University of Pisa

Master's Degree Program in Computer Science

Data Mining - A.A 24/25

From Raw Cycling Data to Predictions

By

Riccardo Marcaccio

Margherita Pensa

Oleksiy Nedobiychuk

1 Introduction

In this project, we applied data mining techniques to analyze, clean, and preprocess two datasets on cyclists and races, engineered relevant features, detected outliers, applied clustering algorithms, and built predictive models to classify whether a cyclist would finish in the top 20 positions of a race. Sequence of action made for elaborate datasets are:

1. Analyze distribution and correlation between features.
2. Fill missing data and delete useless features for cleaning datasets.
3. Feature engineering for managing missing values and adding more useful informations.
4. Applying clustering for analyzing pattern inside the datasets.
5. Prepare a dataset for machine learning and train a set of models to it.

2 Data understanding

Data understanding phase was applied for have a general view about: distributions value, missing value distributions, semantically checking of values for each feature. From this analyis it was taked decision about data transformation and feature engineering.

2.1 Feature distributions

After a discrete analysis, it is possible to see that almost all the features have a gaussian distributions (except for points, uci_points and profile). This allow to use gaussian distributions when sampling can be necessary. In this analysis, features like nationality, team_cyclists etc. are not analyzed because string values are not relevant (also because is difficult to analyze distribution to a non numerical domain).

2.2 Correct values check

The objective of this analysis is to ensure that the dataset is free of potential errors, such as invalid or incorrectly formatted entries, duplicates, and discrepancies that could hinder the analysis. Validating these aspects is crucial for data integrity and consistency before proceeding to complex tasks, such as feature engineering or machine learning. .

Observations

Analysis of the Cyclists and Race datasets reveals key findings:

- Some cyclists only appear in one dataset (for example, in `Race` but not in `Cyclists`), requiring updates to ensure that all cyclists in one dataset are also in the other.
- Three main issues are identified in the `Race` dataset:
  1. **Points Duplication Issue**: The `points` and `uci_points` columns contained identical values across all cyclists within each race. These values appeared to be copied from the first-place finisher's point allocation.
  2. **Position Recording Issue**: Disqualified cyclists retained their original positions in the dataset. Subsequent riders were incorrectly shifted down one position. For example, if a cyclist in 15th place was disqualified, the replacement rider was incorrectly listed in 16th place.
  3. **Negative Deltas**: The `delta` column contained negative values for some cyclists. These negatives arose from different representations where the first rider's time was used as a reference point, and all other riders' times were subtracted from it. For example, if the first rider's time was `0:46:18`, subsequent riders had negative deltas like `-44:05`, `-44:02`, etc. We assumed that the correct delta should represent the difference between the first rider's time and subsequent riders' times.

Proposed Solution

To resolve these issues, we scraped `ProCyclingStats` for the correct data.
By scraping the data, we observed the following:

- Some cyclists were present only in the `Cyclists` dataset, which is normal since disqualified cyclists were excluded from our updated dataset.
- Some cyclists were present only in the updated `Race` dataset, which is not normal. To address this, we enhanced the scraping process to look for the missing cyclists in the `Cyclists` dataset and add them to the updated `Race` dataset. The missing cyclists' information was obtained from the same source.

2.3 Missing values

An analysis was done on the datasets to better understand the distribution of missing values in the datasets. This analysis served to better understand which applications need to be made in the data transformation phase, understanding how to fill the missing data and whether a merging phase between the datasets is necessary to fill some columns.
In the cyclist dataset we can see that:

- **birth_year** = the column has 21 missing values and will be filled, as it is relevant as a characteristic.
- **weight** / **height** = There are 2 columns with 3241 and 3176 missing values respectively and they will be filled.
- **Nationality** = Even if it is not a relevant feature, it will be filled by searching for the only missing value on the internet.

In the race dataset, we can see that:

- **uci_points** = the column has 346245 missing values, so a transformation is necessary, as it contains an important alternative score for the races (international cycling union score).
- **climb_total** = the column has 153766 missing values and a transformation is necessary, as the total distance of the route is relevant.
- **profile** = the column has 155351 missing values, and contains the difficulty of each race, therefore an important feature to fill.
- **average_temperature** = the column contains more missing values than all: 567440 missing values but it will not be filled, because it is not correlated to other characteristics and it is not safe to sample values from a distribution, as the temperature is an influencing factor of the race, being that it influences the athlete's body during the race.
- **cyclist_team** = the column contains 2457 missing values but will not be filled, as the team name is not relevant.
- **delta** = the column contains 50259 missing values, and the difference between the first comer is a feature with its impact of importance.

2.4 Correlation between features

The subsequent analysis of column relationships will investigate various elements of the riders' performance, identify traits for data imputation and explore potential correlations that could inform predictive modeling.

2.4.1 Correlation Matrix

We begin the feature correlation exploration by examining the correlation matrix (Figure 1), which includes all the numerical features from both datasets combined.

We can observe strong positive correlations: `height` vs. `weight` (0.74), `climb_total` vs. `profile` (0.71), and `points` vs. `uci_points` (0.84). We also find a notable positive correlation between `climb_total` and `length` (0.52) and a weaker one for `delta` and `length` (0.25).

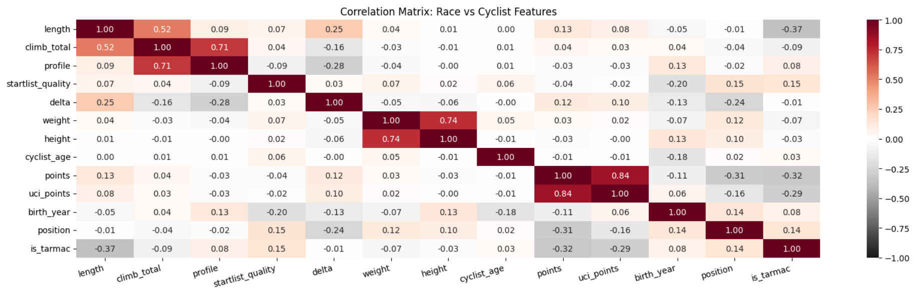

Figure 1: Correlation matrix between all relevant features.

On the negative side, `length` is weakly correlated with `is_tarmac` (0.36), suggesting that longer races are less likely to be primarily on tarmac. `Points` also shows a negative correlation with `is_tarmac` (0.32), implying that higher-point races often include varied terrain. Other negative correlations include `points` vs. `position` (0.31) and `profile` vs. `delta` (0.28), where the latter suggests that more challenging profiles may result in smaller time gaps, possibly due to tighter rider grouping.

2.4.2 Points, Age, Temperature and Career Span

Even though the number of available `average_temperature` samples is limited, it is informative to compare how age groups perform across different temperature ranges.

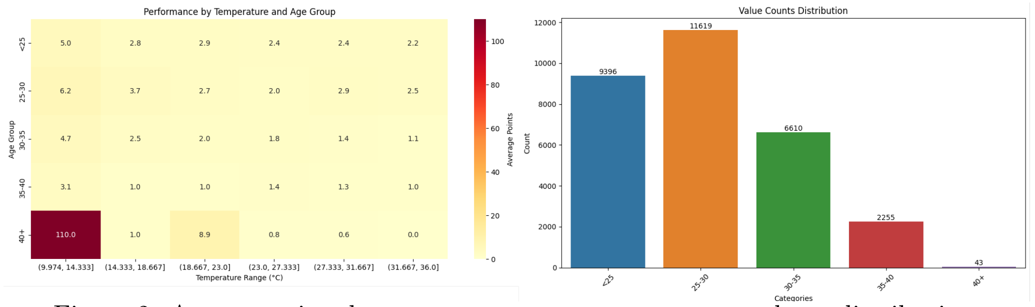

Figure 2: Average points by age group, temperature range and age distribution.

As shown in Figure 2 (first plot), the 40+ group exhibits high average points (110.0) within the (9.9–14.3) temperature range, and 8.9 points within the (18.6–110.0) range. However, this group has a very small sample size (Figure 2 second plot), partly because riders often retire before reaching 40. Consequently, the limited data for this group may skew the average.
Although there is no observed correlation between cyclist age and points, it remains compelling to examine how age influenced rider performance and cyclists' career length. Therefore next, we examine how age correlates with riders' performance and career duration. Figure 3, first plot, shows a bell-shaped distribution, indicating that for most cyclists, age has a limited impact on performance. Some riders experience a decline, while others improve as they age. The age-performance correlation plot highlights outliers at the tail of the distribution. The strong negative correlation suggests some cyclists excelled early in their careers, but then experienced a marked decline. Conversely, others started with weaker performances and improved significantly over time. The second right-skewed distribution, Figure 3 second plot, likely reflects the sport's high physical demands: many riders competed for no more than three years, and relatively few pursued extended careers.

2.4.3 Physical Attributes, Performance and Race Profile

Finally, we explore how physical attributes may affect a rider's final position across different race profiles. We also identify which personal features dominate in each profile. The Figure 3, third plot, indicates that as climb difficulty increases, `weight` and `height` become more critical in determining a rider's final position. Notably, `weight` strongly influences outcomes in high mountain races, compared to `height`.

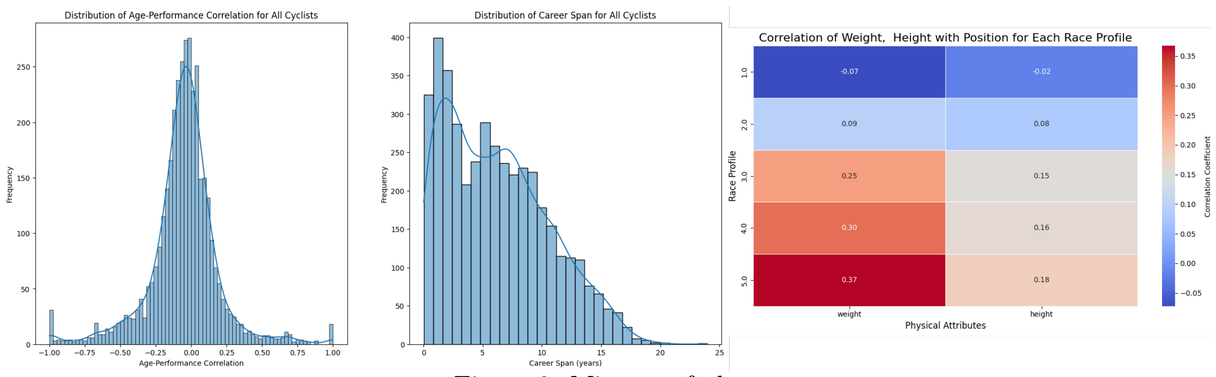

Figure 3: Mixture of plots

For each race profile, we highlight the top 10 riders for that profile alongside their five key attributes, illustrating the qualities required to stand out in a specific race type. In the Figure 4, BMI stands for Body Mass Index. The `experience` is calculated using the formula: `(race_date - career_start).dt.days / 365.25`.

Here, `race_date` is the date of the race, and `career_start` is the date of the cyclist's first race. Heavier, taller, and more experienced mid-career riders tend to excel on flat profiles. Heavier,

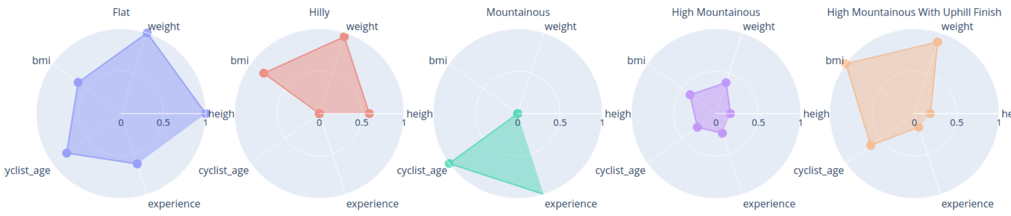

Figure 4: Mean physical features of 10 best riders per profile

average-height, young cyclists excel in hilly race profiles. Heavier, shorter, and experienced adult cyclists stand out in mountainous race profiles. However, in high mountainous terrain, slightly heavier, taller, less young, and less experienced riders perform well. Finally, in high mountainous races with an uphill finish, the best cyclists are on average heavier, shorter, middle-aged, and less experienced

2.4.4 Motivation

We focused on these analyses in the report because they highlight key aspects of professional cycling performance that can help better understand the factors influencing a cyclist's career. For example, the career span distribution shows that, on average, riders do not remain active in professional cycling for long. Additionally, the bell-shaped distribution suggests that riders must train consistently to stay competitive. The temperature analysis is compelling because it shows how different age groups perform in various conditions, and on average, cyclists tend to achieve better results in lower temperatures. The physical attributes analysis is informative because it reveals how a rider's weight and height influence their performance across different race profiles. Finally, the top 10 riders' physical attributes for each profile provide insights into the qualities required to excel in different race types.

We believe that these analyses can help us in subsequent tasks, such as feature engineering and model building, by providing a better understanding of the relationships between different features and how they influence a cyclist's final position. By identifying the key attributes that contribute to a rider's success, we can create more accurate models.

3 Data transformation

This is a crucial phase because in this part, every missing value was filled using knoweldge discovered by data understanding (distributions and correlations). Every column with at least one missing value were filled with appropriate technique (sampling, derived by other column(s), missing values searched manually on internet etc.).

3.1 Fill missing data

According to results coming from analyzing missing data distributions from data understanding, it was necessary to fill missing data values. First of all, the semantic and statistical correlation between the columns with missing data was analyzed, in fact some values were derived from others, and other values were filled by searching for them on the Internet.

For **cyclist dataset**, filling data was in:

- **Nationality** column = nationality column had only 1 missing values, so it was filled manually (searched by Google).
- **birth year** column = birth year column had few missing values (13), so they were filled manually (searched by Google).
- **weight** and **height** columns = It was find out that these 2 columns are correlated each other, so it was used 2 linear regression, one linear regression trained with weight values for predict height value for fill height values (with weight aviable) and other linear regression for viceversa. For training was used of course all raw where both weight and height was not missing). For filling values where both weight and height was missing, it was used a multivariate gaussian distribution and values where filled with sampling from this distribution.

For **race dataset**, filling data was in:

- **average temperature** column = Considering that was so much missing values (557121), there was not any correlations with other features and was useless (after a discussion), it was decided do delete the feature.
- **cyclist age** column = This column was derived from birth year column (from cyclist dataset) and date column (from race dataset). For each row, cyclist age was calculated doing difference between date year of race and birth year of cyclist. This operation was applied also for substitute aviable values, for be sure that age values was correct.

3.1.1 Imputation of `uci_points`, `climb_total`, `profile` and `delta` null values

We imputed missing values with special filling, since these features are crucial for subsequent analysis and modeling tasks. Consequently, basic imputation methods alone would be insufficient to maintain the foundational data distributions. As stated earlier, some are related linearly, while others are not. The table 1 indicates that adopting more sophisticated imputation techniques was beneficial. It helped to have small fraction of absent and some columns correlations (e.g., `uci_points` with `points`).
The main benefit of this approach is the preservation of the dataset's original size, hence more samplers for classification/prediction tasks. The disadvantages are that the imputed values must align with existing feature patterns to maintain data integrity, which can be guaranteed in some cases only with advance strategies.

3.1.2 Methodology Overview

We tested three main approaches to impute the missing data:

- **Regression method**: It trains 10 regressors with 5-fold cross-validation, compares the average results, and selects the best model by RMSE (Root Mean Square Error) metric.
  We used random search to determine well performing hyperparameters, which remained fixed in subsequent experiments to reduce computation time. The hyperparameters starting ranges were set to typical values found in the literature.

- **Densities method**: It clusters data by shared characteristics, identifies the best-fitting statistical distribution from one of ten possible ones, for each segment (cluster), and uses the identified best-fit distribution to predict missing target values within the respective cluster.
- **Clustering method**: It uses scikit-learn's `KNNImputer` to fill missing values by locating their nearest neighbors with a KNN-based approach and imputing with the neighbors' mean.

3.1.3 Set up

The results analyses will focus only on `climb_total` imputation, but the same methodology was applied to the other features, which gave similar outcomes.
The numerical features were scaled using `MinMaxScaler(feature_range=(-1, 1))`, and the categorical data were encoded as positive integers.
To evaluate with supervised metrics, we excluded empty value rows where was necessary. We split the dataset 80-20 into training and test sets. The training set was used for model fitting and validation, while the test set evaluated performance on unseen data.

3.1.4 Motivation

These three methods address relationships among variables, data distribution, and sample proximity. Regression imputation uses feature relationships, density-based imputation preserves variability, and KNN captures local patterns. Using the right evaluation strategy and metrics for the above methods, we identified suitable values to fill missing data.
We selected these methods because they offer flexibility, they balance complexity, and they can capture a wide range of data patterns.

3.1.5 Multiple Regressor Approach

We decided to use these regressors: `Random Forest`, `Gradient Boosting`, `XGBoost`, `Huber`, `Linear Regression`, `Ridge`, `Lasso`, `KNeighborsRegressor`,
`HistGradientBoostingRegressor` and `VotingRegressor` because they cover a wide spectrum of modeling approaches and have complementary strengths:

- Traditional linear methods (Linear Regression, Ridge, Lasso) are simple and interpretable.
- Tree-based methods (Random Forest, Gradient Boosting, XGBoost, HistGradientBoostingRegressor) can capture non-linear relationships. They are robust to outliers and can handle missing values.
- Non-parametric methods (KNeighborsRegressor) can capture local patterns, which is useful when the data is not globally linear.
- Ensemble methods (VotingRegressor) combine multiple models to improve performance and mitigate overfitting.

We chose these regressors because they span different modeling approaches, handle large datasets efficiently, and need minimal hyperparameter tuning, making them easier to train than more complex methods like neural networks.
After multiple tests, we identified these features— `length`, `points`, `is_tarmac`, `profile`, `position`, `delta`, `season`, `total_teams`, and `adjusted_delta` — as the ones that improve the regressors predicted values. `season` derives from classifying the race date into *spring*, *fall*, *winter*, or *summer*. `total_teams` is how many teams took part. `adjusted_delta` is delta divided by the race length.
A downside of multiple regression is that each model may require distinct data preprocessing steps to meet its underlying assumptions. To simply the process, we used the same data for all regressors, which may have limited the performance of some models.

Results: The first plot in Figure 5 compares the average RMSE across different regressors. The second plot shows how well the best regressor's predictions matched the actual values. The plot reveals that tree-based regressors performed best with an RMSE of 710 meters. The scatter plot displays a positive correlation with significant variance, indicating that the model captures the general trend of longer routes having higher elevations but struggles with precise predictions. The model tends to underestimate higher climb values (>4000m), suggesting that important predictive features may be missing, or that non-linear relationships exist beyond what our tree model can capture.

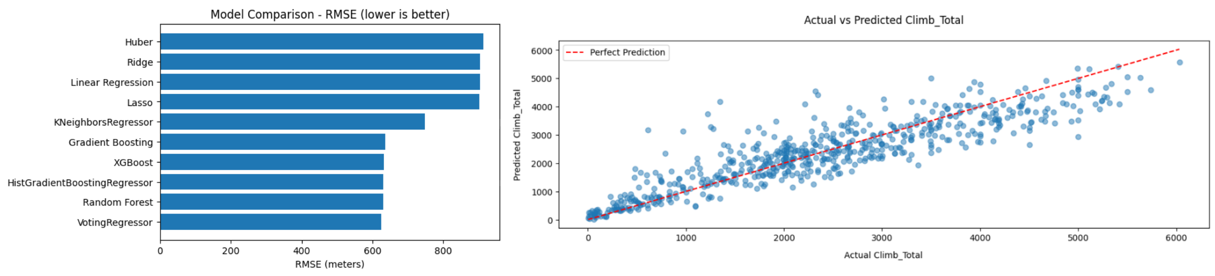

Figure 5: Average RMSE for each regressor and Predicted vs Actual values.

3.1.6 Distribution-based Imputation

Briefly, the density-based approach clusters races with the `KMeans` algorithm by using these features: `length`, `points`, `profile` and `delta`, which demonstrated most effective results. It groups points :

- `length_category` which can be `VS` (Very Short), `S` (Short), `M` (Medium), `L` (Long), or `VL` (Very Long).
- `season_seg` which can be `Spring`, `Summer`, `Fall`, or `Winter`.

It tests 10 probability distributions - `dgamma`, `skewnorm`, `Student's t`, `cauchy`, `laplace`, `levy`, `exponnorm`, `logistic`, `gennorm`, and `genextreme` - to best model data scaled to [-1, 1]. We chose these distributions because they can represent both symmetric and asymmetric data patterns around zero and because they offer flexibility in capturing different tail behaviors and skewness while maintaining computational efficiency.
The distribution selection minimizes the `total_score`, calculated as weighted sum of three metrics:

- **KS statistic** (50%, is a robust metric): Measures goodness of fit using the Kolmogorov-Smirnov test.
- **Moment score** (30%, sensible to outliers): Compares mean, standard deviation, skewness, and kurtosis between original data and samples.
- **Density score** (20%, not always computable): Measures the difference between kernel density estimates of the original and fitted distributions.

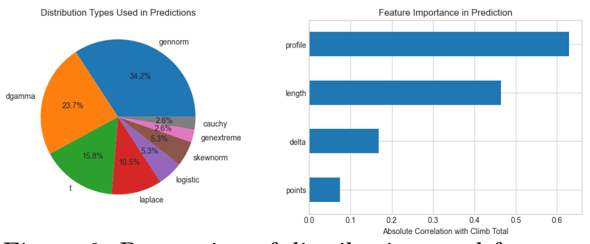

Figure 6: Proportion of distributions and features importance for predictions.

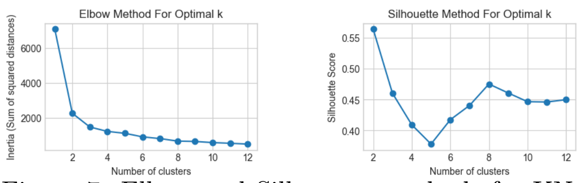

Figure 7: Elbow and Silhouette methods for KNNImputer.

Results: The pie chart in Figure 6 shows that the `gennorm` distribution was selected most frequently, followed by `dgamma` and `Student's t`. The feature importance plot indicates that `profile` and `length` are the most important features for predicting `climb_total`. The left

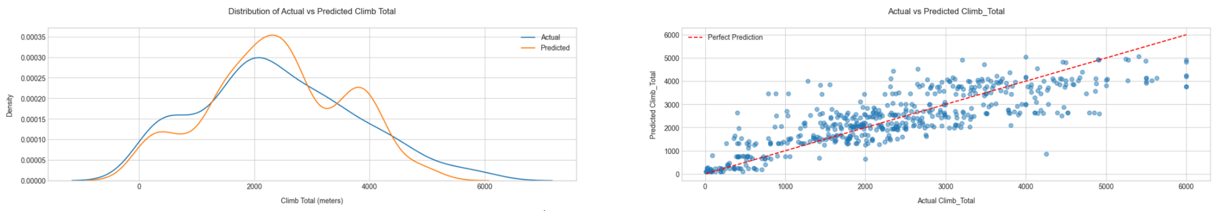

Figure 8: Predicted and Actual density plots and scatter plot.

density plot (Figure 8) shows that an ensemble of densities captures some underlying patterns but not all. The actual values have a wider spread, indicating more variability in real-world data than the model predicts. It overestimates around 2000m, 3800m and underestimates above 4300m, with an average RMSE of around 800m, performing worse than regressors. The density-based approach is reasonably good for predicting `climb_total` empty values but is difficult to interpret, stochastic, and requires appropriate distributions picking.

3.1.7 KNN Imputation

Our simplest and last approach uses scikit-learn's `KNNImputer` with n_neighbors=8 establish by Elbow and Silhouette methods (Figure 7). The plots suggest that k=8 is a local optimal value, which offers a good balance between cluster tightness and separation.
We used this method for the reason that is simple to implement and computationally efficient and can approximate local patterns. We used these features for the imputation: `length`, `points`, `profile`, `position`, `is_tarmac` and `cyclist_age`, since they showed top results. Its main limitations are:

- Sensitive to the scale of input features
- Performance degrades with high-dimensional data
- Cannot extrapolate beyond the range of training data
- Struggles with complex, non-linear relationships
- Might overfit.

Results: The KNN imputer achieved an RMSE of 780m, which worse than the regressors. Looking at the plots (Figure 9), we can say that the model shows good predictive performance

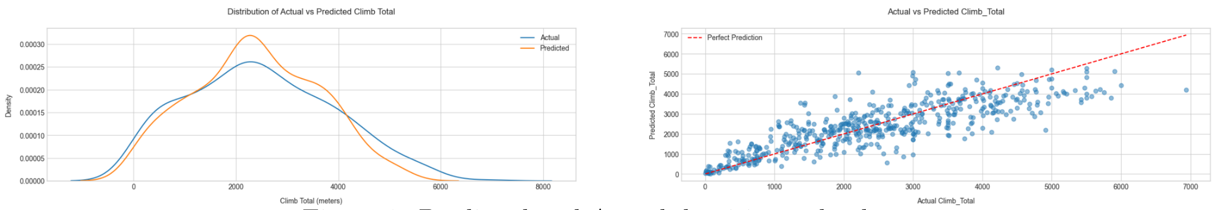

Figure 9: Predicted and Actual densities and values.

with some tendency to overpredict around 2100m values and underpredict at higher values (above 4000m). The distribution plot (left) shows largely overlapping actual and predicted distributions, though the predicted distribution has a slightly higher peak. The scatter plot (right) demonstrates a strong positive correlation between actual and predicted values, with points clustering around the perfect prediction line but showing increasing variance at higher values.

3.1.8 Comparative Analysis

Metric	VotingRegressor	XGBoost	Gradient Boosting	Random Forest	HistGradientBoosting	KNeighborsRegressor	Lasso	Ridge	Linear Regression	Densities	KNN Imputation
RMSE (m)	659	662	661	674	665	789	944	944	945	792	789
R^2 Score	0.78	0.78	0.78	0.77	0.78	0.68	0.55	0.55	0.55	0.66	0.69

Table 1: Comparative analysis of different imputation methods

Based on the Table 1, we might say that the VotingRegressor demonstrates superior performance among the tested imputation methods, achieving both the lowest RMSE (659.18m) and highest R^2 score (0.78). The Density-based and KNN Imputation methods show comparable performance, with KNN having a slightly better R^2 score (0.69 vs 0.66) despite similar RMSE values. We ought to say that density-based RMSE metric is varying due to the stochastic nature of the method. The value in the table is the average of 10 runs. The table suggest that ensemble-based regression approaches may be more effective for handling missing values in this particular dataset. Indeed, during other columns imputations, the regressors showed better performance than the other methods. However, the density-based and KNN imputation methods are still valuable alternatives, especially because they obtained better results than the linear regressors.

3.2 Feature engineering

Based on our previous analyses, we identified several potential features that could enhance our models. We engineered new features to capture the relationships between existing ones and to provide additional information for our outliers detection and clustering tasks. The new features are:

1. `pca_race_points`: We created it by applying PCA to the `points` and `uci_points` features. We got the best results with 1 component, which explained 95.1% of the variance. We used it to reduce the dimensionality of the data while preserving the most important information.
2. `physical_score`: We calculated it by combining the `height`, `weight`, and `cyclist_age` features with the formula:
   `(weight / height * 0.6) + (cyclist_age * 0.4)`.
   It captures the physical attributes of the cyclists.
3. `race_difficulty`: We produced it by combining the `length` and `climb_total` features with the formula:
   `(length * 0.325 + climb_total * 0.525 + is_tarmac * 0.15) / 100`.
   It encapsulates the overall race's challenge level.
4. `combined_score_per_km`: We determined it by the formula:
   `points_per_km * 0.6 + startlist_quality * 0.4`.
   This attribute accounts for both the achievements of the cyclists and the quality of the start list.
5. `consistency_score`: We derived it by computing the mean standard deviation of the `delta`, `position`, and `pca_race_points` features. This property captures the consistency of the cyclists' performance across races.
6. `race_performance_score`: As the last engineered feature, we generated it by the formula:
   `position_modified * 0.7 + points_per_km * 0.3`.
   It provides an overall assessment of the race' performance

We chose these features because they help reduce dimensionality, while preserving the most important information.

3.3 Outliers detection

For outliers detection, we applied three distinct algorithms: Local Outlier Factor (LOF), Isolation Forest (IF), and One-Class SVM (OCSVM). We chose these algorithms for their complementary strengths in handling different types of outliers:

1. **Local Outlier Factor**: We selected LOF for its effectiveness in detecting outliers in datasets with varying densities. It compares the local density of a point to the local densities of its neighbors, making it particularly useful for our dataset where race performances and cyclist attributes can form clusters of different densities.

2. **Isolation Forest**: We utilized IF for its ability to handle high-dimensional data efficiently and its effectiveness in detecting global outliers.
3. **One-Class SVM**: We employed OCSVM using GPU acceleration for its robustness in creating a decision boundary that encompasses "normal" data points. This method was particularly useful for identifying outliers in the high-dimensional feature space.

The combination of these three methods provided comprehensive coverage for detecting different types of outliers in our dataset. We categorized the detected outliers into two main groups:

- **Race Outliers**: Races were classified as outliers if they were flagged by at least one detection methods and had participation numbers above the 50th percentile of all races. E.g. if the same race was consistently identified as an outlier more than 50% out of all possibilities, we considered it an outlier.
- **Rider Outliers**: Individual cyclists were marked as outliers when they were flagged by at least one detection method and had participated in more than half of races considered as outliers. For example, if a cyclist is listed as participating in more than 50% of his total races as outliers, we considered him an outlier.

3.3.1 Motivation

The ensemble approach of combining LOF, IF, and OCSVM provided robust outlier detection by leveraging each method's strengths while mitigating their individual weaknesses. We opted for these methods rather than alternatives because our dataset contains continues and one categorical features with varying densities, making it more effective to use a combination of methods with different strengths. These tools provide clear anomaly scores and decision boundaries, making it easier to interpret the results and identify the most significant outliers. In addition, is possible to utilize hyperparameters tuning to improve the performance of the algorithms. At last, the trio offers complementary perspective on the data, allowing us to capture a wide range of outliers.

3.3.2 Set up

Following our experiments, we chose to use these columns in the end: `nationality`, `pca_race_points`, `physical_score`, `race_difficulty`, `combined_score_per_km`, `consistency_score` and `race_performance_score` as the most effective for outlier detection. We normalized the numerical features and encoded the categorical data as positive integers. To evaluate model performance without ground truth outlier labels, we developed a scoring system for outlier detection based on established heuristics described below.

3.3.3 Tuning and Scoring

Local Outlier Factor: We randomly selected 30 hyperparameters combinations: `n_neighbors` $\sim$ *randint*(15,50), `contamination` $\sim$ *random.choice*([0.01, 0.05, 0.1, 0.15, 0.2]), `leaf_size` $\sim$ *randint*(20-50), `p` $\sim$ *random.choice*([1, 2]) and `metric` $\sim$ *random.choice*(['minkowski', 'manhattan', 'euclidean']) possible ones. These values were selected based on our dataset characteristics: `n_neighbors` range captures local density variations in our feature space, while contamination levels reflect expected outlier percentages in cycling performance data. The `leaf_size` range optimizes the algorithm's performance for our dataset size.
The best LOF model is selected by maximizing a weighted combination of three key components:

1. **Separation Component (35%):** it measures how well outliers are distinguished from normal points by comparing their mean scores, normalized by the overall standard deviation. A higher score indicates better separation between outlier and normal populations.
2. **Density Component (40%):** it evaluates how different the local density is around outliers compared to the overall dataset density. This component receives the highest weight (0.40) because local density variations are fundamental to LOF's effectiveness.

3. **Reachability Component (25%):** it assesses the reachability patterns of outliers versus the overall dataset. It helps identify points that are significantly different from their local neighborhood in terms of distance relationships.

Isolation Forest: We tested 20 configurations with `n_estimators` $\sim$ *randint*(100, 300), `contamination` $\sim \mathcal{U}(0.001, 1.0)$, and `max_samples` $\sim \mathcal{U}(0.1, 0.4)$. The number of trees (`n_estimators`) and `max_samples` were chosen to balance computational cost and model robustness, while the contamination level was set to reflect the expected proportion of outliers in the dataset. Our algorithm selection criteria maximizes a composite score based on several key metrics:

1. **Distribution Metrics (45%):** considers score STD and skewness to evaluate the spread and symmetry of anomaly scores.
2. **Density Analysis (35%):** determines the gaps between extreme percentiles and Separation between normal and anomalous points.
3. **Control Mechanisms (20%):** includes outlier penalty and extreme value control limitation mechanism to prevent excessive flagging and manage extreme score distributions.

The final score undergoes logistic transformation and contamination penalty adjustment, ensuring balanced outlier detection. Models with excessive contamination receive exponential penalties, favoring conservative outlier identification.
One-Class SVM: We explored 20 hyperparameters combinations with `nu` $\sim \mathcal{U}(0.005, 0.015)$, `gamma` sampled logarithmically in the range $[10^{-5}, 1]$ using $\gamma = e^x$ where $x \sim \mathcal{U}(\ln(10^{-5}), \ln(1))$ to ensure better coverage of different scales, and `kernel` $\sim$ *random.choice*(['rbf', 'sigmoid']). The `nu` range was selected to capture the expected proportion of outliers in the dataset, while the `gamma` range was chosen to optimize the model's performance for our feature space. The kernels were selected based on speed and quality of the results. The best OCSVM model is chosen based on a weighted combination of multiple components equal to the IF model. The scores undergo logistic transformation followed by a `nu` penalty adjustment to ensure balanced outlier detection across the dataset.
The score components of each detector were carefully selected and tested over and over to ensure that they favor models that can detect a wide range of outliers.
Results: The leftmost plot (LOF) in Figure 10 identifies outliers at the periphery and in low-densities regions, mostly. Most outliers are found on the opposite side of the plot. The middle plot (IF) reveals more global distribution of outliers, regardless of local density. The rightmost plot (OCSVM) demonstrates a more structured decision boundary, with outliers clearly separated in specific regions of the feature space.

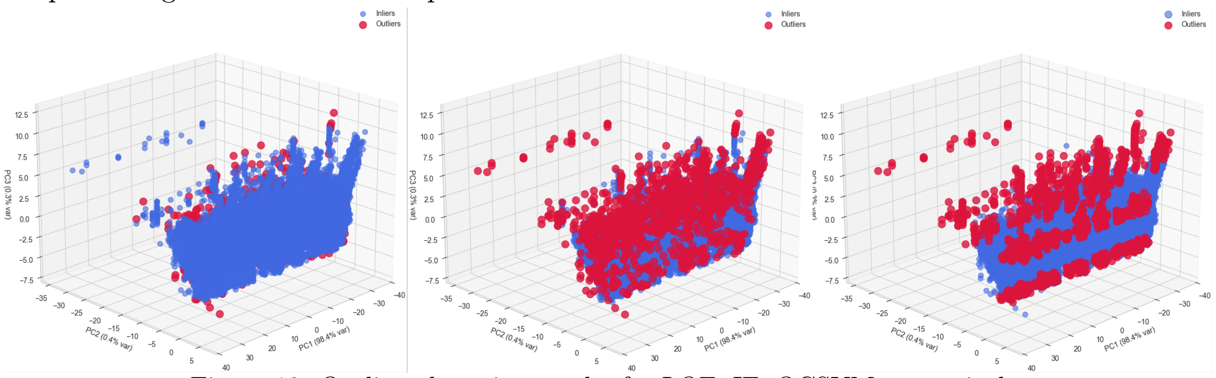

Figure 10: Outliers detection results for LOF, IF, OCSVM respectively.

3.4 Outliers analysis

We identified 26 races outliers and 493 cyclists outliers. The Venn diagram in Figure 13 shows the distribution of row outliers across the three detection methods. LOF has detected the most outliers, followed by IF and OCSVM because of its ability to capture local density variations, which cycling performance data often exhibits. In addition, LOF score metric doesn't penalize outliers as much as IF and OCSVM, which may explain the higher number of detected outliers. IF and OCSVM tend to behave more conservatively.

Figure 11 displays the feature importance for the outlier detection. We can see that for races (left plot):

- `length` and `climb_total` are the most crucial features for detecting outliers, which is expected.
- `profile` and `startlist_quality` contribute less to the detection, but they still hold some importance.

For cyclists (right plot):

- `cyclist_age` and `delta` are fundamental for detecting outliers, which is consistent with our previous analyses.
- `points` and `uci_points` have less importance, because they are more volatile metrics that can fluctuate significantly based on short-term performance and race participation, making them less reliable indicators of systematic outliers in cyclist performance patterns.
- `height` and `weight` surprisingly have very low importance, especially height, which can be explained by the relatively narrow range of these physiological characteristics among professional cyclists. While these metrics influence performance, their variance within the elite cycling population is limited, making them less discriminative for outlier detection compared to performance-based metrics like `delta` and age-related factors.

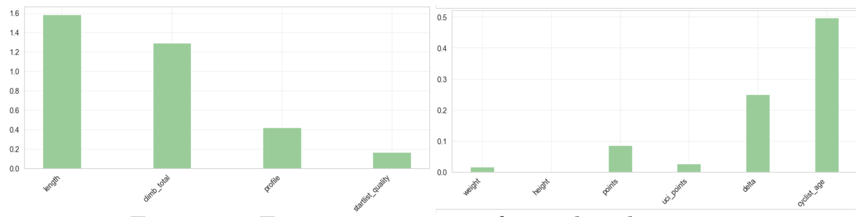

Figure 11: Feature importance for outlier detection.

4 Data clustering

4.1 K-means

For this clustering, more tests were done, modifying the value of K and observing how the SSE loss value moved (Figure 12). Based on the experiments done with K-means, we can observe that as the value of K increases, the SSE loss value decreases. This means that the dataset does not have sufficiently dense regions to be able to define well-defined clusters. Range tried of K values start from 2 and finish to 20, so in this way is proved a large range of values.

4.2 DBSCAN clustering

This section provides information on our implementation and analysis of DBSCAN (Density-Based Spatial Clustering of Applications with Noise), a clustering algorithm used to identify dense regions in the data and classify points not belonging to any cluster as noise. DBSCAN does not require the number of clusters to be specified in advance. The two critical parameters of DBSCAN, `eps` (radius of the neighbourhood) and `min_samples` (minimum number of points in a neighbourhood to define a cluster), have a significant impact on the quality of clustering. By adjusting these parameters, cluster formation and noise detection can be refined.

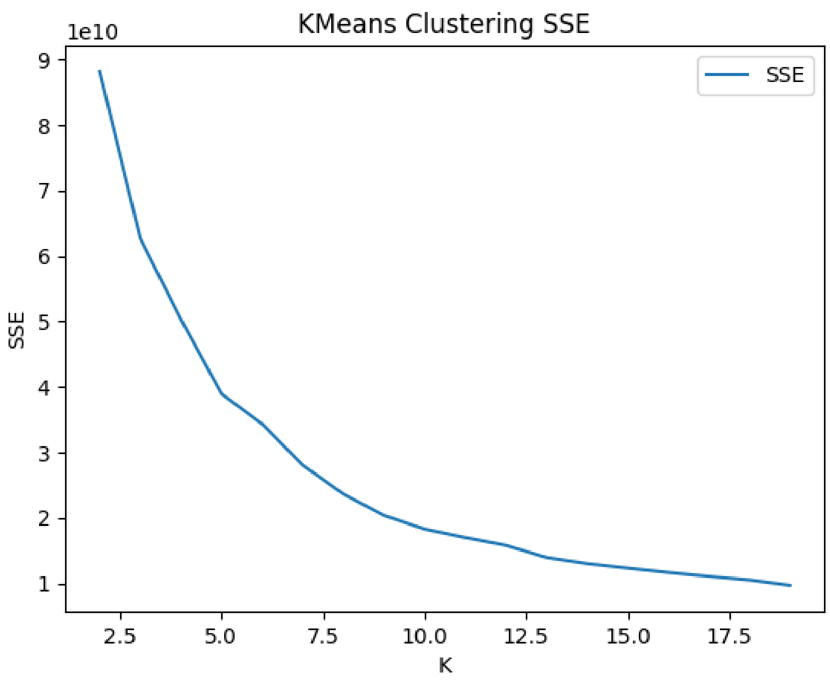

Figure 12: Average SSE by K value.

Data Preprocessing The data were scaled using `StandardScaler`, which standardises features so that they have zero mean and unit variance. Scaling is crucial in DBSCAN, as distances between points are calculated during clustering. For large datasets, scaling helps prevent features with higher number ranges from dominating the clustering process. To avoid computational problems with large datasets, a random sample of 50,000 points was taken to perform the clustering. Since the dataset does not have a structure that requires a particular metric we used the euclidean distance. For the choice of min-samples since our dataset is 6-dimensional appropriate values can be from 7 to 12. We tried only a few values due to the high computational cost that would have required a more complete choice.

Optimization of Parameters In the analysis, different combinations of the following parameters were tested.

- `min_samples_values`: we chose some values between 7 and 12 due to the dimensionality of the dataset.
- **eps**: K-distance graphs were generated for 'k' values (7, 8, 10, 11, 12) corresponding to potential 'min_samples.' The elbow point consistently suggested 'eps' values between 0.9 and 1.2 as optimal for distinguishing clusters.

We try different combination, DBSCAN was run and the silhouette score calculated to assess the quality of the clusters.

Results The optimal combination of parameters was identified as:

```
'eps': 1.2, 'min_samples': 10, 'Number of clusters': 4, 'Silhouette Score':
        0.683, 'Total points': 50,000, 'Total noise points':283.
```

The figure 14 shows the resulting clustering visualized with PCA with clusters differentiated by color.

Finally we calculate the statistical summary for each cluster:

- `Cluster 0`: encapsulates the average rider population.
- `Clusters 2 and 3`: capture specialized subgroups, possibly elite performers requiring further domain analysis.

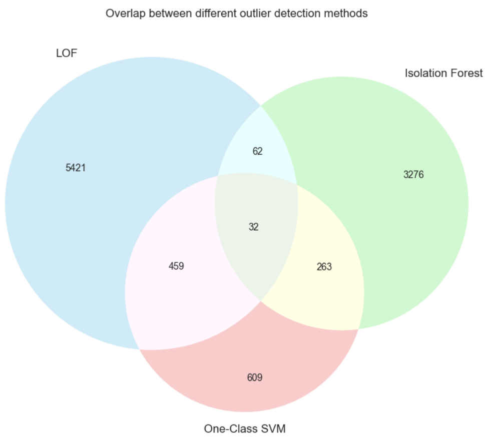

Figure 13: Distribution of outliers across detection methods.

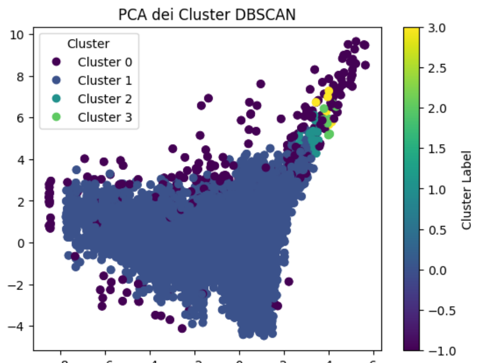

Figure 14: Clustering results with DBSCAN

- `Minimal noise`: suggests robust parameter tuning.

Conclusions DBSCAN effectively clustered dense, high-dimensional data with minimal noise. Limitations:

- sensitive to parameter variations
- Computational cost.

4.3 Hierarchical clustering

Hierarchical clustering creates clusters progressively, allowing evaluation at different levels of granularity. Using the silhouette score as a measure of clustering quality, we aimed to gain insight into the structure of the dataset and identify the most suitable cluster counting and linking method.

Data Preprocessing The data was scaled using `StandardScaler`, which standardizes features to have zero mean and unit variance. This step ensures that features are comparable, since hierarchical clustering relies on distance metrics (e.g., Euclidean distance) that are sensitive to differences in scale. To address the computational challenges associated with large datasets, a random sample of 50,000 data points was drawn for clustering analysis.

Hyperparameter Tuning Hierarchical clustering was performed using four linkage methods: `ward`, `complete`, `average`, and `single`. The dataset was scaled and further reduced to two dimensions using PCA for ease of visualization. The silhouette score was calculated for cluster counts between $t = 2$ and $t = 14$ to assess the quality of clustering.

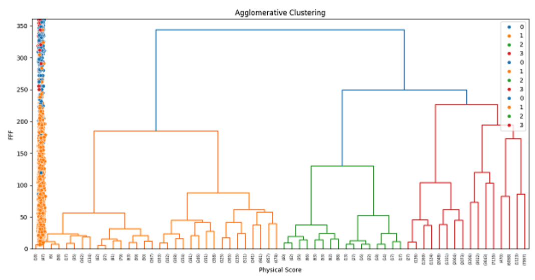

Figure 15: Dendogram

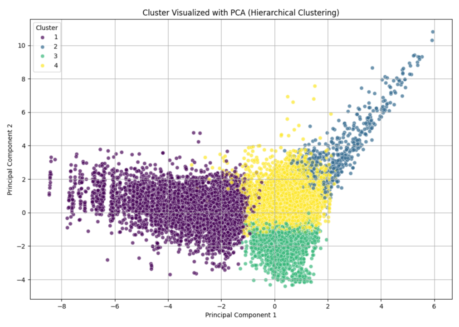

Figure 16: Cluster Visualized with PCA (Hierarchical Clustering)

Results

- The Ward method outperformed other linking methods in forming compact and meaningful clusters, as confirmed by higher silhouette scores and better visual separations. The best

silhouette score was 0.637 at $t = 2$. Figure 15 shows the resulting dendogram, and Figure 16 shows the resulting clustering visualized with PCA with clusters differentiated by color.
- The silhouette score peaked at $t = 2$ and decreased as more clusters were added, suggesting overfitting or excessive granularity with higher cluster counts.
- **Complete and average methods:** These methods demonstrated less consistency and often produced a dominant cluster, especially at lower cluster counts.
- **Single method:** consistently produced a large cluster, resulting in reduced clustering quality and poor silhouette scores.

Finally we calculate the statistical summary for each cluster:

- `Cluster 2`: is clearly the most interesting cluster, as it represents elite cyclists with exceptionally high scores for both 'pca_race_points' and 'race_performance_score'. The small number of points (179) supports this interpretation.
- `Cluster 3`: includes competitive cyclists, but less exclusive than Cluster 2. However, their good overall performance makes them relevant for the analysis.
- `Cluster 1 and Cluster 0`: represent the worst performing groups, with Cluster 1 appearing to be the weakest overall.
- The clusters are unbalanced. Cluster 2 is very small compared to the others, reflecting a clear hierarchical structure in the data.

Conclusion The choice of linkage method significantly affected the clustering results and visual interpretations, highlighting the importance of selecting the most appropriate approach. Hierarchical clustering with the Ward linkage method effectively identified significant patterns in the dataset. It revealed distinct clusters and provided insight into the underlying structure of the dataset. However, were observed some limitations: scalability and hyperparameter sensitivity.

5 Predictive models

The purpose of this classification analysis is to predict whether a cyclist finished in the top 20 positions of a race, represented as a binary outcome (1 for top 20, 0 for other positions). In order to train the set of models explored during model selection on the data, the dataset on which the data was trained was extracted:

1. The race dataset was taken after the data transformation was processed.
2. The label label (the target label for training) was calculated by extracting it from the position label: if the position value was >= 20, then label was 0, otherwise label was 1.
3. The labels: **_url**, **name**, **date**, **position**, **cyclist**, **delta**, **race_year** and **position** have been removed.

During training, all racing rows from 2022 onwards are considered in the test set, the others in the training set. The input features considered are: **points**, **uci_points**, **length**, **climb_total**, **profile**, **startlist_quality**, **cyclist_age** and **delta**. Note that the classes are unbalanced, in fact both in the training set and in the test set class 1 has a distribution of approximately 14% and class 0 has a distribution of approximately 86%.

5.1 Linear classification

A linear classifier was used in model selection to see if the problem could be linearly separable. This hypothesis derives from the fact that there may be a hyper-plane that divides the points based on position. The optimization algorithm was created with the sklearn library, as it allowed the model to be trained very quickly. The training reported metrics with good training values: The training reported metrics with good training values: **Accuracy**: **train** value 92.6%, **validation** value 92.6% and **test** 91.7%. **Log-Loss**: **train** value 0.2230, **validation** value 0.2250 and **test** 0.2664. **AUC**: **train** value 89.8%, **validation** value 89.6% and **test** 83.0%.

5.1.1 Analysis pattern

Observing the results obtained by the SHAP tool, it can be concluded that the importance of the most important features are:

- the points feature has a great influence on predicting the class with a positive label, this is explained by the fact that between points and position there is a correlation of -31%.
- the feature uci_points has a great influence on predicting the class with a negative label and this justifies why there is a correlation of -16% between uci_points and position.
- the delta feature has a little influence on predicting the class with a positive label and this can be confirmed with the correlation between delta and position of -24%.

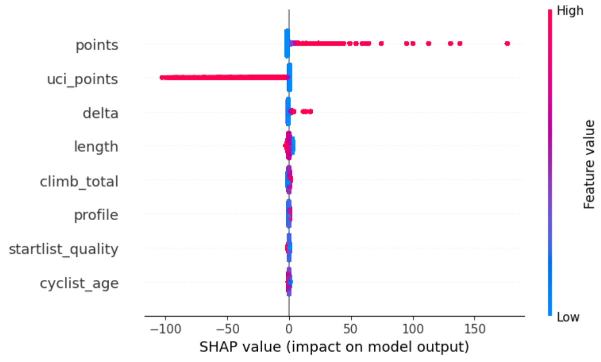

Figure 17: SHAP linear classification

Overall, the features contribute to recognizing the label 1 little more than 0, and this is explained that model considering more 1 label, because it is in minor distribution.

5.2 Esembling linear classification

An example stacking composed of 4 linear classifiers was used: **LogisticRegression**, **RidgeClassifier**, **SGDClassifier** and **PassiveAggressiveClassifier**, and a **LogisticRegression meta classifier**. This structure was chosen to see the behavior of this example by putting multiple classifiers to collaborate with each other, and to see if the important features are the same for each classifier. Also for this model the optimization algorithm was created with the sklearn library for very fast training times. The training reported metrics with good training values: **Accuracy**: **train** value 92.8%, **validation** value 92.8% and **test** 91.8%. **Log-Loss**: **train** value 0.2253, **validation** value 0.2269 and **test** 0.2576, **AUC**: **train** value 89.7%, **validation** value 89.5% and **test** 84%.

5.2.1 Analysis pattern

The analysis of the patterns captured for this model must be done analyzing classifier by classifier. From the analyzes it can be seen that for the meta classifier, the most important classifier is the **RidgeClassifier**, with a contribution mainly on the positive class, followed by the **LogisticRegression** with little more contributions to the positive class, followed by **PassiveAggressiveClassifier**, with small positive class contribution, and **SGDClassifier**, with very small contribution more on negative label.

However, LogisticRegression and RidgeClassifier considers same pattern like important, as like as SDGClassifier and PassiveAggressiveClassifier do.

For the **LogisticRegression** model and **RidgeClassifier** models:

- **points** gives a contribution on the positive class, while **uci_points** gives an opposite contribution, i.e. on the negative class - **delta** gives a little stronger contribution on the positive class

The **SGDClassifier** and **PassiveAggressiveClassifier** classifiers have learned the patterns by giving weight to the same features, and the most important ones are: **delta**, **points** and

uci_points, which gives a powerful contribution on the positive class and less powerful on the positive class.

Looking at the general overview, it can be said that the most relevant features, which generally influence the entire ensemble, are: delta, with a correlation with position of -24%, points, with a correlation with position of -31% and uci_points, correlated with position of -16%. These percentages justify and explain why even in an ensemble of linear classifiers, as in the case of a simple linear classifier, these 3 features are chosen.

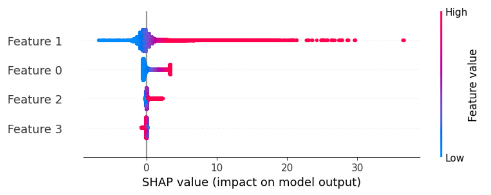

Figure 18: SHAP Logistic Regression classifier

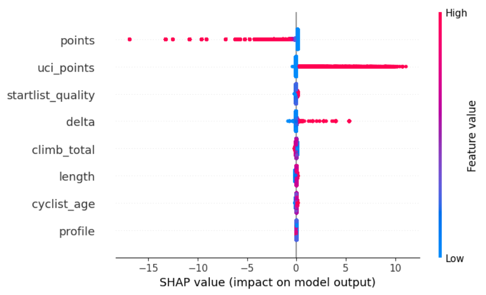

Figure 19: SHAP Ridge classifier

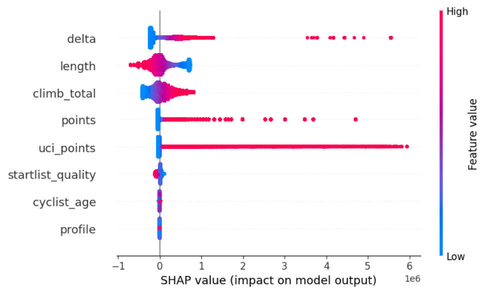

Figure 20: SHAP SGD classifier

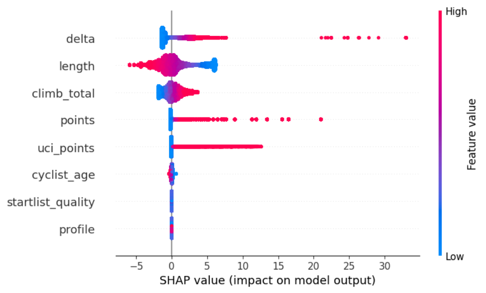

Figure 21: SHAP Passive Aggressive classifier

5.3 Neural network

We selected a classifier based on Neural Network because of its ability to learn complex non-linear patterns in the data. The drawback of this model is that it requires substantial computational resources to train and can become prone to overfitting if not carefully regularized or tuned. Furthermore, its intricate architecture complicates interpretability relative to linear approaches, making it harder to understand the inner decision-making process. The major issue we faced was a pronounced class imbalance, with only 15.41% of the samples in class 1 and 84.59% in class 0, which can skew predictions toward the majority class. To address the inherent class imbalance in our dataset, we employed SMOTE (Synthetic Minority Over-sampling Technique). This technique generates synthetic samples for the minority class by interpolating between existing instances and their k-nearest neighbors. Subsequently, we performed feature engineering by combining linearly the `points` and `uci_points` features into a consolidated metric representing overall performance to introduce the categorical performance feature. This transformation was necessary to avoid the unfortunate (uci)points data distribution. We implemented an ordinal categorization system to classify the merged metrics into 7 bins: *None*(0), *Very Low*(1-5), *Low*(6-10), *Medium*(11-20), *High*(21-50), *Very High* (51-100), *Exceptional* (>100). We facilitated NN learning/predicting by transforming these categorical variables through one-hot encoding. To improve learning efficiency and mitigate the impact of extreme values, we standardized the features using `StandardScaler`, and we applied outlier clamping using the Interquartile Range (IQR) method. Scaled values were clamped to the range $[Q_1 - 2.0 \cdot IQR, Q_3 + 2.0 \cdot IQR]$.
We trained each model for a maximum of 20 epochs, selecting the optimal model configuration

through 5-fold cross-validation. The model selection was based on a custom evaluation metric, `combined_metrics`, which aggregates multiple performance indicators into a single score. This metric is defined as: `combined_metrics := 0.5*F1 + 0.5*AUROC - 0.5*val_loss`. For hyperparameter tuning, we adopted advance search algorithm such as `PopulationBasedTraining` to explore the hyperparameter space efficiently. We systematically explored over 100 hyperparameter combinations during the project developing process. Only the final, most significant results are showcased to maintain brevity. The Table 2 presents a comparison of different NN configurations and their metrics. The experiment explored various activation functions, across different architecture choices and hyperparameters settings. The last row of the table shows the best configuration found, with a combined metrics of 0.694572, a validation loss of 0.300863 and a validation F1 score of 0.750685. We see that, generally, configurations with a batch size of 128 outperformed their counterparts. This is particularly evident comparing `Tanh` with batch sizes 64 and 128. The results indicate that deeper models tend to achieve better metrics, but the dropout rate should be carefully tuned to avoid overfitting.

act_fn	batch_size	lr	weight_decay	dropout	hidden_dims	combined metrics	val_loss	val_f1
Tanhshrink	128	0.00073585	3.44516e-05	0.116877	[64, 32]	0.668594	0.331595	0.746213
LeakyReLU	32	0.00340419	0.000329709	0.169617	[64, 32]	0.675667	0.326251	0.751268
Tanh	64	0.00042957	2.05985e-05	0.161241	[16, 8]	0.624142	0.369298	0.716978
Tanhshrink	32	0.00413141	5.56076e-05	0.060391	[16, 8]	0.656856	0.345965	0.743663
Tanhshrink	128	2.35645e-05	0.00011351	0.116952	[32, 96, 16]	0.657043	0.346217	0.743549
Tanh	128	0.00595542	5.43163e-05	0.170031	[64]	0.634499	0.349344	0.711443
Tanh	64	2.93174e-05	1.66244e-05	0.095803	[64, 32]	0.614435	0.372533	0.705451
Tanhshrink	64	2.2432e-05	2.81081e-05	0.234155	[32]	0.60883	0.387657	0.715794
SELU	128	3.86703e-05	0.00071524	0.059431	[32, 96, 16]	0.663192	0.331887	0.734896
LeakyReLU	128	0.0095655	0.00010954	0.778031	[32, 96, 16]	0.694572	0.300863	0.750685

Table 2: Comparison of Neural Network Hyperparameters and Performance Metrics

We can see that the model correctly classifies approximately 93.3% of the test samples. The F1 scores might reflect the class imbalance, but the AUROC score of 0.826 indicates that the model maintains good discriminative ability between the two classes. These results, when considered alongside the hyperparameter optimization findings from Table 2, validate our architectural and training choices. The model demonstrates strong predictive performance, with a good balance between accuracy and generalization, despite imbalance issue and the complexity of the dataset.

Metric	**Value**
Test Accuracy	0.941
Test F1 Score	0.760
Test AUROC	0.839

Figure 22: Final Model Performance on Test Set

5.3.1 Analysis pattern

The global feature importance plot, Figure 23, displays that one-hot encoded feature `points_category_None` is by far the most influential feature, followed by `delta`. It is reasonable to see that those are crucial, as they are the most correlated with the target label (Figure 1). In the SHAP summary plot (Figure 24), `points_category_None` strongly influences classification: lower values increase top-20 finish probability, while higher values decrease it. The `delta` is the opposite, with higher values leading to a lower probability of being classified as a top 20 finisher. The `length` distribution might suggest a complex, non-linear relationship with the target label. Notably, both `startlist_quality` and `cyclist_age` show minimal impact on the model's predictions, with SHAP values tightly distributed around zero. This is particularly interesting for `cyclist_age`, as its low influence likely stems from our previous preprocessing step where we clamped extreme values. The clamping, may have inadvertently reduced this feature's discriminative power. It seems counterintuitive that `climb_total` and `profile` have minimal impact on the model's predictions, given its typical importance in cycling. One problematic aspect might be that the

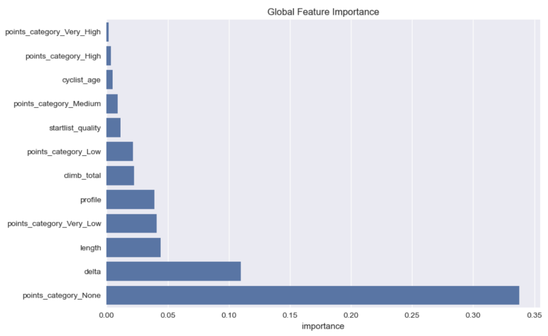

Figure 23: NN Global Feature Importance

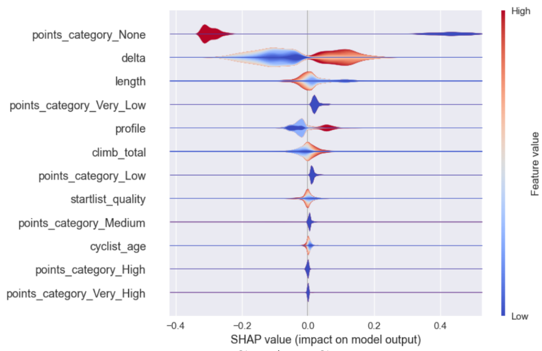

Figure 24: NN SHAP Summary Plot

model shows high reliance on categorical features over continuous ones, which could indicate that the model is not fully capturing the underlying relationships in the data. In addition, it might indicate overfitting. Overall, the model's performance is strong, but further investigation is needed to understand the model's reliance on certain features and its potential limitations.

For the methods described in the following sections (SVM, Random Forest and Decision Tree), we selected a sample equal to 20% of the original dataset, the same for all methods, to make the calculations manageable.

5.4 Support vector machine

SVM models are well suited for binary classification, especially with high-dimensional feature spaces. These models excel at separating classes by finding a hyperplane that maximizes the margin between them, particularly when the data has a non-linear decision boundary.

Feature Scaling

The data was pre-processed with `StandardScaler` to standardize all features, which is essential for SVMs. This helps ensure that no single feature dominates the model's decision making, especially when kernel-based methods are applied. SVM is sensitive to the scale of the features, and feature scaling is a critical step when working with SVM.

Hyperparameter Selection

To optimize the SVM model, I ran a `RandomizedSearchCV` using the following hyperparameters:

- **C (Regularization)**: Different values of **C** were explored to tune the strength of regularization. A higher value of C allows the model to better fit the training data, whereas smaller values allow for a smoother decision boundary that could generalize better on unseen data.
- **Gamma**: Different values of `gamma` were tested, including `auto` and `scale` settings. Gamma controls the influence of each data point on the decision boundary.
- **Kernel**: `rbf`, `poly`, and `sigmoid` kernels often provide higher predictive power for non-linear problems. However, their computational cost is significantly higher. Given these constraints, the linear kernel was chosen to ensure feasibility without significantly compromising model quality.

Model Performance

`RandomizedSearchCV` was run with 3-level cross-validation to identify optimal hyperparameters based on the F1 score. This allows for efficient tuning while also taking into account model performance on different subsets of data. The SVM model performed well obtaining the results in 4, with an accuracy of 0.89 and a ROCAUC of 0.8347, indicating a good ability to distinguish classes. However, the minority class (1) has a precision of 0.61 and a recall of 0.51, indicating that the model struggles to correctly identify it. The major class (0) performs excellently with high precision and recall (0.93 and 0.95).

In summary, the model performs well overall, but could improve the detection of observations of class 1, probably due to a class imbalance. To improve, you can balance the classes or optimize the classification threshold.

Limitations

- **Class Imbalance**: Class imbalance is evident in the model performance, with significantly higher recall for Class 1 than for Class 0. Methods such as oversampling, undersampling, or weighting could help improve recall for Class 0.
- **Computation Constraints**: The choice of the **linear kernel** was driven by the high computational cost of alternative kernels that would typically be preferred for problems where the decision boundary is expected to be non-linear.
- **Hyperparameter Space**: While `RandomizedSearchCV` allowed for exploration, the number of tested hyperparameter combinations was limited (5 combinations). A more exhaustive search (e.g., expanding the range of possible values for **C** and **gamma**) could potentially improve results.

5.4.1 Analysis pattern

We conduct pattern analysis with LIME (Local Interpretable Model-Agnostic Explanations), which aims to provide explanations for the pattern(s) from the previous step.

Feature	points	uci_points	length	delta	profile	startlist_quality	climb_total	cyclist_age
Value	-0.24	-0.15	0.37	-0.45	1.01	-0.84	0.09	-0.90

Table 3: Feature's contribution and values

Predictions detected with 'LIME' show that the rider has a 95

5.5 Decision tree

The **Decision Tree Classifier** is a great algorithm for classification, as it can handle both numerical and categorical data and offers a high level of interpretability.

Since the dataset might be unbalanced (with fewer cyclist instances in the top 20), class balancing was handled with `class_weight='balanced'`.

Model Selection and Hyperparameter Optimization We used a grid search to optimize its hyperparameters, considering different values for:

- **criterion**: the function used to measure the quality of a split. Both `'gini'` and `'entropy'` were evaluated.
- **max_depth**: a limit on the maximum depth of the tree. We tested values between 5 and 10, as deeper trees can capture more complexity but can also lead to overfitting.
- **min_samples_split**: The minimum number of samples required to split an internal node, tested between 5 and 15.
- **min_samples_leaf**: The minimum number of samples required to be in a leaf node, tested between 1 and 5.
- **class_weight**: Handled class imbalance by assigning different weights to classes (tested with `'balanced'` and `None`).

The best performing model produced the following hyperparameters:

```
'class_weight': None, 'criterion': 'gini', 'max_depth': 10,
      'min_samples_leaf': 1, 'min_samples_split': 10.
```

Performance Evaluation After training, the **accuracy** of the model on the **training set** was approximately 96.4%, and the accuracy on the **test set** was 94.5%. These values suggest that the model generalizes well to unseen data.

The **ROC-AUC** score on the test set was 0.89, indicating that the model does a good job of distinguishing between the two classes.

The (weighted) F1 score of 0.94 confirms a good balance between precision and recall in both classes. In particular, while the precision and F1 score for class 1 (the top 20 ranked) are high, the precision for class 0 (not the top 20) is a bit lower, but still significant.

5.5.1 Analysis model

To better understand the model's decision-making process, SHAP values were used to interpret the importance of each feature. Two SHAP explanation methods were explored:

Interventional explanations This approach involves perturbing the data and analyzing how predictions change based on individual features.

Distributive Explanations This approach uses path-dependent perturbation of the tree, assessing the importance of features through the structural properties of the decision tree.

Both explanation techniques yielded insights into which features were most influential. **Swarm Plots** provide a visual representation of the relative impact of each feature on the model's predictions. Features with the highest importance and variance in their contribution were highlighted.

5.5.2 Analysis pattern

To better understand the model's decision-making process, SHAP values were used to interpret the importance of each feature. Two SHAP explanation methods were explored: **Interventional Explanations** and **Distributive Explanations**

Both explanation techniques yielded insights into which features were most influential. **Swarm Plots** provide a visual representation of the relative impact of each feature on the model's predictions. Features with the highest importance and variance in their contribution were highlighted. The two plots in figures 25 and 26 are similar, this suggests that the model could be quite robust and that it does not have a strong interaction between the features and the predictions. From these graphs we can observe the importance of the features, in particular, 'points' is the feature that impacts the result of the classifier the most, followed by 'delta', while 'cyclist_age' is the least impactful feature. Therefore significant changes in 'points' and 'delta' will have an important effect on the prediction. Furthermore we can see for each feature how the different values influence the prediction of the class. For example: "points" has a low value in the negative points and a high value in the positive points. This means that an increase in the value of "points" is associated with a greater probability of obtaining a positive result for the target class. For the feature 'length' the opposite is true.

During the initial analysis, we detected a moderate correlation between position and points (-0.31) and a weaker negative correlation with delta (-0.24). The model confirmed these features as the most important for its predictions. Furthermore, the model also identified 'climb_total' and 'length' as relevant, despite the preliminary correlations with 'position' were very low, almost negligible (-0.04 and -0.01 respectively), suggesting the presence of complex non-linear patterns.

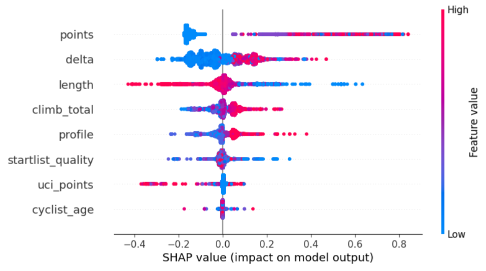

Figure 25: SHAP interventional (Decision tree)

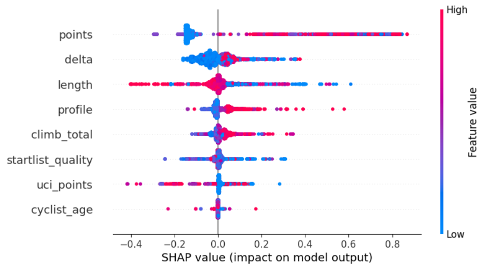

Figure 26: SHAP distributional(Decision tree)

5.6 Random forest

The Random Forest Classifier is well-suited for tabular data with complex feature interactions due to its ensemble nature and robustness against overfitting.

RandomizedSearchCV was selected over *GridSearchCV* for hyperparameter tuning due to its ability to efficiently explore a wide range of hyperparameter settings while reducing computational cost and time.

Model Selection and Hyperparameter Optimization The following hyperparameters were tuned using *RandomizedSearchCV* with 20 iterations:

- `n_estimators`: Number of trees in the forest {50, 100, 200}.
- `max_depth`: Maximum depth of a tree {4, 6, 8, None}.
- `max_features`: Number of features to consider when searching for the best split {"sqrt", "log2", None}.
- `min_samples_split`: Minimum number of samples required to split an internal node {2, 10, 20}.
- `min_samples_leaf`: Minimum number of samples required for a leaf node {1, 5, 10}.
- `class_weight`: Handling of class imbalance {None, "balanced", user-defined weights}.

The model was evaluated using the F1-weighted score to account for imbalanced class distribution, considering both precision and recall.

The best performing model produced the following hyperparameters:

`n_estimators: 100, max_depth: None, max_features: None, min_samples_split: 2, min_samples_leaf: 5, class_weight: None`.

The optimal hyperparameters from RandomizedSearchCV were used to train the model. The model used: A Random Forest classifier with

`n_estimators=30` (changed to default for illustration), `criterion='gini'`, `max_features=3`, `max_depth=4`, `min_samples_split=2`, `min_samples_leaf=8` and `bootstrap=True`.

Performance Evaluation The trained model was evaluated on the test data set, obtaining the values for the metrics shown in the table 4.

Accuracy: The model achieved a high accuracy of 0.95 for the >50 class, suggesting that most predictions for cyclists in the top 20 positions are accurate. The accuracy for the <=50 class was slightly lower at 0.94.

Recall: The recall for the >50 class was exceptionally high (0.99), suggesting that almost all cyclists in the top 20 positions were correctly identified. In contrast, recall for the <=50 class was moderate (0.66), meaning that many drivers outside the top 20 were misclassified.

F1 Score: The weighted F1 score was 0.95, reflecting a strong balance between precision and recall between classes.

Limitations Class Imbalance Despite reasonable performance, class imbalance (939 cases of <=50 vs. 6125 of >50) likely contributed to the disparity in recall and precision between the two classes.

Hyperparameter Coverage Although RandomizedSearchCV is computationally efficient, it does not guarantee that all possible hyperparameter combinations are explored. Some potentially advantageous configurations may have been overlooked.

Metric	DT (Class 0)	DT (Class 1)	RF (Class 0)	RF (Class 1)	SVM (Class 0)	SVM (Class 1)
Precision	0.95	0.91	0.96	0.93	0.93	0.61
Recall	0.99	0.67	0.99	0.70	0.95	0.51
F1-Score	0.97	0.77	0.97	0.80	0.94	0.56
Support	6139	6125	6139	939	6139	939
Accuracy	0.95		0.95		0.89	
Macro avg Precision	0.93		0.95		0.77	
Macro avg Recall	0.83		0.85		0.73	
Macro avg F1-Score	0.87		0.89		0.75	
Weighted avg Precision	0.95		0.95		0.88	
Weighted avg Recall	0.95		0.95		0.89	
Weighted avg F1-Score	0.94		0.95		0.89	

Table 4: Confronto dei risultati di classificazione tra Decision Tree Classifier e Random Forest

5.6.1 Analysis pattern

The **Swarm graphs** obtained for the random forest are very similar to those obtained for the decision tree model, so we can confirm the observations made in Section 5.5.2.

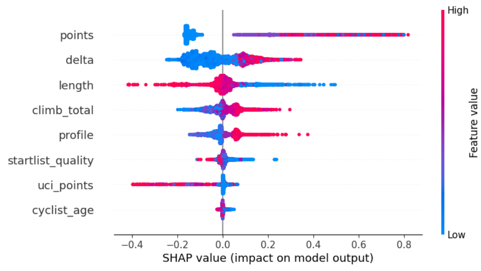

Figure 27: SHAP interventional (Random forest)

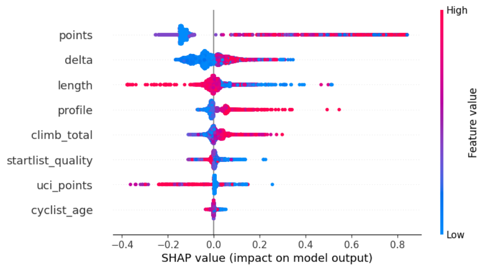

Figure 28: SHAP distributional(Random forest)

6 Conclusion

This project focused on analyzing cyclist performance through exploratory data analysis (EDA) and machine learning techniques. By extracting insights from a variety of models, the analysis highlights the power of feature engineering, model selection, and interpretability for real-world performance prediction tasks. Key findings include the importance of both linear and non-linear models in capturing key patterns and their interplay for effective prediction.

Key Findings

- **Initial Correlations and Feature Insights:** Initial data analysis revealed significant correlations between key features such as `points`, `uci_points`, and `position`. These were confirmed through model outputs, reinforcing their importance in performance prediction.
- **Impact of New Features:** Newly engineered features, such as `race_difficulty` and `race_performance_score`, provided further insights into cyclist performance. These features revealed additional clustering patterns, helping differentiate between various performance groups and subgroups.
- **Outlier Detection:** Advanced methods such as Local Outlier Factor (LOF), Isolation Forest, and One-Class Support Vector Machine (OCSVM) were employed to identify 493 outlier riders and 26 race outliers. These outliers often corresponded to extreme cases, such as atypical `delta` values or riders with points well outside the distribution of the dataset.
- **Clustering Patterns:** Clustering analysis highlighted clear performance patterns, identifying elite subgroups as well as noise points.
- **Model Evaluation and Feature Importance:** Linear classifiers and ensemble methods, such as Random Forests, yielded highly informative results. Decision Trees provided strong

insights into the importance of various features, revealing relationships between features like `delta` and `climb_total`.

- **Neural Networks:** The neural network model excelled in capturing non-linear dependencies between features, achieving high F1 scores despite class imbalance. SHAP (Shapley Additive Explanations) and LIME (Local Interpretable Model-agnostic Explanations) confirmed that features like `points`, `delta`, and `uci_points` were consistently crucial for prediction across different model families.
- **Conditional Dependencies:** Tree models (such as Decision Trees and Random Forests) revealed complex, conditional dependencies between features, including the interaction between `delta` and `climb_total`, and the impact of the `profile` feature. These interactions often had low initial correlations but emerged as impactful in specific clusters or race scenarios.
- **Comparing Model Performance:** Random Forests and Neural Networks emerged as the most effective models for this task, with their ability to capture complex non-linear relationships and conditional dependencies, resulting in high predictive performance despite class imbalance. However, for tasks requiring greater interpretability, the Decision Tree model was recommended, as it offered clearer, more transparent insights into feature relationships, albeit with slightly lower accuracy. Ensemble methods using Linear Classifiers offered a practical middle ground, combining efficiency with a simple interpretative approach, making them suitable for computationally constrained environments.

This analysis underscores the value of combining classical machine learning techniques with a thoughtful data preprocessing pipeline, which includes effective feature engineering and the discovery of relevant conditional dependencies. The insights gained from various machine learning models, particularly Random Forest and Neural Networks, provide strong predictions for cyclist performance, with sufficient flexibility to handle unbalanced class distributions. At the same time, for use cases where interpretability is essential, Decision Trees offer transparent results, ensuring that actionable insights can be derived.