Formula 1 Race Prediction

Petar Stamenkovic — Aleksa Mitrovcan

April 2025

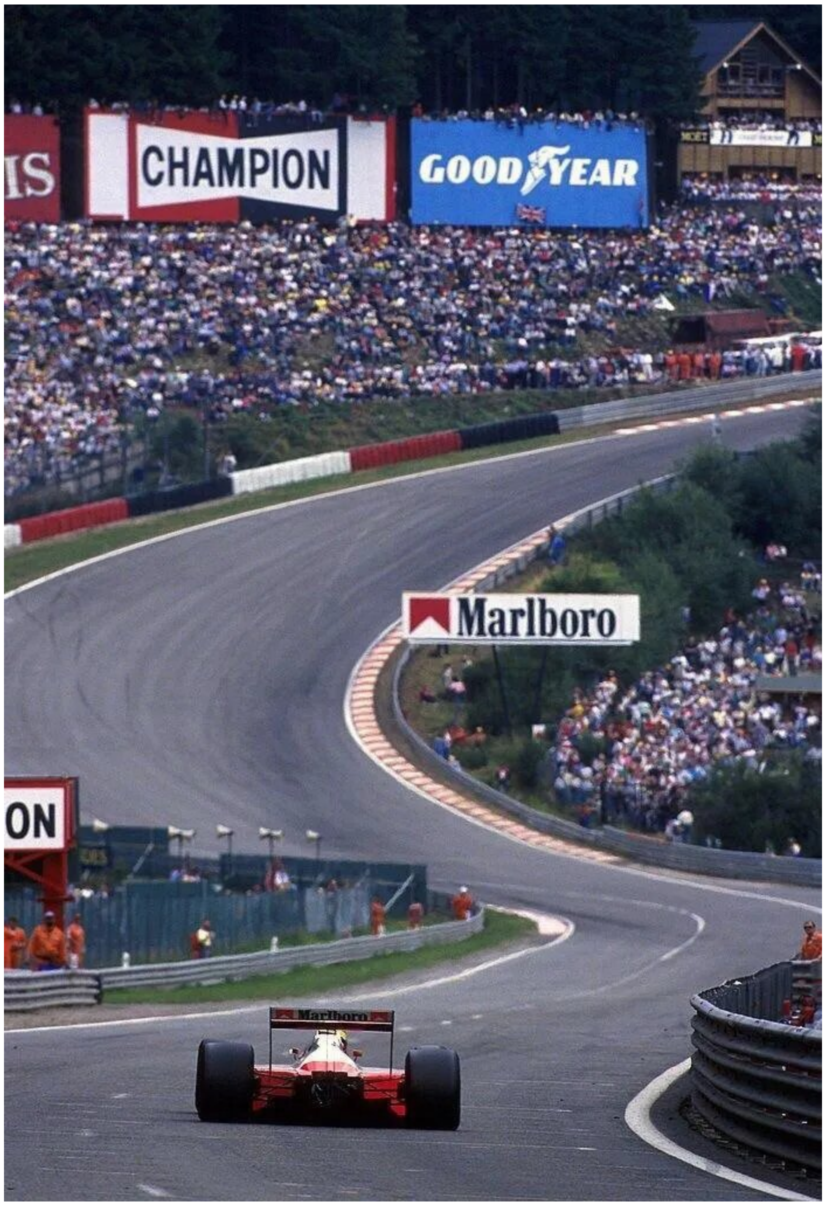


Figure 1: Ayrton Senna - Spa 1988

1 Introduction

This project features another crossover between Formula 1 and machine learning, and it marks a second collaboration between Aleksa and me. In this project, we developed two machine learning models, one to predict the final position of each driver at the end of the race and one to predict the fastest lap of the race, again, for every driver. This time we did not create a data set from scratch, but instead we used a *fast f1* Python package to retrieve the necessary data for calculations.

We wrote the code inside a Jupyter Notebook file, where you can find a brief explanation of all steps, but we advise you to look into this paper for better understanding.

2 What will you need?

As mentioned, the programming language used in this project is **Python**, integrated in **Jupyter Notebook** file. You should also import **fast f1** package for Python, and various graph drawing packages. You can find which in the actual notebook file. Interest in Formula 1 is, of course, beneficial but not necessary for understanding this project.

The first machine learning model does the prediction of final driver position, by the end of the race. It uses Gradient Boosting, specifically XGBoost. The second machine learning model does the prediction of the fastest lap for every driver on the grid, and it is a Random Forest model. While we won't dive into details about either of them, we suggest you to do a little bit of research in case you want to optimize it. Here you will find only basic information about both algorithms. Finally, some basic concepts of programming and data structures are necessary, but we assume that you have that, since you are here, reading this paper.

Finally, for the evaluation of the model we will use a well-known metric, MAE also known as Mean Absolute Error.

Figure 2: Jim Clark - Silverstone 1967

3 Jupyter Notebook file

In this section, we will cover a Jupyter Notebook file from start to end, and explain everything in it. When it comes to Jupyter, you can run it either from something like Visual Studio Code, or from terminal. Once you run it, it will open up a repo with all the files attached to this Notebook file. You want to open a *.ipynb* file and start editing it (*or running it*).

3.1 Importing packages

As always, we begin by importing all the necessary packages. We use mentioned *fast f1* package along with the *panda* for necessary Data Frame manipulation and *sklearn* for machine learning model training and creation. In this section, you can also find *cache* enabling for Fast F1. This allows us to store fetched data for a required race in order to have a faster access to it in future usage. We import our Driver table, that contains information about experience of each driver, measured in the number of races. Also, here we select which race data we want to fetch.

3.2 Necessary function declaration

In this part of code, we declared three functions for better scalability, in case we want to add more years into training. First function *race_exist* is created because Fast F1 will autocorrect your input in case that selected race did not happen in selected year. For example, Chinese Grand Prix did not happen in 2022, and in case you try to fetch data for it, Fast F1 will autocorrect it to Japanese Grand Prix. Luckily, Fast F1 has a function *get_event_schedule* that can retrieve all events during one year. At this point, we simply check whether our race input happened. If it did not, we just skip it.

The second function is called *get_race_data*, and it is the heart of the data fetching process. API call *get_session* loads all data for a selected session that we need to reorganize for our purposes. Also, to increase the accuracy of the model, we fetched the weather data, and calculated average value for temperature, humidity and rain chance. What does *average* in this context? Fast F1 returns a weather information for every minute of the session, so in order to make it compatible with our data frame, we calculate one, average value. After all merges, we have a data frame for the race from the selected year.

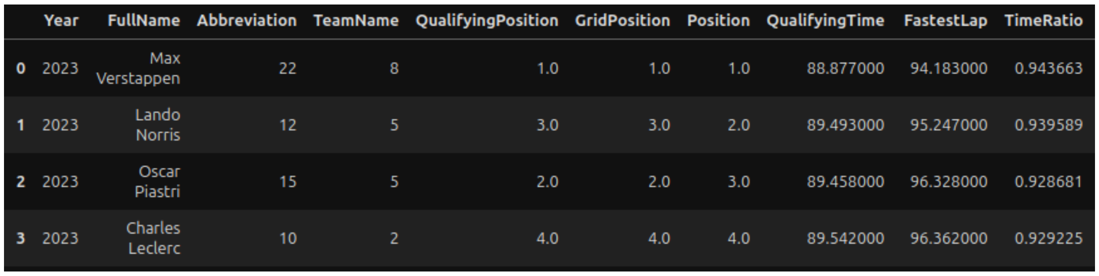
The third and final function *get_qualifying_data* is similar to a race one. Here we fetch qualifying times and grid positions for every driver.

After all this, we call our functions, merge our data frames and add the driver experience information to our final data frame, called **final data**. Note that we added couple of features, like time ratio. Features will be explained later in this paper. Also, we needed to fix up some values in our data frame. Even though the core of the teams did not change throughout the years, their names did, so we needed to address those changes for the model to have more accurate information. Team Alfa Romeo changed its name to Kick Sauber and team Alpha Tauri changed its name to RB.

3.3 Encoding, NaN resolving and scaling

Since machine learning models do not work with *string* values, we need to change them to numbers. This process is called **encoding**, and it can be done manually, as we did with drivers or by importing a specific encoder like *Label Encoder*, as we did for teams. After encoding, our new data frame is called ***final_data_encoded***.

Once you fetch data this way, there is a possibility that some data might be missing. This is usually marked with a **NaN**, and it creates issues for our model. This can be resolved again, either manually, in case that there are a few missing fields, or with code, replacing those fields with mean, average, min or max value from that column. For this project, mean value makes most sense, so we did it that way! Finally, we add some more features and finalize data frame creation. Once we verify that there are no missing NaN values and that all important training data is in **number** format, we can move on to the next step.



	Year	FullName	Abbreviation	TeamName	QualifyingPosition	GridPosition	Position	QualifyingTime	FastestLap	TimeRatio
0	2023	Max Verstappen	22	8	1.0	1.0	1.0	88.877000	94.183000	0.943663
1	2023	Lando Norris	12	5	3.0	3.0	2.0	89.493000	95.247000	0.939589
2	2023	Oscar Piastri	15	5	2.0	2.0	3.0	89.458000	96.328000	0.928681
3	2023	Charles Leclerc	10	2	4.0	4.0	4.0	89.542000	96.362000	0.929225

Figure 3: Example of correct part of table

The last step before we get to model training is data scaling. Machine learning models like it when all data is in similar range, it makes it easier for them to find patterns and therefore, make more accurate predictions. In this project, we used two scalers, *StandardScaler* and *MinMaxScaler*, for columns that have values with wider range, like qualifying time, humidity, temperature and number of races. Here, after creating both scalers, we just need to use their *fit_transform* functions and we have better data.

Notice that throughout the project we used various methods of encoding, mapping and models. This is not as optimal as it can get, and the only reason we did this is to get to know what options we have when it comes to machine learning.

Figure 4: Jackie Steward - Jarama 1970

3.4 Machine learning model - Creation and training

Finally, the most important section of the project, machine learning models. As mentioned, we used both Gradient Boosting (*XGBoost*) and *Random Forest* models. Initially, we planned to use XGBoost for both predictions, but then we decided to try the combination of both. In this section, we will talk about our models, the way that they are trained, splitting of the data, but firstly, features!

3.4.1 Features

Features are the core part of a machine learning model. They are the main topics for the model from which it finds patterns, and later on, makes a prediction, based on a target we provide. For the first model, target is **Position**, and for the second one **Time Ratio**. Let's talk about features for our models. They vary just a bit for each model, but mostly the point is the same.

- **Abbreviation** - This is the shortcut for a driver's name, and it represents the *driver's skill* as long as the model is concerned. It will predict better results for VER than BEA, since Max Verstappen (VER) is a better driver than Oliver Bearman (BEA), sorry to all Ollie's fans.
- **YearWeight** - In Formula 1, technology changes very fast. Each year, laps timer are lower and cars are faster. Initially, we started out with a year gap from 2022 to 2024, and this feature is supposed to suggest that years closer to 2025 should have more impact than the 2022. In the project, we removed a 2022 call, since the model had issues with it, but if you are eager to optimize and find the adequate combination of year weights, go for it!
- **TeamName** - Similar to a driver, this feature represents the driver's team. Again, for example, Red Bull is much better than Kick Sauber, so model can use this information to place driver on a better position, even though he might be a beginner in a Red Bull team. This is a rookie problem, data for them is not available in previous years, so we needed to manage them differently, with some abstractions.
- **GridPosition** - Represents the postion from which position did the driver start the race.
- **QualifyingTime** - Represents the fastest lap of a qualyfing session for a driver on a selected event.
- **Temperature, Humidity and Rain** - These features represent basic weather parameters for the event.
- **NumberOfRaces** - Represents the experience of the driver in the number of race starts. The bigger the number, more experience does a driver have.
- **TimeRatio** - Represents the division product of qualyfing time and fastest lap time. This shows how well does a driver convert his qualyfing speed into a race speed.
- **MeanSpeedTrap and MeanTimeRatio** - Since we do not know what will be a speed trap for each driver of his time ratio, we calculate the average values for each driver(*from prevous years*) and include those values in our prediction sample.

In the final data frame, you can also find a full name for the driver, but that column is there for readability purposes only.

3.4.2 Machine learning models

For predicting, we used a well known procedure when it comes to machine learning models.

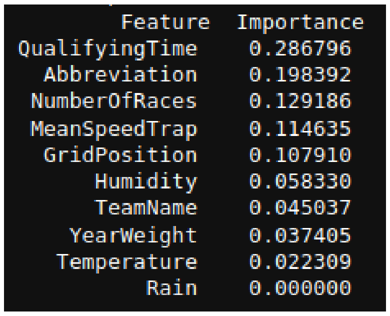
1. List out all the necessary features
2. Declare a target variable - Position
3. Divide a data set in training (X) and validation set (y)
4. Split the data using *train_test_split* function
5. Create a model with some standard parameters - GradientBoostingRegressor
6. Fit the model
7. Predict and evaluate

The second model procedure is fairly similar :

1. List out all the necessary features
2. Declare a target variable - TimeRatio
3. Divide a data set in training (X2) and validation set (y2)
4. Split the data using *train_test_split* function
5. Create a model with some standard parameters - RandomForestRegressor
6. Fit the model
7. Predict and evaluate

You might wonder why is the target in the second one Time Ratio and not Fastest Lap. I mean, we are predicting lap times, no?

You are absolutely right. However, due to a problem of rapid technology development, lap times are drastically different, like 6 or 7 seconds slower, which affect our model big time. So we decided to predict TimeRatio, since that is a parameter that does not depend that much on year we are in but more on the driver and his experience. As we mentioned, time ratio represents how well can a driver convert his qualyfing pace into a race pace, which is super important for this prediciton. Lap times are found by dividing new qualifying times with the predicted time radio, reaching out target. Below you can see the importance of each feature.



```
        Feature  Importance
QualifyingTime    0.286796
  Abbreviation    0.198392
 NumberOfRaces    0.129186
 MeanSpeedTrap    0.114635
  GridPosition    0.107910
      Humidity    0.058330
      TeamName    0.045037
    YearWeight    0.037405
   Temperature    0.022309
          Rain    0.000000
```

Figure 5: Feature importance

3.5 Prediction sample

Finally, it is time to input our real time parameters in a prediction sample. What will happen in a race in the future? Our sample consists of all the features we mentioned before, and we input them through a list. In *Abbreviation* list we input the driver codes from the fastest one to the slowest one from the 2025(*or any year your are today in*) qualyfing session. We do the same for all other features. For example, the fifth value in *QualyfingTime* is the fifth fastest time in the qualyfing session and it belongs to a fifth driver code in *Abbreviation* list.

Code following the prediction sample if fairly intuitive so we will skip the explanation of it. You can see how we manipulate the results in a resulting data frame, sort the data frame and format lap times.

Also, you can see there are few cells that are used to graphically present the times, position and driver improvment through years. This is **FIXED** and works only for the prediction sample we input (*Japanese Grand Prix 2025*), so in case you want to try that part of the code for yourself, change the values in prediction samples.

4 Results and outro

After everything we saw, what did we create and how well did we do it? Well, graphs can show you for a specific race that we tested it on. MAE was around 2.96 for positions and around 0.1 for lap times. Of course it is not fantastic, but as always there is room to improve, and since both of us are just beginners we are on a mission to make this results even better with some new features.

Thank you for taking your time to read this and check out our project. We hope you found it insightfull and learned something. If you have some comments, critics or request for collaboration, you can find us through our GitHub and LinkedIn accounts. Let's code together, your Smooth Operators!

Figure 6: Charles Lecler - Monza 2019