

Final Project: Predict F1 pitstops

Overview:

In this project, I used the Formula One dataset on the 2025 Shanghai Grand Prix and output predictions around type degradation and pit stop (when to change type) strategies. This write-up has the following sections:

Section Name	Purpose
Dataset	Provide an overview of the used dataset, outlining the motivation for this project
data.rs	This module is responsible for handling the raw lap data. It reads the data from a CSV file, performs initial cleaning and processing, calculates useful metrics like time delta relative to a driver's best lap, and crucially, calculates the average stint lengths based on pit stop information and compound changes.
model.rs	This module focuses on building a mathematical model to predict tyre degradation. It uses processed lap data to train a linear regression model that estimates the time loss per lap based on factors like tyre life and track temperature.
strategy.rs	This module takes the degradation model and average stint length calculations and uses them to simulate different race strategies. It calculates the cumulative time loss for each strategy throughout a race, including pit stop time loss.

The overall strategy is:

**Data Cleaning and Conversion:** Reads raw lap data from a CSV file, filters out invalid entries, and converts the data into a structured format with standardized types and consistent compound names.

**Time Delta Calculation:** For each lap, calculates the time difference relative to the driver's fastest non-pit lap in the session. This “time delta” serves as a measure of performance loss and is used as the target variable in the degradation model.

**Average Stint Length Calculation:** Analyzes the processed data to identify individual tyre stints based on pit stops (pit-in and pit-out laps) and compound changes. It then calculates the average duration (in laps) for stints on each tyre compound (Medium and Hard) across all drivers.

**Degradation Modeling:** Uses the processed lap data (excluding pit laps) to train a linear regression model. This model predicts the expected time delta (degradation) for a given lap based on the tyre's life in the current stint and the track temperature.

**Strategy Simulation and Comparison:** Takes the calculated average stint lengths and the trained degradation model to simulate common race strategies. It calculates the cumulative time loss over the race distance for each strategy, including pit stop time penalties, and visualizes the results in a plot to compare their predicted performance.

## Dataset

The dataset is retrieved from the FastF1 Python Module. FastF1 is a popular database in Python that provides detailed information about a Grand Prix. It also allows the user to download customized datasets with specific columns. The one that I used looks like this:

Time	Driver	DriverNum	LapTime	LapNum	Stint	PitOutTim	PitInTime	IsPersonal	Compound	TyreLife	FreshTyre	Team	Position	LapTimeS	TrackTemp	AirTemp	Humidity	WindSpeed
0 days 01: PIA		81	0 days 00:	1	1			FALSE	MEDIUM	1	TRUE	McLaren	1	98.344	35.6	27.3	18	2.2
0 days 01: NOR		4	0 days 00:	1	1			FALSE	MEDIUM	1	TRUE	McLaren	2	99.268	35.6	27.3	18	2.2
0 days 01: RUS		63	0 days 00:	1	1			FALSE	MEDIUM	1	TRUE	Mercedes	3	100.282	35.6	27.3	18	2.2
0 days 01: HAM		44	0 days 00:	1	1			FALSE	MEDIUM	1	TRUE	Ferrari	4	100.792	35.6	27.3	18	2.2
0 days 01: LEC		16	0 days 00:	1	1			FALSE	MEDIUM	1	TRUE	Ferrari	5	101.763	35.6	27.3	18	2.2
0 days 01: VER		1	0 days 00:	1	1			FALSE	MEDIUM	1	TRUE	Red Bull R	6	102.249	35.6	27.3	18	2.2
0 days 01: ANT		12	0 days 00:	1	1			FALSE	MEDIUM	1	TRUE	Mercedes	7	103.008	35.6	27.3	18	2.2
0 days 01: TSU		22	0 days 00:	1	1			FALSE	MEDIUM	1	TRUE	Racing Bu	8	103.656	35.6	27.3	18	2.2
0 days 01: HAD		6	0 days 00:	1	1			FALSE	MEDIUM	1	TRUE	Racing Bu	9	104.196	35.6	27.3	18	2.2
0 days 01: OCO		31	0 days 00:	1	1			FALSE	MEDIUM	1	TRUE	Haas F1 T	10	104.831	35.6	27.3	18	2.2
0 days 01: ALB		23	0 days 00:	1	1			FALSE	MEDIUM	1	TRUE	Williams	11	105.273	35.6	27.3	18	2.2
0 days 01: ALO		14	0 days 00:	1	1			FALSE	MEDIUM	6	FALSE	Aston Ma	12	105.761	35.6	27.3	18	2.2
0 days 01: GAS		10	0 days 00:	1	1			FALSE	MEDIUM	1	FALSE	Alpine	13	106.04	35.6	27.3	18	2.2
0 days 01: STR		18	0 days 00:	1	1			FALSE	HARD	1	TRUE	Aston Ma	14	106.646	35.6	27.3	18	2.2
0 days 01: SAI		55	0 days 00:	1	1			FALSE	MEDIUM	1	TRUE	Williams	15	107.312	35.6	27.3	18	2.2
0 days 01: BEA		87	0 days 00:	1	1			FALSE	HARD	1	TRUE	Haas F1 T	16	107.687	35.6	27.3	18	2.2
0 days 01: DOO		7	0 days 00:	1	1			FALSE	MEDIUM	1	FALSE	Alpine	17	107.904	35.6	27.3	18	2.2
0 days 01: LAW		30	0 days 00:	1	1	0 days 00:59:12.7550		FALSE	HARD	1	TRUE	Red Bull R	18	108.928	35.6	27.3	18	2.2
0 days 01: HUL		27	0 days 00:	1	1			FALSE	MEDIUM	1	TRUE	Kick Saubr	19	109.895	35.6	27.3	18	2.2
0 days 01: BOR		5	0 days 00:	1	1		0 days 01:	FALSE	MEDIUM	1	TRUE	Kick Saubr	20	123.657	35.6	27.3	18	2.2

(Also feel free to explore this dataset in Shanghai2025.csv)

Some of the important columns:

- Driver: driver name
- Lap number: the current number of laps the driver has run
- Stint: the “period number” that represents the tyre compound strategy. For example, if a driver is running with the first set of tyres from lap 1 to lap 20, and then changes tyres to run from lap 21 to lap 56, 1-20 would be considered a stint 1, and 21-56 would be considered a stint 2.
- Pit-in and Pit-out: the time the driver pits (changes tyres). In the Rust program, we use this column to identify the stint changes for a driver.
- Compound: medium and hard, in our case
- Laptime: the lap time in seconds
- Environmental factors: track temperature, air temperature, humidity, and wind speed.

## data.rs

### 1. Purpose of the Module

The module encapsulates all the logic related to data input and initial preparation. Its function includes:

- Reading data from a specified CSV file.
- Deserializing CSV records into a structured format.
- Filtering out invalid or incomplete data points.
- Standardizing data formats (converting compound names to uppercase).
- Calculating derived metrics, such as the time delta relative to a driver's fastest lap.
- Grouping data by driver and ensuring laps are in chronological order.
- Identifying tyre stints based on pit stops and compound changes.
- Calculating the average length of stints for each tyre compound.

## 2. Data Structures

Two main data structures are defined within `data.rs` to represent the lap data at different stages of processing:

- `RawLapData`:
  - This struct is designed to match the column headers and data types in the input dataset. It uses `serde` to map the column names of the CSV file to Rust struct field names. Example: `#[serde(rename = "Driver")] driver: String`
- `struct ProcessedLapData`:
  - This struct represents a single lap after it has been cleaned and processed. It converts some of the numerical columns (lap number, tyre life) into `u32` whole numbers, as well as some string and boolean value conversions & validations.
  - One key addition here is the `time_delta`, an `f64` value representing how much slower or faster this lap was compared to the driver's fastest lap in the session.
  - This processed format of lap data points is used in the later modules.

## 3. Key Functionality

The `data.rs` module contains the `DataProcessor` struct and two main functions:

Name	Functionality
<pre>pub fn new&lt;P: AsRef&lt;Path&gt;&gt;(filename: P) -&gt; Result&lt;Self, Box&lt;dyn Error&gt;&gt;</pre>	<ul style="list-style-type: none"><li>• The <code>new</code> constructor of the <code>DataProcessor</code>. It initializes a CSV reader and opens the file. After loading the file, it would iterate through each record under the <code>deserialize()</code> method.</li><li>• It would apply a filter to filter out invalid or nonsensical data points from the raw data. For instance, if the track temperature is negative, that would be</li></ul>

	<p>nonsensical based on the geological location of Shanghai.</p> <ul style="list-style-type: none"> <li>• For valid laps, it would create a <code>ProcessedLapData</code> instance. All the lap records are stored in a hashmap, grouped by driver name.</li> <li>• For each driver's list of laps, it sorts them by lap number using <code>laps.sort_by_key( d  d.lap_number)</code>; Therefore, all the laps are now in chronological order (lap 1, lap 2, lap 3...)</li> <li>• Time delta: first, it finds the minimum laptime seconds among all the non-pit laps for the current driver (why non-pit? Since the lap time of the pitted lap is dramatically slower, it contains the pit-stop time, which would deviate our result). Then, for every lap, it calculates the time delta as <code>laptime seconds - min_laptime</code>. If no non-pit laps are found, <code>time_delta</code> is set to 0.0.</li> <li>• Time delta is later on being inserted into the hashmap with the driver name and chronological laps.</li> </ul>
<pre>pub fn calculate_average_stint_lengths(dat a: &amp;HashMap&lt;String, Vec&lt;ProcessedLapData&gt;&gt;) -&gt; (f64, f64)</pre>	<ul style="list-style-type: none"> <li>• This function takes a reference to the processed driver data as input. It initializes two vectors, <code>medium_stints</code> and <code>hard_stints</code>, to store the calculated lengths of individual stints for each compound across all drivers.</li> <li>• <b>Stint Identification Logic:</b> This is the core of the function. It iterates through a driver's laps and identifies the end of a stint based on several conditions: <ul style="list-style-type: none"> <li>○ The current lap is the last lap for the driver.</li> <li>○ The current lap is a pit-in lap (<code>is_pit_in_lap</code>).</li> <li>○ The <i>next</i> lap is a pit-out lap (<code>is_pit_out_lap</code>). This indicates the current lap is the last lap before entering the pit lane and exiting on new tires.</li> <li>○ The compound changes on the <i>next</i> lap, and that next lap is <i>not</i> a pit-out lap (handles</li> </ul> </li> </ul>

	<p>potential data inconsistencies or rare on-track tire changes not explicitly marked by pit times).</p> <ul style="list-style-type: none"> <li>• When a stint end is detected on the <code>current_lap</code>, the length of the completed stint is calculated as <code>curr.lap_number - start_lap + 1</code>. (Why adding 1? Since the first lap starts with 1 instead of 0)</li> <li>• The calculated <code>stint_length</code> (<code>len</code>) is then added to the appropriate vector <code>medium_stints</code> or <code>hard_stints</code> based on the compound of the stint (which was recorded at the start of the stint).</li> <li>• Average Calculation: After processing all laps for all drivers, it calculates the average length for the <code>medium_stints</code> and <code>hard_stints</code> vectors.</li> <li>• A helper closure <code>avg</code> is used to perform the average calculation, safely handling cases where a stint vector might be empty by returning a predefined default value (14.0 for Medium, 42.0 for Hard).</li> <li>• It returns a tuple containing the calculated average medium and hard stint lengths.</li> </ul>
--	--

## model.rs

The `model.rs` module is designed to predict tyre degradation. It takes the processed lap data and builds a linear regression model that can predict the time delta of a tyre based on its usage and environmental conditions. This model is then used by the `strategy.rs` module to simulate race performance under different strategies.

### 1. Data Structures

- `FittedLinearRegression`
  - A type of `linfa_linear`
  - Holds the learned parameters and the intercept of the linear equation
- `DegradationModel`
  - Perform prediction by using the fitted degradation model for different tyre compounds
  - Fields: `hard_model` and `medium_model`, as different tyre compounds should

have different performance.

## 2. Key Functionality

Name	Functionality
<pre>pub fn new(laps: &amp;[ProcessedLapData]) -&gt; Self</pre>	<ul style="list-style-type: none"><li>• New constructor for degradation model, it takes a slice of processed lap data as input.</li><li>• Inside the function, the input data is filtered for training. For example, it excludes any laps that are marked as pit-in or pit-out (as these lap times are biased by pit-stop, around 20 seconds slower than a usual lap).</li><li>• It then calls the <code>build_model</code> for both medium and hard compound types.</li><li>• It returns a <code>DegradationModel</code> struct containing the result of the model fitting.</li></ul>
<pre>fn build_model(data: &amp;[ProcessedLapData], comp: &amp;str) -&gt; Option&lt;FittedLinearRegression&gt;</pre>	<ul style="list-style-type: none"><li>• Helper function for a single compound tyre</li><li>• Set up minimum number: It includes a check to ensure there are at least 5 data points for the compound. Linear regression requires a minimum number of data points to be fitted, and a small number like 5 is a heuristic to avoid trying to fit a model to insufficient data. If there aren't enough data points, it prints a warning and returns <b>None</b>.</li><li>• Linfa crate: feature and target (x and y) are created through <code>Array2</code> and <code>Array1</code> from <code>ndarray</code>.<ul style="list-style-type: none"><li>◦ X: tyre life, track temperature, the square of tyre life (if the model is non-linear)</li><li>◦ Y: time delta (time loss)</li></ul></li><li>• Model fitting: input x and y to train the model</li></ul>
<pre>pub fn predict_degradation(&amp;self, tyre_lap: u32, temp: f64, comp: &amp;str) -&gt; f64</pre>	<ul style="list-style-type: none"><li>• Function to get a prediction, it takes the lap number, temperature, and compound type as input</li><li>• Prediction: If a model exists for the given compound, it prepares an x</li></ul>

	vector in the same format used during training. Then, use the <code>predict</code> function.
--	--

## strategy.rs

The strategy.rs module is the final piece of the project’s analysis. It takes the insights gained from processing the data (data.rs) and modeling degradation (model.rs) to simulate various potential race strategies and compare their predicted performance over a full race distance.

### 1. Key Functionalities

Name	Functionality
<pre>pub fn simulate_and_print(     model: &amp;DegradationModel,     avg_med: f64,     avg_hard: f64, ) -&gt; Result&lt;(), Box&lt;dyn Error&gt;&gt;</pre>	<ul style="list-style-type: none"> <li>Inputs model, average medium stint length (calculated before), and average hard stint length (calculated before)</li> <li>Constants:             <ul style="list-style-type: none"> <li>Pit loss: a fixed time penalty during each pit stop, set as 21 seconds. This is a simplified value (we can calculate this, but it would make the project too long)</li> <li>Temperature: set to 32 degrees for simplification</li> <li>Total laps: We know the total laps of the grand prix is 56 laps</li> </ul> </li> <li>Strategy Definition: It defines a set of strategies (See section below):             <ul style="list-style-type: none"> <li>1-stop: change tyres for once</li> <li>2-stop: change tyres twice</li> <li>3-stop?: not common in F1, therefore is excluded.</li> </ul> </li> <li>Loop:             <ol style="list-style-type: none"> <li>Initialize total time loss and current pit to zero</li> <li>Iterate through the stints for the current strategy (for instance, we go for 1-stop, it should only have 2 stints)</li> </ol> </li> <li>Degradation simulation: for each lap</li> </ul>

	<p>within the current stint, it increments total delta time by the stint delta time. The stint delta time is calculated through the <code>model.predict_degradation</code>.</p>
<pre> let strategies = vec![     ("1S (M-H)", { let s1=avg_med_u32; let s2=total_laps.saturating_sub(s1); if s1&gt;0&amp;&amp;s2&gt;0&amp;&amp;s1+s2==total_laps { Some(vec! [("MEDIUM", s1), ("HARD", s2) ])}else{None} }},     ("1S (H-M)", { let s1=avg_hard_u32; let s2=total_laps.saturating_sub(s1); if s1&gt;0&amp;&amp;s2&gt;0&amp;&amp;s1+s2==total_laps { Some(vec! [("HARD", s1), ("MEDIUM", s2) ])}else{None} }},     ("1S (H-H)", { let s1=avg_hard_u32; let s2=total_laps.saturating_sub(s1); if s1&gt;0&amp;&amp;s2&gt;0&amp;&amp;s1+s2==total_laps { Some(vec! [("HARD", s1), ("HARD", s2)]) }else{None} }},     ("2S (M-H-H)", { let s1=avg_med_u32; let rem=total_laps.saturating_sub(s1); let s2=(rem as f64/2.0).round() as u32; let s3=rem.saturating_sub(s2); if s1&gt;0&amp;&amp;s2&gt;0&amp;&amp;s3&gt;0&amp;&amp;s1+s2+s3==total_l aps{Some(vec! [("MEDIUM", s1), ("HARD" , s2), ("HARD", s3)])}else{None} }},     ("2S (H-M-M)", { let s1=avg_hard_u32; let rem=total_laps.saturating_sub(s1); let s2=(rem as f64/2.0).round() as u32; let s3=rem.saturating_sub(s2); if </pre>	<ul style="list-style-type: none"> <li>• Saturating sub: saturate potential min values. This post is very useful for me (Piter76. (2020, December). *Saturating, what does it really do, and when is it useful?* The Rust Programming Language Forum. Retrieved April 25, 2025, from <a href="https://users.rust-lang.org/t/saturating-what-does-it-really-do-and-when-is-it-useful/52720">https://users.rust-lang.org/t/saturating-what-does-it-really-do-and-when-is-it-useful/52720</a> )</li> <li>• Due to rounding the average stint lengths to whole numbers, the calculated length of a previous stint (s1 or s2) might theoretically end up being slightly larger than the total laps or the remaining laps (total_laps or rem).</li> <li>• Now if the result is a negative value, it would just round it up to zero</li> </ul>



```
s1>0&& s2>0&& s3>0&& s1+s2+s3==total_laps {Some (vec! [ ("HARD", s1), ("MEDIUM", s2), ("MEDIUM", s3)]) } else {None} } ),
    ];
```

## main.rs

Basically it just calls all the things we have with some error handling... Nothing special here.

## Tests

1. Test average stint calculation:

```
// Driver 1: 5-lap MEDIUM stint, 4-lap HARD stint
driver_data.insert("DRIVER1".to_string(), vec![
    ProcessedLapData {
        driver: "DRIVER1".to_string(),
        lap_number: 1,
        compound: "MEDIUM".to_string(),
        tyre_life: 0,
        lap_time_seconds: 100.0,
        track_temp: 30.0,
        time_delta: 1.0,
        is_pit_out_lap: false,
        is_pit_in_lap: false,
    },
],
```

```
running 1 test
test data::stint_length_tests::test_average_stint_length_calculation ... ok

successes:

successes:
  data::stint_length_tests::test_average_stint_length_calculation

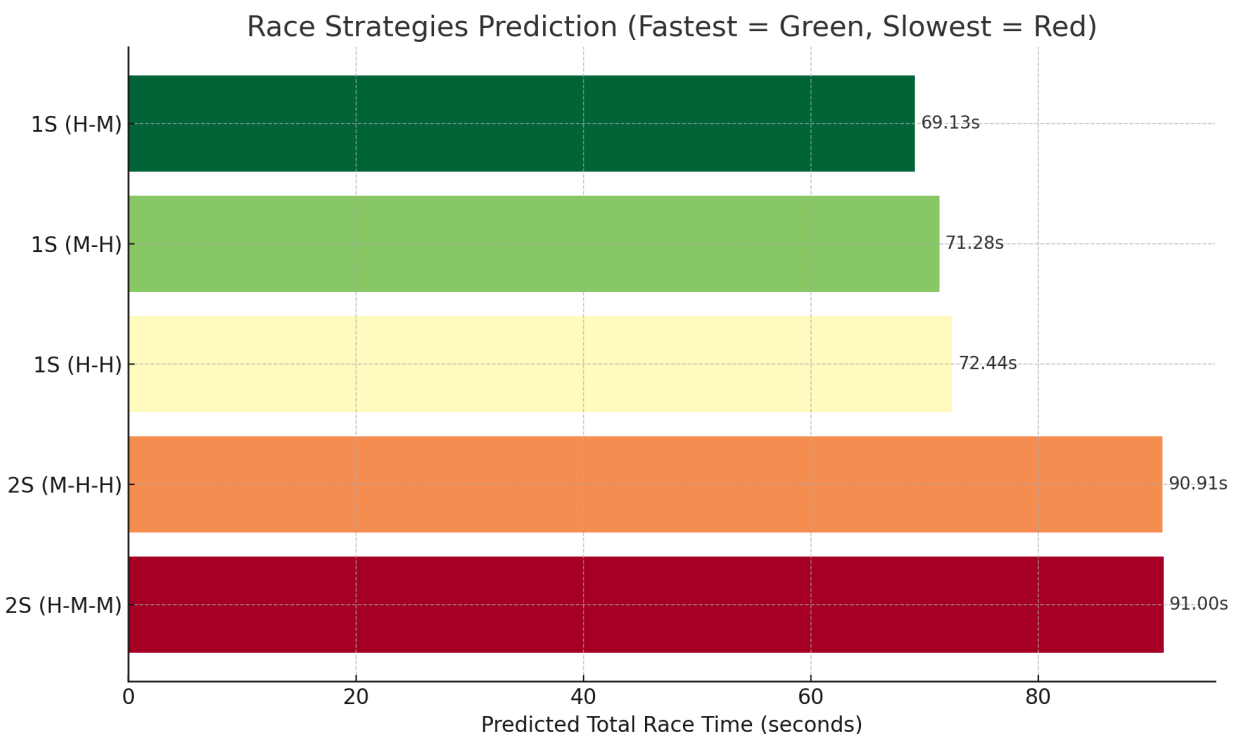
test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

- Manually set up the 2 drivers' information
- To see if our function outputs the average

# Output and interpretations

```
[/opt/app-root/src/finalproject]
● $ cargo run
   Compiling finalproject v0.1.0 (/opt/app-root/src/finalproject)
   Finished `dev` profile [unoptimized + debuginfo] target(s) in 4.22s
   Running `target/debug/finalproject`
Starting Simulation...
Avg Stints - Med: 13.5, Hard: 32.5

--- Strategies (Laps: 56, Pit Loss: 21s) ---
- 1S (M-H)   : 71.28s (1 stops)
- 1S (H-M)   : 69.13s (1 stops)
- 1S (H-H)   : 73.44s (1 stops)
- 2S (M-H-H) : 90.91s (2 stops)
- 2S (H-M-M) : 91.00s (2 stops)
```



## Interpretation of the result

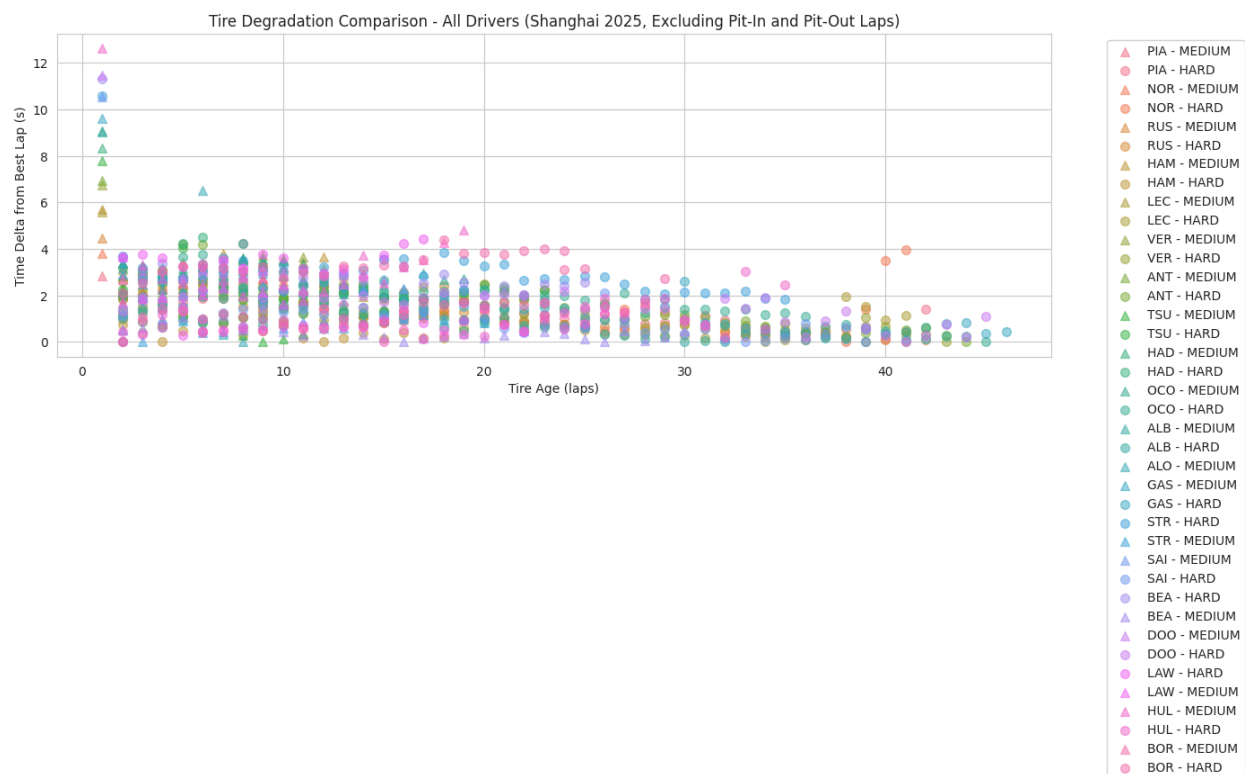
The result showed that the 1-stop strategy is the favorable option based on the drivers' average pace. This is not surprising, since the Shanghai Grand Prix often takes place in the middle of March or April, the weather and temperature conditions are very gentle. Tire degradation would not be that severe. Thus, 2-stop strategies waste an extra pit stop time for drivers. As we can see from the graph, the gap between the slowest 1-stop (H-H) and the fastest 2-stop (M-H-H) is

around 28 seconds, which contains one full pit-in-pit-out time (21 seconds average) and an extra loss in insufficient tyre runs.

Comparing our prediction to the actual pit stop strategies of the Grand Prix, our model preferred drivers to start with hard and then switch to medium, whereas in reality, most of the teams choose to start with medium and then with hard. Here raises the question: Why does our prediction model tend to favor the hard tyre? I think possible answers are:

- Our degradation model collects the data solely from the formal race, so that most of the datapoints are collected around lap 13-20, where the track temperature has risen to the optimal condition for tyres to gain maximum grip. Therefore, their degradation would be less affected by the track condition. However, for medium tyres, their performance is based on the initial track condition, where it is still cold for tyres. For F1 teams, they would collect data from testing sessions like Free Practice, therefore avoiding biases.

We can also validate this insight by taking a look at the driver's time delta by laps:



As we can see, for tire age around 1-3, there is a great time delta, due to the tyres are not heated up enough.

(The image is first image in the section is done by ChatGPT, see sections below)

(The second image is done by Python, see sections below)

# Appendix & Use of AI

Python code for generating plot 2 (using Google Colab, data is pre-processed):

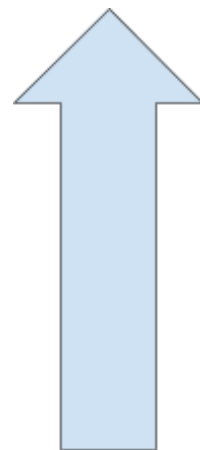
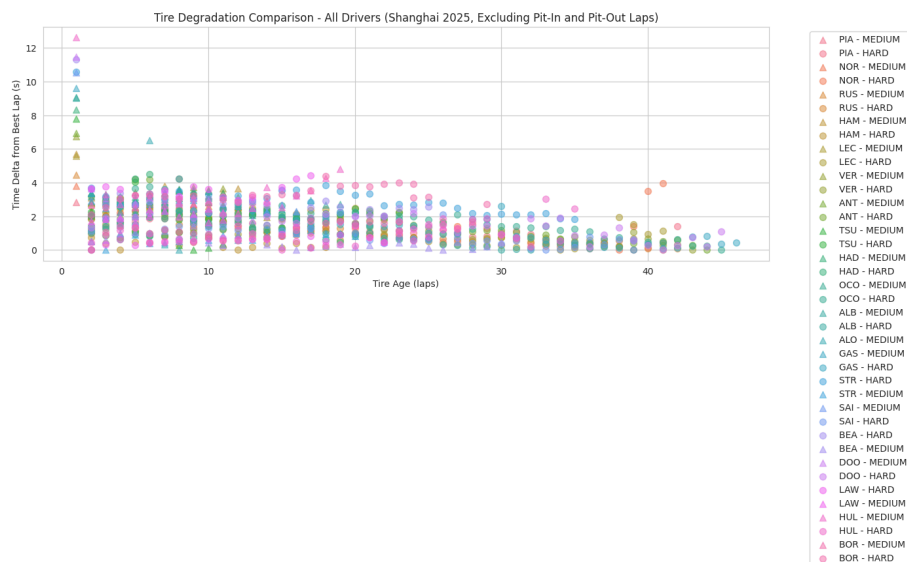
```
# Visualization: Tire Degradation (All Drivers)
plt.figure(figsize=(14, 7))
sns.set_style("whitegrid")

# Define markers for each compound
compound_markers = {'HARD': 'o', 'MEDIUM': '^'} # Circle for HARD, Triangle for MEDIUM

# Get unique drivers and assign colors
drivers = combined_df['Driver'].unique()
colors = sns.color_palette("husl", len(drivers))

for driver, color in zip(drivers, colors):
    driver_data = combined_df[combined_df['Driver'] == driver]
    for compound in driver_data['Compound_lap'].unique():
        compound_data = driver_data[driver_data['Compound_lap'] == compound]
        compound_data = compound_data[(~compound_data['IsPitOutLap']) & (~compound_data['IsPitInLap'])]
        plt.scatter(
            compound_data['TyreLife'],
            compound_data['TimeDelta'],
            label=f"{driver} - {compound}",
            color=color,
            marker=compound_markers.get(compound, 'o'),
            alpha=0.5,
            s=50
        )

plt.title("Tire Degradation Comparison - All Drivers (Shanghai 2025, Excluding Pit-In and Pit-Out Laps)")
plt.xlabel("Tire Age (laps)")
plt.ylabel("Time Delta from Best Lap (s)")
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.tight_layout()
plt.show()
```



AI conversation around Plot 1 (simple bar chart for the results):

ChatGPT

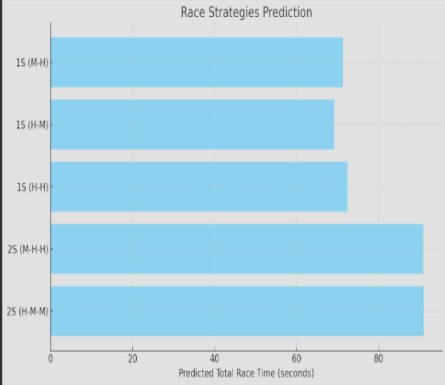
Saved memory full

Share

```
[/opt/app-root/src/finalproject]
$ cargo run
   Compiling finalproject v0.1.0 (/opt/app-root/src/finalproject)
   Finished dev profile [unoptimized + debuginfo] target(s) in 4.22s
   Running target/debug/finalproject
Starting simulation...
Avg Stints - Med: 13.5, Hard: 32.5

--- Strategies (Laps: 56, Pit Loss: 21s) ---
- 15 (H-H) : 71.28s (1 stops)
- 15 (H-M) : 69.13s (1 stops)
- 15 (M-H) : 73.44s (1 stops)
- 25 (H-H-H) : 90.91s (2 stops)
- 25 (H-M-M) : 91.00s (2 stops)
```

plot the strategies in the result of my prediction



Strategy	Predicted Total Race Time (seconds)
15 (M-H)	71.28
15 (H-M)	69.13
15 (H-H)	73.44
25 (M-H-H)	90.91
25 (H-M-M)	91.00

Here's the plot showing your predicted race strategy times!

Would you also like a version that highlights the fastest strategy in a different color? 🚗💡🔍