
        try:
            client = pymongo.MongoClient("mongodb://localhost:27017/")  # Update the
MongoDB connection string
            db = client["cricket"]
            collection = db["users"]
        except Exception as e:
            messagebox.showerror("Database Connection Error", f"Failed to connect to the
database: {e}")
            return

        # Check if either the User or Admin radio button is selected
        if user_radio_var.get() == 1:
            # Check if the entered credentials are valid for User
            user_query = {"user_id": entered_user_id, "password": entered_password,
"user_type": "user"}
            user_result = collection.find_one(user_query)

            if user_result:
                messagebox.showinfo("Login Successful", f"Welcome, {entered_user_id}!")
            else:
                messagebox.showerror("Login Failed", "Invalid user ID or password for User")
        elif admin_radio_var.get() == 1:
            # Check if the entered credentials are valid for Admin
            admin_query = {"user_id": entered_user_id, "password": entered_password,
"user_type": "admin"}
            admin_result = collection.find_one(admin_query)

            if admin_result:
                messagebox.showinfo("Login Successful", f"Welcome, {entered_user_id}
(Admin)!")
                launch_next_file("crud.py")
            else:
```

```
                messagebox.showerror("Login Failed", "Invalid user ID or password for Admin")
            else:
                messagebox.showerror("Login Failed", "Please select User or Admin")

            # Close the MongoDB connection
            client.close()
        # Rest of the code remains unchanged
        # Function to handle user/admin selection and open login page
        def open_login_page(user_type, user_radio_var, admin_radio_var):
            login_window = tk.Toplevel(root)
            login_window.title("Cricket Data Analysis - Login")

            # Calculate the window position for centering
            window_width = 300
            window_height = 150
            screen_width = root.winfo_screenwidth()
            screen_height = root.winfo_screenheight()
            x = (screen_width - window_width) // 2
            y = (screen_height - window_height) // 2

            # Set the window dimensions and position
            login_window.geometry(f"{window_width}x{window_height}+{x}+{y}")

            # Create widgets for login page
            username_label = tk.Label(login_window, text="User Id:")
            username_label.grid(row=0, column=0, padx=10, pady=5, sticky=tk.E)

            username_entry = tk.Entry(login_window)
            username_entry.grid(row=0, column=1, padx=10, pady=5)

            password_label = tk.Label(login_window, text="Password:")
            password_label.grid(row=1, column=0, padx=10, pady=5, sticky=tk.E)

            password_entry = tk.Entry(login_window, show="*")
            password_entry.grid(row=1, column=1, padx=10, pady=5)

            login_button = tk.Button(login_window, text="Login", command=lambda:
login(user_radio_var, admin_radio_var, username_entry, password_entry),
                                     width=15, height=2, relief=tk.RAISED, bg="#4CAF50", fg="white",
font=("Helvetica", 12, "bold"))
            login_button.grid(row=2, column=1, pady=10)


        # Function to update radio button variables
        def update_radio_vars(value, user_radio_var, admin_radio_var):
            user_radio_var.set(0)
            admin_radio_var.set(0)
            if value == "user":
                user_radio_var.set(1)
            elif value == "admin":
```

```
        admin_radio_var.set(1)

    def launch_next_file(filename):
        try:
            subprocess.run(["python", filename], check=True)
        except subprocess.CalledProcessError as e:
            messagebox.showerror("Error", f"Failed to launch the next file: {e}")


    root = tk.Tk()
    root.title("Cricket Data Analysis")

    # Variables for storing radio button values
    user_radio_var = tk.IntVar()
    admin_radio_var = tk.IntVar()

    # Function to open login page for User
    def open_user_login():
        update_radio_vars("user", user_radio_var, admin_radio_var)
        open_login_page("User", user_radio_var, admin_radio_var)

    # Function to open login page for Admin
    def open_admin_login():
        update_radio_vars("admin", user_radio_var, admin_radio_var)
        open_login_page("Admin", user_radio_var, admin_radio_var)

    # Load the background image using PIL
    try:
        image_path = r"D:\Photos\Index.jpg"  # Replace with your image file path
        background_image = Image.open(image_path)
        background_photo = ImageTk.PhotoImage(background_image)
    except FileNotFoundError:
        messagebox.showerror("Image Error", f"Image file not found at: {image_path}")
        root.destroy()
    else:
        # Set window size and position
        screen_width = root.winfo_screenwidth()
        screen_height = root.winfo_screenheight()

        root.geometry(f"{screen_width}x{screen_height}+0+0")

        # Set window to full screen without borders
        root.attributes("-fullscreen", True)

        # Create a label to display the background image
        background_label = tk.Label(root, image=background_photo)
        background_label.place(relwidth=1, relheight=1)

        # Customize button appearance and size
        button_width = 15
```

```
        button_height = 2
        button_relx = 0.5
        user_button = tk.Button(root, text="User", command=open_user_login,
width=button_width, height=button_height, relief=tk.RAISED,
                                bg="#3366CC", fg="white", font=("Helvetica", 12, "bold"), bd=3,
highlightthickness=0)
        admin_button = tk.Button(root, text="Admin", command=open_admin_login,
width=button_width, height=button_height, relief=tk.RAISED,
                                 bg="#FF6347", fg="white", font=("Helvetica", 12, "bold"), bd=3,
highlightthickness=0)

        # Place buttons in the center of the window with some spacing
        user_button.place(relx=button_relx, rely=0.49, anchor="center")
        admin_button.place(relx=button_relx, rely=0.72, anchor="center")

        # Run the main loop
        root.mainloop()
```

2.CRUD.py

```
`import tkinter as tk
from tkinter import messagebox
from pymongo import MongoClient
from PIL import Image, ImageTk
import os

class AdminPage:
    def __init__(self, root):
        self.root = root
        self.root.title("Admin Page")

        # MongoDB connection
        self.client = MongoClient('mongodb://localhost:27017/')
        self.db = self.client['cricket']
        self.collection = self.db['users']

        # Background Image
        image_path = 'D:\\Photos\\Index.jpg'
        if not os.path.isfile(image_path):
            messagebox.showerror("Image Error", f"Image file not found at: {image_path}")
        else:
            self.img = Image.open(image_path)
            self.img = ImageTk.PhotoImage(self.img)

            background_label = tk.Label(self.root, image=self.img)
            background_label.place(relwidth=1, relheight=1)

        self.frame = tk.Frame(self.root)
        self.frame.pack(padx=20, pady=20)
```

```
        self.label = tk.Label(self.frame, text="Admin Page", font=("Helvetica", 18, "bold"))
        self.label.grid(row=0, column=0, columnspan=2, pady=10)

        # Buttons for different modules
        create_users_button = tk.Button(self.frame, text="Create Users",
command=self.create_users)
        create_users_button.grid(row=1, column=0, padx=10, pady=10)

        view_users_button = tk.Button(self.frame, text="View/Update Users",
command=self.view_users)
        view_users_button.grid(row=1, column=1, padx=10, pady=10)

        delete_users_button = tk.Button(self.frame, text="Delete Users",
command=self.delete_users)
        delete_users_button.grid(row=2, column=0, columnspan=2, padx=10, pady=10)

        # User database (in-memory representation)
        self.users = []

      def add_user_to_mongodb(self, user_id, password, name, phone, age,
create_window):
        user_info = {
            "User ID": user_id,
            "Password": password,
            "Name": name,
            "Phone Number": phone,
            "Age": age
        }

        try:
            # Insert user information into MongoDB
            result = self.collection.insert_one(user_info)

            if result.inserted_id:
                messagebox.showinfo("User Created", "User created successfully and added to
MongoDB!")
                create_window.destroy()  # Close the create user window
            else:
                messagebox.showerror("Error", "Failed to add user to MongoDB.")
        except Exception as e:
            messagebox.showerror("Error", f"An error occurred: {str(e)}")

    def create_users(self):
        create_window = tk.Toplevel(self.root)
        create_window.title("Create User")

        # Create entry widgets for user input
        user_id_label = tk.Label(create_window, text="User ID:")
        user_id_label.grid(row=0, column=0, padx=10, pady=5, sticky=tk.E)
        user_id_entry = tk.Entry(create_window)
```

```
        user_id_entry.grid(row=0, column=1, padx=10, pady=5)

        password_label = tk.Label(create_window, text="Password:")
        password_label.grid(row=1, column=0, padx=10, pady=5, sticky=tk.E)
        password_entry = tk.Entry(create_window, show="*")
        password_entry.grid(row=1, column=1, padx=10, pady=5)

        name_label = tk.Label(create_window, text="Name:")
        name_label.grid(row=2, column=0, padx=10, pady=5, sticky=tk.E)
        name_entry = tk.Entry(create_window)
        name_entry.grid(row=2, column=1, padx=10, pady=5)

        phone_label = tk.Label(create_window, text="Phone Number:")
        phone_label.grid(row=3, column=0, padx=10, pady=5, sticky=tk.E)
        phone_entry = tk.Entry(create_window)
        phone_entry.grid(row=3, column=1, padx=10, pady=5)

        age_label = tk.Label(create_window, text="Age:")
        age_label.grid(row=4, column=0, padx=10, pady=5, sticky=tk.E)
        age_entry = tk.Entry(create_window)
        age_entry.grid(row=4, column=1, padx=10, pady=5)

        submit_button = tk.Button(create_window, text="Submit", command=lambda:
self.add_user_to_mongodb(
            user_id_entry.get(), password_entry.get(), name_entry.get(), phone_entry.get(),
age_entry.get(), create_window))
        submit_button.grid(row=5, column=1, pady=10)

    def view_users(self):
        view_window = tk.Toplevel(self.root)
        view_window.title("View/Update Users")

        # Fetch all users from MongoDB
        users_from_mongo = self.collection.find()

        for i, user in enumerate(users_from_mongo):
            user_info_label = tk.Label(view_window, text=f"User {i + 1}:")
            user_info_label.grid(row=i * 2, column=0, padx=10, pady=5)

            # Display user information
            user_id_label = tk.Label(view_window, text=f"User ID: {user['User ID']}")
            user_id_label.grid(row=i * 2 + 1, column=0, padx=10, pady=5)

            password_label = tk.Label(view_window, text=f"Password: {user['Password']}")
            password_label.grid(row=i * 2 + 1, column=1, padx=10, pady=5)

            name_label = tk.Label(view_window, text=f"Name: {user['Name']}")
            name_label.grid(row=i * 2 + 1, column=2, padx=10, pady=5)
```

```
            phone_label = tk.Label(view_window, text=f"Phone Number: {user['Phone
Number']}")
            phone_label.grid(row=i * 2 + 1, column=3, padx=10, pady=5)

            age_label = tk.Label(view_window, text=f"Age: {user['Age']}")
            age_label.grid(row=i * 2 + 1, column=4, padx=10, pady=5)

            # Add an "Update" button for each user
            update_button = tk.Button(view_window, text="Update", command=lambda
u=user: self.open_update_window(u))
            update_button.grid(row=i * 2 + 1, column=5, padx=10, pady=5)

            # Separator below each user
            separator = tk.Label(view_window, text="------------------------")
            separator.grid(row=i * 2 + 2, column=0, columnspan=6, padx=10, pady=5)

        if users_from_mongo.count() == 0:
            messagebox.showinfo("No Users", "No users found in MongoDB.")

    def open_update_window(self, user):
        update_window = tk.Toplevel(self.root)
        update_window.title("Update User Info")

        # Center the update window on the screen
        window_width = 300
        window_height = 200
        screen_width = update_window.winfo_screenwidth()
        screen_height = update_window.winfo_screenheight()
        x = (screen_width - window_width) // 2
        y = (screen_height - window_height) // 2

        update_window.geometry(f"{window_width}x{window_height}+{x}+{y}")

        # Bring the update window to the front
        update_window.lift(self.root)
        update_window.attributes('-topmost', True)
        update_window.attributes('-topmost', False)  # Allow other windows to be on top

        # Populate the entry fields with the current user information
        password_entry = tk.Entry(update_window, show="*")
        password_entry.insert(0, user["Password"])
        password_entry.grid(row=1, column=1, padx=10, pady=5)

        name_entry = tk.Entry(update_window)
        name_entry.insert(0, user["Name"])
        name_entry.grid(row=2, column=1, padx=10, pady=5)

        phone_entry = tk.Entry(update_window)
        phone_entry.insert(0, user["Phone Number"])
        phone_entry.grid(row=3, column=1, padx=10, pady=5)
```

```
        age_entry = tk.Entry(update_window)
        age_entry.insert(0, user["Age"])
        age_entry.grid(row=4, column=1, padx=10, pady=5)

        update_button = tk.Button(update_window, text="Update",
                                  command=lambda: self.update_user_info(user, password_entry.get(),
name_entry.get(),
                                                                        phone_entry.get(), age_entry.get(),
update_window))
        update_button.grid(row=5, column=1, pady=10)

        messagebox.showinfo("Update", "User information update initiated. Click 'Update' to
confirm changes.")
    def update_user_info(self, user, password, name, phone, age, update_window):
        user["Password"] = password
        user["Name"] = name
        user["Phone Number"] = phone
        user["Age"] = age
        messagebox.showinfo("Update Successful", "User information updated
successfully!")

        # Update the user information in MongoDB
        self.collection.update_one({"User ID": user["User ID"]},
                                   {"$set": {"Password": password, "Name": name, "Phone Number":
phone, "Age": age}})

        update_window.destroy()  # Close the update user window

    def delete_users(self):
        delete_window = tk.Toplevel(self.root)
        delete_window.title("Delete Users")

        # Fetch all users from MongoDB
        users_from_mongo = self.collection.find()

        for i, user in enumerate(users_from_mongo):
            user_info_label = tk.Label(delete_window, text=f"User {i + 1}:")
            user_info_label.grid(row=i * 2, column=0, padx=10, pady=5)

            # Display user information
            user_id_label = tk.Label(delete_window, text=f"User ID: {user['User ID']}")
            user_id_label.grid(row=i * 2 + 1, column=0, padx=10, pady=5)

            name_label = tk.Label(delete_window, text=f"Name: {user['Name']}")
            name_label.grid(row=i * 2 + 1, column=1, padx=10, pady=5)

            # Add a "Delete" button for each user
            delete_button = tk.Button(delete_window, text="Delete", command=lambda
u=user: self.delete_user(u, delete_window))
```

```
            delete_button.grid(row=i * 2 + 1, column=2, padx=10, pady=5)

            # Separator below each user
            separator = tk.Label(delete_window, text="------------------------")
            separator.grid(row=i * 2 + 2, column=0, columnspan=3, padx=10, pady=5)

        if users_from_mongo.count() == 0:
            messagebox.showinfo("No Users", "No users found in MongoDB.")

    def delete_user(self, user, delete_window):
        self.collection.delete_one({"User ID": user["User ID"]})
        messagebox.showinfo("Delete Successful", "User deleted successfully!")
        delete_window.destroy()  # Close the delete user window

if __name__ == "__main__":
    root = tk.Tk()
    admin_page = AdminPage(root)
    root.mainloop()
```

3.Analysis.py

```
import tkinter as tk
from tkinter import ttk
from PIL import Image, ImageTk
import subprocess


class AnalysisPage(tk.Tk):
    def __init__(self):
        super().__init__()

        self.title("Analysis Page")

        # Create a notebook to contain multiple pages
        self.notebook = ttk.Notebook(self)

        # Create five sections with titles, descriptions, images, and dropdown lists
        sections = [
            {"title": "Head to Head", "description": "Head to Head", "image_path":
r"C:\Users\Nandan\Downloads\IPL.gif"},
            {"title": "Lucky Venue", "description": "Lucky Venue", "image_path":
r"C:\Users\Nandan\Downloads\IPL.gif"},  # Add an empty list for teams
            {"title": "Impact of toss", "description": "Impact of Toss", "image_path":
r"C:\Users\Nandan\Downloads\IPL.gif"},
            {"title": "Top Batsmen", "description": "Top runs scorer", "image_path":
r"C:\Users\Nandan\Downloads\IPL.gif"},
            {"title": "Top Bowlers", "description": "Top Wicket Takers", "image_path":
r"C:\Users\Nandan\Downloads\IPL.gif"},
        ]
```

```
            for section in sections:
                frame = ttk.Frame(self.notebook)
                self.create_section(frame, section)
                self.notebook.add(frame, text=section["title"])

            self.notebook.pack(expand=1, fill="both")
            def create_section(self, frame, section):
            # Add title label
            title_label = ttk.Label(frame, text=section["title"], font=("Helvetica", 16, "bold"))
            title_label.pack(pady=(10, 0))

            # Add description label
            description_label = ttk.Label(frame, text=section["description"], wraplength=300,
padding=(10, 10))
            description_label.pack()

            # Add Canvas for the background image
            canvas = tk.Canvas(frame, width=800, height=600)  # Set the canvas size according
to your image size
            canvas.pack()

            # Add image as background
            if "image_path" in section:
                image = Image.open(section["image_path"])
                section["photo_image"] = ImageTk.PhotoImage(image)
                canvas.create_image(0, 0, anchor=tk.NW, image=section["photo_image"])

            # Add button to lead to a specific page
            button = ttk.Button(frame, text="Go to Page", command=lambda:
self.show_specific_page(section["title"]))
            button.pack()
        def show_specific_page(self, section_title):
            if section_title == "Head to Head":
                # Launch HeadToHead.py as a separate process
                subprocess.Popen(["python", "HeadToHead.py"])
            elif section_title == "Impact of toss":
                subprocess.Popen(["Python", "ImpactOfToss.py"])
            elif section_title == "Lucky Venue":
                subprocess.Popen(["Python","LuckyVenues.py"])
            elif section_title == "Top Batsmen":
                subprocess.Popen(["Python","TopBatsmen.py"])
            elif section_title == "Top Bowlers":
                subprocess.Popen(["Python","TopBowler.py"])
            else:
                print(f"Button clicked for {section_title}")

    if __name__ == "__main__":
        app = AnalysisPage()
        app.mainloop()
```

4.Headtohead.py

```
import pandas as pd
import plotly.express as px
import tkinter as tk
from tkinter import ttk
import pandas as pd
from pymongo import MongoClient

# MongoDB connection parameters
mongodb_host = 'mongodb://localhost:27017/'
mongodb_port = 27017
database_name = 'cricket'

# Connect to MongoDB
client = MongoClient(mongodb_host, mongodb_port)
db = client[database_name]

# Read data from MongoDB collections
deliveries_data_cursor = db.All_scores.find()
match_data_cursor = db.All_matches.find()

# Convert MongoDB cursors to pandas DataFrames
deliveries_data = pd.DataFrame(list(deliveries_data_cursor))
match_data = pd.DataFrame(list(match_data_cursor))

# Define colors
colors = ['red', 'blue', 'green', 'orange', 'purple']  # You can customize the list of colors as
needed

class TeamComparisonApp(tk.Tk):
    def __init__(self, sections):
        super().__init__()

        self.title("Team Comparison App")

        # Create a notebook to contain multiple pages
        self.notebook = ttk.Notebook(self)

        # Create a frame for team selection
        team_frame = ttk.Frame(self.notebook)
        self.create_team_selection_frame(team_frame, sections)
        self.notebook.add(team_frame, text="Team Selection")

        self.notebook.pack(expand=1, fill="both")

    def create_team_selection_frame(self, frame, sections):
        # Add title label
```

```
            title_label = ttk.Label(frame, text="Select Teams for Comparison",
font=("Helvetica", 16, "bold"))
            title_label.pack(pady=(10, 20))

            # Add dropdown lists for team selection
            team1_label = ttk.Label(frame, text="Team 1:")
            team1_label.pack()

            self.team1_combobox = ttk.Combobox(frame, values=sections[0]["teams"])
            self.team1_combobox.pack()

            team2_label = ttk.Label(frame, text="Team 2:")
            team2_label.pack()

            self.team2_combobox = ttk.Combobox(frame, values=sections[0]["teams"])
            self.team2_combobox.pack()

            # Add button to initiate comparison
            compare_button = ttk.Button(frame, text="Compare Teams",
command=self.compare_teams)
            compare_button.pack()

        def compare_teams(self):
            # Get selected teams from dropdown lists
            team1 = self.team1_combobox.get()
            team2 = self.team2_combobox.get()

            # Check if both teams are selected
            if not team1 or not team2:
                tk.messagebox.showinfo("Error", "Please select both Team 1 and Team 2.")
                return

            self.show_comparison(team1, team2)

        def show_comparison(self, team1, team2):
            compare = match_data[((match_data['team1'] == team1) | (match_data['team2'] ==
team1)) & ((match_data['team1'] == team2) | (match_data['team2'] == team2))]

            # Print columns for debugging
            print(compare.columns)

            # Check if 'date' is in the columns
            if 'date' not in compare.columns:
                raise ValueError("The 'date' column is not present in the DataFrame.")

            fig = px.histogram(data_frame=compare, x='date', color='winner',
labels=dict(x="date", y="Count"), barmode='group', nbins=16,
color_discrete_sequence=colors)
```

```
            fig.update_layout(title=f"Team Comparison: {team1} vs {team2}", titlefont={'size':
26}, template='simple_white')

            fig.update_traces(marker_line_color='black', marker_line_width=2.5, opacity=1)

            fig.show(full_screen=True)

        # Example usage
        if __name__ == "__main__":
            # Pass the sections with team names to the TeamComparisonApp
            sections = [
                {"title": "Head to Head", "description": "Head to Head", "image_path":
r"C:\Users\Nandan\Downloads\IPL.gif", "teams": ["Royal Challengers Bangalore", "Kings
XI Punjab", "Delhi Daredevils", "Mumbai Indians", "Kolkata Knight Riders", "Rajasthan
Royals", "Deccan Chargers", "Chennai Super Kings", "Kochi Tuskers Kerala", "Pune
Warriors", "Sunrisers Hyderabad", "Gujarat Lions", "Rising Pune Supergiants", "Delhi
Capitals"]},
                {"title": "Lucky Venue", "description": "Lucky Venue", "image_path":
r"C:\Users\Nandan\Downloads\IPL.gif", "teams": ["Royal Challengers Bangalore", "Kings
XI Punjab", "Delhi Daredevils", "Mumbai Indians", "Kolkata Knight Riders", "Rajasthan
Royals", "Deccan Chargers", "Chennai Super Kings", "Kochi Tuskers Kerala", "Pune
Warriors", "Sunrisers Hyderabad", "Gujarat Lions", "Rising Pune Supergiants", "Delhi
Capitals"]},
                {"title": "Impact of toss", "description": "Impact of Toss", "image_path":
r"C:\Users\Nandan\Downloads\IPL.gif", "teams": []},
                {"title": "Top Batsmens", "description": "Top runs scorer", "image_path":
r"C:\Users\Nandan\Downloads\IPL.gif", "teams": []},
                {"title": "Top Bowlers", "description": "Top Wicket Takers", "image_path":
r"C:\Users\Nandan\Downloads\IPL.gif", "teams": []},
            ]

            app = TeamComparisonApp(sections)
            app.mainloop()
```

5.Impactoftoss.py

```
        import pandas as pd
        import plotly.express as px
        import tkinter as tk
        from tkinter import ttk
        import pandas as pd
        from pymongo import MongoClient
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        import plotly
        import plotly.express as px
        import plotly.graph_objs as go
        from plotly.offline import init_notebook_mode, plot, iplot
```

```
from plotly import tools
from warnings import filterwarnings
filterwarnings('ignore')

# MongoDB connection parameters
mongodb_host = 'mongodb://localhost:27017/'  # Change this to your MongoDB host
mongodb_port = 27017  # Change this to your MongoDB port
database_name = 'cricket'

# Connect to MongoDB
client = MongoClient(mongodb_host, mongodb_port)
db = client[database_name]

# Read data from MongoDB collections
deliveries_data_cursor = db.All_scores.find()
match_data_cursor = db.All_matches.find()

# Convert MongoDB cursors to pandas DataFrames
deliveries_data = pd.DataFrame(list(deliveries_data_cursor))
match_data = pd.DataFrame(list(match_data_cursor))

match_data['toss_win_game_win'] = np.where((match_data.toss_winner ==
match_data.winner),'Yes','No')
labels =["Yes",'No']
values = match_data['toss_win_game_win'].value_counts()
colors = ['turquoise', 'crimson']
fig = go.Figure(data=[go.Pie(labels=labels,
                             values=values,hole=.3)])
fig.update_traces(hoverinfo='label+percent', textinfo='label+percent', textfont_size=20,
                  marker=dict(colors=colors, line=dict(color='#000000', width=3)))
fig.update_layout(title="Winning toss implies winning matches?",
                  titlefont={'size': 30},
                  )
fig.show()

labels =["Batting first",'Batting second']
values=match_data['result'].value_counts()
colors = ['turquoise', 'crimson']
fig = go.Figure(data=[go.Pie(labels=labels,
                             values=values,hole=.3)])
fig.update_traces(hoverinfo='label+percent', textinfo='label+percent', textfont_size=20,
                  marker=dict(colors=colors, line=dict(color='#000000', width=3)))
fig.update_layout(title="results based on batting first and second",
                  titlefont={'size': 30},
                  )
fig.show()

6.Luckyvenues.py
import pandas as pd
import plotly.express as px
```

```
import tkinter as tk
from tkinter import ttk
import pandas as pd
from pymongo import MongoClient
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly
import plotly.express as px
import plotly.graph_objs as go
from plotly.offline import init_notebook_mode, plot, iplot
from plotly import tools
from warnings import filterwarnings
import pandas as pd
import plotly.graph_objects as go
import tkinter as tk
from tkinter import ttk
filterwarnings('ignore')

# MongoDB connection parameters
mongodb_host = 'mongodb://localhost:27017/'  # Change this to your MongoDB host
mongodb_port = 27017  # Change this to your MongoDB port
database_name = 'cricket'

# Connect to MongoDB
client = MongoClient(mongodb_host, mongodb_port)
db = client[database_name]

# Read data from MongoDB collections
deliveries_data_cursor = db.All_scores.find()
match_data_cursor = db.All_matches.find()

# Convert MongoDB cursors to pandas DataFrames
deliveries_data = pd.DataFrame(list(deliveries_data_cursor))
match_data = pd.DataFrame(list(match_data_cursor))

def lucky(match_data, team_name):
    return match_data[match_data['winner'] ==
team_name]['venue'].value_counts().nlargest(10)

class LuckyVenueApp(tk.Tk):
    def __init__(self, sections):
        super().__init__()

        self.title("Lucky Venue Analysis")

        # Create a frame for the analysis
        analysis_frame = ttk.Frame(self)
        self.create_analysis_frame(analysis_frame, sections)
```

```
            analysis_frame.pack()

        def create_analysis_frame(self, frame, sections):
            # Add title label
            title_label = ttk.Label(frame, text="Lucky Venue Analysis", font=("Helvetica", 16,
"bold"))
            title_label.pack(pady=(10, 20))

            # Add dropdown list for team selection
            team_label = ttk.Label(frame, text="Select Team:")
            team_label.pack()

            self.team_combobox = ttk.Combobox(frame, values=sections[0]["teams"])
            self.team_combobox.pack()

            # Add button to initiate analysis
            analyze_button = ttk.Button(frame, text="Analyze Lucky Venues",
command=self.analyze_venues)
            analyze_button.pack()

        def analyze_venues(self):
            # Get selected team from dropdown list
            team_name = self.team_combobox.get()

            # Check if a team is selected
            if not team_name:
                tk.messagebox.showinfo("Error", "Please select a team.")
                return

            # Perform analysis
            lucky_venues = lucky(match_data, team_name)

            # Plot the analysis using Plotly
            values = lucky_venues
            labels = lucky_venues.index
            colors = ['turquoise', 'crimson']
            fig = go.Figure(data=[go.Pie(labels=labels, values=values, hole=.3)])
            fig.update_traces(hoverinfo='label+percent', textinfo='value', textfont_size=20,
                              marker=dict(colors=colors, line=dict(color='#000000', width=3)))
            fig.update_layout(title=f"Wins at different Venues for {team_name}:",
                              titlefont={'size': 30})
            fig.show()

    # Example usage
    if __name__ == "__main__":
        # Pass the sections with team names to the LuckyVenueApp
        sections = [
            {"title": "Head to Head", "description": "Head to Head", "image_path":
r"C:\Users\Nandan\Downloads\IPL.gif", "teams": ["Royal Challengers Bangalore", "Kings
XI Punjab", "Delhi Daredevils", "Mumbai Indians", "Kolkata Knight Riders", "Rajasthan
```

```
Royals", "Deccan Chargers", "Chennai Super Kings", "Kochi Tuskers Kerala", "Pune
Warriors", "Sunrisers Hyderabad", "Gujarat Lions", "Rising Pune Supergiants", "Delhi
Capitals"]},
            {"title": "Lucky Venue", "description": "Lucky Venue", "image_path":
r"C:\Users\Nandan\Downloads\IPL.gif", "teams": ["Royal Challengers Bangalore", "Kings
XI Punjab", "Delhi Daredevils", "Mumbai Indians", "Kolkata Knight Riders", "Rajasthan
Royals", "Deccan Chargers", "Chennai Super Kings", "Kochi Tuskers Kerala", "Pune
Warriors", "Sunrisers Hyderabad", "Gujarat Lions", "Rising Pune Supergiants", "Delhi
Capitals"]},
            {"title": "Impact of toss", "description": "Impact of Toss", "image_path":
r"C:\Users\Nandan\Downloads\IPL.gif", "teams": []},
            {"title": "Top Batsmens", "description": "Top runs scorer", "image_path":
r"C:\Users\Nandan\Downloads\IPL.gif", "teams": []},
            {"title": "Top Bowlers", "description": "Top Wicket Takers", "image_path":
r"C:\Users\Nandan\Downloads\IPL.gif", "teams": []},
        ]

        app = LuckyVenueApp(sections)
        app.mainloop()
```

7.Topbatsmen.py

```
import pandas as pd
import plotly.express as px
import tkinter as tk
from tkinter import ttk
import pandas as pd
from pymongo import MongoClient
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly
import plotly.express as px
import plotly.graph_objs as go
from plotly.offline import init_notebook_mode, plot, iplot
from plotly import tools
from warnings import filterwarnings
filterwarnings('ignore')

# MongoDB connection parameters
mongodb_host = 'mongodb://localhost:27017/'  # Change this to your MongoDB host
mongodb_port = 27017  # Change this to your MongoDB port
database_name = 'cricket'

# Connect to MongoDB
client = MongoClient(mongodb_host, mongodb_port)
db = client[database_name]

# Read data from MongoDB collections
deliveries_data_cursor = db.All_scores.find()
```

```
match_data_cursor = db.All_matches.find()

# Convert MongoDB cursors to pandas DataFrames
deliveries_data = pd.DataFrame(list(deliveries_data_cursor))
match_data = pd.DataFrame(list(match_data_cursor))

runs=deliveries_data.groupby(['batsman'])['batsman_runs'].sum().reset_index()
runs.columns=['Batsman','runs']
y=runs.sort_values(by='runs',ascending=False).head(10).reset_index().drop('index',axis=
1)
y.style.background_gradient(cmap='PuBu')

colors = ['turquoise',] * 13
colors[0] = 'crimson'
fig=px.bar(x=y['Batsman'],y=y['runs'],labels=dict(x="Player",y="Total Runs"),)
fig.update_layout(title="Top 10 leading run-scrorer",
                  titlefont={'size': 26},template='simple_white'
                  )
fig.update_traces(marker_line_color='black',
                  marker_line_width=2.5, opacity=1,marker_color=colors)
fig.show()

8.TopBowler.py
import pandas as pd
import plotly.express as px
import tkinter as tk
from tkinter import ttk
import pandas as pd
from pymongo import MongoClient
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly
import plotly.express as px
import plotly.graph_objs as go
from plotly.offline import init_notebook_mode, plot, iplot
from plotly import tools
from warnings import filterwarnings
filterwarnings('ignore')

# MongoDB connection parameters
mongodb_host = 'mongodb://localhost:27017/'  # Change this to your MongoDB host
mongodb_port = 27017  # Change this to your MongoDB port
database_name = 'cricket'

# Connect to MongoDB
client = MongoClient(mongodb_host, mongodb_port)
db = client[database_name]
```

```
# Read data from MongoDB collections
deliveries_data_cursor = db.All_scores.find()
match_data_cursor = db.All_matches.find()

# Convert MongoDB cursors to pandas DataFrames
deliveries_data = pd.DataFrame(list(deliveries_data_cursor))
match_data = pd.DataFrame(list(match_data_cursor))

deliveries_data['dismissal_kind'].unique()
dismissal_kinds = ['caught', 'bowled', 'lbw', 'caught and bowled',
      'stumped', 'hit wicket']
hwt=deliveries_data[deliveries_data["dismissal_kind"].isin(dismissal_kinds)]
bo=hwt['bowler'].value_counts()
colors = ['turquoise',] * 13
colors[0] = 'crimson'
fig=px.bar(x=bo[:10].index,y=bo[:10],labels=dict(x="Bowler",y="Total Wickets"),)
fig.update_layout(title="Leading wicket-takers",
                  titlefont={'size': 26},template='simple_white'
                  )
fig.update_traces(marker_line_color='black',
                  marker_line_width=2.5, opacity=1,marker_color=colors)
fig.show()
```

7. TESTING

Testing is a vital part of software development, and it is important to start it as early as possible, and to make testing a part of the process of deciding requirements. To get the most useful perspective on your development project, it is worthwhile devoting some thought to the entire lifecycle including how feedback from users will influence the future of the application. The tools and techniques we've discussed in this book should help your team to be more responsive to changes without extra cost, despite the necessarily wide variety of different development processes. Nevertheless, new tools and process improvements should be adopted gradually, assessing the results after each step.

Testing is part of a lifecycle. The software development lifecycle is one in which you hear of a need, you write some code to fulfill it, and then you check to see whether you have pleased the stakeholders—the users, owners, and other people who have an interest in what the software does. Hopefully they like it, but would also like some additions or changes, so you update or augment your code; and so, the cycle continues, or every few years,

SOFTWARE DEVELOPMENT LIFE CYCLE

Testing is a proxy for the customer. You could conceivably do your testing by releasing it into the wild and waiting for the complaints and compliments to come back. Some companies have been accused of having such a strategy as their business model even before it became fashionable. But overall, the books are better balanced by trying to make sure that the software will satisfy the customer before we hand it over. This software “Drive X(Motor Driving School” is developed using Incremental Model and Spiral Model.

SOFTWARE TESTING TYPES:

1.FUNTIONAL TESTING – This type of testing ignores the internal parts and focus on the output is as per requirement or not. Black-box type testing geared to functional requirements of an application.

They are:

Black box testing – Internal system design is not considered in this type of testing. Tests are based on requirements and functionality.

White box testing – This testing is based on knowledge of the internal logic of an applications code. Also known as Glass box Testing. Internal software and code working should

be known for this type of testing. Tests are based on coverage of code statements, branches, paths, conditions.

Grey box testing – Also called Grey box analysis, is a strategy for software debugging in which the tester has limited knowledge of the internal details of the program. A Grey box is a device, program or system whose workings are partially understood.

Unit testing – Testing of individual software components or modules. Typically done by the programmer and not by testers, as it requires detailed knowledge of the internal program design and code. May require developing test drive modules or test harnesses.

Incremental integration testing – Bottom up approach for testing i.e. continuous testing of an application as new functionality is added; Application functionality and modules should be independent enough to test separately. Done by programmers or by testers.

System testing – Entire system is tested as per the requirements. Black-box type testing that is based on overall requirements specifications, covers all combined parts of a system.

Acceptance testing -Normally this type of testing is done to verify if system meets the customer specified requirements. User or customers do this testing to determine whether to accept application.

Alpha testing – In house virtual user environment can be created for this type of testing. Testing is done at the end of development. Still minor design changes may be made as a result of such testing.

2. NON-FUNCTIONAL TESTING

Security testing – Can system be penetrated by any hacking way. Testing how well the system protects against unauthorized internal or external access. Checked if system, database is safe from external attacks.

Performance testing – Term often used interchangeably with ‘stress’ and ‘load’ testing.

To check whether system meets performance requirements. Used different performance and load tools to do this.

Usability testing – User-friendliness check. Application flow is tested, can new user understand the application easily, Proper help documented whenever user stuck at any point, basically system navigation is checked in this testing.

Test Cases:

Sl.no	Test id	Form	Test Description	Step to execute	Test Data	Expected Result	Actual Result	Status
1	T 01	Login Form	Username checking	Type User ID	101	Login successful	Login successful	Pass
2	T 02	Login Form	Username checking	Type User ID	kkk	Login Failed	Login Failed	Pass
3	T 03	Login Form	Password checking	Type password	123	Login successful	Login successful	Pass
4	T 04	Login Form	Password Checking	Type password	kkk	Login Failed	Login Failed	Pass
5	T 05	Create User Form	Empty Fields	Leave Text field empty		Cannot insert in Database	Cannot insert in Database	Pass
6	T 06	Create User Form	Phone no field checking	Type phone number	987654321	Successfully inserted	Successfully inserted	Pass
7	T 07	Create User Form	Phone no field checking	Type phone number	98765432109	Cannot insert in Database	Cannot insert in Database	Pass
8	T 08	Update User form	Updating user details	User details insert check	Upload existing customer details	Cannot insert in Database	Cannot insert in Database	Pass

8. IMPLEMENTATION

The Cricket Data Analysis project can be implemented in various settings where there is a need for analyzing cricket match data, particularly focusing on IPL (Indian Premier League) matches. Here are some potential implementation scenarios:

- Sports Analytics Companies:

Companies specializing in sports analytics can utilize this project to provide insights and analysis services to cricket fans, teams, and sponsors.

They can integrate this tool into their existing platforms or offer it as a standalone product for cricket enthusiasts.

- Media Outlets and Broadcasting Companies:

Media outlets covering cricket matches, especially those broadcasting IPL matches, can incorporate this project to enhance their coverage.

They can use the analysis results to provide deeper insights into match statistics, player performances, and historical trends, enriching the viewing experience for their audience.

- Fantasy Cricket Platforms:

Fantasy cricket platforms can leverage this project to provide valuable statistics and insights to their users for making informed decisions while creating fantasy teams.

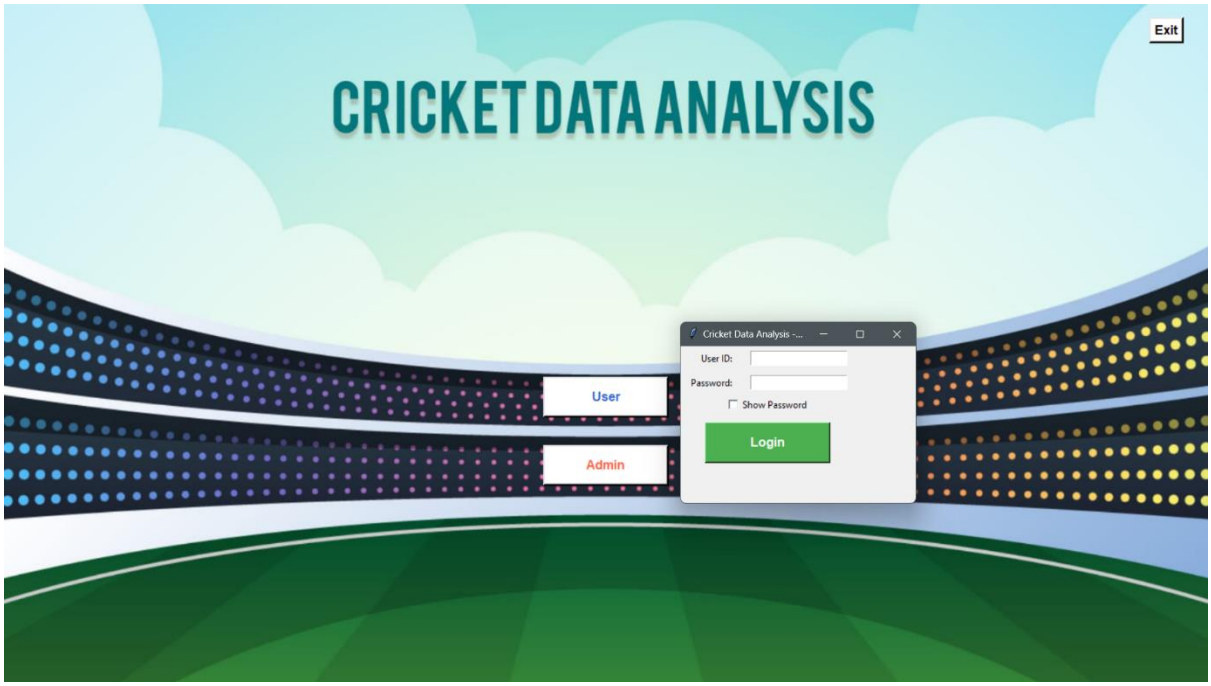
Analysis modules such as top run scorers, highest wicket-takers, and lucky venues can help fantasy cricket players strategize effectively.

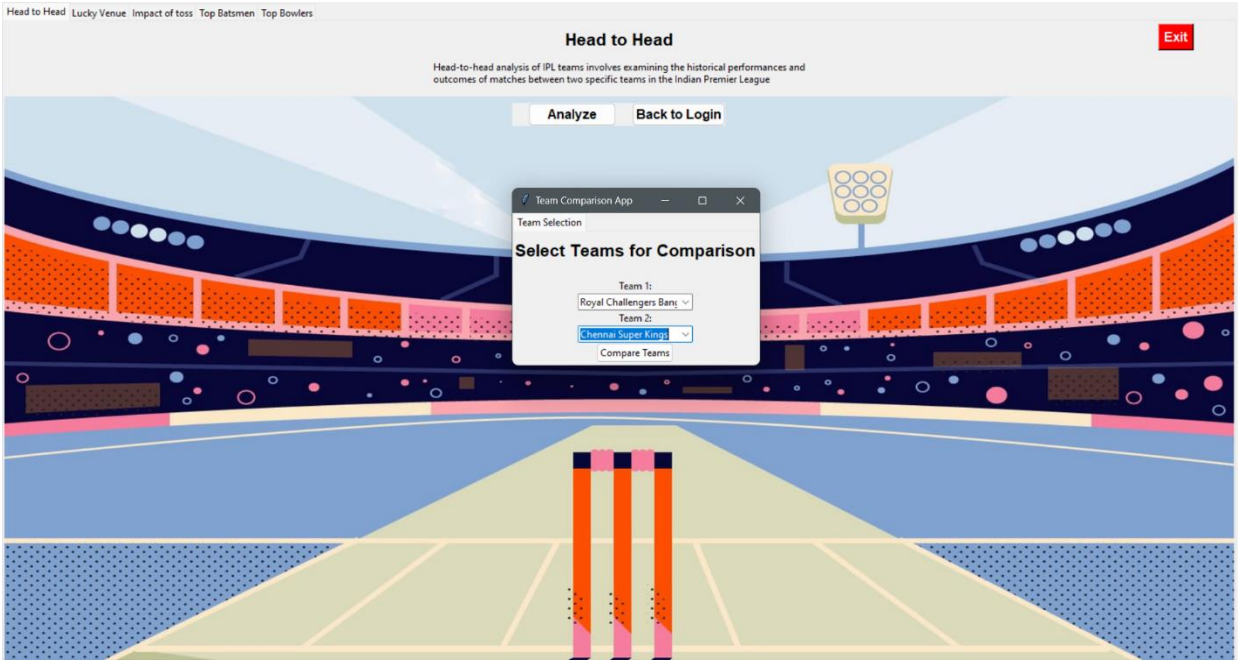
- Cricket Teams and Coaches:

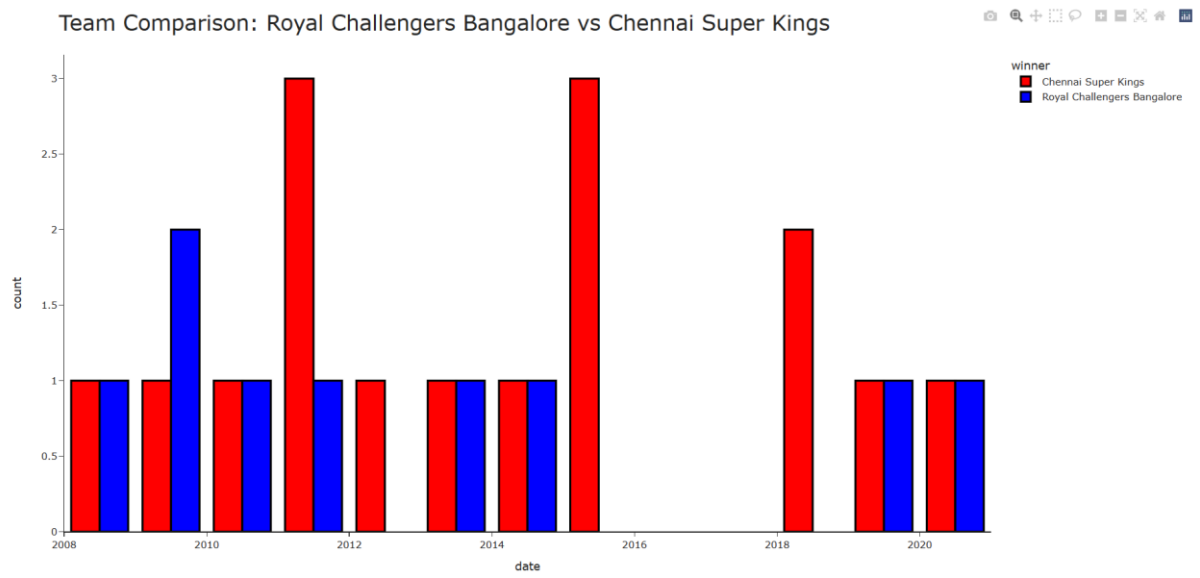
Cricket teams and coaches can use this project for performance analysis, opponent scouting, and strategic planning.

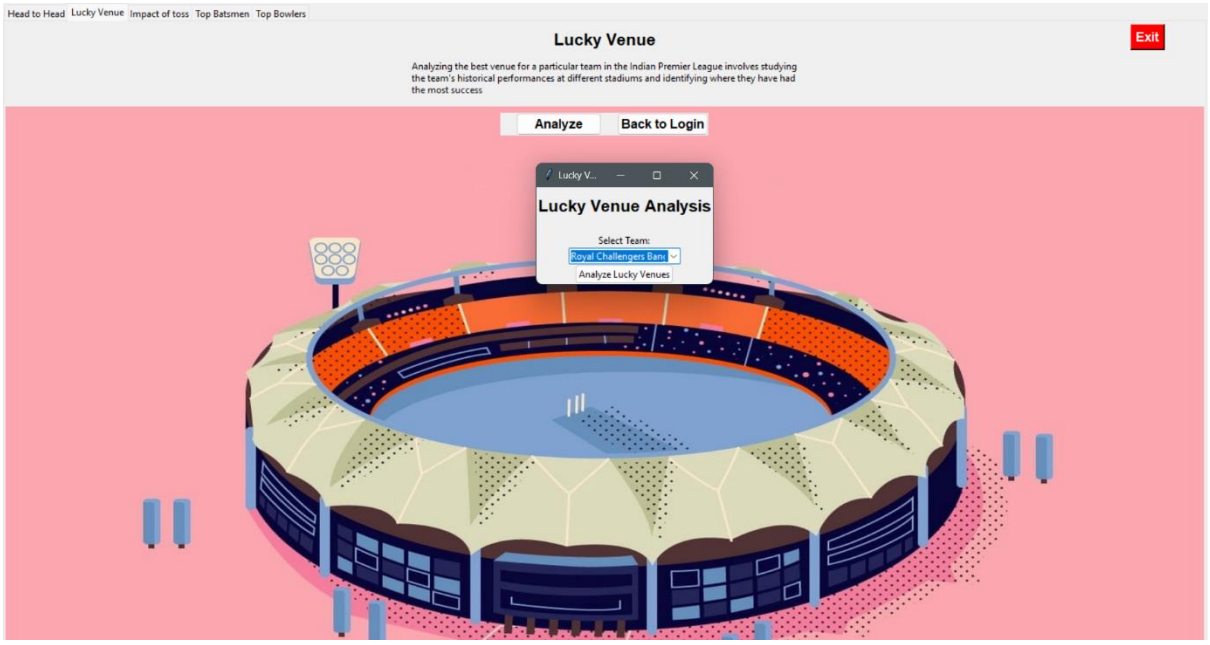
Insights from head-to-head analysis, lucky venues, and toss decision impact can assist teams in optimizing their game plans and identifying areas for improvement.

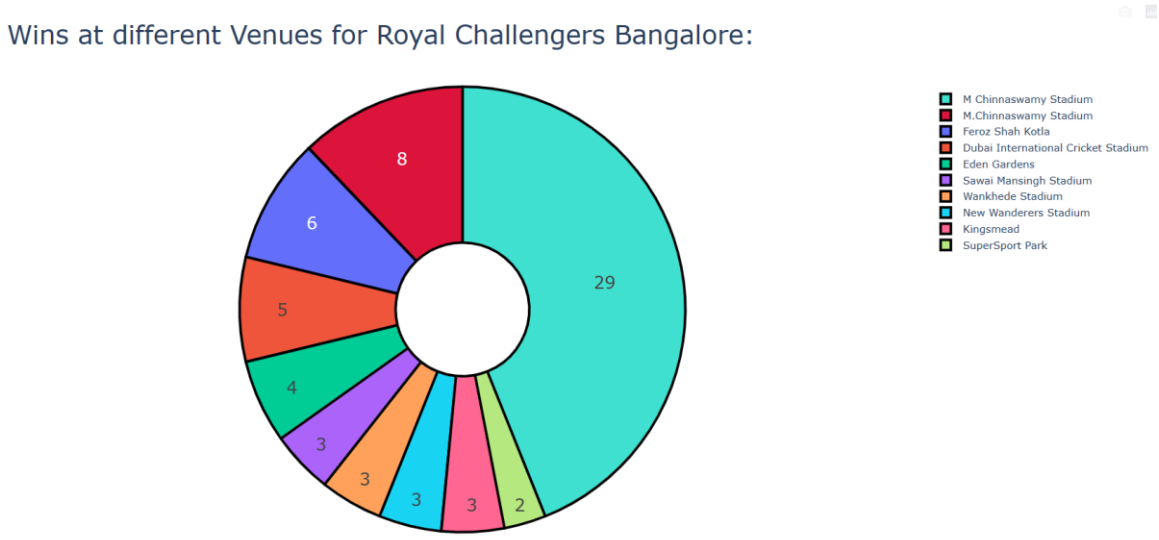
9. SCREENSHOTS

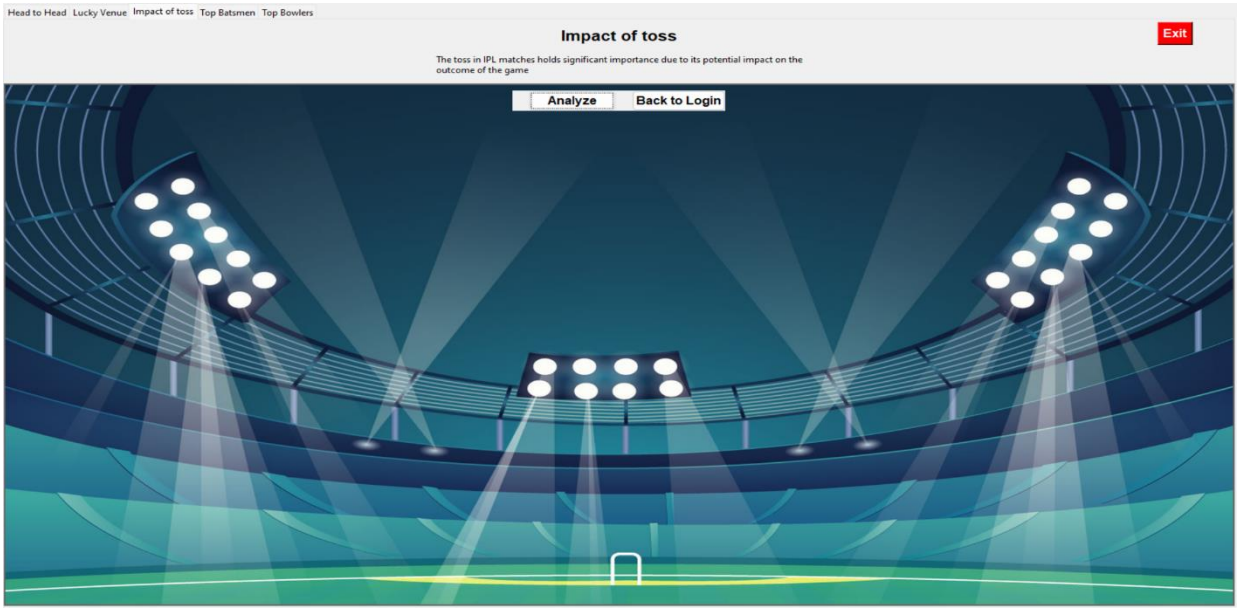


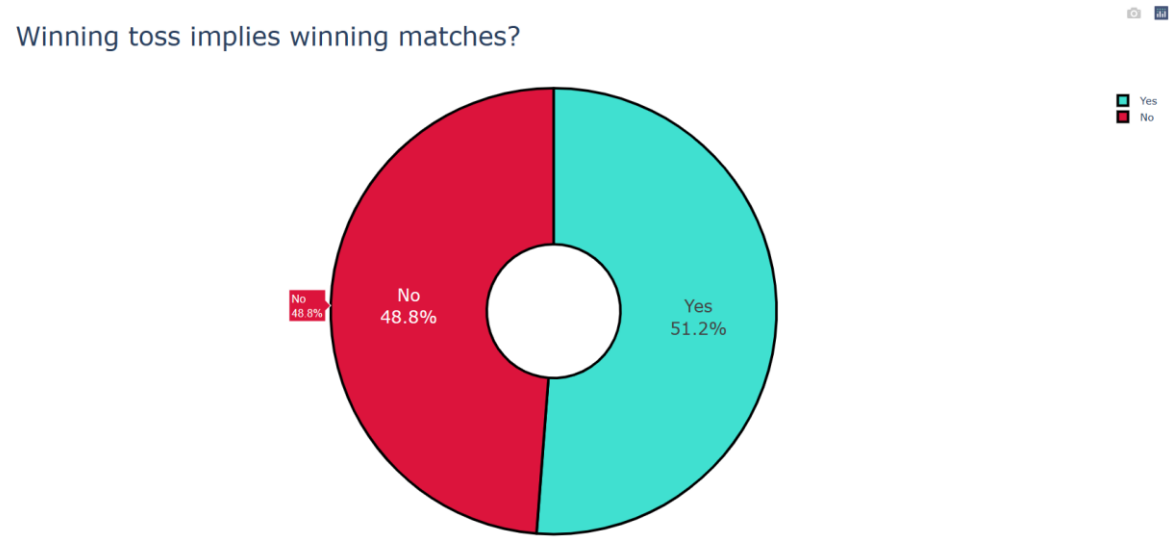


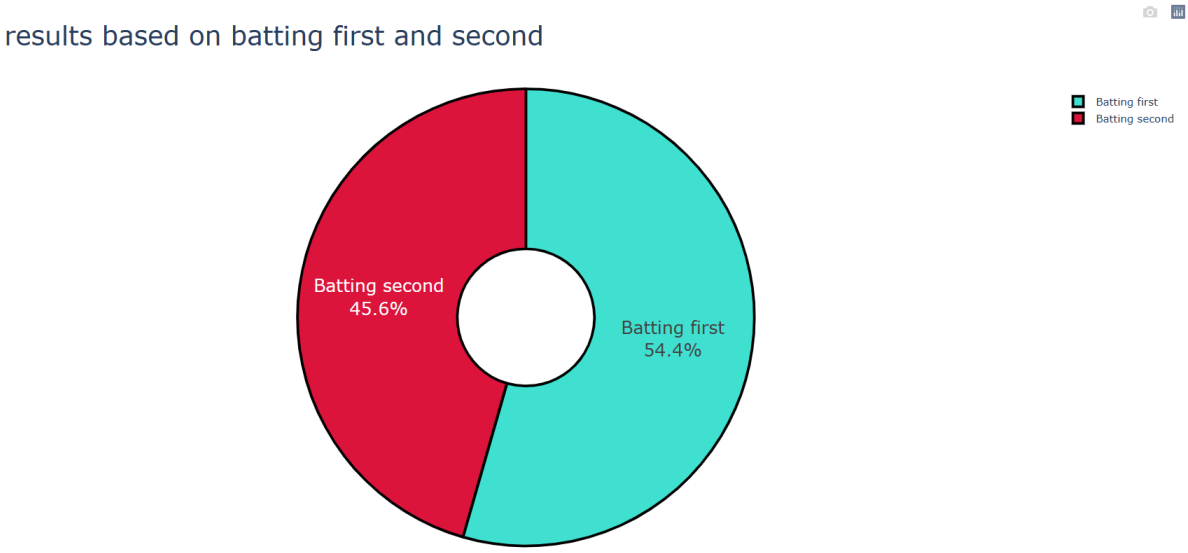

Team Comparison: Royal Challengers Bangalore vs Chennai Super Kings
winner
Chennai Super Kings
Royal Challengers Bangalore
count
date
0
0.5
1
1.5
2
2.5
3
2008
2010
2012
2014
2016
2018
2020

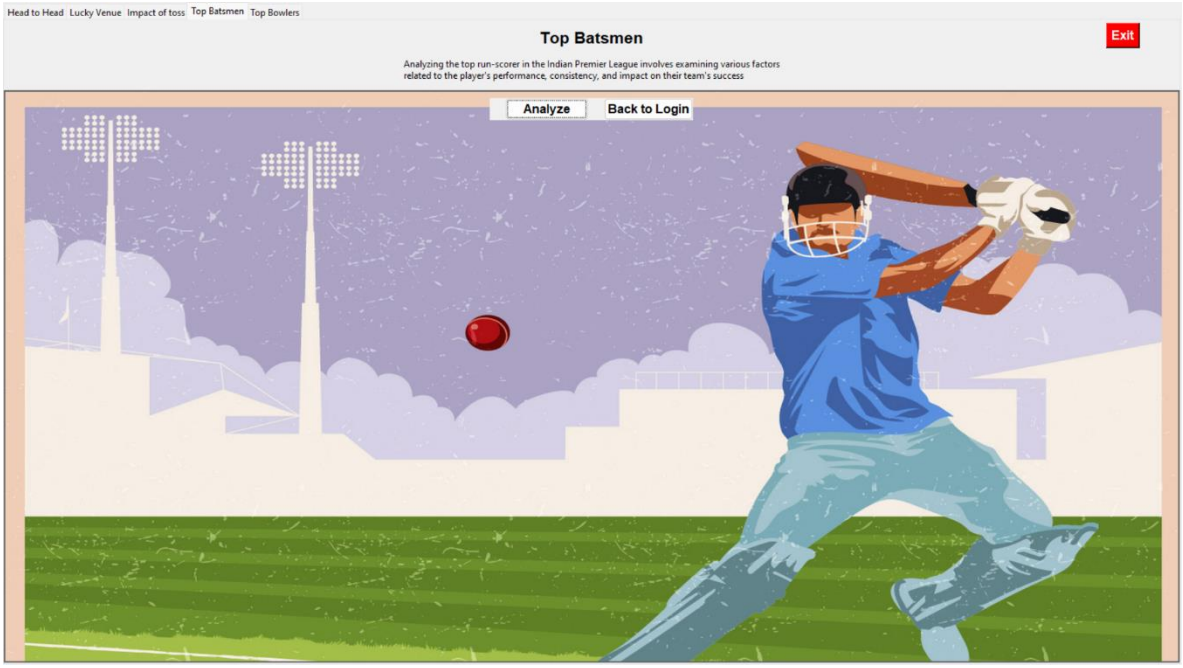

Head to Head
Lucky Venue
Impact of toss
Top Batsmen
Top Bowlers
Lucky Venue
Exit
Analyzing the best venue for a particular team in the Indian Premier League involves studying the team's historical performances at different stadiums and identifying where they have had the most success
Analyze
Back to Login
Lucky V...
Lucky Venue Analysis
Select Team:
Royal Challengers Banl
Analyze Lucky Venues

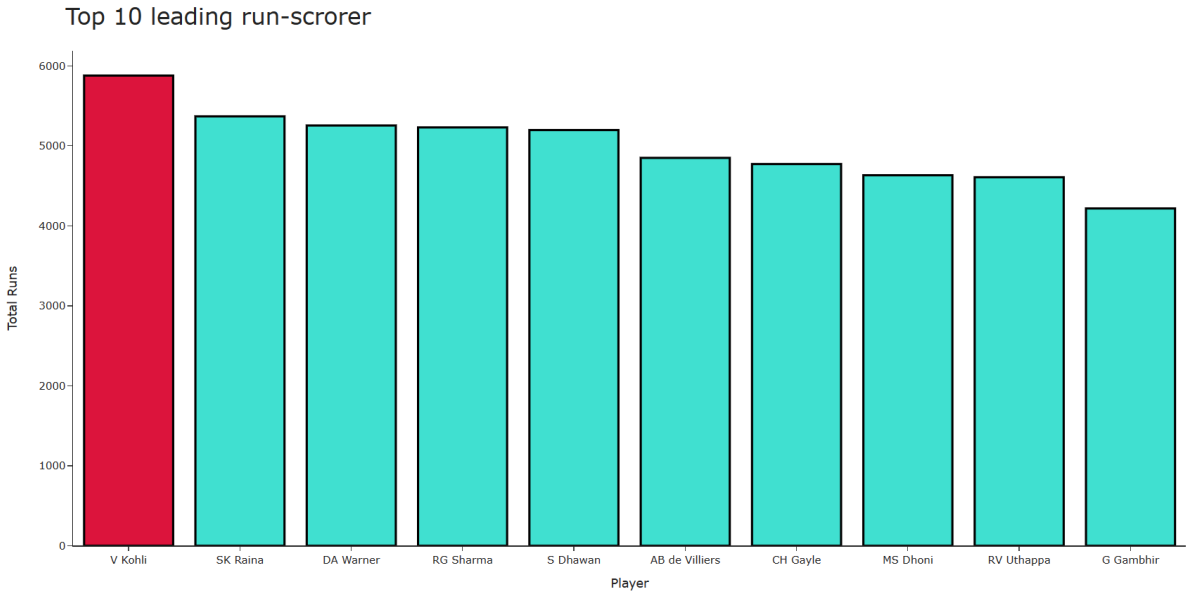

Wins at different Venues for Royal Challengers Bangalore:
8
6
5
4
3
3
3
3
2
29
M Chinnaswamy Stadium
M.Chinnaswamy Stadium
Feroz Shah Kotla
Dubai International Cricket Stadium
Eden Gardens
Sawai Mansingh Stadium
Wankhede Stadium
New Wanderers Stadium
Kingsmead
SuperSport Park

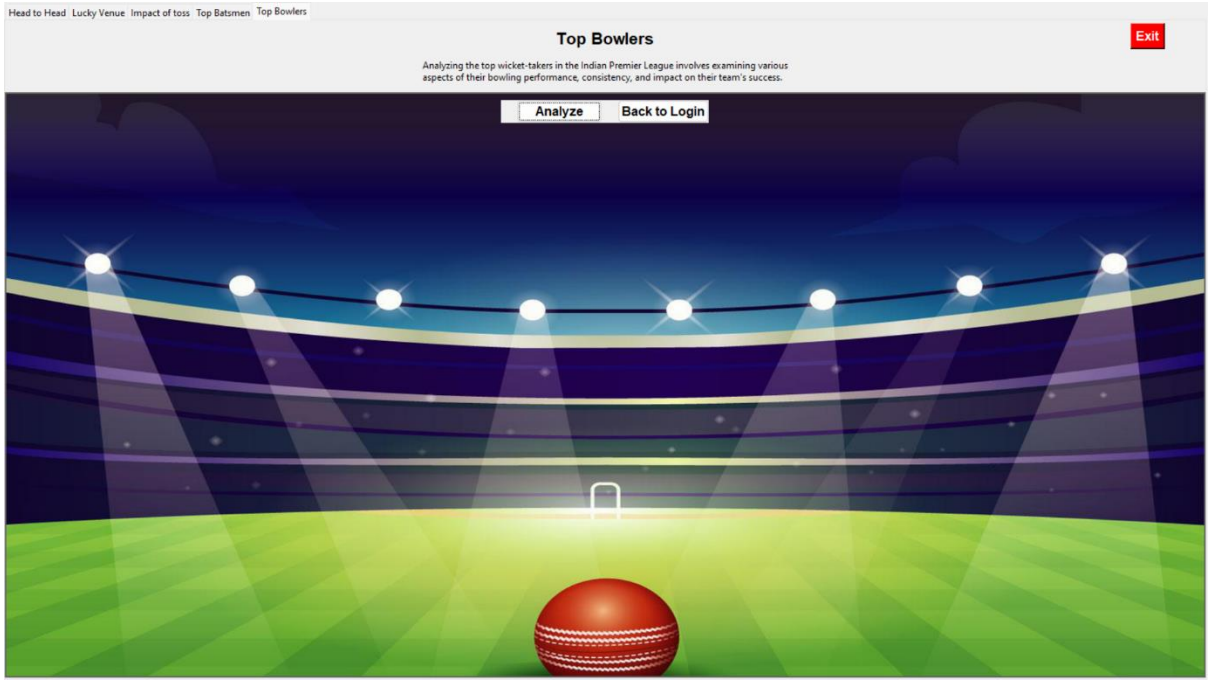

Head to Head
Lucky Venue
Impact of toss
Top Batsmen
Top Bowlers
Impact of toss
Exit
The toss in IPL matches holds significant importance due to its potential impact on the outcome of the game
Analyze
Back to Login

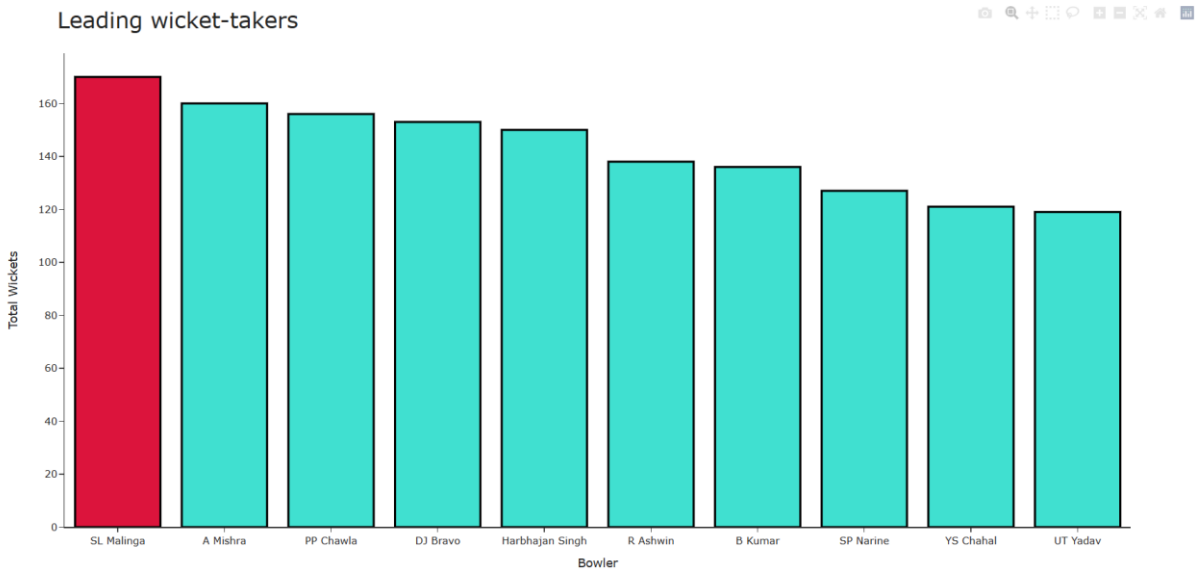

Winning toss implies winning matches?
Yes
No
No
48.8%
No
48.8%
Yes
51.2%


results based on batting first and second
Batting first
Batting second
Batting second
45.6%
Batting first
54.4%


Head to Head
Lucky Venue
Impact of toss
Top Batsmen
Top Bowlers
Top Batsmen
Exit
Analyzing the top run-scorer in the Indian Premier League involves examining various factors related to the player's performance, consistency, and impact on their team's success
Analyze
Back to Login


Top 10 leading run-scorer
Total Runs
6000
5000
4000
3000
2000
1000
0
V Kohli
SK Raina
DA Warner
RG Sharma
S Dhawan
AB de Villiers
CH Gayle
MS Dhoni
RV Uthappa
G Gambhir
Player


Head to Head
Lucky Venue
Impact of toss
Top Batsmen
Top Bowlers
Top Bowlers
Exit
Analyzing the top wicket-takers in the Indian Premier League involves examining various aspects of their bowling performance, consistency, and impact on their team's success.
Analyze
Back to Login


Leading wicket-takers
Total Wickets
0
20
40
60
80
100
120
140
160
SL Malinga
A Mishra
PP Chawla
DJ Bravo
Harbhajan Singh
R Ashwin
B Kumar
SP Narine
YS Chahal
UT Yadav
Bowler

10. CONCLUSION

The "Cricket Data Analysis" project serves as a comprehensive bridge between cricket enthusiasts and insightful IPL data analysis. The combination of a user-friendly frontend and a robust MongoDB backend ensures a seamless and efficient user experience. This synopsis offers a brief yet comprehensive overview of the system's components, functionalities, and its significant role in revolutionizing IPL data analysis.

11.BIBLIOGRAPHY

- "Data Science for Sports" by Mark McClure - Techniques: Statistical Analysis, Data Visualization
- "Python for Data Analysis" by Wes McKinney - Techniques: Data Analysis with Python
- "MongoDB: The Definitive Guide" by Kristina Chodorow - Techniques: MongoDB Database Management
- "Sports Analytics: A Guide for Coaches, Managers, and Other Decision Makers" by Benjamin C. Alamar - Techniques: Sports Analytics
- "R in Action" by Robert I. Kabacoff - Techniques: Data Analysis with R
- "Python Machine Learning" by Sebastian Raschka - Techniques: Machine Learning for Analysis
- "Cricket and Data Science" by Sai Kumar S - Techniques: Data Science in Cricket
- "Data Science from Scratch" by Joel Grus - Techniques: Basics of Data Science
- "The Art of Data Science" by Roger D. Peng and Elizabeth Matsui - Techniques: Data Science Strategies
- "Data Science for Business" by Foster Provost and Tom Fawcett - Techniques: Business Applications of Data Science