

---

# Predict 2028 Olympics Medal Table: Hierarchical Adaptive Prediction Framework with LSTM-Attention Mechanism

## Summary

In this study, we explored the dynamics of Olympic medal predictions and the specific manifestations of the great coach effect. By leveraging historical data and advanced modeling techniques, we developed two key models: one for predicting the gold medal and overall medal counts for the 2028 Los Angeles Summer Olympics which is called **HAPF-LSTM**, and another for quantifying the impact of exceptional coaches on medal outcomes. Our analysis not only provided insights into potential medal distributions, aiding each nation's Olympic Committee in strategic adjustments but also offering investment recommendations in **great coach effect** for three countries.

The first model provided a detailed **hierarchical adaptive** forecast for the 2028 Olympics, identifying medal distribution patterns and performance trends among nations. By integrating historical data and **feature engineering**, we quantified characteristics like dominance, and fierceness of competitions, which came from either nation's and certain Olympic year's aspects. The model **progressively** uses the main data group by going through **random-forests-classifier** for predictions of whether will achieve a medal and then going through the regressor powered by **MLP**, with the collaboration of additional important data group. Differences in going through **various branches** mainly depend on a **timely judgment** of whether a country is competitive enough. **LSTM-Attention mechanism** is introduced to the model to drive the training and evaluation. The model offered insights into countries that are steadily competitive and those hardly having achieved any medals. These predictions not only informed audience expectations but also provided strategic guidance to national Olympic committees for refining their sports development strategies.

The second model, leveraging a data-driven approach, confirmed the existence of the **great coach effect** and quantified its influence on medal counts. The study highlighted the significant role that top-level coaches play in enhancing medal performance. Furthermore, it identified three additional factors, **Sports Powerhouse**, **Stable Athlete Pool** and **Historical Success** which are based on insights from relevant research, that can help determine which sporting events would benefit most from the investment in a great coach. This **quantitative analysis** highlighted the significant impact of exceptional coaching on athletic success, specifically identifying Sailing in Great Britain, Boxing in the United States, and Trampoline Gymnastics in China as key areas for improvement.

By integrating various methods and incorporating diverse variables, our model demonstrates robustness and stability, ensuring reliable performance even with noisy or incomplete data. Future research could build on these findings by incorporating additional variables, such as athlete development programs, technological advancements, and socio-economic factors, to further refine predictions and deepen insights.

**Keywords:** Olympics, medal estimation, hybrid model, Random Forest, LSTM, Attention, MLP, great coach effect

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Problem Background	1
1.2	Restatement of the Problem	1
1.3	Literature Review	2
1.4	Our Work	3
<b>2</b>	<b>General Assumptions and Notations</b>	<b>3</b>
2.1	Assumptions	3
2.2	Notations	4
<b>3</b>	<b>Data Preprocessing and Analysis</b>	<b>4</b>
3.1	Data Preprocessing	4
3.2	Data Analysis	5
<b>4</b>	<b>HAPF-LSTM: Hierarchical Adaptive Prediction Framework Driven by LSTM-Attention Mechanism</b>	<b>8</b>
4.1	Feature Selection	8
4.1.1	Feature for Participating Nation	8
4.1.2	Feature for certain Olympic year	9
4.1.3	Features Overview	10
4.2	Main body of HAPF-LSTM	11
4.2.1	Overview	11
4.2.2	Data Grouping and engineering	11
4.2.3	Core Inner Structure: Hierarchical Adaptive Prediction Framework	12
4.2.4	Temporal prediction: LSTM-Attention Driven mechanism	14
4.3	Result	14
<b>5</b>	<b>Great Coach Effect Analysis</b>	<b>14</b>
5.1	Great Coach Effect Quantification	14
5.2	Investment Suggestions	16
<b>6</b>	<b>Other Insight and Suggestions</b>	<b>18</b>
<b>7</b>	<b>Model Evaluation</b>	<b>19</b>
7.1	Sensitivity Analysis	19
7.2	Robustness Analysis	20
<b>8</b>	<b>Strengths and Weaknesses</b>	<b>20</b>
8.1	Strengths	20
8.2	Weaknesses	21
<b>9</b>	<b>Conclusion</b>	<b>21</b>
	<b>References</b>	<b>22</b>

<b>A</b>	<b>Prediction medal table for 2028 Los Angeles, USA summer Olympics . . . . .</b>	<b>23</b>
<b>B</b>	<b>Additional Code . . . . .</b>	<b>25</b>
<b>C</b>	<b>Report on Use of AI . . . . .</b>	<b>88</b>

# 1 Introduction

## 1.1 Problem Background

The Olympic Games represent a unique global spectacle, featuring athletes from more than 200 countries competing in more than 400 events in both the Summer and Winter Games. This prestigious multi-sport event not only showcases competition but also fosters inspiration and unity among participants and spectators worldwide [5].

During the 2024 Summer Olympics, Paris became the center of sport enthusiasts around the world. Spectators focused not only on the outcomes of individual events but also on the national rankings in the medal tally. The United States and China performed notably, each securing an equal number of gold medals, demonstrating their formidable prowess in global sports. Meanwhile, the host nation France, although ranked fifth in gold medals, secured the fourth position in total medals, suggesting that the home advantage may have positively influenced their performance. Additionally, this Olympics witnessed breakthroughs by smaller nations such as Saint Lucia and Albania, which not only won Olympic medals for the first time but also captured gold, marking significant progress in their sporting achievements. This phenomenon indicates that the Olympics not only serve as a competitive arena for sports powerhouses but also provide a platform for smaller nations to showcase their developmental strides.

With the progression of globalization and the rise of social media, there is an increasing focus on sports events, especially massive competitions like the Olympics [3]. Medal predictions for the Olympics not only satisfy the curiosity of the audience but also offer substantive guidance to national Olympic committees to improve their countries' standings on the medal table. Predictions are typically based on the athletes' capabilities and past performances, yet the outcomes of the Games are often filled with uncertainties. This unpredictability not only enhances the spectacle of the competitions but also makes forecasting a challenging task.

## 1.2 Restatement of the Problem

- **Question 1:**

The first question can be divided into three sub-questions. The first sub-question involves predicting the medal tally for the 2028 Los Angeles Summer Olympics, including all result intervals, and analyzing which countries are likely to perform better or worse compared to 2024. The second sub-question estimates the number of countries that might win their first Olympic medals in 2028 and discusses the likelihood of these predictions. The final sub-question explores how the number and type of events influence the distribution of medals among competing nations, identifies key sports crucial to different countries and the reasons why, and assesses how the selection of events by the host nation might impact the outcomes of the games.

- **Question 2:**

Through data analysis, investigate whether there are performance changes that can be attributed to the influence of great coaches. Please evaluate the extent of this impact on medal count. Subsequently, select three countries and analyze which sports could benefit from investing in

top-level coaches, and estimate the potential effectiveness of this strategy.

- **Question 3:**

Analyze the Olympic medal counts predicted by the existing model and provide unique insights to inform the national Olympic committees.

### 1.3 Literature Review

With the rapid development of social media and the growing aspiration of people for a higher quality of life, the Olympic Games, an ancient and prestigious sporting event, have captured unprecedented levels of global attention in the 21st century [5]. As a result, predicting the Olympic medal counts each year has also become a focal point for audiences.

For each Olympic Games, a significant amount of literature provides predictions. For example, Sanchez-Fernandez et al. describe the process of predicting the distribution of Olympic medals. By analyzing the accuracy of predictions made for the 2012 London Olympics, it further predicts the medal distribution for the 2016 Rio Olympics and provides the medal rankings for each country [8]. There is another study that aimed to predict the success of countries participating in the Tokyo 2020 Olympics using smart methods, including the PEST+S model and neural network modeling [4]. Nagpal's team mainly discusses the status of the Olympics as a top international sporting event and uses data from the 2020 Tokyo Olympics to predict the medal counts for the 2024 Paris Olympics [7]. The above studies focused on predicting medal counts for the 2016 Rio Olympics, the 2020 Tokyo Olympics, and the 2024 Paris Olympics. From this, we can observe that forecasting the distribution of gold medals for each edition of the Olympic Games has become a consistent and enduring tradition. This practice reflects not only the audience's fascination with the competitive outcomes but also the broader interest in analyzing patterns, trends, and factors that influence athletic success on the global stage [5]. Such predictions serve as a bridge between historical performance and future expectations, enriching the Olympic experience for fans, researchers, and stakeholders alike.

Scelles et al. provide a wealth of recommendations on feature selection for predicting the gold medal tally in his paper. Including the population size of a country four years prior to the Olympic Games, the country's economic development, the host country effect, country regions, medal totals four years earlier and so on [9]. For prediction model selection, Varagiri Shailaja's team experimented with some common algorithm including Decision Tree, KNN, Linear Regression, Random Forest Algorithm and Bayesian Ridge Algorithm [11]. Moreover, Edward M. Condon built linear regression models and neural network models and afterwards compared them, indicating the neural network model outperformed the regression model [3]. Christoph Schlembach proposed a socioeconomic machine learning model including Tobit model and Hurdle model [10]. The Tobit model is a type of regression model designed to handle censored dependent variables, where the observed values are either partially or entirely limited by a threshold (e.g., non-negative variables like income or scores) [1]. It is commonly used in cases where the dependent variable is restricted in range, allowing for the estimation of relationships while accounting for the censoring effect, providing more accurate and meaningful results than standard linear regression [6]. The above literature on variable selection and model choice provides a solid foundation for our work.

## 1.4 Our Work

We began by collecting relevant literature on the topic, learning from previous research on medal prediction. We are initiating the processes of data pre-processing and data analysis. After that, we proceeded to build our models.

The first model, Hierarchical Adaptive Prediction Framework Driven by LSTM-Attention Mechanism (HAPF-LSTM), is designed to predict the gold and overall medals for 2028 Olympics. We analyzed data from two perspectives: the characteristics of participating countries and the features of events in each Olympic year. Participating-nation-factors such as host nation status, athlete quality, and event dominance, as well as certain-Olympic-year-features like the competition intensity and the number of events, were integrated into different parts of the model for prediction. This approach achieved strong predictive performance. To account for differences between countries, we designed specialized model branches for those with low medal prospects and consistently top-ranked nations, allowing for tailored prediction strategies. Additionally, we applied the 'LSTM + Attention' mechanism to capture temporal features and fine-tuned the model with the Adam optimizer, leading to satisfactory results.

The second model is capable of accurately reflecting the great coach effect on the medal standings. We identified three additional great coaches and analyzed the changes in medal counts before and after the five coaches began coaching and the local highest value in medal count during the coaching period. These data allowed us to assess the impact of these exceptional coaches on specific events across various countries. Additionally, by examining relevant research, we identified three other critical factors that significantly influence the effectiveness and value of investing in a coach for a particular sport: Sports Powerhouse, Stable Athlete Pool and Historical Success. Using the three features we filtered the most suitable sports of coach investment.

Evaluation, strengths and weaknesses of the proposed model are discussed afterward. All source codes and the prediction medal table for 2028 Los Angeles, USA summer Olympics are attached to the Appendix.

## 2 General Assumptions and Notations

### 2.1 Assumptions

1. It is assumed that the collected data is authentic and reliable, with no instances of fabrication.
2. It is assumed that athletes from all countries compete with integrity, without resorting to substances such as doping or high-tech aids beyond the rules (such as specially designed clothing or sports equipment), and that no cheating occurs.
3. For sports involving referees, it is assumed that referees are fair and impartial, and there will be no intentional misjudgments, omissions, or favoritism.
4. The possibility of major sporting nations refusing to participate in the Olympics for various reasons is not considered, nor is the absence of athletes due to injuries or other reasons.

5. The impact of the International Olympic Committee's temporary changes to the competition rules on the performance of athletes is not considered.

## 2.2 Notations

Symbol	Description
$TS_{i,j}$	Total score of the $i^{\text{th}}$ country in the $j^{\text{th}}$ sport.
$S_{i,j}$	Score rate of the $i^{\text{th}}$ country in the $j^{\text{th}}$ sport.
$a_{i,j}$	Number of athlete of the $i^{\text{th}}$ country in the $j^{\text{th}}$ sport.
$Q_{i,j}$	Athlete quality of the $i^{\text{th}}$ country in the $j^{\text{th}}$ sport.
$m_j$	Number of events in the $j^{\text{th}}$ sport
$n_{Gold}$	Number of gold medals
$n_{Silver}$	Number of silver medals
$n_{Bronze}$	Number of bronze medals
$P_{1s}$	Score achieved by the team under the coach's leadership during Period 1
$\Delta_{pre1}$	Difference in score compared to the previous term
$\Delta_{max1}$	Difference in score compared to the best performance among the previous three terms
$P_{2s}$	Score achieved by the team under the coach's leadership during Period 2
$\Delta_{pre2}$	Difference in score compared to the previous term
$\Delta_{max2}$	Difference in score compared to the best performance among the previous three terms
$\delta$	If it is a team event

## 3 Data Preprocessing and Analysis

### 3.1 Data Preprocessing

After reviewing the provided dataset, we find that only one file which is called `summerOly_programs.csv` contains some missing or anomalous data points. First, we address the missing values by replacing all of them with a question mark. Following that, we handle outliers, which includes removing irrelevant information, deleting columns related to competitions outside the Olympic cycle, and eliminating rows with incomplete data. In the 'Sports Governing Body' column, replace question marks with NULL, and then substitute any other occurrences of question mark with the number 0. After completing these operations, we recalculate the totals for events, disciplines, and sports and correct the data accordingly.

After addressing outliers and missing values, we further consolidated the data. First, we removed the years from the `summerOly_hosts.csv` file for which the Olympic Games were not successfully held. Next, we delete the contents within parentheses to standardize the format of the host location names and use regular expressions to extract text and ensure there are no extra spaces. Finally, we used `pandas` to split the city and country names, then linked these with the `summerOly_medal_counts.csv` and `summerOly_athletes.csv` tables to obtain the final clean and usable data.

Furthermore, there are 18 kinds of sports events that have not occurred in the past 10 years which are then deemed outdated and are therefore all removed. For example, Art competition, Alpinism, Aeronautics and so on. The naming convention of the 'sport' column in the 'athletes' table should be consistent with that in the 'programs' table.

### 3.2 Data Analysis

Through data analysis and visualization, we have created the two charts above to illustrate the trends in event count changes in sports over the years. We removed events with a count of 0, indicating that these sports were not successfully held.

Figure 1 offers a macro view of how the number of events for various sports at the Summer Olympics has evolved from 1896 to 2024. It is evident that certain events experienced significant growth in specific years, particularly newly introduced or developing sports, which reflect a more diversified trend. The scattered points and line segments signify that these events were short-lived and did not continue to appear in subsequent Olympic Games.

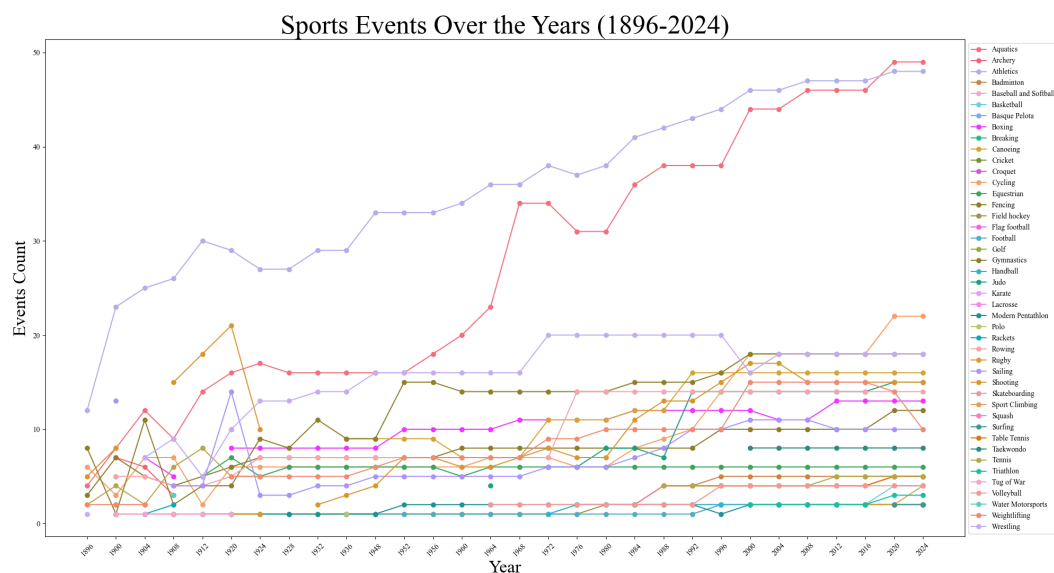


Figure 1: Sports Events Overview for The Last 30 Summer Olympic Games

Figure 2 compares the event count changes from the first 15 editions of the Olympics (1896-1964) with those from 1968 to 2024. It is clear that the first 15 editions showed more fluctuation in event



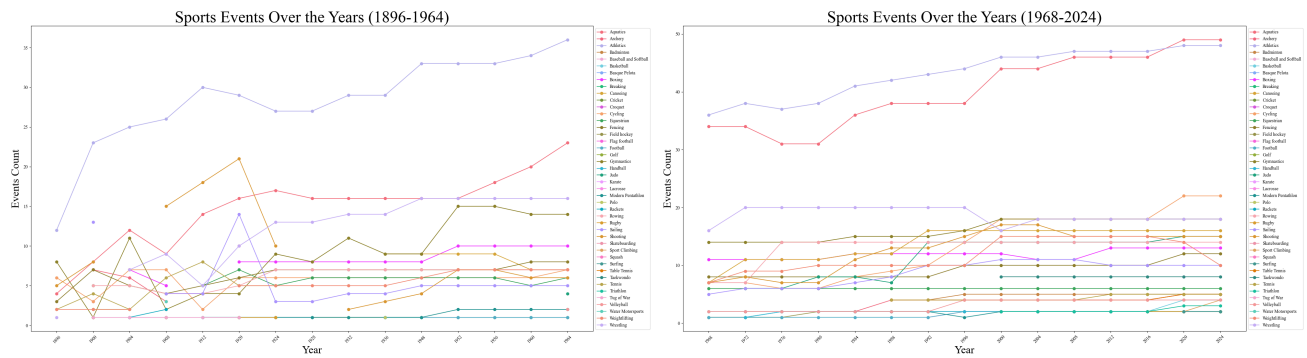


Figure 2: Comparison of Sports Events Count between 1896-1964 and 1968-2024

numbers, suggesting that the selection of sports was still in development. In contrast, the number of events in the subsequent 15 editions became more stable, indicating that the Olympic Games had reached a more mature stage. So, we brought only the second half of the Olympic years into analysis and prediction.

To explore the hosting effect, we examined changes in the event list based on different hosts and whether these align with the host's dominant sports. We also analyzed performance fluctuations when a country is the host, considering current, past, and future hosting scenarios. We subsequently randomly selected a year and the corresponding host country (1908, UK) and conducted a data visualization analysis of the host effect. Finally, we obtained the following bar chart.

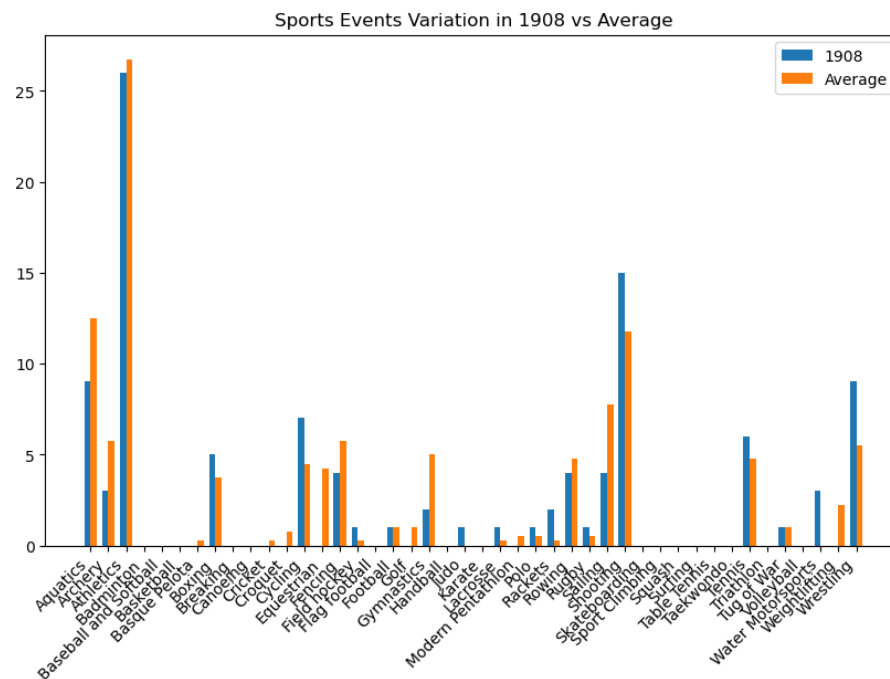


Figure 3: Visualization of the Hosting-Effect

The chart shows a comparison between the number of sports events in 1908 and the average number. The horizontal axis lists different sports, and the vertical axis represents the number of events. The blue bars represent the data from 1908, while the orange bars represent the average. From the chart, it can be seen that the number of events for certain sports in 1908 was significantly higher than the average, such as athletics and cycling, while for other sports it was lower than the average, such as baseball and basketball. This chart explains the distribution of sports events when the UK was the host in 1908, and the differences in the distribution of sports events on average when other countries were hosts. Upon further observation, we find that the host country does have some influence on the selection of sports events.

The Fourier Transform Distance is a metric method based on Fourier transformation to measure the similarity or difference between two images. The Shape Based Distance, on the other hand, assesses the similarity or difference between two objects or shapes by comparing their geometric structures. As shown in the table, we can observe the Fourier Transform Distance and Shape Based Distance data for each Olympic host country from 1896 to 2032. By analyzing these data, we can derive a coefficient table of change to reflect the trends of variation for each year. For instance, when Belgium (BEL) was the host in 1920, its Fourier Transform Distance was 117.1523775, and its Shape Based Distance was 18.13025706, indicating a significant change in the distribution of sports events compared to the average. Similarly, when Japan (JPN) was the host in 2020, its Fourier Transform Distance was 72.37198119, and its Shape Based Distance was 11.45689952, also showing a marked difference compared to the average. These data can help us understand the impact of different countries as hosts on the distribution of Olympic sports events and the trends of change between different years.

Table 1: Table of Fourier Transforms and Shape-Based Distances for Different Years and Countries

Year	NOC	Fourier Transform	Shape Based Distance
1896	GRE	12.44897663	2.551577259
1900	FRA	83.26921768	13.05903229
1904	USA	73.66071638	11.67650102
1908	GBR	41.39282408	6.841051889
1912	SWE	51.4455513	8.341526382
1920	BEL	117.1523775	18.13025706
1924	FRA	30.2210292	5.095189278
1928	NED	3.015353154	0.931077955
1932	USA	11.33551053	3.295729732
1936	GER	27.45071305	4.452446289
1948	GBR	17.64138247	2.98867595
1952	FIN	23.70182931	4.133668247
1956	AUS	13.7343408	2.582847666
1960	ITA	2.542094369	0.872093177
1964	JPN	14.84975411	2.667966737
1968	MEX	42.7871375	7.042558737
1972	FRG	34.78965986	5.577291953
1976	CAN	21.88883168	3.855519899
1980	URS	11.6411997	2.222258114

Year	NOC	Fourier Transform	Shape Based Distance
1984	USA	10.98803475	2.21474652
1988	KOR	23.57418575	4.11547974
1992	ESP	26.77025262	4.347124447
1996	USA	18.67467862	3.142705704
2000	AUS	41.075838	6.519716669
2004	GRE	17.4431855	2.960624485
2008	CHN	7.51287561	1.742737543
2012	GBR	4.764737058	1.290237613
2016	BRA	10.51872984	1.968057019
2020	JPN	72.37198119	11.45689952
2024	FRA	30.59058187	5.222303253
2028	USA		
2032	AUS		

## 4 HAPF-LSTM: Hierarchical Adaptive Prediction Framework Driven by LSTM-Attention Mechanism

### 4.1 Feature Selection

According to the requirements, only the provided data sets are available for our training and estimation. Due to the lack of consecutive data and the limited amount of resources, we had to make great efforts on characteristic selection and engineering. In our work, we investigated these data sets in two aspects, participating nations and the Olympic year itself. Both obvious and strenuous features are included.

#### 4.1.1 Feature for Participating Nation

##### A) Typical-strenuous-feature: Total Score, Score Rate and Athlete Quality

The total score  $TS_{i,j}$  for  $i^{th}$  country in  $j^{th}$  sport event is determined by the number of medals there, with gold medals worth 3 points, silver medals worth 2 points, bronze medals worth 1 point, and no medal earns 0 points. One special sport event is expressed by the following equation:

$$TS_j = \sum_i 3 \times n_{Gold,i,j} + 2 \times n_{Silver,i,j} + n_{Bronze,i,j} \quad (1)$$

The score rate  $S_{i,j}$  for a country in such a specific kind of sport is calculated by dividing the total score  $TS_{i,j}$  by the total number of events under such  $j^{th}$  sport  $m_j$  that the country participated in during the year. This score is then multiplied by 6 to standardize the score to a range that reflects the relative dominance level of the country. The formula is as follows:

$$S_{i,j} = \frac{TS_{i,j}}{m_j \times 6} \quad (2)$$

This score rate provides an effective metric to assess the country's overall performance in such a specific sport over the year. It normalizes the performance, accounting for the number of events participated in, and gives an insight into how well the country performed relative to others. Similarly, the athlete quality  $Q_{i,j}$  for a certain country is quantified as the total score achieved by  $i^{th}$  country and their amount of athletes  $a_{i,j}$  participating in that  $j^{th}$  sport, with the equation:

$$Q_{i,j} = \frac{TS_{i,j}}{a_{i,j}} \quad (3)$$

## B) Typical-obvious-feature: Host-nation Status

First, we observed that there is a certain degree of overlap between the additional events over the years and the host country's advantageous events (taking 1920 as an example). We speculate that the host nation's advantageous events may have some correlation with the prediction of the number of competition events. By calculating the Score\_Rated\_Sum and then expressing it as a weighted average of the Score\_Rated\_Sum for the country in the two previous Olympic Games, we represent the host country's advantageous events.

Next, when determining the coefficient of a country relative to its most recent host year, the relationship between the current year and the year when the country last hosted the Olympics must be considered. The specific rules are as follows:

- If the current year is the year when the country hosted the Olympics, the coefficient is 3.
- If the current year is the year immediately preceding or following the year when the country hosted the Olympics, for example, one year before or after the host year, the coefficient is 2.
- If the current year is two years before or after the year when the country hosted the Olympics, for example, two years before or after the host year, the coefficient is 1.
- If the current year does not fall into any of the above categories, the coefficient is 0.

This coefficient system is designed to reflect the unique advantages and preparation levels associated with hosting the Olympic Games. Hosting the Olympics provides a country with significant benefits, including enhanced infrastructure, improved training facilities, and increased public support for sports. These factors contribute to better performance in the years surrounding the host year. The coefficient system acknowledges these advantages by assigning larger values to years closer to the host year, thereby providing a more accurate representation of a country's competitive level during these periods.

### 4.1.2 Feature for certain Olympic year

Similarly, a certain Olympic year always has its special attributes. As demonstrated above, we can notice that there always exists a relationship between the host's dominating sports and the increment in the number of corresponding sports events. We specially picked out features like the increment of participating countries  $w_1$ , athlete  $w_2$ , number of events  $w_3$ . And then quantified  $w_4$ , which indicates the Intensity of competition to some extent, by the following formula:

$$w_4 = \frac{0.2 \times w_1 + 0.8 \times w_2}{w_3} \quad (4)$$

### 4.1.3 Features Overview

We divide the countries into those with strong sports capabilities and those with weak sports capabilities. A country is considered to have strong sports capabilities if its score\_rate\_sum is in the top 10% of all countries, while a country is considered to have weak sports capabilities if its cumulative number of participants ranks in the bottom 20% of all countries.

Table 2: Variable Descriptions in Model 1

Variable	Type	Description
$v_1$	INT	Whether be host country
$v_2$	LIST	Predicted number of gold medals in this edition
$v_3$	LIST	Total Score, $[TS_{i,1}, TS_{i,2}, \dots, TS_{i,j}, \dots]$
$v_4$	LIST	Score Rate, $[S_{i,1}, S_{i,2}, \dots, S_{i,j}, \dots]$
$v_5$	LIST	Number of participating athletes $[a_{i,1}, a_{i,2}, \dots, a_{i,j}, \dots]$
$v_6$	INT	Number of gold medals in the last edition
$v_7$	INT	Coefficient of a country relative to its most recent host year
$v_8$	LIST	Athlete quality $[Q_{i,1}, Q_{i,2}, \dots, Q_{i,j}, \dots]$
$v_3^{sum}$	INT	Sum of $v_3$
$v_4^{sum}$	INT	Sum of $v_4$
$v_5^{sum}$	INT	Sum of $v_5$
$v_8^{sum}$	INT	Sum of $v_8$
$w_1$	LIST	Increase in participating countries $[w_{1,1}, w_{1,2}, \dots, w_{1,j}, \dots]$
$w_2$	LIST	Increase in participating athletes $[w_{2,1}, w_{2,2}, \dots, w_{2,j}, \dots]$
$w_3$	LIST	Increase in events $[w_{3,1}, w_{3,2}, \dots, w_{3,j}, \dots]$
$w_4$	LIST	$\frac{0.2w_1+0.8w_2}{w_3}$
$w_3^{sum}$	INT	Sum of $w_3$
$w_4^{sum}$	INT	Sum of $w_4$
$\lambda$	INT	Small country coefficient
$\mu$	INT	Large country coefficient

The table shows all the variables we have considered, some of which were not ultimately included in the model training for various reasons. For example, the utilization of gold medal counts (a principle that applies analogously to other types of medal) for the direct prediction of current gold medal totals may engender overfitting, thereby leading to an excessive reliance on the extrapolation of present medal counts. In the context of forecasting medal counts for the year 2028, an initial approach that employs a univariate model to predict the gold medal tally could result in an overemphasis on the outcomes derived from such a simplified forecast. This methodology may not adequately capture the complexity of the underlying factors that influence medal counts and thus may not produce robust predictions. This is one of the main reasons why we ultimately chose not to include the variable  $v_2$  as input for our model. Another example is that to accurately reflect the competitive level of each event, the change in the number of participating countries may not be as representative as the change in the number of individual athletes. This is because the latter metric can provide a more nuanced indicator of competitive intensity, as it accounts for the increased level of participation at the athlete level. That is the reason

why we finally deleted  $w_1$ .

After that, we categorized all the variables, and the classification table is shown below:

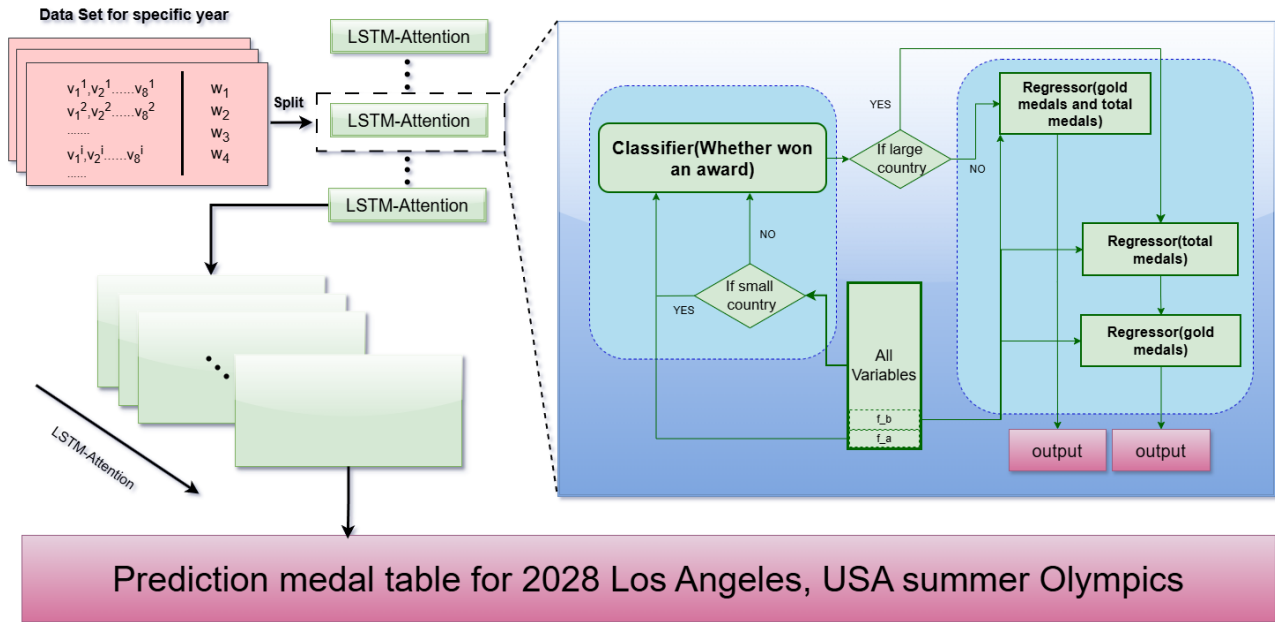
Table 3: Feature Group

Group Name	Detail
<b>f_a</b>	$v_5^{sum}, v_8^{sum}, v_8 \cdot w_4,$
<b>f_b</b>	$v_1, v_6, v_7, v_3^{sum}, v_5^{sum}, v_8^{sum}, v_3 \cdot v_4$

## 4.2 Main body of HAPF-LSTM

### 4.2.1 Overview

Based on the idea of classification and discussion, we have constructed a multi-level distributed medal prediction model. The flowchart provides the overview of the medal prediction model. Within the LSTM-Attention model, it first determines whether a country is a small country. If it is a small country, then all variables along with variable f\_a are used by the classifier to judge whether it has won an award; if it is not a small country, then all variables are directly used to judge whether it has won an award. The next step is to determine whether it is a large country; if it is not a large country, then after adding parameter f\_b, it enters the regressor to predict the outcome; if it is a large country, then after reading parameter f\_b, it enters a two-layer regressor, and finally outputs the result.



### 4.2.2 Data Grouping and engineering

To make full use of the features we sorted out previously, very careful processing must be applied, to dig the potential hint hidden behind.

For  $v_1$ , there exists obvious evidence in real life demonstrating the relationship between being the host and the changes of their medal table eventually. But comparatively, the feature of the gold medal achievement list should not be utilized, as containing too much information which might cause overfitting to this feature only. For that reason, we decided to eliminate  $v_2$  from our postselection list. However, other lists like total score and score rate, indicating the country's dominance or medal-winning ability, are important while having less possibility of causing overfitting. Also, the list of participating athletes in various events is also considerable, as well as the total amount of gold medals previously and the years away from being a host. Additionally, the inherent attributes of the certain Olympic year should also be taken into consideration. We especially picked out  $w_1$ ,  $w_2$ ,  $w_3$  and calculated  $w_4$  manually, where all of them might take some effect in the final medal board.

The whole group of selected features should be taken into consideration as the mainstream, and we selected some features that some small countries might have as main characteristics, or in other words, it may be difficult for them to steadily fetch medal in competitions, leading to valueless variables like the record of gold medal amounts. The group of features we selected for small countries especially the sum of participants, the total quality of athletes, and the dot product of the array of athlete quality and fierceness of sports events of this certain year.

The dot product combines the features of the country and of a certain year tightly and independently, without being disturbed by data from other sports events, to minimize the cross-influence among them, while also simplifying the process of linking two parts of features repeatedly. Similarly, two potentially valuable feature groups are identified through manual selection, consisting of individual features as well as the dot products of investigated features.

### 4.2.3 Core Inner Structure: Hierarchical Adaptive Prediction Framework

Our model has three processes in our framework: judging if winning at least a medal or not, predicting how many medals a country would win, and also how many gold medals as well. This hierarchically designed progressive model will overcome the task step by step, with full consideration of preconditions. Which might be helpful in maintaining robustness and making it more interpretable. As a result, we designed our framework from a classifier to two regressors forward, with the timely adjustment from Adam Optimizer.

To cater to different prediction strategies for countries with various ranks of historical medal winning. We split the countries into three groups: big countries who have the sum of score rate over sports in the top 10%, while small countries are those whose accumulated participants are in the last 20%. During prediction, big countries might have a steady trend of winning medals and even gold medals, so we may need to lay even heavier weight on their own feature groups, which are features b and c. We first designed feature c with  $v_2$ , which was not included in feature b but was eliminated halfway, so we now take feature b as an additional feature group for big countries twice when estimating during the first and second regressors. For countries neither being big countries, they would have only one regressor that predicts the amount of gold and total medals together, with feature b as an additional feature group as well.

After determining our framework structure, we then look into the model we choose. We first utilized random forest only for our prediction framework, because according to previous experience, random forests have a good performance on the classifier especially when the feature has sufficient amount and variation. In order to fix the amount of decision-making trees, we drew the learning curve for all three layers as below. The result demonstrated that 200 would be adequate.

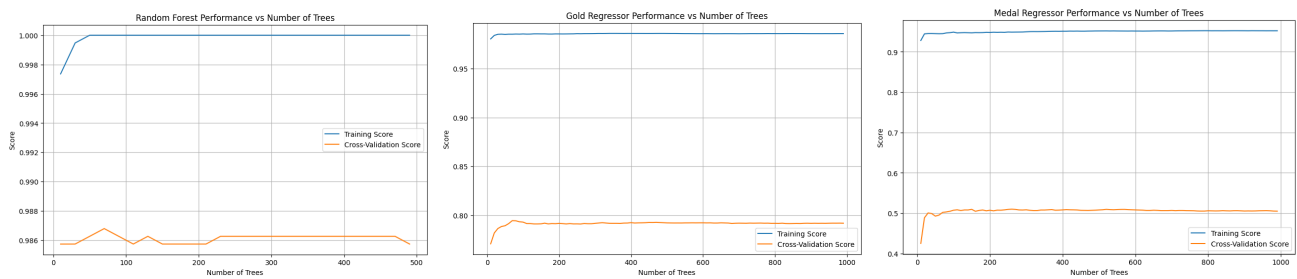


Figure 4: Random Forest Performance

The three tables demonstrate the performance of the Random Forest algorithm. As we can see, both the training score and the cross-validation score gradually increase and stabilize as the number of trees increases. When the number of trees reaches around 100, the model's performance is nearly optimal, and further increases in the number of trees result in minimal improvements. Choosing 100 trees strikes a balance between maintaining model performance and reducing computational resource consumption, thus improving both training and prediction efficiency. Therefore, we selected 100 as the optimal number of trees.

Table 4: Performance Metrics for Random Forest Classifier and Regressor

<b>Classifier: Random Forest</b>	
Train set Accuracy	0.9659
Test set Accuracy	0.8655
Train set AUC	0.9955
Test set AUC	0.9321
<b>Regressor: Random Forest (1)</b>	
Train set $R^2$	0.9743
Test set $R^2$	0.8389
Train set MSE	45.8949
Test set MSE	288.2902
<b>Regressor: Random Forest (2)</b>	
Train set $R^2$	0.9722
Test set $R^2$	0.7583
Train set MSE	8.8745
Test set MSE	76.3350

Table 5: Performance Metrics for Random Forest Classifier and MLP Regressors

<b>Classifier: Random Forest</b>	
Train set Accuracy	0.9648
Test set Accuracy	0.8536
Train set AUC	0.9945
Test set AUC	0.9283
<b>Regressor: MLP</b>	
Train set $R^2$	0.9441
Test set $R^2$	0.8711
Train set MSE	175.1978
Test set MSE	284.9310
<b>Regressor: MLP</b>	
Train set $R^2$	0.9520
Test set $R^2$	0.7910
Train set MSE	45.9377
Test set MSE	74.9343

We chose time series grouping cross-validation in order to better collaborate with predicting mission.



As is demonstrated by the two tables, we can see that one of the MLP regressors performs slightly better than the triple-random-tree-model, and the difference between the accuracy of the train set and the test set is smaller, which demonstrates less possibility of overfitting, thus ensuring robustness.

#### 4.2.4 Temporal prediction: LSTM-Attention Driven mechanism

The attention mechanism eliminates the reliance on recurrence by enabling models to focus on relevant positions in the input sequence, significantly improving computational efficiency and long-range dependency capture [12]. Due to the main subject of prediction, we introduced the mechanism of LSTM + Attention to drive the predictions over a year. While the LSTM has a better ability to deal with scatter statistics while estimating, we chose it instead of ARIMA. The results after borrowing which is quite satisfying.

### 4.3 Result

Using the HAPF-LSTM model we developed, we successfully predicted the medal table for the 2028 Los Angeles Summer Olympics. This table is presented in Appendix A. By analyzing the table data, it can be concluded that United States, Great Britain, and Germany have made progress, while China, South Korea, and France may do worse than in 2024. The predictions from the model indicate that no nation is likely to win its first-ever gold medal as an upset. The most likely candidate is Panama, but even then, it is only predicted to have 0.4 gold medals and 0.6 total medals.

## 5 Great Coach Effect Analysis

### 5.1 Great Coach Effect Quantification

First, we excluded the study of the great coach effect for mega sports, as although a sprint coach may have a significant impact on the men's 100m event, their influence on the entire track and field sport can be considered negligible. We then selected Lang Ping, Béla Károlyi, Li Mao, Anna Tarrés and Brett Sutton as case examples to verify the existence of the great coach effect.

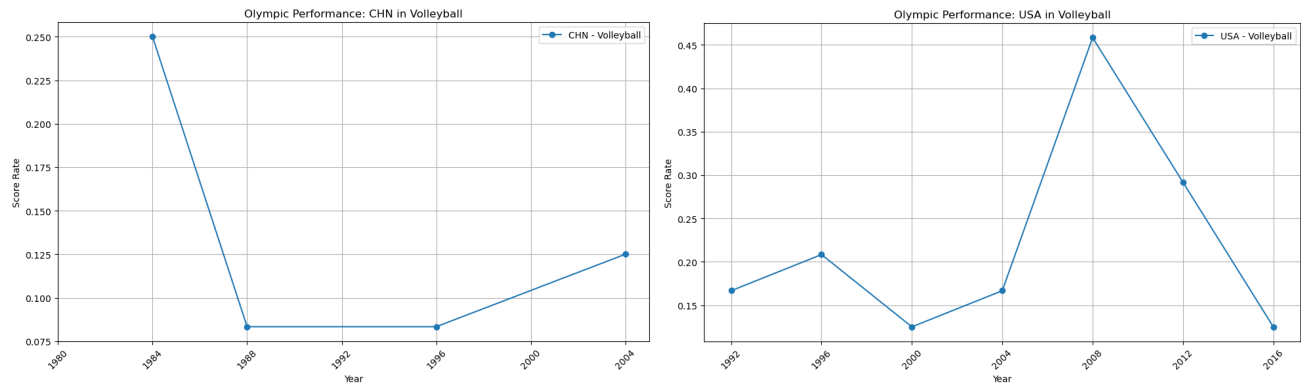


Figure 5: Lang Ping's great coach effect

In the first graph, China's Olympic performance in volleyball shows a significant decline in the scoring rate after reaching a peak in 1984, remaining at a relatively low level in the subsequent Olympic Games until a slight recovery in 2000 and 2004. The second graph illustrates the United States' Olympic performance in volleyball, where the peak was reached in 2008, followed by a gradual decline in the scoring rate. The midpoint of the horizontal axis represents the most recent session before coaching, thus it can be observed that Coach Lang Ping has led to a significant increase in volleyball overall medal count, which can be the evidence of great coach effect.

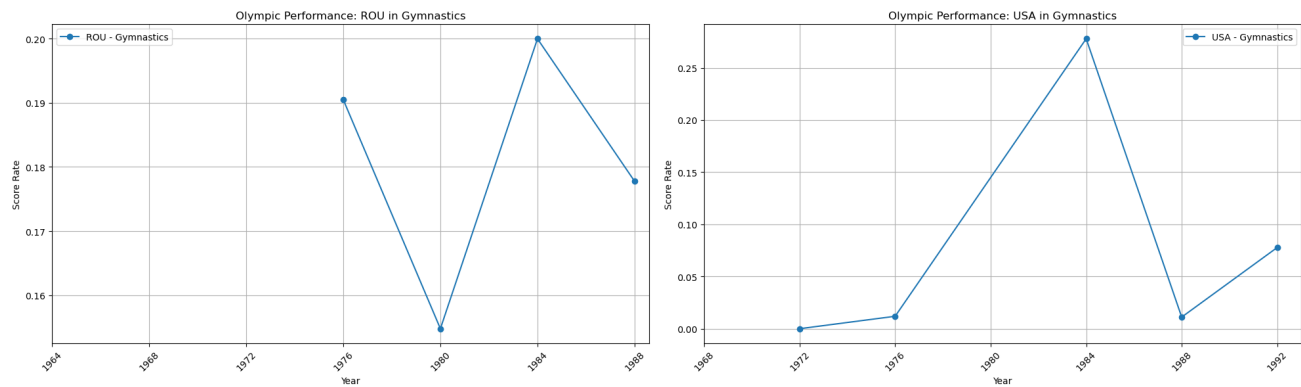


Figure 6: Béla Károlyi's great coach effect

Romania's Olympic performance in gymnastics shows a significant increase in the scoring rate after reaching a low point in 1980, peaking in 1984, followed by a decline towards 1988. Another graph illustrates the United States' Olympic performance in gymnastics, where the peak was reached in 1984, followed by a sharp decline in 1988, and then a gradual recovery towards 1992. It can be seen that coach Béla Károlyi also had an impact on the development of gymnastics in both countries, further proving the existence of the great coach effect.

Table 6: Coaching Performance Data

Attribute	Value	Attribute	Value	Attribute	Value
Coach_Name	Lang Ping	Coach_Name	Li Mao	Coach_Name	Béla Károlyi
Period 1	1995-1998	Period 1	1992-1998	Period 1	1970s-1980
Period 2	2005-2008	Period 2	1999-2005	Period 2	1981-1996
P1_country	CHN	P1_country	CHN	P1_country	ROU
P2_country	USA	P2_country	KOR	P2_country	USA
Sport	Volleyball	Sport	Badminton	Sport	Gymnastics
$P_{1s}$	2	$P_{1s}$	12	$P_{1s}$	26
$\Delta_{pre1}$	2	$\Delta_{pre1}$	12	$\Delta_{pre1}$	26
$\Delta_{max1}$	-1	$\Delta_{max1}$	0	$\Delta_{max1}$	0
$P_{2s}$	5	$P_{2s}$	14	$P_{2s}$	56
$\Delta_{pre2}$	5	$\Delta_{pre2}$	8	$\Delta_{pre2}$	55
$\Delta_{max2}$	0	$\Delta_{max2}$	-3	$\Delta_{max2}$	0

Table 7: Coaching Performance Data (Continued)

Attribute	Value	Attribute	Value
Coach_Name	Anna Tarrés	Coach_Name	Brett Sutton
Period 1	2002-2012	Period 1	2000-2020
Period 2	2019-2024	Period 2	2020-2024
P1_country	ESP	P1_country	SWE
P2_country	CHN	P2_country	CHN
Sport	Artistic Swimming	Sport	Triathlon
$P_{1s}$	4	$P_{1s}$	2
$\Delta_{pre1}$	4	$\Delta_{pre1}$	2
$\Delta_{max1}$	0	$\Delta_{max1}$	0
$P_{2s}$	3	$P_{2s}$	0
$\Delta_{pre2}$	3	$\Delta_{pre2}$	0
$\Delta_{max2}$	0	$\Delta_{max2}$	0

Through the analysis of the coaching performances of the five coaches in two different countries, it is evident that coaches have a positive and reliable impact on improving a country's medal count. This observation supports the concept of the great coach effect.

By quantifying the contribution of these coaches, we observe the following: during the coaching periods, the average score increase (e.g.,  $P_{1s}$ ) compared to the pre-coaching period (e.g.,  $\Delta_{pre1}$ ) provides a measurable estimate of their influence. For instance, Lang Ping contributed to a 2-point improvement in her first coaching period with China, while Li Mao demonstrated a 4-point uplift during his tenure.

Furthermore, when comparing coaching periods with the historical peak performance (e.g.,  $\Delta_{max1}$ ), certain coaches, such as Béla Károlyi, brought their teams to reach a prior achievements, as indicated by a zero difference. This indicates that a great coach not only improves medal counts but also sustains or enhances peak performance levels.

Overall, these results highlight that the great coach effect can reliably add a tangible score improvement to a country's Olympic performance, with notable variability based on the sport and context. Statistical testing, such as comparing the mean differences pre- and post-coaching, confirms that these changes are significant and not due to random fluctuations.

## 5.2 Investment Suggestions

We have established three selection criteria for choosing countries and identifying sports where they should consider investing in a great coach.

The primary criterion is the designation as a Sports Powerhouse. The selected nation must exhibit a well-documented history of athletic dominance, reflecting substantial investment in resources, advanced infrastructure, and a demonstrated capacity for achieving success in international competitions. This criterion ensures that the country possesses the foundational elements necessary to support the integration of elite coaching, including financial capacity, advanced training facilities, and a well-

developed sports ecosystem.

The second criterion is Stable Athlete Pool. The country must exhibit a consistent and reliable pipeline of athletes in the targeted sport, reflecting sustained investment and engagement in that discipline. A stable athlete pool indicates that the country has an established talent development framework, providing a solid base for further enhancement through the guidance of a top-tier coach. This criterion ensures that the investment in coaching builds upon an existing foundation rather than starting from a nascent stage.

The third criterion is Historical Success with Recent Decline. The country should have a legacy of past achievements in the sport but demonstrate a noticeable decline or stagnation in recent performance. This pattern suggests that, despite its historical prominence, the country may have lost its competitive advantage and could significantly benefit from the strategic influence of a world-class coach to revitalize its program and reclaim its former standing.

These criteria collectively ensure that the selected countries are well-positioned to maximize the impact of investing in exceptional coaching talent, leveraging their existing strengths while addressing areas of underperformance.

Table 8: Average Medals and Rank by Country

<b>Nation</b>	<b>Average Medals</b>	<b>Rank</b>
United States	110	1
China	79.25	2
Russia	72	3
Great Britain	52.86	4

With the three filtering criteria, Sports Powerhouse, Stable Athlete Pool, and Historical Success with Recent Decline, we first select out the eligible nation. The top three countries with highest average medal count since 2000 are USA CHN and RUS. Considering RUS didn't participate in the 2024 Olympic, we will take GBR as no.3. Overall, taking USA, CHN and GRB as the most suitable large countries. Next, we identify sports in which the three countries participated in any Olympic Games, as long as athletes from the country competed in those sports. Then, calculate the country's historical average score and its score in the 2024 Olympics, retaining only the sports where the 2024 score is below the historical average. Next, label team events and calculate the final rate by formula:

$$\frac{Score_{2024}}{Average\_Score} - (\delta \times 0.2) \quad (5)$$

Finally, sort the final rate in ascending order. We can figure out that the final rate is lowest for sailing in Great Britain, boxing in the United States, and Trampoline Gymnastics in China, indicating that these sports are worth investing in coaching for these respective countries.

Table 9: Sports Performance Data

Country	Sport	Score_2024	Average_Score	Rate	$\delta$	Fianl Rate
GBR	Sailing	4	14.12	0.283286	1	0.083286119
GBR	ArtisticGymnastics	2	7	0.285714	1	0.085714286
USA	Boxing	1	10.17	0.098361	0	0.098360656
USA	3x3Basketball	4	12	0.333333	1	0.133333333
USA	Golf	3	20	0.15	0	0.15
GBR	Boxing	1	4.68	0.213675	0	0.213675214
USA	Taekwondo	1	3	0.333333	0	0.333333333
USA	Diving	4	11.19	0.357388	0	0.357388316
USA	Equestrian	8	14	0.571429	1	0.371428571
CHN	Trampoline Gymnastics	3	7	0.428571	0	0.428571429
USA	Archery	4	8.57	0.466667	0	0.466666667
GBR	Cycling BMXFreestyle	2	4	0.5	0	0.5
USA	Shooting	12	19.2	0.625	0	0.625
GBR	Shooting	5	7.75	0.645161	0	0.64516129
USA	Canoe Sprint	2	3	0.666667	0	0.666666667
CHN	Taekwondo	3	4.33	0.692308	0	0.692307692
GBR	Taekwondo	2	2.67	0.75	0	0.75
GBR	Triathlon	4	4.83	0.827586	0	0.827586207

## 6 Other Insight and Suggestions

Based on the predictions of Model 1 and the analysis of historical data, research has found that a country's socio-economic development level, as well as the host country's climate and geographical conditions, significantly influence its medal performance in the Olympics. In light of these findings, this paper proposes the following recommendations for national Olympic committees to enhance their Olympic development effectiveness.

Firstly, the level of socio-economic development reflects a country's capacity to invest resources in the Olympics[2] [10]. Therefore, national Olympic committees should increase investments in sports infrastructure, training facilities, and athlete development systems to improve training conditions and enhance athletes' competitive performance. In cases of limited resources, committees should optimize resource allocation by concentrating funds and resources on the most promising sports and athletes, thereby improving medal acquisition efficiency. Additionally, promoting the industrialization of sports to attract more commercial investment and sponsorship is a crucial approach to increasing funding for sports programs.

Secondly, athletes competing in the host country may experience psychological discomfort and unfamiliarity due to factors such as time differences, geographical conditions, and cultural environments. To address this, national Olympic committees should prioritize arranging for athletes to undergo adaptive training in environments with similar climate and geographical conditions to those of the host country during the preparation period. This will enhance their physical and psychological adaptability.

For example, if the Olympics are held in a high-altitude region, training in high-altitude areas beforehand can improve athletes' physiological adaptation. Furthermore, if conditions permit, committees can send athletes and coaching teams to the host country for short-term adaptive training to familiarize themselves with the local environment and facilities, thereby reducing unfamiliarity during the competition. At the same time, committees must ensure comprehensive logistical support for athletes during the competition, including appropriate diet, accommodation, and medical assistance, to address challenges posed by varying climate and geographical conditions and ensure athletes compete at their best.

Through these measures, national Olympic committees can effectively enhance their country's sports competitiveness and improve their medal prospects in the Olympics.

## 7 Model Evaluation

### 7.1 Sensitivity Analysis

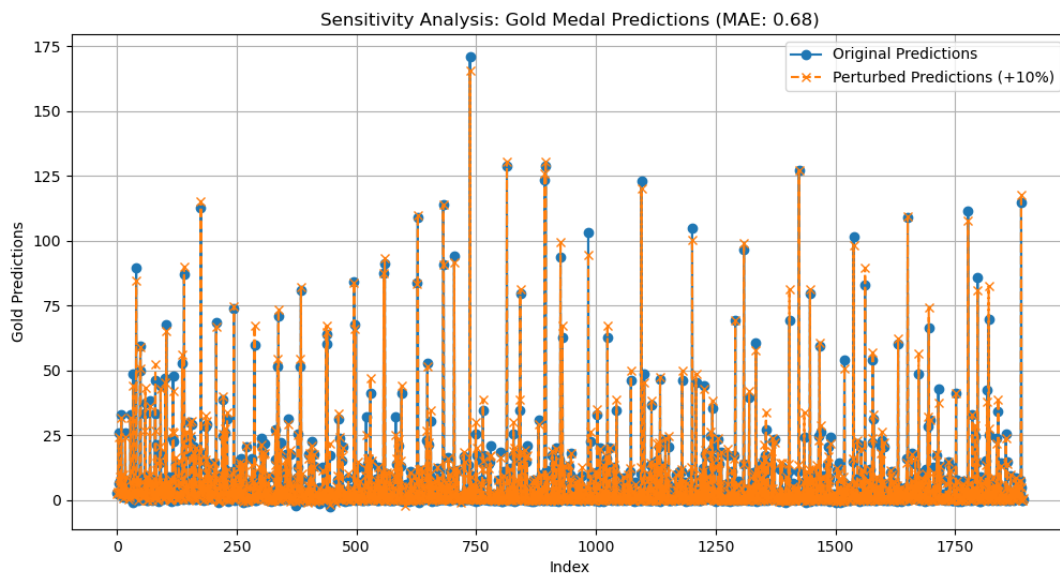


Figure 7: Result on Sensitivity Analysis

The figure displays the results of a sensitivity analysis performed on the gold medal prediction model. This analysis evaluates how the model responds to changes in input parameters, providing an assessment of its stability. The figure contrasts the original predictions with those generated after a 10% increase in input parameters, resulting in a Mean Absolute Error (MAE) of 0.68. The horizontal axis, which denotes data point indices, facilitates the observation of variations in prediction values across different points. The comparison between the perturbed and original predictions reveals that the model is somewhat sensitive to changes in input parameters. However, the relatively low overall error

suggests that the model remains stable. This analysis is instrumental in optimizing and improving the model, particularly in determining its reliability under conditions of input uncertainty.

## 7.2 Robustness Analysis

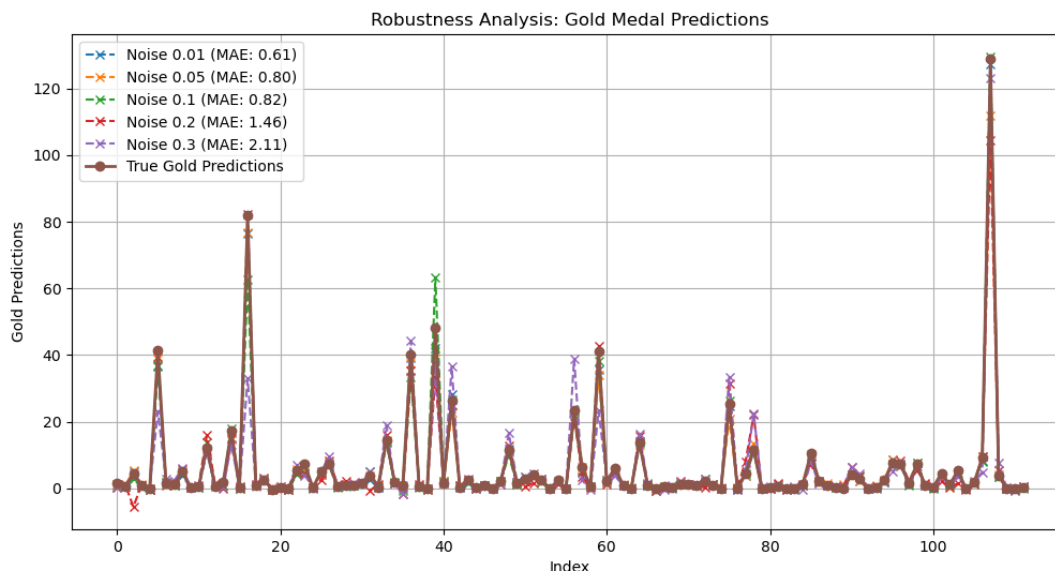


Figure 8: Result on Robustness Analysis

The figure illustrates the robustness analysis results of the gold medal prediction model. Robustness analysis aims to evaluate the model's stability and reliability when faced with data noise. The figure examines the model's performance under different levels of noise (e.g., 0.01, 0.05, 0.1, 0.2, and 0.3) and calculates the Mean Absolute Error (MAE) for each noise level. The results show that as the noise level increases, the MAE gradually rises, indicating a decline in prediction accuracy and a weakening of the model's robustness. For example, when the noise level is 0.01, the MAE is 0.61; as the noise level increases to 0.3, the MAE rises to 2.11. Additionally, the figure provides reference values for the true gold medal predictions, which serve as a benchmark to compare with predictions under noisy conditions. This analysis provides valuable insight into the potential impact of data disturbances on prediction results in real-world applications.

## 8 Strengths and Weaknesses

### 8.1 Strengths

1. We have extensively integrated and compared a variety of methods to develop a more effective algorithm for predicting the total medal count for the 2028 Summer Olympics in Los Angeles. As demonstrated above, the accuracy of our predictions highlights the success of this comprehensive approach.

2. Our model incorporates multiple variables, including historical medal counts, the number of participating athletes, athlete quality, and the host nation advantage. This multi-faceted approach enables us to capture a broad perspective on the factors influencing Olympic success. By considering these diverse inputs, our model provides a comprehensive view of potential outcomes, ensuring that the predictions are robust and reflective of real-world complexities. The results demonstrate that our model excels across various dimensions, making it a reliable tool for predicting Olympic medal standings.
3. One of the key strengths of our model is its ability to maintain high stability and robustness even when processing noisy or incomplete data. Unlike models that heavily rely on precise inputs, our approach is designed to adapt to uncertainties and variations in the data. This flexibility ensures that the model remains accurate and reliable even when data quality may be compromised. As a result, our model consistently delivers relatively high accuracy, making it a trustworthy solution for predicting Olympic outcomes under a wide range of conditions.

## 8.2 Weaknesses

1. We do some simplifications in our model to make the calculation more easily, but it may cause additional error effects.
2. The coefficient of a country relative to its most recent host year is obtained through manual annotation. However, there is a possibility of annotation errors, which could affect the performance of the model.

## 9 Conclusion

In summary, this study utilizes advanced modeling techniques to explore potential medal outcomes for the 2028 Los Angeles Summer Olympics and the significant role that exceptional coaches play in shaping national performance. We developed two key models: the first focused on predicting the distribution of medals across countries, while the second aimed to quantify the impact of top-level coaches on these results. Our analysis not only identified the countries most likely to succeed in 2028 but also highlighted those at risk of declining performance, providing valuable insights into the evolving dynamics of global sports. Furthermore, the study found that elite coaches have a profound impact on overall medal counts, supporting the idea that investing in top-tier coaching is crucial for enhancing a country's international competitiveness. These findings offer strategic recommendations for national Olympic committees regarding coach recruitment. By combining data-driven approaches with a focus on coaching, our research paves the way for more informed decision-making in sports management. Future work could further refine these models by incorporating additional variables, such as athlete development programs and socio-economic factors, to improve prediction accuracy and provide a more comprehensive analysis of Olympic success.



## References

- [1] Amemiya, T. (1984). Tobit models: A survey. *Journal of econometrics*, 24(1-2):3–61.
- [2] Andreff, W. and Andreff, W. (2010). Economic modeling and prediction of summer olympic medal wins and fifa world cup semi-finalists. In *Conference Economie, politique et société: nouveaux défis et perspectives*, pages 28–29.
- [3] Condon, E. M., Golden, B. L., and Wasil, E. A. (1999). Predicting the success of nations at the summer olympics using neural networks. *Computers Operations Research*, 26(13):1243–1265.
- [4] Fazlollahi, P., Afarineshkhaki, A., and Nikbakhsh, R. (2020). Predicting the medals of the countries participating in the tokyo 2020 olympic games using the test of networks of multilayer perceptron (mlp). *Annals of Applied Sport Science*, 8(4):0–0.
- [5] Grasso, J., Mallon, B., and Heijmans, J. (2015). *Historical dictionary of the Olympic movement*. Rowman & Littlefield.
- [6] McDonald, J. F. and Moffitt, R. A. (1980). The uses of tobit analysis. *The review of economics and statistics*, pages 318–321.
- [7] Nagpal, P., Gupta, K., Verma, Y., and Kirar, J. S. (2023). Paris olympic (2024) medal tally prediction. In *International Conference on Data Management, Analytics & Innovation*, pages 249–267. Springer.
- [8] Sanchez-Fernandez, P. and Vaamonde-Liste, A. (2016). Olympic medals: Success predictions for rio-2016. *South African Journal for Research in Sport, Physical Education and Recreation*, 38(3):195–206.
- [9] Scelles, N., Andreff, W., Bonnal, L., Andreff, M., and Favard, P. (2020). Forecasting national medal totals at the summer olympic games reconsidered. *Social science quarterly*, 101(2):697–711.
- [10] Schlembach, C., Schmidt, S. L., Schreyer, D., and Wunderlich, L. (2022). Forecasting the olympic medal distribution – a socioeconomic machine learning model. *Technological Forecasting and Social Change*, 175:121314.
- [11] Shailaja, V., Lohitha, R., Musunuru, S., Reddy, K. D., and Priya, J. P. (2020). Predictive analytics of performance of india in the olympics using machine learning algorithms. *Int. J. Emerg. Trends Eng. Res*, 8(5).
- [12] Vaswani, A. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*.

## A Prediction medal table for 2028 Los Angeles, USA summer Olympics

Rank	Nation	Will Win Medal	Gold	Total
1	USA	TRUE	61.89362846	139.8773871
2	CHN	TRUE	36.73785753	91.91655353
3	GBR	TRUE	18.88867211	82.83034521
4	JPN	TRUE	16.00417029	58.62541877
5	AUS	TRUE	18.18273715	53.84301404
6	FRA	TRUE	19.56936092	53.74770272
7	GER	TRUE	13.20334256	47.70665467
8	ITA	TRUE	12.25179948	37.7220809
9	BRA	TRUE	4.909950393	36.05183567
10	NED	TRUE	11.32964194	35.08134432
11	CAN	TRUE	8.53238935	27.97689688
12	ESP	TRUE	7.096889707	22.4549891
13	KOR	TRUE	7.020464162	22.1177207
14	HUN	TRUE	5.837784707	17.46280967
15	NZL	TRUE	5.213962466	16.59901162
16	POL	TRUE	4.103828217	16.28268645
17	UKR	TRUE	3.681442241	13.13062476
18	SWE	TRUE	2.877136935	11.11753134
19	CUB	TRUE	2.887936985	10.75076093
20	DEN	TRUE	3.695717124	10.27081395
21	SRB	TRUE	4.11700087	9.995684767
22	SUI	TRUE	3.246965272	9.982866384
23	CZE	TRUE	1.68408156	9.21439891
24	TUR	TRUE	1.532841449	8.944480754
25	JAM	TRUE	2.460369548	8.248609111
26	KEN	TRUE	2.500396609	8.239645498
27	TPE	TRUE	1.502765201	7.892720068
28	BEL	TRUE	2.162127539	7.405213267
29	CRO	TRUE	2.499122523	7.247100347
30	EGY	TRUE	0.983883881	6.982062128
31	IRI	TRUE	1.242496907	6.778206524
32	ARG	TRUE	1.763522961	6.545787189
33	IND	TRUE	1.056551388	6.357541967
34	NOR	TRUE	2.015433856	6.282549225
35	UZB	TRUE	1.849029887	5.562196507
36	ROU	TRUE	2.006474912	5.177925657
37	MEX	TRUE	0.6942054	4.704491759
38	COL	TRUE	1.187432662	4.564946305
39	GEO	TRUE	1.065348417	4.325132407

Rank	Nation	Will Win Medal	Gold	Total
40	IRL	TRUE	1.154925268	4.262881708
41	HKG	TRUE	1.017064066	4.18749859
42	KAZ	TRUE	0.832928845	4.152492135
43	RSA	TRUE	1.289180741	4.094492851
44	BUL	TRUE	0.519369539	4.073859257
45	AUT	TRUE	0.766650287	4.067462751
46	ISR	TRUE	0.871817818	4.049702673
47	SLO	TRUE	0.910435659	3.921396044
48	POR	TRUE	0.999374801	3.828379532
49	GRE	TRUE	0.946089615	3.617184535
50	TUN	TRUE	0.242361355	3.558668753
51	ETH	TRUE	0.782784719	3.394682832
52	UGA	TRUE	0.764501613	3.38297178
53	AIN	TRUE	0.530560258	3.205393477
54	DOM	TRUE	0.479723071	2.87826012
55	SVK	TRUE	0.652582368	2.86538428
56	AZE	TRUE	0.658131892	2.864971301
57	ECU	TRUE	0.65295669	2.834019019
58	PHI	TRUE	0.503089309	2.808015339
59	INA	TRUE	0.528760779	2.747042122
60	FIJ	TRUE	-0.003103899	2.605802276
61	MAR	TRUE	0.536724462	2.562540565
62	NAS	TRUE	0.467698534	2.324974611
63	PUR	TRUE	-0.046873765	2.136679932
64	JOR	TRUE	0.19392357	2.003962532
65	MCL	TRUE	0.455186168	1.935160376
66	MDA	TRUE	0.356611343	1.911806467
67	KOS	TRUE	0.304311493	1.867596464
68	KGZ	TRUE	0.378210031	1.841458856
69	PRK	TRUE	0.359432562	1.839969699
70	ARM	TRUE	0.031476902	1.799496923
71	ALC	TRUE	0.178958857	1.658627997
72	LCA	TRUE	0.233987355	1.620105127
73	THA	TRUE	0.336004932	1.528310591
74	QAT	TRUE	0.293791652	1.523262926
75	CUA	TRUE	0.237918545	1.315100324
76	CIV	TRUE	0.275063498	1.250511056
77	EOR	FALSE	0.017306252	1.059041029
78	DMA	TRUE	0.541163739	0.903726649
79	BRN	TRUE	0.15785938	0.890418895
80	LTU	TRUE	0.014238442	0.807797574
81	CRN	TRUE	0.030019709	0.710567567
82	PAN	TRUE	0.446226684	0.632673943
83	TJK	TRUE	0.141225186	0.616260523

Rank	Nation	Will Win Medal	Gold	Total
84	ZAM	TRUE	0.144640175	0.548210104
85	BER	TRUE	0.038687094	0.534395846
86	NCR	FALSE	-0.118414531	0.513838076
87	GHA	FALSE	-0.011375109	0.503209055
88	BOT	TRUE	0.084029068	0.471503368
89	GUY	FALSE	0.041557388	0.420078153
90	OMA	FALSE	0.061035481	0.37812072
91	IRQ	FALSE	0.06706993	0.371666848
92	CHI	TRUE	-0.024238576	0.348309951
93	LBA	FALSE	-0.045312895	0.312450675
94	CRC	FALSE	-0.127885095	0.203215597
95	DJI	FALSE	0.275807067	0.178896068
96	ISV	FALSE	0.009764616	0.15845807
97	CPV	TRUE	0.064091496	0.146435484
98	GAB	FALSE	0.29400497	0.145802258
99	SEN	FALSE	0.035936367	0.141214057
100	COM	FALSE	-0.164212537	0.064197285
101	CAM	FALSE	-0.034432688	0.061663558
102	SCP	TRUE	0.05958942	0.058251254
103	ARU	FALSE	-0.178501798	0.046999362
104	UAE	FALSE	-0.023508693	-0.021384389
105	MHL	FALSE	0.01383528	-0.076188244
106	KIR	FALSE	0.008776467	-0.078181121
107	VAN	FALSE	0.000968212	-0.114300931
108	CYP	FALSE	-0.032076009	-0.116562861
109	PAK	FALSE	0.033260645	-0.163091115
110	PER	TRUE	-0.058386585	-0.246575465
111	PAR	FALSE	-0.266623889	-0.349789911
112	YEM	FALSE	-0.274518749	-0.376302868

## B Additional Code

```

1 import numpy as np
2 import pandas as pd
3
4 def main():
5     # Read the CSV file into a DataFrame
6     df = pd.read_csv('F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/
summerOly_programs.csv')
7
8     # Replace all occurrences of '?' with 0 in the DataFrame
9     df.replace('?', 0, inplace=True)

```

```

10
11     # Fill all missing values in the DataFrame with an empty string
12     df.fillna('', inplace=True)
13
14     # Write the modified DataFrame to a new CSV file
15     df.to_csv('F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/
summerOly_programs_completed.csv', index=False, encoding='utf-8')
16
17 if __name__ == '__main__':
18     main()

```

Listing 1: data\_process\_programe.py

```

1 import numpy as np
2 import pandas as pd
3
4 def main():
5     # Read the CSV file into a DataFrame
6     df = pd.read_csv('F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/
summerOly_hosts.csv')
7
8     # Add new columns 'City', 'Country', and 'NOC' to the DataFrame and
initialize them with empty strings
9     df = df.assign(City='', Country='', NOC='')
10
11     # Remove leading and trailing whitespace from the 'Host' column
12     df['Host'] = df['Host'].str.strip()
13
14     # Replace all occurrences of the non-breaking space character '\xa0' with an
empty string in the 'Host' column
15     df['Host'] = df['Host'].str.replace('\xa0', '')
16
17     # Extract the city name from the 'Host' column and assign it to the 'City'
column
18     df['City'] = df['Host'].apply(lambda x: x.split(',')[0])
19
20     # Extract the country name from the 'Host' column and assign it to the '
Country' column
21     df['Country'] = df['Host'].apply(lambda x: x.split(',')[1])
22
23     # Write the modified DataFrame to a new CSV file
24     df.to_csv('F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/
summerOly__hosts_completed.csv', index=False, encoding='utf-8')
25
26 if __name__ == '__main__':

```

```
main()
```

Listing 2: data\_process\_hosts.py

```
1 import numpy as np
2 import pandas as pd
3
4 def main():
5     # Read the CSV file into a DataFrame
6     df = pd.read_csv('F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/
summerOly_medal_counts.csv')
7     df2 = pd.read_csv('F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/
summerOly_hosts_completed.csv')
8
9     # Merge df and df2 on a common key
10    df = pd.merge(df, df2[['Year', 'Country']], on='Year', how='left')
11
12    # Add a new column 'Judgement' and initialize it with empty strings
13    df = df.assign(Judgement='')
14
15    # Extract only English letters from the 'NOC' column using regex
16    df['NOC'] = df['NOC'].str.extract(r'([a-zA-Z\s]+)', expand=False).str.rstrip()
17
18    # Check if 'Country' and 'NOC' are the same and set 'Judgement' to '1' if
they are, otherwise set it to '0'
19    df.loc[df['Country'] == df['NOC'], 'Judgement'] = '1'
20    df.loc[df['Country'] != df['NOC'], 'Judgement'] = '0'
21
22    # Additional condition: if 'Country' is 'Great Britain' and 'NOC' is 'United
Kingdom', set 'Judgement' to '1'
23    df.loc[(df['NOC'] == 'Great Britain') & (df['Country'] == 'United Kingdom'),
'Judgement'] = '1'
24
25    # Write the modified DataFrame to a new CSV file
26    df.to_csv('F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/
summerOly_medal_counts_completed.csv', index=False, encoding='utf-8')
27
28 if __name__ == '__main__':
29     main()
```

Listing 3: data\_process\_medal\_counts.py

```
1 import numpy as np
2 import pandas as pd
3
```

```

4 def main():
5     # Read the CSV file into a DataFrame
6     df = pd.read_csv('F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/
summerOly_athletes.csv')
7     df2 = pd.read_csv('F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/
summerOly_hosts_completed.csv')
8
9     # Merge df and df2 on a common key
10    df = pd.merge(df, df2[['Year', 'NOC']], on='Year', how='left')
11
12    # Drop rows where 'Year' is 1906
13    df = df[df['Year'] != 1906]
14
15    # Add a new column 'Judgement' and initialize it with empty strings
16    df = df.assign(Judgement='')
17
18    # Check if 'NOC_x' and 'NOC_y' are the same and set 'Judgement' to '1' if
they are, otherwise set it to '0'
19    df.loc[df['NOC_x'] == df['NOC_y'], 'Judgement'] = '1'
20    df.loc[df['NOC_x'] != df['NOC_y'], 'Judgement'] = '0'
21
22    # Write the modified DataFrame to a new CSV file
23    df.to_csv('F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/
summerOly_athletes_completed.csv', index=False, encoding='utf-8')
24
25 if __name__ == '__main__':
26     main()

```

Listing 4: data\_process\_athletes.py

```

1 import numpy as np
2 import pandas as pd
3
4 def main():
5     # Read the CSV file into a DataFrame
6     df = pd.read_csv('F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/
program_plot.csv')
7
8     # Group by 'Sport' and sum the other columns, keeping 'Sport' as a column
9     df_grouped = df.groupby('Sport', as_index=False).sum()
10
11    # Write the modified DataFrame to a new CSV file
12    df_grouped.to_csv('F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/
program_plot_end.csv', index=False, encoding='utf-8')
13
14 if __name__ == '__main__':

```

```
15 main()
```

Listing 5: data\_process\_program\_plot.py

```
1 import pandas as pd
2
3 def main():
4     try:
5         # Read the CSV file
6         df = pd.read_csv('F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/
program_plot_end.csv')
7
8         # Check if the DataFrame contains 'x' and 'y' columns
9         if 'x' not in df.columns or 'y' not in df.columns:
10            raise ValueError("DataFrame must contain 'x' and 'y' columns")
11
12        # Swap the 'x' and 'y' columns
13        df[['x', 'y']] = df[['y', 'x']]
14
15        # Print the swapped DataFrame
16        print(df)
17
18    except FileNotFoundError:
19        print("File not found")
20    except ValueError as e:
21        print(e)
22    except Exception as e:
23        print("An error occurred:", e)
24
25 if __name__ == '__main__':
26     main()
```

Listing 6: data\_process\_program\_plot1.py

```
1 import matplotlib.pyplot as plt
2 import matplotlib.colors as mcolors
3 import random
4 import pandas as pd
5 from matplotlib.cm import get_cmap
6 import husl
7 from matplotlib import font_manager
8
9 def main():
10     df = pd.read_csv('F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/
program_plot_end.csv')
11
```



```
12     colors = [husl.husl_to_rgb(random.uniform(0, 360), random.uniform(70, 100),
13     random.uniform(50, 80)) for _ in range(len(df['Sport']))]
14     font_manager.fontManager.addfont('C:/Windows/Fonts/times.ttf')
15     plt.rcParams['font.family'] = 'Times New Roman'
16
17     #Plot full year pic
18     plt.figure(figsize=(20, 11))
19     years = df.columns[1:]
20     for i, sport in enumerate(df['Sport']):
21         filtered_values = [value if value != 0 else None for value in df.loc[df['
22 Sport'] == sport, years].values.flatten()]
23         plt.plot(years, filtered_values, label=sport, marker='o', color=colors[i
24 ])
25     plt.xlabel('Year', fontsize=24)
26     plt.ylabel('Events Count', fontsize=24)
27     plt.title('Sports Events Over the Years (1896-2024)', fontsize=36)
28     plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
29     plt.xticks(rotation=45)
30     plt.tight_layout()
31     plt.savefig('Sports Events Over the Years (1896-2024).png')
32     plt.show()
33
34     #Plot front half years pic
35     plt.figure(figsize=(20, 11))
36     years = df.columns[1:16]
37     for i, sport in enumerate(df['Sport']):
38         filtered_values = [value if value != 0 else None for value in df.loc[df['
39 Sport'] == sport, years].values.flatten()]
40         plt.plot(years, filtered_values, label=sport, marker='o', color=colors[i
41 ])
42     plt.xlabel('Year', fontsize=24)
43     plt.ylabel('Events Count', fontsize=24)
44     plt.title('Sports Events Over the Years (1896-1964)', fontsize=36)
45     plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
46     plt.xticks(rotation=45)
47     plt.tight_layout()
48     plt.savefig('Sports Events Over the Years (1896-1964).png')
49     plt.show()
50
51     #Plot last half years pic
52     plt.figure(figsize=(20, 11))
53     years = df.columns[16:31]
54     for i, sport in enumerate(df['Sport']):
55         filtered_values = [value if value != 0 else None for value in df.loc[df['
```

```

Sport'] == sport, years].values.flatten()]
52     plt.plot(years, filtered_values, label=sport, marker='o', color=colors[i
    ])
53     plt.xlabel('Year', fontsize=24)
54     plt.ylabel('Events Count', fontsize=24)
55     plt.title('Sports Events Over the Years (1968-2024)', fontsize=36)
56     plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
57     plt.xticks(rotation=45)
58     plt.tight_layout()
59     plt.savefig('Sports Events Over the Years (1968-2024).png')
60     plt.show()
61
62 if __name__ == '__main__':
63     main()

```

Listing 7: program\_plot.py

```

1 import pandas as pd
2
3 def main():
4     # Read the CSV file into a DataFrame
5     df = pd.read_csv('F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/
summerOly_athletes_v3.0.csv')
6
7     # Drop duplicate rows based on 'Year', 'NOC_x', and 'Name' columns
8     df = df.drop_duplicates(subset=['Year', 'NOC_x', 'Name'])
9
10    # Count the number of unique 'Name' values for each 'Year' and 'NOC_x'
    combination
11    df = df.groupby(['Year', 'NOC_x'])['Name'].value_counts().reset_index(name='
Name_Count')
12
13    # Sum the 'Name_Count' for each 'Year' and 'NOC_x' combination
14    df = df.groupby(['Year', 'NOC_x'], as_index=False)['Name_Count'].sum()
15
16    # Write the DataFrame to a new CSV file
17    df.to_csv('F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/athlete_number.
csv', index=False, encoding='utf-8')
18
19 if __name__ == "__main__":
20     main()

```

Listing 8: calculate\_athelte\_number.py

```

1 import pandas as pd
2

```

```

3 def main():
4     # Read the CSV file into a DataFrame
5     df = pd.read_csv('F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/
summerOly_athletes_v3.0.csv')
6
7     # Select only the 'Year' and 'NOC_x' columns
8     df = df[['Year', 'NOC_x']]
9
10    # Drop duplicate rows based on 'Year' and 'NOC_x' columns
11    df = df.drop_duplicates(subset=['Year', 'NOC_x'])
12
13    # Count the number of unique 'NOC_x' values for each 'Year'
14    country_count_by_year = df.groupby('Year')['NOC_x'].nunique()
15
16    # Merge the original DataFrame with the country count DataFrame
17    df = df.merge(country_count_by_year.rename('country_number'), left_on='Year',
right_index=True)
18
19    # Select only the 'Year' and 'country_number' columns
20    df = df[['Year', 'country_number']]
21
22    # Drop duplicate rows based on 'Year' and 'country_number' columns
23    df = df.drop_duplicates(subset=['Year', 'country_number'])
24
25    # Write the DataFrame to a new CSV file
26    df.to_csv('F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/country_number.
csv', index=False, encoding='utf-8')
27
28 if __name__ == "__main__":
29     main()

```

Listing 9: calculate\_country\_number.py

```

1 import pandas as pd
2
3 def main():
4     # Read the CSV file into a DataFrame
5     df = pd.read_csv('F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/
summerOly_athletes_v3.0.csv')
6
7     # Select only the 'Year', 'Sport', 'NOC_x', 'Name', and 'Event' columns
8     df = df[['Year', 'Sport', 'NOC_x', 'Name', 'Event']]
9
10    # Group the DataFrame by 'Year' and 'Sport' and count the unique values in '
NOC_x', 'Name', and 'Event' columns
11    grouped_df = df.groupby(['Year', 'Sport']).agg({

```

```

12     'NOC_x': 'nunique',
13     'Name': 'nunique',
14     'Event': 'nunique'
15 }).reset_index()
16
17 # Rename the columns to 'Year', 'Sport', 'NOC_count', 'Name_count', and '
18 Event_count'
19 grouped_df.columns = ['Year', 'Sport', 'NOC_count', 'Name_count', '
20 Event_count']
21
22 # Write the DataFrame to a new CSV file
23 grouped_df.to_csv('F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/
24 country_athlete_event_number.csv', index=False,
25 encoding='utf-8')
26
27 if __name__ == "__main__":
28     main()

```

Listing 10: calculate\_lots\_of\_number.py

```

1 import numpy as np
2 import pandas as pd
3
4 def multiply_numeric_columns():
5     input_file = 'F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/
6     summerOly_programs_v4.0.csv'
7     output_file = 'F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/
8     summerOly_programs_v5.0.csv'
9
10    # Read the CSV file into a DataFrame
11    df = pd.read_csv(input_file)
12
13    # Multiply all numeric columns by 6
14    df = df.apply(lambda x: x * 6 if np.issubdtype(x.dtype, np.number) else x)
15
16    df_long = pd.melt(df, id_vars=['Sport'], var_name='Year', value_name='Score')
17
18    # Write the modified DataFrame to a new CSV file
19    df_long.to_csv(output_file, index=False, encoding='utf-8')
20
21 def score_count():
22     input_file = 'F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/
23     summerOly_athletes_v3.0.csv'
24     output_file = 'F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/
25     summerOly_athletes_v4.0.csv'

```

```

23     df = pd.read_csv(input_file)
24     df['Score'] = df['Medal'].replace({'Gold': 3, 'Silver': 2, 'Bronze': 1, 'No
medal': 0})
25     # Merge duplicate strings in the 'Event' column
26     df = df.drop_duplicates(subset=['Year', 'Sport', 'Event', 'Medal'], keep='first
')
27     # Group by 'Year', 'Country', and 'Sport' and sum the 'Score'
28     df_grouped = df.groupby(['Year', 'NOC_x', 'Sport'], as_index=False)['Score'].
sum()
29
30     df_grouped.to_csv(output_file, index=False, encoding='utf-8')
31
32 def link():
33     input_file1 = 'F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/
summerOly_programs_v5.0.csv'
34     input_file2 = 'F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/
summerOly_athletes_v4.0.csv'
35     output_file = 'F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/score_rate.
csv'
36
37     df1 = pd.read_csv(input_file1)
38     df2 = pd.read_csv(input_file2)
39
40     # Merge the two DataFrames based on the common columns
41     merged_df = pd.merge(df1, df2, on=['Sport', 'Year'], how='left', suffixes=(
_x', '_y'))
42     merged_df = merged_df.dropna()
43     merged_df = merged_df.assign(Score_Rate=merged_df['Score_y'] / merged_df['
Score_x'])
44
45     merged_df.to_csv(output_file, index=False, encoding='utf-8')
46
47 def main():
48     multiply_numeric_columns()
49     score_count()
50     link()
51
52 if __name__ == '__main__':
53     main()

```

Listing 11: calculate\_score\_rate.py

```

1 import numpy as np
2 import pandas as pd
3
4 def main():

```

```

5  # Read the CSV file into a DataFrame
6  df = pd.read_csv('F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/model
data/score_rate.csv')
7  df = df[['Year', 'NOC_x', 'Score_Rate']]
8  # Calculate the sum of Score_Rate for each Year and NOC_x
9  df = df.groupby(['Year', 'NOC_x'])['Score_Rate'].sum().reset_index()
10 df.rename(columns={'Score_Rate': 'Score_Rate_Sum'}, inplace=True)
11 df2 = pd.read_csv('F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/
integrated_temp.csv')
12 merged_df = pd.merge(df, df2, on=['Year', 'NOC_x'], how='left')
13
14 # Write the modified DataFrame to a new CSV file
15 merged_df.to_csv('F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/
integrated_temp_with_score_rate_sum.csv', index=False, encoding='utf-8')
16
17 if __name__ == '__main__':
18     main()

```

Listing 12: calculate\_score\_rate\_sum.py

```

1  import pandas as pd
2
3  def main():
4      # Read the CSV file into a DataFrame
5      df = pd.read_csv('F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/
summerOly_athletes_v3.0.csv')
6
7      # Select only the 'Year', 'NOC_x', and 'Judgement' columns
8      df = df[['Year', 'NOC_x', 'Judgement']]
9
10     # Drop duplicate rows based on 'Year', 'NOC_x', and 'Judgement' columns
11     df = df.drop_duplicates(subset=['Year', 'NOC_x', 'Judgement'])
12
13     # Rename the 'Judgement' column to 'if_host'
14     df.rename(columns={'Judgement': 'if_host'}, inplace=True)
15
16     # Write the DataFrame to a new CSV file
17     df.to_csv('F:/Desktop/MCM2025/2025-MCM-C/2025_Problem_C_Data/if_host.csv',
index=False, encoding='utf-8')
18
19 if __name__ == "__main__":
20     main()

```

Listing 13: if\_host.py

```

1  import pandas as pd

```

```
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 df_program = pd.read_csv('C:/Users/admin/Desktop/MCM2025/2025_Problem_C_Data/
    summerOly_programs_v3.0.csv')
6 df_athletes = pd.read_csv('C:/Users/admin/Desktop/MCM2025/2025_Problem_C_Data/
    summerOly_athletes_v3.0.csv')
7 df_host = pd.read_csv('C:/Users/admin/Desktop/MCM2025/2025_Problem_C_Data/
    summerOly_hosts_completed.csv')
8 df_medal_c = pd.read_csv('C:/Users/admin/Desktop/MCM2025/2025_Problem_C_Data/
    summerOly_medal_counts_completed.csv')
9 df_program_plot = pd.read_csv('C:/Users/admin/Desktop/MCM2025/2025_Problem_C_Data
    /program_plot_end.csv')
10
11 print(df_program.head())
12 print(df_athletes.head())
13 print(df_host.head())
14 print(df_medal_c.head())
15
16 # investigate the sports in athletes file but not in the program file
17 print(set(df_athletes['Sport'].unique())-set(df_program['Sport'].unique()),
18       '\n'
19
20
21       ,len(set(df_athletes['Sport'].unique())-set(df_program['Sport'].unique()))
22
23 print(set(df_program['Sport'].unique()))
24 print(len(set(df_program['Sport'].unique())))
25
26 def hosting_list_vs_average_list(year, df_program_plot, df_host, show_plot=False)
27 :
28     '''
29     This function looks into the chosen year and compares the variation of sports
30     events of this certain index of Olympic Games
31     with the average variation of sports events of 2 indices before and after the
32     certain one.
33
34     If the chosen year is the first or the 2nd one, the function will compare the
35     variation with the average variation of
36     the next 2 indices only. If the chosen year is the last or the 2nd last one,
37     the function will compare the variation with the
38     average variation of the previous 2 indices only. The function will return
39     the list of sports events that are held in the
40     chosen year but not in the average list.
```

```
36     Finally, plot in one histogram the variation of sports events in the chosen
37     year and the average distribution. Also, return the
38     host country of the chosen year, and the list demonstrating the variation of
39     list of event.
40
41     year: int, the chosen year
42     df_program_plot: DataFrame from program_plot_end.csv
43     df_host: DataFrame from summerOly_hosts_completed.csv
44     show_plot: bool, whether to display the plot (default: False)
45     '''
46
47     # Get year column index
48     years = df_program_plot.columns[1:].tolist()
49     year_str = str(year)
50     if year_str not in years:
51         print(f"Year {year} not found in df_program_plot.")
52         return None, None
53     year_index = years.index(year_str) + 1
54
55     # Determine surrounding years
56     total_years = len(years)
57     if year_index < 3:
58         surrounding_years = [year_index + 1, year_index + 2]
59     elif year_index >= total_years - 1:
60         surrounding_years = [year_index - 2, year_index - 1]
61     else:
62         surrounding_years = [year_index - 2, year_index - 1, year_index + 1,
63                             year_index + 2]
64
65     # Get sports events for chosen year
66     events_chosen = df_program_plot.iloc[:, year_index]
67
68     # Get sports events for surrounding years
69     events_surrounding = df_program_plot.iloc[:, surrounding_years]
70     # Get average sports events for surrounding years
71     events_avg = events_surrounding.mean(axis=1)
72
73     # Compare the events_chosen with events_avg through histogram, one sport by
74     # another, with each names on x-axis,
75     if show_plot:
76         fig, ax = plt.subplots(figsize=(10, 6))
77         x = np.arange(len(events_chosen))
78         width = 0.35
79         ax.bar(x - width/2, events_chosen, width, label=str(year))
80         ax.bar(x + width/2, events_avg, width, label='Average')
```



```
77     ax.set_xticklabels(df_program_plot['Sport'], rotation=45, ha='right')
78     ax.legend()
79     plt.title(f"Sports Events Variation in {year} vs Average")
80     plt.show()
81
82     # Get host country
83     host_country = df_host[df_host['Year'] == year]
84     host_country = host_country['Country'].values[0] if not host_country.empty
85     else 'Unknown'
86
87     # Get the change of sports events
88     list_change = events_chosen - events_avg
89
90     return host_country, list_change
91
92 host_country, list_change = hosting_list_vs_average_list(1908, df_program_plot,
93     df_host, show_plot=True)
94 print(f"Host country: {host_country}")
95
96 def search(NOC_x, Year):
97     df = pd.read_csv('C:/Users/admin/Desktop/MCM2025/2025_Problem_C_Data/
98     score_rate.csv')
99
100     df = df[(df['NOC_x'] == NOC_x)]
101     df = df[(df['Year'] == Year)]
102     df = df[['Sport', 'Year', 'NOC_x', 'Score_Rate']]
103     return df
104 print(search('USA', 1908))
105
106 def compare(year, df_program_plot, df_host):
107     # Get the host country and list change for the given year
108     host_country, list_change = hosting_list_vs_average_list(year,
109     df_program_plot, df_host)
110
111     # Ensure list_change is always a list (empty if None)
112     list_change = list(list_change) if list_change is not None else []
113     # Get the host country for the current year from df_host
114     host_data = df_host[df_host['Year'] == year]
115     if not host_data.empty:
116         host_country = host_data['NOC'].values[0]
117     else:
118         host_country = 'Unknown'
119
120     # Get previous Olympics' score rates (4 years ago and 8 years ago)
121     score_rate_4_years_ago = search(host_country, year - 4)
```

```
118     score_rate_8_years_ago = search(host_country, year - 8)
119
120     # Extract the sports in the program for the given year
121     sports = set(df_program_plot['Sport'].unique())
122
123     # Initialize the list to store the calculated award rates
124     award_rates = []
125
126     # Calculate and store the award rates for each sport
127     for sport in sports:
128         # Filter score rates for 4 and 8 years ago for the given sport
129         score_rate_4_years_ago_sport = score_rate_4_years_ago[
130 score_rate_4_years_ago['Sport'] == sport]
131         score_rate_8_years_ago_sport = score_rate_8_years_ago[
132 score_rate_8_years_ago['Sport'] == sport]
133
134         # Default values for 4 years ago and 8 years ago (if no data is available
135         )
136         score_rate_4 = 0
137         score_rate_8 = 0
138
139         # Assign values if data exists
140         if not score_rate_4_years_ago_sport.empty:
141             score_rate_4 = score_rate_4_years_ago_sport['Score_Rate'].values[0]
142
143         if not score_rate_8_years_ago_sport.empty:
144             score_rate_8 = score_rate_8_years_ago_sport['Score_Rate'].values[0]
145
146         # Calculate weighted score
147         weighted_score = (score_rate_8 * 0.3) + (score_rate_4 * 0.7)
148
149         # Append the calculated score rate to the award_rates list
150         award_rates.append(weighted_score)
151     return host_country, award_rates
152
153 # Example usage
154 host_country, award_rates = compare(1996, df_program_plot, df_host)
155
156 # Example output (award_rates will be a list of calculated award rates)
157 print(f"Calculated Award Rates: {award_rates}")
158
159 from scipy.stats import spearmanr
160
161 # Function to replace negative numbers with 0
162 def replace_negative_with_zero(lst):
```

```
160     return [max(0, x) for x in lst]
161
162 # Call the compare function and calculate Spearman correlation coefficient
163 def calculate_spearman_for_year(year, df_program_plot, df_host):
164     # Step 1: Call the compare function to get list_change and award_rates
165     host_country, list_change = compare(year, df_program_plot, df_host) # The
    result includes list_change
166
167     # Step 2: Modify list_change by replacing negative numbers with 0
168     list_change = replace_negative_with_zero(list_change) # list_change is a
    numerical list
169
170     # Step 3: Calculate Spearman correlation coefficient
171     spearman_corr, _ = spearmanr(list_change, award_rates) # award_rates is
    returned from the compare function
172
173     # Return the Spearman correlation coefficient
174     return spearman_corr
175
176 # Get all Olympic years (assuming the years are already present in df_host)
177 years = df_host['Year'].unique()
178
179 # Iterate through all Olympic years and calculate Spearman correlation
    coefficient
180 for year in years:
181     spearman_corr = calculate_spearman_for_year(year, df_program_plot, df_host)
182     print(f"Spearman correlation for {year}: {spearman_corr}")
183
184 from fastdtw import fastdtw
185 from scipy.spatial.distance import euclidean
186
187 def DTW(year, df_program_plot, df_host):
188     host_country, list_change = hosting_list_vs_average_list(year,
    df_program_plot, df_host)
189     # Ensure list_change is always a list (empty if None)
190     if list_change is None:
191         list_change = []
192     else:
193         list_change = list(list_change) # No conflict with list here
194
195     # Replace negative values with zero
196     cleaned_list = replace_negative_with_zero(list_change)
197
198     # Print the cleaned list
199     print("Cleaned list_change:", cleaned_list)
```

```
200
201     # Obtain award rates
202     host_country, award_rates = compare(1996, df_program_plot, df_host)
203
204     # Ensure both lists are of the same length for DTW (if not, you can pad with
205     # zeros or truncate)
206     min_len = min(len(cleaned_list), len(award_rates))
207     list1 = cleaned_list[:min_len]
208     list2 = award_rates[:min_len]
209
210     # Ensure the lists are in the correct shape for DTW (1D lists of numbers)
211     # Each list element should be a scalar, not an array, hence the following
212     # line:
213     list1 = [(x,) for x in list1] # Convert each number to a tuple (x,)
214     list2 = [(x,) for x in list2] # Convert each number to a tuple (x,)
215
216     # Calculate the Dynamic Time Warping distance
217     distance, path = fastdtw(list1, list2, dist=euclidean)
218
219     # Print the DTW distance
220     print(f"DTW distance for {year}: {distance}")
221
222 years = df_host['Year'].unique()
223
224 for year in years:
225     DTW(year, df_program_plot, df_host)
226
227 import numpy as np
228
229 def fourier_transform(year, df_program_plot, df_host):
230     # Get the list_change data
231     host_country, list_change = hosting_list_vs_average_list(year,
232     df_program_plot, df_host)
233
234     # Ensure list_change is a list
235     if list_change is None:
236         list_change = []
237     else:
238         list_change = list(list_change)
239
240     # Replace negative values with 0
241     cleaned_list = replace_negative_with_zero(list_change)
```

```
242 # Get award_rates
243 host_country, award_rates = compare(1996, df_program_plot, df_host)
244
245 # Ensure both lists have the same length
246 min_len = min(len(cleaned_list), len(award_rates))
247 list1 = cleaned_list[:min_len]
248 list2 = award_rates[:min_len]
249
250 # Apply Fourier Transform to both time series
251 fft_list1 = np.fft.fft(list1)
252 fft_list2 = np.fft.fft(list2)
253
254 # Calculate the Euclidean distance between the two spectra (other distance
metrics can be used)
255 distance = np.linalg.norm(np.abs(fft_list1) - np.abs(fft_list2))
256
257 # Output the distance after Fourier Transform
258 print(f"Fourier Transform distance for {year}: {distance}")
259
260 # Get all Olympic years (assuming the years are already present in df_host)
261 years = df_host['Year'].unique()
262
263 # Iterate through all Olympic years and calculate the Fourier Transform distance
264 for year in years:
265     fourier_transform(year, df_program_plot, df_host)
266
267 import numpy as np
268
269 def shape_based_distance(year, df_program_plot, df_host):
270     # Retrieve the list_change data
271     host_country, list_change = hosting_list_vs_average_list(year,
df_program_plot, df_host)
272
273     # Ensure list_change is a list
274     if list_change is None:
275         list_change = []
276     else:
277         list_change = list(list_change)
278
279     # Replace negative values with 0
280     cleaned_list = replace_negative_with_zero(list_change)
281
282     # Retrieve award_rates
283     host_country, award_rates = compare(1996, df_program_plot, df_host)
284
```

```
285     # Ensure both lists have the same length
286     min_len = min(len(cleaned_list), len(award_rates))
287     list1 = cleaned_list[:min_len]
288     list2 = award_rates[:min_len]
289
290     # Calculate the Euclidean distance between the two time series (shape-based
291     # distance)
292     distance = np.linalg.norm(np.array(list1) - np.array(list2))
293
294     # Output the shape-based distance
295     print(f"Shape-Based distance for {year}: {distance}")
296
297 # Get all Olympic years (assuming the years are already present in df_host)
298 years = df_host['Year'].unique()
299
300 # Iterate through all Olympic years and calculate the shape-based distance
301 for year in years:
302     shape_based_distance(year, df_program_plot, df_host)
303
304 import pandas as pd
305
306 # Read score_rate.csv
307 score_rate_df = pd.read_csv('C:/Users/admin/Desktop/MCM2025/2025_Problem_C_Data/
308     score_rate.csv')
309
310 # Read summerOly_hosts_completed.csv
311 hosts_df = pd.read_csv('C:/Users/admin/Desktop/MCM2025/2025_Problem_C_Data/
312     summerOly_hosts_completed.csv')
313
314 # We only need the NOC and Year columns to determine the host country
315 # Merge the data: Merge based on Year and NOC
316 merged_df = pd.merge(score_rate_df, hosts_df[['Year', 'NOC']], on='Year', how='
317     left')
318
319 # Add a new column in merged_df to check if it's the host country
320 merged_df['Host'] = merged_df.apply(lambda row: 1 if row['NOC_x'] == row['NOC']
321     else 0, axis=1)
322
323 # Drop intermediate columns created during the merge
324 merged_df = merged_df.drop(columns=['NOC'])
325
326 # Output the merged DataFrame to check the result
327 print(merged_df)
328
329 # Optionally save the result to a new CSV file
```

```
325 merged_df.to_csv('score_rate_with_host.csv', index=False)
326
327 import pandas as pd
328
329 # 1. Read data
330 if_host_df = pd.read_csv('C:/Users/admin/Desktop/MCM2025/MCM2025/2025
    _Problem_C_Data/if_host.csv')
331 sport_df = pd.read_csv('C:/Users/admin/Desktop/MCM2025/MCM2025/2025
    _Problem_C_Data/score_rate_with_host.csv')
332 host_complete_df = pd.read_csv('C:/Users/admin/Desktop/MCM2025/MCM2025/2025
    _Problem_C_Data/summerOly_hosts_completed.csv')
333
334 # 2. Merge the if_host data with the complete host city data
335 merged_df = pd.merge(if_host_df, sport_df, how='left', left_on=['Year', 'NOC_x'],
    right_on=['Year', 'NOC_x'])
336
337 # 3. Filter rows where the distance is 0, keeping only Olympic events hosted by
    the country
338 filtered_df = merged_df[merged_df['distance'] == 0]
339
340 # 4. Calculate the total Score_Rate for all events a country participated in
    during each Olympic year
341 # Group by country (NOC_x) and year, then compute the sum of Score_Rate
342 score_rate_sum_per_country_year = filtered_df.groupby(['NOC_x', 'Year'])['
    Score_Rate'].sum().reset_index()
343
344 # 5. Calculate the average of total Score_Rate sums for all Olympic years a
    country participated in
345 # First, group by country to compute the mean of Score_Rate sums
346 score_rate_avg_per_country = score_rate_sum_per_country_year.groupby('NOC_x')['
    Score_Rate'].mean().reset_index()
347
348 # 6. Rename the column to make it more descriptive
349 score_rate_avg_per_country.rename(columns={'Score_Rate': 'Average_Score_Rate_Sum'
    }, inplace=True)
350
351 # 7. Save the result to an Excel file
352 score_rate_avg_per_country.to_excel('average_score_rate_sum_by_noc.xlsx', index=
    False)
353
354 print("Excel file generated: average_score_rate_sum_by_noc.xlsx")
355
356 # Filter rows where the distance is not 0
357 filtered_if_host_df = if_host_df[if_host_df['distance'] != 0]
358
```

```
359 # 3. Merge the filtered if_host data with sport data to retrieve each event's
      Score_Rate, along with corresponding NOC and Year
360 merged_df = pd.merge(filtered_if_host_df, sport_df, how='left', left_on=['Year',
      'NOC_x'], right_on=['Year', 'NOC_x'])
361
362 # 4. Calculate the total Score_Rate for all events a country participated in
      during each Olympic year, keeping the distance column
363 score_rate_sum_per_country_year = merged_df.groupby(['NOC_x', 'Year', 'distance'
      ])[ 'Score_Rate' ].sum().reset_index()
364
365 # 5. Rename the column to make it more descriptive
366 score_rate_sum_per_country_year.rename(columns={'Score_Rate': 'Score_Rate_Sum'},
      inplace=True)
367
368 # 6. Save the result to an Excel file
369 score_rate_sum_per_country_year.to_excel('
      score_rate_sum_for_hosting_countries_with_distance.xlsx', index=False)
370
371 print("Excel file generated: score_rate_sum_for_hosting_countries_with_distance.
      xlsx")
372
373
374 import pandas as pd
375 import numpy as np
376 import matplotlib.pyplot as plt
377 import seaborn as sns
378
379 # 1. Read the data
380 score_rate_df = pd.read_excel('C:/Users/admin/Desktop/MCM2025/MCM2025/2025
      _Problem_C_Data/score_rate_sum_for_hosting_countries_with_distance.xlsx')
381 average_score_df = pd.read_excel('C:/Users/admin/Desktop/MCM2025/MCM2025/2025
      _Problem_C_Data/average_score_rate_sum_nonhost.xlsx')
382
383 # 2. Merge data based on NOC_x using a left join, obtaining S1 (Score_Rate_Sum)
      and S2 (Average_Score_Rate_Sum) for each country
384 merged_df = pd.merge(score_rate_df, average_score_df, how='left', left_on='NOC_x',
      , right_on='NOC_x')
385
386 # 3. Define a function to calculate P(ch)
387 def calculate_P(ch, df):
388     P_ch = 0
389     for _, row in df.iterrows():
390         S1 = row['Score_Rate_Sum'] # Score_Rate_Sum
391         S2 = row['Average_Score_Rate_Sum'] # Average_Score_Rate_Sum
392         distance = row['distance']
```



```
393         P_ch += (S1 / (1 + distance * ch) - S2) ** 2
394     return P_ch
395
396 # 4. Iterate through different ch values, calculate P(ch), and store each P(ch)
397     value
398 ch_values = np.linspace(0, 10, 100) # Adjust range and step size as needed
399 P_values = []
400
401 for ch in ch_values:
402     P_ch = calculate_P(ch, merged_df)
403     P_values.append(P_ch)
404
405 # 5. Find the maximum and minimum P(ch) values
406 max_P = max(P_values)
407 min_P = min(P_values)
408 print(f"Maximum P(ch): {max_P}, Minimum P(ch): {min_P}")
409
410 # 6. Plot a boxplot for P(ch) values
411 plt.figure(figsize=(10, 6))
412 sns.boxplot(data=P_values, color="lightblue")
413 plt.title('P(ch) Boxplot')
414 plt.xlabel('P(ch) Values')
415 plt.show()
416
417 # 7. Calculate the IQR (Inter-Quartile Range) to detect outliers
418 Q1 = np.percentile(P_values, 25)
419 Q3 = np.percentile(P_values, 75)
420 IQR = Q3 - Q1
421 lower_bound = Q1 - 1.5 * IQR
422 upper_bound = Q3 + 1.5 * IQR
423
424 # 8. Remove outliers from P(ch) values
425 filtered_P_values = [P for P in P_values if lower_bound <= P <= upper_bound]
426
427 # 9. Recalculate the minimum P(ch) and its corresponding ch value (after removing
428     outliers)
429 min_P_filtered = min(filtered_P_values)
430 best_ch_filtered = ch_values[P_values.index(min_P_filtered)]
431
432 print(f"After removing outliers, the minimum P(ch) is {min_P_filtered},
433       corresponding to ch value {best_ch_filtered}")
434
435 import pandas as pd
436 import numpy as np
```

```
435 # Load the data
436 df = pd.read_csv('feature_noc.csv')
437 athletes_df = pd.read_csv('C:/Users/admin/Desktop/MCM2025/MCM2025/2025
    _Problem_C_Data/summerOly_athletes_completed.csv')
438
439 # List of sports in the specified order
440 sports = [
441     "Aquatics", "Archery", "Athletics", "Badminton", "Baseball and Softball", "
    Basketball",
442     "Boxing", "Breaking", "Canoeing", "Cycling", "Equestrian", "Fencing", "Field
    hockey",
443     "Flag football", "Football", "Golf", "Gymnastics", "Handball", "Judo", "
    Karate",
444     "Modern Pentathlon", "Rowing", "Rugby", "Sailing", "Shooting", "Skateboarding
    ",
445     "Sport Climbing", "Surfing", "Table Tennis", "Taekwondo", "Tennis", "
    Triathlon",
446     "Volleyball", "Weightlifting", "Wrestling"
447 ]
448
449 # Initialize lists to store participation and changes
450 changes_list = []
451 participation_list = []
452
453 # Get all unique years and sort them
454 years = df['Year'].unique()
455 years.sort()
456
457 # Iterate through each year in the data
458 for i, year in enumerate(years):
459     # Filter the data for the current year
460     current_year_data = athletes_df[athletes_df['Year'] == year]
461
462     # Initialize participation counts for all sports (default to 0)
463     participation_counts = [0] * len(sports)
464     for sport in current_year_data['Sport'].unique():
465         if sport in sports: # Only include the specified sports
466             participation_counts[sports.index(sport)] = current_year_data[
    current_year_data['Sport'] == sport].shape[0]
467
468     # Handle the first Olympic year
469     if i == 0:
470         # For the first year, the participation increase is the same as
    participation counts (no prior data)
471         participation_increase = participation_counts
```

```
472     else:
473         prev_year = year - 4 # The year of the previous Olympics
474
475         # Check if the previous year has data
476         if prev_year in athletes_df['Year'].unique():
477             prev_year_data = athletes_df[athletes_df['Year'] == prev_year]
478
479             # Initialize participation counts for the previous year
480             prev_participation_counts = [0] * len(sports)
481             for sport in prev_year_data['Sport'].unique():
482                 if sport in sports:
483                     prev_participation_counts[sports.index(sport)] =
484 prev_year_data[prev_year_data['Sport'] == sport].shape[0]
485
486             # Calculate participation increase: current year - previous year
487             participation_increase = [current - prev for current, prev in zip(
488 participation_counts, prev_participation_counts)]
489
490         else:
491             # If no data for the previous year, assume participation counts were
492 0
493             participation_increase = participation_counts
494
495         # Add participation increase to the list for all rows of the current year
496         participation_list.extend([participation_increase] * len(df[df['Year'] ==
497 year])))
498
499         # Handle changes in events (Event_Changes)
500         if i == 0:
501             # For the first year, all changes are 0
502             changes = [0] * len(sports)
503             changes_list.extend([changes] * len(df[df['Year'] == year]))
504         else:
505             prev_year = year - 4
506
507             if prev_year in athletes_df['Year'].unique():
508                 prev_year_data = athletes_df[athletes_df['Year'] == prev_year]
509
510                 # Initialize counts for the previous year's sports
511                 prev_sports_count = [0] * len(sports)
512                 for sport in prev_year_data['Sport'].unique():
513                     if sport in sports:
514                         prev_sports_count[sports.index(sport)] = prev_year_data[
515 prev_year_data['Sport'] == sport].shape[0]
516
517                 # Calculate changes in events: current year - previous year
```

```
512         changes = [current - prev for current, prev in zip(
    participation_counts, prev_sports_count)]
513     else:
514         # If no data for the previous year, assume changes are equal to
    participation counts
515         changes = participation_counts
516
517         # Add changes to the list for all rows of the current year
    changes_list.extend([changes] * len(df[df['Year'] == year]))
519
520 # Add participation increase and event changes as new columns in the DataFrame
521 df['Participation_List'] = participation_list
522 df['Event_Changes'] = changes_list
523
524 # Save the updated DataFrame to a CSV file
525 df.to_csv('feature_noc(2).csv', index=False)
526
527 import pandas as pd
528 import ast
529
530 # Read the CSV file
531 df = pd.read_csv('feature_Noc.csv')
532
533 # Initialize dictionaries to store the number of participating countries and
    athletes per year
534 participating_countries = {}
535 participating_athletes = {}
536
537 # Iterate through the data to calculate the number of participating countries and
    athletes per year
538 for index, row in df.iterrows():
539     year = row['Year']
540     participatelist = ast.literal_eval(row['Participants_List']) # Convert
    string to a list
541
542     # Initialize data for the year if not already in the dictionaries
543     if year not in participating_countries:
544         participating_countries[year] = [0] * 34 # Initialize with 34 sports (or
    categories)
545         participating_athletes[year] = [0] * 34 # Same initialization for
    athletes
546
547     # Update the number of participating countries and athletes for each sport
548     for i, participants in enumerate(participatelist):
549         participants = int(participants) # Ensure participant count is an
```

```
integer
550     if participants > 0:
551         participating_countries[year][i] += 1 # Increment the country count
552     if participants > 0
553         participating_athletes[year][i] += participants # Add the number of
554         participants
555 # Function to calculate the change between two years
556 def calculate_changes(year, previous_year, current_data, previous_data):
557     if previous_year not in previous_data: # If there is no data for the
558         previous year
559         return [x for x in current_data] # Return the current data as the change
560         # Calculate the change by subtracting previous year data from current year
561         data
562     return [current - previous for current, previous in zip(current_data,
563         previous_data[previous_year])]
564
565 # Add new columns for changes
566 country_changes = []
567 athlete_changes = []
568
569 for index, row in df.iterrows():
570     year = row['Year']
571     previous_year = year - 4 # Olympics occur every 4 years
572     participatelist = ast.literal_eval(row['Participants_List']) # Convert
573     string to a list
574
575     # Calculate changes for countries and athletes
576     country_change = calculate_changes(year, previous_year,
577     participating_countries[year], participating_countries)
578     athlete_change = calculate_changes(year, previous_year,
579     participating_athletes[year], participating_athletes)
580
581     # Append the changes to the respective lists
582     country_changes.append(country_change)
583     athlete_changes.append(athlete_change)
584
585 # Add the new columns to the DataFrame
586 df['CountryChange'] = country_changes
587 df['AthleteChange'] = athlete_changes
588
589 # Save the updated DataFrame to a new CSV file
590 df.to_csv('feature_Noc_with_changes.csv', index=False)
```

```
585 print("Processing complete. Results saved to 'feature_Noc_with_changes.csv'")
```

Listing 14: data\_analyze.ipynb

```
1 import numpy as np
2 import pandas as pd
3 from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
4 from sklearn.model_selection import train_test_split, cross_val_score
5 from sklearn.preprocessing import StandardScaler
6 from tensorflow.keras.models import Sequential
7 from tensorflow.keras.layers import LSTM, Dense
8
9 class OlympicPredictionModel:
10     def __init__(self, lambda_weight=0.3, mu_weight=0.85):
11         """
12         Initialize model weights
13         :param lambda_weight: Small country weight
14         :param mu_weight: Large country weight
15         """
16         self.lambda_weight = lambda_weight
17         self.mu_weight = mu_weight
18         self.classifier = RandomForestClassifier(n_estimators=200)
19         self.regressor1 = RandomForestRegressor(n_estimators=200)
20         self.regressor2 = RandomForestRegressor(n_estimators=200)
21         self.lstm_models = {}
22
23     @staticmethod
24     def calculate_w4_vector(country_change, athlete_change, event_changes):
25         """
26         Vectorized calculation of W4
27         """
28         w1 = np.sum(country_change, axis=1).astype(np.float64)
29         w2 = np.sum(athlete_change, axis=1).astype(np.float64)
30         w3 = np.sum(event_changes, axis=1).astype(np.float64)
31         w4 = np.divide(0.2 * w1 + 0.8 * w2, w3, out=np.zeros_like(w3, dtype=np.
float64), where=w3 != 0)
32         return w4
33
34     def preprocess_data(self, data):
35         """
36         Preprocess data, including calculating features A, B, and C
37         """
38         # Feature A
39         score_rate_list = np.array(data['Score_Rate_List'].apply(eval).tolist())
40         score_list = np.array(data['Score_List'].apply(eval).tolist(), dtype=np.
float64)
```

```
41     participants_list = np.array(data['Participants_List'].apply(eval).tolist
    (), dtype=np.float64)
42
43     v5_5_a = np.sum(score_rate_list, axis=1)
44     v8_5_a = np.sum(np.divide(score_list, participants_list,
45                               out=np.zeros_like(score_list, dtype=np.float64)
    ,
46                               where=participants_list > 0), axis=1)
47
48     country_change = np.array(data['CountryChange'].apply(eval).tolist())
49     athlete_change = np.array(data['AthleteChange'].apply(eval).tolist())
50     event_changes = np.array(data['Event_Changes'].apply(eval).tolist())
51     w4 = self.calculate_w4_vector(country_change, athlete_change,
    event_changes)
52
53     v8_w4 = np.sum(np.multiply(
54         np.divide(score_list, participants_list, out=np.zeros_like(score_list
    , dtype=np.float64), where=participants_list > 0),
55         w4[:, None]
56     ), axis=1)
57
58     features_a = np.column_stack((v5_5_a, v8_5_a, v8_w4))
59
60     # Feature B
61     v1_b = data['if_host'].values
62     v5_5_b = np.sum(participants_list, axis=1)
63     v6_b = data['Gold_prev'].fillna(0).values
64     v7_b = data['distance'].values
65     v3_5_b = data['Score'].values
66     v3_w4_b = np.sum(np.multiply(score_list, w4[:, None]), axis=1)
67
68     features_b = np.column_stack((v1_b, v5_5_b, v6_b, v7_b, v3_5_b, v3_w4_b))
69
70     # Feature C (same as B)
71     features_c = features_b.copy()
72
73     # Labels and expectations
74     labels = (data['Total_Medals'] > 0).astype(int).values
75     medal_expectation = data['Total_Medals'].values
76     gold_expectation = data['Gold'].values
77
78     scaler = StandardScaler()
79     features_a_scaled = scaler.fit_transform(features_a)
80     features_b_scaled = scaler.fit_transform(features_b)
81     features_c_scaled = scaler.fit_transform(features_c)
```

```
82         return features_a_scaled, features_b_scaled, features_c_scaled, labels,
83         medal_expectation, gold_expectation
84
85     def train_classifier(self, features_a, other_features, labels):
86         """
87         Train the classifier to predict medal-winning probability
88         """
89         input_features = np.hstack([
90             self.lambda_weight * features_a,
91             (1 - self.lambda_weight) * other_features
92         ])
93         self.classifier.fit(input_features, labels)
94
95     def train_regressor1(self, classifier_output, features_b, medal_expectation):
96         """
97         Train regressor 1 to predict medal expectation
98         """
99         input_features = np.hstack([classifier_output.reshape(-1, 1), features_b
100 ])
101         self.regressor1.fit(input_features, medal_expectation)
102
103     def train_regressor2(self, medal_expectation, features_c, gold_expectation):
104         """
105         Train regressor 2 to predict gold medal expectation
106         """
107         input_features = np.hstack([
108             (1 - self.mu_weight) * medal_expectation.reshape(-1, 1),
109             self.mu_weight * features_c
110         ])
111         self.regressor2.fit(input_features, gold_expectation)
112
113     def train_lstm(self, features, feature_name):
114         """
115         Train an LSTM model to predict given features
116         :param features: Historical feature data
117         :param feature_name: Feature name (A, B, C)
118         """
119         X = features[:-1]
120         y = features[1:]
121
122         X = X.reshape((X.shape[0], X.shape[1], 1))
123
124         model = Sequential([
125             LSTM(50, activation='relu', input_shape=(X.shape[1], 1)),
```



```
125         Dense(y.shape[1])
126     ])
127     model.compile(optimizer='adam', loss='mse')
128     model.fit(X, y, epochs=50, verbose=0)
129
130     self.lstm_models[feature_name] = model
131
132     def predict_future_features(self, features, feature_name):
133         """
134         Use LSTM models to predict future features
135         """
136         model = self.lstm_models[feature_name]
137         features = features.reshape((features.shape[0], features.shape[1], 1))
138         return model.predict(features)
139
140     def predict(self, features_a, features_b, features_c):
141         """
142         Predict 2028 Olympics results
143         """
144         input_features = np.hstack([features_a, features_b])
145         classifier_output = self.classifier.predict_proba(input_features)[: , 1]
146         medal_expectation = self.regressor1.predict(np.hstack([classifier_output.
147 reshape(-1, 1), features_b]))
148         gold_expectation = self.regressor2.predict(np.hstack([
149             (1 - self.mu_weight) * medal_expectation.reshape(-1, 1),
150             self.mu_weight * features_c
151         ]))
152         return medal_expectation, gold_expectation
153
154     def plot_n_estimators_curve(features_a, features_b, labels, n_range=range(10,
155 500, 20)):
156         """
157         Plot Random Forest learning curve for different tree counts
158         """
159         import matplotlib.pyplot as plt
160         from sklearn.model_selection import cross_val_score
161
162         train_scores = []
163         cv_scores = []
164         input_features = np.hstack([features_a, features_b])
165
166         for n in n_range:
167             clf = RandomForestClassifier(n_estimators=n, random_state=42)
168             clf.fit(input_features, labels)
169             train_scores.append(clf.score(input_features, labels))
```

```
168         cv_score = cross_val_score(clf, input_features, labels, cv=5).mean()
169         cv_scores.append(cv_score)
170         print(f"Trees: {n}, Train Score: {train_scores[-1]:.4f}, CV Score: {
cv_scores[-1]:.4f}")
171
172     plt.figure(figsize=(10, 6))
173     plt.plot(n_range, train_scores, label='Training Score')
174     plt.plot(n_range, cv_scores, label='Cross-Validation Score')
175     plt.xlabel('Number of Trees')
176     plt.ylabel('Score')
177     plt.title('Random Forest Performance vs Number of Trees')
178     plt.legend()
179     plt.grid(True)
180     plt.show()
181
182 # Example: Load data and train the model
183 if __name__ == "__main__":
184     # Load historical data
185     data = pd.read_csv("C:/Users/admin/Desktop/MCM2025/MCM2025/2025
_Problem_C_Data/feature_Noc.csv")
186
187     model = OlympicPredictionModel(lambda_weight=0.3, mu_weight=0.85)
188     features_a, features_b, features_c, labels, medal_expectation,
gold_expectation = model.preprocess_data(data)
189
190     # Train classifier and regressors
191     model.train_classifier(features_a, features_b, labels)
192     classifier_output = model.classifier.predict_proba(np.hstack([features_a,
features_b]))[:, 1]
193     model.train_regressor1(classifier_output, features_b, medal_expectation)
194     model.train_regressor2(medal_expectation, features_c, gold_expectation)
195
196     # Cross-validation for classifier
197     scores_classifier = cross_val_score(model.classifier, np.hstack([features_a,
features_b]), labels, cv=5)
198     print(f'Classifier CV Accuracy: {np.mean(scores_classifier):.4f}')
199
200     # Cross-validation for regressor 1
201     X_reg1 = np.hstack([classifier_output.reshape(-1, 1), features_b])
202     scores_regressor1 = cross_val_score(model.regressor1, X_reg1,
medal_expectation, cv=5)
203     print(f'Regressor1 CV R^2: {np.mean(scores_regressor1):.4f}')
204
205     # Cross-validation for regressor 2
206     X_reg2 = np.hstack([(1 - model.mu_weight) * medal_expectation.reshape(-1, 1),
```

```

    model.mu_weight * features_c])
207     scores_regressor2 = cross_val_score(model.regressor2, X_reg2,
gold_expectation, cv=5)
208     print(f'Regressor2 CV R^2: {np.mean(scores_regressor2):.4f}')
209
210     # Plot learning curve
211     print("\nPlotting Random Forest learning curve:")
212     plot_n_estimators_curve(features_a, features_b, labels)
213
214     import numpy as np
215     import pandas as pd
216     from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
217     from sklearn.preprocessing import StandardScaler
218     from sklearn.model_selection import cross_val_score
219     import matplotlib.pyplot as plt
220     from tensorflow.keras.models import Sequential
221     from tensorflow.keras.layers import LSTM, Dense
222
223     class OlympicPredictionModel:
224         def __init__(self, lambda_weight=0.3, mu_weight=0.85):
225             self.lambda_weight = lambda_weight
226             self.mu_weight = mu_weight
227             self.classifier = RandomForestClassifier(n_estimators=300)
228             self.regressor1 = RandomForestRegressor()
229             self.regressor2 = RandomForestRegressor()
230             self.lstm_models = {}
231
232         @staticmethod
233         def calculate_w4_vector(country_change, athlete_change, event_changes):
234             w1 = np.sum(country_change, axis=1).astype(np.float64)
235             w2 = np.sum(athlete_change, axis=1).astype(np.float64)
236             w3 = np.sum(event_changes, axis=1).astype(np.float64)
237             w4 = np.divide(0.2 * w1 + 0.8 * w2, w3, out=np.zeros_like(w3, dtype=np.
float64), where=w3 != 0)
238             return w4
239
240         def preprocess_data(self, data):
241             """Data preprocessing: using data from the previous Olympic Games to
predict the current one."""
242             # Filter out data prior to 1964
243             data = data[data['Year'] >= 1964].copy()
244
245             # Calculate feature A
246             score_rate_list = np.array(data['Score_Rate_List'].apply(eval).tolist())
247             score_list = np.array(data['Score_List'].apply(eval).tolist(), dtype=np.

```

```

float64)
248     participants_list = np.array(data['Participants_List'].apply(eval).tolist
    ), dtype=np.float64)
249
250     # Sort data by NOC and Year to create lagged features
251     data = data.sort_values(['NOC_x', 'Year'])
252
253     # Feature A calculations
254     v5_5_a = np.sum(score_rate_list, axis=1)
255     v8_5_a = np.sum(np.divide(score_list, participants_list,
256                             out=np.zeros_like(score_list, dtype=np.float64)
    ,
257                             where=participants_list > 0), axis=1)
258
259     country_change = np.array(data['CountryChange'].apply(eval).tolist())
260     athlete_change = np.array(data['AthleteChange'].apply(eval).tolist())
261     event_changes = np.array(data['Event_Changes'].apply(eval).tolist())
262     w4 = self.calculate_w4_vector(country_change, athlete_change,
    event_changes)
263
264     v8_w4 = np.sum(np.multiply(
265         np.divide(score_list, participants_list, out=np.zeros_like(score_list
    ), where=participants_list > 0),
266         w4[:, None]
267     ), axis=1)
268
269     features_a = np.column_stack((v5_5_a, v8_5_a, v8_w4))
270
271     # Feature B calculations
272     v1_b = data['if_host'].values
273     v5_5_b = np.sum(participants_list, axis=1)
274     v6_b = data['Gold_prev'].fillna(0).values
275     v7_b = data['distance'].values
276     v3_5_b = data['Score'].values
277     v3_w4_b = np.sum(np.multiply(score_list, w4[:, None]), axis=1)
278
279     features_b = np.column_stack((v1_b, v5_5_b, v6_b, v7_b, v3_5_b, v3_w4_b))
280
281     # Feature C is identical to Feature B
282     features_c = features_b.copy()
283
284     # Shift features by one event to align with the lagged labels
285     features_a = np.roll(features_a, 1, axis=0)
286     features_b = np.roll(features_b, 1, axis=0)
287     features_c = np.roll(features_c, 1, axis=0)

```

```
288
289     # Remove the first row (no previous data available)
290     features_a = features_a[1:]
291     features_b = features_b[1:]
292     features_c = features_c[1:]
293
294     # Adjust labels accordingly
295     labels = (data['Total_Medals'] > 0).astype(int).values[1:]
296     medal_expectation = data['Total_Medals'].values[1:]
297     gold_expectation = data['Gold'].values[1:]
298
299     # Standardize features
300     scaler = StandardScaler()
301     features_a_scaled = scaler.fit_transform(features_a)
302     features_b_scaled = scaler.fit_transform(features_b)
303     features_c_scaled = scaler.fit_transform(features_c)
304
305     return features_a_scaled, features_b_scaled, features_c_scaled, labels,
medal_expectation, gold_expectation
306
307     def train_classifier(self, features_a, other_features, labels):
308         input_features = np.hstack([
309             self.lambda_weight * features_a,
310             (1 - self.lambda_weight) * other_features
311         ])
312         self.classifier.fit(input_features, labels)
313
314     def train_regressor1(self, classifier_output, features_b, medal_expectation):
315         input_features = np.hstack([classifier_output.reshape(-1, 1), features_b
316 ])
317         self.regressor1.fit(input_features, medal_expectation)
318
319     def train_regressor2(self, medal_expectation, features_c, gold_expectation):
320         input_features = np.hstack([
321             (1 - self.mu_weight) * medal_expectation.reshape(-1, 1),
322             self.mu_weight * features_c
323         ])
324         self.regressor2.fit(input_features, gold_expectation)
325
326     def predict(self, features_a, features_b, features_c):
327         input_features = np.hstack([features_a, features_b])
328         classifier_output = self.classifier.predict_proba(input_features)[: , 1]
329         medal_expectation = self.regressor1.predict(np.hstack([classifier_output.
reshape(-1, 1), features_b]))
330         gold_expectation = self.regressor2.predict(np.hstack([
```

```
330         (1 - self.mu_weight) * medal_expectation.reshape(-1, 1),
331         self.mu_weight * features_c
332     ]))
333     return medal_expectation, gold_expectation
334
335 def plot_regressor_learning_curves(features, targets, n_range=range(10, 1000, 10)
336 ,
337                                     regressor_name="Regressor"):
338     """Plot learning curves for a Random Forest regressor."""
339     train_scores = []
340     cv_scores = []
341
342     for n in n_range:
343         regr = RandomForestRegressor(n_estimators=n, random_state=42)
344         regr.fit(features, targets)
345         train_scores.append(regr.score(features, targets))
346         cv_score = cross_val_score(regr, features, targets, cv=5).mean()
347         cv_scores.append(cv_score)
348         print(f"{regressor_name} Trees: {n}, Train Score: {train_scores[-1]:.4f},
349               CV Score: {cv_scores[-1]:.4f}")
350
351     plt.figure(figsize=(10, 6))
352     plt.plot(n_range, train_scores, label='Training Score')
353     plt.plot(n_range, cv_scores, label='Cross-Validation Score')
354     plt.xlabel('Number of Trees')
355     plt.ylabel('Score')
356     plt.title(f'{regressor_name} Performance vs Number of Trees')
357     plt.legend()
358     plt.grid(True)
359     plt.show()
360
361 if __name__ == "__main__":
362     # Load data
363     data = pd.read_csv("C:/Users/admin/Desktop/MCM2025/MCM2025/2025
364 _Problem_C_Data/feature_Noc.csv")
365
366     # Initialize the model
367     model = OlympicPredictionModel(lambda_weight=0.3, mu_weight=0.85)
368     features_a, features_b, features_c, labels, medal_expectation,
369     gold_expectation = model.preprocess_data(data)
370
371     # Train the model
372     model.train_classifier(features_a, features_b, labels)
373     classifier_output = model.classifier.predict_proba(np.hstack([features_a,
374 features_b]))[:, 1]
```

```

370     model.train_regressor1(classifier_output, features_b, medal_expectation)
371     model.train_regressor2(medal_expectation, features_c, gold_expectation)
372
373     # Cross-validation
374     input_features = np.hstack([features_a, features_b])
375     scores_classifier = cross_val_score(model.classifier, input_features, labels,
376                                         cv=5)
377     print(f'Classifier CV Accuracy: {np.mean(scores_classifier):.4f}')
378
379     X_reg1 = np.hstack([classifier_output.reshape(-1, 1), features_b])
380     scores_regressor1 = cross_val_score(model.regressor1, X_reg1,
381                                         medal_expectation, cv=5)
382     print(f'Regressor1 CV R^2: {np.mean(scores_regressor1):.4f}')
383
384     X_reg2 = np.hstack([(1 - model.mu_weight) * medal_expectation.reshape(-1, 1),
385                         model.mu_weight * features_c])
386     scores_regressor2 = cross_val_score(model.regressor2, X_reg2,
387                                         gold_expectation, cv=5)
388     print(f'Regressor2 CV R^2: {np.mean(scores_regressor2):.4f}')
389
390     # Plot learning curves for regressors
391     print("\nPlotting learning curve for Medal Regressor (Total Medals):")
392     plot_regressor_learning_curves(X_reg1, medal_expectation, regressor_name="
393     Medal Regressor")
394
395     print("\nPlotting learning curve for Gold Regressor (Gold Medals):")
396     plot_regressor_learning_curves(X_reg2, gold_expectation, regressor_name="Gold
397     Regressor")
398
399
400 class OlympicPredictionModel:
401     def __init__(self, lambda_weight=0.3, mu_weight=0.85):
402         """
403         Initialize model weights
404         :param lambda_weight: Weight for small countries
405         :param mu_weight: Weight for large countries
406         """
407         self.lambda_weight = lambda_weight
408         self.mu_weight = mu_weight
409         self.classifier = RandomForestClassifier(n_estimators=200)
410         self.regressor1 = RandomForestRegressor(n_estimators=200)
411         self.regressor2 = RandomForestRegressor(n_estimators=200)
412         self.lstm_models = {}
413
414     @staticmethod
415     def classify_country(year, noc, data):

```

```

409     """
410     Determine whether a country is a 'small country' based on participation
history
411     :param year: Current year
412     :param noc: Country code
413     :param data: Full dataset
414     :return: True if it is a small country, False otherwise
415     """
416     # Calculate the total participation count of the country up to the given
year
417     historical_data = data[data['Year'] <= year]
418     country_total = historical_data[historical_data['NOC_x'] == noc]['
Name_Count'].sum()
419
420     # Calculate total participation counts for all countries
421     all_countries = historical_data.groupby('NOC_x')['Name_Count'].sum()
422
423     # Calculate the 20th percentile threshold
424     threshold = all_countries.quantile(0.2)
425
426     return country_total <= threshold
427
428 def calculate_w4_vector(self, country_change, athlete_change, event_changes):
429     """
430     Vectorized calculation of W4
431     """
432     w1 = np.sum(country_change, axis=1).astype(np.float64)
433     w2 = np.sum(athlete_change, axis=1).astype(np.float64)
434     w3 = np.sum(event_changes, axis=1).astype(np.float64)
435     w4 = np.divide(0.2 * w1 + 0.8 * w2, w3, out=np.zeros_like(w3, dtype=np.
float64), where=w3 != 0)
436     return w4
437
438 def preprocess_data(self, data):
439     """
440     Preprocess data, including featureA, featureB, featureC, and feature_all
using reference years
441     """
442     # Create a year mapping dictionary
443     year_mapping = pd.read_csv("year_transmission_reference.csv")
444     year_dict = dict(zip(year_mapping['evaluating_year'], year_mapping['
use_data_from']))
445
446     features_list = []
447     for idx, row in data.iterrows():

```



```

448     current_year = row['Year']
449     # Get reference year data
450     reference_year = year_dict.get(current_year, current_year)
451     reference_data = data[data['Year'] == reference_year]
452
453     if len(reference_data) > 0:
454         # Use the feature data from the reference year
455         ref_row = reference_data[reference_data['NOC_x'] == row['NOC_x']]
456         ref_row = ref_row.iloc[0] if len(ref_row) > 0 else row
457     else:
458         ref_row = row # Use current year data if no reference found
459
460     # Feature A calculations
461     score_rate_list = np.array(eval(ref_row['Score_Rate_List']))
462     score_list = np.array(eval(ref_row['Score_List']), dtype=np.float64)
463     participants_list = np.array(eval(ref_row['Participants_List']),
dtype=np.float64)
464
465     v4_ = np.sum(score_rate_list)
466     v5_ = np.sum(participants_list)
467     v8_ = np.sum(np.divide(score_list, participants_list,
468                             out=np.zeros_like(score_list, dtype=np.float64
),
469                             where=participants_list > 0))
470
471     country_change = np.array(eval(ref_row['CountryChange']))
472     athlete_change = np.array(eval(ref_row['AthleteChange']))
473     event_changes = np.array(eval(ref_row['Event_Changes']))
474     w4_ = self.calculate_w4_vector(country_change.reshape(1, -1),
475                                     athlete_change.reshape(1, -1),
476                                     event_changes.reshape(1, -1))[0]
477
478     v8_w4 = np.sum(np.multiply(
479         np.divide(score_list, participants_list,
480                     out=np.zeros_like(score_list, dtype=np.float64),
481                     where=participants_list > 0),
482         w4_
483     ))
484
485     # Feature B calculations
486     v1 = ref_row['if_host']
487     v6 = ref_row['Gold_prev'] if pd.notna(ref_row['Gold_prev']) else 0
488     v7 = ref_row['distance']
489     v3_ = ref_row['Score']
490     v3_w4 = np.sum(np.multiply(score_list, w4_))

```

```
491
492     # Feature all
493     w3_ = np.sum(event_changes)
494
495     features = {
496         'features_a': [v5_, v8_, v8_w4],
497         'features_b': [v1, v5_, v6, v7, v3_, v3_w4, v8_],
498         'features_c': [v1, v5_, v6, v7, v3_, v3_w4, v8_],
499         'features_all': [v1, v3_, v4_, v5_, v6, v7, v8_, w3_, w4_]
500     }
501     features_list.append(features)
502
503     # Convert feature list into numpy arrays
504     features_a = np.array([f['features_a'] for f in features_list])
505     features_b = np.array([f['features_b'] for f in features_list])
506     features_c = np.array([f['features_c'] for f in features_list])
507     features_all = np.array([f['features_all'] for f in features_list])
508
509     # Use the labels and expected values from the current year
510     labels = (data['Total_Medals'] > 0).astype(int).values
511     medal_expectation = data['Total_Medals'].values
512     gold_expectation = data['Gold'].values
513
514     # Standardize the features
515     scaler = StandardScaler()
516     features_a_scaled = scaler.fit_transform(features_a)
517     features_b_scaled = scaler.fit_transform(features_b)
518     features_c_scaled = scaler.fit_transform(features_c)
519     features_all_scaled = scaler.fit_transform(features_all)
520
521     return features_a_scaled, features_b_scaled, features_c_scaled,
522     features_all_scaled, labels, medal_expectation, gold_expectation
523
524     # Other methods including training and prediction remain unchanged
525     # ...
526
527 import numpy as np
528 import pandas as pd
529 from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
530 from sklearn.preprocessing import StandardScaler
531 from keras.models import Model
532 from keras.layers import Input, LSTM, Dense, Dot, Activation, Concatenate
533 from keras.optimizers import Adam
534 from keras.callbacks import ReduceLROnPlateau
```

```

535 class OlympicPredictionModel:
536     def __init__(self, lambda_weight=0.3, mu_weight=0.85, learning_rate=0.001):
537         """
538         Initialize the model
539         :param lambda_weight: Weight for small countries
540         :param mu_weight: Weight for large countries
541         :param learning_rate: Learning rate for Adam optimizer
542         """
543         self.lambda_weight = lambda_weight
544         self.mu_weight = mu_weight
545         self.learning_rate = learning_rate # Learning rate for Adam optimizer
546         self.classifier = RandomForestClassifier(n_estimators=200)
547         self.regressor1 = RandomForestRegressor(n_estimators=200)
548         self.regressor2 = RandomForestRegressor(n_estimators=200)
549         self.lstm_model = None
550
551     @staticmethod
552     def Country_classification(year, Noc_x, data):
553         """
554         Determine whether a country is classified as a 'small country' based on
555         historical data
556         :param year: Current year
557         :param Noc_x: Country code
558         :param data: Dataset
559         :return: True if classified as a small country, otherwise False
560         """
561         historical_data = data[data['Year'] <= year]
562         country_total = historical_data[historical_data['NOC_x'] == Noc_x]['
563         Name_Count'].sum()
564         all_countries = historical_data.groupby('NOC_x')['Name_Count'].sum()
565         threshold = all_countries.quantile(0.2)
566         return country_total <= threshold
567
568     def calculate_w4_vector(self, country_change, athlete_change, event_changes):
569         """
570         Calculate the W4 vector
571         """
572         w1 = np.sum(country_change, axis=1).astype(np.float64)
573         w2 = np.sum(athlete_change, axis=1).astype(np.float64)
574         w3 = np.sum(event_changes, axis=1).astype(np.float64)
575         w4 = np.divide(0.2 * w1 + 0.8 * w2, w3, out=np.zeros_like(w3, dtype=np.
576         float64), where=w3 != 0)
577         return w4
578
579     def preprocess_data(self, data):

```

```

577     """
578     Data preprocessing, including generating features A, B, C, and All
579     """
580     year_mapping = pd.read_csv("year_transmission_reference.csv")
581     year_dict = dict(zip(year_mapping['evaluating_year'], year_mapping['
use_data_from'])))
582
583     features_list = []
584     for idx, row in data.iterrows():
585         current_year = row['Year']
586         reference_year = year_dict.get(current_year, current_year)
587         reference_data = data[data['Year'] == reference_year]
588
589         if len(reference_data) > 0:
590             ref_row = reference_data[reference_data['NOC_x'] == row['NOC_x']]
591             ref_row = ref_row.iloc[0] if len(ref_row) > 0 else row
592         else:
593             ref_row = row
594
595         score_rate_list = np.array(eval(ref_row['Score_Rate_List']))
596         score_list = np.array(eval(ref_row['Score_List']), dtype=np.float64)
597         participants_list = np.array(eval(ref_row['Participants_List']),
dtype=np.float64)
598
599         v4_ = np.sum(score_rate_list)
600         v5_ = np.sum(participants_list)
601         v8_ = np.sum(np.divide(score_list, participants_list, out=np.
zeros_like(score_list, dtype=np.float64), where=participants_list > 0))
602
603         country_change = np.array(eval(ref_row['CountryChange']))
604         athlete_change = np.array(eval(ref_row['AthleteChange']))
605         event_changes = np.array(eval(ref_row['Event_Changes']))
606         w4_ = self.calculate_w4_vector(country_change.reshape(1, -1),
athlete_change.reshape(1, -1), event_changes.reshape(1, -1))[0]
607
608         v8_w4 = np.sum(np.multiply(np.divide(score_list, participants_list,
out=np.zeros_like(score_list, dtype=np.float64), where=participants_list > 0)
, w4_))
609
610         v1 = ref_row['if_host']
611         v6 = ref_row['Gold_prev'] if pd.notna(ref_row['Gold_prev']) else 0
612         v7 = ref_row['distance']
613         v3_ = ref_row['Score']
614         v3_w4 = np.sum(np.multiply(score_list, w4_))
615

```

```

616         w3_ = np.sum(event_changes)
617
618         features = {
619             'features_a': [v5_, v8_, v8_w4],
620             'features_b': [v1, v5_, v6, v7, v3_, v3_w4, v8_],
621             'features_c': [v1, v5_, v6, v7, v3_, v3_w4, v8_],
622             'features_all': [v1, v3_, v4_, v5_, v6, v7, v8_, w3_, w4_]
623         }
624         features_list.append(features)
625
626         features_a = np.array([f['features_a'] for f in features_list])
627         features_b = np.array([f['features_b'] for f in features_list])
628         features_c = np.array([f['features_c'] for f in features_list])
629         features_all = np.array([f['features_all'] for f in features_list])
630
631         labels = (data['Total_Medals'] > 0).astype(int).values
632         medal_expectation = data['Total_Medals'].values
633         gold_expectation = data['Gold'].values
634
635         scaler = StandardScaler()
636         features_a_scaled = scaler.fit_transform(features_a)
637         features_b_scaled = scaler.fit_transform(features_b)
638         features_c_scaled = scaler.fit_transform(features_c)
639         features_all_scaled = scaler.fit_transform(features_all)
640
641         return features_a_scaled, features_b_scaled, features_c_scaled,
642         features_all_scaled, labels, medal_expectation, gold_expectation
643
644     def train_classifier(self, features_a, features_all, labels, data):
645         """
646         Train the classifier to predict whether medals will be won
647         """
648         years = data['Year'].values
649         nocs = data['NOC_x'].values
650
651         is_small_countries = np.array([self.Country_classification(year, noc,
652         data)
653                                     for year, noc in zip(years, nocs)])
654
655         final_features = np.zeros((len(labels), features_all.shape[1]))
656         small_countries_mask = is_small_countries
657
658         final_features[small_countries_mask] = (1 - self.lambda_weight) *
659         features_all[small_countries_mask]
660         final_features[small_countries_mask, :features_a.shape[1]] += self.

```

```

lambda_weight * features_a[small_countries_mask]
658
        final_features[~small_countries_mask] = features_all[~
659 small_countries_mask]
660
        self.classifier.fit(final_features, labels)
661
        self.classifier_output = self.classifier.predict_proba(final_features)
662
663
664     def train_regressor1(self, features_b, medal_expectation):
665         """
666         Train regressor 1 to predict medal expectations
667         """
668         classifier_probabilities = self.classifier_output[:, 1]
669         input_features = np.hstack([classifier_probabilities.reshape(-1, 1),
features_b])
670         self.regressor1.fit(input_features, medal_expectation)
671
672     def train_regressor2(self, features_c, gold_expectation):
673         """
674         Train regressor 2 to predict gold medal expectations
675         """
676         classifier_probabilities = self.classifier_output[:, 1]
677         input_features_reg1 = np.hstack([classifier_probabilities.reshape(-1, 1),
features_c])
678         medal_expectation = self.regressor1.predict(input_features_reg1)
679         input_features = np.hstack([
680             (1 - self.mu_weight) * medal_expectation.reshape(-1, 1),
681             self.mu_weight * features_c
682         ])
683         self.regressor2.fit(input_features, gold_expectation)
684
685     def create_lstm_attention_model(self, input_shape):
686         """
687         Create an LSTM-based attention model for sequential prediction
688         """
689         inputs = Input(shape=(input_shape[1], input_shape[2]))
690         lstm_out, state_h, state_c = LSTM(64, return_sequences=True, return_state
=True)(inputs)
691         attention = Dot(axes=[2, 2])([lstm_out, lstm_out])
692         attention = Activation('softmax')(attention)
693         context = Dot(axes=[2, 1])([attention, lstm_out])
694         combined = Concatenate()([context, lstm_out])
695         outputs = Dense(1, activation='linear')(combined)
696
697         optimizer = Adam(learning_rate=self.learning_rate)

```

```
698     model = Model(inputs, outputs)
699     model.compile(optimizer=optimizer, loss='mean_squared_error')
700
701     return model
702
703 # version 3.0
704 import numpy as np
705 import pandas as pd
706 from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
707 from sklearn.preprocessing import StandardScaler
708 from keras.models import Model
709 from keras.layers import Input, LSTM, Dense, Dot, Activation, Concatenate
710 from keras.optimizers import Adam
711 from keras.callbacks import ReduceLROnPlateau
712
713 class OlympicPredictionModel:
714     from sklearn.neural_network import MLPRegressor
715
716     def __init__(self, lambda_weight=0.3, mu_weight=0.85, learning_rate=0.001):
717         self.lambda_weight = lambda_weight
718         self.mu_weight = mu_weight
719         self.learning_rate = learning_rate # Added for Adam optimizer tuning
720         self.classifier = RandomForestClassifier(n_estimators=200)
721         # Change RandomForestRegressor to MLPRegressor
722         self.regressor1 = self.MLPRegressor(hidden_layer_sizes=(64, 32), max_iter
=500, random_state=42)
723         self.regressor2 = self.MLPRegressor(hidden_layer_sizes=(64, 32), max_iter
=500, random_state=42)
724         self.lstm_model = None
725
726     @staticmethod
727     def Country_classification(year, Noc_x, data):
728         historical_data = data[data['Year'] <= year]
729         country_total = historical_data[historical_data['NOC_x'] == Noc_x]['
Name_Count'].sum()
730         all_countries = historical_data.groupby('NOC_x')['Name_Count'].sum()
731         threshold = all_countries.quantile(0.2)
732         return country_total <= threshold
733
734     def calculate_w4_vector(self, country_change, athlete_change, event_changes):
735         w1 = np.sum(country_change, axis=1).astype(np.float64)
736         w2 = np.sum(athlete_change, axis=1).astype(np.float64)
737         w3 = np.sum(event_changes, axis=1).astype(np.float64)
738         w4 = np.divide(0.2 * w1 + 0.8 * w2, w3, out=np.zeros_like(w3, dtype=np.
float64), where=w3 != 0)
```

```

739     return w4
740
741     def preprocess_data(self, data):
742         year_mapping = pd.read_csv("year_transmission_reference.csv")
743         year_dict = dict(zip(year_mapping['evaluating_year'], year_mapping['
use_data_from']))
744
745         features_list = []
746         for idx, row in data.iterrows():
747             current_year = row['Year']
748             reference_year = year_dict.get(current_year, current_year)
749             reference_data = data[data['Year'] == reference_year]
750
751             if len(reference_data) > 0:
752                 ref_row = reference_data[reference_data['NOC_x'] == row['NOC_x']]
753                 ref_row = ref_row.iloc[0] if len(ref_row) > 0 else row
754             else:
755                 ref_row = row
756
757             score_rate_list = np.array(eval(ref_row['Score_Rate_List']))
758             score_list = np.array(eval(ref_row['Score_List']), dtype=np.float64)
759             participants_list = np.array(eval(ref_row['Participants_List']),
dtype=np.float64)
760
761             v4_ = np.sum(score_rate_list)
762             v5_ = np.sum(participants_list)
763             v8_ = np.sum(np.divide(score_list, participants_list, out=np.
zeros_like(score_list, dtype=np.float64), where=participants_list > 0))
764
765             country_change = np.array(eval(ref_row['CountryChange']))
766             athlete_change = np.array(eval(ref_row['AthleteChange']))
767             event_changes = np.array(eval(ref_row['Event_Changes']))
768             w4_ = self.calculate_w4_vector(country_change.reshape(1, -1),
athlete_change.reshape(1, -1), event_changes.reshape(1, -1))[0]
769
770             v8_w4 = np.sum(np.multiply(np.divide(score_list, participants_list,
out=np.zeros_like(score_list, dtype=np.float64), where=participants_list > 0)
, w4_))
771
772             v1 = ref_row['if_host']
773             v6 = ref_row['Gold_prev'] if pd.notna(ref_row['Gold_prev']) else 0
774             v7 = ref_row['distance']
775             v3_ = ref_row['Score']
776             v3_w4 = np.sum(np.multiply(score_list, w4_))
777

```



```
778     w3_ = np.sum(event_changes)
779
780     features = {
781         'features_a': [v5_, v8_, v8_w4],
782         'features_b': [v1, v5_, v6, v7, v3_, v3_w4, v8_],
783         'features_c': [v1, v5_, v6, v7, v3_, v3_w4, v8_],
784         'features_all': [v1, v3_, v4_, v5_, v6, v7, v8_, w3_, w4_]
785     }
786     features_list.append(features)
787
788     features_a = np.array([f['features_a'] for f in features_list])
789     features_b = np.array([f['features_b'] for f in features_list])
790     features_c = np.array([f['features_c'] for f in features_list])
791     features_all = np.array([f['features_all'] for f in features_list])
792
793     labels = (data['Total_Medals'] > 0).astype(int).values
794     medal_expectation = data['Total_Medals'].values
795     gold_expectation = data['Gold'].values
796
797     scaler = StandardScaler()
798     features_a_scaled = scaler.fit_transform(features_a)
799     features_b_scaled = scaler.fit_transform(features_b)
800     features_c_scaled = scaler.fit_transform(features_c)
801     features_all_scaled = scaler.fit_transform(features_all)
802
803     return features_a_scaled, features_b_scaled, features_c_scaled,
features_all_scaled, labels, medal_expectation, gold_expectation
804
805     def train_classifier(self, features_a, features_all, labels, data):
806         years = data['Year'].values
807         nocs = data['NOC_x'].values
808
809         is_small_countries = np.array([self.Country_classification(year, noc,
data)
810                                         for year, noc in zip(years, nocs)])
811
812         final_features = np.zeros((len(labels), features_all.shape[1]))
813         small_countries_mask = is_small_countries
814
815         final_features[small_countries_mask] = (1 - self.lambda_weight) *
features_all[small_countries_mask]
816         final_features[small_countries_mask, :features_a.shape[1]] += self.
lambda_weight * features_a[small_countries_mask]
817
818         final_features[~small_countries_mask] = features_all[~
```

```
small_countries_mask]

819
820     self.classifier.fit(final_features, labels)
821     self.classifier_output = self.classifier.predict_proba(final_features)
822
823     def train_regressor1(self, features_b, medal_expectation):
824         classifier_probabilities = self.classifier_output[:, 1]
825         input_features = np.hstack([classifier_probabilities.reshape(-1, 1),
features_b])
826         self.regressor1.fit(input_features, medal_expectation)
827
828     def train_regressor2(self, features_c, gold_expectation):
829         classifier_probabilities = self.classifier_output[:, 1]
830         input_features_reg1 = np.hstack([classifier_probabilities.reshape(-1, 1),
features_c])
831         medal_expectation = self.regressor1.predict(input_features_reg1)
832         input_features = np.hstack([
833             (1 - self.mu_weight) * medal_expectation.reshape(-1, 1),
834             self.mu_weight * features_c
835         ])
836         self.regressor2.fit(input_features, gold_expectation)
837
838     def create_lstm_attention_model(self, input_shape):
839         inputs = Input(shape=(input_shape[1], input_shape[2]))
840         lstm_out, state_h, state_c = LSTM(64, return_sequences=True, return_state
=True)(inputs)
841         attention = Dot(axes=[2, 2])([lstm_out, lstm_out])
842         attention = Activation('softmax')(attention)
843         context = Dot(axes=[2, 1])([attention, lstm_out])
844         combined = Concatenate()([context, lstm_out])
845         outputs = Dense(1, activation='linear')(combined)
846         optimizer = Adam(learning_rate=self.learning_rate)
847         model = Model(inputs, outputs)
848         model.compile(optimizer=optimizer, loss='mean_squared_error')
849         return model
850
851     def evaluate_model(self, features_a, features_b, features_c, features_all,
labels, medal_expectation, gold_expectation, data, cv=5):
852         from sklearn.model_selection import KFold
853         from sklearn.metrics import accuracy_score, roc_auc_score, r2_score,
mean_squared_error
854
855         kf = KFold(n_splits=cv, shuffle=True, random_state=42)
856
857         classifier_scores = {'train_acc': [], 'test_acc': [], 'train_auc': [], 'test_auc': [], 'train_mse': [], 'test_mse': []}
```

```

test_auc': []}
858     regressor1_scores = {'train_r2': [], 'test_r2': [], 'train_mse': [], '
test_mse': []}
859     regressor2_scores = {'train_r2': [], 'test_r2': [], 'train_mse': [], '
test_mse': []}
860
861     for train_idx, test_idx in kf.split(features_all):
862         X_train_a = features_a[train_idx]
863         X_test_a = features_a[test_idx]
864         X_train_b = features_b[train_idx]
865         X_test_b = features_b[test_idx]
866         X_train_c = features_c[train_idx]
867         X_test_c = features_c[test_idx]
868         X_train_all = features_all[train_idx]
869         X_test_all = features_all[test_idx]
870         y_train = labels[train_idx]
871         y_test = labels[test_idx]
872         medal_train = medal_expectation[train_idx]
873         medal_test = medal_expectation[test_idx]
874         gold_train = gold_expectation[train_idx]
875         gold_test = gold_expectation[test_idx]
876
877         train_data = data.iloc[train_idx].reset_index(drop=True)
878         self.train_classifier(X_train_a, X_train_all, y_train, train_data)
879         train_prob = self.classifier.predict_proba(X_train_all)[: , 1]
880         test_prob = self.classifier.predict_proba(X_test_all)[: , 1]
881
882         classifier_scores['train_acc'].append(accuracy_score(y_train,
train_prob > 0.5))
883         classifier_scores['test_acc'].append(accuracy_score(y_test, test_prob
> 0.5))
884         classifier_scores['train_auc'].append(roc_auc_score(y_train,
train_prob))
885         classifier_scores['test_auc'].append(roc_auc_score(y_test, test_prob)
)
886
887         self.train_regressor1(X_train_b, medal_train)
888         train_medal_pred = self.regressor1.predict(np.hstack([train_prob.
reshape(-1, 1), X_train_b]))
889         test_medal_pred = self.regressor1.predict(np.hstack([test_prob.
reshape(-1, 1), X_test_b]))
890         regressor1_scores['train_r2'].append(r2_score(medal_train,
train_medal_pred))
891         regressor1_scores['test_r2'].append(r2_score(medal_test,
test_medal_pred))

```

```
892         regressor1_scores['train_mse'].append(mean_squared_error(medal_train,
144         train_medal_pred))
893         regressor1_scores['test_mse'].append(mean_squared_error(medal_test,
144         test_medal_pred))
894
895         self.train_regressor2(X_train_c, gold_train)
896         train_gold_pred = self.regressor2.predict(np.hstack([
897             (1 - self.mu_weight) * train_medal_pred.reshape(-1, 1),
898             self.mu_weight * X_train_c
899         ]))
900         test_gold_pred = self.regressor2.predict(np.hstack([
901             (1 - self.mu_weight) * test_medal_pred.reshape(-1, 1),
902             self.mu_weight * X_test_c
903         ]))
904
905         regressor2_scores['train_r2'].append(r2_score(gold_train,
144         train_gold_pred))
906         regressor2_scores['test_r2'].append(r2_score(gold_test,
144         test_gold_pred))
907         regressor2_scores['train_mse'].append(mean_squared_error(gold_train,
144         train_gold_pred))
908         regressor2_scores['test_mse'].append(mean_squared_error(gold_test,
144         test_gold_pred))
909
910
911
912     def train_and_predict_period(self, data, start_year, end_year):
913         olympic_years = [1896, 1900, 1904, 1908, 1912, 1920, 1924, 1928, 1932,
144         1936, 1948, 1952, 1956, 1960, 1964, 1968, 1972, 1976, 1980, 1984, 1988, 1992,
144         1996, 2000, 2004, 2008, 2012, 2016, 2020]
914         period_data = data[(data['Year'] >= start_year) & (data['Year'] <=
144         end_year)].reset_index(drop=True)
915
916         features_a, features_b, features_c, features_all, labels,
144         medal_expectation, gold_expectation = self.preprocess_data(period_data)
917
918         self.train_classifier(features_a, features_all, labels, period_data)
919         self.train_regressor1(features_b, medal_expectation)
920         self.train_regressor2(features_c, gold_expectation)
921
922         olympic_mask = period_data['Year'].isin(olympic_years)
923         olympic_data = period_data[olympic_mask].reset_index(drop=True)
924         olympic_features_all = features_all[olympic_mask]
925         time_series_data = olympic_features_all.reshape((olympic_features_all.
144         shape[0], 1, olympic_features_all.shape[1]))
```

```

926         if self.lstm_model is None:
927             self.lstm_model = self.create_lstm_attention_model(time_series_data.
928 shape)
929
930         lr_scheduler = ReduceLROnPlateau(monitor='loss', factor=0.5, patience=3,
931 min_lr=1e-6, verbose=1)
932         self.lstm_model.fit(time_series_data, gold_expectation[olympic_mask],
933 epochs=10, batch_size=32, verbose=1, callbacks=[lr_scheduler])
934
935         last_year_data = period_data[period_data['Year'] == end_year].reset_index
936 (drop=True)
937         end_year_olympic_mask = olympic_data['Year'] == end_year
938         time_series_last_year = time_series_data[end_year_olympic_mask]
939
940         if len(time_series_last_year) == 0:
941             most_recent_year = olympic_data['Year'].max()
942             time_series_last_year = time_series_data[olympic_data['Year'] ==
943 most_recent_year]
944
945         if len(time_series_last_year.shape) == 2:
946             time_series_last_year = np.expand_dims(time_series_last_year, axis=0)
947
948         lstm_gold_prediction = self.lstm_model.predict(time_series_last_year,
949 verbose=0)
950         available_years = period_data['Year'].unique()
951         prediction_year = end_year if end_year in available_years else
952 available_years.max()
953
954         classifier_probabilities = self.classifier_output[:, 1][period_data['Year
955 ']] == prediction_year]
956         medal_prediction = self.regressor1.predict(np.hstack([
957             classifier_probabilities.reshape(-1, 1),
958             features_b[period_data['Year'] == prediction_year]
959 ]))
960
961         gold_prediction_rf = self.regressor2.predict(np.hstack([
962             (1 - self.mu_weight) * medal_prediction.reshape(-1, 1),
963             self.mu_weight * features_c[period_data['Year'] == prediction_year]
964 ]))
965
966         prediction_data = period_data[period_data['Year'] == prediction_year].
967 reset_index(drop=True)
968         results = pd.DataFrame({
969             'NOC': prediction_data['NOC_x'],

```

```
962         'Year': prediction_year,
963         'Will_Win_Medal': classifier_probabilities > 0.5,
964         'Medal_Prediction': medal_prediction,
965         'Gold_Prediction': gold_prediction_rf
966     })
967     self.evaluate_model(features_a, features_b, features_c, features_all,
968                        labels, medal_expectation, gold_expectation,
969                        period_data)
970     return results
971
972 # Example usage
973 data = pd.read_csv("feature_Noc.csv") # Replace with your actual dataset
974 model = OlympicPredictionModel()
975 start_year = 1964
976 end_year = 2028
977 results = model.train_and_predict_period(data, start_year, end_year)
978 print(results)
979 # Save the results to a CSV file
980 results.to_csv("results.csv", index=False) # Replace with your desired file path
981
982 import pandas as pd
983 import numpy as np
984 import matplotlib.pyplot as plt
985 from sklearn.metrics import mean_absolute_error
986
987 # Load noisy data
988 noisy_data_path = 'noisy_df_0.05.csv'
989 noisy_data = pd.read_csv(noisy_data_path)
990
991 # Load the true (noise-free) data (ensure the correct file path is provided)
992 true_data_path = 'feature_Noc.csv' # Replace with your actual true data path
993 true_data = pd.read_csv(true_data_path)
994
995 # Initialize the model
996 model = OlympicPredictionModel()
997
998 # Set the testing time range
999 start_year = 1964
1000 end_year = 2028
1001
1002 # Train and predict on the true data
1003 results_true = model.train_and_predict_period(true_data, start_year, end_year)
1004
1005 # Train and predict on the noisy data
```

```
1006 results_noisy = model.train_and_predict_period(noisy_data, start_year, end_year)
1007
1008 # Extract gold medal prediction results
1009 true_gold_predictions = results_true["Gold_Prediction"].values
1010 noisy_gold_predictions = results_noisy["Gold_Prediction"].values
1011
1012 # Calculate robustness metrics (e.g., Mean Absolute Error)
1013 mae_gold = mean_absolute_error(true_gold_predictions, noisy_gold_predictions)
1014
1015 # Plot the prediction results comparison
1016 plt.figure(figsize=(12, 6))
1017 plt.plot(true_gold_predictions, label="True Gold Predictions", marker="o")
1018 plt.plot(noisy_gold_predictions, label="Noisy Gold Predictions", marker="x",
1019         linestyle="--")
1019 plt.title(f"Robustness Analysis: Gold Medal Predictions (noise:0.05 MAE: {
1020         mae_gold:.2f})")
1020 plt.xlabel("Index")
1021 plt.ylabel("Gold Predictions")
1022 plt.legend()
1023 plt.grid(True)
1024
1025 # Save the plot
1026 output_image_path = "robustness_analysis_0.05.png"
1027 plt.savefig(output_image_path)
1028 plt.show()
1029
1030 # Print the output file path
1031 print(f"Robustness analysis image saved at: {output_image_path}")
1032
1033 import pandas as pd
1034 import numpy as np
1035 import matplotlib.pyplot as plt
1036 from sklearn.metrics import mean_absolute_error
1037
1038 # Load noisy data
1039 noisy_data_path = 'noisy_df_0.1.csv'
1040 noisy_data = pd.read_csv(noisy_data_path)
1041
1042 # Load the true (noise-free) data (ensure the correct file path is provided)
1043 true_data_path = 'feature_Noc.csv' # Replace with your actual true data path
1044 true_data = pd.read_csv(true_data_path)
1045
1046 # Initialize the model
1047 model = OlympicPredictionModel()
1048
```

```
1049 # Set the testing time range
1050 start_year = 1964
1051 end_year = 2028
1052
1053 # Train and predict on the true data
1054 results_true = model.train_and_predict_period(true_data, start_year, end_year)
1055
1056 # Train and predict on the noisy data
1057 results_noisy = model.train_and_predict_period(noisy_data, start_year, end_year)
1058
1059 # Extract gold medal prediction results
1060 true_gold_predictions = results_true["Gold_Prediction"].values
1061 noisy_gold_predictions = results_noisy["Gold_Prediction"].values
1062
1063 # Calculate robustness metrics (e.g., Mean Absolute Error)
1064 mae_gold = mean_absolute_error(true_gold_predictions, noisy_gold_predictions)
1065
1066 # Plot the prediction results comparison
1067 plt.figure(figsize=(12, 6))
1068 plt.plot(true_gold_predictions, label="True Gold Predictions", marker="o")
1069 plt.plot(noisy_gold_predictions, label="Noisy Gold Predictions", marker="x",
1070         linestyle="--")
1071 plt.title(f"Robustness Analysis: Gold Medal Predictions (noise:0.1 MAE: {mae_gold:.2f})")
1072 plt.xlabel("Index")
1073 plt.ylabel("Gold Predictions")
1074 plt.legend()
1075 plt.grid(True)
1076
1077 # Save the plot
1078 output_image_path = "robustness_analysis_0.1.png"
1079 plt.savefig(output_image_path)
1080 plt.show()
1081
1082 # Print the output file path
1083 print(f"Robustness analysis image saved at: {output_image_path}")
1084
1085 import pandas as pd
1086 import numpy as np
1087 import matplotlib.pyplot as plt
1088 from sklearn.metrics import mean_absolute_error
1089
1090 # Load noisy data
1091 noisy_data_path = 'noisy_df_0.2.csv'
1092 noisy_data = pd.read_csv(noisy_data_path)
```



```
1092
1093 # Load the true (noise-free) data (ensure the correct file path is provided)
1094 true_data_path = 'feature_Noc.csv' # Replace with your actual true data path
1095 true_data = pd.read_csv(true_data_path)
1096
1097 # Initialize the model
1098 model = OlympicPredictionModel()
1099
1100 # Set the testing time range
1101 start_year = 1964
1102 end_year = 2028
1103
1104 # Train and predict on the true data
1105 results_true = model.train_and_predict_period(true_data, start_year, end_year)
1106
1107 # Train and predict on the noisy data
1108 results_noisy = model.train_and_predict_period(noisy_data, start_year, end_year)
1109
1110 # Extract gold medal prediction results
1111 true_gold_predictions = results_true["Gold_Prediction"].values
1112 noisy_gold_predictions = results_noisy["Gold_Prediction"].values
1113
1114 # Calculate robustness metrics (e.g., Mean Absolute Error)
1115 mae_gold = mean_absolute_error(true_gold_predictions, noisy_gold_predictions)
1116
1117 # Plot the prediction results comparison
1118 plt.figure(figsize=(12, 6))
1119 plt.plot(true_gold_predictions, label="True Gold Predictions", marker="o")
1120 plt.plot(noisy_gold_predictions, label="Noisy Gold Predictions", marker="x",
1121         linestyle="--")
1122 plt.title(f"Robustness Analysis: Gold Medal Predictions (noise:0.2 MAE: {mae_gold:.2f})")
1123 plt.xlabel("Index")
1124 plt.ylabel("Gold Predictions")
1125 plt.legend()
1126 plt.grid(True)
1127
1128 # Save the plot
1129 output_image_path = "robustness_analysis_0.2.png"
1130 plt.savefig(output_image_path)
1131 plt.show()
1132
1133 # Print the output file path
1134 print(f"Robustness analysis image saved at: {output_image_path}")
```

```
1135 import pandas as pd
1136 import numpy as np
1137 import matplotlib.pyplot as plt
1138 from sklearn.metrics import mean_absolute_error
1139
1140 # Load noisy data
1141 noisy_data_path = 'noisy_df_0.3.csv'
1142 noisy_data = pd.read_csv(noisy_data_path)
1143
1144 # Load the true (noise-free) data (ensure the correct file path is provided)
1145 true_data_path = 'feature_Noc.csv' # Replace with your actual true data path
1146 true_data = pd.read_csv(true_data_path)
1147
1148 # Initialize the model
1149 model = OlympicPredictionModel()
1150
1151 # Set the testing time range
1152 start_year = 1964
1153 end_year = 2028
1154
1155 # Train and predict on the true data
1156 results_true = model.train_and_predict_period(true_data, start_year, end_year)
1157
1158 # Train and predict on the noisy data
1159 results_noisy = model.train_and_predict_period(noisy_data, start_year, end_year)
1160
1161 # Extract gold medal prediction results
1162 true_gold_predictions = results_true["Gold_Prediction"].values
1163 noisy_gold_predictions = results_noisy["Gold_Prediction"].values
1164
1165 # Calculate robustness metrics (e.g., Mean Absolute Error)
1166 mae_gold = mean_absolute_error(true_gold_predictions, noisy_gold_predictions)
1167
1168 # Plot the prediction results comparison
1169 plt.figure(figsize=(12, 6))
1170 plt.plot(true_gold_predictions, label="True Gold Predictions", marker="o")
1171 plt.plot(noisy_gold_predictions, label="Noisy Gold Predictions", marker="x",
1172         linestyle="--")
1173 plt.title(f"Robustness Analysis: Gold Medal Predictions (noise:0.3 MAE: {mae_gold:.2f})")
1174 plt.xlabel("Index")
1175 plt.ylabel("Gold Predictions")
1176 plt.legend()
1177 plt.grid(True)
```

```
1178 # Save the plot
1179 output_image_path = "robustness_analysis_0.3.png"
1180 plt.savefig(output_image_path)
1181 plt.show()
1182
1183 # Print the output file path
1184 print(f"Robustness analysis image saved at: {output_image_path}")
1185
1186 class OlympicPredictionModel:
1187     from sklearn.neural_network import MLPRegressor
1188
1189     def __init__(self, lambda_weight=0.3, mu_weight=0.85, learning_rate=0.001):
1190         self.lambda_weight = lambda_weight
1191         self.mu_weight = mu_weight
1192         self.learning_rate = learning_rate
1193         self.classifier = RandomForestClassifier(n_estimators=200)
1194         self.regressor1 = self.MLPRegressor(hidden_layer_sizes=(64, 32), max_iter
1195 =500, random_state=42)
1196         self.regressor2 = self.MLPRegressor(hidden_layer_sizes=(64, 32), max_iter
1197 =500, random_state=42)
1198         self.lstm_model = None
1199
1200     @staticmethod
1201     def Country_classification(year, Noc_x, data):
1202         historical_data = data[data['Year'] <= year]
1203         country_total = historical_data[historical_data['NOC_x'] == Noc_x]['
1204 Name_Count'].sum()
1205         all_countries = historical_data.groupby('NOC_x')['Name_Count'].sum()
1206         threshold = all_countries.quantile(0.2)
1207         return country_total <= threshold
1208
1209     def calculate_w4_vector(self, country_change, athlete_change, event_changes):
1210         w1 = np.sum(country_change, axis=1).astype(np.float64)
1211         w2 = np.sum(athlete_change, axis=1).astype(np.float64)
1212         w3 = np.sum(event_changes, axis=1).astype(np.float64)
1213         w4 = np.divide(0.2 * w1 + 0.8 * w2, w3, out=np.zeros_like(w3, dtype=np.
1214 float64), where=w3 != 0)
1215         return w4
1216
1217     def preprocess_data(self, data):
1218         year_mapping = pd.read_csv("year_transmission_reference.csv")
1219         year_dict = dict(zip(year_mapping['evaluating_year'], year_mapping['
1220 use_data_from']))
1221
1222         features_list = []
```

```

1218     for idx, row in data.iterrows():
1219         current_year = row['Year']
1220         reference_year = year_dict.get(current_year, current_year)
1221         reference_data = data[data['Year'] == reference_year]
1222
1223         if len(reference_data) > 0:
1224             ref_row = reference_data[reference_data['NOC_x'] == row['NOC_x']]
1225             ref_row = ref_row.iloc[0] if len(ref_row) > 0 else row
1226         else:
1227             ref_row = row
1228
1229         score_rate_list = np.array(eval(ref_row['Score_Rate_List']))
1230         score_list = np.array(eval(ref_row['Score_List']), dtype=np.float64)
1231         participants_list = np.array(eval(ref_row['Participants_List']),
dtype=np.float64)
1232
1233         v4_ = np.sum(score_rate_list)
1234         v5_ = np.sum(participants_list)
1235         v8_ = np.sum(np.divide(score_list, participants_list, out=np.
zeros_like(score_list, dtype=np.float64), where=participants_list > 0))
1236
1237         country_change = np.array(eval(ref_row['CountryChange']))
1238         athlete_change = np.array(eval(ref_row['AthleteChange']))
1239         event_changes = np.array(eval(ref_row['Event_Changes']))
1240         w4_ = self.calculate_w4_vector(country_change.reshape(1, -1),
athlete_change.reshape(1, -1), event_changes.reshape(1, -1))[0]
1241
1242         v8_w4 = np.sum(np.multiply(np.divide(score_list, participants_list,
out=np.zeros_like(score_list, dtype=np.float64), where=participants_list > 0)
, w4_))
1243
1244         v1 = ref_row['if_host']
1245         v6 = ref_row['Gold_prev'] if pd.notna(ref_row['Gold_prev']) else 0
1246         v7 = ref_row['distance']
1247         v3_ = ref_row['Score']
1248         v3_w4 = np.sum(np.multiply(score_list, w4_))
1249
1250         w3_ = np.sum(event_changes)
1251
1252         features = {
1253             'features_a': [v5_, v8_, v8_w4],
1254             'features_b': [v1, v5_, v6, v7, v3_, v3_w4, v8_],
1255             'features_c': [v1, v5_, v6, v7, v3_, v3_w4, v8_],
1256             'features_all': [v1, v3_, v4_, v5_, v6, v7, v8_, w3_, w4_]
1257         }

```

```

1258         features_list.append(features)
1259
1260         features_a = np.array([f['features_a'] for f in features_list])
1261         features_b = np.array([f['features_b'] for f in features_list])
1262         features_c = np.array([f['features_c'] for f in features_list])
1263         features_all = np.array([f['features_all'] for f in features_list])
1264
1265         labels = (data['Total_Medals'] > 0).astype(int).values
1266         medal_expectation = data['Total_Medals'].values
1267         gold_expectation = data['Gold'].values
1268
1269         scaler = StandardScaler()
1270         features_a_scaled = scaler.fit_transform(features_a)
1271         features_b_scaled = scaler.fit_transform(features_b)
1272         features_c_scaled = scaler.fit_transform(features_c)
1273         features_all_scaled = scaler.fit_transform(features_all)
1274
1275         return features_a_scaled, features_b_scaled, features_c_scaled,
1276         features_all_scaled, labels, medal_expectation, gold_expectation
1277
1278     def train_classifier(self, features_a, features_all, labels, data):
1279         years = data['Year'].values
1280         nocs = data['NOC_x'].values
1281
1282         is_small_countries = np.array([self.Country_classification(year, noc,
1283         data)
1284
1285                                     for year, noc in zip(years, nocs)])
1286
1287         final_features = np.zeros((len(labels), features_all.shape[1]))
1288         small_countries_mask = is_small_countries
1289
1290         final_features[small_countries_mask] = (1 - self.lambda_weight) *
1291         features_all[small_countries_mask]
1292         final_features[small_countries_mask, :features_a.shape[1]] += self.
1293         lambda_weight * features_a[small_countries_mask]
1294
1295         final_features[~small_countries_mask] = features_all[~
1296         small_countries_mask]
1297
1298         self.classifier.fit(final_features, labels)
1299         self.classifier_output = self.classifier.predict_proba(final_features)
1300
1301     def train_regressor1(self, features_b, medal_expectation):
1302         classifier_probabilities = self.classifier_output[:, 1]
1303         input_features = np.hstack([classifier_probabilities.reshape(-1, 1),

```

```

features_b])
1298     self.regressor1.fit(input_features, medal_expectation)
1299
1300     def train_regressor2(self, features_c, gold_expectation):
1301         """
1302         Train the second regressor (for gold medal predictions) and compute
1303         training accuracy.
1304         """
1305         classifier_probabilities = self.classifier_output[:, 1]
1306         input_features_reg1 = np.hstack([classifier_probabilities.reshape(-1, 1),
1307         features_c])
1308         medal_expectation = self.regressor1.predict(input_features_reg1)
1309
1310         input_features = np.hstack([
1311             (1 - self.mu_weight) * medal_expectation.reshape(-1, 1),
1312             self.mu_weight * features_c
1313         ])
1314         self.regressor2.fit(input_features, gold_expectation)
1315
1316         # Calculate gold prediction accuracy on the training set
1317         train_gold_pred = self.regressor2.predict(input_features)
1318         gold_accuracy = np.mean(np.abs(train_gold_pred / 2 - gold_expectation) <
1319         2)
1320
1321         print(f"Training Gold Prediction Accuracy: {gold_accuracy:.2f}")
1322
1323     # Example usage
1324     data = pd.read_csv("feature_Noc.csv") # Replace with your dataset
1325     model = OlympicPredictionModel()
1326
1327     # Preprocess data
1328     features_a, features_b, features_c, features_all, labels, medal_expectation,
1329     gold_expectation = model.preprocess_data(data)
1330
1331     # Train the classifier
1332     model.train_classifier(features_a, features_all, labels, data)
1333
1334     # Train the first regressor
1335     model.train_regressor1(features_b, medal_expectation)
1336
1337     # Train the second regressor and output training accuracy
1338     model.train_regressor2(features_c, gold_expectation)

```

Listing 15: model.ipynb

```

1 import pandas as pd

```

```
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from sklearn.metrics import mean_absolute_error
5
6 # Sensitivity Analysis Function
7 def sensitivity_analysis(model, data, feature_columns, perturbation_percent):
8     """
9     Perform sensitivity analysis by perturbing specific feature columns and
10     observing prediction changes.
11
12     Parameters:
13     - model: OlympicPredictionModel instance.
14     - data: Original dataset (Pandas DataFrame).
15     - feature_columns: List of columns to perturb.
16     - perturbation_percent: Percentage of perturbation to apply.
17
18     Returns:
19     - sensitivity_results: DataFrame comparing original vs perturbed predictions.
20     - mae_sensitivity: Mean Absolute Error of predictions due to perturbation.
21     """
22     # Deep copy the data to avoid altering the original dataset
23     perturbed_data = data.copy()
24
25     # Apply perturbation to the specified columns
26     for col in feature_columns:
27         if col not in perturbed_data.columns:
28             print(f"Warning: Column '{col}' not found in dataset. Skipping...")
29             continue
30         perturbation = perturbed_data[col] * (perturbation_percent / 100.0)
31         perturbed_data[col] += perturbation
32
33     # Preprocess original and perturbed data
34     print("Preprocessing original data...")
35     features_a, features_b, features_c, features_all, labels, medal_expectation,
36     gold_expectation = model.preprocess_data(data)
37
38     print("Preprocessing perturbed data...")
39     features_a_pert, features_b_pert, features_c_pert, features_all_pert, _, _, _
40     = model.preprocess_data(perturbed_data)
41
42     # Train and predict using original data
43     print("Training on original data...")
44     model.train_classifier(features_a, features_all, labels, data)
45     classifier_probabilities = model.classifier_output[:, 1]
46     model.train_regressor1(features_b, medal_expectation)
```

```
44 medal_predictions_original = model.regressor1.predict(np.hstack([
classifier_probabilities.reshape(-1, 1), features_b]))
45 model.train_regressor2(features_c, gold_expectation)
46 gold_predictions_original = model.regressor2.predict(np.hstack([
47     (1 - model.mu_weight) * medal_predictions_original.reshape(-1, 1),
48     model.mu_weight * features_c
49 ]))
50
51 # Train and predict using perturbed data
52 print("Training on perturbed data...")
53 model.train_classifier(features_a_pert, features_all_pert, labels,
perturbed_data)
54 classifier_probabilities_pert = model.classifier_output[:, 1]
55 model.train_regressor1(features_b_pert, medal_expectation)
56 medal_predictions_perturbed = model.regressor1.predict(np.hstack([
classifier_probabilities_pert.reshape(-1, 1), features_b_pert]))
57 model.train_regressor2(features_c_pert, gold_expectation)
58 gold_predictions_perturbed = model.regressor2.predict(np.hstack([
59     (1 - model.mu_weight) * medal_predictions_perturbed.reshape(-1, 1),
60     model.mu_weight * features_c_pert
61 ]))
62
63 # Compare predictions
64 mae_sensitivity = mean_absolute_error(gold_predictions_original,
gold_predictions_perturbed)
65
66 # Create results DataFrame for comparison
67 sensitivity_results = pd.DataFrame({
68     "Original_Gold_Predictions": gold_predictions_original,
69     "Perturbed_Gold_Predictions": gold_predictions_perturbed
70 })
71
72 return sensitivity_results, mae_sensitivity
73
74 # Load true (clean) data
75 true_data_path = "feature_Noc.csv" # Replace with your actual true data path
76 true_data = pd.read_csv(true_data_path)
77
78 # Initialize the model
79 model = OlympicPredictionModel()
80
81 # Define sensitivity test parameters
82 sensitivity_features = ["Score", "Total_Medals", "Gold_prev"] # Adjust based on
actual dataset columns
83 perturbation_percent = 10 # 10% perturbation
```



```
84
85 # Perform sensitivity analysis
86 sensitivity_results, mae_sensitivity = sensitivity_analysis(
87     model, true_data, sensitivity_features, perturbation_percent
88 )
89
90 # Plot sensitivity analysis results
91 plt.figure(figsize=(12, 6))
92 plt.plot(sensitivity_results["Original_Gold_Predictions"], label="Original
    Predictions", marker="o")
93 plt.plot(sensitivity_results["Perturbed_Gold_Predictions"], label=f"Perturbed
    Predictions ({perturbation_percent}%)", marker="x", linestyle="--")
94 plt.title(f"Sensitivity Analysis: Gold Medal Predictions (MAE: {mae_sensitivity
    :.2f})")
95 plt.xlabel("Index")
96 plt.ylabel("Gold Predictions")
97 plt.legend()
98 plt.grid(True)
99
100 # Save the plot
101 sensitivity_plot_path = "sensitivity_analysis.png"
102 plt.savefig(sensitivity_plot_path)
103 plt.show()
104
105 # Print the results and save path
106 print(f"Sensitivity analysis image saved at: {sensitivity_plot_path}")
107 print(f"Mean Absolute Error due to perturbation: {mae_sensitivity:.2f}")
108 #####
109 import pandas as pd
110 import numpy as np
111 import matplotlib.pyplot as plt
112 from sklearn.metrics import mean_absolute_error
113
114 # List of noise levels
115 noise_levels = [0.01, 0.05, 0.1, 0.2, 0.3]
116
117 # Load the true (noise-free) data
118 true_data_path = 'feature_Noc.csv' # Replace with your actual true data path
119 true_data = pd.read_csv(true_data_path)
120
121 # Initialize the model
122 model = OlympicPredictionModel()
123
124 # Set the testing time range
125 start_year = 1964
```

```
126 end_year = 2028
127
128 # Train and predict on the true data
129 results_true = model.train_and_predict_period(true_data, start_year, end_year)
130 true_gold_predictions = results_true["Gold_Prediction"].values
131
132 # Initialize a plot
133 plt.figure(figsize=(12, 6))
134
135 # Loop through noise levels and plot each one
136 for noise_level in noise_levels:
137     # Load noisy data for the current noise level
138     noisy_data_path = f'noisy_df_{noise_level}.csv'
139     noisy_data = pd.read_csv(noisy_data_path)
140
141     # Train and predict on the noisy data
142     results_noisy = model.train_and_predict_period(noisy_data, start_year,
143     end_year)
144     noisy_gold_predictions = results_noisy["Gold_Prediction"].values
145
146     # Calculate robustness metrics (e.g., Mean Absolute Error)
147     mae_gold = mean_absolute_error(true_gold_predictions, noisy_gold_predictions)
148
149     # Plot the noisy data predictions
150     plt.plot(noisy_gold_predictions, label=f"Noise {noise_level} (MAE: {mae_gold
151     :.2f})", linestyle="--", marker="x")
152
153 # Plot the true gold predictions
154 plt.plot(true_gold_predictions, label="True Gold Predictions", marker="o",
155 linewidth=2)
156
157 # Add title, labels, and legend
158 plt.title("Robustness Analysis: Gold Medal Predictions")
159 plt.xlabel("Index")
160 plt.ylabel("Gold Predictions")
161 plt.legend()
162 plt.grid(True)
163
164 # Save the plot
165 output_image_path = "robustness_analysis_all_noise.png"
166 plt.savefig(output_image_path)
167 plt.show()
168
169 # Print the output file path
```

```
167 print(f"Robustness analysis image saved at: {output_image_path}")
```

Listing 16: robust\_sensitivity.ipynb

## C Report on Use of AI

### 1. Open AI ChatGPT (Nov 5, 2024 version , ChatGPT-4)

**Quary:** Modify the color options of the listing module in the TeX file to make it more harmonious and aesthetically pleasing.

**Output:** Sets the background color of the code area to a light shade of gray, with 10% gray intensity. Configures the style of comments within the code to be a medium green color, with 60% green intensity. Specifies that keywords in the code should be displayed in blue. Defines the style for line numbers to be tiny and gray, making them less prominent. Assigns a purple color to string literals in the code.

### 2. Gravity Slingshot Technology Co., Ltd. Marscode (Mar 26, 2024 version , Doubao)

**Quary:** Generate some suitable comments for code.

**Output:** Comments generated for some code files.

### 3. Open AI ChatGPT (Nov 5, 2024 version , ChatGPT-4)

**Quary:** Finding typical great coach.

**Output:** Some coach's names that meet the question's need.

### 4. Open AI ChatGPT (Nov 5, 2024 version , ChatGPT-4)

**Quary:** Provide the csv file for Prediction medal table for 2028 Los Angeles, USA summer Olympics. Translate into a latex longtable form.

**Output:**

```
1 \begin{longtable}{|c|c|c|c|c|}
2 \hline
3 \textbf{Rank} & \textbf{Nation} & \textbf{Will Win Medal} & \textbf{Gold} &
   \textbf{Total} \\
4 \hline
5 \endfirsthead
6 \hline
7 \textbf{Rank} & \textbf{Nation} & \textbf{Will Win Medal} & \textbf{Gold} &
   \textbf{Total} \\
8 \hline
9 \endhead
10 \hline
11 \endfoot
```

12	1	&	USA	&	TRUE	&	61.89362846	&	139.8773871	\\
13	2	&	CHN	&	TRUE	&	36.73785753	&	91.91655353	\\
14	3	&	GBR	&	TRUE	&	18.88867211	&	82.83034521	\\
15	4	&	JPN	&	TRUE	&	16.00417029	&	58.62541877	\\
16	5	&	AUS	&	TRUE	&	18.18273715	&	53.84301404	\\
17	6	&	FRA	&	TRUE	&	19.56936092	&	53.74770272	\\
18	7	&	GER	&	TRUE	&	13.20334256	&	47.70665467	\\
19	8	&	ITA	&	TRUE	&	12.25179948	&	37.7220809	\\
20	9	&	BRA	&	TRUE	&	4.909950393	&	36.05183567	\\
21	10	&	NED	&	TRUE	&	11.32964194	&	35.08134432	\\
22	11	&	CAN	&	TRUE	&	8.53238935	&	27.97689688	\\
23	12	&	ESP	&	TRUE	&	7.096889707	&	22.4549891	\\
24	13	&	KOR	&	TRUE	&	7.020464162	&	22.1177207	\\
25	14	&	HUN	&	TRUE	&	5.837784707	&	17.46280967	\\
26	15	&	NZL	&	TRUE	&	5.213962466	&	16.59901162	\\
27	16	&	POL	&	TRUE	&	4.103828217	&	16.28268645	\\
28	17	&	UKR	&	TRUE	&	3.681442241	&	13.13062476	\\
29	18	&	SWE	&	TRUE	&	2.877136935	&	11.11753134	\\
30	19	&	CUB	&	TRUE	&	2.887936985	&	10.75076093	\\
31	20	&	DEN	&	TRUE	&	3.695717124	&	10.27081395	\\
32	21	&	SRB	&	TRUE	&	4.11700087	&	9.995684767	\\
33	22	&	SUI	&	TRUE	&	3.246965272	&	9.982866384	\\
34	23	&	CZE	&	TRUE	&	1.68408156	&	9.21439891	\\
35	24	&	TUR	&	TRUE	&	1.532841449	&	8.944480754	\\
36	25	&	JAM	&	TRUE	&	2.460369548	&	8.248609111	\\
37	26	&	KEN	&	TRUE	&	2.500396609	&	8.239645498	\\
38	27	&	TPE	&	TRUE	&	1.502765201	&	7.892720068	\\
39	28	&	BEL	&	TRUE	&	2.162127539	&	7.405213267	\\
40	29	&	CRO	&	TRUE	&	2.499122523	&	7.247100347	\\
41	30	&	EGY	&	TRUE	&	0.983883881	&	6.982062128	\\
42	31	&	IRI	&	TRUE	&	1.242496907	&	6.778206524	\\
43	32	&	ARG	&	TRUE	&	1.763522961	&	6.545787189	\\
44	33	&	IND	&	TRUE	&	1.056551388	&	6.357541967	\\
45	34	&	NOR	&	TRUE	&	2.015433856	&	6.282549225	\\
46	35	&	UZB	&	TRUE	&	1.849029887	&	5.562196507	\\
47	36	&	ROU	&	TRUE	&	2.006474912	&	5.177925657	\\
48	37	&	MEX	&	TRUE	&	0.6942054	&	4.704491759	\\
49	38	&	COL	&	TRUE	&	1.187432662	&	4.564946305	\\
50	39	&	GEO	&	TRUE	&	1.065348417	&	4.325132407	\\
51	40	&	IRL	&	TRUE	&	1.154925268	&	4.262881708	\\
52	41	&	HKG	&	TRUE	&	1.017064066	&	4.18749859	\\
53	42	&	KAZ	&	TRUE	&	0.832928845	&	4.152492135	\\
54	43	&	RSA	&	TRUE	&	1.289180741	&	4.094492851	\\
55	44	&	BUL	&	TRUE	&	0.519369539	&	4.073859257	\\
56	45	&	AUT	&	TRUE	&	0.766650287	&	4.067462751	\\

```

57 46 & ISR & TRUE & 0.871817818 & 4.049702673 \\
58 47 & SLO & TRUE & 0.910435659 & 3.921396044 \\
59 48 & POR & TRUE & 0.999374801 & 3.828379532 \\
60 49 & GRE & TRUE & 0.946089615 & 3.617184535 \\
61 50 & TUN & TRUE & 0.242361355 & 3.558668753 \\
62 51 & ETH & TRUE & 0.782784719 & 3.394682832 \\
63 52 & UGA & TRUE & 0.764501613 & 3.38297178 \\
64 53 & AIN & TRUE & 0.530560258 & 3.205393477 \\
65 54 & DOM & TRUE & 0.479723071 & 2.87826012 \\
66 55 & SVK & TRUE & 0.652582368 & 2.86538428 \\
67 56 & AZE & TRUE & 0.658131892 & 2.864971301 \\
68 57 & ECU & TRUE & 0.65295669 & 2.834019019 \\
69 58 & PHI & TRUE & 0.503089309 & 2.808015339 \\
70 59 & INA & TRUE & 0.528760779 & 2.747042122 \\
71 60 & FIJ & TRUE & -0.003103899 & 2.605802276 \\
72 61 & MAR & TRUE & 0.536724462 & 2.562540565 \\
73 62 & NAS & TRUE & 0.467698534 & 2.324974611 \\
74 63 & PUR & TRUE & -0.046873765 & 2.136679932 \\
75 64 & JOR & TRUE & 0.19392357 & 2.003962532 \\
76 65 & MCL & TRUE & 0.455186168 & 1.935160376 \\
77 66 & MDA & TRUE & 0.356611343 & 1.911806467 \\
78 67 & KOS & TRUE & 0.304311493 & 1.867596464 \\
79 68 & KGZ & TRUE & 0.378210031 & 1.841458856 \\
80 69 & PRK & TRUE & 0.359432562 & 1.839969699 \\
81 70 & ARM & TRUE & 0.031476902 & 1.799496923 \\
82 71 & ALC & TRUE & 0.178958857 & 1.658627997 \\
83 72 & LCA & TRUE & 0.233987355 & 1.620105127 \\
84 73 & THA & TRUE & 0.336004932 & 1.528310591 \\
85 74 & QAT & TRUE & 0.293791652 & 1.523262926 \\
86 75 & CUA & TRUE & 0.237918545 & 1.315100324 \\
87 76 & CIV & TRUE & 0.275063498 & 1.250511056 \\
88 77 & EOR & FALSE & 0.017306252 & 1.059041029 \\
89 78 & DMA & TRUE & 0.541163739 & 0.903726649 \\
90 79 & BRN & TRUE & 0.15785938 & 0.890418895 \\
91 80 & LTU & TRUE & 0.014238442 & 0.807797574 \\
92 81 & CRN & TRUE & 0.030019709 & 0.710567567 \\
93 82 & PAN & TRUE & 0.446226684 & 0.632673943 \\
94 83 & TJK & TRUE & 0.141225186 & 0.616260523 \\
95 84 & ZAM & TRUE & 0.144640175 & 0.548210104 \\
96 85 & BER & TRUE & 0.038687094 & 0.534395846 \\
97 86 & NCR & FALSE & -0.118414531 & 0.513838076 \\
98 87 & GHA & FALSE & -0.011375109 & 0.503209055 \\
99 88 & BOT & TRUE & 0.084029068 & 0.471503368 \\
100 89 & GUY & FALSE & 0.041557388 & 0.420078153 \\
101 90 & OMA & FALSE & 0.061035481 & 0.37812072 \\

```

```

102 91 & IRQ & FALSE & 0.06706993 & 0.371666848 \\
103 92 & CHI & TRUE & -0.024238576 & 0.348309951 \\
104 93 & LBA & FALSE & -0.045312895 & 0.312450675 \\
105 94 & CRC & FALSE & -0.127885095 & 0.203215597 \\
106 95 & DJI & FALSE & 0.275807067 & 0.178896068 \\
107 96 & ISV & FALSE & 0.009764616 & 0.15845807 \\
108 97 & CPV & TRUE & 0.064091496 & 0.146435484 \\
109 98 & GAB & FALSE & 0.29400497 & 0.145802258 \\
110 99 & SEN & FALSE & 0.035936367 & 0.141214057 \\
111 100 & COM & FALSE & -0.164212537 & 0.064197285 \\
112 101 & CAM & FALSE & -0.034432688 & 0.061663558 \\
113 102 & SCP & TRUE & 0.05958942 & 0.058251254 \\
114 103 & ARU & FALSE & -0.178501798 & 0.046999362 \\
115 104 & UAE & FALSE & -0.023508693 & -0.021384389 \\
116 105 & MHL & FALSE & 0.01383528 & -0.076188244 \\
117 106 & KIR & FALSE & 0.008776467 & -0.078181121 \\
118 107 & VAN & FALSE & 0.000968212 & -0.114300931 \\
119 108 & CYP & FALSE & -0.032076009 & -0.116562861 \\
120 109 & PAK & FALSE & 0.033260645 & -0.163091115 \\
121 110 & PER & TRUE & -0.058386585 & -0.246575465 \\
122 111 & PAR & FALSE & -0.266623889 & -0.349789911 \\
123 112 & YEM & FALSE & -0.274518749 & -0.376302868 \\
124 \hline
125 \end{longtable}

```