National Football League Data Analysis Guidebook

Introduction

What is the NFL?

The **National Football League (NFL)** is the premier professional football league in the United States, featuring the best teams and players in the sport. Established in 1920, the NFL has grown into one of the most-watched sports leagues globally, captivating millions of fans every season.

A Brief History of the NFL

- Founded in 1920 as the American Professional Football Association (APFA) and renamed the NFL in 1922.
- Originally composed of 10 teams, the league expanded and merged with the American Football League (AFL) in 1970 to form the modern NFL.
- Today, the NFL comprises 32 teams, divided into the AFC and NFC conferences.

Key historical moments:

- 1967 – The first Super Bowl took place.
- 1982 – The NFL introduced a 16-game schedule, bringing more action-packed seasons.
- 2010s – The NFL embraced advanced technology and analytics to improve player safety and enhance fan engagement.

Rules of the NFL

American Football, also known as Rugby Football or Gridiron, originated in the United States.

- Each team has **11 players** on the field at a time, divided into three main units:
    - **Offense**: Focuses on moving the ball toward the opponent's end zone to score points.
    - **Defense**: Works to stop the offensive team and regain possession of the ball.
    - **Special Teams**: Handles kickoffs, field goals, and punts.

Player Roles

- **Quarterback**: Leads the offense, calls plays, and passes or hands off the ball.
- **Running Backs** and **Wide Receivers**: Advance the ball by running or catching passes.
- **Offensive Linemen**: Protect the quarterback and create openings for runners.
- **Linebackers** and **Cornerbacks** (Defense): Cover the field to stop offensive players from advancing.

Game Structure

- The game starts with a coin toss to decide whether to kick or receive the ball.
- The offense has four downs (attempts) to gain at least 10 yards.
    - If they succeed, they get a new set of downs.
    - If they fail, the ball goes to the other team.
- Teams alternate between offense and defense over four 15-minute quarters.
- When the clock runs out, the team with the most points wins the game.

How to Score Points in American Football

In American football, teams can score points in several ways. The goal is to accumulate the most points by the end of the game.

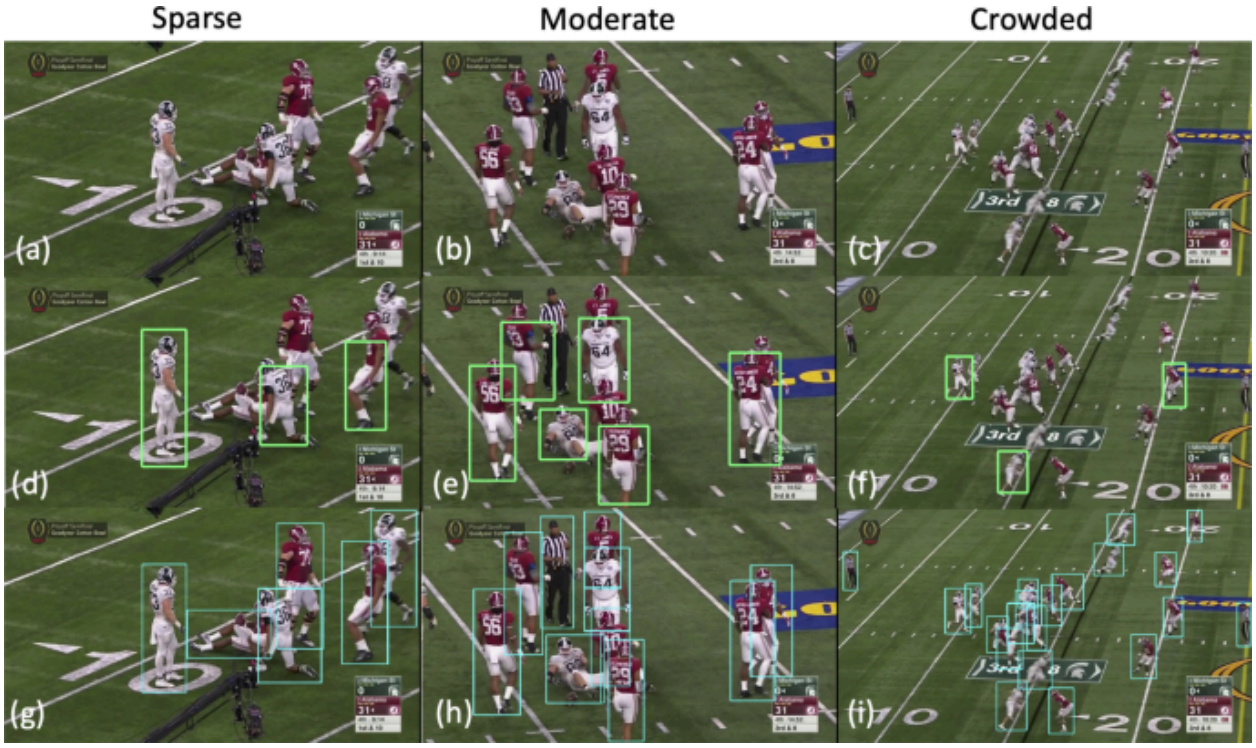
- **Touchdown (6 points)**
    - Scored when the offense crosses the opponent's goal line by running or catching the ball in the end zone.
- **Field Goal (3 points)**
    - The kicker attempts to kick the ball through the uprights and over the crossbar when the offense is close to the opponent's goal.
- **Extra Point (1 or 2 points)**
    - **1 Point:** After a touchdown, the team kicks the ball through the uprights.
    - **2 Points:** The offense runs or passes the ball into the end zone again after the touchdown. This is called a **two-point conversion** and is harder to achieve.
- **Safety (2 points)**
    - The defense scores two points by tackling an offensive player with the ball in their own end zone.

Geospatial Data in the NFL

What is Geospatial Data?

Geospatial data is information that represents objects or events with a specific geographic location. In the context of the NFL, this data can include:

- Player movement tracking during games
- Stadium coordinates and weather conditions
- Fan engagement and regional interest trends



How it's collected

Figure 1 - NGS system Housing

Geospatial data in the NFL is primarily collected through the **Next Gen Stats (NGS)** system. This is a system that's developed with a partnership between the NFL and Amazon Web Services (AWS).

The system uses **RFID (Radio Frequency Identification) chips** embedded in players' shoulder pads, footballs, and sometimes referees' gear (fig.1). These chips track movement at high precision. The tracking system captures the location of each player and the ball **every tenth of a second (decisecond resolution)** during plays. What gets collected?

- **X and Y Coordinates**: The position of players and the football on the field.
- **Speed and Acceleration**: Calculated based on movement data (yards per second).
- **Distance Metrics**: Tracks the total and relative distances between players, including defenders and special teams' positioning.

The collected data is merged with traditional game data (e.g., play type, time on the clock, and scoring events) to perform advanced analytics. Spatial analysis techniques like **hotspot mapping** identify areas on the field where successful plays occur more frequently, helping teams strategize.

And finally, the bit that we are interested in, data processing.The dataset is massive, often containing **tens of millions of rows**, requiring machine learning models such as **decision trees, regression models, and clustering algorithms** to extract insights.

How it can be useful

Geospatial data collected is highly valuable for NFL teams, analysts, and strategists. **Performance Analysis is an integral part of the team.** Coaches can identify fast players and **optimize their roles in special teams,** such as kick returners or punt coverage units. The Tracking data also helps analyze how players move in real-time, allowing teams to optimize formations and player placements.

Teams can use **hotspot analysis** to determine the best areas to target punts and kickoffs to minimize return yardage. The data helps identify ideal kicking positions and how environmental factors (wind, field conditions) affect accuracy. Coaches can **analyze where most successful returns happen** and instruct returners to take advantage of those lanes.

Data on player speed, acceleration, and total distance covered can help monitor player fatigue and prevent injuries. Teams can **analyze the opposing special teams' patterns,** such as where they usually kick or how their players move during plays. This can also be obtained using real time processing. In terms of **player development**, tracking changes in a player's speed, agility, and positioning over time, teams can fine-tune training regimens. Finally, it also allows teams to evaluate how college players move compared to current NFL players, **making the draft process more data-driven.**

Let's talk about the data

All of the data which we talked about above has to be stored in a sequential and organized manner. We use a spreadsheet-like format called csv (comma separated values to store this data). This allows us to clearly distinguish between the information and also helps us to put it into categories.

Sometimes, the data that we have has to be stored in multiple tables in order to avoid cluttering of information. In this case, the tracking data which is more mathematical data that has a lot of integers has been stored in different dataset. This data is then linked to another dataset which has more of values that we can interpret easily by a primary key. For example, the players dataset that we have contains all the information regarding the players such as their name, height, weight etc.**Our primary key is the "nflid". Each nfl id is unique to the player.** But we have also linked another dataset along with this dataset which is the game dataset. It stores all the information that is related to the game.

Let's Start Working!

For this guidebook, we are going to use Google collab as our development environment. We have gone with google collab as:

- It requires minimal to no setup

- You don't have to download anything on your computer (works on cloud).
- Can access from anywhere without your personal computer.
- Easy to collaborate and share work with others.

Required Python Libraries:

```
import pandas as pd             # Data manipulation with DataFrames.
import matplotlib.pyplot as plt # Visualization with charts and plots.
import seaborn as sns           # Simplified statistical graphics.
import numpy as np              # Math operations and array handling.
import os # Interact with the operating system (e.g., file paths).
import warnings # Manage warning messages.
warnings.filterwarnings('ignore') # Suppress non-critical warnings.
```

Loading Data

Why Load Data?

Before performing any analysis, the first step is to load data into a workspace. Proper data loading ensures that datasets are accessible for cleaning, preprocessing, and visualization.

```
→df_games = pd.read_csv
```

df_games = pd.read_csv(...) is a common way to read a CSV (Comma-Separated Values) file into a Pandas DataFrame (df_games). Here's a breakdown:

- pd.read_csv(...): This function is used to read a CSV file and convert it into a Pandas DataFrame.
- df_games: This is the variable that stores the DataFrame.
- The argument inside pd.read_csv(...) is the file path or file name of the CSV file being read.

Pandas DataFrames provide a structured format similar to Excel tables, making it easy to filter, sort, and modify data. Loading datasets separately enables efficient merging and querying to analyze relationships between different tables. The structure of these datasets is essential for linking player performances with game events and tracking data.

Code:-

```
df_games = pd.read_csv(r"C:\Users\My PC\NFL\data\games.csv")
df_player_play=pd.read_csv(r"C:\Users\My
PC\NFL\data\player_play.csv")
df_players=pd.read_csv(r"C:\Users\My PC\NFL\data\players.csv")
```

```
df_plays=pd.read_csv(r"C:\Users\My PC\NFL\data\plays.csv")
df_tracking_week_1=pd.read_csv(r"C:\Users\My
PC\NFL\data\tracking_week_1.csv")
df_tracking_week_2=pd.read_csv(r"C:\Users\My
PC\NFL\data\tracking_week_2.csv")
df_tracking_week_3=pd.read_csv(r"C:\Users\My
PC\NFL\data\tracking_week_3.csv")
df_tracking_week_4=pd.read_csv(r"C:\Users\My
PC\NFL\data\tracking_week_4.csv")
df_tracking_week_5=pd.read_csv(r"C:\Users\My
PC\NFL\data\tracking_week_5.csv")
df_tracking_week_6=pd.read_csv(r"C:\Users\My
PC\NFL\data\tracking_week_6.csv")
df_tracking_week_7=pd.read_csv(r"C:\Users\My
PC\NFL\data\tracking_week_7.csv")
df_tracking_week_8=pd.read_csv(r"C:\Users\My
PC\NFL\data\tracking_week_8.csv")
df_tracking_week_9=pd.read_csv(r"C:\Users\My
PC\NFL\data\tracking_week_9.csv")
```

Displaying Basic Info

Why Check Column Names?

Understanding column names helps in selecting relevant data for analysis and identifying inconsistencies in data formatting.

Code:

```
# Creating a dictionary with dataset names as keys
dfs = {
    "df_games": df_games,
    "df_player_play": df_player_play,
    "df_players": df_players,
    "df_plays": df_plays,
    "df_tracking_week_1": df_tracking_week_1,
    "df_tracking_week_2": df_tracking_week_2,
    "df_tracking_week_3": df_tracking_week_3,
    "df_tracking_week_4": df_tracking_week_4,
    "df_tracking_week_5": df_tracking_week_5,
    "df_tracking_week_6": df_tracking_week_6,
    "df_tracking_week_7": df_tracking_week_7,
```

```
    "df_tracking_week_8": df_tracking_week_8,
    "df_tracking_week_9": df_tracking_week_9,
}

# Iterating through the dictionary to print dataset names with columns
for name, df in dfs.items():
    print(f"{name}: {df.columns.tolist()}")
```

Explanation

"For" loop is used to iterate through the dictionary

f"{...}": This is an f-string, a way to format strings in Python.
{name}: Assumes that name is a variable holding a string (e.g., the name of the dataset or DataFrame).
{df.columns.tolist()}:

- df.columns retrieves the column names of the DataFrame.
- .tolist() converts the column names from an Index object to a regular Python list.
- print(...): Displays the formatted output.

This loop prints the column names of each dataset, allowing us to inspect the available features. Knowing the structure of each dataset is essential for merging data tables and selecting specific columns for analysis.

Output

Sample of the games data

	gameId	season	week	gameDate	gameTimeEastern	homeTeamAbbr	visitorTeamAbbr	homeFinalScore	visitorFinalScore
0	2022090800	2022	1	09/08/2022	20:20:00	LA	BUF	10	31
1	2022091100	2022	1	09/11/2022	13:00:00	ATL	NO	26	27
2	2022091101	2022	1	09/11/2022	13:00:00	CAR	CLE	24	26
3	2022091102	2022	1	09/11/2022	13:00:00	CHI	SF	19	10
4	2022091103	2022	1	09/11/2022	13:00:00	CIN	PIT	20	23

Summary Statistics

To gain an overview of our dataset, we can use descriptive summary functions in Python and R. In R, we utilize the `summary()` function, which provides insights into the distribution of numerical values in a dataset.

Code Example:

```
# Load necessary libraries
library(dplyr)
library(ggplot2)
```

```
# Load the datasets
games <- read.csv("Games.csv")
players <- read.csv("Players.csv")
player_plays <- read.csv("Player-Plays.csv")
week1 <- read.csv("Week1.csv")

# Basic summary statistics
summary(games)
summary(players)
summary(player_plays)
summary(week1)
```

Output:

```
Summary of Games dataset:
     gameId              season          week         gameDate
 Min.   :2.022e+09   Min.   :2022   Min.   :1.000   Length:136
 1st Qu.:2.022e+09   1st Qu.:2022   1st Qu.:3.000   Class :character
 Median :2.022e+09   Median :2022   Median :5.000   Mode  :character
 Mean   :2.022e+09   Mean   :2022   Mean   :4.846
 3rd Qu.:2.022e+09   3rd Qu.:2022   3rd Qu.:7.000
 Max.   :2.022e+09   Max.   :2022   Max.   :9.000
 gameTimeEastern    homeTeamAbbr       visitorTeamAbbr    homeFinalScore
 Length:136         Length:136         Length:136         Min.   : 3.00
 Class :character   Class :character   Class :character   1st Qu.:17.00
 Mode  :character   Mode  :character   Mode  :character   Median :22.50
                                                          Mean   :22.67
                                                          3rd Qu.:27.00
                                                          Max.   :49.00

 visitorFinalScore
 Min.   : 0.00
 1st Qu.:14.75
 Median :20.00
 Mean   :20.95
 3rd Qu.:27.00
```

Here, min is the minimum value, max is the maximum value, mean is the average value, median is the middle value, 1st quartile (Q1) is the 25th percentile, and 3rd quartile (Q3) is the 75th percentile.

Great, now you have a basic idea on what the dataset is about and the attributes it has. Now before proceeding toward visualizing the data (the fun part) we need to first check if there are any abnormalities in the dataset. We can do this by cleaning and normalizing the dataset

Data Cleaning and Normalization

Before visualizing data, it is crucial to clean and preprocess it. The raw data often contains missing values, duplicate records, and inconsistencies that can lead to incorrect analyses. **Data cleaning involves handling missing values by filling them with appropriate values** (such as the mean or median), removing duplicates, and ensuring consistency in formatting.

Normalization is another essential step. It ensures that numerical data is scaled properly, especially when combining multiple datasets with different ranges. By standardizing data, we ensure that no single variable disproportionately influences the analysis.

Example Techniques that we used:

- Handling Missing Values: Replace missing numeric values with the mean or median.
- Removing Duplicates: Identify and remove duplicate entries to maintain data integrity.
- Normalization: Scale numerical data to a common range (e.g., between 0 and 1) to ensure fair comparisons.

Summary Statistics of dataframe:

→ Summarize_dataframe(df_games)

The summarize_dataframe(df) function creates a customized summary of the dataframe. It lists all column names using df.columns, shows the data type of each column with df.dtypes.values, counts missing values using df.isnull().sum().values, and displays the number of unique values per column with df.nunique().values. The resulting table is visually enhanced with a purple gradient using .style.background_gradient(cmap='Purples'), making it easier to interpret the data.

Code:

```
df_games.describe().style.background_gradient(cmap='Purples')

def summarize_dataframe(df):
    summary = pd.DataFrame({
        'Column': df.columns,
        'Data Type': df.dtypes.values,
        'Missing Values': df.isnull().sum().values,
        'Unique Values': df.nunique().values
    })
    return summary

summarize_dataframe(df_games).style.background_gradient(cmap='Purples
')
```

	Column	Data Type	Missing Values	Unique Values
0	gameId	int64	0	136
1	season	int64	0	1
2	week	int64	0	9
3	gameDate	object	0	27
4	gameTimeEastern	object	0	8
5	homeTeamAbbr	object	0	32
6	visitorTeamAbbr	object	0	32
7	homeFinalScore	int64	0	38
8	visitorFinalScore	int64	0	35

Identifying and Handling Missing Values (Page 15-17)

Why Handle Missing Values?

Missing values can lead to biased analysis and errors in computations. It is essential to detect and handle them before performing further data processing.

Identify Missing Values

→`df.isnull().sum()`

- df.isnull() creates a boolean mask where True represents missing (NaN) values.
- .sum() counts the number of missing values for each column.

```
for name, df in dfs.items():
print(f"{name} Missing
Values:\n{df.isnull().sum()}\n{'
-'*50}")
```

```
df_games

Missing Values:
gameId                0
season                0
week                  0
gameDate              0
gameTimeEastern       0
homeTeamAbbr          0
visitorTeamAbbr       0
homeFinalScore        0
visitorFinalScore     0
dtype: int64
------------------------------------------------
df_player_play Missing Values:
gameId                                      0
playId                                      0
nflId                                       0
teamAbbr                                    0
hadRushAttempt                              0
rushingYards                                0
hadDropback                                 0
passingYards                                0
sackYardsAsOffense                          0
hadPassReception                            0
receivingYards                              0
wasTargettedReceiver                        0
yardageGainedAfterTheCatch
0
fumbles                                     0
fumbleLost                                  0
fumbleOutOfBounds                           0
assistedTackle                              0
forcedFumbleAsDefense
0
halfSackYardsAsDefense
0
passDefensed                                0
quarterbackHit                              0
sackYardsAsDefense                          0
safetyAsDefense                             0

------------------------------------------------
soloTackle                                  0
tackleAssist                                0
tackleForALoss                              0
tackleForALossYardage                       0
hadInterception                             0
interceptionYards                           0
fumbleRecoveries                            0
fumbleRecoveryYards                         0
penaltyYards                                0
penaltyNames                           354351
wasInitialPassRusher
247447
causedPressure                              0
timeToPressureAsPassRusher
350399
getOffTimeAsPassRusher
306695
inMotionAtBallSnap
246879
shiftSinceLineset                      178549
motionSinceLineset
264489
wasRunningRoute
311948
routeRan                               311948
blockedPlayerNFLId1
305623
blockedPlayerNFLId2
350354
blockedPlayerNFLId3
354720
pressureAllowedAsBlocker
301683
timeToPressureAllowedAsBlocker
350647
pff_defensiveCoverageAssignment
288953
pff_primaryDefensiveCoverageMatchupNflI
d      311243
pff_secondaryDefensiveCoverageMatchupN
flId    352340
dtype: int64
```

Drop Rows with Critical Missing Values

Code

```
critical_columns = ['gameId', 'playId', 'nflId']
for name, df in dfs.items():
    df.dropna(subset=[col for col in critical_columns if col in
df.columns], inplace=True)
```

`critical_columns`: List of key columns that must not have missing (NaN) values.

- **`for name, df in dfs.items()`: Iterates through the dictionary of DataFrames.**
  - **name: The dataset name (e.g., "Games", "Players").**
  - **df: The actual DataFrame.**
- **`df.dropna(subset=[col for col in critical_columns if col in df.columns], inplace=True):[col for col in critical_columns if col in df.columns]:`**
    - **Ensures only existing columns are considered (some DataFrames may not have all critical_columns).**
  - **`df.dropna(subset=..., inplace=True)`:**
    - **Removes rows where any of the specified critical_columns contain NaN.**
    - **inplace=True modifies the DataFrame directly instead of creating a copy.**

Fill Numeric Columns with Mean/Median.

Code

```
for name, df in dfs.items():
    numeric_cols = df.select_dtypes(include=['number']).columns
    df[numeric_cols] =
df[numeric_cols].fillna(df[numeric_cols].median())
```

Explanation

- `df.select_dtypes(include=['number']).columns`:
  Selects only numeric columns (int, float).
- `df[numeric_cols].fillna(df[numeric_cols].median())` **:Replaces missing values (NaN) in numeric columns with their median.**

Why median: More robust than the mean since it's less affected by outliers.

Fill Categorical Columns with Mode

Code

```
for name, df in dfs.items():
    categorical_cols = df.select_dtypes(include=['object']).columns
    for col in categorical_cols:
        df[col].fillna(df[col].mode()[0], inplace=True)
```

Explanation

`df.select_dtypes(include=['object']).columns` : **Selects columns with categorical (string) data.**

`df[col].mode()[0]` :

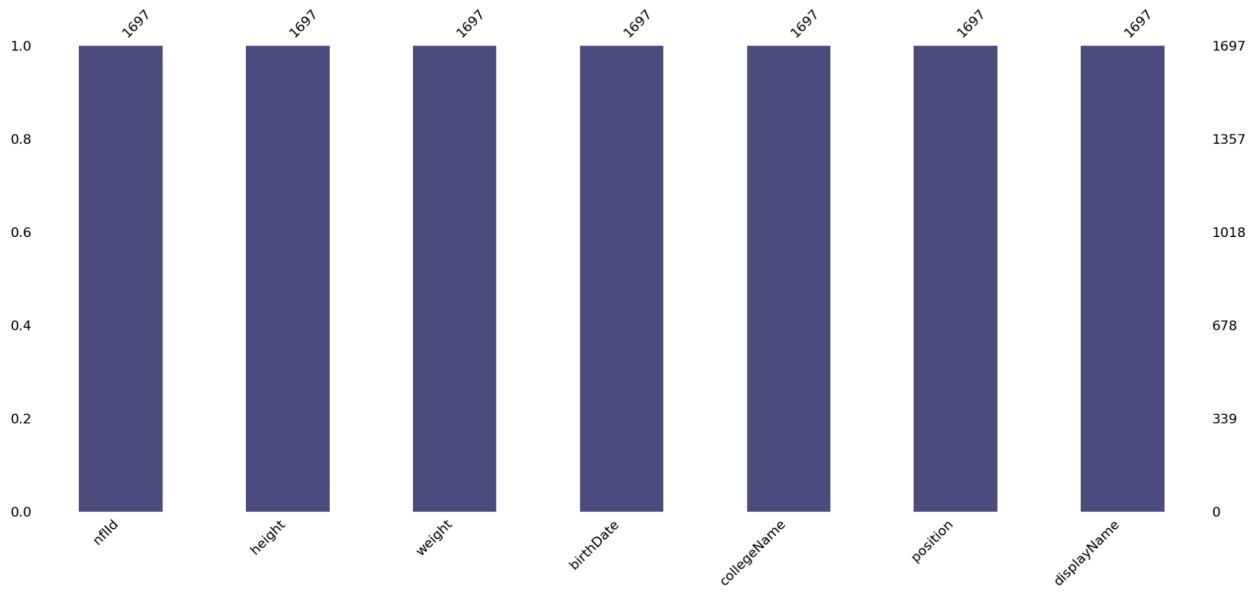
- **mode() returns the most frequent value in a column.**
- **[0] selects the first mode (in case of ties).**

`inplace=True` : **Modifies the DataFrame directly (no need to reassign df[col]).**

Visualizing Missing values

We used the Missingno library for visualizing missing values.

msno.bar(df_players, color=(0.3,0.3,0.5))



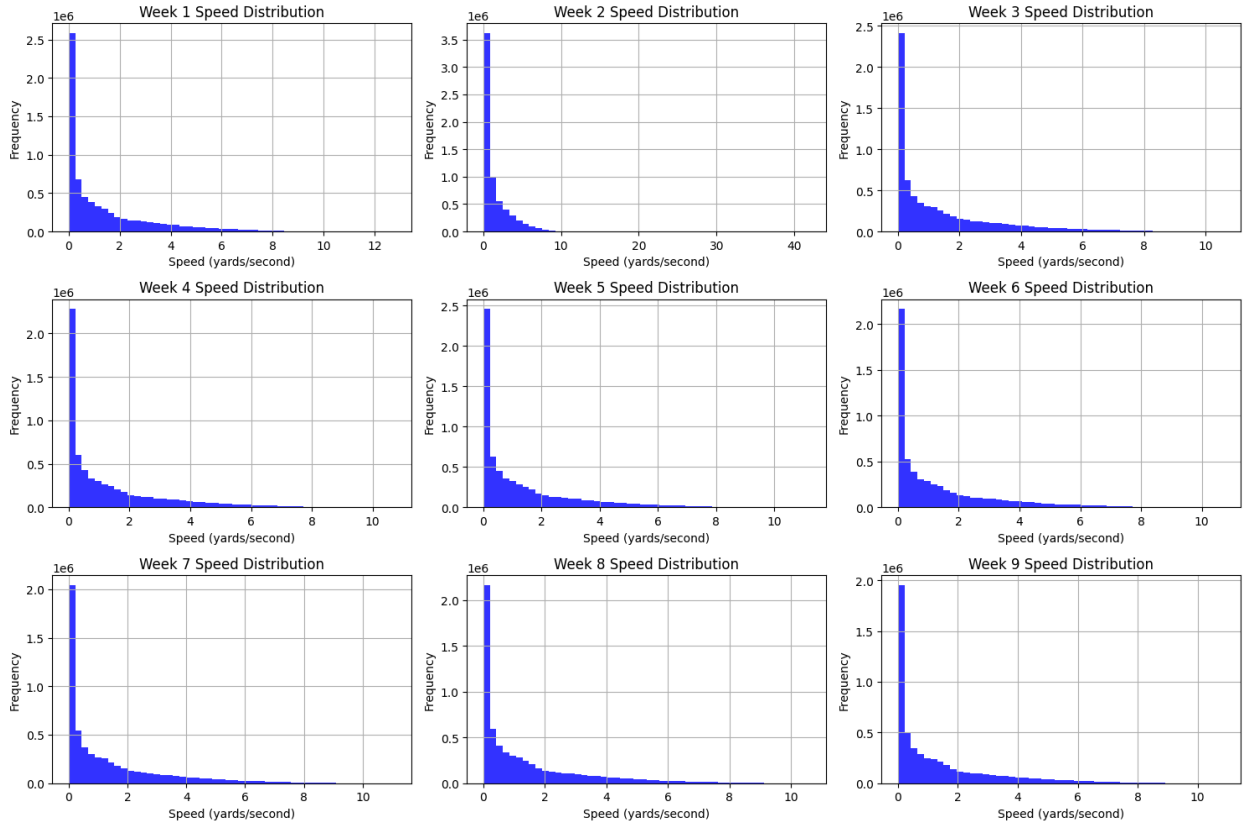
Descriptive Statistics:

```
df_tracking_week_1[['s', 'a', 'dis']].describe()
```

	s	a	dis
count	6.795800e+06	6.795800e+06	6.795800e+06
mean	1.338137e+00	9.340563e-01	1.363029e-01
std	1.731075e+00	1.115224e+00	1.734388e-01
min	0.000000e+00	0.000000e+00	0.000000e+00
25%	8.000000e-02	8.000000e-02	1.000000e-02
50%	5.800000e-01	5.400000e-01	6.000000e-02
75%	1.960000e+00	1.350000e+00	2.000000e-01
max	1.280000e+01	2.315000e+01	4.280000e+00

Subplots

Histograms showing the speed distribution (s) of players for each week's tracking data (weeks 1 to 9). Each subplot represents a week, with the x-axis showing speed in yards/second and the y-axis showing the frequency of occurrences.



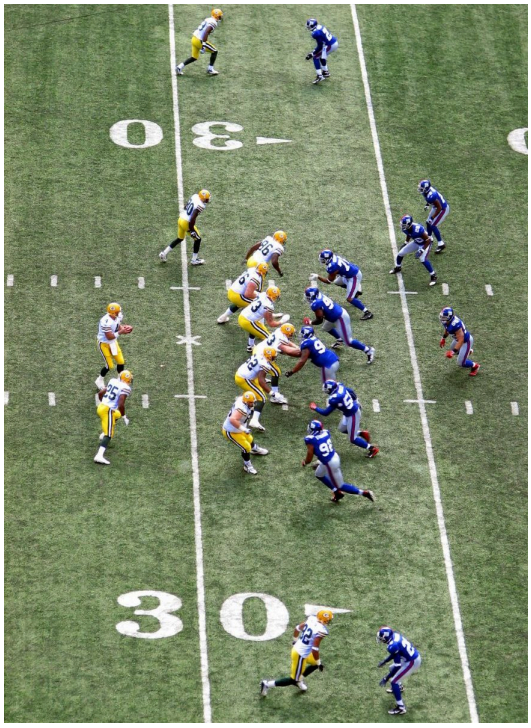
Formations

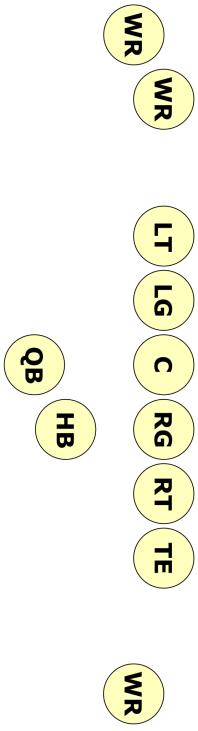
Player formations refer to the pre-snap arrangement of offensive and defensive players on the field. Formations are essential for analyzing team tendencies, as they provide insight into potential plays before the snap.

Types of Player Formations

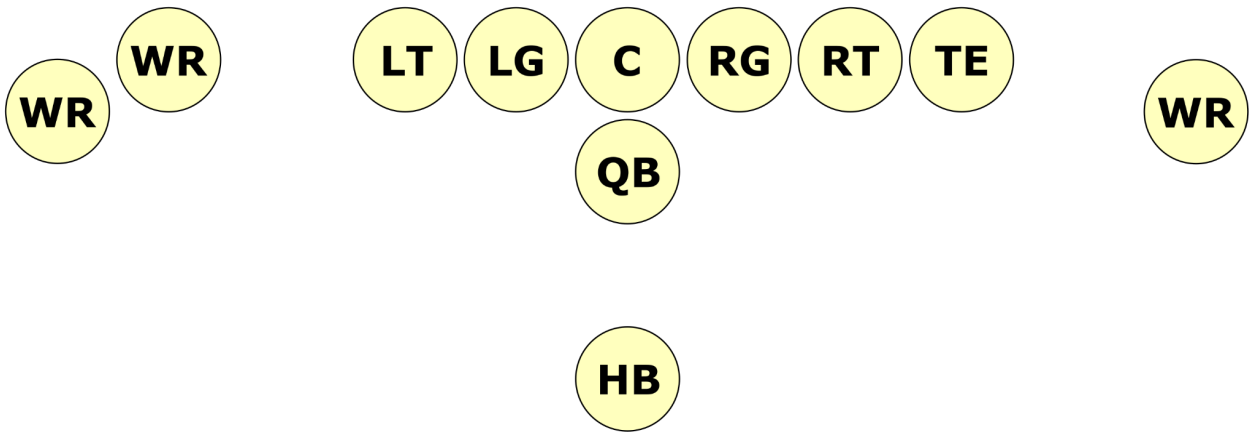
Offensive Formations

1. **Shotgun**: The quarterback (QB) stands several yards behind the center, allowing more time to read the defense. This formation is primarily used for passing plays, as the increased distance from the line of scrimmage allows the QB to scan the field more effectively. It is also commonly seen in RPO (Run-Pass Option) schemes, where the QB can decide to hand off, run, or pass based on defensive movement. Shotgun Spread emphasizes quick, short passes, while the Pistol formation, a hybrid between Shotgun and I-Formation, places a running back directly behind the QB for a balanced attack.

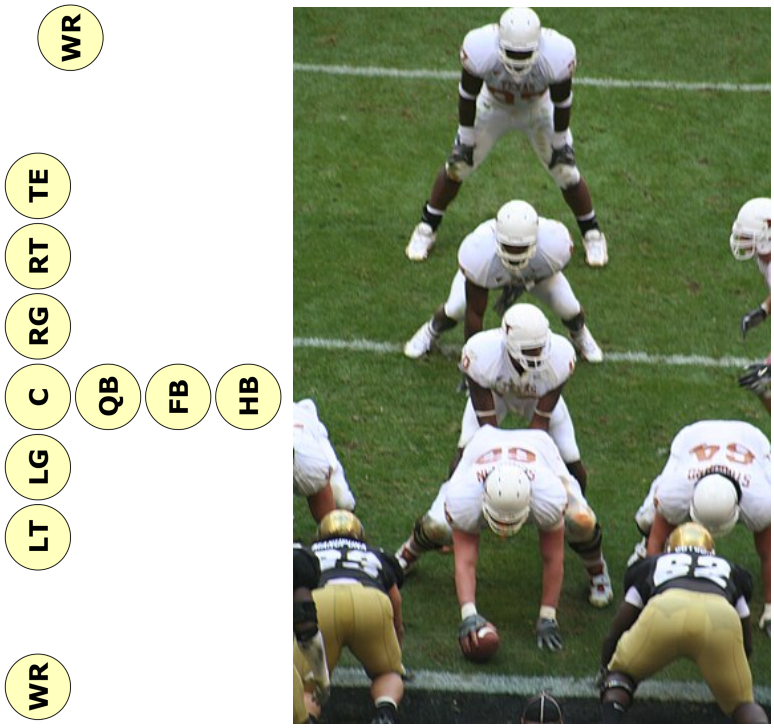




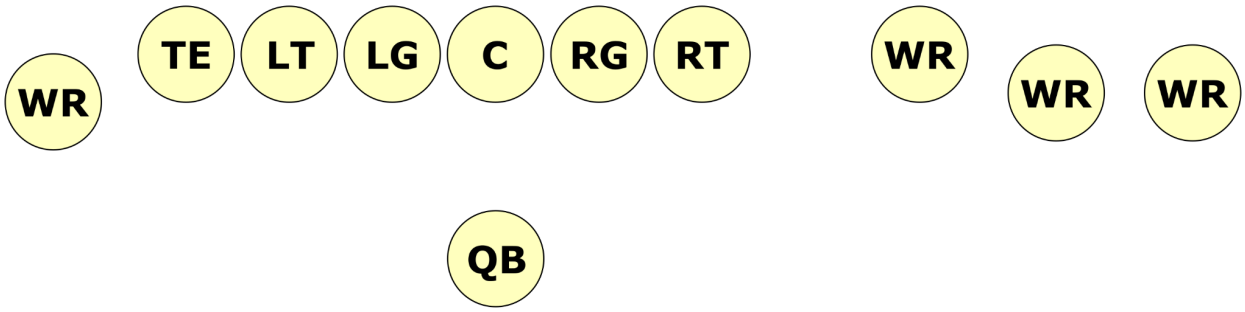
2. **Singleback**: The **Singleback formation** features a quarterback under center with only one running back (RB) behind him, eliminating the fullback often found in power formations. This formation is frequently used for **play-action plays**, where the quarterback fakes a handoff to deceive defenders before passing. It is effective for **zone running schemes**, allowing the RB to read blocking lanes and make decisive cuts.



3. **I-Formation:** The **I-Formation** is a classic offensive setup where the fullback (FB) and running back (RB) line up directly behind the quarterback, forming an "I" shape. This formation is heavily used for **power running plays**, as the fullback acts as a lead blocker to clear space for the RB.

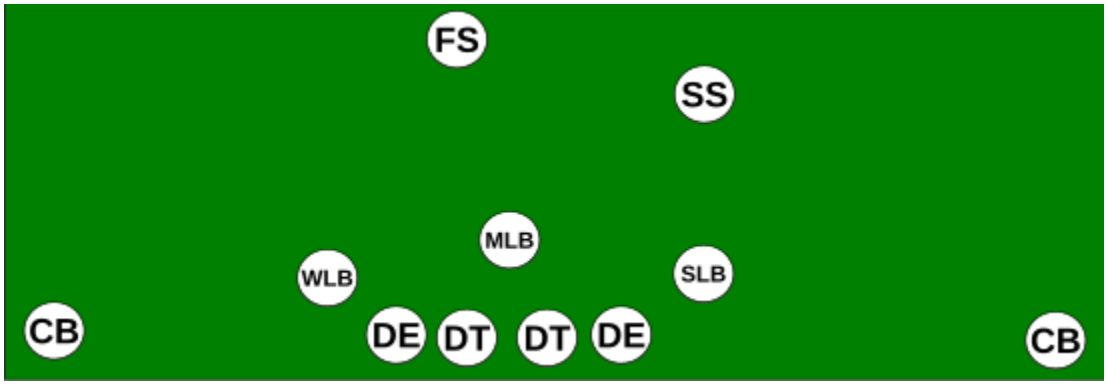


4. **Trips**: The **Trips formation** aligns three wide receivers on one side of the field, forcing the defense to adjust its coverage. This setup is designed to create mismatches, particularly against **zone defenses**, as it can overload one side of the field and create easy passing lanes.

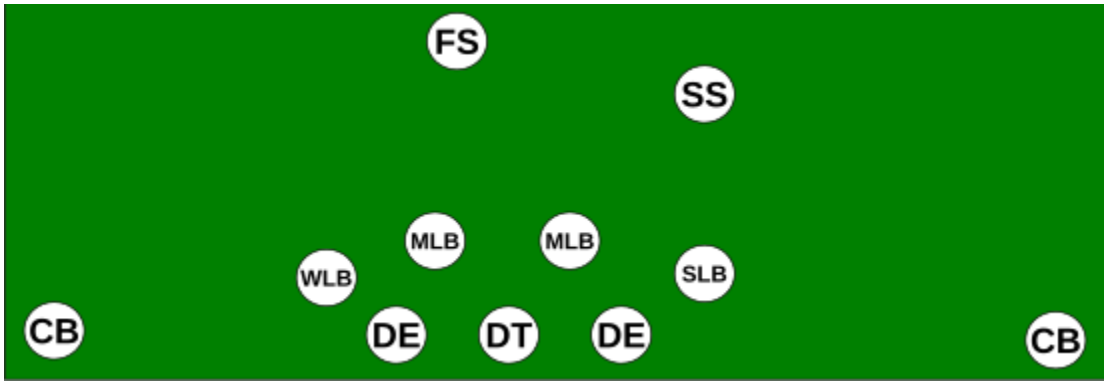


Defensive Formations

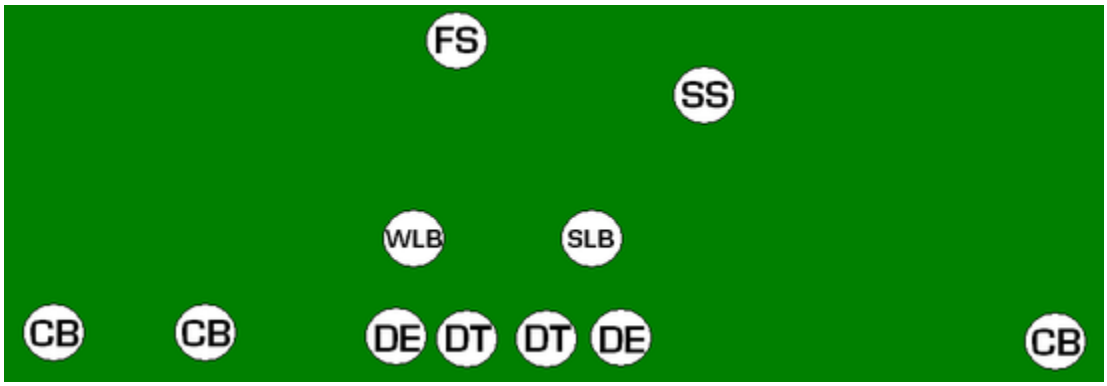
4-3 Defense: The **4-3 defense** consists of four defensive linemen and three linebackers, offering a balanced approach to stopping both the run and pass. This is one of the most common defensive setups because it provides **strong gap control** against the run while allowing flexibility in pass coverage. However, it can struggle against pass-heavy offenses that force linebackers into coverage mismatches.



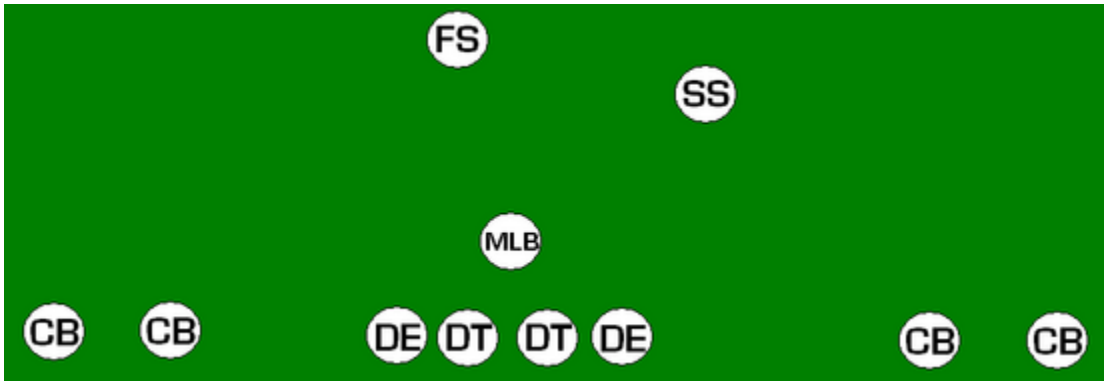
3-4 Defense: The **3-4 defense** features three defensive linemen and four linebackers, giving it greater versatility in **blitzing schemes** and coverage disguises. Unlike the 4-3, where defensive linemen generate most of the pass rush, the 3-4 relies on its linebackers to create pressure.



Nickel Defense: The **Nickel defense** replaces one linebacker with a fifth defensive back (DB), improving coverage against pass-heavy offenses. The additional defensive back allows for more **flexibility in coverage**, enabling defenses to play **man-to-man or zone schemes more effectively**.



Dime Defense: The **Dime defense** takes the Nickel concept a step further by adding a sixth defensive back while removing either a second linebacker or a defensive lineman. This formation is specifically designed for **obvious passing situations**, such as third-and-long plays, where additional coverage is more valuable than run defense.



Identifies where defenders are most concentrated

```
# Define Game ID for analysis
GAME_ID = 2022090800  # Change this for different games

# Filter data for the selected game
df_game = df_tracking_week_1[df_tracking_week_1['gameId'] == GAME_ID]

# Merge with player positions
df_merged = df_game.merge(df_players[['nflId', 'position']],
on='nflId', how='left')

# Filter defensive players
defensive_positions = ['DE', 'NT', 'SS', 'FS', 'OLB', 'DT', 'CB',
'ILB', 'MLB', 'DB', 'LB']
df_defense =
df_merged[df_merged['position'].isin(defensive_positions)]

# Plot Defensive Heatmap
plt.figure(figsize=(12, 6))
sns.kdeplot(x=df_defense['x'], y=df_defense['y'], cmap='Reds',
fill=True)
plt.xlim(0, 120)
plt.ylim(0, 53.3)
plt.xlabel("Field Length (yards)")
plt.ylabel("Field Width (yards)")
plt.title(f"Defensive Heatmap - Game {GAME_ID}")
plt.xticks(range(0, 130, 10))

# Add vertical field markings
for x in range(0, 130, 10):
    plt.axvline(x, color='gray', linestyle='--', alpha=0.6)

plt.show()
```

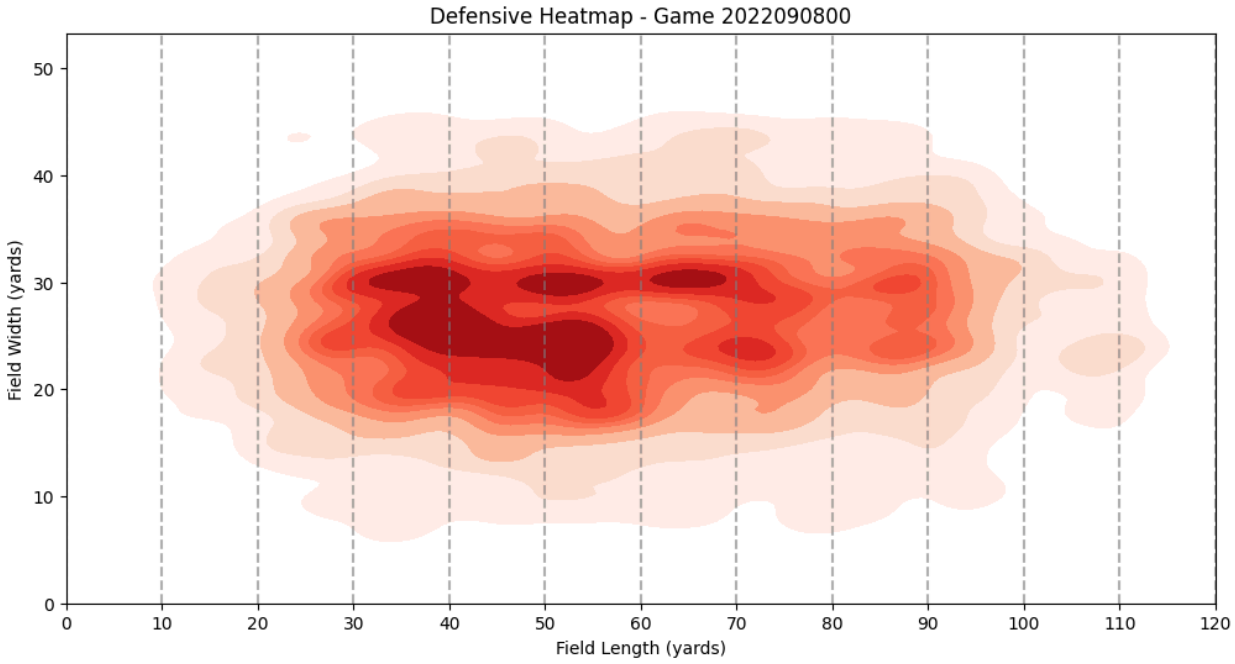
Explanation

→ defensive_positions = ['DE', 'NT', 'SS', 'FS', 'OLB', 'DT', 'CB', 'ILB', 'MLB', 'DB', 'LB']
df_defense = df_merged[df_merged['position'].isin(defensive_positions)]

- **Creates a list of defensive positions.**
- **Filters df_merged to include only defensive players.**

→ `sns.kdeplot(x=df_defense['x'], y=df_defense['y'], cmap='Reds', fill=True)`

- **sns.kdeplot** plots a **Kernel Density Estimate (KDE) heatmap**.
- Uses cmap='Reds' to show density in shades of red.
- fill=True ensures a smooth density visualization.



How It Relates to Pre-Snap Behavior Analysis

In American football, **pre-snap** and **post-snap** refer to different phases of a play:

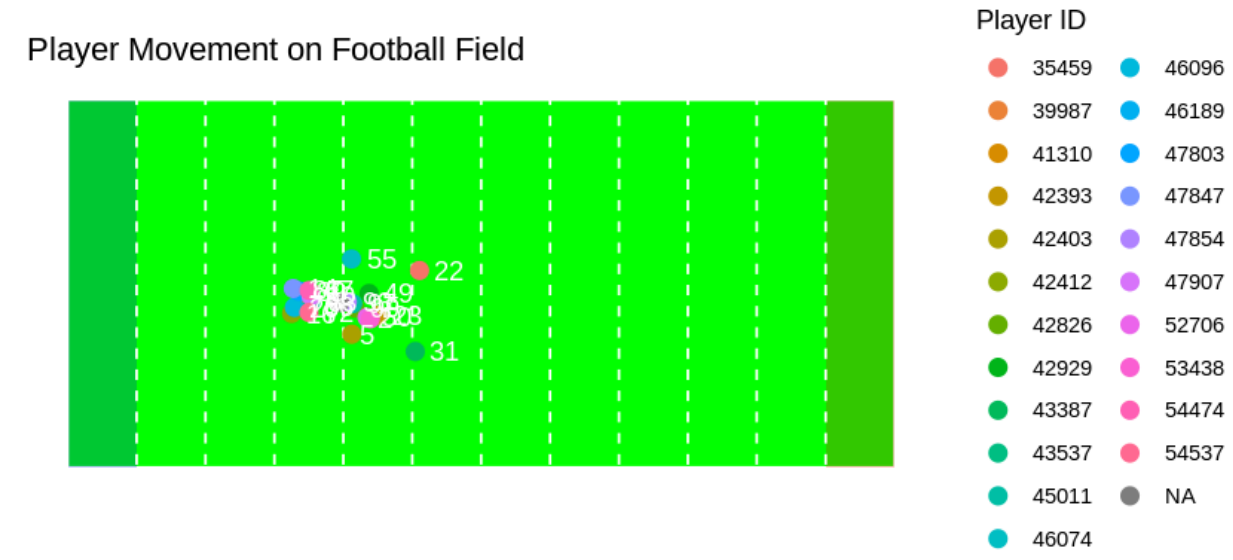
1. **Pre-snap:**
    - This is the period before the ball is snapped (hiked) to the quarterback.
    - Teams line up in their formations, and the quarterback may call audibles (changing the play) based on the defense.
    - Defenders may shift their alignment, showing or disguising their strategy.
    - Motion by offensive players (e.g., a wide receiver moving across the formation) can happen before the snap to create favorable matchups or confuse the defense.
2. **Post-snap:**
    - This begins the moment the ball is snapped.
    - Offensive players execute their routes, blocks, or ball-carrying assignments.
    - Defensive players react, trying to stop the offense by tackling the ball carrier, sacking the quarterback, or covering receivers.
    - The play develops dynamically based on how both teams react to each other.

The formation provides predictive value for determining whether a team is likely to run or pass. Defensive formations help **anticipate coverages and blitz schemes.** Tracking formations over time allows for pattern recognition, aiding in metric development for tendencies.

Code:

```
library(ggplot2)
library(dplyr)

week1_data <- read.csv("gdrive/MyDrive/nfl/Week1.csv")
# Filter data for a specific game and play
game_id <- 2022091200
play_id <- 64
filtered_data <- week1_data %>%
  filter(gameId == game_id, playId == play_id)
# Draw the football field
draw_field <- function() {
  ggplot() +
    # Field outline
    geom_rect(aes(xmin = 0, xmax = 120, ymin = 0, ymax = 53.3), fill = "green",
color = "white") +
    # End zones
    geom_rect(aes(xmin = 0, xmax = 10, ymin = 0, ymax = 53.3), fill = "blue",
alpha = 0.2) +
    geom_rect(aes(xmin = 110, xmax = 120, ymin = 0, ymax = 53.3), fill = "red",
alpha = 0.2) +
    # Yard lines
```

```
    geom_vline(xintercept = seq(10, 110, by = 10), color = "white", linetype =
"dashed") +
    # Labels
    labs(title = "Player Movement on Football Field",
         x = "Yard Line",
         y = "Field Width") +
    theme_minimal() +
    theme(panel.grid = element_blank(),
          axis.text = element_blank(),
          axis.title = element_blank())
}
# Add player positions
plot_player_positions <- function(frame_data) {
  draw_field() +
    geom_point(data = frame_data, aes(x = x, y = y, color = as.factor(nflId)),
size = 3) +
    geom_text(data = frame_data, aes(x = x, y = y, label = jerseyNumber), hjust
= -0.5, color = "white") +
    scale_color_discrete(name = "Player ID") +
    coord_fixed(ratio = 1)
}
# Visualize a single frame
frame_id <- 1  # Change this to visualize different moments in time
frame_data <- filtered_data %>%
  filter(frameId == frame_id)

plot <- plot_player_positions(frame_data)
print(plot)
```



Heatmaps for Player Positioning

What is a Heatmap?

A heatmap is a visualization technique that represents data density using color intensity. Darker regions indicate higher concentrations of data points, helping analysts understand player positioning trends. We can see Player positions and ball movement heatmaps to visualize density of player activity during the game.

```
import seaborn as sns
import matplotlib.pyplot as plt

heatmap_data = df_tracking_week_1[['x', 'y']].dropna().sample(n=500,
random_state=42)

plt.figure(figsize=(12, 6))
sns.kdeplot(x=heatmap_data['x'], y=heatmap_data['y'], cmap='Blues',
fill=True)
plt.xlim(0, 120)
plt.ylim(0, 53.3)
plt.xlabel("Field Length (yards)")
plt.ylabel("Field Width (yards)")
plt.title("Player Position Heatmap")
plt.show()
```

Function Definition

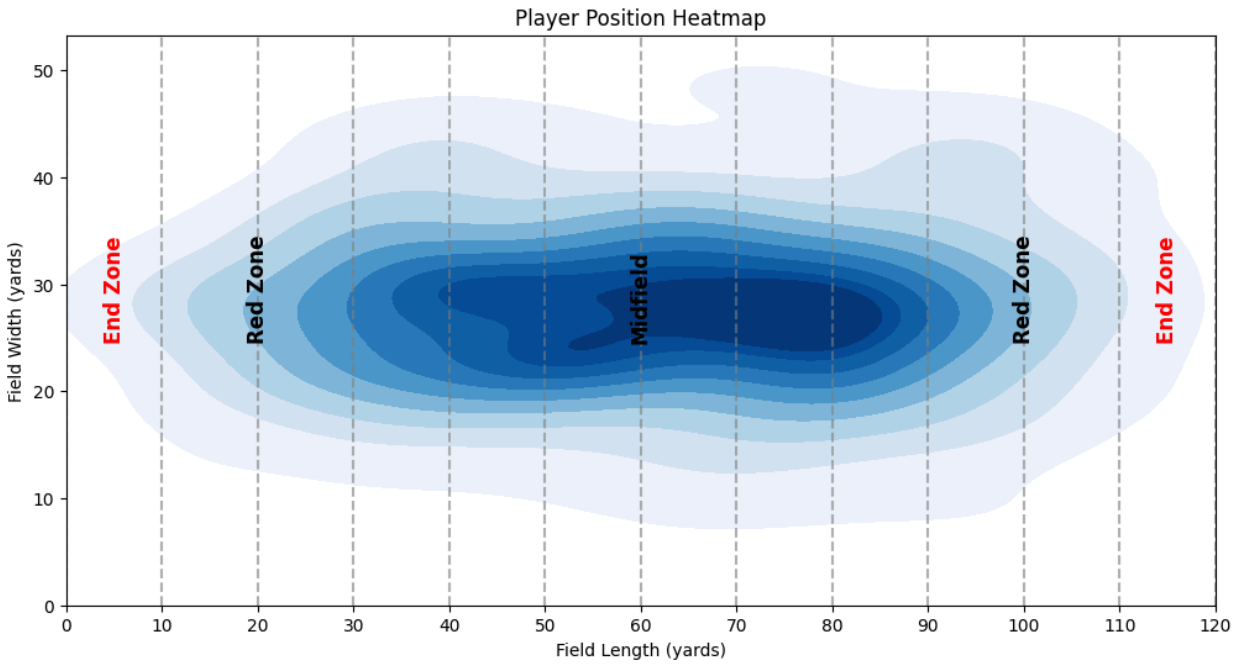
- `sns.kdeplot()`: Creates a kernel density estimation (KDE) plot for visualizing player movement trends.
- `.dropna()`: Removes rows with missing values to ensure accurate plotting.
- `.sample(n, random_state)`: Selects a subset of data points for faster computation.

Detailed Explanation:

- The X-axis represents the length of the football field, while the Y-axis represents the field width.
- This visualization helps coaches and analysts understand player positioning patterns and tendencies over multiple plays.

Code:

```
# Sample 500 data points for visualization
heatmap_data = df_tracking_week_1[['x', 'y']].dropna().sample(n=500,
random_state=42)

# Create the heatmap
plt.figure(figsize=(12, 6))
sns.kdeplot(x=heatmap_data['x'], y=heatmap_data['y'], cmap='Blues',
fill=True)

# Set the field dimensions (NFL field: 0 to 120 yards in length, 0 to
53.3 yards in width)
plt.xlim(0, 120)
plt.ylim(0, 53.3)

# Set axis labels and title
plt.xlabel("Field Length (yards)")
plt.ylabel("Field Width (yards)")
plt.title("Player Position Heatmap")

# Set x-axis ticks at every 10-yard mark
plt.xticks(range(0, 130, 10))

# Add vertical lines at every 10 yards to mimic field markings
for x in range(0, 130, 10):
    plt.axvline(x, color='gray', linestyle='--', alpha=0.6)

# Add field section labels
plt.text(5, 25, "End Zone", fontsize=12, color='red',
fontweight='bold', ha='center',rotation=90)
plt.text(20, 25, "Red Zone", fontsize=12, color='black',
fontweight='bold', ha='center',rotation=90)
plt.text(60, 25, "Midfield", fontsize=12, color='black',
fontweight='bold', ha='center',rotation=90)
plt.text(100, 25, "Red Zone", fontsize=12, color='black',
fontweight='bold', ha='center',rotation=90)
plt.text(115, 25, "End Zone", fontsize=12, color='red',
fontweight='bold', ha='center',rotation=90)

# Show the plot
plt.show()
```



Correlation Matrix

A **correlation matrix** is a table that displays the correlation coefficients between multiple variables in a dataset. Each cell in the matrix shows the correlation between two variables, measuring how they move together.

Key Aspects of a Correlation Matrix:

- **Values range from -1 to 1**:
  - **+1**: Perfect positive correlation (when one variable increases, the other increases).
  - **-1**: Perfect negative correlation (when one variable increases, the other decreases).
  - **0**: No correlation (the variables do not have a linear relationship).
- The diagonal of the matrix always contains

Code:-

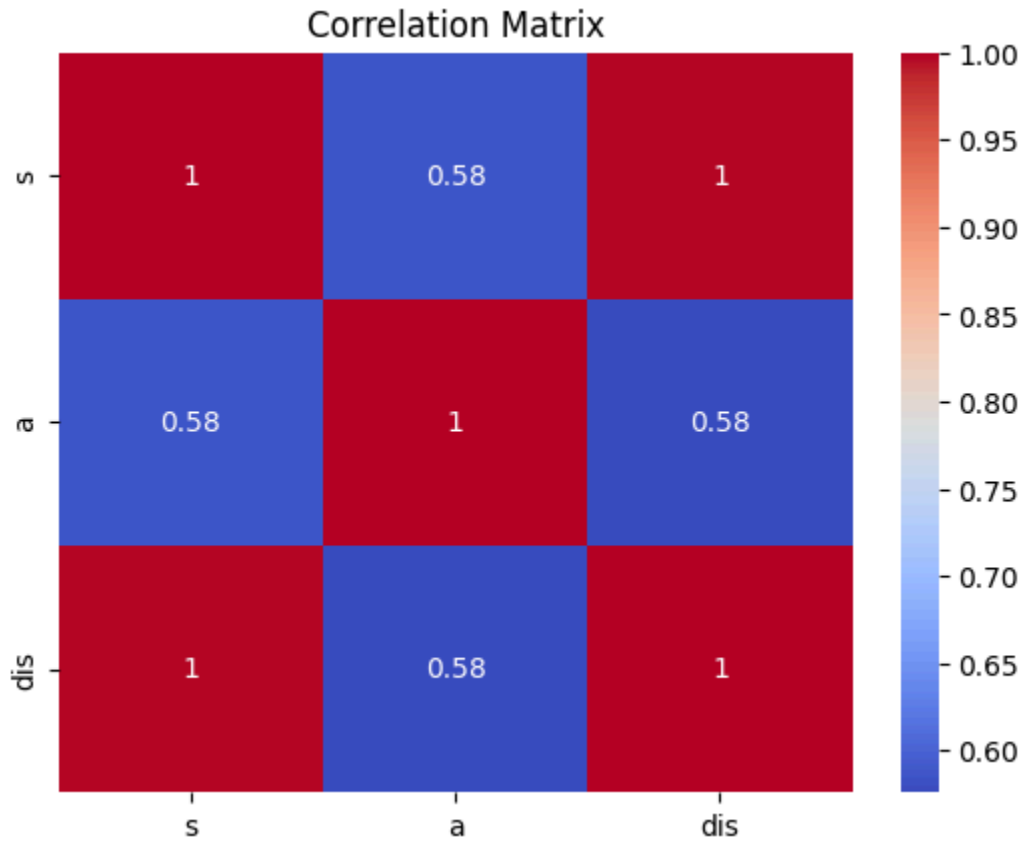
```
corr = df_tracking_week_1[['s', 'a', 'dis']].corr()

sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()
```

```
→ df_tracking_week_1[['s', 'a', 'dis']].corr()
```

- **Computes the correlation matrix for the selected columns ('s', 'a', 'dis').**
- **Uses Pearson's correlation by default.**

```
→ sns.heatmap(corr, annot=True, cmap='coolwarm')
```

- **Creates a heatmap using Seaborn.**
- **annot=True → Displays correlation values on the heatmap.**
- **cmap='coolwarm' → Uses a blue-red colormap where:**
  - **Blue = Positive Correlation**
  - **Red = Negative Correlation**



The heatmap suggests that s and dis are strongly related, while a has a weaker but still positive relationship with both.

Distances and efficiency ratio

```
# Compute total distance traveled by summing 'dis' column
total_distance = df_tracking_week_1.groupby(['gameId', 'playId',
'nflId'])['dis'].sum().reset_index()

# Compute Euclidean distance (straight-line distance from start to
end)
start_positions = df_tracking_week_1.groupby(['gameId', 'playId',
'nflId']).first().reset_index()
end_positions = df_tracking_week_1.groupby(['gameId', 'playId',
'nflId']).last().reset_index()

start_positions['euclidean_distance'] = np.sqrt(
    (end_positions['x'] - start_positions['x'])**2 +
    (end_positions['y'] - start_positions['y'])**2
)

# Merge distances and compute efficiency ratio
distance_features = total_distance.merge(start_positions[['gameId',
'playId', 'nflId', 'euclidean_distance']], on=['gameId', 'playId',
'nflId'])
distance_features['movement_efficiency'] =
distance_features['euclidean_distance'] / distance_features['dis']

print(distance_features.head())
```

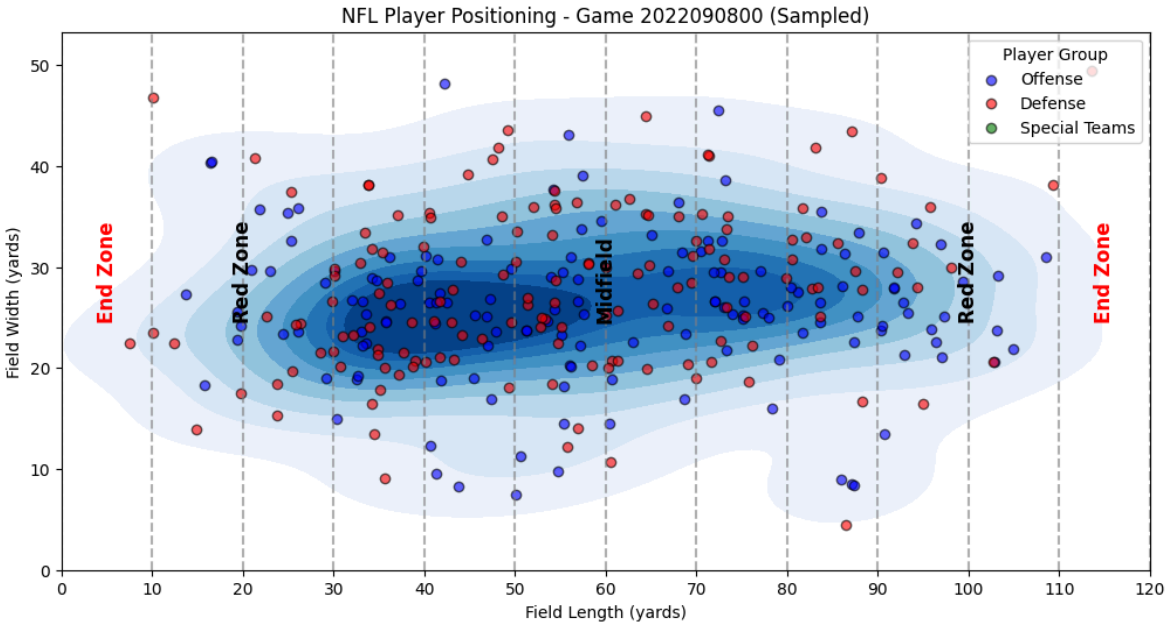
	gameId	playId	nflId	dis	euclidean_distance	movement_efficiency
0	2022090800	56	35472.0	9.67	2.789588	0.288479
1	2022090800	56	38577.0	14.17	5.027733	0.354815
2	2022090800	56	41239.0	18.23	7.479285	0.410273
3	2022090800	56	42392.0	10.27	2.383653	0.232099
4	2022090800	56	42489.0	43.65	12.845283	0.294279

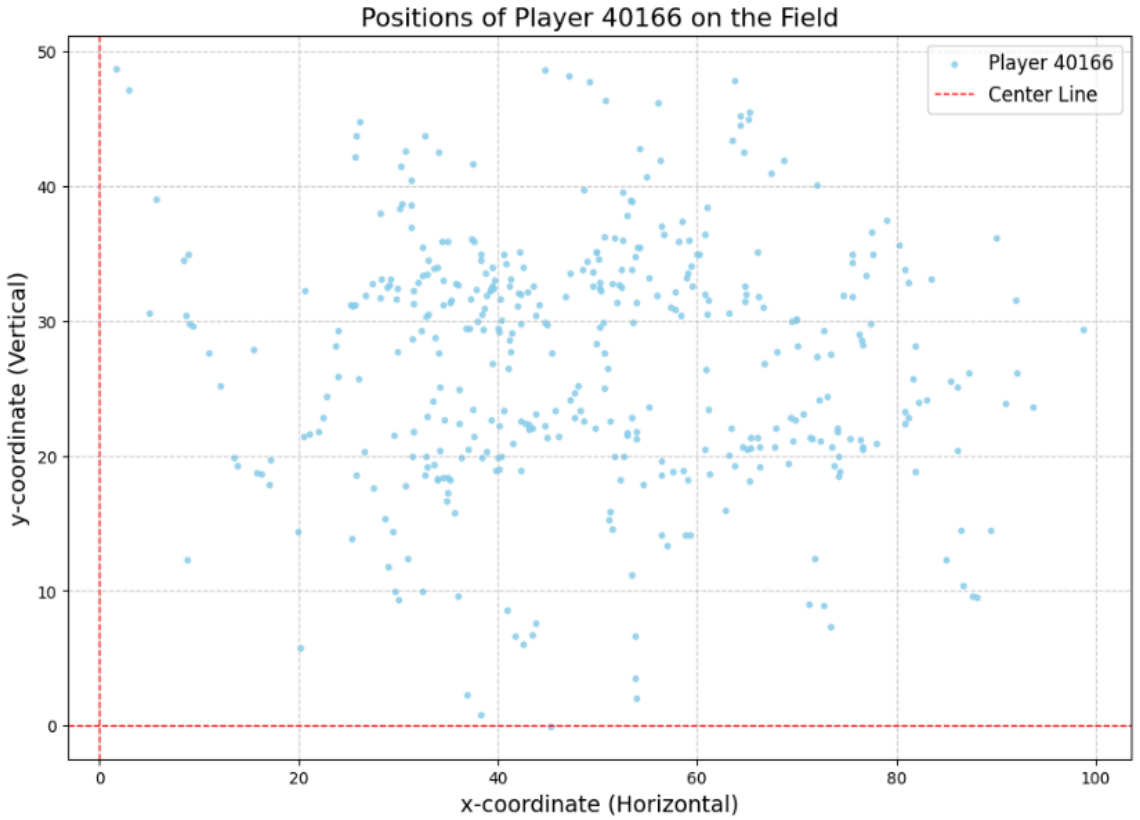
Position of Players in specific Game ID

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Select a specific game ID
GAME_ID = 2022090800
df_game = df_tracking_week_1[df_tracking_week_1['gameId'] == GAME_ID]

# Merge tracking data with player positions
df_merged = df_game.merge(df_players[['nflId', 'position']],
on='nflId', how='left')

# Drop NaNs
df_merged = df_merged[['x', 'y', 'position']].dropna()

# Sample 300 players to speed up visualization
df_sample = df_merged.sample(n=300, random_state=42)  # Adjust 'n' as
needed

# Define position groups
offense_positions = ['QB', 'T', 'TE', 'WR', 'G', 'RB', 'C', 'FB']
defense_positions = ['DE', 'NT', 'SS', 'FS', 'OLB', 'DT', 'CB',
'ILB', 'MLB', 'DB', 'LB']
special_positions = ['K', 'P', 'LS']

# Classify players into groups
def classify_position(pos):
    if pos in offense_positions:
        return 'Offense'
    elif pos in defense_positions:
        return 'Defense'
    elif pos in special_positions:
        return 'Special Teams'
    else:
        return 'Unknown'

df_sample['group'] = df_sample['position'].apply(classify_position)

# Define color mapping
```

```
position_colors = {'Offense': 'blue', 'Defense': 'red', 'Special
Teams': 'green'}

# Plot the field and heatmap
plt.figure(figsize=(12, 6))
sns.kdeplot(x=df_sample['x'], y=df_sample['y'], cmap='Blues',
fill=True)

# Scatter plot for different player groups
for group, color in position_colors.items():
    subset = df_sample[df_sample['group'] == group]
    plt.scatter(subset['x'], subset['y'], c=color, label=group,
alpha=0.6, edgecolors='black')

# Set field dimensions
plt.xlim(0, 120)
plt.ylim(0, 53.3)

# Set axis labels and title
plt.xlabel("Field Length (yards)")
plt.ylabel("Field Width (yards)")
plt.title(f"NFL Player Positioning - Game {GAME_ID} (Sampled)")

# Set x-axis ticks at every 10-yard mark
plt.xticks(range(0, 130, 10))

# Add vertical field markings at every 10 yards
for x in range(0, 130, 10):
    plt.axvline(x, color='gray', linestyle='--', alpha=0.6)

# Add field section labels rotated along the Y-axis direction
plt.text(5, 25, "End Zone", fontsize=12, color='red',
fontweight='bold', ha='center', rotation=90)
plt.text(20, 25, "Red Zone", fontsize=12, color='black',
fontweight='bold', ha='center', rotation=90)
plt.text(60, 25, "Midfield", fontsize=12, color='black',
fontweight='bold', ha='center', rotation=90)
plt.text(100, 25, "Red Zone", fontsize=12, color='black',
fontweight='bold', ha='center', rotation=90)
plt.text(115, 25, "End Zone", fontsize=12, color='red',
fontweight='bold', ha='center', rotation=90)
```

```
# Add a legend
plt.legend(title="Player Group", loc="upper right")

# Show the plot
plt.show()
```





What is a play?

In the NFL, a **play** is a single down or action that begins with the snap of the ball and ends when the play is whistled dead by the officials. Each play is an opportunity for the offensive team to advance the ball toward the opponent's end zone or for the defense to stop them.

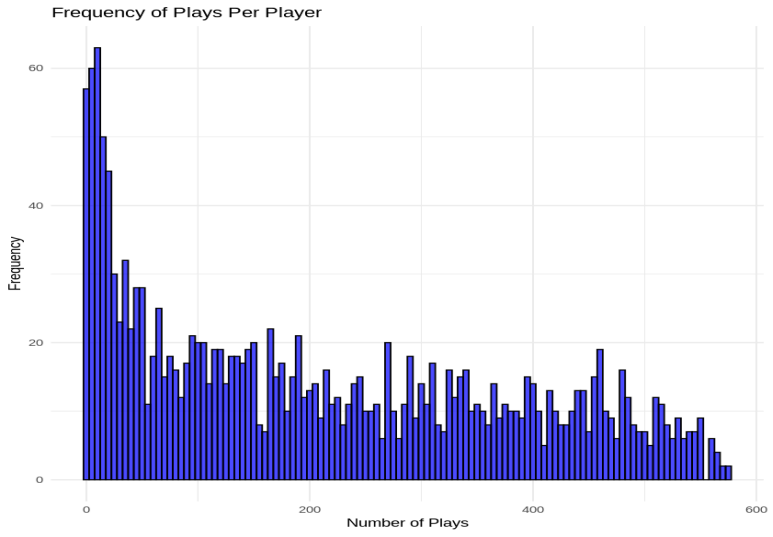
Types of Plays:

1. **Offensive Plays** – Designed to gain yardage or score points (e.g., passing, rushing, trick plays).
2. **Defensive Plays** – Aimed at stopping the offense, creating turnovers, or forcing punts.
3. **Special Teams Plays** – Include punts, field goals, kickoffs, and extra-point attempts.

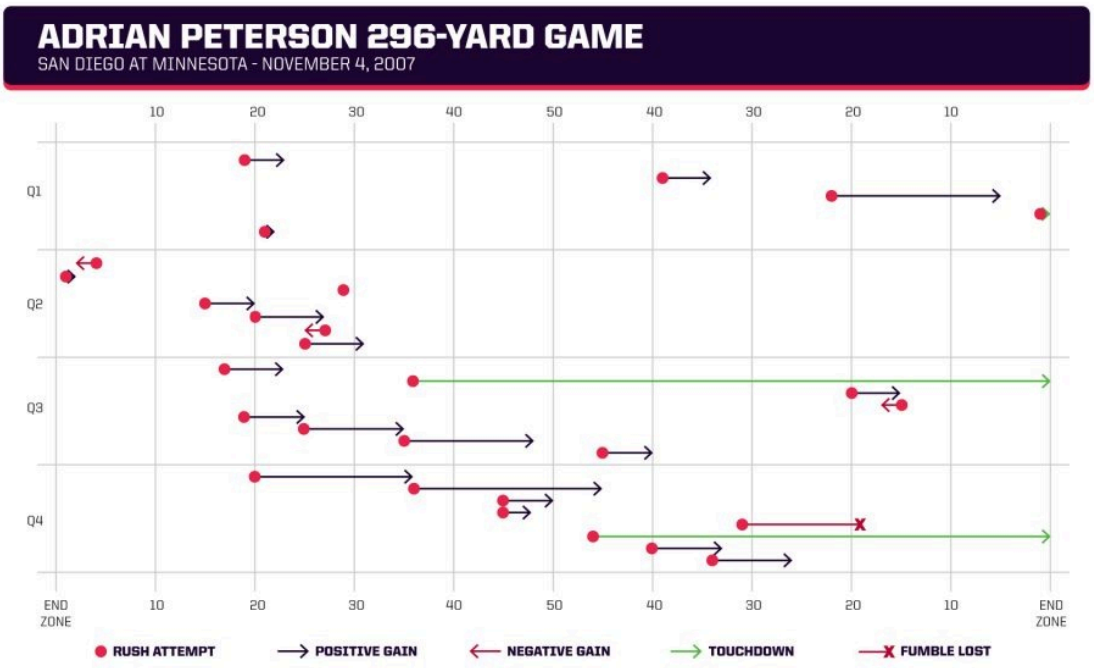
A play is only considered legal if it follows NFL rules, such as the correct formation, snap timing, and avoiding penalties like false starts or holding. A typical NFL game consists of hundreds of plays, each contributing to the overall strategy and outcome of the game.

Code :

```
play_count <- player_plays %>%
  group_by(nflId) %>%
  summarize(Play_Count = n())
# Plot frequency of plays per player
ggplot(play_count, aes(x = Play_Count)) +
  geom_histogram(binwidth = 5, fill = "blue", color = "black", alpha = 0.7) +
  theme_minimal() +
  labs(
    title = "Frequency of Plays Per Player",
    x = "Number of Plays",
    y = "Frequency"
  )
```



Rushing yards:



Source - Opta Analysts

Rushing yards refer to the total **number of yards a player gains by running with the ball** (rushing) during a play from scrimmage. This statistic is primarily used for running backs but also applies to quarterbacks and other players who carry the ball on designed runs or scrambles.

How Rushing Yards Are Calculated:

Rushing yards start from the line of Scrimmage where the play begins.The number of yards gained (or lost) before the ball carrier is tackled, steps out of bounds, or scores a touchdown is counted. Negative rushing yards occur if the ball carrier is tackled behind the line of scrimmage.

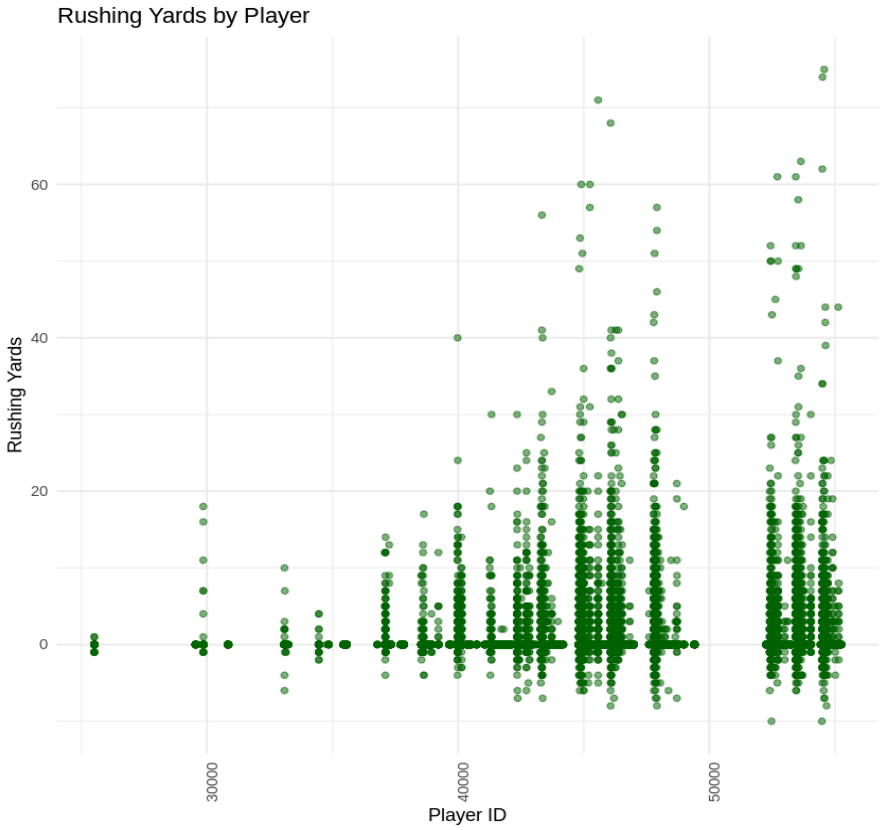
Why Rushing Yards Matter:

A strong rushing attack can control the clock and wear down defenses. Running backs with high rushing yards are key offensive weapons. A good rushing game forces defenses to adjust, opening up opportunities for passing plays. Some of the top all-time rushing leaders in the NFL include Emmitt Smith, Walter Payton, and Barry Sanders, who accumulated thousands of rushing yards throughout their careers.

In a dataset, we can calculate the dataset using the following code:

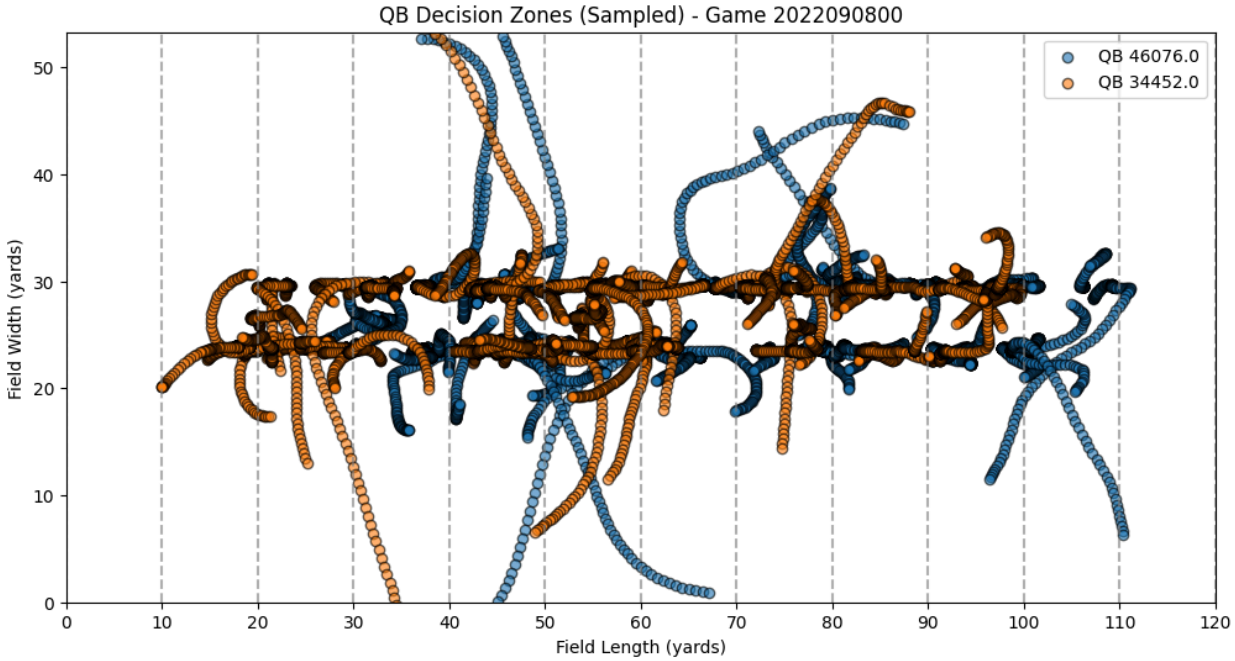
```
if("rushingYards" %in% colnames(player_plays) && "nflId" %in%
colnames(player_plays)) {
  ggplot(player_plays, aes(x = nflId, y = rushingYards)) +
    geom_point(color = "darkgreen", alpha = 0.5) +
    theme_minimal() +
    labs(
      title = "Rushing Yards by Player",
      x = "Player ID",
      y = "Rushing Yards"
    ) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))
} else {
  cat("Column 'rushingYards' or 'nflId' not found in player_plays dataset.\n")
}
```

Output:-



Tracks QB positions at pass attempts

```
# Define Game ID for analysis
GAME_ID = 2022090800  # Change this for different games

# Filter data for the selected game
df_game = df_tracking_week_1[df_tracking_week_1['gameId'] == GAME_ID]

# Merge with player positions
df_merged = df_game.merge(df_players[['nflId', 'position']],
on='nflId', how='left')

# Filter for QB positions at pass events
df_qb = df_merged[df_merged['position'] == 'QB']

# Randomly sample a few QBs if multiple exist
qb_sample = df_qb['nflId'].unique()
if len(qb_sample) > 3:  # Adjust the number of QBs as needed
    qb_sample = np.random.choice(qb_sample, size=3, replace=False)

df_qb_sampled = df_qb[df_qb['nflId'].isin(qb_sample)]

# Scatter plot of sampled QB decision zones
plt.figure(figsize=(12, 6))

for qb in qb_sample:
    qb_data = df_qb_sampled[df_qb_sampled['nflId'] == qb]
    plt.scatter(qb_data['x'], qb_data['y'], label=f'QB {qb}',
alpha=0.6, edgecolors='black')

plt.xlim(0, 120)
plt.ylim(0, 53.3)
plt.xlabel("Field Length (yards)")
plt.ylabel("Field Width (yards)")
plt.title(f"QB Decision Zones (Sampled) - Game {GAME_ID}")
plt.xticks(range(0, 130, 10))

# Add vertical field markings
for x in range(0, 130, 10):
    plt.axvline(x, color='gray', linestyle='--', alpha=0.6)

plt.legend()
plt.show()
```



We will track the positions of quarterbacks (QBs) during pass attempts in a specific NFL game. It filters tracking data for a given game ID, merges it with player position data to identify QBs. This helps us to analyze decision-making patterns and tendencies.

Average Speed and distance covered for Player

```
# Select tracking data for a specific player
player_tracking = df_tracking_week_1[df_tracking_week_1['nflId'] ==
42392]  # Example Player ID

# Compute average speed (yards per second)
player_avg_speed = player_tracking['s'].mean()

# Compute average distance covered per frame
player_avg_distance = player_tracking['dis'].mean()

# Display results
print(f"Average Speed and Distance Covered for Player 42392:
{player_avg_speed:.2f} yds/sec and {player_avg_distance:.2f}
yds/frame")
```

Average Speed and Distance Covered for Player 42392: 0.65 yds/sec and 0.06 yds/frame

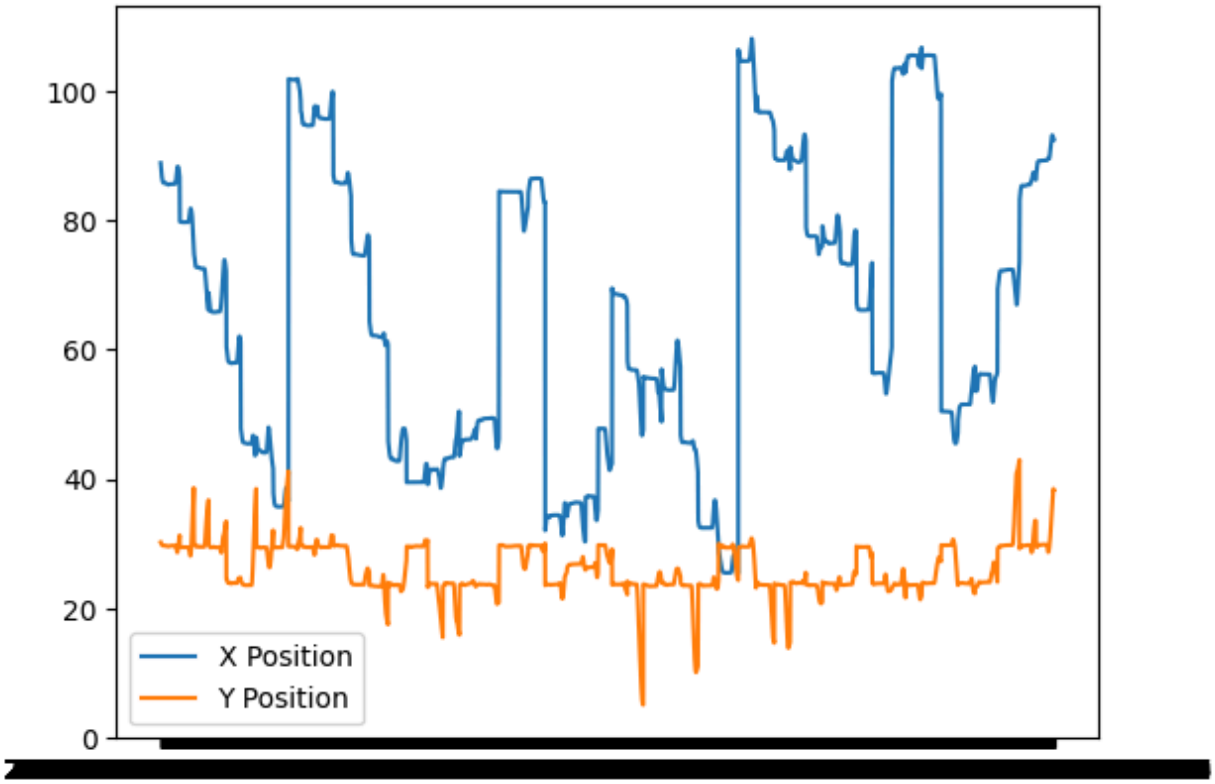
Time Series and Motion Tracking:

Code

```
player_tracking = df_tracking_week_1[df_tracking_week_1['nflId'] ==
42392]  # Example nflId
plt.plot(player_tracking['time'], player_tracking['x'], label='X
Position')
plt.plot(player_tracking['time'], player_tracking['y'], label='Y
Position')
plt.legend()
plt.show()
```

Explanation

This code plots the X and Y positions of a specific NFL player (with nflId = 42392) over time using tracking data from df_tracking_week_1. It shows the player's movement on the field during the tracked time period.



Thank You..