The Data Driven Formula 1 Strategist: Using Modelling to Optimise Pit Stop Decision Making

Jonathan Chan, Student ID: 201386043
Supervised by: Dr. Cédric Beaume

Department of Mathematics, University of Leeds

September 4, 2020

Abstract

Formula One (F1) teams that implement the correct strategy around pit stop timings and tyre choices have a competitive advantage over the teams that don't. Therefore, it is in their best interests to build data-driven methodologies that can optimize a race strategy in a reproducible way. A driver's lap times are fitted against a simple model that considers fuel usage and tyre wear. This is then used as a basis to find the strategy that completes the race in the fastest time possible using a brute force search. Analysis of the optimal strategies returned by these tools found them to be relatively robust to small changes in the model parameters. Following this, a case study is examined using these tools which illustrate how F1 teams can implement them to improve upon their decision making. Finally, additional improvements to both the modelling and strategy searching aspects of this project are suggested in the discussion.

SUBMITTED IN ACCORDANCE WITH
THE REQUIREMENTS OF THE MODULE

MATH5872M: DISSERTATION IN
DATA SCIENCE AND ANALYTICS

AS PART OF THE DEGREE OF

MASTER of SCIENCE
in
DATA SCIENCE
and ANALYTICS

SCHOOL OF MATHEMATICS
UNIVERSITY OF LEEDS

The candidate confirms that the work submitted is his own and that appropriate credit has been given where reference has been made to the work of others.

Contents

1 Introduction **1**

1.1 Formula One 1

1.2 Strategy 1

2 Statistical Background **5**

2.1 Data Fitting 5

2.1.1 Least-Squares 5

2.1.2 Trust Region Searches 7

2.1.3 Solving Trust Region Subproblems 7

2.1.4 Error in Fitted Parameters 11

2.1.5 Limitations to Least-Squares Data Fitting 13

2.2 Approximating Functions with Taylor Polynomials 13

2.3 Isolation Forests 14

3 Methodology **16**

3.1 Data Discovery and Wrangling 16

3.1.1 Ergast Dataset 16

3.1.2 Racefans Tyre Strategy Dataset 17

3.1.3 Sampling Data for a Driver in a Race. 19

3.2 Outlier Detection 20

3.3 Building a Lap-Time Model 22

3.3.1 Fitting the Lap-Time Model 23

3.3.2 Fitting Multiple Drivers at Once . . . 26
3.4 Modelling a strategy . . . 27
3.4.1 Finding the Time to Complete a Stint . . . 27
3.4.2 Finding the Time to Complete a Pit Stop . . . 28
3.5 Optimising strategy . . . 29

4 Analysis **31**
4.1 Comparison of Different Strategies Found with the Optimiser. . . . 31
4.1.1 Refuelling Scenario . . . 34
4.2 Robustness of Found Strategy . . . 36
4.3 Vettel Canada 2012 Case Study: Strategy Debrief. . . . 39
4.3.1 Vettel's Optimal Strategy . . . 40
4.3.2 Analysing the One-Stop Strategy . . . 41

5 Discussion **45**
5.1 Limitations and Further Considerations. . . . 45
5.2 Conclusion . . . 47

Bibliography **48**

Appendices **51**

A Code used **52**
A.1 Data gathering . . . 52
A.2 Sampler . . . 54
A.3 Modelling . . . 58
A.3.1 Composite model. . . . 63
A.4 Optimizer . . . 66
A.5 Refuelling-optimiser . . . 74

List of Figures

2.1 Illustration of trust region reflective algorithm method. 11

3.1 First five rows of lap times data frame 17

3.2 Racefans tyre strategy data-table example 18

3.3 First five rows of pit stop data-frame 18

3.4 First five rows of sampler script output 19

3.5 Sample scatter plot of lap times using Bottas in the 2015 Spanish GP 19

3.6 Outlier algorithm classifications with different cut-offs 21

3.7 Bottas in 2015 Spanish GP lap times fitted against composite model & its components 24

3.8 Model Residuals for Bottas in 2015 Spanish GP 26

4.1 Plots of the optimal and quick-lap strategies for Bottas in 2015 Spanish GP 33

4.2 Plots of the optimal and quick-lap strategies for Bottas in the 2015 Spanish GP under refuelling scenario 35

4.3 Refuelling vs non-refuelling lap times comparison 36

4.4 Vettel 2012 Canadian GP modelled lap times 40

4.5 Vettel 2012 Canadian GP optimal strategy 41

4.6 Comparison of suggested strategies for Vettel in 2012 Canadian GP . 43

List of Tables

3.1 Table listing the relevant fields from the Ergast dataset used in this project. 17

4.1 Strategies found by strategy optimiser for Bottas in 2015 Spanish GP 32

4.2 Strategies found by strategy optimiser under refuelling scenario using data for Bottas in 2015 Spanish GP 34

4.3 Table of parameters for the penalty coefficients when modelled on re-centered data from Bottas and Massa simultaneously. The subscript T_m represents the medium tyre compound, and the subscript T_h represents the hard tyre compound. 37

4.4 Recalculations of optimal strategies while adjusting parameters 38

4.5 Fitted model parameters on Vettel's lap times on the 2012 Canadian GP 40

4.6 Comparison of different strategies for Vettel in 2012 Canadian GP . . 42

1. Introduction

1.1 Formula One

Formula One (F1) is the highest class of single-seat road course racing in the world. Since it started in 1950, it has become the pinnacle of car racing in terms of speed, viewership and technical innovation. Each year the F1 season is made up of multiple Grand Prix (races, referred to as GPs) held on different road course circuits across the world. F1 is competed with both teams who build the car in the constructors' championship as well as two drivers per team who compete for the separate drivers' championship. In the current season (2020) there are 10 teams competing with 20 drivers across 22 planned GPs.

In each GP, points are awarded to drivers for finishing in the top ten with points distributed from 1st to 10th as follows: 25, 18, 15, 10, 8, 6, 4, 2, 1 with a bonus point going to the driver of the fastest lap of the race. Investments by the teams into the cars, drivers, race performance and race strategy are critical as success in F1 can attract major sponsorship deals as well as a share of the over $1 billion prize pool [11]. As race strategy can have a significant effect on how a driver finishes the race, it has become very important for teams to implement data-driven tools for developing race strategy in an optimal and reproducible way.

1.2 Strategy

In a race, it is the drivers' role to complete a set number of laps in as little time as possible. Throughout each GP, drivers are allowed to come off the track into the 'pit lane' (or 'the pits') where the mechanics of the teams are on standby. In the pits, the mechanics can exchange worn tires for new ones, refuel the car as well as make any repairments necessary. Although there is a cost in time for the driver to come into the pits, it has been long established that the trade-off is worth it in exchange for having a faster car throughout the race.

Race strategy refers to making the decisions around how a driver takes their pit stops throughout the race. This includes how many pit stops, on what lap should the pit stops be taken and what tyres the driver should change to at each stop. Choosing the optimal strategy for a race is a complex optimisation problem, and so the burden of designing the race strategy is taken from the driver and given to the

teams' strategy engineers.

In order for the strategy engineer to be able to find the optimal strategy in a reproducible way, it would be helpful for them to be able to estimate the driver's performance using a potential strategy ahead of time. Therefore a key part of the strategy engineers job is to develop models of the car performance. From this, the engineer can estimate the lap times of the driver over each of the proposed stints (drives in-between stops) to get a lap time model for each strategy. Using these, the engineer can then choose the strategy that suits their driver's needs the most at that time.

There are three main factors that a race engineers might consider when formulating a race strategy.

Timing and Frequency of Pit Stops Every time the driver comes into pit he will be taking time out of the race. There is a balance in the number of pits stops. On the one hand, if they take too many stops, even though they are running a faster car, they will not be able to make up for the added time spent in the pits. On the other hand, not taking enough pit stops entails that the driver is driving on worn-out tires for the majority of the race, meaning that they will have less grip. Not only does this mean that the driver is going to be making slower lap times, but he will also be less able to make defensive manoeuvres to prevent being overtaken by quicker drivers.

Tyre Choices There are several different types of tyre that might be available to a driver when coming into the pits: Hyper Soft, Ultra Soft, Super Soft, Soft, Medium, Hard as well as Intermediate for when the track is too wet for dry tyres and Wet for when there is standing water on the track. The choice of tyres matters when formulating a strategy. As a general rule softer tyres tend to run quicker when fresh but also tend to wear out in fewer laps. The choice of tyres ties in with the number of pit stops a team has to make. If a team opts to run softer tyres then usually they are forced to run more pit stops than a team that doesn't. This is because the soft tyre wearing out quicker will force the driver to pit sooner.

The tyres throughout the season are provided to teams by the official F1 tyre provider, which has been Pirelli for the past 9 seasons. The classes of each tyre are just markers for their relative softness, and Perelli can change the chemical compounds that make up each tyre each year at their discretion. This means that each year teams don't know precisely how the different tyres will perform at each circuit until they run practice laps at the event.

Before each GP Pirelli will choose three dry-weather compounds of differing softness to be made available for the event [14]. Which tyres Perelli chooses is track specific as different tracks apply differing amounts of strain to the tyres. In general, Pirelli chooses the compounds with the aim of encouraging a two-stop strategy at each GP. Before the GP, the driver can choose 13 sets of dry weather tyres to use throughout the whole event, which includes free practice, qualifying and the race. As per the rules of tyre usage throughout the event,

the driver will have 7 tyres available over qualifying and the race. This often means that some of the softer compound tyres available to the drivers at the beginning of each race will have a couple of laps worth of wear on them. In addition, drivers that made it to the last stage of qualifying must start the race on the tyre used during the quickest lap of the previous stage of qualifying, which is usually the softer compound of the three available. Finally, drivers must use at least two different types of tyre throughout the race unless the race is affected by wet weather. The night before each race, the strategy engineer must assess the tyres available to each driver and their condition to find the best strategy that complies with these restrictions.

Fuel In modern-day F1, cars start the race with all the fuel they need for the race. The amount of fuel that the car starts with is important because the weight of the fuel is a major contributor to performance. According to Newton's second law, as the car is heavier with fuel at the beginning of the race, it needs to expend more energy to accelerate around the track. However, the strategists need to be careful, drivers need to finish the race with enough fuel so that the total weight of the car and driver is above a minimum limit. If the strategist is thinking about an aggressive strategy with more pit stops than the other drivers, then the driver will need to push the car's engine to make up the time spent in the pit lane. This will affect the burn rate of the fuel, meaning that the driver should start the race with more fuel. On the other hand, if they're opting for a more conservative strategy, then the driver might want to reduce the amount of fuel they start with.

Refuelling used to be allowed in F1 before it was banned in 2009 after several incidents raised safety concerns. When refuelling was permitted, fuel played a much larger role in strategy formulation. Teams were limited in the rate of refuelling to 12 litres/second, and so refuelling became the dominating factor in how long a pit stop would last. The longer their proposed stint length was in their strategy, the longer the car would have to stay in the pit lane to take on enough fuel to complete the stint. This also meant that fuel management wasn't a factor as drivers knew they could drive their car at its maximum performance knowing that they could refuel if needed.

Other factors There are quite a few other factors that may affect how a driver changes their strategy, but these are usually race dependant and decided in the moment. Weather effects, especially rain are a major influence in the cars performance, and unexpected rainfall will throw off any pre-planned tyre strategy, as a dry tyres performance significantly drops off in the rain due to changes in track surface and temperature. Strategy in changing weather conditions often reduces to making sure the driver has the correct tyres on for the track conditions and ideally before anyone else.

Another reason for changing the pit strategy during a race could be the driver's position. For example, a common pitting strategy is known as the 'undercut'. This is when a driver looking to overtake someone pits early, letting the driver in front keep driving. He then switches to fresh tires and comes out of the pit lane with a faster car than the driver in front. Suppose the driver can close

the gap in time to be less than the time it takes for the front driver to do a pit stop, an achievable goal since they have the faster car. Then when the car in front comes into do a pit stop the car behind will come out ahead albeit with slightly more worn-out tires. If the team is confident in their driver's ability to do defensive driving to maintain that lead, then they may implement this strategy.

The last major factor that sometimes changes a driver's strategy is when a race is interrupted by a safety car. Safety cars control the track speed so that the track can be cleared if there is a racing incident such as debris on the track from a crash. Just before a safety car becomes active drivers often come into the pits so that they can adopt a more aggressive racing strategy. This is for two reasons: If they time it right, they can essentially get a 'free' pit stop as drivers around them aren't driving quick enough to overtake them while they are in the pits. The other reason is that while the safety car is active, the drivers aren't demanding much out of the cars. This means that they can afford to drive more aggressively for the remainder of the race as the tyres won't have as much wear on them if there hadn't been a safety car. While strategists can never predict with 100% confidence that a safety car will be a factor in a race, they can make an educated guess. Some races such as the Monaco GP are much more likely on average to require a safety car than others. In these races, a strategist might hedge their bets and pick a slightly more aggressive strategy than usual so that they can take advantage of a safety car if it does occur.

In this project, We are going to be focussing on the main three factors for deciding a strategy. The other factors such as weather, position and safety car in a race, while important, are much less predictable. The aim of the project is to use modelling to find optimal strategies in a reproducible way. With that in mind, we are not going to consider unpredictable factors when finding our strategies.

The rest of this report will be focused on modelling and utilising the built models to search for an optimal strategy. In chapter 2, the mathematical background behind the different techniques used on the data will be developed. Chapter 3 presents the descriptions of the F1 dataset, and how these mathematical techniques were applied to that dataset to build a model and subsequently, a strategy optimiser. Turning to chapter 4, the optimiser built in chapter 3 will be presented and tested for robustness by analysing the sensitivity of its results for small changes in racing conditions. On top of this, a case study of a poorly executed strategy will be analysed using the tools built in this project. This is to showcase how a real F1 team could use these tools to find out where they went wrong and how they could improve their future strategy decision making. At the end in chapter 5, is a discussion on the tools developed during this project. It is focused around their potential uses, limitations as well as possible suggestions for further work.

2. Statistical Background

In this section is a brief description of the mathematical background behind the techniques used to gain insight into our dataset. This section aims to provide the reader with a baseline understanding of how the different techniques work. This is so that when we come to use these techniques, the discussion about whether or not their results are reliable becomes much easier.

2.1 Data Fitting

The first technique presented is data fitting, which is used to fit a model to the F1 lap times. The ideas behind data fitting, specifically least-squares fitting, are fairly well established. As such, we primarily take our inspirations for the explanations, definitions and theorems from the textbooks on the subject by Strutz, Bard, Bevington and Ryan [27] [5] [3] [25].

Data fitting is based on trying to infer the underlying functional relationship between a measurement and some independent variable(s) (in this case measured lap time as a function of lap number) [27]. This relationship is described by a mathematical model, called a model function, the structure of which is defined beforehand. The role of data fitting is to find the parameters of the model function, called model parameters, that best describe the data observed. This relationship between the measured data points y_i, and the corresponding independent variable x_i is described by equation 2.1, where a is the set of parameters to optimise and ϵ_i are error terms for each point.

$$y_i = f(x_i|a) + \epsilon_i \tag{2.1}$$

2.1.1 Least-Squares

In data fitting, we are looking for the optimal set of model parameter values for the measured data. This is an optimisation problem which in practice is typically solved by minimising some kind of loss function. The most common loss function, and the one that used to fit the parameters in all models for this project, is the summation of the squared differences between the measured values and the values calculated by the model function. Algorithms that calculate the parameters that minimise this loss function, as shown in equation 2.2 are known as 'Least-Squares

methods'. Where w_i are the weightings given to each residual based on how reliable the measurement is (in our case we give each point equal weighting and so can disregard w_i).

$$argmin\, L(a) := \sum_i w_i \cdot [f(x_i|a) - y_i]^2 \tag{2.2}$$

If you make the assumption that your measurements follow equation 2.2 so that y_i is drawn from a Gaussian distribution around $f(x_i|a)$ with a standard deviation of σ_i, then you can think about the problem as one of likelihood [27] [5]. In other words, what is the probability that the set of observations y is observed, given the parameters a? There will be a set of parameters that give the maximum likelihood given the model function and the set of observations. The likelihood for a single observation assuming a Gaussian distribution is given by equation 2.3.

$$P(y_i|a) = \frac{1}{\sigma_i\sqrt{2\pi}} \cdot exp\left\{-\frac{1}{2}\left[\frac{f(x_i|a) - y_i}{\sigma_i}\right]^2\right\} \tag{2.3}$$

Assuming that all observations are independent of each other, the likelihood for the whole set of observations is then just the product of all the individual likelihoods. This is shown in equation 2.4. If σ_i are held to be the same constant, then they have no influence on the fitting outcome, otherwise they act as weights on each data point. We want to find the values of a that maximise the likelihood $P(y|a)$ and the only term that depends on a in 2.4 is $\chi^2(a)$, which is taken to the negative exponential. Therefore we need to minimise $\chi^2(a)$ in order to maximise $P(y|a)$. Therefore minimising the least-squares loss function gives you the maximum likelihood estimates for your parameters.

This is all based on the assumption that the errors are normally distributed around the model function, which is usually never the case exactly in the real world. However, as long as you are careful and can show that your errors are not completely divergent, the least-squares fit should give you a reasonable estimate of your optimal parameters.

$$\begin{aligned} P(y|a) &= \prod_{i=1}^{N}\left(\frac{1}{\sigma_i\sqrt{2\pi}}\right) \cdot exp\left\{-\frac{1}{2}\sum_{i=1}^{N}\left[\frac{f(x_i|a) - y_i}{\sigma_i}\right]^2\right\} \\ &= \prod_{i=1}^{N}\left(\frac{1}{\sigma_i\sqrt{2\pi}}\right) \cdot exp\left\{-\frac{1}{2}\chi^2(a)\right\} \\ where\ \chi^2(a) &\overset{\text{def}}{=} \sum_{i=1}^{N}\left[\frac{f(x_i|a) - y_i}{\sigma_i}\right]^2 \end{aligned} \tag{2.4}$$

The optimiser used during this project is the least-squares optimiser in the `scipy` library in python [30]. The least-squares optimiser in `scipy` minimises the squared residual loss function using a trust region reflective search algorithm [6].

2.1.2 Trust Region Searches

The discussion concerning trust region searches is necessary for completeness as it describes the implementation behind how the used algorithm finds the fitted values. As well as utilising the textbooks previously mentioned, a lot of the information behind the trust region discussion was inspired by the papers cited by the `scipy` documentation, Branch and Gould [6] [17].

As opposed to the more common line search methods, which find a direction of improvement on the error surface and then adjust the parameters in that direction. Trust region searches work by approximating the error surface to a simpler function around an initial guess, usually by a quadratic using a 2^{nd} order Taylor approximation (explained in a later section). Then it solves for the minimum of the approximation within a small radius δ around the initial guess in what's called a 'trust region subproblem'. Then it repeats the process again with the solution of the subproblem becoming the new guess and with δ being increased or decreased based on whether the approximation of the error function is good at the new guess. The idea being that if the approximation is accurate, then we can afford to be more ambitious and increase the trust region. If the approximation isn't very good then perhaps the quadratic approximation isn't very good for far away points, and so we should reduce the trust region. Also, in the case where the error of the new guess is larger than the error of the old guess, then the trust region needs to be reduced.

Trust Region Minimisation Algorithm

Initialise $a_0^*, \delta, n = 0$
Repeat until convergence:

$n \leftarrow n + 1$

Approximate L(a) around a_{n-1}^* to get $\tilde{\text{L}}$(a)

Solve $a_n^* = argmin\ \tilde{L}(a)$ subject to $\|a - a_{n-1}^*\| \leq \delta$

Sample $L(a_n^*)$

If $L(a_n^*) < L(a_{n-1}^*)$ **then** Accept a_n^*

Else: Decrease δ

If $L(a_n^*) \approx \tilde{L}(a_n^*)$ **then** Increase δ

Else: Decrease δ

2.1.3 Solving Trust Region Subproblems

The hardest part in running the trust region algorithm described in the earlier section is solving the trust region subproblems. The general idea behind how to

solve the trust region subproblems is more or less the same across most optimisers, with minor tweaks to increase performance in specific circumstances [27] [3].

The first step is to find the approximation of the function around the latest guess. This is done by constructing the Jacobian matrix (J) which is then used to find the gradient vector (g) and the Hessian matrix (H) of the error function. The gradient vector and the Hessian matrix are the multi-variable equivalents of the first and second order derivatives at the point of the latest guess. The Jacobian matrix contains the partial derivatives of the function describing each data point with respect to each of the parameters, as shown in equation 2.5.

$$J \stackrel{\text{def}}{=} \begin{bmatrix} \frac{\partial f(x_1|a)}{\partial a_1} & \frac{\partial f(x_2|a)}{\partial a_1} & \dots & \frac{\partial f(x_N|a)}{\partial a_1} \\ \frac{\partial f(x_1|a)}{\partial a_2} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \frac{\partial f(x_1|a)}{\partial a_m} & \dots & \dots & \frac{\partial f(x_N|a)}{\partial a_m} \end{bmatrix} \tag{2.5}$$

From the Jacobian matrix and a vector of the residuals, we can calculate the gradient vector, which represents the first derivative of the least-squared error with respect to the parameters. The elements of the gradient vector are simply a sum over each row of the Jacobian matrix weighted by the residual for each point, as shown in 2.6. If we have the residual vector $\mathbf{r}$ then g can be expressed in matrix form as in equation 2.7.

$$g_j \stackrel{\text{def}}{=} \frac{\partial L(a)}{\partial a_j} = \sum_i^N \frac{\partial f(x_i|a)}{\partial a_j} \cdot [f(x_i|a) - y_i] \tag{2.6}$$

$$g = J^T \cdot \mathbf{r} \tag{2.7}$$

We are also interested in the matrix of second order partial derivatives of our least-squares loss function, which is called the Hessian matrix and is described by equation 2.8. The elements of the Hessian matrix can be found by taking the elements of g and taking their partial derivatives with respect to each of the parameters, as shown in equation 2.9. In practice, most optimisers approximate the elements of the Hessian matrix by only taking the second term on the last line of equation 2.9. This is justified because the second derivative term is always zero if the model function is linear, and even if it's not, the multiplying term will randomly fluctuate around zero if a is close to the optimum and tends to cancel out. By taking this approximation, the Hessian matrix can be easily defined once the Jacobian matrix has been found as in equation 2.10.

$$H \stackrel{\text{def}}{=} \begin{bmatrix} \frac{\partial^2 L(a)}{\partial a_1 \partial a_1} & \frac{\partial^2 L(a)}{\partial a_1 \partial a_2} & \dots & \frac{\partial^2 L(a)}{\partial a_1 \partial a_m} \\ \frac{\partial^2 L(a)}{\partial a_2 \partial a_1} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \frac{\partial^2 L(a)}{\partial a_m \partial a_1} & \dots & \dots & \frac{\partial^2 L(a)}{\partial a_m \partial a_m} \end{bmatrix} \tag{2.8}$$

$$
\begin{aligned}
H_{jk} &= \frac{\partial g_j}{\partial a_k} \\
&= \frac{\partial}{\partial a_k} \sum_i^N \frac{\partial f(x_i|a)}{\partial a_j} \cdot [f(x_i|a) - y_i] \\
&= \sum_i^N \frac{\partial^2 f(x_i|a)}{\partial a_j \partial a_k} \cdot [f(x_i|a) - y_i] + \frac{\partial f(x_i|a)}{\partial a_j} \cdot \frac{\partial f(x_i|a)}{\partial a_k}
\end{aligned} \tag{2.9}
$$

$$
H \approx J^T J \tag{2.10}
$$

In single variable calculus, the local minimum is the point where the first derivative is equal to zero, and the second derivative is greater than or equal to zero. In the multivariate case the minimum will be where $g = 0$ and where H is positive semi-definite. If H is positive semi-definite, then we want to find Δa so that the derivative of $L(a + \Delta a)$ is equal to zero. This step, also known as the 'Newton Step' is easily calculated from g and H, as shown in equation 2.11.

$$
\begin{aligned}
F'(a + \Delta a) &= g + H \cdot \Delta a = 0 \\
\Delta a &= -H^{-1} g
\end{aligned} \tag{2.11}
$$

The result from equation 2.11 makes the assumption that the Hessian matrix is positive semi-definite and that the step Δa is unconstrained. These assumptions aren't always the case, therefore what the optimiser does after calculating the gradients and the Hessian matrix depends on their values and the size of the trust region [17]. What follows is a description of the different possible scenarios and what a trust region optimiser would typically do when faced with them:

The Hessian is positive-semi definite and the Newton step is within the trust region: If the trust region is large enough so that the problem can be minimised within the trust region using the Newton step method. The Newton step is taken as a local minimum and the algorithm carries on as described in the earlier section.

The Hessian is positive semi-defenite but the Newton step falls outside of the trust region: If the trust region is too small so that the Newton step no longer falls within the region. Then the solution to the constrained minimisation problem will no longer be within the trust region but will lie somewhere on the boundary. To get a reasonable guess of where on the boundary, most optimisers will simply take the piecewise linear path made from the gradient vector and return where it intersects the boundary.

The Hessian is indefinite: If the Hessian matrix is not positive semi-definite, then the point at where $g = 0$ is no longer guaranteed to be a minimum. Therefore

calculating the Newton step is no longer appropriate. In this case, an optimiser will follow the piecewise linear path until it either leaves the trust region or it encounters a negative curvature. If negative curvature is found, then it is followed as far as the trust region boundary.

Some optimisers build on the process described above to improve upon the number of iterations needed before convergence as well as to be able to handle bounds on the parameters. In particular, the optimiser in `scipy` uses a trust region reflective algorithm that adds two tricks, that may or may not be used depending on the scenario, as described by Branch [6]. The first trick is a change in variables by way of a linear transformation as described in equation 2.12, where u_i and l_i are the upper and lower bounds on a_i. The transformation effectively changes the units of a so that the current guess $\hat{a}^*$ is equidistant to its boundaries. This is useful in the case where the trust region overlaps a variable's pre-defined bound. By centring first, a sufficient reduction of the objective function can be obtained before the variable bound is reached [10]. The effect of the transformation depends on where a is compared to the trust region. If a is within the trust region bounds, and the trust region is circular, then D becomes the identity matrix and has no effect. For points outside the trust region D acts as a reflection from the trust region surface like a beam of light. If the trust region encounters a boundary, then the bounds on a are no longer circular and D re-centres the guess.

$$
\begin{aligned}
v_i(a) &\stackrel{\text{def}}{=} \begin{cases} a_i - u_i & \text{if } \Delta L(a_i) < 0 \ \& \ u_i < \infty \\ a_i - l_i & \text{if } \Delta L(a_i) \geq 0 \ \& \ l_i > -\infty \\ -1 & \text{if } \Delta L(a_i) < 0 \ \& \ u_i = \infty \\ 1 & \text{if } \Delta L(a_i) \geq 0 \ \& \ l_i = -\infty \end{cases} \\
D &\stackrel{\text{def}}{=} diag(|v(a)|^{\frac{1}{2}}) \\
\hat{a} &\stackrel{\text{def}}{=} D^{-1}a
\end{aligned}
\tag{2.12}
$$

The optimiser solves for a minimum in this reflected 'hat' subspace by finding the Newton step described above in this new set of variables, which it can then transform back to the original space afterwards. Because D is non-differentiable on the boundary of the trust region, the minimum in this reflected subspace has to be calculated strictly within the trust region interior. However, the minimum could also be on the boundary, and so the optimiser draws a linear line down the gradient of the loss function to the boundary. It then compares the value of the loss function for the two points to see which is the true minimum.

The second trick of the optimiser is in the case where the trust region overlaps a bound and the linear gradient line hits it. In this case, a third point is considered where the line is reflected off the bound onto a point on the trust region surface as illustrated in figure 2.1. The intuition behind this trick is that if the boundary is close to the true minimum, then the reflected angle will be small and the reflected point on the trust region boundary will also be close. If however, the point on the boundary is not close to the true minimum, then the reflected angle will be large so that the optimiser can explore points further away from the 'incorrect' boundary.

Branch purports that this trick reduces convergence time significantly when simple bounds are imposed on the variables [6].

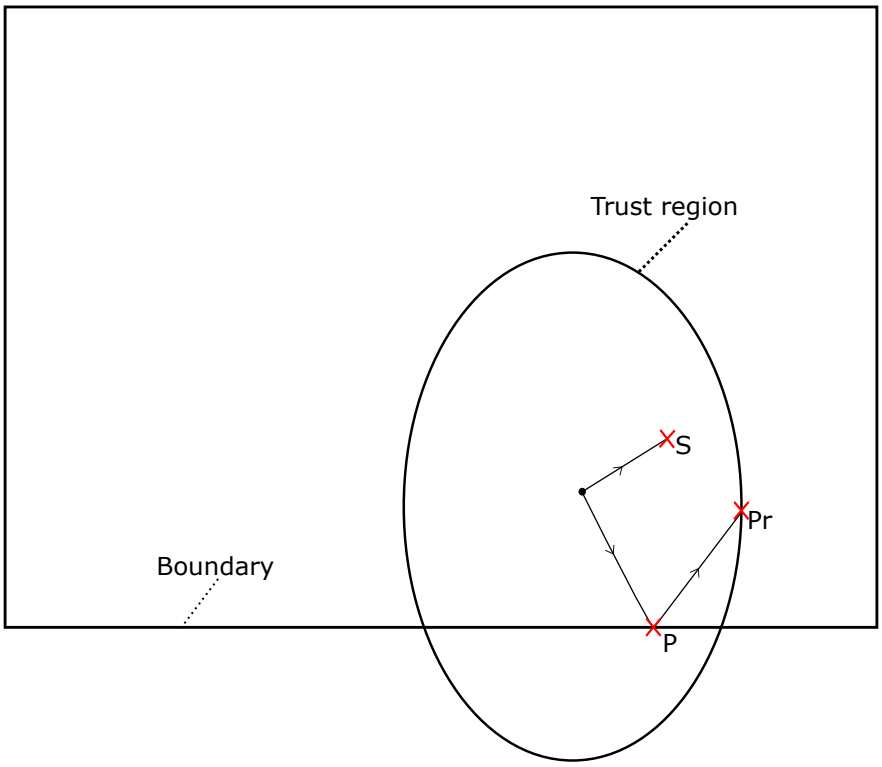


Figure 2.1: Illustration that shows the three points considered as potential minimums of the error function by the trust region reflective algorithm in `scipy`: The cross within the boundary represents the first point (S) and is found by minimising the reflected subspace with a Newton step. The second point (P) is the linear line along the gradient vector from the current guess to the variable boundary, represented by a box. The last point (Pr) is found by the reflection of this line from the boundary to the surface of the trust region. Image adapted from the original paper by Branch [6].

2.1.4 Error in Fitted Parameters

It is often useful to know what our level of confidence is in our found parameters. This can be expressed in their standard errors. A large standard error would signify that we are unsure in the true exact value of the optimal parameter. This project will utilise the standard errors of the fitted parameters when testing the robustness of the strategies found by our built strategy optimiser. The parameters reflect the car's behaviour during the race, and so if the optimal strategy found remains optimal after changing them by their standard errors, we can be more confident that strategy will perform as expected.

Once we have found the maximum likelihood values for the parameters given our data using least-squares optimisation, finding an approximation of the standard errors is an straightforward calculation [3] [25] [12].

If you have optimal values of a (a^*) it is possible to determine the variance-

covariance matrix of a by letting the experimental values change by a small amount and computing the changes in a^*. If we follow the breakdown as in Bard [3], if we are at the minimum of some objective function $L(a)$. To do error analysis, we need to note that L also depends on the measured data w (where w is made up of both x & y) and we replace $L(a)$ with $L(a, w)$. Then the minimum follows equation 2.13.

$$\frac{\partial L(a^*, w)}{\partial a} = 0 \tag{2.13}$$

If you change w by a small amount δw then a^* will also change by a small amount δa^*, as in equation 2.14.

$$\frac{\partial L(a^* + \delta a^*, w + \delta w)}{\partial a} = 0 \tag{2.14}$$

We can expand equation 2.14 as a Taylor series and if we make the assumption that δa is small in response to a small δw, we can be justified to keep only the first order terms letting us rewrite it as in equation 2.15.

$$\left(\frac{\partial^2 L}{\partial a^2}\right) \partial a^* + \left(\frac{\partial^2 L}{\partial a \partial w}\right) \partial w \approx 0 \tag{2.15}$$

Rearranging and substituting $H^* = \left(\frac{\partial^2 L}{\partial a^2}\right)$ gives 2.16.

$$\partial a^* \approx -H^{*-1} \left(\frac{\partial^2 L}{\partial a \partial w}\right) \partial w \tag{2.16}$$

The definition of the covariance matrix we are looking for is given by 2.17.

$$V \stackrel{\text{def}}{=} \mathbb{E}(\partial a^* \partial a^{*T}) \tag{2.17}$$

Plugging in equation 2.16 into 2.17 and solving the expectation, we find that for normally distributed measurements V reduces to the inverse of the Hessian matrix evaluated at the point of maximum likelihood.

$$V \approx H^{*-1} \tag{2.18}$$

From this result, the procedure to gather the standard errors of the parameters is clear. Simply take the inverse of the Hessian matrix at the values of a^* and then take the square roots of the diagonals of that matrix.

It is worth noting however that this is only a first order estimate. The quality of the estimate improves the better the model function describes the measurements and the smaller the measurements' variance about the model function is [3]. In general, this point isn't given much concern as in the cases where the model function isn't a good fit, we wouldn't give much attention to the parameters or their errors anyway. In such cases, we should seek to improve the fitting either by using a better model function or considering more data. If that turns out to be unfeasible, then Dovi proposes a Monte Carlo based alternative for finding the standard errors [12]. However, that is not something explored in this project.

2.1.5 Limitations to Least-Squares Data Fitting

The main limitation to the least-squares fitting algorithms is that they are local optimisers, not global ones. This means that the initialisation of the parameters is very important, not just for fast convergence but also to avoid local minima. This means that if the parameters are initialised too far away from their true values, then it's possible the optimiser either returns the wrong parameters that don't fit the data well or doesn't converge to a solution at all. This means that a priori knowledge about the parameters should be incorporated as much as possible, as often a random initialisation or just initialising at zero is insufficient. In the case where insufficient a priori knowledge is available, then a researcher may consider either multiple random re-initialisations or a grid search of initial parameters until a satisfactory fit is found.

A second limitation comes from calculating the Hessian matrix (something which most optimisers require). If many parameters are being fitted then the computation time to calculate the Hessian matrix scales quadratically, and therefore data fitting can become resource-intensive for a very large amount of parameters. Furthermore, if the values of the Hessian matrix are very large, then taking the inverse to find the variances can lead to values that are below machine precision leading to incorrect results. In this case, it is often sufficient to rescale the values of the data's attributes, so that the elements in the Jacobian matrix are at similar orders of magnitude.

Finally, if one of the fitted model parameters is found to be at a boundary, then the Hessian matrix can't be (easily) calculated, and therefore an estimate for the standard errors in the parameters can't be found. In this scenario, alternative methods of finding the standard errors should be considered, such as the Monte Carlo based method outlined by Dovi [12]. The only drawback being that these methods may take more computational resources to run, and building algorithms to implement them will be more complicated than finding the inverse of the Hessian.

2.2 Approximating Functions with Taylor Polynomials

What follows next is a short discussion on the use of Taylor polynomials for approximation purposes. As well as being used by the least-squares optimiser described throughout section 2.1.2 to approximate the error surface, it is also used to build the model functions that will be fitted to the data in section 3.3. This is because if you can't derive a function analytically, you can approximate it using this technique.

Taylor's Theorem states that any function that is infinitely differentiable about some point can be expressed as an infinite sum of its derivatives around that point [28]. If the sum isn't infinite, then the sum is an approximation and the precision of the approximation increases as more terms are added. The approximation of $f(x)$

around a point a is described by equation 2.19.

$$f(x) \approx \sum_{n=0}^{N} f^{(n)}(a)\frac{(x-a)^n}{n!} = f(a)+f'(a)(x-a)+f''(a)\frac{(x-a)^2}{2}+f'''(a)\frac{(x-a)^3}{6}+... \tag{2.19}$$

If we take a to be zero, then equation 2.19 becomes what is known as a Maclaurin series. In this case, the only terms in equation 2.19 that are non-constant are the polynomial terms of x. Using this theorem, we can approximate any smooth function of x as a sum of polynomials and the problem becomes one of finding the coefficients in front of each polynomial, as shown in equation 2.20.

$$f(x) \approx a + bx + cx^2 + ... \tag{2.20}$$

$$where \begin{cases} a = f(0) \\ b = f'(0) \\ c = f''(0)/2 \\ \vdots \end{cases}$$

If you do use this technique to approximate a function, it is important to note that the approximation gets worse the further away from your starting point (0 in the case a Maclaurin series) that you're looking at. This is because the terms that you neglect to find when doing your approximation get larger the farther away from your starting point you go.

2.3 Isolation Forests

The isolation forest (iForest) algorithm is an outlier detection algorithm introduced in 2008 by Lui [21]. Like most datasets, the F1 lap times contain outlier points that are extreme to the point where we don't want to model on them. Throughout the project, the iForest algorithm was used to identify such points as outliers as described in section 3.2. It is therefore important to understand how it works so that we can discuss why certain points are labelled as outliers and the effect that has on the analysis.

The iForest algorithm works off the assumptions that outliers are relatively few in number and that they have unusual features. Of course outliers don't always follow these assumptions across all problems, but we have found that they hold in the case of F1 lap times. The basic idea behind the algorithm is that it tries to separate all the data points from each other by using random splits. The more isolated and unusual a point is then the easier it should be to separate in this manner. The isolation forest is a type of ensemble algorithm that incorporates many of what the paper calls 'Isolation Trees' (iTree). Each iTree is given a subset of the data and breaks it down as described in the box that follows.

Isolation Tree

Initialise first node in tree that takes in whole dataset as input.
Repeat at each node in tree:
Let **X** be the set of input data to a node.
Let **Q** be a list of attributes in **X**.

If height of node > max height **or** $|\mathbf{X}| = 1$:

return: Height of node, list of **X**

else:

Randomly select an attribute $q \epsilon \mathbf{Q}$

Randomly select p between the max and min of q in **X**

$X_l \leftarrow filter(X, q < p)$

$X_r \leftarrow filter(X, q \geq p)$

Make 2 new nodes one inputting X_l, the other X_r

Repeat process on each new node.

After each tree is done separating the data, the iForest algorithm calculates the an anomaly score for each data point by finding its expected height $\mathbb{E}(h(x))$ throughout the iForest. The equation the iForest uses to calculate the anomaly score is given in equation 2.21, where n is the number of points in the data set. The score is calculated by normalising by $c(n)$, which represents the average path length of an unsuccessful search in a binary search tree. This is what the average path length would be if there were no anomalous instances in the considered dataset. The calculation for $c(n)$ is given by Preiss [24], and is shown in equation 2.22, where H is the harmonic number. Then this product is taken as an exponential. This might seem counter-intuitive but is done in this way to account for the fact that the maximum height of the tree grows linearly with n whereas the average height grows with $log_2(n)$.

Calculating the anomaly score in this way gives rise to the following relationship between the expected height of a point and its score: As $\mathbb{E}(h(x)) \to 0$ then $s(x) \to 1$, as $\mathbb{E}(h(x)) \to c(n)$ then $s(x) \to 0.5$ and as $\mathbb{E}(h(x)) \to n-1$ then $s(x) \to 0$. If points have a score significantly below 0.5, then they should be safe to consider as normal instances. If their score is close to 1, then they are highly likely to be anomalies. If the entire dataset scores close to 0.5, then its likely that it does not contain any anomalous instances.

$$s(x, n) = 2^{-\frac{\mathbb{E}(h(x))}{c(n)}} \tag{2.21}$$

$$c(n) \stackrel{\text{def}}{=} 2 \cdot H(n-1) - (2(n-1)/n) \tag{2.22}$$

3. Methodology

In the previous chapter, the mathematical backgrounds behind the techniques used in this project were outlined. In this chapter the data will be described, including where it was sourced and how it was wrangled into a usable format. Following this, the methodology and the reasoning behind how the techniques described in the previous section were applied to the data will be described. The aim of the chapter is to provide for the reader a detailed overview of the process behind how an optimal strategy is derived from the race data. All of the coding for this project was done using the Python language. The code is available to read in the appendices A.

3.1 Data Discovery and Wrangling

In this section the data acquisition and reformatting process is described. The data for this project comes from two sources. The first dataset was downloaded from a Kaggle project by user 'Chris G' [9]. The project references the Ergast API, [1] a non-profit web-service that provides an historical record of F1 races. The second source of data comes from the Racefans website, an independent motor sport news website specialising in F1 [2].

3.1.1 Ergast Dataset

The Ergast dataset for the years between 1950-2017 was downloaded from the Kaggle project as 13 `.csv` files. The files were read into Python using the 'Latin-1' encoding and loaded into a Pandas [23] data dictionary, with the name of each file being used as a key for each data frame. The relevant fields for our analysis and the files that they came from are outlined in table 3.1.

From these files a single data frame was made by first doing a left join on the Laptimes data frame with the Races data frame on the raceID field. Then doing a join with the Circuits data frame on the circuitID field followed by a join with the Drivers frame on the driverID field. The frame was then cleaned by removing columns so that only the year, circuitRef, driverRef, lap and time fields remained as shown in figure 3.1.

Field name	File(s)	Description
driverID	Drivers, Laptimes, Pitstops	Number used as a key to refer to a specific driver
raceID	Races, Laptimes, Pitstops	Number used as a key to refer to a specific GP
circuitID	Races, Circuits	Number used as a key to refer to a specific circuit
driverRef	Drivers	Unique name to reference a specific driver
circuitRef	Circuits	Unique name to reference a specific circuit
year	Races	Year that GP took place
forename	Drivers	Forename of driver
surname	Drivers	Surname of Driver
lap	Laptimes	Lap number
milliseconds	Laptimes	Time taken for driver to complete lap in milliseconds
time	Laptimes	Time taken for driver to complete lap in seconds. Made by dividing entries in milliseconds field by 1000
stop	Pitstops	Number counting which stint the driver in each race is on
(stop)lap	Pitstops	Lap on which driver came in for a pit stop
duration	Pitstops	Time taken to complete the pit stop from pit lane entry to exit in seconds.

Table 3.1: Table listing the relevant fields from the Ergast dataset used in this project.

Index	year	circuitRef	driverRef	lap	time
0	2011	albert_park	vettel	1	98.109
1	2011	albert_park	vettel	2	93.006
2	2011	albert_park	vettel	3	92.713
3	2011	albert_park	vettel	4	92.803
4	2011	albert_park	vettel	5	92.342

Figure 3.1: First five rows of the lap time data frame made from joining the Laptimes, Races, Circuits and Drivers files.

3.1.2 Racefans Tyre Strategy Dataset

Although the Ergast dataset had pit stop lap, number and duration for each race it was missing which tyres the drivers started on and what they switched to at each stop. The Racefans website has this data for all races starting from 2011 as they do a summary of statistics and strategies for each race. To acquire this data the webpage for each individual race was found. At each web page the tyre strategy table, such as in figure 3.2 was located and manually copied into a `.csv` file along with the race's corresponding raceID. After the Racefans `.csv` had been made it was loaded using the 'utf-8' encoding into our Pandas data dictionary. Because the encodings on the names were different for the two data sources, some of the names in the forename and surname fields in the Drivers file had to be changed so that they matched. Specifically Raikkonen, Vergne, Hulkenburg, Gutierrez, Sainz, Buemi and

Abu Dhabi Grand Prix pit stop times

How long each driver's pit stops took:

	Stint 1	Stint 2	Stint 3	Stint 4
Lewis Hamilton	Super soft (10)	Soft (21)	Soft (24)	
Felipe Massa	Super soft (13)	Soft (30)	Super soft (12)	
Valtteri Bottas	Super soft (10)	Soft (25)	Soft (20)	
Daniel Ricciardo	Soft (27)	Soft (20)	Super soft (8)	
Jenson Button	Super soft (6)	Soft (22)	Soft (27)	

Figure 3.2: Screenshot of the first five rows of the Racefans' tyre strategy table for the 2014 Abu Dhabi GP. The rest of the table can be found at https://www.racefans.net/2014/11/23/massa-doubts-victory-possible/.

Index	stop	lap	duration	illiseconc	year	circuitRef	driverRef	stint 1	stint 2	stint 3	stint 4	stint 5	stint 6
0	1	1	26.898	26898	2011	albert_park	alguersuari	Soft	Soft	Hard	Hard	nan	nan
1	1	1	25.021	25021	2011	albert_park	michael_schumacher	Soft	Soft	Hard	nan	nan	nan
2	1	11	23.426	23426	2011	albert_park	webber	Soft	Hard	Soft	Soft	nan	nan
3	1	12	23.251	23251	2011	albert_park	alonso	Soft	Soft	Soft	Hard	nan	nan
4	1	13	23.842	23842	2011	albert_park	massa	Soft	Soft	Hard	Soft	nan	nan

Figure 3.3: First five rows of the pit stop data-frame made by joining the Pitstop, Races, Circuits, Drivers and Racefans data together.

Pirez had to be altered in some way. Then the name field in the Racefans data frame was split at the first space to create forename and surname fields. Then a left join on the forename and surname fields was done with the Drivers data frame. Some data entries in the racefans data frame had number enclosed in parentheses as in figure 3.2. This number and parenthesis were isolated using a regular expression and removed from each entry if they existed.

To get a data frame that included all of the data about the pit stops that we needed, some steps needed to be carries out. First, the Pitstop data frame was left joined with the Races data frame on the RaceID field. Then this was joined with the Circuit data frame on the circuitID field and then followed by a join with the Driver data frame on the driverID field. Finally the Racefans data frame was joined using the raceID and driverID fields. The pit stop data frame was then cleaned so that the stop, (stop) lap, duration, duration in milliseconds, year, circuitRef, driverRef and the tyre choices were included as in figure 3.3.

On reflection, even though it did the job, this isn't the cleanest looking data frame as the tyre strategy data is getting repeated as a new row is created for each stop. To get it in normal form, a second data frame should be used that just contains the year, circuitRef, driverRef and tyre data.

3.1.3 Sampling Data for a Driver in a Race.

The two data frames for lap time and pit stops described in the last two sections were written into their own `.csv` files. A script was written that reads in these two files, and samples the lap times for a given driver and a given race, which we will refer to as the sampler script. It does this by filtering for the correct driverRef, circuitRef and year field within the lap time data frame. The sampler also retrieves the stint and tyre information, as well as the lap time for each lap.

A new variable called 'stint lap' was added that was simply a counter that started at 1 and increased every lap, but reset back to 1 whenever the lap equalled an entry in the pit lap list plus one. A second variable was also added called 'stint' which represents what stint number each lap was raced in. The 'stint' variable was also a counter that started at 1 and increased every time the lap variable went higher than a value in the pit lap list. Finally, the tyre variable for each lap was added by searching for the entry in the tyre data columns that matched the 'stint' variable, i.e when stint=1 use the entry in the 'stint 1' column. A sample for the output of the sampler when called with Bottas, Catalunya, 2015 is shown in figure 3.4, and a scatter plot of the lap times is shown in 3.5.

Index	lap	time	stint lap	stint	tyre
95794	1	99.104	1	1	Medium
95795	2	93.075	2	1	Medium
95796	3	92.708	3	1	Medium
95797	4	92.857	4	1	Medium
95798	5	92.913	5	1	Medium

Figure 3.4: First five rows of the output of the sampler script for Bottas, Catalunya, 2015

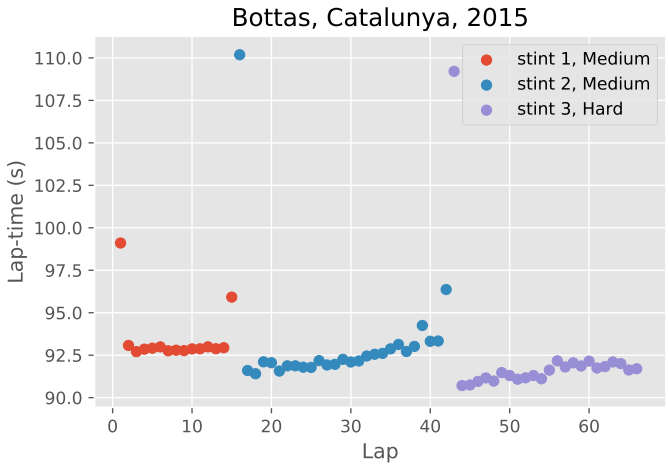


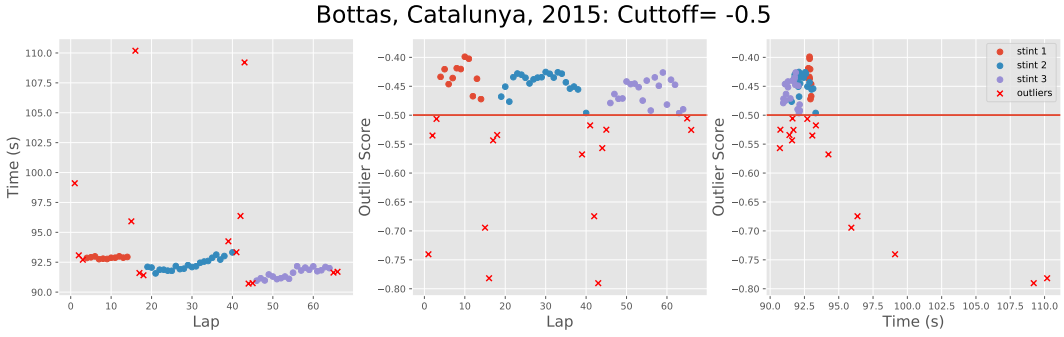
Figure 3.5: Scatter plot of the lap times against the laps. Plot made using data from the sampling script called using Bottas in the 2015 Spanish GP (the track name is Catalunya). Laps have been grouped by stint and shown in different colours, with the stint information as well as the tyre used shown in the legend.

3.2 Outlier Detection

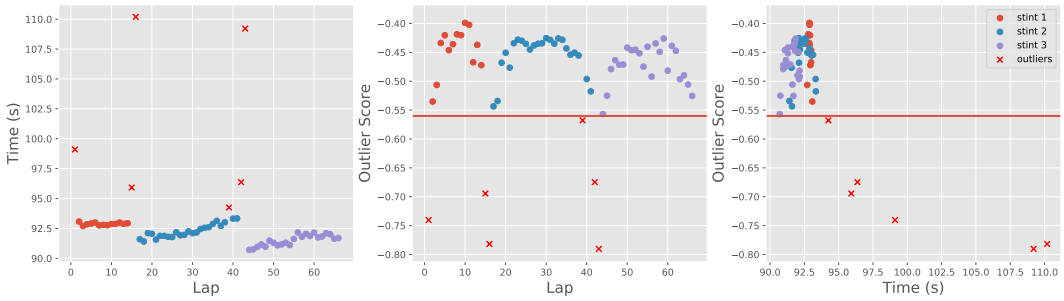
Dealing with outliers is an issue that needs to be solved in most projects involving data. We would like to keep the data that describes the underlying relationship between the independent variable (lap) and the dependant variable (lap time). Scattered throughout the dataset are laps affected by other effects that we are not interested in. For example, coming into the pit lane, making a mistake or getting held up by a slower driver. The simple method of calculating the standard deviation outside of the mean, a technique often used for normally distributed data, is not sufficient as the relationship between lap and lap time isn't necessarily a constant. Therefore if we eliminated points for being to far away from a constant value we would be introducing a large bias towards laps closer to the mean stint time.

The points were given outlier scores using the isolation forest algorithm outlined in section 2.3. The isolation forest is an unsupervised algorithm that has the advantage of not requiring the need to build a profile of the data in order to reject points outside of that profile. It just identifies points that are easily isolated from the rest. The data was grouped by the stint variable, as the model function can change between each stint, and then was fed into the isolation forest algorithm built into `scipy`. The number of iTrees contained in the iForest was 100 as the original paper found that most datasets converge within that number and that any more give diminishing returns [21].

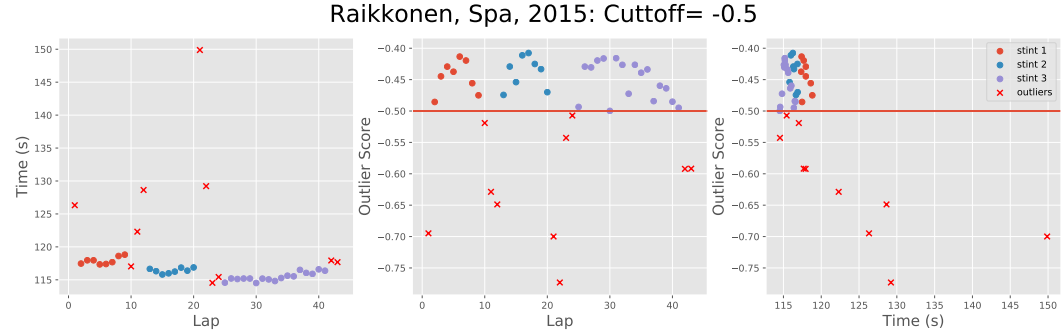
The implementation of the algorithm by `scipy` follows all the same processes and calculations as in the original paper but returns the negative value of the scores (0 indicates no chance of being an outlier, -1 indicates very high chance of being an outlier). The default decision function in `scipy` classifies a point as an outlier if its anomaly score is lower than -0.5. A score lower than -0.5 represents that the point was isolated quicker across the 100 iTrees, than the average unsuccessful search in a binary search tree. It was found that this value was slightly over tuned for the F1 data and so the decision was made to decrease the threshold to -0.56. Examples of the effect of this change is visualised in figure 3.6, which shows how the points are classified with different thresholds as well as the anomaly scores plotted against the different features of the data.



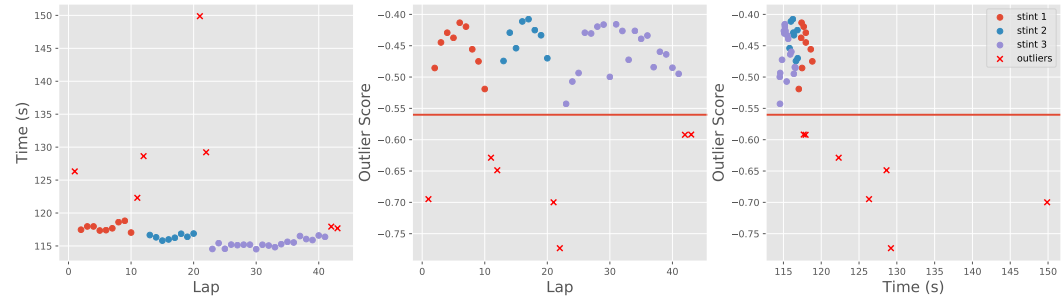
(a)



(b)



(c)



(d)

Figure 3.6: Isolation forest algorithm classifications with different thresholds for outliers. The scatter plots from left to right show the lap time vs lap variables, the outlier score vs lap variable and the outlier score vs lap time variable. In (a) the threshold is -0.5, in (b) the threshold is -0.56. Both (a) and (b) made using race data from **Bottas in the 2015 Spanish GP.** The thresholds are plotted in the middle and right graphs as a red line. The points are coloured according to their stints and outliers are shown as crosses. Figures (c) and (d) show the same informations as (a) and (b) but using data from **Raikkonen in the 2015 Belgium GP.**

3.3 Building a Lap-Time Model

The model that was used throughout this project was a relatively simple one. It was assumed that the lap time is only affected by the fuel load and the tyre wear. From this assumption, each effect is treated separately and is assigned its own penalty. The penalties are added up linearly to make up the model as in equation 3.1. Where t_{base} is a baseline lap-time for that driver in race and P_{fuel} and P_{tyre} are the fuel and tyre penalties.

$$t = t_{base} + P_{fuel} + P_{tyre} \tag{3.1}$$

This is a relatively simple model and there are quite a few other effects that you could consider for a lap time model such as track changes, weather effects, other driver interactions, flags and safety cars or even driving style. The model works best in GPs where these effects don't influence the race as much as others. A good example of this is the Spanish GP on the Catalunya circuit. Catalunya tends to avoid getting safety cars as well as usually having dry weather. Catalunya is also a fast circuit that has mostly high speed corners. Drivers put a lot of work through the tyres through high speed corners, changing direction quickly requires a lot of force that is applied on the car through the tyres. Circuits with many high speed corners therefore tend to have more predicable tyre wear.

As a consequence of adding these terms linearly we are also assuming that there isn't any non-linear coupling between these two terms. This is likely not the case as a heavier car is going to be pushing more force upon the tyres gravitationally. On the other hand, a lighter car can go through corners faster which puts more lateral strain on the tires so there are competing effects. So even though there are physical reasons to assume that there is some relationship between fuel load and tyre wear, these relationships were not explored in this project. Although these effects weren't considered, equation 3.1 should still be an adequate approximation of the lap times for the project's purposes.

As outlined in section 2.1.1 a model function for each of the penalties needs to be given to the least-squares optimiser in order to do data fitting. Deriving a model function for the two penalties is not trivial. For the fuel penalty, the car gets lighter as it burns fuel meaning that it can accelerate and decelerate faster. Having a lighter car also means that the car can grip through the corners at a faster pace due to having more downforce and producing less lateral force.

For the tyre penalties there are two main effects that contribute to a slower car as it's tyres lose rubber. First as the tyre loses rubber its radius gets smaller and therefore its circumference gets smaller which means that for every rotation the car will travel less. The second reason is to do with the change of the physical properties of the tyre surface as a function of its depth. As the tyre loses rubber its surface composition may change due to different depths of the tyre being exposed to different levels of treatment during its manufacturing. At the same time, as the tyre loses rubber it becomes harder for it to maintain its temperature, which it needs to keep high to maintain grip. The bottom line is that as the tyre loses rubber its surface becomes less sticky resulting in less grip and slower lap times.

As a consequence of the difficulties behind the deriving of a model for the penalties, it is easier to approximate them using Taylor polynomials as described in section 2.2. The fuel penalty was modelled as a linear effect, with the penalty decreasing as the laps increase so that the penalty equals zero on the final lap. The tyre penalty was modelled as a quadratic equation with the penalty increasing for every lap that the tyre is run for. For non-refuelling races the maximum lap is the last lap of the race and so the fuel penalty is measured as a function of race lap (l_r). On the other hand, the tyre penalty is measured as a function of stint lap (l_s) as fresh tyres are applied at every pit stop. Therefore two independent variables are used in the final lap-time model, as shown in equation 3.2. Where l_{max} is the number of laps in the race, F is the rate of the fuel penalty decline, T_1 and T_2 are the linear and quadratic components of the tyre penalty and T_b is the relative penalty for running a certain compound of tyre. As different tyres have different initial grip and different rates of wear, the T_1, T_2 and T_b coefficients should be found separately for each compound of tyre used.

$$t(l_r, l_s, F, T_1, T_2, T_{base}, l_{max}) = t_{base} + F(l_{max} - l_r) + T_1 l_s + T_2 l_s^2 + T_b \tag{3.2}$$

3.3.1 Fitting the Lap-Time Model

The fitting of the model was done using the `lmfit` package [22], which is a fitting package built around `scipy`, but with some added functionality that allows for ease of use with using multiple independent variables, and implementing restrictions on the fitting parameters. To deal with different tyres having different behaviour, each of the tyres were given their own set parameters (e.g. $T_{1soft}, T_{2soft}, T_{bsoft}$). Then a third independent variable, 'tyre choice', was added. This variable acted as a flag so that if it equalled a certain tyre then the output of the model would only be affected by the tyre parameters for that tyre, as shown in equation 3.3

$$t(l_r, l_s, tyre) = \begin{cases} t_{base} + F(l_{max} - l_r) + T_{1soft} l_s + T_{2soft} l_s^2 + T_{bsoft} & \text{if } tyre = \text{“}Soft\text{”} \\ t_{base} + F(l_{max} - l_r) + T_{1medium} l_s + T_{2medium} l_s^2 + T_{bmedium} & \text{if } tyre = \text{“}Medium\text{”} \\ \vdots \end{cases} \tag{3.3}$$

As outlined in section 2.1.1 it's extremely important to have a reasonable parameter initialisation for the fitting to work. The variable F was initialised at 0.02 and bounded to be above 0 as it doesn't make sense for the car to get slower as it loses fuel. The tyre variables were all initialised at 0 and the T_2 variables were bounded to be above zero so that the 2^{nd} derivative of the tyre penalty was always positive. This bound was introduced because we would like to impose that the tyre penalty is going to be increasing eventually. If this bound wasn't imposed then often the T_2 variable would be negative and the model would end up predicting a negative tyre penalty for the later laps which resulted in the optimiser choosing a zero stop strategy on that tyre. The t_{base} variable was initialised to be the fastest lap in the dataset and was bound to be ± 5 seconds of that value.

Before fitting the tyre dataset is checked for the types of tyre present. If a type of tyre is not in the dataset then the parameters corresponding to that compound of

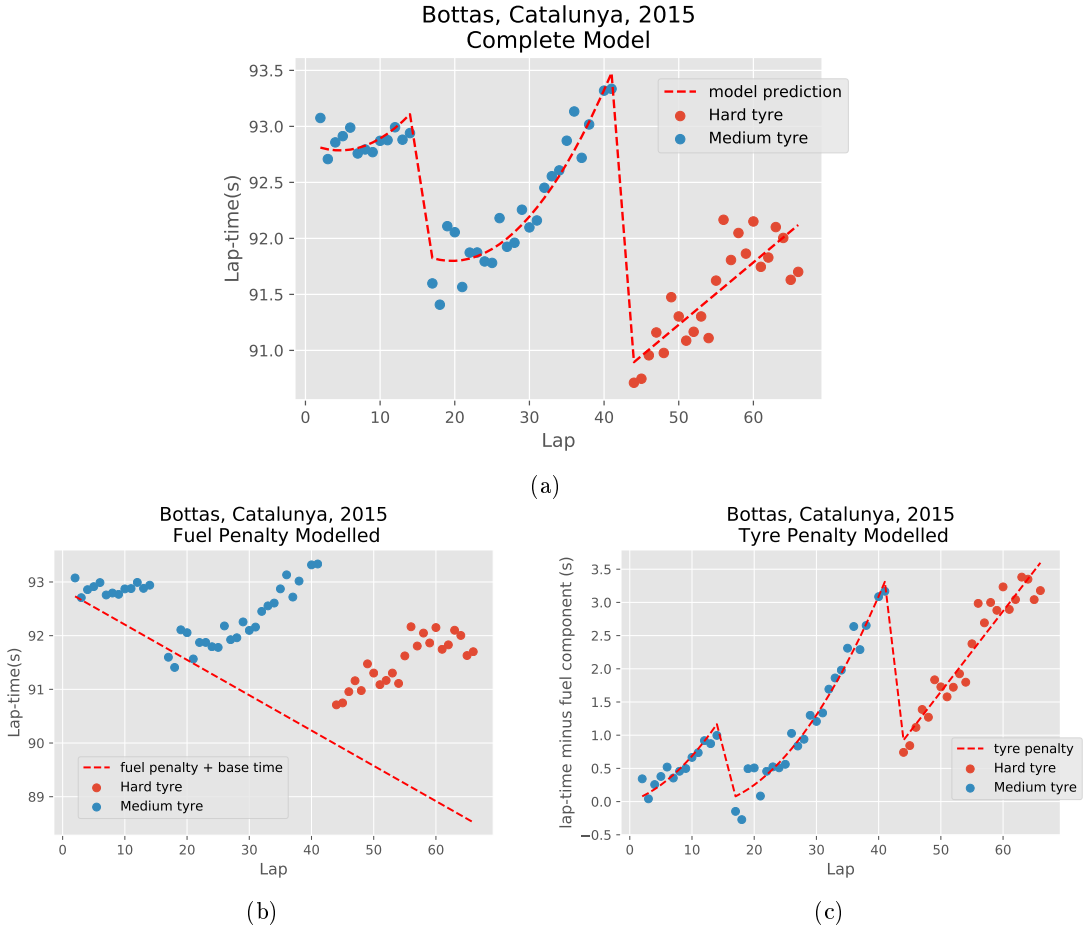


Figure 3.7: Data for Bottas in the 2015 Catalunya GP fitted against the different components of the model. (a) Shows the complete model as described in equation 3.3. (b) Shows the linear fuel penalty component added to the t_{base} variable. (c) The scatter plot is of the lap time variable minus the fuel penalty component against lap. Overlaid is the quadratic tyre penalty component, which separately fits the different types of tyre compound used. The coefficient for the squared component of the quadratic model was bound to be above zero, which is why the hard tyre may appear to be modelled as a linear fit.

tyre are set to be non-varying as they will not affect the output of the model. The l_{max} variable is set to be the final lap of the race and is also set to be non-varying. Finally for the most used tyre in the input data the corresponding T_{base} is set to be non-varying as well. This is because the purpose of the T_{base} is to find the relative differences in base time between the tyres. If all the T_b variables are allowed to be free then the bias in the data will be fitted on two variables t_{base} and T_b which would not make any sense. The base of the most used tyre was set to be the non-varying one as it was found this let the optimiser make a reasonable guess of the base time most often.

Fitting was done using the least-squares trust region reflective algorithm discussed in section 2.1.2. As an example the fitted model for Bottas' race in the 2015 Catalunya GP is shown in figure 3.7. It is also possible to take the fitted parameters and show the model in its components. Looking at the fuel penalty and base time

component in figure 3.7b we can see that the fuel penalty draws a line between the beginning laps of the stints that share the same tyres. This is working as intended because according to our assumptions there should be no difference between those laps except for how much fuel the car has burnt, as they are both using the same tyre when it's fresh. For the tyre penalty we can see that the hard tyre has a relative bias to the medium of about +1 second but also seems to increase linearly. The medium tyre has a smaller bias (set to 0) but increases proportional to the stint lap squared. This is more or less what we would expect going in with the assumption that softer tyres are quicker but wear out quicker at the same time.

The residuals of this model can be calculated by taking the difference between the true lap times and the modelled lap times as shown in figure 3.8. The residuals in the scatter plot are centered around zero and seem to be randomly distributed with a standard deviation of 0.20 seconds. Looking at the histogram the residuals seem to have a bell-shape appearance, perhaps with a slight skew to the right. Which is what we expected since it is harder for a driver to perform a faster lap than average than a slower one.

Even though the histogram isn't perfectly symmetrical we can do normality tests on the residuals to check. The plot of the residuals' quantiles against their theoretical quantiles if they followed a perfectly normal distribution is shown in figure 3.8c. Looking at this plot the residuals seem to follow a straight line which is what we would explect from normal data. As a second check we can perform the Shaprio-Wilks test [26], which is a hypothesis test that tests the normality of the data based on the correlation of the quantile-quantile plot. The Shapiro-Wilks p value for this data is 0.526 which indicates that there isn't sufficient evidence to reject the null hypothesis, which is that the residuals are normally distributed. Considering everything, we would conclude that the errors are normally distributed enough to say that the least-squares fitting is an appropriate way to find the model parameters.

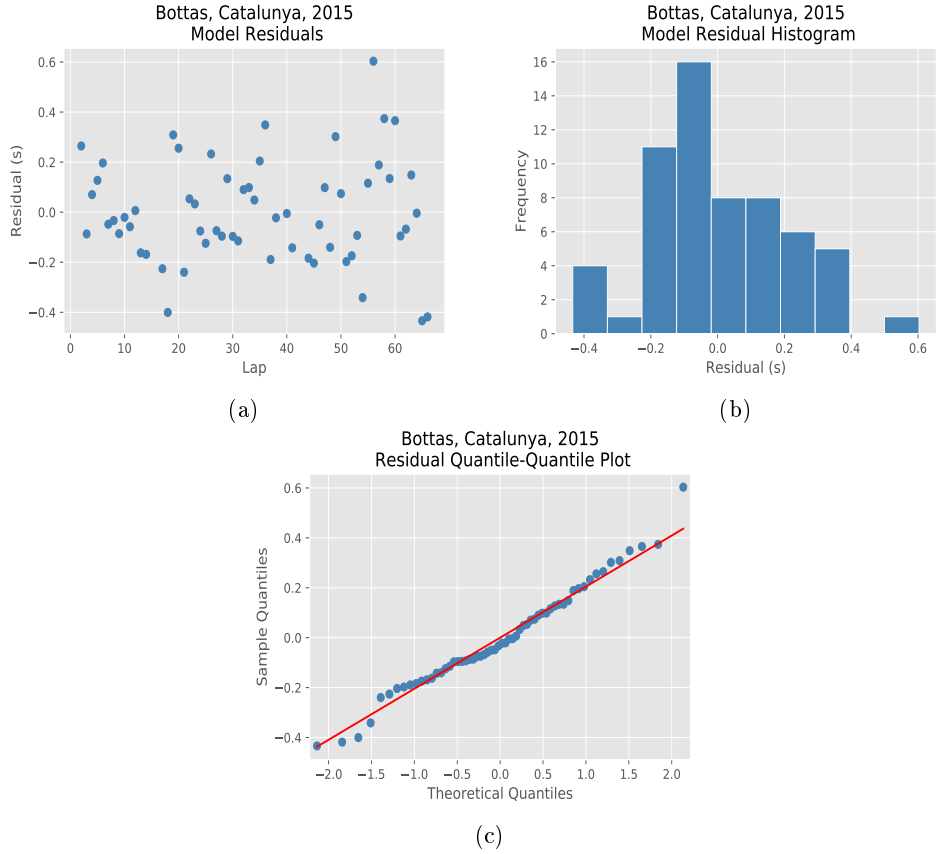


Figure 3.8: Residuals of the fitted model using the data for Bottas in the 2015 Spanish GP. (a) Scatter plot of residuals against the race lap. (b) Histogram of the residuals.(c) Quantile-quantile plot of the residuals, the red line represents perfect normality.

3.3.2 Fitting Multiple Drivers at Once

Some parts of the analysis required data from multiple drivers to be fitted at once. This is a problem as different drivers have different t_{base} values and so putting all the data into a big data frame and fitting it all at once will produce nonsensical results. To get around this the data from each driver needs to be re-centered. This is done by finding the mean lap time for each driver and subtracting it from their respective lap times. The re-centered points are then added to a data frame containing all the adjusted data points. This data frame can then be fitted against the model function to get the penalty coefficients for the group of drivers.

For this method the assumption was made that the biggest differences between the drivers is the t_{base} variable. In other words we are assuming that the drivers get quicker or slower at the same rate due to fuel and tyre penalties but have different starting speeds. In general we don't think this is a great assumption as different cars and different driving styles should affect things like tyre wear rates and fuel consumption. Doing fitting on multiple drivers at once however does give the advantage of having more data to fit on as often when using only one driver there is only one stint using a certain tyre type to consider. F1 teams don't have

this problem as they have access to practice lap data as well as presumably their own in-house testing. Having more data to fit on should make our model parameters more robust as long as the assumption made about drivers having similar coefficients for their penalties doesn't fail too badly.

3.4 Modelling a strategy

Once the model parameters have been found the next step is to take the model and calculate our estimate for the time for a driver to complete a race following a given strategy. This time calculation is split into two functions. First is the time for the driver to complete each lap in each stint and the second is the time for the driver to complete the number of pit stops required of the strategy. Both of these functions are affected by whether or not we are considering the scenario where the driver is permitted to refuel.

3.4.1 Finding the Time to Complete a Stint

The time to complete a lap of the race given the lap, stint lap, tyre and model parameters is given by equation 3.2. The time taken to complete a whole race is just the summation of all the times for each lap as in equation 3.4. A function was built that creates a data frame of the lap, stint lap and tyre variables for a given strategy and then runs each row through the model and adds up all the times.

$$Laptimes = \sum_{i}^{maxlap} t(l_{ri}, l_{si}, tyre_i) = \sum_{i}^{maxlap} t_{base} + F(l_{max} - l_{ri}) + T_1 l_{si} + T_2 l_{si}^2 + T_b \quad (3.4)$$

This method requires calculating the lap times for each lap and so its relatively resource heavy for when we need to compare thousands of strategies. A second way is to calculate the time it takes to complete each stint and then add up the stint times.

The lap and stint lap variables are integers increasing by one for each lap throughout the stint. We can take advantage of this fact by replacing the summations in equation 3.4 with the formulas for the summations of a series of integers raised to a power, as shown in equation 3.5 and 3.7 . For our purposes we are only considering the second order polynomial for the tyre penalty so we are interested in the sum of the squared integers. The summation of integers raised to a higher power is given by Faulhaber's formula [19] and so the thought process could easily be expanded to include higher level approximations of the penalties. For the case where we want to find the sum of consecutive integers starting at m and not 1, as we need to do for

the fuel penalty, that formula is given by equation 3.6.

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2} \tag{3.5}$$

$$\sum_{k=m}^{n} k = \frac{(n-m)(m+n+1)}{2} \tag{3.6}$$

$$\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6} \tag{3.7}$$

Plugging these equations into 3.4 gives us equation 3.8 which is the time taken for the driver to complete a stint. Where *startlap* and *endlap* are the lap numbers of the start and end of the stint, and *maxlap* is the maximum lap of the race.

$$\begin{aligned} \Delta &\stackrel{\text{def}}{=} startlap - endlap \\ Stint\ time =& t_{base} \cdot \Delta + (F \cdot maxlap \cdot \Delta) - F\frac{\Delta(startlap + endlap + 1)}{2} + \\ & T_1\frac{\Delta(\Delta+1)}{2} + T_2\frac{\Delta(\Delta+1)(2\Delta+2)}{6} + T_b \cdot \Delta \end{aligned} \tag{3.8}$$

If refuelling is allowed, and we make the assumption that teams will fill their cars up with just enough fuel to finish the planned stint, then the max-lap variable becomes the end of the stint. At the same time the race lap variable l_r in the fuel penalty equation becomes the stint lap l_s. Then to calculate the stint time we just replace the term in equation 3.8 that calculates the sum of the race laps with the sum of the stint laps. Under this scenario, equation 3.2 becomes equation 3.9, and equation 3.8 becomes equation 3.10.

$$t(l_s, F, T_1, T_2, T_{base}, l_{s\,max}) = t_{base} + F(l_{s\,max} - l_s) + T_1 l_s + T_2 l_s^2 + T_b \tag{3.9}$$

$$\begin{aligned} \Delta &\stackrel{\text{def}}{=} startlap - endlap \\ Stint\ time =& t_{base} \cdot \Delta + (F \cdot \Delta^2) - F\frac{\Delta(\Delta+1)}{2} + T_1\frac{\Delta(\Delta+1)}{2} + \\ & T_2\frac{\Delta(\Delta+1)(2\Delta+2)}{6} + T_b \cdot \Delta \end{aligned} \tag{3.10}$$

Both methods of calculating the drive time of the strategies were implemented. The method of calculating the stint times and adding them up was found to run 30-40 times faster. The method of calculating each lap time individually and adding them up had the advantage of containing information about each lap. This information may prove useful to teams when their strategies need to be altered to compete against other drivers or when choosing a strategy to get the fastest lap of the race.

3.4.2 Finding the Time to Complete a Pit Stop

For the non-refuelling case, calculating the time the driver spends in the pit lane is easy because the Ergast dataset provides data for the pit stop durations. For the

non-refuelling case the estimate of pit stop duration was found by grouping pit stop duration times by circuit and then taking the median. The median is taken over the mean, as the mean can be sensitive to outliers and pit stops can sometimes go wrong resulting in much longer pit stops than usual. While different teams may be more proficient at pit stops than others on average, these differences are fairly small in comparison to the whole pit stop time.

For the refuelling scenario, the pit stop calculation is more complicated. If a driver wants to do a long stint then refuelling time is the determining factor to the overall time of the pit stop. As a consequence of only have pit stop data from 2011 onward, an analysis on refuelling time isn't available to us. We can however, make an educated guess. We know that per the guidelines when refuelling was allowed that refuelling rates were limited to $12l/s$. From an archived article on the official F1 website [15] an estimate of the average F1 car fuel efficiency is reported to be $75km/100\ l$. Then we just need to know the distance of each track to get our estimate for how long per lap refuelling the car should cost. For example, the Catalunya circuit is $4.655km$ long and so the calculation as in equation 3.11 gives us an estimate of $0.29s/lap$ to refuel.

$$\frac{75}{100}\left(\frac{l}{km}\right)\cdot 4.655\left(\frac{km}{lap}\right)\cdot\frac{1}{12}\left(\frac{s}{l}\right)=0.29\left(\frac{s}{lap}\right) \tag{3.11}$$

A function was written that takes the median time of the pit stops across each circuit as a base time. Then from that base time it subtracts 2.92 seconds which is the median time it takes a team to service a car without refuelling [13]. The function looks at how many laps the strategy is calling for in the next stint and multiplies that by the circuits estimated refuelling rate. Whichever is longest out of the refuelling time or the car service time gets added on to the base time and is returned as the pit stop penalty.

3.5 Optimising strategy

After making the function that takes in a strategy and returns a race time, to find the optimal strategy we would like to find the minimum of that function. In theory this could be done with some sort of minimiser such as a trust region minimiser in section 2.1.2 however because 'strategy' isn't a continuous variable an integer programming version of such algorithms would have to be implemented.

Our implementation of finding the minimum of the race time function is to do a brute force grid search of the entire strategy space. A script was written that takes: the model parameters, the tyres available and the optional 'first tyre' argument for when a driver qualified in the top ten. The script then generates every possible legal strategy that the driver could use in the race. If the 'first tyre' argument is used then it only returns the strategies that run the same compound for the first stint. For each strategy the script then calculates the modelled race time as in section 3.4. The strategy with the quickest time is recorded as the 'best' or 'optimal' strategy and strategies that came within 10 seconds of that are recorded as 'close strategies'.

Then because teams may care about the fastest lap as it offers the opportunity to get a bonus point, the script loops through each of the 'close strategies' to calculate their fastest laps. It then also returns the strategy with the fastest lap as the 'quick-lap strategy'.

The methodology for finding the optimal strategy for the refuelling and non-refuelling scenarios is exactly the same except for the differences in calculating the strategy race time as in equation 3.10.

Part of the analysis is testing the robustness of the strategy to changes in the model parameters. For this analysis the parameters for the coefficients of the penalties were found by fitting the data against multiple drivers as outlined towards the end of section 3.3.1. The least-squares optimiser in `lmfit` returns the standard errors of the parameters by calculating the covariance matrix of the parameters as in section 2.1.4. Then the optimiser script is rerun using parameters adjusted by a factor of their standard error to see how the optimal strategy changes and how well the 'best strategy' performs with such changes.

4. Analysis

4.1 Comparison of Different Strategies Found with the Optimiser.

One of the biggest choices a strategy engineer has to make is the number of stops their driver should take during the race. In this section the differences between the optimal strategies for different number of stops are explored. This analysis was done using the parameters found by modelling the race data from Bottas during the 2015 Spanish GP, but it's important to note that any set of parameters could be used, either from adjusting Bottas' parameters or by modelling a different driver.

The optimiser was run several times, each time the considered strategies were restricted to contain a certain number of stops. In this way the optimal strategies restricted to a certain number of stops are calculated. Additionally, because Bottas qualified in the top ten, it's known that he was obligated to start on the medium tyre and so the 'first tyre' argument in the optimiser was set to 'Medium'. The strategies found as well as their modelled times are shown in table 4.1. The differences between the modelled times of the different strategies for different number of stops, highlight just how important it is to get this decision right. The quickest strategy is the three-stop with a time of 6106 seconds followed by the two-stop strategy which is just under 10 seconds slower. This is more than enough time to change a drivers position in a race. To put it into perspective a common punishment for dangerous driving is a 5 second penalty. In this instance, choosing the wrong number of pit stops is equivalent to two of these penalties. On the other hand, if a team feels like the difference isn't that bad and trust their driver not to overtaken, then they may choose the less risky option and go for a two-stop. Decisions like this are often made on tracks like Catalunya that are difficult to overtake on.

Looking at table 4.1 we can see that as the number of pit stops increases, the lap on when the driver first pits comes earlier and generally the number of laps between pits decreases. Bottas on the quickest one-stop strategy first comes into pit on lap 32 compared to lap 17 on the four-stop. Strategy engineers watching other drivers can use this information to deduce what strategies their rivals are running. If they can see that their rival is pitting early enough to do an aggressive 3 or 4-stop strategy then they might choose to react to that and follow suit.

What's interesting is how different the actual strategy that Bottas used is to the optimal two-stop strategy. There may be two reasons for this. The first could be

Description	Strategy(Stop laps: Tyres used in stints)	Modelled Time (s)
Used strategy	14, 42: Medium, Medium, Hard	6126.1
Best 1 stop	32: Medium, Hard	6159.8
Best 2 stop	27, 50: Medium, Hard, Medium	6115.6
Best 3 stop	19, 29, 48: Medium, Hard, Medium, Medium	6106.3
Best 4 stop	17, 34,51,58: Medium, Medium, Medium, Hard, Hard	6123.61
Quick-lap	22, 43, 57: Medium, Medium, Hard, Medium	6115.7

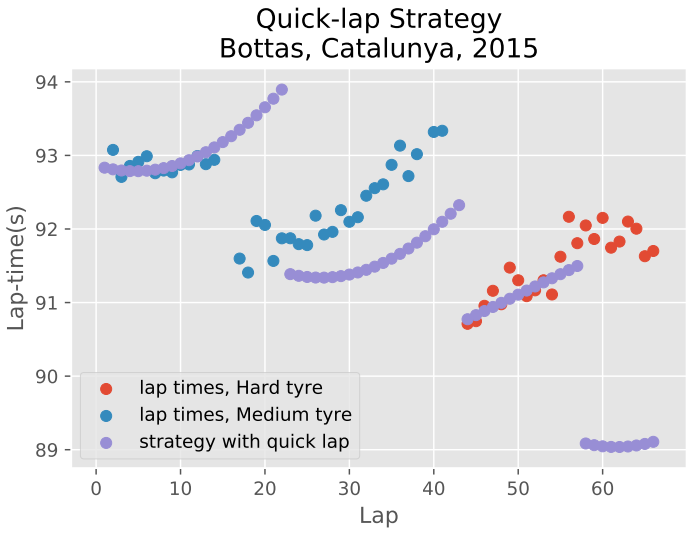
Table 4.1: Table of the optimised strategies for n number of stops, along with their modelled times. Also shown is the 'quick-lap' strategy which is the strategy with the fastest lap out of those modelled within 10 seconds of the optimal strategy. All strategies were found by the strategy optimiser using the parameters fitted on Bottas in the 2015 Spanish GP. The fastest performing strategy is highlighted in the table for ease of identification.

that the strategy engineers for Bottas found different parameters for the fuel and tyre penalties when measuring them from the practice laps. One of the issues that strategy engineers have to deal with is the changing conditions on the track that affect how the car performs over the course of the event. The second reason could be that the tyres that Bottas started on were the same ones that he got his fastest lap in the second round of qualifying. This means that the tyres will have some wear on them before entering the race. Bottas' engineers will likely have taken this wear into account when forming their strategy the night before. The pre-race tyre wear was not implemented into our optimiser, mostly because the data for how many laps each tyre runs before the race isn't available on the datasets we looked at.

It's also worth pointing out that the modelled time of the race strategy that Bottas used was around 6126 seconds whereas if you add up all of the actual lap times it comes out to 6132 seconds. While we would expect there to be a difference as there is always going to be a certain amount of variability in the lap times around the model. This six second difference may also be due to other factors. For example, it could be due to outliers not as a result of pit stops throughout Bottas' race that would not be accounted for in the model. Or it could be due to the model not accounting for changing conditions throughout the race. Despite this difference we still think its fair to compare the relative differences between strategies. For example, we can still conclude that a three-stop strategy is going to be faster than the one-stop, it just that we have to be careful about the precision that we can report that difference to be.

The optimal strategy as well as the quick stop strategy for Bottas are plotted in figure 4.1. Both are 3 stop strategies which is unsurprising given that the optimiser was restricted to looking for a quick-lap strategy amongst those that were within ten seconds of the fastest one. It seems like the goal of the quick-lap strategy is to get the driver on the softest tyre available with as few laps to go in the race as possible, while keeping within the allowed maximum time. This makes sense as the soft tyres will be the quickest and they will be at their maximum grip early on into the stint.

(a)



(b)

Figure 4.1: (a) Optimal and (b) quick-lap strategies found from the model parameters fitted on the data from Bottas in the 2015 Spanish GP, as described in table 4.1. The modelled strategies' lap times, which are calculated using equation 3.2 are shown in purple and are overlaid on top of the lap times of Bottas, which are coloured according to the tyre compound he used for each lap.

4.1.1 Refuelling Scenario

The procedure of finding the optimal strategies with different number of stops was repeated with the lap times being calculated with refuelling being allowed as in section 3.4. The optimal 1, 2, 3 and 4-stop refuelling strategies as well as the quick-lap strategy were found and are shown in table 4.2. The parameters were kept the same as the non-refuelling analysis. While this would likely not be the case as the cars will be able to drive faster on average which may mean that the tyres wear out quicker. It is still an interesting comparison to see how allowing refuelling might affect the optimal strategies.

Looking at table 4.2 we can see that the times to complete the refuelling strategies are much quicker than the non-refuelling ones. The quickest strategy under the refuelling scenario is over 100 seconds faster than the quickest strategy without refuelling. This really highlights how important fuel weight is to a driver's lap time and shows that for the non-refuelling scenario it's critical to put in just enough fuel to complete the race.

Description	Strategy (Stop laps: Tyres used in stints)	Modelled Time with refuelling (s)
Used strategy	14, 42: Medium, Medium, Hard	6042.1
Best 1 stop	34: Medium, Hard	6095.1
Best 2 stop	24, 43: Medium, Hard, Medium	6027.0
Best 3 stop	19, 31, 48: Medium, Hard, Medium, Medium	6005.2
Best 4 stop	16, 26, 41, 56: Medium, Hard, Medium, Medium, Hard	6014.3
Quick-lap Strategy	21, 41 ,56: Medium, Medium, Hard, Medium	6013.5

Table 4.2: Table of the optimised strategies for n number of stops, along with their modelled times **under the scenario that refuelling is allowed**. Also shown is the 'quick-lap' strategy which is the strategy with the fastest lap out of those modelled within 10 seconds of the optimal strategy. All strategies were found by the strategy optimiser using the parameters fitted on Bottas in the 2015 Spanish GP. The fastest performing strategy is highlighted in the table for ease of identification

Comparing the refuelling strategies to their non-refuelling counterparts in table 4.1 we can also see that the stint durations are more even across the race. For example, for the two-stop strategy, the non-refuelling strategy calls for stints of length 27, 23 and 16 laps whereas the refuelling stints are 24, 19 and 23 laps long. In the refuelling optimiser long stints like the 27 lap one are discouraged as longer stints mean more fuel on board which means longer lap times. It's also interesting to note that strategies with more stops might be encouraged under refuelling. The difference between the two and three-stop strategies under refuelling is 21.8 seconds where as it's 9.3 seconds without refuelling. Going more aggressive by choosing a four-stop strategy over the three-stop strategy would cost you an extra 8.9 seconds under refuelling versus 17.3 seconds without. Teams opting to risk different strategies than everyone else under these conditions are more likely to choose an aggressive four-stop than the conservative two-stop.

The optimal strategy and the quick-lap strategy for the refuelling optimiser are

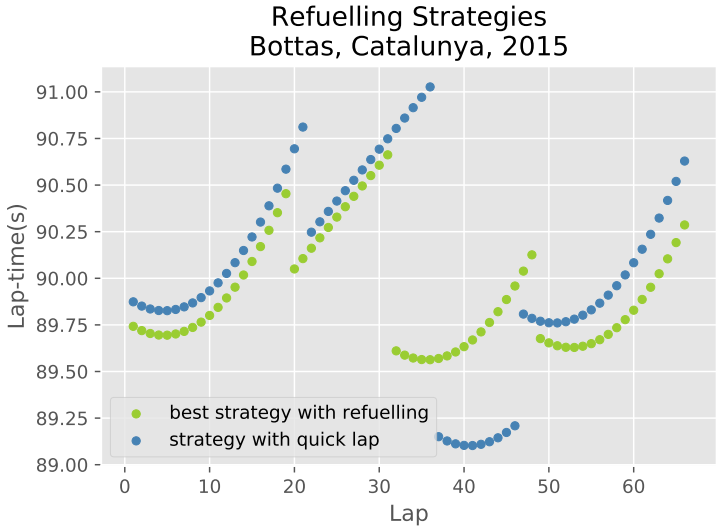


Figure 4.2: Modelled optimal and quick-lap strategies found with the optimiser under the scenario that refuelling is allowed using parameters fitted on Bottas in the 2015 Spanish GP, as described in table 4.2. Lap times are calculated using equation 3.9, which makes the assumption that cars take on just enough fuel to finish the stint.

plotted in figure 4.2. Like in the non-refuelling optimiser both are three-stop strategies. Unlike the non-refuelling optimiser however, is that the quickest lap of the quick-lap strategy is on the 3^{rd} stint and not the last stint. This is interesting because it shows that teams going for the quick-lap bonus point don't have to wait until the end of the race to go for the quickest lap. Another interesting point is that if you commit to a quick-lap strategy then you are likely also committing to a higher stop strategy. If we increase the maximum time constraint for strategies considered then quite a few four-stop strategies have a quicker lap time. Adding another stop lets you run a very short stint, which means that the car will have hardly any fuel at all to give the best opportunity to get the fastest lap. All this means that teams can be much more reactive about trying to get the fastest lap bonus point. For example, if a team thinks they have a lot of time to work with before losing position they can weigh the decision to add a short stint into their strategy to go for the bonus point.

For comparison, the optimal refuelling strategy is plotted in figure 4.3 along with the same strategy but with lap times calculated using the non-refuelling function. The times of the stints in the non-refuelling race are offset by where in the race the stint starts. Whereas the times of the stint in the refuelling race are offset proportional to how long each stint lasts.

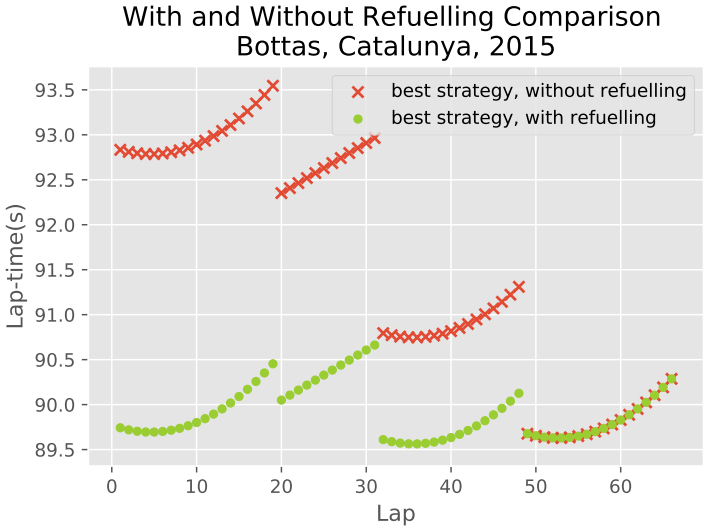


Figure 4.3: Plot of refuelling vs non-refuelling lap times. Both races are modelled running the optimal refuelling strategy. The lap times for the scenario without refuelling are calculated using equation 3.2. The lap times for the strategy with refuelling are calculated with equation 3.9.

4.2 Robustness of Found Strategy

In this section we will to try to quantify how reliable the optimal strategy found by the strategy engineer using this methodology is. For this analysis, the parameters for the penalties were found by modelling on the two Williams-Mercedes drivers Bottas and Massa using the methodology described in section 3.3.2. With the standard errors being taken from the covariance matrix as described in section 2.1.4. This choice was made so that there was more data for each type of tyre to fit on so that we had more reasonable standard errors for this analysis. This may also more accurately reflect the amount of data available to the strategy engineers as they at least will have data from both their team's drivers from the free practice at each event.

The parameter values used for the penalties in this analysis are shown in table 4.3. For the t_{base} parameter, Bottas' base time was used, but that doesn't matter as the base time will have no contribution to the optimal strategy as for each strategy its contribution to the total time is just t_{base} multiplied by the number of laps. All the analysis for this section was done with the non-refuelling optimiser. Although if you were able to find appropriate parameters for a race with refuelling, the same process could easily be repeated for the refuelling case.

Using the parameters in table 4.3 the optimal strategy (also referenced as the 'original best strategy' in this analysis) was found to be a two-stop on laps 23 and 43 switching from a medium to a hard tyre, then back to a medium. This strategy was found to have a modelled time of 6039.1 seconds. In this analysis a series of experiments were ran. In each experiment, one or more of the variables would be changed by a multiple of their standard error and the optimiser would be run against

	F	T_m	T_{m2}	T_{mb}	T_h	T_{h2}	T_{hb}
Parameter values	0.04239	0.05181	0.001471	0 (fixed)	0.09342	0	0.04158
Standard errors	0.00226	0.0245	$9.63e^{-4}$	0	0.03783	0.00148	0.250

Table 4.3: Table of parameters for the penalty coefficients when modelled on re-centered data from Bottas and Massa simultaneously. The subscript T_m represents the medium tyre compound, and the subscript T_h represents the hard tyre compound.

this new set of parameters. The best strategy for the set of parameters was found and recorded as well as the difference between the modelled times of the new best strategy and the original best strategy. In this way, we can see how far behind the optimal strategy we would be if the true parameters were different than what we thought they were. We can treat this difference as the error in our strategy. The results of the experiments are shown in table 4.4.

The first thing to note from table 4.4 is that the fuel penalty doesn't change the optimal strategy at all. In our model the fuel penalty is only dependant on the "lap" variable and will be the same no matter how you change the stintlap or tyre variables. Of course if we were using a more complex model where we consider if the tyre wear is affected by how much fuel is in the car, then this may change. The fuel penalty did however affect how quickly the car was modelled to complete the race. As the fuel penalty was increased by 2 standard errors the modelled time was slower, increasing from 6039 seconds to 6049 seconds. An increased fuel penalty implies that the car would need to be filled up with more fuel to complete the race, and therefore would have slower lap times. Of course in practice, if the strategy engineer underestimates the fuel burn rate then the driver will have a faster car but might not have enough fuel to push the car as hard as he wants for the duration of the race. If he overestimates and fuels up the car too much then the driver will have a slower car throughout the race. This emphasises the need to correctly estimate the fuel penalty parameter so that the driver can have a competitive race.

The optimal strategy does change however when the tyre penalty coefficients are adjusted. These coefficients can be thought of as measures of the tyres' performance, the better the tyres perform the more the engineer is encouraged to use them in their strategy and vice versa. It seems like getting the relative tyre performance wrong by a small amount doesn't affect the optimal strategy too much. Changing any of the tyre parameters on their own by plus or minus one standard error results in the optimal strategy still being a two-stop. The strategy just changes slightly with one change in the tyre choices or small adjustments to the optimal stop laps. The biggest change was when the T_h variable was adjusted by -1 standard error, with the optimal strategy changing the first stop lap to 18 from 23. This also gave us our biggest error in strategy of 6.5 seconds.

From this, we would argue that the optimal strategy is robust to small changes in the modelled parameters. Strategy engineers can be confident that their proposed strategies will be fairly close to the optimal one, assuming that their model parameters are mostly accurate. It might be that their planned pit stops are a few laps

Experiment	Best strategy (Stop laps: Tyres used in stints)	Time difference to original best (s)
no change	23, 43: Medium, Hard, Medium	0
F+2 se	23, 43: Medium, Hard, Medium	0
F-2 se	23, 43: Medium, Hard, Medium	0
T_m+1se	20, 43: Medium, Hard, Hard	1.56
T_m-1se	25, 41: Medium, Hard, Medium	0.84
T_{m2}+1se	21, 42: Medium, Medium, Hard	1.02
T_{m2}-1se	25, 41: Medium, Hard, Medium	0.78
T_h+1se	25, 41: Medium, Hard, Medium	1.26
T_h-1se	18, 42: Medium, Hard, Hard	6.52
T_{h2}+1se	24, 48: Medium, Medium, Hard	0.66
T_{hb}+1se	23, 47: Medium, Medium, Hard	0.0689
T_{hb}-1se	22, 44: Medium, Medium, Hard	0.34
T_m+2se	18, 42: Medium, Hard, Hard	10.4
T_m-2se	27, 39: Medium, Hard, Medium	4.41
T_{m2}+2se	19, 42: Medium, Hard, Hard	4.43
T_h+2se	19, 28, 47: Medium, Hard, Medium, Medium	7.3
T_h-2se	13: Medium, Hard	34.3
T_{hb}+2se	25, 42: Medium, Hard, Medium	0.55
T_{hb}+3se	19, 27, 46: Medium, Hard, Medium, Medium	3.1
T_{hb}-2se	20, 43: Medium, Hard, Hard	6.19
T_{hb}-3se	19, 43: Medium, Hard, Hard	12.78
T_m+1se, T_h+1se	18, 31, 49: Medium, Hard, Medium, Medium	2.22
All +1 se	18, 30, 48: Medium, Hard, Medium, Medium	9.62
All -1se (except T_{h2})	34: Medium, Hard	3.024
T_m, T_{m2} +1se, T_h, T_{hb} -1se	12, 39: Medium, Hard, Hard	32.4
T_m, T_{m2} -1se, T_h, T_{hb} +1se	29, 36: Medium, Hard, Medium	13.4

Table 4.4: Table of how the optimal strategy changes as the parameters are adjusted. The highlighted row is the optimal strategy for the parameters with no changes. The 'time difference to original best' column represents the difference in time if the highlighted strategy was ran rather than the best strategy found for those conditions. The values for the parameters as well as their standard errors (se) are the same as in table 4.3

either side of the optimal but this probably wouldn't concern them anyway. This is because during the race pit stops are often adjusted by a couple of laps anyway in efforts to keep the driver from re-entering the race into traffic. A strategy engineer knowing all this could give a range of laps that would be acceptable for the driver to pit in. Then during the race they would try to find the best opportunity for the driver to pit within that window.

If however, we get the relative performance of the tyres too wrong, then the error in our strategy starts to increase quite quickly. If you under or over estimate the tyre performance by too much then you are at risk of getting the optimal strategy significantly wrong. For example, if it turns out the hard tire penalty coefficient T_h is two standard errors below what we initially thought, then the optimal strategy becomes a one-stop. The original strategy that we would have proposed becomes 34 seconds slower than the optimal. Perhaps a more realistic scenario would be underestimating one tyres performance while at the same time overestimating the other by a standard error. For example, if we increase T_m and T_{m2} by 1 standard

error and decrease T_h and T_{hb} (not T_{h2} as we restrict the T_2 variables to be positive) by one standard error, the error in our proposed strategy becomes 32 seconds.

One possible solution for the strategy engineer to avoid such situations would be to do a Bayesian update, as described in [16], of the parameters as the race goes on, using the fitted parameters found from the practice session as prior guesses. This would give a mathematical framework for how the strategy engineer could adjust their proposed strategy during the race. For example, switching to a one-stop strategy if the hard tyre looks to be performing much better during the race than expected.

4.3 Vettel Canada 2012 Case Study: Strategy Debrief.

In this section we will look at the strategy of a specific driver of a specific race and try to use the tools we've developed in this project to look at the decisions behind their strategy and what could have gone better. The driver we're looking at is Sebastian Vettel during the Canadian GP on the Villeneuve circuit in 2012.

Vettel started the race in pole position ahead of Hamilton and Alonso. He started the race well, building a two second lead ahead of Hamilton by lap 10. After this Vettel's tyres started to show signs of degradation and Hamilton gained that time back until he was within overtaking distance by lap 16. In response, Vettel came into pit on lap 17 switching from the super soft to the soft tyre. However, it was a relatively slow pit stop by his team and by lap 20 after the end of the first round of pit stops between the top three, Vettel was in third. Red Bull would have thought that it was mandatory for Vettel to come into the pits on lap 17 in order to keep his position as his tyres were showing clear signs of wear by that point.

After lap 20 Hamilton had taken the lead and was building an advantage at a pace of around half a second a lap. This advantage in pace for Hamilton was either due to his car performing relatively better on the soft tyre in comparison to Alonso and Vettel, or because Hamilton was committed to doing a two-stop and so could drive faster without having to worry about tyre wear. On lap 50 Hamilton came into the pits changing to a fresh set of soft tyres. Both Vettel and Alonso made the decision to gamble on the soft tyres holding for the rest of the race in order to gain the position on Hamilton back by not pitting. However, the gamble didn't pay off as their tyres' performance dropped off quicker than they hoped and Hamilton took Vettel on lap 64 and Alonso on lap 65. After being taken by Hamilton, Vettel decides to abandon the one-stop strategy and comes in for a set of super softs for the remainder of the race. He comes out of the second pit stop in 5th behind Grosjean and Perez who both did one-stops, but pitting at later laps than the top three. The race ends with a podium of Hamilton, Grosjean and Perez followed by Vettel in 4th, who also go the fastest lap of the race. Alonso who committed to the one-stop dropped from second to 5th in the space of the final 5 laps.

Clearly Vettel missed out on a podium finish as a result of a strategy gamble that didn't pay off. With hindsight and now being able to measure Vettel's tyres' true

performance in the race, we can now answer the following questions:

- What was Vettel's theoretical optimal strategy?
- Should Vettel have stayed out beyond lap 17 if he was committing to a one-stop?
- If Vettel was forced to do a stop on lap 17, what should have been the strategy from that point on?

4.3.1 Vettel's Optimal Strategy

Vettel's lap times were cleaned and modelled using the same methodology described throughout section 3.3.1. The fitted parameters are shown in table 4.5 and modelled lap times are shown in figure 4.4. The tyres seem to be behaving as expected as the harder tyre (the 'Soft' compound) wears out at a less severe rate but runs at an estimated 0.36 seconds a lap slower. Vettel's modelled time for the whole race is 5564.5 seconds while his actual time found by adding all of his lap times together is 5556.9 seconds. During this analysis it's therefore important to keep in mind that there is around a plus 7.6 second difference between the modelled time and Vettel's actual race time. We are not predicting what his final race times would be, we are simply estimating the relative differences in the various strategies he could have chosen.

	F	T_{ss}	T_{ss2}	T_{ssb}	T_s	T_{s2}	T_{sb}	t_{base}
Parameter values	0.061	-0.171	0.0137	0 (fixed)	-0.077	0.0022	0.358	76.66

Table 4.5: Fitted model parameters on Vettel's lap times on the 2012 Canadian GP. The subscript T_ss represents the super soft compound and the subscript T_s represents the soft compound.

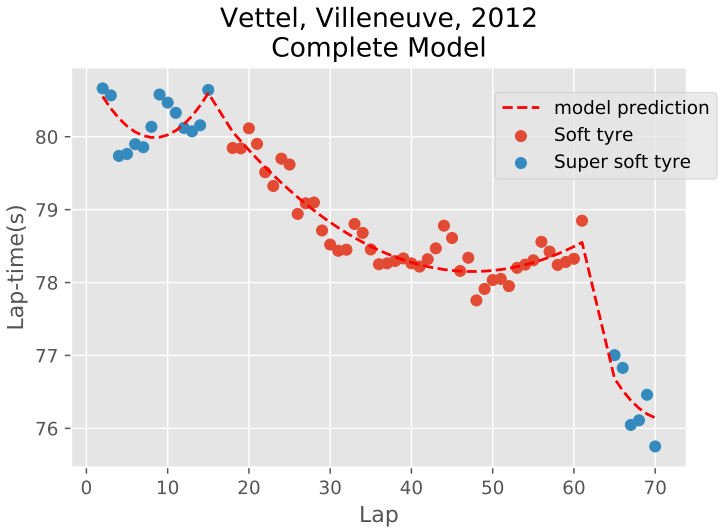


Figure 4.4: Scatter plot of the data for Vettel in the 2012 Canadian GP. The data was fitted against the model function as described in equation 3.3, which is shown as a red dashed line. The lap-time data points are coloured according to the tyre compound he used for each lap

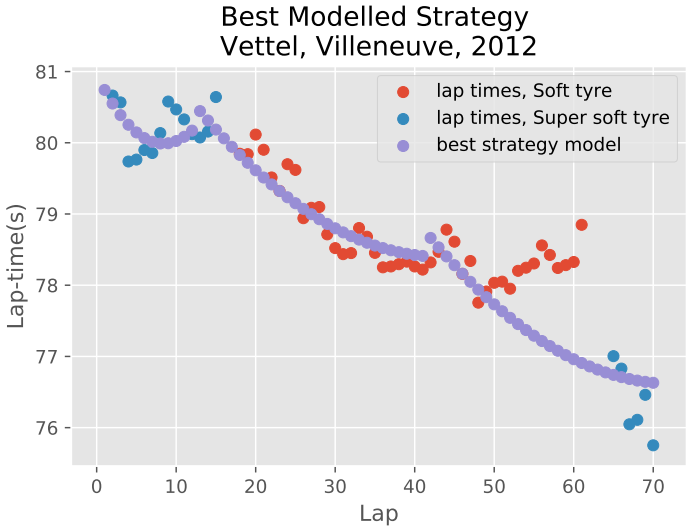


Figure 4.5: Optimal strategy found for Vettel in the 2012 Canadian GP overlaid on top of his actual race data. The modelled strategies' lap times, which are calculated using equation 3.2 are shown in purple and are overlaid on top of the lap times of Vettel, which are coloured according to the tyre compound he used for each lap

The parameters were put through the strategy optimiser and the considered strategies were restricted so that they all used the super soft tyre in the first stint. The quickest strategy found was a two-stop with stop laps on lap 12 and 41, both switching to soft tyres. A visualisation of the modelled lap times under this strategy is shown in figure 4.5. The total modelled time for this strategy is 5548.5 seconds which is around 16 seconds faster than the modelled time of the strategy he used in race. This 16 seconds if applied as a flat decrease to Vettel's overall race time would put him in first place as he finished 7.3 seconds behind the lead. Of course we can't claim that running this strategy would have actually given him the win as there are lots of other variables to consider. For example, we don't know if pitting on lap 12 would have released Vettel into traffic. It's also quite likely that once Hamilton knew that he had the first place position, which he secured around lap 65, he stopped pushing his car to the maximum in order to increase the likelihood he would finish the race. Therefore the time differential to secure 1st is probably substantially more than 7.3 seconds. On the other hand, we think its fair to claim that if Red Bull could pull off a similar strategy so that Vettel wouldn't be affected by traffic, it would have made him more competitive in fighting for a podium position.

4.3.2 Analysing the One-Stop Strategy

Red Bull decided to run Vettel on the one-stop strategy. We can use his lap time model to check with hindsight whether that was a good idea or not. The comparison of the different strategies considered is shown in table 4.6 and their visualisations are shown in figure 4.6. Firstly the optimiser was run but restricting strategies to a one-stop. This gave us our best one-stop strategy which was stopping at lap 20 and

had a modelled time of 5556 seconds. This is the same strategy that Ferrari with Alonso ran. Compared to the one-stop on lap 17 which Red Bull opted to try for, stopping that three laps later gives an estimated advantage of around 1.5 seconds. Both one-stops were modelled to perform better than the late two-stop that Vettel ended up going for by 7.1 and 8.6 seconds respectively.

Description	Strategy (Stop laps: Tyres used in stints)	Modelled Time(s)
Used strategy	17, 64: Super soft, Soft, Super soft	5564.5
Optimal strategy	12, 41: Super soft, Soft, Soft	5548.5
Best one-stop	20: Super soft, Soft	5555.9
Proposed one-stop	17: Super soft, Soft	5557.4
Best strategy from pitting on 17	17, 44: Super soft, Soft, Soft	5551.8

Table 4.6: Table of the strategies relevant to Vettel's race found by the strategy optimiser and their modelled times. Made using the parameters fitted on Vettel in the 2012 Canadian GP.

Since Red Bull decided to gain the position lost in the first round of pit stops by following through with a one-stop and failed. A question that a strategy engineer may be interested in is what should they have done in hindsight? This was answered by running the strategy optimiser but simply restricting the considered strategies to be the ones with the first pit stop on lap 17. The highest performing strategy found was to do a second stop, switching to a fresh set of soft tyres on lap 44. This is more in line with the strategy Maclauren ran with Hamilton who used the same tyres but stopping on laps 18 and 50. Running this strategy is modelled to run at 5551.8 seconds, around 5.6 seconds better than the proposed one-stop, and 12.7 seconds better than his used strategy.

Vettel finished less that 5 seconds behind 2nd place Grosjean and 2 seconds behind Perez who both ran one-stops, and presumably couldn't have pushed much harder. While its hard to gauge how quickly Hamilton could have finished the race if he pushed for the whole duration. We think its fair to say that Vettel probably could have kept a place on the podium had he gone for a more optimised two-stop strategy given the difference in modelled times.

Interestingly, even though in this analysis we found results that suggest that Vettel should have stayed out on lap 64, as his used strategy is modelled to be 7 seconds slower than if he stayed out, Vettel and Red Bull praised this decision after the race. Vettel in a post-race interview said " We decided in the end to come in again which was a great call, given what you can lose with seven laps to go" [4]. Perhaps this means that Red Bull had reason to believe that the tyre penalty wouldn't follow the same quadratic behaviour that we were observing prior to the pit on lap 64. Or they may have felt vindicated by Alonso who committed to the one-stop but went from 2nd to 5th in the space of the last five laps. As explained in section 2.2 we're using Taylor polynomials to approximate the tyre penalty using the first lap as a starting point. It may be that around 40 laps into the stint, the 2nd order approximation becomes insufficient. The behaviour of tyre penalties beyond

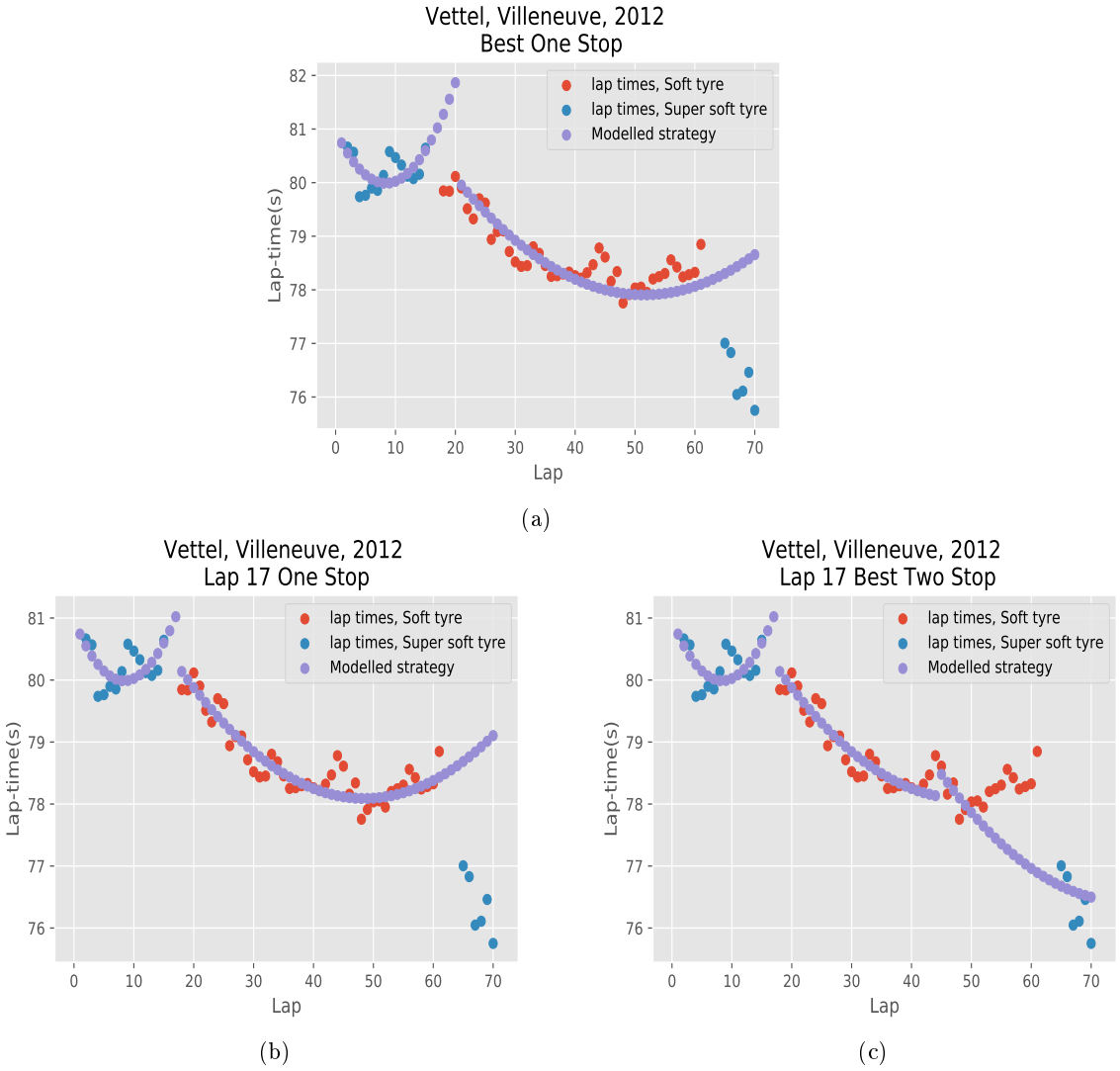


Figure 4.6: Comparison of the modelled strategies that Red Bull may have considered for Vettel around lap 17. (a) Is the best one-stop strategy found that stops on lap 20. (b) Is the one-stop strategy that Red Bull attempted, stopping on lap 17. (c) Is the best strategy found given that Vettel stops on lap 17, making a second stop on lap 44. The modelled strategies' lap times, which are calculated using equation 3.2 are shown in purple and are overlaid on top of the lap times of Vettel, which are coloured according to the tyre compound he used for each lap.

the point of when teams would normally bring the cars into pit is an interesting new line of inquiry.

5. Discussion

5.1 Limitations and Further Considerations.

In this project the goal was to use modelled lap times to compare different F1 strategies and return the "optimal" strategy. The optimal strategy was defined as the strategy with the shortest overall race time. In reality, F1 teams care about their position at the end of the race rather than the theoretical time it should take them to complete it. Of course, race pace is a big predictor of how well drivers finish in a race but it isn't the only factor. A team wanting to replicate this project but adapt it so it predicts race position could add to the model by considering the interactions between other drivers. For example, adding effects such as an overtaking model, lap time variability and tracking the probabilities of a safety car would allow the engineer the possibility of running a Monte Carlo simulation of a race. Then the engineer could choose the strategy that does the best on average out of all the simulations. Work that was published while we were working on this project by Heilmeier [18], suggests that strategies that just consider lap times are fragile when probabilistic effects are considered. However, testing all the possible strategies against a Monte-Carlo simulation probably isn't computationally feasible. A possible use for research such as this project could be to provide or inform candidate strategies for a Monte-Carlo race simulation to consider.

The lap time model itself could be improved by considering the relationships between the fuel weight and fuel penalty and the tyre wear and the tyre penalties. These physical properties are measurable by F1 teams during the race, and so if they could be tied to their respective penalties, teams would be able to measure how fast their cars are capable of going before the lap starts. Then the modelling process would change to predicting the tyre wear and fuel load throughout the race instead of predicting the lap times directly. This approach of modelling would allow for more precision in decisions about changing strategies mid-race. It would also allow engineers to track how the tyres are performing and relay clearly defined instructions to drivers so that the tyre wear keeps to target for the planned strategy.

The fitting of the model parameters could be improved by having access to more data that isn't published by F1 such as the free practice lap times. This data is less likely to be affected by drivers interacting with other drivers. However, this data is also taken a couple of days before the race, so track conditions may change. To repeat a point made in section 4.2, a possible method to infer a set of parameters

in a reproducible way would be to do a Bayesian update on the ones found during practice. This would allow the teams to define how much they trust the practice session data numerically. Then they could strike a balance between an over-reliance on parameters that might have changed between practice and race day, and not utilising previously taking data at all.

It would also be useful to have data about the wear on the tyres available to a driver before the race. Right now, the model assumes every tyre available to the driver is fresh. However, it's often the case that several laps will have been done on a set of tyres either during qualification or free practice. Having this information would ensure that the engineer doesn't strategise around using worn tyres. For example, using a soft tyre without taking into account that the only set of soft tyres already has five laps worth of wear on it.

Another potential limitation to this methodology was in the outlier detection section 3.2. For an isolation forest, the usual decision threshold for a point to be considered an outlier is when its anomaly score is larger than 0.5. This represents the point having an expected height less than the average unsuccessful search in a binary search tree. The decision was made to increase this threshold to 0.56 as that seemed to give "reasonable" results for the races and drivers that were tested. Someone trying to implement their own version of this research may choose a different value that works better.

Looking at the anomaly scores against the lap variable as in figure 3.6, it appears that the laps toward the beginning and end of the stints have larger anomaly scores on average than the laps in the middle. To consider and improve on this situation, we would have to investigate the cause and effect of this trend. It may be that lap times are less likely to conform to a quadratic model towards the beginning and end of the stints. For example, drivers are often told to drive more cautiously as the engineers try to fit them into a pitting window and therefore we wouldn't want to model those laps. On the other hand, it may be that they are laps that do describe the relationship between lap and lap-time, it's just that the algorithm is biased against them. If this is the case, then we may want to consider either a different outlier detection method or change how we treat the outliers when we find them.

In this project when we found an outlier, we assumed that it was extremely large to the point of uselessness due to an event that was irrelevant to our model such as a pit stop or a mistake. However, the conditions that led to that event may themselves be relevant to our analysis and may have caused a large lap time anyway had that event not occurred. For example, a driver that took a spin on a later lap in the stint would have recorded a large lap time that would have been rejected for modelling. However, it's possible that being late into the stint and having worn tires contributed to the spin, and even without the spin, the driver would have recorded a slow lap anyway. This in conjunction with the potential biasing against later laps in the stint, may mean that we are biasing toward quicker lap times in our modelling. One possible solution to help this would be instead of ignoring all the outliers we identify, we could impute their values. Multiple imputation methods such as the multiple imputation chained equation (MICE) algorithm [7], would be ideal for this

job as they impute values that preserve the structure within the data as well as that structure's uncertainty. This approach is used extensively and found to work well in practice in other fields [29], and we see no reason why it wouldn't work for F1 lap times.

The way the optimiser found the minimum time out of all the possible strategies was a brute force search with every possible strategy being simulated. While this guarantees that the fastest strategy is found, it's slow when there are a large number of strategies to consider. The number of possible strategies that are possible increases factorially with the number of stops and the number of tyres that are included. For example, the number of possible four-stop strategies for a 70 lap race with 2 different tyre choices is around 27.5 million, which takes 90 minutes to run through on our computer (Intel Core i7-8550U, 1.80GHz). Although F1 teams have access to more computational resources than we do, this may become a problem if strategies with more than four stops are considered. This is especially true if a team wants to adjust strategies mid-race where quick decisions are critical. A possible area of future research in this project could be in implementing either genetic algorithms [20], or integer programming algorithms [8], for a more efficient search of the strategy space. Successfully implementing these algorithms may allow research like this to be applied to racing formats such as endurance racing or NASCAR, which typically have many more pit stops than F1.

5.2 Conclusion

In this project we have built a simple model to simulate a F1 driver's performance in a GP while running different types of tyres. We then built a tool that takes this model and finds the optimal pit stop strategy for that driver during the race. We found that the suggested strategies are robust to small changes in the tyre parameters. However, they don't hold up when large mistakes in the parameters are made. We then applied these tools to a case study on Vettel's strategy during the 2012 Canadian GP, which showcased how F1 teams could use similar tools to improve their strategy decision making, both before and during the race. In the discussion, we explored some of the limitations to this analysis as well as suggesting improvements to both the modelling and the strategy optimisation. We have shown that if F1 teams can pick the correct strategy, then they will have a significant competitive advantage over their rivals. Tools such as those developed during this project, that can help them select good strategies in a reproducible way, should be implemented to help a team achieve consistently better results throughout a racing season.

Bibliography

[1] Ergast developer api. URL http://ergast.com/mrd/.

[2] URL https://www.racefans.net/statistics/.

[3] Y. Bard. *Nonlinear parameter estimation.* Academic Press, New York, 1974. ISBN 0120782502.

[4] A. Benson. Lewis hamilton becomes the seventh winner of 2012 in canada, 2012. URL https://www.bbc.co.uk/sport/formula1/18384099.

[5] P. R. Bevington and D. K. Robinson. *Data reduction and error analysis for the physical sciences; 3rd ed.* McGraw-Hill, New York, NY, 2003. URL https://cds.cern.ch/record/1305448.

[6] M. A. Branch, T. F. Coleman, and Y. Li. A subspace, interior, and conjugate gradient method for large-scale bound-constrained minimization problems. *SIAM Journal on Scientific Computing*, 21(1):1–23, 1999.

[7] S. v. Buuren and K. Groothuis-Oudshoorn. mice: Multivariate imputation by chained equations in r. *Journal of statistical software*, pages 1–68, 2010.

[8] D. Chen, R. Batson, and Y. Dang. *Applied integer programming: modeling and solution. 2011.* John Wiley & Sons.

[9] ChrisG. Formula 1 race data. URL https://www.kaggle.com/cjgdev/formula-1-race-data-19502017.

[10] T. F. Coleman and Y. Li. On the convergence of interior-reflective newton methods for nonlinear minimization subject to bounds. *Mathematical programming*, 67(1-3):189–224, 1994.

[11] K. C. Dieter Rencken. Formula 1 teams' prize money payments for 2019 revealed. 2019. URL https://www.racefans.net/2019/03/03/formula-1-teams-prize-money-payments-for-2019-revealed/.

[12] V. Dovì, O. Paladino, and A. Reverberi. Some remarks on the use of the inverse hessian matrix of the likelihood function in the estimation of statistical properties of parameters. *Applied Mathematics Letters*, 4(1):87–90, 1991.

[13] f1bythenumbers.com. 2019 f1 season: The pit crew, 2019. URL https://f1bythenumbers.com/2019-f1-season-the-pit-crew/.

[14] *2020 FORMULA ONE SPORTING REGULATIONS*. Fédération Internationale de l'Automobile, July 2020. URL https://www.fia.com/regulation/category/110.

[15] Formula1.com. Engine / gearbox (understanding the sport), 2012. URL https://web.archive.org/web/20120412041247/http://www.formula1.com/inside_f1/understanding_the_sport/5280.html.

[16] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. *Bayesian data analysis*. CRC press, 2013.

[17] N. I. Gould, S. Lucidi, M. Roma, and P. L. Toint. Solving the trust-region subproblem using the lanczos method. *SIAM Journal on Optimization*, 9(2): 504–525, 1999.

[18] A. Heilmeier, M. Graf, J. Betz, and M. Lienkamp. Application of monte carlo methods to consider probabilistic effects in a race simulation for circuit motorsport. *Applied Sciences*, 10(12):4229, 2020.

[19] D. E. Knuth. Johann faulhaber and sums of powers. *Mathematics of Computation*, 61(203):277–294, 1993.

[20] M. Kumar, M. Husain, N. Upreti, and D. Gupta. Genetic algorithm: Review and application. *Available at SSRN 3529843*, 2010.

[21] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422. IEEE, 2008.

[22] M. Newville, T. Stensitzki, D. B. Allen, and A. Ingargiola. LMFIT: Non-Linear Least-Square Minimization and Curve-Fitting for Python, Sept. 2014. URL https://doi.org/10.5281/zenodo.11813.

[23] T. pandas development team. pandas-dev/pandas: Pandas, Feb. 2020. URL https://doi.org/10.5281/zenodo.3509134.

[24] B. R. Preiss. *Data structures and algorithms with object-oriented design patterns in C++*. John Wiley & Sons, 2008.

[25] T. Ryan. *Modern Regression Methods*. John Wiley & Sons, 1997. ISBN 0-471-52912-5.

[26] S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, 1965.

[27] T. Strutz. *Data Fitting and Uncertainty: A Practical Introduction to Weighted Least Squares and beyond, Vieweg*. Teubner Verlag, Berlin, 2010.

[28] B. Taylor. *Methodus incrementorum directa & inversa. Auctore Brook Taylor, LL. D. & Regiae Societatis Secretario*. typis Pearsonianis: prostant apud Gul. Innys ad Insignia Principis in ..., 1715.

[29] S. Van Buuren. Multiple imputation of discrete and continuous data by fully conditional specification. *Statistical methods in medical research*, 16(3):219–242, 2007.

[30] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. . . Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. URL https://doi.org/10.1038/s41592-019-0686-2.

Appendices

A. Code used

A.1 Data gathering

This script was used to read in all our data files and make new dataframes that we could then sample from.

```
import pandas as pd
import os
import re

#%% make data dictionary containing all the kaggle files
wdir='C:/Users/jxnny/OneDrive/Documents/school/Leeds/Dissertation/data'
os.chdir(wdir)

#gather name of files
files = []
for dirname, _, filenames in os.walk("."):
    for filename in filenames:
        files.append(str(os.path.join(dirname, filename)))
names = [re.findall('\w*.csv', x)[0].split('.')[0] for x in
 ↪  files]
#intialize dictionary
data_dict = {}
#load data into dictionary using name of files for keys
for i, file in enumerate(files):
    name = names[i]
    data_dict[name] = pd.read_csv(file, encoding = 'latin-1')
#load in tyre data file, note some names in tyres.csv and
 ↪  Drivers.csv had to be changed so that they could be joined
 ↪  due to the different encodings
data_dict['tyres'] = pd.read_csv('tyres.csv',encoding='utf8')

# drop some df feilds to prepare for joins
data_dict['drivers'].drop(columns = ['number', 'code',
 ↪  'url','code','dob','nationality'], inplace = True)
```

```
data_dict['circuits'].drop(columns = ['name', 'location', 'url','alt','country','lat','lng'], inplace = True)
data_dict['races'].drop(columns = ['round', 'name', 'url','date','time'], inplace = True)

data_dict['tyres'][['forename','surname']]=data_dict['tyres'].name.str.split(" ",1,expand=True)
data_dict['tyres']=data_dict['tyres'].drop(columns=['name'])
data_dict['tyres']=data_dict['tyres'].merge(data_dict['drivers'], how='left', on=['forename', 'surname'])

#get rid of (number) in tyre dataframe!!
data_dict['tyres']=data_dict['tyres'].replace(to_replace ='(\s\(\d+\))', value = '', regex = True)

#new dataframe of laptime

df = data_dict['lapTimes'].copy()
df = df.merge(data_dict['races'], how = 'left')
df = df.merge(data_dict['circuits'], how = 'left')
df = df.merge(data_dict['drivers'], how = 'left')
df.drop(columns=['raceId','driverId','circuitId','forename','surname','time'], inplace=True)
#new df for pit data
pitdf=data_dict['pitStops'].copy()
pitdf=pitdf.merge(data_dict['races'], how = 'left')
pitdf = pitdf.merge(data_dict['circuits'], how = 'left')
pitdf = pitdf.merge(data_dict['drivers'], how = 'left')
pitdf=pitdf.merge(data_dict['tyres'], how = 'left')
#some cleaning on the new dataframes
pitdf.drop(columns=['raceId','driverId','circuitId','forename','surname','time'], inplace=True)
#make new field for seconds
df['time']=df['milliseconds']/1000
df=df.drop(columns='milliseconds')
#reorder df
orderme=['year','circuitRef','driverRef','lap','time','position']
df=df[orderme]
#df.to_csv('kaggle_laptimes_cleaned.csv',index=False)
#pitdf.to_csv('kaggle_pitstops_cleaned.csv',index=False)
#%%
#for calculating the median pit time for each circuit
pittimes=(pitdf.groupby(['circuitRef']).milliseconds.median()/1000)\
    .rename("Seconds")

#pittimes.to_csv('pittimes.csv')
```

A.2 Sampler

These functions were used to sample and clean data from the dataframes built in the previous script.

```
import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import IsolationForest
from scipy.optimize import curve_fit

#os.chdir(wdir)


datadir='C:/Users/jxnny/OneDrive/Documents/school/Leeds/Dissertation/data'
laptime_df=pd.read_csv(datadir+'/kaggle_laptimes_cleaned.csv')
pitdf=pd.read_csv(datadir+'/kaggle_pitstops_cleaned.csv')


def polynomial(lap,a,b,c):
    return a*lap**2+b*lap+c

def predict_outlier(data):
    #function that returns a list of whether it thinks a point is
    ↪  an outlier, works  using the isolation forest algorithm
    clf = IsolationForest(n_estimators=100 , max_samples='auto',
    ↪  random_state = 1)
    clf.fit(data)
    out_score=clf.score_samples(data)
    preds = clf.predict(data)
    outlier=[]
    for point in range(len(preds)):
        if clf.score_samples(data)[point]<-0.56 :
            dpoint='outlier'
        else:
            dpoint='inlier'
        outlier.append(dpoint)
    return outlier, out_score
#predic_outlier_cedric and predict_outlier_cedric_2 are an
↪  alternative outlier detection method that I didnt end up
↪  using. Idea is to recursively model data as a polynimal and
↪  remove points 1.96 (or whatever) standard deviations above
↪  the modelled line.
def predict_outlier_cedric(data):
    outlier_detected=False
```

```
    pabc,pcov=curve_fit(polynomial, data['stint lap'],
    ↪ data['time'],p0=[0.1,0,90])
    model_pred=polynomial(data['stint
    ↪ lap'],pabc[0],pabc[1],pabc[2])
    stdev=data['time'].std()
    outlier=[]
    outlier2=[]
    for point in range(len(data)):
        if
        ↪ data['time'].iloc[point]>model_pred.iloc[point]+1.96*stdev
        ↪ or data['stint lap'].iloc[point]==1:
            dpoint='outlier'
            outlier2.append(data.iloc[point]['stint lap'])
            outlier_detected=True
        else :
            dpoint='inlier'
        outlier.append(dpoint)

    if outlier_detected==False:
            return outlier2
    else:
        data['out']=outlier
        return
        ↪ outlier2+predict_outlier_cedric(data[data['out']=='inlier'])

def predict_outlier_cedric2(data):
    outlierlaps=predict_outlier_cedric(data)
    outlier=[]
    for point in range(len(data)):
        if data['stint lap'].iloc[point] in outlierlaps:
            point='outlier'
        else:
            point='inlier'
        outlier.append(point)
    return outlier


#this one cleans using whatever algorithm predict_outlier
↪ function uses
def
↪ sample_race(driver,circuit,year,clean=True,tyres=True,out='isolation'):

    ↪ sample_laptimes=laptime_df[['lap','time']][(laptime_df.year==year)
    ↪ & (laptime_df.driverRef==str(driver)) &
    ↪ (laptime_df.circuitRef==str(circuit))]
```

```
71      pits=pitdf[(pitdf.year==year) &
        ↪  (pitdf.driverRef==str(driver)) &
        ↪  (pitdf.circuitRef==str(circuit))]
72      stint=[]
73      tyre=[]
74      stintlap=[]
75      lapofstint=0
76      sample_laptimes=sample_laptimes.sort_values(by=['lap'])
77      strategy=[]
78      strategy.append(tuple(pits['lap'].values))
79      strategy.append(tuple(pits.iloc[0,7:].dropna().values))
80      for lap in sample_laptimes['lap']:
81          stintnum=1
82          lapofstint+=1
83          for stoplap in pits['lap']:
84              if lap > stoplap:
85                  stintnum+=1
86          #reset stint lap if stint restarts
87          if lap in pits['lap'].values+1:
88              lapofstint=1
89          stint.append(stintnum)
90          stintlap.append(lapofstint)
91          if tyres==True:
92              tyreused=pits.iloc[0,6+stintnum]
93              tyre.append(tyreused)
94      sample_laptimes['stint lap']=stintlap
95      sample_laptimes['stint']=stint
96      if tyres==True:
97          sample_laptimes['tyre']=tyre
98      if clean==True:
99          grouped = sample_laptimes.groupby('stint')
100         outlier1=[]
101         outlier_scores1=[]
102         for stint, group in grouped:
103             #here I find outliers depending on what algorithm I
                ↪  want to use
104             #set expected contamination to be 2 outliers per
                ↪  stint
105             if out == 'isolation':
106                 dummy=predict_outlier(group[['stint
                    ↪  lap','time']])
107                 outlier1.append(dummy[0])
108                 outlier_scores1.append(dummy[1])
109             elif out == 'cedric':
```

```
110
                    ↪  outlier1.append(predict_outlier_cedric2(group[['stint
                    ↪  lap','time']]))
111             outlier2 = [item for sublist in outlier1 for item in
                ↪  sublist]
112             outlier2=np.asarray(outlier2)
113             outlier_scores2=[item for sublist in outlier_scores1
                ↪  for item in sublist]
114             outlier_scores2=np.asarray(outlier_scores2)
115
116         sample_laptimes['outliers']=outlier2
117         sample_laptimes['outlier score']=outlier_scores2
118         group_outlier=sample_laptimes.groupby('outliers')
119         inliers=group_outlier.get_group('inlier')
120
            ↪  inliers=inliers.assign(Driver=driver,race=circuit,year=year)
121         outliers=group_outlier.get_group('outlier')
122
            ↪  outliers=outliers.assign(Driver=driver,race=circuit,year=year)
123         return inliers, outliers,strategy
124
125     return sample_laptimes,[],strategy
126
127 #reads file, calls sample_race for each of the rows in that file
    ↪  and then puts it all together (assumes tyres and clean =true)
128 def sample_file(csvfile):
129     sample_me=pd.read_csv(csvfile)
130     drivers=sample_me['driver']
131     races=sample_me['race']
132     year=sample_me['year']
133     inliers=pd.DataFrame()
134     outliers=pd.DataFrame()
135     for i in range(len(drivers)):
136         print(drivers[i])
137         ins,outs,strat=sample_race(drivers[i],races[i], year[i])
138
            ↪  ins=ins.assign(Driver=drivers[i],race=races[i],year=year[i])
139
            ↪  outs=outs.assign(Driver=drivers[i],race=races[i],year=year[i])
140         inliers=inliers.append(ins,ignore_index=True)
141         outliers=outliers.append(outs,ignore_index=True)
142
143     return inliers, outliers
```

A.3 Modelling

These functions could be used to model the data returned by the sampler script.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import lmfit
#function that generates individual vectors for tyre choices
def tyre_flags(tyres):
    f_soft=[]
    f_supersoft=[]
    f_ultrasoft=[]
    f_medium=[]
    f_hard=[]
    for tyre in tyres:
        if tyre =='Soft':
            f_soft.append(1)
        else:
            f_soft.append(0)
        if tyre =='Super soft':
            f_supersoft.append(1)
        else:
            f_supersoft.append(0)
        if tyre =='Ultra soft':
            f_ultrasoft.append(1)
        else:
            f_ultrasoft.append(0)
        if tyre =='Medium':
            f_medium.append(1)
        else:
            f_medium.append(0)
        if tyre =='Hard':
            f_hard.append(1)
        else:
            f_hard.append(0)
    return f_soft, f_supersoft, f_ultrasoft, f_medium, f_hard

def tyre_model(stintlap, lap, f_soft, f_supersoft, f_ultrasoft,
↪  f_medium, f_hard, F, Ts, Ts2, Tsb, Tss, Tss2, Tssb, Tus,
↪  Tus2, Tusb, Tm, Tm2, Tmb, Th, Th2, Thb, t_base, max_lap):
#functions for different tyre choices multiplied by 1 if tyre
↪  used, 0 if tyre not used.
    tsoft=(Ts*stintlap+Ts2*(stintlap**2)+Tsb)*f_soft
    ↪  tsupersoft=(Tss*stintlap+Tss2*(stintlap**2)+Tssb)*f_supersoft
```

```
38
        ↪ tultrasoft=(Tus*stintlap+Tus2*(stintlap**2)+Tusb)*f_ultrasoft
39      tmedium=(Tm*stintlap+Tm2*(stintlap**2)+Tmb)*f_medium
40      thard=(Th*stintlap+Th2*(stintlap**2)+Thb)*f_hard
41
        ↪ return(t_base+tfuel+tsoft+tsupersoft+tultrasoft+tmedium+thard)
42
43  def model_data(dataframe,report=False):
44
45      f_soft, f_supersoft, f_ultrasoft ,f_medium, f_hard
        ↪ =tyre_flags(dataframe['tyre'])
46      max_lap=max(dataframe['lap'])
47      quickest=min(dataframe['time'])
48      fmodel=lmfit.Model(tyre_model,independent_vars=['stintlap',
        ↪ 'lap', 'f_soft', 'f_supersoft', 'f_ultrasoft',
        ↪ 'f_medium', 'f_hard'])
49
50      #make params in model and set max_lap param to be max lap in
        ↪ dataset
51      params = fmodel.make_params(F=0.02, Ts=0.0, Ts2=0, Tsb=0,
        ↪ Tss=0, Tss2=0, Tssb=0, Tus=0, Tus2=0, Tusb=0, Tm=0.03,
        ↪ Tm2=0, Tmb=0, Th=0, Th2=0, Thb=0, t_base=quickest,
        ↪ max_lap=max_lap)
52      params['max_lap'].vary=False
53      params['t_base'].max=quickest+2
54      params['t_base'].min=quickest-5
55
56      #setting some vars to be min of zero becaue physics
57      params['F'].min=0
58      params['Ts'].min=-1
59      params['Tss'].min=-1
60      params['Tus'].min=-1
61      params['Tm'].min=-1
62      params['Th'].min=-1
63
64      #need to make one t_base vary=false!!! otherwise the errors
        ↪ will fail, choose tyre with most number of stints. I
        ↪ think if its a tie then mode() just chooses the first
        ↪ option.
65      most_used=dataframe.tyre.mode().values[0]
66      if most_used=="Soft":
67          params['Tsb'].vary=False
68      if most_used=="Super soft":
69          params['Tssb'].vary=False
70      if most_used=="Ultra soft":
71          params['Tusb'].vary=False
```

```
72          if most_used=="Medium":
73              params['Tmb'].vary=False
74          if most_used=="Hard":
75              params['Thb'].vary=False
76
77          #enforce 2nd derivitive of model is positive.
78          params.add('dtsdl',0.1,vary=True,min=0)
79          params.add('dtssdl',0.1,vary=True,min=0)
80          params.add('dtusdl',0.1,vary=True,min=0)
81          params.add('dtmdl',0.1,vary=True,min=0)
82          params.add('dthdl',0.1,vary=True,min=0)
83          params['Ts2'].expr='dtsdl /2'
84          params['Tss2'].expr='dtssdl/2'
85          params['Tus2'].expr='dtusdl /2'
86          params['Tm2'].expr='dtmdl /2'
87          params['Th2'].expr='dthdl/2'
88          #for loop that determins if tyre is not used
89          tyres_used=dataframe.tyre.unique()
90          possible_tyres=['Soft','Super soft','Ultra
             ↪  soft','Medium','Hard']
91          redundant=[tyre for tyre in possible_tyres if tyre not in
             ↪  tyres_used]
92          #if tyre is redundant then turn off fitting parameters for
             ↪  that tyre
93          if 'Soft' in redundant:
94              params['Ts'].vary = False
95              params['Ts2'].vary = False
96              params['Tsb'].vary = False
97              params['Ts2'].set(expr='')
98              params['dtsdl'].vary = False
99          if 'Super soft' in redundant:
100             params['Tss'].vary = False
101             params['Tss2'].vary = False
102             params['Tss2'].set(expr='')
103             params['Tssb'].vary = False
104             params['dtssdl'].vary = False
105         if 'Ultra soft' in redundant:
106             params['Tus'].vary = False
107             params['Tus2'].vary = False
108             params['Tus2'].set(expr='')
109             params['Tusb'].vary = False
110             params['dtusdl'].vary = False
111         if 'Medium' in redundant:
112             params['Tm'].vary = False
113             params['Tm2'].vary = False
114             params['Tm2'].set(expr='')
```

```
            params['Tmb'].vary = False
            params['dtmdl'].vary = False
        if 'Hard' in redundant:
            params['Th'].vary = False
            params['Th2'].vary = False
            params['Th2'].set(expr='')
            params['Thb'].vary = False
            params['dthdl'].vary = False

        times=dataframe['time'].to_numpy()
        stintlaps=dataframe['stint lap'].to_numpy()
        laps=dataframe['lap'].to_numpy()
        #fit data
        fit=fmodel.fit(times, params,  stintlap=stintlaps, lap=laps,
        ↪  f_soft=f_soft, f_supersoft=f_supersoft,
        ↪  f_ultrasoft=f_ultrasoft, f_medium=f_medium,
        ↪  f_hard=f_hard, method='least_squares')

        param=[]
        param_value=[]
        param_stderr=[]
        for key in fit.params:
            param.append(key)
            param_value.append(fit.params[key].value)
            param_stderr.append(fit.params[key].stderr)
        paramdf=pd.DataFrame(data=np.array(param_value).reshape(-1,
        ↪  len(param_value)), columns=param)

        ↪  param_stderrdf=pd.DataFrame(data=np.array(param_stderr).reshape(-1,
        ↪  len(param_stderr)), columns=param)
        if report==True:
            print(fit.fit_report())
        return paramdf,param_stderrdf

#%% script for moddelling multiple drivers at once
expdir='C:/Users/jxnny/OneDrive/Documents/school/Leeds/Dissertation/experiments/'
#file that contains driverref, race, year for the data you want
↪  to fit
expfile='catalunya_2012_will_merc'
csvtag='.csv'
datainall,dataoutall=sf(expdir+expfile+csvtag)
#%% ## getting parameters for each driver

##group data by driver
datain_d_groups=datainall.groupby('Driver')

```

```
#empty df for parameters
parameter_df = pd.DataFrame()
parameter_stderr_df=pd.DataFrame()
adjusted_data=pd.DataFrame()
#loop through all drivers in group object, save driver, race,
 ↪  year ,parameter data
#minus the bias for each driver from their lap times save to new
 ↪  df
for driver, driver_data in datain_d_groups:
    try:
        Driver=driver
        year=driver_data.iloc[0].year
        race=driver_data.iloc[0].race
        driver_param, driver_param_stderr=
         ↪  model_data(driver_data)
        driver_param= driver_param.assign(Driver=Driver,
         ↪  year=year, race=race)
        driver_param_stderr=
         ↪  driver_param_stderr.assign(Driver=Driver, year=year,
         ↪  race=race)
        #driver_adjusted= driver_data.assign(time=
         ↪  driver_data.time-driver_param.iloc[0].t_base)
        driver_adjusted= driver_data.assign(time=
         ↪  driver_data.time-driver_data.time.mean())
        parameter_df= parameter_df.append(driver_param,
         ↪  ignore_index = True)
        parameter_stderr_df=
         ↪  parameter_stderr_df.append(driver_param_stderr,
         ↪  ignore_index = True)
        adjusted_data= adjusted_data.append(driver_adjusted,
         ↪  ignore_index= True)
    except:
        pass

parameter_df.to_csv(expdir+ 'parameters_{}.csv'.format(expfile),
 ↪  index=False)

driver_all_param, driver_all_param_stderr=
 ↪  model_data(adjusted_data, report=True)
driver_all_param.to_csv(expdir+
 ↪  'parameters_{}.csv'.format(expfile), mode='a', index=False)
driver_all_param_stderr.to_csv(expdir+
 ↪  'parameters_{}.csv'.format(expfile), mode='a', index=False)
```

A.3.1 Composite model.

This script was used to make a composite model in `lmfit` which I could then use to make plots such as figure 3.7c. It makes use of the **tyre_flags** function defined above.

```
import os
wdir='C:/Users/jxnny/OneDrive/Documents/school/Leeds/Dissertation'
os.chdir(wdir)
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
#sampler function
from Sample_Race import sample_race as sr
import lmfit

def fuel_component(lap,F,t_base,max_lap):
    return(F*(max_lap-lap)+t_base)

def tyre_component(stintlap ,lap, f_soft, f_supersoft,
↪  f_ultrasoft, f_medium, f_hard, Ts, Ts2, Tsb, Tss, Tss2, Tssb,
↪  Tus, Tus2, Tusb, Tm, Tm2, Tmb, Th, Th2, Thb):
    tsoft=(Ts*stintlap+Ts2*(stintlap**2)+Tsb)*f_soft
    tsupersoft=(Tss*stintlap+Tss2*(stintlap**2)+Tssb)*f_supersoft
    tultrasoft=(Tus*stintlap+Tus2*(stintlap**2)+Tusb)*f_ultrasoft
    tmedium=(Tm*stintlap+Tm2*(stintlap**2)+Tmb)*f_medium
    thard=(Th*stintlap+Th2*(stintlap**2)+Thb)*f_hard
    return (tsoft+tsupersoft+tultrasoft+tmedium+thard)

#composite model by adding fuel component and tyre component
compmodel=lmfit.Model(tyre_component,independent_vars=['stintlap',
↪  'f_soft', 'f_supersoft', 'f_ultrasoft', 'f_medium', 'f_hard',
↪  'lap'])+lmfit.Model(fuel_component, independent_vars=['lap'])
# print('parameter names: {}'.format(compmodel.param_names))
# print('independent variables:
↪  {}'.format(compmodel.independent_vars))

#data
driver='bottas'
circuit='catalunya'
year=2015
tyre_data=True
clean_data=True
#sample the database and get the times and outliers
datain, dataout, used_strat=sr(driver, circuit, year,
↪  clean=clean_data, tyres=tyre_data)
```

```
35
36  #find some parameters for model
37  max_lap=max(datain['lap'])
38  quickest=min(datain['time'])
39  #tyreflags
40  f_soft, f_supersoft, f_ultrasoft, f_medium, f_hard=
    ↪  tyre_flags(datain['tyre'])
41  #initialise parameters
42
43  params = compmodel.make_params(F=0.02, Ts=0.0, Ts2=0, Tsb=0,
    ↪  Tss=0, Tss2=0, Tssb=0, Tus=0, Tus2=0, Tusb=0, Tm=0, Tm2=0,
    ↪  Tmb=0, Th=0, Th2=0, Thb=0, t_base=quickest,max_lap=max_lap)
44
45  most_used=datain.tyre.mode().values[0]
46  most_used=dataout[dataout['stint lap']==1].tyre.mode().values[0]
47  if most_used=="Soft":
48      params['Tsb'].vary=False
49  if most_used=="Super soft":
50      params['Tssb'].vary=False
51  if most_used=="Ultra soft":
52      params['Tusb'].vary=False
53  if most_used=="Medium":
54      params['Tmb'].vary=False
55  if most_used=="Hard":
56      params['Thb'].vary=False
57
58
59  params['max_lap'].vary=False
60  params['t_base'].max=quickest+2
61  params['t_base'].min=quickest-5
62
63  #setting some vars to be min of zero
64  params['F'].min=0
65  params['Ts'].min=-1
66  params['Tss'].min=-1
67  params['Tus'].min=-1
68  params['Tm'].min=-1
69  params['Th'].min=-1
70
71  params.add('dtsdl',0.1,vary=True,min=0)
72  params.add('dtssdl',0.1,vary=True,min=0)
73  params.add('dtusdl',0.1,vary=True,min=0)
74  params.add('dtmdl',0.1,vary=True,min=0)
75  params.add('dthdl',0.1,vary=True,min=0)
76  #restrict T2 to be above 0
77  params['Ts2'].expr='dtsdl /2'
```

```
78  params['Tss2'].expr='dtssdl/2'
79  params['Tus2'].expr='dtusdl /2'
80  params['Tm2'].expr='dtmdl /2'
81  params['Th2'].expr='dthdl/2'
82
83
84  #not all tyres will be used in a race so I need to be able to
    ↪  turn off fitting on the useless parameters
85  #list of tyres in the dataset
86  tyres_used=datain.tyre.unique()
87
88  possible_tyres=['Soft', 'Super soft', 'Ultra soft', 'Medium',
    ↪  'Hard']
89  #for loop that determins if tyre is not used
90  redundant=[tyre for tyre in possible_tyres if tyre not in
    ↪  tyres_used]
91
92  if 'Soft' in redundant:
93      params['Ts'].vary = False
94      params['Ts2'].vary = False
95      params['Tsb'].vary = False
96      params['Ts2'].set(expr='')
97      params['dtsdl'].vary = False
98  if 'Super soft' in redundant:
99      params['Tss'].vary = False
100     params['Tss2'].vary = False
101     params['Tss2'].set(expr='')
102     params['Tssb'].vary = False
103     params['dtssdl'].vary = False
104 if 'Ultra soft' in redundant:
105     params['Tus'].vary = False
106     params['Tus2'].vary = False
107     params['Tus2'].set(expr='')
108     params['Tusb'].vary = False
109     params['dtusdl'].vary = False
110 if 'Medium' in redundant:
111     params['Tm'].vary = False
112     params['Tm2'].vary = False
113     params['Tm2'].set(expr='')
114     params['Tmb'].vary = False
115     params['dtmdl'].vary = False
116 if 'Hard' in redundant:
117     params['Th'].vary = False
118     params['Th2'].vary = False
119     params['Th2'].set(expr='')
120     params['Thb'].vary = False
```

```
121     params['dthdl'].vary = False
122 
123 times=datain['time'].to_numpy()
124 stintlaps=datain['stint lap'].to_numpy()
125 laps=datain['lap'].to_numpy()
126 #fit data
127 compfit=compmodel.fit(times, params,  stintlap=stintlaps,
    ↪  lap=laps, f_soft=f_soft, f_supersoft=f_supersoft,
    ↪  f_ultrasoft=f_ultrasoft, f_medium=f_medium, f_hard=f_hard,
    ↪  method='least_squares')
128 
129 param=[]
130 param_value=[]
131 for key in compfit.params:
132     param.append(key)
133     param_value.append(compfit.params[key].value)
134 paramdf= pd.DataFrame(data= np.array(param_value).reshape(-1,
    ↪  len(param_value)), columns=param)
135 #reoder paramlist
136 col=['F', 'Ts', 'Ts2' ,'Tsb' ,'Tss' ,'Tss2' ,'Tssb' ,'Tus'
    ↪  ,'Tus2' ,'Tusb' ,'Tm' ,'Tm2' ,'Tmb' ,'Th' ,'Th2' ,'Thb'
    ↪  ,'t_base']
137 paramdf=paramdf[col]
138 #happily it gives the same results as non-composite model
139 
140 comps = compfit.eval_components()
141 #can then call component data from comps object
```

A.4 Optimizer

This is the script used to find the optimal strategies of the drivers. Lines 17 to 22 is how I would interact with the script. The data used and the parameters for the optimisation would come from the 'modeler3' script, which is shown in appendix A.3.1.

```
1   import os
2 
    ↪  wdir='C:/Users/jxnny/OneDrive/Documents/school/Leeds/Dissertation'
3   os.chdir(wdir)
4   import pandas as pd
5   import numpy as np
6   import matplotlib.pyplot as plt
7   import itertools
8   import lap_models
```

```
#modeler3 is script for composite model, used it to automatically get model parameters
import modeler3
from tqdm import tqdm
photodir='C:/Users/jxnny/OneDrive/Documents/school/Leeds/Dissertation/latex/'
datadir='C:/Users/jxnny/OneDrive/Documents/school/Leeds/Dissertation/data'
pittimes= pd.read_csv(datadir+'/pittimes.csv', index_col='circuitRef')

#uses same data as modelled in composite code script,see line 28-34 in the composite code script. This is mainly out of convienience, could easily make this script standalone and retrieve the parameters using model_data function defined earlier.
data=modeler3.datain.append(modeler3.dataout)
N_laps=max(data['lap'])]
#min-max number of stops you want optimiser to look for
Min_stop=1
Max_stop=3
#tyres available to driver, make sure you have enough for the number of stops your looking for!
soft=4
super_soft=4
ultra_soft=0
medium=0
hard=0
#first tyre
first_stint=None
#first stop lap
first_stop=None
savefig=False
#define parameters used for optimiser here.
model_params=modeler3.paramdf.iloc[0].values[0:17]

def tyre_options_maker(soft=3, supersoft=3, ultrasoft=3, medium=3, hard=3):
    s = np.array(['Soft' for _ in range(soft)])
    ss = np.array(['Super soft' for _ in range(supersoft)])
    us = np.array(['Ultra soft' for _ in range(ultrasoft)])
    m = np.array(['Medium' for _ in range(medium)])
    h = np.array(['Hard' for _ in range(hard)])
    tyre_list = [y for x in [s,ss,us,m,h] for y in x]
    return tyre_list

```

```
45      tyre_options=tyre_options_maker(soft=soft,
        ↪  supersoft=super_soft, ultrasoft=ultra_soft, medium=medium,
        ↪  hard=hard)
46
47      def
        ↪  list_options(N_lap,N_stop,tyre_options=tyre_options,**kwargs):
48          first_stint=kwargs.get('first_stint',None)
49          first_stop=kwargs.get('first_stop',None)
50          lap_list=np.linspace(1,N_lap,num=N_lap,dtype=int)
51          stop_list=[i for i in itertools.combinations(lap_list,
            ↪  r=N_stop)]
52          tyre_combos= [i for i in
            ↪  itertools.permutations(tyre_options, r=N_stop+1)]
53          tyre_combos=set(tyre_combos)
54          #code that gets rid of tyre combos with the same tyre in
            ↪  every stint, as thats not allowed
55          #also if option of first stint is not none then it will get
            ↪  rid of all strategies where
56          #the first stint tyre is not first_stint. This is so if you
            ↪  got to q3 the strategy engineer can set the first tyre.
57          dummycombos=[]
58          if first_stint==None:
59              for combo in tyre_combos:
60                  if combo.count(combo[0])!=len(combo):
61                      dummycombos.append(combo)
62          else:
63              for combo in tyre_combos:
64                  if combo.count(combo[0])!=len(combo) and
                    ↪  combo[0]==first_stint:
65                      dummycombos.append(combo)
66          tyre_combos=dummycombos
67          #meld  all possible stop and tyre combos
68          potential_strategies=[list(i) for i in
            ↪  itertools.product(stop_list, tyre_combos)]
69          #just if your setting the first stop lap, only keep
            ↪  strategies that have correct first stop lap
70          if first_stop!=None:
71              dummystrat=[]
72              for strat in potential_strategies:
73                  if strat[0][0]==first_stop:
74                      dummystrat.append(strat)
75              potential_strategies=dummystrat
76
77          print("{} potential strategies
            ↪  found".format(len(potential_strategies)))
78          return (potential_strategies)
```

```
79
80      def tyre_flags(tyre):
81
82          if tyre =='Soft':
83              f_soft=1
84          else:
85              f_soft=0
86          if tyre =='Super soft':
87              f_supersoft=1
88          else:
89              f_supersoft=0
90          if tyre =='Ultra soft':
91              f_ultrasoft=1
92          else:
93              f_ultrasoft=0
94          if tyre =='Medium':
95              f_medium=1
96          else:
97              f_medium=0
98          if tyre =='Hard':
99              f_hard=1
100         else:
101             f_hard=0
102         return f_soft,f_supersoft,f_ultrasoft,f_medium,f_hard
103
104     def stint_time_integrator(start,stop,params,tyre,max_lap):
105         F, Ts, Ts2, Tsb, Tss, Tss2, Tssb, Tus, Tus2, Tusb, Tm, Tm2,
            ↪  Tmb, Th, Th2, Thb, t_base=params
106         f_soft, f_supersoft, f_ultrasoft, f_medium, f_hard=
            ↪  tyre_flags(tyre)
107         delta=stop-start
108         fuel=(F*max_lap*delta) - F*(0.5*delta*(start+stop+1)) +
            ↪  t_base*(delta)
109         soft=(((delta+1)*delta/2)*Ts +
            ↪  ((2*delta+1)*(delta+1)*delta/6)*Ts2 + Tsb*delta)*f_soft
110         supersoft=(((delta+1)*delta/2)*Tss +
            ↪  ((2*delta+1)*(delta+1)*delta/6)*Tss2 +
            ↪  Tssb*delta)*f_supersoft
111         ultrasoft=(((delta+1)*delta/2)*Tus +
            ↪  ((2*delta+1)*(delta+1)*delta/6)*Tus2 +
            ↪  Tusb*delta)*f_ultrasoft
112         medium=(((delta+1)*delta/2)*Tm +
            ↪  ((2*delta+1)*(delta+1)*delta/6)*Tm2 +
            ↪  Tmb*delta)*f_medium
113         hard=(((delta+1)*delta/2)*Th +
            ↪  ((2*delta+1)*(delta+1)*delta/6)*Th2 + Thb*delta)*f_hard
```

```
        time=fuel+soft+supersoft+ultrasoft+medium+hard
        return time

    def strategy_to_laps(stops,tyres,N_lap):
        lap_df=pd.DataFrame()
        lap_list=np.linspace(1,N_lap,num=N_lap,dtype=int)
        stops=np.asarray(stops)
        stintlap=[]
        tyre=[]
        lapofstint=0
        for lap in lap_list:
            stintnum=1
            lapofstint+=1
            for stoplap in stops:
                if lap > stoplap:
                    stintnum+=1
            if lap in stops+1:
                lapofstint=1
            stintlap.append(lapofstint)
            tyreused=tyres[stintnum-1]
            tyre.append(tyreused)
        lap_df['lap']=lap_list
        lap_df['stintlap']=stintlap
        lap_df['tyre']=tyre
        return lap_df

    def
     ↪  laps_to_time(strategy,model,model_params,plot=False,**kwargs):
        label=kwargs.get('label','best strategy model')
        nstop=len(strategy[strategy['stintlap']==1])-1
        total_time=0
        times=[]
        laps=strategy['lap']
        stintlaps=strategy['stintlap']
        tyres=strategy['tyre']
        for lap in range(len(laps)):
            time=model(laps[lap], stintlaps[lap], tyres[lap],
             ↪  model_params, len(strategy))
            total_time+=time
            times.append(time)
        total_time+=nstop*pittimes.loc[modeler3.circuit].values[0]
         ↪  #median pit stop time
        if plot==True:
            plt.scatter(laps.values,times,label=label)
        mintime=min(times)
        return total_time,mintime
```

```
157
158     def strategy_to_time(strategy,model_params,max_lap=N_laps):
159         total_time=0
160         nstop=len(strategy[0])
161         tyres=strategy[1]
162         #starting laps and end laps
163         stint_starts=[]
164         stint_starts.append(0)
165         for stoplap in strategy[0]:
166             stint_starts.append(stoplap)
167         stint_ends=list(strategy[0])
168         stint_ends.append(max_lap)
169         #integrate over each stint and add up the times
170         for i in range(len(stint_starts)):
171            add_time=stint_time_integrator(stint_starts[i],
               ↪  stint_ends[i], model_params, tyres[i],
               ↪  max_lap=N_laps)
172            #print(add_time)
173            total_time+=add_time
174         #add up the pit stop penalties
175         total_time+=nstop*pittimes.loc[modeler3.circuit].values[0]
176         return total_time
177
178     def optimize_strategy(N_lap, min_stop, max_stop, model,
        ↪  model_params, tyre_options, **kwargs):
179         first_stint=kwargs.get('first_stint',None)
180         first_stop=kwargs.get('first_stop',None)
181         close_cut=kwargs.get('close_cut',20)
182         best_time=1000000
183         close_times=[]
184         close_strategy=[]
185         for n_stop in range(min_stop,max_stop+1):
186             print('exploring {} stop strategy'.format(n_stop))
187             strategy_list=list_options(N_lap=N_lap, N_stop=n_stop,
                ↪  first_stint=first_stint, first_stop=first_stop)
188
189             for strategy in
                ↪  tqdm(strategy_list,position=0,leave=True):
190                 model_time=strategy_to_time(strategy,
                    ↪  model_params,N_lap)
191                 if model_time < best_time:
192                     best_time=model_time
193                     best_strategy=strategy
194                 elif model_time < best_time+close_cut:
195                     close_times.append(model_time)
196                     close_strategy.append(strategy)
```

```
197         close_times=pd.DataFrame(list(zip(close_times,
            ↪  close_strategy)),columns=['time','strategy'])
198         close_times=close_times[close_times['time'] <
            ↪  best_time+close_cut]
199         close_times=close_times.sort_values(by=['time'])
200         #laps_to_time(strategy_to_laps(best_strategy[0],
            ↪  best_strategy[1], N_lap=N_lap), model,
            ↪  model_params,plot=True)
201         return best_strategy, best_time, close_times
202     #%%
203     best_strat, best_time, close_times=
        ↪  optimize_strategy(N_lap=N_laps, min_stop=Min_stop,
        ↪  max_stop=Max_stop, model=lap_models.tyre_model,
        ↪  model_params=model_params, tyre_options=tyre_options,
        ↪  first_stint=first_stint, first_stop=first_stop,
        ↪  close_cut=10)
204     print("quickest_strategy {}".format(best_strat))
205
206     #%%
207     if len(close_times)>0:
208         fastest_lap=10000
209         print("found {} close times, checking for fastest
            ↪  lap".format(len(close_times)))
210         for strat in close_times.strategy:
211             totalt,
                ↪  sfastest_lap=laps_to_time(strategy_to_laps(strat[0],
                ↪  strat[1], N_lap=N_laps),
                ↪  model=lap_models.tyre_model,
                ↪  model_params=model_params, plot=False)
212             if sfastest_lap<fastest_lap:
213                 fastest_lap=sfastest_lap
214                 quick_lap_strat=strat
215
216         print("quickest_lap_strategy {}".format(quick_lap_strat))
217         print("quick_lap: {}".format(fastest_lap))
218     #%%
219     title='{}, {}, {}'.format(modeler3.driver.capitalize(),
        ↪  modeler3.circuit.capitalize(), modeler3.year)
220     fig,ax = plt.subplots()
221         #for loop adds to the figure by looping over each stint
222     for k,d in modeler3.datain.groupby(['tyre']):
223         ax.scatter(d['lap'], d['time'], label='lap times, {}
            ↪  tyre'.format(k))
```

```
224     laps_to_time(strategy_to_laps(best_strat[0], best_strat[1],
        ↪  N_lap=N_laps), model=lap_models.tyre_model,
        ↪  model_params=model_params, plot=True, label="Modelled
        ↪  strategy")
225     title1='Best Modelled Strategy \n{}, {},
        ↪  {}'.format(modeler3.driver.capitalize(),
        ↪  modeler3.circuit.capitalize(), modeler3.year)
226     #title1='{}, {}, {} \n Lap 17 Best Two
        ↪  Stop'.format(modeler3.driver.capitalize(),
        ↪  modeler3.circuit.capitalize(), modeler3.year)
227     plt.legend()
228     plt.xlabel('Lap')
229     plt.ylabel('Lap-time(s)')
230     plt.title(title1)
231     plt.show()
232
233     if len(close_times)>0:
234         fig2,ax2 = plt.subplots()
235             #for loop adds to the figure by looping over each stint
236         for k,d in modeler3.datain.groupby(['tyre']):
237             ax2.scatter(d['lap'], d['time'], label='lap times, {}
                ↪  tyre'.format(k))
238         laps_to_time(strategy_to_laps(quick_lap_strat[0],
            ↪  quick_lap_strat[1], N_lap=N_laps),
            ↪  model=lap_models.tyre_model,
            ↪  model_params=model_params,plot=True,label="strategy
            ↪  with quick lap")
239         title2='Quick-lap Strategy\n{}, {},
            ↪  {}'.format(modeler3.driver.capitalize(),modeler3.circuit.capitalize(),mo
240         plt.legend()
241         plt.xlabel('Lap')
242         plt.ylabel('Lap-time(s)')
243         plt.title(title2)
244     #%%
245     # comparing optimal strategy to the actual performance
246
        ↪  actual_race_time=modeler3.datain.time.sum()+modeler3.dataout.time.sum()
247     pred_time=strategy_to_time(modeler3.used_strat ,
        ↪  model_params=model_params,max_lap=N_laps)
248     if len(close_times)>0:
249         quick_lap_time=strategy_to_time(quick_lap_strat ,
            ↪  model_params=model_params,max_lap=N_laps)
250         print('optimised strategy predicted time : {} \n actual
            ↪  race time: {} \n predicted time of actual strategy: {}
            ↪  \n time of strategy with quick
            ↪  lap:{}'.format(best_time,actual_race_time,pred_time,quick_lap_time))
```

```
    else:
        quick_lap_time=0
        print('optimised strategy predicted time : {} \n actual
        ↪  race time: {} \n predicted time of actual strategy:
        ↪  {}'.format(best_time,actual_race_time,pred_time))
    print("difference: {}".format(pred_time-best_time))
```

A.5 Refuelling-optimiser

The implementation of the strategy optimiser under the refuelling scenario was the same as in appendix section A.4, except that the stint and lap times were calculated using the *strategy_to_time_ref* and *laps_to_time_ref* functions.

```
def pit_pen_calculator(strategy, max_lap=N_laps,
↪  track='catalunya'):
    pit_pen=0
    #starting laps and end laps
    stint_starts=[]
    stint_starts.append(0)
    for stoplap in strategy[0]:
        stint_starts.append(stoplap)
    stint_ends=list(strategy[0])
    stint_ends.append(max_lap)
    refuel_time=pittimes.loc[track].values[2]
    av_pit=2.92
    base_time=pittimes.loc[track].values[0]-av_pit
    for i in range(1,len(stint_starts)):
        delta=stint_ends[i]-stint_starts[i]
        if delta*refuel_time>av_pit:
            pit_time=base_time+delta*refuel_time
            pit_pen+=pit_time
        else:
            pit_time=base_time+av_pit
            pit_pen+=pit_time
    return pit_pen

def strategy_to_time_ref(strategy,model_params,max_lap=N_laps):
    total_time=0
    tyres=strategy[1]
    #starting laps and end laps
    stint_starts=[]
    stint_starts.append(0)

    for stoplap in strategy[0]:
        stint_starts.append(stoplap)
```

```
    stint_ends=list(strategy[0])
    stint_ends.append(max_lap)
    #integrate over each stint and add up the times
    deltas=[stint_ends[i]-stint_starts[i] for i in range(len(stint_starts))]
    stint_starts2=[0 for i in range(len(stint_starts))]

    for i in range(len(stint_starts)):
        add_time=stint_time_integrator(stint_starts2[i], deltas[i], model_params, tyres[i], max_lap=deltas[i])
        #print(add_time)
        total_time+=add_time
    #add up the pit stop penalties
    total_time+= pit_pen_calculator(strategy, max_lap=max_lap, track=track)
    return total_time



def laps_to_time_ref(strategy, model, model_params, strat, plot=False, **kwargs):
    label=kwargs.get('label', 'best strategy model')
    shape=kwargs.get('shape','o')
    size=kwargs.get('size',20)
    color=kwargs.get('color',None)
    total_time=0
    times=[]
    laps=strategy['lap']
    stintlaps=strategy['stintlap']
    max_laps=strategy['stint_max_laps']
    tyres=strategy['tyre']
    for lap in range(len(laps)):
        time=model(stintlaps[lap], stintlaps[lap], tyres[lap], model_params,max_laps[lap])
        total_time+=time
        times.append(time)
    total_time+=pit_pen_calculator(strat, max_lap=N_laps, track=track) # refuelling pit stop time time
    if plot==True:
        if shape=='line':
            plt.plot(laps.values,times,label=label)
        elif color!=None:
            plt.scatter(laps.values, times, marker=shape, color=color, s=size, label=label)
        else:
```

```
70              plt.scatter(laps.values, times, marker=shape,
                ↪  label=label)
71      mintime=min(times)
72      return total_time,mintime
```