

**PIMPRI CHINCHWAD COLLEGE OF ENGINEERING NIGDI , PUNE -411044**

**DEPARTMENT OF COMPUTER ENGINEERING**

Semester – 3

Academic year 2024-25

**Under the Guidance of Prof. Namrata Gawande**

**Submitted by**

##### Pratik Gaikwad (124B2B028)

##### Krishna Shinde(124B2B035)

##### Mayank Udapurkar(124B2B036)

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **Chapter** | **Content s** | | **Page No.** |
| 1. |  | Introduction |  |
|  | *a.* | Problem Statement | 2 |
|  | b. | Project Idea | 2 |
|  | c. | Scope | 3 |
|  |  |  |  |
| 2. |  | Dataset Collection |  |
|  | a. | Data Overview | 3 |
|  | b. | Data Attributes | 3 |
|  | c. | Data Source | 5 |
| 3. | d. | Exploratory Data Analysis and Data Visualization | 5 |
|  | a. | Outliers’ Treatment | 15 |
|  | b. | Univariate and Bivariate Analysis | 28 |
|  |  |  |  |
| 4 |  | Conclusion | 34 |
| 5. |  | References | 34 |

# Introduction

##### Background :

**Overview of IPL Auctions:** Provide a brief description of the IPL auction process, explaining the significance

of player selection, team strategies, and financial investment in players.

**Objective of the Project**: Your goal is to analyze the 2022-2023 IPL auction data to uncover insights such

as the spending trends of teams, distribution of player prices, the influence of nationality on pricing, and

which types of players (Type) fetched the highest prices.

##### Problem Statement:

The Indian Premier League (IPL) auction is a crucial event in professional cricket, where franchises

bid for players to build their teams for the season. The prices at which players are sold vary

significantly, and understanding the factors influencing these prices can provide valuable insights

into team strategies, player valuation, and market trends.

##### Project Idea/Objectives

The main objectives of this project are:

 **Price Distribution Analysis**: Investigate the distribution of player prices to understand the auction

dynamics and identify trends (e.g., how many players were bought at higher or lower prices).

 **Team Spending Behavior**: Compare the total spending by each team, identifying which franchises

invested the most and how their spending correlates with the types of players they purchased.

 **Player Type Analysis**: Examine the correlation between player roles (batsman, bowler, all-rounder)

and their auction prices, and determine if certain roles are valued higher than others.

 **Nationality Trends**: Analyze the distribution of players based on nationality (Indian vs. foreign players)

and their price differences.

1. **Scope :**

This project aims to provide a comprehensive analysis of the IPL 2022-2023 auction, helping teams

and analysts understand the factors that drive player prices and spending patterns. The insights derived

from this project could contribute to more informed decision-making in future IPL auctions, particularly

in optimizing player acquisition strategies based on player roles, age, and nationality.

2: Data Collection

##### Dataset Information

The dataset contains auction data for the IPL 2022-2023 season, detailing the players sold during the

auction. It includes key fields such as Name (player's name), Team (the IPL franchise that bought the

player), Price (the auction price in crores), Nationality (the player's country of origin), and Type (the

player's role, such as batsman, bowler, or all-rounder). This dataset allows for analysis of player pricing

trends, team spending behavior, the impact of nationality and player roles on pricing, and the distribution

of player acquisition across various teams and types. The data helps to identify patterns in the IPL auction

and understand how franchises allocate their resources.

##### Dataset Overview:

The IPL 2022-2023 Auction Dataset provides detailed information about the players who were sold

during the IPL 2022-2023 auction, including their auction prices, teams, and roles. This dataset includes

the following key columns:

* season: The IPL season (2022-2023).
* Name: The name of the player bought at the auction.
* Nationality: The country of origin of the player (e.g., India, Australia, South Africa).
* Type: The player's role or type, such as Batsman, Bowler, or All-rounder.
* Team: The IPL franchise that purchased the player (e.g., Mumbai Indians, Chennai Super Kings).
* Price: The auction price of the player, generally recorded in crores (1 crore = ₹10 million).

This dataset enables an in-depth analysis of player acquisition strategies by IPL teams, including trends

in player pricing, team spending behavior, the influence of nationality and player roles on pricing, and

the distribution of players by team and type. By analyzing this data, we can gain valuable insights into how

teams prioritize their bids and what factors contribute most to a player’s auction price.

##### Dataset Attributes:

### 1.season

* **Description**: This field indicates the IPL season for which the auction data is relevant. In your case, it will primarily contain the value **2022-2023** to specify that the data pertains to the IPL 2023-2024 auction.
* **Type**: Categorical/String (e.g., '2023-2024')
* **Purpose**: Helps to filter the dataset by season, especially if you have data for multiple seasons.

### ****2.**** Name

* **Description**: The name of the player who was sold in the IPL 2023-2024 auction.
* **Type**: String (e.g., 'Shubman Gill', 'Kieron Pollard')
* **Purpose**: Identifies the individual player who was bought during the auction. It can be used to search specific players or compare the prices of multiple players.

### ****3.**** Nationality

* **Description**: The country of origin of the player (e.g., India, Australia, South Africa, etc.).
* **Type**: Categorical/String (e.g., 'India', 'Australia', 'South Africa')
* **Purpose**: Useful for analyzing the nationality distribution of players and determining how nationality affects pricing trends.

### ****4.**** Type

* **Description**: This field represents the type of player. It categorizes players based on their roles in cricket:
  + **Batsman**: A player whose primary role is batting.
  + **Bowler**: A player whose primary role is bowling.
  + **All-rounder**: A player who can both bat and bowl effectively.
* **Type**: Categorical/String (e.g., 'Batsman', 'Bowler', 'All-rounder')
* **Purpose**: To understand how player roles influence auction prices and the distribution of players across teams.

### ****5.**** Team

* **Description**: The IPL franchise that bought the player during the auction (e.g., 'Mumbai Indians',
* 'Chennai Super Kings', 'Delhi Capitals').
* **Type**: Categorical/String (e.g., 'Mumbai Indians', 'Kolkata Knight Riders')
* **Purpose**: To analyze spending patterns of different teams, team-specific strategies, and compare how much
* each team invested in player acquisition.

### ****6.**** Price

* **Description**: The auction price of the player, typically recorded in **Indian Rupees (₹)** and often shown in
* **Crores** (1 Crore = ₹10 million).
* **Type**: Numeric (e.g., 8.0, 12.5, 4.5)
* **Purpose**: The main variable for analysis, as it is the most significant indicator of the player's value in the
* auction. This will be used for comparisons, visualizations, and price distribution analysis.

##### Data Source

The dataset was sourced from [www.kaggle.com](http://www.kaggle.com/).

This data is publicly accessible and provides a comprehensive view of factors that may affect sleep quality and health, making it valuable for analysis in health and wellness research.

## Exploratory Data Analysis (EDA)

* + **Data Preprocessing:**Data preprocessing is an important step in the data mining process. It refers to the cleaning, transforming, and integrating of data in order to make it ready for analysis. The goal of data preprocessing is to improve the quality of the data and to make it more suitable for the specific data mining task.

**Steps to clean and prepare the dataset (handling missing values, removing outliers, etc.).**

##### PROGRAM:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

 **pandas (pd)**:

Used for data manipulation and analysis. It provides data structures like DataFrames, which allow you to efficiently store and work with data in table format.

 **numpy (np)**:

Provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.

 **matplotlib.pyplot (plt)**:

A plotting library for creating static, interactive, and animated visualizations. particularly useful for creating basic charts like histograms, bar charts, and scatter plots.

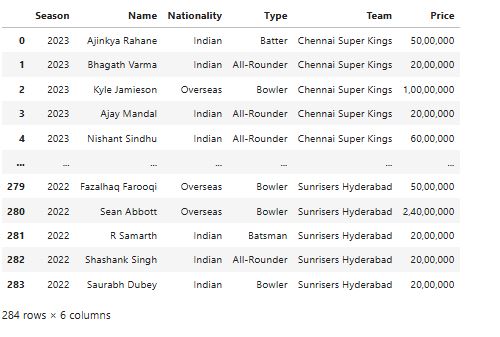
 **seaborn (sns)**:

Built on top of matplotlib, seaborn is a statistical data visualization library that makes it easier to generate more complex plots and improve the aesthetics of your visualizations.

##### PROGRAM:

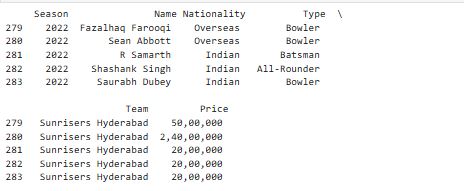
dataset = pd.read\_csv("F:\PCCOE\Sem 3\DEVL\IPL\_2023-22\_Sold\_Players.csv")

dataset

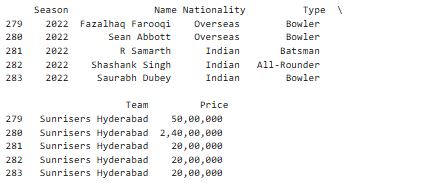


**Data Preprocessing :**

print(dataset.head())



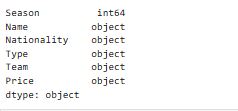
print(dataset.tail())



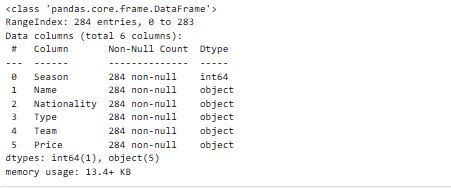
print(dataset.isnull().sum())



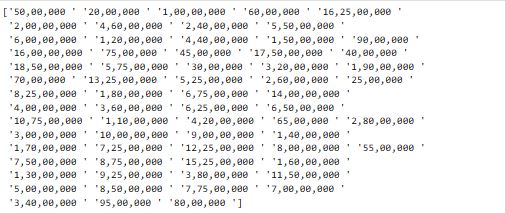
dataset.dtypes



dataset.info()

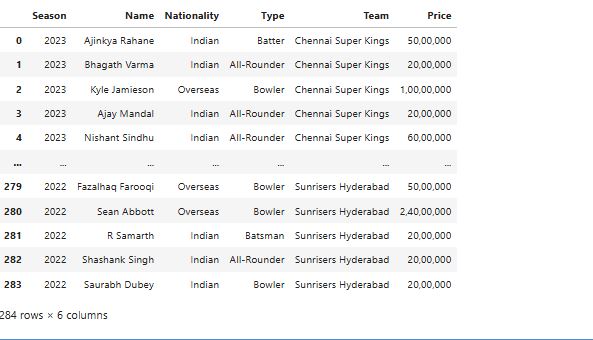


print(dataset['Price'].unique())

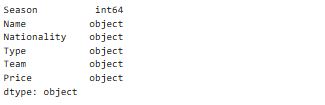


dataset['Price'] = pd.to\_numeric(dataset['Price'], errors='coerce')

dataset



dataset.dtypes

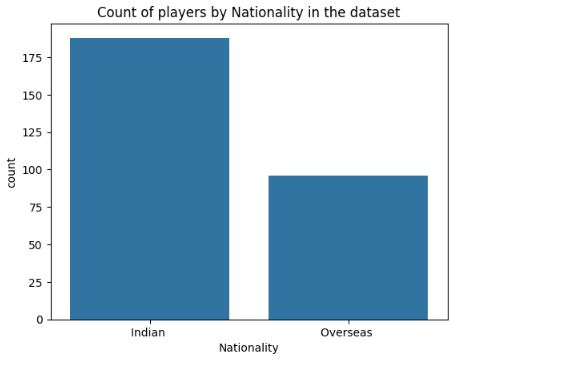




sns.countplot(x='Nationality',data=dataset)

plt.title("Count of players by Nationality in the dataset")

plt.show()



### Explanation:

* sns.countplot(x='Nationality', data=dataset):
  + sns.countplot is used to create a bar plot where each bar represents the count of occurrences of
  + each unique value in the Nationality column.
  + x='Nationality' tells seaborn to plot the counts for each nationality.
  + data=dataset specifies the DataFrame you're using.
* plt.title("Count of players by Nationality in the dataset"):
  + This sets the title of the plot, making it clear that you're visualizing the count of players based on their nationality.
* plt.show():
  + This displays the plot on the screen.

total\_price\_by\_season = dataset.groupby('Season')['Price'].sum()

# Creating the figure

fig = plt.figure(figsize=(10, 6))

# Creating the bar plot

plt.bar(total\_price\_by\_season.index, total\_price\_by\_season.values, color='green', width=0.4)

# Adding labels and title

plt.xlabel("Season")

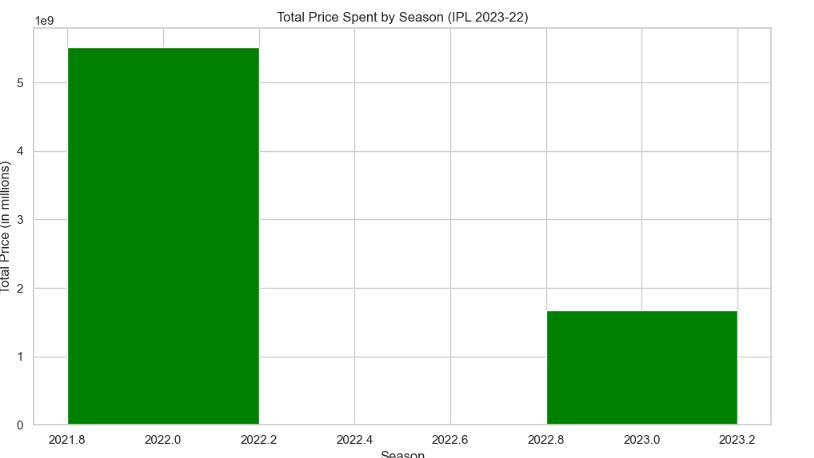
plt.ylabel("Total Price (in millions)")

plt.title("Total Price Spent by Season (IPL 2023-22)")

# Displaying the plot

plt.tight\_layout() # Adjust layout to prevent clipping of tick-labels

plt.show()



**Explanation :**

 dataset.groupby('Season'): This groups your data by the Season column (assuming the season is

in the dataset).

 ['Price'].sum(): For each season, it calculates the sum of the Price column, which represents the

total money spent on players in that season.

 total\_price\_by\_season.index: This is the x-axis (season).

 total\_price\_by\_season.values: This is the y-axis (total price spent).

 color='green': Sets the color of the bars to green.

 width=0.4: Adjusts the width of the bars to make them thinner for better spacing.

team\_counts = dataset['Team'].value\_counts()

# Creating the figure

fig = plt.figure(figsize=(12, 6))

# Creating the bar plot

plt.bar(team\_counts.index, team\_counts.values, color='green', width=0.4)

# Adding labels and title

plt.xlabel("Teams")

plt.ylabel("Number of Players")

plt.title("Number of Players in Each Team (IPL 2023-22)")

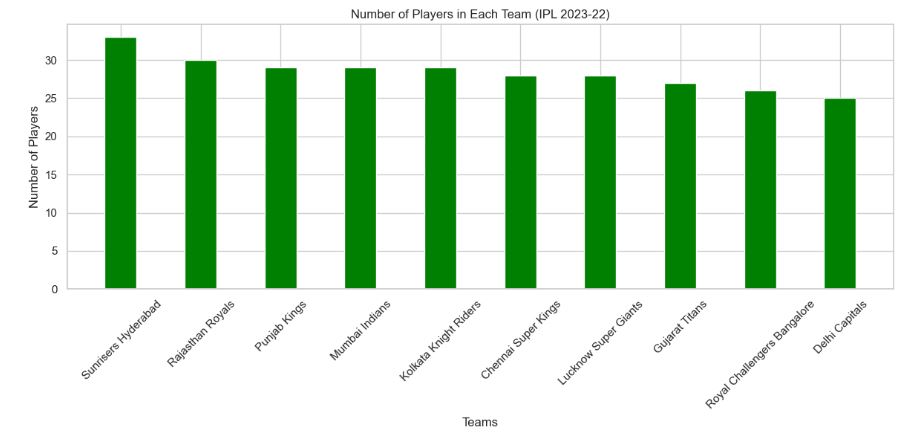
# Rotate team names for better visibility

plt.xticks(rotation=45)

# Displaying the plot

plt.tight\_layout() # Adjust layout to prevent clipping of tick-labels

plt.show()



**Explanation**:

dataset['Team'].value\_counts() counts the number of occurrences of each unique value in the Team column,

which gives you the number of players bought by each team.

 team\_counts.index: The x-axis will represent the names of the IPL teams.

 team\_counts.values: The y-axis will show the number of players each team has bought.

 color='green': This sets the color of the bars to green.

 width=0.4: This adjusts the width of the bars to make them thinner and avoid overlap.

type\_counts = dataset['Type'].value\_counts()

# Creating the figure

fig = plt.figure(figsize=(10, 5))

# Creating the bar plot

plt.bar(type\_counts.index, type\_counts.values, color='blue', width=0.4)

# Adding labels and title

plt.xlabel("Player Type")

plt.ylabel("Number of Players")

plt.title("Number of Players by Type (IPL 2023-22)")

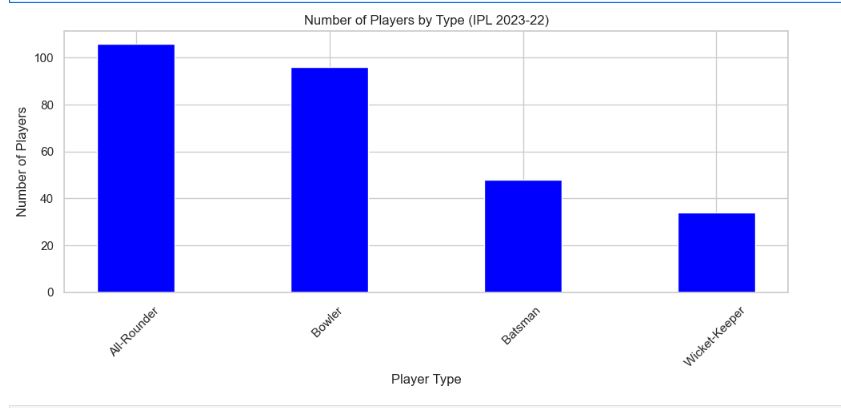
# Rotate type names for better visibility

plt.xticks(rotation=45)

# Displaying the plot

plt.tight\_layout() # Adjust layout to prevent clipping of tick-labels

plt.show()



player\_count = dataset.groupby(['Team', 'Type']).size().reset\_index(name='Player\_Count')

sns.set(style="whitegrid")

plt.figure(figsize=(12, 8))

sns.barplot(data=player\_count, x='Team', y='Player\_Count', hue='Type', errorbar=None)

plt.title('Number of Players by Type in Each Team')

plt.xlabel('Team Name')

plt.ylabel('Number of Players')

plt.xticks(rotation=90)

plt.legend(title='Player Type')

plt.show()

## 

## total\_price\_by\_team = dataset.groupby('Team')['Price'].sum()

## # Creating the figure

## fig = plt.figure(figsize=(12, 6))

## # Creating the bar plot

## plt.bar(total\_price\_by\_team.index, total\_price\_by\_team.values, color='orange', width=0.4)

## # Adding labels and title

## plt.xlabel("Teams")

## plt.ylabel("Total Price (in millions)")

## plt.title("Total Price Spent by Each Team (IPL 2023-22)")

## # Rotate team names for better visibility

## plt.xticks(rotation=45)

## # Displaying the plot

## plt.tight\_layout() # Adjust layout to prevent clipping of tick-labels

## plt.show()

## 

## total\_price\_by\_nationality = dataset.groupby('Nationality')['Price'].sum()

## # Creating the figure

## fig = plt.figure(figsize=(12, 6))

## # Creating the bar plot

## plt.bar(total\_price\_by\_nationality.index, total\_price\_by\_nationality.values, color='purple', width=0.4)

## # Adding labels and title

## plt.xlabel("Nationality")

## plt.ylabel("Total Price (in millions)")

## plt.title("Total Price Spent by Nationality (IPL 2023-22)")

## # Rotate nationality names for better visibility

## plt.xticks(rotation=45)

## # Displaying the plot

## plt.tight\_layout() # Adjust layout to prevent clipping of tick-labels

## plt.show()

## 

## price\_by\_team\_and\_type = dataset.groupby(['Team', 'Type'])['Price'].sum().unstack()

## # Creating the figure

## fig = plt.figure(figsize=(12, 6))

## # Creating the bar plot

## price\_by\_team\_and\_type.plot(kind='bar', stacked=True, ax=fig.add\_subplot(111))

## # Adding labels and title

## plt.xlabel("Teams")

## plt.ylabel("Total Price (in millions)")

## plt.title("Total Price Spent by Team on Each Type (IPL 2023-22)")

## # Rotate team names for better visibility

## plt.xticks(rotation=45)

## # Displaying the plot

## plt.tight\_layout() # Adjust layout to prevent clipping of tick-labels

## plt.legend(title='Player Type')

## plt.show()

## 

## fig = plt.figure(figsize=(10, 6))

## # Creating the histogram

## plt.hist(dataset['Price'], bins=10, color='blue', edgecolor='black')

## # Adding labels and title

## plt.xlabel("Price (in millions)")

## plt.ylabel("Number of Players")

## plt.title("Distribution of Player Prices (IPL 2023-22)")

## # Displaying the plot

## plt.tight\_layout() # Adjust layout to prevent clipping of tick-labels

## plt.show()

## 

## plt.figure(figsize=(10,6))

## sns.boxplot(x=dataset['Price']) # Adjust 'price' with the actual column name if needed

## plt.title('Box Plot of Player Prices')

## plt.xlabel('Price (in Crores)')

## plt.show()

## 

## # Create a box plot of Player Prices by Team

## plt.figure(figsize=(15, 8)) # Increase the figure size if necessary

## sns.boxplot(x='Team', y='Price', data=dataset)

## plt.title('Box Plot of Player Prices by Team')

## plt.xlabel('Team')

## plt.ylabel('Price (in Crores)')

## plt.xticks(rotation=90) # Rotate team names if they overlap

## plt.show()

## 

## import matplotlib.pyplot as plt

## import seaborn as sns

## # Assuming 'auction\_order' is the column representing the order in which players were sold

## # and 'Price' is the column for player prices. Adjust if needed.

## plt.figure(figsize=(12, 6))

## sns.lineplot(x=dataset['Type'], y=dataset['Price'], marker='o') # Adjust column names if necessary

## # Title and labels

## plt.title('Player Prices Over Auction Order')

## plt.xlabel('Auction Order')

## plt.ylabel('Price (in Crores)')

## # Show the plot

## plt.show()

## 

## fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 10))

## axes = axes.flatten() # Flatten the 2D array of axes to 1D

## # List of player types

## player\_types = dataset['Type'].unique()

## # Plotting histograms for each player type

## for i, player\_type in enumerate(player\_types):

## # Filter dataset for the current type

## prices = dataset[dataset['Type'] == player\_type]['Price']

## 

## # Creating the histogram

## axes[i].hist(prices, bins=10, color='blue', edgecolor='black')

## 

## # Adding labels and title

## axes[i].set\_xlabel("Price (in millions)")

## axes[i].set\_ylabel("Number of Players")

## axes[i].set\_title(f"Distribution of Prices for {player\_type}")

## # Adjust layout

## plt.tight\_layout()

## plt.show()

## 

## teams = dataset['Team'].unique()

## num\_teams = len(teams)

## # Calculate the number of rows and columns needed for subplots

## cols = 2

## rows = np.ceil(num\_teams / cols).astype(int)

## # Creating the figure with dynamic subplots

## fig, axes = plt.subplots(nrows=rows, ncols=cols, figsize=(15, 5 \* rows))

## axes = axes.flatten() # Flatten the 2D array of axes to 1D

## # Plotting histograms for each team

## for i, team in enumerate(teams):

## # Filter dataset for the current team

## prices = dataset[dataset['Team'] == team]['Price']

## 

## # Creating the histogram

## axes[i].hist(prices, bins=10, color='blue', edgecolor='black')

## 

## # Adding labels and title

## axes[i].set\_xlabel("Price (in millions)")

## axes[i].set\_ylabel("Number of Players")

## axes[i].set\_title(f"Distribution of Prices for {team}")

## # Hide any unused subplots

## for j in range(i + 1, len(axes)):

## axes[j].axis('off')

## # Adjust layout

## plt.tight\_layout()

## plt.show()

## 

## team\_counts = dataset['Team'].value\_counts()

## # Plot the pie chart

## plt.figure(figsize=(10, 7))

## plt.pie(team\_counts, labels=team\_counts.index, autopct='%1.2f%%', startangle=0)

## plt.title('Distribution of Players by Team\n\n')

## plt.axis('equal')

## 

## average\_prices = dataset.groupby('Type')['Price'].mean().reset\_index()

## # Sort the results for better visualization

## average\_prices.sort\_values(by='Price', ascending=False, inplace=True)

## # Display the average prices

## print("Average Prices by Player Type:\n", average\_prices)

## plt.figure(figsize=(10, 6))

## plt.barh(average\_prices['Type'], average\_prices['Price'], color='lightblue')

## plt.xlabel('Average Price')

## plt.title('Average Price of Players by Type')

## plt.show()

## 

## price\_stats = dataset.groupby('Type')['Price'].agg(['max', 'min']).reset\_index()

## # Rename the columns for clarity

## price\_stats.columns = ['Type', 'Highest\_Price', 'Lowest\_Price']

## # Convert prices to string format with commas

## price\_stats['Highest\_Price'] = price\_stats['Highest\_Price'].astype(str).str.replace('.', '').str.replace('e', '').str.replace(' ', '')

## price\_stats['Lowest\_Price'] = price\_stats['Lowest\_Price'].astype(str).str.replace('.', '').str.replace('e', '').str.replace(' ', '')

## # Melt the DataFrame for easier plotting

## price\_stats\_melted = price\_stats.melt(id\_vars='Type', value\_vars=['Highest\_Price', 'Lowest\_Price'],

## var\_name='Price\_Type', value\_name='Price')

## # Set the visual style of seaborn

## sns.set(style="whitegrid")

## # Create a bar plot

## plt.figure(figsize=(12, 6))

## sns.barplot(data=price\_stats\_melted, x='Type', y='Price', hue='Price\_Type', palette='viridis')

## # Customize the plot

## plt.title('Highest and Lowest Player Prices by Type')

## plt.xlabel('Player Type')

## plt.ylabel('Price')

## plt.xticks(rotation=45)

## plt.legend(title='Price Type')

## plt.tight\_layout()

## # Show the plot

## plt.show()

## 

## pip install scikit-learn

## Once the installation is complete, you can verify that scikit-learn has been installed correctly by running the following command in a Python shell:

## import pandas as pd

## from sklearn.model\_selection import train\_test\_split

## from sklearn.linear\_model import LinearRegression

## from sklearn.metrics import mean\_squared\_error

## import numpy as np

## # read\_csv() is an import pandas function

## data=pd.read\_csv('F:\PCCOE\Sem 3\DEVL\IPL\_2023-22\_Sold\_Players.csv')

## print(data)

## 

## data.fillna(data.mean(), inplace=True)

## print(data)

## 

## data = pd.get\_dummies(data)

### Explanation:

### pd.get\_dummies(data): This function automatically identifies categorical columns in the DataFrame (data) and converts them into numerical columns, with each category represented as a binary (0 or 1) column.

* + For each unique category in a categorical column, a new binary column is created.
  + If the original value matches the category, the column will have a 1; otherwise, it will have a 0.

This is a common step in preparing data for machine learning models, as most models expect numerical input categorical data.

## import pandas as pd

## from sklearn.model\_selection import train\_test\_split

## # Load your dataset (adjust path if needed)

## dataset = pd.read\_csv("F:/PCCOE/Sem 3/DEVL/IPL\_2023-22\_Sold\_Players.csv")

## # Drop any non-numeric columns if needed, or choose specific features

## # You can use 'Price' or any other numerical columns to create the split

## # If you want to use all features to train a model, make sure to separate X (features) and y (target)

## # Example: Splitting on features and target variable

## X = dataset.drop(columns=['Price']) # Features (everything except 'Price')

## y = dataset['Price'] # Target variable (Price)

## # Split the entire dataset into training and testing

## train\_data, test\_data = train\_test\_split(dataset, test\_size=0.2, random\_state=42)

## # Check the shape of the resulting datasets

## print(f"Training data size: {train\_data.shape[0]} rows")

## print(f"Testing data size: {test\_data.shape[0]} rows")

### Explanation:

train\_test\_split() splits the entire dataset into training and testing subsets.

* dataset: The DataFrame to be split.
* test\_size=0.2: 20% of the data will be used for testing, and 80% will be used for training.
* random\_state=42: This ensures reproducibility. If you run the code multiple times, you'll always get the same random split.
* train\_data.shape[0] gives the number of rows (samples) in the training dataset.
* test\_data.shape[0] gives the number of rows (samples) in the testing dataset.



## import matplotlib.pyplot as plt

## # Plot actual vs predicted prices

## plt.figure(figsize=(10, 6))

## plt.scatter(y\_test, y\_pred, alpha=0.6, color='b')

## plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=2) # 45-degree line for reference

## plt.xlabel("Actual Price")

## plt.ylabel("Predicted Price")

## plt.title("Actual vs. Predicted Prices")

## plt.show()

## 

## from sklearn.model\_selection import learning\_curve

## import numpy as np

## train\_sizes, train\_scores, test\_scores = learning\_curve(

## model, X, y, cv=5, scoring='neg\_mean\_squared\_error', n\_jobs=-1,

## train\_sizes=np.linspace(0.1, 1.0, 10)

## )

## # Calculate average and std deviation of training and test scores

## train\_scores\_mean = -train\_scores.mean(axis=1)

## test\_scores\_mean = -test\_scores.mean(axis=1)

## # Plot learning curve

## plt.figure(figsize=(10, 6))

## plt.plot(train\_sizes, train\_scores\_mean, 'o-', color='r', label='Training Error')

## plt.plot(train\_sizes, test\_scores\_mean, 'o-', color='g', label='Validation Error')

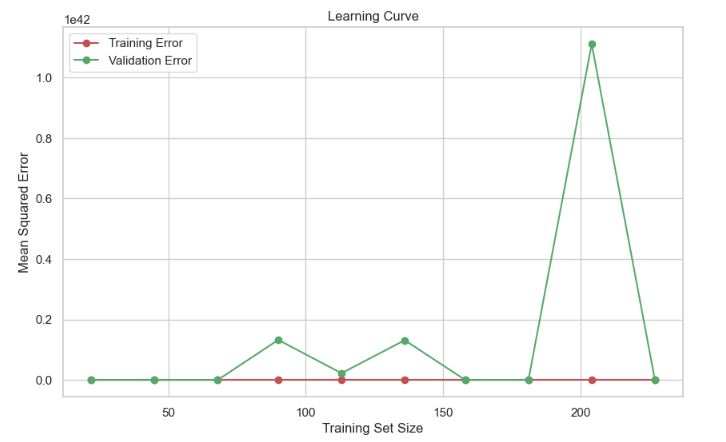
## plt.xlabel("Training Set Size")

## plt.ylabel("Mean Squared Error")

## plt.title("Learning Curve")

## plt.legend(loc="best")

## plt.show()

dataset = pd.get\_dummies(dataset, columns=['Nationality', 'Type', 'Team']) dataset = pd.get\_dummies(dataset, columns=['Nationality', 'Type', 'Team'])

dataset = pd.get\_dummies(dataset, columns=['Nationality', 'Type', 'Team'])

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, confusion\_matrix, roc\_auc\_score, classification\_report

from sklearn.preprocessing import StandardScaler

# Step 1: Create binary target variablemedian\_price = dataset['Price'].median()

dataset['High\_Price'] = (dataset['Price'] > median\_price).astype(int)

# Step 2: Prepare feature matrix X and target vector y

X = dataset.drop(columns=['Price', 'High\_Price', 'Name']) # Drop target columns from features

y = dataset['High\_Price']

# Optional: Standardize features (can help logistic regression)

scaler = StandardScaler()

X = scaler.fit\_transform(X)

# Step 3: Split into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 4: Train the logistic regression model

logreg = LogisticRegression()

logreg.fit(X\_train, y\_train)

# Step 5: Make predictions

y\_pred = logreg.predict(X\_test)

y\_pred\_proba = logreg.predict\_proba(X\_test)[:, 1] # Probabilities for ROC-AUC

# Step 6: Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

roc\_auc = roc\_auc\_score(y\_test, y\_pred\_proba)

print(f"Accuracy: {accuracy}")

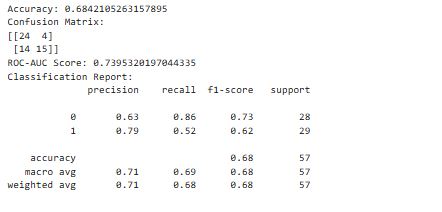
print("Confusion Matrix:")

print(conf\_matrix)

print(f"ROC-AUC Score: {roc\_auc}")

print("Classification Report:")

print(classification\_report(y\_test, y\_pred))



import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_proba)

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, label=f'ROC Curve (area = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--')

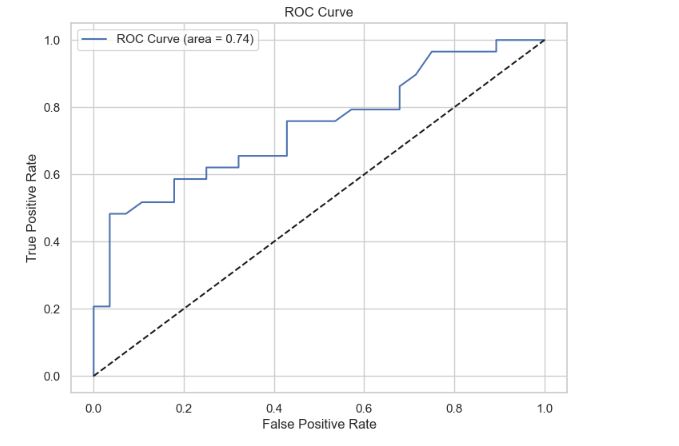
plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve')

plt.legend()

plt.show()



import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix

# Get the confusion matrix

cm = confusion\_matrix(y\_test, y\_pred)

# Plot the confusion matrix using a heatmap

plt.figure(figsize=(6, 5))

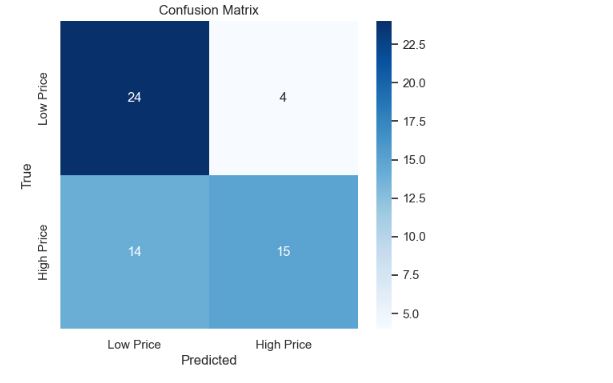
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Low Price', 'High Price'], yticklabels=['Low Price', 'High Price'])

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()



# Plot predicted probabilities for the positive class (High Price)

plt.figure(figsize=(8, 6))

sns.histplot(y\_pred\_proba, kde=True, color='skyblue', label='Predicted Probabilities')

plt.axvline(x=0.5, color='red', linestyle='--', label='Threshold (0.5)')

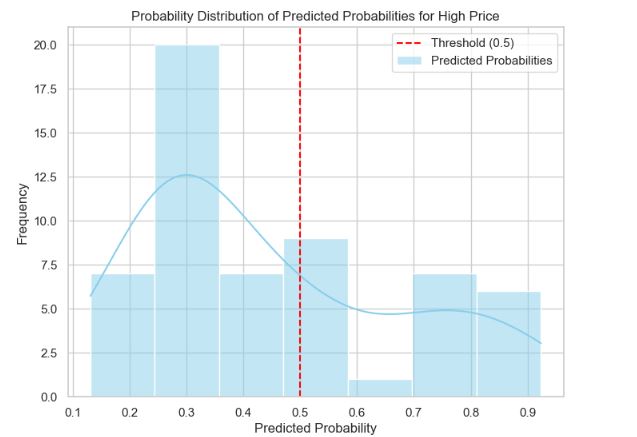
plt.title('Probability Distribution of Predicted Probabilities for High Price')

plt.xlabel('Predicted Probability')

plt.ylabel('Frequency')

plt.legend()

plt.show()



### Conclusion

In conclusion, the analysis of the sold players during the IPL 2023-24 auction provides valuable insights into the evolving dynamics of player valuation, team strategies, and the broader trends within the IPL ecosystem. The detailed examination of the auction data, we have been able to identify key patterns in player performance, pricing, and team composition, offering a clearer picture of how franchises are adapting to changing market conditions.

# References

##### Datasets:

* + IPL 2023-22 Players sold Dataset (used in our code):
    - Source: [www.kaggle.com](http://www.kaggle.com/)

##### Machine Learning Algorithms and Techniques:

* + Random Forest Classifier:

"Breiman, L. (2001). Random Forests." *Machine Learning*, 45(1), 5-32. https://doi.org/10.1023/A:1010933404324

* + Label Encoding:

Scikit-learn documentation on LabelEncoder:

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEnco der.html

* + Feature Importance:

"Scikit-learn documentation on Random Forest: Feature importance": https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForest Classifier.html

* + StandardScaler:

Scikit-learn documentation on StandardScaler:

[https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardS](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html) [caler.html](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html)

##### Visualization:

* + Matplotlib Documentation:
    - "Matplotlib: Plotting with Python." <https://matplotlib.org/stable/contents.html>
  + Visualization of Feature Importance:
    - "Visualizing Feature Importance in Random Forests." *Data Science Handbook*

(online resource):

<https://www.datacamp.com/community/tutorials/random-forests-classifier-python>

##### General Resources:

* + Scikit-learn Documentation:

Official documentation for machine learning in Python: <https://scikit-learn.org/stable/>

* + Pandas Documentation:

Official pandas documentation for data manipulation: <https://pandas.pydata.org/pandas-docs/stable/>