🏀 Python for Basketball Data Science: Comprehensive Student Guide (Weeks 1-10)

This guide provides a detailed walkthrough of the concepts, formulas, and required steps for each week of your Python for Basketball Data Science course.

Week 1: Python Basics & Foundational Metrics (Pace & Efficiency)

This week introduces Python fundamentals (variables and data types) and applies them to calculate two core modern basketball metrics: Possessions (Pace) and Offensive Efficiency.

🧠 Core Concepts

- **Data Types:** Python uses specific types to handle basketball stats.
  - **Integer (`int`):** For whole numbers like FGM, TO, or PTS (e.g., `shots = 85`).
  - **Float (`float`):** For numbers with decimal points, used for rates or factors (e.g., `free_throw_rate = 0.75`).
  - **String (`str`):** For text data like player names (e.g., `player = "Zach LaVine"`).
- **Formulas:**
  - Possessions: Estimates the number of offensive opportunities a team had.

    $$\text{Possessions} = \text{FGA} + (0.44 \times \text{FTA}) - \text{OREB} + \text{TO}$$

  - Offensive Efficiency: Measures points scored per 100 possessions.

    $$\text{Offensive Efficiency} = (\text{PTS} / \text{Possessions}) \times 100$$

📝 Assignment Walkthrough

Your goal is to use real or sample NBA box score data to calculate Possessions and Offensive Efficiency.

Step 1: Enter Game Stats (First Code Cell)

Define Python variables for the 5 required box score statistics using sample data (e.g., Celtics: FGA=90, FTA=25, OREB=12, TO=14, PTS=115).

Python
```
# Step 1: Enter game stats here
FGA = 90  # Field Goal Attempts
FTA = 25  # Free Throw Attempts
OREB = 12 # Offensive Rebounds
TO = 14   # Turnovers
PTS = 115 # Points Scored
```
Step 2: Calculate and Print Results (Second Code Cell)

Apply the formulas and use the print() function to display clear, rounded results. The 0.44 constant is a crucial float value in the possession formula.

Python
```
# Calculate possessions and offensive efficiency
possessions = FGA + (0.44 * FTA) - OREB + TO

# Calculate offensive efficiency (using a safety check to avoid division by zero)
off_efficiency = (PTS / possessions) * 100 if possessions > 0 else 0

# Print the results, rounding for a clean output
print("The team scored", PTS, "points on", round(possessions, 2), "possessions.")
print("Offensive Efficiency:", round(off_efficiency, 2))
# Expected Output: Offensive Efficiency: 123.66
```

Week 2: Control Flow & Functions (Advanced Stats)

This week introduces **functions** to encapsulate formulas and **control flow** (if/elif/else, for loops) for decision-making and repetition.

🧠 Core Concepts & Formulas

- **Functions:** Defined using def to perform a specific task and reuse code (e.g., calculating eFG%).
- **Conditionals:** Used for logic and branching.

  Python

  ```
  if points > 110: # Condition 1
      print("High scoring game")
  elif points >= 90: # Condition 2 (if Condition 1 is False)
      print("Average scoring game")
  else: # If all preceding conditions are False
      print("Low scoring game")
  ```

- **Loops:** Used to iterate over items in a list or range (e.g., summing player points).
- **Formulas for Advanced Metrics:**
    - Effective Field Goal % (eFG%): Adjusts FG% to credit 3-pointers for their extra point.

      $$\text{eFG\%} = \frac{\text{FGM} + 0.5 \times \text{threePM}}{\text{FGA}}$$

    - True Shooting % (TS%): The most comprehensive measure of scoring efficiency, including free throws.

      $$\text{TS\%} = \frac{\text{PTS}}{2 \times (\text{FGA} + 0.44 \times \text{FTA})}$$

📝 Assignment Walkthrough: Advanced Stats Calculator

Your task is to build a calculator using functions and an if/else statement.

Step 1: Define Functions (Code Cell 2)

Define the two required functions using the formulas above.

Python
```
# Define a function to calculate eFG%
def effective_fg(FGM, threePM, FGA):
    # Ensure FGA > 0 to prevent division by zero
    if FGA == 0:
        return 0.0
    return (FGM + 0.5 * threePM) / FGA

# Define a function to calculate TS%
def true_shooting(PTS, FGA, FTA):
    # Calculate True Shooting Attempts (TSA)
    TSA = 2 * (FGA + 0.44 * FTA)
    if TSA == 0:
        return 0.0
    return PTS / TSA
```

Step 2 & 3: Calculate and Use Control Flow (Code Cells 4 & 5)

Call your functions and use the result to drive an if/else message.

Python
```
# Example Stats (from Assignment Code Cell 4)
FGM = 10; FGA = 20; threePM = 3; FTA = 8; PTS = 29

# Calculate eFG%
efg = effective_fg(FGM, threePM, FGA)
```

```
print("Effective FG%:", round(efg, 3)) # Expected: 0.575

# Step 3: Efficiency Message
if efg > 0.6:
    print("Elite shooting night!")
elif efg >= 0.5: # 0.50 is generally considered average for eFG%
    print("Above average efficiency.") # This line executes for the example data
else:
    print("Below average efficiency.")
```

Extra Challenge: Multiple Players (Code Cell 6)

Use a **for loop** to calculate a metric (like TS%) and find the most efficient player among a list of players.

Python

```
# Assuming the 'players' list of dictionaries is defined...
players = [
    {"name": "Player A", "FGM": 10, "FGA": 20, "3PM": 3, "FTM": 6, "FTA": 8, "PTS": 29},
    # ... other players ...
]
most_efficient_player = None
highest_ts = -1

for player in players:
    # Calculate TS% for each player
    ts_percent = true_shooting(player["PTS"], player["FGA"], player["FTA"])
    player["TS%"] = round(ts_percent, 3)

    # Track the leader
    if ts_percent > highest_ts:
        highest_ts = ts_percent
        most_efficient_player = player["name"]

print("The most efficient player is:", most_efficient_player)
```

Week 3: Data Manipulation with Pandas (Workload)

This week introduces the **pandas** library for handling structured basketball data (DataFrames), focusing on filtering, grouping, and basic visualization.

🧠 Core Concepts & Methods

Concept	Purpose	Pandas Method	Example
DataFrame (DF)	The central data structure (like a	pd.DataFrame(...)	df.head()

Concept	Purpose	Pandas Method	Example
	spreadsheet).		
Filtering	Selecting a subset of rows based on a condition.	Boolean Indexing	games_df = df[df["Session_Type"] == "Game"]
Grouping/Aggregation	Summarizing data by category (e.g., finding the mean workload per player).	.groupby('COL')['STAT'].mean()	df.groupby("Player")["Workload_AU"].mean()

Assignment Walkthrough: Workload Analysis

The assignment requires you to analyze a synthetic Chicago Bulls performance dataset.

Task 1: Inspect the Dataset

Always start by checking the structure and identifying any missing data.

Python
```
print(df.info())    # Check data types and non-null counts
print(df.describe()) # Summary statistics of numerical columns
print(df.isnull().sum()) # Check for missing values in each column
```

Task 2: Data Filtering and Player Averages

Isolate "Game" sessions and find the average workload for each player.

Python
```
# 1. Filter for only "Game" sessions
games_df = df[df["Session_Type"] == "Game"]

# 2. Group by player and calculate the mean Workload_AU
avg_workload_games = games_df.groupby("Player")["Workload_AU"].mean()
print(avg_workload_games)
```

Task 3: Summary Statistics by Session Type

Group the *entire* dataset by Session_Type to see how metrics like Workload and RPE vary across different types of training/game activities.

Python
```
# Group by Session_Type and find the mean of Workload_AU and RPE
```

```
avg_by_session = df.groupby("Session_Type")[["Workload_AU", "RPE"]].mean().round(1)
print(avg_by_session)
```

Task 4: Visualization

Visualize the results from Task 2 (average game workload per player) using a bar chart.

Python
```
import matplotlib.pyplot as plt

plt.figure(figsize=(8,5))
avg_workload_games.sort_values(ascending=False).plot(kind='bar', color='crimson')
plt.title("Average Game Workload by Player")
plt.ylabel("Workload (AU)")
plt.xlabel("Player")
plt.show()
```

Week 4: Box Score Analysis & Data Export

This week focuses on using Pandas to perform quantitative analysis on a single box score, including calculating metrics, finding leaders, and saving the results.

🧠 Core Concepts & Methods

- **Team Aggregation:** Use the .mean() method to quickly find the average of all numerical columns.
  - df.mean(numeric_only=True)
- **Finding Leaders:** Combine two methods to find the index (row) of the maximum value (.idxmax()) and then retrieve a different column's value (the player's name) from that row (.loc[]).
  - df.loc[df["PTS"].idxmax(), "Player"]
- **Adding/Modifying Columns:** Create new metrics by operating directly on existing columns. This is a vector operation in Pandas/NumPy.
  - df["EFF"] = df["PTS"] + df["REB"] + df["AST"]
- **Sorting:** Rank your players by a new metric.
  - df.sort_values(by="EFF", ascending=False)
- **Exporting Data:** Save the final clean results to a CSV file for future use.
  - df.to_csv("file_name.csv", index=False)

📝 Assignment Walkthrough

Your task is to analyze a single game's box score and determine the most efficient players.

Step 1: Create Data (Code Cell 1)

The DataFrame with core box score stats (PTS, REB, AST, STL, BLK) is provided.

Step 2: Team Averages (Code Cell 2)

Calculate the mean for all numerical columns.

Python
```
# Calculate team averages, ignoring non-numeric columns like 'Player'
print("Team Average Stats:\n", df.mean(numeric_only=True))
# Expected Output (example): PTS 16.4, REB 7.0, AST 5.0
```
Step 3: Identify Leaders (Code Cell 3)

Find the player who leads in Points, Rebounds, and Assists using .idxmax() and .loc[].

Python
```
# Find the index of the highest PTS value, then retrieve the 'Player' name at that index
print("Top Scorer:", df.loc[df["PTS"].idxmax(), "Player"])
print("Best Rebounder:", df.loc[df["REB"].idxmax(), "Player"])
print("Top Assister:", df.loc[df["AST"].idxmax(), "Player"])
```
Step 4: Add Efficiency and Sort (Code Cell 4)

Calculate a simple Efficiency (EFF) score and use it to rank the players.

Python
```
# Calculate EFF: PTS + REB + AST + STL + BLK (using vector arithmetic)
df["EFF"] = df["PTS"] + df["REB"] + df["AST"] + df["STL"] + df["BLK"]

# Sort the DataFrame by the new EFF column
df_sorted = df.sort_values(by="EFF", ascending=False)
df_sorted
```
Step 5: Export Results (Code Cell 5)

Save the sorted data to a CSV file.

Python
```
# Save the results, ensuring index=False is set
df_sorted.to_csv("team_performance.csv", index=False)
print("Saved team_performance.csv")
```

Week 5: Data Visualization (Visual Report)

This week introduces **Matplotlib** and **Seaborn** to create visual reports of basketball data, emphasizing bar charts, scatter plots, and line charts.

🧠 Core Concepts & Methods

- **Import Libraries:** Always import matplotlib.pyplot as plt and seaborn as sns.
- **Bar Plot (sns.barplot):** Used to compare a single metric (e.g., Points) across categorical items (e.g., Players).
- **Scatter Plot (sns.scatterplot):** Used to visualize the relationship between two numerical variables (e.g., REB vs. AST).
- **Line Plot (plt.plot):** Used to show a trend or performance over a sequence (e.g., Points over 5 Games).
- **Aesthetics:** Always include titles, axis labels, and proper figure sizing.

 Assignment Walkthrough: Visual Report

You will visualize a simulated box score DataFrame (df).

Step 1: Create Data (Code Cell 2)

The df with Points, Rebounds, and Assists is provided.

Step 2: Bar Chart – Points per Player (Code Cell 3)

Compare scoring across players.

Python
```
plt.figure(figsize=(8,5))
# Barplot: x=Player (category), y=PTS (value)
sns.barplot(x="Player", y="PTS", data=df, palette="viridis")
plt.title("Player Scoring Comparison")
plt.ylabel("Points")
plt.xticks(rotation=15)
plt.show()
```
Step 3: Scatter Plot – Rebounds vs Assists (Code Cell 4)

Visualize the relationship between rebounding and passing impact.

Python
```
plt.figure(figsize=(6,5))
# Scatterplot: x=REB (value 1), y=AST (value 2)
sns.scatterplot(x="REB", y="AST", data=df, s=100, color="crimson")
plt.title("Rebounds vs Assists")
plt.xlabel("Rebounds")
plt.ylabel("Assists")
plt.show()
```
Step 4: Line Chart – Example Player Trend (Code Cell 6)

Visualize a player's performance trend across hypothetical games.

Python
```
games = ["G1", "G2", "G3", "G4", "G5"] # x-axis categories
points = [15, 20, 25, 18, 27]         # y-axis values
```

```
plt.figure(figsize=(7,4))
# Line plot: plots the points list against the games list
plt.plot(games, points, marker="o", color="blue")
plt.title("Player A – Points Over 5 Games")
plt.xlabel("Game")
plt.ylabel("Points")
plt.show()
```

Week 6: NumPy & Statistical Analysis

This week introduces the **NumPy** library for high-speed vector arithmetic and statistical functions, such as mean, standard deviation, and correlation.

🧠 Core Concepts & Methods

Concept	Purpose	NumPy Method	Example
NumPy Array	The fundamental high-speed data structure.	np.array([...])	points = np.array([27, 22, 15, 18, 8])
Descriptive Stats	Quick calculation of central tendency and spread.	np.mean(), np.median(), np.std()	np.mean(points)
Broadcasting	Performing element-wise calculations across entire arrays without a loop (e.g., custom efficiency formulas).	array * scalar + array2	efficiency = points * 0.5 + rebounds * 1.2
Correlation (r)	Measures the linear relationship between two variables ($r \in [-1, 1]$).	np.corrcoef(arr1, arr2)[0, 1]	np.corrcoef(points, rebounds)[0, 1]

📝 Assignment Walkthrough: Statistical Basketball Analysis

You will analyze a simple box score using NumPy functions.

Step 1: Create Your Dataset (Code Cell 2)

Define the player metrics as NumPy arrays.

Python
```
import numpy as np
players = np.array(["Player A", "Player B", "Player C", "Player D", "Player E"])
points = np.array([25, 19, 22, 15, 10])
rebounds = np.array([9, 6, 11, 8, 5])
assists = np.array([7, 5, 6, 4, 3])
```

Step 2: Calculate Key Metrics (Code Cell 3 & 4)

Calculate the team's central tendency (mean, median) and variability (standard deviation).

Python
```
# Descriptive Stats
avg_points = np.mean(points)
std_points = np.std(points)
median_rebounds = np.median(rebounds)

# Correlation (e.g., between Points and Rebounds)
corr_pr = np.corrcoef(points, rebounds)[0, 1]
print("Correlation (Points vs Rebounds):", round(corr_pr, 2)) # Expected approx. 0.73
```
Step 3: Compute Player Efficiency (Code Cell 5)

Use broadcasting to calculate a weighted efficiency score and put the results into a DataFrame.

Python
```
# Custom Efficiency Formula (example)
efficiency = points * 0.5 + rebounds * 1.2 + assists * 0.8

# Create a DataFrame for organized output
df = pd.DataFrame({
    "Player": players,
    "Points": points,
    "Rebounds": rebounds,
    "Assists": assists,
    "Efficiency": efficiency
})
df
```
Step 4: Visualize Your Findings (Code Cell 6)

Visualize the calculated efficiency scores.

Python
```
import matplotlib.pyplot as plt

plt.figure(figsize=(8,5))
plt.bar(df["Player"], df["Efficiency"], color="purple")
plt.title("Player Efficiency Comparison")
plt.ylabel("Efficiency Score")
plt.show()
```

Week 7: Data Cleaning & Preprocessing

This week covers the essential data cleaning steps: handling missing data, fixing data types, and ensuring column consistency.

🧠 Core Concepts & Methods

Data Cleaning Task	Goal	Pandas Method	Example
Missing Values	Locate where data is missing (NaN, None).	.isnull().sum() or .isna().sum()	df.isnull().sum()
Imputation	Fill missing numeric values intelligently.	.fillna(df['COL'].mean()) or .fillna(df['COL'].median())	df['PTS'].fillna(df['PTS'].mean(), inplace=True)
Type Conversion	Ensure numerical columns are correctly typed (e.g., Minutes should be integers).	.astype()	df["MIN"].astype(int)
Rename Columns	Standardize labels for clarity.	.rename(columns={...})	df.rename(columns={"PTS": "Points"}, inplace=True)

📝 Assignment Walkthrough: Cleaning and Preparing Data

You will clean a raw DataFrame containing missing values (simulated as NaN and None).

Step 1: Create Raw Dataset (Code Cell 2)

A dataset with missing values is generated.

Step 2: Check for Missing Values (Code Cell 3)

Verify the locations of missing data.

Python
```
print("Missing Values Summary:")
print(df.isnull().sum())
# Expected: PTS: 1, REB: 1, AST: 1, MIN: 1
```
Step 3: Fill Missing Data (Code Cell 4)

Apply imputation methods (mean/median) to fill the missing values in numerical columns.

Python
```
# Fill PTS and REB with the Mean, as it's typically fine for core stats
```

```
df["PTS"].fillna(df["PTS"].mean(), inplace=True)
df["REB"].fillna(df["REB"].mean(), inplace=True)

# Fill AST with the Median, as it's less sensitive to high-assist outliers
df["AST"].fillna(df["AST"].median(), inplace=True)

# Fill MIN with the Mean
df["MIN"].fillna(df["MIN"].mean(), inplace=True)
```

Step 4: Fix Data Types (Code Cell 5)

Convert the MIN column to an integer type.

Python
```
df["MIN"] = df["MIN"].astype(int)
```

Step 5: Rename Columns (Code Cell 6)

Rename the columns for better readability.

Python
```
df.rename(columns={"PTS": "Points", "REB": "Rebounds", "AST": "Assists"}, inplace=True)
```

Step 7: Export Clean Data (Code Cell 9)

Save your final clean and structured dataset.

Python
```
df.to_csv("cleaned_week7_assignment.csv", index=False)
print("✅ Cleaned dataset saved as 'cleaned_week7_assignment.csv'")
```

Week 8: Exploratory Data Analysis (EDA) Project

This week involves conducting a full EDA on a large basketball dataset, combining descriptive statistics, correlation, and visualization.

🧠 Core Concepts & Methods

- **EDA Purpose:** Quickly understand the data's shape, distribution, and relationships before modeling.
- **Correlation Review (Heatmap):** Use sns.heatmap to visualize the linear relationships between all pairs of metrics, looking for strong ties ($r \approx 1.0$).
- **Univariate Analysis (Histogram):** Use sns.histplot to examine the shape of a single variable's distribution.
- **Categorical Analysis (Box Plot):** Use sns.boxplot to compare statistical distributions across different positions (G, F, C).

📝 Assignment Walkthrough: EDA Project

You will use the large simulated clean_box_scores.csv file created in the setup cell.

Step 1: Summary Statistics (Code Cell 5)

Get a quantitative overview of the data's central tendency and spread.

Python
```
# Display summary statistics for all numerical columns
df.describe()
```
Step 2: Visualize Distributions (Code Cell 9)

Create a histogram to understand the shape of the Points distribution.

Python
```
sns.histplot(df['PTS'], kde=True) # KDE=True draws a kernel density estimate line
plt.show()
```
Step 3: Explore Correlations (Code Cell 15)

Generate a correlation heatmap to identify relationships between the core stats.

Python
```
# Calculate correlation matrix for the core stats and shooting percentages
sns.heatmap(df[['PTS','REB','AST','FG%','3P%','FT%']].corr(),
            annot=True, # Display correlation values on the map
            cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```
Step 4: Identify Performance Trends (Code Cell 17)

Calculate a **rolling average** (a contextual feature) to capture performance trends over time for individual players.

Python
```
# Group by 'Position' and transform the PTS column using a 5-game rolling mean
df['Rolling_PTS'] = df.groupby('Position')['PTS'].transform(lambda x: x.rolling(5, min_periods=1).mean())
```
Step 5: Write a Summary of Insights (Code Cell 18)

Summarize the statistical results and visual trends from your analysis.

- **Example Insight:** If PTS and $\text{FG\%}$ are highly correlated, scoring volume is strongly tied to shooting efficiency. If Centers (C) have a wider range (taller box) in Rebounds than Guards (G), it indicates greater variability in rebounding for Centers.

Week 9: Feature Engineering in Practice

This week applies advanced feature engineering techniques to prepare the basketball data for machine learning models.

🧠 Core Concepts & Methods

- **Feature Engineering:** Creating new variables from existing data to capture more specific performance aspects (e.g., efficiency, volume, form).
- **Formulas:** Calculated advanced metrics like **Efficiency**, **TS%**, and **USG%** (Usage Rate).
- **Rolling Averages:** Creating **Rolling_Efficiency** provides a temporal feature indicating recent player form.
- **Redundancy Check:** Use a correlation matrix (sns.heatmap) to check if new features are too closely related to old features, which can complicate modeling.

📝 Assignment Walkthrough: Feature Engineering Project

You will engineer and analyze several advanced features using the provided per-game box score data.

1. Engineer Features (Code Cell 4)

Create at least 5 new features.

Python
```
# 1. Player Efficiency (EFF)
df['Efficiency'] = ((df['PTS'] + df['REB'] + df['AST'] + df['STL'] + df['BLK']) - (df['FGA'] - df['FGM']) - (df['FTA'] - df['FTM']) - df['TO']) / df['GP']

# 2. True Shooting % (TS%)
df['TS%'] = df['PTS'] / (2 * (df['FGA'] + 0.44 * df['FTA']))

# 3. Usage Rate (USG%)
df['USG%'] = 100 * ((df['FGA'] + 0.44 * df['FTA'] + df['TO']) * df['MIN']) / df['Team_MIN']

# 4. Workload Ratio
df['Workload_Ratio'] = df['MIN'] / df['Team_MIN']

# 5. Rolling 5-Game Efficiency (Contextual Feature)
df['Rolling_Efficiency'] = df.groupby('Player')['Efficiency'].transform(lambda x: x.rolling(5, min_periods=1).mean())
```

2. Visualize New Features (Code Cell 5)

Visualize the relationship between **TS%** (efficiency) and **USG%** (volume) to separate high-efficiency/low-volume players from low-efficiency/high-volume players.

Python
```
# Scatterplot: TS% vs USG%
sns.scatterplot(data=df, x='USG%', y='TS%')
plt.title('True Shooting vs Usage Rate')
plt.show()
```
3. Correlation Analysis (Code Cell 7)

Check the full correlation matrix for redundancy between the original box score metrics and your new features.

Python
```
# Check for redundancy among all numerical features
plt.figure(figsize=(10,6))
sns.heatmap(df.select_dtypes(include=np.number).corr(), cmap='coolwarm', annot=False)
plt.title('Feature Correlation Matrix')
plt.show()
```
4. Interpretation (Code Cell 8)

Explain your findings, focusing on which new metrics (like TS% and USG%) capture unique and predictive aspects of player impact that simple box score stats miss.

Week 10: Machine Learning for Injury Risk Prediction

This week applies machine learning models, specifically **Logistic Regression** and **Random Forest**, to a sports science problem: predicting injury risk based on player workload and recovery data.

🧠 Core Concepts & Methods

ML Concept	Purpose	Sklearn Method
Split Data	Separates data into **training** (70-80%) and **testing** (20-30%) sets for proper model evaluation.	train_test_split()
Scaling	Normalizes features so they have a mean of 0 and Std Dev of 1 (essential for Logistic Regression).	StandardScaler()

ML Concept	Purpose	Sklearn Method
Logistic Regression	A simple linear model for binary classification.	LogisticRegression()
Random Forest	An ensemble (tree-based) model used for prediction and determining feature importance.	RandomForestClassifier()
Evaluation	Metrics to assess model performance (Accuracy, Precision, Recall, F1-Score).	classification_report()
Feature Importance	Determines which input variables were most influential in the model's predictions.	RandomForestClassifier.feature_importances_

Assignment Walkthrough: Injury Risk Prediction

The core challenge is dealing with **class imbalance** (very few injury cases), which the provided assignment code addresses by generating data with a low injury rate.

1. Load, Define, and Split Data (Code Cells 3 & 4)

Load the generated bulls_injury_data.csv, define your $\mathbf{X}$ (features) and $\mathbf{y}$ (target: InjuryNextWeek), and split the data.

Python
```
# Define Features (X) and Target (y)
X = df[['Minutes_Load', 'HighSpeedRuns', 'JumpLoad', 'HeartRate', 'SleepHours', 'PreviousInjury']]
y = df['InjuryNextWeek']

# Split the data (80% Train, 20% Test)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

2. Scale Features (Code Cell 5)

Scale the training and testing features.

Python
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

3. Train and Evaluate Logistic Regression (Code Cells 6 & 8)

Train the linear model using the scaled data.

Python
```
log_reg = LogisticRegression()
log_reg.fit(X_train_scaled, y_train)
```

```
y_pred_log = log_reg.predict(X_test_scaled)
print(classification_report(y_test, y_pred_log))
```

4. Train and Evaluate Random Forest (Code Cell 9)

Train the ensemble model (Note: Random Forest often performs well **without** explicit scaling, so the code uses the original `X_train` and `X_test`).

Python

```
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
print(classification_report(y_test, y_pred_rf))
```

5. Visualize Feature Importance (Code Cell 10)

Determine which factors the Random Forest model prioritized in its predictions.

Python

```
feat_importance = pd.Series(rf.feature_importances_, index=X.columns)
plt.figure(figsize=(8,5))
feat_importance.sort_values().plot(kind='barh', color='red')
plt.title('Feature Importance - Injury Risk Prediction (Bulls)')
plt.show()
```

6. Summary Interpretation (Code Cell 12)

Address the challenge of the 100% accuracy (due to imbalanced data) and interpret the **Feature Importance** plot to explain the real risk factors for player injuries.

- **Focus on:** Why were `PreviousInjury` and `SleepHours` often the most important features in the prediction?