

University of Brighton

Department of Computer Science

Telemetry Visualisation and Statistical Analysis System for Motorsport

Max Sherman

Supervisor: Dr. Almas Baimagambetov

Second reader: Dr. Anestis Touloumis

A report submitted in partial fulfilment of the requirements of
the University of Brighton for the degree of
Bachelor of Science in *Computer Science*

02/05/2025

Abstract

The development of data-driven tools in motorsport, particularly in Formula 1 (F1), is critical for enhancing performance and decision-making. This project aims to create a telemetry visualisation and statistical analysis tool, designed to provide actionable insights from complex F1 data. The tool leverages Python-based libraries, including Pandas, Matplotlib, Plotly, SciPy, and Statsmodels, to enable users to explore key metrics such as lap times, driver performance, and race strategies. The methodology covers requirement gathering, literature review, and software design to ensure a robust and effective solution. Results will demonstrate the capability of the tool to present data interactively and perform meaningful statistical analyses, such as regression and time-series analysis. These techniques will enable performance evaluation and trend identification. As a result, F1 data analysis will become more accessible for fans, analysts, and amateur motorsport teams. The scalable architecture promotes incremental development, such as refining the application through user feedback, integrating advanced statistical features, and enhancing usability to maximise the tool's impact in the field of motorsport data analysis.

Keywords: Python, Data Analysis, Data Visualisation, Statistical Analysis, Formula 1

Report's total word count: 10,670 words

Contents

1 Introduction **1**
- 1.1 Problem statement . . . 1
- 1.2 Solution approach . . . 1
  - 1.2.1 Data visualisation . . . 1
  - 1.2.2 Statistical analysis . . . 1
- 1.3 Project scope . . . 2

2 Aims and Objectives **3**
- 2.1 Project aim . . . 3
- 2.2 Project objectives . . . 3

3 Research **4**
- 3.1 Literature review . . . 4
  - 3.1.1 Data-driven analysis in motorsport . . . 5
  - 3.1.2 Data visualization techniques . . . 5
  - 3.1.3 Statistical analysis methods . . . 5
- 3.2 Competitor analysis . . . 6

4 Planning **10**
- 4.1 Schedule of activities . . . 10
- 4.2 Risk analysis . . . 11

5 Requirements **13**
- 5.1 Functional requirements . . . 13
- 5.2 Non-functional requirements . . . 15
  - 5.2.1 Traceability matrix . . . 16
- 5.3 Requirement determination process . . . 17

6 Methodology **18**
- 6.1 Project management methodology . . . 18
- 6.2 Development tools and techniques . . . 18
  - 6.2.1 Programming language . . . 19
  - 6.2.2 Development environment . . . 19
  - 6.2.3 Version control . . . 19
  - 6.2.4 API selection and data integration . . . 20
  - 6.2.5 Visualisation libraries . . . 20
  - 6.2.6 Statistical analysis libraries . . . 20
  - 6.2.7 GUI framework . . . 21
- 6.3 Evaluation and testing tools and techniques . . . 21

7 Product description **23**
7.1 Design . . . 23
7.1.1 System architecture . . . 23
7.1.2 GUI layout sketch . . . 25
7.2 Development and Implementation . . . 26
7.2.1 Environment setup and configuration . . . 26
7.2.2 Key implementation phases . . . 26
7.2.3 Code structure and core modules . . . 27
7.2.4 Challenges faced and solutions implemented . . . 28
7.2.5 Key features and functionalities . . . 34
7.3 Testing and evaluation . . . 40
7.3.1 In-line testing . . . 40
7.3.2 Unit testing . . . 40
7.3.3 Bugs and known issues . . . 41

8 Critical Review **42**
8.1 Evaluation of project success . . . 42
8.2 Lessons leaned . . . 42
8.3 Future improvements . . . 43
8.4 Limitations . . . 44

9 Conclusion **45**
9.1 Evaluation of project outcomes . . . 45
9.2 Overall project reflection . . . 45
9.3 Final remarks . . . 46

Appendices **51**

A Record of engagement **51**

B The ethics checklist **69**

C The Gantt chart **70**

D Functional and non-functional requirements **75**

E GUI sketches **80**

F requirements.txt **85**

G Source code **86**
G.1 `jolpica_api.py` . . . 86
G.2 `data_preprocessing.py` . . . 91
G.3 `plot_generator.py` . . . 94
G.4 `analysis_main.py` . . . 94
G.5 `main.py` . . . 96

H Test cases and results **97**
H.1 `test_api.py` . . . 97
H.2 `test_visualisation.py` . . . 107
H.3 `test_analysis.py` . . . 110

List of Figures

3.1 Screenshot of the F1 Tempo web application 6
3.2 Screenshot of the tracing insights lap chart 7
3.3 Screenshot of the F1 analysis 2024 Pre-Season Predictions article 8

7.1 The package diagram for the F1 data analysis tool 24
7.2 The sequence diagram for the F1 data analysis tool 25
7.3 Initial sketch of the GUI for the F1 data analysis tool 26
7.4 Mean calculation for pitstop duration 34
7.5 Paired T-test results for grid and finish position 34
7.6 The tool's GUI for data visualisation, showing the points per driver bar chart for the 2023 season 35
7.7 The tool's GUI for data analysis, showing the statistic and p-value of the Spearman correlation between qualifying and race position for the 2024 season 35
7.8 Points per constructor bar chart for the 2023 season 36
7.9 Race position per constructor box plot for the 2021 season 36
7.10 Pitstop duration per driver box plot for round 1 of the 2020 season 37
7.11 Fastest lap rank vs race position grouped by constructor scatter plot for round 1 of the 2021 season 37
7.12 Constructors nationality pie chart for the 2021 season 38
7.13 Position per lap grouped by driver line chart for round 1 of the 2020 season, static plot 38
7.14 Position per lap grouped by driver line chart for round 1 of the 2020 season, interactive plot 39
7.15 Standard Deviation calculation for pitstop duration 39
7.16 ANOVA test results for lap times and driver ID 39
7.17 Linear Regression predictions for lap times 39
7.18 ARIMA forecasting results for lap times 39

List of Tables

3.1 Comparison of existing Formula 1 data analysis tools and platforms 9

4.1 Development timeline and task breakdown 11
4.2 Risk Analysis and Mitigation Plan 12

5.1 Functional Requirements 15
5.2 Non-Functional Requirements 16
5.3 Traceability matrix: Objectives, Functional, and Non-functional requirements 16

7.1 Project architecture: Package modules and responsibilities 24
7.2 Core modules: Functions and responsibilities 28

8.1 Overview of future improvements, description, benefit, and future tasks . . . 44

Listings

7.1 get data, get all data, and get cleaned data functions in jolpica_api.py 28
7.2 extend inner data function in json_handler.py 29
7.3 plot chart function in plot_generator.py 30
7.4 update filters function in callbacks_data.py 32
7.5 update plot function in callbacks_plots.py 33
7.6 The convert to milliseconds function before and after bug fixing 41
G.1 jolpica_api.py 86
G.2 data_preprocessing.py 91
G.3 plot_generator.py 94
G.4 analysis_main.py 94
G.5 main.py 96
H.1 test_api.py 97
H.2 test_visualisation.py 107
H.3 test_analysis.py 110

List of Abbreviations

F1	Formula 1
API	Application Programming Interface
UI	User Interface
UX	User Experience
GUI	Graphical User Interface
IDE	Integrated Development Environment
SAS	Statistical Analysis Software
ATLAS	Advanced Telemetry Linked Acquisition System
JSON	JavaScript Object Notation
TDD	Test-Driven Development
UML	Unified Modeling Language
p-value	Probability Value
df	DataFrame
CSV	Comma-Separated Values
VCS	Version Control System

Chapter 1

Introduction

1.1 Problem statement

Despite Formula 1 (F1) teams using advanced data analytics to optimise performance and leverage data-driven decisions, there remains a lack of easily accessible tools for independent users, such as fans, analysts, and amateur teams. Existing solutions are either basic applications that are unable to provide insightful visualisations and analysis, or are only available to high-budget teams, such as the Advanced Telemetry Linked Acquisition System (ATLAS), which is used by all current F1 teams (McLaren Applied, 2024).

Similar open-source tools lack crucial functionalities, such as trend identification, performance metrics analysis, and interactive data visualisation. Furthermore, only a few have the features required to perform statistical analysis, such as regression, correlation, and time-series analysis, on session-specific data. The lack of accessible telemetry data analysis software prevents non-professional users from gaining meaningful insights or fully comprehending race dynamics and tactics. A fitting solution to this problem requires user-friendly data visualisation and powerful statistical analysis, allowing users to effectively explore and interpret F1 data.

1.2 Solution approach

To approach the problem described previously, this project focuses on developing a web-based application that intuitively combines visualisation and statistical analysis of F1 race data to provide comprehensible and actionable insights. The application will leverage the Jolpica-F1 API to retrieve session and season-specific data, whilst utilising Python-based libraries to implement the data analysis features.

1.2.1 Data visualisation

Data visualisation plays a critical role in data analysis, as it is responsible for converting raw data into user-comprehensible visualisations. To achieve effective and interactive data visualisation, the project uses Python libraries such as Matplotlib, Seaborn and Plotly. These libraries provide flexibility in creating both static and interactive visualisations.

1.2.2 Statistical analysis

Statistical analysis methods play a key role in this project. Suitable methods have been selected to aid trend, relationship, and pattern identification. Several suitable methods have

been applied to cover the project's objectives.

Descriptive statistics are used to summarise key metrics such as average lap times, finishing positions, and qualifying performance. These features provide a clear and global summary of driver and team performance.

Hypothesis testing supports the comparison of performance between drivers or constructors. For instance, the statistical significance of the difference between two drivers' performance can be determined by using hypothesis testing. Correlation analysis measures the degree to which variables are related, facilitating the identification and measurement of relationships such as the correlation between qualifying positions and race outcomes.

Regression analysis is used to model the relationship between a dependent variable and one or more independent variables. This allows users to forecast outcomes, such as race outcomes based on qualifying results, offering insights into the impact of specific variables. Time series analysis enables users to monitor performance trends throughout a race or season. Techniques such as moving averages help smooth the data and highlight the underlying patterns in lap-by-lap or race-by-race performance.

By combining these techniques, the solution will offer an intuitive and comprehensive tool that meets the project's aims and objectives (see chapter 2).

1.3 Project scope

This project focuses on the development of a user-friendly, web-based application for the retrieval, visualisation, and statistical analysis of F1 data. Key functionalities include API data retrieval and filtering, the generation and rendering of both static and interactive plots, and the implementation of descriptive, comparative, and trend-based statistical analysis techniques. These components are integrated into an interactive Graphical User Interface (GUI) built using the Dash framework. The report covers the full development cycle, including research, planning, design, implementation, and testing.

Certain aspects were deliberately excluded from the project scope to ensure feasibility within the given time frame. These include real-time data analysis, system deployment, and the use of advanced machine learning models. These features, whilst valuable, were considered beyond the scope of this project and are proposed as potential future enhancements.

Chapter 2

Aims and Objectives

2.1 Project aim

Aims: The main goal of this project is to develop a user-friendly web-based application that provides comprehensive tools for visualising and statistically analysing Formula 1 data. The tool will allow access to F1 insights for users such as fans, analysts, and amateur teams by utilising the Jolpica-F1 API. Furthermore, the tool will allow users to investigate advanced race dynamics, identify performance trends, and make data-driven conclusions. Consequently, the tool will bridge the gap between raw data and actionable insights, encouraging a deeper understanding of F1 and its data through effective data visualisation and suitable statistical methods.

2.2 Project objectives

1. Develop a dependable data retrieval system that retrieves session and season-specific race data by utilising the Jolpica-F1 API.
2. Implement a set of suitable data visualisation tools to provide race metrics in an intuitive and interactive format.
3. Apply statistical analysis techniques, such as descriptive statistics, regression analysis, and time-series analysis, to uncover patterns and relationships in race data.
4. Develop a user-friendly graphical user interface (GUI) for the web-based application, providing effortless navigation between data analysis and visualisation features.
5. Test and validate the tool to ensure effectiveness in visual representations and accuracy in statistical calculations.
6. Provide comprehensive documentation, including a user guide and technical report, detailing the system's functionality, implementation, and capabilities.

Chapter 3

Research

This chapter presents the research undertaken to inform and justify the design, implementation, and features of the F1 data analysis tool. The research covers academic literature and existing platforms, which collectively guided key design, integration, and implementation strategies.

The literature review establishes a foundation by examining the role of data in motorsport, with an emphasis on telemetry systems, data visualisations, and statistical modelling. These insights highlight the importance of tools that not only present data but also provide advanced analysis to support performance evaluation and decision making. In reviewing existing work, both academic and practical, the research identifies a gap in systems that combine accessible interfaces with statistically meaningful insights, helping define the contribution of this project.

The competitor analysis evaluates popular F1 data tools and platforms, highlighting their strengths and limitations in terms of accessibility, interactivity, and analytical capabilities. These findings directly informed the inclusion or exclusion of specific features and methods within the tool.

3.1 Literature review

Formula 1 (F1) is one of the world's most technologically advanced sports, driven by continuous data collection and analytics. The sport has historically relied on engineering expertise, strategic intuition, and rapid decision-making, but with the uprise of advanced telemetry systems, F1 now captures an extraordinary amount of real-time data from every race or session. Telemetry data is widely regarded as the most important source of information by motorsport engineers (Bugeja et al., 2017). F1 cars have over 300 sensors, producing 1.5 terabytes of data across the whole race weekend, all of which are analysed to inform everything from car design to racing strategy (Shapiro, 2023). This valuable data can then be leveraged to optimise the team's decision-making and overall performance.

A critical aspect of data analysis in motorsport is effective data visualisation. The integration of telemetry data visualisation tools enhances the ability to interpret data, converting raw data to actionable insights (Ahmad, 2023). Visualisation frameworks allow for the exploration of complex data by displaying the telemetry and race data in a more intuitive and insightful format (Frost et al., 2020). Effective visualisation supports both fans and analysts in identifying trends and understanding relationships between variables such as lap time, speed, tyre degradation, and circuits.

Furthermore, statistical analysis techniques support a deeper understanding of performance trends and comparative metrics. Through the analysis of the received data, it is possible to study a specific system behaviour, interact with it, and properly control its evolution

(Fontanella et al., 2019). Time-series data, which forms the majority of the telemetry data, requires specific methods for analysing and identifying patterns or performance trends. For instance, statistical models like linear regression and time-series forecasting are crucial to understanding how certain factors affect certain outcomes. The results can offer predictive insights, enabling data-driven decision-making.

3.1.1 Data-driven analysis in motorsport

Telemetry systems are essential in motorsport, enabling both real-time and retrospective analysis capabilities. Riedel (2022) discusses how telemetry has become integral to vehicle performance monitoring and regulation compliance. By collating data from multiple sensors, telemetry systems enable comprehensive analysis of vehicle behaviour and race dynamics (Fontanella et al., 2019). These systems serve as the foundation for data-driven decision-making, offering critical insights into various race metrics.

The Advanced Telemetry Linked Acquisition System (ATLAS) is currently used by all F1 teams and throughout the motorsport and automotive industry (McLaren Applied, 2024). The system's data processing capabilities, high performance, and monopoly in the field suggest that ATLAS is the industry benchmark for this project.

Shapiro (2023) emphasises how data analytics is revolutionising the motorsport industry by facilitating strategic decision making during races. Whilst this project focuses on post-race analysis, these insights justify the tool's emphasis on various data analysis methods.

Additionally, Bugeja et al. (2017) examines how telemetry systems can be used to optimise driver training in racing simulators. Telemetry data, when analysed effectively, can improve driver performance by identifying specific areas of improvement, reinforcing the value of statistical interpretation in identifying performance gaps. These findings support the inclusion of comparative analysis functions in the tool to help users evaluate drivers and teams across sessions and seasons.

3.1.2 Data visualization techniques

Understanding complex telemetry data requires effective data visualisation. Healy (2018) emphasises the importance of comprehensible and intuitive visual representations, whilst Wilke (2019) provides guidelines on determining suitable visualisation methods. They discuss best practices for graph design and data presentation, whilst providing a foundational understanding of conveying valuable data insights through visual means. These resources are essential for designing effective visualisations.

Fontanella et al. (2019) and Riedel (2022) both highlight the importance of effective and intuitive interfaces for exploring telemetry data. The project reflects these principles by offering both interactive and static plots via an accessible interface. The insights provided are crucial for developing a system that balances functionality with usability.

3.1.3 Statistical analysis methods

Statistical methods are critical for uncovering performance trends and conducting predictive analysis in motorsport. Cowan (1998), Bowman and Robinson (1987) provide an overview of hypothesis testing and regression analysis, which support performance comparisons across drivers, sessions, or teams. These methods support the project's comparative analysis features, enabling users to assess whether observed differences are statistically significant.

Montgomery et al. (2008) explains several time-series forecasting techniques, including ARIMA and Holt-Winters models, which are particularly relevant to lap time predictions.

These models are designed to identify underlying trends and seasonality within time-dependent data, allowing for more accurate forecasting. The ARIMA model is used to forecast points in a time series based on its past values. The Holt-Winters model, a form of exponential smoothing, is suited for data with seasonality, making it ideal for analysing performance containing regular fluctuations. Their implementation in this project supports the analysis of race dynamics and driver consistency, offering users predictive insights beyond descriptive and comparative statistics.

The models were selected due to their ability to handle time-dependent data, which is common in motorsport data. Alternative models, such as decision trees or clustering, were considered but deemed less suitable as they do not handle time-series data as efficiently. While these methods are well-established, they require careful parameter selection and are sensitive to data quality. For instance, ARIMA models are sensitive to overfitting, and their accuracy depends heavily on the proper selection of order parameters (p, d, q). Similarly, Holt-Winters requires careful tuning of smoothing parameters to ensure meaningful results. Their implementation in the tool was supported by data validation and cleaning, ensuring users receive reliable outputs even without statistical expertise.

3.2 Competitor analysis

This section examines existing tools and platforms used for Formula 1 data analysis and visualisation. The purpose of this analysis is to identify the strengths and limitations of current solutions (see table 3.1), and to highlight opportunities for improvement, directly influencing the development of this project. Particular attention is given to the usability and overall user experience of each tool. Specific features were either adopted, improved upon, or deliberately excluded from the system.

F1 Tempo is a web-based application that leverages the FastF1 data back-end to provide interactive lap time and telemetry visualisations for each session and driver (F1 Tempo, 2025). The platform excels at visualising core telemetry metrics such as speed, throttle, braking, RPM, gear, and tyre usage. However, the tool is limited in customisation and analysis, limiting the user to interactive line charts only (see figure 3.1). These constraints informed the decision to implement both interactive and static plots in this project, as well as to include comparative and trend-based statistical methods.



Figure 3.1: Screenshot of the F1 Tempo web application

TracingInsights.com offers one of the most comprehensive F1 data exploration tools currently available (see figure 3.2), with features including corner analysis, lap charts, G-force heatmaps, and tyre degradation models (Tracing Insights, 2025). However, the platform suffers from several usability and performance issues. In contrast, this project emphasises usability and responsiveness by providing an intuitive layout, data export capabilities, and support for plot customisation and analysis configuration.

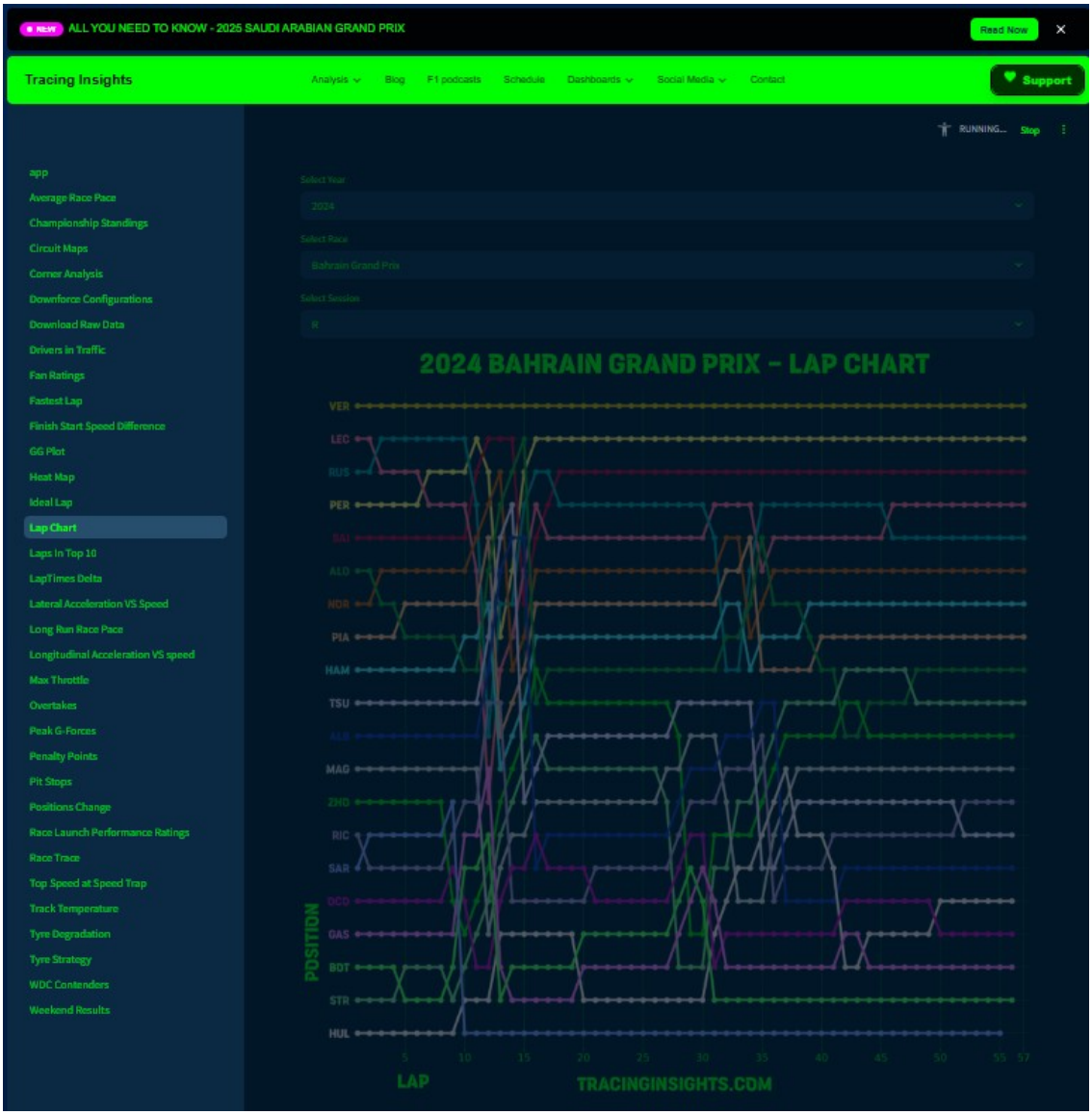


Figure 3.2: Screenshot of the tracing insights lap chart

F1-Analysis.com is a content-driven platform that publishes detailed articles (see figure 3.3) on F1 strategy, driver performance, and prediction using mathematical models (F1 Analysis, 2025). The articles cover a wide range of topics such as driver rankings, driver comparisons, and career retrospectives. However, the site is limited by its static nature. As a result, it serves as a passive information source rather than a tool for hands-on data exploration. Instead, this project is designed to encourage users to perform their own analyses through an accessible and intuitive interface that allows selection of datasets, filters, plot types, and analysis methods.



Figure 3.3: Screenshot of the F1 analysis 2024 Pre-Season Predictions article

FastF1 is a Python package that provides access to a wide range of Formula 1 data, including lap timing, car telemetry, tyre information, and weather data (The FastF1 Development Team, 2025). One of its primary strengths lies in its Python-friendly interface. However, FastF1 lacks support for interactive visualisations and statistical analysis functions, and does not offer any GUI. This highlights the need for an accessible tool that combines customisable visualisations with configurable statistical analysis methods.

Tool	Strengths	Limitations
F1 Tempo	- Interactive lap time and telemetry visualisations - Highlights invalid laps and outliers - Identifies fastest lap	- No static plots for export or offline use - Only supports single driver and single lap telemetry visualisations - No statistical insights or predictive functionality

Continued on next page

Tool	Strengths	Limitations
TracingInsights	- A vast number of data analysis methods specific to motorsport - Customisable inputs and a wide range of visualisations	- Visually cluttered interface - Slow load times (up to 10 minutes) - Frequent errors and poor system feedback - Limited plot customisation and configuration
F1-Analysis.com	- Provides accessible, in-depth articles - Does not require user inputs or technical knowledge - Covers a wide range of strategic and performance topics	- Static content only - No interactive plots or configurable visualisations - No user-controlled analysis
FastF1	- Python package for developers and data scientists - Native integration with `Matplotlib` for visualisation	- No GUI, thus inaccessible to non-programmers - No support for interactive visualisations - Lacks statistical analysis features

Table 3.1: Comparison of existing Formula 1 data analysis tools and platforms

Chapter 4

Planning

This chapter describes the project timeline, highlighting the key milestones and activities required to achieve the project's objectives (see chapter 2). In addition, a comprehensive risk analysis has been conducted to identify potential challenges and strategies to mitigate their impact.

4.1 Schedule of activities

The project has been divided into distinct stages, each focusing on specific objectives. These stages are aligned with the project's timeline, as illustrated in the Gantt chart (See appendix C). Considering the project's strict time constraints, the Waterfall methodology has been used to plan and develop the application (see section 6.1). The Waterfall methodology is visualised through the Gantt chart, where all stages are consecutively planned. This structured approach ensured consistent progress towards meeting all objectives within the time frame (see chapter 2). The project stages and their corresponding tasks are as follows:

Stage	Task description
Research and Design (Nov 2024 – Dec 2024)	**Literature review**: Literature related to the scope of this project is reviewed to establish a solid foundation of data analysis, particularly in motorsport.
	Competitor analysis: Competitor platforms are analysed, highlighting strengths and limitations which will help shape the requirements of the tool.
	Requirement definition: Based on prior research, the functional and non-functional requirements are defined whilst ensuring alignment with the project's aim and objectives (see chapter 2 and 5).
	Methodology research: Research is performed to determine which approach is most suitable and effective for achieving the system's requirements.
	System architecture design: High-level system architecture diagrams, such as package or component diagrams, are created to show the abstraction of the application into packages and how these packages communicate with each other.
	GUI design: Sketches of the user interface, which the user will directly interact with to perform data analysis functions, are designed.
	Finalising designs: Designs are finalised by cross-referencing them against the requirements to ensure accuracy and feasibility.

Continued on next page

Stage	Task description
Implementation (Jan 2025 – Mar 2025)	**Set-up development environment**: Create the project structure and GitHub repository.
	API integration: Retrieve the data through the Jolpica-F1 API and convert it to a suitable format for data analysis.
	Data visualisation development: Develop the plot generation features, such as lap time and comparative visualisations.
	Statistical analysis development: Develop the statistical analysis features, such as correlation, regression, and time series analysis.
	User interface development: Implement the UI, ensuring usability and seamless navigation between visualisation and data analysis features.
	Data export functionality: For further analysis and evaluation, implement a data export function, offering the results in various formats.
Testing (Mar 2025 – Apr 2025)	**Unit testing**: The functionality of each package is tested to ensure the code acts as expected and meets the requirements.
	External calculations: Statistical analysis results are calculated externally and compared to ensure accuracy of the tool's output.
	Bug fixing and refinement: Bugs or unexpected behaviours are resolved and tested to verify the fixes.
	Final validation: Test results and bugs are documented, ensuring the system is fully tested and verified.
Finalisation (Mar 2025 – May 2025)	**Technical report writing**: The full process is documented in the technical report.
	Final review and submission: Feedback is incorporated, and the final report is submitted.

Table 4.1: Development timeline and task breakdown

4.2 Risk analysis

A thorough risk analysis of the project has been conducted to identify potential challenges and to describe suitable mitigation strategies. Table 4.2 summarises the risks, their impact, probability, and suitable contingency plans. The analysis is used to ensure smooth progress.

Risk	Impact	Probability	Mitigation Plan
Delay in accessing or retrieving data from the Jolpica-F1 API	High	Medium	Implement caching to reduce API dependency; use local sample datasets for development and testing.
Inaccuracies or inconsistencies in data	High	Low	Perform data validation during preprocessing; implement error-handling and fallback logic to handle missing or incorrect values.
Performance issues with large datasets	High	Medium	Optimise data handling using efficient libraries (e.g., Pandas, NumPy); Use design patterns and good practices to minimise required computational cost.

Continued on next page

Risk	**Impact**	**Probability**	**Mitigation Plan**
Technical difficulties in integrating complex statistical methods	Medium	Low	Focus on the modular development of statistical methods; Validate methods independently using controlled datasets.
Time constraints impacting project milestones	High	High	Follow a structured project plan with buffer periods; Prioritise critical tasks to ensure punctual completion of key stages.
Compatibility issues with external libraries or tools	Medium	Low	Use well-documented and accessible libraries.
Incorrect development or implementation of statistical methods	High	Medium	Validate statistical methods against known results; Compare results to external or manual calculations.

Table 4.2: Risk Analysis and Mitigation Plan

Chapter 5

Requirements

Through a combination of literature review, analysis of existing systems, and consultation with the project's supervisor, the requirements for this project have been established (see Appendix D for full requirements). Each requirement has been categorised into either functional or non-functional requirements. This section outlines the requirements, which have been prioritised based on their significance to the project's objectives (see chapter 2).

5.1 Functional requirements

The functional requirements define the functionality of the system (see table 5.1 for the primary functional requirements). They describe the capabilities and attributes of the system which are necessary to meet the project's objectives (see chapter 2). They are prioritised as high, medium, or low based on their importance and impact on the system's functionality.

ID	Description	Inputs	Outputs	Priority
FR1	The system must retrieve and clean data from the Jolpica-F1 API and provide it in a suitable format for data analysis.	Resource type, filters parameters	Cleaned, analysis-ready data	High
FR1.1	Data retrieval must support all resource types available through the Jolpica-F1 API.	Resource type selection	Raw JSON data for the selected resource	Medium
FR1.2	The user must be able to specify data retrieval parameters such as limit and offset.	Parameters: limit, offset	Paginated data entries based on specified limits	low
FR1.3	The system must allow filtering by both mandatory and optional parameters, based on the selected resource type.	Filter name and value	Filtered API response matching the criteria	Medium
FR1.4	Data retrieval must return all data entries for the specified resource type and filters (via a "get all data" function).	Resource type, filters	JSON data containing all data entries	High

Continued on next page

ID	Description	Inputs	Outputs	Priority
FR1.5	Optional caching must be implemented to reduce redundant API calls.	Cache flag, file name	Cached JSON file	High
FR1.6	Data preprocessing must be integrated within the data retrieval process.	Raw API data (JSON)	Cleaned Pandas DataFrame	High
FR2	The tool must generate plots using the preprocessed datasets.	Cleaned data (DataFrame), plot type	Generated plot	High
FR2.1	The system must support generation of the following plot types: line, bar, scatter, box, and pie.	Plot type	Corresponding generated plot	High
FR2.2	The visualisation component must support both static and interactive plots.	Plot type	Static or interactive plot	Medium
FR2.3	The system must support exporting static plots in JPG, PNG, SVG, and PDF formats. Interactive plots should additionally support export in HTML format.	Generated plot, export format	Plot in selected format saved to the plots directory	High
FR3	The tool must support statistical analysis on the preprocessed dataset.	Cleaned data (DataFrame), analysis type	Analysis result or model output	High
FR3.1	The system must include descriptive statistical calculations: mean, median, mode, standard deviation, and variance.	DataFrame, numerical column name	Single value result per metric	High
FR3.2	The system must support comparative statistical methods, including: paired and unpaired t-tests, ANOVA, Pearson and Spearman correlation analysis, Wilcoxon signed-rank test, and Chi-squared test.	DataFrame, two column names	Test statistic, p-value	High
FR3.3	The system must implement trend analysis methods, including: simple moving average, exponential moving average, linear regression, ARIMA, and Holt-Winters exponential smoothing.	Dataframe, time-series column name	Predicted values	High

Continued on next page

ID	Description	Inputs	Outputs	Priority
FR4	The graphical user interface (GUI) must allow the user to fully access and utilise all tool functionalities, including data retrieval, configuration, visualisation, and analysis.	User interactions across GUI components	Plot visualisation and analysis results	High
FR4.1	The GUI must enable users to specify the resource type and provide all required and optional filters for retrieving data from the Jolpica-F1 API.	Resource type selection, filter inputs	Retrieval of the corresponding dataset	High
FR4.2	The GUI must support custom configuration for data visualisation.	Plot parameters and configuration options	Rendered plot	High
FR4.3	The GUI must support input of parameters required for performing statistical analysis.	Selected analysis function, columns, extra parameters	Analysis results	High
FR4.4	The GUI must include a dedicated area for displaying generated plots and analysis results.	Plot or analysis result from backend	Plot or result rendering in GUI area	High

Table 5.1: Functional Requirements

5.2 Non-functional requirements

The non-functional requirements describe the qualities and constraints of the system (see table 5.2). Collectively, these quality attributes define how the system should behave.

ID	Description	Inputs	Outputs	Priority
NFR1	The system must efficiently handle large datasets, maintaining low response times and consistent performance across all operations.	Large datasets, complex operations	Fast response times, system responsiveness	High
NFR2	The graphical user interface (GUI) must be intuitive, user-friendly, and accessible, allowing seamless navigation across all tool features.	User actions	Intuitive layout and navigation	High
NFR3	The system must validate data retrieved from the API to ensure correctness, completeness, and accuracy, reducing the risk of processing invalid or inconsistent data.	Raw API data	Cleaned, validated data	Medium

Continued on next page

ID	Description	Inputs	Outputs	Priority
NFR4	The system architecture must be modular and scalable to support future feature extensions with minimal impact on existing components.	New features or modules	Seamless integration, minimal modification to existing codebase	Medium
NFR5	Clear documentation and user guides must be provided to assist both end-users and developers in understanding the system's functionality and usage.	User attempts to access documentation	Clear, accessible guides and documentation	Medium
NFR6	The system must implement robust error handling and provide informative feedback to users regarding any encountered issues, including the nature and cause of the error.	Runtime errors, API issues, or invalid user inputs	Error messages with specific details	Medium
NFR7	The codebase should be maintainable and modular, following consistent naming conventions and structured design to simplify debugging and future development.	Code modifications, debugging activities	Clean, modular code	High
NFR8	The system must support automated testing practices, including unit and integration tests, to facilitate ongoing verification and validation.	Test cases, unit/integration test functions	Test reports, bug identification	High

Table 5.2: Non-Functional Requirements

5.2.1 Traceability matrix

The traceability matrix provides a clear link between the project's objectives (see chapter 2), functional requirements, and non-functional requirements. This matrix helps track how each part of the system meets the project's goals, supporting both functional and non-functional needs.

Objective	Functional requirement	Non-functional requirement
1	FR1, FR1.1, FR1.2, FR1.3, FR1.4, FR1.5, FR1.6	NFR1, NFR3, NFR4, NFR7
2	FR2, FR2.1, FR2.2, FR2.3	NFR2, NFR4, NFR7
3	FR3, FR3.1, FR3.2, FR3.3	NFR4, NFR7
4	FR4, FR4.1, FR4.2, FR4.3, FR4.4	NFR2, NFR4, NFR7
5	All	NFR8
6	None	NFR5

Table 5.3: Traceability matrix: Objectives, Functional, and Non-functional requirements

5.3 Requirement determination process

The requirements were determined through a structured process. The literature review was conducted to identify best practices for telemetry data analysis in motorsport, drawing insights from both academic research and industry standards. Research into competitor platforms revealed limitations and opportunities for improvement, helping to determine the tool's features and goals. Additionally, regular consultations with the project supervisor, Almas Baimagambetov, were critical for refining the project's objectives and clarifying functional expectations. The prioritised requirements outlined in this chapter provide a clear roadmap for system development, ensuring the tool aligns with user needs whilst maintaining high standards of performance, usability, and accessibility.

Chapter 6

Methodology

This chapter discusses the methodology for developing the data visualisation and statistical analysis tool. The approach was determined using extensive research into best practices for software development and data analysis, and all methodology decisions have corresponding justification. This section includes the project management methodology, the selection of development tools and techniques, including external Python libraries and version control, and the evaluation and testing strategies.

6.1 Project management methodology

Considering the scope and timeline of this project, the Waterfall model was selected as the project management methodology. This decision was based on several key factors outlined by Boehm (1988). First, the Waterfall model offers a linear and structured process, separating the project into distinct stages such as design, implementation, and testing. This structure aligns well with the project's academic requirements and hard deadline. Secondly, it promotes predictability by ensuring that each stage is completed before the next begins. This helps minimise the risk of scope creep or mismanagement through clearly defined deliverables and objectives (see chapter 2). Finally, its strong emphasis on documentation is particularly beneficial in an academic context. While alternative approaches like Agile were considered, they were deemed less appropriate given the fixed timeline and the emphasis on predefined deliverables rather than iterative development.

6.2 Development tools and techniques

This section evaluates and justifies the selection of key tools, such as the programming language, corresponding libraries, and the data source API used for this project. Each choice was evaluated against alternatives based on performance, usability, ease of integration, and suitability for achieving the project's aim and objectives (see chapter 2). The goal was to strike a balance between scalability, maintainability, and usability, supported by extensive research.

For scientific computing and data analysis, Imarticus Learning (2024) encourages the use of Python libraries such as Pandas, Matplotlib, SciPy, and Statsmodels. These libraries form the foundation of the project, offering reliable, well-documented, and efficient tools for data analysis. Their widespread use, combined with strong community support, ensures efficient and accurate development of the visualisation and statistical analysis features and methods.

6.2.1 Programming language

Python was selected as the primary programming language for this project due to its widespread use in the data science and analytics community. It is widely regarded as an industry standard for data-driven applications, offering an extensive ecosystem of libraries specifically designed for data manipulation, visualisation, and statistical analysis (W3Schools, 2025, GeeksforGeeks, 2025).

In comparison to alternatives such as R or MATLAB, Python offers greater flexibility for full-stack development and broader capabilities with web frameworks in a single environment. Whilst R is well-suited for statistical modelling, it lacks interactive and general-purpose capabilities required for developing a web-based application (The R Project, 2025). MATLAB, while powerful in numerical analysis, is proprietary and less accessible for open-source development and deployment (MathWorks, 2025).

Python's strong community support and extensive documentation further reinforced its suitability for this project. Although Python may face performance limitations when handling extremely large datasets, this project mitigated such issues through efficient data handling strategies, such as caching and selective filtering.

6.2.2 Development environment

PyCharm was selected as the integrated development environment (IDE) due to its robust support for Python, built-in debugging tools, and excellent integration with version control systems (JetBrains, 2025). Its advanced features, including intelligent autocompletion and integrated test runner, make it particularly well-suited for large-scale, professional software development. Consequently, its relatively high resource usage may impact performance on less powerful machines.

Less resource-intensive alternatives, such as Visual Studio Code and JupyterLab, were also considered. Visual Studio Code provides a lightweight development environment with a rich ecosystem of extensions (Microsoft, 2025), offering the majority of the features available in PyCharm. However, it requires additional configurations for full Python support, including debugging, linting, and testing capabilities. JupyterLab, whilst suitable for data science, lacks support for large-scale projects and advanced debugging features (Project Jupyter, 2025). As a result, it was deemed unsuitable for the structured and modular development required in this project.

6.2.3 Version control

Git was chosen as the Version Control System (VCS) for this project due to its reputation as the industry standard and its widespread adoption across software development environments. It provides robust version tracking, allowing developers to revert to previous versions when necessary. Despite its relatively steep learning curve, Git was deemed more appropriate than alternatives such as Mercurial or Subversion, mainly due to its popularity, extensive community support, and compatibility with all major Python development platforms (The Git Community, 2025).

GitHub was selected as the remote repository platform, offering secure, cloud-based storage and version control. Its accessibility, collaboration features, and integration with Git make it a reliable solution for managing project code and ensuring consistent access across development environments.

6.2.4 API selection and data integration

The Jolpica-F1 API was selected as the primary data source for this project due to its lightweight responses, flexibility, and RESTful architecture (Jolpica Project, 2025). As the successor to the now-depreciated Ergast API (Ergast Developer API, 2024), Jolpica-F1 offers access to a wide range of F1 data, including driver standings, lap times, pit stop durations, and qualifying results. Additionally, the API offers extensive filtering, enabling precise data selection. These features make it well-suited for applications requiring targeted, low-latency responses in a web-based environment.

In contrast to the FastF1 Python package, which can return up to 100 MB of data, Jolpica-F1 provides lightweight JSON responses, typically only a few kilobytes. Furthermore, its RESTful architecture simplifies integration and removes the need for additional dependencies. Given that the FastF1 relies on Jolpica-F1 as one of its backend sources, accessing the API directly reduces unnecessary abstraction and improves integration with the frontend.

For handling and processing data, this project uses Pandas and NumPy, both of which are widely adopted in the data science community for their performance and practical syntax (The Pandas Development Team, 2024, The NumPy Development Team, 2022). These libraries ensure efficient data transformation, validation, and formatting before visualisation or advanced statistical analysis.

6.2.5 Visualisation libraries

The project applies a dual-library approach to visualisation, using both `Matplotlib` and `Plotly` to support static and interactive chart generation. `Matplotlib`, one of the most widely used Python libraries for static visualisation, was selected for its extensive configuration capabilities and ability to produce high-quality, publication-ready charts (The Matplotlib Development Team, 2025). Its static output is well-suited for exporting plots for reuse in documentation or presentations. To enhance the aesthetic appeal of static plots, `Seaborn` was integrated alongside `Matplotlib`. Built on top of `Matplotlib`, `Seaborn` offers similar configurability but with more refined default styling and layout options (Waskom, M. and the Seaborn Development Team, 2024).

For interactive visualisations, `Plotly` was chosen due to its responsive, browser-compatible rendering and its robust integration with the Dash framework (Plotly Technologies Inc., 2025*b*). `Plotly` supports a similar range of plot types and customisation features as `Matplotlib`, promoting the implementation of a unified, general-purpose plot generation interface. This promotes modularity and maintainability, as similar parameters can be used across both static and interactive plots.

Alternative libraries such as `Bokeh` and `Altair` were considered but ultimately not adopted. `Plotly`'s rich documentation, larger community, and native compatibility with Dash made it the more practical choice for this project. However, one notable limitation of Plotly is its lack of flexibility for static plot export, which is why Matplotlib and Seaborn were retained for static plot generation. The use of both libraries ensures the system supports diverse user needs, from interactive data exploration to high-quality static plot generation.

6.2.6 Statistical analysis libraries

To support a wide range of statistical analysis capabilities, the project integrates several well-established Python libraries, selected for their suitability, strong documentation, and extensive functionality. For comparative analysis, the `SciPy` library was included due to its extensive support for statistical methods, including t-tests, ARIMA, and correlation analysis (The SciPy

Community, 2025). These methods allow users to perform meaningful statistical comparisons and assess the significance of observed differences.

Trend-based analysis was facilitated by the `Scikit-learn` and `Statsmodels` libraries. While `Scikit-learn` is predominantly a machine learning library, it provides a simple and dependable implementation of linear regression, which can be applied to forecast performance trends and test the significance of temporal predictors (The Scikit-learn Developers, 2025). In contrast, `Statsmodels` is specifically designed for statistical modelling and time-series forecasting. It offers a robust implementation of advanced models such as ARIMA and Holt-Winters, both of which are well-suited to motorsport telemetry data (The Statsmodels Development Team, 2024).

ARIMA (AutoRegressive Integrated Moving Average) models are widely used for forecasting time-dependent data by combining autoregression with differencing and moving averages to capture both trend and seasonality. Holt-Winters exponential smoothing extends this approach by applying weighted averages and incorporating level, trend, and seasonal components, making it suitable for datasets with recurring patterns (Montgomery et al., 2008). These models were selected based on their interpretability, ease of implementation, and suitability for post-race analysis, such as forecasting lap times or performance trends.

Alternative approaches, such as implementing these models from scratch or using higher-level libraries like `PyCaret` or `TensorFlow`, were considered but ultimately rejected due to their additional complexity and lack of transparency. By leveraging the selected libraries, the system maintains interpretability, performance, and alignment with the project's focus on statistically sound, user-accessible insights.

6.2.7 GUI framework

`Dash` was selected as the primary library for developing the application's graphical user interface due to its Python-native framework and compatibility with `Plotly`, making it particularly appropriate for data-driven web applications (Plotly Technologies Inc., 2025*a*). `Dash` facilitates the development of interactive and web-based interfaces without requiring knowledge of JavaScript or front-end development. This supports the front-end development process, as the primary focus of this project lies on the data analysis aspect of the tool. Built on top of `Flask`, `Dash` inherits robust server-side functionality, including routing, request handling, and session management (The Pallets Projects, 2024). This architecture provides the flexibility necessary to manage dynamic content, user interactions, and real-time updates.

Alternative frameworks, such as `Tkinter` and `PyQt`, were considered but dismissed due to their focus on desktop applications and lack of support for interactive visualisations. `Streamlit`, whilst similar to Dash in purpose, was not selected as it offers less control over layout structure and component behaviour.

A potential limitation of `Dash` is its relatively limited layout customisation compared to JavaScript-based frameworks or CSS-rich front-ends. However, its native `Plotly` support, strong documentation, and ease of implementation outweigh this constraint, especially considering the scope of this project. `Dash` facilitates the efficient development of a functional and interactive GUI that supports the project's visualisation and analytical requirements (see chapter 5).

6.3 Evaluation and testing tools and techniques

Evaluation and testing of the application are crucial to ensure accuracy, reliability, and usability. A combination of testing techniques is employed to validate the functionality and

correctness of the system. Unit testing verifies the implementation of individual components, such as data retrieval, plot generation, and statistical analysis methods. Once the functionalities of these components were validated, integration testing was conducted to ensure proper communication and cohesion across modules. Additionally, the results of statistical analyses were validated by cross-referencing outputs with those generated by alternative statistical tools, such as Microsoft Excel with the XLSTAT extensions, or through manual calculations. Python's `pytest` framework was used throughout the testing process due to its support for parameterisation and approximation, which proved particularly useful when verifying API data retrieval and statistical outputs.

Chapter 7

Product description

This chapter describes the F1 Data Analysis Tool, covering key aspects such as design, implementation, development, and testing, including justification for all choices made.

7.1 Design

This section covers the design of the tool's architecture and Graphical User Interface (GUI). The designs emphasise usability, offering support for a wide range of analysis methods whilst providing an intuitive interface that is accessible to amateur analysts with minimal coding experience. Additionally, data abstraction is achieved through the use of a dedicated API package, which decouples external data sources from the core system logic, facilitating easier future development without requiring changes to the data retrieval module.

7.1.1 System architecture

The tool is architected following a modular design paradigm that enables separation of concerns, maintainability, and scalability. The system is divided into distinct packages, each responsible for covering several functional requirements, including data retrieval, preprocessing, visualisation, analysis, and user interaction (see chapter 5). This layered approach ensures that core logic remains decoupled from the presentation logic (see figure 7.2), thus promoting maintainability and scalability whilst enabling easier testing. As shown in the package diagram (see figure 7.1), the system is organised into five main packages:

Package	Modules	Responsibilities
`api`	`jolpica_api.py`, `data_preprocessing.py`, `json_handler.py`, `cache_manager.py`	Retrieves, preprocesses, and caches API data.
`visualisation`	`plot_generator.py`, `static_plot.py`, `interactive_plot.py`, `plot_saving.py`, `plot_utils.py`	Generates static and interactive plots; supports configuration like hue, theme, and axis orientation.

Continued on next page

Package	Modules	Responsibilities
`analysis`	`analysis_main.py`, `descriptive_analysis.py`, `comparative_analysis.py`, `trend_analysis.py`	Provides descriptive, comparative, and trend analysis through a wrapper module.
`enumeration`	`resource_types.py`, `plot_types.py`, `analysis_functions.py`	Defines enums to ensure consistent referencing across the system.
`gui`	`main.py`, `layout.py`, `callbacks/data.py`, `callbacks/plots.py`, `callbacks/analysis.py`	Manages GUI layout and interactive logic; displays plots and analysis results.

Table 7.1: Project architecture: Package modules and responsibilities

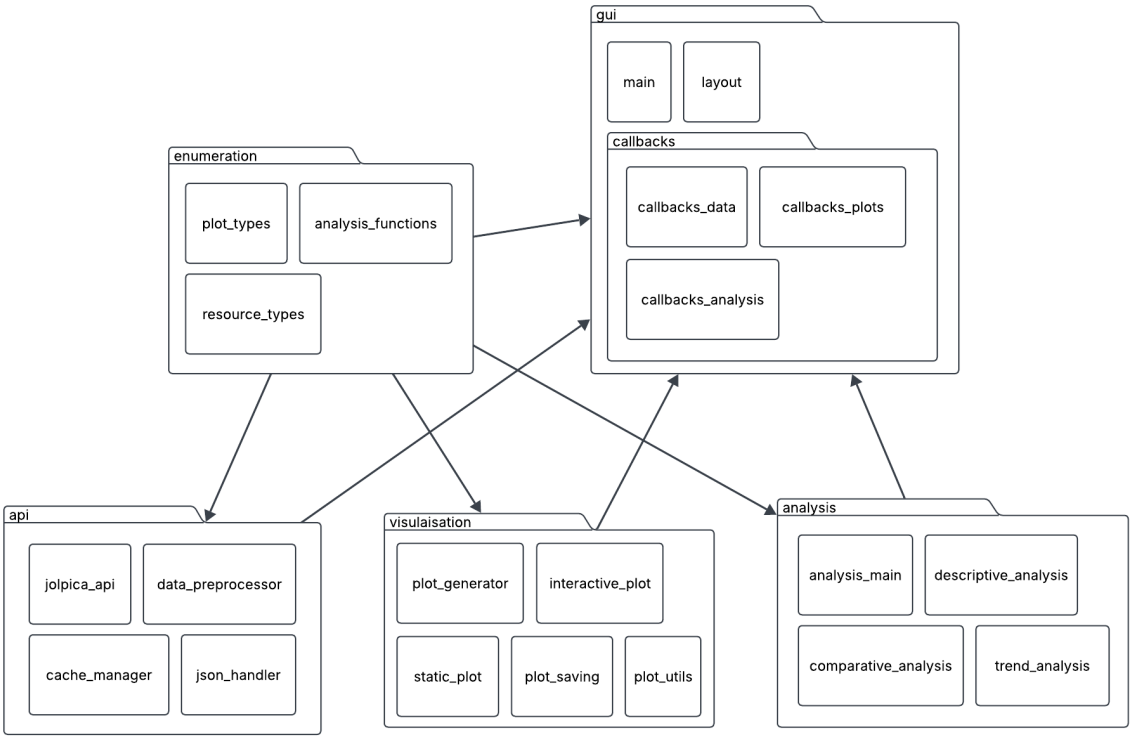


Figure 7.1: The package diagram for the F1 data analysis tool

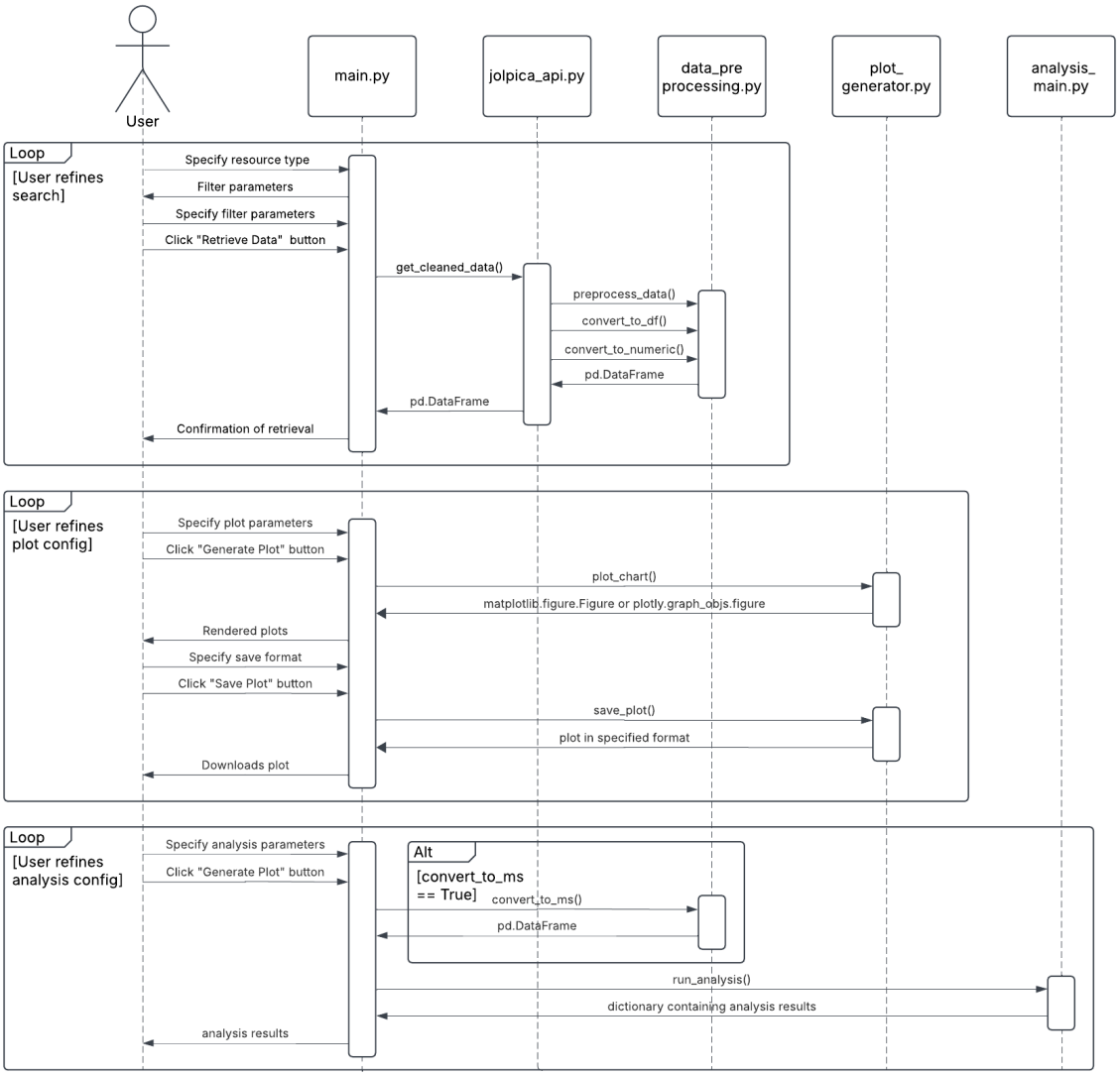


Figure 7.2: The sequence diagram for the F1 data analysis tool

7.1.2 GUI layout sketch

The initial user interface sketches are structured around a multi-tabbed layout, with each tab corresponding to a specific analysis type. A consistent sidebar navigation panel enables easy navigation between different views. The sketches visualise several core functional areas. The line chart, scatter plot, bar chart, and box plot tabs allow users to dynamically visualise F1 data based on selected input parameters, with generated plots displayed in a central results area (see figure 7.3). Similarly, the regression analysis, time series analysis, and hypothesis testing tabs share a common layout but will allow users to specify different inputs tailored to each statistical method. The export data tab includes fields for saving analysis results for offline use, while the about tab provides additional information about the tool, including access to a user guide and supporting documentation. Additional sketches are provided in the appendix E.

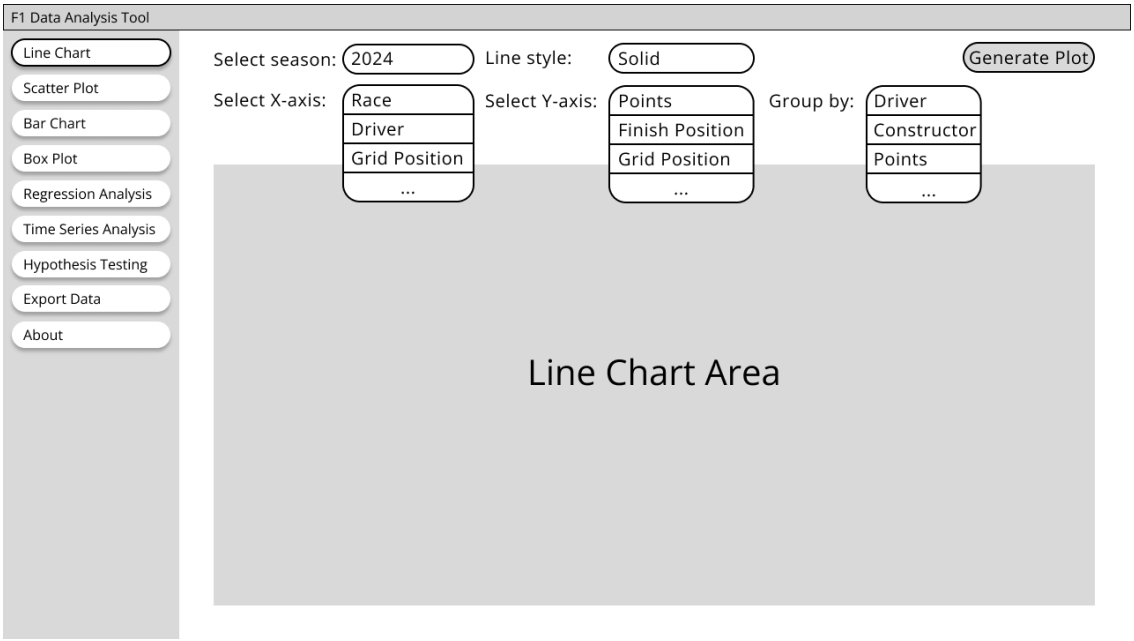


Figure 7.3: Initial sketch of the GUI for the F1 data analysis tool

7.2 Development and Implementation

This section outlines the development and implementation of the F1 Data Analysis Tool, detailing the process from environment setup to challenges faced and key features.

7.2.1 Environment setup and configuration

The application was developed using Python 3.11. A new Python project was initiated in PyCharm, named 'F1DataAnalysisTool'. A corresponding GitHub repository was created to store and manage the project. After setting up the repository, the project files were pushed to GitHub. The project was then populated with the packages and modules outlined in the package diagram (see figure 7.1), although the modules initially contained no code. Furthermore, the `requirements.txt` file was created to list the required libraries and their versions, ensuring consistent development across various environments. Once these changes were pushed to Git, the environment setup and configuration were complete, enabling an efficient development workflow for the tool.

7.2.2 Key implementation phases

The development of the application is separated into four key phases, which correspond with the four main functional requirements (see table 5.1):

1. **Backend development (FR1)**: The first phase of development focused on data retrieval and cleaning. The task was to fetch and clean data from the Jolpica-F1 API using the `requests` library and preprocess it using `pandas`. I implemented functions to handle the paginated and nested JSON responses from the API. Preprocessing functions such as flattening JSON structures and conversions to `Pandas` DataFrames ensured the data could be easily used for further analysis. The functions required low complexity to ensure the timely handling of large datasets. The backend development was a

fundamental step, as the analysis depends heavily on accurate and correctly structured data.

2. **Data visualisation (FR2)**: Once the backend was ready, I moved on to the visualisation phase. This involved creating both static and interactive plots using `matplotlib`, `seaborn`, and `plotly`. The tool was designed to support various types of plots. I focused on static plot generation first, integrating optional yet useful plot configurations. The interactive chart generation function includes the same optional configurations, which enabled the development of a generic `generate_plot` wrapper function, promoting scalability and maintainability. The plot configuration functions were converted to generic functions to ensure compatibility with the wrapper function.

3. **Statistical analysis (FR3)**: After visualising the data, I implemented the statistical analysis features using `scipy`, `statsmodels` and `scikit-learn`. The implemented functions range from simple descriptive analysis to complex analysis methods. Collectively, the analysis functions allow users to perform advanced, meaningful statistical computations on the datasets.

4. **GUI development (FR4)**: The final phase involved designing and implementing the GUI with `Dash`. The layout of the interface differs from the initial design due to the incomplete understanding of the input fields and filtering required by the Jolpica-F1 API during the design stage. Instead, a consistent data retrieval section was implemented, positioned above two primary tabs: one for data visualisation and another for statistical analysis (see figures 7.6 and 7.7). I ensured that input elements and filter options update dynamically based on the selected resource type, making the interface intuitive and responsive. Given the extensive number of analysis functions, assigning each one to an individual tab would have resulted in an excessively long and clustered sidebar, negatively impacting usability. Furthermore, many analysis functions share a similar layout, which allows for a modular implementation. To enhance usability and accessibility, drop-down menus were implemented to allow users to select the desired plot type or analysis function, replacing the multi-tabbed layout with a fixed sidebar navigation panel originally proposed in section 7.1.2. Additionally, drop-down menus are provided for parameter configurations, with safeguards in place to prevent duplicate column selections. Emphasis was placed on creating a user-friendly interface that remained adaptable and scalable for the integration of future features.

The division of development into distinct phases aligns well with the Waterfall project management methodology (see section 6.1), as each phase was fully implemented before proceeding. This approach supported structured development, minimal reworking, and enabled focused attention on individual requirements, making it well-suited for this project.

7.2.3 Code structure and core modules

The code was structured in a modular fashion to ensure readability, maintainability, and scalability. Table 7.2 shows an overview of the key modules and their responsibilities. See appendix G for the full source code of the modules described below.

Module	Functions	Responsibilities
`jolpica_api.py`	JolpicaAPI class: API interaction, filter specification, JSON handling, cache management, data cleaning pipeline	Handles API interaction and data cleaning pipeline; promotes modularity and reusability through an object-oriented approach
`data_peprocessing.py`	Save and load CSV, flatten JSON data, validate data, convert to DataFrame, milliseconds, and numeric values	Handles data cleaning and preparation for analysis by handling data transformation and validation tasks.
`plot_generator.py`	Generate filename, plot chart: arguments for title, plot type, save format, and additional parameters (`**kwargs`)	Provides a unified plotting interface for generating static and interactive plots with file saving and dynamic filename generation.
`analysis_main.py`	Run analysis: Unified dispatcher for data analysis types	Facilitates various statistical analyses and handles error validation for parameters.
`main.py`	Web app setup, loads layout, registers callbacks, serves static plots via `Flask` route	Implements core functionality of the system, processes user interactions, and renders analysis results.

Table 7.2: Core modules: Functions and responsibilities

7.2.4 Challenges faced and solutions implemented

During development, I encountered several challenges, which I was able to overcome through various solutions. These challenges and their corresponding solutions are detailed below.

Handling Large Datasets

Given the volume of F1 data, processing large datasets posed a significant performance challenge. To optimise data processing, I leveraged the efficient data structures of `pandas` and used `numpy` for heavy numerical computations. Additionally, caching was implemented on both the raw data from the API and the cleaned data, which helped minimise redundant retrieval or cleaning operations. The `get_data`, `get_all_data`, and `get_cleaned_data` functions first check for a cached version and return it if available, preventing any unnecessary processing. If caching is enabled (which it is by default), these functions store the data after successful retrieval or cleaning. Since the API provides historical data, there is no concern about data changing between requests. The caching implementation at multiple stages ensures the data retrieval and cleaning pipeline operates efficiently, providing timely responses and reducing processing overhead.

```
# Retrieves data from the Jolpica-F1 API with optional caching
def get_data(self, use_cache: bool = True) -> Dict[str, Any]:
    # Return cached file if cache file exists
    cache_file = self.get_cache_file_path_params()
    if cache_manager.is_cached(cache_file):
        return cache_manager.load_cache(cache_file)

    # --- omitted: Data retrieval logic ---

    # Cache data if data caching is enabled
```

```
    if use_cache:
        cache_manager.cache_data(cache_file, data)
    return data

# Retrieves all data from endpoint using pagination
def get_all_data(self, use_cache: bool = True) -> Dict[str, Any]:
    # Return cached file if cache file exists
    cache_file = self.get_cache_file_path_all()
    if cache_manager.is_cached(cache_file):
        return cache_manager.load_cache(cache_file)

    # --- omitted: Data retrieval logic ---

    # Cache data if cache is enabled
    if use_cache:
        cache_manager.cache_data(cache_file, all_data)
    return all_data

def get_cleaned_data(self, use_cache: bool = True) -> pd.DataFrame:
    # Return cached data if file exists
    file_name = self.get_cleaned_file_name()
    if dp.is_loaded_csv(file_name):
        return dp.load_from_csv(file_name)

    # --- omitted: Data cleaning pipelines ---

    # Cache data if cache is enabled
    if use_cache:
        dp.save_to_csv(df, file_name)
    return df
```

Listing 7.1: get data, get all data, and get cleaned data functions in jolpica_api.py

Pagination handling

The API returns data in nested dictionaries, which requires careful handling to ensure it is correctly concatenated. I overcame this challenge by implementing a custom concatenation function that properly concatenates paginated data based on the format of the provided data. The function compares the last entry in the first provided list to the first entry in the second provided list, and then merges the entries based on the comparison results. Consequently, duplicate entries are avoided, and unique entries are merged, solving the problem of overlaps at pagination boundaries. Additionally, the function handles input validation.

```
# Concatenates the inner data with additional provided data
def extend_inner_data(data: List, additional_data: List) -> List:
    # --- omitted: Data validation and empty argument handling ---

    # Retrieve the entries that require concatenating
    last_inner_entry = data[-1]
    first_additional_entry = additional_data[0]

    # --- omitted: Data validation and handling ---

    # Identify the keys present in both entries
```

```
    common_keys = set(last_inner_entry.keys()) & set(first_additional_entry.keys())

    # Compare the values of the common keys
    common_data_found = any(last_inner_entry[key] == first_additional_entry[key] for key in common_keys) if common_keys else False

    # If no common keys or common data entries have been identified, extend and return the data
    if not common_keys or not common_data_found:
        data.extend(additional_data)
        return data

    # Ensure the values of the common keys both contain lists
    for key in common_keys:
        if isinstance(last_inner_entry[key], list) and isinstance(first_additional_entry[key], list):
            # Identify unique entries and merge them
            existing_values=[item for item in last_inner_entry[key]]
            new_values=[item for item in first_additional_entry[key] if item not in existing_values]
            last_inner_entry[key].extend(new_values)

    # Skip first additional entry and extend remaining data
    data.extend(additional_data[1:])
    return data
```

Listing 7.2: extend inner data function in json_handler.py

Generalising plot generation

Developing a reusable function capable of generating both static and interactive visualisations posed a significant challenge. To address this, a general-purpose wrapper was implemented, designed to accept flexible parameters and dynamically generate the appropriate plot type based on user input. Furthermore, plot utility functions, such as those for axis flipping and tick configurations, were also generalised. This approach promotes modularity and reusability across the visualisation package, improving development efficiency and simplifying future extensions.

```
def plot_chart(
        df: pd.DataFrame, x_col: str, y_col: str = None, title: str = "", plot_type: tuple = ("static", "line"), **kwargs
):
    # Extract plot mode and type
    mode, chart_type = plot_type

    # Generate plot
    if mode == "static":
        plot_function = plot_static_chart
    else:
        plot_function = plot_interactive_chart
    fig = plot_function(df, x_col=x_col, y_col=y_col, title=title, plot_type=chart_type, **kwargs)
```

```
13
14      # --- omitted: Plot saving logic ---
15
16      return fig
17
18  def apply_axis_flip(fig: go.Figure | plt.Axes, flip_axis: list,
        plot_type: str):
19      # define the axis flipping methods
20      flip_methods = {
21          "static": {
22              "x": lambda f: f.invert_xaxis(),
23              "y": lambda f: f.invert_yaxis()
24          },
25          "interactive": {
26              "x": lambda f: f.update_layout(xaxis=dict(autorange="
                    reversed")),
27              "y": lambda f: f.update_layout(yaxis=dict(autorange="
                    reversed"))
28          }
29      }
30
31      # flip the axis
32      for axis in flip_axis:
33          flip_methods[plot_type][axis](fig)
34
35
36  def configure_axis_ticks(fig: go.Figure | plt.Axes, df: pd.
        DataFrame, x_col: str, y_col: str = None):
37      # Function for setting the axis ticks
38      def set_tick(fig: go.Figure | plt.Axes, axis: str, col: str):
39          # Get min max values of the column
40          min_val, max_val = dp.get_column_min_max(df, col)
41          if min_val is not None and max_val is not None:
42
43              # Get difference between min and max
44              tick_vals = list(range(int(min_val),int(max_val)))
45
46              # Only configure ticks if diff is less than 30
47              if len(tick_vals) < 30:
48                  # Configure ticks based on plot type
49                  if isinstance(fig, plt.Axes):
50                      if axis == "x":
51                          fig.set_xticks(tick_vals)
52                      elif axis == "y":
53                          fig.set_yticks(tick_vals)
54                  elif isinstance(fig, go.Figure):
55                      fig.update_layout(**{f"{axis}axis": dict(
                            tickmode='array', tickvals=tick_vals)})
56
57      # Configure X-axis ticks
58      set_tick(fig, "x", x_col)
59      # Configure Y-axis ticks if provided
60      if y_col:
61          set_tick(fig, "y", y_col)
```

Listing 7.3: plot chart function in plot_generator.py

Dynamic GUI Filters

One of the most complex aspects of the project was implementing dynamic filters in the GUI. Different resource types require different filtering criteria, including mandatory filters, and these filters need to update dynamically based on user selections. I handled this by structuring the filter logic in a modular way, allowing it to update based on the selected resource type. Other callback functions could then refer to the dynamically updated filters through pattern-matching callbacks.

```
# Callback to render dynamic filter input fields based on resource type
@app.callback(
    Output('filter_inputs', 'children'),
    [Input('resource_type', 'value')]
)
def update_filters(resource_type):
    # --- omitted: input validation ---
    try:
        filter_inputs = []
        # Function for creating the filter input groups
        def create_inputs(filter_names, label_suffix):
            for name in filter_names:
                label = html.Label(f" {name.capitalize()} ({label_suffix}):")
                input_field = dcc.Input(
                    id={'type': 'dynamic-filter', 'index': name},
                    type='text',
                    placeholder=f'Enter {name}',
                    value=''
                )
                filter_inputs.extend([label, input_field])

        # Create the filter groups for mandatory and optional filters
        create_inputs(
            ResourceType.get_mandatory(resource_type),
            'required'
        )
        create_inputs(
            ResourceType.get_optional(resource_type),
            'optional'
        )

        # Return the filter inputs list
        return filter_inputs

    except Exception as e:
        logging.error(...)
        return [html.Div(...)]
```

Listing 7.4: update filters function in callbacks_data.py

Rendering static plots

Another user interface-related challenge involved visualising static plots within the `Dash` GUI. Interactive plots can be passed directly to the `dcc.Graph` component, which will handle the rendering seamlessly. However, static plots are not compatible with this component. To address this limitation, the `update_plot` function was designed to dynamically modify the `children` property of the plot container. Depending on the user-selected plot type, it renders either a `dcc.Graph` component for interactive plots or a `html.img` for static images.

For static plots, this approach required saving the generated figure as a PNG file before referencing it in the HTML image tag. A Flask-compatible route was implemented to serve these PNG files to the front end. To prevent browser caching and to ensure the latest version of the plot is displayed, a Unix timestamp is appended to the image URL as a query string. Additionally, the timestamp is used as a unique Dash component key to trigger UI re-rendering when the image changes. This approach enables seamless integration of static and interactive visualisations within the same `Dash` interface.

```
@app.callback(
    [Output('plot_area', 'children'),
     Output('plot_figure', 'data')],
    [Input('generate_plot', 'n_clicks')],
    # --- omitted: additional arguments ---
)
def update_plot(n_clicks, stored_data, plot_mode, plot_type, x_col,
    y_col, group_by, flip_axis):
    # --- omitted: input validation, case handling, data retrieval
        ---

    # Generate plot
    fig = plot_chart(
        data, x_col, y_col,
        title="F1 Data Analysis Plot",
        plot_type=(plot_mode, plot_type),
        hue=group_by,
        flip_axis=flip_axis
    )

    if plot_mode == 'static':
        # Save the generated plot to PNG format
        save_plot(fig, "cache_static", plot_type=plot_mode,
            file_format='png')

        # Return an HTML image tag containing the PNG file
        timestamp = int(time.time())
        return (html.Img(
            src=f"/data/plots/cache_static.png?v={timestamp}",
            style={'width': '100%', 'height': 'auto'},
            key=str(timestamp)
        ), {'figure': None, 'plot_mode': plot_mode})
    # --- omitted: interactive plot rendering ---
```

Listing 7.5: update plot function in callbacks_plots.py

7.2.5 Key features and functionalities

The system incorporates several core features that collectively enhance its analytical capabilities and user accessibility. One of the central components is its comprehensive data filtering functionality. Users can retrieve and filter various types of F1 data before performing analysis, with support for a wide range of resource types, including *Circuits*, *Constructors*, *Constructor Standings*, *Drivers*, *Driver Standings*, *Laps*, *Pit Stops*, *Qualifying*, *Races*, *Results*, *Seasons*, *Sprint*, and *Status*. Each resource type offers a unique set of filter parameters, such as *season*, *round*, *drivers*, *constructors*, *circuits*, *grid position*, *results*, *status*, *fastest laps*, *position*, *laps*, and *pitstops*. This flexible filtering capability enables precise data extraction, allowing users to extract the data most relevant to their analysis.

The system also supports the generation of both static and interactive plots, offering flexibility for diverse use cases. Interactive visualisations enable users to dynamically explore data through hover, zoom, and pan features, while static plots provide high-quality figures suitable for export and reporting. The supported plot types include *line*, *bar*, *scatter*, *box*, and *pie* charts, allowing users to identify trends, distributions, and comparisons effectively.

In addition, the system supports a range of statistical analysis methods categorised into three main groups: descriptive, comparative, and trend-based analysis. Descriptive statistics allow users to compute metrics such as *mean*, *median*, *mode*, *standard deviation*, and *variance*. Comparative analysis methods include statistical tests such as *paired and unpaired t-test*, *ANOVA*, *Spearman and Pearson correlations*, *the Wilcoxon test*, and *the Chi-square test*. These tools help determine the significance of differences or relationships within the data. For analysing performance trends over time, users can employ trend analysis methods such as *simple moving average*, *exponential moving average*, *linear regression*, *ARIMA model*, and *Holt-Winters model*.

Collectively, these analytical tools enable users to perform everything from straightforward data summarisation to complex data comparisons and forecasting, which aligns closely with the project's goal of transforming raw data into meaningful insights.

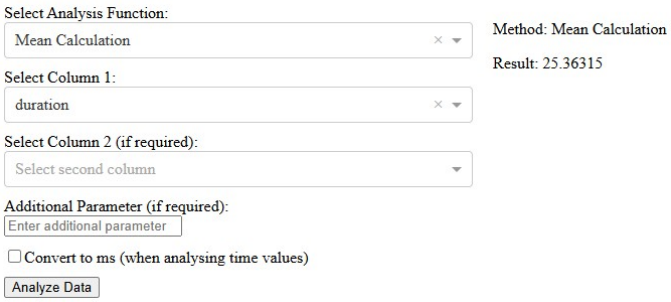


Figure 7.4: Mean calculation for pitstop duration

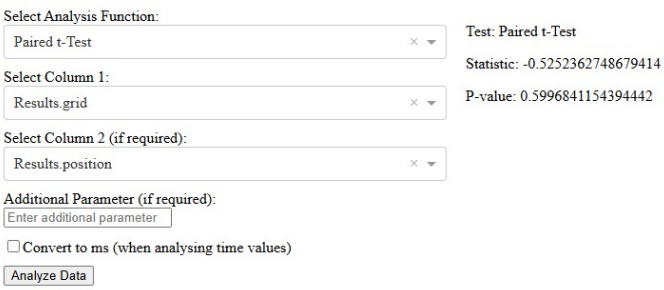


Figure 7.5: Paired T-test results for grid and finish position

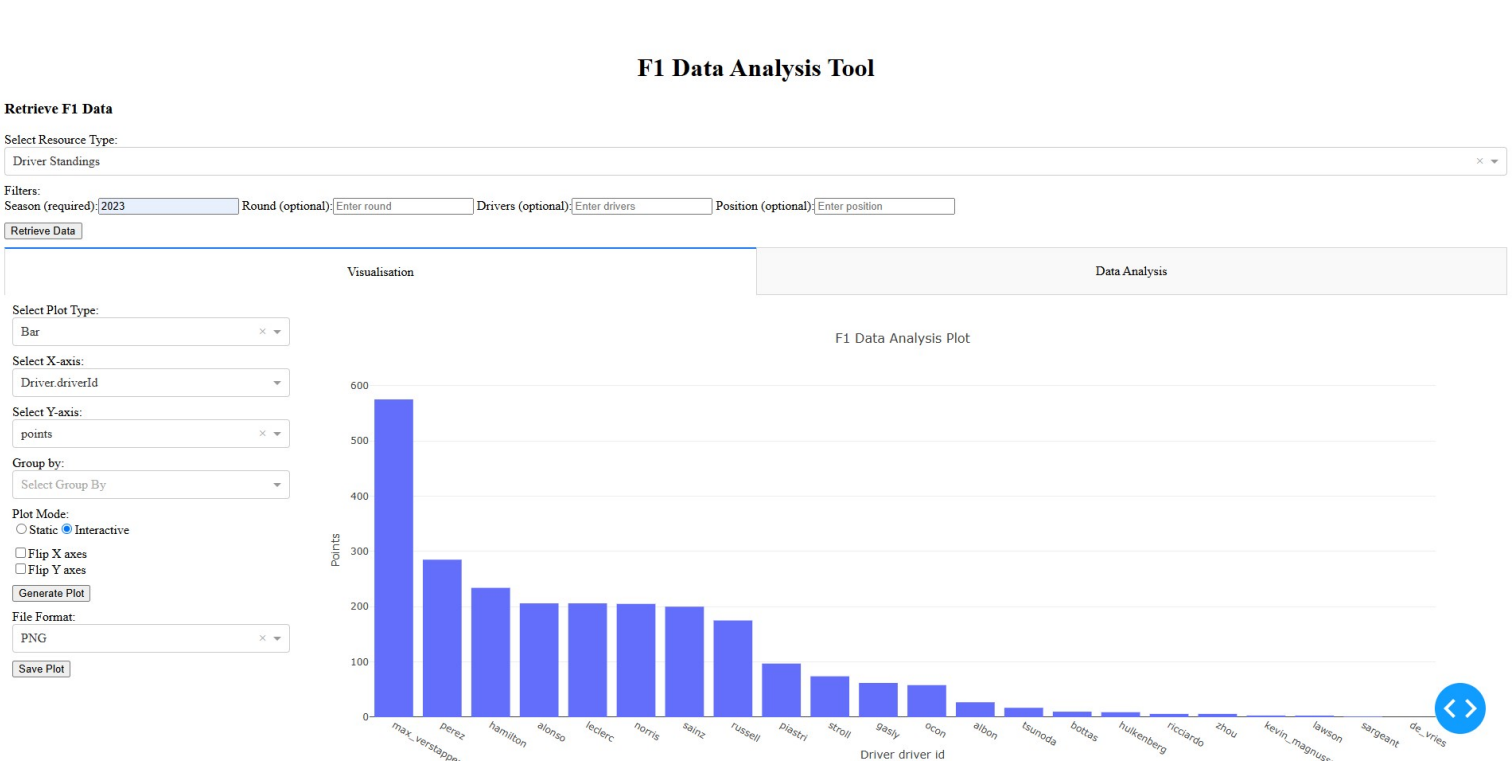


Figure 7.6: The tool's GUI for data visualisation, showing the points per driver bar chart for the 2023 season

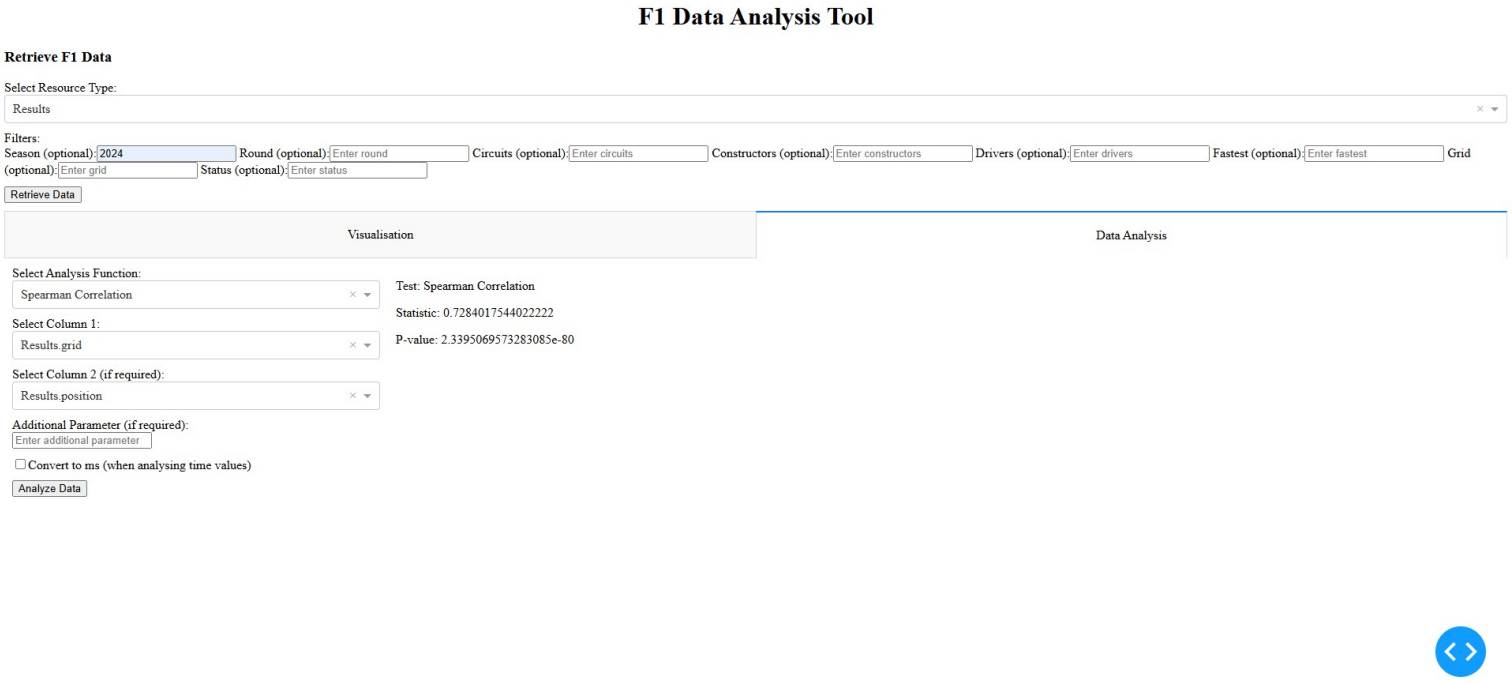


Figure 7.7: The tool's GUI for data analysis, showing the statistic and p-value of the Spearman correlation between qualifying and race position for the 2024 season

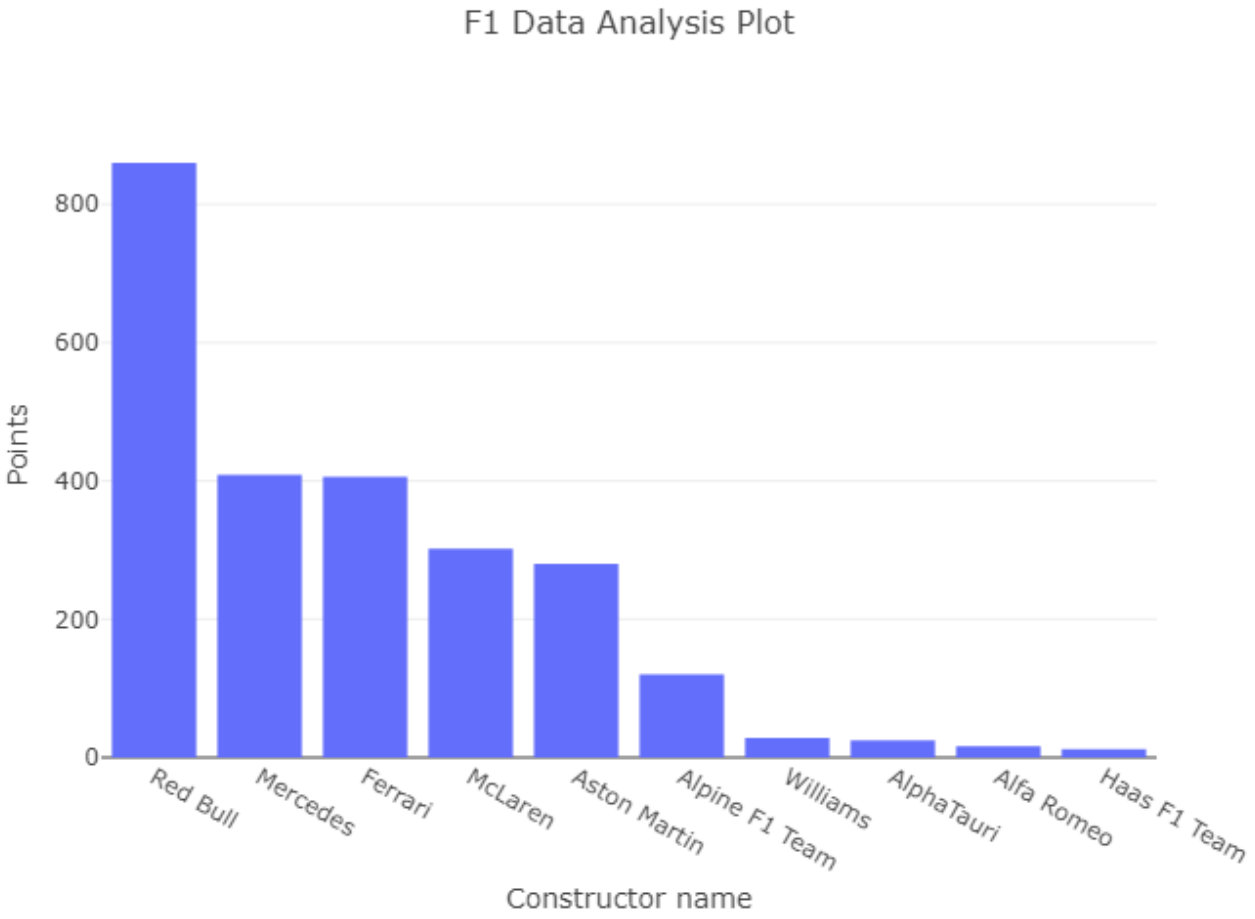


Figure 7.8: Points per constructor bar chart for the 2023 season

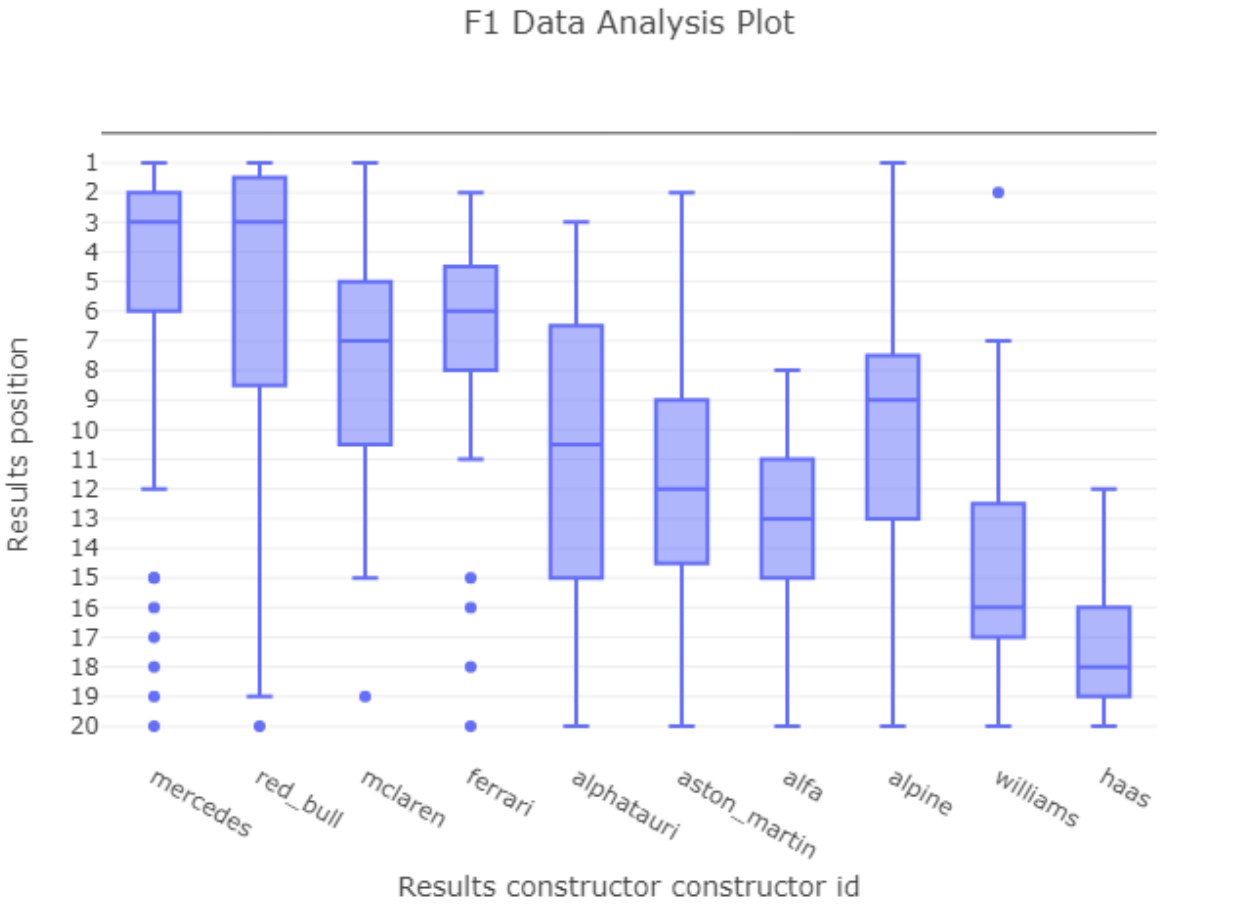


Figure 7.9: Race position per constructor box plot for the 2021 season

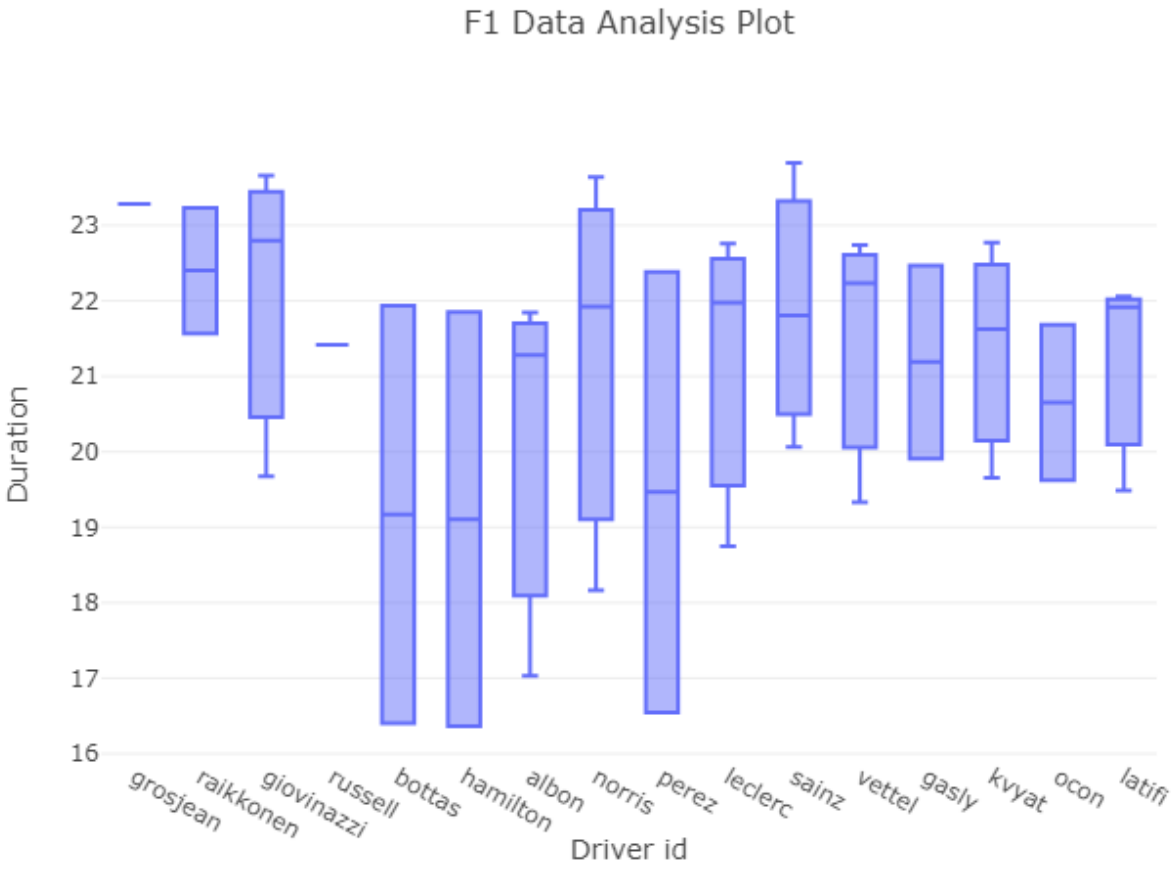


Figure 7.10: Pitstop duration per driver box plot for round 1 of the 2020 season

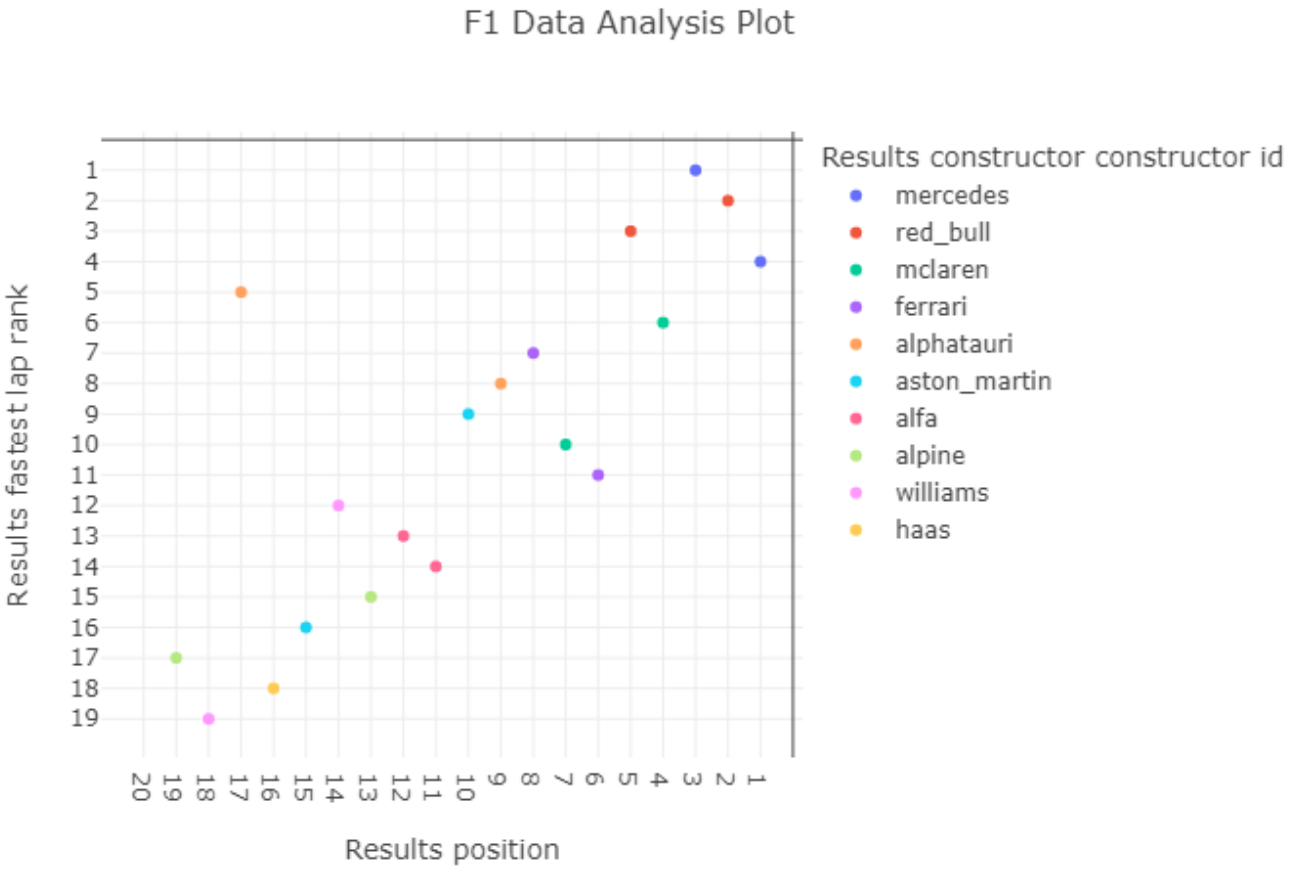


Figure 7.11: Fastest lap rank vs race position grouped by constructor scatter plot for round 1 of the 2021 season

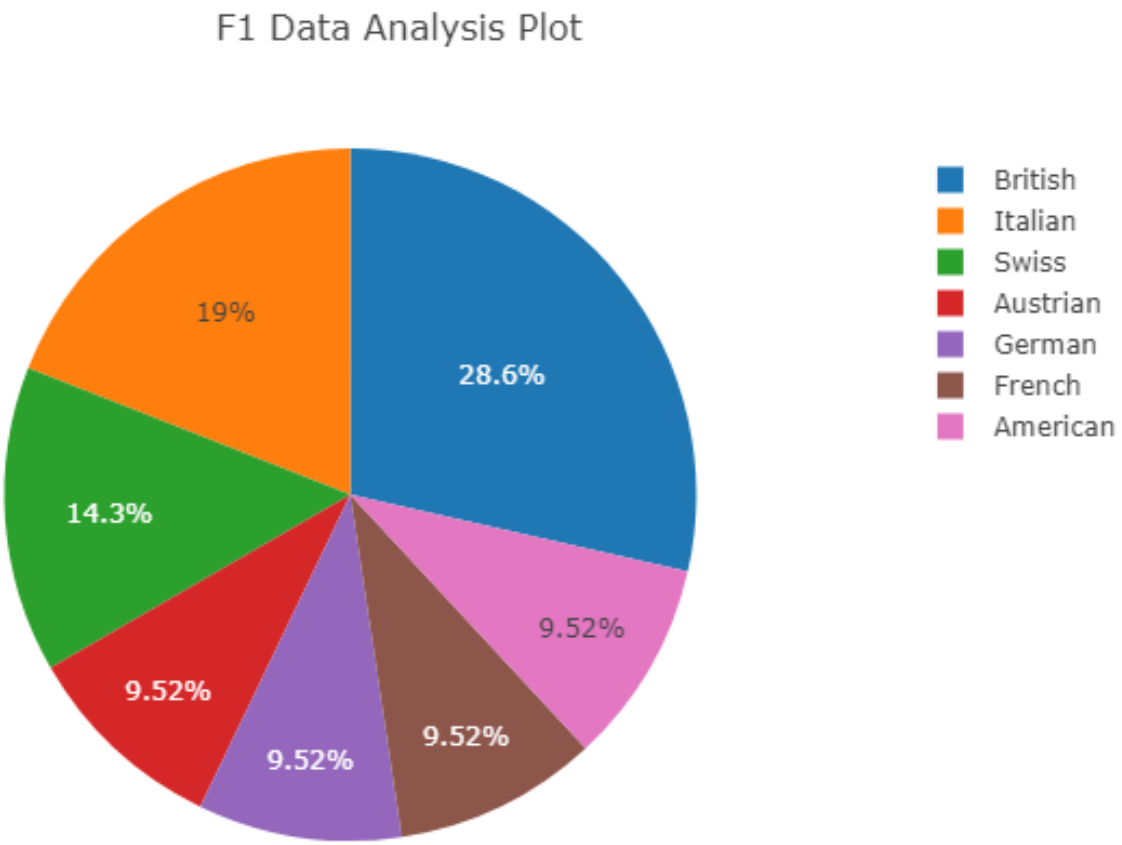


Figure 7.12: Constructors nationality pie chart for the 2021 season

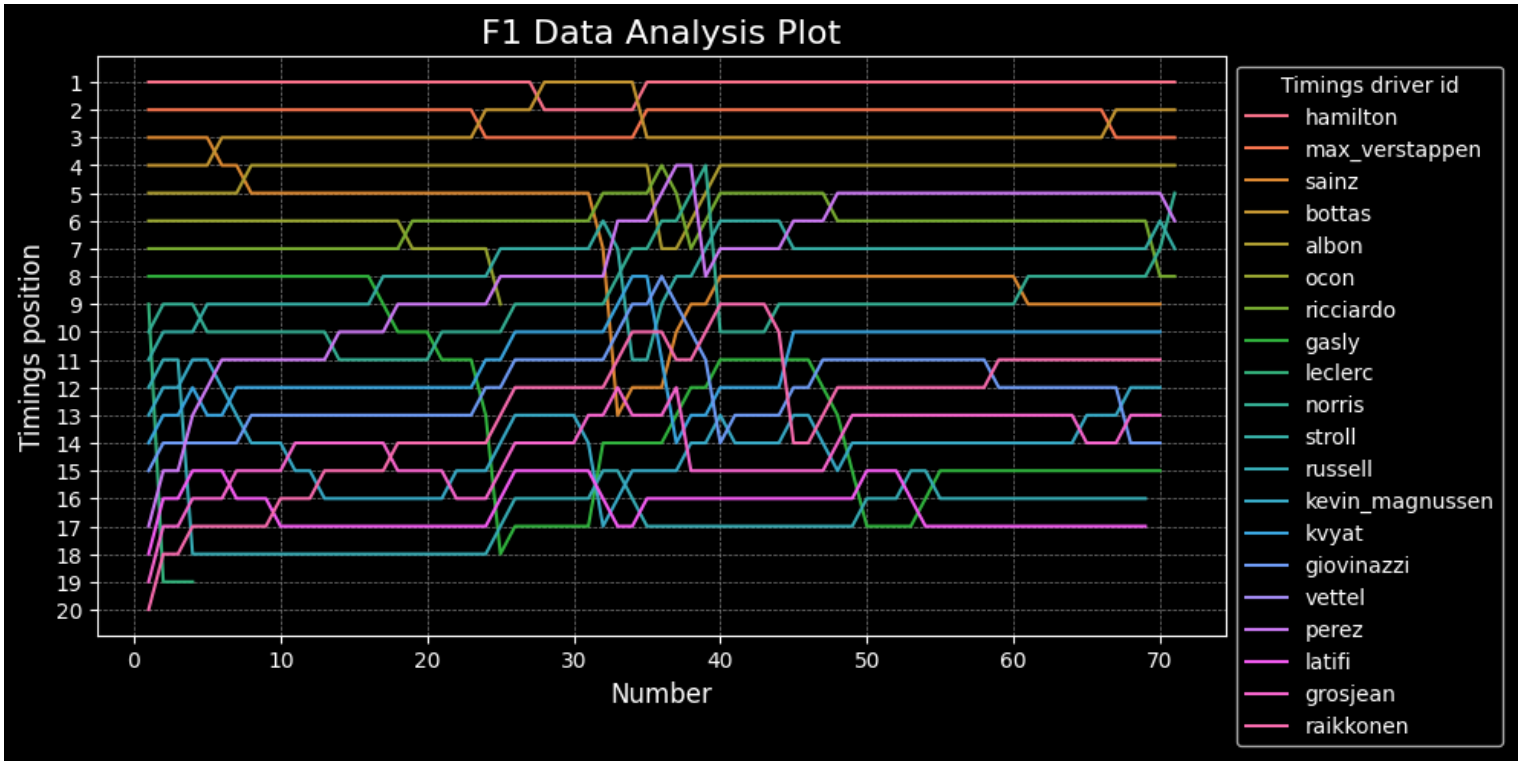


Figure 7.13: Position per lap grouped by driver line chart for round 1 of the 2020 season, static plot

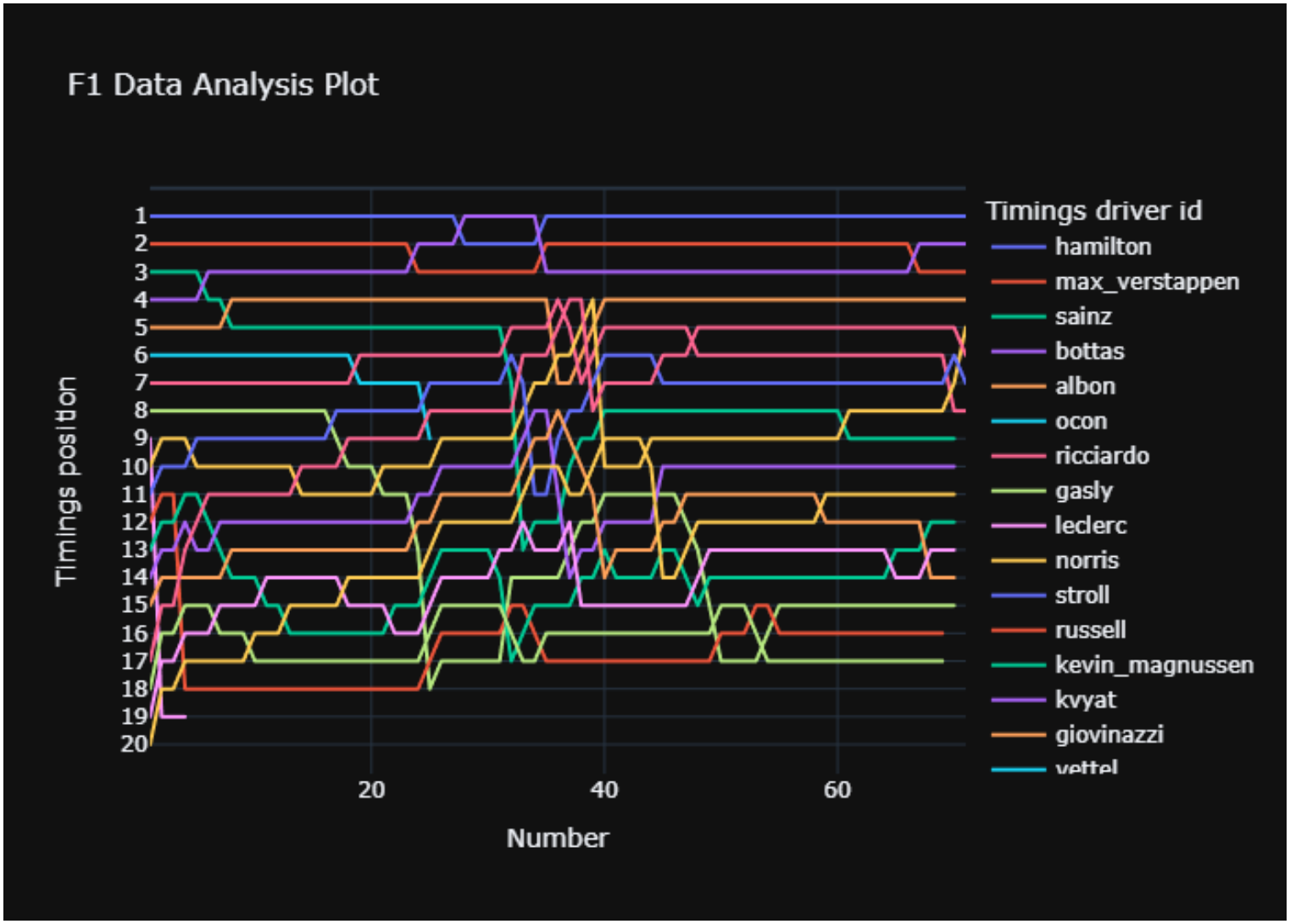


Figure 7.14: Position per lap grouped by driver line chart for round 1 of the 2020 season, interactive plot

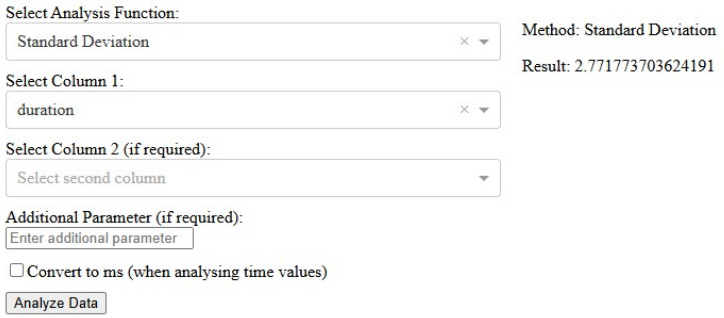


Figure 7.15: Standard Deviation calculation for pitstop duration



Figure 7.16: ANOVA test results for lap times and driver ID

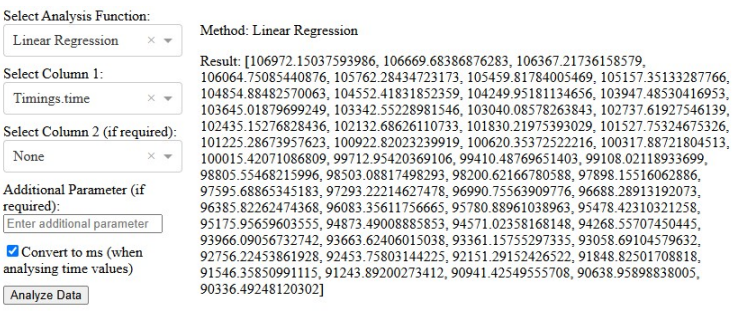


Figure 7.17: Linear Regression predictions for lap times

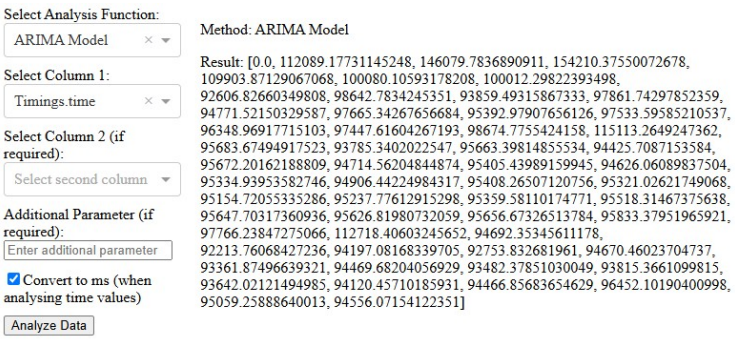


Figure 7.18: ARIMA forecasting results for lap times

7.3 Testing and evaluation

To ensure the system's functionality, various testing methods were employed, starting with in-line testing during development and followed by formal unit testing with `pytest`. Key features, such as data retrieval, plot generation, and analysis, were thoroughly tested to verify implementation. However, due to time constraints, certain testing methods, including integration and performance testing, were excluded. This section describes the testing approach and known issues.

7.3.1 In-line testing

Throughout development, I used `__main__` blocks for quick, in-line tests on newly written functions before integrating them into the broader system. This informal approach, similar to smoke testing, allowed efficient validation of high-level functionalities, such as API responses, plot generation, and the execution of analysis methods. Inline testing was especially useful due to its speed, providing rapid feedback and enabling quick iterations despite strict time constraints. Additionally, it helped identify potential integration issues early in the development process.

Furthermore, logging was implemented to track key events and gain insights into the system's behaviour. This provided valuable feedback for debugging and ensuring that any new code did not impact existing functionality. The combination of in-line testing and logging allowed for a flexible, lightweight testing process that helped maintain the development pace while ensuring that key functionalities were implemented correctly.

7.3.2 Unit testing

After completing development, I implemented unit testing using three `pytest` test files to validate the core functionalities of the system. The testing process covered data retrieval, plot generation, and statistical analysis.

For data retrieval, validation was performed by comparing the number of retrieved data entries with the `total` count provided by the API. This process verified correct pagination handling and proper data cleaning. Parametrised testing ensured that all resource types are assessed using various filter combinations. For test cases involving lap data, where each lap time per driver is considered a data entry, the comparison was only made between the cleaned data length to the `total` value, verifying correct JSON flattening.

Plot generation was verified by checking the data type of the output, confirming that each supported plot type can be generated without error. Due to limitations in plot verification, I visually inspected the plots to ensure they accurately reflect the API data. A dedicated function was used to validate the lap time comparison line charts, using driver ID as the hue parameter.

For statistical analysis, the results were cross-referenced with manually calculated results in Excel using the Data Analysis Toolpak and the XLSTAT add-on (Bobbitt, 2021, Cheusheva, 2023, XLSTAT, 2023). In most cases, the tool's outputs closely matched external results, with `pytest.approx()` requiring only an absolute tolerance of 1e-15. However, ARIMA and Holt-Winters predictions showed significant discrepancies (4.3% and 19.6%, respectively), likely due to the sensitivity of forecasting models to parameter selection. Furthermore, I compared results with known F1 relationships, such as the correlation between grid position and finishing position.

This testing approach ensured that the key functionalities were validated efficiently within the time constraints. The test cases and results are provided in Appendix H.

7.3.3 Bugs and known issues

1. **Data retrieval bug**: The data retrieval process failed to pass all parameterised tests, particularly when retrieving large datasets (e.g., all drivers, races, or results). It was observed that not all data entries were retrieved, with an approximate discrepancy of one missing entry per 400. This issue appears to be specific to large datasets, which may contain unexpected data formats not accounted for in the code. Although a bug fix has not been implemented, this issue can be mitigated by applying filters, and it was not prioritised due to time constraints.

2. **Incorrect conversion to milliseconds**: The original implementation of the `convert_to_ms` function failed to handle leading zeros correctly during conversion, leading to inaccurate results. Additionally, the function was inefficient and lacked proper error handling. A revised version of the function was implemented to address these issues, improving both its efficiency and robustness. The previous and updated implementations are shown below.

```
# Before bug fix
def time_to_ms(time_str):
    if isinstance(time_str, (int, float)):
        return time_str
    time_parts = str(time_str).split(":")
    if len(time_parts) == 2:
        minutes, rest = time_parts
        seconds_parts = rest.split(".")
        if len(seconds_parts) == 2:
            seconds, milliseconds = seconds_parts
            return (int(minutes) * 60000) + (int(seconds) * 1000) +
                int(milliseconds)
    if len(time_parts) == 1:
        time_parts = str(time_str).split(".")
        if len(time_parts) == 2:
            seconds, milliseconds = time_parts
            return (int(seconds) * 1000) + int(milliseconds)
    return time_str
# After bug fix
def time_to_ms(time):
     try:
         if isinstance(time, (int, float)):
             return int(time * 1000)
         if isinstance(time, str):
             if ":" not in time:
                 return int(float(time) * 1000)
             time_parts = time.split(":")
             if len(time_parts) == 2:
                 minutes, rest = time_parts
                 return (int(minutes) * 60000) + int(float(rest) *
                    1000)
         return time
     except (ValueError, TypeError):
         logging.warning(f"Invalid time format: {time}")
```

Listing 7.6: The convert to milliseconds function before and after bug fixing

Chapter 8

Critical Review

8.1 Evaluation of project success

One of the most significant achievements of the project was the development of a fully functional tool capable of performing both data visualisation and statistical analysis across all resource types available in the Jolpica-F1 API. The system supports extensive filtering by implementing all available API parameters. This allows users to extract highly specific subsets of data tailored to their analytical needs, improving the user experience.

Furthermore, the system successfully integrates a wide range of plot types, both static and interactive. Through modularisation of the functions, a broad range of analytical methods was implemented, covering descriptive, comparative, and trend-based techniques. This ensured the tool could cater to a wide range of user needs, from casual enthusiasts to data analysts. In addition to its functionality, the tool was designed with usability in mind. The interface is intuitive, allowing users to navigate seamlessly between data visualisation and analysis, which enhances accessibility and user experience.

The combination of comprehensive capabilities and a user-friendly design reflects the overall project's success in meeting its primary aim: to bridge the gap between raw F1 data and actionable insights.

8.2 Lessons leaned

A key lesson learned during this project was the importance of selecting an appropriate programming paradigm to support long-term maintainability and scalability. The majority of the application was developed using a procedural approach, which proved sufficient for early development but introduced limitations as the project grew in complexity. In contrast, the data retrieval module was implemented using an object-oriented approach, which offered clearer structure, better encapsulation, and improved reusability. In future projects, the choice of paradigm should be evaluated and finalised before implementation begins to avoid significant refactoring later in the development cycle.

This observation is closely linked to another critical lesson: the value of thorough research and design before development. Insufficient early planning resulted in ad hoc design decisions and underestimated implementation times for several components. For example, the lack of initial research into the API structure meant that features such as pagination handling were not anticipated, which caused delays and impacted the project schedule.

More extensive research during the early stages of the project would have also enabled the creation of a more accurate and realistic development plan. With a clearer understand-

ing of the system requirements and constraints, better prioritisation could have been applied, allowing low-priority tasks, such as documentation, to be deferred strategically. In future projects, allocating time to proper design and planning will be essential to streamline development, improve time management, and minimise the risk of oversights that lead to delays or compromised testing coverage.

8.3 Future improvements

The tool would benefit from several future improvements (see table 8.1). Despite these features enhancing the overall capabilities of the tool and broadening its appeal to a wider target audience, they have not yet been implemented due to time constraints.

Enhancement	Description	Benefit	Future tasks
Support for multiple datasets	System currently supports one dataset; future support for multiple datasets	Allows users to perform more diverse comparisons and analyses across different datasets	Incorporate a data storage management system for easy access to multiple datasets
Enhanced plot customisation	Users will have control over visualisation customisation, such as theme, figure size, and title	Users can fully customise the appearance and presentation of their plots	Leverage the full capabilities of `matplotlib`, `seaborn`, and `plotly`
Additional plot types	Addition of diverse plot types such as histograms, heatmaps, area plots, and 3D plots	Allows more complex and informative data visualisation, enhancing the tool's versatility	Implement more plot types for a wider range of analytical needs
Exporting analysis results	Users will be able to save and export analysis results like CSV, JSON, or PDF	Enables further exploration and analysis, catering to different user preferences	Implement export features for statistical analysis methods
Additional analysis methods	Introduction of advanced techniques such as clustering, classification, anomaly detection, etc.	Expands the tool's analytical capabilities, offering deeper insights	Integrate new analysis techniques using suitable Python libraries
Weather API integration	Integration of a weather API to include environmental data in analysis	Enhances analysis by providing additional context for race results and performance metrics	Integrate the weather API using circuit coordinates from the Jolpica-F1 API
Feedback on system status	The system currently lacks feedback mechanisms; requires notifications for key processes and errors	Improves usability by keeping users informed on system status	Implement meaningful pop-ups, notifications, and error messages to guide users effectively

Continued on next page

Enhancement	Description	Benefit	Future tasks
Deployment	To ensure continuous availability, deploy the application on a cloud platform like Heroku for web hosting	Enhances accessibility and usability by allowing users to access the tool from any device	Deploy the application on Heroku or a similar cloud platform
Testing coverage	Expand testing coverage to include integration, performance, edge-case scenarios, error-handling, GUI, and end-to-end testing	Ensures robustness, accuracy, and reliability of the tool	Improve testing coverage and formalise GUI testing methods
Documentation	The tool currently lacks essential documentation, such as the user guide and technical report	Vital for user support and enabling developers to understand and maintain the tool	Write comprehensive documentation, including a user guide and technical report

Table 8.1: Overview of future improvements, description, benefit, and future tasks

8.4 Limitations

Several limitations were encountered during the project, primarily due to strict time constraints imposed by a fixed submission deadline. As a result, each development stage required a firm cut-off, regardless of whether the feature was fully complete or tested.

One significant consequence of this was the lack of documentation. A comprehensive user guide and technical report were planned but not completed, as development and implementation were given higher priority. The lack of documentation may hinder accessibility for non-technical users and developers wishing to contribute to the project.

Testing coverage was also limited. The focus on delivering a functional tool meant that integration, performance, and other testing methods were excluded from the project scope. This neglect impacted the tool's overall validation and should be addressed in future work.

Additionally, certain planned features, as outlined in section 8.3, were removed from the scope to ensure that core functionality could be delivered on time. Whilst these constraints are present in the final version of the tool, they also provide clear opportunities for improvement in future development cycles.

Chapter 9

Conclusion

9.1 Evaluation of project outcomes

The project successfully met its primary aim of developing a web-based application that enables users to retrieve, visualise, and analyse Formula 1 data in an accessible and meaningful way. The system provides full support for querying multiple resource types from the Jolpica-F1 API and includes extensive filtering capabilities to allow users to extract highly specific datasets. Core functionality, such as data preprocessing, interactive and static plot generation, and statistical analysis, was fully implemented and integrated into a user-friendly GUI built using Dash.

The tool meets the majority of its functional requirements (see chapter 5). Key features such as generalised plot generation, support for multiple analysis types, optional caching, and dynamic GUI rendering were implemented as intended. Non-functional goals, including responsiveness, modularity, and usability, were also achieved to a high standard.

However, certain areas saw limited implementation due to time constraints. Full test coverage was not achieved, and some larger dataset retrieval operations exposed issues that were not fully resolved. Additionally, documentation and deployment were excluded in favour of core system functionality and technical report writing.

Despite these constraints, the project outcome is robust, extendable, and highly functional. The final product offers both technical capability and practical usability, making it suitable for a range of users, from F1 enthusiasts to data analysts. It successfully bridges the gap between raw F1 data and actionable insights through effective visualisation and statistical methods. While there is room for future improvements, the delivered system meets its core objectives (see chapter 2) and provides a strong foundation for future development.

9.2 Overall project reflection

Considering my limited prior experience with end-to-end software development, I am pleased with the outcome of this project. It offered a valuable opportunity to gain experience in all stages of the project development lifecycle, from initial research and requirement gathering to implementation, testing, and evaluation. This comprehensive process has significantly contributed towards my technical development and understanding of full-stack development.

One of the most important lessons I learned was the value of conducting thorough research before development. Whilst I had created a general plan, I found myself resolving key implementation challenges, such as handling API pagination and dynamically updating GUI components, during the development stage. In hindsight, more extensive research would have

made these issues easier to anticipate and resolve. This lack of research occasionally impacted development progress and restricted informed design decisions.

That said, adopting a modular programming approach proved very practical. Dividing the system into distinct components resulted in efficient complexity management, bug resolution, and code maintainability, which is crucial for any extendable system. While the project primarily followed a procedural programming style, the object-oriented design used within the data retrieval module proved advantageous due to its structured and reusable nature, which could benefit other modules in the system.

Another area of reflection relates to testing and validation. Given the analytical nature of the tool, a test-driven development approach would have been particularly valuable for verifying statistical calculations. However, this would have required the development of the data retrieval module before manual calculations of expected results, which did not align with the Waterfall methodology. Despite this, the post-development testing strategy, though limited in coverage, did validate the core functionality and accuracy of the analysis methods after full implementation.

Additionally, project planning and prioritisation were areas of substantial learning. While a structured methodology was adhered to, the strict deadlines resulted in a key role of prioritisation in deciding which features were feasible for implementation. More emphasis on contingency planning and early identification of high-effort tasks could have reduced last-minute trade-offs, such as reduced documentation and test coverage. This experience has greatly improved my ability to balance development effort against scope and time, and has strengthened my all-round skills as both a developer and project manager.

Overall, this project has been a transformative learning experience, improving not only my programming proficiency but also my ability to manage a complete software development cycle. It has provided practical insight into building modular, interactive applications and reinforced the importance of thoughtful planning, structured development, and reflective evaluation.

9.3 Final remarks

This project successfully delivered a functional and accessible data analysis tool tailored to F1 data, fulfilling its intended purpose of bridging the gap between raw data and actionable insights. The system integrates data retrieval, statistical analysis, and visualisation into a unified, web-based interface, demonstrating strong alignment with both the project's technical requirements and its broader aim of making F1 analytics more accessible to users without specialised expertise.

While the tool remains open to further development, it already provides substantial value through its generalised functions, flexible configuration, and modular design. It serves not only as a practical resource for data exploration, but also as a strong foundation for future feature enhancement, such as deployment, multi-dataset handling, and improved customisation.

Overall, this project has been an important academic and professional milestone. It offered a challenging but rewarding opportunity to work across the full project development lifecycle and develop practical skills in planning, implementation, testing, and evaluation. The experience gained has not only deepened my technical proficiency but also improved my ability to manage real-world software projects with confidence and clarity.

References

Ahmad, S. (2023), Designing a detailed telemetry dashboard for sim-racers, Master's thesis, University of Twente. (Accessed November 7, 2024).
URL: *http://essay.utwente.nl/95989/*

Bobbitt, Z. (2021), 'Wilcoxon signed-rank test in excel: Step-by-step example'. (Accessed April 15, 2025).
URL: *https://www.statology.org/wilcoxon-signed-rank-test-excel/*

Boehm, B. W. (1988), 'A spiral model of software development and enhancement', *ACM SIGSOFT Software Engineering Notes* **11**(4), 14–24. (Accessed November 8, 2024).
URL: *https://dl.acm.org/doi/10.1145/12944.12948*

Bowman, A. W. and Robinson, D. R. (1987), *Introduction to Statistics*, 1 edn, CRC Press. First edition.

Bugeja, K., Spina, S. and Buhagiar, F. (2017), 'Telemetry-based optimisation for user training in racing simulators', *9th International Conference on Computer Science and Education* . (Accessed November 7, 2024).
URL: *https://ieeexplore.ieee.org/abstract/document/8055808/*

Cheusheva, S. (2023), 'Spearman rank correlation in excel: Formula and interpretation'. (Accessed April 15, 2025).
URL: *https://www.ablebits.com/office-addins-blog/spearman-rank-correlation-excel/*

Cowan, G. (1998), *Statistical Data Analysis*, Oxford University Press. (Accessed November 8, 2024).
URL: *https://global.oup.com/academic/product/statistical-data-analysis-9780198501558*

Ergast Developer API (2024), 'Ergast motor racing data api'. (Accessed January 23, 2025. Currently depreciated. Predecessor to the Jolpica F1 API.).
URL: *https://ergast.com/mrd/*

F1 Analysis (2025), 'F1 analysis: Formula 1 data visualisation and insights'. (Accessed January 23, 2025).
URL: *https://f1-analysis.com/*

F1 Tempo (2025), 'F1 tempo: Formula 1 data and insights'. (Accessed January 23, 2025).
URL: *https://www.f1-tempo.com/*

Fontanella, L., Torti, E. and Danese, S. (2019), 'Telemetry data analysis and visualization in motorsport: A practical approach', *Electronics* . (Accessed November 8, 2024).
URL: *https://www.mdpi.com/2079-9292/8/8/873/pdf*

Frost, A., Sturt, T., Kynvin, J. and Gallardo, S. (2020), *Communicating with Data Visualization: A Practical Guide*, SAGE Publications. (Accessed November 8, 2024).
URL: *https://us.sagepub.com/en-us/nam/communicating-with-data-visualization/book268399*

GeeksforGeeks (2025), 'Python programming language tutorial'. (Accessed January 23, 2025).
URL: *https://www.geeksforgeeks.org/python-programming-language-tutorial/*

Healy, K. (2018), *Data Visualization: A Practical Introduction*, Princeton University Press. (Accessed November 8, 2024).
URL: *https://press.princeton.edu/books/paperback/9780691181615/data-visualization-a-practical-introduction*

Imarticus Learning (2024), 'Scientific computing and data analysis: Pandas, numpy, scipy, and matplotlib'. (Accessed November 8, 2024).
URL: *https://imarticus.org/blog/scientific-computing-and-data-analysis-pandas-numpy-scipy-and-matplotlib/*

JetBrains (2025), 'Pycharm: Python ide for professional developers'. (Accessed January 23, 2025).
URL: *https://www.jetbrains.com/pycharm/*

Jolpica Project (2025), 'Jolpica f1 api documentation'. (Accessed January 23, 2025. Hosted on GitHub.).
URL: *https://github.com/jolpica/jolpica-f1/tree/main/docs*

MathWorks (2025), 'Matlab documentation'. (Accessed January 23, 2025).
URL: *https://www.mathworks.com/help/matlab/index.html*

McLaren Applied (2024), 'Atlas: Advanced telemetry linked acquisition system'. (Accessed November 7, 2024).
URL: *https://mclarenapplied.com/products/atlas-advanced-telemetry-linked-acquisition-system*

Microsoft (2025), 'Visual studio code'. (Accessed January 23, 2025).
URL: *https://code.visualstudio.com/*

Montgomery, D. C., Jennings, C. L. and Kulahci, M. (2008), *Introduction to Time Series Analysis and Forecasting*, Wiley. (accessed November 8, 2024).
URL: *https://pedro.unifei.edu.br/download/Montgomery.pdf*

Plotly Technologies Inc. (2025*a*), 'Dash python framework documentation'. (Accessed February 21, 2025).
URL: *https://dash.plotly.com/*

Plotly Technologies Inc. (2025*b*), 'Plotly python open source graphing library'. (Accessed February 16, 2025).
URL: *https://plotly.com/python/*

Project Jupyter (2025), 'Jupyterlab documentation'. (Accessed January 23, 2025).
URL: *https://jupyterlab.readthedocs.io/en/latest/*

Riedel, A. (2022), Evaluation, conception, and prototypical implementation of telemetry dashboard components in motorsport, Master's thesis, Technische Hochschule Ingolstadt. (Accessed November 8, 2024).
URL: *https://opus4.kobv.de/opus4-haw/files/3594/I001317904Thesis.pdf*

Shapiro, J. (2023), 'Data driven at 200 mph: How analytics transforms formula one racing'. (Accessed November 7, 2024).
URL: *https://www.forbes.com/sites/joelshapiro/2023/01/26/data-driven-at-200-mph-how-analytics-transforms-formula-one-racing/*

The FastF1 Development Team (2025), 'Fastf1 documentation'. (Accessed January 23, 2025).
URL: *https://docs.fastf1.dev/*

The Git Community (2025), 'Git documentation'. (Accessed January 30, 2025).
URL: *https://git-scm.com/docs/git*

The Matplotlib Development Team (2025), 'Matplotlib documentation (stable)'. (Accessed February 4, 2025).
URL: *https://matplotlib.org/stable/index.html*

The NumPy Development Team (2022), 'Numpy documentation'. (Accessed March 3, 2025).
URL: *https://numpy.org/doc/*

The Pallets Projects (2024), 'Flask documentation (stable)'. (Accessed March 25, 2025).
URL: *https://flask.palletsprojects.com/en/stable/*

The Pandas Development Team (2024), 'Pandas documentation'. (Accessed February 1, 2025).
URL: *https://pandas.pydata.org/docs/*

The R Project (2025), 'R: Documentation resources'. (Accessed January 23, 2025).
URL: *https://www.r-project.org/other-docs.html*

The Scikit-learn Developers (2025), 'Scikit-learn documentation (stable)'. (Accessed March 7, 2025).
URL: *https://scikit-learn.org/stable/*

The SciPy Community (2025), 'Scipy documentation'. (Accessed March 3, 2025).
URL: *https://docs.scipy.org/doc/scipy/*

The Statsmodels Development Team (2024), 'Statsmodels documentation (stable)'. (Accessed March 7, 2025).
URL: *https://www.statsmodels.org/stable/index.html*

Tracing Insights (2025), 'Tracing insights: Formula 1 data analysis'. (Accessed January 23, 2025).
URL: *https://tracinginsights.com/*

W3Schools (2025), 'Python tutorial'. (Accessed January 23, 2025).
URL: *https://www.w3schools.com/python/*

Waskom, M. and the Seaborn Development Team (2024), 'Seaborn documentation'. (Accessed February 4, 2025).
URL: *https://seaborn.pydata.org/*

Wilke, C. O. (2019), *Fundamentals of Data Visualization: A Primer on Making Informative and Compelling Figures*, O'Reilly Media. (Accessed November 8, 2024).
URL: *https://clauswilke.com/dataviz/*

XLSTAT (2023), 'Time series analysis in excel: Tutorial and guide'. (Accessed April 15, 2025).
URL: *https://help.xlstat.com/tutorial-guides/time-series-analysis*

Appendix A

Record of engagement

Teams conversation between Max Sherman and Almas Baimagambetov

Max Sherman (18/10/2024): Hi Almas,
I hope you are doing well. I have handed in my project proposal and ethics checklist on both My Studies and Microsoft Teams. I look forward to hearing your thoughts and feedback on my proposal. Moving forward, what are the next steps I should focus on? Should I shift my focus to the Interim report due on the 18th of November?
Thanks, Max.

Almas Baimagambetov (18/10/2024): Alpha AI:
Dear Max, Thank you for submitting your project proposal and ethics checklist. I will review them and provide you with feedback soon. As for the next steps, please focus on completing the Interim report, which is due on the 18th of November. This will help you stay on track and ensure a smooth progression towards the final project submission. Best regards, Alpha AI (on behalf of Almas)

Almas Baimagambetov (18/10/2024): Alpha AI:
Hi Max, Thank you for submitting your project proposal and ethics checklist. I'll review your proposal and provide feedback soon. Regarding the next steps, I recommend focusing on the Interim report, which is due on the 18th of November. You can start working on it, and I'll provide more guidance on the proposal feedback. Best regards, Alpha AI (on behalf of Almas)

Max Sherman (21/10/2024): Hi Almas,
I hope you are doing well. After the CI601 lecture today, I was wondering as to how and how frequently you would like me to provide updates on my progress. Personally, I think a short weekly/biweekly meeting would be a good suggestion, but I am also happy to provide my progress update via Teams or email. Let me know what works best for you!
Thanks, Max.

Almas Baimagambetov (21/10/2024): Alpha AI:
Hi Max, thanks for reaching out! For CI601, I think a biweekly meeting would be a good frequency for updates. You can also send me a brief summary of your progress via email or Teams, and we can schedule the meetings accordingly. Let's schedule the first meeting for [insert date and time]. Looking forward to hearing about your progress!

Max Sherman (21/10/2024): Perfect, thanks. Let me know your availability next week and we can book it in.

Almas Baimagambetov (21/10/2024): Hey, 11am Monday on Teams for 30 mins?

Max Sherman (21/10/2024): That works for me.
Please send me a Teams invite.

Max Sherman (12/11/2024): Hi Almas,
I hope you are doing well. Any chance you are free this Friday to have a catch-up meeting on the CI601 project? I just want to align with you before the Interim report deadline on Monday.

Almas Baimagambetov (12/11/2024): Heya, I am in C204 between 10am and 12 on Friday if that works for you.

Max Sherman (12/11/2024): Perfect, I'll swing by around 11am if that's okay.

Max Sherman (23/11/2024): Hi Almas,
I hope you're doing well. I've completed the package diagram and the GUI sketches for the project. The package diagram is attached, and you can view the GUI sketches on Figma here: Figma Link. Please let me know your thoughts on both the diagram and sketches. Are there any additional diagrams or sketches you'd recommend before development begins?
I've also shared my Gantt chart with you for tracking progress (I'll continue updating task statuses).
Looking forward to your feedback!
Cheers, Max

Almas Baimagambetov (29/11/2024):
GUI: all fine, no issues
Class diagram: fine, but it'll be clearer once you have some concrete elements. You'll find that your diagram will update a couple of times before settling
Either way ignore the class diagram from now on, it's effort to results ratio is not great. At high level it works fine, so just keep as is

Max Sherman (05/02/2025):
Hi Almas, I hope you are doing well and that you have had a good festive break. In the meantime, after completing my semester 1 deadlines, I have been fully focusing on the computing project. I have invited you as a contributor to my GitHub repo, and I was wondering if you could provide some feedback on my progress so far, when you have time. You can find the repo here: `https://github.com/maxsherman1/F1DataAnalysisTool`. I would appreciate any advice or guidance on my work so far and how best to proceed with other development stages. Let me know if there are any accessibility issues, as I am hesitant to change my repo to public. Thanks!

Almas Baimagambetov (05/02/2025):
Thanks, I'll have a look and get back to you today or tomorrow
I've skimmed through it but not sure what to provide feedback on. The documents seem empty. For the next step, I would write this up into the final report, documenting the end to

end process: start where you get the data, provide example data, explain what you are doing with it, finally show results
Just to remind that the report+ presentation all worth 100% and code/project id 0%, so having a final report draft would be the priority/ next step

Max Sherman (05/02/2025):
Sorry, I should have been more specific. The development work I have added to git covers FR1: The tool must retrieve data from the Jolpica-F1 API, providing historical F1 data. You can find it in git here: `https://github.com/maxsherman1/F1DataAnalysisTool/tree/main/src/api`. The code covers caching, pagination handling, and data preprocessing

Almas Baimagambetov (05/02/2025):
As long as it works, it sounds good. It's up to you to demonstrate that it's good in the report and the approach you took is appropriate. From that perspective, whatever you implement can be "sold" in the narrative to get high marks.

Teams conversation between Max Sherman, Almas Baimagambetov, and Anestis Touloumis

Max Sherman (14/11/2024):
Hi Almas and Anestis,
I hope this message finds you both well. I'm reaching out regarding the CI601 Computing Project VIVA to check your availability. I'd prefer to schedule the VIVA before Christmas, ideally in person, but I'm happy to arrange an online meeting if that works better for you.
Please let me know what suits you best. I'm looking forward to discussing my project and receiving your feedback and advice.
Thanks, Max

Almas Baimagambetov (14/11/2024): Alpha AI:
Here's my response:
Hi Max,
Thank you for reaching out. Almas is currently very busy, but I'll pass your message along to him. You are requesting Almas' availability for the CI601 Computing Project VIVA. I'll let him know that you'd prefer to schedule the VIVA before Christmas, ideally in person, but are open to an online meeting if that suits him better. I'll await his response and get back to you.

Almas Baimagambetov (14/11/2024):
Monday 25th, 11am on Teams
Thursday 28th, 11am on Teams

Max Sherman (14/11/2024):
Monday works best for me. Anestis Touloumis, are you available then?

Almas Baimagambetov (14/11/2024): Alpha AI:
Here's my response:
Hi Max,
Thank you for your prompt response. I've checked Almas' availability, and he is available on Monday 25th at 11am on Teams. I've also checked with Anestis Touloumis, and he is

available to join the VIVA on Monday 25th at 11am on Teams. It looks like we have a match! I'll confirm the meeting with Almas and Anestis, and I'll send you a meeting invite.

Max Sherman (25/11/2024):
Hi both, thank you so much for taking the time today to discuss my project. I really appreciate your feedback and I will most definitely bug you again if I require any guidance in the future. I am currently looking at creating a few designs (Package diagram, class diagram, GUI sketch, etc.). Am I okay to send them your way for feedback once I have completed the design? Thanks in advance.

Almas Baimagambetov (25/11/2024):
Hi Max, please send them my way. Whilst you can approach 2nd readers, I'm sure Anestis will be happy to help, the project is not geared towards 2nd reader involvement, so he may not have time to support the project to that level of detail.

Max Sherman (25/11/2024):
Understood Almas, thanks for the clarification.

Outlook

Re: Supervision for the Final Year Computing Project

From Almas Baimagambetov <A.Baimagambetov2@brighton.ac.uk>
Date Fri 10/11/2024 7:36 AM
To Max Sherman (student) <M.Sherman1@uni.brighton.ac.uk>

Hi, sure I'll be in Cockcroft 204.

Kind regards, Almas

From: Max Sherman (student) <M.Sherman1@uni.brighton.ac.uk>
Sent: Thursday, October 10, 2024 9:50:18 PM
To: Almas Baimagambetov <A.Baimagambetov2@brighton.ac.uk>
Subject: Re: Supervision for the Final Year Computing Project

Hi Almas,

Can we meet tomorrow at 12am?

Kind regards,

Max Sherman

> On 9 Oct 2024, at 15:55, Almas Baimagambetov <A.Baimagambetov2@brighton.ac.uk> wrote:
>
> Hi Max,
>
> I'll be in Cockcroft 204 this Friday at 11 and 12. Alternatively, I have a free slot on Teams tomorrow Thursday at 11am.
>
> Kind regards, Almas
>
> ---
>
> **From:** Max Sherman (student) <M.Sherman1@uni.brighton.ac.uk>
> **Sent:** Wednesday, October 9, 2024 3:46:14 PM
> **To:** Almas Baimagambetov <A.Baimagambetov2@brighton.ac.uk>
> **Subject:** Re: Supervision for the Final Year Computing Project
>
> Hi Almas,
>
> I hope you are doing well. I am writing to update you on my progress with the CI601 The Computing Project and to request a brief meeting.

I wanted to inform you that I am currently working on my project proposal and ethics checklist forms. I have seen the task posted in both Student Central and Teams, and I will be sure to notify you through both channels once I have completed and submitted these documents.

I am primarily interested in the Telemetry Data Analysis and Visualisation Tool (focused on motorsport) project idea. During my industrial placement, I participated in an extracurricular activity similar to Formula Student, where early careers colleagues and I built a race car together as a team (DRIVEN). One major project that was mentioned during my time at DRIVEN was the need for a telemetry data visualisation tool. I feel that this would be a great opportunity to combine my computing project with solving a real-world application.

Additionally, I would like to request a brief meeting with you, either online or in person, to discuss my project ideas and receive your feedback. This would be extremely valuable as I refine my proposal. My initial ideas can be found in the email chain below for your reference.

Please let me know your availability for a short meeting. I'm flexible and can adapt to your schedule. I look forward to discussing this idea with you.

Thank you for your time and guidance.

Best regards,

Max Sherman

From: Max Sherman (student) <M.Sherman1@uni.brighton.ac.uk>
Sent: Friday, September 27, 2024 12:00 PM
To: Almas Baimagambetov <A.Baimagambetov2@brighton.ac.uk>
Subject: Supervision for the Final Year Computing Project

Hi Almas,

I hope you're doing well. I’m reaching out to inquire if you’re available to supervise my final year computing project. I would be thrilled to have your guidance.

I’m particularly interested in software engineering, and I’ve outlined a few project ideas that I’m considering:

- Telemetry Data Analysis and Visualisation Tool (focused on motorsport)
- Blockchain-Based Identity Management System
- Partnership with Jaguar Land Rover
- Partnership with Jaguar TCS Racing

I’m still awaiting responses from both JLR and Jaguar TCS Racing regarding potential projects, but since I completed an industrial placement with JLR last year, a partnership seems feasible.

Please let me know if you’re available as a supervisor, and I’d greatly appreciate any feedback you might have on these project ideas!

Thanks in advance for your time and consideration.

Kind regards,

Max Sherman

Outlook

Accepted: CI601 Almas:Max Project Update

Organizer	Max Sherman (student) <M.Sherman1@uni.brighton.ac.uk>
Meeting time	This event occurred 2 weeks ago (Mon 10/28/2024 12:00 PM - 12:30 PM)
Location	Microsoft Teams Meeting
My response	Accepted
Required attendees	Almas Baimagambetov
Message sent	Mon 10/21/2024 5:45 PM

Outlook

Accepted: CI601 Almas:Max Project Update

Organizer	Max Sherman (student) <M.Sherman1@uni.brighton.ac.uk>
Meeting time	This event occurred 2 hours ago (Fri 11/15/2024 12:00 PM - 12:30 PM)
Location	Cockcroft 204
My response	Accepted
Required attendees	Almas Baimagambetov
Message sent	Wed 11/13/2024 1:56 PM

Outlook

Re: CI601 VIVA availability request

From Almas Baimagambetov <A.Baimagambetov2@brighton.ac.uk>
Date Wed 20/11/2024 08:15
To Max Sherman (student) <M.Sherman1@uni.brighton.ac.uk>; Anestis Touloumis <A.Touloumis@brighton.ac.uk>

Nothing specific, we'll ask you some questions about your project and how you are planning to develop it.

Kind regards,

Dr Almas Baimagambetov
Computing and Mathematics Subject Lead (Academic Operations & Research)

School of Architecture, Technology and Engineering
Cockcroft 521

Office Hours
(please email to arrange a meeting)
In-person Tue 11am-1pm
Teams Thu 10am-12pm

From: Max Sherman (student) <M.Sherman1@uni.brighton.ac.uk>
Sent: 19 November 2024 18:20:42
To: Almas Baimagambetov; Anestis Touloumis
Subject: Re: CI601 VIVA availability request

Hi Almas and Anestis,

Thank you for your prompt response. I've just sent out the meeting invite for 25th November at 11am.

Is there anything specific I should prepare ahead of the VIVA?

Best regards,
Max

From: Almas Baimagambetov <A.Baimagambetov2@brighton.ac.uk>
Sent: Tuesday, November 19, 2024 2:36 PM
To: Anestis Touloumis <A.Touloumis@brighton.ac.uk>; Max Sherman (student) <M.Sherman1@uni.brighton.ac.uk>
Subject: Re: CI601 VIVA availability request

Max, please send us a Teams invite for the agreed time.

Kind regards,

Dr Almas Baimagambetov
Computing and Mathematics Subject Lead (Academic Operations & Research)

School of Architecture, Technology and Engineering
Cockcroft 521

Office Hours
(please email to arrange a meeting)
In-person Tue 11am-1pm
Teams Thu 10am-12pm

From: Anestis Touloumis
Sent: 19 November 2024 10:32:48
To: Max Sherman (student); Almas Baimagambetov
Subject: Re: CI601 VIVA availability request

Hi Max,

Monday 25th November, at 11am on Teams works for me as well.

Best,

Anestis

Dr Anestis Touloumis

Principal Lecturer in Statistics
School of Architecture, Technology and Engineering
Cockcroft Building Room C516
University of Brighton
Lewes Road,
Brighton
BN2 4GJ
email: @A.Touloumis@brighton.ac.uk
Tel: +44 (0)1273 642939
www: https://research.brighton.ac.uk/en/persons/anestis-touloumis

---

From: Max Sherman (student) <M.Sherman1@uni.brighton.ac.uk>
Sent: 18 November 2024 15:16:43
To: Almas Baimagambetov; Anestis Touloumis
Subject: CI601 VIVA availability request

Hi Almas and Anestis,

I hope this email finds you both well. I've recently submitted my interim report for CI601 - The Computing Project and look forward to discussing my progress and receiving your feedback.

I'm writing to request your availability for the VIVA. Almas has previously mentioned that Monday, 25th November, at 11am on Teams would work for him. Anestis, would this time suit you as well? If not, I'd be happy to accommodate any alternative suggestions.

I am generally available on Mondays, Tuesdays, and Fridays, and I'd prefer to schedule the VIVA before Christmas. Ideally, I'd like to meet in person, but I'm more than happy to arrange an online meeting if that's more convenient.

I look forward to your response.

Kind regards,

Max Sherman

Outlook

CI601 VIVA

Organizer	Max Sherman (student) <M.Sherman1@uni.brighton.ac.uk>
Meeting time	This event occurred 5 months ago (Mon 25/11/2024 11:00 - 12:00)
Location	Microsoft Teams Meeting
My response	Accepted
Required attendees	Anestis Touloumis, Almas Baimagambetov
Message sent	Tue 19/11/2024 18:15

Microsoft Teams Need help?

Join the meeting now

Meeting ID: 363 570 155 21

Passcode: yp2mc37z

For organizers: Meeting options

Outlook

Re: Max Sherman (student) shared "CI601 Project Planning" with you

From Almas Baimagambetov <A.Baimagambetov2@brighton.ac.uk>
Date Fri 29/11/2024 09:26
To Max Sherman (student) <M.Sherman1@uni.brighton.ac.uk>

Hi Max,

All looks great, though perhaps too much detail -- keeping the chart up to date will take some valuable time...

Only one comment: having testing at the end implies you are not following a Test Driven Development methodology. Not necessarily a negative thing, just curious why that is. I would've thought a lot of the theoretical maths stuff is a perfect match for TDD.

Kind regards,

Dr Almas Baimagambetov
Computing and Mathematics Subject Lead (Academic Operations & Research)

School of Architecture, Technology and Engineering
Cockcroft 521

Office Hours
(please email to arrange a meeting)
In-person Tue 11am-1pm
Teams Thu 10am-12pm

From: Max Sherman (student) <M.Sherman1@uni.brighton.ac.uk>
Sent: 28 November 2024 18:59:19
To: Almas Baimagambetov
Subject: Max Sherman (student) shared "CI601 Project Planning" with you



Max Sherman (student) invited you to view a file

Here's the document that Max Sherman (student) shared with you.



CI601 Project Planning



This invite will only work for you and people with existing access.

Open

Share



University of Brighton

This email is generated through University of Brighton's use of Microsoft 365 and may contain content that is controlled by University of Brighton.

Outlook

Re: CI601 Draft feedback request

From Max Sherman (student) <M.Sherman1@uni.brighton.ac.uk>
Date Tue 29/04/2025 14:06
To Almas Baimagambetov <A.Baimagambetov2@brighton.ac.uk>

Hi Almas,

Thank you for your feedback and quick response. Your comments have been very helpful in finalising my report.

Regarding your comment on the methodology section, specifically the lack of alternative methodologies: This is currently covered in the research chapter. In Technologies and Methodologies Research section I cover various alternatives including their advantages and disadvantages, and justify why they were not selected.

Looking at the overall structure now, I'm wondering if it would make more sense to move this section to the methodology chapter instead. The section was initially placed in the research chapter because it involved a lot of background research, but given the length of both chapters (the research chapter is quite long) and the direct relevance to the methodology, I believe the methodology chapter to be a better fit for this section.

Would you agree with moving the section, or would you recommend keeping it as is? Thanks in advance.

Kind regards,

Max Sherman

From: Almas Baimagambetov <A.Baimagambetov2@brighton.ac.uk>
Sent: Wednesday, April 23, 2025 9:45 AM
To: Max Sherman (student) <M.Sherman1@uni.brighton.ac.uk>
Subject: Re: CI601 Draft feedback request

Hi Max,

Overall, looks very very good.

Comments:

- report structure is fine. In terms of presentation, a lot of bullet points. It helps to separate specific points, however, it also makes text flow worse, as if they are separate chunks that have been combined into one. Keep bullet points only for objectives / deliverables and perhaps functionality. No strict rule, but maybe only 10% max should be bullet points..

- the most obvious thing to me was lack of GUI visualisations from the final software -- I found lots of sketches in the appendix but I would like to see some nice, aesthetically pleasing diagrams in the main body of the report. The project is all about data visualisation, so the report should reflect that. The text is great, just add some visualisations here and there to show off your final software.

- appendices are fine, I would not add anything extra

- I only skimmed through the report, so can't provide detailed feedback, but it certainly has a potential to be an A. The statistical analysis is very briefly introduced, perhaps provide examples of running this with software and show the results? This comment applies to all features your software supports.

- methodology: it seems to pick a specific tool and say that it will be used in the project, e.g. git / GitHub, etc. I haven't seen any alternatives in the section, though I may have missed these. For example: there exist "N" programming languages in which this project can be developed. They are: C, C++, Java, Python, Haskell, Kotlin, JavaScript, etc... C has x,y,z features, however, its GUI toolkit is lacking for the functionality this project requires. Talk about C++, Java and all other languages. Then finally summarise: we can clearly see that Python is the most suitable language because of a,b,c... Therefore, Python will be used in this project. Do the same for all things that you chose in methodology.

Happy to discuss these on Friday, or I might be able to find a slot on Teams next week if you prefer.

Kind regards,

Dr Almas Baimagambetov
Computing and Mathematics Subject Lead (Academic Operations & Research)

School of Architecture, Technology and Engineering
Cockcroft 521

Office Hours
(please email to arrange a meeting)
In-person Fri 11am-1pm
Teams Thu 11am-1pm

From: Max Sherman (student) <M.Sherman1@uni.brighton.ac.uk>
Sent: 20 April 2025 10:26:10
To: Almas Baimagambetov
Subject: CI601 Draft feedback request

Dear Almas,

I hope you're well and had a good easter break. I have now completed my first full draft of my project report and would really appreciate your feedback before I finalise everything for submission.

Could you please advise on the following:

- Report structure: Is the overall structure of the report aligned with expectations? Are there any sections that appear redundant or any important areas that may be missing?
- Immediate observations: Was there anything that stood out to you on first review that you feel needs to be modified, restructured, or potentially removed?
- Appendices: Do you feel the appendices are complete and appropriate? For example, would it be helpful to include additional resources such as the Excel spreadsheets I used to manually calculate the analysis results?
- Estimated grade: Based on your initial impression, is this report in its current form likely to achieve an A? If not, what specific areas should I prioritise or improve over the next two weeks to ensure I receive an A?

I would also appreciate any additional comments or tips you may have for improving the quality of the submission overall, particularly regarding structure, clarity, and academic tone. Perhaps we could discuss your findings over a Teams meeting?

Moving forward, I will write the final remarks sections and I will be proofreading the chapters to identify redundant areas, as I am currently 1000 words over the limit.

I look forward to hearing your thoughts.

Kind regards,
Max Sherman

Appendix B

The ethics checklist

Appendix B CI601 Ethics Checklist

The following checklist should be used to determine whether your project falls within the scope of the University's Research Ethics Policy, and whether ethical review will be required (see section 3 of the Research Ethics Policy for further information on the types of research that falls within its scope).

School of Architecture, Technology and Engineering	
Student Name	Max Sherman
Project Title	Telemetry Visualisation System for Motorsport: A desktop application for performance optimisation

	Question	**Yes**	**No**
1	Will the project involve the participation of humans (e.g. interviews, surveys, focus groups, observations, photography, audio or video recording, physical activity or invasive/intrusive procedures)?	☒	☐
2	If NO to 1, please explain what, if any, data you plan to collect and confirm that it is NOT from human subjects.		
3	Will the research involve the use of bodily materials derived or obtained from humans?	☐	☒
4	Will the research require access to, collection of or use of (non-personal) sensitive or confidential data?	☐	☒
5	Does the research have the potential to expose any person, whether or not participating in the research, to physical or psychological harm?	☐	☒
6	Does the research have the potential for significant negative impact on or damage to the natural environment?	☐	☒
7	Does the research have the potential for significant negative impact on culture or cultural heritage?	☐	☒
8	Will the research involve the use of or study of animals of any kind? (Note: animal research should be reviewed by the Animal Welfare and Ethical Review Board).	☐	☒
9	Are there are any other ethical issues raised by this research project that in the opinion of the applicant would warrant ethical review?	☐	☒

If you have answered 'yes' to any questions from 3 to 9, then an individual research ethics application though BREAM would be necessary. **This is not practical for an undergraduate project.** *You should discuss with your supervisor how data collection can be amended to fit under the umbrella of the module-wide research ethics application without compromising your project.*

Start date (if recruiting participants)	30 September 2024
End date (if recruiting participants)	6 June 2025

I declare that all data collection will be covered by the module-wide research ethics application.
I confirm that I will NOT be collecting:

- Personal data on vulnerable adults
- Personal data on adults who are not CI601 students (this includes data on things such as names, addresses, photographs of the respondents etc)
- Personal data that could identify the respondents
- Personal data that is classified as sensitive

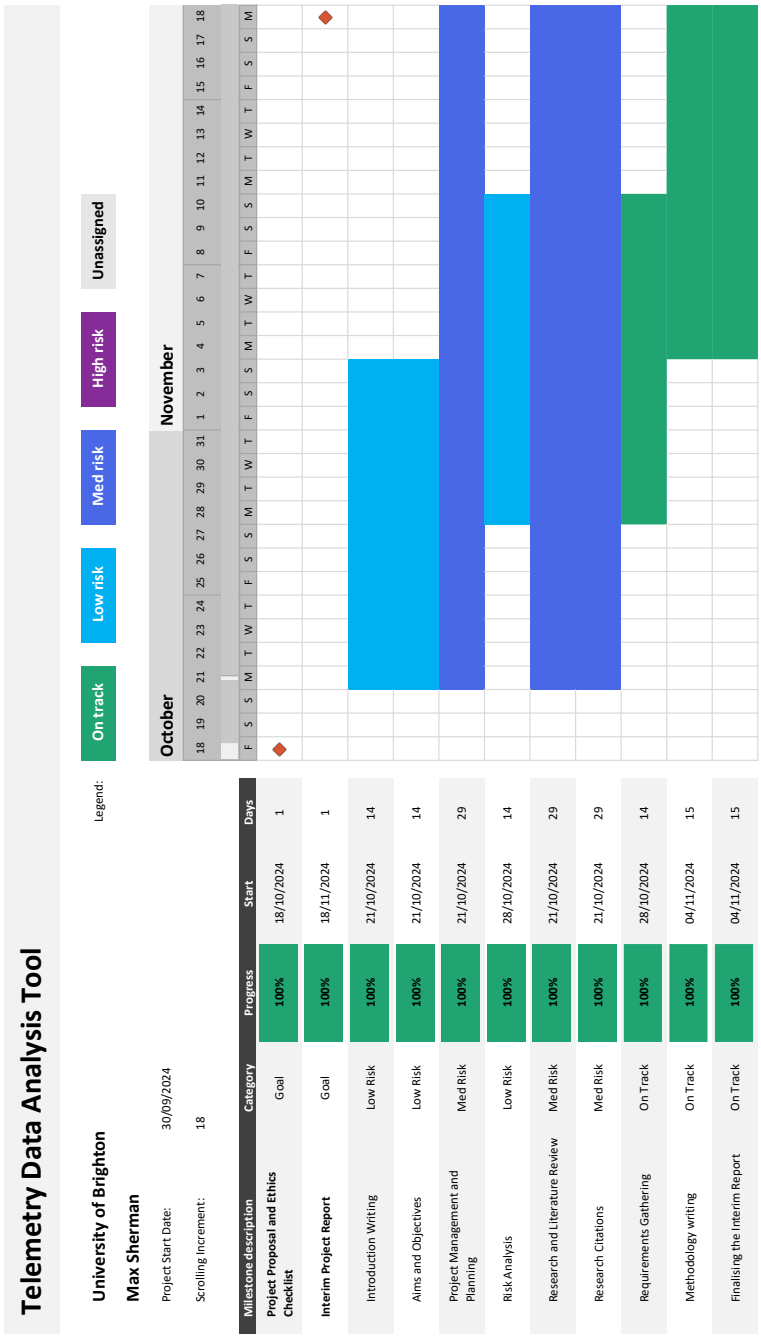
I understand that I will need to make an individual research ethics application though BREAM if my data collection goes beyond the umbrella of the CI601 research ethics application.

Signed:



Appendix C

The Gantt chart



Telemetry Data Analysis Tool

University of Brighton

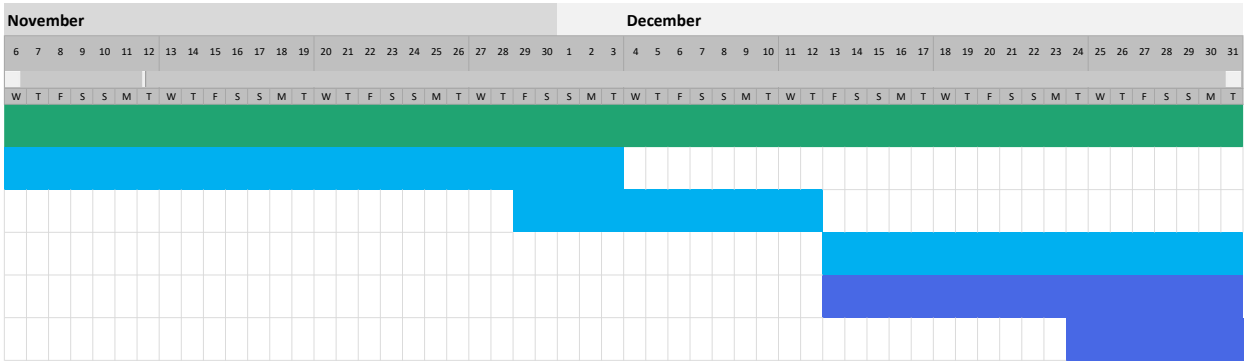
Max Sherman

Project Start Date: 30/09/2024

Scrolling Increment: 37

Legend: On track | Low risk | Med risk | High risk | Unassigned

Milestone description	Category	Progress	Start	Days
Research and design	On Track	100%	04/11/2024	58
Research	Low Risk	100%	04/11/2024	30
Requirement Definition	Low Risk	100%	29/11/2024	14
System Architecture Design	Low Risk	100%	13/12/2024	19
GUI design	Med Risk	100%	13/12/2024	19
Finalising Designs	Med Risk	100%	24/12/2024	8



Telemetry Data Analysis Tool

University of Brighton

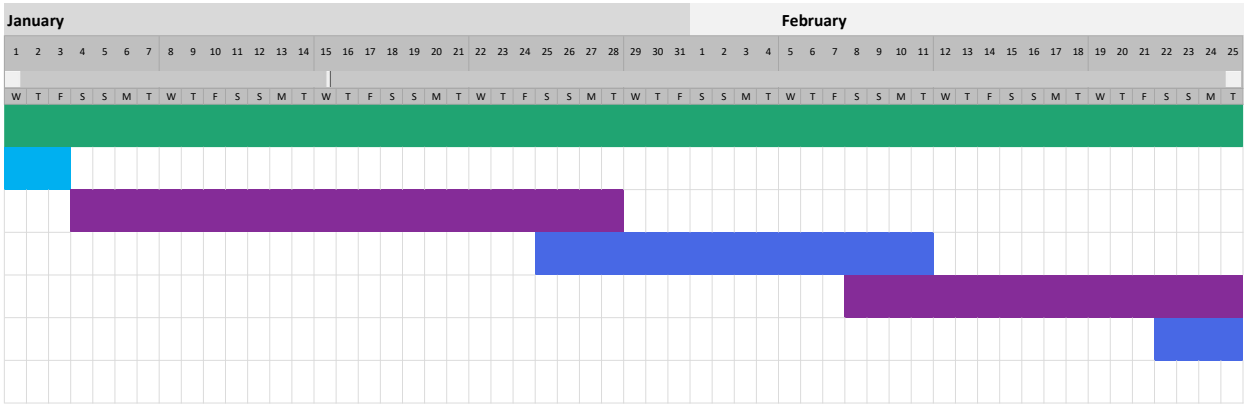
Max Sherman

Project Start Date: 30/09/2024

Scrolling Increment: 93

Legend: On track | Low risk | Med risk | High risk | Unassigned

Milestone description	Category	Progress	Start	Days
Development and Implementation	On Track	100%	01/01/2025	90
Set-up Development Environment	Low Risk	100%	01/01/2025	3
API Integration	High Risk	100%	04/01/2025	25
Data Visualisation Development	Med Risk	100%	25/01/2025	18
Statistical Analysis Development	High Risk	100%	08/02/2025	18
User Interface Development	Med Risk	100%	22/02/2025	30
Data Export Functionality	Low Risk	100%	19/03/2025	5



Telemetry Data Analysis Tool

University of Brighton

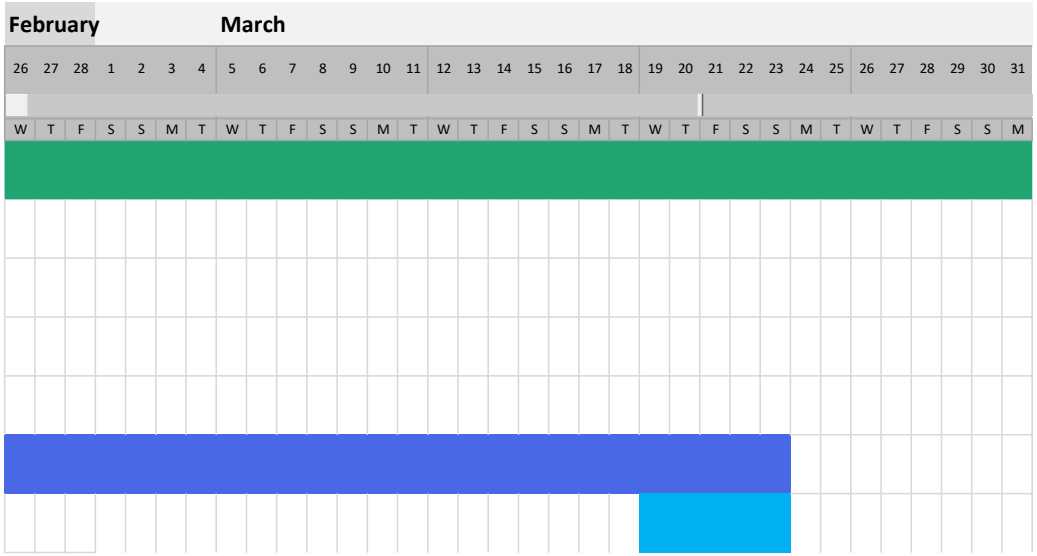
Max Sherman

Project Start Date: 30/09/2024

Scrolling Increment: 149

Legend: On track | Low risk | Med risk | High risk | Unassigned

Milestone description	Category	Progress	Start	Days
Development and Implementation	On Track	100%	01/01/2025	90
Set-up Development Environment	Low Risk	100%	01/01/2025	3
API Integration	High Risk	100%	04/01/2025	25
Data Visualisation Development	Med Risk	100%	25/01/2025	18
Statistical Analysis Development	High Risk	100%	08/02/2025	18
User Interface Development	Med Risk	100%	22/02/2025	30
Data Export Functionality	Low Risk	100%	19/03/2025	5



Telemetry Data Analysis Tool

University of Brighton

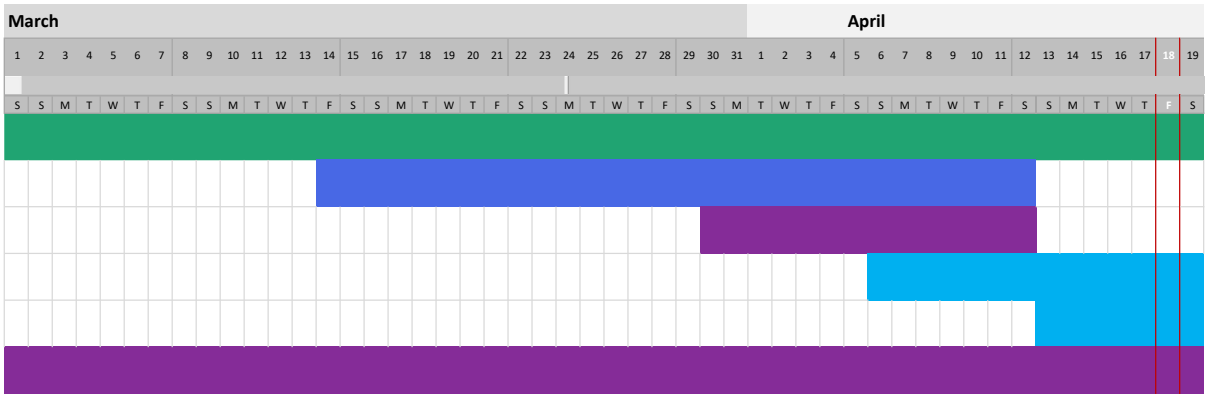
Max Sherman

Project Start Date: 30/09/2024

Scrolling Increment: 152

Legend: On track | Low risk | Med risk | High risk | Unassigned

Milestone description	Category	Progress	Start	Days
Testing and Evaluation	On Track	100%	01/03/2025	50
Unit Testing	Med Risk	100%	14/03/2025	30
External calculations	High Risk	100%	30/03/2025	14
Bug Fixing and Refinement	Low Risk	100%	06/04/2025	14
Final Validation	Low Risk	100%	13/04/2025	7
Technical Report Writing	High Risk	100%	04/11/2024	180



Appendix D

Functional and non-functional requirements

ID	Description	Inputs	Outputs	Priority	Status	Implemented	Tested
FR1	The system must retrieve and clean data from the Jolpica-F1 API and provide it in a suitable format for data analysis.	Resource type, filters parameters	Cleaned, analysis-ready data	High	Frozen	Yes	Yes
FR1.1	Data retrieval must support all resource types available through the Jolpica-F1 API.	Resource type selection	Raw JSON data for the selected resource	Medium	Frozen	Yes	Yes
FR1.2	The user must be able to specify data retrieval parameters such as limit and offset.	Parameters: limit, offset	Paginated data entries based on specified limits	low	Frozen	No	No
FR1.3	The system must allow filtering by both mandatory and optional parameters, based on the selected resource type.	Filter name and value	Filtered API response matching the criteria	Medium	Frozen	Yes	Yes
FR1.4	Data retrieval must return all data entries for the specified resource type and filters (via a "get all data" function).	Resource type, filters	JSON data containing all data entries	High	Frozen	Yes	Yes
FR1.4.1	Pagination must be handled when the total number of data entries exceeds the maximum per API call (limit = 100).	API metadata	Automatic iteration to retrieve all pages	High	Frozen	Yes	Yes
FR1.4.2	The system must distinguish between metadata (static across requests) and inner data (which varies across pages).	Resource type, paginated JSON data	Path to inner data	High	Frozen	Yes	Yes
FR1.4.3	Paginated data must be concatenated using a generic, reusable function to ensure graceful handling of pagination.	Paginated JSON responses	fully concatenated and correctly formatted JSON data	High	Frozen	Yes	Yes
FR1.5	Optional caching must be implemented to reduce redundant API calls.	Cache flag, file name	Cached JSON file	High	Frozen	Yes	Yes
FR1.5.1	Cached data should be saved as JSON files in a designated cache folder under a specified name.	JSON data, file name	JSON file written to cache directory	High	Frozen	Yes	Yes
FR1.5.2	Cached data must be loaded automatically when attempting to retrieve previously cached resources.	File name	Previously cached JSON data	High	Frozen	Yes	No
FR1.6	Data preprocessing must be integrated within the data retrieval process.	Raw API data (JSON)	Cleaned Pandas DataFrame	High	Frozen	Yes	Yes

FR1.6.1	Nested dictionaries must be flattened so that unique data fields are extracted.	Inner data	Flattened inner data	High	Frozen	Yes	Yes
FR1.6.2	Flattened data must be converted to a pandas DataFrame.	Flattened inner data	Pandas DataFrame	High	Frozen	Yes	Yes
FR1.6.3	The data must be validated by removing entries with missing values.	DataFrame	DataFrame without missing values	low	Frozen	Yes	No
FR1.6.4	Any non-numeric fields containing numeric data must be converted to appropriate numeric types.	DataFrame	DataFrame with correct numeric types	Medium	Frozen	Yes	No
FR1.6.5	If applicable, time-related fields (e.g., "time", "duration", "Q1", "Q2", "Q3") must be converted to milliseconds (ms).	DataFrame with time-formatted columns	DataFrame with time values converted to milliseconds (int)	High	Frozen	Yes	Yes
FR1.6.6	Cleaned data must be saved as a .csv file and reloaded from this file when preprocessing has already been performed.	Cleaned DataFrame, file name	CSV file saved and/or loaded	Medium	Frozen	Yes	No
FR2	The tool must generate plots using the preprocessed datasets.	Cleaned data (DataFrame), plot type	Generated plot	High	Frozen	Yes	Yes
FR2.1	The system must support generation of the following plot types: line, bar, scatter, box, and pie.	Plot type	Corresponding generated plot	High	Frozen	Yes	Yes
FR2.2	The visualisation component must support both static and interactive plots.	Plot type	Static or interactive plot	Medium	Frozen	Yes	Yes
FR2.2.1	Plot functions should allow for configurable parameters including title, hue (categorical variable), figure size, axis flipping, and theme selection.	Optional configuration parameters	Customised visualisation	low	Frozen	Yes	Yes
FR2.2.2	All plot labels must be clearly formatted, using appropriate font size, alignment, and naming conventions.	Labels	Formated text within plot	Low	Frozen	Yes	No
FR2.3	The system must support exporting static plots in JPG, PNG, SVG, and PDF formats. Interactive plots should additionally support export in HTML format.	Generated plot, export format	Plot in selected format saved to the plots directory	High	Frozen	Yes	No

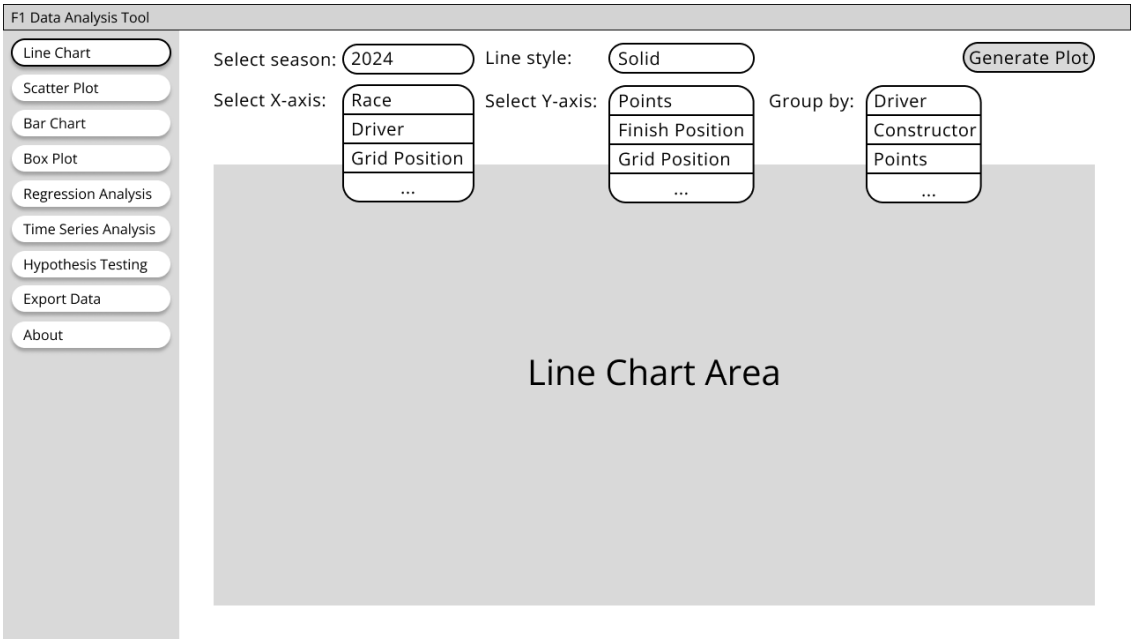
FR3	The tool must support statistical analysis on the preprocessed dataset.	Cleaned data (DataFrame), analysis type	Analysis result or model output	High	Frozen	Yes	Yes
FR3.1	The system must include descriptive statistical calculations: mean, median, mode, standard deviation, and variance.	DataFrame, numerical column name	Single value result per metric	High	Frozen	Yes	Yes
FR3.2	The system must support comparative statistical methods, including: paired and unpaired t-tests, ANOVA, Pearson and Spearman correlation analysis, Wilcoxon signed-rank test, and Chi-squared test.	DataFrame, two column names	Test statistic, p-value	High	Frozen	Yes	Yes
FR3.3	The system must implement trend analysis methods, including: simple moving average, exponential moving average, linear regression, ARIMA, and Holt-Winters exponential smoothing.	Dataframe, time-series column name	Predicted values	High	Frozen	Yes	Yes
FR4	The graphical user interface (GUI) must allow the user to fully access and utilise all tool functionalities, including data retrieval, configuration, visualisation, and analysis.	User interactions across GUI components	Plot visualisation and analysis results	High	Frozen	Yes	Yes
FR4.1	The GUI must enable users to specify the resource type and provide all required and optional filters for retrieving data from the Jolpica-F1 API.	Resource type selection, filter inputs	Retrieval of the corresponding dataset	High	Frozen	Yes	Yes
FR4.2	The GUI must support custom configuration for data visualisation.	Plot parameters and configuration options	Rendered plot	High	Frozen	Yes	Yes
FR4.2.1	Drop-down menus must allow users to select plot type, x-axis, y-axis, grouping variable (e.g., hue), and file export format.	Selections for plot type, axes, group by, file format	Rendered plot	High	Frozen	Yes	Yes
FR4.2.2	The GUI must include radio buttons for selecting plot mode (static/interactive) and a checklist for flipping the x- or y-axis.	Plot mode selection, axis flip options	Adjusted plot rendering	Medium	Frozen	Yes	Yes

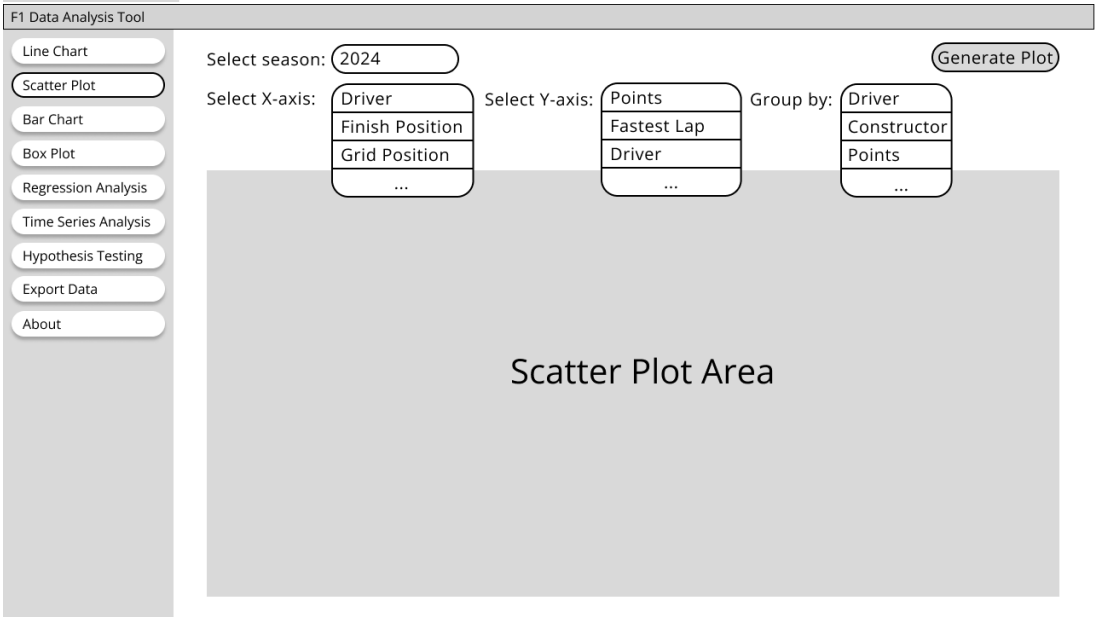
FR4.2.3	The interface must include buttons to generate and save plots.	Generate/save button click	Plot generation and rendering or exported plot	High	Frozen	Yes	Yes
FR4.3	The GUI must support input of parameters required for performing statistical analysis.	Selected analysis function, columns, extra parameters	Analysis results	High	Frozen	Yes	Yes
FR4.3.1	Drop-down menus must allow users to select the analysis function and required input columns.	Drop-down selections for function and data columns	Analysis configurations	High	Frozen	Yes	Yes
FR4.3.2	The interface must include a text input for entering any additional parameters for the selected analysis method.	User-specified parameters (e.g., window size)	Adjusted analysis configurations	Low	Frozen	No	No
FR4.3.3	A checkbox must be available for enabling the conversion of time values to milliseconds prior to analysis.	Tim conversion flag	Preprocessed, analysis-ready data	Medium	Frozen	Yes	Yes
FR4.3.4	A button must be provided to initiate the analysis operation.	Analyse button click	Analysis output rendering	High	Frozen	Yes	Yes
FR4.4	The GUI must include a dedicated area for displaying generated plots and analysis results.	Plot or analysis result from backend	Plot or result rendering in GUI area	High	Frozen	Yes	Yes

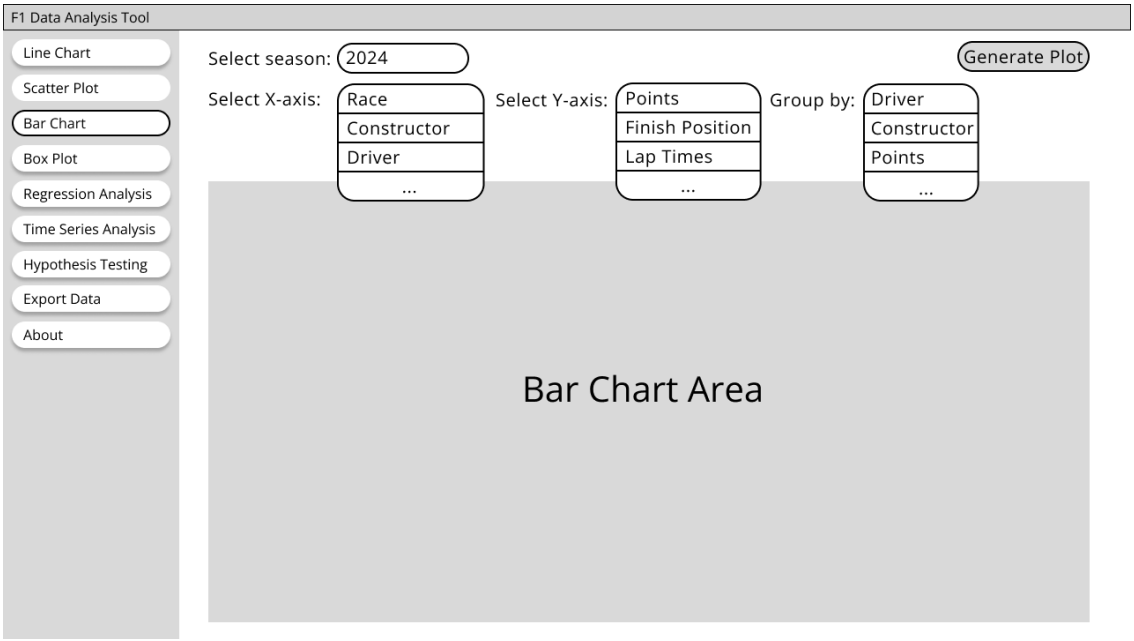
ID	Description	Inputs	Outputs	Priority	Status	Implemented	Tested
NFR1	The system must efficiently handle large datasets, maintaining low response times and consistent performance across all operations.	Large datasets, complex operations	Fast response times, system responsiveness	High	Frozen	Yes	No
NFR2	The graphical user interface (GUI) must be intuitive, user-friendly, and accessible, allowing seamless navigation across all tool features.	User actions	Intuitive layout and navigation	High	Frozen	Yes	Yes
NFR3	The system must validate data retrieved from the API to ensure correctness, completeness, and accuracy, reducing the risk of processing invalid or	Raw API data	Cleaned, validated data	Medium	Frozen	Yes	No
NFR4	The system architecture must be modular and scalable to support future feature extensions with minimal impact on existing components.	New features or modules	Seamless integration, minimal modification to existing codebase	Medium	Frozen	Yes	No
NFR5	Clear documentation and user guides must be provided to assist both end-users and developers in understanding the system's functionality and	User attempts to access documentation	Clear, accessible guides and documentation	Medium	Frozen	No	No
NFR6	The system must implement robust error handling and provide informative feedback to users regarding any encountered issues, including the	Runtime errors, API issues, or invalid user inputs	Error messages with specific details	Medium	Frozen	No	No
NFR7	The codebase should be maintainable and modular, following consistent naming conventions and structured design to simplify	Code modifications, debugging activities	Clean, modular code	High	Frozen	Yes	No
NFR8	The system must support automated testing practices, including unit and integration tests, to facilitate ongoing verification and validation.	Test cases, unit/integration test functions	Test reports, bug identification	High	Frozen	Yes	Yes

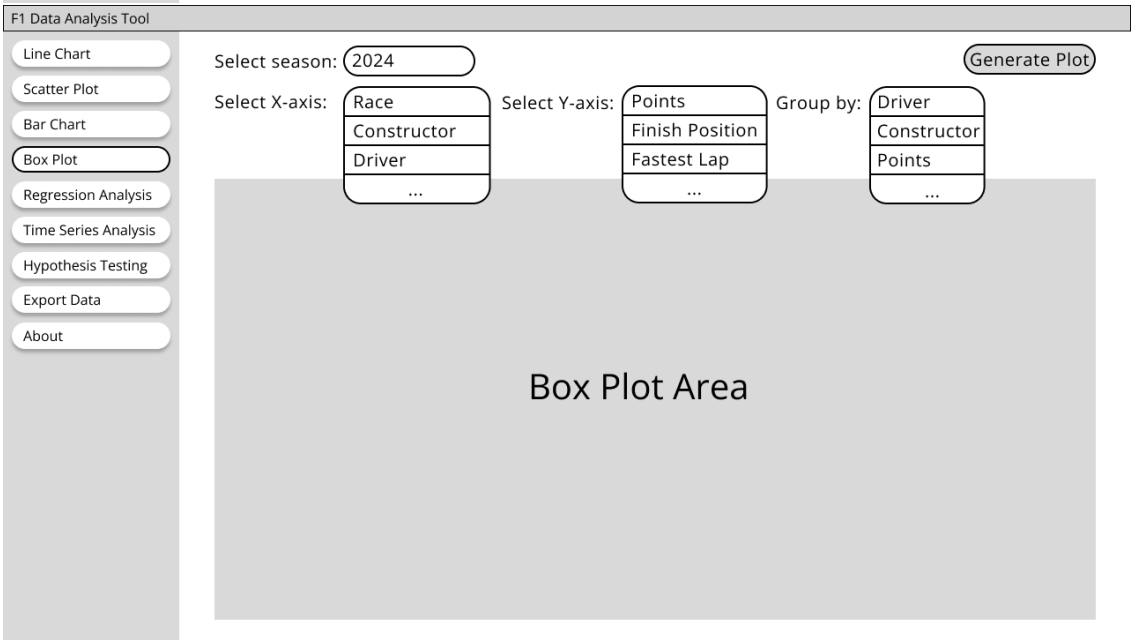
Appendix E

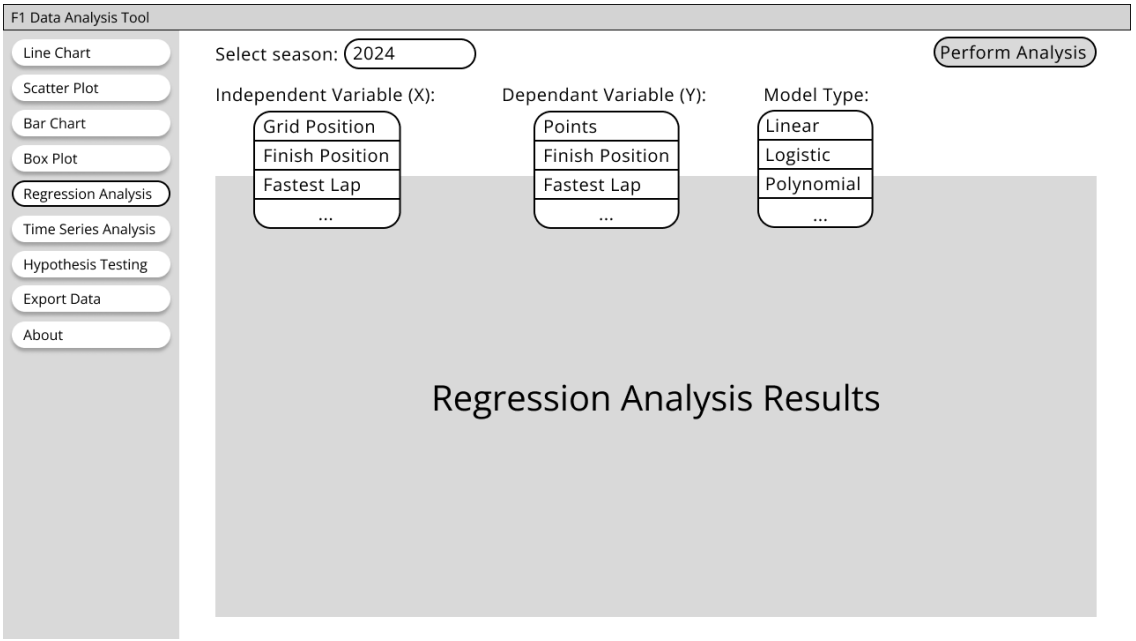
GUI sketches

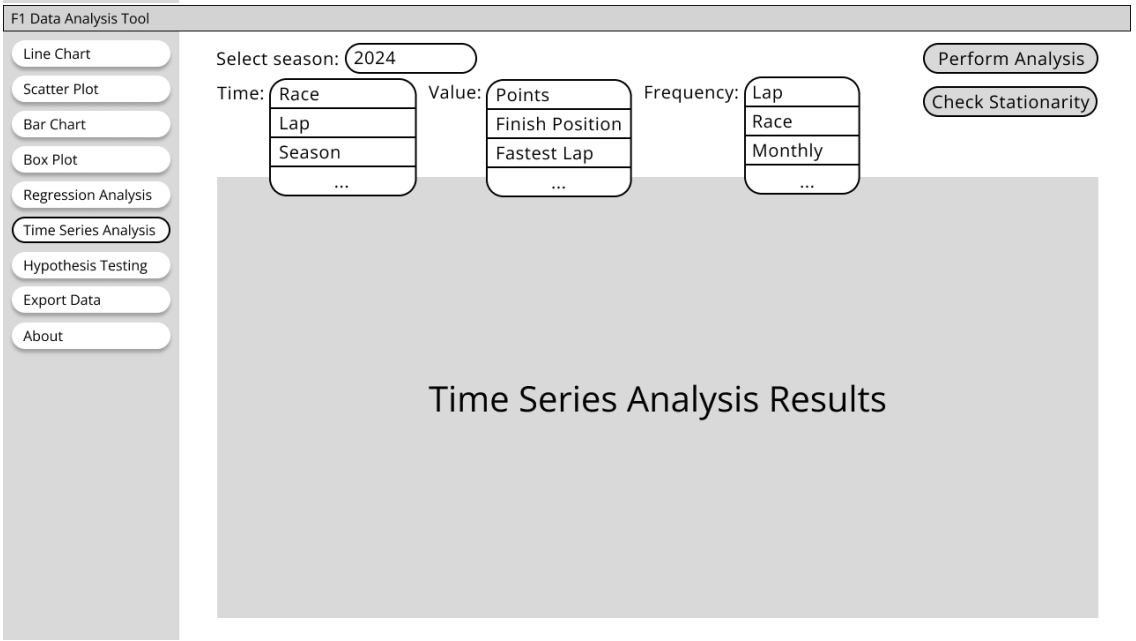


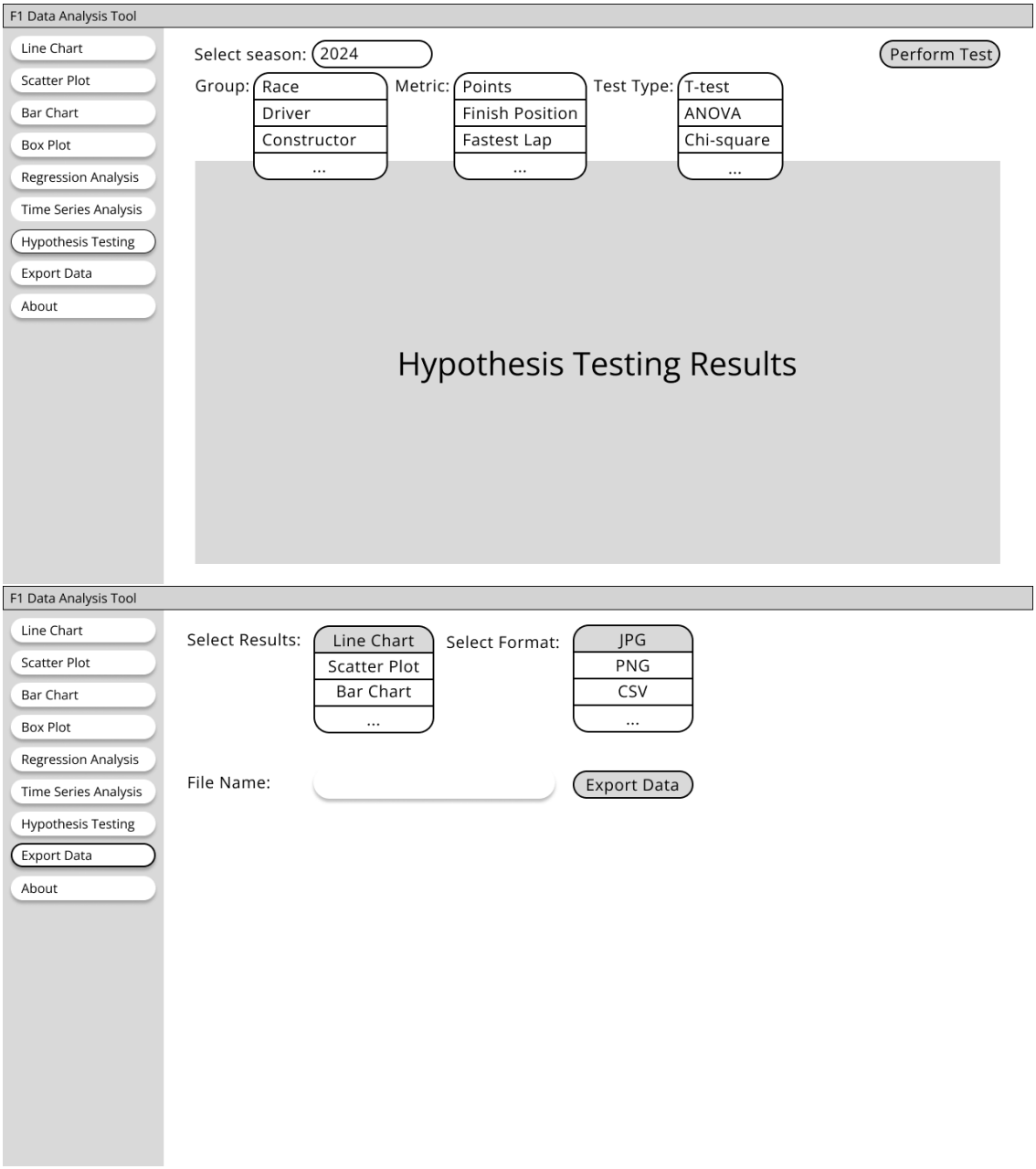


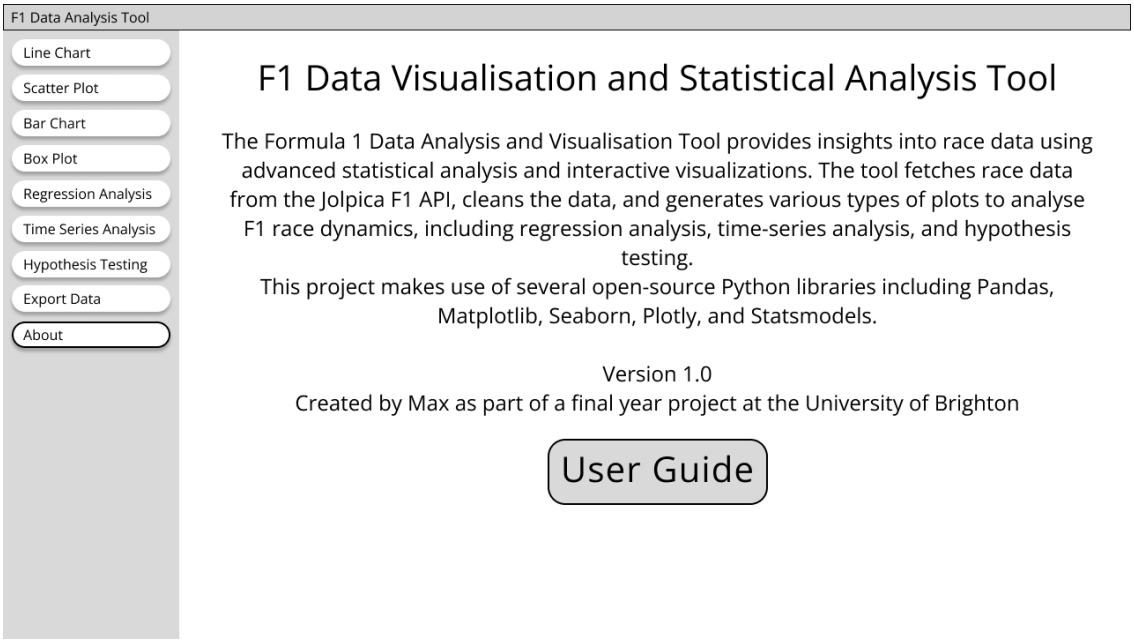

F1 Data Analysis Tool
Line Chart
Scatter Plot
Bar Chart
Box Plot
Regression Analysis
Time Series Analysis
Hypothesis Testing
Export Data
About
Select season: 2024
Generate Plot
Select X-axis: Race Constructor Driver ...
Select Y-axis: Points Finish Position Lap Times ...
Group by: Driver Constructor Points ...
Bar Chart Area


F1 Data Analysis Tool
Line Chart
Scatter Plot
Bar Chart
Box Plot
Regression Analysis
Time Series Analysis
Hypothesis Testing
Export Data
About
Select season: 2024
Generate Plot
Select X-axis: Race Constructor Driver ...
Select Y-axis: Points Finish Position Fastest Lap ...
Group by: Driver Constructor Points ...
Box Plot Area


F1 Data Analysis Tool
Line Chart
Scatter Plot
Bar Chart
Box Plot
Regression Analysis
Time Series Analysis
Hypothesis Testing
Export Data
About
Select season: 2024
Perform Analysis
Independent Variable (X):
Grid Position
Finish Position
Fastest Lap
...
Dependant Variable (Y):
Points
Finish Position
Fastest Lap
...
Model Type:
Linear
Logistic
Polynomial
...
Regression Analysis Results


F1 Data Analysis Tool
Line Chart
Scatter Plot
Bar Chart
Box Plot
Regression Analysis
Time Series Analysis
Hypothesis Testing
Export Data
About
Select season: 2024
Perform Analysis
Check Stationarity
Time:
Race
Lap
Season
...
Value:
Points
Finish Position
Fastest Lap
...
Frequency:
Lap
Race
Monthly
...
Time Series Analysis Results


F1 Data Analysis Tool
Line Chart
Scatter Plot
Bar Chart
Box Plot
Regression Analysis
Time Series Analysis
Hypothesis Testing
Export Data
About
Select season: 2024
Perform Test
Group: Race Driver Constructor ...
Metric: Points Finish Position Fastest Lap ...
Test Type: T-test ANOVA Chi-square ...
Hypothesis Testing Results
F1 Data Analysis Tool
Line Chart
Scatter Plot
Bar Chart
Box Plot
Regression Analysis
Time Series Analysis
Hypothesis Testing
Export Data
About
Select Results: Line Chart Scatter Plot Bar Chart ...
Select Format: JPG PNG CSV ...
File Name:
Export Data


F1 Data Analysis Tool
Line Chart
Scatter Plot
Bar Chart
Box Plot
Regression Analysis
Time Series Analysis
Hypothesis Testing
Export Data
About
F1 Data Visualisation and Statistical Analysis Tool
The Formula 1 Data Analysis and Visualisation Tool provides insights into race data using advanced statistical analysis and interactive visualizations. The tool fetches race data from the Jolpica F1 API, cleans the data, and generates various types of plots to analyse F1 race dynamics, including regression analysis, time-series analysis, and hypothesis testing.
This project makes use of several open-source Python libraries including Pandas, Matplotlib, Seaborn, Plotly, and Statsmodels.
Version 1.0
Created by Max as part of a final year project at the University of Brighton
User Guide

Appendix F

requirements.txt

```
requests==2.32.0          # For API interaction with Jolpica-F1
pandas==2.1.2             # Data manipulation and preprocessing
matplotlib==3.8.0         # Data visualization (static plots)
seaborn==0.13.2           # Data visualization (static plots)
plotly==5.15.0            # Data visualization (interactive plots)
scipy==1.11.2             # Statistical functions (comparative analysis)
statsmodels==0.14.0       # Advanced statistical modeling
scikit-learn==1.6.1       # Linear regression
numpy==1.25.2             # Efficient numerical computations
dash==2.18.2              # GUI framework
Flask==3.0.3              # Micro web framework
pytest==8.3.3             # Python testing framework
```

Appendix G

Source code

G.1 jolpica_api.py

```
import logging
import requests
import api.cache_manager as cache_manager
import api.json_handler as json_handler
import api.data_preprocessing as dp
import pandas as pd
from typing import Dict, Any, List, Optional
from pathlib import Path
from enumeration.resource_types import ResourceType

# Logging configuration
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

class JolpicaAPI:

    # Constants
    BASE_URL = "https://api.jolpi.ca/ergast/f1/"
    CACHE_DIR = Path(__file__).resolve().parent.parent.parent / "data/cache"
    CACHE_DIR.mkdir(parents=True, exist_ok=True)

    # Parameter constants
    DEFAULT_LIMIT = 30
    MAXIMUM_LIMIT = 100
    DEFAULT_OFFSET = 0

    # Init function validating resource type and filters
    def __init__(self, resource_type: str, params: Optional[Dict[str, Any]] = None, filters: Optional[Dict[str, Any]] = None):
        # Set instance variables
        self.params = {}
        if params is not None:
            self.set_params(params)
        else:
            self.set_params({"limit": self.DEFAULT_LIMIT, "offset": self.DEFAULT_OFFSET})
```

```
34
35          self.resource_type = None
36          self.set_resource_type(resource_type)
37
38          self.filters = {}
39          if filters is not None:
40              self.set_filters(filters)
41
42          self.endpoint = None
43          self.set_endpoint()
44
45      def set_params(self, params: Dict[str, Any]) -> None:
46          self.params = params
47
48      def get_params(self) -> Dict[str, Any]:
49          return self.params
50
51      def set_resource_type(self, resource_type: str) -> None:
52          self.resource_type = resource_type
53
54      def get_resource_type(self) -> str:
55          return self.resource_type
56
57      # Set all the filters provided
58      def set_filters(self, filters: Dict[str, Any]) -> None:
59          # Get mandatory and optional filters for the resource type
60          mandatory = ResourceType.get_mandatory(resource_type=self.
                resource_type)
61          optional = ResourceType.get_optional(resource_type=self.
                resource_type)
62
63          # Check mandatory filters to see if they are present
64          for key in mandatory:
65              if key not in filters:
66                  raise ValueError(f"Mandatory filter '{key}' not
                        provided in {filters}")
67
68          # Check other filters to see if they are allowed for this
                resource typee
69          for key in filters.keys():
70              if key not in mandatory + optional:
71                  raise ValueError(f"Invalid filter {key}")
72
73          # Set the filters
74          self.filters = filters
75
76      # Get all filters
77      def get_filters(self) -> Dict[str, Any]:
78          return self.filters
79
80      # Set the endpoint of the API request
81      def set_endpoint(self) -> None:
82          endpoint_parts = []
83          filters = self.get_filters().copy()
84
85          # Add season and round first if present
```

```
86          season = filters.pop("season", None)
87          round_number = filters.pop("round", None)
88          if season:
89              endpoint_parts.append(season)
90              if round_number:
91                  endpoint_parts.append(round_number)
92
93          # Add other filters dynamically excluding position
94          position = filters.pop("position", None)
95          for key, value in filters.items():
96              if value:
97                  endpoint_parts.extend([key, value])
98
99          # Append resource type at the end if not already added
100         if self.get_resource_type() not in endpoint_parts:
101             endpoint_parts.append(self.get_resource_type())
102
103         if position:
104             endpoint_parts.append(position)
105
106         self.endpoint = "/".join(endpoint_parts)
107
108     # Get endpoint
109     def get_endpoint(self) -> str:
110         return self.endpoint
111
112
113     def get_data(self, use_cache: bool = True) -> Dict[str, Any]:
114         """
115         Retrieves data from the Jolpica-F1 API with optional
                caching.
116
117         :param use_cache: Whether to use cached data if available
118         :return: JSON response from the API
119         """
120
121         # Return cached file if cache is enabled and cache file
                exists
122         if use_cache and cache_manager.is_cached(self.
                get_cache_file_path_params()):
123             return cache_manager.load_cache(self.
                    get_cache_file_path_params())
124
125         # Add endpoint to the base url
126         url = f"{self.BASE_URL}{self.get_endpoint()}"
127
128         # Make API call
129         try:
130
131             # Get API data and check for errors
132             response = requests.get(url, params=self.get_params())
133             response.raise_for_status()
134             data = response.json()
135
136             # Save data to cache file if cache is enabled
137             if use_cache:
```

```
138                 cache_manager.cache_data(self.
                        get_cache_file_path_params(), data)
139
140             # Return the response in JSON format
141             return data
142
143         # Error handling if an error occurs during data retrieval
144         except requests.exceptions.RequestException as e:
145             logging.error(f"Error retrieving data from {url} with
                    params {self.get_params()}: {e}")
146             return {"error": str(e)}
147
148     # Retrieve all data from endpoint using pagination
149     def get_all_data(self, use_cache: bool = True) -> Dict[str, Any
        ]:
150
151         # Return cached file if cache is enabled and cache file
            exists
152         if use_cache and cache_manager.is_cached(self.
            get_cache_file_path_all()):
153             return cache_manager.load_cache(self.
                get_cache_file_path_all())
154
155         # Retrieve initial data
156         data = self.get_data(use_cache=False)
157
158         # Retrieve total number of datapoints if there are no
            errors during data retrieval
159         if "error" in data:
160             return data
161
162         # Retrieve total number of datapoints
163         total = int(data.get("MRData", {}).get("total", 0))
164
165         # Get the path of the inner key
166         inner_key_path = json_handler.get_inner_key_path(data, self
            .get_resource_type())
167
168         # Error handling for inner key identification
169         if not inner_key_path:
170             return {"error": "Inner data path not identified in
                    response"}
171
172         # Extract metadata
173         inner_data = []
174         all_data = json_handler.set_inner_data(data, inner_key_path
            , inner_data)
175
176         # Set parameters
177         self.set_params({"limit": self.MAXIMUM_LIMIT, "offset":
            self.DEFAULT_OFFSET})
178
179         # Pagination handler loop
180         for offset in range(0, total, self.get_params()["limit"]):
181
182             # Set offset and retrieve data
```

```
183             self.set_params({"limit": self.MAXIMUM_LIMIT, "offset":
                    offset})
184             paginated_data = self.get_data(use_cache=False)
185 
186             # Error handling
187             if "error" in paginated_data:
188                 logging.warning(f"Error during pagination at offset
                        {offset}")
189                 break
190 
191             # Append data to the inner key list
192             inner_paginated_data = json_handler.get_inner_data(
                    paginated_data, inner_key_path)
193             inner_data = json_handler.extend_inner_data(inner_data,
                    inner_paginated_data)
194 
195         all_data = json_handler.set_inner_data(all_data,
                inner_key_path, inner_data)
196 
197         # Cache data if cache is enabled
198         if use_cache:
199             cache_manager.cache_data(self.get_cache_file_path_all()
                    , all_data)
200 
201         # Return the paginated data
202         return all_data
203 
204 
205     # Get inner data function using the JSON handler
206     def get_inner_data(self) -> List:
207         data = self.get_all_data()
208         inner_key_path = json_handler.get_inner_key_path(data,
                resource_type=self.get_resource_type())
209         return json_handler.get_inner_data(data, inner_key_path)
210 
211     def get_cache_file_path_params(self) -> Path:
212         return self.CACHE_DIR / f"{self.get_file_name()}_{self.
                get_params()['limit']}_{self.get_params()['offset']}.
                json"
213 
214     def get_cache_file_path_all(self) -> Path:
215         return self.CACHE_DIR / f"{self.get_file_name()}_all.json"
216 
217     def get_cleaned_file_name(self) -> str:
218         return f"{self.get_file_name()}_cleaned.csv"
219 
220     def get_file_name(self) -> str:
221         return f"{self.get_endpoint().replace('/', '_')}"
222 
223     def get_cleaned_data(self) -> pd.DataFrame:
224         file_name = self.get_cleaned_file_name()
225 
226         if dp.is_loaded_csv(file_name):
227             return dp.load_from_csv(file_name)
228 
229         flattened_data = dp.preprocess_data(self.get_inner_data())
```

```
230         df = dp.convert_to_dataframe(flattened_data)
231         df = dp.convert_to_numeric(df)
232         dp.save_to_csv(df, file_name)
233         return df
```

Listing G.1: jolpica_api.py

G.2 data_preprocessing.py

```
1  import logging
2  import pandas as pd
3  from pathlib import Path
4  from typing import List, Dict, Tuple
5
6  # Logging configuration
7  logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(
       levelname)s - %(message)s')
8
9  # Define cache and processed directories
10 CLEANED_DIR = Path(__file__).resolve().parent.parent.parent / "data
       /cleaned"
11 CLEANED_DIR.mkdir(parents=True, exist_ok=True)  # Ensure the
       processed directory exists
12
13 # Converts the provided list to a pandas dataframe
14 def convert_to_dataframe(data: List) -> pd.DataFrame:
15     # Checking for data
16     if not data:
17         logging.warning("No data provided for conversion to
               DataFrame. Returning empty DataFrame.")
18         return pd.DataFrame()
19     # Normalise JSON data
20     try:
21         return pd.json_normalize(data)
22     # Log an error if the conversion fails
23     except Exception as e:
24         logging.error("Error converting data to DataFrame: %s", str
               (e))
25         return pd.DataFrame()
26
27 # Save the dataframe to the file path
28 def save_to_csv(data: pd.DataFrame, file_name: str) -> None:
29     # Check if there is data to save
30     if data.empty:
31         logging.warning("No data provided for saving.")
32         return
33
34     # Save data to csv
35     file_path = CLEANED_DIR / file_name
36     try:
37         data.to_csv(file_path, index=False)
38         logging.info("Saved cleaned data to %s", file_path)
39     # Log an error if saving fails
40     except Exception as e:
```

```
        logging.error("Error saving cleaned data: %s", str(e))

# Load data from the csv file
def load_from_csv(file_name: str) -> pd.DataFrame:
    file_path = CLEANED_DIR / file_name
    # Load from filepath
    try:
        data = pd.read_csv(file_path)
        logging.info("Loaded cleaned data from %s", file_path)
        return data
    # Log error if csv loading fails
    except Exception as e:
        logging.error("Error loading cleaned data from csv: %s",
            str(e))
        return pd.DataFrame()

# Checks if csv file is loaded
def is_loaded_csv(file_name: str) -> bool:
    file_path = CLEANED_DIR / file_name
    return file_path.exists()

# Function that removes any nested dictionaries by flattening the
    inner data.
def preprocess_data(inner_data: List[Dict]) -> List[Dict]:
    def flatten(data: Dict) -> List[Dict]:
        flattened_entries = []
        base_data = {k: v for k, v in data.items() if not
            isinstance(v, list) or not v or not isinstance(v[0],
            dict)}

        nested_lists = {k: v for k, v in data.items() if isinstance
            (v, list) and v and isinstance(v[0], dict)}

        if not nested_lists:
            return [base_data]

        for key, nested_dicts in nested_lists.items():
            for nested_dict in nested_dicts:
                new_entry = base_data.copy()
                for k, v in nested_dict.items():
                    new_entry[f"{key}.{k}"] = v
                flattened_entries.append(new_entry)

        return flattened_entries

    flattened_data = []
    for entry in inner_data:
        flattened_data.extend(flatten(entry))

    return flattened_data

def get_columns(df: pd.DataFrame) -> List[str]:
    return list(df.columns)

def validate_data(df: pd.DataFrame) -> pd.DataFrame:
    df = df.dropna()
```

```
92      if df.empty:
93          logging.warning("Dataframe is empty after removing missing
                values. Returning empty DataFrame.")
94      return df
95
96  def get_column_min_max(df: pd.DataFrame, column: str) -> Tuple[
        float, float] | Tuple[None, None]:
97      if pd.api.types.is_numeric_dtype(df[column]):
98          return df[column].min(), df[column].max()
99      return None, None
100
101 def convert_to_ms(df: pd.DataFrame, column: List = None) -> pd.
        DataFrame:
102     def time_to_ms(time_str):
103
104         time_parts = str(time_str).split(":")
105
106         if len(time_parts) == 2:
107             minutes, rest = time_parts
108             seconds_parts = rest.split(".")
109             if len(seconds_parts) == 2:
110                 seconds, milliseconds = seconds_parts
111                 return (int(minutes) * 60000) + (int(seconds) *
                        1000) + int(milliseconds)
112         if len(time_parts) == 1:
113             time_parts = str(time_str).split(".")
114             if len(time_parts) == 2 and isinstance(time_str, float)
                    :
115                 return time_str * 1000
116         return time_str
117
118     try:
119         if column is None:
120             for col in get_columns(df):
121                 if sum(part in ["time", "duration", "Q1", "Q2", "Q3
                        "] for part in col.split(".")) == 1:
122                     df[col] = df[col].apply(time_to_ms)
123             logging.info("Time values converted to milliseconds in
                    all time or duration columns.")
124         else:
125             for col in column:
126                 df[col] = df[col].apply(time_to_ms)
127     except Exception as e:
128         logging.error(f"Error converting time to milliseconds: {e}"
                )
129     return df
130
131 def convert_to_numeric(df: pd.DataFrame) -> pd.DataFrame:
132     for col in get_columns(df):
133         if not pd.api.types.is_numeric_dtype(df[col]):
134             df[col] = df[col].apply(lambda x: pd.to_numeric(x,
                    errors="ignore"))
135     logging.info("Converted applicable columns to numeric values.")
136     return df
```

Listing G.2: data_preprocessing.py

G.3 plot_generator.py

```python
import pandas as pd
from visualisation.static_plot import plot_static_chart
from visualisation.interactive_plot import plot_interactive_chart
from visualisation.plot_saving import save_plot,
    get_plots_directory

def generate_filename(mode: str, chart_type: str, x_col: str, y_col
    : str, title: str) -> str:
    filename = f"{mode}_{chart_type}_{x_col.replace('.', '_')}{'_'
        + y_col.replace('.', '_') if y_col else ''}_{title.replace('
        ', '_')}"
    return filename.lower()

def plot_chart(
        df: pd.DataFrame, x_col: str, y_col: str = None, title: str
            = "",
        plot_type: tuple = ("static", "line"), saving: bool = False
            , save_format: str = None, **kwargs
):
    # Extract mode and specific plot type
    mode, chart_type = plot_type
    default_format = "html" if mode == "interactive" else "png"
    save_format = save_format if save_format else default_format

    # Construct filename for caching
    filename = generate_filename(mode, chart_type, x_col, y_col,
        title)
    save_path = get_plots_directory() / f"{filename}.{save_format}"

    # Generate plot
    if mode == "static":
        fig = plot_static_chart(df, x_col=x_col, y_col=y_col, title
            =title,
                                plot_type=chart_type, **kwargs)
    else:
        fig = plot_interactive_chart(df, x_col=x_col, y_col=y_col,
            title=title,
                                     plot_type=chart_type, **kwargs
                                         )
    if saving and not save_path.exists():
        save_plot(fig, filename=filename, plot_type=mode,
            file_format=save_format)

    return fig
```

Listing G.3: plot_generator.py

G.4 analysis_main.py

```python
from enumeration.analysis_functions import AnalysisFunction

```

```
def run_analysis(df, analysis_type, column_1, column_2,
    additional_param):
    try:
        # Get the corresponding analysis function using the
            analysis_type
        analysis_func = AnalysisFunction.get_function(analysis_type
            )

        # Handle Descriptive Analysis functions (e.g., mean, median
            )
        if analysis_type in [AnalysisFunction.MEAN.value["label"],
                             AnalysisFunction.MEDIAN.value["label"
                                 ],
                             AnalysisFunction.MODE.value["label"],
                             AnalysisFunction.STD_DEV.value["label"
                                 ],
                             AnalysisFunction.VARIANCE.value["label
                                 "]]:
            result = analysis_func(df, column_1)
            return {"result": result, "method": analysis_type}

        # Handle Comparative Analysis (e.g., paired t-test, ANOVA)
        elif analysis_type in [AnalysisFunction.PAIRED_T_TEST.value
            ["label"],
                               AnalysisFunction.UNPAIRED_T_TEST.
                                   value["label"],
                               AnalysisFunction.ANOVA_TEST.value["
                                   label"],
                               AnalysisFunction.SPEARMAN_CORR.value
                                   ["label"],
                               AnalysisFunction.PEARSON_CORR.value[
                                   "label"],
                               AnalysisFunction.WILCOXON_TEST.value
                                   ["label"],
                               AnalysisFunction.CHI_SQUARE_TEST.
                                   value["label"]]:
            if not column_2:
                raise ValueError("Column 2 is required for
                    comparative analysis")
            result = analysis_func(df, column_1, column_2)
            return {"statistic": result[0], "p_value": result[1], "
                test": analysis_type}

        # Handle Trend Analysis (e.g., moving average, regression,
            ARIMA)
        elif analysis_type in [AnalysisFunction.SIMPLE_MOVING_AVG.
            value["label"],
                               AnalysisFunction.
                                   EXPONENTIAL_MOVING_AVG.value["
                                   label"],
                               AnalysisFunction.LINEAR_REGRESSION.
                                   value["label"],
                               AnalysisFunction.ARIMA_MODEL.value["
                                   label"],
                               AnalysisFunction.HOLT_WINTERS.value[
                                   "label"]]:
```

```
            result = analysis_func(df, column_1)
            return {"result": result.tolist(), "method":
                analysis_type}

        else:
            raise ValueError(f"Analysis type '{analysis_type}' is
                not supported.")

    except Exception as e:
        return {"error": str(e)}
```

Listing G.4: analysis_main.py

G.5 main.py

```
from dash import Dash
from layout import create_layout
from callbacks import register_callbacks
from flask import send_from_directory
from visualisation.plot_saving import get_plots_directory

# Initialize Dash app
app = Dash(__name__, suppress_callback_exceptions=True)
app.layout = create_layout(app)

# Register callbacks
register_callbacks(app)

@app.server.route('/data/plots/<path:filename>')
def serve_plot(filename):
    return send_from_directory(get_plots_directory(), filename)

if __name__ == "__main__":
    app.run_server(debug=True)
```

Listing G.5: main.py

Appendix H

Test cases and results

H.1 test_api.py

```python
import pytest
from api import json_handler
from api.jolpica_api import JolpicaAPI

@pytest.mark.parametrize("resource_type, filters", [
    ("circuits", {"season": "2024"}),
    ("circuits", {"constructors": "mercedes"}),
    ("circuits", {}),
    ("constructors", {"season": "2024", "round": "1"}),
    ("constructors", {}),
    ("drivers", {}),
    ("driverStandings", {"season": "2023"}),
    ("results", {"season": "2021", "drivers": "max_verstappen"}),
    ("pitstops", {"season": "2019", "round": "4", "pitstops": "2"}),
    ("qualifying", {"season": "2024", "round": "5", "drivers": "leclerc"}),
    ("qualifying", {"season": "2024", "round": "5"}),
    ("qualifying", {}),
    ("constructorStandings", {"season": "2020"}),
    ("laps", {"season": "2023", "round": "6"}),
    ("races", {}),
    ("results", {}),
    ("seasons", {}),
    ("sprint", {}),
    ("status", {"status": "2"}),
    ("status", {})
])
def test_data_retrieval(resource_type, filters):
    # Get data retrieval class instance
    api = JolpicaAPI(resource_type=resource_type, filters=filters)

    # Get all data, total number of data entries, inner data length
    data = api.get_all_data()
    total = data["MRData"]["total"]
    inner_key_path = json_handler.get_inner_key_path(data, resource_type)
    inner_data = json_handler.get_inner_data(data, inner_key_path)
```

```
36      length_inner_data = len(inner_data) if inner_data else 0
37
38      # Get cleaned data
39      cleaned_data = api.get_cleaned_data()
40
41      print()
42      print(f"Total data entries on Jolpica API: {total}")
43      print(f"Inner data entries retrieved from API: {
            length_inner_data}")
44      print(f"Cleaned data entries retrieved from API: {cleaned_data.
            shape[0]}")
45
46      if resource_type != "laps":
47          assert str(length_inner_data) == total
48          assert cleaned_data.shape[0] == length_inner_data
49      else:
50          assert str(cleaned_data.shape[0]) == total
```

Listing H.1: test_api.py

pytest in test_api.py:
20 total, 6 failed, 14 passed

494 ms

Collapse | Expand

_ .

test_data_retrieval 494 ms

(circuits-filters0) passed 7 ms

PASSED [5%]
Total data entries on Jolpica API: 24
Inner data entries retrieved from API: 24
Cleaned data entries retrieved from API: 24

(circuits-filters1) passed 4 ms

PASSED [10%]
Total data entries on Jolpica API: 40
Inner data entries retrieved from API: 40
Cleaned data entries retrieved from API: 40

(circuits-filters2) passed 4 ms

PASSED [15%]
Total data entries on Jolpica API: 77
Inner data entries retrieved from API: 77
Cleaned data entries retrieved from API: 77

(constructors-filters3) passed 4 ms

PASSED [20%]
Total data entries on Jolpica API: 10
Inner data entries retrieved from API: 10
Cleaned data entries retrieved from API: 10

(constructors-filters4) passed 4 ms

PASSED [25%]
Total data entries on Jolpica API: 212
Inner data entries retrieved from API: 212
Cleaned data entries retrieved from API: 212

(drivers-filters5) failed 9 ms

FAILED [30%]
Total data entries on Jolpica API: 864
Inner data entries retrieved from API: 862
Cleaned data entries retrieved from API: 862
test_api.py:4 (test_data_retrieval[drivers-filters5])
'862' != '864'
resource_type = 'drivers', filters = {}
@pytest.mark.parametrize("resource_type, filters", [
("circuits", {"season": "2024"}),
("circuits", {"constructors": "mercedes"}),
("circuits", {}),
("constructors", {"season": "2024", "round": "1"}),
("constructors", {}),
("drivers", {}),

pytest in test_api.py:
20 total, 6 failed, 14 passed

494 ms

Collapse | Expand

```
("races", {}),
("results", {}),
("seasons", {}),
("sprint", {}),
("status", {"status": "2"}),
("status", {})
])
def test_data_retrieval(resource_type, filters):
# Get data retrieval class instance
api = JolpicaAPI(resource_type=resource_type, filters=filters)

# Get all data, total number of data entries, inner data length
data = api.get_all_data()
total = data["MRData"]["total"]
inner_key_path = json_handler.get_inner_key_path(data, resource_type)
inner_data = json_handler.get_inner_data(data, inner_key_path)
length_inner_data = len(inner_data) if inner_data else 0

# Get cleaned data
cleaned_data = api.get_cleaned_data()

print()
print(f"Total data entries on Jolpica API: {total}")
print(f"Inner data entries retrieved from API: {length_inner_data}")
print(f"Cleaned data entries retrieved from API: {cleaned_data.shape[0]}")

if resource_type != "laps":
> assert str(length_inner_data) == total
E AssertionError: assert '862' == '864'
E
E - 864
E + 862
test_api.py:47: AssertionError
```

(driverStandings-filters6) passed 5 ms

PASSED [35%]
Total data entries on Jolpica API: 22
Inner data entries retrieved from API: 22
Cleaned data entries retrieved from API: 22

(results-filters7) passed 5 ms

PASSED [40%]
Total data entries on Jolpica API: 22
Inner data entries retrieved from API: 22
Cleaned data entries retrieved from API: 22

(pitstops-filters8) passed 2 ms

pytest in test_api.py:
20 total, 6 failed, 14 passed

494 ms

Collapse | Expand

Inner data entries retrieved from API: 1
Cleaned data entries retrieved from API: 1

(qualifying-filters10) passed 3 ms

PASSED [55%]
Total data entries on Jolpica API: 20
Inner data entries retrieved from API: 20
Cleaned data entries retrieved from API: 20

(qualifying-filters11) failed 17 ms

FAILED [60%]
Total data entries on Jolpica API: 10554
Inner data entries retrieved from API: 1305
Cleaned data entries retrieved from API: 1305
test_api.py:4 (test_data_retrieval[qualifying-filters11])
'1305' != '10554'
resource_type = 'qualifying', filters = {}
@pytest.mark.parametrize("resource_type, filters", [
("circuits", {"season": "2024"}),
("circuits", {"constructors": "mercedes"}),
("circuits", {}),
("constructors", {"season": "2024", "round": "1"}),
("constructors", {}),
("drivers", {}),
("driverStandings", {"season": "2023"}),
("results", {"season": "2021", "drivers": "max_verstappen"}),
("pitstops", {"season": "2019", "round": "4", "pitstops": "2"}),
("qualifying", {"season": "2024", "round": "5", "drivers": "leclerc"}),
("qualifying", {"season": "2024", "round": "5"}),
("qualifying", {}),
("constructorStandings", {"season": "2020"}),
("laps", {"season": "2023", "round": "6"}),
("races", {}),
("results", {}),
("seasons", {}),
("sprint", {}),
("status", {"status": "2"}),
("status", {})
])
def test_data_retrieval(resource_type, filters):
# Get data retrieval class instance
api = JolpicaAPI(resource_type=resource_type, filters=filters)

# Get all data, total number of data entries, inner data length
data = api.get_all_data()
total = data["MRData"]["total"]
inner_key_path = json_handler.get_inner_key_path(data, resource_type)
inner_data = json_handler.get_inner_data(data, inner_key_path)

pytest in test_api.py:
20 total, 6 failed, 14 passed

494 ms

Collapse | Expand

print(f"Cleaned data entries retrieved from API: {cleaned_data.shape[0]}")

if resource_type != "laps":
> assert str(length_inner_data) == total
E AssertionError: assert '1305' == '10554'
E
E - 10554
E + 1305
test_api.py:47: AssertionError

(constructorStandings-filters12) passed 3 ms

PASSED [65%]
Total data entries on Jolpica API: 10
Inner data entries retrieved from API: 10
Cleaned data entries retrieved from API: 10

(laps-filters13) passed 7 ms

PASSED [70%]
Total data entries on Jolpica API: 1513
Inner data entries retrieved from API: 78
Cleaned data entries retrieved from API: 1513

(races-filters14) failed 24 ms

FAILED [75%]
Total data entries on Jolpica API: 1149
Inner data entries retrieved from API: 1139
Cleaned data entries retrieved from API: 1139
test_api.py:4 (test_data_retrieval[races-filters14])
'1139' != '1149'
resource_type = 'races', filters = {}
@pytest.mark.parametrize("resource_type, filters", [
("circuits", {"season": "2024"}),
("circuits", {"constructors": "mercedes"}),
("circuits", {}),
("constructors", {"season": "2024", "round": "1"}),
("constructors", {}),
("drivers", {}),
("driverStandings", {"season": "2023"}),
("results", {"season": "2021", "drivers": "max_verstappen"}),
("pitstops", {"season": "2019", "round": "4", "pitstops": "2"}),
("qualifying", {"season": "2024", "round": "5", "drivers": "leclerc"}),
("qualifying", {"season": "2024", "round": "5"}),
("qualifying", {}),
("constructorStandings", {"season": "2020"}),
("laps", {"season": "2023", "round": "6"}),
("races", {}),
("results", {}),
("seasons", {}),

pytest in test_api.py:
20 total, 6 failed, 14 passed

494 ms

Collapse | Expand

```
# Get all data, total number of data entries, inner data length
data = api.get_all_data()
total = data["MRData"]["total"]
inner_key_path = json_handler.get_inner_key_path(data, resource_type)
inner_data = json_handler.get_inner_data(data, inner_key_path)
length_inner_data = len(inner_data) if inner_data else 0

# Get cleaned data
cleaned_data = api.get_cleaned_data()

print()
print(f"Total data entries on Jolpica API: {total}")
print(f"Inner data entries retrieved from API: {length_inner_data}")
print(f"Cleaned data entries retrieved from API: {cleaned_data.shape[0]}")

if resource_type != "laps":
> assert str(length_inner_data) == total
E AssertionError: assert '1139' == '1149'
E
E - 1149
E ? ^
E + 1139
E ? ^
test_api.py:47: AssertionError
```

(results-filters15) failed 377 ms

```
FAILED [ 80%]
Total data entries on Jolpica API: 25434
Inner data entries retrieved from API: 1119
Cleaned data entries retrieved from API: 25434
test_api.py:4 (test_data_retrieval[results-filters15])
'1119' != '25434'
resource_type = 'results', filters = {}
@pytest.mark.parametrize("resource_type, filters", [
("circuits", {"season": "2024"}),
("circuits", {"constructors": "mercedes"}),
("circuits", {}),
("constructors", {"season": "2024", "round": "1"}),
("constructors", {}),
("drivers", {}),
("driverStandings", {"season": "2023"}),
("results", {"season": "2021", "drivers": "max_verstappen"}),
("pitstops", {"season": "2019", "round": "4", "pitstops": "2"}),
("qualifying", {"season": "2024", "round": "5", "drivers": "leclerc"}),
("qualifying", {"season": "2024", "round": "5"}),
("qualifying", {}),
("constructorStandings", {"season": "2020"}),
("laps", {"season": "2023", "round": "6"}),
```

pytest in test_api.py:
20 total, 6 failed, 14 passed

494 ms

Collapse | Expand

```
# Get data retrieval class instance
api = JolpicaAPI(resource_type=resource_type, filters=filters)

# Get all data, total number of data entries, inner data length
data = api.get_all_data()
total = data["MRData"]["total"]
inner_key_path = json_handler.get_inner_key_path(data, resource_type)
inner_data = json_handler.get_inner_data(data, inner_key_path)
length_inner_data = len(inner_data) if inner_data else 0

# Get cleaned data
cleaned_data = api.get_cleaned_data()

print()
print(f"Total data entries on Jolpica API: {total}")
print(f"Inner data entries retrieved from API: {length_inner_data}")
print(f"Cleaned data entries retrieved from API: {cleaned_data.shape[0]}")

if resource_type != "laps":
> assert str(length_inner_data) == total
E AssertionError: assert '1119' == '25434'
E
E - 25434
E + 1119
test_api.py:47: AssertionError
```

(seasons-filters16) passed 4 ms

```
PASSED [ 85%]
Total data entries on Jolpica API: 76
Inner data entries retrieved from API: 76
Cleaned data entries retrieved from API: 76
```

(sprint-filters17) failed 5 ms

```
FAILED [ 90%]
Total data entries on Jolpica API: 380
Inner data entries retrieved from API: 80
Cleaned data entries retrieved from API: 80
test_api.py:4 (test_data_retrieval[sprint-filters17])
'80' != '380'
resource_type = 'sprint', filters = {}
@pytest.mark.parametrize("resource_type, filters", [
("circuits", {"season": "2024"}),
("circuits", {"constructors": "mercedes"}),
("circuits", {}),
("constructors", {"season": "2024", "round": "1"}),
("constructors", {}),
("drivers", {}),
("driverStandings", {"season": "2023"}),
```

pytest in test_api.py:
20 total, 6 failed, 14 passed

494 ms

Collapse | Expand

```
("results", {}),
("seasons", {}),
("sprint", {}),
("status", {"status": "2"}),
("status", {})
])
def test_data_retrieval(resource_type, filters):
# Get data retrieval class instance
api = JolpicaAPI(resource_type=resource_type, filters=filters)

# Get all data, total number of data entries, inner data length
data = api.get_all_data()
total = data["MRData"]["total"]
inner_key_path = json_handler.get_inner_key_path(data, resource_type)
inner_data = json_handler.get_inner_data(data, inner_key_path)
length_inner_data = len(inner_data) if inner_data else 0

# Get cleaned data
cleaned_data = api.get_cleaned_data()

print()
print(f"Total data entries on Jolpica API: {total}")
print(f"Inner data entries retrieved from API: {length_inner_data}")
print(f"Cleaned data entries retrieved from API: {cleaned_data.shape[0]}")

if resource_type != "laps":
> assert str(length_inner_data) == total
E AssertionError: assert '80' == '380'
E
E - 380
E ? -
E + 80
test_api.py:47: AssertionError
```

(status-filters18) passed 3 ms

```
PASSED [ 95%]
Total data entries on Jolpica API: 1
Inner data entries retrieved from API: 1
Cleaned data entries retrieved from API: 1
```

(status-filters19) failed 3 ms

```
FAILED [100%]
Total data entries on Jolpica API: 136
Inner data entries retrieved from API: 135
Cleaned data entries retrieved from API: 135
test_api.py:4 (test_data_retrieval[status-filters19])
'135' != '136'
resource_type = 'status', filters = {}
```

pytest in test_api.py:
20 total, 6 failed, 14 passed

494 ms

Collapse | Expand

```
("results", {"season": "2021", "drivers": "max_verstappen"}),
("pitstops", {"season": "2019", "round": "4", "pitstops": "2"}),
("qualifying", {"season": "2024", "round": "5", "drivers": "leclerc"}),
("qualifying", {"season": "2024", "round": "5"}),
("qualifying", {}),
("constructorStandings", {"season": "2020"}),
("laps", {"season": "2023", "round": "6"}),
("races", {}),
("results", {}),
("seasons", {}),
("sprint", {}),
("status", {"status": "2"}),
("status", {})
])
def test_data_retrieval(resource_type, filters):
# Get data retrieval class instance
api = JolpicaAPI(resource_type=resource_type, filters=filters)

# Get all data, total number of data entries, inner data length
data = api.get_all_data()
total = data["MRData"]["total"]
inner_key_path = json_handler.get_inner_key_path(data, resource_type)
inner_data = json_handler.get_inner_data(data, inner_key_path)
length_inner_data = len(inner_data) if inner_data else 0

# Get cleaned data
cleaned_data = api.get_cleaned_data()

print()
print(f"Total data entries on Jolpica API: {total}")
print(f"Inner data entries retrieved from API: {length_inner_data}")
print(f"Cleaned data entries retrieved from API: {cleaned_data.shape[0]}")

if resource_type != "laps":
> assert str(length_inner_data) == total
E AssertionError: assert '135' == '136'
E
E - 136
E + 135
test_api.py:47: AssertionError
```

Generated by PyCharm on 17/04/2025, 15:34

H.2 test_visualisation.py

```
import pytest
from api.jolpica_api import JolpicaAPI
from visualisation.plot_generator import plot_chart
import matplotlib.figure
import plotly.graph_objs as go

@pytest.mark.parametrize("resource, filters, x_col, y_col, hue, plot_kind", [
    ("pitstops", {"season": "2019", "round": "3"}, "driverId", "duration", None, "box"),
    ("pitstops", {"season": "2019", "round": "3"}, "driverId", "duration", None, "bar"),
    ("pitstops", {"season": "2019", "round": "3"}, "duration", None, None, "box"),
    ("constructorStandings", {"season": "2023"}, "Constructor.name", "points", None, "bar"),
    ("constructorStandings", {"season": "2023"}, "wins", None, None, "pie"),
    ("results", {"season": "2021", "drivers": "max_verstappen"}, "raceName", "Results.points", None, "line"),
    ("constructors", {"season": "2024", "round": "1"}, "nationality", None, None, "pie"),
    ("driverStandings", {"season": "2024"}, "Driver.code", "points", None, "bar"),
    ("driverStandings", {"season": "2024"}, "wins", "points", None, "scatter"),
    ("laps", {"season": "2023", "round": "18"}, "number", "Timings.position", "Timings.driverId", "line")

])
def test_plot_static_and_interactive(resource, filters, x_col, y_col, hue, plot_kind):
    df = JolpicaAPI(resource_type=resource, filters=filters).get_cleaned_data()

    static_fig = plot_chart(df, x_col=x_col, y_col=y_col, title=f"Static {plot_kind} plot", plot_type=("static", plot_kind), saving=True)
    interactive_fig = plot_chart(df, x_col=x_col, y_col=y_col, title=f"Interactive {plot_kind} plot", plot_type=("interactive", plot_kind), saving=True)

    assert isinstance(static_fig, matplotlib.figure.Figure)
    assert isinstance(interactive_fig, go.Figure)

def test_laps_position_plot():
    df = JolpicaAPI(resource_type="laps", filters={"season": "2023", "round": "18"}).get_cleaned_data()

    static_fig = plot_chart(df, x_col="number", y_col="Timings.position", title="Static Position per Lap", hue="Timings.driverId", plot_type=("static", "line"), flip_axis=["y"], saving=True)
```

```
33      interactive_fig = plot_chart(df, x_col="number", y_col="Timings
            .position", title="Interactive Position per Lap", hue="
            Timings.driverId", plot_type=("interactive", "line"),
            flip_axis=["y"], saving=True)
34
35      assert isinstance(static_fig, matplotlib.figure.Figure)
36      assert isinstance(interactive_fig, go.Figure)
```

Listing H.2: test_visualisation.py

pytest in test_visualisation.py:
11 total, 11 passed

4.87 s

Collapse | Expand

–

test_plot_static_and_interactive 4.56 s

(pitstops-filters0-driverId-duration-None-box)	passed	1.04 s
PASSED [9%]		
(pitstops-filters1-driverId-duration-None-bar)	passed	537 ms
PASSED [18%]		
(pitstops-filters2-duration-None-None-box)	passed	179 ms
PASSED [27%]		
(constructorStandings-filters3-Constructor_name-points-None-bar)	passed	254 ms
PASSED [36%]		
(constructorStandings-filters4-wins-None-None-pie)	passed	190 ms
PASSED [45%]		
(results-filters5-raceName-Results_points-None-line)	passed	447 ms
PASSED [54%]		
(constructors-filters6-nationality-None-None-pie)	passed	162 ms
PASSED [63%]		
(driverStandings-filters7-Driver_code-points-None-bar)	passed	347 ms
PASSED [72%]		
(driverStandings-filters8-wins-points-None-scatter)	passed	284 ms
PASSED [81%]		
(laps-filters9-number-Timings_position-Timings_driverId-line)	passed	1.12 s
PASSED [90%]		

test_laps_position_plot	passed	306 ms

Generated by PyCharm on 17/04/2025, 15:34

H.3 test_analysis.py

```
import pytest
import pandas as pd
from api.jolpica_api import JolpicaAPI
from api.data_preprocessing import save_to_csv, load_from_csv,
    convert_to_ms
import analysis.descriptive_analysis as descriptive_analysis
import analysis.comparative_analysis as comparative_analysis
import analysis.trend_analysis as trend_analysis

def test_descriptive_analysis():
    df = JolpicaAPI(resource_type="pitstops", filters={"season": "
        2023", "round": "5"}).get_cleaned_data()
    df = convert_to_ms(df)

    save_to_csv(df, "descriptive_analysis.csv")

    mean = descriptive_analysis.calculate_mean(df, 'duration')
    median = descriptive_analysis.calculate_median(df, 'duration')
    mode = descriptive_analysis.calculate_mode(df, 'duration')
    std_dev = descriptive_analysis.calculate_std_dev(df, 'duration'
        )
    variance = descriptive_analysis.calculate_variance(df, '
        duration')
    stats = descriptive_analysis.descriptive_statistics(df, '
        duration')

    assert mean == 23916.3
    assert median == 22549
    assert mode is None
    assert round(std_dev, 6) == 5913.982237
    assert round(variance, 2) == 34975185.91
    assert stats == {
        'Mean': mean,
        'Median': median,
        'Mode': mode,
        'Standard Deviation': std_dev,
        'Variance': variance
    }

def test_paired_t_test():
    qualifying = JolpicaAPI(resource_type="qualifying", filters={"
        season": "2023", "round": "5"}).get_cleaned_data()
    qualifying = convert_to_ms(qualifying)
    results = JolpicaAPI(resource_type="results", filters={"season"
        : "2023", "round": "5"}).get_cleaned_data()
    results = convert_to_ms(results)

    qualifying["Fastest_Qualifying_Time"] = qualifying[["Q1", "Q2",
        "Q3"]].min(axis=1, numeric_only=True)
    merged = pd.merge(
        qualifying[["Driver.driverId", "Fastest_Qualifying_Time"]],
        results.rename(columns={"Results.Driver.driverId": "Driver.
            driverId"})[["Driver.driverId", "Results.FastestLap.Time
```

```
            .time"]],
        on="Driver.driverId",
        how="inner"
    )
    save_to_csv(merged, "paired_t_test.csv")

    stat, p_value = comparative_analysis.paired_t_test(merged, "
        Fastest_Qualifying_Time", "Results.FastestLap.Time.time")

    assert stat == pytest.approx(-26.3835436607094, abs=1e-13)
    assert p_value == pytest.approx(1.96725589915244e-16, abs=1e
        -30)

def test_unpaired_t_test():
    df = JolpicaAPI(resource_type="laps", filters={"season": "2023"
        , "round": "6"}).get_cleaned_data()
    df = convert_to_ms(df)
    df1, df2 = (df[df["Timings.driverId"] == "max_verstappen"].
        rename(columns={"Timings.time": "max_verstappen.time"}),
                df[df["Timings.driverId"] == "hamilton"].rename(
                    columns={"Timings.time": "hamilton.time"}))
    merged = pd.merge(df1, df2, on="number")

    save_to_csv(merged, "unpaired_t_test.csv")

    stat, p_value = comparative_analysis.unpaired_t_test(merged, "
        max_verstappen.time", "hamilton.time")

    assert stat == pytest.approx(-0.313855177502919, abs=1e-15)
    assert p_value == pytest.approx(0.754055768792724, abs=1e-15)

def test_anova():
    df = JolpicaAPI(resource_type="laps", filters={"season": "2023"
        , "round": "5"}).get_cleaned_data()
    df = convert_to_ms(df)

    save_to_csv(df, "anova.csv")

    stat, p_value = comparative_analysis.anova_test(df, "Timings.
        time", "Timings.driverId")

    assert stat == pytest.approx(2.18381414084741, abs=1e-13)
    assert p_value == pytest.approx(0.0023771601197084, abs=1e-15)

def test_spearman_correlation():
    df = JolpicaAPI(resource_type="results", filters={"season": "
        2024"}).get_cleaned_data()
    save_to_csv(df, "spearman_correlation.csv")
    corr, p_value = comparative_analysis.perform_spearman_analysis(
        df, "Results.grid", "Results.position")

    assert corr == pytest.approx(0.728401754402222, abs=1e-15)
    assert p_value == pytest.approx(2.33950695732824e-80, abs=1e
        -93)

def test_pearson_correlation():
```

```
89      df = JolpicaAPI(resource_type="results", filters={"season": "
            2024"}).get_cleaned_data()
90      save_to_csv(df, "pearson_correlation.csv")
91      corr, p_value = comparative_analysis.perform_pearson_analysis(
            df, "Results.grid", "Results.position")
92
93      assert corr == pytest.approx(0.727964654699956, abs=1e-15)
94      assert p_value == pytest.approx(3.23405719792189e-80, abs=1e
            -92)
95
96  def test_wilcoxon():
97      df = JolpicaAPI(resource_type="results", filters={"season": "
            2024"}).get_cleaned_data()
98      save_to_csv(df, "wilcoxon.csv")
99      stat, p_value = comparative_analysis.wilcoxon_test(df, "Results
            .grid", "Results.position")
100
101     assert stat == 39243
102     assert p_value == pytest.approx(0.428954014489374, abs=0.01)
103
104 def test_chi_square():
105     df = JolpicaAPI(resource_type="results", filters={"season": "
            2024"}).get_cleaned_data()
106     save_to_csv(df, "chi_square.csv")
107     stat, p_value = comparative_analysis.chi_square_test(df, "
            Results.grid", "Results.position")
108
109     assert stat == pytest.approx(909.44524044796, abs=1e-11)
110     assert p_value == pytest.approx(2.26782891112557E-45, abs=1e
            -56)
111
112 def test_simple_moving_average():
113     df = JolpicaAPI(resource_type="laps", filters={"season": "2023"
            , "round": "6"}).get_cleaned_data()
114     df_max = df[df["Timings.driverId"] == "max_verstappen"]
115     df_max = convert_to_ms(df_max)
116     save_to_csv(df_max, "simple_moving_average.csv")
117
118     result = trend_analysis.simple_moving_average(df_max, column="
            Timings.time", window=10)
119     result = result.fillna(0)
120
121     assert result.to_list() == [
122         78561.2, 77823.2, ... 88881.9, 88036.1
123         ]
124
125 def test_exponential_moving_average():
126     df = JolpicaAPI(resource_type="laps", filters={"season": "2023"
            , "round": "6"}).get_cleaned_data()
127     df_max = df[df["Timings.driverId"] == "max_verstappen"]
128     df_max = convert_to_ms(df_max)
129     save_to_csv(df_max, "exponential_moving_average.csv")
130
131     result = trend_analysis.exponential_moving_average(df_max,
            column="Timings.time", span=5)
132     expected_result = [
```

```
133         84238.0000000000, 82614.3333333333, ... 86572.6336604708,
                86115.4224403139
134     ]
135
136
137     assert [round(num, 8) for num in result.to_list()] == [round(
            num, 8) for num in expected_result]
138
139 def test_linear_regression():
140     df = JolpicaAPI(resource_type="laps", filters={"season": "2023"
            , "round": "6"}).get_cleaned_data()
141     df = convert_to_ms(df)
142     #save_to_csv(df, "linear_regression.csv")
143
144     result = trend_analysis.linear_regression(df, "Timings.time", [
            'Timings.position'])
145
146     expected_result = load_from_csv('linear_regression.csv')
147     expected_result = expected_result["predicted.y"]
148
149     assert result.tolist() == pytest.approx(expected_result.tolist
            (), abs=1e-10)
150
151 def test_arima():
152     df = JolpicaAPI(resource_type="laps", filters={"season": "2023"
            , "round": "6"}).get_cleaned_data()
153     df = convert_to_ms(df)
154     df_max = df[df["Timings.driverId"] == "max_verstappen"]
155     save_to_csv(df_max, "arima.csv")
156
157     result = trend_analysis.arima_model(df_max, "Timings.time", (1,
             1, 1))
158     expected_result = [
159         84238.0, 84160.327, ... 85456.306, 85475.318
160     ]
161
162     assert result.tolist()[2:] == pytest.approx(expected_result
            [2:], abs=3300) # First two values ignored due to
            inconsistencies
163
164 def test_holt_winters():
165     df = JolpicaAPI(resource_type="laps", filters={"season": "2023"
            , "round": "6"}).get_cleaned_data()
166     df = convert_to_ms(df)
167     df_max = df[df["Timings.driverId"] == "max_verstappen"]
168     save_to_csv(df_max, "holt_winters.csv")
169
170     result = trend_analysis.holt_winters(df_max, "Timings.time")
171
172     expected_result = [
173         84238, 83068.96, ... 82777.5100460942, 81779.8457574504
174     ]
175
176     assert result.tolist()[:-3] == pytest.approx(expected_result
            [2:], abs=15000)
```

Listing H.3: test_analysis.py

pytest in test_analysis.py:
13 total, 13 passed

443 ms

Collapse | Expand

test_descriptive_analysis	passed	11 ms
PASSED [7%]		
test_paired_t_test	passed	56 ms
PASSED [15%]		
test_unpaired_t_test	passed	15 ms
PASSED [23%]		
test_anova	passed	20 ms
PASSED [30%]		
test_spearman_correlation	passed	34 ms
PASSED [38%]		
test_pearson_correlation	passed	24 ms
PASSED [46%]		
test_wilcoxon	passed	18 ms
PASSED [53%]		
test_chi_square	passed	36 ms
PASSED [61%]		
test_simple_moving_average	passed	7 ms
PASSED [69%]		
test_exponential_moving_average	passed	5 ms
PASSED [76%]		
test_linear_regression	passed	34 ms
PASSED [84%]		
test_arima	passed	74 ms
PASSED [92%]		
test_holt_winters	passed	109 ms
PASSED [100%]		

Generated by PyCharm on 17/04/2025, 15:34