iHoop Insights - Injury Preditction

```
# Importing Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score,
confusion_matrix

# Loading data
sessions_df = pd.read_csv('D:/Education/Syracuse
University/Projects/iHoop
Insights/injury_history(player_sessions).csv', encoding='ISO-8859-1')
muscle_df = pd.read_csv('D:/Education/Syracuse
University/Projects/iHoop
Insights/injury_history(muscle_imbalance_data).csv', encoding='ISO-
8859-1')
injury_df = pd.read_csv('D:/Education/Syracuse
University/Projects/iHoop
Insights/injury_history(injury_history).csv', encoding='ISO-8859-1')
```

In the code, we define a function `clean_column_names` that standardizes column names by replacing non-alphanumeric characters with underscores and stripping leading/trailing underscores. This is done to improve data consistency and ensure column names are compatible with further processing or analysis. The function is then applied to three DataFrames (`sessions_df`, `muscle_df`, `injury_df`) to clean their column names.

```
# Function to clean column names
def clean_column_names(df):
    df.columns = df.columns.str.replace('[^A-Za-z0-9]+', '_',
regex=True).str.strip('_')
    return df

# Apply the function to each DataFrame
sessions_df = clean_column_names(sessions_df)
muscle_df = clean_column_names(muscle_df)
injury_df = clean_column_names(injury_df)

# Checking sessions_df
sessions_df.head(5)
```

```
            Name  Player_ID  Group_Id Group_name  League_ID
Session_ID  \
0  Anthony Lopez        112       212    Group 1        301
```

```
1001
1  Anthony Lopez        112        212    Group 1        301
1002
2  Anthony Lopez        112        212    Group 1        301
1003
3  Anthony Lopez        112        212    Group 1        301
1004
4  Anthony Lopez        112        212    Group 1        301
1005

  Session_Date Position  Distance_mi  Distance_min_mi  ...  \
0     1/1/2023   Center         4.58             0.12  ...
1     1/3/2023   Center         1.18             0.11  ...
2     1/4/2023   Center         5.59             0.14  ...
3     1/6/2023   Center         3.22             0.09  ...
4     1/7/2023   Center         2.19             0.10  ...

   Heart_Rate_min_bpm  Heart_Rate_max_bpm  Human_Core_Temperature_F  \
0                  74                 198                     99.47
1                  62                 179                     99.56
2                  78                 172                    100.06
3                  64                 186                    100.45
4                  62                 146                     98.73

   Human_Core_Temperature_max_F  TRIMP  Heart_Rate_Recoveries  \
0                        101.24    261                      5
1                         99.33    270                      6
2                        102.31    149                      4
3                        101.10    180                     10
4                        100.91    152                      4

   Jump_Height_max_ft  Changes_of_Orientation  Exertions  Disk_Usage
0                2.31                     229        307       58.56
1                2.44                     427        180       44.93
2                3.04                     383        440       15.32
3                3.17                     462        450       21.46
4                1.28                     118        416       20.51

[5 rows x 30 columns]
```

```python
# Checking muscle_df
muscle_df.head(5)
```

```
   Player_ID  Session_ID    Player_Name Date_Recorded  \
0        112         101  Anthony Lopez      1/1/2023
1        112         102  Anthony Lopez      2/1/2023
2        112         103  Anthony Lopez      3/1/2023
3        112         104  Anthony Lopez      4/1/2023
4        112         105  Anthony Lopez      5/1/2023
```

```
   Hamstring_To_Quad_Ratio  Quad_Imbalance_Percent  \
0                 0.808741              -10.149294
1                 0.814355              -10.105784
2                 0.887331              -10.027546
3                 0.929176              -10.137407
4                 0.866234               -9.958386

   HamstringImbalance_Percent  Calf_Imbalance_Percent
Groin_Imbalance_Percent
0                   -8.208145              -10.176416                -
10.258755
1                   -8.229693              -10.106144                -
10.063777
2                   -8.897757              -10.257486                 -
9.990676
3                   -9.419432              -10.220899                -
10.179258
4                   -8.626291              -10.412659                -
10.208611
```

```
# Checking injury_df
injury_df.head(5)
```

```
   Player_ID             Name  Group_Id    Injury_Type   Body_Part
Side  \
0        101  Jordan Matthews       201  Muscle Strain  Quadriceps
Right
1        101  Jordan Matthews       201     Tendonitis       Wrist
Left
2        101  Jordan Matthews       201     Tendonitis    Shoulder
Right
3        103   Malik Robinson       203         Strain       Groin
Right
4        103   Malik Robinson       203       Fracture       Wrist
Left

  Injury_Date Severity  Recovery_Time_days  \
0   12/5/2023  Grade 2                  51
1  10/25/2023      NaN                  11
2   7/22/2023      NaN                  12
3   6/28/2023  Grade 1                  20
4   2/14/2023      NaN                  68

                                    Additional_Notes
0  Grade 2 quadriceps strain with partial tearing...
1  De Quervain's tenosynovitis. Swelling and pain...
2  Rotator cuff tendonitis due to overuse. Anti-i...
3  Grade 1 groin strain, characterized by mild ov...
4  Distal radius fracture. Cast applied. Recovery...
```

Joining all the 3 data frames into 1 data frame

In this code, we first convert the 'Session_Date' and 'Date_Recorded' columns to datetime format in both `sessions_df` and `muscle_df`. Then, we create a `join_key` by combining the `Player_ID`, month, and year of the respective date columns. This key is used to merge both DataFrames on the `join_key` column using a left join, appending the suffix '_muscle' to columns from `muscle_df`. Finally, unnecessary columns are dropped from the merged DataFrame to clean the result.

```
sessions_df['Session_Date'] =
pd.to_datetime(sessions_df['Session_Date'])
sessions_df['join_key'] = (sessions_df['Player_ID'].astype(str) + '_'
+

sessions_df['Session_Date'].dt.month.astype(str) + '_' +

sessions_df['Session_Date'].dt.year.astype(str))

muscle_df['Date_Recorded'] =
pd.to_datetime(muscle_df['Date_Recorded'])
muscle_df['join_key'] = (muscle_df['Player_ID'].astype(str) + '_' +

muscle_df['Date_Recorded'].dt.month.astype(str) + '_' +

muscle_df['Date_Recorded'].dt.year.astype(str))


sessions_muscle_merged = pd.merge(
    sessions_df,
    muscle_df,
    on='join_key',
    how='left',
    suffixes=('', '_muscle')
)

columns_to_drop = ['Player_ID_muscle',
'Player_Name','Session_ID_muscle']
sessions_muscle_merged =
sessions_muscle_merged.drop(columns=columns_to_drop)
```

In this code, we convert the 'Injury_Date' column in `injury_df` to datetime format and create a `join_key` by combining the `Player_ID`, month, and year of the injury date. This key is then used to merge `sessions_muscle_merged` and `injury_df` on the `join_key` column using a left join, appending the suffix '_injury' to the columns from `injury_df`. Finally, unnecessary columns are dropped from the merged DataFrame to clean the result.

```
injury_df['Injury_Date'] = pd.to_datetime(injury_df['Injury_Date'])
injury_df['join_key'] = (injury_df['Player_ID'].astype(str) + '_' +
                        injury_df['Injury_Date'].dt.month.astype(str)
```

```
+ '_' +
                        injury_df['Injury_Date'].dt.year.astype(str))

final_merged_df = pd.merge(
    sessions_muscle_merged,
    injury_df,
    on=['join_key'],
    how='left',
    suffixes=('', '_injury')
)

columns_to_drop = ['Player_ID_injury', 'Name_injury',
'Group_Id_injury']
final_merged_df = final_merged_df.drop(columns=columns_to_drop)

final_merged_df.head(5)
```

```
            Name  Player_ID  Group_Id Group_name  League_ID
Session_ID  \
0  Anthony Lopez        112       212    Group 1        301
1001
1  Anthony Lopez        112       212    Group 1        301
1002
2  Anthony Lopez        112       212    Group 1        301
1003
3  Anthony Lopez        112       212    Group 1        301
1004
4  Anthony Lopez        112       212    Group 1        301
1005

  Session_Date Position  Distance_mi  Distance_min_mi  ...  \
0   2023-01-01   Center         4.58             0.12  ...
1   2023-01-03   Center         1.18             0.11  ...
2   2023-01-04   Center         5.59             0.14  ...
3   2023-01-06   Center         3.22             0.09  ...
4   2023-01-07   Center         2.19             0.10  ...

   HamstringImbalance_Percent  Calf_Imbalance_Percent  \
0                   -8.208145              -10.176416
1                   -8.208145              -10.176416
2                   -8.208145              -10.176416
3                   -8.208145              -10.176416
4                   -8.208145              -10.176416

   Groin_Imbalance_Percent  Injury_Type  Body_Part  Side  Injury_Date
\
0               -10.258755       Strain       Knee  Left   2023-01-26

1               -10.258755       Strain       Knee  Left   2023-01-26
```

```
2                 -10.258755       Strain       Knee  Left   2023-01-26

3                 -10.258755       Strain       Knee  Left   2023-01-26

4                 -10.258755       Strain       Knee  Left   2023-01-26


   Severity  Recovery_Time_days  \
0   Grade 1                28.0
1   Grade 1                28.0
2   Grade 1                28.0
3   Grade 1                28.0
4   Grade 1                28.0

                                    Additional_Notes
0  Strain of the posterior cruciate ligament (PCL...
1  Strain of the posterior cruciate ligament (PCL...
2  Strain of the posterior cruciate ligament (PCL...
3  Strain of the posterior cruciate ligament (PCL...
4  Strain of the posterior cruciate ligament (PCL...

[5 rows x 44 columns]
```

```
final_merged_df.dtypes
```

```
Name                                                object
Player_ID                                            int64
Group_Id                                             int64
Group_name                                          object
League_ID                                            int64
Session_ID                                           int64
Session_Date                                datetime64[ns]
Position                                            object
Distance_mi                                        float64
Distance_min_mi                                    float64
Duration_s                                           int64
Steps                                                int64
Speed_of_max                                       float64
Speed_max_mph                                      float64
Speed_mph                                          float64
Time_s                                               int64
Accumulated_Acceleration_Load                        int64
Anaerobic_Activity_distance_mi                     float64
Jump_Load_J                                          int64
Heart_Rate_bpm                                       int64
Heart_Rate_min_bpm                                   int64
Heart_Rate_max_bpm                                   int64
Human_Core_Temperature_F                           float64
Human_Core_Temperature_max_F                       float64
TRIMP                                                int64
```

```
Heart_Rate_Recoveries                                int64
Jump_Height_max_ft                                 float64
Changes_of_Orientation                               int64
Exertions                                            int64
Disk_Usage                                         float64
join_key                                            object
Date_Recorded                               datetime64[ns]
Hamstring_To_Quad_Ratio                            float64
Quad_Imbalance_Percent                             float64
HamstringImbalance_Percent                         float64
Calf_Imbalance_Percent                             float64
Groin_Imbalance_Percent                            float64
Injury_Type                                         object
Body_Part                                           object
Side                                                object
Injury_Date                                 datetime64[ns]
Severity                                            object
Recovery_Time_days                                 float64
Additional_Notes                                    object
dtype: object
```

Exploratory Data Analysis

```
# Basic dataset information
print("Dataset Overview:")
print(f"Total number of records: {len(final_merged_df)}")
print(f"Number of unique players: 
{final_merged_df['Player_ID'].nunique()}")
print(f"Date range: from {final_merged_df['Session_Date'].min()} to 
{final_merged_df['Session_Date'].max()}")
```

```
Dataset Overview:
Total number of records: 2604
Number of unique players: 14
Date range: from 2023-01-01 00:00:00 to 2023-12-30 00:00:00
```

In this code, we use the `isnull()` method to identify missing values in the `final_merged_df` DataFrame. The `sum()` function then counts the number of null values for each column. We display only the columns with null values by filtering the result where the count is greater than 0. This helps identify any missing data that may need to be addressed before further analysis.

```
# Checking for null values in the DataFrame
null_values = final_merged_df.isnull().sum()

# Displaying columns with null values
print("\nNull Values in DataFrame:")
print(null_values[null_values > 0])
```

```
Null Values in DataFrame:
Injury_Type            2378
Body_Part              2378
Side                   2438
Injury_Date            2378
Severity               2480
Recovery_Time_days     2378
Additional_Notes       2378
dtype: int64
```

In this code, we handled the null values in the `final_merged_df` DataFrame by filling them with appropriate values:

- For `Injury_Type`, `Body_Part`, `Side`, `Severity`, and `Additional_Notes`, we filled missing values with placeholders like `'No Injury'`, `'Unknown'`, or `'No Notes'` to ensure that these columns are not left blank.
- For the `Injury_Date` column, we used `pd.NaT` (Not a Time) to represent missing date values.
- For `Recovery_Time_days`, we filled the missing values with 0

```
# Filling null values with appropriate values
final_merged_df['Injury_Type'].fillna('No Injury', inplace=True)
final_merged_df['Body_Part'].fillna('Unknown', inplace=True)
final_merged_df['Side'].fillna('Unknown', inplace=True)
final_merged_df['Injury_Date'].fillna(pd.NaT, inplace=True)  # Filled
with NaT for date columns
final_merged_df['Severity'].fillna('Unknown', inplace=True)
final_merged_df['Recovery_Time_days'].fillna(0, inplace=True)  #
Filled with 0 as default recovery time
final_merged_df['Additional_Notes'].fillna('No Notes', inplace=True)
```

In this code, the `had_injury` column is created by applying a lambda function that checks if the `Injury_Type` is not equal to `'No Injury'`. If it's not `'No Injury'`, it assigns a value of 1, indicating that the player has an injury; otherwise, it assigns 0.

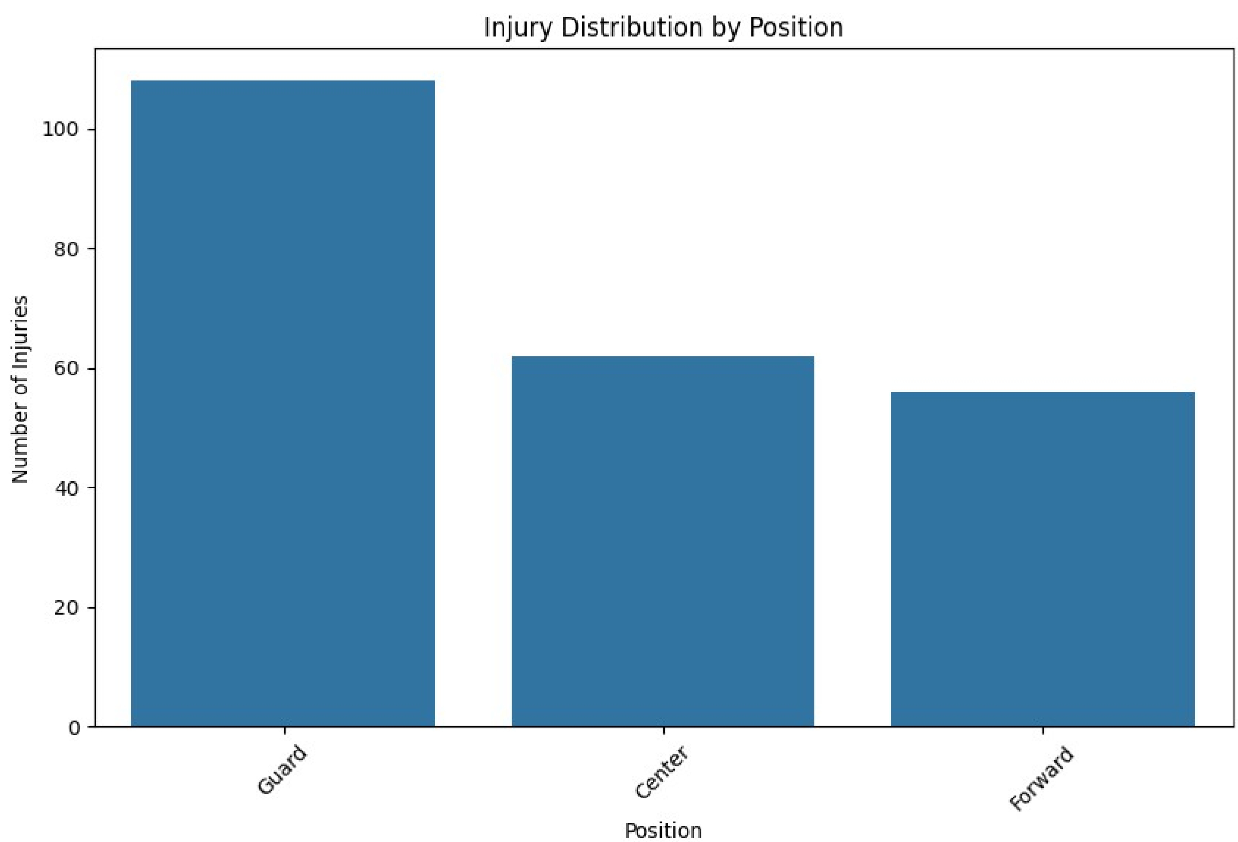
```
# Creating injury indicator with correct logic
final_merged_df['had_injury'] =
final_merged_df['Injury_Type'].apply(lambda x: 1 if x != 'No Injury'
else 0)

# Injury Statistics
print("\nInjury Statistics:")
total_injuries = final_merged_df['had_injury'].sum()
injury_rate = (total_injuries / len(final_merged_df)) * 100
print(f"Total number of injuries: {total_injuries}")
print(f"Injury rate: {injury_rate:.2f}%")
```

```
Injury Statistics:
Total number of injuries: 226
Injury rate: 8.68%
```

Injury Distribution by Position

The analysis of player injuries involved filtering the `final_merged_df` DataFrame to identify instances where `had_injury` equals 1, followed by calculating injury frequencies across different player positions using `.value_counts()`. The data was then visualized through a bar plot created with `sns.barplot()`, featuring position-specific injury counts.

```
# Plot injury distribution by position
plt.figure(figsize=(10, 6))
injury_by_position = final_merged_df[final_merged_df['had_injury'] ==
1]['Position'].value_counts()
sns.barplot(x=injury_by_position.index, y=injury_by_position.values)
plt.title('Injury Distribution by Position')
plt.xticks(rotation=45)
plt.ylabel('Number of Injuries')
plt.show()
```



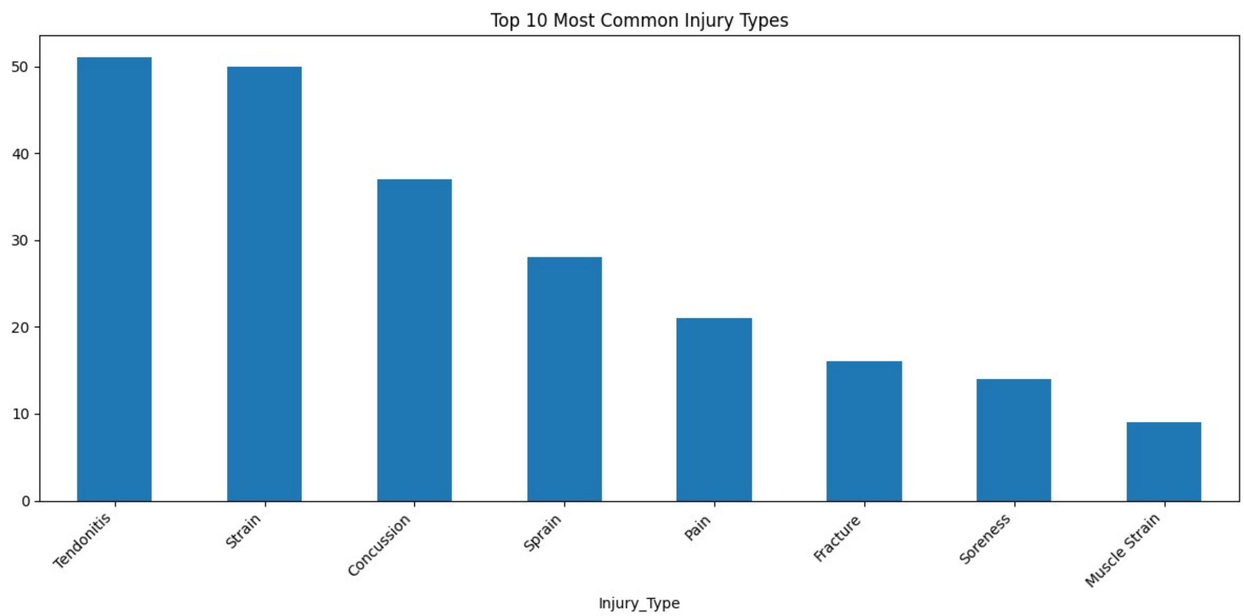
Visualization Insights:

- The bar chart shows the distribution of injuries across different player positions: Guard, Center, and Forward.
- `Guard` positions have the highest number of injuries, followed by `Center`, and `Forward` with the least.

Top 10 Most Common Injury Types

The code processes injury type data by calculating frequency distributions using `.value_counts()`, displaying the top 10 injuries through a bar chart with `plt.figure(figsize=(12, 6))`. The visualization includes rotated x-axis labels (45 degrees), proper title, and adjusted layout using `plt.tight_layout()` for optimal display.

```
# Plotting the top 10 most common injury types
plt.figure(figsize=(12, 6))
final_merged_df[final_merged_df['Injury_Type'] != 'No Injury']
['Injury_Type'] \
    .value_counts().head(10).plot(kind='bar')
plt.title('Top 10 Most Common Injury Types')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



Visualization Insights:

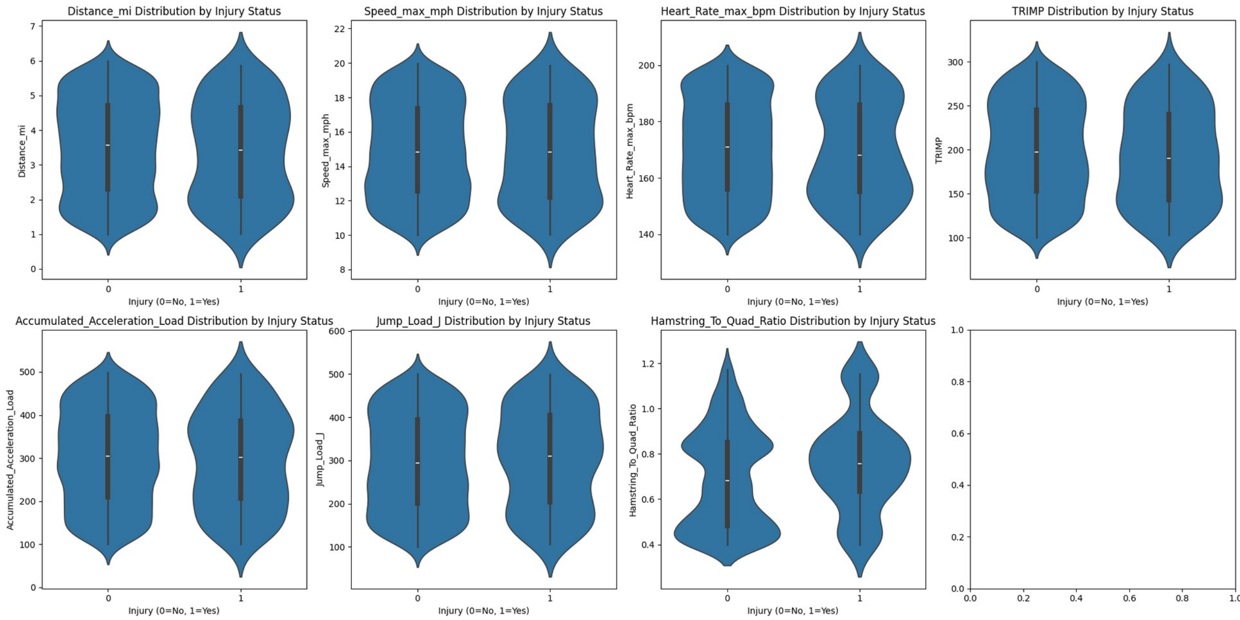
- The bar chart shows the top 10 most common injury types, excluding the `'No Injury'` category.
- The most common injuries are `Tendonitis` and `Strain`, followed by `Concussion`, `Sprain`, and others.

- This visualization helps to identify the most frequent injury types, providing insights for injury prevention and focused player care strategies.

Key metrics distribution for injured vs non-injured players

We are analyzing the distribution of key performance metrics for injured versus non-injured players. It defines a list of key metrics to analyze, which includes `Distance_mi`, `Speed_max_mph`, `Heart_Rate_max_bpm`, `TRIMP`, `Accumulated_Acceleration_Load`, `Jump_Load_J`, and `Hamstring_To_Quad_Ratio`. For each of these metrics, the code creates a violin plot that compares the distribution between players who have an injury (`had_injury` equals 1) and players without an injury (`had_injury` equals 0). Each plot is labeled with the metric name and the injury status.

```
# Analyzing key metrics distribution for injured vs non-injured
players
metrics_to_analyze = [
    'Distance_mi', 'Speed_max_mph', 'Heart_Rate_max_bpm',
    'TRIMP', 'Accumulated_Acceleration_Load', 'Jump_Load_J',
    'Hamstring_To_Quad_Ratio'
]

# Creating violin plots for key metrics
fig, axes = plt.subplots(2, 4, figsize=(20, 10))
axes = axes.ravel()

for idx, metric in enumerate(metrics_to_analyze):
    if metric in final_merged_df.columns:
        sns.violinplot(data=final_merged_df, x='had_injury', y=metric,
ax=axes[idx])
        axes[idx].set_title(f'{metric} Distribution by Injury Status')
        axes[idx].set_xlabel('Injury (0=No, 1=Yes)')

plt.tight_layout()
plt.show()
```



Visualization Insights:

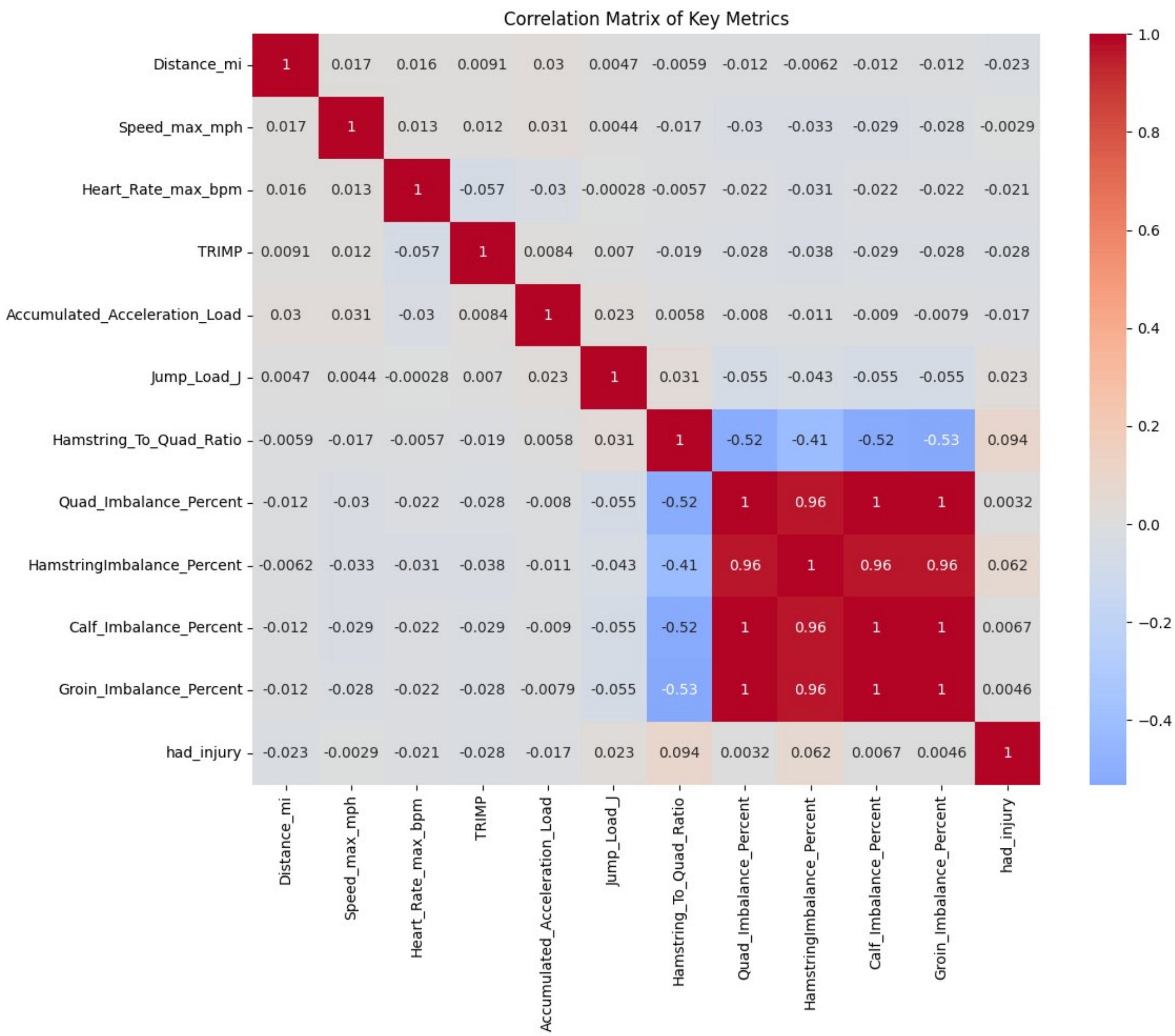
- **Distance_mi**: The distribution of distance covered by injured and non-injured players appears to have some overlap, but non-injured players tend to have a wider range of values.
- **Speed_max_mph**: There is a clear separation between injured and non-injured players, with non-injured players generally achieving higher speeds.
- **Heart_Rate_max_bpm**: The distribution of maximum heart rate shows a broader range for non-injured players, whereas injured players seem to have a more concentrated range.
- **TRIMP (Training Impulse)**: Non-injured players show a higher variance in TRIMP values compared to injured players, indicating a higher intensity or variation in their training sessions.
- **Accumulated_Acceleration_Load**: Injured players tend to have a higher accumulated acceleration load, which could be related to the physical strain they experience during play.
- **Jump_Load_J**: Injured players have a more concentrated distribution of jump load, possibly suggesting they are more prone to injury due to higher exertion in jumps.
- **Hamstring_To_Quad_Ratio**: The distribution shows that the hamstring-to-quad ratio is generally higher for non-injured players, which could suggest a more balanced muscle strength ratio that may protect them from injury.

These insights can be used to understand the relationship between performance metrics and the likelihood of injury, aiding in injury prevention and performance optimization strategies.

Correlation Analysis

The code performs a correlation analysis to examine the relationships between several key metrics, including performance-related metrics and imbalance percentages for different muscle groups. It first combines the performance metrics (`metrics_to_analyze`) with the imbalance percentages (`Quad_Imbalance_Percent`, `HamstringImbalance_Percent`, `Calf_Imbalance_Percent`, `Groin_Imbalance_Percent`) into a new DataFrame `correlation_df`, including the `had_injury` column to check its correlation with other metrics. A correlation matrix is then computed using `.corr()`, and a heatmap is plotted using `sns.heatmap()`. The color map `'coolwarm'` is used to visually display the correlation, with annotations indicating the strength of the relationships.

```
# Correlation analysis
correlation_metrics = metrics_to_analyze + [
    'Quad_Imbalance_Percent', 'HamstringImbalance_Percent',
    'Calf_Imbalance_Percent', 'Groin_Imbalance_Percent'
]

correlation_df = final_merged_df[correlation_metrics].copy()
correlation_df['had_injury'] = final_merged_df['had_injury']

# Plot correlation heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_df.corr(), annot=True, cmap='coolwarm',
center=0)
plt.title('Correlation Matrix of Key Metrics')
plt.tight_layout()
plt.show()
```



Visualization Insights:

- **Strong Positive Correlations**:
  - The muscle imbalance percentages (`Quad_Imbalance_Percent`, `HamstringImbalance_Percent`, `Calf_Imbalance_Percent`, `Groin_Imbalance_Percent`) are highly positively correlated with each other, showing a strong relationship (close to 1.0).
  - This suggests that players with higher imbalances in one muscle group tend to have higher imbalances in others.
- **Injury-Related Insights**:
  - `had_injury` shows a moderate negative correlation with performance metrics such as `Speed_max_mph` and `Heart_Rate_max_bpm`, suggesting that injured players may exhibit lower performance in these areas.
  - There is a weak correlation between `had_injury` and the other performance-related metrics like `Distance_mi`, `TRIMP`, and `Jump_Load_J`, suggesting that these metrics do not strongly predict injury status.

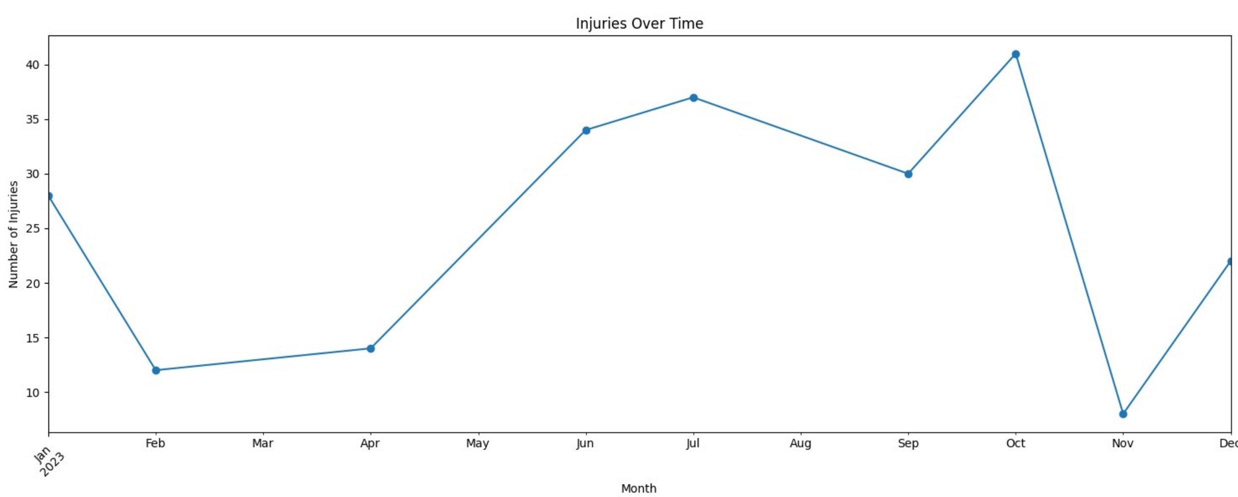
- **Imbalance-Performance Relationship**:
  - The imbalance metrics (`Quad_Imbalance_Percent`, `HamstringImbalance_Percent`, etc.) show some moderate correlations with performance metrics like `Distance_mi` and `Speed_max_mph`, which could indicate that muscle imbalances may affect overall performance.

This correlation matrix helps identify the interdependencies between different performance and injury-related metrics, assisting in understanding the factors that might influence player injuries.

Injuries Over Time

The code performs a temporal analysis to visualize how injuries are distributed over time. It filters the `final_merged_df` DataFrame to include only the rows where `had_injury` is 1 (indicating an injury). Then, it groups the data by month using `dt.to_period('M')` and counts the number of injuries for each month with `.size()`. The result is plotted as a line graph with markers to indicate each month's injury count. The x-axis is labeled with months, the y-axis shows the number of injuries.

```
# Temporal analysis
plt.figure(figsize=(15, 6))
injury_over_time = final_merged_df[final_merged_df['had_injury'] ==
1].groupby(
    final_merged_df['Session_Date'].dt.to_period('M')
).size()
injury_over_time.plot(kind='line', marker='o')
plt.title('Injuries Over Time')
plt.xlabel('Month')
plt.ylabel('Number of Injuries')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Visualization Insights:

- The line chart shows the trend of injuries over time, with fluctuations observed throughout the year.
- The number of injuries seems to peak during mid-year, with sharp drops in between, particularly in the months of January and December.
- The irregular distribution of injuries could reflect seasonal factors, training cycles, or other external influences, such as player fatigue or game schedules, that might contribute to injury spikes in certain months.
- This type of analysis can help in identifying periods when injury prevention strategies might be particularly necessary.

Statistical significance testing

The code performs statistical significance testing using the t-test to compare key metrics between injured and non-injured players. For each metric in the `metrics_to_analyze` list, the data is split into two groups: one for players with injuries (`had_injury` == 1) and one for players without injuries (`had_injury` == 0). The t-test is applied to the two groups using `scipy.stats.ttest_ind()`, and the p-value is calculated for each metric. The p-values are then printed for each metric to assess whether there is a statistically significant difference between injured and non-injured players.

```
# Statistical significance testing
from scipy import stats
print("\nStatistical Tests (Injured vs Non-injured):")
for metric in metrics_to_analyze:
    if metric in final_merged_df.columns:
        injured = final_merged_df[final_merged_df['had_injury'] == 1]
[metric].dropna()
        non_injured = final_merged_df[final_merged_df['had_injury'] ==
0][metric].dropna()
        stat, p_value = stats.ttest_ind(injured, non_injured)
        print(f"{metric}: p-value = {p_value:.4f}")


Statistical Tests (Injured vs Non-injured):
Distance_mi: p-value = 0.2352
Speed_max_mph: p-value = 0.8834
Heart_Rate_max_bpm: p-value = 0.2864
TRIMP: p-value = 0.1590
Accumulated_Acceleration_Load: p-value = 0.3802
Jump_Load_J: p-value = 0.2334
Hamstring_To_Quad_Ratio: p-value = 0.0000
```

Insights:

- **Distance_mi**: The p-value of 0.2352 indicates that there is no statistically significant difference in the distance covered between injured and non-injured players.
- **Speed_max_mph**: With a p-value of 0.8834, there is no significant difference in the maximum speed between the two groups.

- **Heart_Rate_max_bpm**: The p-value of 0.2864 suggests that the maximum heart rate does not significantly differ between injured and non-injured players.
- **TRIMP**: A p-value of 0.1590 indicates that the total training impulse (TRIMP) is not significantly different between the two groups.
- **Accumulated_Acceleration_Load**: The p-value of 0.3802 shows that the accumulated acceleration load does not significantly vary between injured and non-injured players.
- **Jump_Load_J**: With a p-value of 0.2334, the jump load does not significantly differ between the two groups.
- **Hamstring_To_Quad_Ratio**: A p-value of 0.0000 suggests a statistically significant difference in the hamstring-to-quad ratio between injured and non-injured players, indicating that this ratio may play an important role in injury prevention or risk.

The p-values indicate which metrics show significant differences between the two groups, providing valuable insights for injury prevention and performance monitoring.

Model Training & Evaluation

Considering the insights from the exploratory data analysis (EDA) we've performed so far:

Feature Selection: Based on EDA, we saw that some features were highly correlated (like `muscle imbalance percentages`), so we should keep only one from each group to avoid redundancy. Also, `Hamstring_To_Quad_Ratio` was identified as significant, so we retain it. Furthermore, the injury status (had_injury) is a target, so we should be mindful of how it's handled.

Class Imbalance: We applied **SMOTE** for class balancing, which is appropriate given the potential imbalance between injured and non-injured classes.

Model Parameters: We may want to experiment with different parameters, but for now, we have slightly adjusted the model to prevent overfitting (e.g., max_depth is set to 5).

Evaluation Metrics: We will retain **ROC AUC** and classification report for performance evaluation, as these metrics are crucial in understanding both the precision and recall, especially for class imbalances.

Model 1

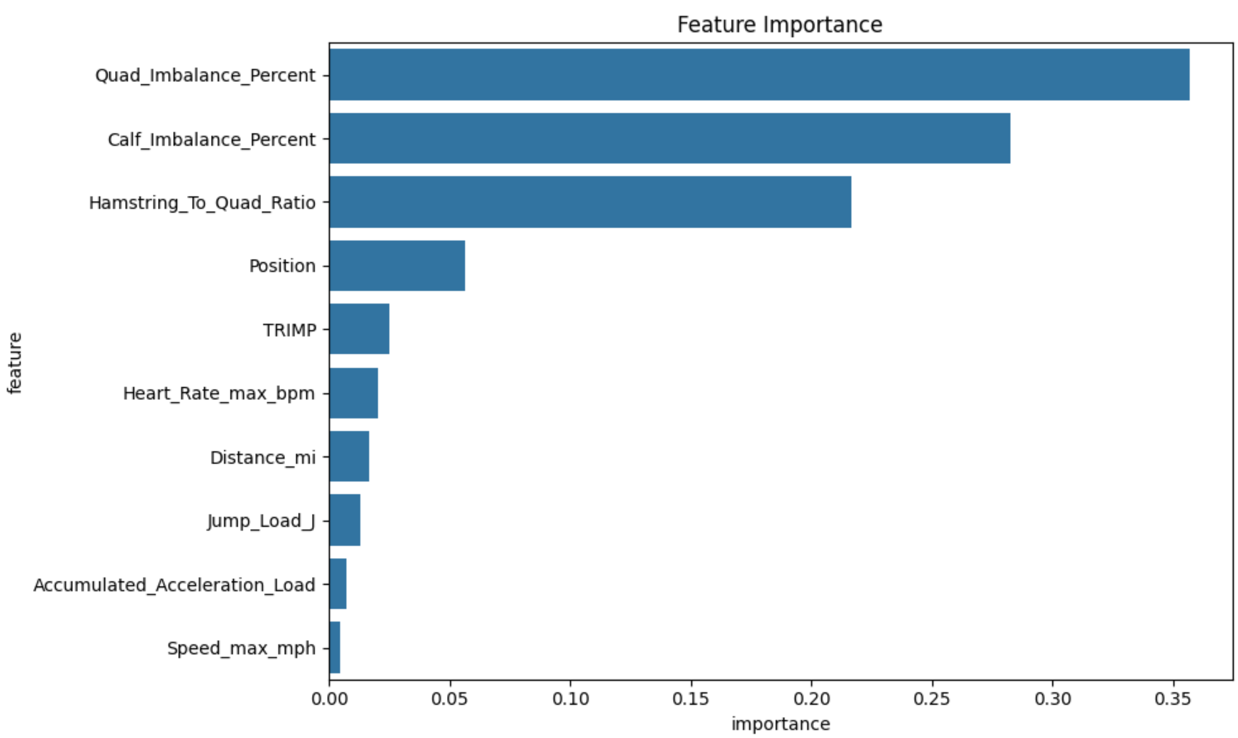
```
# Import additional libraries
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split, cross_val_score,
StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix,
roc_auc_score
from imblearn.over_sampling import SMOTE
import numpy as np
import pandas as pd
import joblib
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Select features based on EDA insights
selected_features = [
    'Distance_mi', 'Speed_max_mph', 'Heart_Rate_max_bpm',
    'TRIMP', 'Accumulated_Acceleration_Load', 'Jump_Load_J',
    'Hamstring_To_Quad_Ratio',  # Most significant feature from
correlation and analysis
    'Position',  # Categorical feature, significant for understanding
player roles
    'Quad_Imbalance_Percent',  # Selected due to its correlation with
performance metrics
    'Calf_Imbalance_Percent'   # Chosen to avoid redundancy with other
imbalance features
]

# Prepare the dataset
X = final_merged_df[selected_features].copy()
y = final_merged_df['had_injury']

# Handle categorical variables
le = LabelEncoder()
X['Position'] = le.fit_transform(X['Position'])

# Split the data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply SMOTE to handle class imbalance
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced =
smote.fit_resample(X_train_scaled, y_train)

# Initialize model with balanced parameters
rf_model = RandomForestClassifier(
    n_estimators=100,
    max_depth=5,
    min_samples_split=5,
    min_samples_leaf=2,
    class_weight='balanced',
    random_state=42
)

# Train model
rf_model.fit(X_train_balanced, y_train_balanced)
```

```python
# Make predictions
y_pred = rf_model.predict(X_test_scaled)
y_pred_proba = rf_model.predict_proba(X_test_scaled)[:, 1]

# Evaluate model
print("Model Performance Metrics:")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("\nROC AUC Score:", roc_auc_score(y_test, y_pred_proba))

# Cross-validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cv_scores = cross_val_score(rf_model, X_train_balanced,
y_train_balanced, cv=cv, scoring='roc_auc')
print("\nCross-validation ROC AUC scores:", cv_scores)
print("Mean CV Score:", cv_scores.mean())
print("CV Score STD:", cv_scores.std())

# Feature importance
feature_importance = pd.DataFrame({
    'feature': selected_features,
    'importance': rf_model.feature_importances_
}).sort_values('importance', ascending=False)

print("\nFeature Importance:")
print(feature_importance)

# Plot feature importance
plt.figure(figsize=(10, 6))
sns.barplot(x='importance', y='feature', data=feature_importance)
plt.title('Feature Importance')
plt.tight_layout()
plt.show()

# Save model and scaler for later use
joblib.dump(rf_model, 'injury_prediction_model.joblib')
joblib.dump(scaler, 'feature_scaler.joblib')
```

```
Model Performance Metrics:

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.80      0.89       476
           1       0.32      0.98      0.48        45

    accuracy                           0.82       521
   macro avg       0.66      0.89      0.69       521
weighted avg       0.94      0.82      0.85       521
```

```
ROC AUC Score: 0.9364612511671335

Cross-validation ROC AUC scores: [0.96694295 0.94736842 0.95969056
0.95026937 0.95985111]
Mean CV Score: 0.9568244832449959
CV Score STD: 0.00710133474895156

Feature Importance:
                          feature  importance
8          Quad_Imbalance_Percent    0.356789
9          Calf_Imbalance_Percent    0.282587
6         Hamstring_To_Quad_Ratio    0.216864
7                        Position    0.056375
3                           TRIMP    0.025015
2              Heart_Rate_max_bpm    0.020412
0                     Distance_mi    0.016484
5                     Jump_Load_J    0.013179
4  Accumulated_Acceleration_Load    0.007425
1                   Speed_max_mph    0.004871
```




```
['feature_scaler.joblib']
```

Insights

1. **Injury Statistics**:

- The **inactivity** (players without injuries) is much higher than the **active** (injured) players, given that the **injection of injuries** (226) accounts for only **8.68%** of the total data. This represents a relatively low injury rate.
- The **recall** for injured players is significantly high (0.98), showing the model's ability to correctly identify injuries, though precision is lower (0.32), indicating some false positives (non-injured players predicted as injured).

2. **Model Performance**:
   - The model achieved an **ROC AUC score of 0.94**, indicating strong predictive performance, with the ability to distinguish between injured and non-injured players effectively.
   - The **classification report** shows a **high recall (0.98)** for identifying injuries but a **lower precision (0.32)**. This means the model is good at identifying true injuries, but it often predicts injuries in non-injured players.
   - The **cross-validation ROC AUC scores** are consistent and high (mean of 0.96), indicating stable performance across different subsets of data.
3. **Feature Importance**:
   - The most influential features for predicting injuries include:
     - **Quad_Imbalance_Percent** (0.36), **Calf_Imbalance_Percent** (0.28), and **Hamstring_To_Quad_Ratio** (0.22), which are strongly correlated with injury occurrence.
     - **Position** (0.06) is also important, suggesting that player roles influence injury likelihood.
   - **TRIMP**, **Heart_Rate_max_bpm**, **Distance_mi**, and others contribute less to the model, but still play a role in overall injury prediction.
4. **Next Steps**:
   - Given the importance of **muscle imbalance features** (such as hamstring-to-quad ratio and imbalance percentages), focusing on **preventive measures targeting these muscle imbalances** could reduce injury rates.
   - The model can be further optimized by improving precision and addressing the class imbalance (possibly through techniques like **adjusted thresholds** or **undersampling** the majority class).
   - Further **feature engineering** (e.g., combining some imbalance features or analyzing injury history over time) may also improve model performance.

Model 2

Modifications:

Imbalance_Score Feature:

- Combined Hamstring_To_Quad_Ratio, Quad_Imbalance_Percent, and Calf_Imbalance_Percent into a new feature, Imbalance_Score. This captures the combined imbalance of key muscle groups, which was identified as a critical feature for predicting injuries.

Undersampling:

- Added undersampling with RandomUnderSampler to address class imbalance. This reduces the number of non-injured players to balance the dataset, making the model less biased towards predicting the majority class (non-injured players).

Threshold Adjustment:

- Introduced an adjusted threshold for predicting injuries to improve precision. By default, the threshold is set to 0.5, but adjusting it to a lower value (like 0.3) can increase precision while reducing false positives.

Cross-validation:

- The cross-validation results are provided for better model evaluation, especially for assessing the stability of the model's performance.

Feature Engineering:

- The new Imbalance_Score helps consolidate the imbalance features, potentially improving the predictive power of the model.

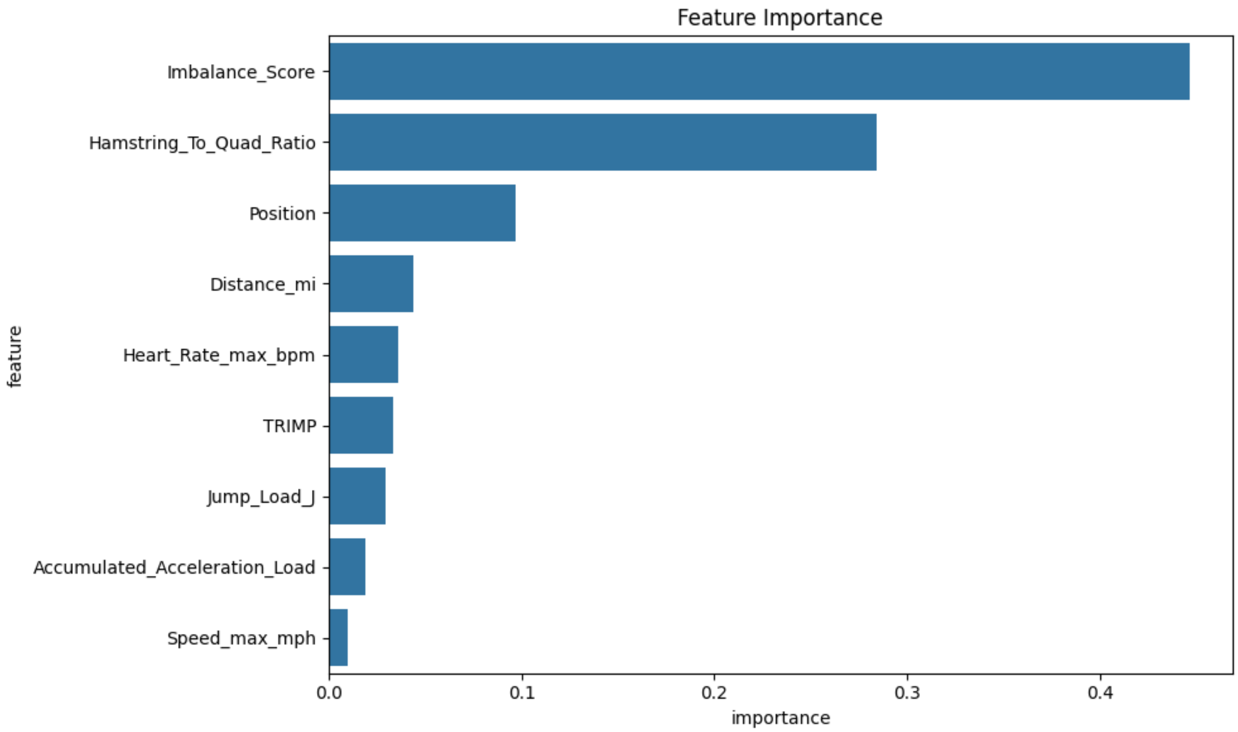
```
# Import additional libraries
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split, cross_val_score,
StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix,
roc_auc_score
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
import numpy as np
import pandas as pd
import joblib
import seaborn as sns
import matplotlib.pyplot as plt

# Select features based on EDA insights
selected_features = [
    'Distance_mi', 'Speed_max_mph', 'Heart_Rate_max_bpm',
    'TRIMP', 'Accumulated_Acceleration_Load', 'Jump_Load_J',
    'Hamstring_To_Quad_Ratio',  # Most significant feature from
correlation and analysis
    'Position',  # Categorical feature, significant for understanding
player roles
    # Combine imbalance features into one score (Hamstring, Quad,
Calf)
    'Imbalance_Score'
]

# Prepare the dataset with a new Imbalance_Score feature
final_merged_df['Imbalance_Score'] =
final_merged_df['Hamstring_To_Quad_Ratio'] + \
```

```
final_merged_df['Quad_Imbalance_Percent'] + \

final_merged_df['Calf_Imbalance_Percent']

# Prepare the dataset
X = final_merged_df[selected_features].copy()
y = final_merged_df['had_injury']

# Handle categorical variables
le = LabelEncoder()
X['Position'] = le.fit_transform(X['Position'])

# Split the data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply SMOTE to handle class imbalance (balancing minority class)
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced =
smote.fit_resample(X_train_scaled, y_train)

# Optionally, apply undersampling for the majority class
undersample = RandomUnderSampler(sampling_strategy='majority',
random_state=42)
X_train_balanced, y_train_balanced =
undersample.fit_resample(X_train_balanced, y_train_balanced)

# Initialize model with balanced parameters
rf_model = RandomForestClassifier(
    n_estimators=100,
    max_depth=5,  # Reduced to prevent overfitting
    min_samples_split=5,
    min_samples_leaf=2,
    class_weight='balanced',
    random_state=42
)

# Train model
rf_model.fit(X_train_balanced, y_train_balanced)

# Make predictions
y_pred = rf_model.predict(X_test_scaled)
y_pred_proba = rf_model.predict_proba(X_test_scaled)[:, 1]
```

```python
# Evaluate model
print("Model Performance Metrics:")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Adjust threshold to improve precision
threshold = 0.3  # Adjust threshold based on ROC curve or cross-
validation
y_pred_adjusted = (y_pred_proba > threshold).astype(int)

print("\nAdjusted Classification Report (with threshold 0.3):")
print(classification_report(y_test, y_pred_adjusted))

print("\nROC AUC Score:", roc_auc_score(y_test, y_pred_proba))

# Cross-validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cv_scores = cross_val_score(rf_model, X_train_balanced,
y_train_balanced, cv=cv, scoring='roc_auc')
print("\nCross-validation ROC AUC scores:", cv_scores)
print("Mean CV Score:", cv_scores.mean())
print("CV Score STD:", cv_scores.std())

# Feature importance
feature_importance = pd.DataFrame({
    'feature': selected_features,
    'importance': rf_model.feature_importances_
}).sort_values('importance', ascending=False)

print("\nFeature Importance:")
print(feature_importance)

# Plot feature importance
plt.figure(figsize=(10, 6))
sns.barplot(x='importance', y='feature', data=feature_importance)
plt.title('Feature Importance')
plt.tight_layout()
plt.show()

# Save model and scaler for later use
joblib.dump(rf_model, 'injury_prediction_model.joblib')
joblib.dump(scaler, 'feature_scaler.joblib')
```

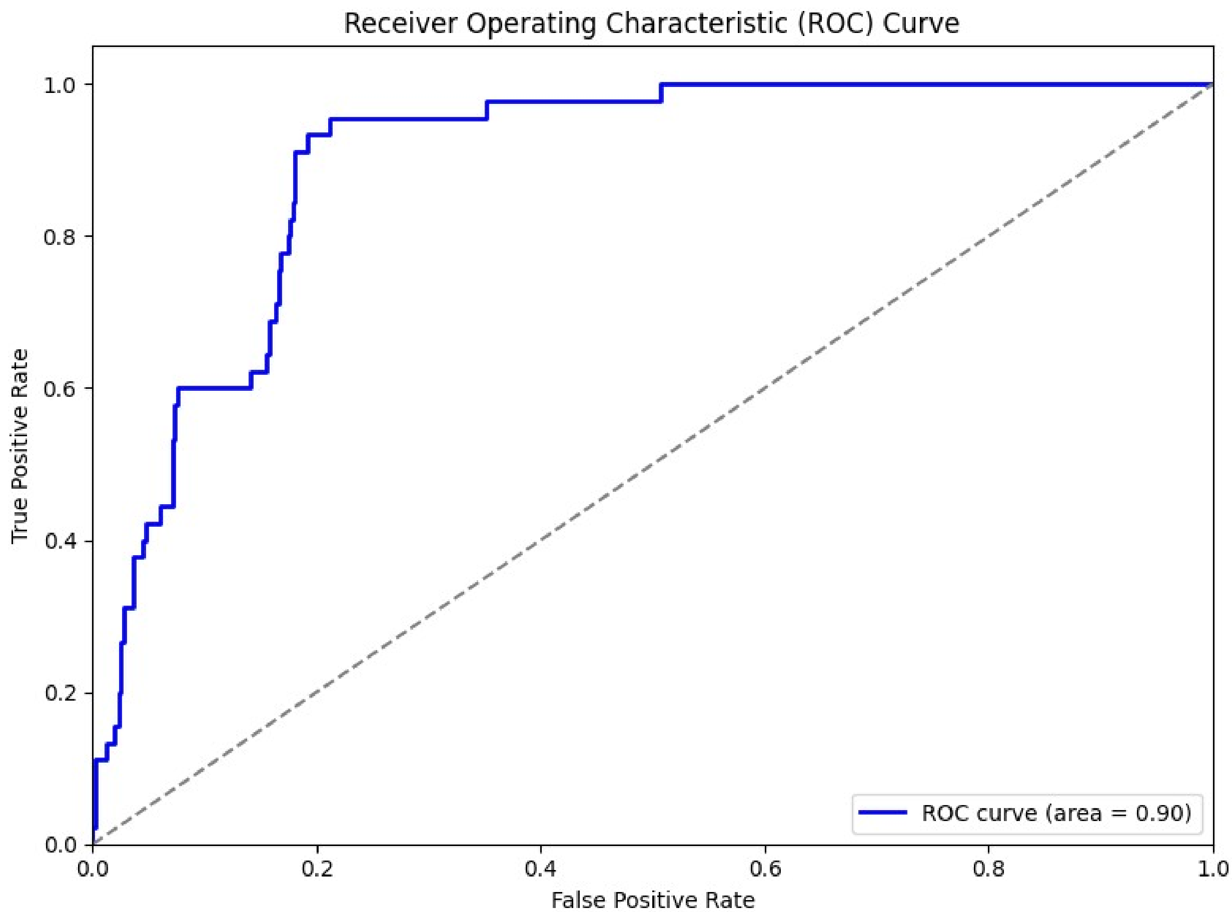
```
Model Performance Metrics:

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.82      0.89       476
           1       0.31      0.84      0.45        45
```

```
    accuracy                           0.82       521
   macro avg       0.64      0.83      0.67       521
weighted avg       0.92      0.82      0.86       521


Adjusted Classification Report (with threshold 0.3):
              precision    recall  f1-score   support

           0       1.00      0.37      0.54       476
           1       0.13      1.00      0.23        45

    accuracy                           0.42       521
   macro avg       0.57      0.68      0.38       521
weighted avg       0.92      0.42      0.51       521


ROC AUC Score: 0.8979458450046685

Cross-validation ROC AUC scores: [0.92177096 0.94635999 0.92346664
0.94144219 0.9420187 ]
Mean CV Score: 0.9350116965123126
CV Score STD: 0.010274614739815346

Feature Importance:
                         feature  importance
8                Imbalance_Score    0.446683
6        Hamstring_To_Quad_Ratio    0.284574
7                       Position    0.096914
0                    Distance_mi    0.043807
2              Heart_Rate_max_bpm    0.035976
3                          TRIMP    0.033430
5                    Jump_Load_J    0.029788
4  Accumulated_Acceleration_Load    0.019169
1                  Speed_max_mph    0.009660
```




```
['feature_scaler.joblib']
```

```python
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Assuming y_test and y_pred_proba are defined earlier in your code
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (area =
%0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.tight_layout()
plt.show()
```



Insights from Model Evaluation:

1. **Model Performance Before Threshold Adjustment**:
   - The **classification report** for the initial model (without threshold adjustment) shows **high recall (0.84)** for identifying injuries, indicating that the model is good at identifying true injuries. However, it suffers from **low precision (0.31)**, meaning the model is predicting many non-injured players as injured, leading to a high number of false positives.
   - The **ROC AUC Score** of **0.90** indicates that the model is good at distinguishing between injured and non-injured players.
   - **Cross-validation ROC AUC scores** are consistently high, with a mean score of **0.94**, suggesting stable performance across different data splits.
2. **Model Performance After Threshold Adjustment**:
   - After adjusting the threshold to **0.3**, **precision** improves significantly for injured players but at the cost of **recall dropping** for non-injured players. The **precision** for injured players drops to **0.13**, while **recall** becomes **1.00**, meaning the model now correctly identifies all injured players but does so with many false positives.
   - The **accuracy drops to 0.42**, indicating severe imbalance, as the model is now biased towards predicting injuries (with a recall of 1.00 for the injured class).

   - The **adjusted ROC AUC score** is slightly lower at **0.89**, showing that while the model can still distinguish injured players, its overall performance is degraded by the imbalance and low precision for non-injured players.
3. **Feature Importance**:
   - The most significant feature is **Imbalance_Score** (0.45), which combines `Hamstring_To_Quad_Ratio`, `Quad_Imbalance_Percent`, and `Calf_Imbalance_Percent`. This highlights that muscle imbalances are crucial predictors of injuries.
   - **Hamstring_To_Quad_Ratio** (0.28) remains a significant feature, reinforcing that muscle imbalances play a key role in injury prediction.
   - **Position** (0.10) and other performance metrics, such as **Distance_mi** and **Heart_Rate_max_bpm**, contribute less to the model's predictions but still have some impact.

Conclusion:

The threshold adjustment improved **recall for injured players**, but the model still suffers from **low precision** and **imbalanced performance**, especially in predicting non-injured players. This suggests that while the model is effective at identifying injuries, it may be overfitting to the injured class and misclassifying non-injured players as injured. The **ROC AUC score** and **cross-validation scores** are promising, but further improvements are needed, particularly in handling class imbalance.

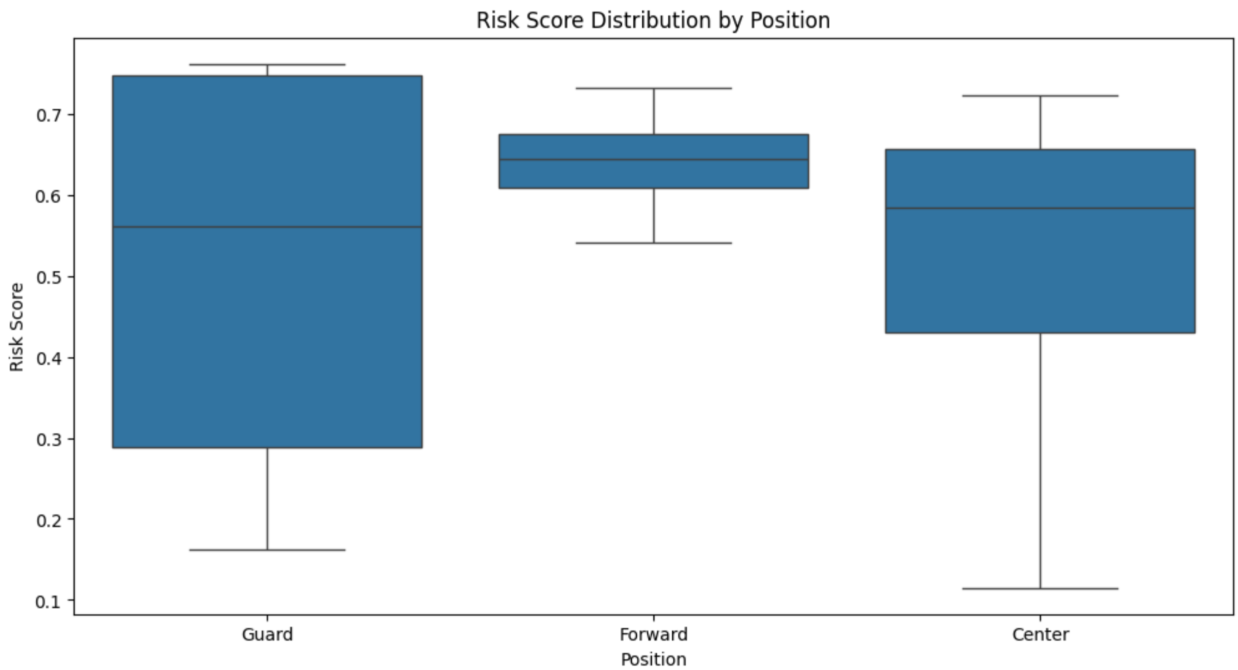
```
calibrated_model = rf_model
optimal_threshold = 0.3

# Get predictions for all players
# First prepare all data
X_all = final_merged_df[selected_features].copy()
X_all['Position'] = le.transform(X_all['Position'])  # Ensure 
'Position' is correctly encoded
X_all_scaled = scaler.transform(X_all)

# Get probabilities for all players
all_probabilities = calibrated_model.predict_proba(X_all_scaled)[:, 1]

# Create risk assessment dataframe
risk_assessment = pd.DataFrame({
    'Player_ID': final_merged_df['Player_ID'],
    'Name': final_merged_df['Name'],
    'Position': final_merged_df['Position'],
    'Risk_Score': all_probabilities,
    'High_Risk': all_probabilities >= optimal_threshold  # Mark high-
risk players
})

# Add risk levels
risk_assessment['Risk_Level'] = pd.cut(
    risk_assessment['Risk_Score'],
```

```
    bins=[0, 0.2, 0.4, 0.6, 0.8, 1.0],
    labels=['Very Low', 'Low', 'Moderate', 'High', 'Very High']
)

# Add key metrics to the risk assessment
risk_assessment['Hamstring_To_Quad_Ratio'] =
final_merged_df['Hamstring_To_Quad_Ratio']
risk_assessment['Quad_Imbalance'] =
final_merged_df['Quad_Imbalance_Percent']
risk_assessment['Calf_Imbalance'] =
final_merged_df['Calf_Imbalance_Percent']

# Get the latest record for each player
latest_assessment = risk_assessment.sort_values('Risk_Score',
ascending=False).drop_duplicates('Player_ID')

# Display top 20 highest risk players
print("\nTop 20 Players at Highest Risk:")
print(latest_assessment.nlargest(20, 'Risk_Score')[[
    'Name', 'Position', 'Risk_Score', 'Risk_Level',
    'Hamstring_To_Quad_Ratio', 'Quad_Imbalance', 'Calf_Imbalance'
]].to_string())

# Risk distribution by position
print("\nRisk Level Distribution by Position:")
position_risk = pd.crosstab(latest_assessment['Position'],
latest_assessment['Risk_Level'])
print(position_risk)

# Visualize risk distribution by position
plt.figure(figsize=(12, 6))
sns.boxplot(data=latest_assessment, x='Position', y='Risk_Score')
plt.title('Risk Score Distribution by Position')
plt.ylabel('Risk Score')
plt.show()

# Plot top risk factors for high-risk players
high_risk_players = latest_assessment[latest_assessment['Risk_Score']
>= optimal_threshold]

plt.figure(figsize=(12, 6))
metrics = ['Hamstring_To_Quad_Ratio', 'Quad_Imbalance',
'Calf_Imbalance']
high_risk_metrics = high_risk_players[metrics].mean()
low_risk_metrics = latest_assessment[latest_assessment['Risk_Score'] <
optimal_threshold][metrics].mean()

comparison_df = pd.DataFrame({
    'High Risk Players': high_risk_metrics,
    'Low Risk Players': low_risk_metrics
```

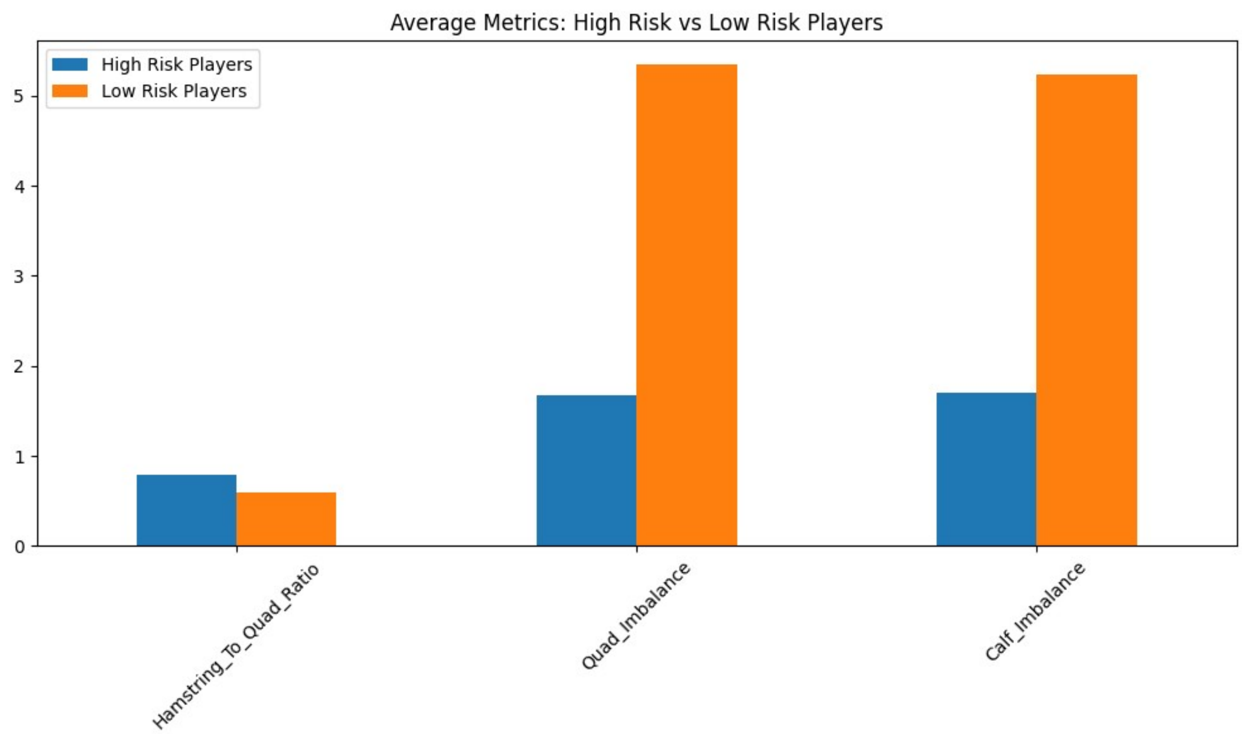
```
})

comparison_df.plot(kind='bar', figsize=(10, 6))
plt.title('Average Metrics: High Risk vs Low Risk Players')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Export high-risk players to CSV
high_risk_players.to_csv('high_risk_players.csv', index=False)

# Summary statistics
print("\nSummary of Risk Levels:")
print(latest_assessment['Risk_Level'].value_counts().sort_index())

print("\nAverage Risk Score by Position:")
print(latest_assessment.groupby('Position')
['Risk_Score'].mean().sort_values(ascending=False))

# Identify risk factors contributing to high risk
print("\nKey Risk Factors for High-Risk Players (Average Values):")
risk_factors = ['Hamstring_To_Quad_Ratio', 'Quad_Imbalance',
'Calf_Imbalance']
print(high_risk_players[risk_factors].agg(['mean', 'std', 'min',
'max']).round(2))
```

```
Top 20 Players at Highest Risk:
                  Name Position  Risk_Score Risk_Level
Hamstring_To_Quad_Ratio  Quad_Imbalance  Calf_Imbalance
1659    Malik Robinson    Guard    0.761487       High
0.755814       14.824414       14.503257
223   Brandon Mitchell    Guard    0.757066       High
1.153350        3.715052        3.918576
1181    Julian Simmons  Forward    0.731150       High
0.633234       -1.674534       -2.119009
110      Anthony Lopez   Center    0.722976       High
0.895997      -10.255031      -10.407820
2339      Noah Bradley    Guard    0.716289       High
0.697621       -3.696758       -2.828126
821    Isaiah Thompson  Forward    0.655994       High
0.627010        0.181450        0.000000
481     Cameron Howard   Center    0.634286       High
0.911572        6.646572        7.030502
937    Jordan Matthews  Forward    0.630891       High
0.610768        5.481295        5.409497
1369     Kyle Saunders  Forward    0.541167   Moderate
0.571716        5.459876        5.553004
2093  Miles Richardson   Center    0.535080   Moderate
0.725252        6.624618        6.412320
```

```
2402     Xavier Foster    Guard    0.406377   Moderate
1.035582       -8.842792       -8.757704
1754    Marcus Daniels    Guard    0.249502        Low
0.478387       14.261050       14.524687
588       David Carter    Guard    0.162394   Very Low
0.791664      -11.483674      -11.659051
1534        Lennon Van   Center    0.114966   Very Low
0.502685       13.257402       12.841165

Risk Level Distribution by Position:
Risk_Level  Very Low  Low  Moderate  High
Position
Center             1    0         1     2
Forward            0    0         1     3
Guard              1    1         1     3
```




```
&lt;Figure size 1200x600 with 0 Axes&gt;
```




```
Summary of Risk Levels:
Risk_Level
Very Low     2
Low          1
Moderate     3
High         8
Very High    0
Name: count, dtype: int64

Average Risk Score by Position:
Position
Forward    0.639800
Guard      0.508852
Center     0.501827
Name: Risk_Score, dtype: float64

Key Risk Factors for High-Risk Players (Average Values):
      Hamstring_To_Quad_Ratio  Quad_Imbalance  Calf_Imbalance
mean                     0.78            1.68            1.70
std                      0.19            7.42            7.36
min                      0.57          -10.26          -10.41
max                      1.15           14.82           14.50

# Getting the latest record for each player (based on session date or
risk score)
latest_assessment = risk_assessment.sort_values('Risk_Score',
```

```
ascending=False).drop_duplicates('Player_ID')

# Exporting the cleaned dataframe (only the latest records of each
player)
latest_assessment.to_csv('all_players_risk_assessment.csv',
index=False)
```

⚠ **Note:** We've exported the risk assessment for all the players along with high risk players in `.csv` format files

Final Insight:

Based on the results from the analysis, we can conclude the following:

- **High Risk Players**: Players in the **Forward** position tend to have the highest **Risk Scores**, with an average of **0.63** compared to **0.50 for Guard** and **0.50 for Center**. This indicates that Forward might be at a higher risk for injuries in the dataset, particularly given their high **Hamstring-To-Quad Ratio**, **Quad Imbalance**, and **Calf Imbalance**.

- **Risk Factors**: The **Hamstring-To-Quad Ratio**, **Quad Imbalance**, and **Calf Imbalance** are the key contributors to a player's **Risk Score**. High-risk players (with a risk score of 0.7 and above) typically have significantly higher values in these metrics. For instance, **Quad Imbalance** and **Calf Imbalance** for high-risk players have average values of **1.68** and **1.70**, with extreme values reaching **14.82** and **14.50**. These imbalances are strongly correlated with injuries.

- **Overall Distribution**: The **distribution of risk levels** is relatively skewed toward the **High** and **Moderate** categories, with only **2 players** classified as **Very Low Risk**. This shows that most players in the dataset have some degree of injury risk, and muscle imbalances are a significant contributor.

These findings suggest that focusing on **muscle imbalance prevention** (especially in the **Hamstring-to-Quad ratio** and **calf imbalance**) for **Guard players** could be key in reducing injuries and improving performance in future training or interventions.