Preliminary Analysis

AUTHOR
Vishnu, Mia, and Julia

```r
library(tidyverse)
```

```
── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
✔ dplyr     1.1.4     ✔ readr     2.1.5
✔ forcats   1.0.0     ✔ stringr   1.5.1
✔ ggplot2   3.5.1     ✔ tibble    3.2.1
✔ lubridate 1.9.3     ✔ tidyr     1.3.1
✔ purrr     1.0.4     
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts
to become errors
```

```r
library(gtsummary)
library(ggplot2)
library(psych)
```

```

Attaching package: 'psych'

The following objects are masked from 'package:ggplot2':

    %+%, alpha
```

```r
library(cluster)
library(factoextra)
```

```
Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```r
library(dplyr)
library(reticulate)
library(here)
```

```
here() starts at /Users/Vishnu/Documents/Classes/MATH 244 - Intermediate Data
Science/M244-Final-JVM
```

Introduction and Data

The data set includes statistics from the 2024 Division 1 and 3 Men's and Women's Ultimate Frisbee Championships. The statistics were found on USA Ultimate, the non-profit organization serving as the governing body for ultimate in the United States, and were taken from a data visualization titled "USA Ultimate 2024 Nationals Stats Dashboard", which was created by Ben Ayres. The data set includes 1665 rows which each correspond to an individual player, and it includes 15 variables which categorize the players by Division, Gender, and Team, and provide game statistics for each player.

Ultimate frisbee is a sport growing in popularity at the collegiate level and within the Vassar student body as well. However, not much data analysis is available for Ultimate compared to other popular sports. We want to fit a model that would help players analyze their game performance. Therefore, via linear, LASSO, and ridge regression we will fit a prediction model that we train under supervised conditions to be able to predict a player's plus/minus score (AKA individual impact) based on the variables turns_per_game, ds_per_game, ast_per_game, and pts_per_game. These variables all relate to a player's effectiveness on the field, which is why we will use them to predict pls_mns_per_game. We will also stratify by division and gender, so that we ensure that we have an accurate model for each group. We will have four models in the final product, DI Men, DI Women, DIII Men, and DIII Women.

Since the data was already pretty clean, we did not have to do much data tidying. We just did some string manipulation to extract the school name from the team name, creating a new column called 'school', and had to convert some character variables to factors.

```
ultimate_data <- read_csv("/Users/Vishnu/Documents/Classes/MATH 244 - Intermec
```

```
Rows: 1665 Columns: 17
── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
chr  (6): player, level, gender, division, team_name, school
dbl (11): Turns, Ds, Assists, Points, plus_minus, team_games, turns_per_game...

ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Methodology

Summary Statistics

```
ultimate_data %>% select(-c(player, team_name)) %>% gtsummary::tbl_summary()
```

Characteristic	**N = 1,665**[1]
level	
Division 1	973 (58%)
Division 3	692 (42%)
gender	
Men	893 (54%)
Women	772 (46%)

[1] n (%); Median (Q1, Q3)

Characteristic	**N = 1,665**[1]
division	
Division 1 Men	521 (31%)
Division 1 Women	452 (27%)
Division 3 Men	372 (22%)
Division 3 Women	320 (19%)
Turns	2 (0, 7)
Ds	1.00 (0.00, 3.00)
Assists	1.0 (0.0, 3.0)
Points	1.0 (0.0, 4.0)
plus_minus	1 (0, 5)
team_games	
5	780 (47%)
6	738 (44%)
7	122 (7.3%)
8	25 (1.5%)
turns_per_game	0.40 (0.00, 1.17)
ds_per_game	0.20 (0.00, 0.50)
ast_per_game	0.17 (0.00, 0.60)
pts_per_game	0.20 (0.00, 0.71)
pls_mns_per_game	0.20 (0.00, 0.83)
school	
Alabama-Huntsville	23 (1.4%)
Bates	17 (1.0%)
Berry	26 (1.6%)
British Columbia	23 (1.4%)
Brown	25 (1.5%)
Cal	23 (1.4%)
Cal Poly SLO	24 (1.4%)

[1] n (%); Median (Q1, Q3)

Characteristic	N = 1,665[1]
Carleton	101 (6.1%)
Claremont	24 (1.4%)
Colorado	49 (2.9%)
Colorado College	22 (1.3%)
Colorado State	21 (1.3%)
Davenport	31 (1.9%)
Franciscan	22 (1.3%)
Georgia	45 (2.7%)
Grinnell	15 (0.9%)
Haverford/Bryn	24 (1.4%)
Lewis & Clark	44 (2.6%)
Macalester	20 (1.2%)
Massachusetts	28 (1.7%)
Michigan	54 (3.2%)
Middlebury	53 (3.2%)
Minnesota	28 (1.7%)
Missouri S&T	15 (0.9%)
Mount Holyoke	12 (0.7%)
NC State	30 (1.8%)
North Carolina	61 (3.7%)
Oberlin	22 (1.3%)
Occidental	23 (1.4%)
Oklahoma Christian	24 (1.4%)
Oregon	51 (3.1%)
Oregon State	22 (1.3%)
Ottawa	16 (1.0%)
Penn State	24 (1.4%)
Pennsylvania	22 (1.3%)

[1] n (%); Median (Q1, Q3)

Characteristic	N = 1,665[1]
Pittsburgh	28 (1.7%)
Portland	24 (1.4%)
Richmond	39 (2.3%)
Rochester	25 (1.5%)
St Olaf	51 (3.1%)
Stanford	22 (1.3%)
SUNY Binghamton	22 (1.3%)
Texas	26 (1.6%)
Tufts	21 (1.3%)
UCSB	23 (1.4%)
UCSD	25 (1.5%)
Union	18 (1.1%)
USCS	21 (1.3%)
Utah	22 (1.3%)
Vermont	46 (2.8%)
Victoria	20 (1.2%)
Washington	26 (1.6%)
Washington University	29 (1.7%)
Wellesley	20 (1.2%)
Wesleyan	22 (1.3%)
Western Washington	23 (1.4%)
Whitman	22 (1.3%)
Williams	26 (1.6%)

[1] n (%); Median (Q1, Q3)

```
summary_data <- ultimate_data[, c("pls_mns_per_game", "turns_per_game", "ds_pe

summary(summary_data)
```

```
 pls_mns_per_game  turns_per_game     ds_per_game      ast_per_game
 Min.   :-8.8333   Min.   : 0.0000   Min.   :0.0000   Min.   :0.0000
 1st Qu.: 0.0000   1st Qu.: 0.0000   1st Qu.:0.0000   1st Qu.:0.0000
 Median : 0.2000   Median : 0.4000   Median :0.2000   Median :0.1667  
 Mean   : 0.4430   Mean   : 0.9229   Mean   :0.3543   Mean   :0.5050  
 3rd Qu.: 0.8333   3rd Qu.: 1.1667   3rd Qu.:0.5000   3rd Qu.:0.6000  
 Max.   : 5.8750   Max.   :14.8333   Max.   :7.1667   Max.   :7.6000  
  pts_per_game   
 Min.   :0.0000  
 1st Qu.:0.0000  
 Median :0.2000  
 Mean   :0.5066  
 3rd Qu.:0.7143  
 Max.   :4.4000  
```

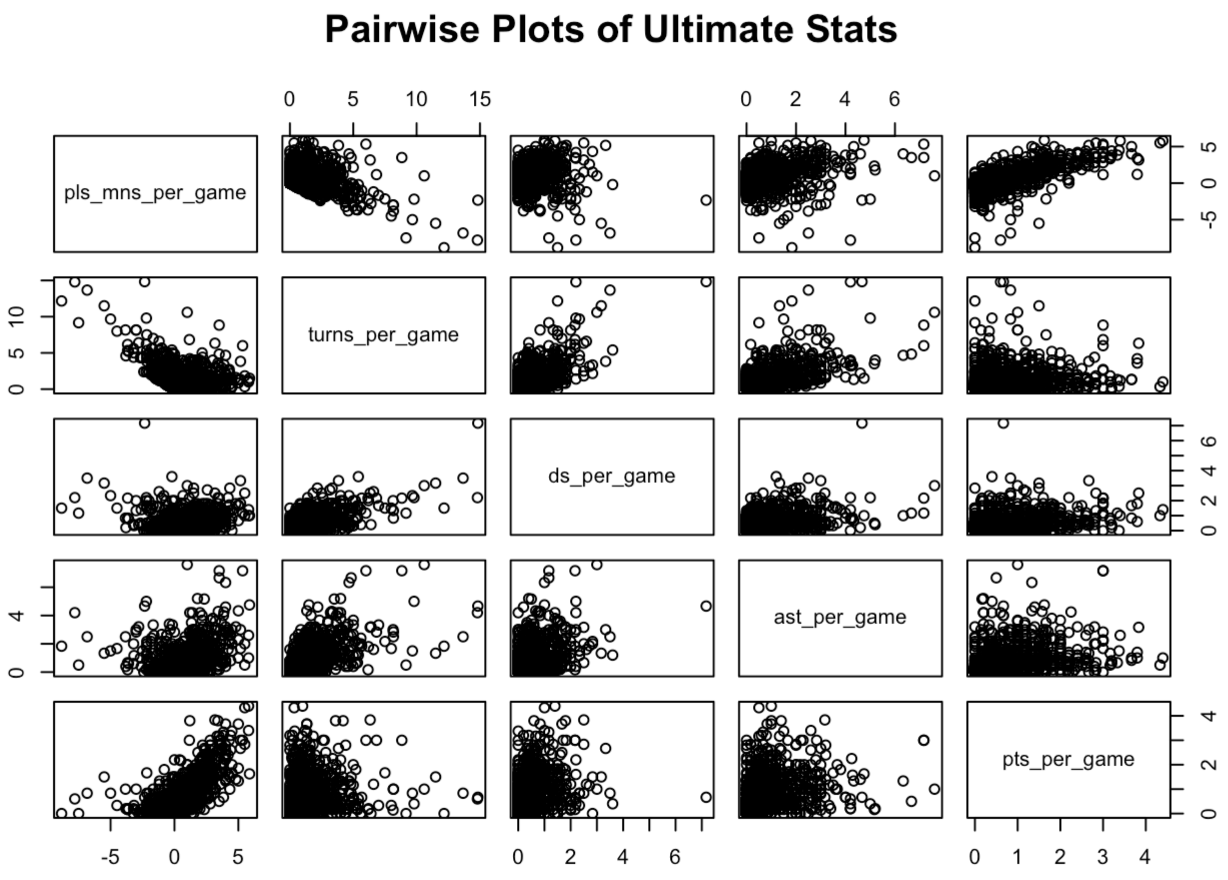
```
describe(summary_data)
```

```
                 vars    n mean   sd median trimmed  mad   min   max range skew
pls_mns_per_game    1 1665 0.44 1.27   0.20    0.35 0.59 -8.83  5.88 14.71 0.12
turns_per_game      2 1665 0.92 1.46   0.40    0.61 0.59  0.00 14.83 14.83 3.86
ds_per_game         3 1665 0.35 0.51   0.20    0.25 0.30  0.00  7.17  7.17 3.37
ast_per_game        4 1665 0.51 0.88   0.17    0.30 0.25  0.00  7.60  7.60 3.05
pts_per_game        5 1665 0.51 0.67   0.20    0.37 0.30  0.00  4.40  4.40 2.04
                 kurtosis   se
pls_mns_per_game     6.68 0.03
turns_per_game      22.72 0.04
ds_per_game         23.71 0.01
ast_per_game        12.78 0.02
pts_per_game         4.88 0.02
```

```
cor(summary_data, use = "complete.obs")
```

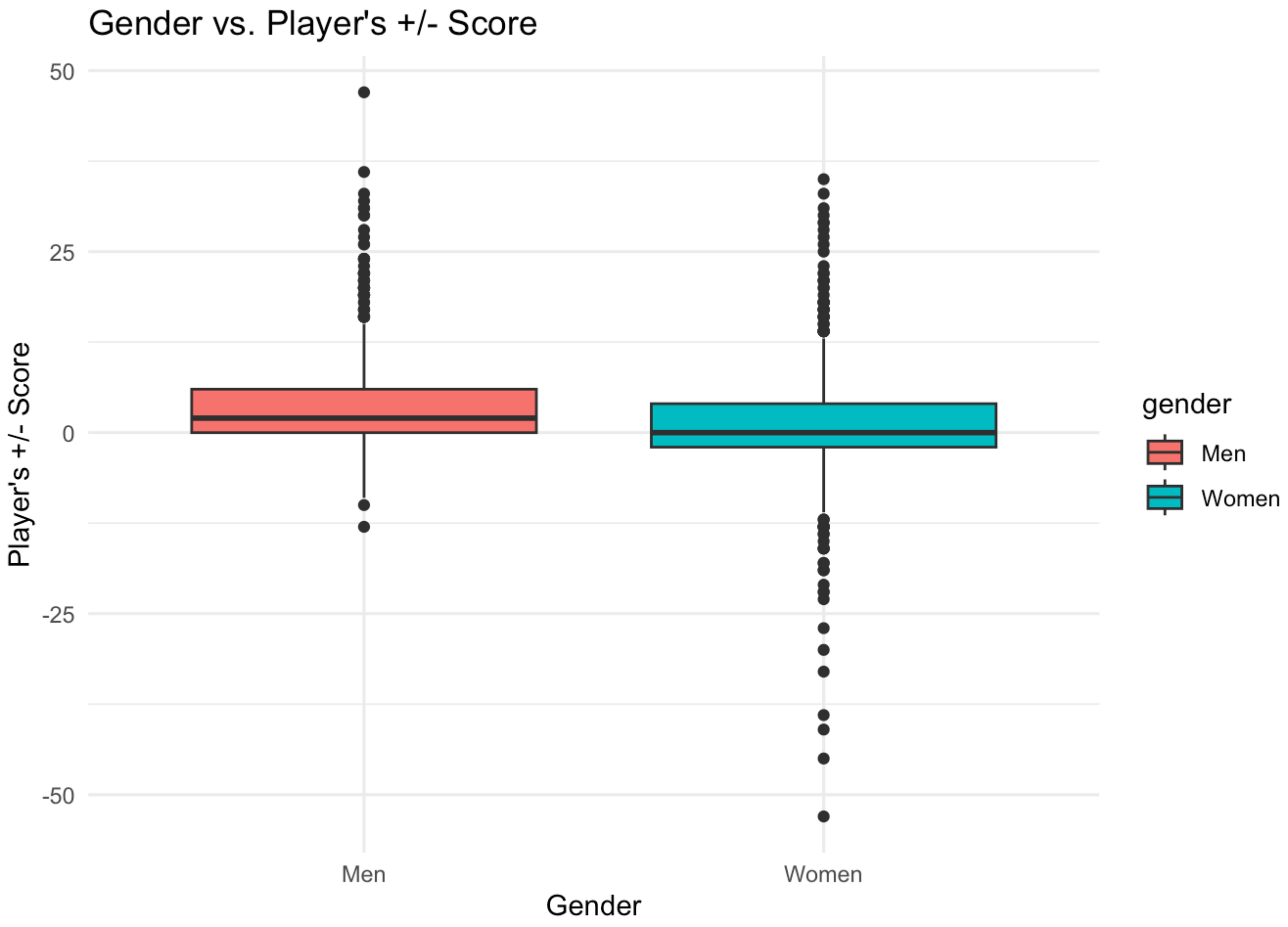
```
                 pls_mns_per_game turns_per_game ds_per_game ast_per_game
pls_mns_per_game        1.0000000     -0.2626174   0.1892149    0.3830593
turns_per_game         -0.2626174      1.0000000   0.6852003    0.6550684
ds_per_game             0.1892149      0.6852003   1.0000000    0.5028222
ast_per_game            0.3830593      0.6550684   0.5028222    1.0000000
pts_per_game            0.6805235      0.2856562   0.4187957    0.4498837
                 pts_per_game
pls_mns_per_game    0.6805235
turns_per_game      0.2856562
ds_per_game         0.4187957
ast_per_game        0.4498837
pts_per_game        1.0000000
```

```
pairs(summary_data, main = "Pairwise Plots of Ultimate Stats")
```

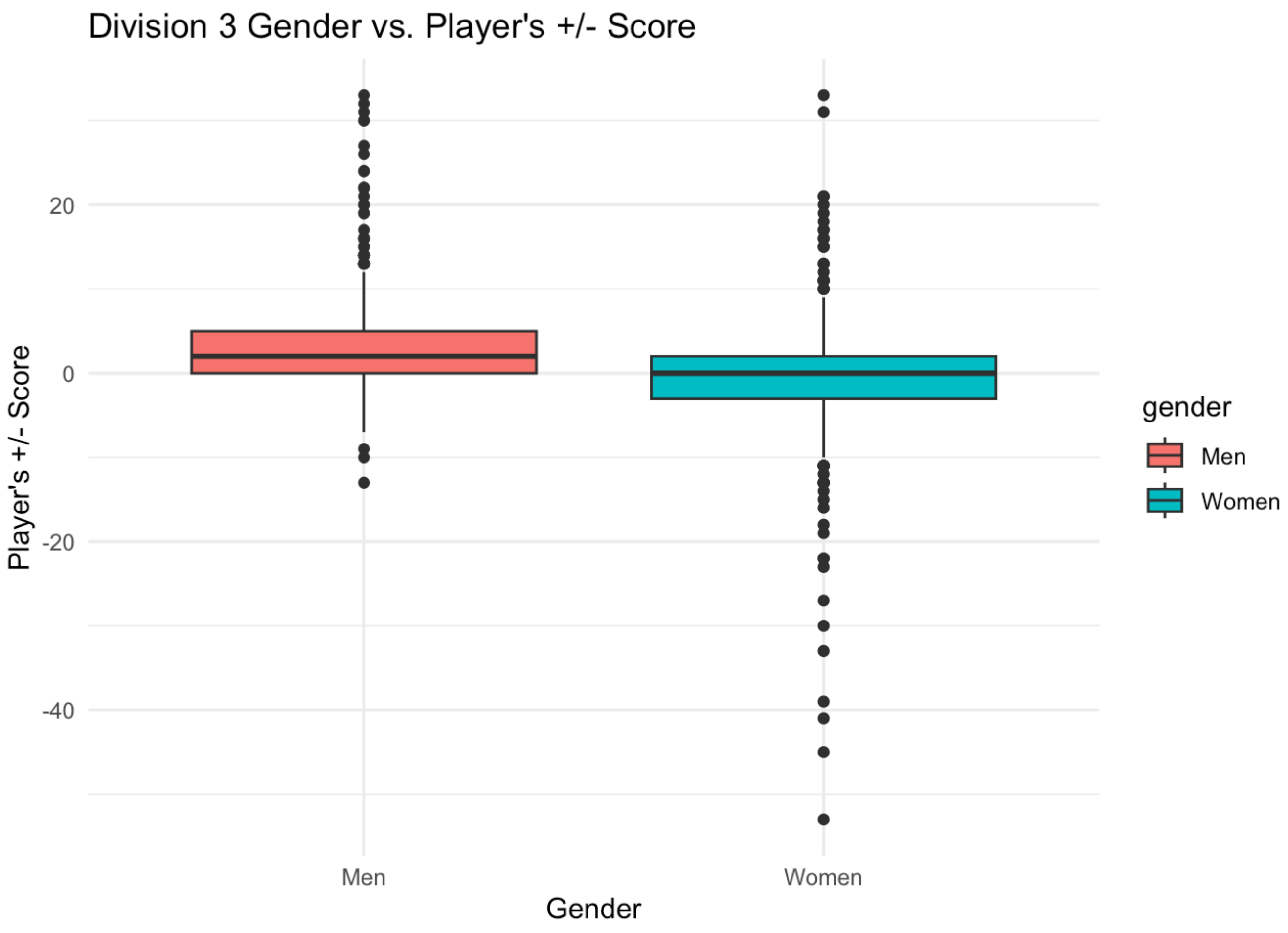


Visualizations/EDA

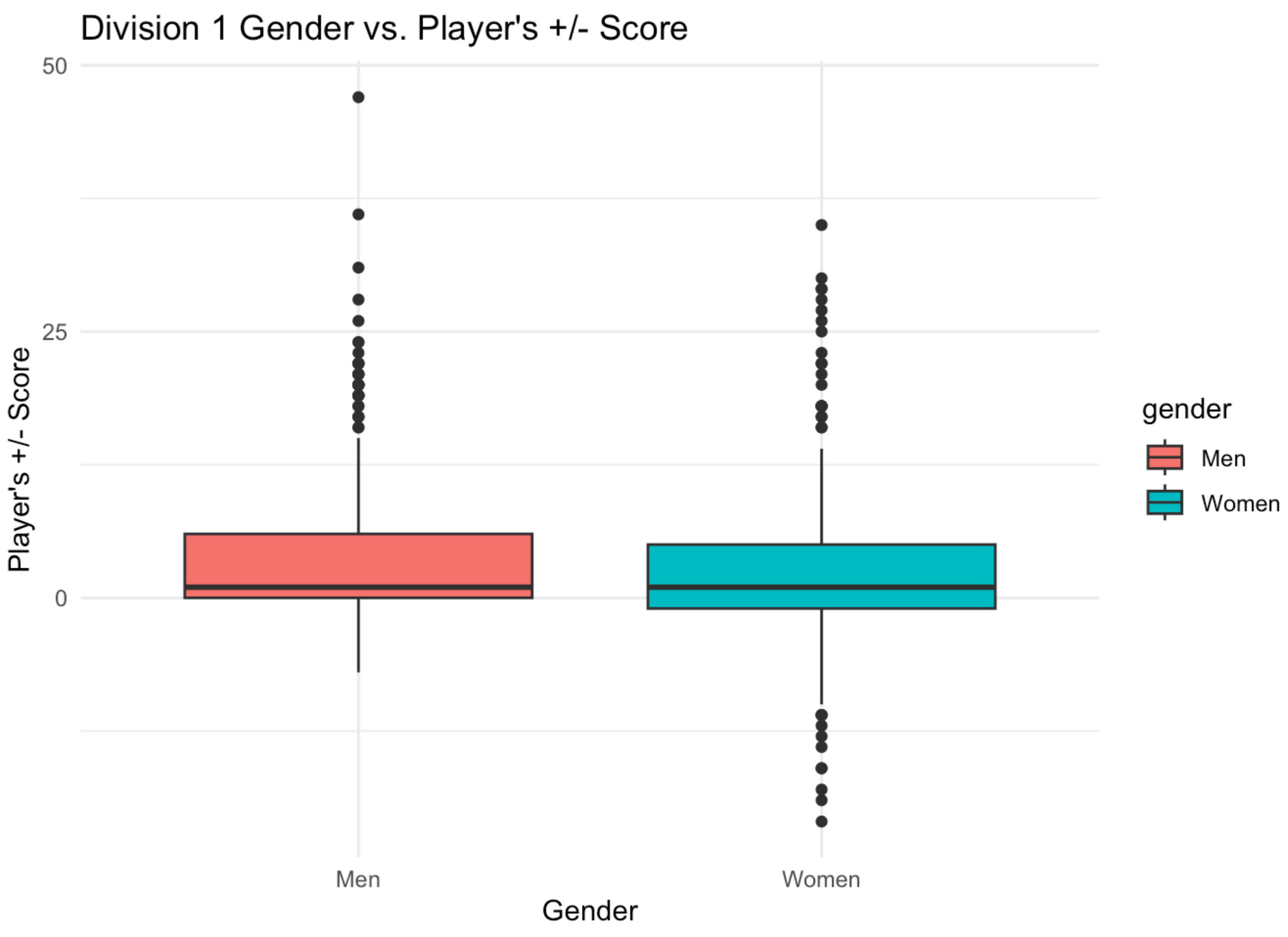
```
# Plot player's plus–minus scores and their gender
ggplot(ultimate_data, aes(x = gender, y = plus_minus, fill = gender)) +
  geom_boxplot() +
  labs(
    title = "Gender vs. Player's +/– Score",
    x = "Gender",
    y = "Player's +/– Score"
  ) +
  theme_minimal()
```



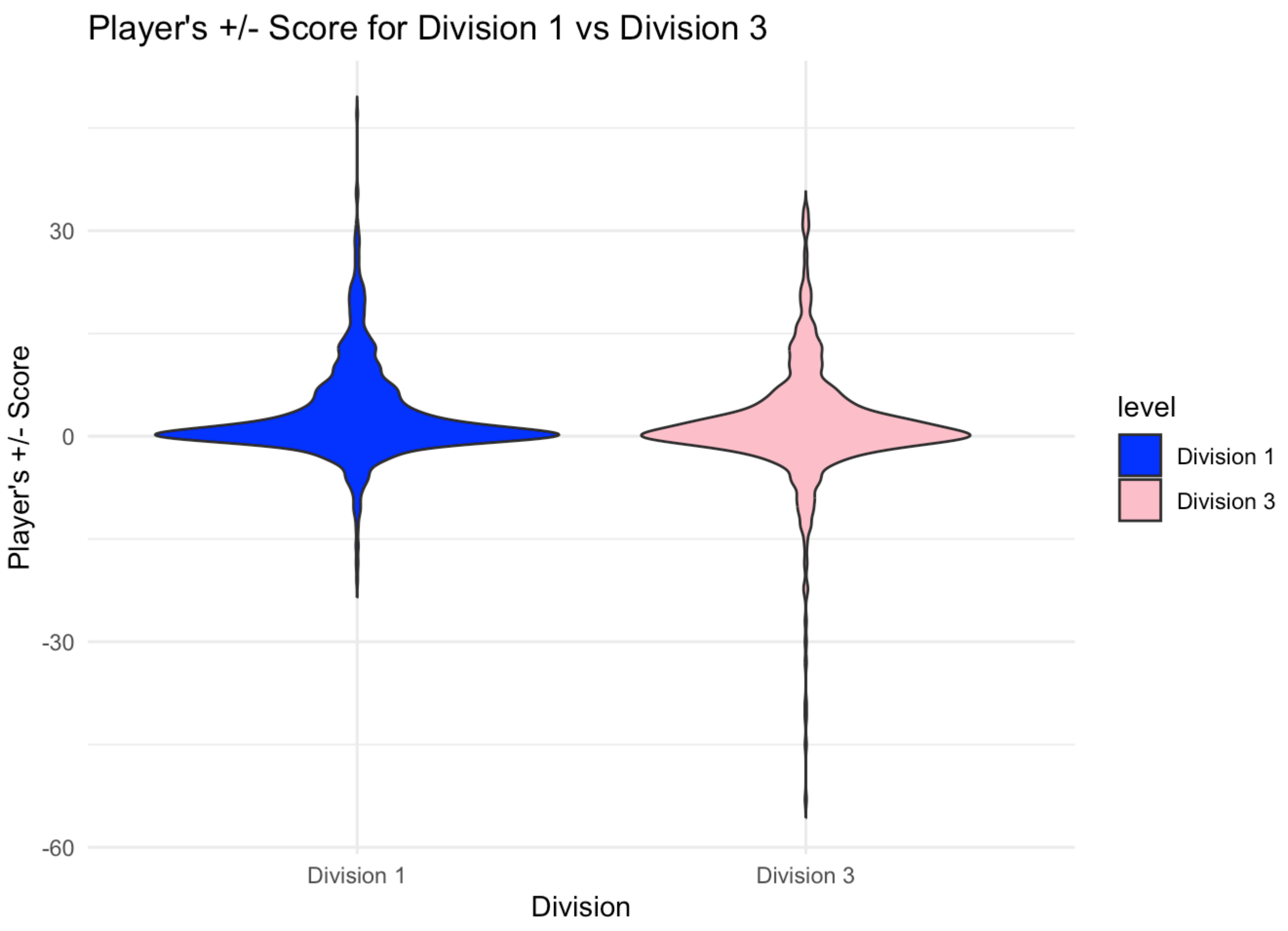
```r
# Filter table to create a new table called d3 that only contains d3 level pla
d3 <- ultimate_data[ultimate_data$level == "Division 3",]

# Plot  Division 3 level player's plus-minus scores and their gender
ggplot(d3, aes(x = gender, y = plus_minus, fill = gender)) +
  geom_boxplot() +
  labs(
    title = "Division 3 Gender vs. Player's +/- Score",
    x = "Gender",
    y = "Player's +/- Score"
  ) +
  theme_minimal()
```



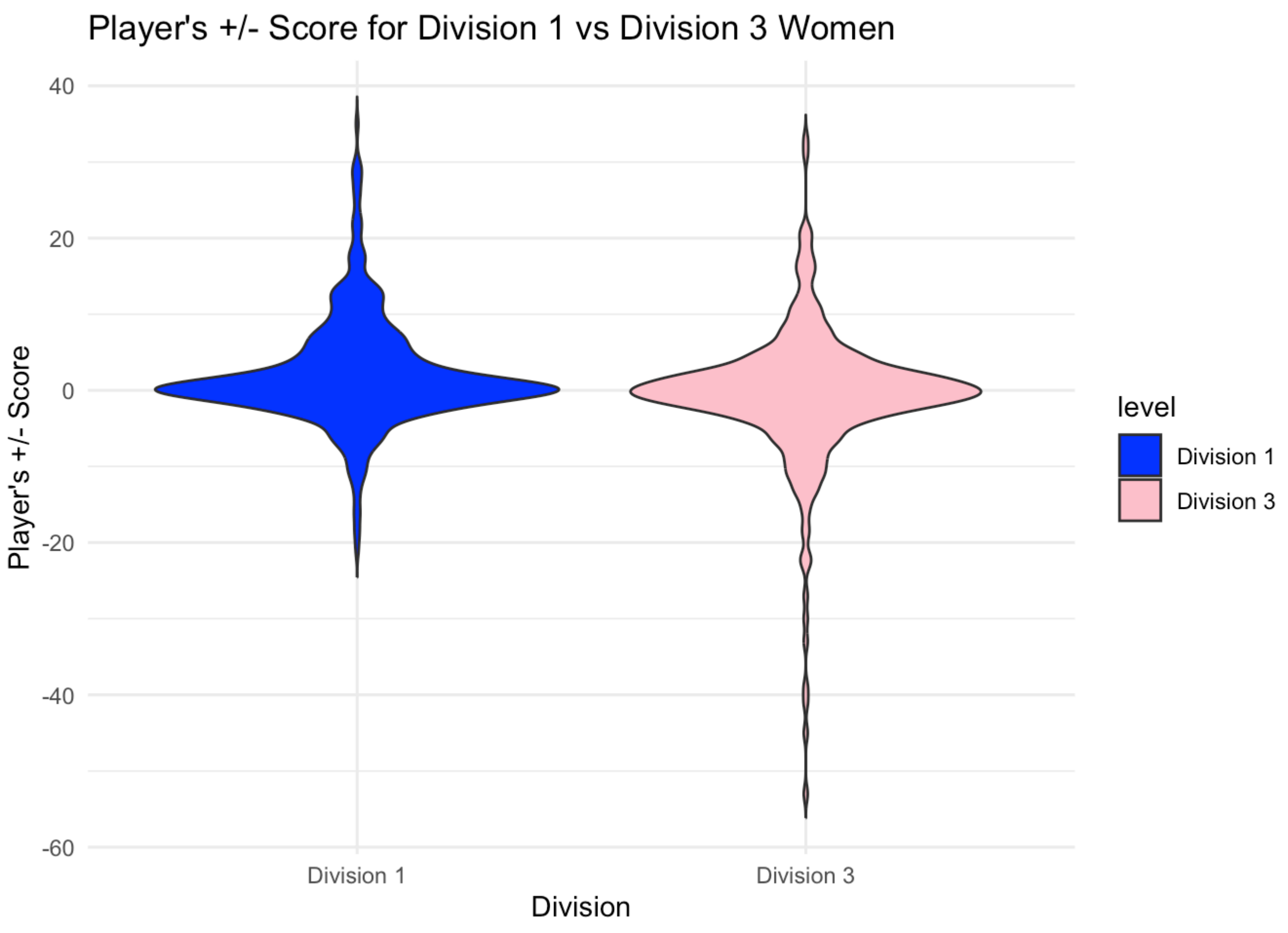
```
# Filter table to create a new table called d1 that only contains d3 level pla
d1 <- ultimate_data[ultimate_data$level == "Division 1",]

# Plot  Division 1 level player's plus-minus scores and their gender
ggplot(d1, aes(x = gender, y = plus_minus, fill = gender)) +
  geom_boxplot() +
  labs(
    title = "Division 1 Gender vs. Player's +/- Score",
    x = "Gender",
    y = "Player's +/- Score"
  ) +
  theme_minimal()
```



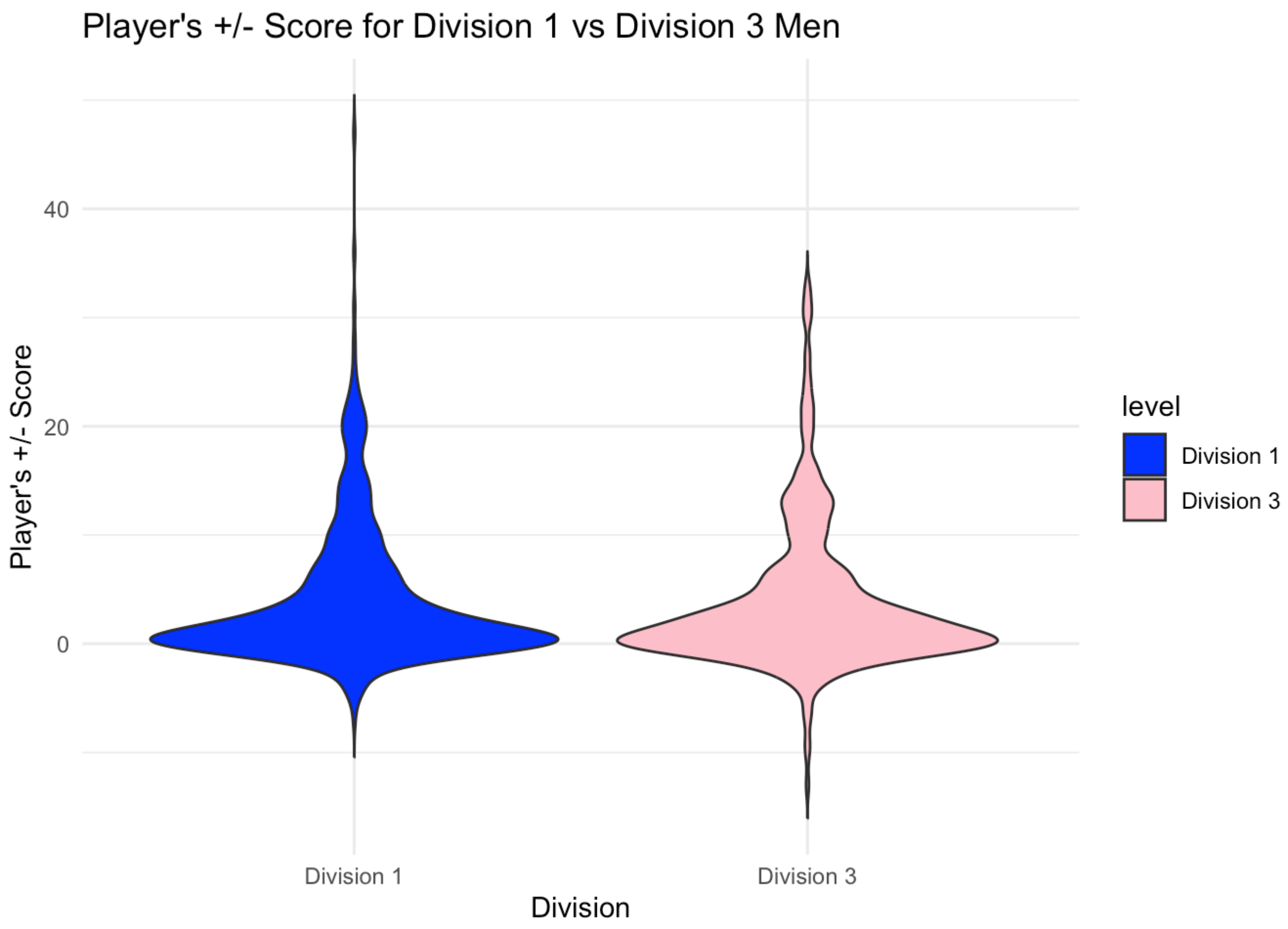
```
# Plot player's plus-minus score against the divisional level they are playing
ggplot(ultimate_data, aes(x = level, y = plus_minus, fill = level)) +
  geom_violin(trim = FALSE) +
  labs(
    title = "Player's +/- Score for Division 1 vs Division 3",
    x = "Division",
    y = "Player's +/- Score"
  ) +
  scale_fill_manual(values = c("Division 1" = "blue", "Division 3" = "pink"))
  theme_minimal()
```



```r
# Filter table to create a new table called women_ultimate_data that only cont
women_ultimate_data <- ultimate_data %>%
  filter(gender == "Women")

# Plot female player's plus-minus score against the divisional level they are
ggplot(women_ultimate_data, aes(x = level, y = plus_minus, fill = level)) +
  geom_violin(trim = FALSE) +
  labs(
    title = "Player's +/- Score for Division 1 vs Division 3 Women",
    x = "Division",
    y = "Player's +/- Score"
  ) +
  scale_fill_manual(values = c("Division 1" = "blue", "Division 3" = "pink"))
  theme_minimal()
```

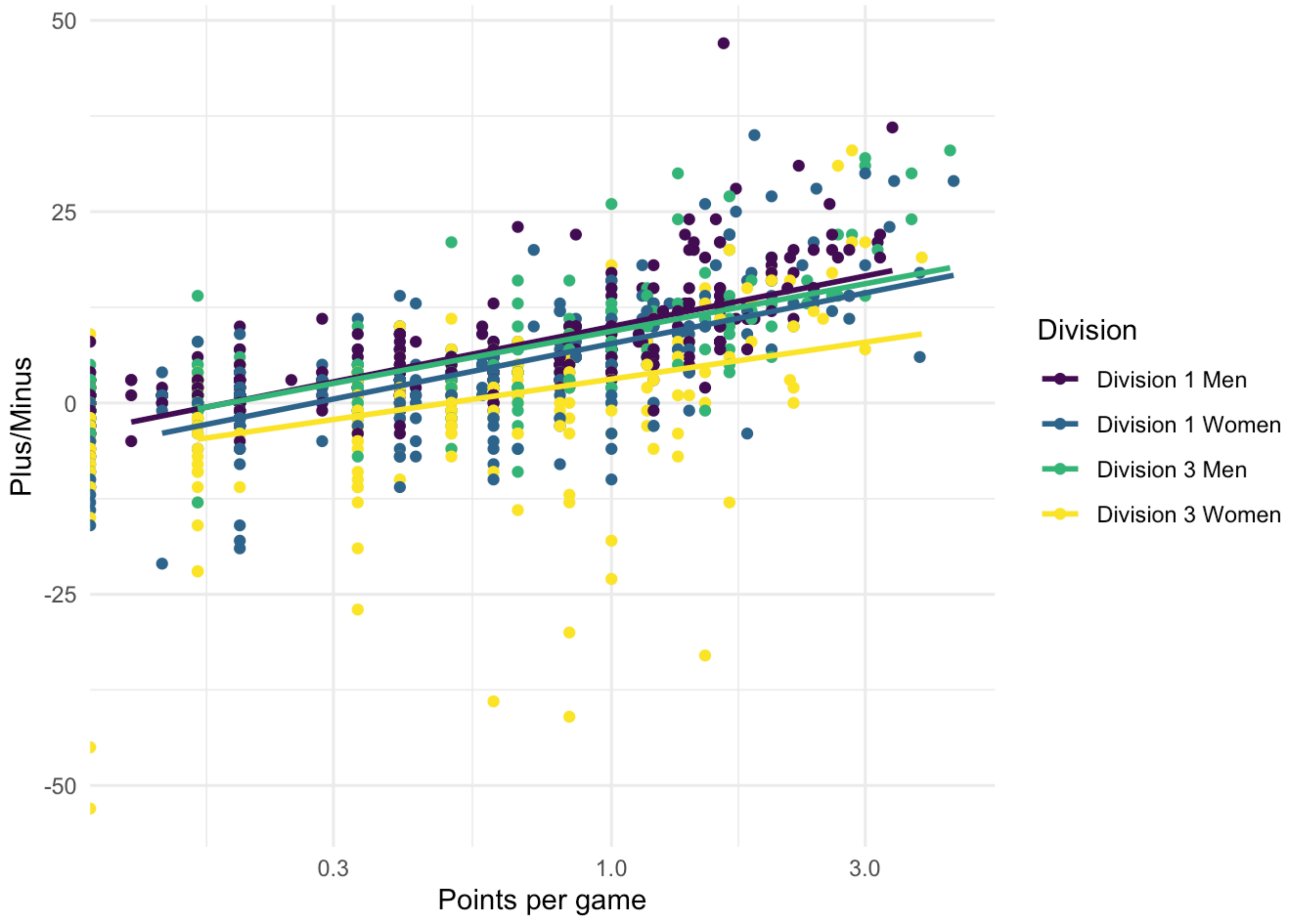


```
# Filter table to create a new table called men_ultimate_data that only contai
men_ultimate_data <- ultimate_data %>%
  filter(gender == "Men")

# Plot male player's plus-minus score against the divisional level they are pl
ggplot(men_ultimate_data, aes(x = level, y = plus_minus, fill = level)) +
  geom_violin(trim = FALSE) +
  labs(
    title = "Player's +/- Score for Division 1 vs Division 3 Men",
    x = "Division",
    y = "Player's +/- Score"
  ) +
  scale_fill_manual(values = c("Division 1" = "blue", "Division 3" = "pink"))
  theme_minimal()
```



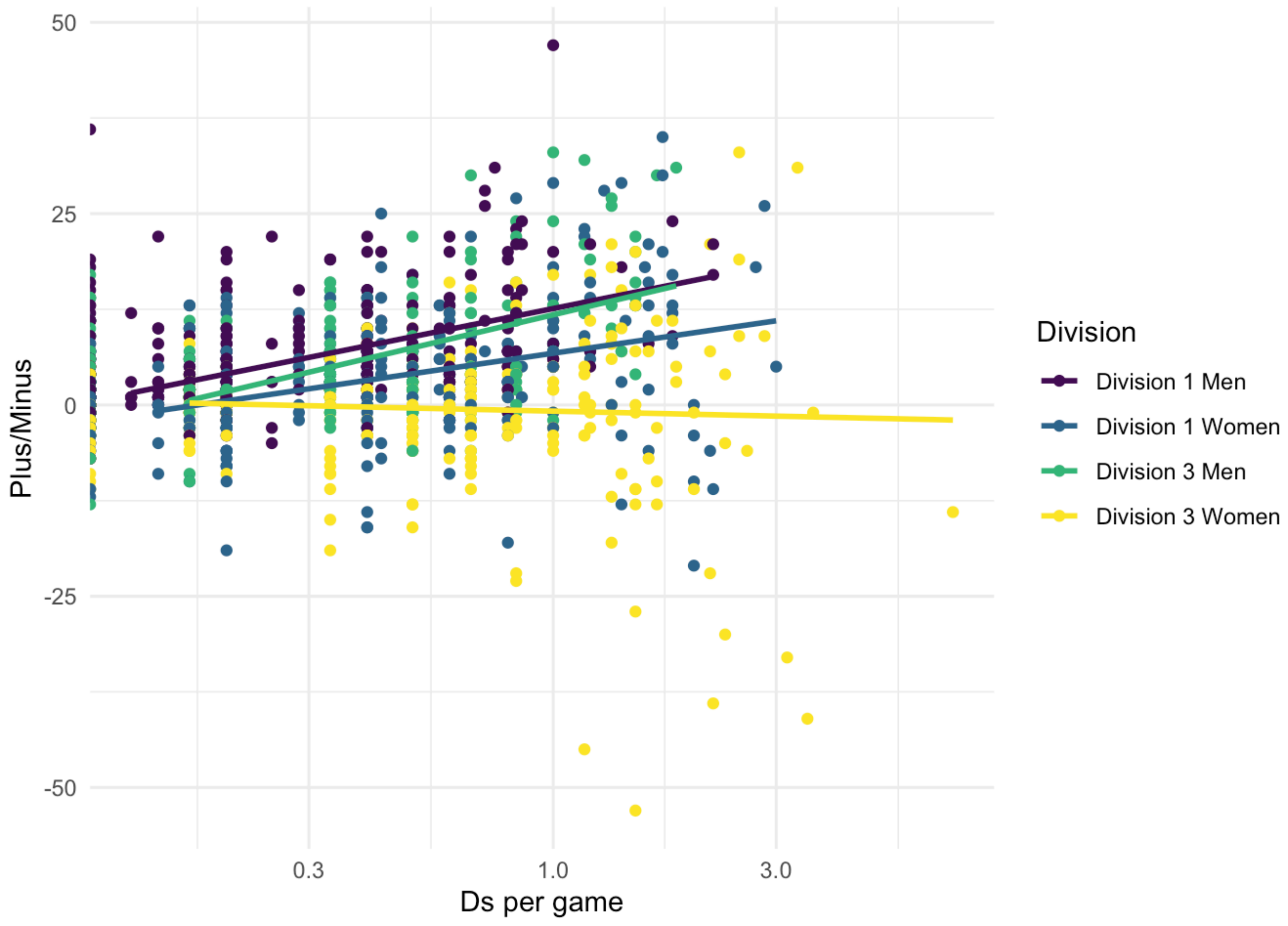
```
# Plot a player's plus-minus score and their points per game for the different
ultimate_data %>% ggplot(aes(x = pts_per_game, y = plus_minus, color = divisio
   geom_point() + geom_smooth(method = 'lm', se = F) + scale_x_log10() +
  labs(x = "Points per game", y = "Plus/Minus", color = "Division") +
  theme_minimal() +
  scale_color_viridis_d()
```

```
Warning in scale_x_log10(): log-10 transformation introduced infinite values.
log-10 transformation introduced infinite values.

`geom_smooth()` using formula = 'y ~ x'

Warning: Removed 550 rows containing non-finite outside the scale range
(`stat_smooth()`).
```



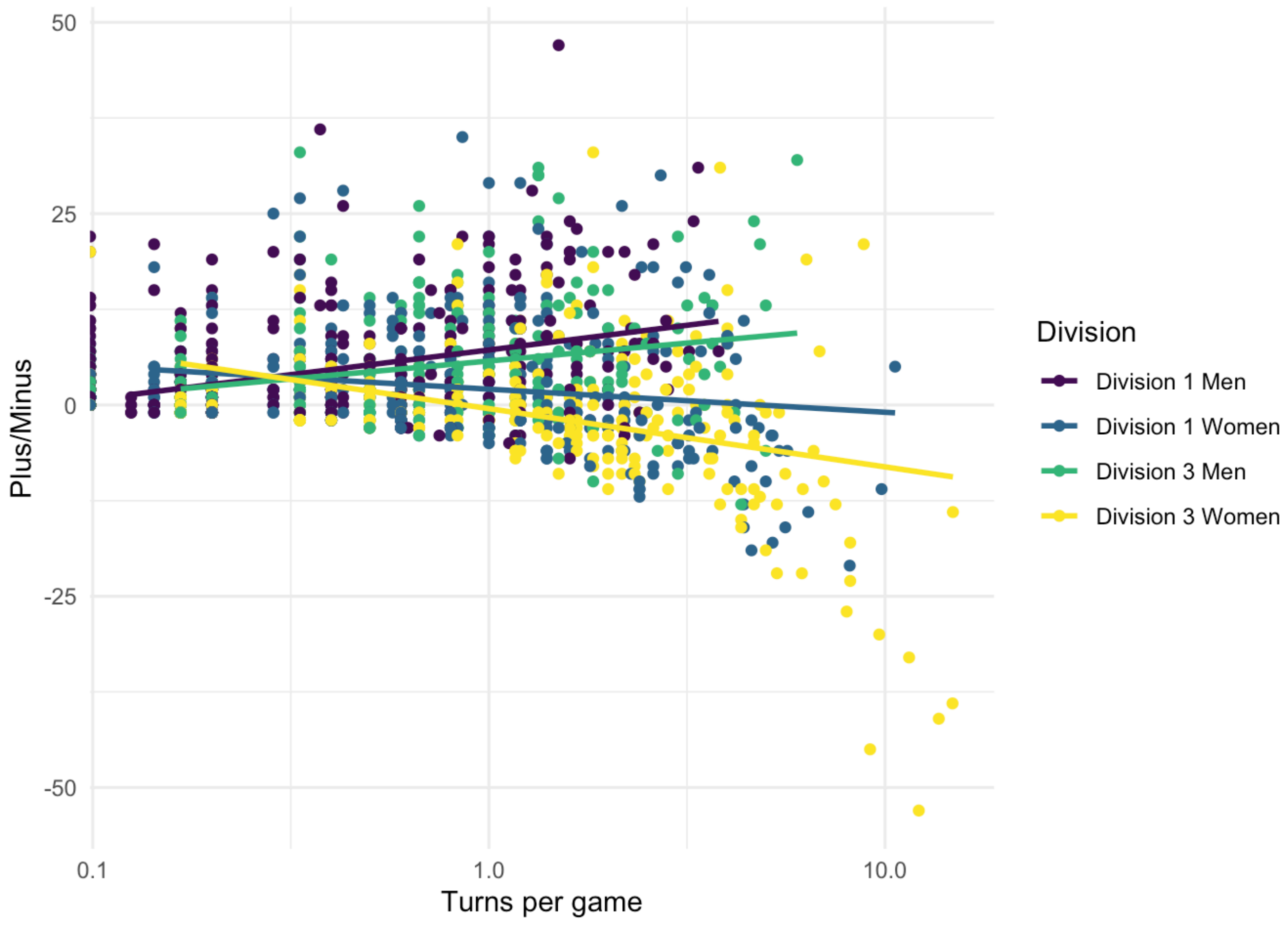
```
# Plot a player's plus-minus score and their D's per game for the different di
ultimate_data %>% ggplot(aes(x = ds_per_game, y = plus_minus, color = divisior
   geom_point() + geom_smooth(method = 'lm', se = F)+
  theme_minimal() + scale_x_log10() +
  labs(x = "Ds per game", y = "Plus/Minus", color = "Division") +
  scale_color_viridis_d()
```

```
Warning in scale_x_log10(): log-10 transformation introduced infinite values.
log-10 transformation introduced infinite values.

`geom_smooth()` using formula = 'y ~ x'

Warning: Removed 638 rows containing non-finite outside the scale range
(`stat_smooth()`).
```



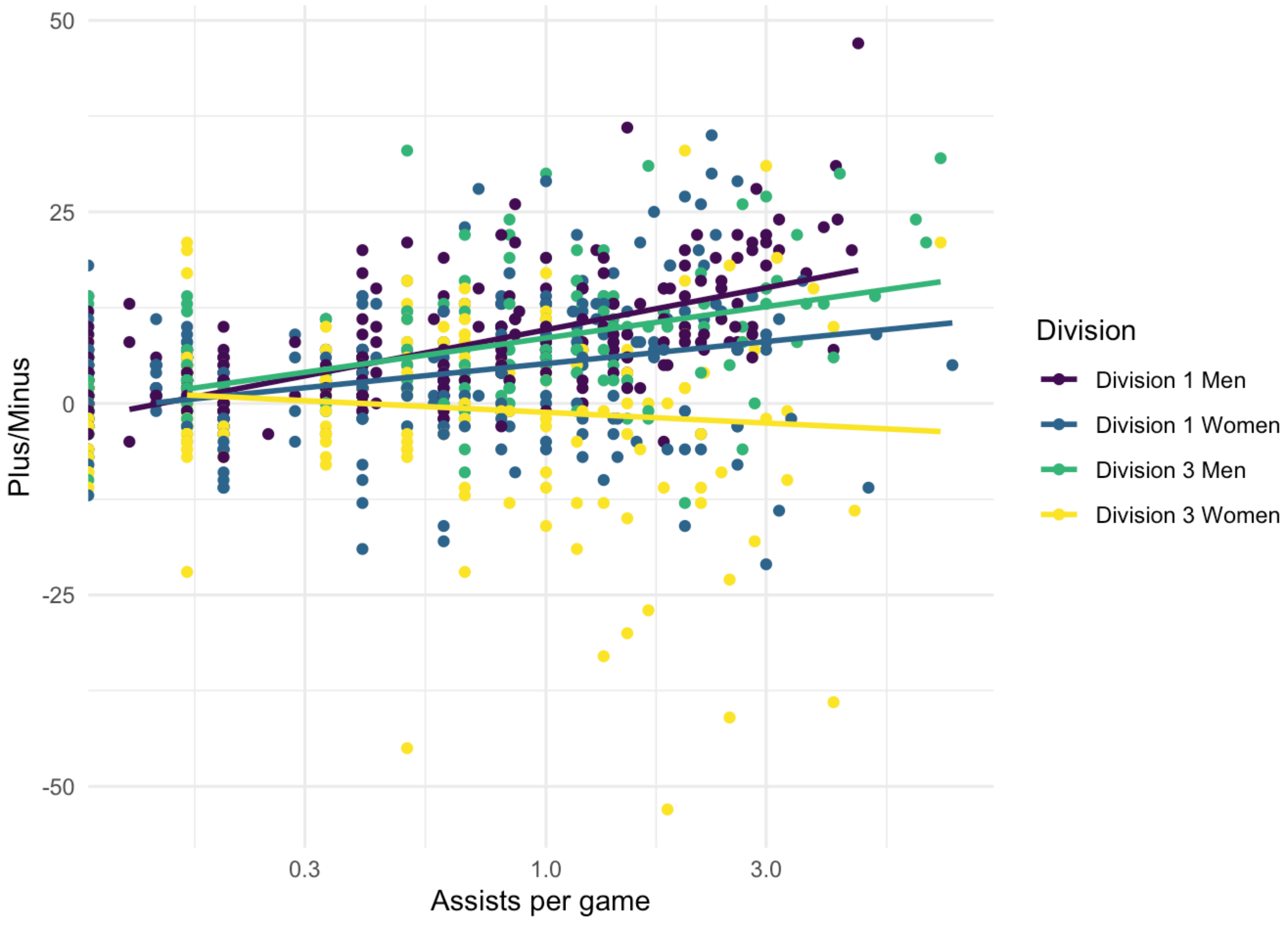
```
# Plot a player's plus-minus score and their turns per game for the different
ultimate_data %>% ggplot(aes(x = turns_per_game, y = plus_minus, color = divis
   geom_point() + geom_smooth(method = 'lm', se = F) + scale_x_log10() +
  labs(x = "Turns per game", y = "Plus/Minus", color = "Division") + theme_mir
  scale_color_viridis_d()
```

```
Warning in scale_x_log10(): log-10 transformation introduced infinite values.
log-10 transformation introduced infinite values.

`geom_smooth()` using formula = 'y ~ x'

Warning: Removed 424 rows containing non-finite outside the scale range
(`stat_smooth()`).
```

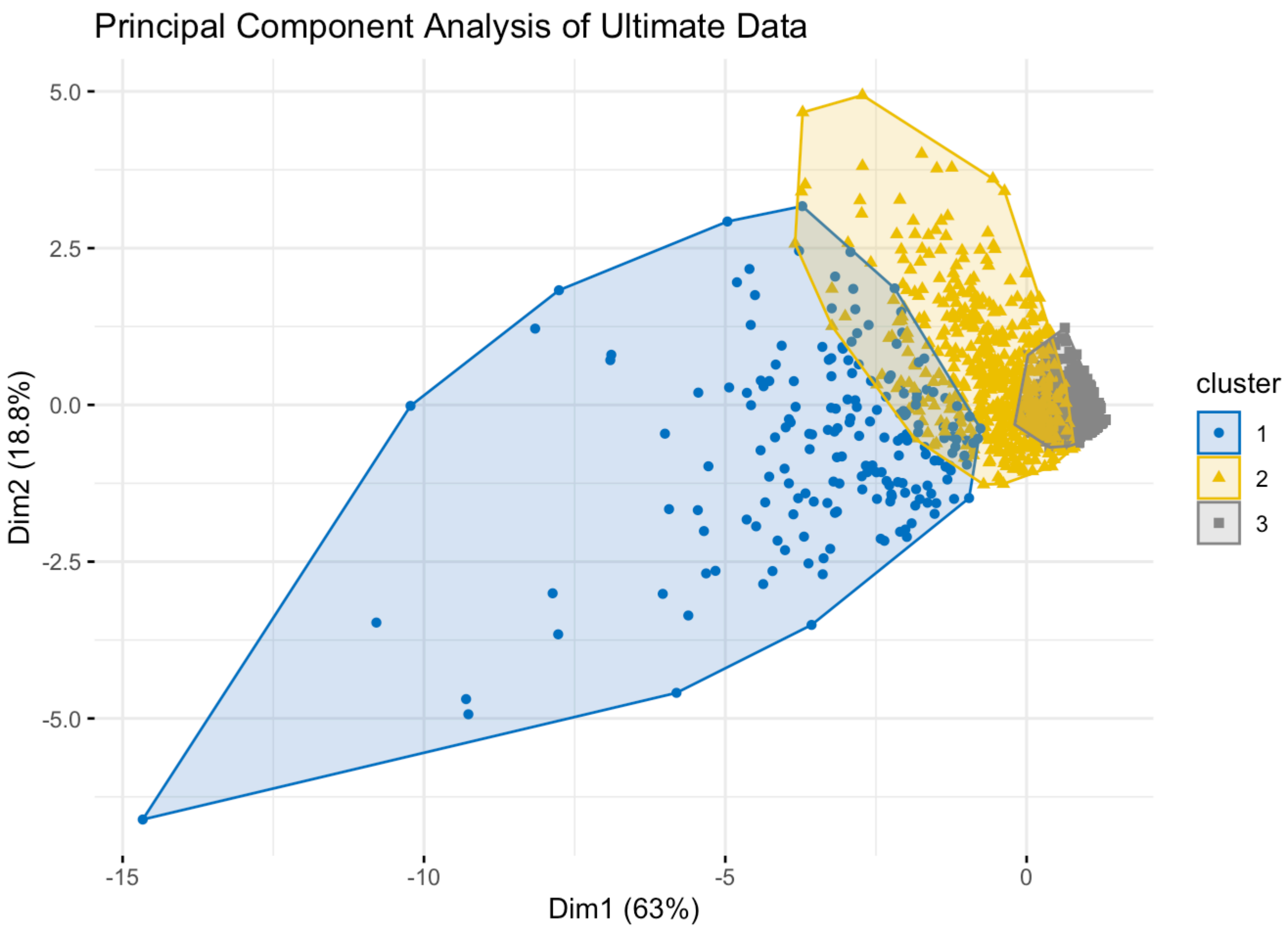


```
# Plot a player's plus-minus score and their assists per game for the differen
ultimate_data %>% ggplot(aes(x = ast_per_game, y = plus_minus, color = divisio
   geom_point() + geom_smooth(method = 'lm', se = F) + scale_x_log10() +
  labs(x = "Assists per game", y = "Plus/Minus", color = "Division") + theme_m
  scale_color_viridis_d()
```

```
Warning in scale_x_log10(): log-10 transformation introduced infinite values.
log-10 transformation introduced infinite values.

`geom_smooth()` using formula = 'y ~ x'

Warning: Removed 743 rows containing non-finite outside the scale range
(`stat_smooth()`).
```



```r
# Create a dataset df1 by selecting the following the columns below
df1 <- ultimate_data %>% select(c(
  turns_per_game, ds_per_game, pts_per_game, pls_mns_per_game, ast_per_game
))

# Perform principle component analysis on df1
pam_res <- pam(df1[-4], 3)

# Plots the three clusters
fviz_cluster(pam_res, data = df1[-4],
             geom = "point",
             ellipse.type = "convex",
             palette = "jco",
             ggtheme = theme_minimal()) +
  labs(title = "Principal Component Analysis of Ultimate Data")
```



We are fitting all three models that predict continuous variables: linear, LASSO, and ridge regression. Based on the results, we will select the model that is most accurate. Additionally, although the EDA for a player's +/- score seems similar for division and gender, there are slight differences; therefore, we will fit the model for each combination of division and gender.

Results

```
ultimate_data = r.ultimate_data

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.exceptions import DataConversionWarning
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import make_column_transformer
from sklearn.model_selection import cross_validate
from sklearn.model_selection import GroupKFold
from sklearn.linear_model import LinearRegression, Ridge, RidgeCV
from sklearn.linear_model import ElasticNet, ElasticNetCV
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_absolute_error, mean_squared_error

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

Linear Regression

Function to Fit Model to Complete Data Set

```
def linear_reg(ultimate_data, test_size=0.2, random_state=42):
    feature_cols = ['turns_per_game', 'ds_per_game', 'ast_per_game', 'pts_per_
    target_col = 'pls_mns_per_game'

    data_clean = ultimate_data[feature_cols + [target_col]].dropna()

    X = data_clean[feature_cols]
    y = data_clean[target_col]

    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size

    model = LinearRegression()
    model.fit(X_train, y_train)

    score = model.score(X_test, y_test)
    coef = dict(zip(feature_cols, model.coef_))
    intercept = model.intercept_

    return {
        'model': model,
        'score (R^2)': score,
        'coefficients': coef,
        'intercept': intercept,
        'scaler': scaler
    }

results_linear = linear_reg(ultimate_data)


print("R² Score:", results_linear['score (R^2)'])
```

```
R² Score: 1.0
```

```
print("Coefficients:", results_linear['coefficients'])
```

```
Coefficients: {'turns_per_game': np.float64(-1.4551043880863068), 'ds_per_game':
np.float64(0.5143738492398979), 'ast_per_game': np.float64(0.880593153659139),
'pts_per_game': np.float64(0.6703769963024492)}
```

```
print("Intercept:", results_linear['intercept'])
```

```
Intercept: 0.44304876304331375
```

Function to Fit Model by Division and Gender

```
def linear_by_div_gender(ultimate_data, test_size=0.2, random_state=42):
    feature_cols = ['turns_per_game', 'ds_per_game', 'ast_per_game', 'pts_per_
    target_col = 'pls_mns_per_game'

    results_linear = {}

    for (division, gender), group in ultimate_data.groupby(['level', 'gender']
        group_clean = group[feature_cols + [target_col]].dropna()

        if len(group_clean) < 5:
            continue

        X = group_clean[feature_cols]
        y = group_clean[target_col]

        scaler = StandardScaler()
        X_scaled = scaler.fit_transform(X)

        X_train, X_test, y_train, y_test = train_test_split(
            X_scaled, y, test_size=test_size, random_state=random_state
        )

        model = LinearRegression()
        model.fit(X_train, y_train)

        results_linear[(division, gender)] = {
            'model': model,
            'score (R^2)': model.score(X_test, y_test),
            'coefficients': dict(zip(feature_cols, model.coef_)),
            'intercept': model.intercept_,
            'scaler': scaler,
            'n_samples': len(group_clean)
        }

    return results_linear


resultsdg_linear = linear_by_div_gender(ultimate_data)

for (level, gender), res in resultsdg_linear.items():
    print(f"{level} {gender} — R²: {res['score (R^2)']:.3f}")
    for feat, coef in res['coefficients'].items():
        print(f"    {feat}: {coef:.3f}")
    print()
```

```
Division 1 Men — R²: 1.000
    turns_per_game: -0.647
    ds_per_game: 0.309
    ast_per_game: 0.816
    pts_per_game: 0.658

Division 1 Women — R²: 1.000
    turns_per_game: -1.383
    ds_per_game: 0.498
    ast_per_game: 0.849
    pts_per_game: 0.671

Division 3 Men — R²: 1.000
    turns_per_game: -0.962
    ds_per_game: 0.366
    ast_per_game: 0.989
    pts_per_game: 0.709

Division 3 Women — R²: 1.000
    turns_per_game: -2.328
    ds_per_game: 0.770
    ast_per_game: 0.890
    pts_per_game: 0.643
```

Ridge Regression

Function to Fit Model to Complete Data Set

```
def ridge_reg(ultimate_data, alpha=1.0, test_size=0.2, random_state=42):
    # Define features and target
    feature_cols = ['turns_per_game', 'ds_per_game', 'ast_per_game', 'pts_per_
    target_col = 'pls_mns_per_game'

    # Drop rows with missing values
    data_clean = ultimate_data[feature_cols + [target_col]].dropna()

    # Separate X and y
    X = data_clean[feature_cols]
    y = data_clean[target_col]

    # Standardize features
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Train/test split
    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size

    # Train Ridge Regression
    model = Ridge(alpha=alpha)
    model.fit(X_train, y_train)

    # Output
    score = model.score(X_test, y_test)
    coef = dict(zip(feature_cols, model.coef_))
    intercept = model.intercept_

    return {
        'model': model,
        'score (R^2)': score,
        'coefficients': coef,
        'intercept': intercept,
        'scaler': scaler
    }
```

```
results = ridge_reg(ultimate_data)

print("R² Score:", results['score (R^2)'])
```

```
R² Score: 0.999996195907342
```

```
print("Coefficients:", results['coefficients'])
```

```
Coefficients: {'turns_per_game': np.float64(-1.4513537992217975), 'ds_per_game':
np.float64(0.5125036949559031), 'ast_per_game': np.float64(0.8782533350986946),
'pts_per_game': np.float64(0.670704220022051)}
```

```
print("Intercept:", results['intercept'])
```

```
Intercept: 0.4430078472816218
```

Function to Fit Model by Division and Gender

```
def ridge_by_div_gender(ultimate_data, alpha=1.0, test_size=0.2, random_state=
    feature_cols = ['turns_per_game', 'ds_per_game', 'ast_per_game', 'pts_per_
    target_col = 'pls_mns_per_game'

    results = {}

    # Group by actual column values
    for (division, gender), group in ultimate_data.groupby(['level', 'gender']
        group_clean = group[feature_cols + [target_col]].dropna()

        if len(group_clean) < 5:
            continue  # Skip small groups

        X = group_clean[feature_cols]
        y = group_clean[target_col]

        scaler = StandardScaler()
        X_scaled = scaler.fit_transform(X)

        X_train, X_test, y_train, y_test = train_test_split(
            X_scaled, y, test_size=test_size, random_state=random_state
        )

        model = Ridge(alpha=alpha)
        model.fit(X_train, y_train)

        results[(division, gender)] = {
            'model': model,
            'score (R^2)': model.score(X_test, y_test),
            'coefficients': dict(zip(feature_cols, model.coef_)),
            'intercept': model.intercept_,
            'scaler': scaler,
            'n_samples': len(group_clean)
        }

    return results
```

```
resultsdg = ridge_by_div_gender(ultimate_data)

for (level, gender), res in resultsdg.items():
    print(f"{level} {gender} — R²: {res['score (R^2)']:.3f}")
    for feat, coef in res['coefficients'].items():
        print(f"    {feat}: {coef:.3f}")
    print()
```

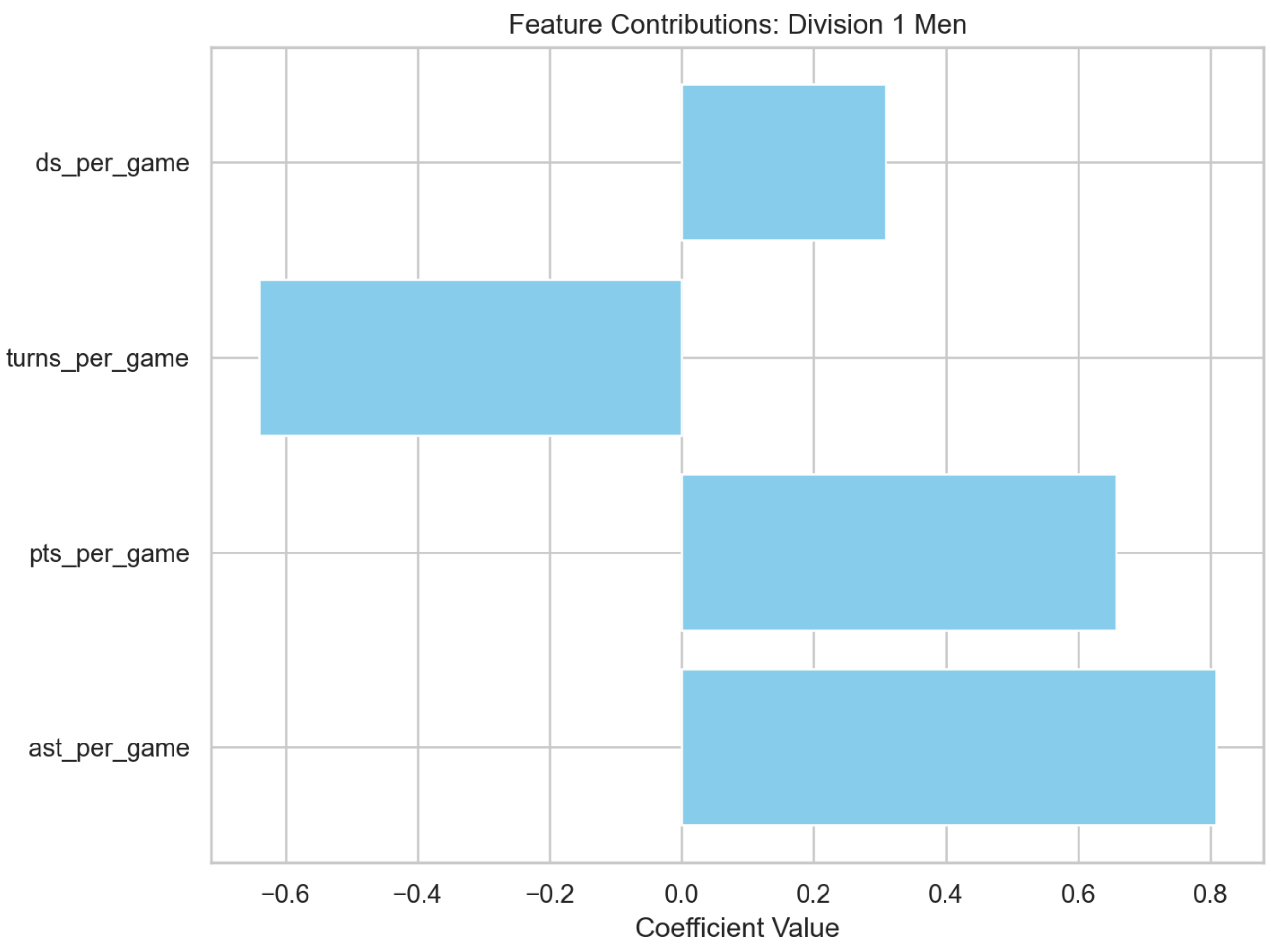
```
Division 1 Men — R²: 1.000
    turns_per_game: -0.639
    ds_per_game: 0.308
    ast_per_game: 0.809
    pts_per_game: 0.657

Division 1 Women — R²: 1.000
    turns_per_game: -1.367
    ds_per_game: 0.493
    ast_per_game: 0.837
    pts_per_game: 0.672

Division 3 Men — R²: 1.000
    turns_per_game: -0.942
    ds_per_game: 0.364
    ast_per_game: 0.971
    pts_per_game: 0.706

Division 3 Women — R²: 1.000
    turns_per_game: -2.293
    ds_per_game: 0.754
    ast_per_game: 0.867
    pts_per_game: 0.646
```

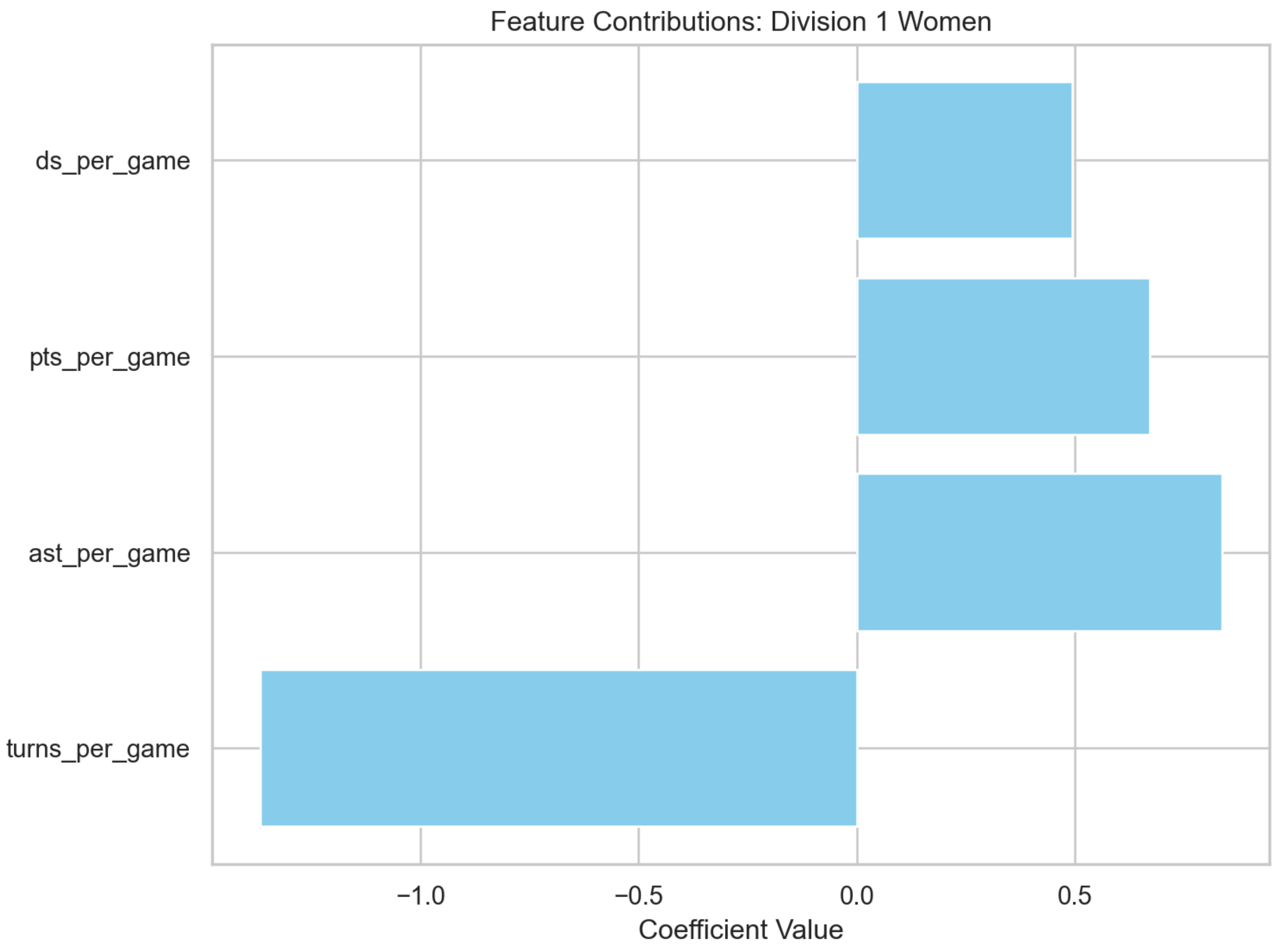
Plotting Feature Contributions

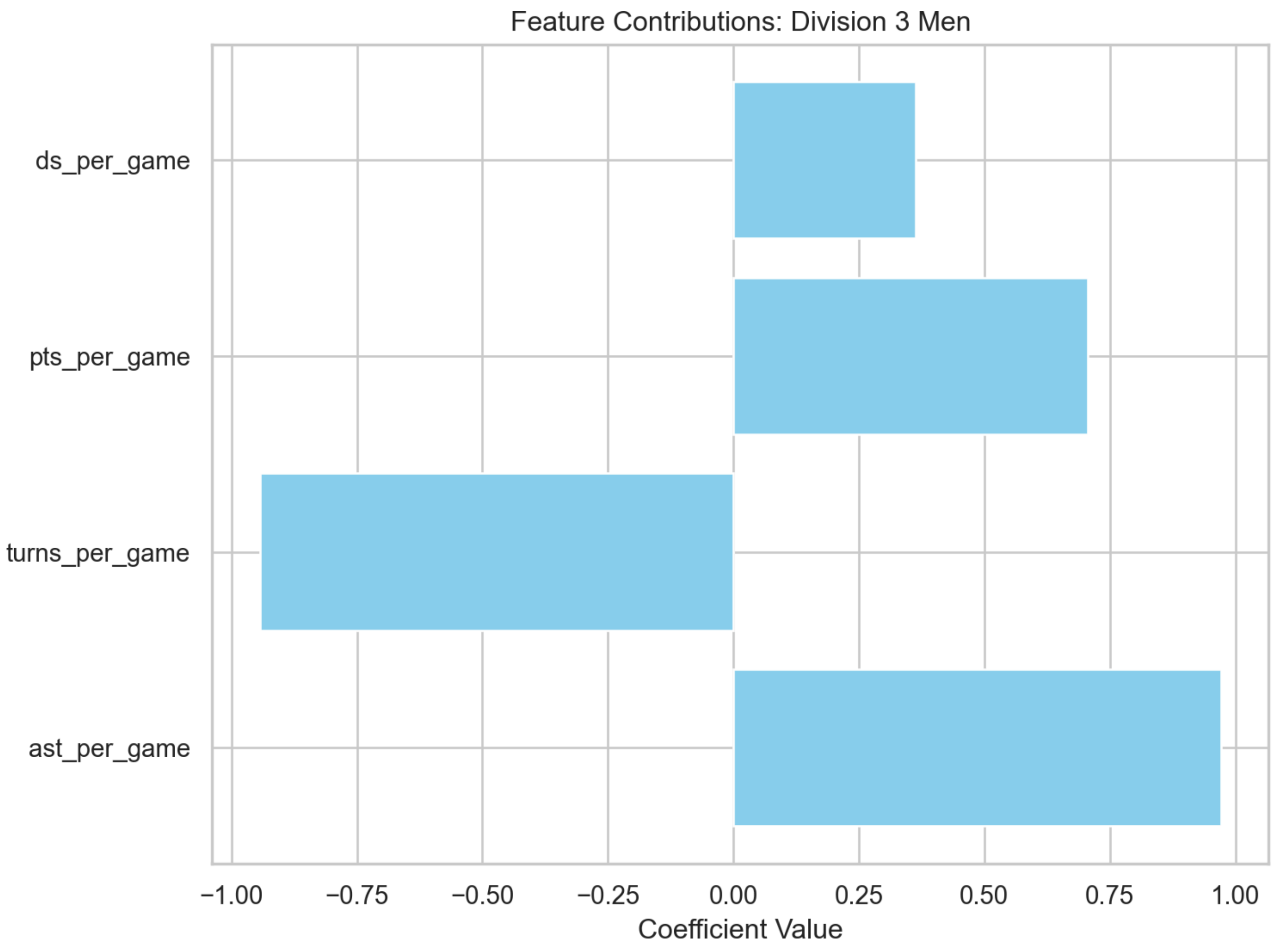
```
def plot_feature_contributions(results):

    sns.set(style="whitegrid")

    # For each group (level, gender), plot the feature coefficients
    for (level, gender), res in results.items():
        coefs = res['coefficients']

        # Sort features by absolute coefficient values
        sorted_feats = sorted(coefs.items(), key=lambda x: abs(x[1]), reverse=

        # Prepare data for plotting
        features = [feat for feat, _ in sorted_feats]
```

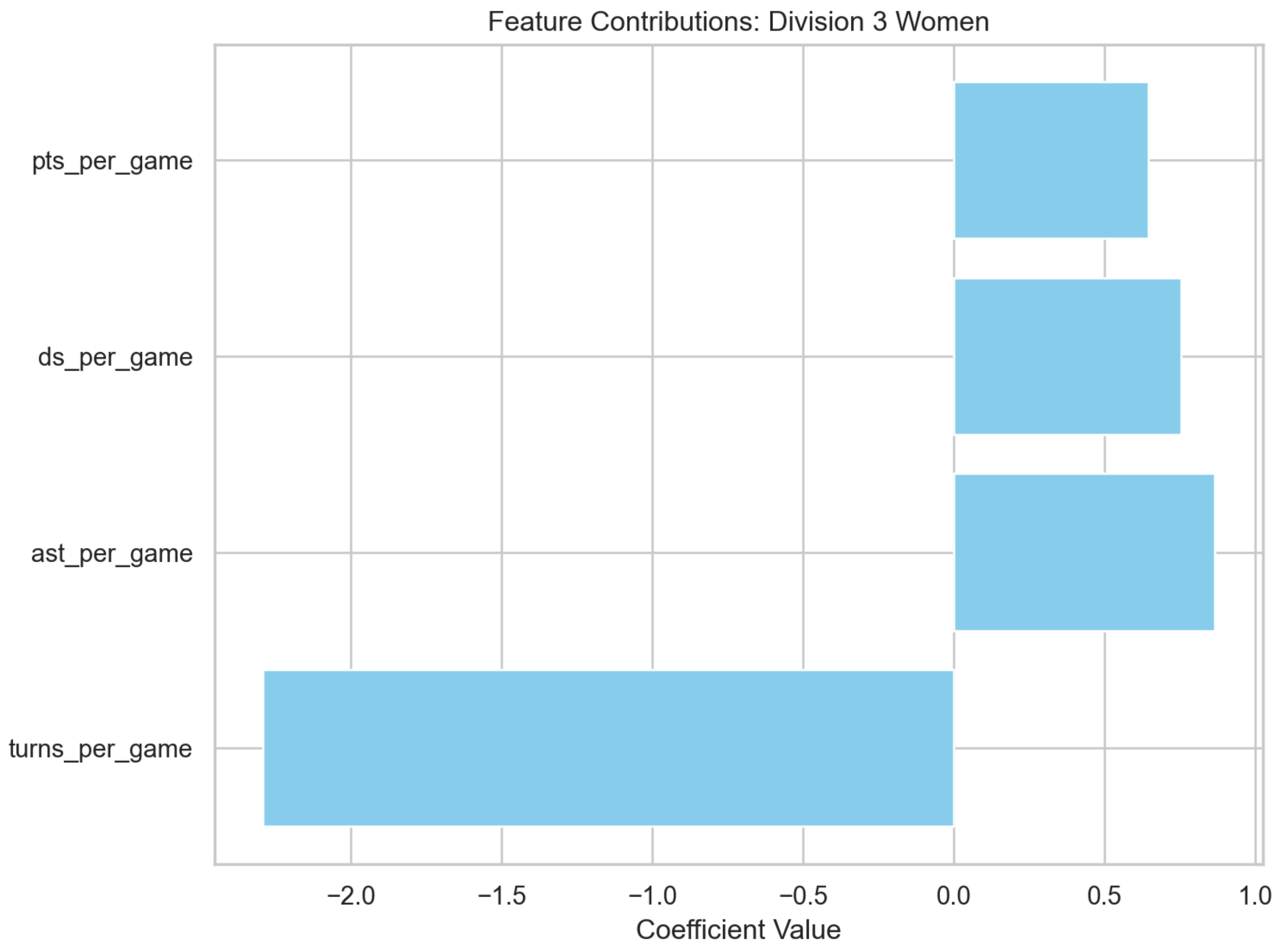
```
                coef_values = [coef for _, coef in sorted_feats]

                # Plot the bar chart
                plt.figure(figsize=(8, 6))
                plt.barh(features, coef_values, color="skyblue")
                plt.xlabel("Coefficient Value")
                plt.title(f"Feature Contributions: {level} {gender}")
                plt.tight_layout()
                plt.show()
```

```
        plot_feature_contributions(resultsdg)
```




Feature Contributions: Division 1 Women
ds_per_game
pts_per_game
ast_per_game
turns_per_game
−1.0
−0.5
0.0
0.5
Coefficient Value


Feature Contributions: Division 3 Men
ds_per_game
pts_per_game
turns_per_game
ast_per_game
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
Coefficient Value



Lasso Regression

Function to Fit Model to Complete Data Set

```
def lasso_reg(ultimate_data, alpha=1.0, test_size=0.2, random_state=56):
    # Define features and target
    feature_cols = ['turns_per_game', 'ds_per_game', 'ast_per_game', 'pts_per_
    target_col = 'pls_mns_per_game'

    # Drop rows with missing values
    data_clean = ultimate_data[feature_cols + [target_col]].dropna()

    # Separate X and y
    X = data_clean[feature_cols]
    y = data_clean[target_col]

    # Standardize features
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Train/test split
    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size

    # Train Ridge Regression
    model = Lasso(alpha=alpha)
```

```
            model.fit(X_train, y_train)

            # Make predictions
            y_pred = model.predict(X_test)

            # Output
            score = model.score(X_test, y_test)
            mae = mean_absolute_error(y_test, y_pred)
            rmse = mean_squared_error(y_test, y_pred) ** 0.5
            coef = dict(zip(feature_cols, model.coef_))
            intercept = model.intercept_

            return {
                'model': model,
                'score (R^2)': score,
                'MAE': mae,
                'RMSE': rmse,
                'coefficients': coef,
                'intercept': intercept,
                'scaler': scaler
            }
```

```
        lasso_results = lasso_reg(ultimate_data)

        print("R² Score:", lasso_results['score (R^2)'])
```

```
R² Score: -0.002484439831602625
```

```
        print("Coefficients:", lasso_results['coefficients'])
```

```
Coefficients: {'turns_per_game': np.float64(-0.0), 'ds_per_game': np.float64(0.0),
'ast_per_game': np.float64(0.0), 'pts_per_game': np.float64(0.0)}
```

```
        print("Intercept:", lasso_results['intercept'])
```

```
Intercept: 0.43019179894069065
```

Function grouped by division and gender

```
        def lasso_by_div_gender(ultimate_data, alpha=1.0, test_size=0.2, random_state=
            feature_cols = ['turns_per_game', 'ds_per_game', 'ast_per_game', 'pts_per_
            target_col = 'pls_mns_per_game'

            results = {}

            # Group by actual column values
            for (division, gender), group in ultimate_data.groupby(['level', 'gender']
                group_clean = group[feature_cols + [target_col]].dropna()

                if len(group_clean) < 5:
```

```
                continue

            X = group_clean[feature_cols]
            y = group_clean[target_col]

            scaler = StandardScaler()
            X_scaled = scaler.fit_transform(X)

            X_train, X_test, y_train, y_test = train_test_split(
                X_scaled, y, test_size=test_size, random_state=random_state
            )

            model = Lasso(alpha=alpha)
            model.fit(X_train, y_train)

            results[(division, gender)] = {
                'model': model,
                'score (R^2)': model.score(X_test, y_test),
                'coefficients': dict(zip(feature_cols, model.coef_)),
                'intercept': model.intercept_,
                'scaler': scaler,
                'n_samples': len(group_clean)
            }

        return results
```

```
resultsdg = lasso_by_div_gender(ultimate_data)

for (level, gender), res in resultsdg.items():
    print(f"{level} {gender} — R²: {res['score (R^2)']:.3f}")
    for feat, coef in res['coefficients'].items():
        print(f"    {feat}: {coef:.3f}")
    print()
```

```
Division 1 Men — R²: -0.019
    turns_per_game: 0.000
    ds_per_game: 0.000
    ast_per_game: 0.000
    pts_per_game: 0.000

Division 1 Women — R²: -0.000
    turns_per_game: -0.000
    ds_per_game: 0.000
    ast_per_game: 0.000
    pts_per_game: 0.000

Division 3 Men — R²: -0.040
    turns_per_game: 0.000
    ds_per_game: 0.000
    ast_per_game: 0.000
    pts_per_game: 0.000

Division 3 Women — R²: -0.063
```

```
turns_per_game: -0.103
ds_per_game: -0.000
ast_per_game: -0.000
pts_per_game: 0.000
```

Prediction Model

```
def predict_plus_minus(ultimate_data, level, gender, player_stats, ridge_resul
    feature_cols = ['turns_per_game', 'ds_per_game', 'ast_per_game', 'pts_per_

    if (level, gender) not in ridge_results:
        raise ValueError(f"No Ridge model available for {level} {gender}")
    if (level, gender) not in linear_results:
        raise ValueError(f"No Linear model available for {level} {gender}")
    if (level, gender) not in lasso_results:
        raise ValueError(f"No Lasso model available for {level} {gender}")

    ridge_model = ridge_results[(level, gender)]['model']
    ridge_scaler = ridge_results[(level, gender)]['scaler']

    linear_model = linear_results[(level, gender)]['model']
    linear_scaler = linear_results[(level, gender)]['scaler']

    lasso_model = lasso_results[(level, gender)]['model']
    lasso_scaler = lasso_results[(level, gender)]['scaler']

    player_df = pd.DataFrame([player_stats], columns=feature_cols)

    ridge_scaled = ridge_scaler.transform(player_df)
    linear_scaled = linear_scaler.transform(player_df)
    lasso_scaled = lasso_scaler.transform(player_df)

    predicted_ridge = ridge_model.predict(ridge_scaled)[0]
    predicted_linear = linear_model.predict(linear_scaled)[0]
    predicted_lasso = lasso_model.predict(lasso_scaled)[0]

    return {
        'ridge': predicted_ridge,
        'linear': predicted_linear,
        'lasso': predicted_lasso
    }
```

Example Use of Prediction Model - Using Filler D1 Men Stats

```
player_stats = {
    'turns_per_game': 2.1,
    'ds_per_game': 1.8,
    'ast_per_game': 3.2,
    'pts_per_game': 5.4}

# Get the results from Ridge regression (run the model first)
ridge_results = ridge_by_div_gender(ultimate_data)
linear_results = linear_by_div_gender(ultimate_data)
```

```
        lasso_results = lasso_by_div_gender(ultimate_data)


        # Predict for Division 1 Men
        predicted_plus_minus_d1_men = predict_plus_minus(ultimate_data, 'Division 1',

        print("Predicted +/- score per game (D1 Men):")
```

```
Predicted +/- score per game (D1 Men):
```

```
        print(f"    Ridge: {predicted_plus_minus_d1_men['ridge']:.2f}")
```

```
    Ridge: 8.28
```

```
        print(f"    Linear: {predicted_plus_minus_d1_men['linear']:.2f}")
```

```
    Linear: 8.30
```

```
        print(f"    Lasso: {predicted_plus_minus_d1_men['lasso']:.2f}")
```

```
    Lasso: 0.68
```

Showcase how you arrived at answers to your research question using the techniques we have learned in class (and beyond, if you're feeling adventurous).

Provide only the main results from your analysis. The goal is not to do an exhaustive data analysis (calculate every possible statistic and perform every possible procedure for all variables). Rather, you should demonstrate that you are proficient at asking meaningful questions and answering them using data, that you are skilled in interpreting and presenting results, and that you can accomplish these tasks using R and Python. More is not always better.

Discussion and Conclusion

In this project, we explored the feasibility of building a predictive model to estimate a player's plus-minus score in college Ultimate Frisbee based on individual game statistics: turnovers per game, defensive blocks per game, assists per game, and points per game. We implemented three distinct regression models—Linear Regression, Ridge Regression, and Lasso Regression—to evaluate how well these features could predict the target outcome.

Both **Linear** and **Ridge Regression** models produced very similar predictions across multiple gender-division combinations, suggesting that the underlying relationships in the data are fairly linear and do not require strong regularization to prevent overfitting.

In contrast, **Lasso Regression** consistently underperformed. For example, when provided with the input player_stats = { 'turns_per_game': 2.1, 'ds_per_game': 1.8, 'ast_per_game': 3.2, 'pts_per_game': 5.4 } its predicted plus-minus score was Ridge: 8.28, Linear: 8.30 and Lasso: 0.68. This significant deviation from the other two models suggests that Lasso may be overly aggressive in shrinking coefficients to zero—likely eliminating valuable predictors entirely. Given that we only have four features, the benefits of Lasso's feature selection are minimal, and its use may be unwarranted in this case.

Methodological Evaluation

To ensure consistent model evaluation, we split the dataset by both **division (D-I vs. D-III)** and **gender (men's vs. women's)**, and trained separate models for each subgroup. This decision was grounded in the understanding that the dynamics of ultimate frisbee can vary substantially across these categories, and that a model trained on pooled data might obscure those differences.

Limitations

1. **Small feature set**: Only four predictor variables were used. While these are intuitively related to performance, plus-minus is inherently a team-dependent metric influenced by many contextual factors (e.g., opponent strength, line composition, playing time) that are not captured here.
2. **Potential data imbalance**: The performance of the models across different (division, gender) groups could vary depending on how many observations we had per group. If some subgroups had fewer examples, model stability would be affected.
3. Lack of cross-validation metrics: Although we evaluated the models qualitatively through predictions, we did not include quantitative metrics such as RMSE or R^2 for each subgroup. Including these would allow for a more robust evaluation of model accuracy and generalizability.

Data Reliability and Validity

The dataset was sourced from a publicly available website and published within the past year. While not a first-party dataset, its reliability is bolstered by several key factors:

1. It reflects official statistics from the most recent USA Ultimate College National Championships, meaning it's drawn from the highest college-level competition.
2. As active Frisbee players ourselves, we were able to recognize nearly all the teams listed and even some individual players, affirming the data's authenticity and face validity.
3. The dataset has had multiple downloads from the site, indicating community engagement and a degree of scrutiny.

Appropriateness of the Analysis

Our analytical choices—using regularized regression methods and controlling for gender and division—are appropriate given our objective and dataset size. We standardized our features using `StandardScaler()` and implemented a consistent 80/20 train-test split across all models (as shown in the `lasso_reg` function), which helped avoid data leakage and ensured fair comparison between models.

Using multiple models allowed us to assess the sensitivity of plus-minus predictions to different regularization schemes. Ridge, in particular, proved effective in slightly dampening noisy relationships while preserving feature contributions. However, our choice of a fixed alpha parameter (rather than using cross-validation to tune hyperparameters) could limit model optimization and generalizability.

Ethical Considerations

Benefits:

- Enhanced Performance Insights: Our model can provide players and coaches with data-driven insights, potentially informing training and strategy.
- Objective Evaluation: By quantifying performance metrics, we aim to reduce subjective biases in player assessments.

Risks:

- Privacy Concerns: Even though our data is publicly available, using it for predictive modeling raises questions about consent and the extent to which athletes are aware of or agree to such analyses.
- Potential for Misuse: There's a risk that our model could be used to make decisions about player selection or playing time without considering the broader context of an athlete's performance or potential.
- Bias and Fairness: If our model inadvertently reflects existing biases in the data, it could perpetuate inequalities, especially if used in decision-making processes.

Overall, this project demonstrates the potential of data-driven approaches to enhance our understanding of individual performance in ultimate Frisbee. While our models show promise within the scope of elite college-level play, future work must address broader data inclusion to effective application.