Business Intelligence - ADSA H6012 (2024-25) Assignment 2 report

Mariem Nsiri
MSc in Applied Data Science and Analytics
Technological University Dublin

This report, developed within a local Jupyter Notebook environment, provides a comprehensive assessment of Business Intelligence module labs using a relevant dataset. It offers a detailed analysis of the selected datasets, evaluating their suitability for implementing various data processing, transformation, and analytical techniques. The report emphasizes the key stages of data profiling, quality assessment, and the selection of relevant attributes for in-depth analysis. Additionally, it addresses the ETL (Extract, Transform, Load) process, encompassing essential steps such as data cleansing, integration, and aggregation. Ultimately, the report aims to demonstrate how the processed data can be effectively visualized using Python plotting libraries and Tableau toolbox, highlighting the value and insights generated through dashboards, and showcasing the practical application of the methods employed.

The report begins with an introduction to the background context in Section 1, offering an overview of the theoretical background of the module. It then moves on to assess the selected dataset's suitability for applying various data processing and analysis techniques in Section 2, examining its structure and key features. Following this, the data quality is evaluated through a profiling process in Section 3, which highlights the strengths and weaknesses of the data, along with the selection of relevant attributes for further analysis. The report then shifts to the ETL process in Section 4, where the focus is on cleansing, integrating, and aggregating the data to ensure it is in the optimal format for analysis. Finally, the report concludes with a discussion of how the processed data is visualized using Tableau dashboard plots in Section 5.

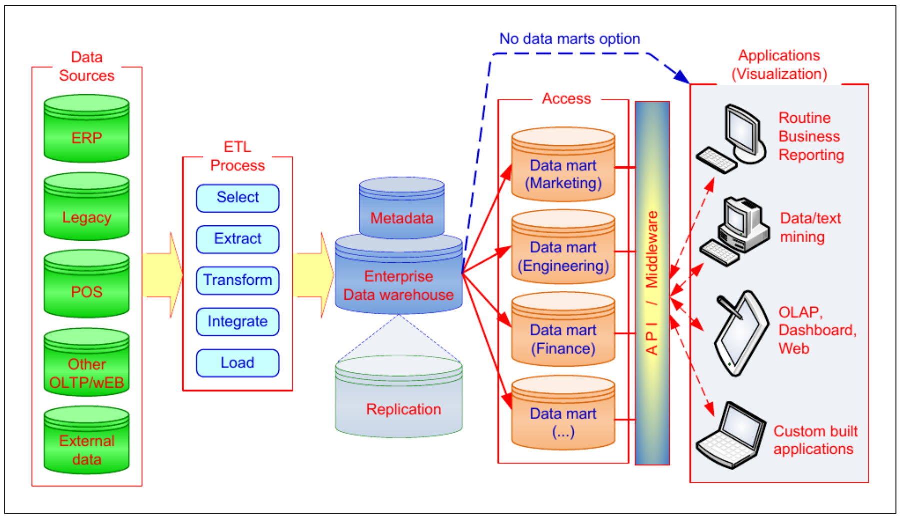


Figure 1: Data warehousing.

1 Background context

In today's data-driven business world, data warehouses play a crucial role in organizing and storing data for decision-making and Business Intelligence (BI). A data warehouse is a central repository where data from different sources across an enterprise is collected, cleaned, and structured. This data is then used for analysis, helping organizations make informed decisions. Key characteristics of a data warehouse include being subject-oriented, integrated, time-variant, non-volatile, summarized, and metadata-driven. These features enable users to efficiently access and analyze historical data. A data mart, which is a smaller, more focused version of a data warehouse, serves the specific needs of a particular department or business unit. There are two main types of data marts: dependent and independent. The dependent data mart is created as a subset of the larger data warehouse, while the independent data mart is developed separately to meet the needs of a specific department (see Figure 1).

The benefits of data warehousing extend beyond data storage. It improves data accuracy, provides a consolidated view of data, and enhances overall system performance. These advantages help organizations make better decisions, improve customer service, and gain a competitive edge.

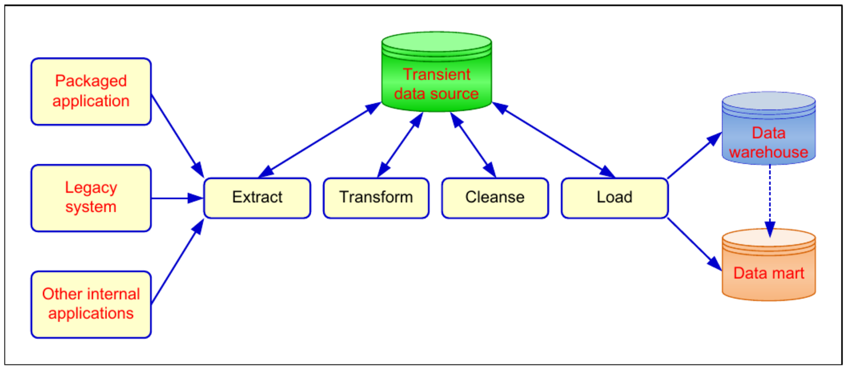


Figure 2: ETL process.

Data integration is another critical element in BI, where data from various sources is combined and prepared for analysis. The process involves accessing, federating, and capturing changes in data sources. One of the most important processes in data integration is the ETL (Extraction, Transformation, and Load) process. This process involves extracting data from different systems, transforming it to meet the needs of the data warehouse, and loading it into the warehouse for analysis (see Figure 2).

Given this background on data warehousing and data integration, the next step is to focus on selecting an appropriate dataset that can effectively showcase the methods and tools introduced. In the following section, the suitability of the selected dataset for demonstrating data quality profiling, ETL process, and BI dashboarding techniques will be discussed.

2 Dataset selection: suitability for showcasing methods

The NBA dataset from Kaggle (see link) has been selected for this assignment due to its comprehensive structure and rich data, making it ideal for showcasing a variety of BI techniques. From a data quality and profiling perspective, this dataset offers clear, structured, and detailed records of basketball players, teams, and game statistics, which allows for effective data cleaning, validation, and profiling activities. The dataset's consistency and format enable the identification of missing values, outliers, and patterns, which are crucial steps in ensuring high data quality for analysis.

Moreover, the dataset is well-suited for ETL processes. The data can be easily extracted, transformed, and integrated with other relevant datasets, such as player performance over time or team statistics, to provide a deeper analysis. By applying ETL techniques, new insights can be generated, and the dataset can be enriched to enhance its analytical potential. In terms of dashboarding, the NBA dataset provides a variety of metrics, such as player scores, team rankings, and game results, which are perfect for creating interactive and insightful visualizations. The dataset's breadth allows for the development of BI dashboards that can track trends, compare performances, and present key metrics in an easily understandable format. This makes the dataset a suitable choice for demonstrating how BI tools, such as Tableau, can be used to visualize and analyze sports data, ultimately aiding decision-making in the sports industry.

3 Data quality assessment and profiling

In this section, we focus on assessing data quality and profiling, important steps to make sure the data used in BI applications is accurate, reliable, and suitable for analysis. In any project, especially those using tools like Tableau, good data quality is essential. Poor-quality data can lead to wrong analysis and decisions, often called "*garbage in, garbage out.*" This is especially important when working with sports data like the NBA dataset, where accurate and complete player stats and game results are crucial for meaningful insights.

Data profiling is the process of assessing the quality of data by examining its structure, content, and relationships. This includes identifying issues such as missing or incorrect values, ensuring data consistency, and verifying data formats. Profiling helps uncover potential problems early in the process, enabling better planning and resource allocation. It involves various activities like descriptive statistics, data type analysis, metadata validation, and inter-table analysis.

While data profiling focuses on assessing the current quality and integrity of the data, the ETL process is concerned with preparing the data for analysis. In Section 4, the ETL process will be explored, focusing on how data is extracted, transformed to meet the assignment's requirements, and loaded into the data warehouse or visualization tool using Tableau Data Prep. Data profiling will lay the foundation for the ETL process by identifying any issues that need to be addressed before proceeding with the transformation and integration of the data.

This section will help provide a solid understanding of how data profiling ensures that the dataset is clean, accurate, and ready for further analysis, setting the stage for the subsequent ETL and dataset integration with Tableau Data Prep.

3.1 Methods, techniques, and tools

In the data profiling step, several methods and techniques will be applied to assess the quality and structure of the data before ETL. Based on Ralph Kimball's steps, data profiling will be used at the start to determine feasibility, correct data issues, and identify unanticipated business rules or relationships. The profiling techniques include:

- Distinct counts to find unique values;
- Null/blank percentages to identify missing data;
- Integrity checks (referential, orphan key) for data consistency;
- Pattern/frequency distributions to detect trends;
- Variance, outlier, and correlation analyses to identify unusual data points and relationships.

For the data profiling analysis, DBeaver (see link) and Python Jupyter will be used. DBeaver is an open-source tool that works well on Debian Linux, making it a great choice for this assignment, as the work environment is based on Linux. Unlike MySQL workbench, which is either difficult to install or takes significant time to set up properly on recent versions of Debian, DBeaver provides a smooth and quick installation process. It is also versatile and supports various databases, making it ideal for the analysis.

The SQLite version of the NBA dataset, which is 2.2 GB in size, will be used for this assignment. This version will be loaded into DBeaver for data profiling. Although 2.2 GB is not a small dataset, the `.sql` script version of the data (automatically generated from the SQLite file using the sqlite3 tool) has a larger size of 3.7 GB. It is expected that loading this `.sql` file into DBeaver will take significant time, as was observed with previous lab exercises. For example, the 21M `tbi1_new.sql` file took a long time to load, even though it was significantly smaller than the 3.7 GB file. In contrast, loading the compressed 2.2 GB SQLite database into DBeaver was much quicker (almost instantaneous) and more efficient. One key advantage of using the SQLite version of the dataset is that there is no need to install or configure a server. The SQLite database is a self-contained file, and we can interact directly with the file without needing any additional database server setup. This makes the process more straightforward and saves time compared to other database systems that require server installations or complex configurations.

Python Jupyter was also used for data profiling because it is widely popular and extremely useful in data science. Jupyter allows Python code to be written and executed in a flexible and interactive way, making it easier to explore and visualize data. With its extensive libraries and support for data manipulation and analysis, Jupyter is an excellent tool for profiling the NBA dataset and performing various quality checks.

3.2 Shortlist of tables and selected attributes

The following tables were selected for analysis: `common_player_info`, `team`, `game`, and `line_score`. These tables were chosen for their relevance and manageability, focusing on key attributes to simplify data integration, transformation, and profiling during the ETL process.

- `common_player_info`: Contains player details such as `person_id`, `first_name`, `last_name`, `birthdate`, `team_name`, and others, key for profiling and linking to other tables;
- `team`: Holds information on teams, including `id`, `full_name`, and `year_founded`, useful for having complete information about teams;
- `game`: This table provides key details about each game, such as `game_id`, `game_date`, `team_id_home`, `team_id_away`, and performance metrics like `pts_home` (points scored by home team), `pts_away` (points scored by the away team), `fg_pct_home` (field goal percentage of home team), and `fg_pct_away` (field goal percentage of away team). These attributes are crucial for understanding the outcome of games, player and team performance, and for linking this data to `common_player_info` and `team` tables;
- `line_score`: This table provides detailed information about scores breakdown for each quarter of a game, including `game_id`, `team_id_home`, `team_id_away`, and quarter scores (`pts_qtr1_home`, `pts_qtr2_home`, etc.). This table is useful for analyzing team performance throughout a game, with quarter-level data enabling detailed profiling.

Several tables were discarded from the analysis due to their limited relevance, size, and complexity, which could complicate the ETL and profiling processes. The `game_summary` table, while containing game status information, does not focus on player or team performance, so it was excluded. The `officials` table, which contains data about game officials, was not relevant to the analysis of team or player performance. The `draft_combine_stats` table, focused on scouting data, was also discarded as it doesn't relate to in-game performance. The `draft_history` table, which details a player's draft history, was excluded as it doesn't provide information on player or team performance in games. The `other_stats` table was deemed redundant due to overlapping data in the selected tables. Lastly, the `play_by_play` table, despite its detailed play-by-play data, was discarded due to its massive size and excessive specificity for the assignment's objectives.

```python
import sqlite3
# Connect to the SQLite database
nba_connection = sqlite3.connect('data/nba_database/nba.sqlite')
# Create a cursor for executing SQL queries
nba_cursor = nba_connection.cursor()
# Execute a query to retrieve 'person_id' from 'common_player_info'
nba_cursor.execute("SELECT person_id FROM common_player_info;")
# Fetch all results from the query.
person_ids = nba_cursor.fetchall()
# Print each 'person_id' value
for identifier in person_ids:
    print(identifier[0])
```

3.3 Profiling to gain insights

Table `common_player_info`

The previous Python code demonstrates how the `nba.sqlite` database can be accessed and queried directly using Python. As mentioned earlier, instead of connecting to a traditional MySQL server, the SQLite database file is interacted with locally. This approach allows SQL queries to be executed in a familiar format to retrieve data, such as the `person_id` column from the `common_player_info` table in this example. Additionally, access to the database is maintained through DBeaver, which offers a graphical interface to explore and query the dataset alongside the Python-based analysis. This dual access ensures flexibility and efficiency during the profiling process.

The following code focuses on profiling the `common_player_info` table by selecting its most relevant attributes. By connecting to the SQLite database, running a SQL query, and loading the data into a Pandas DataFrame, the table is prepared for analysis. Basic commands like `.info()` and `.describe()` are used to understand its structure and summary statistics, laying the foundation for further exploration.

```python
import pandas as pd

# SQL query to select core attributes
query = "SELECT * FROM common_player_info ORDER BY birthdate ASC;"
# Load the data into a pandas DataFrame
df_common_player_info = pd.read_sql_query(query, nba_connection)
# Display basic information about the DataFrame
print("DataFrame info:")
df_common_player_info.info()
# Display descriptive statistics
print("\nDescriptive statistics:")
print(df_common_player_info.describe(include='all'))
# Display missing values if any
print("\nChecking missing values:")
print(df_common_player_info.isnull().sum())
```

```
DataFrame info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3632 entries, 0 to 3631
Data columns (total 33 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   person_id                         3632 non-null   object
 1   first_name                        3632 non-null   object
 2   last_name                         3632 non-null   object
 3   display_first_last                3632 non-null   object
 4   display_last_comma_first          3632 non-null   object
 5   display_fi_last                   3632 non-null   object
 6   player_slug                       3632 non-null   object
 7   birthdate                         3632 non-null   object
 8   school                            3631 non-null   object
 9   country                           3631 non-null   object
 10  last_affiliation                  3632 non-null   object
 11  height                            3632 non-null   object
 12  weight                            3632 non-null   object
 13  season_exp                        3632 non-null   float64
 14  jersey                            3632 non-null   object
 15  position                          3632 non-null   object
 16  rosterstatus                      3632 non-null   object
 17  games_played_current_season_flag  3632 non-null   object
 18  team_id                           3632 non-null   int64
 19  team_name                         3632 non-null   object
 20  team_abbreviation                 3632 non-null   object
 21  team_code                         3632 non-null   object
 22  team_city                         3632 non-null   object
 23  playercode                        3631 non-null   object
 24  from_year                         3632 non-null   float64
 25  to_year                           3632 non-null   float64
 26  dleague_flag                      3632 non-null   object
 27  nba_flag                          3632 non-null   object
 28  games_played_flag                 3632 non-null   object
 29  draft_year                        3632 non-null   object
 30  draft_round                       3486 non-null   object
 31  draft_number                      3434 non-null   object
 32  greatest_75_flag                  3632 non-null   object
dtypes: float64(3), int64(1), object(29)
memory usage: 936.5+ KB
```

```
Descriptive statistics:
       person_id first_name last_name display_first_last  ... draft_round  draft_number greatest_75_flag
count       3632       3632      3632               3632  ...        3486          3434             3632
unique      3632       1291      2308               3608  ...          18           155                2
top       100263       John  Williams      Charles Smith  ...           1     Undrafted                N
freq           1         79        70                  3  ...        1193           931             3573
mean         NaN        NaN       NaN                NaN  ...         NaN           NaN              NaN
std          NaN        NaN       NaN                NaN  ...         NaN           NaN              NaN
min          NaN        NaN       NaN                NaN  ...         NaN           NaN              NaN
25%          NaN        NaN       NaN                NaN  ...         NaN           NaN              NaN
50%          NaN        NaN       NaN                NaN  ...         NaN           NaN              NaN
75%          NaN        NaN       NaN                NaN  ...         NaN           NaN              NaN
max          NaN        NaN       NaN                NaN  ...         NaN           NaN              NaN

Checking missing values:
person_id                        0
first_name                       0
...
player_slug                      0
birthdate                        0
school                           1
country                          1
last_affiliation                 0
...
team_city                        0
playercode                       1
from_year                        0
to_year                          0
...
draft_year                       0
draft_round                    146
draft_number                   198
greatest_75_flag                 0
```

The profiling of `common_player_info` provided insights into its structure and the significance of its attributes. The dataset contains 3,632 entries and 33 columns, which encompass a wide range of player-related details. Among these, certain attributes stand out as most relevant for ETL and analysis purposes:

- `person_id`: Serves as a unique identifier for players, essential for linking with other datasets or tables;
- `first_name` and `last_name`: Provide the basic identity of players, necessary for any human-readable reporting or profiling;
- `birthdate`: Helps analyze age distributions, career timelines, and demographic trends.
- `height` and `weight`: Important physical metrics, enabling comparisons of player physiques and performance-related analysis;
- `team_id`: Associates players with their current teams, critical for team-based profiling;
- `position`: Indicates a player's role on the court, useful for positional analysis and comparisons;
- `nba_flag`: Indicates whether a player is currently active in the NBA, helping filter relevant data;
- `draft_year`: Provides historical context for career progression and draft trends.

The profiling of the table has revealed several important characteristics about the dataset, which are summarized below:

- **Completeness**: Most of the attributes have no missing values, but columns such as `draft_round` and `draft_number` exhibit some null entries, requiring imputation or filtering based on analysis needs;
- **Redundancy**: Fields like `display_first_last` and `player_slug` are useful for display purposes but redundant for analytical goals. These can be excluded in streamlined workflows;
- **Unique identifiers**: The high uniqueness of attributes like `person_id` and `player_slug` confirms their reliability as keys for joins or further data linkage.

The next step is to create visualizations that highlight key insights from the data:

- **Demographics**: Histogram of player birth years to analyze age distribution;
- **Physical attributes**: Scatterplot of height vs. weight to explore player physiques by position;
- **Career trends**: Bar chart of draft years to observe historical trends;
- **Team associations**: Pie chart or bar chart of player counts per team to visualize team distributions.

```python
import matplotlib.pyplot as plt

# Ensure the birthdate column is in datetime format
df_common_player_info['birthdate'] = pd.to_datetime(df_common_player_info['birthdate'], errors='coerce')
# Extract the year from the birthdate
df_common_player_info['birth_year'] = df_common_player_info['birthdate'].dt.year
# Drop rows with missing birth years
birth_years = df_common_player_info['birth_year'].dropna()
# Plot the histogram
plt.figure(figsize=(10, 6))
plt.hist(birth_years, bins=20, color='skyblue', edgecolor='black')
plt.xlabel('Birth Year', fontsize=14)
plt.ylabel('Number of Players', fontsize=14)
plt.grid(axis='y', linestyle='--', alpha=0.7)
# Show the plot
plt.tight_layout()
plt.show()
```

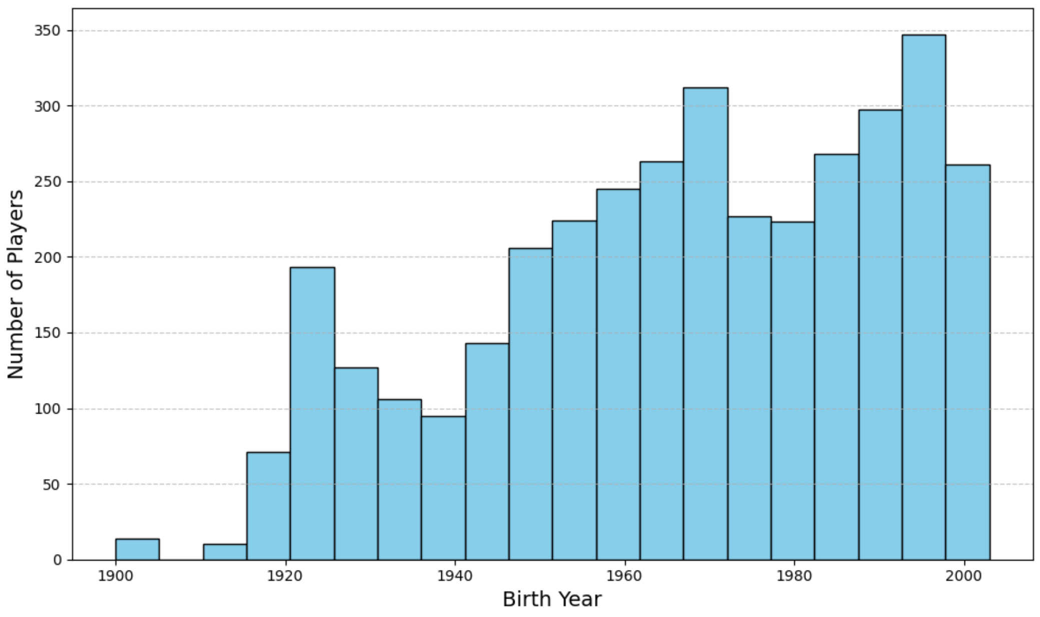


Figure 3: Distribution of players per birth years.

The dataset provides a historical overview of NBA player birth years, ranging from 1900 to 2003. It reveals trends in the number of players born each year, with significant growth over the decades. Earlier years such as 1900 and 1914 have very few players, while the 1990s and early 2000s show a notable increase in the number of players. This information can be valuable for analyzing demographic shifts in the NBA, as well as understanding how the league has expanded in terms of player participation over time (see Figure 3).

```python
import seaborn as sns
invalid_height_count = 0
def convert_height_to_inches(height):
    global invalid_height_count
    if height is None or pd.isna(height): # Check if the height is None or NaN
        return None
    try:
        # Ensure height is a string and contains a hyphen (e.g., '6-9')
        height = str(height).strip() # Remove leading/trailing spaces
        if '-' in height:
            feet, inches = height.split('-')
            return int(feet) * 12 + int(inches)
        else:
            # Count the invalid height format
            invalid_height_count += 1
            return None
    except Exception as e:
        # Count the invalid height due to an exception
        invalid_height_count += 1
        return None
# Create new variables for height and weight for the scatter plot
df_physical_stats = df_common_player_info.copy() # Copy original data to avoid changes to the original
# Clean the height and weight data for plotting
df_physical_stats['height_in_inches'] = df_physical_stats['height'].apply(convert_height_to_inches)
df_physical_stats['weight'] = pd.to_numeric(df_physical_stats['weight'], errors='coerce')
# After applying the function, print the count of invalid height formats
print(f"Number of invalid height formats: {invalid_height_count}")
# Filter data to remove rows with missing or invalid height or weight for plotting
df_physical_stats_clean = df_physical_stats[['height_in_inches', 'weight', 'position']].dropna()
# Debugging: Check the cleaned data for plotting
print("Data used for plotting:\n", df_physical_stats_clean.head())
# Create a scatterplot of height vs. weight with position as hue
plt.figure(figsize=(10, 6))
sns.scatterplot(
  data=df_physical_stats_clean,
  x='height_in_inches',
  y='weight',
  hue='position', # Color by position
  alpha=0.7
)
# Add plot title and labels
plt.title('Height vs. Weight of Players by Position', fontsize=16)
plt.xlabel('Height (inches)', fontsize=14)
plt.ylabel('Weight (lbs)', fontsize=14)
# Show the plot
plt.tight_layout()
plt.show()
```

The provided code processes NBA player height and weight data to generate a scatter plot. It first defines a function `convert_height_to_inches()` that converts height from feet-inches format (e.g., "6-9") to inches. The function also handles invalid or missing values, counting them and returning `None` for invalid entries. The data is then cleaned by applying this function to the `height` column, while the `weight` column is converted to numeric values. After cleaning, the code filters out rows with missing or invalid height and weight values and creates a scatter plot that visualizes the relationship between player height and weight, with different colors representing player positions.

The data reveals key trends in the physical attributes of basketball players. Centers are typically the tallest, ranging from 6'8" to 7'0", while Forwards are generally between 6'6" and 6'9", offering a mix of height and agility. Guards tend to be shorter, ranging from 5'9" to 6'4", focusing on speed and ball-handling. Centers also have the highest weights, while Guards and Forwards show more variation. Guard is the most common position, followed by Forward, with Center being the least frequent. This highlights the diverse body types tailored to different roles (see Figure 4).

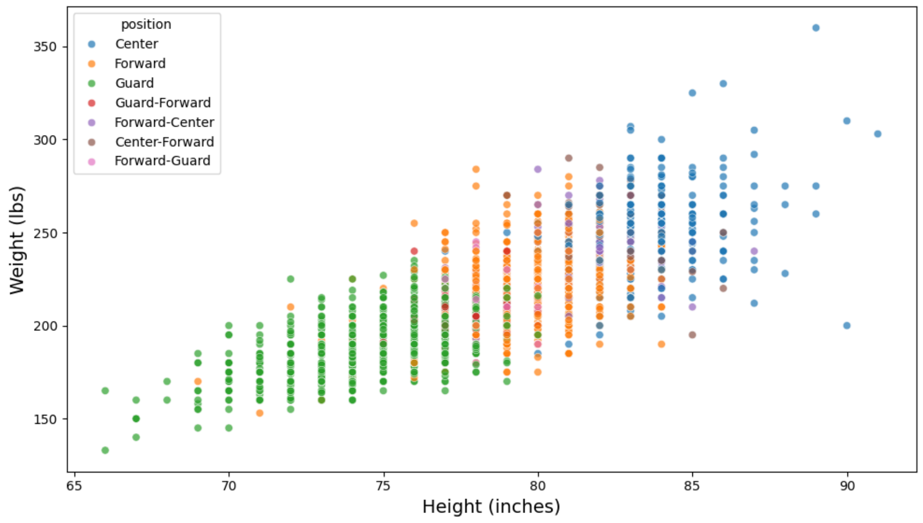


Figure 4: Correlation between height and weight of players by position.

```
# Query average height and weight by position
avg_query =
"SELECT position, AVG(height) AS avg_height, AVG(weight) AS avg_weight
 FROM common_player_info GROUP BY position;"
# Load the result of the average calculation into a pandas DataFrame
df_position_avg = pd.read_sql_query(avg_query, nba_connection)
# Display the results
print(df_position_avg)
```

```
         position  avg_height  avg_weight
0                    0.266667    9.555556
1          Center    6.299799  239.392354
2  Center-Forward    6.191176  249.955882
3         Forward    5.958652  215.786371
4  Forward-Center    6.125000  241.767857
5   Forward-Guard    6.000000  217.425926
6           Guard    5.865819  187.494350
7   Guard-Forward    6.000000  211.097015
```

This query serves as an additional example, demonstrating the expressive power of SQL in extracting and summarizing data efficiently. It calculates the average height and weight for each player position in a basketball dataset. It first constructs a SQL query that groups the data by position and computes the average height and weight for each group. The query is executed on the database using `pd.read_sql_query()`, which loads the results into a Pandas DataFrame. Finally, the resulting DataFrame, containing the average height and weight by position, is printed for display.

Table `game`

The profiling of the `game` table provides valuable insights into the structure and significance of its attributes, offering essential information for analyzing basketball games. The dataset contains 65,698 entries and 16 selected columns, capturing a wide range of game-related details. Among these, certain attributes are particularly important for ETL and analysis:

- `game_id`: Unique identifier for each game;
- `game_date`: Date when the game was played;
- `season_id`: Season in which the game occurred;
- `team_id_home` and `team_id_away`: Home and away team identifiers;

- `matchup_home` and `matchup_away`: Labels for each matchup;
- `wl_home` and `wl_away`: Win or loss outcome for home and away teams;
- `pts_home` and `pts_away`: Total points scored by home and away teams;
- `fg_pct_home` and `fg_pct_away`: Field goal percentages for each team;
- `reb_home` and `reb_away`: Total rebounds for each team.

```
# SQL query to select core attributes from the game table
query = "SELECT game_id, game_date, season_id, team_id_home,
                team_id_away, matchup_home, matchup_away, wl_home,
                wl_away, pts_home, pts_away, fg_pct_home, fg_pct_away,
                reb_home, reb_away, ast_home
         FROM game ORDER BY game_date ASC;"
# Load the data into a pandas DataFrame
df_game = pd.read_sql_query(query, nba_connection)
# ...
# Display missing values if any (other information are discarded)
print("Checking for missing values:")
print(df_game.isnull().sum())
```

```
Checking for missing values:
game_id             0
game_date           0
season_id           0
team_id_home        0
team_id_away        0
matchup_home        0
matchup_away        0
wl_home             2
wl_away             2
pts_home            0
pts_away            0
fg_pct_home     15490
fg_pct_away     15489
reb_home        15729
reb_away        15725
ast_home        15805
dtype: int64
```

The output highlights missing data in key performance metrics such as field goal percentages (`fg_pct_home` and `fg_pct_away`), rebounds (`reb_home` and `reb_away`), and assists (`ast_home`). While game-level identifiers and basic attributes such as game dates and team matchups are complete, these missing values in performance data suggest potential gaps in data collection or recording. Addressing these gaps is crucial for ensuring accurate analysis, particularly for evaluating team and player performance trends over time.

The next step of profiling would be to generate visualizations that highlight key insights from the game data:

- **Game outcomes**: Bar chart showing win/loss outcomes for home and away teams, highlighting performance trends;
- **Scoring distribution**: Histogram of points scored by home and away teams, revealing scoring patterns;
- **Shooting efficiency**: Field goal percentages scatterplot for both teams, examining offensive efficiency;
- **Rebounds**: Bar chart of total rebounds per team, assessing control of possessions.

To conserve space, only one example of the scoring distribution pattern will be displayed. The descriptive statistics for home and away team points show that home teams score an average of 104.6 points with a range from 18 to 192 points. Most home team scores fall between 95 and 114 points. Away teams, on average, score slightly fewer points (101.0), with scores ranging from 19 to 196 points. Both teams exhibit a similar scoring distribution, with most scores clustering between 90 and 115 points, reflecting consistent performance but with occasional extremes (see Figure 5).

```python
# Create histograms for the points scored by home and away teams
plt.figure(figsize=(12, 6))
# Plot for home team points
sns.histplot(df_game['pts_home'], kde=True, color='blue', label='Home Team Points', bins=30)
# Plot for away team points
sns.histplot(df_game['pts_away'], kde=True, color='red', label='Away Team Points', bins=30)
# Add labels and title
plt.xlabel('Points Scored', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
# Show legend
plt.legend(title='Team', loc='upper right')
# Display the plot
plt.tight_layout()
plt.show()
```

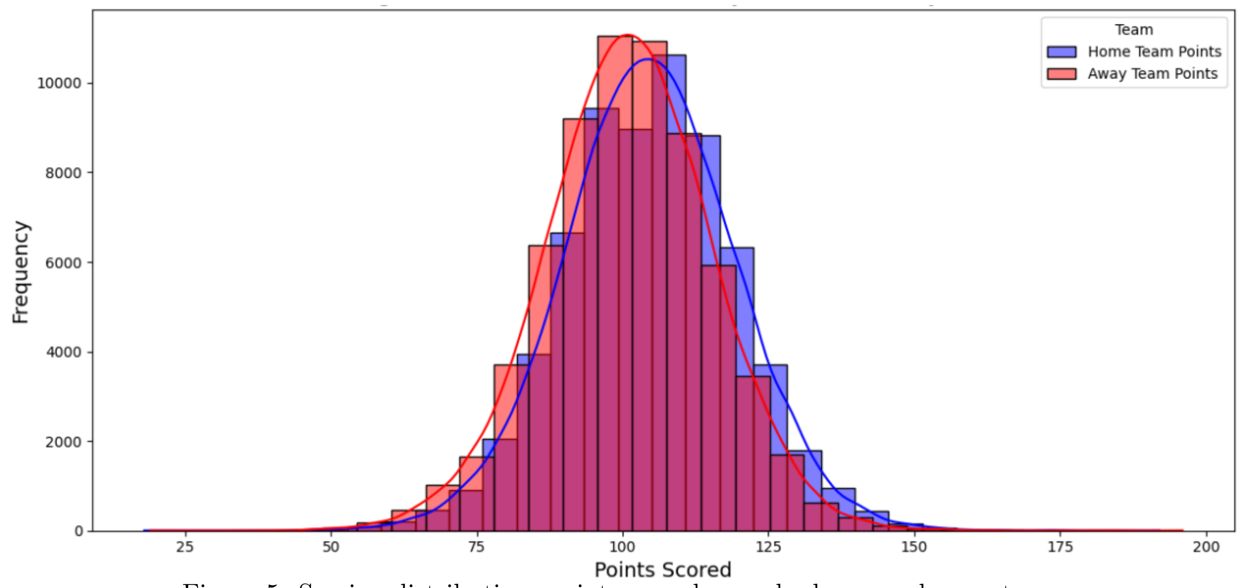


Figure 5: Scoring distribution: points scored resp. by home and away teams

Table `line_score`

The profiling of the `line_score` table provided a comprehensive overview of its structure and key attributes related to game outcomes. The dataset comprises 58,053 entries and 10 columns, capturing crucial aspects of each game, such as scoring, team identifiers, and game-specific details. Among the core attributes, the following stand out:

- `game_id`: A unique identifier for each game, essential for linking the dataset with other game-related information;
- `game_date_est`: Represents the date of the game, crucial for analyzing trends over time, such as seasonal patterns or historical performance;
- `team_id_home` and `team_id_away`: Denote the identifiers for the home and away teams, enabling team-based analyses, such as home vs. away performance comparisons;
- `pts_home` and `pts_away`: Store the total points scored by the home and away teams, respectively, serving as the primary measure of game outcomes;
- `pts_qtr1_home` and `pts_qtr1_away`: Capture the points scored by each team in the first quarter, providing insights into early game dynamics and performance;
- `team_nickname_home` and `team_nickname_away`: Specify the nicknames of the home and away teams, useful for team-specific analysis and reporting.

```
# SQL query to select core attributes from the line_score table
query = "SELECT game_id, game_date_est, team_id_home, team_id_away,
                pts_home, pts_away, pts_qtr1_home, pts_qtr1_away,
                team_nickname_home, team_nickname_away
         FROM line_score
         ORDER BY game_date_est ASC;"
# Load the data into a pandas DataFrame
df_line_score = pd.read_sql_query(query, nba_connection)
# ...
# Handle missing values if any (other information are discarded)
print("Checking for Missing Values:")
print(df_line_score.isnull().sum())
```

```
Checking for Missing Values:
game_id                  0
game_date_est            0
team_id_home             0
team_id_away             0
pts_home                 0
pts_away                 0
pts_qtr1_home         1004
pts_qtr1_away         1010
team_nickname_home       0
team_nickname_away       0
dtype: int64
```

The output of this code shows some selected core attributes from the `line_score` table. Key fields such as `game_id`, `game_date_est`, `team_id_home`, `team_id_away`, `pts_home`, `pts_away`, and team nicknames have no missing values. However, attributes related to first-quarter points (`pts_qtr1_home` and `pts_qtr1_away`) have significant missing data, with 1004 and 1010 missing values respectively. It is also important to note that this query leaves out several other columns from the table that contain additional missing data. Compared to the `game` table, the `line_score` table offers more granular information, especially with its inclusion of quarter-by-quarter scoring data, which the `game` table lacks. While both tables contain basic information like game dates and team IDs, the `line_score` table goes further by offering detailed insights into the performance of teams during specific quarters, making it possible to analyze game flow and identify early trends.

```
# Number of unique nicknames for home and away teams
unique_nicknames_home = df_line_score['team_nickname_home'].nunique()
unique_nicknames_away = df_line_score['team_nickname_away'].nunique()
# Display the results
print(f"Number of unique nicknames for home teams in table line_score: {unique_nicknames_home}")
print(f"Number of unique nicknames for away teams in table line_score: {unique_nicknames_away}")
```

```
Number of unique nicknames for home teams in table line_score: 79
Number of unique nicknames for away teams in table line_score: 85
```

```
query_team = "SELECT id, full_name, nickname FROM team;"
# Load the data into a pandas DataFrame
df_team = pd.read_sql_query(query_team, nba_connection)
# Number of unique nicknames in the team table
unique_nicknames_team = df_team['nickname'].nunique()
# Display the result
print(f"Number of unique team nicknames in table team: {unique_nicknames_team}")
```

```
Number of unique team nicknames: 30
```

The outputs reveal a discrepancy in the number of unique team nicknames across the datasets. The `line_score` table contains 79 unique nicknames for home teams and 85 for away teams, whereas the `team` table lists only 30 unique nicknames. This difference suggests that joining these tables will exclude many team nicknames that are not present in the `team` table. This limitation will become more apparent later in this section and the ETL analysis in Section 4.

The visualizations that could be derived from the `line_score` table can be used to analyze game dynamics, including:

- **Win trends**: A line chart illustrating home vs. away wins across seasons, enabling the identification of performance patterns over time;
- **Team performance**: A heatmap visualizing the frequency of home and away wins for each team, offering insights into potential home-court advantages;
- **Quarter performance**: Boxplots comparing the points scored by home and away teams in the first quarter, aiding in the identification of early-game trends;
- **Total points distribution**: A histogram comparing total points scored by home and away teams, highlighting possible scoring disparities.

For this analysis, only variants of the second visualization focusing on team performance will be presented. These visualizations differ from those based on the `game` table by emphasizing game dynamics and team-specific insights, rather than just overall outcomes, providing a more detailed understanding of team performance. The code calculates home and away wins for each team, aggregates them using `groupby`, and normalizes the results with Min-Max scaling. A heatmap is then generated to visualize win rates and highlight differences in team performance (see Figure 6).

```python
# Create a column to determine if the home team won
df_line_score['home_win'] = df_line_score['pts_home'] > df_line_score['pts_away']
# Create a column to determine if the away team won
df_line_score['away_win'] = df_line_score['pts_away'] > df_line_score['pts_home']
# Count home and away wins per team using team nicknames
home_wins = df_line_score.groupby('team_nickname_home')['home_win'].sum()
away_wins = df_line_score.groupby('team_nickname_away')['away_win'].sum()
# Align the indices of home_wins and away_wins using an outer join
all_teams = home_wins.index.union(away_wins.index)
home_wins = home_wins.reindex(all_teams, fill_value=0)
away_wins = away_wins.reindex(all_teams, fill_value=0)
# Combine into a single DataFrame
team_performance = pd.DataFrame({
  'Team Nickname': all_teams,
  'Home Wins': home_wins.values,
  'Away Wins': away_wins.values})
# Sort the teams for better visualization
team_performance = team_performance.sort_values(by='Team Nickname')
# Normalize the values according a MinMax scaler
team_performance_normalized = team_performance.copy()
team_performance_normalized['Home Wins'] =
  (team_performance_normalized['Home Wins'] - team_performance_normalized['Home Wins'].min()) /
  (team_performance_normalized['Home Wins'].max() - team_performance_normalized['Home Wins'].min())
team_performance_normalized['Away Wins'] =
  (team_performance_normalized['Away Wins'] - team_performance_normalized['Away Wins'].min()) /
  (team_performance_normalized['Away Wins'].max() -team_performance_normalized['Away Wins'].min())
# Reshape for heatmap (pivot format)
heatmap_data =
  team_performance_normalized.melt(id_vars='Team Nickname', var_name='Location', value_name='Win Rate')
# Create the heatmap using seaborn
plt.figure(figsize=(12, 6))
heatmap = sns.heatmap(heatmap_data.pivot(index='Team Nickname', columns='Location', values='Win Rate'),
                        cmap='coolwarm', annot=True, fmt='.2f', linewidths=0.5, cbar_kws={'label': 'Win Rate'})
# Plotting code
plt.xlabel('Location', fontsize=12)
plt.ylabel('Team Nickname', fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
# Display the heatmap
plt.tight_layout()
plt.show()
```

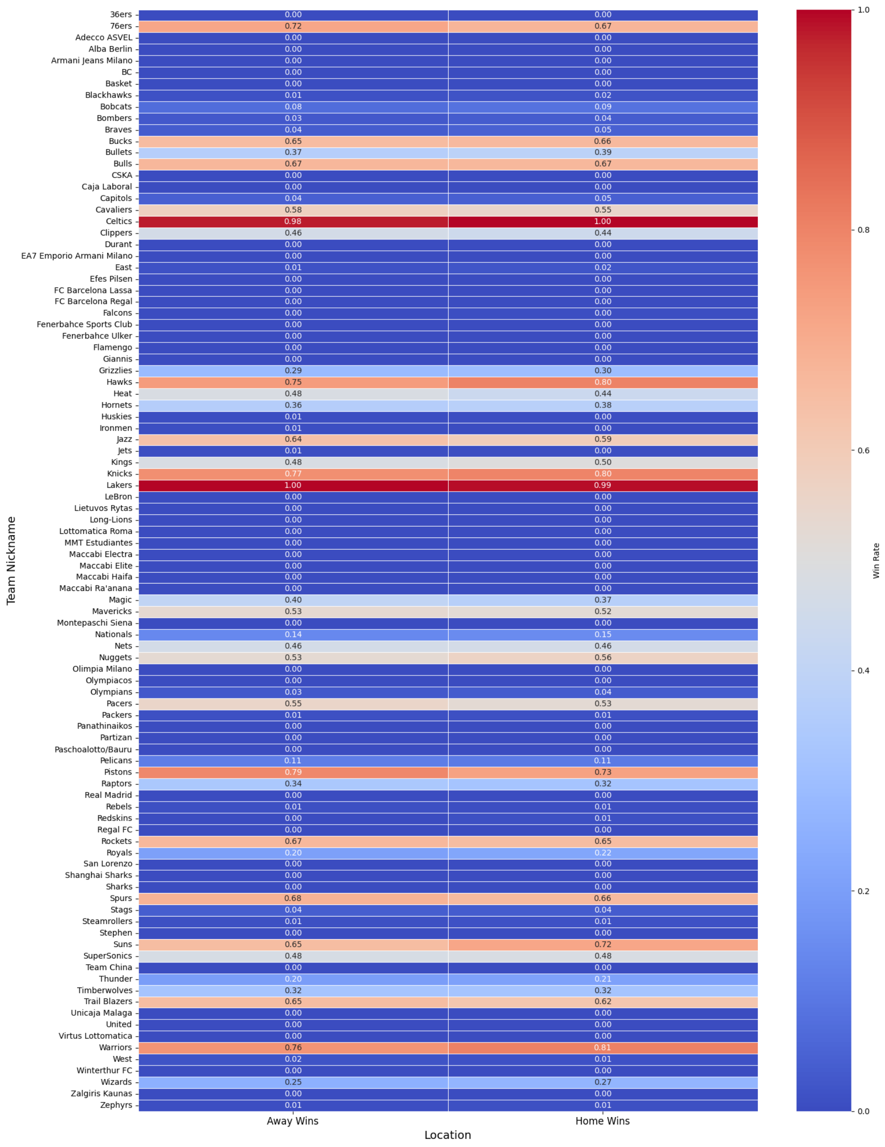


Figure 6: Team performance: home vs. away wins.

```
DROP VIEW IF EXISTS line_score_team_view;
CREATE VIEW line_score_team_view AS
SELECT ls.game_id, ls.game_date_est,
      ls.team_id_home, th.nickname AS team_nickname_home,
      ls.team_id_away, ta.nickname AS team_nickname_away,
      ls.pts_home, ls.pts_away, ls.pts_qtr1_home, ls.pts_qtr1_away
FROM line_score AS ls
LEFT JOIN team AS th ON ls.team_id_home = th.id
LEFT JOIN team AS ta ON ls.team_id_away = ta.id;
```

To address the mismatch in team nicknames, an SQL query was executed to create a view in the SQLite database using DBeaver. This view, `line_score_team_view`, was designed to combine data from the `line_score` table with the `team` table. The query joins the `line_score` table with the `team` table twice: once for the home team (`team_nickname_home`) and once for the away team (`team_nickname_away`), linking the teams via their IDs. The view includes relevant data such as game IDs, game dates, and scores, along with the corresponding team nicknames. This view provides a consolidated representation of game results, which can be used in subsequent analyses. However, the `team` table contains only the most important teams, meaning that many from the `line_score` table will be excluded due to their absence from the `team` table.

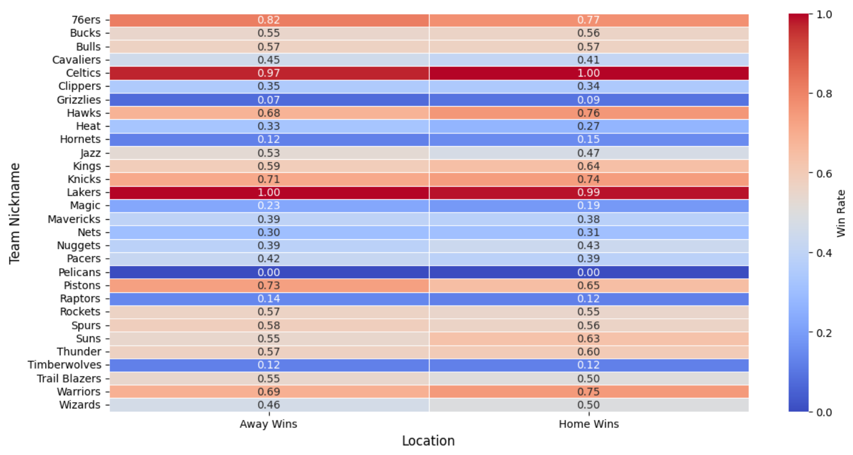


Figure 7: Team performance: home vs. away wins (applied to the view `line_score_team_view`).

```python
df_line_score_team_view['home_win'] = df_line_score_team_view['pts_home'] > df_line_score_team_view['pts_away']
df_line_score_team_view['away_win'] = df_line_score_team_view['pts_away'] > df_line_score_team_view['pts_home']
home_wins = df_line_score_team_view.groupby('team_nickname_home')['home_win'].sum()
away_wins = df_line_score_team_view.groupby('team_nickname_away')['away_win'].sum()
all_teams = home_wins.index.union(away_wins.index)
home_wins = home_wins.reindex(all_teams, fill_value=0)
away_wins = away_wins.reindex(all_teams, fill_value=0)
team_performance =
  pd.DataFrame({'Team Nickname': all_teams,'Home Wins': home_wins.values,'Away Wins': away_wins.values})
team_performance = team_performance.sort_values(by='Team Nickname')
# Normalize the values according a MinMax scaler
team_performance_normalized = team_performance.copy()
team_performance_normalized['Home Wins'] =
    (team_performance_normalized['Home Wins'] - team_performance_normalized['Home Wins'].min()) /
    (team_performance_normalized['Home Wins'].max() - team_performance_normalized['Home Wins'].min())
team_performance_normalized['Away Wins'] =
    (team_performance_normalized['Away Wins'] - team_performance_normalized['Away Wins'].min()) /
    (team_performance_normalized['Away Wins'].max() - team_performance_normalized['Away Wins'].min())
heatmap_data =
  team_performance_normalized.melt(id_vars='Team Nickname',var_name='Location', value_name='Win Rate')
# Plotting code ...
```

This code extends the previous analysis on the `line_score` table by using the `line_score_team_view`. Figure 7 shows the output of the code reducing the number of teams to the 30 most important and well known teams in NBA.

In Section 3, the importance of data integrity, consistency, and completeness was explored. The profiling process provided insight into the data's structure, revealing irregularities that could affect later processes. The next phase focuses on integrating, transforming, and preparing datasets for analysis.

4 ETL and dataset integration with Tableau Data Prep

The ETL (Extraction, Transformation, and Loading) process is the backbone of modern data integration. It enables the efficient transfer of data from diverse source systems to centralized data warehouses. This workflow involves three primary steps:

- **Extraction:** Retrieving raw data from various sources, ensuring completeness and minimal disruption to source systems;
- **Transformation:** Refining and aligning the extracted data to match the target system's business rules, ensuring consistency and usability;
- **Loading:** Delivering the transformed data into a designated storage environment, ready for analysis.

Although indispensable, ETL can be resource-intensive, requiring careful planning and tool selection to balance cost, complexity, and functionality. For this project, Tableau Data Prep is used for ETL. It is a user-friendly and powerful tool designed to streamline ETL processes by offering intuitive interfaces and robust functionality, making it well-suited for managing data transformations and integrations effectively.

Metadata plays a crucial role in the ETL process by providing descriptive information about data that supports both technical teams and end-users. For IT personnel, it ensures streamlined data governance through details on structure, sources, and refresh schedules, while for users, it enhances accessibility with clarity on content and reporting availability. Similarly, effective ETL tools must align with project needs, offering compatibility with diverse systems, automated metadata capture, and user-friendly interfaces to balance complexity and functionality. With these principles, we proceed to implement ETL processes using Tableau Data Prep, focusing on transforming and integrating data for analysis.

4.1 Initial critics

Before diving into the detailed ETL process, it is crucial to identify potential data quality and structural issues in each table. This step ensures that the transformation and integration efforts address any inconsistencies from the outset. The absence of adequate metadata further limits the usability of the data, as it is unclear which measurement standards are being used.

The `height` and `weight` attributes in `common_player_info` present significant issues related to standardization and usability. The height data is stored in a non-standard `feet-inches` format (`6-10`, for example), complicating integration and requiring conversion to a consistent unit, such as inches or centimeters. The conversion formula for height is: `height_in_inches` = (`feet` × `12`) + `inches`. To address these issues, it is recommended to standardize the height by converting it to inches or centimeters.

Regarding the date attributes across the tables, the format is consistent (`YYYY-MM-DD HH:MM:SS`), but the inclusion of a default time (`00:00:00`) is unnecessary for date analysis, as it adds no relevant information. Storing the birthdate in the `YYYY-MM-DD` format would enhance clarity and usability.

The presence of `null` columns and non-numeric entries, such as "Undrafted," in fields like `draft_year`, `draft_round`, and `draft_number` in `common_player_info` introduces inconsistencies that complicate analysis. While "Undrafted" is a valid entry, it should be standardized to `NaN` or `null` rather than a string, to align with numeric data. This standardization would facilitate better handling of missing data during ETL processes and subsequent analysis.

While the following has not been addressed for several attributes in the different tables (reasons will be explained later), handling missing or `null` values in numeric fields is crucial to prevent interference with aggregations and calculations. A consistent approach to managing such data—whether through imputation, exclusion, or clear labeling—will significantly improve data quality and lead to more reliable analysis. Additionally, it will ensure that downstream processes are not impacted by missing or inconsistent data. Furthermore, while not directly addressed in this report, providing comprehensive metadata documentation that clearly defines units, formats, and data sources will enhance data clarity, improve traceability, and facilitate smoother ETL processes, thereby increasing overall trust in the dataset.

4.2 Cleansing step

Before integrating and transforming the data for analysis, it is essential to cleanse and standardize the datasets to address inconsistencies, null values, and non-standard formats. The following steps outline the specific cleansing processes applied to each of the four tables: `common_player_info`, `team`, `game`, and `line_score`. These steps ensure the data is prepared for efficient ETL and analysis workflows.

The following steps outline the data cleansing and transformation process applied to the `common_player_info` table. These steps address issues related to missing values, standardization, and unnecessary attributes to ensure the data is consistent, reliable, and ready for analysis:

- Remove unnecessary attributes:
  - `games_played_current_season_flag`, `team_code`,
  - `display_first_last`, `display_fi_last`,
  - `player_slug`, `playercode`, and `team_abbreviation`.
- Standardize `birthdate` by converting it from `YYYY-MM-DD HH:MM:SS` to `YYYY-MM-DD`;
- Replace `null` values:
  - `school` and `country`: replace `null` value swith `Unknown`;
  - `height`: replace `null` values with `0-0`;
  - `position`, `team_name`, and `team_city`: Replace `null` values with `Unknown`;
- Process the `height` attribute:
  - Split `height` into `height_1_feet` and `height_2_inches`;
  - Create `height_inches` using the formula: `height_inches` = `height_1_feet` × 12 + `height_2_inches`.
  - Remove `height_1_feet` and `height_2_inches`;
  - Create `height_inches_imputed` to replace `0` values in `height_inches` with the median (78 inches);
- Process the `weight` attribute:
  - Create a fixed level of detail (LOD) for `weight_median`;
  - Impute missing `weight` values using this code:
    `IF ISNULL(Weight) THEN weight_median ELSE Weight END`;
  - Remove `weight_median`;
- Retain `null` values for `jersey` (player) numbers, as this attribute is not relevant to the analysis and the missing values do not impact the process;
- Create `active_nb_years` using the formula: `IF ISNULL(from_year) OR ISNULL(to_year) THEN NULL ELSE to_year - from_year END`;
- Standardize `draft_year` by converting `Undrafted` to `NULL`;
- Rename `team_name` to `team_nickname` for consistency with the `team` table.

Before proceeding with further transformations, the following steps are taken to clean the `game` table:

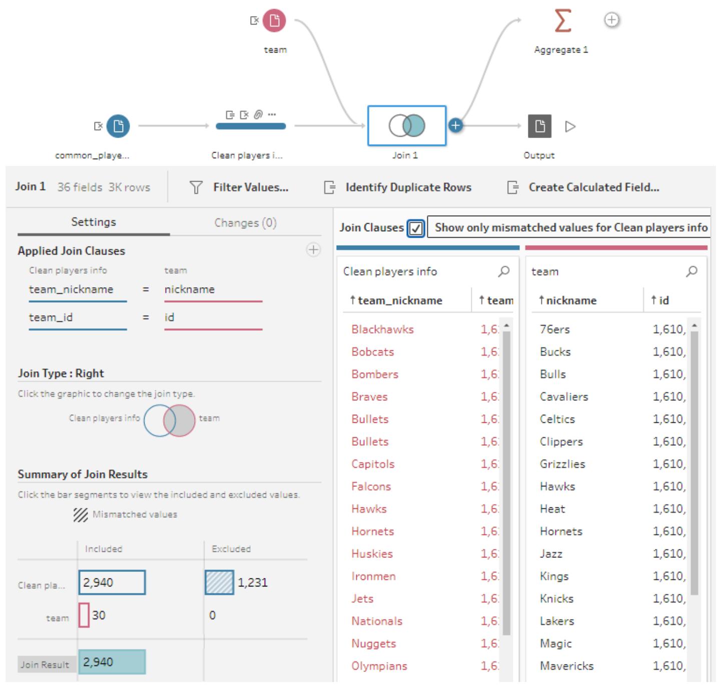
- Change the `date` format from `YYYY-MM-DD HH:MM:SS` to `YYYY-MM-DD`.
- Retain `null` values for numeric attributes to maintain data integrity, as missing values should remain unchanged, especially since 0 values are already present.
- Attributes flagged for removal in Section 3 will be retained for now in the ETL process to ensure data completeness and avoid potential disruptions in analysis.

For Table `line_score`, the following cleaning steps are applied to enhance data consistency and optimize the dataset for analysis:

- Convert `game_date_est` from `YYYY-MM-DD HH:MM:SS` to a more streamlined `YYYY-MM-DD` format;
- Eliminate the columns `team_city_name_home` and `team_city_name_away`, as they are irrelevant for the analysis and are already included in the `team` table;
- Split `team_wins_losses_home` and `team_wins_losses_away` into individual attributes for better clarity, then remove the original columns;

- Remove the `pts_ot6_home`, ..., `pts_ot10_home`, and `pts_ot6_away`, ..., `pts_ot10_away` attributes, as they contain null or zero values that do not provide useful information. These fields track the points scored by the home and away teams during overtime periods 6 to 10. The data only includes up to the fifth overtime period;
- Remove `pts_ot5_home` and `pts_ot5_away`, as these attributes predominantly contain null, zero, or a single non-unique value, which limits their usefulness for analysis and does not contribute meaningful insights.

The `team` table does not require any cleansing, as it contains clean and well-structured data that is already in a suitable format for analysis. No further modifications or transformations are necessary for this table.



Join 1 36 fields 3K rows

Applied Join Clauses

Clean players info		team
team_nickname	=	nickname
team_id	=	id

Join Type : Right

Click the graphic to change the join type.

Clean players info team

Summary of Join Results

Click the bar segments to view the included and excluded values.

▨ Mismatched values

	Included	Excluded
Clean pla...	2,940	▨ 1,231
team	30	0
Join Result	2,940	

Join Clauses [✓] Show only mismatched values for Clean players info

Clean players info

↑ team_nickname	↑ team
Blackhawks	1,6
Bobcats	1,6
Bombers	1,6
Braves	1,6
Bullets	1,6
Bullets	1,6
Capitols	1,6
Falcons	1,6
Hawks	1,6
Hornets	1,6
Huskies	1,6
Ironmen	1,6
Jets	1,6
Nationals	1,6
Nuggets	1,6
Olympians	1,6

team

↑ nickname	↑ id
76ers	1,610,
Bucks	1,610,
Bulls	1,610,
Cavaliers	1,610,
Celtics	1,610,
Clippers	1,610,
Grizzlies	1,610,
Hawks	1,610,
Heat	1,610,
Hornets	1,610,
Jazz	1,610,
Kings	1,610,
Knicks	1,610,
Lakers	1,610,
Magic	1,610,
Mavericks	1,610,

Figure 8: Join step between cleansed `common_player_info` and `team` tables.

4.3 Join steps

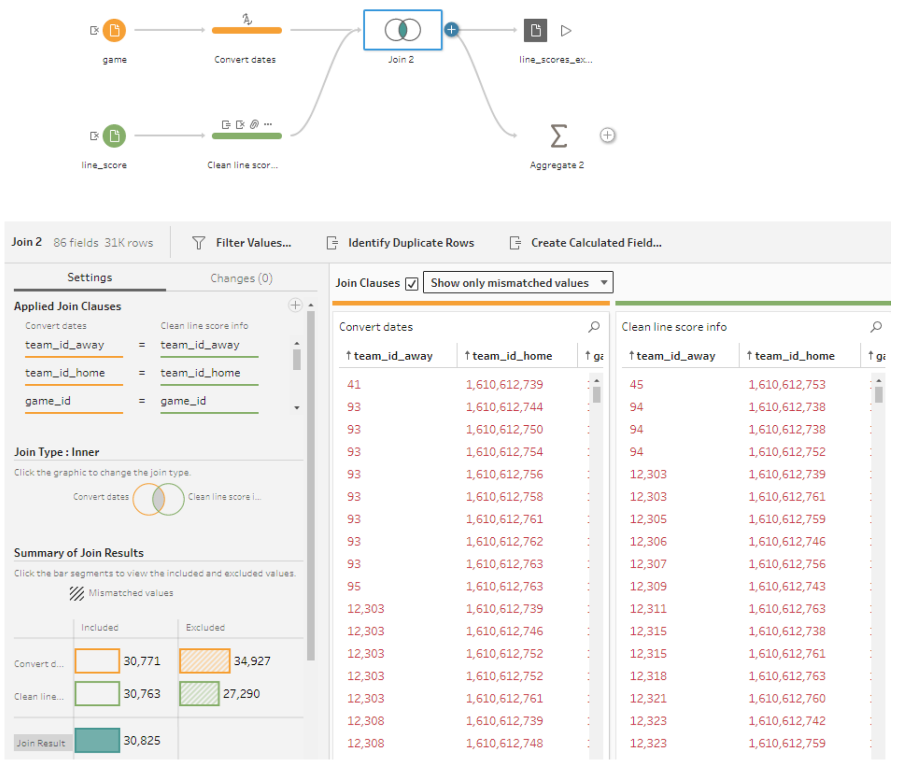
The first join step (see Figure 8) is between the cleansed table `common_player_info` and `team` table, based on the join clause $\overline{\texttt{jc1}}$ and the following equalities:

- $\texttt{team\_nickname} = \texttt{nickname}$;
- $\texttt{team\_id} = \texttt{id}$.

As indicated by the information provided, this join step produces mismatches. These mismatches primarily arise from missing `team_nickname` and `team_id` values in the `team` table, which only contains data for the most well-known teams, as mentioned in Section 3.3. This is why the join step results in the values specified by the following formula:

$$(\texttt{common\_player\_info} \cap_{\overline{\texttt{jc1}}} \texttt{team}) \cup \texttt{team} \setminus_{\overline{\texttt{jc1}}} \texttt{common\_player\_info} = \texttt{common\_player\_info} \cap_{\overline{\texttt{jc1}}} \texttt{team}$$

since $\texttt{team} \setminus_{\overline{\texttt{jc1}}} \texttt{common\_player\_info} = \emptyset$. The notation $\cap_{\overline{\texttt{jc1}}}$ represents the $\overline{\texttt{jc1}}$-intersection and $\setminus_{\overline{\texttt{jc1}}}$ represents the $\overline{\texttt{jc1}}$-set-minus.



Join 2 86 fields 31K rows

Applied Join Clauses

Convert dates		Clean line score info
team_id_away	=	team_id_away
team_id_home	=	team_id_home
game_id	=	game_id

Join Type : Inner

Click the graphic to change the join type.

Summary of Join Results

Click the bar segments to view the included and excluded values.

▨ Mismatched values

	Included	Excluded
Convert d...	30,771	34,927
Clean line...	30,763	27,290
Join Result	30,825	

Join Clauses ✓ Show only mismatched values

Convert dates

team_id_away	team_id_home
41	1,610,612,739
93	1,610,612,744
93	1,610,612,750
93	1,610,612,754
93	1,610,612,756
93	1,610,612,758
93	1,610,612,761
93	1,610,612,762
93	1,610,612,763
95	1,610,612,763
12,303	1,610,612,739
12,303	1,610,612,746
12,303	1,610,612,752
12,303	1,610,612,752
12,303	1,610,612,761
12,308	1,610,612,739
12,308	1,610,612,748

Clean line score info

team_id_away	team_id_home
45	1,610,612,753
94	1,610,612,738
94	1,610,612,738
94	1,610,612,752
12,303	1,610,612,739
12,303	1,610,612,761
12,305	1,610,612,759
12,306	1,610,612,746
12,307	1,610,612,756
12,309	1,610,612,743
12,311	1,610,612,763
12,315	1,610,612,738
12,315	1,610,612,761
12,318	1,610,612,763
12,321	1,610,612,760
12,323	1,610,612,742
12,323	1,610,612,759

Figure 9: Join step between cleansed `common_player_info` and `team` tables.

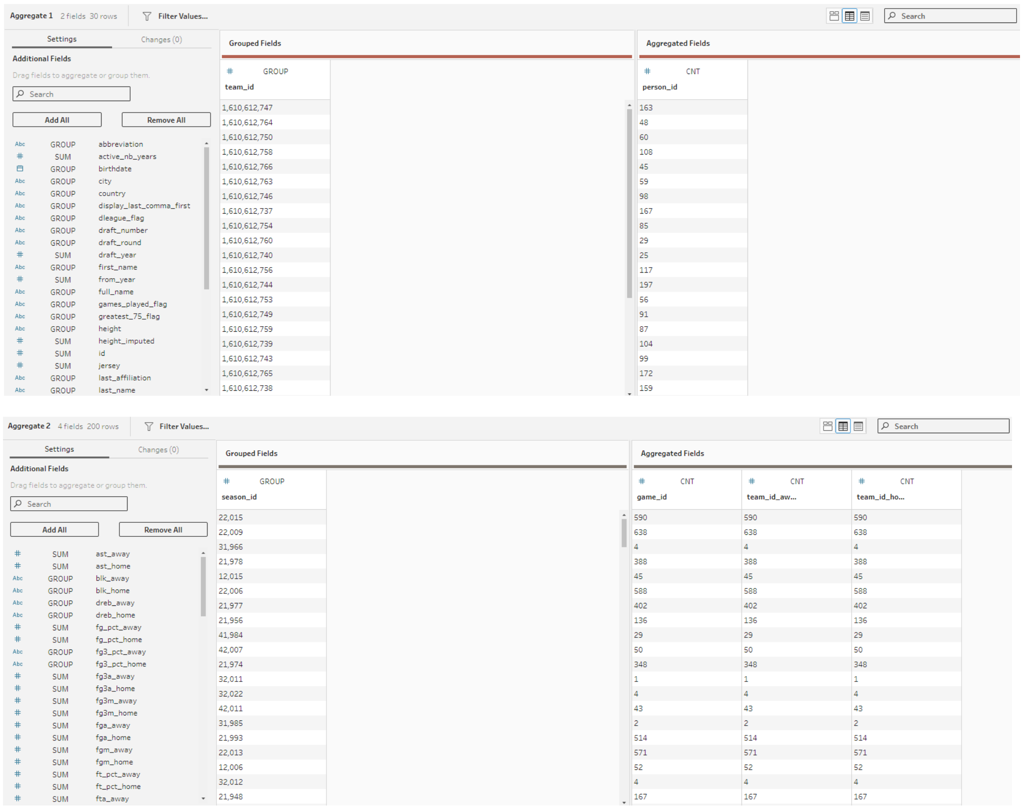
The second join step (see Figure 9) is between the cleansed tables `game` and `line_score`, based on the join clause $\overline{\texttt{jc2}}$ and the following equalities:

- $\texttt{team\_id\_home}_{\texttt{game}} = \texttt{team\_id\_home}_{\texttt{line\_score}}$;
- $\texttt{team\_id\_away}_{\texttt{game}} = \texttt{team\_id\_away}_{\texttt{line\_score}}$;
- $\texttt{game\_id}_{\texttt{game}} = \texttt{game\_id}_{\texttt{line\_score}}$;
- $\texttt{team\_abbreviation\_home}_{\texttt{game}} = \texttt{team\_abbreviation\_home}_{\texttt{line\_score}}$;
- $\texttt{team\_abbreviation\_away}_{\texttt{game}} = \texttt{team\_abbreviation\_away}_{\texttt{line\_score}}$;
- $\texttt{pts\_home}_{\texttt{game}} = \texttt{pts\_home}_{\texttt{line\_score}}$;
- $\texttt{pts\_away}_{\texttt{game}} = \texttt{pts\_away}_{\texttt{line\_score}}$.

As indicated by the provided information, this join step produces mismatches in both tables. These mismatches primarily arise from missing `game_id` and `team_id` values in both tables. The join results can be formalized as $\texttt{game} \cap_{\overline{\texttt{jc2}}} \texttt{line\_score}$ where $\cap_{\overline{\texttt{jc2}}}$ represents the $\overline{\texttt{jc2}}$-intersection. While the `team` table could have been included in this join to reduce the number of rows, it was omitted for simplicity, as the necessary data is already represented within the `line_score` and `game` tables.

4.4 Aggregations

The first aggregation was applied to the first join $\overline{\texttt{jc1}}$ between the cleansed `common_player_info` table and the `team` table. This aggregation counts the number of players per team (see Figure 10, top). The second aggregation was applied to the result of the second join $\overline{\texttt{jc2}}$ between the cleansed `game` and `line_score` tables. It counts the number of games, home teams, and away teams, which are logically equal as expected (see Figure 10, bottom).



Aggregate 1 2 fields 30 rows

Settings | Changes (0)

Additional Fields

Drag fields to aggregate or group them.

Abc	GROUP	abbreviation
#	SUM	active_nb_years
	GROUP	birthdate
Abc	GROUP	city
Abc	GROUP	country
Abc	GROUP	display_last_comma_first
Abc	GROUP	dleague_flag
Abc	GROUP	draft_number
Abc	GROUP	draft_round
#	SUM	draft_year
Abc	GROUP	first_name
#	SUM	from_year
Abc	GROUP	full_name
Abc	GROUP	games_played_flag
Abc	GROUP	greatest_75_flag
Abc	GROUP	height
#	SUM	height_imputed
#	SUM	id
#	SUM	jersey
Abc	GROUP	last_affiliation
Abc	GROUP	last_name

Grouped Fields	Aggregated Fields
GROUP	CNT
team_id	person_id
1,610,612,747	163
1,610,612,764	48
1,610,612,750	60
1,610,612,758	108
1,610,612,766	45
1,610,612,763	59
1,610,612,746	98
1,610,612,737	167
1,610,612,754	85
1,610,612,760	29
1,610,612,740	25
1,610,612,756	117
1,610,612,744	197
1,610,612,753	56
1,610,612,749	91
1,610,612,759	87
1,610,612,739	104
1,610,612,743	99
1,610,612,765	172
1,610,612,738	159

Aggregate 2 4 fields 200 rows

Settings | Changes (0)

Additional Fields

Drag fields to aggregate or group them.

#	SUM	ast_away
#	SUM	ast_home
Abc	GROUP	blk_away
Abc	GROUP	blk_home
Abc	GROUP	dreb_away
Abc	GROUP	dreb_home
#	SUM	fg_pct_away
#	SUM	fg_pct_home
Abc	GROUP	fg3_pct_away
Abc	GROUP	fg3_pct_home
#	SUM	fg3a_away
#	SUM	fg3a_home
#	SUM	fg3m_away
#	SUM	fg3m_home
#	SUM	fga_away
#	SUM	fga_home
#	SUM	fgm_away
#	SUM	fgm_home
#	SUM	ft_pct_away
#	SUM	ft_pct_home
#	SUM	fta_away

Grouped Fields	Aggregated Fields		
GROUP	CNT	CNT	CNT
season_id	game_id	team_id_aw...	team_id_ho...
22,015	590	590	590
22,009	638	638	638
31,966	4	4	4
21,978	388	388	388
12,015	45	45	45
22,006	588	588	588
21,977	402	402	402
21,956	136	136	136
41,984	29	29	29
42,007	50	50	50
21,974	348	348	348
32,011	1	1	1
32,022	4	4	4
42,011	43	43	43
31,985	2	2	2
21,993	514	514	514
22,013	571	571	571
12,006	52	52	52
32,012	4	4	4
21,948	167	167	167

Figure 10: Aggregations performed on the join results of the cleansed `common_player_info` and `team` tables (top), and the cleansed `line_score` and `game` tables (bottom), respectively.

With the data cleansed and aggregated, the next step involves using Tableau Desktop to design dashboards that highlight key metrics and trends. This section explores how Tableau's visualization tools transform prepared data into actionable insights.

5 Dashboard visualizations using Tableau Desktop

In this section, dashboard creation was explored using Tableau Desktop, with a focus on visualizing key outputs from the data profiling process described earlier. Specifically, the diagrams introduced in Section 3 were revisited and applied to the output data generated by the first flow, which involved joining the cleansed `common_player_info` and `team` tables, and the second flow, which combined the cleansed `game` and `line_score` tables. Although advanced features of Tableau Desktop were not extensively explored, its core functionalities were utilized to produce the following visualizations.

The first visualization (see Figure 11, left) presents the distribution of players by their birth years, derived from the data produced in the first flow, which joined the cleansed `common_player_info` and `team` tables. Unlike the histogram shown in Figure 3, this visualization employs an area diagram to depict the distribution, offering a smoother and more continuous representation of the data. The area diagram highlights the concentration of players across different birth years, emphasizing trends and patterns over time.

Figure 11 (right) shows a scatter plot that is nearly identical to the one presented in Figure 4, with the key difference being the use of Tableau for its creation. As mentioned earlier, the plot visualizes the relationship between player height and weight, categorized by position.

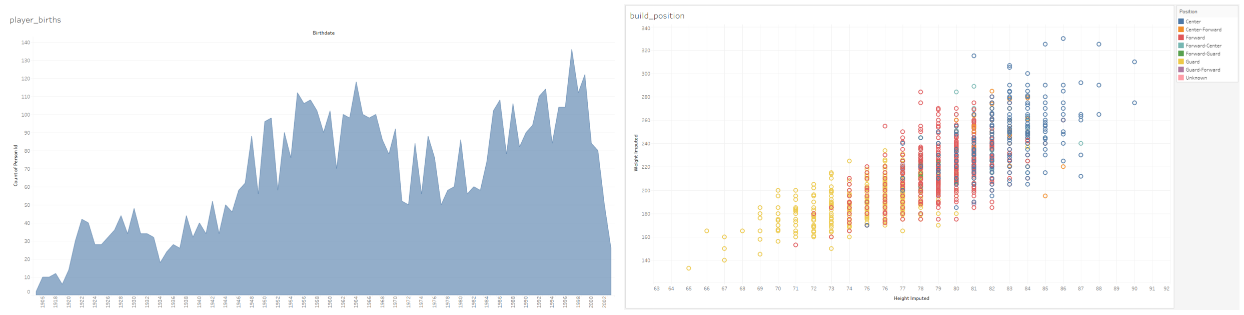


Figure 11: Area diagram showing the distribution of players by birth year (left) and height vs. weight of players by position (right), derived from the first flow derived from the first flow joining the cleansed `common_player_info` and `team` tables.

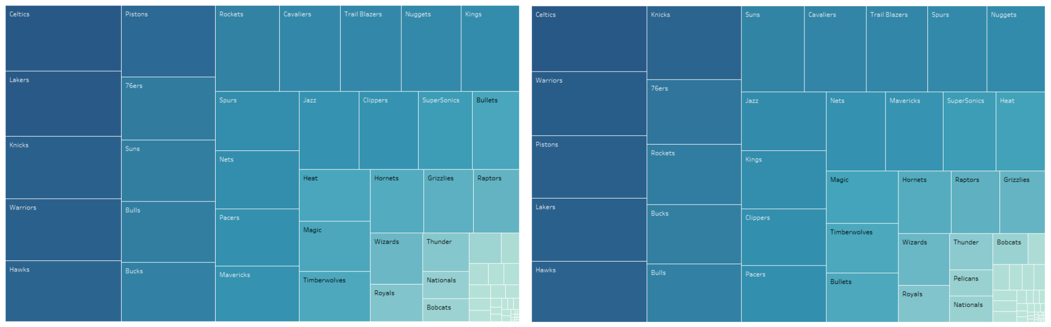


Figure 12: Home wins (left) and away wins (right) tree maps derived from the second flow joining the cleansed `team` and `line_score` tables.

Figure 12 displays two treemaps representing home wins (left) and away wins (right) for each team, derived from the second flow that joins the cleansed `team` and `line_score` tables. The treemaps allow a quick visual comparison of the number of wins for each team, with the size of each box indicating the volume of wins. This visualization provides an intuitive understanding of the performance distribution between home and away games.

A key difference with Figure 6 is that the Celtics are at the top of both home and away wins in both treemaps. In the heatmap shown in Figure 6, the Celtics are at the top for home wins, while the Lakers are at the top for away wins. This discrepancy is likely due to the join action in Tableau Data Prep, which excluded over 27,290 rows from the `line_score` table, as shown in Figure 9.

All inputs and outputs related to this work are available for download and review at the following link.