Dauphine | PSL
UNIVERSITÉ PARIS

MIDO
MATHÉMATIQUES ET INFORMATIQUE
DE LA DÉCISION ET DES ORGANISATIONS

M1 IDD

Machine Learning Project / Data Science
Topic: Formula 1 Lap Time Prediction

Sepanta Farzollahi

Instructor: Pierre Wolinski

May 26, 2025

Abstract

This report presents a machine learning project aiming to explore and evaluate various regression techniques on a dataset related to Formula 1 lap times.

The primary goal is to apply machine learning to a real-world task by identifying a relevant dataset, understanding the context through external documentation, and selecting suitable learning algorithms.

Several regression models — including Linear Regression, K-Nearest Neighbors (KNN), Decision Trees, Bagging, Random Forests, Gradient Boosting, and XGBoost — were trained and evaluated using appropriate metrics such as Mean Squared Error (MSE) and R^2 score. Their performance is analyzed and compared to assess their relevance, strengths, and limitations in predicting lap time values.

Contents

Chapter 1

Introduction

Formula 1 (F1) is a sport where milliseconds matter, and every decision is driven by data. Predicting lap times plays a key role in optimizing performance, strategizing races, and understanding track dynamics. This project aims to explore how machine learning can be applied to lap time prediction by clearly defining its objectives, motivations, and analytical scope within the broader context of Formula 1.

1.1 Context and Importance of Formula 1

Formula 1 is widely regarded as the pinnacle of motorsport, combining engineering excellence, cutting-edge technology, and elite athletic performance. Each season, ten teams and twenty of the world's best drivers compete on circuits across the globe, pushing both man and machine to their limits. However, beyond the thrill of high-speed racing, F1 is fundamentally a sport of data.

Modern F1 cars are equipped with hundreds of sensors that generate gigabytes of telemetry data during a single race weekend. This data includes everything from tire temperatures and engine pressures to GPS coordinates and lap times. Teams rely on this information to make real-time strategic decisions and to continuously improve their performance through data analysis. As a result, data science and machine learning (ML) have become integral to the sport, offering a competitive edge to those who use them effectively.

1.2 The Importance of Lap Time Prediction

Among the various metrics available in F1, lap time is one of the most crucial. It encapsulates the overall performance of a car on a given circuit and is influenced by a multitude of factors: driver skill, car setup, tire compound, track conditions, fuel load, and even the timing within the session. Accurately predicting lap times can have practical applications such as optimizing race strategy, evaluating driver performance, or simulating race outcomes. For fans and analysts alike, it offers an insightful way to quantify and understand performance.

1.3 Project Objectives

The objective of this project is to apply machine learning techniques to the prediction of F1 lap times based on historical data. The task involves identifying or constructing a suitable dataset,

preprocessing it to handle real-world imperfections, selecting relevant features, training and evaluating multiple regression models, and analyzing their strengths and weaknesses.

This project serves not only as a technical challenge in machine learning but also as an exploration into the application of machine learning in a real-world, high-stakes environment. It allows us to reflect on which models generalize well to complex, multi-factor phenomena and how predictive modeling can contribute to a domain as sophisticated and competitive as Formula 1.

1.4 Scope of the Analysis

For simplicity and to avoid unnecessary complications in this study, all analyses will be based on a single Formula 1 season. The selected season is 2024, as it is the most recent year for which complete and consistent data across all Grand Prix events is available [1] . By focusing on one season, we ensure that the dataset is homogeneous in terms of regulations, technical changes, and track calendar, which helps to reduce variability caused by external factors such as rule changes between seasons or missing data. This approach allows for a clearer understanding of the performance dynamics and the effectiveness of the applied machine learning models on a consistent and representative dataset.

[1] For more information about the 2024 Formula 1 season, check https://www.formula1.com/en/racing/2024.

Chapter 2

Dataset Analysis and Preprocessing

A thorough understanding of the data is essential for effective modeling. This phase involves selecting a relevant dataset, ensuring its quality, and transforming it into a suitable format through cleaning, encoding, and exploratory analysis.

2.1 Dataset Selection

As outlined in the introduction, the objective of this project is to predict the lap times of Formula 1 drivers during the 2024 season. This requires obtaining lap-level data: that is, information for every single lap completed by each driver in every Grand Prix.

Initially, several datasets were explored on *Kaggle*. Unfortunately, most of them lacked sufficient features per lap, limiting their usefulness for detailed analysis. As a second step, an alternative strategy was considered: merging multiple datasets available online to enrich the feature space. However, two main issues arose with this approach. First, even after merging, the resulting dataset still lacked a sufficient number of relevant features. More critically, even for features common across datasets, values often differed due to rounding inconsistencies. These discrepancies were particularly problematic given the high precision required for lap time predictions — even thousandths of a second can make a difference. Attempting to harmonize values through rounding would compromise the integrity of the dataset and the validity of the predictions.

Moreover, merging datasets with just 5 or 6 shared features across roughly 27,000 rows (based on an estimated 20 drivers $\times$ 24 races $\times$ around 55 laps per race) would be computationally expensive and yield only marginal benefit.

Given these limitations, a third and more robust solution was adopted: to collect the raw lap-by-lap data ourselves. Several APIs provide access to such data, including Ergast API (by Ergast Developer API (2024)), FastF1 (by Schaefer (2025)), and OpenF1 (by Godefroy (2025)), among others. Although the Ergast API was historically popular, it was shut down in early 2025. As a result, FastF1 has emerged as the preferred option due to its ease of use and accessibility, in contrast to other APIs like OpenF1 which require an access key.

FastF1 was ultimately selected as the data source for this project. Using this API, all available information for each lap, for every driver, in every Grand Prix of the 2024 season, was successfully retrieved and used for further analysis.

2.2 Description of the Dataset

The lap-by-lap data for the 2024 Formula 1 season was extracted using the FastF1 API and stored in a CSV file named `grand_prix_laps_data_2024.csv`. The dataset initially contains 27,076 rows and 32 columns. Each row corresponds to a single completed lap by a driver in a Grand Prix.

Below is a sample representation of the information contained in one row of the dataset:

Feature	Example Value
Time	0 days 01:40:36.628000
Driver	VER
DriverNumber	1
LapTime	0 days 00:01:35.342000
LapNumber	25.0
Stint	2.0
PitOutTime	-
PitInTime	-
Sector1Time	0 days 00:00:30.518000
Sector2Time	0 days 00:00:41.282000
Sector3Time	0 days 00:00:23.542000
Sector1SessionTime	0 days 01:39:31.825000
Sector2SessionTime	0 days 01:40:13.107000
Sector3SessionTime	0 days 01:40:36.649000
SpeedI1	233.0
SpeedI2	250.0
SpeedFL	279.0
SpeedST	295.0
IsPersonalBest	False
Compound	HARD
TyreLife	8.0
FreshTyre	True
Team	Red Bull Racing
LapStartTime	0 days 01:39:01.286000
LapStartDate	2024-03-02 15:42:43.717
TrackStatus	1
Position	1.0
Deleted	False
DeletedReason	-
FastF1Generated	False
IsAccurate	True
GrandPrix	Bahrain

Table 2.1: Lap details for Verstappen's 25^{th} lap of the 2024 Bahrain Grand Prix

Let us now describe the different features available for each lap in the dataset:

- **Time:** Session time when the lap was completed. This marks the end of the lap in session time. The name *Time* is actually misleading; it should be *SessionTime*, but is kept for backward compatibility.

- **Driver:** Three-letter code representing the driver (e.g., `VER` for Max Verstappen).

Figure 2.1: Formula 1 2024 season driver line-up per team

- **DriverNumber:** Official number assigned to the driver (e.g., 1 for Max Verstappen).
- **LapTime:** Total time taken to complete the lap.
- **LapNumber:** The lap number in sequence during the session.
- **Stint:** A session in the car on the track. For example, if a driver completed 29 laps, then pitted for a tyre change and continued for another 40 laps, the driver had 29 laps in their first stint and 40 in their second stint.
- **PitOutTime:** Session time when the driver exited the pit lane (available only if a pit stop was made).
- **PitInTime:** Session time when the driver entered the pit lane (available only if a pit stop was made).
- **Sector1Time, Sector2Time, Sector3Time:** Time taken to complete each of the three sectors into which the track is divided.

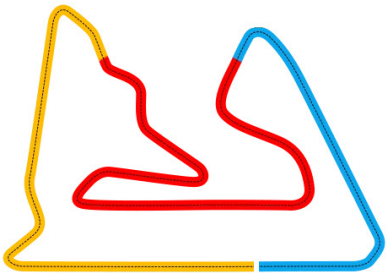

Figure 2.2: The three sectors of the Bahrain Grand Prix circuit

- **Sector1SessionTime, Sector2SessionTime, Sector3SessionTime:** Session time when each sector was completed.
- **SpeedI1, SpeedI2, SpeedFL, SpeedST:** Speeds recorded at specific points on the track:
  - **SpeedI1:** Speed trap in Sector 1
  - **SpeedI2:** Speed trap in Sector 2
  - **SpeedFL:** Speed at the finish line
  - **SpeedST:** Speed on the longest straight
- **IsPersonalBest:** Indicates whether the lap is officially the driver's best lap. Invalid faster laps (e.g., due to track limits) are not considered personal bests.
- **Compound:** Tyre compound used, such as `SOFT`, `MEDIUM`, `HARD`, `INTERMEDIATE`, `WET`, or `UNKNOWN`.

Figure 2.3: Formula 1 tyre compounds supplied by Pirelli®

- **TyreLife:** Number of laps completed on the current set of tyres, possibly including laps from previous sessions.
- **FreshTyre:** Indicates if the tyre was new at the start of the stint (`TyreLife = 0`).
- **Team:** Name of the team (e.g., `Red Bull Racing`).
- **LapStartTime:** Session time at the beginning of the lap.
- **LapStartDate:** Timestamp indicating the exact time (including date) when the lap started.
- **TrackStatus:** Sequence of status flags during the lap:
  - '1': Track clear
  - '2': Yellow flag
  - '3': Unknown
  - '4': Safety Car
  - '5': Red Flag
  - '6': Virtual Safety Car (VSC) deployed
  - '7': Virtual Safety Car (VSC) ending
- **Position:** Driver's position at the end of the lap.

- **Deleted:** Indicates whether the lap was deleted by race stewards (e.g., due to track limits violation).
- **DeletedReason:** Explains why the lap was deleted (available only with race control messages).
- **FastF1Generated:** Set to `True` if the lap was added by the `FastF1` library. Such laps may be partially interpolated and not fully accurate.
- **IsAccurate:** Indicates that lap start and end times are properly synchronized with other laps.

2.3 Data Exploration and Cleaning

After the initial dataset creation, an Exploratory Data Analysis (EDA) phase was conducted. First, a statistical summary of the numerical data was attempted, but it only worked for 10 out of 32 columns. This led to a thorough inspection of the column data types:

`Time`	object
`Driver`	object
`DriverNumber`	int64
`LapTime`	object
`LapNumber`	float64
`Stint`	float64
`PitOutTime`	object
`PitInTime`	object
`Sector1Time`	object
`Sector2Time`	object
`Sector3Time`	object
`Sector1SessionTime`	object
`Sector2SessionTime`	object
`Sector3SessionTime`	object
`SpeedI1`	float64
`SpeedI2`	float64
`SpeedFL`	float64
`SpeedST`	float64
`IsPersonalBest`	object
`Compound`	object
`TyreLife`	float64
`FreshTyre`	bool
`Team`	object
`LapStartTime`	object
`LapStartDate`	object
`TrackStatus`	int64
`Position`	float64
`Deleted`	bool
`DeletedReason`	object
`FastF1Generated`	bool
`IsAccurate`	bool
`GrandPrix`	object

Then, a check for missing values revealed that several features had null entries:

LapTime	224
PitOutTime	26226
PitInTime	26219
Sector1Time	540
Sector2Time	31
Sector3Time	53
Sector1SessionTime	600
Sector2SessionTime	31
Sector3SessionTime	53
SpeedI1	4163
SpeedI2	54
SpeedFL	885
SpeedST	2168
IsPersonalBest	15
LapStartDate	15
Position	28
DeletedReason	26745

Regarding `LapTime`, missing values were due either to a driver not completing a lap (e.g., due to an accident or early retirement) or not participating in the Grand Prix at all. When a lap was started but not finished, the missing time was filled using the mean lap time of the same driver for that specific Grand Prix. Rows with missing `LapTime` due to complete absence from a Grand Prix were dropped.

A similar treatment was applied to `Sector1Time`, `Sector2Time`, and `Sector3Time`. Missing values were filled using the corresponding average sector time of the same driver for the Grand Prix. Remaining rows with missing sector times were removed.

To unify data formats, all time-related columns stored as `object` (i.e., timedelta) were converted into seconds as `float`.

The same logic was extended to the speed-related columns (`SpeedI1`, `SpeedI2`, `SpeedFL`, `SpeedST`), where missing values were filled with the average speed of the driver in that Grand Prix.

Rows with missing `Position` values were also removed.

The `FreshTyre` and `IsPersonalBest` columns, originally boolean, were converted to integer values (1.0 for True, 0.0 for False). Missing values in `IsPersonalBest` were replaced with `False`.

Finally, columns not relevant to the prediction task were dropped:

- `Driver`, `DriverNumber`: identifiers that do not directly influence lap performance.
- `PitOutTime`, `PitInTime`: missing in over 97% of cases, as not all laps involve a pit stop.
- `Deleted`, `DeletedReason`: administrative markers indicating steward-deleted laps.

- `FastF1Generated`: technical flag added by the API.
- `IsAccurate`: indicates time syncing correctness, not performance.
- `LapStartDate`: contains absolute timestamps, irrelevant to lap prediction.
- `TrackStatus`: race control information (e.g., yellow/red flags, VSC) not used in modeling.

The cleaned dataset was saved into a new file: `grand_prix_laps_data_2024_clean.csv`, making further manipulation more convenient.

2.4 Feature Encoding and Exploratory Analysis

After cleaning the dataset, attention was directed to the categorical features: `Compound`, `Team`, and `GrandPrix`. As these variables represent nominal categories with no intrinsic ordering, one-hot encoding was applied. This transformation converts each categorical variable into a set of binary variables, ensuring compatibility with most machine learning algorithms and avoiding unintended ordinal relationships.

Boxplots were created to describe the distributions, central tendencies, variability, and potential outliers of the numerical features.

To make the boxplots easier to interpret and to ensure consistency in scale across all features, the numerical values were first scaled between 0 and 1. This transformation preserves the shape of each feature's distribution while allowing for direct comparison, especially in terms of variability and relative position of outliers :

- **Time:** Concentrated between 0.4 and 0.6, with a median near 0.5 and a narrow interquartile range, indicating low variability. Some outliers are present near 0 and near 1.
- **LapTime:** Median around 0.01, box ranging from 0 to 0.02. Low variability but outliers near 0.02 and 1.
- **LapNumber:** Median near 0.4, box from 0.2 to 0.6. Distribution fairly symmetrical without visible outliers.
- **Stint:** Median around 0.25, box from 0 to 0.5. Symmetrical distribution without outliers.
- **Sector1Time:** Median near 0.15, box from 0.1 to 0.18. Asymmetric distribution with numerous outliers extending up to 1.
- **Sector2Time:** Median around 0.2, box from 0.18 to 0.32. Asymmetric with many outliers up to 1.
- **Sector3Time:** Median around 0.1, box from 0.08 to 0.18. Asymmetric with many outliers up to 1.
- **Sector1SessionTime**, **Sector2SessionTime**, **Sector3SessionTime:** Centered between 0.4 and 0.6, medians near 0.5, with some outliers near 0 and 1.
- **SpeedI1:** Median near 0.75, box from 0.55 to 0.8. Asymmetric with outliers near 0.

- **SpeedI2:** Median around 0.72, box from 0.58 to 0.78. Similar to SpeedI1, with many outliers near 0.
- **SpeedFL:** Median near 0.78, box from 0.7 to 0.8. Asymmetric with outliers in the middle and near 0.
- **SpeedST:** Median near 0.8, box from 0.75 to 0.82. Slightly asymmetric with many outliers near 0 and some near 1.
- **TyreLife:** Median near 0.17, box from 0.08 to 0.26. Slightly asymmetric with outliers from the middle to near 1.
- **LapStartTime:** Median around 0.5, box from 0.4 to 0.6. Low variability with some outliers near 0.
- **Position:** Median near 0.5, box from 0.2 to 0.7. Slightly asymmetric without outliers.

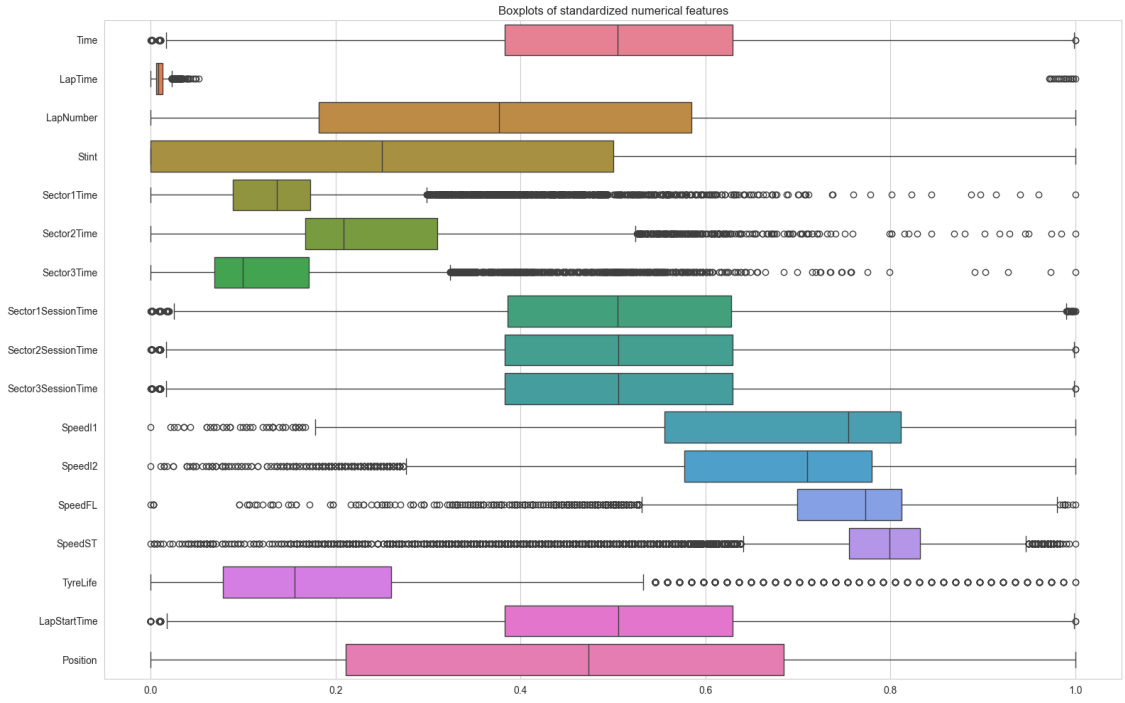

Figure 2.4: Boxplots of numerical features

Regarding time-related features, `LapTime` equals the sum of `Sector1Time`, `Sector2Time`, and `Sector3Time`, indicating these three sector features cannot be used as independent predictors since they directly determine `LapTime`.

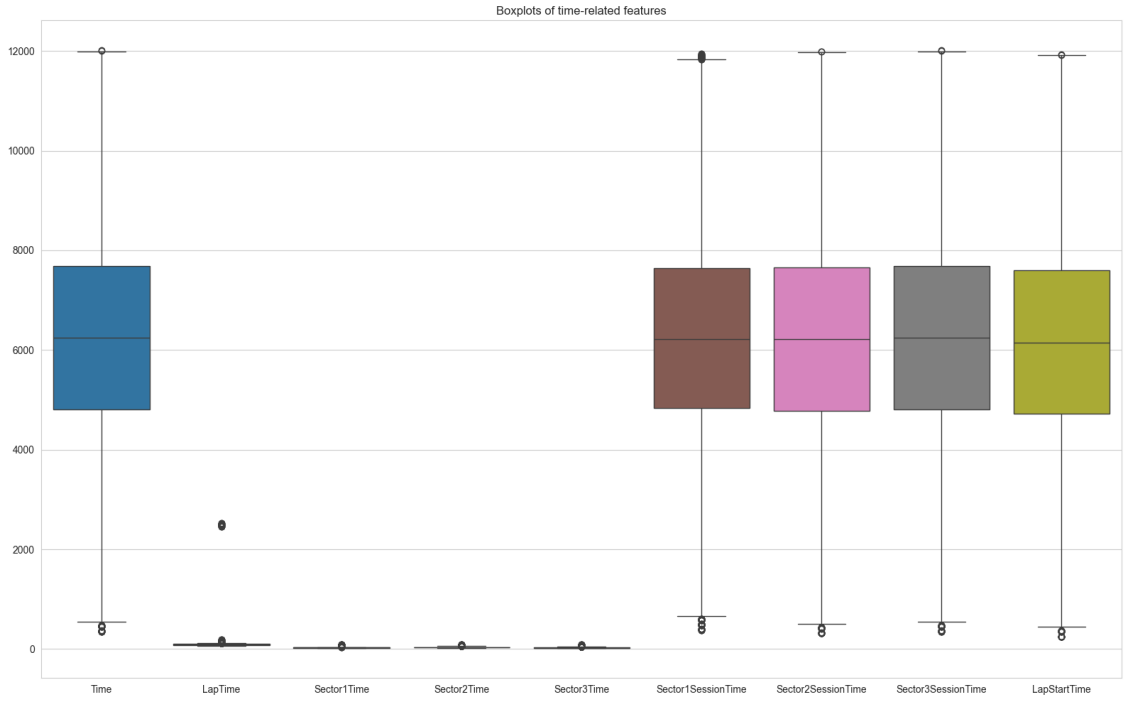

Figure 2.5: Boxplots of time-related features

The speed-related features exhibit similar median values, although numerous outliers appear near zero and within the distributions.

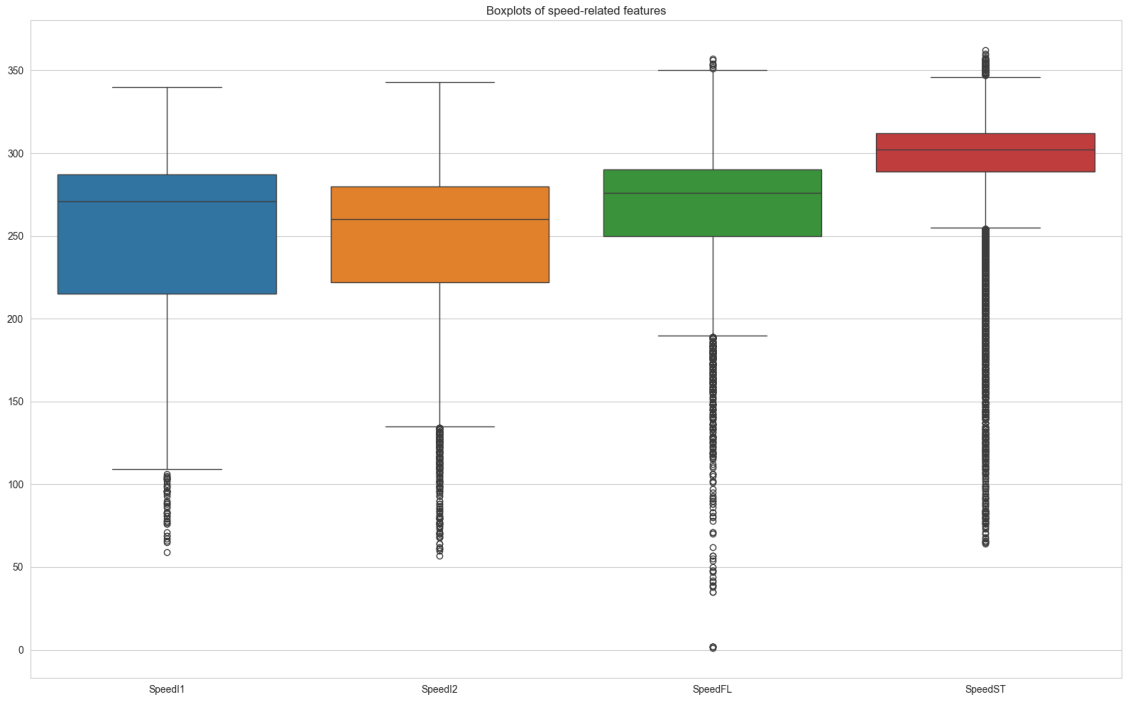

Figure 2.6: Boxplots of speed-related features

In a second step, histograms were used to visualize the distributions of the numerical features:

- **Time:** Distribution centered around 0.5, indicating a strong central tendency.
- **LapTime:** Values are concentrated around a small central point, reflecting performance consistency.
- **LapNumber:** Early laps are the most frequent, with frequency decreasing as lap number increases.
- **Stint:** Several distinct peaks suggest specific racing phases such as tire changes or strategic stints.
- **Sector1Time:** Stronger concentration near lower values, corresponding to faster sector times.
- **Sector2Time:** Similar to Sector1Time, but with slightly higher peaks.
- **Sector3Time:** Similar shape to Sector2Time, with slightly lower peaks.
- **Sector1SessionTime**, **Sector2SessionTime**, **Sector3SessionTime:** All show central distributions around 0.5.
- **SpeedI1**, **SpeedI2:** Multimodal distributions indicating varying speed depending on context (e.g., corners, straights).
- **SpeedFL:** Several peaks in a narrower range, indicating consistent high-speed performance.
- **SpeedST:** A single dominant peak, suggesting that speeds at that point in the track are stable.
- **IsPersonalBest:** Binary distribution with peaks at 0 and 1, more frequent at 0.
- **TyreLife:** Increasing at first, with a heavier tail due to longer stints leading to tire wear.
- **FreshTyre:** Binary distribution similar to `IsPersonalBest`, but more frequent at 1.
- **LapStartTime:** Central distribution around 0.5, with symmetrical frequency.
- **Position:** Multimodal distribution showing frequent rank changes during the session.

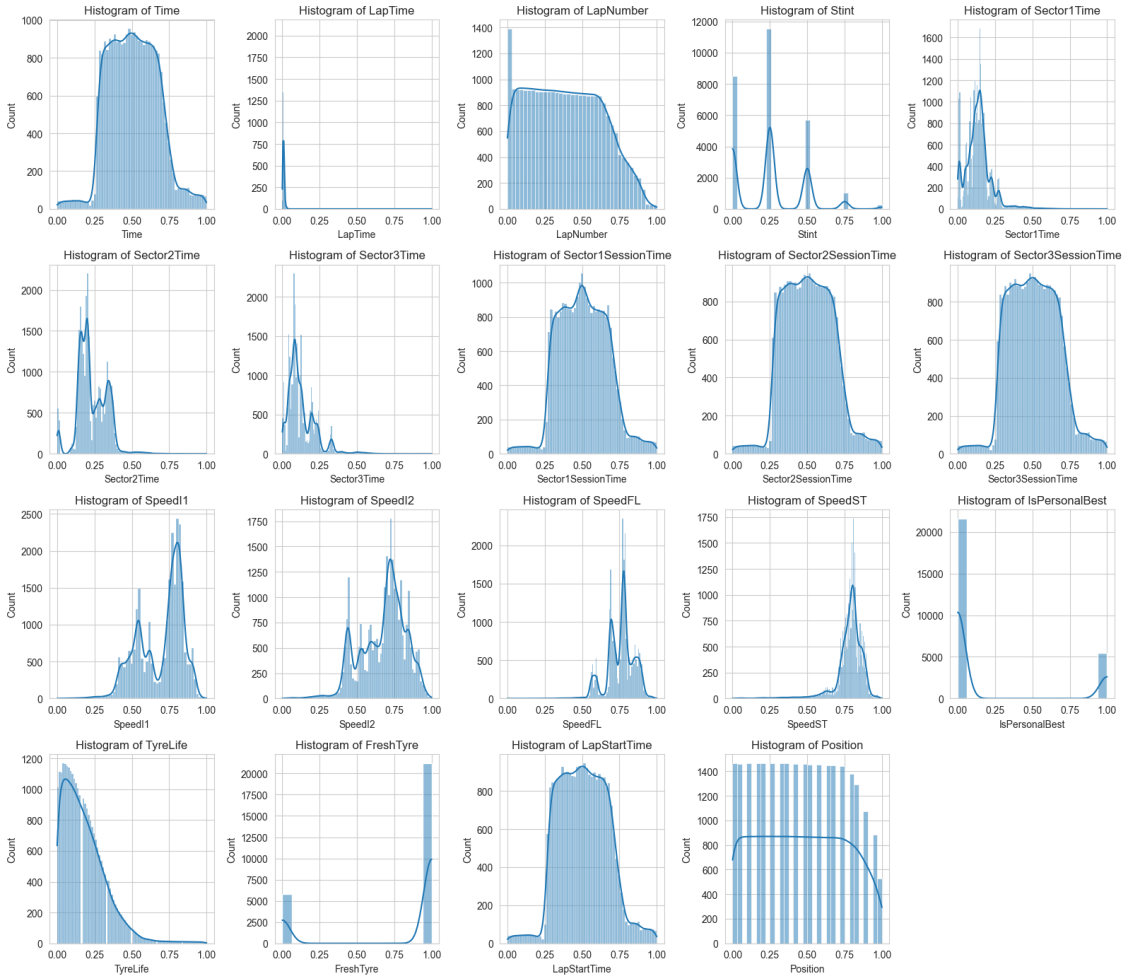

Figure 2.7: Distribution of features represented as histograms

To quantify the asymmetry of these distributions, skewness was computed for each numerical feature. Skewness indicates the degree of asymmetry in the distribution of values:

- A skewness close to 0 suggests a symmetrical distribution.
- A positive skewness indicates a longer tail on the right (toward higher values).
- A negative skewness indicates a longer tail on the left (toward lower values).

The table below summarizes the skewness values:

Time	0.1508
LapTime	38.3709
LapNumber	0.1880
Stint	0.6793
Sector1Time	1.7363
Sector2Time	0.5746
Sector3Time	1.8829
Sector1SessionTime	0.1503
Sector2SessionTime	0.1508
Sector3SessionTime	0.1483
SpeedI1	-0.6749
SpeedI2	-0.6001
SpeedFL	-0.9782
SpeedST	-2.9336
IsPersonalBest	1.4897
TyreLife	1.4340
TyreLife	-1.3805
LapStartTime	0.1511
Position	0.0619

Among these, LapTime, Sector1Time, Sector3Time, SpeedST, IsPersonalBest, TyreLife and TyreLife exhibit strong skewness, which may affect the learning algorithms.

A pair plot was also generated to visualize the relationships between the different features, including LapTime. As can be seen (although the plots are somewhat small), the second row corresponding to LapTime does not show any clearly significant positive or negative linear correlation with the other features. Even with the red regression lines, no strong visual pattern or trend is noticeable.

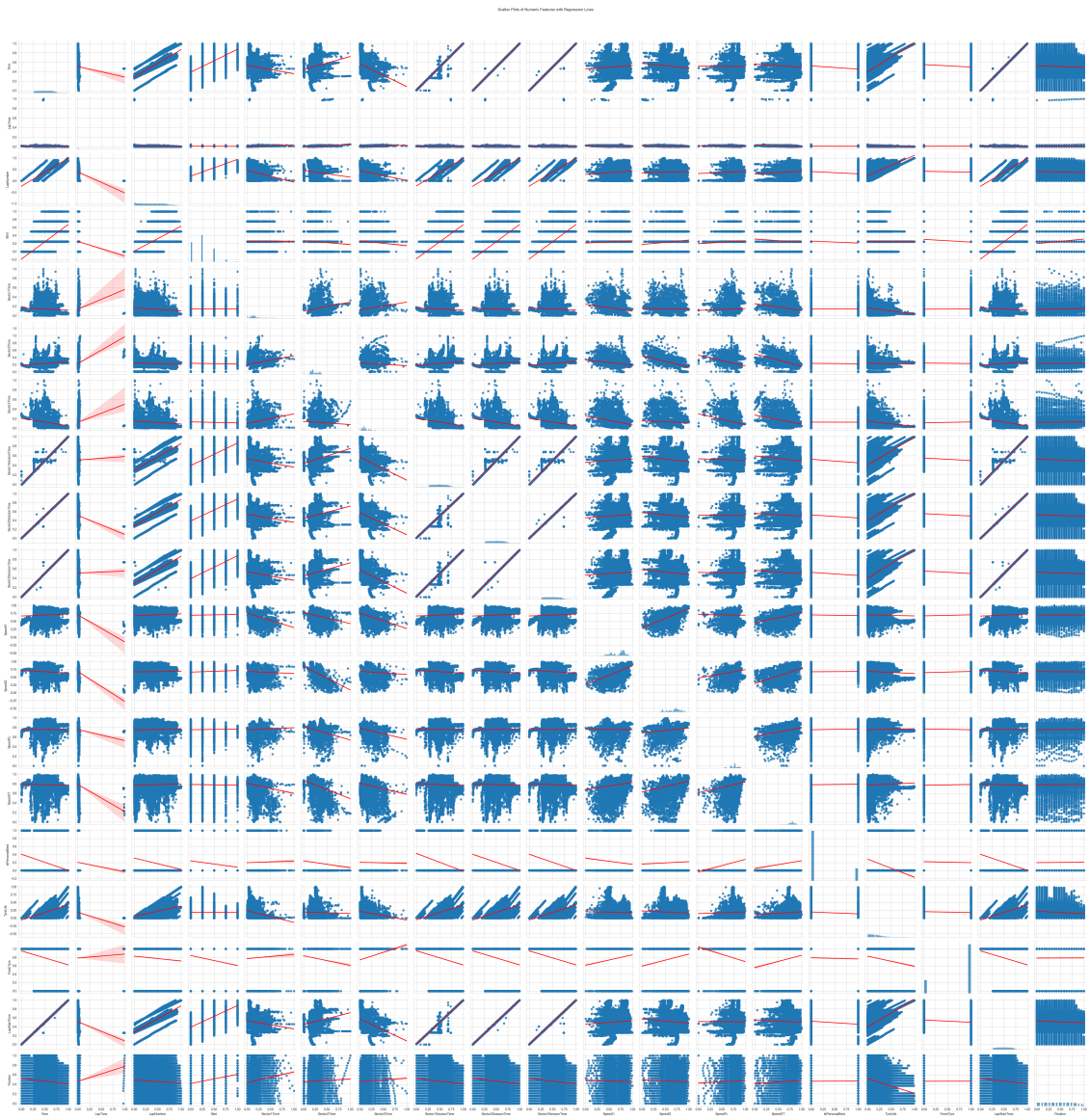

Figure 2.8: Pair plot showing the relationships between features

So in order to better understand the relationships between variables, correlation heatmaps were generated. Initially, the heatmap was computed only between numerical (non-categorical) features.

The target variable, `LapTime`, shows the following correlation values with other numerical features:

- `Time`: -**0.03**
- `LapNumber`: -**0.09**
- `Stint`: -**0.04**
- `Sector1Time`: **0.12**
- `Sector2Time`: **0.19**

- `Sector3Time`: **0.10**
- `Sector1SessionTime`: **0.01**
- `Sector2SessionTime`: **-0.06**
- `Sector3SessionTime`: **0.01**
- `SpeedI1`: **-0.14**
- `SpeedI2`: **-0.16**
- `SpeedFL`: **-0.06**
- `SpeedST`: **-0.13**
- `IsPersonalBest`: **-0.01**
- `TyreLife`: **-0.08**
- `FreshTyre`: **0.01**
- `LapStartTime`: **-0.06**
- `Position`: **0.03**

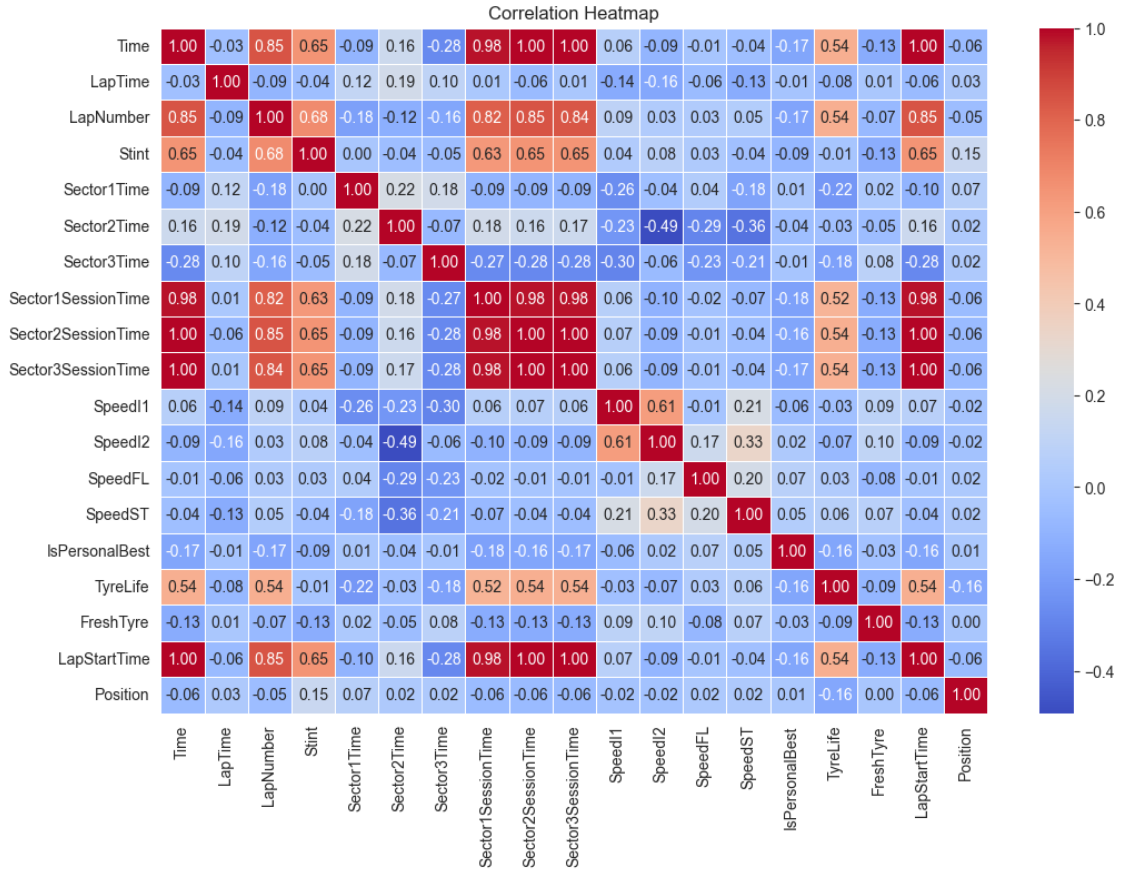

Figure 2.9: Correaltion heatmap of numerical features

From this analysis, we can observe that the strongest positive correlations with LapTime are with the three sector times, especially Sector2Time and Sector1Time, indicating that these sectors significantly contribute to the total lap time.

However, this result is expected and should be interpreted with caution. As mentioned previously, the sum of Sector1Time, Sector2Time, and Sector3Time gives the exact LapTime. Therefore, using these features to predict LapTime would be equivalent to using parts of the answer to predict the answer itself — which would introduce data leakage and invalidate the predictive modeling process. For this reason, these three features will be excluded from the modeling step. We will elaborate further on this decision in the following chapter.

That said, we can still observe some correlations between LapTime and other features such as SpeedI1, SpeedI2, SpeedFL, SpeedST, LapNumber, TyreLife, LapStartTime, Stint, and Position. Although these correlations are not particularly high, they are relatively meaningful in the context of this dataset. Indeed, since the overall level of correlation among the features is generally low, we must consider even small correlations, as they may still carry useful predictive signals. This further supports the previous conclusion that most features have only weak linear relationships with LapTime. Ignoring them could result in underfitting and missed opportunities for improving model performance. Thus, these features will be retained for the modeling phase.

To verify whether categorical features (after one-hot encoding) could provide additional predictive power, a second heatmap was generated between the one-hot encoded variables and the numerical features, including LapTime. This was done by analyzing the mean differences in LapTime across the different categories, effectively measuring whether belonging to a certain category results in a significant shift in the average lap time.

Unfortunately, this analysis showed no significant correlation between any of the one-hot encoded features and LapTime. The lack of meaningful linear relationships suggests that these categorical indicators do not add predictive value in the context of our regression task.

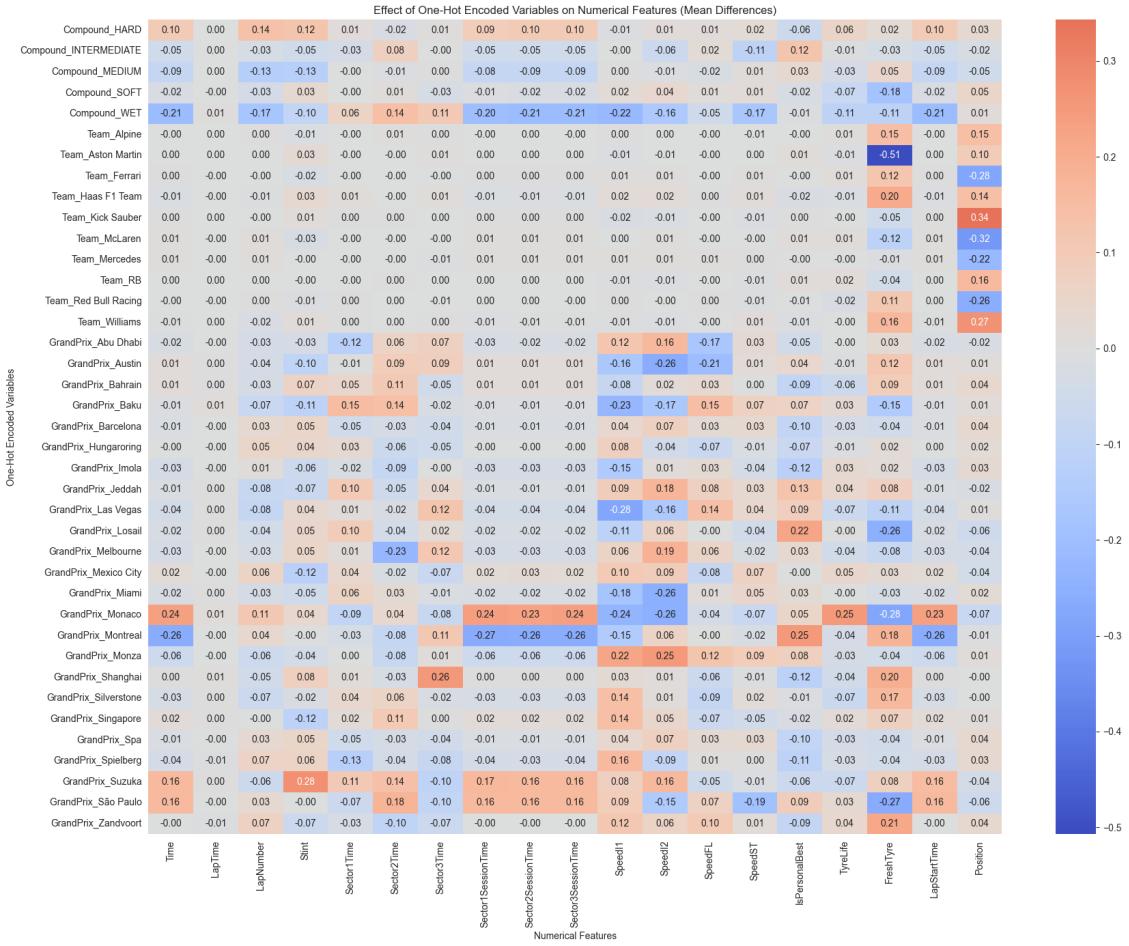

Figure 2.10: Effect of categorical features on numerical features (mean differences)

As a result, the one-hot encoded variables were not included in the prediction task. This decision not only reflects their limited relevance, but also contributes to reducing feature dimensionality and mitigating the risk of noise or overfitting in the subsequent modeling phase.

To gain insights into the structure of the dataset and reduce dimensionality, Principal Component Analysis (PCA) was performed. This technique allows for the identification of directions (principal components) along which the data varies the most, helping to uncover underlying patterns and simplify the feature space.

Prior to applying PCA, all numerical features were standardized to have zero mean and unit variance. This step is crucial because PCA is sensitive to the scale of the features—variables with larger scales can dominate the principal components. Standardization ensures that all features contribute equally to the analysis.

As observed in the *explained variance by principal components* and *cumulative explained variance* plots:

- The first principal component captures approximately 35% of the total variance. This indicates that a significant amount of the variability in the dataset can be explained along this single axis.

- The second component captures an additional 13%, bringing the cumulative explained variance of the first two components to nearly 50%.

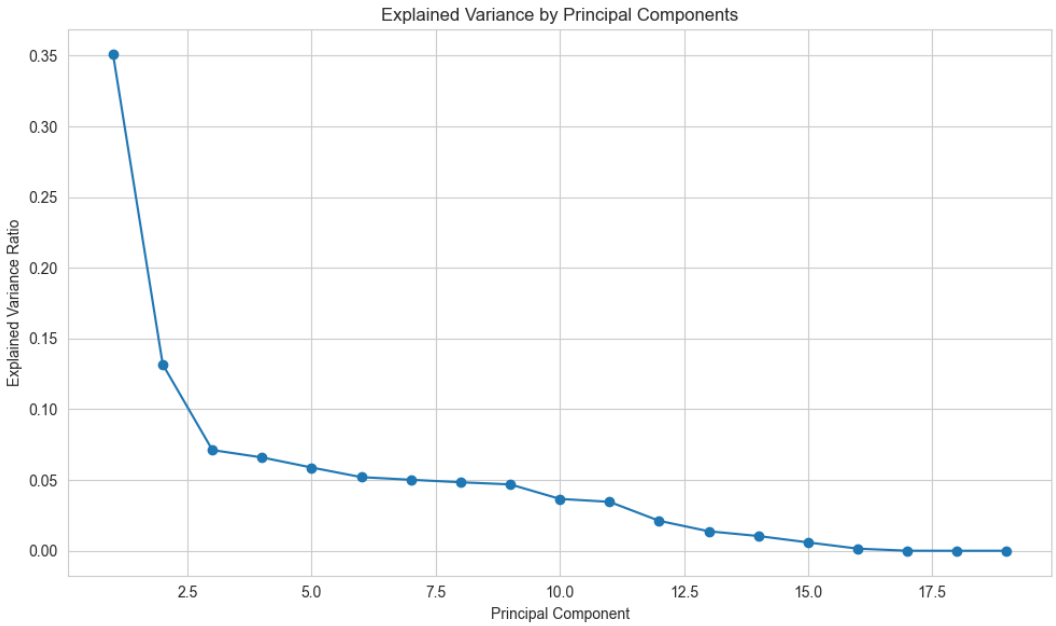

Figure 2.11: Explained variance by principal components

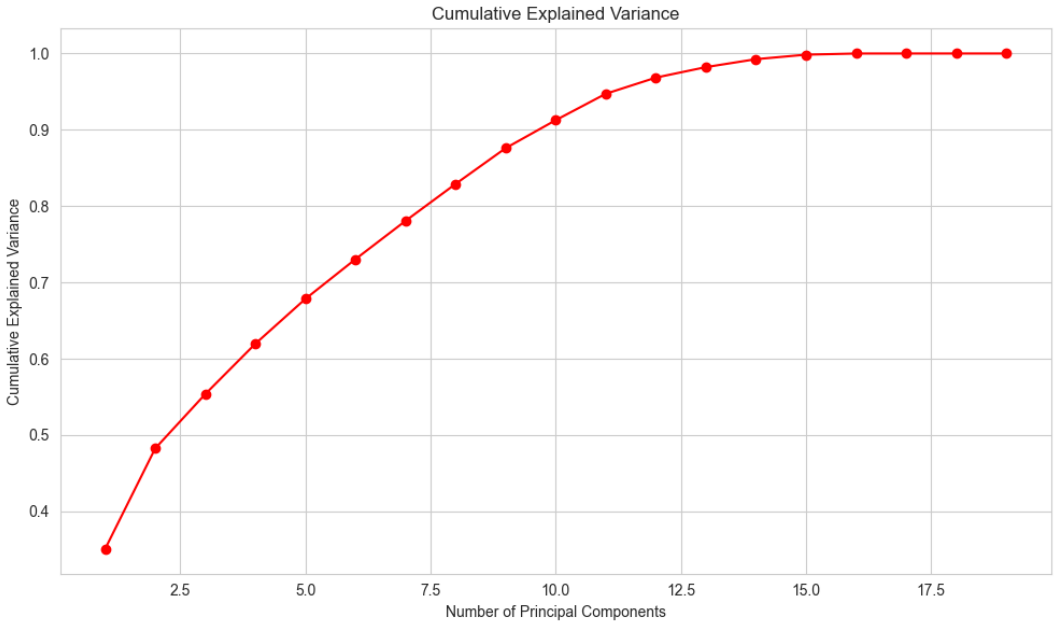

Figure 2.12: Cumulative explained variance

This result suggests that while the dimensionality of the dataset is relatively high, a substantial portion of the variance can already be represented in just two dimensions. These findings will be useful for both visualization and potentially for dimensionality reduction in future modeling steps.

The PCA Correlation Circle, based on the first two principal components, provides a visual interpretation of how the original features are projected in the new component space and how they relate to each other.

In particular, when considering the position of `LapTime` in this space:

- The variables `Sector1Time`, `Sector2Time`, and `Sector3Time` are strongly and positively correlated with `LapTime`, as they appear close to its direction in the plot. This is expected since the lap time is composed of the sum of these sector times.
- Conversely, the speed-related features (`SpeedI1`, `SpeedI2`, `SpeedFL`, and `SpeedST`) are largely anti-correlated with `LapTime`. They point in the opposite direction on the circle, indicating that higher speeds tend to be associated with shorter lap times.
- Other numerical features show weaker correlations with `LapTime`, as they are either closer to the origin or positioned more orthogonally, indicating a smaller influence on the overall variance explained in relation to the lap time.

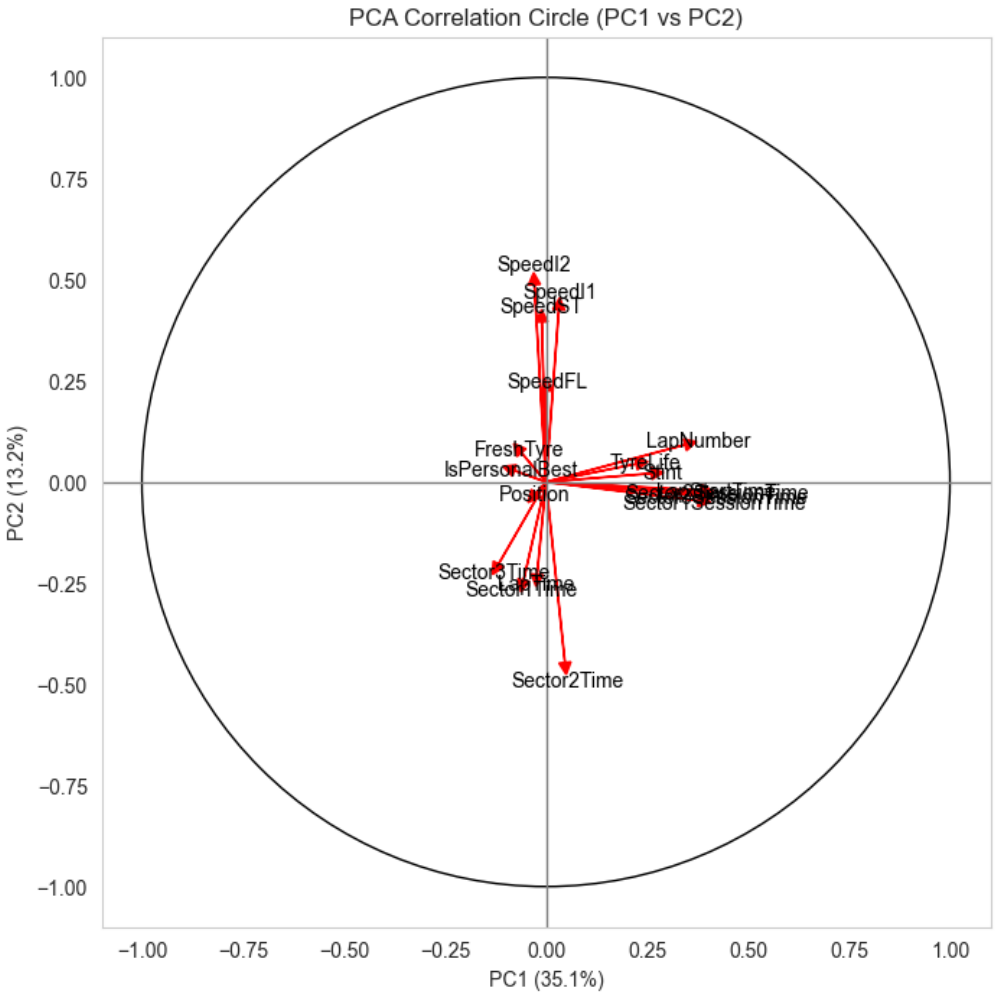

Figure 2.13: PCA correlation circle (first two components)

This visualization further confirms the relationships observed in the correlation matrix and highlights the importance of speed and sector times in determining the lap time.

Chapter 3

Prediction and Model Evaluation

This part of the document focuses on the methodology adopted for building and evaluating predictive models. Starting from a careful selection of features and a principled data splitting strategy, various algorithms were implemented and assessed using appropriate metrics. Particular attention was given to identifying signs of underfitting and overfitting, ensuring that the final models offer not only strong performance but also reliable generalization to unseen data.

3.1 Feature Selection and Data Splitting

Based on the previous exploratory analysis of the dataset, the features selected for the prediction task (**X**) are:

> `SpeedI1`, `SpeedI2`, `SpeedFL`, `SpeedST`, `LapNumber`, `TyreLife`, `LapStartTime`, `Stint`, `Position`

Other features were excluded either due to:

- A lack or very weak correlation with `LapTime` (the target variable),
- Or because they have obvious direct links to `LapTime`, such as `Sector1Time`, `Sector2Time`, and `Sector3Time`. Including these sector times as features would be equivalent to "cheating," since the lap time is essentially the sum of these sector times. Initially, when these features were included, models showed unrealistically high performances (around 97% to 98%), which is probably misleading given the weak correlations observed between most features and `LapTime`, as discussed in the previous chapter.

The target variable (**y**) is `LapTime`.

Two successive splits were performed on the dataset to prepare the data for training and evaluation:

- **First split:** 80% of the data was used for training and validation, and 20% reserved for testing. This ensures an unbiased evaluation of the final model's generalization.
- **Second split:** The 80% training/validation set was further split into 75% training and 25% validation subsets. This split allows tuning hyperparameters and model selection without peeking at the test set.

To find the best model and hyperparameters, each algorithm was evaluated using cross-validation (CV) with varying numbers of folds, from 3 up to 10, in order to identify the optimal fold number that maximizes performance. For each model, the best performance achieved under its best CV configuration was recorded.

Additionally, hyperparameter tuning was performed via exhaustive grid search over a predefined parameter space. We will detail this tuning process later.

3.2 Evaluation Metrics and Overfitting Considerations

Since the task is a regression problem, the evaluation metrics chosen to assess model effectiveness are:

- **Root Mean Squared Error (RMSE):** measures the magnitude of prediction errors. *Objective: minimize RMSE. A lower RMSE indicates better performance, with the ideal value being 0 (perfect predictions).*
- **Coefficient of Determination (R^2):** indicates the proportion of variance in the target variable that is predictable from the input features.
  Objective: maximize R^2. Values closer to 1 indicate better performance. A value of 1 means perfect prediction, while values near 0 or negative indicate poor model performance.

Model comparisons are based primarily on these metrics evaluated on the test set.

Particular attention is also paid to **overfitting**, which occurs when a model performs well on the training data but poorly on unseen data. This is monitored by comparing the performance on the training and validation sets throughout the model selection process.

3.3 Models Considered and Their Performances

Several regression algorithms were evaluated. Below we describe the main ones and their results.

3.3.1 Linear Models: Linear Regression, Ridge, Lasso, ElasticNet

The first and simplest model considered was the **Linear Regression**, which fits a linear relationship between features and target. Given the weak correlations observed between the features and `LapTime`, as well as the simplicity of this model, we expected relatively poor performance. Indeed, it consistently ranked at the bottom with:

$$\text{Test RMSE} \approx 0.00520, \quad \text{Test } R^2 \approx -0.02531$$

across various CV settings.

Two regularized linear models were also tested:

- **Ridge Regression:** adds an ℓ_2 penalty on coefficients to reduce overfitting by shrinking them toward zero. The strength of this penalty is controlled by the hyperparameter `alpha`.

- **Lasso Regression:** uses an ℓ_1 penalty, which can shrink some coefficients exactly to zero, performing implicit feature selection.

Despite these regularizations, neither Ridge nor Lasso improved results significantly:

$$\text{Ridge: Test RMSE} \approx 0.00516, \quad R^2 \approx -0.00942$$

$$\text{Lasso: Test RMSE} \approx 0.00519, \quad R^2 \approx -0.01981$$

Testing various `alpha` values (controlling penalty strength) did not yield any noticeable improvement, confirming that these models are not well suited to the task given the features at hand.

ElasticNet, a combination of Ridge and Lasso penalties, was also evaluated. It blends ℓ_1 and ℓ_2 regularizations to balance coefficient shrinkage and sparsity. Its performance was similar to Lasso:

$$\text{ElasticNet: Test RMSE} \approx 0.00519, \quad R^2 \approx -0.01981$$

Overall, the linear family of models showed limited predictive ability on this dataset.

3.3.2 K-Nearest Neighbors (KNN)

The **KNN Regressor** is a non-parametric model which predicts a sample's target by averaging the targets of its k nearest neighbors in feature space.

Various values of neighbors (`k`) were tested initially:

- `k` $= 3$ was discarded because it led to severe overfitting: using too few neighbors causes the model to be too sensitive to noise and local fluctuations,
- `k` $= 5$ gave the best initial results:

$$\text{Test RMSE} \approx 0.00303, \quad R^2 \approx 0.65123$$

To further improve the model, a grid search was performed to tune the hyperparameters, i.e., an exhaustive search over combinations of parameters to find the best configuration. The main parameters explored were:

- `n_neighbors` – the number of neighbors used for prediction (e.g., values like 5, 7, 9), which controls the bias-variance tradeoff: a smaller value leads to more variance (overfitting), while a larger value leads to more bias (underfitting),
- `weights` – the weighting function used in prediction, with two options:
  - `uniform`: all neighbors contribute equally,
  - `distance`: closer neighbors have a higher influence, which can improve predictions when the nearby points are more relevant.

The grid search selected the combination that yielded slightly different metrics:

$$\text{Test RMSE} \approx 0.00361, \quad R^2 \approx 0.50579$$

This somewhat lower performance after grid search might seem counterintuitive. It can be explained by the fact that grid search aims to find a model that generalizes well on unseen data by reducing overfitting, sometimes at the cost of slightly worse performance on the specific test set used. Additionally, KNN is sensitive to noise and feature scaling, which can affect results. Overall, the KNN regression captures some nonlinear relationships in the data, but its moderate performance suggests it may not be the best fit for this problem.

3.3.3 Decision Tree

The **Decision Tree Regressor** operates by building a tree-like model of decisions and their possible consequences. Specifically, it recursively partitions the data space into smaller regions by selecting the feature and threshold that best minimize the variance of the target variable within each subset. This makes it a non-parametric model well-suited to capturing complex nonlinear relationships in the data.

In its basic form—using default hyperparameters and without any feature engineering or cross-validation adjustments—the model already yielded a very strong performance:

$$\text{Test RMSE} \approx 0.00196, \quad R^2 \approx 0.85474$$

This made it the best-performing model among those tested at this stage, as long as no hyperparameter tuning or model-specific optimization (like grid search) was applied.

To evaluate whether performance could be improved, a grid search was conducted over the following hyperparameters:

- `max_depth`: the maximum depth allowed for each tree (e.g., values like None, 5, 10, 20),
- `min_samples_split`: the minimum number of samples required to split an internal node (e.g., values like 2, 5, 10).

These parameters are critical for controlling the model's complexity and avoiding overfitting. However, the performance gain achieved through tuning was minimal:

$$\text{Test RMSE} \approx 0.00194, \quad R^2 \approx 0.85729$$

While this change was statistically minor, it suggests that the default configuration of the decision tree was already near-optimal for this dataset. Overall, the Decision Tree Regressor proved to be a strong and interpretable baseline model with excellent out-of-the-box performance.

3.3.4 Random Forest

The **Random Forest Regressor** is a widely-used ensemble learning technique that builds upon decision trees. Unlike a single decision tree, a random forest combines the predictions of multiple trees to improve generalization. Each tree is trained on a different bootstrap sample of the training data (a random sample with replacement), and at each split, only a random subset of features is considered. This strategy reduces variance and mitigates the risk of overfitting commonly associated with individual decision trees.

Given its ensemble nature, the Random Forest is generally expected to outperform a single tree. However, the initial results were somewhat surprising: the model underperformed compared to the standalone decision tree:

$$\text{Test RMSE} \approx 0.00249, \quad R^2 \approx 0.76480$$

To investigate further, a grid search was carried out over key hyperparameters:

- `n_estimators`: the number of trees in the forest (e.g., values like 50, 100, 200),
- `max_depth`: the maximum depth of each tree (e.g., values like None, 5, 10),
- `min_samples_split`: the minimum number of samples required to split a node (e.g., values like 2, 5).

These parameters control the complexity of the individual trees and the size of the ensemble. Despite this optimization effort, the performance did not improve significantly:

$$\text{Test RMSE} \approx 0.00249, \quad R^2 \approx 0.76480$$

This result suggests that, in this particular case, the benefit of variance reduction offered by the ensemble did not outweigh the potential loss of precision due to averaging many slightly underfitted trees. It also highlights that ensemble methods are not always superior by default, and their effectiveness can vary depending on the data characteristics and feature distributions.

3.3.5 Bagging

The **Bagging Regressor**, short for Bootstrap Aggregating, is an ensemble learning method that builds multiple base learners (here, Decision Trees) on different random subsets of the training data. These subsets are generated via bootstrapping, i.e., sampling with replacement. The final prediction is obtained by averaging the predictions of all the individual models, which generally helps reduce variance and improve model stability.

The initial performance of the Bagging Regressor, using default parameters, was quite solid:

$$\text{Test RMSE} \approx 0.00248, \quad R^2 \approx 0.76641$$

To explore potential improvements, a grid search was conducted over the following key hyperparameters:

- `n_estimators`: the number of base estimators (trees) in the ensemble (e.g., values like 50, 100, 200),
- `max_samples`: the proportion or number of training samples to draw for each base estimator (e.g., values like 0.5, 1).

However, contrary to expectations, hyperparameter tuning did not lead to better results. In fact, a slight drop in performance was observed:

$$\text{Test RMSE} \approx 0.00280, \quad R^2 \approx 0.70319$$

This outcome may seem counterintuitive, but it highlights an important aspect of grid search : it aims to identify models that generalize best across multiple validation folds, rather

than optimizing performance on a single test set. As such, it tends to favor models that are more stable and less prone to overfitting. In this case, although the tuned model had a slightly lower test score, it may still be preferable due to its improved generalization ability and robustness across different subsets of data.

Moreover, the initial default parameters might have already been near-optimal for this dataset, and further tuning introduced small instabilities due to increased model complexity or decreased sample diversity among base estimators.

3.3.6 Gradient Boosting

The **Gradient Boosting Regressor** is a powerful ensemble technique that builds decision trees sequentially. Each new tree attempts to correct the errors made by the previous ensemble by fitting to the negative gradient of the loss function with respect to the current predictions. This approach allows the model to iteratively reduce the prediction error, making it highly effective for capturing complex nonlinear relationships in the data.

Initial results using basic hyperparameter settings were surprisingly poor:

$$\text{Test RMSE} \approx 0.00715, \quad R^2 \approx -0.93699$$

The poor performance is likely due to a mismatch between the model complexity and the nature of the dataset. Specifically, the initial configuration was:

- `n_estimators = 100` – the number of boosting stages (trees) to perform,
- `learning_rate = 0.1` – the shrinkage rate that controls how much each tree contributes to the final prediction,
- `max_depth = 3` – the maximum depth of each individual regression tree.

While these are common default values, they may not have been suitable here. For example, a depth of 3 might be too shallow to capture important patterns, and 100 trees might be insufficient if the learning rate is too low or the model underfits.

To improve the model, an extensive grid search was performed over the following hyperparameters:

- `n_estimators` – controlling the number of boosting rounds (e.g., values like 100, 200),
- `learning_rate` – adjusting the contribution of each new tree (e.g., 0.05, 0.1, 0.2),
- `max_depth` – modifying tree complexity (e.g., 3, 5).

This led to a significant improvement in performance:

$$\text{Test RMSE} \approx 0.00161, \quad R^2 \approx 0.90706$$

This was the best performance obtained among all the models tested. The improvement can be attributed to the core mechanism of gradient boosting, which allows the model to sequentially refine its predictions and focus more on difficult-to-fit instances. With proper tuning, the model was able to find a good balance between bias and variance, achieving both accuracy and generalization. This highlights the importance of hyperparameter optimization, especially for complex models like boosting algorithms.

3.3.7 XGBoost

XGBoost Regressor (Extreme Gradient Boosting) is a highly efficient and scalable implementation of gradient boosting. It includes regularization terms to reduce overfitting, handles missing values internally, and supports parallel computation for faster training. Due to these enhancements, it is widely used in data science competitions and real-world applications.

The initial model used the following standard configuration:

- `n_estimators = 100` – number of boosting rounds,
- `learning_rate = 0.1` – step size shrinkage applied to update weights,
- `max_depth = 3` – maximum depth of a tree (controlling complexity).

With this setup, the model underperformed significantly:

$$\text{Test RMSE} \approx 0.00467, \quad R^2 \approx 0.17442$$

This relatively poor result may be due to underfitting: the trees were not deep enough to capture complex patterns, and the number of trees may have been insufficient for the model to converge to a good solution. Moreover, while XGBoost includes regularization, an overly simple configuration can prevent it from leveraging its full potential.

To improve performance, a grid search was conducted over a broader space of hyperparameters, notably:

- `n_estimators` – values like 100, 200, to control the number of boosting rounds,
- `learning_rate` – values such as 0.05, 0.1, 0.2, to adjust the pace of learning,
- `max_depth` – values like 3, 5, to vary the capacity of each individual tree.

This tuning process led to a notable improvement:

$$\text{Test RMSE} \approx 0.00234, \quad R^2 \approx 0.79317$$

Although it did not outperform the classic Gradient Boosting Regressor in this case, XGBoost proved to be a robust and competitive model after optimization. The result confirms that while XGBoost can offer speed and additional regularization benefits, its performance remains highly dependent on careful hyperparameter tuning—especially for relatively small or structured datasets.

Chapter 4

Results and Perspectives

The following sections present an in-depth examination of the results obtained by models developed throughout this project, highlighting their respective performances and generalization abilities. Quantitative comparisons are made using standard evaluation metrics, allowing for a critical assessment of each approach. Beyond this analysis, potential avenues for further improvement are explored, taking into account both the limitations encountered and the practical considerations of model complexity, interpretability, and computational cost.

4.1 Model Comparison and Analysis

As mentioned in the previous chapter, each algorithm was carefully optimized through grid search to identify the best hyperparameter configurations and the most suitable number of cross-validation folds (CV). The aim was to ensure optimal generalization ability for each model tested. The final performance metrics, including the test RMSE and R^2 score, are shown in the table below, providing a clear comparison across all trained models.

Rank	Model	Best Parameters	Test RMSE	Test R^2
1	Gradient Boosting	`learning_rate` = 0.2, `max_depth` = 5, `n_estimators` = 200	0.00157	0.90706
2	Decision Tree	`max_depth` = None, `min_samples_split` = 2	0.00196	0.85474
3	XGBoost	`learning_rate` = 0.2, `max_depth` = 5, `n_estimators` = 200	0.00234	0.79317
4	Random Forest	`max_depth` = None, `min_samples_split` = 2, `n_estimators` = 100	0.00249	0.76480
5	Bagging (Decision Tree)	`max_samples` = 1.0, `n_estimators` = 200	0.00280	0.70319
6	KNN	`n_neighbors` = 5, `weights` = distance	0.00361	0.50579
7	Linear Regression	—	0.00520	-0.02531

Table 4.1: Comparison of Optimized Models

Analyzing the residuals (i.e., the differences between the actual and predicted values) offers

valuable insights into the strengths and weaknesses of each model. Residual patterns help assess how well the model captures the underlying data structure:

- **Linear Regression:** Residuals are widely scattered and lack a clear pattern, indicating that the model fails to account for nonlinear relationships and complex interactions in the data.
- **K-Nearest Neighbors (KNN):** The residuals are moderately dispersed, reflecting some ability to capture local nonlinearities. However, the performance remains limited due to sensitivity to data density and feature scaling.
- **Decision Tree, Bagging, Random Forest:** These models exhibit more compact residual distributions, suggesting improved generalization and a better fit to the training data. Ensemble methods like Bagging and Random Forest help reduce variance and mitigate overfitting.
- **Gradient Boosting, XGBoost:** These models produce residuals that are tightly concentrated around zero, with minimal spread. This indicates strong predictive accuracy and the ability to iteratively correct previous errors, making them well-suited for capturing complex nonlinear patterns.

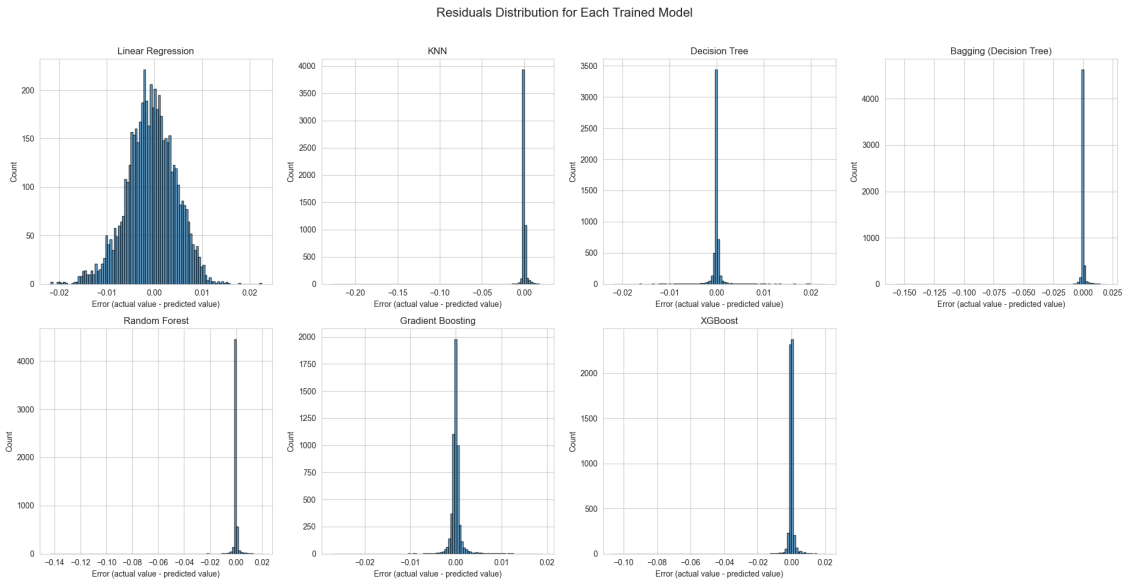

Figure 4.1: Residuals distribution for each trained model

Analyzing the comparison between predicted and actual values provides insight into the ability of each model to approximate the true target values. This evaluation reveals patterns of error, bias, and the general quality of predictions:

- **Linear Regression:** Exhibits a straightforward linear trend, but the presence of notable deviations from the actual values highlights its limited capacity to capture more intricate patterns in the data.
- **K-Nearest Neighbors (KNN):** While based on the proximity of data points, its predictions appear dispersed and inconsistent, indicating a sensitivity to local variations and potential noise.

- **Decision Tree:** Successfully identifies the underlying structure of the data. It strikes a reasonable balance between accuracy and generalization.
- **Bagging (Decision Tree):** Aggregates multiple decision trees trained on different subsets of the data, which improves robustness and prediction stability.
- **Random Forest:** An extension of Bagging that introduces additional randomness when selecting features, further improving prediction reliability and reducing variance.
- **Gradient Boosting:** Builds trees sequentially, each correcting the errors of the previous one. This method results in highly accurate predictions and demonstrates excellent generalization capabilities.
- **XGBoost:** A more computationally efficient and regularized variant of Gradient Boosting. Although slightly less performant in this particular case, it remains a strong predictive model overall.

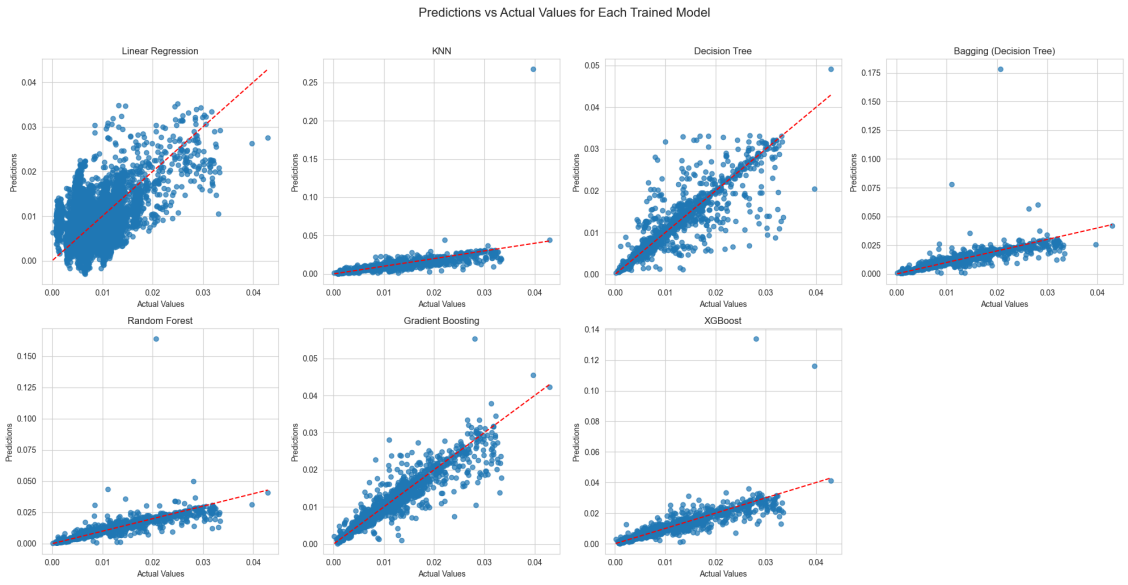

Figure 4.2: Predictions vs actual values for each trained model

Feature importance analysis quantifies the contribution of each input variable to the model's predictions. It serves to better understand model behavior and to identify which features carry the most predictive signal:

- **Linear Regression, KNN:** These models do not inherently provide feature importance scores. Linear Regression coefficients may offer some interpretability, but without normalization or regularization, their relative importance is not meaningful. KNN, being a non-parametric method based solely on instance similarity, lacks any mechanism to rank features.
- **Decision Tree:** This model evaluates importance by measuring the reduction in impurity contributed by each feature across all splits. It identifies `Speed12` (≈ 0.5) as the most influential feature, followed by `LapStartTime` (≈ 0.3) and `Speed11` (≈ 0.15), suggesting that speed-related inputs are crucial for accurate predictions.

- **Random Forest:** Feature importance is computed as the average decrease in impurity over all trees. The model corroborates the relevance of `Speedi2` (≈ 0.5) and `LapStartTime` (≈ 0.3), while also attributing minor but non-negligible weights to `LapNumber` and `SpeedI1` (≈ 0.05 each), hinting at some ensemble-level refinement over a single tree.
- **Gradient Boosting:** As an additive model that iteratively reduces prediction error, it derives importance from the contribution of each feature to the error reduction at each stage. Here, `SpeedI2` (≈ 0.45) and `LapStartTime` (≈ 0.3) remain dominant, while `SpeedI1`, `SpeedST`, and `LapNumber` (≈ 0.05 each) are also involved, suggesting a more nuanced use of multiple features.
- **XGBoost:** This implementation enhances Gradient Boosting through regularization and optimized tree construction. Interestingly, it assigns the highest importance to `LapStartTime` (≈ 0.6), with significantly smaller contributions from `SpeedI2` and `SpeedI1` (≈ 0.1 each). This shift may reflect XGBoost's tendency to favor features that yield early performance gains, possibly due to its greedy and regularized nature.

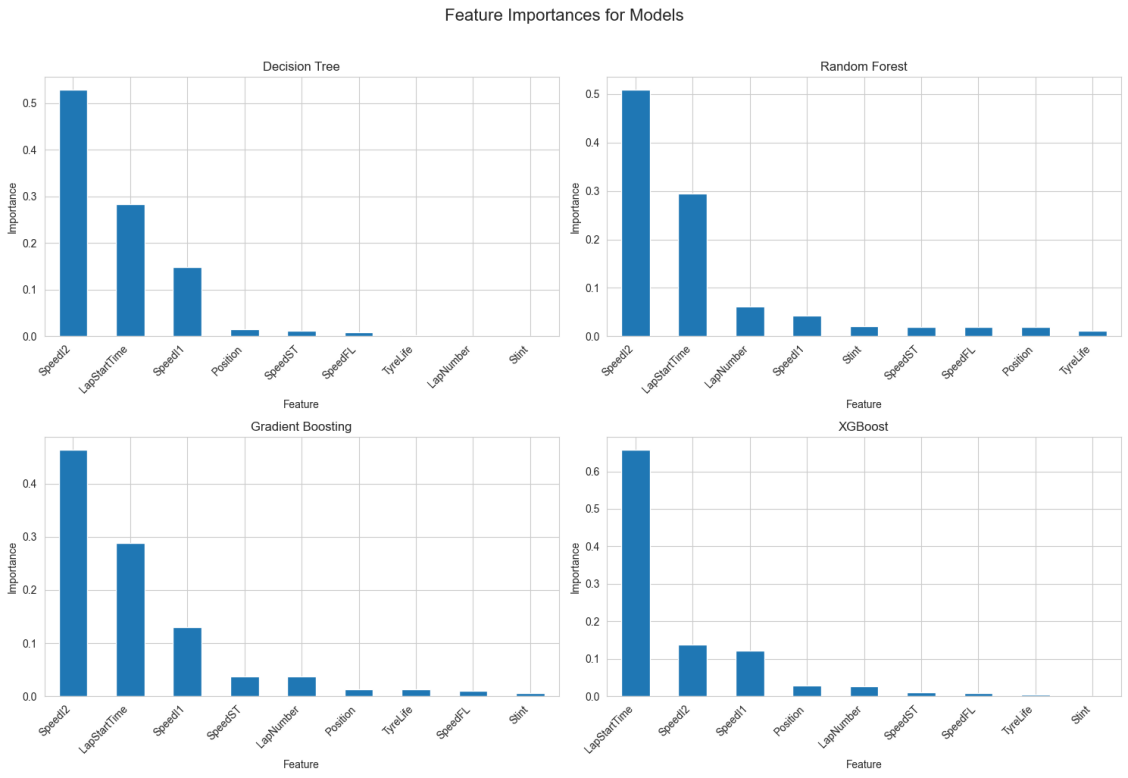

Figure 4.3: Feature importances for models

A heatmap of predictions from the best-performing models was generated to examine the similarity in their output patterns. This visualization provides insights into how closely the models agree with one another in terms of predicted values.

- **High Correlation ($\geq$ 0.85):** Models such as Decision Tree, Bagging, Random Forest, Gradient Boosting, and XGBoost show very strong pairwise correlations. This high agreement is expected as they are all based on decision tree architectures, which tend

to learn similar hierarchical partitionings of the feature space, even if ensemble methods introduce variations or improvements.

- **Moderate to High Correlation (0.72 to 0.86):** K-Nearest Neighbors (KNN) displays moderate to strong correlation with the tree-based models. Although KNN operates on a fundamentally different principle—prediction by similarity in the input space—it may still align partially with tree-based predictions if the data exhibits local patterns that both methods can exploit. However, the lower bound of correlation here also indicates that KNN diverges more often than ensemble methods.
- **Low Correlation (0.45 to 0.53):** Linear Regression exhibits notably lower correlation with all other models. This reflects its simplicity and its limitation to capture non-linear patterns. While tree-based models can adapt to complex, non-linear relationships in the data, Linear Regression relies solely on additive linear combinations of inputs, which may fail to approximate the underlying target function accurately.

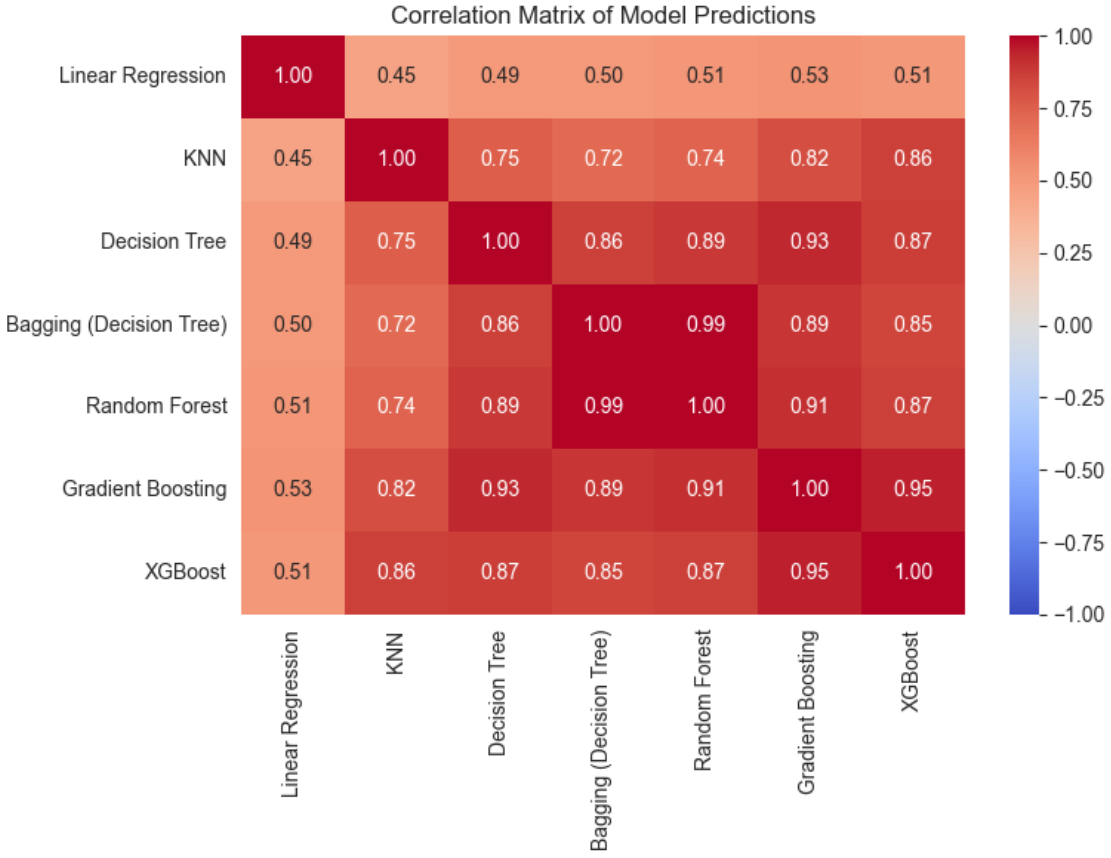

Figure 4.4: Correlation heatmap of models

Learning curves are valuable diagnostic tools that illustrate how a model's performance evolves as a function of training size or complexity. By plotting metrics such as the Root Mean Square Error (RMSE) for different subsets, we can infer whether the model is underfitting, overfitting, or generalizing appropriately. Specifically, if the RMSE on the training set is lower than both validation and test RMSEs, the model is likely generalizing well. Conversely, if the training error is higher than the validation and test errors, this may suggest overfitting. In all

other cases, the model tends to be well-regularized. Based on these comparisons, we draw the following observations:

- **Linear Regression:** The model produces comparable errors across training, validation, and test sets. However, the predictive performance remains limited, suggesting that the model may not fully capture the complexity of the underlying data.
- **K-Nearest Neighbors (KNN):** The model fits the training data perfectly, but generalization remains reasonable on both validation and test sets. This suggests that the model leverages local structure effectively without overfitting. The performance implies strong learning capacity with acceptable generalization.
- **Decision Tree:** While the model achieves perfect accuracy on the training set, it maintains strong predictive performance on unseen data. This indicates that, despite its flexibility, it generalizes well in this scenario.
- **Bagging (Decision Tree)**: The ensemble achieves low and consistent error across all data subsets. This consistency highlights the effectiveness of the method in reducing variance and enhancing model robustness.
- **Random Forest:** The model delivers results similar to the Bagging ensemble, with good generalization and stable performance. The added randomness introduced at the feature selection level contributes to its predictive strength.
- **Gradient Boosting:** The model fits the training data very closely while still maintaining strong generalization. The performance across validation and test sets indicates that boosting successfully improves learning without overfitting, thanks to its stage-wise corrections and regularization.
- **XGBoost:** The model performs consistently well across all datasets. Its integrated regularization mechanisms contribute to stable and reliable predictions, even in the presence of complex patterns.

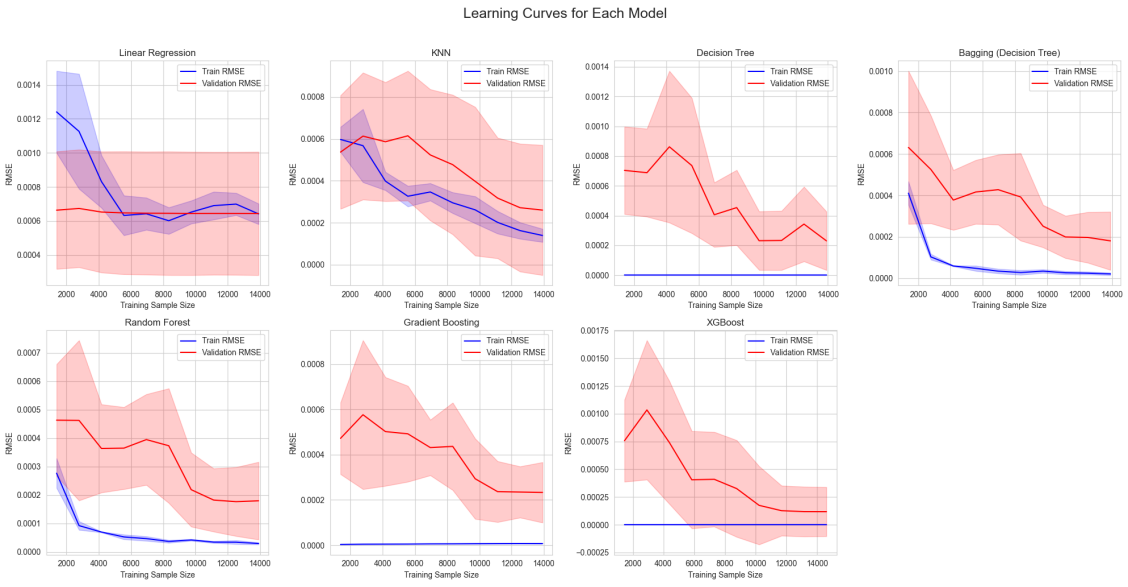

Figure 4.5: Learning curves for each model

4.2 Possible Improvements

Although the results obtained are quite satisfactory—particularly given the very low correlations observed between the various features and the lap time, as well as the exhaustive use of nearly all standard machine learning models taught in the Machine Learning and Machine Learning Project / Data Science courses—it is important to note that achieving substantially better performance may be inherently difficult. The limited predictive power of the input features imposes a natural ceiling on what can be expected, even with more advanced modeling techniques.

That said, several other algorithms and modeling strategies were also explored during the course of this project. However, as they did not lead to notable improvements, they were not included in this document. One alternative that could potentially offer marginal gains involves the use of neural networks. A Multilayer Perceptron (MLP) Regressor was tested in the final stages of the project. While MLPs are capable of modeling complex nonlinear patterns, this approach did not outperform the best tree-based models and was thus excluded from the main analysis.

Similarly, while extensive hyperparameter tuning was performed using Grid Search, it is possible that even broader searches might yield slight improvements for models such as Gradient Boosting or Random Forest. However, most of the impactful configurations have already been investigated, and only those yielding meaningful gains were documented.

Another more complex alternative is model ensembling through techniques such as Stacking Regressor, which integrates multiple base models to potentially enhance prediction accuracy. Although this strategy was briefly implemented, it did not result in improved performance in our setting and was therefore omitted from the final results. Moreover, its computational cost makes it less attractive for marginal gains.

In summary, while there may be room for minor improvements through deeper optimization or advanced architectures, the results achieved in this project are likely close to the upper bound of what can reasonably be expected given the limitations of the available data.

Chapter 5

Conclusion

This project has examined the application of supervised machine learning techniques to predict lap times in Formula 1 racing. Despite the limited size of the dataset and the relatively weak correlations between the available features and the target variable, several algorithms were rigorously implemented, compared, and analyzed.

The analysis showed that ensemble methods, particularly Gradient Boosting, provided the most accurate and stable predictions across training, validation, and test datasets. Classical models like Linear Regression and K-Nearest Neighbors (KNN), while interpretable and simple, were less effective due to the complex, nonlinear nature of the data. Decision Trees and ensemble techniques such as Bagging and Random Forests delivered improved performance, but the stage-wise correction mechanisms and regularization strategies in Gradient Boosting ultimately led to the best generalization.

It is important to note that the prediction performance could have been significantly higher if stronger correlations existed between the lap time and the other features. Additionally, three of the features with the strongest correlation to lap time—`sector1Time`, `sector2Time`, and `sector3Time`—were deliberately excluded. As mentioned earlier in this document, their sum is exactly equal to the lap time, and including them would have amounted to data leakage. In fact, preliminary experiments using these features led to near-perfect R^2 scores (around 0.98) even without overfitting, especially when using tree-based models. However, since the objective of this project was to predict the lap time and not to predict the individual sector times, these features were not used.

Although the best-performing model already achieved low prediction error, further experimentation was conducted to explore potential improvements. Neural network approaches, such as the Multilayer Perceptron (MLP), and model combination strategies like Stacking Regressor were tested, but failed to significantly outperform boosting-based models. Additionally, while extensive hyperparameter tuning using grid search was employed, only the configurations that yielded noticeable improvements were retained in the report.

Given the structure and quality of the available data, and the broad range of models tested—including many that were omitted from this report due to unsatisfactory results—it is likely that the performance achieved here is close to the best attainable under the current constraints. Future improvements could stem from access to richer datasets, including telemetry data, environmental conditions, or driver-specific characteristics. Nonetheless, this work demonstrates the strength of ensemble learning and the importance of informed feature selection and rigorous evaluation in applied machine learning.

References

Ergast Developer API (2024), 'Ergast motor racing data api'. (last accessed May 26, 2025).
URL: *https://ergast.com/mrd/*

Godefroy, B. (2025), 'Openf1: Real-time and historical formula 1 data api'. (last accessed May 26, 2025).
URL: *https://openf1.org/*

Schaefer, P. (2025), 'Fastf1: Formula 1 data analysis library'. (last accessed May 26, 2025).
URL: *https://docs.fastf1.dev/*

Appendix A

Extra Visual Insights

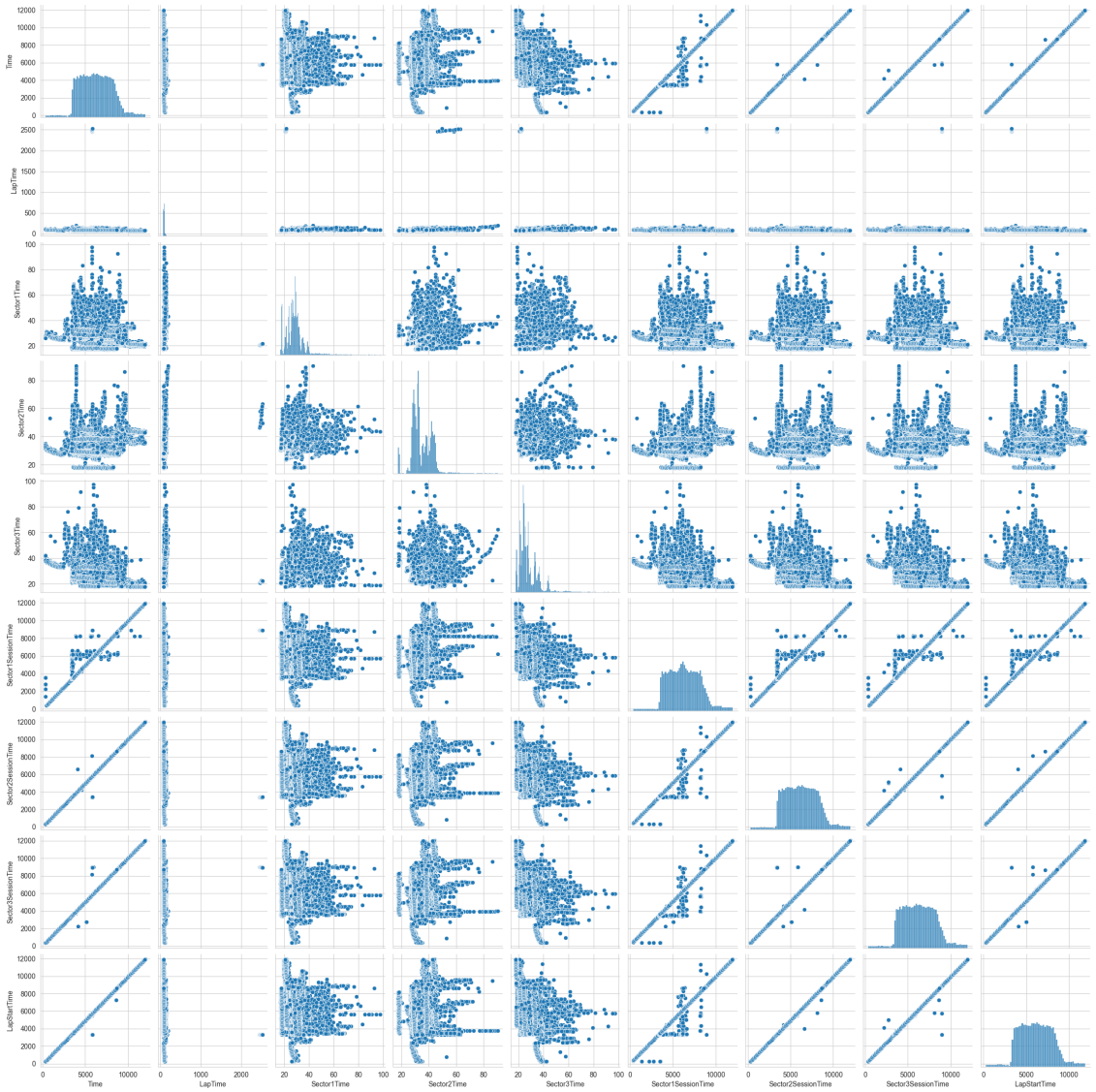

Figure A.1: Pair plot of time-related features

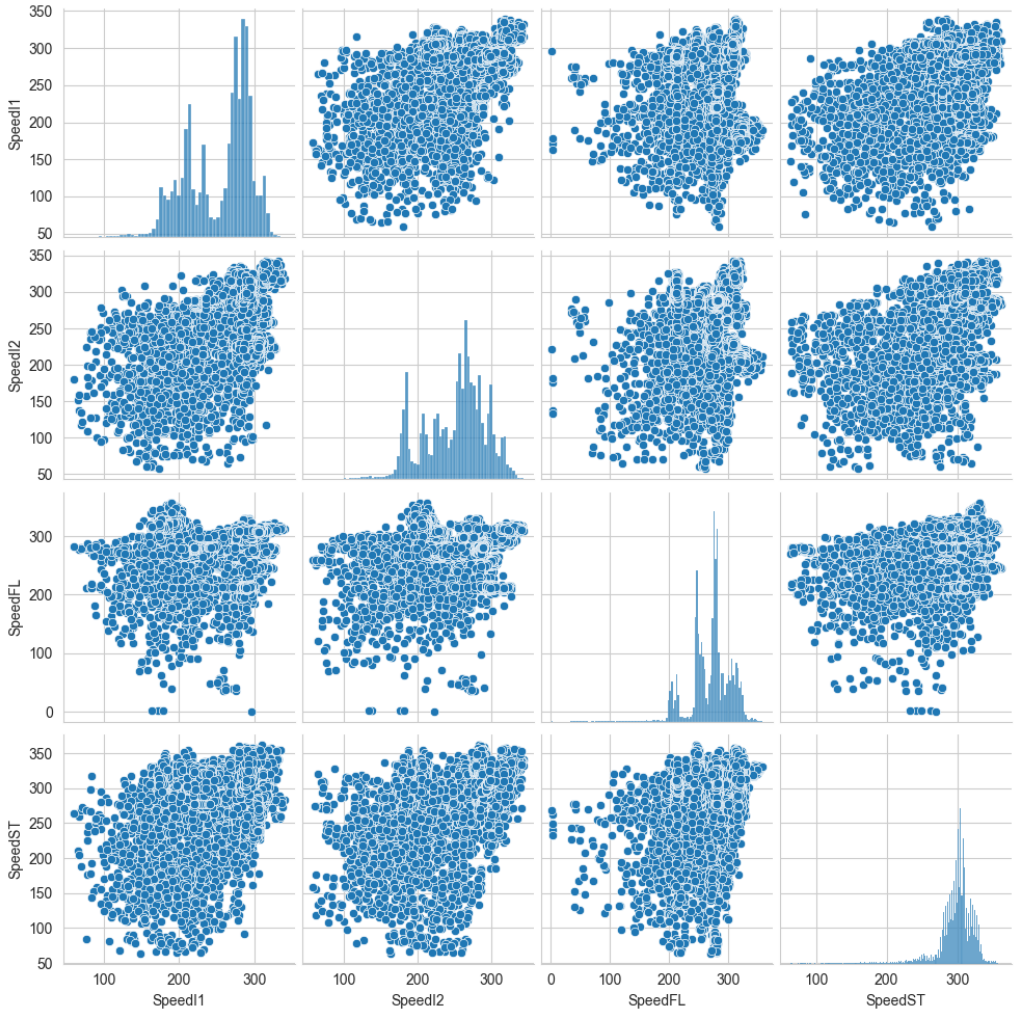

Figure A.2: Pair plot of speed-related features

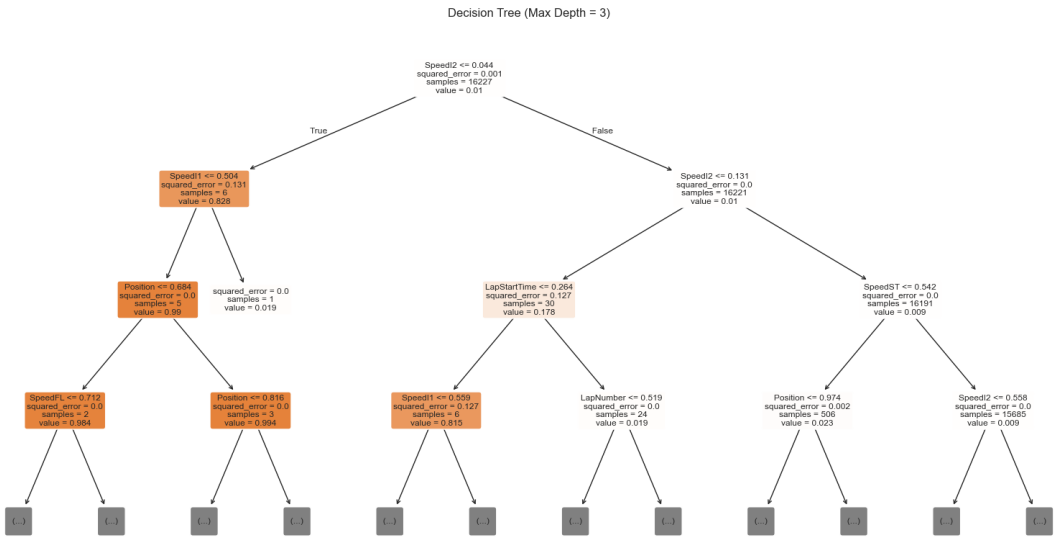

Figure A.3: Optimal Decision Tree with a maximum depth of 3

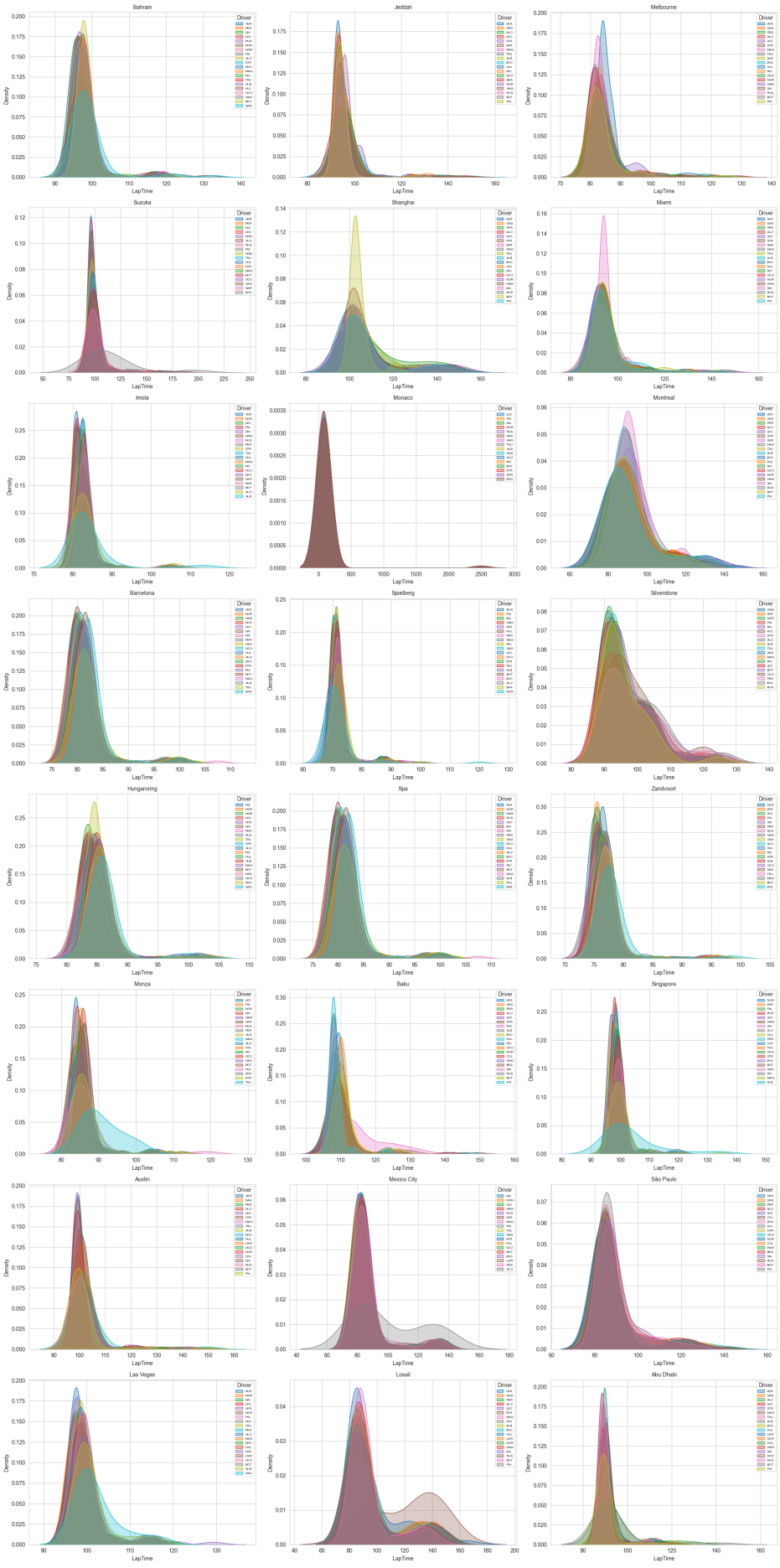

Figure A.4: Lap time distribution of drivers across the 2024 Grand Prix season