UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE GRADUAÇÃO
DEPARTAMENTO CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA E TECNOLOGIA

Lucas Mateus Alves Cruz

Utilizando APIs em Python para aquisição e análise de dados

MOSSORÓ
2025

Lucas Mateus Alves Cruz

Utilizando APIs em Python para aquisição e análise de dados

Monografia apresentada à Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel em Ciência e Tecnologia.

Orientador: Prof. Dr. Victor Wagner Freire de Azevedo

MOSSORÓ

2025

Ficha catalográfica elaborada pelo Sistema de Bibliotecas
da Universidade Federal Rural do Semi-Árido, com os dados fornecidos pelo(a) autor(a)

CC957 u
Cruz, Lucas Mateus Alves.
Utilizando APIs em Python para aquisição e análise de dados / Lucas Mateus Alves Cruz. - 2025.
29 f. : il.

Orientador: Victor Wagner Freire de Azevedo.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de Ciência e Tecnologia, 2025.

1. APIs. 2. análise de dados. 3. FastF1. 4. python. 5. fórmula 1. I. de Azevedo, Victor Wagner Freire , orient. II. Título.

O serviço de Geração Automática de Ficha Catalográfica para Trabalhos de Conclusão de Curso (TCC´s) foi desenvolvido pelo Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (USP) e gentilmente cedido para o Sistema de Bibliotecas da Universidade Federal Rural do Semi-Árido (SISBI-UFERSA), sendo customizado pela Superintendência de Tecnologia da Informação e Comunicação (SUTIC) sob orientação dos bibliotecários da instituição para ser adaptado às necessidades dos alunos dos Cursos de Graduação e Programas de Pós-Graduação da Universidade.

AGRADECIMENTOS

Agradeço ao Prof. Dr. Victor Wagner Freire de Azevedo, meu orientador, pela orientação acadêmica, pelo conhecimento compartilhado e pela dedicação ao longo de todo o desenvolvimento deste trabalho.

Agradeço ao Prof. Valdemar Siqueira Filho, docente de Iniciação à Escrita Acadêmica, pela orientação e suporte na elaboração da parte escrita deste trabalho. Suas contribuições foram fundamentais para o aprimoramento da clareza e coesão textual deste projeto.

Agradeço à Banca Examinadora, composta pelos professores Jakson Gomes de Oliveira Júnior e Bruno Rodrigo Simão, pela disponibilidade em avaliar este trabalho e pelas valiosas contribuições que certamente enriqueceram o conteúdo apresentado. Seus conhecimentos e críticas construtivas são de grande importância para o aprimoramento desta pesquisa.

Agradeço à Cecilia Maria Fernandes da Silva, minha companheira, pelo apoio e incentivo durante a realização deste trabalho. Sua contribuição foi relevante para a conclusão desta pesquisa.

Por fim, expresso minha gratidão a todos que, direta ou indiretamente, contribuíram para a realização deste projeto.

RESUMO

As Application Programming Interfaces (API) são ferramentas versáteis para o compartilhamento de dados nos dias atuais. Elas permitem a comunicação fácil entre diferentes aplicativos, o que torna o processo de análise eficiente. Este trabalho enfatizou a aplicação de APIs para extração e análise de dados, utilizando a biblioteca FastF1 como exemplo prático. A pesquisa abordou como ferramentas de programação, especialmente em Python, podem ser aplicadas para coletar, processar e visualizar informações provenientes de fontes de dados esportivos. A análise concentrou-se em dados da Fórmula 1, como tempos de volta e velocidades em pontos da pista, destacando a relevância dessas informações para o entendimento do desempenho esportivo. Na metodologia foram utilizadas bibliotecas como Pandas, Matplotlib, Seaborn, dentre outras para manipulação e visualização dos dados obtidos por meio da FastF1. Os resultados esperados incluiram gráficos que facilitaram a interpretação das informações coletadas, demonstrando o potencial das APIs para a análise de dados complexos em comparação com métodos tradicionais. Este estudo contribuiu para uma discussão sobre o uso de APIs como ferramentas de suporte à análise de dados esportivos, facilitando a aplicação prática de técnicas e o uso de estatísticas em um contexto de interesse acadêmico e profissional.

Palavras-chave: APIs, análise de dados, FastF1, python, fórmula 1, processamento de dados.

ABSTRACT

The Application Programming Interfaces (API) are versatile tools for sharing data today. They allow easy communication between different applications, which makes the analysis process efficient. This work emphasized the application of APIs for data removal and analysis, using the FastF1 library as a practical example. The research addressed how programming tools, especially in Python, can be applied to collect, analyze and visualize information from sports data sources. The analysis focused on Formula 1 data, such as lap times and speeds at points on the track, highlighting the relevance of this information for understanding sporting performance. In the methodology, libraries such as Pandas, Matplotlib, Seaborn, among others, were used to manipulate and visualize the data obtained through FastF1. The expected results included graphs that facilitated the interpretation of the information collected, demonstrating the potential of APIs for analyzing complex data compared to traditional methods. This study contributed to a discussion about the use of APIs as tools to support the analysis of sports data, facilitating the application of techniques and the use of statistics in a context of academic and professional interest.

Keywords: APIs, data analysis, FastF1, python, formula 1, data processing.

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

Na era contemporânea, o processamento e a utilização de dados são aspectos cada vez mais importantes, especialmente no esporte, onde seu impacto no desempenho de atletas e equipes é determinante. A Fórmula 1, por exemplo, possui uma grande complexidade técnica e um altíssimo nível de competitividade entre as equipes e, até mesmo, entre pilotos de uma mesma equipe, sendo um exemplo claro de como os dados são de grande importância, tanto na definição de estratégias de corrida quanto no desenvolvimento de veículos (Msakamali, 2024).

A coleta e a análise dessas informações exigem ferramentas especializadas, capazes de acessar e interpretar grandes volumes de dados com eficiência. Nesse contexto, as APIs (Applications Programming Interfaces) permitem a automatização da coleta de dados e sua integração a sistemas personalizados de análise, ampliando as possibilidades de uso e, além disso, aumentando a velocidade de todo o processo (MUGGE; ZANDIEH, 2023). Este trabalho abordou a aplicação da API FastF1, que fornece acesso a dados da Fórmula 1 com riqueza de detalhes, como tempos de volta, uso de pneus, condições climáticas e estratégias de corrida.

A escolha deste tema justificou-se pela crescente demanda por ferramentas acessíveis e eficientes para a análise de dados dentro do contexto esportivos, juntamente ao interesse acadêmico e profissional no uso de linguagens de programação. Estudos recentes têm explorado novos modelos, o que inclui o desenvolvimento de sistemas capazes de ler e tratar dados provenientes de esporte a motor (BANERJEE et al., 2022; DE CARLO et al., 2024). Além disso, essas tecnologias também vêm sendo aplicadas em projetos educacionais, em que dados podem ser coletados e analisados de maneira rápida através de programas computacionais em universidades (DE LIMA et al., 2022).

Já existem trabalhos sobre o uso de APIs na análise de dados esportivos, o que indica um vasto campo para o aprofundamento prático de bibliotecas especializadas, como a FastF1, que oferece uma interface direta com os dados oficiais da Formula 1. Um exemplo relevante foi o estudo de Msakamali (2024), que utilizou o FastF1 em conjunto com outras bibliotecas Python, como Matplotlib e Pandas, para analisar dados de telemetria e desempenho de pneus em corridas da Fórmula 1. Seu trabalho demonstrou como a integração de APIs e ferramentas

de análise de dados pode proporcionar insights valiosos sobre estratégias de corrida e desempenho dos pilotos, reforçando a necessidade de aprofundar essas pesquisas.

O principal objetivo deste trabalho é promover uma discussão sobre o uso de APIs e demonstrar como a API FastF1 pode ser aplicada na extração, processamento e visualização de dados esportivos, utilizando ferramentas de programação em Python. A hipótese foi que a aplicação de APIs, juntamente com bibliotecas de manipulação e visualização de dados, pode simplificar, enriquecer e agilizar as análises, proporcionando insights mais detalhados, precisos e rápidos, que são de suma importância no esporte.

A metodologia proposta no trabalho abrange o uso de bibliotecas como Pandas, Numpy, Matplotlib, Seaborn, Tkinter, e outras para manipulação e visualização dos dados extraídos da API FastF1. Com este estudo, demonstrou-se a eficiência da API FastF1, bem como foi incentivado o uso de tecnologias mais acessíveis na análise de dados esportivos.

2. FUNDAMENTAÇÃO TEÓRICA

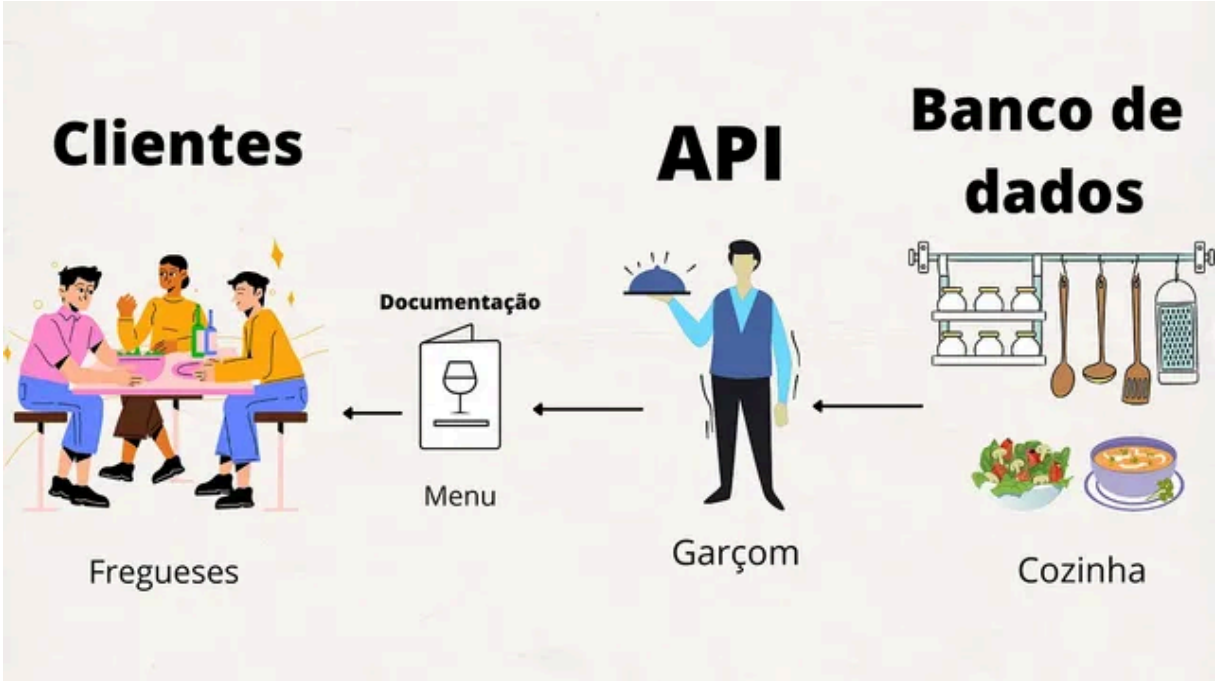
2.1. APIs e seu funcionamento

As APIs são ferramentas poderosas utilizadas nos mais diversos programas, desempenhando um papel fundamental na era da informação. Nesse contexto, seus conceitos foram detalhados.

De acordo com Reddy e Rani (2022), uma API (*Applications Programming Interfaces* ou Interface de Programação de Aplicações) é descrita como um mecanismo e linguagem comum que facilita a interação entre diferentes aplicações, seguindo um conjunto de regras e padrões documentados que especificam como essas interfaces devem ser construídas e utilizadas.

Em outras palavras, API é uma via que permite comunicação entre aplicativos, fazendo com que sistemas conversem entre si. Fazendo uma analogia para entender melhor do que se trata, uma API seria um menu de um restaurante, onde nela estaria uma lista com todas as opções disponíveis para o cliente (ou um desenvolvedor) . Desta forma, utilizando o menu, o cliente faria um pedido ao restaurante, que seria um servidor (ver Figura 1) (REDDY; RANI, 2022).

Figura 1: Ilustração de comparação entre restaurante e uma API.



Fonte: Medium.com[1].

Segundo Reddy e Rani (2022), uma API possui um funcionamento que pode ser dividido em 4 etapas distintas, sendo elas:

Etapa 1: O usuário utiliza-se de algum aplicativo cliente e faz uma solicitação, como por exemplo um navegador;

Etapa 2: A API entra em contato com o servidor web e recupera informações do banco de dados;

Etapa 3: O servidor reúne as informações e disponibiliza para a API ; e

Etapa 4: A API pega os dados e passa para o aplicativo cliente onde o usuário final fez a solicitação.

2.2. Contexto esportivo da Fórmula 1

Dentro da Fórmula 1, a análise de dados é algo de suma importância para o desempenho das equipes participantes, envolvendo diversos aspectos como o desenvolvimento dos carros e estratégias de corrida, treinamento e seleção de pilotos.

[1]Disponível em: https://medium.com/@stefanerefrande/o-que-uma-api-e-um-restaurante-t%C3%AAm-em-comum-f0ede118b33. Acesso em: 15 fev. 2025.

Jenkins e Floyd (2001) destacam que a evolução tecnológica na Fórmula 1 é impulsionada pela transparência do conhecimento técnico e pela coevolução de tecnologias. Quando o conhecimento é facilmente compartilhado, as tecnologias evoluem entre as equipes, levando a designs dominantes no nível do sistema. Já quando o conhecimento é menos transparente, a coevolução ocorre dentro das firmas, resultando em designs dominantes no nível da empresa.

Jenkins e Floyd (2001) observam que o ambiente altamente competitivo da Fórmula 1 acelera a inovação, com ciclos de desenvolvimento curtos e um foco intenso no desempenho. Isso reflete a necessidade de adaptação rápida diante das mudanças regulatórias e avanços dos concorrentes.

Um elemento de suma importância para este esporte é a telemetria. A mesma foi usada pela primeira vez na década de 80, quando a Fórmula 1 conseguiu utilizar dados de origem digital (UOL, Julianne Cerasoli), o que representou um grande avanço quando comparado ao método analógico usado anteriormente.

No início de sua utilização, a telemetria era muito rudimentar e servia uma pequena quantidade de informações, tendo um público muito restrito o qual era possível o acesso a tais dados (UOL, Julianne Cerasoli). A democratização de alguns grupos de dados foi se tornando realidade com o passar dos anos, como por exemplo a velocidade em tempo real, a força centrífuga (chamada também de “força G”) sofrida pelo piloto, tempos das voltas e outros (MAPFRE, 2020), porém com alguns outros grupos de dados continuando restritos às equipes, como a temperatura dos pneus, que poderia indicar desgaste e possível parada nos boxes, determinando fortemente o posicionamento final de um piloto em uma corrida.

No contexto esportivo atual, a Fórmula 1 está diretamente relacionada com a aquisição e análise de dados e (PEETERS; WESSELBAUM, 2023), sem tal relacionamento, o esporte talvez não tivesse tanta repercussão, telespectadores e investimento.

2.3. Sobre a API Fast F1

A API FastF1 é uma biblioteca Python de acesso livre, que fornece dados das competições de Fórmula 1, tais como datas, locais, posições de pilotos, tempos de volta, paradas nos boxes, velocidade, marchas, dentre outros, sendo uma ferramenta rápida e eficiente para tal propósito. Com a utilização desta API, é possível obter informações sobre as

colocações de corridas, seus cronogramas, a telemetria de cada veículo e muito mais (Fastf1, s.d).

No caso da API Fast F1, a extração de dados consiste na obtenção de informações sobre corridas, pilotos, tempos de volta, telemetria e outras métricas básicas da Fórmula 1. Após a solicitação desses dados, eles são usados para análises, como calcular a volta mais rápida ou gerar gráficos dentro da aplicação desenvolvida que permitem a visualização de características do carro em uma corrida específica.

3. DESENVOLVIMENTO DO CÓDIGO

O processo de desenvolvimento do código teve diversas etapas com diferentes ferramentas usadas, além da visualização de exemplos funcionais que utilizam a mesma API presente neste trabalho.

3.1. Ambiente de Programação e Ferramentas Utilizadas

Utilizou-se, durante a o desenvolvimento do código, a linguagem de programação Python, que é uma linguagem poderosa e versátil, possui uma sintaxe clara uma grande comunidade,o que oferece uma ampla variedade de bibliotecas e funcionalidades. À princípio, foi proposto o desenvolvimento de um código que fizesse a leitura de arquivos no formato CSV (*Comma Separated Values* ou Valores separados por vírgula) contendo dados genéricos como queima de calorias, velocidade média e outros. Ao ler os dados do arquivo, o código foi programado para produzir um gráfico onde o usuário pudesse visualizar ver o progresso no treino ou esporte de forma rápida e objetiva.

Durante o desenvolvimento, foram utilizadas diversas bibliotecas de Python, incluindo:

- Pandas: utilizada para a manipulação e estruturação de dados extraídos dos arquivos CSV (e, posteriormente, da API FastF1).
- Matplotlib e Seaborn: utilizadas para gerar gráficos e visualizar os dados processados. Obs.: A Seaborn não foi usada no segundo código pois o Matplotlib já atende às necessidades.
- Tkinter: Para a construção de uma interface gráfica que permitisse uma interação facilitada do usuário com os dados.

- FastF1: Utilizada posteriormente, para a obtenção e tratamento de dados da Fórmula 1, permitindo acesso a diversos dados.
- NumPy: Utilizado para cálculos numéricos e manipulação de arrays, especialmente na geração de mapas de pista com gradientes de velocidade.
- Threading: Utilizado para executar operações demoradas, como o carregamento de dados da API, em segundo plano, garantindo que a interface gráfica permanecesse responsiva.
- Datetime: Utilizado para manipulação de datas, especialmente para determinar o ano atual e limitar a seleção de temporadas na interface gráfica.
- OS: Para manipulação de arquivos e diretórios.

3.2. Primeiras versões

Os códigos iniciais foram desenvolvidos na plataforma Google Colab, utilizando sua infraestrutura em nuvem, o que tornou simples a execução de scripts sem precisar instalar pacotes localmente. Em seguida, transferiu-se para a IDE (*Integrated Development Environmen*t, ou Ambiente de Desenvolvimento Integrado), do Python no sistema operacional Windows, permitindo mais controle sobre os arquivos e a execução dos códigos. A ideia inicial foi ter um primeiro contato com a ciência de dados e como realizar as operações dentro do ambiente do Python.

O código original utilizou a biblioteca *Pandas* para importar os dados e *Matplotlib/Seaborn* para criar gráficos explicativos. Como exemplo, utilizou-se dados genéricos sobre tempo de corrida, distância percorrida, velocidade média e calorias queimadas por um corredor arbitrário. Os dados estão representados na Fig. 2.

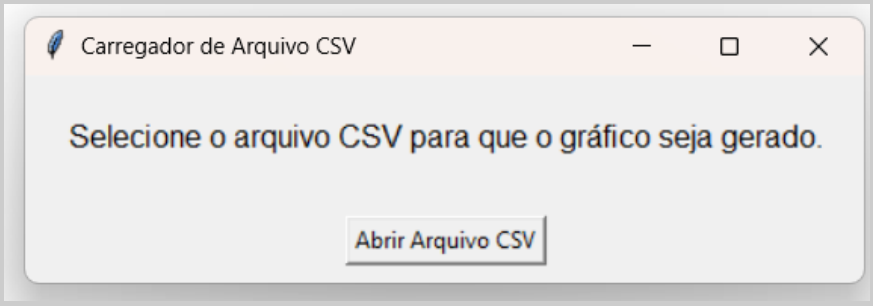
Figura 2: Arquivo CSV contendo dados para a execução do programa.

	A	B	C	D	E	F
1	Data	Hora	Duração	Distância	Velocidade Média	Calorias Queimadas
2	2023-03-08	07:00:00	60 minutos	20 km	20 km/h	500 kcal
3	2023-03-10	08:00:00	90 minutos	30 km	20 km/h	750 kcal
4	2023-03-12	10:00:00	120 minutos	40 km	20 km/h	1000 kcal
5	2023-03-14	12:00:00	150 minutos	50 km	20 km/h	1250 kcal
6	2023-03-16	14:00:00	180 minutos	60 km	20 km/h	1500 kcal
7	2023-03-18	16:00:00	210 minutos	70 km	20 km/h	1750 kcal
8	2023-03-20	18:00:00	240 minutos	80 km	20 km/h	2000 kcal

Fonte: Autoria própria.

A interface desenvolvida com o Tkinter permitiu a abertura do explorador de arquivos do Windows e a seleção do arquivo CSV, como mostra a Fig. 3.

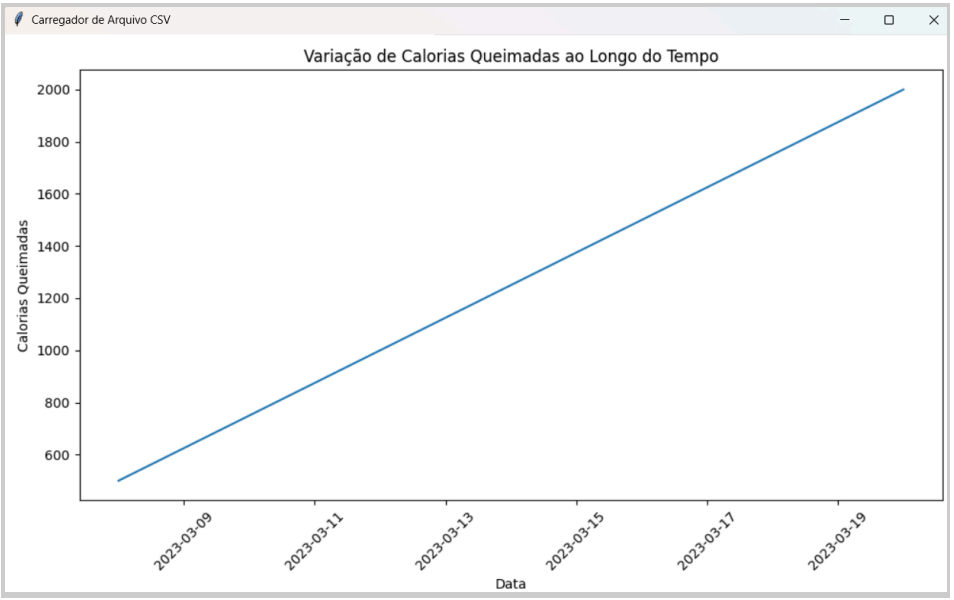
Figura 3: Gerador de gráfico a partir de CSV.



Fonte: Autoria própria.

Após a seleção do arquivo CSV com os dados organizados, o código gerou um gráfico de linha que mostra a variação de calorias queimadas ao longo do tempo, representado na Fig. 4.

Figura 4: Gráfico gerado pelo código inicial.



Fonte: Autoria própria.

Esse código permitiu uma primeira visualização dos dados, representando a quantidade de calorias queimadas com o passar do tempo, auxiliando na identificação de padrões e correlações entre as variáveis analisadas. Observou-se que, guiando-se por esta versão inicial, o código já seria útil para algumas análises esportivas que envolvessem queima de calorias em distância percorrida como caminhadas, ciclismo e outros.

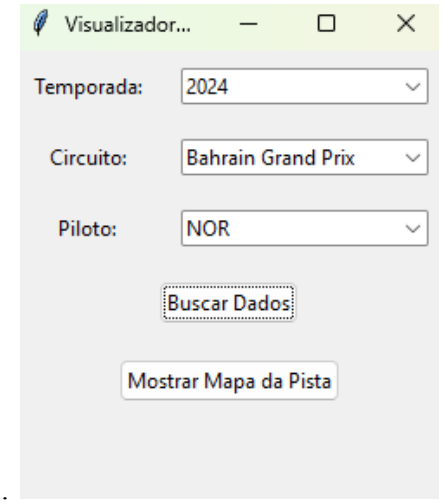
3.3. Desenvolvimento da versão final do programa e sua utilização

Depois da implementação inicial com arquivos CSV, o projeto avançou para a extração de dados diretamente da API FastF1. Esta API oferece acesso a dados completos sobre as corridas da Fórmula 1, eliminando a necessidade de manipular arquivos estáticos manualmente, como realizado no primeiro exemplo. Abaixo está representada, de forma simplificada, a estrutura do código:

1. Entrada: O usuário seleciona a temporada, circuito e piloto na interface Tkinter.
2. Busca de Dados: O código usa a FastF1 para carregar os dados da sessão da F1.
3. Processamento: Filtra as voltas do piloto; obtém a volta mais rápida; se não houver dados, exibe um aviso.
4. Saída: Exibe informações do piloto; plota os tempos de volta; mostra o mapa da pista com dados de telemetria.

A Fig. 5 mostra a nova interface onde foi possível escolher a temporada, o circuito e o piloto, obteve-se, assim, buscar os dados como a volta mais rápida, os tempos em cada volta mostrados em gráfico e uma imagem do circuito com indicadores de velocidade.

Figura 5: Interface gráfica do programa



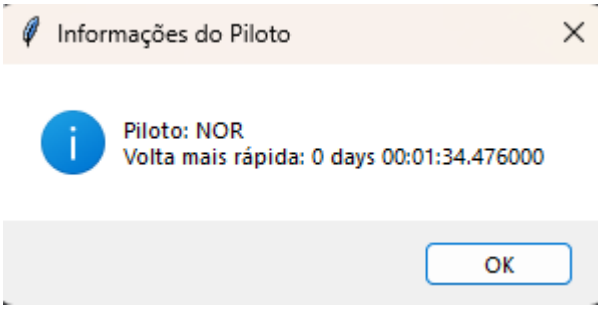
Fonte: Autoria própria.

4. ESTUDO DE CASO: McLAREN NO BAHRAIN

Para demonstração do código, foi analisado o desempenho da equipe McLaren Fórmula 1 no Grande Prêmio do Bahrain, realizado em 2024. Com os dados presentes na Fig. 5, o código permitiu a análise do tempo de volta e especificação da volta mais rápida do piloto

naquela sessão. A Fig. 6 mostrou a volta mais rápida do piloto Lando Norris naquela corrida e a Fig. 7 a evolução de seus tempos de volta.

Figura 6: Janela com a volta mais rápida do piloto de acordo com os itens selecionados (Lando Norris).



Fonte: Autoria própria.

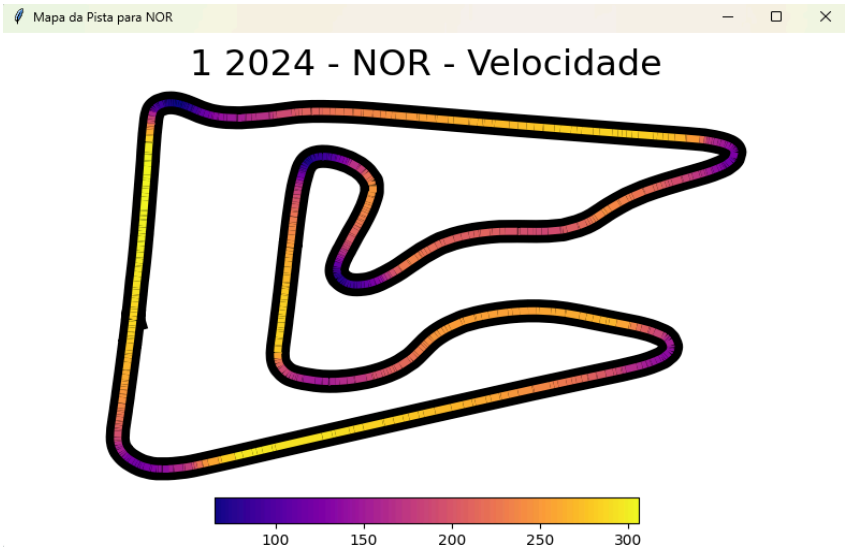
Figura 7: Gráfico de linhas e pontos gerados pelos tempos durante as voltas (Lando Norris).

Fonte: Autoria própria.

Observou-se que, pelos resultados obtidos, a volta mais rápida de Norris teve o tempo de aproximadamente 94,48 segundos (1 minuto, 34 segundos e 476 milissegundos), registrada na volta 35, logo após a sua parada nos boxes para troca de pneu. Os resultados da

Fig. 7 permitiram, também, observar que o piloto manteve um ritmo bem constante durante a corrida nos respectivos espaços em que ficou com o mesmo pneu.

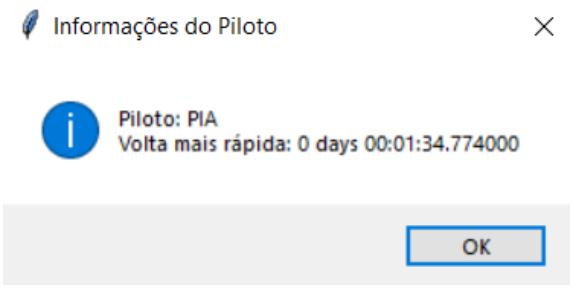
Figura 8: Mapa do circuito com indicadores de velocidade para o circuito no Bahrain (Lando Norris).



Fonte: Autoria própria.

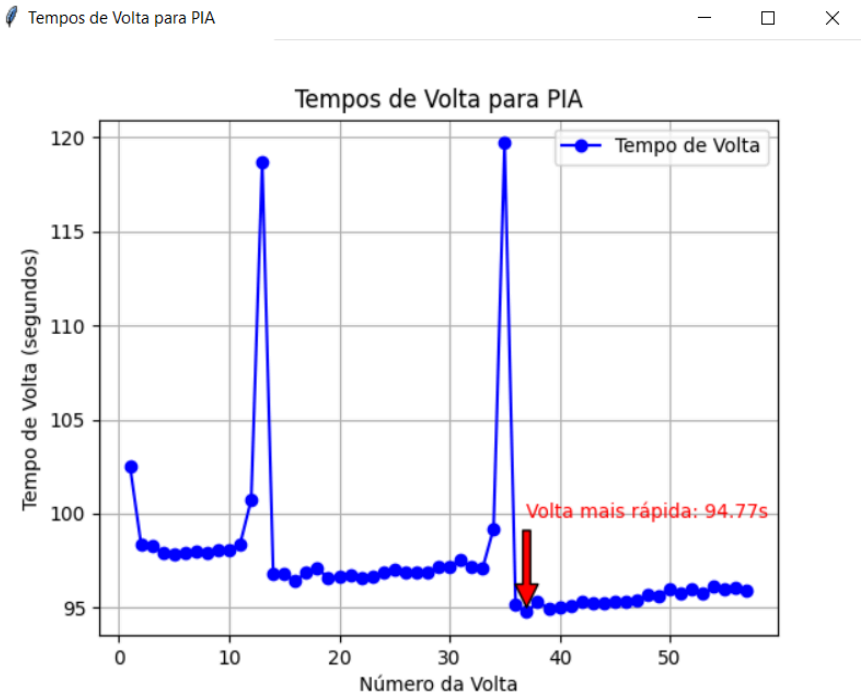
Além da análise de tempos de volta, o código tinha a funcionalidade de "Mostrar mapa da pista" que, como dito anteriormente, mostrou um mapa do circuito com cores que indicaram as velocidades médias do piloto selecionado durante a execução da volta de classificação (Figura 8). Essa funcionalidade do código foi baseada em um exemplo desenvolvido por JSEHV, no Github, e incorporado ao código para complementar informações (JSEHV, 2025). Tal resultado gráfico permitiu a comparação de performance entre diferentes carros em diferentes setores do circuito, como, por exemplo, desempenho em curvas de alta ou baixa velocidade.

Figura 9: Janela com a volta mais rápida do piloto de acordo com os itens selecionados (Oscar Piastri).



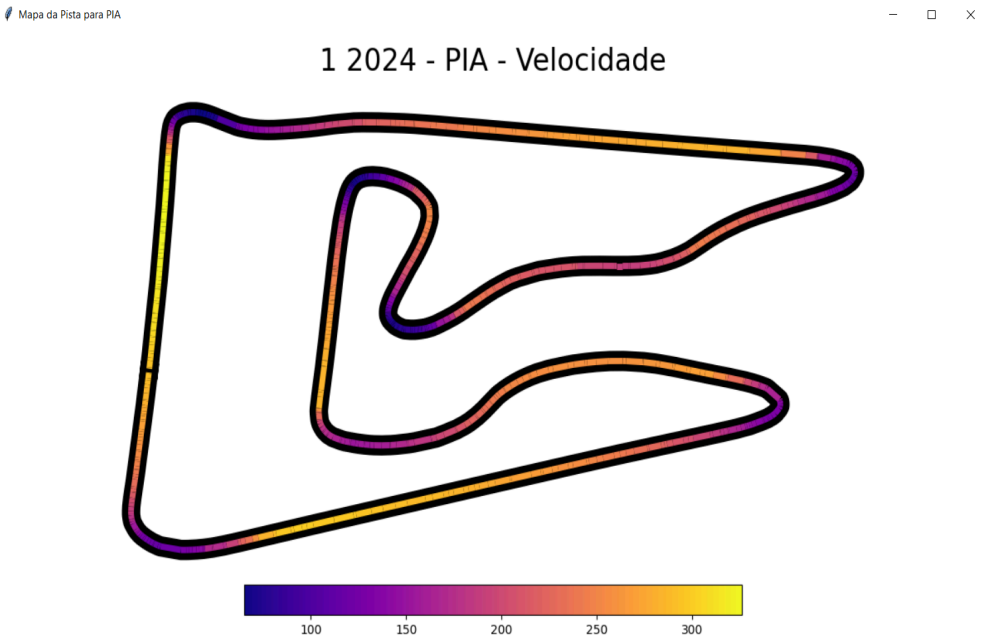
Fonte: Autoria própria.

Figura 10: Gráfico de linhas e pontos gerados pelos tempos durante as voltas (Oscar Piastri).



Fonte: Autoria própria.

Figura 11: Mapa do circuito com indicadores de velocidade para o circuito no Bahrain (Oscar Piastri).



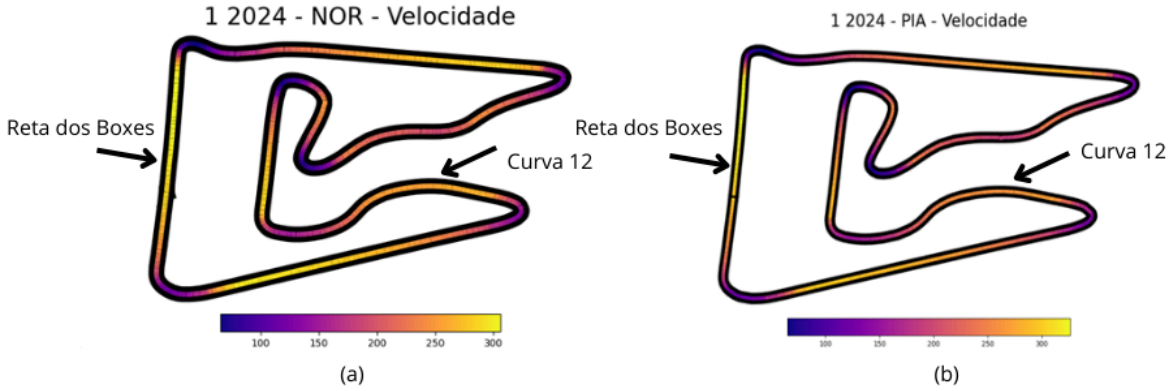
Fonte: Autoria própria.

Ao analisar os resultados obtidos na escolha do segundo piloto, Oscar Piastri, sendo da mesma equipe que Lando Norris (McLaren), foi possível observar que sua volta mais rápida foi mais lenta que a do seu companheiro de equipe, Lando Norris. Oscar Piastri teve o

tempo de volta mais rápida de aproximadamente 94,77 segundos (1 minuto, 34 segundos e 774 milissegundos), registrado na volta 37, executada, também, logo após a sua parada nos boxes para troca de pneu. Assim como os resultados da Fig. 7, na Fig. 9 observou-se, também, que o piloto manteve um ritmo consistente durante a corrida nos respectivos espaços em que ficou com o mesmo pneu.

Na análise dos gráficos de Lando Norris e Oscar Piastri, notou-se, também, que com a progressão do desgaste dos pneus, o tempo médio de volta dos pilotos aumentou e, quando ele passa do limite aceitável, há a parada nos boxes para a troca dos pneus. Outro indicativo desta tática se dá pela diminuição significativa do tempo da volta após a parada nos boxes.

Figura 12: Representação das velocidades na volta mais rápida de (a) Norris e (b) Piastri.



Fonte: Autoria própria.

Comparando-se os resultados dos dois pilotos na Fig. 12, observou-se um comportamento bem similar entre os dois carros da equipe no circuito. No entanto, foi observado que Piastri alcançou maiores velocidades de reta do que Norris, como destacado na "Reta dos Boxes". Isto se justificaria pelo ajuste do carro, pois o mesmo perdia um certo desempenho nas curvas de média velocidade, como destacado também na Curva 12.

Assim, ficou evidente como a análise dos dados tornou-se mais eficiente e acessível com o uso da API FastF1 em conjunto com as demais bibliotecas utilizadas no código Python, permitindo que as equipes tomem decisões mais ágeis e corrija possíveis problemas tanto no veículo quanto na execução dos pilotos.

5. CONSIDERAÇÕES FINAIS

Pelo presente estudo, ficou evidenciado o impacto positivo e significativo da utilização de APIs na aquisição, processamento e análise de dados esportivos, especialmente, no contexto da Fórmula 1. Assim, a utilização da API FastF1 é essencial para quem deseja trabalhar com a extração estruturada de informações, pois ela permite a automação no tratamento de dados, o que reduz os erros da manipulação manual.

A comparação entre os métodos tradicionais, baseados em arquivos CSV e a abordagem com APIs confirmou a superioridade desta última, especialmente no que diz respeito à rapidez na obtenção de dados e na confiabilidade das informações. A eliminação de etapas manuais no pré-processamento resultou em um fluxo mais eficiente, otimizando a análise e a visualização dos dados.

A integração com bibliotecas especializadas, como Pandas, Matplotlib e Seaborn, possibilita a criação de representações gráficas detalhadas, essenciais para a interpretação dos dados de telemetria e desempenho dos pilotos. A capacidade de gerar visualizações interativas, não apenas facilita a identificação de padrões estratégicos, mas também amplia o potencial de aplicação da metodologia desenvolvida.

Ao evidenciar a eficiência e a versatilidade dessas ferramentas, os resultados reforçam a importância da automação na análise de grandes volumes de dados, abrindo caminho para estudos futuros que explorem modelos preditivos, inteligência artificial e aprendizado de máquina aplicados ao automobilismo e outras modalidades esportivas.

REFERÊNCIAS

BANERJEE, A. et al. Motorsport Data Acquisition System and Live Telemetry using FPGA based CAN controller. In: **Journal of Physics: Conference Series**. IOP Publishing, 2022. p. 012041.

DE CARLO, Matteo et al. A telemetry-driven architecture for the development of data-intensive race strategies. In: **2024 International Conference on Artificial Intelligence, Computer, Data Sciences and Applications (ACDSA)**. IEEE, 2024. p. 1-6.

DE LIMA, Guilherme P. et al. Development and Implementation of a Communication Network in a Formula SAE Car. In: **2022 International Conference on Computational Science and Computational Intelligence (CSCI)**. IEEE, 2022. p. 1263-1268.

JENKINS, Mark; FLOYD, Steven W. Trajectories in the Evolution of Technology: A Multi-Level Study of Competition in Formula 1 Racing. In: **Organization Studies**, v. 22, n. 6, 2001. Disponível em: https://oss.sagepub.com. Acesso em: 20 de fev. de 2025.

JSEHV. **Speed visualization on track map**. FastF1 Documentation, [s.d.]. Disponível em: https://docs.fastf1.dev/gen_modules/examples_gallery/plot_speed_on_track.html#sphx-glr-gen-modules-examples-gallery-plot-speed-on-track-py. Acesso em: 20 de fev. de 2025.

MUGGE, E.; ZANDIEH, M. **Using Python and FastF1 to Pull and Analyze Data on Tire Degradation in a Formula 1 Grand Prix**. 2024. Disponível em: https://cisa.asu.edu/sites/default/files/2024-04/Mugge_E_Zandieh.pdf. Acesso em: 20 de fev. de 2025.

MAPFRE. **Data analysis in Formula 1: the difference between victory and defeat**. MAPFRE, 28 fev. 2020. Disponível em: https://www.mapfre.com/en/insights/innovation/data-analysis-in-formula-1-the-difference-between-victory-and-defeat/. Acesso em: 28 fev. 2025.

MSAKAMALI, Baraka. **F1 data analysis and tactical insights: exploring Formula 1 race performance strategies**. 2024. Trabalho de Conclusão de Curso. Tampere University of Applied Sciences.

PEETERS, Ronald; WESSELBAUM, Dennis. **Competitiveness in Formula One**. Sports Economic Review, v. 2, p. 100007, 2023. Disponível em: https://doi.org/10.1016/j.serev.2022.100007. Acesso em: 28 fev. 2025.

REDDY, N. M. A.; RANI, N. R. S. Open Data and APIs: Data Extraction and Exploration Using Python. **4th LIS Academy Conference on 'Open Scholarship and Libraries**', 2021.

UOL; CERASOLI, J. **Fórmula 1: nos anos 80, engenheiros brasileiros superaram tecnologia da Ferrari**. UOL Esporte, 28 fev. 2025. Disponível em: https://www.uol.com.br/esporte/reportagens-especiais/nos-anos-80-engenheiros-brasileiros-superaram-a-tecnologia-da-ferrari-na-f1/. Acesso em: 28 fev. 2025.

APÊNDICES

Apêndice A - Plotador de CSV comentado

```
1.  # Importações necessárias para o funcionamento do programa
2.  import tkinter as tk  # Biblioteca para criar a interface gráfica
3.  from tkinter import filedialog, messagebox  # Módulos para abrir caixas de diálogo e exibir mensagens
4.  import pandas as pd  # Biblioteca para manipulação de dados (leitura de CSV)
5.  import matplotlib.pyplot as plt  # Biblioteca para criar gráficos
6.  import seaborn as sns  # Biblioteca para melhorar a aparência dos gráficos
7.  from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg  # Integração de gráficos do
    matplotlib com tkinter
8.
9.  # Função para carregar e processar o arquivo CSV
10. def load_and_process_csv(file_path):
11.     # Lista de codificações de texto para tentar ler o arquivo CSV
12.     encodings_to_try = ['utf-8', 'latin-1', 'iso-8859-1', 'utf-16']
13.
14.     # Tenta ler o arquivo CSV com cada codificação da lista
15.     for encoding in encodings_to_try:
16.         try:
17.             # Tenta ler o arquivo CSV com o delimitador ';' e a codificação atual
18.             data = pd.read_csv(file_path, delimiter=';', encoding=encoding)
19.             print(f"Arquivo carregado com sucesso usando a codificação: {encoding}")
20.             break  # Se o arquivo for lido com sucesso, interrompe o loop
21.         except UnicodeDecodeError:
22.             print(f"Falha ao carregar o arquivo usando a codificação: {encoding}")
23.
24.     # Verifica se a coluna 'Data' está presente no DataFrame
25.     if 'Data' in data.columns:
26.         # Se a coluna 'Data' não estiver no formato de data, converte para datetime
27.         if not pd.api.types.is_datetime64_any_dtype(data['Data']):
28.             data['Data'] = pd.to_datetime(data['Data'])
29.     else:
30.         # Se a coluna 'Data' não existir, exibe uma mensagem de erro e interrompe a função
31.         messagebox.showerror("Erro", "A coluna 'Data' não está presente no DataFrame.")
32.         return
33.
34.     # Verifica se a coluna 'Calorias Queimadas' está presente no DataFrame
35.     if 'Calorias Queimadas' not in data.columns:
36.         # Se a coluna não existir, exibe uma mensagem de erro e interrompe a função
37.         messagebox.showerror("Erro", "A coluna 'Calorias Queimadas' não está presente no DataFrame.")
38.         return
39.
40.     # Remove a unidade 'kcal' da coluna 'Calorias Queimadas' e converte os valores para inteiros
41.     data['Calorias Queimadas'] = data['Calorias Queimadas'].str.replace(' kcal', '').astype(int)
42.
43.     # Cria o gráfico de linhas usando seaborn
44.     plt.figure(figsize=(10, 6))  # Define o tamanho da figura do gráfico
45.     sns.lineplot(x='Data', y='Calorias Queimadas', data=data)  # Cria o gráfico de linhas
46.     plt.title('Variação de Calorias Queimadas ao Longo do Tempo')  # Título do gráfico
47.     plt.xlabel('Data')  # Rótulo do eixo X
48.     plt.ylabel('Calorias Queimadas')  # Rótulo do eixo Y
49.     plt.xticks(rotation=45)  # Rotaciona os rótulos do eixo X para melhor legibilidade
50.     plt.tight_layout()  # Ajusta o layout para evitar cortes nos elementos do gráfico
51.
```

```
    # Integra o gráfico com a interface gráfica do tkinter
    canvas = FigureCanvasTkAgg(plt.gcf(), master=root)  # Converte o gráfico em um widget do tkinter
    canvas_widget = canvas.get_tk_widget()  # Obtém o widget do gráfico
    canvas_widget.pack(fill=tk.BOTH, expand=True)  # Adiciona o gráfico à janela principal
    canvas.draw()  # Renderiza o gráfico na interface

    # Oculta a mensagem inicial e o botão de abrir arquivo após a geração do gráfico
    initial_message.pack_forget()
    open_button.pack_forget()

# Função para abrir a caixa de diálogo e carregar o arquivo CSV
def open_file():
    # Abre uma caixa de diálogo para o usuário selecionar um arquivo CSV
    file_path = filedialog.askopenfilename(
        filetypes=[("CSV files", "*.csv")],  # Filtra apenas arquivos CSV
        title="Selecionar Arquivo CSV"  # Título da caixa de diálogo
    )
    # Se o usuário selecionar um arquivo, chama a função para processá-lo
    if file_path:
        load_and_process_csv(file_path)

# Cria a janela principal da aplicação
root = tk.Tk()
root.title("Carregador de Arquivo CSV")  # Define o título da janela

# Adiciona uma mensagem inicial à interface gráfica
initial_message = tk.Label(root, text="Selecione o arquivo CSV para que o gráfico seja gerado.",
font=("Helvetica", 12))
initial_message.pack(pady=20, padx=20)  # Adiciona a mensagem à janela com espaçamento

# Cria um botão para abrir o arquivo CSV
open_button = tk.Button(root, text="Abrir Arquivo CSV", command=open_file)
open_button.pack(pady=10, padx=10)  # Adiciona o botão à janela com espaçamento

# Inicia o loop principal da interface gráfica
root.mainloop()
```

Apêndice B - Código com a Fastf1 comentado

```
1.  # Importações necessárias para o funcionamento do programa
2.  import tkinter as tk  # Biblioteca para criar a interface gráfica
3.  from tkinter import ttk, messagebox  # Módulos para widgets(elementos de interação) avançados e
    exibição de mensagens
4.  import fastf1 as ff1  # Biblioteca para acessar dados da Fórmula 1
5.  import pandas as pd  # Biblioteca para manipulação de dados
6.  import matplotlib.pyplot as plt  # Biblioteca para criar gráficos
7.  from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg  # Integração de gráficos do
    matplotlib com tkinter
8.  import numpy as np  # Biblioteca para cálculos numéricos
9.  from matplotlib.collections import LineCollection  # Para criar coleções de linhas no gráfico
10. import matplotlib as mpl  # Biblioteca para personalização de gráficos
11. import os  # Para manipulação de arquivos e diretórios
12. import threading  # Para executar operações em segundo plano
13. from datetime import datetime  # Para obter o ano atual
14.
15. # Criar diretório de cache se não existir
16. if not os.path.exists('cache'):
17.     os.makedirs('cache')
18.
19. # Inicializar cache do FastF1
20. ff1.Cache.enable_cache('cache')  # Opcional, mas recomendado para acelerar o carregamento de dados
21.
22. # Função para buscar e exibir dados
23. def fetch_data():
24.     def fetch():
25.         season = int(season_var.get())  # Obtém a temporada selecionada
26.         circuit = circuit_var.get()  # Obtém o circuito selecionado
27.         driver = driver_var.get()  # Obtém o piloto selecionado
28.
29.         try:
30.             # Carrega a sessão de corrida (R) para a temporada e circuito selecionados
31.             session = ff1.get_session(season, circuit, 'R')
32.             session.load()  # Carrega os dados da sessão
33.         except Exception as e:
34.             # Exibe uma mensagem de erro se houver falha ao carregar os dados
35.             messagebox.showerror("Erro", f"Falha ao carregar dados da sessão: {e}")
36.             return
37.
38.         # Filtra as voltas do piloto selecionado
39.         driver_laps = session.laps.pick_drivers([driver])
40.
41.         # Verifica se há dados para o piloto
42.         if driver_laps.empty:
43.             messagebox.showinfo("Sem Dados", f"Nenhum dado encontrado para o piloto {driver}")
44.             return
45.
46.         # Exibe informações do piloto em uma janela pop-up
47.         fastest_lap = driver_laps['LapTime'].min()  # Encontra a volta mais rápida
48.         messagebox.showinfo("Informações do Piloto", f"Piloto: {driver}\nVolta mais rápida: {fastest_lap}")
49.
50.         # Limpa o gráfico anterior, se houver
51.         for widget in plot_frame.winfo_children():
52.             widget.destroy()
53.
```

```
        # Cria uma nova janela para exibir o gráfico de tempos de volta
        plot_window = tk.Toplevel(root)
        plot_window.title(f"Tempos de Volta para {driver}")

        # Cria o gráfico
        fig, ax = plt.subplots()

        # Obtém os números das voltas e os tempos de volta em segundos
        lap_numbers = driver_laps['LapNumber']  # Números das voltas
        lap_times = driver_laps['LapTime'].dt.total_seconds()  # Tempos de volta convertidos para segundos

        # Plota os tempos de volta
        ax.plot(lap_numbers, lap_times, marker='o', linestyle='-', color='blue', label='Tempo de Volta')
        ax.set_xlabel("Número da Volta")  # Rótulo do eixo X
        ax.set_ylabel("Tempo de Volta (segundos)")  # Rótulo do eixo Y
        ax.set_title(f"Tempos de Volta para {driver}")  # Título do gráfico
        ax.grid(True)  # Adiciona uma grade para melhor legibilidade

        # Adiciona uma anotação para a volta mais rápida
        fastest_lap_index = lap_times.idxmin()  # Índice da volta mais rápida
        fastest_lap_number = lap_numbers[fastest_lap_index]  # Número da volta mais rápida
        fastest_lap_time = lap_times[fastest_lap_index]  # Tempo da volta mais rápida
        ax.annotate(f'Volta mais rápida: {fastest_lap_time:.2f}s',
                    xy=(fastest_lap_number, fastest_lap_time),
                    xytext=(fastest_lap_number, fastest_lap_time + 5),
                    arrowprops=dict(facecolor='red', shrink=0.05),
                    fontsize=10, color='red')

        ax.legend()  # Adiciona uma legenda ao gráfico

        # Integra o gráfico na janela do Tkinter
        canvas = FigureCanvasTkAgg(fig, master=plot_window)
        canvas.draw()
        canvas.get_tk_widget().pack()

    # Executa a função fetch em uma thread separada para evitar travar a interface
    threading.Thread(target=fetch).start()

# Função para mostrar a visualização de velocidade no mapa da pista
def show_track_map():
    season = int(season_var.get())  # Obtém a temporada selecionada
    circuit = circuit_var.get()  # Obtém o circuito selecionado
    driver = driver_var.get()  # Obtém o piloto selecionado

    try:
        # Carrega a sessão de corrida (R) para a temporada e circuito selecionados
        session = ff1.get_session(season, circuit, 'R')
        session.load()  # Carrega os dados da sessão
    except Exception as e:
        # Exibe uma mensagem de erro se houver falha ao carregar os dados
        messagebox.showerror("Erro", f"Falha ao carregar dados da sessão: {e}")
        return

    # Filtra a volta mais rápida do piloto selecionado
    lap = session.laps.pick_drivers([driver]).pick_fastest()

    # Verifica se há dados para o piloto
```

```
    if lap is None:
        messagebox.showinfo("Sem Dados", f"Nenhum dado encontrado para o piloto {driver}")
        return

    # Obtém os dados de telemetria (posição X, Y e velocidade)
    x = lap.telemetry['X']  # Coordenadas X da pista
    y = lap.telemetry['Y']  # Coordenadas Y da pista
    color = lap.telemetry['Speed']  # Velocidade para o gradiente de cor

    # Cria segmentos de linha para o gráfico
    points = np.array([x, y]).T.reshape(-1, 1, 2)
    segments = np.concatenate([points[:-1], points[1:]], axis=1)

    # Cria o gráfico do mapa da pista
    fig, ax = plt.subplots(figsize=(12, 6.75))
    fig.suptitle(f'{session.event.name} {season} - {driver} - Velocidade', size=24, y=0.97)

    # Ajusta as margens e desliga os eixos
    plt.subplots_adjust(left=0.1, right=0.9, top=0.9, bottom=0.12)
    ax.axis('off')

    # Plota a linha de fundo da pista
    ax.plot(lap.telemetry['X'], lap.telemetry['Y'],
            color='black', linestyle='-', linewidth=16, zorder=0)

    # Cria um gradiente de cores para representar a velocidade
    norm = plt.Normalize(color.min(), color.max())
    lc = LineCollection(segments, cmap=mpl.cm.plasma, norm=norm,
                        linestyle='-', linewidth=5)

    # Define os valores usados para o mapeamento de cores
    lc.set_array(color)

    # Adiciona os segmentos de linha ao gráfico
    line = ax.add_collection(lc)

    # Adiciona uma barra de cores como legenda
    cbaxes = fig.add_axes([0.25, 0.05, 0.5, 0.05])
    normlegend = mpl.colors.Normalize(vmin=color.min(), vmax=color.max())
    legend = mpl.colorbar.ColorbarBase(cbaxes, norm=normlegend, cmap=mpl.cm.plasma,
                                       orientation="horizontal")

    # Exibe o mapa da pista em uma nova janela
    track_map_window = tk.Toplevel(root)
    track_map_window.title(f"Mapa da Pista para {driver}")

    canvas = FigureCanvasTkAgg(fig, master=track_map_window)
    canvas.draw()
    canvas.get_tk_widget().pack()

# Função para atualizar circuitos e pilotos com base na temporada selecionada
def update_season_data(event=None):
    season = int(season_var.get())  # Obtém a temporada selecionada

    try:
        # Obtém a lista de circuitos para a temporada selecionada
        schedule = ff1.get_event_schedule(season)
        circuits = schedule['EventName'].tolist()
```

```
        circuit_combobox['values'] = circuits  # Atualiza os valores do combobox de circuitos
        circuit_combobox.current(0)  # Define o primeiro circuito como padrão
    except Exception as e:
        # Exibe uma mensagem de erro se houver falha ao buscar os circuitos
        messagebox.showerror("Erro", f"Falha ao buscar circuitos: {e}")
        return

    try:
        # Carrega a sessão de corrida para o primeiro circuito da temporada
        session = ff1.get_session(season, circuits[0], 'R')
        session.load()
        drivers = session.drivers  # Obtém a lista de pilotos
        driver_combobox['values'] = [session.get_driver(driver)['Abbreviation'] for driver in drivers]
        driver_combobox.current(0)  # Define o primeiro piloto como padrão
    except Exception as e:
        # Exibe uma mensagem de erro se houver falha ao buscar os pilotos
        messagebox.showerror("Erro", f"Falha ao buscar pilotos: {e}")

# Cria a janela principal
root = tk.Tk()
root.title("Visualizador de Dados da F1")

# Cria e posiciona os widgets
ttk.Label(root, text="Temporada:").grid(row=0, column=0, padx=10, pady=10)
season_var = tk.StringVar()
season_combobox = ttk.Combobox(root, textvariable=season_var)

# Define o intervalo de temporadas de 2018 a 2024 (ou o ano atual, o que for menor)
current_year = datetime.now().year  # Obtém o ano atual
max_season = min(current_year, 2024)  # Limita a 2024 ou ao ano atual
season_combobox['values'] = list(range(2018, max_season + 1))  # Define os valores do combobox
season_combobox.grid(row=0, column=1, padx=10, pady=10)
season_combobox.current(max_season - 2018)  # Define a temporada mais recente como padrão
season_combobox.bind('<<ComboboxSelected>>', update_season_data)  # Atualiza os circuitos e
    pilotos ao mudar a temporada

ttk.Label(root, text="Circuito:").grid(row=1, column=0, padx=10, pady=10)
circuit_var = tk.StringVar()
circuit_combobox = ttk.Combobox(root, textvariable=circuit_var)
circuit_combobox.grid(row=1, column=1, padx=10, pady=10)

ttk.Label(root, text="Piloto:").grid(row=2, column=0, padx=10, pady=10)
driver_var = tk.StringVar()
driver_combobox = ttk.Combobox(root, textvariable=driver_var)
driver_combobox.grid(row=2, column=1, padx=10, pady=10)

fetch_button = ttk.Button(root, text="Buscar Dados", command=fetch_data)
fetch_button.grid(row=3, column=0, columnspan=2, pady=10)

track_map_button = ttk.Button(root, text="Mostrar Mapa da Pista", command=show_track_map)
track_map_button.grid(row=4, column=0, columnspan=2, pady=10)

result_label = ttk.Label(root, text="")
result_label.grid(row=5, column=0, columnspan=2, pady=10)

plot_frame = ttk.Frame(root)
plot_frame.grid(row=6, column=0, columnspan=2, padx=10, pady=10)

```

```
226. # Inicializa os dados para a temporada padrão
227. update_season_data()
228. 
229. # Inicia o loop principal da interface gráfica
230. root.mainloop()
```