PREDICCIÓN DE RESULTADOS DE FÓRMULA 1 CON MACHINE LEARNING

Autores: Alex Lopez & Aaron Cano

Nombre del curso: ASIX 2B

Fecha de entrega: 12/04/2025

Nombre de la institución: Institut Puig Castellar

Índice

1. Quienes somos

Somos Alex López y Aaron Cano, dos alumnos de segundo curso del ciclo formativo de Administración de Sistemas Informáticos en Red. Este proyecto es nuestro trabajo final de curso, donde aplicamos lo aprendido a lo largo de estos dos años para desarrollar una solución completa basada en predicción de datos, automatización y visualización en tiempo real.

2. Introducción

En los últimos años, la inteligencia artificial y el machine learning han avanzado de manera significativa, alcanzando múltiples ámbitos, incluida la Fórmula 1. Actualmente, durante la retransmisión de las carreras, se nos muestran datos generados por IA en tiempo real, pero aún no existe una herramienta que los genere antes de la carrera para anticipar posibles resultados.

Este proyecto tiene como objetivo desarrollar CurvaIV, una plataforma web capaz de predecir los resultados de las carreras de F1 mediante técnicas de aprendizaje automático. Para ello, se recopilarán y analizarán distintos datos de los pilotos y escuderías.

Este tipo de sistemas no solo resultan beneficiosos para los aficionados que desean conocer predicciones antes de cada carrera, sino también para analistas y escuderías, quienes pueden aprovechar estas herramientas para mejorar su planificación y toma de decisiones.

A lo largo de este documento, exploraremos en profundidad las distintas tecnologías que utilizaremos para desarrollar CurvaIV y cómo esta herramienta mejorará el análisis de la Fórmula 1.

2.1 Justificación

Nos adentramos en el mundo de la Fórmula 1 con la idea de desarrollar una inteligencia artificial fiable y capaz de predecir los resultados de los grandes premios. Convencidos de su potencial, decidimos crear nuestra propia herramienta para entrenarla y obtener predicciones sobre la fomula 1.

Al principio, pensamos en su utilidad a nivel personal, pero pronto nos dimos cuenta de que podía ser de gran ayuda para las escuderías y los aficionados. Aún así, queríamos llevar nuestra idea más allá, mejorando y ampliando sus posibilidades.

Fue entonces cuando surgió la clave: convertirla en una plataforma web que no solo permitiera hacer predicciones, sino también recopilar y analizar datos, ofrecer información valiosa sin perder de vista nuestro objetivo principal: anticipar los resultados de las carreras con la mayor precisión posible.

2.2 Objetivos

2.2.1 Objetivo general

En CurvaIV, nuestro proyecto de recopilación y pronóstico de carreras, nuestro principal objetivo es desarrollar una plataforma que permita a los espectadores, equipos, periodistas... acceder a datos que ofrecemos, de los pilotos y escuderías para predecir los resultados de las carreras, gracias a nuestros datos y nuestra “IA”.

Para ello, implementaremos un proceso de entrenamiento de modelos de Machine Learning, capaz de estimar y predecir los resultados de las próximas carreras. Este modelo se basará en un análisis de los datos de la última Qualy (carreras de clasificación para ver en el puesto que saldrá cada piloto), las últimas dos carreras, los mejores coches, el talento, la experiencia y consistencia.

El Machine Learning es una disciplina del campo de la Inteligencia Artificial que, a través de algoritmos, dota a los ordenadores de la capacidad de identificar patrones en datos masivos y elaborar predicciones (análisis predictivo).

2.2.2 Objetivos específicos

- Desarrollar un modelo predictivo basado en técnicas de aprendizaje para predecir los resultados de las carreras de Fórmula 1.
- Analizar el impacto de diferentes factores en los resultados, las estadísticas de los equipos y pilotos y el historial de carreras.
- Crear una herramienta que pueda ser utilizada por analistas deportivos y entrenadores para optimizar estrategias.
- Evaluar la precisión de diferentes modelos y seleccionar el más eficiente según los resultados obtenidos.
- Crear una serie de gráficos que ayuden a visualizar estas comparaciones de una manera más gráfica, haciéndolo más vistoso y sencillo para el fan promedio.
- Tener un backup para no perder información y datos de CurvaIV y una seguridad en la web para que los usuarios puedan sentirse cómodos dentro de ella.

2.3 Estrategia y planificación del proyecto

Desde un inicio nos dimos cuenta que el proyecto iba a requerir de mucho tiempo y realizar varias tareas simultáneamente. En la misma semana pensamos la idea y, desde ese día, comenzamos a planear todo lo relacionado con el proyecto. En planear todo tardamos alrededor de un mes, ya que íbamos a usar tecnologías de las que no habíamos hecho uso directo.

Ya sabiendo la estructura, nos pusimos directos recopilando datos para tener el FastAPI, tales como: clasificaciones, equipos, resultados… . Mientras hacíamos esto, ya estábamos haciendo otras tareas. Acabando la recopilación de datos, supimos que era el momento idóneo para comenzar la documentación y ceñirnos a las fechas de entrega de cada parte del proyecto.

La planificación que habíamos planteado nos funcionó. Mientras desarrollamos todas las tecnologías que iba a usar CurvaIV, íbamos haciendo pruebas y programando la web. Además, mientras íbamos avanzando, hacíamos la presentación para tenerla lista antes de las fechas establecidas y así no tener una carga detrás.

			Febrero				Marzo					Abril			Mayo				
	Semana	Inicio	3	10	17	24	3	10	17	24	31	7	14	21	28	5	12	19	26
		Final	9	16	23	2	9	16	23	30	6	13	20	27	4	11	18	25	1
Tarea	Inicio	Final																	
Lluvia de ideas	3/2/2025	6/2/2025																	
Planteamiento inicial	6/2/2025	28/2/2025																	
Recopilación de datos	4/3/2025	26/3/2025																	
Documentación	31/3/2025	25/5/2025																	
Desarrollo de tecnologías	25/3/2025	16/5/2025																	
Pruebas	28/3/2025	18/5/2025																	
Programación web	28/4/2025	18/5/2025																	
Presentación	26/5/2025	31/5/2025																	

Llevando ya varias semanas de testeo con los datos que habíamos obtenido, empezaban a surgir los primeros problemas. La idea principal que teníamos era hacer lo mismo pero llevándolo al mundo del fútbol, el gran problema que estábamos teniendo es que el fútbol es un deporte en el cual se registran cientos datos por partido y por jugador, lo cual hacía que recopilar estos fuese una tarea bastante difícil. Esto se debe a que la mayoría de api 's que nos proporcionaban los datos que necesitábamos eran de pago y bastante difíciles de implementar.

Pero ese no era nuestro único problema que debíamos resolver, una vez teníamos los suficientes datos para realizar nuestra primera Demo, nos encontramos con el problema de que a la hora de hacer las predicciones la IA no las realizaba correctamente, esto era debido al gran número de datos que tenía que manejar, ya que tenia que manejar datos como, los últimos partidos de cada equipo, los últimos partidos de los equipos que se enfrentaban entre ellos, posesiones, tiros a puerta, corners….

Así que viendo que íbamos a tener cada vez más errores por la gran cantidad de datos usados, decidimos modificar la idea ligeramente. Todo esto nos llevó a hacer un cambio de guión y mover nuestra idea de hacer predicciones de fútbol a hacerlas en el mundo del motor, más concretamente de la Fórmula 1, esto se debía ya que en este deporte necesitamos menos datos que el fútbol.

3. Descripción del proyecto

3.1 Análisis de requisitos

Para la parte de la implementación del sistema predictivo, hemos deducido que nos harían falta diferentes cosas, deberíamos conseguir tener acceso a bases de datos de las carreras y que estas se vayan actualizando periódicamente con datos nuevos. Los objetivos o los requisitos que queremos cumplir son los siguientes:

- Implementación de modelos de machine learning para generar predicciones.
- Desarrollo de una API con FastAPI para facilitar la consulta de las predicciones.
- Creación de dashboards en Grafana para la visualización de resultados y tendencias.
- Desarrollo de una página web simple con Apache que sirva como punto de acceso centralizado.

3.2 Tecnologías

Para el correcto desarrollo del proyecto, utilizaremos diferentes herramientas y tecnologías que nos faciliten la implementación y despliegue del sistema.

Requerimientos → Todas estas tecnologías son dependencias de nuestro proyecto en Python, para poder llevar a cabo el trabajo necesitamos todas y cada una de ellas para que el código se pueda ejecutar de buena manera. Esto lo tenemos dentro de un fichero que contiene todas estas dependencias, este fichero se llama requirements.txt

Este fichero contiene varias librerías fundamentales. Pandas se encarga de la manipulación y análisis de datos en formato tabular. Numpy permite realizar cálculos numéricos de forma eficiente. Scikit-learn proporciona las herramientas necesarias para aplicar técnicas de aprendizaje automático, como clasificación y regresión. Scipy ofrece funciones matemáticas avanzadas que complementan el procesamiento de datos. Por último, fastf1 facilita el acceso estructurado a datos históricos y en tiempo real de Fórmula 1, fundamentales para alimentar el sistema de predicción.

FastAPI → Es un framework moderno y de alto rendimiento para la creación de APIs en Python. Su principal ventaja es la facilidad de uso y la rapidez con la que permite desarrollar servicios web escalables. En nuestro proyecto, usaremos "fastf1" https://docs.fastf1.dev/ esta se encarga de gestionar y ofrecer toda los datos al día de la Formula 1 y así tenerlo a mano en caso de tener que hacer uso de ellos.

Grafana → Es una plataforma de código abierto que se usa para la visualización y monitoreo de datos en tiempo real. Lo que nos deja hacer es crear dashboards interactivos que facilitan el análisis de métricas , diversas fuentes como bases de datos, servicios web y herramientas de monitoreo. En nuestro proyecto, Grafana se encargará de mostrar de manera visual los CSV de las predicciones vamos a acompañar Grafana con el plugin CSV Datasource de Marcus Olsson.

Crontab → Esta herramienta de linux se usa para realizar tareas periódicamente sin tener que ejecutarlas, o lo que es lo mismo, puedes programar cuando quieres que esta se realice. Así que aprovechándonos de su utilidad decidimos implementarla en nuestro proyecto. El uso que le estamos dando es para que los resultados y las estadísticas de la semana se vayan actualizando periódicamente y así nosotros poder olvidarnos de tener que ejecutar el script semana a semana.

GitHub → Es una plataforma en la nube que permite a sus usuarios almacenar, compartir, trabajar y editar código de forma colaborativa. Funciona a través de repositorios, donde se gestionan y controlan los archivos de los proyectos. En CurvaIV usamos GitHub para guardar todos nuestros códigos en la nube. Además, nuestro repositorio es de código abierto, lo que significa que cualquier usuario puede acceder libremente a nuestros códigos de predicción, fomentando la colaboración y el aprendizaje compartido.
Este es nuestro GitHub: https://github.com/AlexitoLopez5/Curva-IV/

Apache → Apache es un programa de código abierto de los más utilizados para crear y servir aplicaciones web. Es altamente configurable y compatible con múltiples tecnologías, como PHP, Python y aplicaciones basadas en FastAPI mediante reverse proxy. En cuanto a nuestro proyecto se encarga de alojar la página web en la cual vamos a poder ver nuestras predicciones, gráficos….

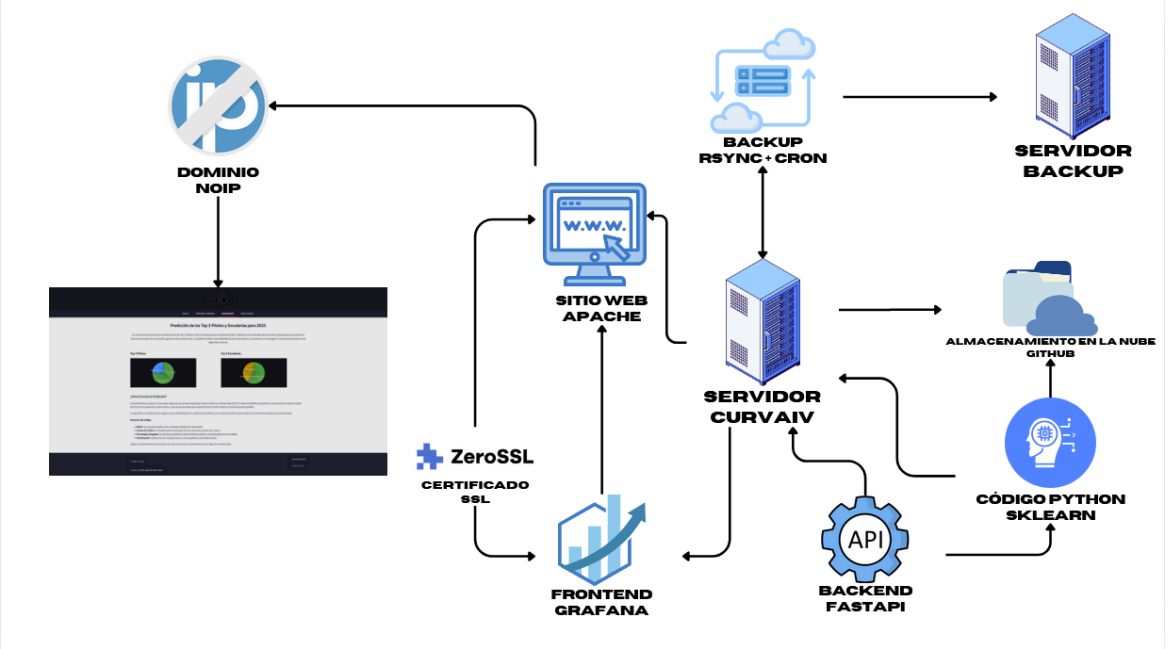
VirtualBox → VirtualBox es una aplicación que permite a cualquier usuario crear máquinas virtuales mediante ISOs. Nosotros lo usaremos para alojar nuestro servidor, ya que nos permitirá tener un control absoluto en él, es decir, tener todos los permisos para no tener ningún tipo de limitación.

Scikit-Learn → Scikit-Learn, conocido como "sklearn", es una herramienta de software libre y una extensión de Python que sirve para el aprendizaje automático. En el proyecto, su uso es para estimar y realizar resultados con los datos que nosotros le ofrecemos de forma automática.

rsync + Cron → Crontab es un planificador de tareas que nos permite ejecutar comandos automáticamente en intervalos de tiempo definidos. Rsync es una herramienta utilizada para sincronizar archivos y directorios de manera eficiente entre ubicaciones locales o remotas. Combinaremos ambas herramientas para automatizar copias de seguridad en otro servidor, asegurando que nuestros datos estén siempre sincronizados y protegidos sin necesidad de intervención manual.

No-IP + ZeroSSL → No-IP es un servicio que permite asociar una dirección IP a un nombre de dominio de forma gratuita. ZeroSSL es un servicio que ofrece certificados SSL públicos, lo que permite que cualquier usuario pueda conectarse a la web de forma segura y sin recibir advertencias en el navegador. En CurvaIV utilizamos No-IP para gestionar nuestro dominio y ZeroSSL para obtener un certificado SSL público, configurando tanto Apache como Grafana para que funcionen con HTTPS y así garantizar conexiones seguras a nuestra web.

3.3 Estructura del proyecto



4. Desarrollo de proyecto

4.1 Identidad

Para dar una identidad, una forma de dar vida al proyecto es crear un logo relacionado con la Fórmula 1 y con nuestro proyecto. También era importante que nos gustara personalmente.

Primero, estuvimos investigando qué color usar según lo relacionado con el marketing, y nos decantamos por el color negro. Además de ser un color esencial, nos gustó su significado: poder, prestigio y elegancia. Aparte, era un color que quedaba bien con todo el mundo de la Fórmula 1, ya que las pistas son negras, los coches tienden a usar el color negro para ahorrar peso, las ruedas son negras y más.

Después, miramos qué tipografía usar y encontramos una que simulaba el aire que se genera cuando hay un objeto a gran velocidad, tal como pasa en la Fórmula 1 con sus coches.

Por último, queríamos disimular el número 4 entre el texto, así que antes de la V pusimos una bandera de carreras simulando el número 1, y así poner el número 4 en números romanos, IV, y agrandando el número, dándole el nombre de “Curva IV”. Este nombre se basa en que, para identificar cada curva en los circuitos de Fórmula 1, se les asigna un número, y nos pareció estético el número IV.

4.2 Configuración Servidor

Nuestro servidor es una máquina virtual con un software de Ubuntu Server, versión 22.04, ya que consideramos que es una buena máquina, no necesita tantos recursos y aun así funciona excelentemente a nuestras necesidades. Además, tenemos bastante más experiencia usando este software en vez de otros.

Hemos configurado nuestro servidor de forma que tenga Adaptador puente, para poder usar el internet con la configuración base.

4.3 Configuración Requisitos

Para iniciar con los scripts, es necesario instalar varias extensiones de Python fundamentales para el procesamiento y análisis de datos.

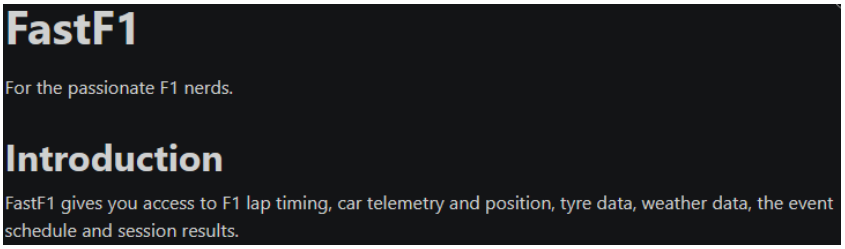
En primer lugar, pandas es clave para trabajar con estructuras de datos en forma de tablas, lo que permite organizar, limpiar y transformar la información de manera eficiente. Junto a esta, numpy complementa el análisis numérico ofreciendo herramientas para manejar arrays y operaciones matemáticas de alto rendimiento.

Por otro lado, scikit-learn hace la función de preprocesamiento de datos y en tareas de machine learning. En este proyecto se utilizan componentes como MinMaxScaler, que permite normalizar los datos para que estén en una escala común, y check_array, que valida la estructura de los datos antes de ser procesados por los modelos. Estas herramientas aseguran que los datos estén correctamente preparados y listos para ser utilizados en algoritmos de aprendizaje automático, lo cual es esencial para obtener resultados precisos.

4.4 Configuración Fast API

Para recoger datos en tiempo real de la F1, estuvimos investigando diferentes APIs disponibles, y estuvimos considerando varias opciones como Formula One API (https://documenter.getpostman.com/view/11586746/SztEa7bL),
OpenF1 (https://openf1.org/), y FastF1 (https://docs.fastf1.dev/).

Nos decidimos por FastF1 debido a que es una API libre y se integra perfectamente con Python, lo que facilita su implementación en nuestro proyecto. FastF1 ofrece acceso a una gran cantidad de datos de la F1, incluyendo tiempos en vivo, resultados de carreras y más, lo que es la opción ideal para nuestra necesidad de obtener información en tiempo real.



4.5 Configuración Scripts

Para llevar a cabo las predicciones de nuestro proyecto CurvaIV, hemos desarrollado un total de cinco scripts. Tres de ellos sirven para la recolección de datos y conversión a formato CSV, mientras que los otros dos están destinados a la predicción.

Recolección de Datos

1. script_carreras.py

   Este script obtiene los resultados de las dos últimas carreras disputadas antes del Gran Premio actual. Utiliza la API FastF1 para cargar la sesión de carrera ('R') y guarda los datos relevantes (posición, piloto, equipo, etc.) en un archivo CSV. También implementa un sistema para filtrar automáticamente solo las carreras ya finalizadas según la fecha actual.

   Este script se ejecuta antes de cada carrera, como parte de la preparación del dataset.

2. script_coches.py

   Este script recoge los mejores tiempos de los equipos durante los test de pretemporada, se encuentran en la primera ronda del calendario. Se extrae el mejor tiempo por equipo y se exporta un ranking ordenado en CSV. Este archivo ofrece una referencia comparativa del rendimiento base de cada coche al inicio del campeonato.

3. script_qualy.py

   Este script se encarga de obtener los resultados de la última sesión de clasificación (qualy) disputada antes de la carrera actual. Filtra el evento más reciente respecto a la fecha actual y carga la sesión correspondiente. Luego, exporta a CSV la posición de cada piloto junto a su equipo.

   Al igual que script_carreras.py, este script también se ejecuta antes de la carrera, ya que la posición de salida influye directamente en la predicción.

Esta fase de recopilación es esencial para alimentar correctamente los algoritmos de machine learning que vienen a continuación.

Procesamiento y Predicción

1. prediccion.py

   Este script fusiona los datos recopilados (carreras, qualy, coches, experiencia, talento y consistencia (estos dos últimos combinándolos como “habilidad”), realiza un preprocesamiento, y calcula múltiples scores ponderados. Además cada circuito cuenta con un porcentaje de valoración de los anteriores factores distintos, ya que todos no son iguales y algunos se identifican mas por cierta razón qualy, coche, experiencia... Para hacer la predicción se utiliza la librería scikit-learn para normalizar las métricas antes de combinar los factores en un score final. Finalmente, genera un ranking de probabilidades de victoria por piloto.

   Este codigo se ejecuta manualmente después de la Qualy de ese gran premio, ya que debemos seleccionar que circuito se disputa.

2. top3.py

   Procesa los resultados de todas las carreras disputadas hasta el momento, calculando los podios acumulados por piloto y equipo. Los transforma en rankings porcentuales y los exporta en CSV para Grafana. Usa LabelEncoder de sklearn para codificar nombres como preparación para análisis avanzados.

   Se ejecuta automáticamente después de cada carrera, actualizando el historial de podios.

4.5.1 Primeros resultados

En los primeros resultados del modelo, se logró una alta precisión sin considerar aún la experiencia de los pilotos. Se usaban solo datos de qualy, test del coche y las dos últimas carreras.

Resultados al incorporar la experiencia:

Predicción	Realidad	Diferencia
Verstappen	1°	Exacto
Norris	2°	Exacto
Piastri	3°	Exacto
Russell	5°	+1
Leclerc	4°	-1
Antonelli	6°	Exacto
Albon	9°	+2
Hadjar	8°	Exacto
Hamilton	7°	-2
Bearman	10°	Exacto

La experiencia fue clave para mejorar. Se detectó que pilotos como Hamilton estaban infravalorados sin este dato. Por eso se añadió un campo manual con años de experiencia y se integró al modelo con un peso bajo pero importante. Esto mejoró aún más las predicciones.

Con el tiempo fuimos mejorándolo, incorporando atributos como talento y consistencia para afinar las predicciones. Finalmente, en el sistema actual llamado “v2 Senna”, añadimos que las métricas de cada circuito se valoren de forma distinta, adaptando los pesos según las características específicas de cada pista.

4.6 Configuración Gráficos (Grafana)

Se configuraron 4 gráficos en Grafana usando el plugin CSV Datasource de Marcus Olsson. Este plugin permite leer directamente archivos CSV ubicados en el sistema, facilitando la visualización de datos sin necesidad de bases de datos.

Gráficos creados

1. Top 10 Predicción de Carrera (tabla)

   Muestra una tabla con los 10 pilotos con mayor probabilidad de victoria. Usa el Fichero: /var/lib/grafana/csv/curva4.csv

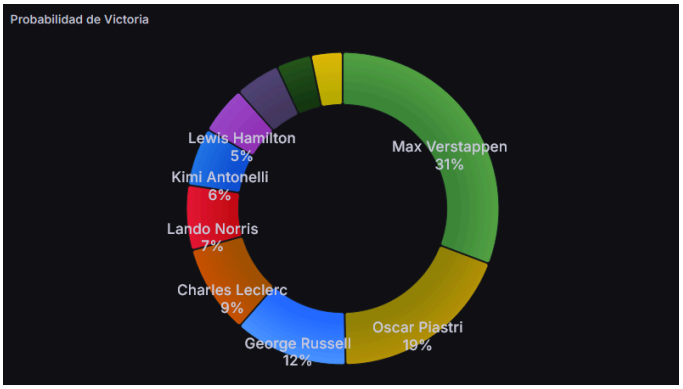
Predicción Proxima Carrera

Ranking	Piloto	Equipo	Probabilidad_Victoria
1	Max Verstappen	Red Bull Racing	26.8
2	Oscar Piastri	McLaren	16.5
3	George Russell	Mercedes	10.2
4	Charles Leclerc	Ferrari	8.0
5	Lando Norris	McLaren	5.8
6	Kimi Antonelli	Mercedes	5.2
7	Lewis Hamilton	Ferrari	4.3
8	Carlos Sainz	Williams	4.0
9	Yuki Tsunoda	Red Bull Racing	3.1
10	Pierre Gasly	Alpine	2.8

2. Probabilidad de Victoria (gráfico circular)

   Muestra un gráfico de tipo "donut" con los porcentajes de victoria, que corresponden al porcentaje de ocupación que cada piloto tiene en el gráfico, redondeado.

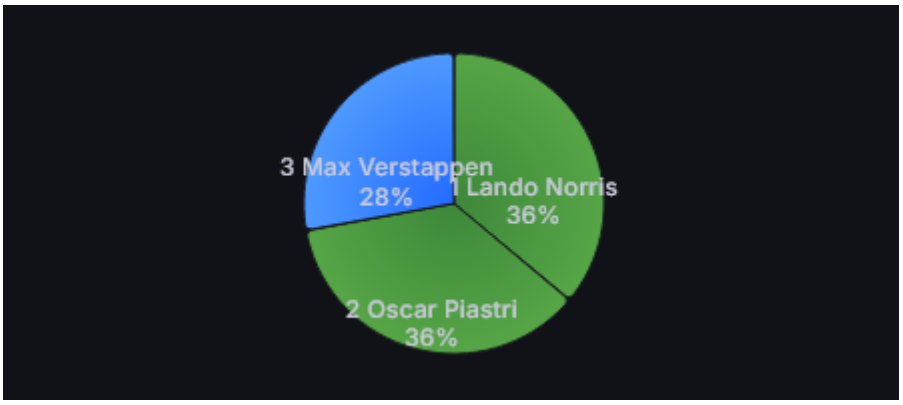
   Fichero: /var/lib/grafana/csv/queso_curva4.csv



3. Top 3 Pilotos del Torneo (queso)

   Gráfico de queso con los tres pilotos con la predicción de fin de temporada.

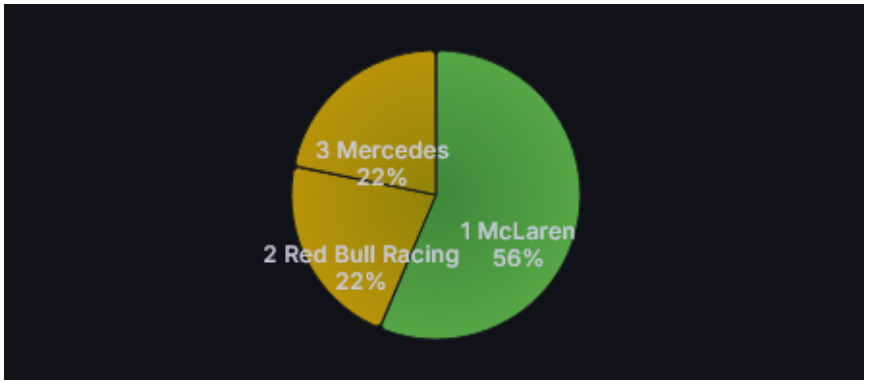
   Fichero: /var/lib/grafana/csv/top3_pilotos.csv



4. Top 3 Escuderías del Torneo (queso)

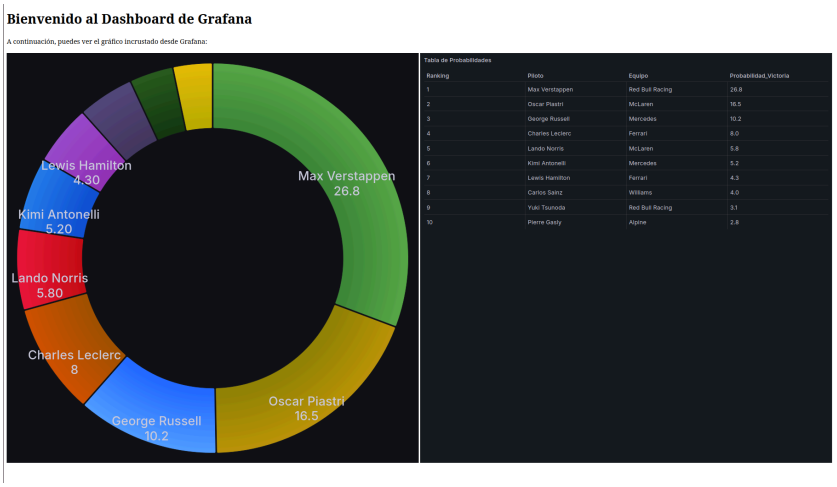
   Muestra las tres escuderías con la predicción de fin de temporada.

   Fichero: /var/lib/grafana/csv/top3_escuderias.csv



4.7 Configuración Web (apache)

La primera versión de nuestra página web fue bastante simple, con un diseño básico para verificar que lo esencial funcionaba. Solo contaba con un pequeño texto y los gráficos de para la predicción de la próxima carrera.



Para la versión final, hemos utilizado Apache como servidor web y hemos organizado los ficheros de manera estructurada para mejorar la gestión del proyecto. Los principales ficheros son:

```
usuario@CurvaIV:~/CurvaIV/datos$ ls /var/www/html/
css  ganadores.html  img  index.html  js  proxima-carrera.html  resultados.html  vds
```

La página web está compuesta por varios archivos principales: index.html para la página de inicio, ganadores.html dedicada a los ganadores del torneo, proxima-carrera.html con la información sobre la próxima carrera, y resultados.html donde se detalla el análisis de los resultados y las predicciones. Además, se organiza en carpetas específicas: css para los archivos de diseño y estilo visual, img que contiene las imágenes utilizadas en la web, js para los archivos JavaScript que gestionan la funcionalidad, y vds que guardar los videos relacionados con el contenido de la página.

La web cuenta con varias características clave:

1. Barra de navegación: Inspirada en varias webs de Fórmula 1, con un diseño sencillo pero funcional. Al hacer clic en el logo, se redirige al inicio de la página.

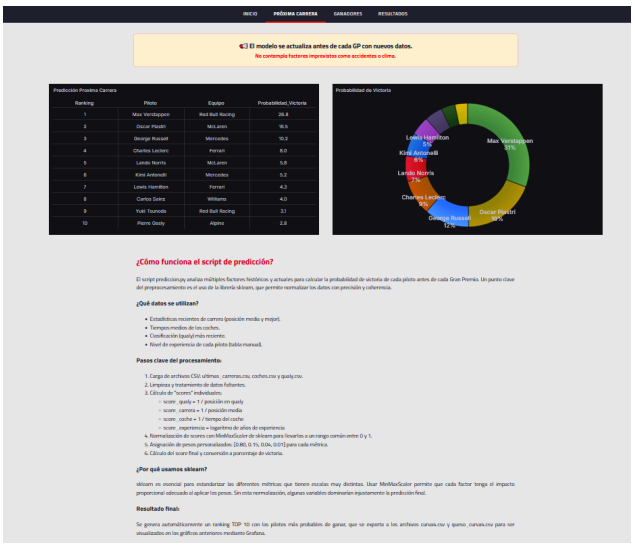
2. Pie de página: Contiene información sobre los creadores del proyecto, un enlace a la documentación y al github, la fecha de inicio del proyecto y el copyright correspondiente.

Secciones de la Página:

1. Página de Inicio:
    - Introducción a CurvaIV, tanto en texto como en video, explicando el propósito y los objetivos del proyecto.

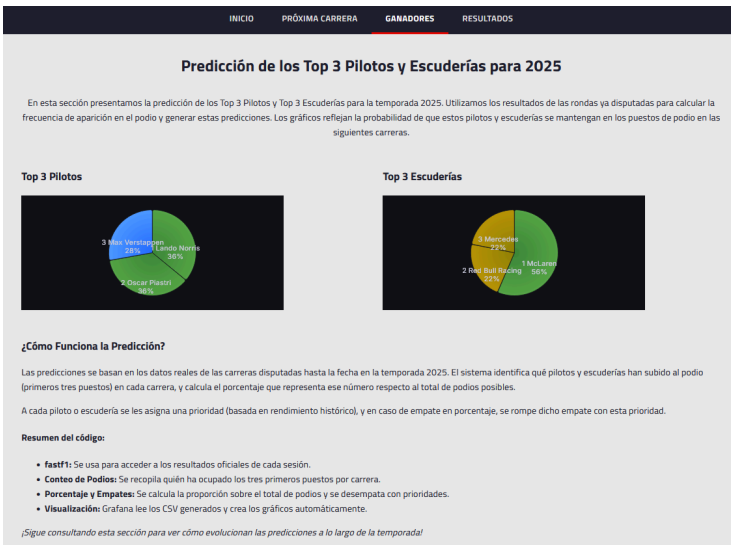
2. Próxima Carrera:

   - Explicación sobre cómo funciona la predicción de la próxima carrera, los datos que usamos, el código detrás del sistema, y los dos gráficos relevantes (tabla y gráfico de tipo "queso").



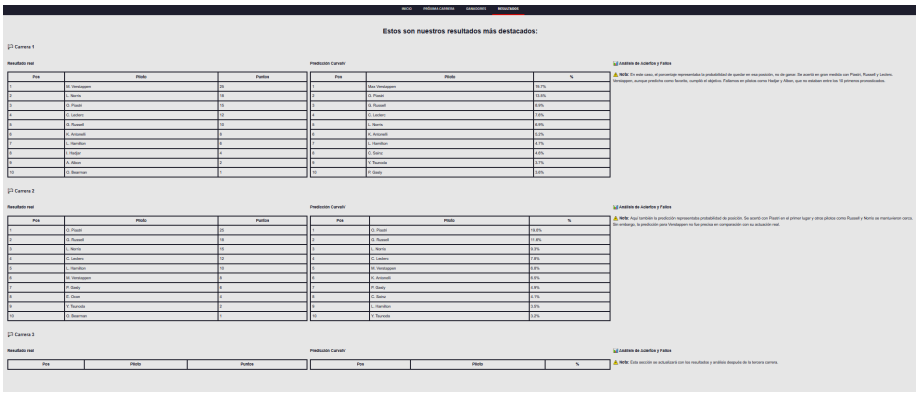
3. Ganadores:

   - Descripción de cómo se predicen los ganadores del torneo, la base de datos utilizada, y los dos gráficos de tipo "queso", uno para los top 3 pilotos y otro para las escuderías.



4. Resultados:

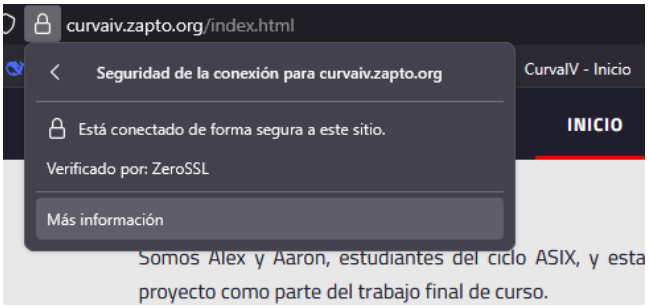
   - Presentación de las predicciones de los resultados para tener un seguimiento y ver que podemos añadir o mejorar , con un análisis detallado de los aciertos y errores.



4.7.1 Configuración seguridad web

SSL → Generamos un certificado SSL público mediante ZeroSSL para asegurar las conexiones a nuestro servidor. Este certificado lo configuramos tanto en Apache como en Grafana, permitiendo que la comunicación se realice mediante HTTPS y evitando advertencias de seguridad en los navegadores, añadiendo un redireccion a de http a https.

NO-IP → Usamos No-IP para obtener un dominio que apunta a la IP pública de nuestro servidor, incluso cuando esta cambia. Configuramos el router para redirigir los puertos 80 y 443 hacia la IP local del servidor, garantizando que el dominio de No-IP sea accesible desde Internet. Así, junto con el certificado SSL, ofrecemos acceso seguro y fiable a la web y a los dashboards de Grafana.



Cabe mencionar que hemos implementado un sistema de copia de seguridad utilizando cron, con tareas programadas que permiten hacer backups automáticos desde otro ordenador de la red. Esta funcionalidad se detalla más a fondo en el apartado "4.9 Configuración Backup" que forma parte de la seguridad.

4.8 Configuración de la automatización

Al iniciar la máquina, se ejecuta automáticamente un script que calcula las predicciones principales, asegurando que el sistema esté listo desde el arranque.

Los viernes por la noche se ejecuta el script encargado de recopilar los datos de las carreras, para tener la información actualizada antes del fin de semana.

Los sábados por la noche se lanza script de la qualy que recoge los datos de la sesión de clasificación, lo que permite incorporar esos resultados en las predicciones.

Finalmente, cada lunes a medianoche, se vuelve a ejecutar el script que calcula las predicciones del top 3, para mantener actualizados los datos con el inicio de la semana.

Con esta configuración, el sistema se mantiene actualizado y funcionando de manera automática, sin necesidad de intervención manual.

```
# ------------ EJECUCIÓN AL INICIAR LA MÁQUINA ------------
@reboot /usr/bin/python3 /home/usuario/CurvaIV/datos/predic_torneo/top3.py

# Ejecutar script_carreras.py los viernes por la noche a las 23:59
59 23 * * 5 /usr/bin/python3 /home/usuario/CurvaIV/datos/script_carreras.py

# Ejecutar script_qualy.py los sábados por la noche a las 23:59
59 23 * * 6 /usr/bin/python3 /home/usuario/CurvaIV/datos/script_qualy.py

# ------------ EJECUCIÓN CADA LUNES ------------
# Ejecuta el script los lunes a las 00:00
0 0 * * 1 /usr/bin/python3 /home/usuario/CurvaIV/datos/predic_torneo/top3.py
```

4.9 Configuración Backup

Hemos usado un ervidor externo limpio donde alojar copias de seguridad semanales.

El primer paso fue configurar la autenticación por clave SSH entre nuestro servidor principal y el servidor de backups, para que las transferencias se puedan hacer automáticamente sin pedir contraseña cada vez.

No solo hacemos backup de los datos principales del proyecto, sino que también guardamos todo lo relacionado con la web y con Grafana, asegurando así una copia completa del sistema y sus configuraciones.

Las copias incluyen:

- Los datos y scripts del proyecto ubicados en la carpeta principal.
- La base de datos y configuraciones internas de Grafana.
- Archivos de configuración específicos de Grafana.
- Los archivos del sitio web que mostramos a los usuarios.
- La configuración del servidor Apache que gestiona la web.

De esta forma, cada lunes a las 3 de la madrugada se sincronizan automáticamente todos estos datos hacia el servidor de backup. Esto asegura que en caso de fallo, pérdida o borrado accidental, podamos restaurar rápidamente la información y mantener el proyecto funcionando.

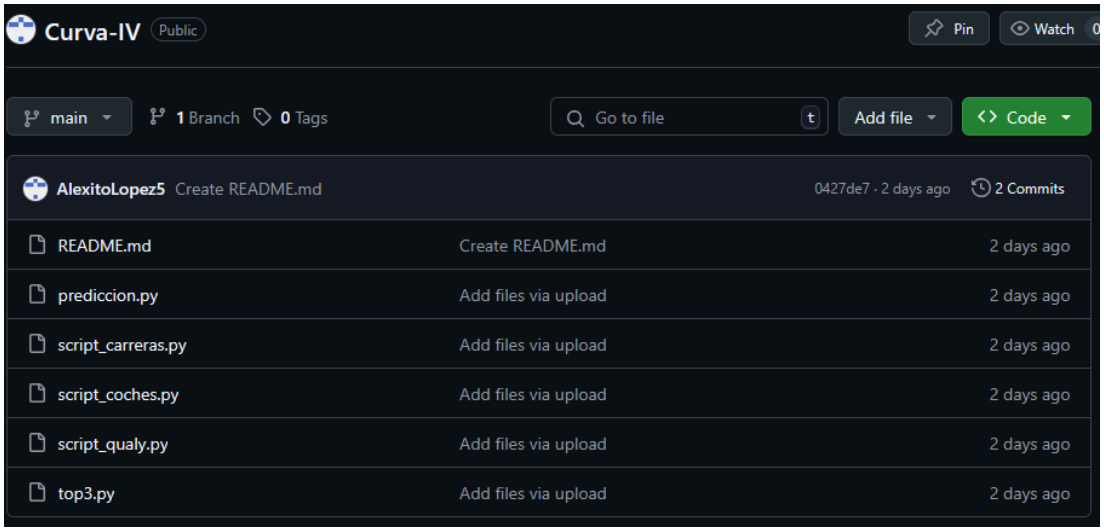
```
# ------------ BACKUP SEMANAL (CADA LUNES 3 AM) ------------
0 3 * * 1 /usr/bin/rsync -avz -e "ssh -i /root/.ssh/id_rsa" /home/usuario/CurvaIV/ usuario@192.168.1.53:/home/usuario/CurvaIV/ >> /root/
5 3 * * 1 /usr/bin/rsync -avz -e "ssh -i /root/.ssh/id_rsa" /var/lib/grafana/ usuario@192.168.1.53:/home/usuario/backups/grafana/lib/ >>
10 3 * * 1 /usr/bin/rsync -avz -e "ssh -i /root/.ssh/id_rsa" /etc/grafana/ usuario@192.168.1.53:/home/usuario/backups/grafana/etc/ >> /r
15 3 * * 1 /usr/bin/rsync -avz -e "ssh -i /root/.ssh/id_rsa" /var/www/html/ usuario@192.168.1.53:/home/usuario/backups/web/apache/html/
20 3 * * 1 /usr/bin/rsync -avz -e "ssh -i /root/.ssh/id_rsa" /etc/apache2/ usuario@192.168.1.53:/home/usuario/backups/web/apache/ >> /ro
```

```
usuario@Backup:~$ ls backups/
grafana  web
usuario@Backup:~$ ls CurvaIV/datos/
cache_f1       resultados          script_coches.py
predic_torneo  script_carreras.py  script_qualy.py
```

4.10 Subir proyecto a la nube

Como nuestro proyecto es de código abierto, decidimos subir la parte de predicción a GitHub para que cualquier persona pueda acceder al código, usarlo, modificarlo y mejorarlo si quiere. Esto ayuda a que más gente pueda aprovechar nuestro trabajo y también mejorarlo si es necesario.

El repositorio está disponible en:
https://github.com/AlexitoLopez5/Curva-IV



Además, junto con el código, incluimos un manual que explica paso a paso cómo instalar todo lo necesario, cómo usar los scripts para obtener los datos y hacer las predicciones. Así, cualquier persona interesada puede seguir las instrucciones y usar el sistema sin problemas.

El proyecto tiene licencia libre, por lo que se puede usar y modificar sin restricciones. También invitamos a que quienes quieran aportar mejoras lo hagan a través de issues o pull requests en GitHub.

5. Sistema Actual

Al principio, nuestro sistema no estaba pensado para ser una página web, sino simplemente un conjunto de códigos que hicieran la predicción de la carrera que se iba a disputar.

```
TOP 10 PREDICCIONES DE VICTORIA

            Piloto          Equipo  Probabilidad_Victoria
1      Lando Norris         McLaren                  42.3
2   Charles Leclerc         Ferrari                  15.0
3     Oscar Piastri         McLaren                   8.8
4    Lewis Hamilton         Ferrari                   5.6
5    Max Verstappen  Red Bull Racing                   4.5
6      Isack Hadjar     Racing Bulls                   3.2
7   Fernando Alonso     Aston Martin                   2.7
8   Alexander Albon         Williams                   2.3
9      Carlos Sainz         Williams                   2.2
10       Liam Lawson     Racing Bulls                   2.1

Resultados guardados en /var/lib/grafana/csv
usuario@CurvaIV:~/CurvaIV/datos/resultados$
```

Sin embargo, a medida que fuimos avanzando y mejorando el proyecto, nos dimos cuenta de que sería muy útil crear una web para mostrar los resultados de forma más accesible. Por eso decidimos usar Grafana, para que los códigos pudieran exportar las predicciones en formato CSV y subirlas a Grafana para su visualización.

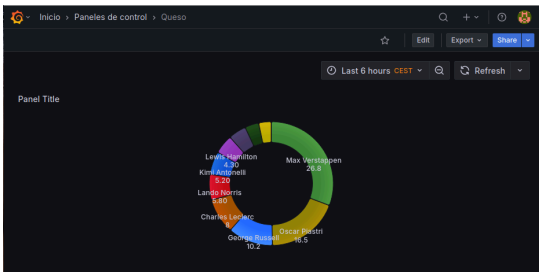
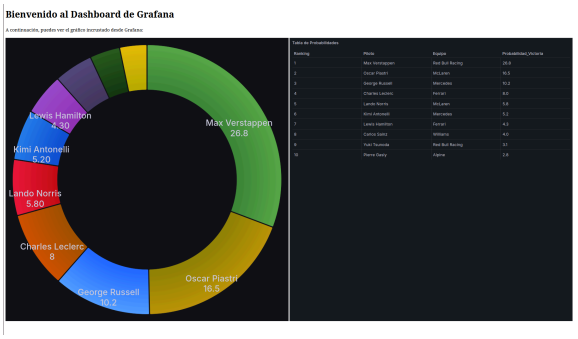


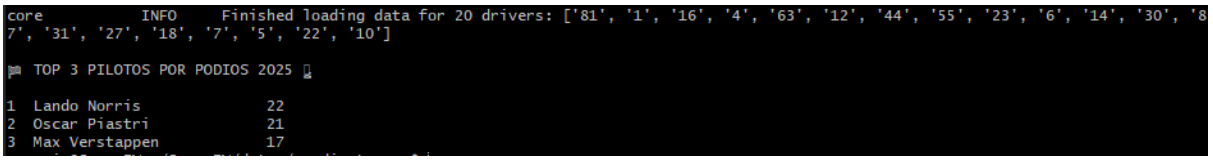
Imagen del primer queso utilizando el código de predicción csv

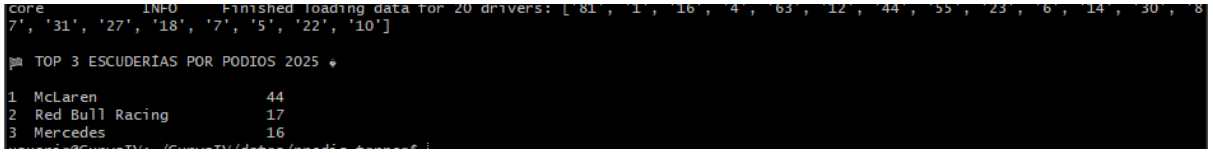
Más adelante, vimos que también podíamos montar una web local usando Apache, y lo implementamos, empezando solo con la predicción de la carrera.



Primera máquina de la web

Con el tiempo, fuimos mejorando el sistema y finalmente añadimos otro código que permite predecir el top 3 final de los pilotos y escuderías al acabar la temporada. Este resultado también lo integramos en Grafana y en la web.





Los códigos que usamos ahora no son los mismos que al inicio del proyecto, los hemos ido perfeccionando poco a poco. Gracias a la sección de “resultados” en la web, hemos podido hacer un seguimiento de cómo van las predicciones y detectar qué aspectos debíamos mejorar para obtener los mejores resultados posibles.

Al principio la web era privada y accesible solo por IP y en red local, pero luego conseguimos configurar un dominio público con certificado SSL, haciendo que la web fuera accesible de forma segura y pública para cualquiera que quiera consultarla.

6. Conclusiones

Como conclusión general, creo que el proyecto y CurvaIV, aunque van de la mano, tienen potencial para algo más grande en el futuro. Pensamos que CurvaIV no es solo un proyecto, sino que podría llegar a tener éxito real, y estamos muy orgullosos de lo que hemos conseguido.

Al principio no teníamos mucha esperanza porque empezamos con una idea diferente: queríamos hacer algo sobre fútbol. Pero al cambiar a Fórmula 1, todo se volvió más claro y nos dio un aire fresco para seguir adelante. Mientras íbamos avanzando, las cosas empezaron a salir y también surgieron más ideas para mejorar.

Estoy sorprendido porque, aunque puede que nos hayamos quedado estancados en algunos puntos, no tiene nada que ver con la situación que tuvimos con el fútbol, donde no conseguíamos encontrar la manera correcta de avanzar. En cambio, con la Fórmula 1 y las tecnologías que rodean este deporte, todo ha fluido mucho mejor.

Por ejemplo, la API FastAPI de Fórmula 1 no nos causó problemas y las demás tecnologías que hemos usado, como Grafana, Apache, rsync, y el uso de máquinas virtuales, funcionaron muy bien. Además, hemos aprendido mucho sobre estas tecnologías y también sobre aspectos prácticos importantes, como abrir puertos en routers para que los servicios sean accesibles desde fuera de nuestra red local, lo cual fue fundamental para que el proyecto funcione correctamente.

Por todo esto, considero que el proyecto ha sido todo un éxito.

6.1 Valoración de la metodología y planificación

En este punto creo que no lo hicimos perfecto. Al principio íbamos bien, documentábamos todo, apuntábamos los cambios y las pruebas, pero a medida que avanzábamos con la tecnología, parecía que íbamos muy rápido y nos quedábamos cortos con la documentación y con apuntar todo.

Creemos que para mejorar en este aspecto sería importante que tanto Alex como Aarón estuviéramos más comunicados y planificáramos mejor quién hace cada cosa. También

deberíamos haber apuntado cada día lo que íbamos avanzando y haciendo, incluyendo los errores y fallos que surgían, para tener un mejor control y aprender de ellos.

6.2 Problemas surgidos y soluciones

Problema 1: Demasiados datos y APIs de pago en el fútbol

Al principio queríamos hacer predicciones de cada partido de LaLiga, pero nos dimos cuenta de que necesitábamos más de una API (la mayoría de pago) y que manejábamos muchísimos datos: historial de partidos, tabla de clasificación, estadísticas de jugadores, etc. Esto hacía que el sistema diera errores al predecir, porque no podía cargar toda la información a la vez.

Solución: Cambiamos al mundo de la Fórmula 1, donde hay menos datos por jornada (solo una carrera semanal) y la API FastF1 nos da todo lo necesario. Así, el sistema empezó a funcionar correctamente.

Problema 2: Dificultades trabajando con GitHub

Queríamos trabajar desde GitHub, pero no teníamos una idea clara al principio. Cada vez que uno subía algo, a veces se perdía lo que había hecho el otro, o directamente lo sobreescribíamos sin querer.

Solución: Decidimos montar el proyecto en una máquina virtual con Ubuntu Server, donde teníamos más control y nos sentíamos más cómodos trabajando.

Problema 3: CSV en Grafana

Aunque habíamos usado Grafana antes, no sabíamos cómo mostrar datos desde archivos CSV. No teníamos experiencia con eso y al principio no aparecía nada en los dashboards.

Solución: Investigamos por nuestra cuenta, vimos tutoriales y encontramos el plugin de Marcus Olsson (CSV Datasource), con el que conseguimos mostrar tablas y gráficos correctamente.

Problema 4: Crontab no ejecutaba los scripts

Intentamos automatizar tareas con crontab, pero los scripts no se ejecutaban. No entendíamos por qué.

Solución: Descubrimos que necesitaban permisos de administrador. Ejecutamos contrab con sudo y configuramos crontab con usuario root. A partir de ahí, todo funcionó. Más adelante, para el backup automático en otro equipo, tuvimos que generar una clave SSH para que no nos pidiera contraseña.

Problema 5: Poco conocimiento de Python y predicción

Al comenzar, sabíamos lo justo de Python, y menos aún de predicción de datos. Los primeros modelos que probamos eran muy básicos y poco precisos.

Solución: Investigamos bastante por nuestra cuenta, usamos foros, tutoriales y mucha prueba y error hasta conseguir predicciones fiables usando scikit-learn.

Problema 6: El SSL privado bloqueaba los gráficos

Cuando montamos la web con dominio propio, pusimos un SSL privado. Pero al acceder desde fuera, los gráficos de Grafana no se cargaban.

Solución: Creamos un certificado SSL público con ZeroSSL. Pero en el instituto no podíamos abrir puertos, así que movimos el servidor a casa de Alex y abrimos los puertos 80 y 443 en el router. Usamos No-IP para el dominio y ya se mostraba todo correctamente desde fuera.

6.3 Planes a futuro

Adquirir un dominio más profesional

Actualmente utilizamos un dominio gratuito, pero en el futuro queremos registrar uno de pago, como .com o .es. Esto aportará más profesionalidad al proyecto, mayor confianza para los usuarios, y mejor visibilidad en buscadores.

Mejorar la web con diseño responsive

Queremos rediseñar la página web para que se adapte automáticamente a todos los dispositivos (ordenadores, móviles, tablets, etc.). De esta forma, cualquier persona podrá visualizar correctamente las predicciones y gráficos desde cualquier lugar.

Implementar sistema de login

De cara al largo plazo, consideramos útil añadir un sistema de autenticación para permitir acceso personalizado. Esto serviría, por ejemplo, para que ciertos usuarios como equipos o analistas puedan acceder a secciones con datos más detallados o privados.

Automatización de la selección de carreras

Actualmente la predicción se lanza seleccionando manualmente el circuito. Uno de nuestros objetivos es que el sistema detecte automáticamente la siguiente carrera del calendario y realice la predicción sin intervención manual.

Completar una temporada entera de datos

Tener todos los datos de una temporada completa nos permitirá generar estadísticas más precisas y entrenar modelos más fiables. Esto hará que en el futuro el sistema sea mucho más robusto y automático.

Alojar el servidor en la nube

A futuro nos gustaría migrar nuestro servidor local a una plataforma en la nube como Amazon Web Services (AWS) u otra alternativa. Esto garantizaría que el sistema esté disponible las 24 horas, todos los días, sin depender de una máquina física local.

Añadir una base de datos

Queremos integrar una base de datos en el sistema para almacenar información clave como predicciones, resultados históricos, configuraciones y usuarios. Esto no solo facilitará las consultas, filtrados y análisis avanzados, sino que también será fundamental para

implementar un sistema de login, permitiendo gestionar permisos y accesos personalizados según cada usuario.

Mejoras técnicas continuas

Nos gustaría seguir mejorando el rendimiento general, desde el procesamiento de datos hasta la visualización en Grafana, optimizando tiempos y resultados. También seguir explorando nuevas herramientas que puedan integrarse al proyecto.

6.4 Impacto del proyecto

Este proyecto ha tenido un impacto importante en nuestro aprendizaje, especialmente en el manejo de Python, ya que antes teníamos conocimientos básicos y ahora hemos podido trabajar con librerías avanzadas y técnicas más complejas. Además, hemos aprendido a usar tecnologías como FastAPI, Grafana, crontab y servidores web, que nos han dado una visión más amplia del desarrollo completo de un sistema.

A nivel personal y grupal, hemos mejorado nuestra comunicación, coordinación y capacidad para resolver problemas técnicos y organizativos.

Por otro lado, CurvaIV tiene potencial para ser una herramienta útil fuera del entorno académico, especialmente para seguidores y analistas de Fórmula 1. Al ser un proyecto open source, puede seguir creciendo y ser mejorado por la comunidad.

7. Manual de instrucciones

En este apartado explicamos de manera detallada, paso a paso, como hemos construido todo el proyecto desde cero, para que cualquier persona pueda reproducirlo o entender cómo lo hemos hecho.

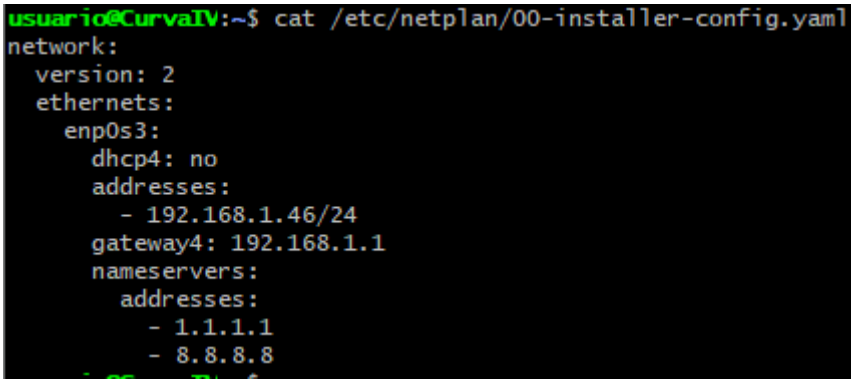
7.1 Montar servidor

Para comenzar, montamos una máquina virtual con VirtualBox usando la imagen de Ubuntu Server 22.04, que nos proporcionó el profesor Víctor Carceler. Descargamos el archivo .ova desde su OwnCloud en el siguiente enlace:

Enlace a la ISO: Ubuntu-22.04-Server-Энергия.ova

Dentro de la máquina virtual hicimos lo siguiente:

- Configuramos el adaptador de red en modo puente, para que la máquina se conecte directamente a la red local, como si fuera otro equipo más.
- Cambiamos el nombre del sistema (hostname) para identificar fácilmente el servidor en la red.
- Asignamos una IP estática usando Netplan, editando el archivo /etc/netplan/00-installer-config.yaml. Esto nos permitió tener siempre la misma IP y así no tener problemas al acceder o redireccionar el servidor.



7.2 API + requisitos

Primero actualizamos los paquetes del sistema e instalamos pip usando *sudo apt install python3-pip*. Luego, instalamos FastAPI y Uvicorn con permisos de superusuario mediante el comando *sudo pip3 install fastapi uvicorn*. Estas herramientas son necesarias para crear y ejecutar la API.

Creamos un archivo requirements.txt con todas las dependencias del proyecto, como pandas, requests, scikit-learn y fastf1. Instalamos todo con un solo comando para facilitar la configuración. Para instalar todas las librerías de golpe usamos el comando *pip3 install -r requirements.txt*.

7.3 Códigos

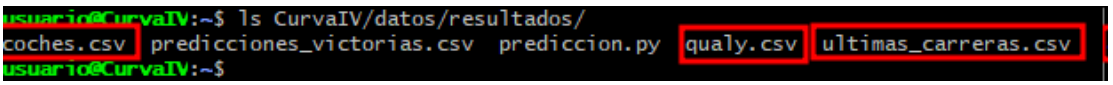
Estos son los tres códigos que usamos para recoger datos:

script_qualy.py: Este script se encarga de obtener los resultados de la última clasificación de Fórmula 1 (qualy) utilizando la librería fastf1. Al ejecutarlo, accede al calendario de carreras del año actual, localiza la última qualy disponible y descarga los datos de los pilotos, equipos y posiciones. La caché de FastF1 debe estar habilitada para que funcione correctamente. Una vez recogidos los datos, los exporta en formato CSV a la ruta /home/usuario/CurvaIV/datos/resultados/qualy.csv. Para usarlo, simplemente se descarga el código fuente del script y se guarda dentro del servidor con el nombre script_qualy.py, en la ruta deseada del proyecto. Luego, se ejecuta con el comando *python3 script_qualy.py*.

script_coche.py: Este script se encarga de analizar los datos de los entrenamientos libres 1 (FP1) de la primera ronda de la temporada 2025 de Fórmula 1. Utiliza la librería fastf1 para acceder a los datos de la sesión y calcula el mejor tiempo de vuelta registrado por cada equipo. Para ello, recorre todas las vueltas de los pilotos, identifica la más rápida de cada uno y compara los tiempos para determinar cuál fue el más rápido por equipo. El resultado es un ranking ordenado de los equipos según su mejor tiempo. Finalmente, los datos se exportan en formato CSV a la ruta /home/usuario/CurvaIV/datos/resultados/coches.csv. Este script está diseñado para ejecutarse una sola vez al inicio del proyecto, ya que utiliza datos de la

primera sesión del año. Para usarlo, simplemente se descarga el código fuente y se guarda en el servidor con el nombre script_coche.py. Luego se ejecuta con el comando *python3 script_coche.py*.

script_carreras.py: Este script obtiene los resultados de las últimas dos carreras completadas de la temporada 2025 de Fórmula 1. Para ello, define un calendario oficial con las fechas y nombres de los Grandes Premios. El script busca la próxima carrera en el calendario y selecciona las tres anteriores para analizarlas. Usa la librería fastf1 para descargar los datos de cada carrera y guarda los resultados (posición, piloto, equipo, nombre y fecha de la carrera) en un archivo CSV ubicado en /home/usuario/CurvaIV/datos/resultados/ultimas_carreras.csv. Para usarlo, se debe descargar el código fuente, guardarlo en el servidor como script_carreras.py y ejecutarlo con el comando *python3 script_carreras.py*.



Estos son los dos códigos que usamos para predecir:

prediccion.py: Este script calcula las probabilidades de victoria para los pilotos en la próxima carrera de Fórmula 1, usando datos históricos de resultados, clasificación y rendimiento del coche. Primero carga los archivos CSV con datos de carreras previas, coches y clasificación, y asigna valores a atributos como experiencia, talento y consistencia para cada piloto. Luego, limpia y normaliza estos datos usando técnicas de preprocesamiento y escalado provistas por la librería scikit-learn, aplicando pesos específicos que dependen del circuito seleccionado por el usuario para calcular un “score final” combinado de cada piloto. Finalmente, genera un ranking con las 10 predicciones más probables de victoria, muestra el resultado en pantalla y guarda los datos en archivos CSV para su visualización en Grafana. Para usarlo, descarga el código fuente, guárdalo en el servidor como prediccion.py y ejecútalo con el comando *sudo python3 prediccion.py*, es importante tener codigo en la misma carpeta que los CSV que generan los script de recopilación de datos para que funcione correctamente.

```
Usando pesos para el circuito: Montecarlo

TOP 10 PREDICCIONES DE VICTORIA

             Piloto           Equipo  Probabilidad_Victoria
1       Lando Norris          McLaren                   42.3
2    Charles Leclerc          Ferrari                   15.0
3      Oscar Piastri          McLaren                    8.8
4     Lewis Hamilton          Ferrari                    5.6
5     Max Verstappen  Red Bull Racing                    4.5
6       Isack Hadjar     Racing Bulls                    3.2
7    Fernando Alonso     Aston Martin                    2.7
8    Alexander Albon         Williams                    2.3
9       Carlos Sainz         Williams                    2.2
10       Liam Lawson     Racing Bulls                    2.1

Resultados guardados en /var/lib/grafana/csv
```

top3.py: Este script calcula y genera rankings de los pilotos y escuderías con más podios en la temporada 2025 de Fórmula 1 hasta la fecha actual. Utiliza la librería fastf1 para obtener datos de sesiones y resultados de carreras, y pandas para manipular y analizar estos datos. Para el manejo y codificación de datos categóricos, emplea herramientas de preprocesamiento como LabelEncoder de scikit-learn. Aplica reglas de prioridad para desempatar entre pilotos y equipos con resultados similares, genera los top 3 de cada categoría con sus porcentajes de podios, y guarda esta información en archivos CSV destinados a visualización en Grafana. El script también verifica permisos y existencia del directorio de salida antes de guardar los archivos. Para ejecutarlo, guarda el código como top3.py y ejecutar con: *sudo python3 top3.py*.

```
Top 3 pilotos:
            Nombre  Porcentaje
1    Oscar Piastri          25
0     Lando Norris          26
2  Max Verstappen           15

Top 3 escuderias:
             Nombre  Porcentaje
0          McLaren          52
1  Red Bull Racing          15
2          Mercedes         14
Proceso completado
```

Después, contamos con varios scripts pequeños, generalmente escritos en Bash, que nos facilitan la ejecución simultánea de múltiples programas. Estos scripts sirven principalmente para pruebas, realizar testeo y agilizar el trabajo.

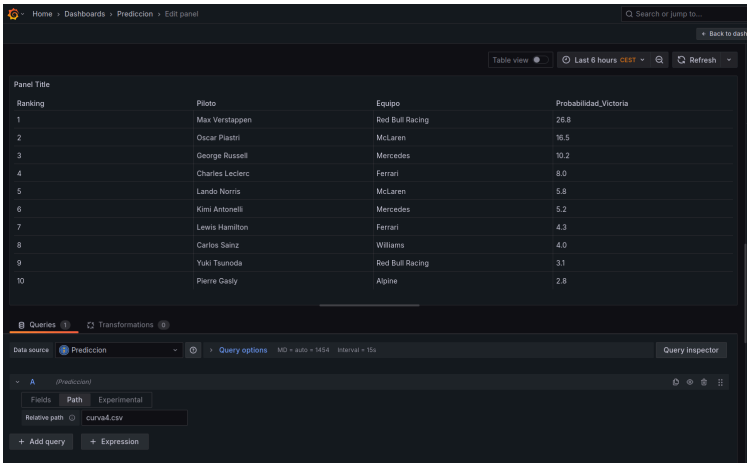
7.4 Grafana

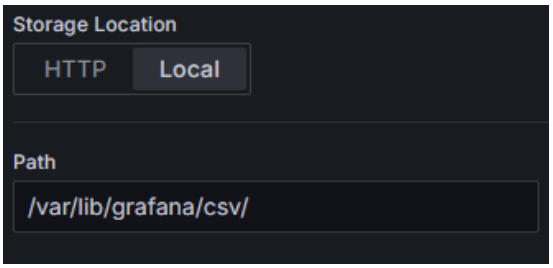
Para subir los gráficos a Grafana usaremos Marcus Olsson (CSV Datasource), donde colocaremos los archivos CSV que generan los códigos de predicción.

Instalación de Grafana: Añadimos el repositorio oficial de Grafana instalando *software-properties-common*. Después, descargamos la clave GPG de Grafana y la añadimos con *apt-key add*. Añadimos el repositorio estable de Grafana con *add-apt-repository*. Actualizamos nuevamente los paquetes con *apt update* y finalmente instalamos Grafana con *sudo apt install grafana*.

Instalación del plugin CSV Datasource de Marcus Olsson: Se instala usando el comando *grafana-cli plugins install marcusolsson-csv-datasource* y luego se reinicia el servicio de Grafana para activar el plugin.

Uso del plugin CSV Datasource: Se accede a la interfaz web de Grafana a través del navegador en la dirección https://IP:3000. Desde el menú de configuración se agrega una nueva fuente de datos seleccionando CSV Datasource. En la configuración se establece la ruta local o URL donde están guardados los archivos CSV generados por los scripts de predicción.





7.5 Apache

Apache lo usamos para la web donde añadiremos los gráficos de Grafana y también para crear una pequeña página web.

Para instalar Apache en el servidor, usando *sudo apt install apache2*. Después de la instalación, el servicio de Apache se inicia automáticamente y se puede verificar con *systemctl status apache2*.

Configuramos Apache para que sirva la web donde incrustaremos los paneles de Grafana usando iframe. También creamos una carpeta en /var/www/html donde colocaremos los archivos estáticos de la pequeña web.

```
usuario@CurvaIV:~$ ls /var/www/html/
css  ganadores.html  img  index.html  js  proxima-carrera.html  resultados.html  vds
```

```
<div style="flex: 1; display: flex; justify-content: center;">
    <iframe
        src="https://curvaiv.zapto.org:3000/d-solo/bejsjr5m4e2gwf/pr
dashboardSceneSolo"
        width="700" height="450" frameborder="0"></iframe>
</div>
```

7.6 Backup y automatización

Para mantener nuestro sistema actualizado y seguro, usaremos las herramientas rsync y cron para automatizar la ejecución de los scripts y la realización de copias de seguridad en otra máquina.

Usaremos cron para programar la ejecución automática de los scripts en horarios específicos. Al iniciar la máquina se ejecuta automáticamente el script top3.py. También programamos la ejecución de scripts de datos en horarios clave del fin de semana: script_carreras.py los viernes a las 23:59, script_qualy.py los sábados a las 23:59. Además, cada lunes a medianoche se vuelve a ejecutar top3.py para mantener los datos actualizados.

Para realizar backups semanales, configuramos tareas cron que ejecutan rsync para sincronizar de forma segura los datos y configuraciones importantes a una máquina remota. Los backups incluyen el directorio principal con los datos, las carpetas de Grafana y Apache, y sus respectivas configuraciones en /etc. Estas tareas se ejecutan cada lunes a partir de las 3 de la madrugada, con unos minutos de diferencia para no solaparse.

Para garantizar que las copias de seguridad con rsync sean seguras y automáticas, se utiliza una conexión SSH con autenticación por claves privadas, lo que permite sincronizar solo los archivos que han cambiado sin necesidad de ingresar contraseñas manualmente. Esto se logra generando un par de claves SSH en la máquina origen con el comando *ssh-keygen -t rsa -b 4096*, y copiando la clave pública al servidor remoto usando *ssh-copy-id usuario@IPSERVIDORBACKUP*, para que acepte la conexión. Luego, en los comandos de rsync se especifica la clave privada con la opción -e ssh *-i /ruta/a/la/clave* para autenticarse automáticamente. Para automatizar la ejecución de los scripts y backups, se usa cron, que permite programar tareas en horarios definidos. El archivo crontab se edita con *crontab -e*.

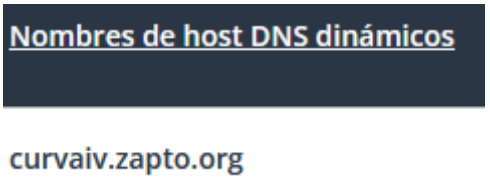
Configuración del contrab:

```
# ------------ EJECUCIÓN AL INICIAR LA MÁQUINA ------------
@reboot /usr/bin/python3 /home/usuario/CurvaIV/datos/predic_torneo/top3.py

# Ejecutar script_carreras.py los viernes por la noche a las 23:59
59 23 * * 5 /usr/bin/python3 /home/usuario/CurvaIV/datos/script_carreras.py

# Ejecutar script_qualy.py los sábados por la noche a las 23:59
59 23 * * 6 /usr/bin/python3 /home/usuario/CurvaIV/datos/script_qualy.py

# ------------ EJECUCIÓN CADA LUNES ------------
# Ejecuta el script los lunes a las 00:00
0 0 * * 1 /usr/bin/python3 /home/usuario/CurvaIV/datos/predic_torneo/top3.py

# ------------ BACKUP SEMANAL (CADA LUNES 3 AM) ------------
0 3 * * 1 /usr/bin/rsync -avz -e "ssh -i /root/.ssh/id_rsa" /home/usuario/CurvaIV/ usuario@192.168.1.53:/home/usuario/CurvaIV/ >>
/root/logs/backup_curvaiv.log 2>&1
5 3 * * 1 /usr/bin/rsync -avz -e "ssh -i /root/.ssh/id_rsa" /var/lib/grafana/ usuario@192.168.1.53:/home/usuario/backups/grafana/lib/ >>
```

```
/root/logs/backup_grafana_lib.log 2>&1
10 3 * * 1 /usr/bin/rsync -avz -e "ssh -i /root/.ssh/id_rsa" /etc/grafana/ usuario@192.168.1.53:/home/usuario/backups/grafana/etc/ >>
/root/logs/backup_grafana_etc.log 2>&1
15 3 * * 1 /usr/bin/rsync -avz -e "ssh -i /root/.ssh/id_rsa" /var/www/html/ usuario@192.168.1.53:/home/usuario/backups/web/apache/html/ >>
/root/logs/backup_web_html.log 2>&1
20 3 * * 1 /usr/bin/rsync -avz -e "ssh -i /root/.ssh/id_rsa" /etc/apache2/ usuario@192.168.1.53:/home/usuario/backups/web/apache/ >>
/root/logs/backup_web_apache.log 2>&1
```

7.7 Dominio + SSL Público

Para hacer que nuestra web sea accesible públicamente y de forma segura con HTTPS, primero nos registramos en el sitio web de NOIP y creamos un subdominio gratuito. Una vez tenemos el dominio, entramos al panel de configuración de nuestro router, accediendo a la dirección 192.168.1.1 desde el navegador. Abrimos los puertos 80 y 443 y los redirigimos a la dirección IP interna de nuestro servidor. Esto permite que cualquier conexión externa a esos puertos llegue directamente a nuestra máquina local.



Con el dominio de NOIP ya apuntando correctamente a nuestro servidor, entramos a la página de ZeroSSL y solicitamos un nuevo certificado. Introducimos nuestro subdominio y seguimos el proceso de validación, que puede ser por método HTTP subiendo un archivo al servidor web. Una vez validado, descargamos los archivos del certificado, que son certificate.crt ca_bundle.crt y private.key.

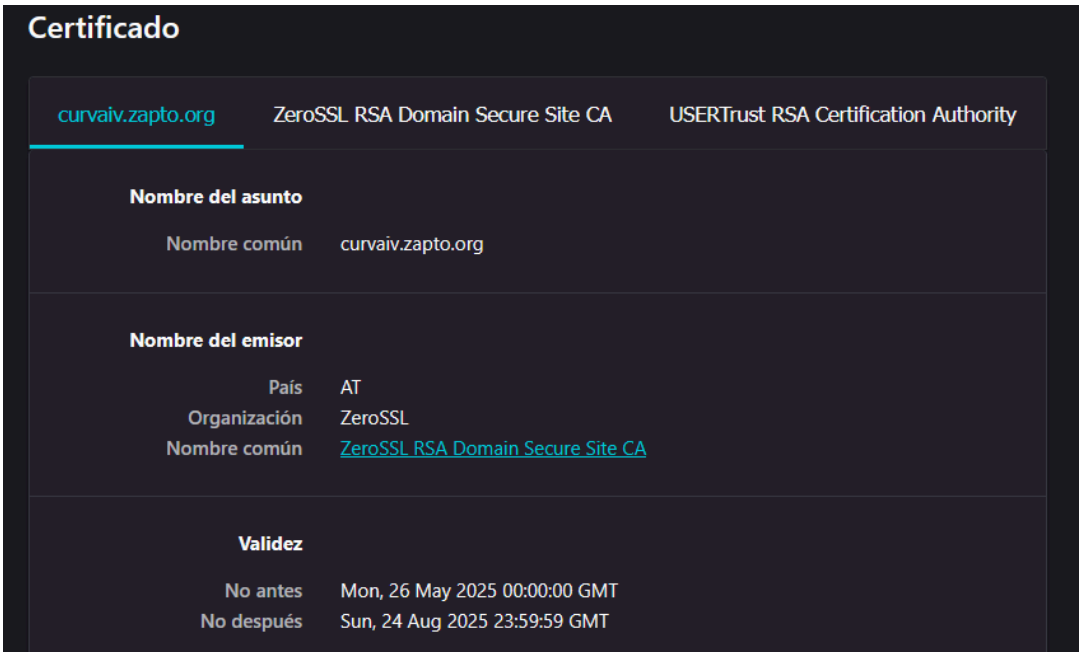
Después, copiamos estos archivos al servidor dentro de la carpeta destino *sudo cp certificate.crt /etc/apache2/ssl*. Repetimos esto para los otros dos archivos.

Luego editamos la configuración del sitio web en Apache, que normalmente se encuentra en */etc/apache2/sites-available/000-default.conf*. Dentro de este archivo añadimos un bloque para el puerto 443 donde indicamos el dominio, la ruta al contenido de la web y las rutas de los archivos del certificado. Activamos el soporte SSL en Apache con el comando *sudo a2enmod ssl* y activamos el sitio seguro con *sudo a2ensite default-ssl*. Reiniciamos el servicio de Apache usando *sudo systemctl restart apache2* para aplicar todos los cambios.

También configuramos una redirección automática de HTTP a HTTPS añadiendo una regla en el bloque que escucha el puerto 80 que diga que redirija todo hacia la dirección segura con HTTPS. Así cualquier visitante es enviado automáticamente al sitio cifrado sin tener que escribir manualmente https en el navegador.

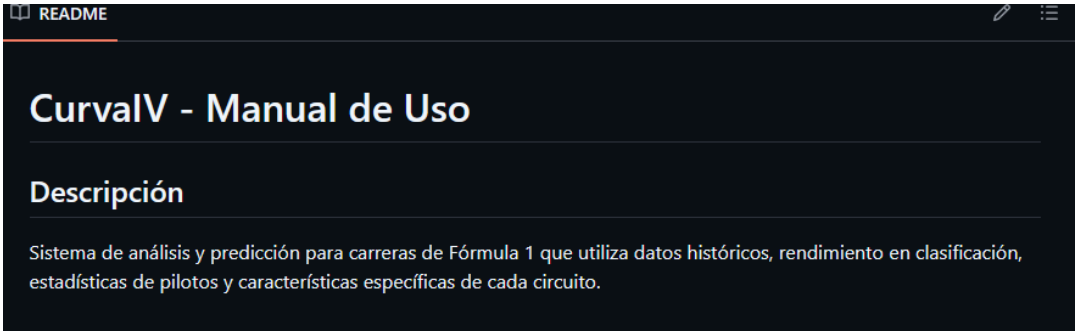
En el caso de Grafana también lo configuramos para que funcione con HTTPS. Para ello abrimos el archivo de configuración que se encuentra en */etc/grafana/grafana.ini* y buscamos la sección server. Allí cambiamos la opción protocol para que diga https y luego indicamos la ruta al archivo certificate.crt y al archivo private.key. Guardamos el archivo y reiniciamos Grafana con el comando *sudo systemctl restart grafana-server*.

Con todo esto ya tenemos tanto el sitio web como el panel de Grafana funcionando públicamente con un dominio gratuito de NOIP y protegidos con un certificado SSL gratuito de ZeroSSL. Así cualquier usuario puede acceder sin advertencias de seguridad y con la tranquilidad de que la conexión está cifrada y autenticada correctamente.



7.8 Github

En Github creamos un repositorio donde subimos los códigos de predicción y recolección de datos. Creamos un nuevo repositorio público, y luego en la máquina local usamos *git init* para inicializar, *git remote add origin* para enlazarlo, *git add .* y *git commit -m* para guardar los cambios, y finalmente *git push origin main* para subirlos. Incluimos un archivo README.md explicando qué hace el proyecto, cómo se usa y quién lo desarrolla.




CURVA

8. Bibliografía

FastAPI Documentation

Sebastián Ramírez. (2023). FastAPI - The modern, fast (high-performance), web framework for building APIs with Python. https://fastapi.tiangolo.com/

FastF1 Documentation

FastF1 Developers. (2024). FastF1 - Python package for Formula 1 data analysis. https://docs.fastf1.dev/

Grafana Documentation

Grafana Labs. (2024). Grafana - Open source analytics and monitoring platform. https://grafana.com/docs/

Scikit-Learn Documentation

Pedregosa et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research. https://scikit-learn.org/stable/

Python Official Documentation

Python Software Foundation. (2024). https://docs.python.org/3/

Crontab Manual

Linux Manual Pages. (2024). https://man7.org/linux/man-pages/man5/crontab.5.html

GitHub

GitHub, Inc. (2024). https://github.com/

Apache HTTP Server Documentation

Apache Software Foundation. (2024). https://httpd.apache.org/docs/

No-IP Official Website

No-IP LLC. (2024). https://www.noip.com/

Rsync Manual

Linux Manual Pages. (2024). https://linux.die.net/man/1/rsync

Stack Overflow

Varios usuarios. (2020-2024). Consultas y respuestas sobre Python, FastAPI, Grafana, Crontab, rsync, etc. https://stackoverflow.com/

Real Python - Tutorials
Real Python Team. (2023). FastAPI tutorials and Python data analysis guides.
https://realpython.com/tutorials/fastapi/

DigitalOcean Community Tutorials
DigitalOcean. (2023). How to deploy FastAPI with Uvicorn and systemd, Setting up Grafana with CSV files, entre otros. https://www.digitalocean.com/community/tutorials

Python Discord
Comunidad Python Discord. (2021-2024). Soporte y discusión sobre librerías Python y buenas prácticas.
https://pythondiscord.com/

Python for Data Science Handbook
Wes McKinney. (2017). Manual completo para análisis de datos con Python, incluyendo uso de pandas y scikit-learn. https://jakevdp.github.io/PythonDataScienceHandbook/

Introduction to Machine Learning with Python
Andreas C. Müller & Sarah Guido. (2016). Libro de referencia para aprender machine learning con scikit-learn.
https://www.oreilly.com/library/view/introduction-to-machine/9781449369880/

Official Scikit-learn Tutorials
Scikit-learn Developers. (2024). Tutoriales oficiales para comenzar a usar scikit-learn desde lo básico a avanzado. https://scikit-learn.org/stable/tutorial/index.html

Real Python - Python Lists and Data Structures
Real Python Team. (2023). Tutorial sobre estructuras de datos en Python, esenciales para manejar datos en machine learning. https://realpython.com/python-lists-tuples/

GitHub - Awesome Machine Learning with Python
GitHub Community. (2024). Repositorio con recopilación de recursos, librerías y tutoriales de machine learning en Python. https://github.com/josephmisiti/awesome-machine-learning

DataCamp - Supervised Learning with scikit-learn
DataCamp. (2023). Curso interactivo para aprender modelos supervisados en scikit-learn.
https://www.datacamp.com/courses/supervised-learning-with-scikit-learn

Python Crash Course
Eric Matthes. (2019). Libro para principiantes en Python que cubre conceptos básicos y buenas prácticas.
https://nostarch.com/pythoncrashcourse2e

KDnuggets - Python Machine Learning Resources
KDnuggets Editorial. (2023). Artículos y tutoriales actualizados sobre Python aplicado a machine learning.
https://www.kdnuggets.com/python-machine-learning-resources.html

YouTube Tutorials - FastAPI y Machine Learning
Varios creadores. (2020-2024). Tutoriales prácticos para desarrollo de APIs y modelos predictivos en Python.
Ejemplo: Corey Schafer, Tech With Tim. https://www.youtube.com/

Linux Hint - Tutoriales Linux
Linux Hint. (2023). Guías para uso de crontab, rsync, y administración básica de servidores.
https://linuxhint.com/

ZeroSSL Documentation
ZeroSSL Team. (2023). Obtención y configuración de certificados SSL gratuitos.
https://zerossl.com/documentation/

9. Anexo

9.1 Códigos

9.1.1 Script qualy

```
# ---------- Importamos las librerías necesarias ----------
import fastf1
import csv
import os
from datetime import datetime


# ---------- Configuración inicial ----------

# Activamos la caché de la API para usar sus datos
fastf1.Cache.enable_cache('cache_f1')


# ---------- Función para obtener la última qualy ----------

def obtener_ultima_qualy():
    """Obtiene la última sesión de clasificación disponible antes de hoy"""

    año_actual = datetime.now().year

    try:
        # Obtenemos el calendario de eventos del año actual
        calendario = fastf1.get_event_schedule(año_actual)

        # Filtramos solo los eventos cuya fecha ya pasó
        eventos_pasados = calendario[calendario['EventDate'] < datetime.now()]

        # Si hay eventos pasados, tomamos el último
        if not eventos_pasados.empty:
```

```
            ultimo_evento = eventos_pasados.iloc[-1]
            return año_actual, ultimo_evento.RoundNumber, 'Q'

    except Exception as e:
        print(f" Error al obtener el calendario {año_actual}:
{str(e)}")
        pass

    # ---------- Fallback si no se encuentra una qualy previa
----------

    try:
        # Si todo falla, cargamos el calendario de nuevo
        calendario = fastf1.get_event_schedule(año_actual)

        # Devolvemos la última ronda del calendario como emergencia
        return año_actual, calendario.iloc[-1].RoundNumber, 'Q'

    except Exception as e:
        print(f" Error al cargar eventos de fallback: {str(e)}")
        return año_actual, 1, 'Q'


# ---------- Función principal ----------

def main():
    print("\n Buscando datos de la última sesión de
clasificación...")

    # Obtenemos la información de la última qualy
    año, ronda, sesion = obtener_ultima_qualy()

    try:
        session = fastf1.get_session(año, ronda, sesion)
        session.load()

        # Configuramos la ruta de salida
        output_dir = os.path.expanduser(
            '/home/usuario/CurvaIV/datos/resultados')
```

```
        os.makedirs(output_dir, exist_ok=True)
        filename = os.path.join(output_dir, 'qualy.csv')

        if os.path.exists(filename):
            os.remove(filename)

        # Exportamos los resultados
        with open(filename, 'w', newline='', encoding='utf-8') as
csvfile:
            writer = csv.DictWriter(csvfile, fieldnames=[
                                    'Posición', 'Piloto', 'Equipo'])
            writer.writeheader()

            for _, driver in session.results.iterrows():
                writer.writerow({
                    'Posición': int(driver.Position),
                    'Piloto': driver.FullName,
                    'Equipo': driver.TeamName
                })

        print("\n Datos de clasificación exportados exitosamente")
        print(f" Archivo: {filename}")

    except Exception as e:
        print(f"\n Error al procesar la sesión: {str(e)}\n")
        exit()


if __name__ == "__main__":
    main()
```

9.1.2 Script coches

```
# ---------- Importamos las librerías necesarias ----------
import fastf1
import csv
import os
from datetime import datetime


# ---------- Configuración inicial ----------

# Activamos la caché de la API para usar sus datos
fastf1.Cache.enable_cache('cache_f1')


# ---------- Preparamos la ruta de salida ----------

# Definimos la ruta donde se guardará el archivo CSV
output_dir =
os.path.expanduser('/home/usuario/CurvaIV/datos/resultados')
os.makedirs(output_dir, exist_ok=True)
filename = os.path.join(output_dir, 'coches.csv')
if os.path.exists(filename):
    os.remove(filename)


# ---------- Obtenemos los datos del evento de tests ----------

# Establecemos el año y la ronda de test
año = 2025
ronda = 1

try:
    # Cargamos la sesión de entrenamientos libres 1
    session = fastf1.get_session(año, ronda, 'FP1')
    session.load()  # Carga los datos de la sesión
```

```
    # Creamos un diccionario para almacenar el mejor tiempo por
equipo
    mejores_tiempos = {}

    # Recorremos cada piloto único que participó en la sesión
    for drv in session.laps['Driver'].unique():
        # Seleccionamos la vuelta más rápida del piloto
        driver_laps = session.laps.pick_driver(drv).pick_fastest()

        # Obtenemos el equipo y el tiempo de esa vuelta
        equipo = driver_laps['Team']
        tiempo = driver_laps['LapTime'].total_seconds()

        # Verificamos si es el mejor tiempo del equipo o si es más
rápido que el anterior
        if equipo not in mejores_tiempos or tiempo <
mejores_tiempos[equipo]['tiempo']:
            mejores_tiempos[equipo] = {
                'tiempo': tiempo,
                'piloto': session.get_driver(drv)['FullName']
            }

    # ---------- Ordenamos los equipos según el mejor tiempo
----------

    # Creamos un ranking ordenado por tiempo más rápido (menor a
mayor)
    ranking = sorted(mejores_tiempos.items(), key=lambda x:
x[1]['tiempo'])

    # ---------- Exportamos los resultados a un archivo CSV
----------

    # Hacemos el archivo CSV
    with open(filename, 'w', newline='', encoding='utf-8') as
csvfile:
        fieldnames = ['Posición', 'Equipo', 'Piloto', 'Mejor Tiempo
(s)']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
```

```
        writer.writeheader()

        # Escribimos cada entrada del ranking con su posición
        for i, (equipo, datos) in enumerate(ranking, start=1):
            writer.writerow({
                'Posición': i,
                'Equipo': equipo,
                'Piloto': datos['piloto'],
                'Mejor Tiempo (s)': round(datos['tiempo'], 3)
            })

    # Confirmamos que todo se ha completado correctamente
    print(f"Coches completado")

except Exception as e:
    print(f"Error al cargar datos de test: {str(e)}")
```

9.1.3 Script carreras

```
# ---------- Importamos las librerías necesarias ----------
import fastf1
import csv
import os
from datetime import datetime


# ---------- Configuración inicial ----------

# Activamos la caché de la API para usar sus datos
fastf1.Cache.enable_cache('cache_f1')


# ---------- Definimos el calendario oficial de F1 2025 ----------

# Lista completa de carreras con sus nombres en español y fechas
CALENDARIO_2025 = [
```

```
    {"nombre": "Australia", "fecha": datetime(2025, 3, 16)},
    {"nombre": "China", "fecha": datetime(2025, 3, 23)},
    {"nombre": "Japón", "fecha": datetime(2025, 4, 6)},
    {"nombre": "Bahréin", "fecha": datetime(2025, 4, 13)},
    {"nombre": "Arabia Saudí", "fecha": datetime(2025, 4, 20)},
    {"nombre": "Miami", "fecha": datetime(2025, 5, 4)},
    {"nombre": "Emilia Romaña", "fecha": datetime(2025, 5, 18)},
    {"nombre": "Mónaco", "fecha": datetime(2025, 5, 25)},
    {"nombre": "España", "fecha": datetime(2025, 6, 1)},
    {"nombre": "Canadá", "fecha": datetime(2025, 6, 15)},
    {"nombre": "Austria", "fecha": datetime(2025, 6, 29)},
    {"nombre": "Gran Bretaña", "fecha": datetime(2025, 7, 6)},
    {"nombre": "Bélgica", "fecha": datetime(2025, 7, 27)},
    {"nombre": "Hungría", "fecha": datetime(2025, 8, 3)},
    {"nombre": "Países Bajos", "fecha": datetime(2025, 8, 31)},
    {"nombre": "Italia", "fecha": datetime(2025, 9, 7)},
    {"nombre": "Azerbaiyán", "fecha": datetime(2025, 9, 21)},
    {"nombre": "Singapur", "fecha": datetime(2025, 10, 5)},
    {"nombre": "Estados Unidos", "fecha": datetime(2025, 10, 19)},
    {"nombre": "México", "fecha": datetime(2025, 10, 26)},
    {"nombre": "São Paulo", "fecha": datetime(2025, 11, 9)},
    {"nombre": "Las Vegas", "fecha": datetime(2025, 11, 22)},
    {"nombre": "Qatar", "fecha": datetime(2025, 11, 30)},
    {"nombre": "Abu Dabi", "fecha": datetime(2025, 12, 7)}
]


# ---------- Función para obtener datos de las últimas carreras
----------

def obtener_ultimas_carreras(num_carreras=3):
    """Obtiene los datos de las últimas carreras completadas antes de
la próxima carrera"""

    # Buscamos cuál es la próxima carrera en el calendario
    hoy = datetime.now()
    carreras_pasadas = []

    proxima_idx = None
```

```
    for i, carrera in enumerate(CALENDARIO_2025):
        if carrera["fecha"] > hoy:
            proxima_idx = i
            break

    # Determinamos qué carreras debemos analizar
    if proxima_idx is None:
        carreras_a_buscar = CALENDARIO_2025[-num_carreras:]
    else:
        inicio = max(0, proxima_idx - num_carreras)
        carreras_a_buscar = CALENDARIO_2025[inicio:proxima_idx]

    # Procesamos cada carrera seleccionada
    carreras_validas = []
    for carrera in reversed(carreras_a_buscar):
        try:
            session = fastf1.get_session(2025, carrera["nombre"],
'R')
            session.load(telemetry=False, weather=False)

            # Guardamos la información relevante
            carreras_validas.append({
                'nombre': carrera['nombre'],
                'fecha': carrera["fecha"],
                'session': session
            })
        except Exception as e:
            print(f" Error al cargar {carrera['nombre']}: {str(e)}")
            continue

    return carreras_validas


# ---------- Función para exportar resultados a CSV ----------

def exportar_resultados(carreras, filename):
    """Guarda los resultados de las carreras en un archivo CSV"""

    # Definimos las columnas del CSV
```

```
    with open(filename, 'w', newline='', encoding='utf-8') as
csvfile:
        writer = csv.DictWriter(csvfile, fieldnames=[
            'Posición',
            'Gran Premio',
            'Fecha',
            'Piloto',
            'Equipo'
        ])
        writer.writeheader()

        # Procesamos cada carrera
        for carrera in carreras:
            fecha_str = carrera['fecha'].strftime(
                '%d/%m/%Y')

            # Escribimos los resultados de cada piloto
            for _, driver in carrera['session'].results.iterrows():
                writer.writerow({
                    'Posición': int(driver.Position),
                    'Gran Premio': carrera['nombre'],
                    'Fecha': fecha_str,
                    'Piloto': driver.FullName,
                    'Equipo': driver.TeamName
                })

            writer.writerow({})


# ---------- Función principal ----------

def main():
    print("\n🔍 Buscando resultados de las últimas carreras...")

    # Obtenemos datos de las últimas 3 carreras
    ultimas_carreras = obtener_ultimas_carreras(3)

    if not ultimas_carreras:
        print(" No se encontraron datos de carreras recientes")
```

```
        return

    # Configuramos la ruta de salida
    output_dir =
os.path.expanduser('/home/usuario/CurvaIV/datos/resultados')

    os.makedirs(output_dir, exist_ok=True)
    filename = os.path.join(output_dir, 'ultimas_carreras.csv')

    if os.path.exists(filename):
        os.remove(filename)

    # Exportamos los nuevos resultados
    exportar_resultados(ultimas_carreras, filename)

    print("\n Resultados exportados exitosamente")
    print(f" Archivo: {filename}")


if __name__ == "__main__":
    main()
```

9.1.4 Script predicción

```
# ---------- Importamos las librerías necesarias para el análisis de
datos ----------
import pandas as pd
import numpy as np
from sklearn.preprocessing import RobustScaler, MinMaxScaler
from sklearn.utils.validation import check_array
from scipy import stats


# ---------- Definimos los pesos para cada circuito ----------

#               [score_qualy, score_carrera, score_coche,
```

```
score_experiencia, score_habilidad]
PESOS_POR_CIRCUITO = {

    # Circuitos donde la qualy es lo más importante
    "Bakú":                          [0.40, 0.10, 0.25, 0.05, 0.20],
    "Hungaroring":                    [0.35, 0.15, 0.25, 0.05, 0.20],
    "Jeddah":                        [0.35, 0.15, 0.25, 0.05, 0.20],
    "Marina Bay":                     [0.38, 0.12, 0.25, 0.05, 0.20],
    "Montecarlo":                     [0.80, 0.05, 0.10, 0.02, 0.03],

    # Circuitos donde el rendimiento en carrera es más importante
    "Interlagos":                     [0.30, 0.20, 0.25, 0.05, 0.20],
    "Red Bull Ring":                  [0.30, 0.20, 0.25, 0.05, 0.20],
    "Silverstone":                    [0.28, 0.22, 0.25, 0.05, 0.20],
    "Spa-Francorchamps":              [0.25, 0.25, 0.25, 0.05, 0.20],

    # Circuitos estándar con equilibrio entre qualy y carrera
    "Barcelona-Catalunya":            [0.33, 0.17, 0.25, 0.05, 0.20],
    "Circuit of the Americas (COTA)": [0.32, 0.18, 0.25, 0.05, 0.20],
    "Imola":                         [0.35, 0.15, 0.25, 0.05, 0.20],
    "Las Vegas Street Circuit":      [0.38, 0.12, 0.25, 0.05, 0.20],
    "Lusail":                        [0.30, 0.20, 0.25, 0.05, 0.20],
    "Melbourne":                     [0.32, 0.18, 0.25, 0.05, 0.20],
    "Miami":                         [0.35, 0.15, 0.25, 0.05, 0.20],
    "Monza":                         [0.30, 0.20, 0.25, 0.05, 0.20],
    "Montreal":                      [0.35, 0.15, 0.25, 0.05, 0.20],
    "Shanghái":                      [0.33, 0.17, 0.25, 0.05, 0.20],
    "Suzuka":                        [0.35, 0.15, 0.25, 0.05, 0.20],
    "Yas Marina":                    [0.32, 0.18, 0.25, 0.05, 0.20],
    "Zandvoort":                     [0.35, 0.15, 0.25, 0.05, 0.20],
    "Hermanos Rodríguez":            [0.32, 0.18, 0.25, 0.05, 0.20],

    # Por defecto
    "default":                       [0.30, 0.20, 0.25, 0.05, 0.20]
}


# ---------- Función para cargar los datos desde los archivos CSV
----------
```

```
def cargar_datos():

    # Leemos los CSV con datos
    df_carreras =
pd.read_csv("ultimas_carreras.csv").dropna(how='all')
    df_coches = pd.read_csv("coches.csv")
    df_qualy = pd.read_csv("qualy.csv")

    # Asignamos un valor a la experiencia, talento y cosistencia a
cada piloto
    pilotos = ['M. Verstappen', 'L. Norris', 'O. Piastri', 'C.
Leclerc', 'G. Russell',
               'K. Antonelli', 'L. Hamilton', 'I. Hadjar', 'A.
Albon', 'O. Bearman',
               'F. Alonso', 'Y. Tsunoda', 'P. Gasly', 'C. Sainz Jr.',
'F. Colapinto',
               'N. Hulkenberg', 'L. Lawson', 'E. Ocon', 'G.
Bortoleto', 'L. Stroll']
    experiencia = [16, 5, 2, 6, 5, 0, 17, 0,
                   5, 1, 20, 4, 7, 9, 0, 12, 1, 7, 0, 7]
    talento = [20, 15, 12, 19, 18, 10, 14, 9, 16,
               11, 20, 13, 14, 17, 8, 12, 9, 14, 7, 10]
    consistencia = [20, 12, 9, 16, 17, 8, 20, 7,
                    15, 9, 18, 10, 13, 14, 6, 11, 8, 13, 5, 9]

    # Creamos un DataFrame con los datos de los pilotos
    df_pilotos = pd.DataFrame({
        'Piloto': pilotos,
        'Experiencia': experiencia,
        'Talento': talento,
        'Consistencia': consistencia
    })

    # Calculamos estadísticas de las carreras
    stats_carreras = df_carreras.groupby('Piloto')['Posición'].agg(
        media_posicion='mean',
        mejor_posicion='min',
        desviacion_posicion='std',
```

```
        carreras_completadas='count'
    ).reset_index()

    # Combinamos todos los datos en un solo DataFrame
    df = df_qualy.merge(
        stats_carreras,
        on='Piloto', how='left'
    ).merge(
        df_coches[['Piloto', 'Mejor Tiempo (s)']],
        on='Piloto', how='left'
    ).merge(
        df_pilotos,
        on='Piloto', how='left'
    )

    return df


# ---------- Función para limpiar y preparar los datos ----------

def preprocesar(df):
    df = df.copy()

    # Rellenamos valores faltantes con valores por defecto
    df['Experiencia'] = df['Experiencia'].fillna(0).astype(int)
    df['Talento'] = df['Talento'].fillna(10).astype(int)
    df['Consistencia'] = df['Consistencia'].fillna(10).astype(int)

    # Rellenamos estadísticas de carrera con el peor caso posible
    max_pos = df['Posición'].max()
    for col in ['media_posicion', 'mejor_posicion']:
        df[col] = df[col].fillna(max_pos)

    # Rellenamos la desviación estándar con la desviación general si
falta
    if 'desviacion_posicion' in df.columns:
        df['desviacion_posicion'] = df['desviacion_posicion'].fillna(
            df['Posición'].std())
```

```
    # Procesamos los tiempos de vuelta del coche
    df['Mejor Tiempo (s)'] = pd.to_numeric(
        df['Mejor Tiempo (s)'], errors='coerce')
    df['Mejor Tiempo (s)'] = df['Mejor Tiempo (s)'].fillna(
        df['Mejor Tiempo (s)'].median())

    return df


# ---------- Función para calcular los scores de cada piloto
----------

def calcular_scores(df, circuito="default"):

    # Calculamos el score de qualy
    df['score_qualy'] = 1 / df['Posición']

    # Calculamos el score de carrera considerando posición media y
consistencia
    df['score_carrera'] = (1 / df['media_posicion']) * \
        (1 / (1 + df.get('desviacion_posicion', 0)))

    # Calculamos el score del coche basado en el mejor tiempo
    df['score_coche'] = 1 / (df['Mejor Tiempo (s)'] ** 0.9)

    # Calculamos el score de experiencia
    df['score_experiencia'] = np.sqrt(df['Experiencia'] + 1)

    # Calculamos el score de habilidad combinando talento y
consistencia
    df['score_habilidad'] = (df['Talento'] * 0.8 + df['Consistencia']
* 0.2)

    # Normalizamos los scores para que sean comparables
    features = ['score_qualy', 'score_carrera',
                'score_coche', 'score_experiencia',
'score_habilidad']
    features = [f for f in features if f in df.columns]
```

```
    # Aplicamos RobustScaler para reducir el efecto de outliers
    scaler = RobustScaler()
    scores_norm = scaler.fit_transform(df[features])

    # Aplicamos MinMaxScaler para llevar todo a escala 0-1
    minmax = MinMaxScaler()
    scores_norm = minmax.fit_transform(scores_norm)

    # Obtenemos los pesos específicos para este circuito
    pesos = np.array(PESOS_POR_CIRCUITO.get(
        circuito, PESOS_POR_CIRCUITO["default"]))

    # Aseguramos que coincida con las features disponibles
    pesos = pesos[:len(features)]

    # Calculamos el score final combinando todos los scores con los
pesos
    df['score_final'] = np.dot(scores_norm, pesos)

    return df


# ---------- Función para generar los resultados finales con
probabilidades ----------

def generar_resultados(df):
    # Calculamos las probabilidades de victoria con un suavizado
    df['Probabilidad_Victoria'] = (np.power(df['score_final'], 1.5) /
                                   np.power(df['score_final'],
1.5).sum() * 100).round(1)

    # Creamos el resultado final ordenado y mostramos solo el top 10
    resultado = (df[['Piloto', 'Equipo', 'Probabilidad_Victoria']]
                 .sort_values('Probabilidad_Victoria',
ascending=False)
                 .head(10)
                 .reset_index(drop=True))
    resultado.index += 1  # Empezamos el ranking desde 1
```

```
    return resultado

# Función para que el usuario seleccione un circuito


def seleccionar_circuito():
    print("Circuitos disponibles:")
    circuitos = sorted(PESOS_POR_CIRCUITO.keys())

    # Mostramos todos los circuitos
    for i, circuito in enumerate(circuitos, 1):
        if circuito != "default":
            print(f"{i}. {circuito}")

    # Bucle para seleccionar el circuito
    while True:
        try:
            seleccion = int(input("\nSeleccione el número del
circuito: "))
            if 1 <= seleccion <= len(circuitos)-1:
                return circuitos[seleccion-1]
            else:
                print("Por favor, ingrese un número válido.")
        except ValueError:
            print("Por favor, ingrese un número.")


# ---------- Punto de entrada principal del programa ----------

if __name__ == "__main__":
    print("Calculando probabilidades de victoria...\n")

    try:

        # Seleccionamos el circuito
        circuito = seleccionar_circuito()
        print(f"\nUsando pesos para el circuito: {circuito}\n")

        # Cargamos, procesamos y calculamos los datos
```

```
        df = cargar_datos()
        df = preprocesar(df)
        df = calcular_scores(df, circuito=circuito)
        resultado = generar_resultados(df)

        # Mostramos los resultados
        print("TOP 10 PREDICCIONES DE VICTORIA\n")
        print(resultado.to_string())

        # Guardamos los resultados en archivos CSV para Grafana
        resultado.to_csv('/var/lib/grafana/csv/curva4.csv',
                         index_label='Ranking')
        resultado[['Probabilidad_Victoria', 'Piloto']].to_csv(
            '/var/lib/grafana/csv/queso_curva4.csv', index=False)

        print("\nResultados guardados en /var/lib/grafana/csv")

    except Exception as e:
        print(f"\nError: {str(e)}")
```

9.1.5 Script top 3

```
# ---------- Importamos las librerías necesarias ----------
import os
import fastf1
import pandas as pd
from datetime import datetime
from sklearn.preprocessing import LabelEncoder

# ---------- Configuración inicial ----------

# Usa solo consultas de fastf1
fastf1.ergast.Ergast.disabled = True
year = 2025

# Carpeta para cachear datos descargados
```

```
cache_path = os.path.expanduser("~/cache_f1")
os.makedirs(cache_path, exist_ok=True)

# Activar cache en fastf1 para mejorar rendimiento
fastf1.Cache.enable_cache(cache_path)

# Carpeta donde se guardarán los CSV para Grafana
grafana_dir = "/var/lib/grafana/csv"


# ---------- Diccionarios para desempate absoluto ----------

# Prioridad para pilotos (número menor = mayor prioridad)
PRIORIDAD_PILOTOS = {
    'Max Verstappen': 1, 'Lewis Hamilton': 2, 'Fernando Alonso': 3,
    'Lando Norris': 4, 'George Russell': 5, 'Charles Leclerc': 6,
    'Carlos Sainz': 7, 'Oscar Piastri': 8, 'Sergio Pérez': 9,
    'Yuki Tsunoda': 10
}

# Prioridad para equipos (similar a pilotos)
PRIORIDAD_EQUIPOS = {
    'Red Bull Racing': 1, 'Mercedes': 2, 'Ferrari': 3,
    'McLaren': 4, 'Aston Martin': 5, 'Alpine': 6,
    'Williams': 7, 'Visa RB': 8, 'Kick Sauber': 9,
    'Haas F1 Team': 10
}


# ---------- Función para verificar permisos y existencia del
directorio de salida ----------

def verificar_directorio():
    if not os.path.exists(grafana_dir):
        os.makedirs(grafana_dir, exist_ok=True)

    test_file = os.path.join(grafana_dir, 'test.tmp')
    try:
        with open(test_file, 'w') as f:
```

```
            f.write('test')
        os.remove(test_file)
        return True
    except PermissionError:
        print(f"Error de permisos en {grafana_dir}")
        return False


# ---------- Función para obtener todas las rondas de carreras ya
disputadas hasta hoy ----------

def obtener_carreras():
    try:
        # Obtiene calendario de la temporada
        calendario = fastf1.get_event_schedule(year)
        hoy = datetime.now().date()
        return calendario[calendario['EventDate'].dt.date <=
hoy]['RoundNumber'].tolist()
    except Exception as e:
        print(f"Error obteniendo calendario: {e}")
        return []


# ---------- Función que calcula rankings de podios (pilotos o
equipos) ----------

def calcular_rankings(podios, es_piloto=True):

    # Convierte la lista de podios a DataFrame con una columna
'Nombre'
    df = pd.DataFrame(podios, columns=['Nombre'])

    # Asigna la posición en el podio basado en el índice del
DataFrame
    df['PosicionPodio'] = (df.index % 3) + 1

    # Cuenta total de podios por nombre
    total_podios = df['Nombre'].value_counts().reset_index()
    total_podios.columns = ['Nombre', 'TotalPodios']
```

```
    # Cuenta de primeros lugares
    primeros = df[df['PosicionPodio'] ==
                  1]['Nombre'].value_counts().reset_index()
    primeros.columns = ['Nombre', 'PrimerosLugares']

    # Cuenta de segundos lugares
    segundos = df[df['PosicionPodio'] ==
                  2]['Nombre'].value_counts().reset_index()
    segundos.columns = ['Nombre', 'SegundosLugares']

    # Combina todos los conteos en un solo DataFrame
    ranking = total_podios.merge(primeros, on='Nombre', how='left')
    ranking = ranking.merge(segundos, on='Nombre', how='left')

    # Rellena valores NaN con 0 (por si alguien no tiene segundos o
primeros lugares)
    ranking.fillna(0, inplace=True)

    # Convierte columnas a enteros
    ranking['TotalPodios'] = ranking['TotalPodios'].astype(int)
    ranking['PrimerosLugares'] =
ranking['PrimerosLugares'].astype(int)
    ranking['SegundosLugares'] =
ranking['SegundosLugares'].astype(int)

    # Selecciona la prioridad correspondiente para desempatar
    prioridad = PRIORIDAD_PILOTOS if es_piloto else PRIORIDAD_EQUIPOS

    # Mapea la prioridad en la columna 'Prioridad'
    ranking['Prioridad'] =
ranking['Nombre'].map(prioridad).fillna(99)

    # Ordena el ranking
    ranking = ranking.sort_values(
        by=['TotalPodios', 'PrimerosLugares', 'SegundosLugares',
'Prioridad'],
        ascending=[False, False, False, True]
    )
```

```
    return ranking
# ---------- Función principal ----------

def main():
    # Verifica que el directorio de salida esté accesible y se pueda
escribir
    if not verificar_directorio():
        return

    # Obtiene lista de rondas disputadas hasta hoy
    rondas = obtener_carreras()
    if not rondas:
        return

    podios_pilotos = []  # Lista para guardar nombres de pilotos que
subieron al podio
    podios_equipos = []   # Lista para guardar nombres de equipos en
podio

    # Recorre cada ronda para extraer los resultados
    for ronda in rondas:
        try:
            carrera = fastf1.get_session(year, ronda, 'R')
            carrera.load(telemetry=False, weather=False)
            resultados = carrera.results.sort_values(
                'Position').head(3)  # Top 3 pilotos

            # Agrega pilotos y equipos a sus listas de podios
            for _, fila in resultados.iterrows():
                podios_pilotos.append(fila['FullName'])
                podios_equipos.append(fila['TeamName'])
        except Exception as e:
            print(f"Error en ronda {ronda}: {e}")

    # Calcula los rankings ordenados de pilotos y equipos con base en
podios
    ranking_pilotos = calcular_rankings(podios_pilotos,
es_piloto=True)
```

```
    ranking_equipos = calcular_rankings(podios_equipos,
es_piloto=False)

    # Total de carreras disputadas, usado para calcular porcentaje de
podios
    rondas_totales = len(rondas)
    # Función para guardar CSV con los top 3 y sus porcentajes
    def guardar_csv(df, tipo):
        df['Porcentaje'] = (df['TotalPodios'] /
                            (rondas_totales * 3) *
100).round().astype(int)

        # Ajusta porcentajes para evitar empates
        for i in range(1, len(df)):
            if df.iloc[i]['Porcentaje'] ==
df.iloc[i-1]['Porcentaje']:
                df.at[i, 'Porcentaje'] = df.iloc[i]['Porcentaje'] - 1

        # Guarda solo columnas Nombre y Porcentaje del top 3 en CSV
en la carpeta Grafana
        df[['Nombre', 'Porcentaje']].head(3).to_csv(
            os.path.join(grafana_dir, f'top3_{tipo}.csv'),
            index=False
        )

        # Muestra por consola el top 3 generado
        print(f"\nTop 3 {tipo}:")
        print(df.head(3)[['Nombre', 'Porcentaje']])

    # Guardamos CSV para pilotos y escuderías
    guardar_csv(ranking_pilotos, 'pilotos')
    guardar_csv(ranking_equipos, 'escuderias')


if __name__ == "__main__":
    print("Calculando podios F1 2025...")
    main()
    print("Proceso completado")
```