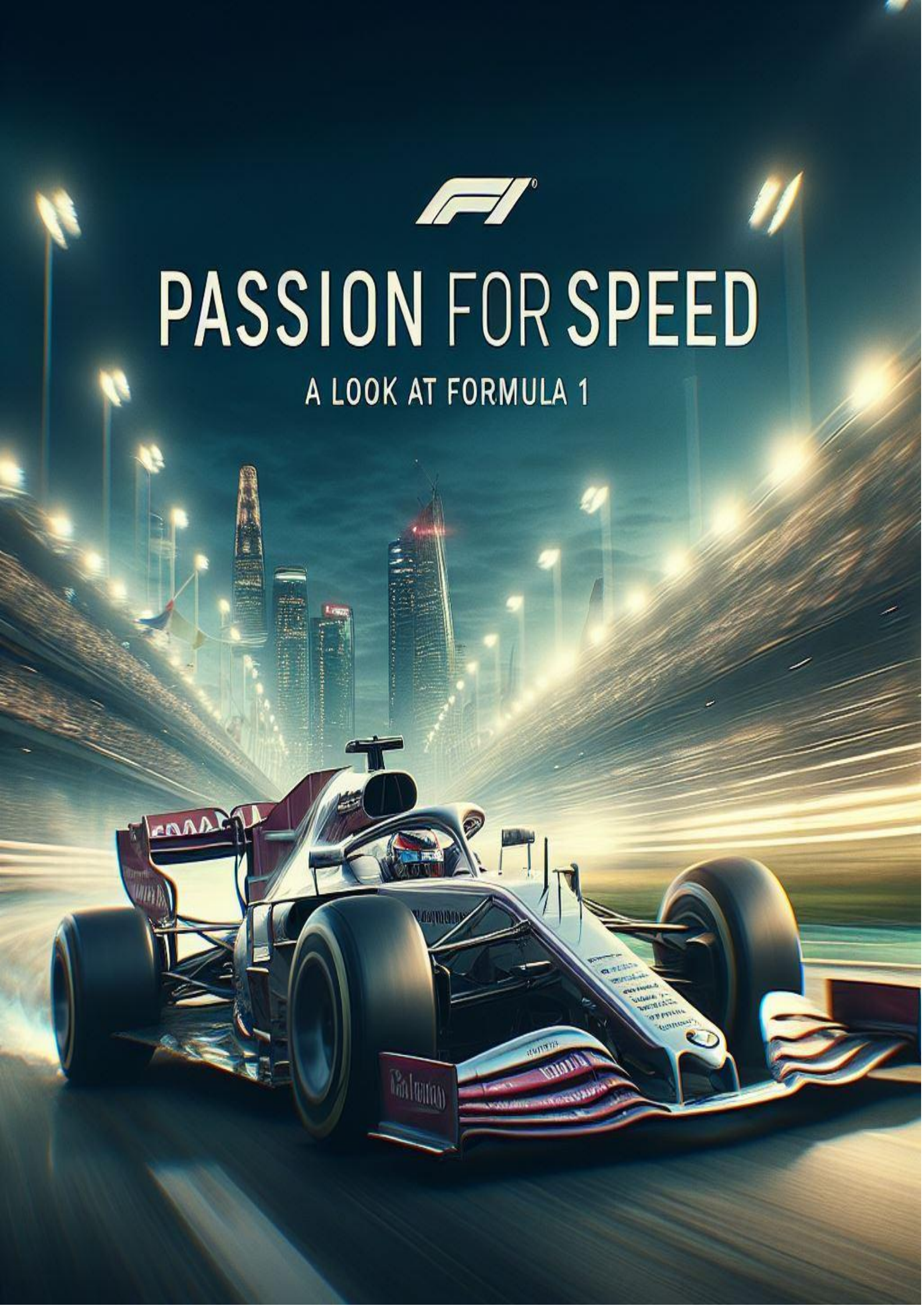

PASSION FOR SPEED
A LOOK AT FORMULA 1

Memoria proyecto

Índice:

Descripción Caso de Uso 4
Objetivos/KPIs 4
Descripción de datos 5
  Descripción del orígen de los datos 5
  Descripción de las fuentes de datos y metadatos 6
Análisis e identificación de problemas 17
Herramientas y arquitectura del DataWarehouse 18
Diseños 19
  Diseño Conceptual 19
  Diseño Lógico 20
  Diseño Físico 21
Diseño y creación del cubo Mondrian 22
Implementación de los ETLs 22
  Transformaciones 24
    Lap_deleted 27
    Lap_number 27
    Schedule 27
    Tabla_de_Hechos 28
Visualización del Cubo OLAP 32
Dashboard 33
Conclusiones Proyecto 40
  Aspectos de mejora 46
  Limitaciones del proyecto 47
  Ampliaciones futuras 47

Índice de figuras:

Diseño Conceptual 19
Diseño Lógico 20
Diseño Físico 21
Diseño Cubo OLAP 22
ETL 1 24
ETL 2 25
ETL 3 26
ETL 4 27
ETL 5 27
ETL 6 28
ETL Tabla de Hechos 1 28
ETL Tabla de Hechos 2 29
ETL Tabla de Hechos 3 29
ETL Tabla de Hechos 4 29
ETL Tabla de Hechos 5 30
ETL Tabla de Hechos 6 30
ETL Tabla de Hechos 7 31
ETL Tabla de Hechos 8 31
Cubo OLAP 32
Visualizaciones 1 33
Visualizaciones 2 34
Visualizaciones 3 34
Visualizaciones 4 35
Visualizaciones 5 36
Visualizaciones 6 36
Visualizaciones 7 37
Visualizaciones 8 38
Visualizaciones 9 38
Visualizaciones 10 39
Visualizaciones 11 40
Conclusión 1 41
Conclusión 2 41
Conclusión 3 42
Conclusión 4 42
Conclusión 5 43
Conclusión 6 43
Conclusión 7 43
Conclusión 8 44
Conclusión 9 45
Conclusión 10 45
Conclusión 11 46
Conclusión 12 46

Descripción Caso de Uso

La idea principal de este trabajo se centra en el tema de las carreras en la Fórmula1.

Actualmente, en la F1 se utilizan medidores muy potentes que almacenan una gran cantidad de información de cada gran premio.

Por lo que, la idea de este proyecto es conocer el rendimiento de un piloto y una escudería a lo largo de una temporada o dos, recabando datos de cada vuelta de cada gran premio de la temporada, para así poder explicar el resultado final del coche y del piloto, y hacer la comparativa con su compañero de equipo y con el resto de las escuderías.

Para ello, hará falta conocer los datos de cada piloto, equipo y gran premio que se dispute a lo largo de la temporada. Obteniendo información de los tiempos de cada piloto, intentando desmenuzar por sectores y obteniendo valores de su telemetría como el porcentaje de frenada del piloto en la vuelta o la velocidad punta en un sector concreto.

Objetivos/KPIs

Algunos de los objetivos y KPIs que se han planteado para este proyecto son:

→Aumentar el ritmo de carrera del equipo. Interesa conocer el ritmo de carrera de cada piloto para reflejar las diferencias de rendimiento con otros equipos e intentar reducir la diferencia con el resto de los pilotos de otras escuderías, disminuyendo el tiempo por vuelta en un 4%. Para ello, como medida se añadirá el tiempo de vuelta de cada piloto, desmenuzando en el tiempo de cada sector por vuelta.

→Asegurarse la competitividad de ambos pilotos. Para un equipo es imprescindible que ambos pilotos que pilotan el mismo coche consigan sacarle el máximo partido. Por eso, se define como KPI conseguir reducir las diferencias entre pilotos a 4 décimas por vuelta como máximo. La diferencia de tiempos con el compañero de equipo será almacenada como medida para analizar la situación de este indicador por vuelta y por sector.

→Aumentar la velocidad. Es fundamental que los coches consigan tener una gran velocidad máxima, para así conseguir adelantar con mayor seguridad en rectas. A partir de este objetivo, se ha definido como KPI conseguir aumentar la velocidad media en un 3%. Para medir esta característica, se han definido medidas de velocidades en sitios concretos del circuito, como en la recta más larga del circuito, en la línea de meta, o en el final de los sectores 1 y 2.

→ Disminuir el tiempo de frenada. En los grandes premios de Fórmula 1, los pilotos evitan frenar puesto que esto reduce el tiempo por vuelta, con lo que se debería de intentar conseguir que el piloto frenase lo mínimo a lo largo de una vuelta. Para esto, se propone conseguir disminuir el porcentaje de frenada en una vuelta un 2%, consiguiendo que el piloto sea más agresivo en la forma de conducir. Para ello, se ha añadido una medida llamada "Brake_lap", que estipula el porcentaje de frenada en una vuelta.

→ Aumentar el efecto DRS. Para adelantar, es fundamental que el efecto DRS aumente en gran medida la velocidad del vehículo en disposición de adelantar. Ante esto, se plantea como KPI conseguir que durante una vuelta en DRS, reducir en un 3% el tiempo de esa vuelta, aumentado la velocidad punto en un 5%. Para evaluar este indicador, se tratarán las medidas del porcentaje de DRS abierto en la vuelta, el tiempo por vuelta, y las diferentes medidas de velocidad como la velocidad media o la velocidad en la línea de meta.

Se destaca que para la consecución de los KPIs se ha recabado información de medidas fuera del espectro, para ampliar la información en futuros desarrollos y trabajo.

Descripción de datos

Descripción del orígen de los datos

Para este trabajo los datos se han extraído de API llamada "FastF1" que se encarga de recabar información y almacenarla. Esta API a su vez consume Información de la API llamada "Ergast API Developer". Para la realización de este trabajo se han implementado llamadas a ambas APIs para complementar ambos datos y conseguir una mayor riqueza en estos junto con una mayor información. Para ello, se ha necesitado de una investigación previa de los

diferentes métodos necesarios para obtener información y la información que estaba disponible de cada gran premio.

En cuanto a la fiabilidad de estas fuentes, FastF1 es una API ampliamente documentado y mantenida por la comunidad, en la cual se va actualizando (hasta finales de este año, que pasará a ser privada), por lo que se lleva un control de bugs y reporte de estos. En cuanto a Ergast, se planteó como un servicio web de manera experimental que proporciona datos históricos de las carreras a motor con propósitos no comerciales, puesto que es gratuita. Sus datos abarcan desde comienzos de 1950, aunque los datos de esos primeros años no son tan completos como los de los últimos años.

Una gran ventaja es que los datos son actualizados cada gran premio, por lo que ambas APIs se encuentran al día. Incluso, la API de FastF1, incluye una función de llamada a los métodos con “live_data”, lo que permite obtener información en tiempo real de la situación de la carrera o clasificación. Obteniendo los tiempos del piloto en un sector o vuelta completa instantáneamente.

Como información adicional, la API FastF1 utilizar para mostrar los datos en DataFrames y series la librería Panda, lo que hace mucho más fácil trabajar con los datos. Estos datos combinados con la librería plot permiten mostrar gráficas con scripts sencillos. En cuanto a las peticiones HTTP a la API, todas pasan por un sistema de limitación de velocidad y se almacenan en caché, opción habilitada por defecto pero que es posible de deshabilitar. Otra característica de la API es que al ejecutar el script muestran los logs sobre el progreso del tratamiento de información, al igual que los warnings. Esto se puede modificar cambiando el nivel de los logs a “Debug” o “Error”, entre otras opciones.

Descripción de las fuentes de datos y metadatos

En este caso, al ser la fuente de datos una API, se ha tomado la decisión de crear un script que realice las llamadas a la API para almacenar los datos en ficheros CSV. De esta manera, se ejecutará el script una vez para almacenar los datos, y una vez almacenados, se trabajará con independencia de la API para evitar que posibles caídas de la API puedan ralentizar el trabajo y facilitando la tarea de los ETLs.

El fichero “DownloadData.py” dentro de la carpeta del proyecto contiene el código utilizado para la realización de la tarea y dentro de la carpeta “CSV_Generados” se encuentran los ficheros recabados de esta tarea.

En total, se han generado 6 ficheros .csv para la realización de este trabajo. A continuación, se dispone a explicar en una tabla las diferentes columnas en cada de estos ficheros.

Drivers.csv:

Nombre Columna	Descripción	Tipo de dato	Ejemplo	A tener en cuenta
Broadcast_Name	Abreviatura de 3 sílabas para el apellido de cada piloto que se utiliza como identificador de estos en cada carrera.	String	ALB	En el caso de algunos pilotos, pueden incluir una letra de su nombre, como es el caso de Mick Schumacher (MSC).
Name	Nombre del piloto	String	Alexander	
Last_Name	Apellido del piloto	String	Albon	
dateOfBirth	Fecha de nacimiento de cada piloto	DateTime	1996-03-23 00:00:00	En este caso, al devolver la API un objeto DateTime, se almacenan todas las fechas con la hora, minutos y segundos a 0.
Nationality	Nacionalidad de cada piloto	String	Thai	Es importante aclarar que se pone el gentilicio de cada piloto, no el nombre del país, y está escrita en inglés.

Format.csv:

Nombre Columna	Descripción	Tipo de dato	Ejemplo	A tener en cuenta
GP_Format	Descripción del formato del fin de semana.	String	testing	Entre los distintos tipos está el fin de semana de tests, fin de semana convencional o fin de semana con carrera al sprint. Esta última aparece de 2 formas (sprint y sprint_shootout) pero el formato es prácticamente el mismo, por lo que se asumirán como uno.
Fecha_Evento	Fecha en la que se realiza el gran premio o evento.	DateTime	2023-02-25 00:00:00	Esta columna se representa por el domingo del fin de semana. De igual manera que antes, todas las horas, minutos y segundos estarán a 0.

Track_status.csv:

Nombre Columna	Descripción	Tipo de dato	Ejemplo	A tener en cuenta
idTrack_status	Identificador del estado de la pista cuando se realiza la vuelta	Int	1	En la API, se utilizan números que van asociados a un estado concreto de la pista, entre los que se distinguen seis: (1,2,4,5,6,7)
Status	Definición asociada al identificador del estado de la pista	String	AllClear	Entre los distintos estados se disciernen: (AllClear, Yellow, SCDeployed, Red, VSCDeployed, VSCEnding)

Grand_prix.csv:

Nombre Columna	Descripción	Tipo de dato	Ejemplo	A tener en cuenta
Name	Nombre completo del gran premio	String	Bahrain Grand Prix	Normalmente suele ser el nombre de la ciudad/país junto a la terminación Grand Prix
Circuit_Name	Nombre del circuito	String	Bahrain International Circuit	En este caso el nombre del circuito tiene una mayor variabilidad, pero siempre irá acompañado de la palabra “circuit” o “autódromo” en otros casos.
Country	País donde se encuentra ubicado el circuito que alberga el gran premio. También se puede representar con las iniciales del país, como es el caso de Reino Unido, Estados Unidos o Emiratos Árabes Unidos aparecen con su abreviatura (UK, USA, UAE).	String	Bahrain	Hay un caso articulo en los grandes premios en Estados Unidos porque gran premio de Las Vegas aparece el país como “United States” pero el gran premio de Miami aparece como “USA” pero es el mismo país.
Location	Ciudad o municipio concreto donde se disputa el gran premio.	String	Sakhir	
Total_distance	Distancia total aproximada en km del circuito	Float	5359.149549796158	Para calcular la distancia total se ha tenido que obtener a partir del último valor recibido de la telemetría por la API,

				por lo que normalmente será ligeramente menor que la distancia total real del circuito.
Laps	Número total de vueltas que se disputan en la carrera del gran premio.	Integer	57	Este valor no cambia ya que la organización de la fórmula 1 estima el número en base a unos parámetros. Por lo que, a no ser que se realicen grandes cambios en el circuito, este número se mantiene a lo largo del tiempo.

Team.csv:

Nombre Columna	Descripción	Tipo de dato	Ejemplo	A tener en cuenta
Name	Nombre de la escudería que representa un equipo de los 10 de una temporada de Fórmula 1.	String	Red Bull Racing	
Nationality	Nacionalidad/País de origen de la escudería	String	Austrian	
Team_id	Identificador utilizado de la escudería	String	red_bull	Este dato es innecesario para los ETLs, pero se ha necesitado para extraer la información, puesto que se ha accedido a la API de Fastf1 y Ergast.

Laps.csv:

Nombre Columna	Descripción	Tipo de dato	Ejemplo	A tener en cuenta
Driver	Abreviatura del piloto que realiza la vuelta (Igual que columna Broadcast_Name en drivers.csv	String	VER	Un mismo piloto puede salir con escuderías(Team) distintas al cambiar de equipo de una temporada a otra.
Team	Nombre del equipo del piloto que realiza la vuelta	String	Red Bull Racing	
Grand_Prix	Nombre del gran premio a disputar	String	Bahrain Grand Prix	
Lap_Number	Número de la vuelta del gran premio	Float	1.0	
Weather	Valor booleano que indica si hay lluvia en el momento en el que se realiza la vuelta o no.	Boolean	False	Indica si hay lluvia, pero puede darse el caso de que justo empiece a llover y están con neumáticos de seco, o que no llueva pero la pista esté mojada y vayan con neumáticos de lluvia.
Tyre	Tipo de neumático utilizado para realizar la vuelta	String	SOFT	Entre los diferentes tipos de neumáticos se encuentran: SOFT, MEDIUM y HARD para seco y INTERMEDIATE y WET para mojado.

Lap_deleted	Devuelve si una vuelta ha sido eliminada o no.	Boolean	False	
Deleted_Reason	En el caso de que la vuelta haya sido eliminada, se almacena el lugar donde el piloto se ha salido de la pista.	String	TRACK LIMITS AT TURN 4 LAP 43	En el caso de que la vuelta no haya sido eliminada, este espacio aparecerá en blanco, también puede darse el caso de que se pierda este valor y aparezca como “nan” en el fichero CSV.
Track_status	Estado de la pista en el momento en el que se realiza la vuelta	Int	1	A lo largo de la vuelta la pista puede pasar por varios estados, como bandera amarilla que aparece y desaparece, por lo que aparecen combinaciones como 2671 que indican los estados por los que ha pasado la pista a lo largo de la vuelta. También hay casos en los que la información obtenida no es precisa y aparece como “nan”.
Date	Fecha y hora del gran premio del cual se está obteniendo información de la vuelta	DateTime	2023-03-05 15:00:00	Este dato se utilizará posteriormente como identificador entre ficheros csv.
Lap_time	Tiempo establecido por el piloto en	Pandas Timedelta	0 days 00:01:37.974000	Al igual que en otras columnas, los datos pueden perderse en el caso de algunas vueltas, por lo que puede

	recorrer el circuito.			aparecer como “NaT”
Sector_1	Tiempo establecido por el piloto en recorrer el primer sector del circuito	Pandas Timedelta	0 days 00:00:31.342000	En algunas vueltas la API no tiene establecido el dato, por lo que aparece como “NaT”.
Sector_2	Tiempo establecido por el piloto en recorrer el segundo sector del circuito	Pandas Timedelta	0 days 00:00:42.504000	En algunas vueltas la API no tiene establecido el dato, por lo que aparece como “NaT”.
Sector_3	Tiempo establecido por el piloto en recorrer el tercer sector del circuito	Pandas Timedelta	0 days 00:00:24.128000	En algunas vueltas la API no tiene establecido el dato, por lo que aparece como “NaT”.
Speed_S1	Velocidad del piloto en la trampa de velocidad ubicada al final del primer sector del circuito en km/h.	Float	227.0	En algunas vueltas este dato no aparece en la API, por lo que se muestra como “nan”
Speed_S2	Velocidad del piloto en la trampa de velocidad ubicada en el final del segundo	Float	238.0	En algunas vueltas este dato no aparece en la API, por lo que se muestra como “nan”

	sector del circuito en km/h.			
Speed_FL	Velocidad del piloto en la línea de meta en km/h.	Float	278.0	En algunas vueltas este dato no aparece en la API, por lo que se muestra como “nan”
Speed_LS	Velocidad máxima del piloto en la recta más larga del circuito, sin contar la recta de meta, en km/h.	Float	288.0	En algunas vueltas este dato no aparece en la API, por lo que se muestra como “nan”
Avg_Speed	Media de la velocidad de la vuelta en km/h.	Float	196.5358592692828	En algunos casos, la telemetría no ha sido posible obtenerla, por lo que aparece como “Sin datos”.
Avg_RPM	Media de las revoluciones por minuto del coche a lo largo de la vuelta.	Float	9776.825439783492	En algunos casos, la telemetría no ha sido posible obtenerla, por lo que aparece como “Sin datos”.
Avg_Gear	Media de la fuerza gravitacional experimentada a lo largo de la vuelta por el piloto.	Float	5.143437077131258	En algunos casos, la telemetría no ha sido posible obtenerla, por lo que aparece como “Sin datos”.
Brake_percent	Porcentaje de la vuelta en la que el piloto ha	Float	19.485791610284167	En algunos casos, la telemetría no ha sido posible obtenerla, por lo que aparece como “Sin datos”.

	estado pulsando el pedal de freno.			
DRS_On_Percent	Porcentaje de la vuelta en la que el piloto ha tenido activado el DRS.	Float	0.0	En algunos casos, la telemetría no ha sido posible obtenerla, por lo que aparece como “Sin datos”.
Time_OnTrack	Porcentaje de la vuelta en la que el piloto ha mantenido al coche dentro de la pista.	Float	100.0	En algunos casos, la telemetría no ha sido posible obtenerla, por lo que aparece como “Sin datos”.
Pit_In	Instante en el que el piloto entra al pit/boxes para cambiar los neumáticos	Pandas Timedelta	0 days 01:25:33.830000	En el caso de que en una vuelta el piloto no entre ni salga de boxes, aparecerá esta campo como “NaT”. Además, es importante recalcar que en el caso de que entre al pit, en esa vuelta aparecerá el tiempo del instante en el que entra pero en la siguiente el instante en el que sale, no en la misma vuelta.
Pit_Out	Instante en el que el piloto sale del pit/boxes después de haber cambiado los neumáticos	Pandas Timedelta	0 days 01:25:58.127000	
Tyrelife	Número de vueltas realizadas con el neumático tras realizar la vuelta.	Float	5.0	
Track_Temperature	Temperatura de la pista al	Float	31.1	

	realizar la vuelta al circuito			

En cuanto al tamaño de los ficheros, todos los ficheros menos el fichero “laps.csv” tienen un tamaño bastante pequeño de entre 1 y 10 KB. El fichero laps.csv es el mayor tamaño, ocupando 14163 KB.

Análisis e identificación de problemas

Antes de realizar el trabajo, la idea era almacenar la información de más características de las almacenadas finalmente. Esto se debe a varios factores, como la información proporcionada por las diferentes APIs, y lo completa que está dicha información. A continuación, se enumeran diferentes problemas que se han encontrado a lo largo de este proyecto:

- En cuanto al formato del fin de semana, en el año 2022, cuando el fin de semana tenía carrera al sprint se llama “sprint_shootout”. Sin embargo, en 2023, cuando ocurre lo mismo en su lugar se ha llamado “sprint”.
- Para los datos por vuelta, hay varias donde se han perdido varios datos como los tiempos por vuelta, velocidades … En el caso de que se perdiese un sector, pero estuviesen los otros dos se podría calcular, pero si no hay ninguna información entonces es imposible sacarlo. Además, a veces esto no es culpa de la API, sino que puede ser que el coche haya sufrido un problema mecánico y se haya detenido en mitad de la vuelta o algo parecido, de ahí la falta de precisión en los datos.
- En un primer momento, se quería clasificar los circuitos en función de su diseño, es decir, si predomina la curva lenta, media o rápida o las rectas. Pero, con la información proporcionada por la API no se podía obtener directamente y hubiera sido necesario mucho trabajo para realizar una clasificación medianamente precisa.

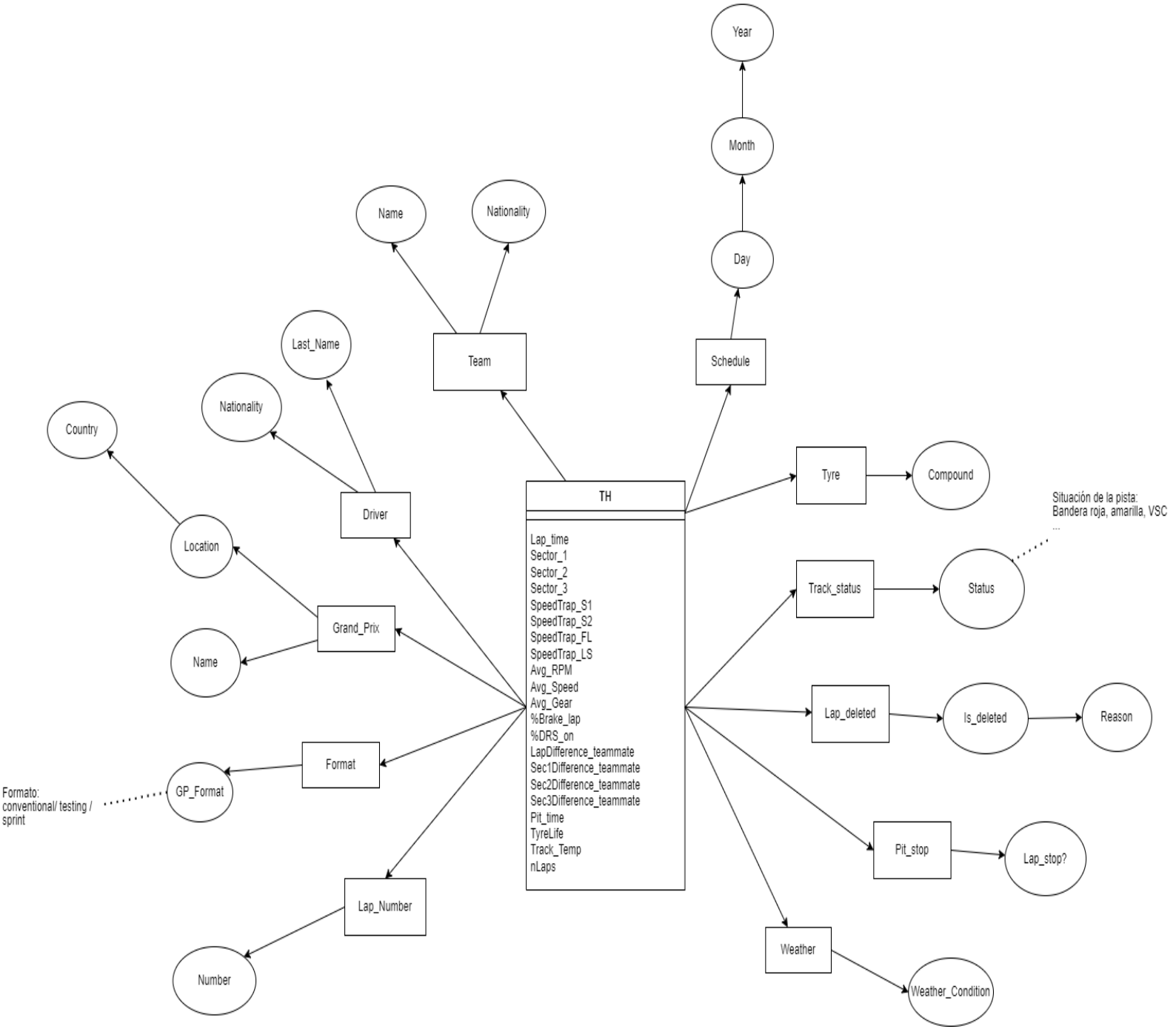
Herramientas y arquitectura del DataWarehouse

Para realizar este trabajo se utilizado las siguientes herramientas:

- ➢ Como fuente de datos, se realizaron consultas a la API de FastF1, junto con la API de Ergast. Juntando las peticiones a ambas APIs, se agrupó la información en varios ficheros CSV.
- ➢ Para la visualización de los ficheros CSV, en temas de "Debug" y cheque de datos, se ha utilizado Microsoft Excel, que permite ver las filas claramente y filtrar columnas por valores únicos.
- ➢ Posteriormente, para la realización del diseño lógico se utilizó la herramienta de Draw.io en Google Drive, juntamente con la Google Docs para la realización de la memoria presente.
- ➢ Para el diseño lógico y física, se utilizó la herramienta de MySQLWorkbench. Se trata de una herramienta visual de diseño de bases de datos que integra desarrollo de software, administración de bases de datos, diseño de bases de datos, gestión y mantenimiento para el sistema de base de datos MySQL además de para almacenar los datos en la BD. Por lo tanto, también se utilizará para almacenar los datos, tras procesarlos con los ETLs.
- ➢ Para la creación de los ETLs, se ha llevado a cabo la instalación de spark en Ubuntu, junto con la descarga de la herramienta Zeppelin, una libreta web para ejecutar el código en Scala, y poder visualizar interactivamente los datos procesados. Scala es un lenguaje de programación que permite trabajar de la manera más eficiente posible desde Spark, un motor multi-lenguaje para ejecutar los datos provenientes de la ingeniería de datos.
- ➢ Posteriormente, para la visualización de los datos mediante un cubo OLAP, se realizará con la herramienta de Pentaho schema workbench, para lo que habrá que diseñar el cubo dentro de la aplicación y almacenarlo. El fichero se utilizará luego para mostrarlo mediante un contenedor de Docker.
- ➢ Además, para la visualización de datos también se utilizará la herramienta Power BI, la cual nos permite mediante su conexión con MySQL Workbench (donde se almacena la base de datos), mostrar los datos en gráficas. Esto permite conocer exhaustivamente la información contenida en los datos y obtener conclusiones consistentes.

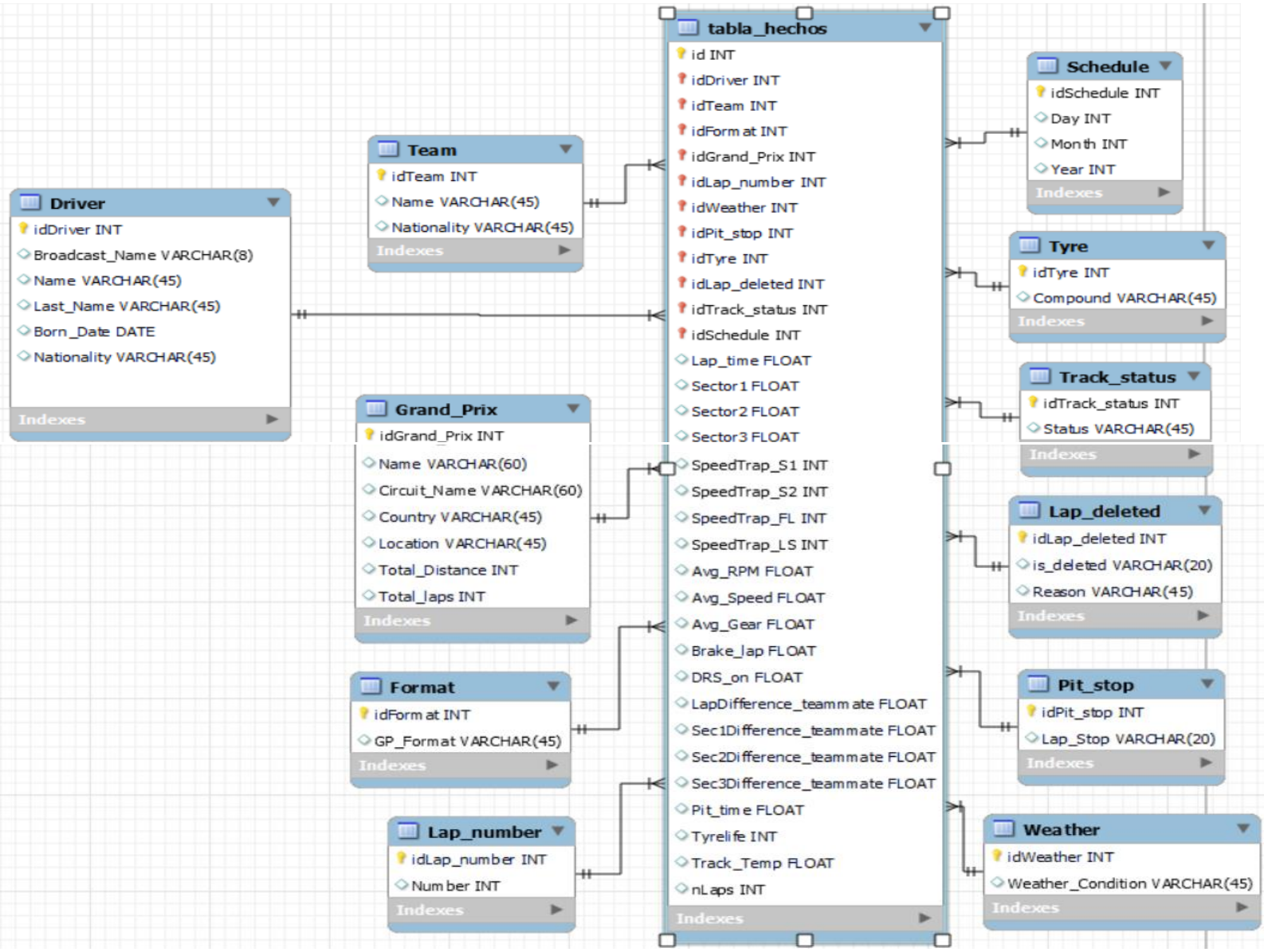
Diseños

Diseño Conceptual



Diseño Conceptual

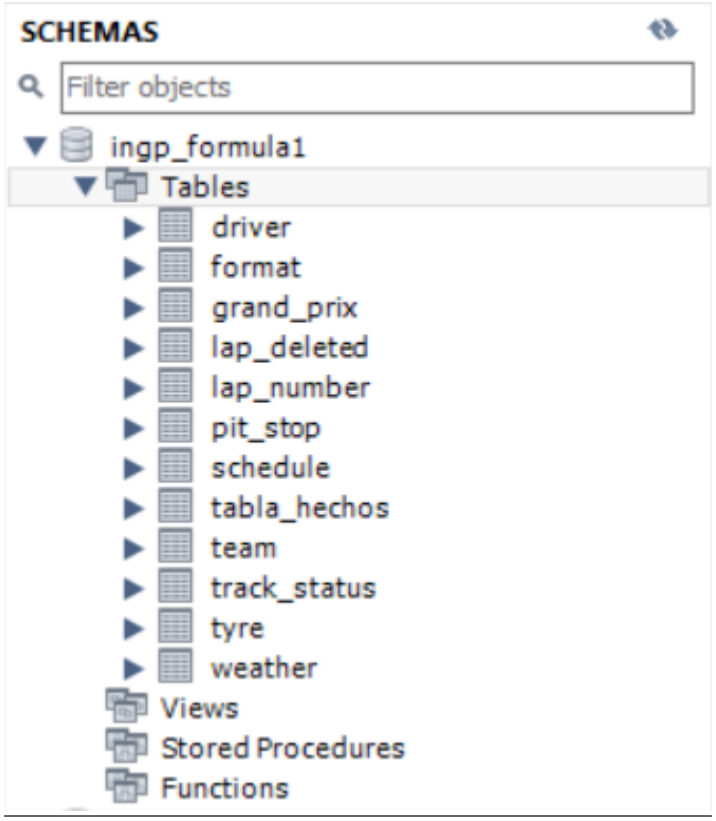
Diseño Lógico



Diseño Lógico

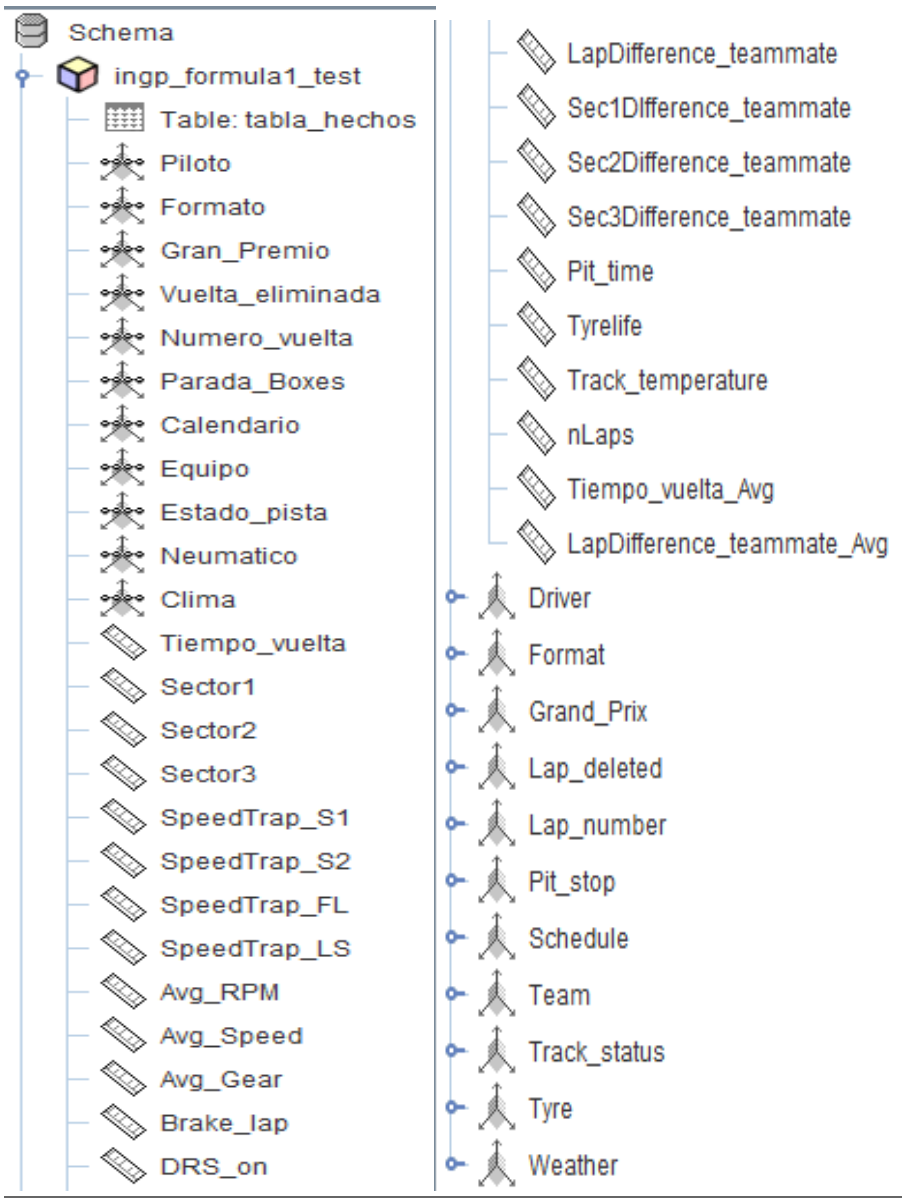
Diseño Físico

El script .sql se encuentra en la carpeta de Diseños. Una vez creado en la parte izquierda de MySQL Workbench aparece el esquema de la siguiente manera:



Diseño Físico

Diseño y creación del cubo Mondrian



Diseño Cubo OLAP

Implementación de los ETLs

La explicación de todas las transformaciones se encuentra dentro de cada apartado, donde se explica el proceso llevado a cabo paso a paso, de cara a futuros cambios o revisiones poder tener más clara la implementación de cada una.

Aclaraciones dimensiones:

- ➢ Durante la realización del ETL de la tabla “grand_prix” se ha detectado la aparición de países iguales nombrados de manera diferente. Por lo tanto, se ha realizado la tarea de agrupación de estos valores.
- ➢ Para la tabla “Track_status”, se ha dejado como id el valor devuelto, puesto que iba de 1 a 7, sin contar con el valor 3. Esto se ha dejado así para facilitar el trabajo, ya que en la API se identifica de esta manera con una definición.
- ➢ Para la dimensión “format”, se ha decidido agrupar el formato de sprint y sprint_shootout, puesto que ambos hacen referencia a un formato muy similar de carrera. Además, como formato se han añadido tanto la opción de “testing”, de cara a futuras ampliaciones, ya que este proyecto únicamente tendrá en cuenta las vueltas de fin de semanas con formato convencional o de sprint.

Aclaraciones tabla de hechos:

- ➢ Se ha decidido eliminar la información contenida en la columna “Time_OnTrack” de los ficheros .csv, puesto que al hacer un primer vistazo de los datos en el fichero .csv se ha detectado que la API siempre devuelve el valor de 100%, por lo que no detecta cuando un piloto abandona la pista.

Transformaciones

Driver

```
%spark
//Driver


val path_route = System.getProperty("user.dir")
val path = path_route + "/data/"

// Carga los datos CSV en un DataFrame de Spark
var df_drivers = spark.read.option("sep",",")
                      .option("header", "true")
                      .option("inferSchema", "true")
                      .option("encoding", "ISO-8859-1")
                      .csv(path + "drivers.csv")

// Muestra los primeros registros para verificar que se hayan leído correctamente
//df_drivers.show()

df_drivers = df_drivers.dropDuplicates("dateOfBirth","Broadcast_Name")
                       .orderBy(asc("Broadcast_Name"))


df_drivers = df_drivers.withColumn("Born_Date", to_date(col("dateOfBirth")))
                      .withColumn("idDriver",monotonically_increasing_id+1)
//z.show(df_drivers)

df_drivers = df_drivers.drop("dateOfBirth")

// Define los parámetros de conexión a MySQL Workbench
val usuarioMySQL = "user_vbox"
val contraseñaMySQL = "admin2"
val urlMySQL = s"jdbc:mysql://192.168.22.1:3306/ingp_formula1"

// Escribe los datos del DataFrame a la tabla de MySQL
df_drivers.write
    .format("jdbc")
    .option("url", urlMySQL)
    .option("dbtable", "driver")
    .option("user", usuarioMySQL)
    .option("password", contraseñaMySQL)
    .mode("append") // Opcional: puedes cambiar a "overwrite" si deseas agregar los datos a una tabla existente
    .save()

//z.show(df_drivers)


// Confirma que los datos se hayan cargado correctamente
println("Los datos de driver se han cargado exitosamente en la tabla de MySQL.")
```

ETL 1

Para el ETL de la dimensión “driver”, se realizó una carga de los datos del fichero CSV llamado “drivers.csv” en un dataframe variable. Es importante recalcar la opción de codificación, establecida como “ISO-8859-1”, ya que la codificación “UTF-8” más común, no procesaba correctamente los acentos o las diéresis del castellano.
Posteriormente, se eliminaron los duplicados de los pilotos en función de su fecha de cumpleaños. También se añadió para el caso remoto de que 2 pilotos cumplieran años el mismo día, el valor de “Broadcast_Name” para asegurarse la distinción en este caso.
Tras realizar una ordenación por la variable “Broadcast_Name”, se convirtió la fecha de cumpleaños al formato correcto en una nueva columna, para insertarse en la base de datos. Además, se generó automáticamente el id del piloto comenzando por uno y aumentando correlativamente.
Finalmente, se eliminó la columna “dateOfBirth” con el formato incorrecto de la fecha de cumpleaños y se realizó la conexión y exportación de los datos dataframe a la base de Datos exterior en MySQL Workbench en Windows. Todo esto fue realizado desde la

herramienta Zeppelin instalada en una máquina virtual con sistema operativo Ubuntu, transfiriendo los archivos csv mediante el servicio scp y utilizando ssh.

Format

```
%spark

val path_route = System.getProperty("user.dir")
val path = path_route + "/data/"

var df_format = spark.read.option("sep",",")
                          .option("header", "true")
                          .option("inferSchema", "true")
                          .option("encoding", "UTF-8")
                          .csv(path + "format.csv")

df_format = df_format.dropDuplicates("GP_Format")
                     .withColumn("idFormat",monotonically_increasing_id+1)

df_format = df_format.filter(col("GP_Format") =!= "sprint_shootout")

df_format = df_format.select("idFormat","GP_Format")

//z.show(df_format)


val usuarioMySQL = "user_vbox"
val contraseñaMySQL = "admin2"
val urlMySQL = s"jdbc:mysql://192.168.22.1:3306/ingp_formula1"

df_format.write
    .format("jdbc")
    .option("url", urlMySQL)
    .option("dbtable", "format")
    .option("user", usuarioMySQL)
    .option("password", contraseñaMySQL)
    .mode("append")
    .save()

println("Los datos de format se han cargado exitosamente en la tabla de MySQL.")
```

ETL 2

Para la dimensión “format”, el proceso seguido fue muy similar, extrayendo los datos del fichero “format.csv”. Cabe destacar, la utilización de la función “filter”, para eliminar el valor de “sprint_shootout”, ya que como se ha comentado anteriormente, se consideran sprint_shootout y sprint como el mismo formato de fin de semana.

Grand_Prix

```
%spark

val path_route = System.getProperty("user.dir")
val path = path_route + "/data/"

var df_grandPrix = spark.read.option("sep",",")
                        .option("header", "true")
                        .option("inferSchema", "true")
                        .option("encoding", "ISO-8859-1")
                        .csv(path + "grand_prix.csv")

df_grandPrix = df_grandPrix.dropDuplicates("Circuit_Name")
                        .withColumn("Total_DistanceInt",col("Total_Distance").cast("Int"))
                        .withColumn("Country", when(col("Country") === "United States", "USA").otherwise(col("Country")))

df_grandPrix = df_grandPrix.select("Name","Circuit_Name","Country","Location","Total_DistanceInt","Total_Laps")

df_grandPrix = df_grandPrix.withColumnRenamed("Total_DistanceInt","Total_Distance")
                        .withColumn("idGrand_Prix", monotonically_increasing_id+1)

//z.show(df_grandPrix)


val usuarioMySQL = "user_vbox"
val contraseñaMySQL = "admin2"
val urlMySQL = s"jdbc:mysql://192.168.22.1:3306/ingp_formula1"

df_grandPrix.write
    .format("jdbc")
    .option("url", urlMySQL)
    .option("dbtable", "grand_prix")
    .option("user", usuarioMySQL)
    .option("password", contraseñaMySQL)
    .mode("append")
    .save()

println("Los datos de Grand Prix se han cargado exitosamente en la tabla de MySQL.")
```

ETL 3

En este caso, se lee el fichero csv “grand_prix.csv”. La opción de codificación se cambia al igual que el ETL anterior al leer el fichero. En este ETL se resalta que se realiza un cast de la variable Total_Distance a entero, ya que estaba como String, con la función cast(). Además, en la columna de países. Se agrupan los valores de país para los grandes premios de Estados Unidos como USA, ya que aparecían como “USA” y “United States”. Para el caso del nombre de los circuitos, se ha localizado una variabilidad en la aparición de acentos para los nombres de los circuitos. Por ejemplo, la palabra “autódromo” aparece en ocasiones con y sin acento. Para este trabajo no se han unificado estos valores, pero sería interesante en un futuro aplicar los cambios en este campo.

Para el resto de las transformaciones, se ha decidido explicar el proceso sin la imagen, debido a que en muchos casos es muy similar y en el caso de la tabla de hechos ocuparía varias páginas mostrar todo el código. Por lo tanto, se detalla cada ETL sin apoyarse en un elemento gráfico. En el caso de querer visualizar el proceso, se adjuntará a la memoria los ficheros de cada uno de ellos.

Lap_deleted

En este caso, la extracción de datos del fichero “laps.csv” se hace igual. Luego se utiliza la función dropDuplicates(“DeletedReason”) para tener una fila por cada razón de eliminación de cada vuelta. En este ETL, también se ha tenido que realizar una función auxiliar, que elimina información de la razón. Esto se debe a que almacenar la vuelta eliminada no interesa, sino saber la curva donde se produce la razón de eliminar la vuelta.

```
// Función UDF para extraer el valor "TURN" junto con el número siguiente
val extractTurnUDF = udf((description: String) => {
  if (description != null) {
    val pattern = """TURN (\d+)""".r
    val result = pattern.findFirstMatchIn(description)
    result match {
      case Some(matched) => "TURN " + matched.group(1) // Devuelve "TURN" seguido del número encontrado
      case None => "N/A" // En caso de que no se encuentre ningún valor "TURN", se puede manejar devolviendo un valor por defecto, como "N/A"
    }
  } else {
    "N/A" // Devuelve un valor por defecto si la descripción es nula
  }
})
```

ETL 4

Lap_number

Tras leer el fichero CSV, a partir de la columna “Total_laps”,se utiliza la función agg(max(“Total_laps”).collect()(0)(0).asInstanceOf[InT]. Este código, a partir del valor máximo de la columna se consigue una instancia del entero con el mayor valor, para así generar todas las filas hasta el número obtenido. Adicionalmente, el método “monotonically_increasing_id” al estar obsoleto, me producía un error en este ETL, por lo que la generación del id autogenerado autoincremental, se ha realizado de la siguiente manera:

```
val window = Window.orderBy("Number")
df_lapNumber = df_lapNumber.withColumn("idLap_number", row_number().over(window))
```

ETL 5

Schedule

Para este ETL, se ha utilizado una función para calcular las fechas necesarias. Para ello, se pasa como parámetros los años a trabajar, que en este caso serán 2022 y 2023. A partir de estos años, se realiza el cálculo mediante la librería time de Java.

```
// DataFrame con todas las fechas de los años especificados, teniendo en cuenta años bisiestos
val year_dates = years.flatMap { year =>
  val startDate = LocalDate.of(year, 1, 1)
  val endDate = LocalDate.of(year, 12, 31)

  // Secuencia de fechas para el año actual, considerando si es bisiesto o no
  val datesForYear = (0L to java.time.temporal.ChronoUnit.DAYS.between(startDate, endDate)).map { day =>
    startDate.plusDays(day)
  }

  // Pasar la secuencia de fechas a un DataFrame
  datesForYear.map(date => (date.getDayOfMonth, date.getMonthValue, date.getYear))
}.toDF("Day", "Month", "Year")
```

ETL 6

Para el caso de los ETLs de “Pit_stop” y “Weather”, se realiza a partir de una secuencia de los valores de “Parada” y “No parada” junto con “Lluvia” y “Soleado”, respectivamente. En cuanto al resto, el proceso es muy simple, semejante a los otros casos. En el caso del ETL de Team, el proceso es muy simple y parecido al de driver. Para el caso de los ETLs de Status y Tyre, únicamente se han eliminado las filas duplicadas, a partir de la columna “Status” del fichero “status.csv” para el primero, y la columna “Tyre” del fichero “laps.csv” para el segundo. Tras este paso, la exportación del dataframe a la base de datos se produce de la misma forma que el resto.

Tabla_de_Hechos

En primer lugar, tras extraer los datos del fichero “laps.csv”, realizó la transformación de la columna “Deleted_Reason” para pasarla al mismo formato que el obtenido en el ETL de “Lap_deleted”, mediante la función auxiliar de la imagen. Luego se realizan condiciones if, para convertir la fecha al formato Date. En los datos había una errata con la fecha al realizar el look up para obtener el id del formato. Concretamente, en el caso del gran premio de Singapur, disputado a las 6:00 de la mañana (Hora UTC). En este caso, en el fichero format.csv aparecía en un día posterior al de laps.csv. Por lo tanto, para conseguir cuadrar ambas fechas y obtener el id en el posterior look up, se adelantó la fecha recuperada en un día, teniendo en cuenta que fue el único gran premio que comenzó a esta hora.

```
df_tablaHechos = df_tablaHechos.withColumn("DateSearch",
                                when(hour(col("Date")) === 6 && minute(col("Date")) === 0,
                                    to_date(date_sub(to_timestamp(col("Date"), "yyyy-MM-dd HH:mm"), 1))
                                ).otherwise(to_date(col("Date")))
                            )
                            .withColumn("WeatherSearch", when(col("Weather"), "Lluvia").otherwise("Soleado"))
                            .withColumn("Lap_deletedSearch", when(col("Lap_deleted"), "Borrada").otherwise("No borrada"))
                            .withColumn("Deleted_ReasonSearch", extractTurnUDF(col("Deleted_Reason")))
```

ETL Tabla de Hechos 1

Tras realizar estos cambios, se pasa a la parte de obtención de los ids de las diferentes dimensiones a partir de la base de datos. El código de obtención de las filas de la tabla es muy similar en todos, como se muestra en la Figura a continuación:

```
val df_driver = spark.read
  .format("jdbc")
  .option("url", urlMySQL)
  .option("dbtable", "driver")
  .option("user", usuarioMySQL)
  .option("password", contraseñaMySQL)
  .load()
```

ETL Tabla de Hechos 2

```
df_tablaHechos = df_tablaHechos.join(df_driver,df_tablaHechos("Driver") === df_driver("Broadcast_Name"), "inner")
                               .select(df_tablaHechos("*"), df_driver("idDriver").alias("idDriver"))
```

ETL Tabla de Hechos 3

Una vez realizado el join, tenemos el dataFrame con el id correspondiente de cada fila para referencia cada dimensión. Como característica a resaltar en este paso, se destaca que para el look up format, ha sido necesaria la lectura del fichero “format_csv”, para obtener el formato del fin de semana de carrera. La vinculación con el dataframe se ha producido mediante la concordancia de la fecha del fin de semana. Una vez obtenido el formato a partir de ese fichero, ya se podía realizar el join correspondiente para obtener el id del formato de cada fila. Otro look up que ha llevado un proceso más tedioso, ha sido el de la dimensión “Track_Status”. En el fichero “laps.csv”, cada vuelta podía tener varios estados a la vez, debido a que a lo largo de una vuelta puede cambiar el estado de la pista rápidamente. A causa de esto se ha realizado una clasificación:

```
// Definir una función para aplicar la lógica de prioridad
def prioridad(col1: Int, col2: Int): Int = {
  if (col2 == 1 && col1 == 12) {
    1
  } else {
    val prioridades = Array(5, 4, 6, 7, 2, 1)
    val col1Array = col1.toString.split("").map(_.toInt)
    prioridades.find(col1Array.contains).getOrElse(0)
  }
}

// Definir la función de prioridad como una UDF (User Defined Function)
val prioridadUDF = udf((col1: Int, col2: Int) => prioridad(col1, col2))

// Aplicar la UDF para generar la nueva columna con prioridad
df_tablaHechos = df_tablaHechos.withColumn("idTrack_statusOld", prioridadUDF(col("Track_status"), col("Lap_Number")))
```

ETL Tabla de Hechos 4

Como podemos ver en la Figura, para el caso de la primera vuelta de carrera, como se observó la repetición continua del valor 12, diciendo que había pista limpia, pero bandera amarilla, se asumió que normalmente, a no ser que hubiera un valor diferente, la salida era en pista limpia, por lo que se pasó a valor 1. Para el resto de caso, se aplicó la lógica de dar prioridad a las situaciones de pista más graves cuando aparecían varios valores. El orden determinado fue el siguiente: Bandera Roja, Safety Car, Virtual Safety Car Deployed, Virtual Safety Car Ending, Yellow Flag, All Clear. Mediante esta clasificación se hacía ya posible la realización del look up correspondiente a la dimensión “Track_Status”.

Al pasar esta fase, se realizó una función para pasar los tiempos por vuelta y de los diferentes sectores a la unidad de medida seleccionada, en este caso, segundos. Como en la Fórmula 1, una décima de tiempo es importantísima, se almacenarán los segundos con 3 decimales, para medir hasta los milisegundos de una vuelta.

```
// Función UDF para convertir la cadena de tiempo a segundos
val toSecondsUDF = udf((timeString: String) => {
  val timeRegex = """(\d+) days (\d+):(\d+):(\d+.\d+)""".r
  timeString match {
    case timeRegex(days, hours, minutes, seconds) =>
      val totalSeconds = days.toInt * 24 * 60 * 60 +
        hours.toInt * 60 * 60 +
        minutes.toInt * 60 +
        seconds.toDouble
      BigDecimal(totalSeconds).setScale(3, RoundingMode.HALF_UP).toDouble
    case _ => 0.0 // Caso en el que la cadena no coincide con el formato esperado
  }
})
```

ETL Tabla de Hechos 5

Cabe recalcar, que para la generación de ids autoincrementados se descartó el uso de la función de monotonically_increasing_id(), debido a que al ejecutar el script salía como un método obsoleto, y en la generación de los ids llegaba un momento que en lugar de continuar la correlación autoincremental en 1, se cambiaba a otro número completamente distintitos. Ante esto se utilizó la función row_number().over("Window.orderBy(“columnas de ordenación”).

Posteriormente, para calcular el tiempo en el Pit_Lane se realizó una función para obtener los en momento de entrada al pit en una vuelta , y de la siguiente el momento de salida. Todo esto teniendo en cuenta que el dataframe está ordenado.

Tras esto, pasaremos a limpiar el dataframe de valores nulos. En este caso, trataremos con las columnas de tiempo de vuelta, y los tiempos de los diferentes 3 sectores. Para comenzar, se utilizará una función que devolverá las filas donde uno de estos 4 valores de las columnas anteriores este a 0, pero el resto no, por lo que en base a los otros 3 se podrá calcular el valor nulo. Una vez conocidas las filas, en otro método se calculará la medida.

```
// Columna que indique si solo una de las columnas tiene un valor cero
val df_with_condition = sel_Column.withColumn("One_zero",
    (when(col("Lap_time") === 0 && col("Sector1") =!= 0 && col("Sector2") =!= 0 && col("Sector3") =!= 0, lit(1))
    .when(col("Sector1") === 0 && col("Lap_time") =!= 0 && col("Sector2") =!= 0 && col("Sector3") =!= 0, lit(1))
    .when(col("Sector2") === 0 && col("Lap_time") =!= 0 && col("Sector1") =!= 0 && col("Sector3") =!= 0, lit(1))
    .when(col("Sector3") === 0 && col("Lap_time") =!= 0 && col("Sector1") =!= 0 && col("Sector2") =!= 0, lit(1))
    .otherwise(lit(0)))
)

// Calcular el valor en la columna correspondiente y sobrescribir el valor cero
val df_with_calculated_value = df_with_condition.withColumn("Lap_time",
    when(col("One_zero") === 1 && col("Lap_time") === 0, round(col("Sector1") + col("Sector2") + col("Sector3"), 3))
    .otherwise(col("Lap_time"))
).withColumn("Sector1",
    when(col("One_zero") === 1 && col("Sector1") === 0, round(col("Lap_time") - col("Sector2") - col("Sector3"), 3))
    .otherwise(col("Sector1"))
).withColumn("Sector2",
    when(col("One_zero") === 1 && col("Sector2") === 0, round(col("Lap_time") - col("Sector1") - col("Sector3"), 3))
    .otherwise(col("Sector2"))
).withColumn("Sector3",
    when(col("One_zero") === 1 && col("Sector3") === 0, round(col("Lap_time") - col("Sector1") - col("Sector2"), 3))
    .otherwise(col("Sector3"))
)
```

ETL Tabla de Hechos 6

Una vez finalizado este paso, se prosiguió la diferencia del tiempo de vuelta y en los sectores entre los compañeros de equipo. Para realizar esta tarea, se utilizó la función que se muestra en la figura siguiente, donde tras ordenar adecuadamente el dataframe, se prosigue haciendo una comparación entre cada par de filas entre compañeros para calcular esta diferencia de tiempos.

```
def calculateDifferences(df: DataFrame): DataFrame = {
  val windowSpec = Window.orderBy("id")
  val resultDF = df.withColumn("LapDifference_teammate",
                               round(when(col("idLap_Number") === lead("idLap_Number", 1).over(windowSpec),
                                    col("Lap_time") - lead("Lap_time", 1, 0).over(windowSpec))
                               .otherwise(col("Lap_time") - lag("Lap_time", 1, 0).over(windowSpec)), 3)
                           )
                  .withColumn("Sec1Difference_teammate",
                               round(when(col("idLap_Number") === lead("idLap_Number", 1).over(windowSpec),
                                    col("Sector1") - lead("Sector1", 1, 0).over(windowSpec))
                               .otherwise(col("Sector1") - lag("Sector1", 1, 0).over(windowSpec)), 3)
                           )
                  .withColumn("Sec2Difference_teammate",
                               round(when(col("idLap_Number") === lead("idLap_Number", 1).over(windowSpec),
                                    col("Sector2") - lead("Sector2", 1, 0).over(windowSpec))
                               .otherwise(col("Sector2") - lag("Sector2", 1, 0).over(windowSpec)), 3)
                           )
                  .withColumn("Sec3Difference_teammate",
                               round(when(col("idLap_Number") === lead("idLap_Number", 1).over(windowSpec),
                                    col("Sector3") - lead("Sector3", 1, 0).over(windowSpec))
                               .otherwise(col("Sector3") - lag("Sector3", 1, 0).over(windowSpec)), 3)
                           )
  resultDF
}
```

ETL Tabla de Hechos 7

Una vez realizada este paso, se hizo necesaria la tarea de limpiar los datos de las diferentes medidas obtenidas a partir de la clase Telemetry() de la API, puesto que en muchas ocasiones eran imprecisos. Para ello, se diseñó la función calculateSpeedTrapMean(). Este método obtiene los valores de fila inmediatamente anterior y posterior, teniendo en cuenta la ordenación correcta del dataframe, Una vez teniendo estos valores calcula la media entre ellos. Con estos 3 datos almacenados en nuevas columnas del dataframe, se pasa al reemplazo de las medidas cuando tiene valores Nan. En el caso de haber obtenido la media entre el valor de la medida de la fila anterior y posterior correctamente, se sustituirá por ese valor. En caso contrario, se sustituirá por el valor de la fila anterior o posterior, siempre que sea posible. Finalmente, si no se puede realizar ninguna de estas acciones, el valor se establecerá como Mal, para eliminar la fila después.
Finalmente, se procedió con las siguientes 2 funciones, encargadas de filtrar el dataframe para que en la inserción de la información a la base de datos no se genere ningún problema. En este proceso, se eliminaron 1215 filas de las 48009 iniciales, dejando un total de 46794 filas a insertar en la base de datos de la misma manera que los ETLs anteriores.

```
def eliminarFilasMalas(df: DataFrame): DataFrame = {
    // Aplicar filtro para eliminar las filas que contienen 0 en alguna de las columnas
    val dfBueno = df.filter(
        $"Lap_time" =!= 0 && $"Sector1" =!= 0 && $"Sector2" =!= 0 && $"Sector3" =!= 0
    )

    // Contar las filas eliminadas
    val filasEliminadas = df.count() - dfBueno.count()
    println(s"Se eliminaron $filasEliminadas filas.")

    // Devolver el DataFrame con las filas válidas
    dfBueno
}

df_tablaHechos = eliminarFilasMalas(df_tablaHechos)

def eliminarFilasMalas2(df: DataFrame): DataFrame = {
    // Aplicar filtro para eliminar las filas que contienen "Mal" en alguna de las columnas
    val dfBueno = df.filter(
        $"SpeedTrap_S1" =!= "Mal" && $"SpeedTrap_S2" =!= "Mal" && $"SpeedTrap_FL" =!= "Mal" && $"SpeedTrap_LS" =!= "Mal"
    )

    val filasEliminadas = df.count() - dfBueno.count()
    println(s"Se eliminaron $filasEliminadas filas.")

    dfBueno
}
```

ETL Tabla de Hechos 8

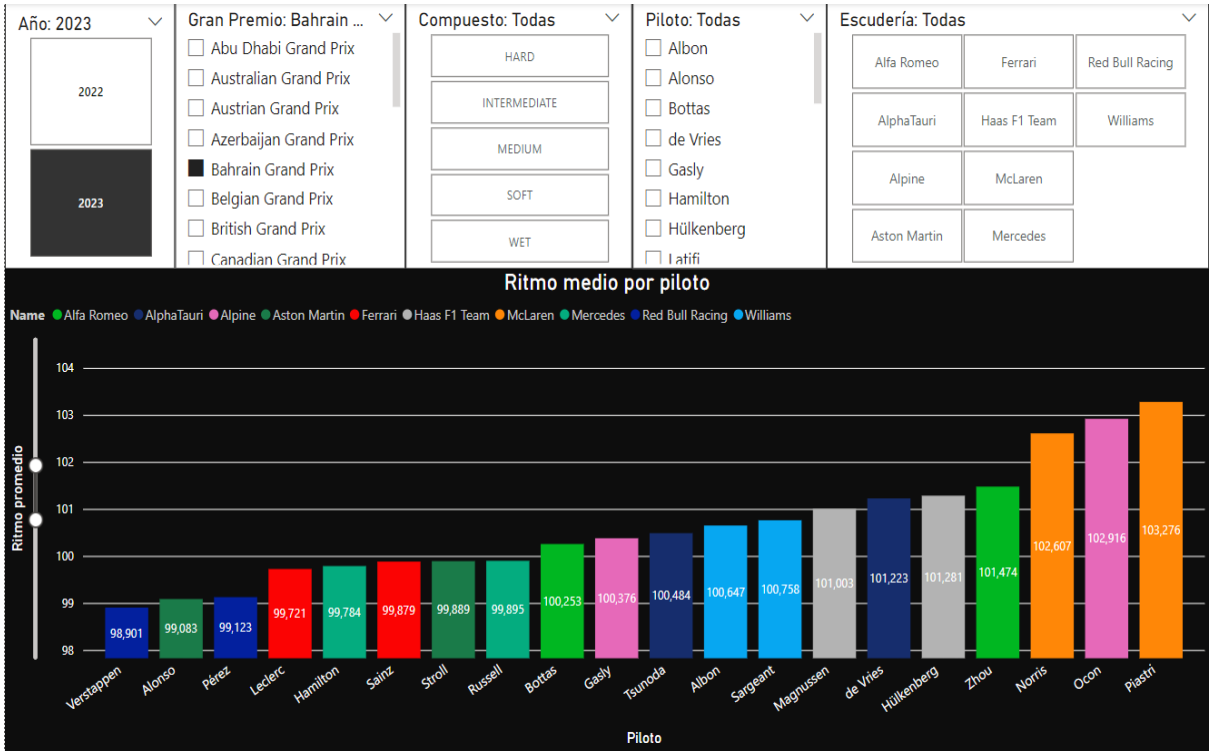
Visualización del Cubo OLAP

	Clima.Tiempo_Actual
	+ All Weather.Tiempo_Actuals
	Neumatico.Tipo
	+ All Tyre.Tipos
	Estado_pista.Estado
	+ All Track_status.Estados
	Equipo.Nacionalidad
	+ All Team.Nacionalidads
	Calendario.Fecha
	+ All Schedule.Fechas
	Parada_Boxes.Parada
	+ All Pit_stop.Paradas
	Numero_vuelta.Numero_Vuelta
	+ All Lap_number.Numero_Vueltas
	Vuelta_eliminada.Vuelta_anulada
	+ All Lap_deleted.Vuelta_anuladas
	Gran_Premio.Ubicación
	+ All Grand_Prix.Ubicacións
	Formato.Formato_GP
	+ All Format.Formato_GPs
	Piloto.Nacionalidad
Measures	+ All Driver.Nacionalidads
Tiempo_vuelta	4.373.989,051
Sector1	1.356.997,627
Sector2	1.685.760,206
Sector3	1.331.231,217
SpeedTrap_S1	251,213
SpeedTrap_S2	243,331
SpeedTrap_FL	263,267
SpeedTrap_LS	293,141
Avg_RPM	9.916,377
Avg_Speed	194,038
Avg_Gear	4,984
Brake_lap	20,802
DRS_on	156.703,7
LapDifference_teammate	11.477,619
Sec1DIfference_teammate	2.652,62
Sec2Difference_teammate	1.017,586
Sec3Difference_teammate	2.440,089
Pit_time	275.937,569
Tyrelife	67
Track_temperature	35,201
nLaps	46.794

Cubo OLAP

En esta visualización se muestra en forma de columna todas las dimensiones, por un lado, y luego por otro lado, se muestran todas las medidas de la tabla de hechos en forma de fila. Gracias a la medida nLaps, con valor a 1 en todas las filas, podemos comprobar como las 46794 filas almacenadas en la base de datos se han conectado de manera correcta y se ha diseñado el cubo OLAP de manera correcta.

Dashboard



Visualizaciones 1

En esta visualización, se detalla el ritmo promedio de cada piloto. Además, esta visualización se puede filtrar en función de un gran premio de un año concreto, como en el caso de la imagen, que muestra la carrera del gran premio de Bahrein en el año 2023. En caso de ver información con más detalle, se puede filtrar por equipo para ver la diferencia de ritmo entre compañeros y por piloto, para comparar entre varios pilotos. Estos filtros están sincronizaos con otra página, donde se visualiza el ritmo promedio de cada sector para cada piloto, para ver la diferencia entre los sectores. Por ejemplo, en cuanto a ritmo de carrera, el piloto Rusell se encuentra en octava posición. Sin embargo, viendo los sectores la mayor diferencia la pierde en el segundo sector, puesto que en los sectores 1 y 3, está en sexta y quinta posición, respectivamente en cuanto a ritmo se refiere. Por lo que, podemos decir que gran parte de su ritmo se pierde en el segundo sector.

Visualizaciones 2

En el caso de querer compara pilotos de varios equipos, en el ritmo seguido a lo largo de la carrera, la siguiente gráfica lo muestre de mejor manera.

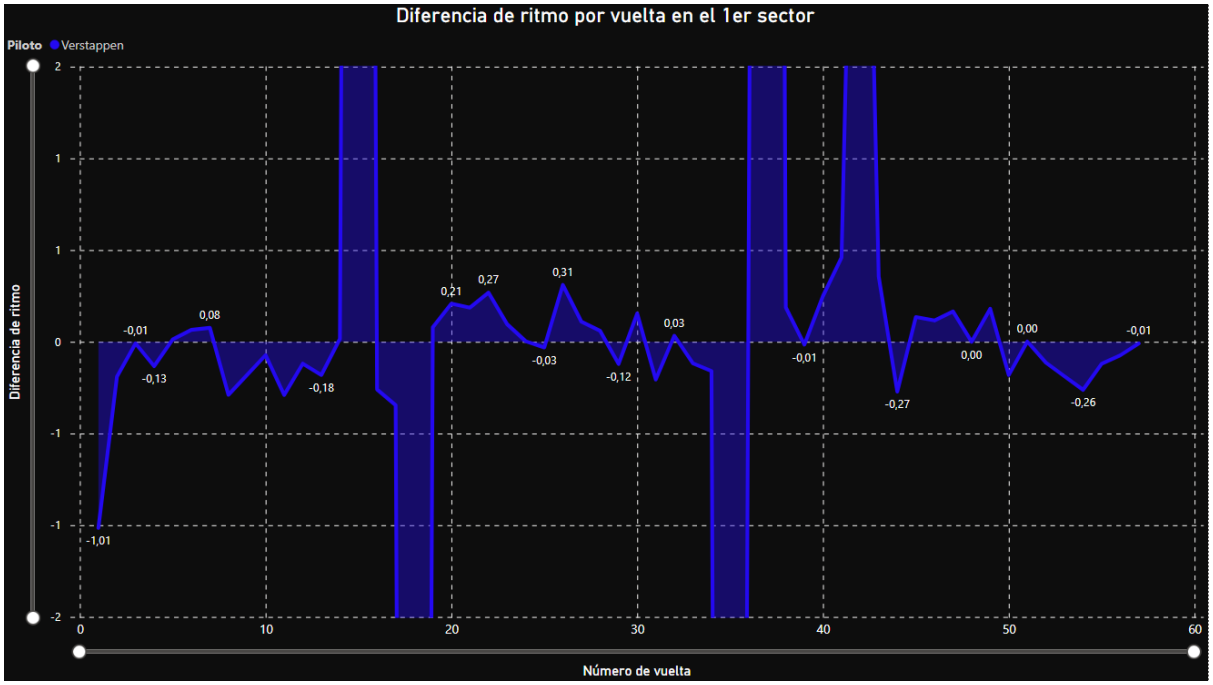
Visualizaciones 3

Descartando los valores extremadamente altos, que representan las paradas en boxes de cada piloto, podemos ver en esta comparación como en el primer stint, Verstappen tuvo un ritmo mucho más superior a Alonso o Hamilton, quienes tuvieron un ritmo muy semejante. No obstante, en el segundo stint, podemos ver

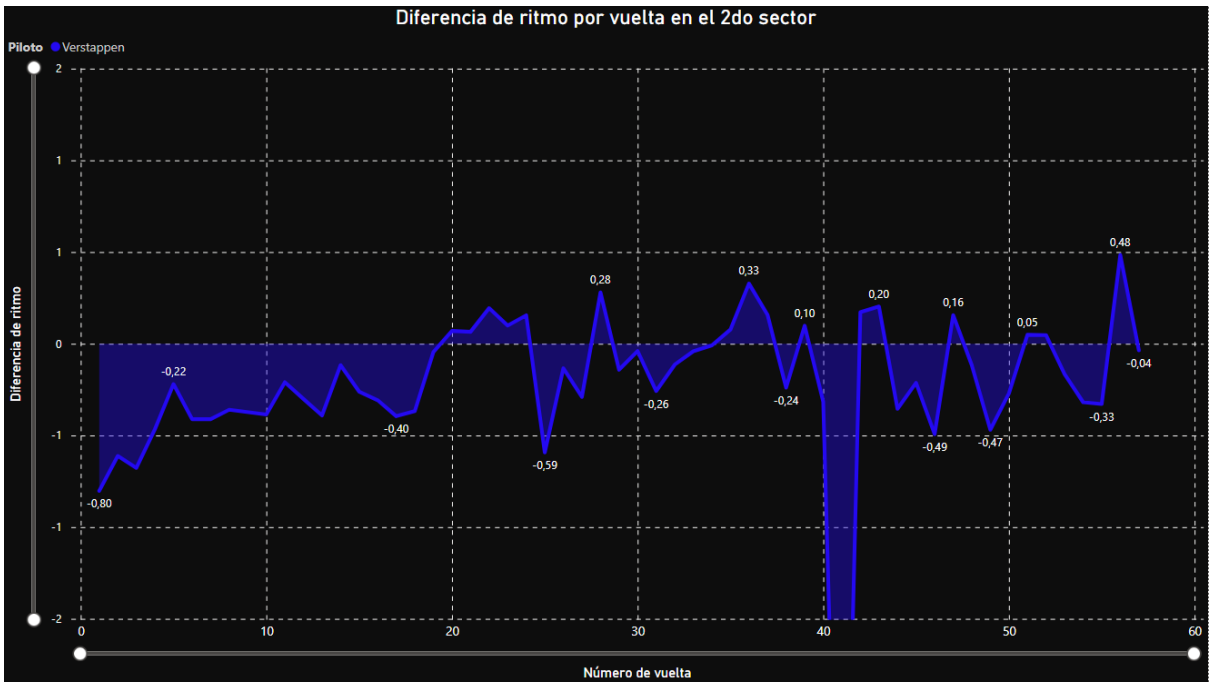
como Alonso es ligeramente superior a Hamilton y se acerca al ritmo de Verstappen. Al final de la carrera las diferencias entre piloto se mantienen, teniendo en cuenta que Alonso rueda en ritmos parecidos a Verstappen y Hamilton mantiene un ritmo unas décimas más lento que los otros dos.

Visualizaciones 4

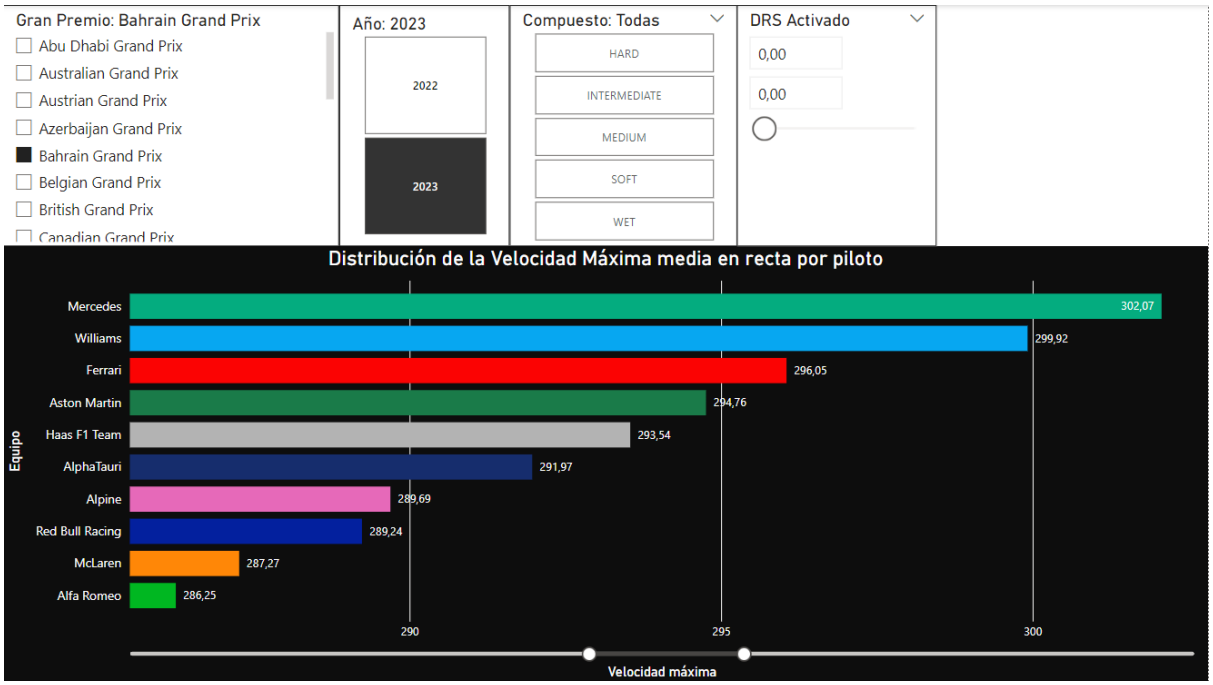
En esta gráfica, podemos ver la diferencia de tiempo a lo largo de un gran premio entre compañeros de equipo. Concretamente, con el filtro marcado en el gran premio de Bahrain del año 2023, el piloto Max Verstappen mantuvo un ritmo superior a su compañero de equipo Checo Perez. El primer stint sobre todo, llego a sacarle un segundo por vuelta, algo que en fórmula 1 es una barbaridad. En los siguiente stints el ritmo fue más similar entre ambos. Además, al igual que la visualización anterior, se ha creado una página para cada sector, para así entender mejor esta gráfica. Todas ellas sincronizadas con el dashboard principal. En las imágenes a continuación, podemos ver como en el primer ambos pilotos están muy igualados, pero en el segundo sector es donde Verstappen predomina contundentemente a su compañero.



Visualizaciones 5

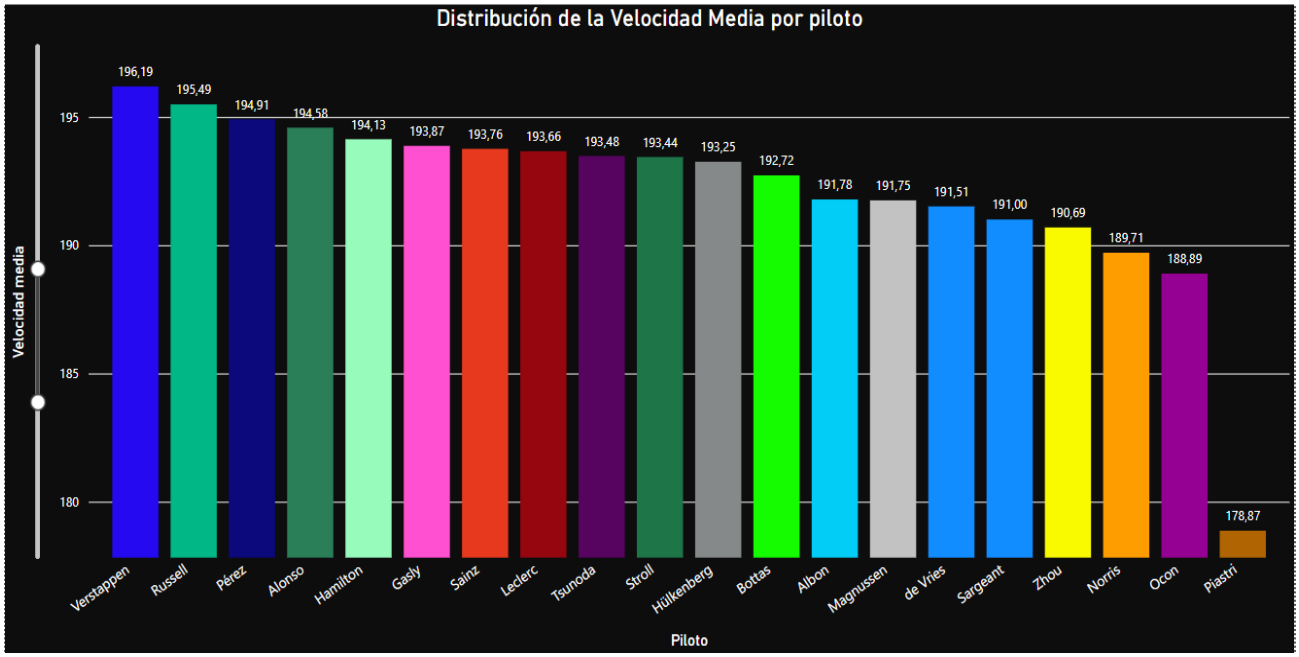


Visualizaciones 6

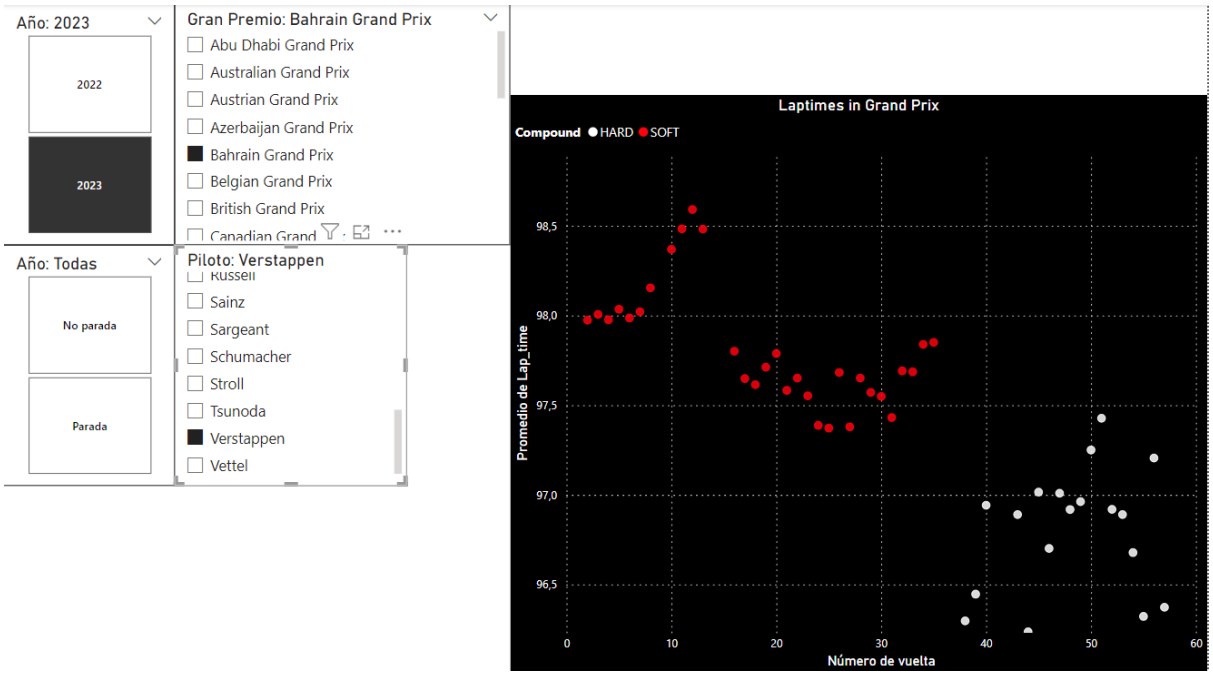


Visualizaciones 7

En este gráfico, vemos como en este gran premio, los equipos más veloces en la recta más larga fueron Mercedes y Williams, no obstante, no obtuvieron los mejores resultados, ya que no todo en la fórmula 1 es la velocidad punta, ya que el objetivo es ser rápida en frenada y en curva también. Si vemos en la gráfica a continuación, creada en otra página y sincronizada con los filtros de la anterior, veremos que la velocidad media nos dice una cosa llamativa, ya que Verstappen y Perez son 2 de los 3 pilotos con la velocidad media más alta del gran premio, pero en la velocidad máxima media en recta se encontraban en el octavo equipo.



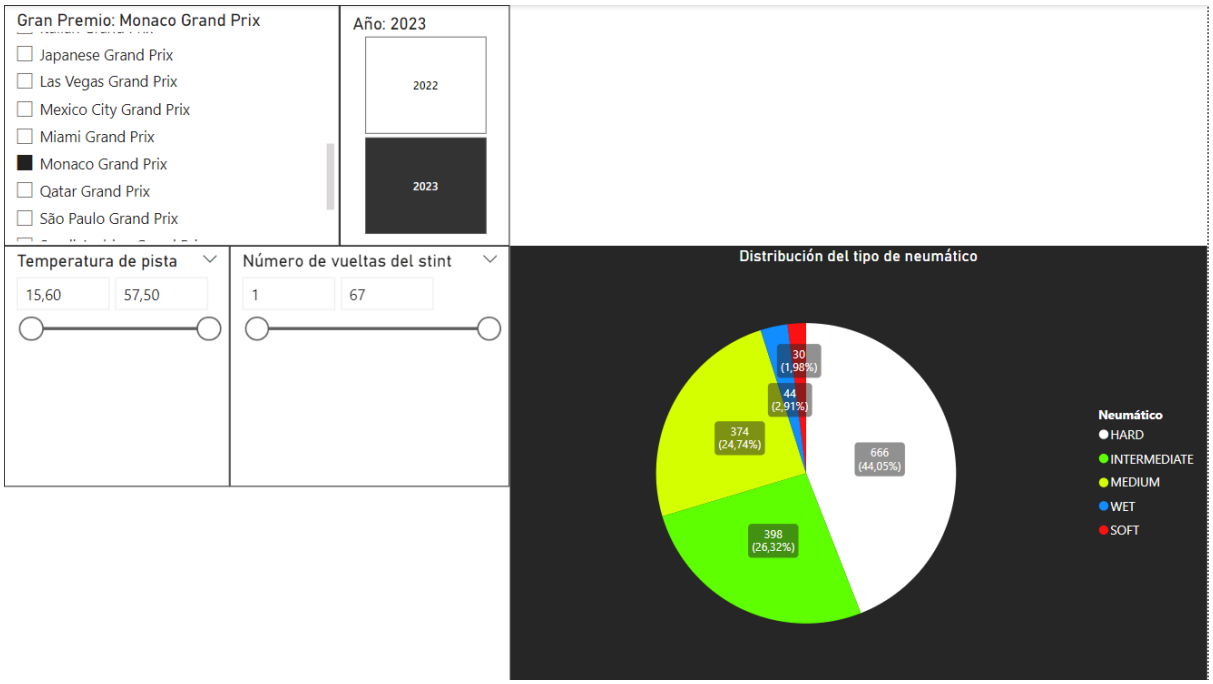
Visualizaciones 8



Visualizaciones 9

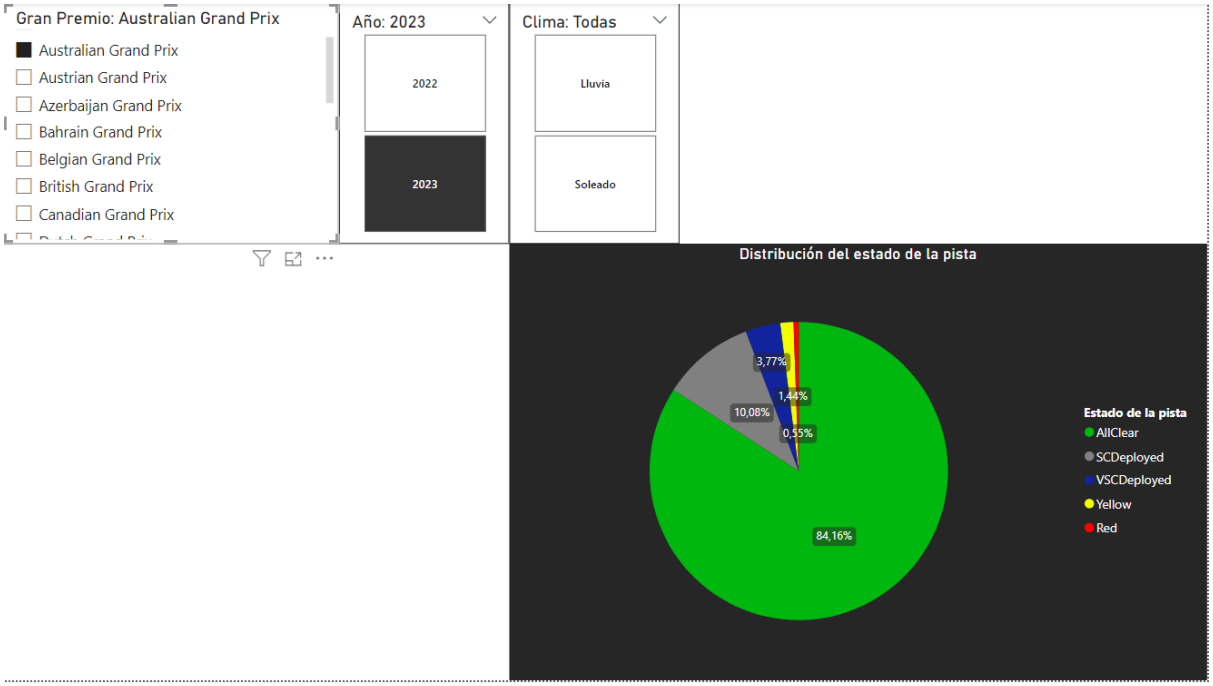
En este caso, observamos el descenso de los tiempos a lo largo del gran premio, donde Verstappen hizo 2 stints con el neumático blando y 1 con duros. En el primer Stint mantuvo un ritmo constante hasta que la degradación hizo retrasar un poco sus tiempos. Posteriormente, en el segundo stint sí que fue disminuyendo los tiempos poco a poco hasta las vueltas finales, donde volvió a aparecer la degradación. Finalmente, con el neumático duro, Verstappen

mantuvo un ritmo constante bastante bueno ya que, aunque algunas vueltas empeoraba el tiempo, lo fue bajando poco a poco sin afectar a la degradación.



Visualizaciones 10

Esta visualización nos ayuda a entender cuál fue el tipo de neumático predominante en cada gran premio. En el caso seleccionado (Mónaco de 2023), encontramos que se utilizaron todos los tipos de neumáticos posibles, ya que hubo lluvia. Esto no suele pasar en fórmula 1, pero cuando se produce una lluvia momentánea, hace que cada equipo tenga que reaccionar rápidamente pensado en la mejor estrategia. Concretamente, el 44% de las vueltas del gran premio fueron con Duro, pero como empezó a llover poco después de pasar el ecuador de la carrera, el neumático intermedio, enfocado para casos de lluvia ligera, fue utilizado en el 26% de las vueltas, siendo el segundo compuesto más predominante.



Visualizaciones 11

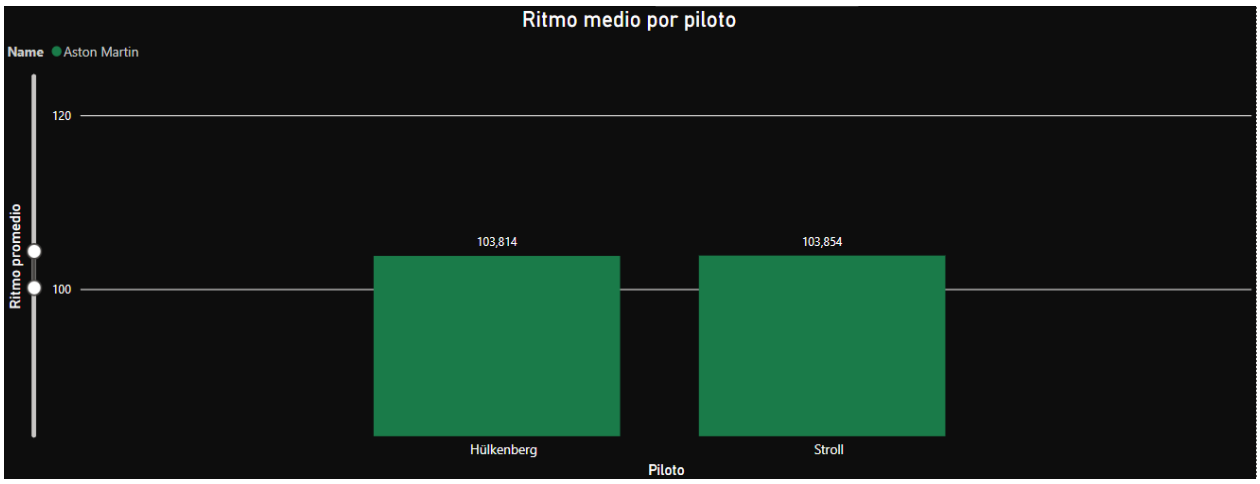
En esta última gráfica, observamos el estado de la pista a lo largo de la carrera. Por ejemplo en el gran premio de Australia del año pasado pasaron bastantes cosas ya que aunque el 84% de la carrera se corrió con la pista limpia, hubo Safety Car, Bandera Roja y Virtual Safety Car. Esto ayuda a entender como gente con tiempos buenos puede haber acabado en una mala posición debido a estos incidentes, al igual que a predecir la posibilidad de banderas rojas o safety car en años posteriores, y prepararse en función de ello.

Conclusiones Proyecto

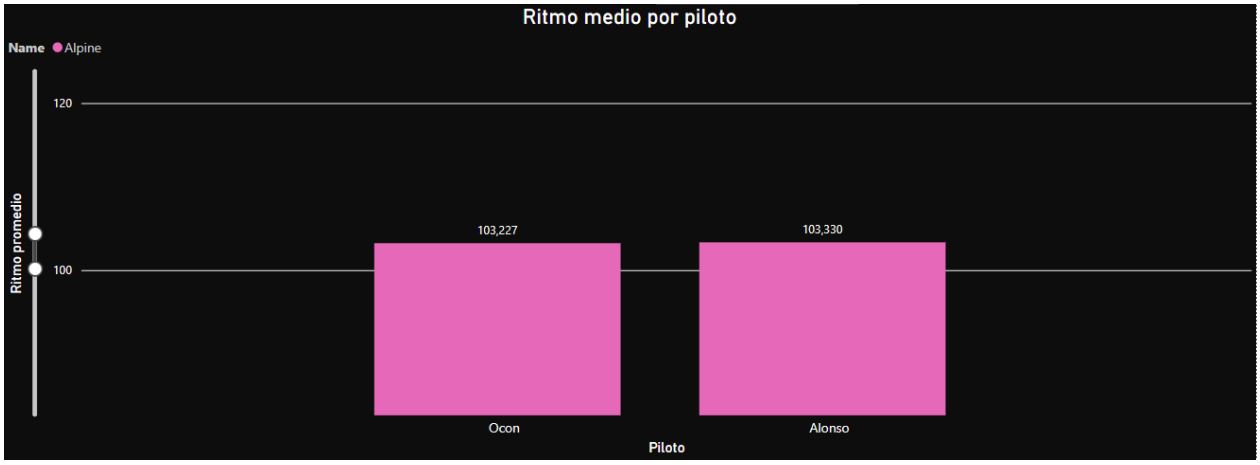
Como conclusión, me gustaría resaltar el aprovechamiento obtenido en el desarrollo de este proyecto, puesto que he aprendido todo el proceso necesario desde el planteamiento de los objetivos y KPIs, hasta la visualización de los resultados de estos, pasando por extracción del dato a partir de diferentes fuentes, junto con los diferentes diseños y tareas ETL necesarias para introducirlos en las bases de datos y mostrar los datos gráficamente. En este proyecto en particular se han conseguido obtener conclusiones bastante satisfactorias a los KPIs planteados gracias a las visualizaciones, donde se muestra la evolución de un año para otro del rendimiento de las diferentes escuderías, además de mostrar las diferencias entre compañeros a lo largo de esos 2 años. Por lo que, si nos fijamos en los resultados, el equipo Aston Martin por ejemplo sí que ha conseguido la mejora de rendimiento esperada con los

KPIs, ya que, si filtramos por equipo en el gran premio de Bahrain en la primera gráfica de las visualizaciones, vemos como ambos pilotos tuvieron un ritmo promedio 103,8 segundos por vuelta. Sin embargo, al año siguiente, el ritmo promedio es de 99,083 y 99,889, lo que hace una media de 99,5, lo que hace una rebaja de tiempo del 4,5%, algo que en fórmula 1 es una barbaridad. En cambio, el equipo Alpine solo ha conseguido una rebaja del 2%. Estos datos se pueden estudiar más en detalle viendo las visualizaciones por sectores, donde en el caso de Aston Martin han reducido 1,3 segundos como mínimo en cada sector de una temporada a otra, destacando el primero donde han reducido su tiempo hasta 1,5 segundos su tiempo promedio en este sector. En cambio, en el caso de Alpine, se ha conseguido una reducción de tiempos de entre 6 y 8 décimas en cada sector, por lo que, aunque también han progresado, no han cumplido con el KPI fijado. A continuación, se muestran las visualizaciones filtradas que han llevado a estas conclusiones:

2022:



Conclusión 1

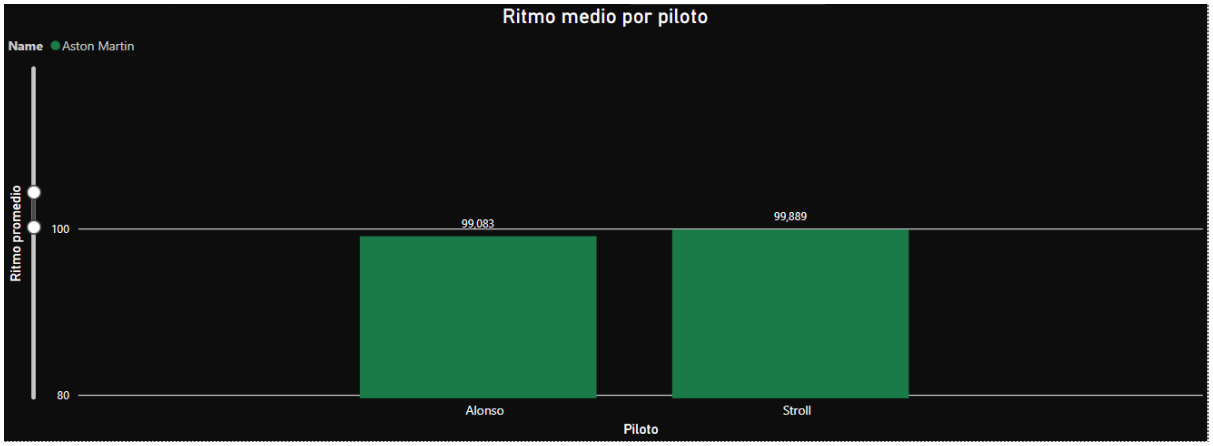


Conclusión 2

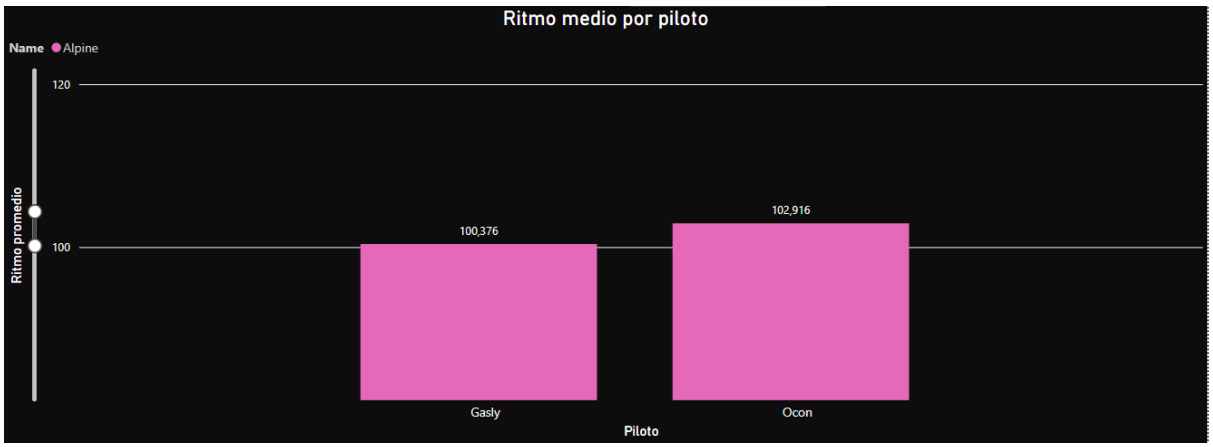
Conclusión 3

Conclusión 4

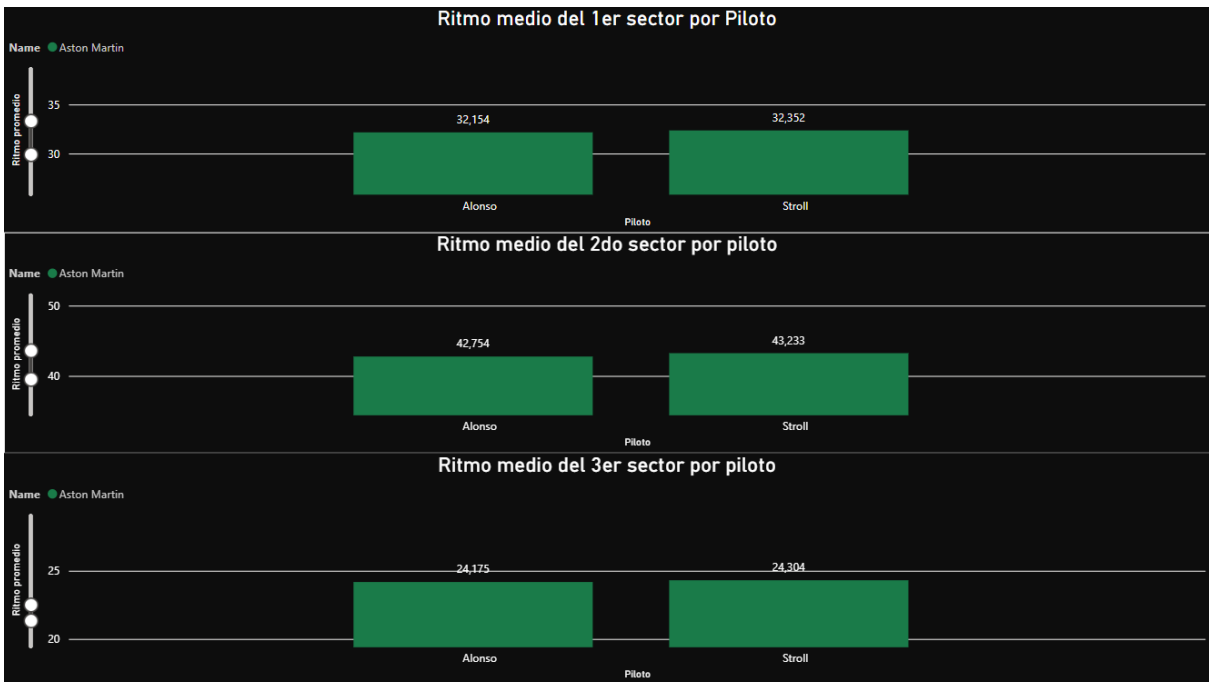
2023:



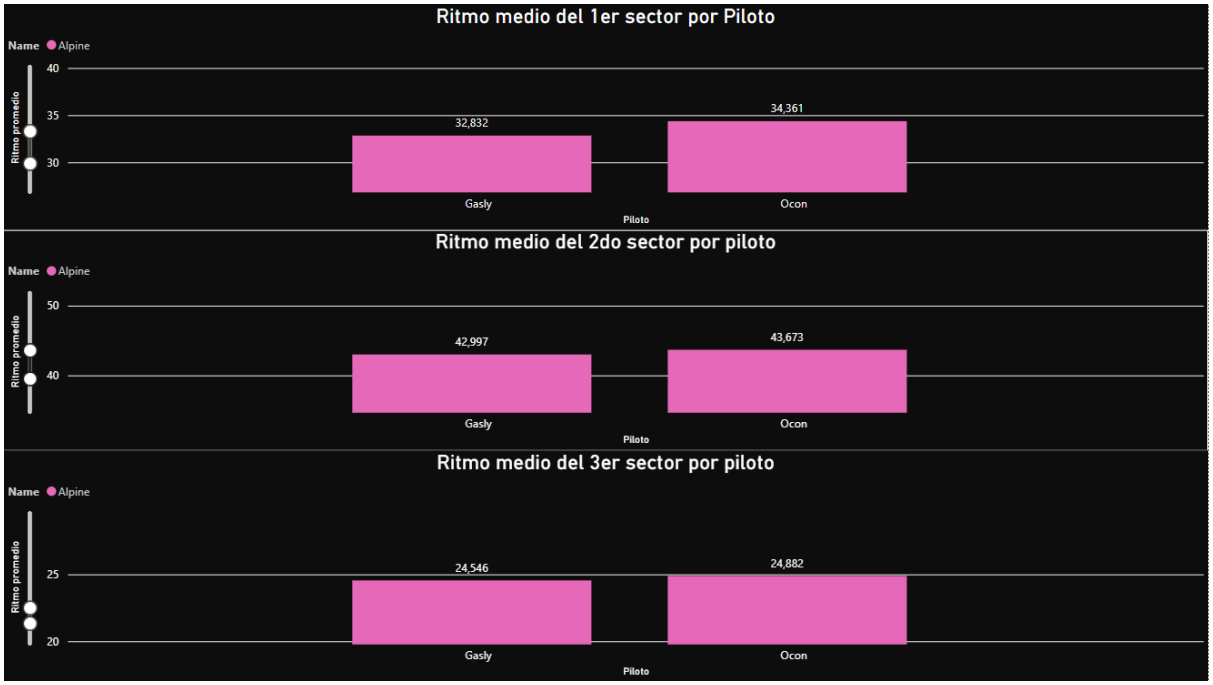
Conclusión 5



Conclusión 6



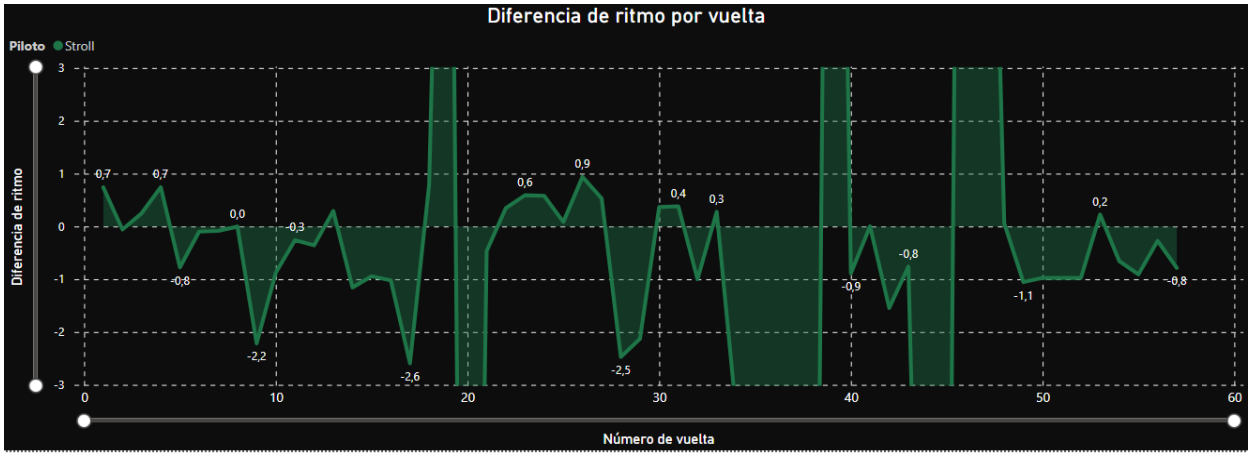
Conclusión 7



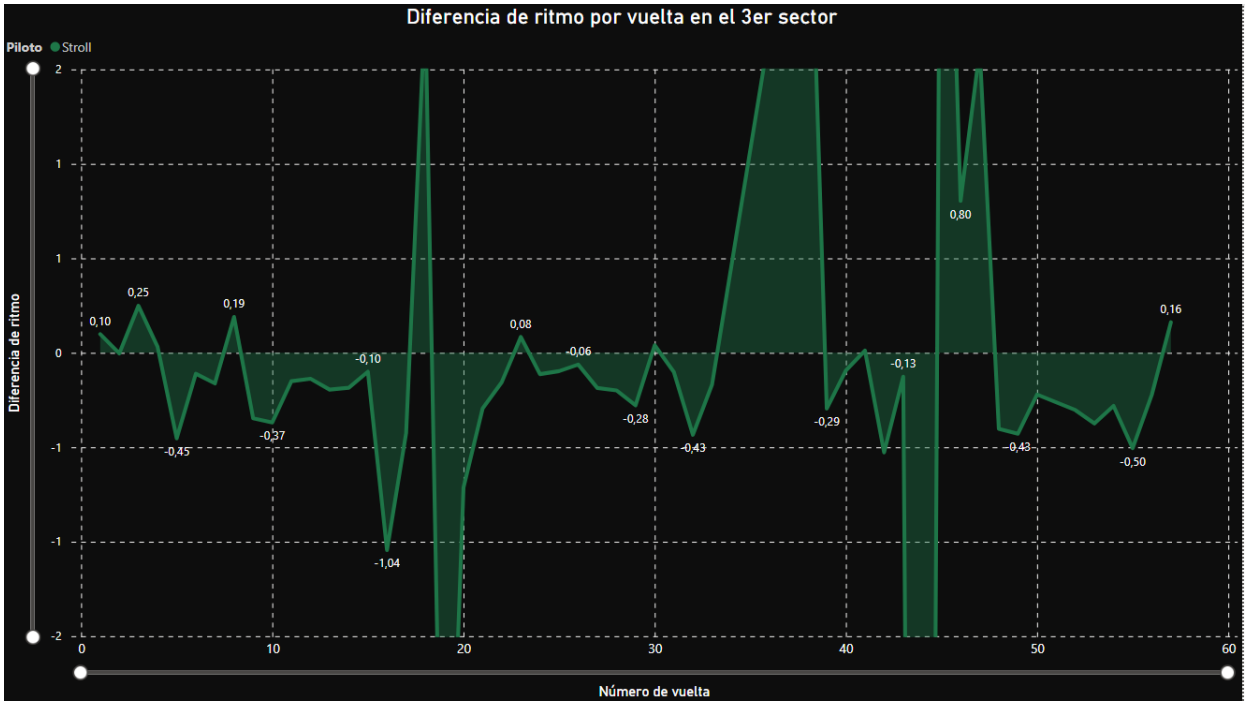
Conclusión 8

En cuanto al segundo KPI, que tenía por objetivo mantener la competitividad de ambos pilotos reduciendo la diferencia entre ellos a decimas por vuelta como máximo, nos centraremos en el caso de Aston Martin, comparando el gran premio de Bahrain de 2022 y 2023 desde el punto de vista del jefe de equipo de la escudería. En el caso de este equipo, hubo un cambio de pilotos de 2022 a 2023, ya que en 2022 corrió Hulkenberg junto a Stroll, mientras que en 2023 corrieron Alonso y Stroll. Al repetir Stroll, podemos hacer la comparación poniendo a este piloto en el foco de las gráficas. En el caso de 2022, Stroll fue superior a su compañero de equipo gran parte del gran premio, y si entramos un poco más en detalle vemos como gran parte de su diferencia de tiempo la obtiene siendo superior en el tercer sector. En cambio, la historia cambia en 2023, donde Stroll es inferior a su compañero de equipo, donde a lo largo del gran premio gran parte de sus tiempos por vuelta son inferior a su compañero de equipo, aunque desde el principio a mitad de carrera la diferencia se mantiene en un rango bastante cercano al KPI establecido. Si analizamos en mayor profundidad por sectores, en el primer sector se mantiene una igualdad bastante notable en todo el gran premio, por lo que la diferencia aparece en el tercer y segundo sector. En el segundo sector, sobre todo, es donde aparece la mayor diferencia de tiempo, donde Stroll sufre en la comparativa con su compañero de equipo. Las siguientes gráficas muestran los datos que han permitido llegar a esta conclusión:

2022:

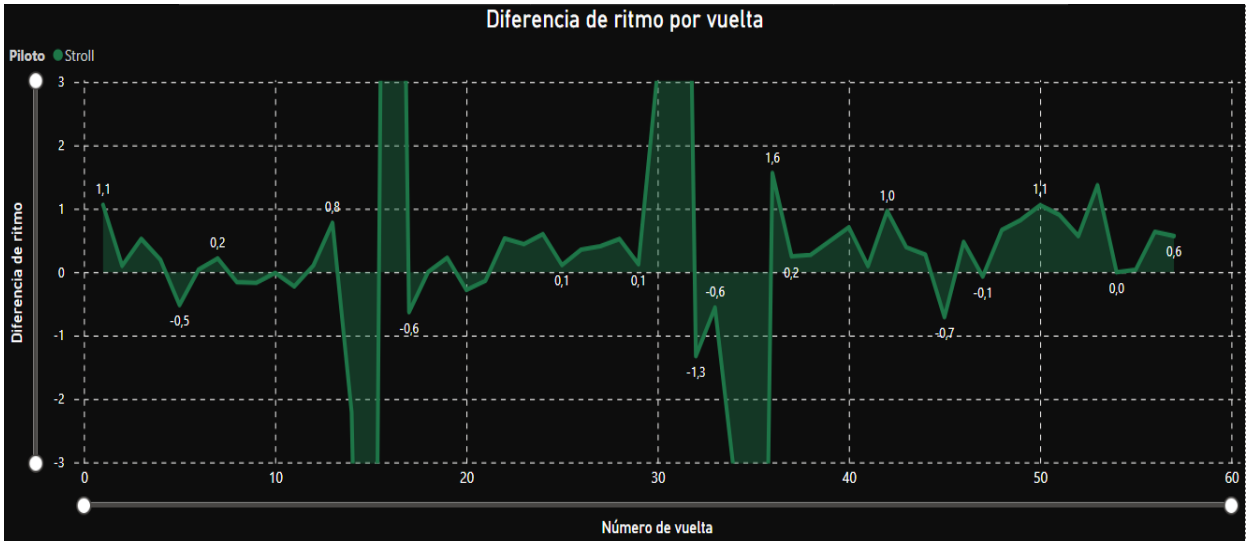


Conclusión 9

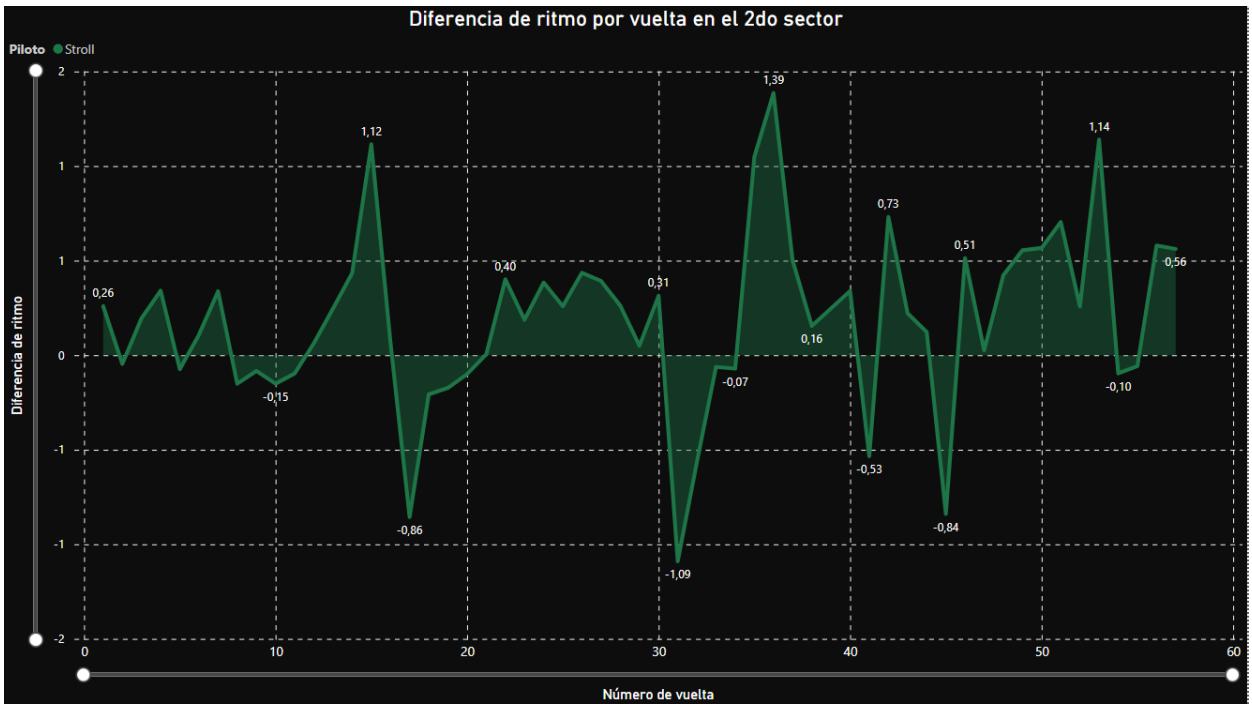


Conclusión 10

2023:



Conclusión 11



Conclusión 12

Debido al tamaño de trabajo no se han puesto los ejemplos para todos los KPIs, pero con las visualizaciones realizadas se pueden resolver en detalle las conclusiones referentes a los KPIs de los diferentes equipos.

Aspectos de mejora

Como mejora, se podría llevar a cabo una ampliación de las características de cada piloto como de la escudería, como ver nombres antiguos de la escudería o el palmarés de cada piloto. Adicionalmente, también podría ser interesante

añadir más medidas, a pesar de contar ya con una gran cantidad, para comprender mejor los resultados producidos a lo largo de un gran premio.

Limitaciones del proyecto

Una limitación importante en este trabajo se ha producido en la recolección de los datos, ya que hay características que no se pueden recoger, como, por ejemplo, una clasificación del circuito en función de los circuitos con mayor número de curvas rápidas o de curvas medias o lentas, para así conocer mejor los puntos fuertes del coche de cada escudería.

Ampliaciones futuras

A lo largo de este proyecto, se han comentado diferentes aspectos que se podrían llevar a cabo como ampliación. Adicionalmente, sería interesante almacenar la información de la clasificación y de los entrenamientos libres resultados a lo largo del fin de semana, de cara a aumentar el espectro de datos del fin de semana y conseguir unos datos más concretos.

En una ampliación futura se podría ampliar este proyecto a la extracción de datos en streaming, ya que la API cuenta con la llamada a los métodos en live_data, de cara a predecir a partir del ritmo de los libres y de la clasificación, el resultado en carrera.