

★ Get unlimited access to the best of Medium for less than \$1/week. [Become a member](#)



Building an AI to Predict F1 Race Outcomes: A Data Science Journey

Introduction:



Fer Sigüenza

Follow

6 min read · Oct 18, 2024

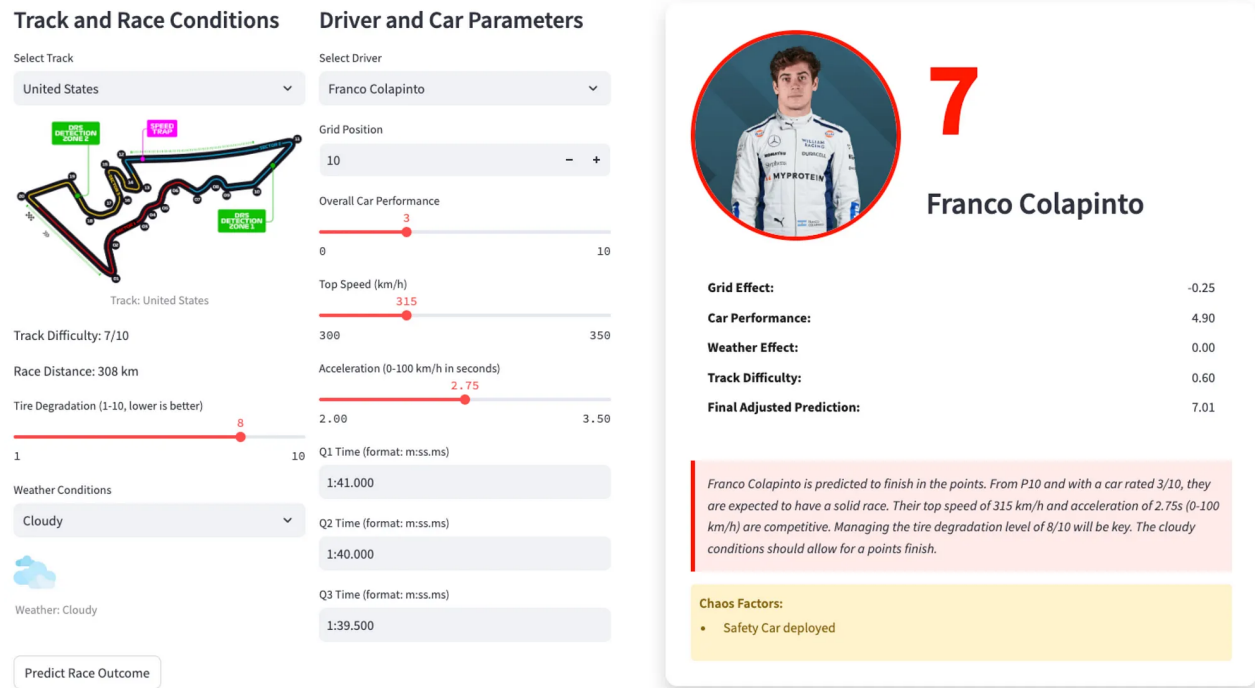


5



Formula 1 racing captivates millions worldwide with its blend of cutting-edge technology and human skill. I'm a big fan of F1 and tech in general. I've been working with Machine Learning and AI for a couple of years, but lately, the boom of generative AI with LLMs has sparked something new in me. I've been experimenting with creating an F1 race prediction model with some help of [Claude Sonnet](#) to optimize a couple of things.

In this article, I'll take you on a journey to show how I trained a custom model for the predictions and built a user interface with the [Streamlit](#) library. I've included several variables to make it even more interesting and interactive, here is a sneak peek at my tool:



The journey:

It all started with a casual chat with some colleagues who are also F1 fans. Now that **Franco Colapinto (Argentinian driver)** has joined the Williams Team, we were tossing around bets on how we thought he'd perform in the races. Then one of them said, "Hey, you're into all that AI stuff, right? Why don't you use it to build something so we can predict the races?" And here we go...

1. Data Collection and Preprocessing

Where do we start? Well...First things first, the data!!!

I downloaded a ton of F1 data available from [Kaggle](https://www.kaggle.com/), including driver stats,

race results, and track info for the past years. Using the pandas library in Python, I kicked things off by cleaning and merging these datasets:

```
import pandas as pd

drivers = pd.read_csv('f1dataset/drivers.csv')
results = pd.read_csv('f1dataset/results.csv')
races = pd.read_csv('f1dataset/races.csv')

merged_data = pd.merge(results, drivers, on='driverId').merge(races[['raceId'],
```

2. Data Normalization

So we've got all this cool F1 data, but we have many differences, some numbers are huge, others are tiny and this can confuse our model. That's where data normalization comes in. Here I'm using something called MinMaxScaler from the **sklearn** library.

In simple terms, It takes all our data — Finds the biggest and smallest values — Squishes everything between 0 and 1

Why are we doing this? Well, imagine trying to compare lap times (in seconds) with a driver's age, without the normalization, the model might think lap times are way more important just because the numbers are bigger and of course, we don't want that.

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

This code is basically telling our computer, **“Hey, take all this data and make it fair and balanced.”** Our model can now look at everything equally, without getting confused by big numbers.

3. Feature Engineering

Next, it was time to extract some key features like grid position, lap times, and pit stop data. These sets of information help our model get a grip on all the little things that can make or break a race.

By creating these features, we’re essentially adding our F1 know-how to the data, transforming raw numbers into something more meaningful that our model can easily understand and use. It’s like we’re telling the model, **“Hey, pay attention to these things these are the ones we think really matter in determining who crosses the finish line first.”**

```
input_data = pd.DataFrame({  
    'grid': [grid_position],  
    'lap_time_mean': [lap_time_mean],  
    'pit_stops_count': [pit_stops_count],  
    # ... other features  
})
```

4. Neural Network Architecture

Alright, now the hardest part, is the brain of our operation. I decided to use

TensorFlow and **Keras** to build a deep neural network. I decided to start with this approach because it gave me the possibility to create a digital brain with multiple layers (thanks to `keras.layers.Dense`), letting my model figure out all the tricky patterns in our F1 data:

Medium

Search

Write

6



```
model = keras.Sequential([
    keras.layers.Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1)
])
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

Ok, hold up, what's the deal with those numbers and weird words?

- The numbers (128, 64, 32) are like our AI's pit crew. We start with a big crew and gradually reduce it as we get closer to the final prediction.
- **relu (Rectified Linear Unit)** is a simple decision-maker for our AI. It keeps positive numbers and turns negatives to zero. This helps the AI focus on the important stuff and learn faster.
- **adam (Adaptive Moment Estimation)** is like our AI's race engineer. It tweaks how fast the AI learns, remembers past lessons, and helps focus on what's important. It's constantly optimizing our model's performance.

- **mse (mean squared error)** is our scoring system. It measures how far off our predictions are.

I chose these tools because they work well for lots of different problems without needing much tweaking, there are others, but these were a good starting point to get my system running faster.

Now the fun part, let's teach our model to predict!

5. Training and Validation

So, we've got our AI model set up and ready to go, but it's like a rookie driver — it needs practice!

Luckily we have the tons of data we already pre-processed.

```
early_stopping = keras.callbacks.EarlyStopping(patience=10, restore_best_weights=True)
history = model.fit(X_train_scaled, y_train,
                    epochs=200,
                    validation_split=0.2,
                    callbacks=[early_stopping])
```

Breaking down that code:

- **epochs=200:** Each epoch is one full pass through all the training data, like doing 200 laps on a track. **Why 200?** It's a decent starting point, enough time to learn, but not so much to be here all day.

- **validation_split=0.2:** We're holding back 20% of our data as a test. It's like having a separate track where we can check if our AI has really learned or if it's just memorizing stuff. **Why 20%?** It's a common split that usually works well.
- **early_stopping:** If our model isn't improving after 10 tries ('patience=10' means), it says, "Alright, that's enough, we're done here." It also makes sure we keep the best version of our model (that's the 'restore_best_weights=True' part).

Why do all this? We want our model to be good at predicting F1 races in general, not just memorizing so it can handle any track.

6. Prediction and Interpretation

The model is trained, and now we're gonna see if it can predict race outcomes or not.

```
def make_prediction(input_scaled):  
    base_prediction = model.predict(input_scaled)[0][0]  
    adjusted_prediction = base_prediction + grid_effect + weather_effect + track  
    return round(max(1, min(20, adjusted_prediction)))
```

Again all these numbers and strange words, English, please!

We feed our model a pre-race briefing of data and input some variables in our application.

The model gives us a '**base_prediction**'. This is its first guess, based purely on the numbers it's learned from.

But it is not that simple! there are a lot of variables that can affect a race result.

- **grid_effect**: Because where you start matters
- **weather_effect**: Rain or shine, it can change everything!
- **tyre_degration** and **track_difficulty**: Some tracks are trickier than others and might heavily impact the tires, and much more!

Here you can see all the parameters and configurations I added to make the prediction more realistic!

I have also added a **Chaos factor** that can affect our prediction, things like crashes, safety cars, etc.

Finally, we make sure our prediction makes sense and round it to a whole number because you can't finish in 3.7th place.

And voila! We've got our prediction, which it's not just a cold, calculated guess as it takes into account all the drama and excitement that makes F1 so thrilling.

Wrapping Up

We've built this F1 prediction model that mixes old-school racing knowledge with some fancy AI tech. Although the AI model is smart, but it can't predict when a driver's gonna make a wild move or when the weather's gonna change drastically.

Right now, this is just for fun. To get really serious, we'd need tons of data and some way to feed in live race information, like the teams do when they're figuring out pit stops on the fly.

What's next?

Fine-tuning pre-trained models and feeding live data to see how we can improve the performance of the results is definitely on my radar, but that's a story for another day.

Stay tuned, and may the best driver (or prediction) win!

AI

Formula 1

Machine Learning

Data Analysis



Written by Fer Sigüenza

10 followers · 6 following

Follow

No responses yet



George Ogden

What are your thoughts?

More from Fer Sigüenza

 Fer Sigüenza

Forgetting as a Feature and Distributed Cognition for Truly...

Introduction: The Monotony of Scale and Nature's Whisper

Jun 26



 Fer Sigüenza

Code Smarter, Not Faster: Essential Tips for Responsible...

AI Programming Assistants: A Double-Edged Sword

May 2

 10  1



 Fer Sigüenza

Building an App: Not Quite Like Building a Car... Or Is It?

A Journey of Parallel Passions: Exploring the Shared Language of Building


May 20

 7  1



See all from Fer Sigüenza

Recommended from Medium


 October

Machine Learning basics in 10 minutes

Machine learning is how machines learn to do stuff. Stuff could be predicting the next...

 Jun 3  54





 Robbie Dudzinski

Trackman Pitching Data—Are You Actually Using It?

There is a lot of data in baseball that is not being used by a majority of programs,...

Feb 24  4




 Abhinaba Banerjee 

MLFlow: An introduction

This blog discusses MLFlow briefly, the users who use it, reasons for usage, and some rea...


★ Jan 15 🖱 5

 Damini Vadrevu

What are Decision Trees, Random Forest and Gradient Boosting...

An all about Ensemble.


★ Mar 26 🖱 8

 In T3CH by Py-Core Python Programming

Python Time Travel with Event Driven Simulations (What It...

When COVID-19 began, flights carried the virus faster than governments could...

★ 5d ago 🖱 53 💬 1

 In Long. Sweet. Valuable. by Ossai Chinedum

I'll Instantly Know You Used Chat Gpt If I See This

Trust me you're not as slick as you think

★ May 16 🖱 16.3K 💬 925



See more recommendations

[Help](#) [Status](#) [About](#) [Careers](#) [Press](#) [Blog](#) [Privacy](#) [Rules](#) [Terms](#) [Text to speech](#)