Introduction

Football is a data-rich sport, employing sports analytics always provide valuable insights into team performance, individual market value, future trends etc. This project focuses on Football League Table and uses Tableau for dashboard which help to display the outcomes of this project, PyCharm for Python-based predictive modelling which helps in showing accuracy, ROC etc and SQL Server Administration Studio 19 for data administration and querying.

The dataset, which have factors like age, nationality, club, position, market value, contract details, and individual statistics. To handle missing information, we get rid of inconsistencies, and arranged the dataset for analysis. SQL queries were used for data transformation and cleaning.

After preprocessing, the dataset was used to analyzed the noteworthy patterns, such as positional analysis, player market value distribution, club spending trends, information based on nationality and so on. During the predictive modeling stage, Python was used in PyCharm to build machine learning models for player performance prediction by using XGBoost, Linear Regression, Random Forest, Logistic Regression, and Decision Trees. These models help forecast player market valuations and assess future performance metrics based on historical data.

Then an interactive Tableau dashboard was ultimately created so to visually assess significant data, including club market value rankings, location comparisons, nationality-based distributions, and historical trends. The stakeholders who can utilize these dashboard to obtain actionable insights that support them in taking data-driven decisions are scouts, analysts, and club management.
By focusing on Tableau for visualization, Python for predictive modeling, and SQL for data processing, this project tells the importance of sports analytics. and forecasting football player performance and market trends.

Scope

This project aims to transfigured football analytics and enhance league performance assessment, player valuation, and football scouting decision-making by using data-driven insights, predictive modeling, and interactive visualization. The project handle data collection, preparation, analysis, predictive modeling, and visualization using Tableau, Python (PyCharm), and SQL Server Management Studio 19. Time of data management and preprocessing stage, it is important to manage large datasets containing player attributes such as age, nationality, position, club, transfer history, market value, contract details, and performance statistics. SQL queries are help to clean, analyze, and organize the dataset in order to ensure consistency and reliability of data. Creating useful information comprises of handling missing numbers, getting rid of duplicates, standardizing data formats, and gathering individual and team statistics.

The analysis part focus on players position distributions, club expenditure trends, market value trends, and contributions based on nationality. To recognize the key factors influencing contract negotiations, team performance, and player valuation, a variety of statistical techniques are employed.

Machine learning techniques like XGBoost, Linear Regression, Decision Trees, and Random Forest, models that help to predict player market value and future performance are developed during the predictive modeling stage. These models help football teams, scouts, and analysts make data-driven decision on transfers, talent scouting, and squad planning by forecasting a player's potential growth and financial worth.

For creating an interactive dashboard that help users to examine data in an understandable way, Tableau is used during the visualization and reporting stages. The dashboard includes club-by-club player values, league ranking comparisons, positional value distributions, and transfer market data. Stakeholders can easily specify the insights that meet their needs by using drill-down analyses, sort options, and filters.

All clubs and scouts, sports analysts, reporters, fantasy football managers, and bookmakers can all benefit from this report in the real world. They can use these data for strategic planning, performance tracking, and financial forecasting. The combination of SQL, Python, and Tableau also tells a scalable approach to football analytics; for more iterations, this strategy may be expanded to include match statistics, injury predictions, and tactical performance assessment.

OBJECTIVES

- The objective of this project is to give useful information about football league standings, player performance trends, and market value projections by using machine learning and data analytics. Tableau used for visualization, Python (PyCharm) for predictive modeling, and SQL for data administration, project aims to increase decision-making for football teams, scouts, analysts, and sports fans.
- One of the primary objectives is to clean, preprocess, and transform raw football player data so that it is reliable and organized for study. This involves utilizing SQL queries to aggregate crucial data, handle missing values, remove duplicates, and normalize data types
- For understanding how many factors influence a player's transfer fee, it is also important to analyze trends in player market value, team spending, and positional worth. By observing traits including age, nationality, contract length, and club achievement, the effort helps to highlight the key elements that affect market valuation. Information about high-value transfers, team budgets, and league-by-league market distribution can also help stakeholders make wise financial choices.
- Objectives is to generate machine learning models for player performance prediction utilizing techniques like XGBoost, Random Forest, Decision Trees, and Linear Regression. These models will be helped to verify potential football transfer market stars and forecast future player value based on historical performance data.
- Scouts and club managers could used the data-driven insights from the predictive analytics component to upgrade player recruitment and squad-building strategies.
- An interactive Tableau dashboard that helps users to visually analyze football league standings, team results, and player market trends is another part of the project.

Stakeholders can use real-time data-driven insights to make better decisions with the dashboard's drill-down tools, comparisons, and filters. The visualizations has league-wise player distributions, nationality heatmaps, position-based player valuation, and club market value rankings to more understanding at the individual and team levels.

- This project helps to bridge the gap between raw football data and important insights by demonstrating how SQL, Python, and Tableau can be used in concert to handle real-world sports analytics problems. They used as a scalable model that might be enlarged in the future to include tactical performance evaluation, injury predictions, and additional match data.
- Project aims to looked at football scouting, transfer market analysis, and performance prediction through data-driven approaches, helping teams, analysts, and sports fans make informed, profitable decisions in the football sector.

Problem Definition

- One important area of research is predicting the long-term performance of a team or a single player.
- Coaches, sports agents, and bettors are very interested in evaluating how teams or players perform throughout a season in relation to previous ones.
- The unique characteristics of football games make long-term forecasting challenging.
- It is more difficult to predict the outcomes because there are so few goals scored in each game.
- It might be difficult to precisely record game events in football due to the game's continuous flow of play.
- Attackers are usually rated higher than defenders, even if defenders might play an equally significant role in a team's strategy.
- The study aims to identify the key skills and traits that effective defenders possess in order to enhance player evaluation.

Innovation To Be Carried By Me

With the intention of providing practical insights into football league performance and player market value, this project merge cutting-edge advancements in data management, predictive modeling, and interactive visualization. By joining Tableau for dynamic displays, Python (PyCharm) for complex machine learning models, and SQL Server for structured data processing, this project helps several innovative approaches to enhance football data analytics.

One important breakthrough is the automatic player performance prediction model. Considering contract details, age, market trends, and historical performance metrics, it used the XGBoost, Random Forest, Decision Trees, Linear Regression, and Logistic Regression to forecast a player's future value. Not like traditional scouting methods that rely on subjective evaluations, data-supported predictions that can used clubs to find underrated talents and high-potential prospects before they become well-known. This could be significantly reduce transfer market inefficiencies and enhance scouting strategies.

A noteworthy innovation is the Tableau dashboard, which Provides interactive, real-time views into league standings, player market trends, club financial performance, and positional value analysis. In place of using static reports, club managers, scouts, and analysts may used this dashboard to apply filters, organize data, and do customized drill-down studies based on their own needs. This dashboard also shows geospatial analysis (maps) in which countries players come from, which helps clubs and agencies better target growing football markets.

This project also use a special SQL-based data transformation technology that helps automated data aggregation, normalization, and cleansing. This invention helps that player data, transfer histories, and contract details are handled appropriately before being used for analysis or prediction. The system's seamless merging of new data sources enables scalability, meaning that in further updates, the model can be modified to add injury reports, tactical performance data, and live match statistics.

To help teams determine if foot preference effect performance and market worth, This creative picture provides insights into positional market trends and allows teams to carefully select individuals based on their playing style and on there fitness.

This project add these cutting-edge developments in machine learning, real-time analytics, and interactive visualization to transform traditional football scouting and market analysis into a cutting-edge, data-driven process. These achivement have the potential to fundamentally alter the decision-making processes of teams, professionals, sports leagues, and football supporters worldwide.

Methodology Which Is Being Applied On My Project (Implementation Plan)

1. Compiling and Bringing in Data:

In this step is to gather player-related data from reliable sources, including contract details, player statistics, club performance records, and transfer market data. These information helps for study of characteristics such as player name, age, height, country, position, club, market value, maximum value, contract duration, and transfer history. This data is added to SQL Server Management Studio 19 and then stored in a structured relational manner for efficient processing and querying.

2. Data Cleaning & Preprocessing:

- To address missing values, remove inconsistencies, and correct formatting issues, the raw dataset is meticulously cleaned before to analysis.
- Handling Missing Values - SQL queries can be used to modify or remove null values in crucial variables like price, club, or position.
- Data Type Standardization - Converting numerical data into study-friendly representations, like contract years and player value.
- Removing Duplicates & Outliers - Identifying and removing duplicate player entries and outlier values that can skew predictions.

- The practice of creating new calculated columns to enhance analysis, such as price difference (peak vs. current price), years at the club (contract term), and positional value ranking, is known as feature engineering.

3. Data Transformation & Structure :

- After being cleaned, the data is arranged for efficient analysis. SQL queries perform important modifications.
- The total player market value each club is totaled in order to analyze club spending trends.
- Finding the average market value based on a position is the first step in comparing positional worth.
- Setting up contract conditions and expiration dates in order to analyze club player retention strategies.
- Creating club-specific performance metrics and calculating player values to assess team strength.

4. Exploratory data analysis (EDA) and statistical insights :
    - Python (Pandas, Matplotlib, and Seaborn) and SQL queries are used in exploratory data analysis (EDA) to identify patterns and correlations in a structured dataset. Crucial analyses include:
      Using SQL aggregate and ranking queries, the most valuable players and clubs may be identified.
    - Analyzing Market Value Trends by Player Age: To determine the peak years for a player's financial worth.
    - Player Distribution by League and Nationality: To display the origins of the most valuable players.
    - Club-wise Spending vs. Performance Metrics: To assess how club investments and success are related.

5. Using Machine Learning to Forecast Player Performance and Market Value :

- Machine learning models and Python (PyCharm) are employed in predictive analytics to forecast player market value and future performance.
- Decision trees and linear regression are used to predict changes in market value based on historical trends.
- Random Forest & XGBoost seek to improve forecast accuracy by considering several aspects, such as player age, position, contract status, club reputation, and previous price changes.
- Feature importance analysis is the process of identifying the factors that most affect a player's value, such as age, country, and length of contract.

6. Developing Interactive Tableau Dashboards:
- The Top 10 Most Important Teams and Players (Bar Chart)
- Player Market Value Distribution by League (Bubble Chart);
- Player Market Value Distribution by Nationality (Geographic Map);

- Positional Market Value Comparison (Heatmap);
- Club-wise Market Value Growth vs Spending (Line Chart);
- Butterfly Chart for Left-footed vs. Right-footed Players

Need Of The Project

In football teams, scouts, analysts, and stakeholders totally rely on the data analytics to make educated decisions regarding player transfers, team performance, and market value assessments. A data-driven approach is important for maximizing decision-making efficiency in football due to its dynamic character, which helps increasing competition, rising transfer fees, and unpredictable player performances. This project join the gap between raw football data and actionable insights by offering a comprehensive and analytical framework for evaluating players, teams, and leagues using SQL, Python, and Tableau.

The important motivators for this project is the need for accurate player market value. Since football teams invested a lot of money in catching talent, it could move to lost opportunities or financial losses if a player's worth is overrated or underestimated. Using these factors such as age, position, performance history, contract length, and transfer trends, this study assists in predicting a player's future market value. Machine learning methods such as XGBoost, Random Forest, Linear Regression, Logistic Regression, and Decision Trees are helps to perform this. This make ensures that teams calculated investments to improve the gains from player transfers.

Another requirements for this project is the switch nature of football leagues and their financial operations. Clubs take the most recent data in league rankings, team performance, and spending for stay competitive. By collecting and analyzing club-by-club expenditure patterns, the research provides a proper picture of the financial sustainability of football leagues and helps to recognize which teams are spending wisely versus excessively on players.

The idea seeks to improved squad management and team planning. By testing player characteristics and market values, clubs could decide which players to retain, loan out, or sell. This enables them to keep a important staff with optimal talent allocation. Because organizations can provides a player's career path, they may also invest in young players with high potential for long-term financial and competitive success.

Supporters, analysts, and bettors can also used this project to analyse club and individual performances, giving them more actual information on league standings, transfer trends, and match predictions. This project joins Tableau for interactive visualization, Python for predictive analytics, and SQL for data administration to create a powerful, scalable, and real-time analytics platform that transforms football decision-making processes on various levels.

Beneficiary

1. Coaches and Football Teams :

Football clubs stand to benefit the most from this project since it provides in-depth data on team performance, player valuation, and market trends. By utilizing machine learning forecasts to identify potential superstars before their value increases, clubs can make cost-effective transfer decisions. Based on contract lengths, injury risks, and projected market value growth, managers can also decide which players to retain, loan, or sell to enhance squad and financial planning.

2. Football scouts and recruitment analysts :

Scouts and recruitment teams mostly use data to identify new talent. The interactive dashboards of this project, which display positional market trends, player distributions by nationality, and age-based performance estimates, are advantageous to them. Clubs can use data-driven recommendations instead of relying solely on human scouting and subjective opinions to find underappreciated yet highly-potential players. By combining player performance patterns with predictive analytics, scouts can make better hiring decisions and reduce the chance of signing players who are underperforming or overvalued.

3. Sports analysts and data scientists :

Sports analysts will greatly benefit from this project's structured and expandable framework for doing in-depth football data analysis. Analysts can use Tableau for visualization, Python for predictive modeling, and SQL queries for data transformation to find league trends, club spending efficiency, and player performance patterns. This information can be used to develop reports, articles, and presentations for coaches, club owners, and the media.

4. Investors and club owners :

The value of the football industry has increased to billions of dollars, and investors seek data-backed insights before funding teams, leagues, or player contracts. This study allows them to look at financial trends, such as which teams have the highest return on investment (ROI) based on player spending. Additionally, predictive models help investors make well-informed financial decisions about partnerships, acquisitions, and sponsorships by helping them identify which players or teams are most likely to see an increase in value.

5. Sports journalists and media professionals :

For journalists covering football transfer news, league statistics, and match analysis, this initiative can offer data-supported reporting. Instead of relying solely on opinions, media professionals may use Tableau dashboards to show club spending trends, top goal scorers, and player market value changes to make their material more engaging and insightful. This enables them to provide fact-based analysis, which improves their coverage of player transactions, team performances, and market trends.

6. Fantasy football fans and bookmakers :

Fantasy football gamers and bookmakers rely on accurate player data and performance trends to inform their choices. By providing data on player form, positional market value, and historical performance trends, this project assists consumers in making well-informed

wagers or optimizing their fantasy football teams. Predictive modeling can also improve betting strategies by identifying which players are most likely to improve in future games.

7. Football Fans & Enthusiasts :

For casual football fans who want to learn more about their favorite clubs and players, this project's interactive dashboards and performance patterns are helpful. By looking into player rankings, club spending patterns, and nationality-based distributions, fans can gain a data-driven understanding of the sport. By allowing people to engage in informed discussions, debates, and predictions, it improves their pleasure of football.

Testing Plan

1) Gathering and Importing Data:
   Testing Focus:
   Verifying the accuracy and completeness of the data during capture and import.

   Methods:
   Confirming that the data was gathered from all of the specified sources.
   Checking each attribute's forms and data kinds (date, text, numeric, etc.).
   Making sure to provide all required details, including name, age, and club.
   Verifying that the data is correctly stored in SQL Server Management Studio 19.

   Measurements:
   The quantity of imported records matches the source data.
   The data type conversion was flawless.
   No missing or damaged data.

2) Data cleaning and preprocessing:

Testing Focus:

Confirming that the data cleansing and transformation processes are executed accurately.

Methods:

Missing Value Handling: Verifying that null values are handled in accordance with the rules (e.g., replaced, removed).

Data Type Standardization: Making sure that all data types are consistent and standardized.

Making sure outliers and duplicate records have been properly identified and removed.

Using feature engineering to confirm the accuracy of recently introduced columns (such price difference and years at the club).

Measures:

After processing, all significant fields are free of null values.

Every record contains the same kinds of data.

The number of records after cleaning is in line with the expected numbers.

For new functionality, precise computations are performed.

3) Data Transformation & Structuring:

Testing goal:
Checking the accuracy of SQL-based data manipulations.
Methods:
Verifying the accuracy of the data's aggregation, including the total player market worth per club.
Average Calculation: Confirming the accuracy of average market value calculations.
Date Handling: Checking the arrangement of the contract's conditions and the termination dates.
Team Strength Calculation: Checking that player values and team strength indicators are correct.

Measures:
The total figures are consistent with the expected results.
The mean values are within acceptable ranges.
The format and arrangement of the date fields are correct.
The calculated values for the team and players are accurate.

4) EDA, or exploratory data analysis, and statistical insights:

Testing Objective: Verifying the accuracy of the EDA and statistical analysis findings.

Methods:
Verifying the accuracy of the SQL queries used for EDA by doing SQL Query Validation.
Python Script Validation: Checking that the Python scripts (Pandas, Matplotlib, Seaborn) used for EDA are accurate.
Checking the results of statistical calculations (e.g., most valuable players, market value trends).

Measures:
SQL queries return the expected results.

Python scripts produce accurate results and operate without a hitch. The results of the EDA and the data agree.

5) Machine Learning-Based Player Performance and Market Value Prediction:

Testing Focus: Evaluating the performance and accuracy of machine learning models.

Methods: Verifying that the correct data is used to train the models in order to validate model training.
Calculating Performance Metrics: Verifying that the Mean Absolute Error (MAE) and R-Squared metrics were computed accurately.
Prediction Accuracy: Assessing the precision of market value and performance projections using test data.
Making sure the results of the feature importance analysis are accurate.

Measures:
The MAE and R-squared values fall within acceptable ranges.
Predictions are correct within a range.
The assessment of feature importance corresponds with domain knowledge.

6) Making Tableau Dashboards That Are Interactive:

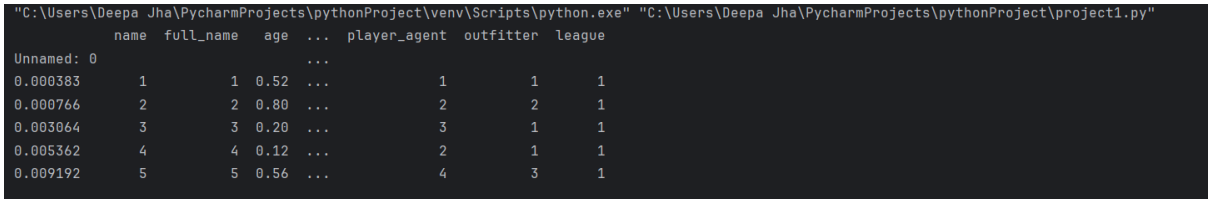
Testing Objective: Confirming the functionality, correctness, and usability of Tableau dashboards.

Methods: Making sure dashboards are connected to the correct SQL Server data source to confirm the data source.
Visualization Accuracy: Verifying that maps, bubble charts, bar charts, and other visualizations accurately depict the data.
Testing for Interactivity: Confirming that dashboard functions (including sorting, drill-downs, and filters) operate as planned.
Testing for usability: Evaluating the dashboards' clarity and usability.

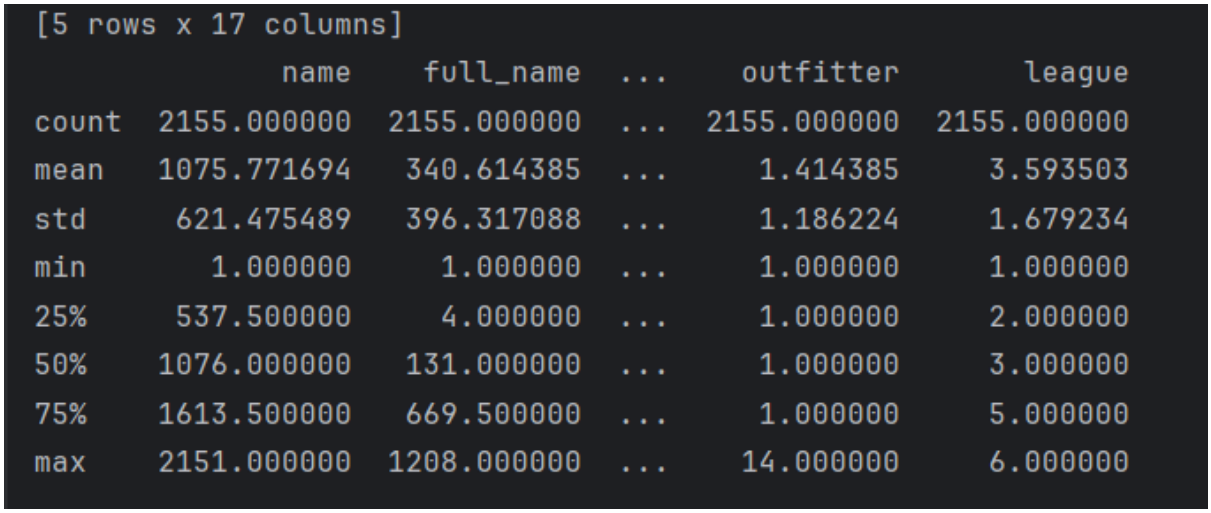
Measures:
Dashboards display accurate and up-to-date information.
The interactive elements function as intended.
Dashboards are easy to understand and use.

Results

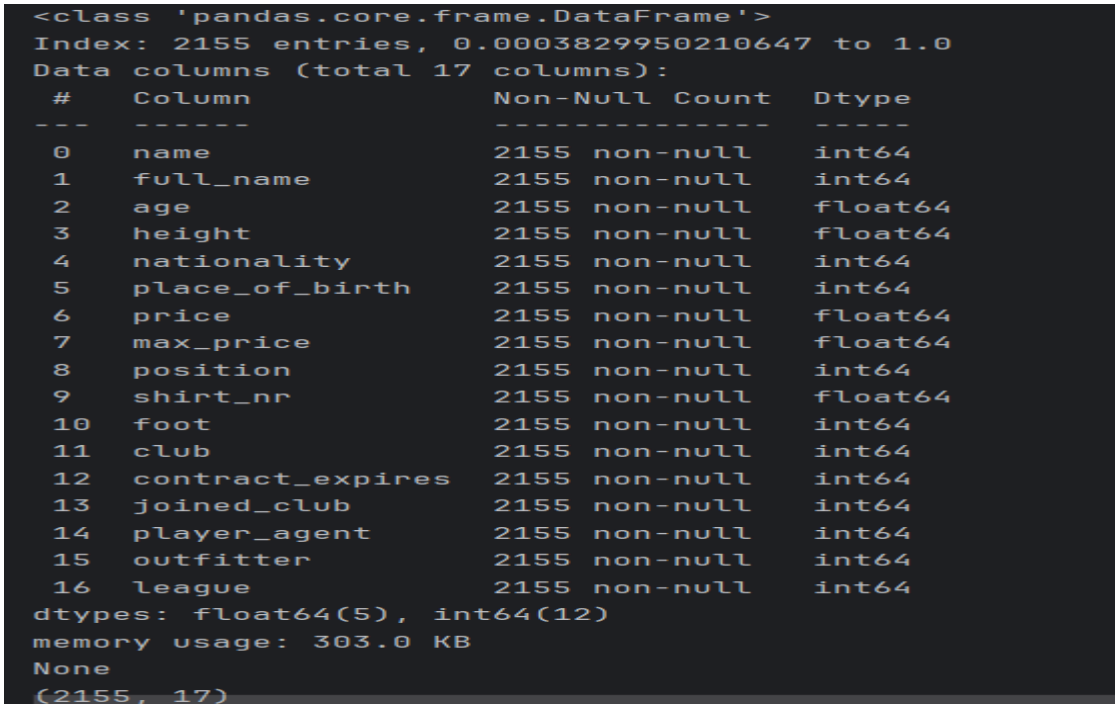
```
# Display basic information
print(df_players.head())
print(df_players.describe())
print(df_players.info())
```



```
"C:\Users\Deepa Jha\PycharmProjects\pythonProject\venv\Scripts\python.exe" "C:\Users\Deepa Jha\PycharmProjects\pythonProject\project1.py"
          name  full_name   age  ...  player_agent  outfitter  league
Unnamed: 0                       ...
0.000383     1          1  0.52  ...             1          1       1
0.000766     2          2  0.80  ...             2          2       1
0.003064     3          3  0.20  ...             3          1       1
0.005362     4          4  0.12  ...             2          1       1
0.009192     5          5  0.56  ...             4          3       1
```



```
[5 rows x 17 columns]
                name     full_name  ...     outfitter       league
count   2155.000000   2155.000000  ...   2155.000000  2155.000000
mean    1075.771694    340.614385  ...      1.414385     3.593503
std      621.475489    396.317088  ...      1.186224     1.679234
min        1.000000      1.000000  ...      1.000000     1.000000
25%      537.500000      4.000000  ...      1.000000     2.000000
50%     1076.000000    131.000000  ...      1.000000     3.000000
75%     1613.500000    669.500000  ...      1.000000     5.000000
max     2151.000000   1208.000000  ...     14.000000     6.000000
```



```
<class 'pandas.core.frame.DataFrame'>
Index: 2155 entries, 0.0003829950210647 to 1.0
Data columns (total 17 columns):
 #    Column               Non-Null Count  Dtype
---   ------               --------------  -----
 0    name                 2155 non-null   int64
 1    full_name            2155 non-null   int64
 2    age                  2155 non-null   float64
 3    height               2155 non-null   float64
 4    nationality          2155 non-null   int64
 5    place_of_birth       2155 non-null   int64
 6    price                2155 non-null   float64
 7    max_price            2155 non-null   float64
 8    position             2155 non-null   int64
 9    shirt_nr             2155 non-null   float64
 10   foot                 2155 non-null   int64
 11   club                 2155 non-null   int64
 12   contract_expires     2155 non-null   int64
 13   joined_club          2155 non-null   int64
 14   player_agent         2155 non-null   int64
 15   outfitter            2155 non-null   int64
 16   league               2155 non-null   int64
dtypes: float64(5), int64(12)
memory usage: 303.0 KB
None
(2155, 17)
```

The DataFrame's top five rows are automatically retrieved by Pandas'.head() method. This is a common way to quickly look at the structure of the data and see the names of the columns and a few example items.

In print(df_players.describe()), the print() method is utilized. The result of using the.describe() function on df_players is returned this time. The describe() method generates descriptive statistics for the numerical columns of the DataFrame. These statistics typically include the count, mean, standard deviation, minimum, maximum, and the 25th, 50th (median), and 75th percentiles for each numerical column.

Dataset details:

Old data: 2500

New data: 2155

Rows: 17

Rows Details:

Name

Full_name

Age

Height

Nationality

Place_of_birth

Price

Max_price

Position

Shirt_nr

Foot

Club

Contract_expires

Joined_club

Player_agent

Outfitter

League

```
---Analysis queries
---Top 10 most expensive players
SELECT TOP 10 name, club, price, position, league
FROM top5_leagues_players
ORDER BY price DESC;
```

	name	club	price	position	league
1	Kylian Mbapp+¬	Paris SG	180	Attack - Centre-Forward	Ligue1
2	Erling Haaland	Man City	170	Attack - Centre-Forward	EPL
3	Jude Bellingham	Bor. Dortmund	120	midfield - Central Midfield	Bundesliga
4	Vinicius Junior	Real Madrid	120	Attack - Left Winger	LaLiga
5	Phil Foden	Man City	110	Attack - Left Winger	EPL
6	Bukayo Saka	Arsenal	110	Attack - Right Winger	EPL
7	Jamal Musiala	Bayern Munich	110	midfield - Attacking Midfield	Bundesliga
8	Victor Osimhen	SSC Napoli	100	Attack - Centre-Forward	Other
9	Federico Valverde	Real Madrid	100	midfield - Central Midfield	LaLiga
10	Pedri	Barcelona	100	midfield - Central Midfield	LaLiga

This SQL query, which is helpful for dashboards, scouting, or investment decision-making, is written to provide a rapid and efficient understanding of high-value football talent by:

• Extracting only the most important player attributes;

• Sorting players by highest market value;

• Restricting results to the top 10.

```
---Average Age of Players by Leagues
SELECT league, AVG(age) AS avg_age
FROM top5_leagues_players
GROUP BY league
ORDER BY avg_age DESC;
```

	league	avg_age
1	LaLiga	27
2	EPL	26
3	SerieA	26
4	Other	25
5	Ligue1	25
6	Bundesliga	25

With the highest average age (27), LaLiga may have older or more seasoned players.
The average age of the EPL and Serie A is 26.
The average age of players in the Bundesliga, Ligue 1, and other leagues is 25.
This type of analysis can clarify following
• Levels of squad maturity and experience;

• Leagues' emphasis on veterans or youth investment;

• Scouting and transfer tactics (e.g., targeting younger talent or seasoned players).

```sql
---Number of Players Per Position
SELECT position, COUNT(*) AS player_count
FROM top5_leagues_players
GROUP BY position
ORDER BY player_count DESC;
```

	position	player_count
1	Defender - Centre-Back	468
2	midfield - Central Midfield	331
3	Attack - Centre-Forward	324
4	Goalkeeper	315
5	Defender - Right-Back	230
6	Defender - Left-Back	209
7	midfield - Defensive Midfield	199
8	midfield - Attacking Midfield	168
9	Attack - Left Winger	162
10	Attack - Right Winger	153
11	midfield - Right Midfield	20
12	Attack - Second Striker	16
13	midfield - Left Midfield	16
14	midfield	1

With 468 players, the Defender-Center-Back position is the most popular, demonstrating the significant desire for a solid core defense.
Additionally well-represented are center-forwards and central midfielders, highlighting their significance in both goal scoring and attack development.
There are also a lot of goalkeepers (315), which is to be expected as every club usually has a number of goalkeepers.
The following positions are less frequently used

Right/Left Midfielders;

Second Strikers;

Generic Midfield (maybe due to inconsistent data entry).

```sql
---top 5 Clubs with Highest Total Market Value
SELECT TOP 5 club, SUM(price) AS total_value
FROM top5_leagues_players
GROUP BY club
ORDER BY total_value DESC;
```

	club	total_value
1	Man City	1051.25
2	Chelsea	1007.5
3	Bayern Munich	979.7
4	Arsenal	890
5	Paris SG	879.4

Manchester City has the most market value overall, at over €1 billion, which suggests that their team is very valuable and strong.

The squads of Chelsea and Bayern Munich are also extremely important. Paris Saint-Germain and Arsenal, who are both close to €900 million, complete the top five. These numbers show

• The caliber and depth of the teams;

• The desire for their players in the market;

• Probability of strong play and a worldwide following.

```
---Players with Highest Market Value Growth
SELECT TOP 10 name, club, price, max_price, price_difference
FROM top5_leagues_players
ORDER BY price_difference DESC;
```

	name	club	price	max_price	price_difference
1	Eden Hazard	Real Madrid	5	150	145
2	Philippe Coutinho	Aston Villa	14	150	136
3	Lionel Messi	Paris SG	45	180	135
4	Antoine Griezmann	Atl+¬tico Madrid	25	150	125
5	Neymar	Paris SG	70	180	110
6	Sadio Man+¬	Bayern Munich	45	150	105
7	Raheem Sterling	Chelsea	60	160	100
8	N'Golo Kant+¬	Chelsea	20	100	80
9	Mohamed Salah	Liverpool	70	150	80
10	Paul Pogba	Juventus	20	100	80

Eden Hazard's market worth has dropped the most, from €150 million to €5 million, a staggering €145 million loss.
Griezmann, Messi, and Philippe Coutinho come next, all of whom had drops of more than €125 million.
Numerous players on the list were once highly rated, but they may have suffered from

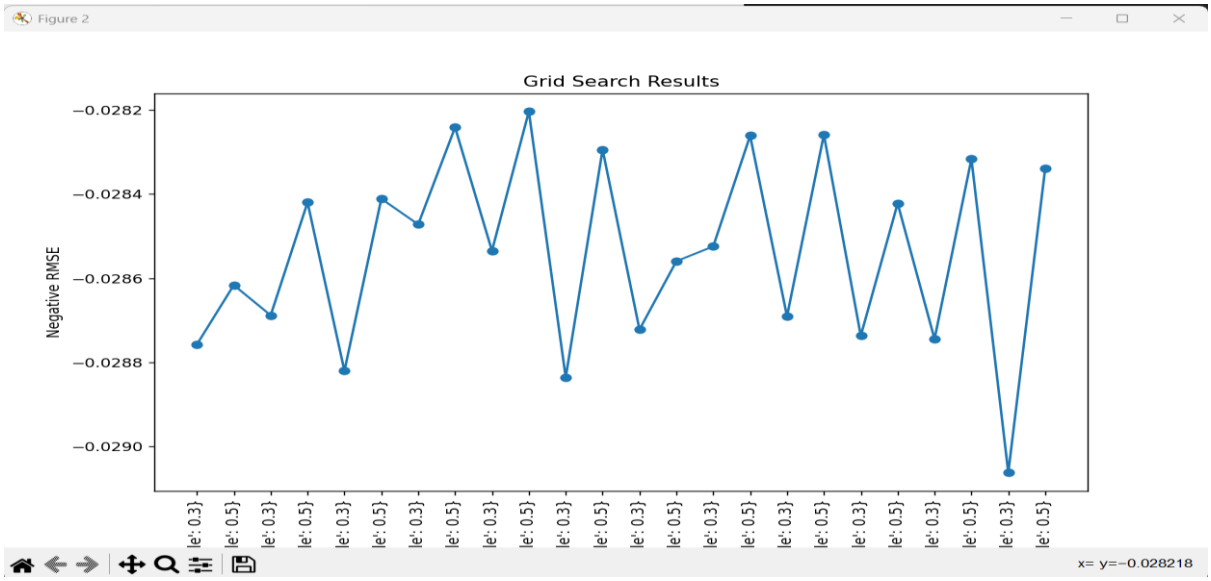
underwhelming performances, age decline, or injury.
One way to understand this list is

- Players who were once considered investment-heavy assets but may now be cheap;
- Candidates for veteran acquisition or retirement research;
- A gradual drop in value

```
# One Hot Encoding
columns_to_encode = ['league', 'foot', 'position', 'club', 'contract_expires',
                     'joined_club', 'player_agent', 'outfitter', 'nationality']
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(handle_unknown='ignore'), columns_to_encode)],
                       remainder='passthrough')
df_features_encoded = ct.fit_transform(df_features)

# Train-test split
x_train, x_test, y_train, y_test = train_test_split( *arrays: df_features_encoded, df_target, test_size=0.3, random_state=22)

# Convert DataFrame to NumPy array and flatten
y_train, y_test = y_train.values.flatten(), y_test.values.flatten()

print(f'x_train: {x_train.shape}, x_test: {x_test.shape}, y_train: {y_train.shape}, y_test: {y_test.shape}')
```

```
x_train: (1508, 1679), x_test: (647, 1679), y_train: (1508,), y_test: (647,)
Fitting 4 folds for each of 24 candidates, totalling 96 fits
Best Parameters: {'colsample_bytree': 0.7, 'learning_rate': 0.03, 'max_depth': 7, 'min_child_weight': 3, 'n_estimators': 300, 'nthread': 4, 'objective': 'reg:squarederror', 'sul
```

```
# Hyperparameter tuning for XGBoost
param_grid = {
    'nthread': [4],
    'objective': ['reg:squarederror'],
    'learning_rate': [0.03, 0.05],
    'max_depth': [4, 7],
    'min_child_weight': [2, 3, 4],
    'subsample': [0.3, 0.5],
    'colsample_bytree': [0.7],
    'n_estimators': [300]
}

xgb = xgboost.XGBRegressor(objective='reg:squarederror')
grid_search = GridSearchCV(estimator=xgb, param_grid=param_grid, scoring='neg_root_mean_squared_error', cv=4, verbose=1)
grid_search.fit(x_train, y_train)

best_params = grid_search.best_params_
print("Best Parameters:", best_params)

# Visualizing parameter tuning results
plt.figure(figsize=(10, 6))
plt.plot( *args: range(len(grid_search.cv_results_['mean_test_score'])), grid_search.cv_results_['mean_test_score'], marker='o')
plt.xlabel('Parameter Combination')
plt.ylabel('Negative RMSE')
plt.title('Grid Search Results')
plt.xticks(range(len(grid_search.cv_results_['params'])), grid_search.cv_results_['params'], rotation=90)
plt.show()
```



Negative Y-Axis RMSE

This is the negative value of the Root Mean Squared Error. Since the objective of scoring metrics is to maximize them, scores are returned as negative integers in scikit-learn when GridSearchCV is used with scoring set to "neg_root_mean_squared_error".
Consequently, the closer this value is to zero (least negative), the better the model performs. This graph displays the range of error across multiple parameter combinations, ranging from -0.0282 to -0.0290.

The X-Axis

Each tick on the x-axis seems to indicate a set of hyperparameters, most likely for a machine learning model like Linear Regression, Random Forest, or XGBoost.

From the labels, we can infer that the 'lei': 0.3 and 'lei': 0.5 hyperparameters are used.

A line plot connects the scores (Negative RMSE) of each parameter combination. It is evident that the model performs differently for different situations.

Near -0.0282, the lowest RMSE discovered, is the best performing site (highest on the y-axis = least negative RMSE).

The lowest performance, which is closer to -0.0290, is indicated by a higher error.

```
# Train final model
best_xgb = xgboost.XGBRegressor(**best_params)
best_xgb.fit(x_train, y_train)

# Predictions and evaluation
pred = best_xgb.predict(x_test)

mae = mean_absolute_error(y_test, pred)
mse = mean_squared_error(y_test, pred)
rmse = np.sqrt(mse)

print(f'Mean Absolute Error: {mae}')
print(f'Mean Squared Error: {mse}')
print(f'Root Mean Squared Error: {rmse}')
```

```
Mean Absolute Error: 0.022650789255133 75
Mean Squared Error: 0.0010236848939831 97
Root Mean Squared Error: 0.031995076089661
```

The first section, under the comment "Train final model", discusses training the chosen model. The line "Best_xgb = xgboost.XGBRegressor(**best_parameters)" is used to instantiate an XGBoost regressor object. It uses the XGBRegressor class from the xgboost library. Remarkably, it passes the **best_params parameter. The optimal collection of hyperparameters found during a previous hyperparameter tuning process likely using GridSearchCV or RandomizedSearchCV is stored in the best_parameters dictionary. This ensures that the final model is trained using the best setup. The following line, "best_xgb.fit(x_train, y_train)", uses the training data to train the XGBoost model. x_train represents the features of the training set, and denotes the target variable's matching values.

The second section, which falls under the comment "Predictions and evaluation", focuses on generating predictions on the test data and evaluating the model's performance. Predictions on the test features, x_test, are made using the trained best_xgb model in the line "pred = best_xgb.predict(x_test)". The resulting predictions are stored in the pred variable. After that, several common metrics for assessing regression are calculated. The average of the absolute differences between the expected values (pred) and the actual target values (y_test) is known as the Mean Absolute Error, or mae = mean_absolute_error(y_test, pred). The average of the squared differences between the true and predicted values is the Mean Squared Error, or mse = mean_squared_error(y_test, pred).

```python
#we are going to create a model that determines how good the metrics can be in predicting a player's market value.
Top_five_leagues_players = df_players.bfill(axis=0).fillna(0)

target = Top_five_leagues_players[['price']]
features = Top_five_leagues_players[['age', 'height', 'league','foot', 'position', 'club',
                        'contract_expires', 'joined_club', 'player_agent', 'outfitter', 'nationality']]

columns_to_encode = ['league','foot', 'position', 'club', 'contract_expires', 'joined_club', 'player_agent', 'outfitter', 'nationality']

features.loc[:, columns_to_encode] = features.loc[:, columns_to_encode].astype(str)

ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), columns_to_encode)], remainder='passthrough')
features_encoded = ct.fit_transform(features)

x_train, x_test, y_train, y_test = train_test_split( *arrays: features_encoded, target, test_size=0.2, random_state=42)

y_train = y_train.values.ravel()
y_test = y_test.values.ravel()

model = LinearRegression()

model.fit(x_train, y_train)

predictions = model.predict(x_test)

mae = mean_absolute_error(y_test, predictions)
mse = mean_squared_error(y_test, predictions)
rmse = mean_squared_error(y_test, predictions, squared=False)
```

```python
print('Mean Absolute Error:', mae)
print('Mean Squared Error:', mse)
print('Root Mean Squared Error:', rmse)
```

```
Mean Absolute Error: 0.03915241896149687
Mean Squared Error: 0.0026793654990406126
Root Mean Squared Error: 0.05176258783175946
```

The above code outlines the initial stages of developing a machine learning model to predict a football player's market value. First, a set of relevant attributes is selected, including age, height, preferred foot, position, club, contract expiration date, joining date, player agency, outfitter, and nationality. The target variable is then set to "price."

It then identifies category columns among these attributes that need to be encoded into a numerical format suitable for machine learning methods. This is achieved by utilizing scikit-learn's ColumnTransformer after first converting these category columns to string type. More specifically, it one-hot encodes the specified categorical columns using OneHotEncoder while leaving the other numerical attributes unchanged.

The encoded features and the target variable are separated into training and testing sets using train_test_split with an 80/20 ratio and a fixed random state for reproducibility. The target variables in the training and testing sets are restructured into one-dimensional arrays using.ravel(). A LinearRegression model is then created and trained using the encoded training features and matching training target values. The encoded testing features are predicted using the trained model, and the three common regression assessment metrics are calculated and saved by comparing the model's predictions with the actual market values in the test set.

```python
# Train Decision Tree Regression
dt_regressor = DecisionTreeRegressor(random_state=42)
dt_regressor.fit(x_train, y_train)
dt_pred = dt_regressor.predict(x_test)

# Evaluate Decision Tree Regressor
dt_mae = mean_absolute_error(y_test, dt_pred)
dt_mse = mean_squared_error(y_test, dt_pred)
dt_rmse = mean_squared_error(y_test, dt_pred, squared=False)

print('\n--- Decision Tree Regressor ---')
print(f'Mean Absolute Error: {dt_mae}')
print(f'Mean Squared Error: {dt_mse}')
print(f'Root Mean Squared Error: {dt_rmse}')
```

```
--- Decision Tree Regressor ---
Mean Absolute Error: 0.02453971174757888
Mean Squared Error: 0.0014295973734281 35
Root Mean Squared Error: 0.037810016839828 77
```

This Python code snippet create on the previous steps of data preparation and splitting and centre on training and evaluating a Decision Tree Regressor model for predicting the football player's market value. To ensure exact data, a DecisionTreeRegressor model is first initialized with a fixed random_state, which make sure that the training process yields the same result each time when it is run. Then model is trained using the previously generated training features (x_train) and matching training target values (y_train) with the help of the.fit() method.

The model's predictive data on the unseen test data (x_test) is measured after training using the.predict() method, and the resulting data are stored in the dt_pred variable. Several regression evaluation metrics are calculated to judge the model's correctness, including the Mean Squared Error (dt_mse), which calculates the average of the squared differences, the Root Mean Squared Error (dt_rmse), which is the square root of the MSE and provides an error metric in the exact units as the target variable, and the Mean Absolute Error (dt_mae), which shows the average absolute difference between the predicted and actual data.

The Decision Tree Regressor's calculated evaluation metrics are delivered to the console in an comprehensible manner so that the model's performance on the test data may be immediately judged. When applied to new, untested data, this result helps to judged how well the Decision Tree Regressor predicts player market prices.

```python
# Train Random Forest Regressor
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)
rf_regressor.fit(x_train, y_train)
rf_pred = rf_regressor.predict(x_test)

# Evaluate Random Forest Regressor
rf_mae = mean_absolute_error(y_test, rf_pred)
rf_mse = mean_squared_error(y_test, rf_pred)
rf_rmse = mean_squared_error(y_test, rf_pred, squared=False)

print('\n--- Random Forest Regressor ---')
print(f'Mean Absolute Error: {rf_mae}')
print(f'Mean Squared Error: {rf_mse}')
print(f'Root Mean Squared Error: {rf_rmse}')
```

```
--- Random Forest Regressor ---
Mean Absolute Error: 0.020644367044629 96
Mean Squared Error: 0.0009375257220473329
Root Mean Squared Error: 0.030619041821182663
```

Tthis Python code sample totally focus on training and evaluating the Random Forest Regressor, an ensemble learning method. The initiative step is to initialize a RandomForestRegressor model. It is configured with n_estimators=100, which tell us that the random forest will contain 100 different decision trees. Same as to the Decision Tree Regressor, a random_state of 42 is set for repeatability. This model is then trained with the help of the.fit() method using the matching training goal values (y_train) and the training features (x_train).

After training, the Random Forest Regressor forecast the market values for the unseen test data (x_test), which are then written in the rf_pred variable with the help of the.predict() method. The performance of this model is analysed using the same regression assessment metrics: Mean Absolute Error (rf_mae), Mean Squared Error (rf_mse), and Root Mean Squared Error (rf_rmse). These metrics are calculated by comparing the model's predictions (rf_pred) with the actual market values in the test set (y_test).

Finally, evaluation metrics calculated by the Random Forest Regressor are moving to the console with a clear label. This output allows a exact comparison of the Random Forest Regressor's performance with the previously evaluated Linear Regression and Decision Tree Regressor models in terms of predicting football player market prices on the test data. Using a various strategy, like Random Forest, often leads to higher predicted accuracy and robustness than single decision tree models.

```
# --- Logistic Regression (Converting Price into Categories) ---

# Convert price into 3 bins: Low, Medium, High
bin_discretizer = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='quantile')
y_binned = bin_discretizer.fit_transform(target).ravel()

# Train-test split with classified target
x_train_log, x_test_log, y_train_log, y_test_log = train_test_split( *arrays: features_encoded, y_binned, test_size=0.2, random_state=42)

# Train Logistic Regression Model
log_reg = LogisticRegression(max_iter=1000, random_state=42)
log_reg.fit(x_train_log, y_train_log)
log_pred = log_reg.predict(x_test_log)

# Evaluate Logistic Regression
log_accuracy = log_reg.score(x_test_log, y_test_log)
log_precision = precision_score(y_test_log, log_pred, average='weighted')
log_recall = recall_score(y_test_log, log_pred, average='weighted')
log_f1 = f1_score(y_test_log, log_pred, average='weighted')
log_conf_matrix = confusion_matrix(y_test_log, log_pred)
```

It is possible to directly compare the Random Forest Regressor's performance to the previously evaluated Linear This Python code snippet separates the price into three groups : low, medium, and high instead of projecting a continuous price number. To begin, discretize the 'price' target variable using scikit-learn's KBinsDiscretizer. Using the n_bins=3 input, three bins are produced; encode='ordinal' assigns an ordered numerical label (0, 1, 2) to each bin; strategy='quantile' ensures that each bin has approximately the same number of samples.

After reshaping with.ravel(), the.fit_transform() function learns the bin edges of the target variable and transforms them into categorical labels that are stored in y_binned.

Then, maintaining an 80/20 split and a constant random_state for repeatability, train_test_split is used to separate the previously encoded features (features_encoded) and the new category target variable (y_binned) into training and testing sets.

A Logistic Regression model is then trained for this categorization task. A LogisticRegression model is launched with random_state=42 and max_iter=1000 to ensure convergence. The encoded training features (x_train_log) and the binned training target (y_train_log) are used to train the model using the.fit() technique. The encoded test characteristics (x_test_log) are analysed wth the help of using the.predict() method, and the results are written in log_pred.

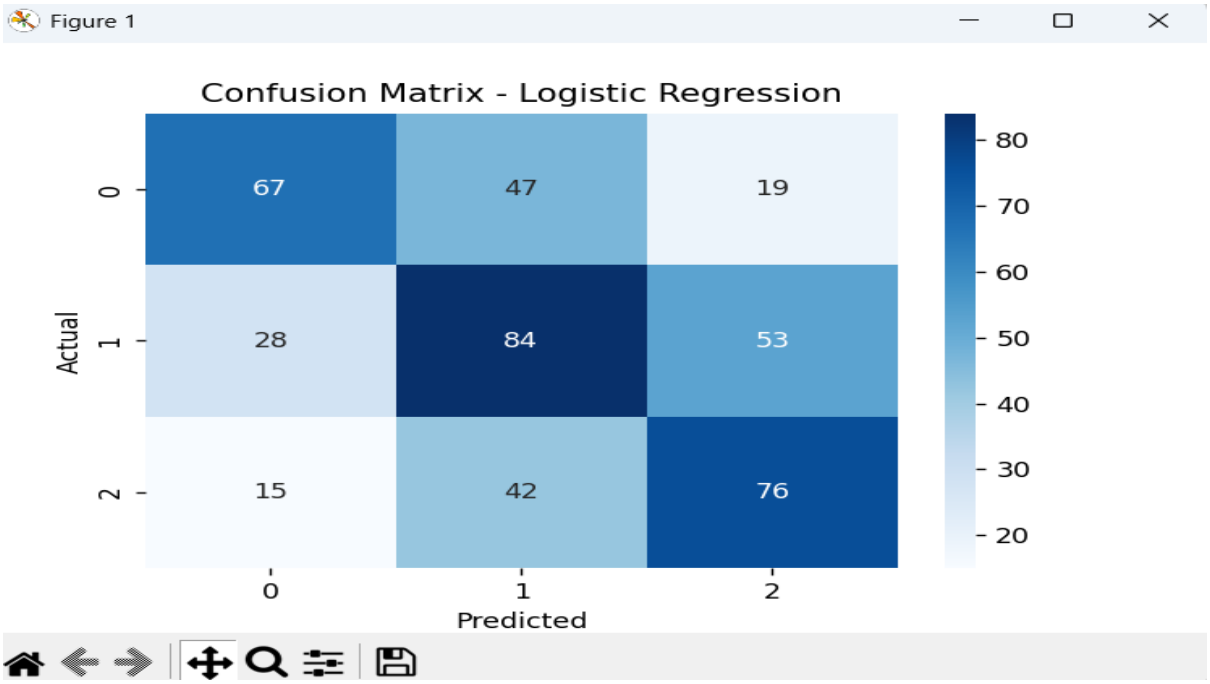
Finally, the result of the Logistic Regression classifier is calculated using several classification measures. The total accuracy of the model is calculated by log_accuracy. Precision_score, recall_score, and f1_score are computed with the help of a 'weighted' average to account for potential class imbalance. A confusion_matrix, which makes the proportions of true positives, true negatives, false positives, and false negatives for each class, is also generated to provide a comprehensive examination of the model's predictions. These metrics give us a comprehensive evaluation of the Logistic Regression algorithm's ability to categorize player market values into the appropriate groups.

```
print('\n--- Logistic Regression (Categorical Price Prediction) ---')
print(f'Accuracy: {log_accuracy:.4f}')
print(f'Precision: {log_precision:.4f}')
print(f'Recall: {log_recall:.4f}')
print(f'F1 Score: {log_f1:.4f}')

# Display confusion matrix
plt.figure(figsize=(6,4))
sns.heatmap(log_conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Logistic Regression')
plt.show()

# Compute ROC curve and AUC
y_prob = log_reg.predict_proba(x_test_log)[:, 1]  # Get probability estimates for positive class
fpr, tpr, _ = roc_curve(y_test_log, y_prob, pos_label=1)
roc_auc = roc_auc_score(y_test_log, log_reg.predict_proba(x_test_log), multi_class='ovr')
```

The above code snippet was written to calculate the metrics and performance visualization of the Logistic Regression model trained for category price prediction. The first classification metrics that are written, formatted to four decimal places, are Accuracy, Precision, Recall, and F1-Score. These labels provide a numerical summary of the model's accuracy in classifying player market values into Low, Medium, and High groupings.

The confusion matrix is then analysed with the help of using the seaborn library. A figure of a specified size can be written using Plt.figure(). The sns.heatmap() function generates a heatmap representation of the log_conf_matrix with the help of annot=True showing the counts within each cell, fmt='d' formatting the annotations as integers, and cmap='Blues' defining the color scheme. The labels for the exact and actual categories are established with the help of using plt.xlabel() and plt.ylabel(), respectively, and the plot is given a title using plt.title(). Finally, using plt.show(), the generated confusion matrix is displayed, creating a graphic depiction of the model's classification performance over the different price points.

The Receiver Operating Characteristic (ROC) curve and Area Under the Curve (AUC) are then calculated by the code. The probability calculated for the positive class are first obtained using log_reg.predict_proba (assuming that class '1' reflects one of the categories, most likely 'Medium' according to normal ordinal encoding).(x_test_log). The scikit-learn roc_curve() algorithm then calculates the false positive rates (fpr), true positive rates (tpr), and thresholds based on the true labels (y_test_log) and the projected probability (y_prob). Finally, the roc_auc_score() function computes the AUC score using the true labels and the forecasted probabilities. The multi_class='ovr' parameter indicates a One-vs-Rest method for solving the multi-class classification problem for calculating the AUC. This part of the code evaluates the model's ability to discriminate between the different price ranges.



Matrix Breakdown: In each cell (i, j), the confusion matrix shows the number of observations that are actually in class I but were projected to be in class J.

Rows = Real classes;

Columns = Classes Predicted;

Matrix Content:

Predicted: 0

Predicted: 1

Predicted: 2

Actual: 0 47 19

Actual: 1 28 84 53

Actual: 2 15 42 76

The main interpretation can be summarised as

• Correct Predictions (Diagonal cells):

i. Class 0 correctly predicted: 67
ii. Class 1 correctly predicted: 84
iii. Class 2 correctly predicted: 76

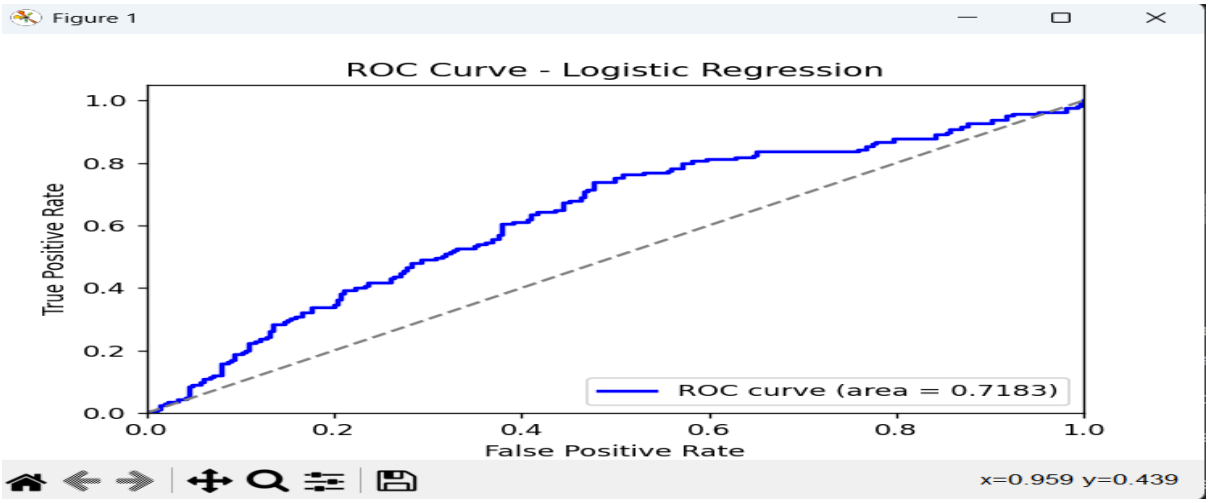
Misclassifications (off-diagonal cells):

i. Class 0 incorrectly identified as class 1: 47
ii. Class 0 incorrectly identified as class 2: 19
iii. Class 1 incorrectly identified as class 0: 28
iv. Class 1 incorrectly identified as class 2: 53

v. Class 2 was misidentified as class 0: 15
vi. class 2 was misidentified as class 1: 42

Estimate of Overall Accuracy:
Total number of accurate forecasts = 67 + 84 + 76 = 227
Sum of all values = 67 + 47 + 19 + 28 + 84 + 53 + 15 + 42 + 76 = 431 is the total number of forecasts.
Therefore, the approximate accuracy is 227 / 431 ≈ 52.7%.

```
# Plot ROC Curve
plt.figure(figsize=(6,4))
plt.plot( *args: fpr, tpr, color='blue', lw=2, label=f'ROC curve (area = {roc_auc:.4f})')
plt.plot( *args: [0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Logistic Regression')
plt.legend(loc="lower right")
plt.show()
```



The percentage of true negatives that were mistakenly labeled as positives is known as the X-Axis (False Positive Rate, or FPR).

FPR is equal to the sum of the false positives and true negatives. Recall or sensitivity are other names for the Y-Axis (True Positive Rate, or TPR). False Negatives + True Positives < True Positives = TPR
The Blue Curve plots the TPR against the FPR for a range of classifier probability output threshold values.

In the picture above, the gray diagonal line shows the random classifier's ROC curve (AUC = 0.5). Above this threshold, any classifier outperforms chance.

- The AUC, which measures classification performance well, is 0.7183.

- The likelihood that the model will rank a randomly selected positive instance higher than a randomly selected negative instance is known as the AUC.
- This is a rough interpretation scale:
    i. 0.90 to 0.90 = Very excellent
    ii. 0.90 to 1.0 = Excellent
    iii. 0.70–0.70 = Good;
    iv. 0.60–0.50 = Poor;
    v. 0.50 or less = Worse than random;
    vi. 0.70–0.60 = Fair

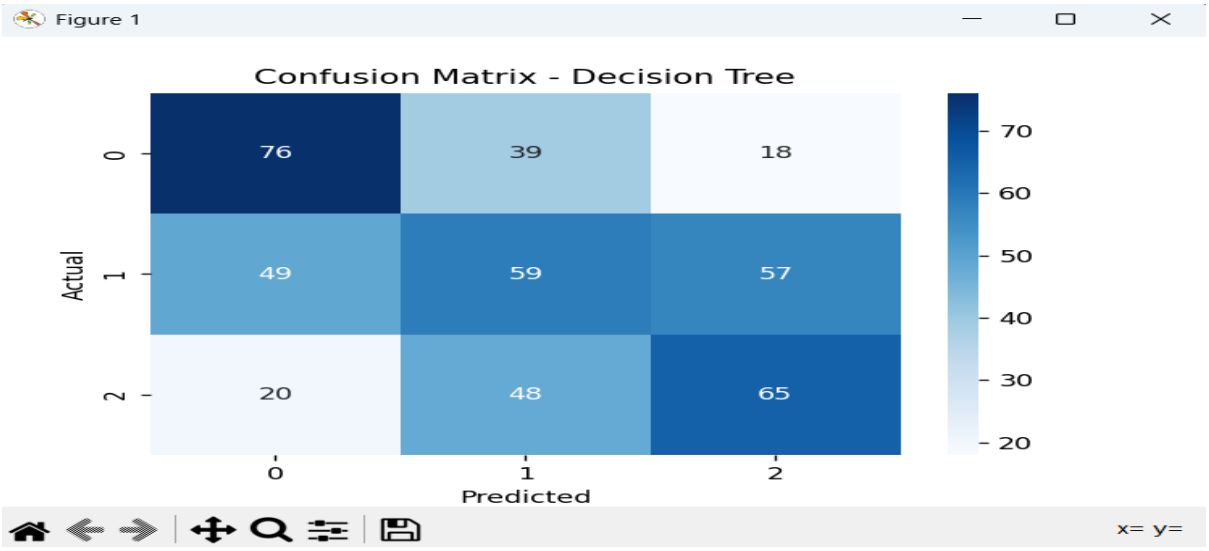
Therefore, this logistic regression model has strong discriminatory ability—it is effectively differentiating between the positive and negative classes—with an AUC of 0.7183.

```
# Convert continuous price predictions into categories (Low, Medium, High)
bin_discretizer = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='quantile')
y_train_class = bin_discretizer.fit_transform(y_train.reshape(-1, 1)).ravel()
y_test_class = bin_discretizer.transform(y_test.reshape(-1, 1)).ravel()

# --- Train Classification Models (Decision Tree, Random Forest, XGBoost, Logistic Regression) ---
dt_clf = DecisionTreeClassifier(random_state=42)
rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)
xgb_clf = xgb.XGBClassifier(objective='multi:softmax', num_class=3, eval_metric='mlogloss', random_state=42)

# Fit models
dt_clf.fit(x_train, y_train_class)
rf_clf.fit(x_train, y_train_class)
xgb_clf.fit(x_train, y_train_class)

# Make Predictions
dt_pred = dt_clf.predict(x_test)
rf_pred = rf_clf.predict(x_test)
xgb_pred = xgb_clf.predict(x_test)
```

```python
# Store models & predictions
models = {
    "Decision Tree": (dt_clf, dt_pred),
    "Random Forest": (rf_clf, rf_pred),
    "XGBoost": (xgb_clf, xgb_pred),
}


# Function to compute & print evaluation metrics
1 usage
def evaluate_model(name, y_true, y_pred):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')
    conf_matrix = confusion_matrix(y_true, y_pred)

    print(f"\n--- {name} ---")
    print(f'Accuracy: {accuracy:.4f}')
    print(f'Precision: {precision:.4f}')
    print(f'Recall: {recall:.4f}')
    print(f'F1 Score: {f1:.4f}')
```

```
    # Confusion Matrix Visualization
    plt.figure(figsize=(6, 4))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(f'Confusion Matrix - {name}')
    plt.show()

    return accuracy, precision, recall, f1, conf_matrix


# Compute and display metrics for all models
results = {}
for model_name, (model, y_pred) in models.items():
    results[model_name] = evaluate_model(model_name, y_test_class, y_pred)

# --- ROC Curves ---
plt.figure(figsize=(8, 6))

for model_name, (model, y_pred) in models.items():
    y_prob = model.predict_proba(x_test)  # Get probability estimates
    fpr, tpr, _ = roc_curve(y_test_class, y_prob[:, 1], pos_label=1)
    roc_auc = roc_auc_score(y_test_class, y_prob, multi_class='ovr')
    plt.plot( *args: fpr, tpr, label=f'{model_name} (AUC = {roc_auc:.4f})')
```

```
plt.plot( *args: [0, 1], [0, 1], color='gray', linestyle='--')  # Random guess line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Classification Models')
plt.legend(loc="lower right")
plt.show()
```



Overview of the Matrix:

Forecasted: 0

Expected: 1

Expected: 2

Real:	0	76	39	18
Real:	1	49	59	57
Real:	2	20	48	65

Accurate Forecasts (Diagonal Cells):

- 76 Class 0 predictions were accurate.
- Class 1 made the right prediction: 59
- Class 2 made the right prediction: 65

Misclassifications (cells that are not diagonal):

• Class 0 was miscalculated as follows:

i. Class 1: 39 times
ii. Class 2: 18 times

• Class 1 was miscalculated as follows:

i. Class 0: 49 times
ii. Class 2: 57 times

• Class 2 was miscalculated as follows:

i. Class 0: 20 times

ii. Class 1: 48 times

Total Forecasts & Estimated Accuracy:

Total number of accurate forecasts = 76 + 59 + 65 = 200

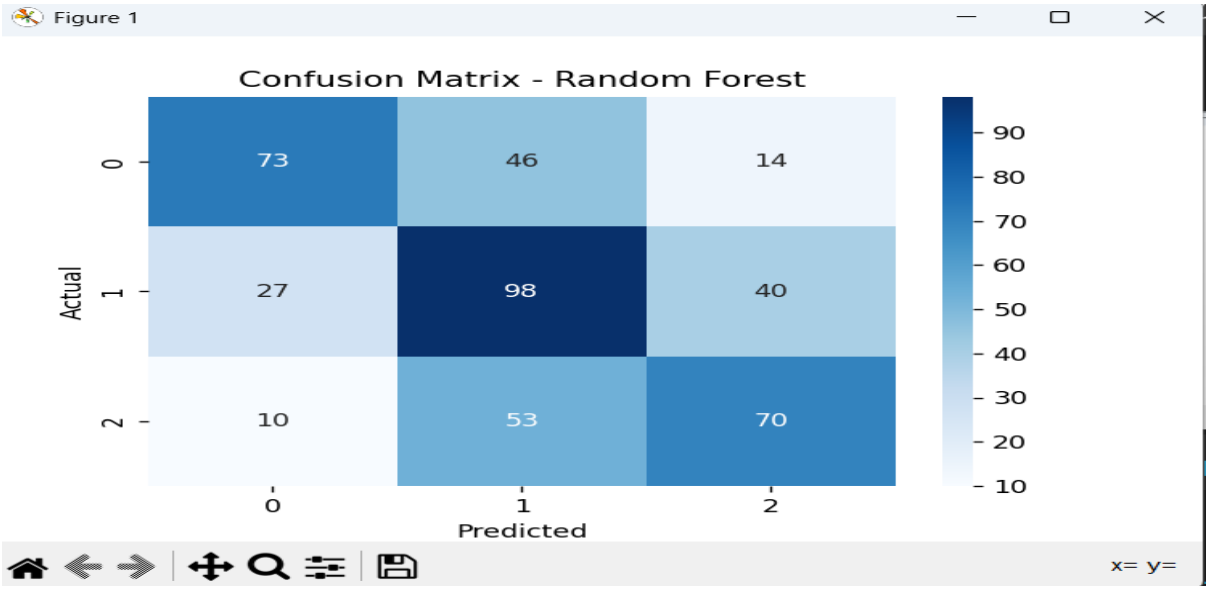
The sum of all values for all samples is equal to 76 + 39 + 18 + 49 + 59 + 57 + 20 + 48 + 65 = 431.
200 / 431 = 46.4% accuracy

Compared to the prior Logistic Regression model, which had ~52.7%, this is lower.

Analysis and insights:

- Class 0 is quite well predicted by the Decision Tree model, whereas class 1 exhibits substantial misclassification.
- Overfitting or class overlap could be happening, which is a common problem with Decision Trees when they are not appropriately tuned or pruned.
- For greater generalization, you may wish to:

  i. Use feature selection/engineering
  ii. Pruning the tree (limit depth, min samples split)
  iii. Attempt ensemble models such as Random Forest or Gradient Boosting.



Matrix Dissection:

Forecasted: 0

Expected: 1

Expected: 2

Real:	0	73	46	14
Real:	1	27	98	40
Real:	2	10	53	70

Diagonal Value Correct Predictions:

i. Class 0 was accurately predicted to be 0: 73,
ii. Class 1 was accurately predicted to be 1: 98.
iii. Class 2 accurately forecasted 2: 70

Errors in classification (values that are not diagonal):

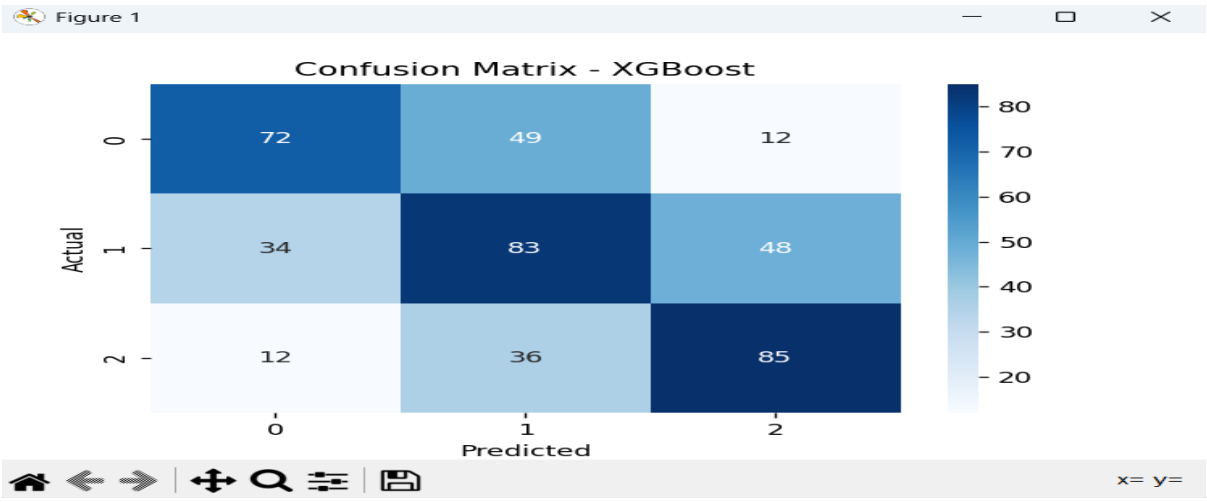
i. Class 0 is frequently misidentified as 1: 46.
ii. Class 0 is 14 times misidentified as 2.
iii. Class 1 was misidentified as 0:27 times.
iv. Class 1 is frequently misidentified as 2: 40.
v. Class 2 is frequently misidentified as 0: 10.
vi. Class 2 was misidentified as 1: 53 53 times.
vii. Total Forecasts and Estimated Accuracy:

Total number of accurate forecasts = 73 + 98 + 70 = 241
Accuracy ≈ 241 / 431 ≈ 55.9%
Total samples = total of all matrix values=73+46+14+27+98+40+10+53+70=431

Findings:

- Out of the three models (Random Forest, Decision Tree, and Logistic Regression), Random Forest performs the best.
- It does particularly well for class 1, with 98 accurate predictions.
- Nevertheless, class 2 is commonly mistaken for class 1, which may indicate the need for:

i. Improved feature separation;
ii. class rebalancing in the event of unbalanced data;
iii. hyperparameter adjustment to increase class 2 precision



Overview of the Matrix

Forecasted: 0

Expected: 1

Expected: 2

Real:	0	72	49	12
Real:	1	34	83	48
Real:	2	12	36	85

Accurate Forecasts (Diagonal Cells):

- 72 was the correct prediction for class 0.
- Class 1 made an accurate prediction of 83
- Class 2 made an accurate prediction of 85

On-diagonal misclassifications:

- Class 1 is 0: 34, as 2: 48;
- Class 2 is 0: 12, as 1: 36;
- Class 0 is 1: 49, as 2: 12.

The biggest misunderstanding appears to be:

- Class 1 is often misclassified as both 0 and 2,
- Class 2 is commonly misclassified as 1,
- Class 0 is frequently misclassified as 1.

Model Estimate of Accuracy:

Predictions correct = 72 + 83 + 85 = 240

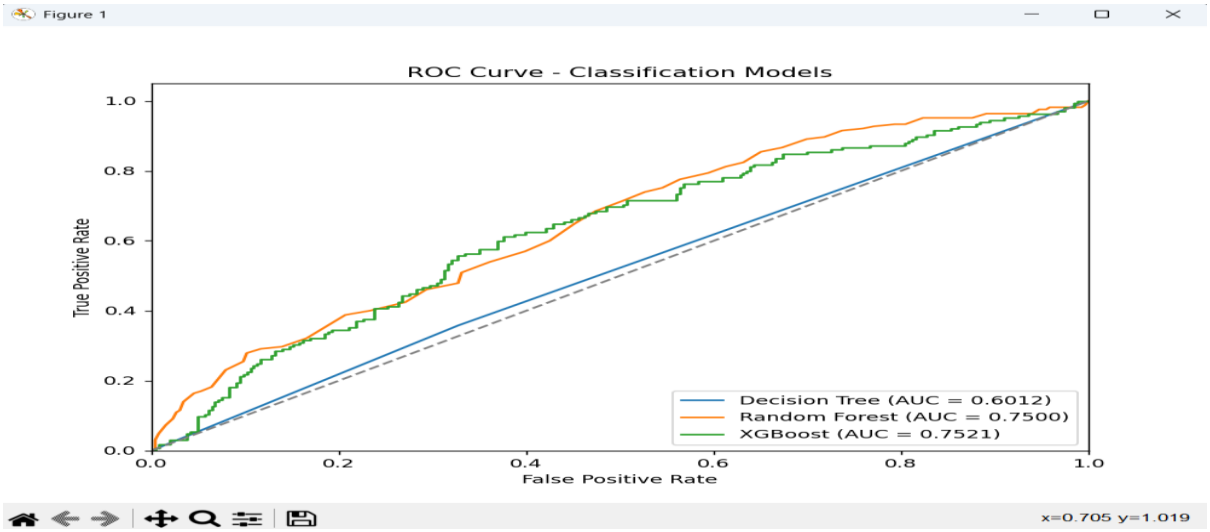
All cells added together equals 72 + 49 + 12 + 34 + 83 + 48 + 12 + 36 + 85 = 431. 240 / 431 = 55.7% accuracy

This is superior to Decision Tree (~46.4%) and nearly identical to Random Forest (~55.9%).

• ~52.7% for logistic regression

Findings:

- Overall, the XGBoost model is doing fairly well, particularly when it comes to accurately predicting class 2.
- Classes 0 and 1 and 1 and 2 are still confused, which may indicate overlapping characteristics or perhaps a class imbalance.
- In contrast with Random Forest:
  i. A little less accurate
  ii. Class 2 prediction is better,
  iii. The misclassification trend is similar.



The ROC Curve's purpose

The ROC curve plots the following to assess the performance of a classification model:

- Sensitivity = Recall = True Positive Rate (TPR)
- Specificity - False Positive Rate (FPR) = 1.

A distinct threshold that is utilized to translate probability into class predictions is represented by each point on the curve.

False Positive Rate (FPR) is the X-Axis.

True Positive Rate (TPR) on the Y-Axis Curves

- AUC for Blue (Decision Tree) is 0.6012.
- Random Forest's Orange — AUC = 0.7500
- AUC for Green (XGBoost) is 0.7521.
- Gray Diagonal Line: Random classifier baseline (AUC = 0.5)

Values for AUC (Area Under Curve)

Model Interpretation of AUC Score

XGBoost 0.7521The best performance overall

Random Forest 0.750

XGBoost 0.7521

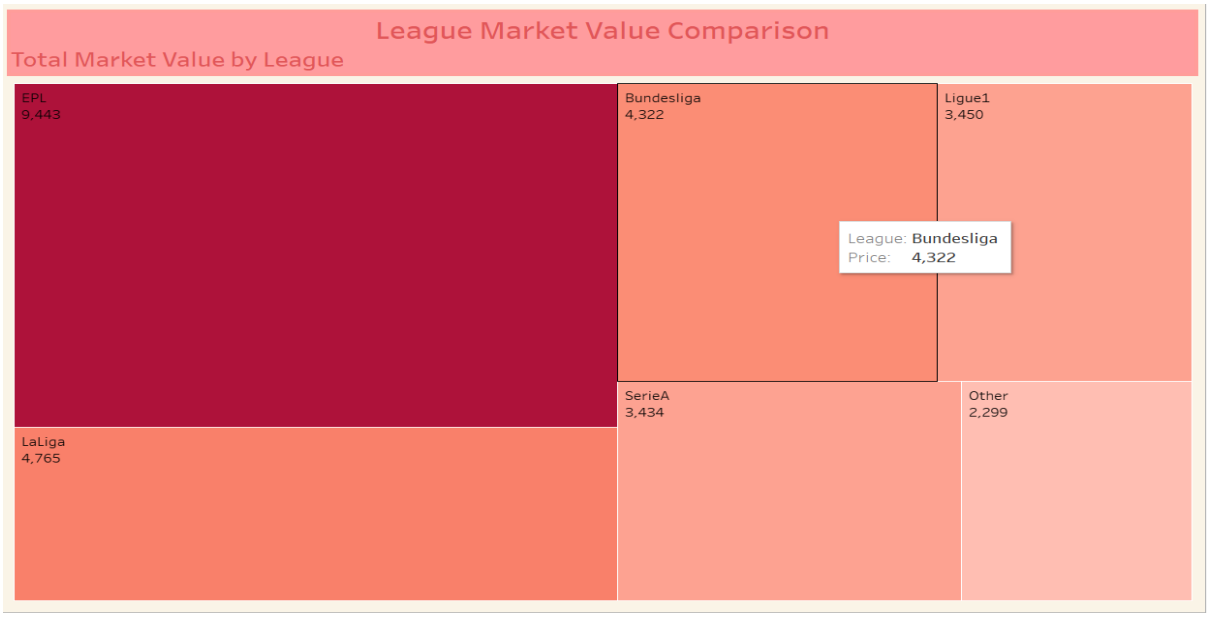
Decision Tree 0.6012 is quite near AUC, which gauges the model's capacity to differentiate between classes, is noticeably lower:

• 0.5 = random guessing < 0.5 = poorer than random < 1.0 = perfect categorization

Interpretation and Perspectives

- Random Forest exhibits great classification abilities,

- whereas XGBoost has the best ROC performance, with a small advantage.
- Despite being practical, decision trees are much less accurate than ensemble models.The distinction between Decision Tree and the other two demonstrates how advantageous ensemble approaches are for classification tasks, particularly when dealing with intricate datasets.



"League Market Value Comparison" is the title.

"Total Market Value by League" is the subtitle.

A football league is represented by each block. Each block's area represents the league's overall market value. Value intensity is also represented by color (greater value = deeper color).

Important Points to Note:

Market Value of the League The English Premier League, or EPL=9,443

La Liga(Spain) =4,765

Germany's Bundesliga= 4,322

France's Ligue =13,450

Italy's Serie A3,=434

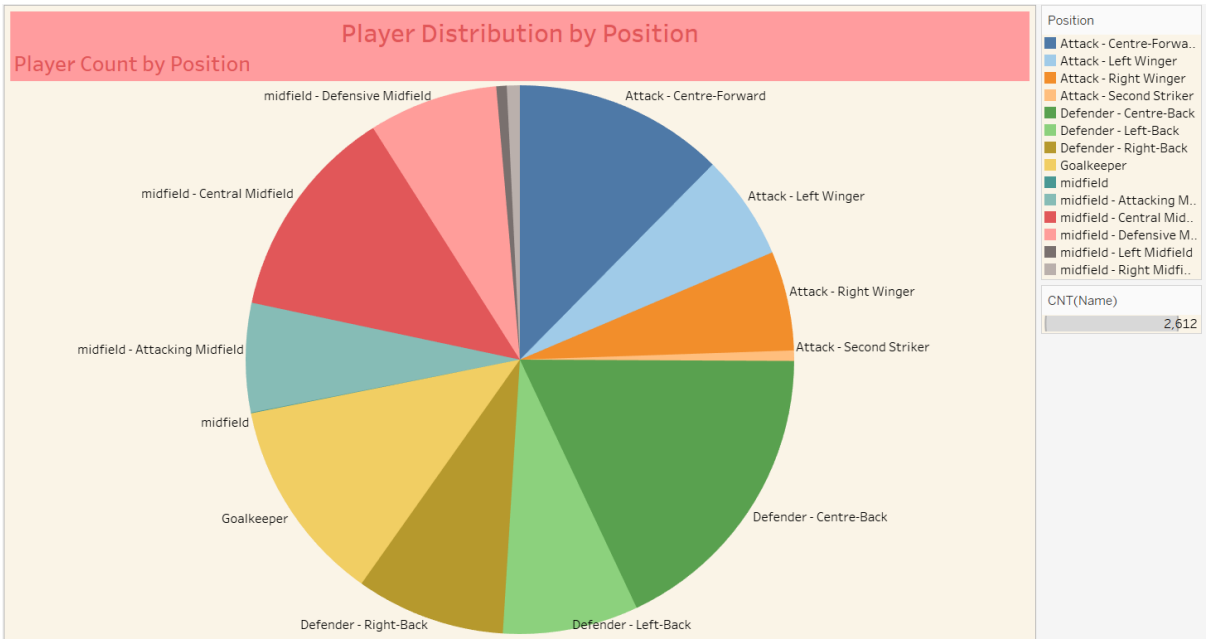
Other (every league else)=2,299

Conclusions:

1. The Premier League dominates the chart, holding the largest overall market value by a significant margin, highlighting its power in the global economy and concentration of elite

talent.
2. The Bundesliga and La Liga come next, although they are worth almost half as much as the Premier League.

3. With a moderate share each, Ligue 1 and Serie A are extremely similar in terms of total value.
4. Despite having a large number of leagues, the "Other" category has the lowest total market value.



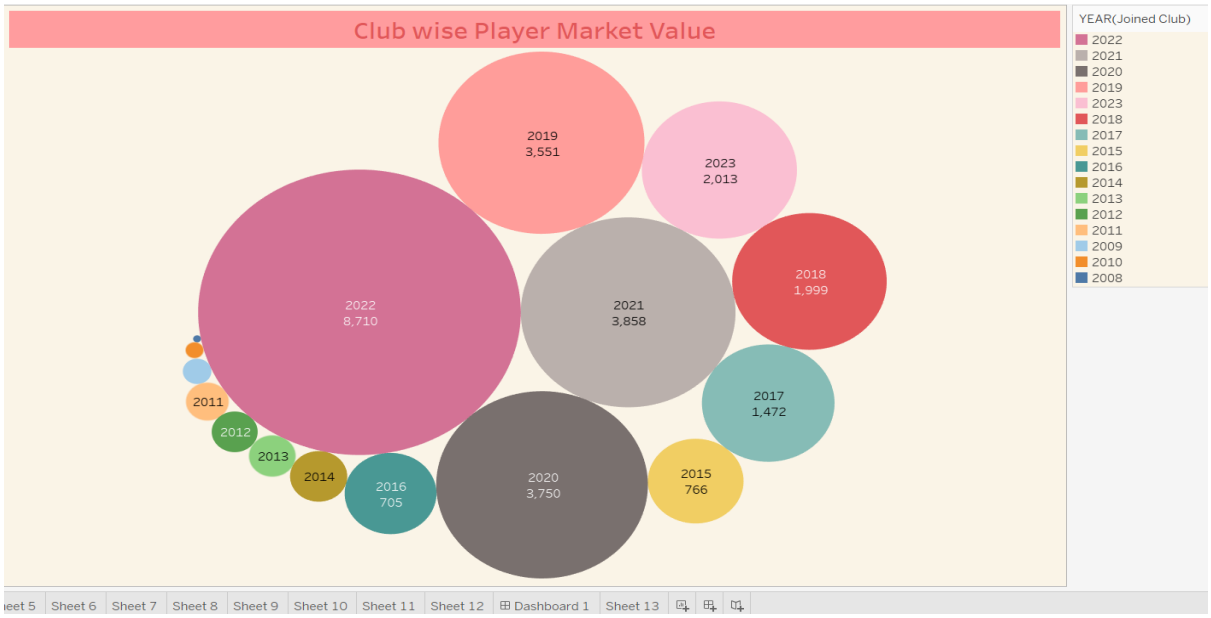
Explanation of different components of above pie charts -

Title: "Player Distribution by Position" makes the chart's objective quite apparent.

Slices: A different player position is represented by each colored slice. A key that associates each hue with a particular location is provided by the legend on the right.

Slice Size: Each slice's area is proportionate to the number of players occupying that place. More players in that role are represented by larger slices, whilst fewer players are represented by smaller slices.

Legend: All of the player positions that are represented are listed in the legend on the right, along with the colors that correspond to each position. Additionally, it displays the total number of players in the data (CNT(Name) = 2,612).



Explanation of different components of above pie charts -

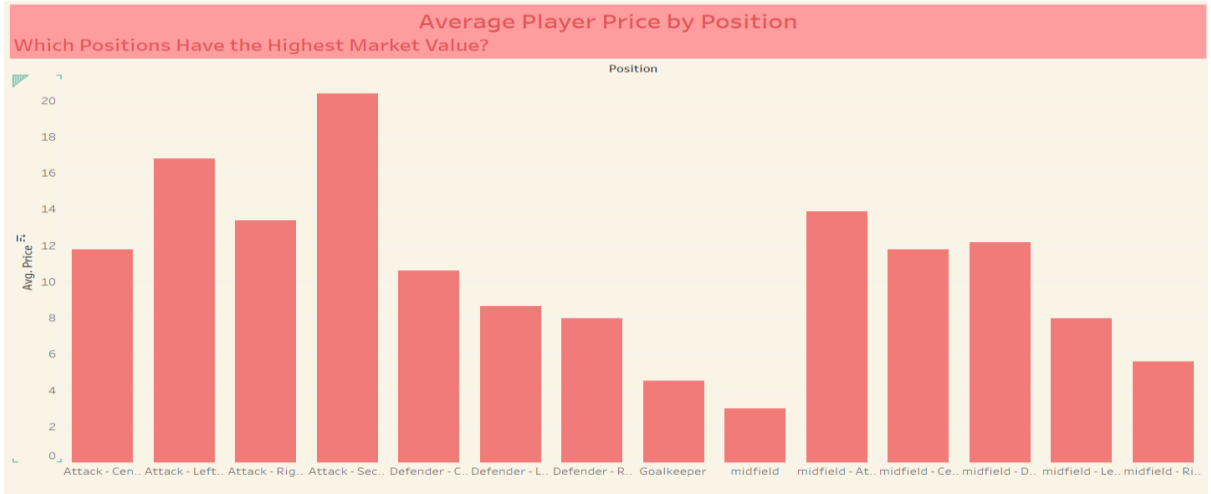
Title: The chart's representation of "Club wise Player Market Value" is evident.

Bubbles: A club is represented by each bubble.

Bubble Size: Each bubble's size (area) corresponds to the total market worth of all players that joined that club during the years in question. Greater market value is indicated by larger bubbles.

Color of Bubbles: Each bubble's color corresponds to the year that the players inside it joined the team. Each year, from 2008 to 2023, has a color key in the legend on the right.

Labels: The year (or a range of years, although it appears to be a single year here) and a number value are written on each bubble. The overall market worth (in some unit, albeit not stated explicitly) for players who joined that particular year and are currently with that club is probably represented by this figure.



Explanation of different components of above pie charts -

Title: "Average Player Price by Position" makes the chart's objective quite apparent. The subject that the graph seeks to address is stated in the subtitle, "Which Positions Have the Highest Market Value?"

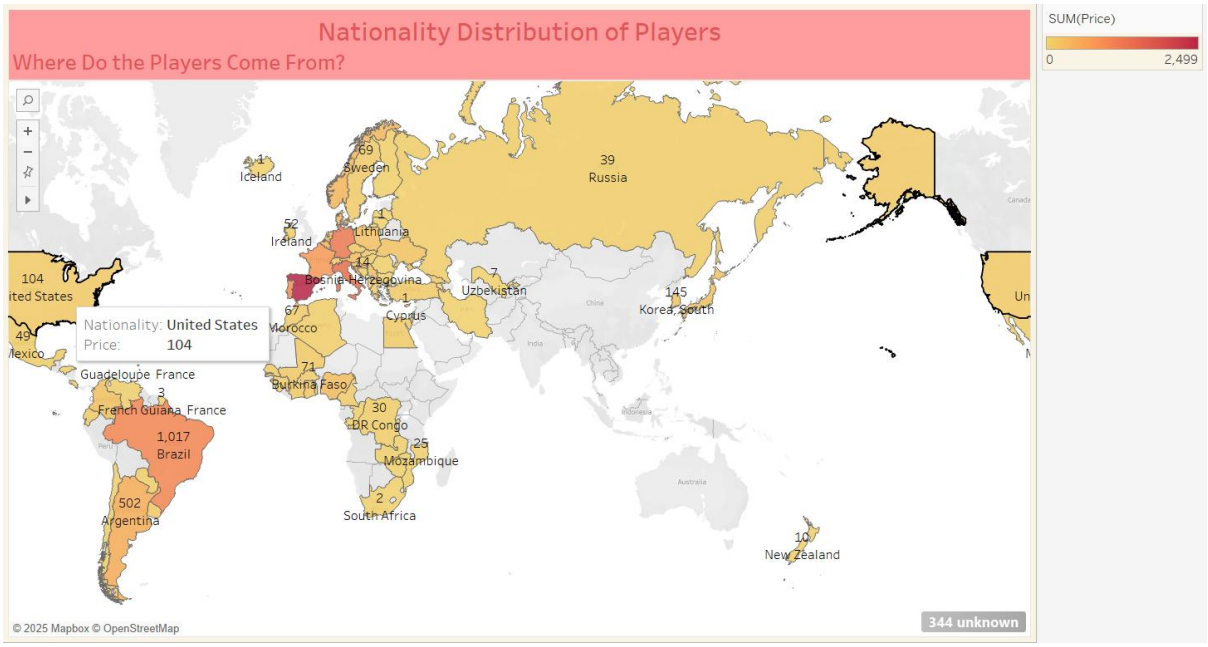
Y-axis, or the vertical axis: This axis, which is labeled "Avg. Price," shows the average market worth of players; values range from 0 to 20 (in an undefined unit, such as millions of dollars or euros).

The horizontal axis (X-axis), which is labelled "Position," shows the various positions of the players. Although they are condensed, we can still deduce these:

- Attack - Left (Attack - Left Winger)
- Attack - Cen. (Attack - Center-Forward)
- Attack - Sec. (Attack - Second Striker)
- Attack - Rig. (Attack - Right Winger)
- Defender - C. (Center-Back)
- Defender - L. (Left-Back)
- Defender - R. (Right-Back) respectively
- The goalkeeper
- A midfielder
- Midfield - At. (attacking midfielder)
- midfield - Ce. (central midfielder)
- midfield - D. (defensive midfielder)
- midfield - Le. (left midfielder)
- midfield - R. (right midfielder)

Bars: A player's position is indicated by each vertical bar.

Bar Height: The average market value of players in a certain position is represented by the height of each bar. A greater average market value is indicated by taller bars, and a lower average market value is indicated by shorter bars.



The objective of the map is made evident by the heading "Nationality Distribution of Players" and the subtitle "Where Do the Players Come From?"

globe Map: A globe map with several nations highlighted serves as the graph's base.

The orange color's intensity for each nation is a representation of the overall market value (SUM(Price)) of players from that nation. A larger total market value of players from that nation is indicated by darker orange hues. A lower total market value is indicated by lighter hues. There are probably no players of such nationality in the dataset for countries that are colorless.

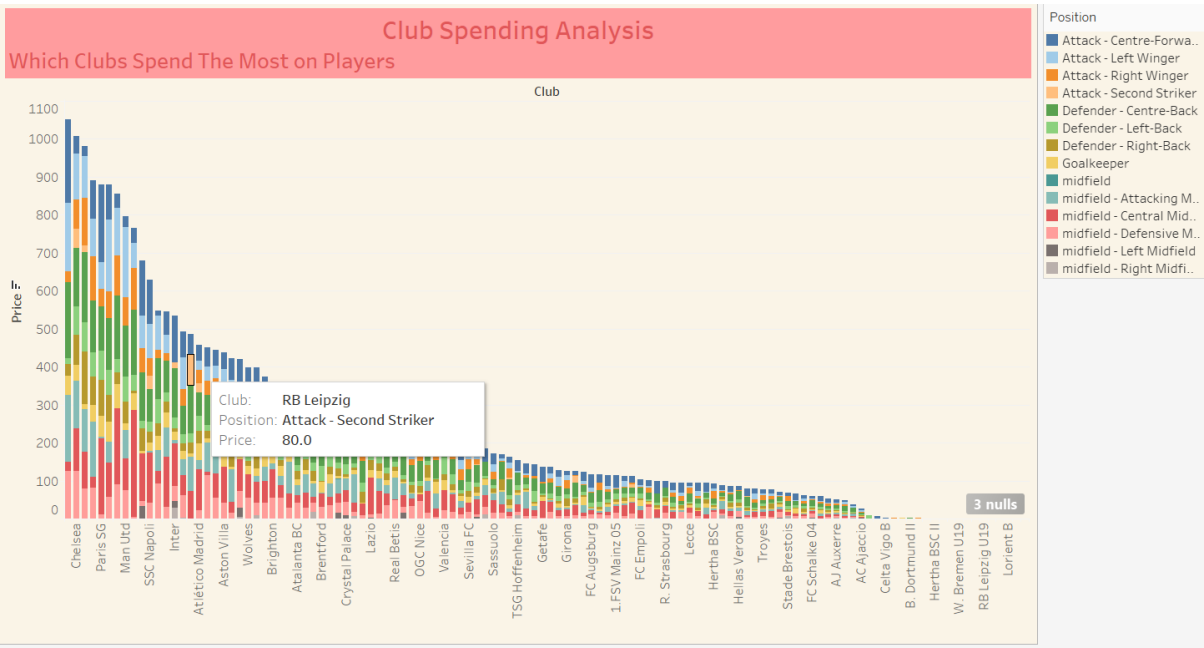
Labels with numbers: Some nations have labels with numbers. These figures most likely reflect the amount of players from that particular nation that are included in the dataset.

Legend: The bright orange to dark orange color gradient in the upper right corner of the legend represents a range of "SUM(Price)" values (from 0 to 2,499). This aids in comprehending the market value scale that the color intensity represents.

Interactive Elements: The zoom and pan buttons on the left imply that this may be an interactive map that lets viewers delve deeper into various areas.

Details on Hover (Implied): A box displaying the following information displays when you hover over a particular nation (as in the case of the "United States" tooltip):

- Nationality: The nation's name.
- Price: The sum of the players from that nationality's market value (SUM(Price)).
- Count (Implied): The number of players from that nationality is probably indicated by the country's numerical identifier.
- "344 unknown": This indicates that there are 344 players in the dataset whose nationality is either unspecified or not mappable, as indicated at the bottom right.



It is evident from the heading "Club Spending Analysis" and the caption "Which Clubs Spend the Most on Players" what the chart is meant to show.

Club spending is represented on the vertical axis (Y-axis), "Price £," which ranges from 0 to 1100 (probably in British pounds).

Horizontal Axis (X-axis): This axis, which is labeled "Club," lists several football teams. The clubs' total spending is arranged in descending order from left to right.

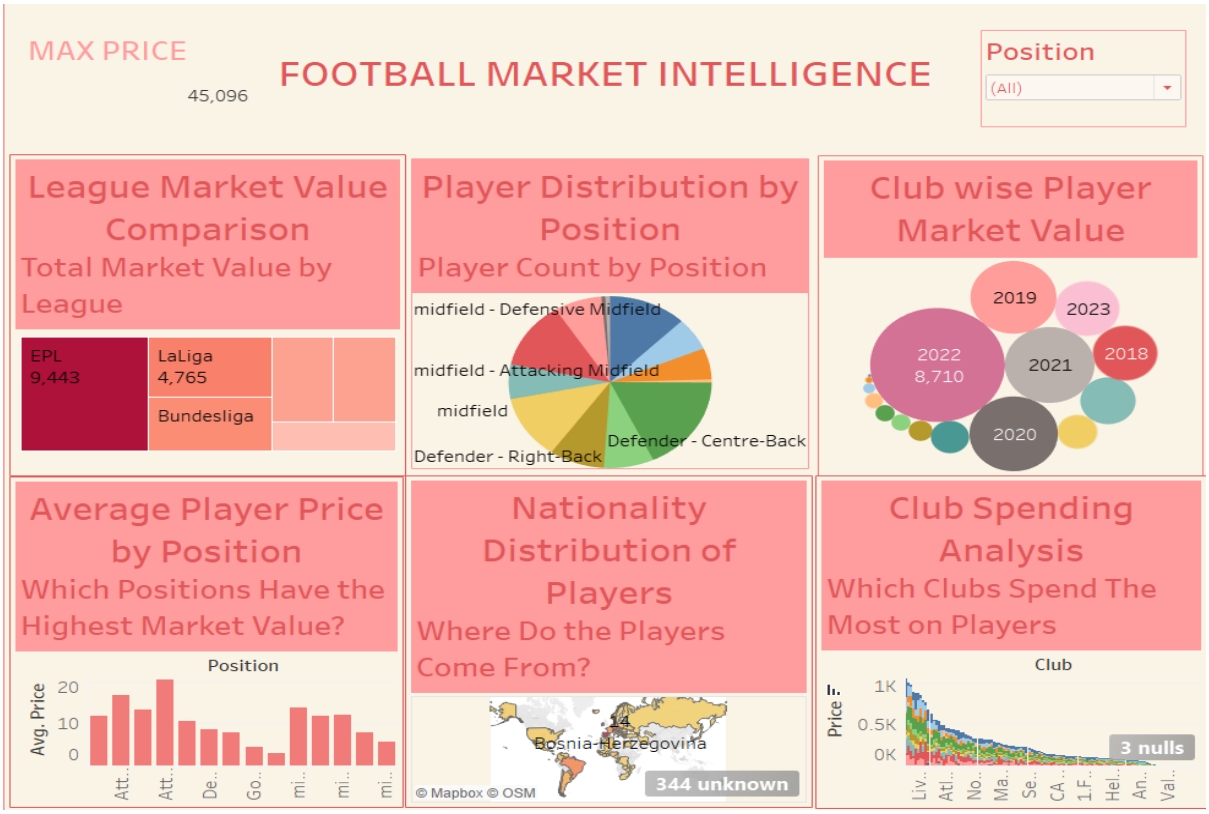
Stacked Bars: Every bar stands for a different football team. The bar's overall height represents the total amount of money that club has spent on players.

Color Segments: The amount that the club spends on players in a specific position is represented by each colored segment inside a bar. A key connecting each hue to a particular player position is provided by the legend on the right (for example, orange for Attack-Left Winger, blue for Attack-Center-Forward, etc.).

Tooltip (For instance): A tooltip displays the following when you hover over a section of a bar, as in the case of RB Leipzig:

- Club: RB Leipzig is the club's name.
- Position: The segment's representation of the player position (Attack - Second Striker).
- Price: The sum of money the club spends on players in that particular position (in this case, £80.00).

"3 nulls": This indicates that there are three data points at the far right of the x-axis where the club information is either absent or not provided.



Bibliography

- https://pandas.pydata.org/
- https://scikit-learn.org/stable/
- https://matplotlib.org/
- https://seaborn.pydata.org/
- https://en.wikipedia.org/wiki/Supervised_learning
- https://en.wikipedia.org/wiki/Regression_analysis
- https://en.wikipedia.org/wiki/Statistical_classification
- https://www.kaggle.com/
- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html
- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.KBinDiscretizer.html
- https://en.wikipedia.org/wiki/Ensemble_learning
- https://www.coursera.org/learn/machine-learning