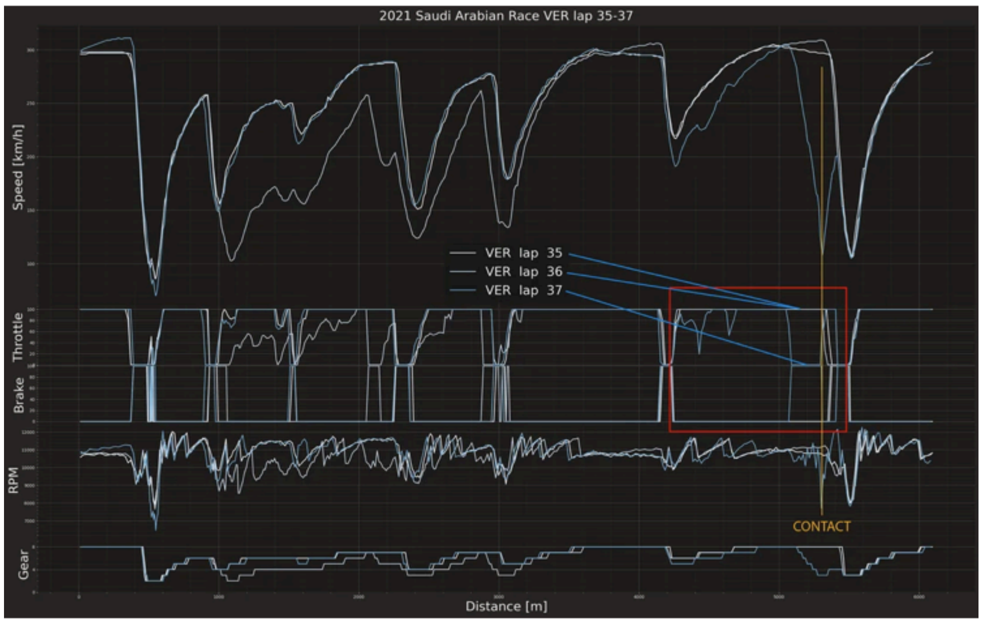
F0: A simplistic user focused Formula One Telemetry Tracker

Abisharan Nedun | ENAE380 | Final Project

F0 is a Formula One Telemetry Tracker built with Python, designed to make it easier for newcomers to enjoy watching Formula One (F1). The main goal of this project is to provide beginner-friendly insights into Grand Prix events by combining technical data with interactive features. It aims to bridge the gap between long-time F1 fans and new viewers, offering a simple way to understand the sport while showcasing how Python can be used for data analysis and user interface design.

This project was inspired by my passion for Formula One and my desire to help more people watch and enjoy the sport. F1 is a world-famous open-wheeled racing series with a rich history and a rapidly growing fan base, especially in the United States. Having followed the sport for almost a decade, I know first hand how exciting the races can be. However, many of my friends and new viewers find it hard to understand the strategies, technical details, and unique terminology that define the sport. The F0 project combines my love for F1 with technology to make the sport more accessible and engaging. By using Python, I aim to create a user interface that explains the complexities of the sport in simple terms. This tool not only helps users watch races but also understand and appreciate the exciting details that make Formula One so special.

Most current F1 telemetry systems are designed for technical enthusiasts, offering detailed data like sector times, lap charts, strategy overlays, and tire performance graphs. While these tools are useful for analysts and long term fans, they can be overwhelming or confusing for newer viewers. For example, sector time data—which shows performance in specific track sections—often lacks context for the average user and requires expertise to interpret. Even as a long-time fan, I find many of these systems provide so much detailed information that it becomes difficult to navigate. On top of that, most telemetry tools don’t cover important aspects like rules, regulations, penalty explanations, or flag meanings. While a few applications include these features, they are often paid services with limited data access. Attached below is an example of the confusing nature of these telemetry systems:



What the data is supposed to show is a brief contact period that Max Verstappen (one of the drivers) has during lap 37 of the 2021 Saudi Arabian Grand Prix. From this image it is clear that these telemetry trackers are not easy to read. Even more, this system fails to show the penalty call or how this result affected the final race.

By observing this blatant gap in user experience, where valuable information about the sport is hard to find without constantly searching online, highlighted a clear need for improvement. Filling this gap of accessibility became the foundation for the F0 project, setting its guidelines and focus on making Formula One data more accessible and easy to understand for everyone from brand new to experienced fans alike.

This project includes 3 main features that combined would make a valuable user experience. Firstly, the interface would provide detailed race summaries for every session during a Grand Prix weekend, including Free Practice (FP) sessions, Qualifying, and the Race. It also supports the newer sprint race format introduced in 2021, covering both Sprint Qualifying and Sprint Races. The app includes explanations of F1 regulations, such as car specifications, tire rules, race procedures, scoring systems, flags, and penalties, to help users understand the sport better. Additionally, it visualizes track data with marked maps showing corners and the start/finish line for the selected race. All of this is presented in a clean and simple interface that avoids overwhelming users, keeping the design both informative and easy to use.

Having these features is the main goal of this project which is to make F1 easy to watch and understand for everyone, especially new fans. Many people find the sport confusing because of its complex rules, strategies, and technical details. This app is designed to simplify all of that by presenting race data, rules, and track information in a way that is clear and straightforward. Instead of overwhelming users with complicated charts or too much information, it explains things step by step. Whether it’s showing the results of a race, explaining the different penalties a driver can receive, helping users understand the layout of a track, or even giving them information on the exact car specification the focus is always on making the sport more enjoyable and less intimidating for new viewers. Additionally, this eliminates the need to constantly search something up as the race is being watched, as all the relevant information is already present.

Building the F0 application required several important resources and came with its own set of challenges. The first key resource was the FastF1 API, which provided detailed race data, including lap times, session results, and telemetry information. This data was essential for creating the app’s features, like race summaries and track visualizations. Another resource was the glossary of F1 regulations, which helped explain technical concepts to new users. One of the biggest challenges was designing a user interface that was both simple and informative, as balancing these two goals required careful planning and testing. Additionally, understanding how to process and display the large amounts of race data in a clear way was tricky at times. Despite these challenges, the combination of the right tools, planning, and problem-solving helped bring this project to life.

The design process for the F0 telemetry tracker involved several important steps, ranging from planning and organizing ideas to solving technical challenges and adjusting the project's scope to make it achievable within the time allotted. One of the first things done was to set up a clear and organized structure for all files and documents, which was crucial for managing the project effectively. This was accomplished by creating a central directory named F0dir, which acted as the main folder containing all the necessary files and subdirectories. Organizing the project this way not only kept everything tidy but also gave a clear overview of what needed to be built and where specific elements would be stored. This preparation set a strong foundation for the development phase.

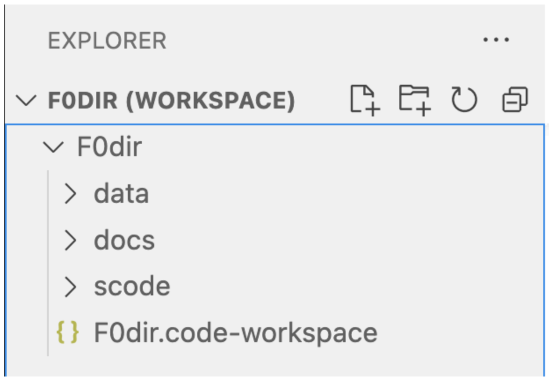
At the start, a few extra Python files were created as placeholders in case the project expanded beyond its initial scope. However, as the work progressed and the goals became clearer, the code was simplified and refined. Ultimately, three main folders were established within the F0dir directory: data, docs, and scode, each serving a specific purpose.

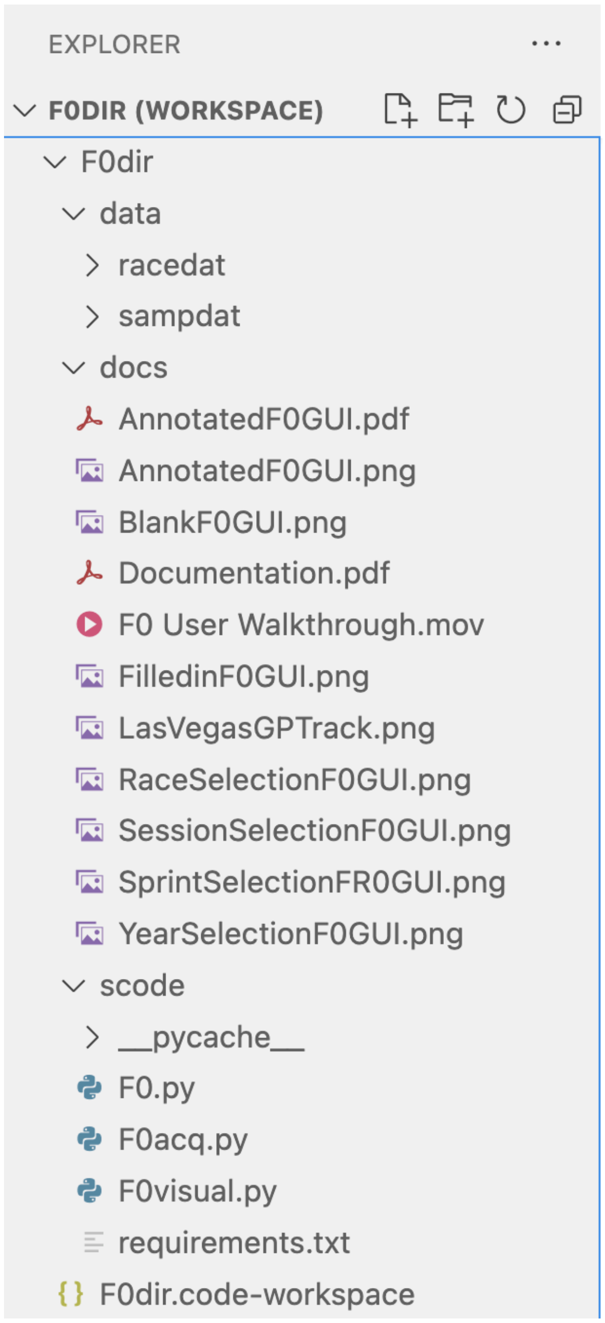
Firstly, the data folder was created to store all the race session data. This was done to make the project more efficient and organized. For instance, when a user loaded data for a specific race session, the information would be saved in this folder. Storing the data locally meant that if the same race session data was accessed again, it would load much faster since it didn't need to be re-fetched from the API. This not only improved performance but also made the user experience smoother.

Next, the docs folder was used as a repository for all documentation related to the project. It included this design document and other resources that could guide users or explain how the application works. Additionally, the docs folder contained multiple video walkthroughs created as part of the project. One of these videos provided a short, simple guide on how to use the final user interface, making it easy for anyone to get started with the app. The other two videos were more detailed and focused on explaining the design decisions and code structure, offering insights into the development process for those interested in the technical aspects.

Lastly, the scode folder was reserved for the main Python code that powered the application. Initially, there were several Python files created as part of the brainstorming process, but these were eventually narrowed down to three core scripts: F0.py, F0visual.py, and F0acq.py. Each of these scripts played a vital role in the project. F0.py handled the core functionality, F0visual.py managed the graphical user interface, and F0acq.py focused on fetching and processing data from the FastF1 API. Additionally a requirements.txt file was added to get the required Python libraries used for this project even easier for the user. Explained further in the Documentation.pdf file, all the user needs to type is "pip3 install -r requirements.txt" which makes setting up the interface incredibly convenient. Together, these files formed the backbone of the application and ensured that all the main features worked seamlessly.

This structured approach to organizing the project made the development process smoother and more manageable. By clearly separating data, documentation, and code, the project not only stayed organized but also became easier to debug, test, and expand. This thoughtful planning and organization ensured that every part of the project, from storing race data to explaining the design through documentation, was well-integrated and accessible. To showcase this visually, attached below are images of the explorer view the user can utilize in vscode:


EXPLORER ...
F0DIR (WORKSPACE)
```
F0dir
  data
  docs
  scode
  F0dir.code-workspace
```


EXPLORER ...
F0DIR (WORKSPACE)
```
F0dir
  data
    racedat
    sampdat
  docs
    AnnotatedF0GUI.pdf
    AnnotatedF0GUI.png
    BlankF0GUI.png
    Documentation.pdf
    F0 User Walkthrough.mov
    FilledinF0GUI.png
    LasVegasGPTrack.png
    RaceSelectionF0GUI.png
    SessionSelectionF0GUI.png
    SprintSelectionFR0GUI.png
    YearSelectionF0GUI.png
  scode
    __pycache__
    F0.py
    F0acq.py
    F0visual.py
    requirements.txt
  F0dir.code-workspace
```

The explorer view in the F0dir workspace is neatly organized into three main folders: data, docs, and scode, making it easy to locate and manage files. The data folder is further divided into subdirectories like racedat and sampdat, ensuring that race session data is clearly separated for better access and performance. The docs folder contains all supporting documents, such as user guides, design files, and visual aids like annotated GUI images and walkthrough videos, providing a comprehensive reference for users. Finally, the scode folder houses the primary Python scripts (F0.py, F0acq.py, and F0visual.py) and the requirements file, keeping the core functionality of the project centralized and streamlined. This thoughtful structure not only improves workflow but also supports scalability

The original prototype of the F0 telemetry tracker was a minimal version designed to test the project's principal idea. It relied entirely on the user interacting through the terminal, where they would input data and view all outputs. This approach was deliberately chosen to keep things simple and focus on understanding how to process and display the necessary information. At the time, the project had many end goals but no clear way to reach said goal, so this terminal-based version served as a starting point to experiment with different ideas and lay the foundation. This method also served as a way to plot the timeline for the project, as there was no clear indication for how long certain features would take to properly implement.

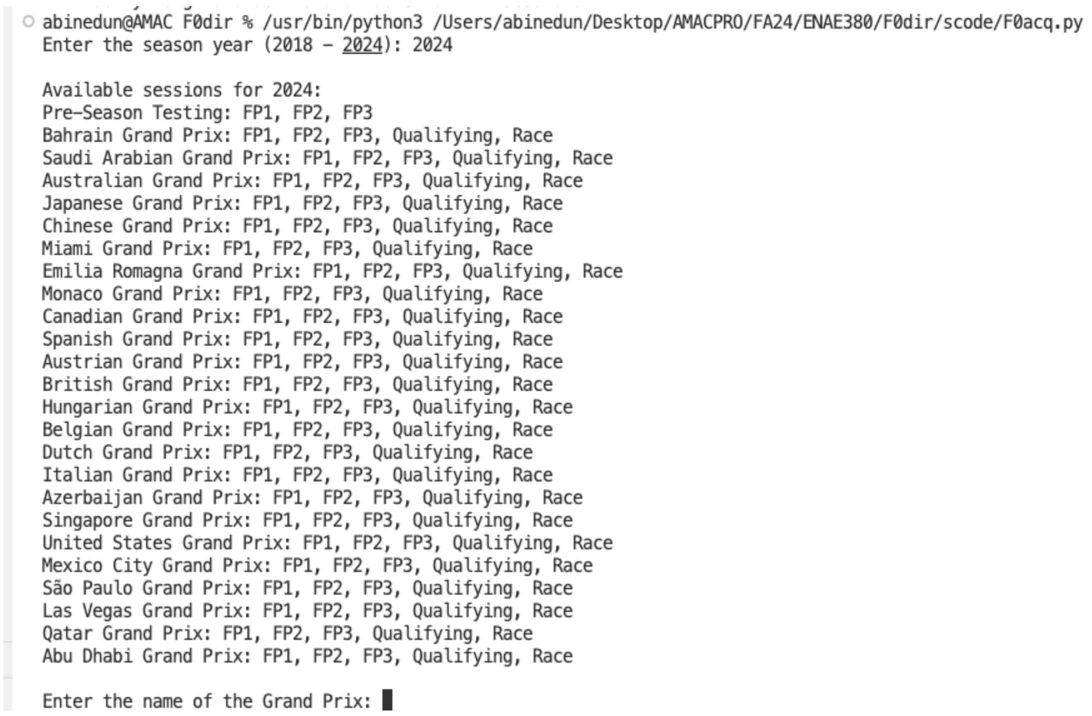
After doing initial research on the Fast F1 API, which provides race session data, it was essential to building this prototype and eventually the final user interface. The detailed GitHub forums were invaluable for troubleshooting and understanding how to retrieve session information, since many people have attempted something similar in the past. The prototype included functionality to gather data from Free Practice (FP1, FP2, FP3), Qualifying, and Race sessions. However, a major discovery during this phase was that the API only provided complete data for seasons starting from 2018, as earlier seasons lacked sufficient detail in certain sessions. Also F1 Sprints, which were a slight alteration of the regular Grand Prix race weekend which was introduced in 2021, posed as a difficult challenge. Initially, the API handled these events inconsistently, requiring trial and error to resolve issues. Fortunately, a helpful guide on the forums provided insights on how to make sprint data work correctly. When observing the code, there needed to be two instances of race data to be created in case an event was a sprint race.

Going back to F0acq.py one of the first key functions implemented was availsessions, which prompted users to enter a season between 2018 and 2024. The function fetched and displayed all available race sessions for that year. For example, sprint race weekends were identified using the event format and displayed unique sessions like Sprint Qualifying and Sprint Race. Error handling, such as try and except blocks, was heavily used to debug the process and ensure that invalid inputs didn't crash the program. Going over the code, these blocks proved to be invaluable, as initially progress was quite slow but eventually there was a way to collect the needed data. For the terminal demonstration, the blocks were even more crucial as user inputs needed to be exact for there not to be an accidental error, and the code to crash. Finally after many hours of testing and debugging The output provided users with a clean list of all available sessions for their chosen year.

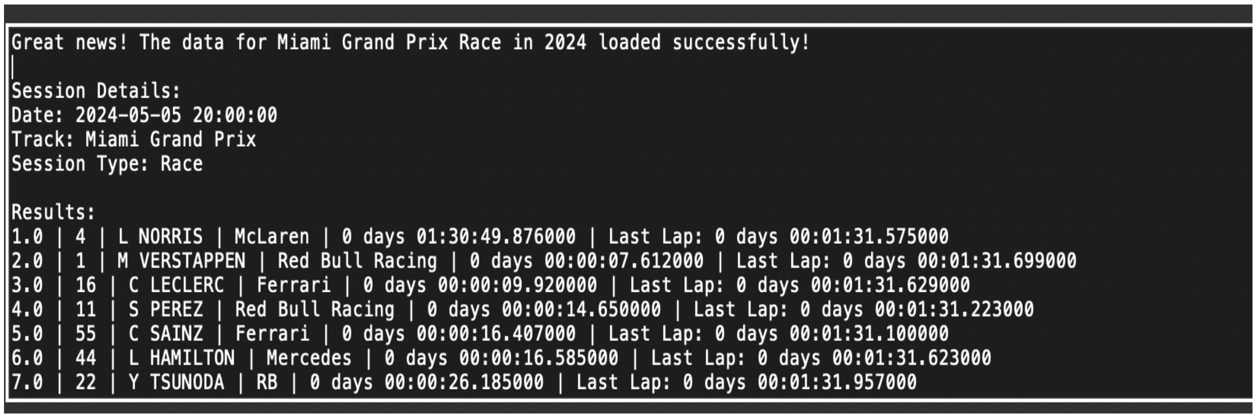
Creating another function, loadrdata, allowed users to load data for specific sessions, like the final grid positions or starting order for a race. This function included features to adapt sprint qualifiers to the session structure and tackled issues with Free Practice data, which the API stores differently. Otherwise there would be two identical data sets, leading to a serious error. Additionally, debugging Free Practice data revealed that it acted more like warmups, requiring custom handling to process correctly. Since the API did not organize the Free Practice data, there was an added function to sort them by their fastest lap times. The values in the Free Practice data are largely inconsequential, yet the option to view them in a similar manner to the qualifying and race data was crucial to the project.

Also, the prototype also laid the groundwork for advanced features like track layouts and telemetry data. Functions such as get_track_layout retrieved sector boundaries and .finish line positions. These early implementations were used to show the potential of the project, even if they weren't yet fully polished at the time.

In the end, the terminal-based prototype successfully demonstrated the core functionality of the F0 telemetry tracker. It allowed users to retrieve session data, load key race details, and test telemetry features. Although simple, this version proved the concept and provided a solid base for expanding the project into a full user interface. Although the final design required that some implementation of the terminal application would stop otherwise leading to constant errors, it was an essential starting point to then develop the full user interface. Attached below is how the terminal application visually looked. Using the example of the 2024 Miami Grand Prix.



```
abinedun@AMAC F0dir % /usr/bin/python3 /Users/abinedun/Desktop/AMACPRO/FA24/ENAE380/F0dir/scode/F0acq.py
Enter the season year (2018 - 2024): 2024

Available sessions for 2024:
Pre-Season Testing: FP1, FP2, FP3
Bahrain Grand Prix: FP1, FP2, FP3, Qualifying, Race
Saudi Arabian Grand Prix: FP1, FP2, FP3, Qualifying, Race
Australian Grand Prix: FP1, FP2, FP3, Qualifying, Race
Japanese Grand Prix: FP1, FP2, FP3, Qualifying, Race
Chinese Grand Prix: FP1, FP2, FP3, Qualifying, Race
Miami Grand Prix: FP1, FP2, FP3, Qualifying, Race
Emilia Romagna Grand Prix: FP1, FP2, FP3, Qualifying, Race
Monaco Grand Prix: FP1, FP2, FP3, Qualifying, Race
Canadian Grand Prix: FP1, FP2, FP3, Qualifying, Race
Spanish Grand Prix: FP1, FP2, FP3, Qualifying, Race
Austrian Grand Prix: FP1, FP2, FP3, Qualifying, Race
British Grand Prix: FP1, FP2, FP3, Qualifying, Race
Hungarian Grand Prix: FP1, FP2, FP3, Qualifying, Race
Belgian Grand Prix: FP1, FP2, FP3, Qualifying, Race
Dutch Grand Prix: FP1, FP2, FP3, Qualifying, Race
Italian Grand Prix: FP1, FP2, FP3, Qualifying, Race
Azerbaijan Grand Prix: FP1, FP2, FP3, Qualifying, Race
Singapore Grand Prix: FP1, FP2, FP3, Qualifying, Race
United States Grand Prix: FP1, FP2, FP3, Qualifying, Race
Mexico City Grand Prix: FP1, FP2, FP3, Qualifying, Race
São Paulo Grand Prix: FP1, FP2, FP3, Qualifying, Race
Las Vegas Grand Prix: FP1, FP2, FP3, Qualifying, Race
Qatar Grand Prix: FP1, FP2, FP3, Qualifying, Race
Abu Dhabi Grand Prix: FP1, FP2, FP3, Qualifying, Race

Enter the name of the Grand Prix:
```



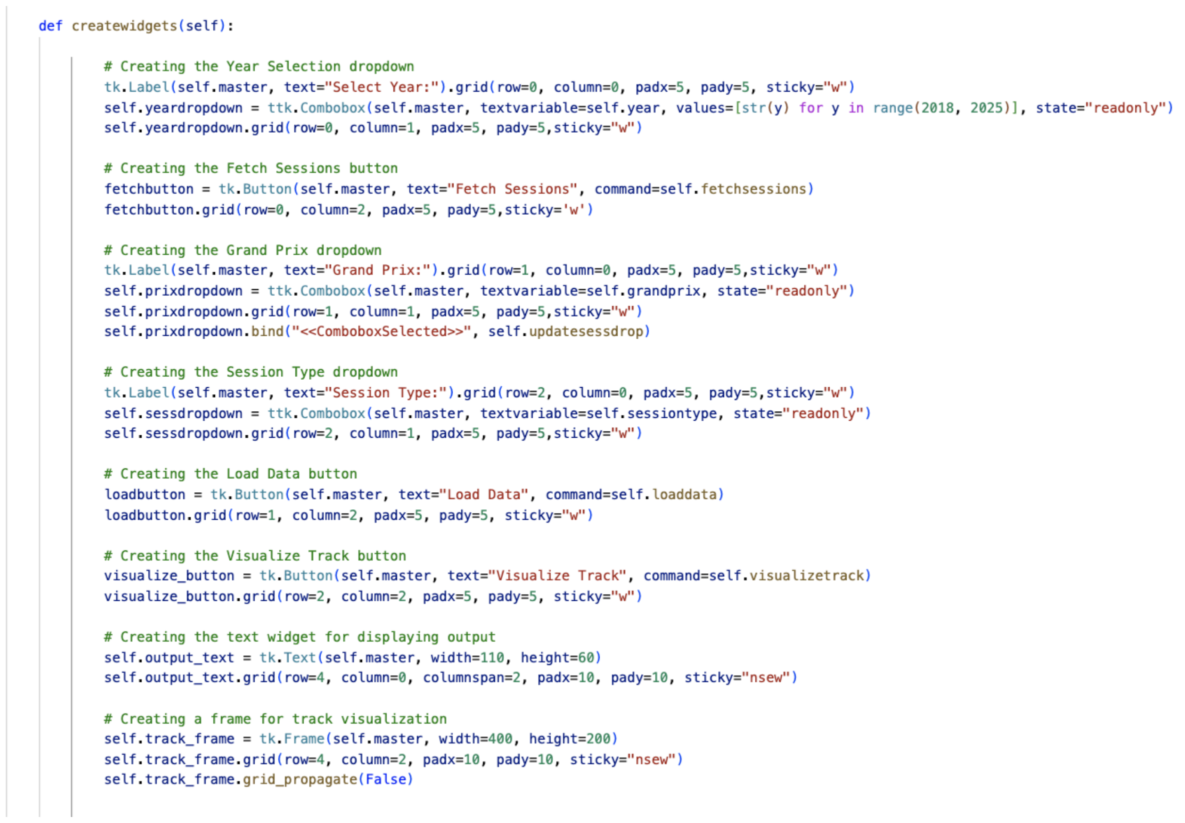
```
Great news! The data for Miami Grand Prix Race in 2024 loaded successfully!

Session Details:
Date: 2024-05-05 20:00:00
Track: Miami Grand Prix
Session Type: Race

Results:
1.0 | 4 | L NORRIS | McLaren | 0 days 01:30:49.876000 | Last Lap: 0 days 00:01:31.575000
2.0 | 1 | M VERSTAPPEN | Red Bull Racing | 0 days 00:00:07.612000 | Last Lap: 0 days 00:01:31.699000
3.0 | 16 | C LECLERC | Ferrari | 0 days 00:00:09.920000 | Last Lap: 0 days 00:01:31.629000
4.0 | 11 | S PEREZ | Red Bull Racing | 0 days 00:00:14.650000 | Last Lap: 0 days 00:01:31.223000
5.0 | 55 | C SAINZ | Ferrari | 0 days 00:00:16.407000 | Last Lap: 0 days 00:01:31.100000
6.0 | 44 | L HAMILTON | Mercedes | 0 days 00:00:16.585000 | Last Lap: 0 days 00:01:31.623000
7.0 | 22 | Y TSUNODA | RB | 0 days 00:00:26.185000 | Last Lap: 0 days 00:01:31.957000
```

With the concept working successfully, the next step was to create a proper user interface. The terminal-based version of the telemetry tracker was useful for testing the idea, but it felt too plain and unfinished. It didn't provide the polished or user-friendly experience needed if the project was going to improve. This made it clear that moving to a graphical interface was the best way to make the project better and easier to use.

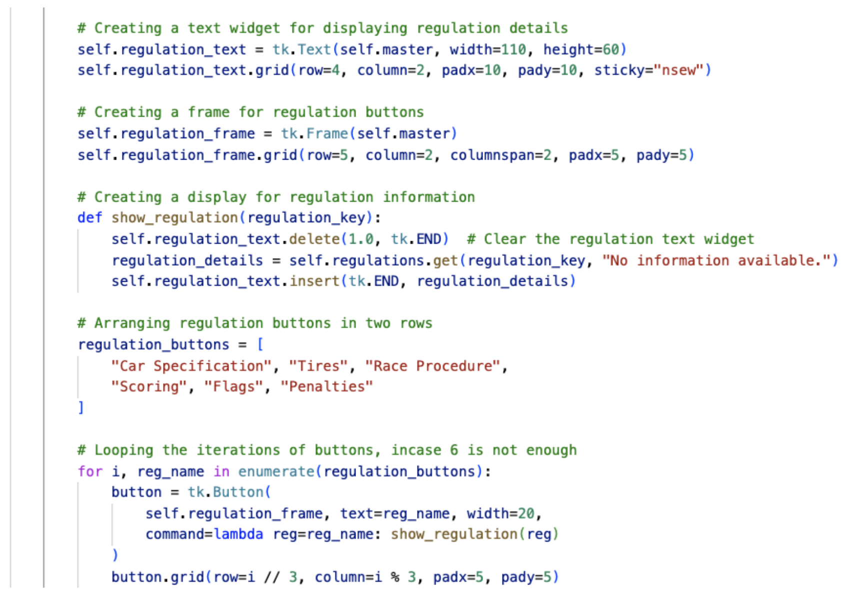
To implement the user interface, the tkinter Python library was chosen. While there are more advanced tools for creating graphical user interfaces (GUIs), tkinter stood out as the simplest option to get started quickly. This decision led to the creation of the F0visual.py file, which was designed to display the data collected from the Foacq.py file in an interactive format.

The first step in building the interface was developing the createwidgets function. This function handled the creation of all the key elements in the GUI, such as buttons, dropdown menus, and text areas. Surprisingly, setting up these basic components was relatively straightforward. Though the early version's buttons lacked functionality, it successfully displayed a basic window when the code was run. This proof of concept was simple but effective, providing a solid foundation for further development.

Over time, more features were added as the project evolved. Buttons were linked to specific actions, dropdowns were populated with race and session data, and the interface gradually became more interactive and functional through continuous testing and updates. Below is the finalized version of the createwidgets function.

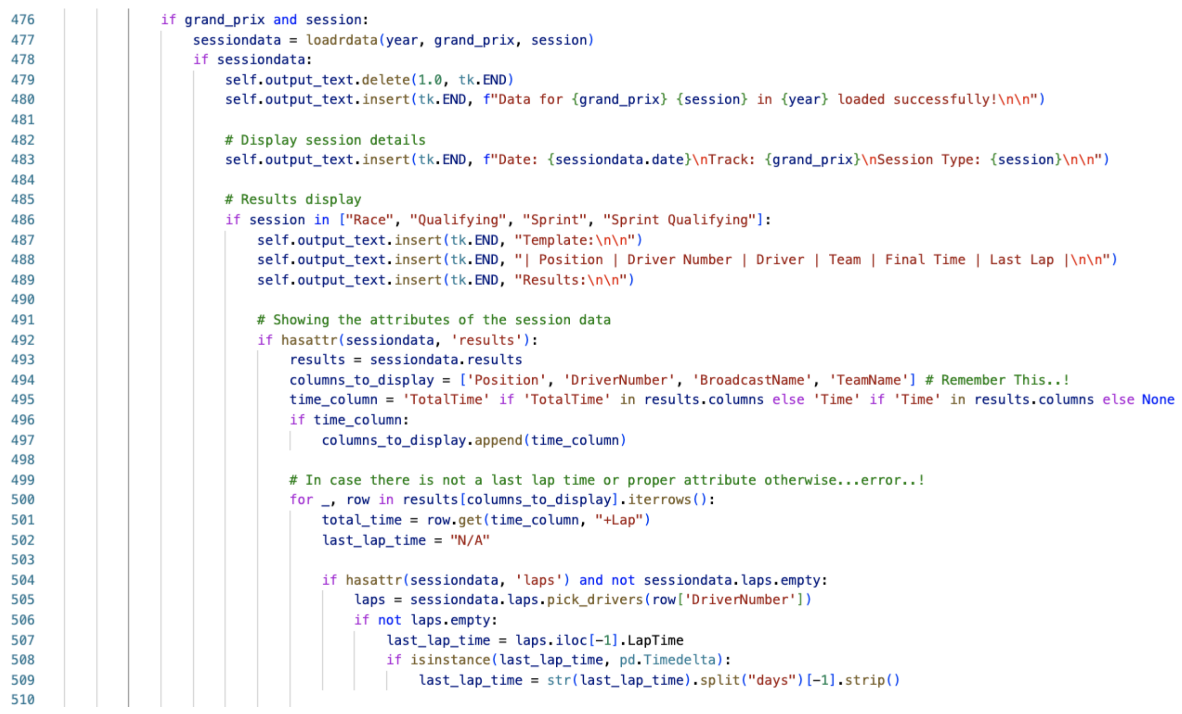


```python
def createwidgets(self):

        # Creating the Year Selection dropdown
        tk.Label(self.master, text="Select Year:").grid(row=0, column=0, padx=5, pady=5, sticky="w")
        self.yeardropdown = ttk.Combobox(self.master, textvariable=self.year, values=[str(y) for y in range(2018, 2025)], state="readonly")
        self.yeardropdown.grid(row=0, column=1, padx=5, pady=5,sticky="w")

        # Creating the Fetch Sessions button
        fetchbutton = tk.Button(self.master, text="Fetch Sessions", command=self.fetchsessions)
        fetchbutton.grid(row=0, column=2, padx=5, pady=5,sticky='w')

        # Creating the Grand Prix dropdown
        tk.Label(self.master, text="Grand Prix:").grid(row=1, column=0, padx=5, pady=5,sticky="w")
        self.prixdropdown = ttk.Combobox(self.master, textvariable=self.grandprix, state="readonly")
        self.prixdropdown.grid(row=1, column=1, padx=5, pady=5,sticky="w")
        self.prixdropdown.bind("<<ComboboxSelected>>", self.updatesessdrop)

        # Creating the Session Type dropdown
        tk.Label(self.master, text="Session Type:").grid(row=2, column=0, padx=5, pady=5,sticky="w")
        self.sessdropdown = ttk.Combobox(self.master, textvariable=self.sessiontype, state="readonly")
        self.sessdropdown.grid(row=2, column=1, padx=5, pady=5,sticky="w")

        # Creating the Load Data button
        loadbutton = tk.Button(self.master, text="Load Data", command=self.loaddata)
        loadbutton.grid(row=1, column=2, padx=5, pady=5, sticky="w")

        # Creating the Visualize Track button
        visualize_button = tk.Button(self.master, text="Visualize Track", command=self.visualizetrack)
        visualize_button.grid(row=2, column=2, padx=5, pady=5, sticky="w")

        # Creating the text widget for displaying output
        self.output_text = tk.Text(self.master, width=110, height=60)
        self.output_text.grid(row=4, column=0, columnspan=2, padx=10, pady=10, sticky="nsew")

        # Creating a frame for track visualization
        self.track_frame = tk.Frame(self.master, width=400, height=200)
        self.track_frame.grid(row=4, column=2, padx=10, pady=10, sticky="nsew")
        self.track_frame.grid_propagate(False)
```

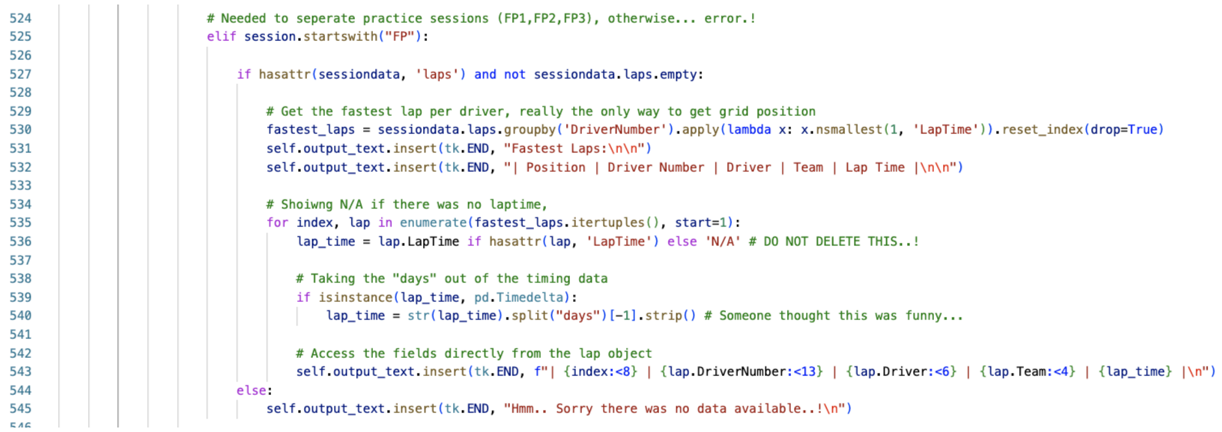


```python
# Creating a text widget for displaying regulation details
self.regulation_text = tk.Text(self.master, width=110, height=60)
self.regulation_text.grid(row=4, column=2, padx=10, pady=10, sticky="nsew")

# Creating a frame for regulation buttons
self.regulation_frame = tk.Frame(self.master)
self.regulation_frame.grid(row=5, column=2, columnspan=2, padx=5, pady=5)

# Creating a display for regulation information
def show_regulation(regulation_key):
    self.regulation_text.delete(1.0, tk.END)  # Clear the regulation text widget
    regulation_details = self.regulations.get(regulation_key, "No information available.")
    self.regulation_text.insert(tk.END, regulation_details)

# Arranging regulation buttons in two rows
regulation_buttons = [
    "Car Specification", "Tires", "Race Procedure",
    "Scoring", "Flags", "Penalties"
]

# Looping the iterations of buttons, incase 6 is not enough
for i, reg_name in enumerate(regulation_buttons):
    button = tk.Button(
        self.regulation_frame, text=reg_name, width=20,
        command=lambda reg=reg_name: show_regulation(reg)
    )
    button.grid(row=i // 3, column=i % 3, padx=5, pady=5)
```

Following the create widgets function, the GUI needs to be populated with data. This is where the loaddata function came in, serving as an important part of the GUI. It was created to fetch and display session data from the F0acq.py file and initially served as a template for presenting the information in the interface. Over time, the function was improved to handle different types of sessions, such as races, qualifying, sprint races, and free practice sessions. Each session type has its own unique details, so the function needed to be flexible to show the right data for each race session. For example, sprint race data had to be handled separately from regular races to avoid data confusion, and free practice sessions required their own format since they focus more on who completed the laps in no particular order. Attached below is how the loaddata function was modified to allow both F1 Sprint and Free Practice events:



```python
if grand_prix and session:
    sessiondata = loadrdata(year, grand_prix, session)
    if sessiondata:
        self.output_text.delete(1.0, tk.END)
        self.output_text.insert(tk.END, f"Data for {grand_prix} {session} in {year} loaded successfully!\n\n")

        # Display session details
        self.output_text.insert(tk.END, f"Date: {sessiondata.date}\nTrack: {grand_prix}\nSession Type: {session}\n\n")

        # Results display
        if session in ["Race", "Qualifying", "Sprint", "Sprint Qualifying"]:
            self.output_text.insert(tk.END, "Template:\n\n")
            self.output_text.insert(tk.END, "| Position | Driver Number | Driver | Team | Final Time | Last Lap |\n\n")
            self.output_text.insert(tk.END, "Results:\n\n")

            # Showing the attributes of the session data
            if hasattr(sessiondata, 'results'):
                results = sessiondata.results
                columns_to_display = ['Position', 'DriverNumber', 'BroadcastName', 'TeamName'] # Remember This..!
                time_column = 'TotalTime' if 'TotalTime' in results.columns else 'Time' if 'Time' in results.columns else None
                if time_column:
                    columns_to_display.append(time_column)

                # In case there is not a last lap time or proper attribute otherwise...error..!
                for _, row in results[columns_to_display].iterrows():
                    total_time = row.get(time_column, "+Lap")
                    last_lap_time = "N/A"

                    if hasattr(sessiondata, 'laps') and not sessiondata.laps.empty:
                        laps = sessiondata.laps.pick_drivers(row['DriverNumber'])
                        if not laps.empty:
                            last_lap_time = laps.iloc[-1].LapTime
                            if isinstance(last_lap_time, pd.Timedelta):
                                last_lap_time = str(last_lap_time).split("days")[-1].strip()

```



```python
# Needed to seperate practice sessions (FP1,FP2,FP3), otherwise... error.!
elif session.startswith("FP"):

    if hasattr(sessiondata, 'laps') and not sessiondata.laps.empty:

        # Get the fastest lap per driver, really the only way to get grid position
        fastest_laps = sessiondata.laps.groupby('DriverNumber').apply(lambda x: x.nsmallest(1, 'LapTime')).reset_index(drop=True)
        self.output_text.insert(tk.END, "Fastest Laps:\n\n")
        self.output_text.insert(tk.END, "| Position | Driver Number | Driver | Team | Lap Time |\n\n")

        # Shoiwng N/A if there was no laptime,
        for index, lap in enumerate(fastest_laps.itertuples(), start=1):
            lap_time = lap.LapTime if hasattr(lap, 'LapTime') else 'N/A' # DO NOT DELETE THIS..!

            # Taking the "days" out of the timing data
            if isinstance(lap_time, pd.Timedelta):
                lap_time = str(lap_time).split("days")[-1].strip() # Someone thought this was funny...

            # Access the fields directly from the lap object
            self.output_text.insert(tk.END, f"| {index:<8} | {lap.DriverNumber:<13} | {lap.Driver:<6} | {lap.Team:<4} | {lap_time} |\n")
    else:
        self.output_text.insert(tk.END, "Hmm.. Sorry there was no data available..!\n")
```

From this, the collection of data had to be slightly modified due to these events being held differently in the Fast F1 API. When developing the function several other features were added to improve functionality. Firstly was to ensure the timing data is clean and properly formatted by removing the day from time values. If no results are available, the function informs the user with the message "Hmm.. Sorry there was no data available". This feature became invaluable as it narrowed down errors when handling data.

Going back to the Free practice sessions, they are handled differently because being sorted by the fastest time, they are just sorted by when the driver finishes the session. To make the session data more interesting than the random collection of drivers, the function displays the fastest laps for each driver. The function does this by processing the lap data during the session to find each driver's best lap and then displays it in a simple table. If there are no laps recorded for the session, the user is notified that no data is available. This was critical as it led to many previous errors. This separation ensures the practice data is presented in a way that makes sense for users.

The function also provides general session statistics, such as the total number of laps and drivers, to give users additional insights. It includes error handling to catch issues like missing inputs or failed data loads. If unexpected errors occur, they are printed in the text area along with debugging information for troubleshooting. This helped greatly in the developing stage, the user most likely won't ever see these messages in the GUI since the data is stable in the finalized version.

Overall, the loaddata function plays a central role in making the GUI user-friendly and informative. By adapting to different session types and handling various scenarios, it ensures that users can easily access and understand the data they are interested in. Its improvements over time have made it a reliable and essential part of the Formula One telemetry tracker project.

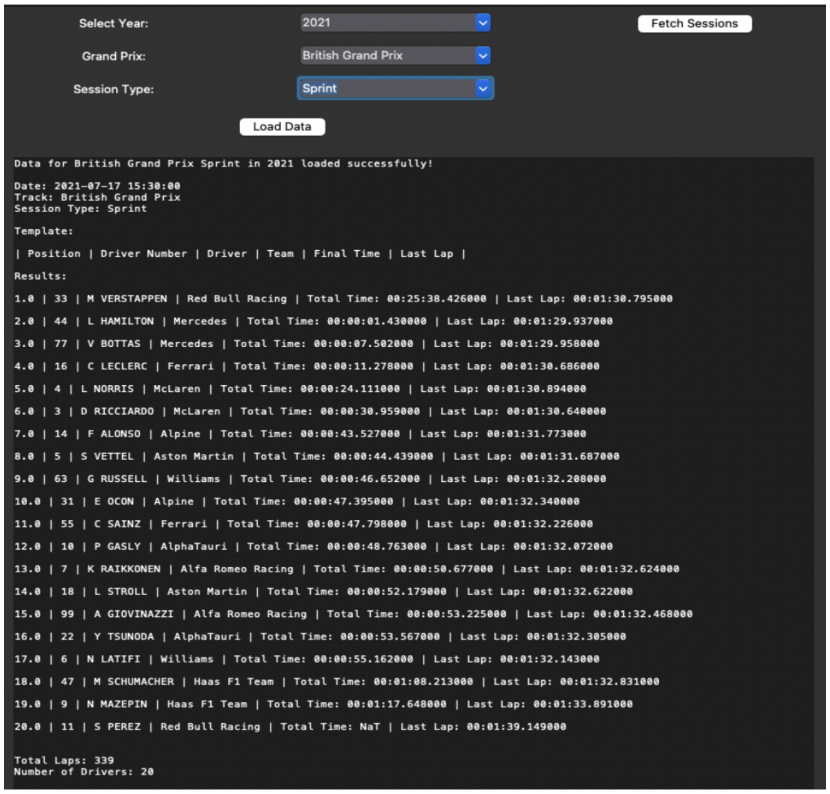
Before the loaddata function could populate the GUI widgets with session data, two additional functions—popsessdictand updatesessdrop—needed to be created. These functions work together to ensure that the dropdown menus in the interface are dynamically updated based on the user's selections. This step was essential to make the GUI interactive and responsive to different race weekends and session types.

The popsessdict function is responsible for creating a dictionary of available sessions for each Grand Prix in a selected year. It pulls data using the FastF1 library's get_event_schedule method, which provides the schedule for all events in a given Formula One season. By looping through each event in the schedule, the function identifies the name of the Grand Prix and whether it follows the sprint race format. This distinction is important because sprint race weekends have a different structure than regular weekends. For sprint weekends, the available sessions include FP1, Sprint Qualifying, Sprint, Qualifying, and Race, while regular weekends typically include FP1, FP2, FP3, Qualifying, and Race.

An additional step ensures that only sessions with valid dates are included in the list. This avoids errors that might occur if a session lacks a scheduled date. After processing, the function populates the sessionsdict dictionary with the Grand Prix names as keys and their respective session lists as values. This dictionary serves as the foundation for dynamically updating the session dropdown in the GUI.

The updatesessdrop function uses the data generated by popsessdict to update the session dropdown menu in the interface. When a user selects a specific Grand Prix from the dropdown menu, this function retrieves the list of available sessions for that event from the sessionsdict dictionary. It then updates the session dropdown with these options, ensuring that the displayed sessions are accurate for the selected Grand Prix. If there are sessions available, the function automatically sets the first option as the default selection.

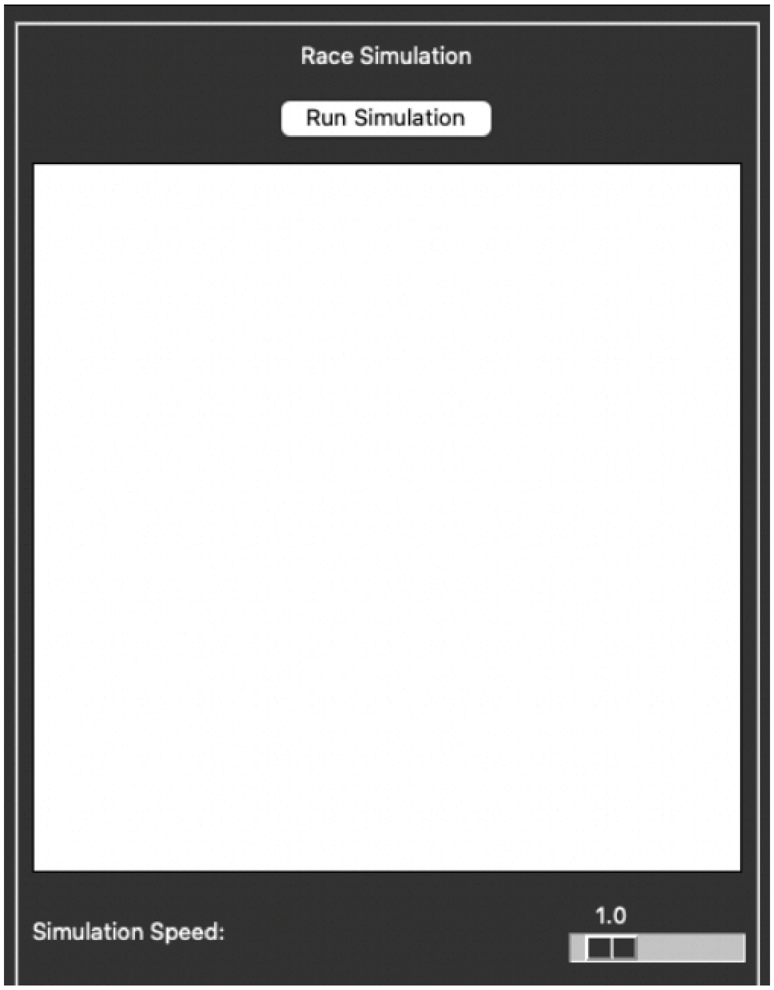
Together, these two functions ensure the GUI remains accurate and user-friendly. The popsessdict function prepares the data by creating a clear structure for session availability, while the updatesessdrop function ensures this information is correctly displayed in the interface. Attached below is what the GUI looked like after implementing these functions:


Select Year: 2021
Grand Prix: British Grand Prix
Session Type: Sprint

```
Data for British Grand Prix Sprint in 2021 loaded successfully!

Date: 2021-07-17 15:30:00
Track: British Grand Prix
Session Type: Sprint

Template:

| Position | Driver Number | Driver | Team | Final Time | Last Lap |

Results:

1.0 | 33 | M VERSTAPPEN | Red Bull Racing | Total Time: 00:25:38.426000 | Last Lap: 00:01:30.795000

2.0 | 44 | L HAMILTON | Mercedes | Total Time: 00:00:01.430000 | Last Lap: 00:01:29.937000

3.0 | 77 | V BOTTAS | Mercedes | Total Time: 00:00:07.502000 | Last Lap: 00:01:29.958000

4.0 | 16 | C LECLERC | Ferrari | Total Time: 00:00:11.278000 | Last Lap: 00:01:30.686000

5.0 | 4 | L NORRIS | McLaren | Total Time: 00:00:24.111000 | Last Lap: 00:01:30.894000

6.0 | 3 | D RICCIARDO | McLaren | Total Time: 00:00:30.959000 | Last Lap: 00:01:30.640000

7.0 | 14 | F ALONSO | Alpine | Total Time: 00:00:43.527000 | Last Lap: 00:01:31.773000

8.0 | 5 | S VETTEL | Aston Martin | Total Time: 00:00:44.439000 | Last Lap: 00:01:31.687000

9.0 | 63 | G RUSSELL | Williams | Total Time: 00:00:46.652000 | Last Lap: 00:01:32.208000

10.0 | 31 | E OCON | Alpine | Total Time: 00:00:47.395000 | Last Lap: 00:01:32.340000

11.0 | 55 | C SAINZ | Ferrari | Total Time: 00:00:47.798000 | Last Lap: 00:01:32.226000

12.0 | 10 | P GASLY | AlphaTauri | Total Time: 00:00:48.763000 | Last Lap: 00:01:32.072000

13.0 | 7 | K RAIKKONEN | Alfa Romeo Racing | Total Time: 00:00:50.677000 | Last Lap: 00:01:32.624000

14.0 | 18 | L STROLL | Aston Martin | Total Time: 00:00:52.179000 | Last Lap: 00:01:32.622000

15.0 | 99 | A GIOVINAZZI | Alfa Romeo Racing | Total Time: 00:00:53.225000 | Last Lap: 00:01:32.468000

16.0 | 22 | Y TSUNODA | AlphaTauri | Total Time: 00:00:53.567000 | Last Lap: 00:01:32.305000

17.0 | 6 | N LATIFI | Williams | Total Time: 00:00:55.162000 | Last Lap: 00:01:32.143000

18.0 | 47 | M SCHUMACHER | Haas F1 Team | Total Time: 00:01:08.213000 | Last Lap: 00:01:32.831000

19.0 | 9 | N MAZEPIN | Haas F1 Team | Total Time: 00:01:17.648000 | Last Lap: 00:01:33.891000

20.0 | 11 | S PEREZ | Red Bull Racing | Total Time: NaT | Last Lap: 00:01:39.149000


Total Laps: 339
Number of Drivers: 20
```

Using these three functions the GUI could now display all race session data from 2018 to 2024. The website already provides the same information, but faster and with a better interface. This made me think of adding a new feature to make the project stand out: a race simulation in the GUI, where the race session would be shown as it happened.

However, the idea seemed good at first, but it quickly became clear that running a multi-hour race simulation was not possible, as the code had trouble handling such a big task.. One solution was to let users control the speed of the simulation, so they could decide how fast the race would play. Unfortunately, even with this, there was still a big problem: the FastF1 API didn't provide enough real-time data on all drivers to make a realistic race simulation. Without enough detailed data, it wasn't possible to create a proper Formula One race simulation. Attached was the widget that handled the race simulation and speed setting:



Although the race simulation feature couldn't be completed, it revealed an important limitation of the project and the available data. After reconsidering the feature, the idea shifted from creating a full race simulation, something that was not possible with the given data, to displaying an image of the track with relevant information, such as marking the corners and the start/finish line.

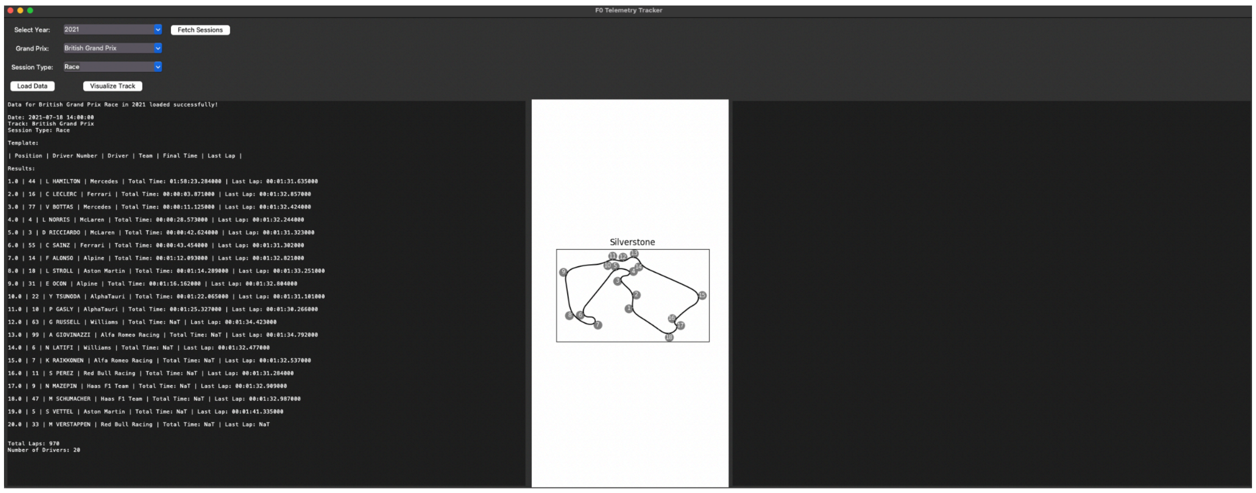
This sparked the idea behind the visualizetrack function, as the GUI needed to enhance the user experience with some type of visual engagement. Although the function itself is straightforward, it serves as an effective and essential feature in the project.

As the function is called, it first retrieves the year, Grand Prix event, and session type selected by the user from the GUI fields. These inputs are necessary to generate the correct track map. If any of the inputs are missing, a warning message is shown, reminding the user to fill in all required fields before proceeding. This ensures that the program has all the information it needs to generate the visualization. This process is highlighted further in the documentation pdf file.

The function also closes any existing figures. This prevents conflicts when a new track map is generated, ensuring the user always sees the latest track. The function then calls the visualize_track function which is gathered in the F0acq.py file, passing in the selected year, event, and session type. If visualize_track returns None or encounters an error, an error message is displayed, letting the user know that the track map could not be created. These error messages were necessary to ensure the program did not constantly crash.

Since the focus shifted towards creating a static but detailed visualization of a Formula One track, it highlights key elements like the corners, start/finish line, and other track-specific details. By presenting this information visually, users gain a clearer understanding of the race environment. This visual cue is especially important when watching a race since frequency camera cutbacks and commentary would leave most people confused on where the cars are on the track. Additionally, by selecting the season the user will be able to see subtle changes in some tracks over the years.

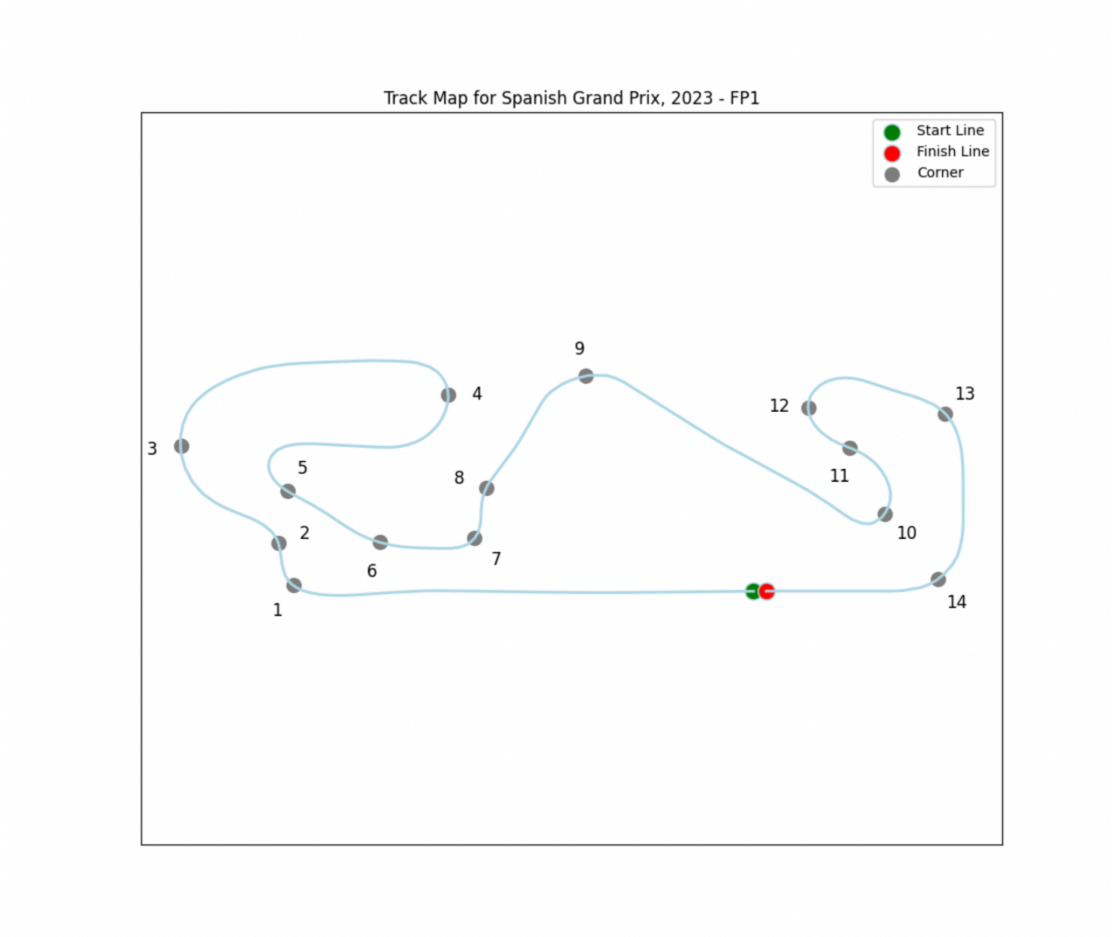
Now that the development of the function had been completed, what remained was to implement this into the GUI. Initially, however, the track visualization population into another text widget into the existing GUI. This approach was much easier to code, though led to formatting issues down the road. Additionally, this would cause the GUI to crash loading larger tracks, primarily why an effort was made to push the track into a new figure. Attached below was one of the first examples of inserting the track visualization into the GUI.



Select Year: 2021
Grand Prix: British Grand Prix
Session Type: Race

```
Data for British Grand Prix Race in 2021 loaded successfully!

Date: 2021-07-18 14:00:00
Track: British Grand Prix
Session Type: Race

Template:

| Position | Driver Number | Driver | Team | Final Time | Last Lap |

Results:

1.0 | 44 | L HAMILTON | Mercedes | Total Time: 01:58:23.284000 | Last Lap: 00:01:31.635000

2.0 | 16 | C LECLERC | Ferrari | Total Time: 00:00:03.871000 | Last Lap: 00:01:32.857000

3.0 | 77 | V BOTTAS | Mercedes | Total Time: 00:00:11.125000 | Last Lap: 00:01:32.424000

4.0 | 4 | L NORRIS | McLaren | Total Time: 00:00:28.573000 | Last Lap: 00:01:32.244000

5.0 | 3 | D RICCIARDO | McLaren | Total Time: 00:00:42.624000 | Last Lap: 00:01:31.323000

6.0 | 55 | C SAINZ | Ferrari | Total Time: 00:00:43.454000 | Last Lap: 00:01:31.302000

7.0 | 14 | F ALONSO | Alpine | Total Time: 00:01:12.093000 | Last Lap: 00:01:32.821000

8.0 | 18 | L STROLL | Aston Martin | Total Time: 00:01:14.289000 | Last Lap: 00:01:33.251000

9.0 | 31 | E OCON | Alpine | Total Time: 00:01:16.162000 | Last Lap: 00:01:32.804000

10.0 | 22 | Y TSUNODA | AlphaTauri | Total Time: 00:01:22.065000 | Last Lap: 00:01:31.101000

11.0 | 10 | P GASLY | AlphaTauri | Total Time: 00:01:25.327000 | Last Lap: 00:01:30.266000

12.0 | 63 | G RUSSELL | Williams | Total Time: NaT | Last Lap: 00:01:34.423000

13.0 | 99 | A GIOVINAZZI | Alfa Romeo Racing | Total Time: NaT | Last Lap: 00:01:34.792000

14.0 | 6 | N LATIFI | Williams | Total Time: NaT | Last Lap: 00:01:32.477000

15.0 | 7 | K RAIKKONEN | Alfa Romeo Racing | Total Time: NaT | Last Lap: 00:01:32.537000

16.0 | 11 | S PEREZ | Red Bull Racing | Total Time: NaT | Last Lap: 00:01:31.284000

17.0 | 9 | N MAZEPIN | Haas F1 Team | Total Time: NaT | Last Lap: 00:01:32.909000

18.0 | 47 | M SCHUMACHER | Haas F1 Team | Total Time: NaT | Last Lap: 00:01:32.987000

19.0 | 5 | S VETTEL | Aston Martin | Total Time: NaT | Last Lap: 00:01:41.335000

20.0 | 33 | M VERSTAPPEN | Red Bull Racing | Total Time: NaT | Last Lap: NaT


Total Laps: 970
Number of Drivers: 20
```

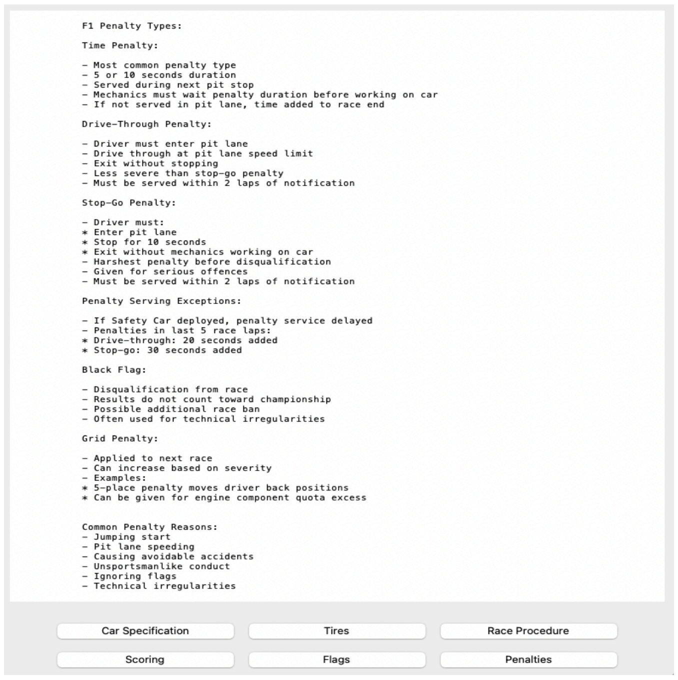
From clear observation, this method did not look great. Which is why the code outputs the track visualization into another figure. Further aesthetic improvements were made, such as rotating the track so it faced the viewer lengthwise, adding a light blue color to the track since the original black looked too bland, including a header displaying the session type and track, and adding grey corner markers with the appropriate corner markers. Attached below is an example of the 2023 Spanish Grand Prix which introduced all of these aesthetic changes:



The final feature to be added was the regulation data, a key component to provide users with quick and direct access to important rules and guidelines. Now for external context, developing the code thus far to extract and display race data and track visualizations for each race season took over 40 hours of work. Given the complexity of these features, the regulation data was intentionally designed to be as simple and efficient as possible. This simplicity was achieved by storing all the regulation data in a dictionary list, which was then integrated into the previously mentioned createwidgets function.

The decision to hardcode the regulation data stemmed from the fact that many of these rules are unlikely to change over time, making this approach both practical and reliable. It also ensures that users have immediate access to this information whenever needed. Initially, the plan was to include 10 regulation statements that users might find helpful. However, this number felt excessive for a compact GUI, potentially cluttering the interface. To address this, a for loop was implemented to dynamically adjust the number of displayed regulation texts based on the available space and design.

After many adjustments, six main topics were selected to summarize the most important aspects of the regulations, as mentioned previously these topics are: car specifications, tire rules, race procedures, scoring systems, flags, and penalties. This choice balanced simplicity and usefulness, ensuring the data is easy to navigate while still being informative. Below is an image showing how the penalty regulation data is presented in the GUI:



```
F1 Penalty Types:

Time Penalty:

- Most common penalty type
- 5 or 10 seconds duration
- Served during next pit stop
- Mechanics must wait penalty duration before working on car
- If not served in pit lane, time added to race end

Drive-Through Penalty:

- Driver must enter pit lane
- Drive through at pit lane speed limit
- Exit without stopping
- Less severe than stop-go penalty
- Must be served within 2 laps of notification

Stop-Go Penalty:

- Driver must:
* Enter pit lane
* Stop for 10 seconds
* Exit without mechanics working on car
- Harshest penalty before disqualification
- Given for serious offences
- Must be served within 2 laps of notification

Penalty Serving Exceptions:

- If Safety Car deployed, penalty service delayed
- Penalties in last 5 race laps:
* Drive-through: 20 seconds added
* Stop-go: 30 seconds added

Black Flag:

- Disqualification from race
- Results do not count toward championship
- Possible additional race ban
- Often used for technical irregularities

Grid Penalty:

- Applied to next race
- Can increase based on severity
- Examples:
* 5-place penalty moves driver back positions
* Can be given for engine component quota excess


Common Penalty Reasons:
- Jumping start
- Pit lane speeding
- Causing avoidable accidents
- Unsportsmanlike conduct
- Ignoring flags
- Technical irregularities
```

After this stage, several minor adjustments were made to enhance the aesthetics of the GUI. Some design ideas worked well, while others didn’t. The button layout was rearranged to make it more intuitive, ensuring users could easily identify the order in which to press the buttons. However, a minor drawback arose due to the tkinter GUI setup using a grid layout, which caused some columns and rows to appear slightly misaligned.

Once all the buttons and dropdown menus were placed in satisfactory locations, significant effort went into adding color to the GUI. Unfortunately, this turned out to be more difficult than expected. Many of the available color options supported by the GUI were too harsh, making the design look overly bright or even childish.

As a result, the focus shifted toward creating a minimalistic look. This decision led to the adoption of a clean, classic appearance with light and dark modes, which balanced simplicity with functionality. After resolving all remaining bugs and refining the design, an effort that took around 50 hours, the final version of the GUI was completed.

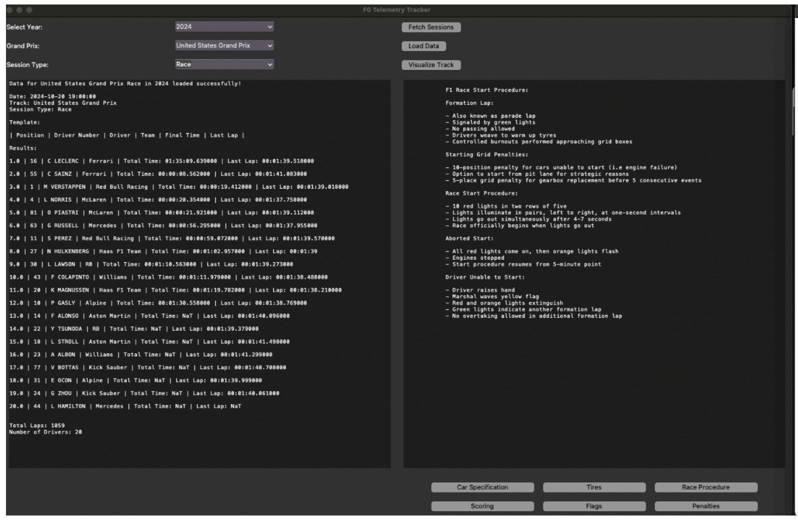
The finalized version brings all the features of the project together. This GUI is the centerpiece of the project, designed to deliver simple yet important information about a Formula One Grand Prix weekend. It is divided into two main sections: race session data on the left and regulation data on the right, each offering a unique set of features to help users better understand the sport.

The race session data section uses the FastF1 API to provide detailed information about different types of race sessions, including free practice, qualifying, race, and even sprint sessions. Users can select a specific season between 2018 and 2024, choose the type of session, and load the data. The results are displayed in the left-hand text widget in an organized grid format. This grid shows key details like the finishing order, driver names, team names, and lap times, making it easy to review and analyze.

The track visualization feature complements the race session data by giving users a static map of the selected circuit. This map includes important elements such as marked corners and the start/finish line, making it a helpful tool for newcomers to visualize the race. The map is displayed in a separate figure, which enhances its clarity and ensures users can focus on the track layout without distractions.

The regulation data section covers six essential topics of Formula One: car specifications, tires, race procedures, scoring, flags, and penalties. This information is displayed in the right-hand text widget, offering a simple and accessible way for users to understand the fundamental rules of the sport. The regulation data does not change as it is based on rules that have been present in the sport for a long time. The regulation data is presented in a minimalistic format, keeping the GUI clean and easy to navigate.

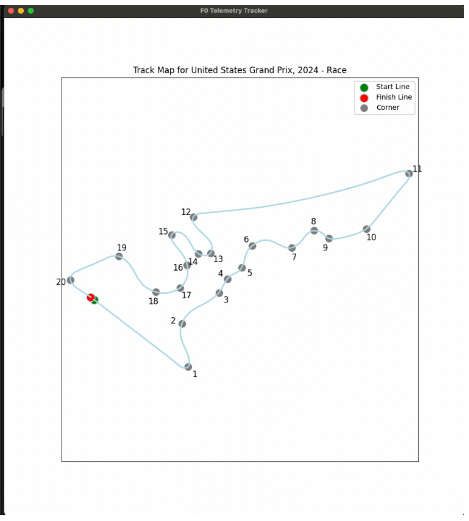
Despite its straightforward design, the telemetry tracker provides a rich set of features for beginners and enthusiasts alike. From fetching detailed race session data to visualizing tracks and presenting essential regulations, the finalized code creates a complete and user-friendly tool that captures the excitement and complexity of Formula One in a clear and accessible way. Attached below is a filled out GUI displaying relevant information for the 2024 United States Grand Prix:



Select Year: 2024
Grand Prix: United States Grand Prix
Session Type: Race

```
Data for United States Grand Prix Race in 2024 loaded successfully!

Date: 2024-10-20 19:00:00
Track: United States Grand Prix
Session Type: Race

Template:

| Position | Driver Number | Driver | Team | Final Time | Last Lap |

Results:

1.0 | 16 | C LECLERC | Ferrari | Total Time: 01:35:09.639000 | Last Lap: 00:01:39.518000

2.0 | 55 | C SAINZ | Ferrari | Total Time: 00:00:08.562000 | Last Lap: 00:01:41.083000

3.0 | 1 | M VERSTAPPEN | Red Bull Racing | Total Time: 00:00:19.412000 | Last Lap: 00:01:39.018000

4.0 | 4 | L NORRIS | McLaren | Total Time: 00:00:20.354000 | Last Lap: 00:01:37.758000

5.0 | 81 | O PIASTRI | McLaren | Total Time: 00:00:21.921000 | Last Lap: 00:01:39.112000

6.0 | 63 | G RUSSELL | Mercedes | Total Time: 00:00:56.295000 | Last Lap: 00:01:37.955000

7.0 | 11 | S PEREZ | Red Bull Racing | Total Time: 00:00:59.072000 | Last Lap: 00:01:39.570000

8.0 | 27 | N HULKENBERG | Haas F1 Team | Total Time: 00:01:02.957000 | Last Lap: 00:01:39

9.0 | 30 | L LAWSON | RB | Total Time: 00:01:10.563000 | Last Lap: 00:01:39.273000

10.0 | 43 | F COLAPINTO | Williams | Total Time: 00:01:11.979000 | Last Lap: 00:01:38.488000

11.0 | 20 | K MAGNUSSEN | Haas F1 Team | Total Time: 00:01:19.782000 | Last Lap: 00:01:38.210000

12.0 | 10 | P GASLY | Alpine | Total Time: 00:01:30.558000 | Last Lap: 00:01:38.769000

13.0 | 14 | F ALONSO | Aston Martin | Total Time: NaT | Last Lap: 00:01:40.096000

14.0 | 22 | Y TSUNODA | RB | Total Time: NaT | Last Lap: 00:01:39.379000

15.0 | 18 | L STROLL | Aston Martin | Total Time: NaT | Last Lap: 00:01:41.498000

16.0 | 23 | A ALBON | Williams | Total Time: NaT | Last Lap: 00:01:41.299000

17.0 | 77 | V BOTTAS | Kick Sauber | Total Time: NaT | Last Lap: 00:01:40.708000

18.0 | 31 | E OCON | Alpine | Total Time: NaT | Last Lap: 00:01:39.999000

19.0 | 24 | G ZHOU | Kick Sauber | Total Time: NaT | Last Lap: 00:01:40.061000

20.0 | 44 | L HAMILTON | Mercedes | Total Time: NaT | Last Lap: NaT


Total Laps: 1059
Number of Drivers: 20
```

```
F1 Race Start Procedure:

Formation Lap:

- Also known as parade lap
- Signaled by green lights
- No passing allowed
- Drivers weave to warm up tyres
- Controlled burnouts performed approaching grid boxes

Starting Grid Penalties:

- 10-position penalty for cars unable to start (i.e engine failure)
- Option to start from pit lane for strategic reasons
- 5-place grid penalty for gearbox replacement before 5 consecutive events

Race Start Procedure:

- 10 red lights in two rows of five
- Lights illuminate in pairs, left to right, at one-second intervals
- Lights go out simultaneously after 4-7 seconds
- Race officially begins when lights go out

Aborted Start:

- All red lights come on, then orange lights flash
- Engines stopped
- Start procedure resumes from 5-minute point

Driver Unable to Start:

- Driver raises hand
- Marshal waves yellow flag
- Red and orange lights extinguish
- Green lights indicate another formation lap
- No overtaking allowed in additional formation lap
```



Developing this telemetry tracker introduced many unique concepts. The project required a deep understanding of how to work with different tools and technologies to achieve the main goals. Learning how to efficiently collect data using the FastF1 API, process it with powerful libraries like Pandas, and present it in meaningful ways through visualization tools like matplotlib were some of the various techniques used. Each step of the process, from gathering raw telemetry data to structuring it for analysis and finally displaying it in an aesthetic GUI taught important lessons about data management and software design.

The main tool for this project was the FastF1 API, which played a crucial role in accessing Formula One telemetry and race data. Through this API, valuable experience was gained in working with external data sources by learning how to send data requests and handle the incoming data effectively. This involved understanding the structure of the data provided, navigating its documentation, and implementing the necessary steps to retrieve specific information, such as lap times, sector performance, and race positions.

However unique challenges needed to be addressed, such as managing differences in sprint race data, which required additional logic for newer seasons (since 2021). Additionally a separate sorting system needed to be implemented for all of the free practice data. Many of the challenges encountered during the project were resolved by exploring GitHub forums and learning from the experiences of others. These forums provided a wealth of knowledge, as numerous users over the years have worked on similar tools and openly shared their code, ideas, and troubleshooting tips. By studying their approaches, valuable insights were gained into best practices for handling complex data structures, and optimizing performance. For example, discussions on common issues with the FastF1 API, such as handling outdated data formats or managing sprint race sessions, offered practical solutions to similar problems faced during this project. The collaborative environment highlighted the importance of open-source communities in fostering innovation and problem-solving. By building on the work of others and adapting their techniques to suit specific requirements, the functionality and reliability of the telemetry system were significantly enhanced. This experience reinforced the value of collaboration and resourcefulness in overcoming technical challenges.

To handle the large amounts of data collected from the FastF1 API, the Pandas library was a vital tool. Pandas is widely used for data manipulation because it provides an easy way to organize, clean, and analyze datasets in tabular form. This project made extensive use of Pandas to sort through the raw telemetry and race data, allowing for clear and structured analysis. By leveraging its DataFrame structures, accessing specific portions of the data, applying transformations, and extracting meaningful insights without needing to write complex, low-level code.

One key feature of Pandas used in this project was its ability to group and filter data. For example, to determine the fastest laps, the data was grouped by drivers and filtered to select the lap with the smallest lap time for each driver. This helped compare performances across drivers during a session, providing valuable insights into how each one performed in terms of speed and consistency. These operations, which might have been time-consuming to implement manually, were simplified with Pandas' functions. Additionally, Pandas made it easier to handle incomplete or inconsistent data. For instance, telemetry data often had missing entries or unexpected formats, such as laps without proper timing information.

For creating graphs and charts, the matplotlib library was essential. This tool made it possible to produce clear and detailed visualizations that presented complex data in an understandable way. For example, lap time comparisons were plotted to show how different drivers performed across various laps, making patterns and differences easy to identify. Additionally, matplotlib was used to create positional maps that illustrated driver movements on the track. These visualizations provided valuable insights into race dynamics and performance trends, transforming raw telemetry data into an accessible format for analysis.

To improve user interaction, tkinter was used to develop a graphical user interface (GUI). This interface allowed users to select specific race sessions, view related data, and interact with the project without needing to write or modify any code. The simplicity of tkinter made it possible to design an intuitive and functional interface that connected seamlessly with the backend data-handling processes. The GUI streamlined the workflow by providing a straightforward way to access and visualize information.

Integrating matplotlib and tkinter demonstrated how design and functionality could work together to enhance the overall project. The GUI acted as a bridge, allowing the user to interact with the backend code while displaying data visually in real time. This combination of tools highlighted the importance of creating systems that are both technically robust and user-friendly. It also emphasized the value of visual representations in making data more engaging and easier to interpret.

Adding the regulation data to the project involved using concepts learned in class, particularly those related to data organization and structure. A dictionary was used to store and sort all the regulation topics. This approach made it easy to categorize and access the information, allowing the data to be efficiently organized. The regulation topics were structured in a way that matched the categories in the Formula One regulation rulebook. This rulebook served as the primary resource for gathering the necessary information to populate the dictionary. By using this method, it was possible to create a clear and well-organized dataset that could be easily accessed and updated.

In addition to organizing data, commenting code was another important skill learned during the project. Proper comments helped clarify the purpose of each section of the code, making it easier to understand how different parts of the project worked together. For example, when using dictionaries to store regulation data, comments explained what each key and value represented, helping anyone reading the code follow the logic more easily. This practice also made it simpler to troubleshoot issues, as the comments provided a clear reference to what each part of the code was intended to do.

Commenting code not only improved the organization of the project but also made it easier to maintain. As the project grew and new features were added, having well-documented code helped keep everything organized and ensured that the purpose of each section was clear. This skill was particularly useful when revisiting the project after some time, as it allowed for a quick understanding of how everything was structured and how the different parts of the project interacted with each other. Overall, commenting code and using structured data storage techniques were key aspects of the project that helped improve its clarity and functionality.

Throughout the course of the project, several features were considered but ultimately not implemented due to limitations such as time constraints, software access, or limits in knowledge. Time constraints were a significant factor, as there were features that could have added value but were not essential for the core functionality of the project. Given the nature of the project's timeline, these features were prioritized as "nice-to-haves" rather than necessary components. For example, advanced visualizations or additional analysis tools could have enhanced the user experience, but they were set aside to ensure that the primary aspects of the project, such as data collection, processing, and visualization, were completed within the available time frame. The focus remained on delivering a functional and user-friendly system, and the decision was made to defer these additional features to future updates.

One example of a feature considered but ultimately excluded was the addition of driver pictures in the GUI. The idea behind this feature was to help users become familiar with the drivers and their respective teams. While integrating the .jpeg files for the pictures was technically straightforward, the images did not display well within the GUI. Their appearance was inconsistent and did not align with the clean and functional design of the interface. Moreover, the pictures were found to add little value to the context of the race session data, as the primary focus of the project was on performance metrics and telemetry analysis rather than visual recognition. Additionally, for users watching the Grand Prix alongside the project, drivers are already prominently shown throughout the broadcast, making this feature redundant. As a result, the decision was made to exclude driver pictures, ensuring that the interface remained focused on presenting meaningful and relevant data.

Another feature considered but not implemented was the use of playful animations created with the Manim Python library. This library, widely recognized for its use by the popular YouTube channel 3Blue1Brown, offers powerful tools for generating engaging mathematical animations. The idea was to incorporate these animations into the project, such as displaying dynamic transitions when users switched between race data or regulation topics. The goal was to make the user experience more visually appealing and interactive. However, preliminary testing of the animations revealed several drawbacks. While the animations were visually impressive, integrating them into the GUI significantly slowed down the application's performance. The added processing time introduced delays that disrupted the smooth operation of the GUI, outweighing any aesthetic benefits. Additionally, the animations did not provide any functional advantage or meaningful enhancement to the user's understanding of the data. Given these limitations, the decision was made to exclude this feature, prioritizing a faster and more efficient user experience.

Lastly, the inclusion of national anthems for the winning driver and the winning constructor of each selected race was considered. This idea was inspired by the tradition in Formula One of playing the national anthem of the winning driver's country and the anthem of the winning team's country during the podium ceremony. The concept was to make this feature exclusive to the specific race sessions chosen by the user, adding an immersive and celebratory element to the experience. However, research revealed several challenges with implementing this feature. Adding the anthems would have significantly increased the amount of data users needed to download, potentially making the application less efficient and user-friendly. Additionally, proper integration of audio playback would have required considerable development time to ensure seamless functionality. At that point, over 45 hours had already been invested in the project, and there were still unresolved bugs that required attention. Given the time constraints and the need to focus on fixing critical issues, the decision was made to forgo this feature to maintain project efficiency and prioritize delivering a stable final product.

Another limitation encountered during the project was related to software access, particularly in implementing a race simulation feature. As previously mentioned, the FastF1 API does not continuously record each driver's position throughout a race or session. This lack of constant positional data made it impossible to create an accurate, real-time simulation of driver movements. While alternative APIs do exist that provide continuous positional tracking, many of them require paid subscriptions, which were not feasible within the scope of this project. Additionally, even if all drivers' positional data could be obtained, storing and processing such a large volume of data would have created challenges for the streamlined design of the GUI. The finalized GUI was specifically designed to handle only essential session data, ensuring that it remained efficient and easy to use. Incorporating a race simulation would have significantly increased the system's complexity and storage requirements, making it impractical under the given constraints.

Additionally there was the challenge of accurately obtaining the final times for all drivers. In the data provided by the FastF1 API, drivers who are more than one lap behind the leader at the end of the race are marked with a "+1 lap" designation instead of a precise finishing time. This makes it difficult to calculate their exact race duration or directly compare their performance to that of the leading driver. While there are paid APIs available that attempt to estimate these final times by extrapolating data, incorporating such tools was not feasible within the scope of the project. The reliance on freely available resources meant that this limitation had to be accepted, focusing instead on other aspects of race analysis that could be accurately captured with the data at hand.

The last time restrictive feature considered for the project was the inclusion of speed zone markers for each driver, highlighting their fastest and slowest sections throughout each session of the racing weekend. This feature aimed to provide a detailed view of driver performance by mapping speed variations across the track. However, it encountered challenges similar to those faced with the race simulation feature. While it was possible to track an individual driver's lap performance through the circuit, attempting to do so for all 20 drivers placed significant strain on the system. Processing and visualizing such a large volume of data for every driver proved too resource intensive, impacting the efficiency of the application. Additionally, this type of information is typically displayed during the live Grand Prix, making its inclusion in the project less critical. Given these constraints, the feature was ultimately excluded to maintain the streamlined functionality and performance of the system.

Several features were not implemented due to a lack of knowledge about their proper implementation, one of which was the display of driver radio messages. These messages, which capture communication between drivers and their teams, are typically shown during live races and provide unique insights into strategy and decision-making. While including this feature in the GUI was considered, it presented multiple challenges. Displaying the radio messages cluttered the interface, making it appear overly crowded. Furthermore, many of the messages in the dataset were left blank, reducing their overall usefulness. An attempt was also made to integrate audio playback for these messages, but this was hindered by time constraints and technical difficulties. The main barrier to implementing this feature, however, was the limited external knowledge available on how to work with radio messages in the API. Despite searching through GitHub forums and other resources, there was little guidance or documentation on extracting and displaying these messages effectively. This lack of reliable information made it difficult to proceed with developing a robust solution. As a result, the decision was made to exclude this feature, prioritizing the completion of other functionalities that were more achievable within the project's scope.

Another feature considered but not implemented was the inclusion of live penalty calls during race sessions. This addition faced similar challenges to other features, such as increased data complexity and the potential to overwhelm the GUI. Despite these difficulties, incorporating a system to familiarize users with penalties was deemed important, as penalties are a significant part of understanding Formula One rules and race outcomes. To address this need, the regulation data was utilized as an alternative. A dictionary was created to store and organize penalty-related information, covering most of the common penalties that occur during race sessions. This solution allowed the project to provide users with relevant details about penalties in a structured and accessible way, even though live penalty updates were not feasible. By focusing on this approach, the project successfully included a measure to enhance user understanding of race regulations without compromising the simplicity or functionality of the interface.

Lastly, a feature explored was the inclusion of strategy calls for each driver during race sessions. Strategies, particularly those involving tire choices and pit stops, play a critical role in Formula One races and offer valuable insight into team decisions. However, several challenges arose during the attempt to implement this feature. Software limitations were a significant factor, as the FastF1 API does not directly provide detailed strategy data for each driver. Additionally, race strategies are highly complex and vary significantly between teams, making it difficult to present this information in a clear and meaningful way within the GUI. To address this challenge, the feature was adapted to focus on the regulation data instead. Users were provided access to the rules governing tire usage, which is often the central element of race strategies. By including this information in the regulation data, the project allowed users to understand the general guidelines that influence strategic decisions without requiring the complexity of live updates. This approach maintained the project's focus on clarity and functionality while still offering valuable insights into an essential aspect of Formula One racing.

In conclusion, this telemetry tracker project brought together a wide array of tools, techniques, and concepts to create a functional and educational application centered around Formula One racing. By leveraging the FastF1 API, Pandas, Matplotlib, and tkinter, the project effectively collected, processed, and displayed race data while providing users with an intuitive graphical interface. Challenges such as software limitations, time constraints, and gaps in knowledge shaped the project's scope. However, alternative solutions, such as incorporating regulation data and streamlining session focused information, ensured the application remained efficient and user-friendly.

Through the process, critical skills such as integrating multiple technologies, organizing data with dictionaries, and commenting code for clarity were applied and further developed. While some features were left unimplemented, the project successfully achieved its primary goals of providing a beginner friendly engaging platform for race session analysis and an introduction to Formula One regulations.