

Digicamp 2025

Introduction to Sport Analytics

The Power of Data: Predicting Sports Performance and Results

Lorenzo Galli, Amedeo Zappulla

Course Introduction

This course introduces students to **sports analytics** with a focus on track and field. By the end, students will be able to analyze race data, clean and visualize performance trends, and **build simple predictive models**.



We're **Lorenzo Galli** and **Amedeo Zappulla**, second-year Computer Science students at USI.

Lorenzo comes from the University of Pisa and has been passionate about track & field for almost 15 years. He's particularly fascinated about data science and how can we use sports data to predict different scenarios, such as winning times, results, injury prevention and more. In 2024, he developed Wind Correction, an iOS app that adjusts athlete times based on wind speed.

Amedeo got into computer science through competitive programming, earning a full scholarship at USI. Last semester he worked as a teaching assistant for Computer Architecture. He enjoys handling data to work on real-world problems and we both believe that teaching is one of the best ways to learn.

During the first lesson the students will be introduced to the **Foundation of Sport Analytics**, understanding what is it. Then, we will cover the following:

- The importance of creating a **good dataset** (.csv, database)
- **Types of data in Sport** (performance stats, biometric data...)
- **Data sources** (public datasets, API to databases, web scraping)
- **Data cleaning** (handling missing data and outliers)
- **Normalization of data** (wind-times, dates...)
- **Mini project:** clean and load our dataset in python

During the second lesson the students will be introduced to the concepts of **Exploratory Data Analysis, Trends & Features**, fundamental for prediction. Then, we will cover the following:

- How to look at a dataset? Identifying **trends & features**
- Introduction to EDA: **Exploratory Data Analysis** (plot visualization)
- **Statistical** concepts in sports (mean, median, variance, correlation...)
- Identifying trends and possible **useful features** in our case
- **Mini project**: create features in python to start training a model

















During the third lesson the students will be introduced to the concepts of **Predictive Modeling**. Then, we will cover the following:

- Introduction to **Predictive Modeling**
- How to train a model: **Training set** and **Test set**
- **Project**: train our model to predict the 100m Olympic Final, Paris 2024



What do we aim to predict?

8

Classifica	Squadra	Partecipante	Risultati
O	 USA	 Noah LYLES	9.79
A	 JAM	 Kishane THOMPSON	9.79
B	 USA	 Fred KERLEY	9.81
4	 RSA	 Akani SIMBINE	9.82
5	 ITA	 Lamont Marcell JACOBS	9.85
6	 BOT	 Letsile TEBOGO	9.86
7	 USA	 Kenneth BEDNAREK	9.88
8	 JAM	 Oblique SEVILLE	9.91

Lesson 1

Sports analytics is the **art and science of using data analysis to improve athlete performance**, predict results, refine strategies, and obtain several advantages in the sports industry. It involves collecting, analyzing, and interpreting sports-related data. It developed really fast in the last decades with the rise of digital technology. Today, it's an essential tool for every sport at every level, from youth academies to elite.

Sports analytics is about **decoding the numbers**: we collect data using dataset or technologies like wearables, video analysis, and tracking systems, then **applying statistical models** and **machine learning** to find patterns and predict outcomes. This process helps coaches, athletes, and teams make smarter decisions, improve performances, reduce injury risks and refine the details. It also improves the fan experience during the competitions.

Data has been used in sports for many years, mostly in simple ways like tracking scores and averages. But real sports analytics, using math and data in a more detailed way, only started to become important at the end of 1900.

A major turning point came with the publication of the book ***Moneyball*** by Michael Lewis in 2003. This book told the story the Oakland Athletics, a small-budget baseball team that used data analytics to compete against richer clubs. Billy Beane focused on undervalued statistics rather than traditional scouting opinions. This strategy allowed the team to find undervalued players and build a winning team despite financial limitations.

Moneyball proved that objective data could outperform subjective judgment. It inspired a wave of interest across other sports, showing that success wasn't just about money or tradition, but about smart analysis.

Track and field analytics focuses on measuring and analyzing the performance of athletes in disciplines like running, jumping, and throwing. It involves collecting data such as results, split times, acceleration, stride length, jump heights, throw distances, and biomechanical details. Coaches and athletes use these insights to refine technique, optimize training programs, monitor fatigue, and maximize competition results. In this course, we will focus on data about athletes' **100m results**, such as:

Athlete	Result	Wind	Date
Noah Lyles	9.83	+0.4 m/s	24/06/24

A good number of season results for each athlete will help us building our predictive model.

When creating a dataset, it's essential to deeply understand what are the **types of data** that can actually help us while training a model.

Here's some major categories of data commonly collected in athletics:

- **Performance Data** (Race times, jump distances, throw distances)
- **Environmental Data** (Wind speed and direction, temperature, altitude)
- **Motion and Tracking Data** (Stride length, ground contact time)
- **Event Data** (Start reaction time, false starts, fouls in jumps/throws)
- **Biometric Data** (Heart rate, blood lactate levels, body temperature)
- **Injury and Medical Data** (Injury history, recovery timelines)

For simplicity, in our course we will consider only a dataset with **performance data** and **environmental data**. In particular:

- **Race times**
 - Athletes' 100m times of all 2024 races before the Olympics
 - 100m Personal best
 - Date of each result
- **Wind speed and direction**
 - Wind conditions of each result, in order to normalize the 100m times and make them comparable.

In sports analytics, data can come from different sources. Some sports provide easy access to large public datasets or APIs, making it simple for data analysts to obtain structured information. However, it's not always the case.

In athletics, for example, **World Athletics** (the global governing body) maintains detailed athlete and competition results, but they do not offer a public API or a downloadable database.

As a result, building datasets often requires **manual data collection**, such as browsing athlete profiles and recording season results manually.

Don't worry, for this course we have already done this for you!

Athletes Dataset

The main sources of data in sports analytics include:

- **Public Datasets:** Pre-collected data available freely online, often from research projects or open databases.
- **APIs:** Automated access to data provided by sports organizations or third parties (e.g., live results, player stats).
- **Manual Data Generation:** Collecting and structuring data manually by searching official websites, documents, or video footage.
- **Federation Websites:** Official websites (like World Athletics) that publish competition results, rankings, and athlete statistics, usually without direct download options.

Noah LYLES's results

* Not legal

Event



2024



100 Metres

10.01

13 APR
2024

Tom Jones Memorial, Percy Beard Track, Gainesville, FL - Open Event


100 Metres

9.96 *

28 APR
2024

Bermuda Grand Prix, Flora Duffy Stadium, Devonshire


100 Metres

9.85

01 JUN
2024

Racers Grand Prix, National Stadium, Kingston


100 Metres

9.92

22 JUN
2024

U.S. Olympic Team Trials, Hayward Field, Eugene, OR


100 Metres

9.80 *

23 JUN
2024

U.S. Olympic Team Trials, Hayward Field, Eugene, OR


For our dataset, we chose to record all the **100m race results from 2024** for the **27 athletes** who qualified for the semifinals of the Olympic Games in Paris 2024. Our goal is to use these records to develop a model capable of predicting the outcome of the Olympic final. Our dataset will look like this:

```
1 Name, Surname, Result, Wind, Date, PB, Olympics
2 Oblique, Seville, 9.82, 0.9, 01/06/24, ,
3 Oblique, Seville, 9.98, 0.4, 27/06/24, ,
4 Oblique, Seville, 9.83, 0.4, 28/06/24, ,
5 Oblique, Seville, 9.82, 0.9, 28/06/24, PB ,
6 Oblique, Seville, 9.99, 0.0, 03/08/24, , Heat
7 Oblique, Seville, 9.81, 0.7, 04/08/24, , Semifinal
8 Oblique, Seville, 9.91, 1.0, 04/08/24, , Final
```

For all the 27 athletes.

Semifinale 1 [\[modifica \]](#) [\[modifica wikitesto \]](#)

Pos. ↕	Corsia ↕	Atleta ↕	Nazionalità ↕	Tempo ↕	Note ↕
1	6	Oblique Seville	 Giamaica	9"81	<u>Q</u> RP
2	4	Noah Lyles	 Stati Uniti	9"83	<u>Q</u>
3	5	Louie Hinchliffe	 Gran Bretagna	9"97	
4	7	Emmanuel Eseme	 Camerun	10"00	
5	9	Shaun Maswanganyi	 Sudafrica	10"02	PS
6	1	Favour Ashe	 Nigeria	10"08	
7	8	Chituru Ali	 Italia	10"14	
8	2	Rikkoi Brathwaite	 Isole Vergini Britanniche	10"15	
9	3	Benjamin Azamati	 Ghana	10"17	

Semifinale 2 [[modifica](#) | [modifica wikitest](#)]

Pos. ↕	Corsia ↕	Atleta ↕	Nazionalità ↕	Tempo ↕	Note ↕
1	5	Akani Simbine	 Sudafrica	9"87	Q
2	4	Letsile Tebogo	 Botswana	9"91	Q
3	8	Marcell Jacobs	 Italia	9"92	q PS
4	7	Kenny Bednarek	 Stati Uniti	9"93	q
5	3	Ackeem Blake	 Giamaica	10"06	
6	6	Kayinsola Ajayi	 Nigeria	10"13	
7	9	Joshua Hartmann	 Germania	10"16	
8	2	Emmanuel Matadi	 Liberia	10"18	
9	1	Reynaldo Espinosa	 Cuba	10"21	

Semifinale 3 [[modifica](#) | [modifica wikitesto](#)]

Pos. ↕	Corsia ↕	Atleta ↕	Nazionalità ↕	Tempo ↕	Note ↕
1	4	Kishane Thompson	 Giamaica	9"80	<u>Q</u>
2	7	Fred Kerley	 Stati Uniti	9"84	<u>Q</u>
3	9	Benjamin Richardson	 Sudafrica	9"95	
4	5	Abdul Hakim Sani Brown	 Giappone	9"96	RP
5	2	Andre De Grasse	 Canada	9"98	PS
6	3	Zharnel Hughes	 Gran Bretagna	10"01	
7	8	Abdul-Rasheed Saminu	 Ghana	10"05	
8	6	Ferdinand Omanyala	 Kenya	10"08	
9	1	Puripol Boonson	 Thailandia	10"14	

The objective of this course is to predict the outcome of the 2024 100m Olympic Final by analyzing the prior performances of the semifinalists.

Here's what we want to predict:

100m Olympic Final



In any data analysis project, **data cleaning** is a crucial step, and it takes a lot of time and attention. It involves identifying and **correcting** or **removing** errors, inconsistencies, and **incomplete entries** that could negatively impact the quality of the analysis. In athletics, some inconsistencies can be

- **Missing wind measurements**, making the result illegal

```
1 Name, Surname, Result, Wind, Date, PB, Olympics
2 Louie, Hinchliffe, 10.40, NWI, 20/04/24, ,
```

- Results marked as **DNS** (Did Not Start) or **DQ** (Disqualified).

```
1 Name, Surname, Result, Wind, Date, PB, Olympics
2 Letsile, Tebogo, DNS, 1.1, 22/06/24, ,
3 Kishane, Thompson, DNF, 1.0, 18/05/24, ,
```

- **Unusually high race times**, possibly due to injuries or lack of effort during qualification rounds.

```
1 Name, Surname, Result, Wind, Date, PB, Olympics  
2 Noah, Lyles, 10.04, -0.2, 03/08/24, , Heat
```

Such entries must be carefully removed to prevent them from interfering with model training and predictions.

However, we must be careful. In more advanced models, factors like injuries could be considered important information, so we shouldn't remove them but give them relevance instead. In our case, for simplicity, we chose to ignore such considerations for a simpler model. We can do it with a simple function:


```
1 def data_cleaning(df):
2     def is_float(value):
3         try:
4             float(value)
5             return True
6         except:
7             return False
8
9     df_cleaned = df[df['Result'].apply(is_float) & \
10                    df['Wind'].apply(is_float)]
11     return df_cleaned
12
13 dataset = data_cleaning(dataset)
14 print(dataset.head(20))
```

In this way, for example, instead of:

```
1 ...  
2 Louie, Hinchliffe, 10.40, NWI, 20/04/24, ,  
3 Louie, Hinchliffe, 10.37, 0.2, 26/04/24, ,  
4 Louie, Hinchliffe, 10.21, 0.8, 27/04/24, ,  
5 ...
```

We obtain:

```
1 ...  
2 Louie, Hinchliffe, 10.37, 0.2, 26/04/24, ,  
3 Louie, Hinchliffe, 10.21, 0.8, 27/04/24, ,  
4 ...
```

In data analysis, **normalization** is the process of adjusting values measured on different scales to a common scale, to make comparison possible.

In athletics, and particularly in sprint events like the 100m, external factors such as **wind speed** can significantly **affect performance times**.

A **tailwind** (wind in the same direction as the sprinter) can make athletes run **faster**, while a **headwind** (wind against the sprinter) slows them down.

So, to fairly compare 100m results across different races, it is essential to normalize the times to a standard condition, typically assuming **no wind**.

To achieve this, we will use a correction formula developed by researchers M. Moinat, O. Fabius, and K.S. Emanuel, published in 2018 in the article: *"Data-driven quantification of the effect of wind on athletics performance"*.

I got in touch with those researchers and I personally developed an iOS application called **Wind Correction**, that implements this correction to adjust race times based on wind conditions. The formula we will use for normalization is as follows:

$$P_{\text{new}} = P - 0.0449w + 0.009459Pw - 0.0042w^2$$

Where:

- P is the original race time,
- w is the wind speed (positive for tailwind, negative for headwind),
- P_{new} is the corrected time under neutral (no wind) conditions.

This formula provides an accurate adjustment and will be applied to all times in our dataset to ensure a fair comparison between the performances. We will create a column with all the corrected times that will be used for prediction. Notice that, then, our output will be based on **neutral wind conditions**.

WIND <div><div></div></div> +4.0 m/s			
Input performance		Predicted performance	
Event	at +4.0 m/s	at 0.0 m/s	at +2.0 m/s
100m	<input type="text" value="9.90"/>	10.03	9.95

Let's see how we would do it in python.

```
1  # Define the function to correct the 100m time based on wind
2  def normalize_time(row):
3      time = row["Result"]
4      wind = row["Wind"]
5      Time_Norm = time - (0.0449 * wind) + (0.009459 * time * \
        wind) - (0.0042 * wind ** 2)
6      return Time_Norm
7
8  # Apply the correction to the dataset
9  dataset["Time_Norm"] = dataset.apply(normalize_time, axis=1)
10
11 # Round the corrected time to two decimal places
12 dataset["Time_Norm"] = dataset["Time_Norm"].round(2)
```

Now, each row in the dataset will have a new column, called Time_Norm:

1	Name,	Surname,	Result,	Wind,	Date,	...,	Time_Norm
2	Oblique,	Seville,	9.82,	0.9,	01/06/24,	...,	9.86
3	Oblique,	Seville,	9.98,	0.4,	27/06/24,	...,	10.00
4	Oblique,	Seville,	9.83,	0.4,	28/06/24,	...,	9.85
5	Oblique,	Seville,	9.82,	0.9,	28/06/24,	...,	9.86
6	Oblique,	Seville,	9.99,	0.0,	03/08/24,	...,	9.99
7	Oblique,	Seville,	9.81,	0.7,	04/08/24,	...,	9.84
8	Oblique,	Seville,	9.91,	1.0,	04/08/24,	...,	9.95

In machine learning, raw data by itself is often not very informative for training a predictive model. What we actually need are **features**, which are transformations or combinations of the original data that capture meaningful relationships or trends.

A **feature** is a measurable property or characteristic extracted from the raw data, used to help the model learn patterns. During model training, each feature is assigned a **coefficient weight**, which quantifies its **importance** in predicting the output.

In the context of athletics, a single 100m race result may not provide much information by itself. However features based on multiple race results can capture important insights about an athlete's condition and progression.

Let's see some examples:

- The difference between the most recent and the oldest 100m time for each athlete, indicating whether an athlete is improving or declining.
- The trend over the last three races, showing short-term performance progression or regression.
- The time difference between the current date and the date of the athlete's personal best, giving information about when the athlete reached their peak and how far they are from that form.

Let's see an easy feature creation example, which can be finding each athlete's **Season Best (SB)**, meaning his fastest 100m time of 2024 season.

```
1 # Filter out Semifinal and Final (we want to predict them)
2 season_races = \
    dataset[~dataset["Olympics"].isin(["Semifinal", "Final"])]
3
4 # Create a new table with Season Best for each athlete
5 SB_table = season_races.groupby(["Name", \
    "Surname"])["Result"].min().reset_index()
6
7 SB_table = SB_table.rename(columns={"Result": "Season_Best"})
8
9 print(SB_table)
```

By designing and selecting the right features, we can provide the model with much more meaningful inputs, improving both the accuracy and the interpretability of its predictions.

In a **simple linear regression model**, the relationship between the input features and the predicted outcome is expressed as:

$$y = b + w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_nx_n$$

Where:

- y is the predicted output,
- b is the bias or intercept term,
- x_1, x_2, \dots, x_n are the features (inputs),
- w_1, w_2, \dots, w_n are the coefficients (weights) associated with each feature.

Each coefficient w_i determines **how much** the corresponding feature x_i contributes to the final prediction. Features with larger absolute coefficients have a greater impact on the output.

We will talk about features more in detail in the next lesson.

For this first mini project we want you to download the dataset you find on ICorsi and import it in a python project.

In Python, two of the most important libraries for data analysis are **NumPy** and **Pandas**. In particular:

- **NumPy** is mainly used for numerical computations with arrays and matrices, where all the data is usually of the same type (e.g., only numbers).
- **Pandas**, on the other hand, is specifically designed to work with structured data, such as tables with different types of information (text, numbers, dates).

Importing a dataset in **Python** means reading a data file (for example, a CSV file) and converting it into a structured object that we can easily manipulate and analyze.

In our case, since our dataset contains mixed types (names, race times, wind readings, dates, and labels), which are strings, numbers, dates etc, we will use Pandas to load the data as a **DataFrame**.

A DataFrame allows us to access columns by name, filter data, group records, and perform complex operations easily, which would be much more difficult using NumPy alone.

```
1 # Import the dataset using pandas (instead of numpy)
2 dataset = pd.read_csv("AthletesDataset.csv", delimiter=",", \
   header=0)
3
4 # Show the first few rows to check
5 print(dataset.head(20))
6
7 # Convert string to numbers
8 dataset["Result"] = pd.to_numeric(dataset["Result"], \
   errors='coerce')
9 dataset["Wind"] = pd.to_numeric(dataset["Wind"], \
   errors='coerce')
10 dataset["Date"] = pd.to_datetime(dataset["Date"], \
   format="%d/%m/%y", errors='coerce')
```

Lesson 2

Exploratory Data Analysis (EDA) is a fundamental step in any data science project that focuses on analyzing a dataset, using visual methods.

The goal of EDA is to understand the structure of the data, detect **patterns**, identify anomalies (outliers), and generate hypotheses that can guide further analysis or modeling.

In the context of our project, visualizing data through graphs, such as scatter plots or line charts, is particularly helpful to analyze athletes' performance data and to uncover **trends** and **relationships**.

In fact, visual analysis helps the creation of meaningful features from raw data, which can then be used to train predictive models and improve their accuracy.

Is measurable property or characteristic of the data you're analyzing

Types of features:

- Demographics
- Temporal
- Geographic
- Regular and continuous
- Derived
- Environmental
- Textual

- country
 - season_best_time
 - average_speed_during_final
 - sponsor
 - previous_olympic_experience
(binary or count)
 - training_country
 - injury_history
 - age
- rank_in_final
 - training_club
 - final_time

- country
 - season_best_time
 - average_speed_during_final
 - sponsor
 - previous_olympic_experience
(binary or count)
 - training_country
 - injury_history
 - age
- rank_in_final **very bad**
 - training_club
 - final_time **cheating**

Let's start with an easy one that we introduced before: **Season Best (SB)** of each athlete. This is the best time result for each athlete that has been run in 2024 season before the Olympic semifinal (we want to predict that). It's different from the all time personal best (PB).

We prefer to not merge the features into our original table because, since we have a lot of results for each athlete, we would have multiple rows with the same feature value. It's better to create a new table with the Surname of each athlete as a **key**, and we will place every feature in that table.

The **Season Best** will be our first feature:

```
1  szn_data = dataset[(dataset['Date'] >= '2024-01-01') & \
    (dataset['Date'] < '2024-08-04')]
2  SB = szn_data.groupby('Surname')['Result'].min().reset_index()
3  SB = SB.sort_values('Result')
4  SB.rename(columns={'Result': 'Season_Best'}, inplace=True)
5
6  features_table = SB.copy()
7  print(features_table)
```

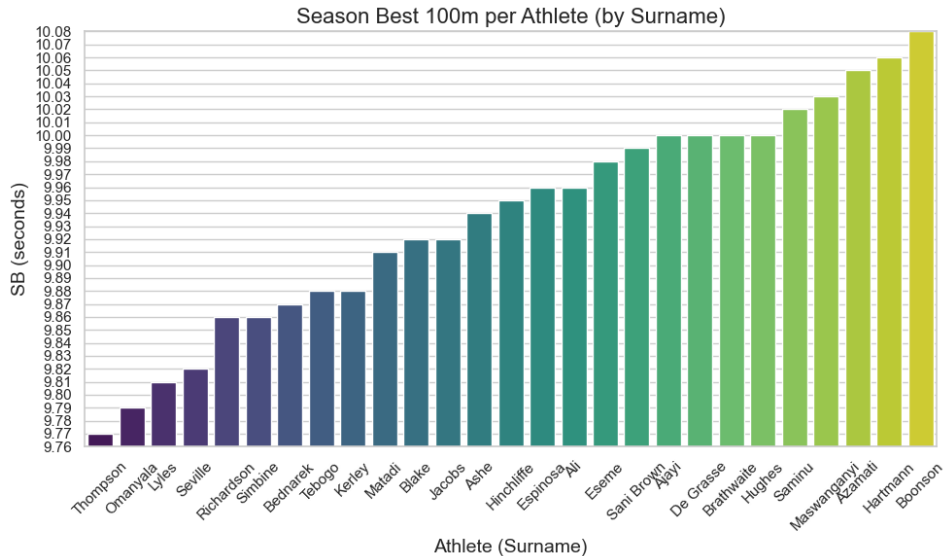
Now, we can plot this feature sorted by the **fastest season best**, to see which athletes run faster during the 2024 season, before the Olympics.

To plot, we use the Seaborn library, with the following code:

```
1  import seaborn as sns
2
3  # Set seaborn style
4  sns.set(style="whitegrid")
5
6  # Create the bar plot
7  plt.figure(figsize=(10, 6))
8  sns.barplot(data=SB, x='Surname', y='Season_Best', \
9              hue='Surname', palette='viridis', legend=False)
10
11 # Add title and labels
12 plt.title('Best 100m Time per Athlete (by Surname)', \
13           fontsize=16)
```

```
12 plt.xlabel('Athlete (Surname)', fontsize=14)
13 plt.ylabel('Best Time (seconds)', fontsize=14)
14
15 # Set y-axis range and ticks (from 9.76 to 10.08, step 0.01)
16 plt.ylim(9.76, 10.08)
17 plt.yticks(np.arange(9.76, 10.08, 0.01))
18
19 # Rotate x-axis labels if needed
20 plt.xticks(rotation=45)
21
22 # Show the plot
23 plt.tight_layout()
24 plt.show()
```

And the histogram will result like the following.

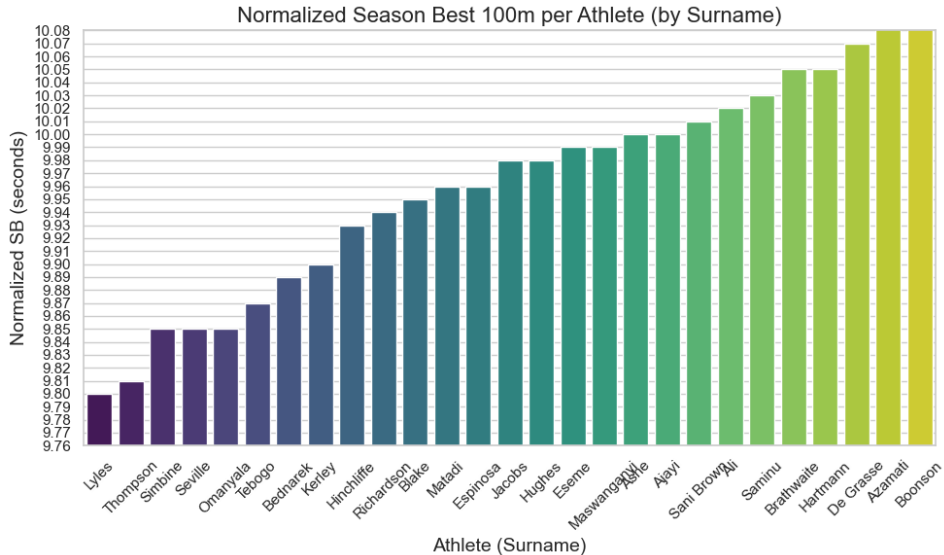


From this plot we can already see the athletes that run faster during the season:

- 1 1. Thompson
- 2 2. Omanyala
- 3 3. Lyles
- 4 4. Bednarek
- 5 5. Seville
- 6 ...

However, some athlete might have a faster season best because it was very windy. Let's indeed utilize the normalized times that we created before!

















I will omit the code here, but we just need to create and add to our new table a new feature `Normalized_SB`, and then plot it.



Here, there are some differences. From this plot we can see the athletes that run faster during the season if we consider neutral wind conditions:

- 1 1. Lyles
- 2 2. Thompson
- 3 3. Simbine
- 4 4. Seville
- 5 5. Omanyala
- 6 6. Tebogo
- 7 7. Bednarek
- 8 8. Kerley
- 9 ...

Let's compare this result to the actual Olympic final qualified athletes.

Team	Participant
 USA	 Noah LYLES
 JAM	 Kishane THOMPSON
 USA	 Fred KERLEY
 RSA	 Akani SIMBINE
 ITA	 Lamont Marcell JACOBS
 BOT	 Letsile TEBOGO
 USA	 Kenneth BEDNAREK
 JAM	 Oblique SEVILLE

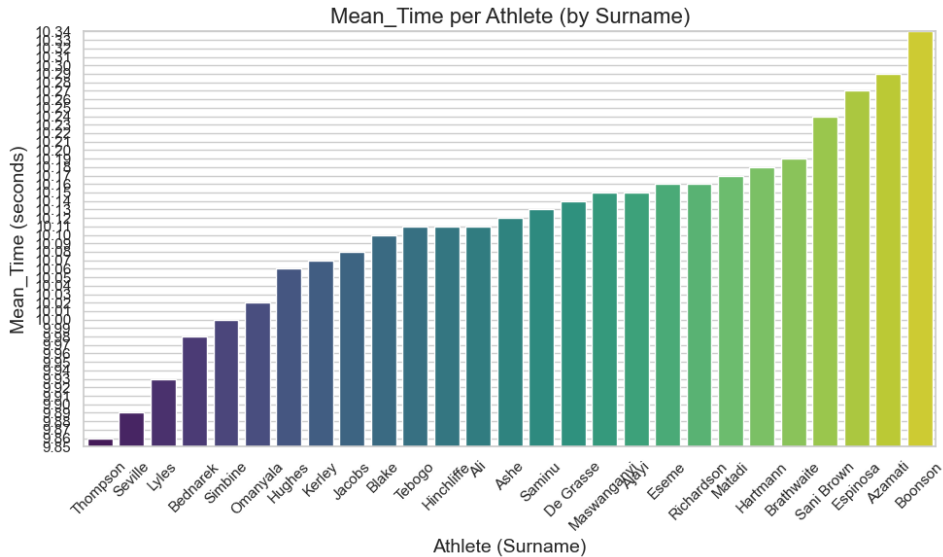
From the true `Season_Best` we center 5 out of 8 athletes of the Olympic final, ignoring the final placing.

From the `Normalized_Season_Best` we center 7 out of 8 athletes that are in the Olympic final. We haven't built a model yet, and this feature already tells us a lot about the athletes.

This is a very good thing because it means that our features are accurate and they will have a good weight when training and using the predictive model.

Now, let's create a new feature, the **Mean Time** of each athlete. This is the average of the 2024 results of each athlete before the Olympics.

```
1  szn_data = dataset[(dataset['Date'] >= '2024-01-01') & \
2      (dataset['Date'] < '2024-08-04')]
3  Mean_Time = \
4      szn_data.groupby('Surname')['Time_Norm'].mean().reset_index()
5  Mean_Time = Mean_Time.sort_values('Time_Norm').round(2)
6  Mean_Time.rename(columns={'Time_Norm': 'Mean_Time'}, \
7      inplace=True)
8  features_table = features_table.merge(Mean_Time, \
9      on='Surname', how='left')
```

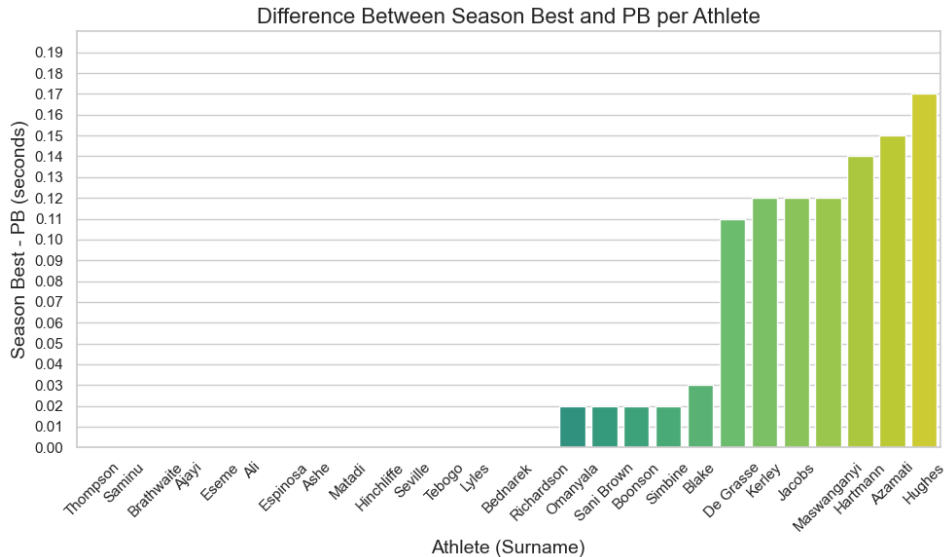


Now, we want to create another feature, the **difference between all time Personal Best (PB) and Season Best (SB)**, to see how close each athlete is this year to his prime. In this case we will not use normalized times, because it wouldn't be realistic.

```
1 # Filter only rows where PB column is 'PB'
2 pb_data = dataset[dataset['PB'] == 'PB']
3 PB = pb_data.groupby('Surname')['Result'].min().reset_index()
4
5 PB.rename(columns={'Result': 'PB_Time'}, inplace=True)
6
7 # Merge PB into features_table (PB it's itself a feature)
8 features_table = features_table.merge(PB, on='Surname', \
    how='left')
```

```
9
10 # Calculate difference: Season_Best - PB_Time
11 # It will be positive if SB is slower than PB, zero if equal)
12 features_table['SB_PB_Diff'] = \
    (features_table['Season_Best'] - \
     features_table['PB_Time']).round(2)
```

Now, we can visualize this. The plot will tell us which athlete are in their best shape during the current season (2024).



In the mid-2010s, Amazon developed an artificial intelligence system to automate the screening and ranking of job applicants' resumes. The goal was to streamline the hiring process by training a machine learning model to identify top candidates based on historical hiring data.

What Went Wrong?

The model became biased against women. It consistently ranked male candidates higher than female candidates, even when qualifications were similar.

Root Cause: Poor Feature Selection and Training Data Bias

The core issue stemmed from the features the model learned to associate with successful candidates:

- The model was trained on resumes submitted over a ten-year period, during which most applicants (especially in technical roles) were men.
- As a result, the model learned to associate male-oriented language and experiences with higher performance.
- Resumes that included terms such as “women’s chess club captain” or degrees from all-women’s colleges were penalized.
- Gender itself was not an explicit feature, but the model identified proxies

for gender, such as word choice and affiliations.

Outcome

Despite initial enthusiasm, Amazon ultimately scrapped the system in 2017 after internal audits revealed that it exhibited gender bias and could not be trusted to make fair hiring decisions.

Key Takeaways

- **Historical bias** in training data can be replicated or amplified by machine learning models.
- **Feature selection must account for ethical and societal impacts**, not just technical correlation.
- Models can pick up on **proxy variables** for sensitive attributes (like gender or race), even when those attributes are not explicitly included.

Lesson 3

Predictive models are mathematical or computational tools used to predict future outcomes based on historical data. These models analyze patterns in data to make informed predictions, often using techniques from statistics, machine learning, or data mining.

In track and field, as well as in other sports, predictive models are commonly used to **estimate athletes' performance**, optimize training strategies, assess injury risks, and more. For example, a coach might use a model to predict an athlete's race time based on training load, nutrition, sleep quality, and previous race times.

Now we will see some examples of predictive models, from the simpler ones to the more complicated ones.

Linear Regression is a fundamental model used for predicting a continuous outcome. It assumes a linear relationship between input variables and the output. The model is expressed as:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

where \hat{y} is the predicted value, β_i are the coefficients, and x_i are the input features.

Linear regression is widely used in sports to estimate results, such as predicting an athlete's finishing time in a race.

Logistic Regression is used for binary classification problems. It models the probability that an instance belongs to a particular class using the logistic function:

$$P(y = 1|x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n)}}$$

In this formula, x_1, \dots, x_n are the input features, β_0 is the intercept, and β_1, \dots, β_n are the model coefficients.

The expression in the exponent, $-(\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n)$, is the linear combination of inputs, and applying the sigmoid function compresses this into a probability between 0 and 1. In athletics, logistic regression can be used to classify outcomes such as determining which of two athletes is more likely to run faster in an upcoming race, without predicting the exact times.

Random Forest is a machine learning model that combines many decision trees to make a more accurate prediction. Each tree gives a prediction, and the final output is the average (for regression) or the majority vote (for classification). This method reduces errors and avoids overfitting.

Gradient Boosting is another model that also uses decision trees but builds them one after another. Each new tree focuses on correcting the errors of the previous ones. This step-by-step improvement makes gradient boosting very powerful for complex problems. In athletics, it can be used to predict outcomes or rankings with high accuracy, using detailed performance data.

Neural Networks are composed of layers of interconnected neurons, capable of learning complex and non-linear relationships. Each neuron computes a weighted sum of inputs, applies an activation function, and passes the result to the next layer. A simple feedforward neuron computes:

$$a = \sigma \left(\sum_{i=1}^n w_i x_i + b \right)$$

where σ is an activation function, w_i are weights, x_i inputs, and b a bias term.

Let's get back to the task. We have a dataset containing the 27 semifinalists of the Olympic 100m event, and our first goal is to predict **which 8 athletes will qualify for the final**.

To keep things simple, and because we don't have a large amount of data, we'll use a **Heuristic Ranking Model**, which is not a machine learning model (it doesn't "learn" from data) but a model that uses fixed rules and weights to make predictions, instead.

This heuristic model ranks athletes based on a weighted combination of three performance features.

In our case, we used:

- Their normalized season best time (**Normalized_SB**)
- Their average time during the season (**Mean_Time**)
- Their most recent result before the Olympic heats (**Recent_Form**)

Each component reflects a different aspect of the athlete's form, and is weighted (0.4, 0.3, 0.3 respectively) to compute a final score where lower is better. The model then selects the top 8 athletes with the lowest scores as the predicted finalists. This method doesn't use machine learning, but it provides a logical and interpretable way to simulate qualification based on relevant past performance.

Let's create a new feature that captures the most recent result for each athlete before the Olympic qualification rounds. This will give us a good indication of each athlete's current form leading into the competition.

```
1 recent_data = dataset[dataset["Date"] < "2024-08-03"]
2 recent_data = recent_data.sort_values("Date")
3
4 recent_form = \
    recent_data.groupby("Surname").tail(1)[["Surname", \
        "Time_Norm"]].rename(columns={"Time_Norm": "Recent_Form"})
5 features_table = features_table.merge(recent_form, \
    on="Surname", how="left")
```



```
1 # --- 1. Calculate the combined score (lower is better) ---
2 # The weights can be adjusted as needed
3 features_table["Score"] = (
4     0.4 * features_table["Normalized_SB"] +
5     0.3 * features_table["Mean_Time"] +
6     0.3 * features_table["Recent_Form"]
7 )
8
9 # --- 2. Rank all athletes by score ---
10 ranked = \
11     features_table.sort_values("Score").reset_index(drop=True)
12
13 # Select the top 8 athletes with the lowest score as \
14     predicted finalists
15 qualified_names = ranked.head(8)["Surname"].tolist()
```

14
15
16
17
18
19
20
21
22
23
24

```
# --- 3. Use the full list as semifinalists obtaining the \
original order
semifinalists_names = dataset[dataset["Olympics"] == \
    "Semifinal"]["Surname"].tolist()[ :27]
semifinalists = (
    features_table[features_table["Surname"].isin(semifinalists_name
    .assign(OriginalOrder=lambda df: \
        df["Surname"].apply(semifinalists_names.index))
    .sort_values("OriginalOrder")
    .reset_index(drop=True)
)

# --- 4. Print the 3 semifinals (3 groups of 9), sorted \
internally by score ---
```

```
25 for i in range(3):
26     print(f"\nSemifinal {i+1}")
27     group = \
        semifinalists.iloc[i*9:(i+1)*9].sort_values("Score")
28     for _, row in group.iterrows():
29         q = " Q" if row["Surname"] in qualified_names else ""
30         print(f"{row['Surname']} ({row['Score']:.3f}){q}")
31
32 # --- 5. Print the qualifieds ---
33 print("\nQualified for the final")
34 for _, row in ranked.head(8).iterrows():
35     print(f"{row['Surname']} ({row['Score']:.3f})")
```

Semifinal 1	Semifinal 2	Semifinal 3
Lyles (9.839) Q	Simbine (9.895) Q	Thompson (9.846) Q
Seville (9.862) Q	Bednarek (9.929) Q	Kerley (9.951) Q
Hinchliffe (9.990)	Tebogo (9.942) Q	Omanyala (9.961) Q
Ali (10.047)	Blake (9.995)	Hughes (10.004)
Maswanganyi (10.059)	Jacobs (10.010)	Richardson (10.036)
Ashe (10.069)	Matadi (10.023)	Saminu (10.060)
Eseme (10.080)	Ajayi (10.045)	De Grasse (10.103)
Brathwaite (10.155)	Hartmann (10.098)	Sani Brown (10.142)
Azamati (10.199)	Espinosa (10.167)	Boonson (10.343)

Aside from Jacobs, this method correctly identifies 7/8 actual finalists.

Now, let's improve our prediction for the final by adding a new feature: each athlete's actual semifinal time. This should help us make more accurate forecasts for the final performance.

```
1 semifinal_times = dataset[dataset['Olympics'] == \
    'Semifinal'][['Surname', 'Result']]
2 semifinal_times.rename(columns={'Result': 'Semifinal_Time'}, \
    inplace=True)
3 features_table = features_table.merge(semifinal_times, \
    on='Surname', how='left')
```

As a first approach, we'll use **linear regression** to predict the outcome of the final, based on the semifinal results we previously predicted, which happen to match 7 out of the 8 actual finalists.

To **train the model**, we'll use the **non-finalists**, treating their semifinal time as the **target variable**. This allows the model to learn the relationship between season features and semifinal performance.

Once trained, we'll **apply the model** to the **finalists**. Although we don't know their actual final times, this approach allows us to generate a predicted ranking based on their features and semifinal performance.

```
1 from sklearn.linear_model import LinearRegression
2
3 # 1. Predicted finalists
4 pred_fin = ranked.head(8)["Surname"].tolist()
5
6 # 2. Test set: predicted finalists
7 final_input = \
8     features_table[features_table["Surname"].isin(pred_fin)].copy()
9 final_input = final_input.dropna(subset=["Semifinal_Time"])
10
11 X_test = final_input[["Season_Best", "Normalized_SB", \
12     "Mean_Time", "Semifinal_Time"]]
13
14 # 3. Training set: all the non finalist athletes
15 non_fin = \
```

```
14     features_table[~features_table["Surname"].isin(pred_fin)].copy()  
15 non_fin = non_fin.dropna(subset=["Semifinal_Time"])  
16 X_train = non_fin[["Season_Best", "Normalized_SB", \  
17     "Mean_Time", "Semifinal_Time"]]  
18 y_train = non_fin["Semifinal_Time"] # Target  
19 # 4. Linear Regression  
20 lr_model = LinearRegression()  
21 lr_model.fit(X_train, y_train)  
22 final_input["LR_Pred"] = lr_model.predict(X_test).round(3)  
23  
24 # --- Print results ---  
25 ...
```


The output is the following:

1	Thompson	9.80 s
2	Seville	9.81 s
3	Lyles	9.83 s
4	Kerley	9.84 s
5	Simbine	9.87 s
6	Tebogo	9.91 s
7	Bednarek	9.93 s
8	Omanyala	10.08 s

When we compare the predicted results to the actual final outcomes, we see that while the prediction isn't perfectly accurate, it's still very close. Achieving an exact match is extremely difficult in such a competitive and unpredictable event.

If we want to go further, we can denormalize the predicted times using the actual wind condition of the final (+1.0 m/s).

Additionally, we could try running the model on the 8 real finalists instead of the ones we originally predicted, to see how the output compares.

Linear Regression builds a single formula that connects your input features (like Season Best, Normalized SB, Mean Time, and Semifinal Time) to the result. It assumes that as these values increase or decrease, the final time also changes in a smooth and predictable way.

In this case, these features are all numerical and have a clear relationship to performance, so Linear Regression can find a good trend.

Differently, random forest gives a different result:

```
1 rf_model = RandomForestRegressor(n_estimators=100, \
    random_state=42)
2 rf_model.fit(X_train, y_train)
3 final_input["RF_Pred"] = rf_model.predict(X_test).round(3)
```

1	Tebogo	9.98 s
2	Thompson	9.98 s
3	Lyles	9.98 s
4	Seville	9.98 s
5	Simbine	9.98 s
6	Bednarek	9.98 s
7	Kerley	9.98 s
8	Omanyala	10.08 s

This because random Forest works differently: it splits the data into groups using decision trees and predicts by averaging results inside each group.

This works well when we have lots of data and complex patterns. But in this case, there are two problems:

- The training set is small, only about 19 athletes, so the trees don't have much to learn from.
- Many athletes have very similar features, so the trees often group them together.

That's why several finalists end up with the same predicted time, the model treats them as if they're nearly identical and gives them the same average value, which is incorrect for our purpose.

Logistic regression can be effectively used to compare athletes in a pairwise way. By modeling the probability that one athlete would beat another based on the differences in their performance features (such as season best, average time, and semifinal result), we can simulate direct matchups.

For each pair, the model predicts which athlete is more likely to win. By repeating this for all possible pairings among finalists, we can count how many "wins" each athlete accumulates and use that to generate a final ranking. This approach is particularly useful when absolute times are less important than relative placement.

Among the different approaches explored in our project, linear regression proved to be the most effective. It provided the most accurate predictions of final performance with a ranking that closely matched the real results.

Vento: +1,0 m/s

Pos. ↕	Corsia ↕	Atleta ↕	Nazionalità ↕	Tempo ↕	Note ↕
	7	Noah Lyles	 Stati Uniti	9"79 (.784)	RP
	4	Kishane Thompson	 Giamaica	9"79 (.789)	
	3	Fred Kerley	 Stati Uniti	9"81	PS
4	5	Akani Simbine	 Sudafrica	9"82	RN
5	9	Marcell Jacobs	 Italia	9"85	PS
6	8	Letsile Tebogo	 Botswana	9"86	RN
7	2	Kenneth Bednarek	 Stati Uniti	9"88	
8	6	Oblique Seville	 Giamaica	9"91	



Thank you

```
1 import pandas as pd
2
3 # Import the dataset
4 dataset = pd.read_csv("athletes.csv", delimiter=";", header=0)
5
6 # Filter 2024 results before the Olympics for each athlete
7 szn_data = dataset[(dataset['Date'] >= '2024-01-01') &
8                    (dataset['Date'] < '2024-08-03')]
9
10 # The season best (SB) is the faster result of the season
11 SB = szn_data.groupby('Surname')['Result'].min().reset_index()
12 SB = SB.sort_values('Result')
13 SB.rename(columns={'Result': 'Season_Best'}, inplace=True)
14
15 # Create a table with all the features
```



```
16 | features_table = SB.copy()  
17 | print(features_table)
```