# **Hania Mekky|211000389|Machine Learning Project**

**Aim of the model:** To analyze formula 1 Laps from different perspectives such as speed of first and second sections, tire compound, and tire life during the race to predict the final lap speed.

**Dataset’s brief description:** the data contains multiple columns regarding the driver, constructer team, and lap results. This includes team names, drivers’ first and last names, pit stop in, pit stop out, speeds and others.

**Link of my presentation video:**  [ML F1 project.mp4](https://nileuniversity-my.sharepoint.com/:v:/g/personal/h_ahmed2189_nu_edu_eg/EVdYbzb42z1CpV34b8BHFKgBRc1IOaJ8Vqu2lkZPIkbCkQ?e=VnbgTS&nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJTdHJlYW1XZWJBcHAiLCJyZWZlcnJhbFZpZXciOiJTaGFyZURpYWxvZy1MaW5rIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXcifX0%3D)

# **Step 1**

A screenshot of a computer

Description automatically generated

In my project, I began by importing essential libraries, namely `pandas` for data manipulation, `seaborn` for visualization, and `matplotlib.pyplot` for plotting. I loaded the dataset from "Laps.csv" into a DataFrame named `df\_notcleaned` and proceeded to drop unnecessary columns such as 'IsPersonalBest', 'IsAccurate', 'PitOutTime', and several others to streamline the dataset. To handle categorical data in the 'Compound' column, I mapped the tire compounds ('MEDIUM', 'HARD', 'SOFT') to numeric values (1, 2, 3). I then addressed missing values by removing any rows containing NaN values. To further process categorical variables, I employed one-hot encoding on the 'Team', 'Driver', and 'EventName' columns, converting them into numerical format while avoiding multicollinearity by dropping the first category. I introduced a new feature, 'SPEE\_DIFF', representing the difference between the speed at the finish line ('SpeedFL') and the start line ('SpeedST'), to enhance the dataset's analytical value. After printing the first few rows of the cleaned DataFrame to verify the results, I saved the processed data to a new CSV file named "df.csv" and returned the cleaned DataFrame for future use.

# A screenshot of a computer Description automatically generated**Results**

A screenshot of a computer

Description automatically generated

# **Step 2**

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

To explore and visualize the dataset, I first imported the necessary libraries: `pandas` for data handling, and `matplotlib.pyplot` and `seaborn` for plotting. I loaded the cleaned dataset from "df.csv" into a DataFrame named `df`. I began by displaying basic information about the dataset using `df.info()` and descriptive statistics for numerical and categorical features using `df.describe()` with the `include=['object']` parameter for categorical data. To visualize the distribution of numerical features, I plotted histograms using `df.hist()` and boxplots with `sns.boxplot()`, which provided insights into the central tendency and variability of these features. I further examined the distribution of specific features, 'SpeedI1' and 'SpeedI2', by plotting their histograms side by side. To understand the relationship between 'SpeedI1' and 'SpeedI2', I created a scatter plot, which revealed any correlations or patterns between these variables. Lastly, I calculated and printed the mean values of 'SpeedI2' grouped by 'SpeedI1' to summarize their average relationship. This comprehensive EDA and visualization process allowed me to gain a deep understanding of the dataset's structure, distributions, and relationships among key features.

# **Results**

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedA graph with numbers and lines

Description automatically generatedA graph of different types of numbers

Description automatically generated with medium confidence A blue dots on a white background

Description automatically generated A graph of a graph of a number of blue and white bars

Description automatically generated with medium confidence

# **Step 3**

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

To preprocess the dataset and evaluate various machine learning models, I first loaded the dataset from "df.csv" into a DataFrame and converted all columns to numeric, coercing any errors to NaN values. I selected 'SpeedI1', 'SpeedI2', 'Compound', 'TyreLife', and 'FreshTyre' as features (X) and 'SpeedFL' as the target variable (y). I split the data into training and test sets. To handle missing values, I used a SimpleImputer with a mean strategy to fill NaNs. I standardized the features using StandardScaler to ensure all features were on the same scale. I then initialized several machine learning models: Logistic Regression, Random Forest, Gradient Boosting, K-Nearest Neighbors (KNN), and Decision Tree. I trained each model on the training data and evaluated their performance on the test data using accuracy as the metric. The accuracy scores for each model were stored in a dictionary and printed, providing a comparative overview of model performance. This process ensured that the data was clean, standardized, and that multiple models were evaluated to identify the best-performing one.

# 

A screenshot of a computer

Description automatically generated

To perform feature selection, I initialized a Gradient Boosting classifier and used Recursive Feature Elimination (RFE) to select the top features. I first instantiated the Gradient Boosting classifier and then initialized RFE with this classifier. After fitting RFE to the training data (`X\_train` and `y\_train`), I retrieved the selected features using `rfe.support\_` and printed them. To further analyze feature importance, I trained the Gradient Boosting classifier on the training data and obtained feature importance scores using the `feature\_importances\_` attribute. I created a DataFrame to visualize these scores, sorted the features by importance, and selected the top 10 features based on their importance values. I then printed these top features, providing a comprehensive view of the most significant features according to both RFE and Gradient Boosting feature importance. This dual approach ensured robust feature selection by combining the strengths of RFE and model-based importance scoring.

# **Results**

A screenshot of a computer

Description automatically generated

Unfortunately, results were very low and this will be discussed in the limitations section.

# **Step 4**

# A screenshot of a computer Description automatically generatedA screenshot of a computer Description automatically generated

To optimize the Random Forest classifier, I first loaded the dataset from "df.csv" and converted all columns to numeric values, handling any errors by converting them to NaN. I then addressed missing values using a SimpleImputer with a mean strategy to fill any NaNs in the training and test sets. I defined a parameter grid for RandomizedSearchCV, specifying a range of values for key hyperparameters such as 'n\_estimators', 'max\_depth', 'min\_samples\_split', 'min\_samples\_leaf', and 'bootstrap'. I initialized the Random Forest classifier and set up RandomizedSearchCV with 50 iterations and 3-fold cross-validation, utilizing all available CPU cores for efficiency. After fitting RandomizedSearchCV to the training data, I identified the best hyperparameters and trained the Random Forest model with these optimal settings. I further assessed the model using a confusion matrix and classification report to provide a detailed performance breakdown. Finally, I saved the best-performing model to a file named "best\_random\_forest\_model.pkl" using the `joblib` library, ensuring the model can be easily loaded for future use.

# **Results**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

# **Step 5**

A screenshot of a computer program

Description automatically generated A screenshot of a computer program

Description automatically generated

To deploy the F1 Final Lap Prediction model using Streamlit, I started by loading the trained Random Forest model from the file "best\_random\_forest\_model.pkl" using `joblib`. I defined a prediction function that takes input features, uses the loaded model to predict the final lap speed (`SpeedFL`), and returns the prediction. For the Streamlit app configuration, I set the page title to "F1 Final Lap Prediction" with a race car icon and an initial collapsed sidebar state. On the home page, I prompted users to input various race statistics such as speeds in sections 1 and 2, tire type, tire life, and whether the tires are fresh. The `FreshTyre` input was converted to a numeric value for model compatibility. These inputs were concatenated into a feature array for prediction. Upon clicking the "Predict" button, the app used the prediction function to display the predicted final lap speed. Additionally, the app included an "About" section explaining its purpose and a "Contact" section with contact information. Lastly, I ensured the application could read data from the CSV file "df.csv" to maintain consistency with the dataset used during model training. This setup in Visual Studio Code enabled a user-friendly and interactive deployment of the F1 prediction model using Streamlit.

# **Results**

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

**REPORT ANALYSIS & CONCLUSION**

# **1.Model Performance**

**The accuracy of the models was as follows:**

- Logistic Regression: 0.09

- Random Forest: 0.16

- Gradient Boosting: 0.15

- K-Nearest Neighbors: 0.14

- Decision Tree: 0.14

The Random Forest model achieved the highest accuracy of 0.16, indicating that it was slightly better at predicting the final lap speed compared to the other models. However, the accuracy values for all models were relatively low, suggesting that none of the models were particularly effective at predicting the final lap speed accurately.

# **2.Confusion Matrix and Classification Report**

The confusion matrix and classification report for the best-performing model (Random Forest) revealed significant misclassification issues. The confusion matrix was mostly filled with zeros, indicating that the model frequently failed to predict the correct class for the final lap speed. The classification report showed poor precision, recall, and F1-scores across almost all speed categories, with many categories having zero support (i.e., no true instances of those categories).

# **3.Interpretation**

**The low performance of the models can be attributed to several factors:**

1. **Data Quality and Quantity:** The dataset was very huge and complicated, and it needed a lot of editing, it also contained a lot of format that made the coding harder and the run time longer as each code took up to 30 minutes to run.

2. **Feature Selection:** The features used (SpeedI1, SpeedI2, Compound, TyreLife, FreshTyre) may not have been sufficient or the most relevant for accurately predicting final lap speed. Other factors, such as track conditions, driver performance, and car setup, which were not included, might play a significant role.

3. **Model Complexity:** More complex models might be needed to capture the intricate relationships between the features and the target variable. The models used might have been too simplistic for this task.

4. **Imbalanced Data:** The classification report indicated an imbalance in the distribution of speed categories, with some categories having very few instances. This imbalance can negatively impact model training and performance.

# **4.Limitations**

1. **Limited Features:** The model was built using only a few features, potentially missing out on other important variables that affect final lap speed.

2. **Imbalanced Dataset:** The imbalance in the speed categories likely contributed to the poor performance, as the model struggled to learn and predict the less frequent categories.

3. **Model Generalization:** The low accuracy indicates that the model may not generalize well to new, unseen data, limiting its practical application in real-world scenarios.

4. **Evaluation Metrics:** Accuracy alone may not be sufficient to evaluate the model's performance. Other metrics such as precision, recall, and F1-score provided a more comprehensive view but also highlighted the model's deficiencies.

# **5.Conclusion**

In conclusion, the current predictive models for final lap speed in F1 races showed low accuracy and poor performance across multiple metrics, highlighting the need for significant improvements to develop a reliable predictive model. Despite these limitations, the model and application functioned as intended, demonstrating successful deployment using Streamlit and enabling user interaction and model testing. The limited features used, the imbalance in the dataset, and the potential complexity of the models contributed to the poor results. Future work should focus on enhancing the dataset by collecting more data and incorporating additional relevant features, addressing data imbalance through techniques like oversampling or synthetic data generation, and exploring advanced machine learning and deep learning models. The successful deployment provides a solid framework for future enhancements, allowing ongoing refinement and development of a more accurate predictive model.