USC Marshall
School of Business

Final Project Report:
Home-Team Bias in MLB Umpiring

DSO 579: Advanced Sports Performance Analytics
Spring 2025

Course Instructor:
Lorena Martin, PhD,
Assistant Professor of
Clinical Data Sciences and Operations

Group Members:
Phuong Vu
Yixin Cui
Corinne Bish

TABLE OF CONTENTS

INTRODUCTION

Background and context of the project

Baseball is a sport where the umpire's role is pivotal, and their calls directly influence the fairness and outcome of each game. An article by the Society for American Baseball Research (SABR) reports that even top-performing umpires miss 12–13 calls per game, though accuracy has improved since Major League Baseball (MLB) introduced performance evaluation systems in 2009 (Mills, 2017). A Boston University study analyzed over 4 million pitches across 11 MLB seasons and found that certain incorrect calls occurred more than 20% of the time, particularly in high-pressure situations (Williams, 2019).

It is not uncommon that fans blame the loss of the game to home advantages and umpire biases. Psychologist research shows that umpire biases exist and even extensive training cannot eliminate such biases (Flannagan & Goldstone, 2024), but how they benefit or harm the teams remain unknown. Some studies argue umpire bias relates more to player race (Tainsky, Mills & Winfree, 2013), while others suggest it directly influences team performance outcomes. Some research equate home advantages and umpire biases; however, home bias is not entirely explained by umpire, nor is it attributable to umpiring inconsistencies (Hsu, 2023). Given that, our project tends to close the gap between whether or not there is a significant home bias among umpires and how it impacts the outcomes.

Research question

Is there significant home-team bias among MLB umpiring? (favor_home and total_run_impact)

Hypotheses

- Null hypothesis: There is no significant home-team bias in MLB umpiring.
- Alternative hypothesis: There is a significant home-team bias in MLB umpiring.

Objectives and scope of the project

Our project seeks to examine the existence of home-team bias in MLB umpiring decisions and to quantify how such bias may affect the game outcome through measures like favor_home and total_run_impact. Understanding umpire bias is critical for maintaining the integrity of the sport, shaping fan perception, and potentially informing team strategies for home and away games.

Overview of the report structure

Our report is organized as follows:

- **Methodology** describes the dataset, preprocessing steps, models used, evaluation metrics, and coding approach.

- **Results & Discussion** present key findings on home-team bias with supporting visualizations, place the results in context with prior studies, address limitations, and highlight the significance of the findings.
- **Conclusion** summarizes the study's contributions, addresses the original research question, and proposes directions for future work.
- **References** list the sources cited in APA format, followed by **Appendices** containing datasets and code used in the analysis.

METHODOLOGY

Dataset description

Our dataset was collected through Kaggle and originally sourced from Umpire Scoreboards, an online platform dedicated to measuring the accuracy, consistency, and favor of MLB umpires. It includes umpire scorecard data of each game for the 2015 - 2022 MLB seasons. For each game, there are 18 columns in total, including umpire, home, away, favor_home, total_run_impact, etc.

This study focused on two columns: **favor_home** and **total_run_impact**. The favor_home column represents the difference between the home team's and the away team's run expectancy impacts for a given game. Meanwhile, the total_run_impact column represents the sum of the favor of every missed call. In addition, we examined the relationship between incorrect_calls, accuracy, consistency, umpire, home, away and favor_home and total_run_impact to see if either of these factors potentially correlate with game outcomes.

Data preprocessing

We began by cleaning the data: removing rows with missing values and converting key columns to numeric format, such as home and away team runs, pitches called, incorrect calls, and important metrics like accuracy, consistency, favor_home, and total_run_impact.

Models selection and justification

This study applied both supervised and unsupervised learning techniques to analyze potential home-team bias among MLB umpires, focusing on favor_home and total_run_impact as primary indicators.

For supervised learning, we used Regression to predict the impact of incorrect calls on total run expectancy, and Classification to determine whether an umpire's calls favor the home team. Four models were adopted as follows:

- **Simple Linear Regression** examined the relationship between incorrect_calls and favor_home and total_run_impact. This model intended to reveal if incorrect calls are directly favoring the home team and to what degree.

- **Multiple Linear Regression** extended the model by adding accuracy and interaction terms with below_expected flags to improve predictive performance.
- **Logistic Regression** classified games based on whether total_run_impact is above or below the median, providing a binary view of bias.
- **Decision Tree Classification** further modeled the categorical outcomes by including additional features such as umpire ID, home team, and away team, allowing nonlinear splits and feature interactions to be captured.

On the unsupervised side, we used two methods: K-Means and Hierarchical Clustering. This allowed us to define clear umpire clusters based on bias and performance metrics.

- **K-Means Clustering** was applied to favor_home and total_run_impact to identify groups of umpires who systematically favor home teams, away teams, or remain neutral.
- **Hierarchical Clustering** was additionally conducted on favor_home, total_run_impact, accuracy, and consistency to explore natural divisions among umpires based on bias and performance levels.

Evaluation metrics

We applied various approaches to evaluate performance of each model.

For supervised models:

- **R-squared** and **Root Mean Squared Error** (RMSE) evaluated the goodness of fit for the linear regressions.
- **Validation Set Approach (Train/Test Split)** measured the model accuracy and fitness as well as the generalization ability of simple and multiple linear regressions.
- **5-Fold Cross-Validation** provided a robust estimate of model stability by averaging RMSE across folds.
- For classification models, **accuracy** was used to assess logistic regression and decision tree classification performance.

For unsupervised models:

- **Inertia** and the **Elbow Method** were used to select the optimal number of clusters for K-Means.
- **Silhouette Scores** measure how well the clusters were separated.
- **Chi-Square Tests** were conducted to test the statistical significance of differences between home-team bias and away-team bias clusters.
- **Dendrograms** and **Merge Heights Plots** were used to determine the number of meaningful clusters in hierarchical clustering.

Description of the Python code

The analysis was performed using Python, employing the following libraries:

- **Pandas** for data preprocessing, feature engineering (e.g., creating below_expected and total_run_impact_category), and group operations.
- **Matplotlib** and **Seaborn** for visualization of scatterplots, dendrograms, elbow plots, and decision trees.
- **Scikit-learn** for:
    - LinearRegression, DecisionTreeClassifier, train_test_split, cross_val_score, StandardScaler, and KMeans.
- **Statsmodels** for:
    - Fitting and interpreting OLS (ordinary least squares) regressions and logistic regression models.
- **Scipy** for:
    - Hierarchical clustering (linkage, dendrogram) and chi-square tests.

RESULTS & DISCUSSION

Regression

We fitted a simple linear regression of incorrect_calls on favor_home. The result shows that there is almost no relationship between incorrect calls and how much the umpire favors the home team compared to the away team. The coefficient is 0.0006, which means that every incorrect call only increases the difference by 0.0006. In addition, the p-value for incorrect calls is 0.593, meaning that it is not significant. The R-squared of this model is around 0.000, so it does not explain any variability.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:      total_run_impact   R-squared:                       0.655
Model:                           OLS   Adj. R-squared:                  0.655
Method:                Least Squares   F-statistic:                 3.441e+04
Date:               Sun, 27 Apr 2025   Prob (F-statistic):               0.00
Time:                       23:53:51   Log-Likelihood:                -11386.
No. Observations:              18093   AIC:                         2.278e+04
Df Residuals:                  18091   BIC:                         2.279e+04
Df Model:                          1
Covariance Type:           nonrobust
===================================================================================
                      coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------
Intercept          -0.0510      0.009     -5.561      0.000      -0.069      -0.033
incorrect_calls     0.1353      0.001    185.512      0.000       0.134       0.137
==============================================================================
Omnibus:                    4634.477   Durbin-Watson:                   1.980
Prob(Omnibus):                 0.000   Jarque-Bera (JB):            15207.346
Skew:                          1.292   Prob(JB):                         0.00
Kurtosis:                      6.673   Cond. No.                         34.4
==============================================================================
```

Incorrect calls were also used to fit the total run impact to see whether incorrect calls favor either of the teams in general or it has no impact on the game. The model accounts for 65.5% of the time and each incorrect call increases the total run expectancy by 0.135. After cross validation, the MSE is 0.2 and the R-squared is 0.659. From this, each incorrect call does foster the home team or the away team to some extent.

To perform multiple linear regressions, some other factors were added to see whether they help in examining the relationship between incorrect calls and umpire biases. However, even after adding accuracy, consistency, and other supplementary variables, all predictors remained insignificant, and the model's R-square remained at 0.000. These results suggest no significant relationship between an umpire's missed calls — including potentially intentional ones — and directly favoring the home team over the away team in terms of run expectancy.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             favor_home   R-squared:                       0.000
Model:                            OLS   Adj. R-squared:                  0.000
Method:                 Least Squares   F-statistic:                     1.143
Date:                Mon, 28 Apr 2025   Prob (F-statistic):              0.335
Time:                        03:58:26   Log-Likelihood:                -17518.
No. Observations:               18093   AIC:                         3.505e+04
Df Residuals:                   18087   BIC:                         3.509e+04
Df Model:                           5
Covariance Type:            nonrobust
========================================================================================
                                   coef    std err          t      P>|t|      [0.025
----------------------------------------------------------------------------------------
Intercept                        0.7469      0.470      1.589      0.112      -0.175
accuracy                        -0.0059      0.005     -1.243      0.214      -0.015
consistency                     -0.0016      0.002     -0.657      0.511      -0.006
incorrect_calls                 -0.0039      0.003     -1.250      0.211      -0.010
expected_incorrect_calls         0.0022      0.003      0.658      0.510      -0.004
accuracy_above_expected         -0.0112      0.014     -0.802      0.423      -0.039
incorrect_calls_above_expected  -0.0060      0.006     -1.026      0.305      -0.018
==============================================================================
Omnibus:                      681.424   Durbin-Watson:                   1.987
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2030.437
Skew:                           0.077   Prob(JB):                         0.00
Kurtosis:                       4.634   Cond. No.                     8.15e+16
==============================================================================
```

However, when adding accuracy as one of the variables in modeling total_run_impact, accuracy actually is significant and increase in accuracy leads to a higher total_run_impact. This means that for umpires who have a higher accuracy rate, each incorrect call has a larger effect on the total_run_impact compared to umpires who have a low accuracy rate. This model also improves the previous simple linear regression with a slightly lower MSE at 0.199 and a higher R-squared at 0.672.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       total_run_impact   R-squared:                       0.671
Model:                            OLS   Adj. R-squared:                  0.671
Method:                 Least Squares   F-statistic:                 1.846e+04
Date:                Sun, 27 Apr 2025   Prob (F-statistic):               0.00
Time:                        22:09:57   Log-Likelihood:                -10964.
No. Observations:               18093   AIC:                         2.193e+04
Df Residuals:                   18090   BIC:                         2.196e+04
Df Model:                           2
Covariance Type:            nonrobust
===================================================================================
                      coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------
Intercept          -8.1884      0.277    -29.584      0.000      -8.731      -7.646
accuracy            0.0824      0.003     29.415      0.000       0.077       0.088
incorrect_calls     0.1801      0.002    107.093      0.000       0.177       0.183
==============================================================================
Omnibus:                     4616.667   Durbin-Watson:                   1.988
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            15622.994
Skew:                           1.276   Prob(JB):                         0.00
Kurtosis:                       6.770   Cond. No.                     7.82e+03
==============================================================================
```

To further examine if different umpires have different biases on a specific home team, the data was grouped by umpire and home team and incorrect_calls and total_run_impact were calculated on mean. The model shows that each incorrect call increases the total_run_impact around 0.1298 when the umpire and home team are paired up. This model has a lower MSE around 0.05 and a higher R-squared around 0.723.

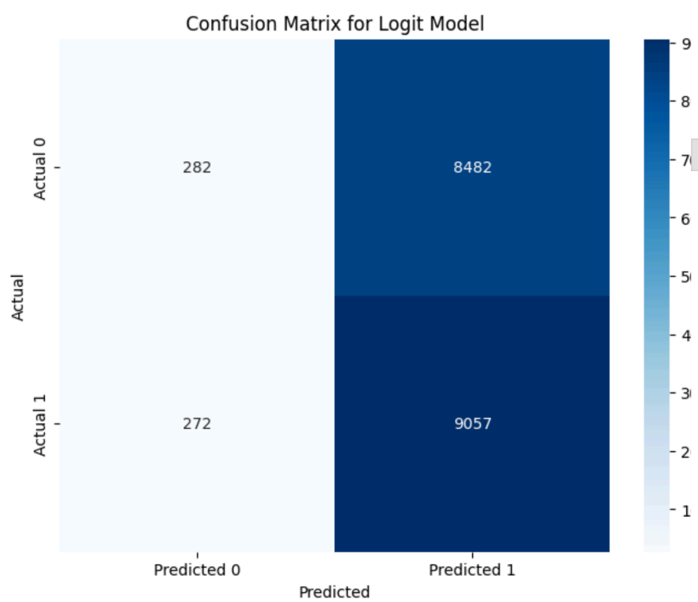
```
                            OLS Regression Results
==============================================================================
Dep. Variable:       total_run_impact   R-squared:                       0.723
Model:                            OLS   Adj. R-squared:                  0.723
Method:                 Least Squares   F-statistic:                     8672.
Date:                Mon, 28 Apr 2025   Prob (F-statistic):               0.00
Time:                        22:59:45   Log-Likelihood:                 56.452
No. Observations:                3321   AIC:                            -108.9
Df Residuals:                    3319   BIC:                            -96.69
Df Model:                           1
Covariance Type:            nonrobust
===================================================================================
                      coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------
Intercept           0.0100      0.017      0.593      0.553      -0.023       0.043
incorrect_calls     0.1298      0.001     93.125      0.000       0.127       0.133
==============================================================================
Omnibus:                      585.495   Durbin-Watson:                   1.997
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2092.397
Skew:                           0.855   Prob(JB):                         0.00
Kurtosis:                       6.492   Cond. No.                         49.7
==============================================================================
```

However, even after grouping by home team and umpire, the relationship between favor_home and incorrect calls remains weak. This may be due to limited repetitions of specific umpire-team pairings, making individual umpire preferences difficult to detect.

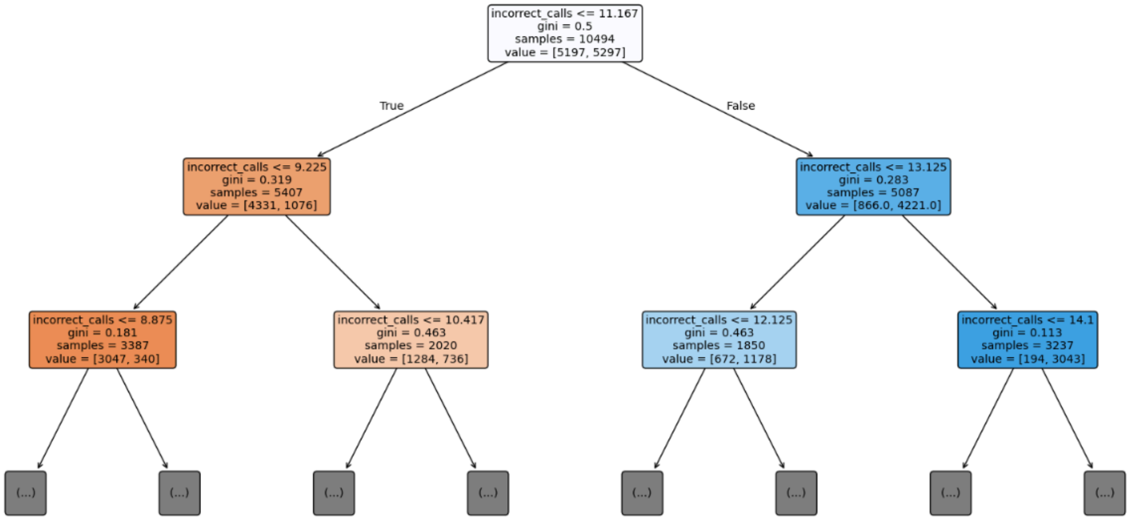
Classification

To further identify whether incorrect calls favor home teams, logistic regression was adopted to examine such relationships using 1 as it favors home teams more and 0 as it does not favor home teams more. The model can showcase if an increase in incorrect calls increases the chance of favoring home.

However, the coefficient seems not significant, suggesting that an increase in incorrect calls may not actually increase the chance that the umpire is favoring home. After several other factors similar to multiple linear regressions were incorporated, no significant parameters stood out. The average cross validation score is around 0.515 and the accuracy is around 0.516, together with the confusion matrix. This indicates that using the number of incorrect calls to conclude whether or not the umpire is favoring the home team is basically no better than random guessing.



Before running a decision tree for incorrect_calls and total_run_impact, a logistic regression was performed. While big impacts are the ones above mean and small impacts are the ones below mean, the more incorrect calls, the higher chance the bigger total run impact happens. The result is significant as every incorrect call increases the chance of the total run impacts to be above average by 59%. The accuracy is around 0.78. However, because the combinations of umpire, home team, and away team are almost all unique, the decision tree shows that while the number of incorrect calls is a primary factor for splitting, the tree spreads too widely as other factors lack repetition.



Reviewing prior studies, such as Hsu (2023), suggests that home bias among umpire calls does exist, particularly when the home team is batting, and that the type of incorrect call also plays a significant role. Simply examining the number of incorrect calls may blur these effects, as it mixes calls made while the home team and away team are batting, potentially evening out the observed bias. To improve accuracy, future studies should use datasets that specify the home team, batting team, and types of incorrect calls. Nevertheless, it remains important to recognize that each incorrect call increases total run impact, meaning no incorrect call is truly neutral. In practice, teams batting as the away team must be especially mindful of the potential disadvantage introduced by such biases.

K-Means Clustering

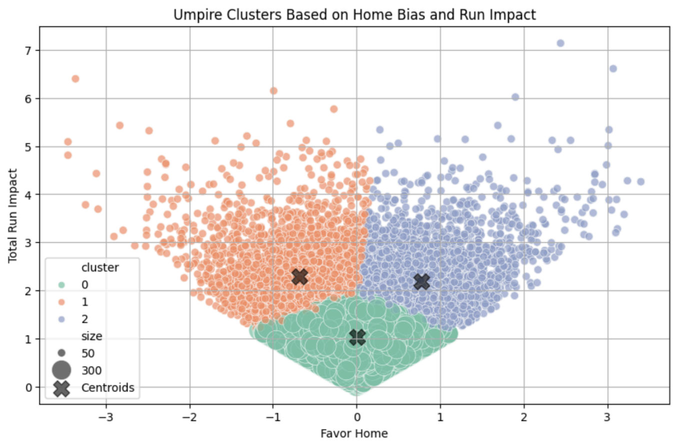
The K-Means algorithm successfully divided the umpires into three distinct clusters based on their bias tendencies. Cluster 0, labeled "Neutral," included 10,577 umpires (58.46%) who demonstrated no strong preference for either the home or away team. The favor_home scores for these umpires were close to zero, indicating a relatively neutral stance.

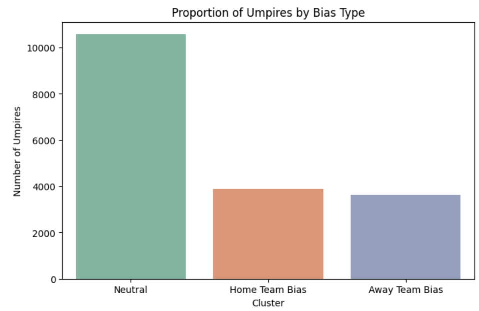
Cluster 1, "Away Team Bias," comprised 3,617 umpires (19.99%) who favored the away team, as evidenced by a negative favor_home score and a higher total_run_impact score, suggesting that their calls were more likely to favor the visiting team's chances.

Finally, Cluster 2, "Home Team Bias," contained 3,899 umpires (21.55%) who favored the home team. These umpires had positive favor_home scores and higher total_run_impact values, indicating that their calls influenced the home team's scoring more significantly.

The distribution of umpires across the clusters shows that most umpires (58.46%) are neutral, but there is still a substantial proportion who display some level of bias. Interestingly, the Home

Team Bias cluster (Cluster 2) had more umpires than the Away Team Bias cluster (Cluster 1), suggesting that home team bias might be more prevalent than away team bias among MLB umpires.



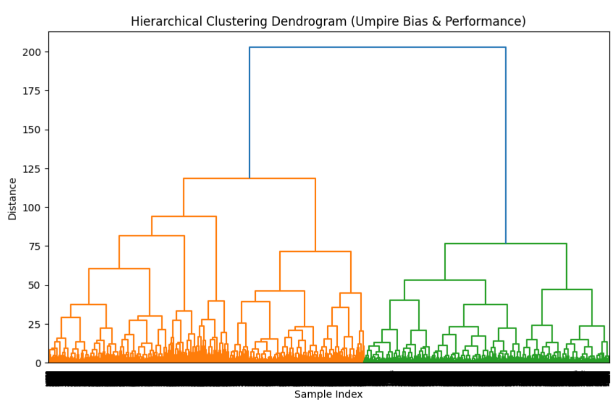


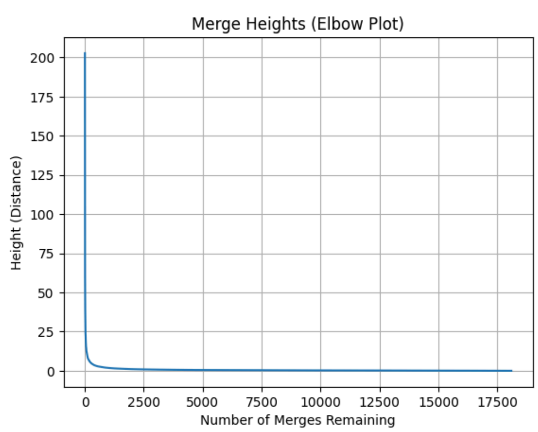

```
Percentage of Umpires in Each Cluster:
Neutral: 10577 umpires (58.46%)
Home Team Bias: 3899 umpires (21.55%)
Away Team Bias: 3617 umpires (19.99%)
Cluster Sizes (Umpires in each cluster):
cluster_name
Neutral           10577
Home Team Bias     3899
Away Team Bias     3617
Name: count, dtype: int64
```

In terms of fairness, these results could have important implications for the sport. If certain umpires consistently favor the home team, this could lead to potential fairness concerns, particularly in close games where decisions might influence the final outcome. This raises questions about how to address this issue, whether through more thorough training for umpires to recognize and correct bias, or through the introduction of more technological tools, such as automated strike zones, to reduce human error.

Hierarchical Clustering

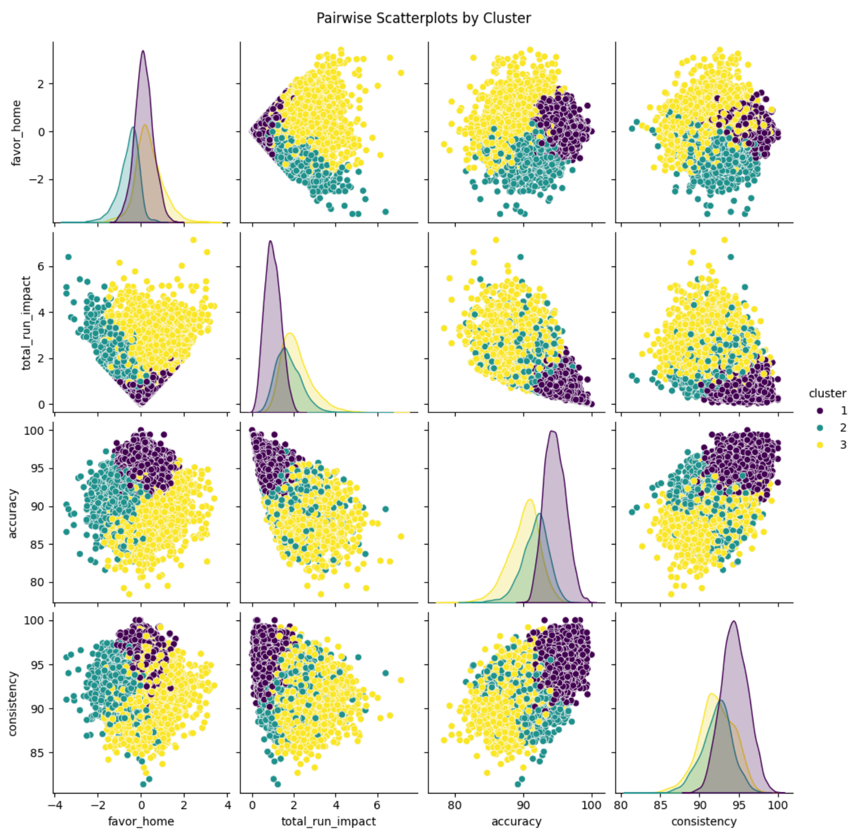
Each game started as its own cluster, and clusters were merged step-by-step based on similarity. We visualized this process with a dendrogram on the left and an elbow plot on the right.





The merge heights drop dramatically after the first few merges, then flatten. After that, there's very little distance between clusters. Based on these graphs, we identified 2-3 meaningful clusters. Given the dataset of over 18,000 games, 3 clusters seem better to capture importance differences. Beyond three, any additional clusters are very close together (small differences).

Pairwise scatterplots visualize relationships among favor_home, total_run_impact, accuracy, and consistency across clusters. Distinct patterns emerge, with Cluster 1 showing higher accuracy and consistency, while Cluster 3 exhibits greater variability in bias and performance. These plots reveal meaningful groupings of umpires based on bias and decision quality, confirming that the clustering captured real behavioral differences.



This table shows the average favor_home, total_run_impact, accuracy, and consistency for each cluster.

cluster	favor_home	total_run_impact	accuracy	consistency
1	0.125919	0.976895	94.589246	94.445686
2	-0.582561	1.743164	91.673700	92.192913
3	0.369925	2.126765	90.043539	92.172754

The results confirm the patterns observed in the pairwise scatterplots. Cluster 1 umpires have the highest accuracy (94.59) and consistency (94.44) with low total_run_impact (0.976) and mild home-team favoring (0.126), suggesting strong performance and neutrality. Cluster 2 umpires show a slight away-team bias (-0.583) with moderate run impact (1.743) and lower accuracy (91.67) compared to Cluster 1. Cluster 3 umpires demonstrate stronger home-team bias (0.37), the highest run impact (2.127), and the lowest accuracy (90.04), indicating more biased and inconsistent decision-making. Overall, the table reinforces that the clusters meaningfully capture differences in umpire bias and performance.

From a business perspective, these findings suggest actionable steps. Cluster 1 umpires could be prioritized for high-stakes games to ensure fairness. Cluster 3 umpires may need additional monitoring or retraining. Overall, clustering provides a way to support more transparent and data-driven officiating.

However, our analysis has some limitations. Hierarchical clustering is sensitive to outliers and depends heavily on a few variables. The results may also change depending on the linkage method we choose — for example, Ward vs single linkage. Lastly, it's worth noting that hierarchical clustering is computationally heavy, especially with large datasets, so it may not scale well in real-time systems.

CONCLUSION

To conclude, supervised learning models confirmed that incorrect calls do increase run expectancy, but not all errors automatically favor home teams. Both hierarchical and K-Means clustering revealed clear umpire groupings based on bias and performance, highlighting a small but critical segment of problematic officiating. These insights can help leagues assign umpires more strategically and monitor officiating quality over time. Future research should bring in richer game context to refine bias detection and prediction.

Looking ahead, there are several ways this research can be extended and improved. First, we can explore more advanced models like Random Forest or XGBoost. These non-linear models may better capture the complex patterns behind umpire behavior and improve prediction of run impact or bias. Second, future work could include more contextual variables such as inning, score differential, pitch type, or even umpire experience to uncover deeper patterns that aren't visible from game-level data alone. Third, we could build a tool to track umpire group assignments over time. This would allow leagues to flag trends or shifts in bias early, before they become problematic. Finally, we see potential for integrating this analysis into actual umpire assignment strategies — especially for playoff or high-stakes games where fairness is critical.

REFERENCES

Mills, M.B. (2017). Umpire Analytics. *The SABR Book of Umpires and Umpiring*. https://sabr.org/journal/article/umpire-analytics/

Williams, W.T. (2019, April 8). *MLB Umpires Missed 34,294 Ball-Strike Calls in 2018. Bring on Robo-umps? After studying four million game pitches, BU researcher suggests how to fix a broken baseball system.* https://www.bu.edu/articles/2019/mlb-umpires-strike-zone-accuracy/

Hsu, M. (2023). Umpire home bias in Major League Baseball. *Journal of Sports Economics*, 25(4), 423–442. https://doi.org/10.1177/15270025231222631

Flannagan, K. S., Mills, B. M., & Goldstone, R. L. (2024). The psychophysics of home plate umpire calls. *Scientific Reports*, 14, 2735. https://doi.org/10.1038/s41598-024-52402-y

Tainsky, S., Mills, B., & Winfree, J. (2015). Further examination of potential discrimination among MLB umpires. *Journal of Sports Economics*, 16(1), 1–15. https://doi.org/10.1177/1527002513487740

Kim, J., & King, B. (2014). Seeing stars: Matthew effects and status bias in Major League Baseball umpiring. *Management Science*, 60(11), 2619–2644. https://doi.org/10.1287/mnsc.2014.1967

Appendices

May 8, 2025

1 Imports

[ ]:
```
from google.colab import files
import pandas as pd
import io

# Upload the file
uploaded = files.upload()

# Get the uploaded file name dynamically
filename = list(uploaded.keys())[0]  # This grabs the first (and only) file␣
 ↪uploaded

# Read the uploaded CSV file into a Pandas DataFrame
umpire = pd.read_csv(io.BytesIO(uploaded[filename]))

# Display the first few rows of the dataset
print(umpire.head())
```

```
<IPython.core.display.HTML object>

Saving mlb-umpire-scorecard.csv to mlb-umpire-scorecard.csv
   id        date           umpire home away  home_team_runs  away_team_runs  \
0   1  2022-11-05  Lance Barksdale  HOU  PHI               4               1
1   2  2022-11-03     Jordan Baker  PHI  HOU               2               3
2   3  2022-11-02     Tripp Gibson  PHI  HOU               0               5
3   4  2022-11-01     Dan Iassogna  PHI  HOU               7               0
4   5  2022-10-29       Pat Hoberg  HOU  PHI               5               2

  pitches_called incorrect_calls expected_incorrect_calls correct_calls  \
0            124               4                       10           120
1            149               6                      7.4           143
2            124               7                      7.1           117
3            140               5                        6           135
4            129               0                      8.7           129

  expected_correct_calls correct_calls_above_expected accuracy  \
0                    114                            6     96.8
```

```
1                141.6                          1.4      96
2                116.9                          0.1    94.4
3                  134                            1    96.4
4                120.3                          8.7     100

  expected_accuracy accuracy_above_expected consistency favor_home  \
0                92                     4.8        97.6       0.09
1                95                     0.9        97.3      -0.12
2              94.3                     0.1        92.7       -0.1
3              95.7                     0.7        92.9       0.63
4              93.2                     6.8        96.1          0

  total_run_impact
0             0.75
1             0.58
2             0.56
3             0.73
4                0
```

2 Preprocessing

[ ]:
```
umpire.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18213 entries, 0 to 18212
Data columns (total 19 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   id                            18213 non-null  int64
 1   date                          18213 non-null  object
 2   umpire                        18213 non-null  object
 3   home                          18213 non-null  object
 4   away                          18213 non-null  object
 5   home_team_runs                18213 non-null  int64
 6   away_team_runs                18213 non-null  int64
 7   pitches_called                18213 non-null  object
 8   incorrect_calls               18213 non-null  object
 9   expected_incorrect_calls      18213 non-null  object
 10  correct_calls                 18213 non-null  object
 11  expected_correct_calls        18213 non-null  object
 12  correct_calls_above_expected  18213 non-null  object
 13  accuracy                      18213 non-null  object
 14  expected_accuracy             18213 non-null  object
 15  accuracy_above_expected       18213 non-null  object
 16  consistency                   18213 non-null  object
 17  favor_home                    18213 non-null  object
 18  total_run_impact              18213 non-null  object
```

```
dtypes: int64(3), object(16)
memory usage: 2.6+ MB
```

[ ]:
```python
# Convert columns to numeric
for col in umpire.columns[5:]:
    umpire[col] = pd.to_numeric(umpire[col], errors='coerce')
```

[ ]:
```python
umpire.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18213 entries, 0 to 18212
Data columns (total 19 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   id                            18213 non-null  int64
 1   date                          18213 non-null  object
 2   umpire                        18213 non-null  object
 3   home                          18213 non-null  object
 4   away                          18213 non-null  object
 5   home_team_runs                18213 non-null  int64
 6   away_team_runs                18213 non-null  int64
 7   pitches_called                18093 non-null  float64
 8   incorrect_calls               18093 non-null  float64
 9   expected_incorrect_calls      18093 non-null  float64
 10  correct_calls                 18093 non-null  float64
 11  expected_correct_calls        18093 non-null  float64
 12  correct_calls_above_expected  18093 non-null  float64
 13  accuracy                      18093 non-null  float64
 14  expected_accuracy             18093 non-null  float64
 15  accuracy_above_expected       18093 non-null  float64
 16  consistency                   18093 non-null  float64
 17  favor_home                    18093 non-null  float64
 18  total_run_impact              18093 non-null  float64
dtypes: float64(12), int64(3), object(4)
memory usage: 2.6+ MB
```

[ ]:
```python
# Check for missing values
umpire.isnull().sum()
```

[ ]:
```
id                                0
date                              0
umpire                            0
home                              0
away                              0
home_team_runs                    0
away_team_runs                    0
pitches_called                  120
```

```
incorrect_calls                  120
expected_incorrect_calls         120
correct_calls                    120
expected_correct_calls           120
correct_calls_above_expected     120
accuracy                         120
expected_accuracy                120
accuracy_above_expected          120
consistency                      120
favor_home                       120
total_run_impact                 120
dtype: int64
```

[ ]:
```python
# Remove the rows containing missing values
umpire.dropna(inplace=True)
```

[ ]:
```python
umpire.isnull().sum()
```

[ ]:
```
id                               0
date                             0
umpire                           0
home                             0
away                             0
home_team_runs                   0
away_team_runs                   0
pitches_called                   0
incorrect_calls                  0
expected_incorrect_calls         0
correct_calls                    0
expected_correct_calls           0
correct_calls_above_expected     0
accuracy                         0
expected_accuracy                0
accuracy_above_expected          0
consistency                      0
favor_home                       0
total_run_impact                 0
dtype: int64
```

3 Methodology

3.1 Supervised Learning

3.1.1 Regression

[ ]:
```
import statsmodels.formula.api as sm

# Fit the linear regression model
model = sm.ols("favor_home ~ incorrect_calls", data=umpire).fit()

# Print the model summary
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             favor_home   R-squared:                       0.000
Model:                            OLS   Adj. R-squared:                  0.000
Method:                 Least Squares   F-statistic:                     1.781
Date:                Thu, 08 May 2025   Prob (F-statistic):              0.182
Time:                        21:02:07   Log-Likelihood:                -17520.
No. Observations:               18093   AIC:                         3.504e+04
Df Residuals:                   18091   BIC:                         3.506e+04
Df Model:                           1
Covariance Type:            nonrobust
================================================================================
===
                      coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
---
Intercept           0.0186      0.013      1.440      0.150      -0.007
0.044
incorrect_calls     0.0014      0.001      1.334      0.182      -0.001
0.003
==============================================================================
Omnibus:                      680.777   Durbin-Watson:                   1.987
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2035.376
Skew:                           0.072   Prob(JB):                         0.00
Kurtosis:                       4.637   Cond. No.                         34.4
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
```

```
# Define features (X) and target (y)
X = umpire[['incorrect_calls']]
y = umpire['favor_home']

# Split data into training (80%) and validation (20%) sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Refit the model using only the training data
model = sm.ols("favor_home ~ incorrect_calls", data=pd.concat([X_train,␣
↪y_train], axis=1)).fit()

# Make predictions on the validation set
y_pred = model.predict(X_val)

# Evaluate the model
r2 = r2_score(y_val, y_pred)
mse = mean_squared_error(y_val, y_pred)

print(f"R-squared: {r2}")
print(f"Mean Squared Error: {mse}")
```

```
R-squared: 4.613132045250268e-07
Mean Squared Error: 0.42796695177436883
```

[ ]:
```
# Fit the linear regression model
model = sm.ols("favor_home ~ accuracy + consistency + incorrect_calls +␣
↪expected_incorrect_calls + accuracy_above_expected", data=umpire).fit()

# Print the model summary
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             favor_home   R-squared:                       0.000
Model:                            OLS   Adj. R-squared:                  0.000
Method:                 Least Squares   F-statistic:                     1.143
Date:                Thu, 08 May 2025   Prob (F-statistic):              0.335
Time:                        21:02:52   Log-Likelihood:                -17518.
No. Observations:               18093   AIC:                         3.505e+04
Df Residuals:                   18087   BIC:                         3.509e+04
Df Model:                           5
Covariance Type:            nonrobust
================================================================================
============
                               coef    std err          t      P>|t|      [0.025
```

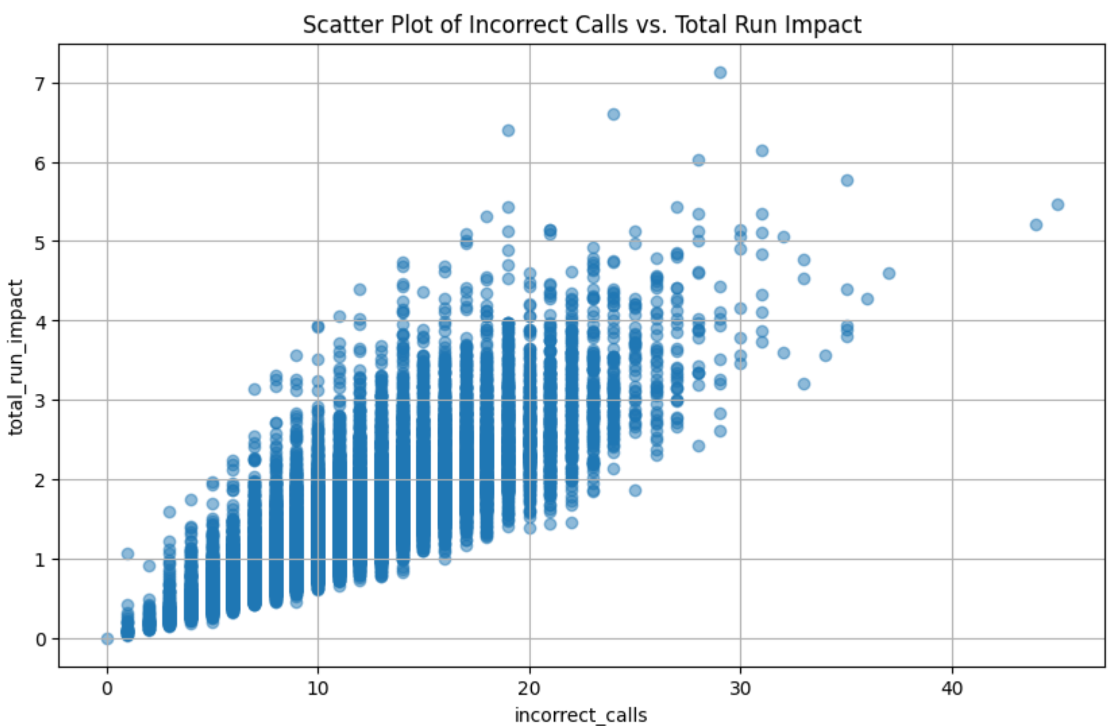
```
0.975]
--------------------------------------------------------------------------------
------------
Intercept                    0.7469      0.470      1.589      0.112      -0.175
1.668
accuracy                    -0.0059      0.005     -1.243      0.214      -0.015
0.003
consistency                 -0.0016      0.002     -0.657      0.511      -0.006
0.003
incorrect_calls             -0.0099      0.009     -1.125      0.261      -0.027
0.007
expected_incorrect_calls     0.0082      0.009      0.910      0.363      -0.009
0.026
accuracy_above_expected     -0.0112      0.014     -0.802      0.423      -0.039
0.016
==============================================================================
Omnibus:                      681.424   Durbin-Watson:                   1.987
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2030.437
Skew:                           0.077   Prob(JB):                         0.00
Kurtosis:                       4.634   Cond. No.                     1.31e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.31e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

[ ]:
```python
# Define features (X) and target (y)
X = umpire[['accuracy', 'consistency', 'incorrect_calls',␣
 ↪'expected_incorrect_calls', 'accuracy_above_expected']]
y = umpire['favor_home']

# Split data into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Refit the model using only the training data
model = sm.ols("favor_home ~ accuracy + consistency + incorrect_calls +␣
 ↪expected_incorrect_calls + accuracy_above_expected", data=pd.
 ↪concat([X_train, y_train], axis=1)).fit()

# Make predictions on the testing set
y_pred = model.predict(X_test)

# Evaluate the model
r2 = r2_score(y_test, y_pred)
```

```
mse = mean_squared_error(y_test, y_pred)

print(f"R-squared: {r2}")
print(f"Mean Squared Error: {mse}")
```

```
R-squared: -0.00012449809273351597
Mean Squared Error: 0.42802043029509396
```

[ ]:
```
import matplotlib.pyplot as plt

# Scatterplot of 'incorrect_calls' vs. 'total_run_impact'
plt.figure(figsize=(10, 6))
plt.scatter(umpire['incorrect_calls'], umpire['total_run_impact'], alpha=0.5)
plt.xlabel('incorrect_calls')
plt.ylabel('total_run_impact')
plt.title('Scatter Plot of Incorrect Calls vs. Total Run Impact')
plt.grid(True)
plt.show()
```



[ ]:
```
# Fit the linear regression model
model = sm.ols('total_run_impact ~ incorrect_calls', data=umpire).fit()

# Print the model summary
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       total_run_impact   R-squared:                       0.655
Model:                            OLS   Adj. R-squared:                  0.655
Method:                 Least Squares   F-statistic:                 3.441e+04
Date:                Thu, 08 May 2025   Prob (F-statistic):               0.00
Time:                        21:04:14   Log-Likelihood:                -11386.
No. Observations:               18093   AIC:                         2.278e+04
Df Residuals:                   18091   BIC:                         2.279e+04
Df Model:                           1
Covariance Type:            nonrobust
================================================================================
===
                      coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
---
Intercept          -0.0510      0.009     -5.561      0.000      -0.069
-0.033
incorrect_calls     0.1353      0.001    185.512      0.000       0.134
0.137
==============================================================================
Omnibus:                     4634.477   Durbin-Watson:                   1.980
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            15207.346
Skew:                           1.292   Prob(JB):                         0.00
Kurtosis:                       6.673   Cond. No.                         34.4
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[ ]:
```
# Validation set approach for simple linear regression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error

# Define features (X) and target (y)
X = umpire[['incorrect_calls']]
y = umpire['total_run_impact']

# Split data into training (80%) and validation (20%) sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
 ↪random_state=42)

# Initialize and train the linear regression model
model = LinearRegression()
```

```
model.fit(X_train, y_train)

# Make predictions on the validation set
y_pred = model.predict(X_val)

# Evaluate the model
r2 = r2_score(y_val, y_pred)
mse = mean_squared_error(y_val, y_pred)

print(f'R-squared on Validation Set (Simple): {r2}')
print(f'Mean Squared Error on Validation Set (Simple): {mse}')
```

```
R-squared on Validation Set (Simple): 0.6592788131034484
Mean Squared Error on Validation Set (Simple): 0.20729338086597374
```

[ ]:
```
# Add 'below_expected' column to the DataFrame
umpire['below_expected'] = umpire['correct_calls_above_expected'].apply(lambda␣
↪x: 1 if x < 0 else 0)
```

[ ]:
```
# Run a multiple linear regression

# Fit the linear regression model
model = sm.ols('total_run_impact ~ accuracy + incorrect_calls', data=umpire).
↪fit()

# Print the model summary
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       total_run_impact   R-squared:                       0.671
Model:                            OLS   Adj. R-squared:                  0.671
Method:                 Least Squares   F-statistic:                 1.846e+04
Date:                Thu, 08 May 2025   Prob (F-statistic):               0.00
Time:                        21:04:19   Log-Likelihood:                -10964.
No. Observations:               18093   AIC:                         2.193e+04
Df Residuals:                   18090   BIC:                         2.196e+04
Df Model:                           2
Covariance Type:            nonrobust
================================================================================
===
                      coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
---
Intercept          -8.1884      0.277    -29.584      0.000      -8.731
-7.646
```

```
accuracy             0.0824      0.003     29.415      0.000       0.077
0.088
incorrect_calls      0.1801      0.002    107.093      0.000       0.177
0.183
==============================================================================
Omnibus:                     4616.667   Durbin-Watson:                   1.988
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            15622.994
Skew:                           1.276   Prob(JB):                         0.00
Kurtosis:                       6.770   Cond. No.                     7.82e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 7.82e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

[ ]:

```python
# Validation set approach for multiple linear regression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error

# Define features (X) and target (y)
X = umpire[['incorrect_calls', 'accuracy']]
y = umpire['total_run_impact']

# Split data into training (80%) and validation (20%) sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Initialize and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the validation set
y_pred = model.predict(X_val)

# Evaluate the model
r2 = r2_score(y_val, y_pred)
mse = mean_squared_error(y_val, y_pred)

print(f'R-squared on Validation Set (Multiple): {r2}')
print(f'Mean Squared Error on Validation Set (Multiple): {mse}')
```

```
R-squared on Validation Set (Multiple): 0.6721802226611603
Mean Squared Error on Validation Set (Multiple): 0.19944421589471334
```

[ ]:
```python
# K-Fold Cross-Validation for simple linear regression
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression

# Define features (X) and target (y)
X = umpire[['incorrect_calls']]
y = umpire['total_run_impact']

# Create a linear regression model
lr_model = LinearRegression()

# Perform 5-fold cross-validation
cv_scores = cross_val_score(lr_model, X, y, cv=5,␣
 ↪scoring='neg_mean_squared_error')

# Print the cross-validation scores
print('Simple Linear Regression:')
print('Cross-validation scores (negative MSE):', cv_scores)
print('Average cross-validation score (negative MSE):', cv_scores.mean())

# Convert negative MSE scores to positive RMSE
rmse_scores = (-cv_scores)**0.5
print('RMSE scores:', rmse_scores)
print('Average RMSE:', rmse_scores.mean())
```

```
Simple Linear Regression:
Cross-validation scores (negative MSE): [-0.17860499 -0.19089846 -0.19132132
-0.23318285 -0.24257466]
Average cross-validation score (negative MSE): -0.207316456920197
RMSE scores: [0.42261684 0.43691928 0.43740292 0.48289011 0.49251869]
Average RMSE: 0.454469568242471
```

[ ]:
```python
# Fit the linear regression model
model = sm.ols('total_run_impact ~ incorrect_calls + accuracy + incorrect_calls␣
 ↪* below_expected', data=umpire).fit()

# Print the model summary
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       total_run_impact   R-squared:                       0.671
Model:                            OLS   Adj. R-squared:                  0.671
Method:                 Least Squares   F-statistic:                     9241.
Date:                Thu, 08 May 2025   Prob (F-statistic):               0.00
Time:                        21:04:27   Log-Likelihood:                -10956.
No. Observations:               18093   AIC:                         2.192e+04
```

```
Df Residuals:                   18088   BIC:                         2.196e+04
Df Model:                           4
Covariance Type:            nonrobust
================================================================================
==================
                                     coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------------------
------------------
Intercept                         -8.5063      0.292    -29.164      0.000
-9.078      -7.935
incorrect_calls                    0.1830      0.002     91.400      0.000
0.179       0.187
accuracy                           0.0854      0.003     28.952      0.000
0.080       0.091
below_expected                     0.0801      0.022      3.571      0.000
0.036       0.124
incorrect_calls:below_expected    -0.0049      0.002     -2.774      0.006
-0.008      -0.001
==============================================================================
Omnibus:                     4616.577   Durbin-Watson:                   1.990
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            15606.763
Skew:                           1.276   Prob(JB):                         0.00
Kurtosis:                       6.766   Cond. No.                     8.27e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 8.27e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

[ ]:
```python
# K-Fold Cross-Validation for multiple linear regression

# Define features (X) and target (y)
X = umpire[['incorrect_calls', 'accuracy', 'below_expected']]
y = umpire['total_run_impact']

# Create a linear regression model
lr_model = LinearRegression()

# Perform 5-fold cross-validation
cv_scores = cross_val_score(lr_model, X, y, cv=5,
 ↪scoring='neg_mean_squared_error')

# Print the cross-validation scores
print('Multiple Linear Regression:')
```

```
print('Cross-validation scores (negative MSE):', cv_scores)
print('Average cross-validation score (negative MSE):', cv_scores.mean())

# Convert negative MSE scores to positive RMSE
rmse_scores = (-cv_scores)**0.5
print('RMSE scores:', rmse_scores)
print('Average RMSE:', rmse_scores.mean())
```

```
Multiple Linear Regression:
Cross-validation scores (negative MSE): [-0.17012396 -0.18300924 -0.18437719
-0.22019352 -0.22851762]
Average cross-validation score (negative MSE): -0.19724430454646763
RMSE scores: [0.41246086 0.42779579 0.42939165 0.46924782 0.47803517]
Average RMSE: 0.443386256706229
```

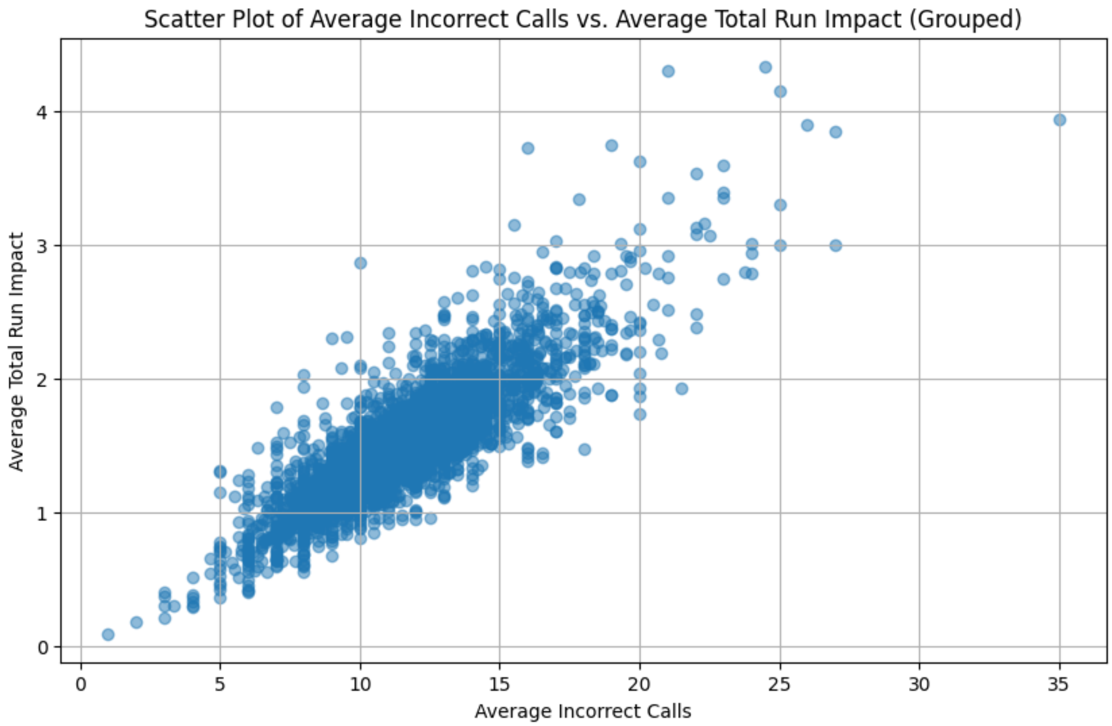
[ ]:
```
# Group by umpire and home, then calculate the average incorrect_calls
grouped_umpire = umpire.groupby(['umpire', 'home'])[['incorrect_calls',␣
↪'total_run_impact']].mean().reset_index()
grouped_umpire
```

[ ]:

	umpire	home	incorrect_calls	total_run_impact
0	Adam Beck	ARI	6.000000	0.450000
1	Adam Beck	ATL	8.285714	1.364286
2	Adam Beck	BAL	8.000000	0.815000
3	Adam Beck	BOS	6.500000	0.750000
4	Adam Beck	CHC	11.000000	1.150000
…	…	…	…	…
3316	Will Little	STL	8.636364	1.180909
3317	Will Little	TB	8.000000	0.960000
3318	Will Little	TEX	8.500000	0.993750
3319	Will Little	TOR	8.800000	1.048000
3320	Will Little	WSH	10.888889	1.456667

[3321 rows x 4 columns]

[ ]:
```
# Scatterplot of 'incorrect_calls' vs. 'total_run_impact' after grouping
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plt.scatter(grouped_umpire['incorrect_calls'],␣
↪grouped_umpire['total_run_impact'], alpha=0.5)
plt.xlabel('Average Incorrect Calls')
plt.ylabel('Average Total Run Impact')
plt.title('Scatter Plot of Average Incorrect Calls vs. Average Total Run Impact␣
↪(Grouped)')
plt.grid(True)
plt.show()
```




```
[ ]: # Fit the linear regression model
model = sm.ols('total_run_impact ~ incorrect_calls', data=grouped_umpire).fit()

# Print the model summary
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       total_run_impact   R-squared:                       0.723
Model:                            OLS   Adj. R-squared:                  0.723
Method:                 Least Squares   F-statistic:                     8672.
Date:                Thu, 08 May 2025   Prob (F-statistic):               0.00
Time:                        21:04:50   Log-Likelihood:                 56.452
No. Observations:                3321   AIC:                            -108.9
Df Residuals:                    3319   BIC:                            -96.69
Df Model:                           1
Covariance Type:            nonrobust
================================================================================
===
                      coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
---
Intercept           0.0100      0.017      0.593      0.553      -0.023
```

```
0.043
incorrect_calls      0.1298       0.001      93.125       0.000       0.127
0.133
==============================================================================
Omnibus:                      585.495   Durbin-Watson:                   1.997
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2092.397
Skew:                           0.855   Prob(JB):                         0.00
Kurtosis:                       6.492   Cond. No.                         49.7
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[ ]:
```python
# K-Fold Cross-Validation for simple linear regression after grouping

# Define features (X) and target (y)
X = grouped_umpire[['incorrect_calls']]
y = grouped_umpire['total_run_impact']

# Create a linear regression model
lr_model = LinearRegression()

# Perform 5-fold cross-validation
cv_scores = cross_val_score(lr_model, X, y, cv=5,␣
 ↪scoring='neg_mean_squared_error')

# Print the cross-validation scores
print('Simple Linear Regression (Grouped):')
print('Cross-validation scores (negative MSE):', cv_scores)
print('Average cross-validation score (negative MSE):', cv_scores.mean())

# Convert negative MSE scores to positive RMSE
rmse_scores = (-cv_scores)**0.5
print('RMSE scores:', rmse_scores)
print('Average RMSE:', rmse_scores.mean())
```

```
Simple Linear Regression (Grouped):
Cross-validation scores (negative MSE): [-0.05311681 -0.05999703 -0.05749804
-0.06207587 -0.05058654]
Average cross-validation score (negative MSE): -0.056654856577248315
RMSE scores: [0.23047084 0.24494291 0.23978748 0.24915029 0.22491452]
Average RMSE: 0.23785320873504928
```

3.1.2 Classification

[ ]:
```
umpire_1 = umpire.copy()
```

[ ]:
```
umpire_1.head()
```

[ ]:
```
   id        date           umpire home away  home_team_runs  away_team_runs  \
0   1  2022-11-05  Lance Barksdale  HOU  PHI               4               1
1   2  2022-11-03     Jordan Baker  PHI  HOU               2               3
2   3  2022-11-02     Tripp Gibson  PHI  HOU               0               5
3   4  2022-11-01     Dan Iassogna  PHI  HOU               7               0
4   5  2022-10-29       Pat Hoberg  HOU  PHI               5               2

   pitches_called  incorrect_calls  expected_incorrect_calls  correct_calls  \
0           124.0              4.0                      10.0          120.0
1           149.0              6.0                       7.4          143.0
2           124.0              7.0                       7.1          117.0
3           140.0              5.0                       6.0          135.0
4           129.0              0.0                       8.7          129.0

   expected_correct_calls  correct_calls_above_expected  accuracy  \
0                   114.0                           6.0      96.8
1                   141.6                           1.4      96.0
2                   116.9                           0.1      94.4
3                   134.0                           1.0      96.4
4                   120.3                           8.7     100.0

   expected_accuracy  accuracy_above_expected  consistency  favor_home  \
0               92.0                      4.8         97.6        0.09
1               95.0                      0.9         97.3       -0.12
2               94.3                      0.1         92.7       -0.10
3               95.7                      0.7         92.9        0.63
4               93.2                      6.8         96.1        0.00

   total_run_impact  below_expected
0              0.75               0
1              0.58               0
2              0.56               0
3              0.73               0
4              0.00               0
```

[ ]:
```
umpire_1['favor_home_or_not'] = umpire_1['favor_home'].apply(lambda x: 1 if x >
 ↪0 else 0)
```

[ ]:
```
import statsmodels.api as sm

# Define the dependent and independent variables
```

```
X = umpire_1['incorrect_calls']
y = umpire_1['favor_home_or_not']

# Add a constant to the independent variable
X = sm.add_constant(X)

# Fit the logit model
logit_model = sm.Logit(y, X).fit()

# Print the model summary
print(logit_model.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.692659
         Iterations 3
                           Logit Regression Results
==============================================================================
Dep. Variable:      favor_home_or_not   No. Observations:                18093
Model:                          Logit   Df Residuals:                    18091
Method:                           MLE   Df Model:                            1
Date:                Thu, 08 May 2025   Pseudo R-squ.:               2.247e-07
Time:                        21:07:38   Log-Likelihood:                -12532.
converged:                       True   LL-Null:                       -12532.
Covariance Type:            nonrobust   LLR p-value:                    0.9402
================================================================================
===
                      coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
---
const               0.0597      0.040      1.475      0.140      -0.020
0.139
incorrect_calls     0.0002      0.003      0.075      0.940      -0.006
0.007
================================================================================
===
```

[ ]:
```
# Define the dependent and independent variables
X = umpire_1[['incorrect_calls', 'accuracy', 'consistency',␣
 ↪'expected_incorrect_calls', 'accuracy_above_expected']]
y = umpire_1['favor_home_or_not']

# Add a constant to the independent variable
X = sm.add_constant(X)

# Fit the logit model
logit_model = sm.Logit(y, X).fit()
```

```
# Print the model summary
print(logit_model.summary())
```

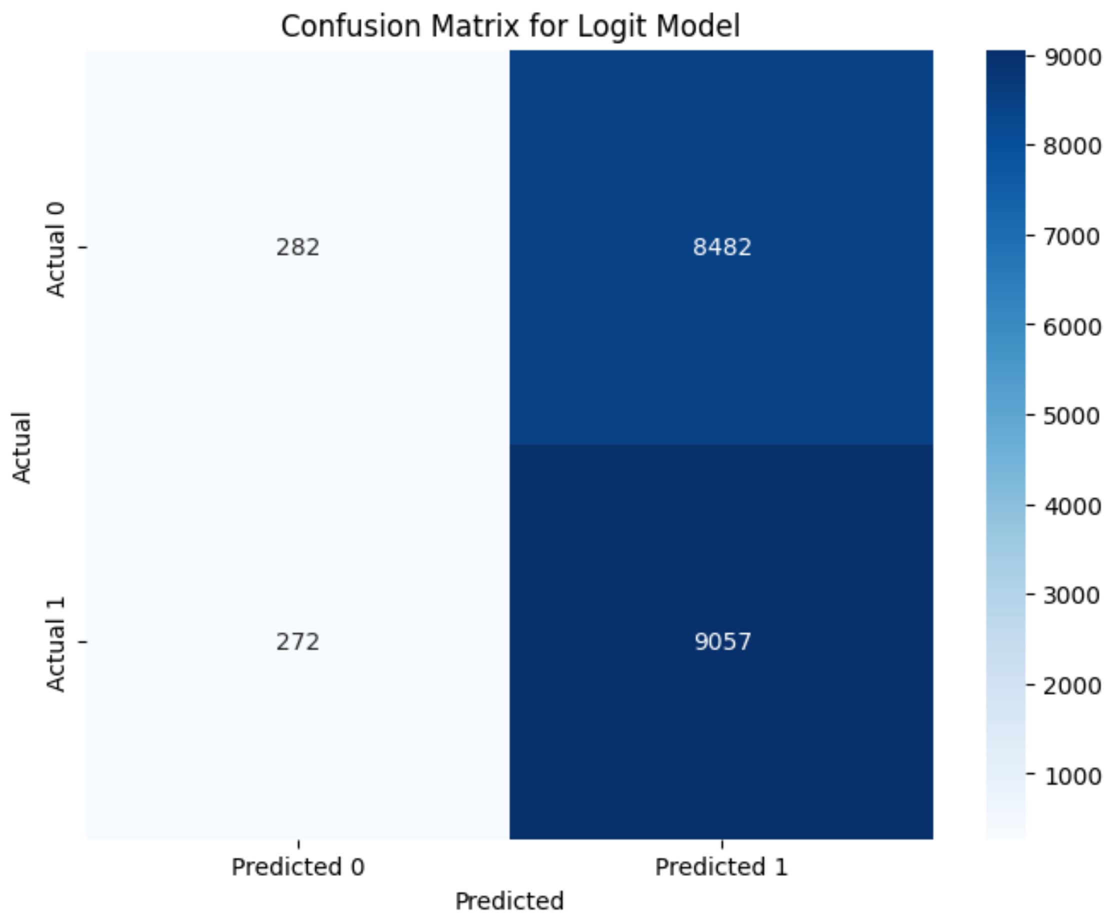
```
Optimization terminated successfully.
         Current function value: 0.692483
         Iterations 4
                           Logit Regression Results
==============================================================================
Dep. Variable:      favor_home_or_not   No. Observations:                18093
Model:                          Logit   Df Residuals:                    18087
Method:                           MLE   Df Model:                            5
Date:                Thu, 08 May 2025   Pseudo R-squ.:               0.0002552
Time:                        21:07:57   Log-Likelihood:                -12529.
converged:                       True   LL-Null:                       -12532.
Covariance Type:            nonrobust   LLR p-value:                    0.2696
================================================================================
============
                             coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
------------
const                      2.8170      1.477      1.907      0.057      -0.078
5.712
incorrect_calls           -0.0466      0.028     -1.685      0.092      -0.101
0.008
accuracy                  -0.0234      0.015     -1.580      0.114      -0.052
0.006
consistency               -0.0048      0.008     -0.616      0.538      -0.020
0.010
expected_incorrect_calls   0.0340      0.028      1.203      0.229      -0.021
0.089
accuracy_above_expected   -0.0497      0.044     -1.131      0.258      -0.136
0.036
================================================================================
============
```

[ ]:
```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

# Define the dependent and independent variables
X = umpire_1[['incorrect_calls', 'accuracy', 'consistency',␣
 ↪'expected_incorrect_calls', 'accuracy_above_expected']]
y = umpire_1['favor_home_or_not']

# Initialize the logistic regression model
```

```python
logreg = LogisticRegression(max_iter=1000) # Increased max_iter to ensure␣
↪convergence

# Perform 5-fold cross-validation
cv_scores = cross_val_score(logreg, X, y, cv=5)

# Print the cross-validation scores
print(cv_scores)

# Print the average cross-validation score
print("Average cross-validation score:", cv_scores.mean())
```

```
[0.51395413 0.51588837 0.51533573 0.51520177 0.51824212]
Average cross-validation score: 0.5157244235364273
```

[ ]:
```python
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Predict the probabilities for the test set
X = sm.add_constant(X)
y_pred_prob = logit_model.predict(X)

# Convert probabilities to class labels (0 or 1) using a threshold of 0.5
y_pred = (y_pred_prob > 0.5).astype(int)

# Create the confusion matrix
cm = confusion_matrix(y, y_pred)

# Plot the confusion matrix using seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for Logit Model')
plt.show()
```



[ ]:
```
accuracy = (cm[0, 0] + cm[1, 1]) / cm.sum()
print(f"Accuracy of the model: {accuracy}")
```

```
Accuracy of the model: 0.516166473221688
```

[ ]:
```
# Categorize 'total_run_impact' into 2 category using quantile

# Calculate quantiles for 'total_run_impact'
quantiles = umpire_1['total_run_impact'].quantile([0.5])

# Categorize 'total_run_impact' based on the median
umpire_1['total_run_impact_category'] = umpire_1['total_run_impact'].
↪apply(lambda x: 1 if x > quantiles[0.5] else 0)
```

[ ]:
```
quantiles
```

[ ]:
```
0.5    1.41
Name: total_run_impact, dtype: float64
```

[ ]:
```
umpire_1.head()
```

[ ]:
```
   id        date          umpire home away  home_team_runs  away_team_runs  \
0   1  2022-11-05  Lance Barksdale  HOU  PHI               4               1
1   2  2022-11-03    Jordan Baker  PHI  HOU               2               3
2   3  2022-11-02    Tripp Gibson  PHI  HOU               0               5
3   4  2022-11-01    Dan Iassogna  PHI  HOU               7               0
4   5  2022-10-29      Pat Hoberg  HOU  PHI               5               2

   pitches_called  incorrect_calls  expected_incorrect_calls  …  \
0           124.0              4.0                      10.0  …
1           149.0              6.0                       7.4  …
2           124.0              7.0                       7.1  …
3           140.0              5.0                       6.0  …
4           129.0              0.0                       8.7  …

   expected_correct_calls  correct_calls_above_expected  accuracy  \
0                   114.0                           6.0      96.8
1                   141.6                           1.4      96.0
2                   116.9                           0.1      94.4
3                   134.0                           1.0      96.4
4                   120.3                           8.7     100.0

   expected_accuracy  accuracy_above_expected  consistency  favor_home  \
0               92.0                      4.8         97.6        0.09
1               95.0                      0.9         97.3       -0.12
2               94.3                      0.1         92.7       -0.10
3               95.7                      0.7         92.9        0.63
4               93.2                      6.8         96.1        0.00

   total_run_impact  below_expected  total_run_impact_category
0              0.75               0                          0
1              0.58               0                          0
2              0.56               0                          0
3              0.73               0                          0
4              0.00               0                          0

[5 rows x 21 columns]
```

[ ]:
```
# Run a logistic regression using X as 'incorrect_calls' and y as␣
 ↪'total_run_impact_category'

import statsmodels.api as sm

# Define features (X) and target (y)
X = umpire_1['incorrect_calls']
X = sm.add_constant(X)
```

```
y = umpire_1['total_run_impact_category']

# Fit the logistic regression model
model = sm.Logit(y, X).fit()

# Print the model summary
print(model.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.390842
         Iterations 7
                              Logit Regression Results
================================================================================
=====
Dep. Variable:     total_run_impact_category   No. Observations:
18093
Model:                                 Logit   Df Residuals:
18091
Method:                                  MLE   Df Model:
1
Date:                       Mon, 28 Apr 2025   Pseudo R-squ.:
0.4361
Time:                               23:06:41   Log-Likelihood:
-7071.5
converged:                              True   LL-Null:
-12541.
Covariance Type:                   nonrobust   LLR p-value:
0.000
================================================================================
===
                      coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
---
const              -6.7466      0.101    -67.089      0.000      -6.944
-6.549
incorrect_calls     0.5916      0.009     67.444      0.000       0.574
0.609
================================================================================
===
```

[ ]:
```
# Add 'below_expected' column to the DataFrame
umpire_1['below_expected'] = umpire_1['correct_calls_above_expected'].
  ↪apply(lambda x: 1 if x < 0 else 0)
```

[ ]:
```
umpire_1.head()
```

```
[ ]:    id        date           umpire home away  home_team_runs  away_team_runs  \
     0   1  2022-11-05  Lance Barksdale  HOU  PHI               4               1
     1   2  2022-11-03     Jordan Baker  PHI  HOU               2               3
     2   3  2022-11-02     Tripp Gibson  PHI  HOU               0               5
     3   4  2022-11-01     Dan Iassogna  PHI  HOU               7               0
     4   5  2022-10-29       Pat Hoberg  HOU  PHI               5               2

        pitches_called  incorrect_calls  expected_incorrect_calls  …  \
     0           124.0              4.0                      10.0  …
     1           149.0              6.0                       7.4  …
     2           124.0              7.0                       7.1  …
     3           140.0              5.0                       6.0  …
     4           129.0              0.0                       8.7  …

        expected_correct_calls  correct_calls_above_expected  accuracy  \
     0                   114.0                           6.0      96.8
     1                   141.6                           1.4      96.0
     2                   116.9                           0.1      94.4
     3                   134.0                           1.0      96.4
     4                   120.3                           8.7     100.0

        expected_accuracy  accuracy_above_expected  consistency  favor_home  \
     0               92.0                      4.8         97.6        0.09
     1               95.0                      0.9         97.3       -0.12
     2               94.3                      0.1         92.7       -0.10
     3               95.7                      0.7         92.9        0.63
     4               93.2                      6.8         96.1        0.00

        total_run_impact  below_expected  total_run_impact_category
     0              0.75               0                          0
     1              0.58               0                          0
     2              0.56               0                          0
     3              0.73               0                          0
     4              0.00               0                          0

     [5 rows x 21 columns]
```

```
[ ]: # Run a logistic regression using X as 'below_expected', 'incorrect_calls' and
      ↪y as 'total_run_impact_category'

     import statsmodels.api as sm

     # Define features (X) and target (y)
     X = umpire_1[['below_expected', 'incorrect_calls']]
     X = sm.add_constant(X)
     y = umpire_1['total_run_impact_category']
```

```
# Fit the logistic regression model
model = sm.Logit(y, X).fit()

# Print the model summary
print(model.summary())
```

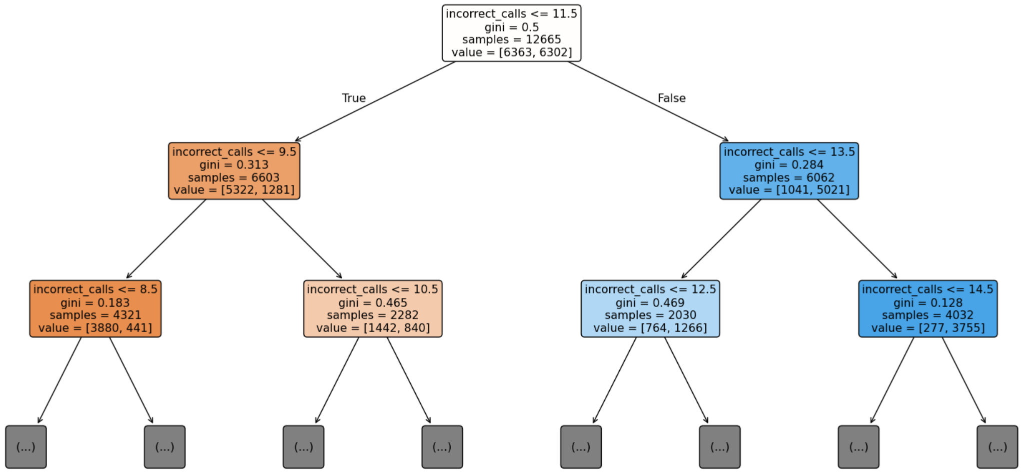
```
Optimization terminated successfully.
         Current function value: 0.390119
         Iterations 7
                              Logit Regression Results
================================================================================
=====
Dep. Variable:     total_run_impact_category   No. Observations:
18093
Model:                                 Logit   Df Residuals:
18090
Method:                                  MLE   Df Model:
2
Date:                       Mon, 28 Apr 2025   Pseudo R-squ.:
0.4372
Time:                               23:07:19   Log-Likelihood:
-7058.4
converged:                              True   LL-Null:
-12541.
Covariance Type:                   nonrobust   LLR p-value:
0.000
================================================================================
===
                      coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
---
const              -6.8716      0.104    -65.835      0.000      -7.076
-6.667
below_expected     -0.2385      0.047     -5.080      0.000      -0.330
-0.146
incorrect_calls     0.6120      0.010     62.756      0.000       0.593
0.631
================================================================================
===
```

[ ]:
```
# Create a decision tree using 'incorrect_calls', 'umpire', 'home', 'away' on
 ↪'total_run_impact_category'

import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
```

```python
from sklearn import metrics

# Define features (X) and target (y)
features = ['incorrect_calls', 'umpire', 'home', 'away']
X = umpire_1[features]
y = umpire_1['total_run_impact_category']

# Convert categorical features to numerical using one-hot encoding
X = pd.get_dummies(X, columns=['umpire', 'home', 'away'], drop_first=True)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=1) # 70% training and 30% test

# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

# Predict the response for test dataset
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy = metrics.accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
```

```
Accuracy: 0.77689756816507
```

[ ]:
```python
# Print the decision tree
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(20, 10))
plot_tree(clf, max_depth=2, filled=True, rounded=True, feature_names=X.columns)
plt.show()
```



[ ]:
```
grouped_umpire
```

[ ]:
```
             umpire home  incorrect_calls  total_run_impact
0         Adam Beck  ARI         6.000000          0.450000
1         Adam Beck  ATL         8.285714          1.364286
2         Adam Beck  BAL         8.000000          0.815000
3         Adam Beck  BOS         6.500000          0.750000
4         Adam Beck  CHC        11.000000          1.150000
…               …    …               …                 …
3316    Will Little  STL         8.636364          1.180909
3317    Will Little   TB         8.000000          0.960000
3318    Will Little  TEX         8.500000          0.993750
3319    Will Little  TOR         8.800000          1.048000
3320    Will Little  WSH        10.888889          1.456667

[3321 rows x 4 columns]
```

[ ]:
```
grouped_umpire_2 = umpire.groupby(['umpire', 'home',␣
↪'away'])[['incorrect_calls', 'total_run_impact']].mean().reset_index()
```

[ ]:
```
grouped_umpire_2['total_run_impact_category'] =␣
↪grouped_umpire_2['total_run_impact'].apply(lambda x: 1 if x > quantiles[0.5]␣
↪else 0)
```

[ ]:
```
grouped_umpire_2.head()
```

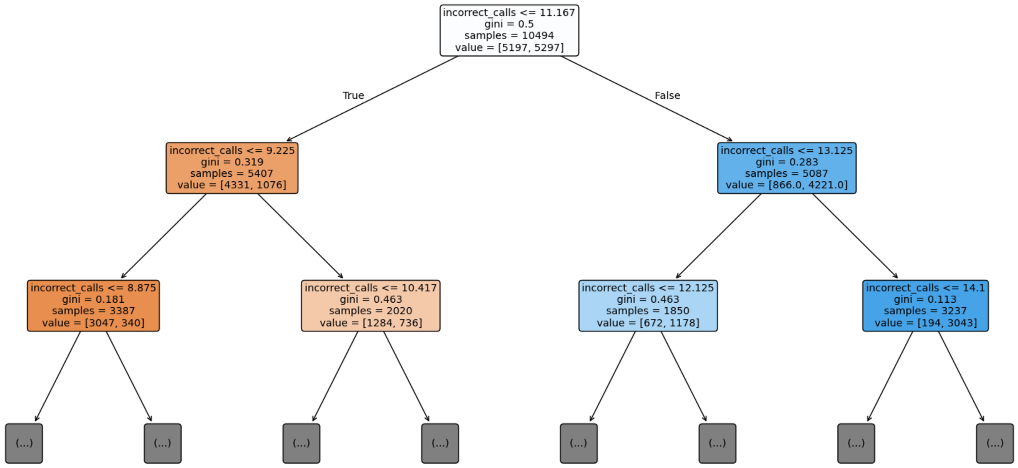
[ ]:
```
      umpire home away  incorrect_calls  total_run_impact  \
0  Adam Beck  ARI  LAD              6.0             0.450
```

```
1  Adam Beck  ATL  ARI          5.0           0.460
2  Adam Beck  ATL  BOS         10.5           1.985
3  Adam Beck  ATL  MIA          3.0           0.280
4  Adam Beck  ATL  NYM         13.0           2.760

   total_run_impact_category
0                          0
1                          0
2                          1
3                          0
4                          1
```

[ ]:
```
# Define features (X) and target (y)
features = ['incorrect_calls','umpire', 'home', 'away']
X = grouped_umpire_2[features]
y = grouped_umpire_2['total_run_impact_category']

# Convert categorical features to numerical using one-hot encoding
X = pd.get_dummies(X, columns=['umpire', 'home', 'away'], drop_first=True)

# Split data into training (70%) and testing (30%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=1)

# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

# Predict the response for test dataset
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy = metrics.accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
```

```
Accuracy: 0.7738995108937305
```

[ ]:
```
# Print the first few levels of the decision tree
plt.figure(figsize=(20, 10))
plot_tree(clf, max_depth=2, filled=True, rounded=True, feature_names=X.columns)
plt.show()
```



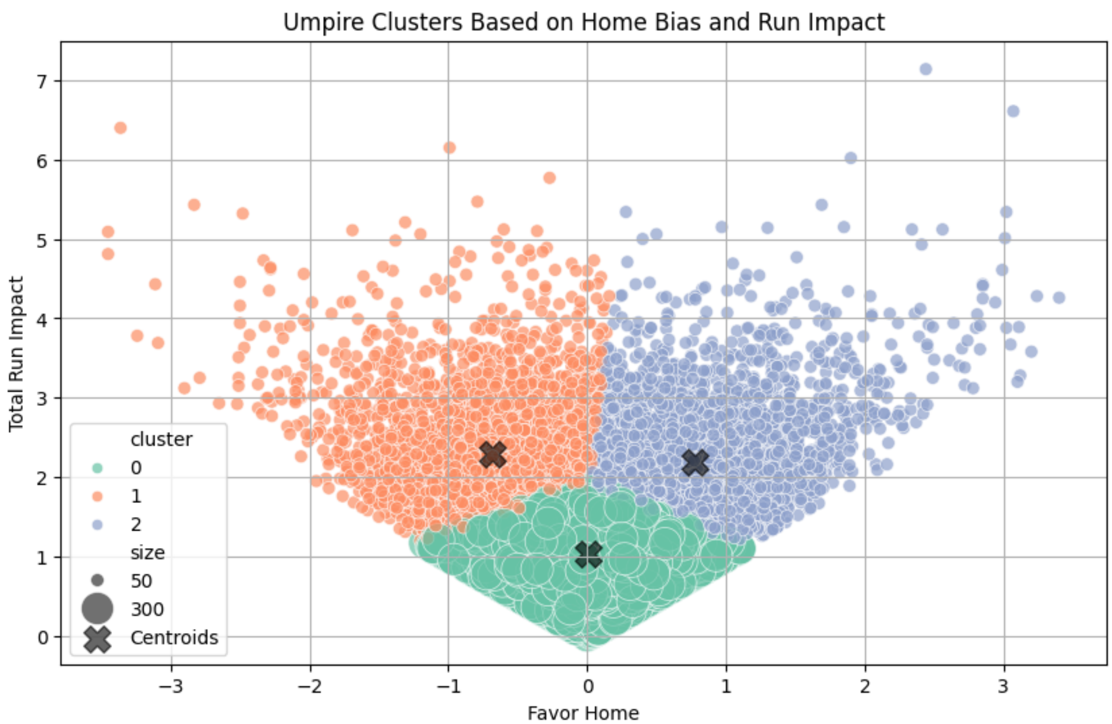
3.2 Unsupervised Learning

3.2.1 K-Means Clustering

[ ]:
```python
from sklearn.cluster import KMeans

# Select the relevant columns
df = umpire[['favor_home', 'total_run_impact']].copy()

# Apply KMeans clustering with 3 clusters
kmeans = KMeans(n_clusters=3, random_state=42)
df['cluster'] = kmeans.fit_predict(df)

# Create a 'size' column where Cluster 0 gets bigger circles and others are␣
↪smaller
df['size'] = df['cluster'].apply(lambda x: 300 if x == 0 else 50)
```

[ ]:
```python
import seaborn as sns

# Plot the clusters with the centroids for each cluster
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='favor_home', y='total_run_impact', hue='cluster',␣
↪palette='Set2', alpha=0.7, size='size', sizes=(50, 300))

# Plot centroids of the clusters
centroids = kmeans.cluster_centers_
plt.scatter(centroids[:, 0], centroids[:, 1], c='black', s=200, alpha=0.6,␣
↪marker='X', label='Centroids')
```

```
# Label the plot
plt.title('Umpire Clusters Based on Home Bias and Run Impact')
plt.xlabel('Favor Home')  # Positive values = home bias
plt.ylabel('Total Run Impact')  # Positive values = increased runs due to␣
↪umpire calls
plt.legend()
plt.grid(True)
plt.show()
```



[ ]:
```
# Get and print centroids
centroids = kmeans.cluster_centers_
print('Cluster centroids (favor_home, total_run_impact):')
for i, center in enumerate(centroids):
    print(f'Cluster {i}: Favor Home = {center[0]:.2f}, Total Run Impact =␣
↪{center[1]:.2f}') # Check the number of umpires in each cluster
cluster_counts = df['cluster'].value_counts()

# Cluster 0 (Neutral), Cluster 1 (Away team bias), Cluster 2 (Home team bias)
cluster_0_count = cluster_counts[0]
cluster_1_count = cluster_counts[1]
cluster_2_count = cluster_counts[2]
```

```
# Output the counts for comparison
print(f'Cluster 0 (Neutral): {cluster_0_count} umpires')
print(f'Cluster 1 (Away team bias): {cluster_1_count} umpires')
print(f'Cluster 2 (Home team bias): {cluster_2_count} umpires')
```

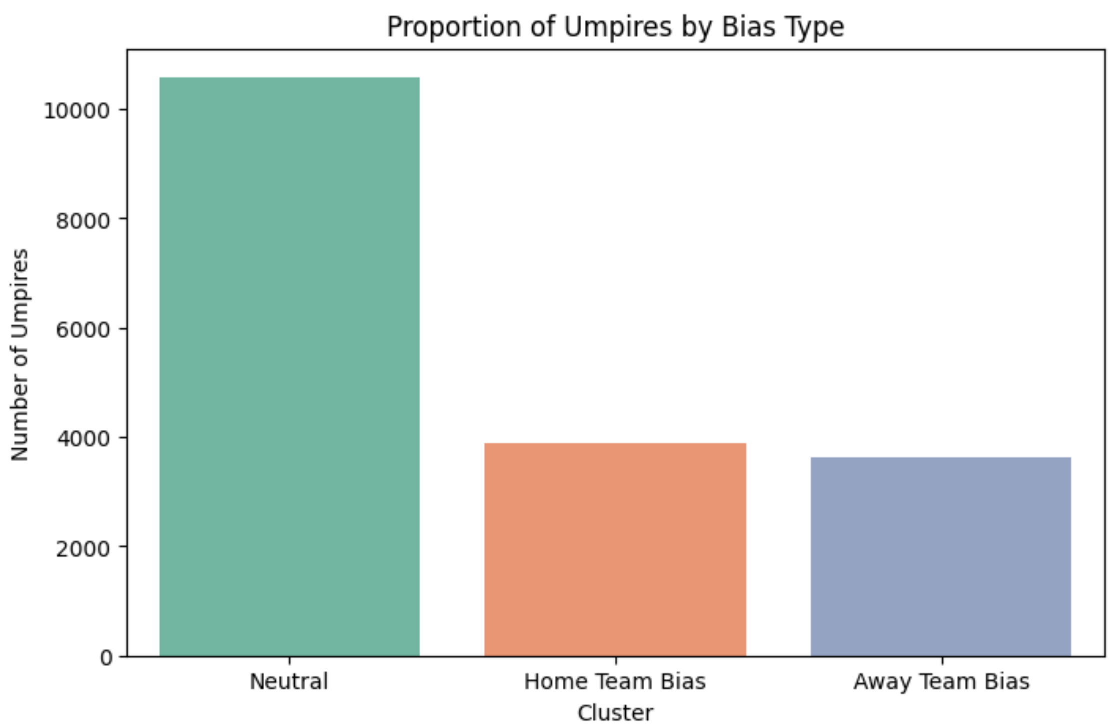
```
Cluster centroids (favor_home, total_run_impact):
Cluster 0: Favor Home = 0.01, Total Run Impact = 1.03
Cluster 1: Favor Home = -0.69, Total Run Impact = 2.28
Cluster 2: Favor Home = 0.78, Total Run Impact = 2.19
Cluster 0 (Neutral): 10577 umpires
Cluster 1 (Away team bias): 3617 umpires
Cluster 2 (Home team bias): 3899 umpires
```

[ ]:
```
# Calculate how many more umpires are in Cluster 2 compared to Cluster 1 and␣
↪Cluster 0
more_cluster_2_than_cluster_1 = cluster_2_count - cluster_1_count
more_cluster_2_than_cluster_0 = cluster_2_count - cluster_0_count

print(f'Cluster 2 has {more_cluster_2_than_cluster_1} more umpires than Cluster␣
↪1')
print(f'Cluster 2 has {more_cluster_2_than_cluster_0} more umpires than Cluster␣
↪0')
```

```
Cluster 2 has 282 more umpires than Cluster 1
Cluster 2 has -6678 more umpires than Cluster 0
```

[ ]:
```
# Calculate the number of umpires in each cluster
cluster_sizes = df['cluster'].value_counts()

# Define cluster names
cluster_names = {0: 'Neutral', 1: 'Away Team Bias', 2: 'Home Team Bias'}

# Map the clusters to their names
df['cluster_name'] = df['cluster'].map(cluster_names)

# Plot the proportions of each cluster
plt.figure(figsize=(8, 5))
sns.countplot(data=df, x='cluster_name', hue='cluster_name', palette='Set2',␣
↪order=['Neutral', 'Home Team Bias', 'Away Team Bias'])
plt.title('Proportion of Umpires by Bias Type')
plt.xlabel('Cluster')
plt.ylabel('Number of Umpires')
plt.show()

# Calculate the percentage of each cluster
total_umpires = df.shape[0]
cluster_percentages = cluster_sizes / total_umpires * 100
```

```
# Print the percentage of umpires in each cluster
print('\nPercentage of Umpires in Each Cluster:')
for cluster, size in cluster_sizes.items():
    print(f'{cluster_names[cluster]}: {size} umpires␣
 ↪({cluster_percentages[cluster]:.2f}%)')

# Get the counts of each cluster
cluster_sizes_names = df['cluster_name'].value_counts()

# Print the cluster sizes
print(f'Cluster Sizes (Umpires in each cluster):')
print(cluster_sizes_names)
```




```
Percentage of Umpires in Each Cluster:
Neutral: 10577 umpires (58.46%)
Home Team Bias: 3899 umpires (21.55%)
Away Team Bias: 3617 umpires (19.99%)
Cluster Sizes (Umpires in each cluster):
cluster_name
Neutral           10577
Home Team Bias     3899
Away Team Bias     3617
```

```
Name: count, dtype: int64
```

[ ]:
```
from scipy.stats import chi2_contingency

# Check if 'cluster_sizes' is a Series and access it using indices
if isinstance(cluster_sizes, pd.Series):
    # Use indices to access values
    observed = [[cluster_sizes[2], cluster_sizes[1]]]
    expected = [[(cluster_sizes[2] + cluster_sizes[1]) / 2, (cluster_sizes[2] +␣
 ↪cluster_sizes[1]) / 2]]
else:
    # For DataFrame, use column names
    observed = [[cluster_sizes['Home Team Bias'], cluster_sizes['Away Team␣
 ↪Bias']]]
    expected = [[(cluster_sizes['Home Team Bias'] + cluster_sizes['Away Team␣
 ↪Bias']) / 2,
                 (cluster_sizes['Home Team Bias'] + cluster_sizes['Away Team␣
 ↪Bias']) / 2]]

# Perform chi-squared test
p_value = chi2_contingency([observed[0], expected[0]])[1]
print(f'P-value: {p_value}')
print('Smaller than 0.05 = significant')
```

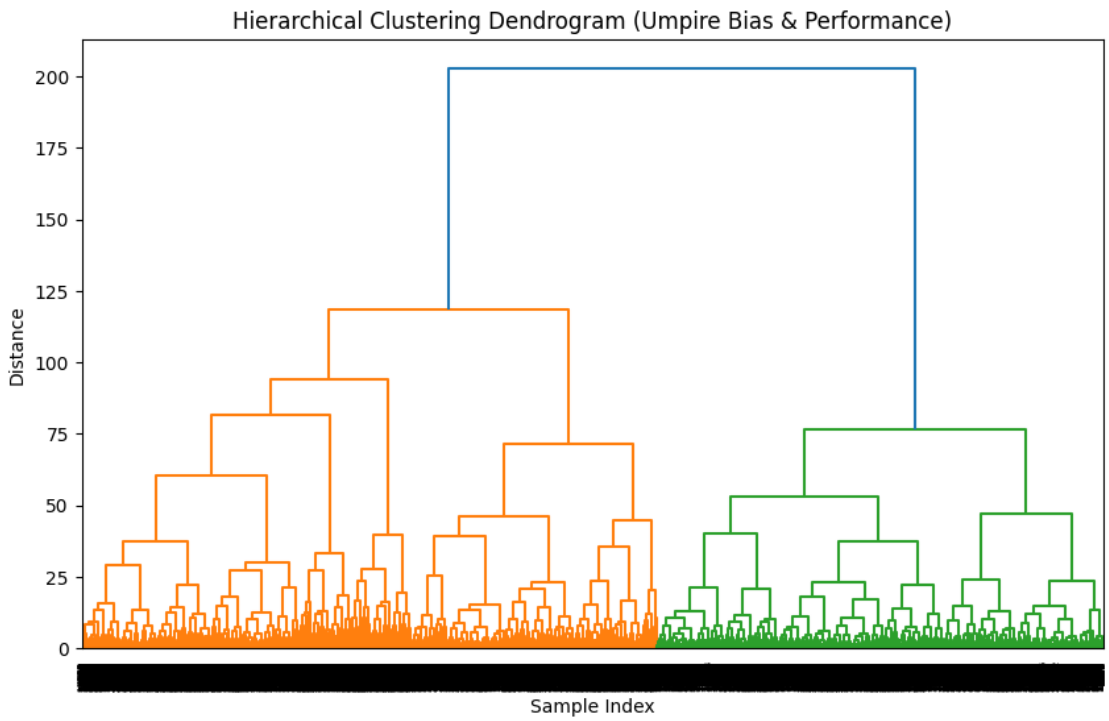
```
P-value: 0.022362264939571504
Smaller than 0.05 = significant
```

3.2.2 Hierarchical Clustering

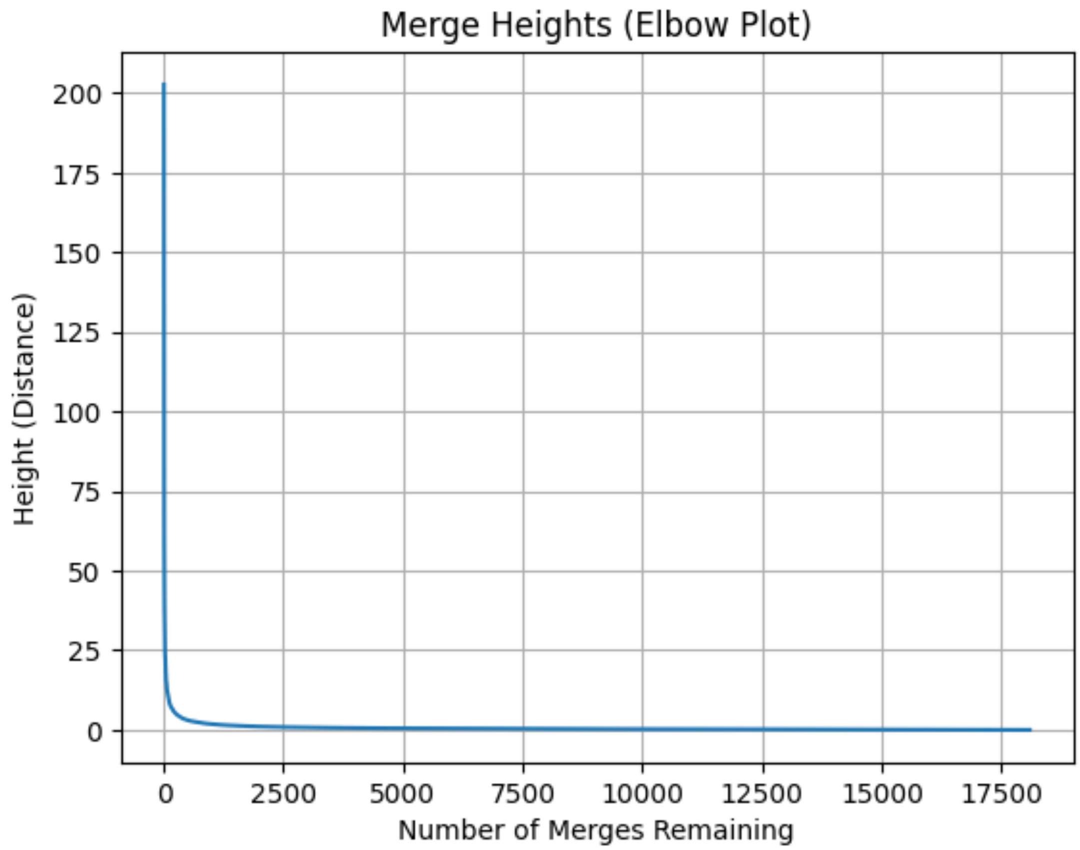
[ ]:
```
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import dendrogram, linkage
from scipy.cluster.hierarchy import fcluster
```

[ ]:
```
# Select relevant columns
cluster_data = umpire[['favor_home', 'total_run_impact', 'accuracy',␣
 ↪'consistency']].copy()

# Standardize the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(cluster_data)

# Perform hierarchical clustering
linked = linkage(scaled_data, method='ward')

# Plot the dendrogram
plt.figure(figsize=(10, 6))
```

```
dendrogram(linked, orientation='top', distance_sort='descending',␣
  ↪show_leaf_counts=False)
plt.title('Hierarchical Clustering Dendrogram (Umpire Bias & Performance)')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()
```



[ ]:
```
# Get the heights at each merge
merge_heights = linked[:, 2]  # 3rd column = distance at each merge

# Plot the merge heights
plt.plot(range(1, len(merge_heights) + 1), merge_heights[::-1])
plt.title('Merge Heights (Elbow Plot)')
plt.xlabel('Number of Merges Remaining')
plt.ylabel('Height (Distance)')
plt.grid(True)
plt.show()
```



The merge heights drop dramatically after the first few merges, then flatten. After that, there's very little distance between clusters.

Therefore, the dataset can be splitted into 2–3 clusters. Beyond 3, any additional clusters are very close together (small differences).

[ ]:
```
from scipy.cluster.hierarchy import fcluster

# For 2 clusters
clusters_2 = fcluster(linked, 2, criterion='maxclust')

# For 3 clusters
clusters_3 = fcluster(linked, 3, criterion='maxclust')
```

Given the dataset of over 18,000 games, 3 clusters seem better to capture importance differences.

[ ]:
```
# Add cluster labels back
cluster_data['cluster'] = clusters_3
```

[ ]:
```
cluster_data
```

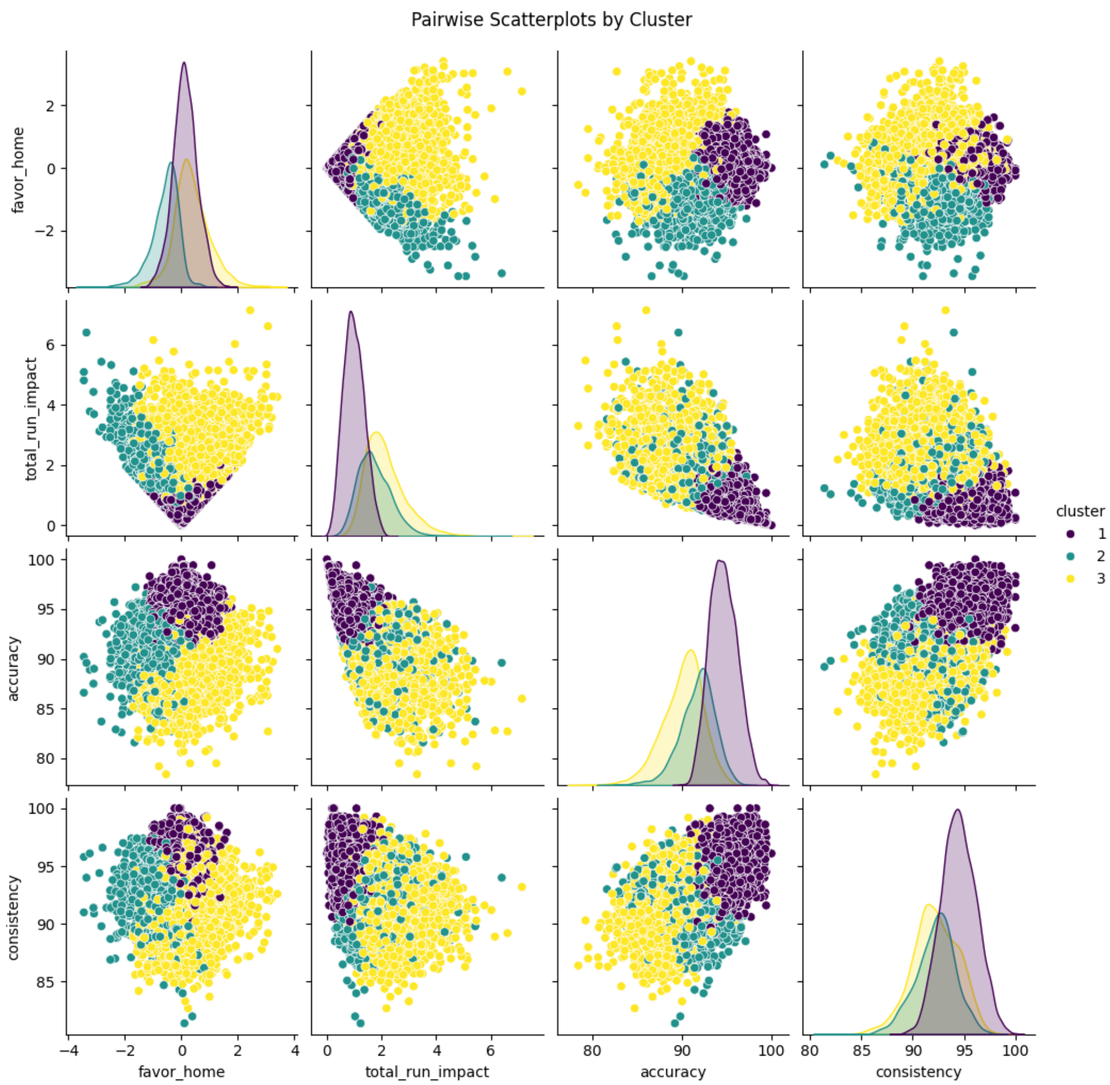
[ ]:

	favor_home	total_run_impact	accuracy	consistency	cluster
0	0.09	0.75	96.8	97.6	1
1	-0.12	0.58	96.0	97.3	1
2	-0.10	0.56	94.4	92.7	1
3	0.63	0.73	96.4	92.9	1
4	0.00	0.00	100.0	96.1	1
…	…	…	…	…	…
18208	-0.28	1.42	90.8	93.8	2
18209	0.51	1.97	83.1	91.9	3
18210	-0.40	2.44	88.3	87.2	3
18211	-0.36	0.84	93.9	94.6	1
18212	1.10	4.36	89.2	94.9	3

[18093 rows x 5 columns]

[ ]:

```
import seaborn as sns

sns.pairplot(cluster_data, vars=['favor_home', 'total_run_impact', 'accuracy',␣
↪'consistency'], hue='cluster', palette='viridis')
plt.suptitle('Pairwise Scatterplots by Cluster', y=1.02)
plt.show()
```



[ ]:
```python
# Compute the average for each cluster
cluster_data.groupby('cluster').mean()
```

[ ]:

cluster	favor_home	total_run_impact	accuracy	consistency
1	0.125919	0.976895	94.589246	94.445686
2	-0.582561	1.743164	91.673700	92.192913
3	0.369925	2.126765	90.043539	92.172754